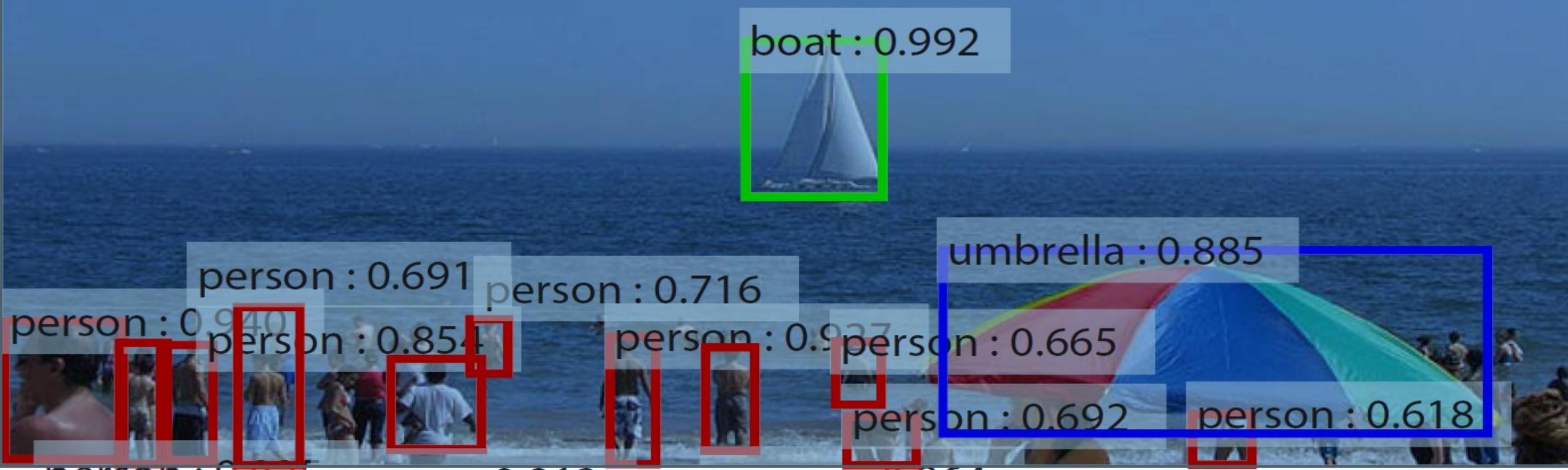


# Object Detection

Fast Campus  
Start Deep Learning with TensorFlow



# Computer Vision Task

**Classification**



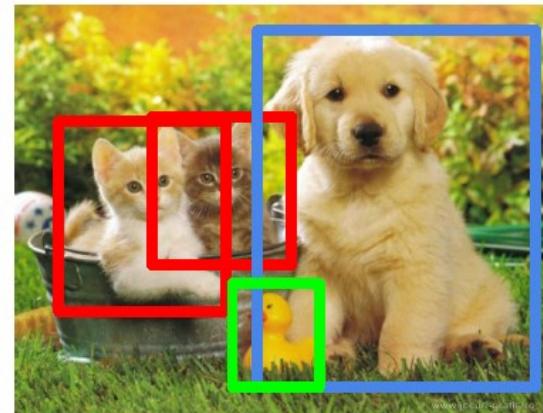
CAT

**Classification + Localization**



CAT

**Object Detection**



CAT, DOG, DUCK

**Instance Segmentation**



CAT, DOG, DUCK

Single object

Multiple objects

# Classification+Localization

**Classification:** C classes

**Input:** Image

**Output:** Class label

**Evaluation metric:** Accuracy



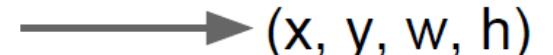
CAT

**Localization:**

**Input:** Image

**Output:** Box in the image ( $x, y, w, h$ )

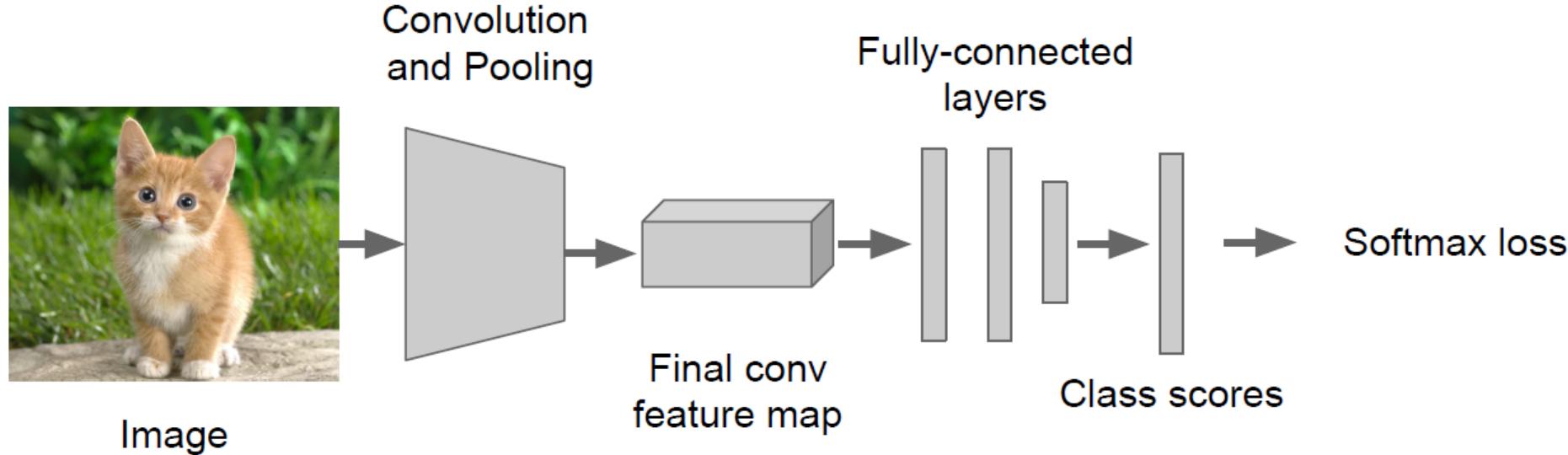
**Evaluation metric:** Intersection over Union



**Classification + Localization:** Do both

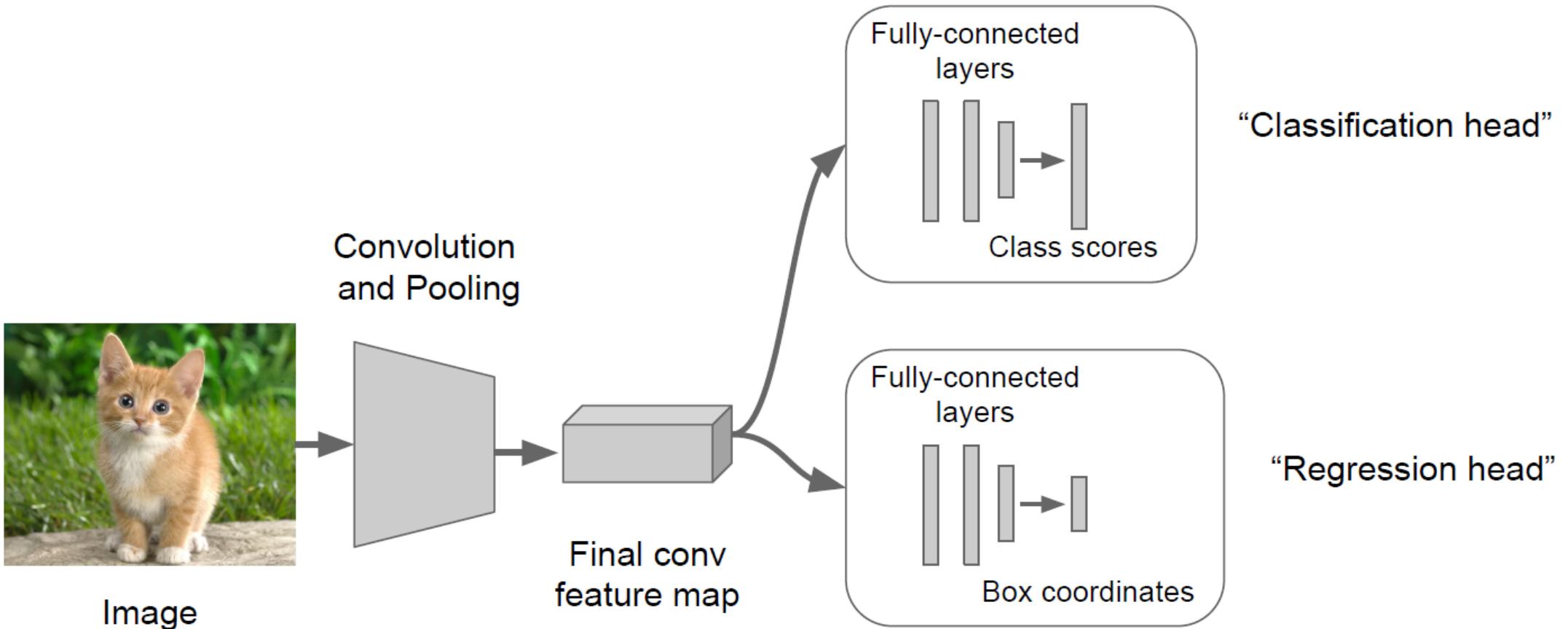
# Simple Idea

**Step 1:** Train (or download) a classification model (AlexNet, VGG, GoogLeNet)



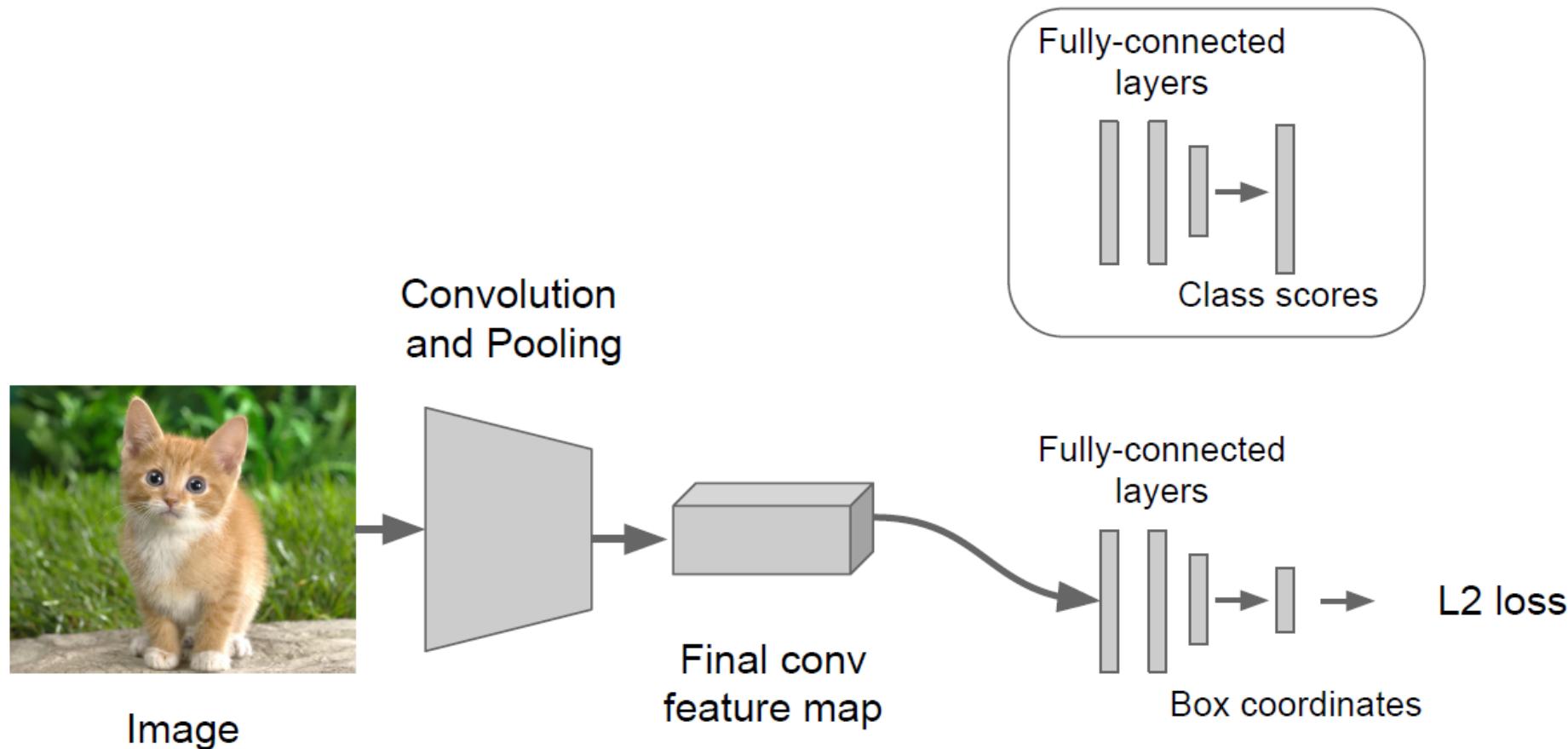
# Simple Idea

**Step 2:** Attach new fully-connected “regression head” to the network



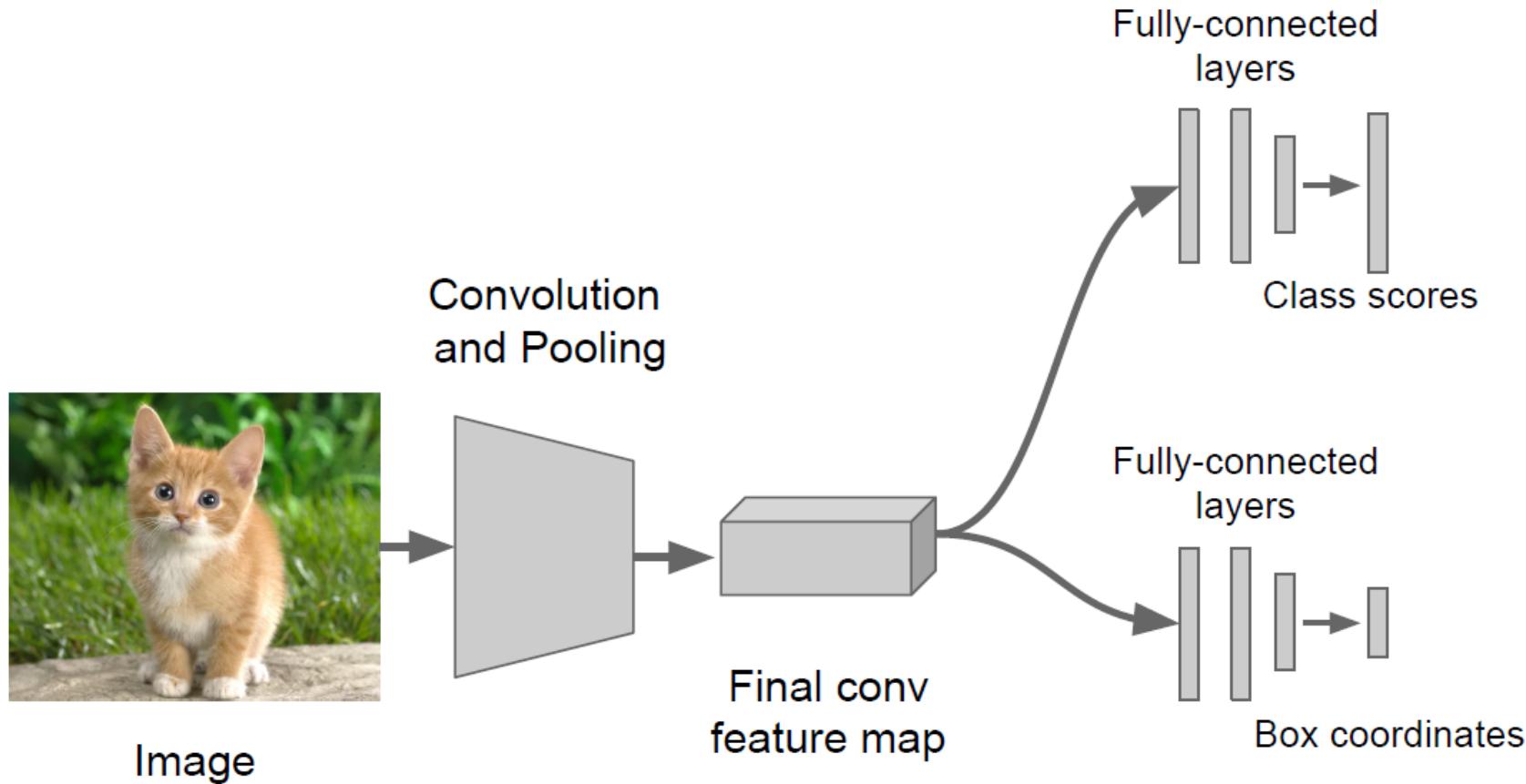
# Simple Idea

**Step 3:** Train the regression head only with SGD and L2 loss



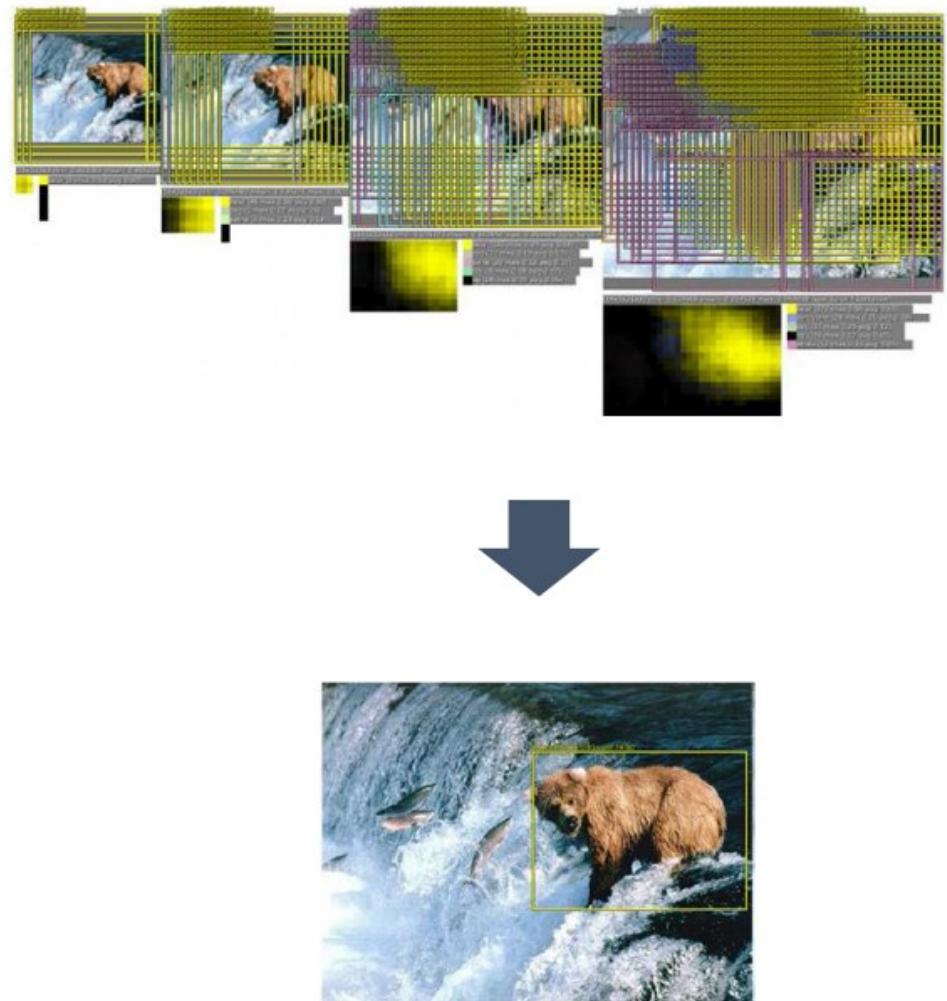
# Simple Idea

**Step 4:** At test time use both heads



# How to Find BBox

- Intuitive Approach – Sliding window
  - Simple but takes a lot of time
  - Multi-scale testing is necessary
  - Procedure
    - Perform classification at each location
    - Perform bounding-box regression on all the classified regions
    - Merge bounding boxes



# Object Detection

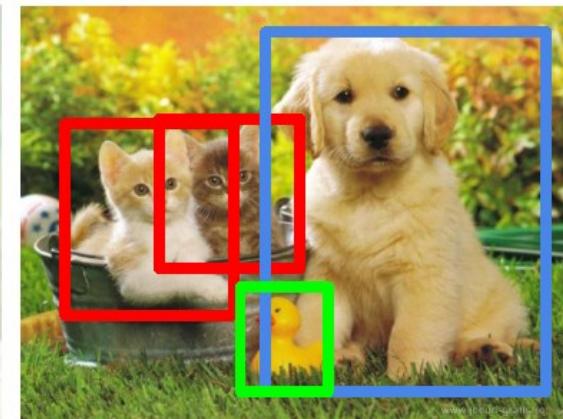
Classification



Classification  
+ Localization



Object Detection



Instance  
Segmentation



# Detection as Regression?



CAT, (x, y, w, h)

CAT, (x, y, w, h)

....

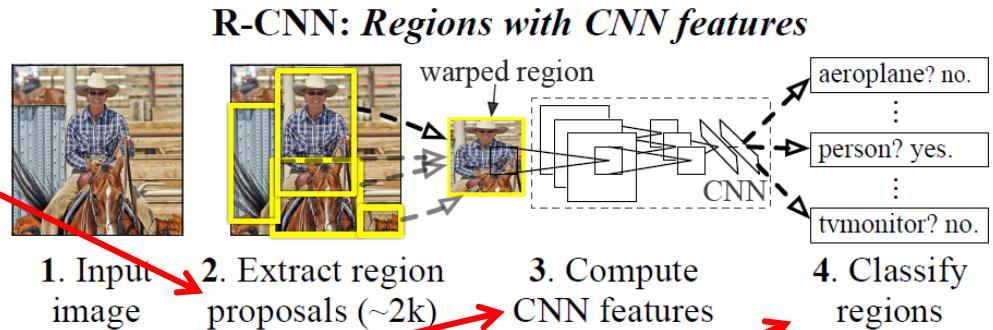
CAT (x, y, w, h)

= many numbers

Need variable sized outputs

# R-CNN

1. Generating category independent region proposals
2. Extracting a fixed length feature vector from CNN
3. Class specific linear SVM



**Figure 1: Object detection system overview.** Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. R-CNN achieves a mean average precision (mAP) of **53.7% on PASCAL VOC 2010**. For comparison, [39] reports 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. The popular deformable part models perform at 33.4%. On the 200-class **ILSVRC2013 detection dataset**, R-CNN's **mAP is 31.4%**, a large improvement over OverFeat [34], which had the previous best result at 24.3%.

