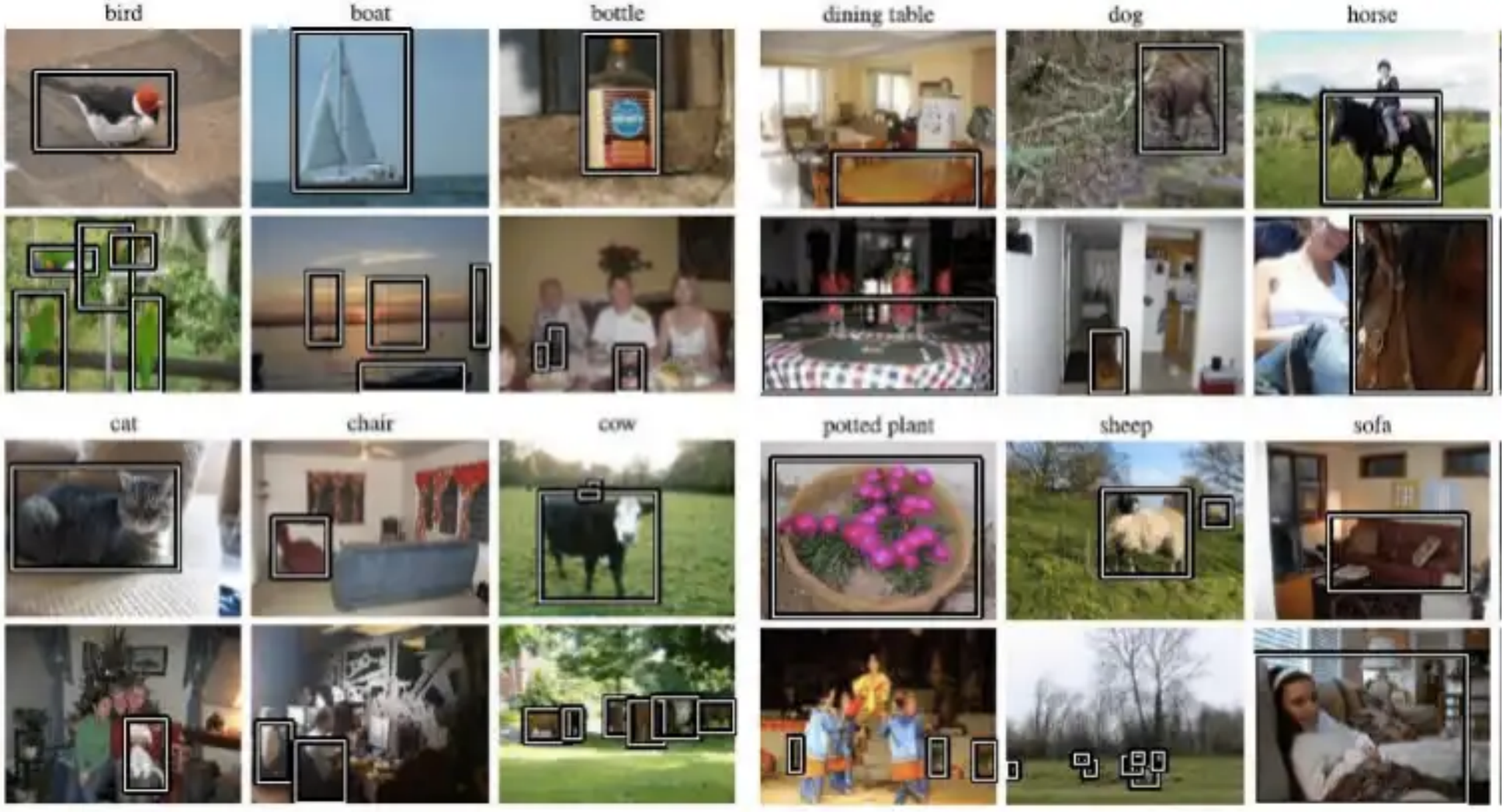
You Only Look Once

31 Oct
Hyeonwoo Yoo

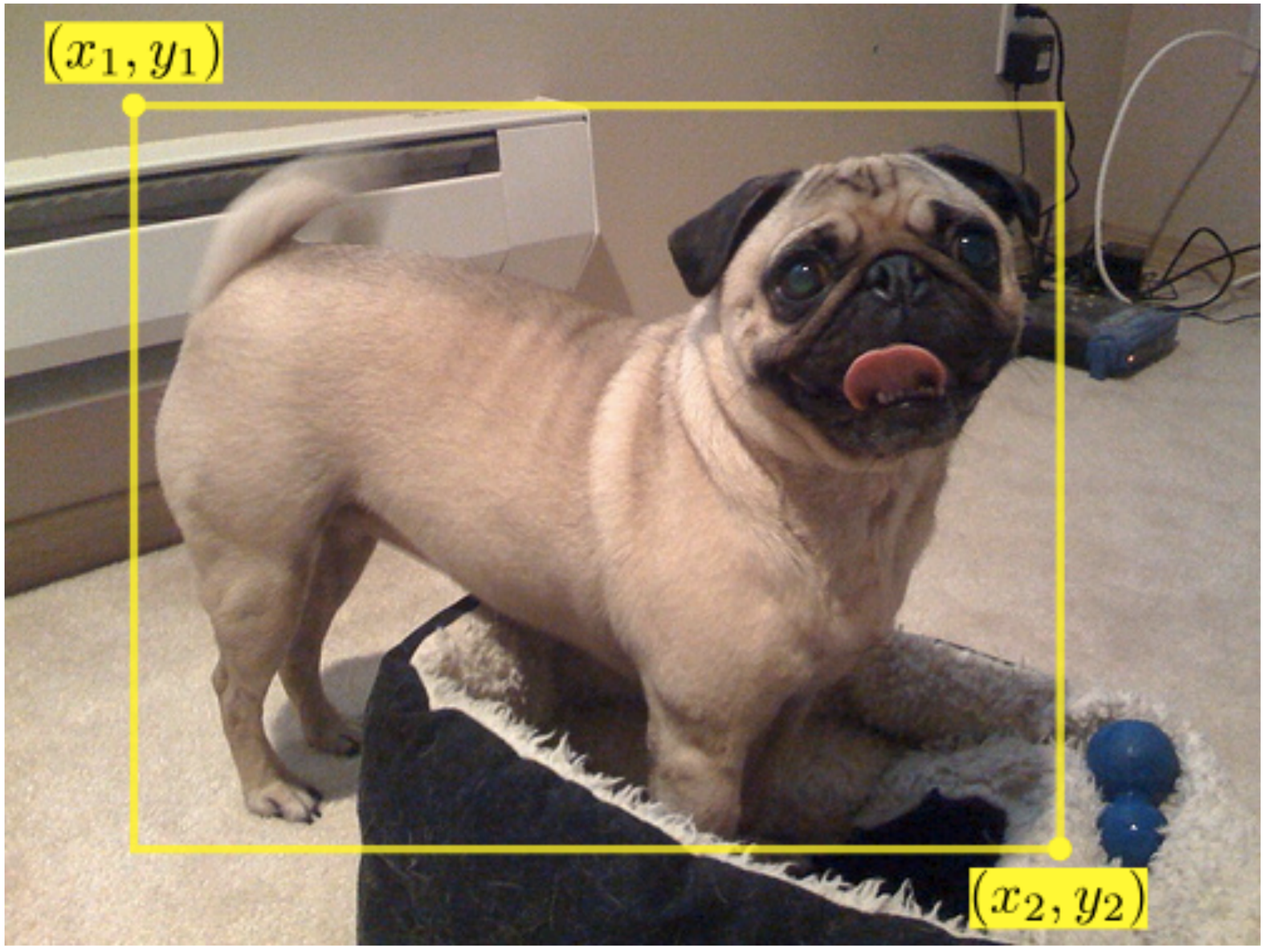
Contents

- Terms
- PASCAL VOC Dataset
- Short history of previous models
- How different YOLO is from that
- Model design
- How they did training
- Error
- Loss function



Classification

At which class it belongs?



Detection

At which position it locates?

