

# 인공지능 고급 과정

**YOLO v1**

YOLO v2

YOLO v3

YOLO v4



# YOLO v1



## You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon\*, Santosh Divvala\*†, Ross Girshick†, Ali Farhadi\*†

University of Washington\*, Allen Institute for AI†, Facebook AI Research†

### Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

기존과 다른 접근방식 (1-stage detector).

Region proposal 방식이 아닌 한번에 예측.

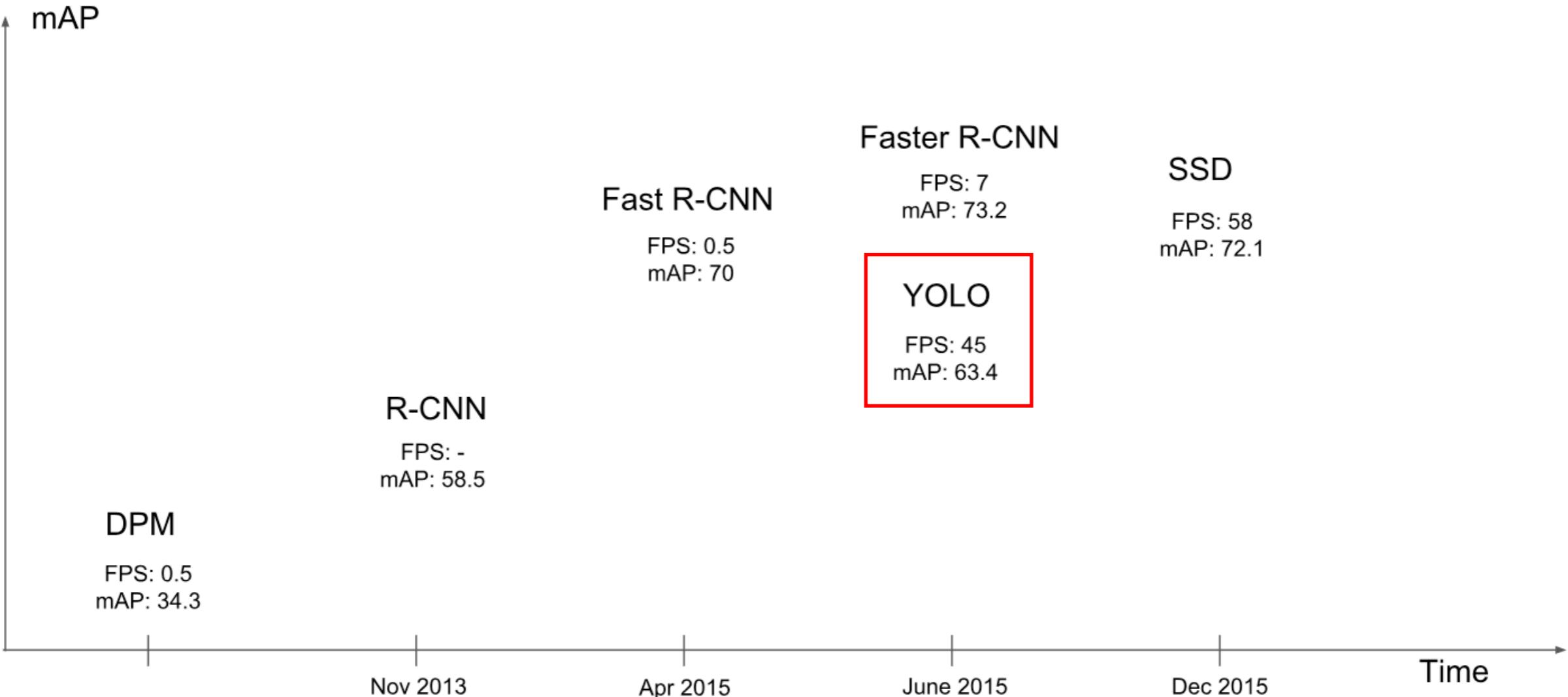
극도로 빠르며, 정확도도 기존 Real-Time Detector보다 상당히 좋음.

Image 전체를 보기 때문에 낮은 background error (False positive)를 보임

## 논문에서 강조하고 있는 YOLO의 특징들

논문 링크 주소 : <https://arxiv.org/pdf/1506.02640.pdf>

# YOLO v1



# YOLO v1

R-CNN과 같은 이전의 object detection 모델과 다르게 YOLO는 1-stage detector로 하나의 네트워크로 detection을 수행한다. 하나의 네트워크를 사용하므로 end-to-end 학습이 가능하다.

YOLO v1의 특징은 다음과 같다.

## 1. 빠르다.

실시간으로 45프레임을 처리할 수 있고, fast YOLO는 초당 155프레임을 처리 할 수 있다. 기존의 다른 real-time detector들 보다 2배 이상의 mAP를 보여 준다.

## 2. 추론 시 이미지를 전역적으로 파악한다.

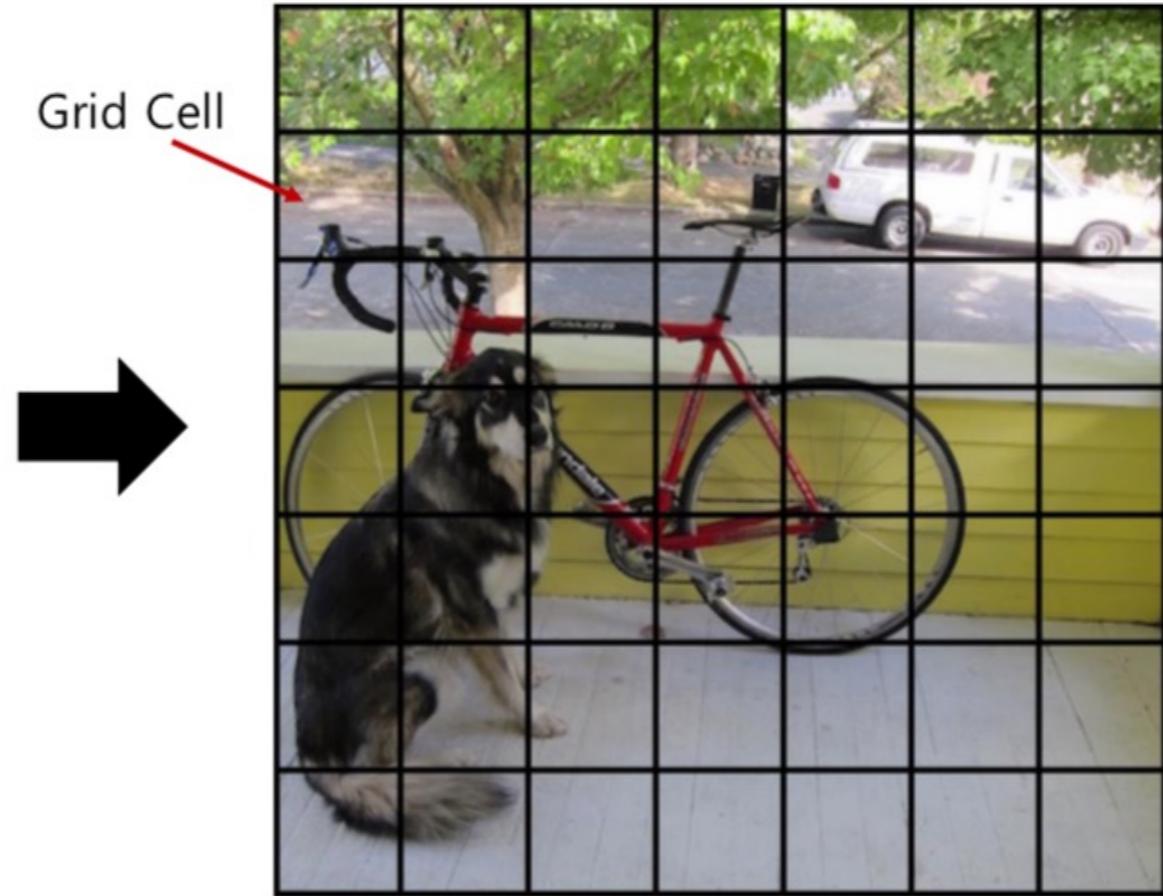
기존의 sliding window와 region proposal과 다르게 이미지의 전체를 본다. 따라서 해당 클래스의 맥락적 정보 뿐만 아니라 모습(appearance)을 encode 해서 fast R-CNN보다 background error를 절반 가량 줄일 수 있다.

## 3. 객체의 일반화 가능한 표현(representation)을 익힌다.

자연 이미지(natural image)로 학습한 후 그림(artwork)에서 test를 진행해도 다른 모델들 보다 좋은 성능을 보인다.

\* 2 stage모델보다 Accuracy가 떨어지지만 속도와 전역적인 특성을 파악하여 background error를 줄일 수 있다.

# YOLO v1 – Grid Cell



이미지를 일정한 크기로 나누고 이를 이용하여 예

측

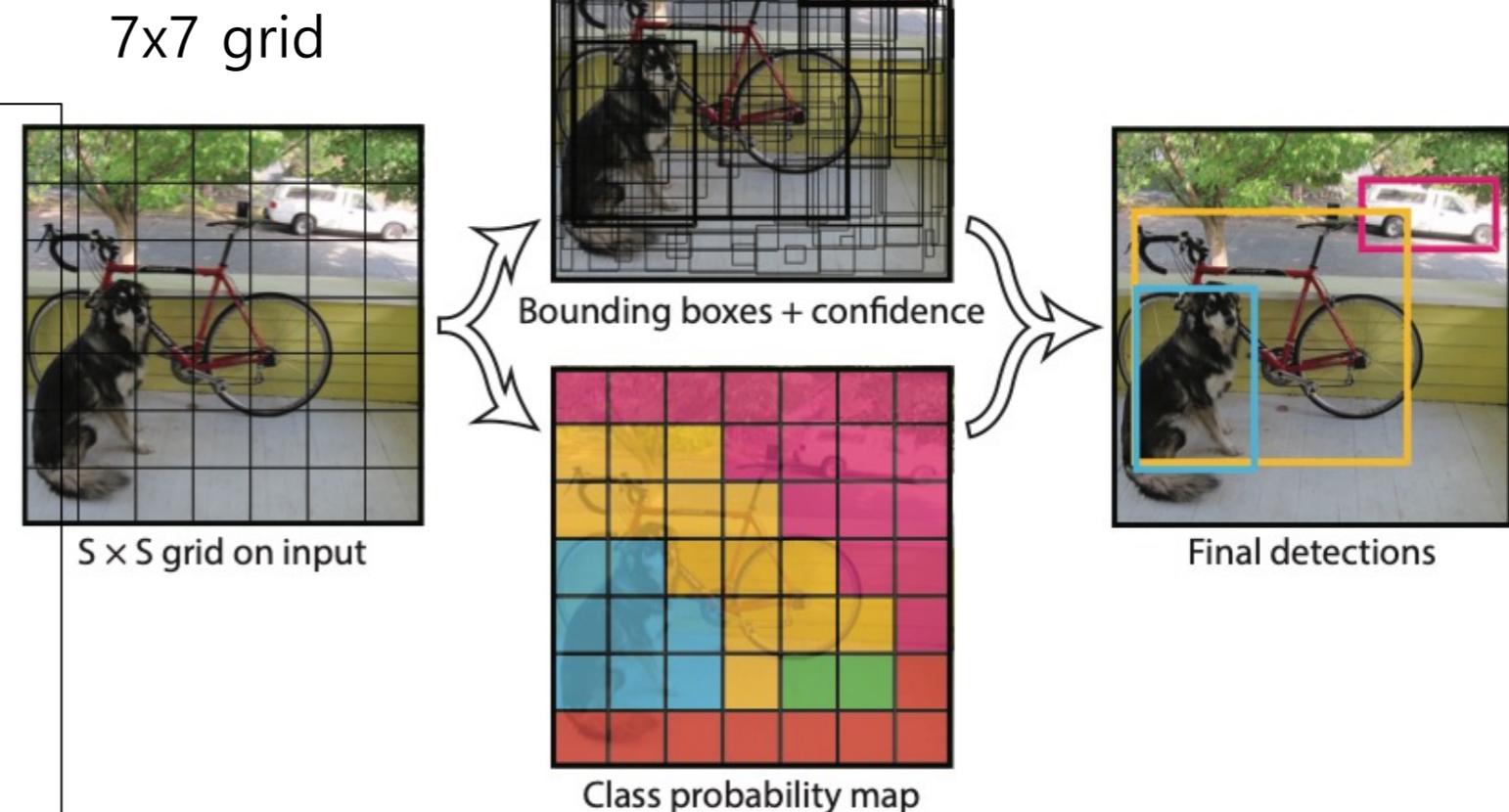
# YOLO v1 – Unified Detection

B : Grid당 바운딩박스의 갯수  
(YOLO v1에서는 2개)

C : Class의 갯수

클래스의 수가 20개라고 가정하면  
출력은  $7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30$   
이 된다.

본 논문에서는 20개의 클래스를  
갖는 VOC 데이터셋을 사용 했다.



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.

# YOLO v1 – Unified Detection

각각의 **bounding box**는  $x,y,w,h,confidence$ 를 예측한다.

$x,y$ 는 중심 좌표를 나타내고,

$w,h$ 는 높이와 너비를 나타낸다.

Confidence score는 박스에 물체를 존재할 확률과  
얼마나 정확하게 bounding Box를 예측했는지를 나타낸다.

$$confidence = \frac{Pr(object)}{\text{박스안에 물체가 존재할 확률}} * \frac{IOU(pred, truth)}{\text{얼마나 정확하게 bounding Box를 예측했는지}}$$

박스안에 물체가 없으면 confidence는 0이 되고, 물체가 존재할 확률이 있으면,  
IoU값과 확률값을 곱을 해서 confidence값을 결정한다.

# YOLO v1 – Unified Detection

그러나, 각 Grid Cell C개(Class갯수)의 conditional class probabilities  $P_r(class_i | object)$ 를 예측한다.  
Bounding box의 수와 관계없이 하나의 grid cell에서 클래스별 확률을 모두 예측한다.  
→ Class-specific confidence

$$Pr(Class_i|object) * Pr(object) * IoU(pred, truth) = Pr(class_i) * IoU(pred, truth)$$

class-specific confidence는 class가 box에 등장할 확률과 객체가 예측된 box에 얼마나 잘 맞는지  
를 나타낸다.

예측시에 출력 되는 최종 텐서의 수를 다시 기억하자.

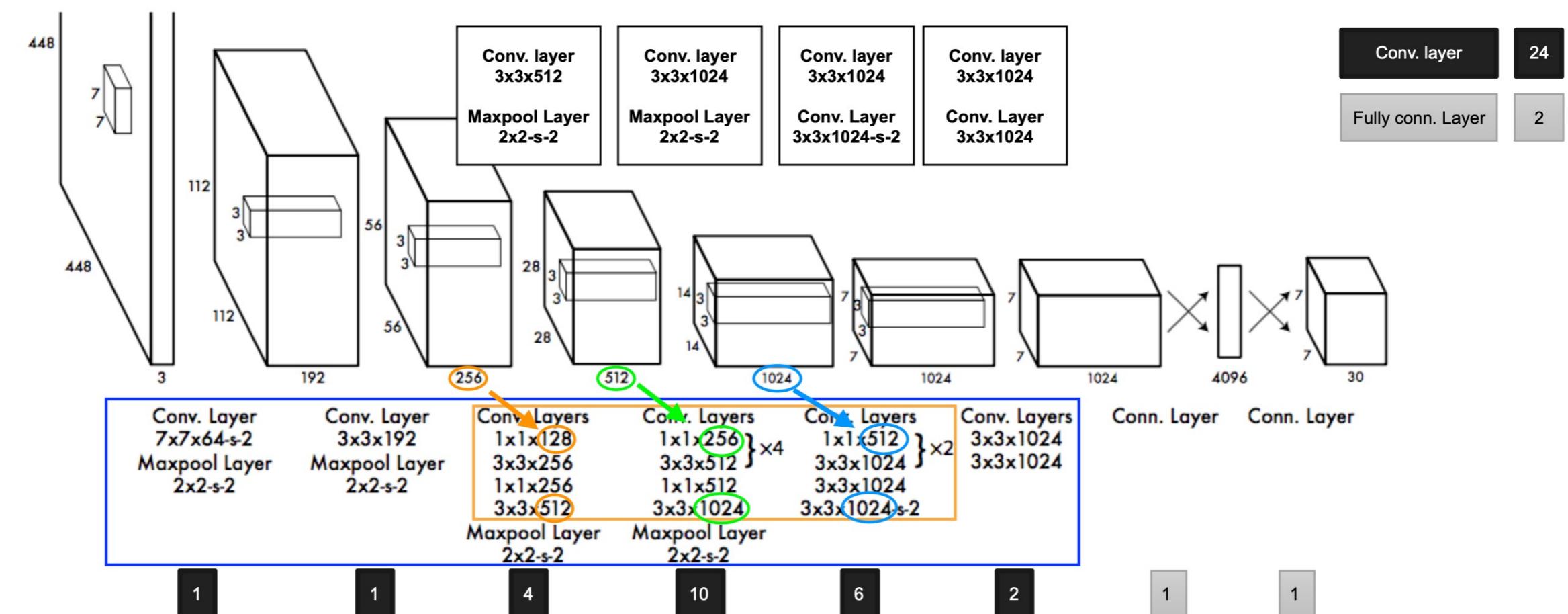
$$S \times S \times (B * 5 + C) \text{ tensor.}$$

# YOLO v1 – Network Design

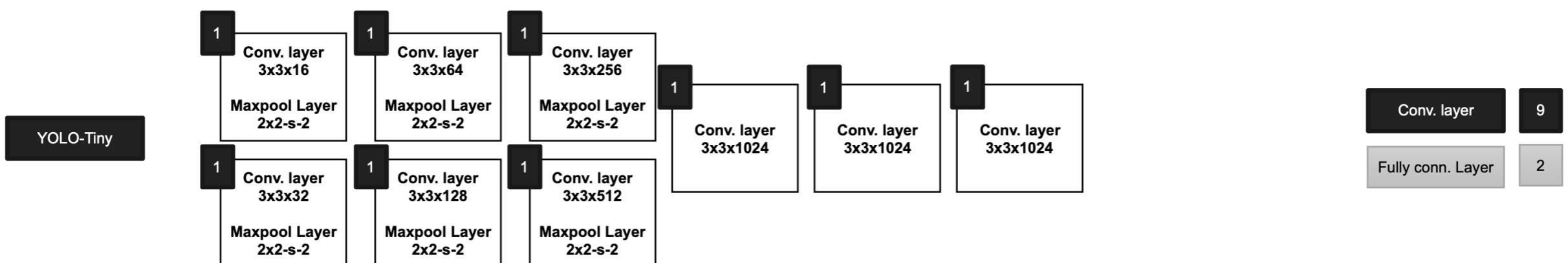
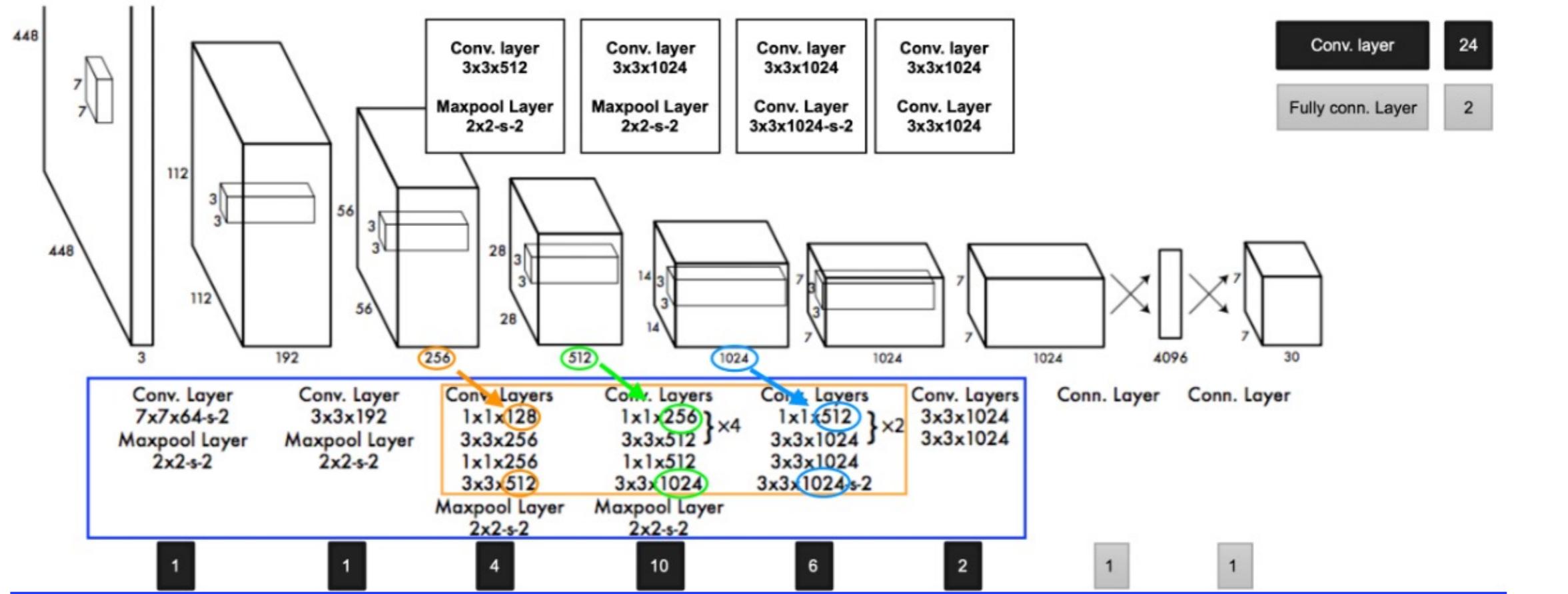
A network graph visualization consisting of numerous small, semi-transparent grey dots representing nodes, connected by thin grey lines representing edges. The nodes are scattered across the frame, with a higher density in the center and fewer in the periphery. Some nodes have multiple connections to other nodes, while others are isolated or have only one connection.

# Modified GoogleNet

Our network architecture is inspired by the GoogLeNet model for image classification [34]. Our network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by GoogLeNet, we simply use  $1 \times 1$  reduction layers followed by  $3 \times 3$  convolutional layers, similar to Lin et al [22]. The full network is shown in Figure 3.



