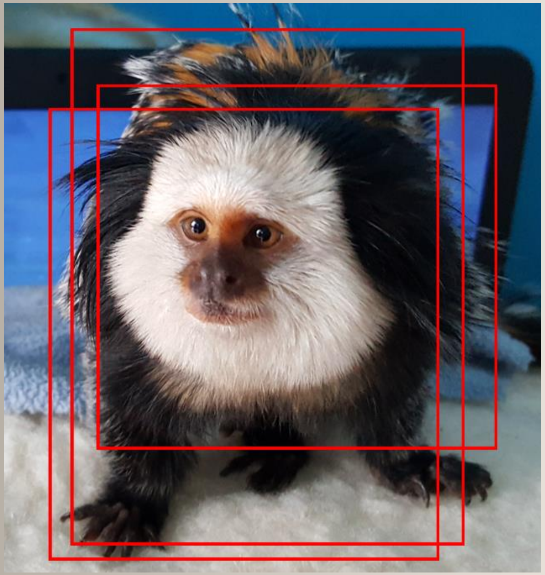
# DEEP LEARNING

## Object Classification, Detection, Localization and Segmentation

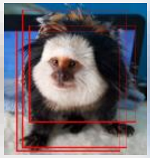


Adrian Horzyk

horzyk@agh.edu.pl

AGH

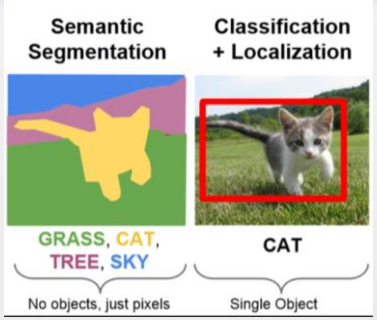
AGH University of
Science and Technology
Krakow, Poland



# Object Classification, Localization and Detection

**Tasks that can be performed on images:**

- **Classification**
- **Classification with localization**
- **Detection**
- **Instance Segmentation**
- **Semantic Segmentation**



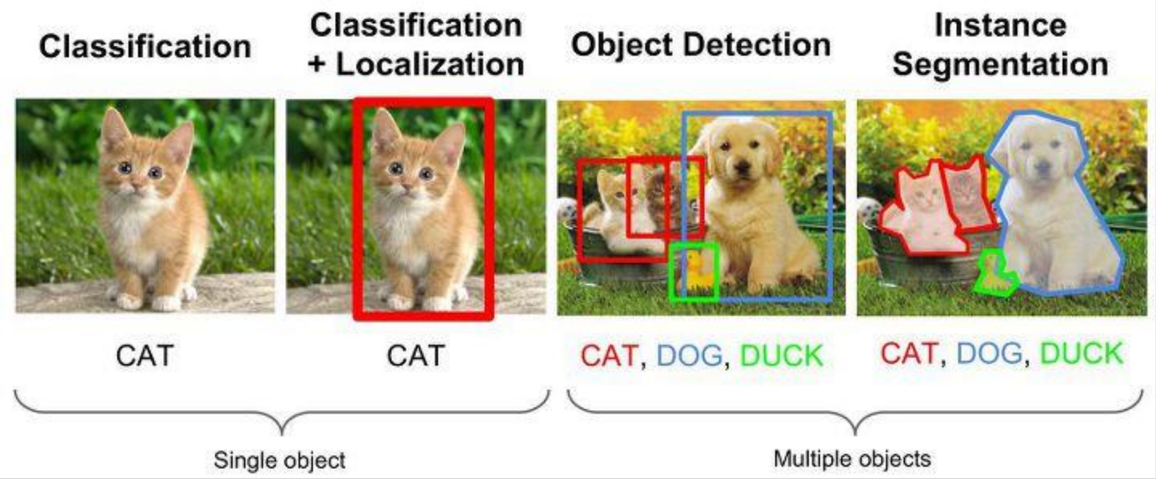
**Classification** is to determine to which class belongs the main object (or sometimes all objects) in the image.

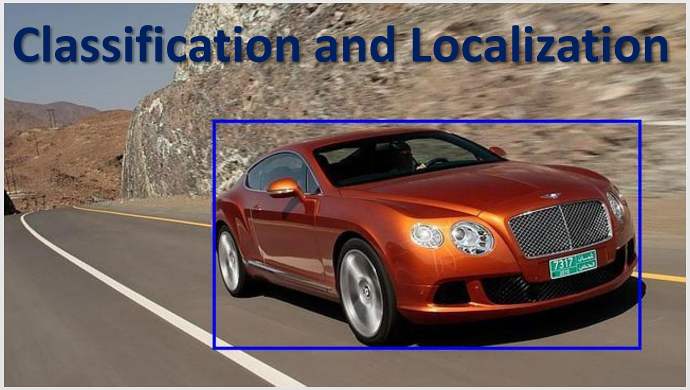
**Classification with localization** not only classifies the main object in the image but also localizes it in the image determining its bounding box (position and size or localization anchors).

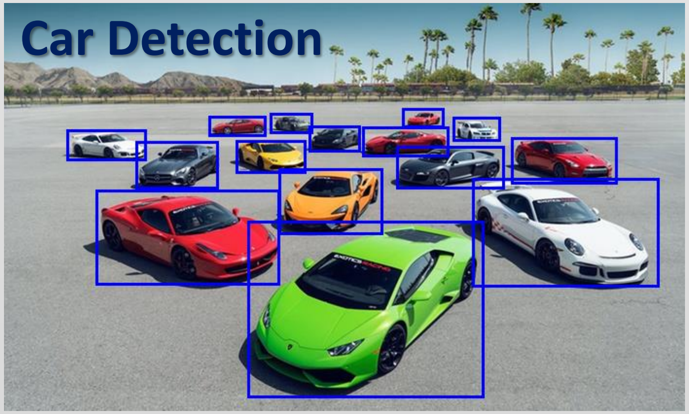
**Detection** tries to find all object of the previously trained (known) classes in the image and localize them.

**Instance Segmentation** is to …

**Semantic Segmentation** is to distinguish between …



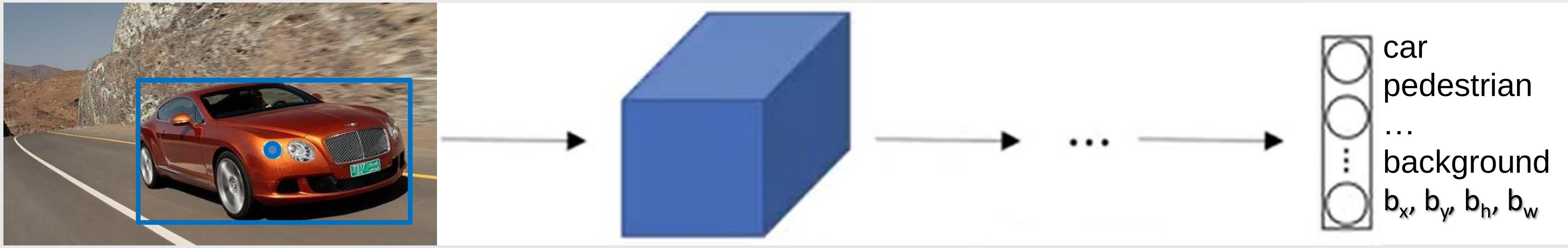




# Classification with Localization

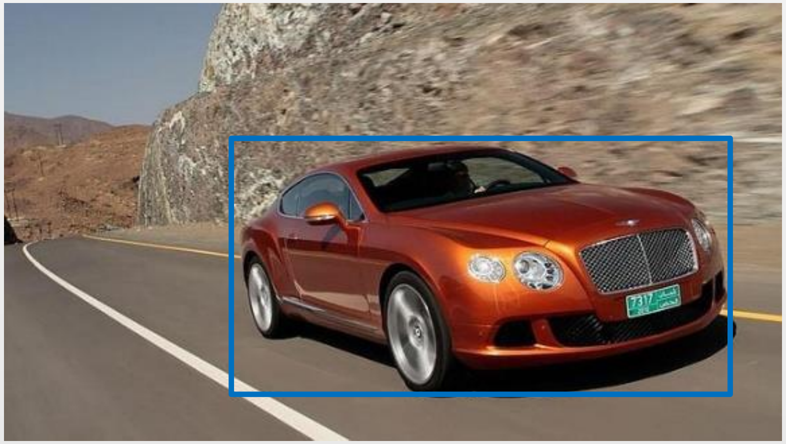
**Classification using DL is to determine the class of the main object (that is usually in the centre of the image):**

- **The number of classes is usually limited and the rest is classified as background or nothing:**



- **When localizing the object the output of the network contains extra outputs for a defining bounding box ($b_x$, $b_y$, $b_h$, $b_w$) of the object:**
- **$b_x$ – x-axis coordinate of the center of the object**
- **$b_y$ – y-axis coordinate of the center of the object**
- **$b_h$ – height of the object (its bounding box)**
- **$b_w$ – width of the object (its bounding box)**

# Defining Target Labels for Training



**Example 1:** If there is an object of class $c_2$:

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

**Example 2:** If there is no object of any of the defined classes:

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

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

Where

$p_c$ – probability of the detection of an object of the specified class in the image, which is equal to 1 when the object is present and 0 otherwise during the training

$b_x$ – x-coordinate of the bounding box of the object

$b_y$ – y-coordinate of the bounding box of the object

$b_h$ – height of the bounding box of the object

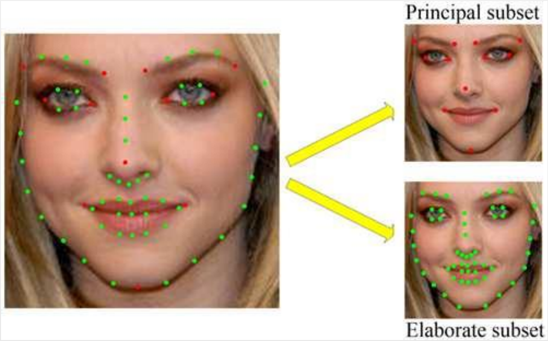
$b_w$ – width of the bounding box of the object