# R-CNN Architecture

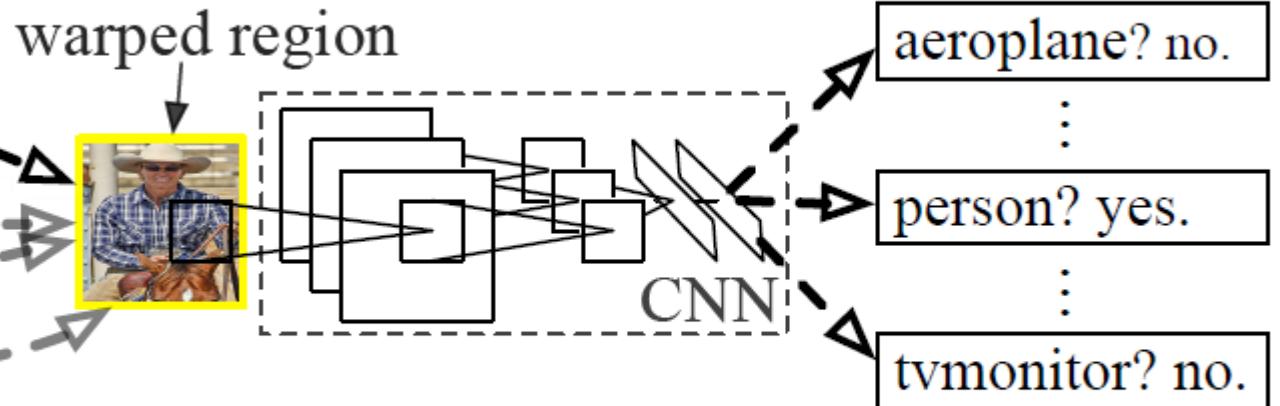
**R-CNN: *Regions with CNN features***



1. Input  
image



2. Extract region  
proposals (~2k)

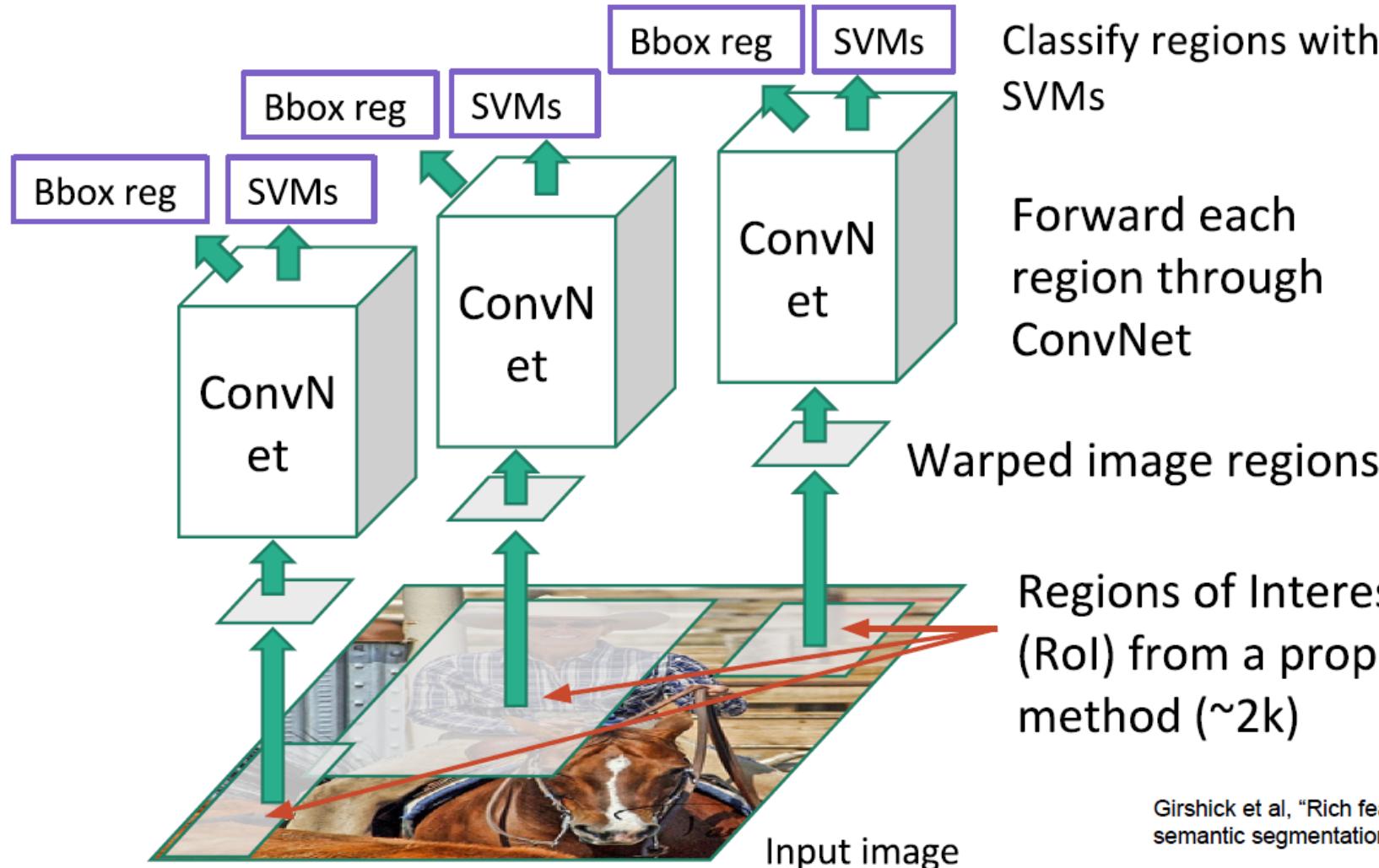


3. Compute  
CNN features

4. Classify  
regions

# R-CNN

Linear Regression for bounding box offsets



Classify regions with SVMs

Forward each region through ConvNet

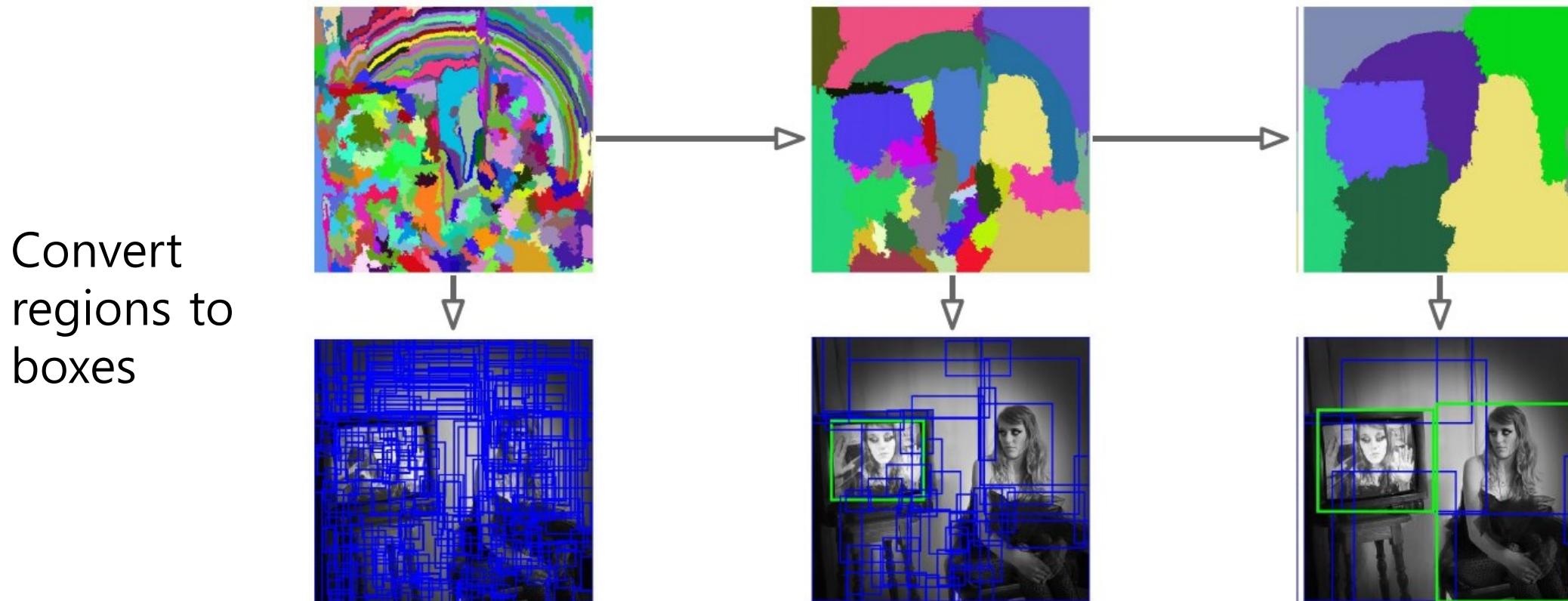
Warped image regions

Regions of Interest (RoI) from a proposal method (~2k)

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

# Region Proposals – Selective Search

- Bottom-up segmentation, merging regions at multiple scales



# Feature Extraction

- AlexNet is used

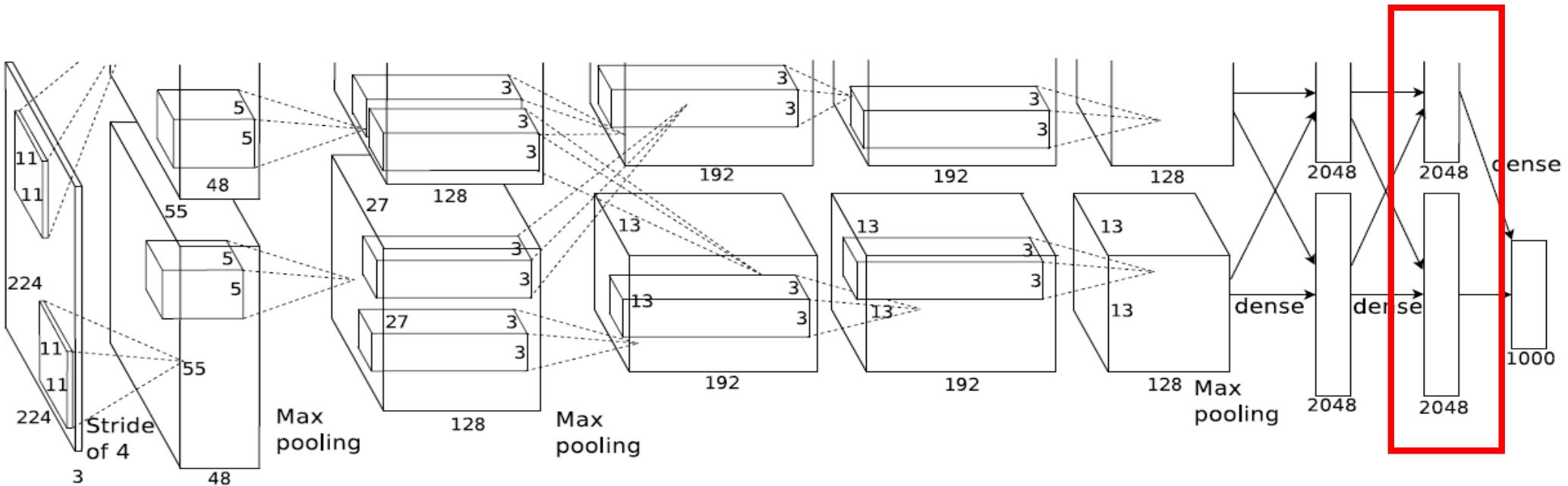


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

# Test Time

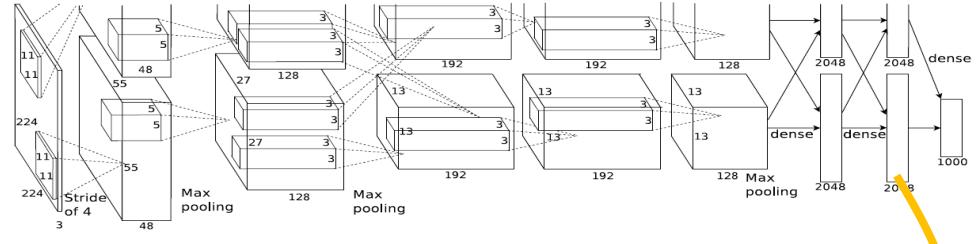
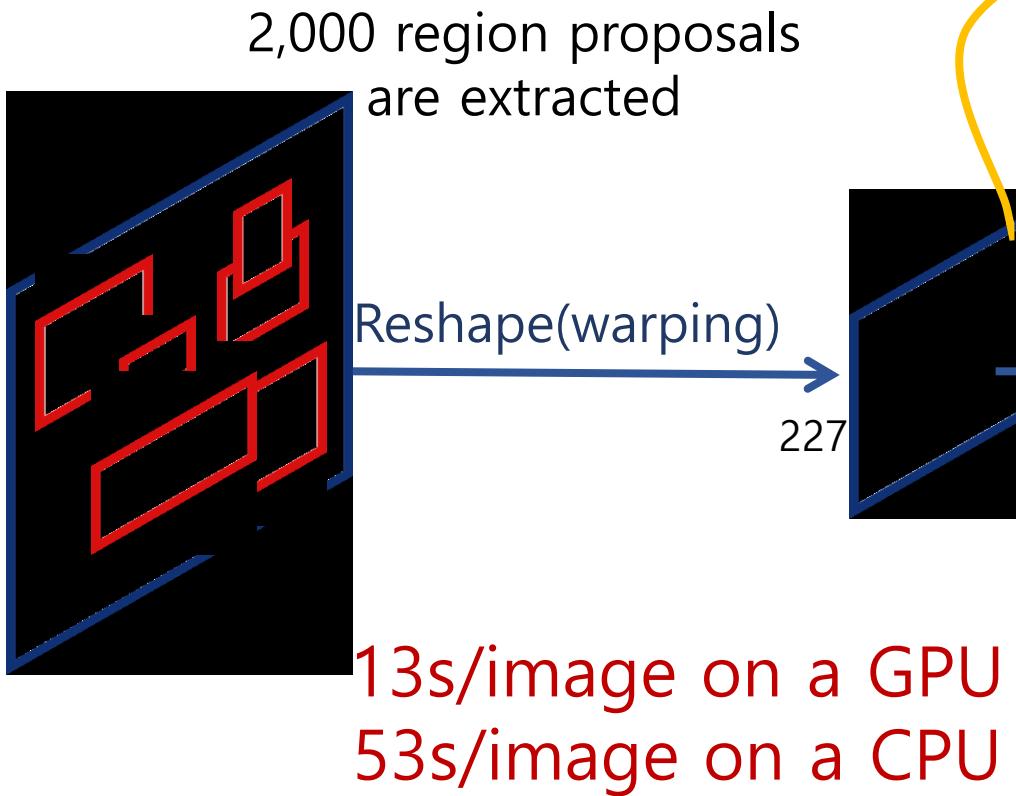


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,214–4096–4096–1000.

# Training(Region Proposal)

Positive : groundtruth

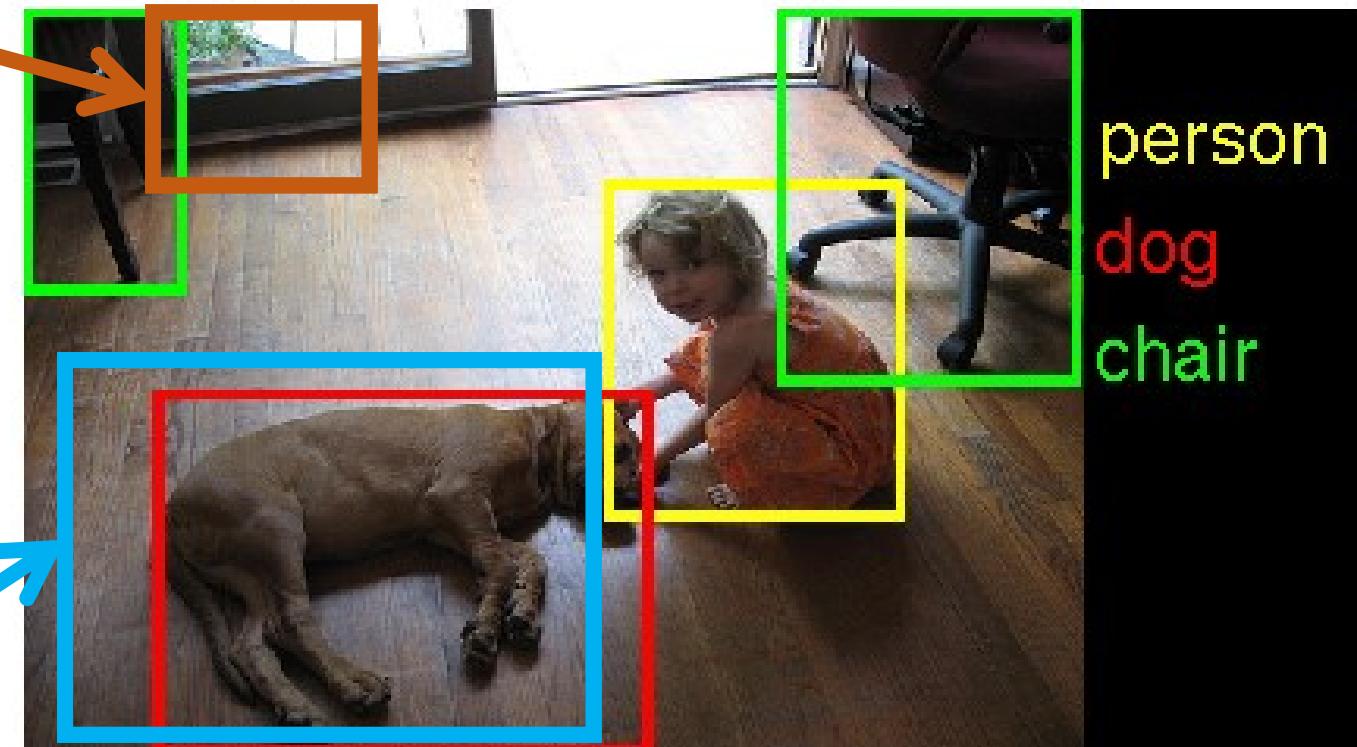
Negative (BG) : if  $\text{IoU} \leq 0.3$

Training SVM

Fine-tuning

Positive (FG) : if  $\text{IoU} \geq 0.5$

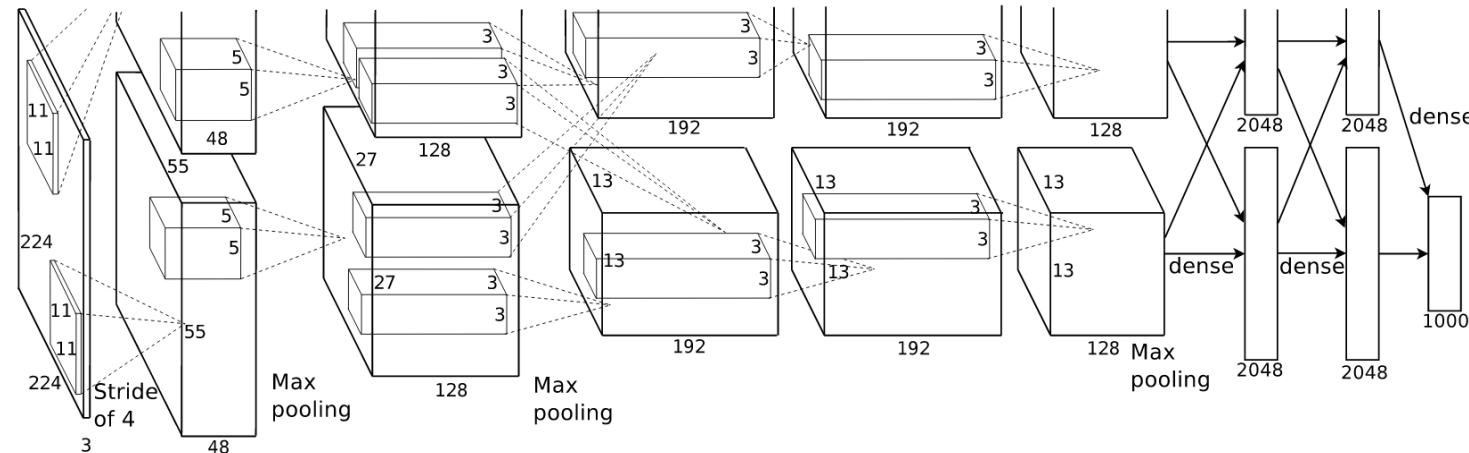
Negative (BG) : else



# R-CNN Training

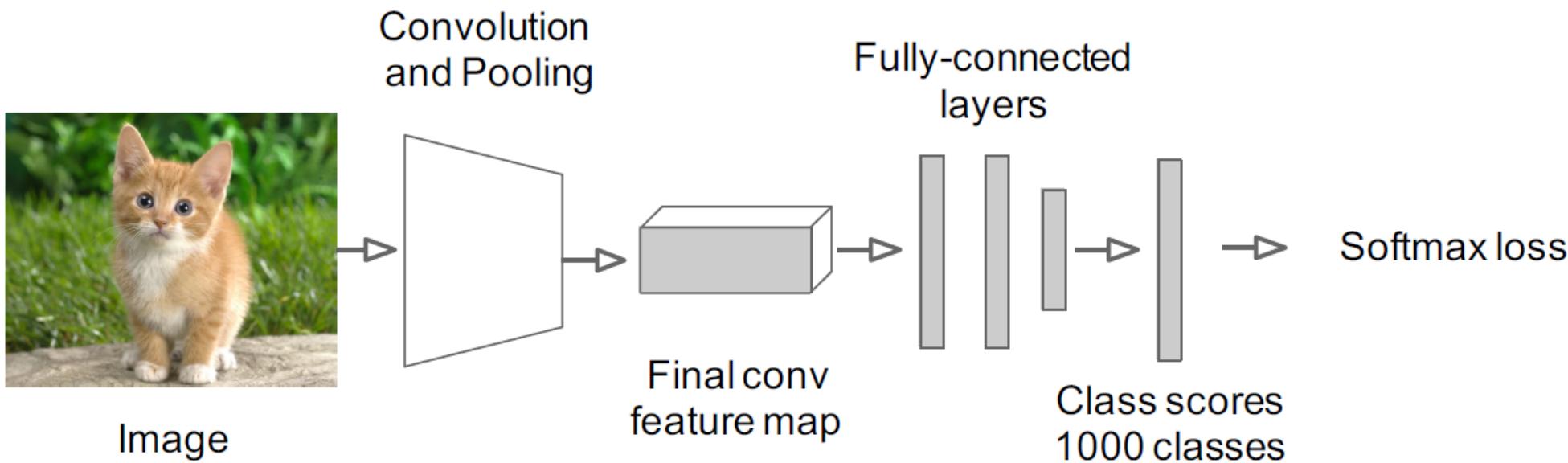
"Post hoc" means the parameters are learned after the ConvNet is fixed

- Pre-train a ConvNet(AlexNet) for ImageNet classification dataset
- Fine-tune for object detection(softmax + log loss)
- Cache feature vectors to disk
- Train post hoc linear SVMs(hinge loss)
- Train post hoc linear bounding-box regressors(squared loss)



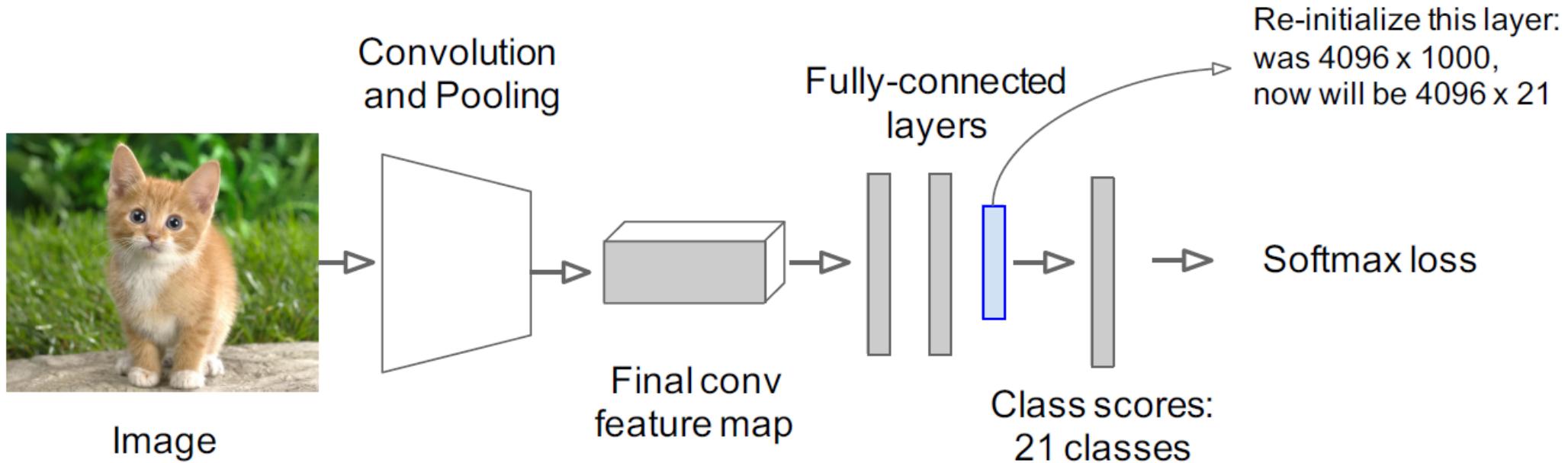
# Training

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



# 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

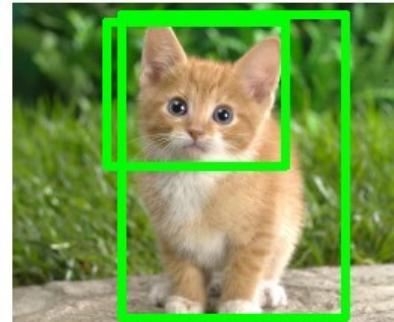


# Training

- 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!



