



# Knowledge-based CI and DM in Biomedicine

Object and Key Point Detection, Localization,  
Classification, and Segmentation



**Adrian Horzyk**  
[horzyk@agh.edu.pl](mailto:horzyk@agh.edu.pl)



# Object and Key Point Detection and Classification and Semantic and Instance Segmentation

What can we detect or segment in images?

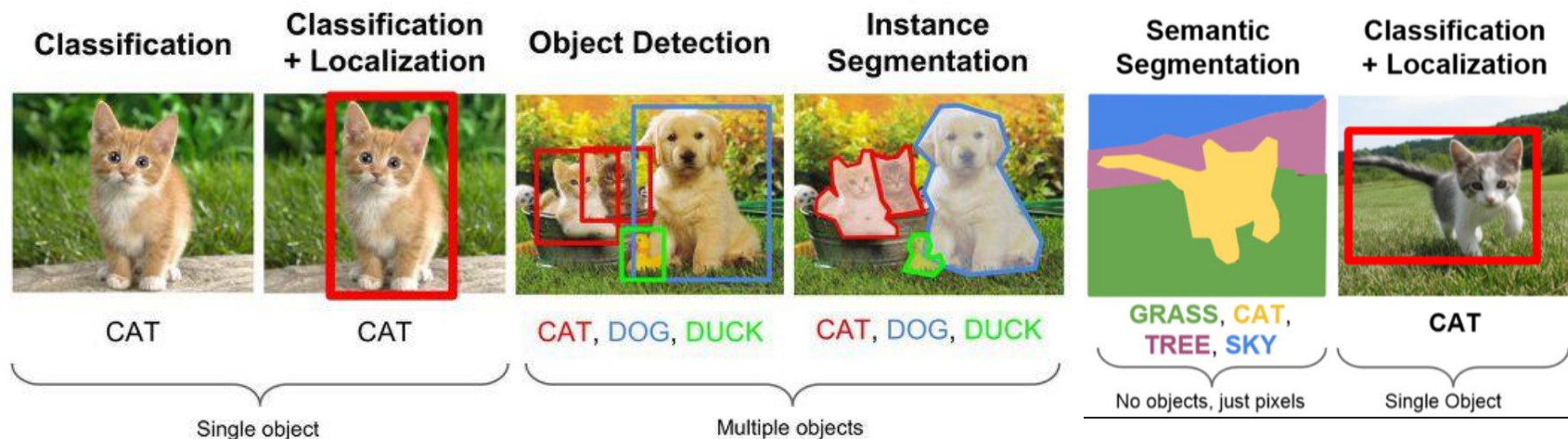
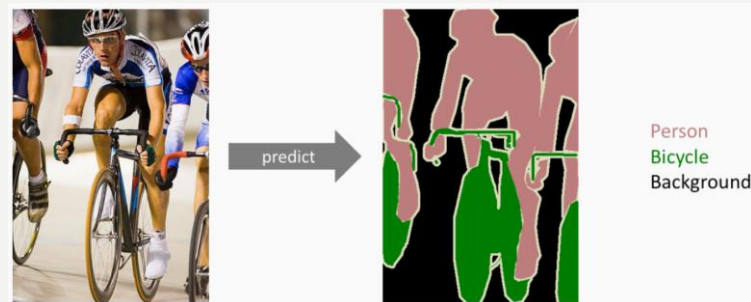


# Classify, Detect, Localize, and Segment



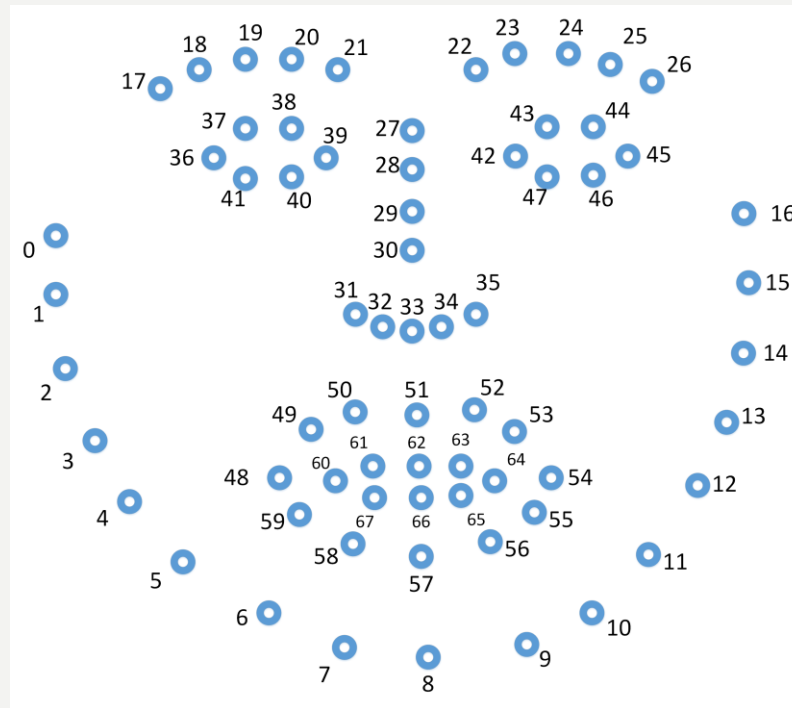
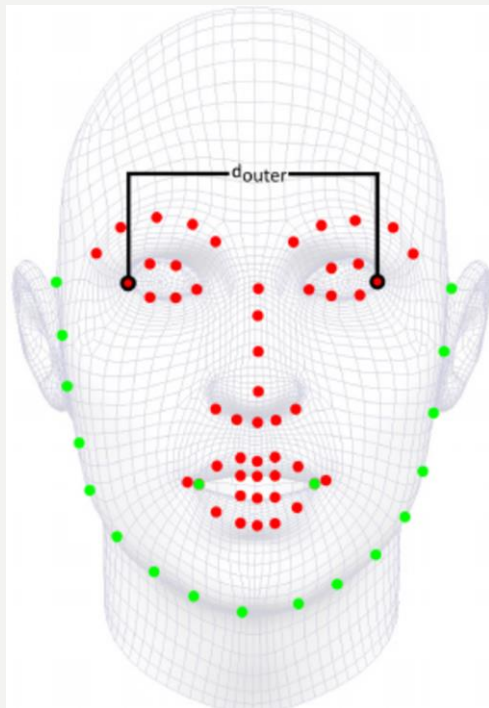
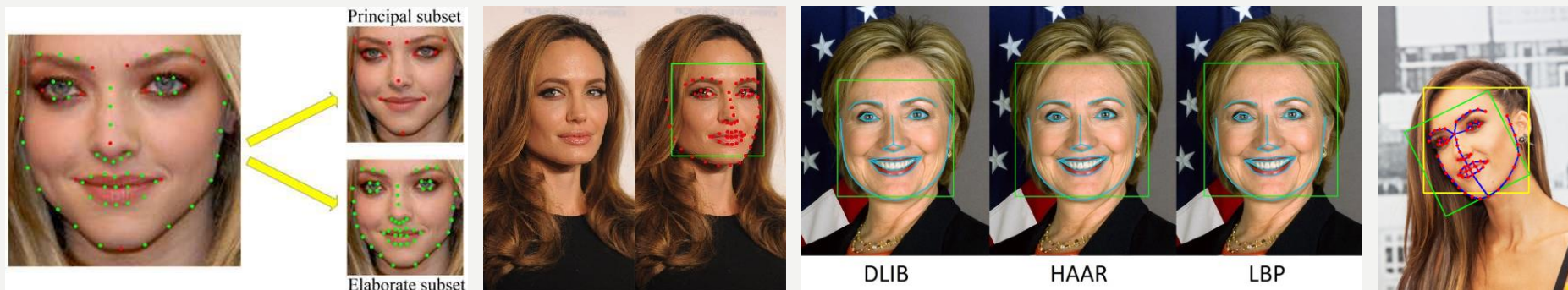
Ordinary and popular tasks performed on images:

- Object Classification
- Object Classification with Localization (using bounding boxes)
- Object Detection
- Object Key Point (Landmark) Detection
- Object Instance Segmentation
- Object Semantic Segmentation
- Scene parsing and understanding



# Landmark (Key Points) Detection

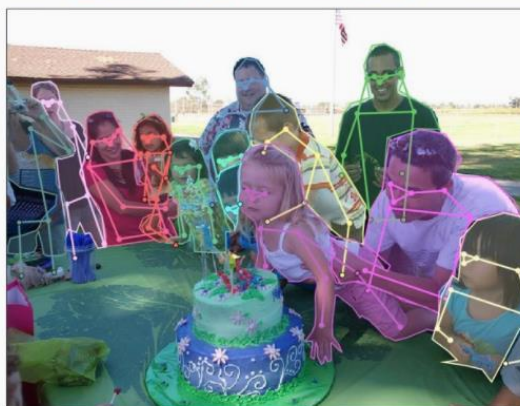
We can detect various landmarks (key points) in images and use them to model and recognize facial gesture, emotion expressions, body poses etc.:





# Landmark (Key-Points) Detection

Key point detection is crucial from the semantic point of view to interpret the states and actions that are visible in the images or movies:



Keypoints annotations along with visualized edges between keypoints. Images are from the [COCO dataset](#).



# Definitions

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

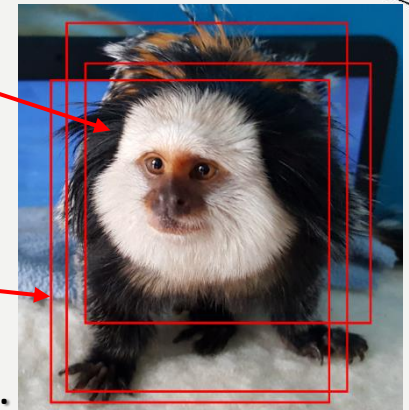
**Classification with localization** not only classifies the main object in the image but also localizes it in the image determining its bounding box (position and size or localization anchors).

**Detection** is to find all object of the previously trained (known) classes in the image and localize them (detect their position and size).

**Semantic Segmentation** is to label specific regions of an image according in the pixel level to understand relationships between objects or recognize important objects in the context (location) of the other objects or their states, actions, and dependencies.

**Instance Segmentation** is the process of dividing an image into parts known as areas that are homogeneous with respect to certain selected properties, where these areas are collections of pixels. We do not only label these areas with class labels but separate individual instances of the same class. Properties that are often selected as criteria for the uniformity of areas are: gray level, color, texture.

monkey



0: Background/Unknown  
1: Person  
2: Purse  
3: Plants/Grass  
4: Sidewalk  
5: Building/Structures

Ground Truth

Image







# Object and Key Point Detection Localization, and Classification

How to detect, localize, and classify objects?

# 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
- $b_h$  – the height of the bounding box of the object
- $b_w$  – the width of the bounding box of the object



# 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:

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

where

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

$p_c$  – 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

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

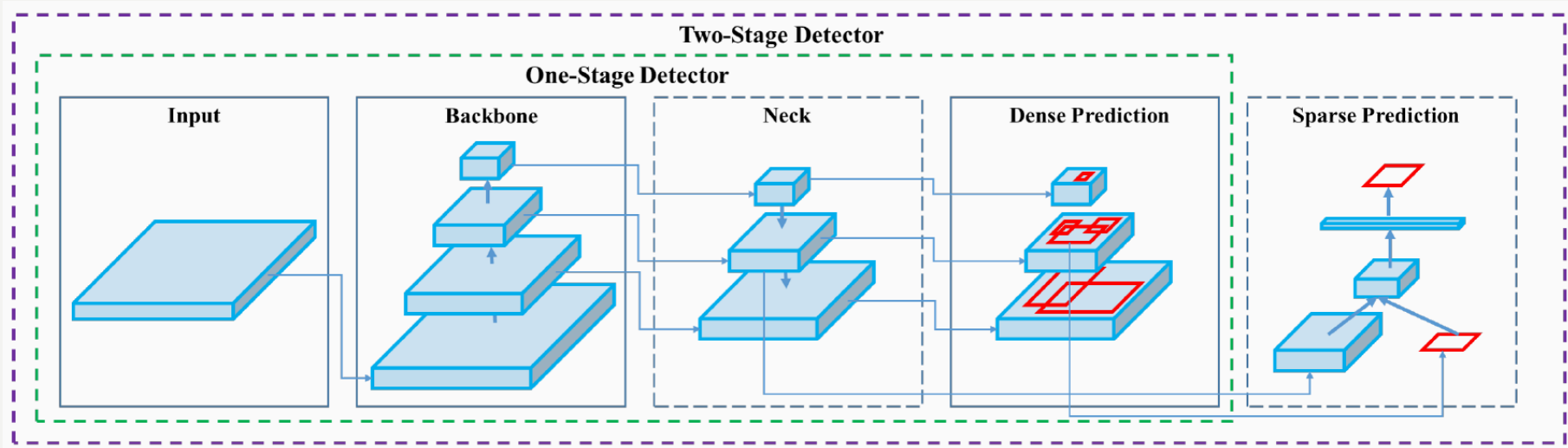
$b_w$  – the width of the bounding box of the object

$c_1, c_2, \dots, c_K$  – 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

# How Do Detectors Work?

Ordinary object detectors are typically composed of several parts:



**Input:** Image, Patches, Image Pyramid

**Backbone:** VGG16, ResNet-50, SpineNet, EfficientNet-B0/B7, CSPResNeXt50, CSPDarknet53

**Neck:** Additional blocks: SPP, ASPP, RFB, SAM

Path-aggregation blocks: FPN, PAN, NAS-FPN, Fully-connected FPN, BiFPN, ASFF, SFAM

**Heads:** Dense Prediction (one-stage):

Anchor-based: RPN, SSD, YOLO, RetinaNet

Anchor-free: CornerNet, CenterNet, MatrixNet, FCOS

Sparse Prediction (two-stage):

Anchor-based: Faster R-CNN, R-FCN, Mask RCNN

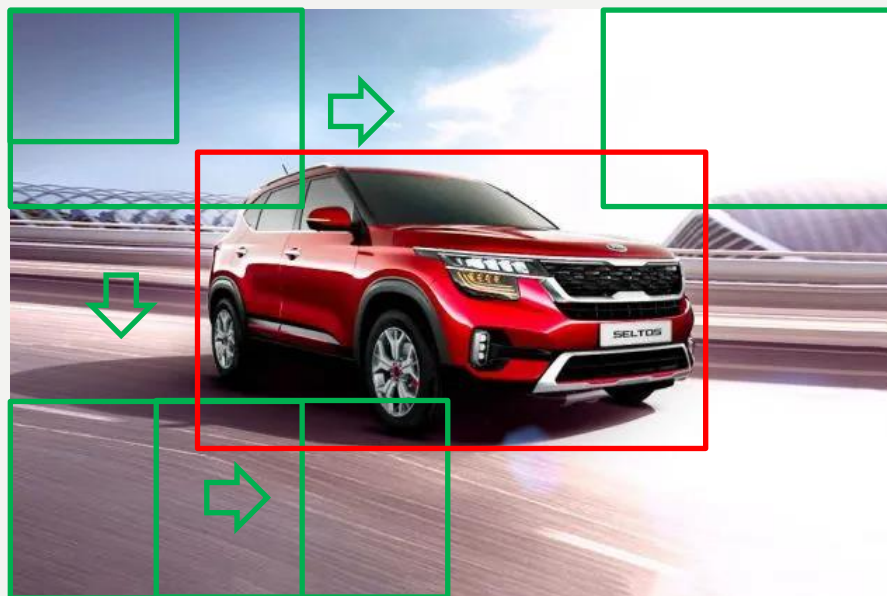
Anchor-free: RepPoints



# Object Detection and Cropping Out

Object detection can be made in a few ways:

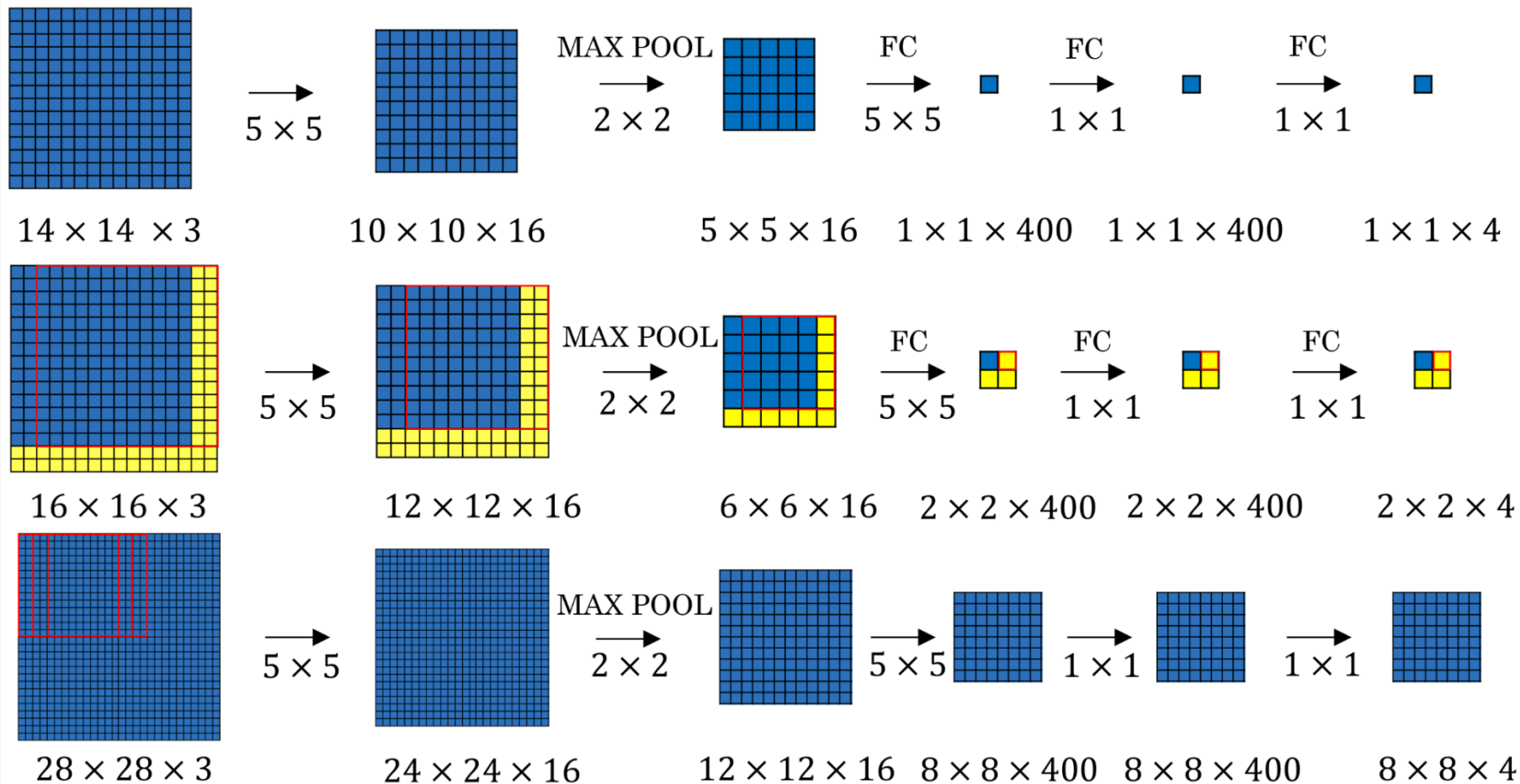
- using **sliding window** of the same size or various sizes with different strides (high computational cost because of many strides) – **sliding window detection**
- using a grid (mesh) of fixed windows (e.g. YOLO – you only look once)
- and put the cropped image on the input of the ConvNet:



# 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.

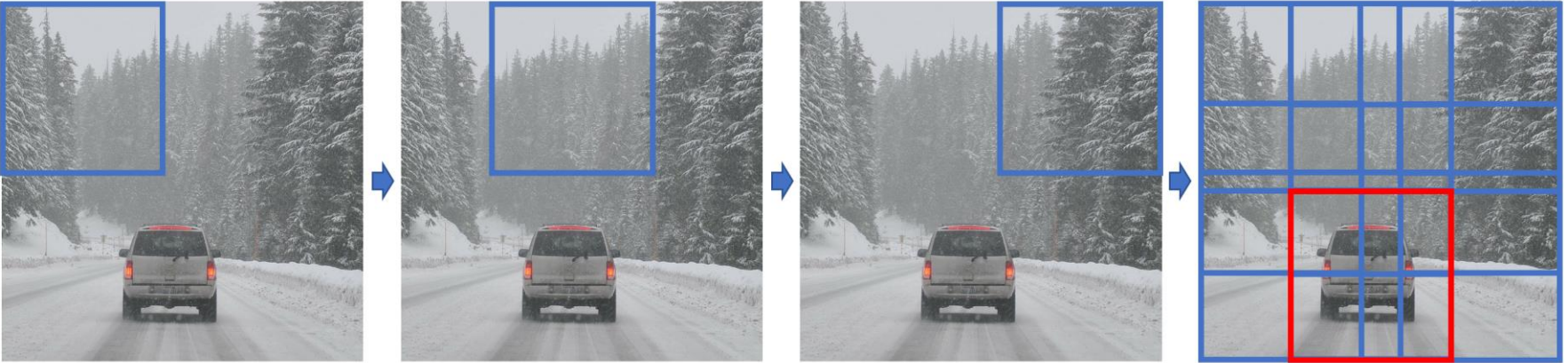
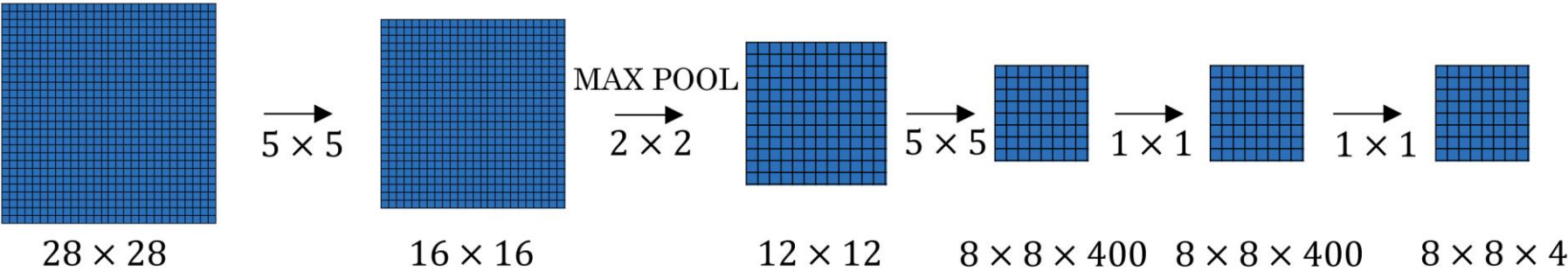


Therefore, we implement sliding windows parallelly and share these computations that are the same for different sliding windows to proceed computations faster.



# Convolutional Implementation of Sliding Windows

How the convolutional implementation of the sliding window works on the image?



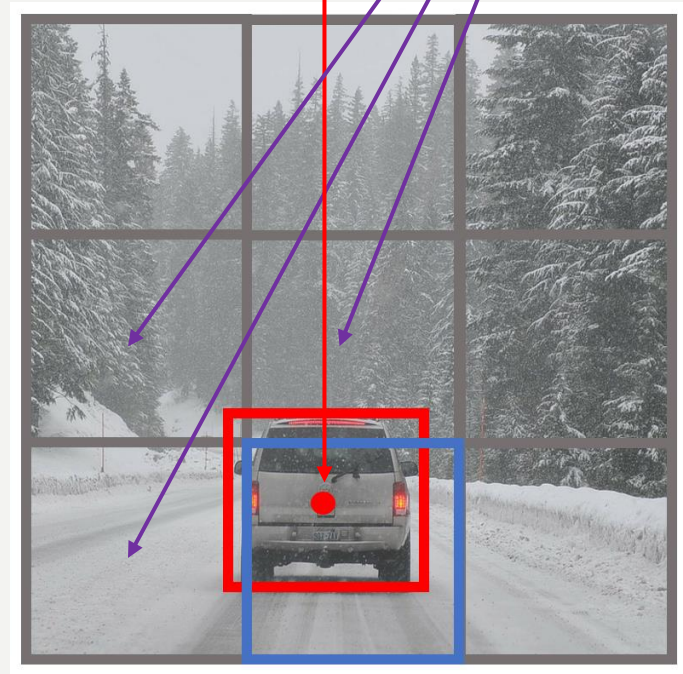
The drawback is the position of the bounding box designated by the sliding window that might not be very accurate. Moreover, if we want to fit each object better, we have to use many such parallel convolutional networks for various sizes of sliding windows. Even though we cannot use appropriately adjusted 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:
- The target (trained) output is a 3D matrix of  $S \times S \times (K+5)$  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.  $19 \times 19$ , so there is a less chance to have more than one middle point of the object inside each grid cell.

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



# YOLO's bounding boxes

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

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

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

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

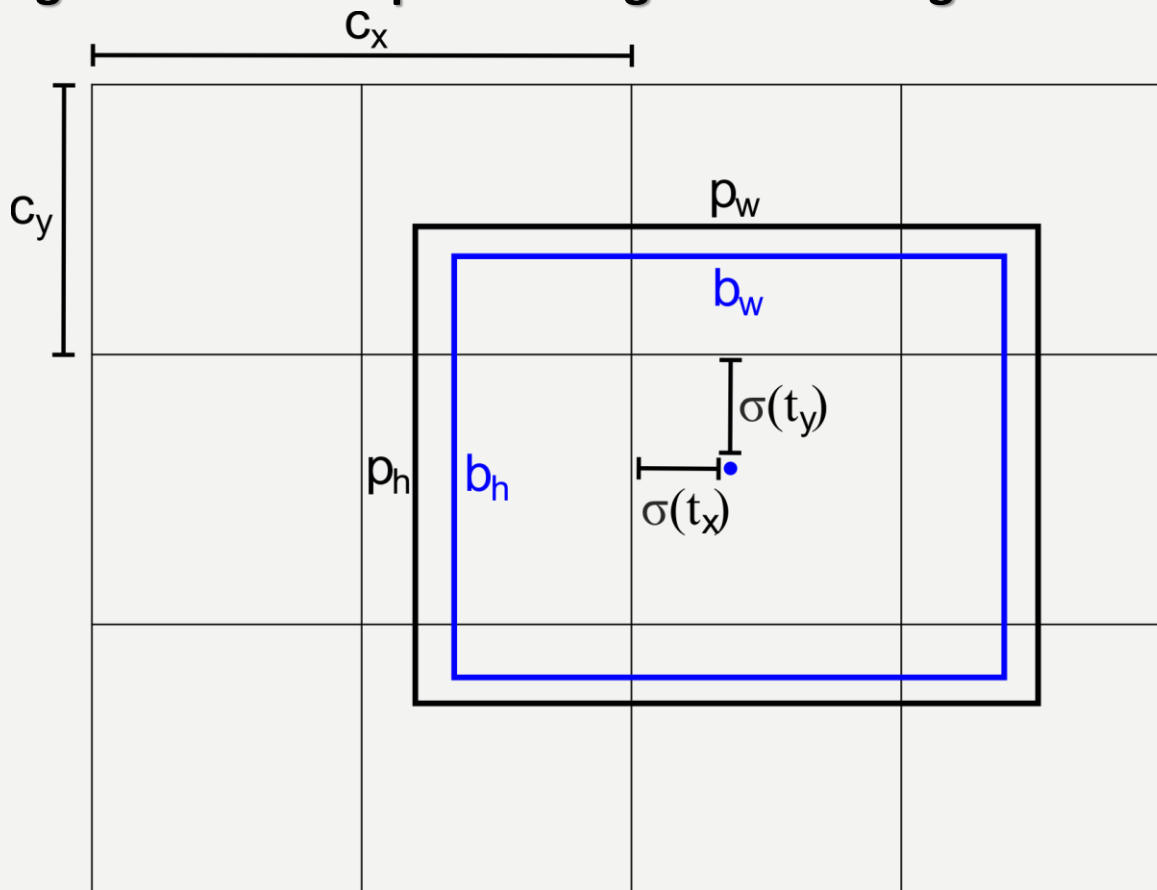
$$b_w = p_w \cdot e^{t_w}$$

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

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).



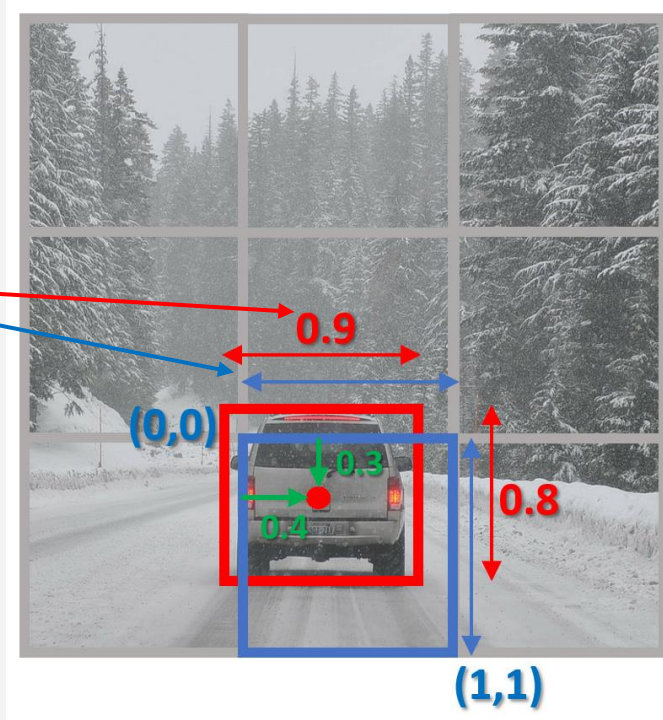


# Specifying the Bounding Boxes in YOLO

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)**.

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

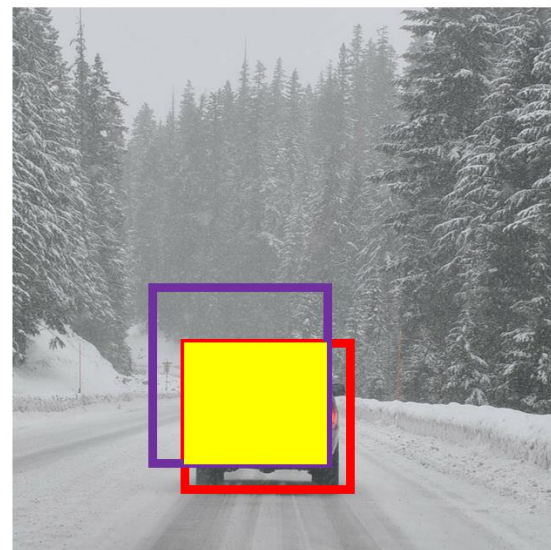
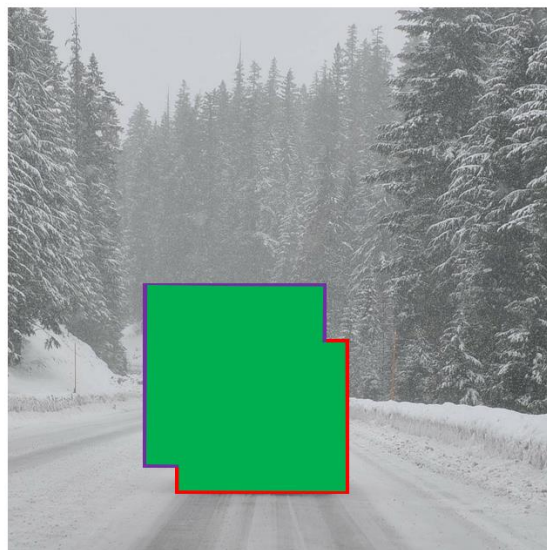
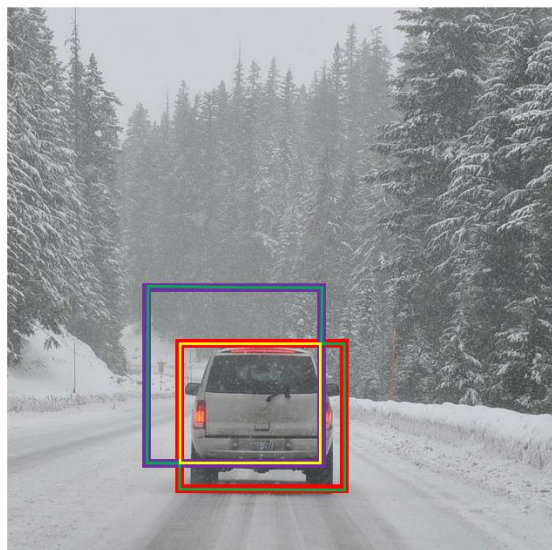


- The midpoints are always between 0 and 1, while widths and heights could be greater than 1.
- If we want to use a sigmoid function (not ReLU) in an output layer and we need to have all widths and heights between 0 and 1, we can divide widths by the number of grid cells in a row ( $b_w/S$ ), and divide 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  $\text{IOU} \geq 0.5$  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:

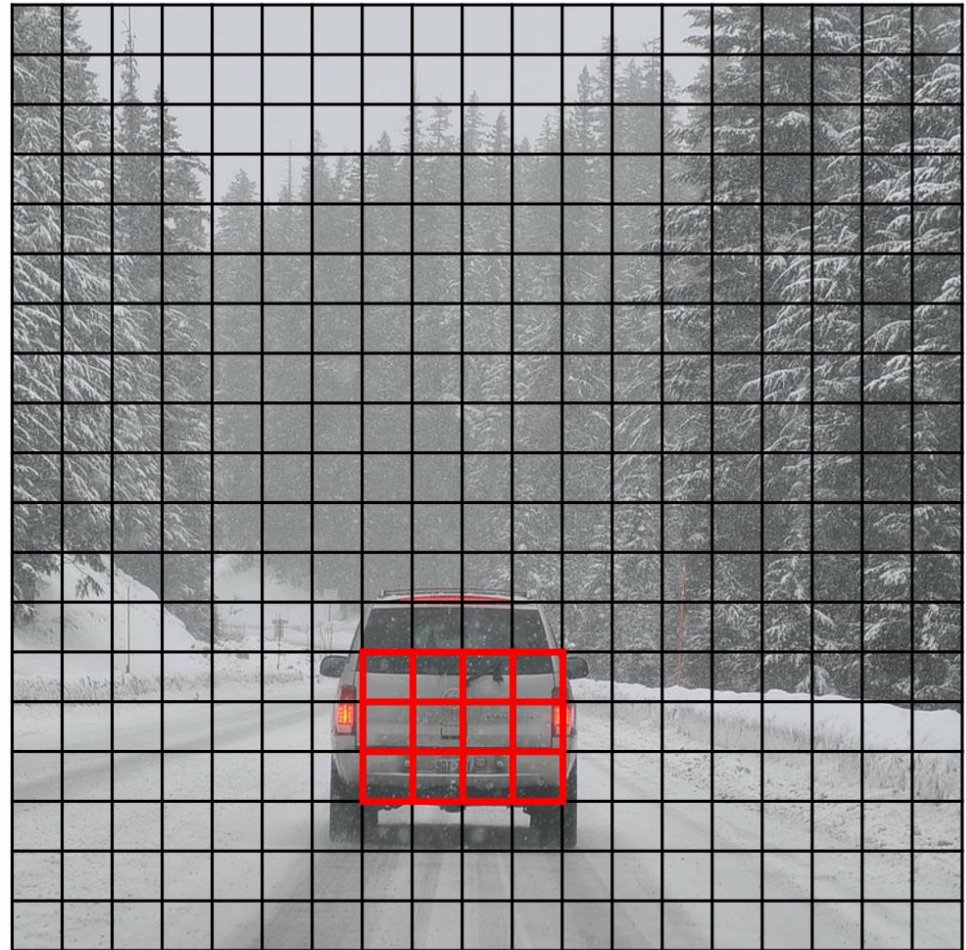
$$\text{IOU} = \frac{\text{size of } \text{[yellow box]}}{\text{size of } \text{[green box]}}$$



# 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, but they will be reduced using Non-Max Suppression.
- YOLO chooses the one with the highest probability  $p_c$  computed for each grid cell.



