

# Fast Object Detection With YOLO

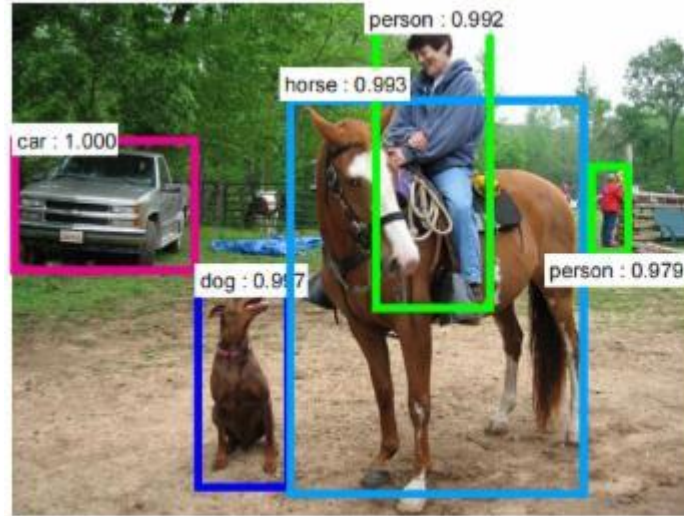
Redmon, Divvala, Girshick, Farhadi. **You Only Look Once: Unified, Real-Time Object Detection (2016)**

# Outline

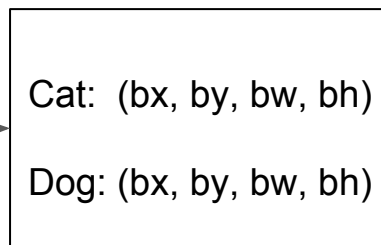
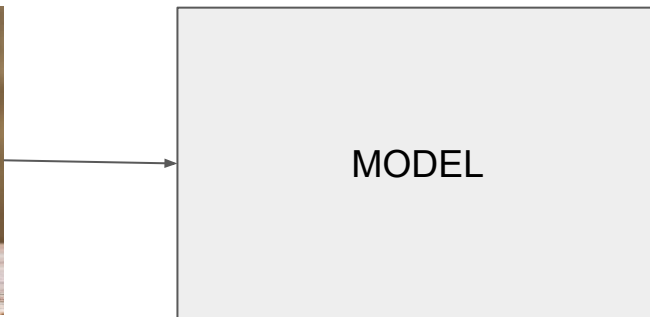
1. Object detection: Problem
2. Previous approaches
  - 2.1. Cascade Classifiers
  - 2.2. CNN based methods
3. YOLO
  - 3.1. Motivation
  - 3.2. Gridcell Detection
  - 3.3. Multitask Loss
  - 3.4. Implementation details
  - 3.5. Performance

# What is object detection?

1. Localization.
2. Classification
3. Multiple Objects



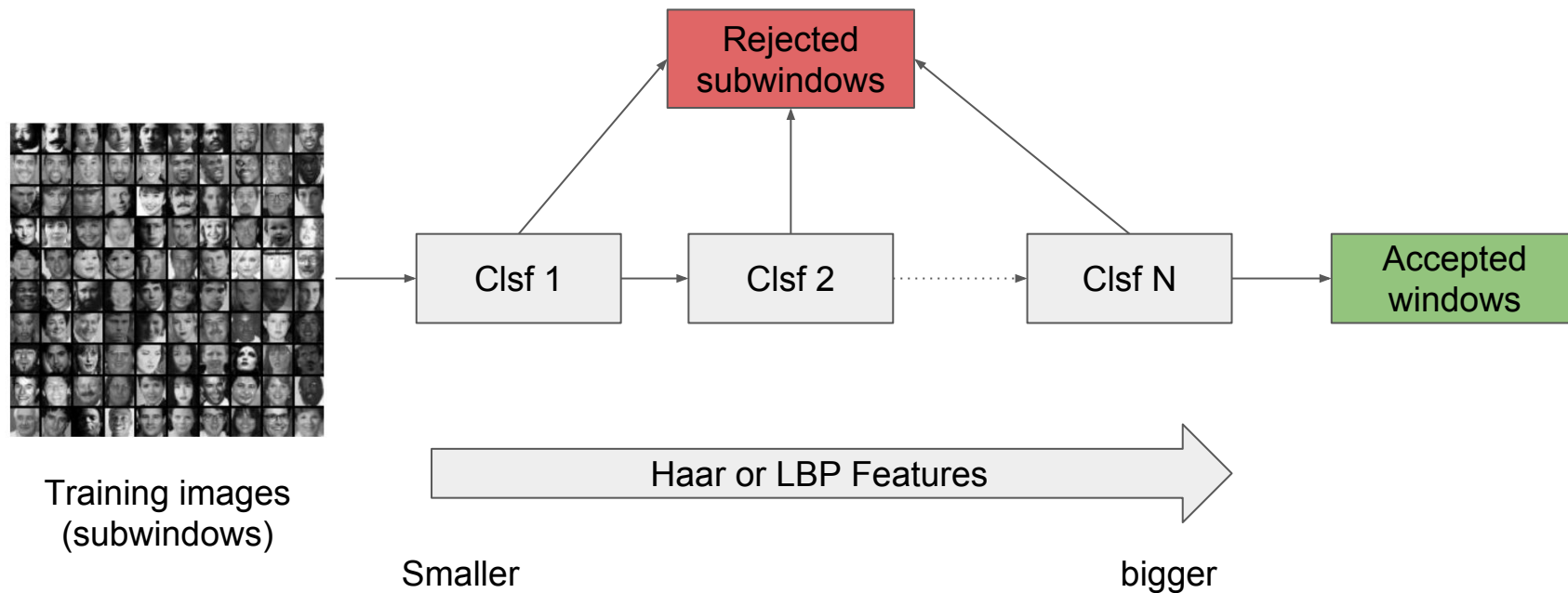
# What is object detection: Pipeline



## **Before YOLO: Approaches to solve the problem**

# Previous Approaches: Cascade Classifiers (Training)

[Example: Viola-Jones Face detection \(2001\)](#)

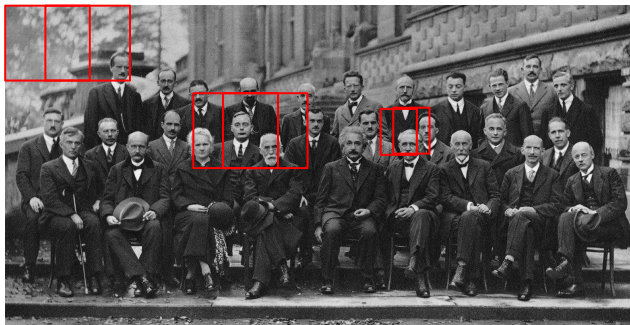


Each classifier (weak) is trained independently !!!!!

# Previous Approaches: Cascade Classifiers (inference)

## Example: Viola-Jones Face detection (2001)

1. Extract image subwindows



2. Feed them to the cascade detector



Cascade detector

Each test image can have  
thousands of subregions !!!!



## Previous Approaches: Cascade Classifiers (Advantages)

[Example: Viola-Jones Face detection \(2001\)](#)

### **Advantages:**

1. Feasible to implement in real time (15 FPS), Features on each stage are computed efficiently (Constant time).
2. Invariant to scale and translation.

### **Disadvantages:**

1. Intolerance to object rotations.
2. Big amount of subregions in which to make inference.
3. Needs a Lot of training examples.
4. Each weak classifier does not generalize very well.