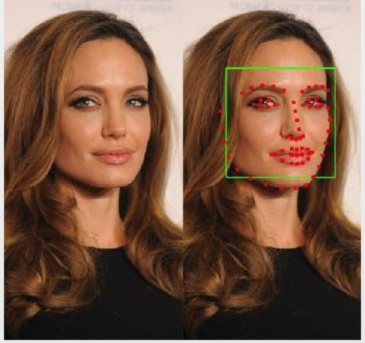
$c_1, c_2, \dots, c_K$ – the possible trained classes of the input image, where only one $c_k$ is equal to 1 and the others are equal to 0

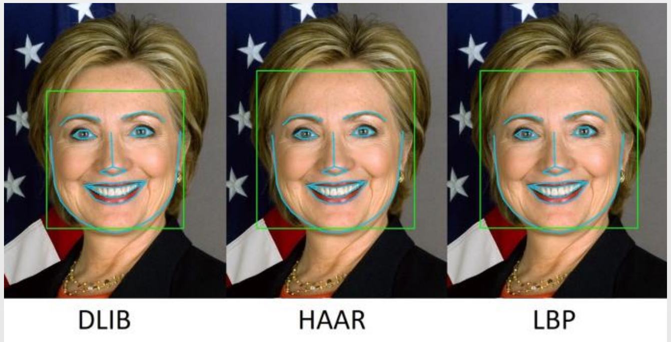
? – are not taken into account in the loss function because we do not care these values while no object is detected

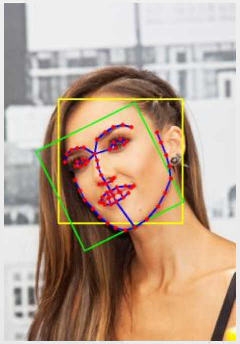
# Landmark Detection

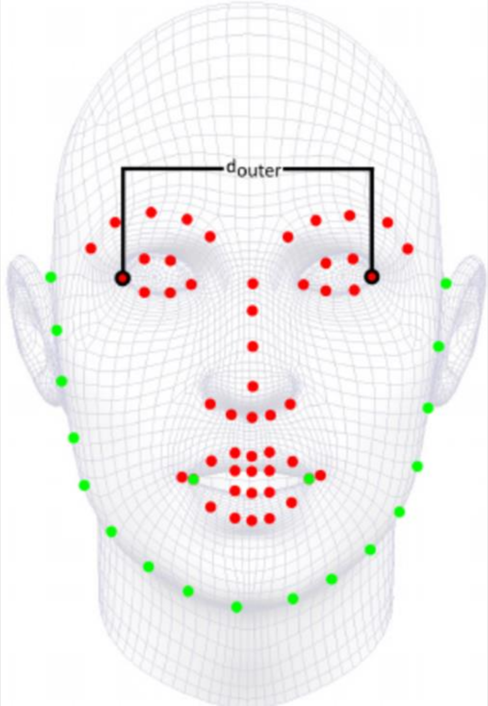
**In the similar way, we can detect various landmarks in the images and use it to compute facial gesture, emotion expressions or model it:**

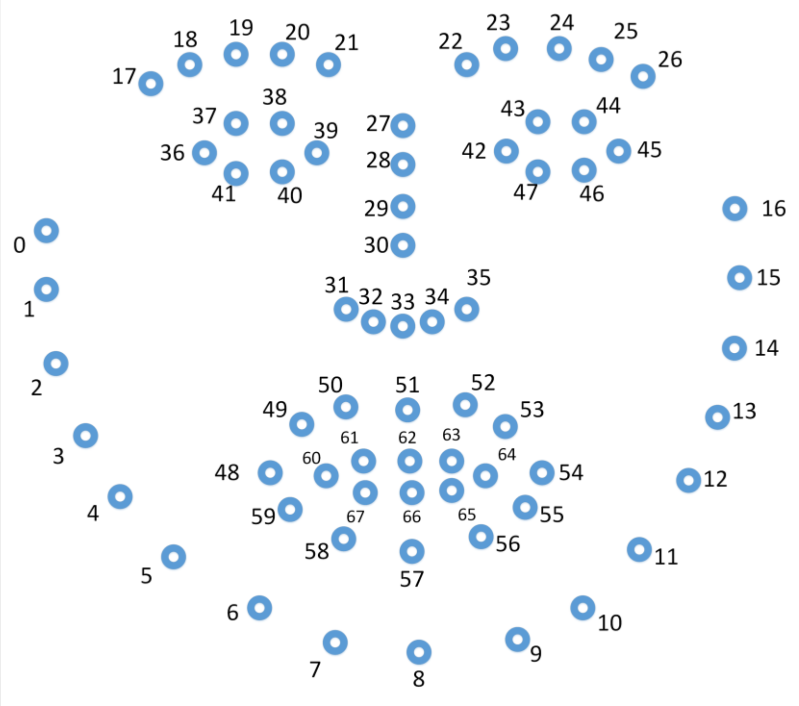


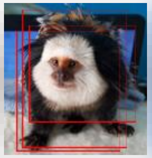










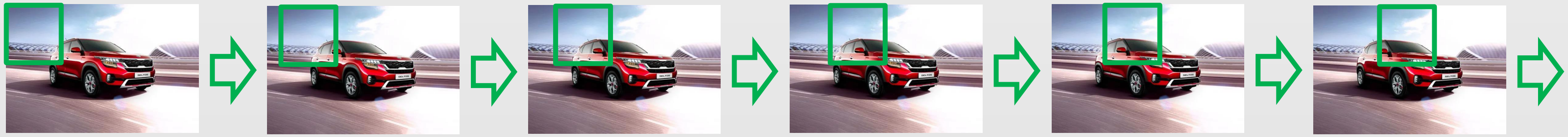


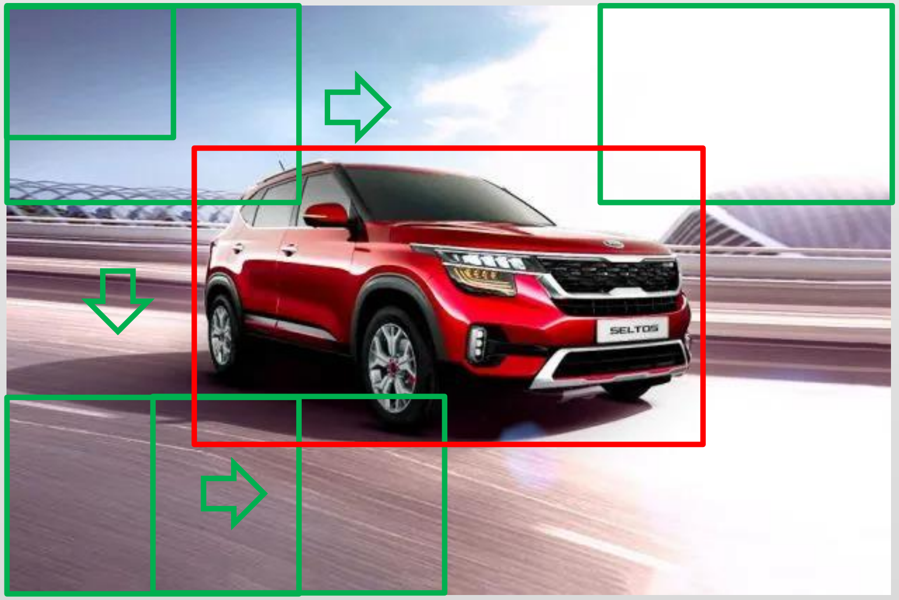
# Object Detection and Cropping Out

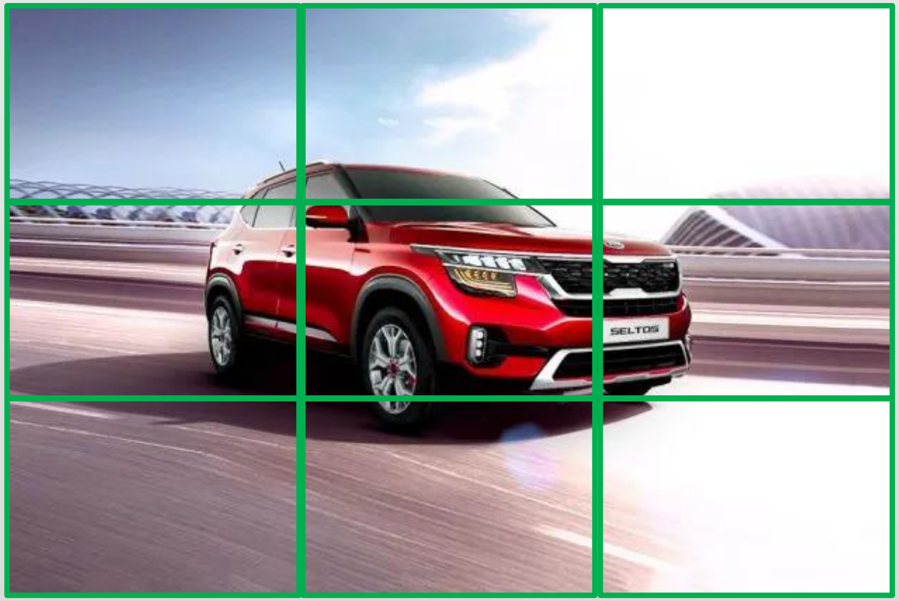
## Object detection can be made in a few ways:

- using sliding window of the same size or various sizes with different strides (high computational cost because of many strides) – sliding window detection
- using a grid (mesh) of fixed windows (YOLO – you only look once)