원본 이미지

Semantic Segmentation

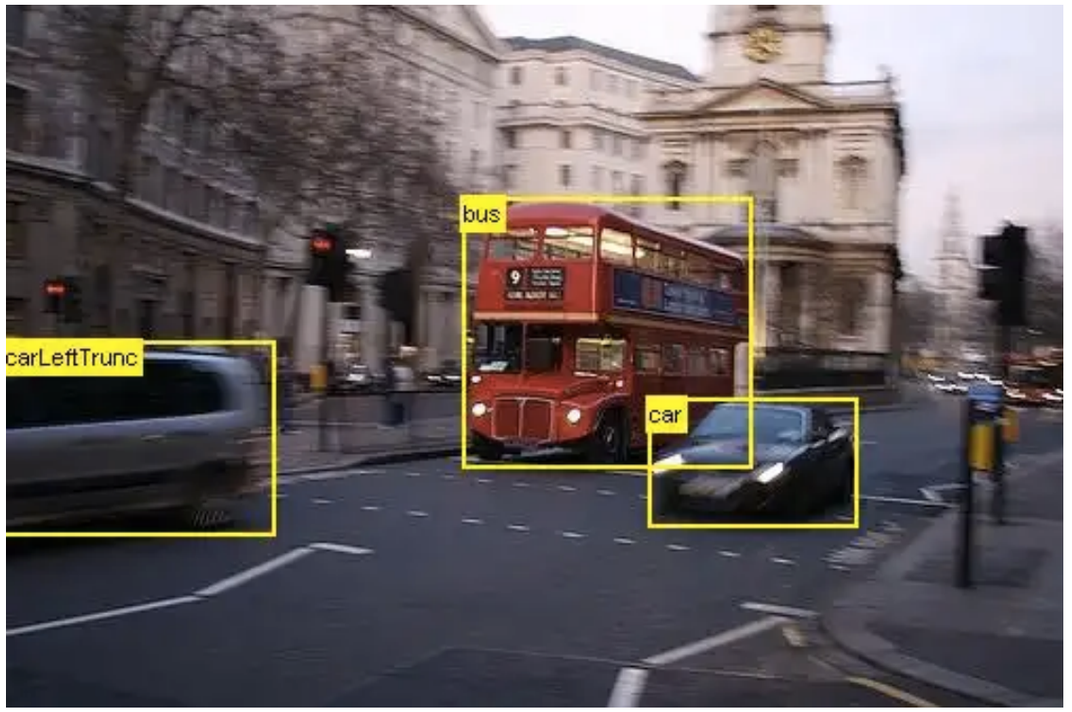
Instance Segmentation

Segmentation

At which pixel it exists?

PASCAL VOC DATASET

- One of the early image-recognizing competition
- Contains more than 10,000 images
- Requires Classification, Detection, Segmentation
- Evaluates with Confidence score overs threshold
- Uses Precision and Recall instead of Accuracy



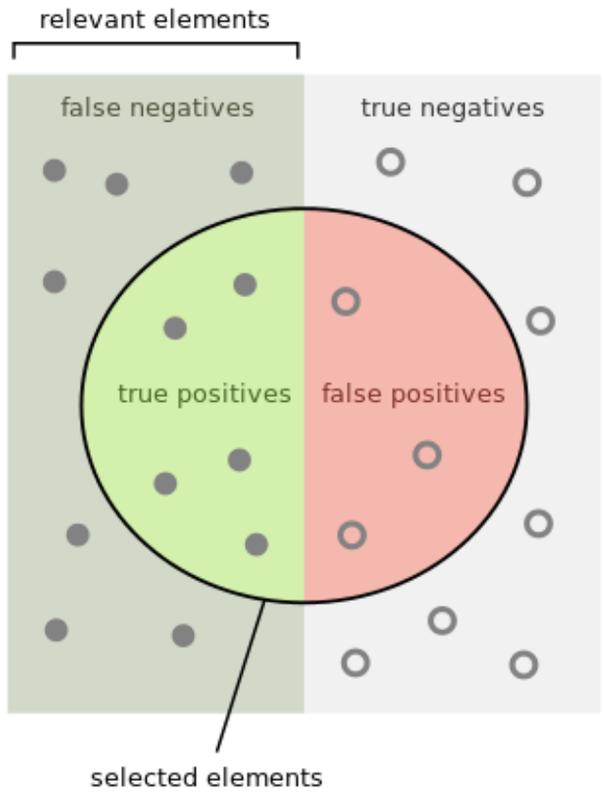
Precision and Recall

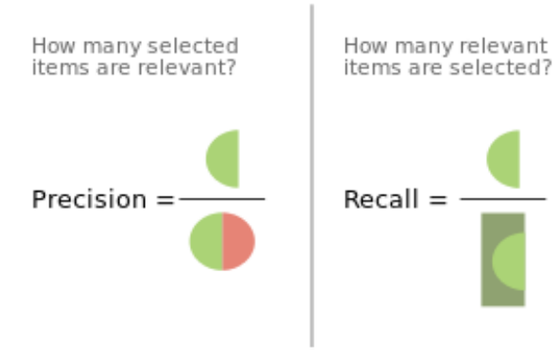
- Precision

- $$\frac{number\ of\ successfully\ classified\ images\ as\ class\ A}{number\ of\ images\ predicted\ to\ be\ in\ classA}$$

- Recall

- $$\frac{number\ of\ successfully\ classified\ images\ as\ class\ A}{entire\ number\ of\ images\ in\ class\ A}$$

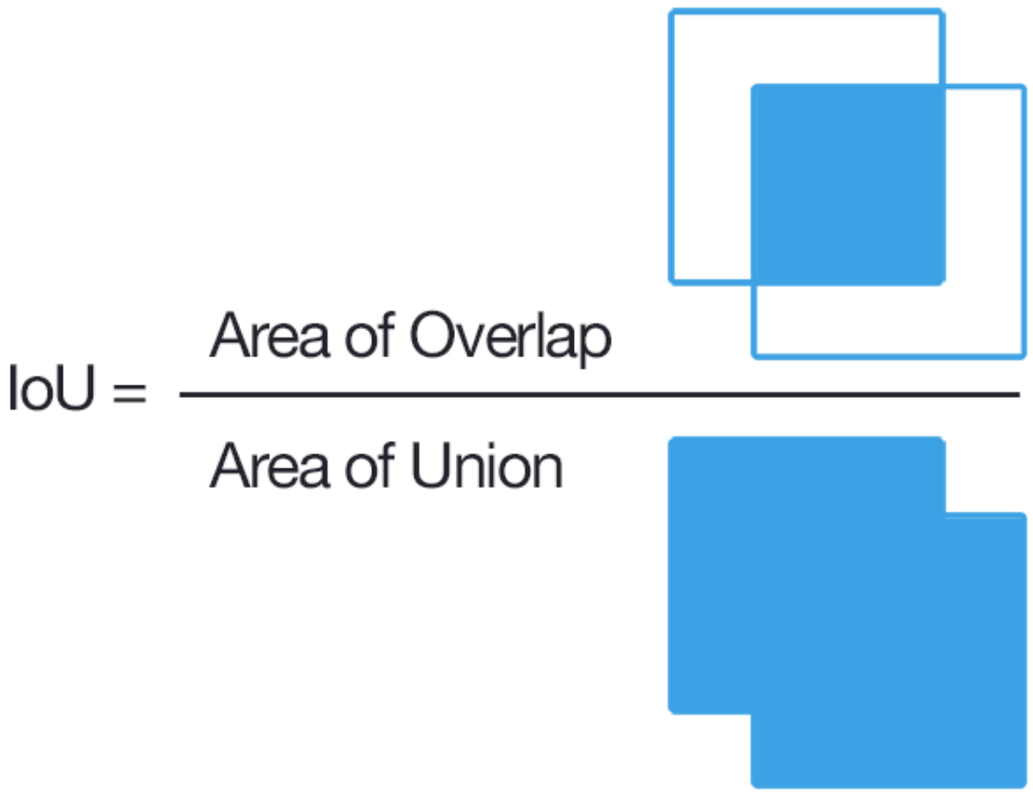




GT and IOU

- Ground truth(GT) : Where bounding box in training data exists

- Intersection over union(IOU) : Metrics will be used to determine how accurate it detects the true bounding box(Ground truth)



Short history of previous models

- DPM : uses sliding window approach. Classifier is run at evenly spaced locations over the entire images

- -> slow, requires lot of computation

- R-CNN : generate potential bounding boxes first, run classifier on these proposed boxes

- -> slow, hard to optimize because each individual component must be trained separately

