ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

APLICAÇÕES DE INTELIGÊNCIA ARTIFICIAL
APPLICATIONS OF ARTIFICIAL INTELLIGENCE

LECTURE 6: Deep Learning - CNN ARCHITECTURES FOR OBJECT DETECTION & LOCALISATION

Petia Georgieva
(petia@ua.pt)

universidade de aveiro

Outline

Part 1: Object detection

- **Sliding windows detection algorithm**
- **YOLO (You Only Look Once) algorithm & achitecture**
- **Other CNNs: Region proposal R-CNN, ResNet, Inception (GoogLeNet)**

universidade de aveiro

CV: classification/localization/detection

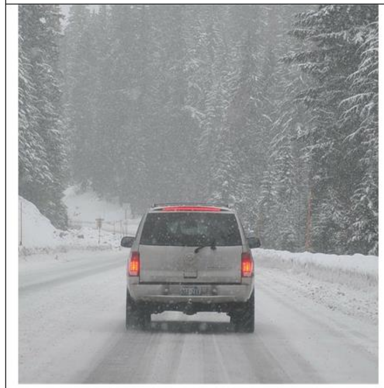
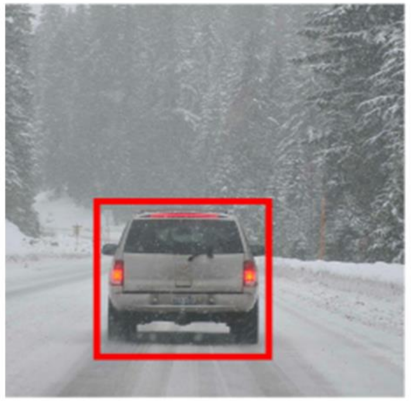
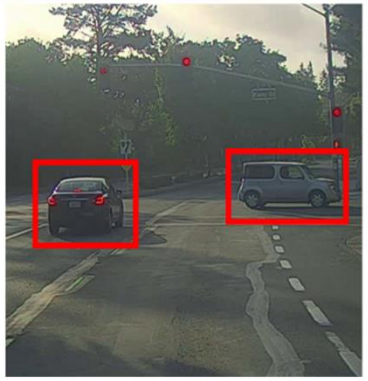
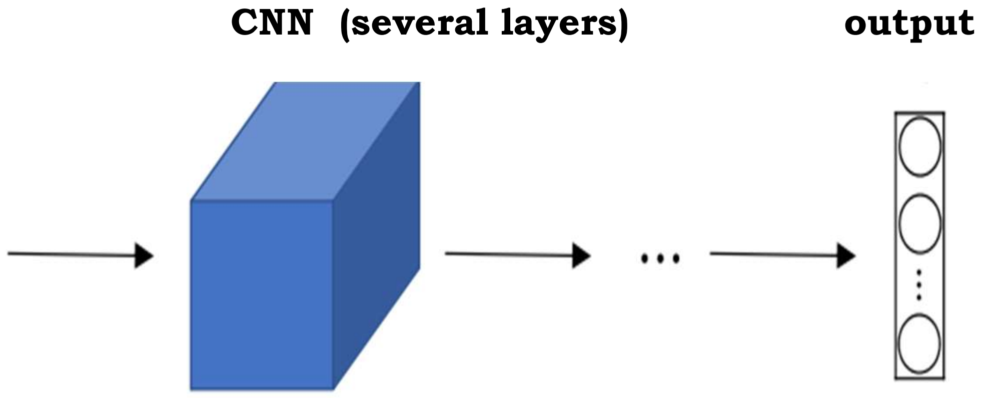
Image classification	Classification & Localization	Detection
	 b_x, b_y, b_h, b_w	

Image classification: input a picture into the model and get the class label (e.g. person, bike, car, background, etc.)

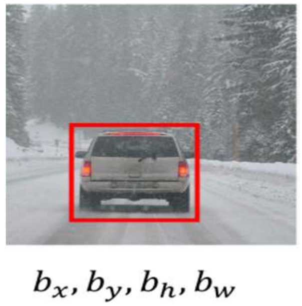
Classification & localization: the model outputs not only the class label of the object but also draws a bounding box (the coordinates) of its position in the image.

Detection: the model detects and outputs the position of several objects.

Object classification with localization



Classes (e.g.):
1. person
2. Bikes
3. Car
4. Background (no object)



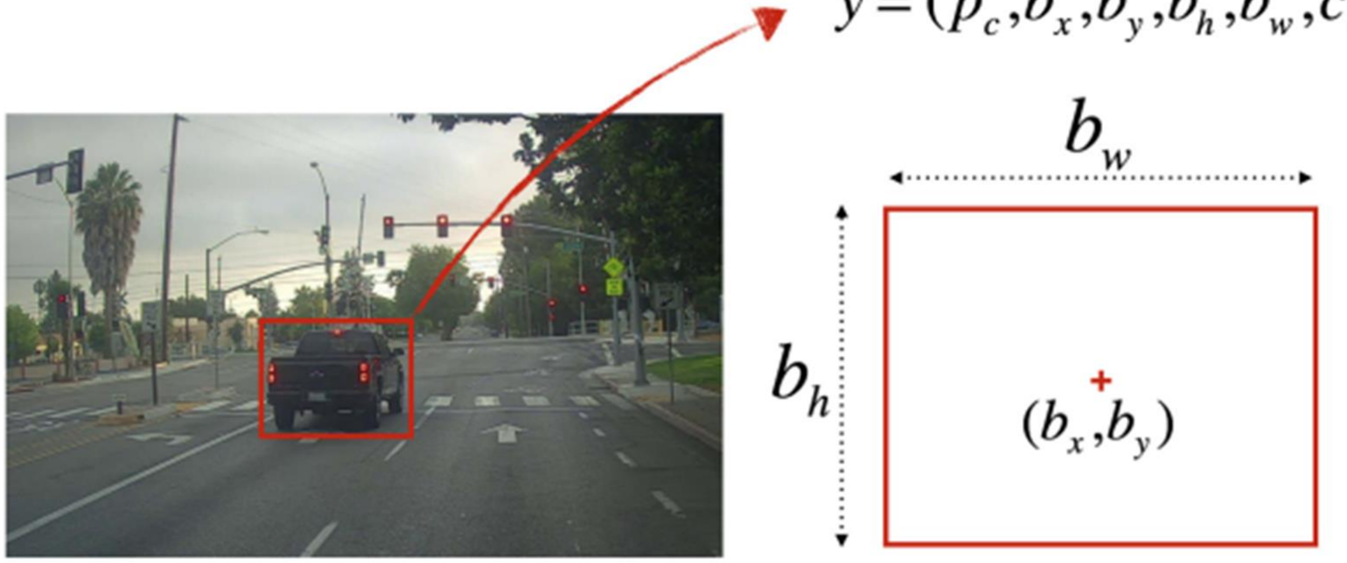
Output: (p_c , b_x , b_y , b_h , b_w , c= [c_1, c_2....c_{end}])
p_c – is there an object or not (1/0)

<= Image label: [1 , b_x , b_y , b_h , b_w , 0, 0, 1]

<= Image label: [0 , ? , ? , ? , ? . ?, ?; ?]
? – "don't care"

Output/label vector

$$y = (p_c, b_x, b_y, b_h, b_w, c)$$



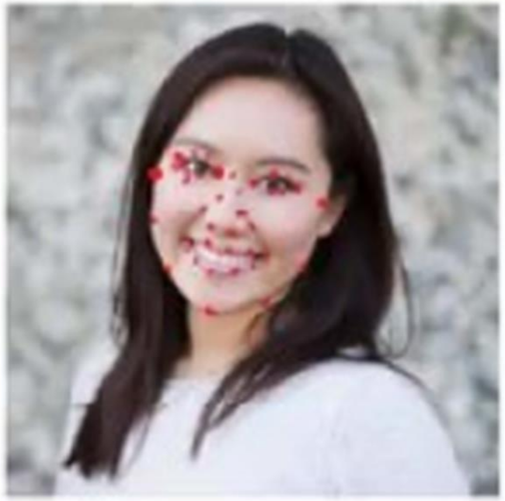
$p_c = 1$: confidence of an object being present in the bounding box

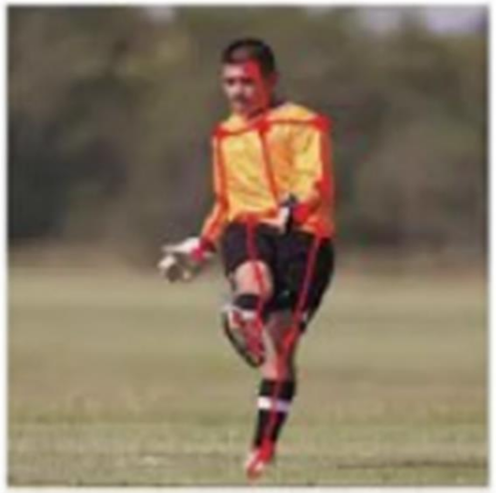
p_c b_x b_y b_h b_w $\longleftarrow$ 80 class probabilities $\longrightarrow$

Landmark Detection

The idea of bounding boxes outputs/labels (b_x, b_y, b_h, b_w) inspired the use of DL for e.g. emotion recognition from faces, person's pose detection.
CNN can output the important coordinates (called landmarks) of targeted objects in the image. For example 64 chosen points on the face, or the body (key positions in the persons pose, e.g. the mid point of the chess, left/right shoulder, etc.).
Need for manually label all landmarks in the training data (supervised learning) !!!
Consistent annotation over several images.