## and put the cropped image on the input of the ConvNet:



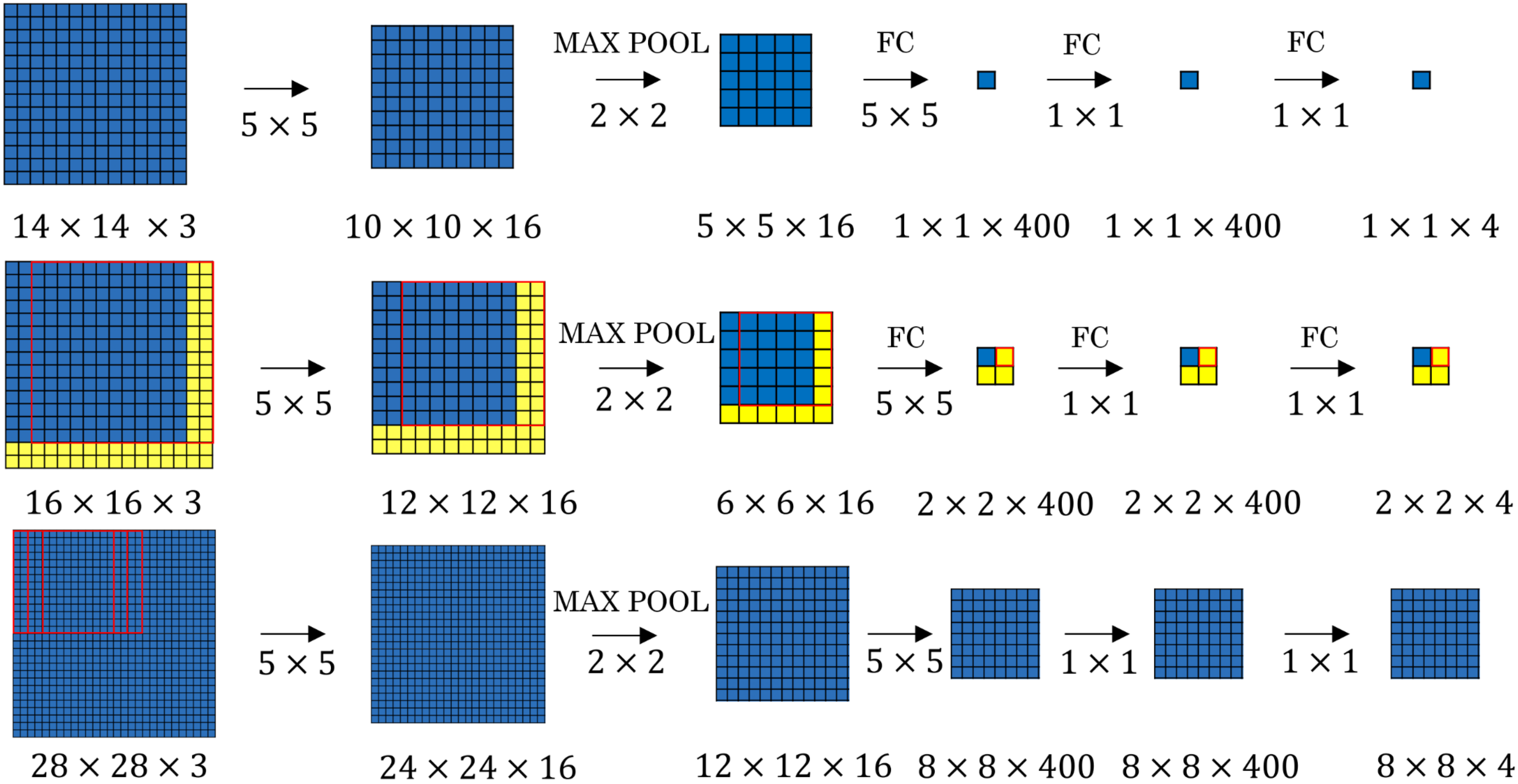


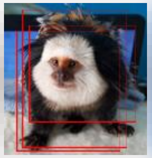


# Convolutional Implementation of Sliding Windows

Many computations for sliding windows repeat as presented by the blue sliding window and the red one (the shared area) after the two-pixel stride.

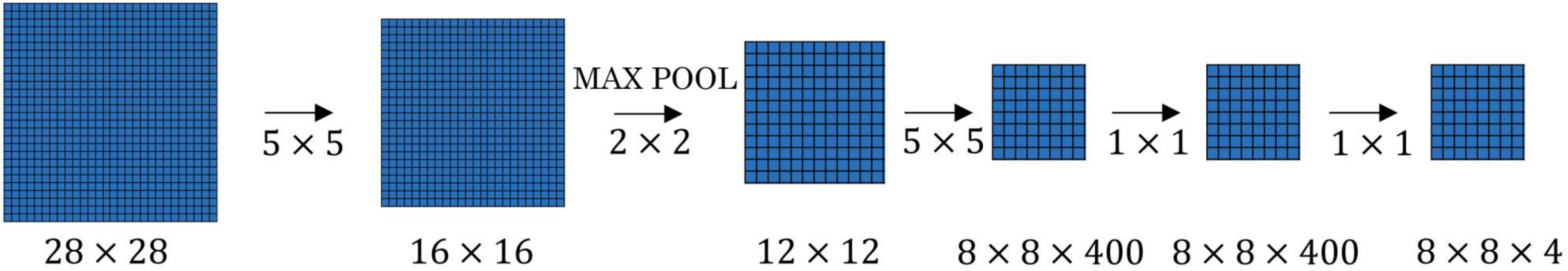
Therefore, we implement sliding windows parallelly and share these computations that are the same for different sliding windows to proceed computations faster.

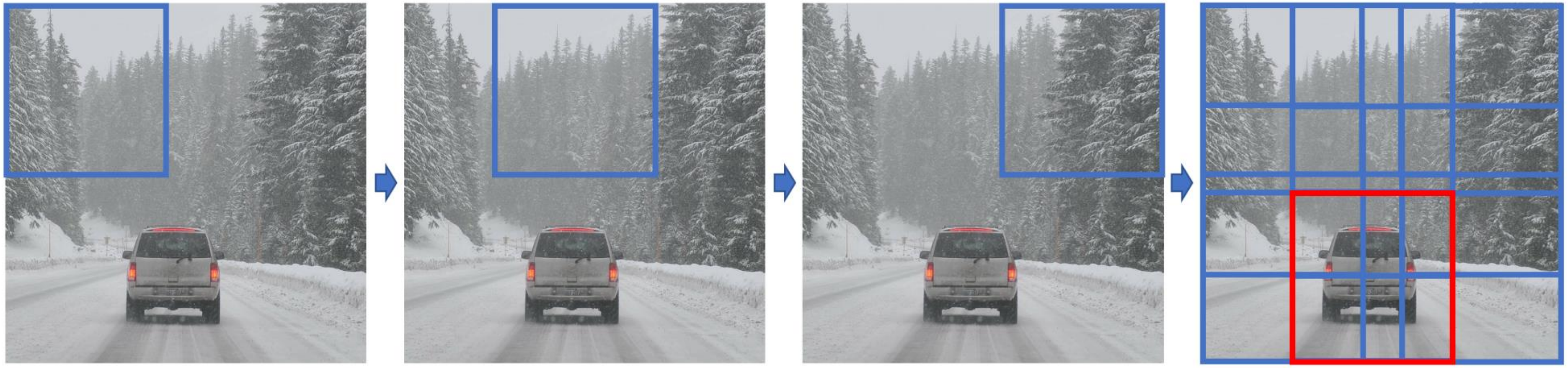




# Convolutional Implementation of Sliding Windows

We can see how the convolutional implementation of the sliding window works on the image. The drawback is the position of the bounding box designated by the sliding window might not be very accurate. Moreover, if we want to fit each object better, we have to use many such parallel convolutional networks for various sizes of sliding windows. Even though we cannot use appropriately adjusted sizes of such windows and achieve poor bounding boxes for the classified objects.





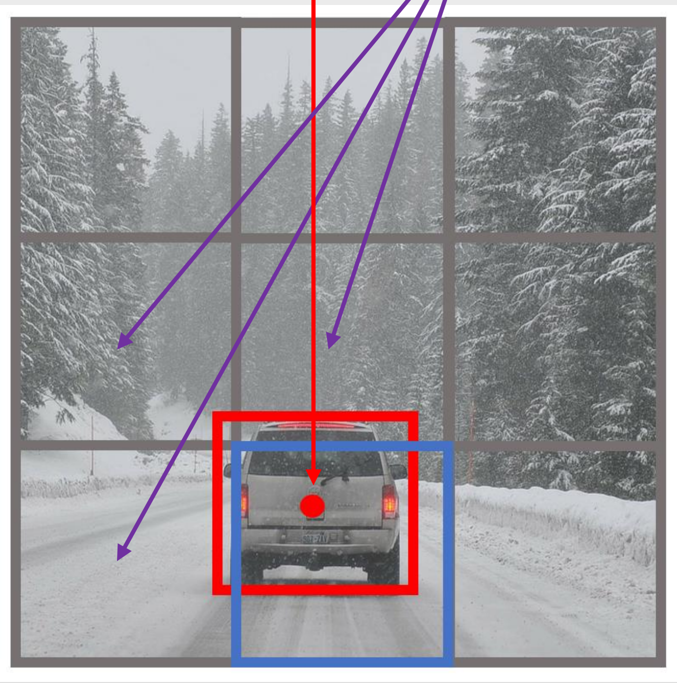
# YOLO – You Only Look Once

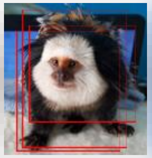
## In YOLO, we put the grid of the fixed sizes on the image:

- Each object is classified only in a single grid cell where is the midpoint of this object taking into account the ground-truth frame of it defined in the training dataset:
- In all other cells, this object is not represented even if they contain fragments of this object or its bounding box (frame).
- For each of the grid cell, we create an (K+5)-dimensional vector storing bounding box and class parameters:

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

- The target (trained) output is a 3D matrix of S x S x (K+5) dimensions, where S is the number of grid cells in each row and column.
- This approach works as long as there is only one object in each grid cell. In practice, the grid is usually bigger than in this example, e.g. 19x19, so there is a less chance to have more one middle point of the object inside each grid cell.