## **Before YOLO: Approaches with CNNs**

## Previous Approaches: CNN Based methods, Naive approach

1. Take an image classifier (e.g. a CNN).
2. Apply the classifier over several regions of the image.

### Disadvantages:

1. Must propagate the network for each image region.
2. Objects have different spatial locations and different sizes.
3. Amount of subregions explode.





Well, its not very good is it?

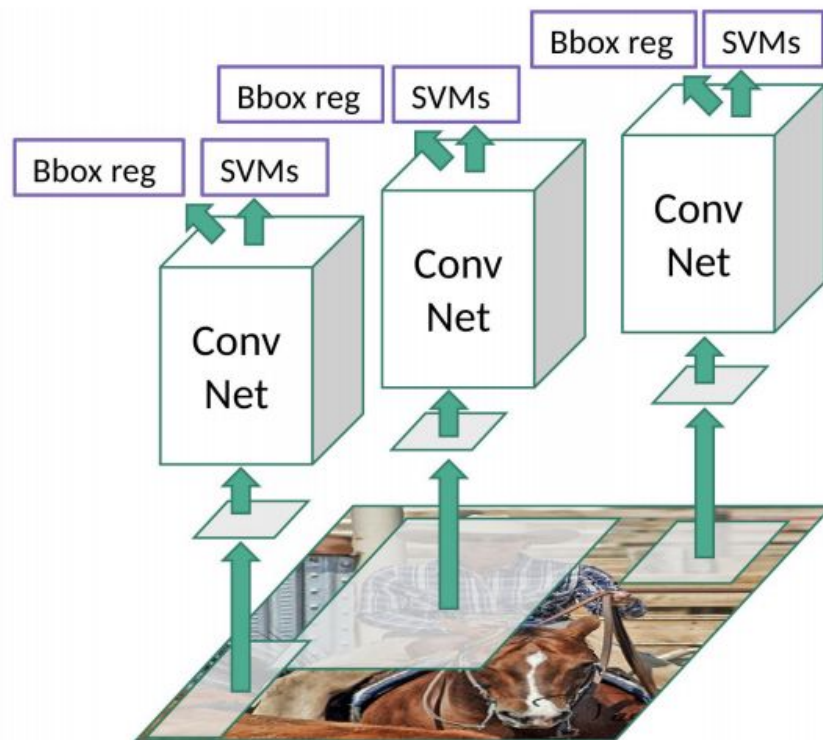
# Previous Approaches: CNN Based methods, RCNNs

## Motivation:

1. Reduce the amount of subregions.

## Key ideas:

1. Now only 2000 subregions (region proposals)
2. The region proposals (RP) are computed by [selective search](#).
3. The RP are feeded to the network(trained on other dataset) to produce 4096 feature vectors.
4. Each Region is classified with SVMs.
5. Bbox regression to increase localization accuracy.



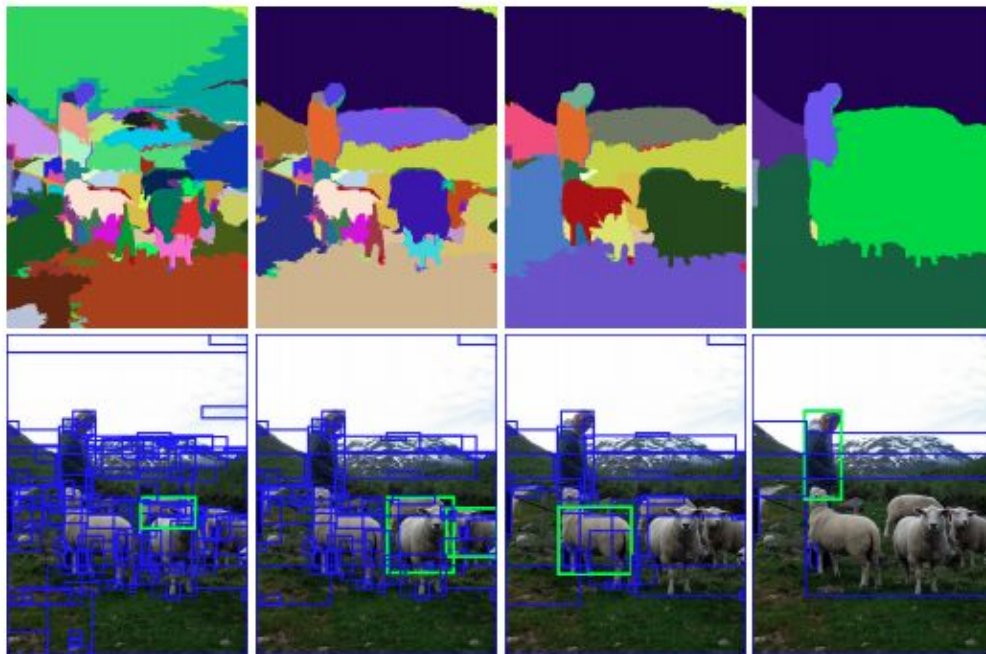
R-CNN

## Previous Approaches: CNN Based methods, RCNNs (2014)

### Selective Search: Bottom up approach

Runs a segmentation algorithm to find it out where an object can be located.

1. Generate an initial sub segmentation (small scale).
2. Use a strategy to recursively combine several groups (Increasing scale).
3. On each level choose object hypotheses to run the detection algorithm.
4. Positive examples  $\text{IOU} \geq 0.5$  with ground truth.



(a)



RCNN comes with lots of drawbacks, What to do Then?



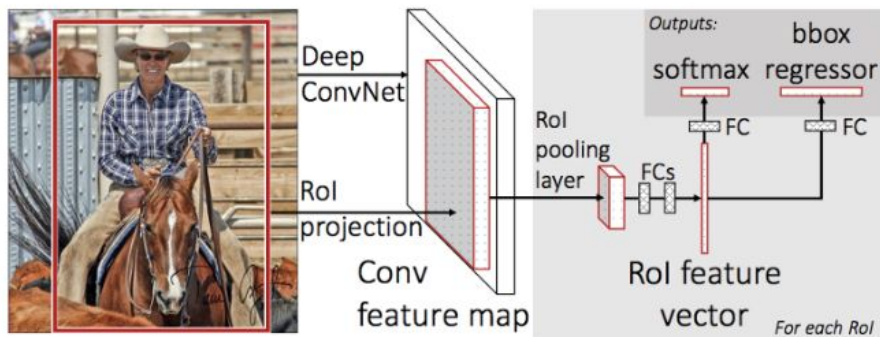
# Previous Approaches: CNN Based methods, Fast R-CNNs (2015)

## Motivation:

1. Reduce the amount of computation involving the region proposals.
2. Avoid multi stage training.

## Key ideas:

1. Still uses a region proposal algorithm.
2. Convolutional implementation of sliding windows over RP.
3. ROI pooling.
4. Train only a CNN that outputs a classification and a bbox coordinates.