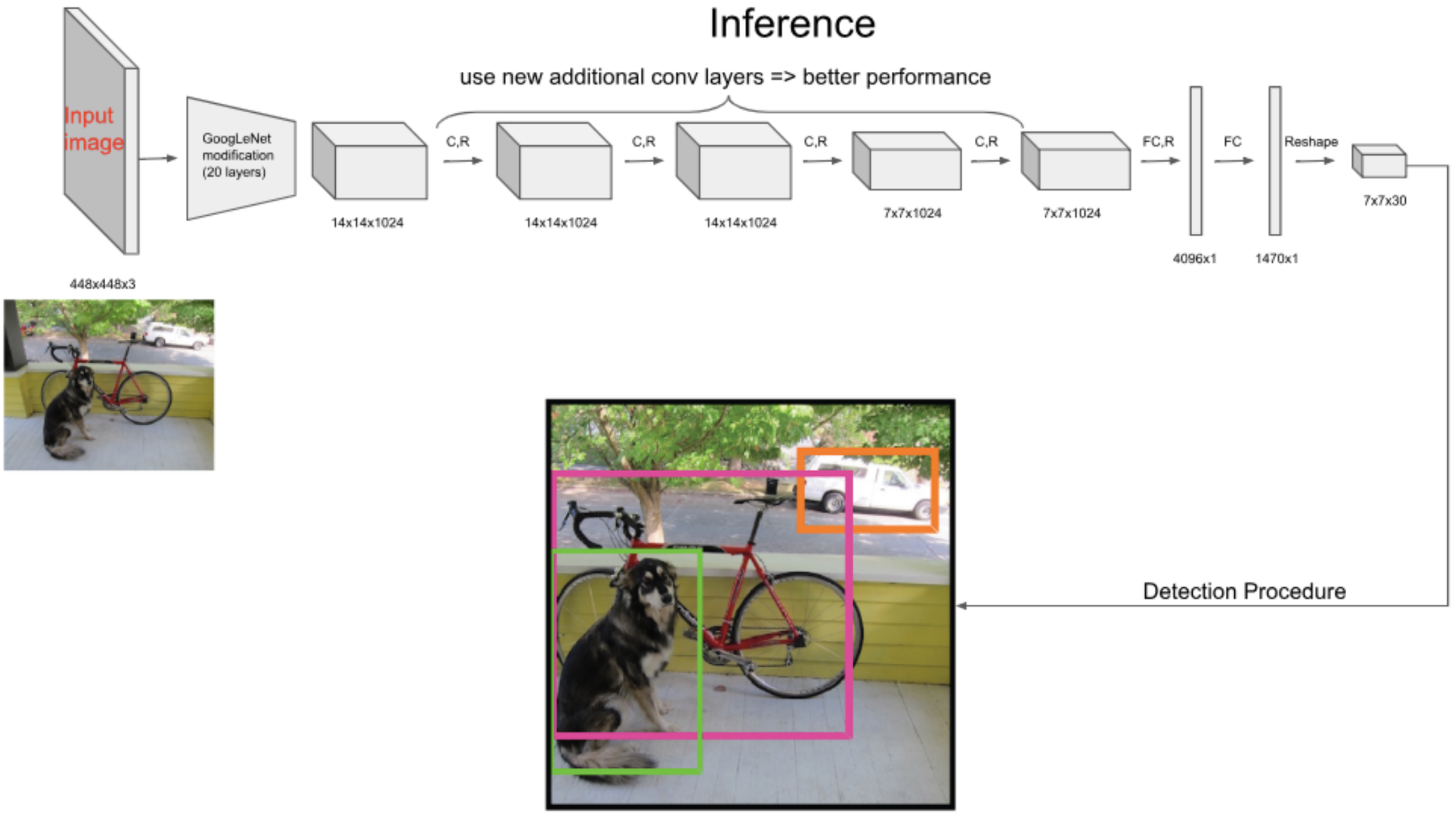
Differentiated points from previous models

- Extremely fast : It doesn't require a complex pipeline

- Not losing the context : Since it sees entire image during the training and test time, contextual informations is expected to be encoded in the model unlike R-CNN

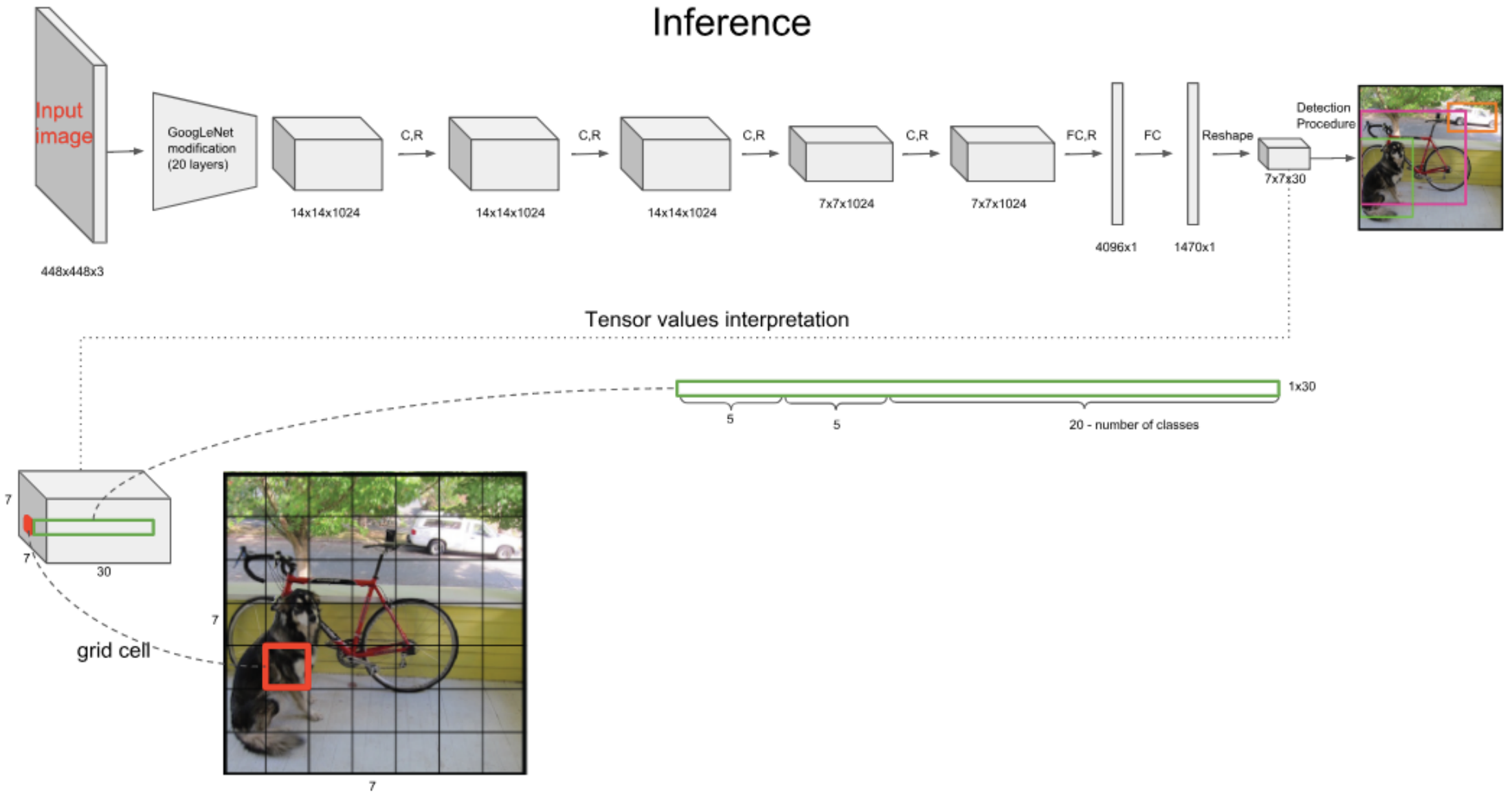
- Learns generalizable representations of objects : It even works with artwork

Model design



Source : https://goo.gl/jmXeFJ

Model design



Source : https://goo.gl/jmXeFJ

How they did train it

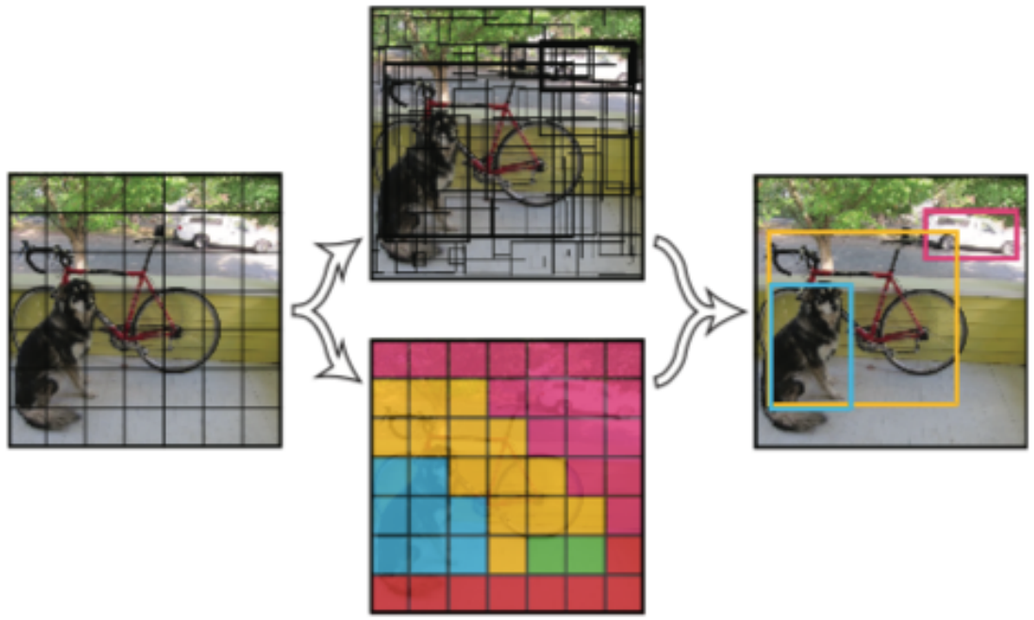
- Pre-train convolutional layers on the ImageNet 1000-class competition dataset
- Add 4 convolutional layers and 2 fully connected layers with randomly initialized weights, resolution is increase in this level, due to detection (It requires higher resolution)
- FINAL LAYER normalizes the bounding box width and height by the image width and height so that they fall between 0 and 1
- (x, y) should also be bounded between 0 and 1
- Activation Function : Linear for final layer, Leaky ReLu for all other

FINAL LAYER

1) System divides the input images into S x S grid
2) Each grid cell predicts B bounding boxes and confidence score for those boxes
3) Confidence is defined as Pr(Object) * IOU(truth over prediction). If no object exists in that cell, confidence score would be 0
4) Each bounding box predicts 5 values : (x, y : coordinates), (w, h : size of box), and confidence
5) Output : S x S X (B * 5 + C) tensor

