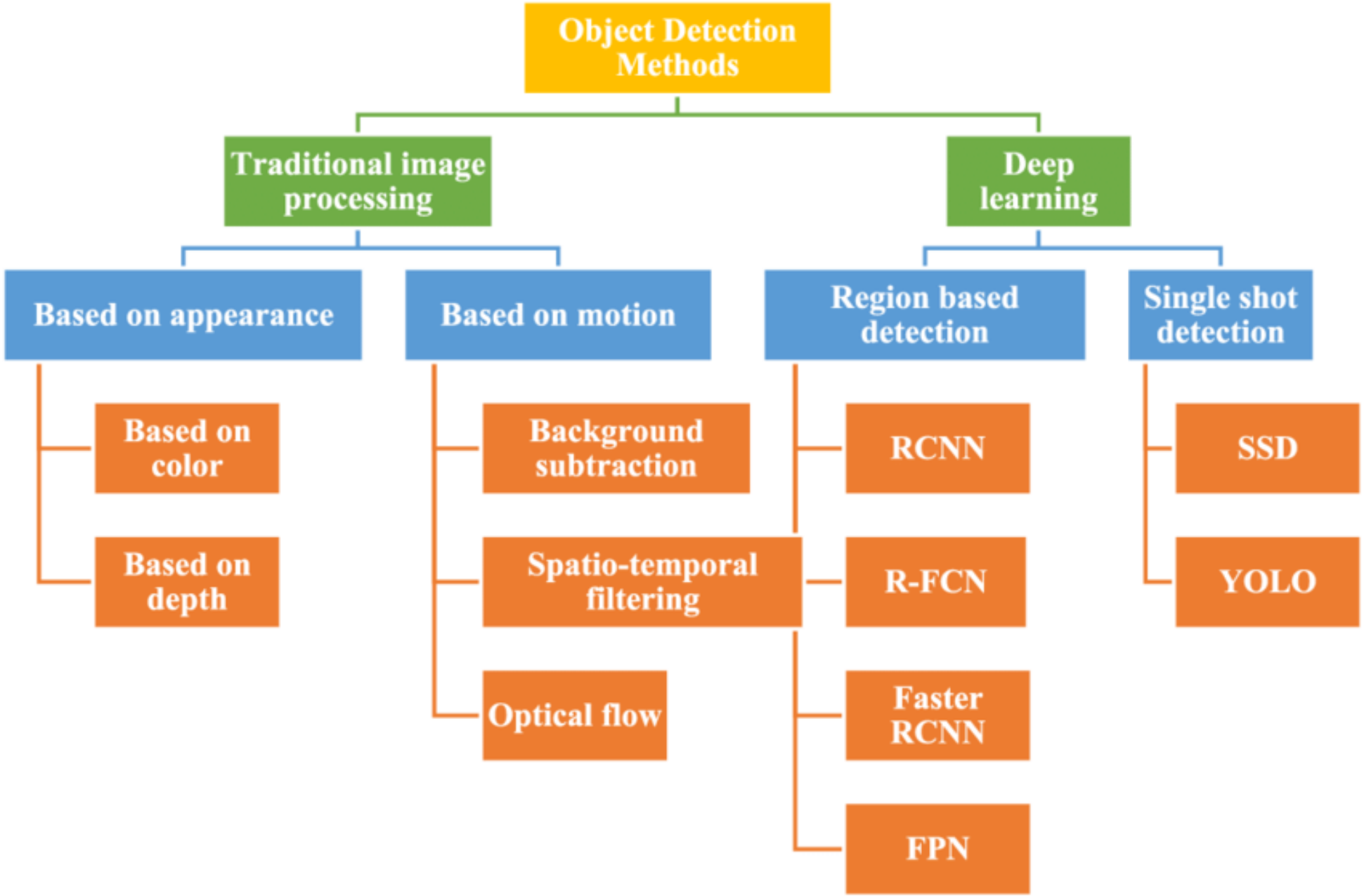
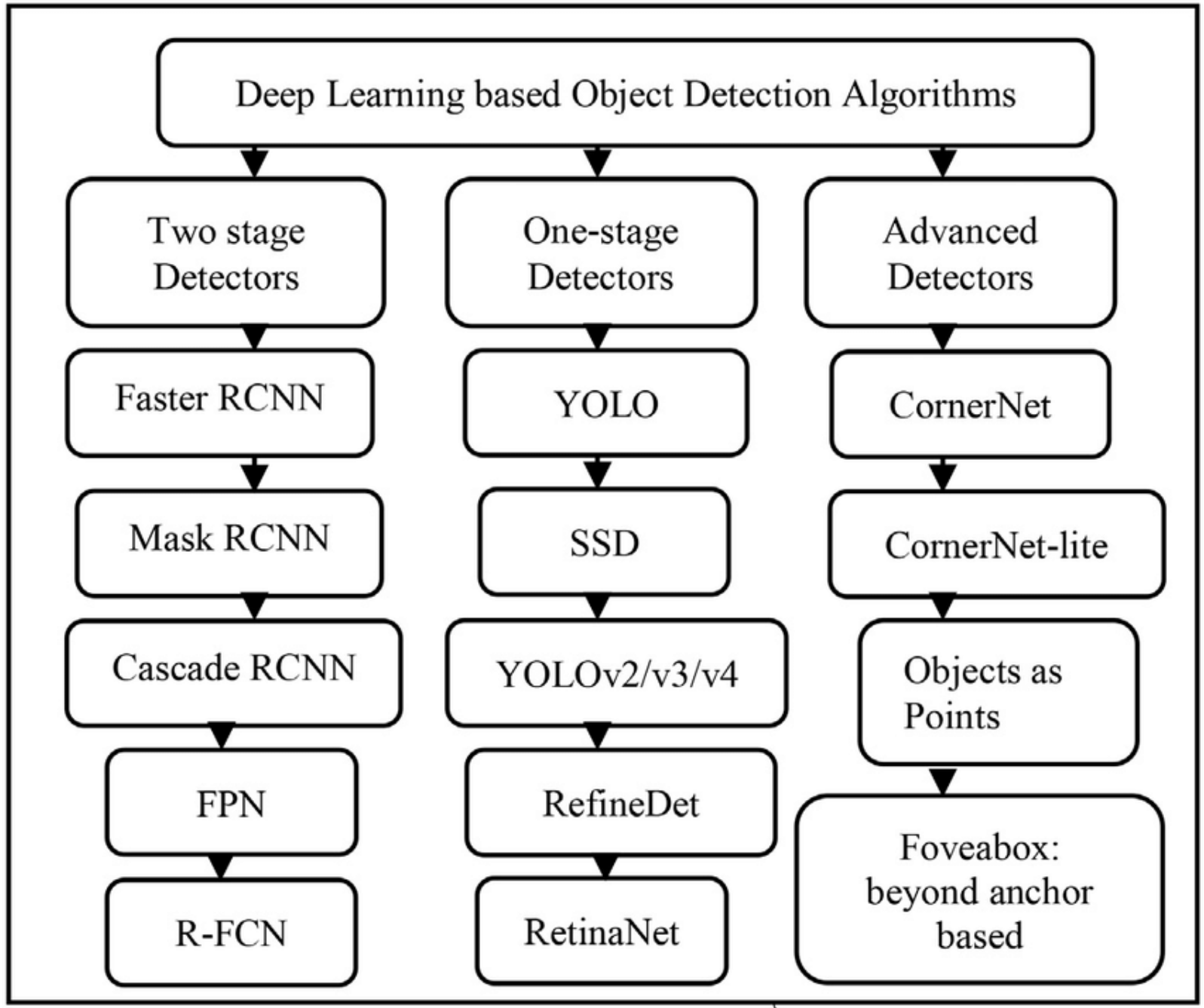
YOLO

YOU LOOK ONLY ONCE

Recap-RCNN-Limitations

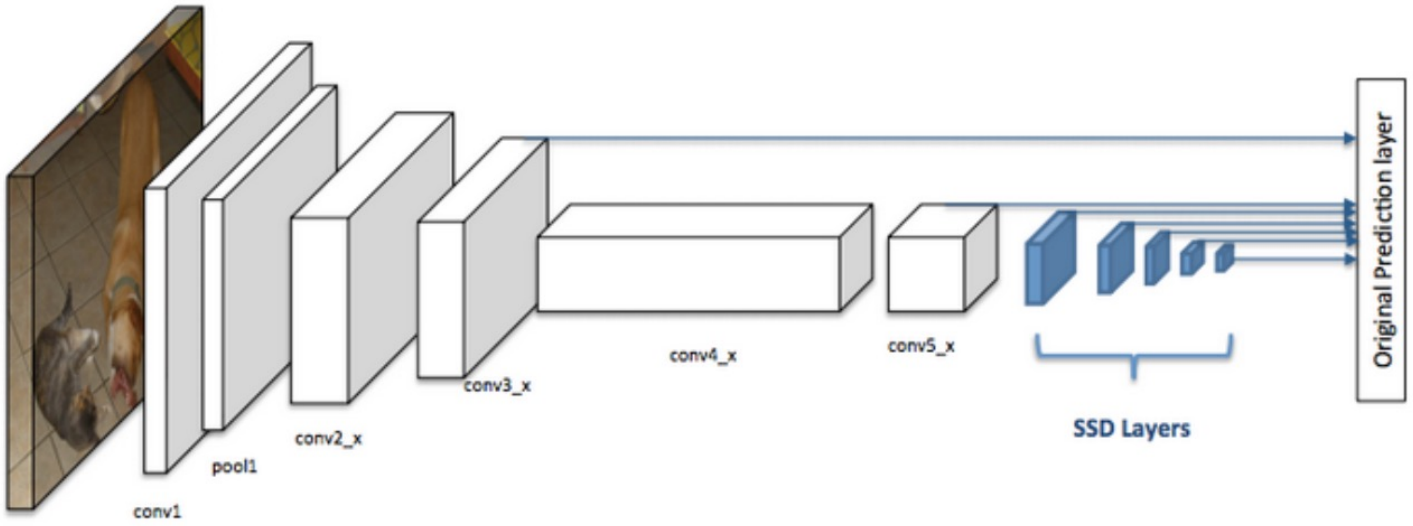
- high computational cost
- Inefficiency
- Limited generalization and
- Lack of context.
- Additionally, they require large labelled datasets, which can be costly and time-consuming to obtain.


Object Detection Methods
Traditional image processing
Deep learning
Based on appearance
Based on motion
Region based detection
Single shot detection
Based on color
Based on depth
Background subtraction
Spatio-temporal filtering
Optical flow
RCNN
R-FCN
Faster RCNN
FPN
SSD
YOLO


Deep Learning based Object Detection Algorithms
Two stage Detectors
One-stage Detectors
Advanced Detectors
Faster RCNN
YOLO
CornerNet
Mask RCNN
SSD
CornerNet-lite
Cascade RCNN
YOLOv2/v3/v4
Objects as Points
FPN
RefineDet
Foveabox: beyond anchor based
R-FCN
RetinaNet

SSD

- SSD has two components: a **backbone** model and **SSD head**.
- *Backbone* model usually is a pre-trained image classification network as a feature extractor.
- This is typically a network like ResNet trained on ImageNet from which the final fully connected classification layer has been removed.
- The first few layers (white boxes) are the backbone, the last few layers (blue boxes) represent the SSD head.

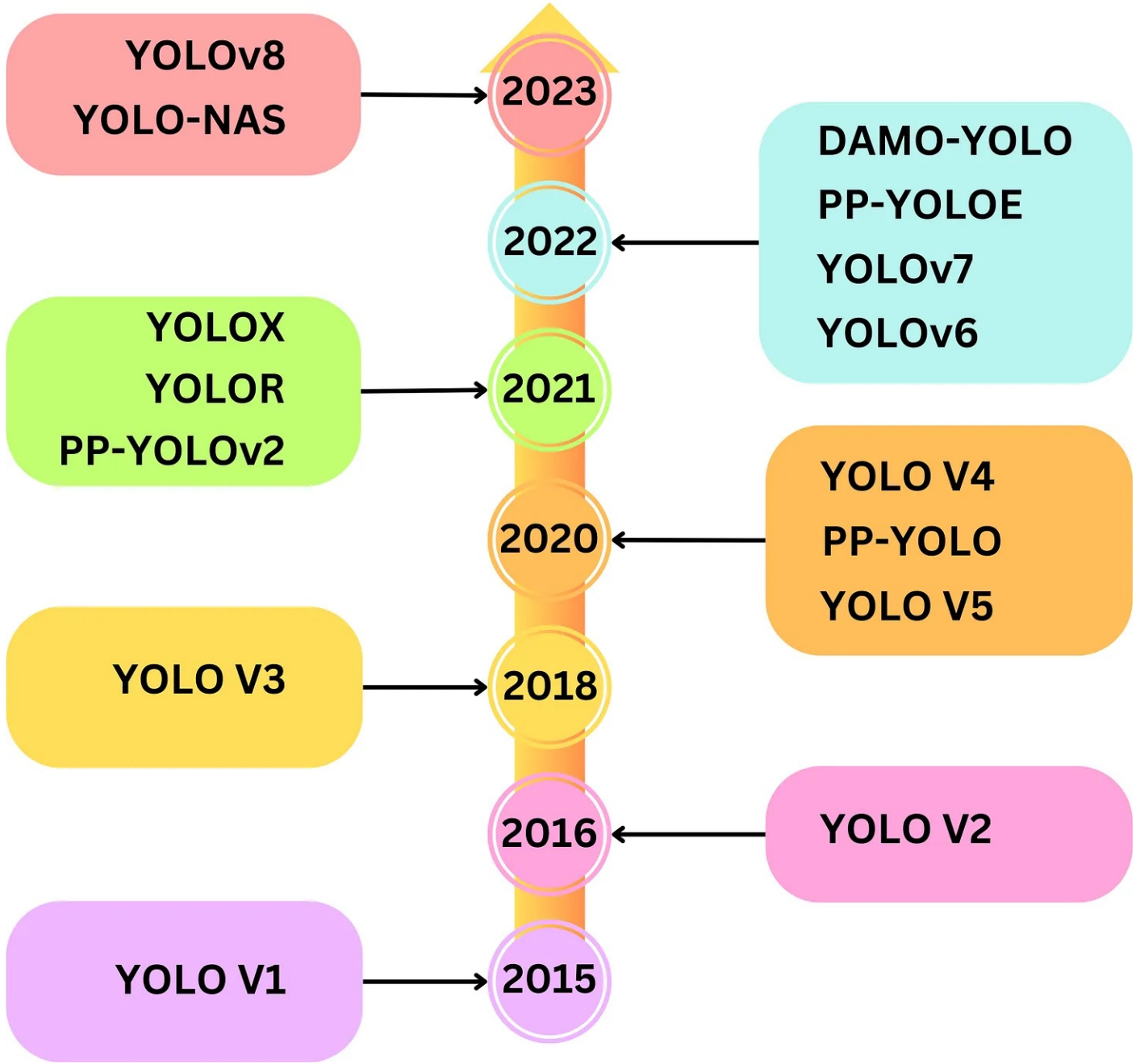


Previous Approaches

- A separate model for generating bounding boxes and for classification (more complicated model pipeline)
- Need to run classification many times (expensive computation)
- Looks at limited part of the image (lacks contextual information for detection)