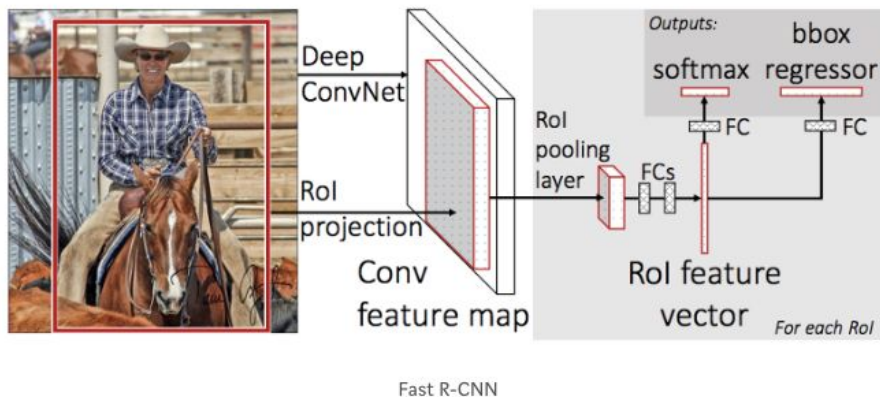
Fast R-CNN



# Previous Approaches: CNN Based methods, Fast R-CNNs (2015)

## Convolutional Region proposals:

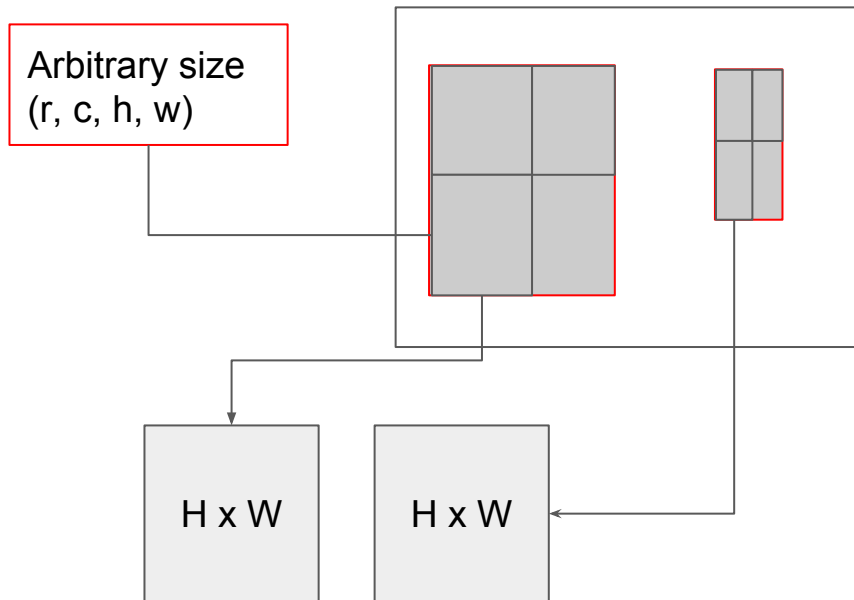
1. Extract the Region Proposals set.
2. Feed the whole image to the CNN.
3. Project each RP Over the feature map
4. Perform ROI pooling over the projected RP on the feature map.
5. Extract a ROI feature vector (fixed length).
6. Feed the feature vector through two sibling layers.
7. Produce softmax probability estimates.
8. Bbox regression.



## Previous Approaches: CNN Based methods, Fast R-CNNs (2015)

### ROI Pooling:

1. Input: ROI over feature map ( $r, c, h, w$ ).  
Top Left corner, height and width.
2. Divides the input into  $H, W$  subwindows, each one of height  $h/H$  and  $w/W$ .
3. Max pooling over each subwindows generating the corresponding output.



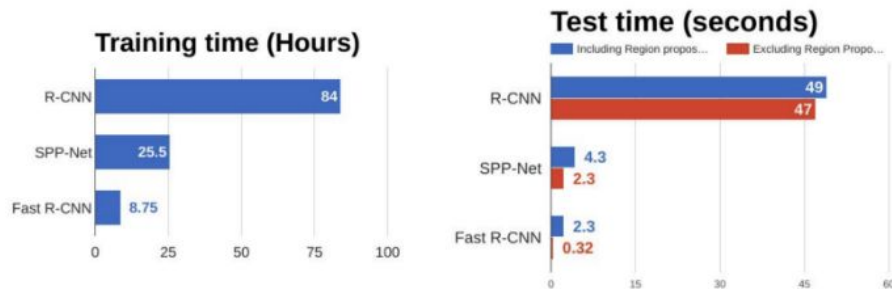
# Previous Approaches: CNN Based methods, Fast R-CNNs (2015)

## Advantages:

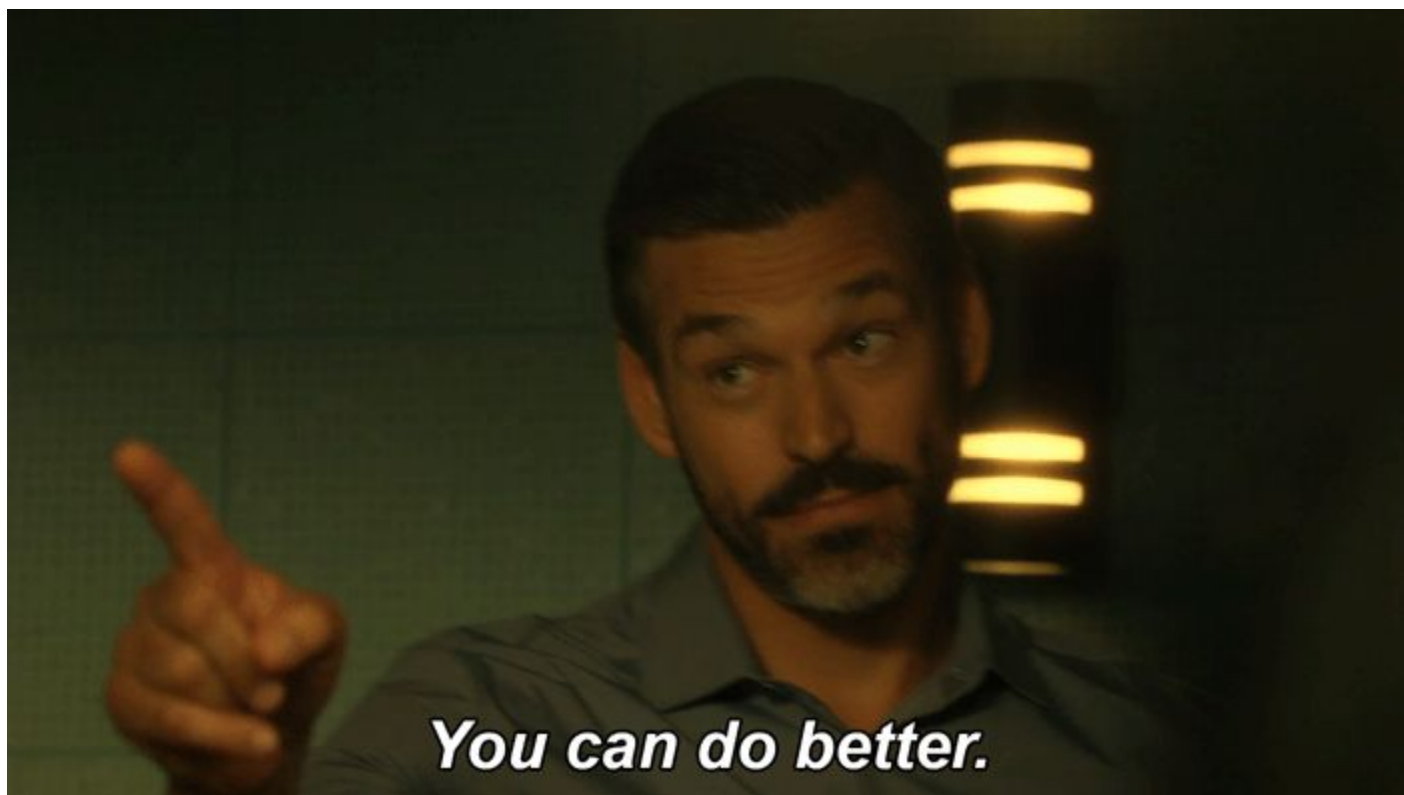
1. Avoids feeding the network with each region proposal.
2. Reduces computational time considerably.
3. Trained end to end from region proposals (Multitask loss).