Image

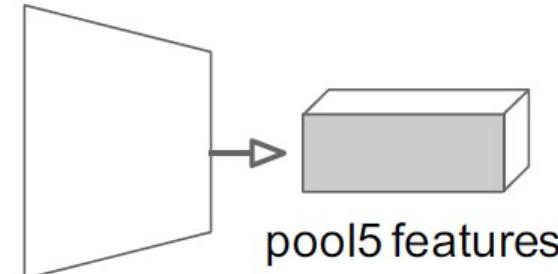


Region Proposals

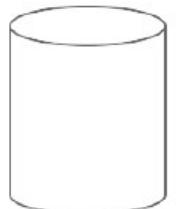


Crop + Warp

Convolution  
and Pooling



Forward pass



Save to disk

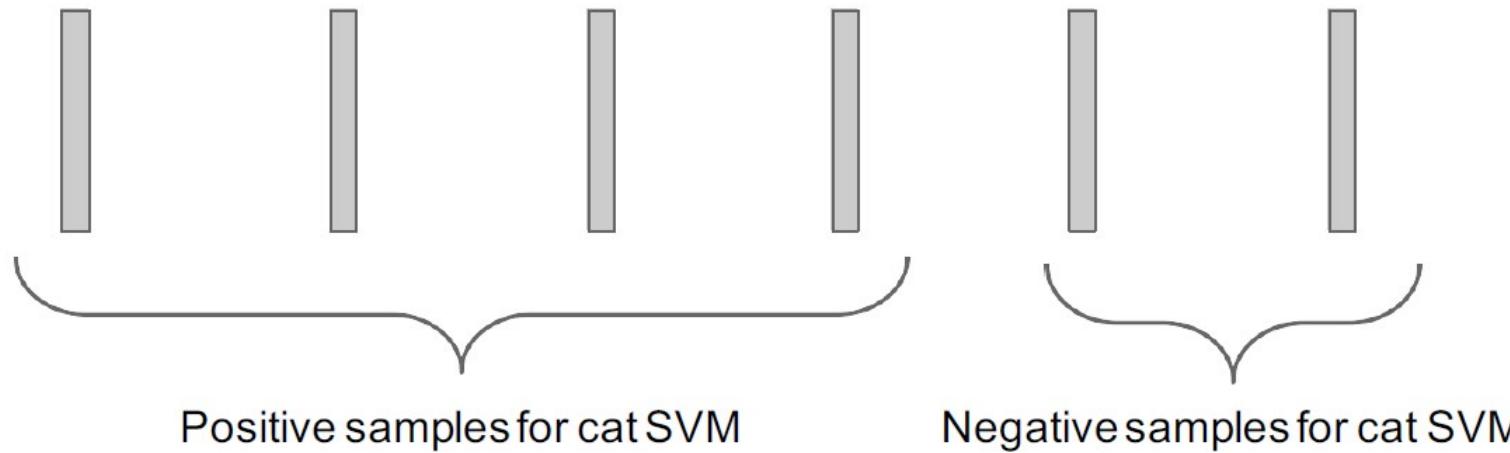
# Training

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

Training image regions



Cached region features



# 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

# Bounding-Box Regression

$P^i = (P_x^i, P_y^i, P_w^i, P_h^i)$  specifies the pixel coordinates of the center of proposal  $P^i$ 's bounding box together with  $P^i$ 's width and height in pixels

$G = (G_x, G_y, G_w, G_h)$  means the ground-truth bounding box

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$

$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$

$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$

$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$

# Bounding-Box Regression

Each function  $d_\star(P)$  (where  $\star$  is one of  $x, y, h, w$ ) is modeled as a linear function of the pool<sub>5</sub> features of proposal  $P$ , denoted by  $\phi_5(P)$ . (The dependence of  $\phi_5(P)$  on the image data is implicitly assumed.) Thus we have

$d_\star(P) = \mathbf{w}_\star^T \phi_5(P)$  where  $\mathbf{w}_\star$  is a vector of learnable model parameters. We learn  $\mathbf{w}_\star$  by optimizing the regularized least squares objective (ridge regression):

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^T \phi_5(P^i))^2 + \lambda \|\hat{\mathbf{w}}_\star\|^2. \quad (5)$$

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$

$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$

$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$

$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$

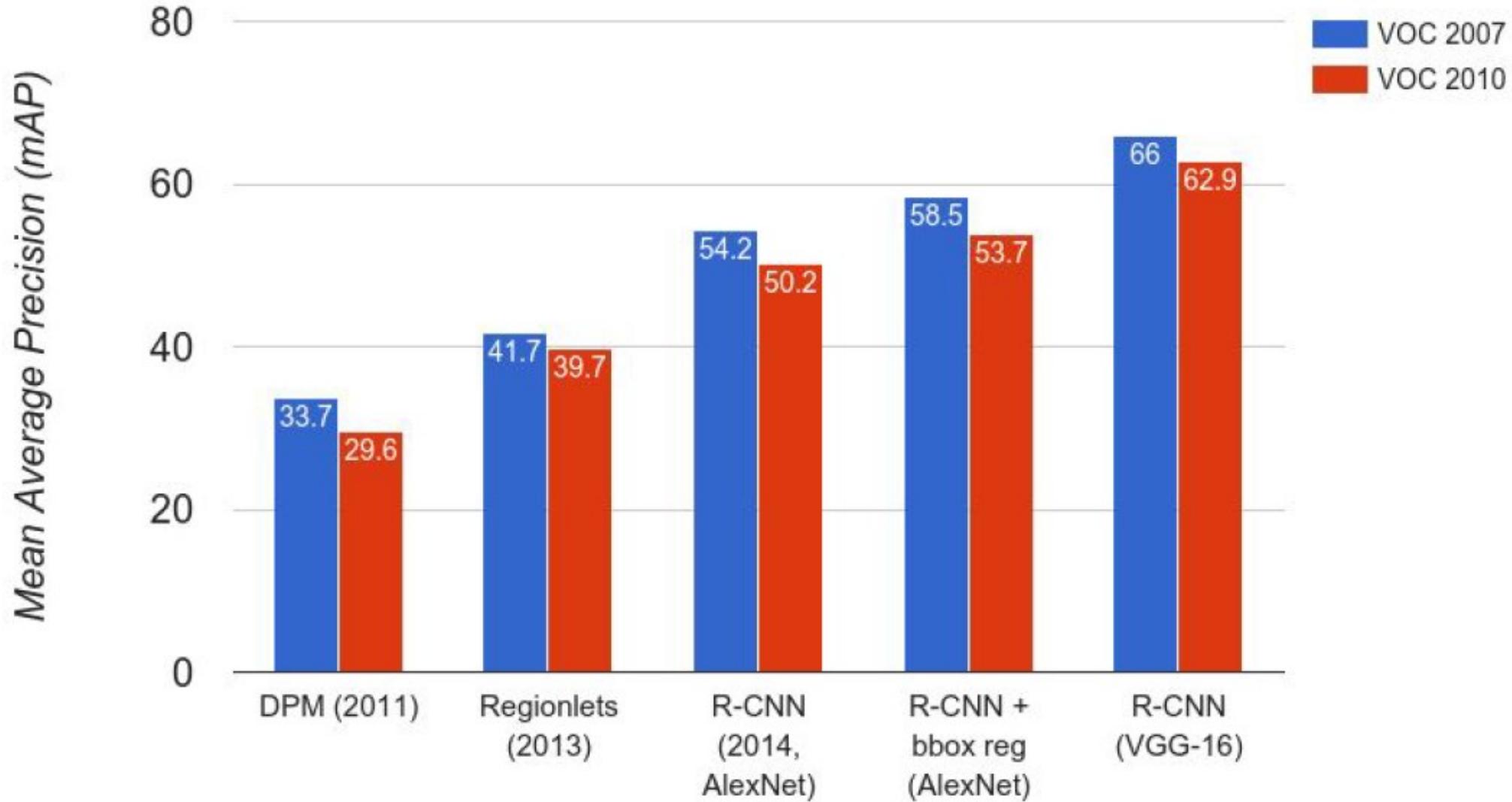
$$t_x = (G_x - P_x)/P_w \quad (6)$$

$$t_y = (G_y - P_y)/P_h \quad (7)$$

$$t_w = \log(G_w/P_w) \quad (8)$$

$$t_h = \log(G_h/P_h). \quad (9)$$

# Results



# Additional Experiment

- Performance layer by layer

VOC 2007 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN pool <sub>5</sub>	51.8	60.2	36.4	27.8	23.2	52.8	60.6	49.2	18.3	47.8	44.3	40.8	56.6	58.7	42.4	23.4	46.1	36.7	51.3	55.7	44.2
R-CNN fc <sub>6</sub>	59.3	61.8	43.1	34.0	25.1	53.1	60.6	52.8	21.7	47.8	42.7	47.8	52.5	58.5	44.6	25.6	48.3	34.0	53.1	58.0	46.2
R-CNN fc <sub>7</sub>	57.6	57.9	38.5	31.8	23.7	51.2	58.9	51.4	20.0	50.5	40.9	46.0	51.6	55.9	43.3	23.3	48.1	35.3	51.0	57.4	44.7
R-CNN FT pool <sub>5</sub>	58.2	63.3	37.9	27.6	26.1	54.1	66.9	51.4	26.7	55.5	43.4	43.1	57.7	59.0	45.8	28.1	50.8	40.6	53.1	56.4	47.3
R-CNN FT fc <sub>6</sub>	63.5	66.0	47.9	37.7	29.9	62.5	70.2	60.2	32.0	57.9	47.0	53.5	60.1	64.2	52.2	31.3	55.0	50.0	57.7	63.0	53.1
R-CNN FT fc <sub>7</sub>	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
R-CNN FT fc <sub>7</sub> BB	<b>68.1</b>	<b>72.8</b>	<b>56.8</b>	<b>43.0</b>	<b>36.8</b>	<b>66.3</b>	<b>74.2</b>	<b>67.6</b>	<b>34.4</b>	<b>63.5</b>	<b>54.5</b>	<b>61.2</b>	<b>69.1</b>	<b>68.6</b>	<b>58.7</b>	<b>33.4</b>	<b>62.9</b>	<b>51.1</b>	<b>62.5</b>	<b>64.8</b>	<b>58.5</b>
DPM v5 [20]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
DPM ST [28]	23.8	58.2	10.5	8.5	27.1	50.4	52.0	7.3	19.2	22.8	18.1	8.0	55.9	44.8	32.4	13.3	15.9	22.8	46.2	44.9	29.1
DPM HSC [31]	32.2	58.3	11.5	16.3	30.6	49.9	54.8	23.5	21.5	27.7	34.0	13.7	58.1	51.6	39.9	12.4	23.5	34.4	47.4	45.2	34.3

fc7 layer : 29%, 16.8M parameter

conv1~pool5 layer : 6%

# More Details...

- Which one is the best?



**Figure 7: Different object proposal transformations.** (A) the original object proposal at its actual scale relative to the transformed CNN inputs; (B) tightest square with context; (C) tightest square without context; (D) warp. Within each column and example proposal, the top row corresponds to  $p = 0$  pixels of context padding while the bottom row has  $p = 16$  pixels of context padding.

# Problems of R-CNN

- **Slow at test-time:** need to run full forward path of CNN for each region proposal
  - 13s/image on a GPU(K40)
  - 53s/image on a CPU
- **SVM and regressors are post-hoc:** CNN features not updated in response to SVMs and regressors
- **Complex multistage training pipeline** (84 hours using K40 GPU)
  - Fine-tune network with softmax classifier(log loss)
  - Train post-hoc linear SVMs(hinge loss)
  - Train post-hoc bounding-box regressions(squared loss)

# SPP-Net



Figure 1: Top: cropping or warping to fit a fixed size. Middle: a conventional CNN. Bottom: our spatial pyramid pooling network structure.

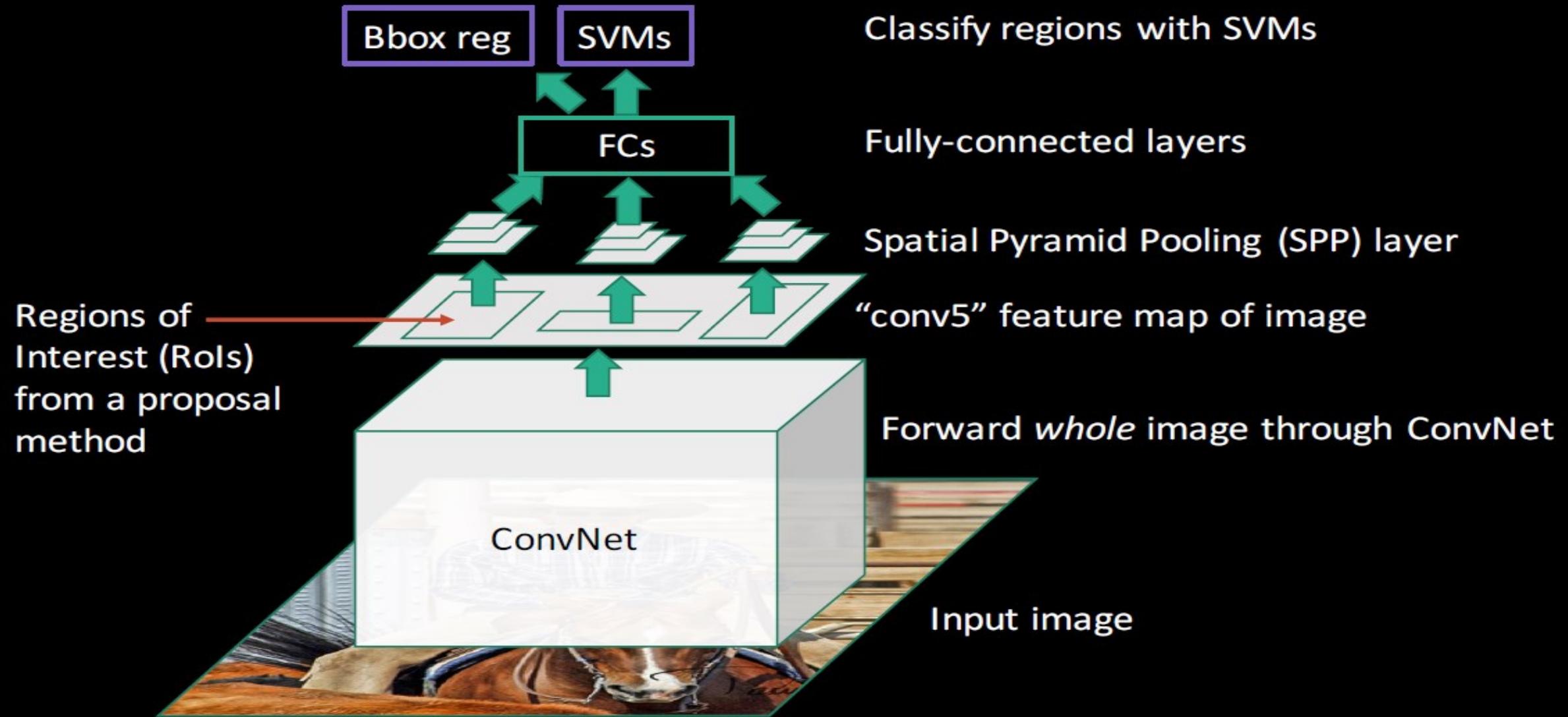
# SPP-Net's Idea

- R-CNN is time-consuming
  - It repeatedly applies the deep CNN to the raw pixels of thousands of warped regions per image

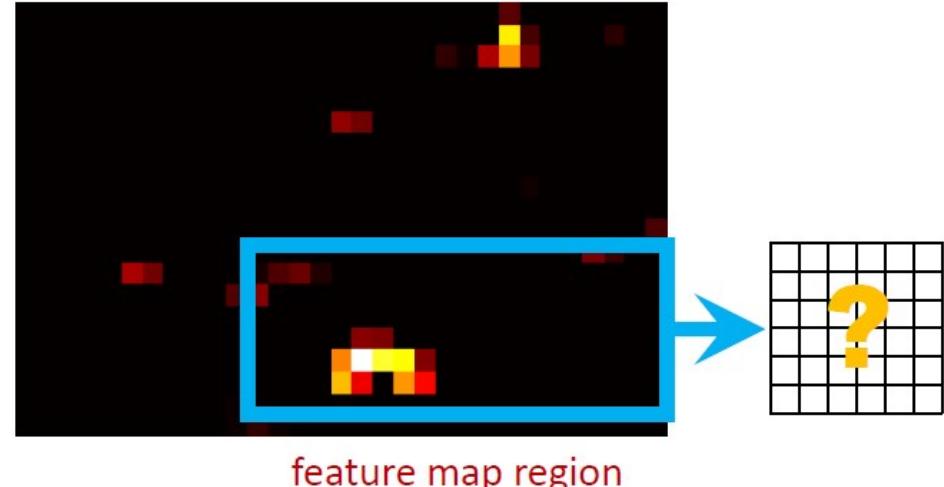
→ Performing CNN ONCE!
- R-CNN requires a fixed input size
  - FC layers need to have fixed size/length input by their definition

→ Using a spatial pyramid pooling layer to remove the fixed-size constraint of the network