YOLO algorithm

- A single neural network for localization and for classification (less complicated pipeline)
- Need to inference only once (efficient computation)
- Looks at the entire image each time leading to less false positives (has contextual information for detection)


YOLOv8
YOLO-NAS
2023
DAMO-YOLO
PP-YOLOE
YOLOv7
YOLOv6
2022
YOLOX
YOLOR
PP-YOLOv2
2021
YOLO V4
PP-YOLO
YOLO V5
2020
YOLO V3
2018
2016
YOLO V2
YOLO V1
2015

Overview of YOLO's Approach:

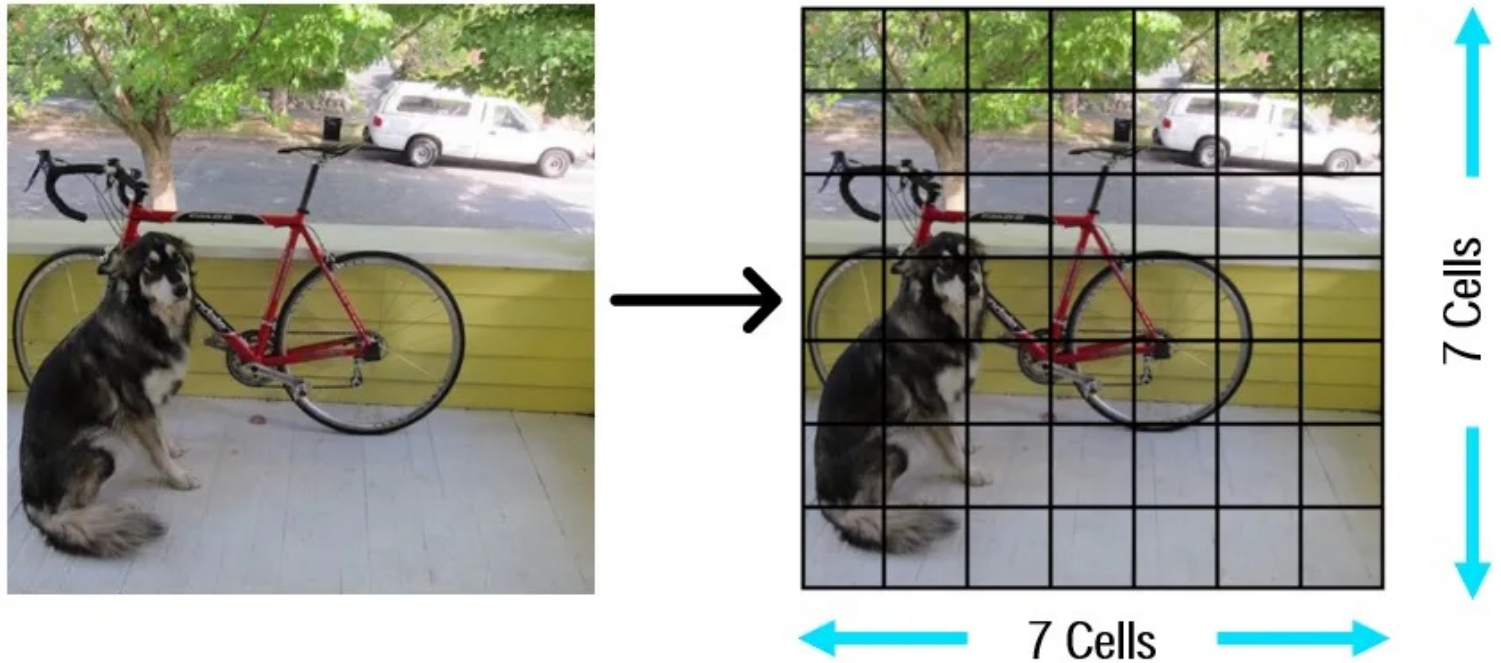
Unlike traditional object detection models (like R-CNN or Fast R-CNN), which use a two-stage process (first generating region proposals and then classifying them), **YOLO** processes the entire image in one step. This makes YOLO faster and more suitable for real-time applications.

In YOLO, the input image is divided into a grid. Each cell in this grid takes on the responsibility of scrutinizing a specific region of the image. This division allows YOLO to consider multiple locations simultaneously, transforming object detection into an efficient, parallel process.

Understanding Grid Cells:

Grid cells serve as the foundation of YOLO's spatial understanding. Each grid cell acts as an anchor point, and within it, YOLO seeks to identify objects. By systematically assessing each grid cell, YOLO can pinpoint objects' locations with precision.

$$\text{Grid Cells} = \frac{\text{Image Dimensions}}{\text{Grid Size}}$$

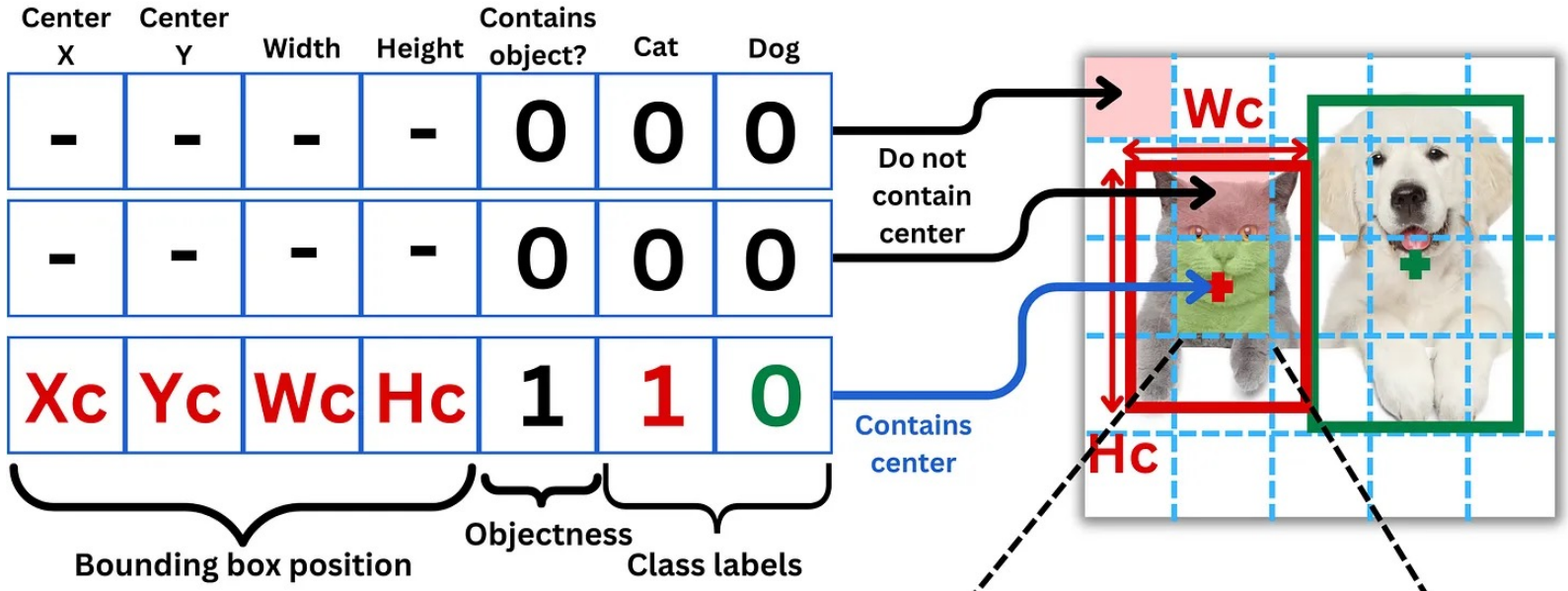


In YOLO v1 the grid size is 7 x 7.

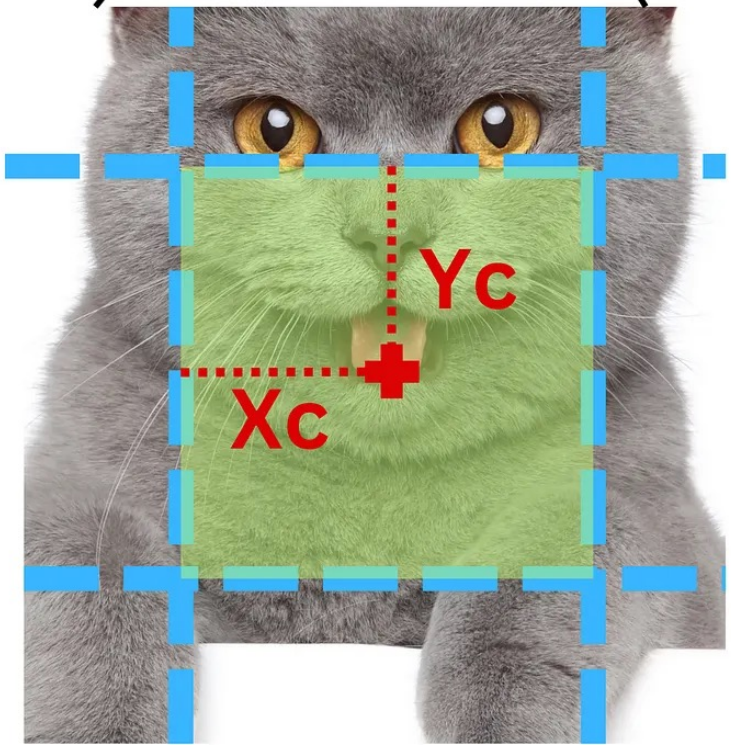
Each square is then analyzed independently, which simplifies the initial detection process as the algorithm focuses on smaller areas at a time rather than the whole image.

The model divides the image into a grid and for each grid cell, it predicts a fixed number of **bounding boxes** and class probabilities. Each box contains a set of values that describe its position, size, and confidence in containing an object of a certain class.

When it comes to labeling the data, a grid cell is labeled to contain an object only if the center of the bounding box is in it. If the grid cell contains a center, the "objectness" is labeled 1 and 0 otherwise. The model will try to predict the probability that a grid cell contains a center. If it contains a center, each class related to those centers will be labeled with 1s.



This means that each grid cell can predict multiple objects if the objects' centers fall inside the same cell.