# YOLO's bounding boxes

**The YOLO's bounding boxes are computed using the following formulas:**

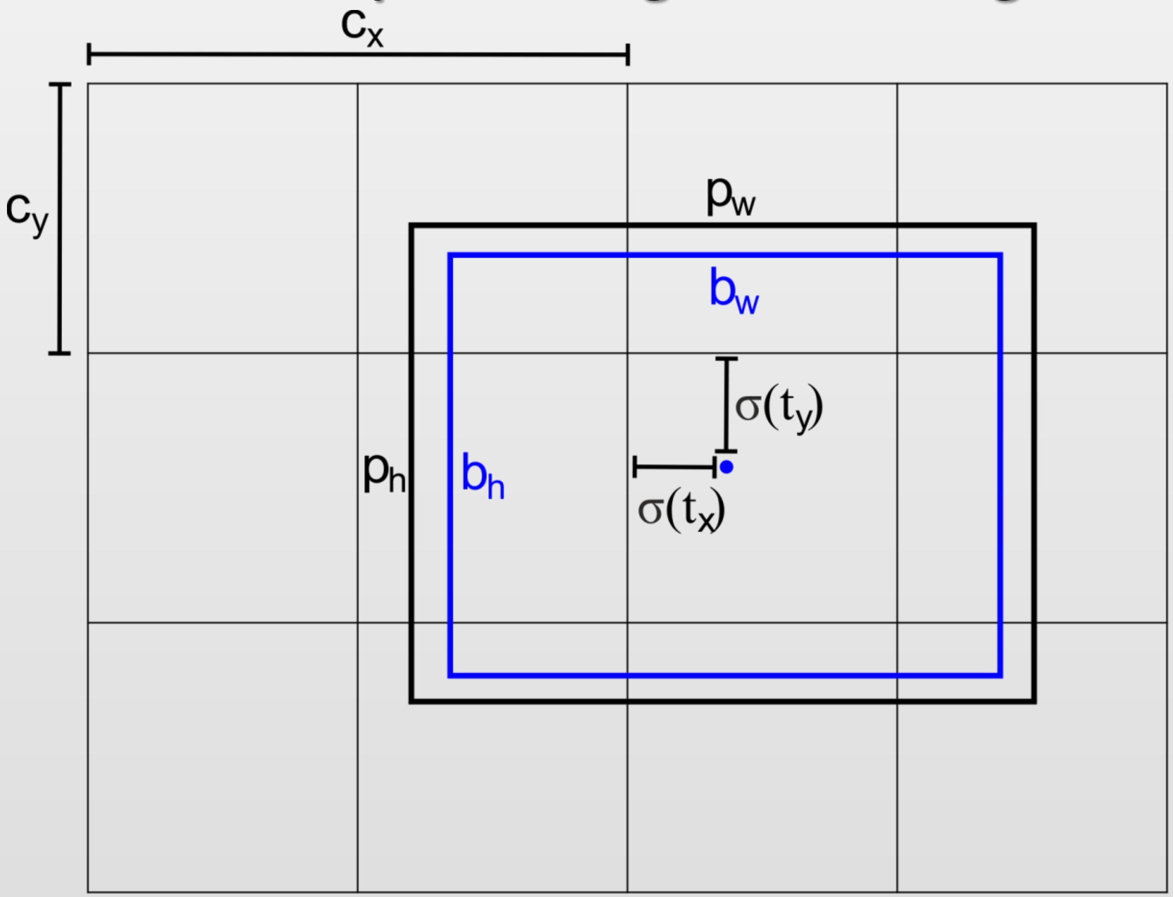
$$(b_x, b_y, b_w, b_h)$$

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w \cdot e^{t_w}$$

$$b_h = p_h \cdot e^{t_h}$$



where

$t_x$, $t_y$, $t_w$, $t_h$ is what the YOLO network outputs,

$c_x$ and $c_y$ are the top-left coordinates of the grid cell, and

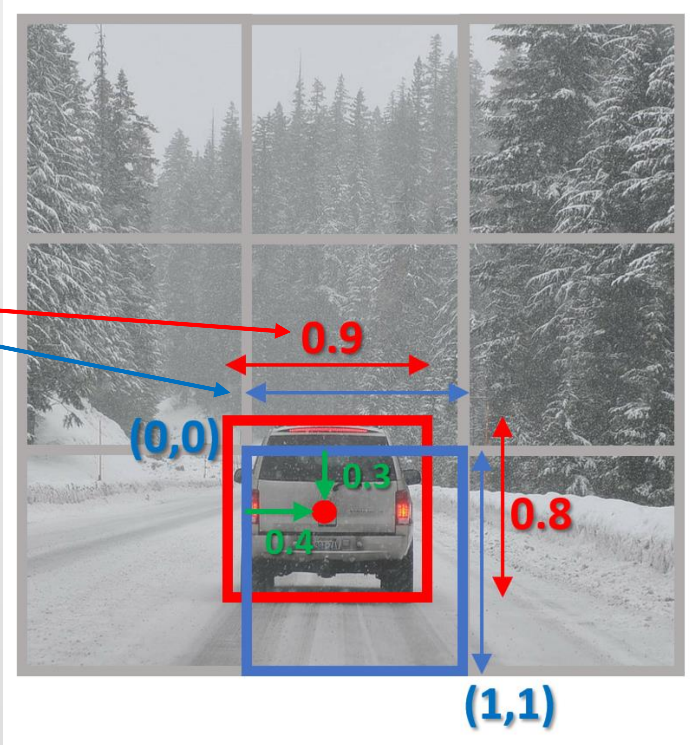
$p_w$ and $p_h$ are the anchors dimensions for the grid cell (box).

# Specifying the Bounding Boxes in YOLO

## We specify the bounding boxes in YOLO in such a way:

- Each upper-left corner of each grid cell has (0,0) coordinates.
- Each bottom-right corner of each grid cell has (1,1) coordinates.
- We measure the midpoint of the object in these coordinates, here (0.4,0.3).
- The width (height) of the object is measured as the fraction of the overall width (height) of this grid cell box (frame).

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ \vdots \\ c_K \end{bmatrix} = \begin{bmatrix} 1 \\ 0.4 \\ 0.3 \\ 0.9 \\ 0.8 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



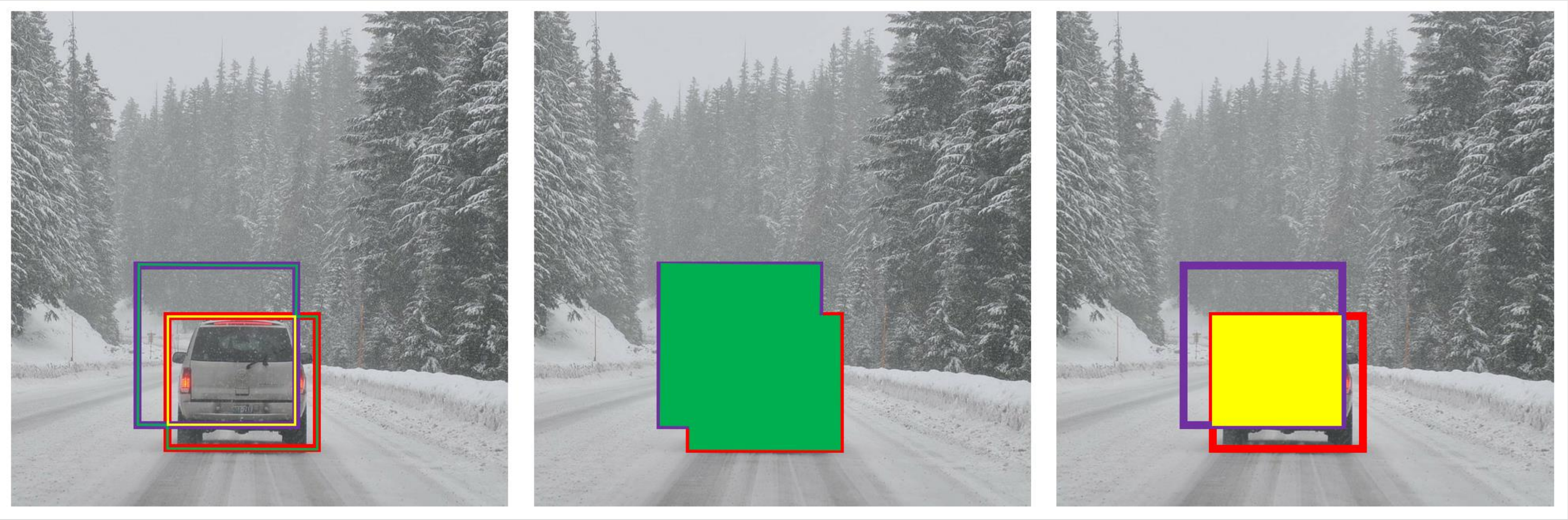
- The midpoints are always between 0 and 1, while widths and heights could be greater than 1.
- If we want to use a sigmoid function (not ReLU) in an output layer and we need to have all widths and heights between 0 and 1, we can divide widths by the number of grid cells in a row ($b_w/S$), and divide heights by the number of grid cells in a column ($b_h/S$).