# YOLO v1 – Fast YOLO



**Yolo v2부터는 tiny라는 표현으로 변경**

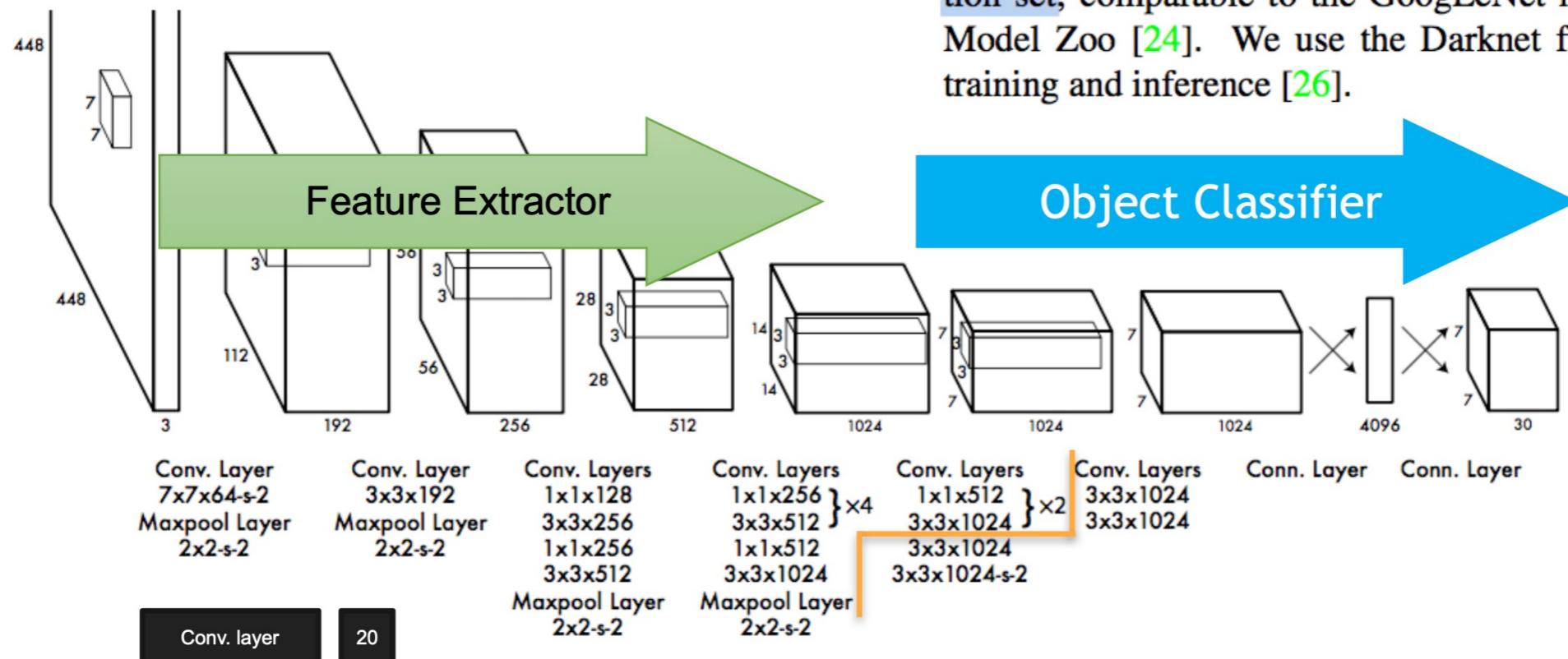
# YOLO v1 – Fast YOLO

## Training

- 1) Pretrain with ImageNet 1000-class competition dataset

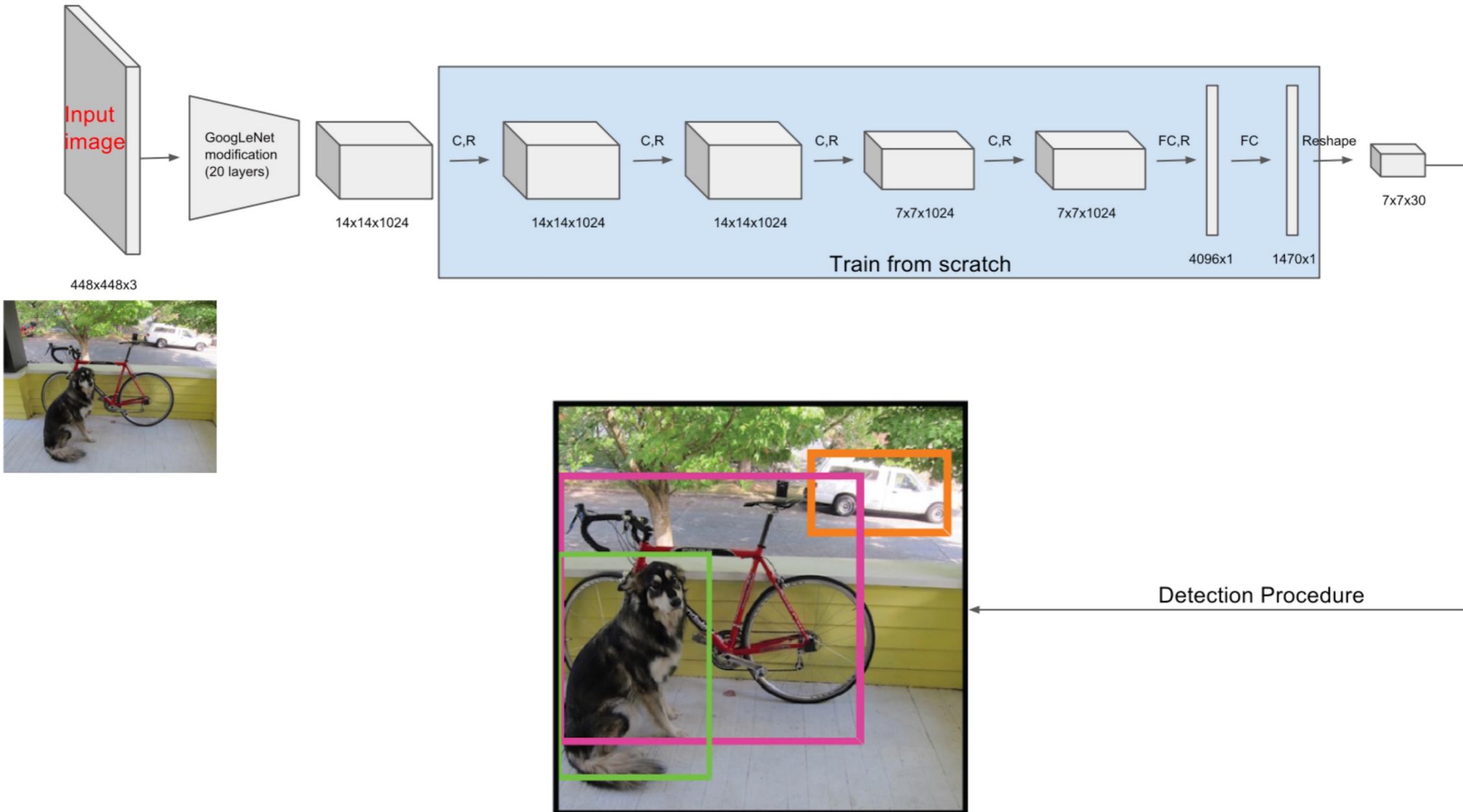


We pretrain our convolutional layers on the ImageNet 1000-class competition dataset [30]. For pretraining we use the first 20 convolutional layers from Figure 3 followed by a average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe’s Model Zoo [24]. We use the Darknet framework for all training and inference [26].

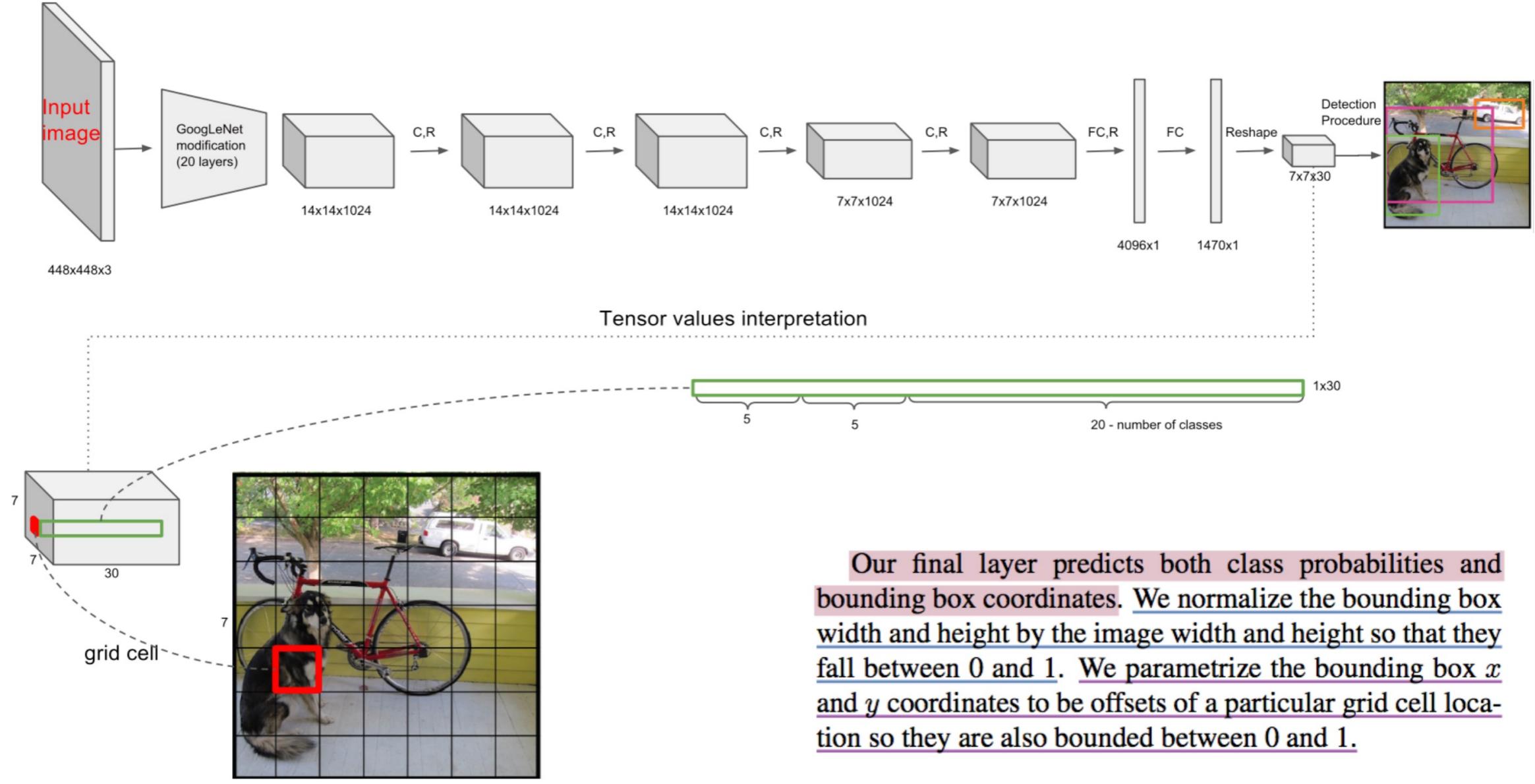


**Figure 3: The Architecture.** Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating  $1 \times 1$  convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution ( $224 \times 224$  input image) and then double the resolution for detection.