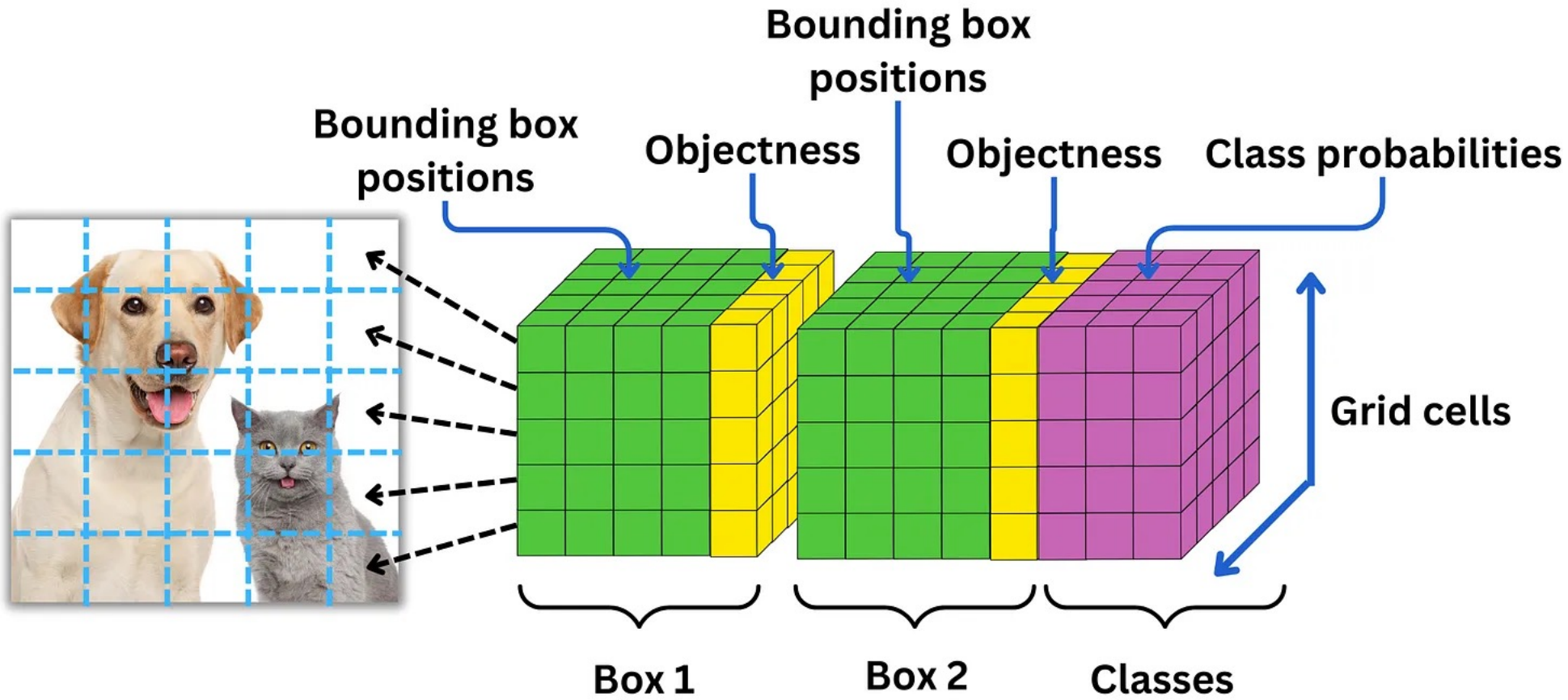


The prediction tensor

For each grid cell, the model will predict parameters for B boxes (for V1, we have B = 2). For each box, the model predicts ***x, y, w,*** and ***h*** along with the probability ***p*** that there is an object in that box. Along with the box's predictions, the model predicts a probability for each of the classes. For each box, we predict 5 parameters and for each class, we predict 1 parameter, so for each grid cell we predict ***B x 5 + C*** where ***C*** is the number of classes. The overall number of parameters is

$$S \times S \times (5B + C)$$

where ***S x S*** is the grid size.



If we call ***P(object)*** the probability there is an object in a box ***b*** and ***P(c|object)*** the probability that the object is of class ***c,*** then the score for the class ***c*** in the box ***b*** in simply

$$P(c) = P(c|\text{object}) \times P(\text{object})$$

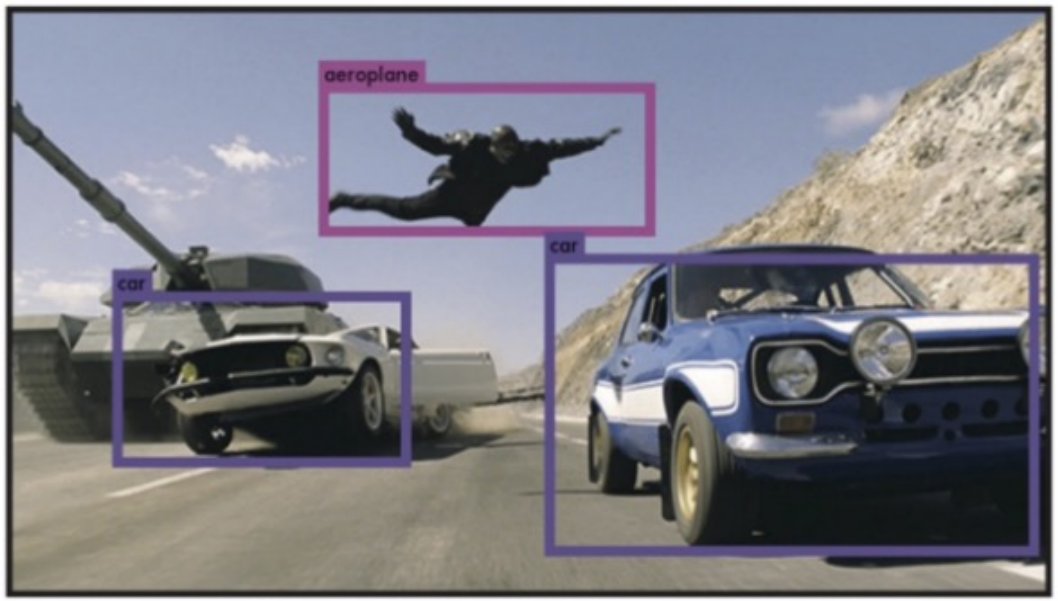
- Each **grid cell** is responsible for predicting **bounding boxes** and **class probabilities** for any object whose **center** lies within that grid cell.

- Each grid cell makes a prediction for **B** bounding boxes. For each bounding box, the model predicts:

    1. **(x, y)**: The center of the bounding box relative to the grid cell.
    2. **w, h**: The width and height of the bounding box, typically predicted relative to the entire image size.
    3. **Confidence score**: A score indicating how likely it is that the box contains an object (objectness score).
    4. **Class probabilities**: The likelihood that the object belongs to each of the predefined classes (e.g., car, person, dog, etc.).

Formal Problem Setting

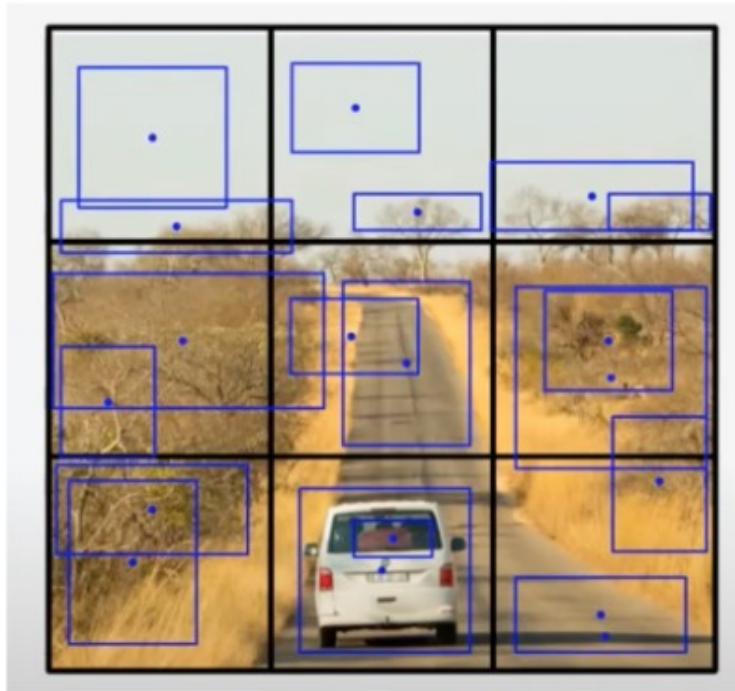
- Given an image generate bounding boxes, one for each detectable object in image
- For each bounding box, output 5 predictions: x, y, w, h, confidence. Also output class



- x, y (coordinates for center of bounding box)
- w,h (width and height)
- confidence (probability bounding box has object)
- class (classification of object in bounding box)

YOLO overview

- First, image is split into a SxS grid
- For each grid square, generate B bounding boxes
- For each bounding box, there are 5 predictions: x, y, w, h, confidence



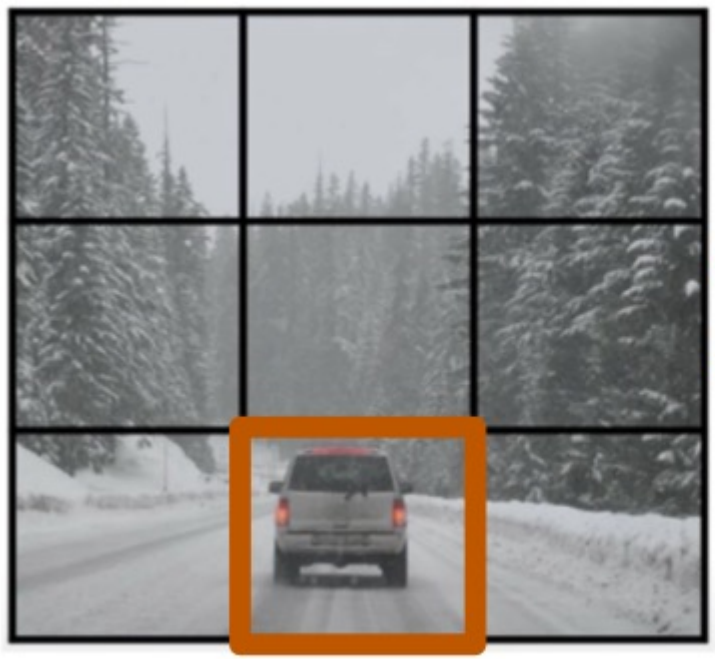
S = 3, B = 2

YOLO Training

- YOLO is a regression algorithm. What is X? What is Y?
- X is simple, just an image width (in pixels) * height (in pixels) * RGB values
- Y is a tensor of size S * S * (B * 5 + C)
- B*5 + C term represents the predictions + class predicted distribution for a grid block

For each grid block, we have a vector like this. For this example B is 2 and C is 2

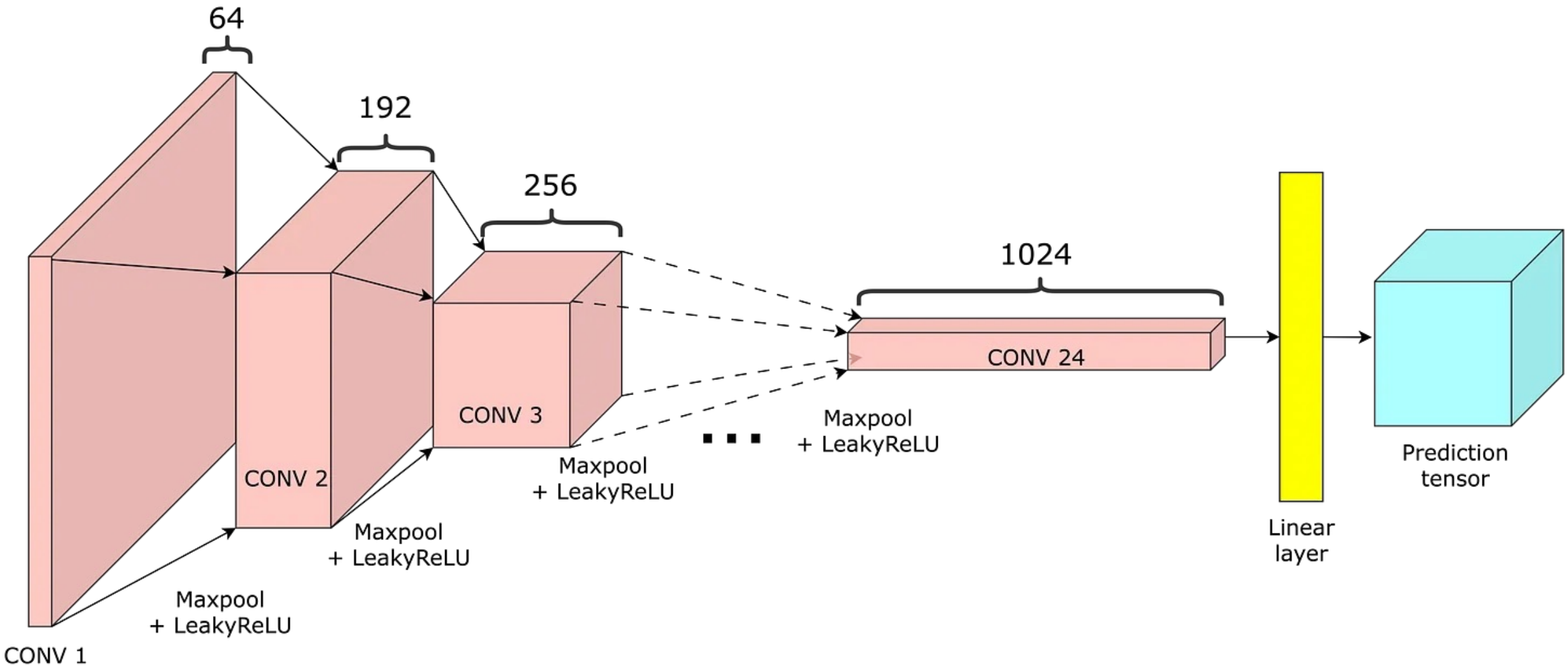
p_1
b_{x_1}
b_{y_1}
b_{h_1}
b_{w_1}
p_2
b_{x_2}
b_{y_2}
b_{h_2}
b_{w_2}
c_1
c_2



GT label example:

1
b_{x_1}
b_{y_1}
b_{h_1}
b_{w_1}
0
?
?
?
?
$c_1 = 1$
$c_2 = 0$

YOLO’s Core Architecture:



The architecture of YOLO v1 is a simple convolutional network with Maxpool layers and LeakyReLU activation functions followed by a linear layer and the prediction tensor.

To improve the speed of the network, they alternated convolutional layers with 3x3 kernel size and convolutional layers with 1x1 kernel size.

