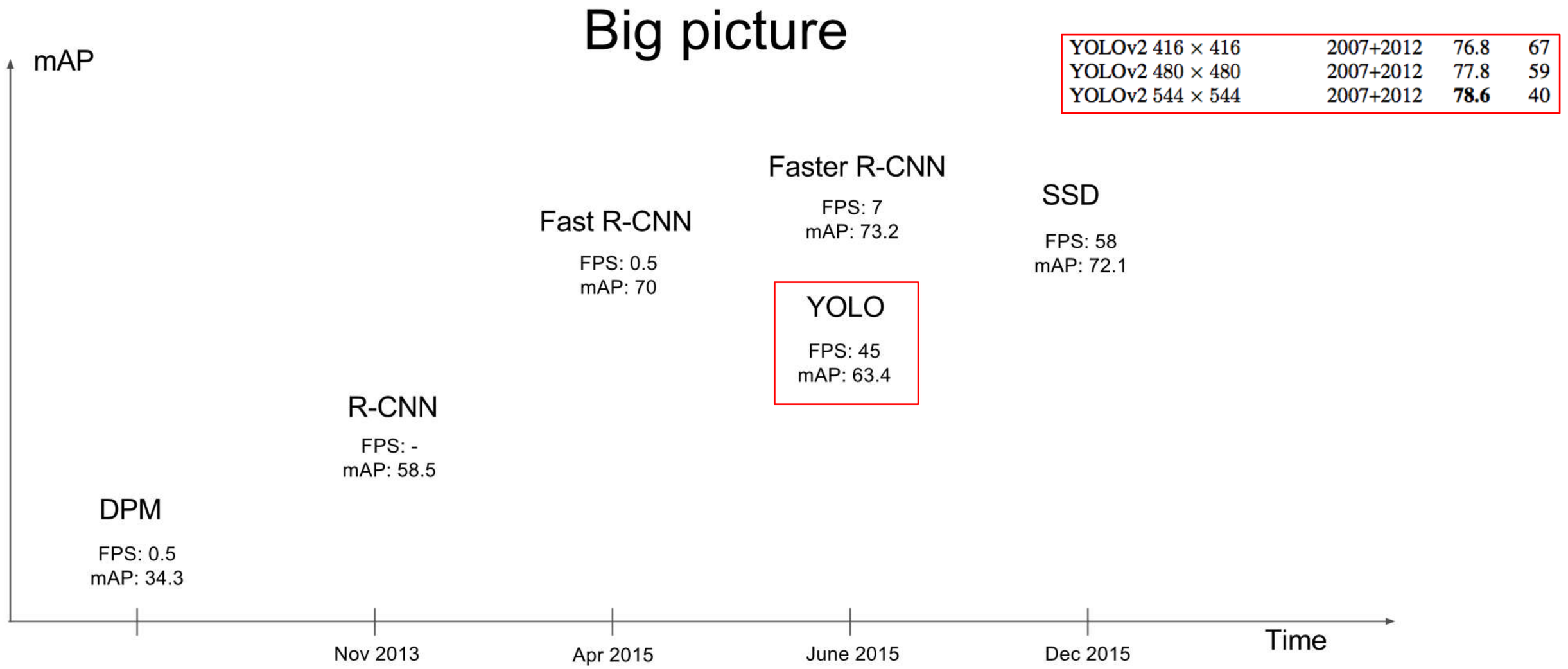
# You Only Look Once:
# Unified, Real-Time Object Detection (2016)


Taegyun Jeon

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

Evaluation on VOC2007




| | | | |
|---|---|---|---|
| YOLOv2 416 × 416 | 2007+2012 | 76.8 | 67 |
| YOLOv2 480 × 480 | 2007+2012 | 77.8 | 59 |
| YOLOv2 544 × 544 | 2007+2012 | **78.6** | 40 |

Slide courtesy of DeepSystem.io “YOLO: You only look once Review”

# Main Concept

- **Object Detection**
  - Regression problem
- **YOLO**
  - Only One Feedforward
  - Global context
- **Unified (Real-time detection)**
  - YOLO: 45 FPS
  - Fast YOLO: 155 FPS
- **General representation**
  - Robust on various background
  - Other domain


## Abstract

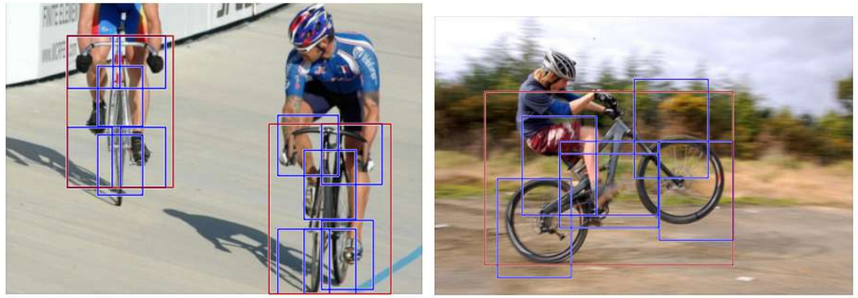
*We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding 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.*

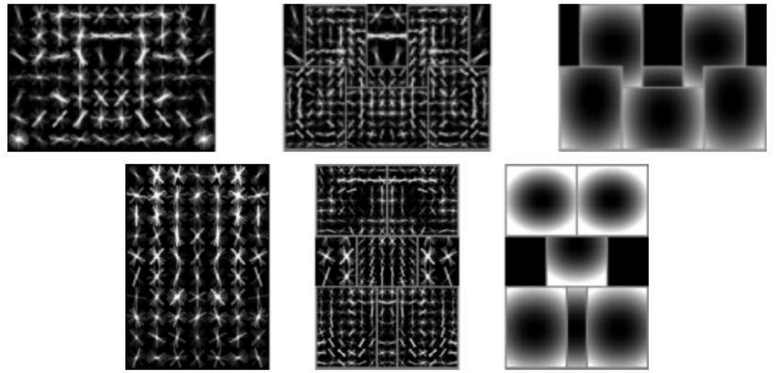
*Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art 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.*

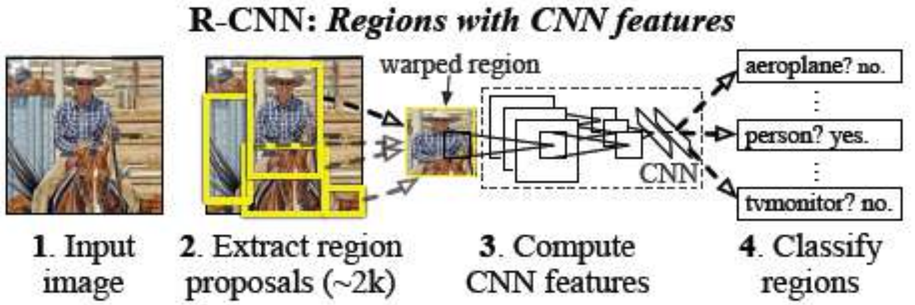

(Real-time system: https://cs.stackexchange.com/questions/56569/what-is-real-time-in-a-computer-vision-context)

# Object Detection as Regression Problem

- **Previous:** Repurpose classifiers to perform detection
  - **Deformable Parts Models (DPM)**
    - Sliding window
  - **R-CNN based methods**
    - 1) generate potential bounding boxes.
    - 2) run classifiers on these proposed boxes
    - 3) post-processing (refinement, elimination, rescore)







# Object Detection as Regression Problem

- **YOLO:** Single Regression Problem
  - Image → bounding box coordinate and class probability.
  - Extremely Fast
  - Global reasoning
  - Generalizable representation

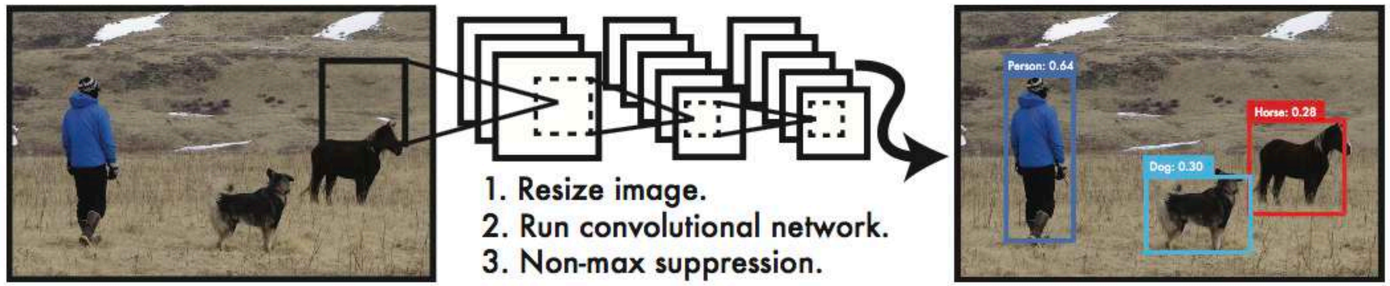



**Figure 1: The YOLO Detection System.** Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

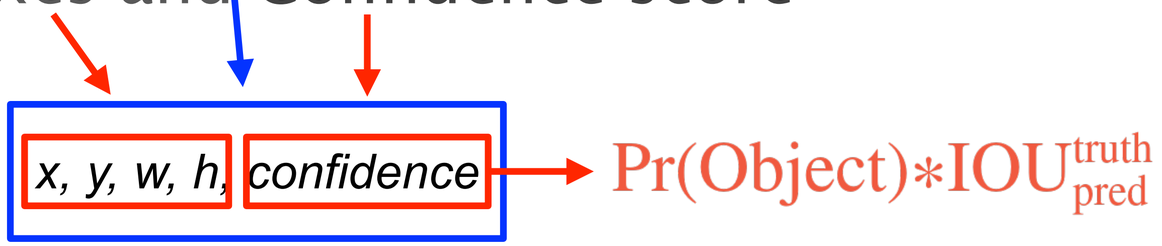
# Unified Detection

- All BBox, All classes

1) Image → **S x S** grids

2) grid cell

→ **B**: BBoxes and Confidence score




→ **C**: class probabilities w.r.t #classes

$Pr(Class_i|Object)$

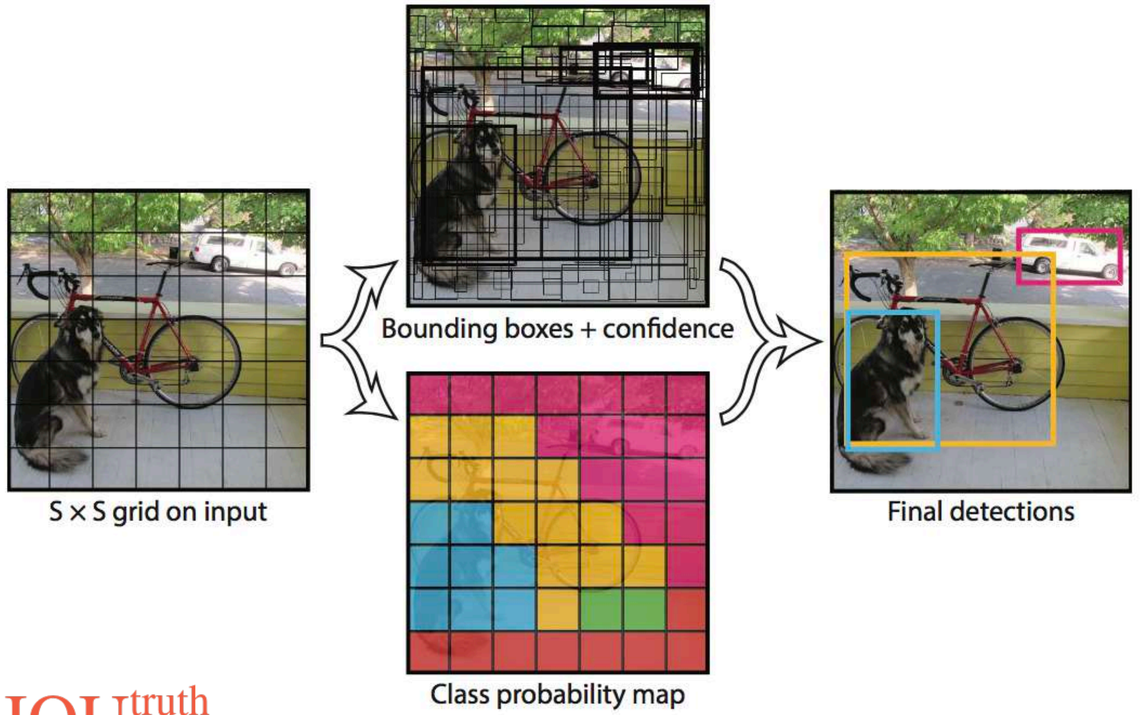



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts $B$ bounding boxes, confidence for those boxes, and $C$ class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$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

# Unified Detection

- Predict one set of class probabilities per grid cell, regardless of the number of boxes B.

- At test time, individual box confidence prediction

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

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

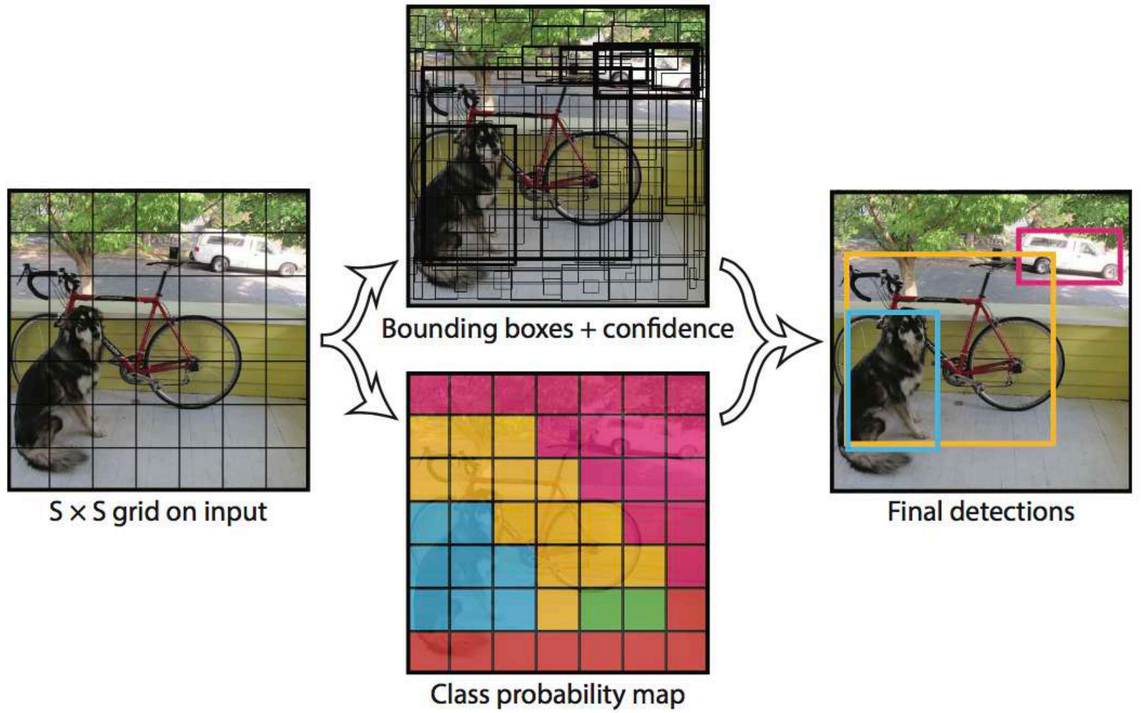
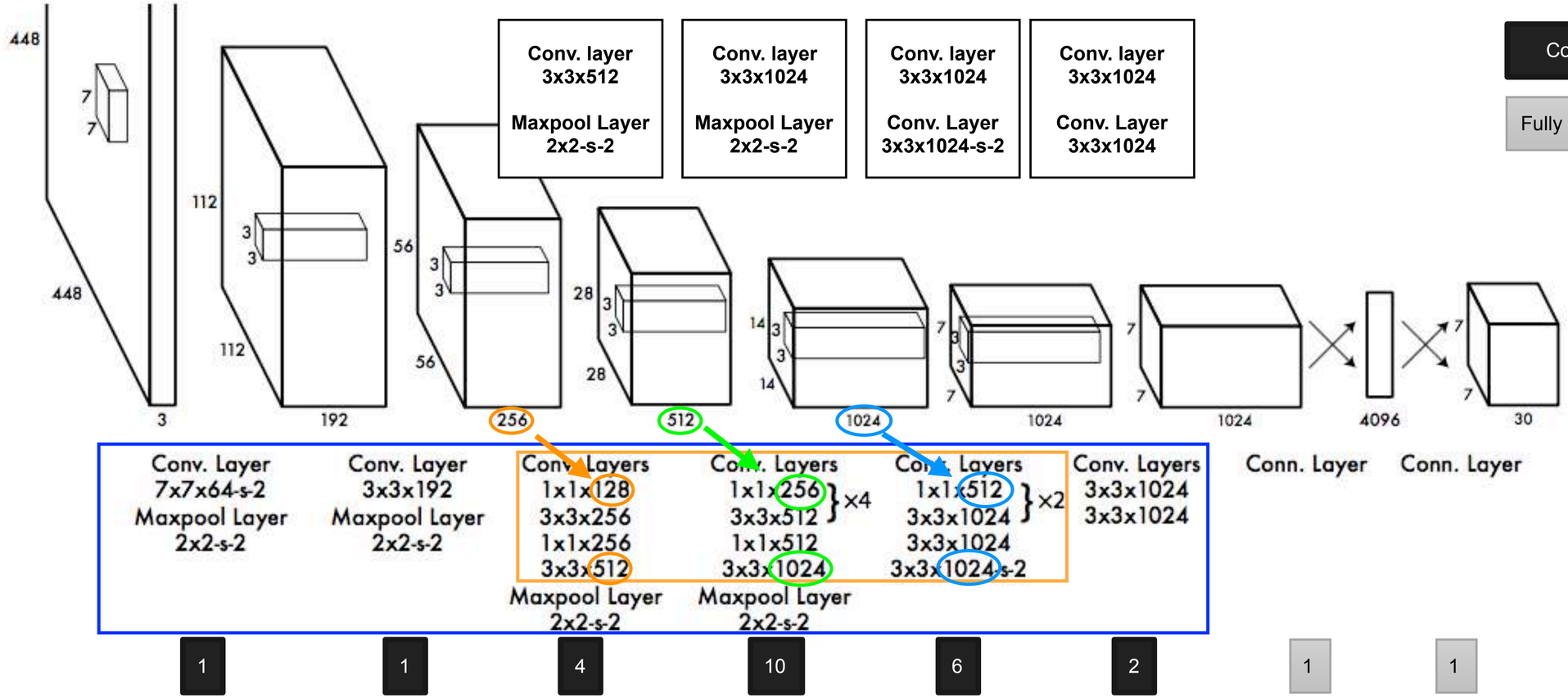



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts $B$ bounding boxes, confidence for those boxes, and $C$ class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$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

# Network Design: YOLO

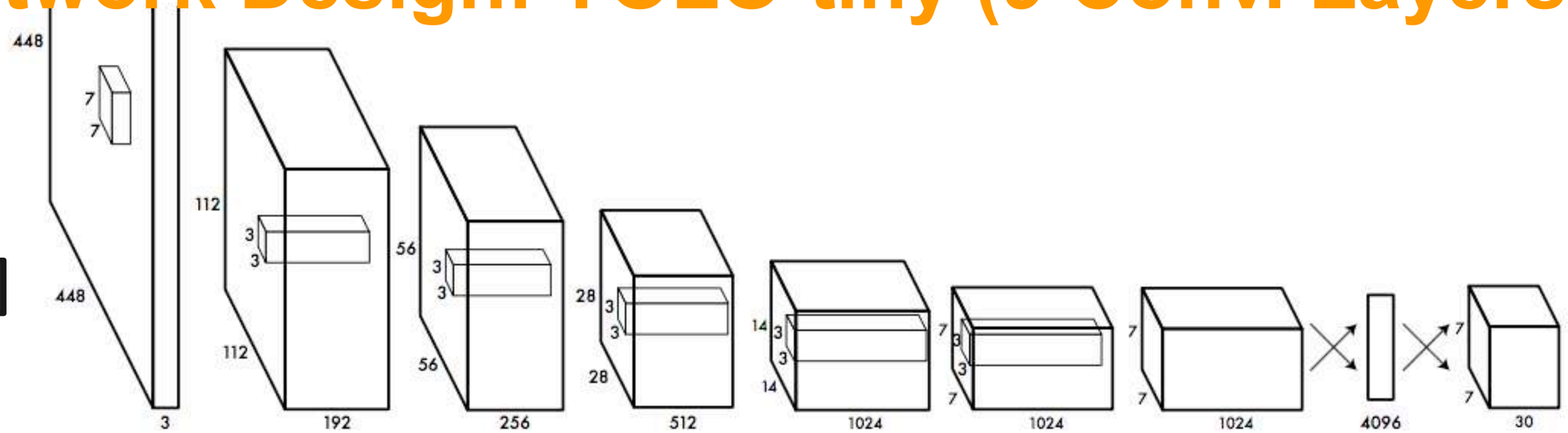
- Modified GoogLeNet
- 1x1 reduction layer ("Network in Network")

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.

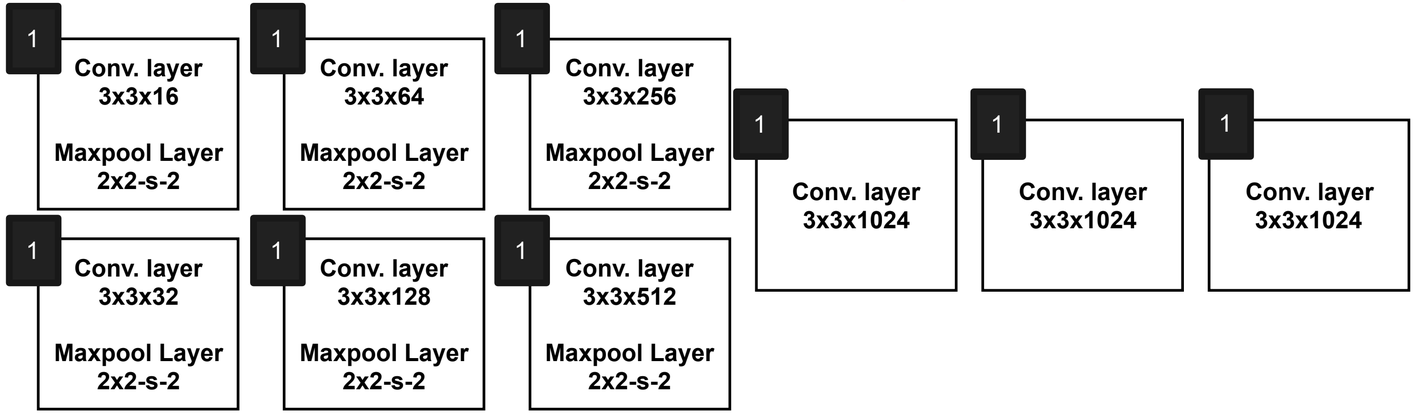


# Network Design: YOLO-tiny (9 Conv. Layers)

YOLO




YOLO-Tiny

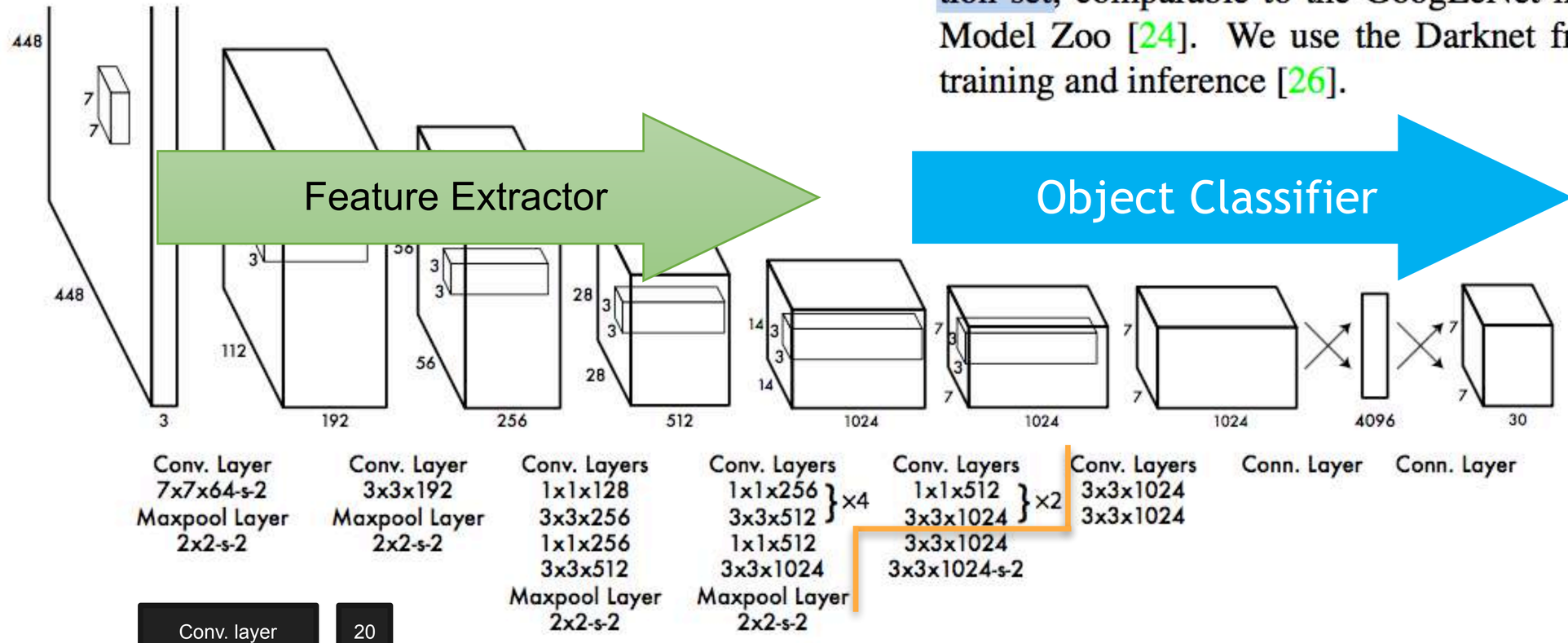



| | |
|---|---|
| Conv. layer | 9 |
| Fully conn. Layer | 2 |

# 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].

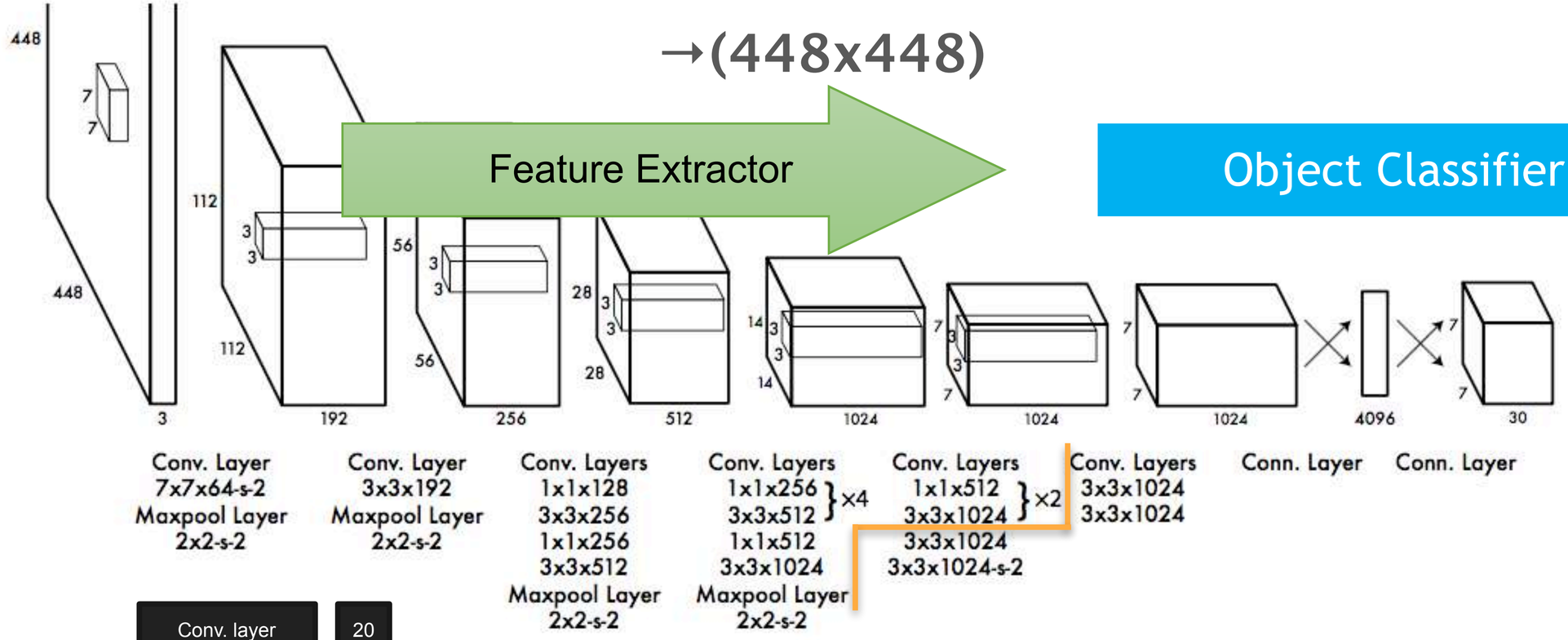


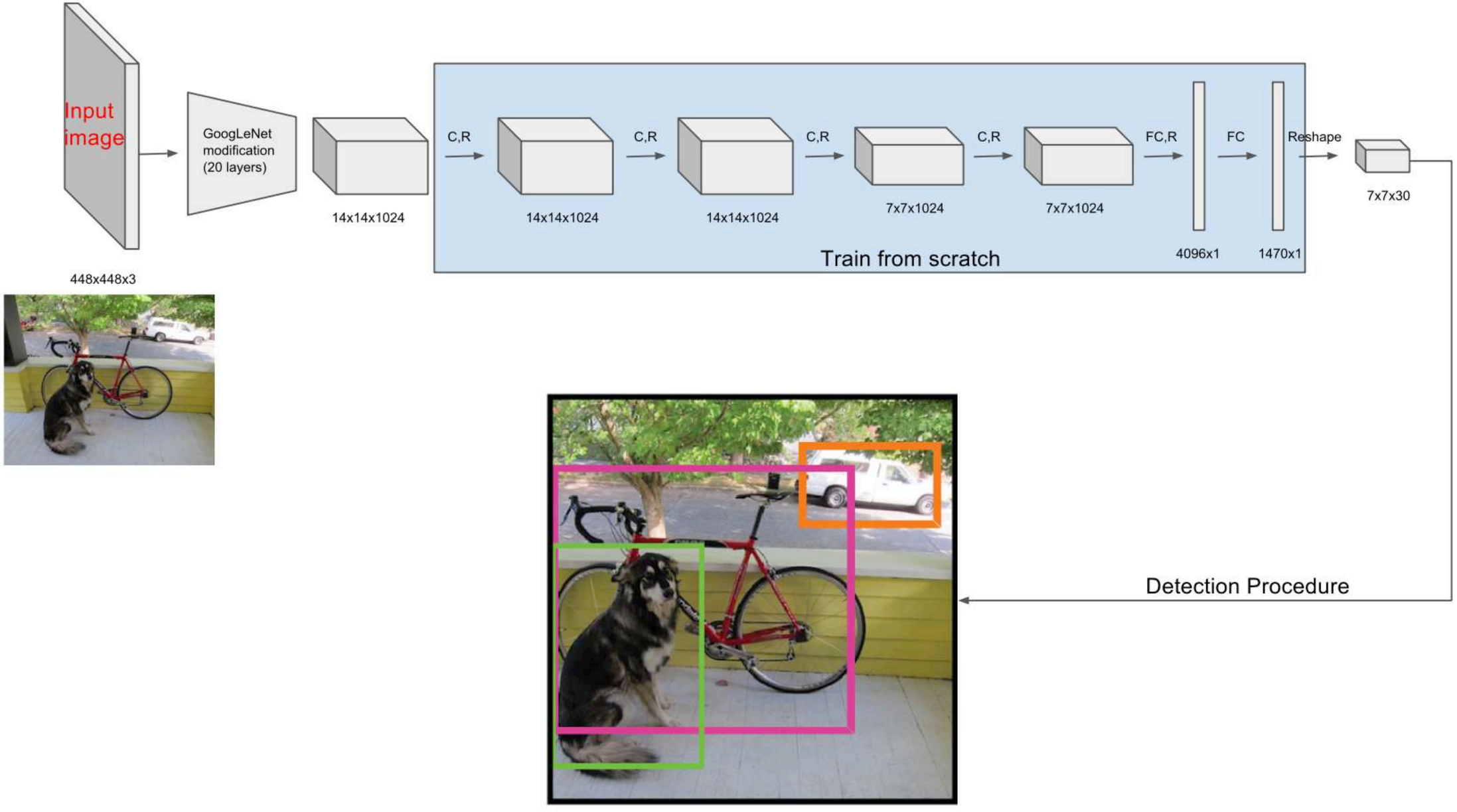
# Training

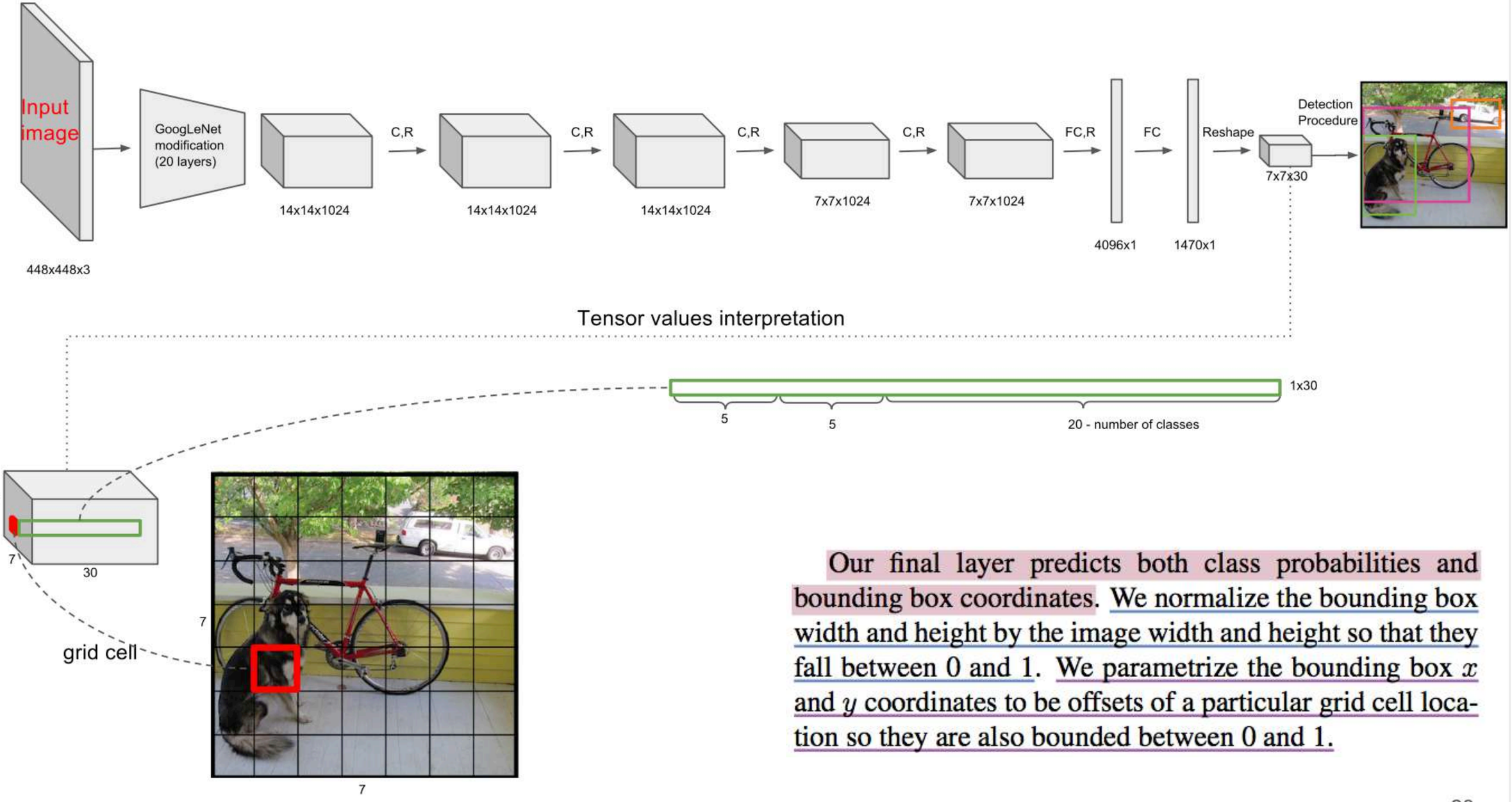
2) “Network on Convolutional Feature Maps”