## Disadvantages:

1. You still have to extract regions proposals (expensive).
2. 2.3 seconds per test image.



Comparison of object detection algorithms



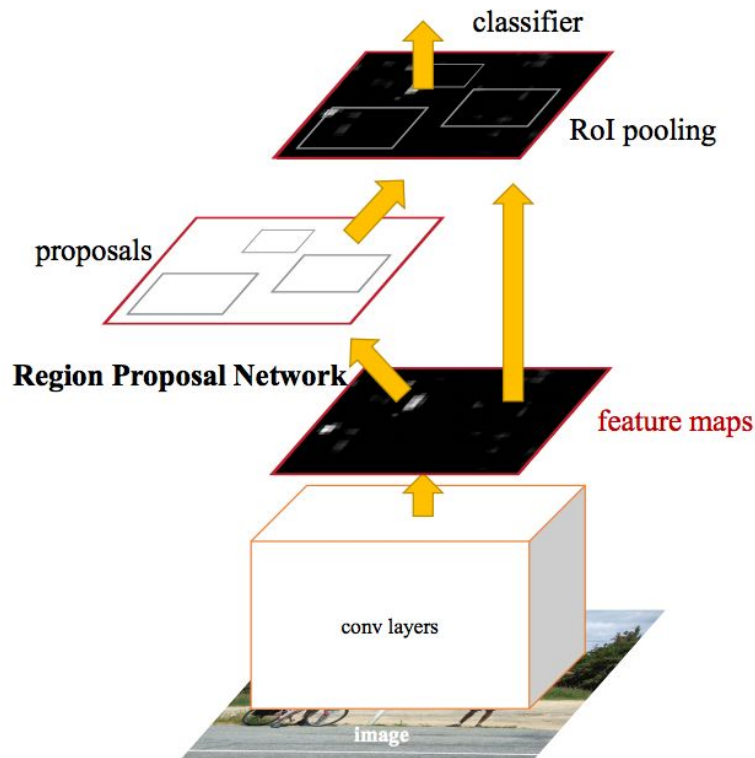
# Previous Approaches: CNN Based methods, FastER R-CNNs (2015)

## Motivation:

1. Bottleneck: Region proposals.
2. Avoid Selective search.

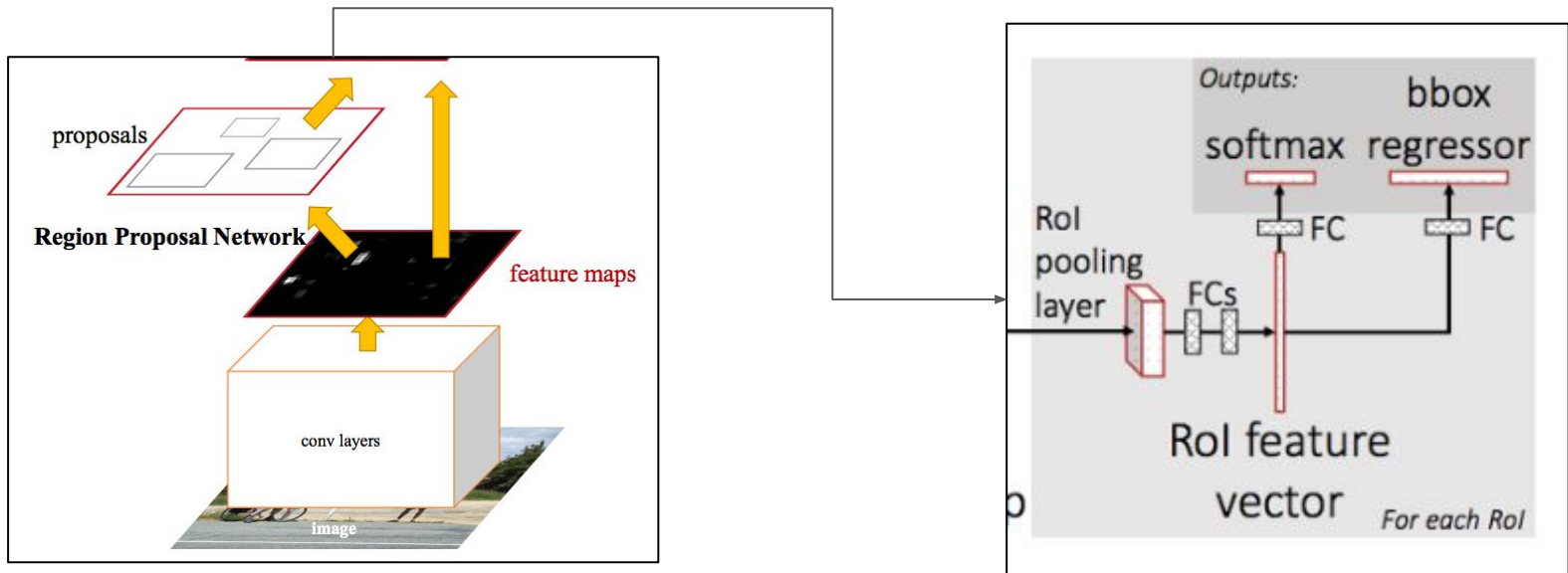
## Key ideas:

1. Let the network learn the region proposals.
2. One network to extract Region proposals(non object loss + region bbox loss).
3. Second network to make object classification(Fast rCNN loss).
4. Third network to extract feature maps.



# Previous Approaches: CNN Based methods, FastER R-CNNs (2015)

## Complete Model:



Fast RCNN Module

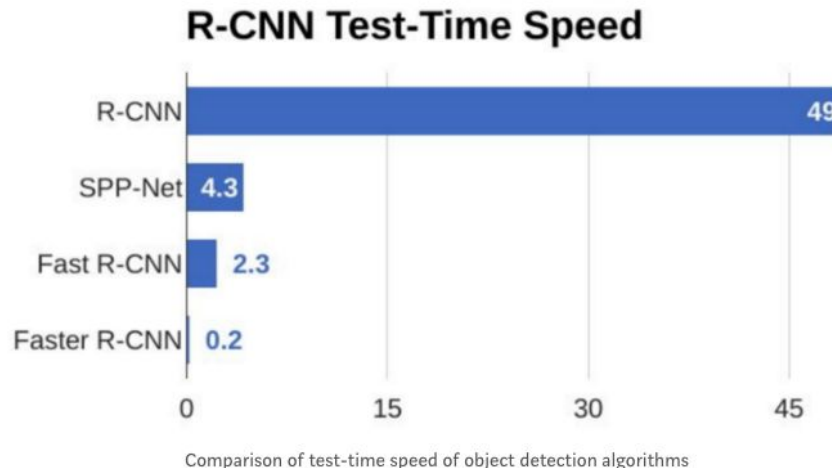
## Previous Approaches: CNN Based methods, FastER R-CNNs (2015)

### Advantages:

1. Reduce computational cost even more.
2. Shares features between both region proposal and classifier network. (*Alternate training, approximate joint training*)

### Disadvantages:

1. 0.2 secs, Good but not terrific (5 Fps approx).
2. Hard to train. Train three different networks.



