Machine Learning, Spring 2020

Deep Learning for Object Localization and Detection

Instructor: Prof. Yi Fang

yfang@nyu.edu

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

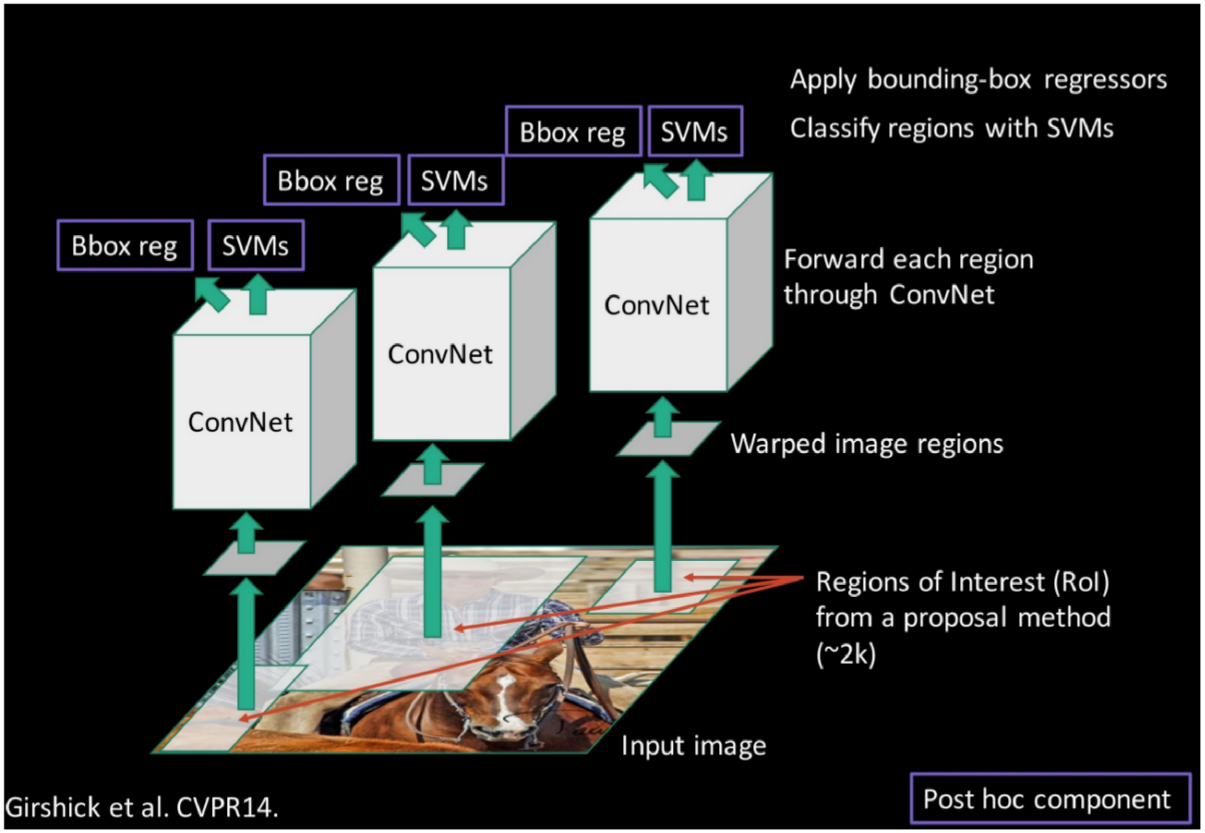
PyTorch tutorial: https://pytorch.org/tutorials/

Acknowledge: The slides are partially referred to the online materials by Taegyun Joen, https://www.slideshare.net/TaegyunJeon1/pr12-you-only-look-once-yolo-unified-realtime-object-detection and online YOLO paper and other materials (from ECS289g by Prof. Lee)

Recap

- R-CNN
- Fast-RCNN
- Faster-RCNN

Detection Algorithm: R-CNN



Girschick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014

Slide credit: Ross Girschick

Recap

Localization:

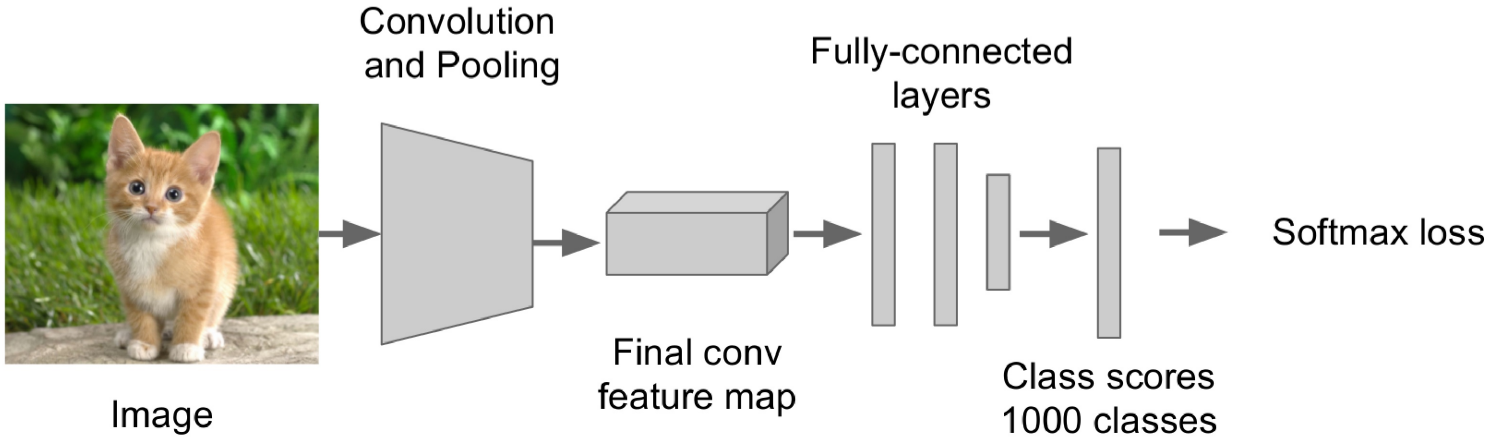
- Find a fixed number of objects (one or many)
- L2 regression from CNN features to box coordinates
- Much simpler than detection; consider it for your projects!
- Overfeat: Regression + efficient sliding window with FC -> conv conversion
- Deeper networks do better

Object Detection:

- Find a variable number of objects by classifying image regions
- Before CNNs: dense multiscale sliding window (HoG, DPM)
- Avoid dense sliding window with region proposals
- R-CNN: Selective Search + CNN classification / regression
- Fast R-CNN: Swap order of convolutions and region extraction
- Faster R-CNN: Compute region proposals within the network
- Deeper networks do better

R-CNN Training

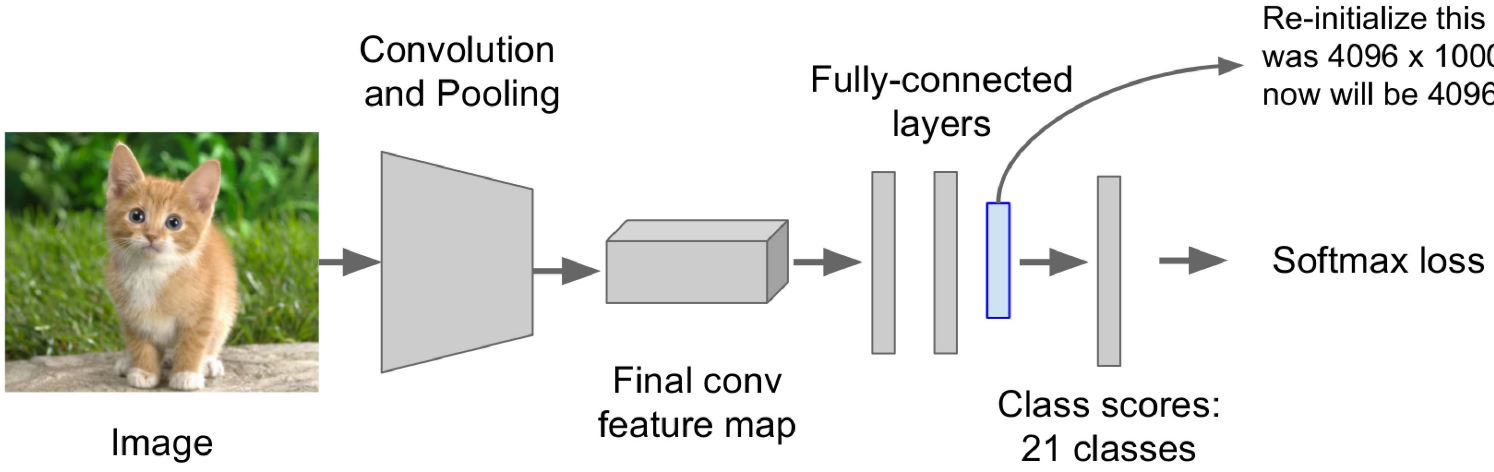
Step 1: Train (or download) a classification model for ImageNet (AlexNet)



R-CNN Training

Step 2: Fine-tune model for detection

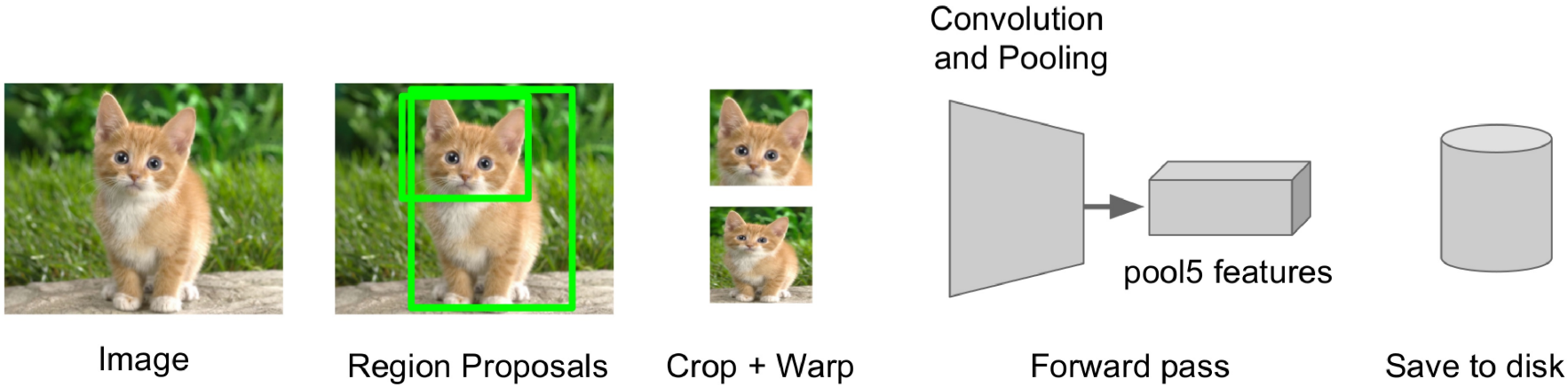
- Instead of 1000 ImageNet classes, want 20 object classes + background
- Throw away final fully-connected layer, reinitialize from scratch
- Keep training model using positive / negative regions from detection images



R-CNN Training

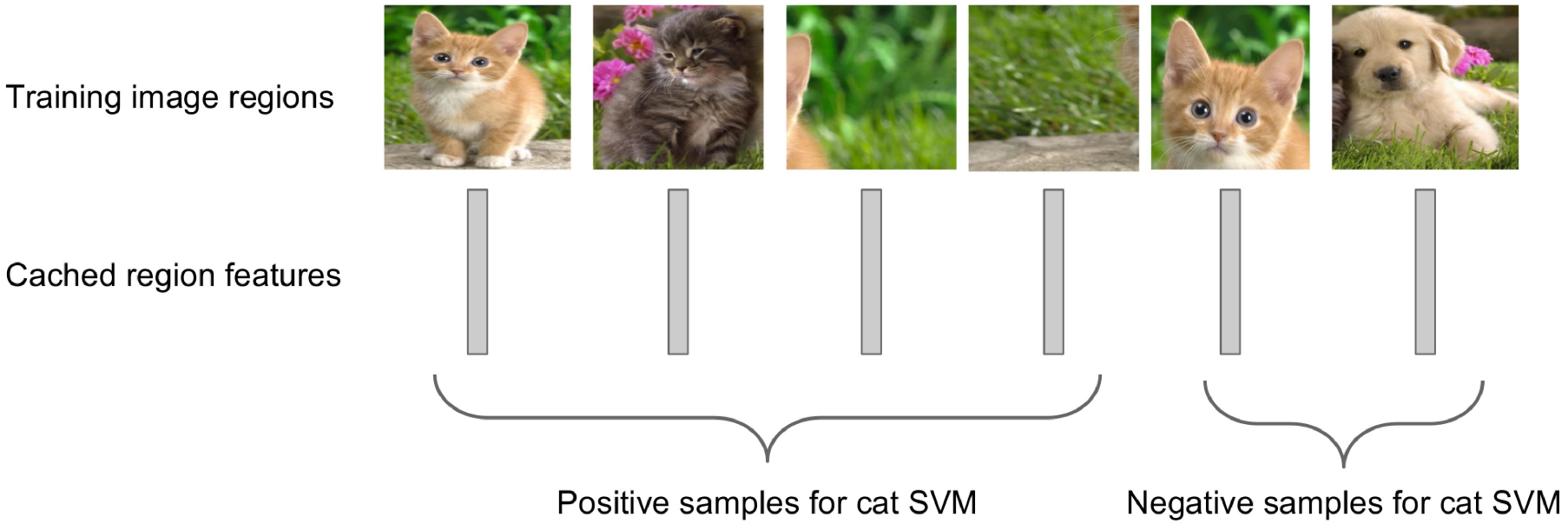
Step 3: Extract features

- Extract region proposals for all images
- For each region: warp to CNN input size, run forward through CNN, save pool5 features to disk
- Have a big hard drive: features are ~200GB for PASCAL dataset!



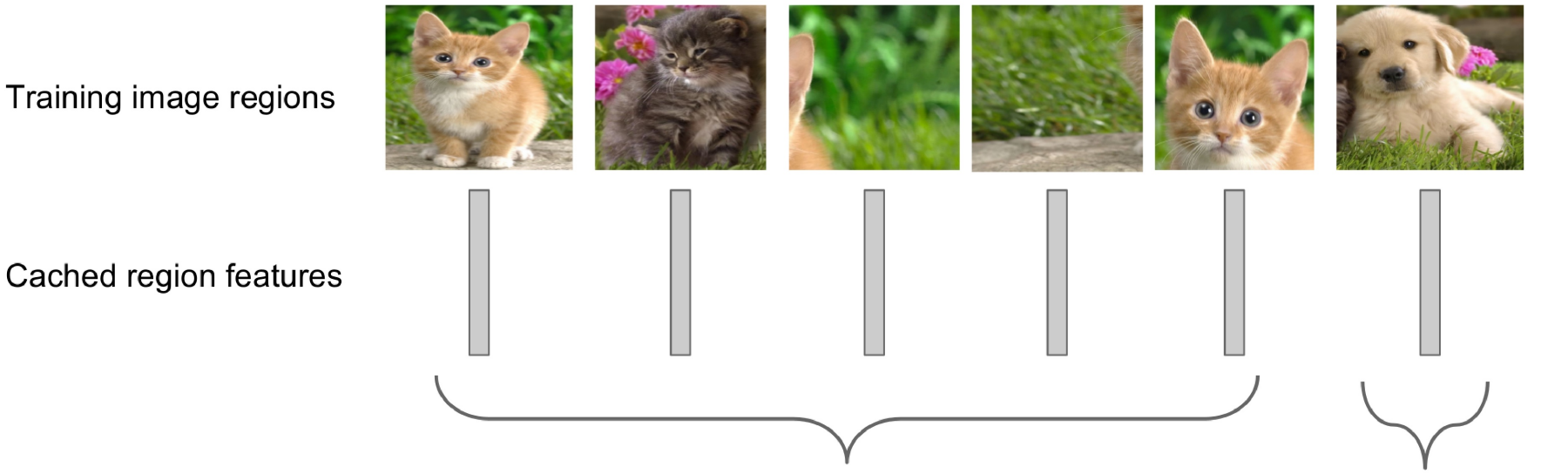
R-CNN Training

Step 4: Train one binary SVM per class to classify region features



R-CNN Training

Step 4: Train one binary SVM per class to classify region features



R-CNN Training

Step 5 (bbox regression): For each class, train a linear regression model to map from cached features to offsets to GT boxes to make up for "slightly wrong" proposals

Training image regions

Cached region features

Regression targets
(dx, dy, dw, dh)
Normalized coordinates

(0, 0, 0, 0)
Proposal is good

(.25, 0, 0, 0)
Proposal too far to left

(0, 0, -0.125, 0)
Proposal too wide

Object Detection: Datasets

	PASCAL VOC (2010)	ImageNet Detection (ILSVRC 2014)	MS-COCO (2014)
Number of classes	20	**200**	80
Number of images (train + val)	~20k	**~470k**	~120k
Mean objects per image	2.4	1.1	**7.2**

Object Detection: Evaluation

We use a metric called “mean average precision” (mAP)

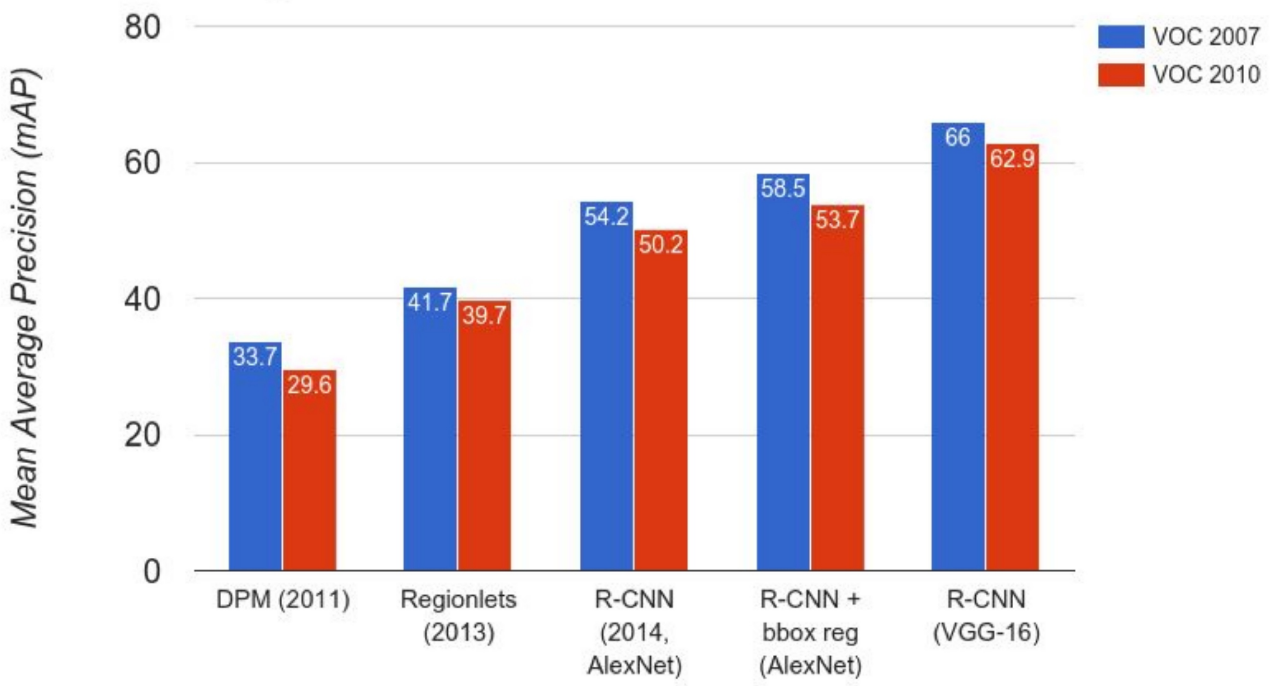
Compute average precision (AP) separately for each class, then average over classes

A detection is a true positive if it has IoU with a ground-truth box greater than some threshold (usually 0.5) (mAP@0.5)

Combine all detections from all test images to draw a precision / recall curve for each class; AP is area under the curve

