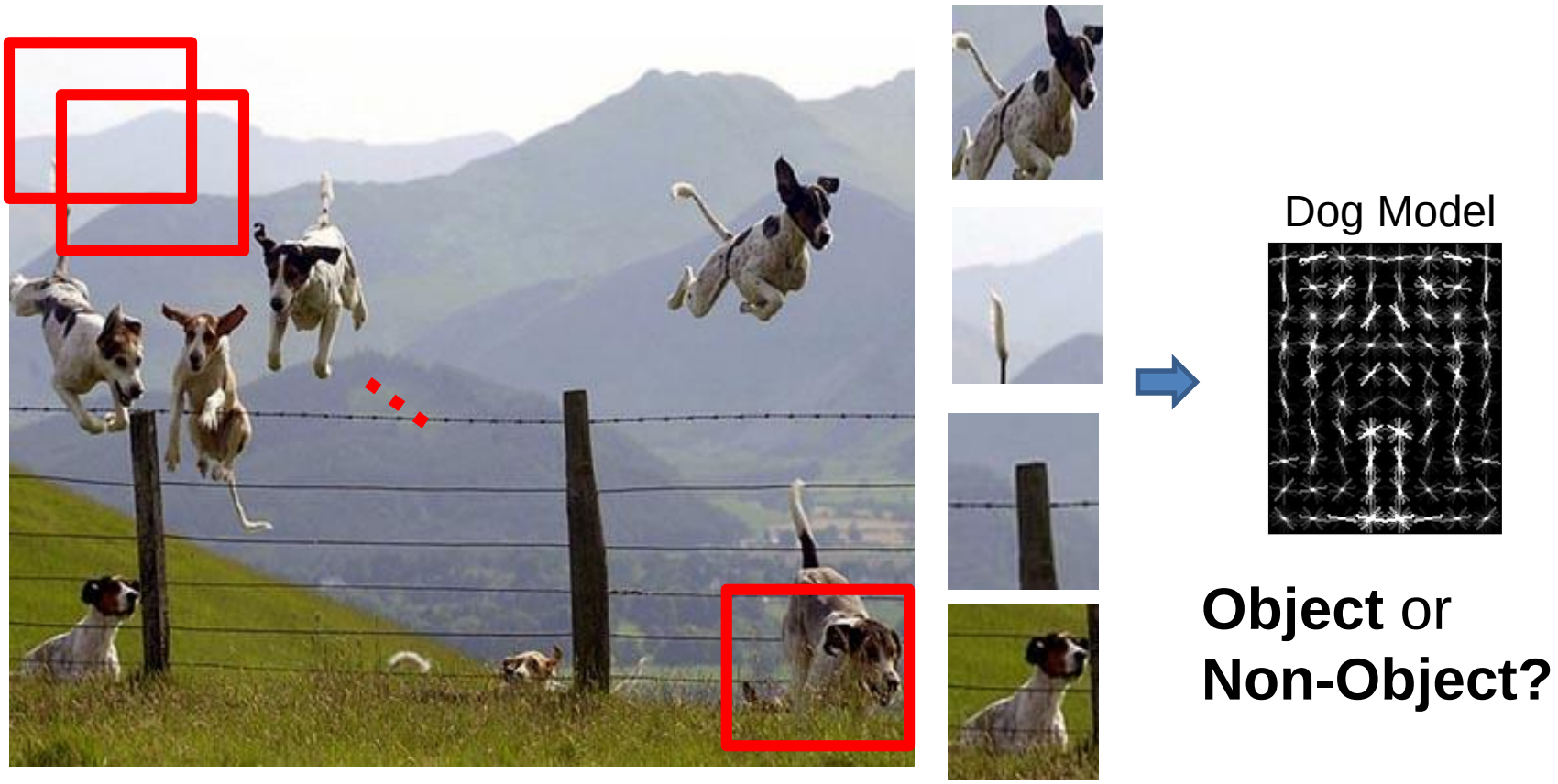
Object Detection

Object Category Detection

- Focus on object search: “Where is it?”
- Build templates that quickly differentiate object patch from background patch



Challenges in modeling the object class

Illumination

Object pose

Clutter

Occlusions

Intra-class appearance

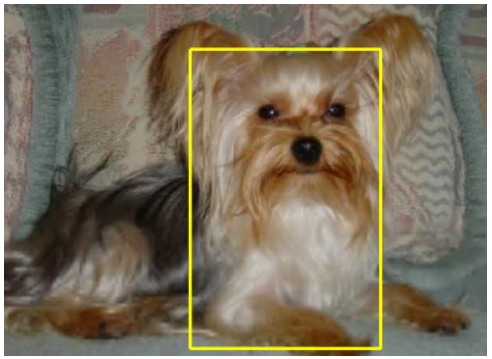
Viewpoint

Slide from K. Grauman, B. Leibe

Challenges in modeling the non-object class

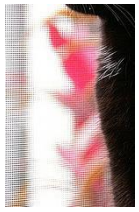
True Detections

Bad Localization



Confused with Similar Object

Misc. Background



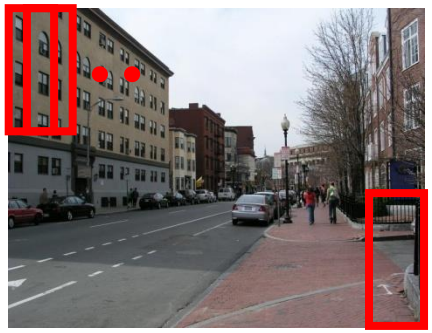
Confused with Dissimilar Objects

Influential Works in Detection

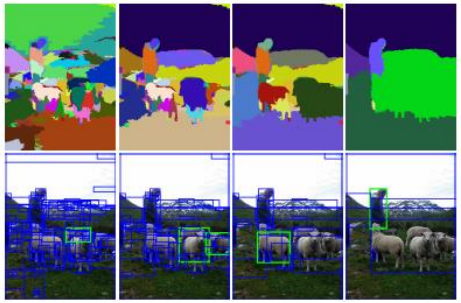
- Sung-Poggio (1994, 1998) : ~2412 citations
  - Basic idea of statistical template detection (I think), bootstrapping to get "face-like" negative examples, multiple whole-face prototypes (in 1994)
- Rowley-Baluja-Kanade (1996-1998) : ~4953
  - "Parts" at fixed position, non-maxima suppression, simple cascade, rotation, pretty good accuracy, fast
- Schneiderman-Kanade (1998-2000,2004) : ~2600
  - Careful feature/classifier engineering, excellent results, cascade
- Viola-Jones (2001, 2004) : ~27,000
  - Haar-like features, Adaboost as feature selection, hyper-cascade, very fast, easy to implement
- Dalal-Triggs (2005) : ~18000
  - Careful feature engineering, excellent results, HOG feature, online code
- Felzenszwalb-Huttenlocher (2000): ~2100
  - Efficient way to solve part-based detectors
- Felzenszwalb-McAllester-Ramanan (2008,2010): ~7200
  - Excellent template/parts-based blend
- Girshick-Donahue-Darrell-Malik (2014): ~4700
  - Region proposals + fine-tuned CNN features (marks significant advance in accuracy over hog-based methods)
- Redmon, Divvala, Girshick, Farhadi (2016): ~210
  - Refine and simplify RCNN++ approach to predict directly from last conv layer

Summary: statistical templates

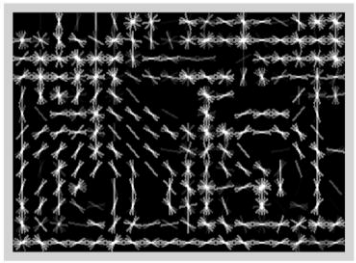
Propose Window → Extract Features → Classify → Post-process



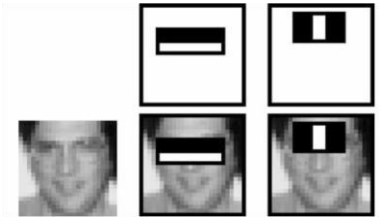
Sliding window: scan image pyramid



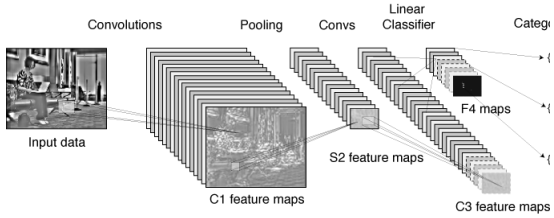
Region proposals: edge/region-based, resize to fixed window



HOG



Fast randomized features



CNN features

SVM

Boosted stubs

Neural network

Non-max suppression

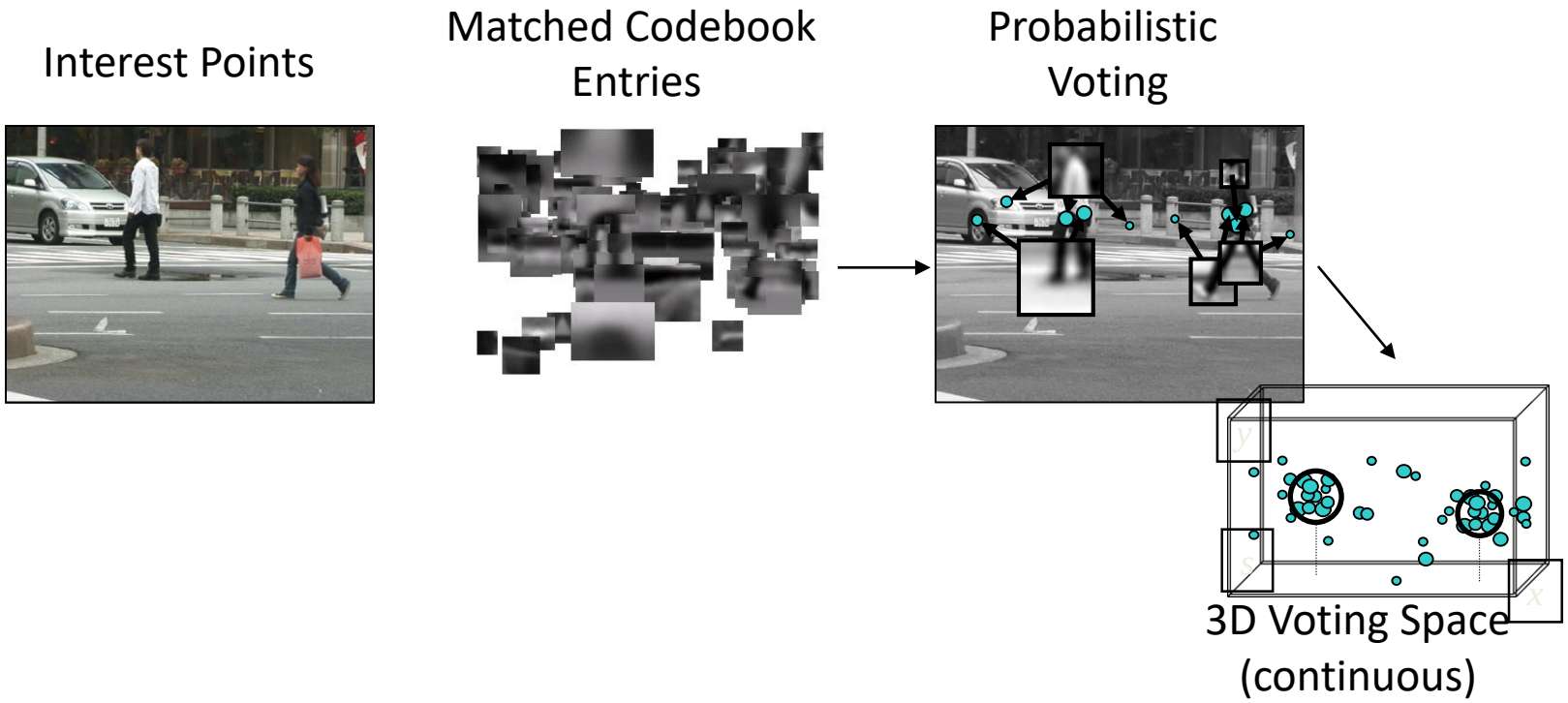
Segment or refine localization

Generating hypotheses

1. Sliding window
   - Test patch at each location and scale

Generating hypotheses

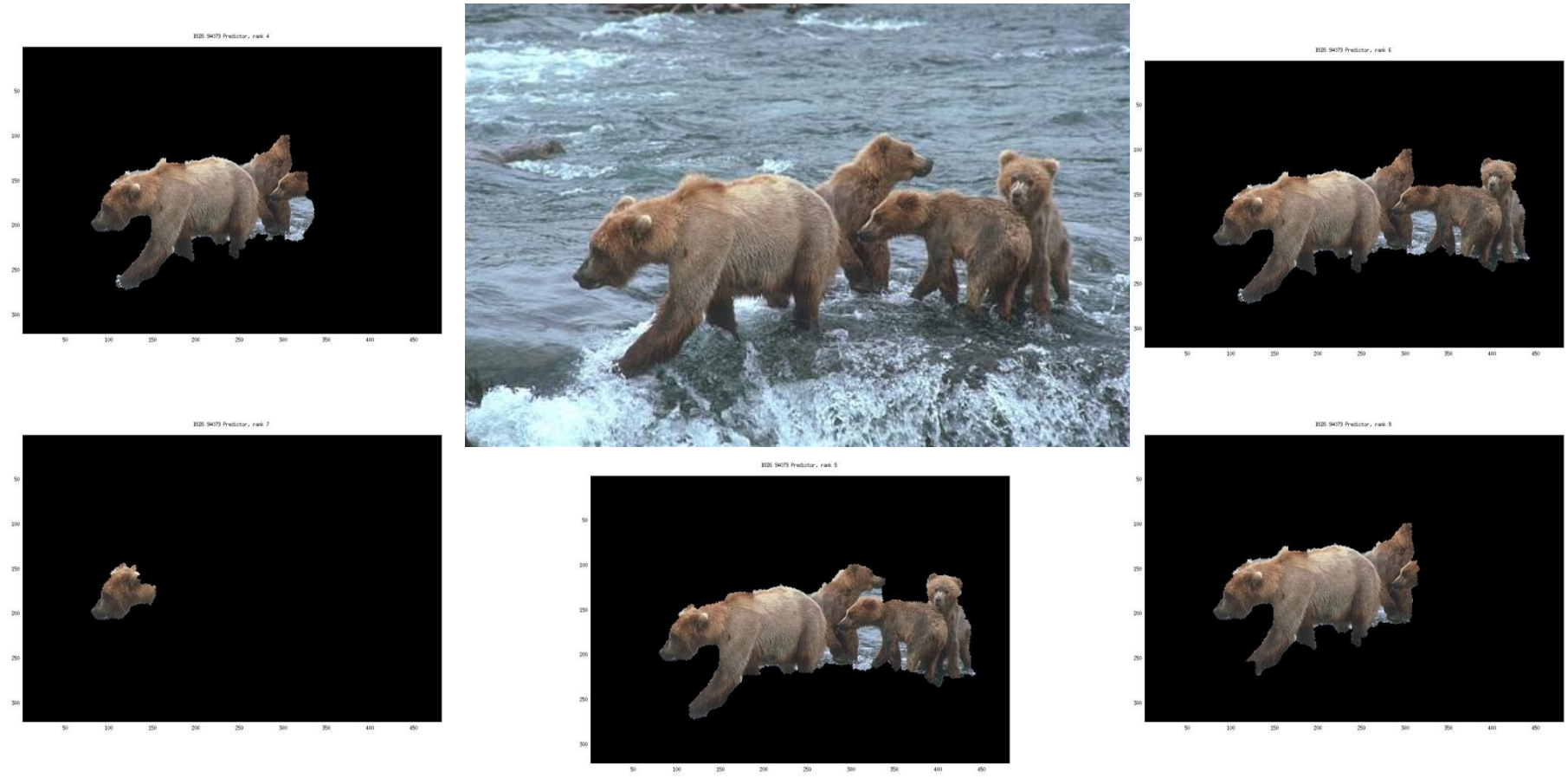
2. Voting from patches/keypoints



ISM model by Leibe et al.

Generating hypotheses

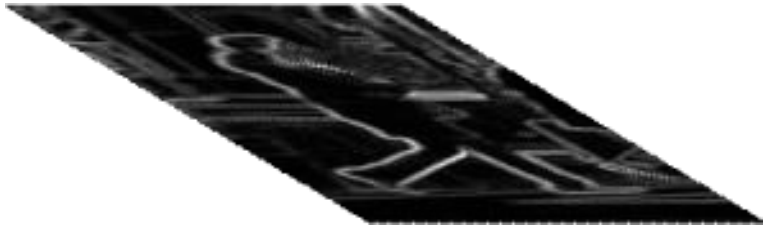
3. Region-based proposal

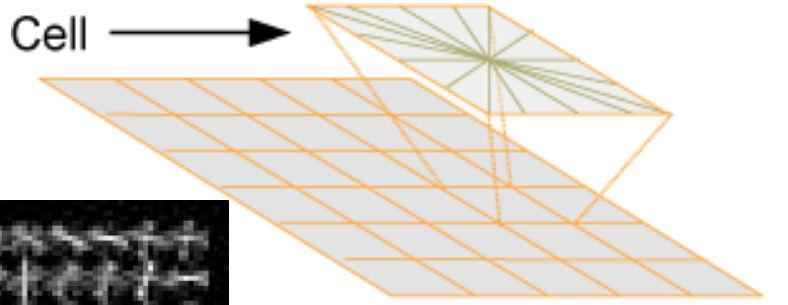


Endres Hoiem 2010

Score Hypotheses

Histograms of Oriented Gradients



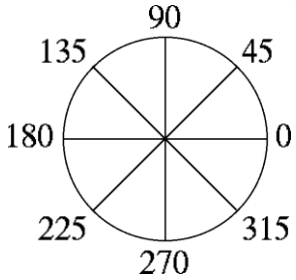


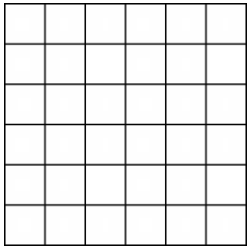
1. Compute gradients on an image region of 64x128 pixels

2. Compute histograms on ‘cells’ of typically 8x8 pixels (i.e. 8x16 cells)

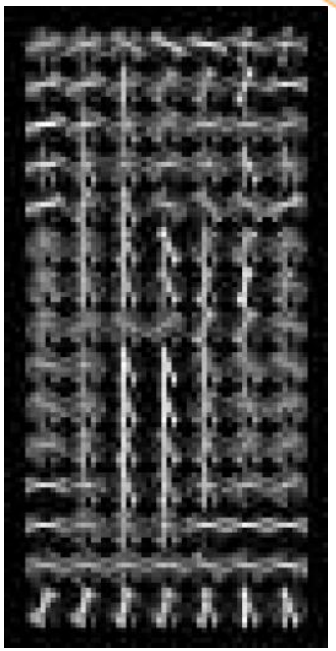
- Histogram of gradient orientations

  -Orientation -Position



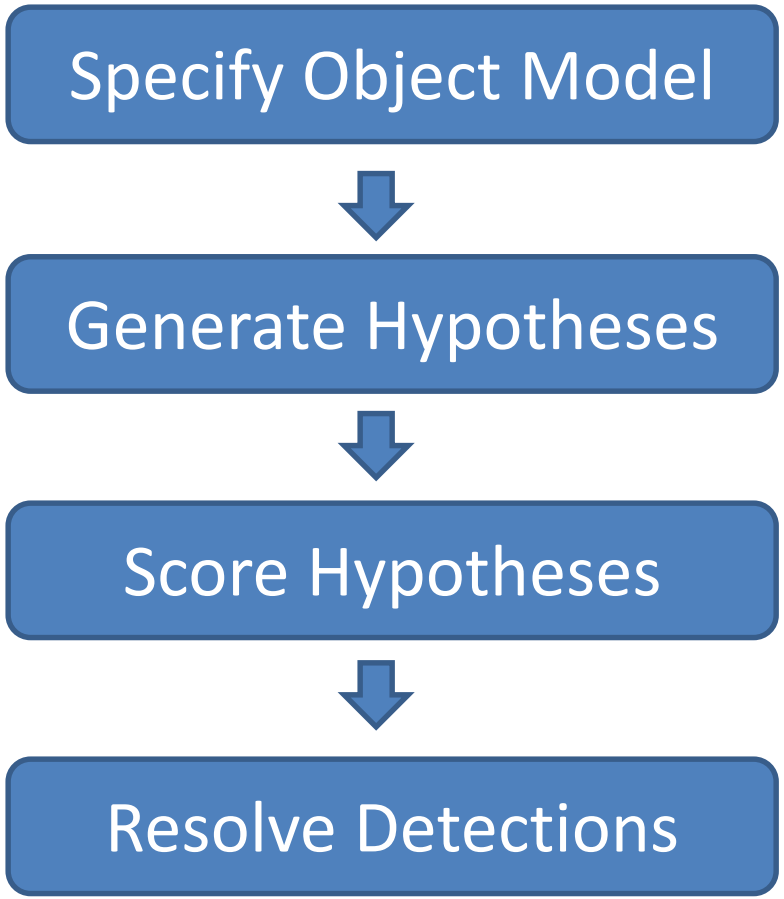


- Weighted by magnitude



7x15 ? Why ?

General Process of Object Recognition



Currently CNN features and classifiers

CNN

Convolution

Max Pooling

Convolution

Max Pooling

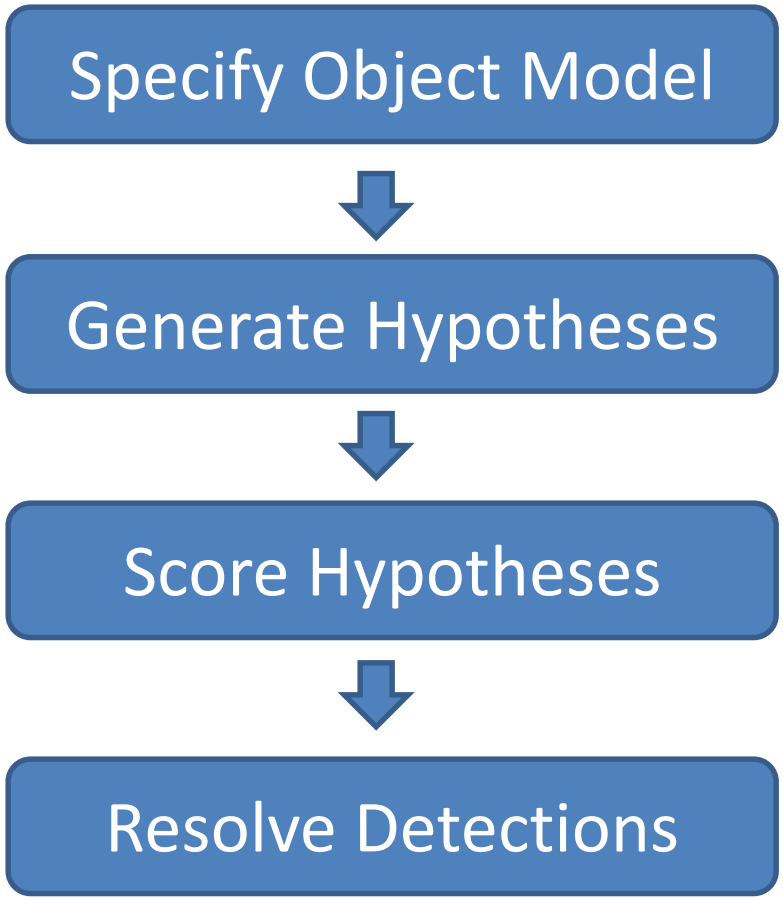
Can repeat many times

Flatten

Fully Connected Feedforward network

cat dog ……

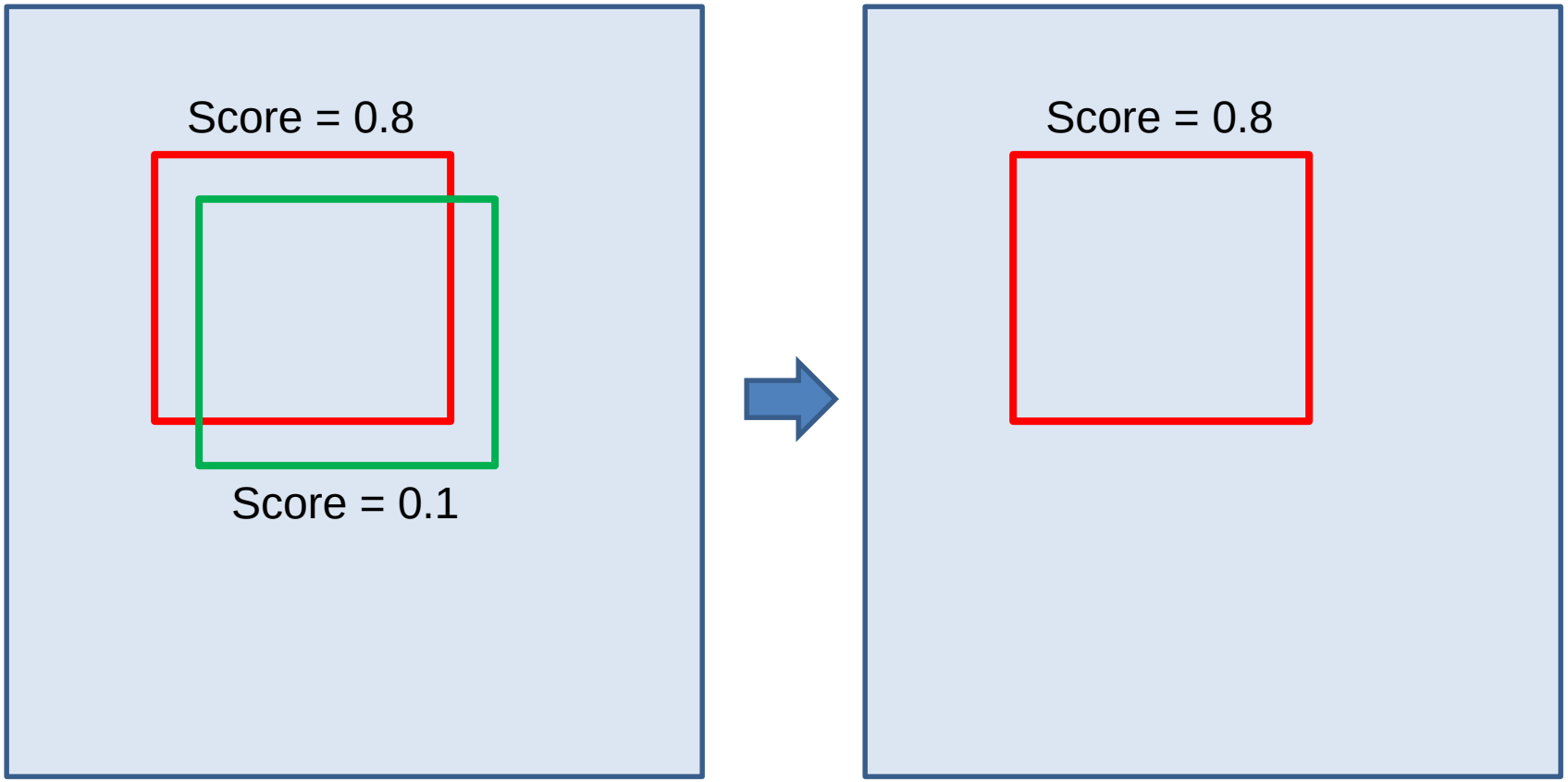
General Process of Object Recognition



Optionally, rescore each proposed object based on whole set

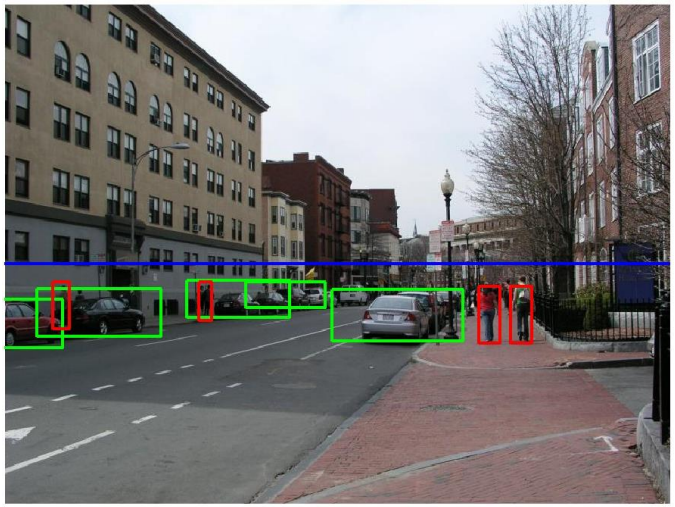
Resolving detection scores

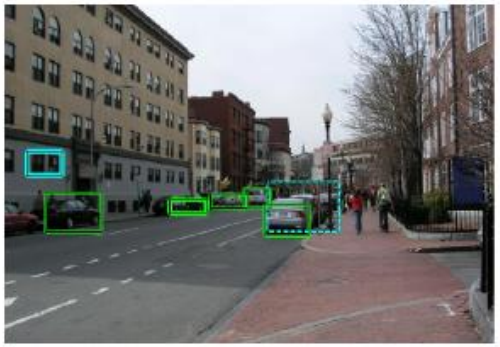
1. Non-max suppression



Resolving detection scores

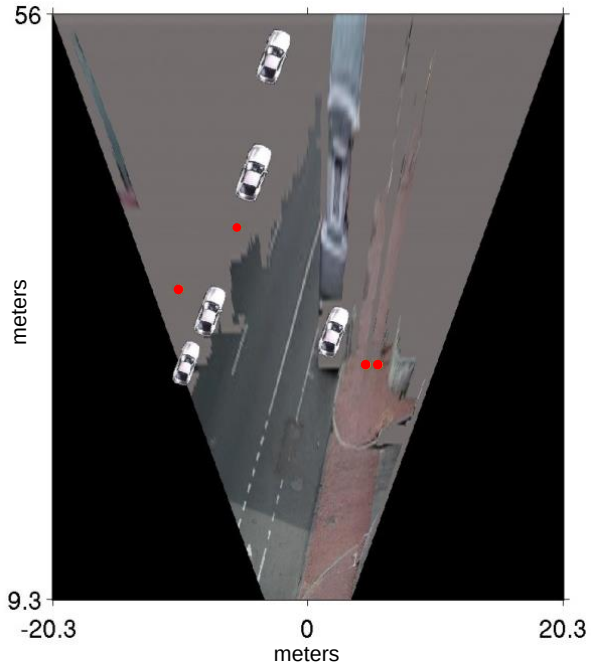
2. Context/reasoning





(g) Car Detections: Local

(h) Ped Detections: Local



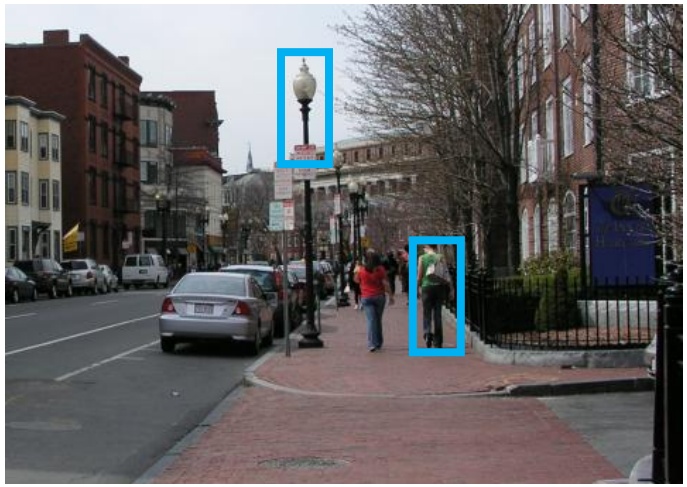
Hoiem et al. 2006

Object detection in computer vision

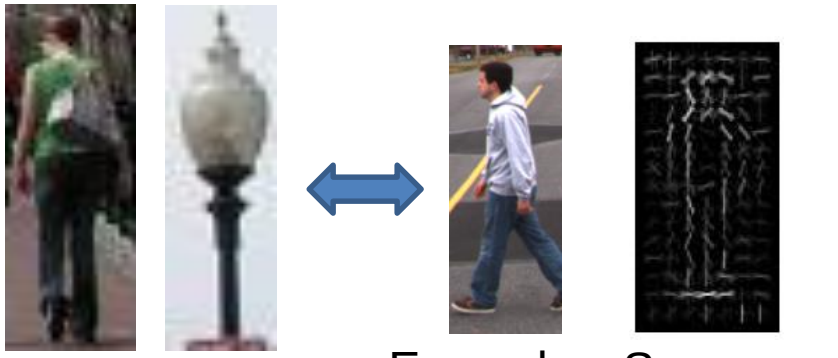
Goal: detect all pedestrians, cars, monkeys, etc in image

Basic Steps of Category Detection

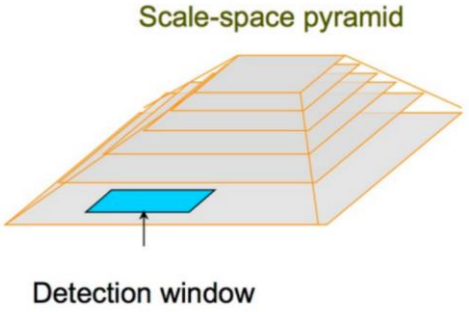
1. Align
   - E.g., choose position, scale orientation
   - How to make this tractable?



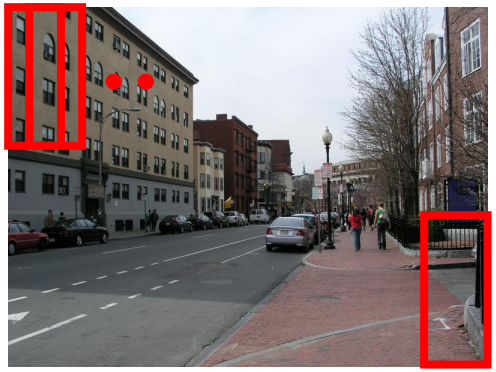
2. Compare
   - Compute similarity to an example object or to a summary representation
   - Which differences in appearance are important?