Increased input resolution (224x224)

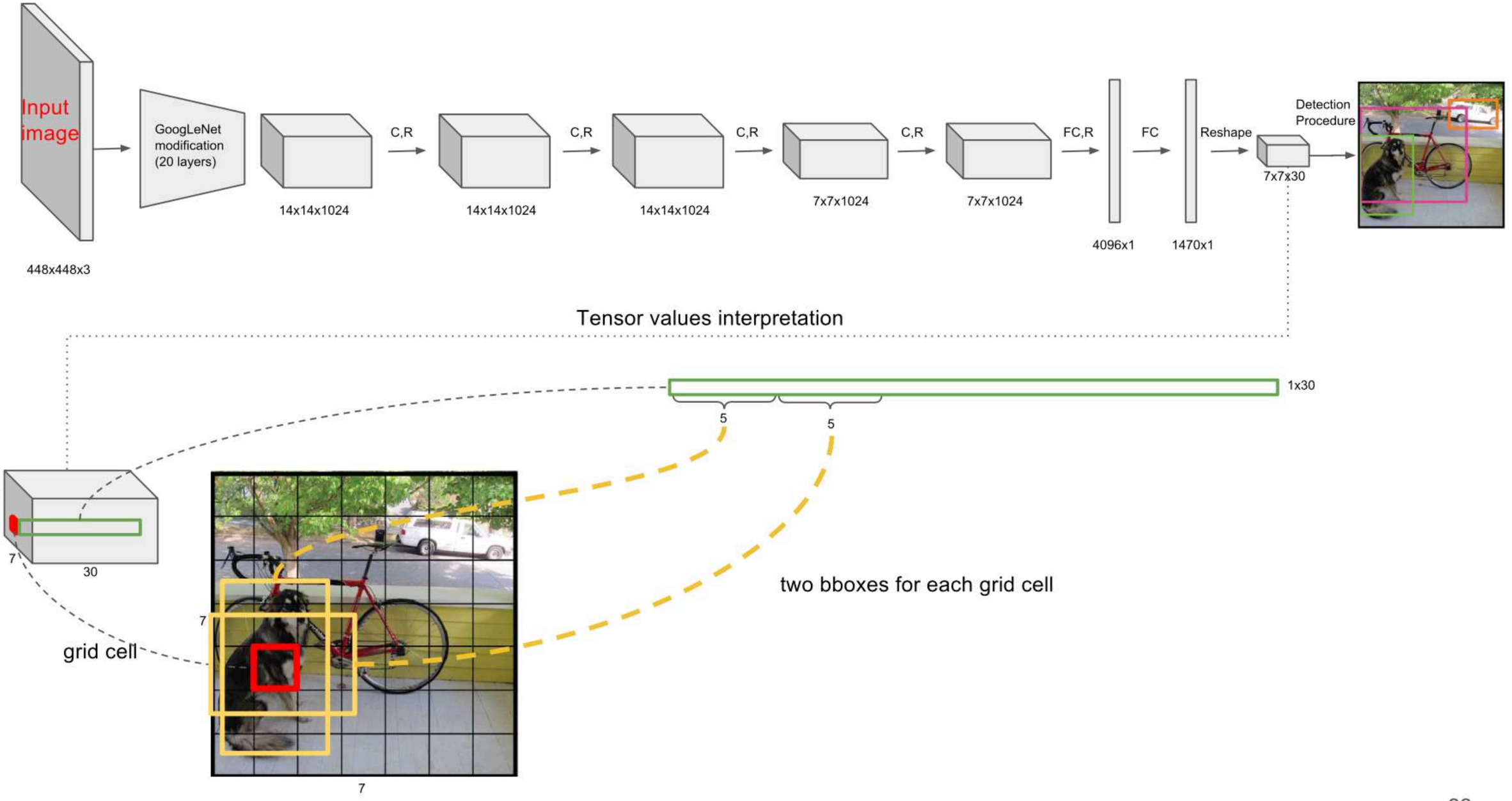
We then convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance [29]. Following their example, we add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from 224 × 224 to 448 × 448.

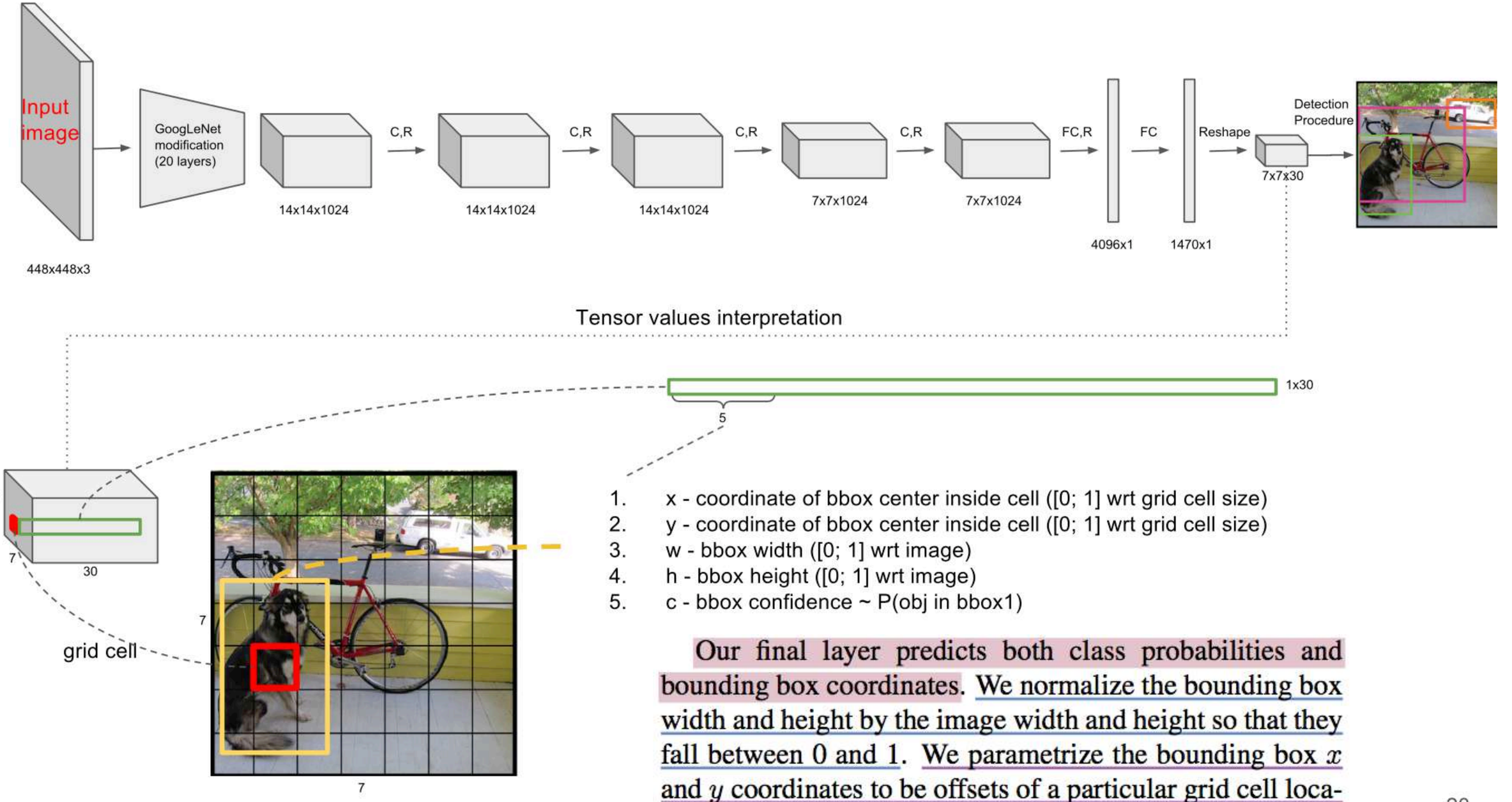



Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Train from scratch
Detection Procedure




Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box $x$ and $y$ coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1.


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
1x30
5
5
7
7
30
grid cell
7
7
two bboxes for each grid cell




Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box $x$ and $y$ coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1.

# Loss Function (sum-squared error)

loss function:

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2 \quad (3)$$

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 $\lambda_{\text{noobj}} = .5$.

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

# Loss Function (sum-squared error)

loss function:

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

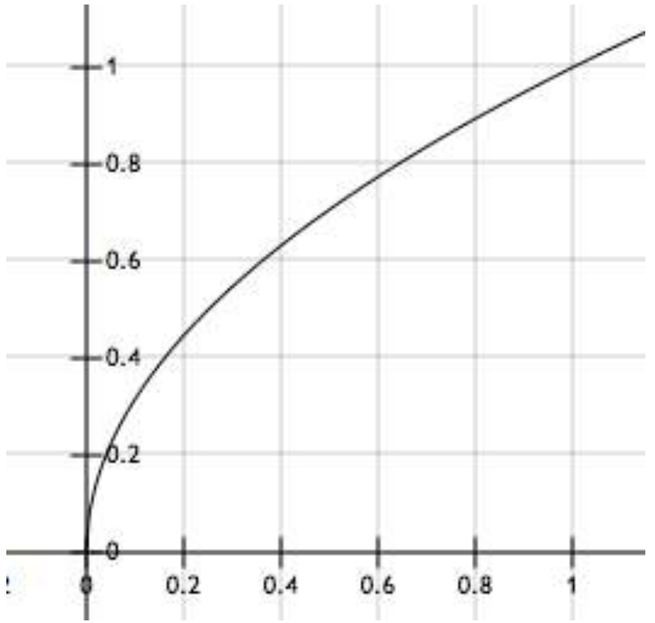
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)$$

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.



# Loss Function (sum-squared error)

loss function:

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)$$

$\mathbb{1}_{ij}^{obj}$ **The *j*th bbox predictor** in ***cell i*** is "responsible" for that prediction

$\mathbb{1}_{ij}^{noobj}$

$\mathbb{1}_{i}^{obj}$ If object appears in ***cell i***

Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).

# Train strategy

**epochs**=135

**batch_size**=64

**momentum_a** = 0.9

**decay**=0.0005

**lr**=[$10^{-3}$, $10^{-2}$, $10^{-3}$, $10^{-4}$]

**dropout_rate**=0.5

**augmentation**
=[scaling, translation, exposure, saturation]

We train the network for about 135 epochs on the training and validation data sets from PASCAL VOC 2007 and 2012. When testing on 2012 we also include the VOC 2007 test data for training. Throughout training we use a batch size of 64, a momentum of 0.9 and a decay of 0.0005.

Our learning rate schedule is as follows: For the first epochs we slowly raise the learning rate from $10^{-3}$ to $10^{-2}$. If we start at a high learning rate our model often diverges due to unstable gradients. We continue training with $10^{-2}$ for 75 epochs, then $10^{-3}$ for 30 epochs, and finally $10^{-4}$ for 30 epochs.

To avoid overfitting we use dropout and extensive data augmentation. A dropout layer with rate = .5 after the first connected layer prevents co-adaptation between layers [18]. For data augmentation we introduce random scaling and translations of up to 20% of the original image size. We also randomly adjust the exposure and saturation of the image by up to a factor of 1.5 in the HSV color space.

# Inference

Just like in training?

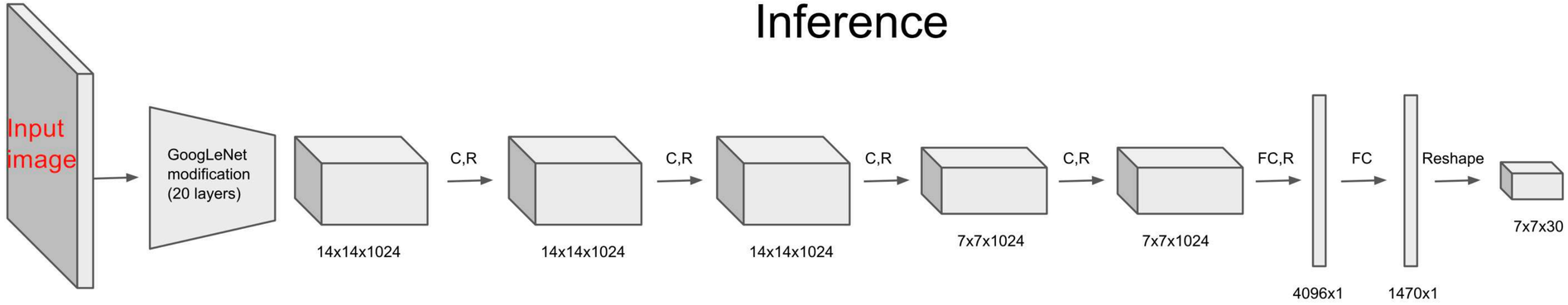
**S=7, B=2 for Pascal VOC**

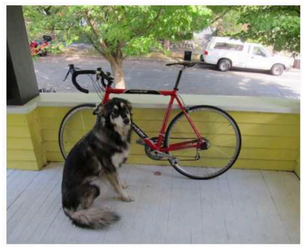
### 2.3. Inference

Just like in training, predicting detections for a test image only requires one network evaluation. On PASCAL VOC the network predicts 98 bounding boxes per image and class probabilities for each box. YOLO is extremely fast at test time since it only requires a single network evaluation, unlike classifier-based methods.

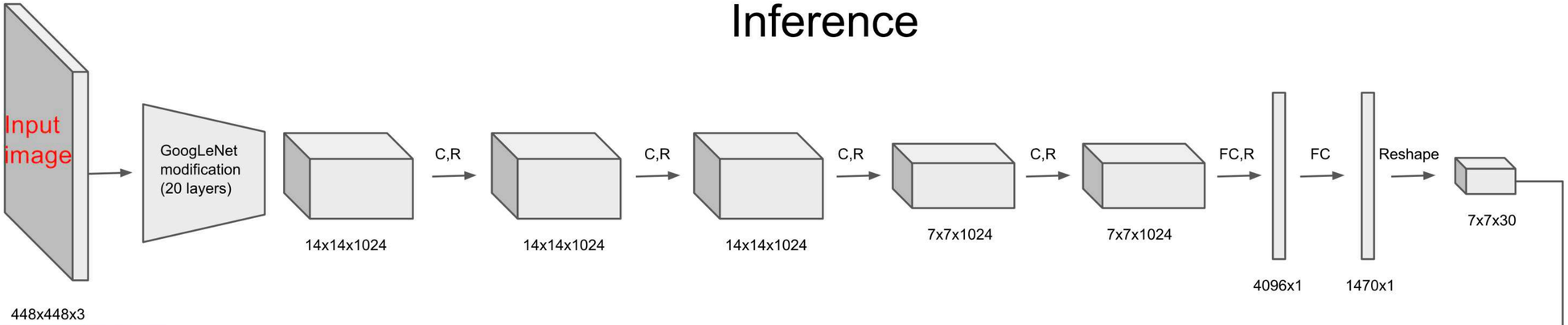
The grid design enforces spatial diversity in the bounding box predictions. Often it is clear which grid cell an object falls in to and the network only predicts one box for each object. However, some large objects or objects near the border of multiple cells can be well localized by multiple cells. Non-maximal suppression can be used to fix these multiple detections. While not critical to performance as it is for R-CNN or DPM, non-maximal suppression adds 2-3% in mAP.

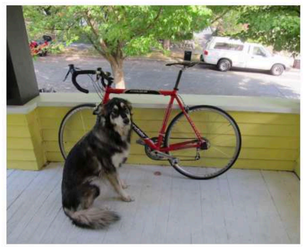
# Inference

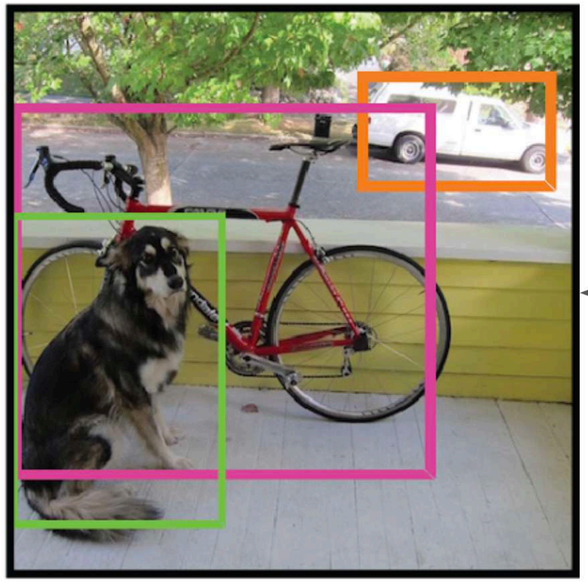

Input image
GoogLeNet modification (20 layers)
C,R
C,R
C,R
C,R
FC,R
FC
Reshape
448x448x3
14x14x1024
14x14x1024
14x14x1024
7x7x1024
7x7x1024
4096x1
1470x1
7x7x30



# Inference

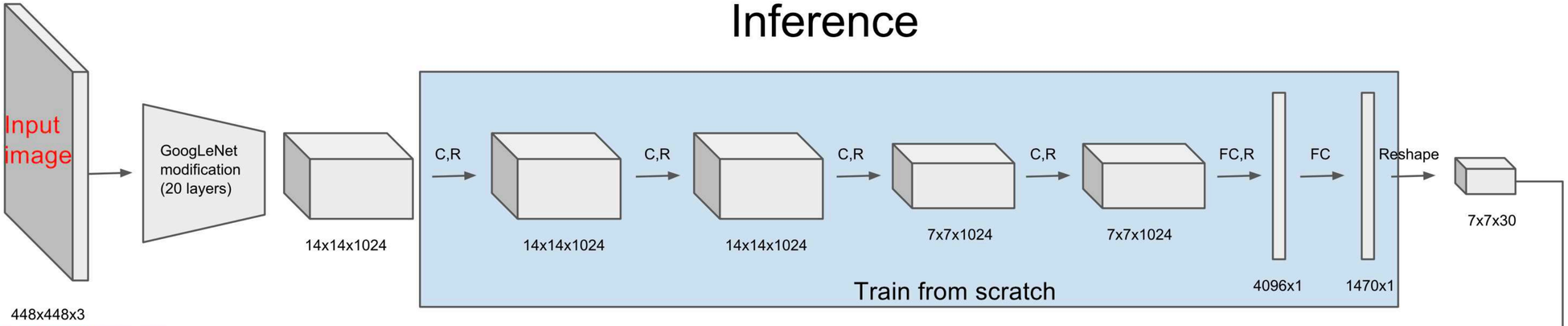


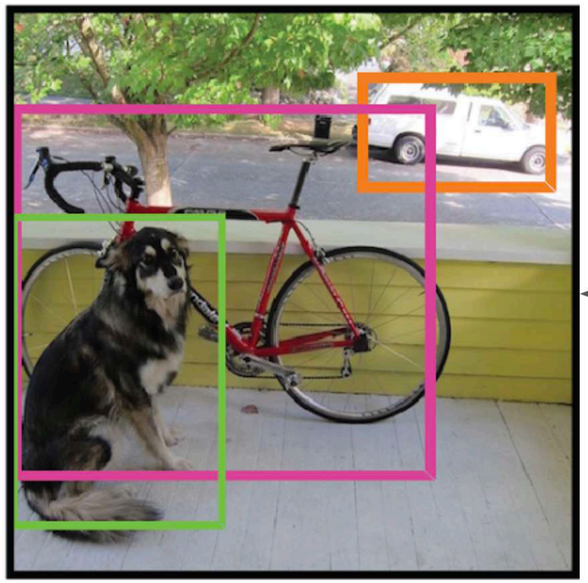





Detection Procedure

# Inference

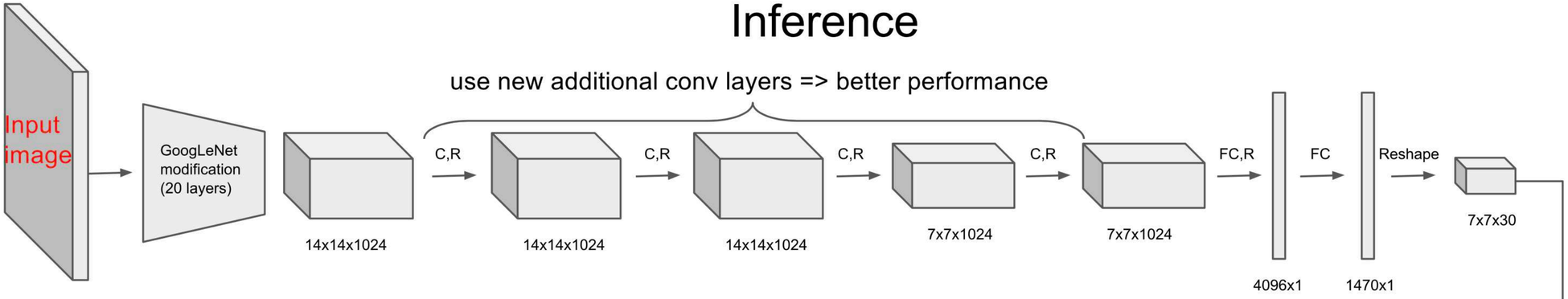


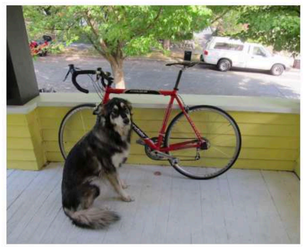



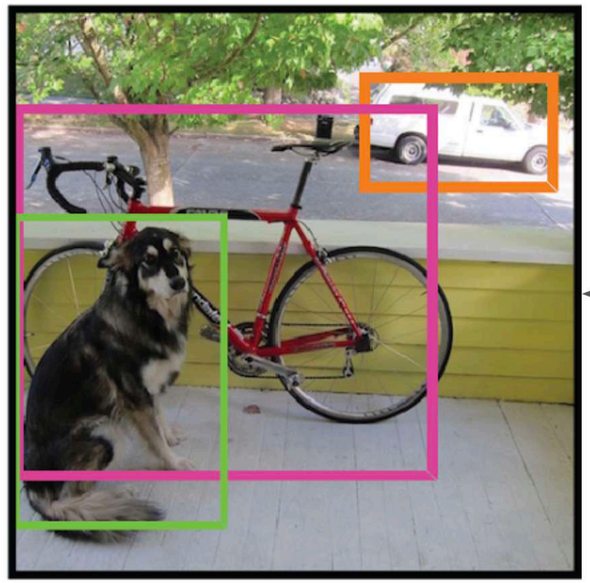
Detection Procedure

# Inference

use new additional conv layers => better performance

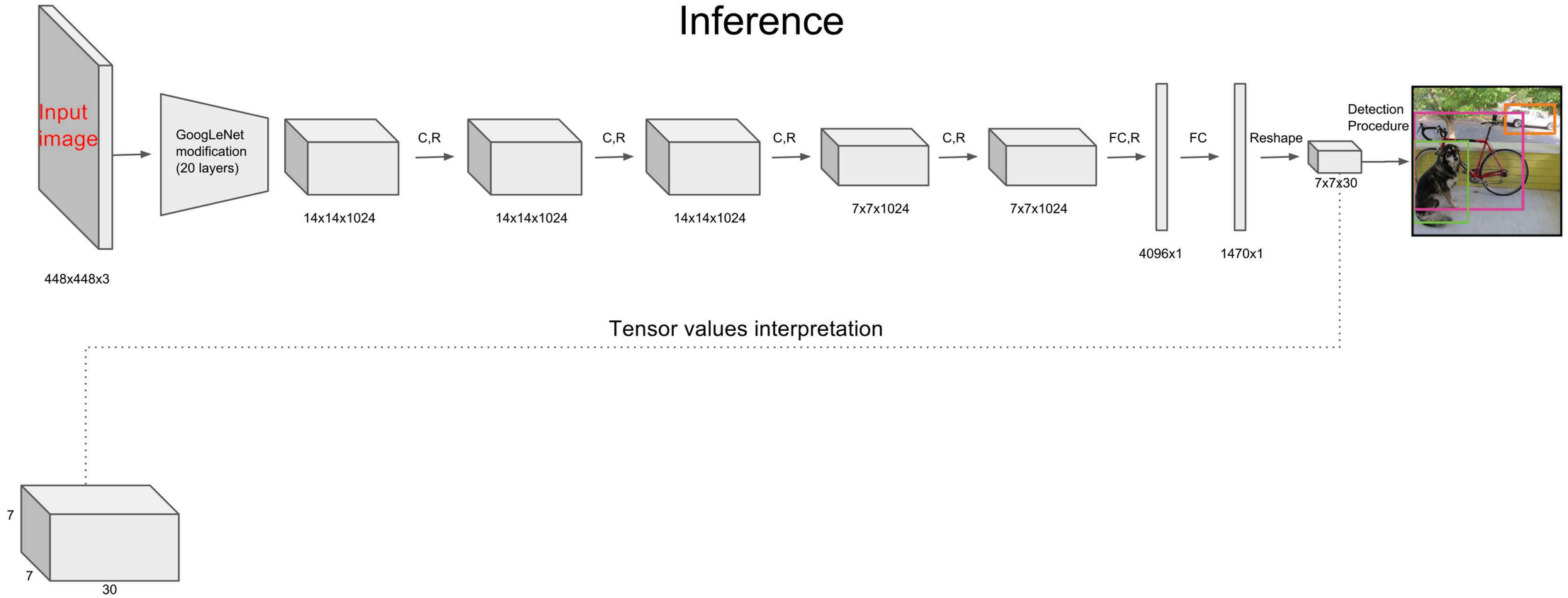


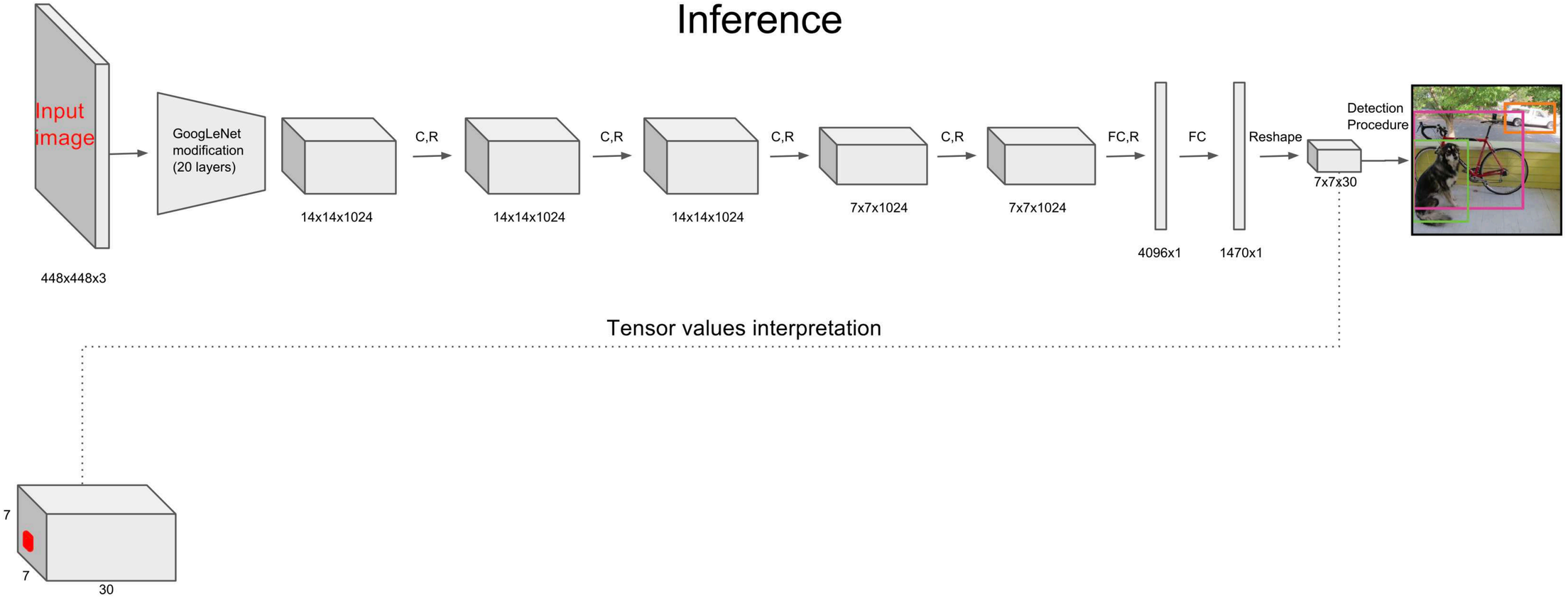


Detection Procedure

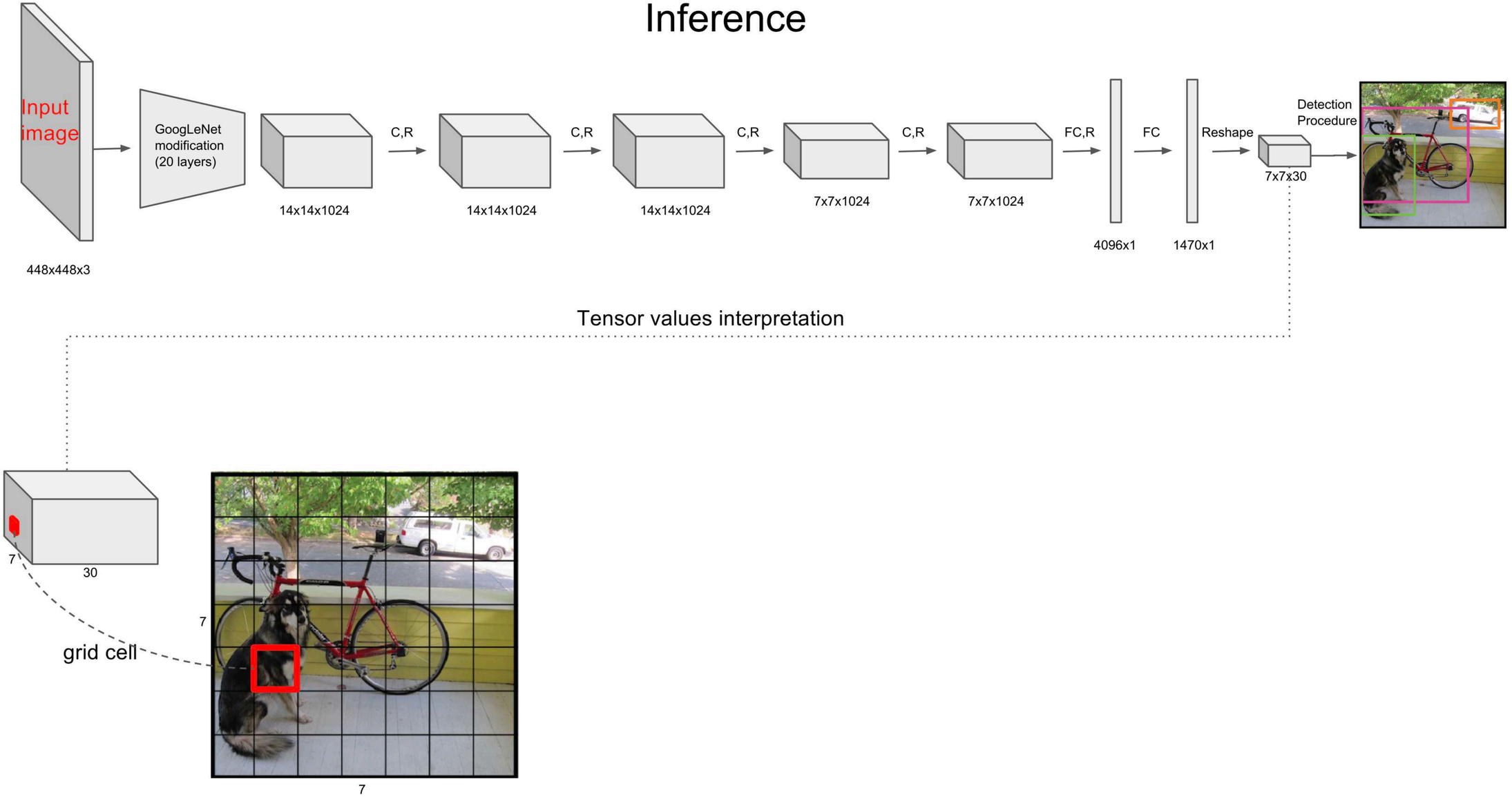
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
7
7
30