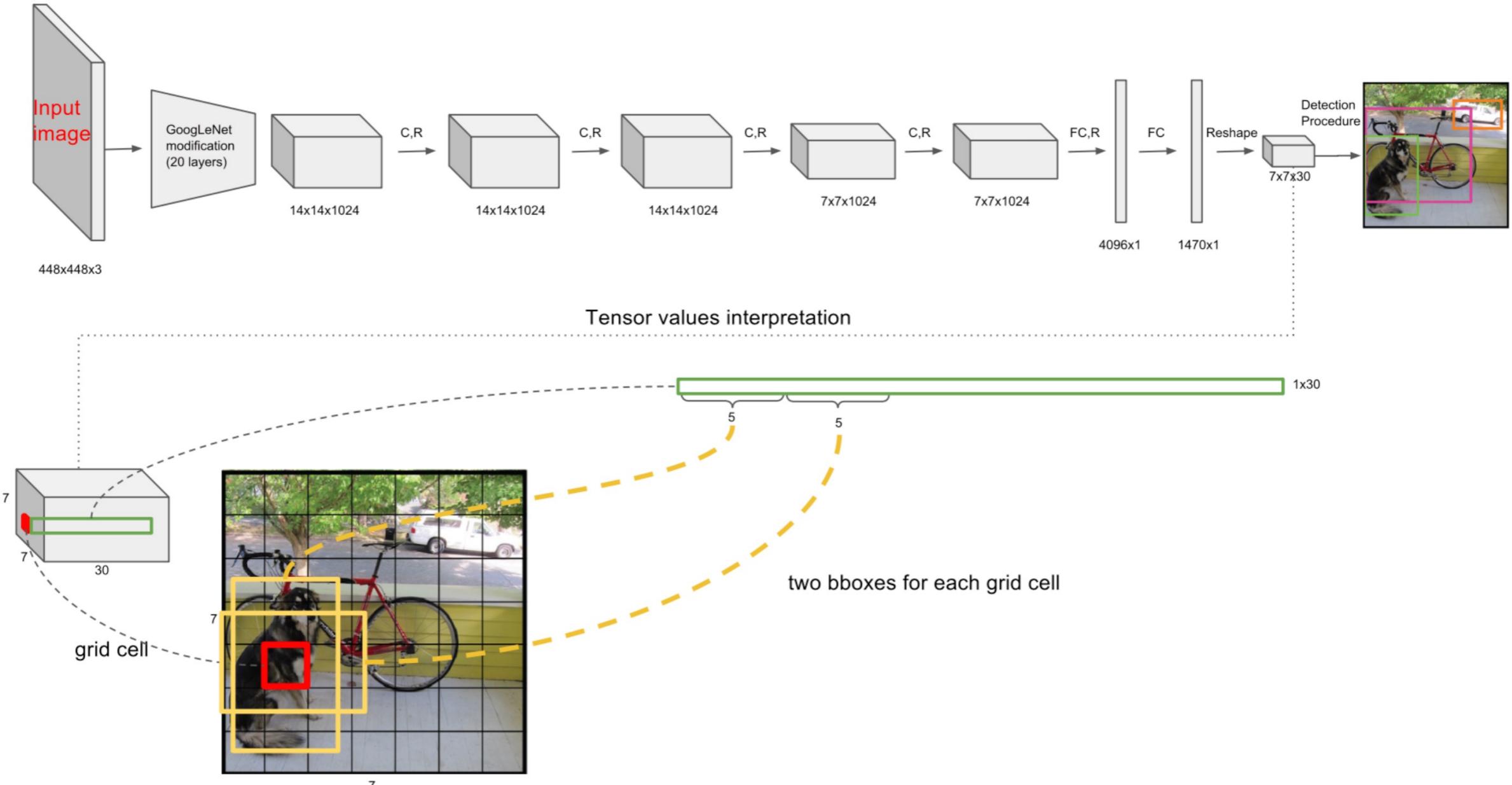
# YOLO v1 – Fast YOLO



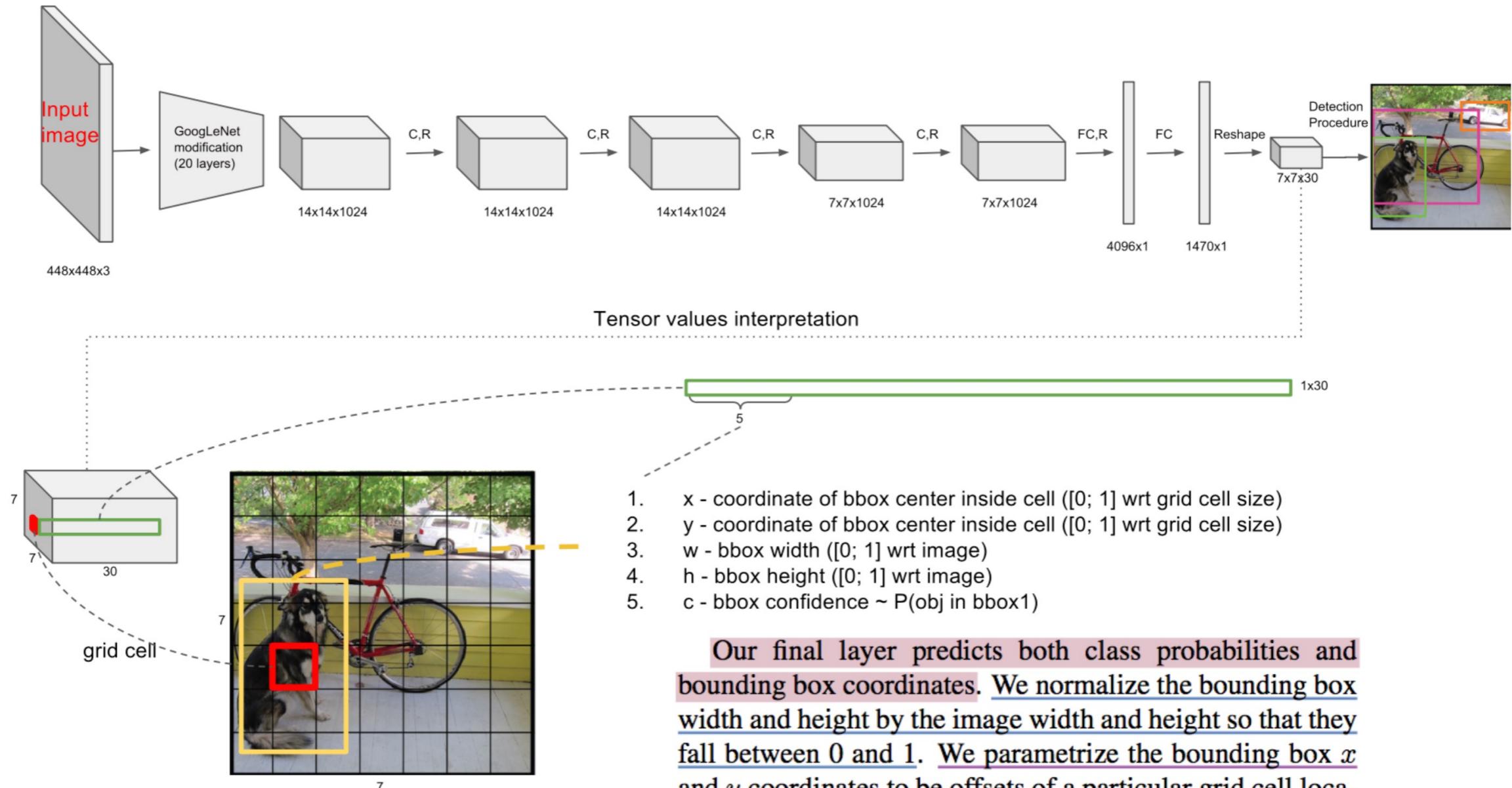
# YOLO v1 – Architecture



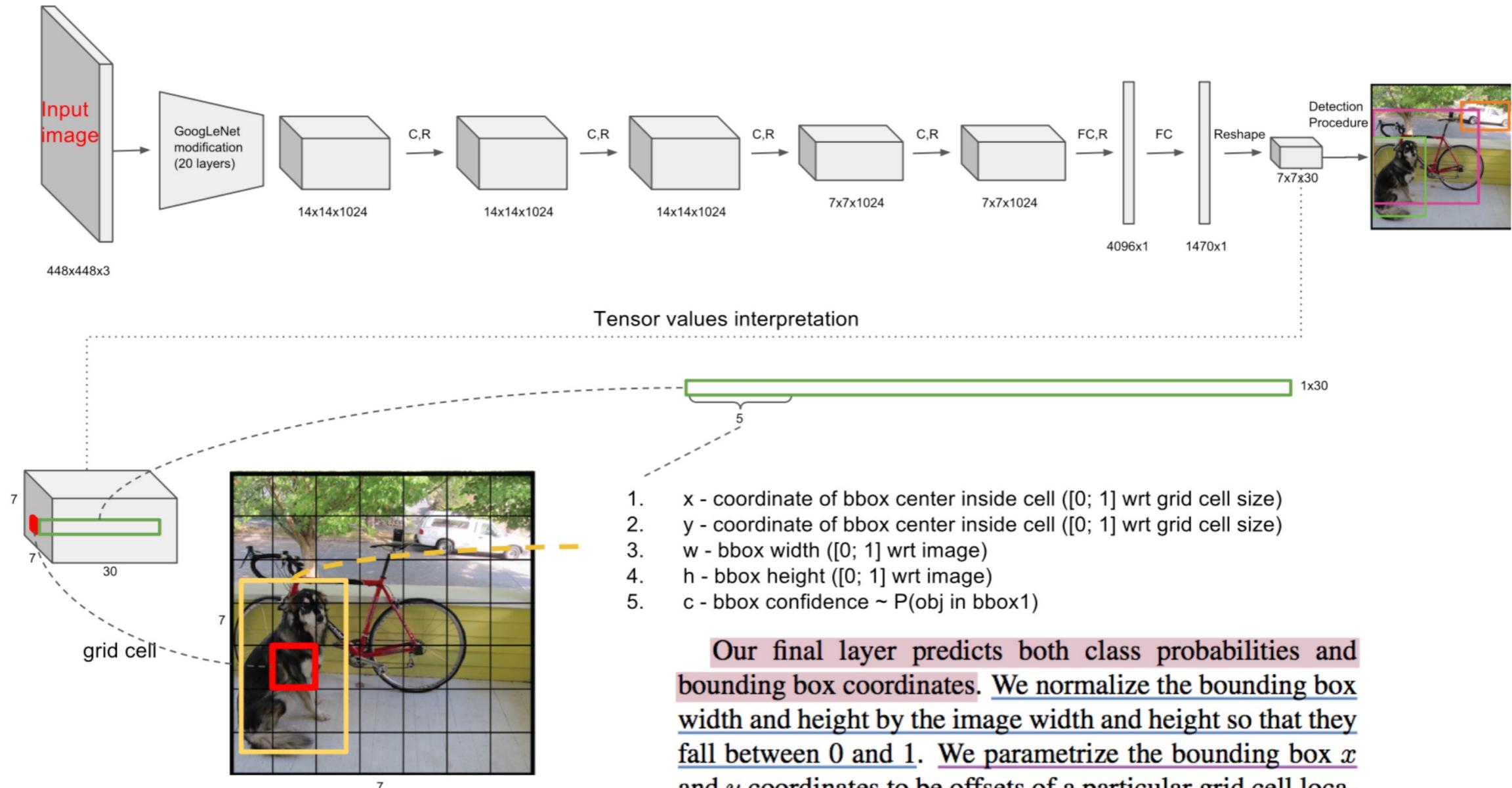
# YOLO v1 – Architecture



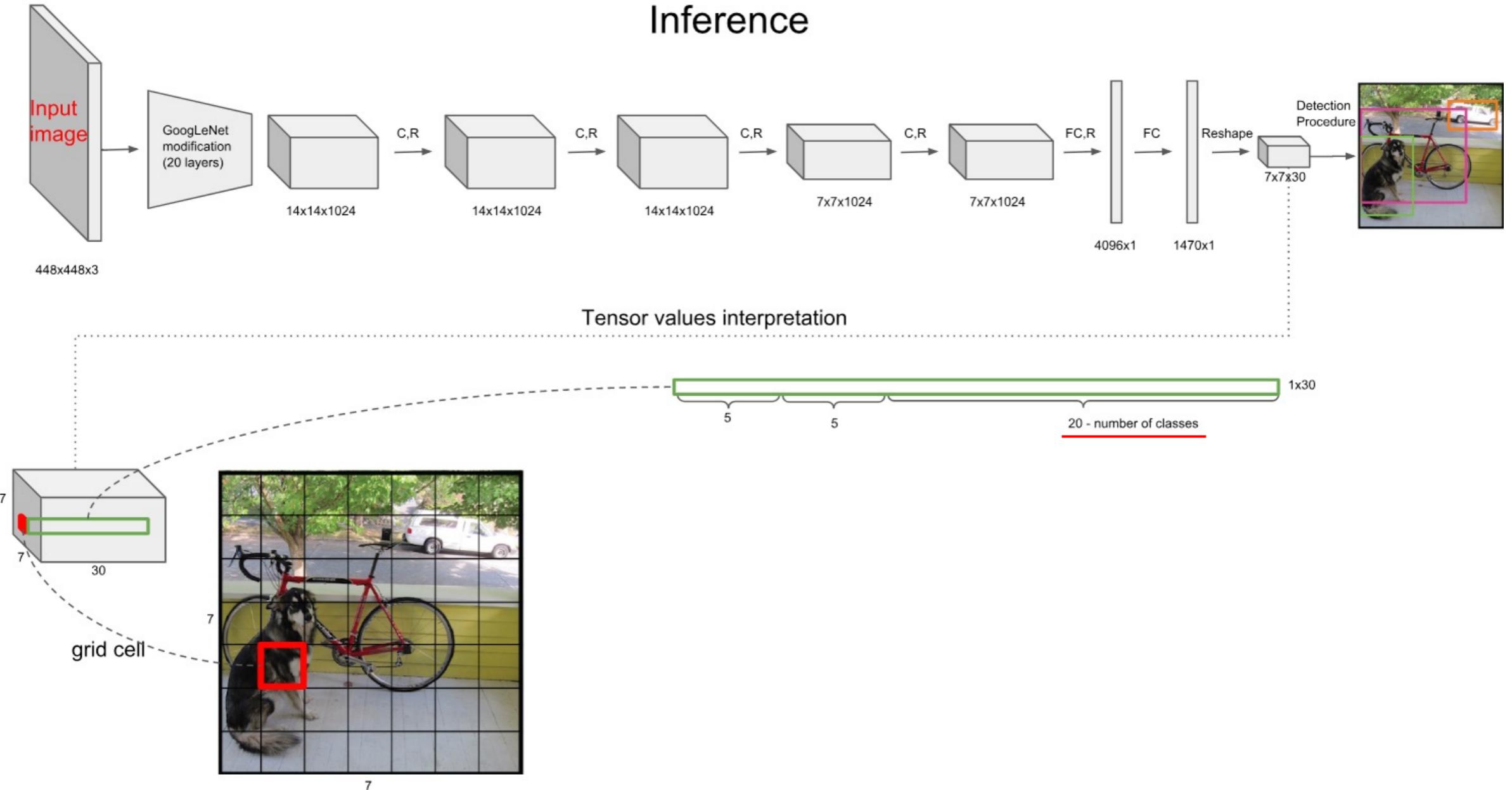
# YOLO v1 – Architecture



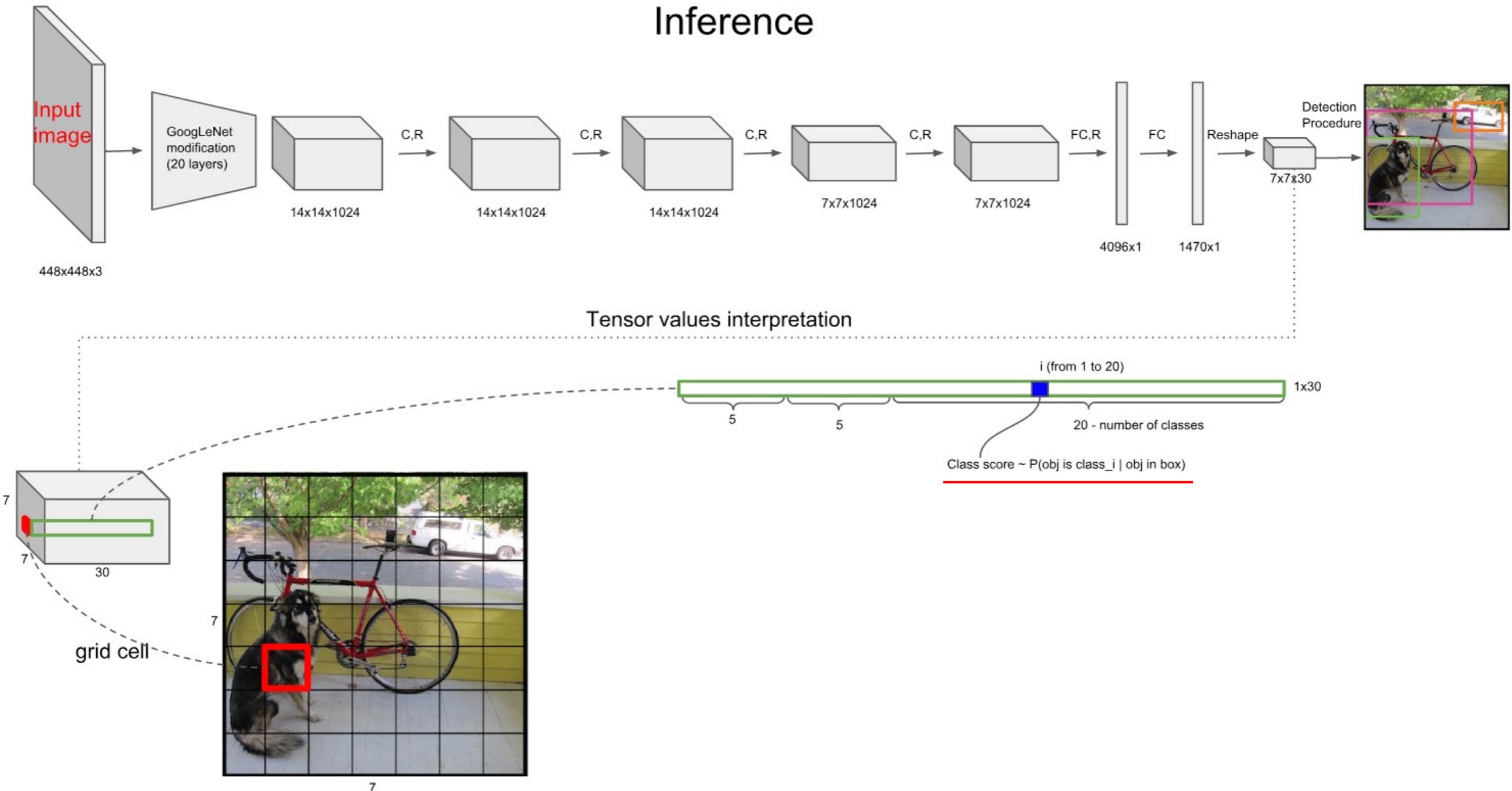
# YOLO v1 – Architecture



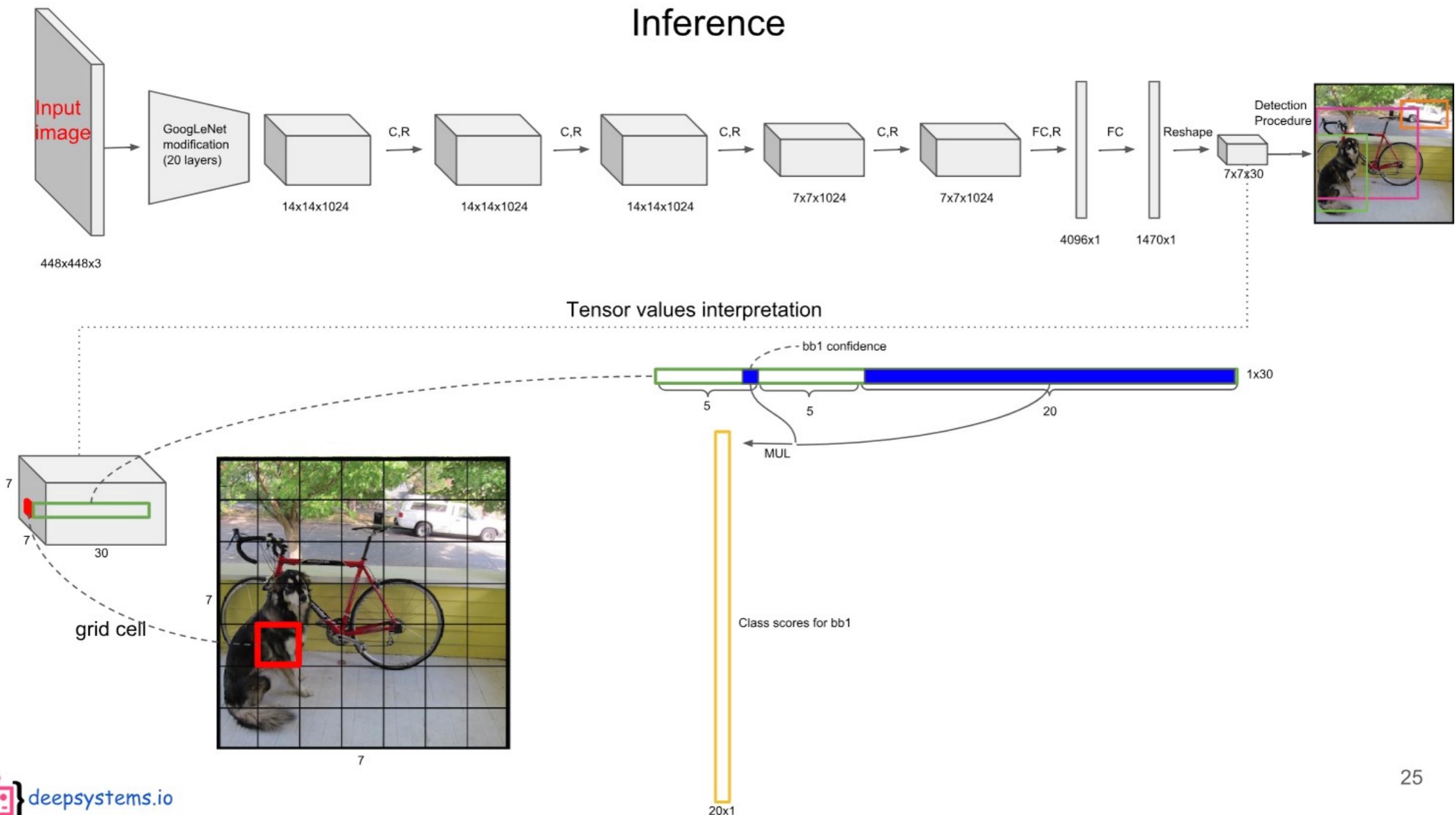
# YOLO v1 – Architecture



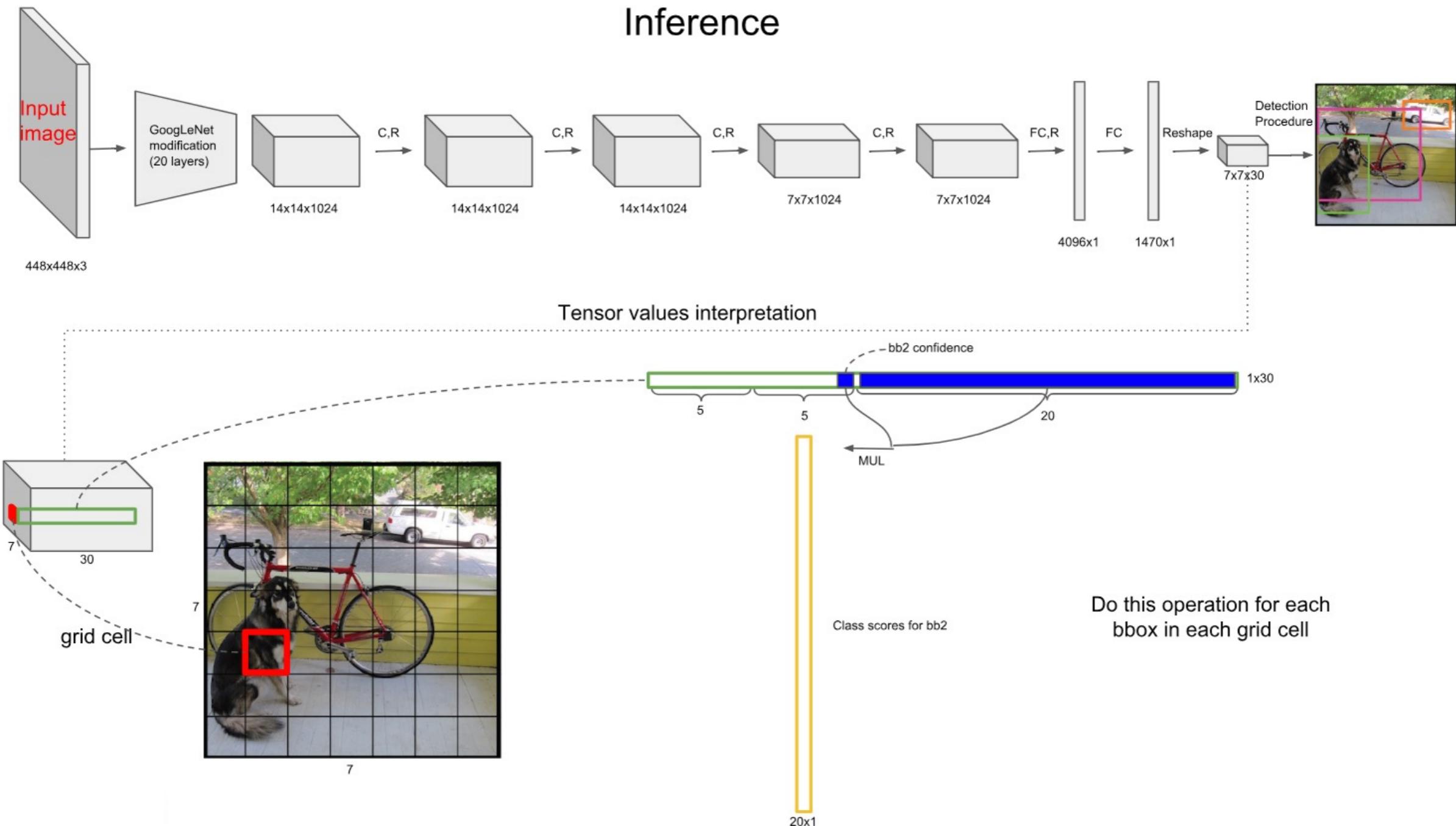
# YOLO v1 – Architecture



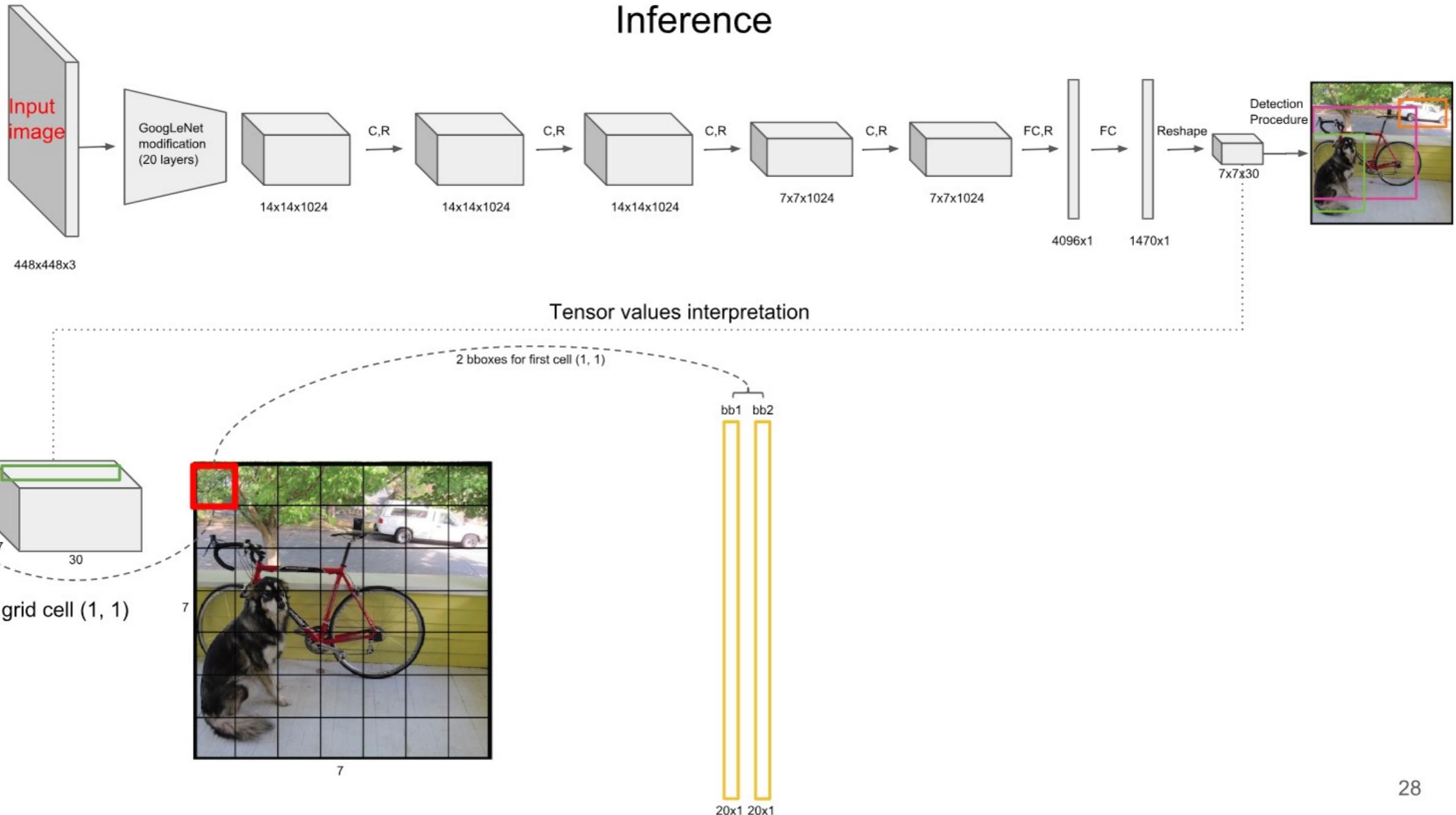
# YOLO v1 – Architecture



# YOLO v1 – Architecture

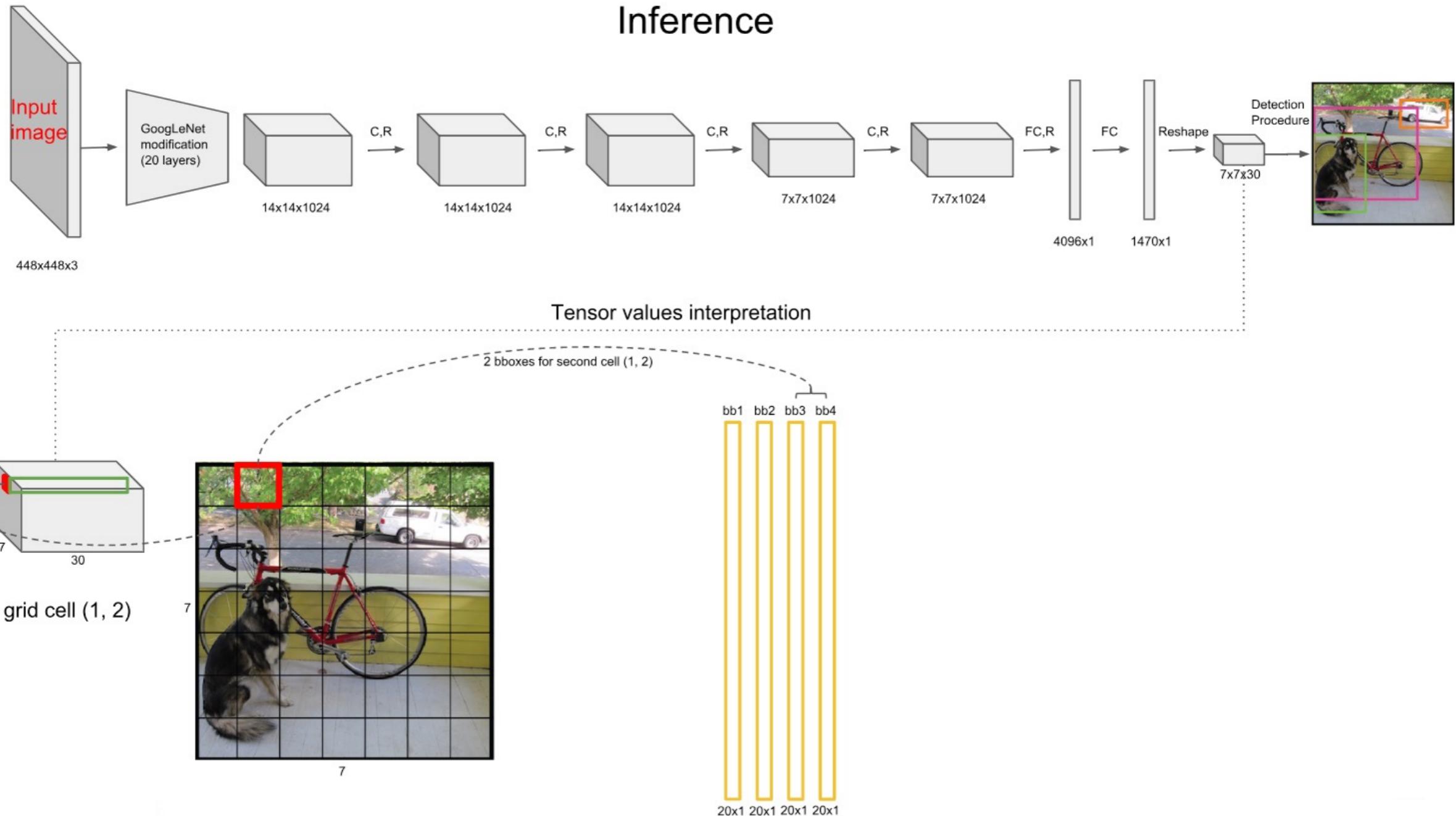


# YOLO v1 – Architecture

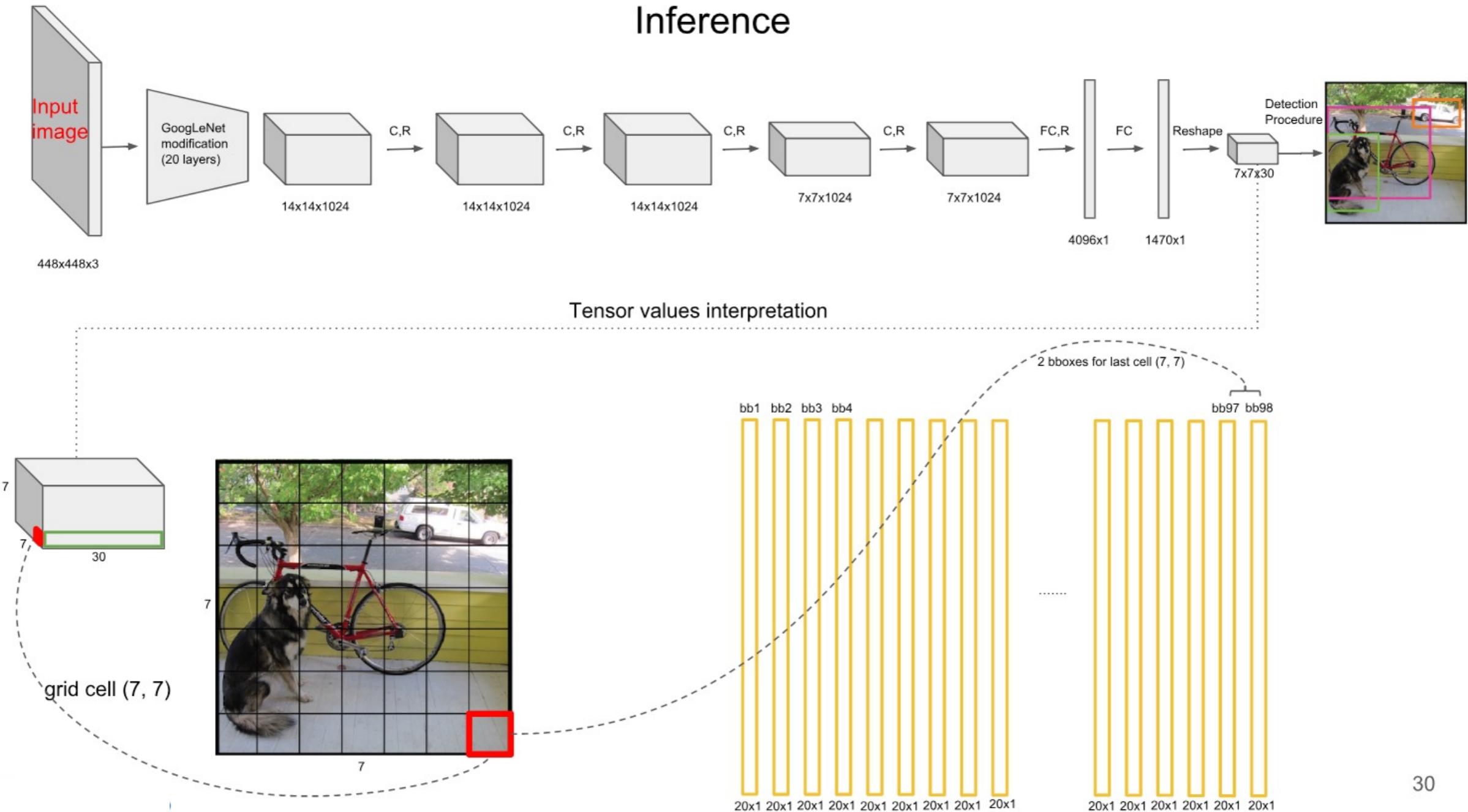


28

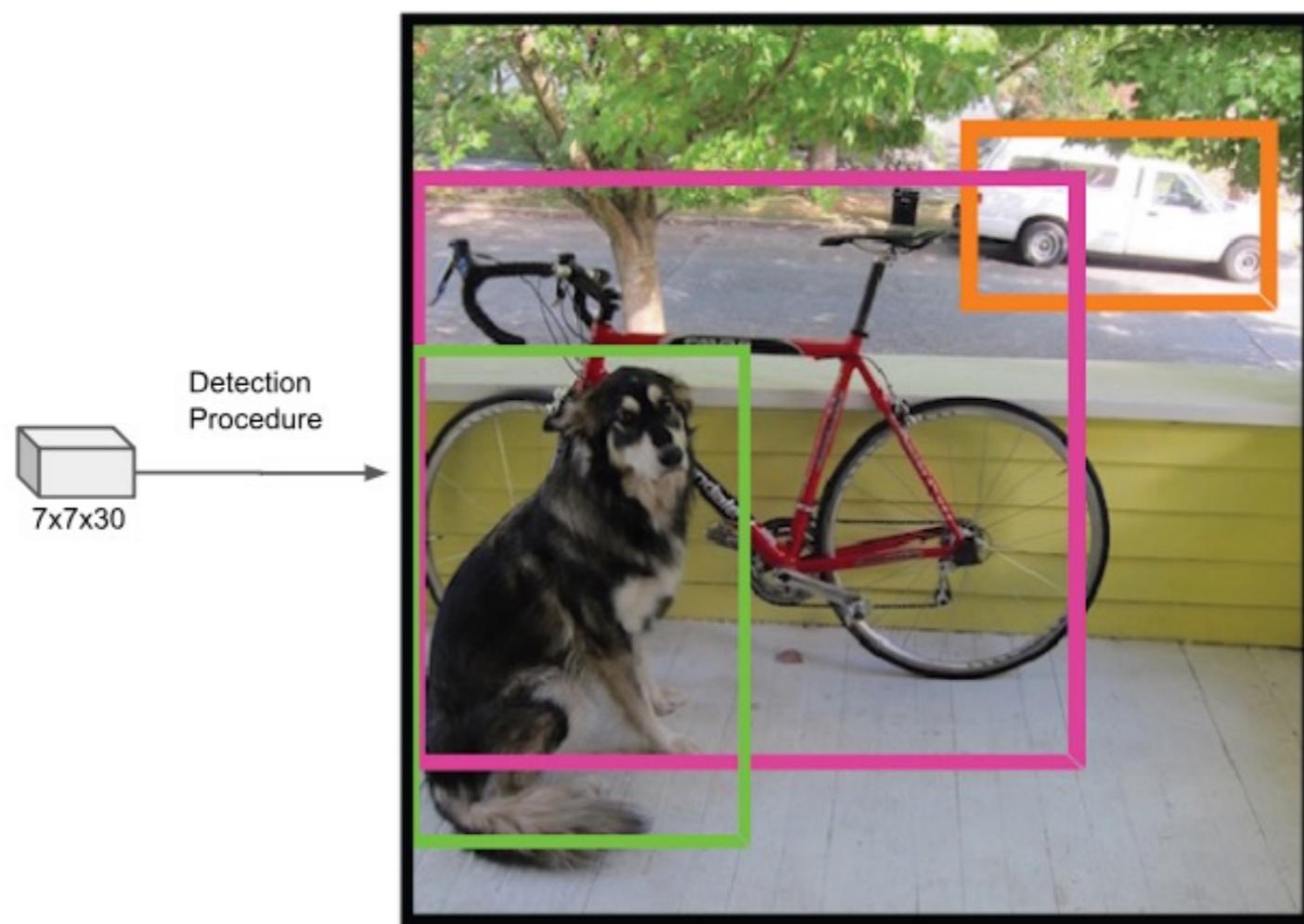
# YOLO v1 – Architecture



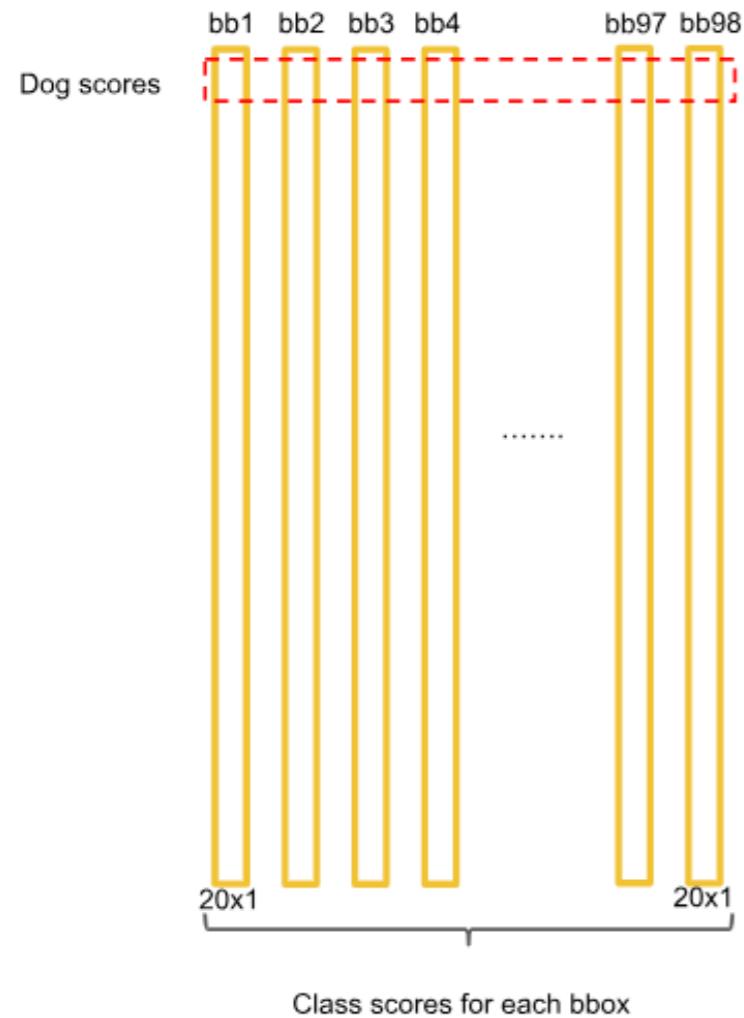
# YOLO v1 – Architecture



# YOLO v1 – Architecture

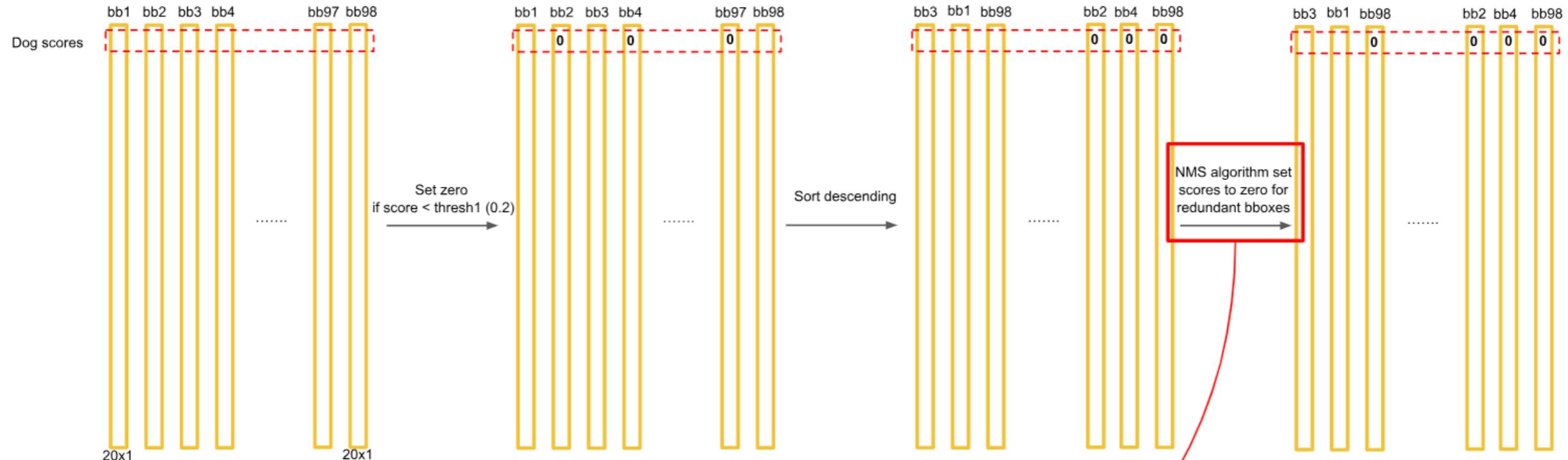


# YOLO v1 – Architecture



**Get first class scores for each bbox**

# YOLO v1 – Architecture

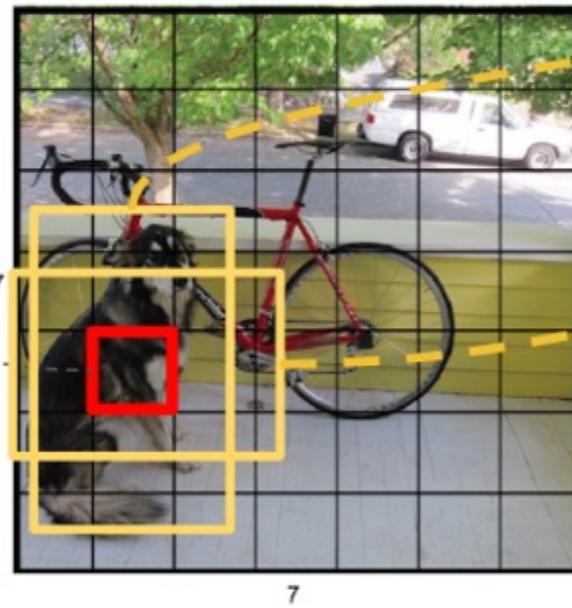


How it works