S: square-root of # of grid
B : # of bounding boxes to detect
C : determines which class is most likely to match with the grid

Thus, B boxes are assigned for each grid cell

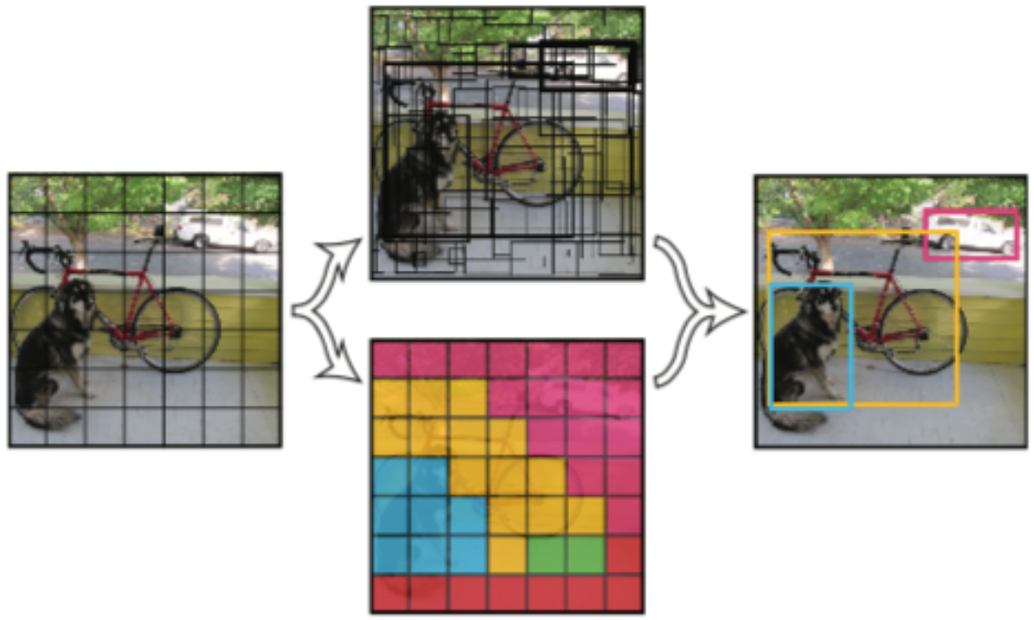


FINAL LAYER

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

1st term at left side
: conditional class probability

1st term at right side
: class probability



FINAL LAYER

S=7, B=2, C=2

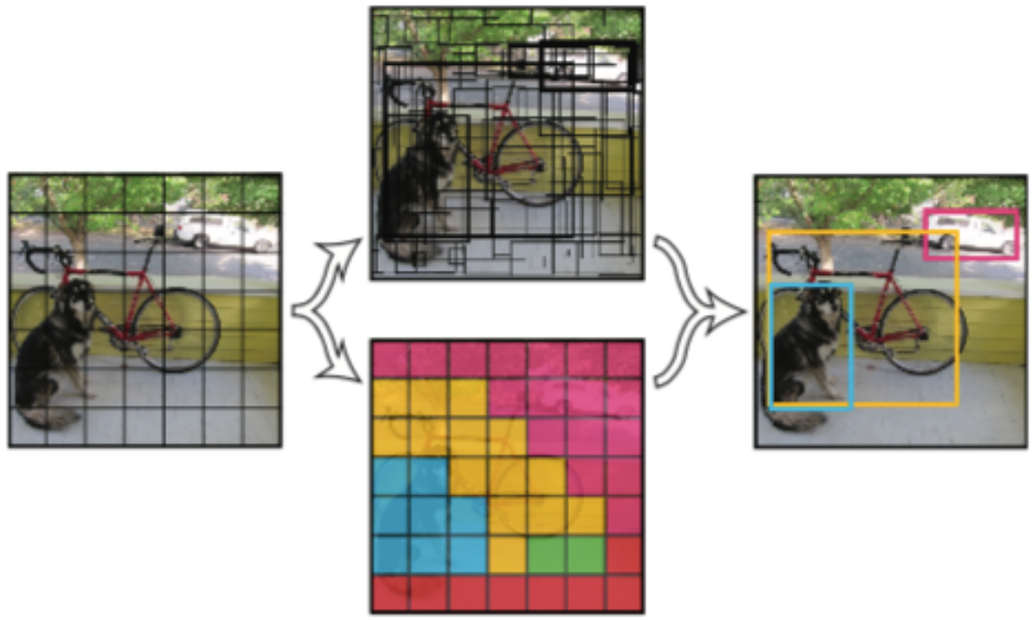
As a outcome of final layer, each grid produces