# SPP-net



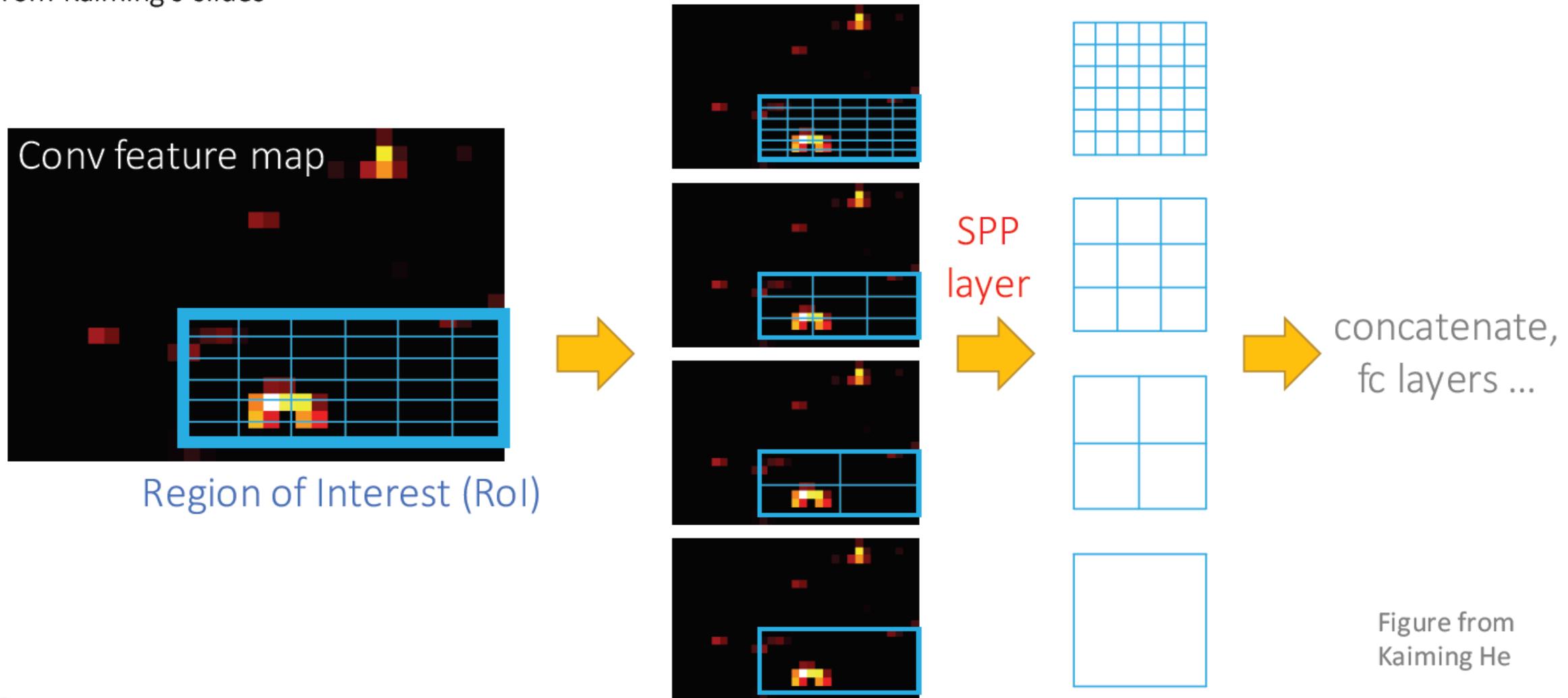
# Regions on Feature Maps



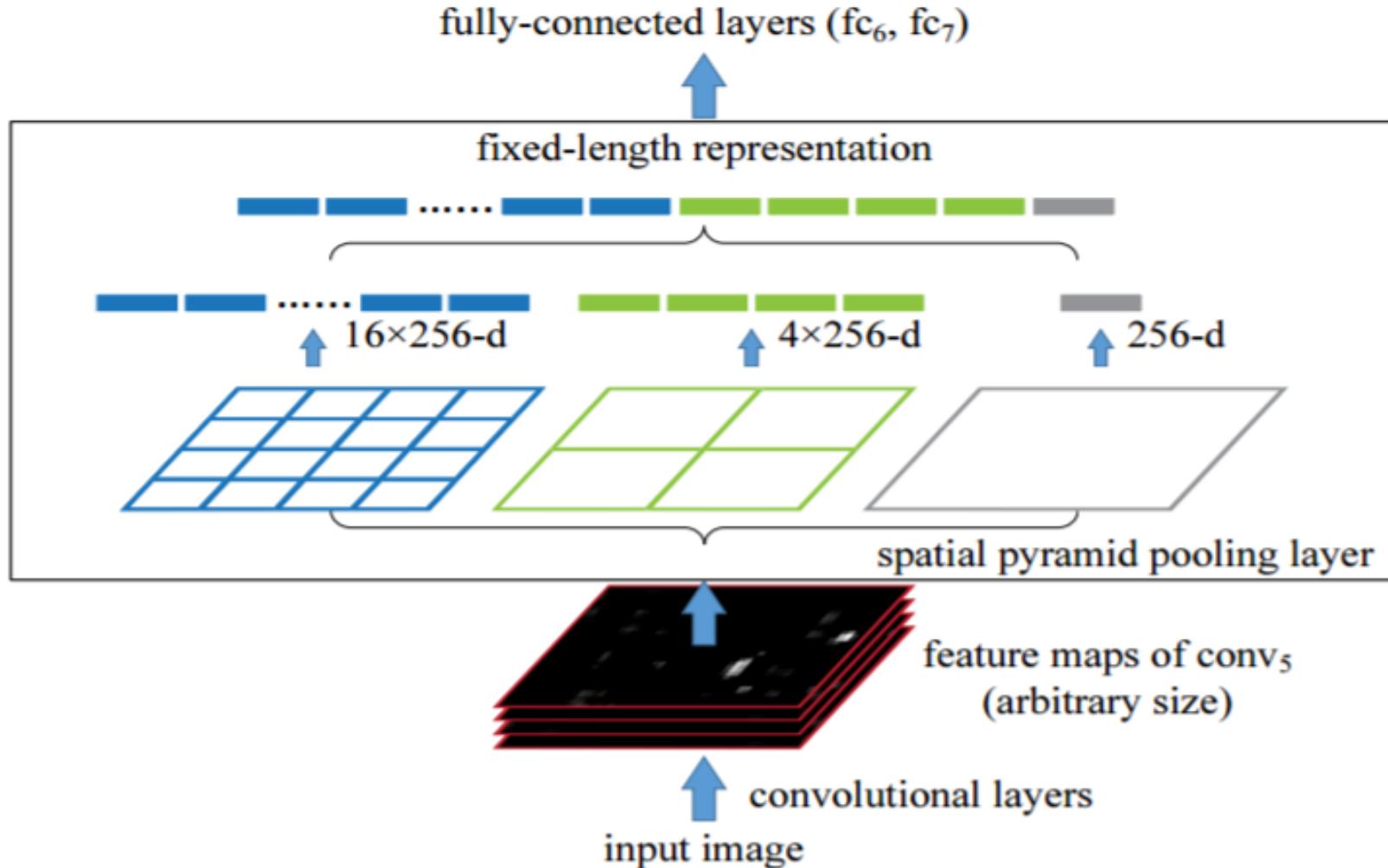
- Compute convolutional feature maps on the entire image **only once**
- Project an image region to a **feature map region**
- Extract a region-based feature from the feature map region...
- **Fixed-length** features are required by fully-connected layers or SVM
- But how to produce a fixed-length feature from a feature map region?

# Spatial Pyramid Pooling(SPP) Layer

From Kaiming's slides



# Spatial Pyramid Pooling(SPP) Layer



# Training of SPP-Net

Note that only classifier layers are fine-tuned, the conv layers are fixed

1. [offline]  $M \leftarrow$  Pre-train a ConvNet for ImageNet classification
2.  $F \leftarrow$  Cache SPP features to disk using  $M$
3.  $M' \leftarrow M.conv +$  Fine-tune 3-layer network fc6-fc7-fc8 on  $F$  (log loss)
4.  $F' \leftarrow$  Cache features on disk using  $M'$
5. Train post hoc linear SVMs on  $F'$  (hinge loss)
6. Train post hoc linear bounding-box regressors on  $F'$  (squared loss)

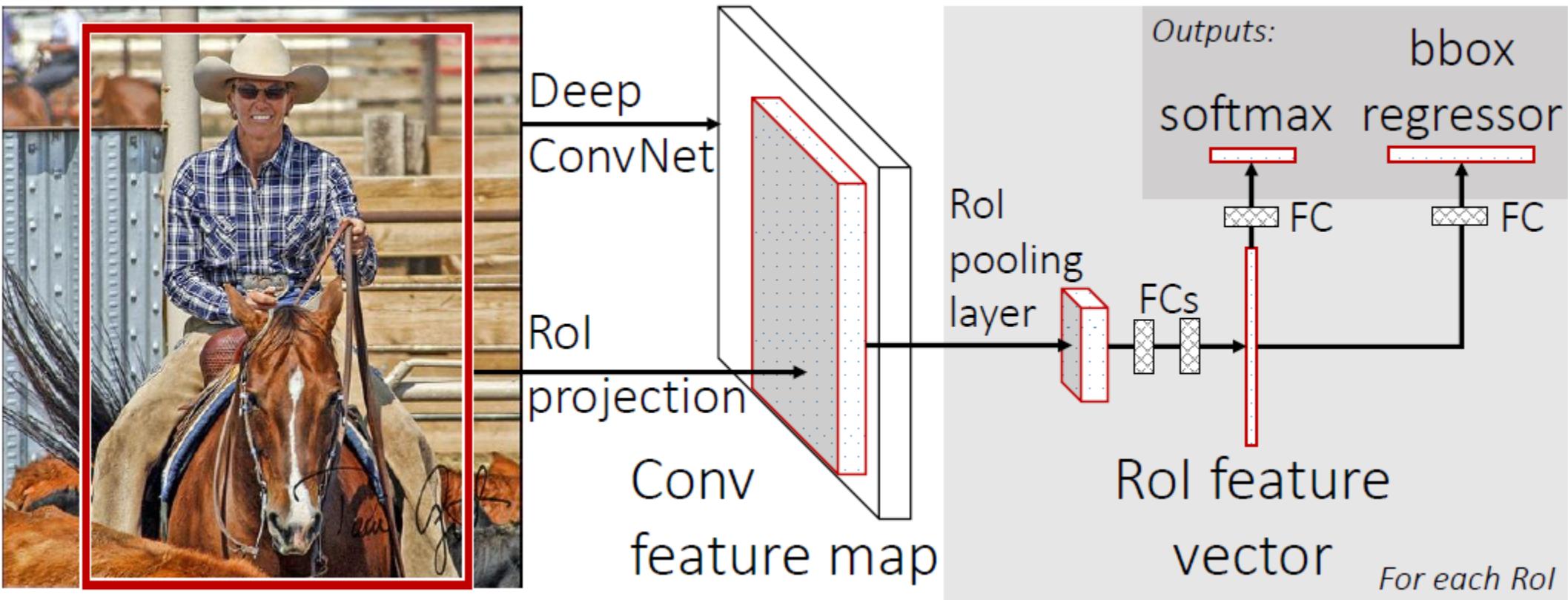
# Problems

- CNN features not updated in response to SVMs
- Complex multistage training pipeline (25.5 hours using K40 GPU)

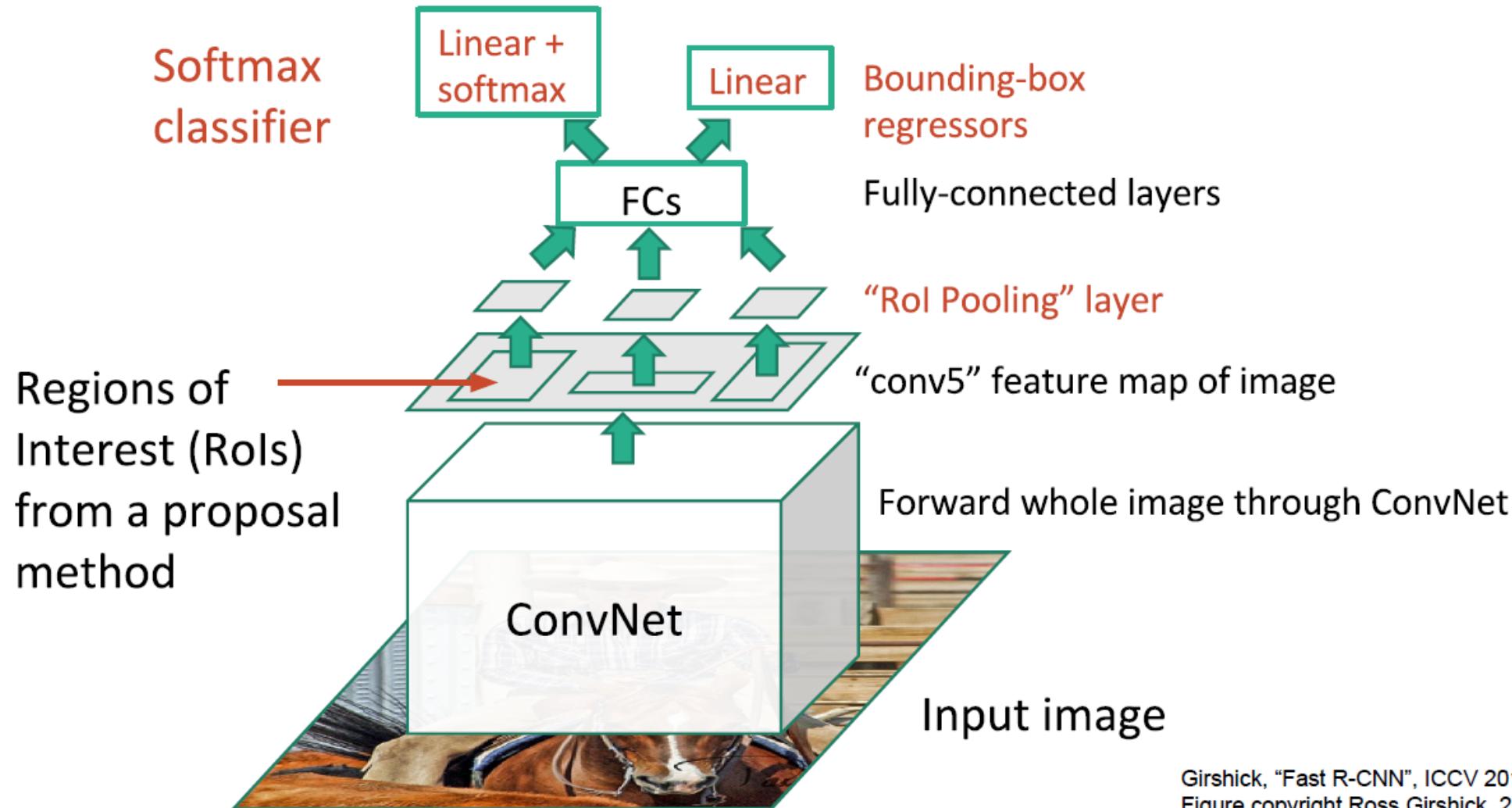
# Fast R-CNN

- Fix most of what's wrong with R-CNN and SPP-net
- Train the detector in a **single stage, end-to-end**
  - No caching features to disk
  - No post hoc training steps
- Train **all layers** of the network

# Fast R-CNN Architecture

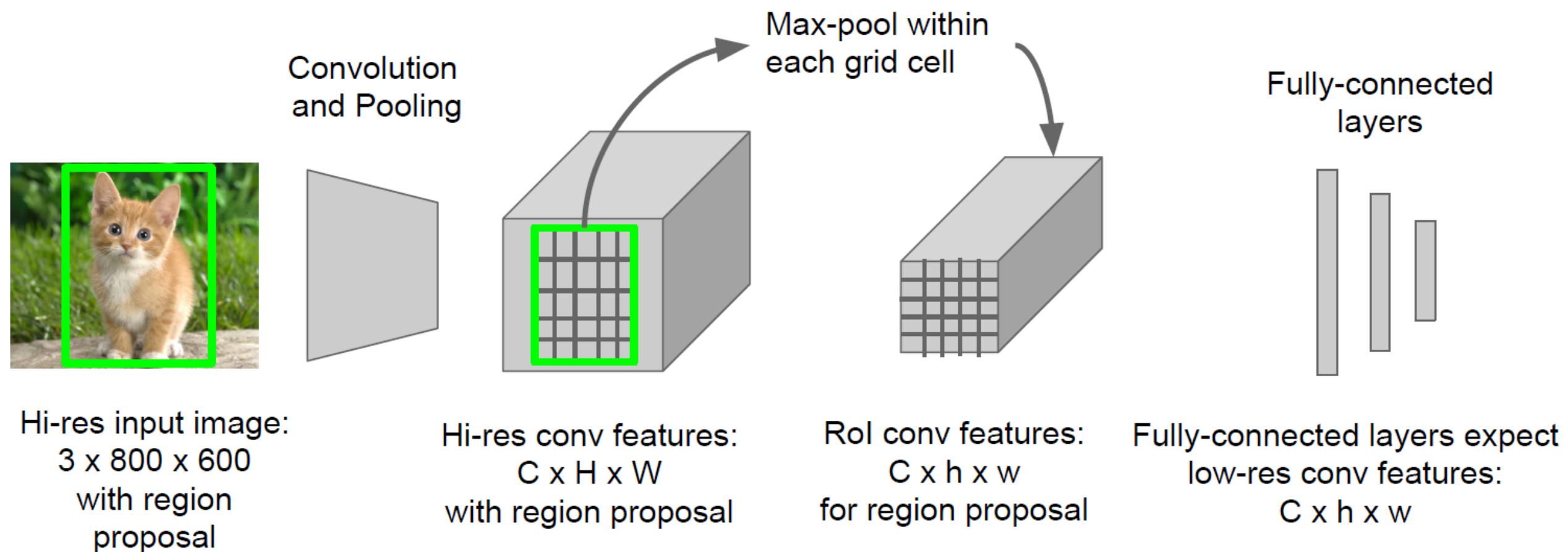


# Fast R-CNN



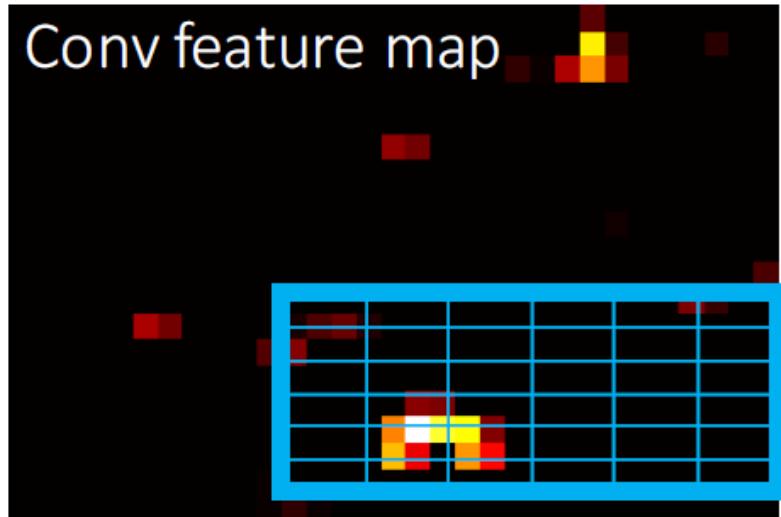
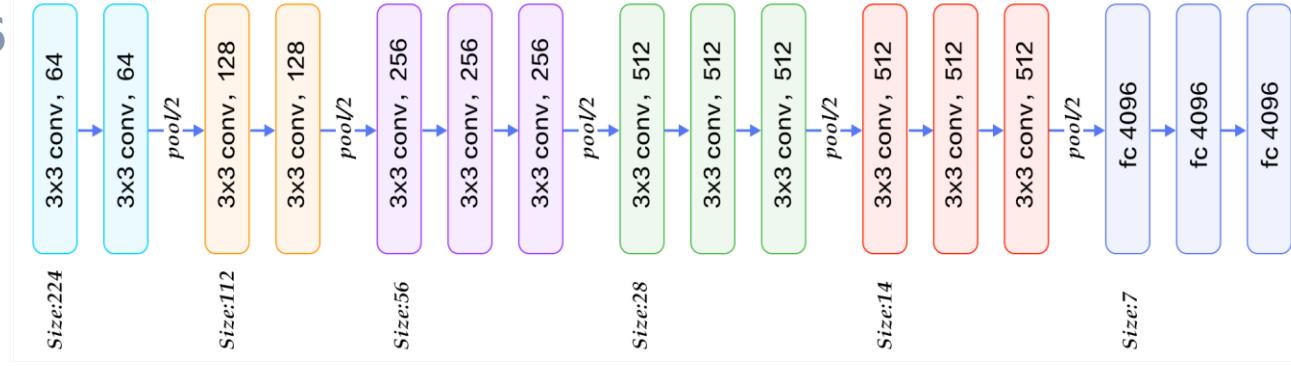
Girshick, "Fast R-CNN", ICCV 2015.  
Figure copyright Ross Girshick, 2015;

# RoI Pooling

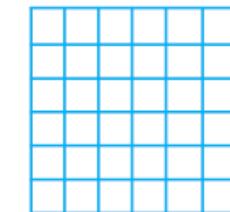


# Rol Pooling

VGG-16



RoI  
pooling  
layer



fc layers ...

Figure adapted  
from Kaiming He

Region of Interest (RoI)

Just a special case of the SPP layer with one pyramid level

RoI in Conv feature map : 21x14 → 3x2 max pooling with stride(3, 2) → output : 7x7  
RoI in Conv feature map : 35x42 → 5x6 max pooling with stride(5, 6) → output : 7x7

# Training & Testing

1. Takes an input and a set of bounding boxes
  2. Generate convolutional feature maps
  3. For each bbox, get a fixed-length feature vector from RoI pooling layer
  4. Outputs have two information
    - $K+1$  class labels
    - Bounding box locations
- Loss function

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$$

True box coordinates  
Predicted box coordinates  
True class scores  
Predicted class scores  
Log loss  
Smooth L1 loss

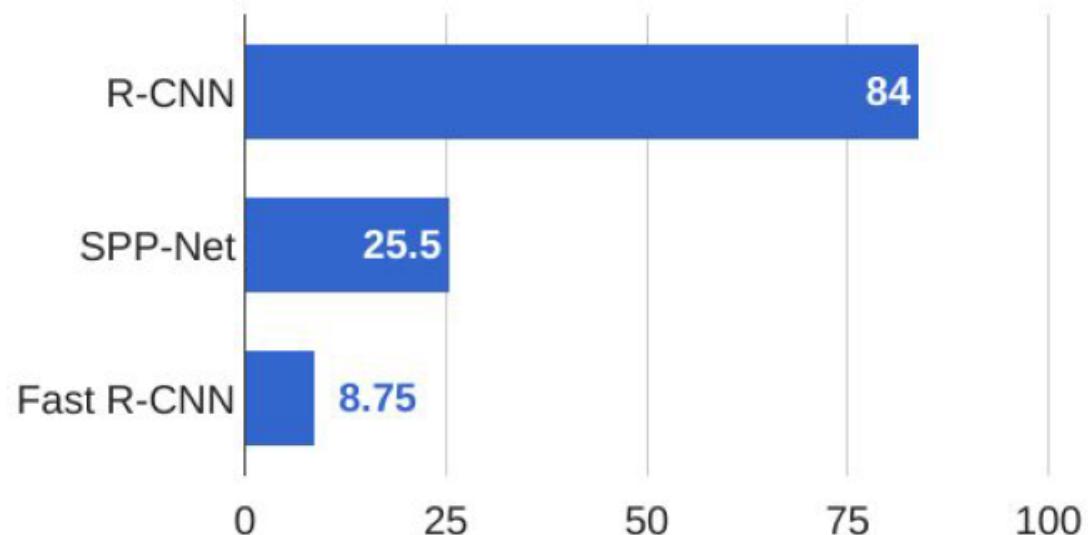
$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

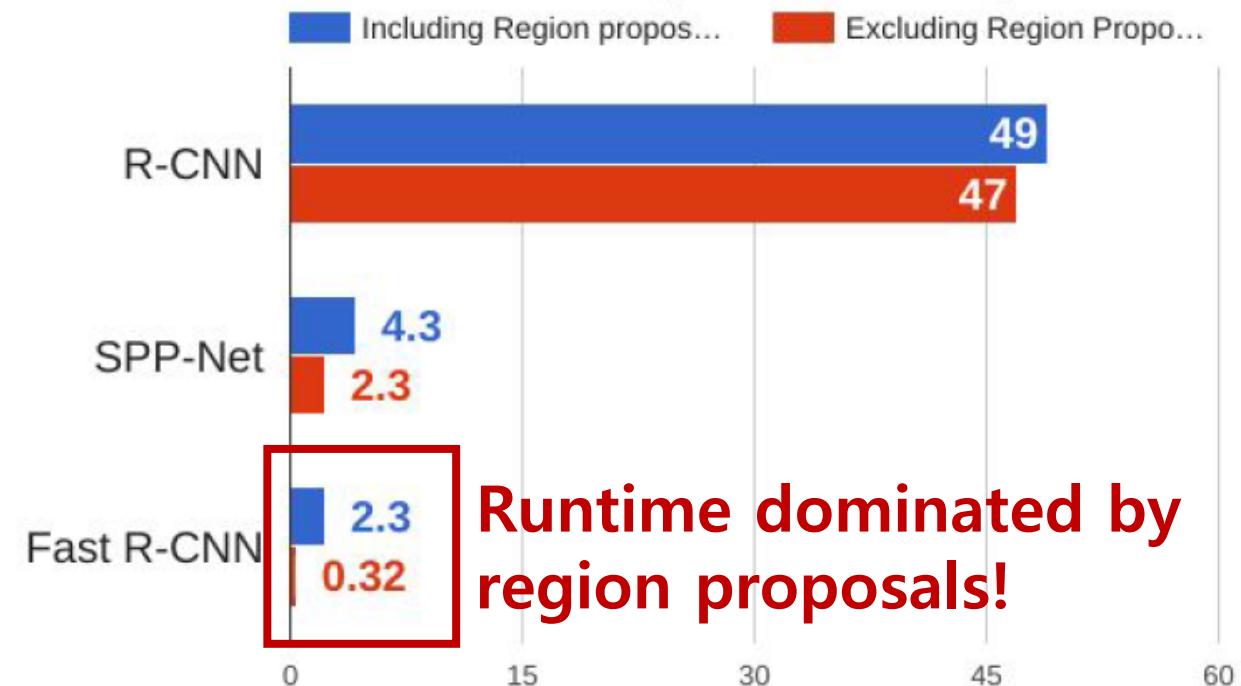
$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