**However, Seems Good ....**






**But.... Can we do ..... Even better .... ?**



Introducing YOLO V1: You  
Only look once

The word "YOLO" is rendered in a bold, black, stylized font. Each letter is outlined with a thick, bright cyan border. The letters are closely spaced, with the 'Y' and 'O' overlapping significantly, and the 'L' and 'O' also overlapping. The overall style is reminiscent of a graffiti or street art aesthetic.

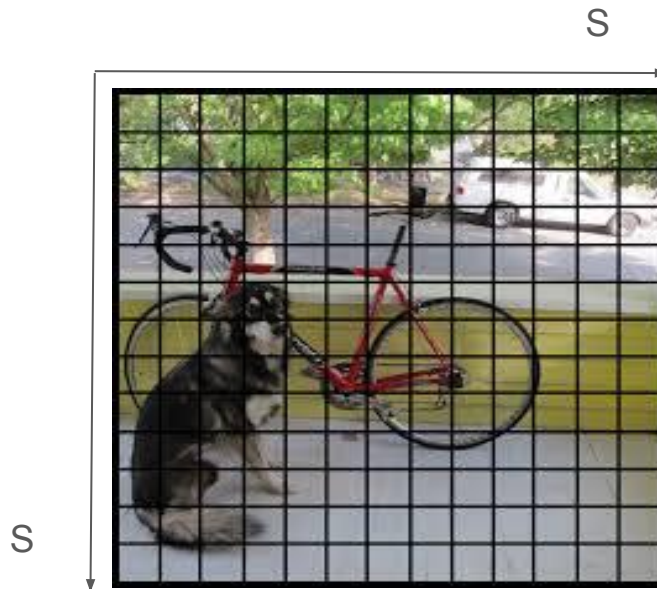
# YOLO: Generalities

## Motivation:

1. Avoid regions proposals and sliding windows.

## Key ideas:

1. Divide the image into grid cells  $S \times S$ .



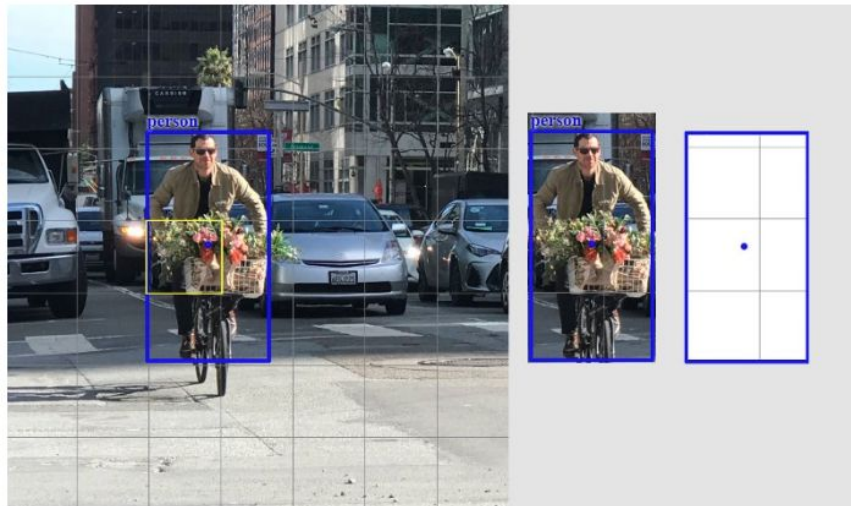
# YOLO: Generalities

## Motivation:

1. Avoid regions proposals and sliding windows.

## Key ideas:

1. Divide the image into grid cells.
2. **Each grid cell is responsible for detecting only ONE object.**



Center of the object is located inside the gridcell

[You only look once: Unified. Real-Time object detection](#)



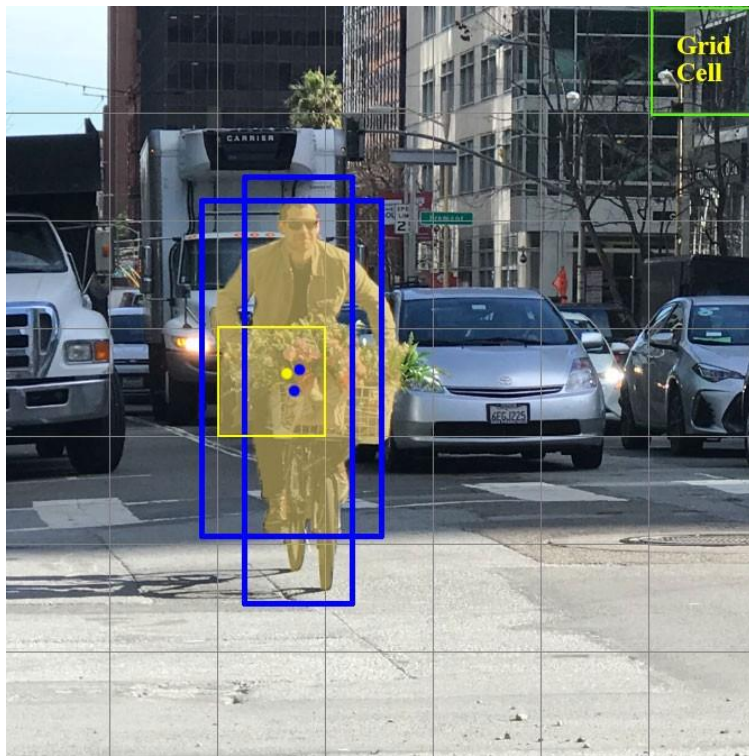
# YOLO: Generalities

## Motivation:

1. Avoid regions proposals and sliding windows.

## Key ideas:

1. Divide the image into grid cells.
2. Each grid cell is responsible for detecting only ONE object.
3. **However, each gridcell predicts B bounding boxes.**



The object in that cell can have different sizes, so B bounding boxes are predicted

[You only look once: Unified. Real-Time object detection](#)



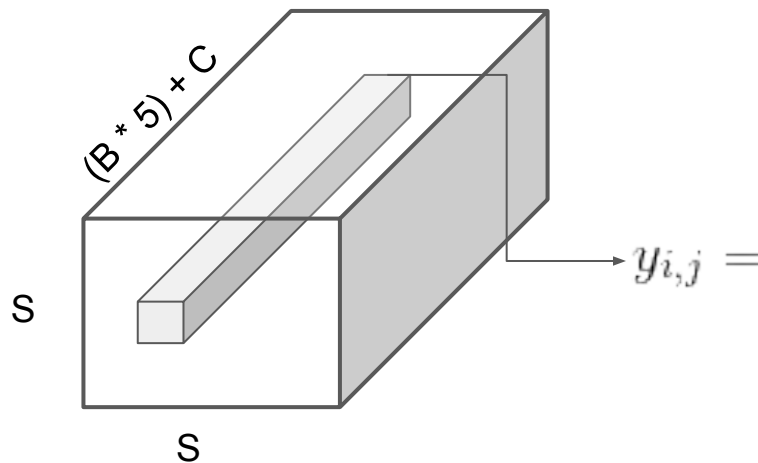
# YOLO: Generalities

## Motivation:

1. Avoid regions proposals and sliding windows.
2. **Let a single network do the job.**

## Key ideas:

1. Divide the image into grid cells.
2. Each grid cell is responsible for detecting only **ONE** object.
3. **Each grid cell can detect B bounding boxes for the same object.**