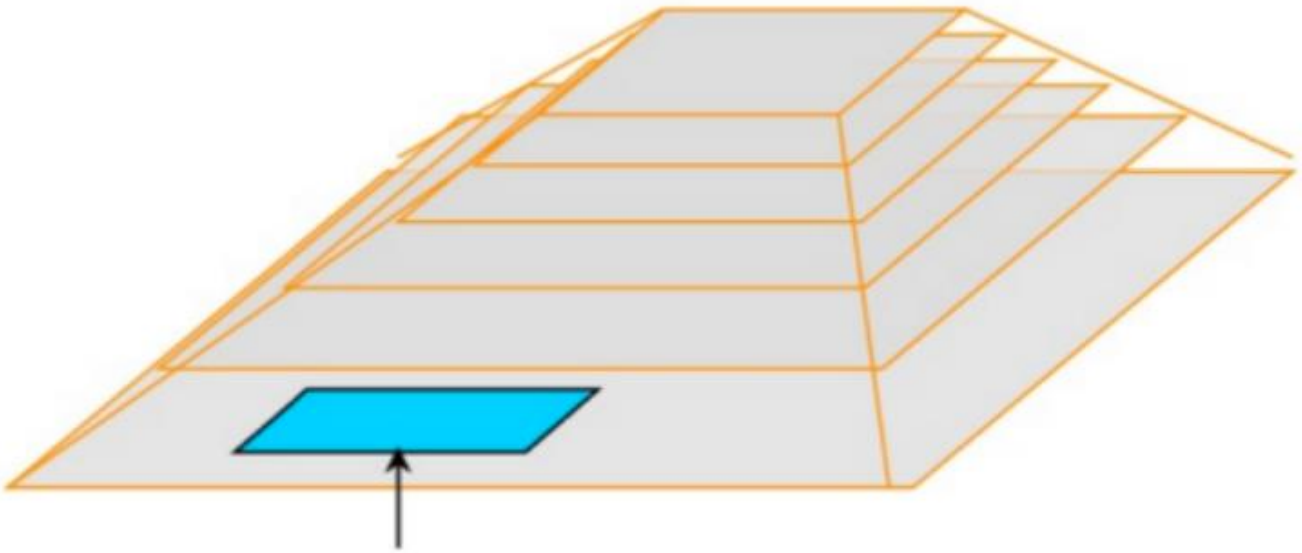
Sliding window: a simple alignment solution



scales



Scale-space Pyramid



Detection window

Each window is separately classified

Statistical Template

- Object model = sum of scores of features at fixed positions

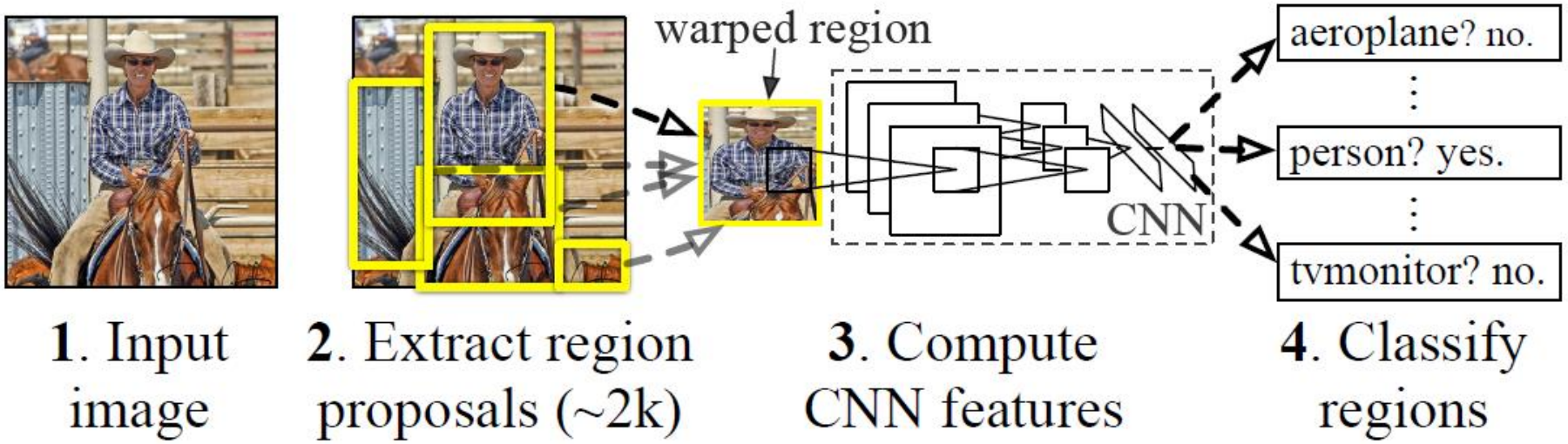
+3 +2 -2 -1 -2.5 = -0.5 ? > 7.5

Non-object

+4 +1 +0.5 +3 +0.5 = 10.5 ? > 7.5

Object

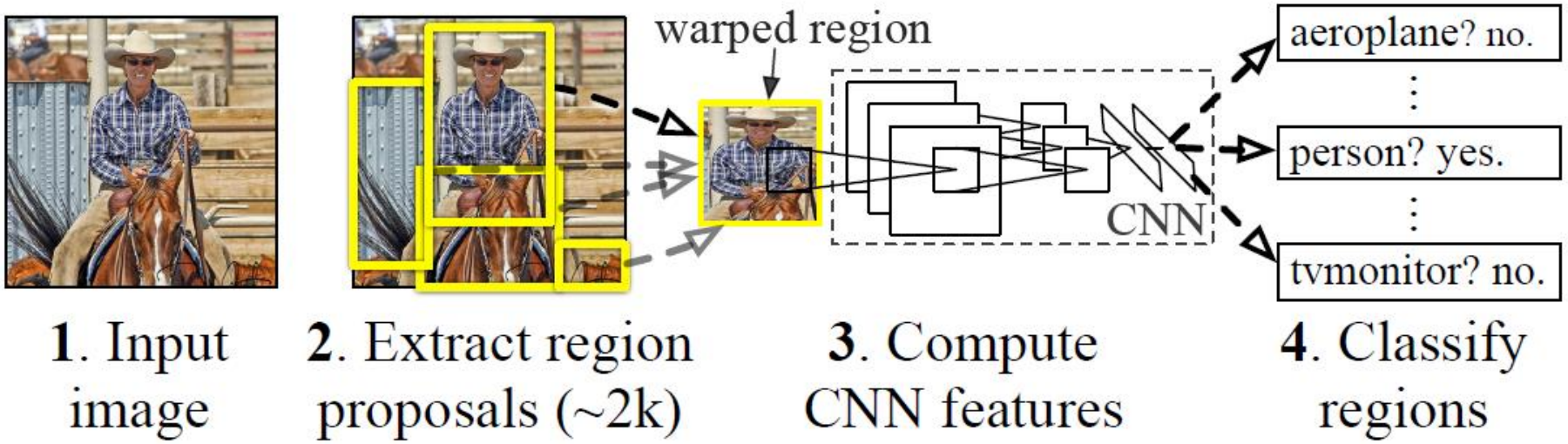
R-CNN (Girshick et al. CVPR 2014)



- Replace sliding windows with “selective search” region proposals (Uijilings et al. IJCV 2013)
- Extract rectangles around regions and resize to 227x227
- Extract features with fine-tuned CNN (that was initialized with network trained on ImageNet before training)
- Classify last layer of network features with SVM

http://arxiv.org/pdf/1311.2524.pdf

R-CNN (Girshick et al. CVPR 2014)



- 產生一群約 2000 個可能的區域 (Region Proposals)
- 經由一個預先訓練好的 CNN 模型如 AlexNet 擷取特徵，將結果儲存起來。
- 然後再以 SVM (Support Vector Machine) 分類器來區分是否為物體或者背景。
- 最後經由一個線性回歸模型來校正 bounding box 位置。

Regions from selective search

Fine-tuning example: ImageNet->VOC

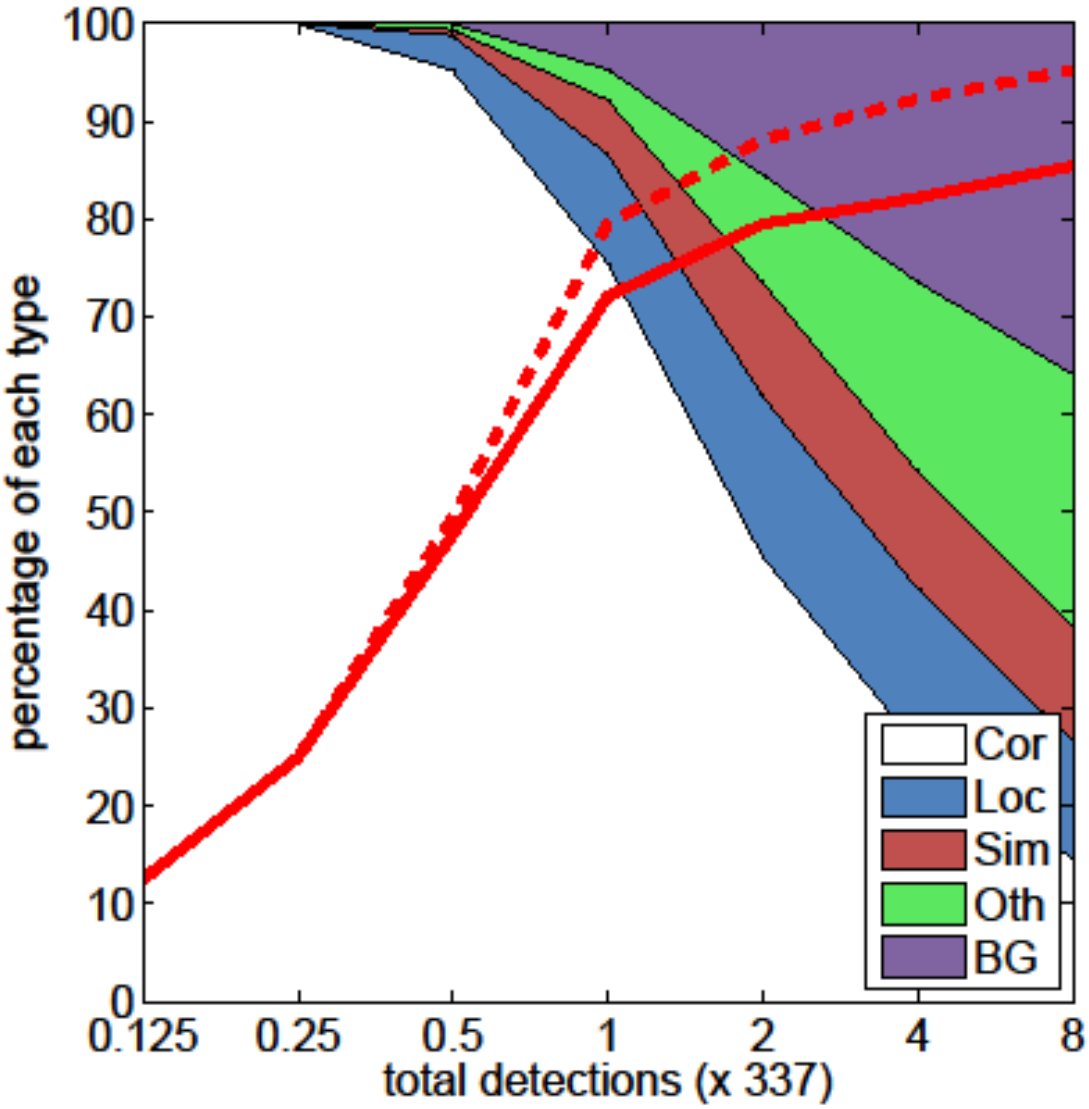
1. Train full network on ImageNet 1000-class classification
2. Replace classification layer with output layer for VOC (e.g. confidences for 20 classes)
3. Train on VOC pos/neg examples with low initial learning rate (1/10th what is used for new network)

Notes

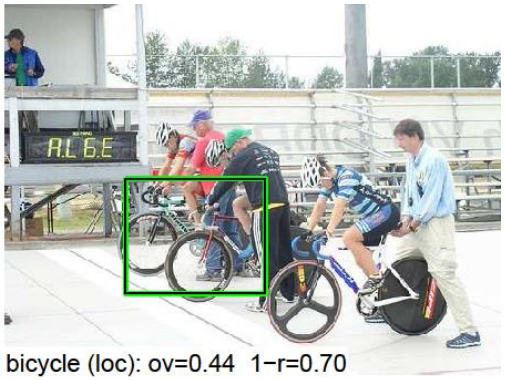
- This usually works well if the "big data" task and target task are similar (object classification vs detection)
  - 0.45 AP without fine-tuning → 0.54 AP with fine tuning; training only on VOC does much worse
- Not necessary if target task is also very big

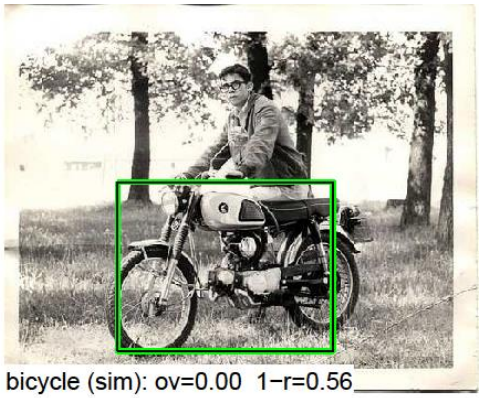
Mistakes are often reasonable

Bicycle: AP = 0.73



Confident Mistakes

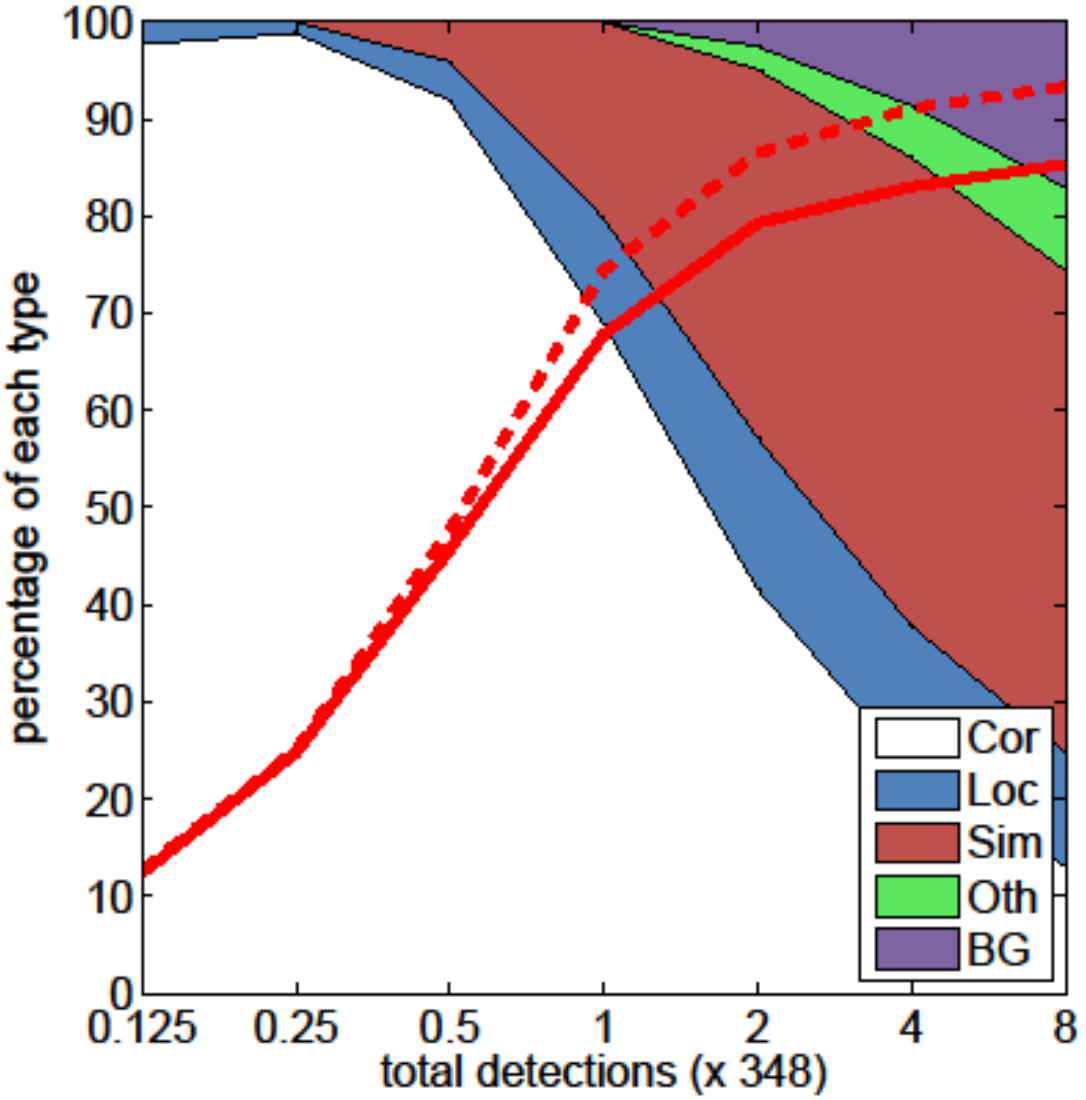




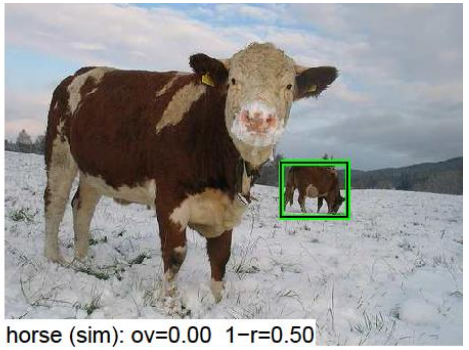
R-CNN results

Mistakes are often reasonable

Horse: AP = 0.69

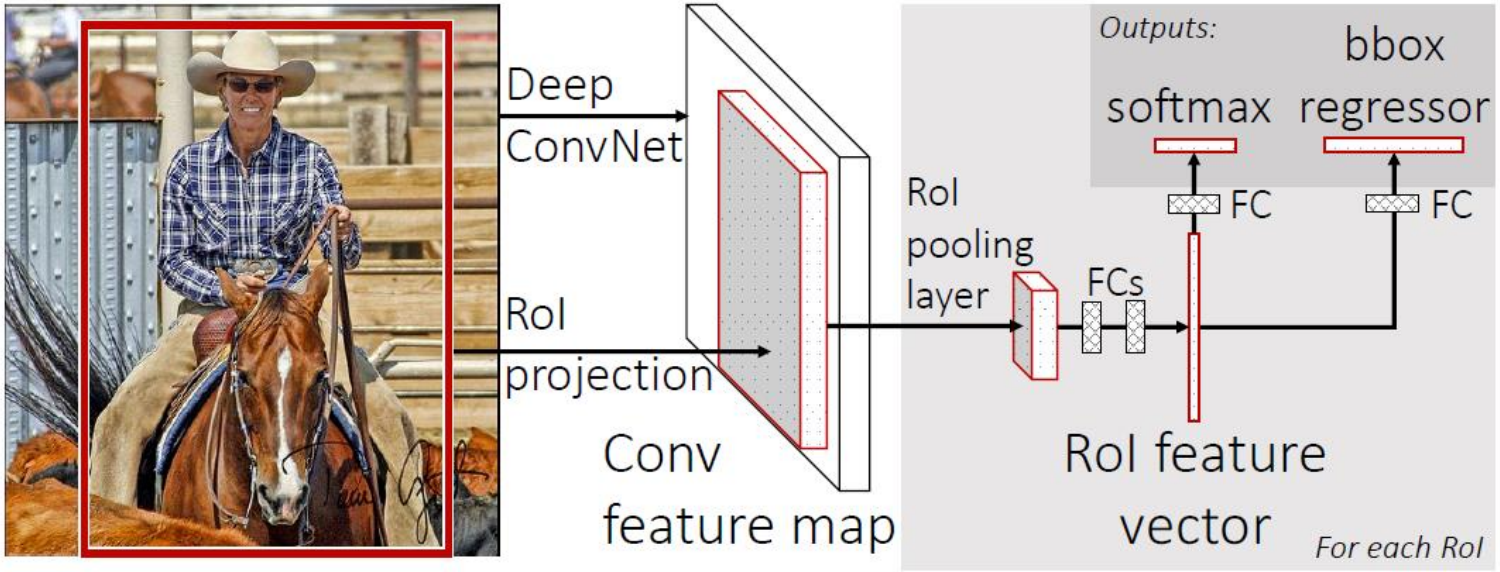


Confident Mistakes



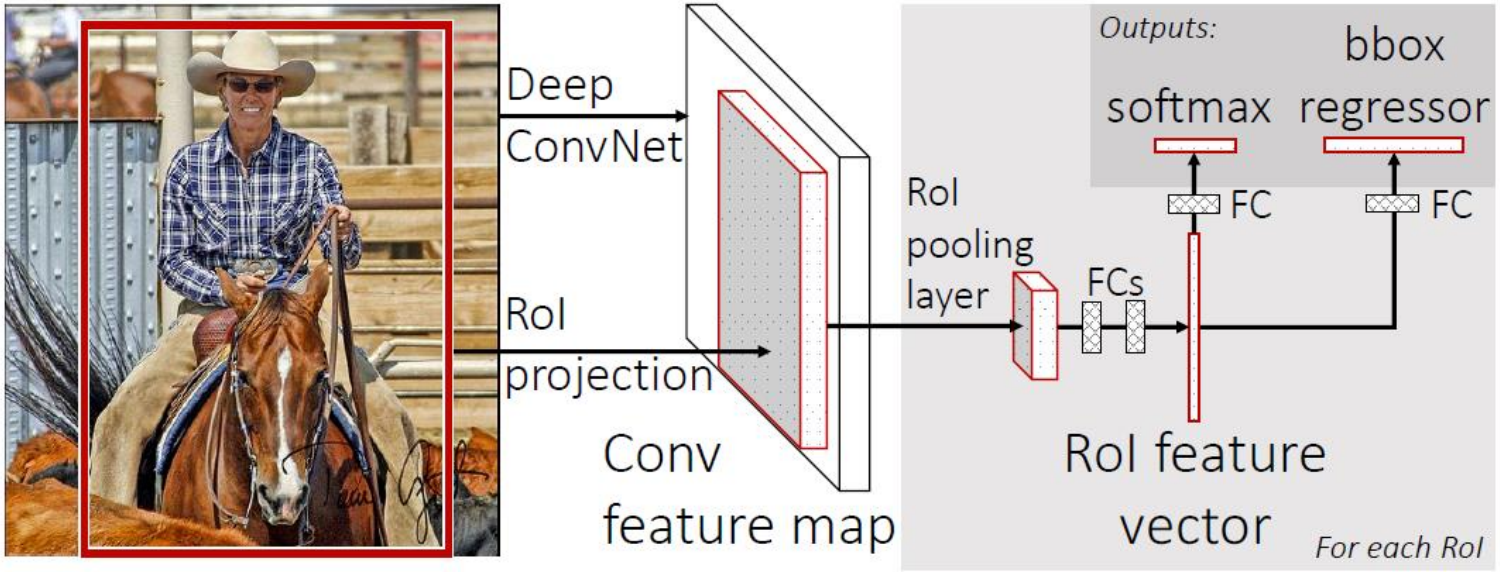
R-CNN results

Fast R-CNN – Girshick 2015



- Compute CNN features for image once
- Pool into 7x7 spatial bins for each region proposal, output class scores and regressed bboxes
- 100x speed up of R-CNN (0.02 – 0.1 FPS → 0.5-20 FPS) with similar accuracy

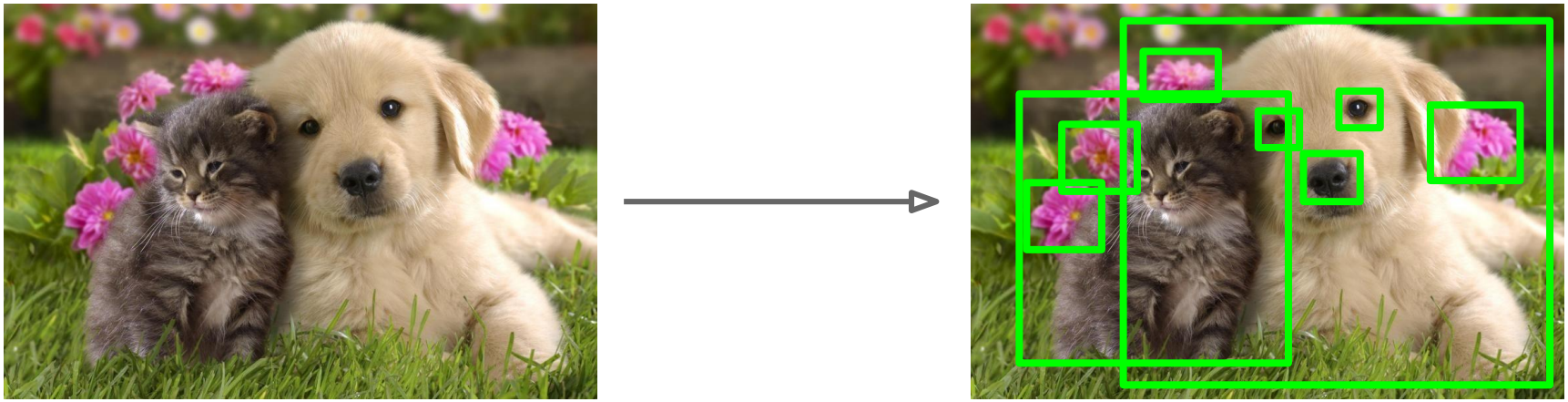
Fast R-CNN – Girshick 2015



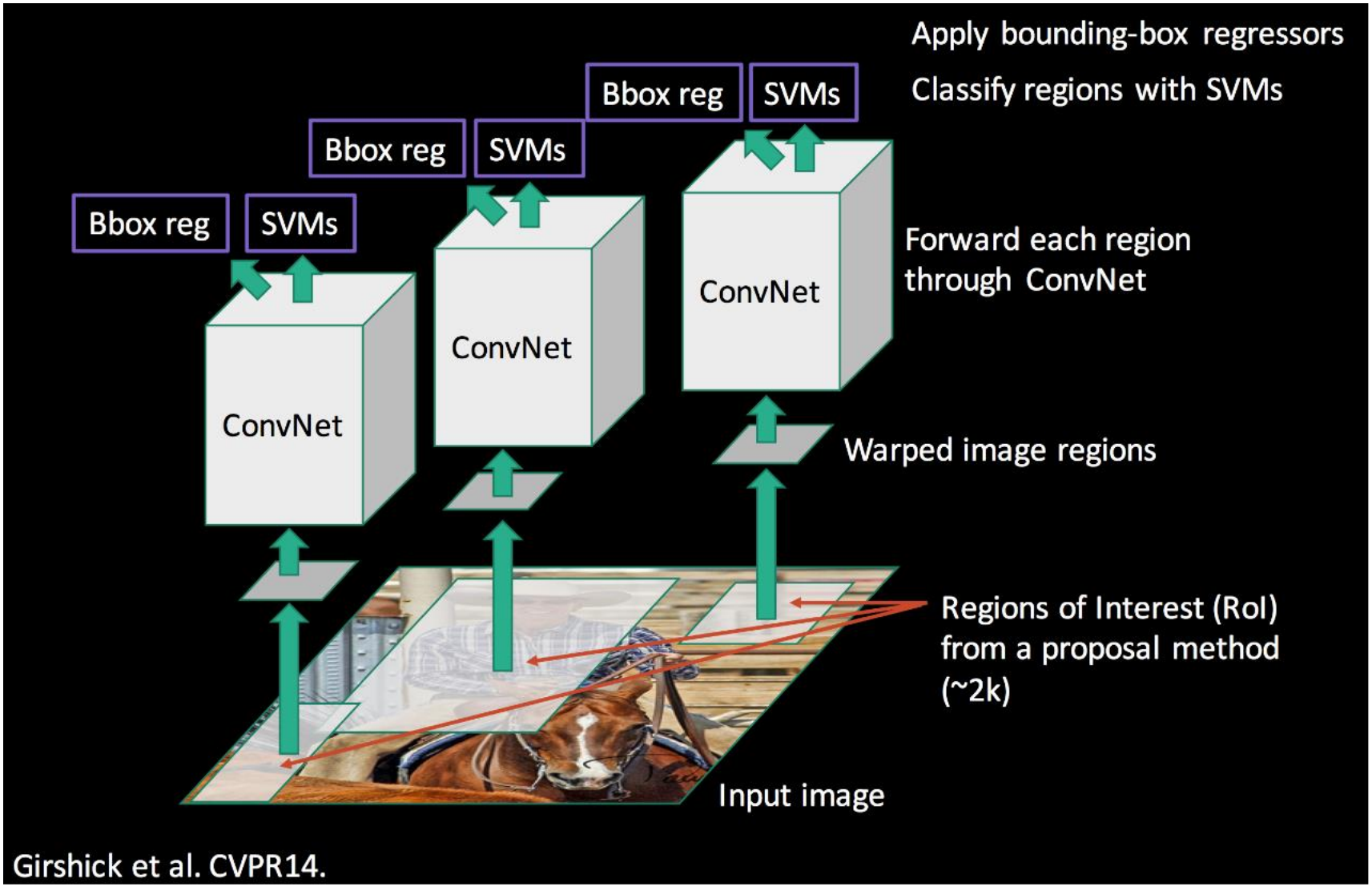
- Fast R-CNN : 在 R-CNN 中，2000 多個區域都要個別去運算 CNN，這些區域很多都是重疊的，也就是說這些重疊區域的 CNN 很多都是重複算的。所以 Fast R-CNN 的原則就是全部只算一次 CNN 就好，CNN 擷取出來的特徵可以讓這 2000 多個區域共用！
- Fast R-CNN: RoIPooling (Region of Interest Pooling)

Region Proposals

- Find “blobby” image regions that are likely to contain objects
- “Class-agnostic” object detector
- Look for “blob-like” regions

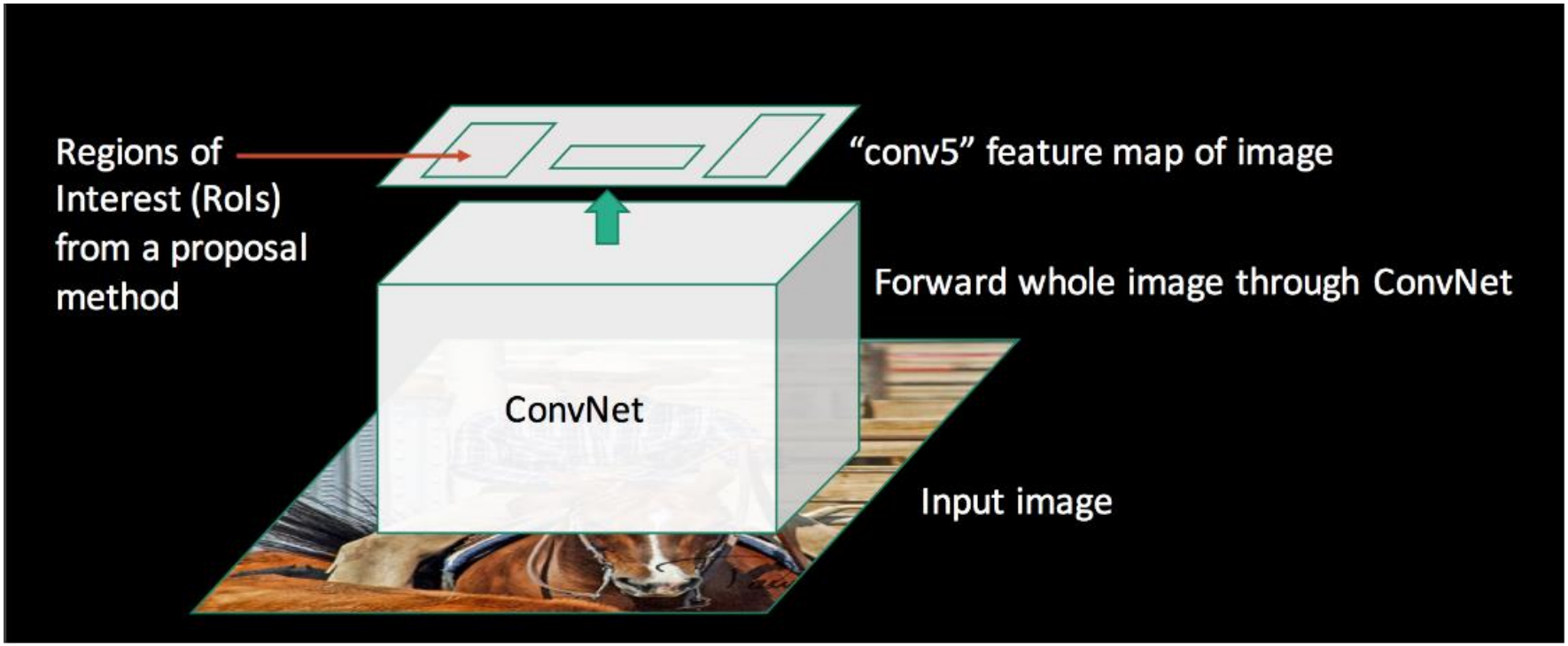


R-CNN



Number of CNN runs = number of proposals → high latency

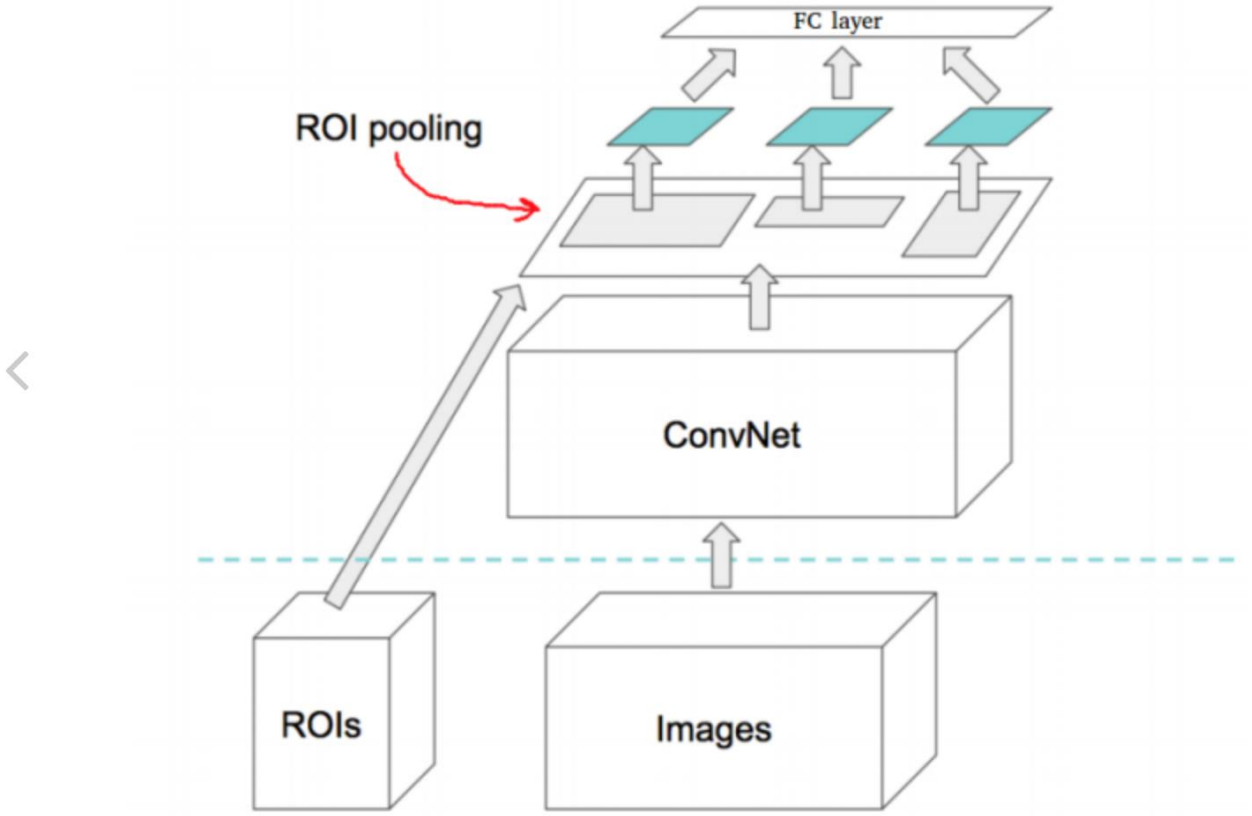
Fast R-CNN (Girschick et. al. 2015)