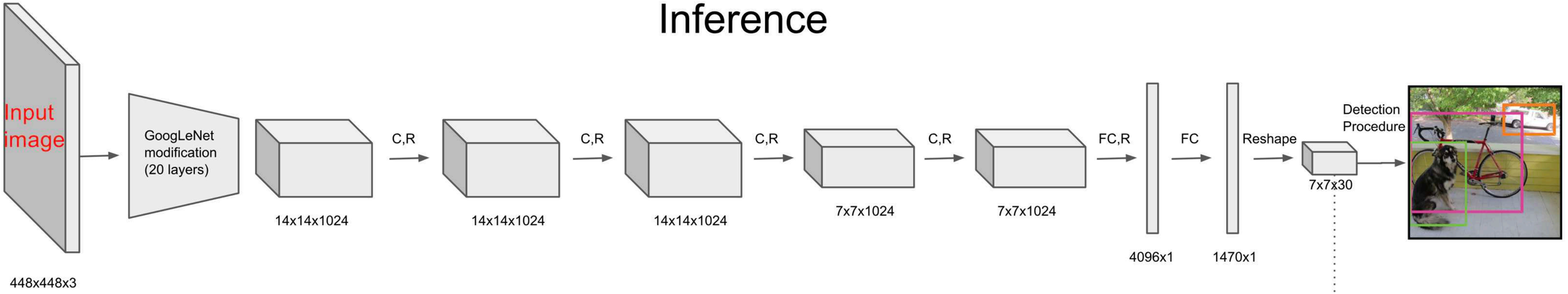
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
7
7
30

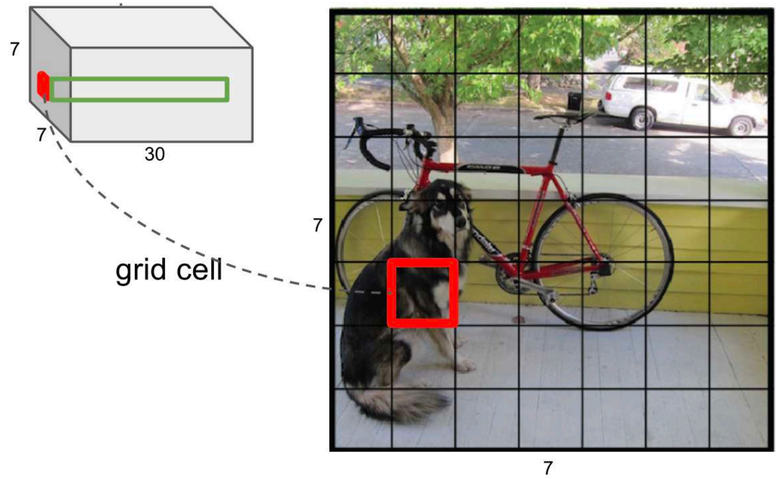
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
7
7
30
grid cell
7
7

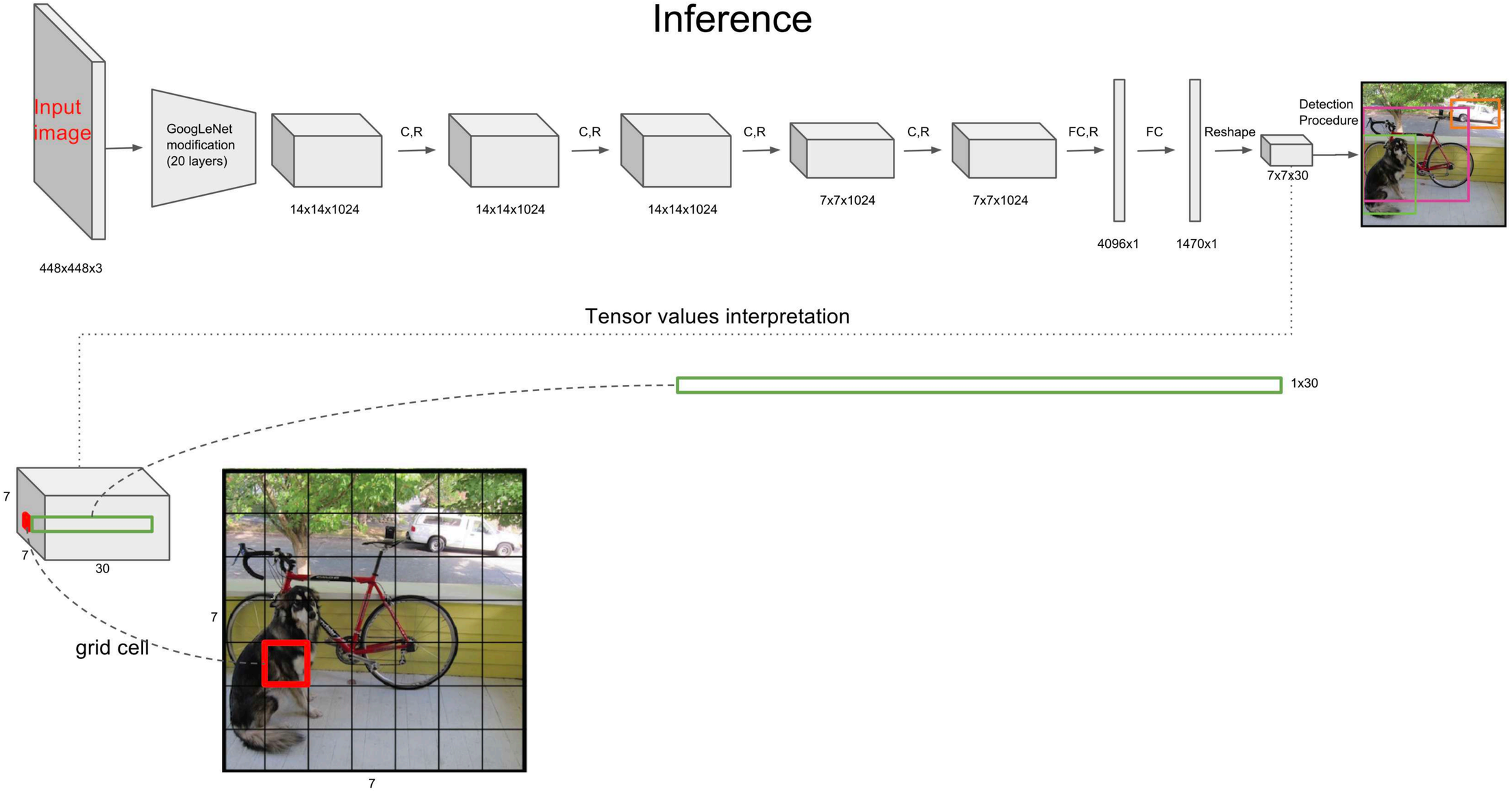
# Inference



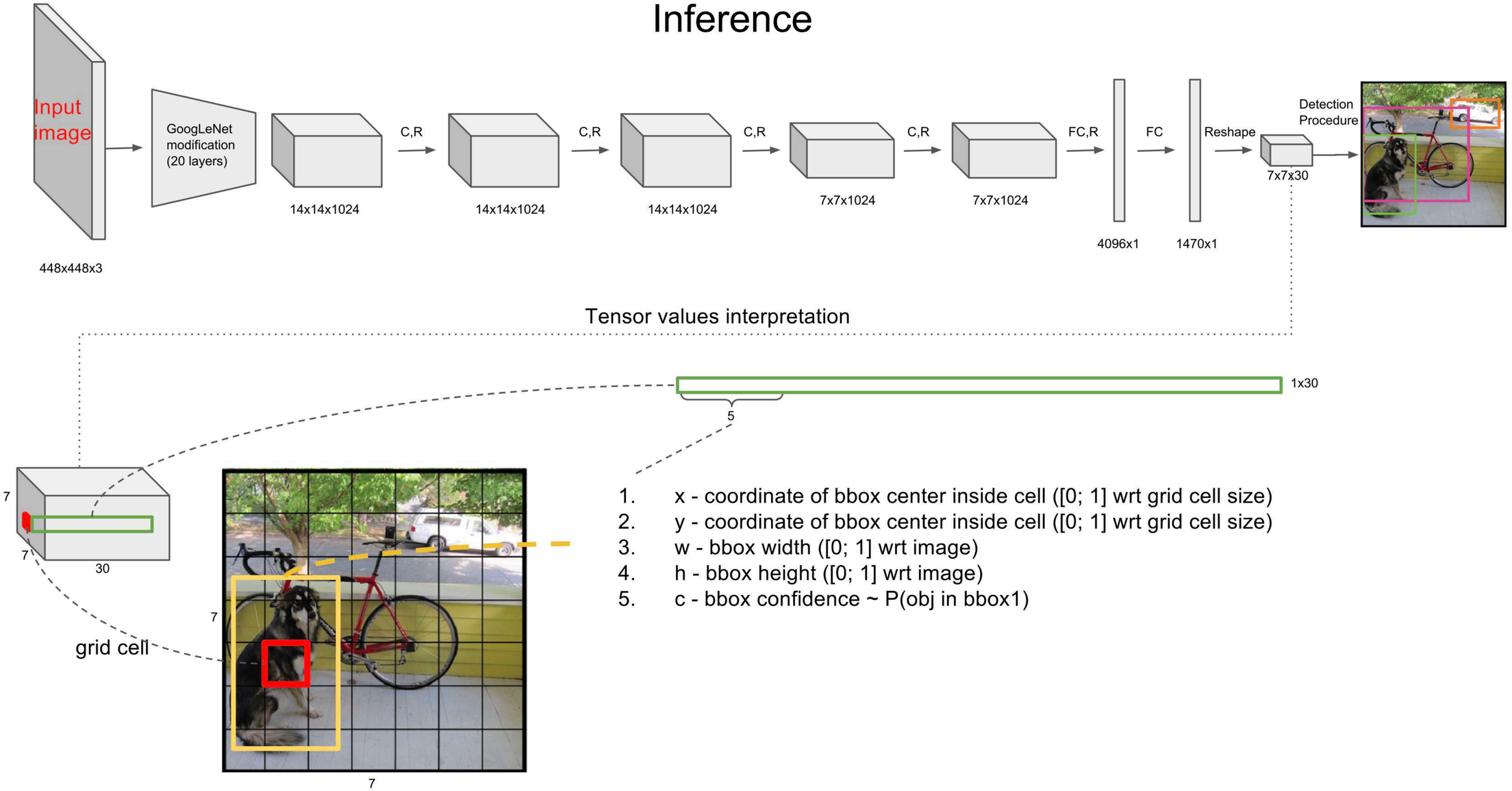

Tensor values interpretation



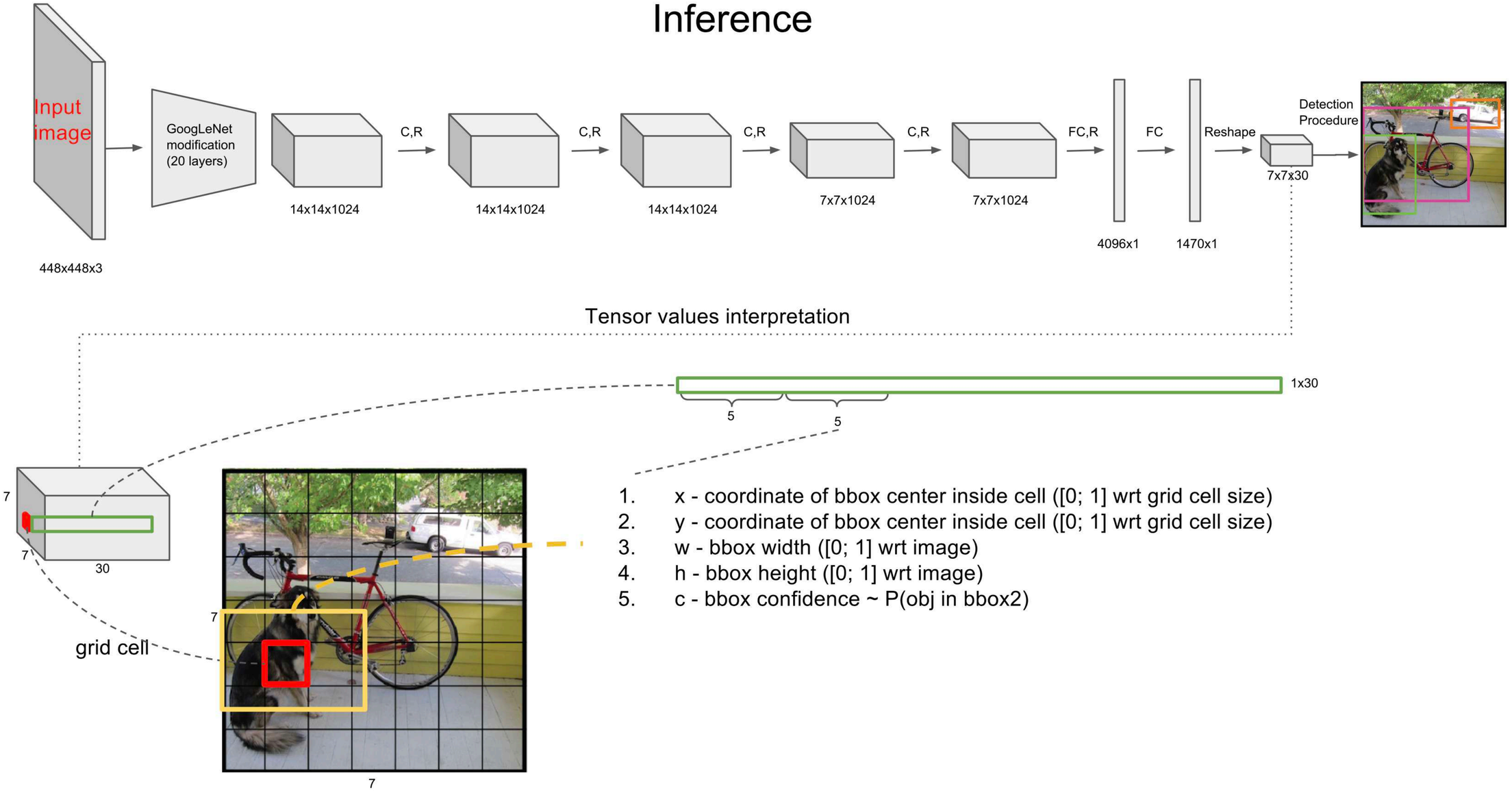
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
1x30
7
7
30
grid cell
7
7

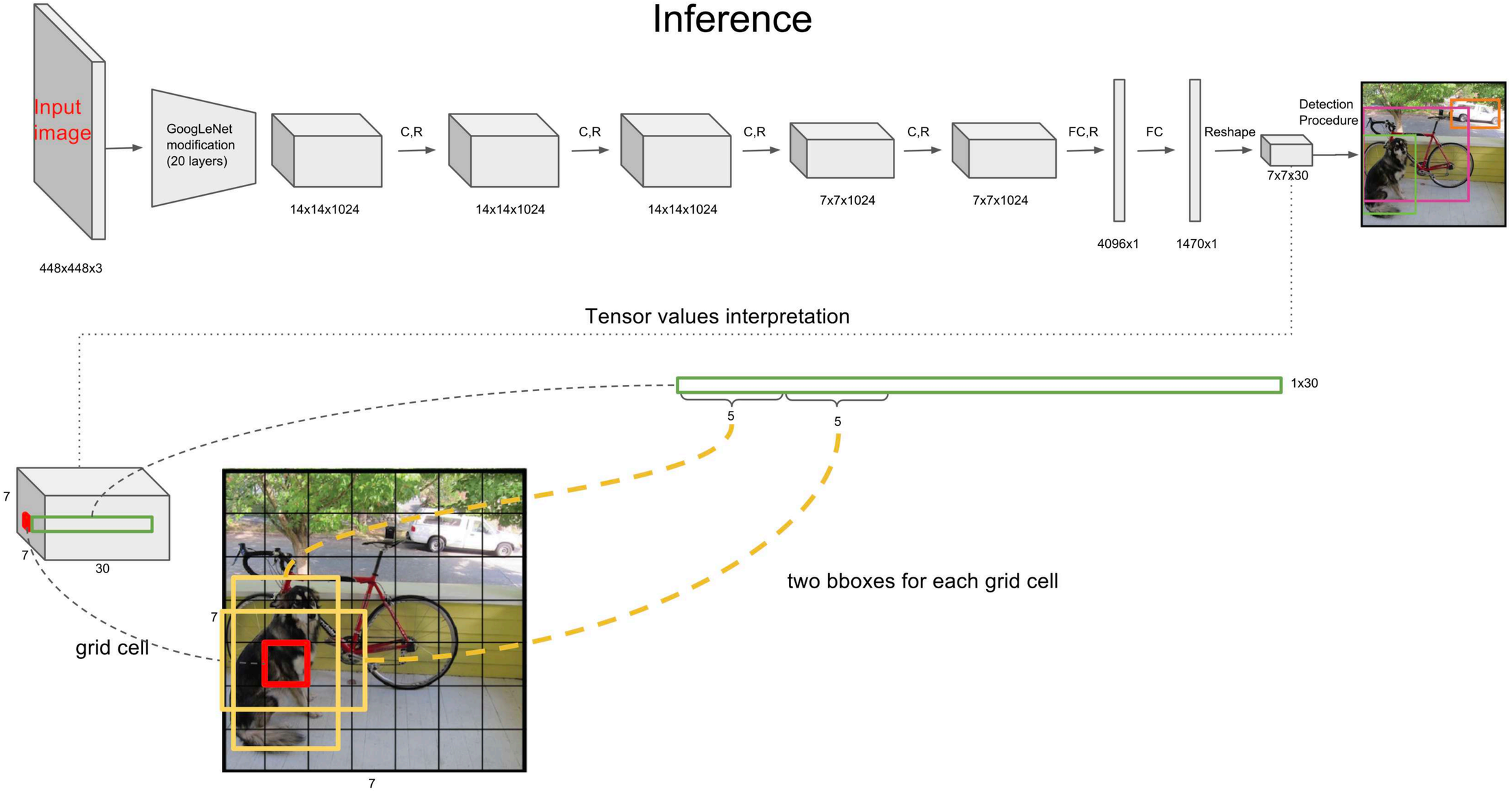
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
1x30
5
7
7
30
grid cell
7
7
1. x - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
2. y - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
3. w - bbox width ([0; 1] wrt image)
4. h - bbox height ([0; 1] wrt image)
5. c - bbox confidence ~ P(obj in bbox1)

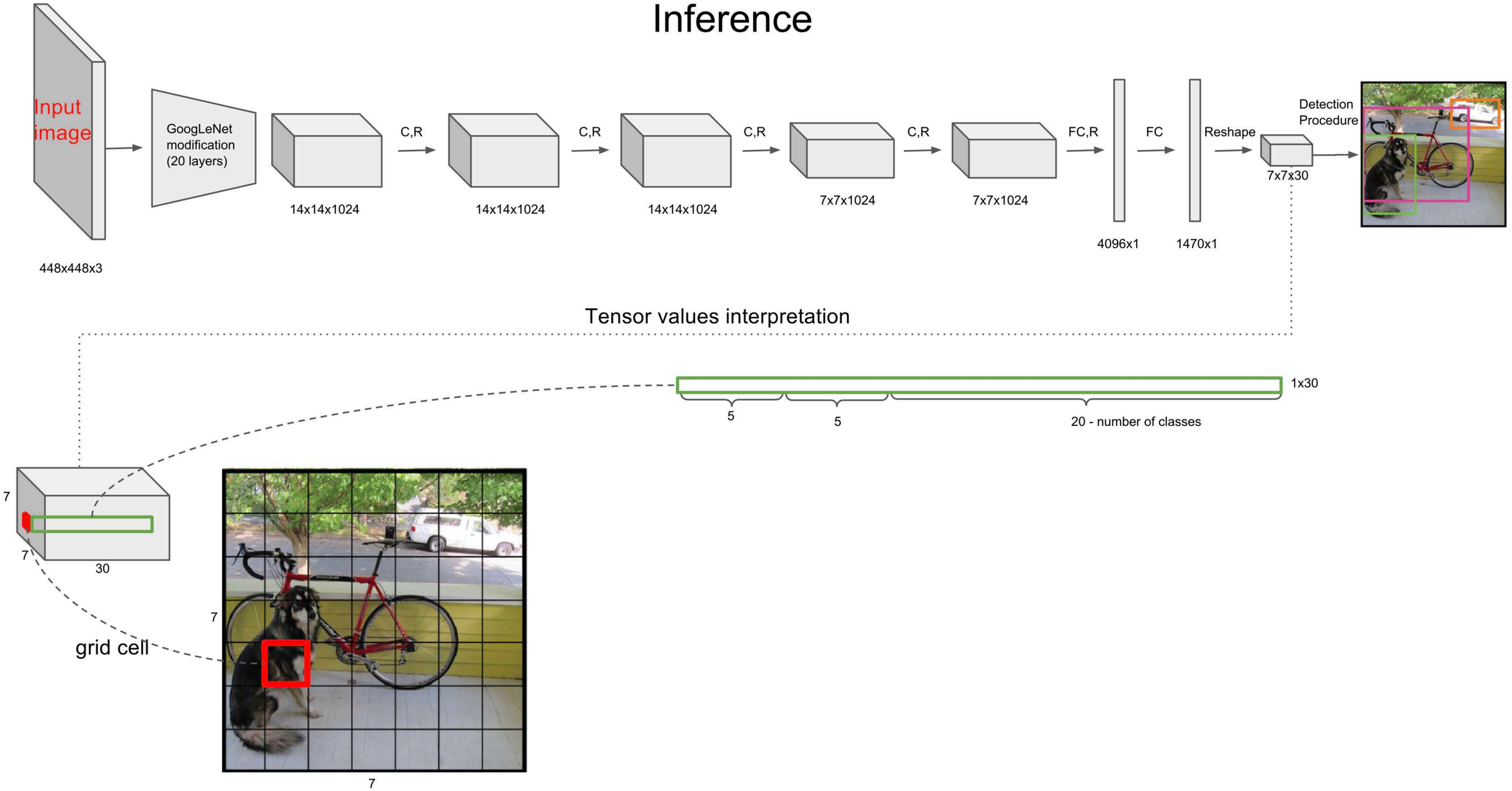
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
1x30
5
5
7
7
30
grid cell
7
7
1. x - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
2. y - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
3. w - bbox width ([0; 1] wrt image)
4. h - bbox height ([0; 1] wrt image)
5. c - bbox confidence ~ P(obj in bbox2)

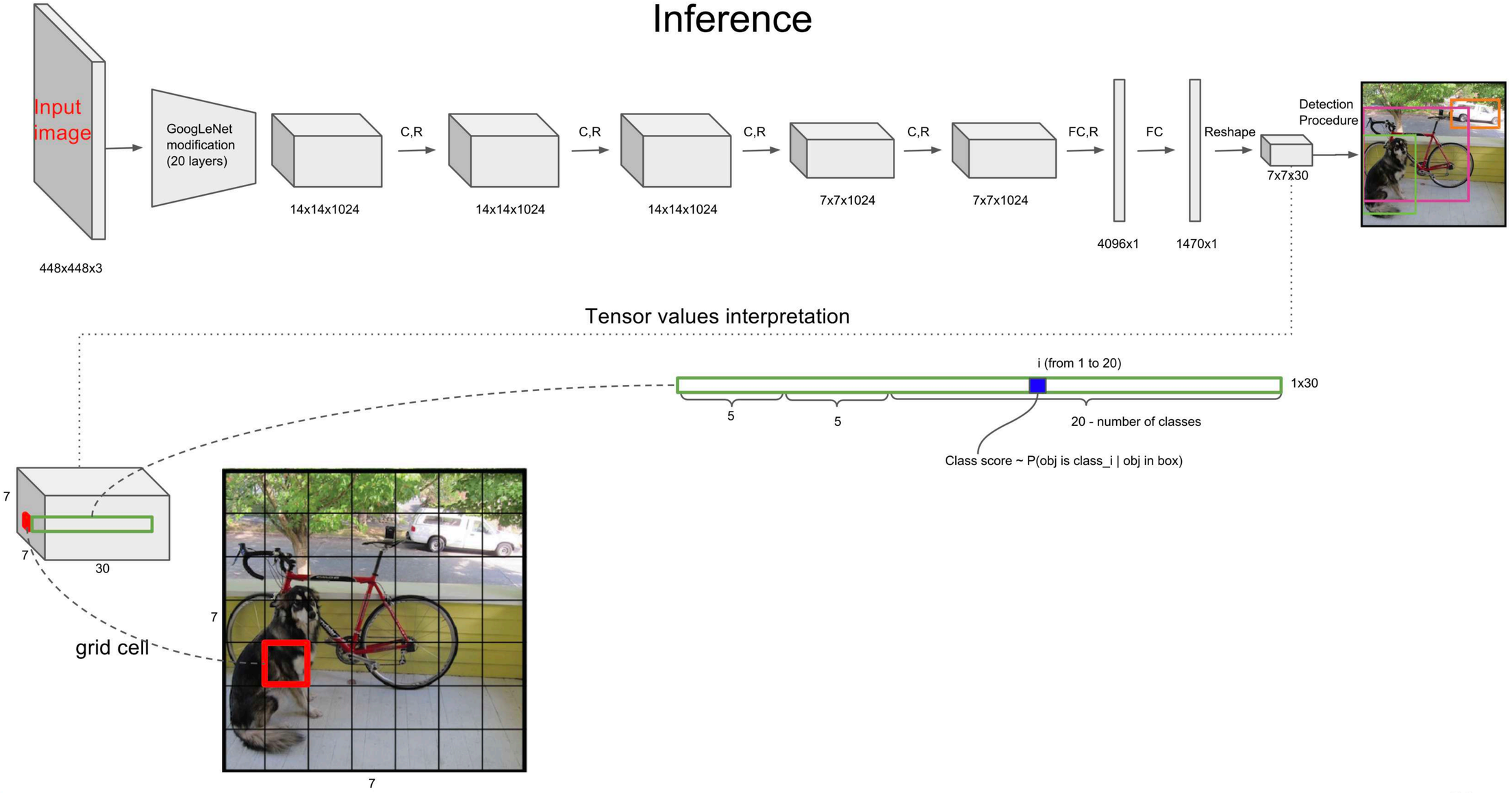
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
1x30
5
5
7
7
30
grid cell
7
7
two bboxes for each grid cell