# R-CNN vs SPP-net vs Fast R-CNN

**Training time (Hours)**



**Test time (seconds)**



# Experimental Findings

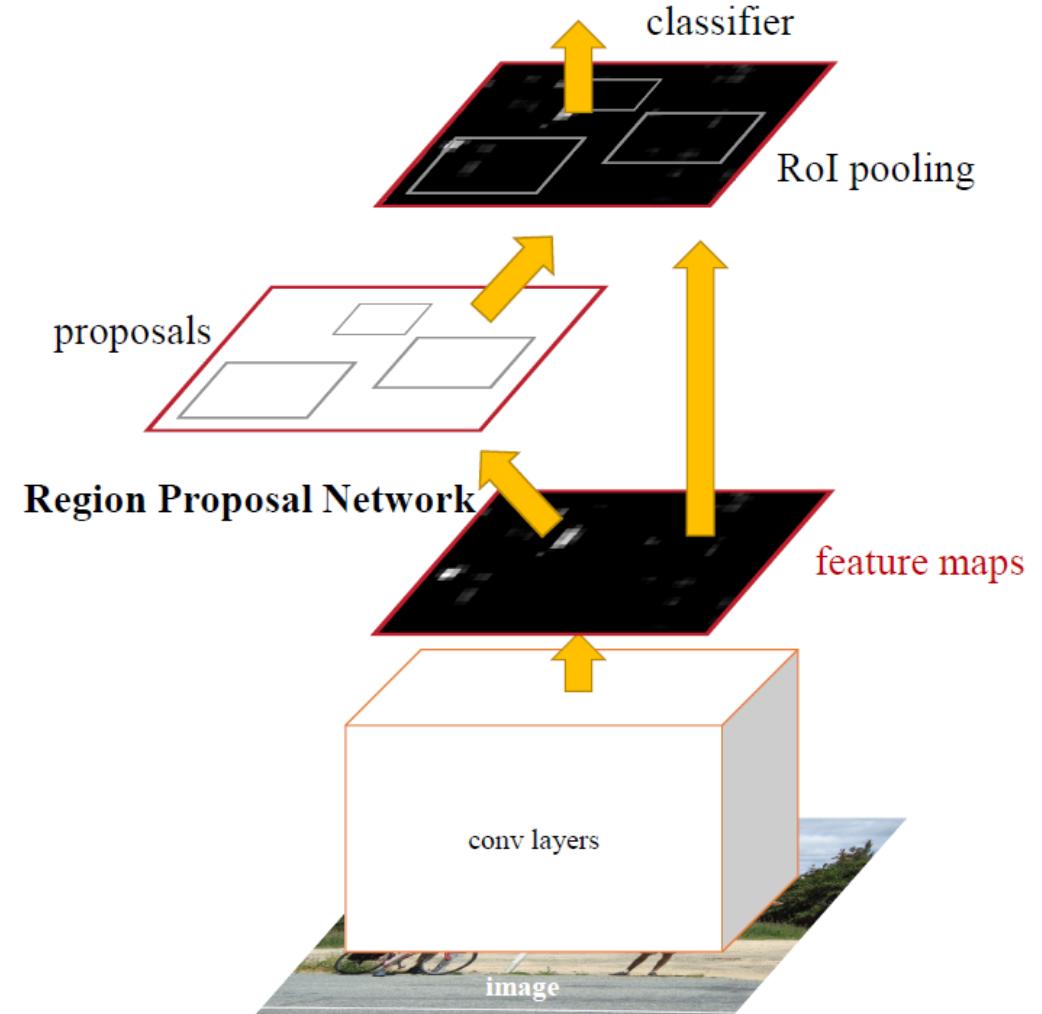
- End-to-end training is important for very deep networks
- Softmax is a fine replacement for SVMs
- Multi-task training is beneficial
- Single-scale testing is a good tradeoff
- Fast training and testing enables new experiments

# Problems of Fast R-CNN

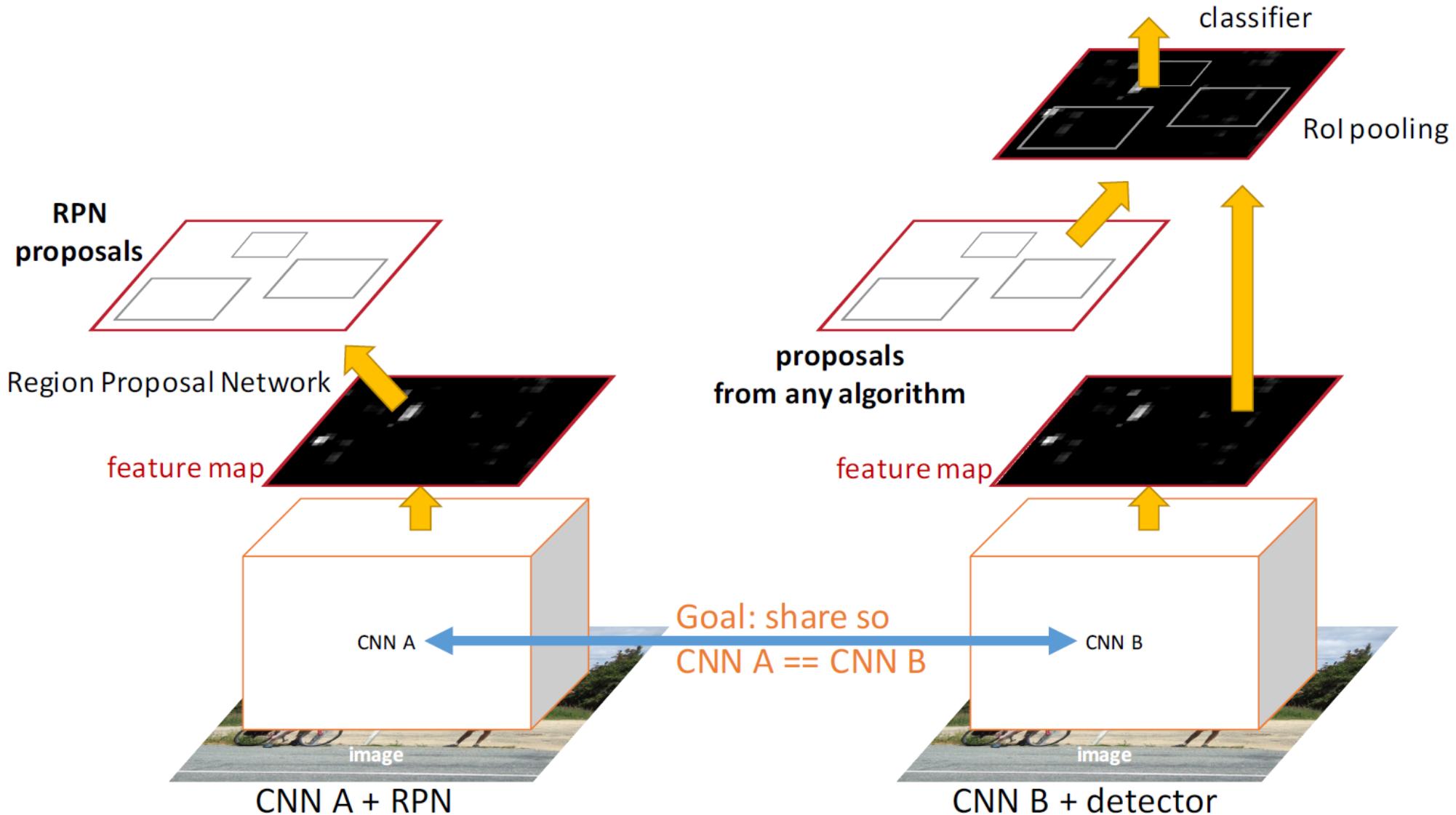
- Out-of-network region proposals are the test-time computational bottleneck
- Is it fast enough??

# Faster R-CNN(RPN + Fast R-CNN)

- Insert a Region Proposal Network (RPN) after the last convolutional layer → using GPU!
- 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

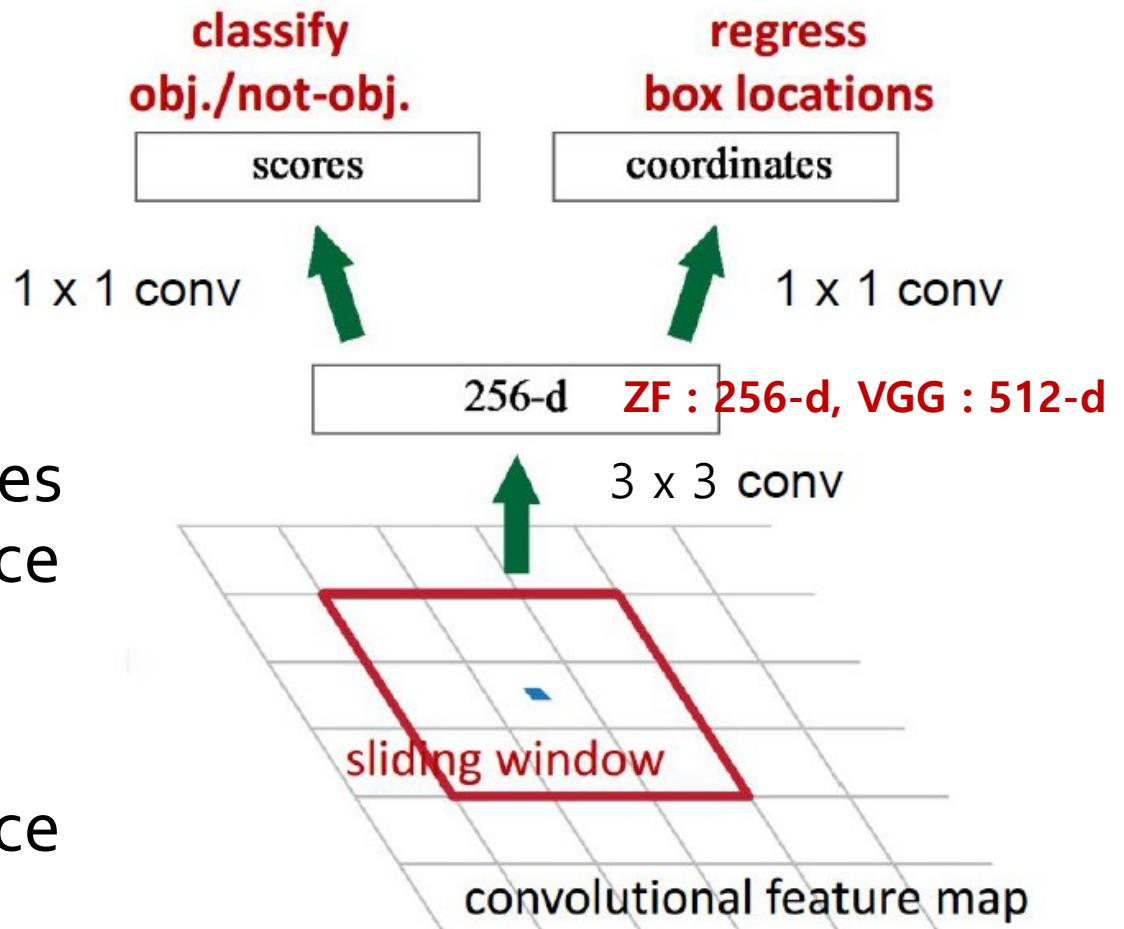


# Training Goal : Share Features



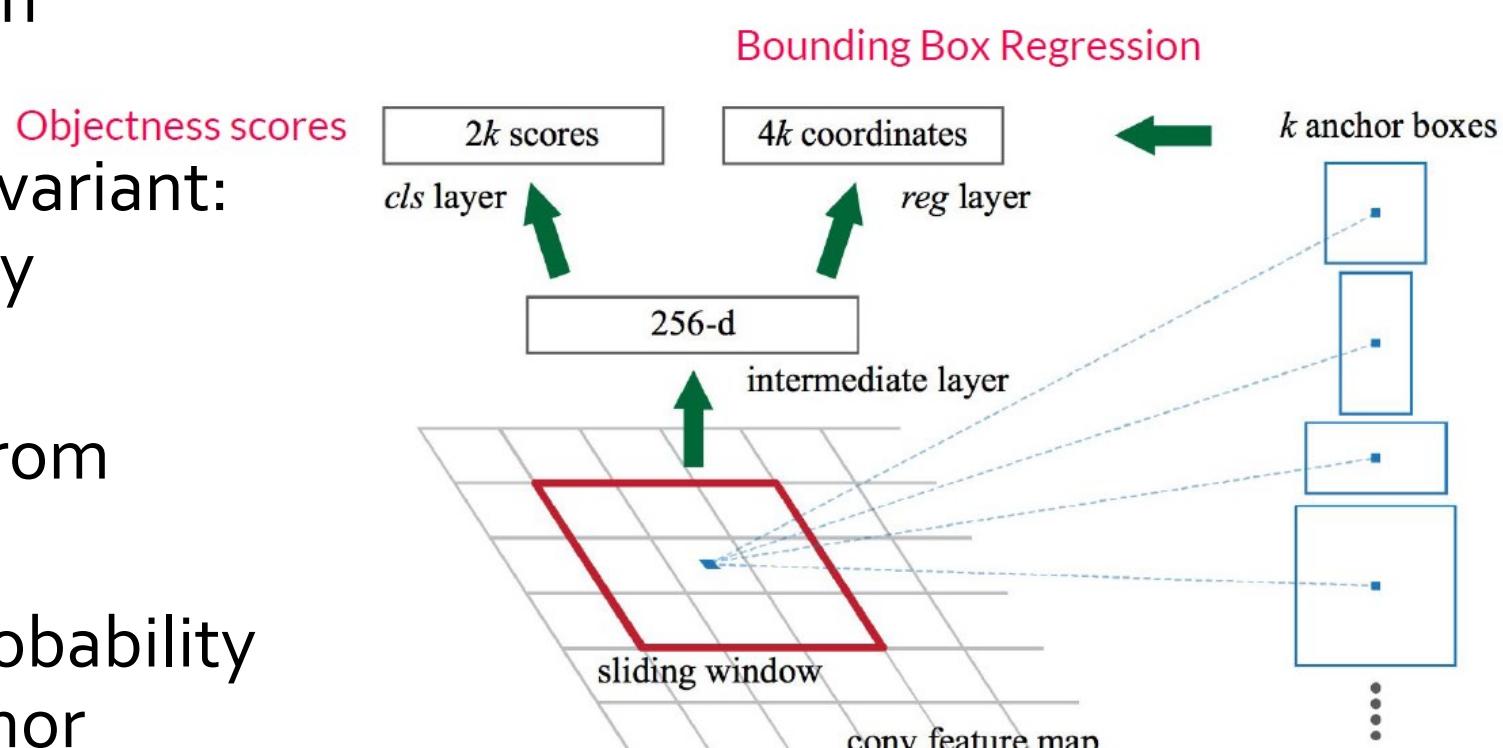
# RPN

- Slide a small window on the feature map
- Build a small network for
  - Classifying object or not-object
  - 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



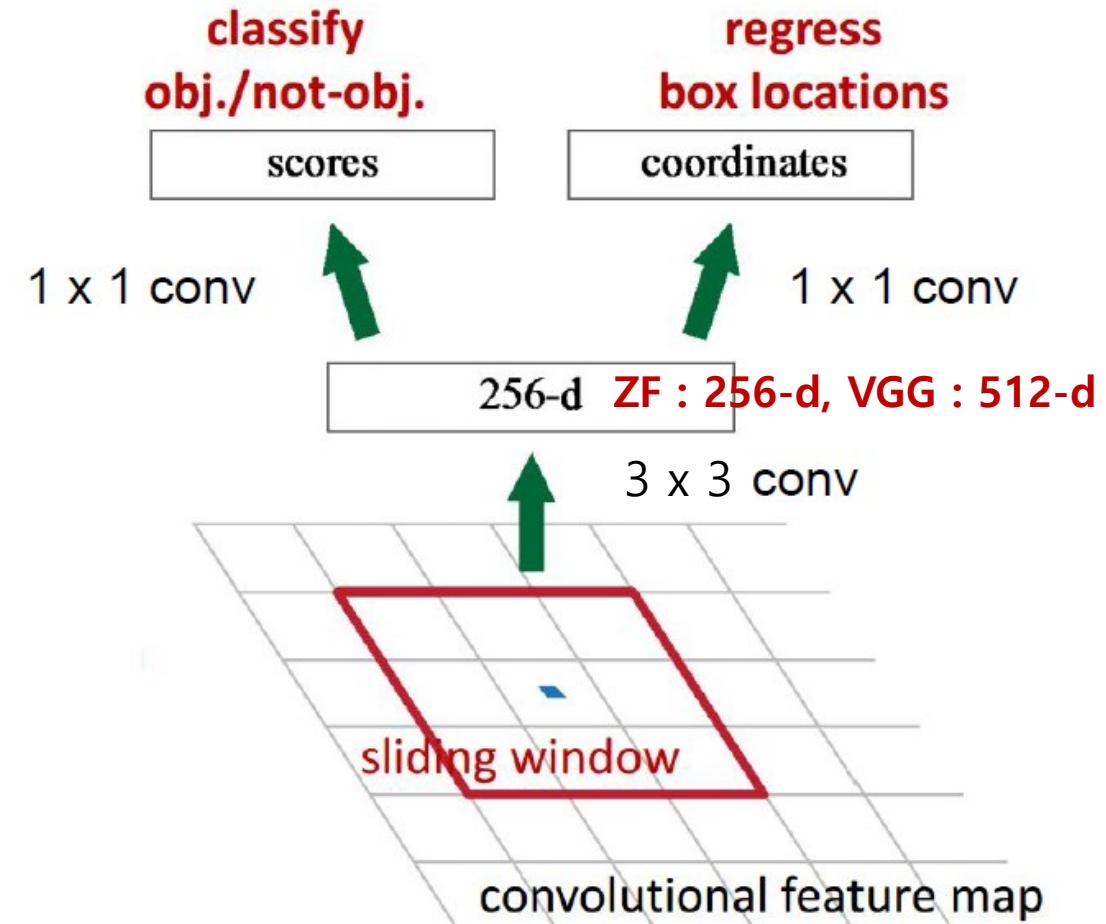
# RPN

- Use  $k$  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



# RPN(Fully Convolutional Network)

- Intermediate Layer – 256(or 512)  $3 \times 3$  filter, stride 1, padding 1
- Cls layer – 18( $9 \times 2$ )  $1 \times 1$  filter, stride 1, padding 0
- Reg layer – 36( $9 \times 4$ )  $1 \times 1$  filter, stride 1, padding 0



# Anchors as references

- Anchors: pre-defined reference boxes
- Multi-scale/size anchors:
  - Multiple anchors are used at each position:
    - 3 scale(128x128, 256x256, 512x512) and 3 aspect ratios(2:1, 1:1, 1:2) yield 9 anchors
  - Each anchor has its own prediction function
  - Single-scale features, multi-scale predictions

# Positive/Negative Samples

- An anchor is **labeled as positive** if
  - The anchor is the one with **highest IoU** overlap with a ground-truth box
  - The anchor has an IoU overlap with a ground-truth box **higher than 0.7**
- **Negative labels** are assigned to anchors with **IoU lower than 0.3** for all ground-truth boxes
- **50%/50%** ratio of positive/negative anchors in a minibatch

# RPN Loss Function

$i$  = anchor index in minibatch

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Annotations:

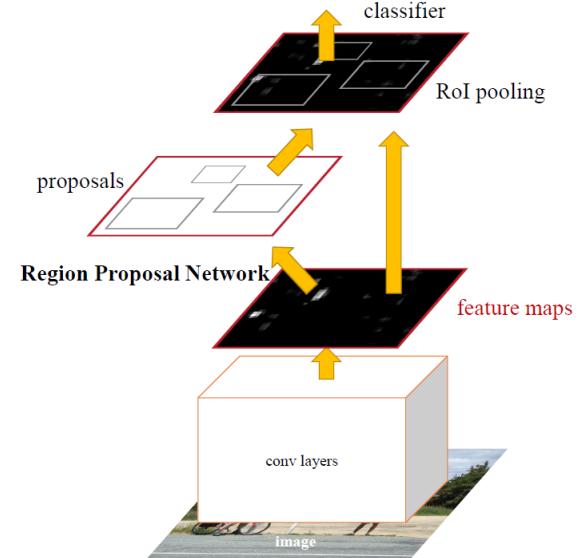
- Coordinates of the predicted bounding box for anchor  $i$  (blue double-headed arrow)
- Predicted probability of being an object for anchor  $i$  (blue double-headed arrow)
- Log loss (purple arrow pointing to the first term)
- Ground truth objectness label (red arrow pointing to  $p_i^*$ )
- Smooth L1 loss (purple arrow pointing to the second term)
- True box coordinates (red arrow pointing to  $t_i^*$ )
- $\lambda$  (red circle)

$N_{cls}$  = Number of anchors in minibatch (~ 256)

$N_{reg}$  = Number of anchor locations (~ 2400)

In practice  $\lambda = 10$ , so that both terms are roughly equally balanced

# 4-Step Alternating Training



# Let M0 be an ImageNet pre-trained network

1. train\_rpn(**M0**) → M1 # Train an RPN initialized from M0, get M1
2. generate\_proposals(M1) → P1 # Generate training proposals P1 using RPN M1
3. train\_fast\_rcnn(**M0**, P1) → M2 # Train Fast R-CNN M2 on P1 initialized from M0
4. train\_rpn\_frozen\_conv(**M2**) → M3 # Train RPN M3 from M2 *without* changing conv layers
5. generate\_proposals(M3) → P2
6. train\_fast\_rcnn\_frozen\_conv(M3, P2) → M4 # Conv layers are shared with RPN M3
7. return add\_rpn\_layers(M4, M3.RPN) # Add M3's RPN layers to Fast R-CNN M4

# Results

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	<b>0.2 seconds</b>
(Speedup)	1x	25x	<b>250x</b>
mAP (VOC 2007)	66.0	<b>66.9</b>	<b>69.9</b>

Table 5: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. “Region-wise” includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	<b>10</b>	47	<b>198</b>	<b>5 fps</b>
ZF	RPN + Fast R-CNN	31	<b>3</b>	25	<b>59</b>	<b>17 fps</b>

# Experiments

Table 1: the learned average proposal size for each anchor using the ZF net (numbers for  $s = 600$ ).

anchor	$128^2, 2:1$	$128^2, 1:1$	$128^2, 1:2$	$256^2, 2:1$	$256^2, 1:1$	$256^2, 1:2$	$512^2, 2:1$	$512^2, 1:1$	$512^2, 1:2$
proposal	$188 \times 111$	$113 \times 114$	$70 \times 92$	$416 \times 229$	$261 \times 284$	$174 \times 332$	$768 \times 437$	$499 \times 501$	$355 \times 715$

Table 8: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different settings of anchors**. The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using 3 scales and 3 aspect ratios (69.9%) is the same as that in Table 3.

settings	anchor scales	aspect ratios	mAP (%)
1 scale, 1 ratio	$128^2$	1:1	65.8
	$256^2$	1:1	66.7
1 scale, 3 ratios	$128^2$	{2:1, 1:1, 1:2}	68.8
	$256^2$	{2:1, 1:1, 1:2}	67.9
3 scales, 1 ratio	$\{128^2, 256^2, 512^2\}$	1:1	69.8
3 scales, 3 ratios	$\{128^2, 256^2, 512^2\}$	{2:1, 1:1, 1:2}	69.9

Table 9: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different values of  $\lambda$**  in Equation (1). The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using  $\lambda = 10$  (69.9%) is the same as that in Table 3.