TL;DR mAP is a number from 0 to 100; high is good

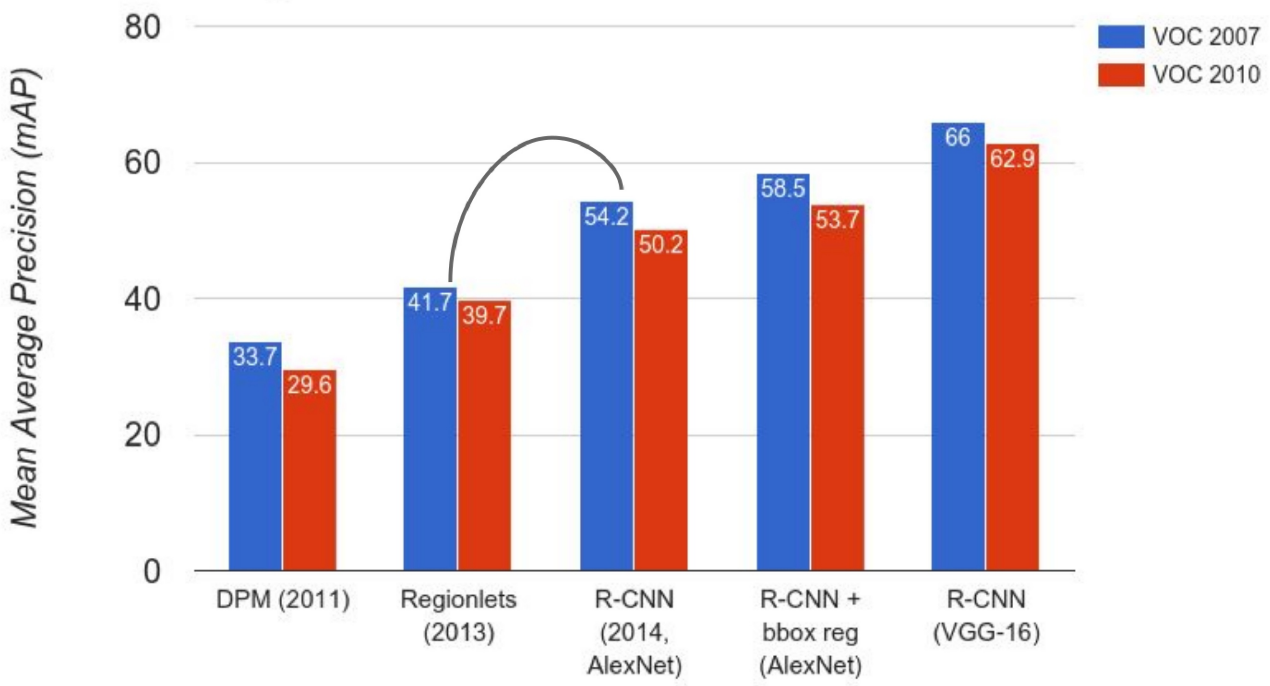
R-CNN Results



Wang et al, “Regionlets for Generic Object Detection”, ICCV 2013

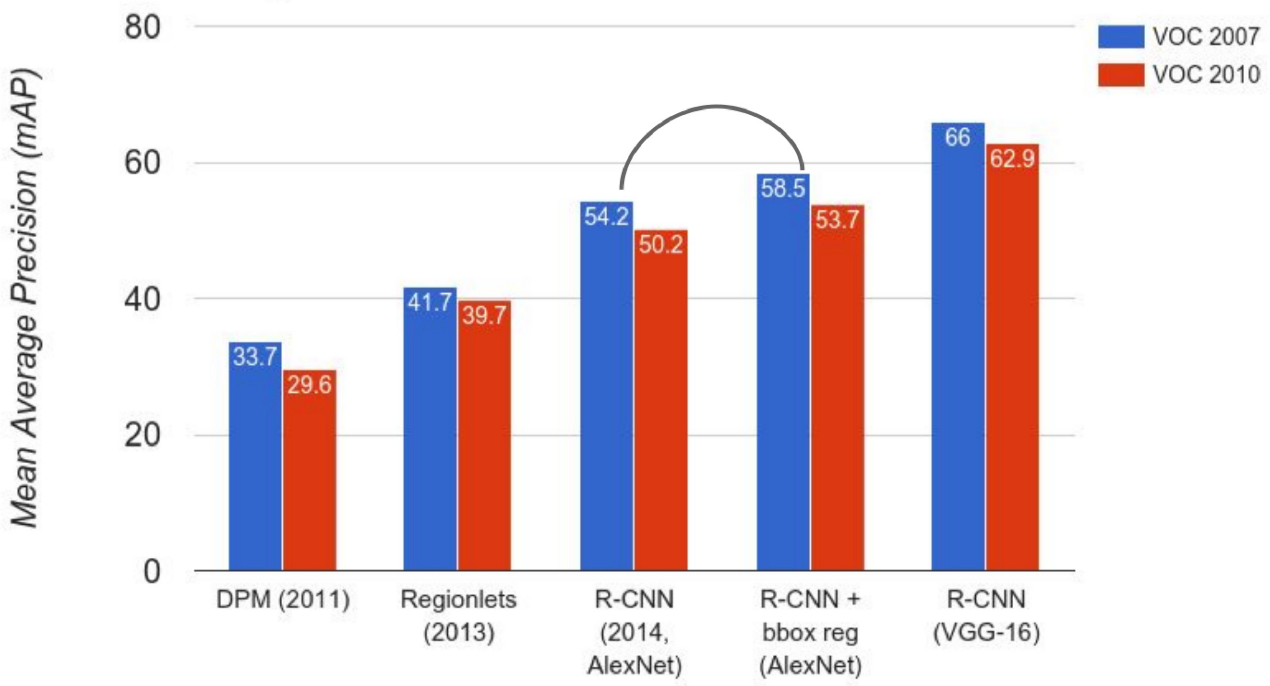
R-CNN Results

Big improvement compared to pre-CNN methods



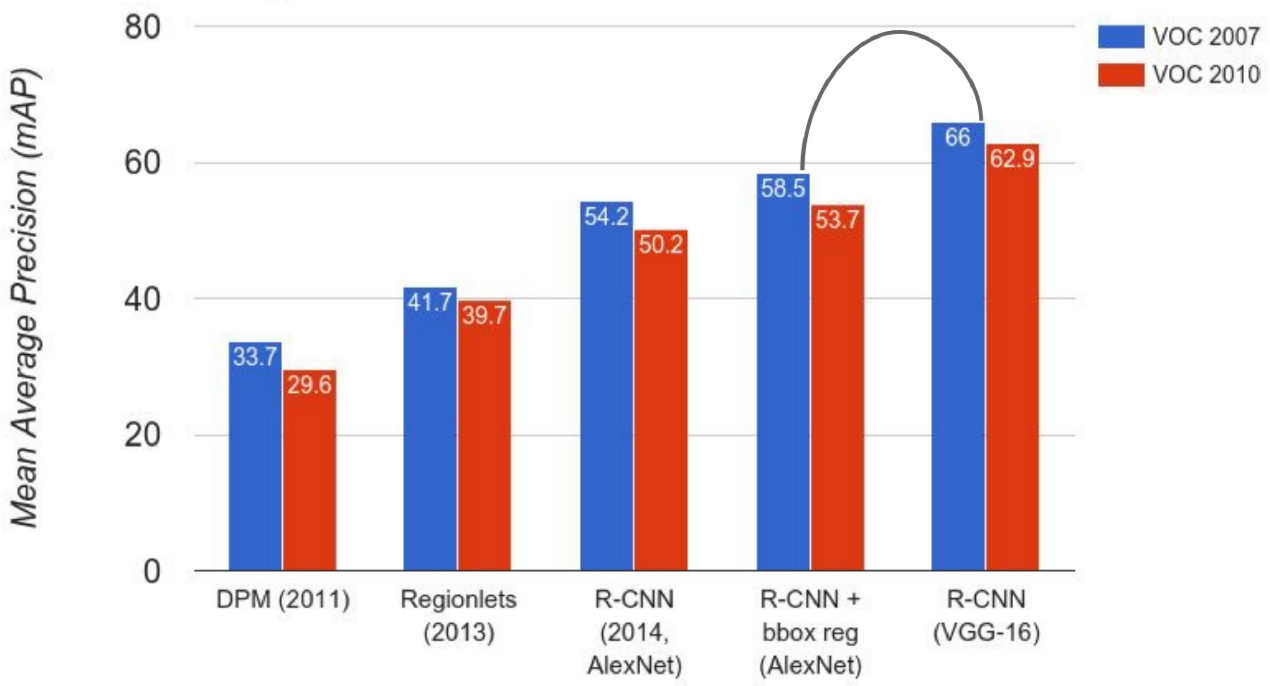
R-CNN Results

Bounding box regression helps a bit



R-CNN Results

Features from a deeper network help a lot

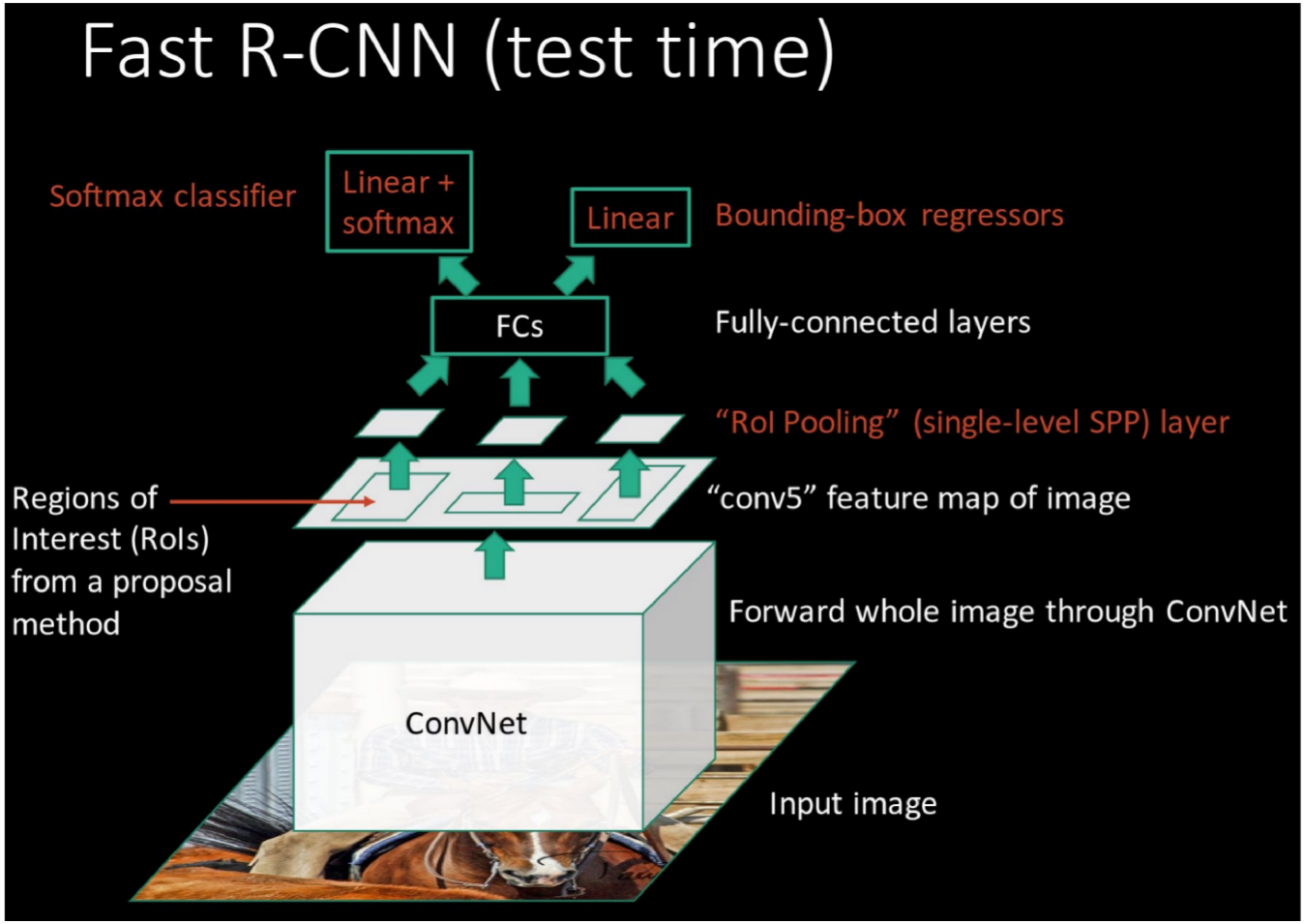


R-CNN Problems

1. Slow at test-time: need to run full forward pass of CNN for each region proposal

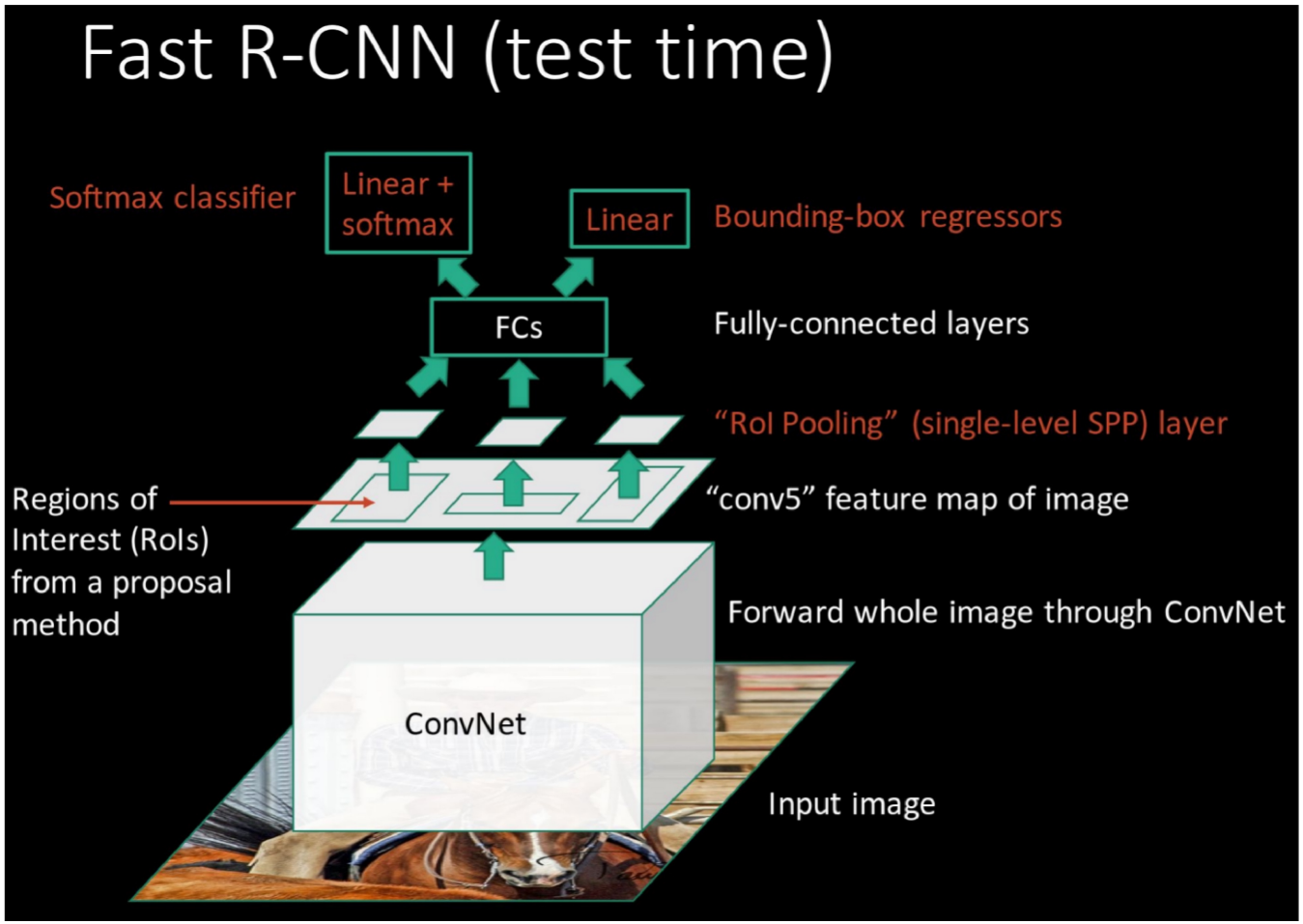
2. SVMs and regressors are post-hoc: CNN features not updated in response to SVMs and regressors

3. Complex multistage training pipeline



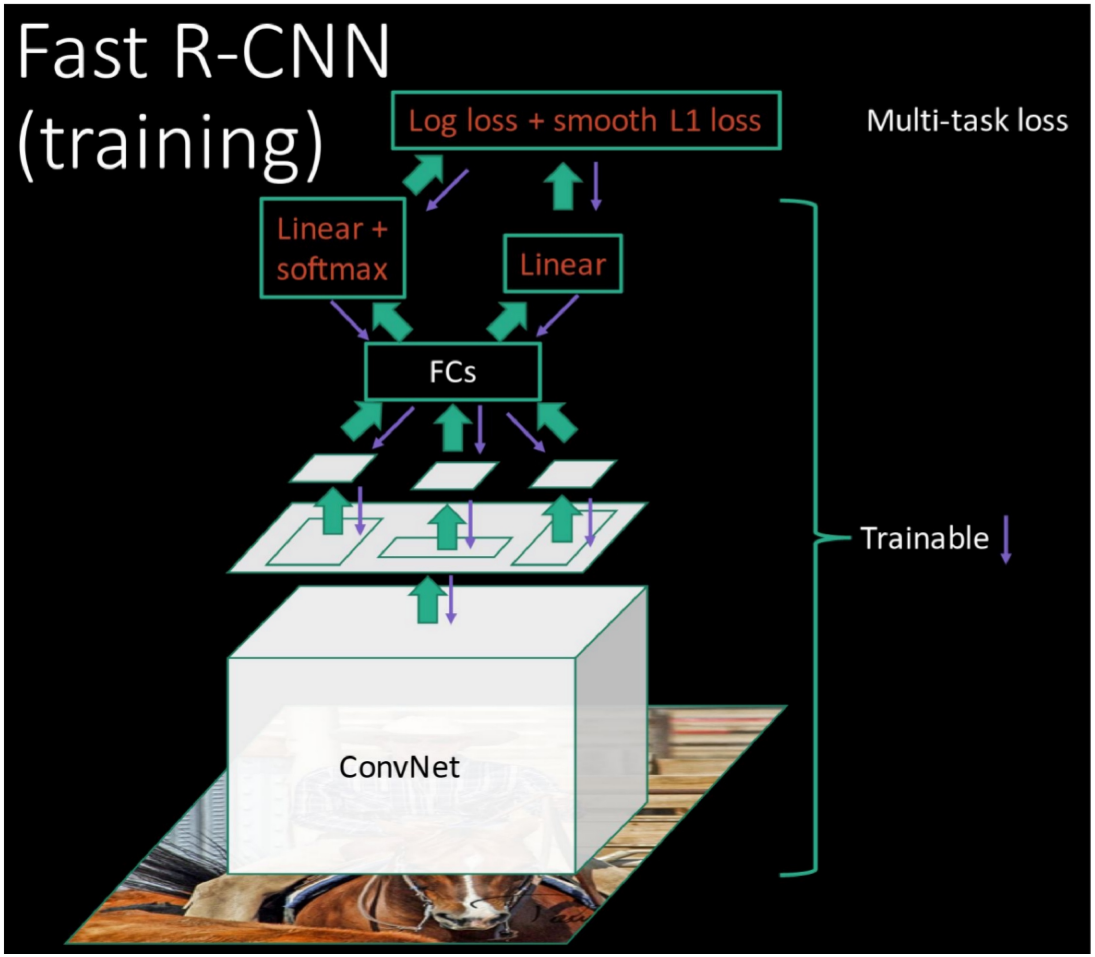
Girschick, "Fast R-CNN", ICCV 2015

Slide credit: Ross Girschick



R-CNN Problem #1: Slow at test-time due to independent forward passes of the CNN

Solution: Share computation of convolutional layers between proposals for an image



R-CNN Problem #2:
Post-hoc training: CNN not updated in response to final classifiers and regressors

R-CNN Problem #3:
Complex training pipeline

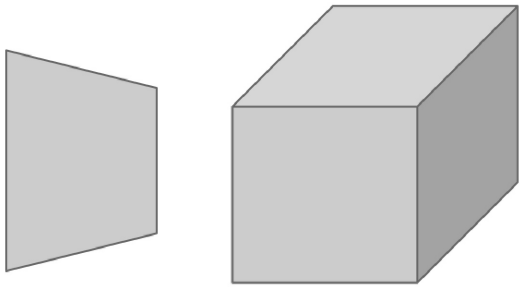
Solution:
Just train the whole system end-to-end all at once!

Slide credit: Ross Girschick

Fast R-CNN: Region of Interest Pooling

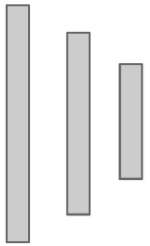
Hi-res input image:
3 x 800 x 600
with region
proposal

Convolution
and Pooling



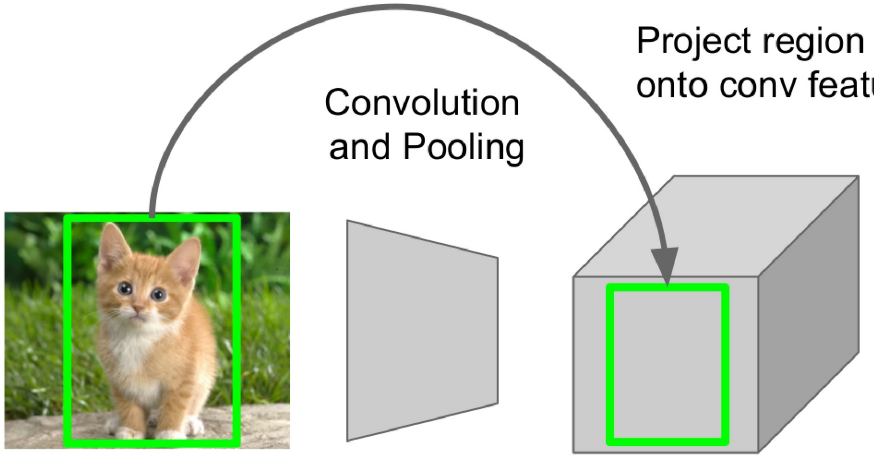
Hi-res conv features:
C x H x W
with region proposal

Fully-connected
layers



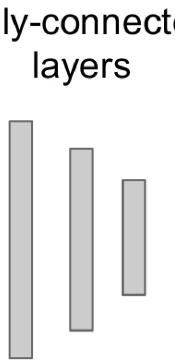
Problem: Fully-connected layers expect low-res conv features: C x h x w

Fast R-CNN: Region of Interest Pooling



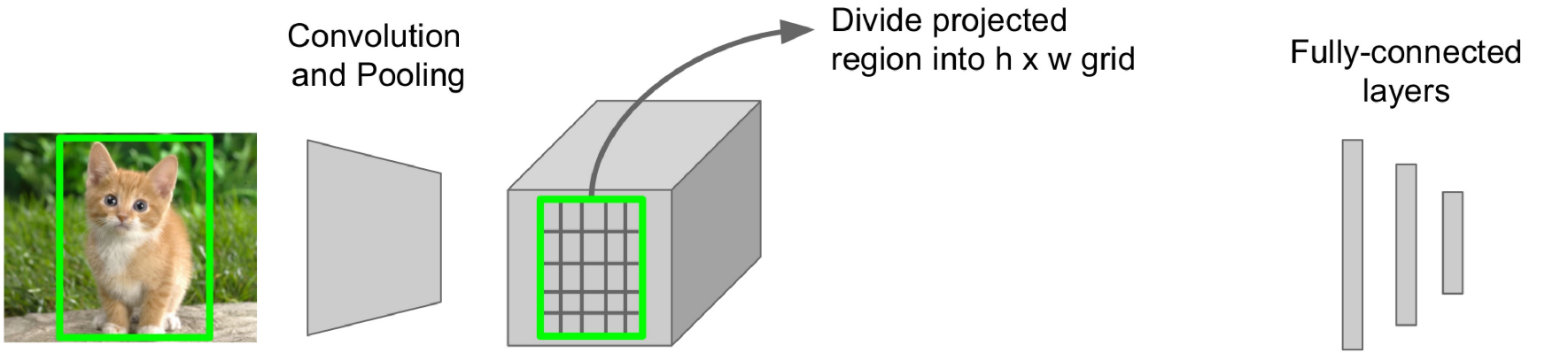
Hi-res input image:
3 x 800 x 600
with region
proposal