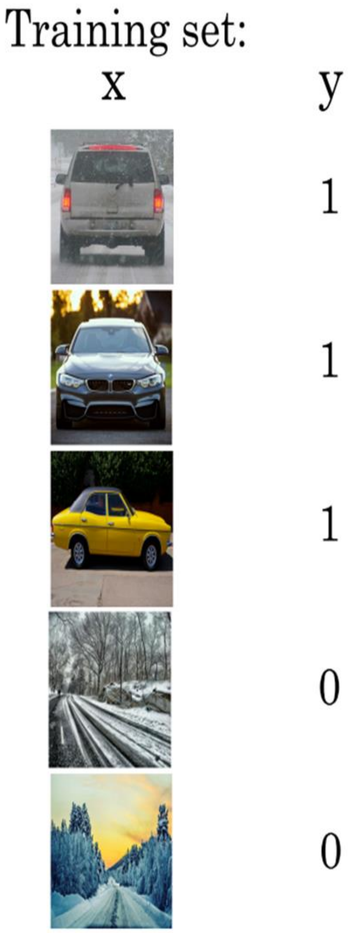
b_x, b_y, b_h, b_w

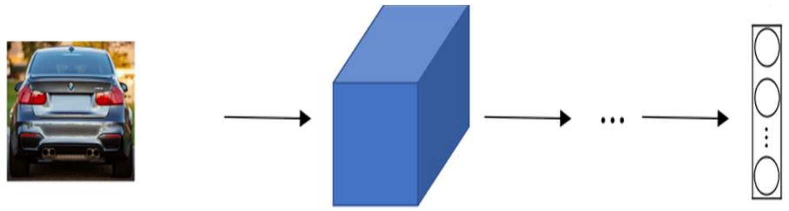




$$\text{Image} \longrightarrow \text{ConvNet} \longrightarrow \underbrace{\begin{bmatrix}\circ\\ \circ\\ \circ\end{bmatrix} \text{Output: face}, l_{1x}, l_{1y}, \ldots, l_{64x}, l_{64y}}_{\text{total of 129 outputs}}$$

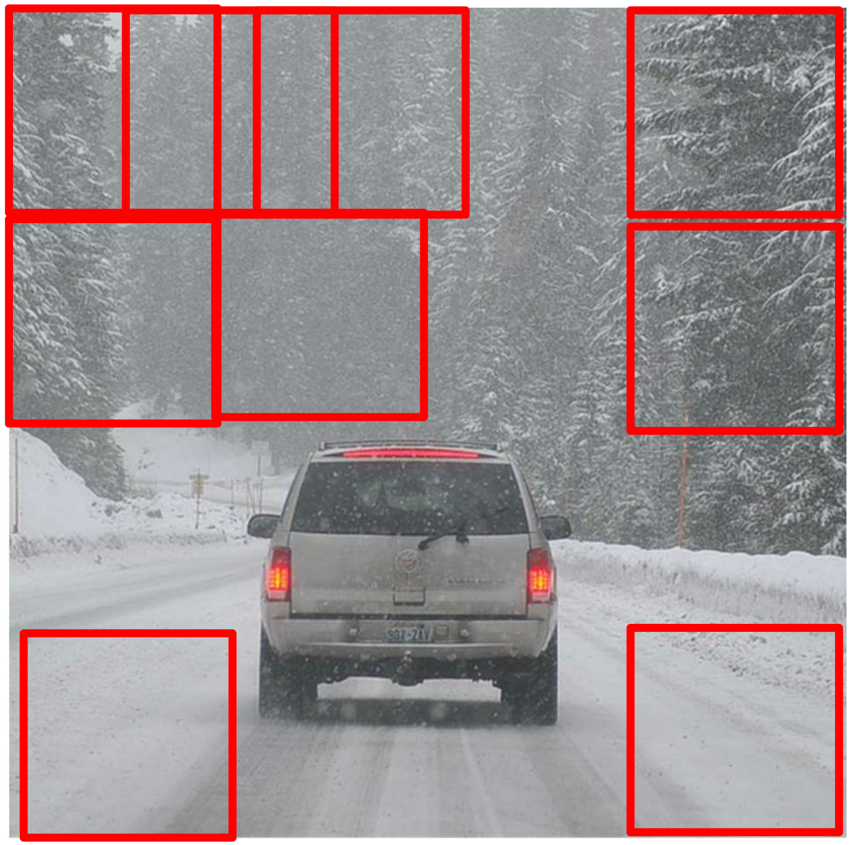
Object Detection algorithm

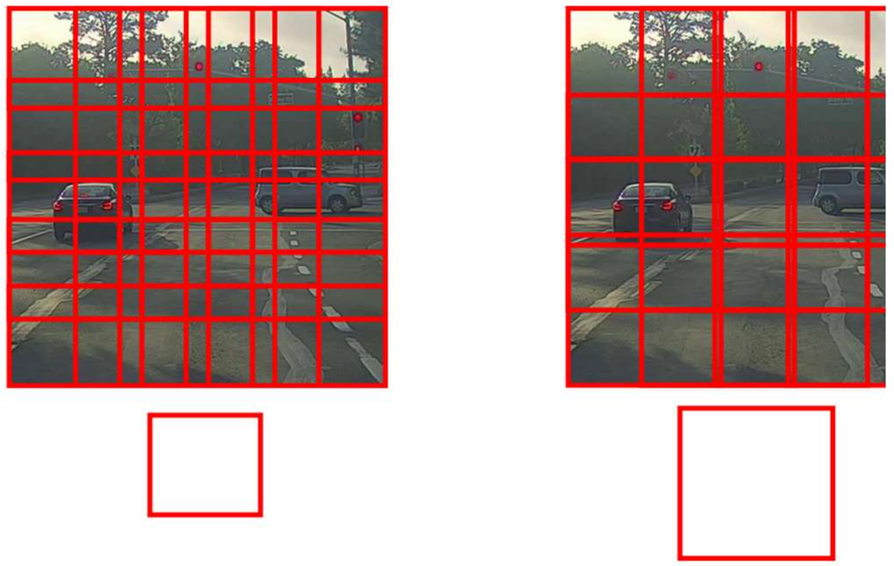




Let's say we want to build a car detection algorithm.
First create a labelled training set.
Take a picture, cut out anything else that's not part of a car and get the car centred in pretty much the entire image (cropped examples of cars).
Train a CNN to output y (0 or 1, is there a car or not).
The trained CNN is used in the Sliding Windows detection algorithm.