	Type	Filters	Size/Stride	Output
	Conv	64	$7 \times 7 / 2$	224×224
	Max Pool		$2 \times 2 / 2$	112×112
	Conv	192	$3 \times 3 / 1$	112×112
	Max Pool		$2 \times 2 / 2$	56×56
$1\times$	Conv	128	$1 \times 1 / 1$	56×56
	Conv	256	$3 \times 3 / 1$	56×56
	Conv	256	$1 \times 1 / 1$	56×56
	Conv	512	$3 \times 3 / 1$	56×56
	Max Pool		$2 \times 2 / 2$	28×28
$4\times$	Conv	256	$1 \times 1 / 1$	28×28
	Conv	512	$3 \times 3 / 1$	28×28
	Conv	512	$1 \times 1 / 1$	28×28
	Conv	1024	$3 \times 3 / 1$	28×28
	Max Pool		$2 \times 2 / 2$	14×14
$2\times$	Conv	512	$1 \times 1 / 1$	14×14
	Conv	1024	$3 \times 3 / 1$	14×14
	Conv	1024	$3 \times 3 / 1$	14×14
	Conv	1024	$3 \times 3 / 2$	7×7
	Conv	1024	$3 \times 3 / 1$	7×7
	Conv	1024	$3 \times 3 / 1$	7×7
	FC		4096	4096
	Dropout 0.5			4096
	FC		$7 \times 7 \times 30$	$7 \times 7 \times 30$

Input Layer:

- The input to YOLO is typically an image resized to a fixed size, such as **416x416** or **608x608**, depending on the version of YOLO (e.g., YOLOv3, YOLOv4).
- This image is passed through the network, which is typically based on a **convolutional neural network (CNN)**, similar to architectures like **VGG** or **ResNet** but designed to efficiently handle object detection tasks.

Convolutional Backbone (Feature Extraction):

The CNN backbone is used to extract feature maps from the input image.

- The CNN backbone consists of several **convolutional layers**, **batch normalization**, and **activation functions** (typically **Leaky ReLU**) to learn hierarchical features at various scales (edges, textures, shapes, etc.).
- The output of the CNN backbone is a **feature map** that contains the learned representations of the image at multiple levels of abstraction.

Bounding Box Prediction (For Each Grid Cell):

For each grid cell, the network predicts **B bounding boxes**

Final Output Layer:

- The final output of YOLO is a tensor of dimensions $S \times S \times (B \times 5 + C)$, where:
  - $S \times S$: The number of grid cells.
  - $B \times 5$: For each grid cell, there are B bounding boxes, each with 5 predictions (center x, center y, width, height, and objectness score).
  - C: The number of class probabilities for each bounding box.
- The output tensor for each grid cell contains predictions for **bounding boxes, objectness scores**, and **class probabilities**.

Anchor Boxes and Bounding Box Predictions

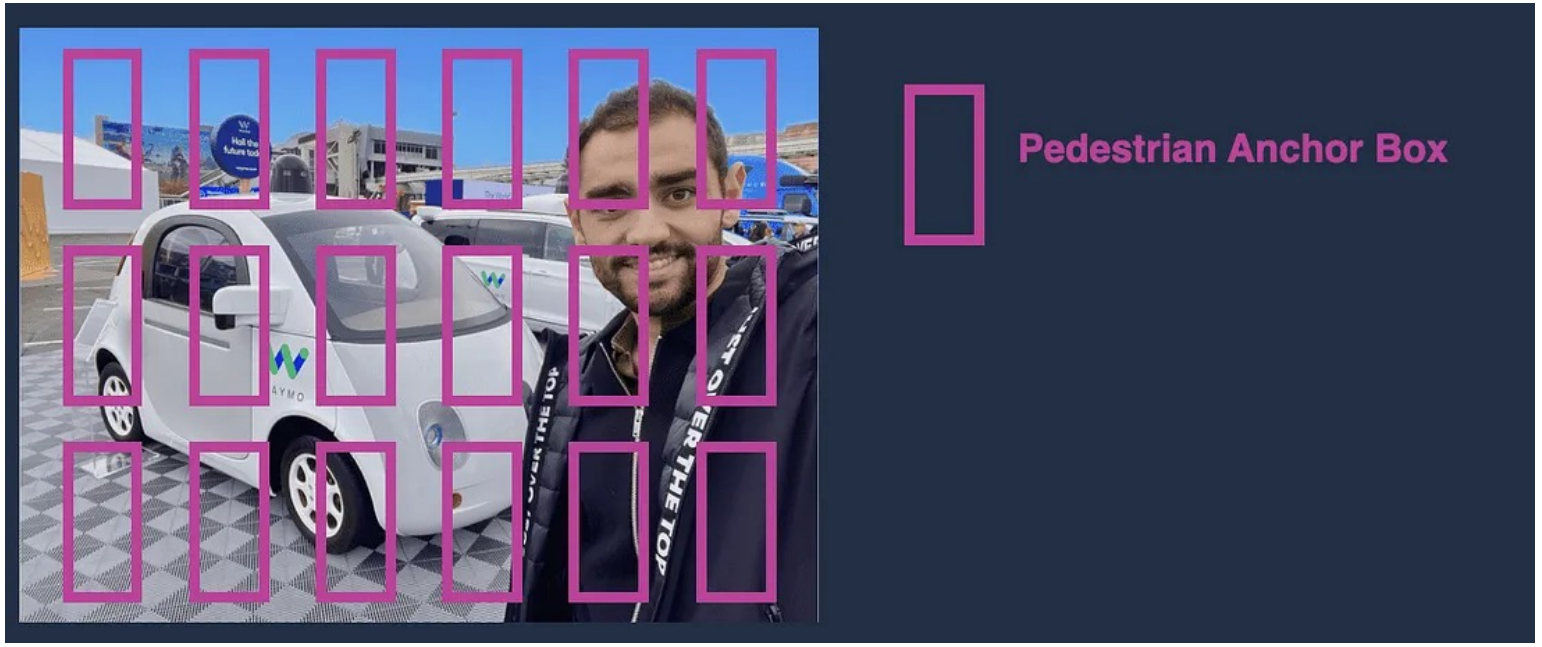
In YOLO, **anchor boxes** are predefined bounding boxes with fixed aspect ratios and sizes, which the model uses as **reference points** for its predictions. These anchor boxes are particularly useful for detecting objects of various sizes and aspect ratios.

Anchor Boxes

Anchor boxes are predefined bounding boxes that serve as reference points for YOLO.

> An Anchor Box is not a Bounding Box!

First, we will take one anchor box, and put it everywhere on the image, just like for sliding windows:



But notice how none of these anchor boxes can be our final bounding box.

When Are Anchor Boxes Generated?

Anchor boxes are **generated before training** begins and are not learned by the model during training. They are predefined based on the dataset and the specific task at hand. They do not change during training, but the model learns to predict the adjustments needed to match ground truth bounding boxes relative to these anchors.

The anchor boxes are chosen based on **statistics** from the dataset—
particularly the **aspect ratios** and **sizes** of the ground truth bounding boxes.

How Are Anchor Boxes Generated?

Anchor boxes are generated using a process that typically involves analyzing the **aspect ratios** and **sizes** of the **bounding boxes** in the **training dataset**.

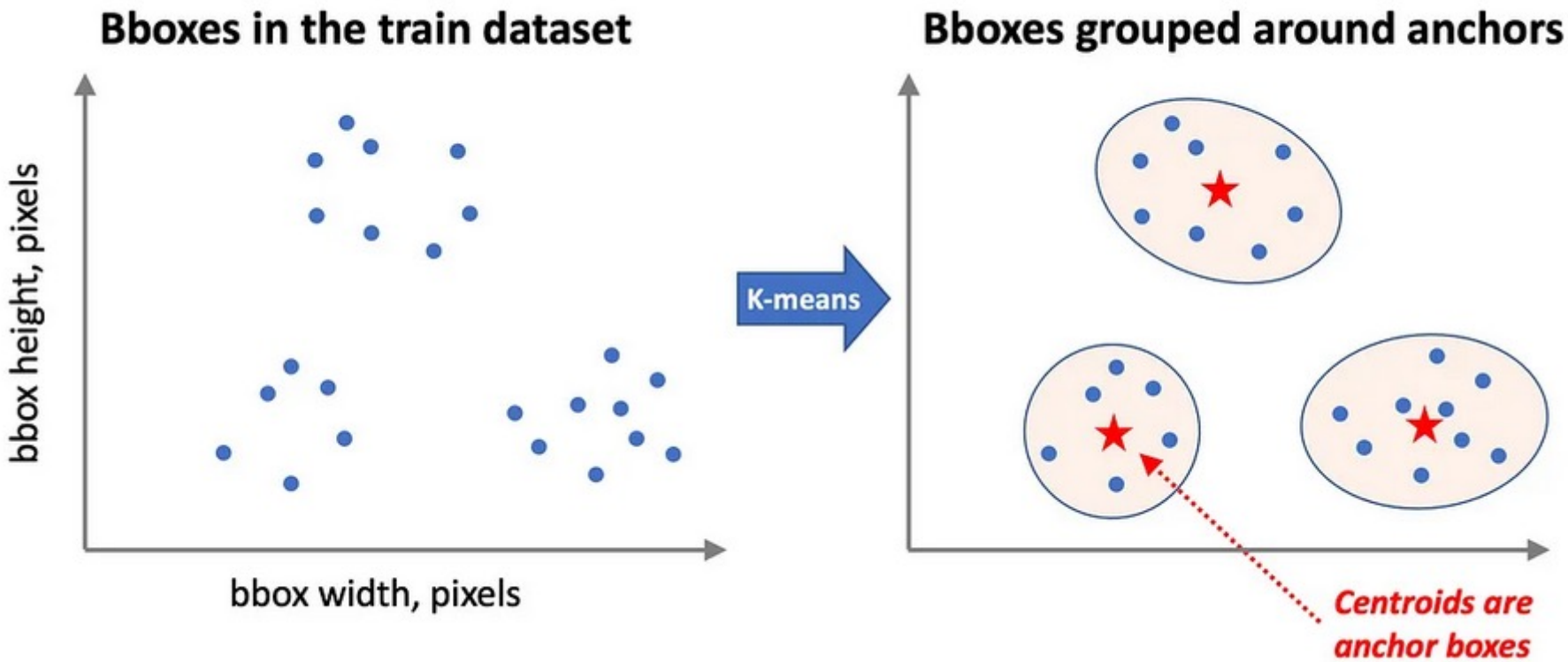
Step 1: Analyze the Dataset

1. **Collect Bounding Box Sizes**: The first step is to gather all the bounding boxes from the dataset. Each bounding box has a width **w** and height **h**.
    - For example, in an object detection task involving the COCO dataset, the bounding boxes for various objects like cars, people, dogs, etc., have different sizes and aspect ratios.
2. **Calculate Aspect Ratios**: For each bounding box in the dataset, calculate its **aspect ratio** (width/height). For example, a bounding box with width 100px and height 50px has an aspect ratio of 2:1.

Step 2: Use Clustering (e.g., K-Means)

3. **Clustering Bounding Box Sizes**: The most common method for generating anchor boxes is **K-means clustering**, a machine learning technique. The idea is to cluster the bounding box dimensions into **k clusters** based on their width and height.

4. **Selecting Anchor Box Sizes**: After clustering, the centroids of each cluster (the most typical width and height values) become the **predefined anchor boxes**. These anchor boxes are usually chosen as pairs of width and height values.