1 CNN run for whole image

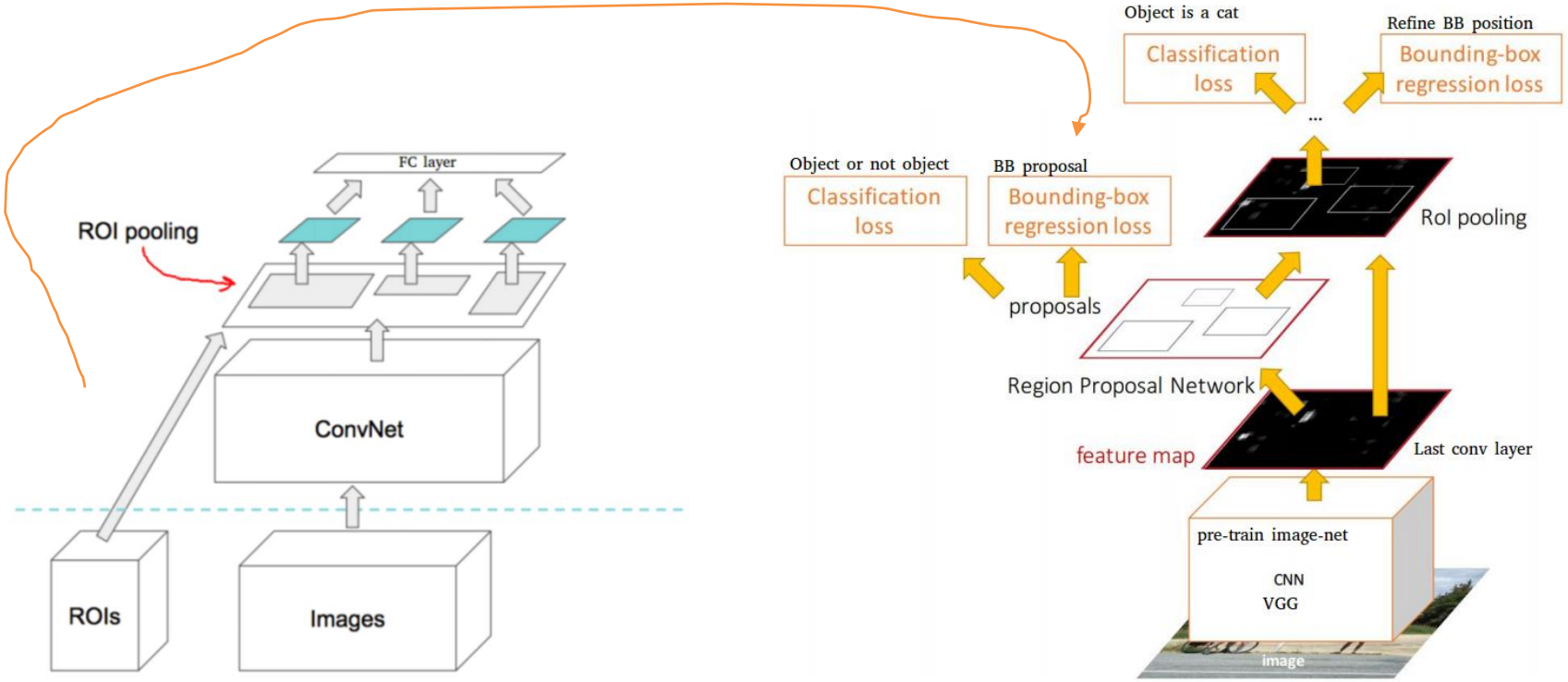
proposed regions extracted from CNN feature map, instead of image

New operation: Region of Interest (ROI) pooling



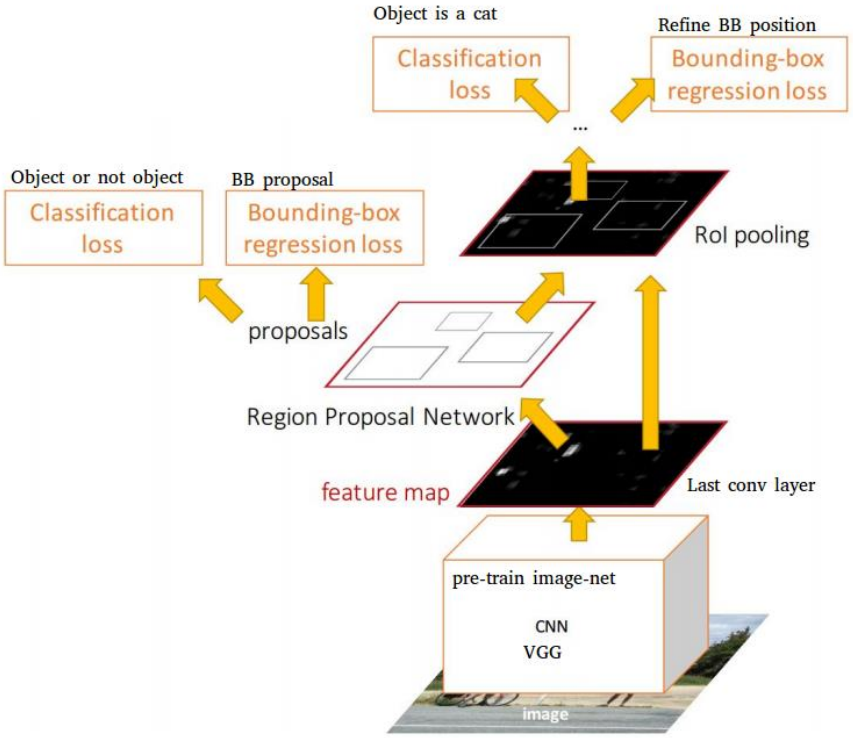
Necessary to convert variable sized ROIs to fixed size inputs expected by the following FC layers
(similar to warping original image patches in R-CNN for fixed sized inputs to conv layers)

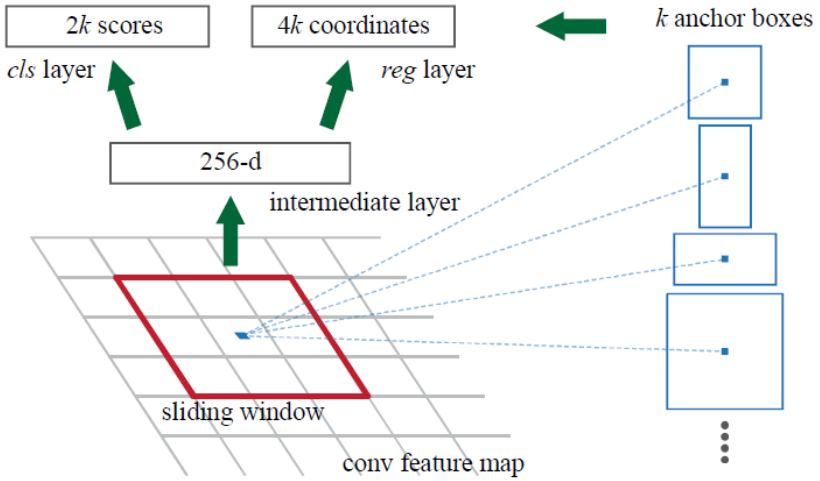
Faster R-CNN – Ren et al. 2016



- Faster R-CNN 的直覺想法: 與其預先篩選 region proposals，到不如從 CNN 的 feature map 上選出 region proposals。

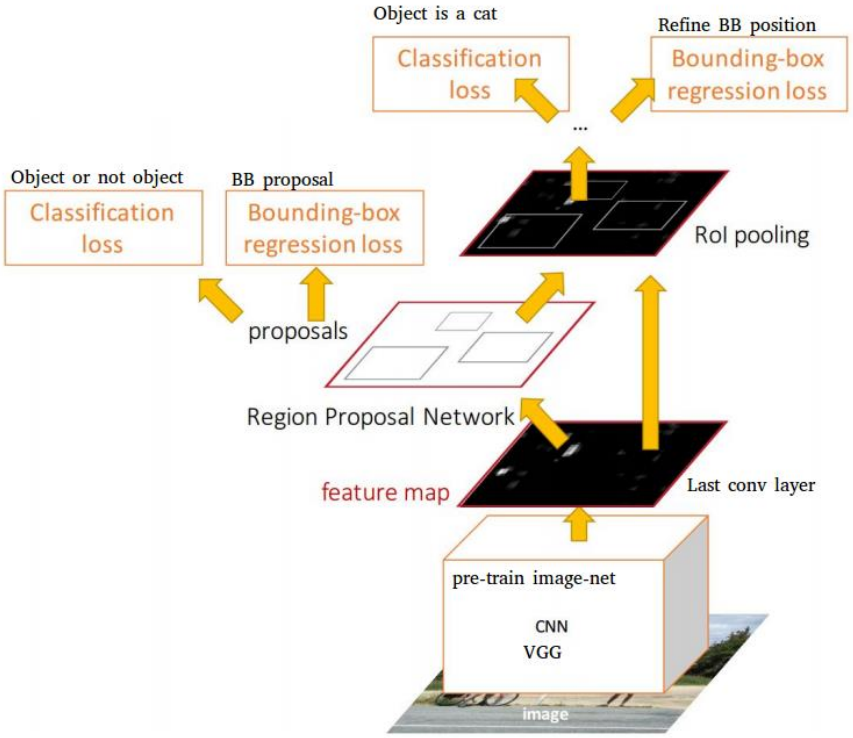
Faster R-CNN – Ren et al. 2016

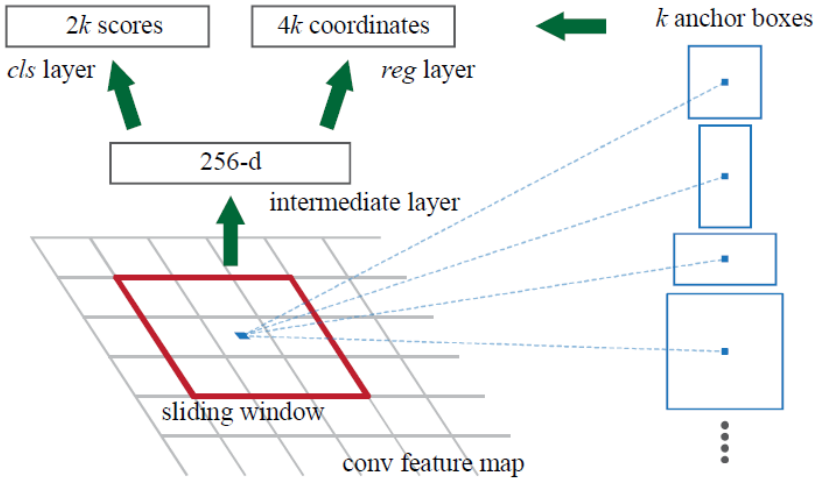




- Faster R-CNN 的直覺想法: 利用一個region proposal network 去產生region candidates

Faster R-CNN – Ren et al. 2016



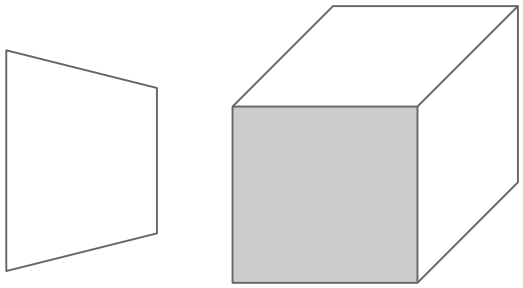


- Convolutional features used for generating proposals and scoring
  - Generate proposals with “objectness” scores and refined bboxes for each of k “anchors”
  - Score proposals in same way as Fast R-CNN
- Similar accuracy to Fast R-CNN with 10x speedup

Fast R-CNN: Region of Interest Pooling

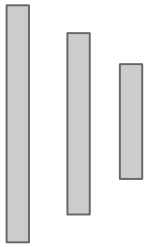
Hi-res input image:
3 x 800 x 600
with region
proposal

Convolution
and Pooling



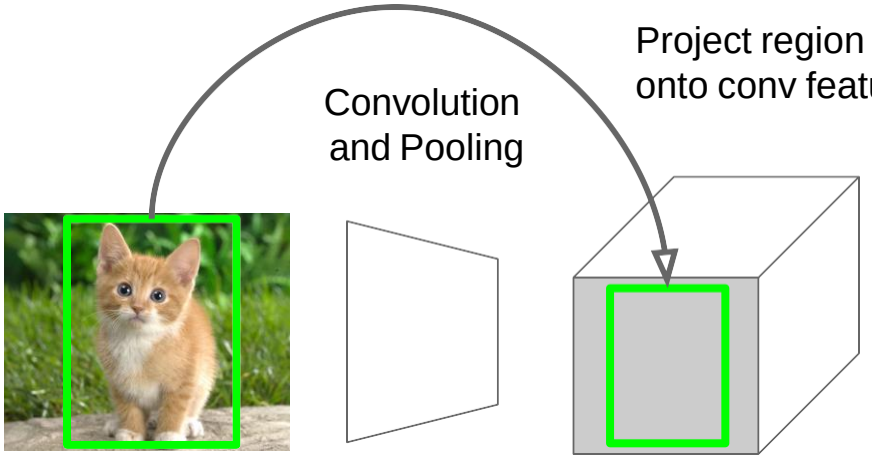
Hi-res conv features:
C x H x W
with region proposal

Fully-connected
layers



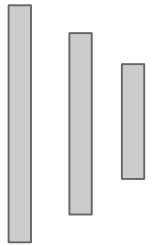
Problem: Fully-connected layers expect low-res conv features: C x h x w

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

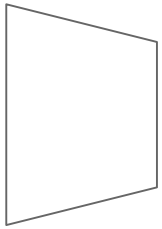


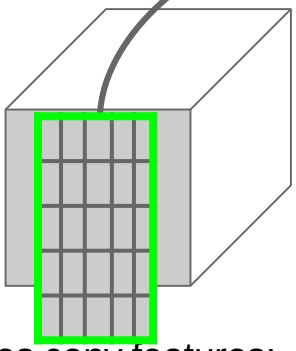
Problem: Fully-connected layers expect low-res conv features: C x h x w

Fast R-CNN: Region of Interest Pooling

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

Convolution
and Pooling

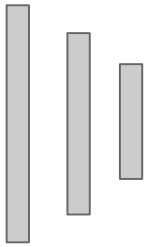




Hi-res conv features:
C x H x W
with region proposal

Divide projected
region into h x w grid

Fully-connected
layers

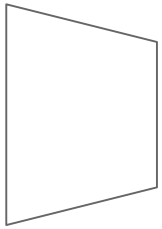


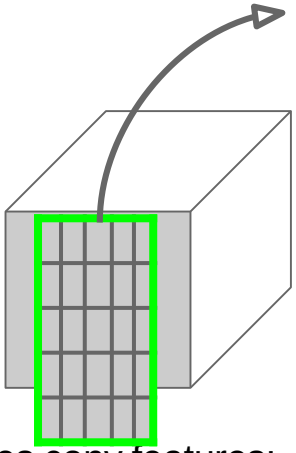
Problem: Fully-connected layers expect low-res conv features: C x h x w

Fast R-CNN: Region of Interest Pooling

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

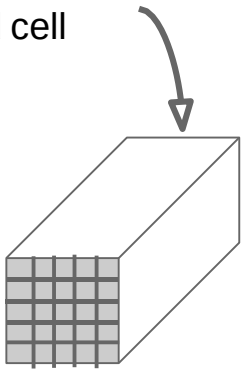
Convolution
and Pooling





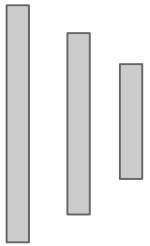
Hi-res conv features:
C x H x W
with region proposal

Max-pool within
each grid cell



RoI conv features:
C x h x w
for region proposal

Fully-connected
layers

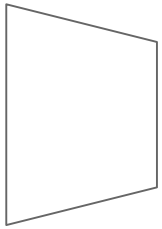


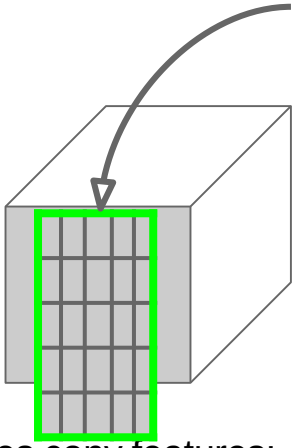
Fully-connected layers expect
low-res conv features:
C x h x w

Fast R-CNN: Region of Interest Pooling

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

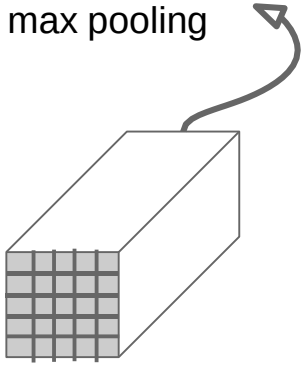
Convolution
and Pooling





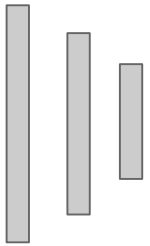
Hi-res conv features:
C x H x W
with region proposal

Can back propagate
similar to max pooling



RoI conv features:
C x h x w
for region proposal

Fully-connected
layers



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

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

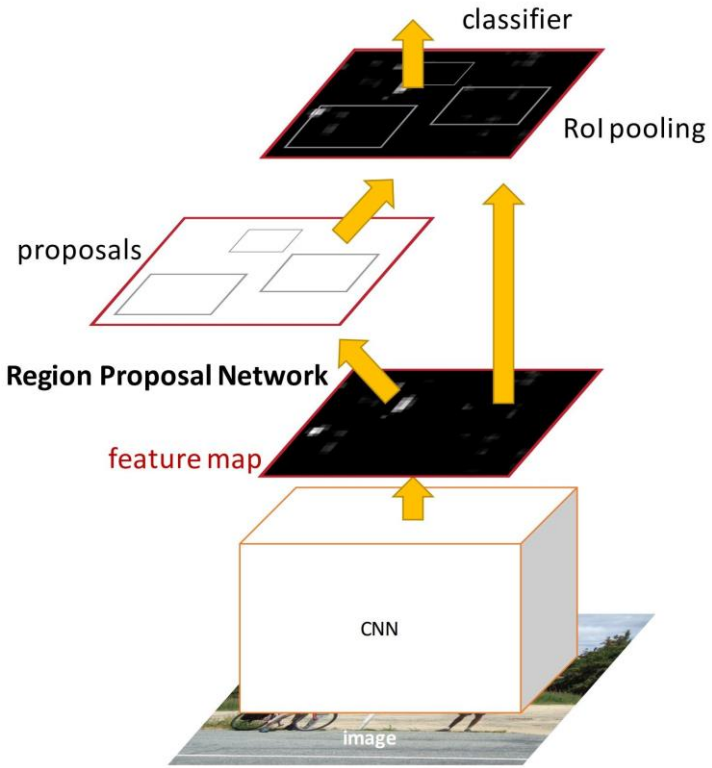
	R-CNN	**Fast R-CNN**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
Test time per image with Selective Search	50 seconds	**2 seconds**
(Speedup)	1x	**25x**

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

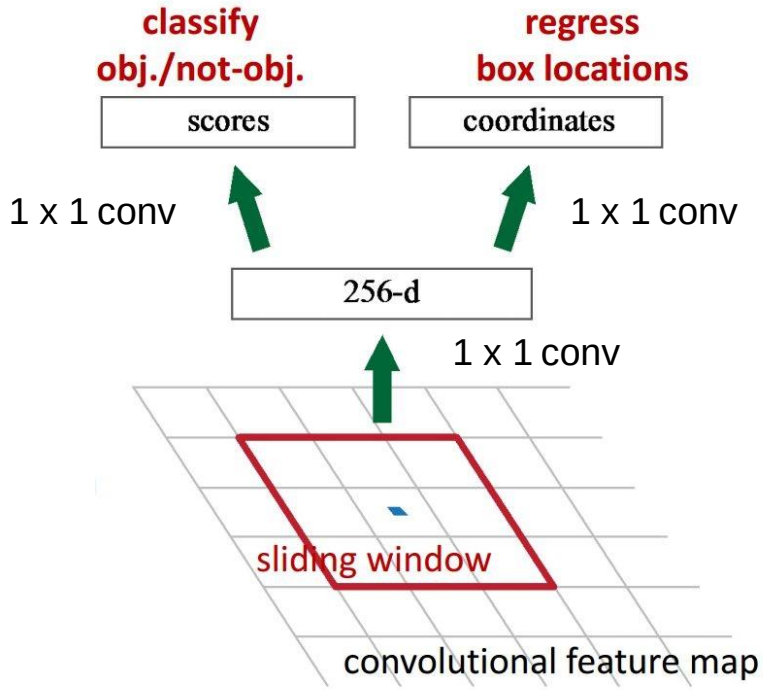
Faster R-CNN: Region Proposal Network

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

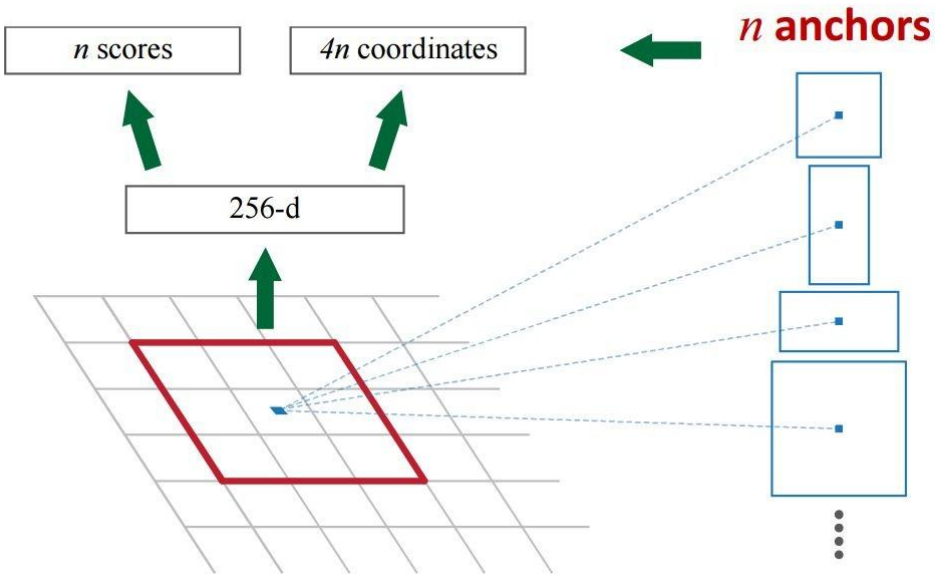
Faster R-CNN: Region Proposal Network

Use **N anchor boxes** at each location

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

Regression gives offsets from anchor boxes

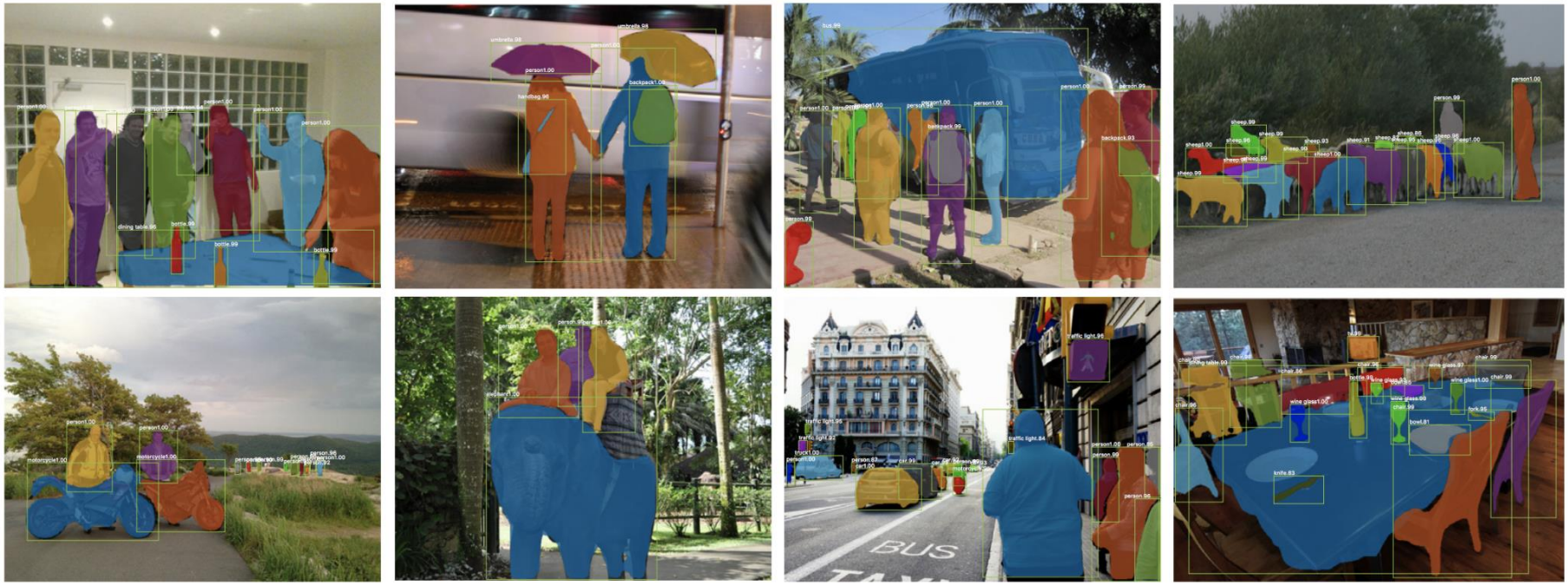
Classification gives the probability that each (regressed) anchor shows an object



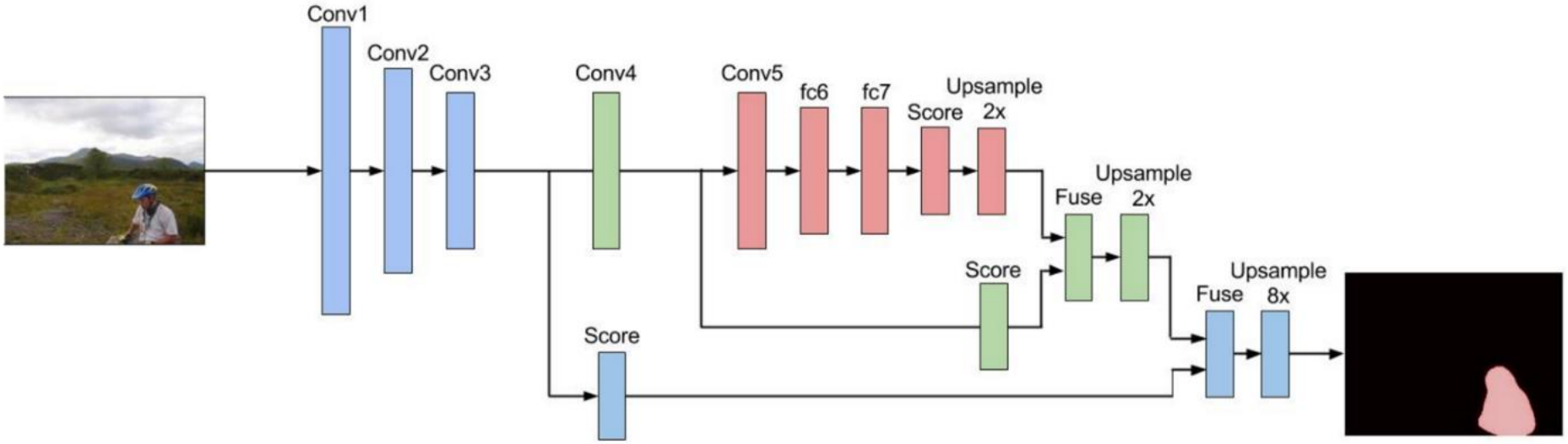
Faster R-CNN: Results

	R-CNN	**Fast R-CNN**	**Faster R-CNN**
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

MASK R-CNN



MASK R-CNN


Conv1
Conv2
Conv3
Conv4
Conv5
fc6
fc7
Score
Upsample 2x
Fuse
Upsample 2x
Score
Fuse
Upsample 8x
Score

One stage method for object detection

Joseph Redmon
Ross Girshick
Santosh Divvala
Ali Farhadi

Most Accurate Real-Time Detector 2016

Fastest Object Detector in the Literature 2016

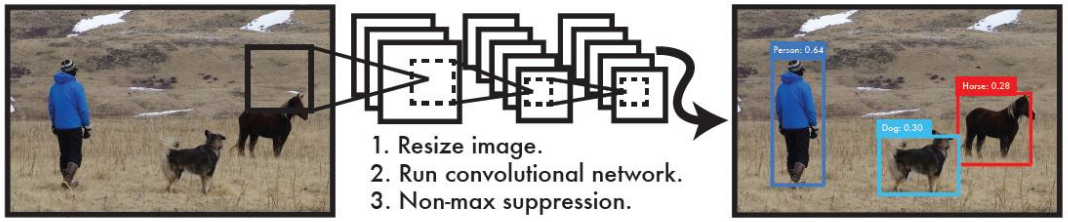
"You Only Look Once"

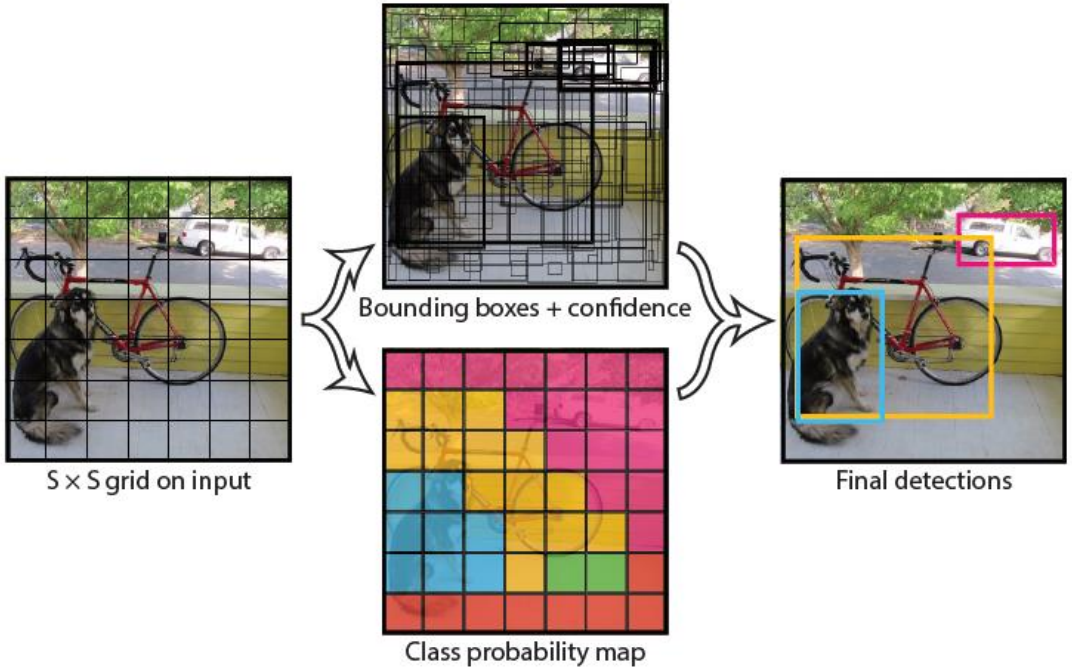
Real-Time Detection

YOLO – Redmon et al. 2016

1. CNN produces 4096 features for 7x7 grid on image (fully conv.)
2. Each cell produces a score for each object and 2 bboxes w/ conf
3. Non-max suppression

- 7x speedup over Faster RCNN (45-155 FPS vs. 7-18 FPS)
- Some loss of accuracy due to lower recall, poor localization





Accurate object detection is slow!

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img

176 feet

Accurate object detection is slow!

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img

Accurate object detection is slow!

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	63.4	45 FPS	22 ms/img

2 feet

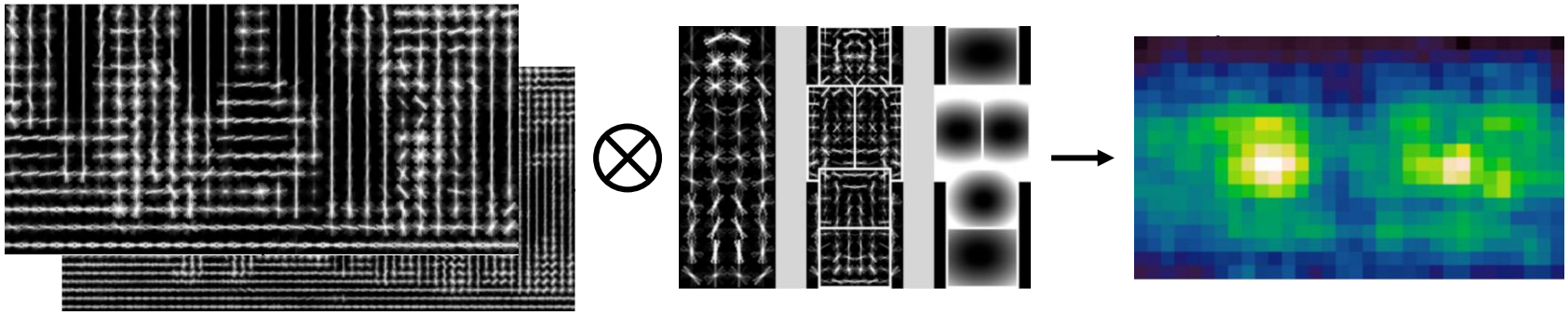
Accurate object detection is slow!