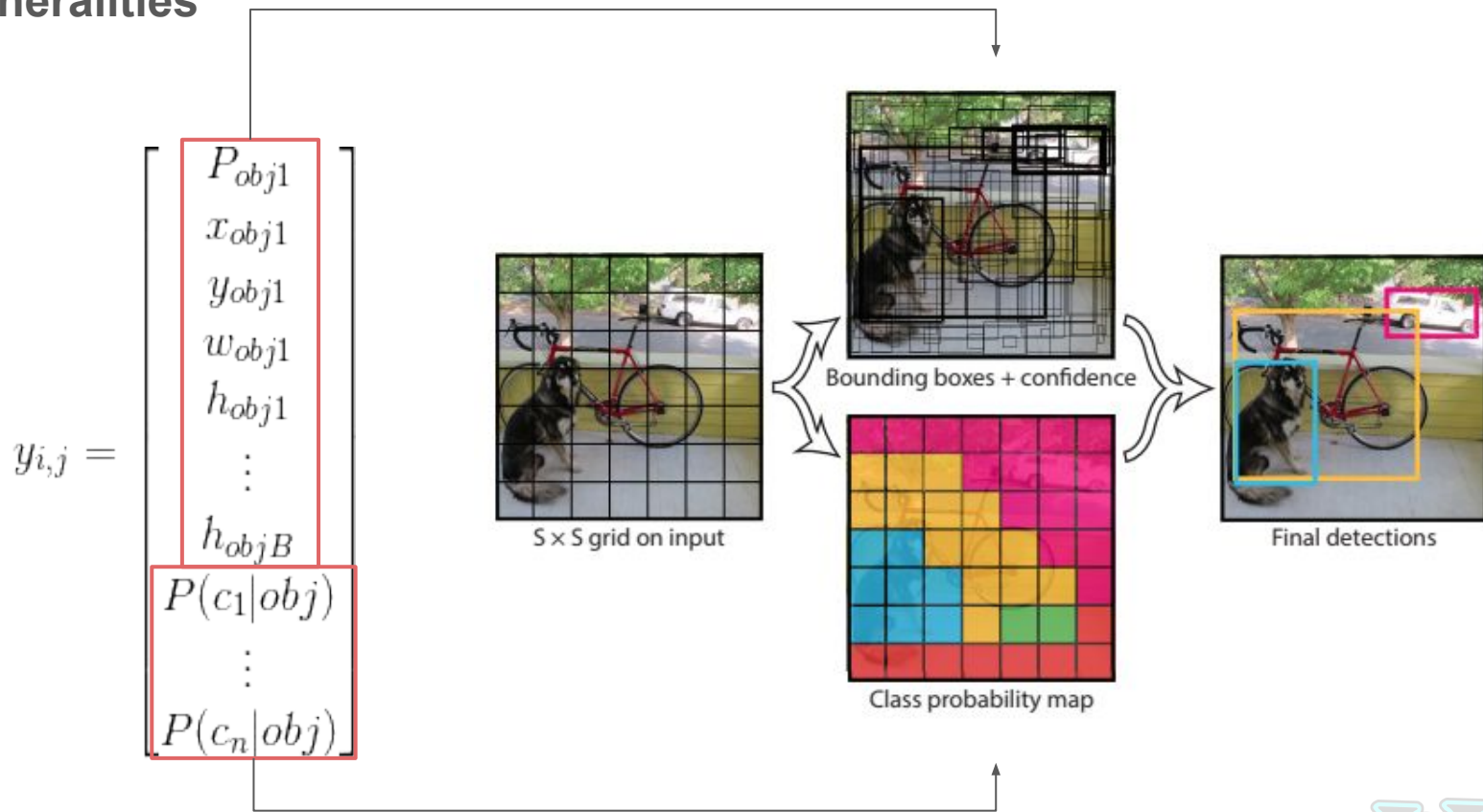
Output Volume of YOLO

$$\begin{bmatrix} P_{obj1} \\ x_{obj1} \\ y_{obj1} \\ w_{obj1} \\ h_{obj1} \\ \vdots \\ h_{objB} \\ P(c_1|obj) \\ \vdots \\ P(c_n|obj) \end{bmatrix}$$

[You only look once: Unified. Real-Time object detection](#)



# YOLO: Generalities



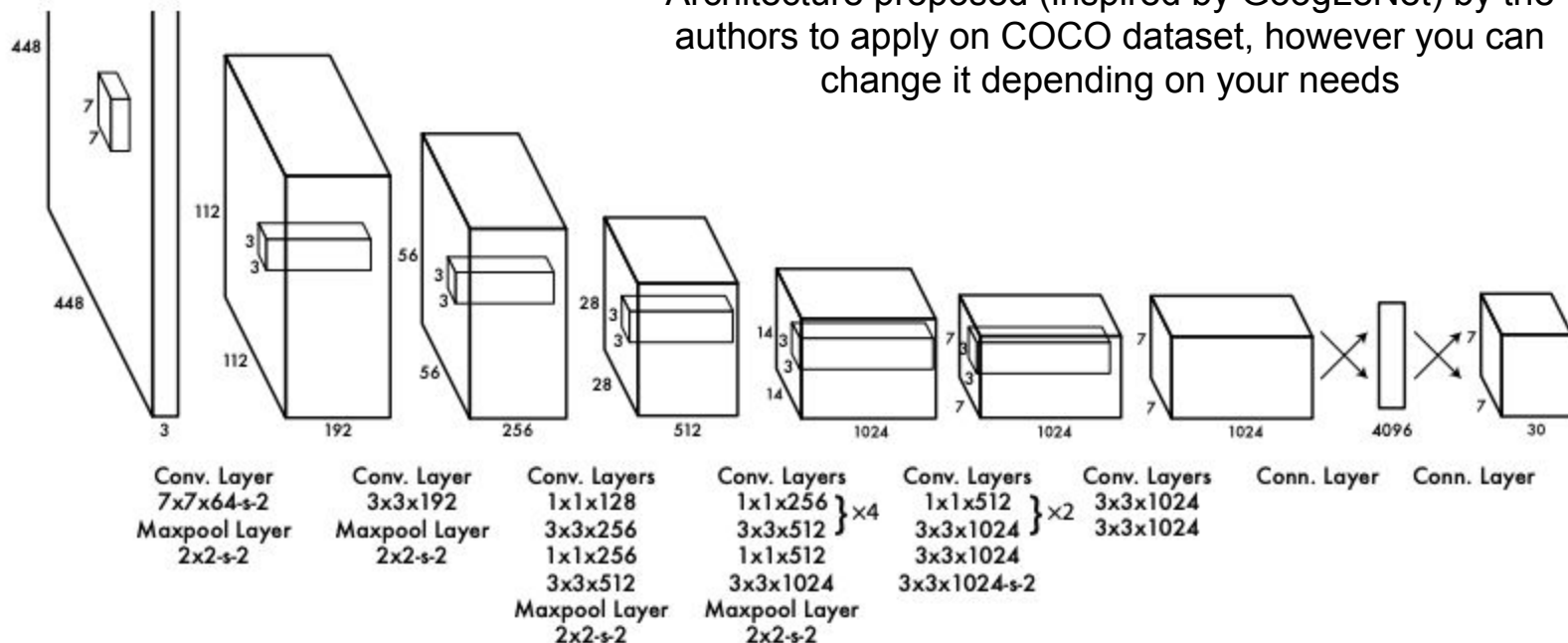
You only look once: Unified. Real-Time object detection





# YOLO: Architecture

Architecture proposed (inspired by GoogLeNet) by the authors to apply on COCO dataset, however you can change it depending on your needs



[You only look once: Unified. Real-Time object detection](#)





# YOLO: Multitask loss function

## Motivation:

1. In classification or regression we have a simple output.
2. Here we have several outputs for each bounding box.
3. A function must be designed to measure the loss with the ground truth.
4. The loss is divided in 5 parts: location loss, shape loss, object loss, non object loss and class probability loss.
5. The hyperparameters  $\lambda$  are for training stability.