$\lambda$	0.1	1	10	100
mAP (%)	67.2	68.9	69.9	69.1

# Experiments

Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

train-time region proposals		test-time region proposals		mAP (%)
method	# boxes	method	# proposals	
SS	2000	SS	2000	58.7
EB	2000	EB	2000	58.6
RPN+ZF, shared	2000	RPN+ZF, shared	300	<b>59.9</b>
<i>ablation experiments follow below</i>				
RPN+ZF, unshared	2000	RPN+ZF, unshared	300	58.7
SS	2000	RPN+ZF	100	55.1
SS	2000	RPN+ZF	300	56.8
SS	2000	RPN+ZF	1000	56.3
SS	2000	RPN+ZF (no NMS)	6000	<b>55.2</b>
SS	2000	RPN+ZF (no <i>cls</i> )	100	44.6
SS	2000	RPN+ZF (no <i>cls</i> )	300	51.4
SS	2000	RPN+ZF (no <i>cls</i> )	1000	<b>55.8</b>
SS	2000	RPN+ZF (no <i>reg</i> )	300	<b>52.1</b>
SS	2000	RPN+ZF (no <i>reg</i> )	1000	51.3
SS	2000	RPN+VGG	300	59.2

# Experiments

Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: “07”: VOC 2007 trainval, “07+12”: union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000.  $\dagger$ : this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9 $\dagger$
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	<b>73.2</b>
RPN+VGG, shared	300	COCO+07+12	<b>78.8</b>

Table 4: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: “07”: VOC 2007 trainval, “07++12”: union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000.  $\dagger$ : <http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html>.  $\ddagger$ : <http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html>.  $\S$ : <http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html>.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared $\dagger$	300	12	67.0
RPN+VGG, shared $\ddagger$	300	07++12	<b>70.4</b>
RPN+VGG, shared $\S$	300	COCO+07++12	<b>75.9</b>

# Is It Enough?

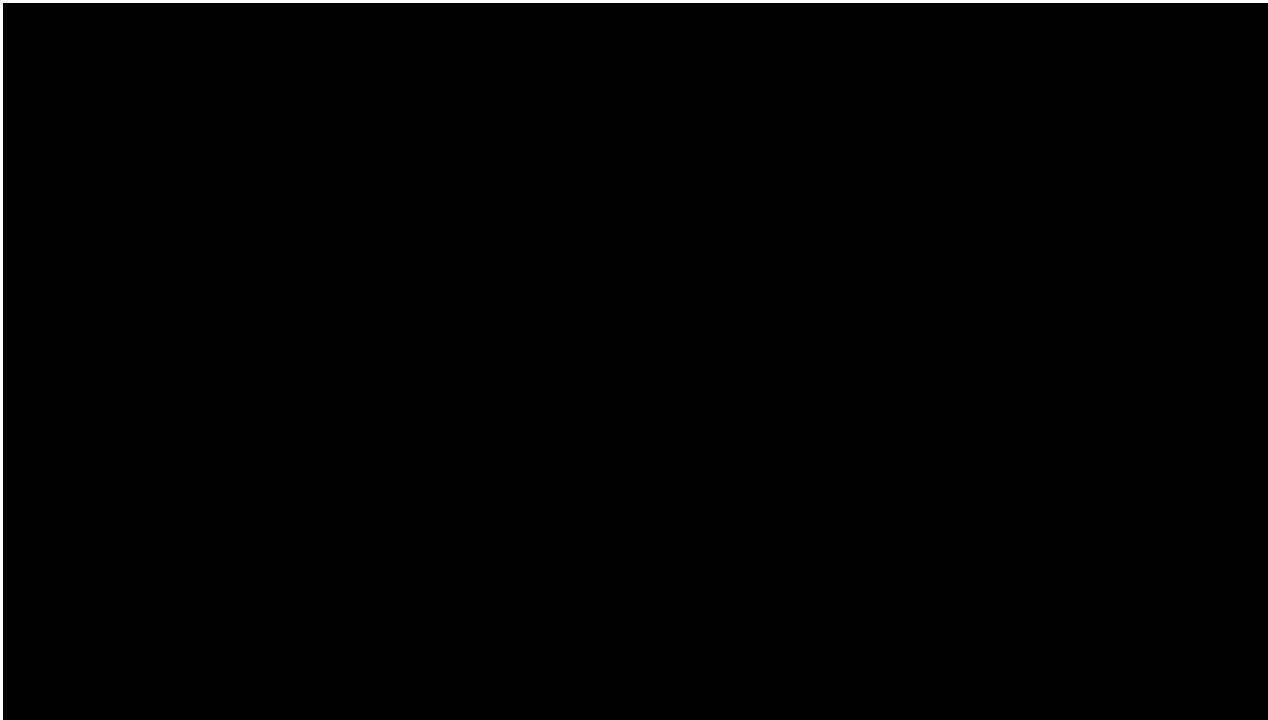
- RoI Pooling has some quantization operations
- These quantizations introduce misalignments between the RoI and the extracted features
- While this may not impact classification, it can make a negative effect on predicting bbox

**YOLO**

**YOU  
ONLY  
LIVE  
ONCE**

# YOLO

- You Only Look Once
- Quite similar with Faster R-CNN and very FAST!

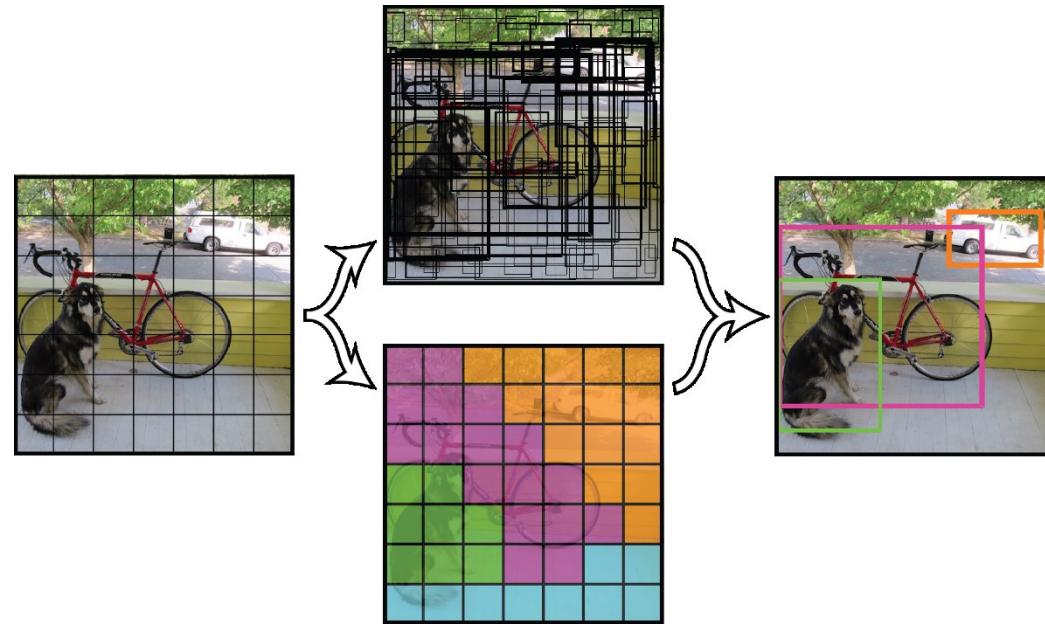


# Concepts

- Detection as Single Regression Problem
- Developed as Single Convolutional Network
- Reason Globally on the Entire Image
- Learns Generalizable Representations

# Unified Detection

- Given an image, divide it into an  $S \times S$  grid
  - If the center of an object falls into the grid cell, that grid cell is responsible
- Each cell predicts  $B$  bounding boxes
  - Five predictions:  $x, y, w, h, \text{confidence}$
- Each cell predicts  $C$  class probabilities
  - Cell →  $C$  class probabilities,  $B \times 5$  bounding box informations
- One cell → One class probability
  - Low false positive!



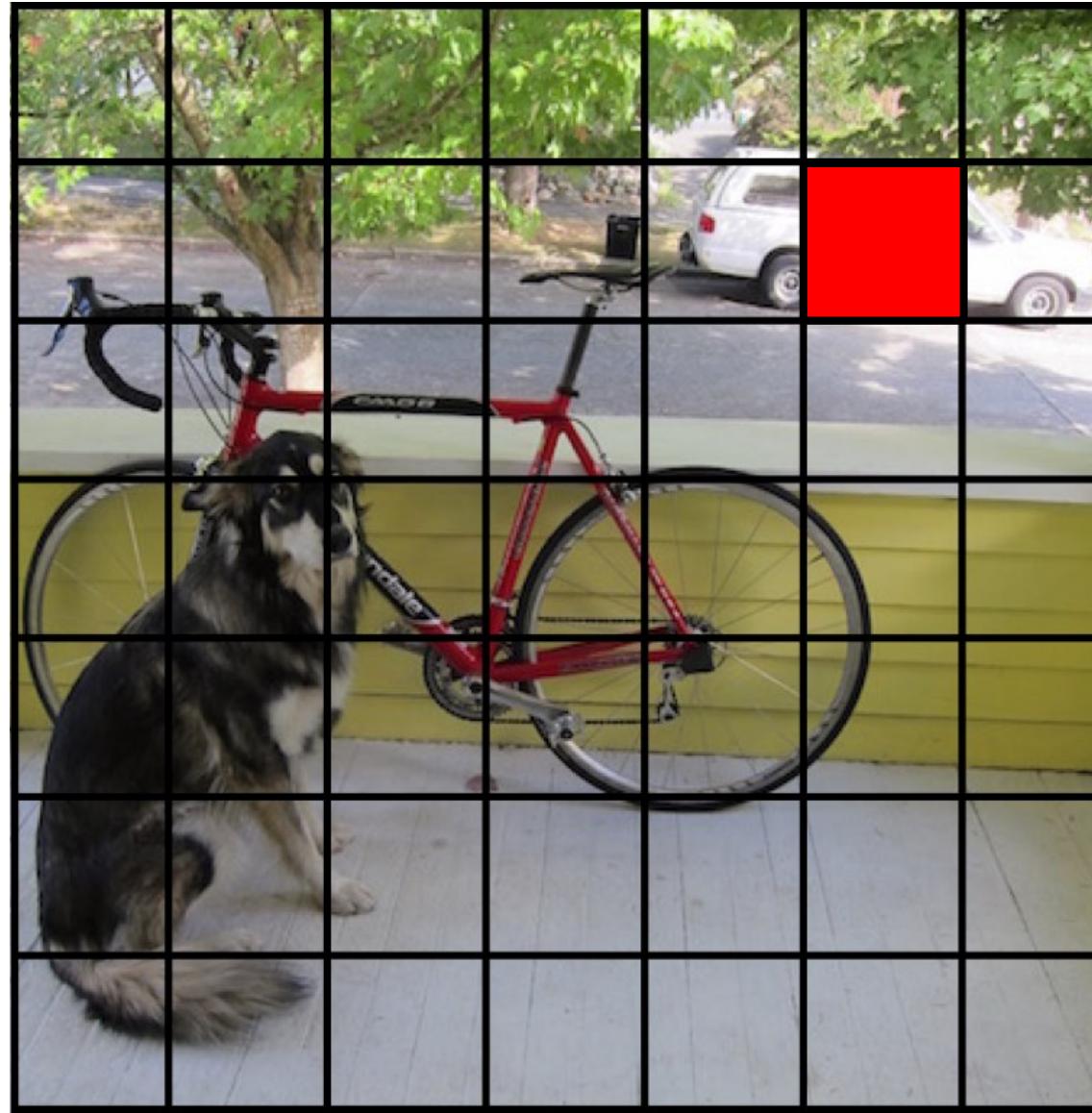
# Detection Flow



# Split the image into a grid(7x7)



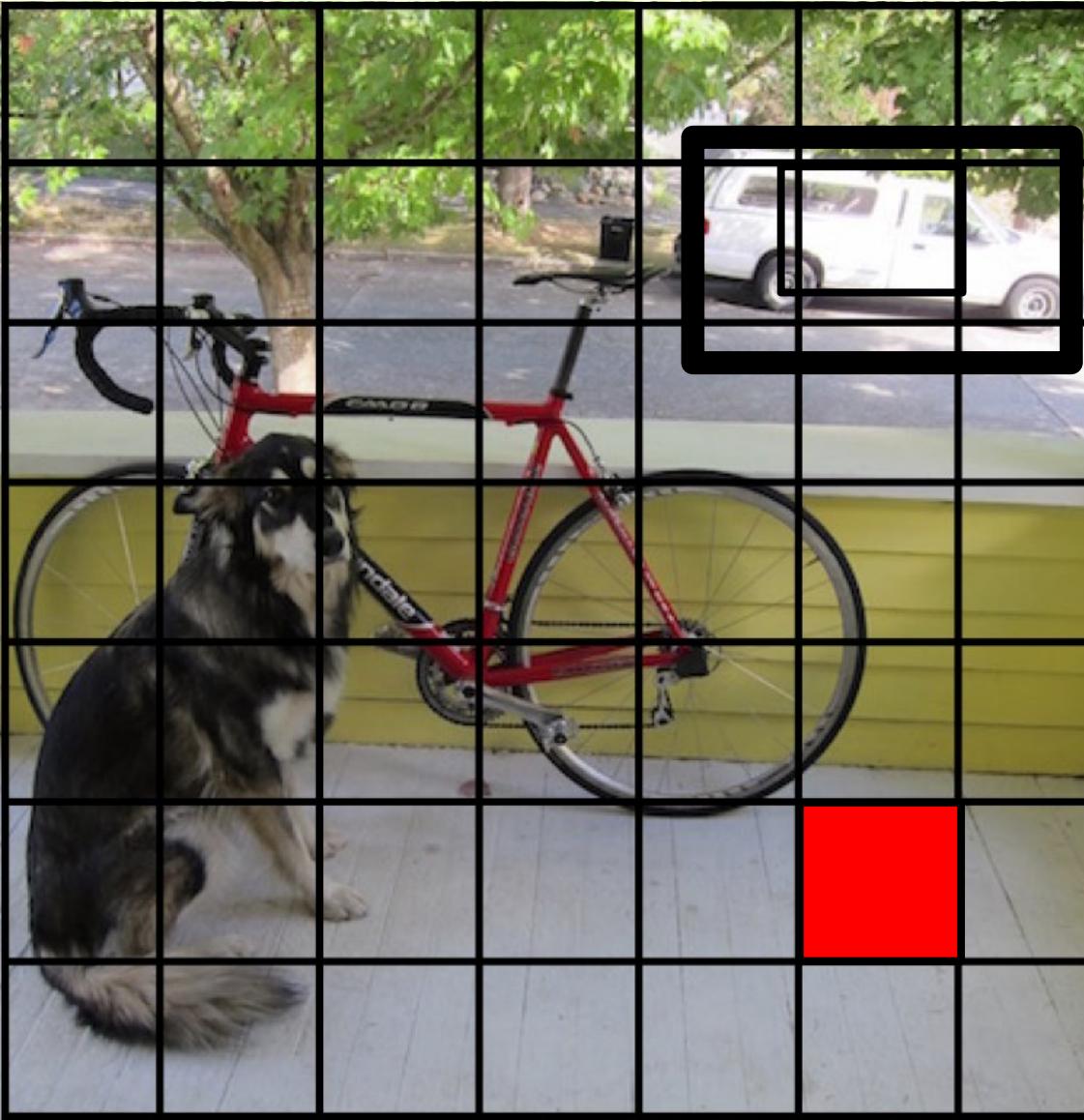
# Each cell predicts boxes and confidences: $P(\text{Object})$



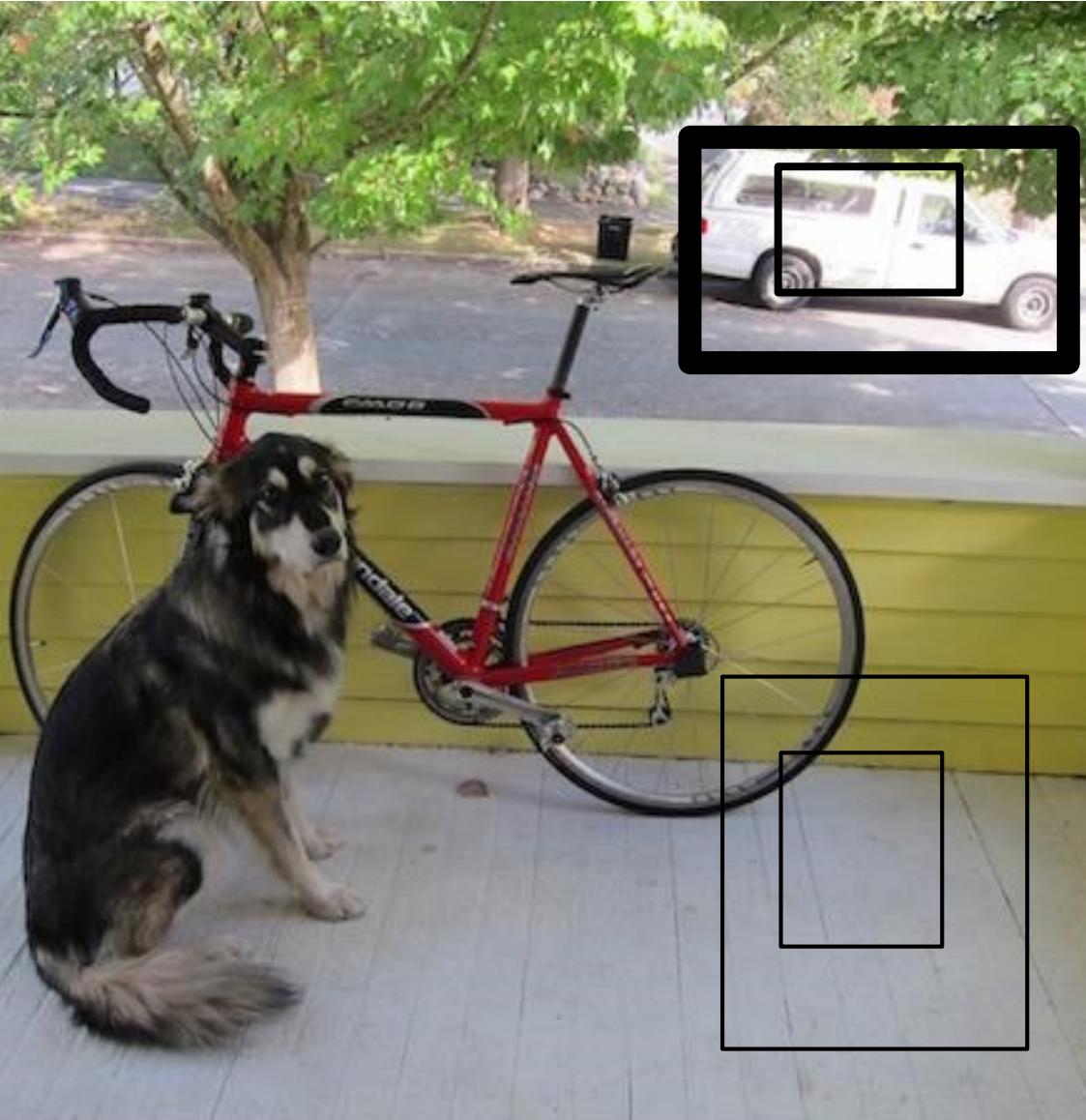
Each cell predicts boxes and confidences:  
 $P(\text{Object})$



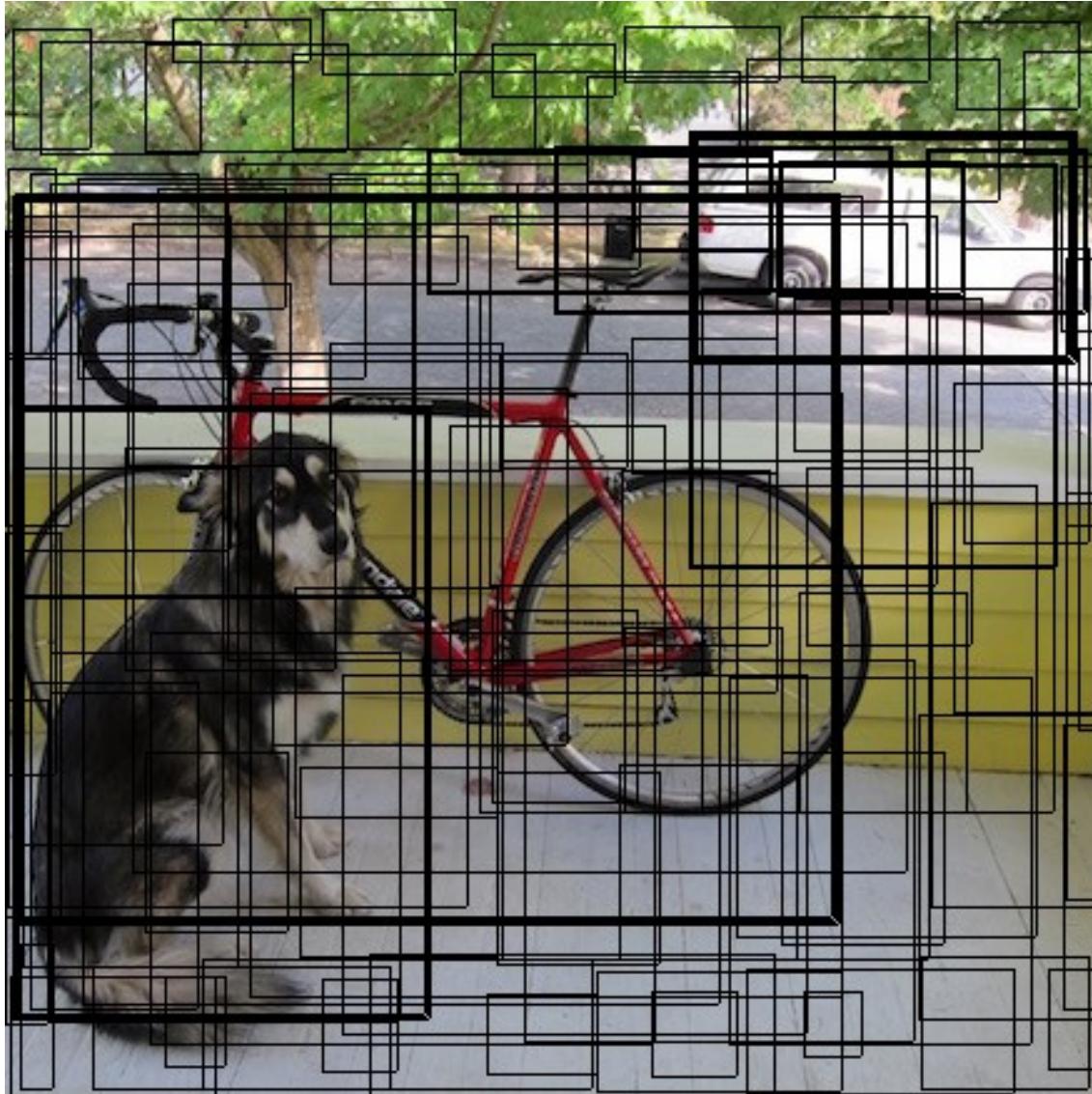
Each cell predicts boxes and confidences:  
 $P(\text{Object})$