Sliding windows detection algorithm

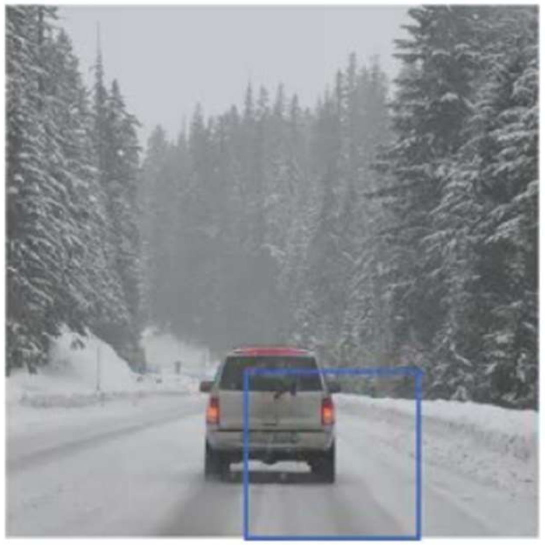




Pick a certain window size, input this sub-picture into already trained CNN to detect objects. Then shift the detection window to the right with one step (stride) and feed the new sub-picture into ConvNet. Go through every region (the stride needs to be small for the detection algorithm to run smoothly). Repeat the same with different sizes of the detection window in order to detect objects with different sizes of the picture.
The Sliding Windows object detection algorithm has **infeasibly high computationally cost.**

Solution: Convolutional Implementation of sliding windows

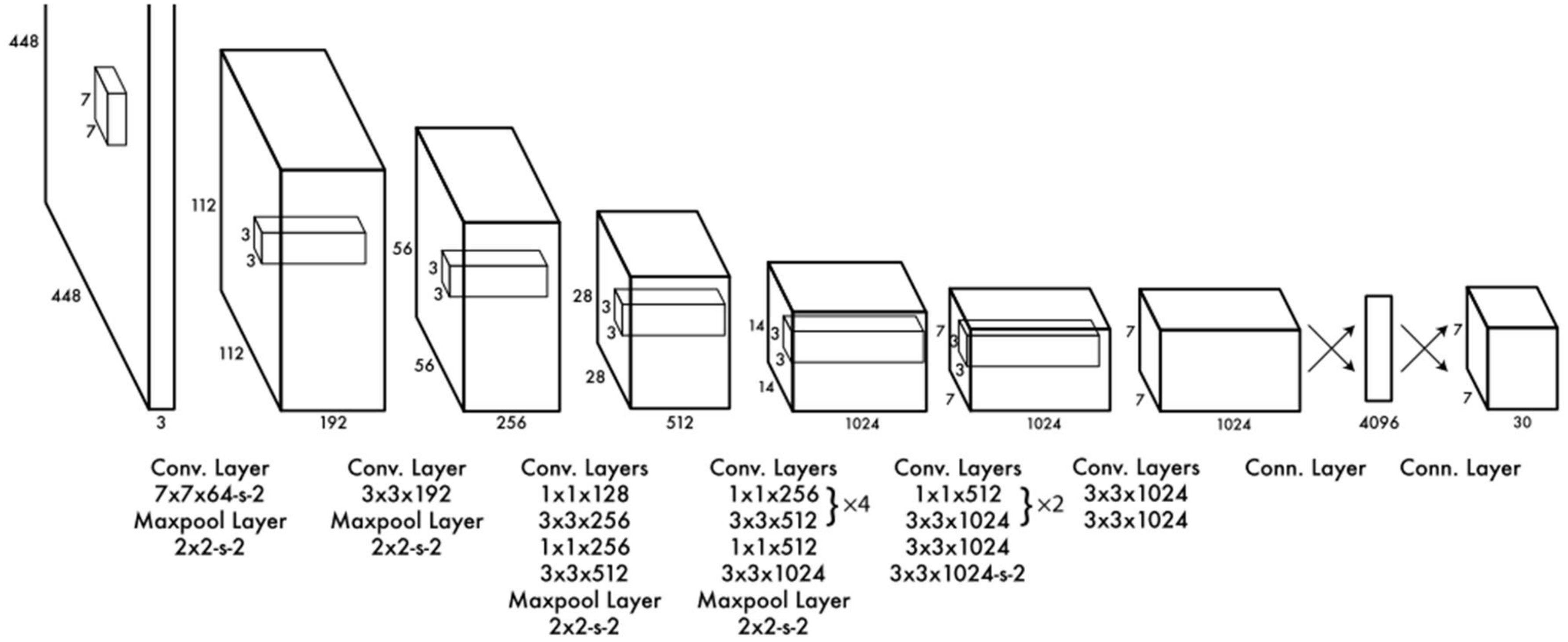
Bounding Box Predictions



With the Sliding Windows algorithm, it may happen that none of the boxes really match perfectly with the position of the target object.
For example, we want to detect the car in this picture. The sliding windows (since we initially set the windows sizes and we don’t not know how big the car is) may match only part of the car.
In some cases, the object may look like a rectangle instead of square.

Solution: YOLO (You Only Look Once) algorithm is a way to output more accurate bounding boxes.

YOLO (You Only Look Once)

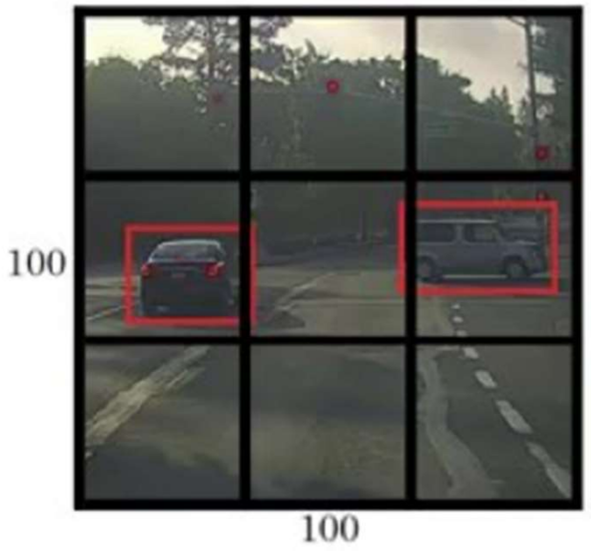


448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512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

YOLO - CNN network for both classification and localising the object using bounding boxes.
24 convolutional layers + 2 fully connected layers.
Conv layers pretrained on ImageNet dataset.

*Redmon et al, 2015, "You Only Look Once: Unified, Real-Time Object Detection" (https://arxiv.org/abs/1506.02640)
Redmon & Farhadi, 2016 (https://arxiv.org/abs/1612.08242).

YOLO (You Only Look Once) algorithm



<= cell label: [0 , ? , ? , ? , ? . ?, ?; ?]
? – "don't care"

<= cell label: [1 , b_x , b_y , b_h , b_w , 0, 0, 1]

<= cell label: [0 , ? , ? , ? , ? . ?, ?; ?]
? – "don't care"

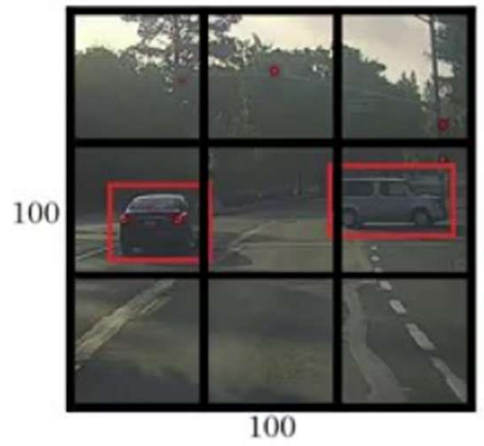
Let we have 100x100 pixels input image. We place a grid on this image.
In the original paper the grid is 19x19, here for illustration is 3x3 grid (9 cells).
Apply object classification and localization to each cell.

How to define the labels used for training:
For each cell, the label is 8 dimensional vector (if we have 3 classes)
y = [p_c, b_x , b_y , b_h , b_w , c1, c2, c3], where

- p_c (0 or 1) specifies if there is or not an object in that cell.
- [b_x , b_y , b_h , b_w] specify the bounding box if there is an object in that cell.
- c1; c2; c3 (0 or 1) - to assign the class (e.g. person, bicycle, car)

YOLO assigns the object only to the cell containing the midpoint of the object.
Since we have 3x3 grid, the total volume of the target output Y is 3x3x8.

YOLO (You Only Look Once) algorithm



$$\underbrace{\text{Image}}_{\text{size: }100\times100\times3} \longrightarrow \text{CONV} \longrightarrow \text{MAXPOOL} \longrightarrow \cdots \longrightarrow \underbrace{\square\square\ldots\square}_{\text{size: }3\times3\times8}$$

<= [b_x= 0.4, b_y=0.6, b_h= 0.7, b_w =0.9]

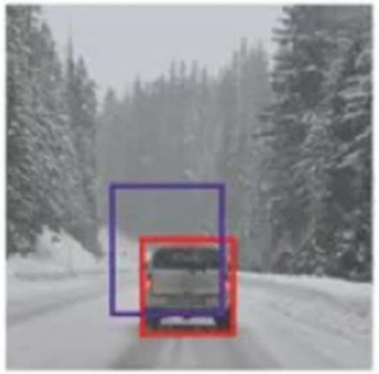
To train YOLO, input e.g. 100x100x3 original image (X).
The network has the usual conv layers, max pool layers, and so on.
It has to end up with 3x3x8 output volume that is compared with the target labels Y (also 3x3x8).