Hi-res conv features:
C x H x W
with region proposal

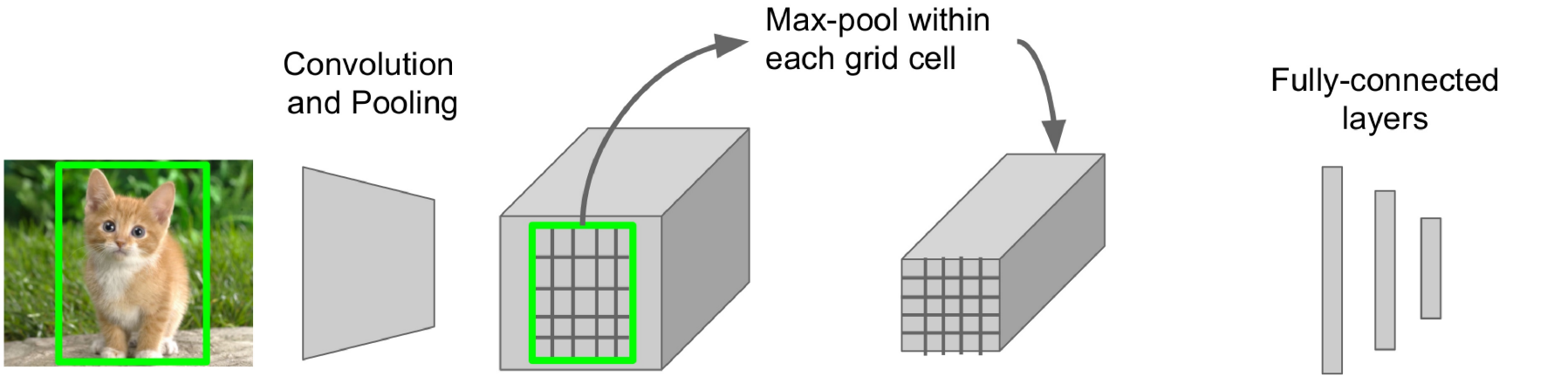


Problem: Fully-connected layers expect low-res conv features: C x h x w

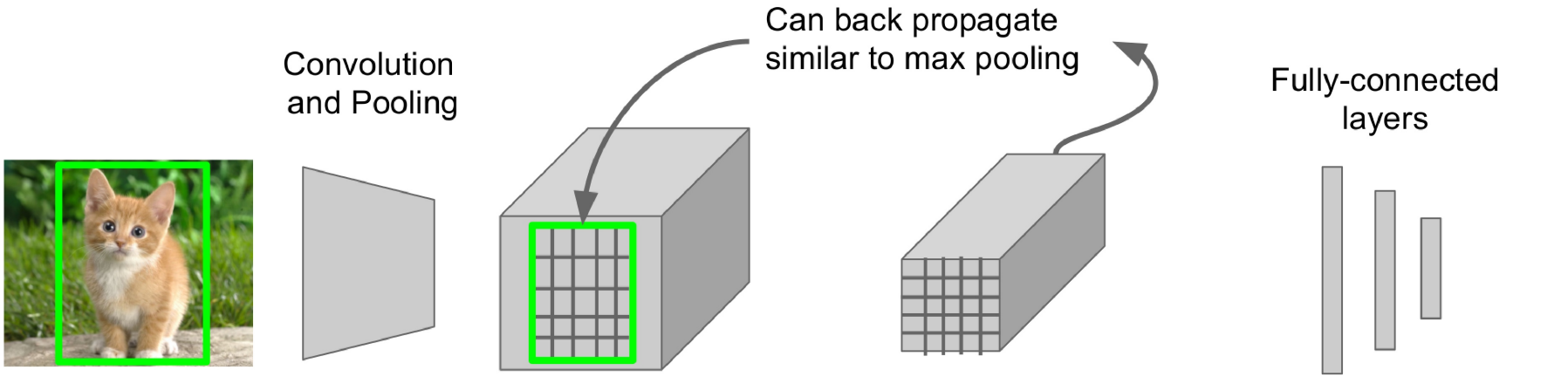
Fast R-CNN: Region of Interest Pooling



Fast R-CNN: Region of Interest Pooling



Fast R-CNN: Region of Interest Pooling



Fast R-CNN Results

Faster!

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**

Using VGG-16 CNN on Pascal VOC 2007 dataset

Fast R-CNN Results

Faster!

FASTER!

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**

Using VGG-16 CNN on Pascal VOC 2007 dataset

Fast R-CNN Results

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
mAP (VOC 2007)	66.0	**66.9**

Faster!

FASTER!

Better!

Using VGG-16 CNN on Pascal VOC 2007 dataset

Fast R-CNN Problem:

Test-time speeds don't include region proposals

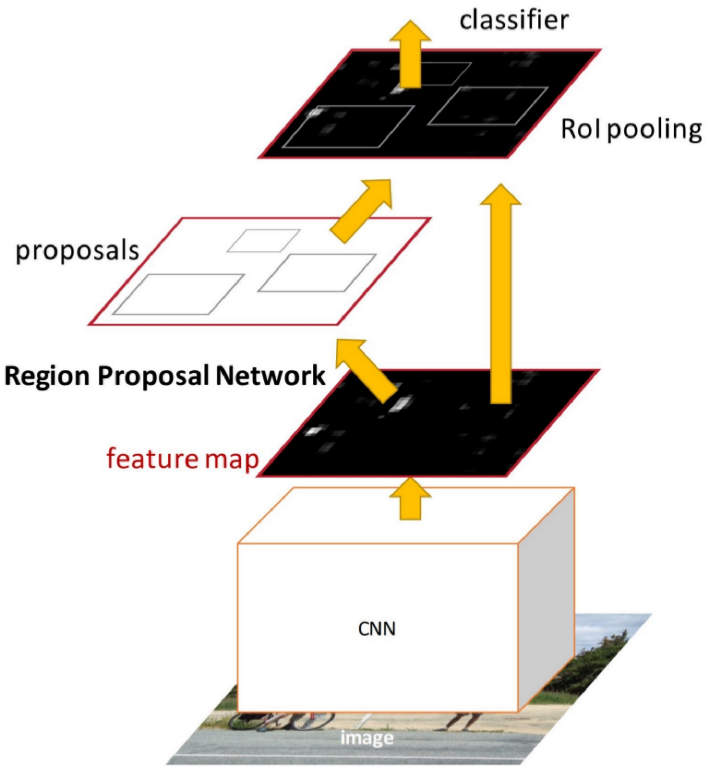
	R-CNN	**Fast R-CNN**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
Test time per image with Selective Search	50 seconds	**2 seconds**
(Speedup)	1x	**25x**

Fast R-CNN ~~Problem~~ Solution:

Test-time speeds don't include region proposals
Just make the CNN do region proposals too!

	R-CNN	**Fast R-CNN**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
Test time per image with Selective Search	50 seconds	**2 seconds**
(Speedup)	1x	**25x**

Faster R-CNN:



Insert a **Region Proposal Network (RPN)** after the last convolutional layer

RPN trained to produce region proposals directly; no need for external region proposals!

After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN

Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Slide credit: Ross Girschick

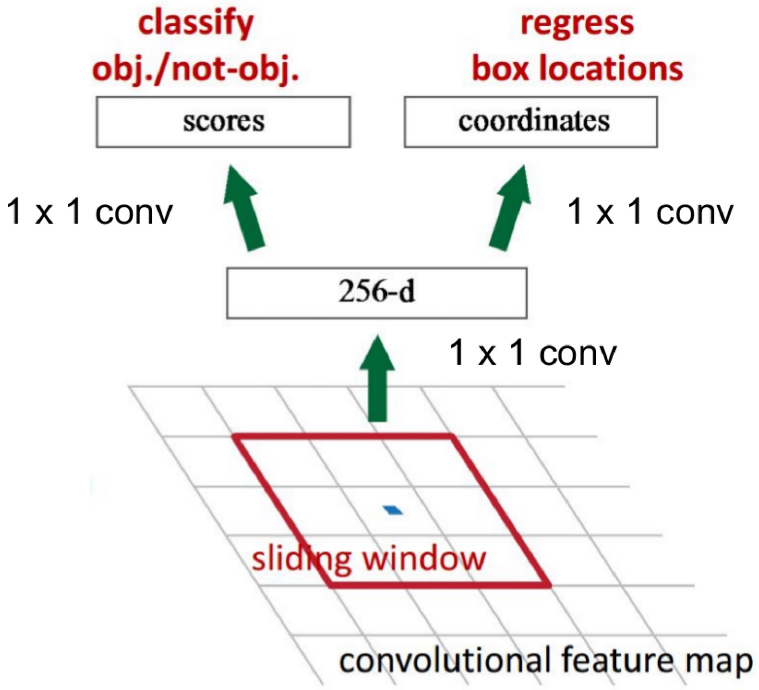
Faster R-CNN: Region Proposal Network

Slide a small window on the feature map

Build a small network for:
- classifying object or not-object, and
- regressing bbox locations

Position of the sliding window provides localization information with reference to the image

Box regression provides finer localization information with reference to this sliding window



Slide credit: Kaiming He

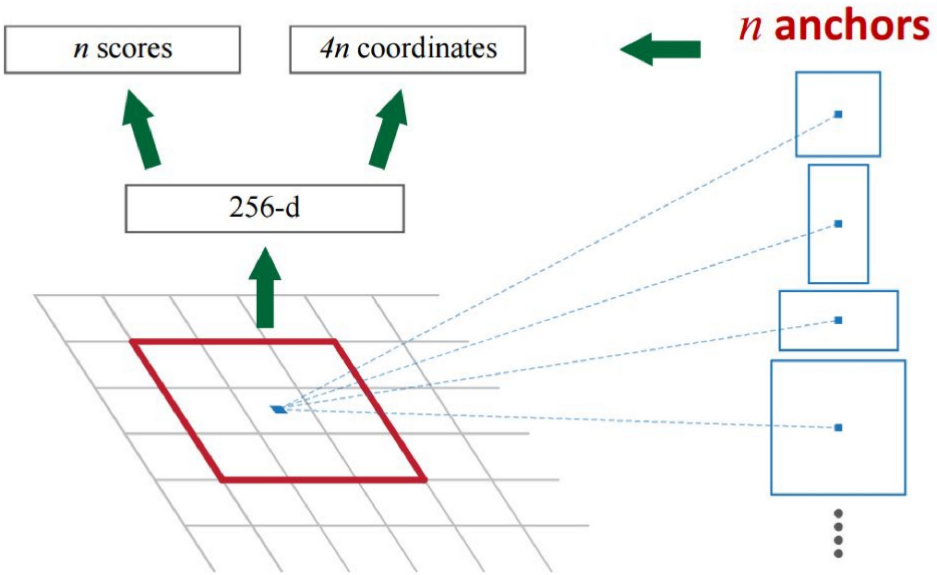
Faster R-CNN: Region Proposal Network

Use **N anchor boxes** at each location

Anchors are **translation invariant**: use the same ones at every location

Regression gives offsets from anchor boxes

Classification gives the probability that each (regressed) anchor shows an object



Faster R-CNN: Training

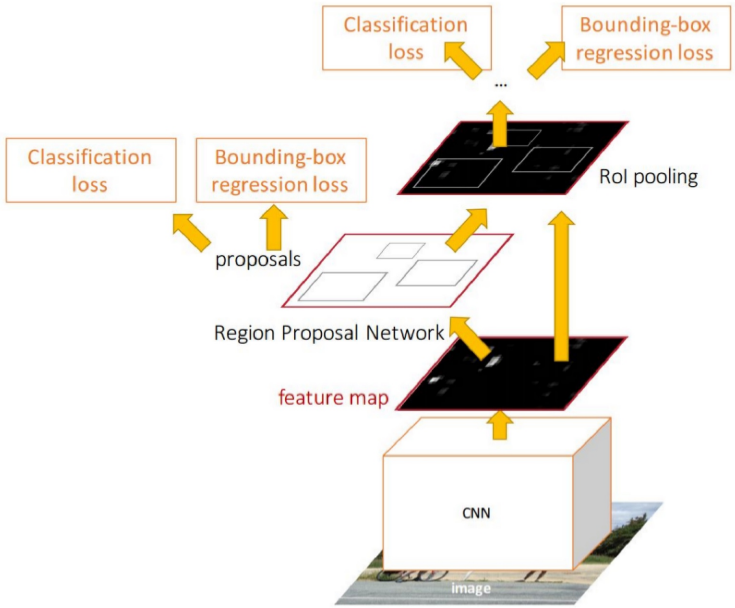
In the paper: Ugly pipeline

- Use alternating optimization to train RPN, then Fast R-CNN with RPN proposals, etc.
- More complex than it has to be

Since publication: Joint training!

One network, four losses

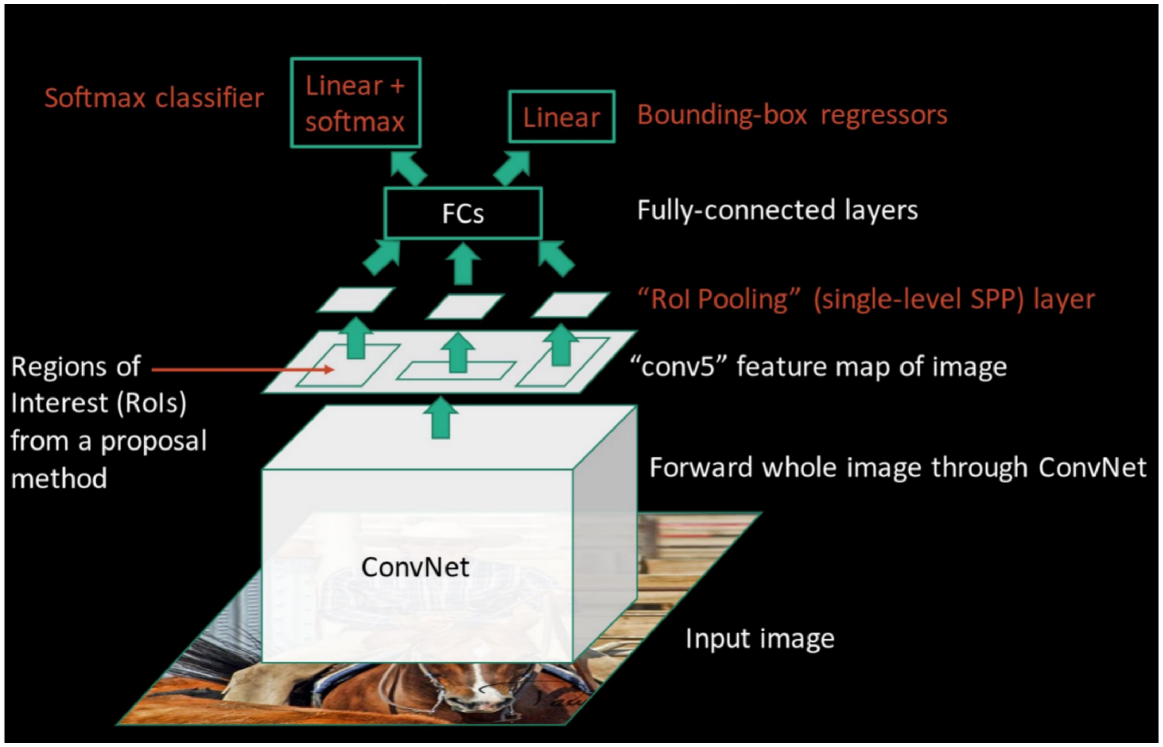
- RPN classification (anchor good / bad)
- RPN regression (anchor -> proposal)
- Fast R-CNN classification (over classes)
- Fast R-CNN regression (proposal -> box)



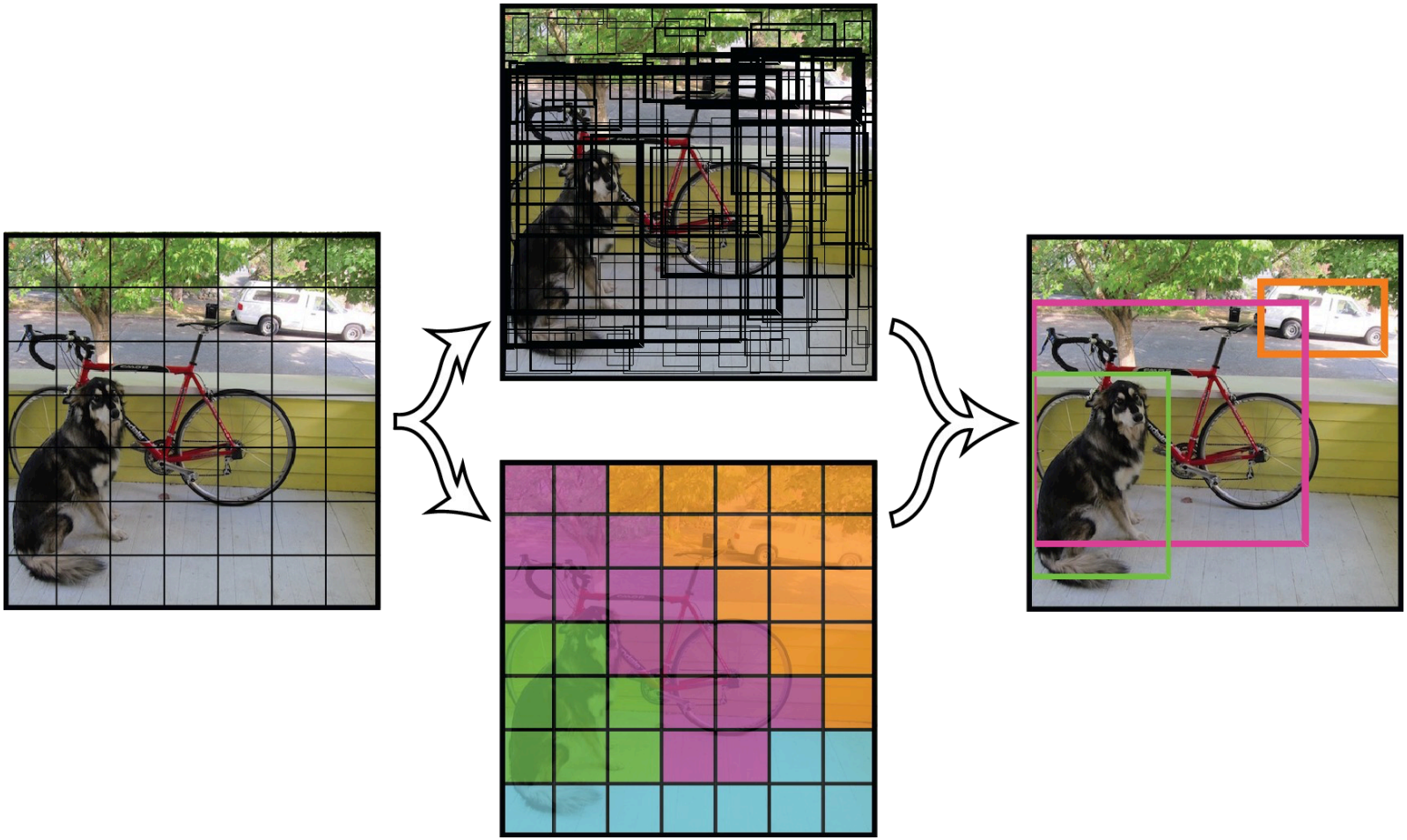
Slide credit: Ross Girschick

R-CNN Based Methods

- Two stages methods



Detection Algorithm: YOLO
You Only Look Once



Object Detection as Regression Problem

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

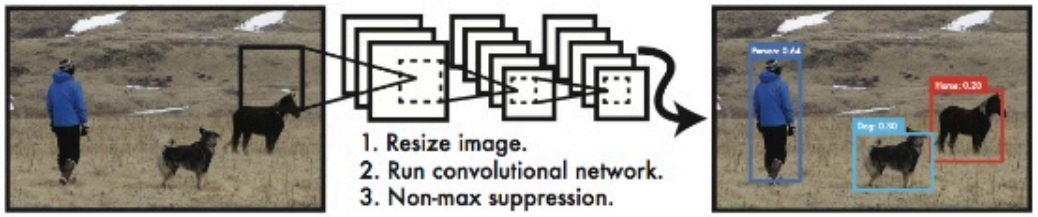


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

Appendix: Intersection over Union (IOU)