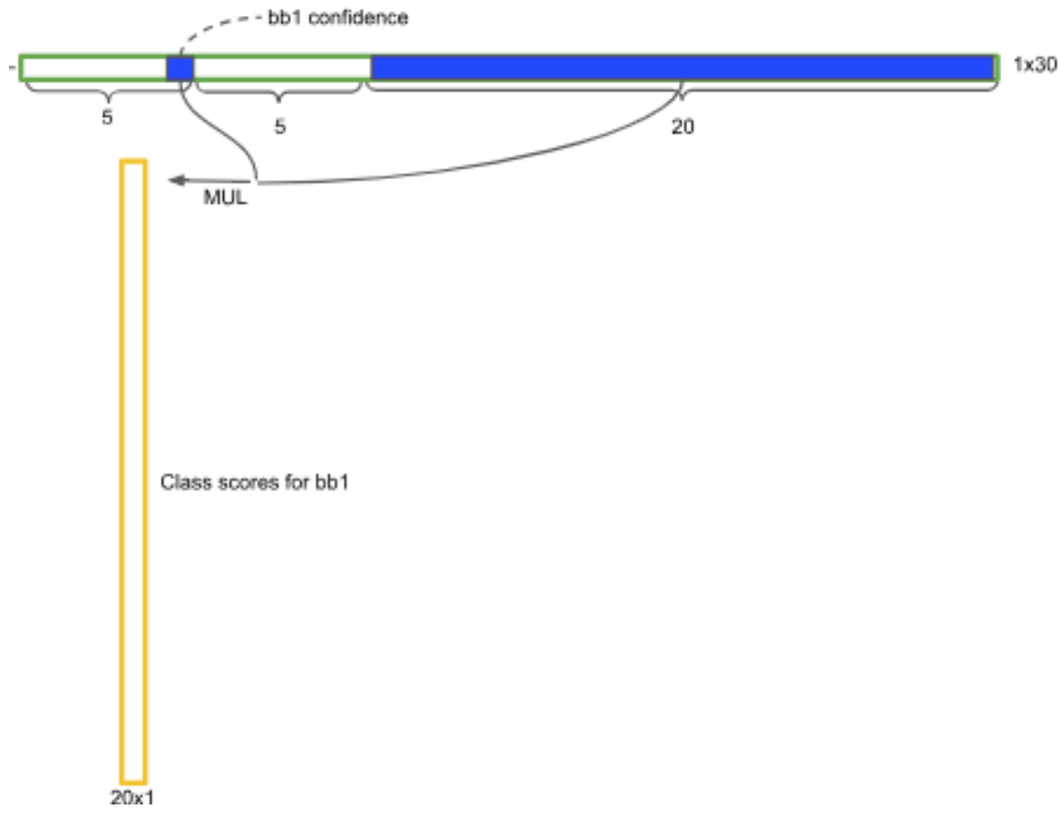
2 * 1 * 20 vector

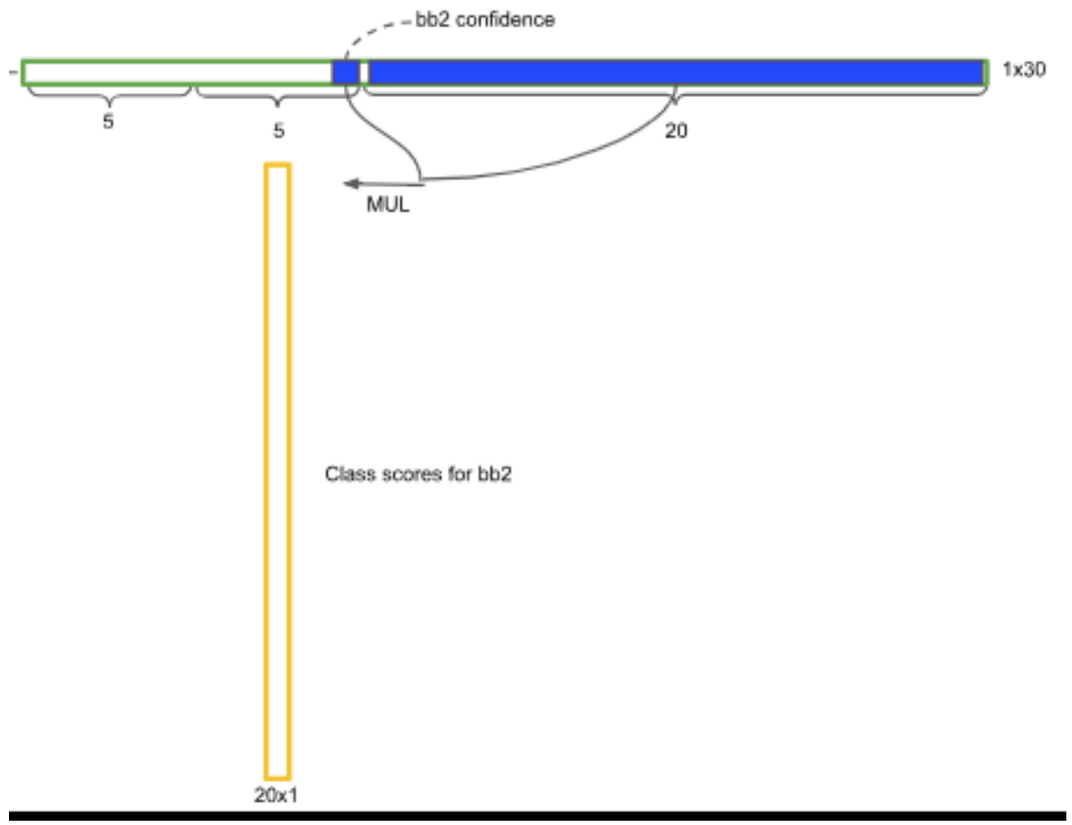
2: # of boxes
20 : # of classes

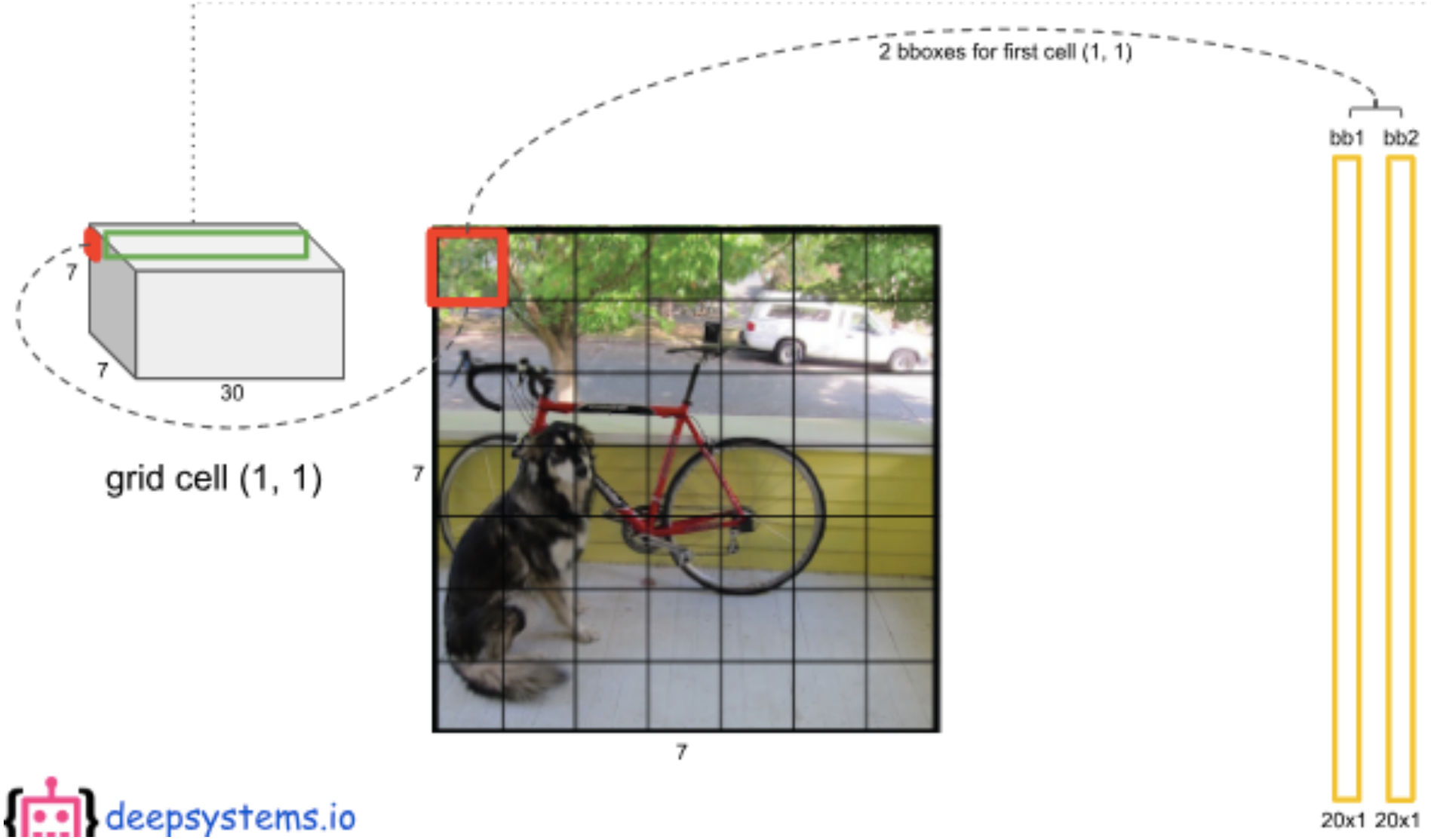
The whole outcome would be
(2 * 1 * 20) * (7 * 7)

(what each grid produces)
* (# of all grids)