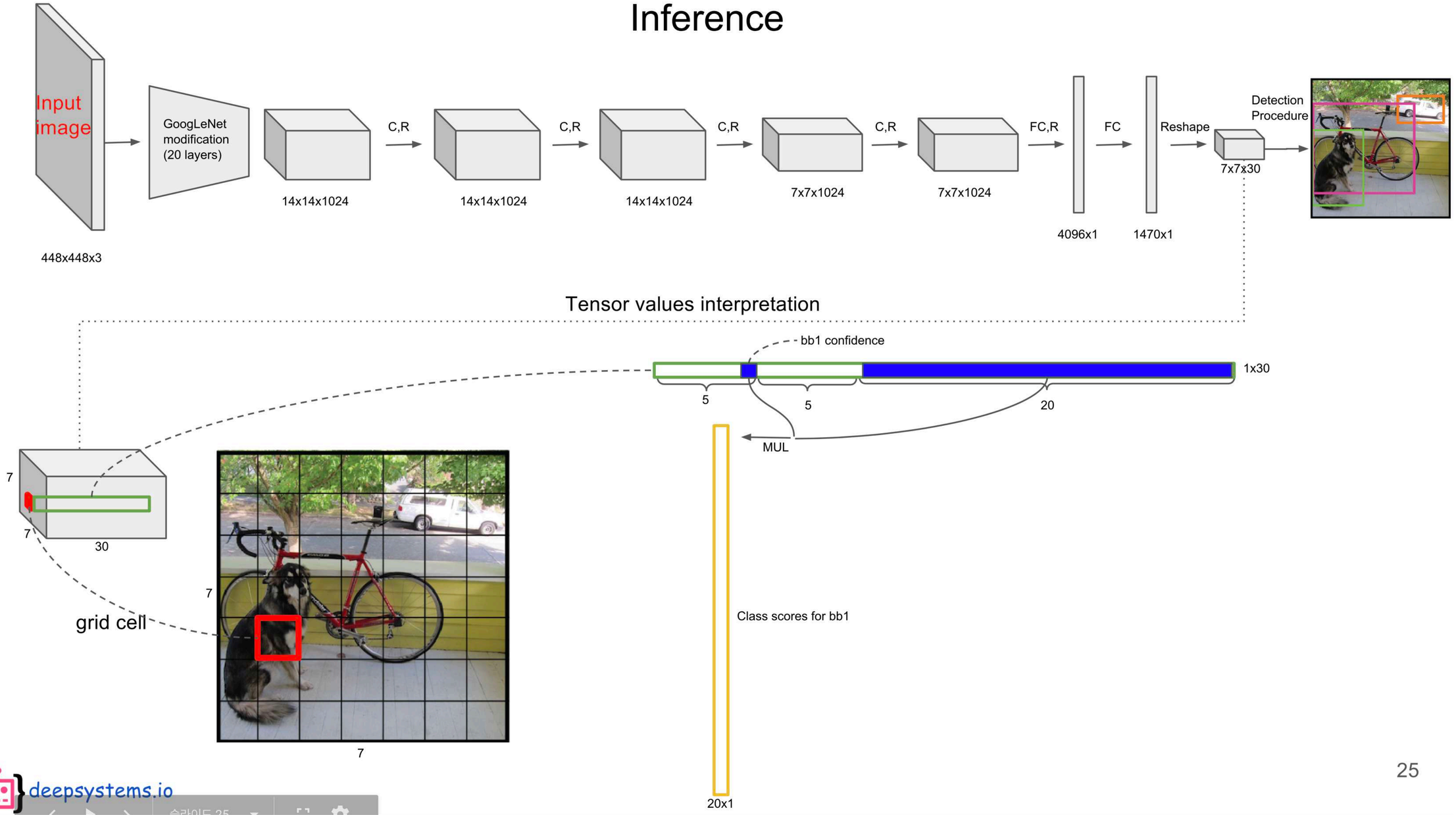
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
1x30
5
5
20 - number of classes
7
7
30
7
grid cell
7

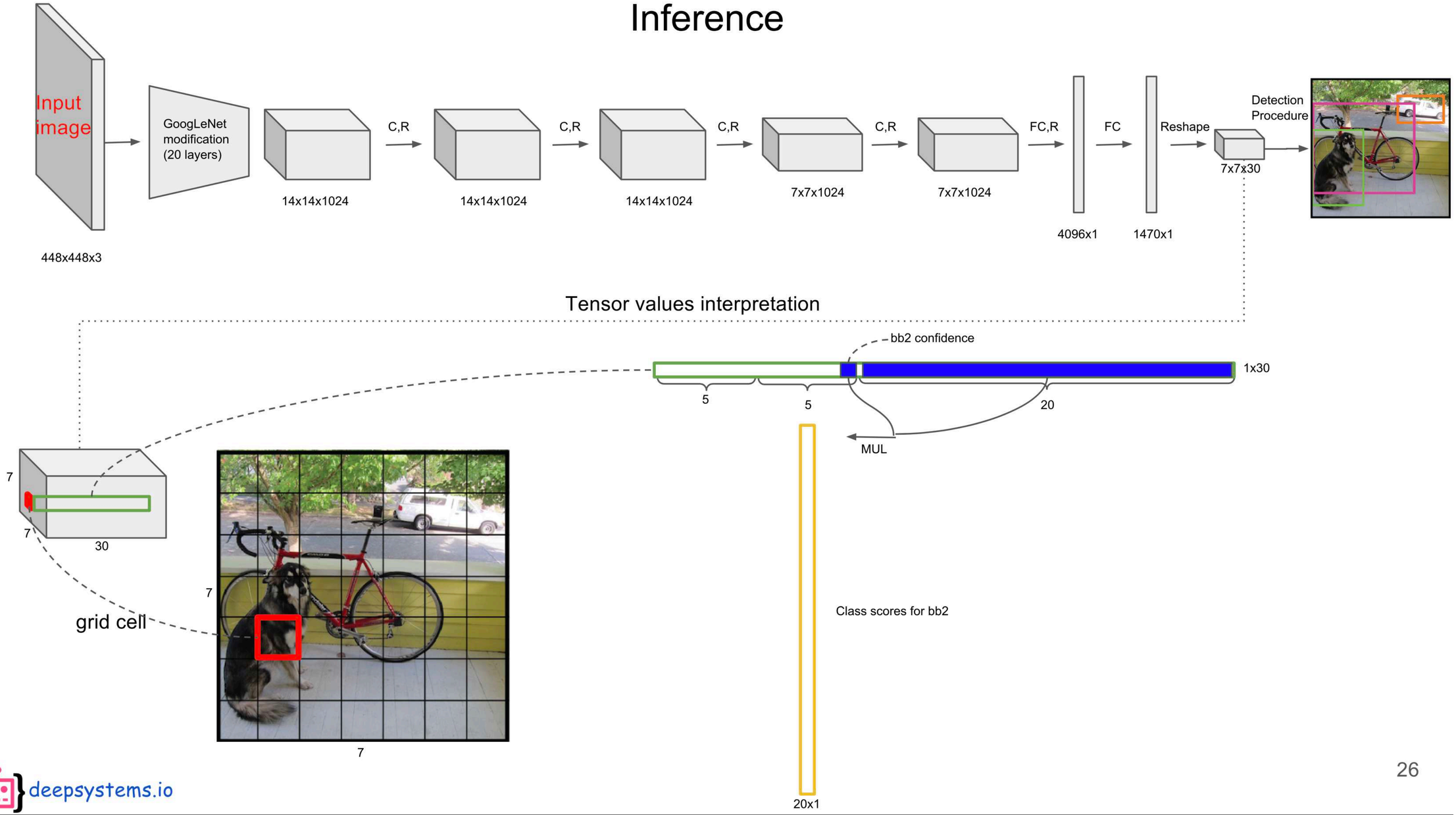
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
i (from 1 to 20)
1x30
5
5
20 - number of classes
Class score ~ P(obj is class_i | obj in box)
7
7
30
7
grid cell
7

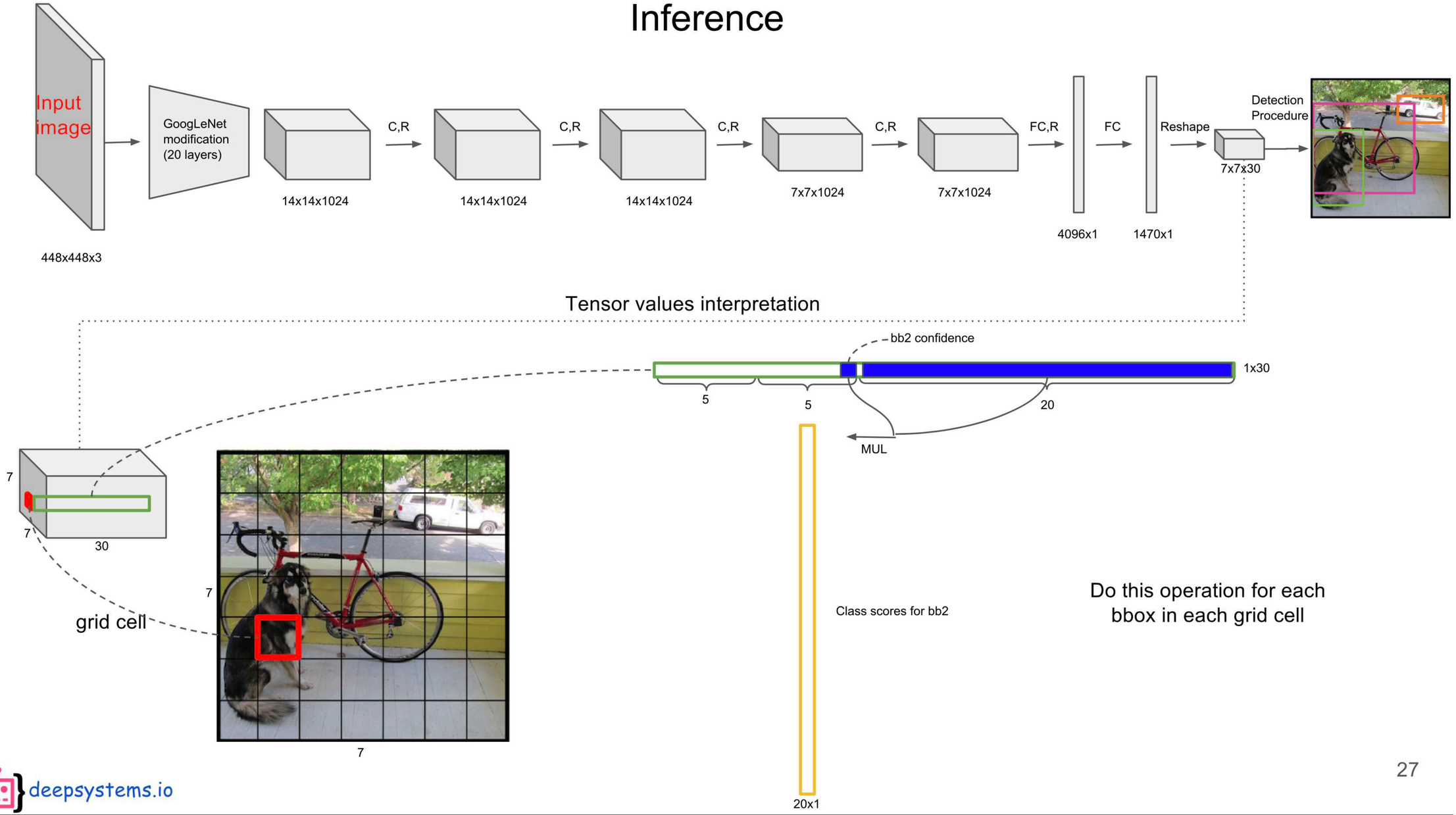
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
bb1 confidence
1x30
5
5
20
MUL
7
7
30
7
grid cell
7
Class scores for bb1
20x1

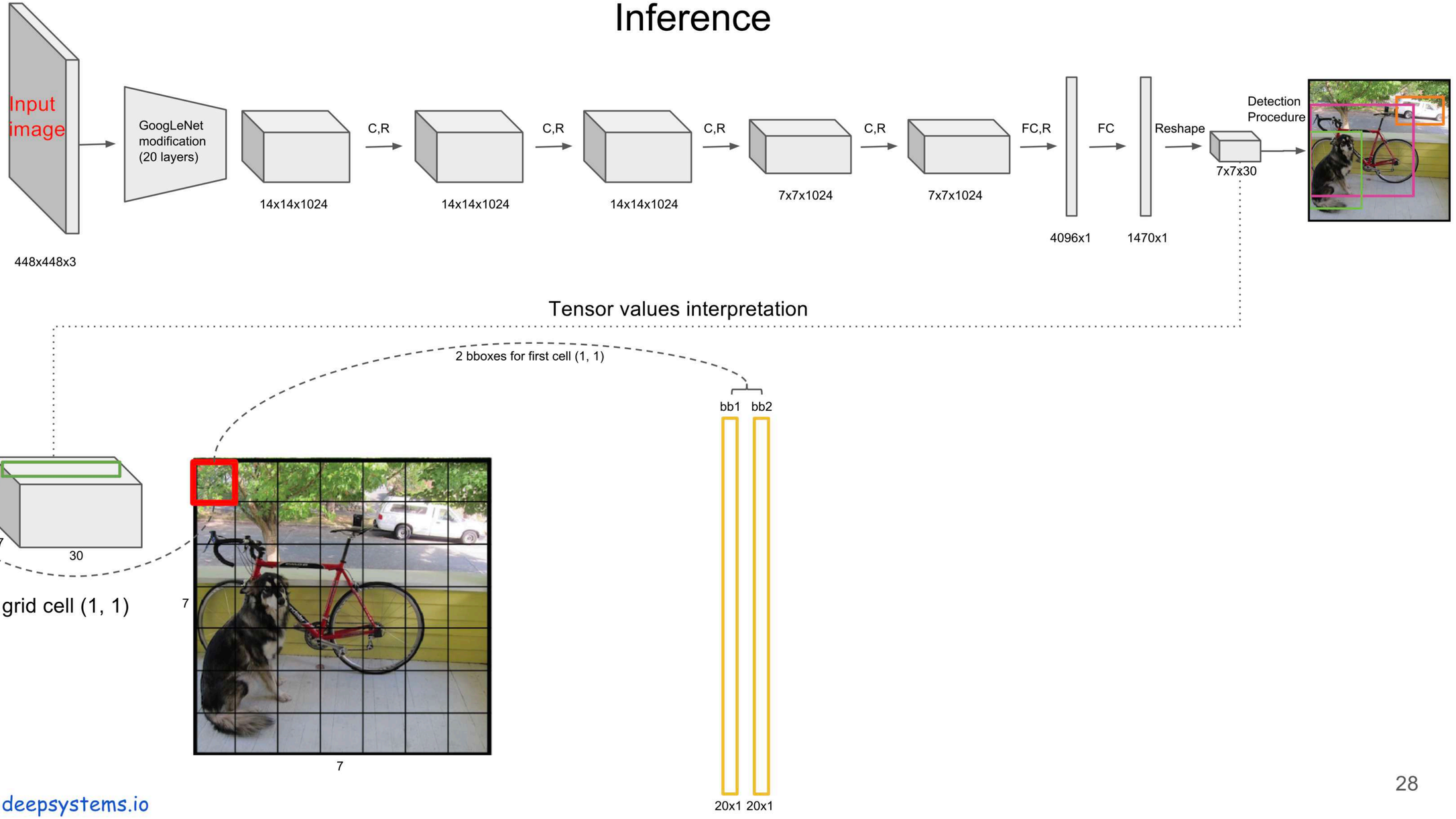
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
bb2 confidence
1x30
5
5
20
MUL
7
7
30
grid cell
7
7
Class scores for bb2
20x1

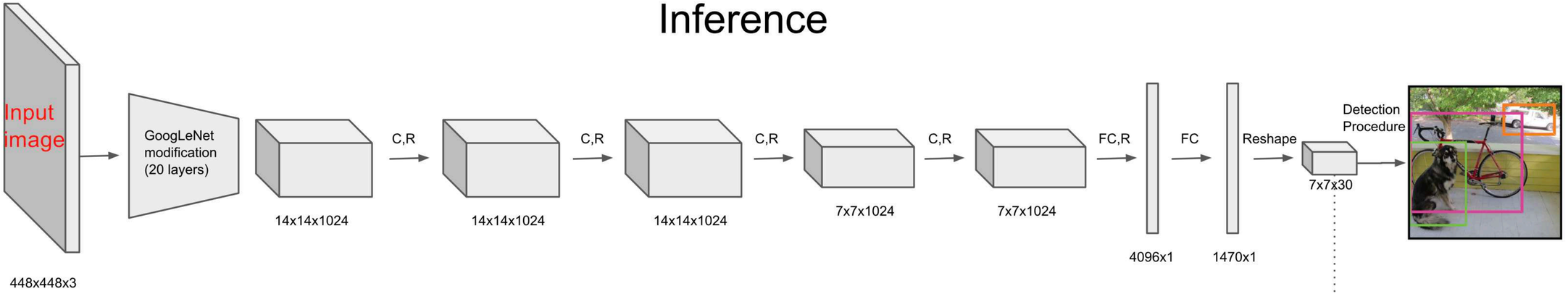
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
bb2 confidence
1x30
5
5
20
MUL
7
7
30
grid cell
7
7
Class scores for bb2
20x1
Do this operation for each bbox in each grid cell

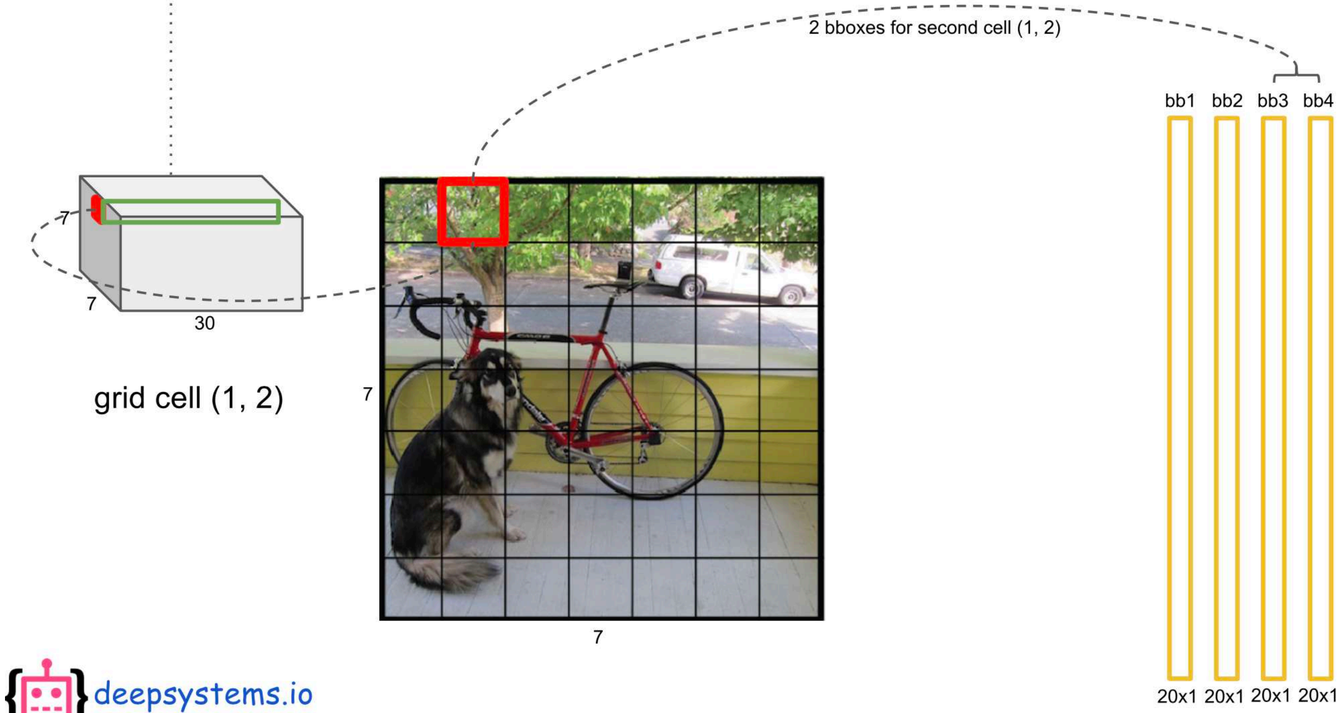
# Inference


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
2 bboxes for first cell (1, 1)
bb1
bb2
7
7
30
grid cell (1, 1)
7
7
20x1
20x1