# Intersection Over Union

## Intersection Over Union (IOU):

- Is used to measure the quality of the estimated bounding box to the ground-truth bounding box defined in the training dataset.
- Is treated as correct if IOU ≥ 0.5 or more dependently on the application.
- Is a measure of the overlap between two bounding boxes.
- Is computed as the ratio of the size of the intersection between two bounding boxes and the union of these bounding boxes:

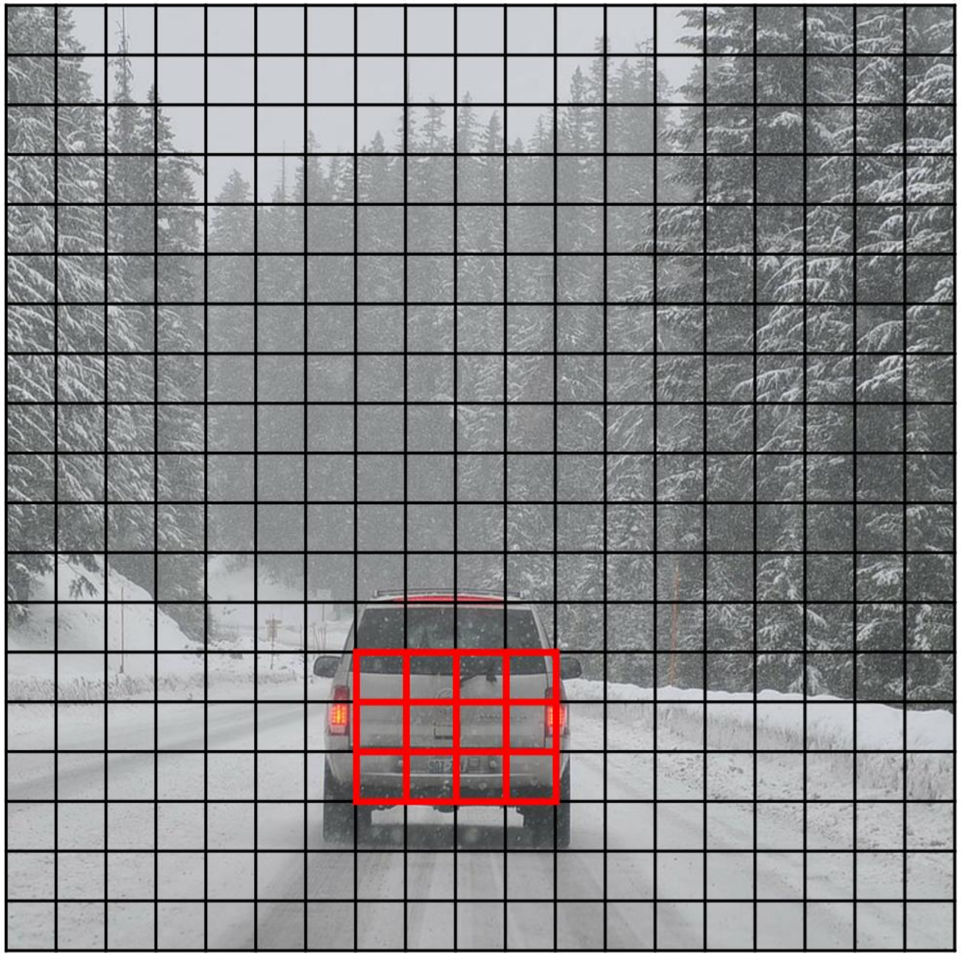
IOU = size of 🟨 / size of 🟩



# Non-Max Suppression of YOLO

**Non-max suppression avoids multiple bounding boxes for the detected objects leaving only one with the highest IOU.**

- **When using bigger grids, many grid cells might think that they represent the midpoint of the detected object.**
- **In result, every such cell will produce a bounding box, so we get multiple bounding boxes for the same object.**
- **YOLO chooses the one with the highest probability $p_c$ computed for each grid cell.**



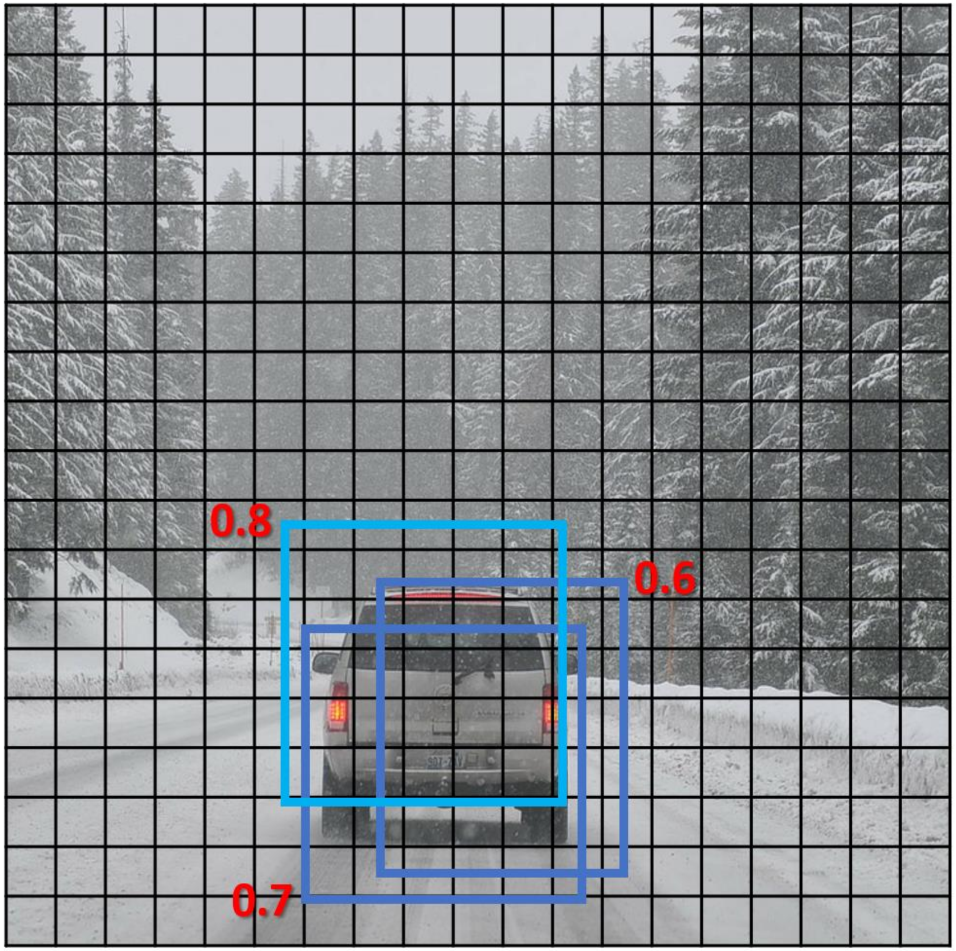
Grid 19x19

# Non-Max Suppression of YOLO

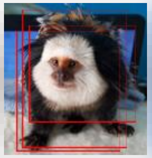
## Non-Max Suppression works as follows:

1. Discard all bounding boxes estimated by the convolutional network which probability is $p_c \leq 0.6$.
2. While there are any remaining bounding boxes:
   1. Pick this one with the largest $p_c$, and output that as a prediction of the detected object. (selection step)
   2. Discard any remaining bounding box with IOU ≥ 0.5 with the box output in the previous step. (pruning/suppression step)

For multiple object detection of the different classes, we perform the non-max suppression for each of these classes independently.



Grid 19x19

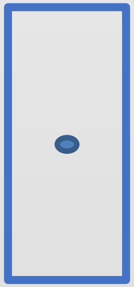


# Anchor Boxes for Multiple Object Detection

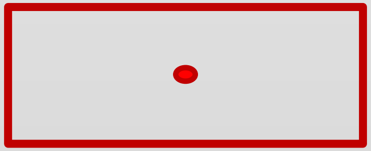
When two or more objects are in almost the same place in the image and their midpoints of their ground-truth bounding boxes fall into the same grid cell, we cannot use the previous algorithm but define a few anchor boxes with the predefined shapes associated with different classes of objects that can occur in the same grid cell:

## Example:

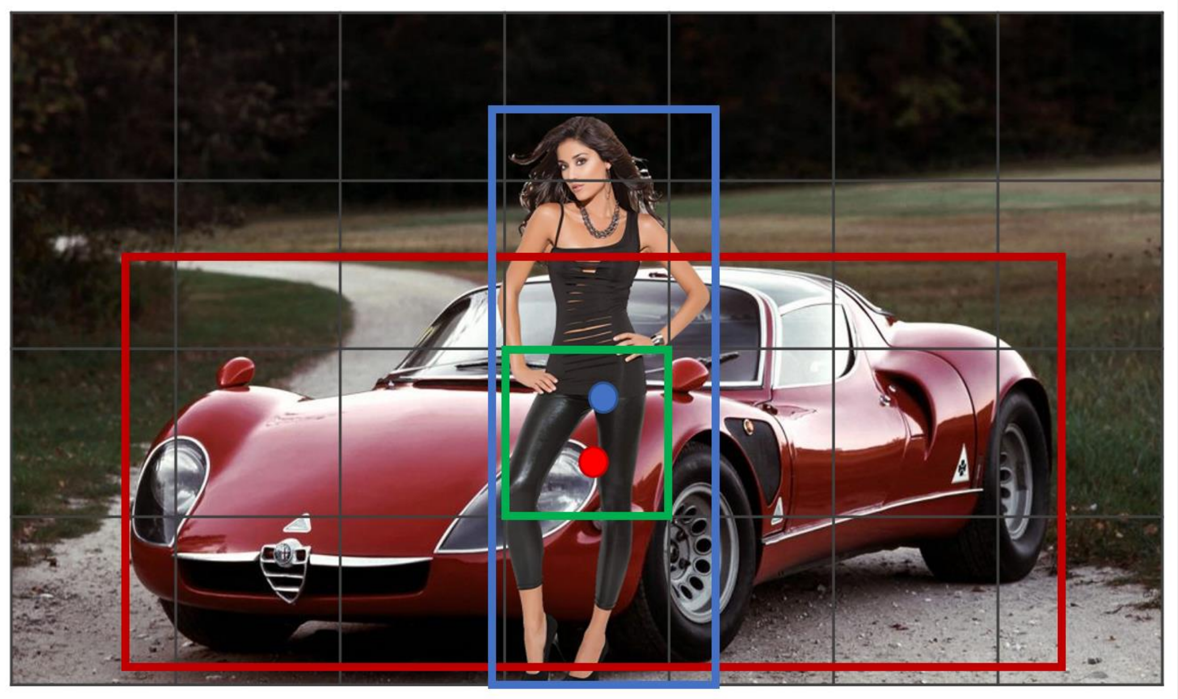
Anchor box 1 (A1):



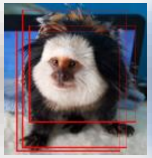
Anchor box 2 (A2):



$$y = \begin{bmatrix} p_c^{A1} \\ b_x^{A1} \\ b_y^{A1} \\ b_h^{A1} \\ b_w^{A1} \\ c_1^{A1} \\ c_2^{A1} \\ \vdots \\ c_K^{A1} \\ p_c^{A2} \\ b_x^{A2} \\ b_y^{A2} \\ b_h^{A2} \\ b_w^{A2} \\ c_1^{A2} \\ c_2^{A2} \\ \vdots \\ c_K^{A2} \end{bmatrix}$$



The YOLO algorithm with anchor boxes assigns each object in training image to the grid cell that contains the object's midpoint and the appropriate anchor box for the grid cell with the highest IOU.