# YOLO v1 – NMS

## NMS란?

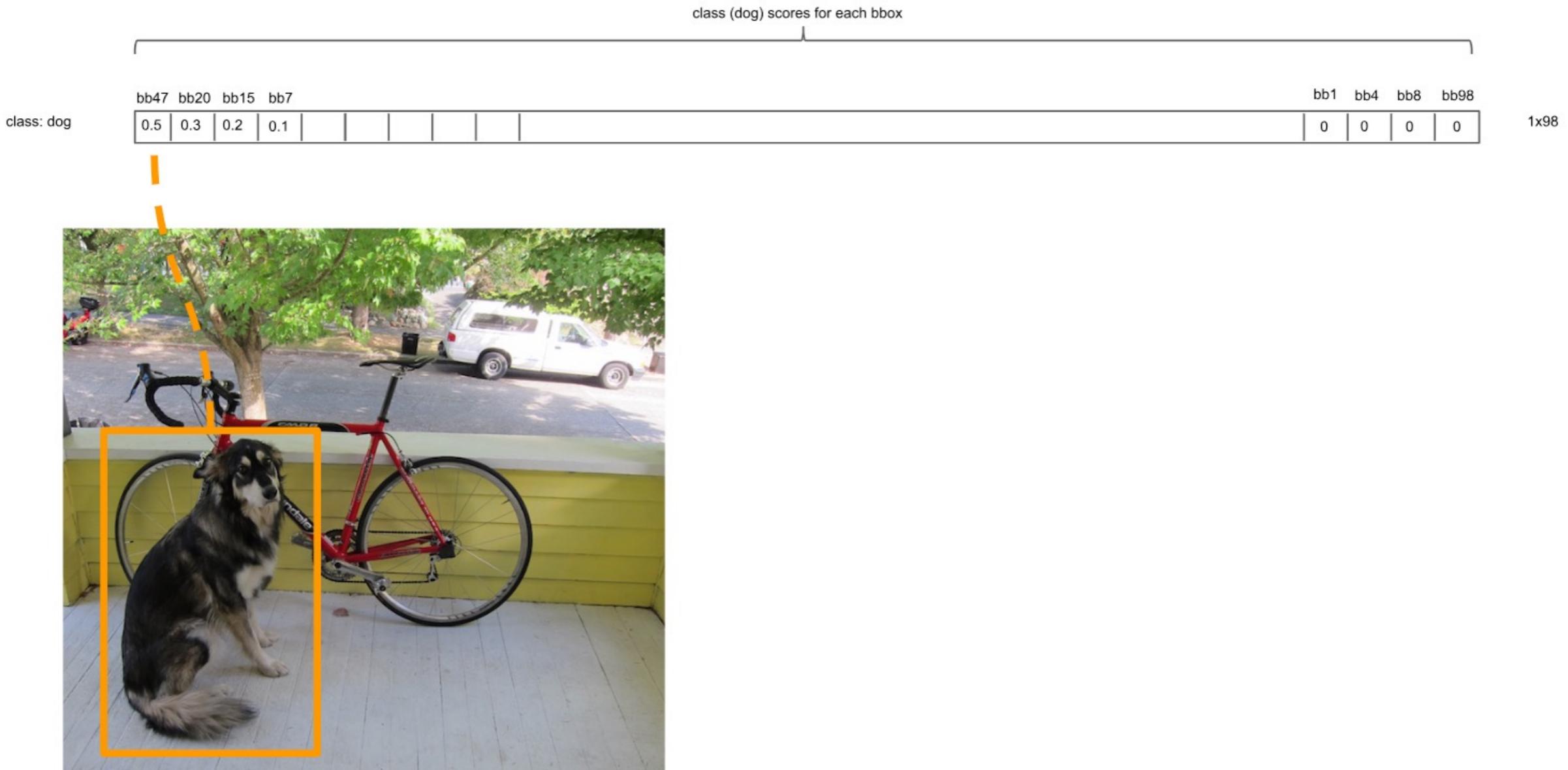
뉴럴 네트워크가 예측한 bounding boxes 중 많이 겹치는 박스들은 제거하는 알고리즘이다.



## Non-Maximum Suppression 이해하기

1. 예측한 Bounding box 들을 예측 점수(confidence/score) 내림차순 정렬 한다.
2. 높은 점수를 가진 박스부터 시작해서 나머지 박스들과의 IoU 값을 계산한다.
3. IoU 값이 threshold보다 높은 박스들은 리스트에서 제거한다.
4. 다음으로 높은 점수를 가진 박스로 Step 2부터 반복한다.

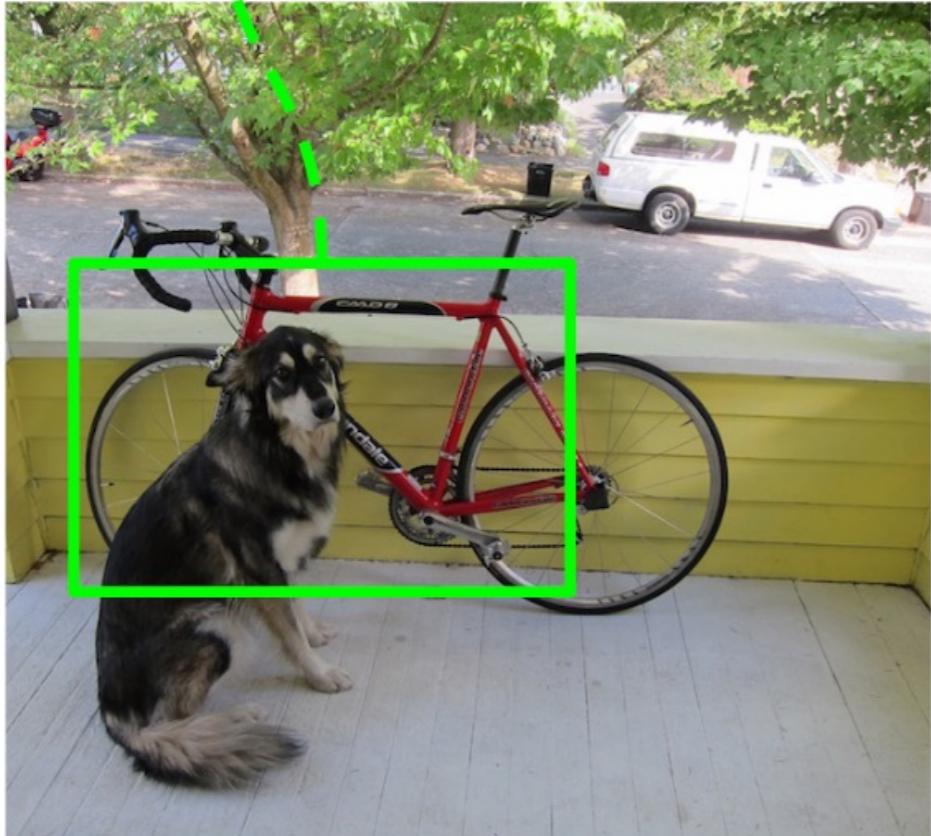
# YOLO v1 – NMS



# YOLO v1 – NMS

class (dog) scores for each bbox

	bb47	bb20	bb15	bb7												bb1	bb4	bb8	bb98
class: dog	0.5	0.3	0.2	0.1												0	0	0	0



# YOLO v1 – NMS



class (dog) scores for each bbox

class: dog

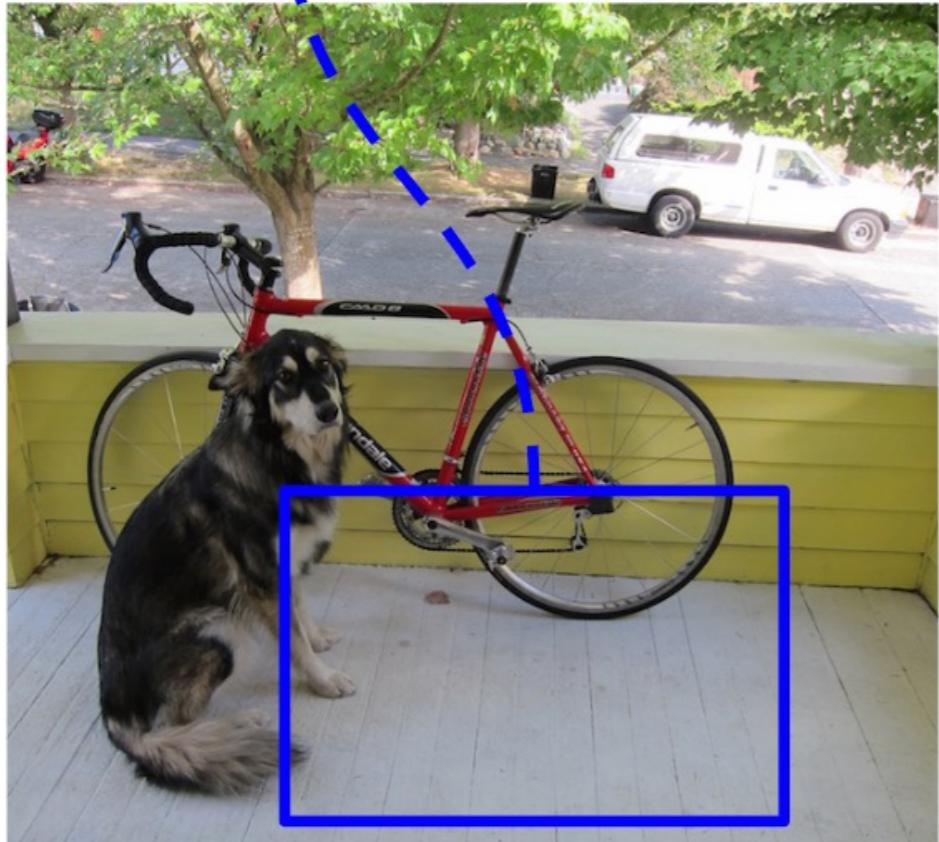
bb47 bb20 bb15 bb7

0.5	0.3	0.2	0.1					
-----	-----	-----	-----	--	--	--	--	--

bb1 bb4 bb8 bb98

0	0	0	0
---	---	---	---

1x98

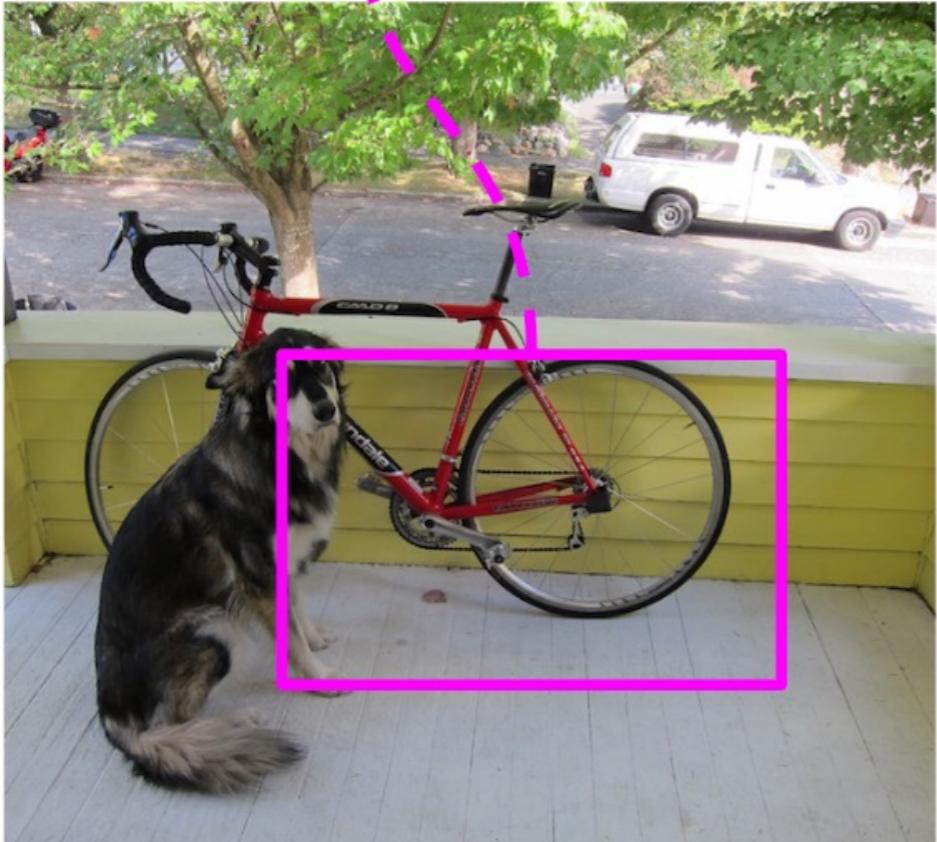


# YOLO v1 – NMS

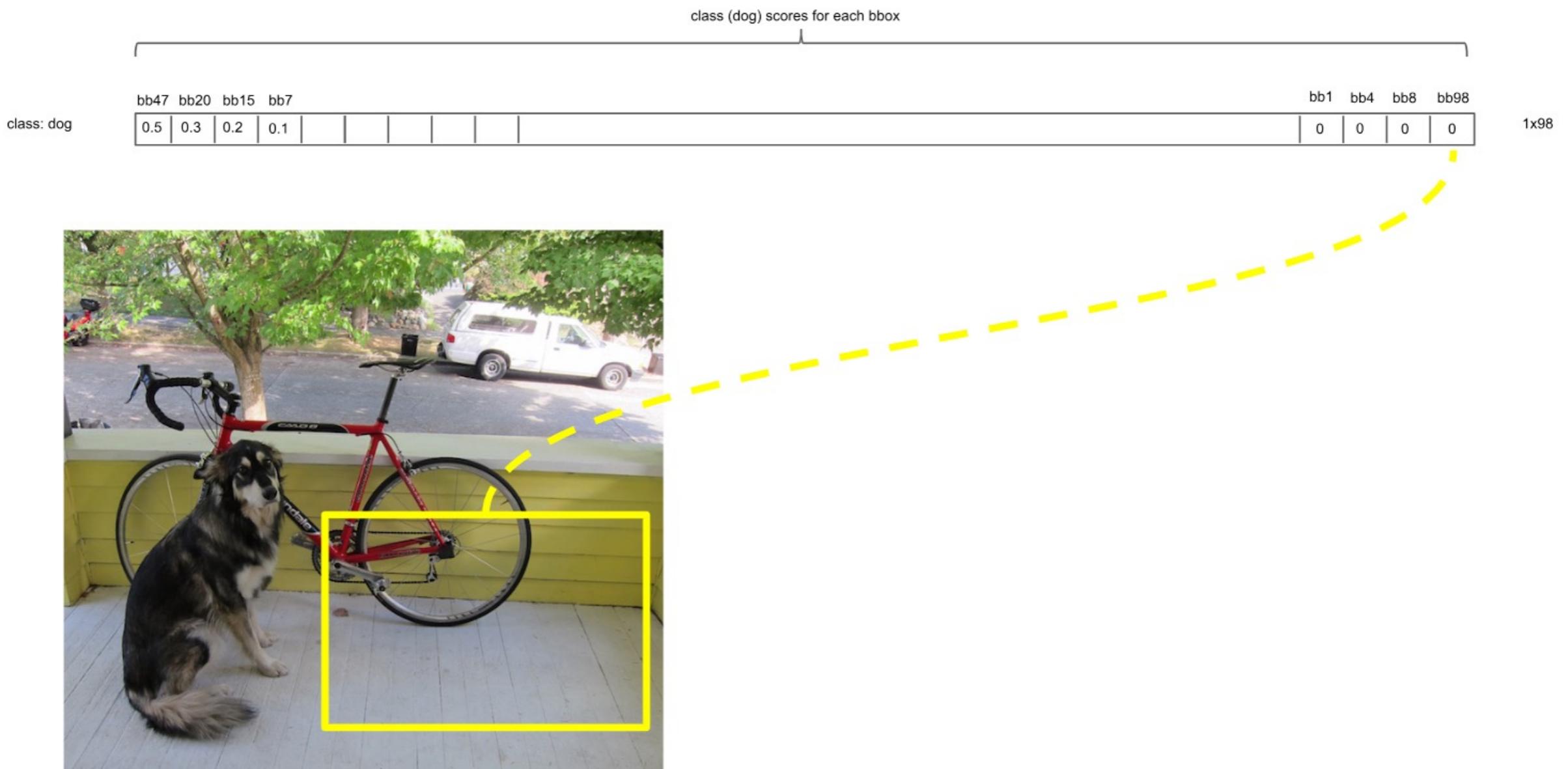


class (dog) scores for each bbox

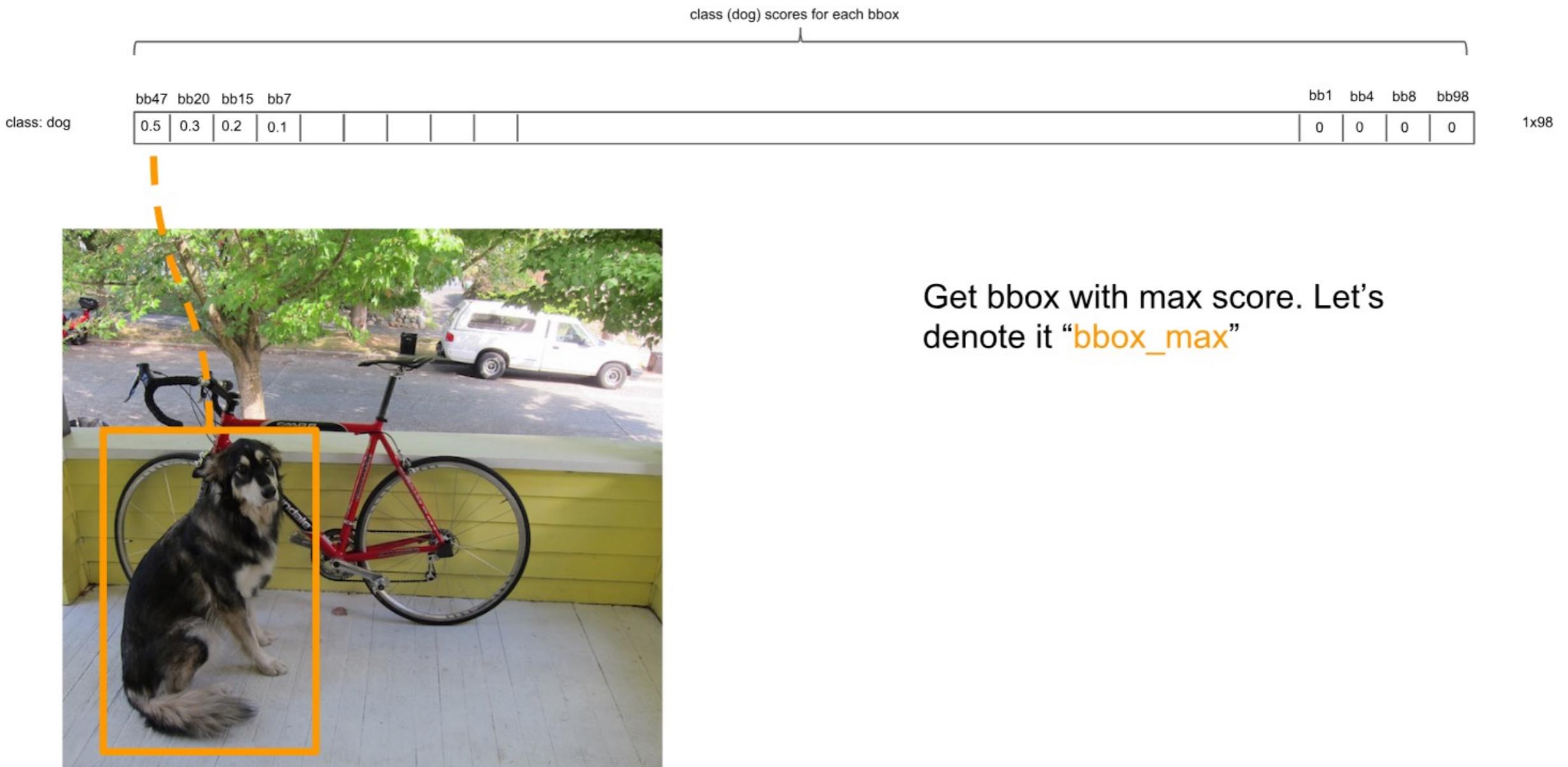
class: dog	bb47	bb20	bb15	bb7												bb1	bb4	bb8	bb98	1x98
	0.5	0.3	0.2	0.1												0	0	0	0	