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

$\Pr(\text{Class}_i|\text{Object})$

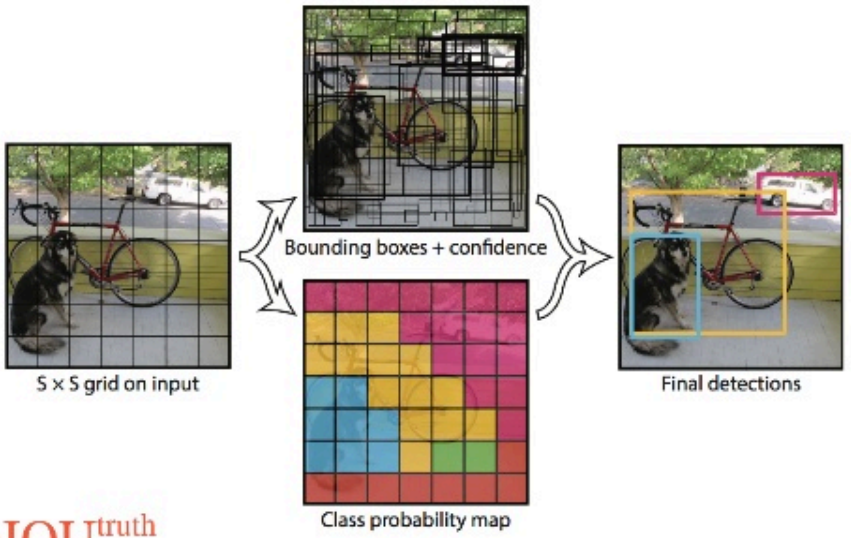


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

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

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

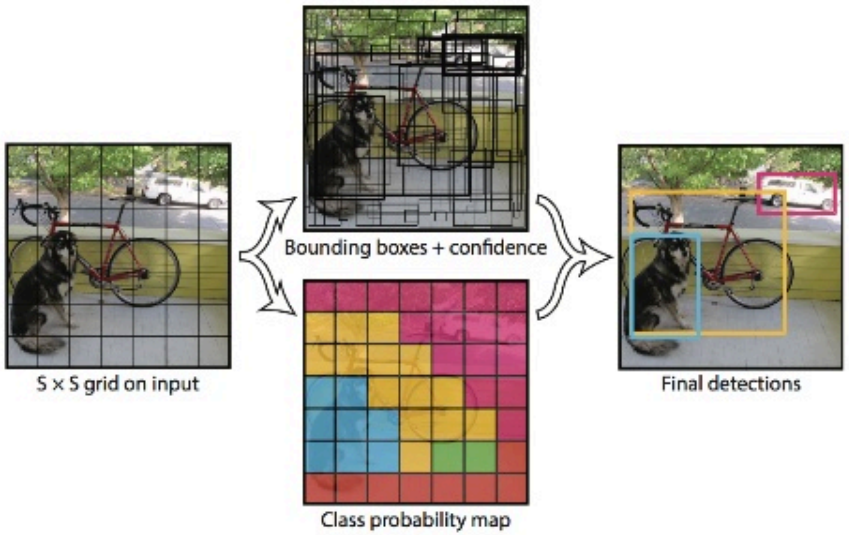


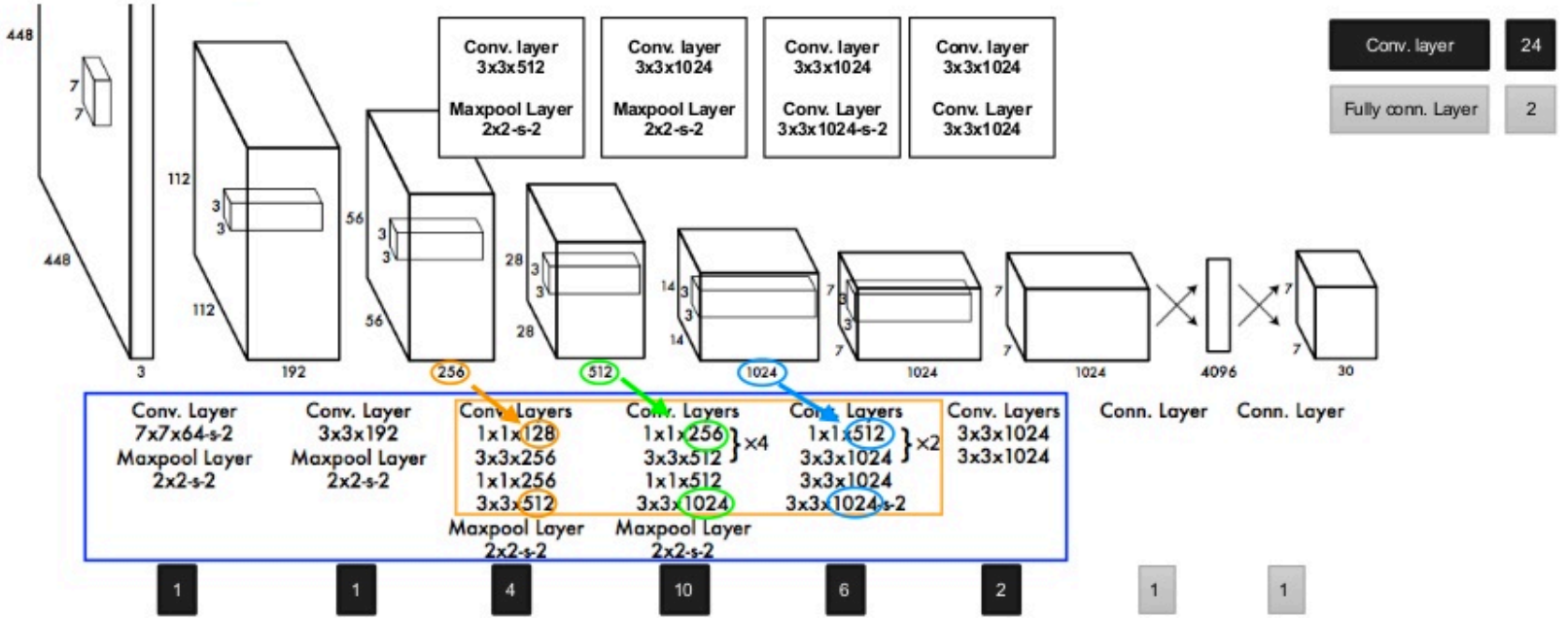
Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

Appendix: GoogLeNet

Network Design: YOLO

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

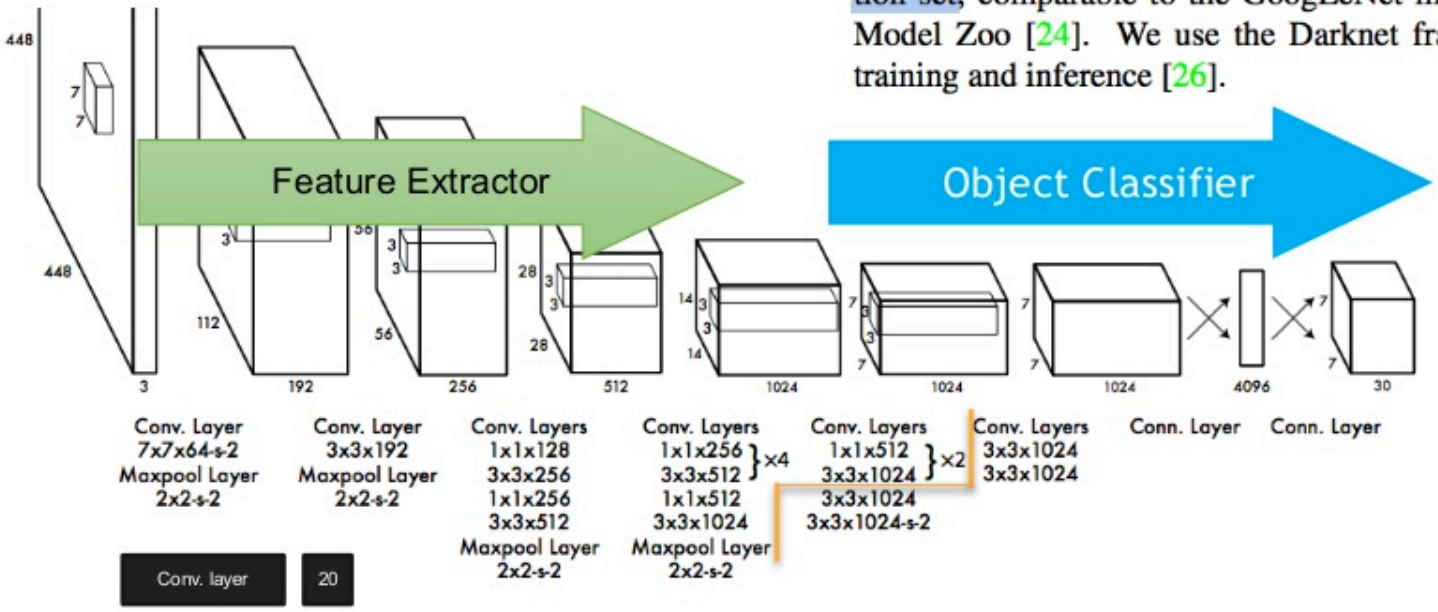
Our network architecture is inspired by the GoogLeNet model for image classification [34]. Our network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by GoogLeNet, we simply use 1×1 reduction layers followed by 3×3 convolutional layers, similar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

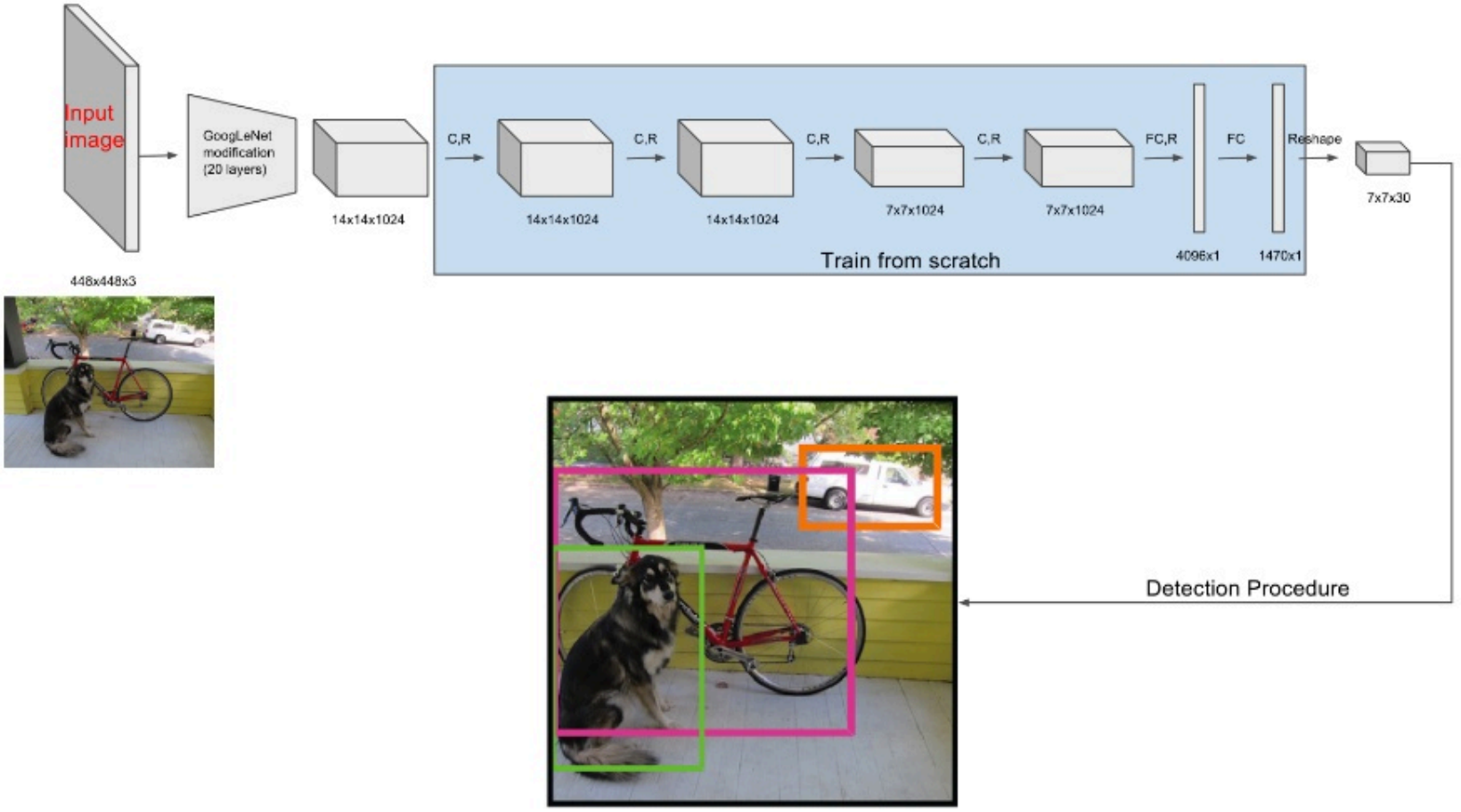


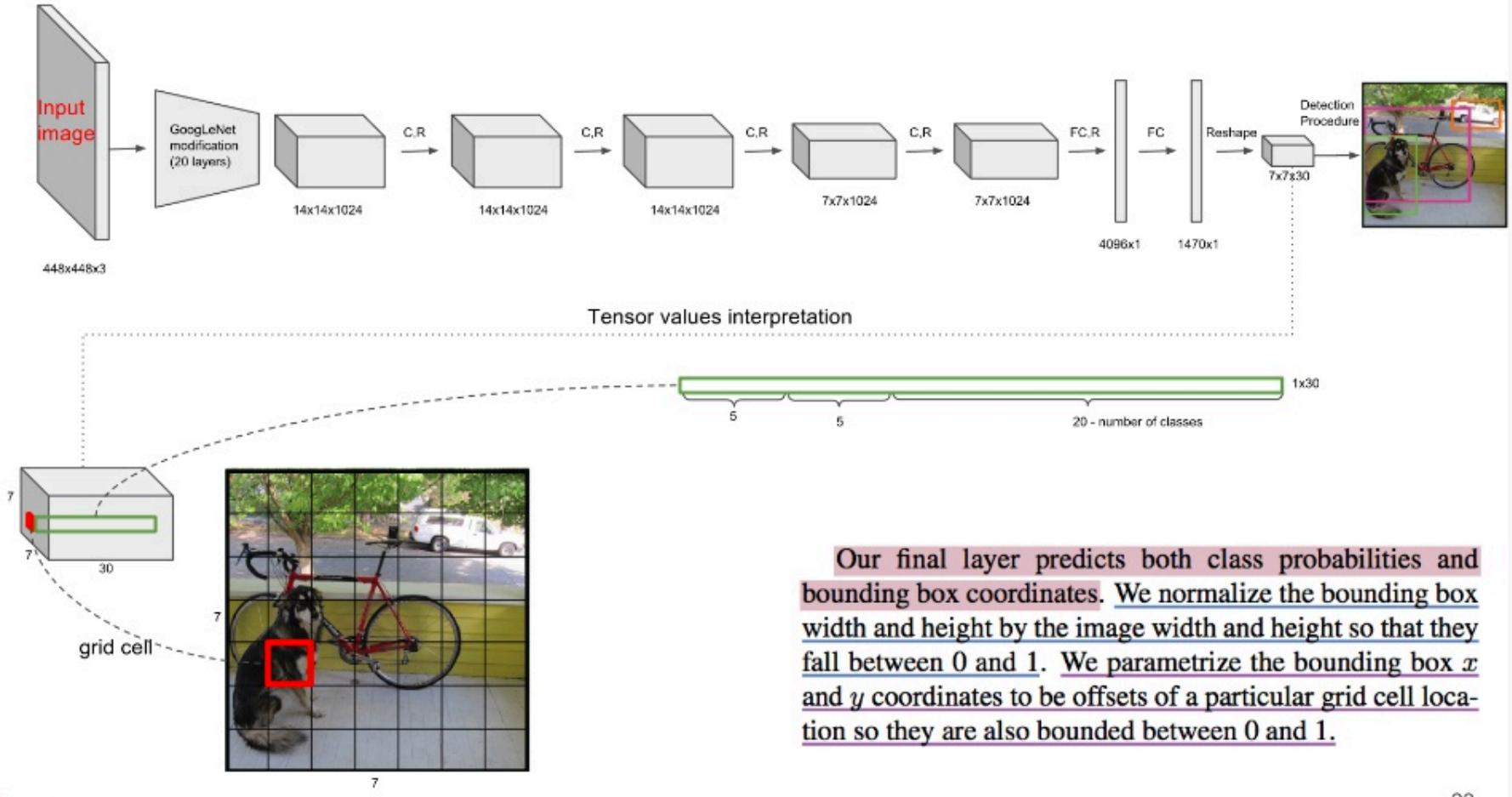
Training

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].

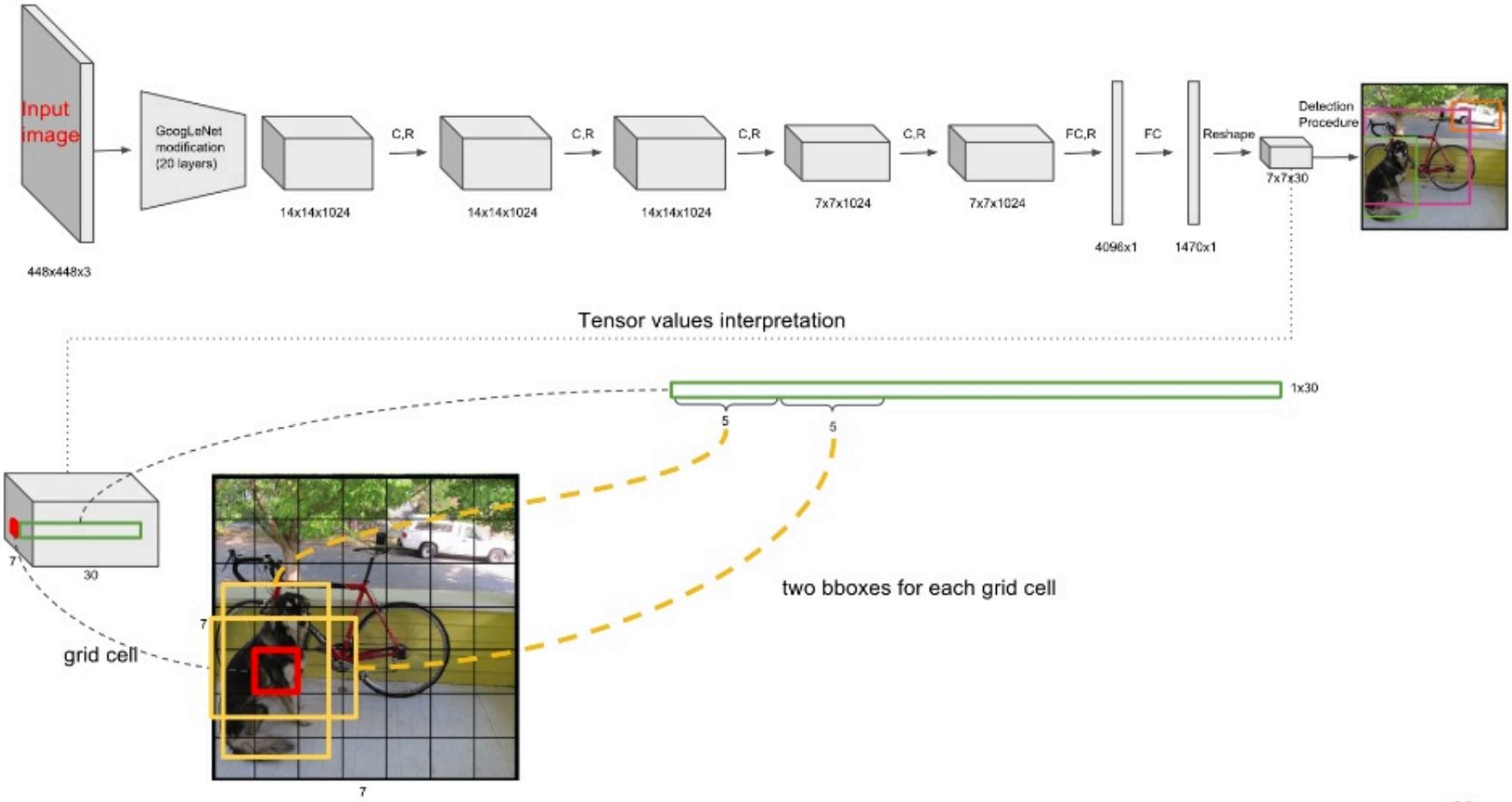


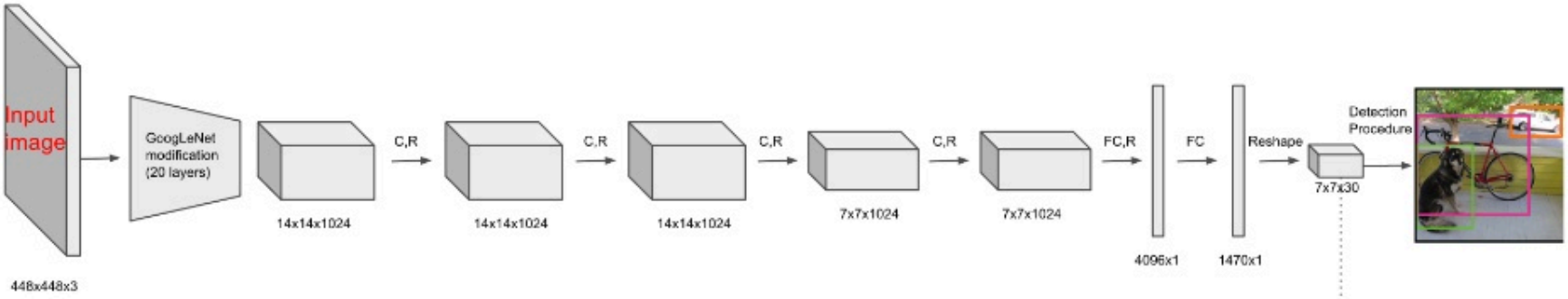

Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Train from scratch
Detection Procedure

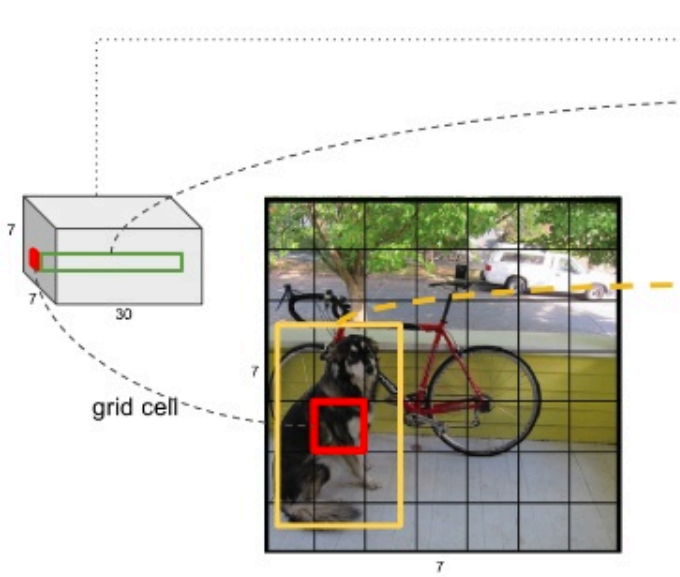


Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they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

deepsystems.io


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
two bboxes for each grid cell
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)

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.

deepsystems.io

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, λ_{coord} and λ_{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:

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

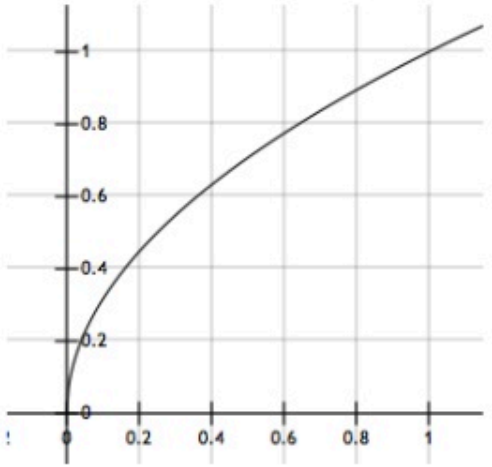
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

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

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

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

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:

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

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

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

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

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

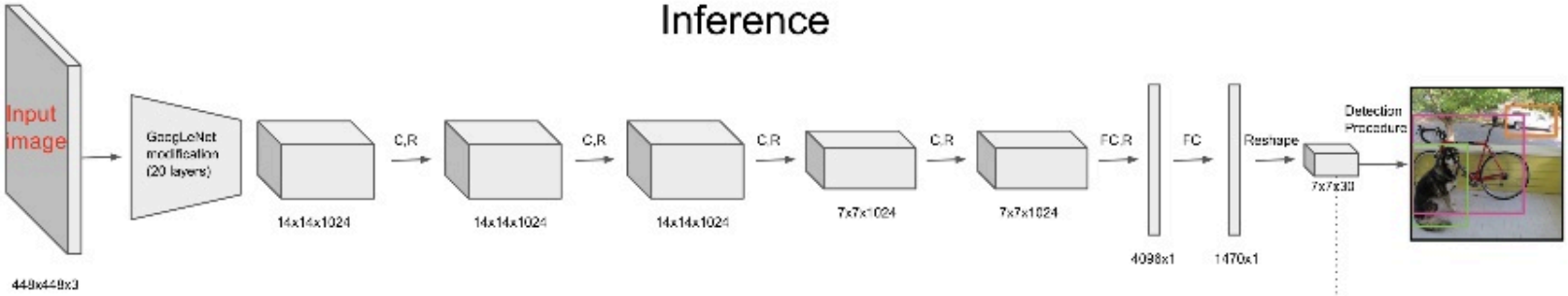
$\mathbb{1}_{ij}^{obj}$ **The *j*th bbox predictor** in ***cell i*** is "responsible" for that prediction

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

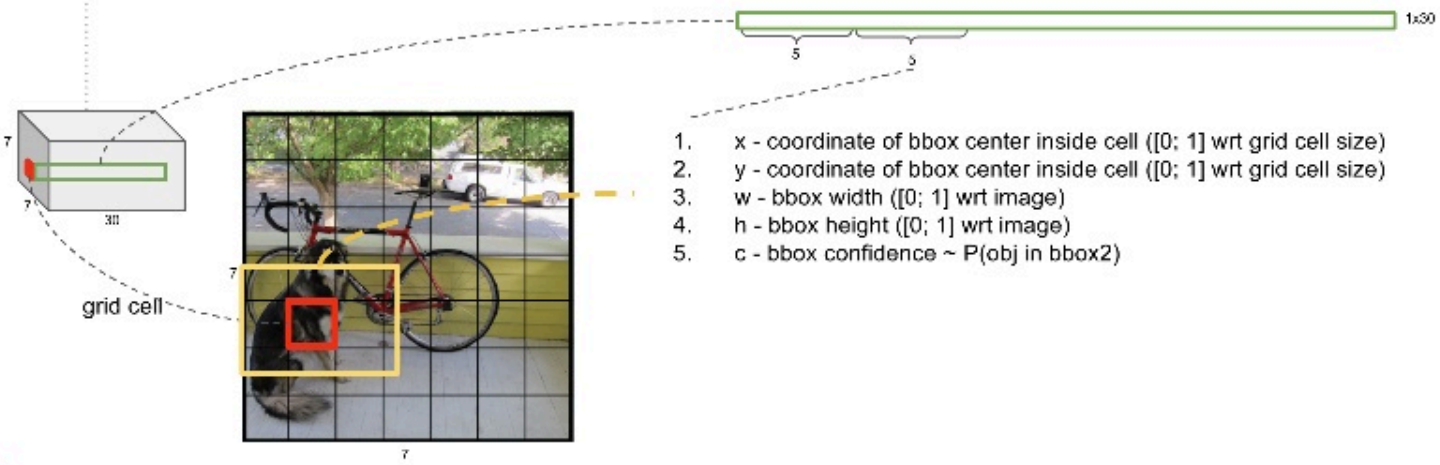
$\mathbb{1}_{i}^{obj}$ If object appears in ***cell i***

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).

Inference

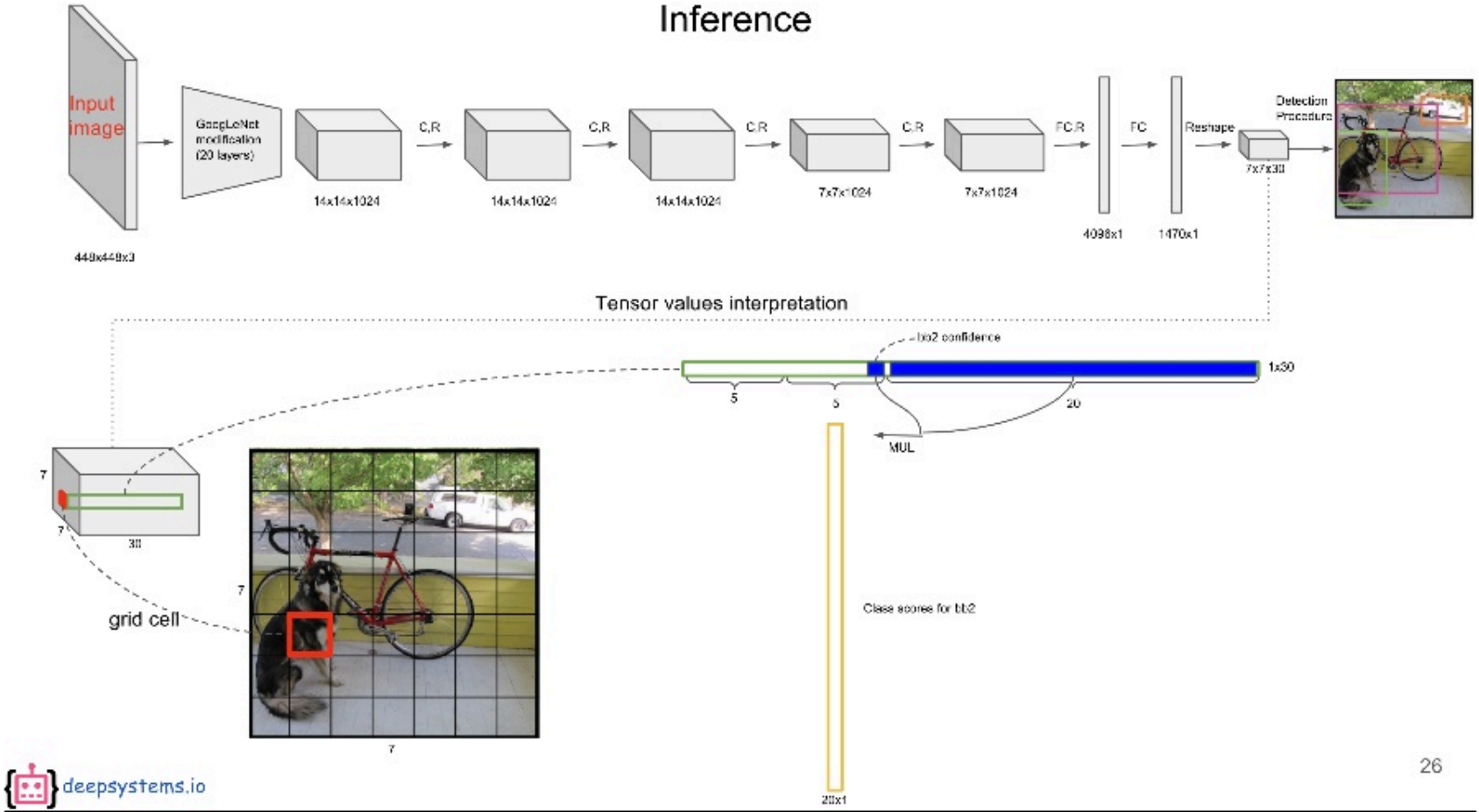


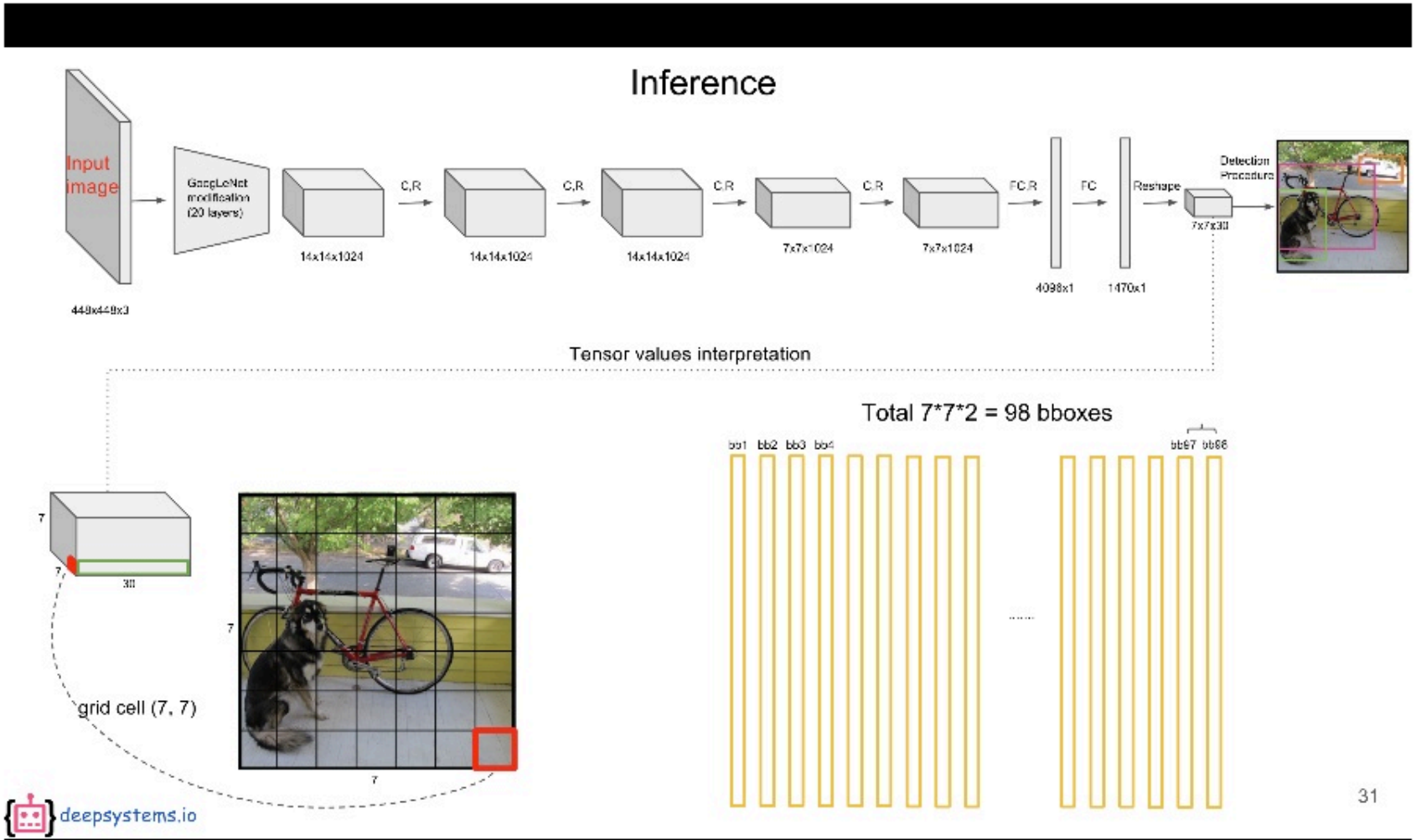
Tensor values interpretation



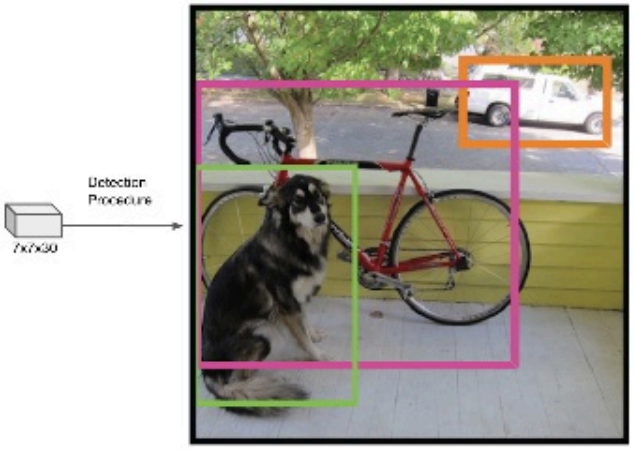
1. x - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
2. y - coordinate of bbox center inside cell ([0; 1] wrt grid cell size)
3. w - bbox width ([0; 1] wrt image)
4. h - bbox height ([0; 1] wrt image)
5. c - bbox confidence ~ P(obj in bbox2)

deepsystems.io

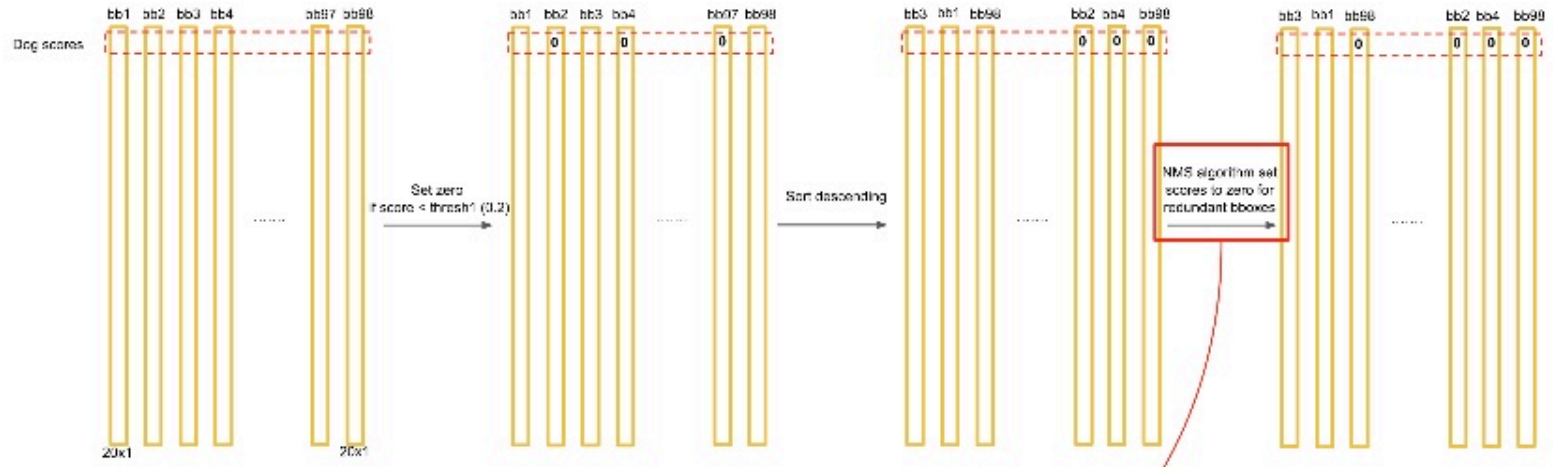

Inference
Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30
Detection Procedure
Tensor values interpretation
bb2 confidence
1x30
5
5
20
MUL
7
7
30
grid cell
7
7
Class scores for bb2
20x1
deepsystems.io
26


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
Total 7*7*2 = 98 bboxes
bb1 bb2 bb3 bb4
bb97 bb98
7
7
30
7
7
grid cell (7, 7)
deepsystems.io

Look at detection procedure



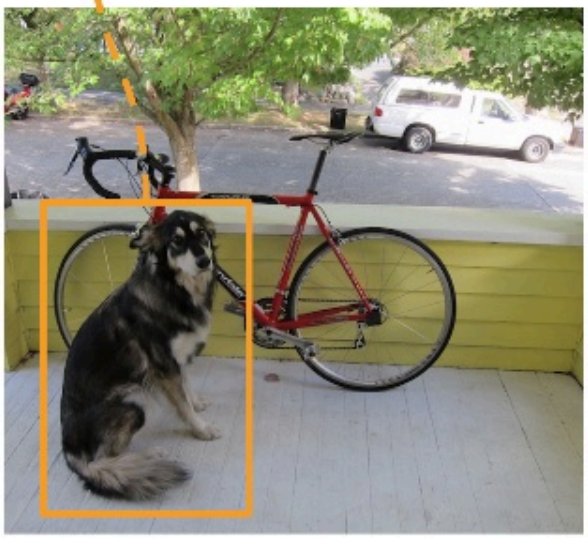
deepsystems.io


bb1 bb2 bb3 bb4
bb97 bb98
Dog scores
20x1
20x1
Set zero
if score < thresh1 (0.2)
bb1 bb2 bb3 bb4
bb97 bb98
Sort descending
bb3 bb1 bb98
bb2 bb4 bb99
NMS algorithm set scores to zero for redundant bboxes
bb3 bb1 bb98
bb2 bb4 bb99

How it works

Non-Maximum Suppression

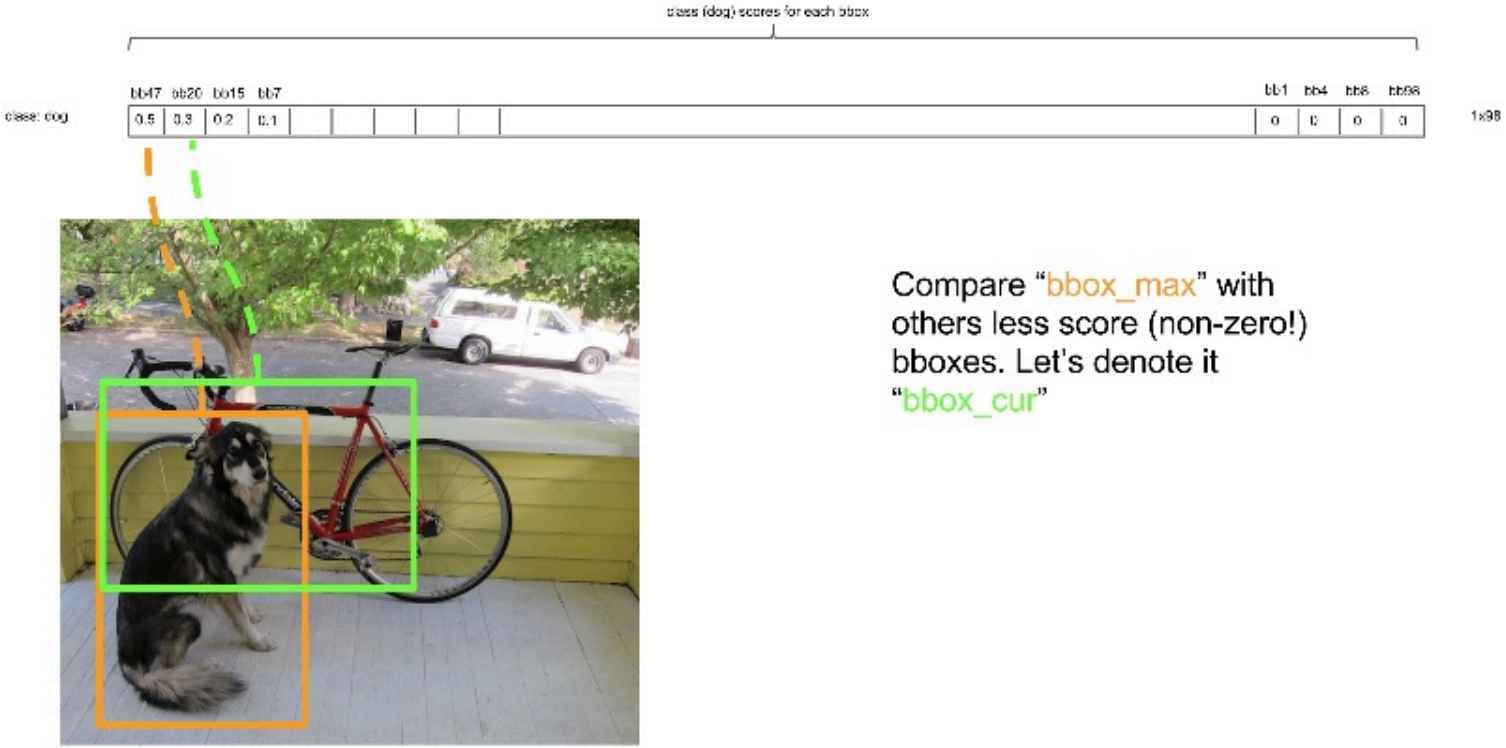
Non-Maximum Suppression: intuition



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

deepsystems.io

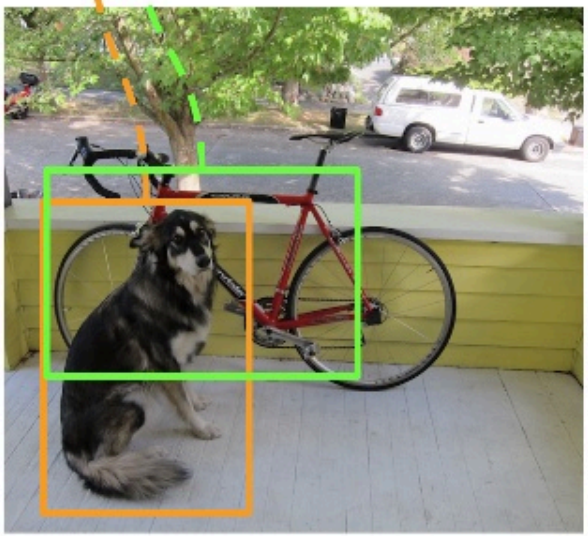
Non-Maximum Suppression: intuition



Compare "bbox_max" with others less score (non-zero!) bboxes. Let's denote it "bbox_cur"

deepsystems.io

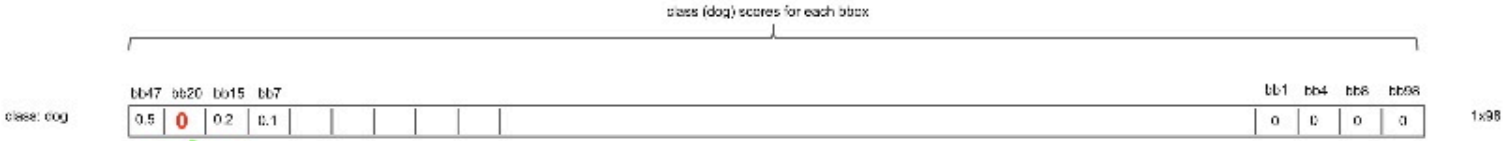
Non-Maximum Suppression: intuition

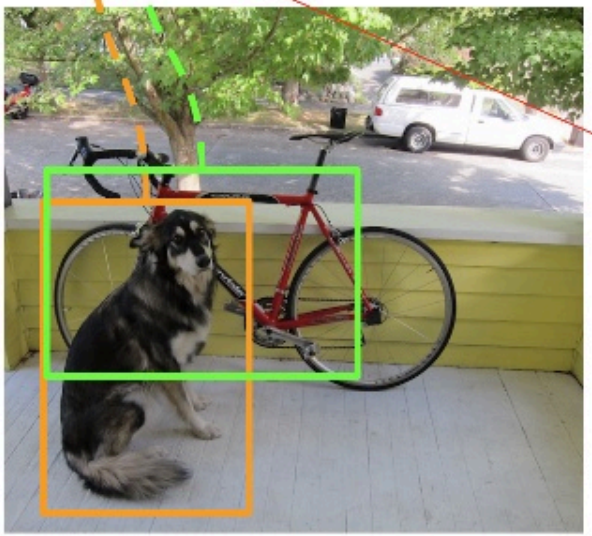


If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

deepsystems.io

Non-Maximum Suppression: intuition



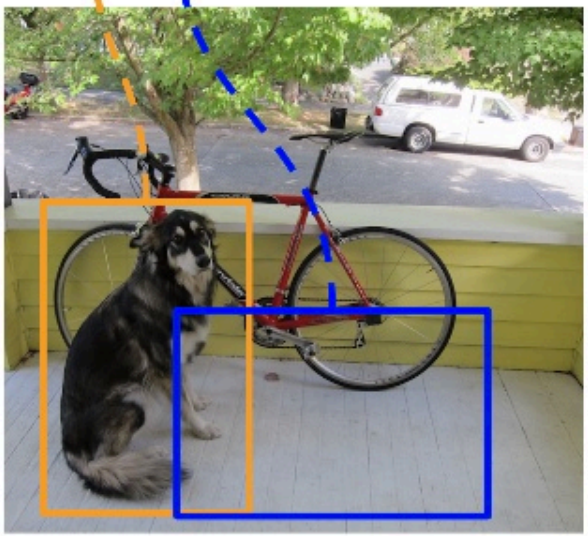


If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: set to 0.

deepsystems.io

Non-Maximum Suppression: intuition



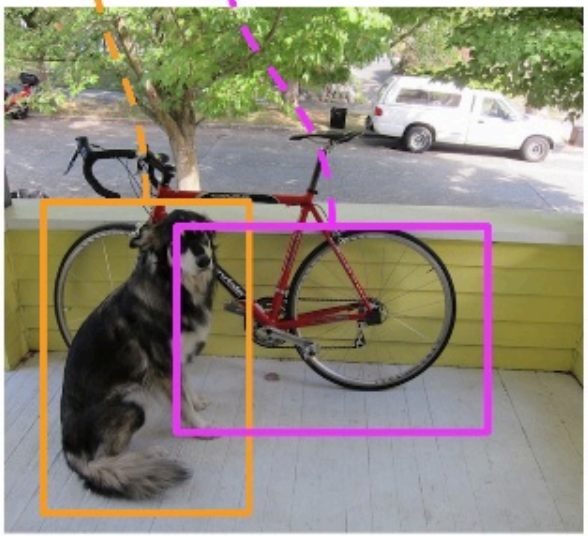
Go to next bbox_cur.