# Inference

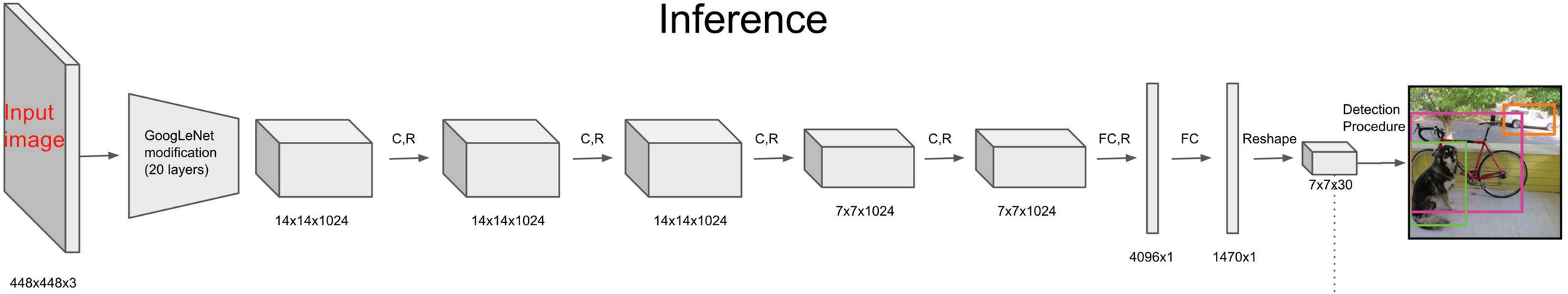



Tensor values interpretation

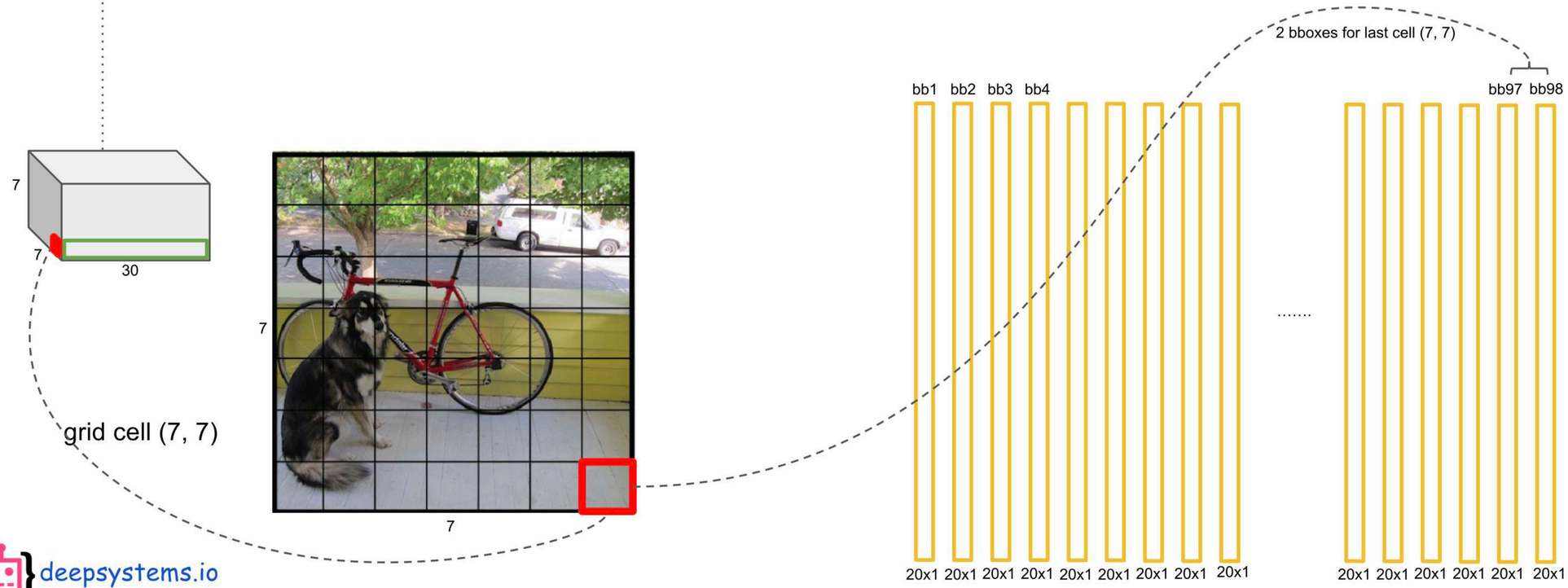



deepsystems.io

# Inference

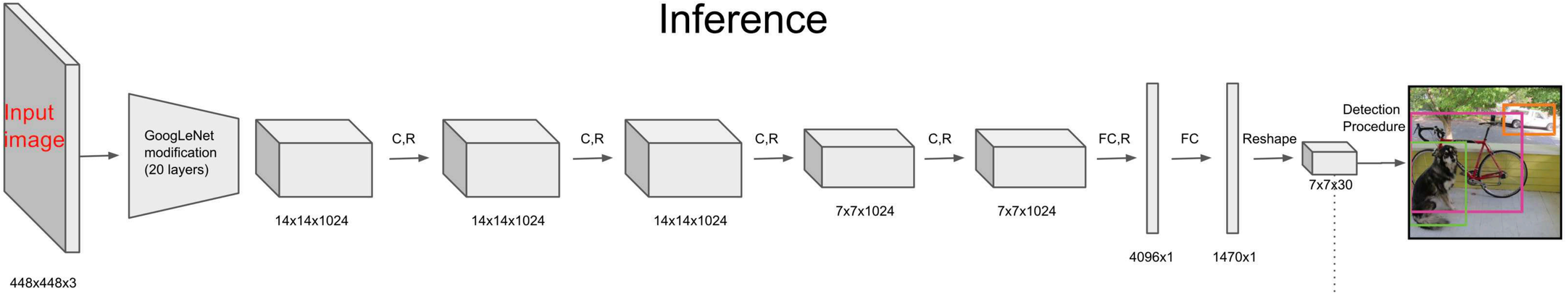



Tensor values interpretation

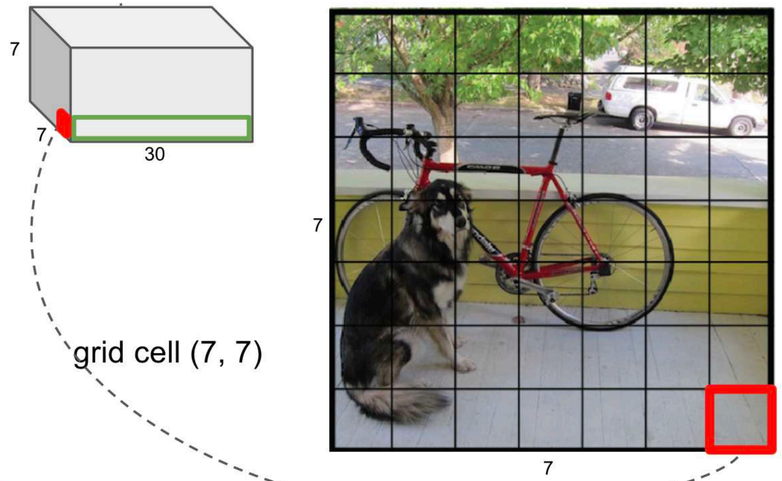



deepsystems.io

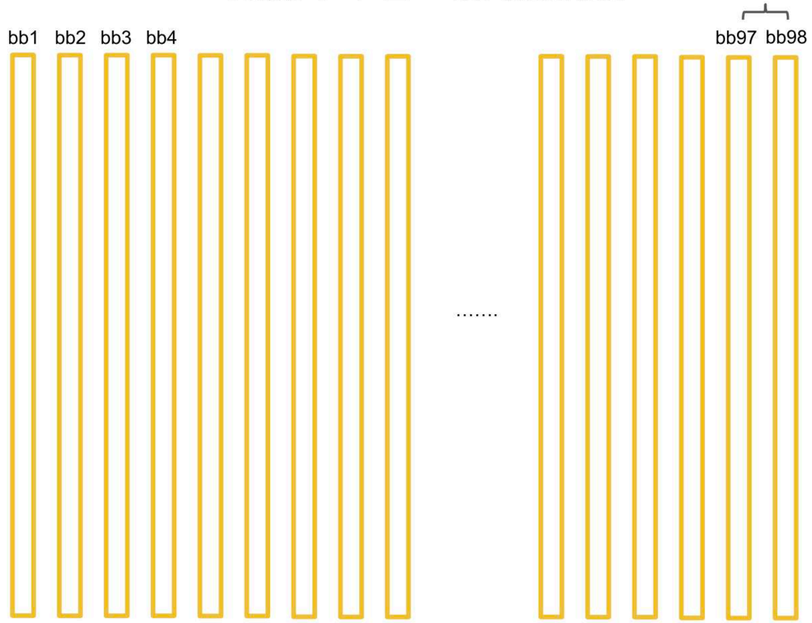
# Inference




Tensor values interpretation

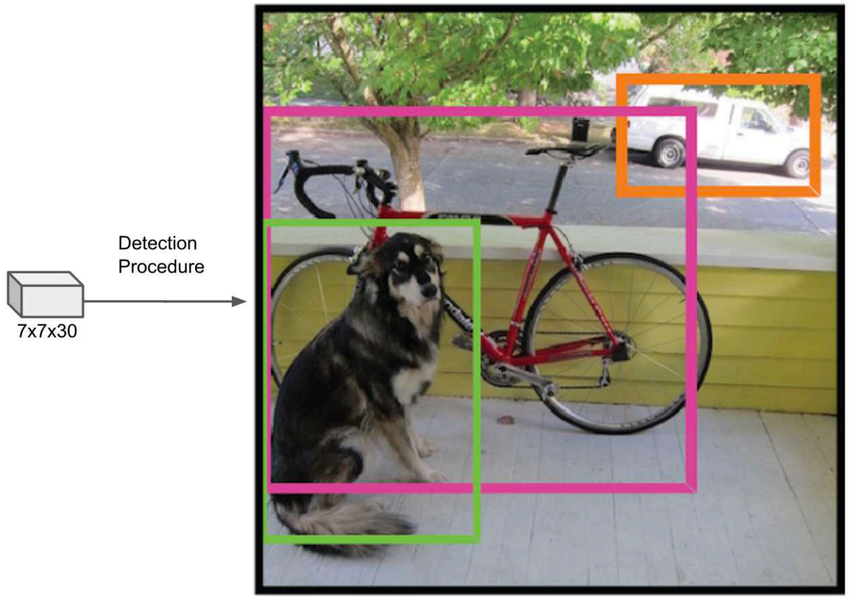


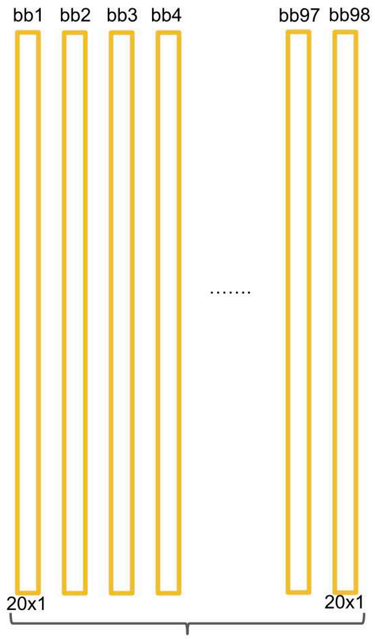

Total 7*7*2 = 98 bboxes

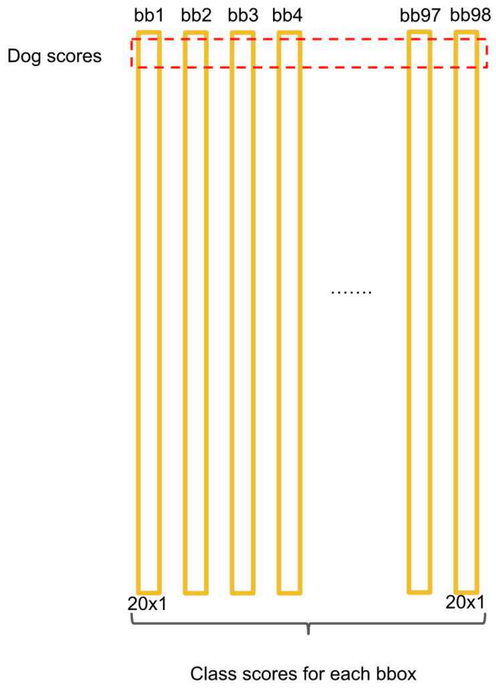



deepsystems.io

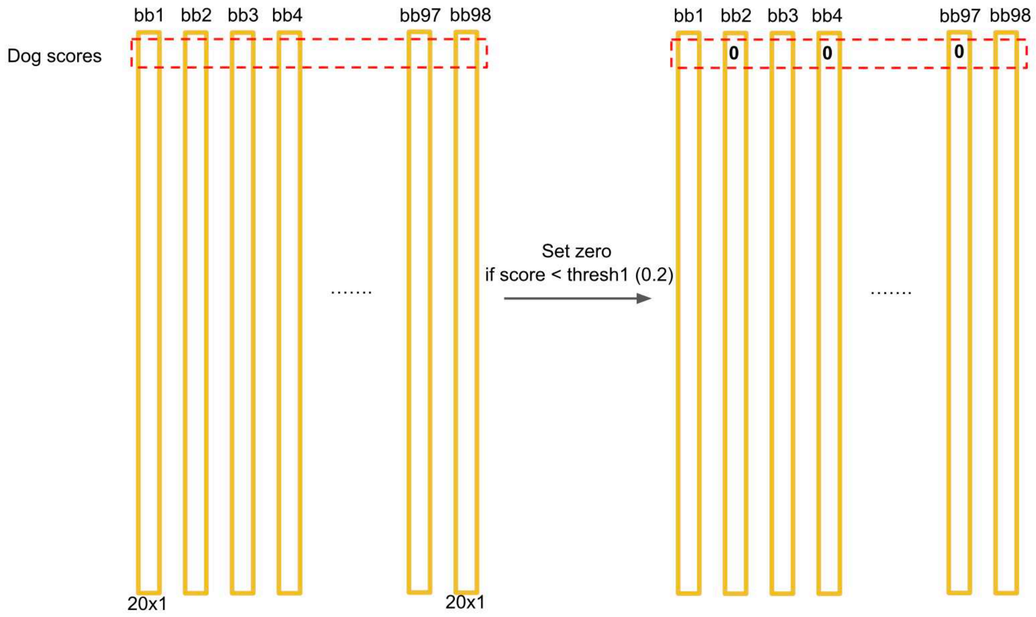
# Look at detection procedure

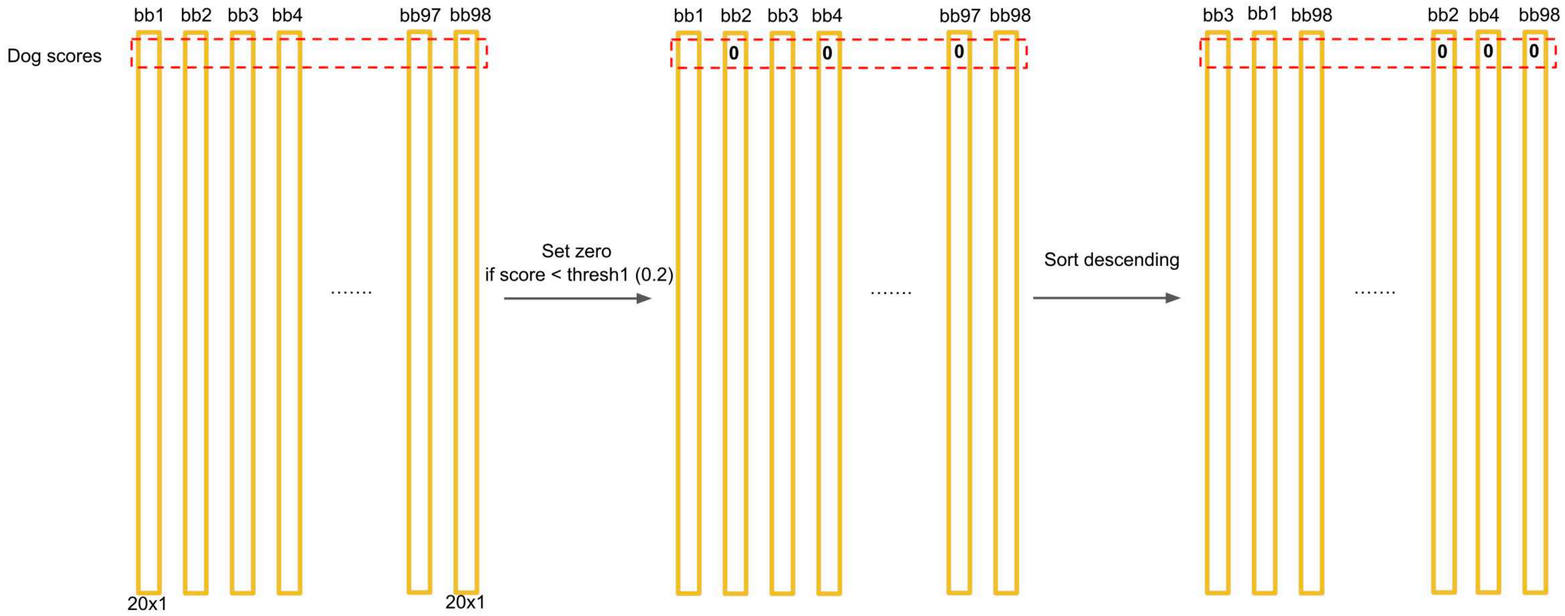


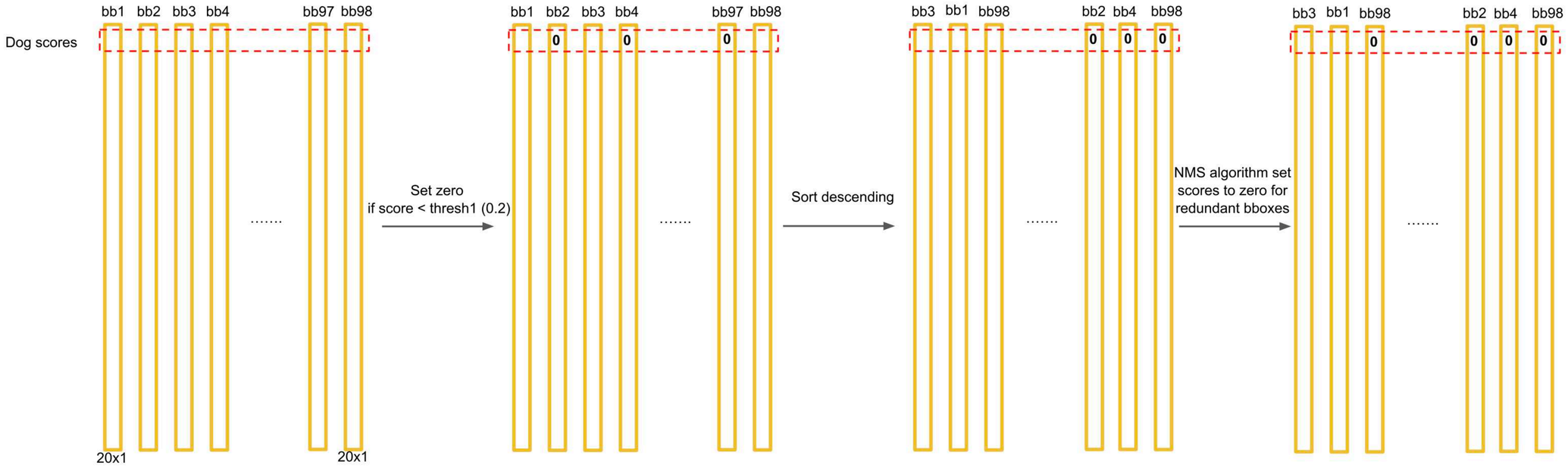

bb1
bb2
bb3
bb4
.....
bb97
bb98
20x1
20x1
Class scores for each bbox

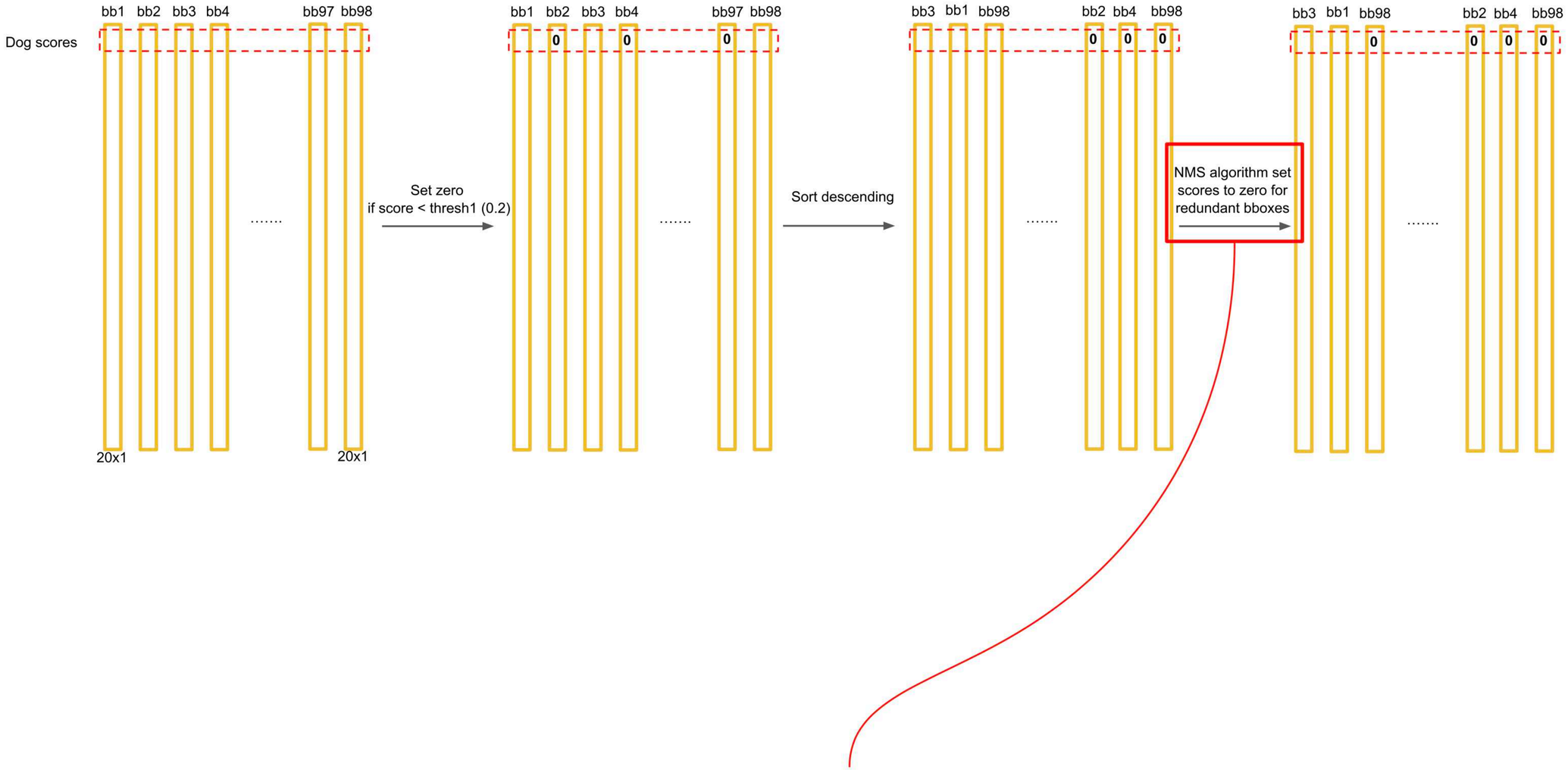



Get first class scores for each bbox


Dog scores
bb1
bb2
bb3
bb4
bb97
bb98
.....
20x1
20x1
Set zero
if score < thresh1 (0.2)
bb1
bb2
bb3
bb4
bb97
bb98
0
0
0
.....


Dog scores
bb1 bb2 bb3 bb4 ..... bb97 bb98
20x1
20x1
Set zero
if score < thresh1 (0.2)
bb1 bb2 bb3 bb4 ..... bb97 bb98
0 0 0
Sort descending
bb3 bb1 bb98 ..... bb2 bb4 bb98
0 0 0

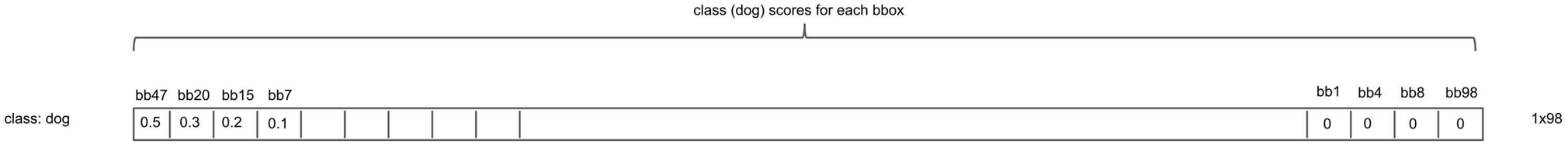

Dog scores
bb1 bb2 bb3 bb4 ..... bb97 bb98
20x1
20x1
Set zero
if score < thresh1 (0.2)
bb1 bb2 bb3 bb4 ..... bb97 bb98
0 0 0
Sort descending
bb3 bb1 bb98 ..... bb2 bb4 bb98
0 0 0
NMS algorithm set
scores to zero for
redundant bboxes
bb3 bb1 bb98 ..... bb2 bb4 bb98
0 0 0


Dog scores
bb1 bb2 bb3 bb4 ..... bb97 bb98
20x1
20x1
Set zero
if score < thresh1 (0.2)
bb1 bb2 bb3 bb4 ..... bb97 bb98
0 0 0
Sort descending
bb3 bb1 bb98 ..... bb2 bb4 bb98
0 0 0
NMS algorithm set
scores to zero for
redundant bboxes
bb3 bb1 bb98 ..... bb2 bb4 bb98
0 0 0

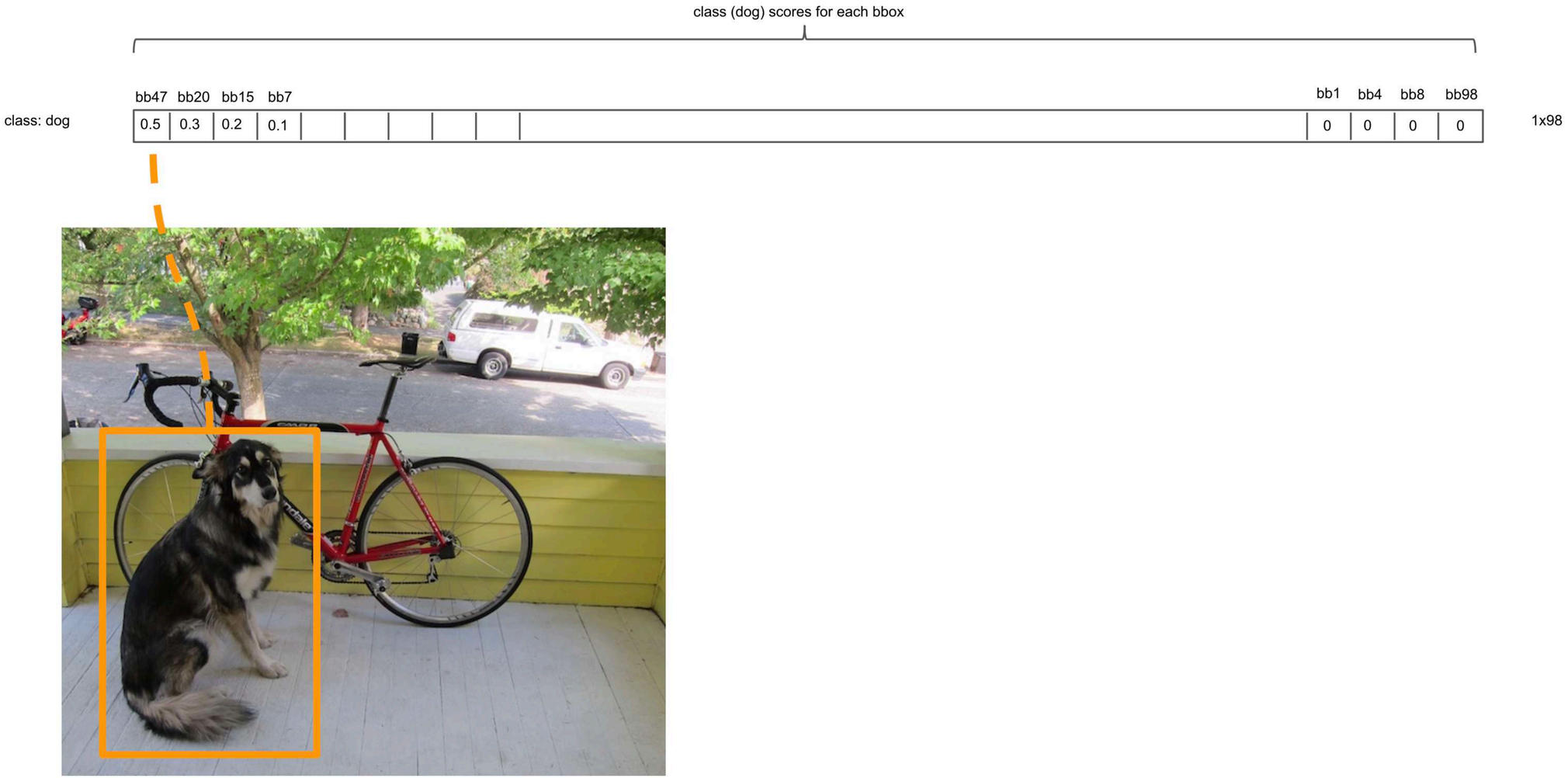
How it works

# Non-Maximum Suppression: intuition

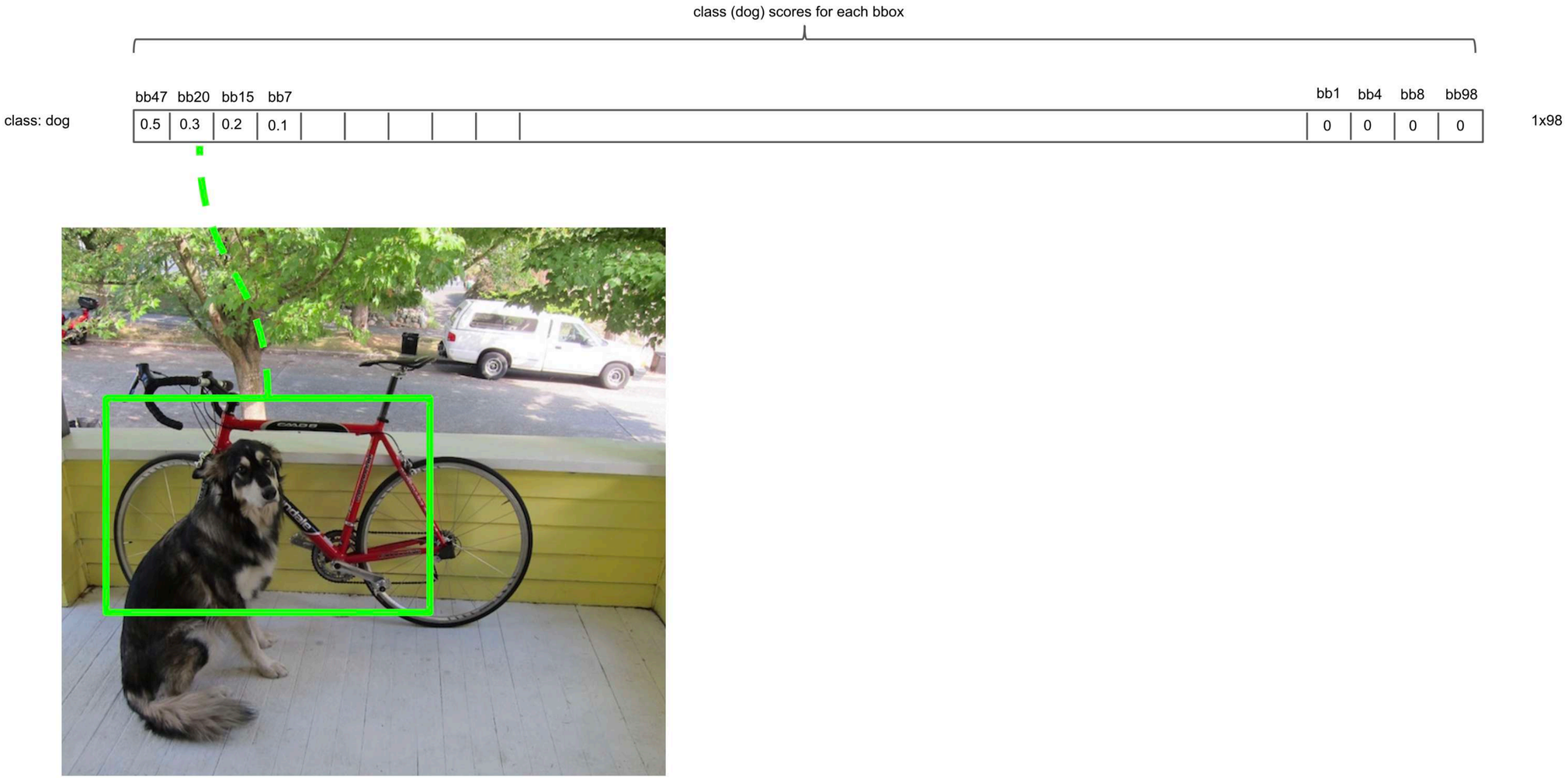
# Non-Maximum Suppression: intuition



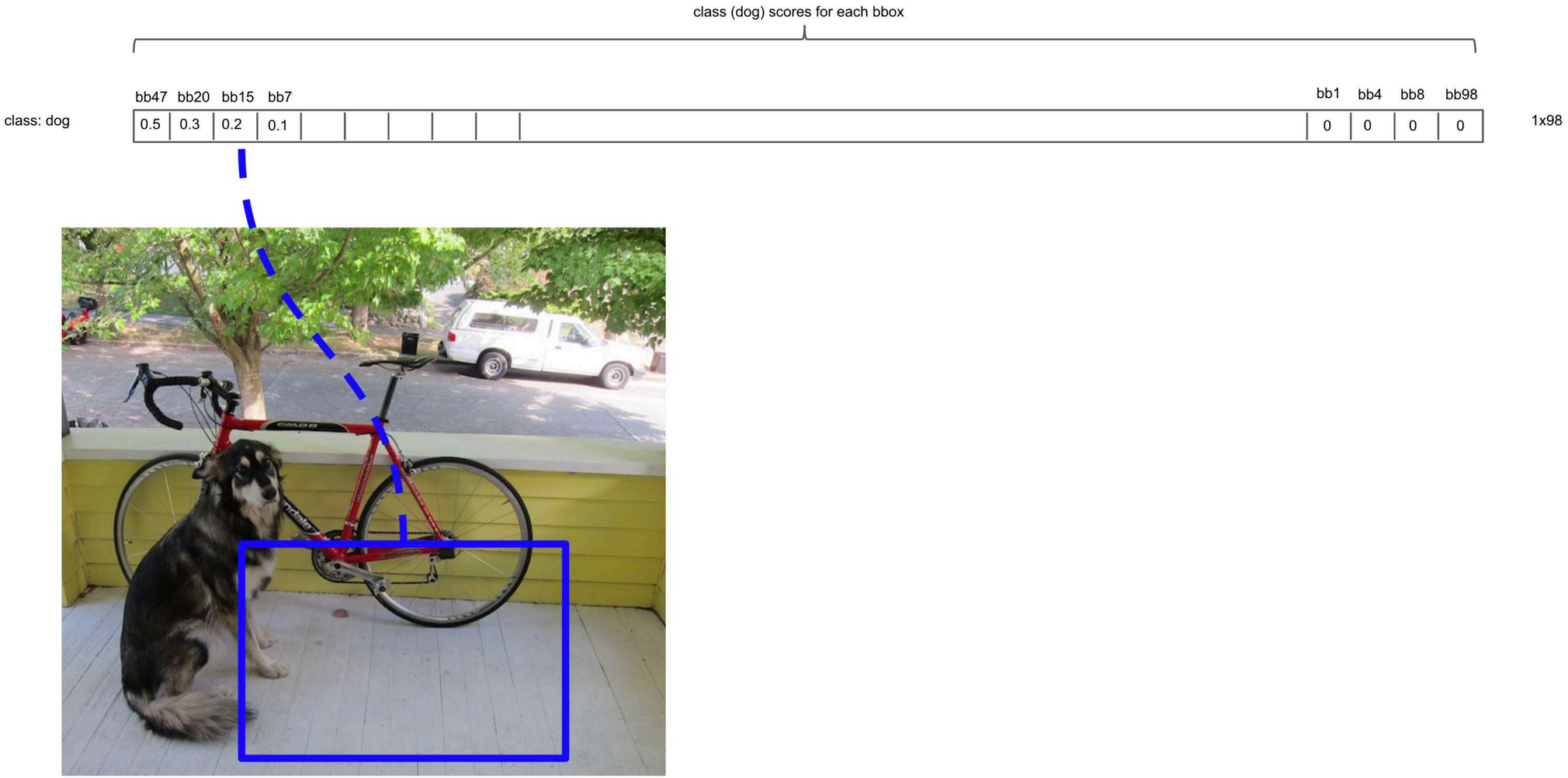
# Non-Maximum Suppression: intuition



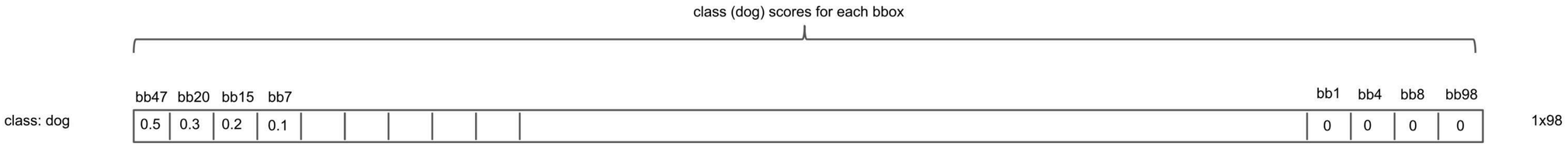
# Non-Maximum Suppression: intuition

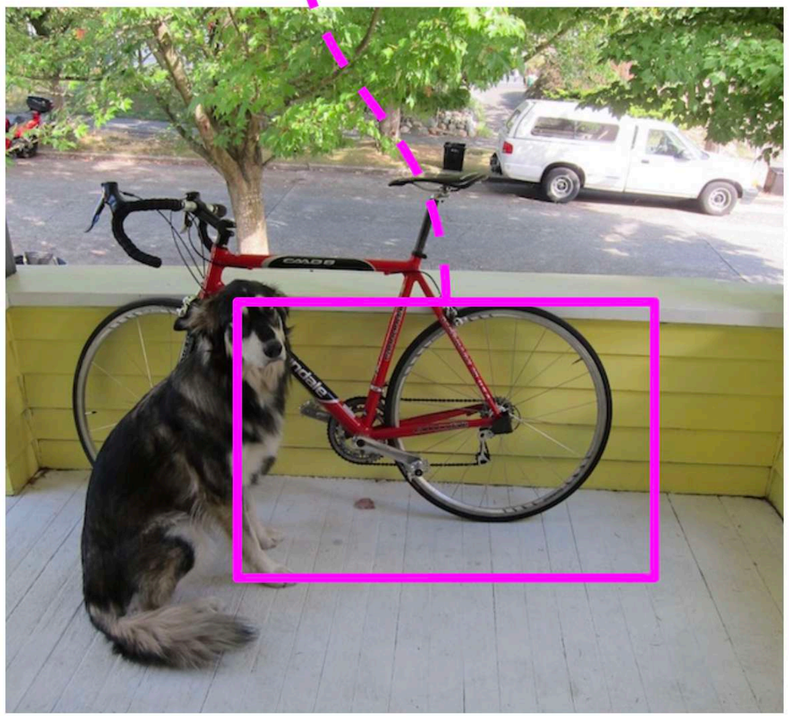


# Non-Maximum Suppression: intuition

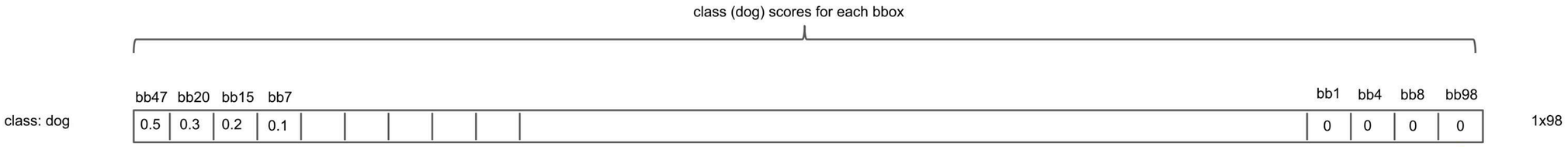


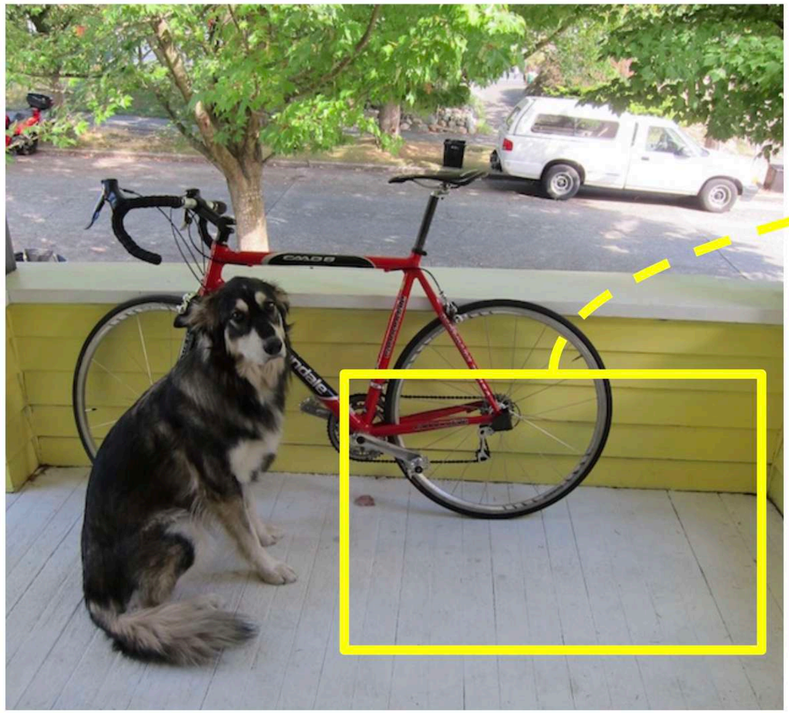
# Non-Maximum Suppression: intuition



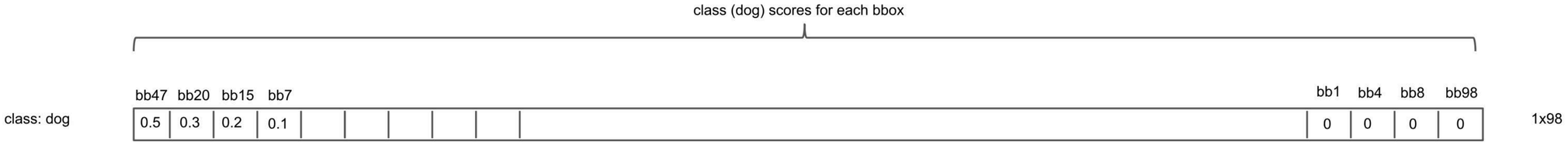



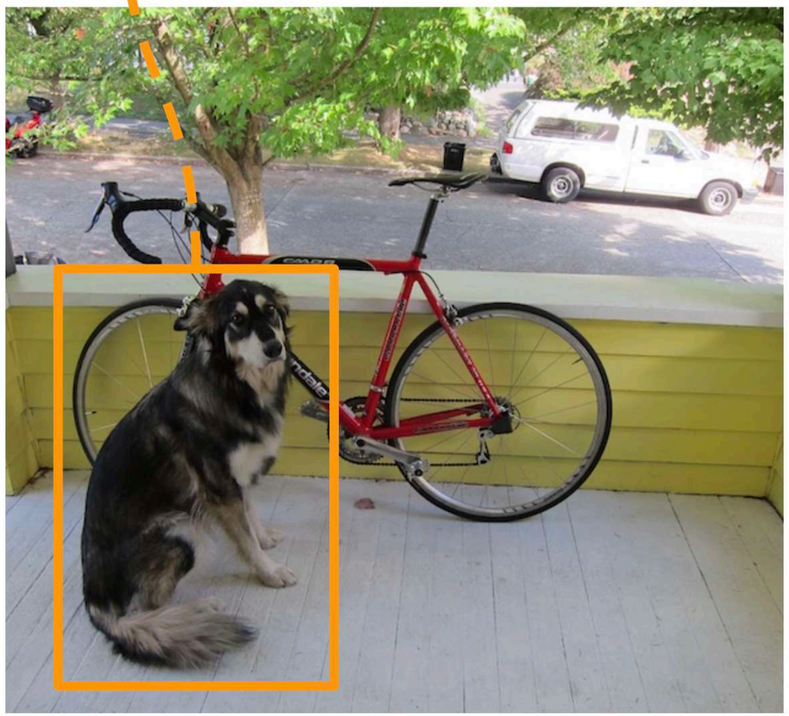
# Non-Maximum Suppression: intuition



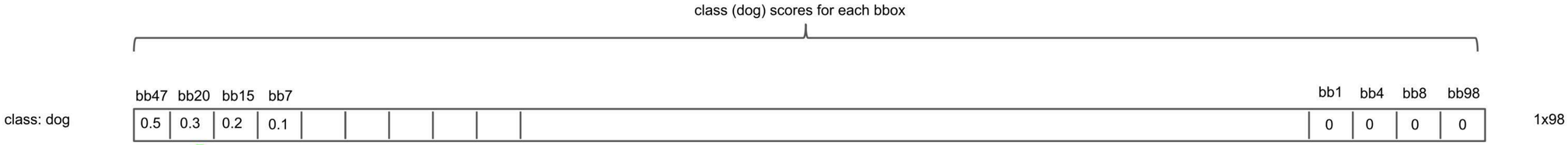
# Non-Maximum Suppression: intuition

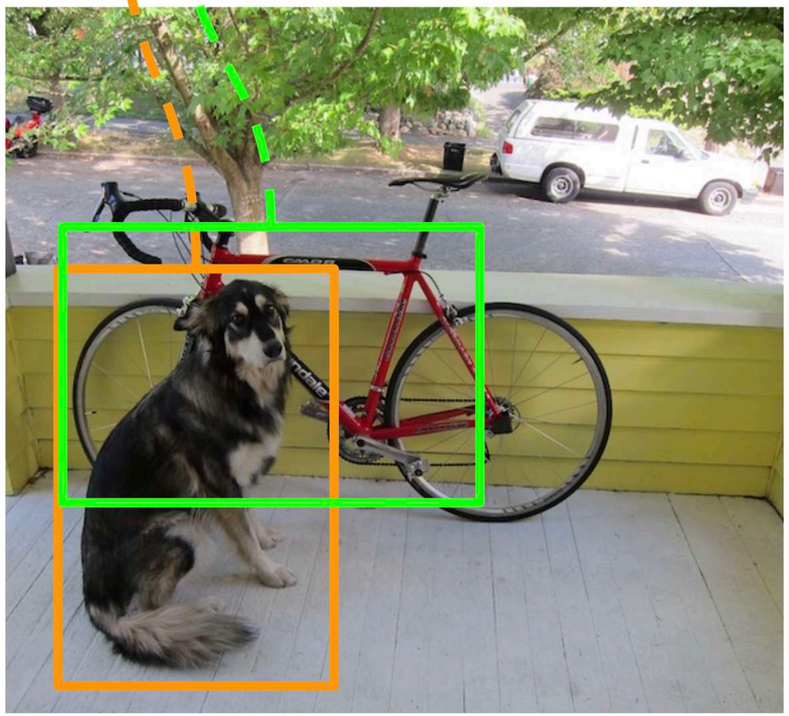


Get ~~bbox with max~~ score. Let's denote it "bbox_max"