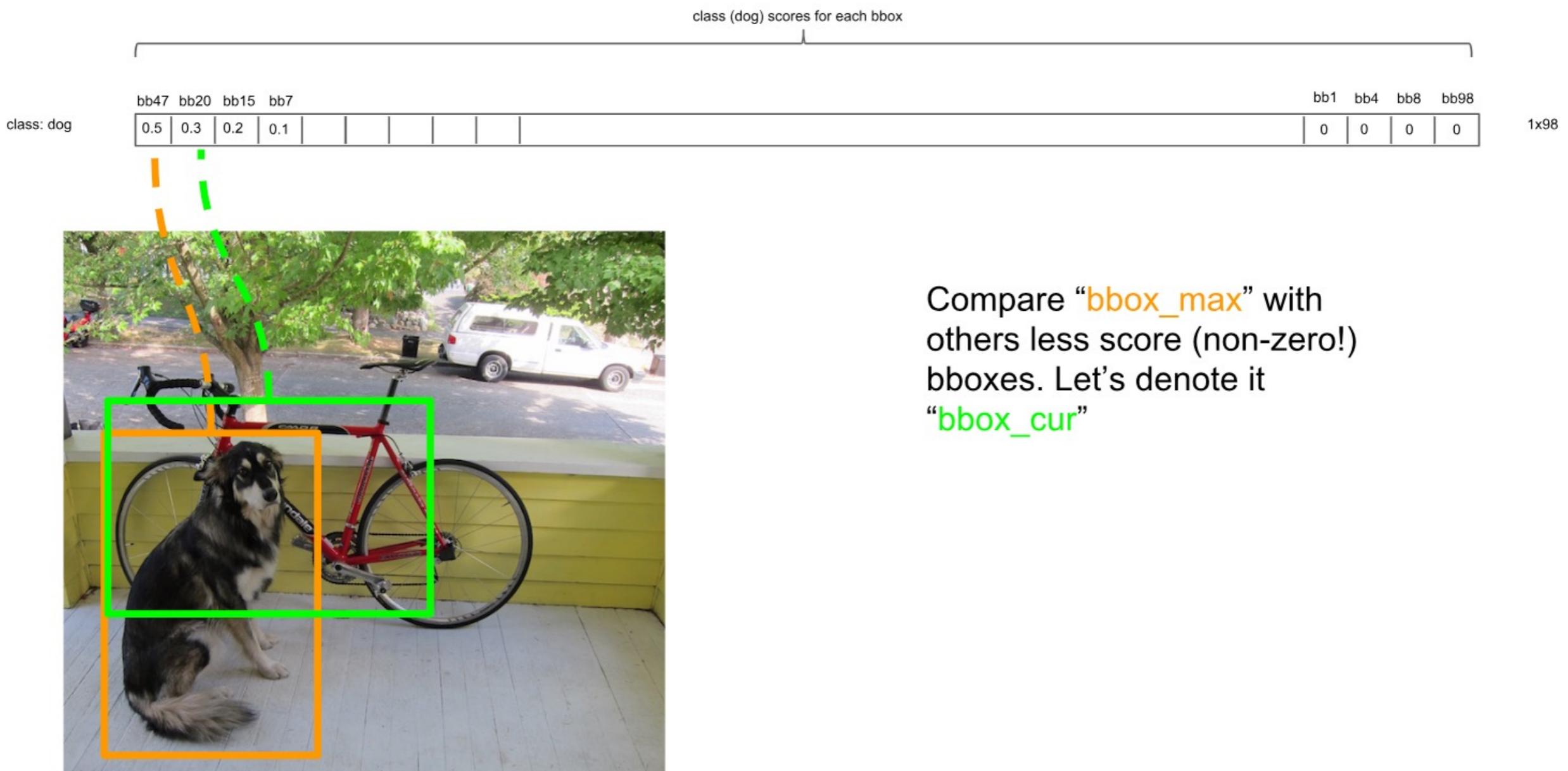
# YOLO v1 – NMS



# YOLO v1 – NMS



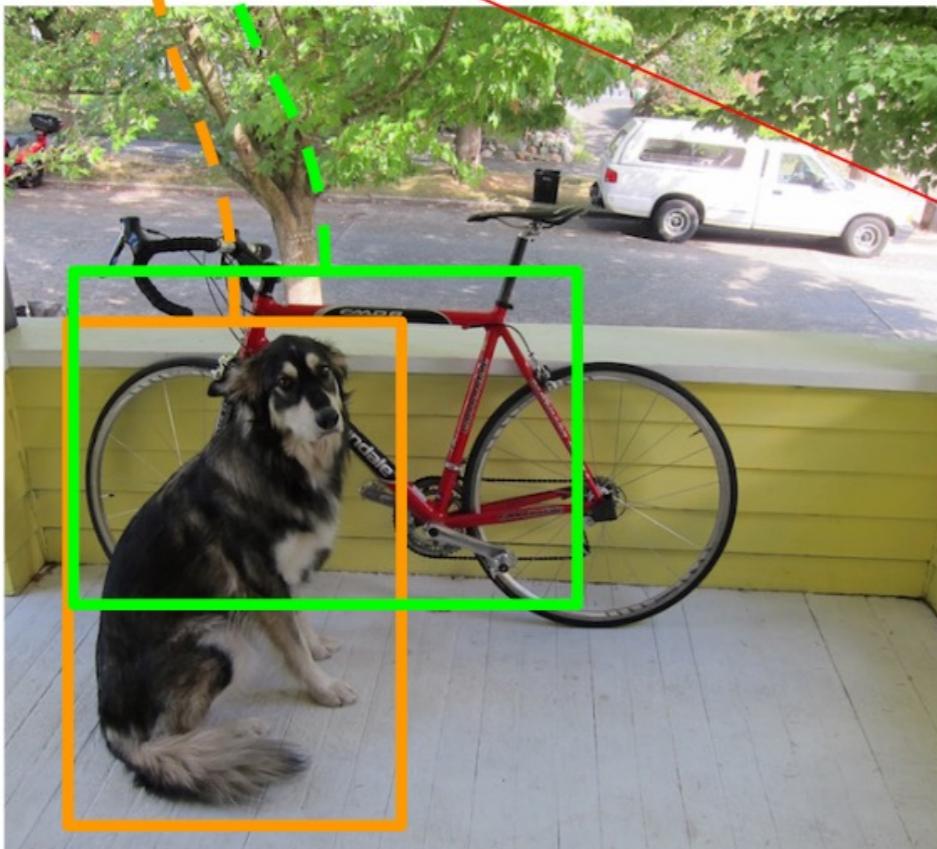
# YOLO v1 - NMS



# YOLO v1 - NMS

class (dog) scores for each bbox

	bb47	bb20	bb15	bb7												bb1	bb4	bb8	bb98
class: dog	0.5	0	0.2	0.1												0	0	0	0



If  $\text{IoU}(\text{bbox\_max}, \text{bbox\_cur}) > 0.5$  then set 0 score to  $\text{bbox\_cur}$ .

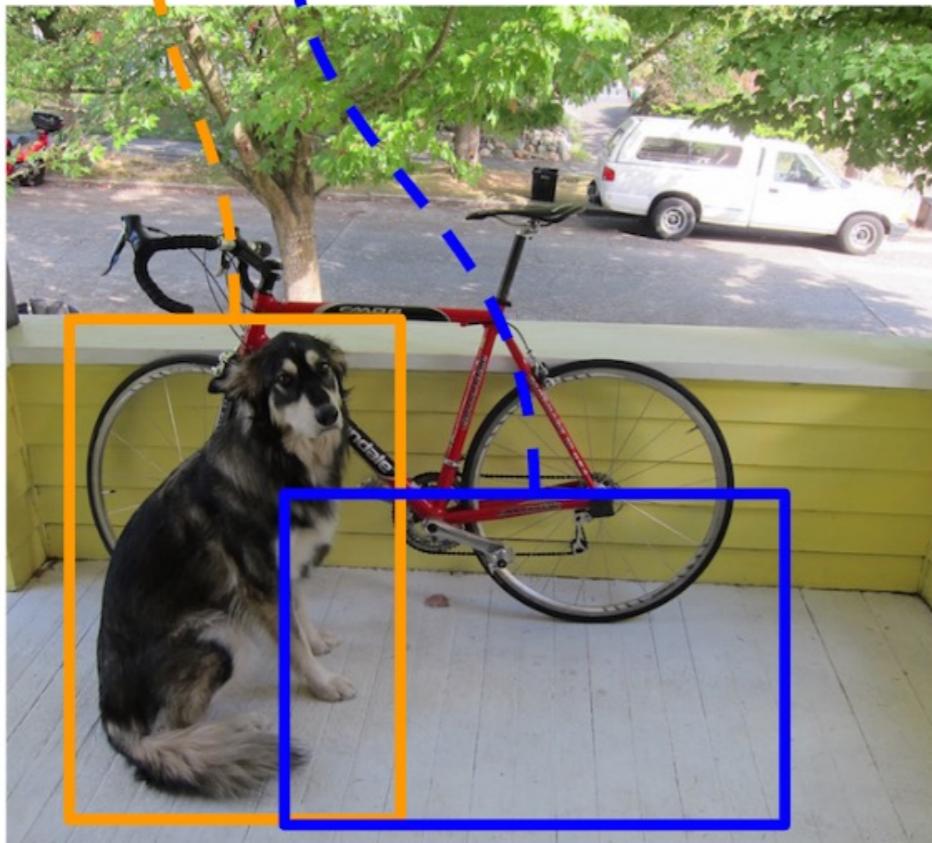
In this case: set to 0.

bb47과 bb20의 겹치는 영역이 0.5를 넘을 경우 해당 grid의 score값을 0으로 설정

# YOLO v1 – NMS

class (dog) scores for each bbox

	bb47	bb20	bb15	bb7		bb1	bb4	bb8	bb98
class: dog	0.5	0	0.2	0.1					
	0	0	0	0					



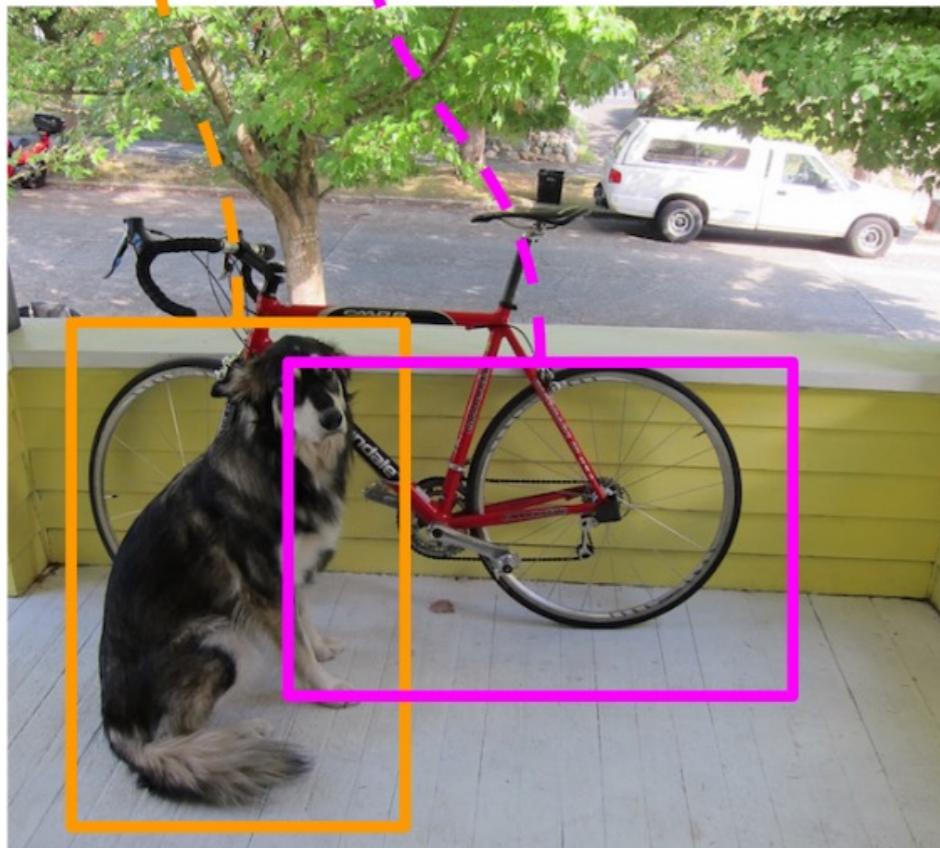
Go to next `bbox_cur`.

If  $\text{IoU}(\text{bbox\_max}, \text{bbox\_cur}) > 0.5$  then set 0 score to  $\text{bbox\_cur}$ .

bb47과 bb15의 겹치는 영역이 0.5를 넘지 않아서 해당 grid의 score값을 그대로 둔다.

# YOLO v1 - NMS

class (dog) scores for each bbox



Go to next `bbox_cur`.

If  $\text{IoU}(\text{bbox\_max}, \text{bbox\_cur}) > 0.5$  then set 0 score to  $\text{bbox\_cur}$ .

In this case: continue.

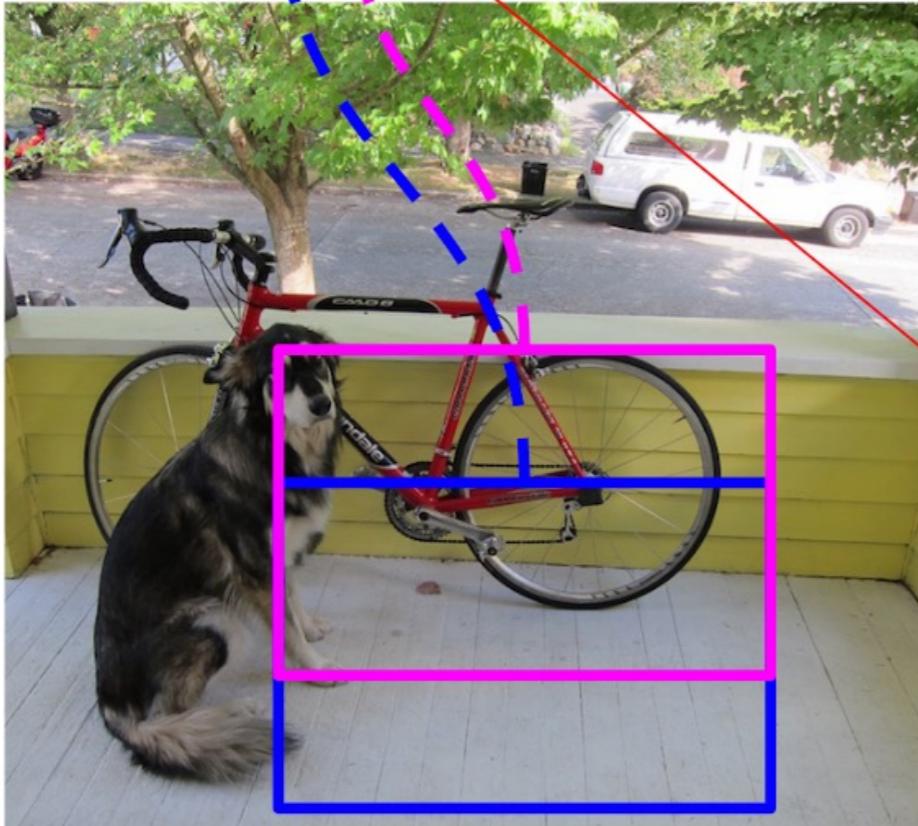
bb47과 bb7의 겹치는 영역이 0.5를 넘지 않아서 해당 grid의 score값을 그대로 둔다.

나머지 bbox cur에 대해서도 동일하게 진행한다.

# YOLO v1 - NMS

class (dog) scores for each bbox

	bb47	bb20	bb15	bb7		bb1	bb4	bb8	bb98
class: dog	0.5	0	0.2	0					
	0	0	0	0					1x98



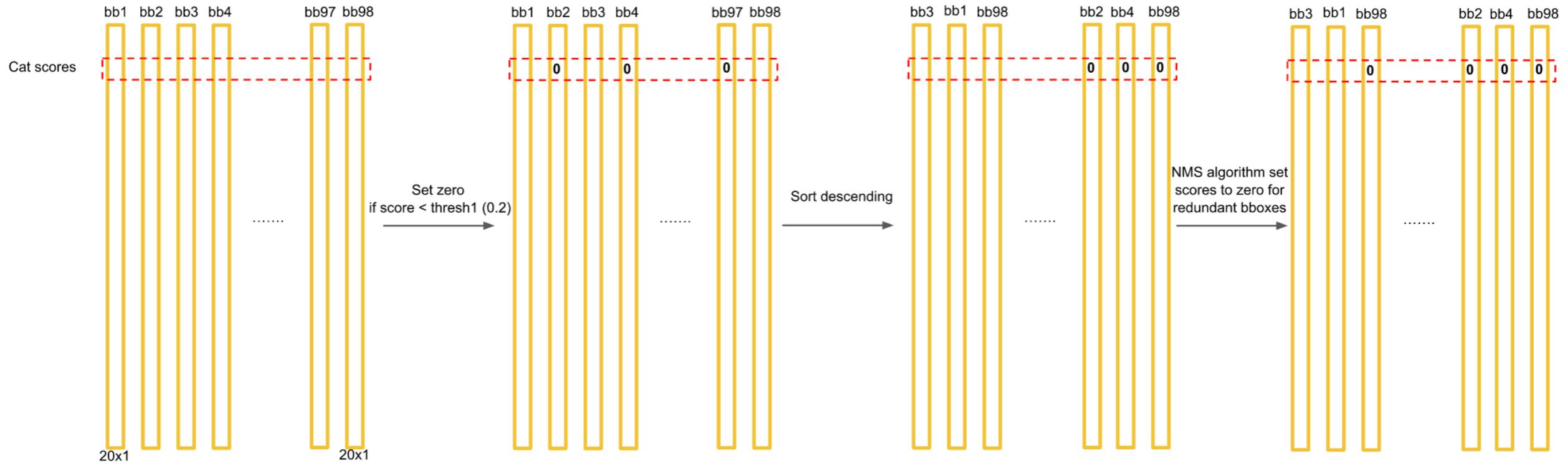
Go to next bbox cur.

If  $\text{IoU}(\text{bbox\_max}, \text{bbox\_cur}) > 0.5$  then set 0 score to `bbox_cur`.

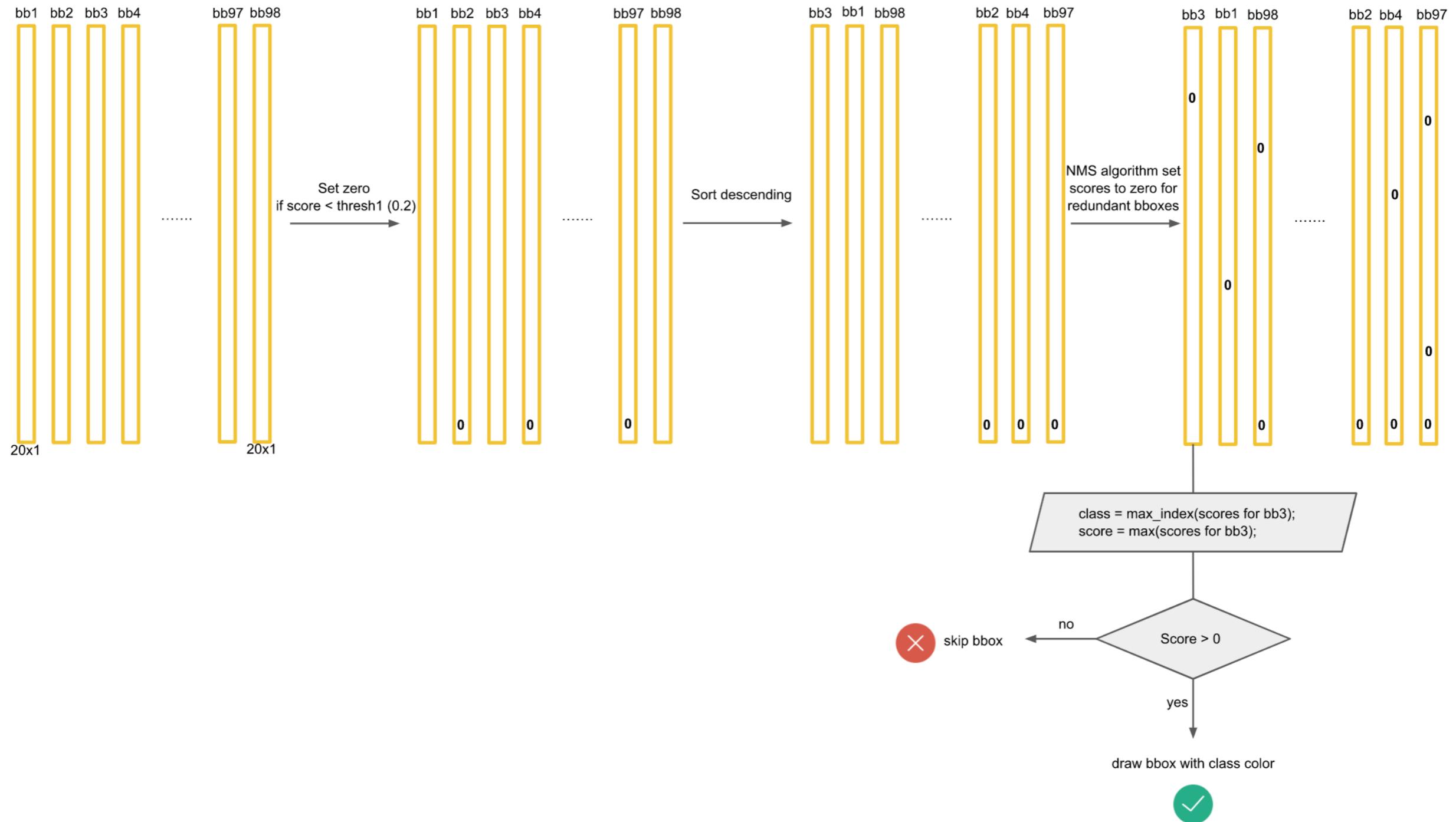
In this case: set to 0.

Do this procedure for other “bbox\_max” and for other corresponding “bbox cur”.