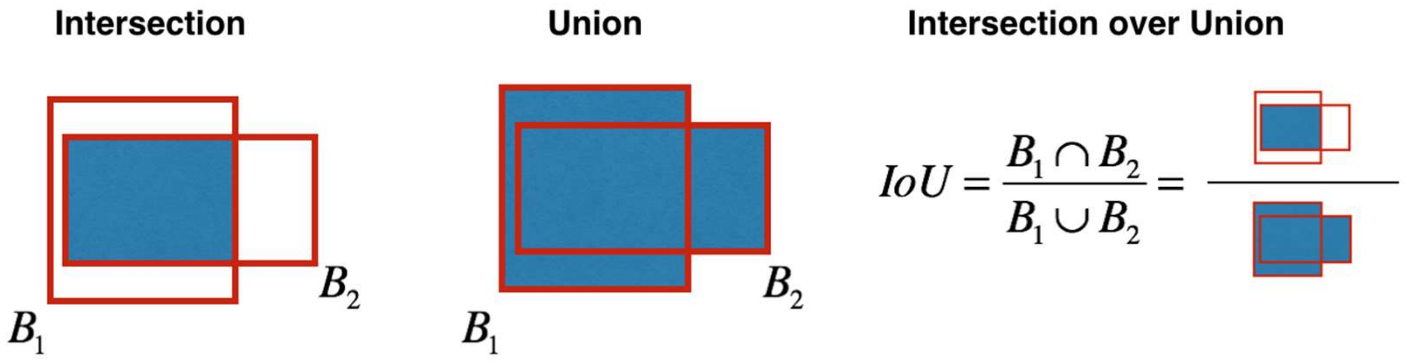
If the grid is much finer (e.g. 19x19), it reduces the chance of multiple objects assigned to the same grid cell.

How to specify the bounding boxes [b_x , b_y , b_h , b_w] ?
YOLO assumes the upper left point of the cell is (0,0), the lower right point is (1,1).
The bounding boxes are specified relative to the grid cell.
[b_x , b_y] represent the midpoint of the object, have to be <1 and >0.
[b_h , b_w] represent the height and width of the bounding box, they could be >1 if the object is bigger than the cell.
There are other ways to specify the bounding boxes.

universidade de aveiro

Intersection Over Union



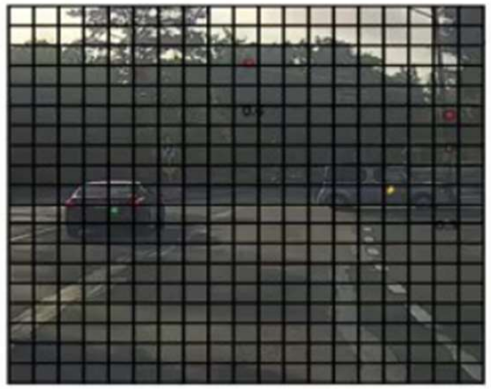


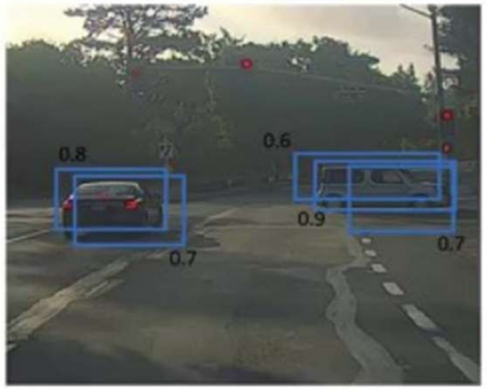
$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} =$$

How to tell if the object detection algorithm is working well?
We use the function called intersection over union (IoU) .
IoU computes the intersection over union of the two bounding boxes (red is the ground true, purple is the YOLO prediction).
IoU is a measure of the overlap between two bounding boxes (measure of similarity).
Higher IoU, better prediction:

IoU=1 => perfect detection
IoU >=0.5 => "correct" (by convention)

Non-max Suppression





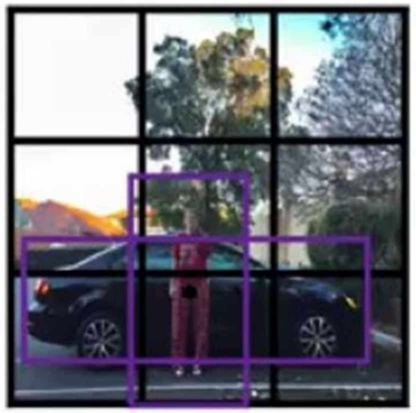
Common object detection problem: <u>the algorithm finds multiple detections of same object</u> =>
Non-max suppression is a way to guarantee that each object is detected only once.

Let's say we want to detect 3 classes - persons, bikes, cars.
We placed 19x19 grid. Technically each car has just one midpoint, so it should be assigned just to one grid cell.
In practice, we run object classification & localization algorithm for every cell.
It's possible that many neighbour cells claim they found a car.

Non-max suppression cleans up these detections **<u>independently for each class</u> (persons, bikes,cars).**

1) Discard all boxes with low probability (let say $p_c \leq 0.6$)
2) Pick the box with the largest p_c. Output that as a prediction.
3) Discard any remaining box with high overlap (e.g. IoU ≥ 0.5) with the box picked in step 2).

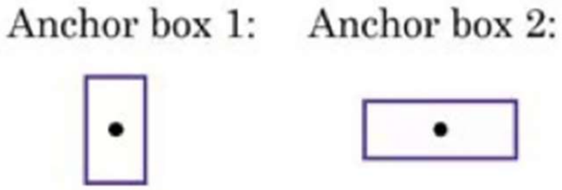
Anchor boxes



Previously, each object in training image is assigned to grid cell that contains that object's midpoint. The label vector is (8 elements)

$y = [p_c, b_x, b_y, b_h, b_w, c1, c2, c3]$

Now 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 the highest IoU. The label vector is



$$y = [\underbrace{p_c; b_x; b_y; b_h; b_w; c_1; c_2; c_3}_{\text{anchor box 1}}; \underbrace{p_c; b_x; b_y; b_h; b_w; c_1; c_2; c_3}_{\text{anchor box 2}}]$$

Object detection problem: Each grid cell can detect only one object. *What if a grid cell has multiple objects?*

One idea is to use anchor boxes. In the image above, the midpoints of the person and the car are very close and fall into the same grid cell => the algorithm has to pick one of the two detections to output.

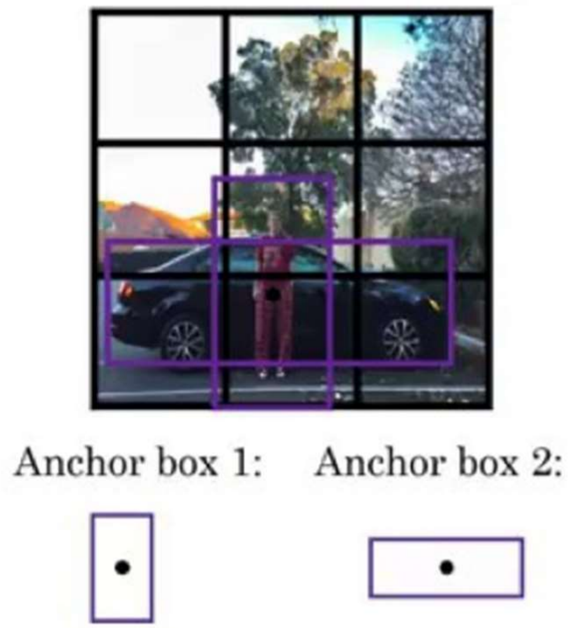
The idea of anchor boxes is to **pre-define two different shapes called, anchor boxes** and associate each prediction with one of the anchor boxes.

For example, anchor box for human may be a vertical rectangle while that for car may be a horizontal rectangle.

e.g. The label for the lower middle grid cell of the image above has 16 elements:

$y = [1, b1_x, b1_y, b1_h, b1_w, 1, 0, 0, \quad 1, b2_x, b2_y, b2_h, b2_w, 0, 0, 1]$

Anchor boxes



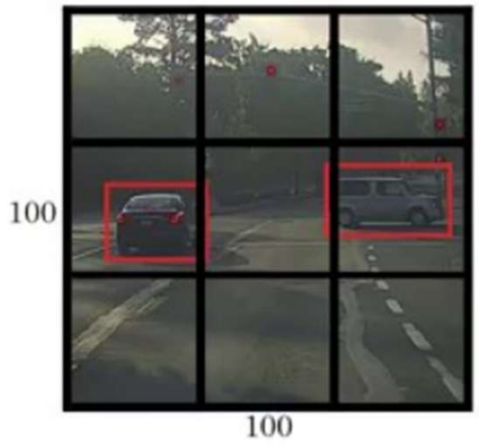
If the cell grid is fine (e.g. 19x19 and not 3x3) the probability of two objects to appear in the same grid cell is low.