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$



# YOLO: Loss function

**Location loss**

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

*Shape loss*

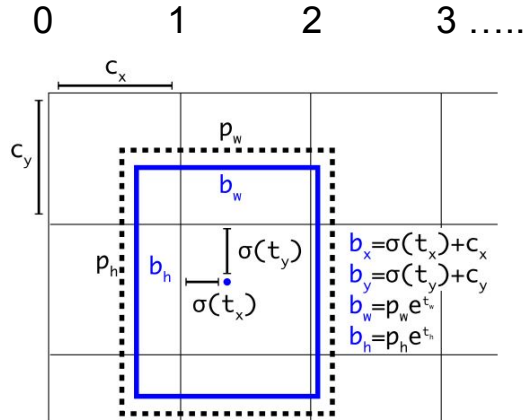
*Objectedness loss*

Non object loss

Classification loss

```
def xy_loss_fn(pred_xy, true_xy, mask):
    with tf.name_scope('xy_loss'):
        xy_diff = tf.boolean_mask(
            tf.square(pred_xy - true_xy),
            mask
        )
    return tf.reduce_sum(xy_diff)
```

Gridcell coordinate convention



[You only look once: Unified. Real-Time object detection](#)



# YOLO: Loss function

*Location loss*

**Shape loss**

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

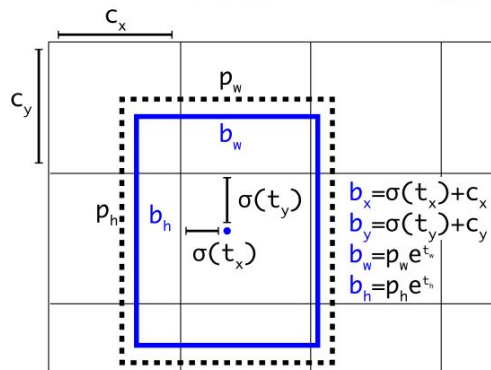
*Objectedness loss*

Non object loss

Classification loss

Square root terms partially give more importance to small deviations in small boxes

```
def wh_loss_fn(pred_wh, true_wh, mask):  
    with tf.name_scope('wh_loss'):  
        wh_diff = tf.boolean_mask(  
            tf.square(tf.sqrt(pred_wh) - tf.sqrt(true_wh)),  
            mask  
        )  
    return tf.reduce_sum(wh_diff)
```



[You only look once: Unified. Real-Time object detection](#)



# YOLO: Loss function

*Location loss*

*Shape loss*

**Objectedness loss** 
$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2$$

Non object loss

Classification loss

```
def obj_loss_fn(pred_obj, iou_scores, mask):  
    # I think iou_scores my be replaced by simply a 1 in this  
    with tf.name_scope('obj_loss'):  
        obj_diff = tf.boolean_mask(  
            tf.square(1 - pred_obj), mask  
        )  
        return tf.reduce_sum(obj_diff)
```

[You only look once: Unified. Real-Time object detection](#)



# YOLO: Loss function

*Location loss*

*Shape loss*

*Objectedness loss*

**Non object loss**

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

Classification loss

```
def no_obj_loss_fn(pred_obj, iou_scores, mask):  
    """For this function mask should tell where there is NO object. """  
    with tf.name_scope('no_obj_loss'):  
        no_obj_diff = tf.boolean_mask(  
            tf.square(pred_obj), mask  
        )  
        return tf.reduce_sum(no_obj_diff)
```

[You only look once: Unified. Real-Time object detection](#)



# YOLO: Loss function

*Location* loss

*Shape* loss

*Objectedness* loss

Non object loss

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

Using Sum of square roots, even in classification task makes the loss easier to optimize (according to the authors)

```
def clf_loss_fn(pred_cls, true_cls, mask):  
    with tf.name_scope('clf_loss'):  
        clf_diff = tf.boolean_mask(  
            tf.square(pred_cls - true_cls), mask  
        )  
        return tf.reduce_sum(clf_diff)
```

[You only look once: Unified. Real-Time object detection](#)



# YOLO: Loss function, putting all together

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2 \end{aligned}$$

```
xy_loss = xy_loss_fn(pred_xy, true_xy, obj_mask)
wh_loss = wh_loss_fn(pred_wh, true_wh, obj_mask)
obj_loss = obj_loss_fn(pred_obj, iou_scores, obj_mask)
no_obj_loss = no_obj_loss_fn(pred_obj, iou_scores, no_obj_mask)
clf_loss = clf_loss_fn(cls_scores, true_cls, obj_mask)
```

```
# Calculate Loss (for both TRAIN and EVAL modes)
task_loss = (params['obj_coord_scale'] * (xy_loss + wh_loss)
             + params['obj_conf_scale'] * obj_loss
             + params['noobj_conf_scale'] * no_obj_loss
             + params['obj_class_scale'] * clf_loss)
```

The bounding box and classification loss will only have effect when there is an object present in the grid cell

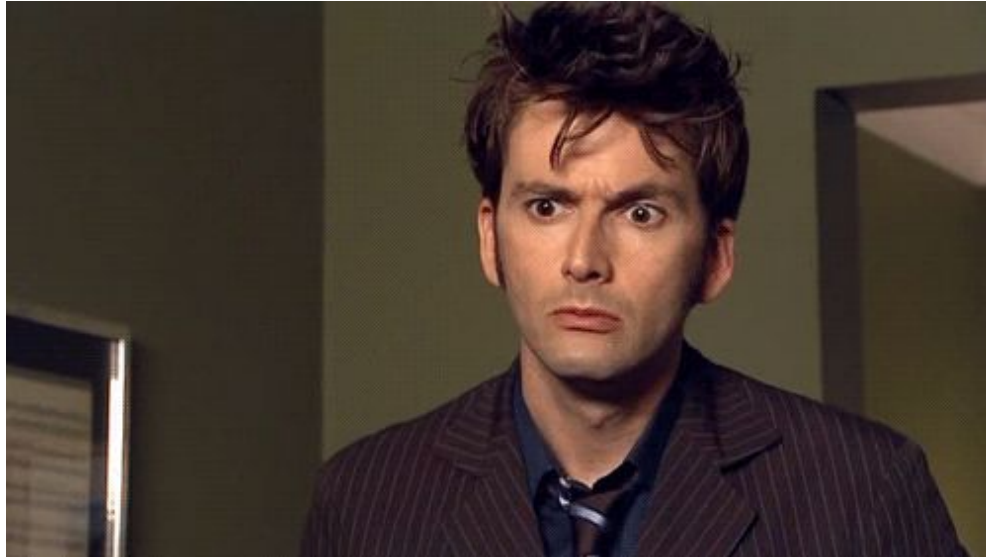
[You only look once: Unified. Real-Time object detection](#)



It seems we have everything set..... But ....



Earlier we said we had several bounding boxes for the same object..... So which one we choose if we are making inference?