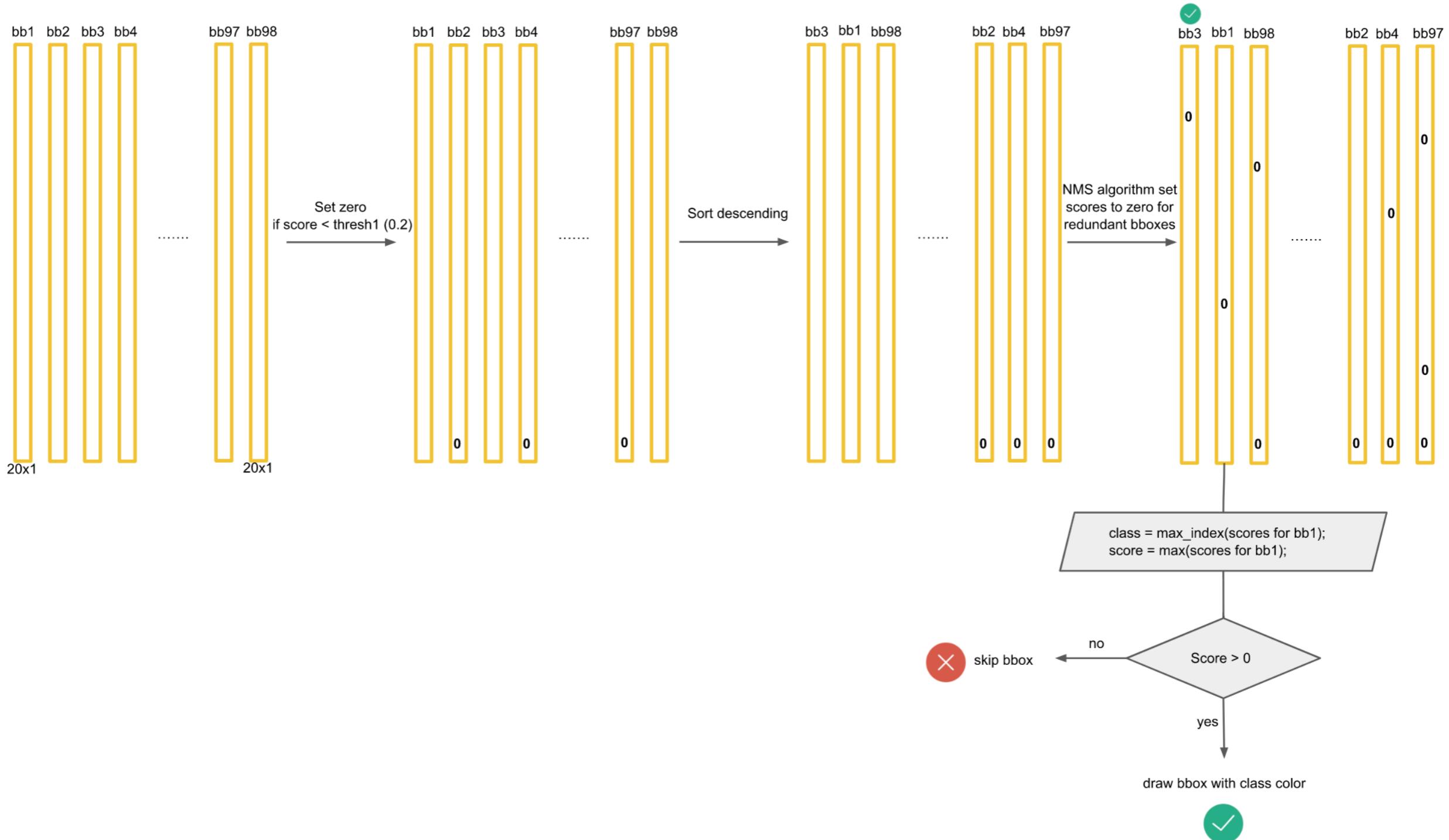
# YOLO v1 – NMS



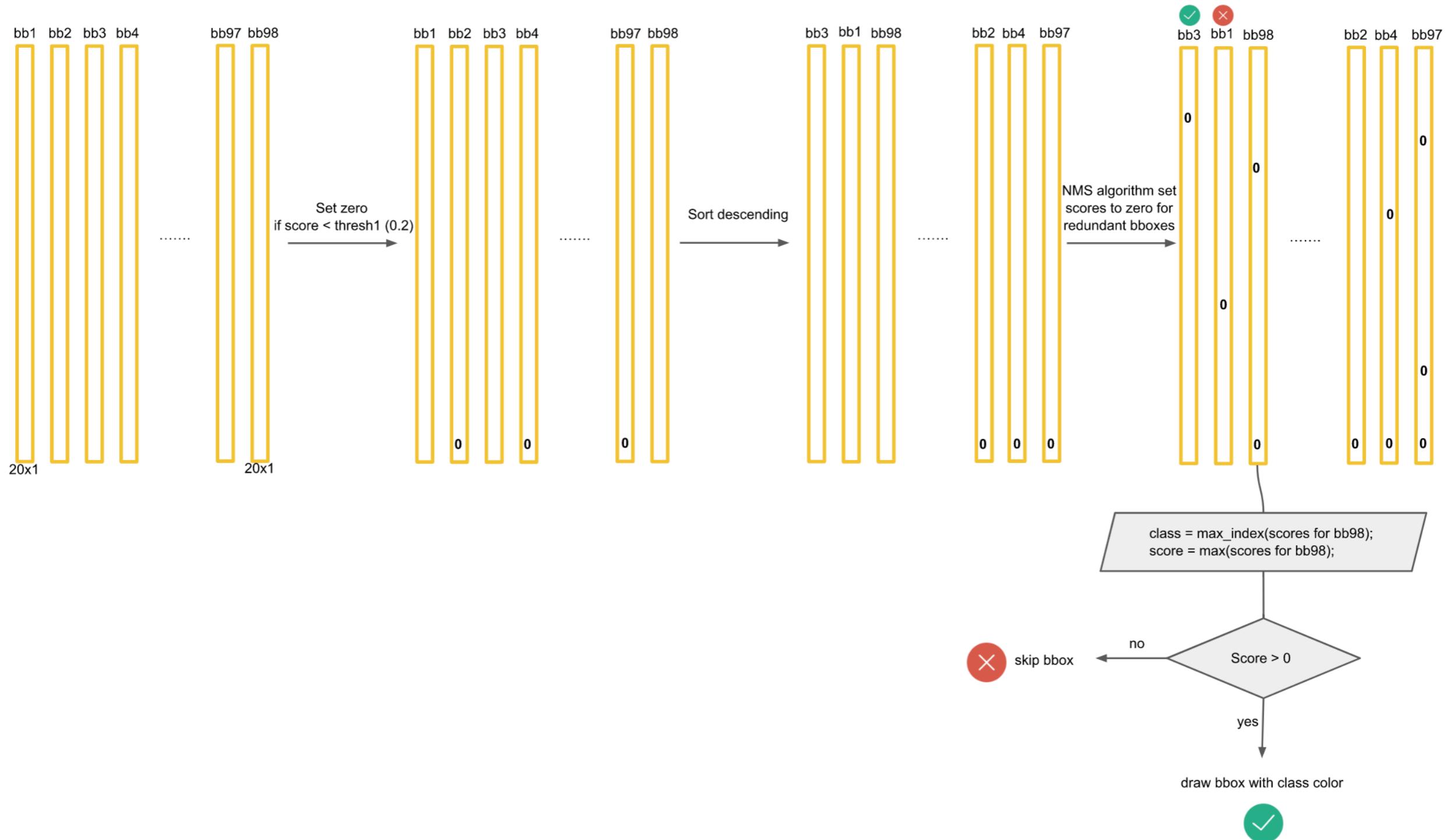
# YOLO v1 – NMS



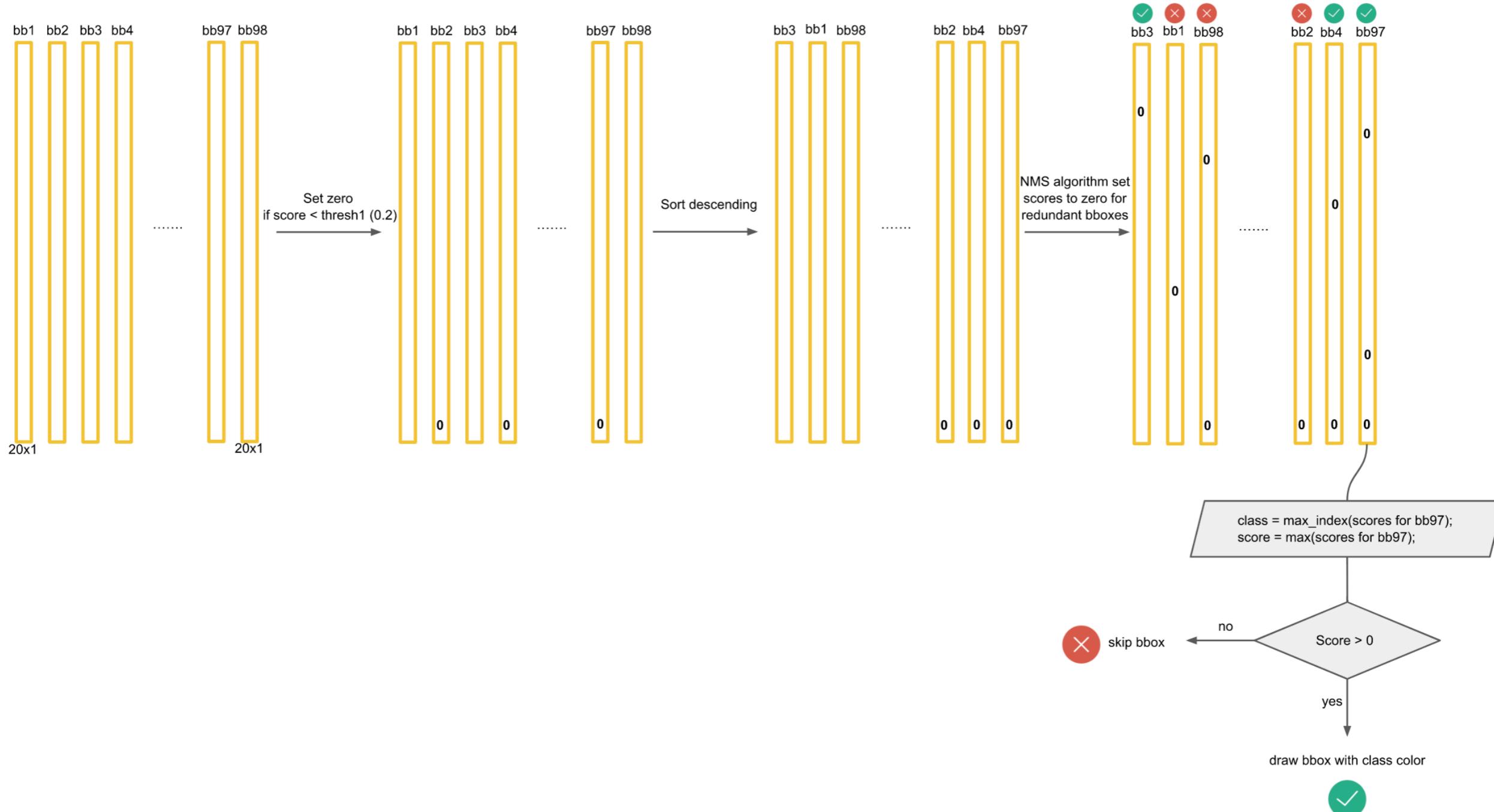
# YOLO v1 – NMS



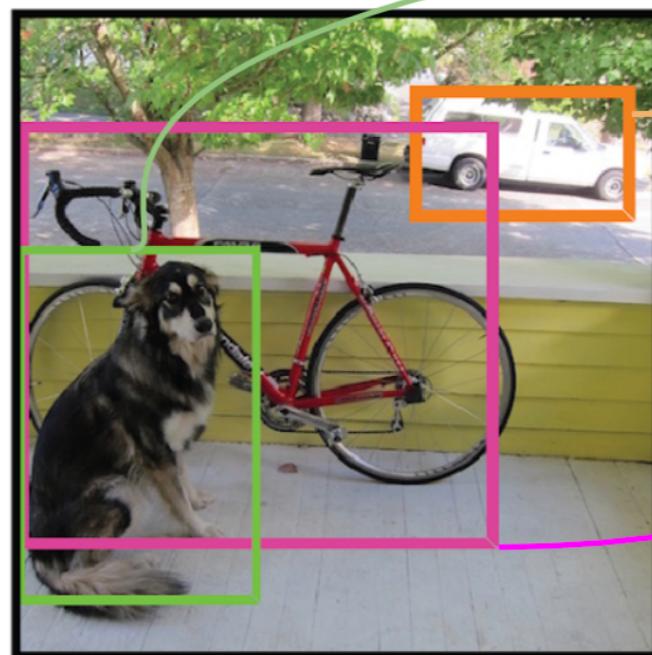
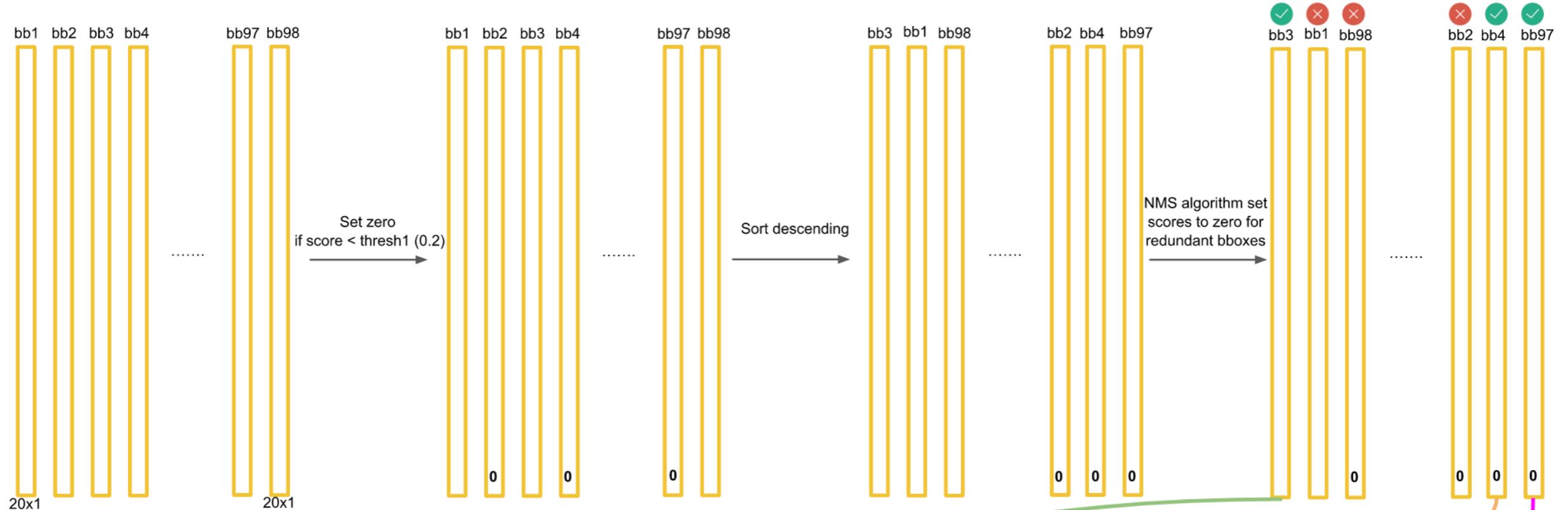
# YOLO v1 – NMS



# YOLO v1 – NMS



# YOLO v1 – NMS



# Loss Function (sum-square error)

$$\text{MSE} = \frac{\text{SSE}}{n-2} = \frac{1}{n-2} \sum (\hat{y} - y)^2$$

\* MSE와 SSE의 관계  
\* YOLO는 SSE를 사용한다.

1. Grid cell의 여러 bounding box들 중, ground-truth box와의 IOU가 가장 높은 bounding box를 predictor로 설정한다.
2. 1의 기준에 따라 아래 기호들이 사용된다.

기호	설명
$\mathbb{1}_{ij}^{\text{obj}}$	Object가 존재하는 grid cell i의 predictor bounding box j
$\mathbb{1}_{ij}^{\text{noobj}}$	Object가 존재하지 않는 grid cell i의 bounding box j
$\mathbb{1}_i^{\text{obj}}$	Object가 존재하는 grid cell i

Ground-truth box의 중심점이 어떤 grid cell 내부에 위치하면, 그 grid cell에는 Object가 존재한다고 판단

# Loss Function (sum-square error)

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

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

$$+ \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad (3)$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad (4)$$

$$+ \sum_{i=0}^S \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (5)$$

- Object의 중심이 위치한 Grid Cell이 물체를 예측할 책임을 갖음.
- Grid Cell에 2개 이상의 Bbox가 존재하면, 이중 Ground-truth box와 IoU(Intersection over Union)가 가장 높은 Bbox가 책임을 갖음( $\mathbb{1}_{ij}^{\text{obj}}$ ).

(1) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, x와 y의 loss를 계산

(2) Object가 존재하는 grid cell i의 predictor bounding box j에 대해, w와 h의 loss를 계산.

큰 box에 대해서는 small deviation을 반영하기 위해 제곱근을 취한 후, sum-squared error를 한다.

(같은 error라도 larger box의 경우 상대적으로 IOU에 영향을 적게 준다.)

# Loss Function (sum-square error)

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

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

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad (3)$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad (4)$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (5)$$

(3) Object가 존재하는 grid cell i의 predictor bounding box j에 대해,  
confidence score의 loss를 계산. ( $C_i = 1$ )

(4) Object가 존재하지 않는 grid cell i의 bounding box j에 대해,  
confidence score의 loss를 계산 ( $C_i = 0$ )

(5) Object가 존재하는 grid cell i에 대해, conditional class probability의 loss 계산.  
(Correct class c:  $p_i(c) = 1$ , otherwise:  $p_i(c) = 0$ )

# Loss Function (sum-square error)

loss function:

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

model. We use sum-squared error because it is easy to optimize, however it does not perfectly align with our goal of maximizing average precision. It weights localization error equally with classification error which may not be ideal. Also, in every image many grid cells do not contain any object. This pushes the “confidence” scores of those cells towards zero, often overpowering the gradient from cells that do contain objects. This can lead to model instability, causing training to diverge early on.

To remedy this, we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects. We use two parameters,  $\lambda_{\text{coord}}$  and  $\lambda_{\text{noobj}}$  to accomplish this. We set  $\lambda_{\text{coord}} = 5$  and  $\lambda_{\text{noobj}} = .5$ .

$$\lambda_{\text{coord}} = 5, \quad \lambda_{\text{noobj}} = 0.5$$

물체가 포함된 것과 그렇지 않는 것의 가중치 차이를 10배로 설정한다.

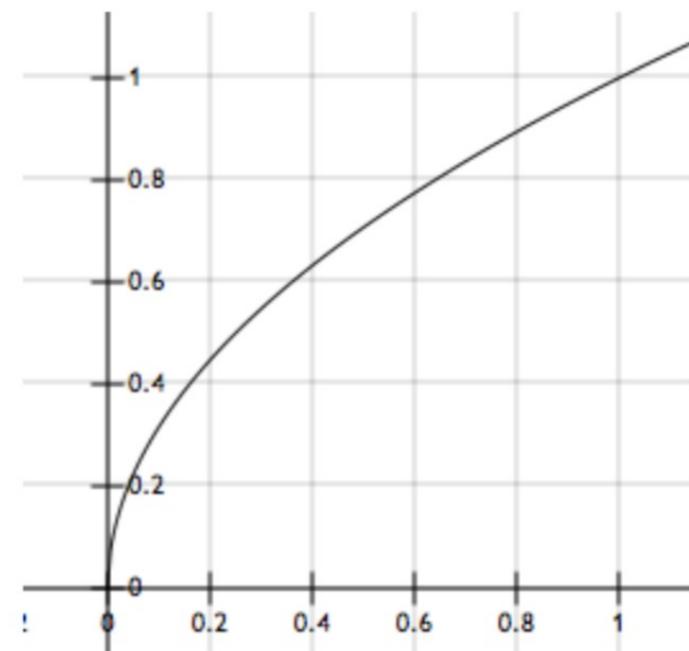
# YOLO v1 – Loss function

loss function:

큰 Bbox에 대하여 상대적으로  
작은 Loss를 주기위해 제곱근을  
취함.

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

Sum-squared error also equally weights errors in large boxes and small boxes. Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.



# YOLO v1 – Hyper-Parameter

- ImageNet 1000-class dataset으로 20개의 convolutional layer를 pre-training
- Pre-training 이후 4 convolutional layers와 2 fully connected layers를 추가
- Bounding Box의 width와 height는 이미지의 width와 height로 normalize (Range: 0~1)
- Bounding Box의 x와 y는 특정 grid cell 위치의 offset값을 사용한다 (Range: 0~1)

$\lambda_{coord} : 5, \lambda_{noobj} : 0.5$

Batch size: 64

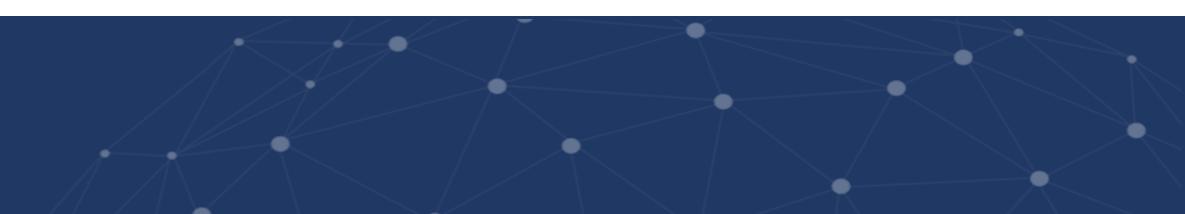
- Momentum: 0.9 and a decay of 0.0005
- Learning Rate: 0.001에서 0.01로 epoch마다 천천히 상승시킴.
- 이후 75 epoch동안 0.01, 30 epoch동안 0.001, 마지막 30 epoch동안 0.0001
- Dropout Rate: 0.5
- Data augmentation: random scaling and translations of up to 20% of the original image size
- Activation function: leaky rectified linear activation

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

# YOLO v1 – 결과

- 각각의 grid cell이 하나의 클래스만을 예측할 수 있으므로, 작은 object 여러 개가 밀접하여 붙어 있으면 제대로 예측하지 못한다.
- bounding box의 형태가 training data를 통해서만 학습되므로, 새로운/독특한 형태의 bouding box의 경우 정확히 예측하지 못한다.
- 몇 단계의 layer를 거쳐서 나온 feature map을 대상으로 bouding box를 예측하므로 localization이 다소 부정확해지는 경우가 있다.

# YOLO v1 – 결과



Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45
<hr/>			
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

다른 real-time object detect system들에 비해  
높은 mAP를 보여준다. Fast YOLO의 경우 가장  
빠른 속도를 보여준다.

\*YOLO v1이 발표될 때에는 딱히 비교할 만한 대상이  
되는 Real-Time Detector가 없었다고 봐도 무방하다.

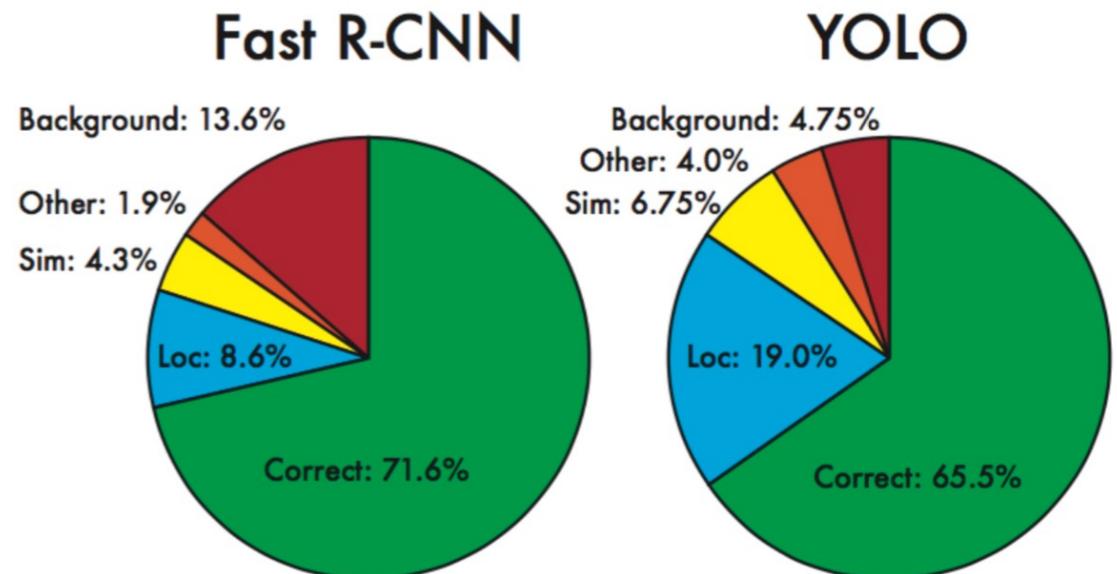
**Table 1: Real-Time Systems on PASCAL VOC 2007.** Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

# YOLO v1 – 결과

- Correct: correct class and IOU > .5
- Localization: correct class, .1 < IOU < .5
- Similar: class is similar, IOU > .1
- Other: class is wrong, IOU > .1
- Background: IOU < .1 for any object

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	<b>66.9</b>	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	<b>75.0</b>	<b>3.2</b>

**Table 2: Model combination experiments on VOC 2007.** We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.



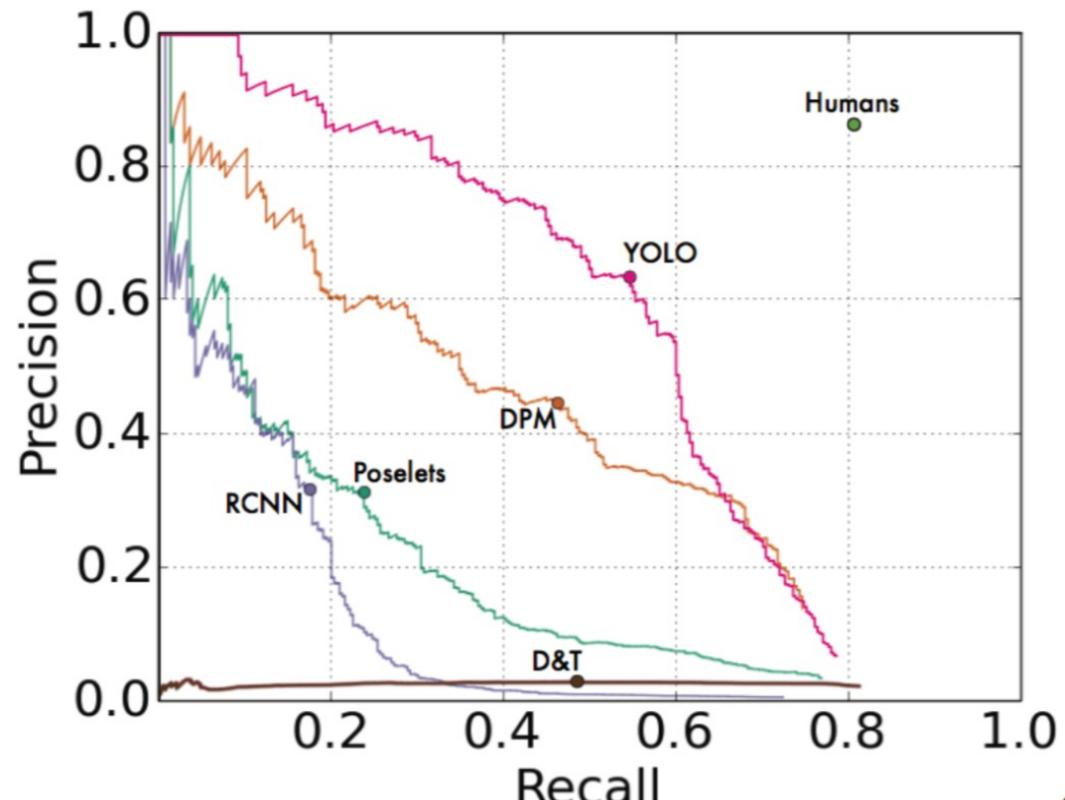
**Figure 4: Error Analysis: Fast R-CNN vs. YOLO** These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