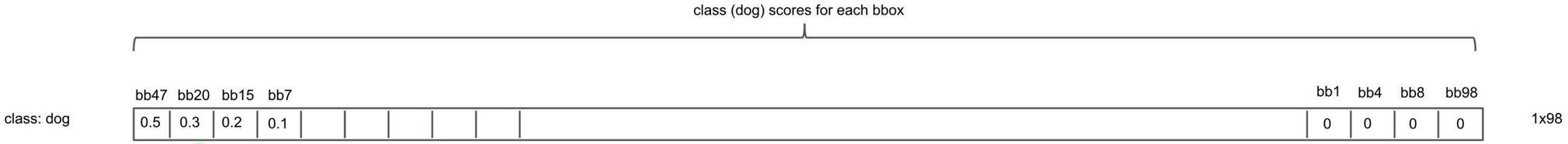
# Non-Maximum Suppression: intuition

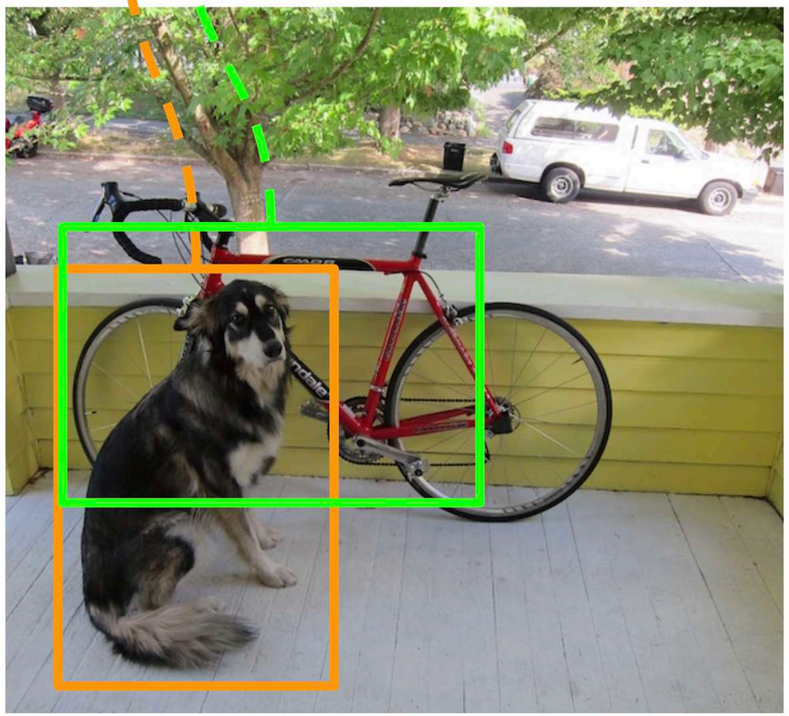


Compare “bbox_max” with others less score (non-zero!) bboxes. Let’s denote it “bbox_cur”

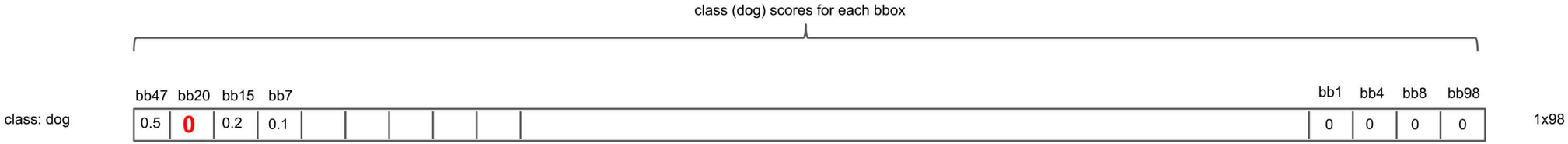
# Non-Maximum Suppression: intuition

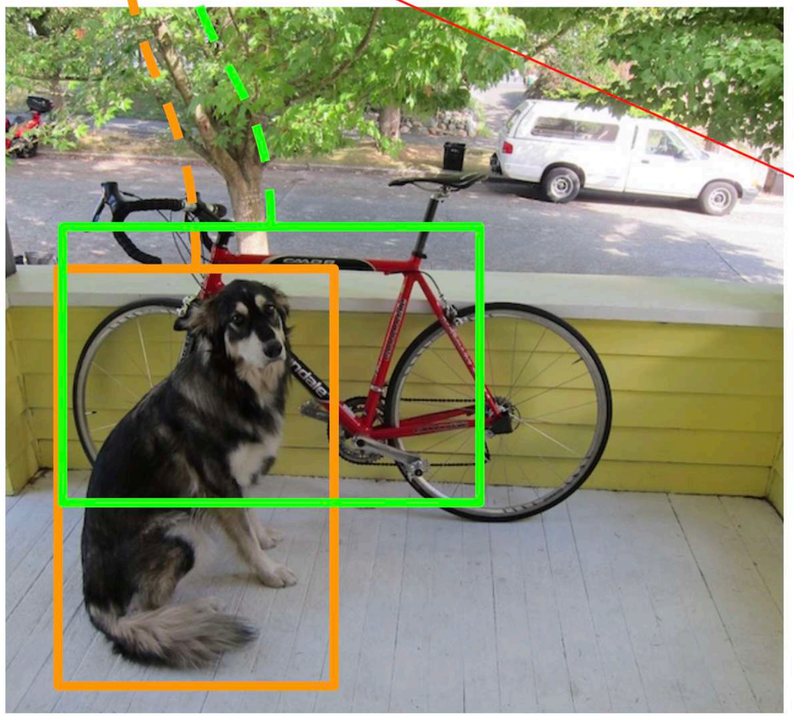


If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

# Non-Maximum Suppression: intuition

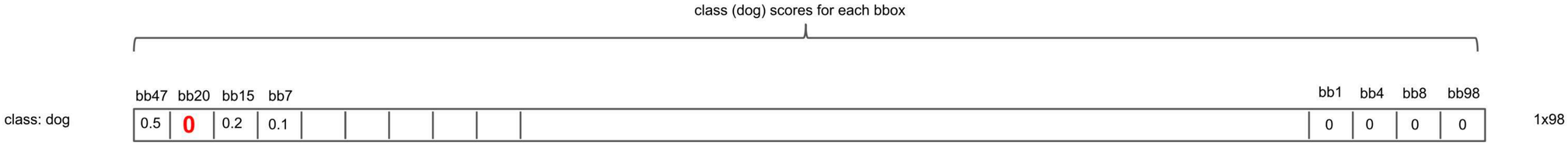


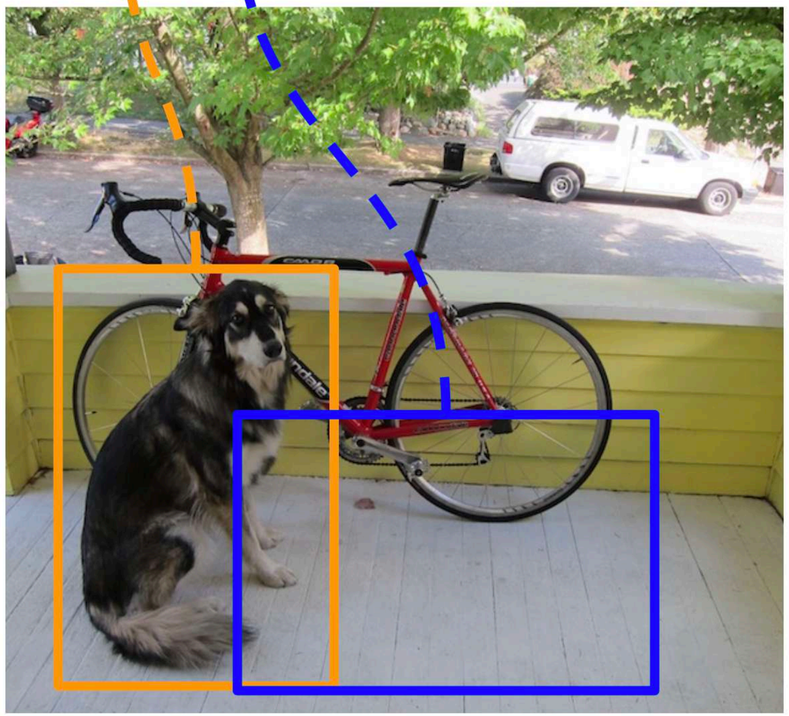



If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: set to 0.

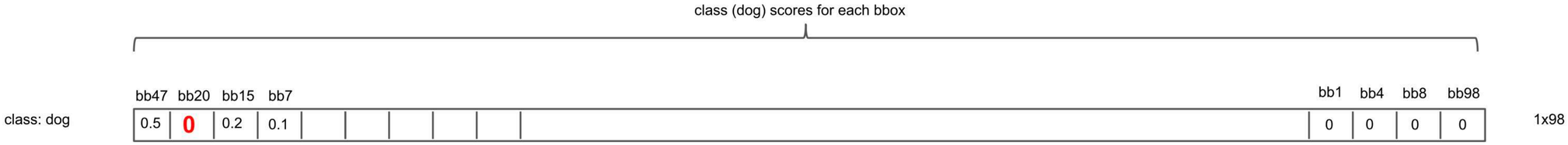
# Non-Maximum Suppression: intuition

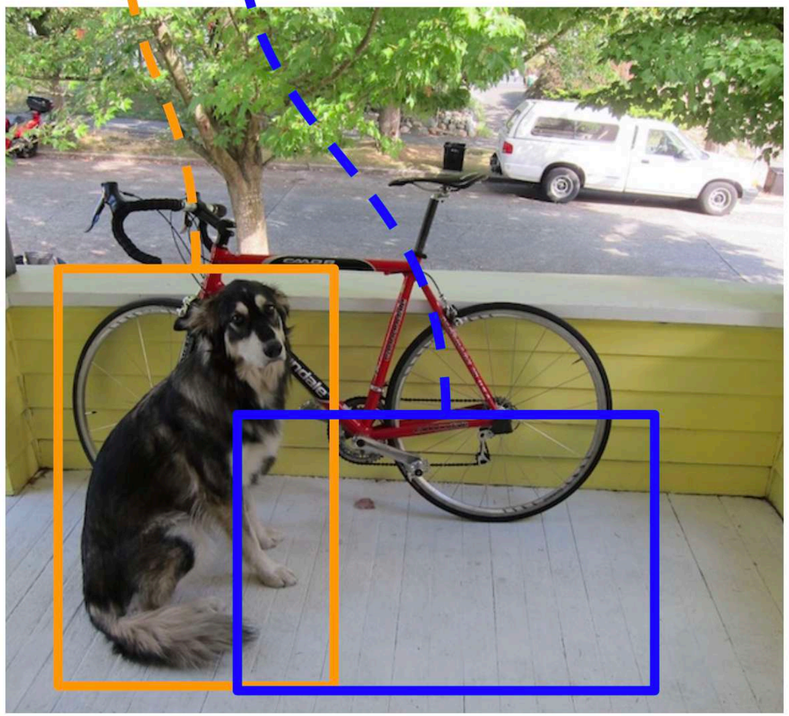


Go to next bbox_cur.

# Non-Maximum Suppression: intuition

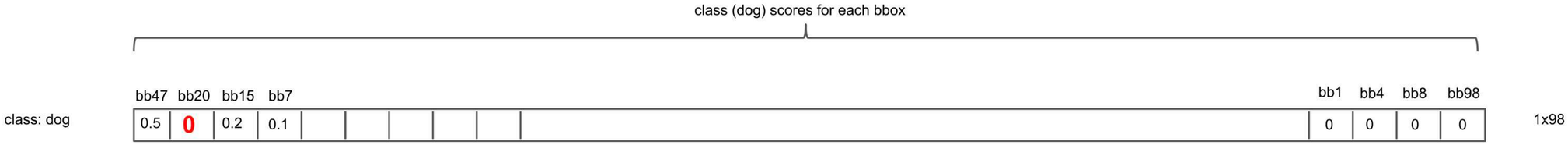


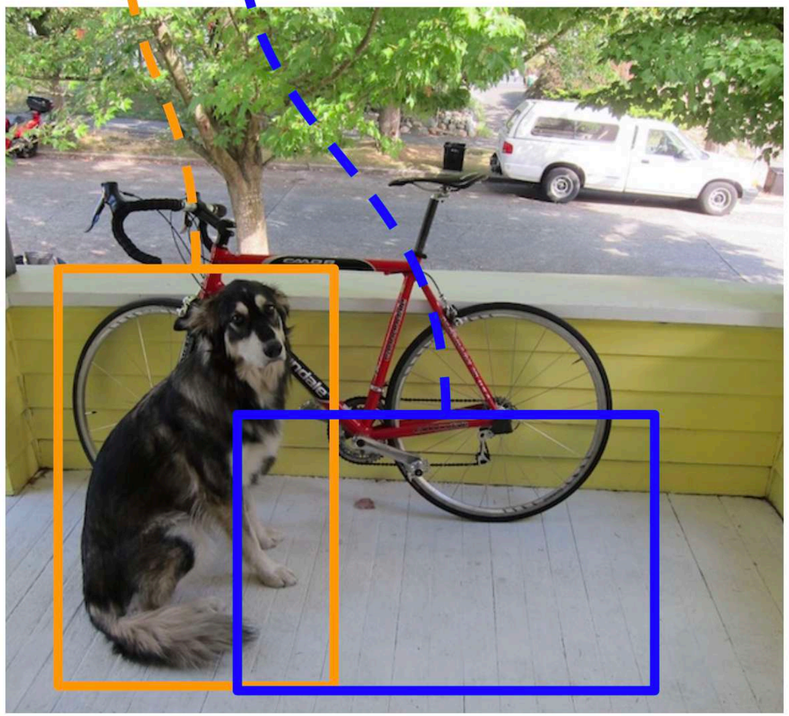



Go to next bbox_cur.

If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

# Non-Maximum Suppression: intuition



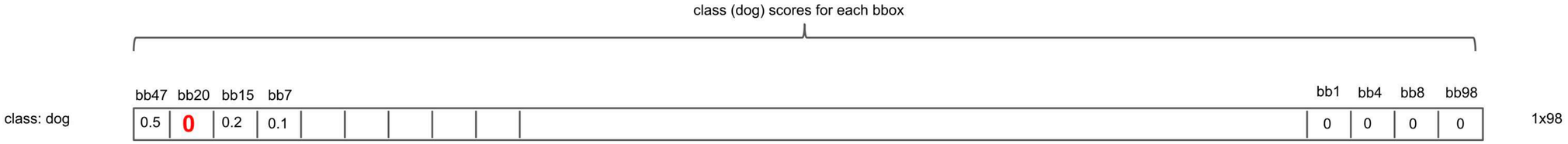



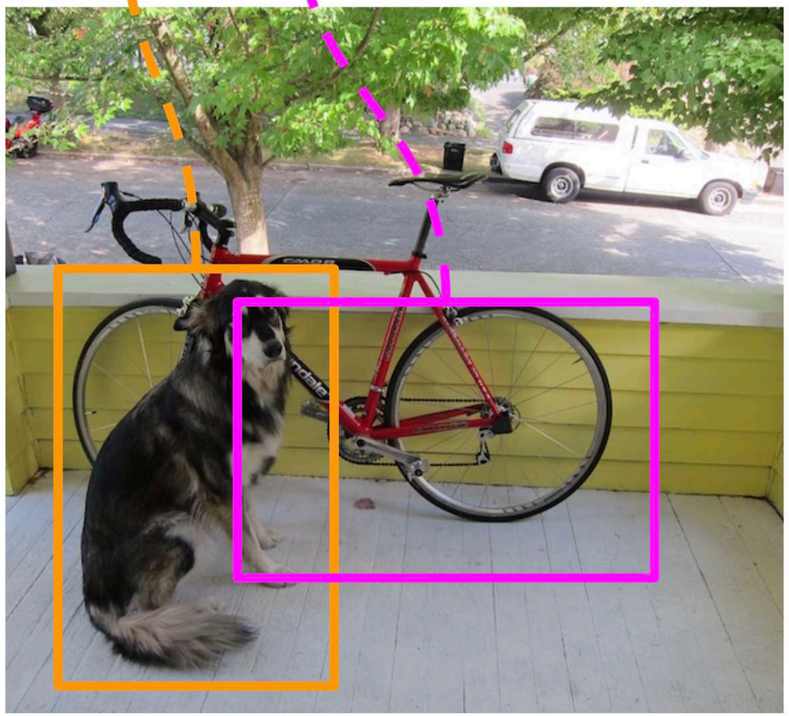
Go to next bbox_cur.

If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: continue.

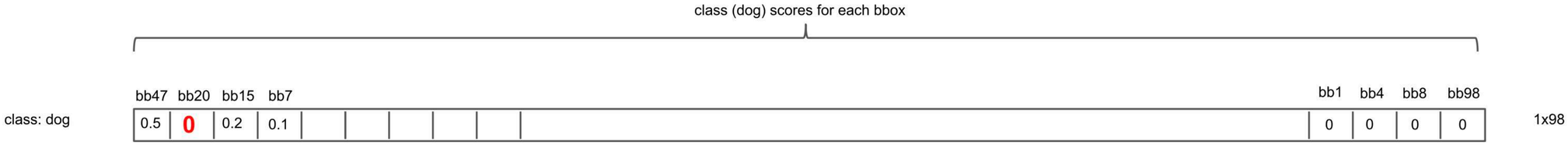
# Non-Maximum Suppression: intuition

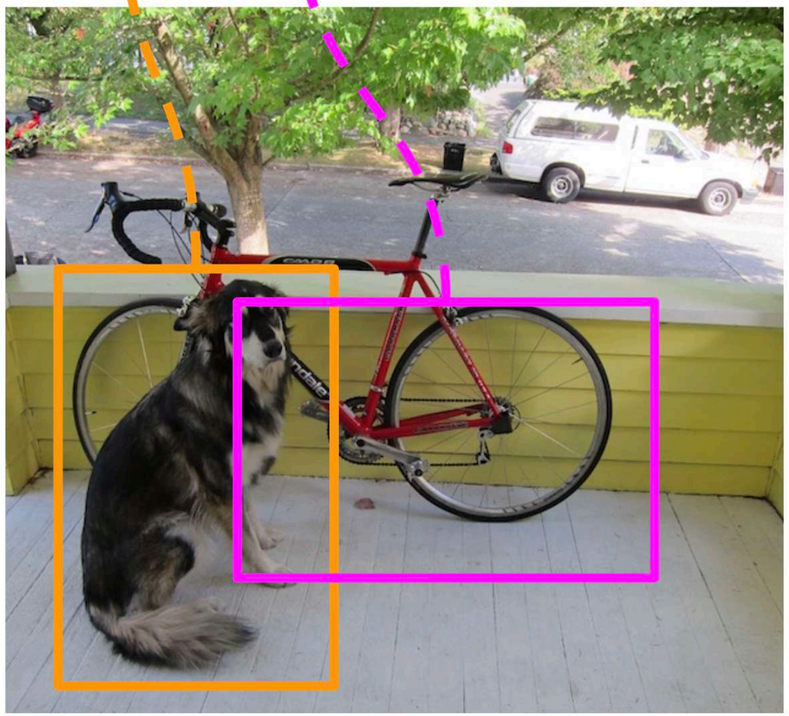


Go to next bbox_cur.

# Non-Maximum Suppression: intuition

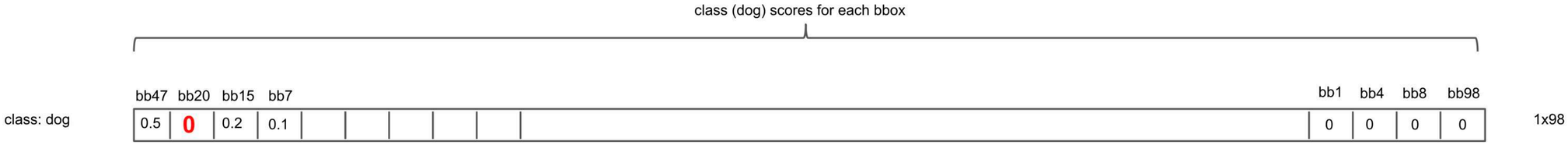


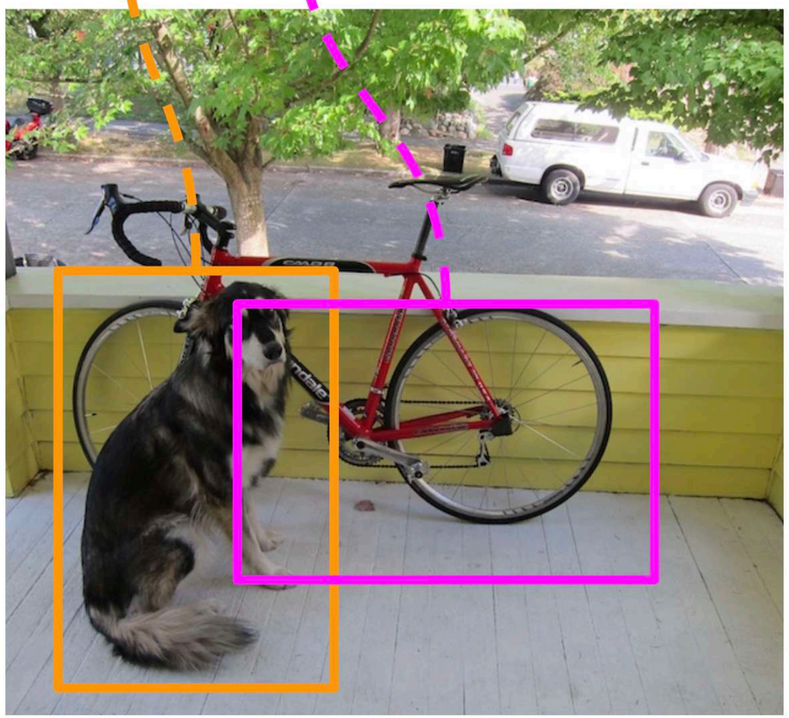



Go to next bbox_cur.

If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

# Non-Maximum Suppression: intuition



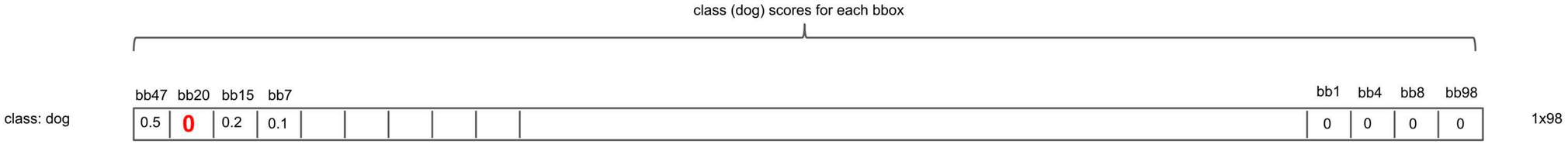



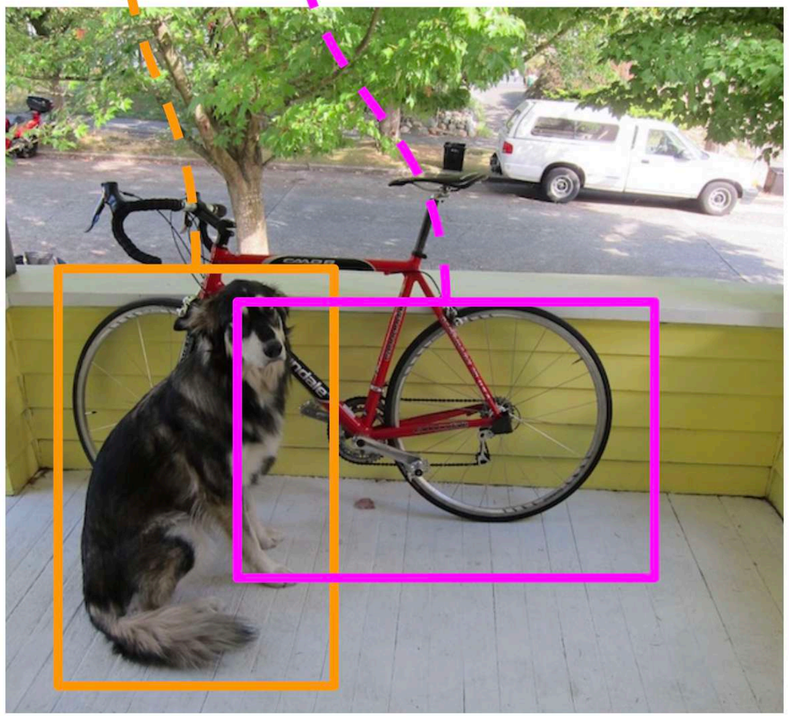
Go to next bbox_cur.

If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: continue.

# Non-Maximum Suppression: intuition



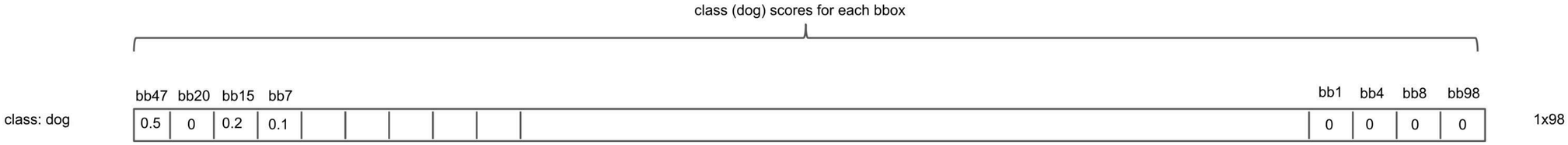
Go to next bbox_cur.

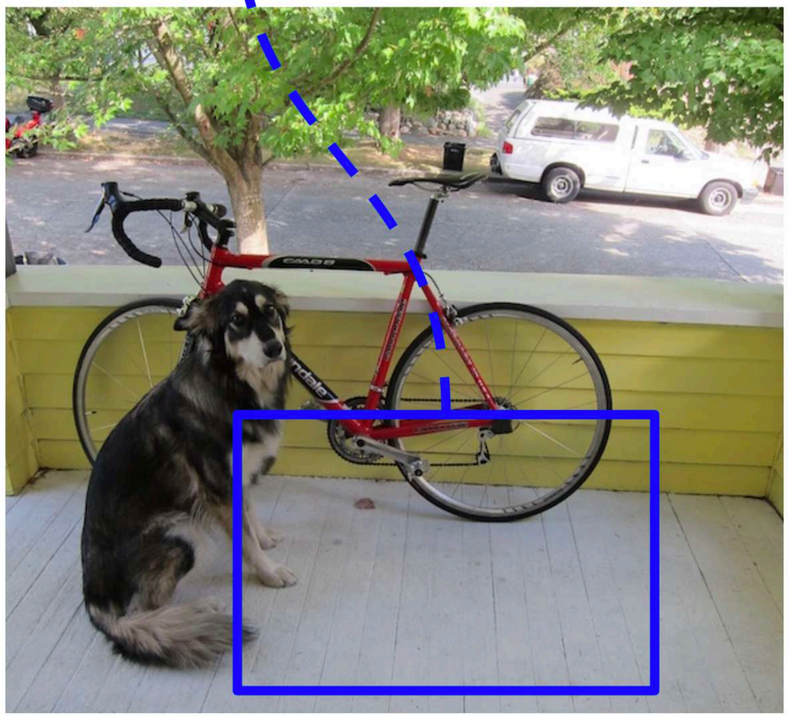
If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: continue.

Do this procedure for other “bbox_cur”. After that ...