Better motivation for the use of anchor boxes is to allow the learning algorithm to specialise better if the data set has specific shapes (tall, wide shapes, others) .

How to choose the anchor boxes: **manually choose 5-10 different shapes** that span the variety of possible shapes in the data set.

More advanced way: **k-means algorithm to group to shapes of multiple classes.**

YOLO Training



Get labelled data: set of Xp x Yp x 3 images;
Label dimension (**tensor):** (grid of cells) x (# anchors) x (p_c+4 box coordinates+#classes)

In general, we may use multiple anchor boxes (e.g.5 in the lab).

Train CNN with the same output dimension as data labels.

Ex. Label for the upper right cell of the image above (assuming 2 anchors (tall, wide), 3 classes)

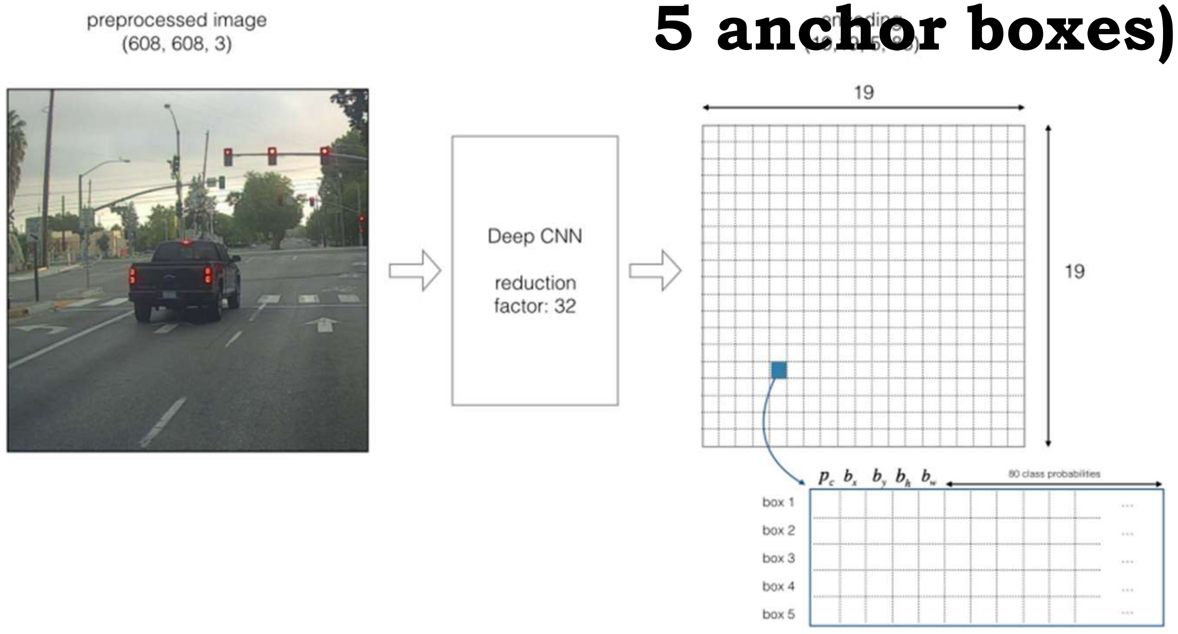
y = [0, ? , ? , ? , ? , ?, ?, ?,   0, ? , ? , ? , ? , ?, ?, ?] (? Don't care)

Label for the right middle cell of the image above:

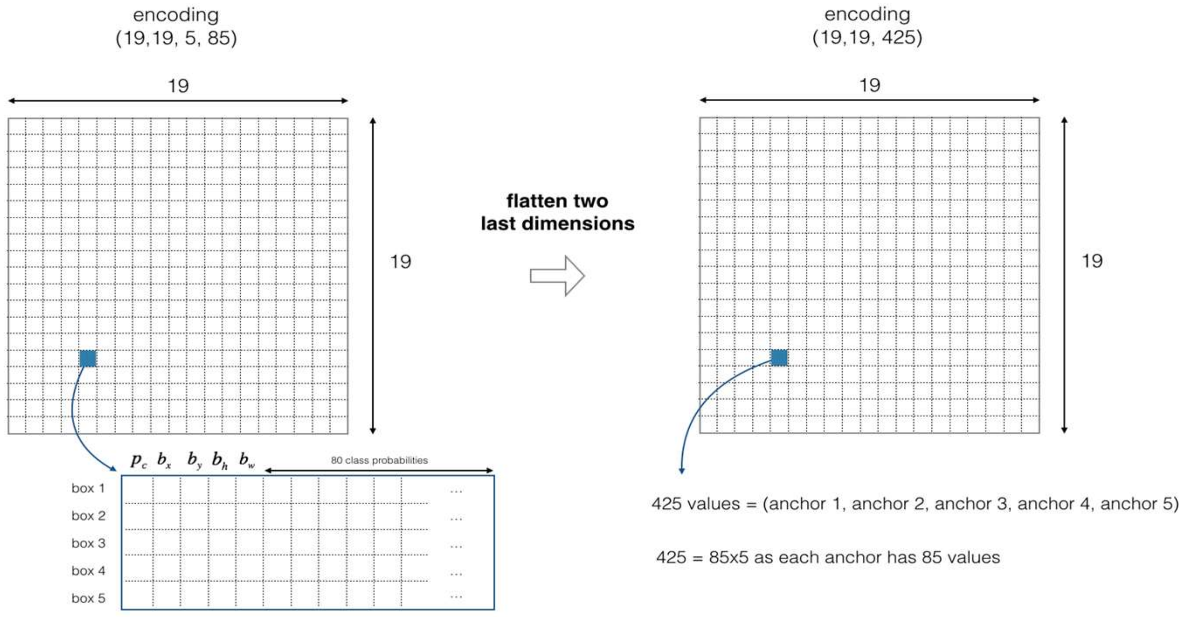
y = [0, ? , ? , ? , ? , ?, ?, ?, 1, $b2_x$, $b2_y$, $b2_h$, $b2_w$, 0, 0, 1]

Ex. Input (100x100x3 image) => CNN => Output (3x3x2x8)
CNN outputs always numbers, cannot output don't care.

Lab work: Image encoding (19x19 cell grid, 80 classes, 5 anchor boxes)

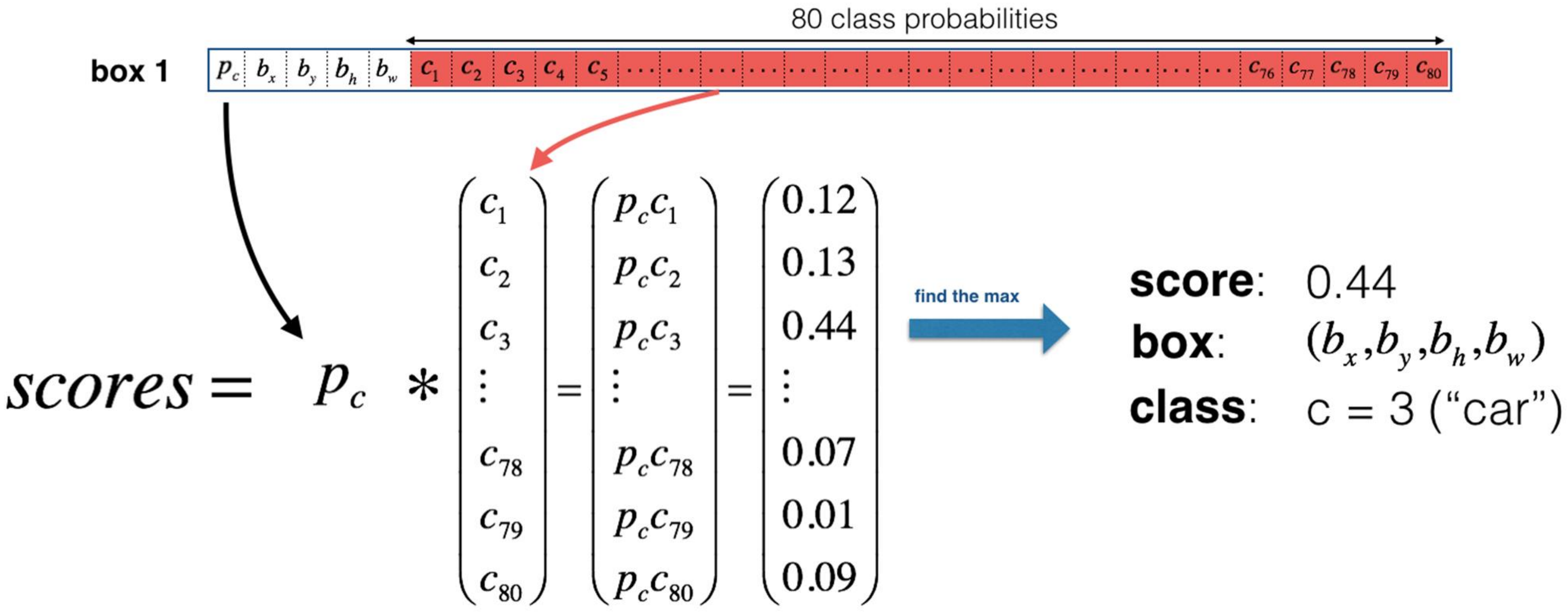


YOLO runs fast and is useful for real time object detection, because it is one pass convolutional implementation. It is not implemented sequentially for each cell (19x19 grid).