# YOLO v1 – 결과

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	<b>73.9</b>	<b>85.5</b>	<b>82.9</b>	<b>76.6</b>	<b>57.8</b>	<b>62.7</b>	<b>79.4</b>	77.2	86.6	<b>55.0</b>	<b>79.1</b>	<b>62.2</b>	87.0	<b>83.4</b>	<b>84.7</b>	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	<b>79.8</b>	87.7	49.6	74.9	52.1	86.0	81.7	83.3	<b>81.8</b>	<b>48.6</b>	<b>73.5</b>	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
<b>Fast R-CNN + YOLO</b>	<b>70.7</b>	<b>83.4</b>	<b>78.5</b>	<b>73.5</b>	<b>55.8</b>	<b>43.4</b>	<b>79.1</b>	73.1	<b>89.4</b>	<b>49.4</b>	<b>75.5</b>	<b>57.0</b>	<b>87.5</b>	80.9	<b>81.0</b>	<b>74.7</b>	<b>41.8</b>	<b>71.5</b>	<b>68.5</b>	<b>82.1</b>	<b>67.2</b>
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	<b>68.8</b>	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	<b>87.5</b>	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	<b>64.0</b>	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	<b>64.7</b>	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
<b>YOLO</b>	<b>57.9</b>	<b>77.0</b>	<b>67.2</b>	<b>57.7</b>	<b>38.3</b>	<b>22.7</b>	<b>68.3</b>	<b>55.9</b>	<b>81.4</b>	<b>36.2</b>	<b>60.8</b>	<b>48.5</b>	<b>77.2</b>	<b>72.3</b>	<b>71.3</b>	<b>63.5</b>	<b>28.9</b>	<b>52.2</b>	<b>54.8</b>	<b>73.9</b>	<b>50.8</b>
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

**Table 3: PASCAL VOC 2012 Leaderboard.** YOLO compared with the full comp4 (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

# YOLO v1 – 결과



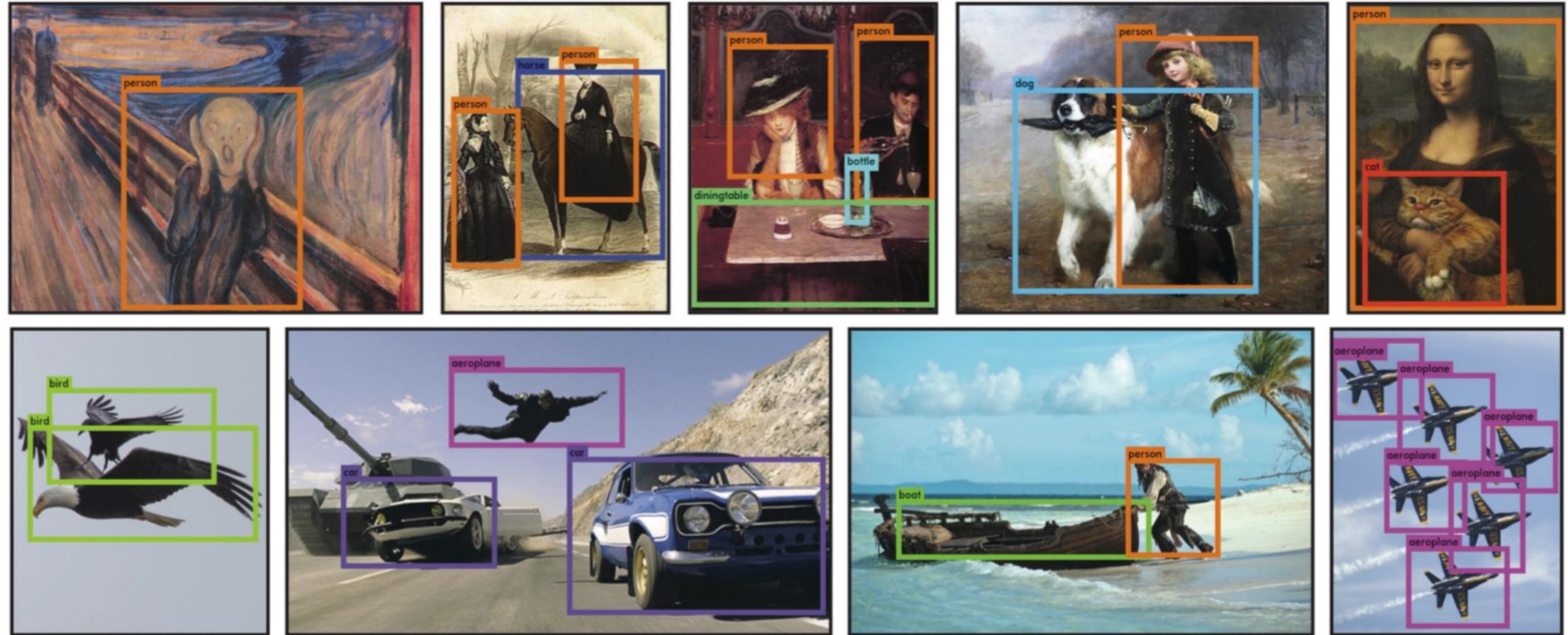
(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best $F_1$	AP
<b>YOLO</b>	<b>59.2</b>	<b>53.3</b>	<b>0.590</b>	<b>45</b>
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets.  
The Picasso Dataset evaluates on both AP and best  $F_1$  score.

Figure 5: Generalization results on Picasso and People-Art datasets.

# YOLO v1 – 결과



**Figure 6: Qualitative Results.** YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

# 인공지능 심화 과정

YOLO v1

**YOLO v2**

YOLO v3

YOLO v4



# YOLO V2 (YOLO9000 : Better, Faster, Stronger)

YOLO V2는 크게 3 부분으로 나뉜다.

- **Better** : Accuracy , mAP 측면의 개선사항
- **Faster** : 속도 개선
- **Stronger** : 더 많은, 다양한 클래스 예측

# YOLO V2 (YOLO9000 : Better, Faster, Stronger)

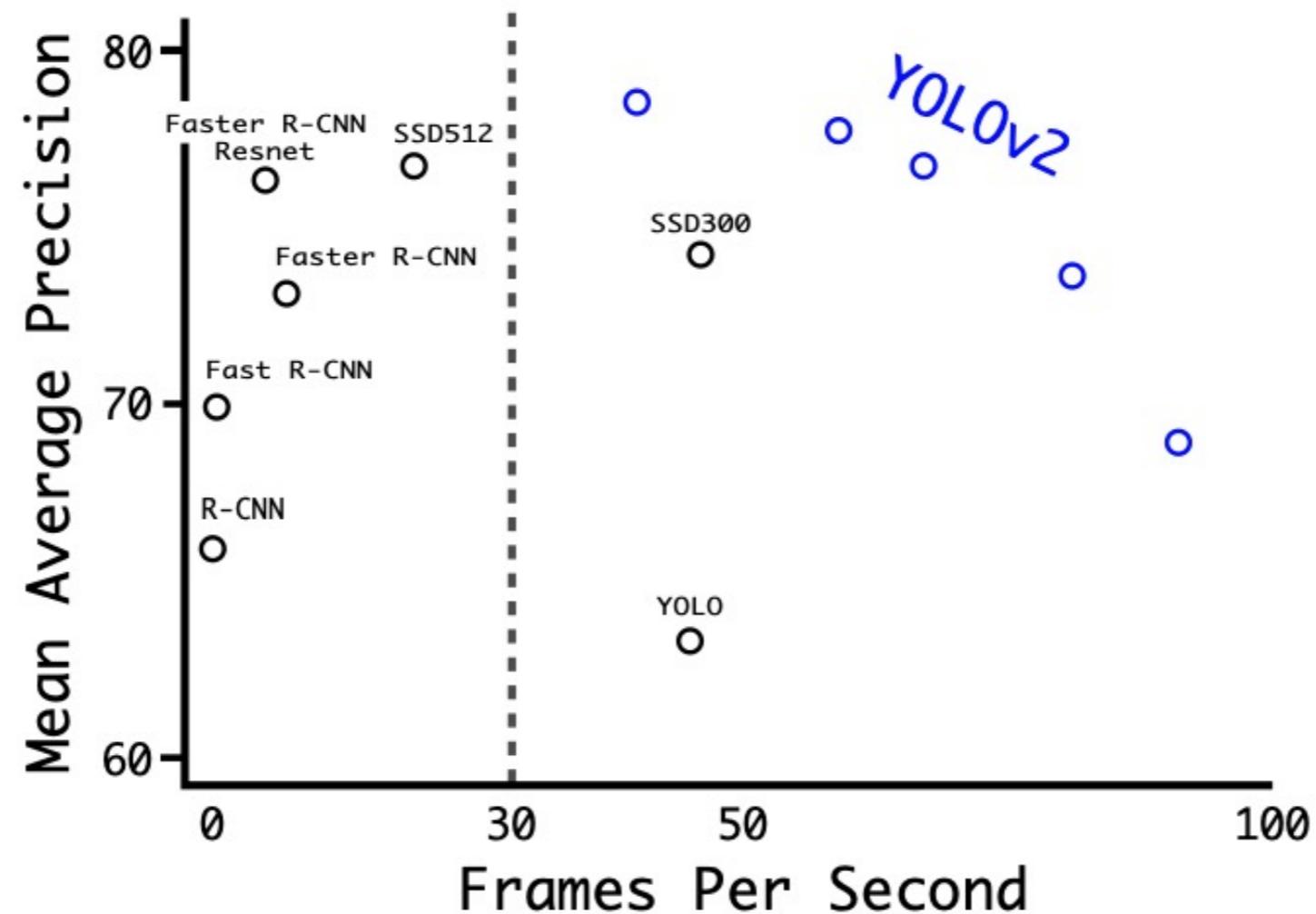


Figure 4: Accuracy and speed on VOC 2007.

# 1. Better

	YOLO							YOLOv2
batch norm?	✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?		✓	✓	✓	✓	✓	✓	✓
convolutional?			✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8
								<b>78.6</b>

**Table 2: The path from YOLO to YOLOv2.** Most of the listed design decisions lead to significant increases in mAP. Two exceptions are switching to a fully convolutional network with anchor boxes and using the new network. Switching to the anchor box style approach increased recall without changing mAP while using the new network cut computation by 33%.

# YOLO V2 (YOLO9000 : Better, Faster, Stronger)

- Better
  - Batch normalization
  - High resolution classifier
  - Convolution with anchor boxes
  - Dimension clusters
  - Direct location prediction
  - Fine-grained features
  - Multi-scale training

- Faster
  - Darknet-19
  - Training for classification
  - Training for detection

- Stronger
  - Hierarchical classification
  - Dataset combination with Word-tree
  - Joint classification and detection

YOLOv2

YOLO9000

# 1. Better



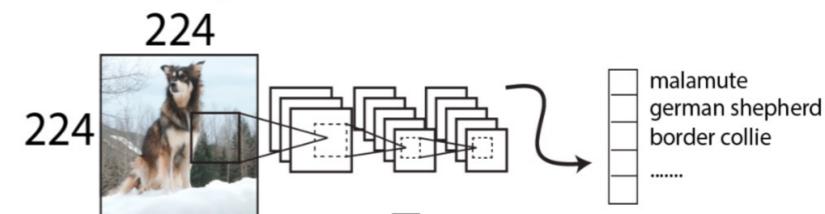
## Batch Normalization

모든 컨볼루션 네트워크에 배치 정규화 추가로 수렴을 크게 향상시킨다. 이에 mAP가 2% 가량 향상되며, 배치 정규화를 사용하면 과적합없이 모델에서 Dropout을 제거할 수 있다.

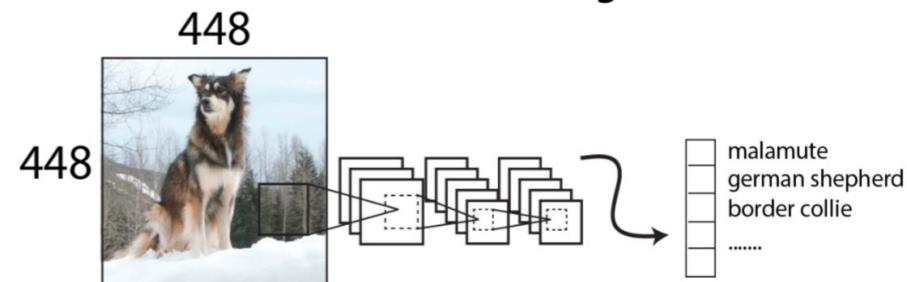
## High Resolution

224x224크기의 이미지에서 사전학습하고, Detection을 위해 448x448 크기로 증가시킨다.  
10 epochs까지 Fine Tune한다. 이렇게 하면 Classification Network의 mAP가 4%가량 증가한다.

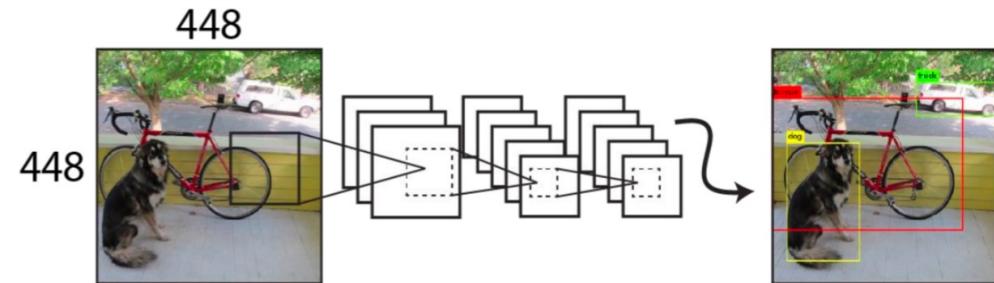
### Train on ImageNet



### Resize and Fine-Tune on ImageNet



### Fine-Tune on Detection



# 1. Better

## Convolutional with Anchor boxes

YOLO v1은 각 grid cell의 bounding box의 좌표가 0~1 사이의 값을 가지도록 랜덤으로 설정한 뒤 학습을 통해 최적의 값을 찾아가는 과정을 거친다. 반면 Faster R-CNN 모델은 사전에 9개의 anchor box를 정의한 후 bounding box regression을 통해 x, y 좌표와 aspect ratio(offset)을 조정하는 과정을 거친다. 좌표 대신 offset을 예측하면 문제가 보다 단순하고 네트워크가 학습하기 쉽다는 장점이 있다.

YOLO v2에서는 **anchor box**를 도입하며, 이 과정에서 네트워크를 수정한다. 먼저 conv layer의 output이 보다 높은 resolution을 가지도록 pooling layer를 제거했다. 또한 입력 이미지를 448x448에서 네트워크를 줄여 416x416 크기의 입력 이미지를 사용한다. 입력 이미지를 수정한 이유는 최종 output feature map의 크기가 홀수가 되도록 하여, feature map 내에 하나의 중심 cell(single center cell)이 존재할 수 있도록 하기 위함이다. 보통 객체의 크기가 큰 경우 이미지 내에서 중심을 차지하기 때문에, 하나의 중심 cell이 있으면 이를 잘 포착할 수 있기 때문이다.

# 1. Better

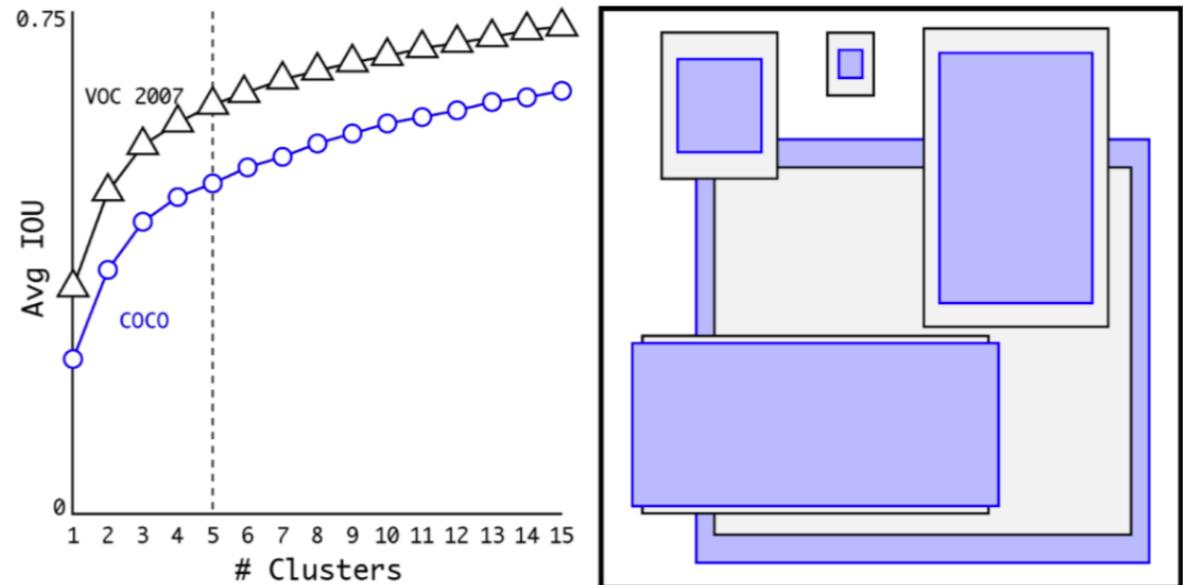
416x416 크기의 입력 이미지를 네트워크에 입력할 경우 최종적으로 13x13 크기의 feature map을 얻을 수 있다.(down sample ratio = 1/32). YOLO v1은 각 cell별로 2 개의 bounding box를 예측하여 총 98(=7x7x2)개의 bounding box를 예측하지만, Y OLO v2는 anchor box를 사용하여 보다 많은 수의 bounding box를 예측합니다. an chor box를 사용하지 않은 경우 mAP 값이 69.5%, recall값은 81%인 반면, anchor b ox를 사용한 경우 mAP 값은 69.2%, recall 값은 88%로 나왔습니다. anchor box를 사용하면 mAP 값이 감소하지만 recall 값이 상승하고, 이는 모델이 더 향상될 여지 가 있음을 나타낸다고 합니다.

**Object detection task**에서 recall 값이 높다는 것은 모델이 실제 객체의 위치를 예 측한 비율이 높음을 의미한다. YOLO v1이 recall 값이 낮은 이유는 region propos al 기반의 모델에 비해 이미지 당 상대적으로 적은 수의 bounding box를 예측하기 때문이다. 하지만 YOLO v2에서 anchor box를 통해 더 많은 수의 bounding box 를 예측하면서 실제 객체의 위치를 보다 잘 포착하게 되고, 이를 통해 recall 값이 상승하게 됩니다.

# 1. Better

## Dimension Clustering

- Faster RCNN의 경우  $\frac{1}{2}$ , 1, 2의 비율로 128, 256, 512크기 총 9개의 Box를 사용한 반면, k-means clustering을 이용하여, 더 효율적으로 Anchor Boxes 생성.  
(5개의 box로 9개의 box보다 좋은 prior box 생성)



Clustering box dimensions on VOC and COCO

Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes [15]	9	60.9
Cluster IOU	9	67.2

Average IOU of boxes to closest priors on VOC 2007

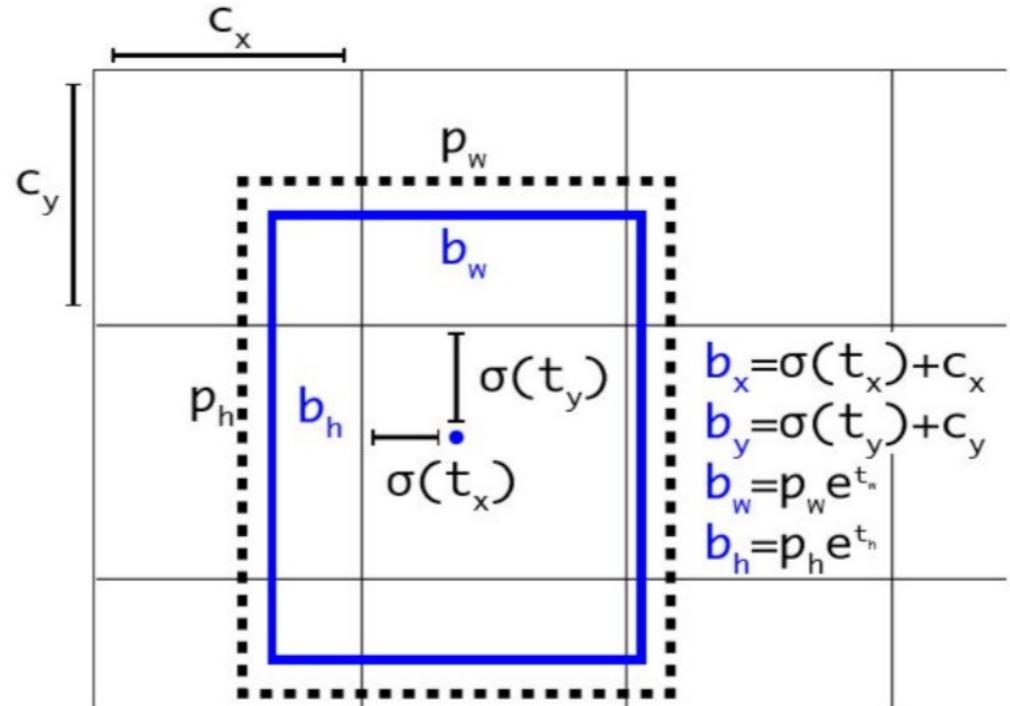
# 1. Better

## Direct location prediction

$$x = (t_x * w_a) - x_a$$

$$y = (t_y * h_a) - y_a$$

anchor box는 bounding box regressor 계수(coefficient)를 통해 위의 공식과 같이 bounding box의 위치를 조정합니다. 하지만  $t_x, t_y, t_x, t_y$ 와 같은 계수는 제한된 범위가 없기 때문에 anchor box는 이미지 내의 임의의 지점에 위치할 수 있다는 문제가 있습니다. 이로 인해 최적화된 값을 찾기 까지 오랜 시간이 걸려 모델은 초기에 불안정하게 됩니다.



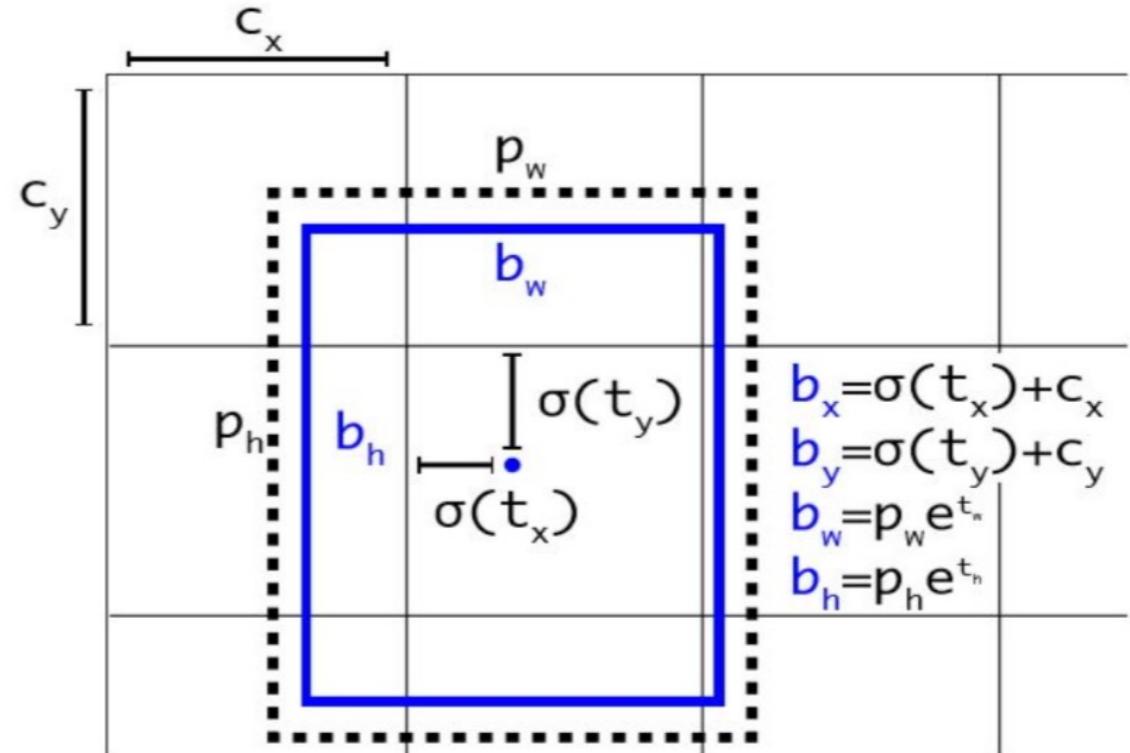
**Figure 3: Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

# 1. Better

## Direct location prediction

논문의 저자는 이러한 문제를 해결하기 위해 YOLO의 방식을 사용하여 grid cell에 상대적인 위치 좌표를 예측하는 방법을 선택했다. 이는 예측하는 bounding box의 좌표는 0~1 사이의 값을 가짐을 의미한다. 위의 그림에서  $c_x, c_y$ ,  $c_x, c_y$ 는 grid cell의 좌상단 offset입니다. bounding box regression을 통해 얻은  $t_{xw}, t_{yw}$  값에 logistic regression 함수( $\sigma$ )를 적용하여 0~1 사이의 값을 가지도록 조정했다.