Each cell predicts boxes and confidences:  
 $P(\text{Object})$



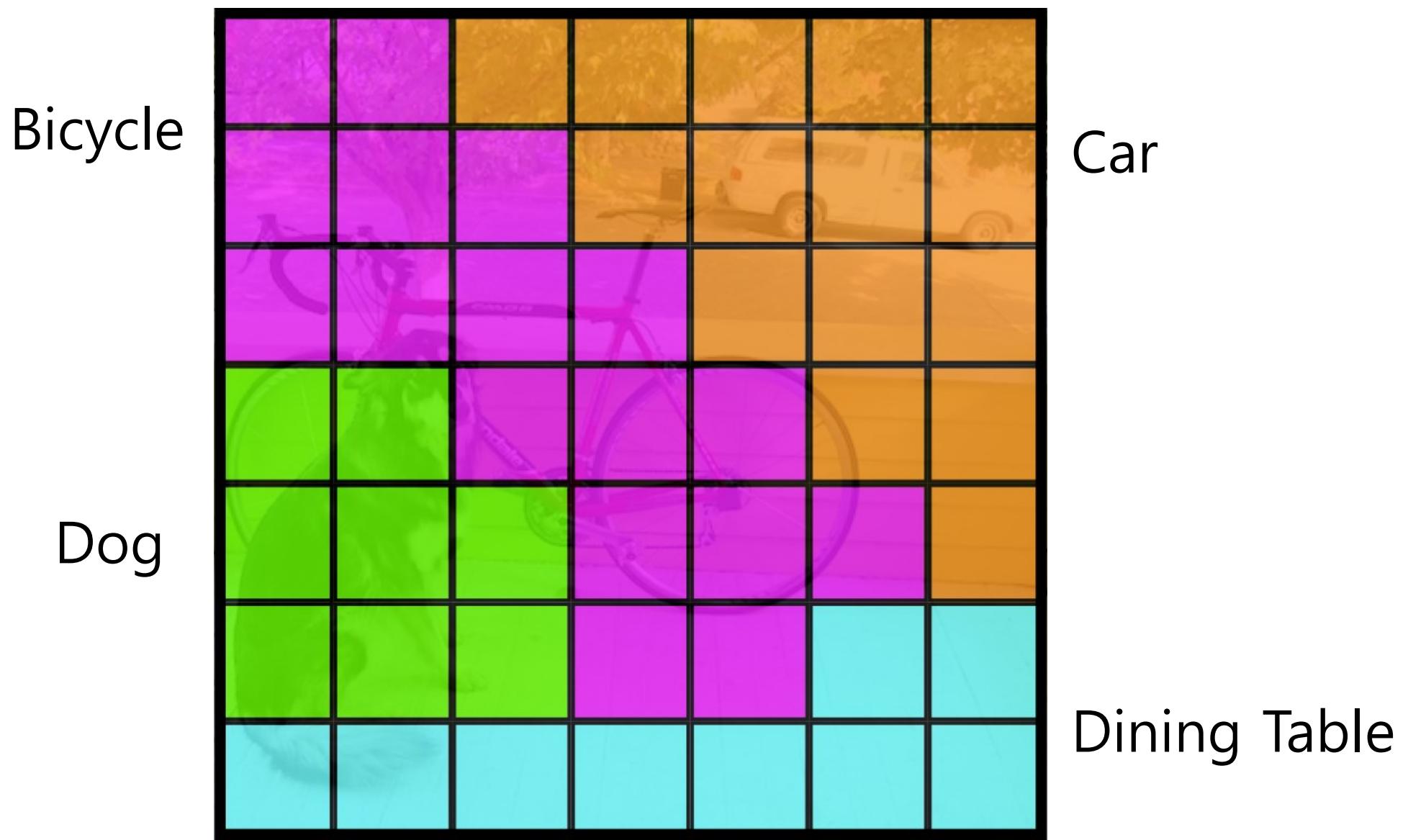
Each cell predicts boxes and confidences:  
 $P(\text{Object})$



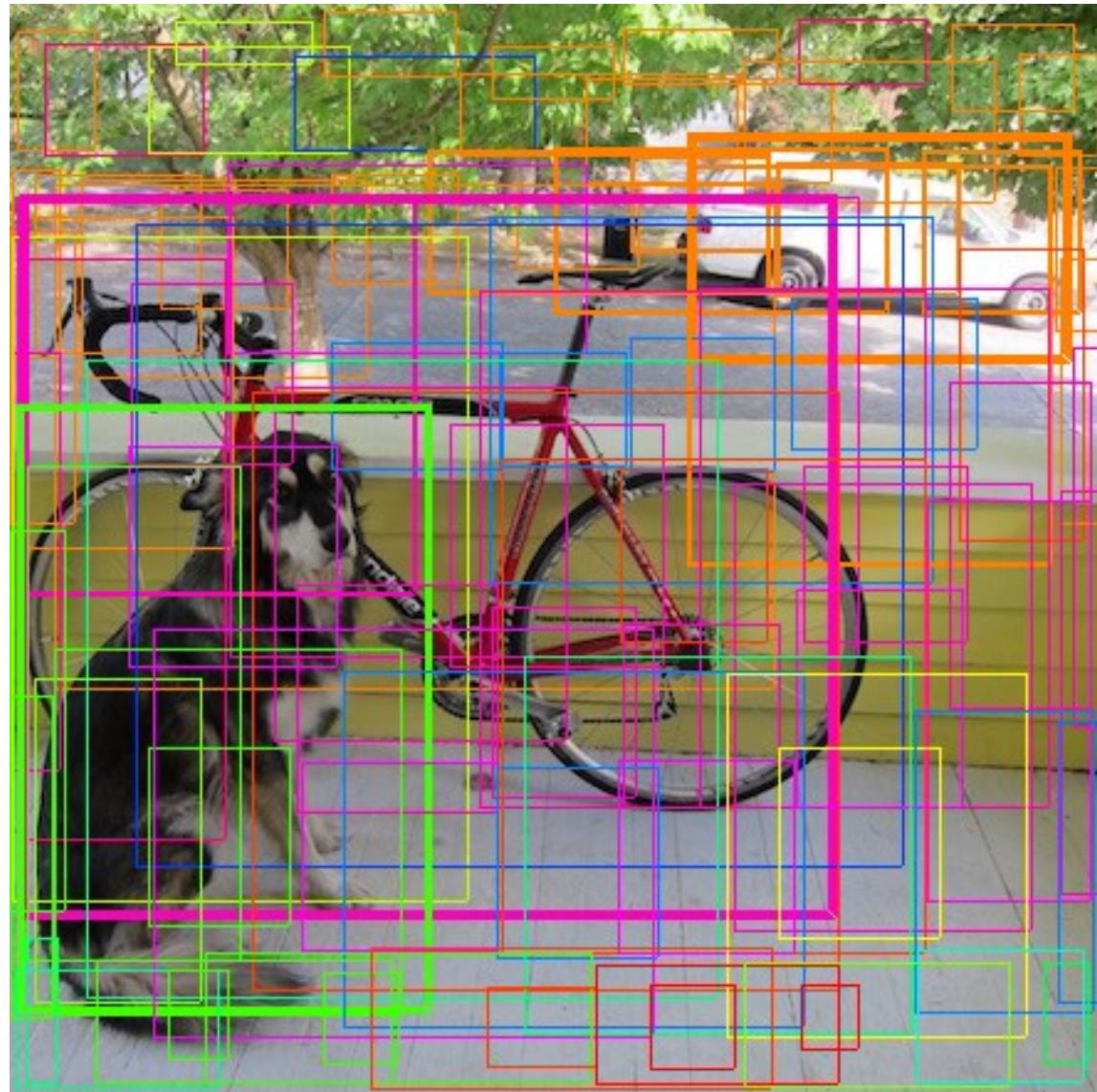
# Each cell also predicts a class probability



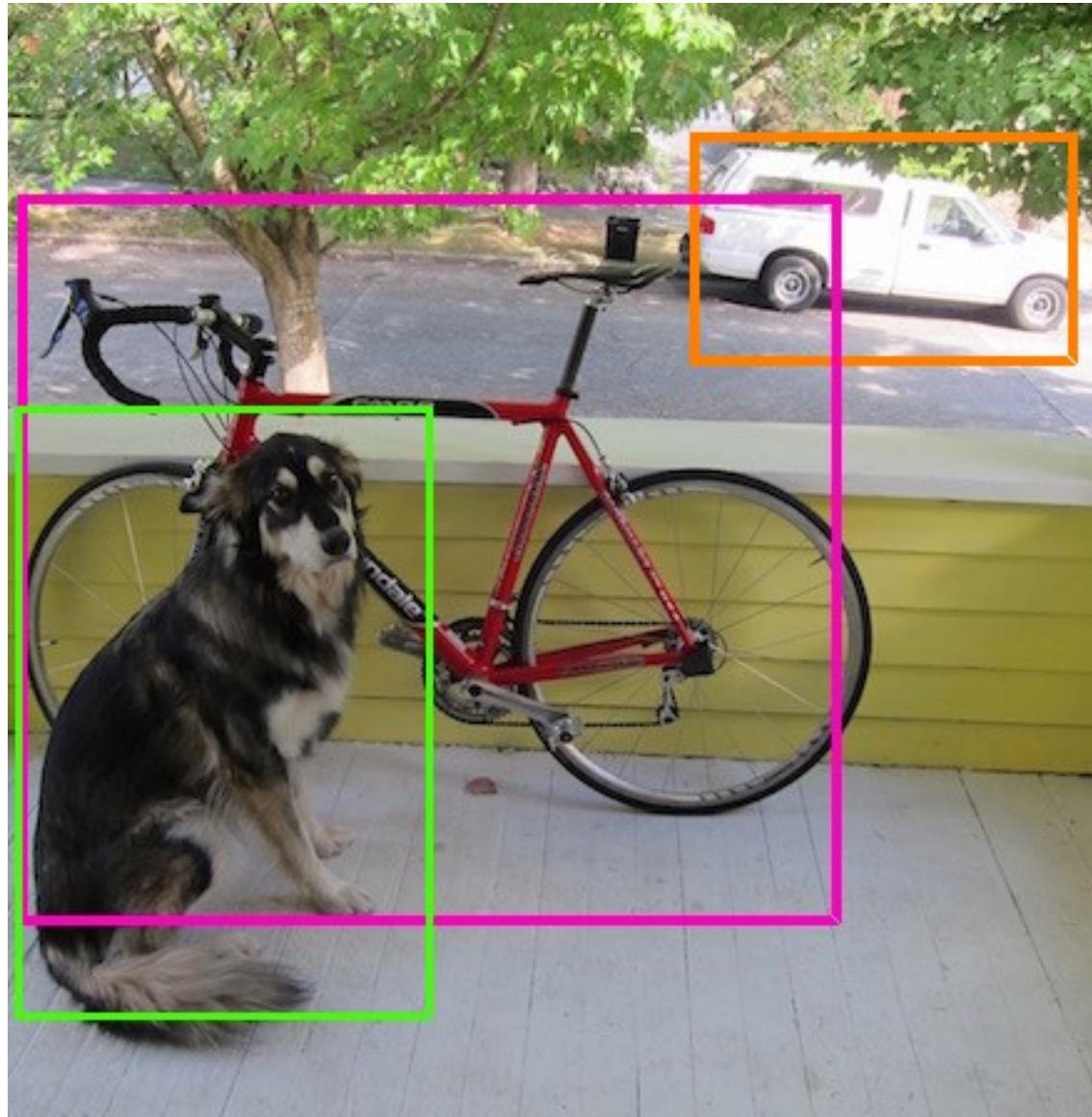
Each cell also predicts a class probability



Then we combine the box and class predictions.



# Finally we do NMS and threshold detections



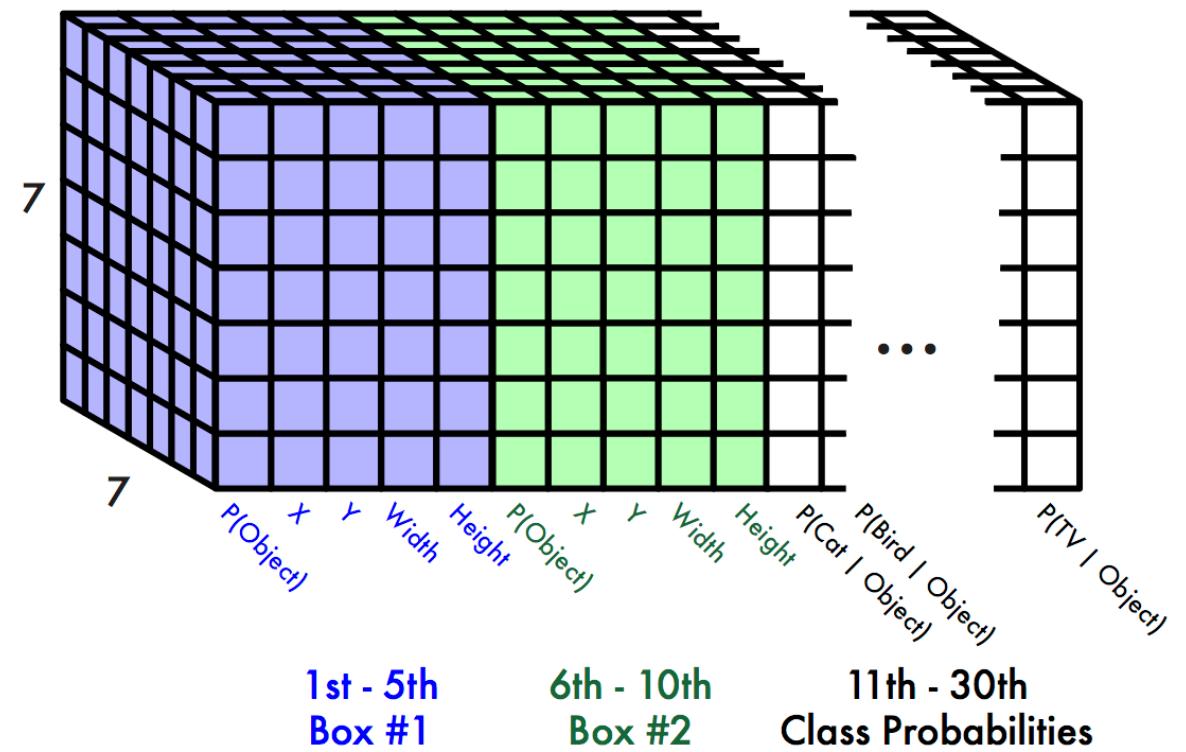
# Outputs

Each cell predicts:

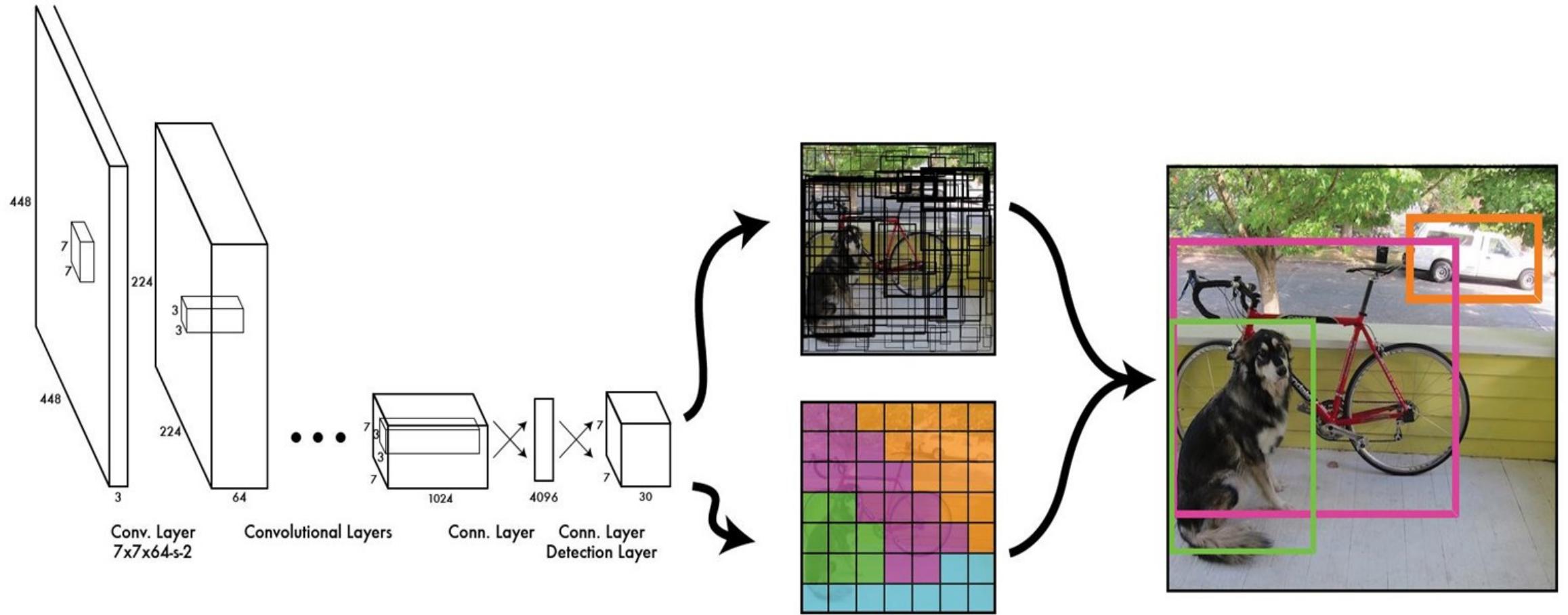
- For each bounding box:
  - 4 coordinates ( $x, y, w, h$ )
  - 1 confidence value
- Some number of class probabilities

For Pascal VOC:

- $7 \times 7$  grid
- 2 bounding boxes / cell
- 20 classes



# YOLO



# Loss Function

- In training, one predictor which has the highest IoU with the ground truth is responsible

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

$\mathbb{1}_{ij}^{\text{obj}}$

The **jth bbox predictor** in **cell i** is “responsible” for that prediction

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

$\mathbb{1}_i^{\text{obj}}$

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).

# Problems

- Each grid cell can predict only  $B(=2)$  bounding boxes and one class probability
  - Not good for small objects that flock together
- Uses relatively coarse features
  - Locations of bboxes are inaccurate
- Loss function treats errors in small bbox and big bbox equally
  - Not good for scoring

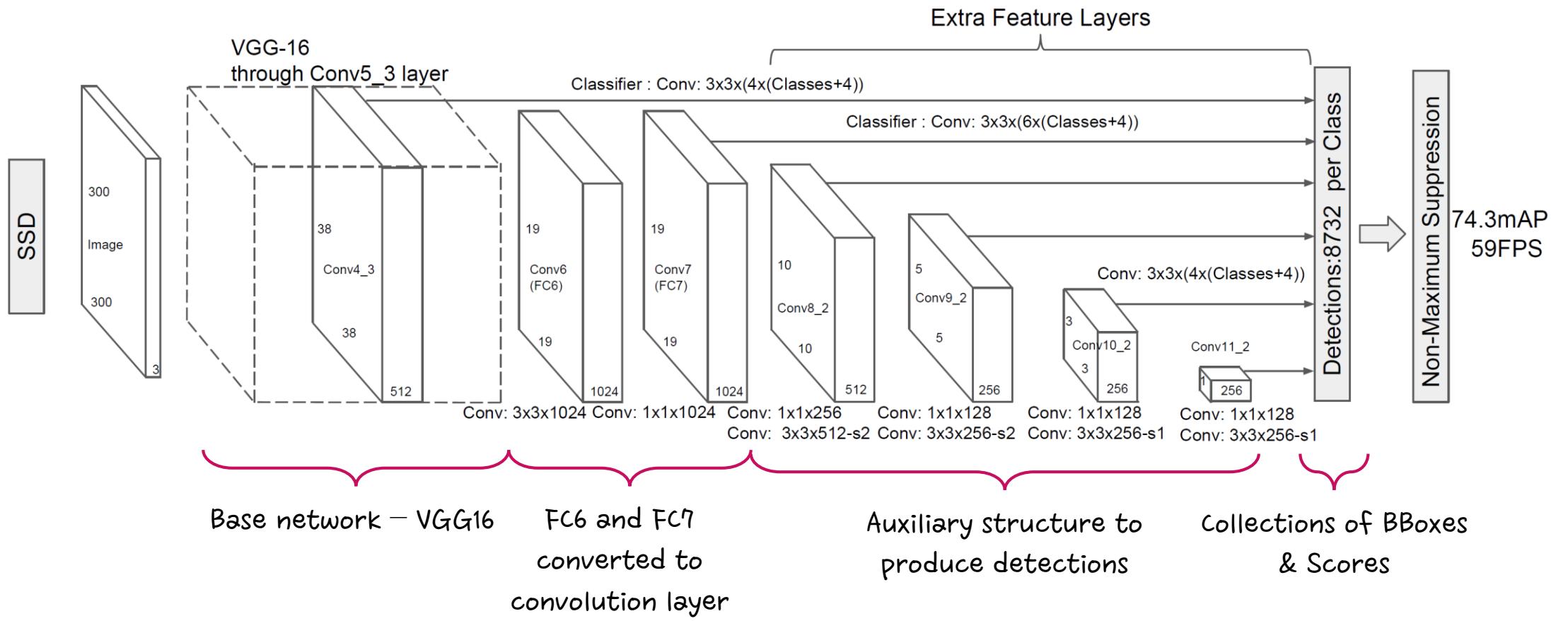
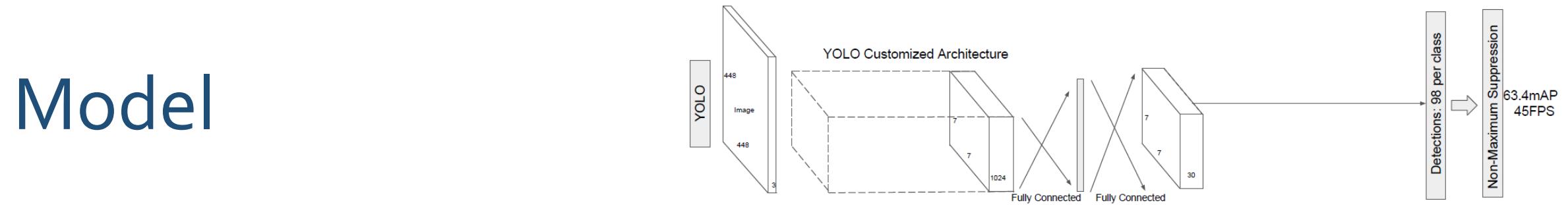
# SSD – Single Shot MultiBox Detector