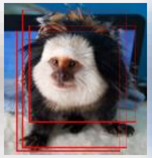


# Anchor Boxes and Target Setup

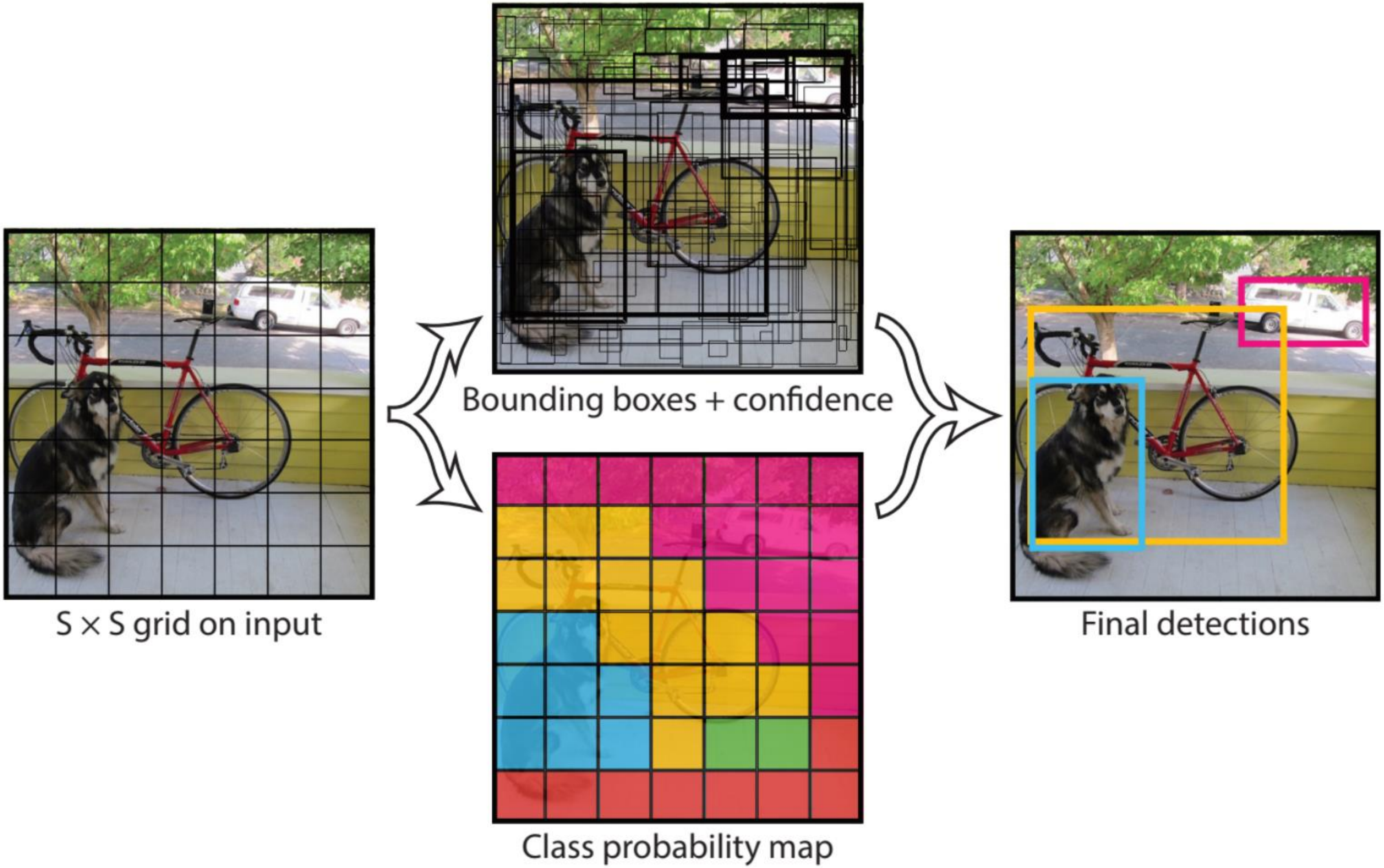
## For two anchor boxes in the grid cell, we consider four cases:

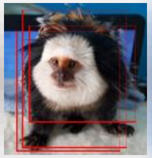
1. There are no midpoints of objects in the cell.
2. There is one midpoint of the object of the anchor 1 and class $c_1$ in the cell.
3. There is one midpoint of the object of the anchor 2 and class $c_2$ in the cell.
4. There is two midpoints of two object of the anchor 1 and the anchor 2 and both classes $c_1$ and $c_2$ in the cell.

$$y = \begin{bmatrix} p_c^{A1} \\ b_x^{A1} \\ b_y^{A1} \\ b_h^{A1} \\ b_w^{A1} \\ c_1^{A1} \\ c_2^{A1} \\ \vdots \\ c_K^{A1} \\ p_c^{A2} \\ b_x^{A2} \\ b_y^{A2} \\ b_h^{A2} \\ b_w^{A2} \\ c_1^{A2} \\ c_2^{A2} \\ \vdots \\ c_K^{A2} \end{bmatrix} \qquad \text{(1)} \quad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ \vdots \\ ? \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ \vdots \\ ? \end{bmatrix} \qquad \text{(2)} \quad y = \begin{bmatrix} 1 \\ b_x^{A1} \\ b_y^{A1} \\ b_h^{A1} \\ b_w^{A1} \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ \vdots \\ ? \end{bmatrix} \qquad \text{(3)} \quad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ \vdots \\ ? \\ 1 \\ b_x^{A2} \\ b_y^{A2} \\ b_h^{A2} \\ b_w^{A2} \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \qquad \text{(4)} \quad y = \begin{bmatrix} 1 \\ b_x^{A1} \\ b_y^{A1} \\ b_h^{A1} \\ b_w^{A1} \\ 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ b_x^{A2} \\ b_y^{A2} \\ b_h^{A2} \\ b_w^{A2} \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$



# YOLO Detection Model



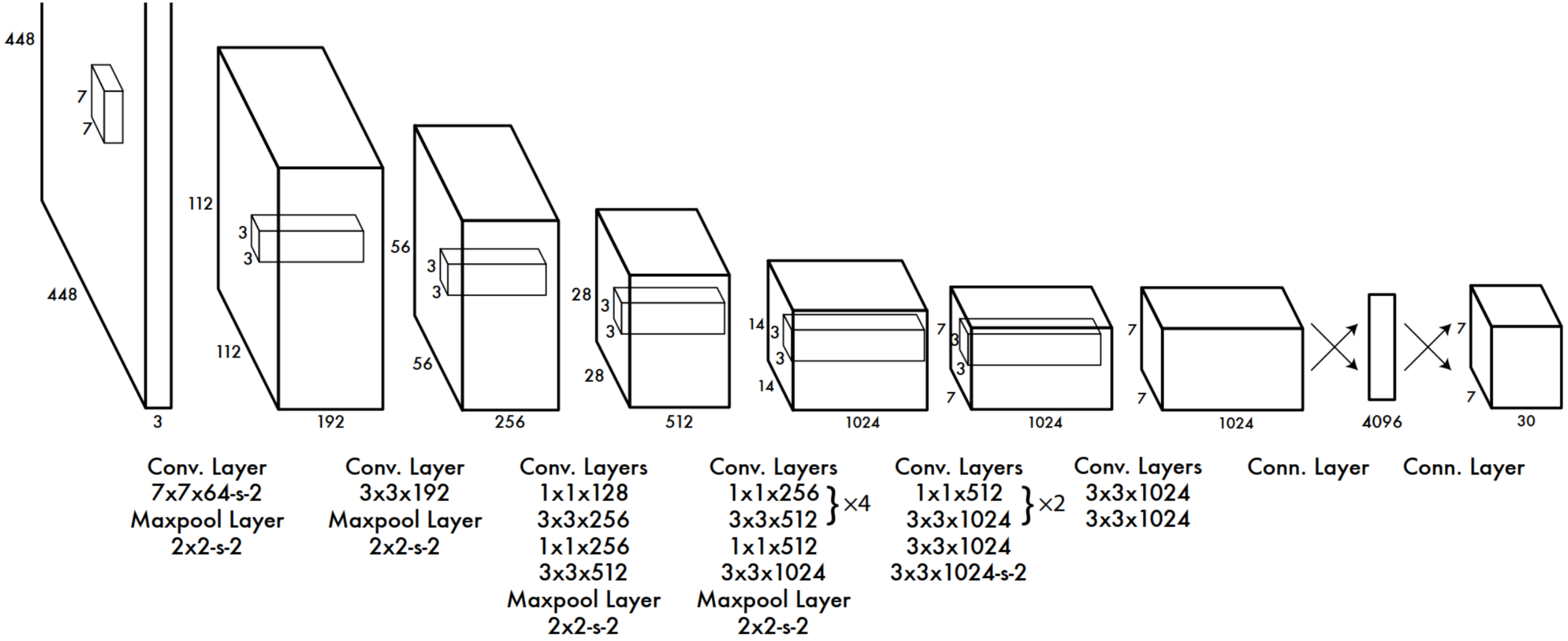


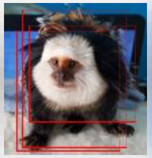
# Classic YOLO Network Architecture

**YOLO network architecture is convolutional with the output defined as a 3D matrix of the S x S x (A x 8) sizes:**

- **S – is the number or cells in each row and column**
- **A – is the number of anchors**