예측하는 위치의 범위가 정해짐으로써 네트워크는 안정적으로 학습을 진행하는 것이 가능해진다. Dimension clustering을 통해 최적의 prior를 선택하고, anchor box 중심부 좌표를 직접 예측함으로서 recall값이 5% 정도 향상된다고 한다.



**Figure 3: Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

# 1. Better

## Direct location prediction

The network predicts 5 bounding boxes at each cell in the output feature map. The network predicts 5 coordinates for each bounding box,  $t_x, t_y, t_w, t_h$ , and  $t_o$ . If the cell is offset from the top left corner of the image by  $(c_x, c_y)$  and the bounding box prior has width and height  $p_w, p_h$ , then the predictions correspond to:

sigmoid 함수를 사용해 0~1사이로  $t_x, t_y$ 값 제약

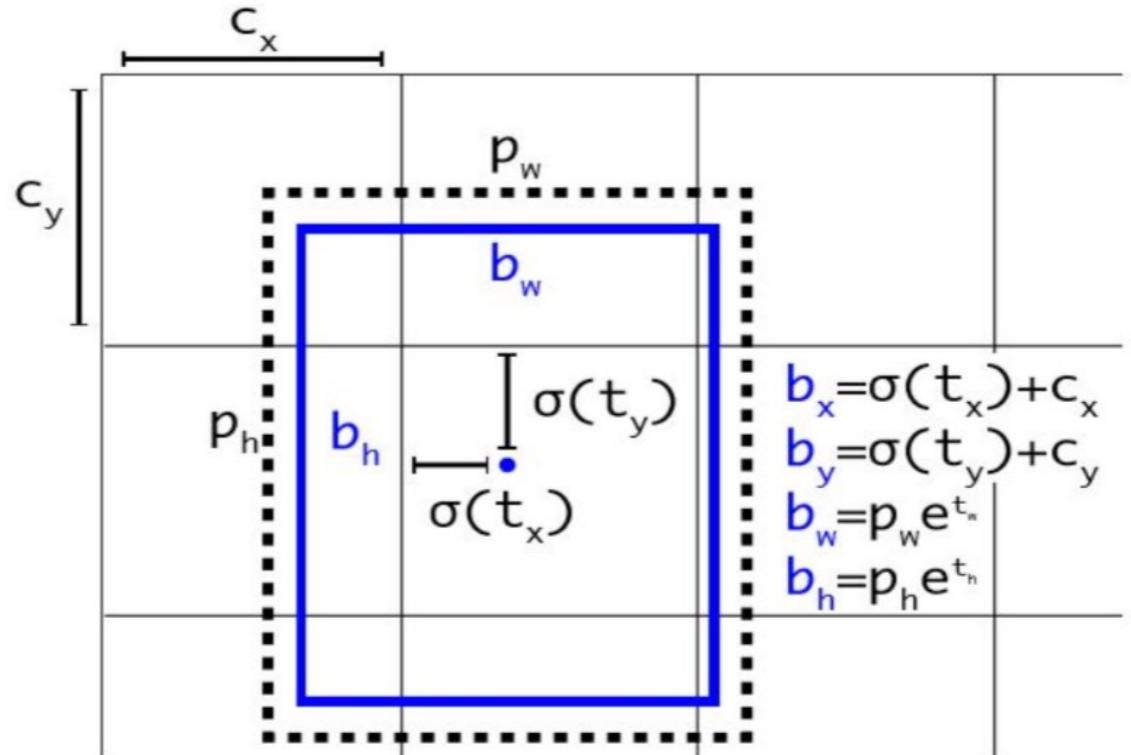
$$b_x = \sigma(t_x) + c_x$$

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

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

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

$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$



**Figure 3: Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

**cx, cy**는 그리드 셀의 좌상단 끝 **offset**

**pw, ph**는 **prior(우선순위 앵커박스)**의 **width, height**

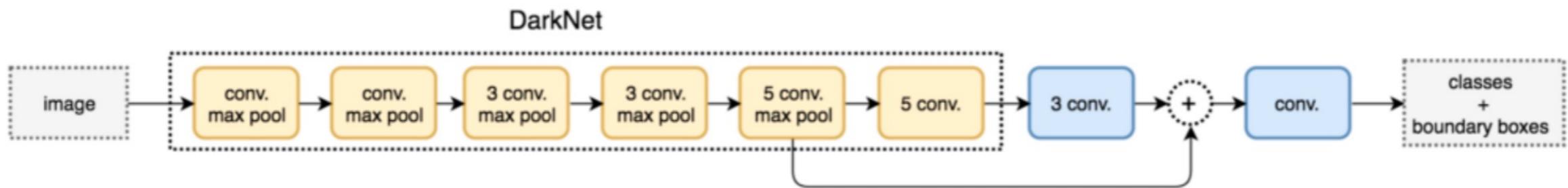
**tx, ty, tw, th**가 우리가 예측해야 할 값들

**bx, by, bw, bh**는 각 값을 조정하여 실제 GT와 IOU를 계산할 최종 **bounding box**의 **offset** 값들

# 1. Better

## Fine grained Feature

YOLOv2에서는 13x13의 Feature map에서 detectiona을 수행한다. 이는 큰 물체에는 충분하지만 작은 물체를 위치시키기 위한 Finer grained features의 이점을 얻을 수 있다. Faster R-CNN과 SSD는 proposal network으로부터 다양한 feature map을 얻는다. 여기서도 26x26 해상도에서 이전 계층의 기능을 가져오는 패스스루 계층을 추가하는 방식을 사용한다.

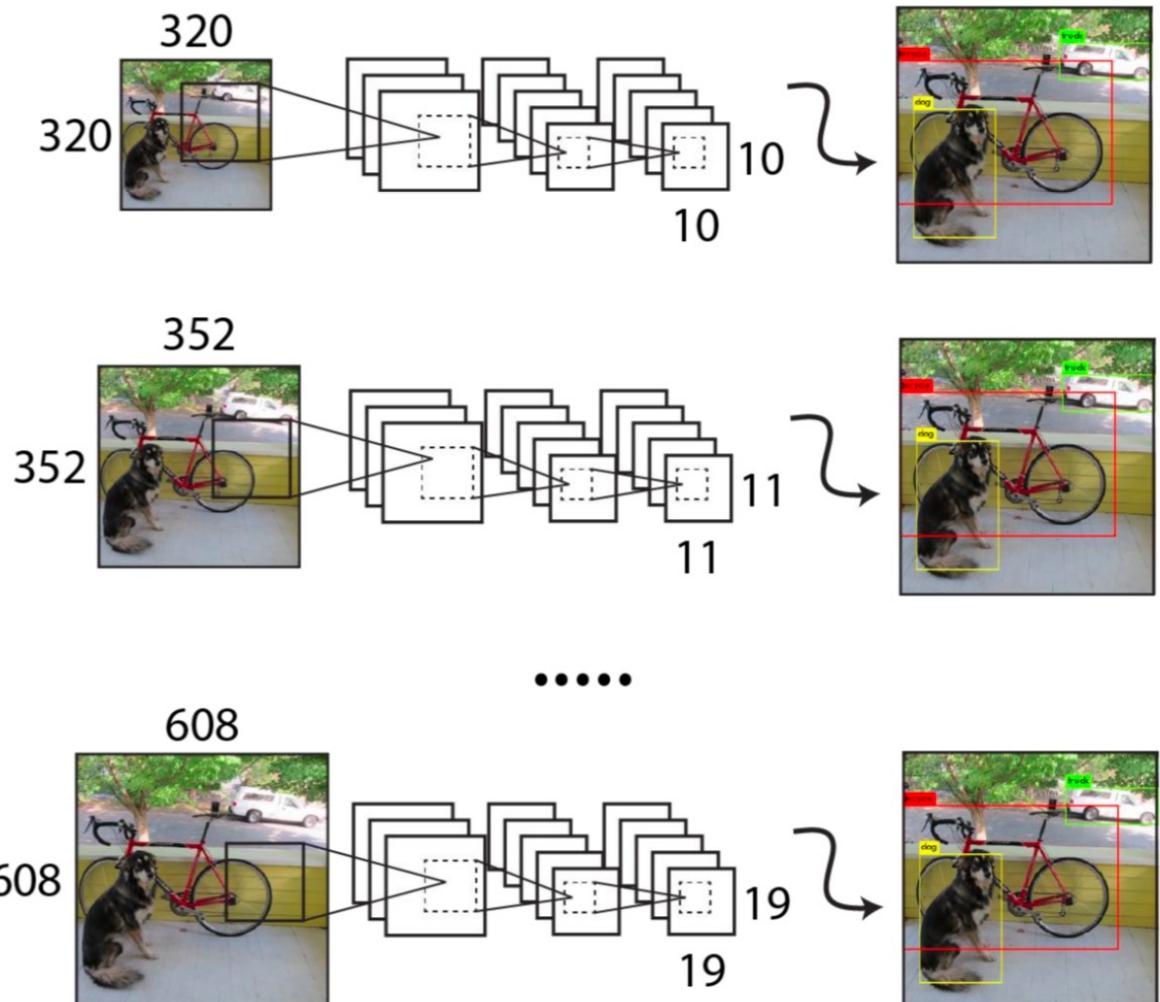


Passthrough network. 26x26x512 feature map과 13x13x2048 feature map을 concatenation

# 1. Better

## Multi-Scale Training

- 이미지를 다양한 사이즈로 resize하여, 다양한 크기의 Object를 예측할 수 있게 함.  
(Fully Convolutional Network이기 때문에 가능)
- 32의 배수의 이미지 크기로 320부터 608까지 이미지를 resize하여 10 배치씩 학습.



## 2. Faster

### Darknet-19

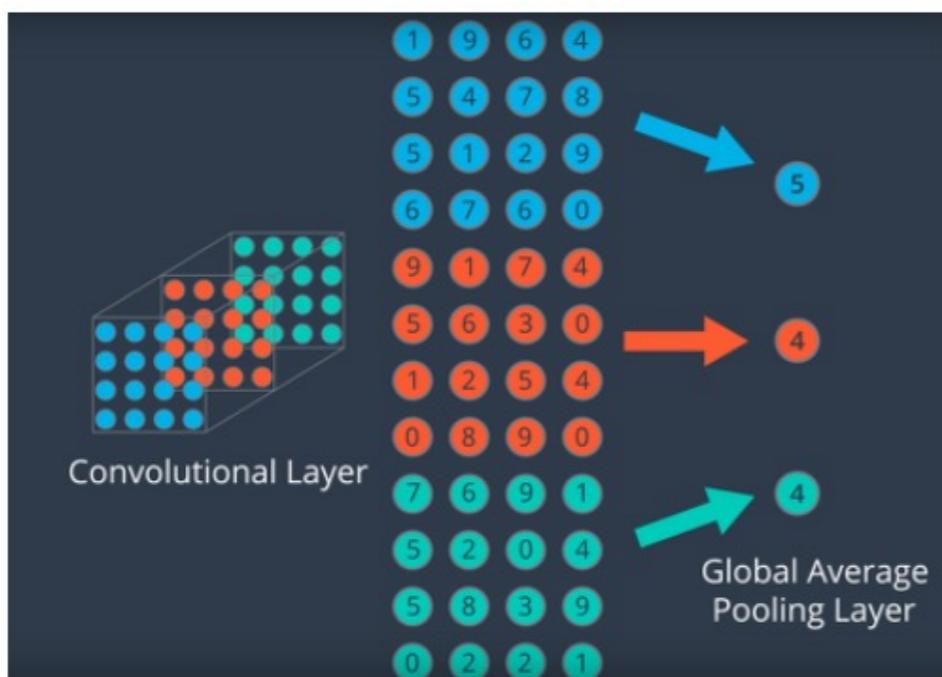
YOLOv2의 베이스로 사용될 새로운 분류 모델을 제안함.

VGGNet와 유사하게, 대부분 3x3필터를 사용하며, 풀링 단계마다 채널 수를 두배로 늘린다.

Network in Network 구조로 필터 수를 줄임

Global average pooling을 사용하여 파라메터 감소

시키고, Detection 속도를 향상시켰다.



Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Maxpool		$2 \times 2/2$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Convolutional	128	$1 \times 1$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Maxpool		$2 \times 2/2$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Maxpool		$2 \times 2/2$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	1000	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax			

Darknet-19 architecture

## 2. Faster

### 2-1. Darknet.

많은 Image Detection Model에서 classifier netowrk(앞단의 네트워크)로 VGG Net을 많이 쓴다. 하지만 VGG-16 은 30.69십억의 floating point 계산을 필요로 한다.(224x224 해상도 imgae의 경우)

YOLO v2에서는 GoogleNet을 기반으로한 독자적인 Darknet을 만들어 30십억의 계산량을 8십억으로

줄였다. (accuracy는 88%로 VGG-16의 90% 성능과 크게 차이나지 않는다.)

### 2-2. Training for classification.

ImageNet 1000개 클래스 구별 데이터셋을 160 epoch동안 학습하면서 learning rate = 0.1, polynomial rate decay = a power of 4(4로 나눈다는 뜻일듯), weight decay = 0.0005, momnetum = 0.9, 처음 튜닝은 224로 하다가 중간에 448로 fine tune.

### 2-3. Traininig for detection.

5 bounding box로 5개의 좌표(Confidence score + coordinate)와 20개의 class 점수를 예측하므로

한 그리드 셀에서는 총  $5 \times (5+20) = 125$ 개의 예측값을 가지게 된다. 또 중간에 passthrough layer로부터

(26x26) concatenate된 예측값도 포함. 160 epoch동안  $10^{-3}$ 에서 시작하여 10, 60, 90 에폭마다 decay하고, weight decay = 0.0005, momentum = 0.9를 사용했다. data augmentation 역시 random crops, color shifting 등을 이용했다.

## 2. Stronger

이 부분은 image classification은 클래스가 몇 천~몇 만개 정도로 많지만 detection의 라벨은 20~몇 백개 정도가 전부이다. 이런 갭을 완화하기 위한 전략을 소개하는 부분이라고 보면 된다.

- training 때 classification과 detection data를 섞어서 쓴다
- data set에서 detection data가 들어오면 원래 loss function을 활용해 계산
- data set에서 classification data가 들어오면 loss function에서 classification loss만 활용해 계산

여기서 드는 의문점은, detection에는 '개'라고 되어 있지만 classification에서는 '시츄', '비글', '푸들' 등과 같이 개 종류만 수백 종류가 있다. 따라서 라벨들을 일관성 있게 통합해야하고, 시츄를 개라고 했다고 해서 완전히 틀린 것은 아니므로 상호 배타적이지 않은 예시에 대해서 multi-label 모델을 사용한다.

## 2. Stronger

Hierarchical classification.

이미지넷(ImageNet)의 라벨들은 워드넷(WordNet)의 구조에 따라 정리되어 있다.

워드넷의 구조를 보면, '노퍽 테리어나, 요크셔 테리어는' 가축 - 개과 - 개 - 사냥개 - 테리어의 hyponym 하위어이다.([워드넷에 대한 설명 위키](#)) 여기서 YOLO 역시 이미지넷의 컨셉인 계층적 트리 구조를 이용해 label을 공통적으로 묶는 작업을 한다. 계층 트리를 만들기 위해서 워드넷 그래프로 어느 경로로 나타내는지 찾고, 많은 관련어들이

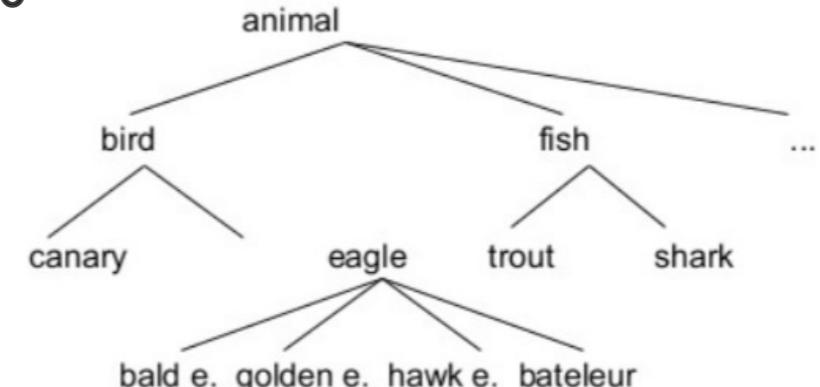
하나의 경로를 가지는 경우가 많으므로 그런 것들부터 먼저 처리하고 트리를 늘려갔다. (개 아래 요크셔 테리어, 시츄,

등등 이런 단어들을 먼저 개로 묶고 그런 카테고리들을 여러개 붙인다는 의미) 루트에서 특정경로가 제일 가까운 것부터

선택해 붙여 나간다. 워드 트리를 구축하기 위해 1000개의 클래스를 갖는 이미지넷 데이터를 이용해 Darknet-19 모델을 학습시켰다. 워드 트리1k를 만들기 위해서는 중간 노드들을 추가해야 했기 때문에 라벨의 갯수가 1000개에서

1369개로 들어났다. (예를 들어 이미지의 라벨이 "노르포크 테리어" 인 경우 워드넷에서 관련어(sysnets)인 "개", "포유동물" 등의 라벨 까지도 얻게 된다. 트리 중간 중간에 중복되는 라벨이 생기고 "포유동물" 드디어 표고 개념으로 갖는

라벨이 생기기 때문에 369개만큼 노드가 늘어난다.)



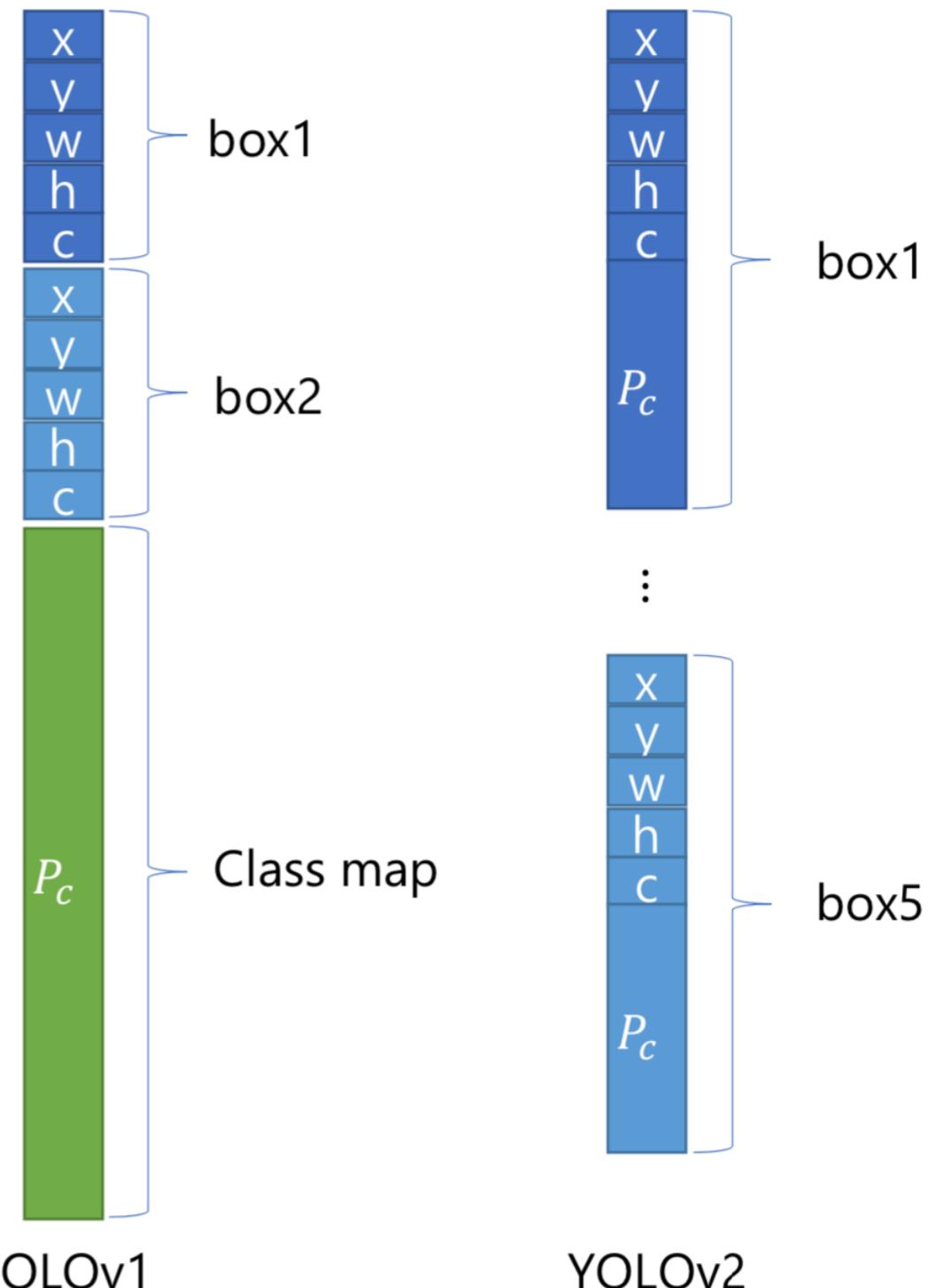
# YOLO V1 VS YOLO V2

## YOLOv1 YOLOv2 출력 형태 비교

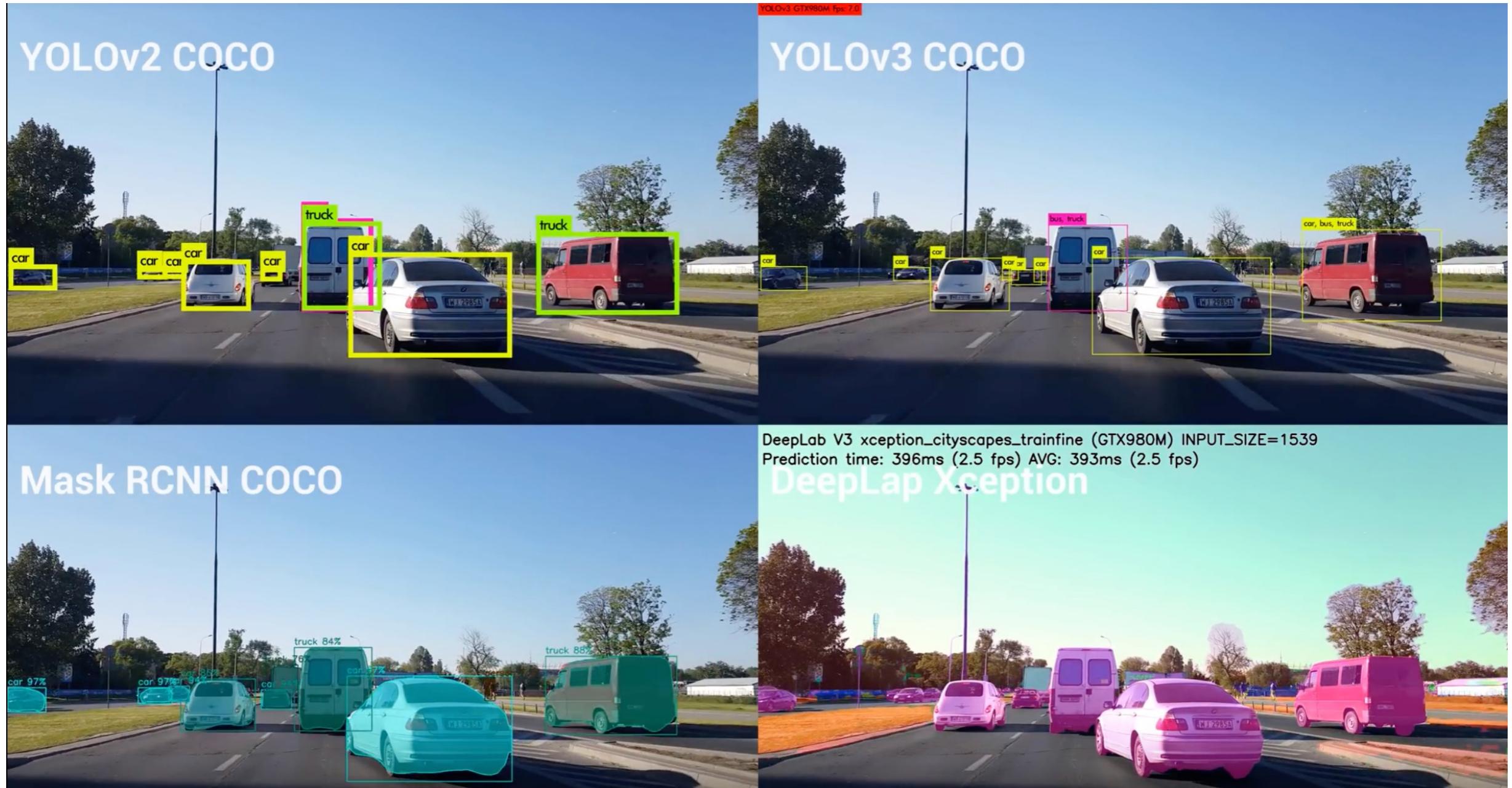
- YOLOv1:  $7 \times 7 \times (5 \times N_b + N_c)$
- YOLOv2:  $13 \times 13 \times (N_b \times (5 + N_c))$

$N_b$  : number of boxes

$N_c$  : number of classes



# YOLO v1 – NMS



[https://www.youtube.com/watch?v=s8Ui\\_kv9dhw](https://www.youtube.com/watch?v=s8Ui_kv9dhw)