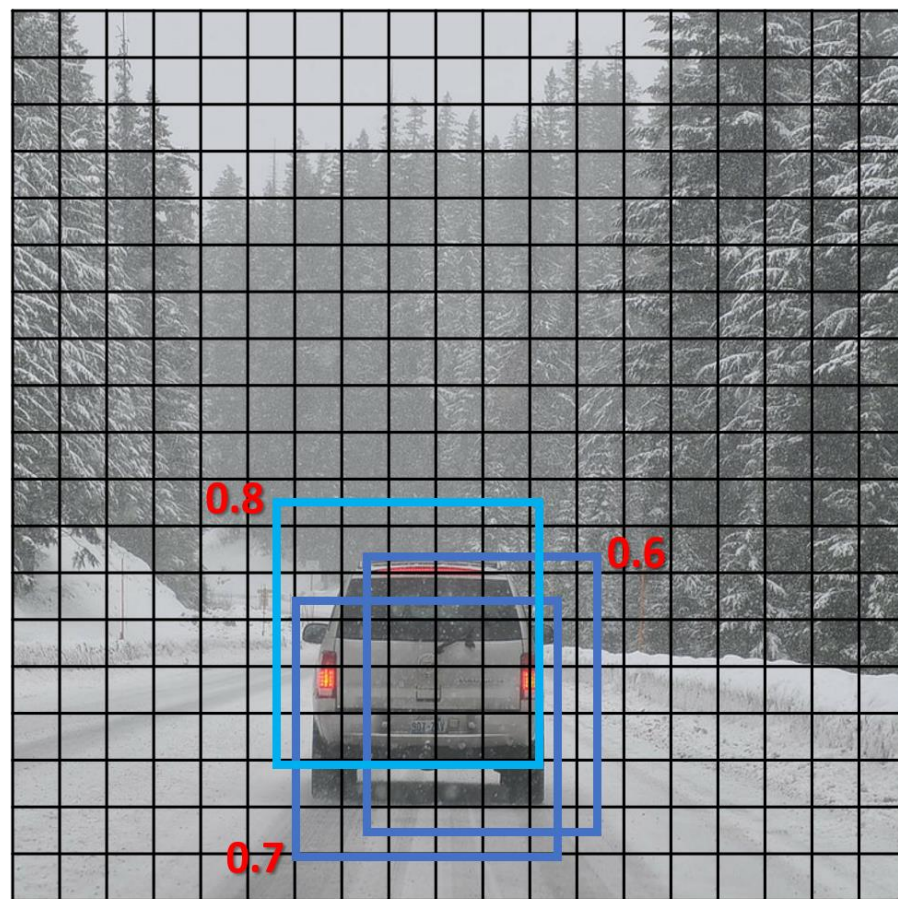
Grid 19x19

# 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  $\text{IOU} \geq 0.5$  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

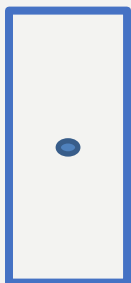


# Anchor Boxes for Multiple Object Detection

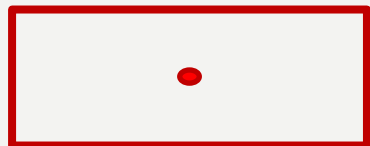
When two or more objects are in almost the same place in the image and their **midpoints** of their ground-truth bounding boxes fall **into the same grid cell**, we cannot use the previous algorithm but define a few **anchor boxes** with the predefined shapes associated with different classes of objects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

Example:

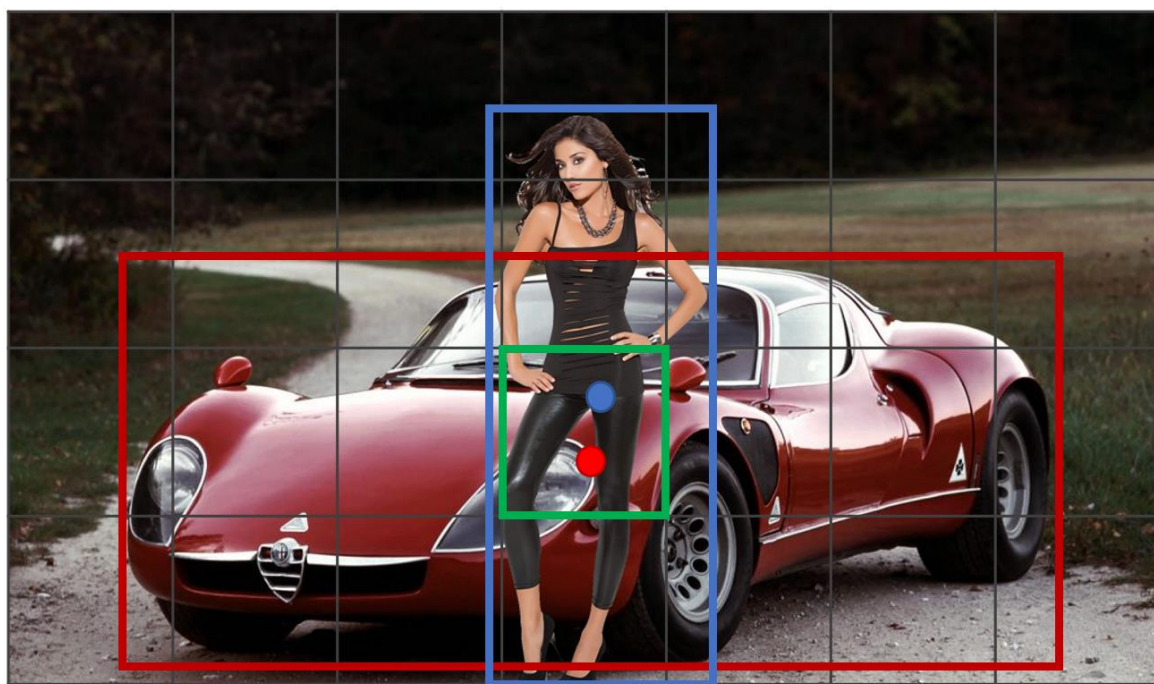
Anchor box 1 (A1):



Anchor box 2 (A2):



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



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.

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

$$(1) \quad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 0 \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

$$(2) \quad y = \begin{bmatrix} 1 \\ b_x^{A1} \\ b_y^{A1} \\ b_h^{A1} \\ b_w^{A1} \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

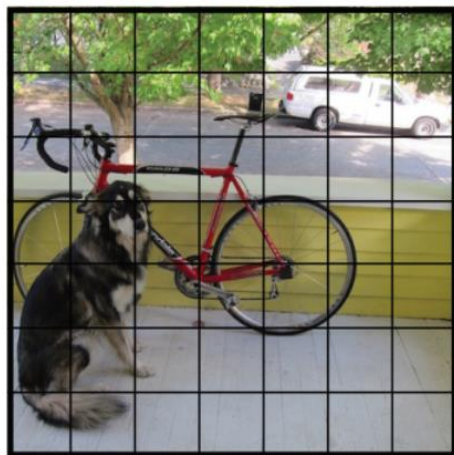
$$(3) \quad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ \vdots \\ ? \\ 1 \\ b_x^{A2} \\ b_y^{A2} \\ b_h^{A2} \\ b_w^{A2} \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$(4) \quad y = \begin{bmatrix} 1 \\ b_x^{A1} \\ b_y^{A1} \\ b_h^{A1} \\ b_w^{A1} \\ 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ b_x^{A2} \\ b_y^{A2} \\ b_h^{A2} \\ b_w^{A2} \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

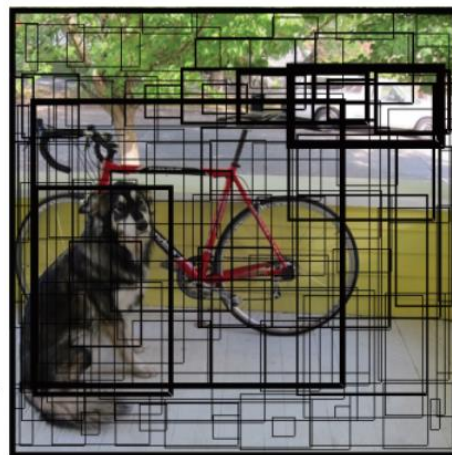


# YOLO Detection Model

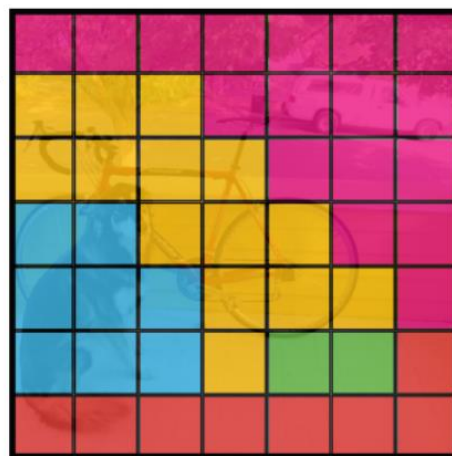
How does it work?



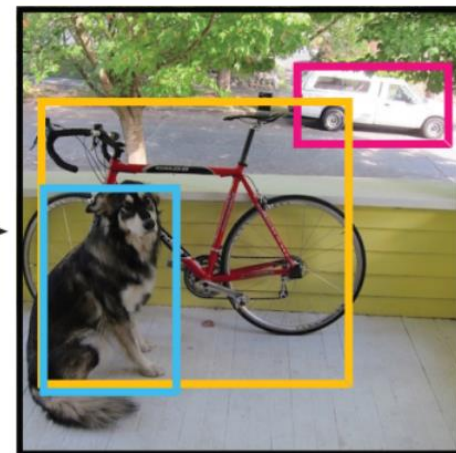
$5 \times 5$  grid on input



Bounding boxes + confidence



Class probability map



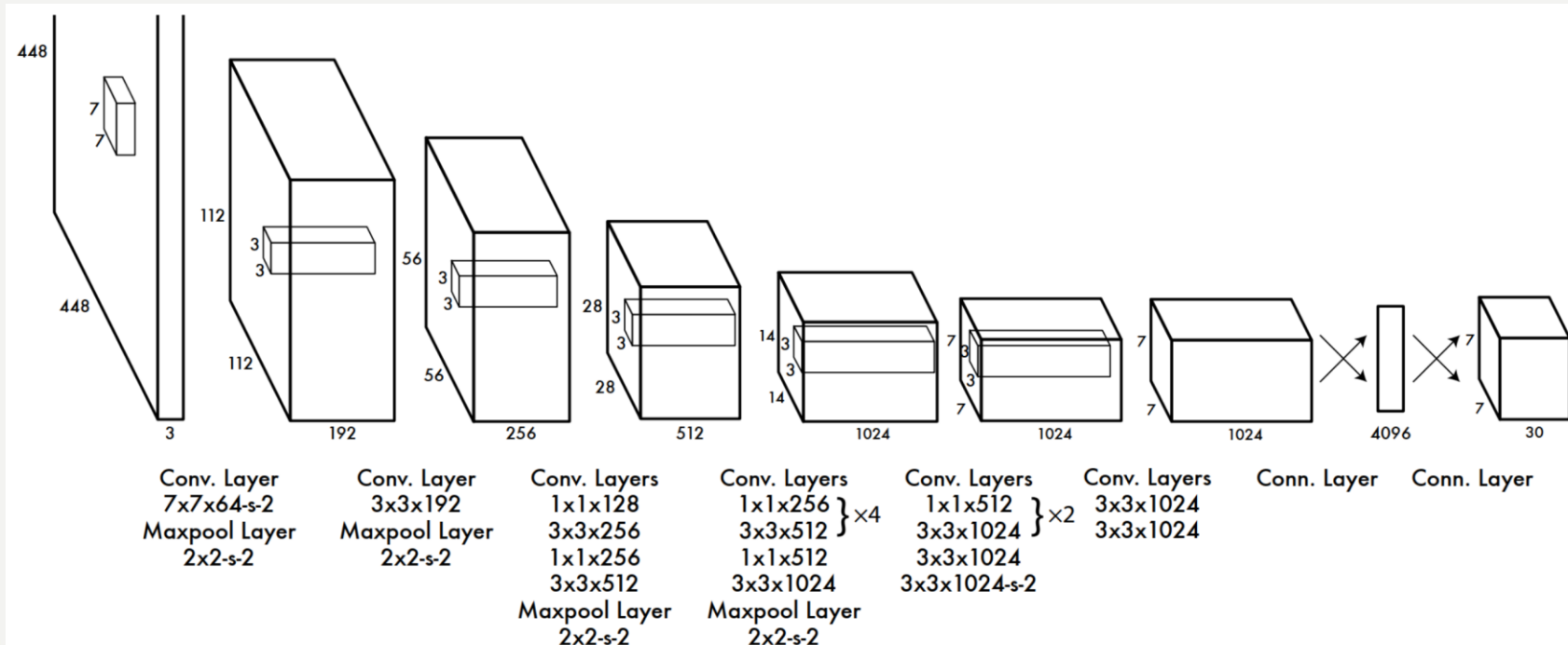
Final detections

# Classic YOLO Network Architecture

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

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

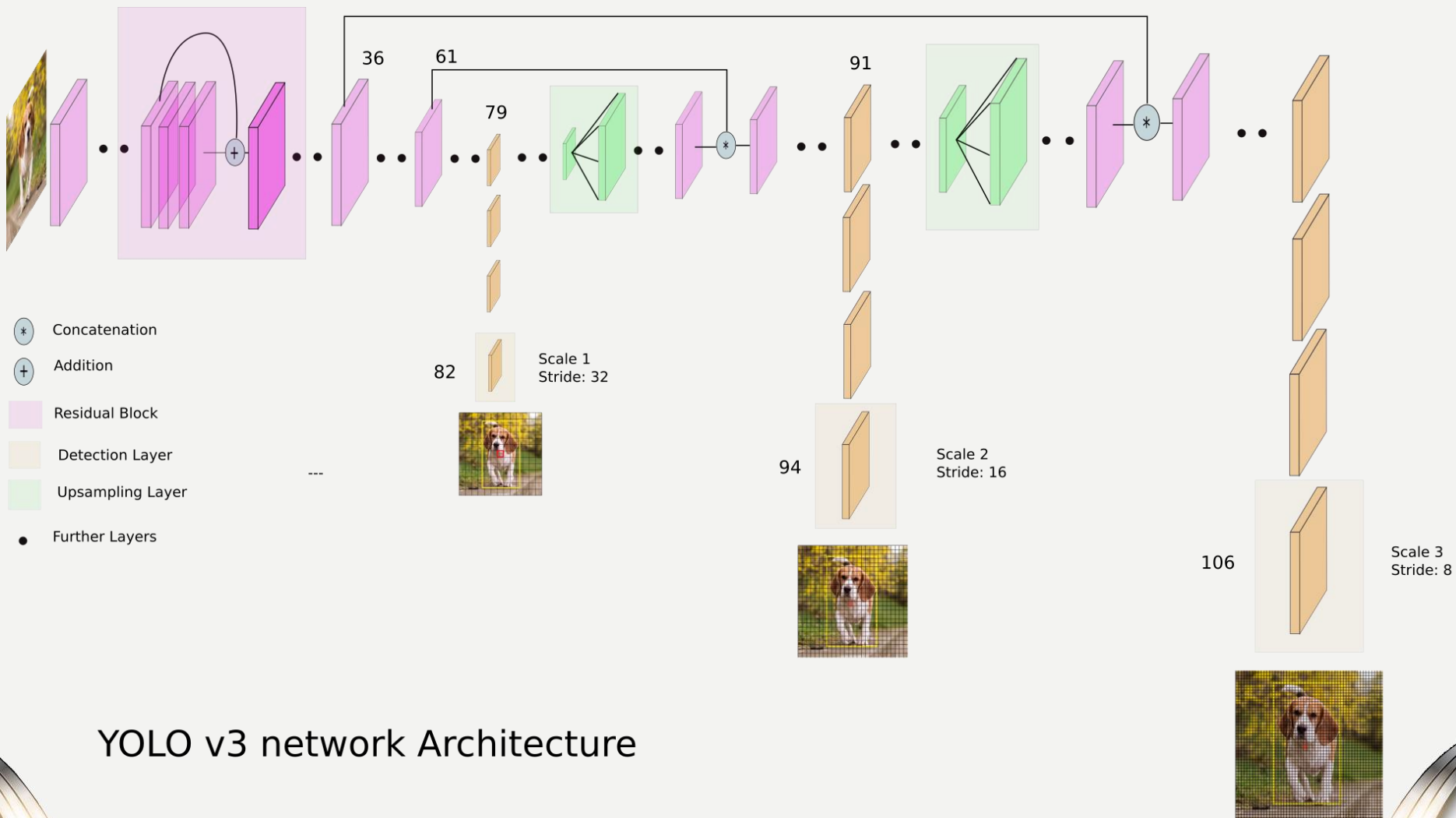
However, we can modify the original YOLO model in such a way that the numbers of cells in rows and columns differ.