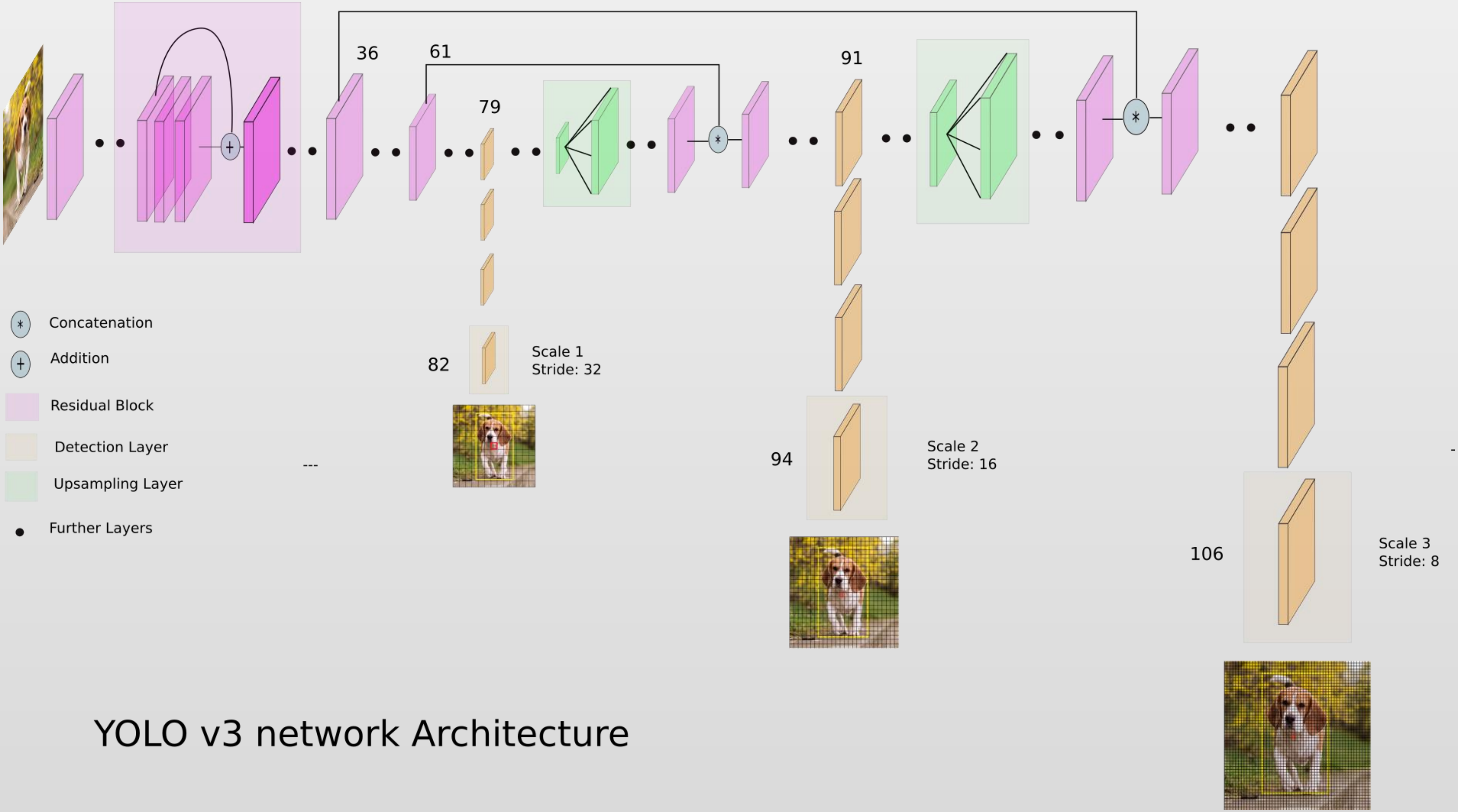
**However, we can modify the original YOLO model in such a way that the number of cells in rows and columns differ.**



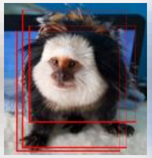


# YOLOv3 Network Architecture

**YOLOv3 network uses extra operations (concatenation and addition) as well as residual blocks, detection and upsampling layers.**



YOLO v3 network Architecture



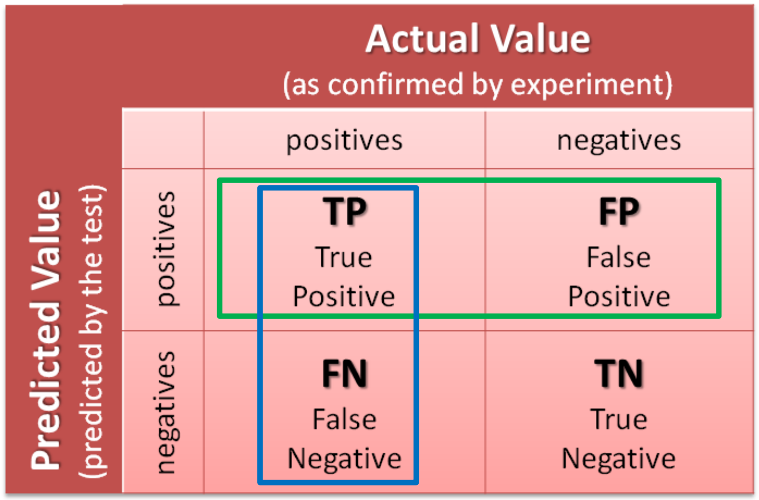
# Precision and Recall

## Confusion Matrix

- Specifies how many examples were correctly classified as positive (TP), negative (TN) and how many were misclassified as positive (FP) or negative (FN).

## Precision

- measures how accurate is your predictions. i.e. the percentage of your predictions are correct.



| | | Actual Value<br>(as confirmed by experiment) | |
|---|---|---|---|
| | | positives | negatives |
| Predicted Value<br>(predicted by the test) | positives | TP<br>True Positive | FP<br>False Positive |
| | negatives | FN<br>False Negative | TN<br>True Negative |

$$\textit{Precision} = \frac{T}{TP}$$

## Recall

- measures how good you find all the positives. For example, we can find 80% of the possible positive cases in our top K predictions.

$$\textit{Recall} = \frac{TP}{TP + FN}$$

# Mean Average Precision

## Average Precision (AP):

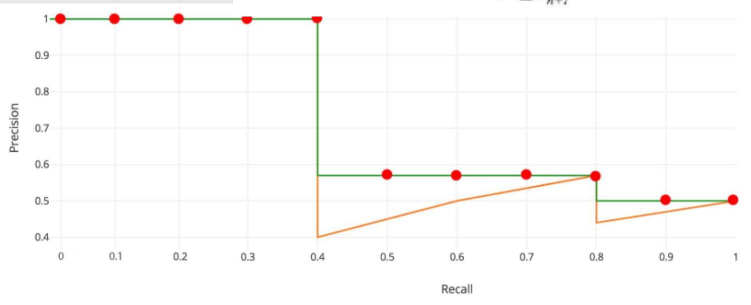
- Is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, YOLO, etc. Average precision computes the average precision value for recall value over 0 to 1.

$$AP = \int_0^1 p(r)\,dr$$

$$\mathrm{AP} = \Sigma\,(\,r_{n+1} - r_n)\,p_{interp}(r_{n+1})$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1}} p(\tilde{r})$$

- where $p(r)$ is a precision-recall curve.



## Mean Average Precision (mAP):

- Is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, YOLO, etc. Average precision computes the average precision value for recall value over 0 to 1.

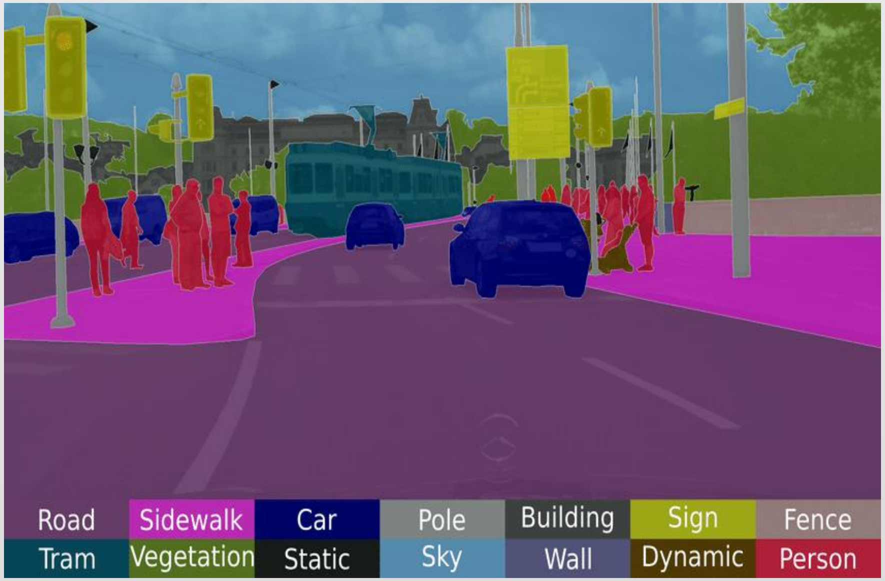
# R-CNN, Fast R-CNN, and Faster R-CNN

## R-CNN stands for Regions with ConvNet detection:

- Is a segmentation algorithm.
- The algorithm is run on a big number of block to classify them
- R-CNN proposes regions at a time.
- We get an output label + bounding box

## Fast R-CNN:

- A convolutional implementation of sliding windows to classify all the proposed regions.