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

In this case: continue.

deepsystems.io

Non-Maximum Suppression: intuition



Go to next bbox_cur.

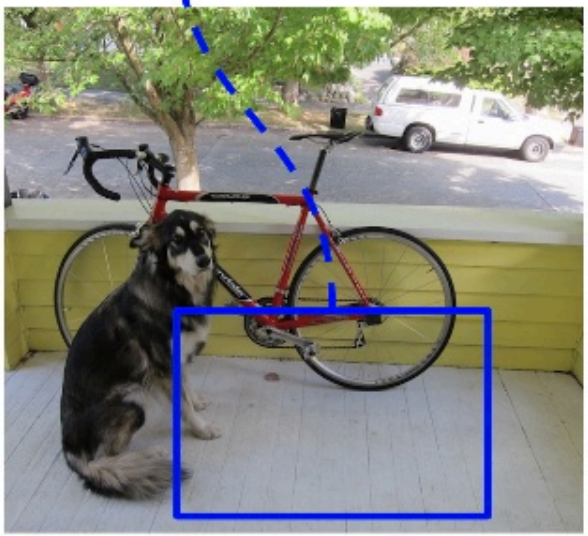
If IoU(bbox_max, bbox_cur) > 0.5 then set 0 score to bbox_cur.

In this case: continue.

Do this procedure for other "bbox_cur". After that ...

deepsystems.io

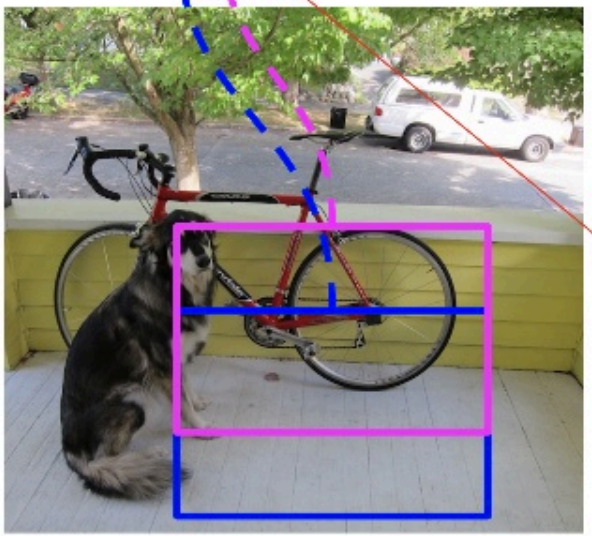
Non-Maximum Suppression: intuition



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

deepsystems.io

Non-Maximum Suppression: intuition



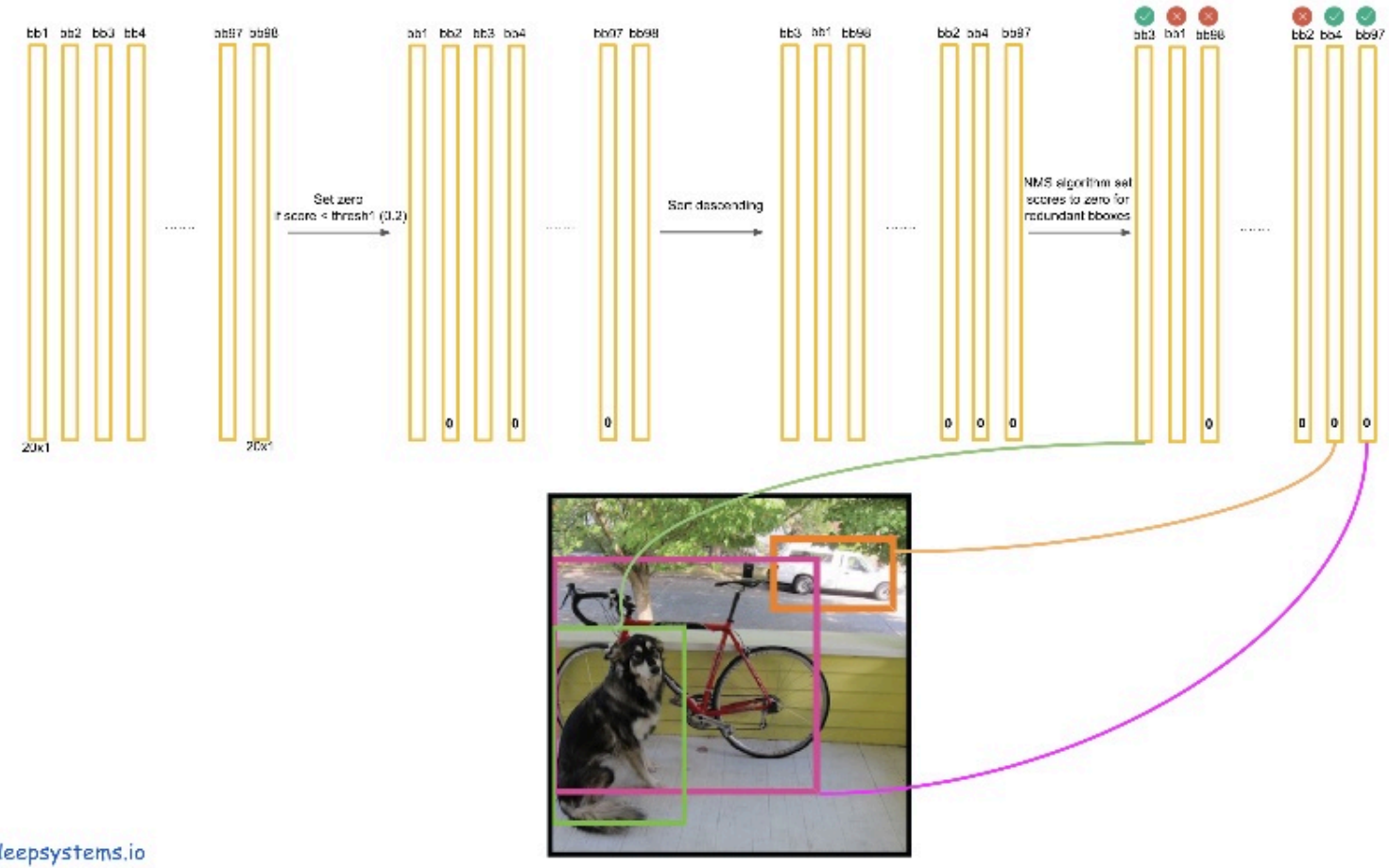
Go to next bbox_cur.

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".

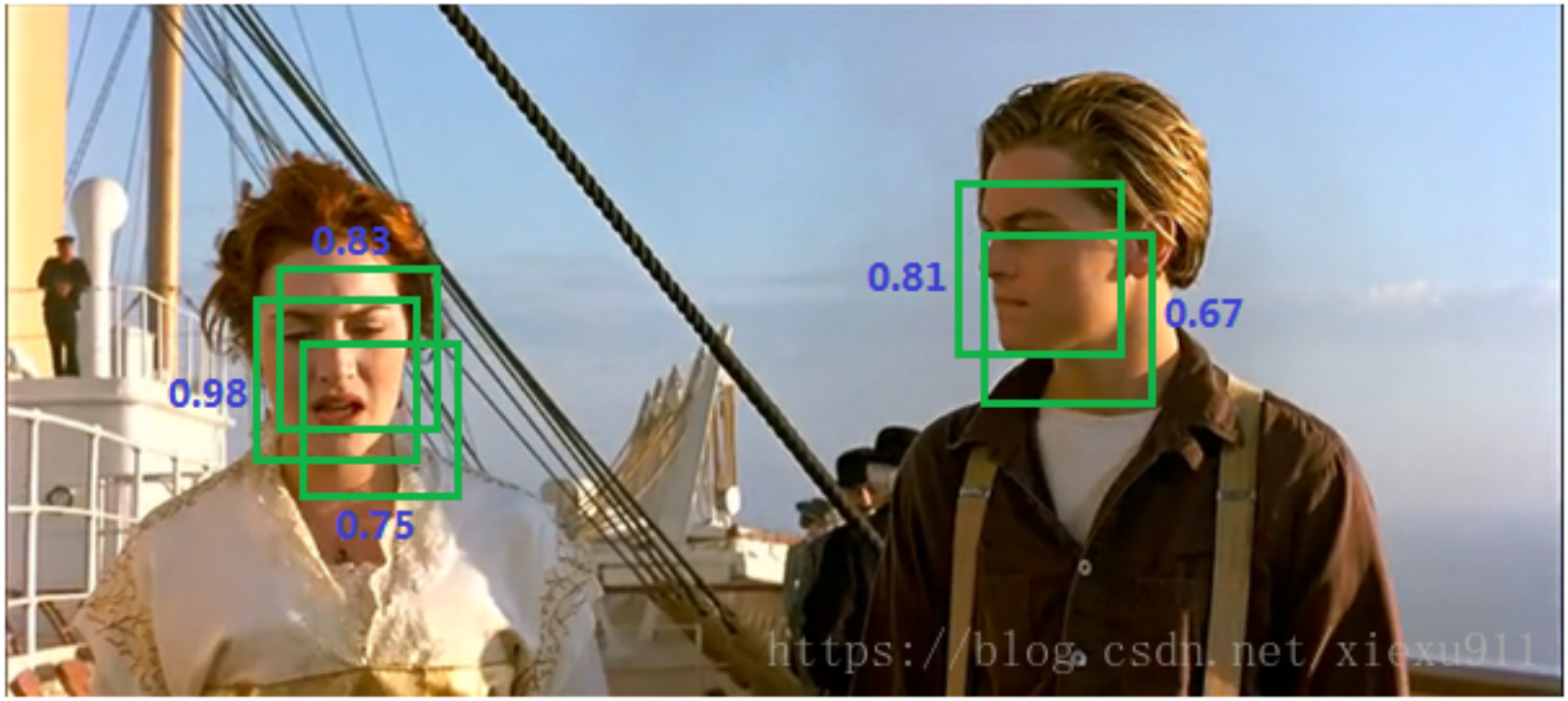
deepsystems.io



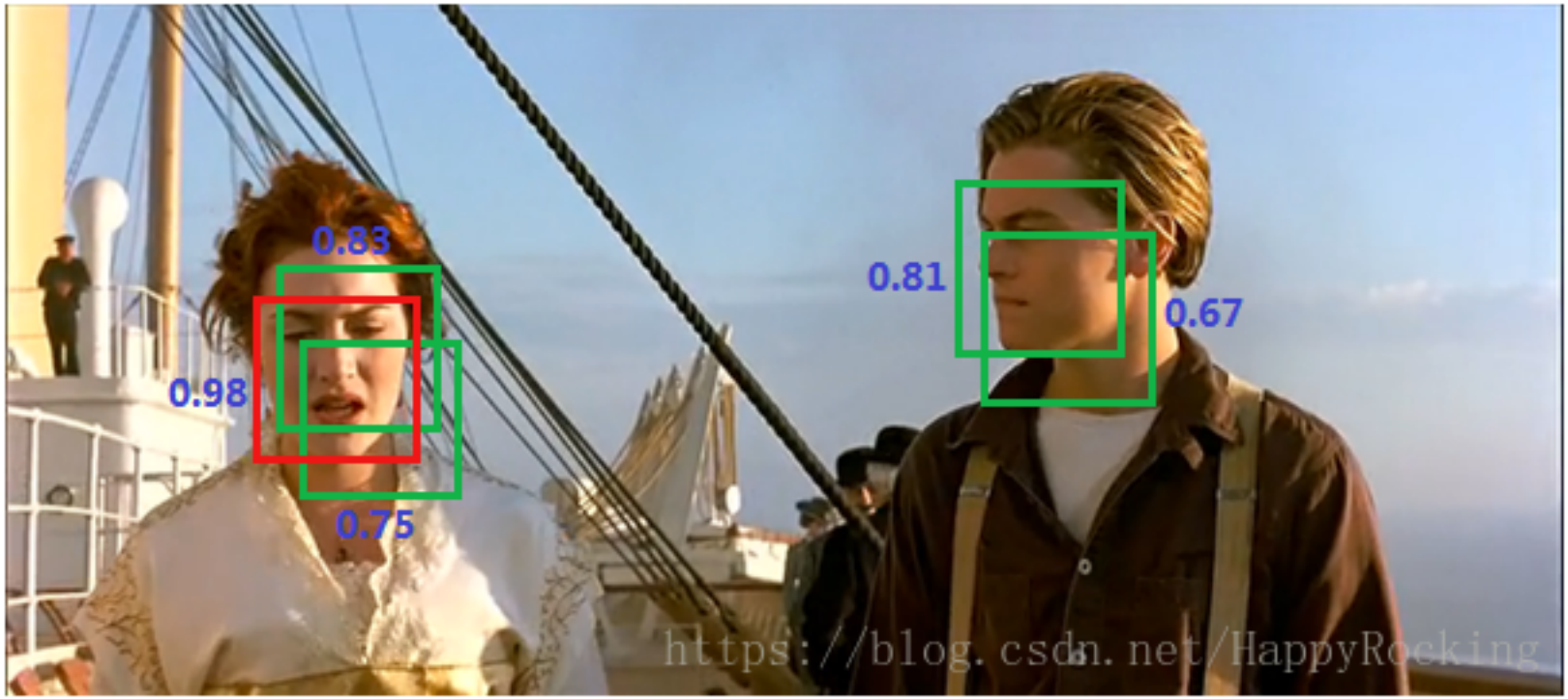
deepsystems.io

Example of NMS

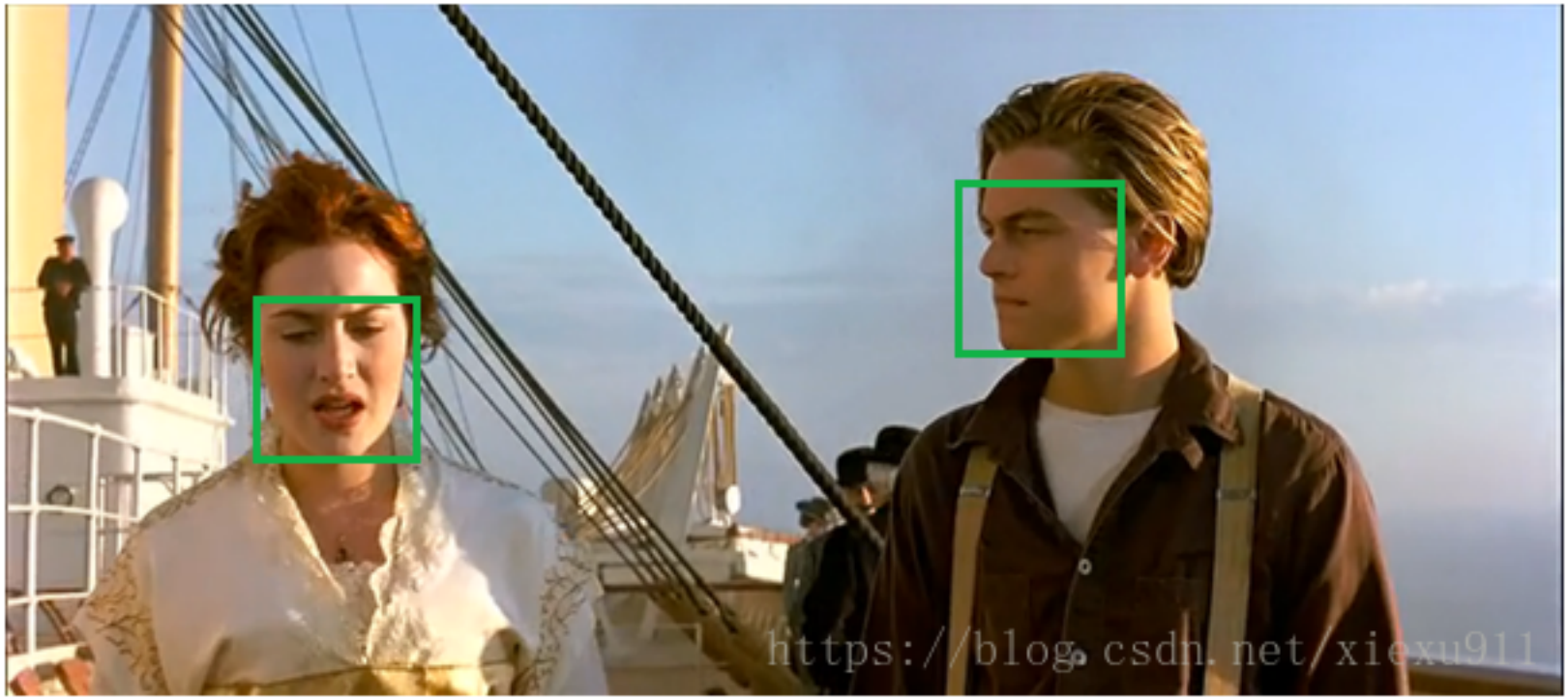
1. Identify the Bbox candidates with scores