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	~~63.4~~ 69.0	45 FPS	22 ms/img

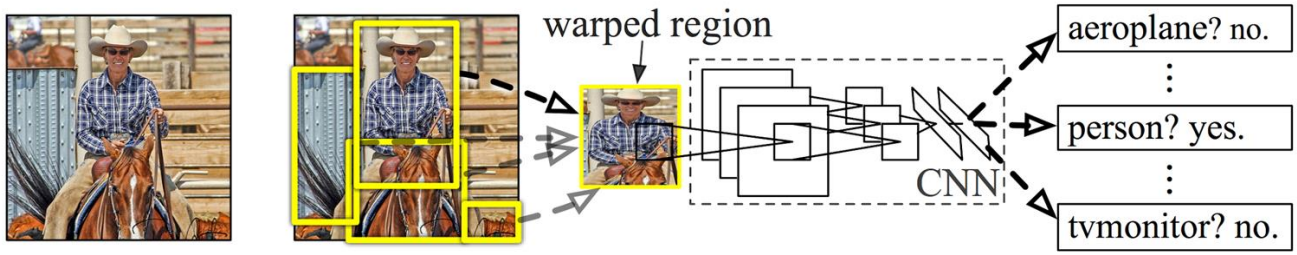
2 feet

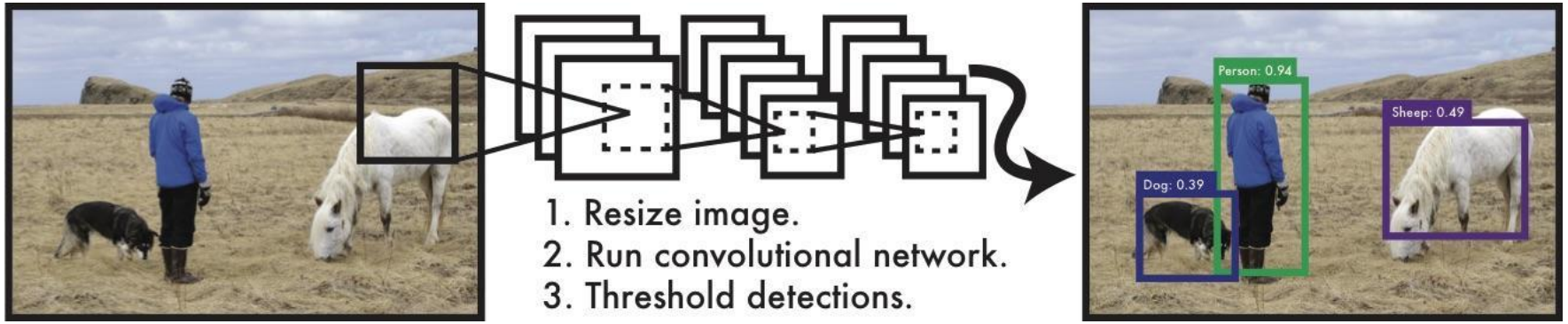
Sliding window, DPM, R-CNN all train region-based classifiers to perform detection

DPM: *Deformable Part Models*

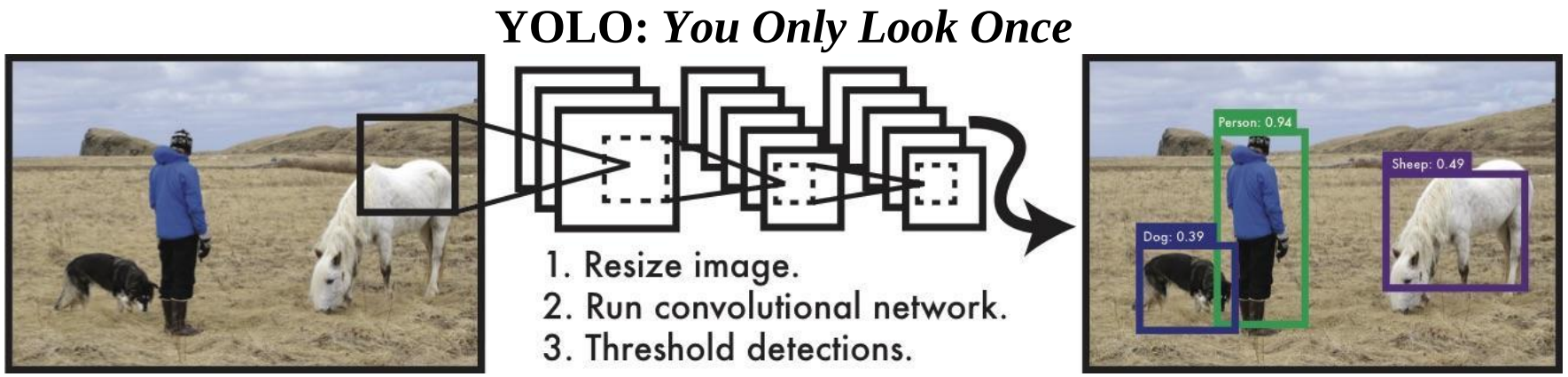


R-CNN: *Regions with CNN features*

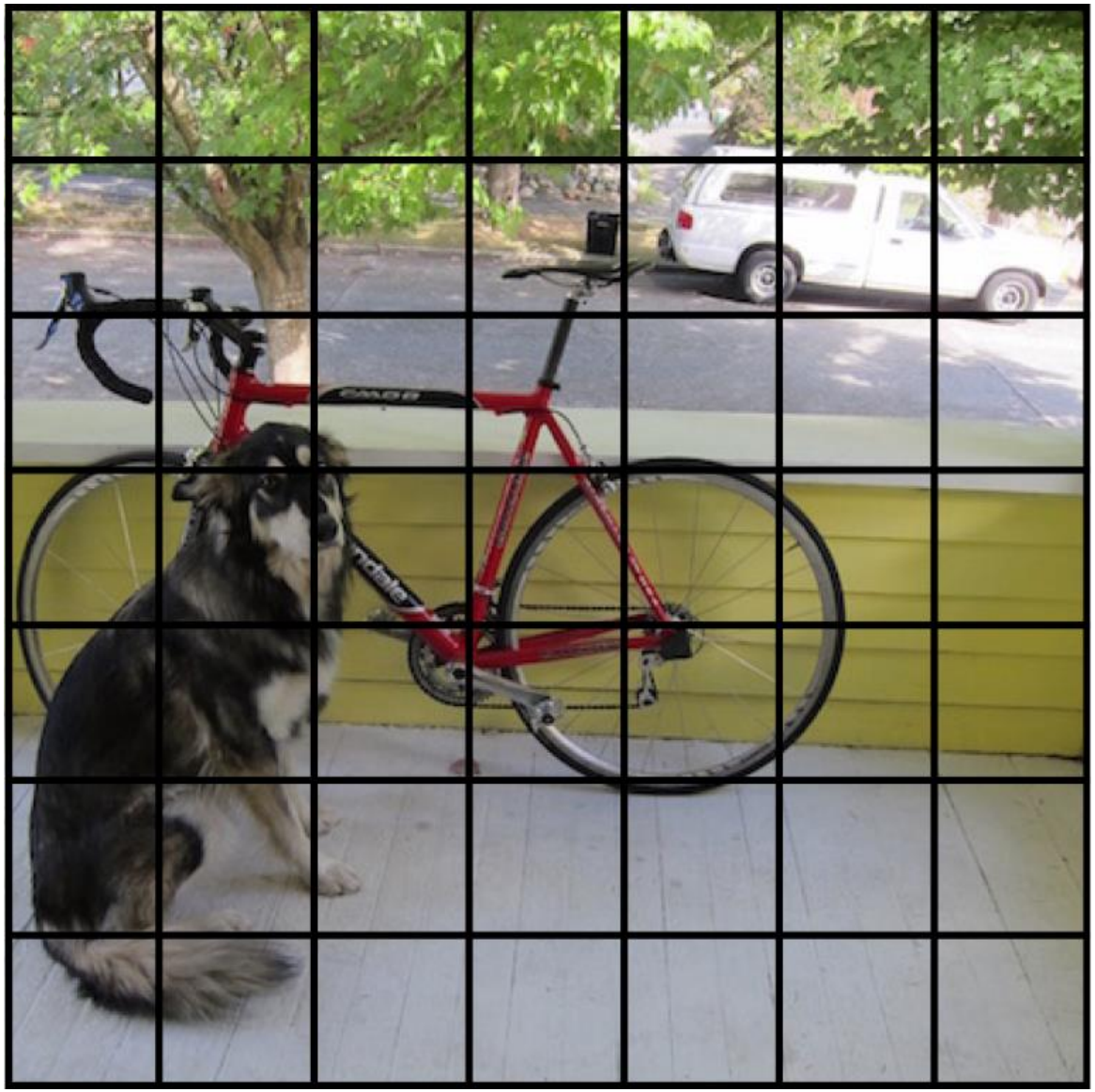



1. Resize image.
2. Run convolutional network.
3. Threshold detections.
Person: 0.94
Sheep: 0.49
Dog: 0.39

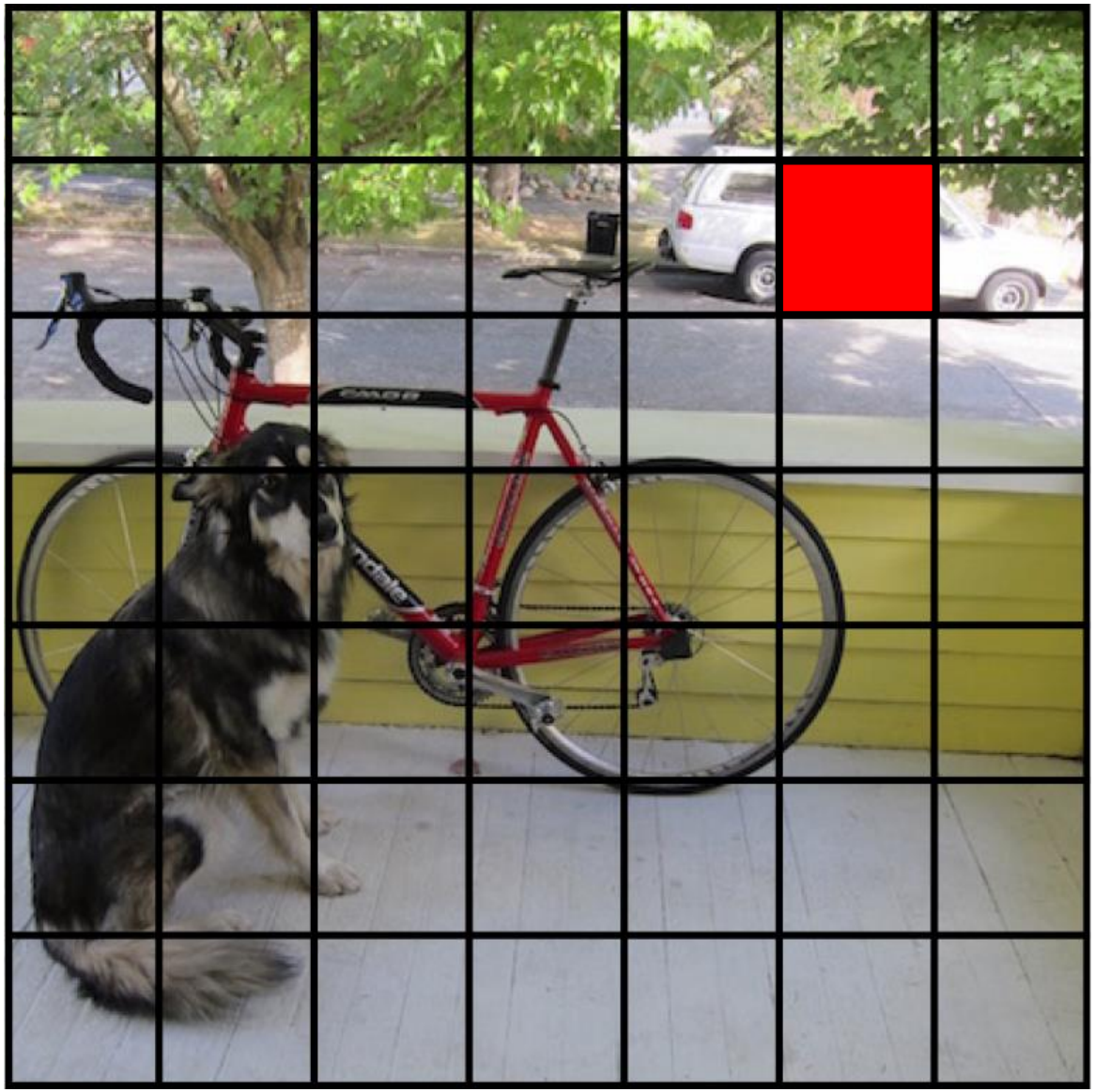
With YOLO, you only look once at an image to perform detection



We split the image into a grid

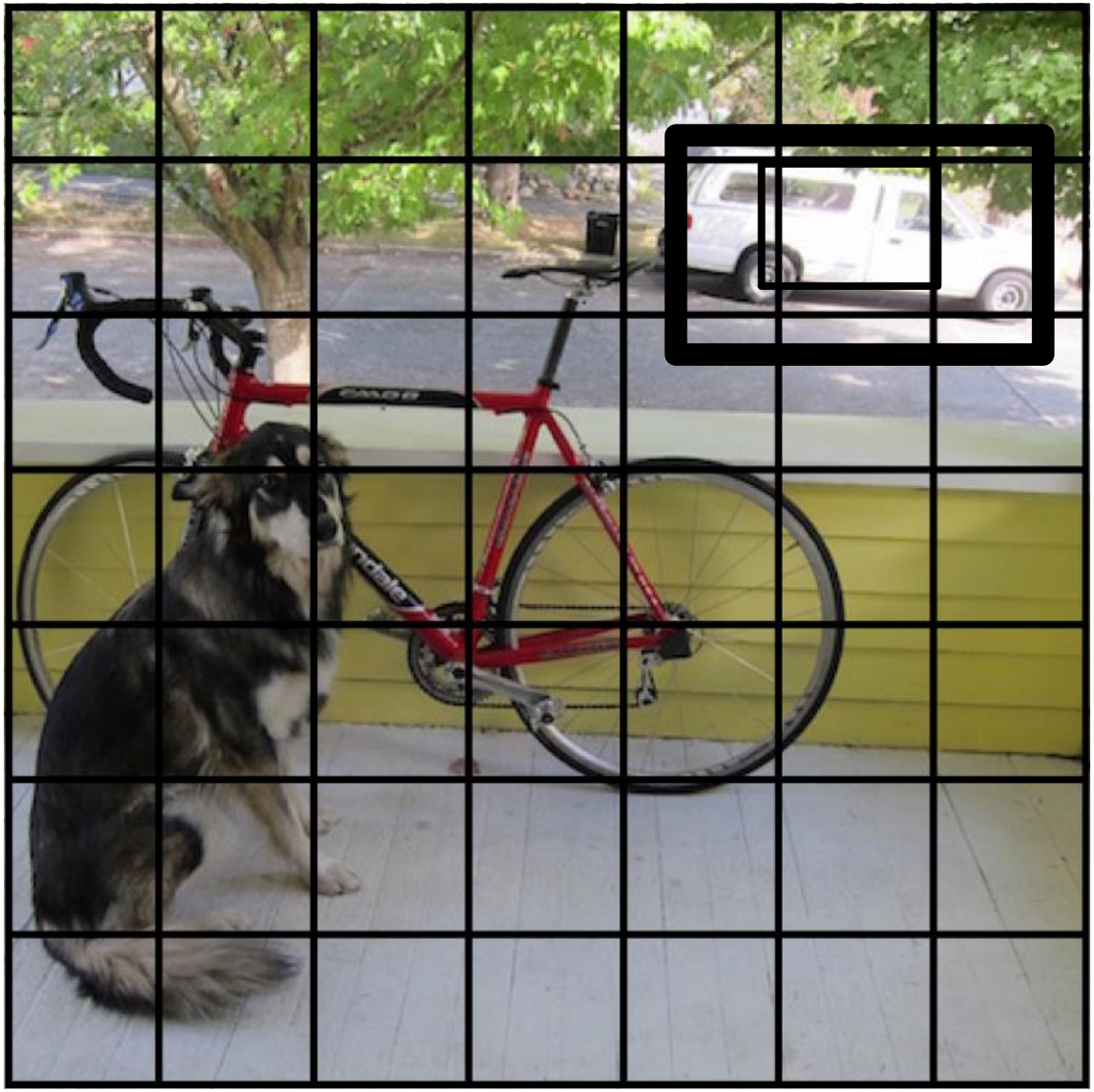


Each cell predicts boxes and confidences: P(Object)

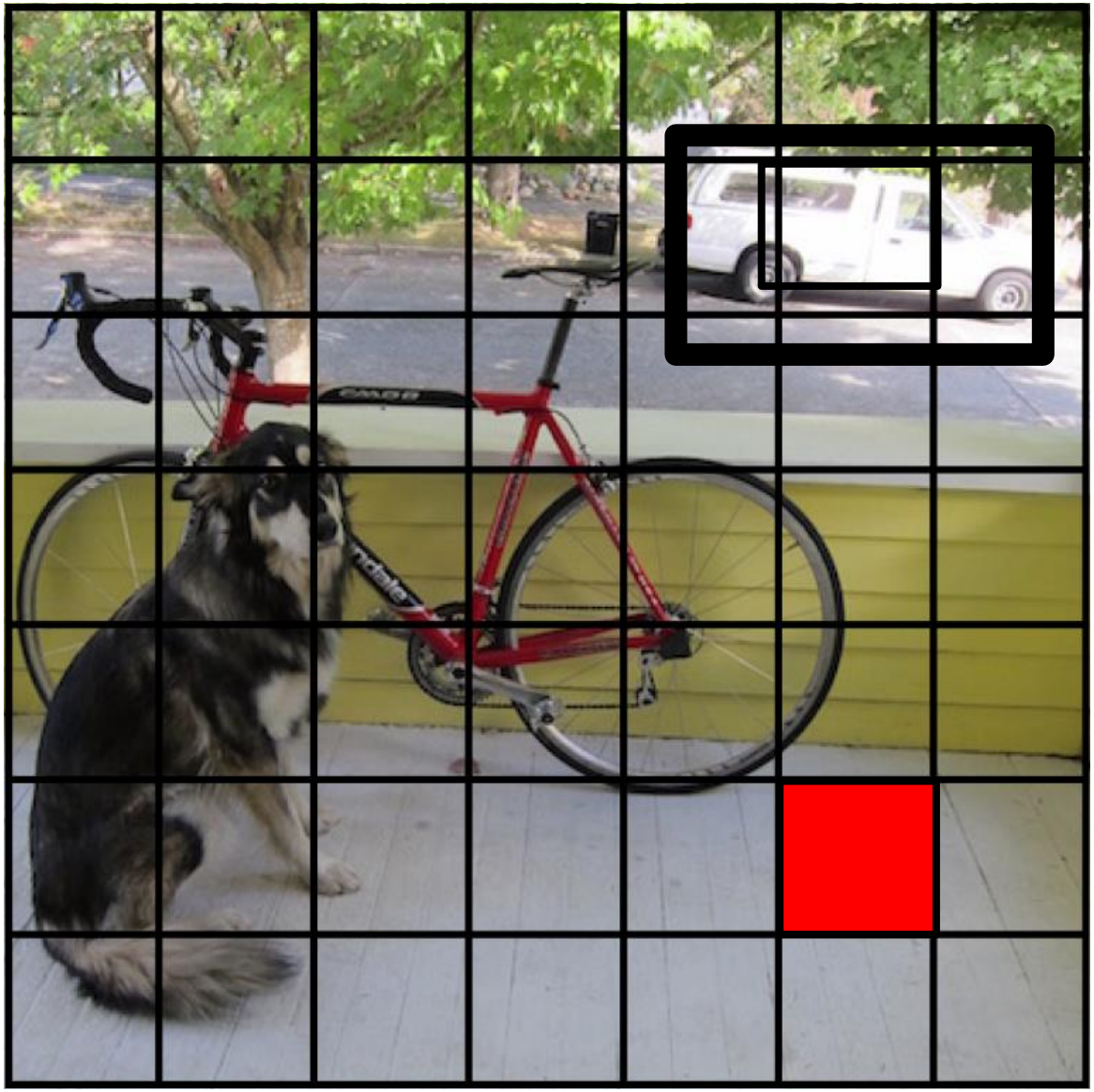


Each cell predicts boxes and confidences: P(Object)

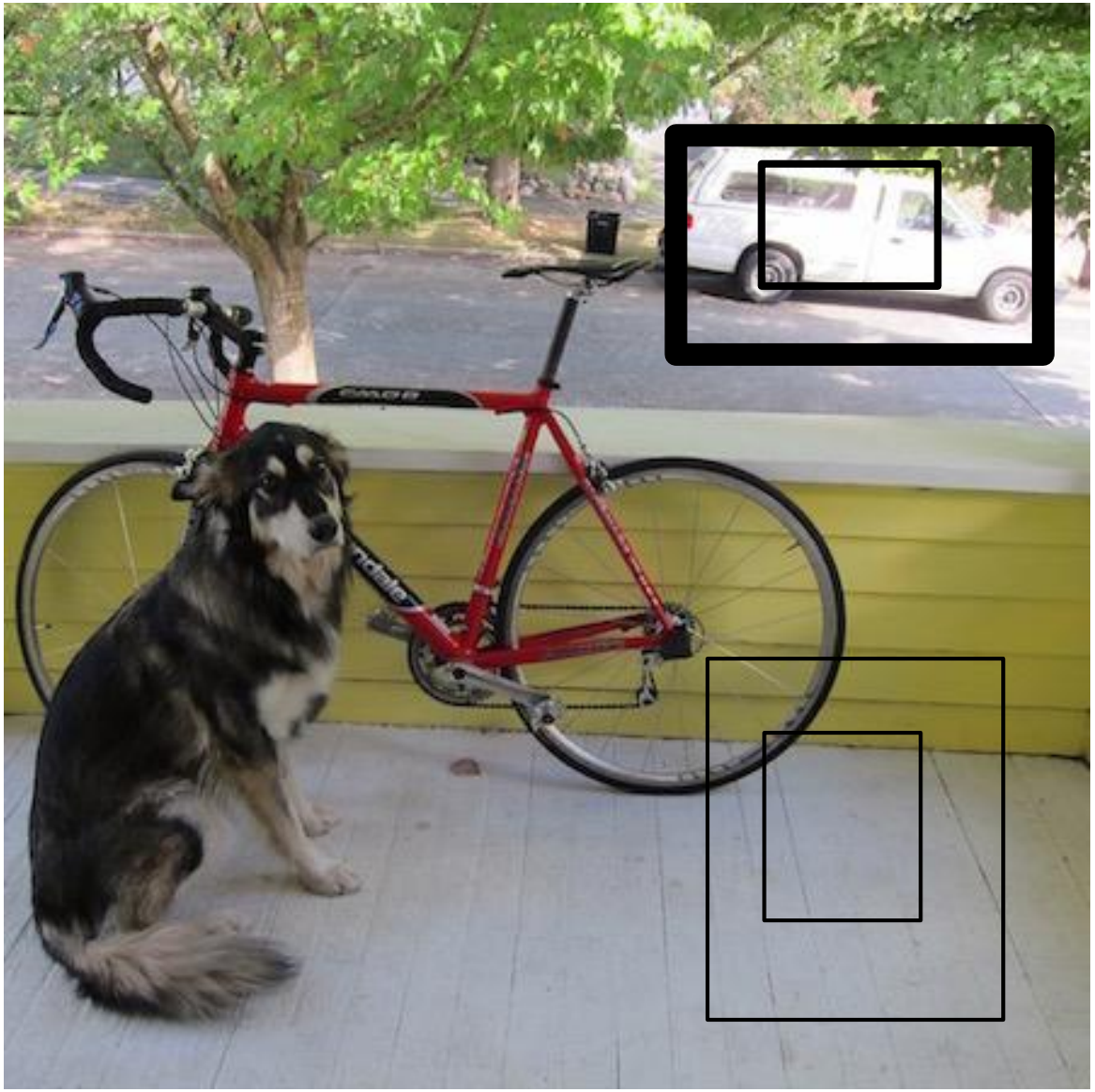
Each cell predicts boxes and confidences: P(Object)



Each cell predicts boxes and confidences: P(Object)

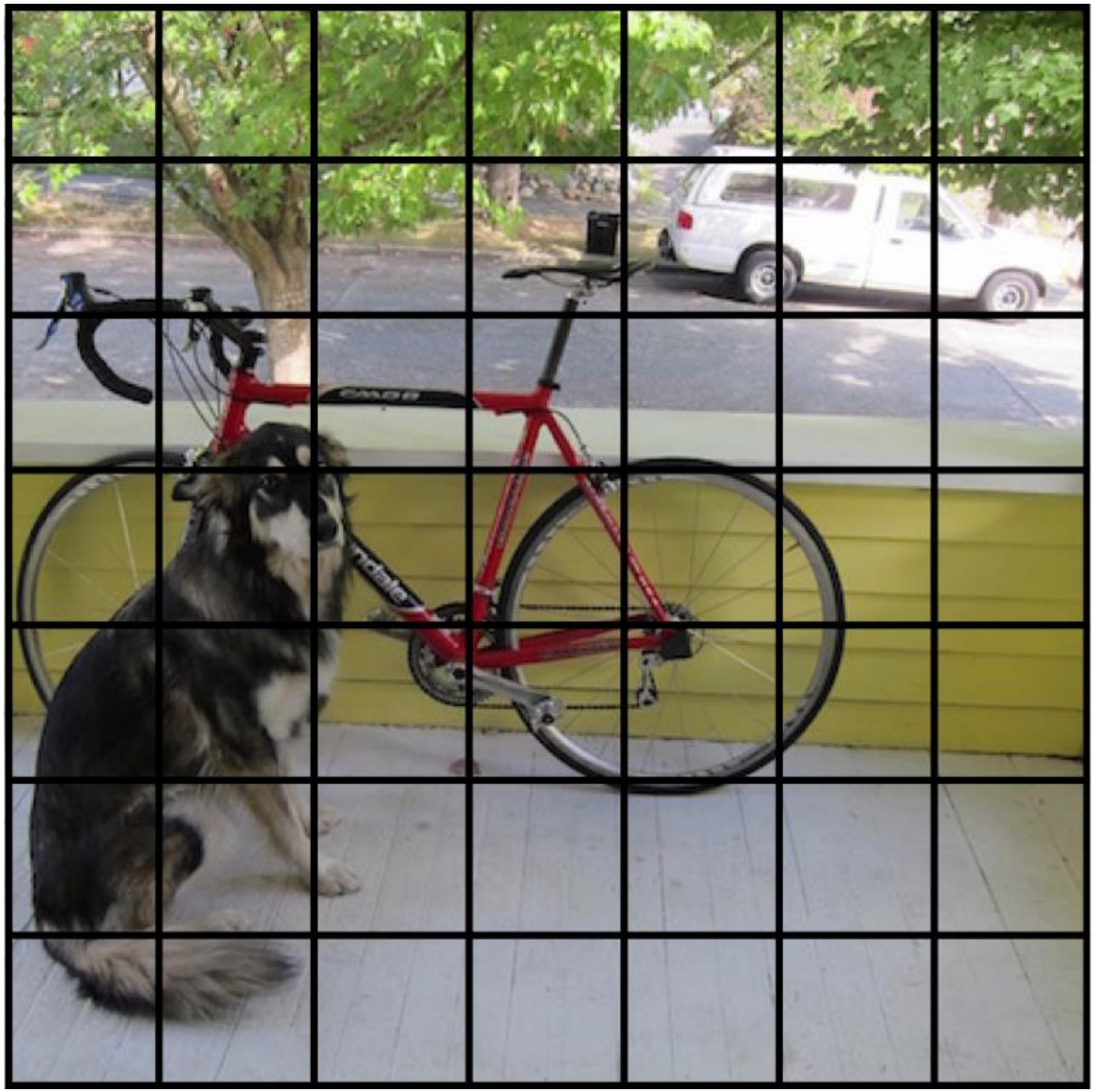


Each cell predicts boxes and confidences: P(Object)

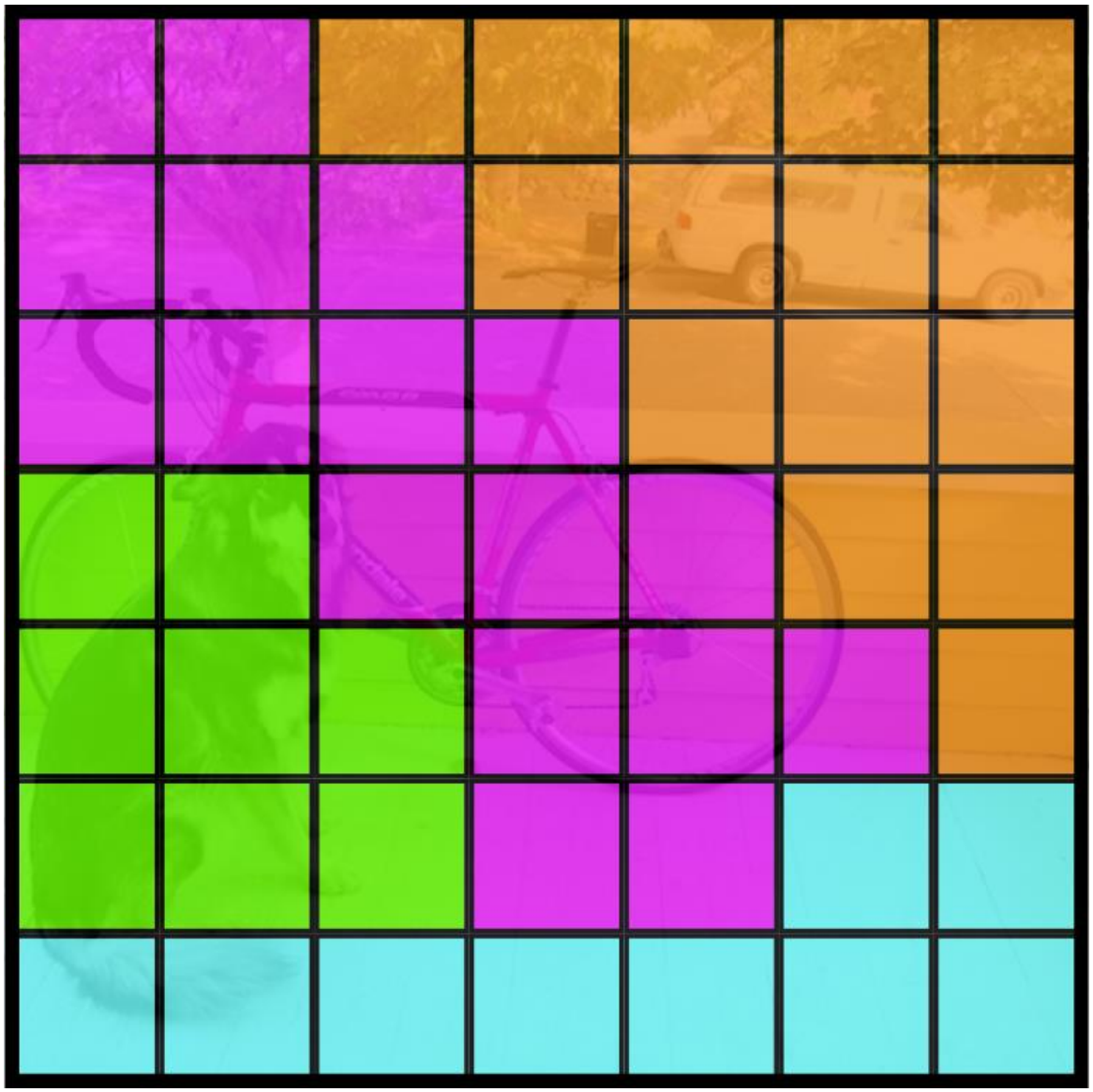


Each cell predicts boxes and confidences: P(Object)

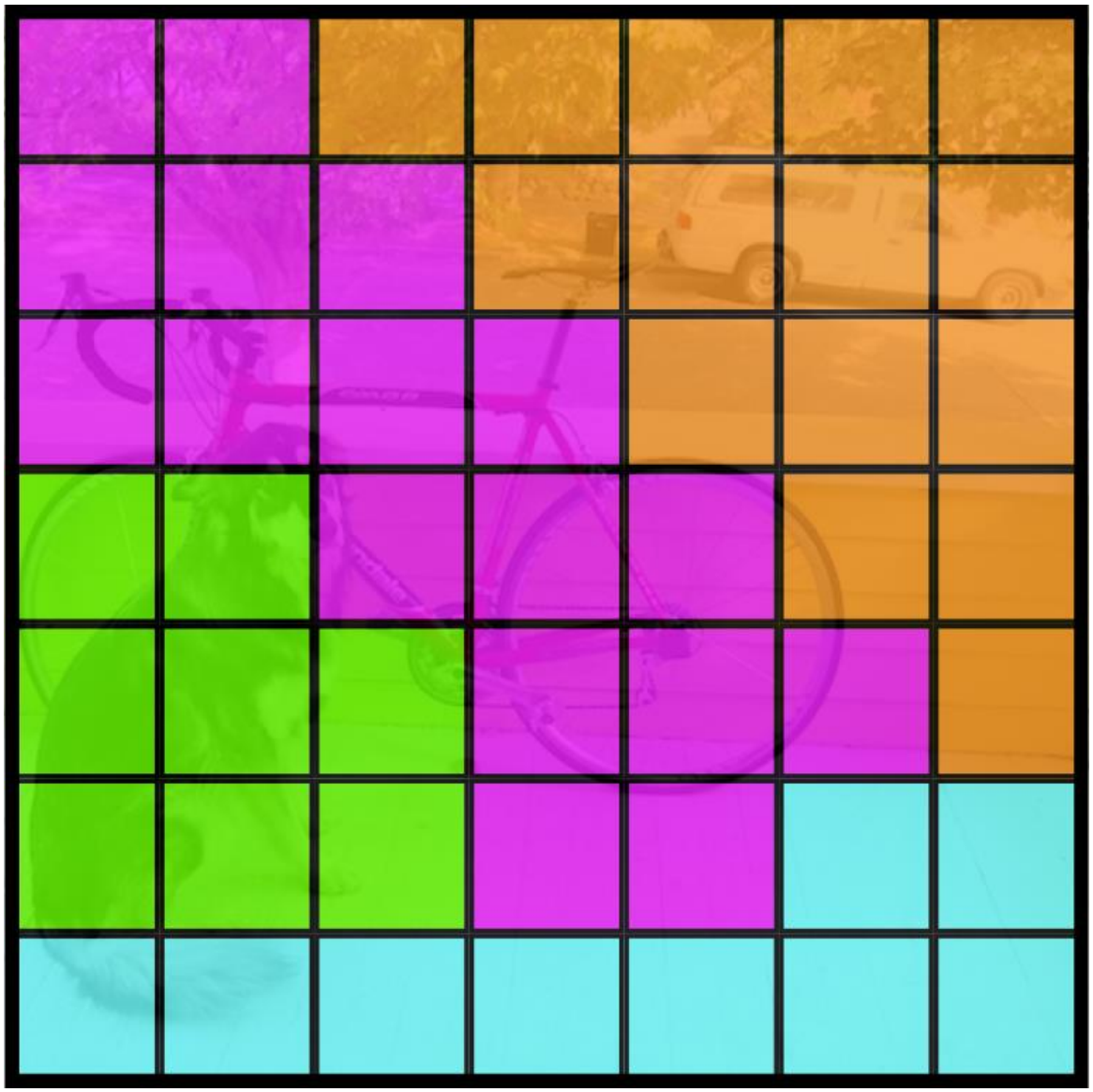
Each cell also predicts a class probability.



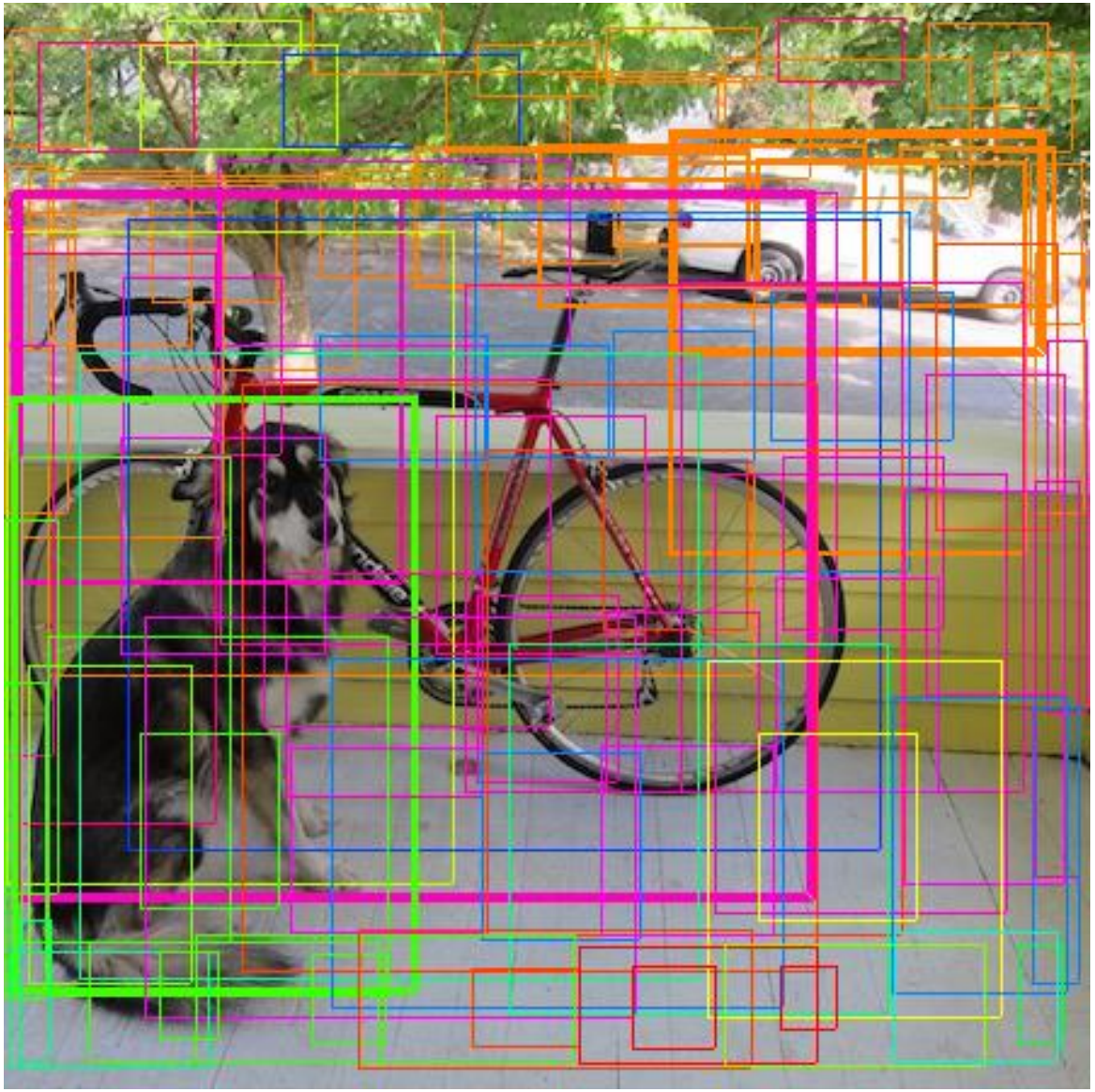
Each cell also predicts a class probability.