# YOLO v3 Network Architecture

It detects better different size objects:



YOLO v3 network Architecture



# Bag of Freebies and Bag of Specials

Usually, a conventional object detector is trained offline. Therefore, researchers always like to take this advantage and develop better training methods which can make the object detector receive better accuracy without increasing the inference cost.

We call these methods that **only change the training strategy or only increase the training cost** as “**bag of freebies**.”

What is often adopted by object detection methods and meets the definition of bag of freebies is data augmentation, which purpose is to increase the variability of the input images, so that the designed object detection model has higher robustness to the images obtained from different environments.

These modules and post-processing methods that **only increase the inference cost by a small amount but can significantly improve the accuracy of object detection**, are call “**bag of specials**”. Generally speaking, these plugin modules are for enhancing certain attributes in a model, such as enlarging receptive field, introducing attention mechanism, or strengthening feature integration capability, etc., and post-processing is a method for screening model prediction results.

Common modules that can be used to enhance receptive field are SPP, ASPP, and RFB.

<https://arxiv.org/pdf/2004.10934.pdf>







# Improving Object Detection Training

For improving the object detection training, a CNN usually uses the following:

- **Activations:** ReLU, leaky-ReLU, parametric-ReLU, ReLU6, SELU, Swish, or Mish
- **Bounding box regression loss:** MSE, IoU, GloU, CloU, DloU
- **Data augmentation:** CutOut, MixUp, CutMix
- **Regularization method:** DropOut, DropPath, Spatial DropOut, or DropBlock
- **Normalization of the network activations by their mean and variance:** Batch Normalization (BN), Cross-GPU Batch Normalization (CGBN or SyncBN), Filter Response Normalization (FRN), or Cross-Iteration Batch Normalization (CBN)
- **Skip-connections:** Residual Connections, Weighted Residual Connections, Multi-input Weighted Residual Connections, or Cross Stage Partial Connections (CSP)

<https://arxiv.org/pdf/2004.10934.pdf>

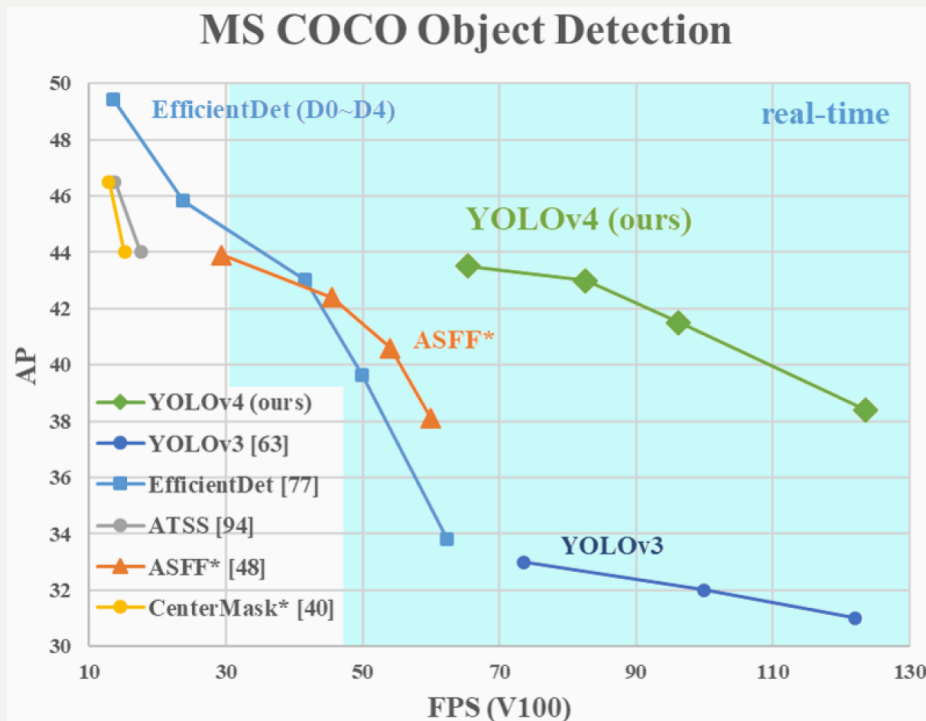


# YOLO v4 Network Architecture

YOLO v4 takes the influence of state of the art **bag of freebies (BoF)** and several **bag of specials (BoS)**:

- The **BoF** improves the accuracy of the detector, without increasing the inference time, only increasing the training cost.
- The **BoS** increases the inference cost by a small amount; however, significantly improving the accuracy of object detection.

YOLO v4 also based on the Darknet and has obtained an AP value of 43.5 percent on the COCO dataset along with a real-time speed of 65 FPS on the Tesla V100, beating the fastest and most accurate detectors in terms of both speed and accuracy.





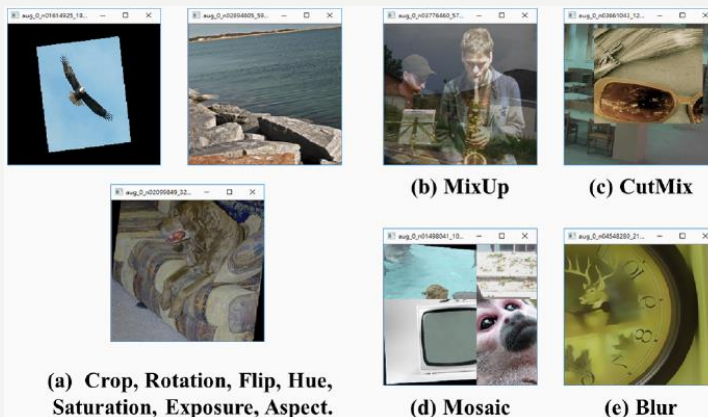
# YOLO v4 Network Architecture

YOLOv4 consists of:

- **Backbone:** CSPDarknet53 [81]
- **Neck:** SPP [25], PAN [49]
- **Head:** YOLOv3 [63]

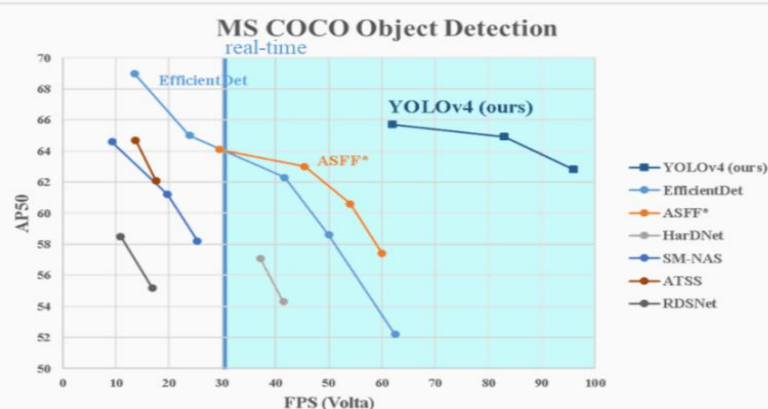
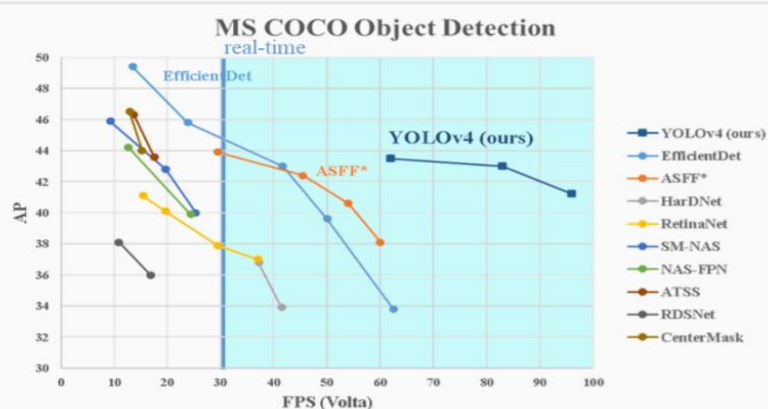
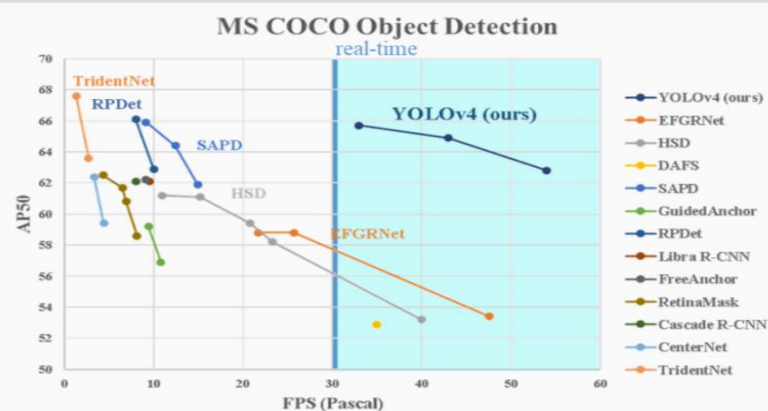
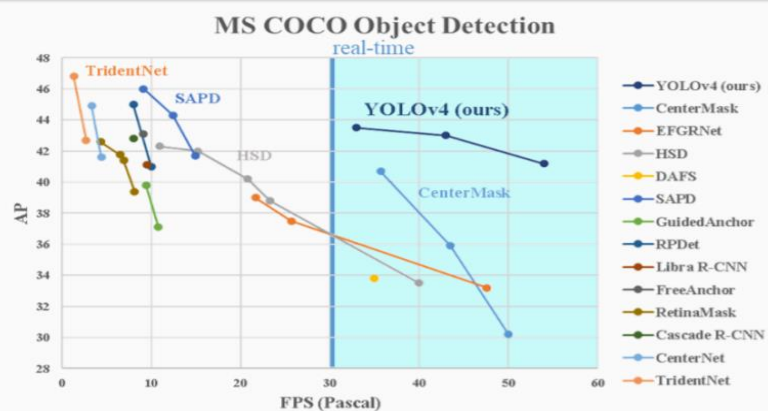
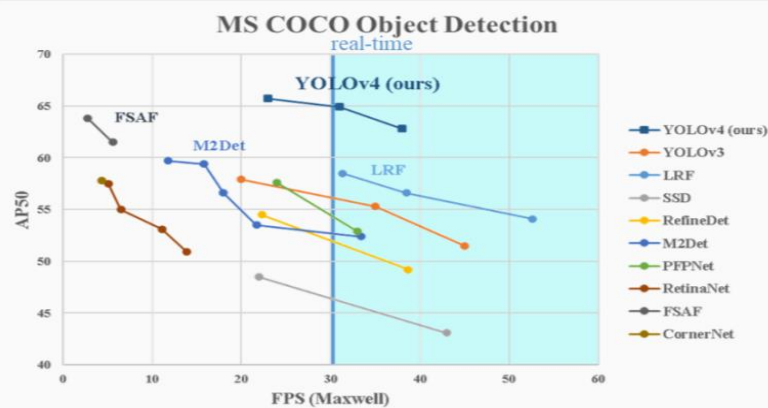
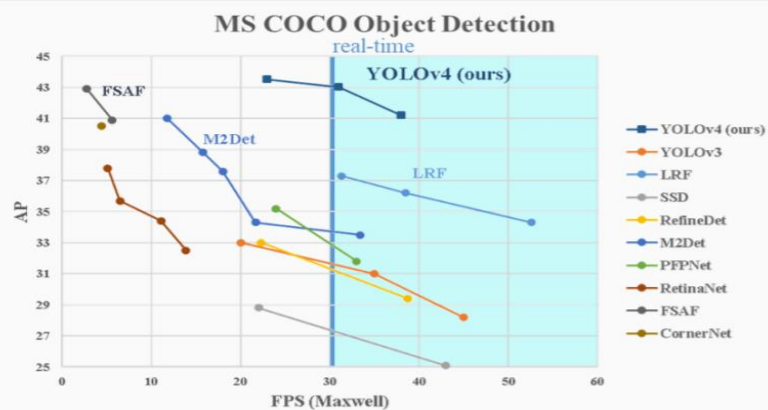
YOLO v4 uses:

- **Bag of Freebies (BoF) for backbone:**
  - CutMix and Mosaic data augmentation,
  - DropBlock regularization,
  - Class label smoothing
- **Bag of Specials (BoS) for backbone:**
  - Mish activation,
  - Cross-stage partial connections (CSP),
  - Multiinput weighted residual connections (MiWRC)



- **Bag of Freebies (BoF) for detector:**
  - CloU-loss,
  - CmBN,
  - DropBlock regularization,
  - Mosaic data augmentation,
  - Self-Adversarial Training,
  - Eliminate grid sensitivity,
  - Using multiple anchors for a single ground truth,
  - Cosine annealing scheduler,
  - Optimal hyperparameters,
  - Random training shapes
- **Bag of Specials (BoS) for detector:**
  - Mish activation,
  - SPP-block,
  - SAM-block,
  - PAN path-aggregation block,
  - DIoU-NMS

# Comparisons of YOLO v4 on the Different GPU Cards: Maxwell, Pascal and Volta

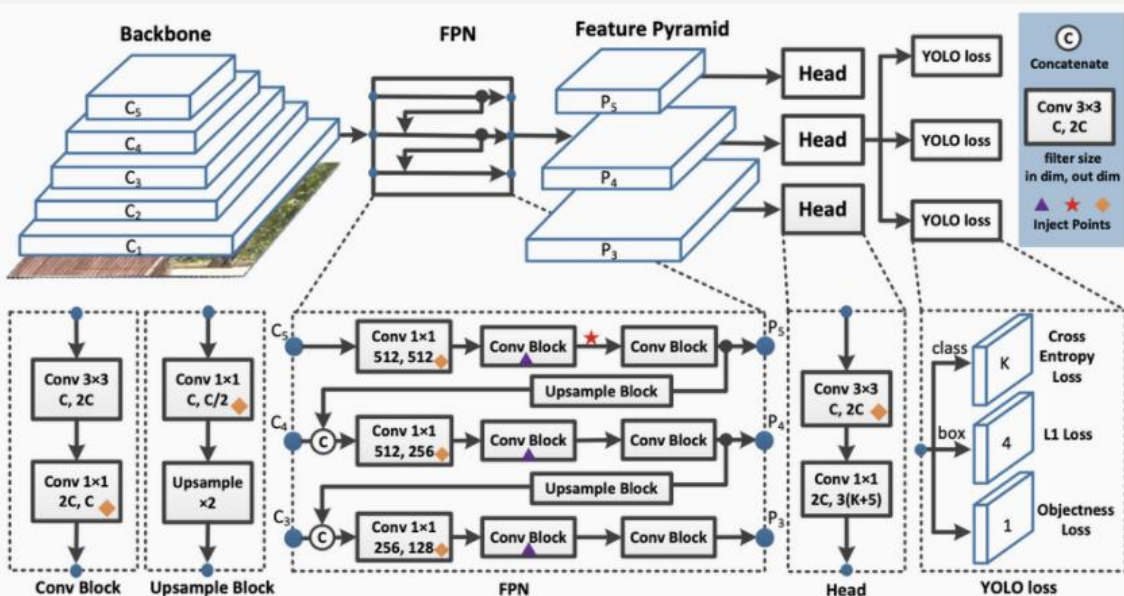




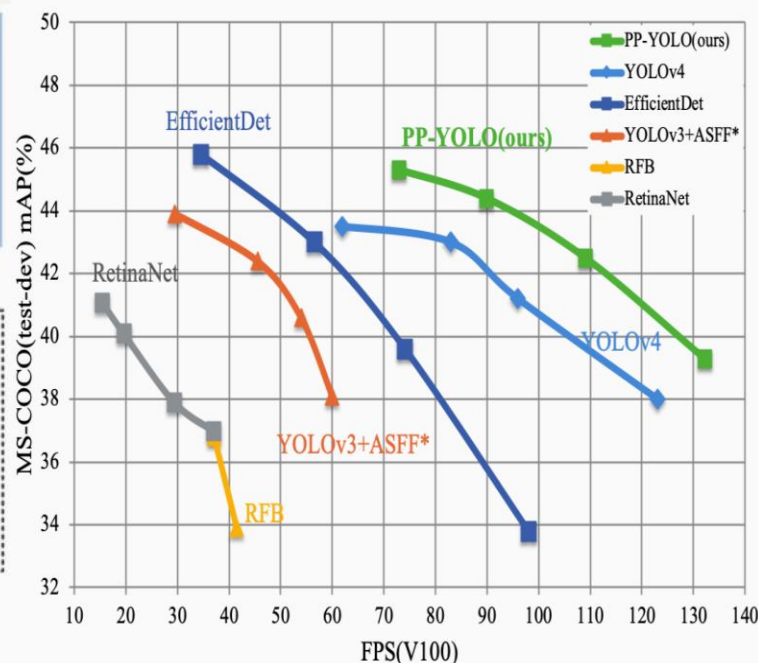


# PP-YOLO

PP-YOLO has been introduced in July 2020. It is based on PaddlePaddle and on YOLO v3. This object detector with relatively balanced effectiveness and efficiency that can be directly applied in actual application scenarios. The notable changes include the replacement of Darknet53 backbone of YOLO v3 with a **ResNet backbone** and increase of training batch size from 64 to 192 (as mini-batch size of 24 on 8 GPUs):