2. Find the first Bbox



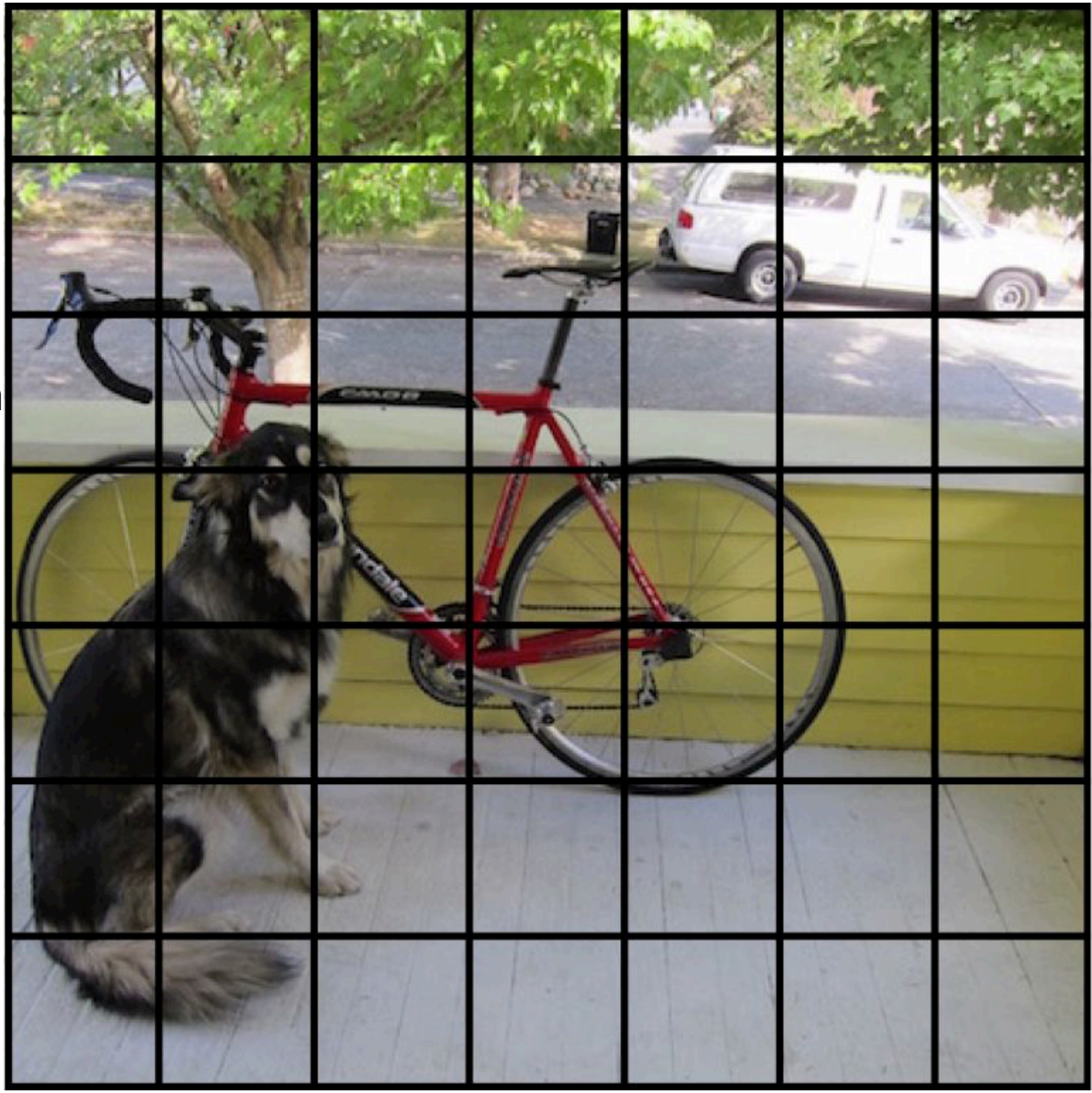
3. Find the second Bbox



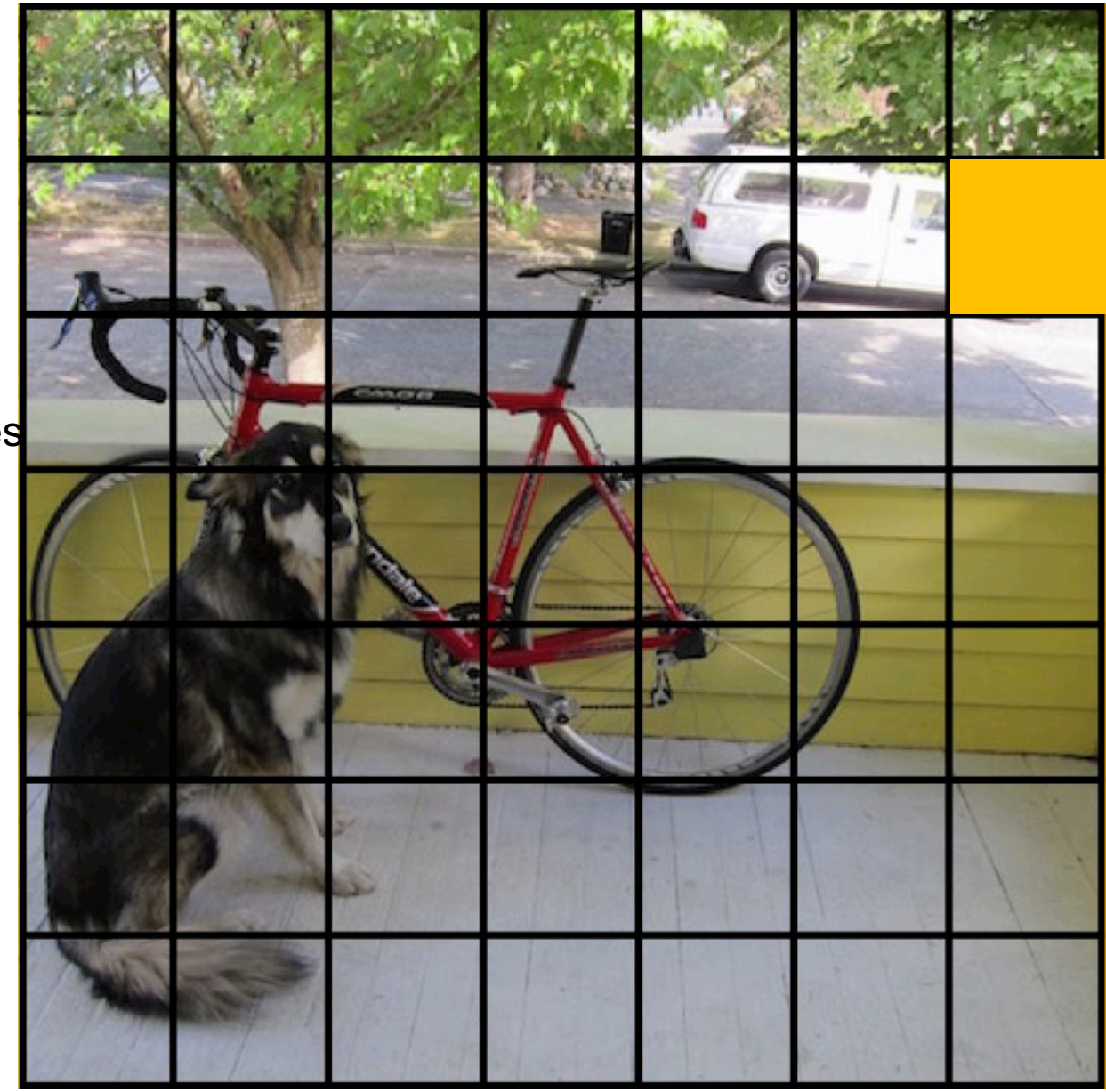
Review of YOLO Steps

Step 0: Given an image Containing multiple Objects.

Step 1: Split image into uniform Grids (S*S) (e.g. 7*7)



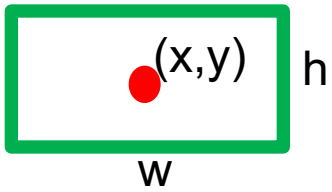
Step 2: Each cell predicts B
Boxes (x,y,w,h) and confidences
Of each box: P(Object)



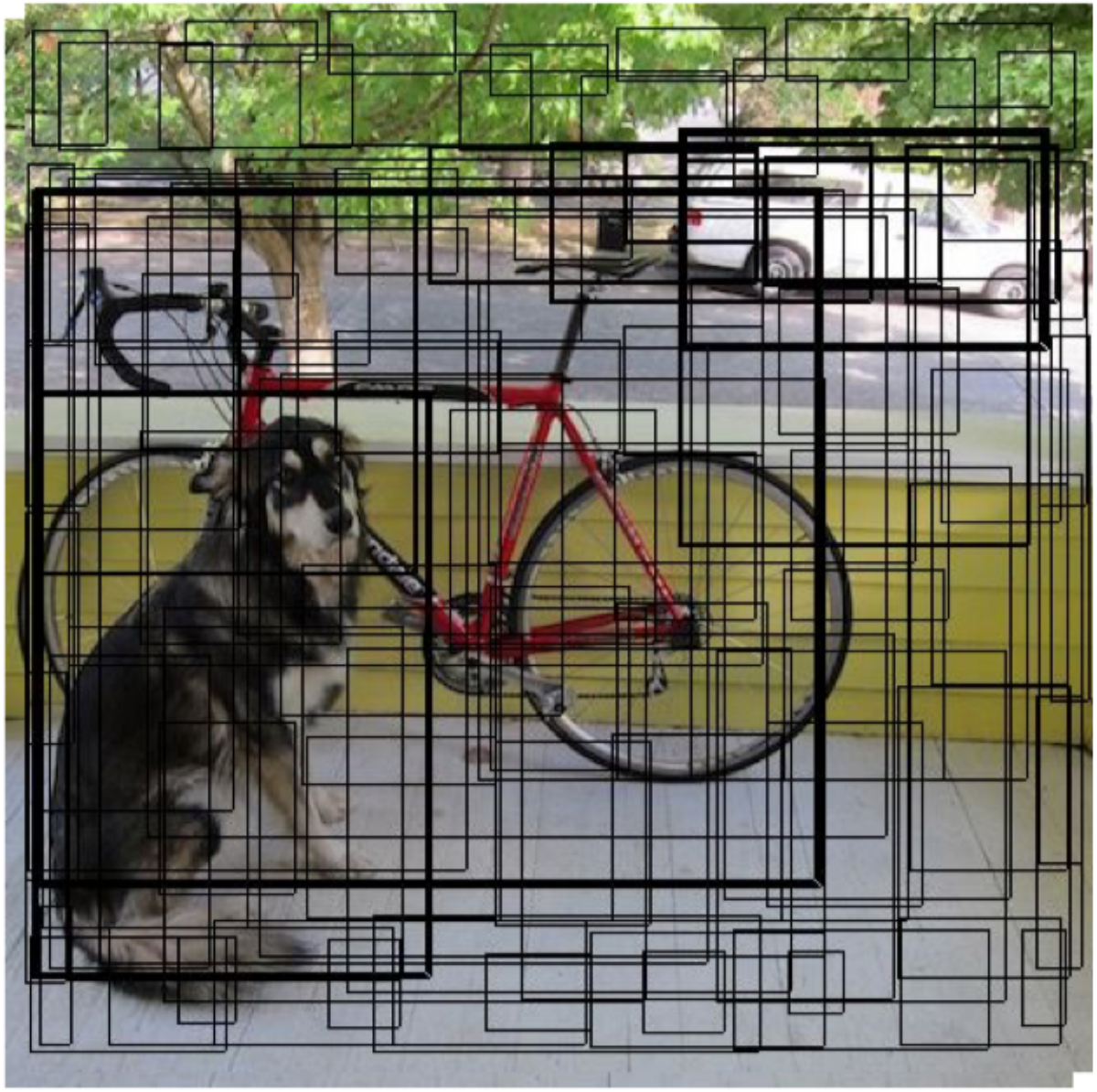
Step 3: Each cell predicts B
Boxes (x,y,w,h) and confidences
Of each box: P(Object)

B =2

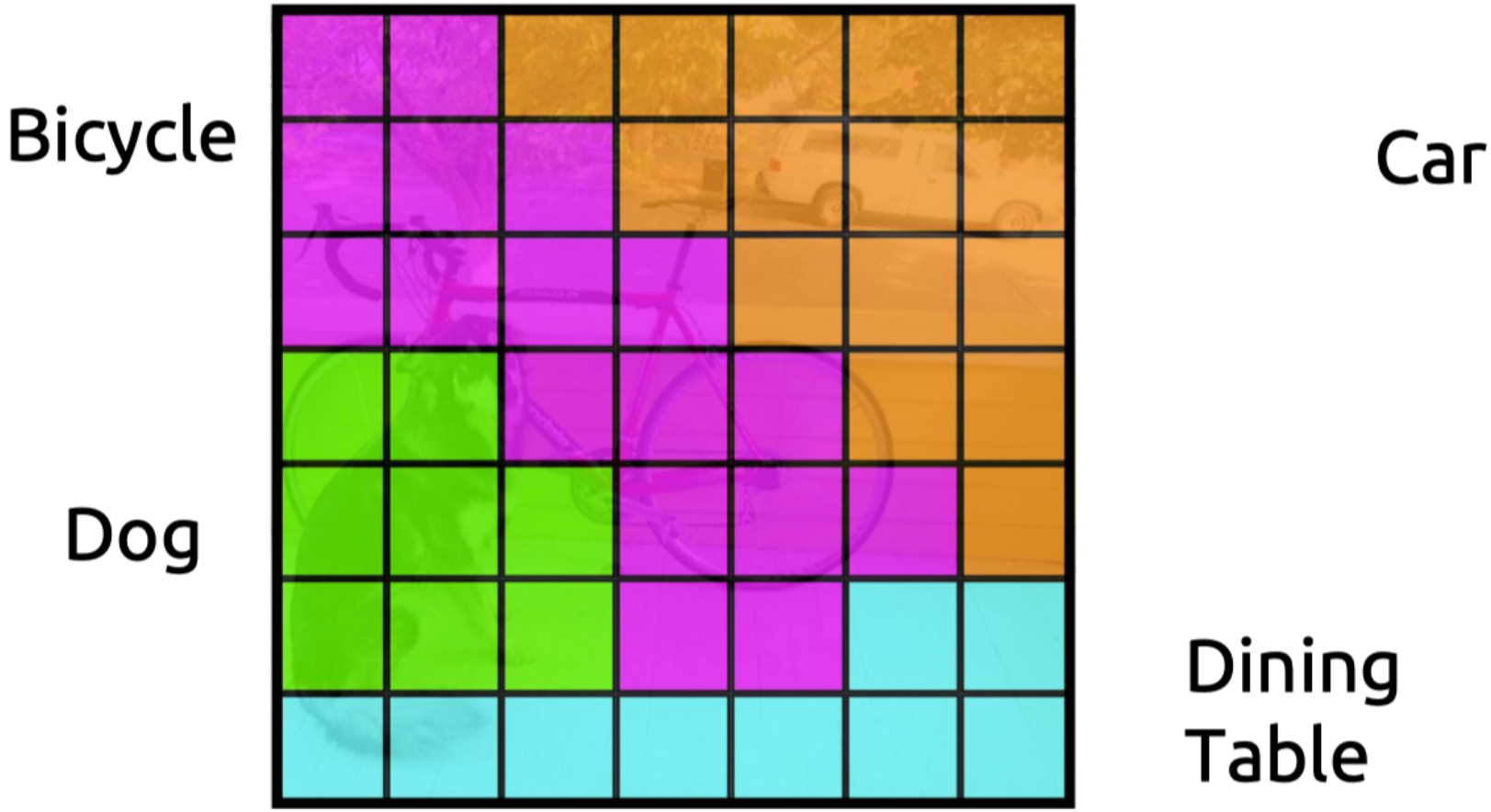
Each Box:



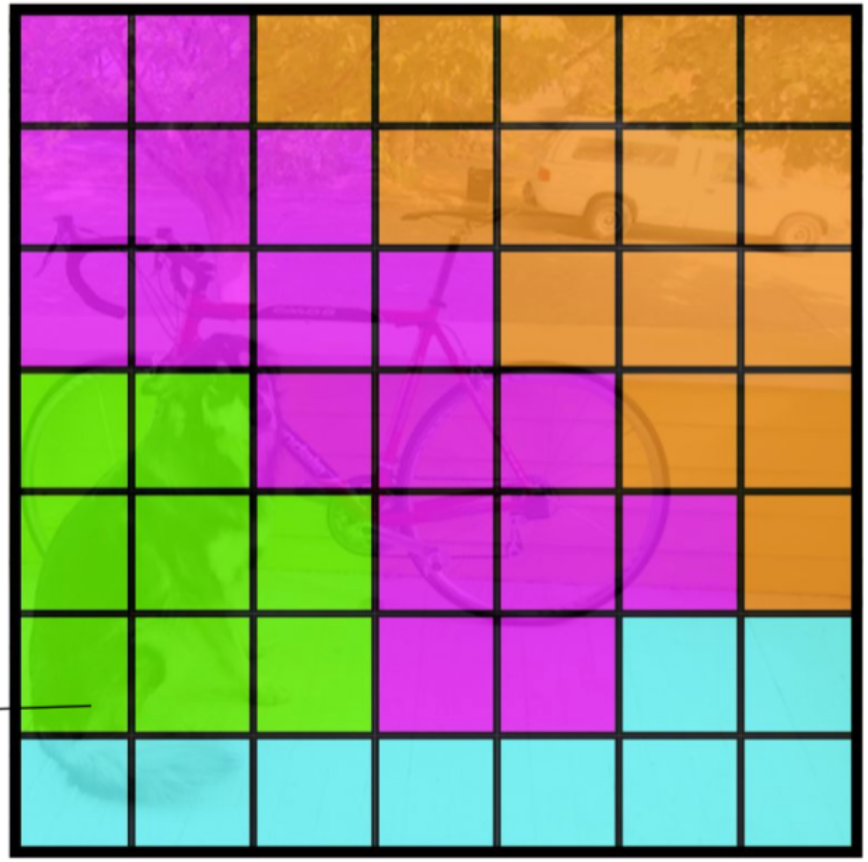
Step 3: Predicted Bboxes by All cells

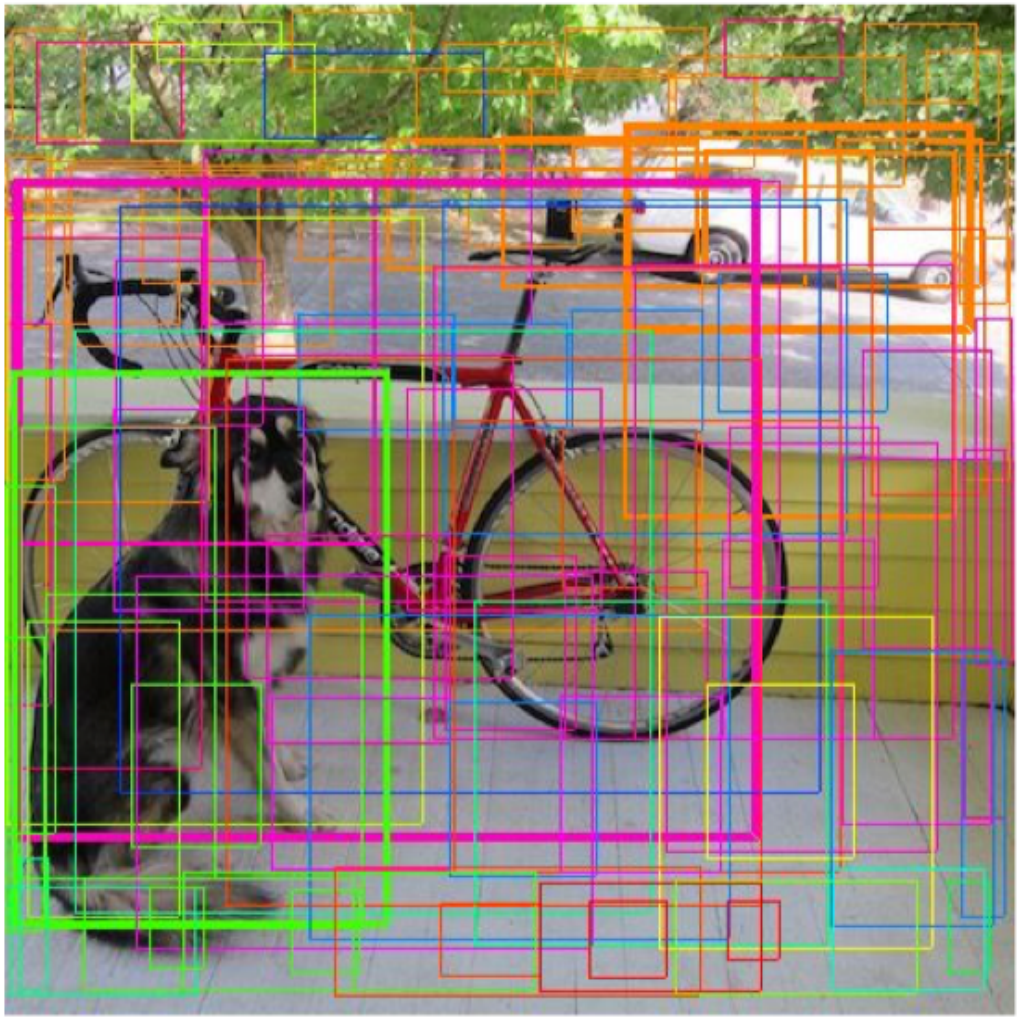


Step 3: Predict the classes for each bbox



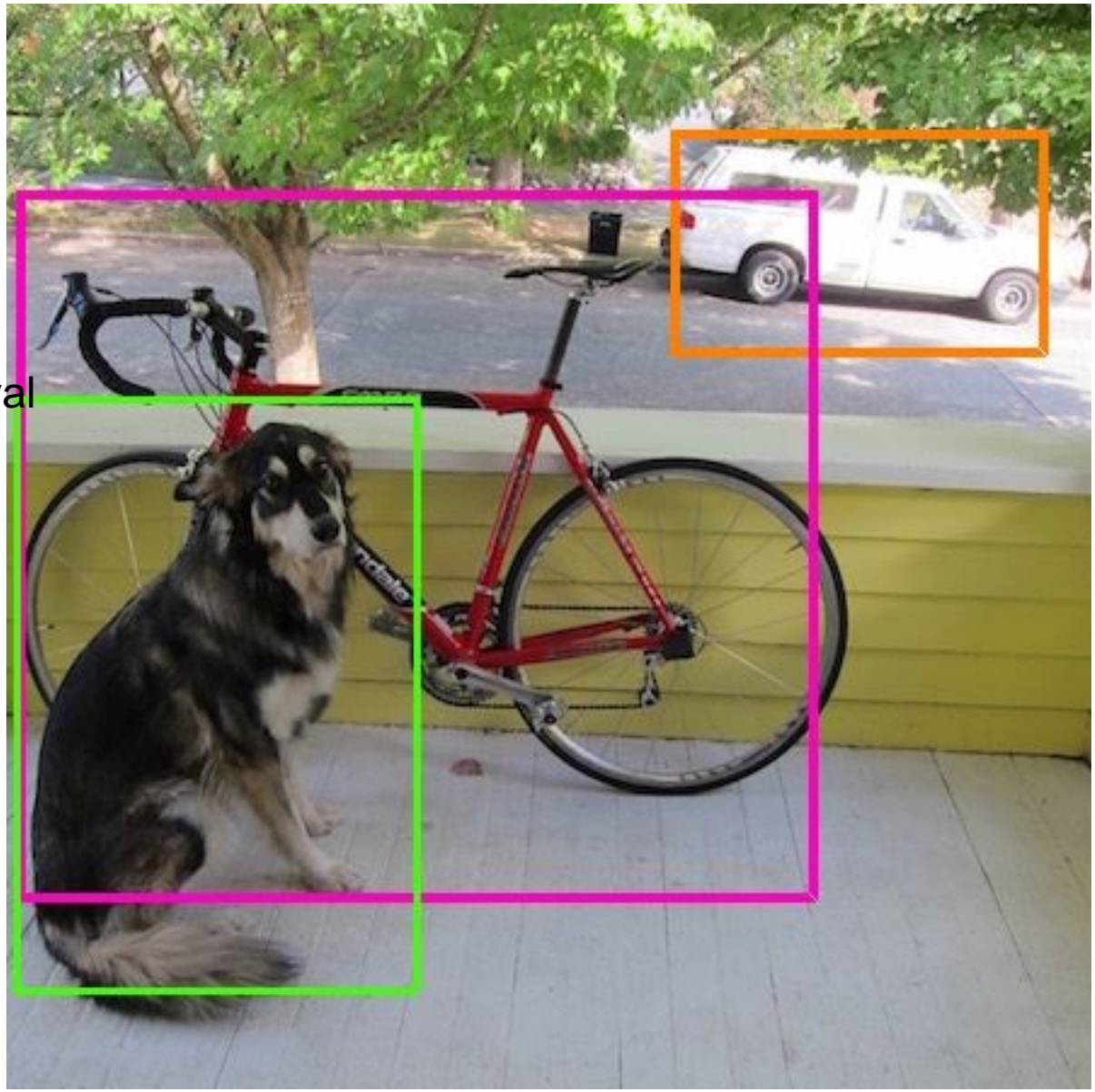
Conditioned on object: P(Car | Object)