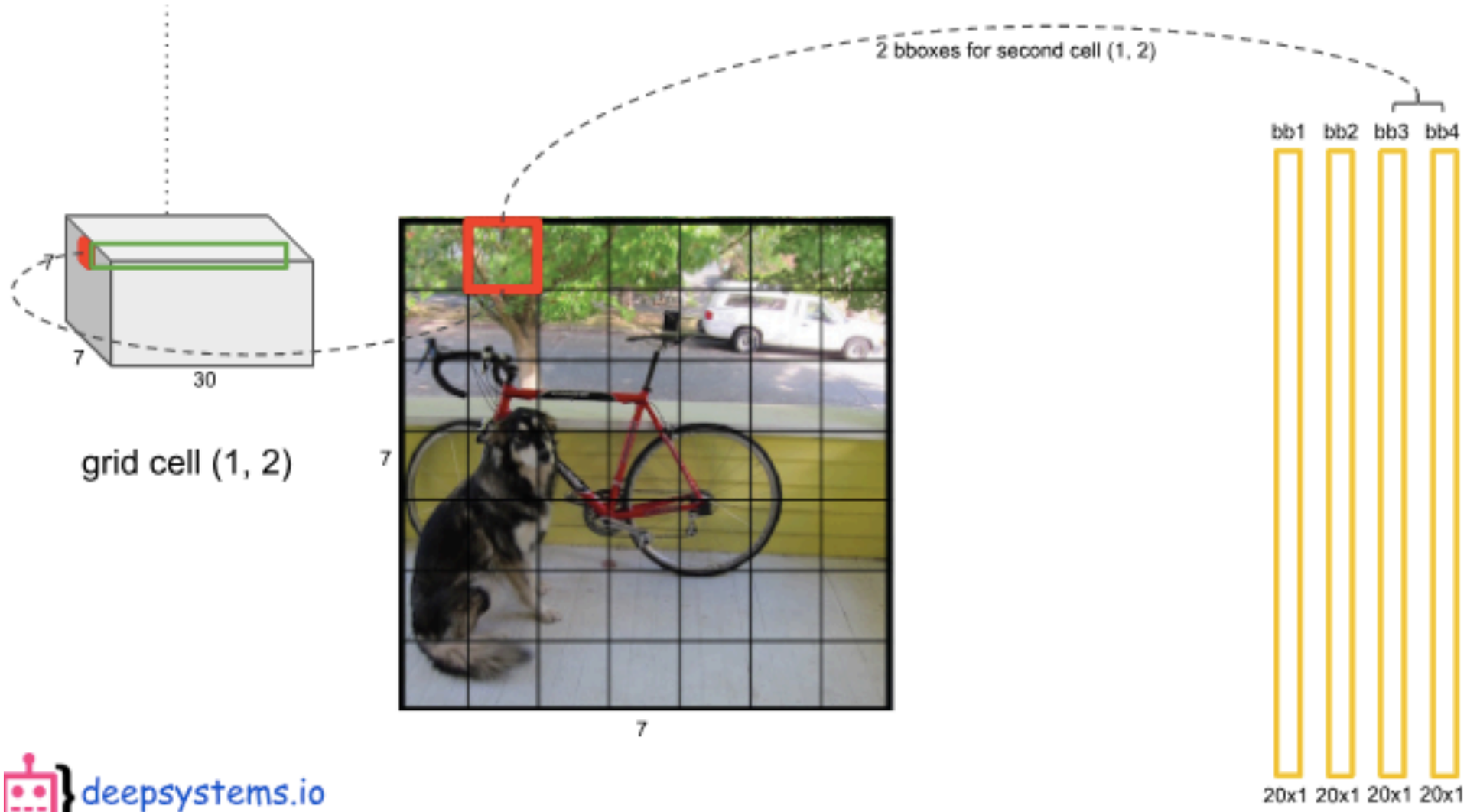



bb1 confidence
1x30
5
5
20
MUL
Class scores for bb1
20x1

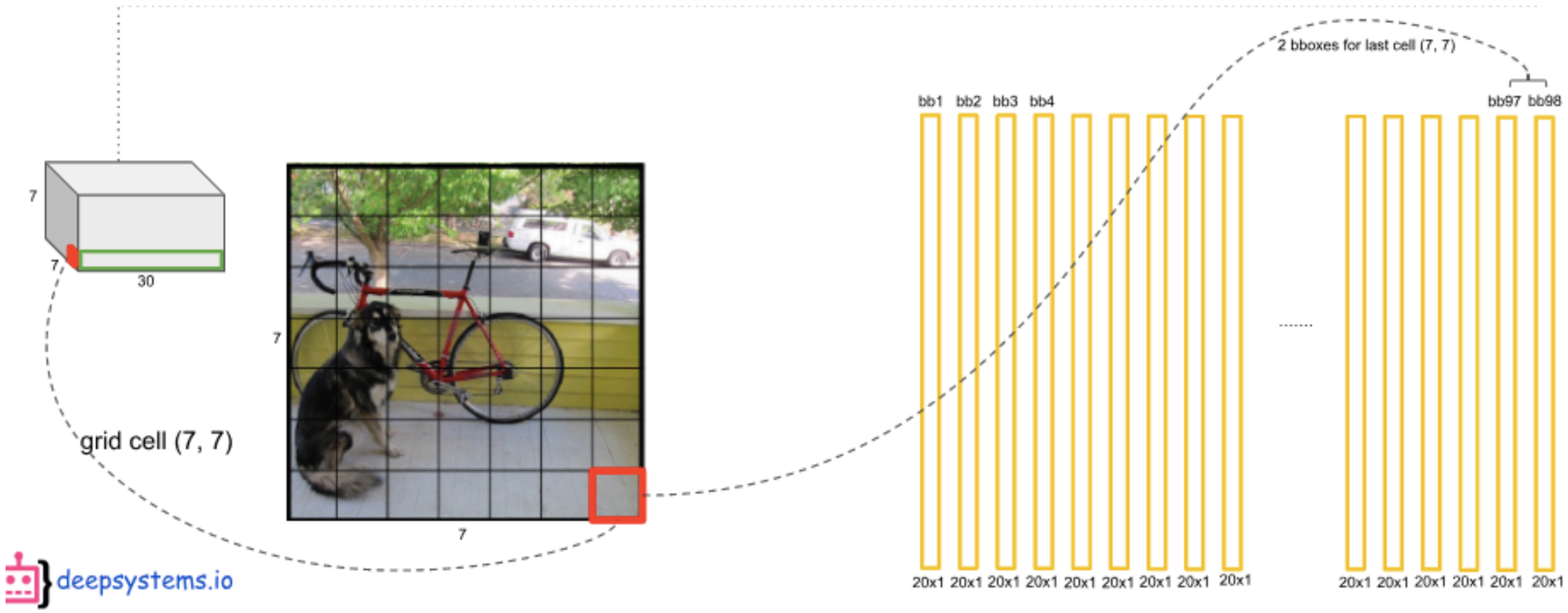

bb2 confidence
1x30
5
5
20
MUL
Class scores for bb2
20x1



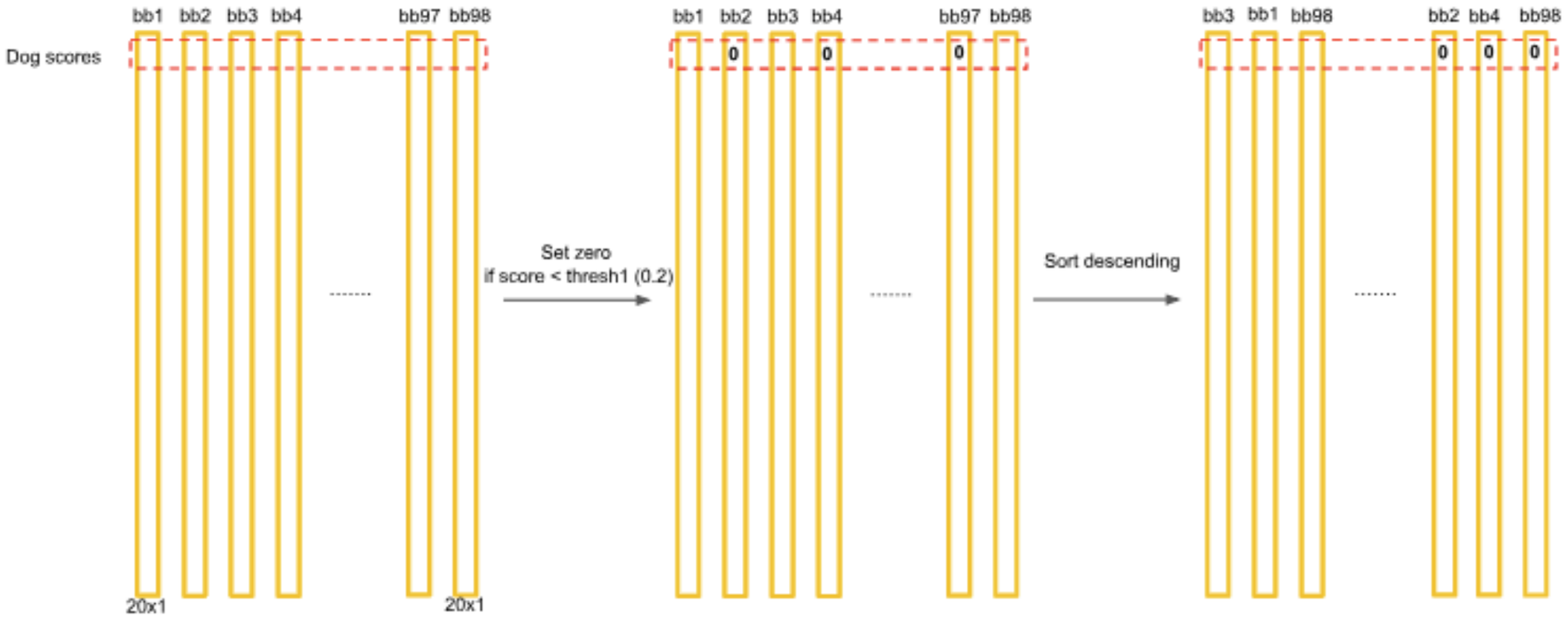
deepsystems.io



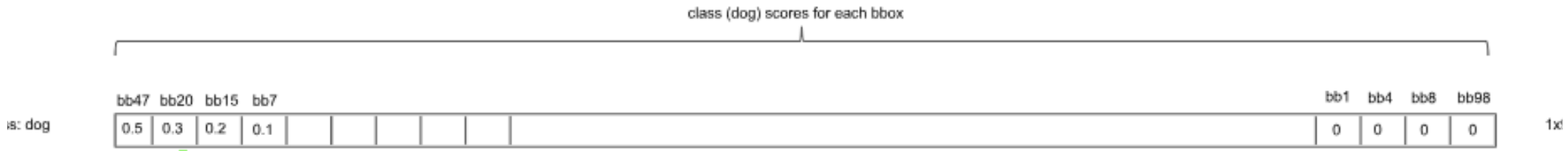
deepsystems.io

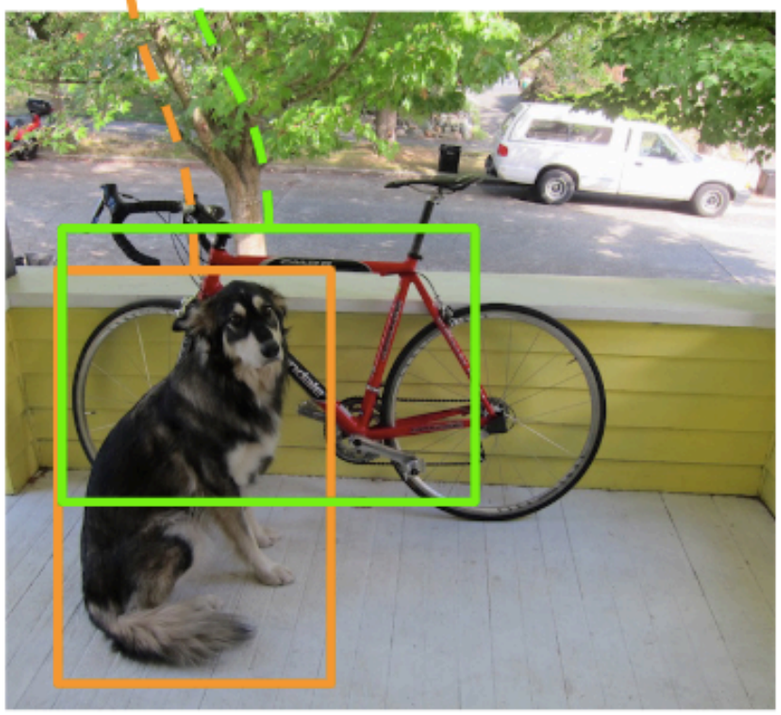

7
7
30
7
7
grid cell (7, 7)
2 bboxes for last cell (7, 7)
bb1 bb2 bb3 bb4
bb97 bb98
20x1 20x1 20x1 20x1 20x1 20x1 20x1 20x1 20x1
20x1 20x1 20x1 20x1 20x1 20x1
deepsystems.io