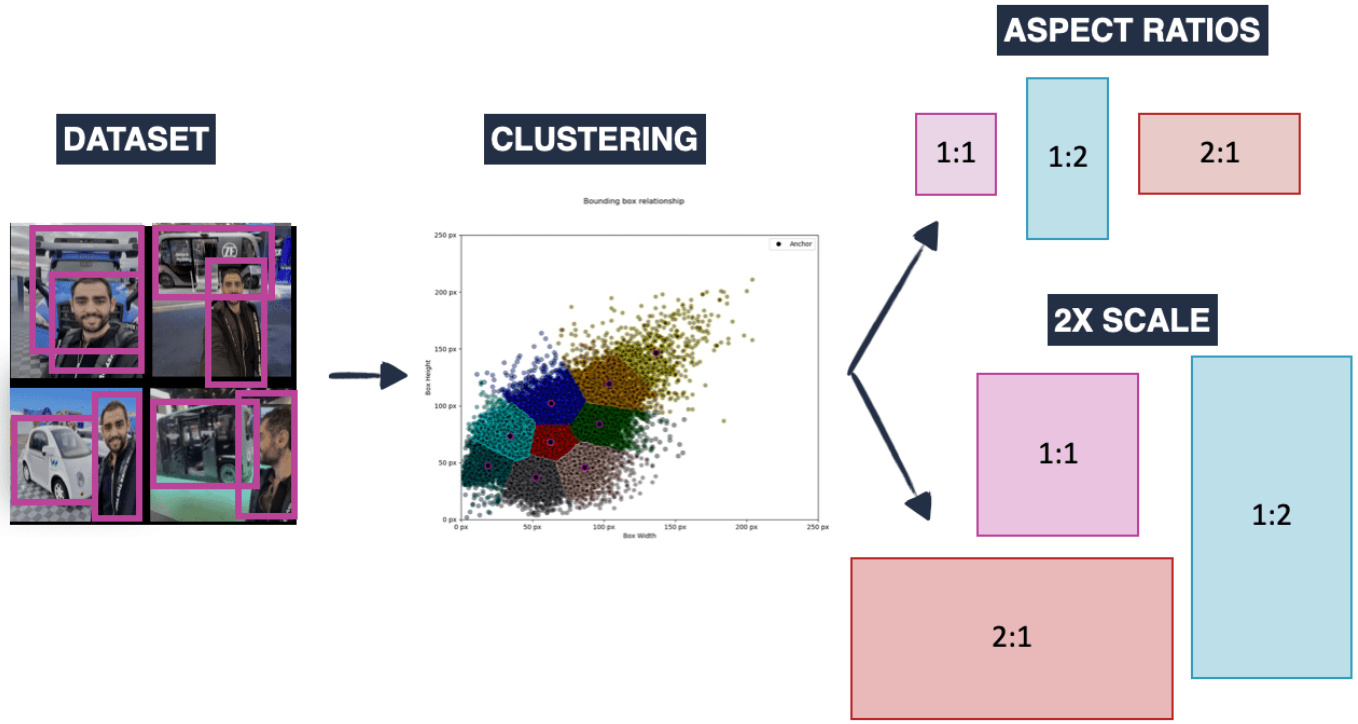

Bboxes in the train dataset
bbox height, pixels
bbox width, pixels
K-means
Bboxes grouped around anchors
Centroids are
anchor boxes

Step 3: Assign Anchor Boxes to Grid Cells

5. **Assigning Anchors to Grid Cells**: Once you have your anchor boxes, the next step is to assign them to each grid cell. During training, for each grid cell in the output grid (e.g., 13x13, 19x19), the model will predict **offsets** relative to the closest anchor box.
    - For each anchor box, the network learns to predict the **center (x, y)** of the bounding box, the **width (w)**, and **height (h)**, as well as the **confidence score** and **class probabilities**.
    - Each grid cell will predict a set of bounding boxes, each corresponding to one of the predefined anchor boxes. For example, if there are 3 anchor boxes per grid cell, each grid cell will predict 3 bounding boxes with different sizes and aspect ratios.

Our anchor boxes will be applied on features, but we still need to "assume" the shape of our boxes.

How? You usually do some of it manually, and some of it by looking at your dataset and using a clustering algorithm like K-Means.



The process of anchor box definition is usually done with clustering

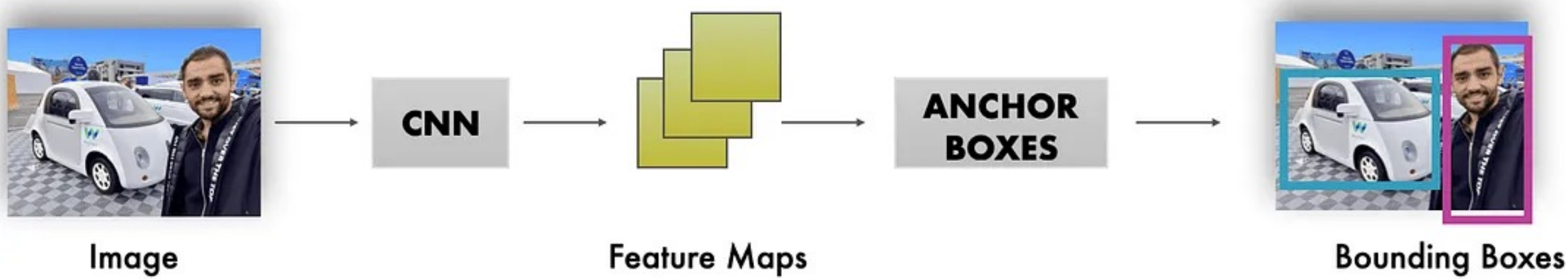
Once you have these, you'll generate multiple anchor boxes of a certain size (16,32), (32,32), (32, 16), and then have variations of these both in aspect ratio and scale (2x, 3x, 3x, ...). You end up with lots of possible boxes!

Example, we can decide that we want an anchor box of size **16x16**, with two types of aspect ratio: **1:1, 1:2, and 2:1**, and with 2 types of scale: x**1, and x2.** Which means we end up with 6 anchor boxes per region:

- 16x16 - the 1:1 box of scale 1
- 16x32 - The 1:2 box of scale 1
- 32x16 - The 2:1 box of scale 1
- 32x32 - The 1:1 box of scale 2
- 32x64 - The 1:2 box of scale 2
- 64x32 - The 2:1 box of scale 2

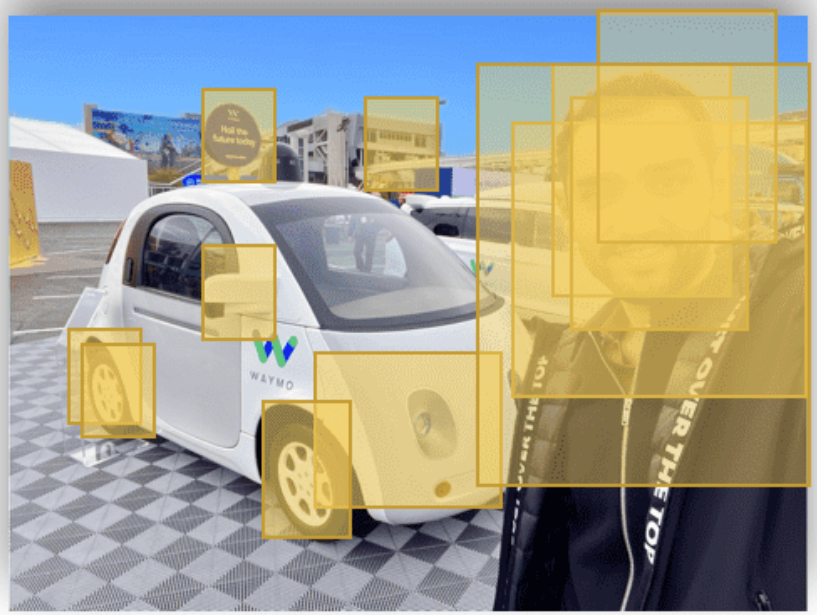
Aspect ratio and zoom level

- The SSD architecture allows pre-defined aspect ratios of the anchor boxes to account for this.
- The ratios parameter can be used to specify the different aspect ratios of the anchor boxes associates with each grid cell at each zoom/scale level.
- The zooms parameter is used to specify how much the anchor boxes need to be scaled up or down with respect to each grid cell.

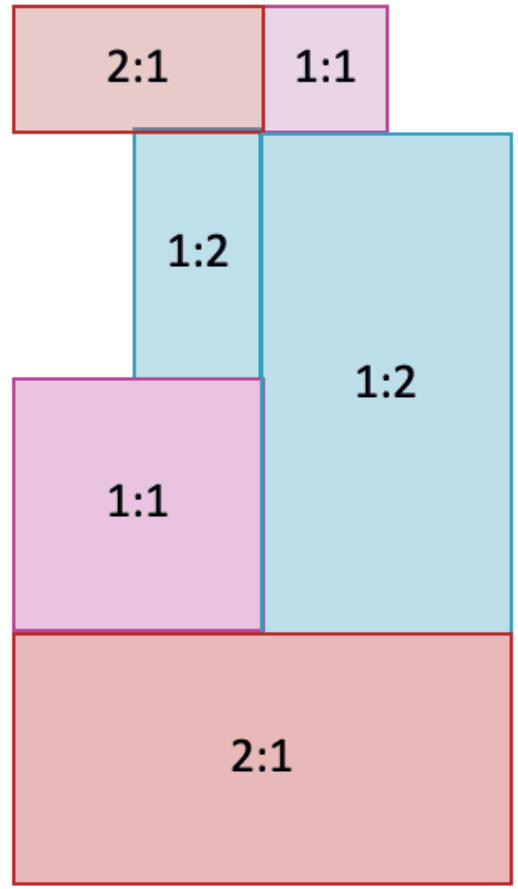


1. Anchor Boxes are used on top of feature maps, not on images
2. Anchor Boxes are used to generate bounding boxes, but they aren't the bounding boxes[1]

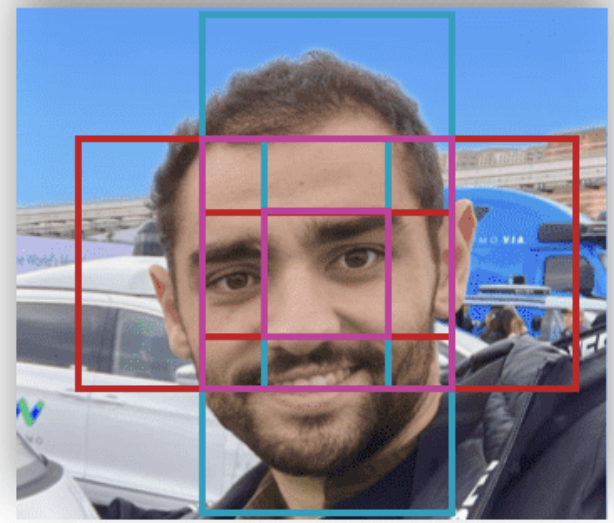
REGIONS



ANCHOR BOXES



ANCHOR BOXES PROPOSED PER REGION

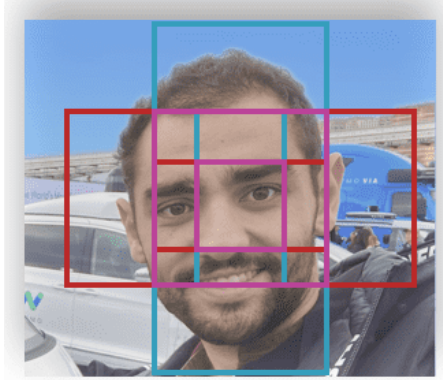


The good thing about region proposals is that you don't have to generate anchor boxes for every possible region, like in a sliding window approach

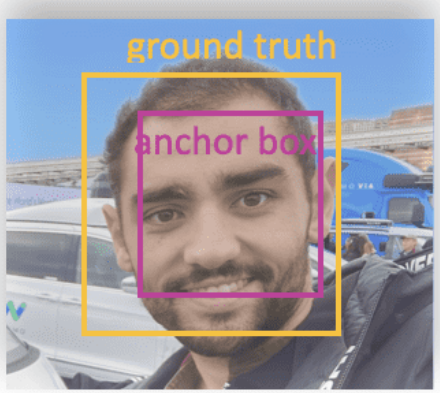
From Anchor Boxes to Bounding Boxes with Anchor Box Offsets

Still on face detection, we've generated lots of boxes. Now let's only keep the most likely 1:1 anchor box. We're going to predict our anchor boxes, then keep the one that has the highest Intersection over union (IOU) with the ground truth box, and finally refine it.

ANCHOR BOXES

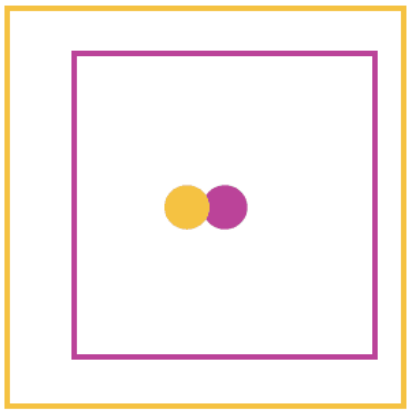


KEPT BOX



I can't stand seeing my face this big this much, so let's soon switch to another example

REFINEMENT



WHAT THE NETWORK PREDICTS

center: Move 5 Pixels left

height: +31 pixels

width: +26 pixels

A bounding box is a refined anchor box!

From Anchor Boxes to Bounding Boxes with Anchor Box Offsets

- Many anchor boxes are seen, How to choose the one which is to be perfect?
- In many cases, not even ONE anchor box becomes the ground truth bounding box.
- predict our anchor boxes, then keep the one that has the highest Intersection over union (IOU) with the ground truth box, and finally refine it.
- To finalize one, look the center of these boxes, and ground truth bounding box, analyze the difference between the center and the ground truth bounding boxes.
- Displacement between the anchor boxes needed to get from the original anchor box position to the ground truth.
- It learns the anchor box offsets!
- We take the anchor boxes centered around the potential final bounding box, and we end up refining the process by learning a shift.

See what the network learns: a displacement between the anchor boxes needed to get from the original anchor box position to the ground truth. It learns the anchor box offsets! We take the anchor boxes centered around the potential final bounding box, and we end up refining the process by learning a shift.

YOLO: YOLO is a grid-based algorithm. It does use anchor boxes, but it doesn't generate millions of them. Each grid cell is responsible to predict a few anchor boxes. So it's fewer anchor boxes, and in some versions of YOLO, it's no anchor box at all!

Limitations:

- Anchor boxes have been an amazing boost in object detection; yet they aren't perfect.
- Another issue is the number of boxes needed to get one final box.
- If we want to detect small objects, we'll need to generate thousands of small anchor boxes we'll probably never use, just for this one time a small object appears.

Center (x, y) Refinement:

For each anchor box, the **center coordinates** of the predicted bounding box are refined by adjusting the anchor box's center with the predicted offset:

- $\hat{x} = \sigma(tx) + cx$
- $\hat{y} = \sigma(ty) + cy$

Where:

- **cx, cy** are the coordinates of the top-left corner of the grid cell (i.e., the center of the grid cell).
- **tx, ty** are the predicted offsets for the bounding box center.
- **σ()** is the **sigmoid** function applied to ensure that the values are between 0 and 1 (in the normalized space).