The modeling process in PP-YOLO



<https://arxiv.org/abs/2007.12099> (Original paper: **PP-YOLO: An Effective and Efficient Implementation of Object Detector**, by Xiang Long et al)

<https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109>



# YOLO v5

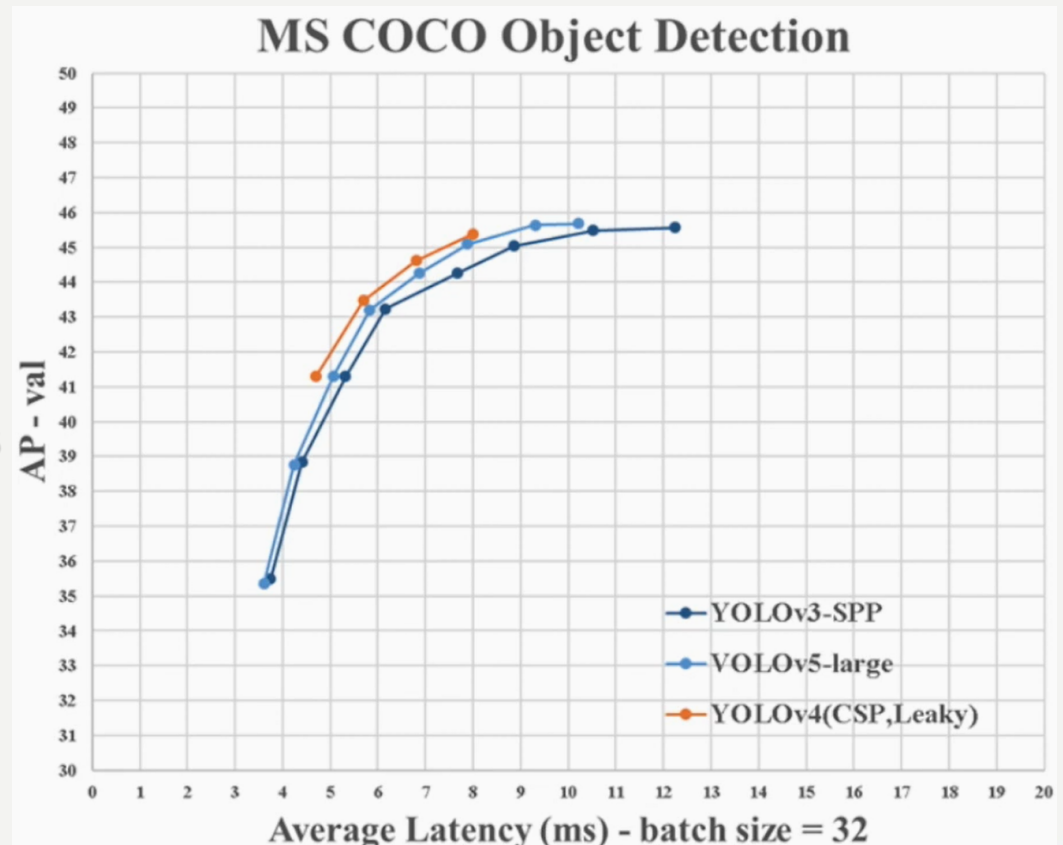
YOLO v5 is different from all other prior releases (developed by Roboflow team), as this is a PyTorch implementation rather than a fork from original Darknet.

Same as YOLO v4, the YOLO v5 has a CSP **backbone** and PA-NET **neck**.

The major improvements includes **mosaic data augmentation** and **auto learning bounding box anchors**.

YOLO v5 is not to achieve the best mAP, but instead:

- easy of use
- exportability
- low memory requirements
- high speed
- high mAP
- market size (small)
- new PyTorch framework





# Let's Play with Object Detection and Segmentation Algorithms in Roboflow:



There is a nice application with build-in modules, datasets and models:

1. <http://app.roboflow.ai>
2. <http://public.roboflow.ai>
3. <http://models.roboflow.ai>

## Create Project

Extract [chessSampleData.zip](#) and have a look at its contents. It has 12 [jpg](#) images of chess boards and 11 [xml](#) files labeling the pieces in [voc format](#).

In this tutorial, we will prepare this dataset for training by

- Uploading the images
- Annotating an unlabeled image
- Splitting the dataset into [train](#), [valid](#) and [test](#) sets
- Downsizing and grayscaling the images
- Generating additional training examples
- Converting the annotation format
- And creating a hosted link to use in our training script

🕒 This guided tutorial will take about 5 minutes.

## Roboflow Train

Roboflow Train is our new one-click model training service that enables you to train your model without writing any code.

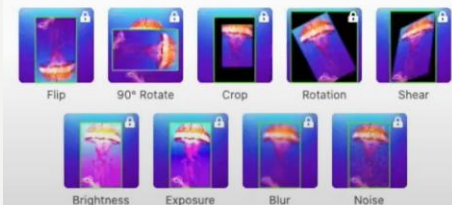
Once training is complete, you'll get the results along with a hosted API endpoint you can use for making predictions in your project.

- ✓ Model Evaluation Metrics
- ✓ Hosted API Endpoint for Inference
- ✓ Use with Model Assisted Labeling **PRO**
- ✓ On-Device Inference **PRO**

### IMAGE LEVEL AUGMENTATIONS



### BOUNDING BOX LEVEL AUGMENTATIONS



Use video tutorials of creating and training YOLO v5 models:

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

<https://www.youtube.com/watch?v=R1Bf067Z5uM>

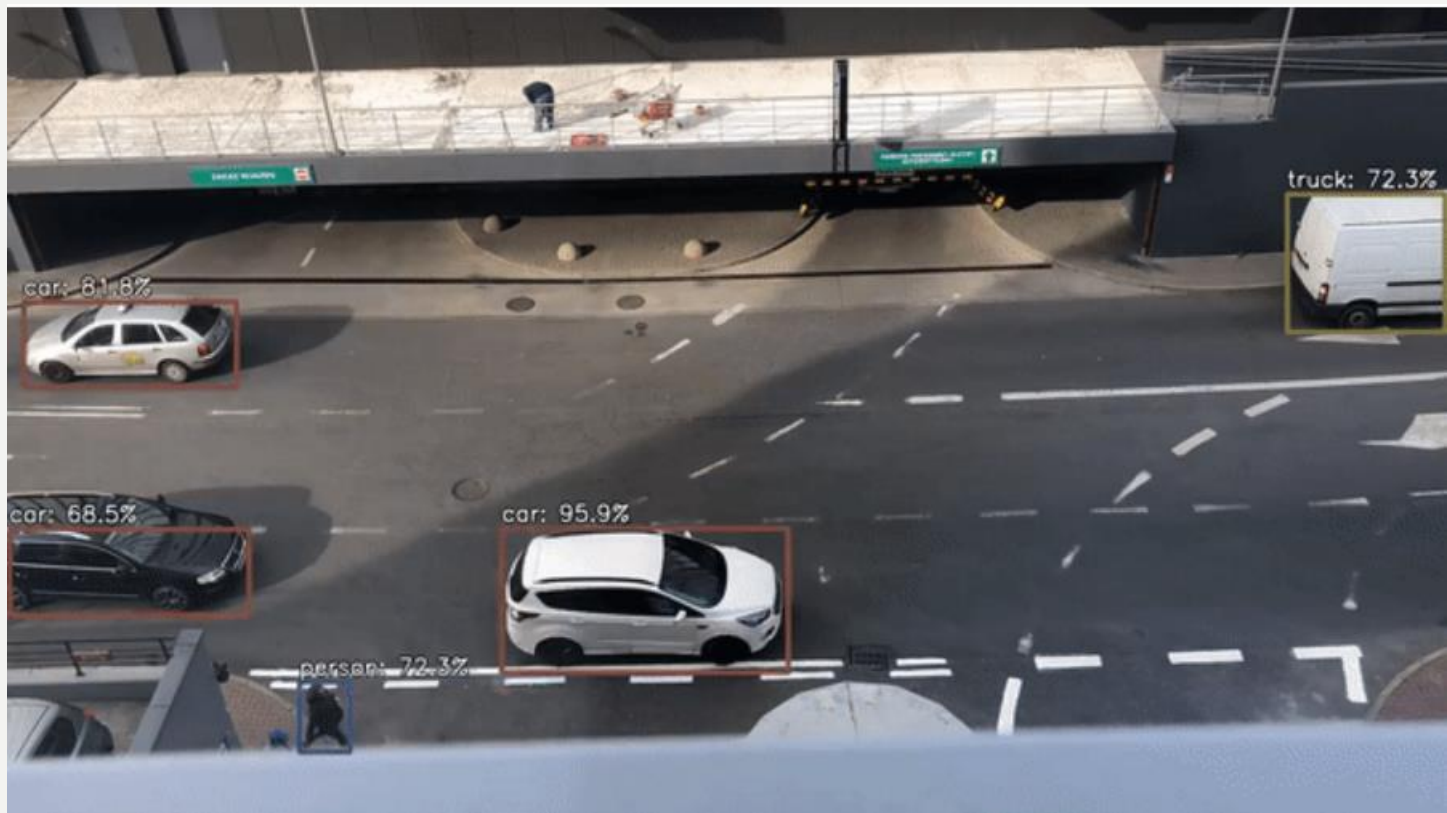
Watch the video and construct your model as an optional assignment if you like?



# RetinaNet

## RetinaNet:

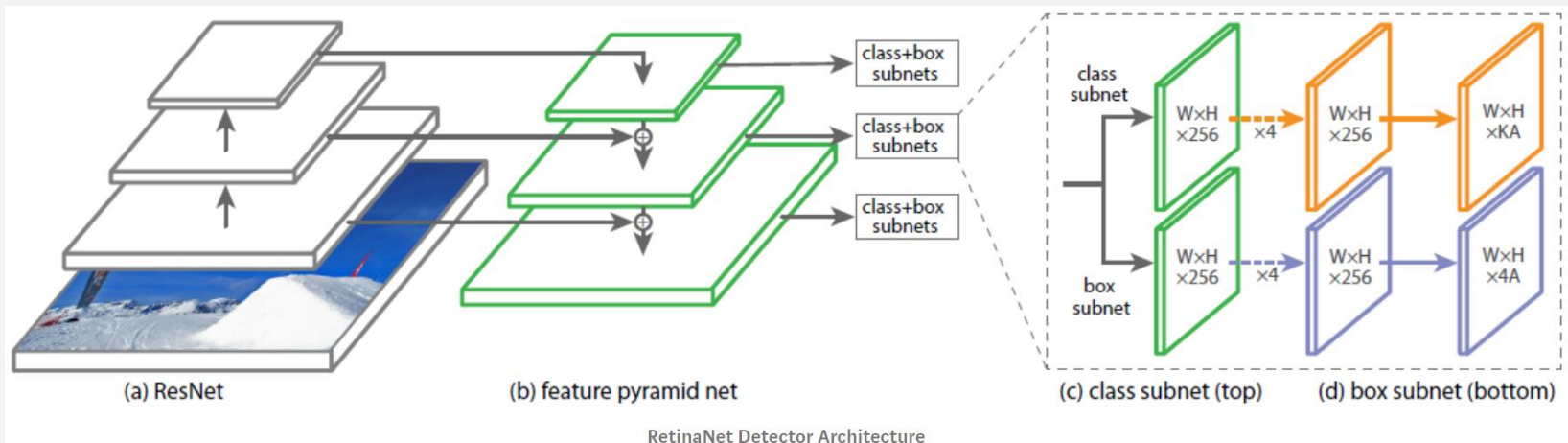
- can have ~100k boxes with the resolve of class imbalance problem using focal loss.
- Many one-stage detectors do not achieve good enough performance, so there are build new two-stage detectors.



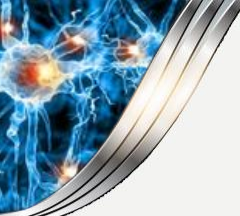
# RetinaNet

## RetinaNet:

- In RetinaNet, a 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>







# Precision and Recall

To define Mean Average Measure (mAP), we will use the following:  
**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.

$$\text{Precision} = \frac{TP}{TP+FP}$$

## 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.

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

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



# Mean Average Precision

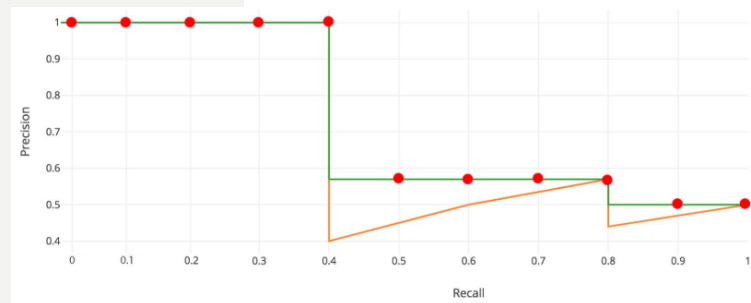
## Average Precision (AP):

- 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:

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

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

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



## Mean Average Precision (mAP):

- is a mean **average precision** computes the average precision value for recall value over 0 to 1.



# Semantic Segmentation and Instance Segmentation

How can we segment objects in images?



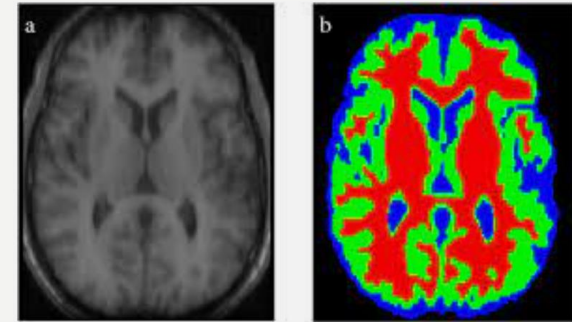
# Semantic Segmentation

Semantic segmentation is one of the key problems in the field of **computer vision**. It paves the way towards complete **scene understanding**. An increasing number of applications nourish from inferring knowledge from imagery. Some of those applications include self-driving vehicles, human-computer interaction, virtual reality etc.

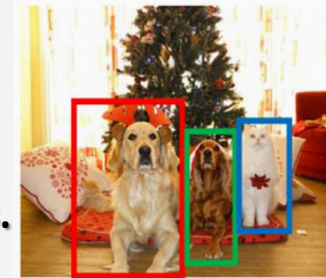
With the popularity of deep learning in recent years, many semantic segmentation problems are being tackled using deep architectures, like CNN, which surpass other approaches in terms of accuracy and efficiency.

**Semantic segmentation** is a natural step in the progression from coarse to fine inference:

1. The origin could be located at **classification of objects**, which consists of making a prediction for a whole input.
2. The next step is **localization / detection of objects**, which provides not only the classes but also additional information regarding **the spatial location** of those classes.
3. Finally, **semantic segmentation of objects** achieves fine-grained inference by making dense predictions inferring labels for every pixel so that **each pixel is labeled with the class of its enclosing object or region**.