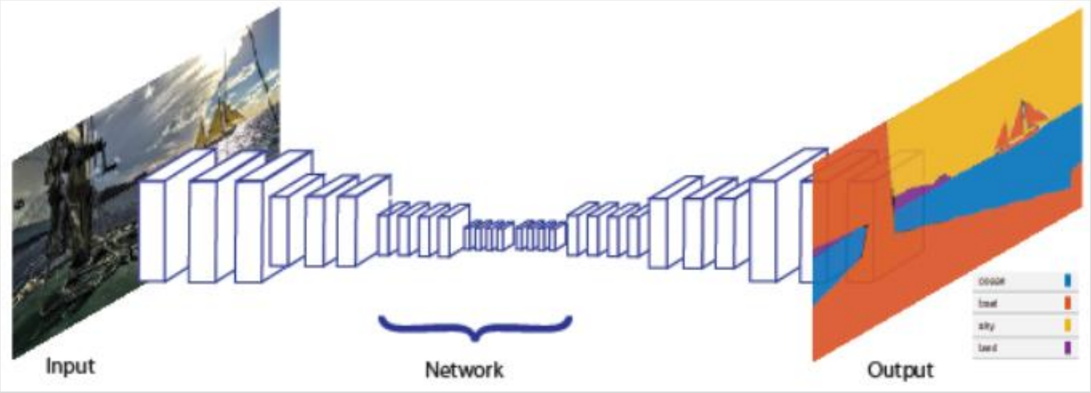
## Faster R-CNN:

- Uses a convolutional network to propose regions.

# Semantic Segmentation Using Deep Learning

## Semantic Segmentation:

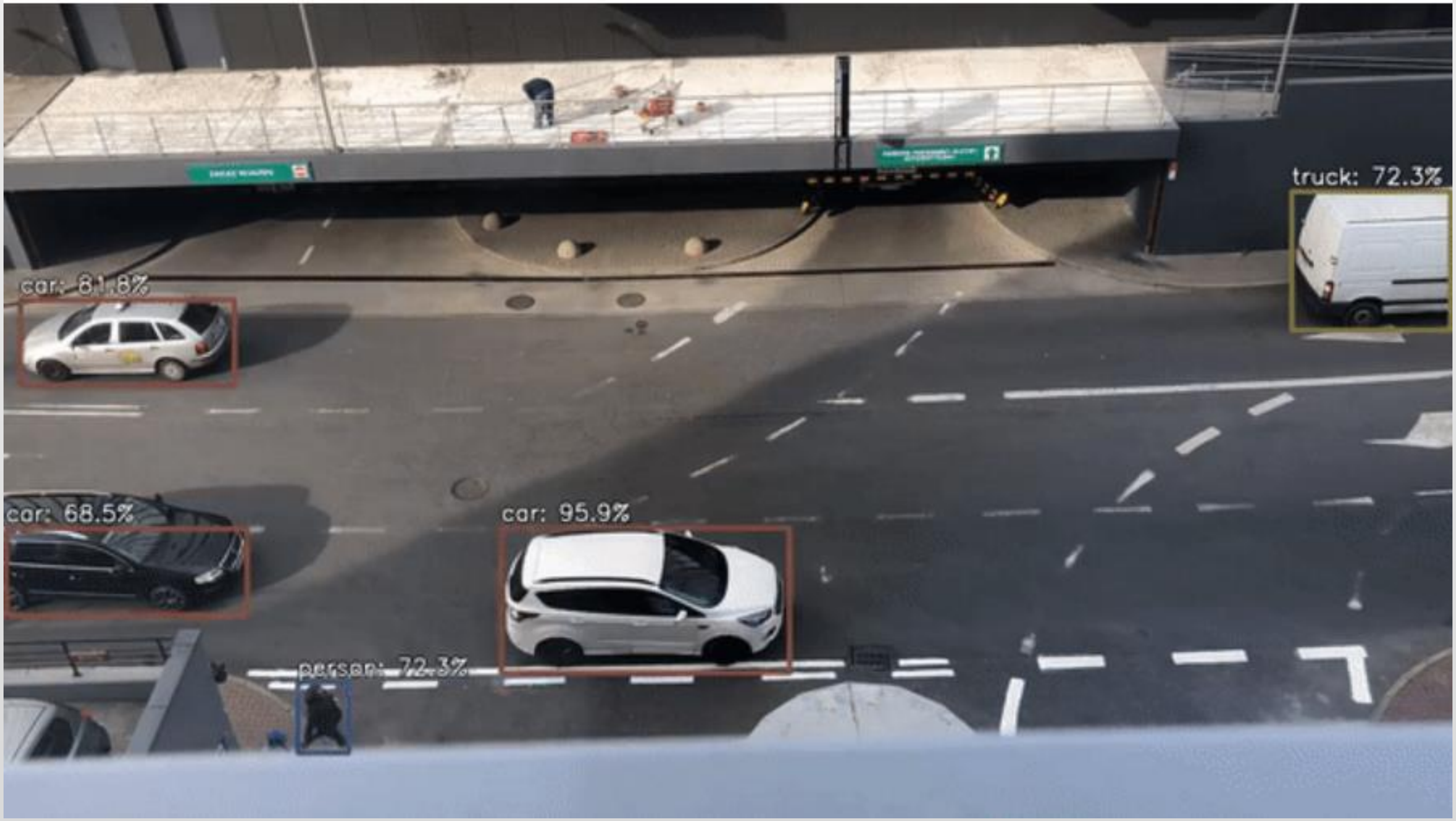
- **Xxxx**
- https://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf
- https://www.mathworks.com/help/vision/ug/getting-started-with-semantic-segmentation-using-deep-learning.html
- https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef

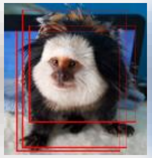


# RetinaNet

**RetinaNet:**

- can have ~100k boxes with the resolve of class imbalance problem using focal loss.
- Many one-stage detectors do not achieve good enough performance, so there are build new two-stage detectors like RetinaNet:

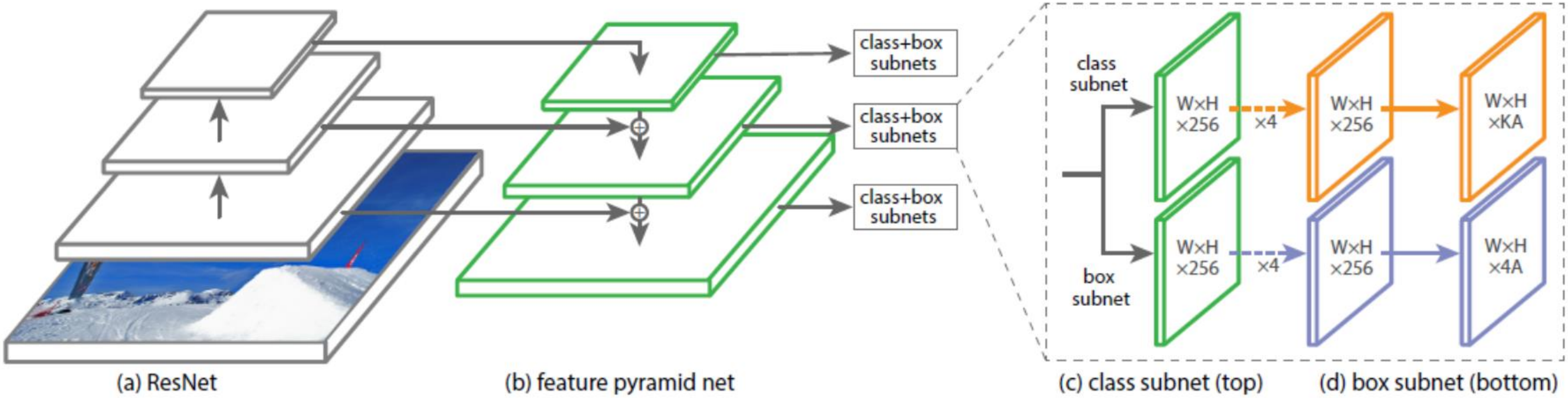




# RetinaNet

## RetinaNet:

- In RetinaNet, an one-stage detector, by using focal loss, lower loss is contributed by “easy” negative samples so that the loss is focusing on “hard” samples, which improves the prediction accuracy. With ResNet+FPN as backbone for feature extraction, plus two task-specific subnetworks for classification and bounding box regression, forming the RetinaNet, which achieves state-of-the-art performance, outperforms Faster R-CNN, the well-known two-stage detectors. It is a 2017 ICCV Best Student Paper Award paper with more than 500 citations. (The first author, Tsung-Yi Lin, has become Research Scientist at Google Brain when he was presenting RetinaNet in 2017 ICCV.) (Sik-Ho Tsang @ Medium).
- https://www.youtube.com/watch?v=44tlnmmt3h0



RetinaNet Detector Architecture

# Let’s start with powerful computations!

- ✓ Questions?
- ✓ Remarks?
- ✓ Suggestions?
- ✓ Wishes?

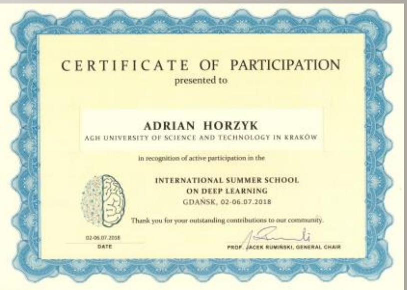
# Bibliography and Literature

1. https://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf
2. https://www.mathworks.com/help/vision/ug/getting-started-with-semantic-segmentation-using-deep-learning.html
3. https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef
4. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173
5. https://pjreddie.com/darknet/yolo/
6. https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/
7. https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-2/
8. https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-3/
9. https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-4/
10. https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-5/
11. https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg
12. https://arxiv.org/pdf/1708.02002.pdf
13. https://www.youtube.com/watch?v=44tlnmmt3h0
14. https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4

**Adrian Horzyk**

**horzyk@agh.edu.pl**

**Google: Horzyk**



CERTIFICATE OF PARTICIPATION
presented to

ADRIAN HORZYK
AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY IN KRAKÓW

in recognition of active participation in the

INTERNATIONAL SUMMER SCHOOL
ON DEEP LEARNING
GDAŃSK, 02-06.07.2018

Thank you for your outstanding contributions to our community.

02-06.07.2018
DATE

PROF. JACEK RUMIŃSKI, GENERAL CHAIR

AGH

**University of Science
and Technology
in Krakow, Poland**