7 * 7 * 2 = 98 boxes


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

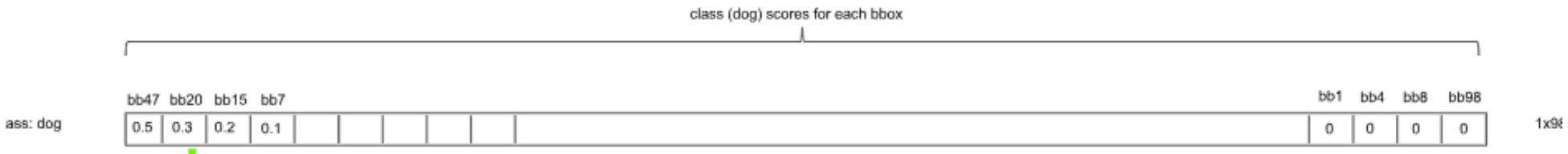
Non-Maximum Suppression: intuition

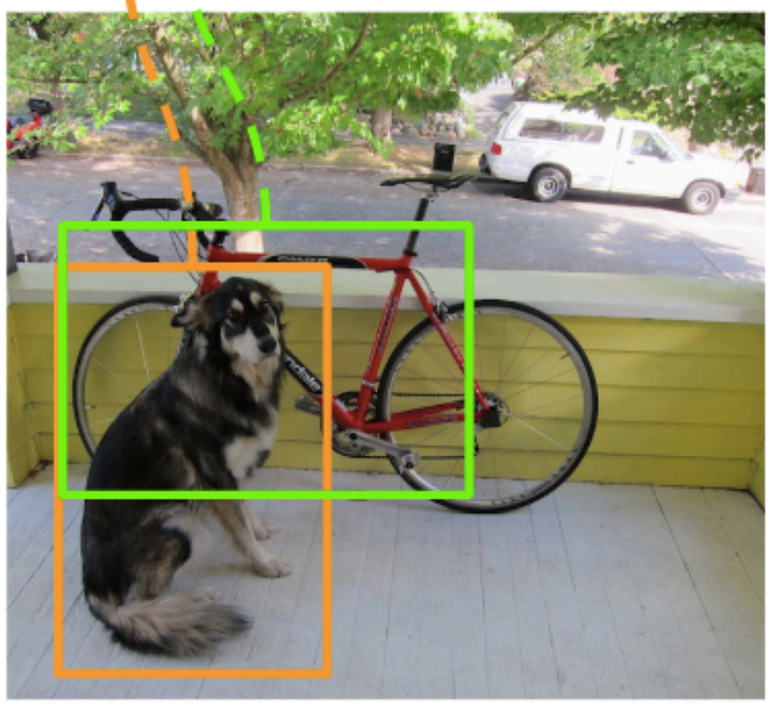




Compare “bbox_max” with others less score (non-zero!) bboxes. Let’s denote it “bbox_cur”

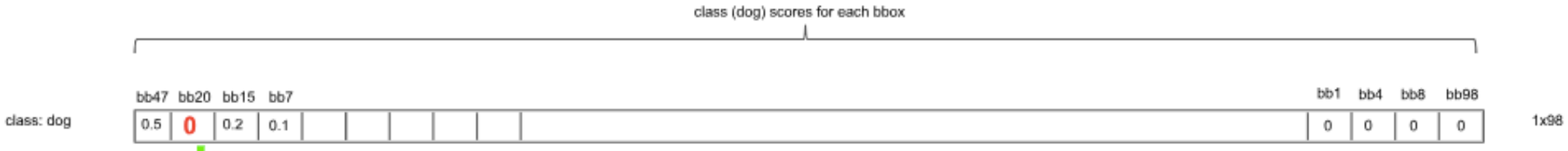
Non-Maximum Suppression: intuition

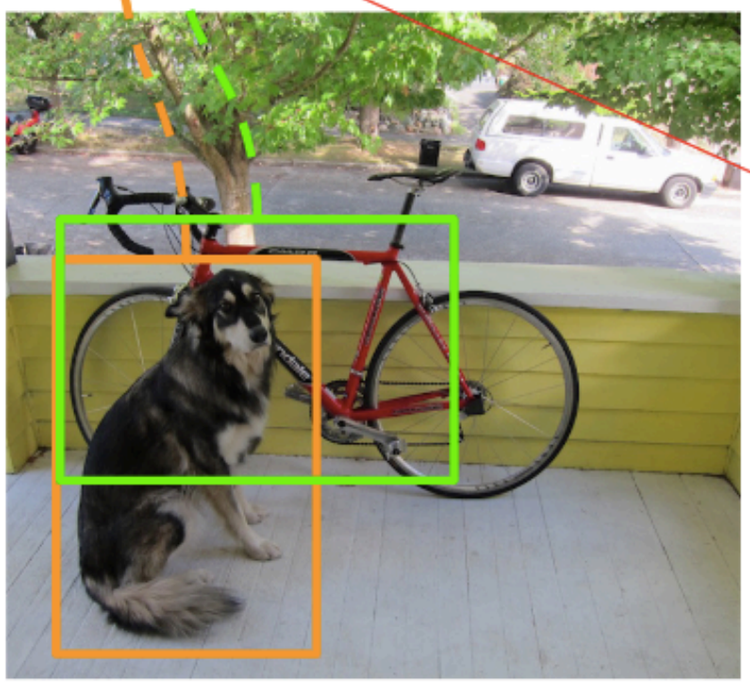




If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

Non-Maximum Suppression: intuition

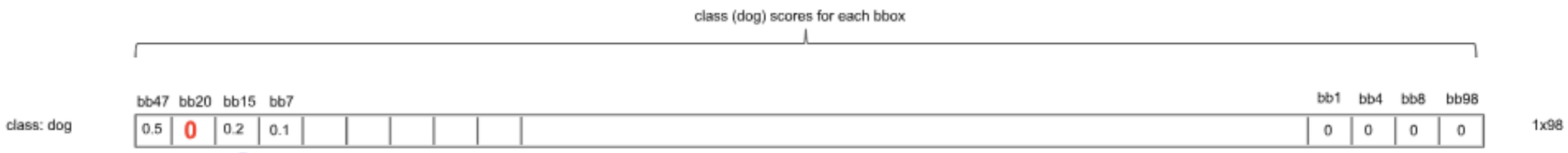


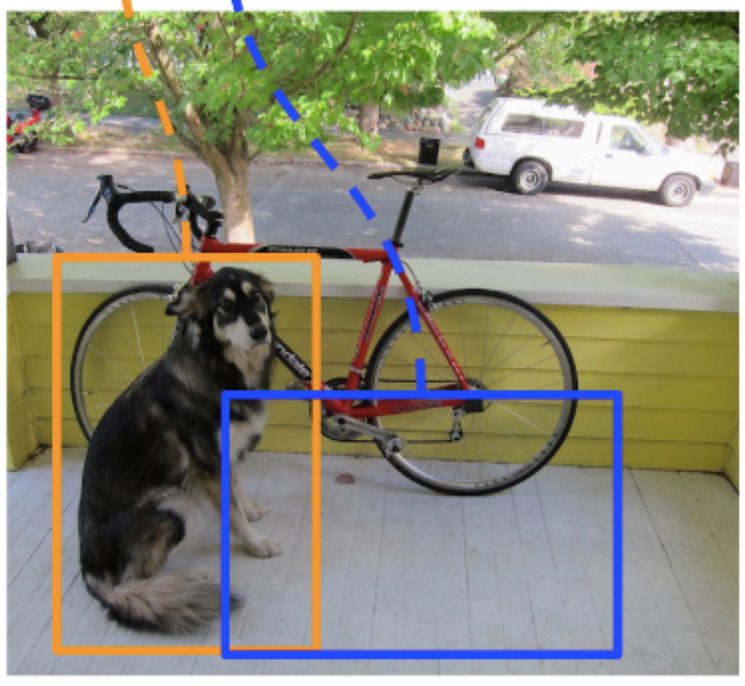


If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: set to 0.

Non-Maximum Suppression: intuition



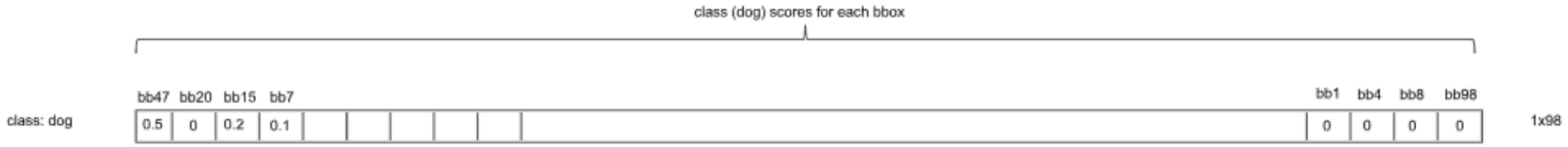


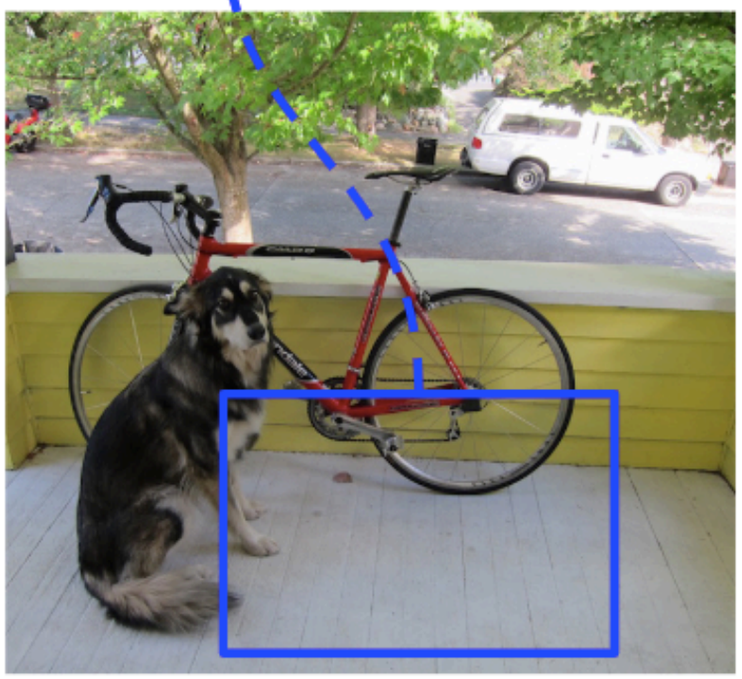
Go to next bbox_cur.