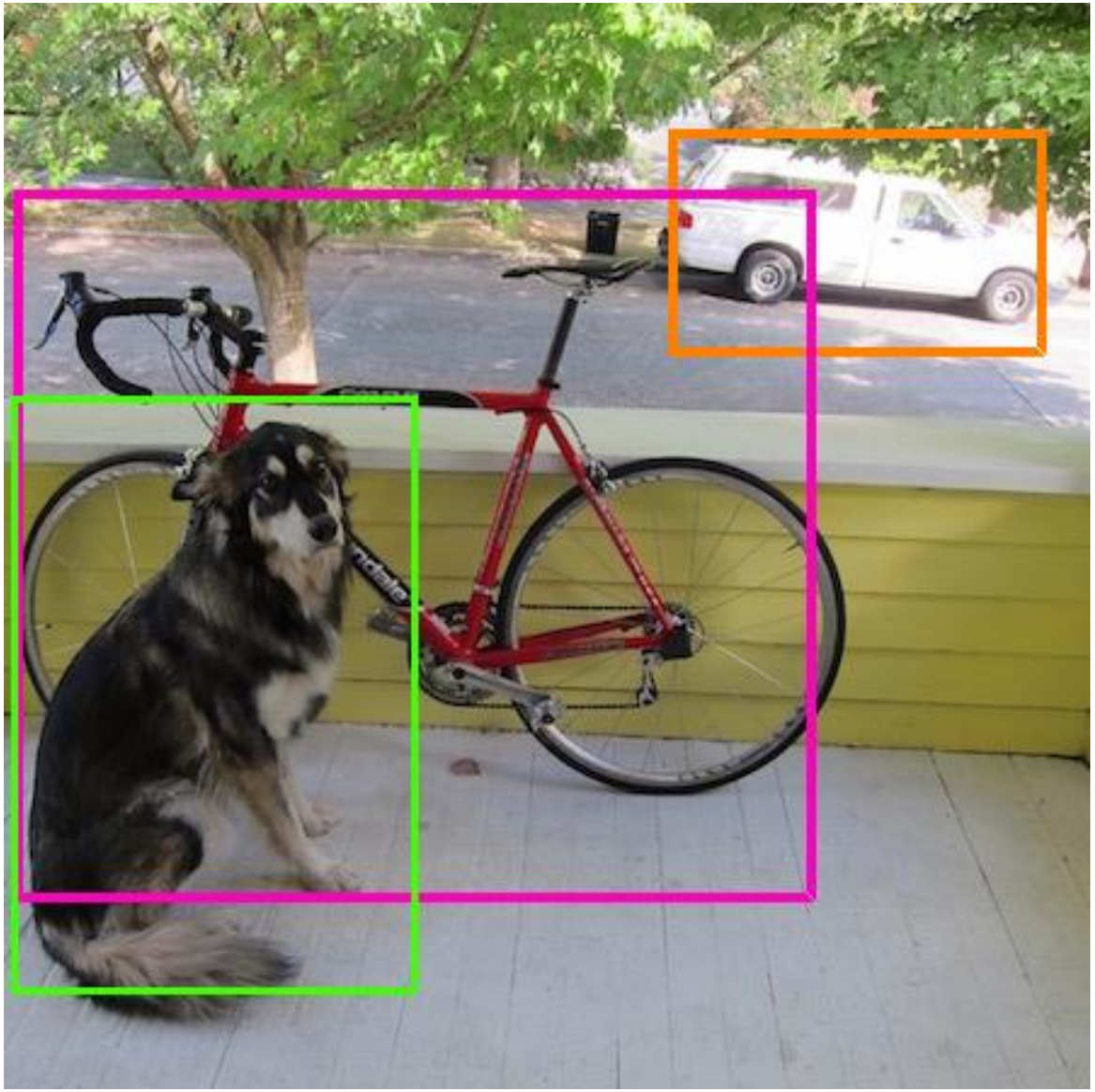
Conditioned on object: P(Car | Object)



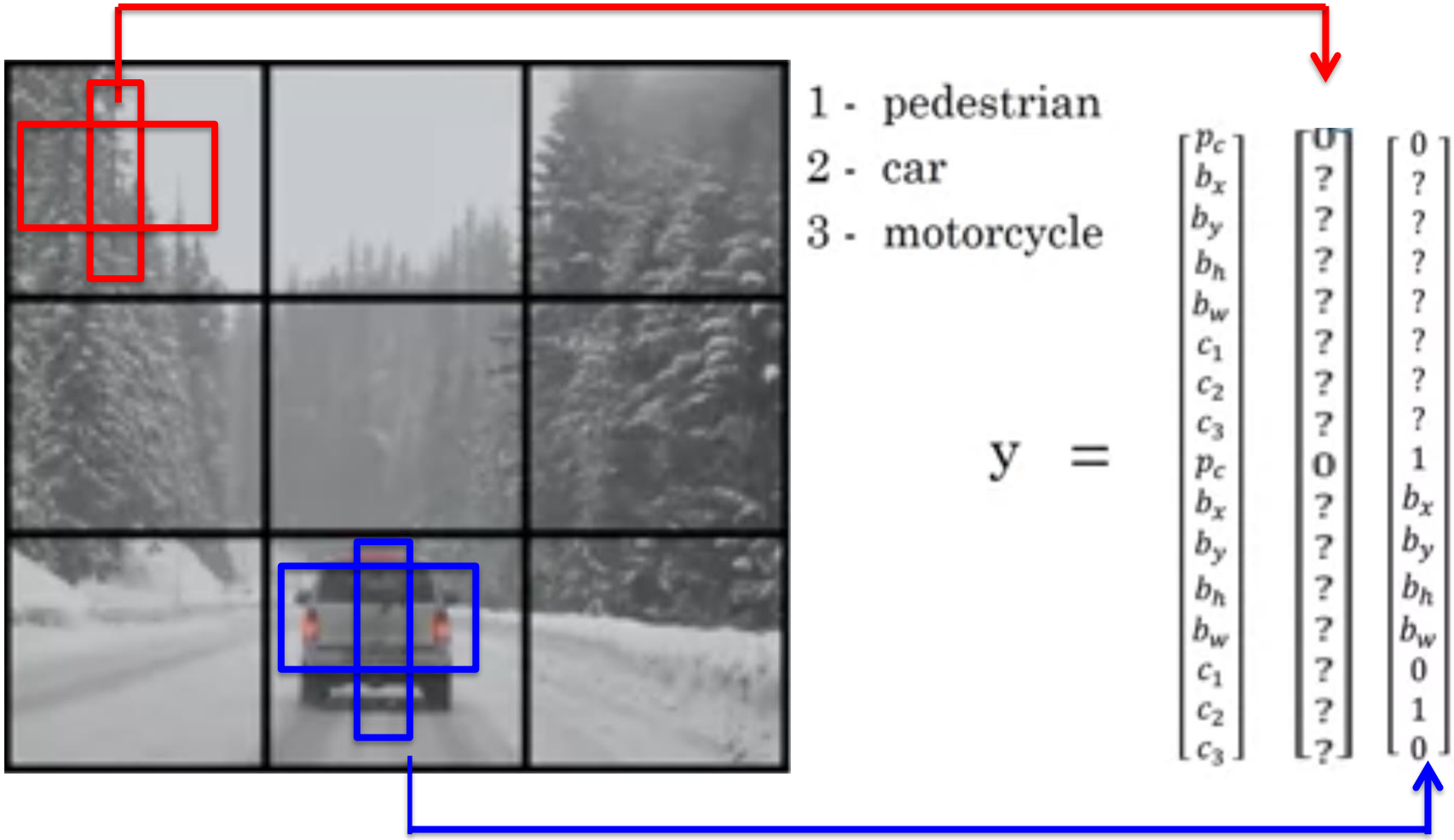
Then we combine the box and class predictions.



Finally we do NMS and threshold detections



Yolo: CNN output for 2 bounding boxes (similar to anchor boxes in RPN)

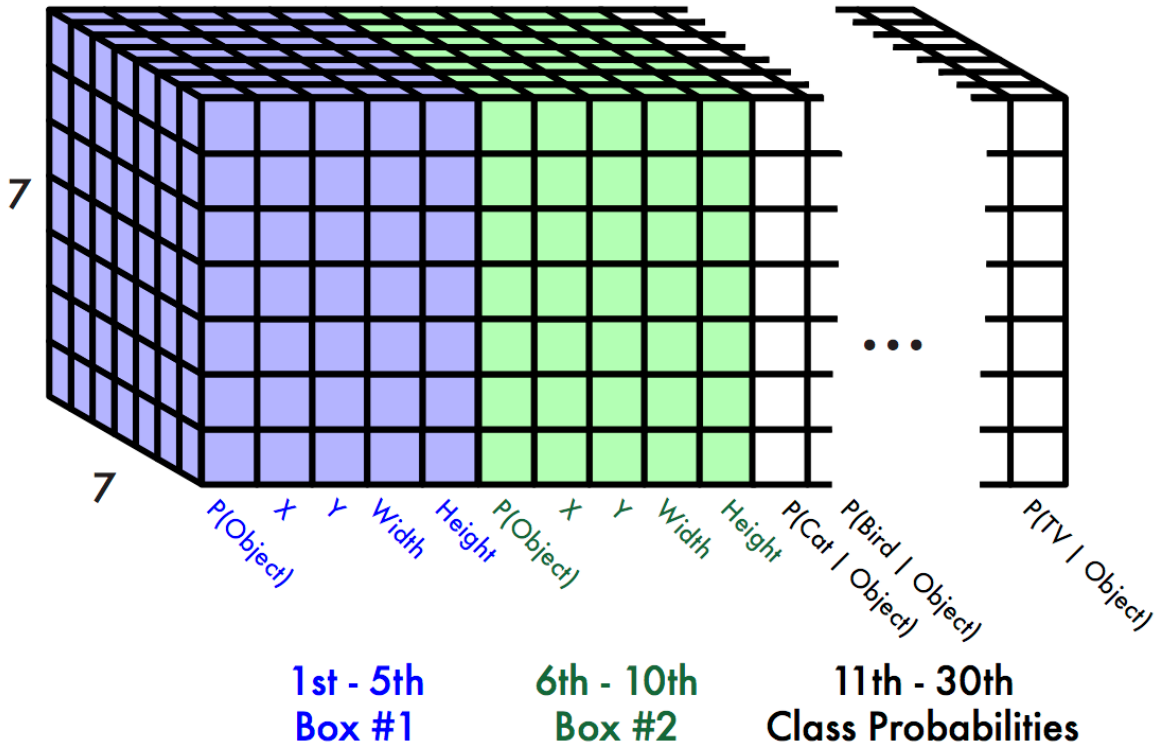


y size: 3 (#grid rows) x 3 (#grid columns) x 2 (#bounding boxes) x 8 (confidence 1, coord 4, class 3)

This parameterization fixes the output size

Each cell predicts:

- For each bounding box:
  - 4 coordinates (x, y, w, h)
  - 1 confidence value
- Some number of class probabilities



For Pascal VOC:

- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

7 x 7 x (2 x 5 + 20) = 7 x 7 x 30 tensor = **1470 outputs**

YOLO

(x, y, w, h) + confidence score (objectness+box)

(x, y, w, h) + confidence score (objectness+box)

car: 0.05
person: 0.10
...
truck : 0.02

...

cell(1, 1)

...

...

cell(S, S)

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

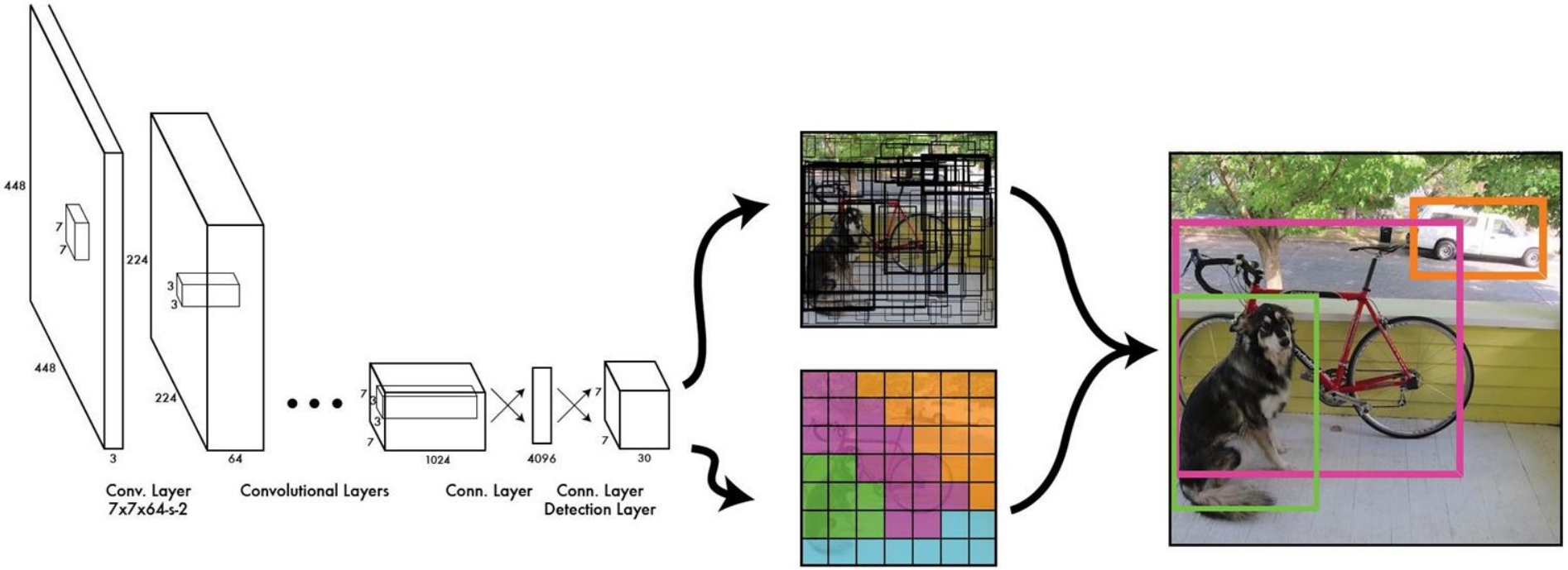
$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

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

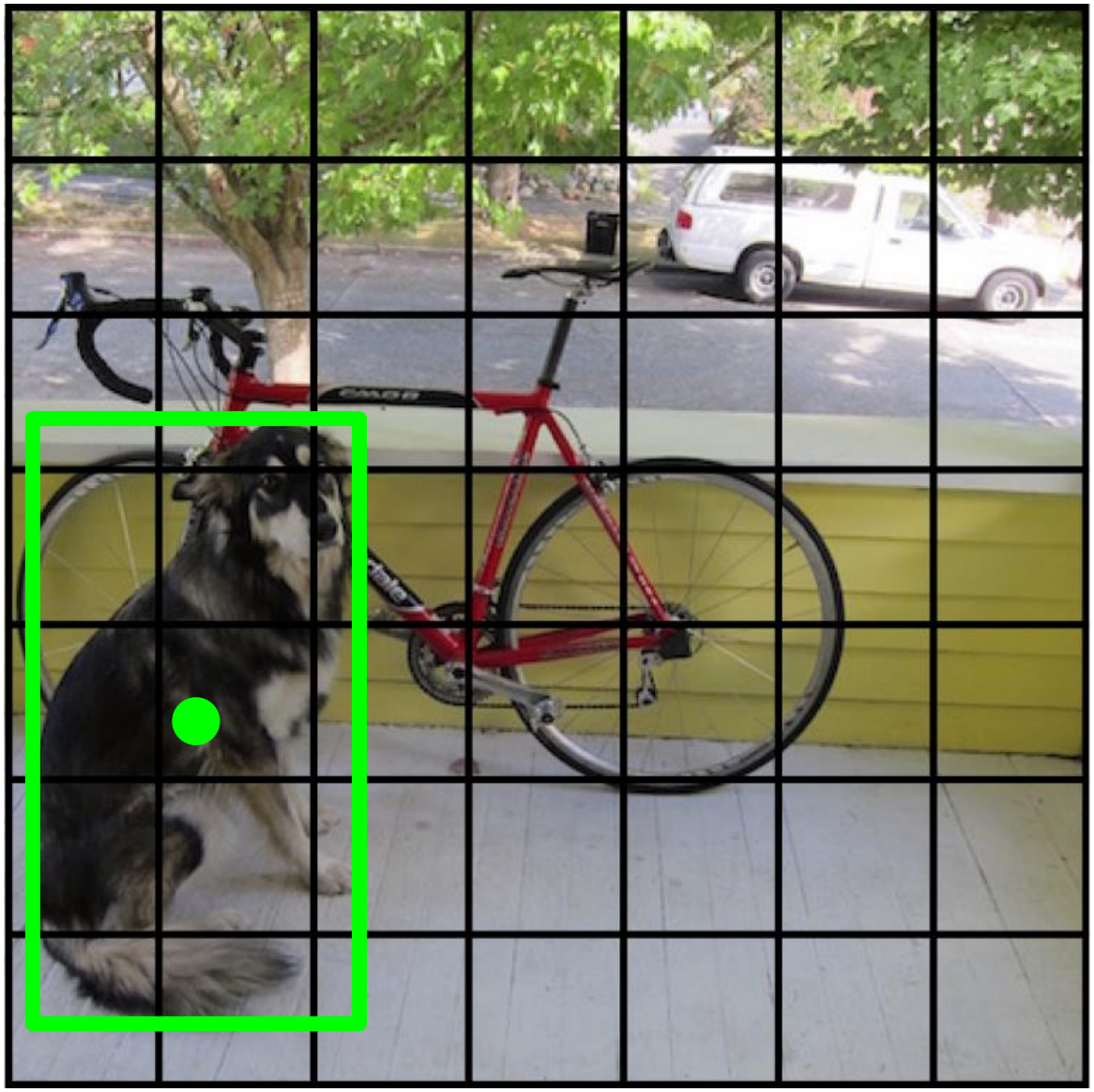
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

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

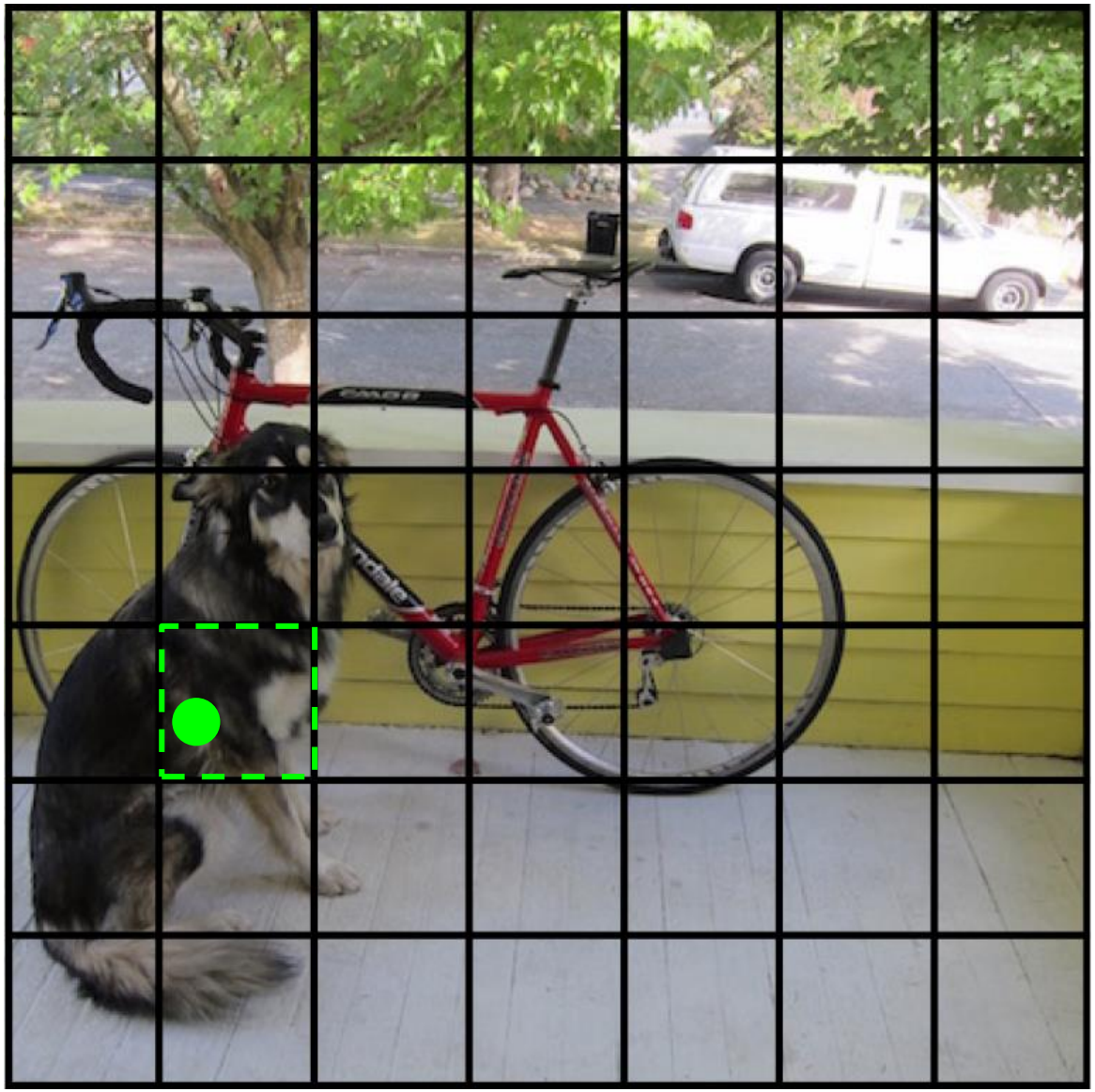
Thus we can train one neural network to be a whole detection pipeline



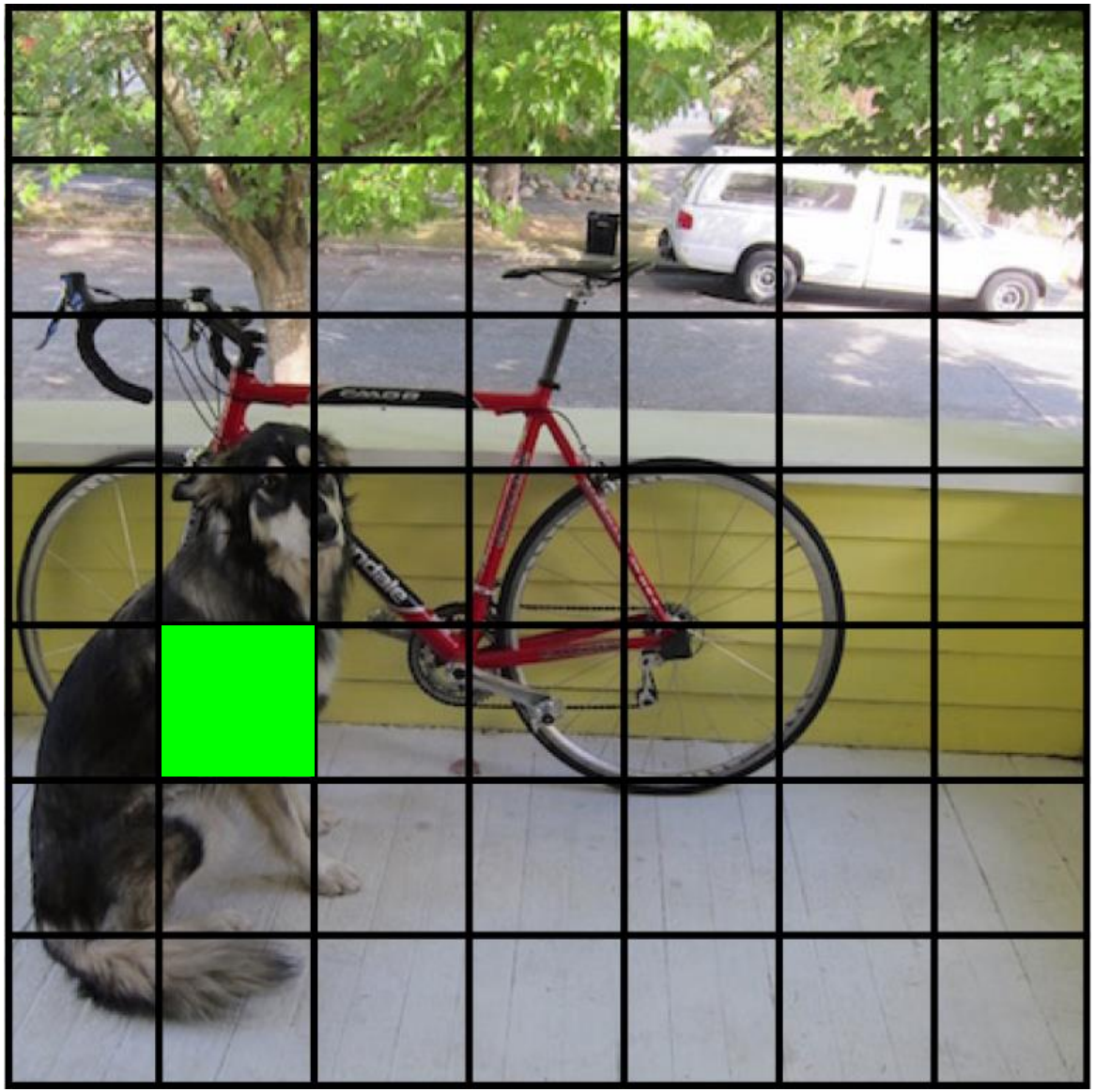
During training, match example to the right cell



During training, match example to the right cell

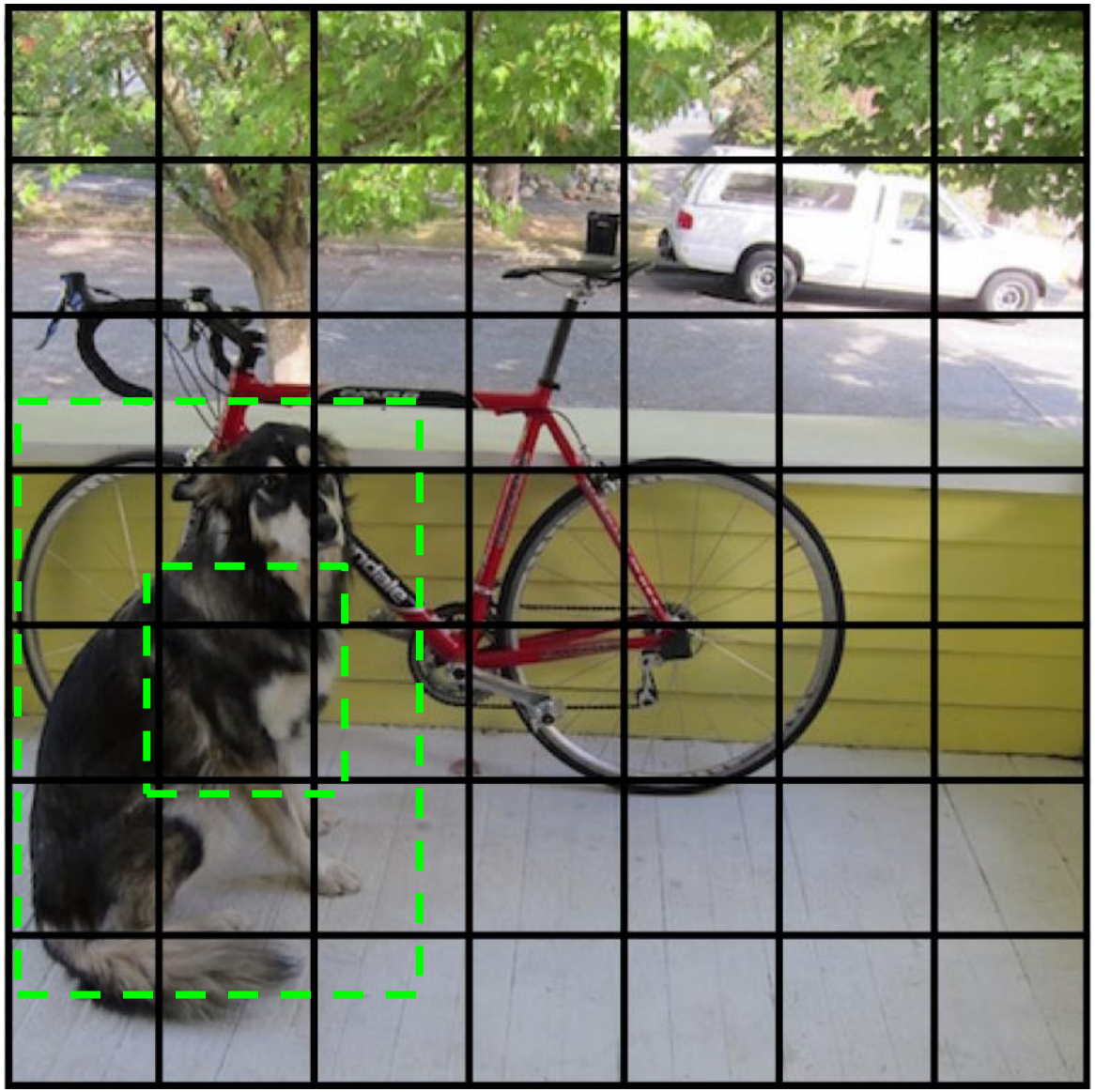


Adjust that cell’ s class prediction

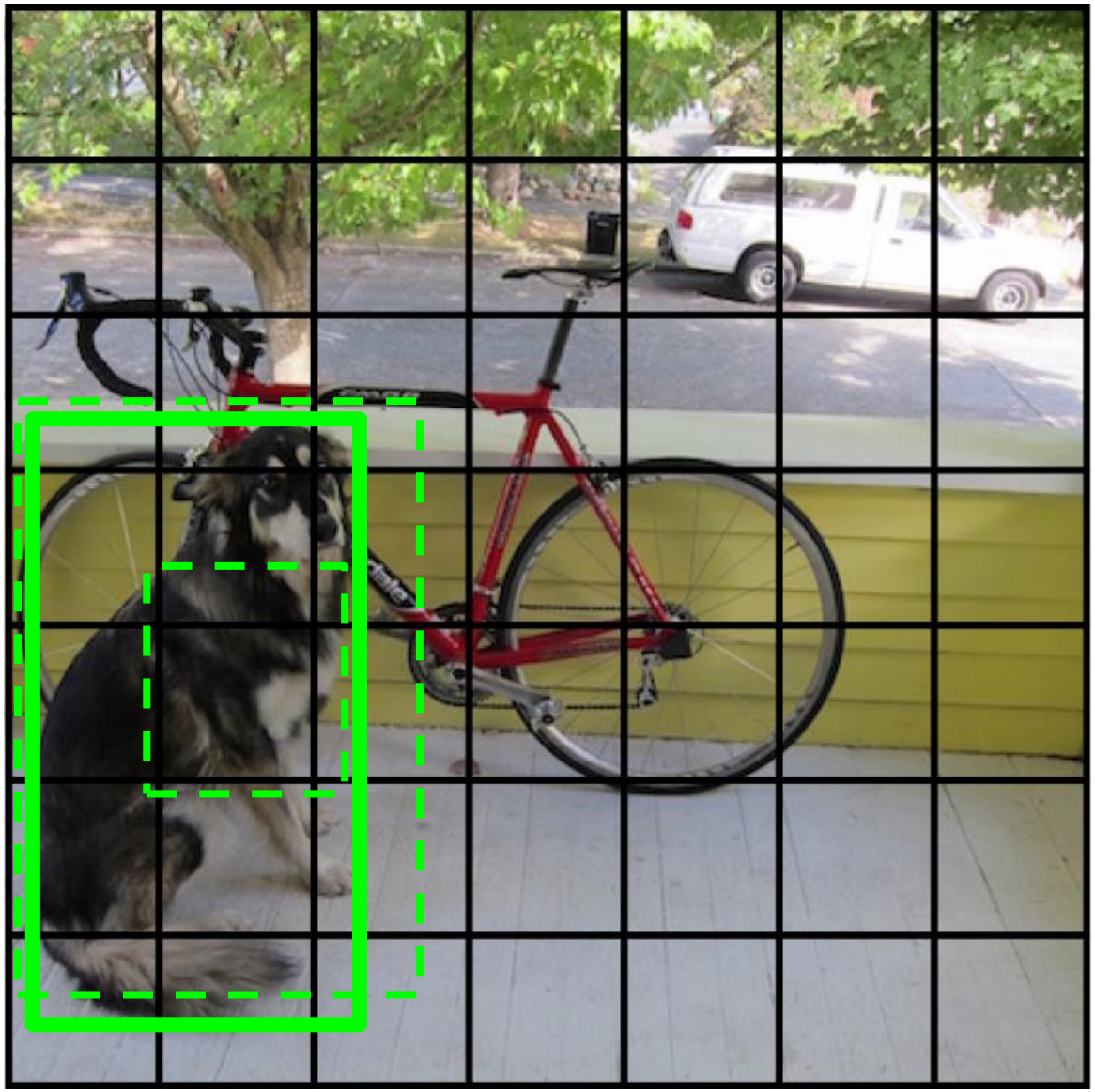


Dog = 1
Cat = 0
Bike = 0
...