Width (w) and Height (h) Refinement:

The **width** and **height** of the bounding box are refined by scaling the predicted values relative to the anchor box dimensions:

- $\hat{w} = pw * e^{tw}$
- $\hat{h} = ph * e^{th}$

Where:

- **pw, ph** are the width and height of the anchor box.
- **tw, th** are the predicted **log-space** offsets (logarithmic transformations to ensure scale invariance).
- **e^tw** and **e^th** scale the width and height by the predicted scaling factors.

Example Anchor Box: A1=(2,3)) (width = 2, height = 3)

Grid Cell Top-Left/Center coordinates:
(cx,cy)=(1,1)

Predictions: tx=0.5, ty=0.4, tw=0.1, th=−0.1

Calculating the actual bounding box:

Center Coordinates: $b_x = \sigma(0.5) + 1 = 1.62$
$b_y = \sigma(0.4) + 1 = 1.60$

Dimensions: $b_w = 2 \times e^{0.1} = 2.22$
$b_h = 3 \times e^{-0.1} = 2.71$

Post-Processing: Non-Maximum Suppression (NMS)

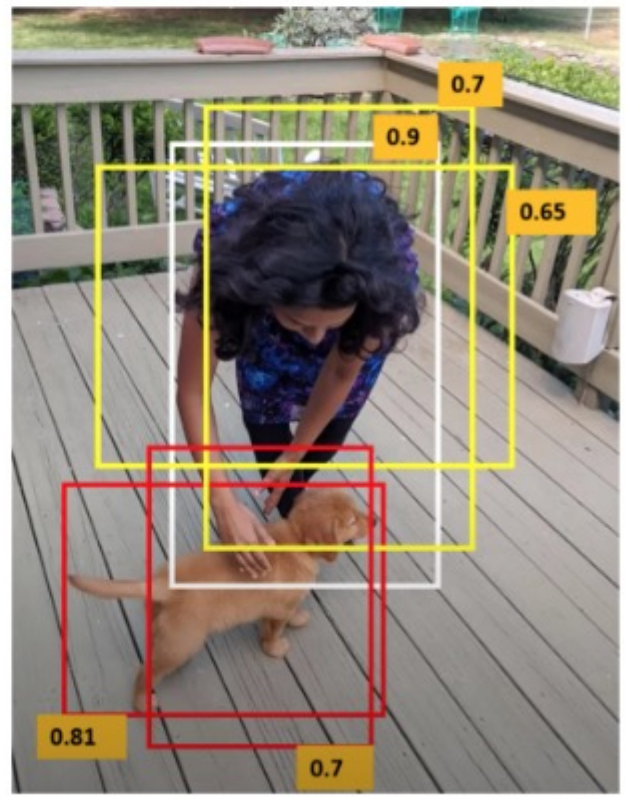
After YOLO predicts a set of bounding boxes, we typically apply **Non-Maximum Suppression (NMS)** to remove redundant or overlapping bounding boxes and retain only the best predictions. NMS works by:

1. Sorting all predicted bounding boxes by their **confidence score**.
2. For each bounding box with the highest confidence score, comparing it to other bounding boxes that have significant overlap (measured by **IoU**).
3. Removing all boxes whose IoU with the current box exceeds a predefined threshold (usually 0.5 or 0.6).

This helps ensure that only the most confident, non-overlapping bounding boxes remain.

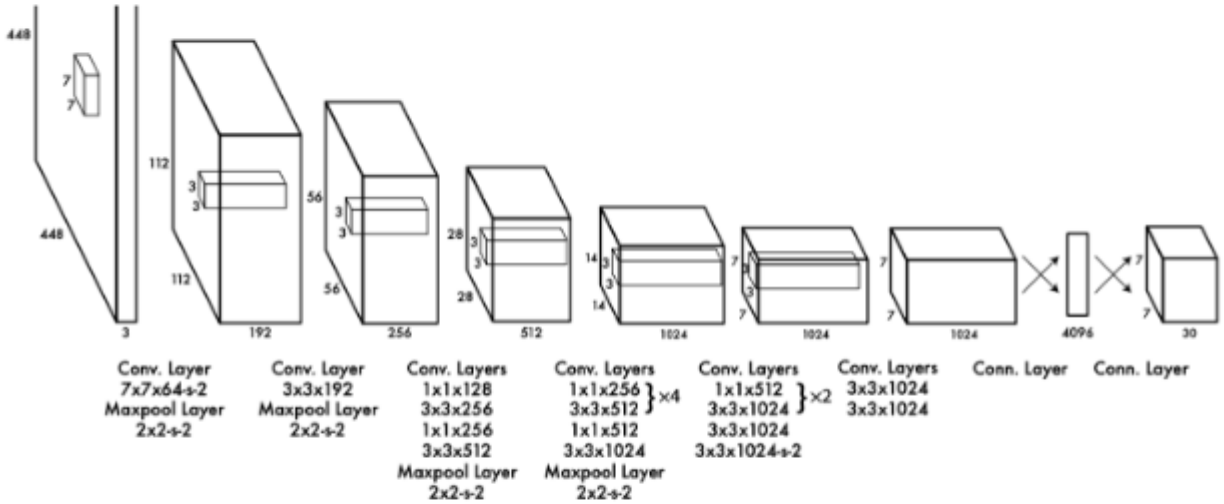
Non-maximal suppression

- Most of the time objects fall in one grid, however it is still possible to get redundant boxes (rare case as object must be close to multiple grid cells for this to happen)
- Discard bounding box with high overlap (keeping the bounding box with highest confidence)
- Adds 2-3% on final mAP score



YOLO Architecture

- Now that we know the input and output, we can discuss the model
- We are given 448 by 448 by 3 as our input.
- Implementation uses 7 convolution layers
- Paper parameters: S = 7, B = 2, C = 20
- Output is S*S*(5B+C) = 7*7*(5*2+20) = 7*7*30



The YOLO algorithm employs a single Convolutional Neural Network (CNN) that divides the image into a grid. Each cell in the grid predicts a certain number of bounding boxes. Along with each bounding box, the cell also predicts a class probability, which indicates the likelihood of a specific object being present in the box.

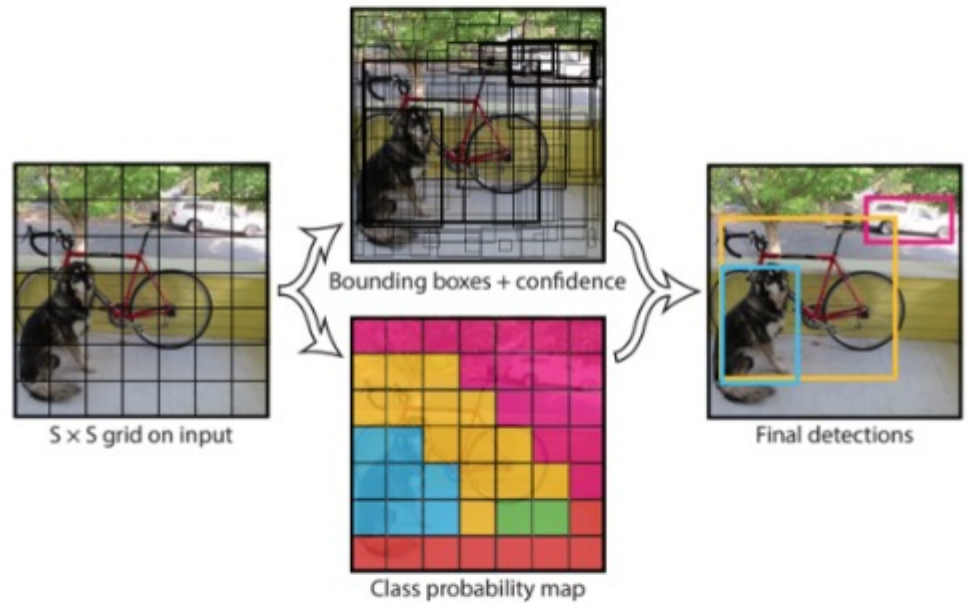
Bounding Box Recognition Process

The bounding box recognition process in YOLO involves the following steps:

- **Grid Creation:** The image is divided into an SxS grid. Each grid cell is responsible for predicting an object if the object's center falls within it.
- **Bounding Box Prediction:** Each grid cell predicts B bounding boxes and confidence scores for those boxes. The confidence score reflects how certain the model is that a box contains an object and how accurate it thinks the box is.
- **Class Probability Prediction:** Each grid cell also predicts C conditional class probabilities (one per class for the potential objects). These probabilities are conditioned on there being an object in the box.

YOLO Prediction

- We then use the output to make final detections
- Use a threshold to filter out bounding boxes with low P(Object)
- In order to know the class for the bounding box compute score take argmax over the distribution Pr(Class|Object) for the grid the bounding box's center is in

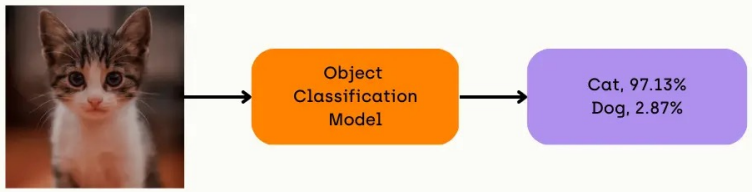


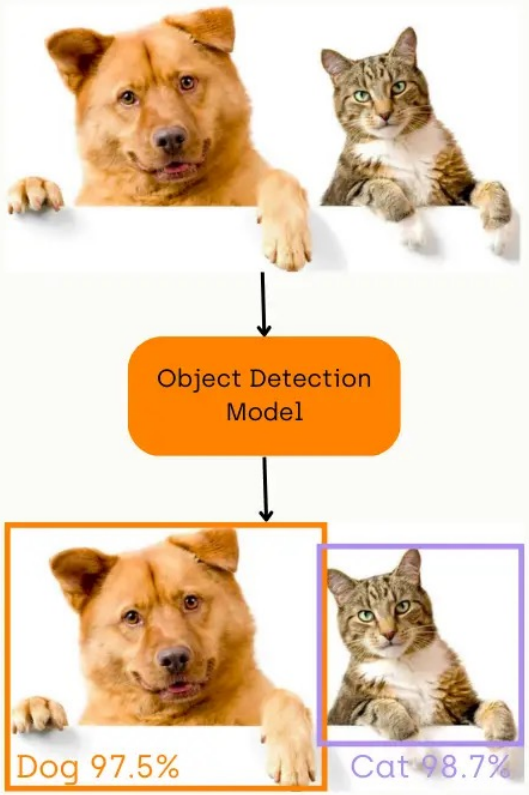
$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

Predictions and Confidence Scores:

1. Bounding Box Predictions: YOLO aims to locate objects within each grid cell by predicting bounding box coordinates. These coordinates specify the position of the bounding box's center (x, y) relative to the grid cell, as well as its width (w) and height (h). These predictions are crucial in defining the precise location of an object.

2. Class Probabilities: YOLO doesn't stop at just locating objects; it's equipped to classify them as well. For each bounding box, YOLO predicts a set of class probabilities. These probabilities represent the likelihood of the object within the bounding box belonging to various predefined classes. In essence, YOLO assigns labels to objects.


Object
Classification
Model
Cat, 97.13%
Dog, 2.87%


Object Detection
Model
Dog 97.5%
Cat 98.7%

- Objectness Score: YOLO's network learns to assign an objectness score for each bounding box. This score signifies the likelihood of a significant object's presence within the box.
- Class Confidence: Additionally, the confidence score takes into account the class probabilities assigned to the bounding box. The highest class probability contributes to the overall confidence score.

$$\text{Confidence Score} = \text{Objectness Score} \times \text{Highest Class Probability}$$

Vector Generalization

Vector generalization is a technique used in the YOLO algorithm to handle the high dimensionality of the output. The output of the YOLO algorithm is a tensor that contains the bounding box coordinates, objectness score, and class probabilities.

This high-dimensional tensor is flattened into a vector to make it easier to process. The vector is then passed through a softmax function to convert the class scores into probabilities. The final output is a vector that contains the bounding box coordinates, objectness score, and class probabilities for each grid cell.