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

In this case: continue.

Non-Maximum Suppression: intuition

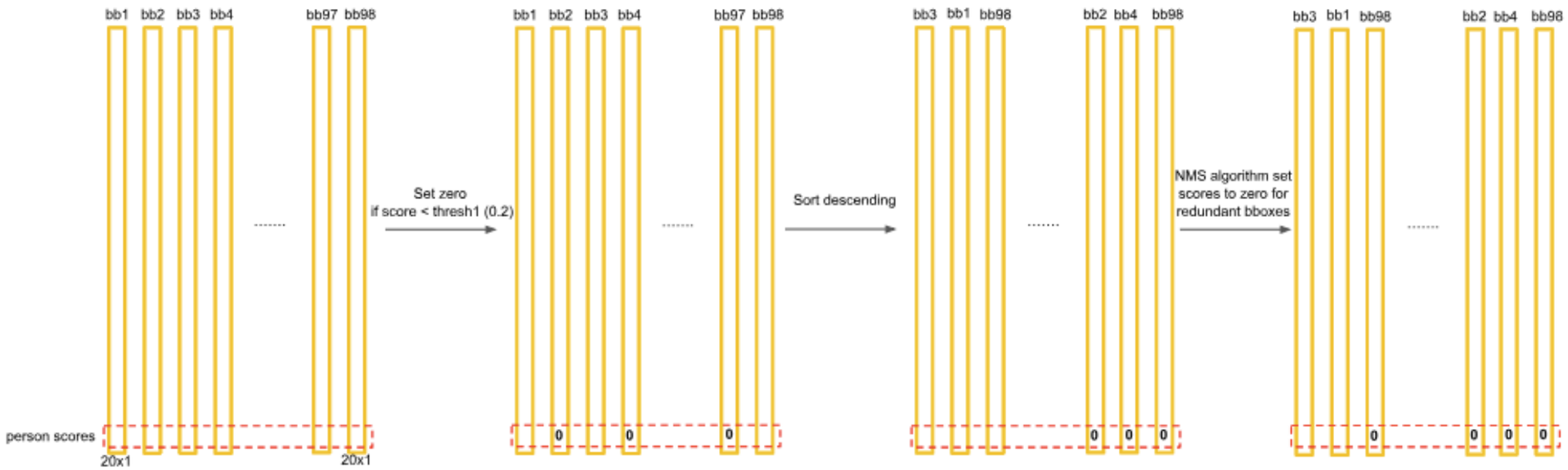




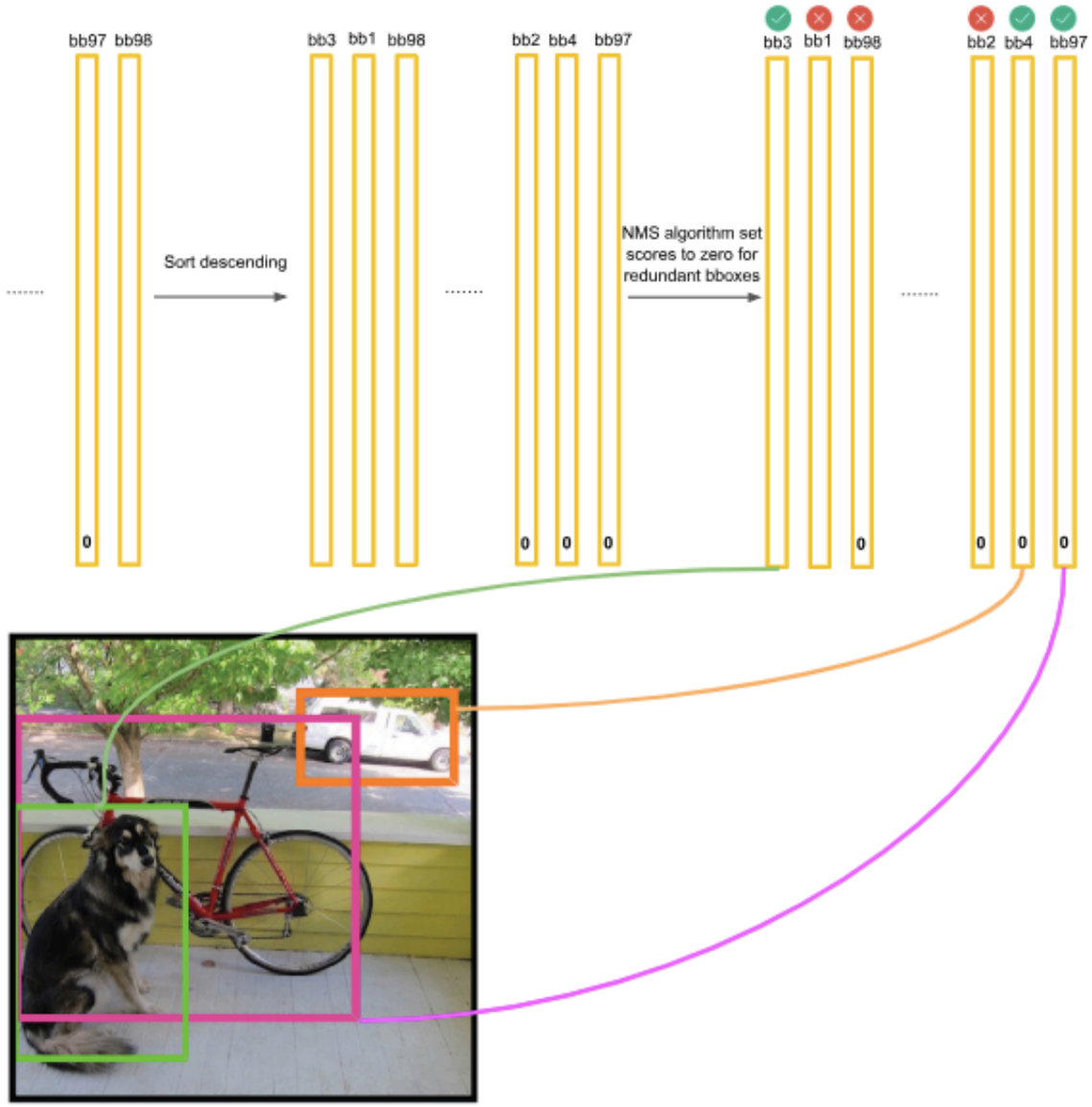
Go to next bbox with big score.
Let's denote it "bbox_max"

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 all classes


bb97 bb98
Sort descending
bb3 bb1 bb98
bb2 bb4 bb97
NMS algorithm set scores to zero for redundant bboxes
bb3 bb1 bb98
bb2 bb4 bb97
0

Error

- Uses sum-squared error
- Pros : easy to optimize
- Cons : doesn't perfectly align with our goal of maximizing average precision
- <ISSUE 1> It weights localization error equally with classification error
- <ISSUE 2> Many grid cells may not contain any object -> confidence value would be 0 -> gradient training problem (guess it means gradient vanishing)

Error

- <ISSUE 1> same weight issue (previously mentioned) : Our error metric should reflect that small deviations in large boxes matter less than in small boxes

- Solution : we predict the square root of the bounding box width and height instead of the width and height directly

Loss function

- Predictor : We want only one predicted bounding box to be responsible for each object as it predicts multiple bounding boxes. We want to focus on ‘highest-likely-to-contain-object-box’ and train weights to it to be more accurate.

- Among the multiple predicted bounding boxes, how do we determine PREDICTOR?

- One has highest IOU with the ground truth will be it.

Loss function

$\mathbb{1}_{ij}^{\text{obj}}$ jth bounding box is predictor, it's on cell i, it contains object

$\mathbb{1}_{ij}^{\text{noobj}}$ jth bounding box, it's on cell i, which doesn't contain any object

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

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

1 : (penalized) sum-squared error for coordinates

2 : (penalized) due to <ISSUE 2>, use square-root sum-squared error for box size

3 : loss of confidence score for box j, cell i

4 : (amplified) when it doesn't exist, amplify the error

5 : error of conditional class probability (multiplication of conditional class probability and object probability) (when it's correct = 1, otherwise = 0)