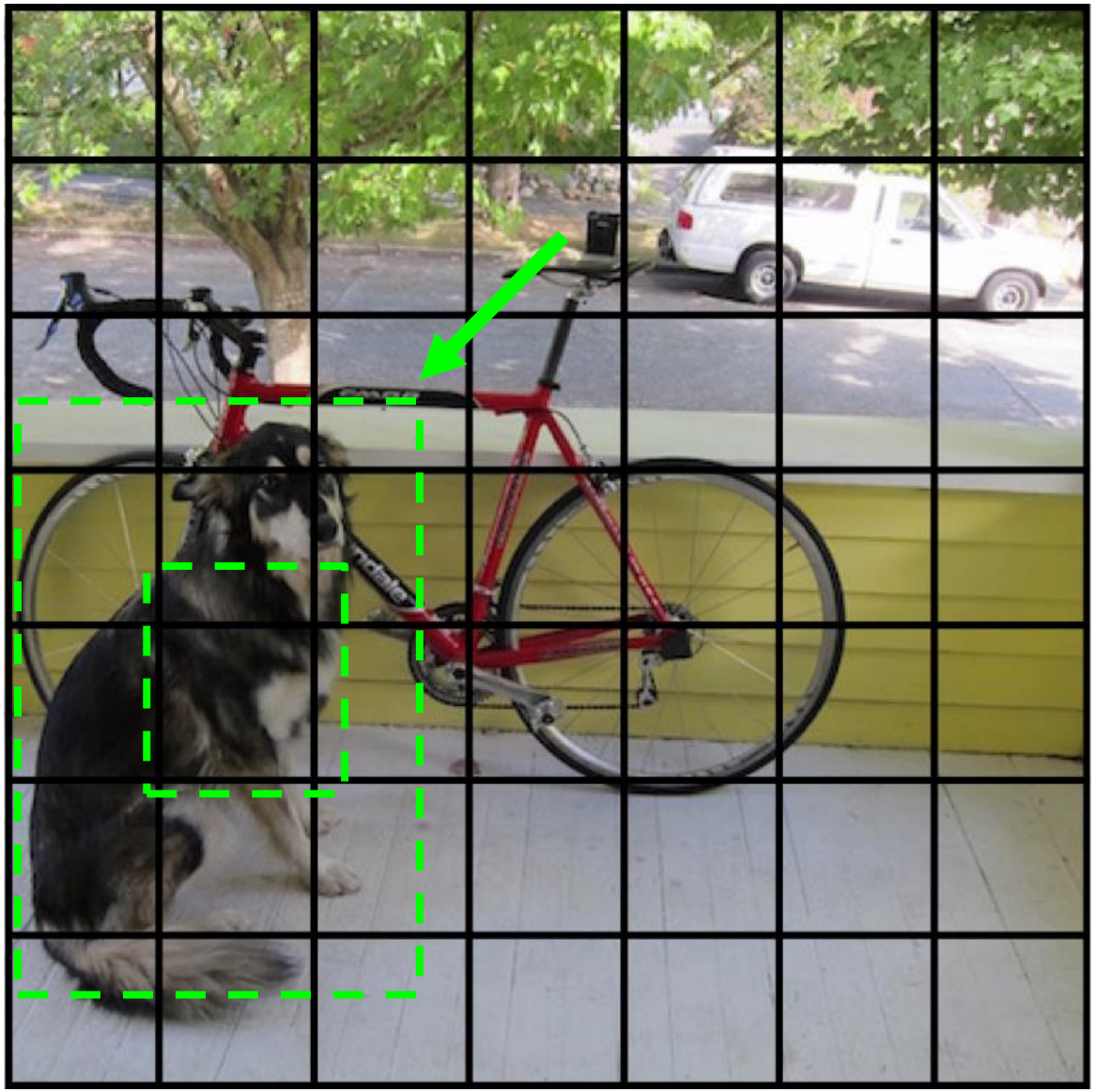
Look at that cell’ s predicted boxes



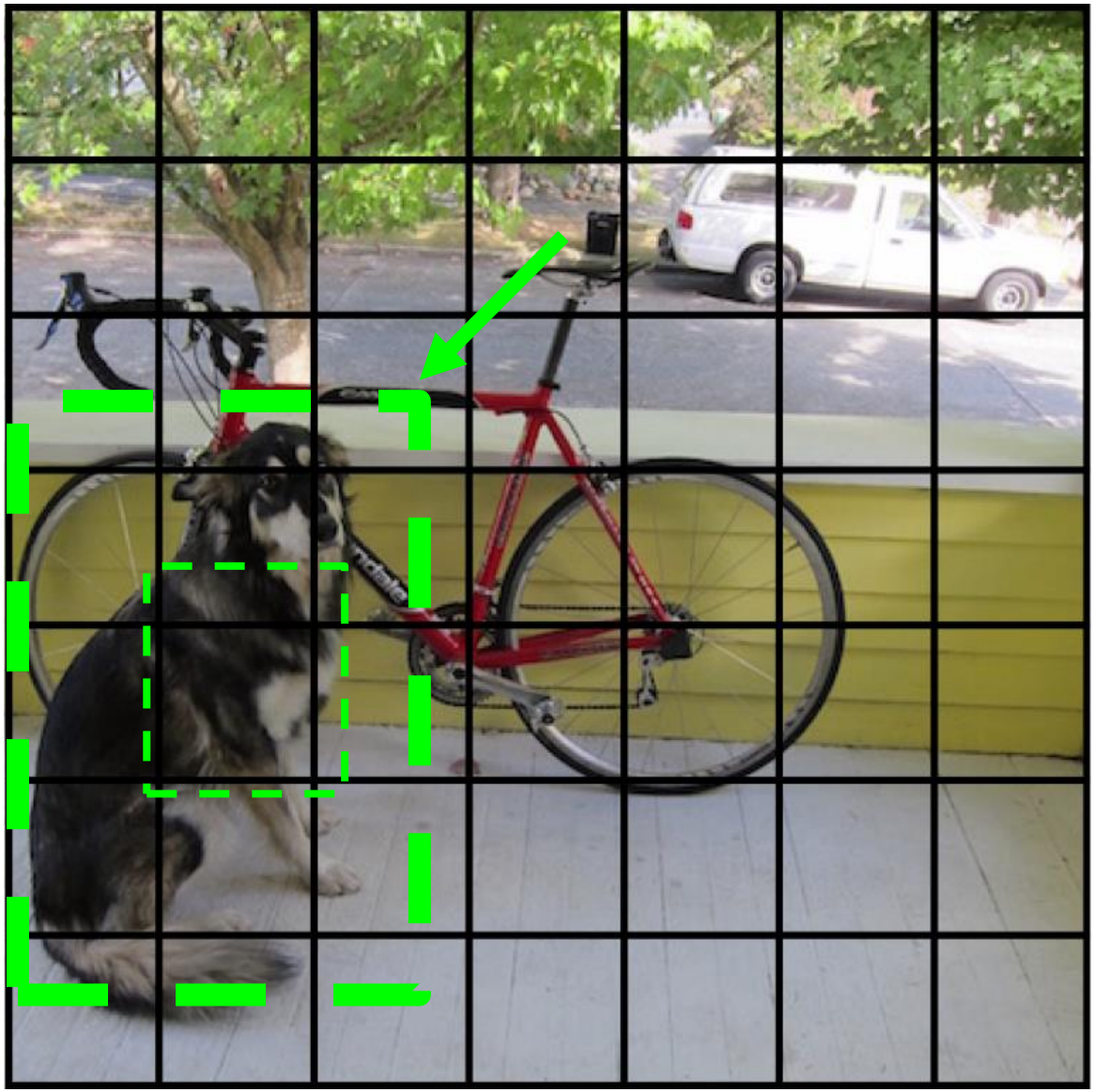
Find the best one, adjust it, increase the confidence



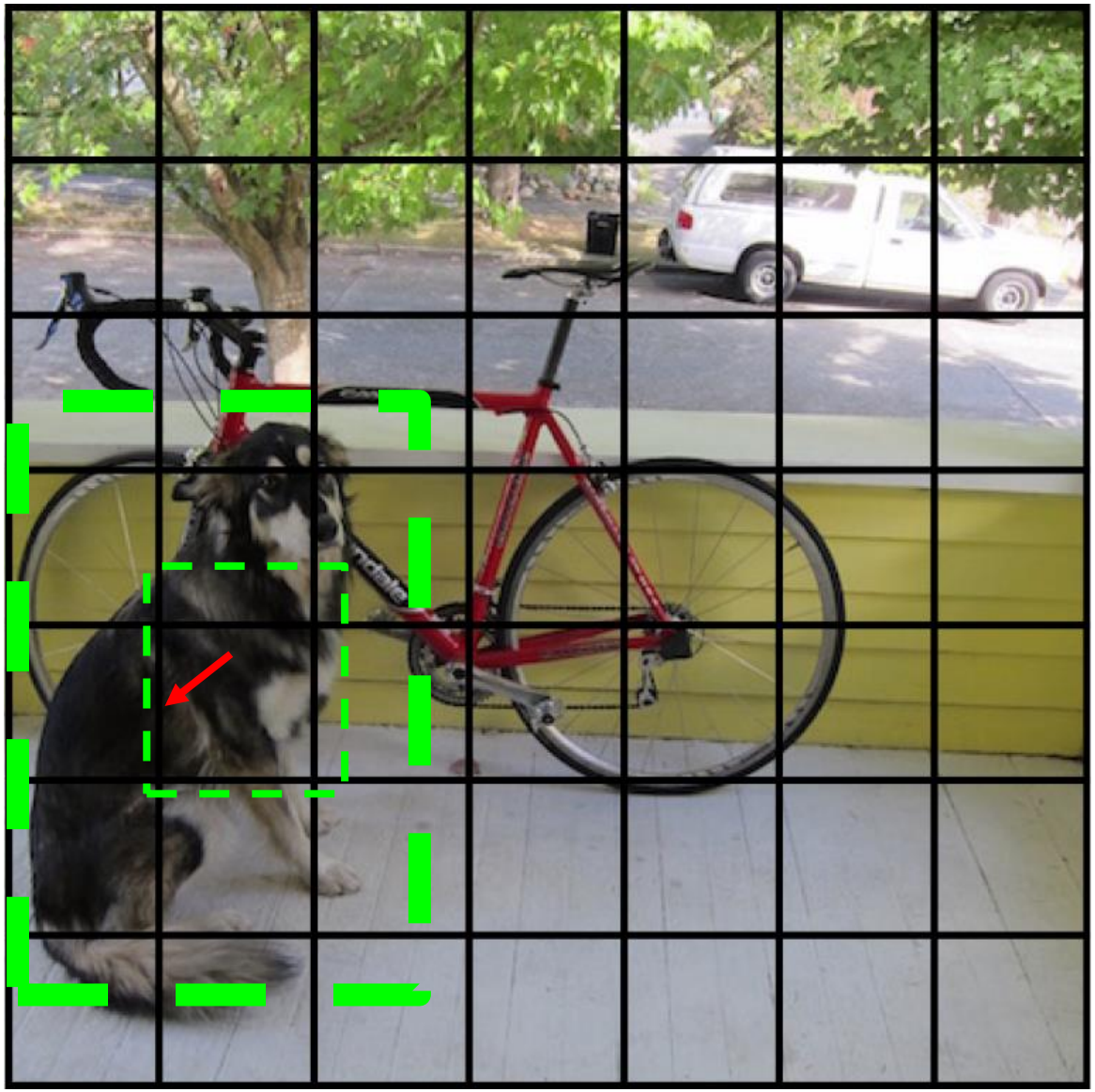
Find the best one, adjust it, increase the confidence



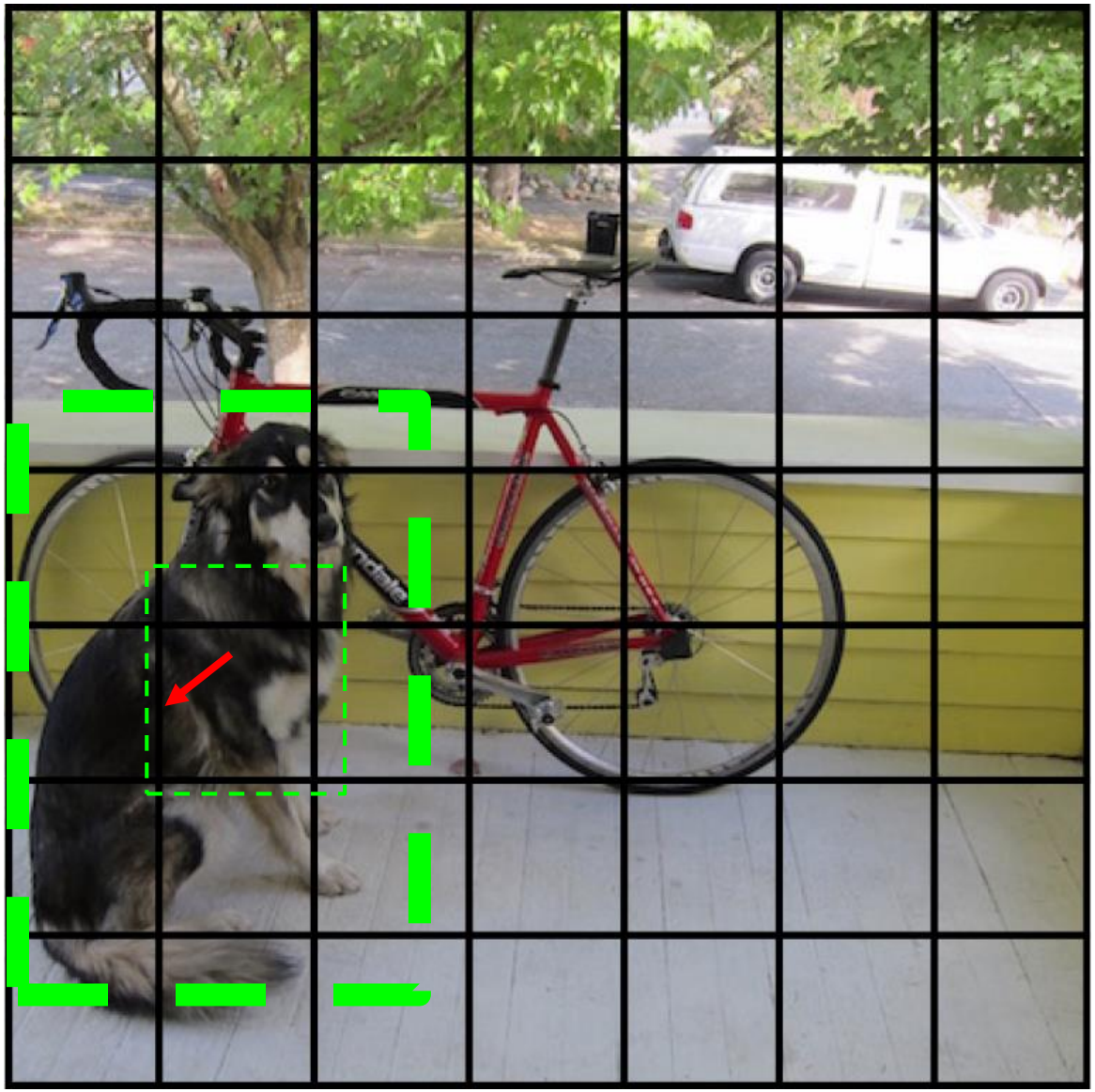
Find the best one, adjust it, increase the confidence



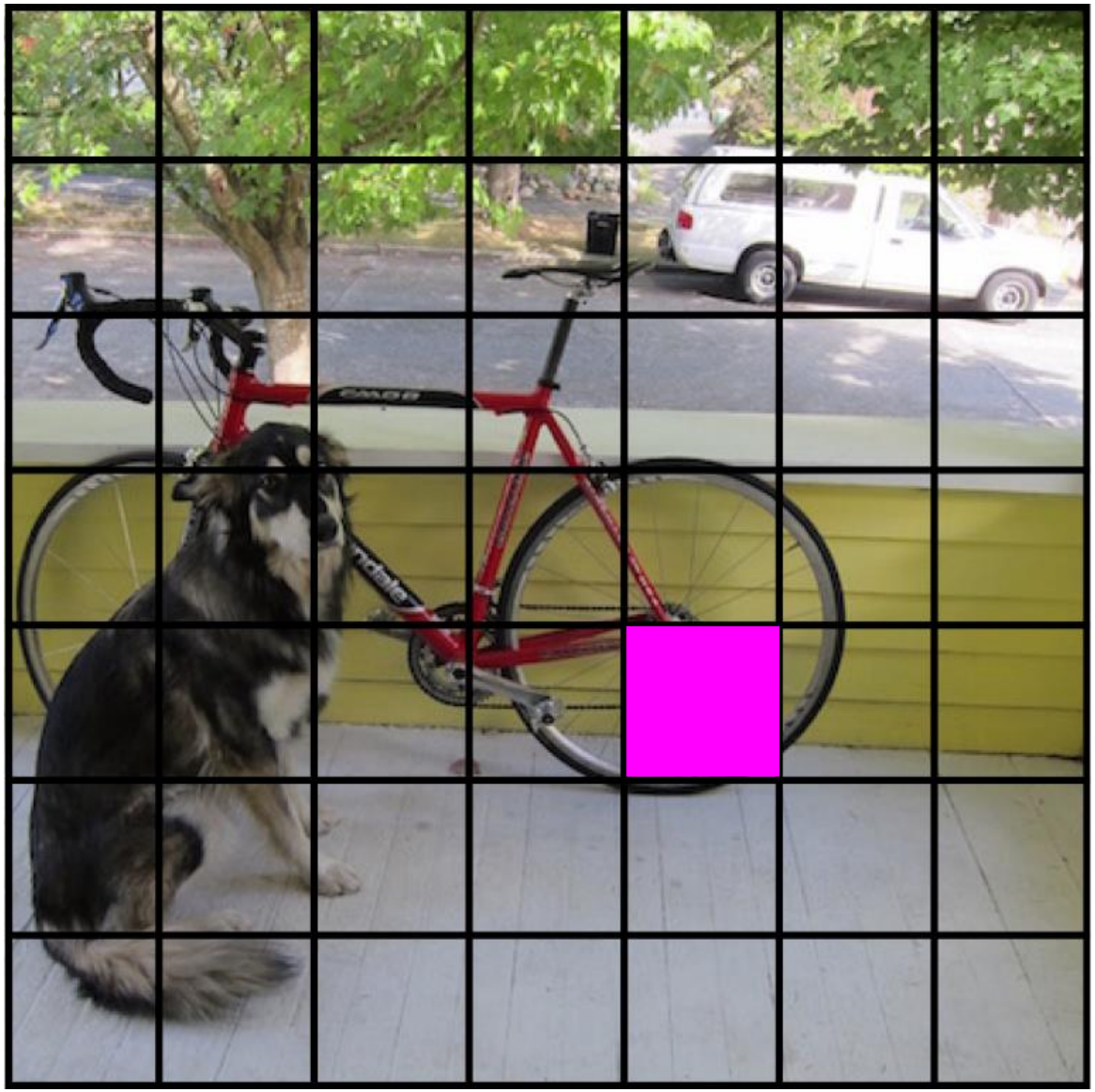
Decrease the confidence of other boxes



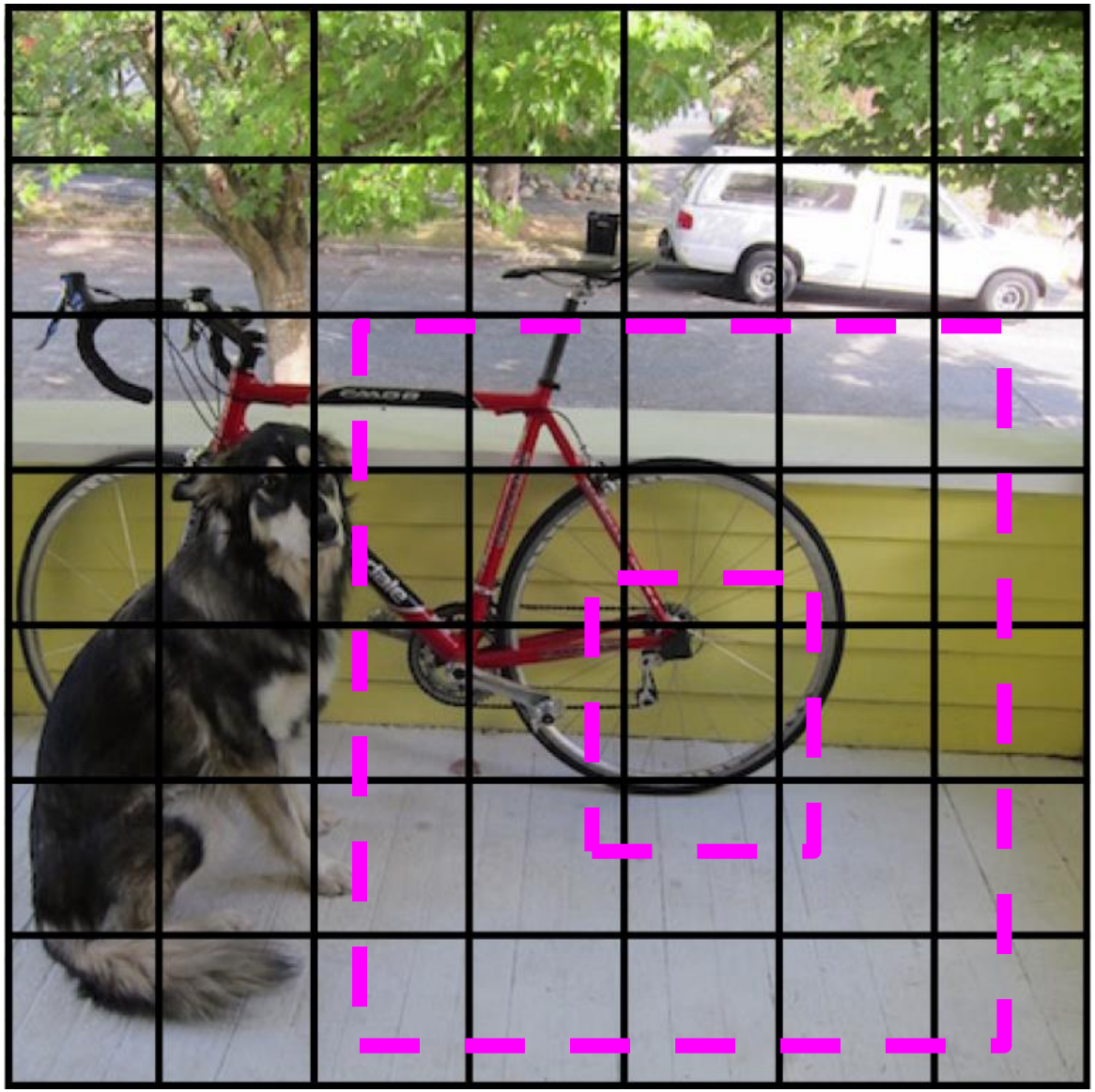
Decrease the confidence of other boxes



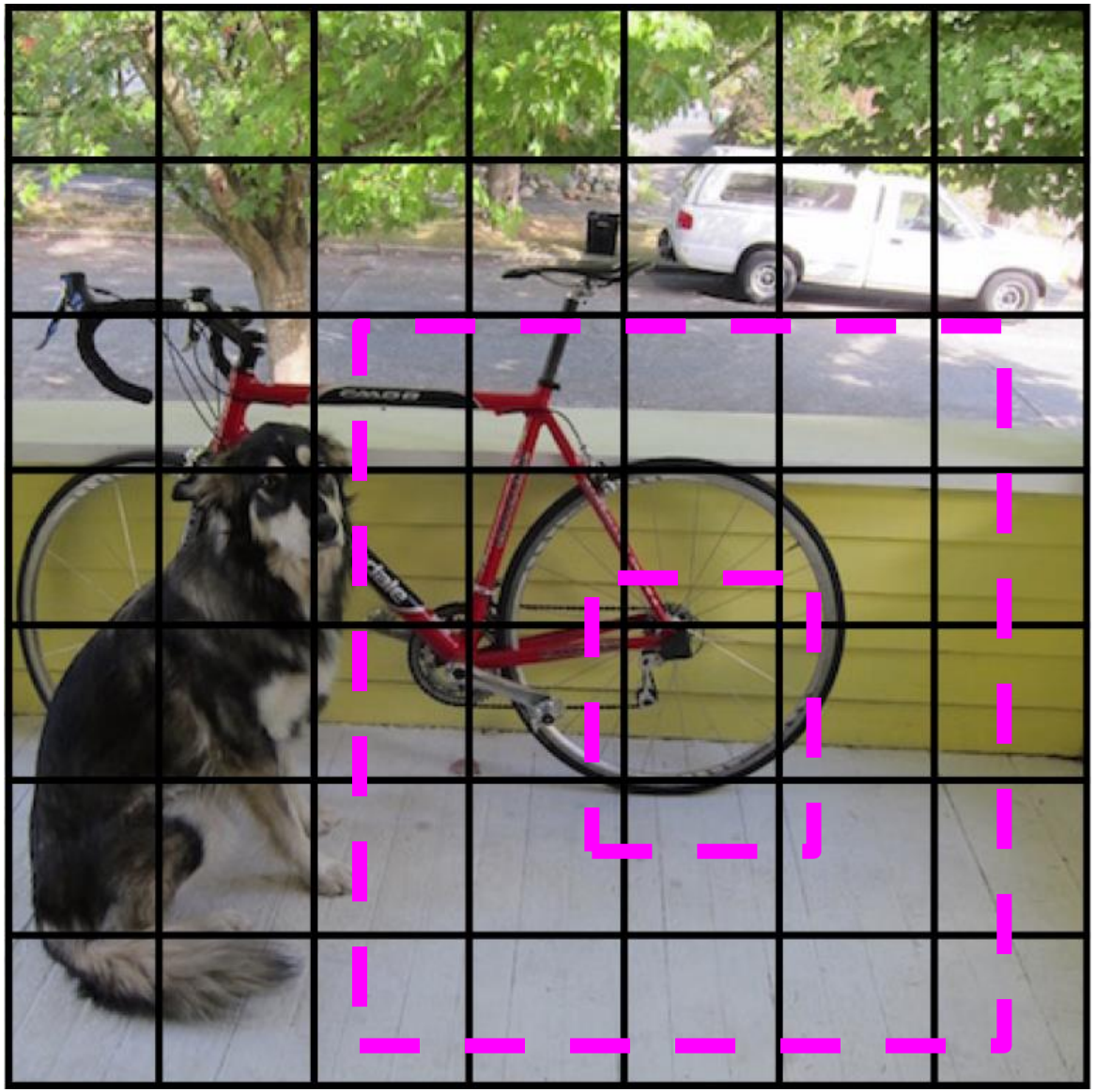
Some cells don’ t have any ground truth detections!



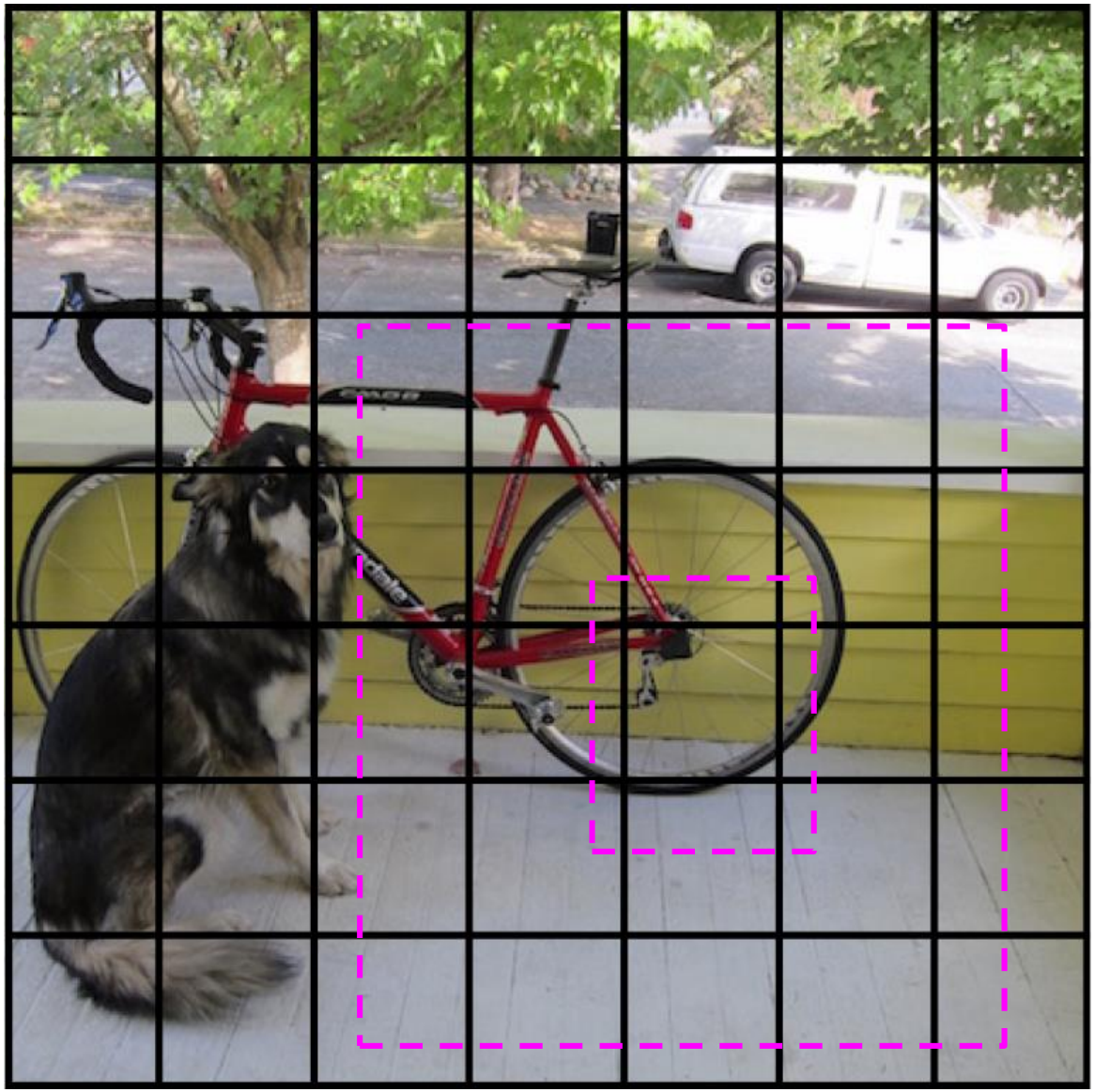
Some cells don’ t have any ground truth detections!



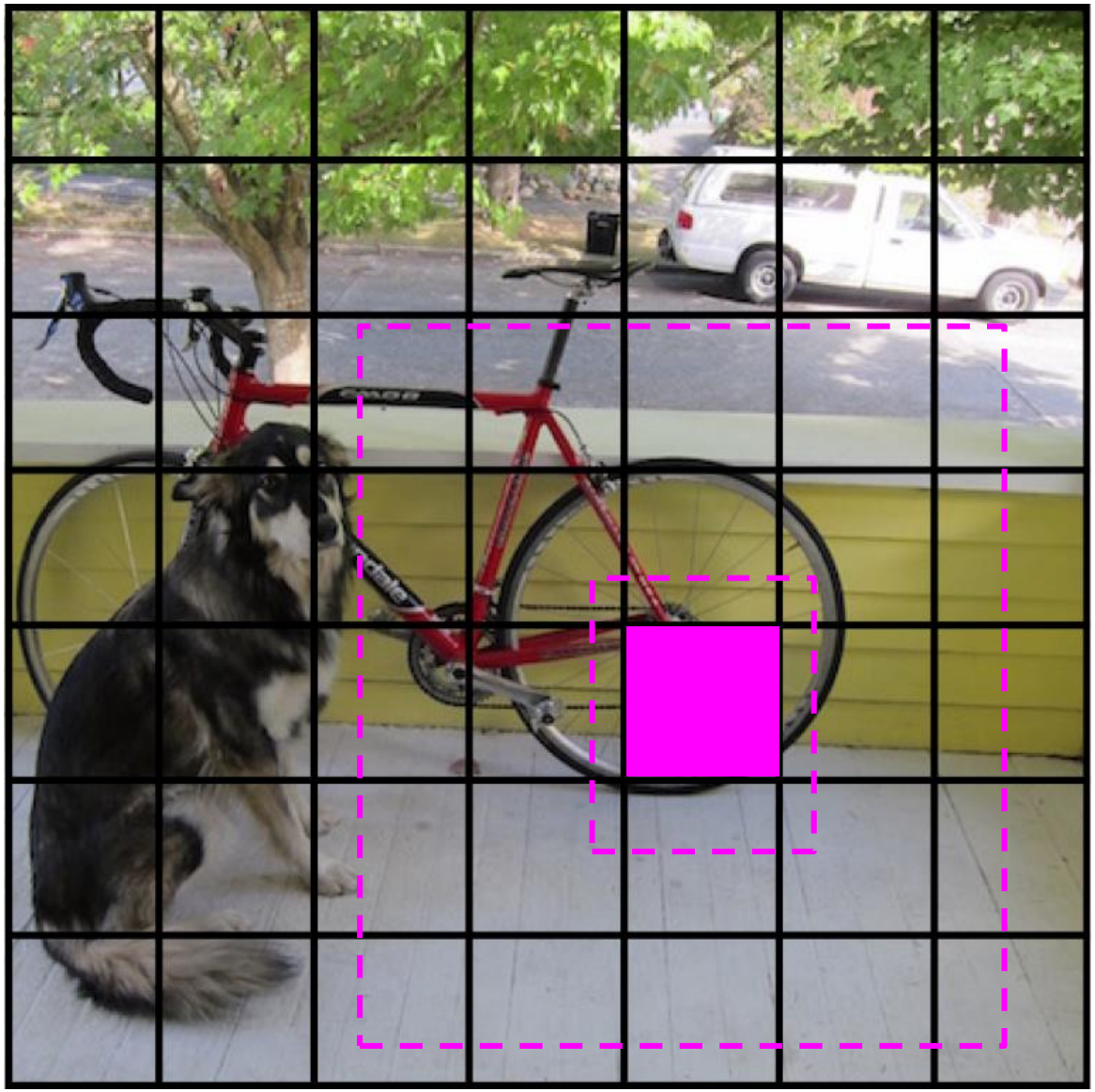
Decrease the confidence of these boxes



Decrease the confidence of these boxes

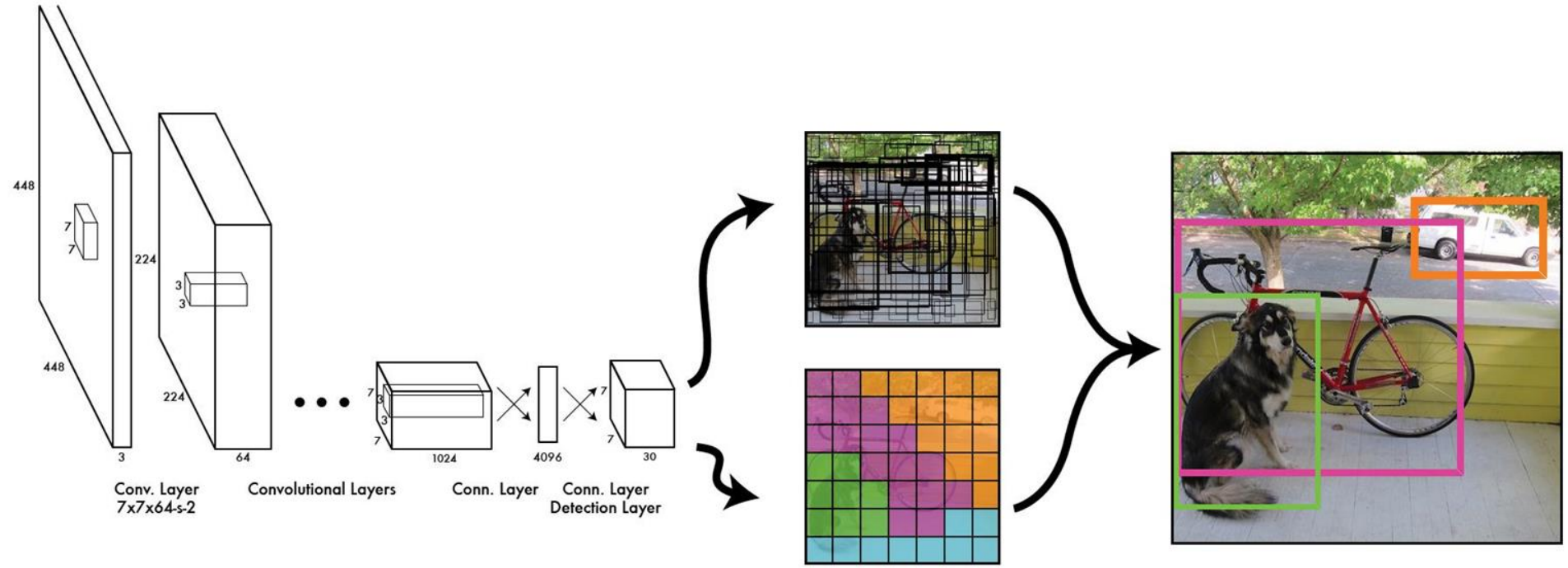


Do not adjust the class probabilities or coordinates

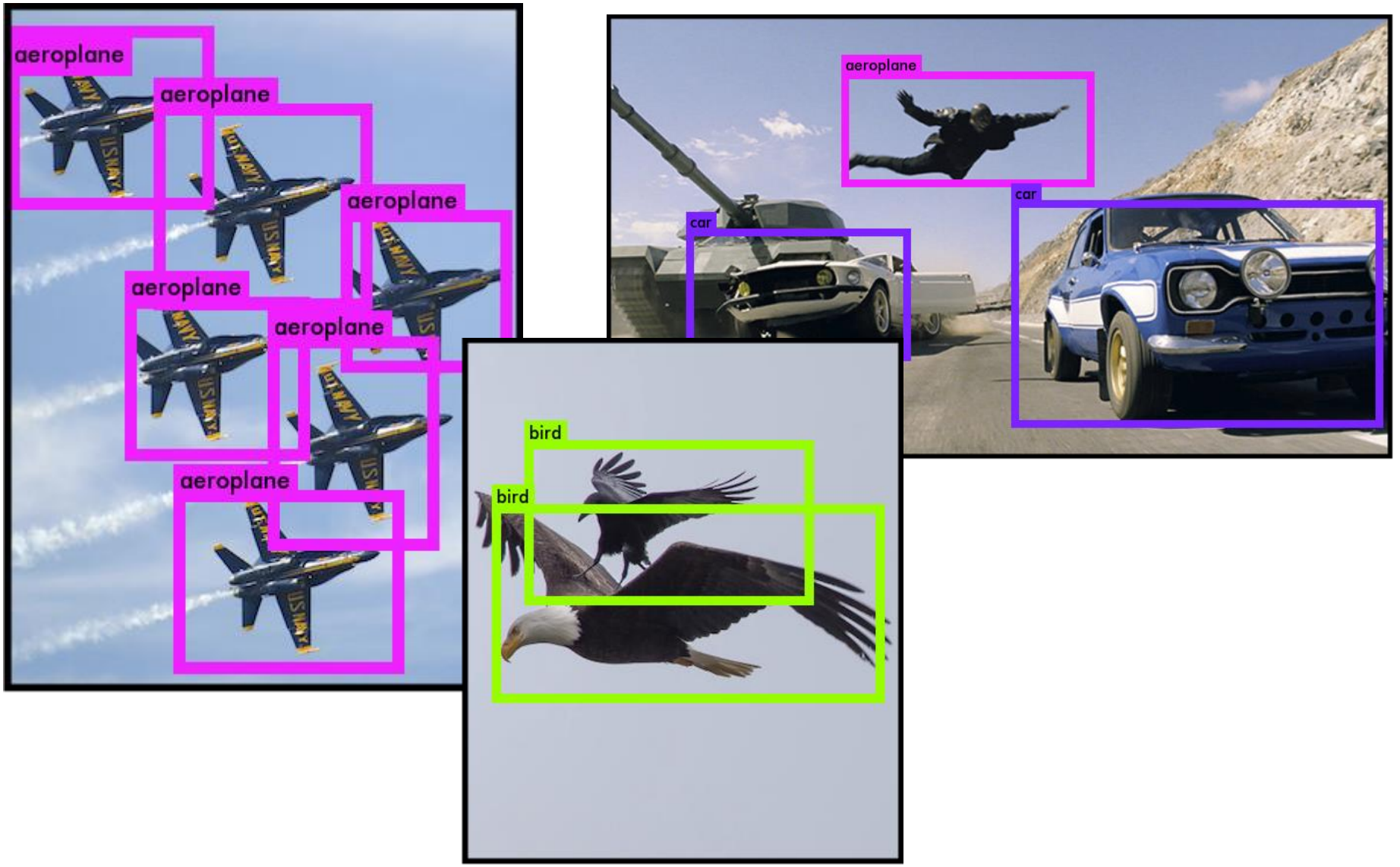


We train with standard tricks:

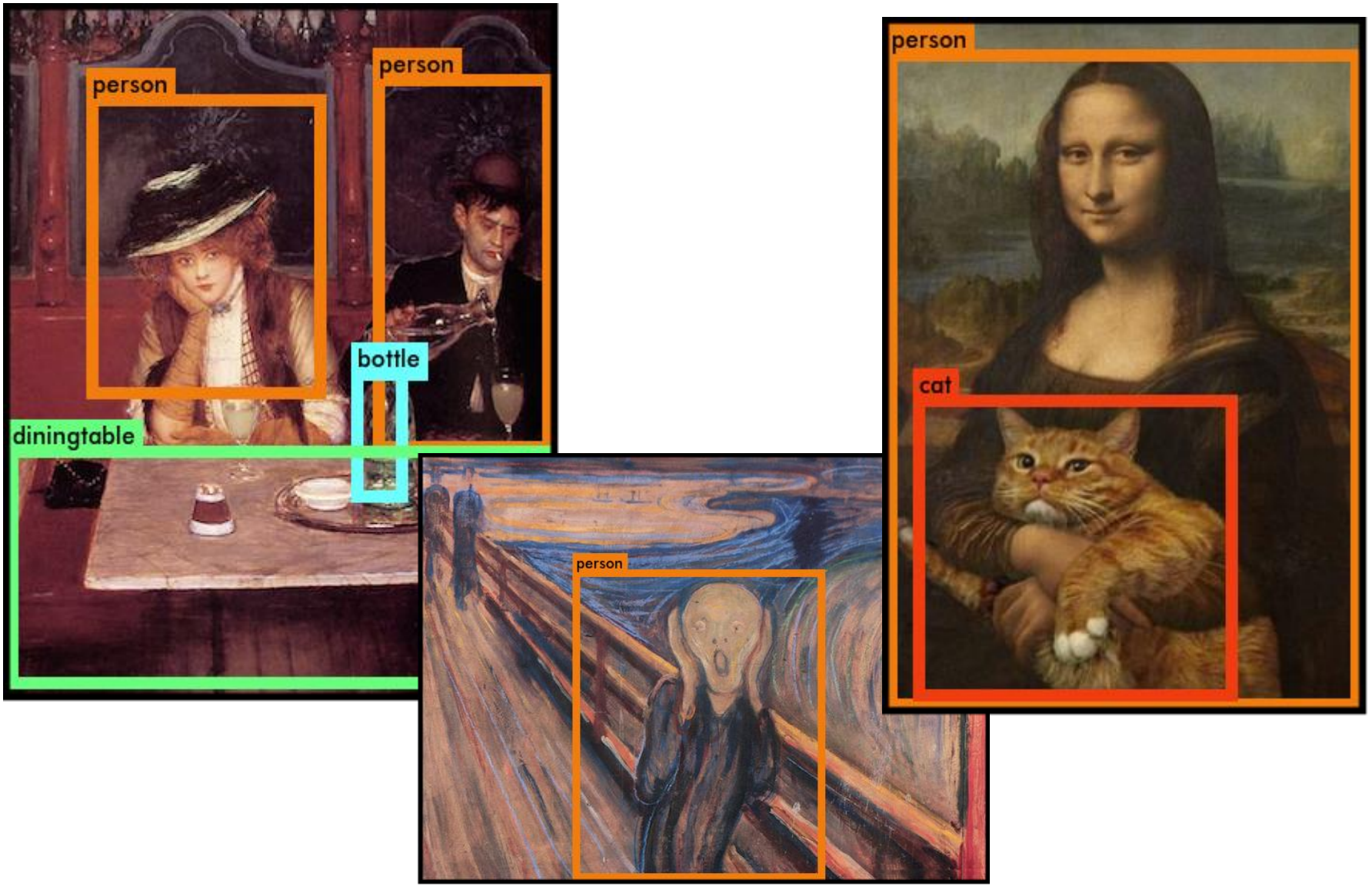
- Pretraining on Imagenet
- SGD with decreasing learning rate
- Extensive data augmentation
- For details, see the paper



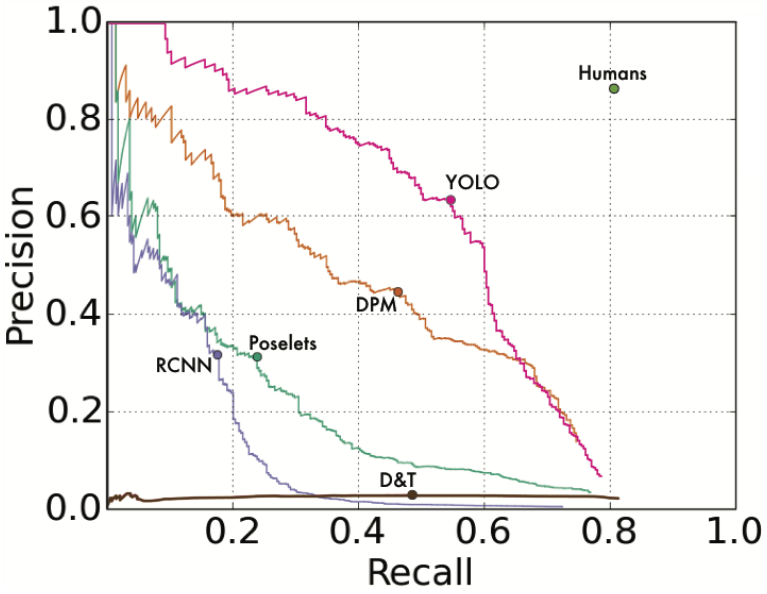
YOLO works across a variety of natural images



It also generalizes well to new domains (like art)



YOLO outperforms methods like DPM and R-CNN when generalizing to person detection in artwork



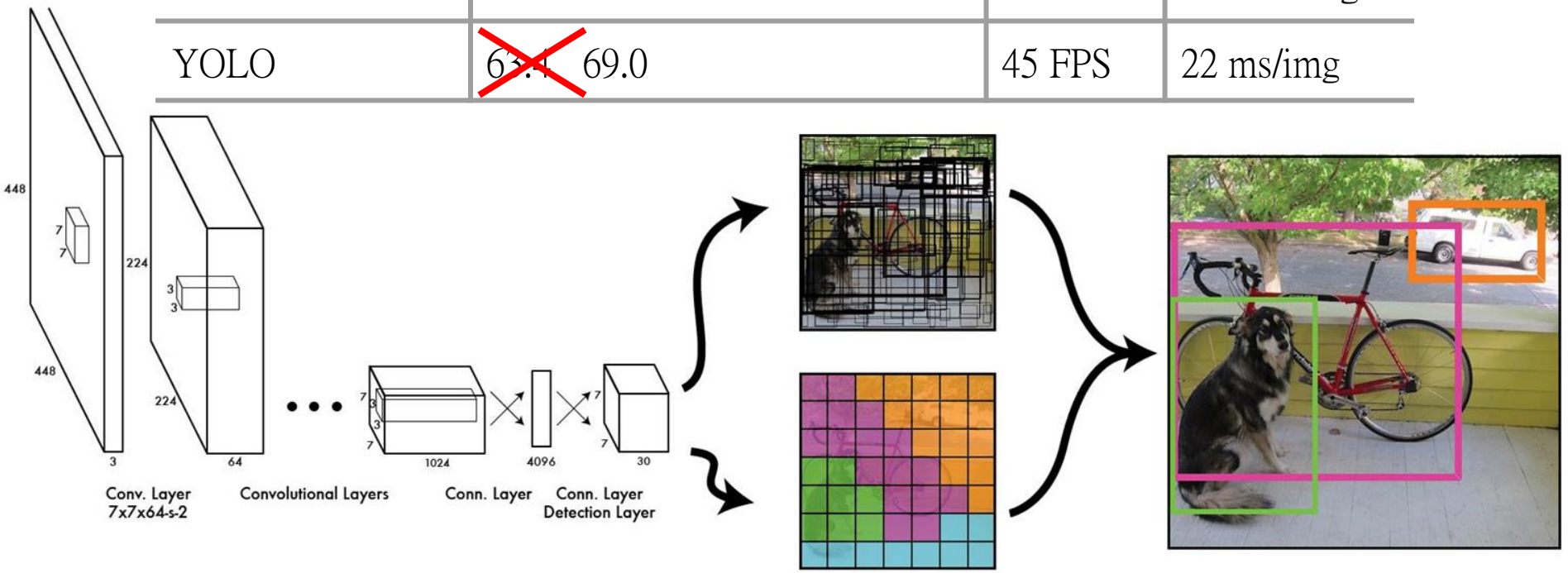
	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32

S. Ginosar, D. Haas, T. Brown, and J. Malik. Detecting people in cubist art. In Computer Vision-ECCV 2014 Workshops, pages 101–116. Springer, 2014.

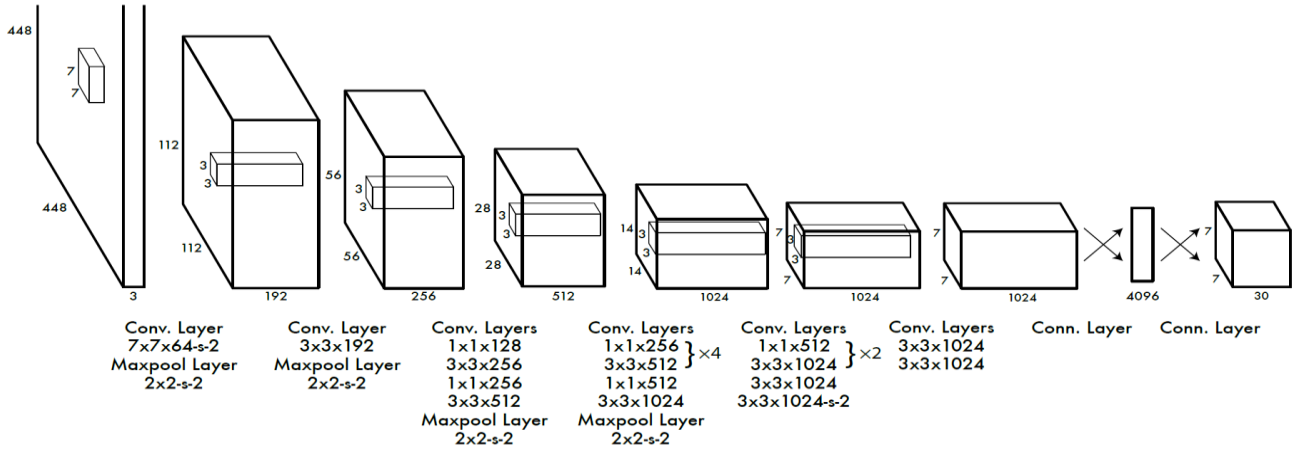
H. Cai, Q. Wu, T. Corradi, and P. Hall. The cross-depiction problem: Computer vision algorithms for recognising objects in artwork and in photographs.

Code available! pjreddie.com/yolo

	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	~~63.4~~ 69.0	45 FPS	22 ms/img



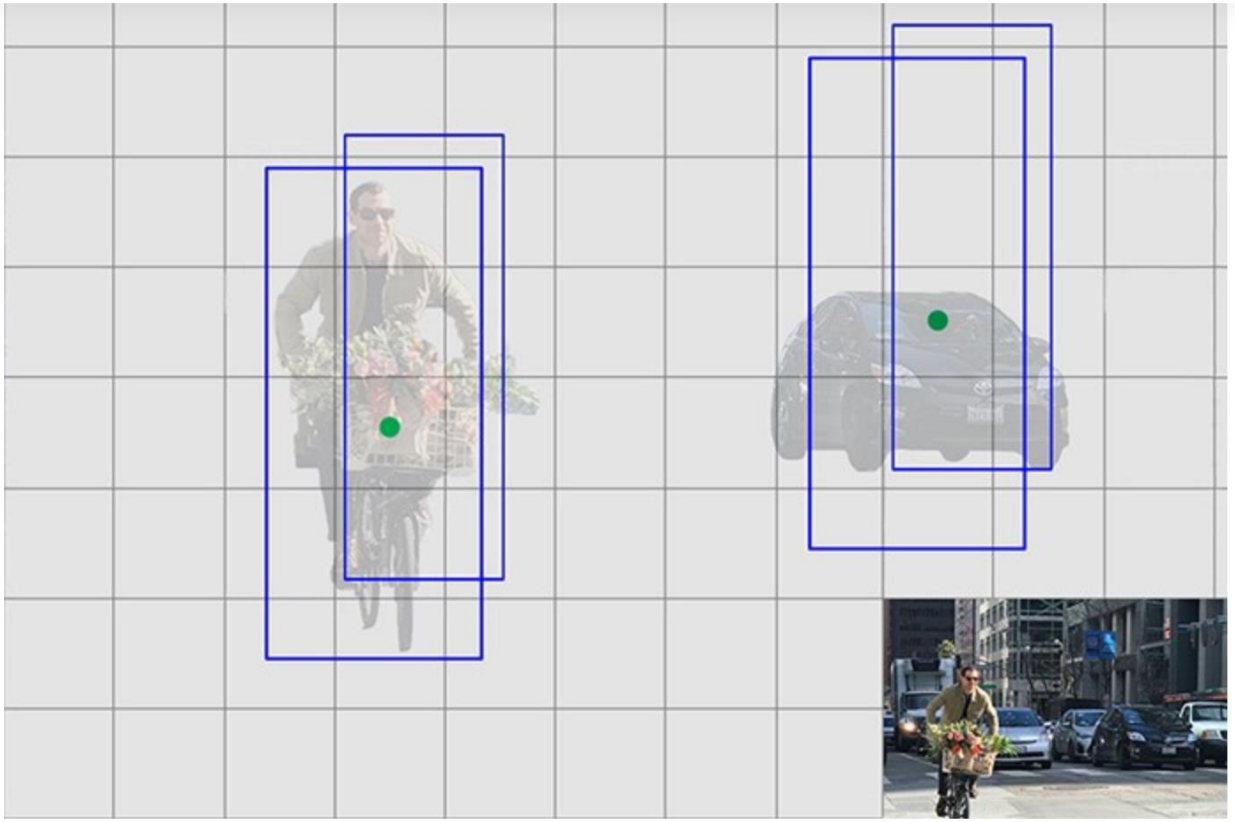
YOLO- You Only Look Once



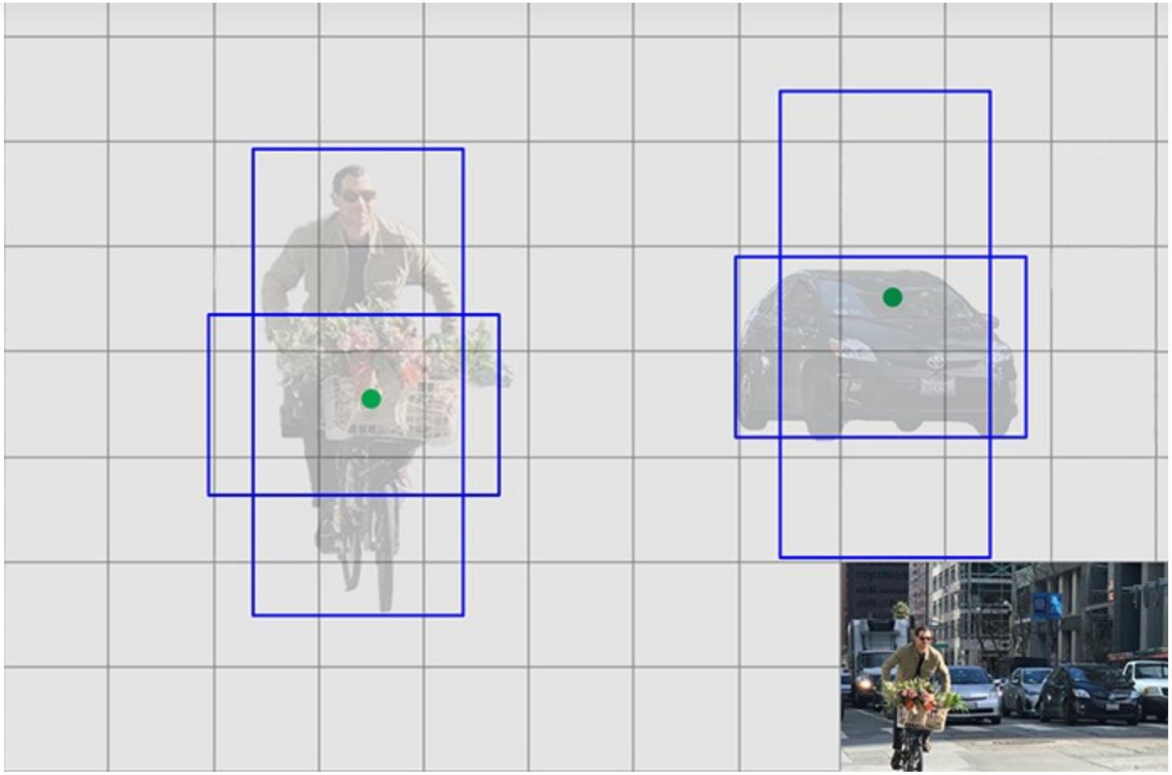
Divide the image into 7x7 cells.
Each cell trains a detector.
The detector needs to predict the object's class distributions.
The detector has 2 bounding-box predictors to predict bounding-boxes and confidence scores.

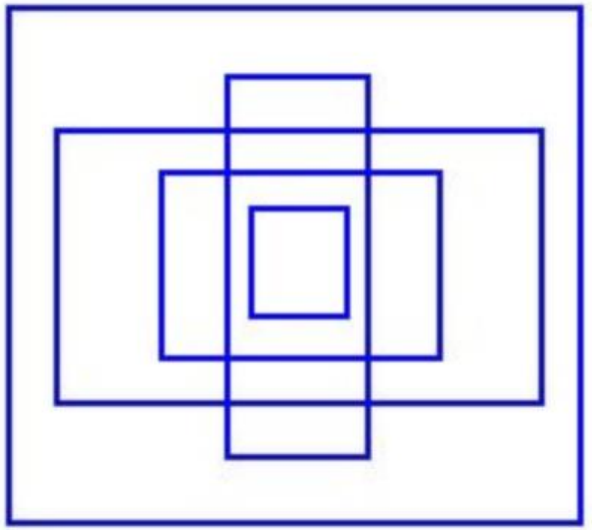
https://arxiv.org/abs/1506.02640 Redmon et al. CVPR 2016.

YOLO- You Only Look Once



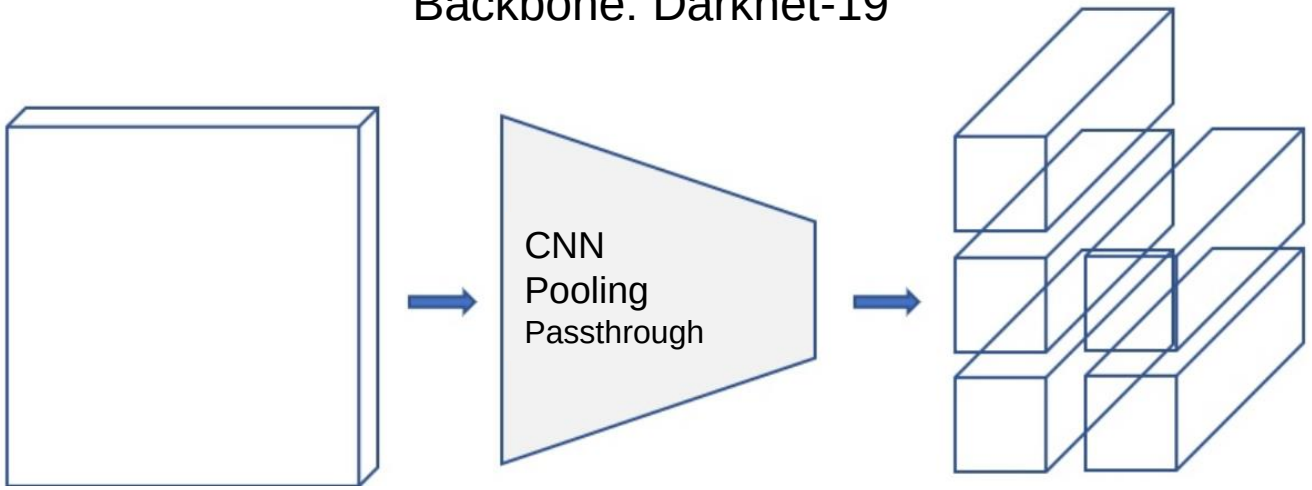
YOLO 2





YOLO 2

Backbone: Darknet-19



Input: 416x416*3

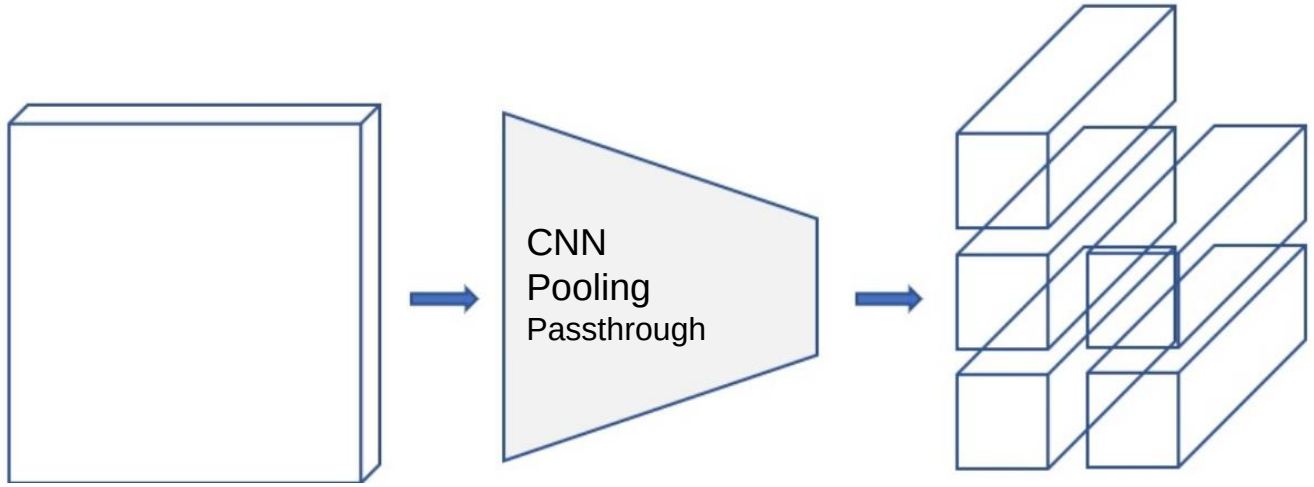
Output: 13x13 cell

Cell: 5 (anchor box) x 25

25: (x,y,w,h, score) + 20 objects

Output: 13x13x5x25

YOLO 2



Input: 416x416*3

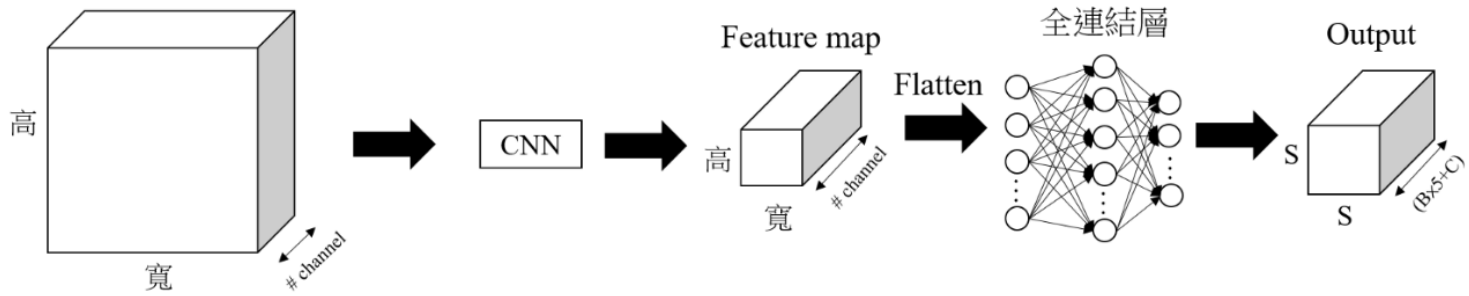
Output: 13x13 cell

414/32 = 13, 降了 2x2x2x2x2(5層)

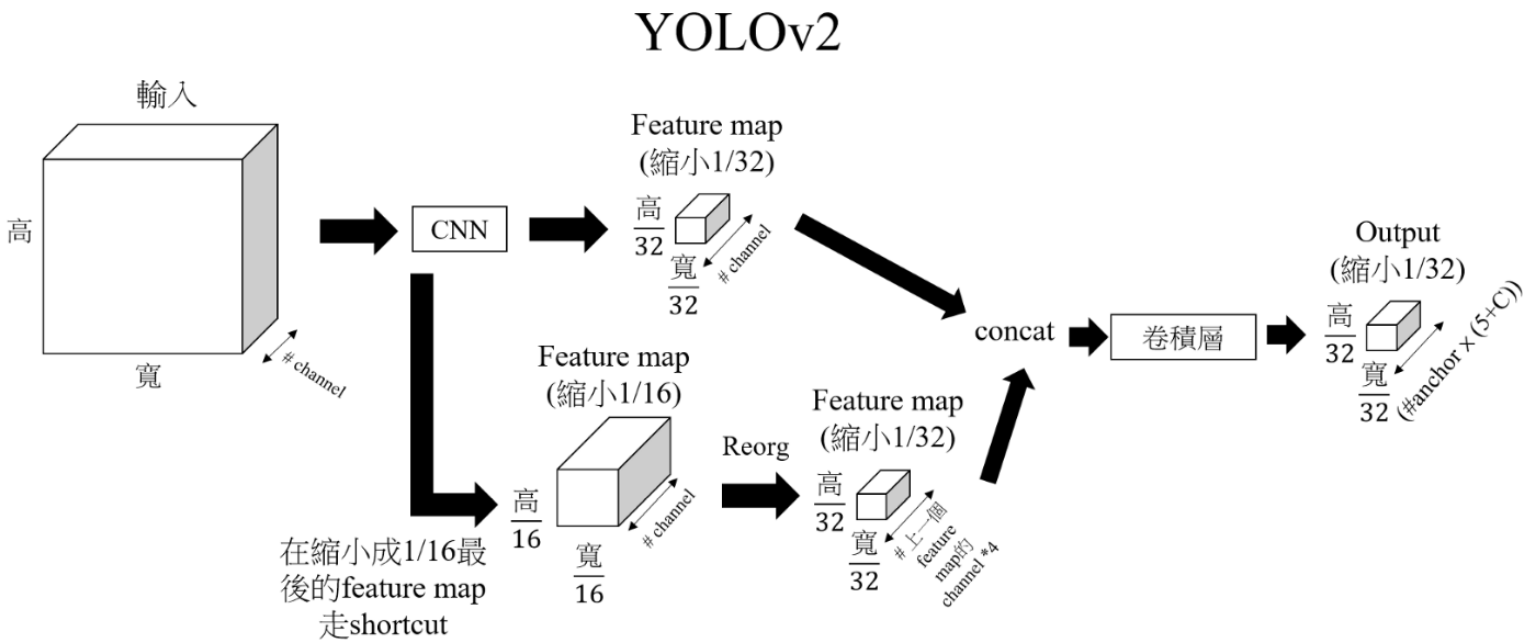
Cell: 5 (anchor box) 如何選呢？

K-means

YOLO 1


高
寬
channel
CNN
Feature map
高
寬
channel
Flatten
全連結層
Output
S
S
(Bx5+C)

YOLO 2


YOLOv2
輸入
高
寬
channel
CNN
Feature map
(縮小1/32)
高/32
寬/32
channel
Feature map
(縮小1/16)
高/16
寬/16
channel
在縮小成1/16最後的feature map走shortcut
Reorg
Feature map
(縮小1/32)
高/32
寬/32
上一個feature map的channel *4
concat
卷積層
Output
(縮小1/32)
高/32
寬/32
(#anchor x (5+C))

Yolo v2 – Redmon et al. 2017

- Batch normalization
- Pre-train on higher resolution ImageNet
- Use and improve anchor box idea from Faster RCNN
- Train at multiple resolutions
- Very good accuracy, very fast

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

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

https://youtu.be/VOC3huqHrss

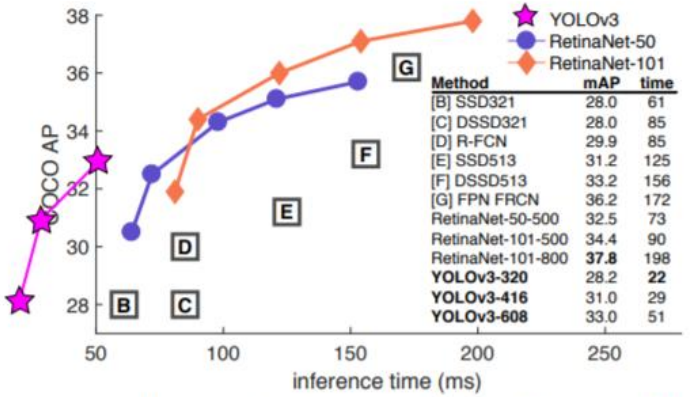
YOLOV3

YOLOv3: An Incremental Improvement

Joseph Redmon, Ali Farhadi
University of Washington

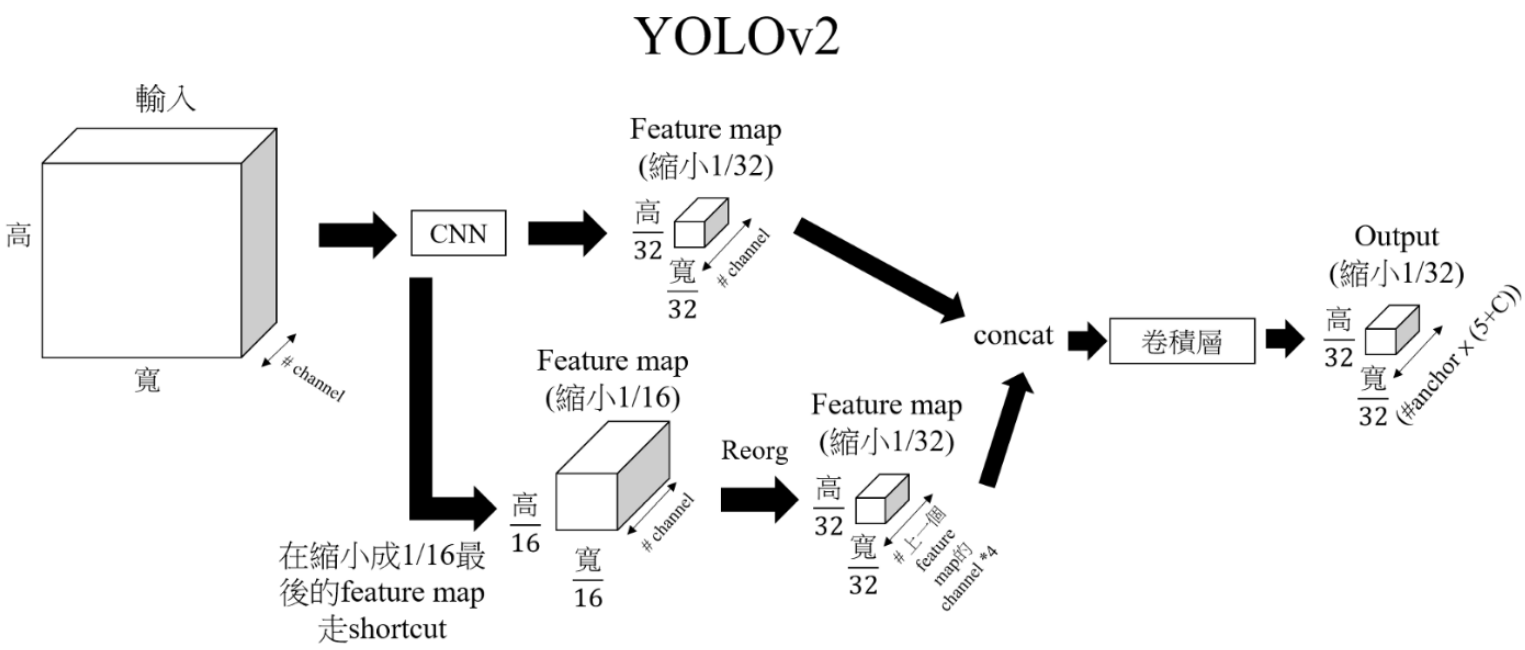
Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At 320 × 320 YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP_{50} in 51 ms on a Titan X, compared to 57.5 AP_{50} in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at https://pjreddie.com/yolo/.