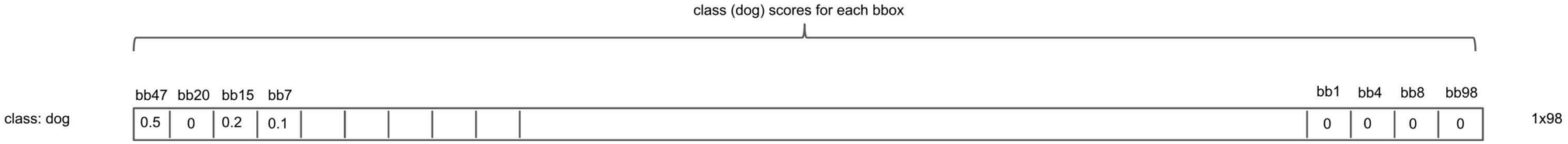
# Non-Maximum Suppression: intuition

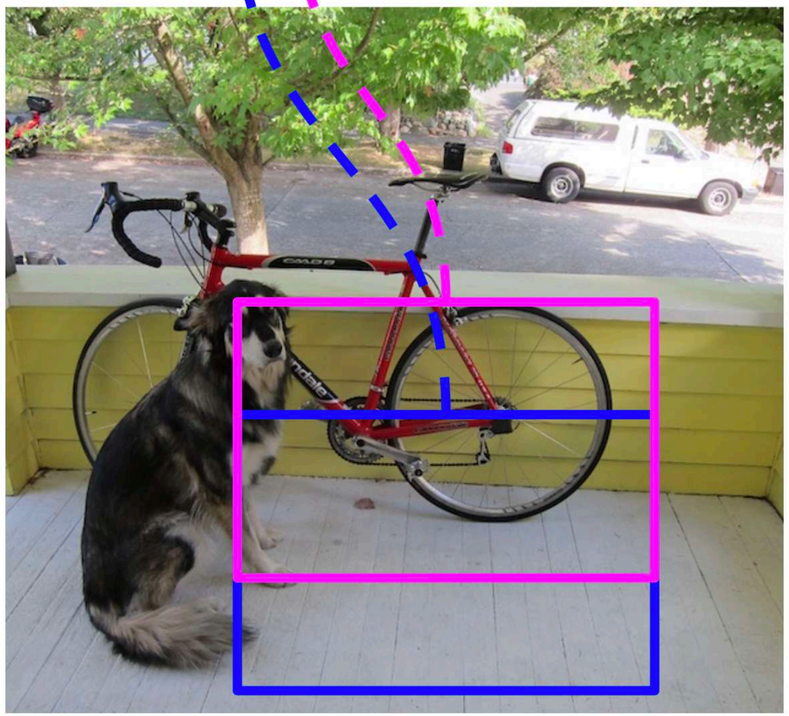


Go to next bbox with big score.
Let's denote it "bbox_max"

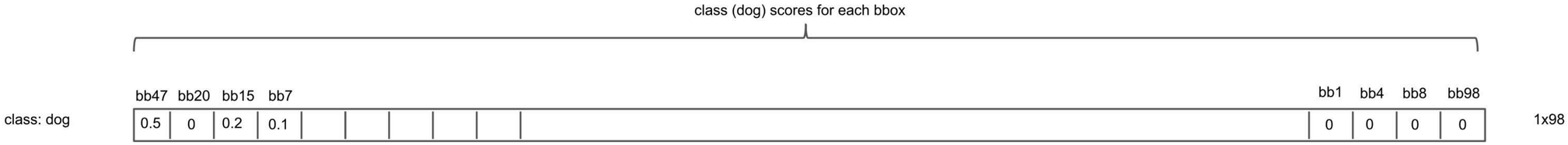
# Non-Maximum Suppression: intuition

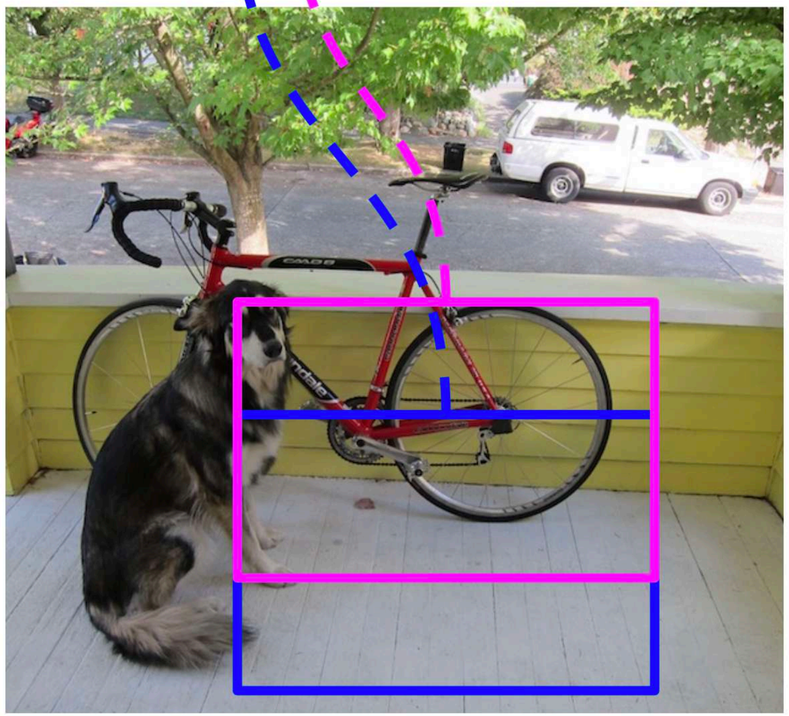


Go to next bbox_cur.

# Non-Maximum Suppression: intuition

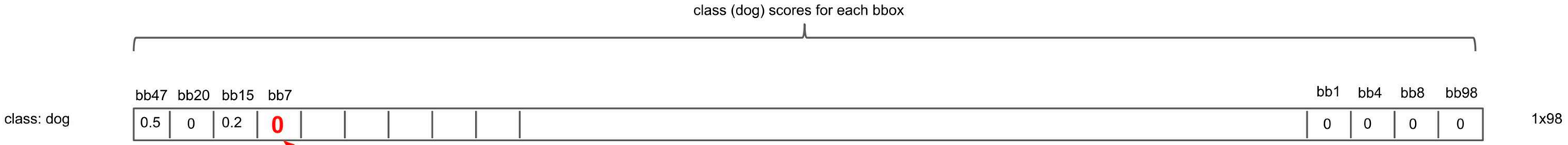


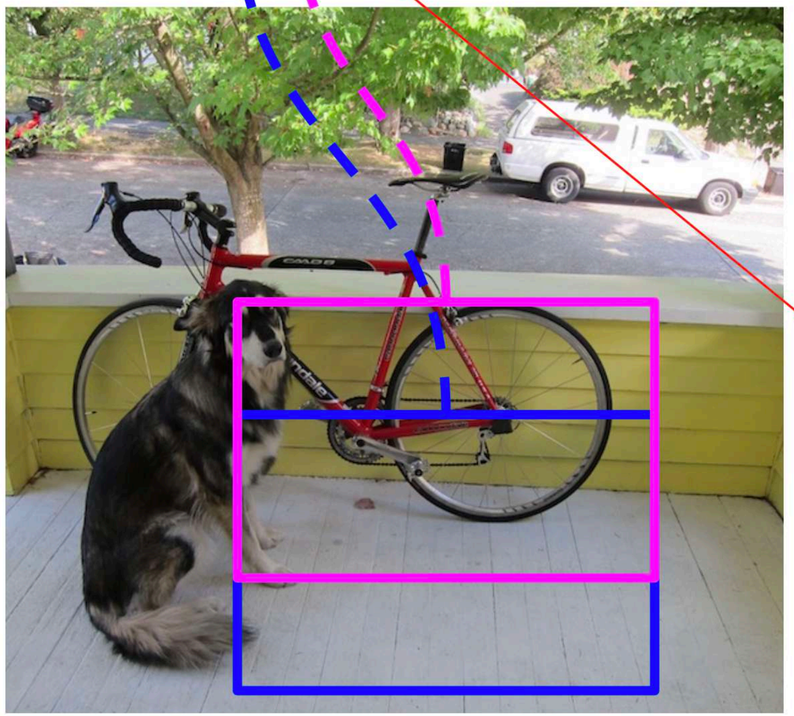



Go to next bbox_cur.

If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

# Non-Maximum Suppression: intuition



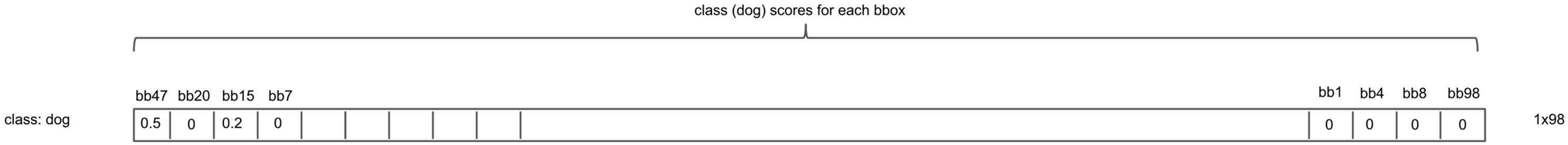
Go to next bbox_cur.

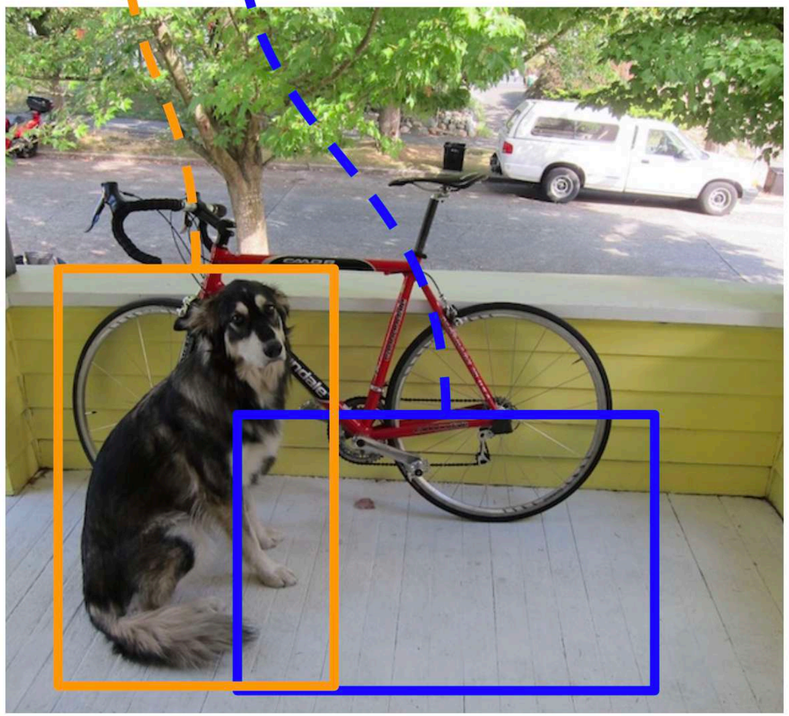
If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: set to 0.

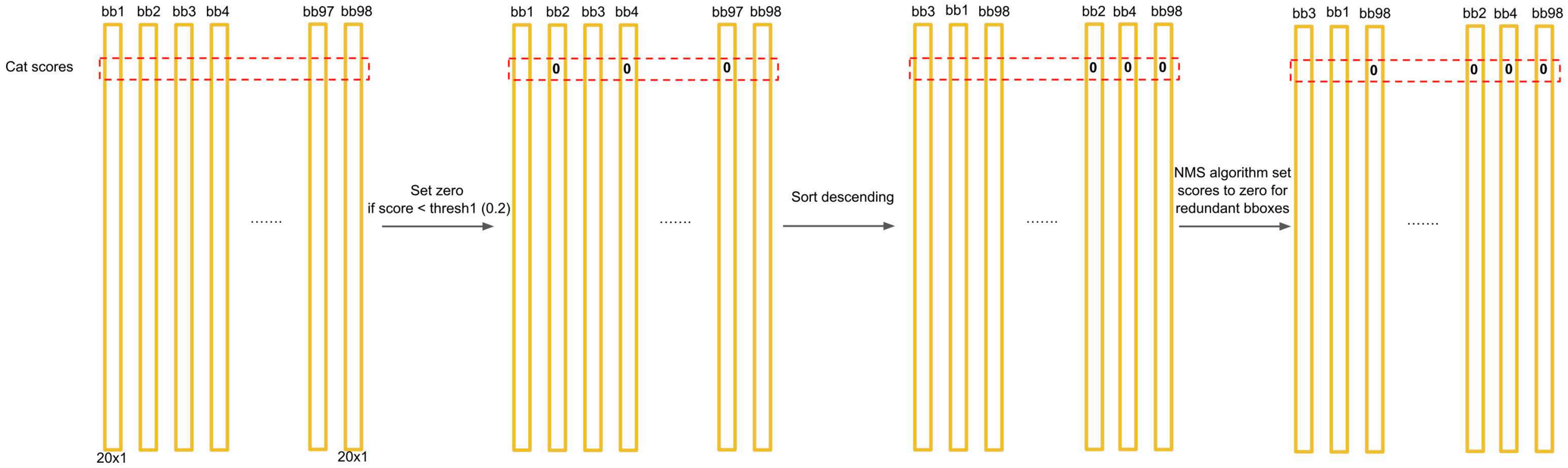
Do this procedure for other “bbox_max” and for other corresponding “bbox_cur”.

# Non-Maximum Suppression: intuition

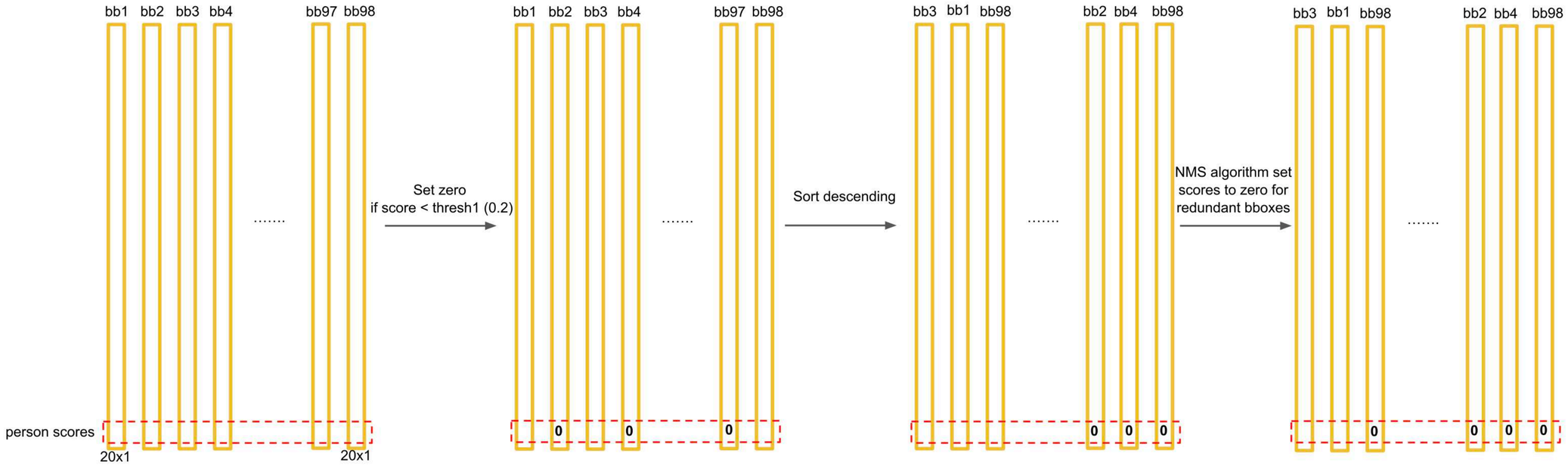


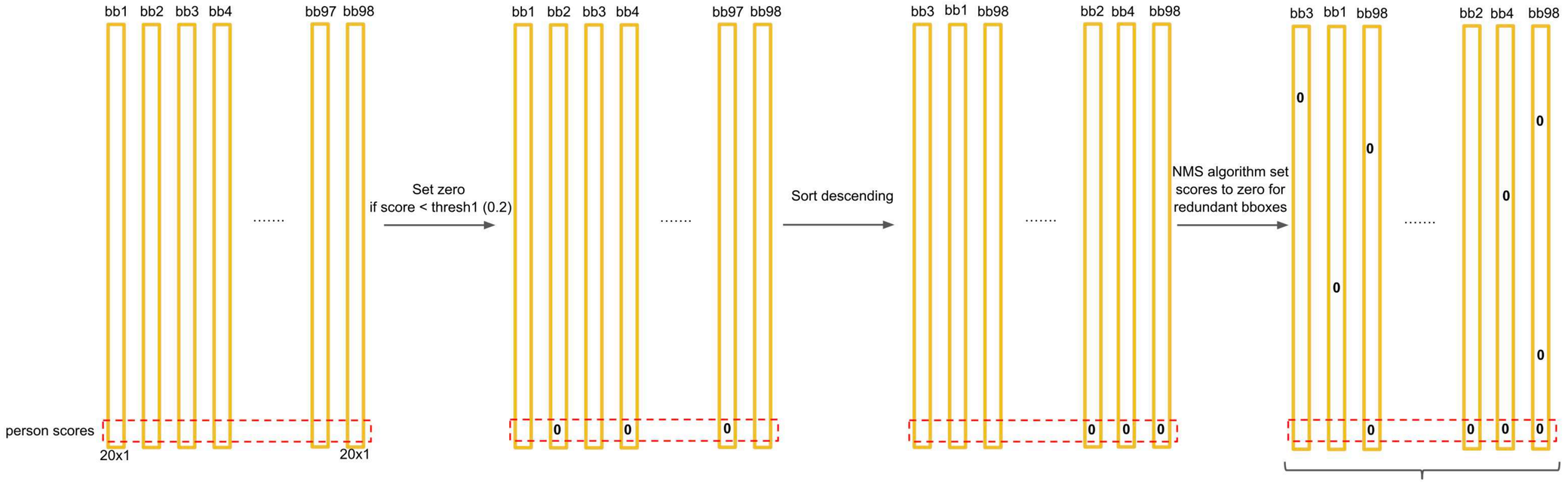
After comparison ~~almost all~~ pairs of bboxes the only two bboxes left with non-zero class score value.



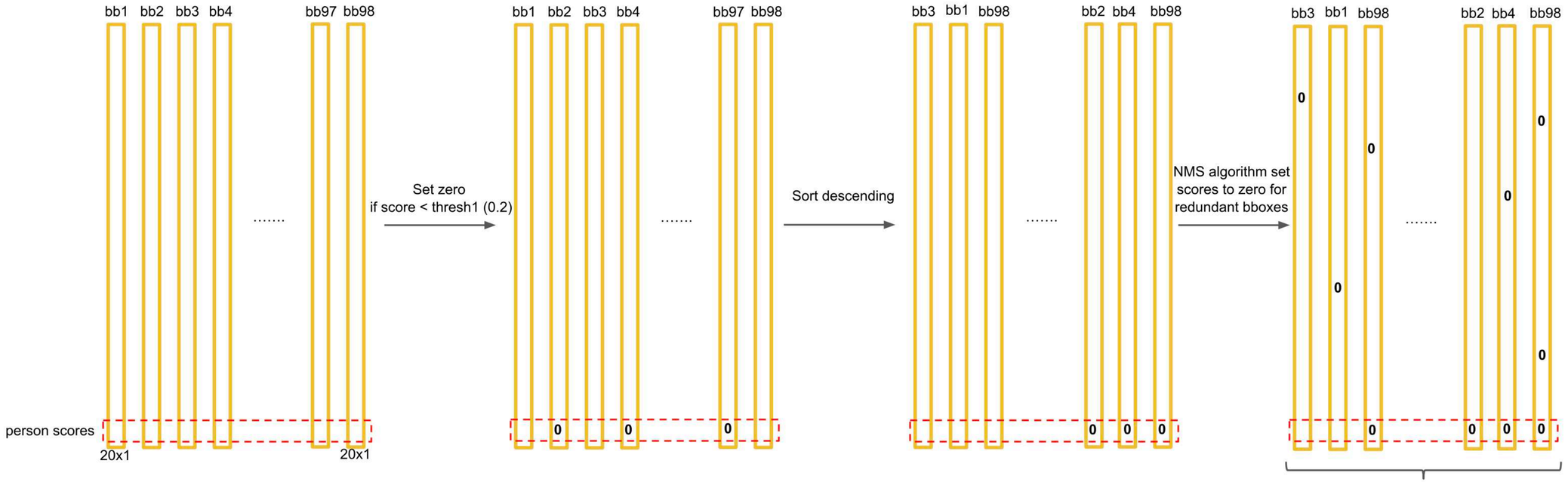

Do this procedure for next class



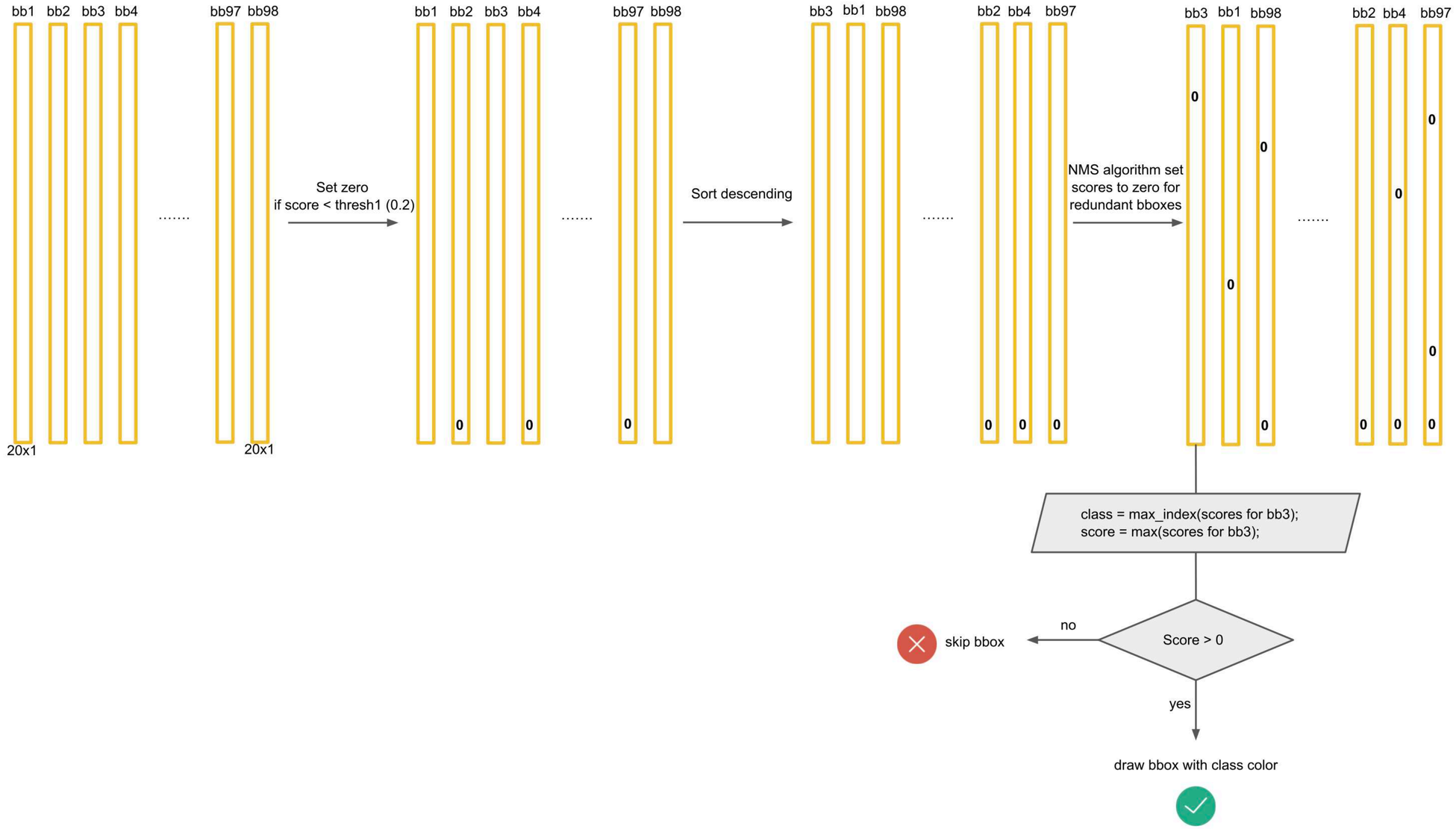

Do this procedure for all classes

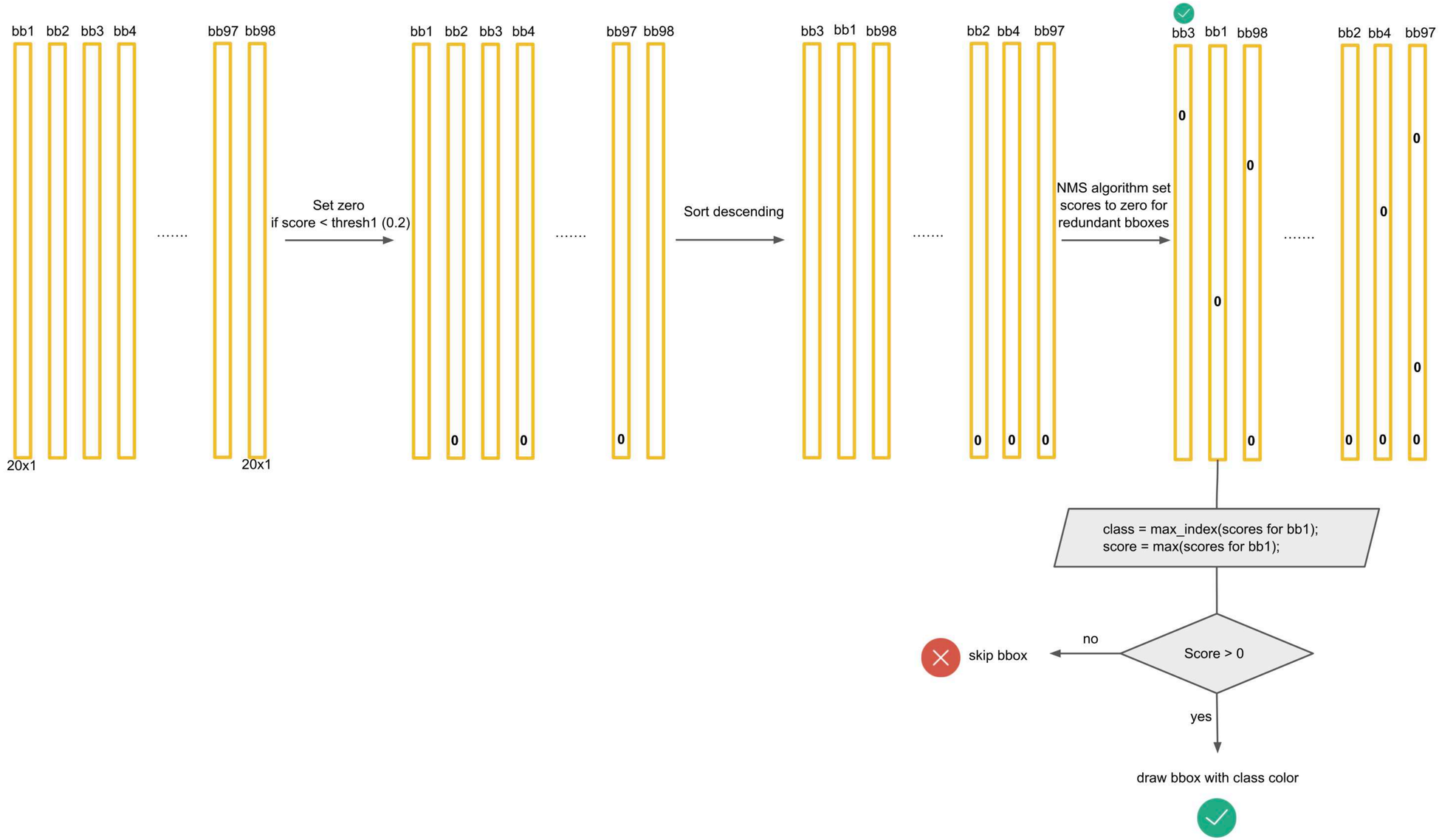


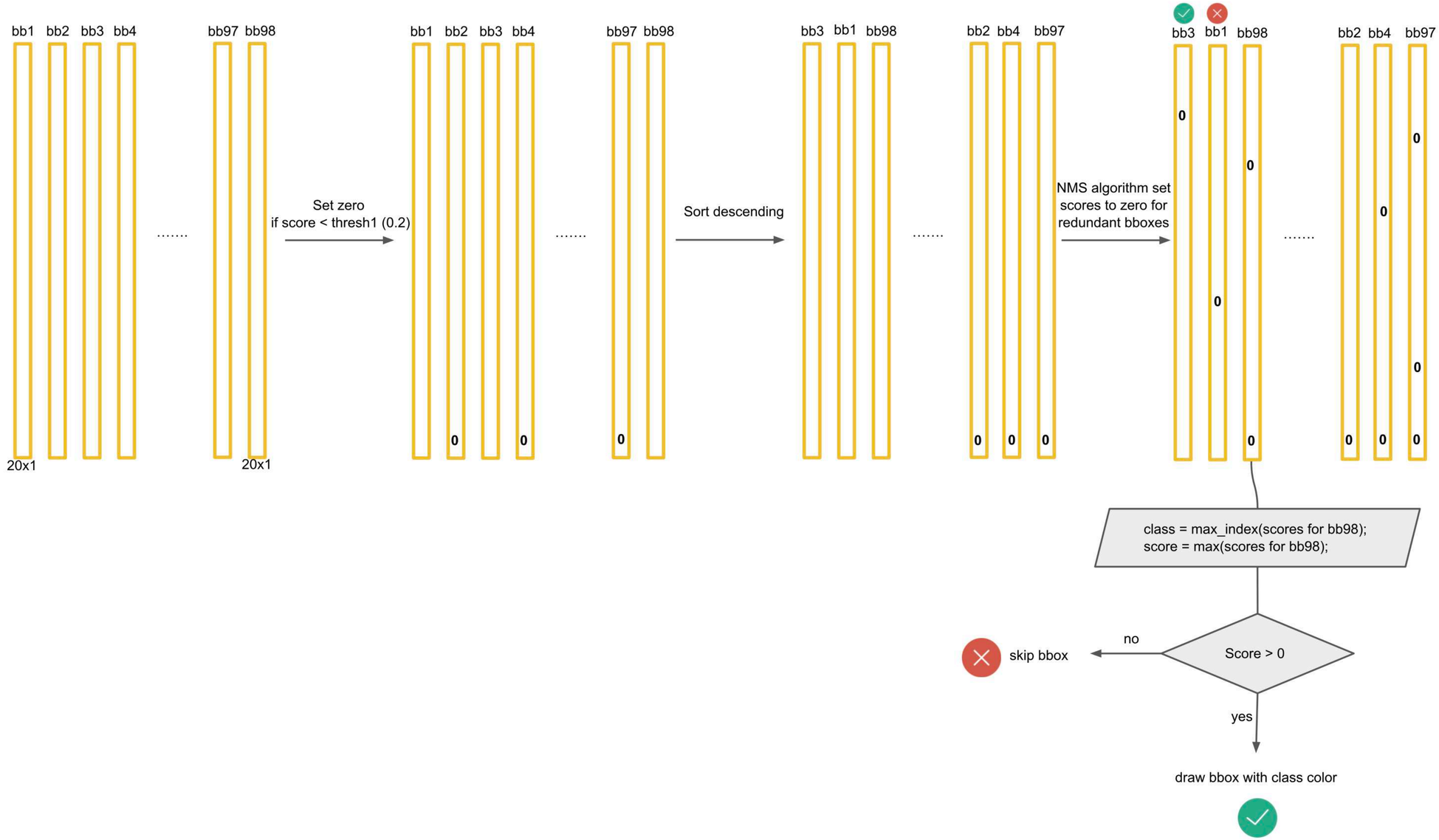

After this procedure -
a lot of zeros



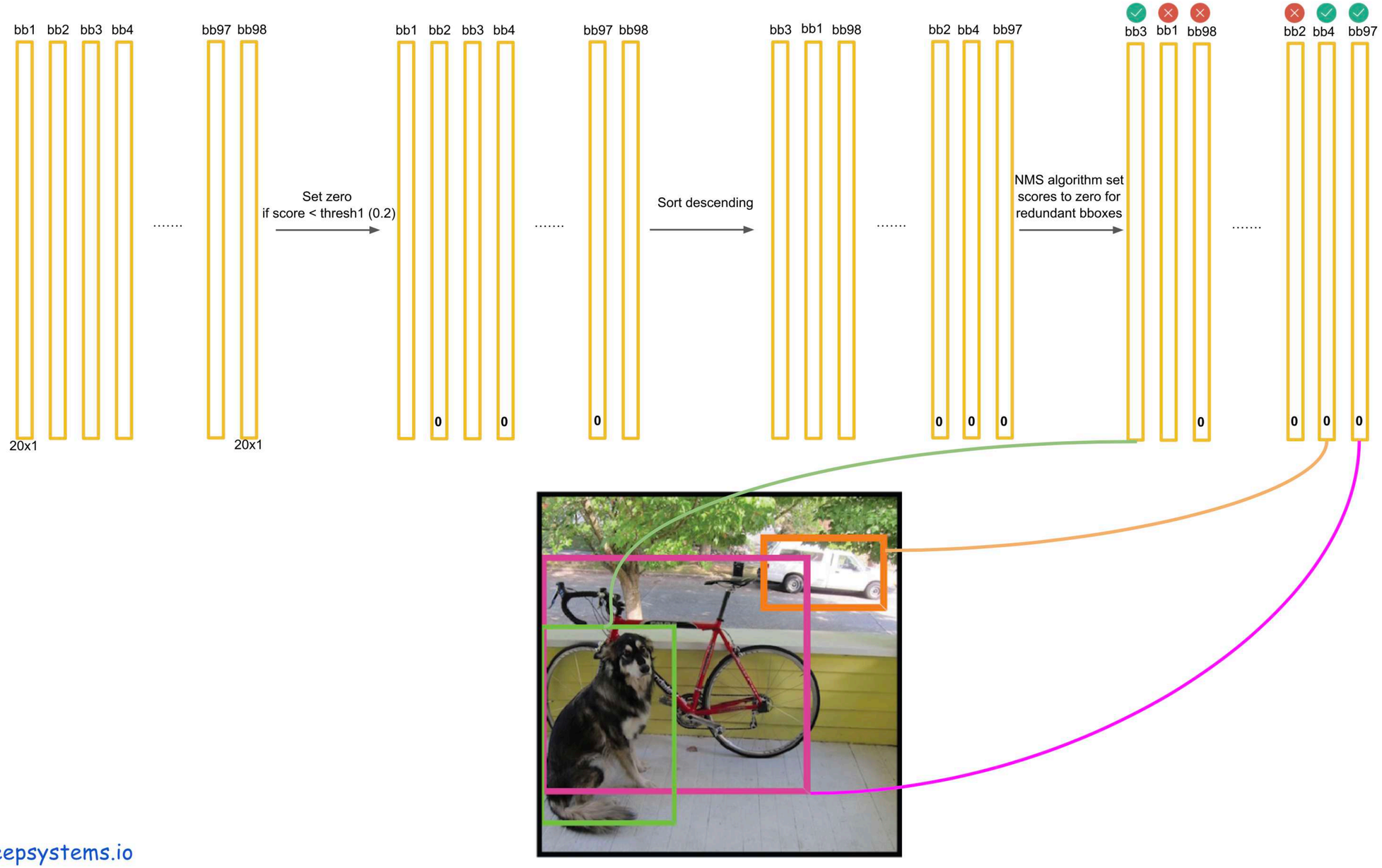

Select bboxes to draw by class score values


bb1 bb2 bb3 bb4
bb97 bb98
20x1
20x1
Set zero
if score < thresh1 (0.2)
Sort descending
NMS algorithm set
scores to zero for
redundant bboxes
class = max_index(scores for bb3);
score = max(scores for bb3);
Score > 0
no
skip bbox
yes
draw bbox with class color


bb1 bb2 bb3 bb4
bb97 bb98
20x1
20x1
Set zero
if score < thresh1 (0.2)
Sort descending
bb3 bb1 bb98
bb2 bb4 bb97
NMS algorithm set
scores to zero for
redundant bboxes
class = max_index(scores for bb1);
score = max(scores for bb1);
Score > 0
no
skip bbox
yes
draw bbox with class color


bb1 bb2 bb3 bb4
bb97 bb98
20x1
20x1
Set zero
if score < thresh1 (0.2)
Sort descending
bb3 bb1 bb98
bb2 bb4 bb97
NMS algorithm set
scores to zero for
redundant bboxes
class = max_index(scores for bb98);
score = max(scores for bb98);
Score > 0
no
skip bbox
yes
draw bbox with class color


bb1 bb2 bb3 bb4
bb97 bb98
20x1
20x1
Set zero
if score < thresh1 (0.2)
Sort descending
bb3 bb1 bb98
bb2 bb4 bb97
NMS algorithm set
scores to zero for
redundant bboxes
class = max_index(scores for bb97);
score = max(scores for bb97);
Score > 0
no
skip bbox
yes
draw bbox with class color


bb1 bb2 bb3 bb4
20x1
bb97 bb98
20x1
Set zero
if score < thresh1 (0.2)
bb1 bb2 bb3 bb4
0 0
bb97 bb98
0
Sort descending
bb3 bb1 bb98
bb2 bb4 bb97
0 0 0
NMS algorithm set
scores to zero for
redundant bboxes
bb3 bb1 bb98
0
bb2 bb4 bb97
0 0 0