encoding
(19,19, 5, 85)
19
19
p_c b_x b_y b_h b_w
80 class probabilities
box 1
box 2
box 3
box 4
box 5
flatten two last dimensions
encoding
(19,19, 425)
19
19
425 values = (anchor 1, anchor 2, anchor 3, anchor 4, anchor 5)
425 = 85x5 as each anchor has 85 values

Lab work: CNN output processing & filtering



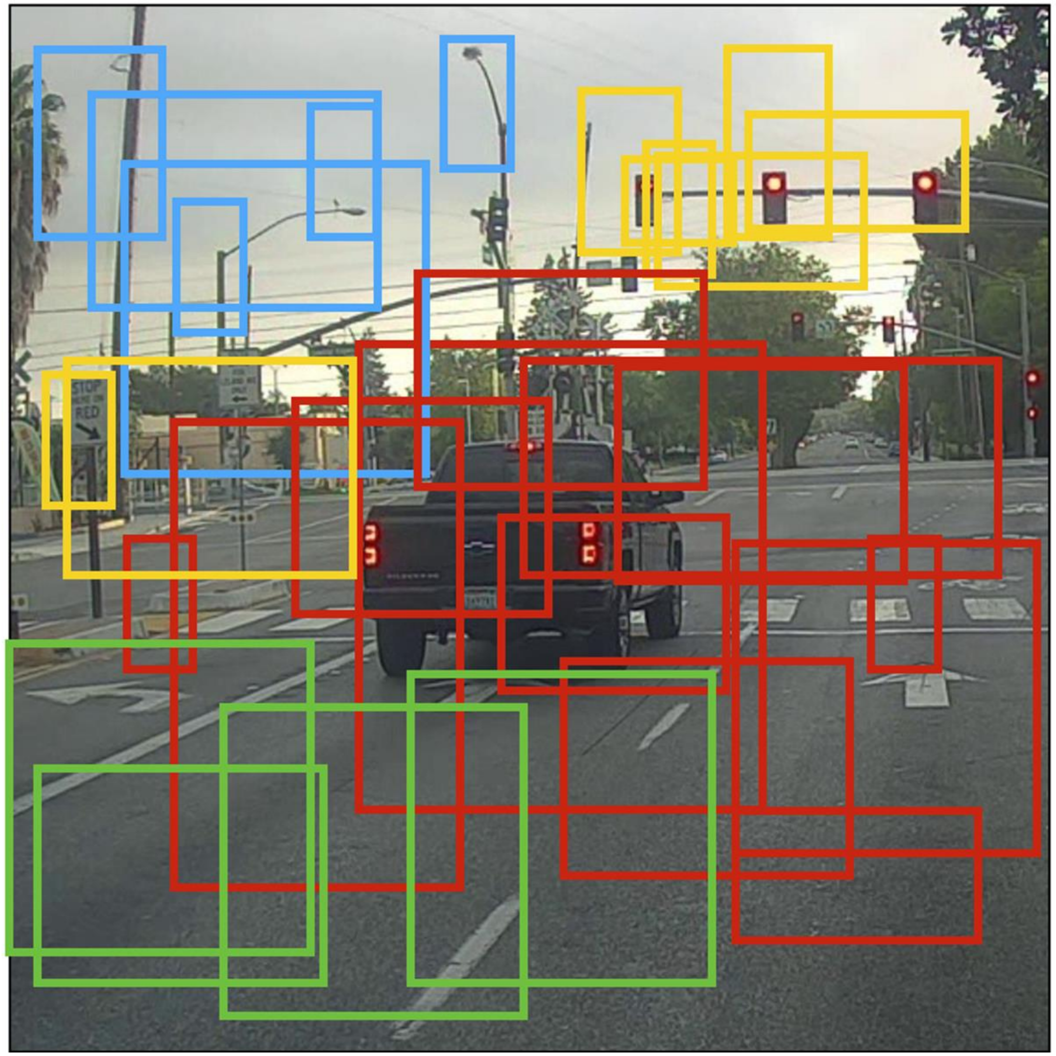
80 class probabilities
box 1
p_c b_x b_y b_h b_w c_1 c_2 c_3 c_4 c_5 … c_{76} c_{77} c_{78} c_{79} c_{80}
find the max

$$scores = p_c * \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{78} \\ c_{79} \\ c_{80} \end{pmatrix} = \begin{pmatrix} p_c c_1 \\ p_c c_2 \\ p_c c_3 \\ \vdots \\ p_c c_{78} \\ p_c c_{79} \\ p_c c_{80} \end{pmatrix} = \begin{pmatrix} 0.12 \\ 0.13 \\ 0.44 \\ \vdots \\ 0.07 \\ 0.01 \\ 0.09 \end{pmatrix}$$

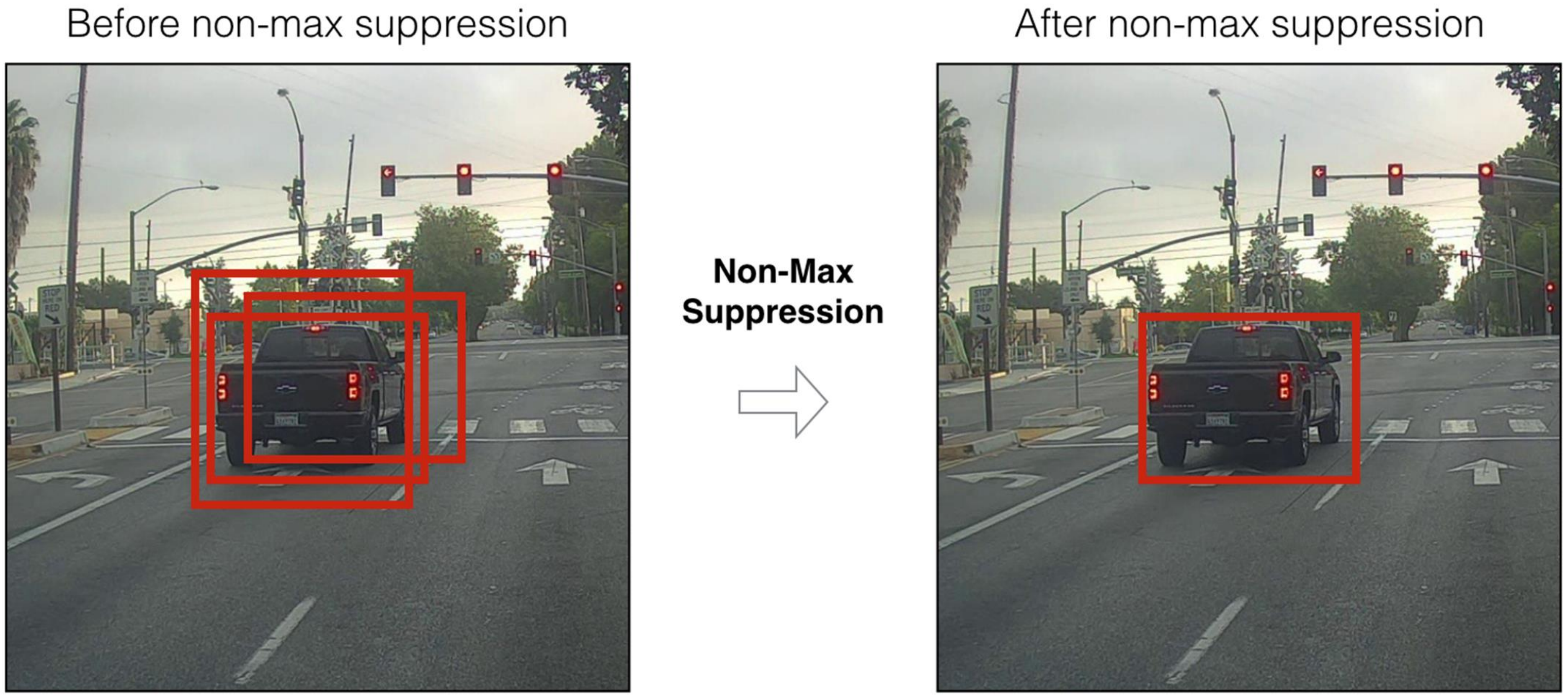
score: 0.44
box: (b_x, b_y, b_h, b_w)
class: c = 3 (“car”)