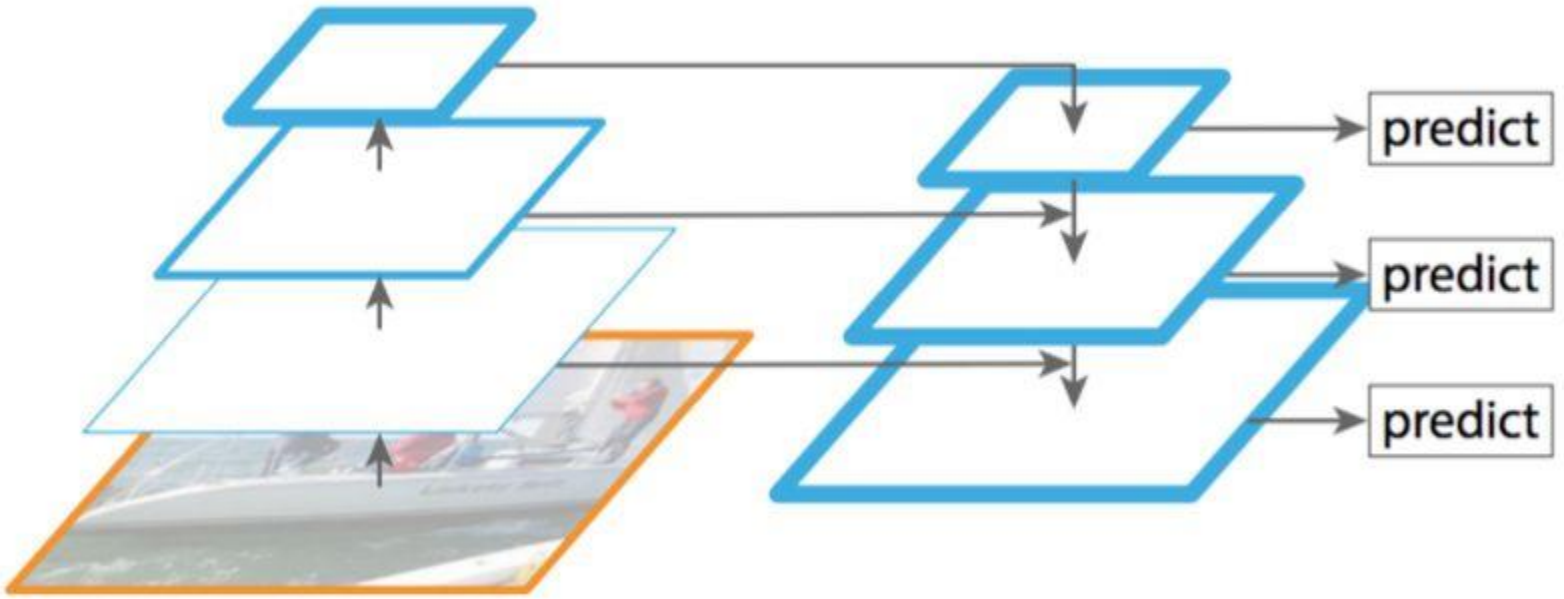
Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods

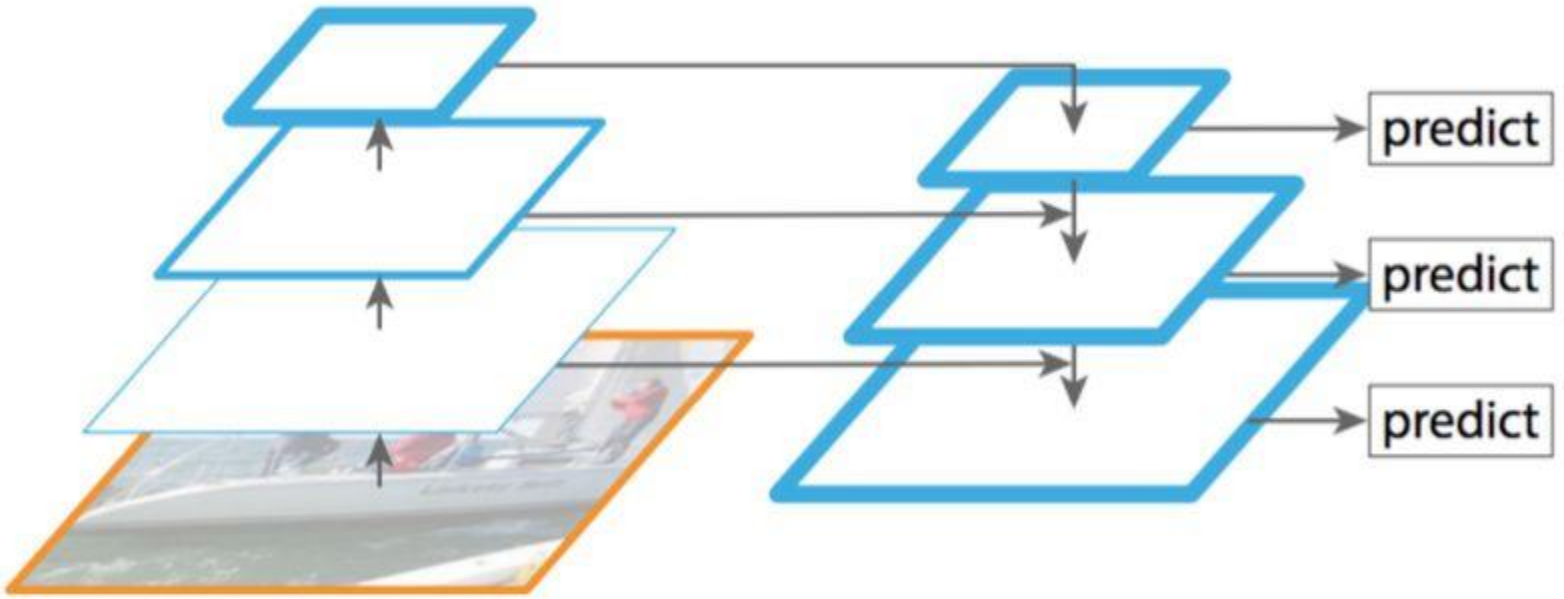


Backbone: Darknet-19

YOLO3

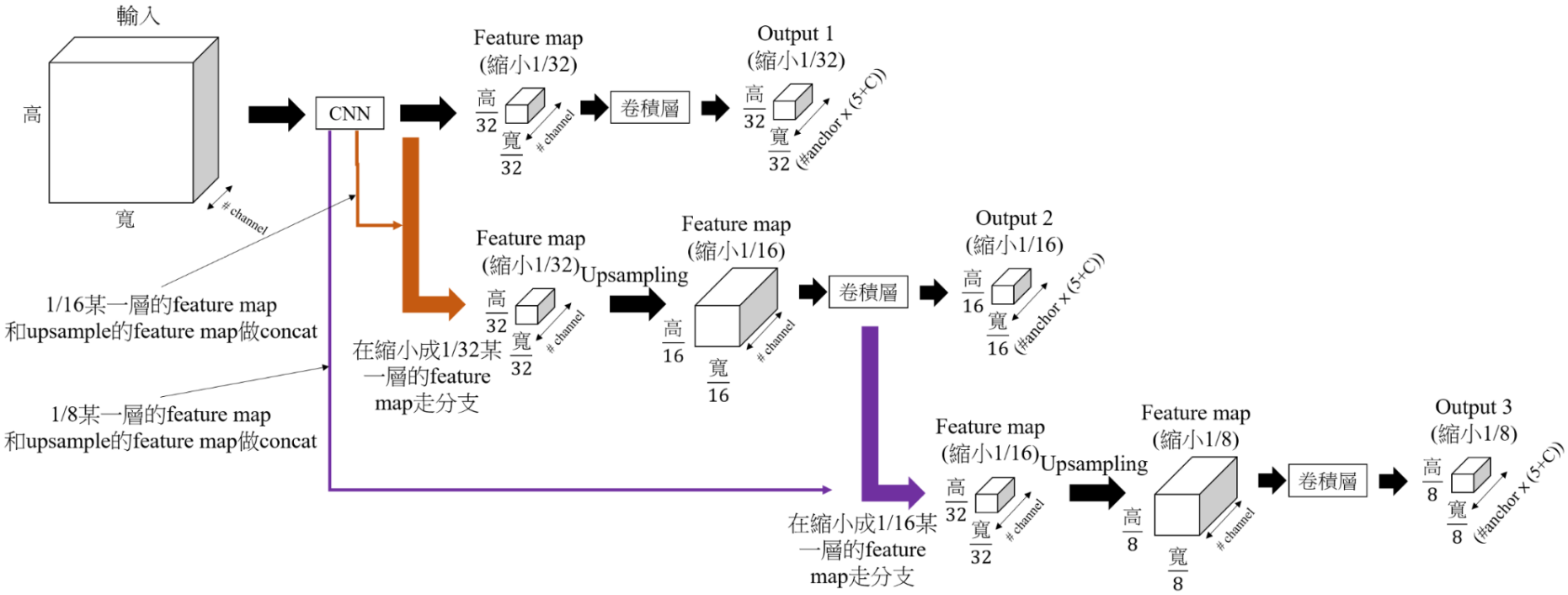


YOLO3



Backbone: Darknet-53

YOLO 3



Backbone: Darknet-53

YOLOV3

YOLO v3中使用53層的卷積網路

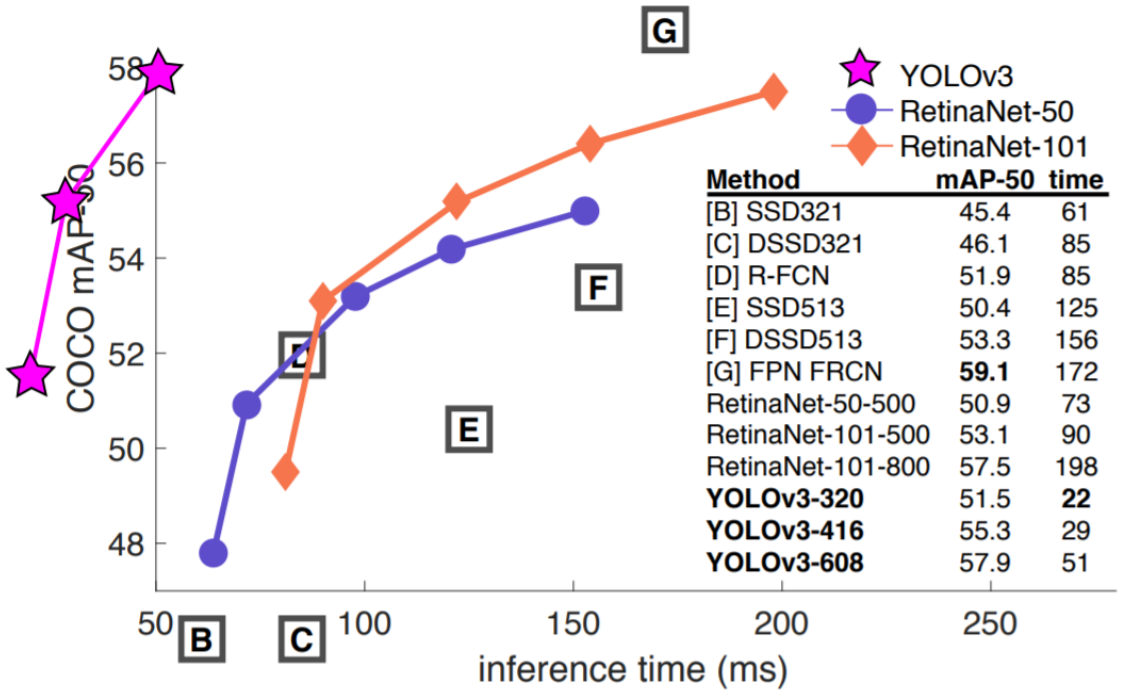
不同於YOLO2，YOLO v3從三種不同尺度的特徵圖譜上進行預測任務

模型不再使用softmax函數作為最終的分類器，使用logistic作為分類器使用 binary cross-entropy作為損失函數

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

YOLOV3 –Performance

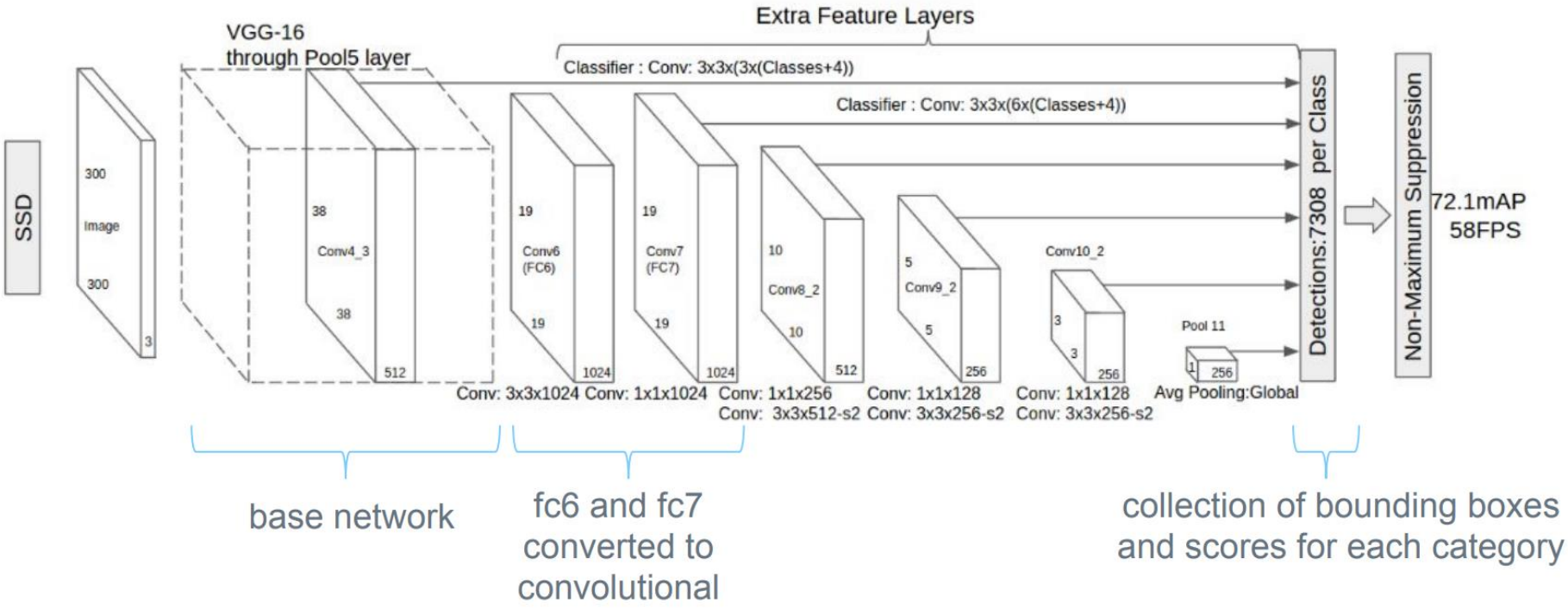


Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

https://pjreddie.com/media/files/papers/YOLOv3.pdf

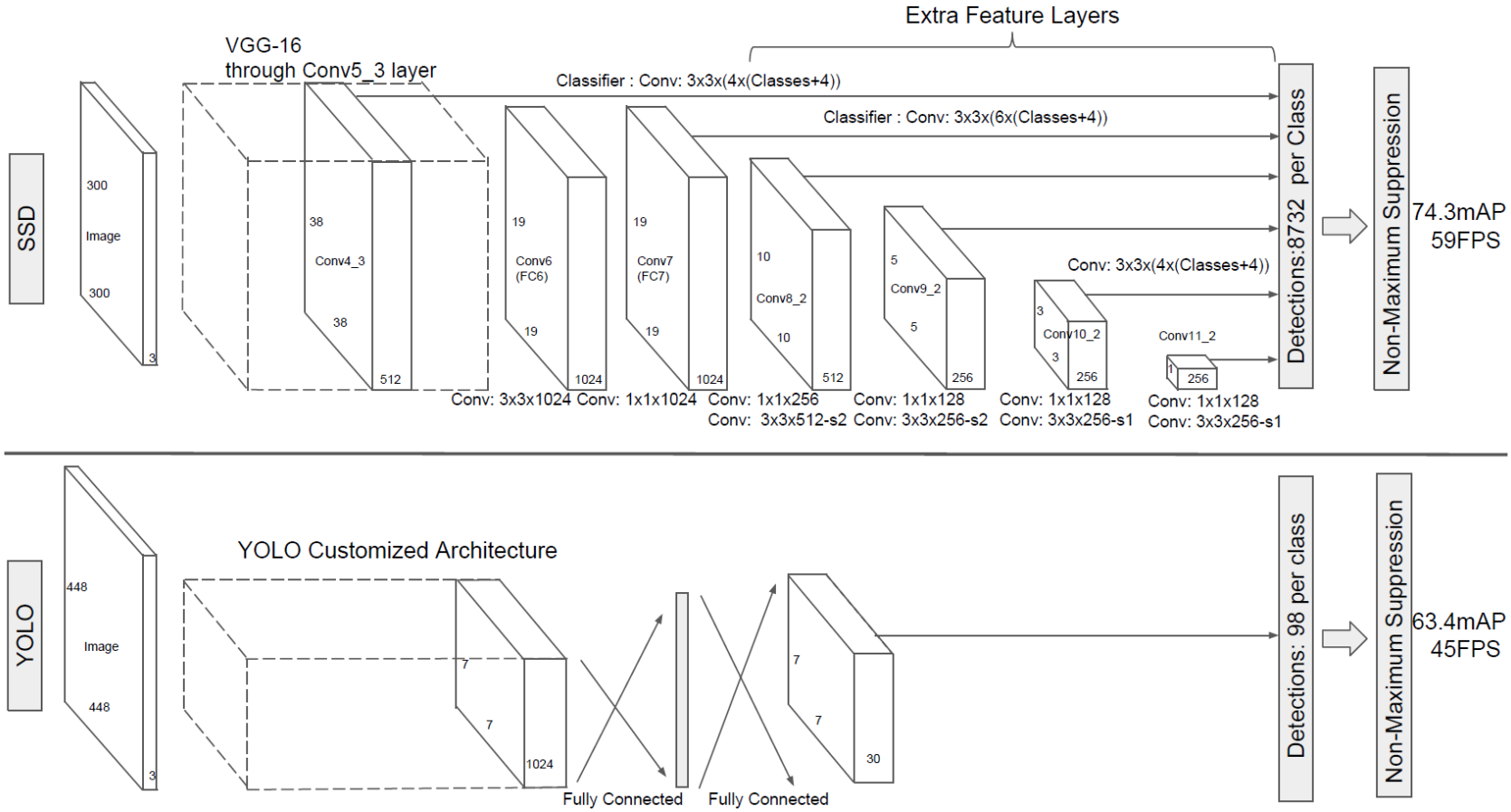
SSD: Single Shot Detector

Liu et al. ECCV 2016.

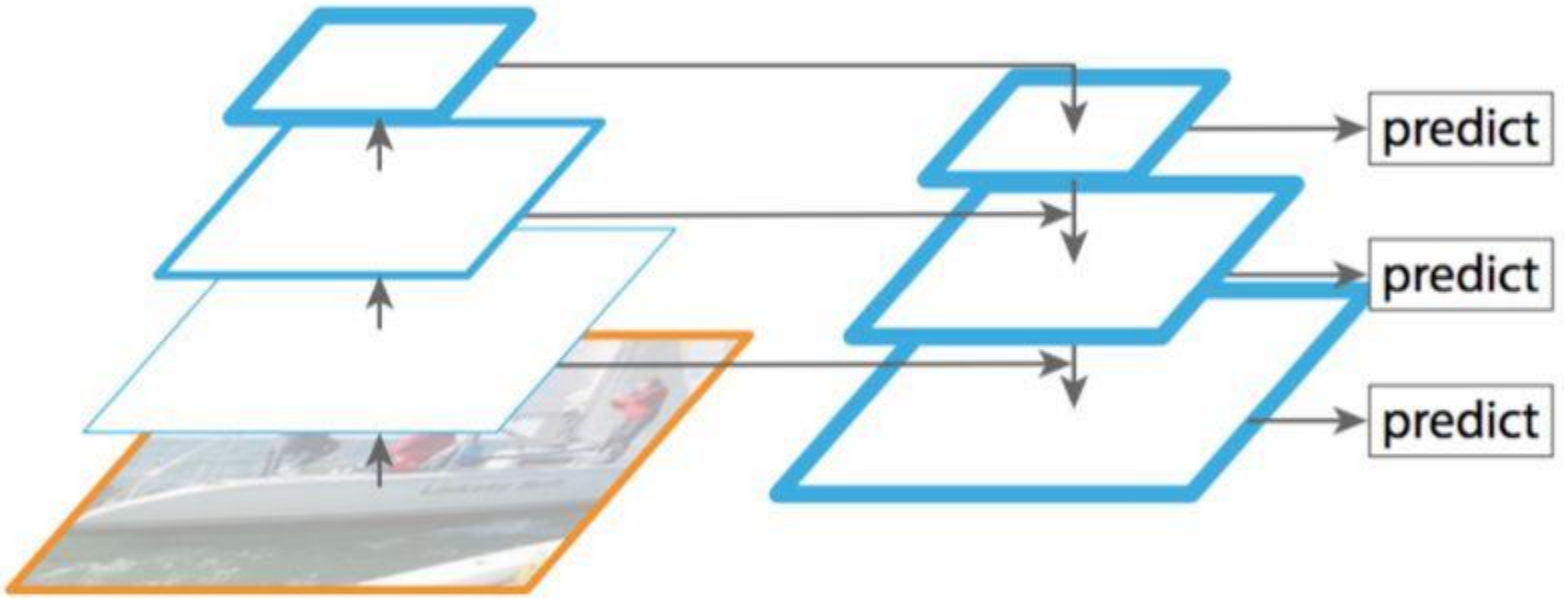


Multi-scale feature maps for detection: observe how conv feature maps decrease in size and allow predictions at multiple scales

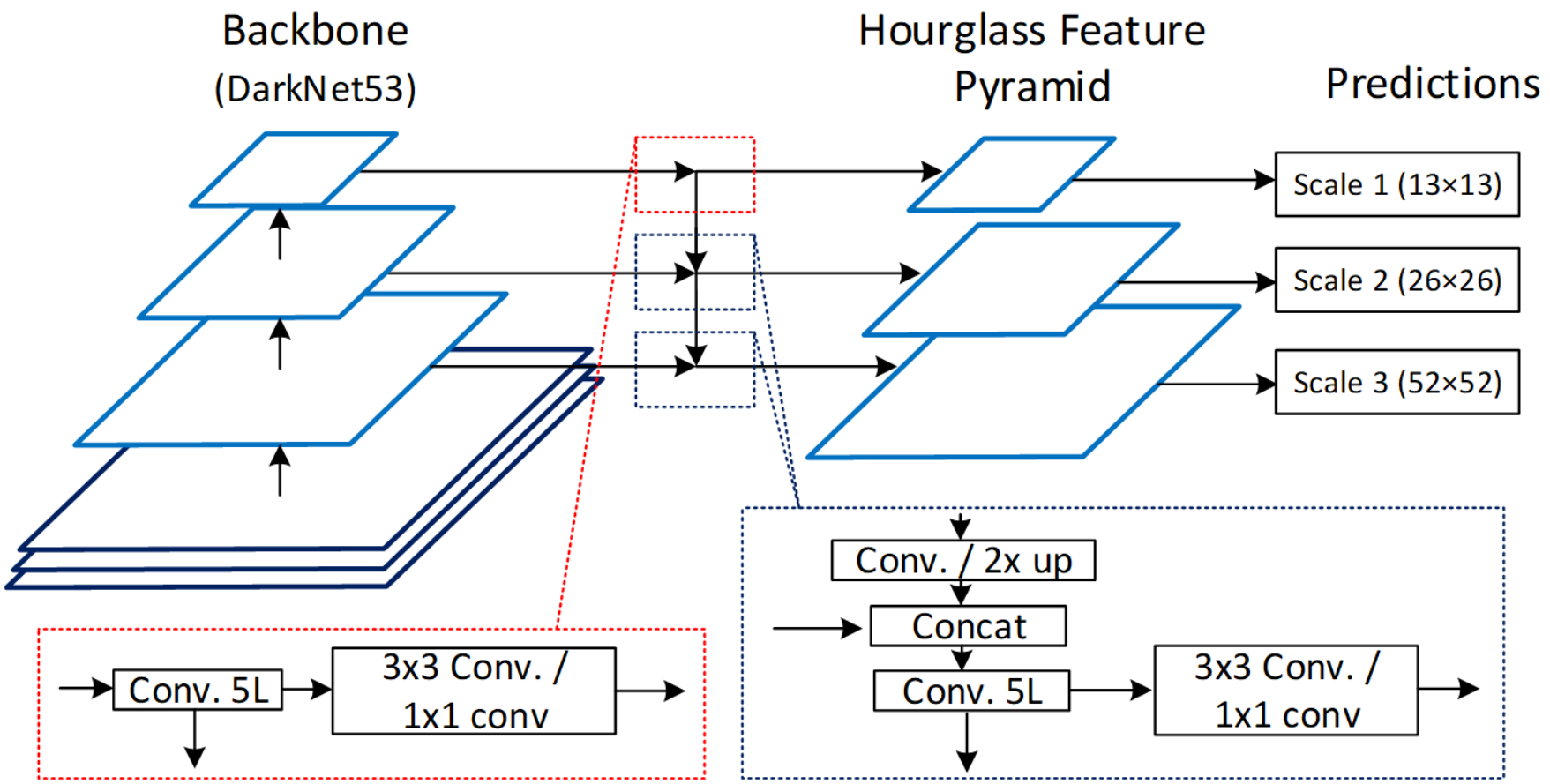
SSD 與 YOLO 比較


SSD
300
Image
300
3
VGG-16
through Conv5_3 layer
38
Conv4_3
38
512
19
Conv6
(FC6)
19
1024
19
Conv7
(FC7)
19
1024
10
Conv8_2
10
512
5
Conv9_2
5
256
3
Conv10_2
3
256
Conv11_2
1
256
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256
Conv: 3x3x512-s2
Conv: 1x1x128
Conv: 3x3x256-s2
Conv: 1x1x128
Conv: 3x3x256-s1
Conv: 1x1x128
Conv: 3x3x256-s1
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP
59FPS
YOLO
448
Image
448
3
YOLO Customized Architecture
7
7
1024
Fully Connected
Fully Connected
7
7
30
Detections: 98 per class
Non-Maximum Suppression
63.4mAP
45FPS

YOLO3

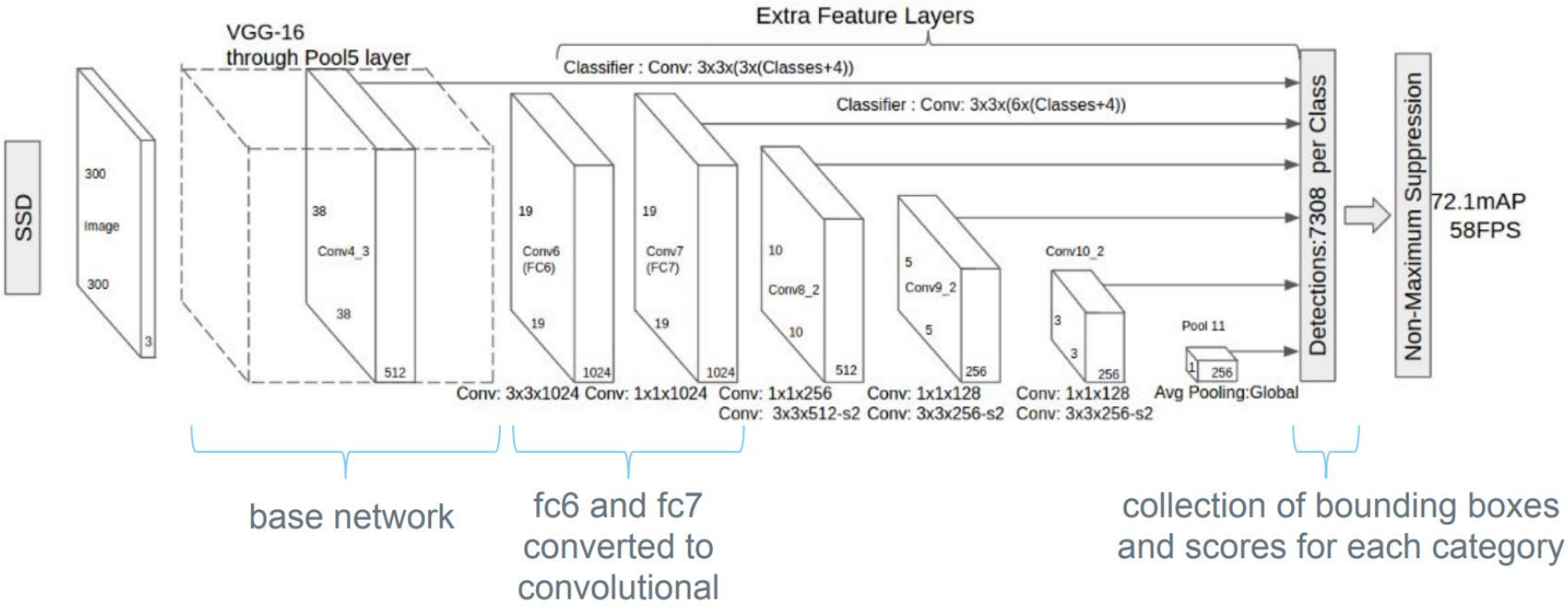


YOLO3


Backbone
(DarkNet53)
Hourglass Feature
Pyramid
Predictions
Scale 1 (13×13)
Scale 2 (26×26)
Scale 3 (52×52)
Conv. 5L
3x3 Conv. /
1x1 conv
Conv. / 2x up
Concat
Conv. 5L
3x3 Conv. /
1x1 conv

SSD: Single Shot Detector

Liu et al. ECCV 2016.



Convolutional predictors for detection: a set of filters that predict detections for different asp

SSD: Single Shot Detector

Liu et al. ECCV 2016.

What is a detection ?

Described by **four parameters** (center bounding box x and y, width and height)

Class category

For all categories we need for a detection a total of #classes + 4 values

SSD: Single Shot Detector

Liu et al. ECCV 2016.

Different "classes" of detections

aspect ratio 2:1
for cats

aspect ratio 1:2
for cats

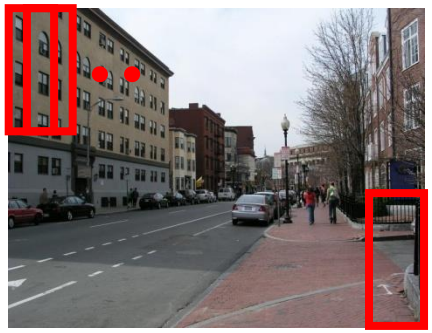
aspect ratio 1:1
for cats

Influential Works in Detection

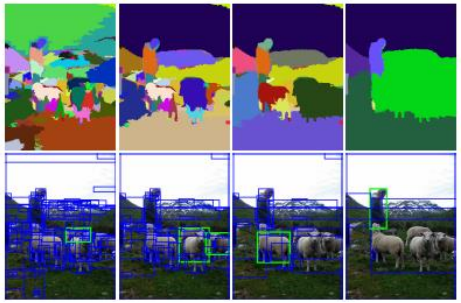
- Sung-Poggio (1994, 1998) : ~2412 citations
  - Basic idea of statistical template detection (I think), bootstrapping to get "face-like" negative examples, multiple whole-face prototypes (in 1994)
- Rowley-Baluja-Kanade (1996-1998) : ~4953
  - "Parts" at fixed position, non-maxima suppression, simple cascade, rotation, pretty good accuracy, fast
- Schneiderman-Kanade (1998-2000,2004) : ~2600
  - Careful feature/classifier engineering, excellent results, cascade
- Viola-Jones (2001, 2004) : ~27,000
  - Haar-like features, Adaboost as feature selection, hyper-cascade, very fast, easy to implement
- Dalal-Triggs (2005) : ~18000
  - Careful feature engineering, excellent results, HOG feature, online code
- Felzenszwalb-Huttenlocher (2000): ~2100
  - Efficient way to solve part-based detectors
- Felzenszwalb-McAllester-Ramanan (2008,2010): ~7200
  - Excellent template/parts-based blend
- Girshick-Donahue-Darrell-Malik (2014): ~4700
  - Region proposals + fine-tuned CNN features (marks significant advance in accuracy over hog-based methods)
- Redmon, Divvala, Girshick, Farhadi (2016): ~210
  - Refine and simplify RCNN++ approach to predict directly from last conv layer

Summary: statistical templates

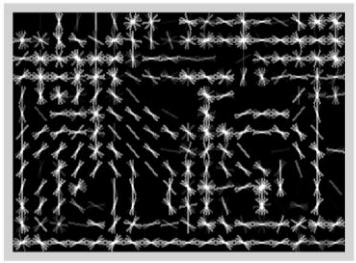
Propose Window → Extract Features → Classify → Post-process



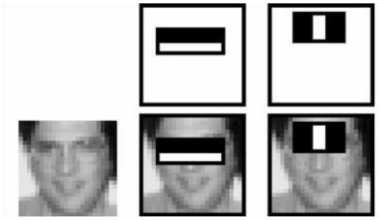
Sliding window: scan image pyramid



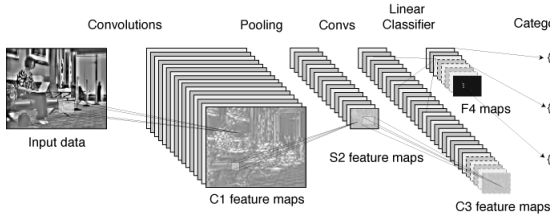
Region proposals: edge/region-based, resize to fixed window



HOG



Fast randomized features



CNN features

SVM

Boosted stubs

Neural network

Non-max suppression

Segment or refine localization

On the Design of Modern Deep Detectors

Backbone: the feature extractor from which the detector draws its discriminative power.

Same backbone: **Region Proposal Network,** VGGNet-16

Faster-RCNN: [Ren et al., 2015]

R-FCN: [Dai et al., 2016]

SSD [Liu et al., 2016]

On the Design of Modern Deep Detectors

Common Backbones

1. AlexNet:
2. VGGNet: [Simonyan and Zisserman, 2014]
3. Inception: [Ioffe, 2017, Szegedy et al., 2015, 2016]
4. ResNet: [He et al., 2016]

On the Design of Modern Deep Detectors

New Backbones

1. Inception-ResNet: [Szegedy et al., 2017]
2. Xception: group convolution [Chollet, 2017]
3. DarkNet [Redmon and Farhadi, 2017]
4. DenseNet: [Huang et al., 2017b]
5. Hourglass [Newell et al., 2016]
6. Wide-Residual Net: [Lee et al., 2017]
7. ResNeXt [Xie et al., 2017]
8. Dilated Residual Networks: [Yu et al., 2017]
9. DetNet [Li et al., 2018]

For speed

10. MobileNet: [Howard et al., 2017]
11. SqueezeNet [Iandola et al., 2016]
12. ShuffleNet [Zhang et al., 2017]
13. PeLLee [NIPS 2018]