Object  
Detection



dog dog cat

Instance  
Segmentation



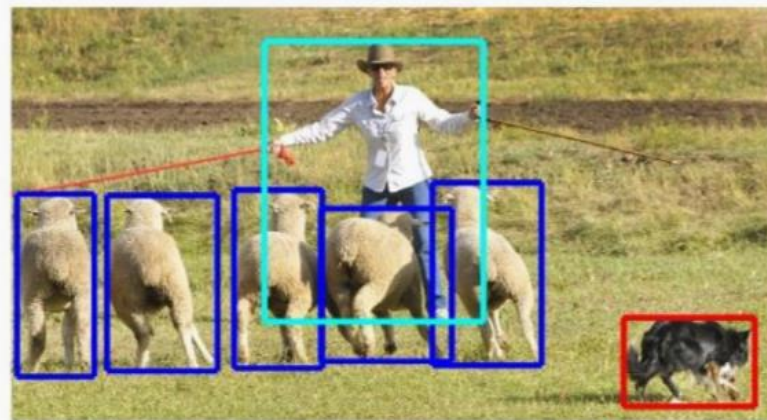
dog dog cat

# Segmentation and Localization

We can localize, segment and describe objects:



(a) Image classification



(b) Object localization



(c) Semantic segmentation

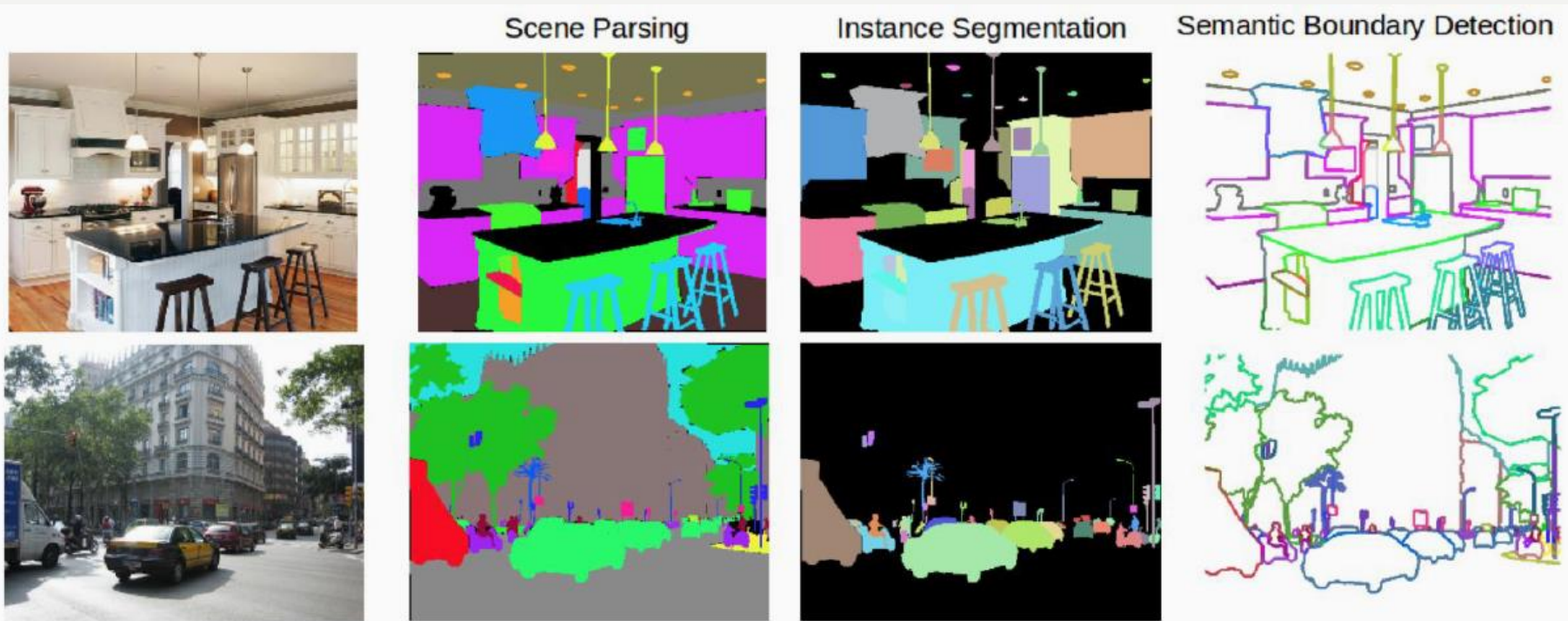


(d) Segmentation in context



# Scene Parsing, Segmentation and Boundary Detection

To understand the scene, we must detect objects, their boundaries, key points, segment them, mask, and process in context.



Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. "Mask R-CNN." ICCV, 2017



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

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

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

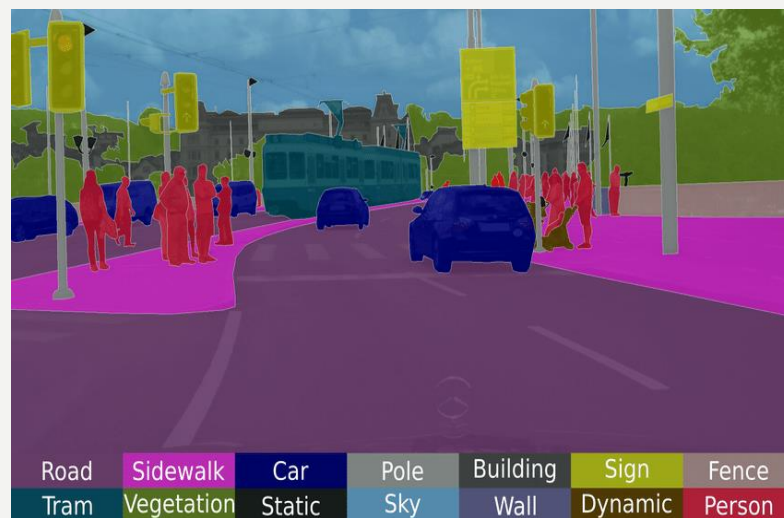


## Fast R-CNN:

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

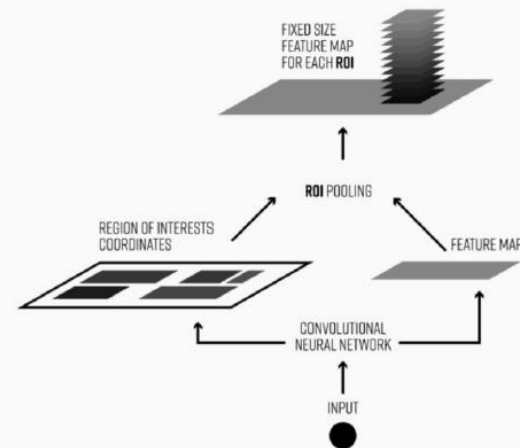
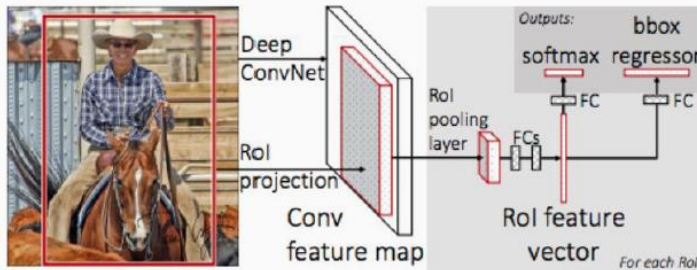
## Faster R-CNN:

- Uses a convolutional network to propose regions.



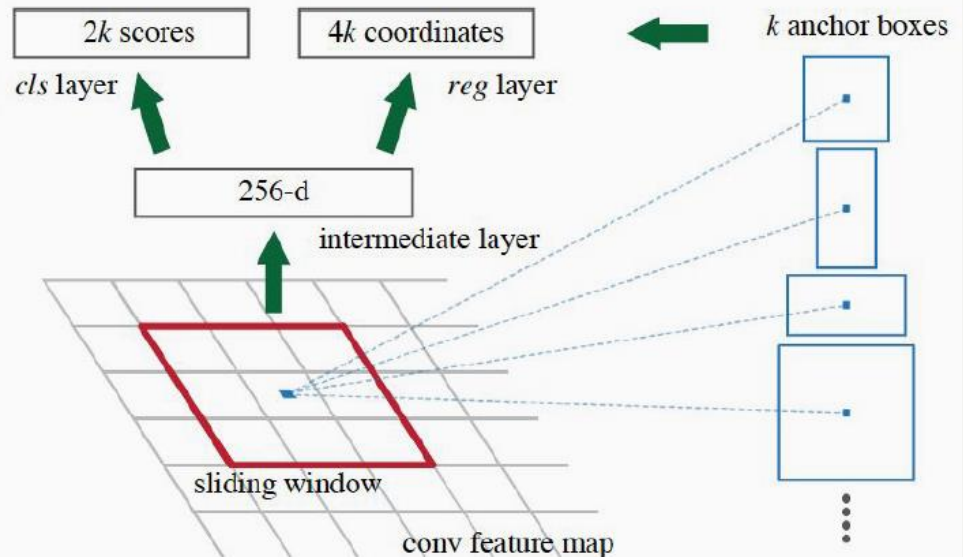
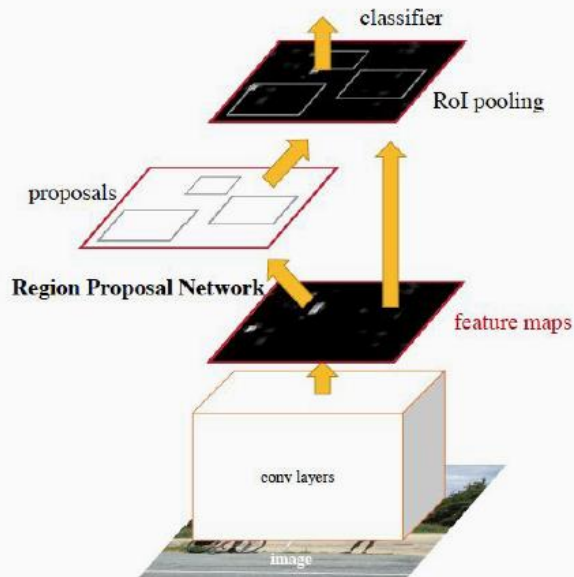
# Fast R-CNN and Faster R-CNN

## Fast R-CNN



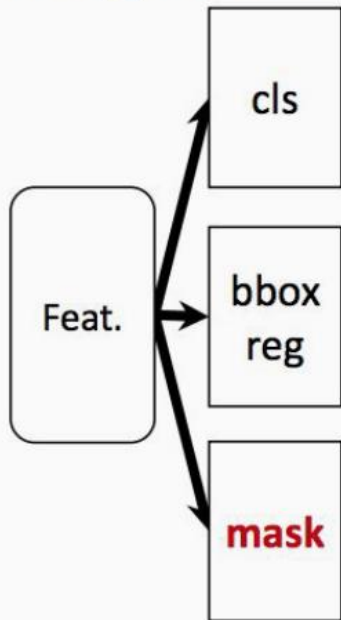
## Faster R-CNN = RPN + Fast R-CNN

### RPN = Fully Convolutional Network

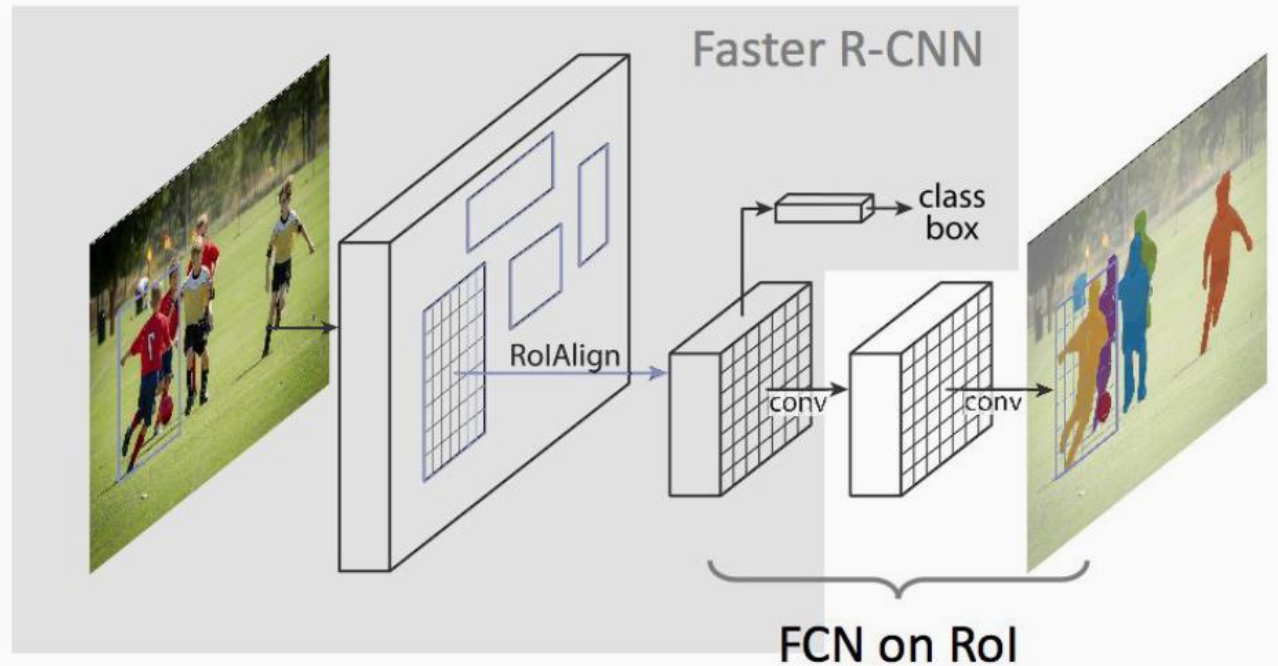


# Mask Prediction using Faster R-CNN

## Insight: Mask Prediction in Parallel



Mask R-CNN

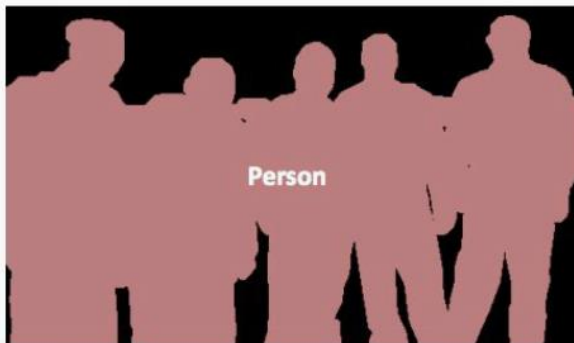




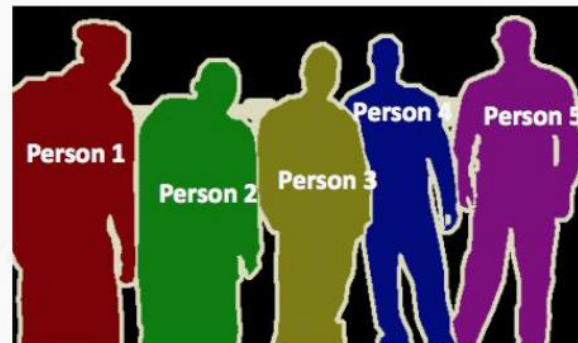
# Semantic and Instance Segmentation



Object Detection



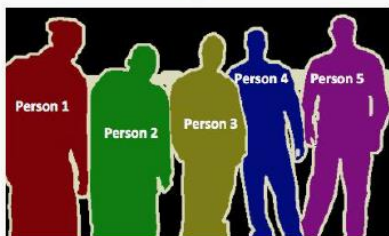
Semantic Segmentation



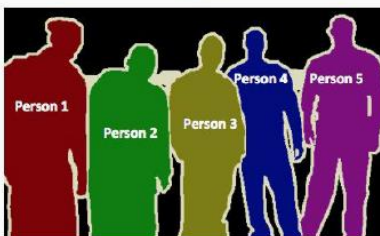
Instance Segmentation

**Instance Segmentation Methods can be divided into:**

**R-CNN driven**



**FCN driven**





# Examples of Masks



Figure 5. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).



# Human Pose Estimations

Human Pose Estimations are used to detect and track actions performed by people to control the motor to react to what they do.

They may be used for movement improvements in sport, to detect undesirable behaviors or gathering training data for robots:



Figure 7. Keypoint detection results on COCO test using Mask R-CNN (ResNet-50-FPN), with person segmentation masks predicted from the same model. This model has a keypoint AP of 63.1 and runs at 5 fps.





# Sample Implementation of Detection Model

Let's use Roboflow to implement detection!

# Implementation of Detection Models

You can find many implemented frameworks for object detections, localization, detection and segmentation online (on the github) and utilize them for free.