# Limitation of YOLO

- Group of small objects
- Unusual aspect ratios
- Coarse feature
- Localization error of bounding box

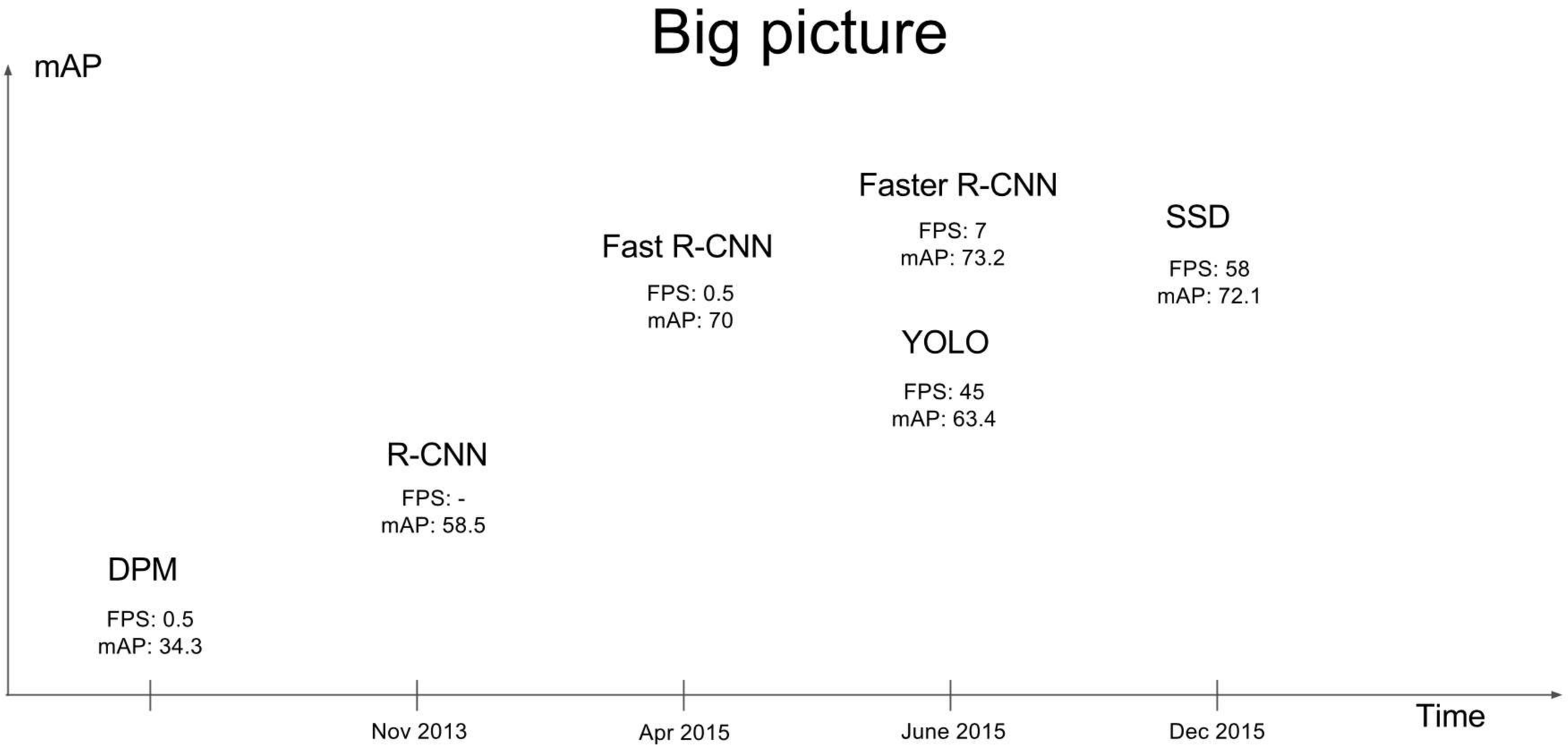
### 2.4. Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image.

Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations.

# Comparison to Other Detection System



# Comparison to Other Real-Time System

| 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 | **155** |
| YOLO | 2007+2012 | **63.4** | 45 |
| 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 |

**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.

# VOC Error

- 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

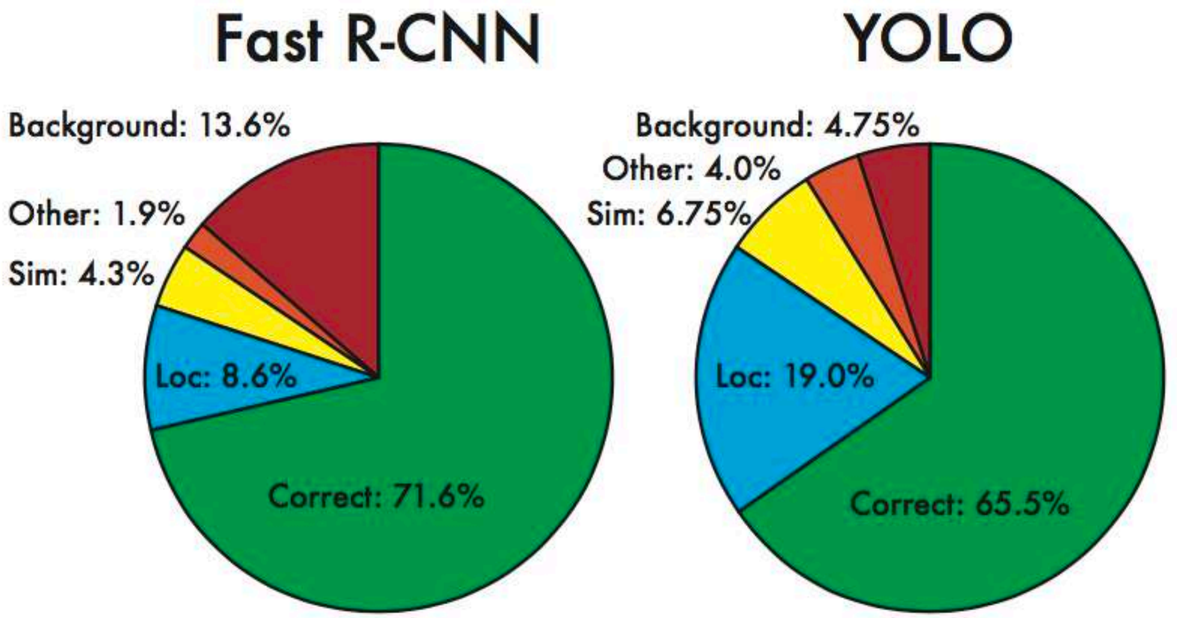



**Figure 4: Error Analysis: Fast R-CNN vs. YOLO** These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

# Combining Fast R-CNN and YOLO

| | mAP | Combined | Gain |
|---|---|---|---|
| Fast R-CNN | 71.8 | - | - |
| Fast R-CNN (2007 data) | **66.9** | 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 | **75.0** | **3.2** |

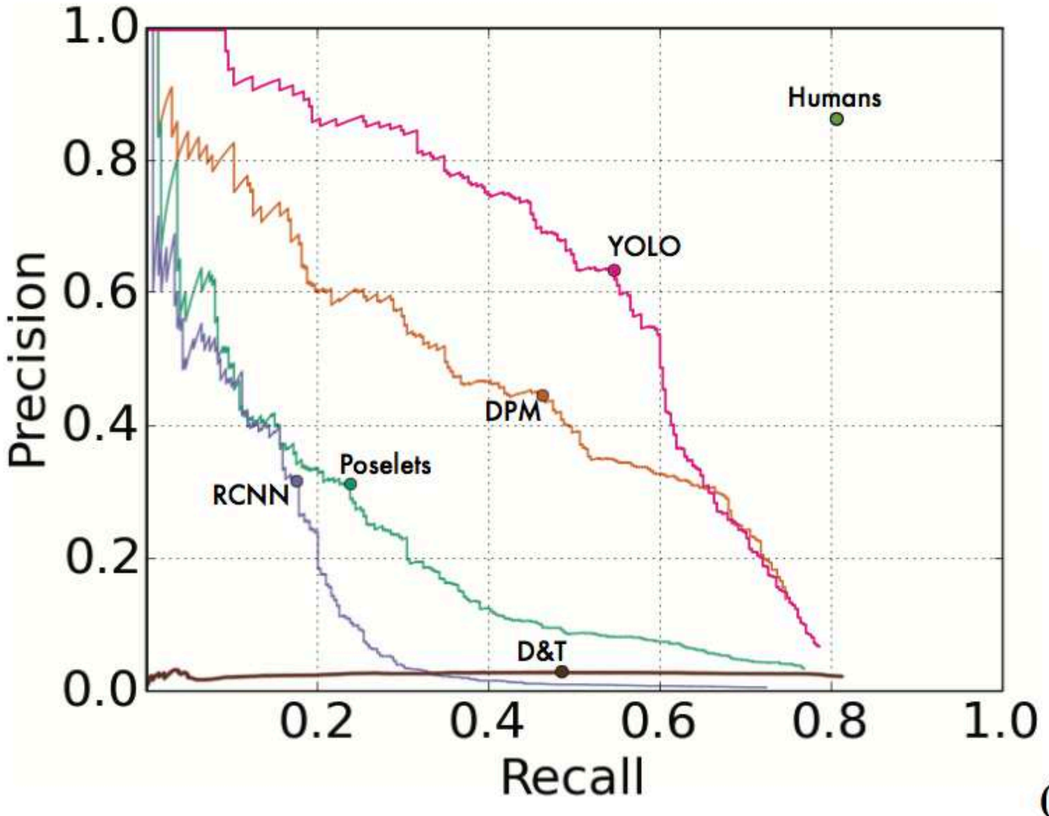
**Table 2: Model combination experiments on VOC 2007.** We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.

# VOC 2012 Leaderboard

| 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] | **73.9** | **85.5** | **82.9** | **76.6** | **57.8** | **62.7** | **79.4** | 77.2 | 86.6 | **55.0** | **79.1** | **62.2** | 87.0 | **83.4** | **84.7** | 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 | **79.8** | 87.7 | 49.6 | 74.9 | 52.1 | 86.0 | 81.7 | 83.3 | **81.8** | **48.6** | **73.5** | 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 |
| **Fast R-CNN + YOLO** | 70.7 | 83.4 | 78.5 | 73.5 | 55.8 | 43.4 | 79.1 | 73.1 | **89.4** | 49.4 | 75.5 | 57.0 | **87.5** | 80.9 | 81.0 | 74.7 | 41.8 | 71.5 | 68.5 | **82.1** | 67.2 |
| 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 | **68.8** | 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 | **87.5** | 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 | 64.0 | 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 | 64.7 | 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 |
| **YOLO** | 57.9 | 77.0 | 67.2 | 57.7 | 38.3 | 22.7 | 68.3 | 55.9 | 81.4 | 36.2 | 60.8 | 48.5 | 77.2 | 72.3 | 71.3 | 63.5 | 28.9 | 52.2 | 54.8 | 73.9 | 50.8 |
| 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.

# Generalizability: Person Detection in Artwork




**(a)** Picasso Dataset precision-recall curves.

| | VOC 2007 | Picasso | | People-Art |
|---|---|---|---|---|
| | AP | AP | Best $F_1$ | AP |
| **YOLO** | **59.2** | **53.3** | **0.590** | **45** |
| 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.**

# Generalizability: Person Detection in Artwork

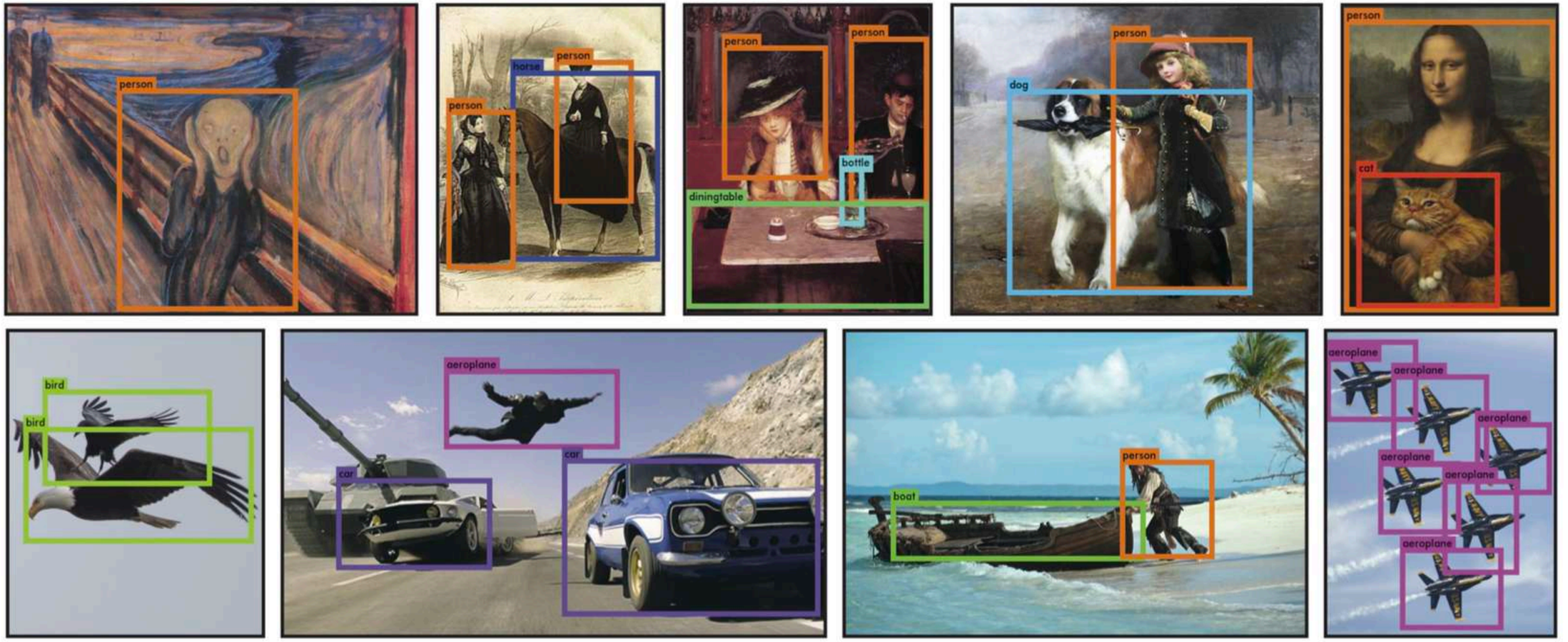
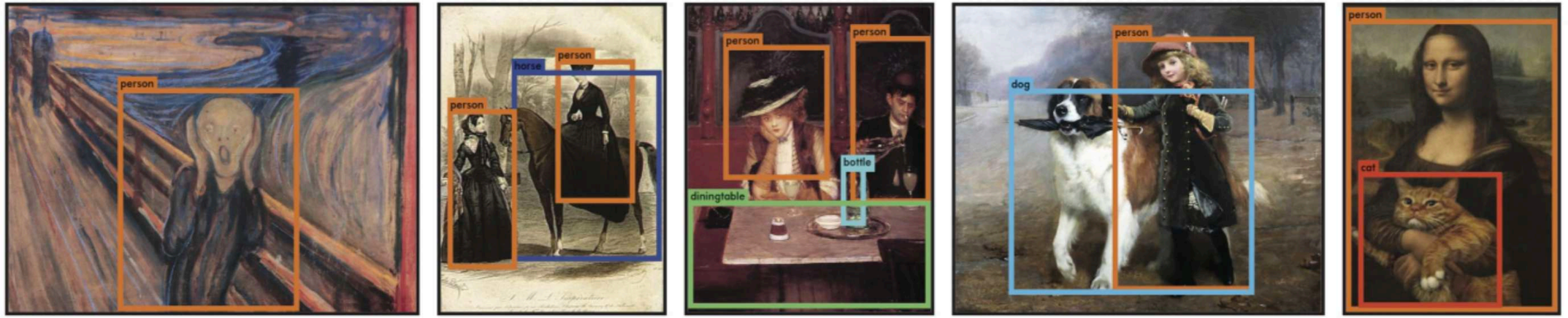



**Figure 6: Qualitative Results.** YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

# Key Points

1. Fast: YOLO - 45 fps, YOLO-tiny - 155 fps.
2. End-to-end training.
3. Makes more localization errors but is less likely to predict false positives on background
4. Performance is lower than the current state of the art.
5. Combined Fast R-CNN + YOLO model is one of the highest performing detection methods.
6. Learns very general representations of objects: it outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains



# Appendix | Implementation

- YOLO (darknet): https://pjreddie.com/darknet/yolov1/
- YOLOv2 (darknet): https://pjreddie.com/darknet/yolo/
- YOLO (caffe): https://github.com/xingwangsfu/caffe-yolo
- YOLO (TensorFlow: Train+Test): https://github.com/thtrieu/darkflow
- YOLO (TensorFlow: Test): https://github.com/gliese581gg/YOLO_tensorflow

# Appendix | Slides

- DeepSense.io (google presentation - "YOLO: Inference")

# Thanks



fb.com/taegyun.jeon


github.com/tgjeon


taylor.taegyun.jeon@gmail.com

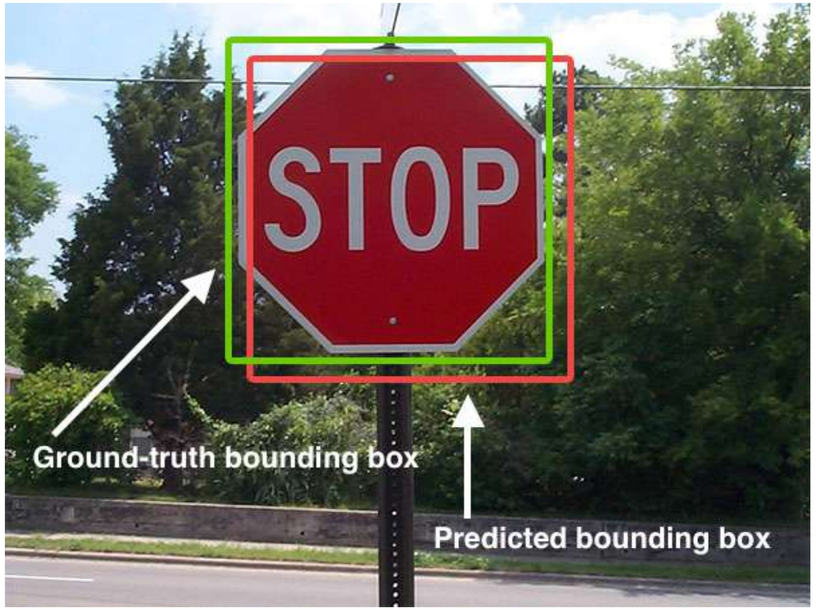

Paper reviewed by

**Taegyun Jeon**

# Appendix | Intersection over Union (IoU)

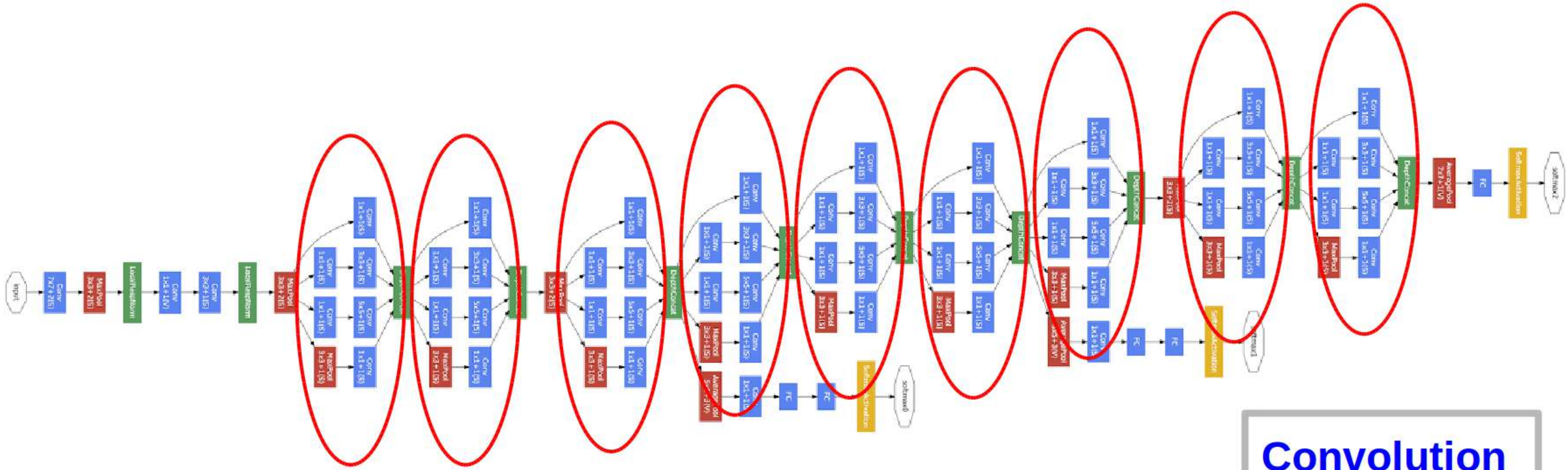
- IoU(pred, truth)=[0, 1]

$$IoU(A,B)=\frac{|A\bigcap B|}{|A\bigcup B|}$$




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

# Appendix | GoogLeNet

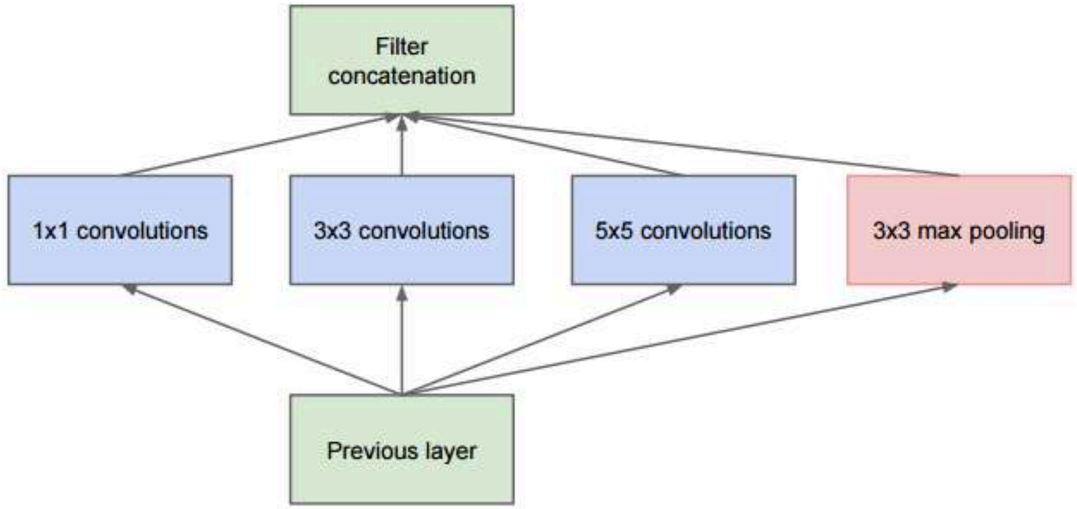


Convolution
Pooling
Softmax
Concat/Normalize

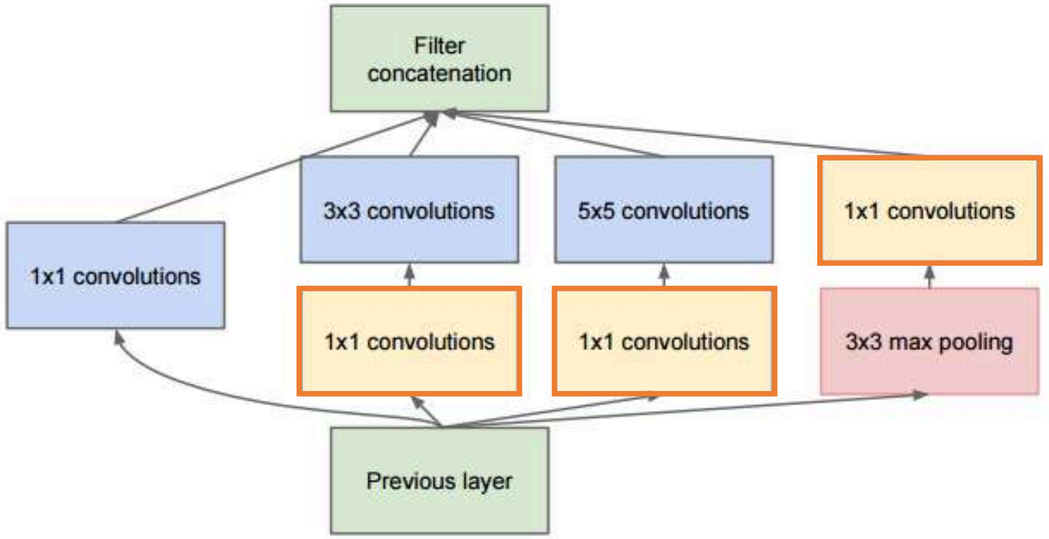
Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

# Appendix | GoogLeNet

**Inception Module** (1x1 convolution for dimension reductions)




(a) Inception module, naïve version




(b) Inception module with dimension reductions

# Appendix | Networks on Convolutional Feature Maps

## Previous: FC

## Proposed : Conv + FC

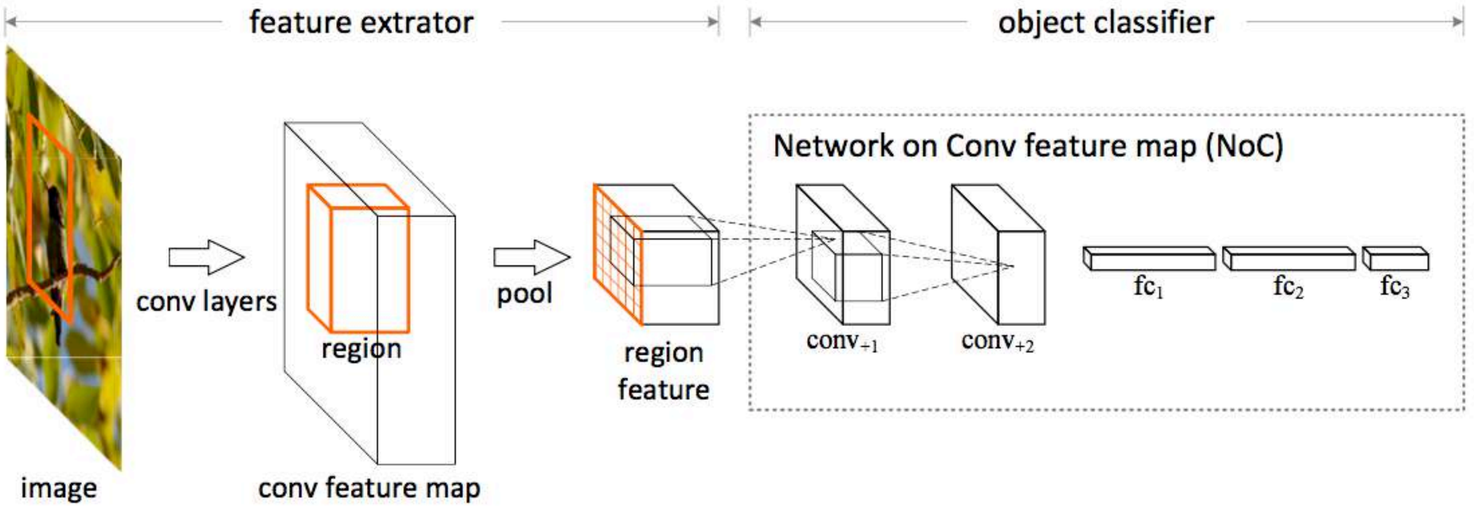



Figure 1: Overview of NoC. The convolutional feature maps are generated by the shared convolutional layers. A feature map region is extracted and RoI-pooled into a fixed-resolution feature. A new network, called a NoC, is then designed and trained on these features. In this illustration, the NoC architecture consists of two convolutional layers and three fully-connected layers.

Ren, Shaoqing, et al. "Object detection networks on convolutional feature maps." IEEE Transactions on Pattern Analysis and Machine Intelligence (2016).

# Appendix | Sum-squared error (SSE)

**sum of squared errors of prediction (SSE)**, is the sum of the squares of residuals (deviations predicted from actual empirical values of data). It is a measure of the discrepancy between the data and an estimation model. A small RSS indicates a tight fit of the model to the data. It is used as an optimality criterion in parameter selection and model selection.

$$SSE = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

https://en.wikipedia.org/wiki/Residual_sum_of_squares