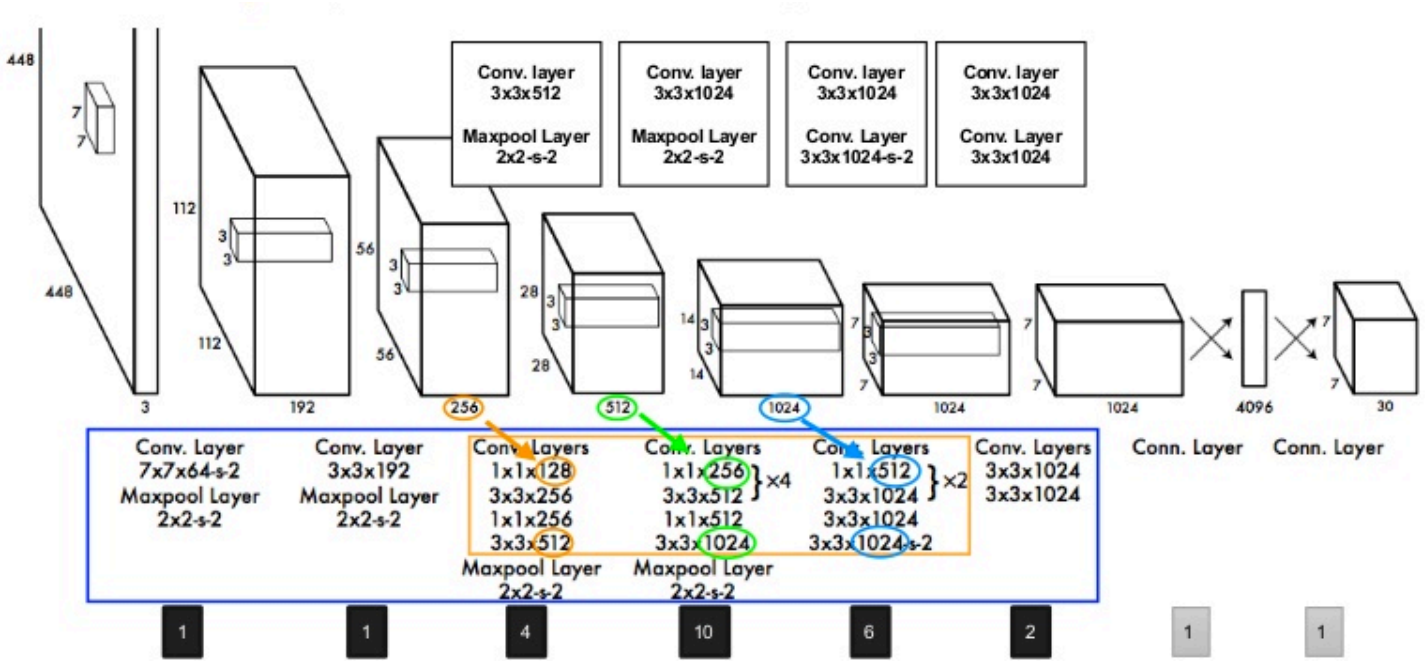


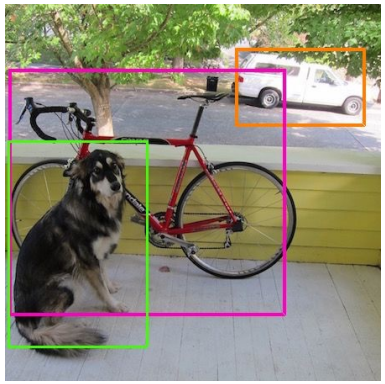
$$P(\text{class}|\text{Object}) * P(\text{Object}) = P(\text{class})$$

Step 4: NMS for Bboxes removal



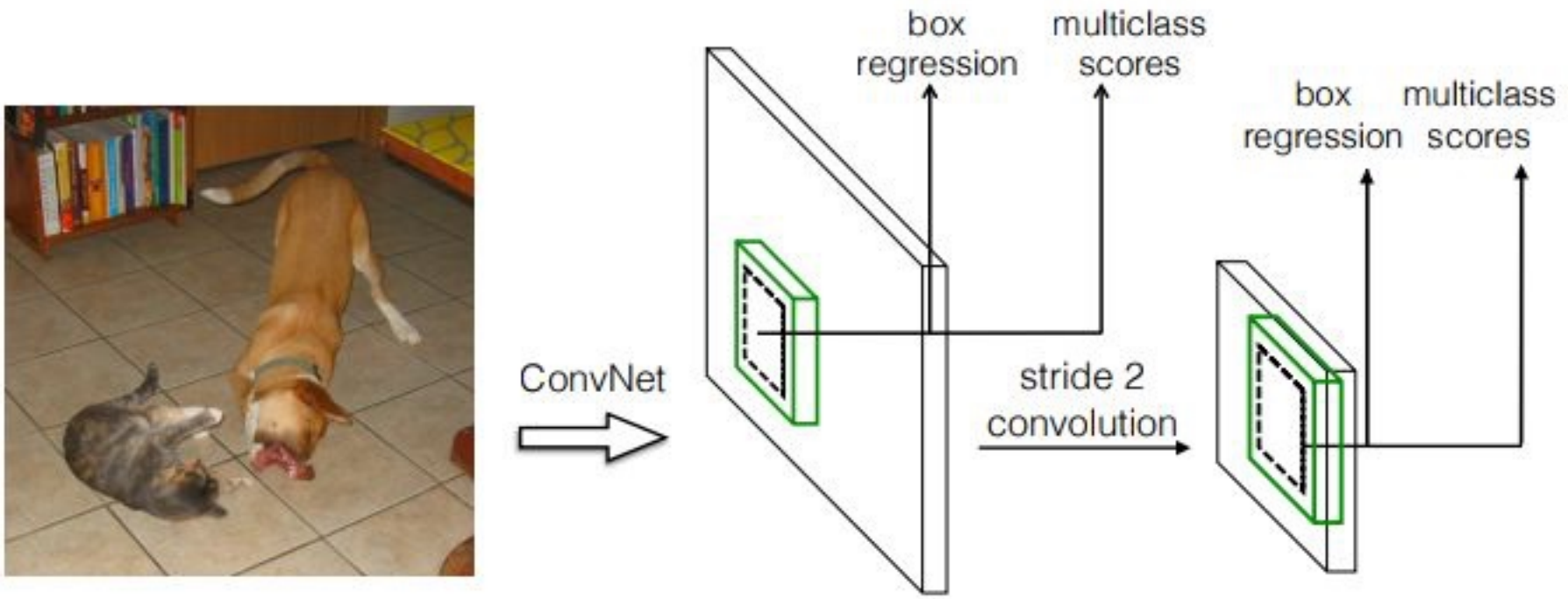
What is issue of YOLO?





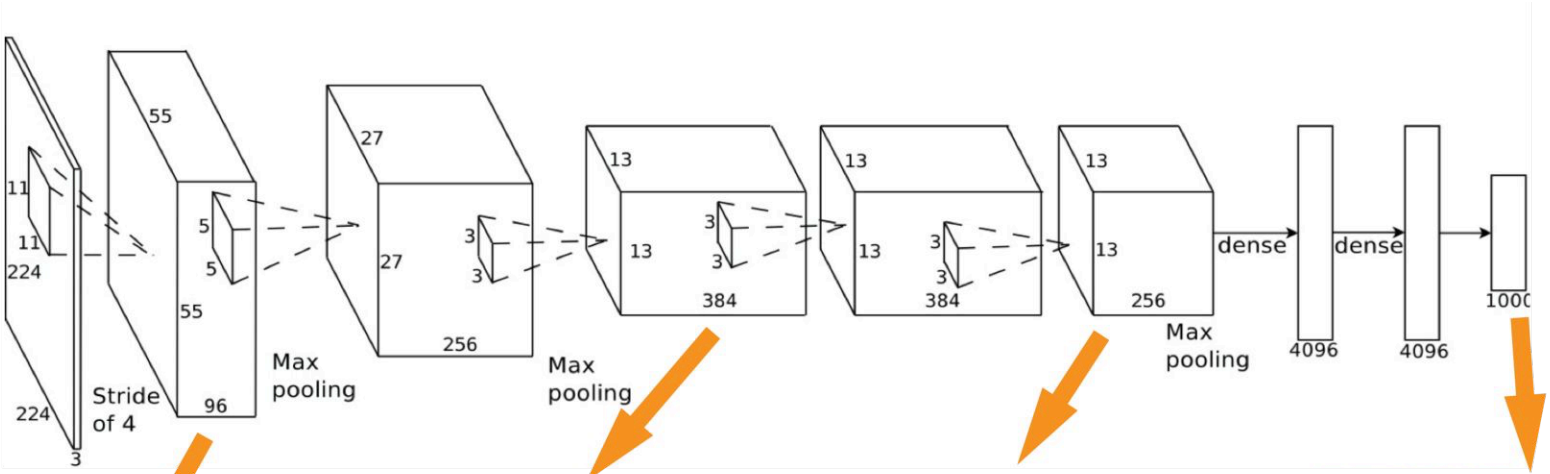
The scales?

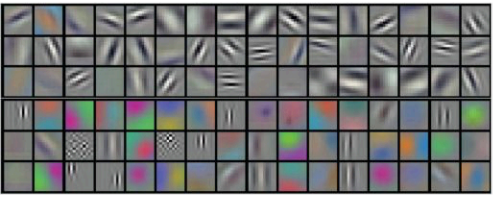
Detection Algorithm: SSD: Single Shot Multi-Box Detector



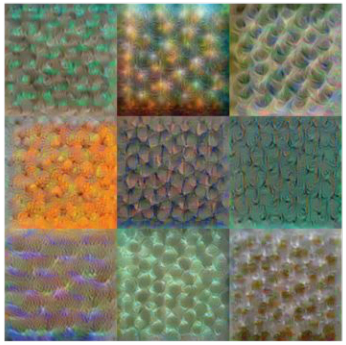
Feature Maps at different layers with different object sizes

Src: SSD paper, ECCV 2016





Conv 1: Edge+Blob



Conv 3: Texture

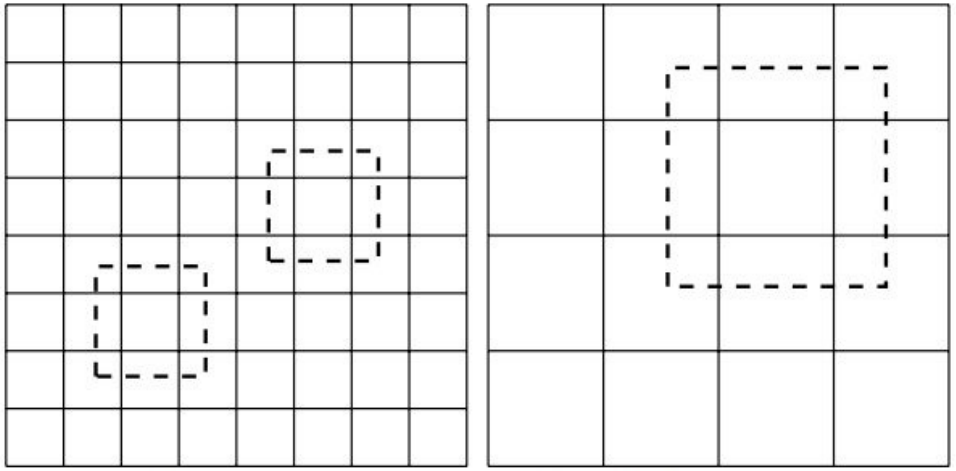
Numerical

Data-driven

Conv 5: Object Parts

Fc8: Object Classes

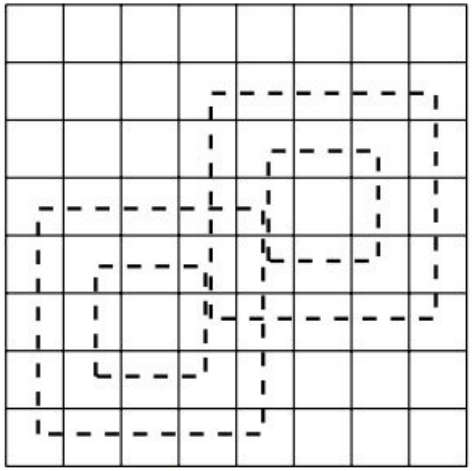
SSD



8×8 feature map 4×4 feature map

vs.

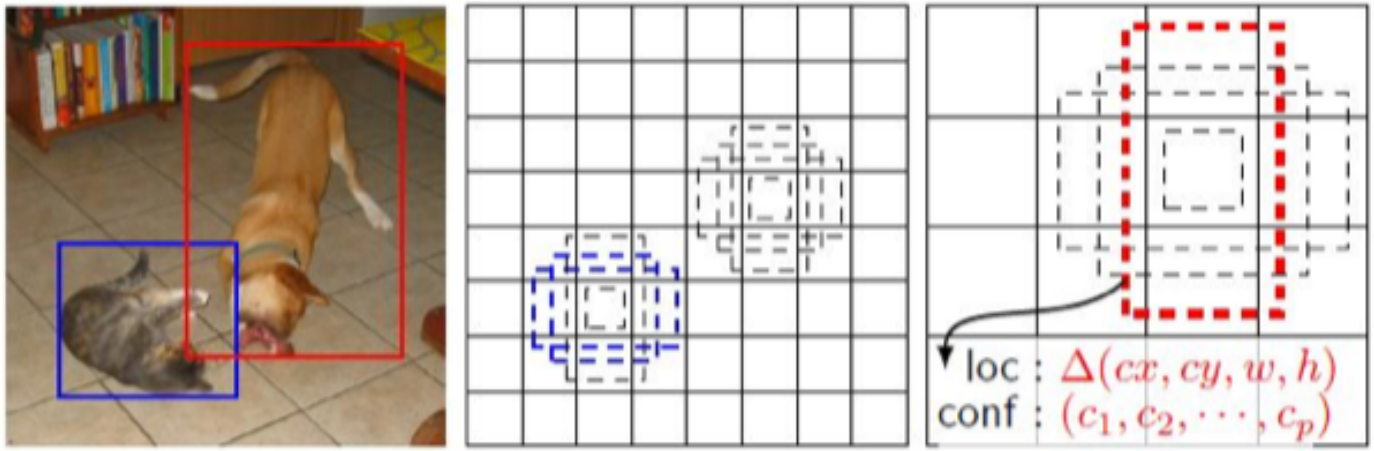
Faster R-CNN Objectness Proposal, Ren 2015



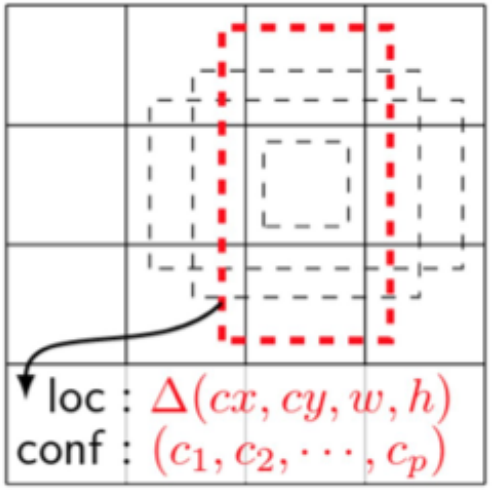
8×8 feature map

Why small boxes in large feature maps?

- large feature map - small receptive field - small object
- small feature map - large receptive field - large object



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map



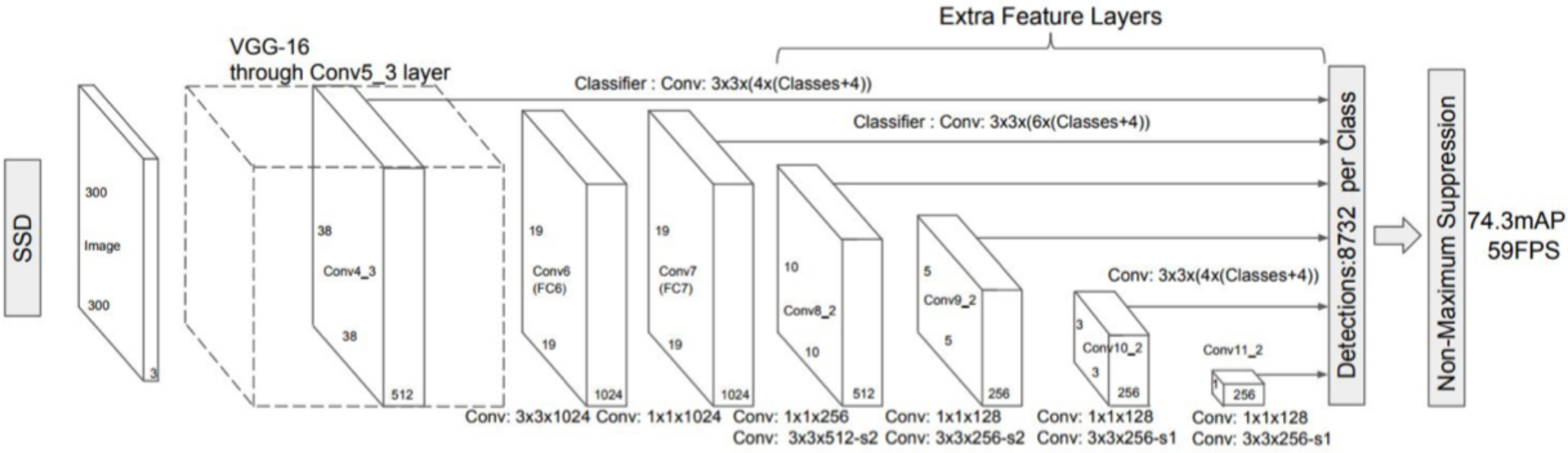
3 x 3 x p kernel

Each box:

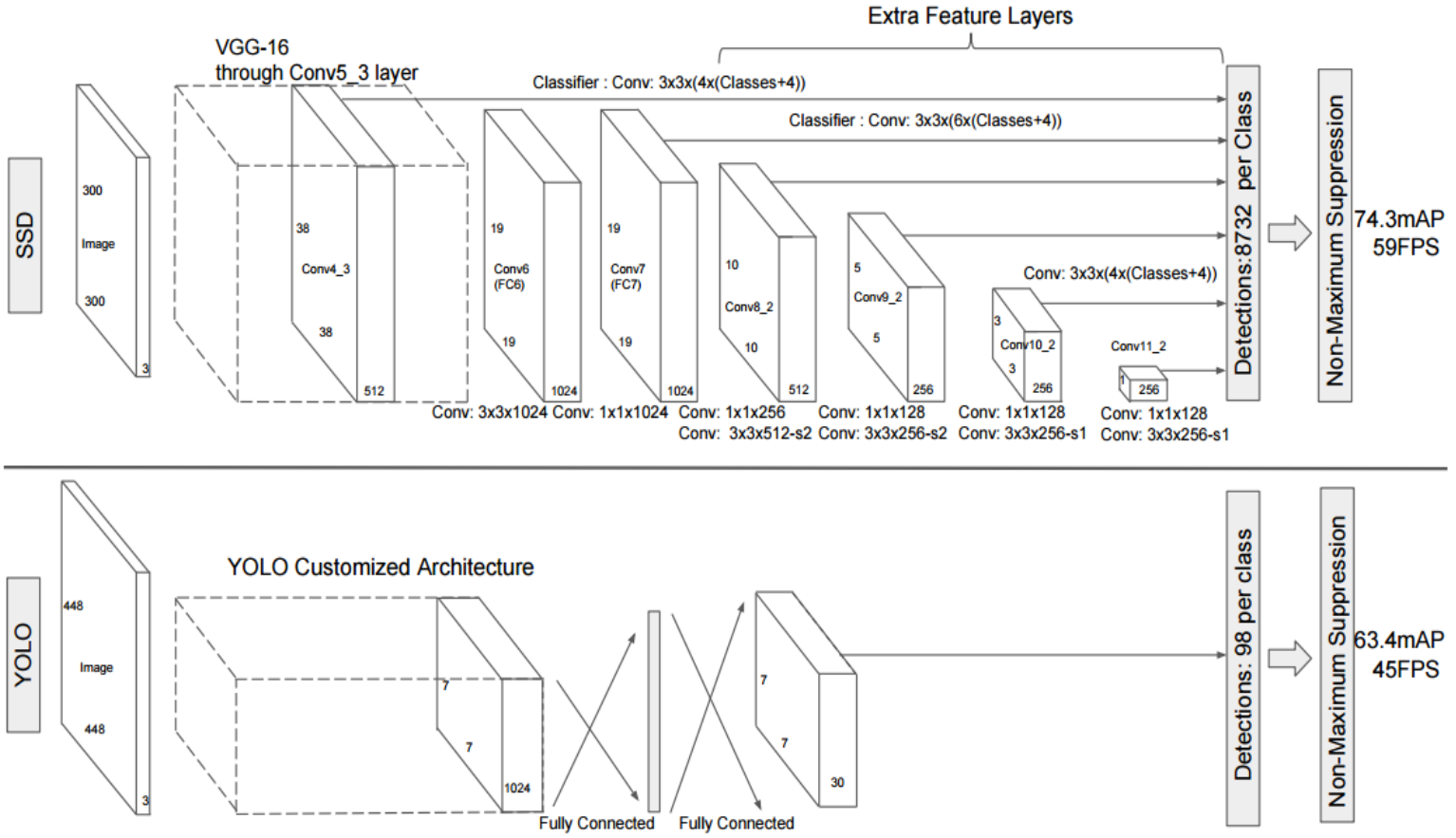
- cls: # classes (C1, C2, …, Cp)
- reg: 4 parameters delta(cx, cy, w, h)

conv kernels: (Classes+4) x (# Default Boxes)

SSD Network Structure



Network Structure Comparison



Detection Results