You can also use applications like Roboflow:

1. <http://app.roboflow.ai>
2. <http://public.roboflow.ai>
3. <http://models.roboflow.ai>

Use video tutorials of creating and training YOLO v5 models:

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

<https://www.youtube.com/watch?v=R1Bf067Z5uM>

	from	n	params	module	arguments
0		-1 1	3520	models.common.Focus	[3, 32, 3]
1		-1 1	18560	models.common.Conv	[32, 64, 3, 2]
2		-1 1	19904	models.common.BottleneckCSP	[64, 64, 1]
3		-1 1	73984	models.common.Conv	[64, 128, 3, 2]
4		-1 1	161152	models.common.BottleneckCSP	[128, 128, 3]
5		-1 1	295424	models.common.Conv	[128, 256, 3, 2]
6		-1 1	641792	models.common.BottleneckCSP	[256, 256, 3]
7		-1 1	1180672	models.common.Conv	[256, 512, 3, 2]
8		-1 1	656896	models.common.SPP	[512, 512, [5, 9, 13]]
9		-1 1	1248768	models.common.BottleneckCSP	[512, 512, 1, False]
10		-1 1	131584	models.common.Conv	[512, 256, 1, 1]
11		-1 1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12		[-1, 6] 1	0	models.common.Concat	[1]
13		-1 1	378624	models.common.BottleneckCSP	[512, 256, 1, False]
14		-1 1	33024	models.common.Conv	[256, 128, 1, 1]
15		-1 1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
16		[-1, 4] 1	0	models.common.Concat	[1]
17		-1 1	95104	models.common.BottleneckCSP	[256, 128, 1, False]
18		-1 1	147712	models.common.Conv	[128, 128, 3, 2]
19		[-1, 14] 1	0	models.common.Concat	[1]
20		-1 1	313088	models.common.BottleneckCSP	[256, 256, 1, False]
21		-1 1	590336	models.common.Conv	[256, 256, 3, 2]
22		[-1, 10] 1	0	models.common.Concat	[1]
23		-1 1	1248768	models.common.BottleneckCSP	[512, 512, 1, False]
24		[17, 20, 23] 1	21576	models.yolo.Detect	[3, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]

Model Summary: 283 layers, 7260488 parameters, 7260488 gradients, 16.8 GFLOPS

```
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes} # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]], # 0-P1/2
  [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
  [-1, 3, BottleneckCSP, [128]],
  [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
  [-1, 9, BottleneckCSP, [256]],
  [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
  [-1, 9, BottleneckCSP, [512]],
  [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
  [-1, 1, SPP, [1024, [5, 9, 13]]],
  [-1, 3, BottleneckCSP, [1024, False]], # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
  [-1, 1, nn.Upsample, [None, 2, 'nearest']],
  [[-1, 6], 1, Concat, [1]], # cat backbone P4
  [-1, 3, BottleneckCSP, [512, False]], # 13

  [-1, 1, Conv, [256, 1, 1]],
  [-1, 1, nn.Upsample, [None, 2, 'nearest']],
  [[-1, 4], 1, Concat, [1]], # cat backbone P3
  [-1, 3, BottleneckCSP, [256, False]], # 17 (P3/8-small)

  [-1, 1, Conv, [256, 3, 2]],
  [[-1, 14], 1, Concat, [1]], # cat head P4
  [-1, 3, BottleneckCSP, [512, False]], # 20 (P4/16-medium)

  [-1, 1, Conv, [512, 3, 2]],
  [[-1, 10], 1, Concat, [1]], # cat head P5
  [-1, 3, BottleneckCSP, [1024, False]], # 23 (P5/32-large)

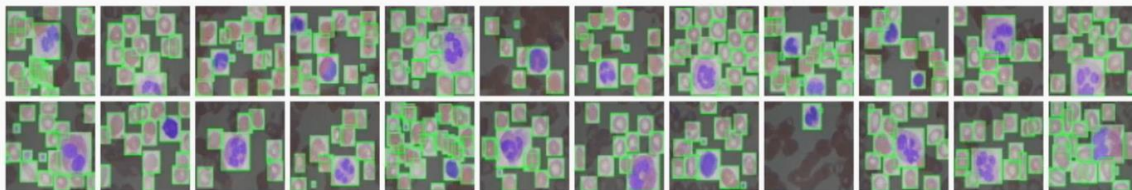
  [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
  ]
```

# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

1. Create a free account: <https://app.roboflow.com/>
2. Fork sample dataset, e.g. BCCD, and use it.

## Preview



roboflow Projects Documentation

ADRIAN HORZYK  
BCCD

Upload Annotate Dataset Health Check

Generate New Version

VERSIONS

To train a model, you must first generate a new version of your dataset.

Choose your dataset settings to get started.

3 Preprocessing

Decrease training time and increase performance by applying image transformations to all images in this dataset.

Resize Stretch to 416x416 Edit

Add Preprocessing Step

Continue

4 Augmentation

5 Generate

## Fork Dataset

Forking a dataset copies the source images to your account so you can choose whichever export settings you want.

If you want to use one of this dataset's preset export settings, browse the available Downloads instead.

Cancel

Fork Dataset

## Rebalance Train/Test Split

You can update your dataset's [train/test split](#) here.

**Note:** changing your test set will invalidate model performance comparisons with previously generated versions.

Train  
255

Valid  
73

Test  
36

Not sure what this is? [Learn more on our blog >>](#)

Cancel

Save

## bccd-yolov5 Dataset Health Check

Images  
364

0 missing annotations  
0 null examples

Annotations  
5,075

13.9 per image (average)  
across 3 classes

### Class Balance

RBC	4,338	over represented
WBC	375	under represented
Platelets	362	under represented



# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

## 3. Preprocessing of the training data:

- Stretching
- Filling
- Fitting
- etc.

3

### Preprocessing

Decrease training time and increase performance by applying image transformations to all images in this dataset.

Resize  
Stretch to 416×416

Edit

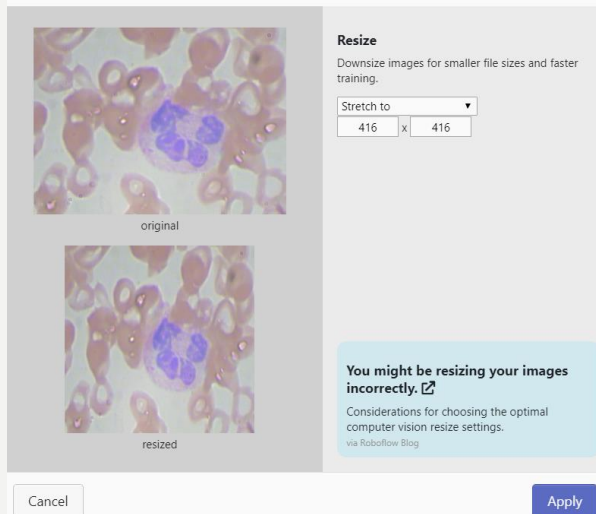
×

+ Add Preprocessing Step

Continue

Resize

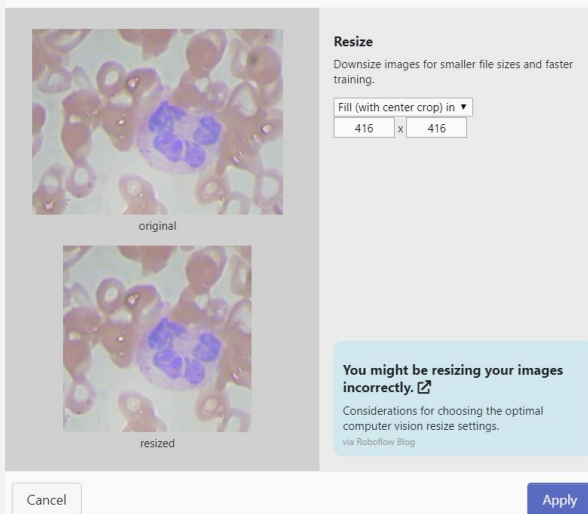
×



The dialog shows a microscopy image being stretched to 416x416. The 'original' image is shown above the 'resized' image, which is wider and taller. A text box at the bottom states: 'You might be resizing your images incorrectly. Considerations for choosing the optimal computer vision resize settings. via Roboflow Blog'.

Resize

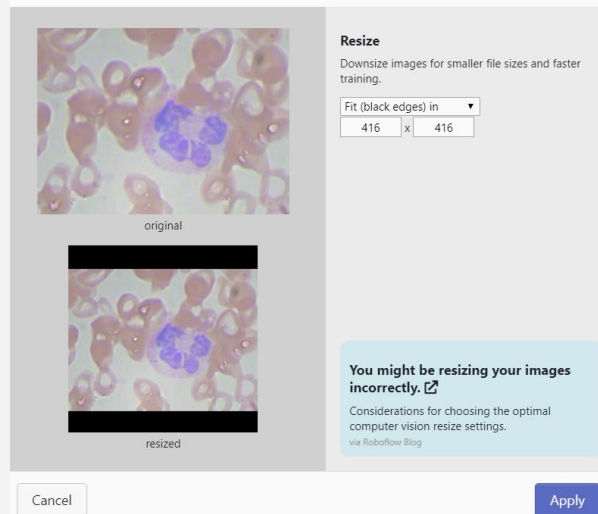
×



The dialog shows a microscopy image being filled with a center crop to 416x416. The 'original' image is shown above the 'resized' image, which is the same size but with black bars on the sides. A text box at the bottom states: 'You might be resizing your images incorrectly. Considerations for choosing the optimal computer vision resize settings. via Roboflow Blog'.

Resize

×



The dialog shows a microscopy image being fitted to 416x416. The 'original' image is shown above the 'resized' image, which is the same size but with black bars on the top and bottom. A text box at the bottom states: 'You might be resizing your images incorrectly. Considerations for choosing the optimal computer vision resize settings. via Roboflow Blog'.

# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>



## 4. Augmentation for the enrichment of training data to achieve better performance

4

### Augmentation

Create new training examples for your model to learn from by generating augmented versions of each image in your training set.

Flip

Horizontal, Vertical

Edit

×

90° Rotate

Clockwise, Counter-Clockwise, Upside Down

Edit

×

Crop

0% Minimum Zoom, 15% Maximum Zoom

Edit

×

Hue

Between -25° and +25°

Edit

×

Saturation

Between -25% and +25%

Edit

×

Brightness

Between -15% and +15%

Edit

×

Exposure

Between -20% and +20%

Edit

×

Blur

Up to 3px

Edit

×

Noise

Up to 10% of pixels

Edit

×



Add Augmentation Step

### Augmentation Options

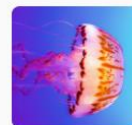


Augmentations create new training examples for your model to learn from.

#### IMAGE LEVEL AUGMENTATIONS



Flip



90° Rotate



Crop



Rotation



Shear



Grayscale



Hue



Saturation



Brightness



Exposure



Blur



Noise



Cutout



Mosaic

#### BOUNDING BOX LEVEL AUGMENTATIONS ?



Flip



90° Rotate



Crop



Rotation



Shear



Brightness



Exposure



Blur



Noise



# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>



### Flip

**Flip**  
Add horizontal or vertical flips to help your model be insensitive to subject orientation.

☒ Horizontal  
☒ Vertical

preprocessed

vertical horizontal

**How Flip Augmentation Improves Model Performance**  
Flipping an image can improve model performance in substantial ways.  
[via Roboflow Blog](#)

### 90° Rotate

**90° Rotate**  
Add 90-degree rotations to help your model be insensitive to camera orientation.

☒ Clockwise  
☒ Counter-Clockwise  
☒ Upside Down

preprocessed clockwise

counter-clockwise upside-down

**When should I rotate my images?**  
If orientation doesn't matter (eg they may be taken in portrait/landscape mode or from above).  
[via Roboflow Blog](#)

### Crop

**Crop**  
Add variability to positioning and size to help your model be more resilient to subject translations and camera position.

0% 15% 99%

0% 15%

**When should I use Random Crop?**  
Short answer: If subjects in the wild may be occluded or may not be fully enclosed in the frame.  
[via Roboflow Blog](#)

### Hue

**Hue**  
Randomly adjust the colors in the image.

0° 25° 180°

original

-25° 25°

**What is hue augmentation?**  
It randomly changes the colors to make your model less sensitive.  
[via Roboflow Blog](#)

### Saturation

**Saturation**  
Randomly adjust the vibrancy of the colors in the images.

0% 25% 99%

original

-25% 25%

**What is the saturation augmentation?**  
It randomly adjusts your images' colors to make them more or less vibrant.  
[via Roboflow Blog](#)

### Brightness

**Brightness**  
Add variability to image brightness to help your model be more resilient to lighting and camera setting changes.

0% 15% 99%

0%

☒ Brighten  
☒ Darken

-15% 15%

### Blur

**Blur**  
Add random Gaussian blur to help your model be more resilient to camera focus.

0px 3px 25px

0px

3px

**When should I use Random Blur?**  
If your subjects in-the-wild might not be in focus or your model is overfitting on hard edges.  
[via Roboflow Blog](#)

### Noise

**Noise**  
Add noise to help your model be more resilient to camera artifacts.

0% 10% 25%

0%

10%

**Why would I add noise to my images?**  
Noise can help defend against adversarial attacks and prevent overfitting.  
[via Roboflow Blog](#)

### Exposure

**Exposure**  
Add variability to image brightness to help your model be more resilient to lighting and camera setting changes.

0% 20% 99%

0%

-20% 20%

Go Back

Apply

Go Back

Apply

Cancel

Apply



# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

## 5. Generation step creates a ready-to-use training data set using the data augmentation:

5

### Generate

Review your selections and select a version size to create a moment-in-time snapshot of your dataset with the applied transformations.

Larger versions take longer to train but often result in better model performance. [See how this is calculated >>](#)

Maximum Version Size

874 images (3x) ▾

Generate

#### BCCD Dataset

Generate New Version

VERSIONS

2021-05-16 1:27pm  
v1 May 16, 2021

2021-05-16 BCCD v1

Save Name

Version 1 Generated May 16, 2021

The images for your new dataset version are now being created. This may take a few moments as machines spin up to process all of the images.

Generating images...

2021-05-16 BCCD v1

Version 1 Generated May 16, 2021

Export

More ▾

#### TRAINING OPTIONS

##### Use Roboflow Train

Let us train your model and get results within 24 hours along with a hosted API endpoint for making predictions. [Learn More >>](#)

Start Training

Available Credits: 0

##### Train Outside Roboflow

Export your data to use a model from [our model library >>](#) with Google Colab or your own machine.

Format

YOLO v5 PyTorch ▾

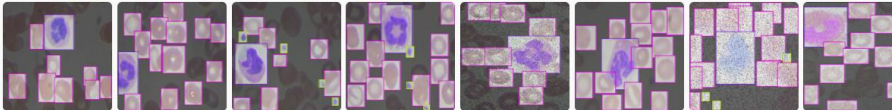
Export

#### IMAGES

874 images

View All Images >>

IMAGES



#### TRAIN / TEST SPLIT

Training Set

88%

765 images

Validation Set

8%

73 images

Testing Set

4%

36 images

## 6. After these five steps, we are ready to Start Training:

Roboflow Train

Roboflow Train is our new one-click model training service that enables you to train your model without writing any code.

Once training is complete, you'll get the results along with a hosted API endpoint you can use for making predictions in your project.

✓ Model Evaluation Metrics

✓ Hosted API Endpoint for Inference

✓ Use with Model Assisted Labeling PRO

✓ On-Device Inference PRO

Cancel

Request Access

53

# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

7. Export the data to use the created model with Google Colab or your own machine:

8. Open YOLOv5 Colab Notebook

PyTorch Object Detection :: YOLOv5 TXT

## YOLOv5

A very fast and easy to use PyTorch model that achieves state of the art (or near state of the art) results. [Read More...](#)

[YOLOv5 Tutorial](#) [YOLOv5 Video](#) [YOLOv5 Repo](#) [YOLOv5 Colab Notebook](#)

9. Use your secret code with your dataset:

### Your Download Code

[Jupyter](#) [Terminal](#) [Raw URL](#)

Paste this snippet into a [notebook from our model library](#) to download and unzip [your dataset](#):

```
!curl -L "https://app.roboflow.com/ds/REPLACE-THIS-LINK" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

**Warning:** Do not share this link beyond your team, it contains a private key that is tied to your Roboflow account. Acceptable use policy applies.

Done

```
# Export code snippet and paste here
%cd /content
!curl -L "https://app.roboflow.com/ds/REPLACE-THIS-LINK" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

10. And start training:

```
# train yolov5s on custom data for 100 epochs
# time its performance
%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 100 --data './data.yaml' --cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results --cache
```

# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

## 11. When training is finished, we can see the result using the tensorboard:

Epoch	gpu_mem	box	obj	cls	total	targets	img_size		
99/99	1.39G	0.03056	0.1046	0.0007596	0.1359	396	416:	100%	45/45 [00:05<00:00, 7.52it/s]
	Class	Images	Targets	P	R	mAP@.5	mAP@.5:.95		
	all	73	967	0.874	0.926	0.931	0.617		
	Platelets	73	76	0.845	0.934	0.912	0.476		
	RBC	73	819	0.81	0.844	0.901	0.62		
	WBC	73	72	0.967	1	0.978	0.753		

Optimizer stripped from runs/train/yolov5s\_results/weights/last.pt, 14.8MB  
Optimizer stripped from runs/train/yolov5s\_results/weights/best.pt, 14.8MB  
100 epochs completed in 0.202 hours.