YOLO Objective Function

- For YOLO, we need to minimize the following loss
- Sum squared error is used

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Coordinate Loss: Minimize the difference between x,y,w,h pred and x,y,w,h ground truth. ONLY IF object exists in grid box and if bounding box is resp for pred

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

Confidence Loss: Loss based on confidence ONLY IF there is object

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

No Object Loss based on confidence if there is no object

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

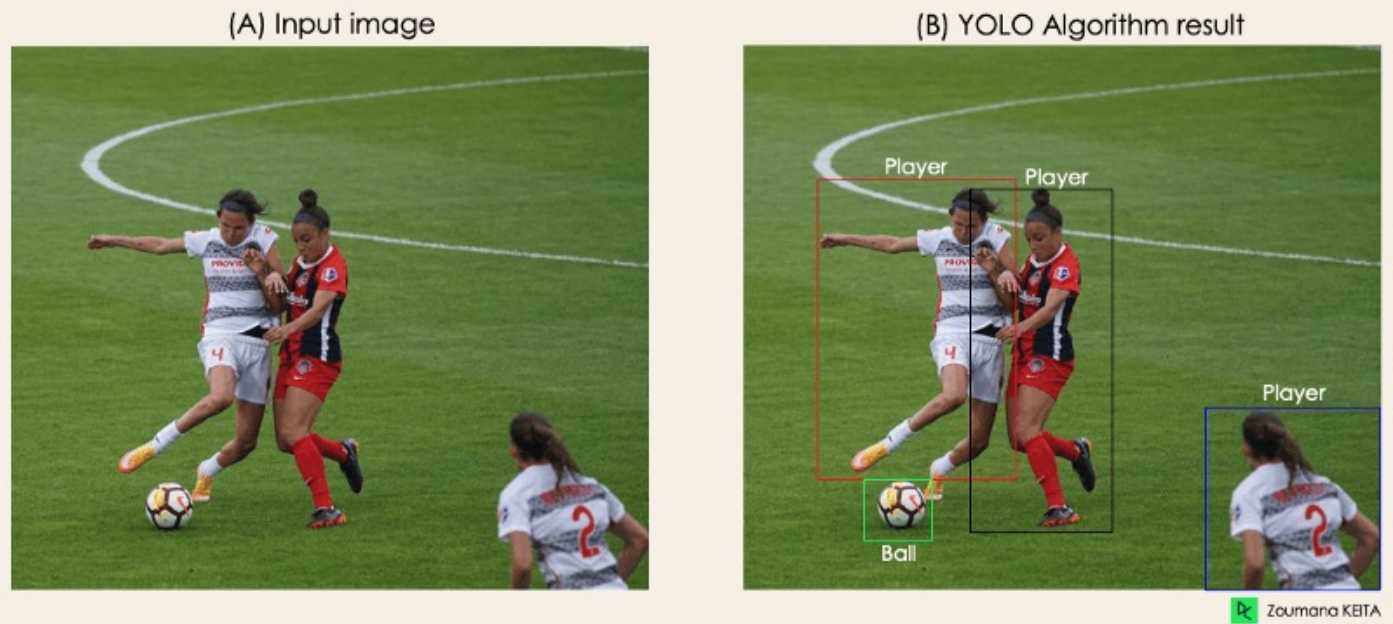
Class loss, minimize loss between true class of object in grid box

Components of YOLO

- **Resize input images:** Before entering the convolutional network, the input picture is scaled to 448x448.
- **Convolutional layers:** The architecture consists of two fully connected layers, four max-pooling layers, and 24 convolutional layers.
- **Regression Issue:** When solving a regression issue for object detection in YOLO, a single neural network predicts bounding boxes and class probabilities from the entire images in a single assessment.
- **Bounding Box Regression**: YOLO use bounding box regression to forecast the coordinates of each object's bounding box.
- **Intersection Over Union (IOU):** YOLO employs Intersection Over Union (IOU) to assess the precision of the projected bounding boxes.
- **Generalization:** YOLO develops generalizable representations of things, which reduces the likelihood that it will fail when used in new domains or with unexpected inputs.
- **Fast and Efficient:** YOLO is incredibly quick and can process photographs at a rate of 45 frames per second in real time. Fast YOLO, a scaled-down version of the network, achieves double the mAP of other real-time detectors while processing an incredible 155 frames per second.
- **High Precision:** YOLO delivers precise results with few background errors.

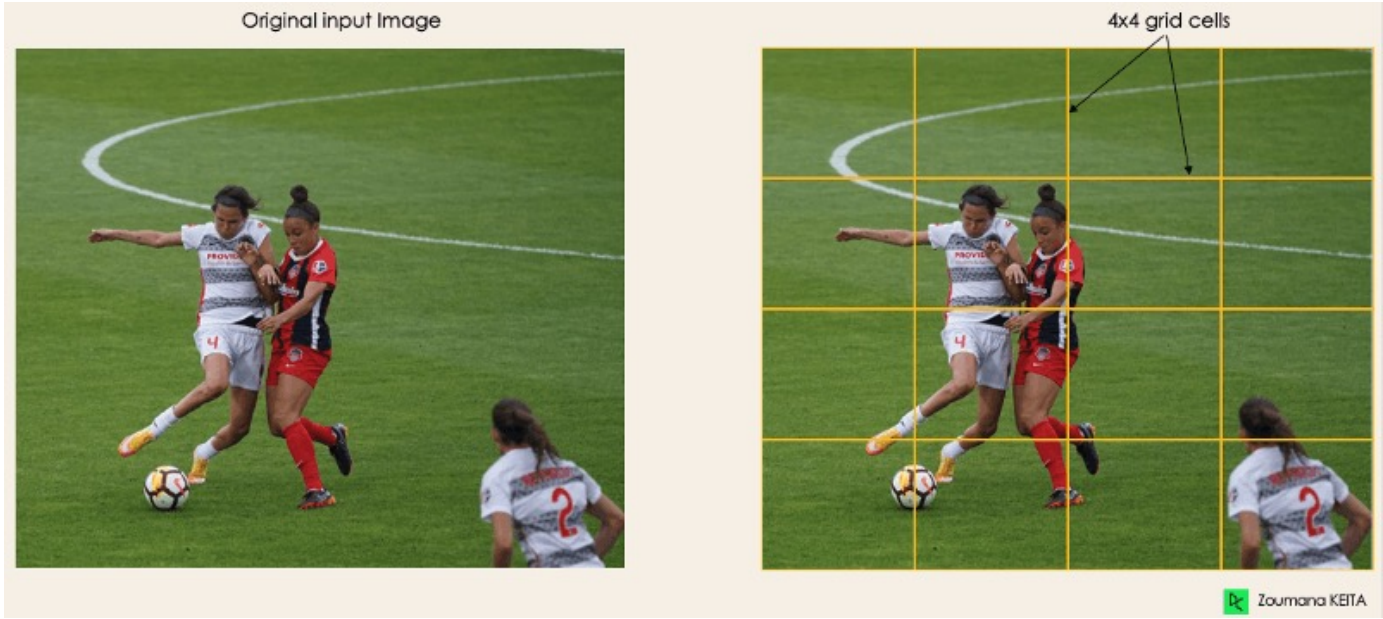
Architecture of YOLO

- Resizes the input image into 448x448 before going through the convolutional network.
- A 1x1 convolution is first applied to reduce the number of channels, followed by a 3x3 convolution to generate a cuboidal output.
- The activation function LeakyReLU, except for the final layer, which uses a linear activation function.
- Some additional techniques, such as batch normalization and dropout, regularize the model and prevent it from overfitting.



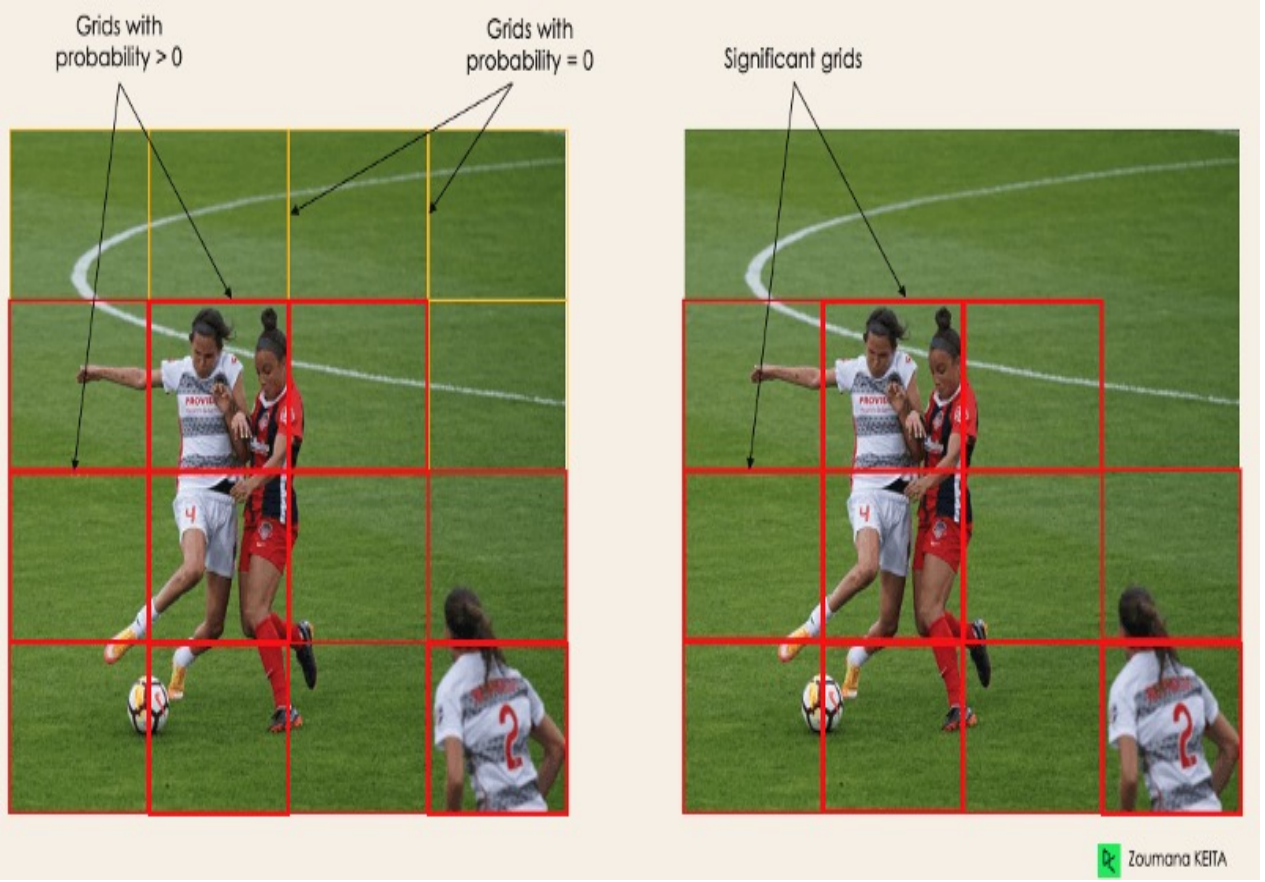
The algorithm works based on the following four approaches:

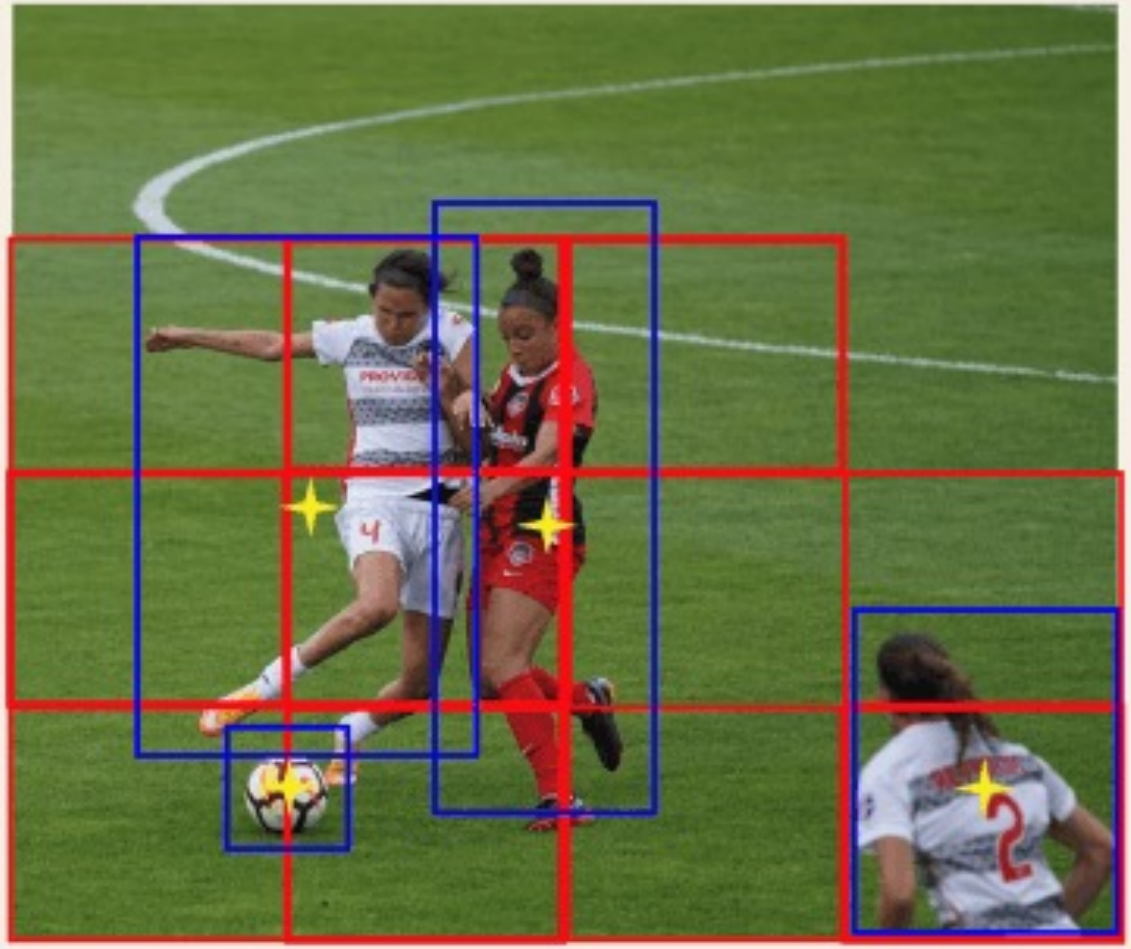
- Residual blocks: divide the original image (A) into NxN grid cells of equal shape, where N, in our case, is 4, as shown in the image on the right. Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value.
- Bounding box regression
- Intersection Over Unions or IOU for short
- Non-Maximum Suppression