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

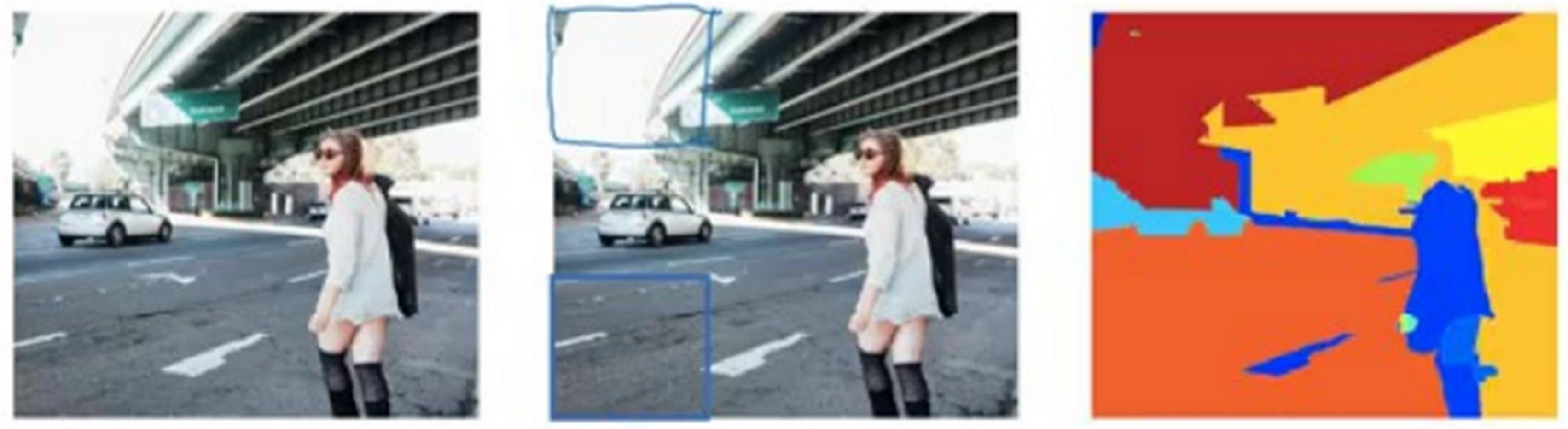
Lab work: Remove boxes with low probability



Lab work: Non Max Supression (NMS)



Object Detection with Region proposal CNN



YOLO algorithm classifies a lot of empty regions.
Region proposal CNNs select just a few regions (windows).

First run a segmentation algorithm, find interesting blobs (e.g. 2000), place bounding boxes around them and run CNN classifier.

Object detection using regions with CNNs goes over the following steps:

1) Find regions in the image that might contain an object. These regions are called *region proposals*.

2) Extract CNN features from the region proposals.

3) Classify the objects using the extracted features.

Summary: R-CNN/Fast R-CNN/Faster R-CNN

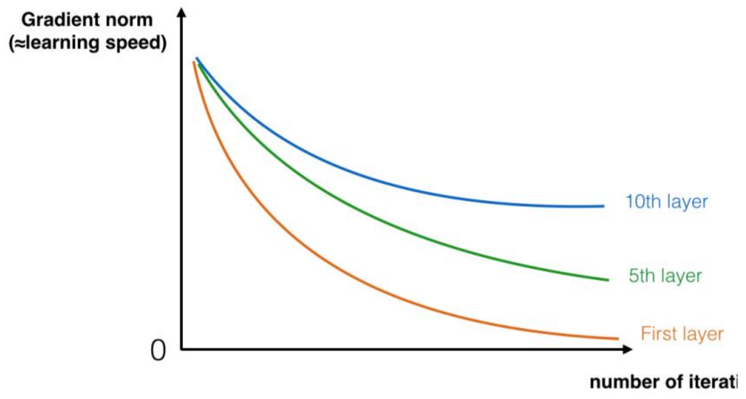
R-CNN*: Apply segmentation algorithms to propose regions.
Classify proposed regions one at a time. Output label + bounding box.

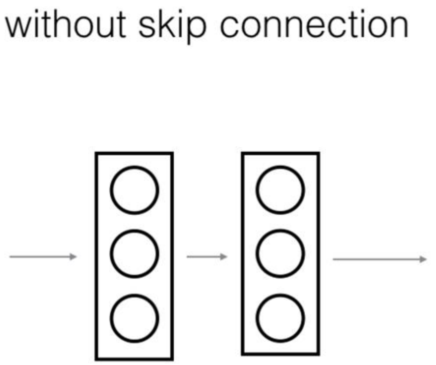
Fast R-CNN**: Apply segmentation algorithms to propose regions.
Use convolution implementation of sliding windows to classify all the proposed regions simultaneously.

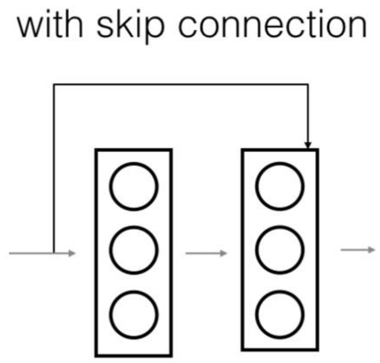
Faster R-CNN***: Use CNN to propose regions (instead of traditional segmentation alg.).
Use convolutional implementation of sliding windows to classify the regions simultaneously.

* 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.

ResNet (skip connection)







Vanishing gradient **Residual block**

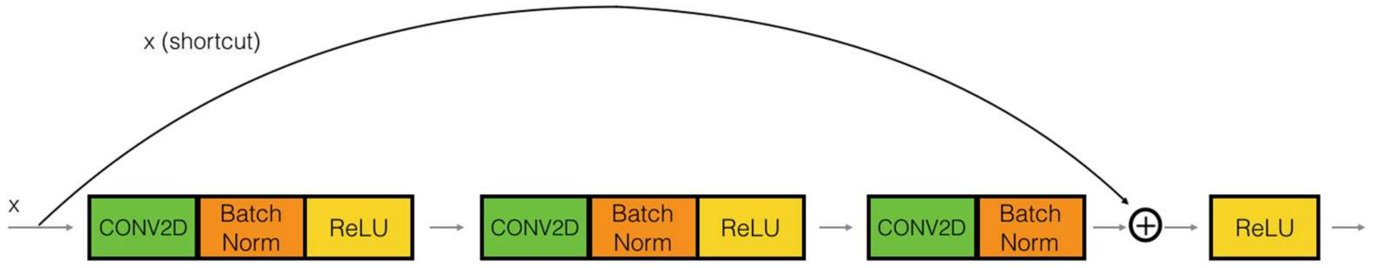
DNN are difficult to train because of vanishing or exploding gradient types of problems.

Skip connection (short cut) takes the activation from one layer and feed it to another layer much deeper in the network.
Skip connection build networks that enable to train very deep structures (even over 100 layers).
ResNets are built with residual blocks.

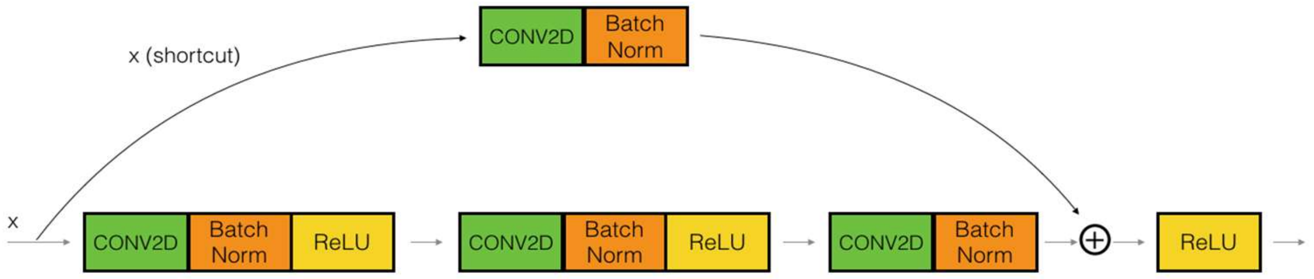
Exploding gradient is easier to spot (NaN in computations – numerical overflow) => apply gradient clipping, i.e. limit the gradients up to some max value.