CPU times: user 8.39 s, sys: 1.01 s, total: 9.4 s  
Wall time: 12min 29s

```
[10] # Start tensorboard
      # Launch after you have started training
      # logs save in the folder "runs"
      %load_ext tensorboard
      %tensorboard --logdir runs
```



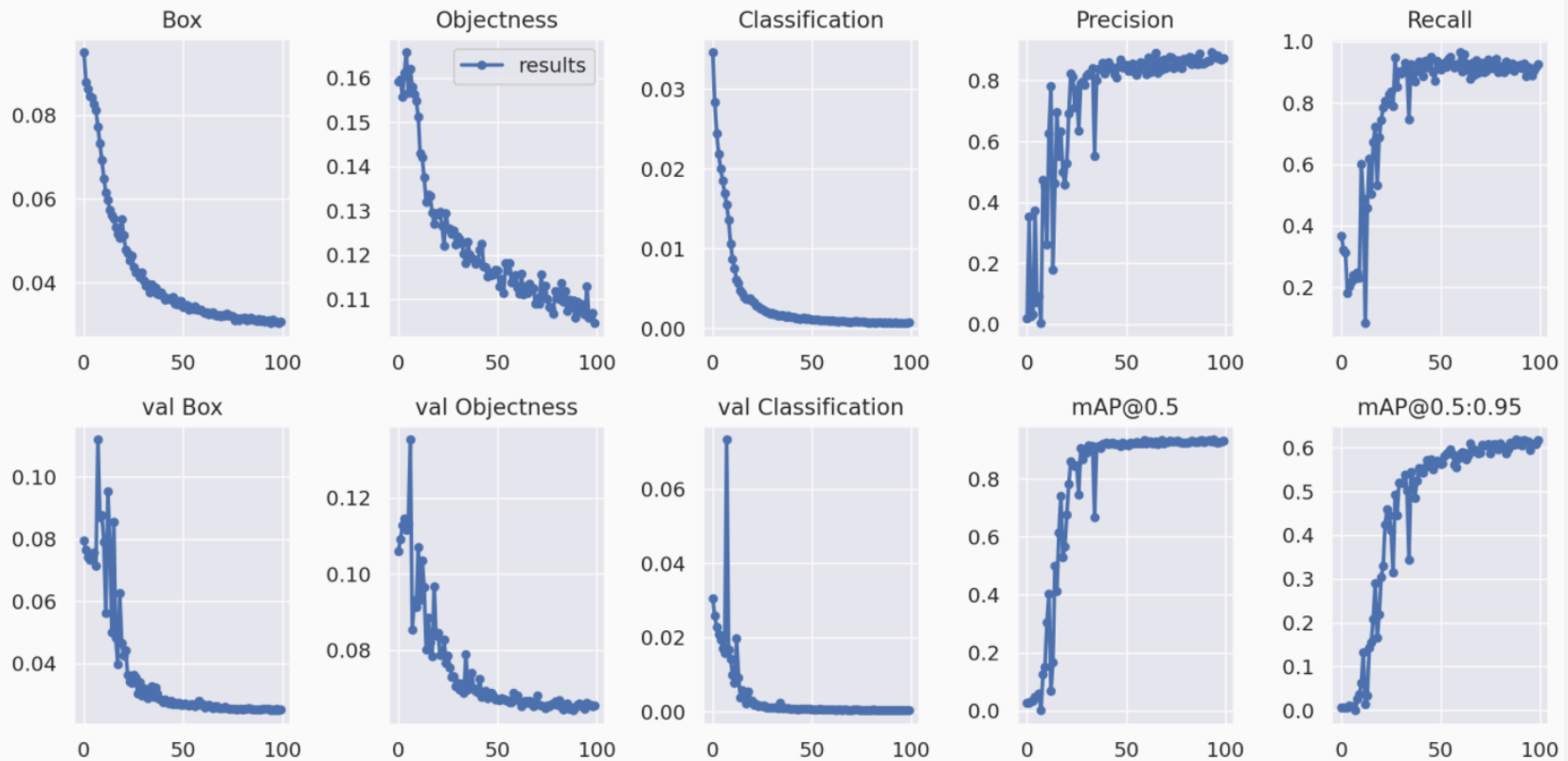


# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

## 12. The view of the training metrics and the model correctness:

```
[11] # we can also output some older school graphs if the tensor board isn't working for whatever reason...  
from utils.plots import plot_results # plot results.txt as results.png  
Image(filename='/content/yolov5/runs/train/yolov5s_results/results.png', width=1000) # view results.png
```

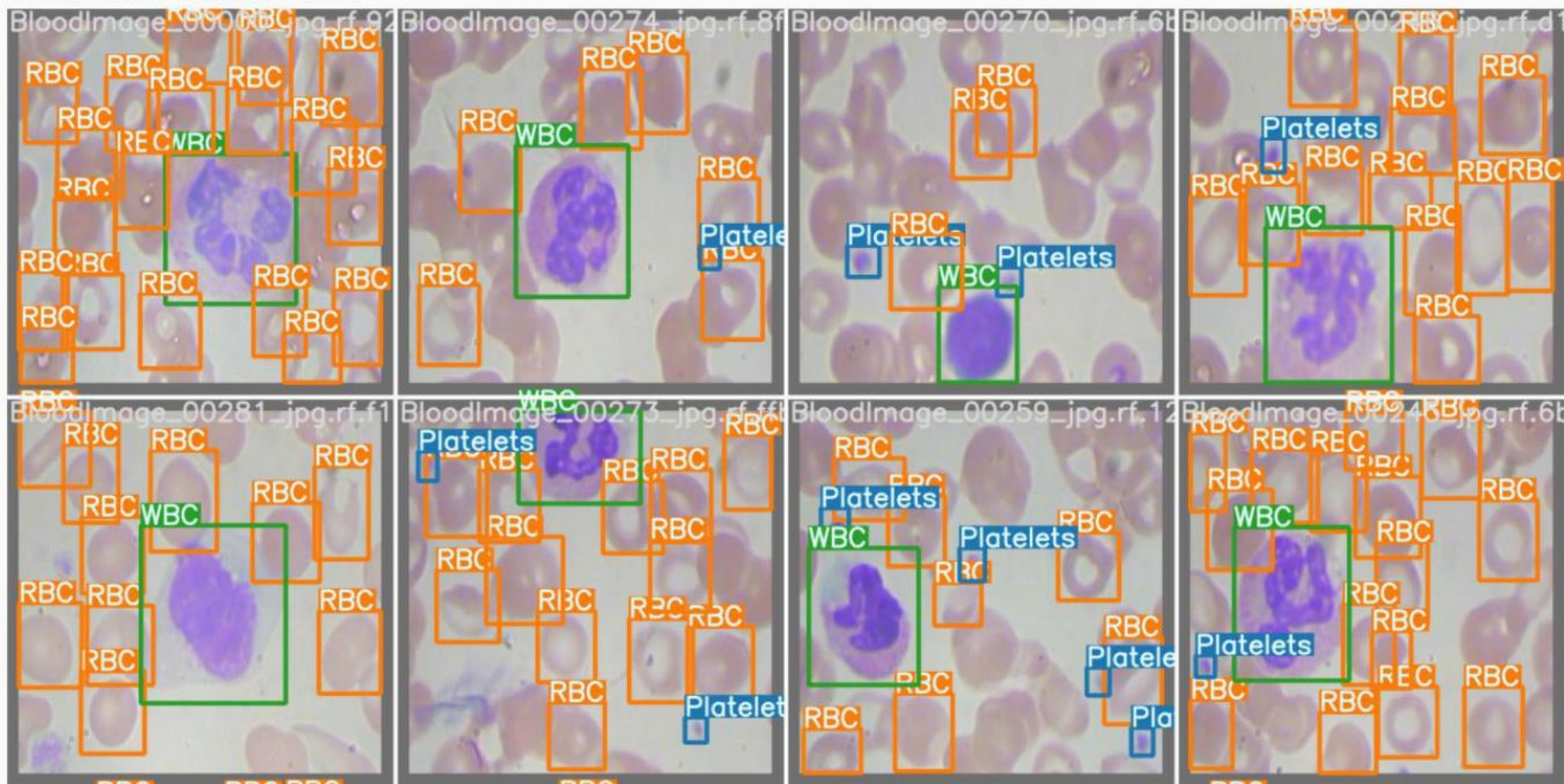


# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

## 13. Look at the ground truth BCCD training data:

GROUND TRUTH TRAINING DATA:



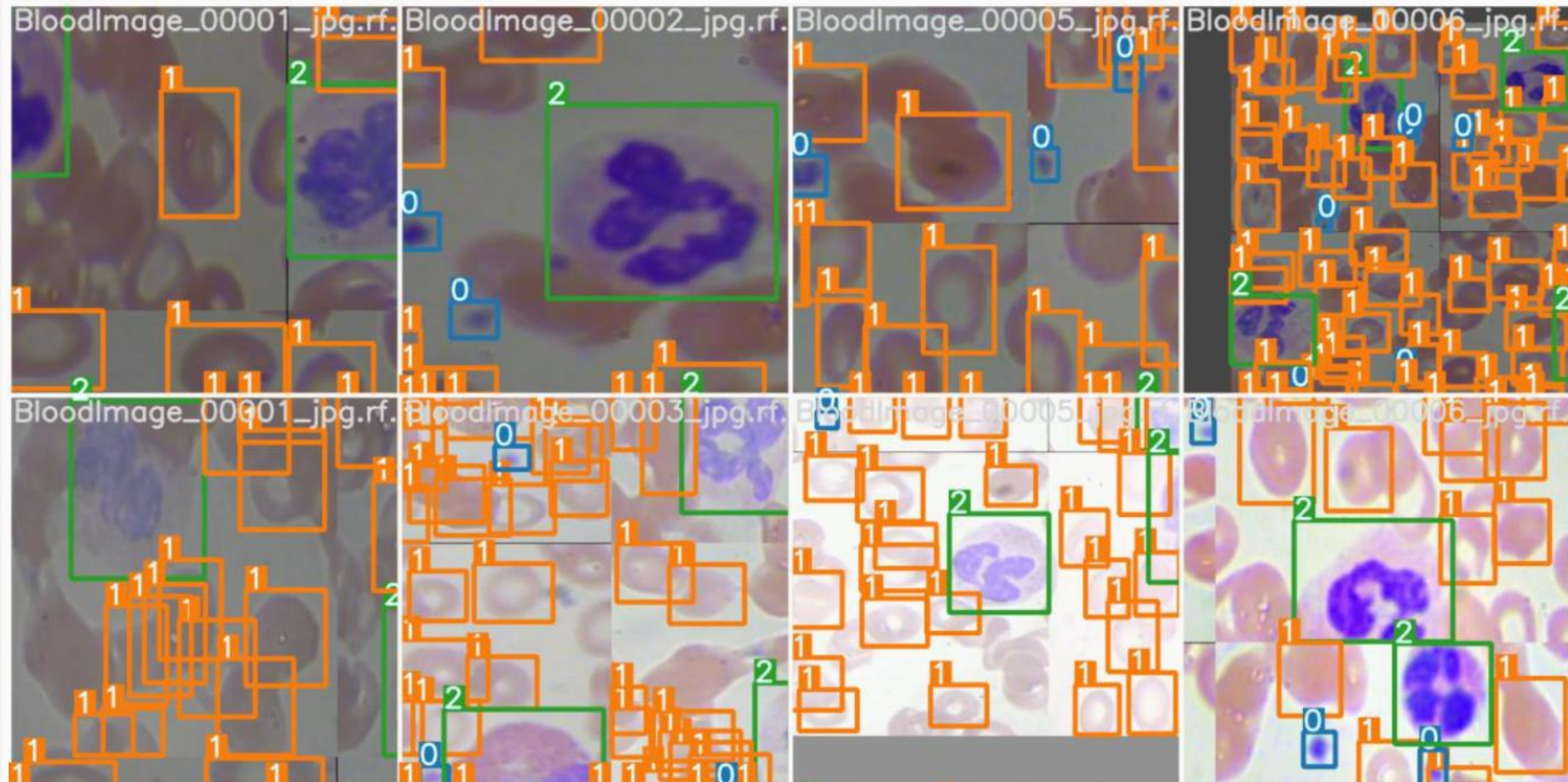


# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

## 14. Look at the augmented ground truth BCCD training data:

GROUND TRUTH AUGMENTED TRAINING DATA:



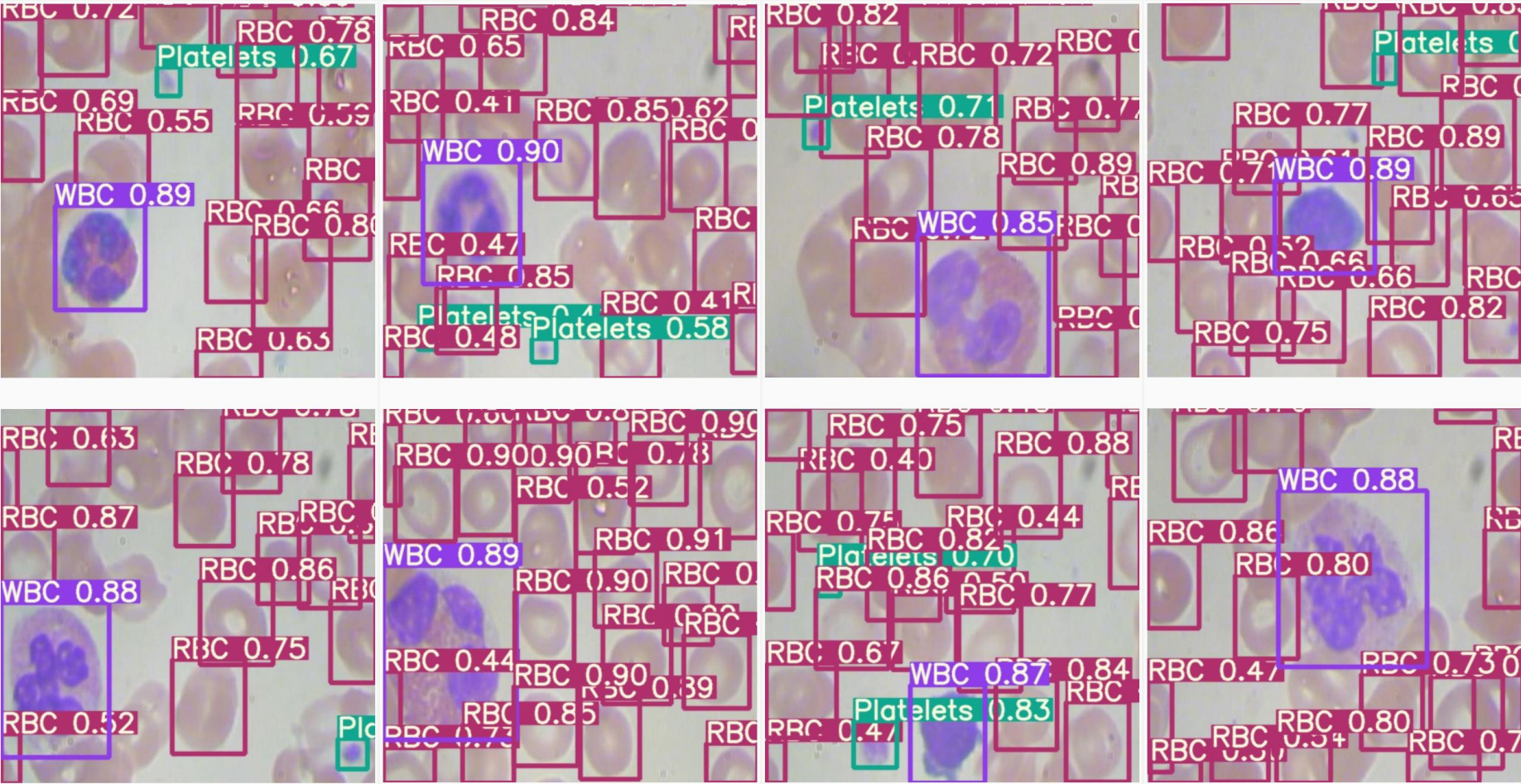




# Roboflow Detection Implementation

<https://www.youtube.com/watch?v=MdF6x6ZmLAY>

15. Finally, we can run inference and look at BCCD test images with detected objects:



Many cells were detected and classified correctly, but some of them are missing!



# BIBLIOGRAPY

1. [https://www.cs.toronto.edu/~tingwuwang/semantic\\_segmentation.pdf](https://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf)
2. <https://www.mathworks.com/help/vision/ug/gettin-g-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](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://arxiv.org/pdf/2004.10934.pdf>





# BIBLIOGRAPY

9. <https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-3/>
10. <https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-4/>
11. <https://blog.paperspace.com/how-to-implement-a-yolo-v3-object-detector-from-scratch-in-pytorch-part-5/>
12. <https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg>
13. <https://arxiv.org/pdf/1708.02002.pdf>
14. <https://www.youtube.com/watch?v=44tlnmmt3h0>
15. <https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4>
16. <https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109>





# BIBLIOGRAPY

17.<https://github.com/AlexeyAB/darknet>

18.[https://www.altexsoft.com/blog/data-science-artificial-intelligence-machine-learning-deep-learning-data-mining/?utm\\_source=newsletter&utm\\_medium=email&utm\\_campaign=NewsletterMay5&utm\\_term=N4&utm\\_content=b](https://www.altexsoft.com/blog/data-science-artificial-intelligence-machine-learning-deep-learning-data-mining/?utm_source=newsletter&utm_medium=email&utm_campaign=NewsletterMay5&utm_term=N4&utm_content=b)

19.A. Horzyk and E. Ergün, YOLOv3 Precision Improvement by the Weighted Centers of Confidence Selection, 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020, IEEE, Xplore, pp. 1-8, doi: [10.1109/IJCNN48605.2020.9206848](https://doi.org/10.1109/IJCNN48605.2020.9206848) - prezentacja - film

20.<https://www.cs.princeton.edu/courses/archive/spring18/cos598B/public/outline/Instance%20Segmentation.pdf>





# BIBLIOGRAPY

17. <https://github.com/ultralytics/yolov5>

18. <https://www.youtube.com/watch?v=MdF6x6ZmLAY>

19. <http://public.roboflow.ai>

20. <http://app.roboflow.ai>

21. <http://models.roboflow.ai>

22. [https://www.cs.toronto.edu/~tingwuwang/semantic\\_segmentation.pdf](https://www.cs.toronto.edu/~tingwuwang/semantic_segmentation.pdf)

23. <https://www.mathworks.com/help/vision/ug/getting-started-with-semantic-segmentation-using-deep-learning.html>

24. <https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef>

25. <https://www.jeremyjordan.me/semantic-segmentation/>





# BIBLIOGRAPHY



Home page for this course:

<http://home.agh.edu.pl/~horzyk/lectures/ahdydkbcidmb.php>