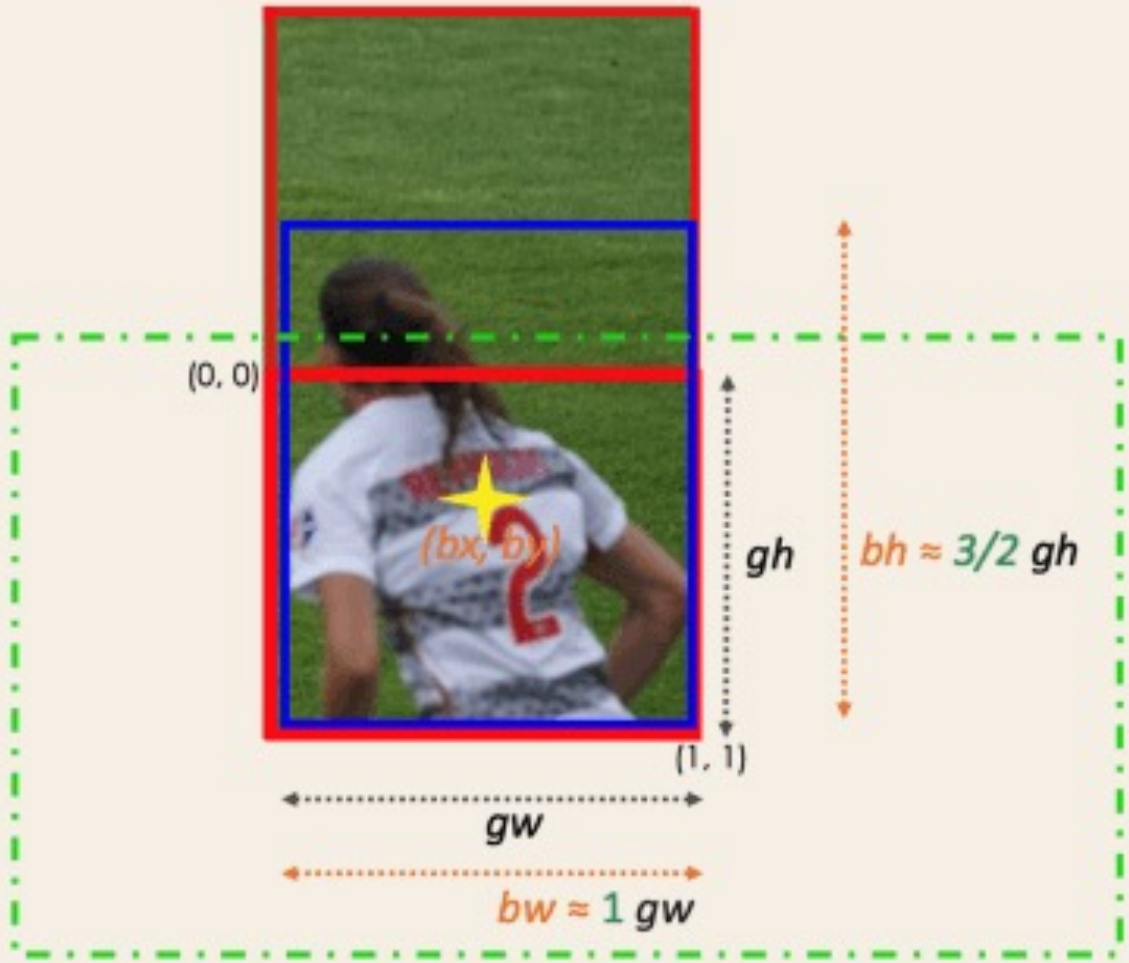
Bounding box regression

- Determine the bounding boxes corresponding to rectangles, highlighting all the objects in the image.
- Y = [pc, bx, by, bh, bw, c1, c2]
- pc corresponds to the probability score of the grid containing an object. For instance, all the grids in red will have a probability score higher than zero. The image on the right is the simplified version since the probability of each yellow cell is zero (insignificant).
- bx, by are the x and y coordinates of the center of the bounding box
- bh, bw correspond to the height and the width of the bounding box
- c1 and c2 correspond to the two classes, Player and Ball.





Bounding box centers



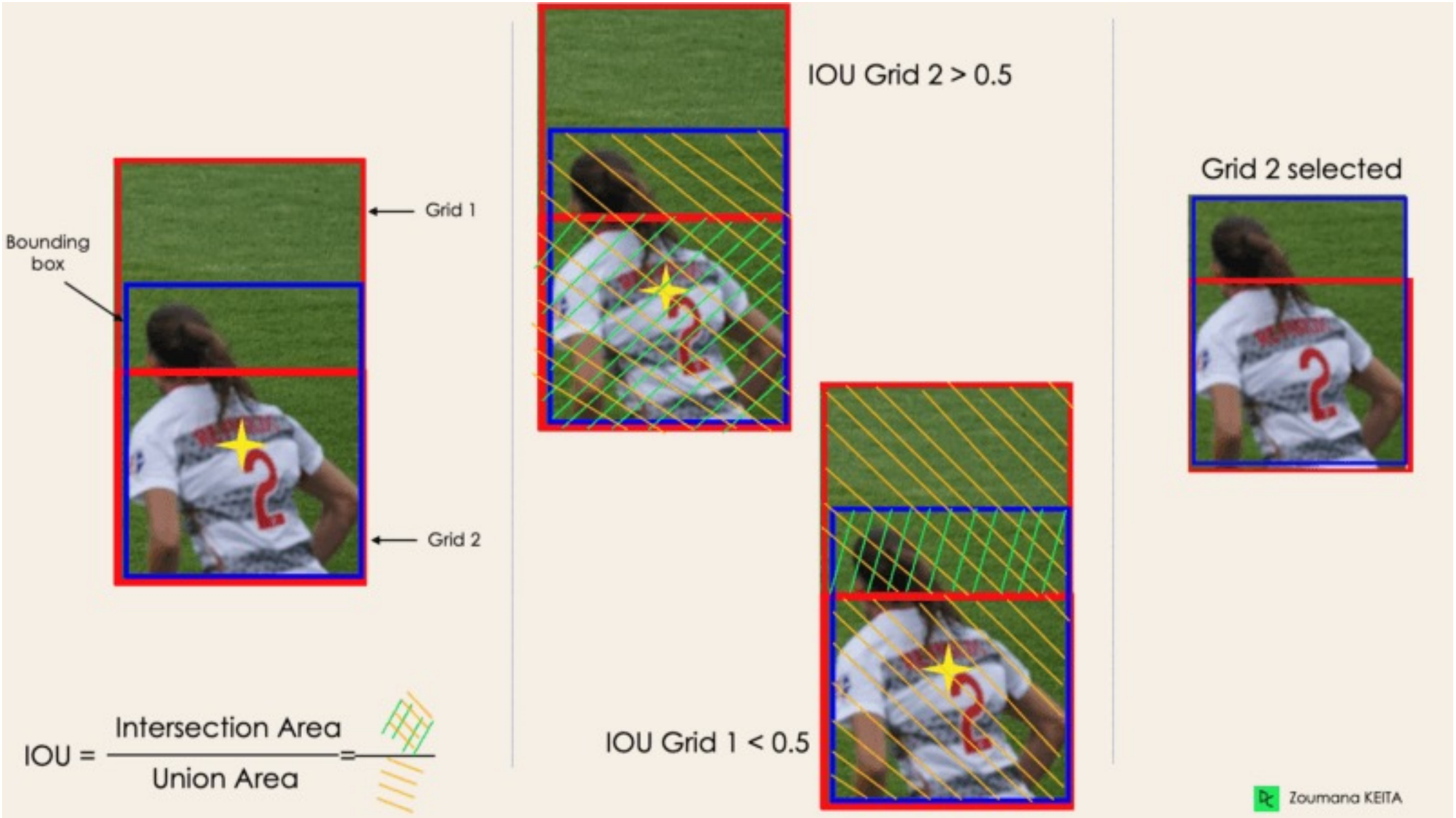
From the previous info we can have for e.g.
Y = [1, bx, by, 3/2, 1, c1, c2]

- First 1 means 100% of object presence

- *gh, gw*: height & width of the grid
- $0 \leq bx \leq 1$
- $0 \leq by \leq 1$
- *bh and bw* can be more than 1

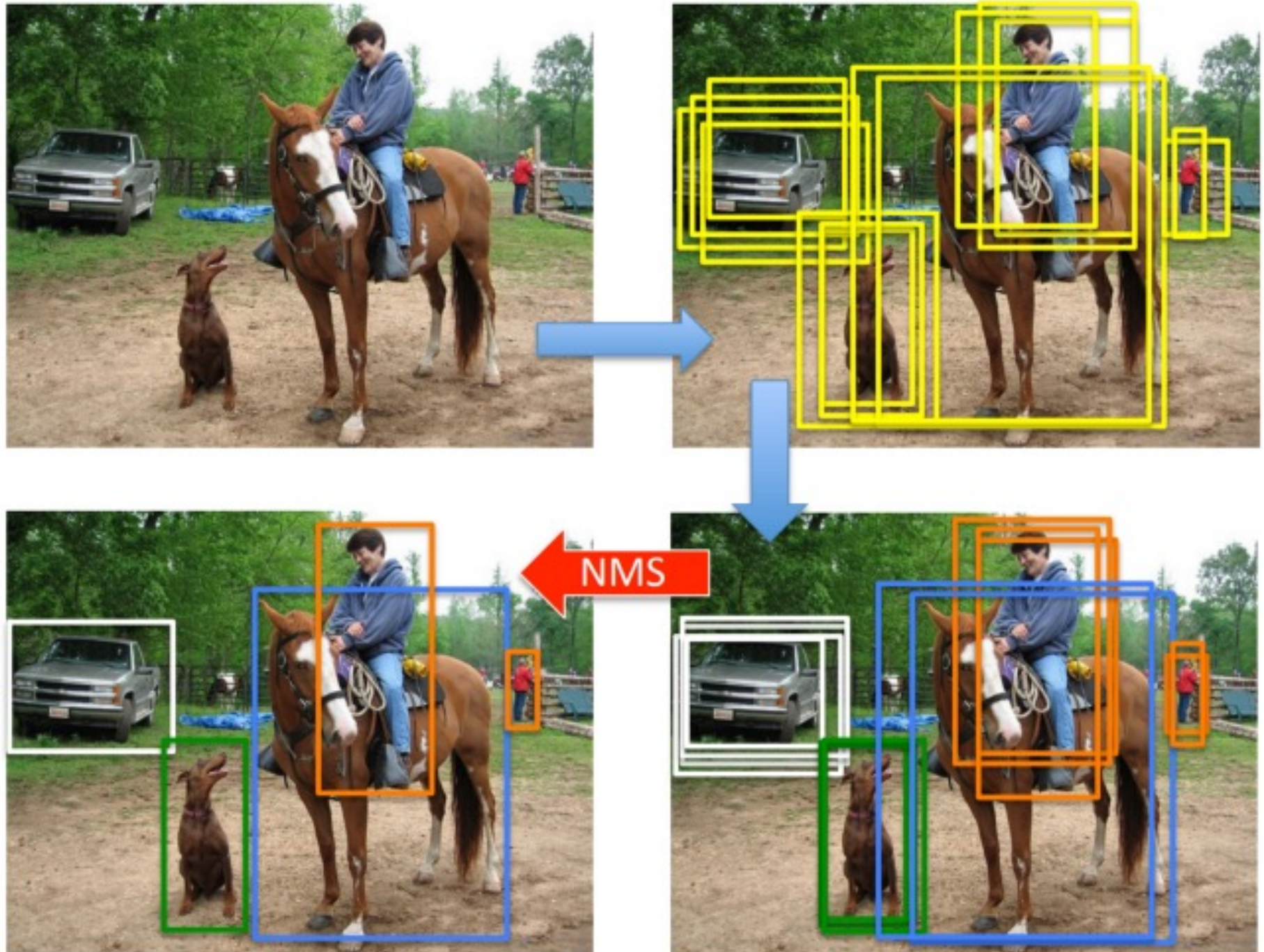
Zoumana KEITA

IoU


Bounding box
Grid 1
Grid 2
IOU Grid 2 > 0.5
IOU Grid 1 < 0.5
Grid 2 selected
IOU = Intersection Area / Union Area
Zoumana KEITA

NMS

- Setting a threshold for the IOU is not always enough because an object can have multiple boxes with IOU beyond the threshold, and leaving all those boxes might include noise.
- Here is where we can use NMS to keep only the boxes with the highest probability detection score.


NMS

Model	Type	Backbone	Neck	Head	Loss Function	Framework
YOLOv3	Fully convolution	Darknet-53	FPN	YOLOv3 head	Binary cross entropy	Darknet
YOLOv4	Fully convolution	CSPDarknet53	SPP and PANet	YOLOv3 head	Binary cross entropy	Darknet
YOLOv5	Fully convolution	CSPDarknet53	PANet	YOLOv3 head	Binary cross entropy and Logits loss	Pytorch
YOLOv6	Fully convolution	EfficientRep	Rep-PANet	Decoupled head	Varifocal Loss and Distribution Focal Loss	Pytorch
YOLOv7	Fully convolution	E-ELAN	FPN and PANet	Multiple heads	Binary cross entropy with Focal loss	Pytorch

3.2.1. YOLOv3