Residual blocks

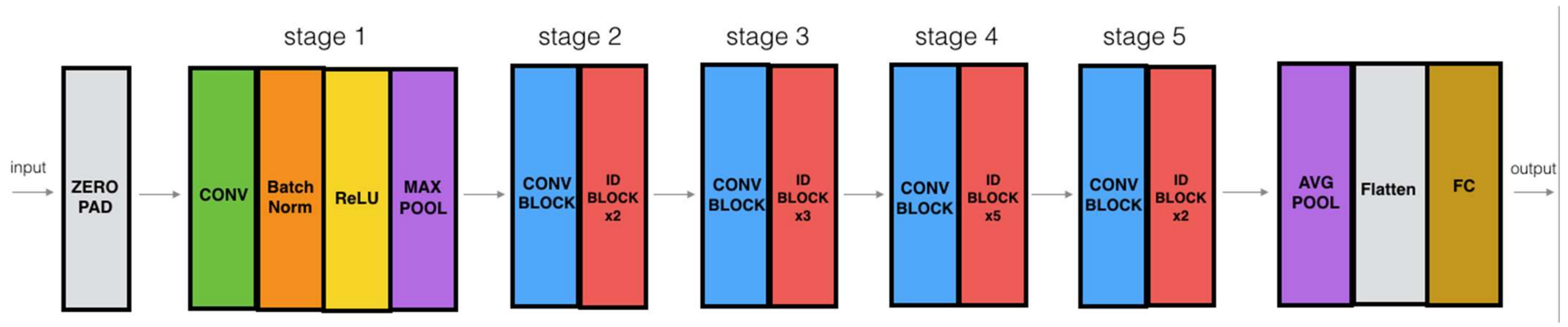
Identity skip connection /shortcut



Convolution skip connection /shortcut



ResNet 50 model



ResNet-50 has 5 stages with Convolution and Identity blocks.

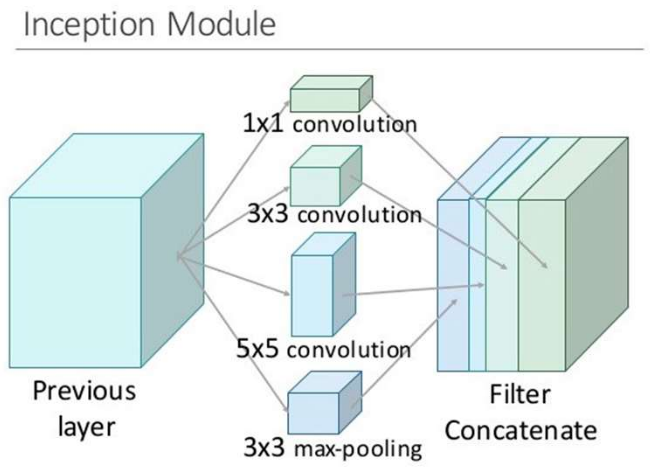
Each Conv BLOCK has 3 convolution layers and each identity block (ID BLOCK) also has 3 convolution layers.

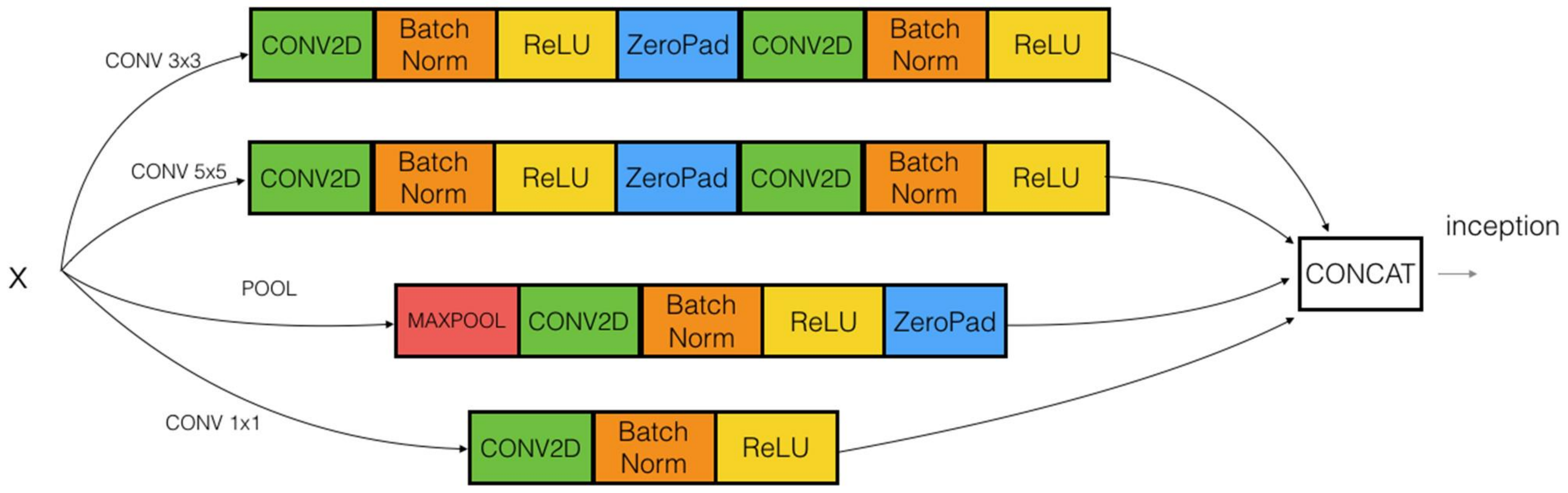
Stage1(1)+Stage2(3+2*3)+ Stage3(3+3*3)+ Stage4 (3+5*3)+ Stage5 (3+2*3)+1FC =50

ResNet-50 has over 23 million trainable parameters !!!

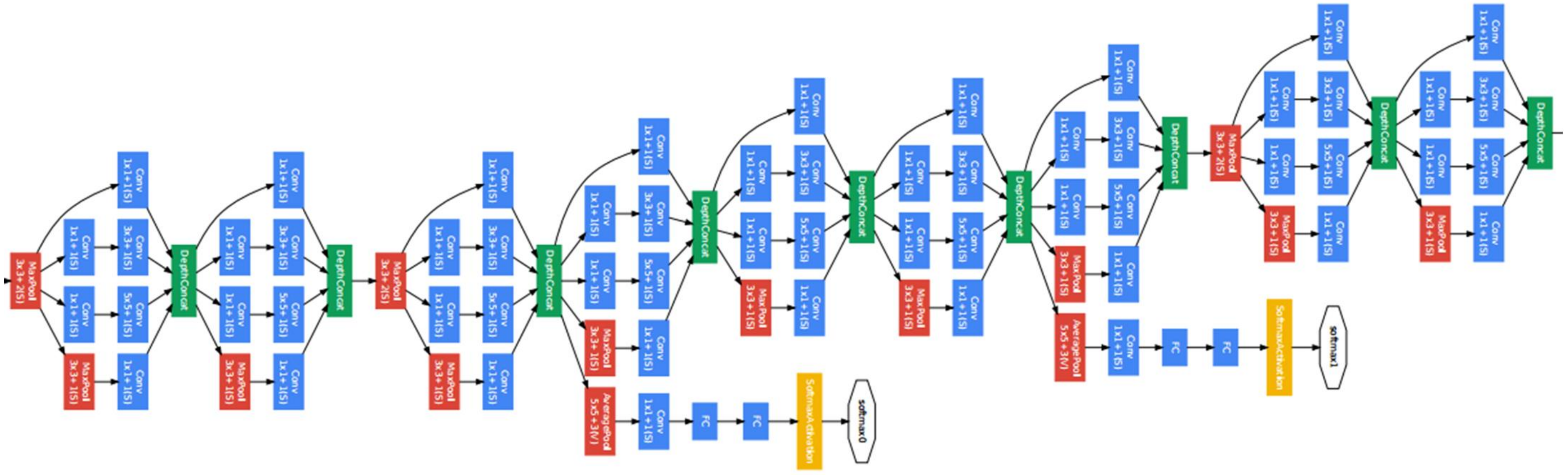
INCEPTION MODULE – various architectures

In the inception module instead of choosing what filter size to use, choose a group of different filters together, e.g. 1x1; 3x3; 5x5 convolution filters, max pooling, etc. and stuck the outputs.





INCEPTION NETWORK (GoogLeNet)



Ref. Szegedy et al 2014, “Going deeper with Convolutions”. Arround 25 mln. Of parameters

universidade de aveiro