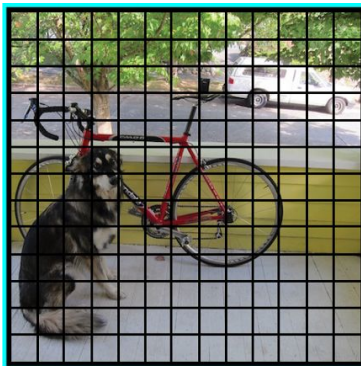


# YOLO: Non max suppression

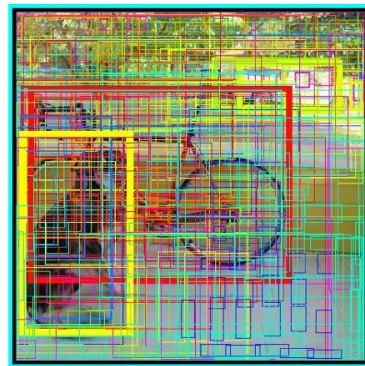
## Situations:

1. Sometimes is clear in which box the center of the object falls.
2. In other cases the object center can be located in several boxes.
3. This generates ambiguity (multiple choices).

Original image



Multiple predictions

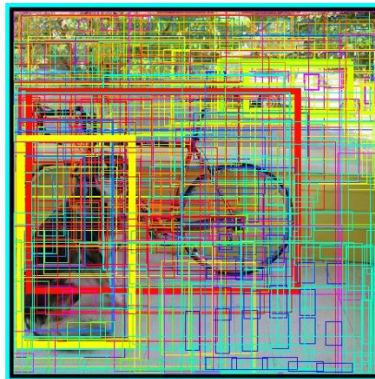


# YOLO: Non max suppression

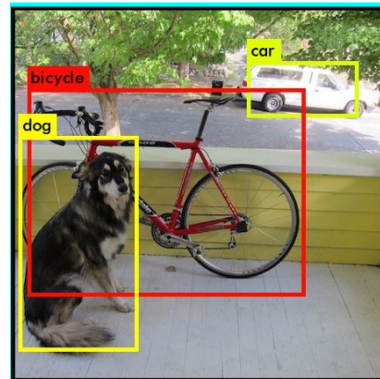
## Solution: For each class

1. Discard all boxes with Probability of class below a threshold.
2. While there are bboxes left pick the one with the highest probability.
3. Discard the remaining boxes with IoU above a threshold (0.5).
4. Repeat step 2.

Original image

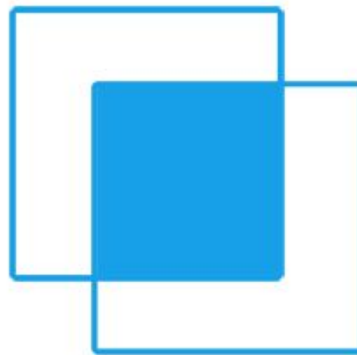


After non max supr



## YOLO: Non max suppression

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



# YOLO: Performance

## Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20



# YOLO: Limitations

1. Strong spatial constraints on the bounding boxes.
2. Only B (B=2 paper) bounding boxes per gridcell and only one as well.
3. Problems with objects appearing in groups close to each other.
4. Does not generalize very well the bounding box shape.
5. The mislocalization of small objects has big impact on the loss function.



When objects are too close YOLO has problems detecting the

[You only look once: Unified. Real-Time object detection](#)



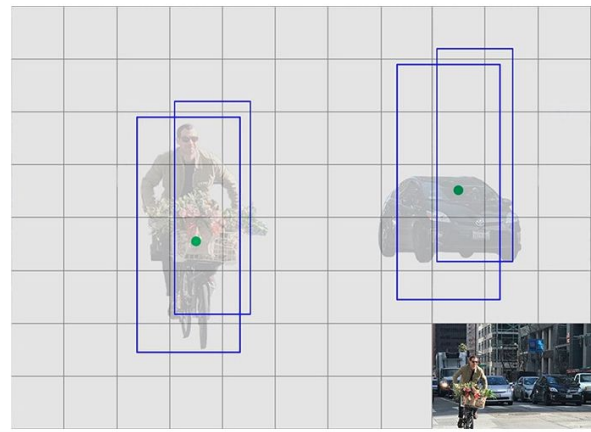
# YOLO: Improvements YOLOv2 (Faster, better, stronger)

## Motivation:

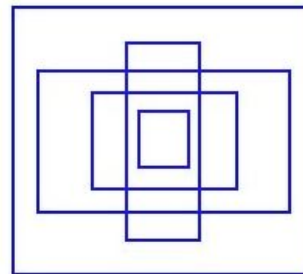
1. Address YOLO v1 downsides.
2. Make it faster.

## Key ideas:

1. Added batch normalization in all layers.
2. Pretrain a classifier network (VGG16) and retrain it end to end with the YOLO output (seen earlier).
3. Integrates anchor boxes (priors). Instead of predicting the bbox directly, offsets are predicted.



Multiple shapes for an object



Anchor boxes

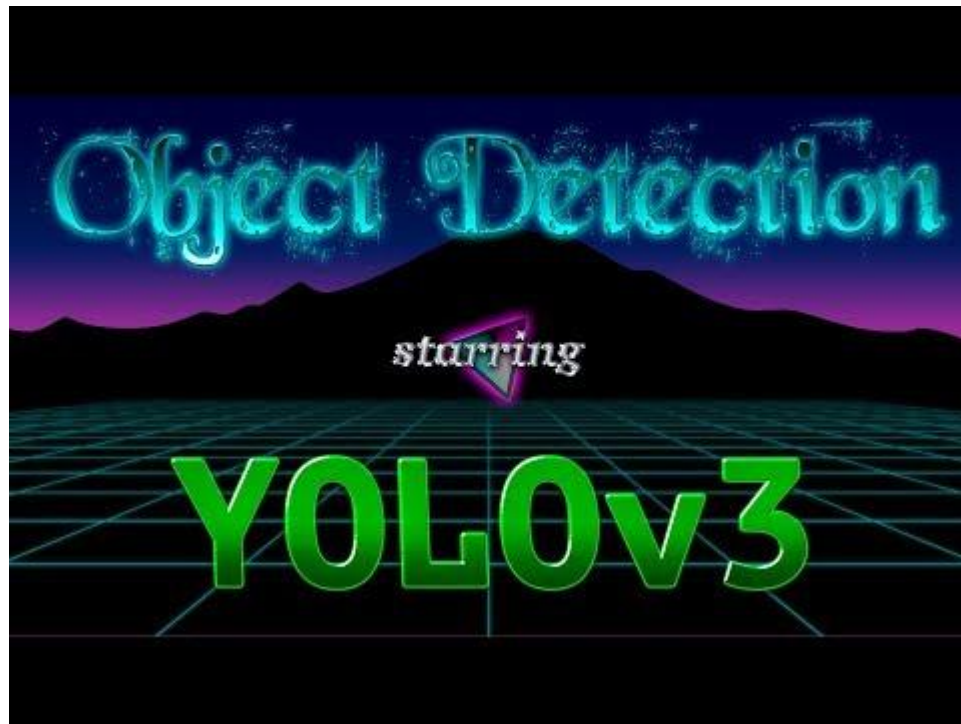




## YOLO: Improvements YOLOv3

### Key ideas:

1. Integrates multi label classification.
2. YOLOv3 uses binary cross-entropy loss for each label.
3. And more details .....





# Thanks...