- Current state-of-the-art object detection systems are variants of the following approach:
  - Hypothesize bounding boxes, resample pixels or features for each box, and apply a high-quality classifier.
- Computationally too intensive and too slow for real-time applications.
  - Faster R-CNN operates at only 7 FPS with mAP 73.2%
- Significantly increased speed comes only at the cost of significantly decreased detection accuracy.
  - YOLO operates at 45FPS with mAP 63.4%

# Single Shot MultiBox Detector

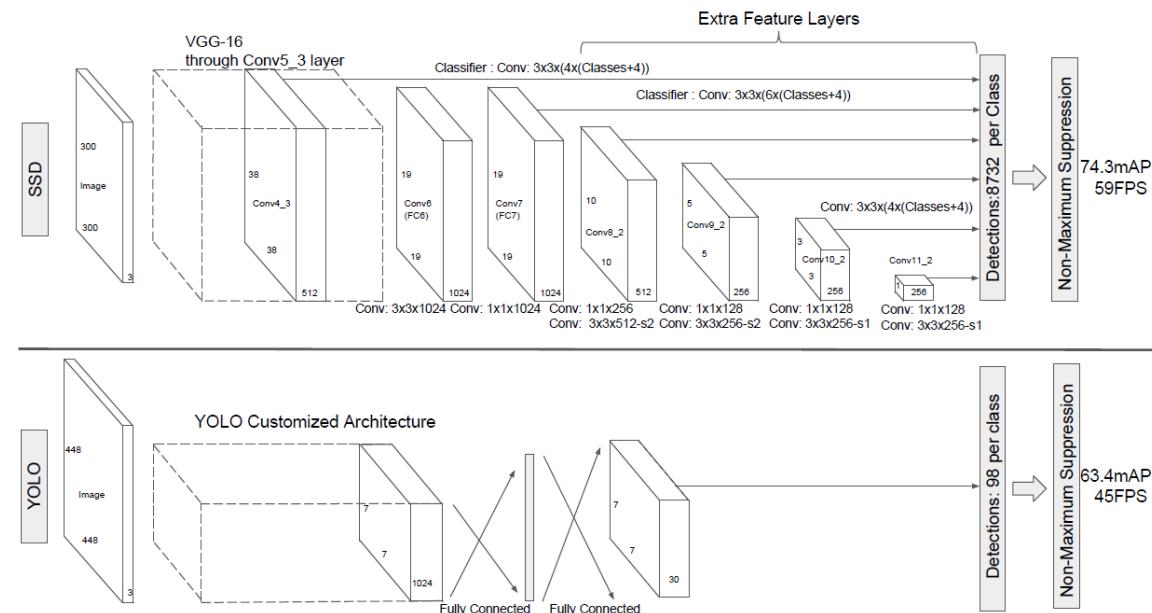
- First deep network based object detector that does not resample pixels or features for bounding box hypotheses(1-stage) and is as accurate as approaches that do.
  - 59 FPS with mAP 74.3% on VOC2007 test

# Model



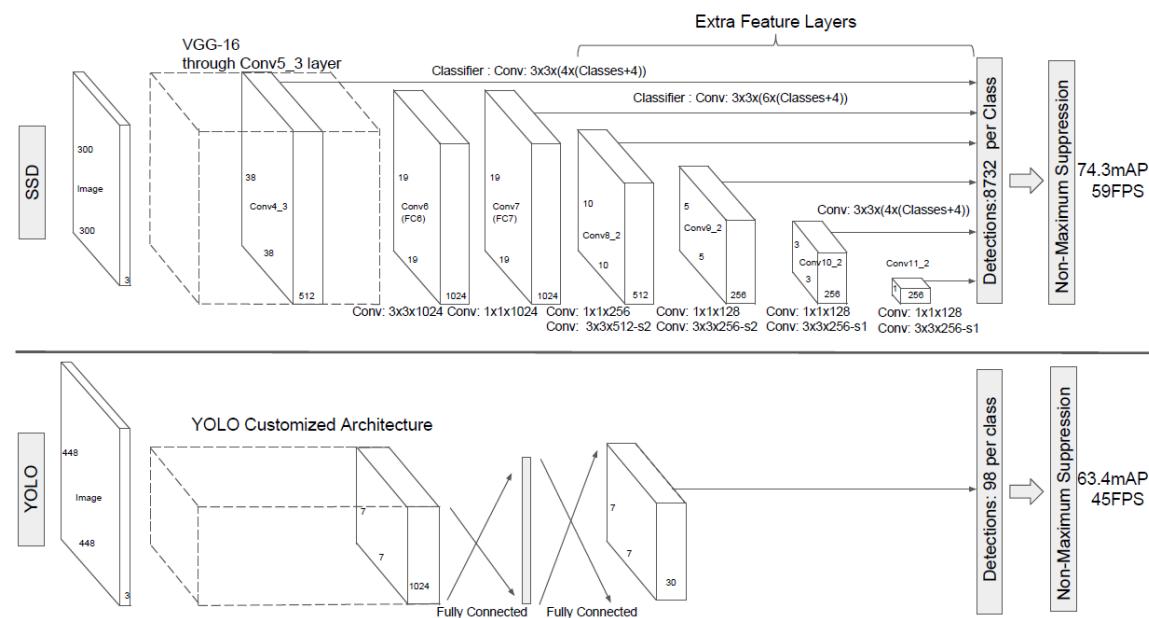
# Multi-scale Feature Maps for Detection

- Adding convolutional feature layers to the end of the truncated base networks.
- These layers decrease in size progressively and allow predictions of detections at multiple scales.
  - YOLO use only single scale feature map.



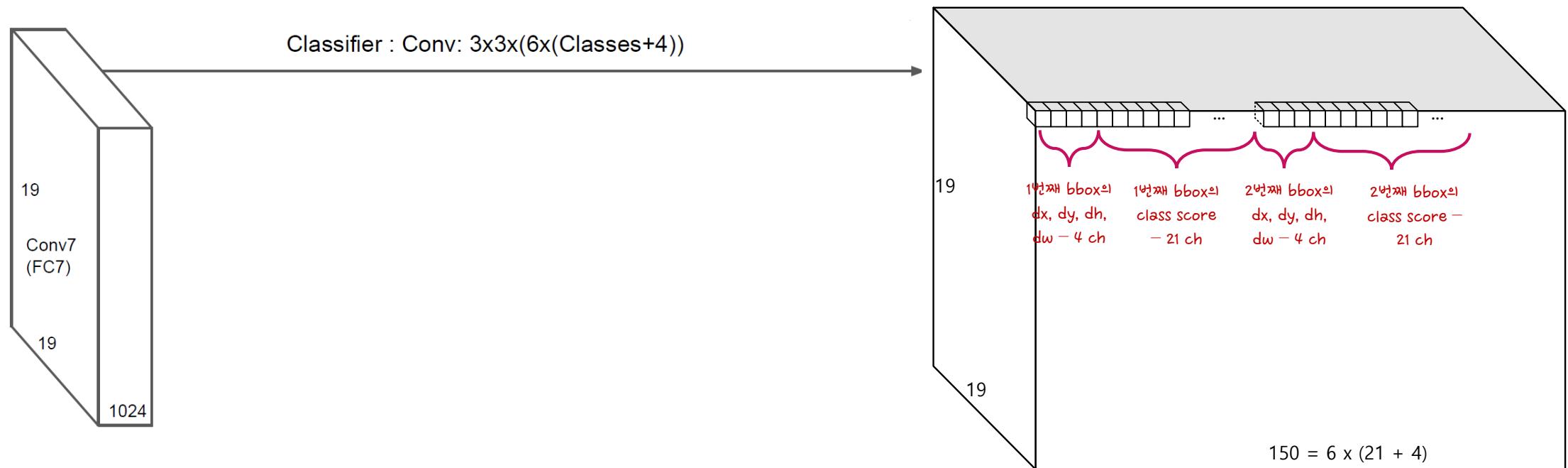
# Convolutional Predictors for Detection

- For a feature layer of size  $m \times n$  with  $p$  channels, the basic element for predicting parameters of a potential detection is a  $3 \times 3 \times p$  small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates.
  - YOLO has a fully connected layer for a score and a bbox coordinates.



# Default Boxes and Aspect Ratios

- For each box out of  $k$  at a given location, we compute  $c$  class scores and the 4 offsets relative to the original default box shape.
- Total of  $(c+4)k$  filters at each location in the feature map.
- Yielding  $(c+4)kmn$  outputs for a  $m \times n$  feature map.



# Training – Matching Strategy

- Matching each ground truth box to **the default box with the best jaccard overlap**. Every ground truth box has at least 1 correspondence.
- Then, matching **default boxes to any ground truth with jaccard overlap higher than a threshold(0.5)**.

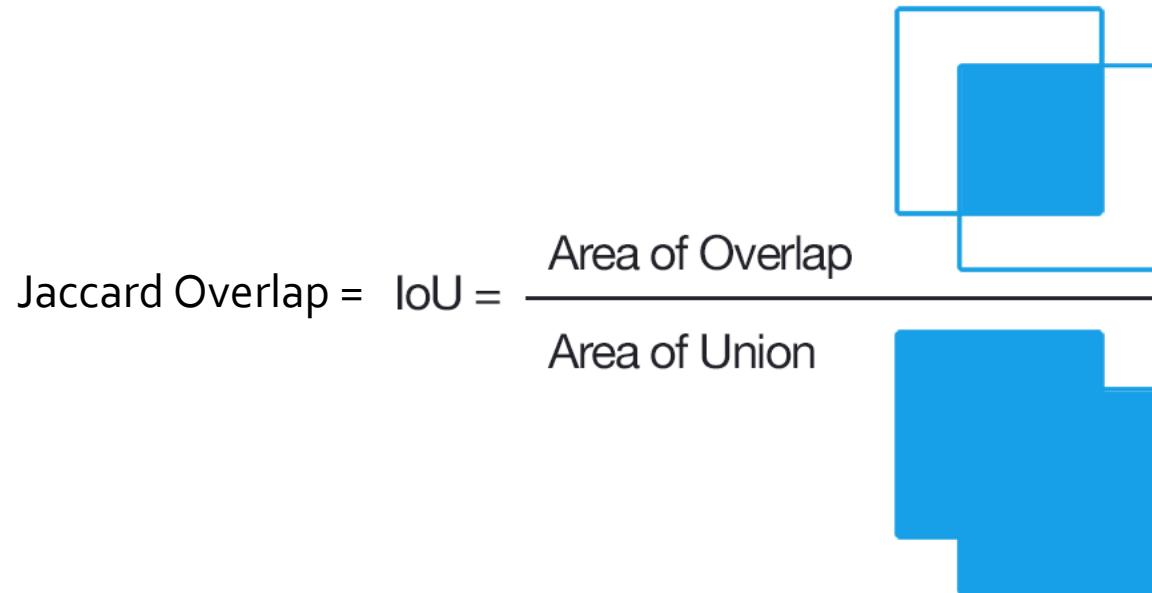


Figure from Wikipedia

# Training Objective

- Similar to Faster R-CNN

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

set to 1 by cross validation

- N : number of default matched bboxes
- $L_{conf}$  : classification loss → cross-entropy

$$L_{conf}(x, c) = - \sum_{i \in Pos}^N x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

- $L_{loc}$  : localization loss → Smooth L1 loss

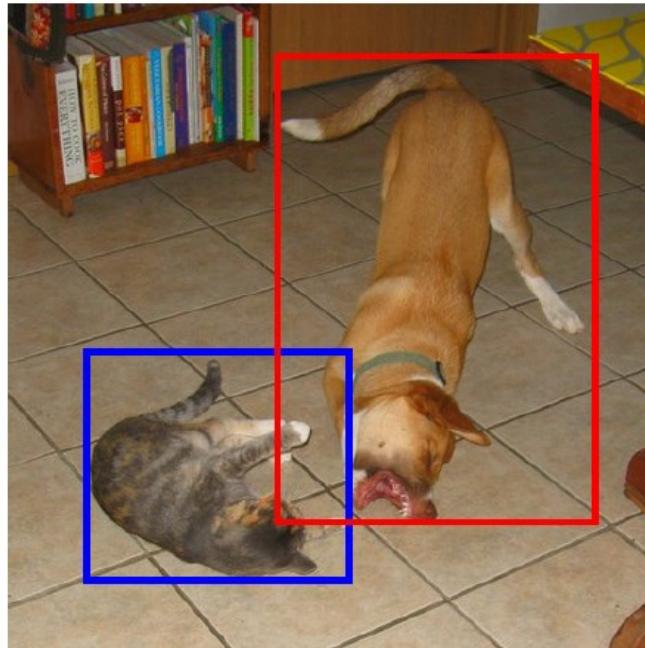
$$L_{loc}(x, l, g) = \sum_{i \in Pos}^N \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

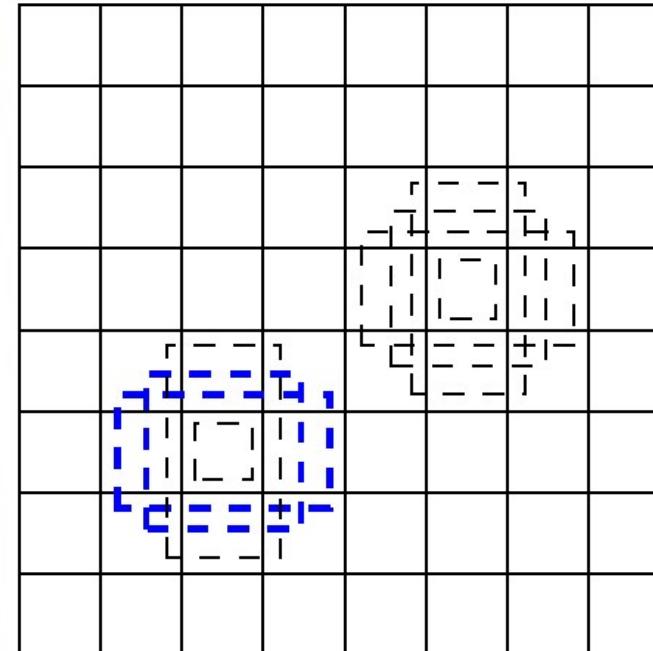
$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \quad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

# Choosing Scales and Aspect Ratios for Default Boxes

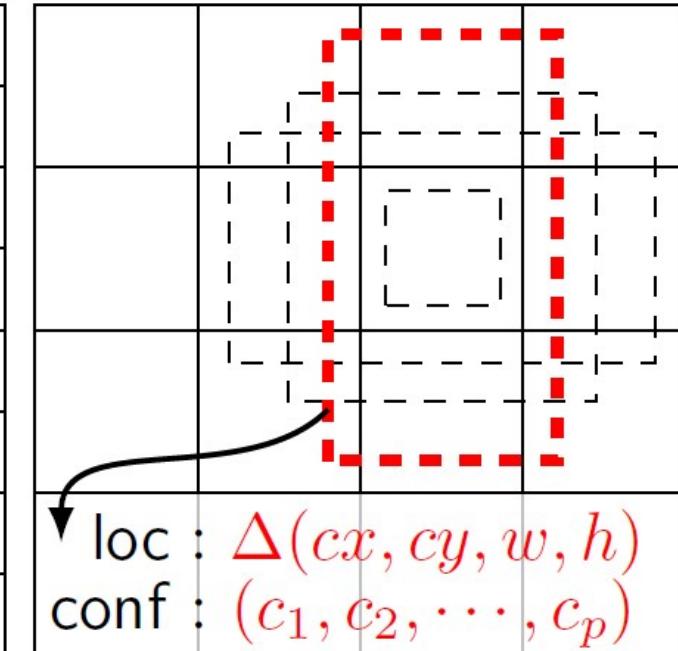
- Feature maps from different layers are used to handle scale variance.
- Specific feature map locations learn to be responsive to specific area of the image and particular scales of objects.



(a) Image with GT boxes



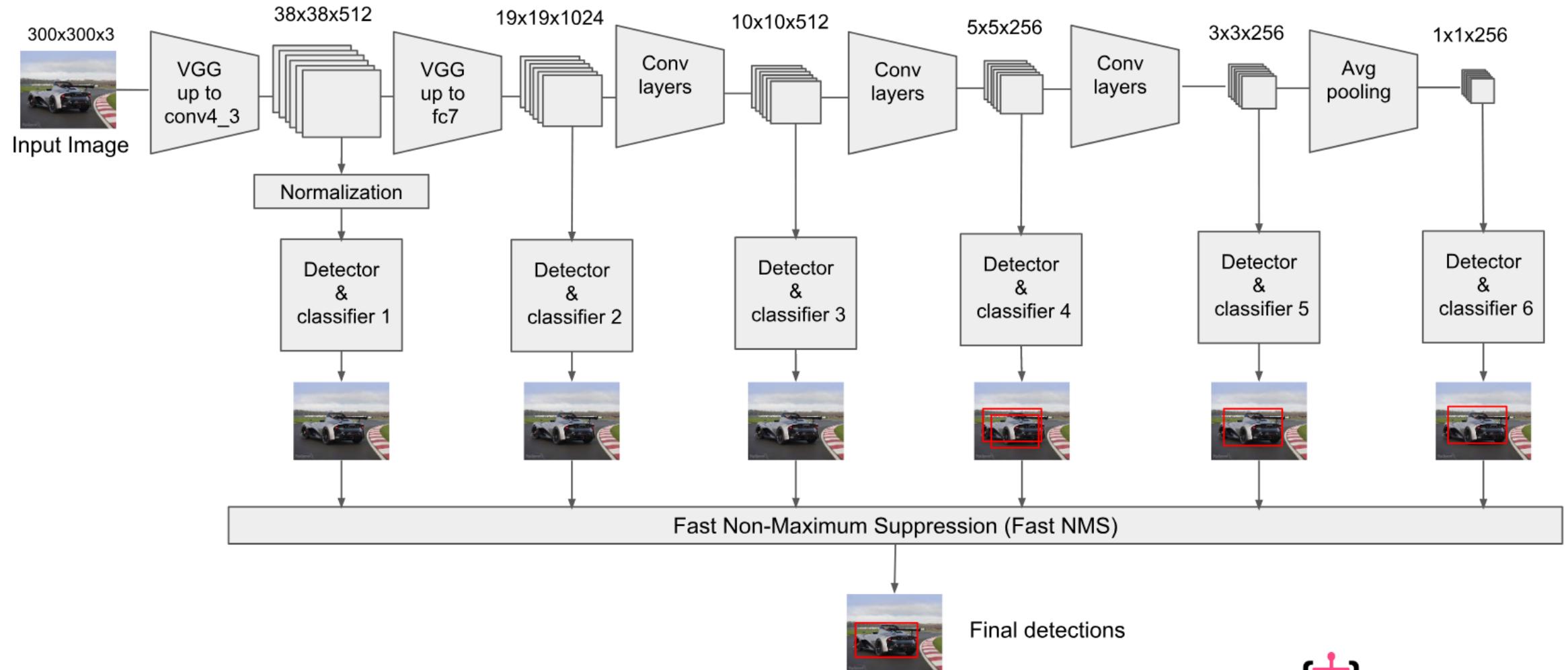
(b)  $8 \times 8$  feature map



(c)  $4 \times 4$  feature map

loc :  $\Delta(cx, cy, w, h)$   
conf :  $(c_1, c_2, \dots, c_p)$

# Choosing Scales and Aspect Ratios for Default Boxes



# Choosing Scales and Aspect Ratios for Default Boxes

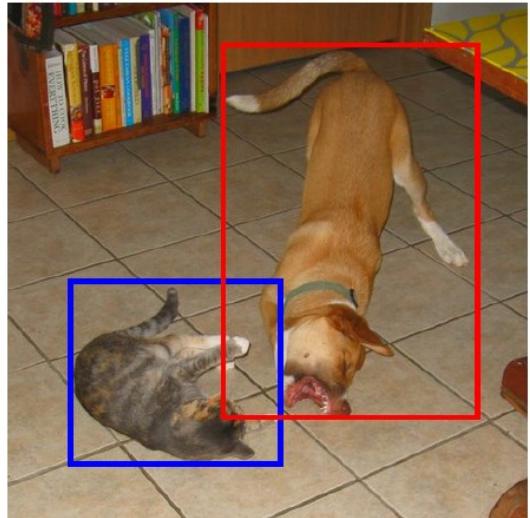
- If  $m$  default maps are used for prediction, the scale of the default boxes for each feature map is computed as:

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m - 1} (k - 1), \quad k \in [1, m]$$

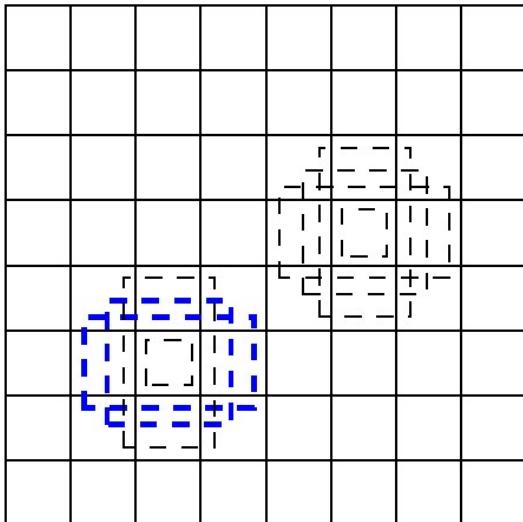
- $s_{\min}$  is 0.2 and  $s_{\max}$  is 0.9
- For example (PASCAL VOC 2007)
  - $s_k$  is 0.1, 0.2, 0.375, 0.55, 0.725, 0.9
  - means 30, 60, 112.5, 165, 217.5, 270 pixels @ input image(300x300)

# Choosing Scales and Aspect Ratios for Default Boxes

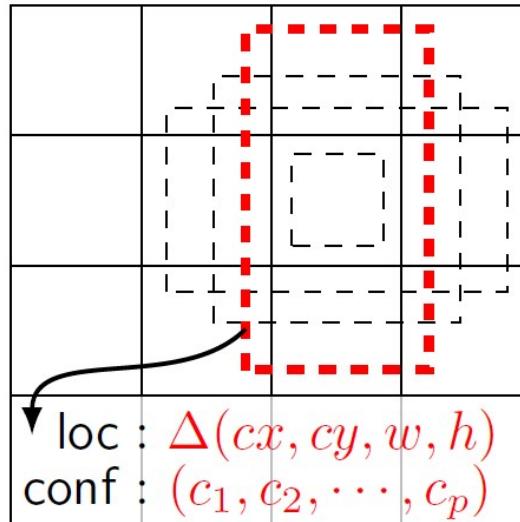
- At each scale, different aspect ratios are considered



(a) Image with GT boxes



(b)  $8 \times 8$  feature map



(c)  $4 \times 4$  feature map

$$a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$$

$$w_k^a = s_k \sqrt{a_r}$$

$$h_k^a = s_k / \sqrt{a_r}$$

For the aspect ratio of 1, one default box is added whose scale is

$$s'_k = \sqrt{s_k s_{k+1}}$$

# Hard Negative Mining

- Significant imbalance between positive and negative training examples.
  - After the matching step, most of the default boxes are negatives, especially when the number of possible default boxes is large.
- Sorting them using **the highest confidence loss** for each default box.
- Pick the top ones so that the ratio **between the negatives and positives is at most 3:1**

# Class Imbalance Problem – RetinaNet

- Class 간의 data가 균형이 안맞는 경우

- Weighted cross-entropy

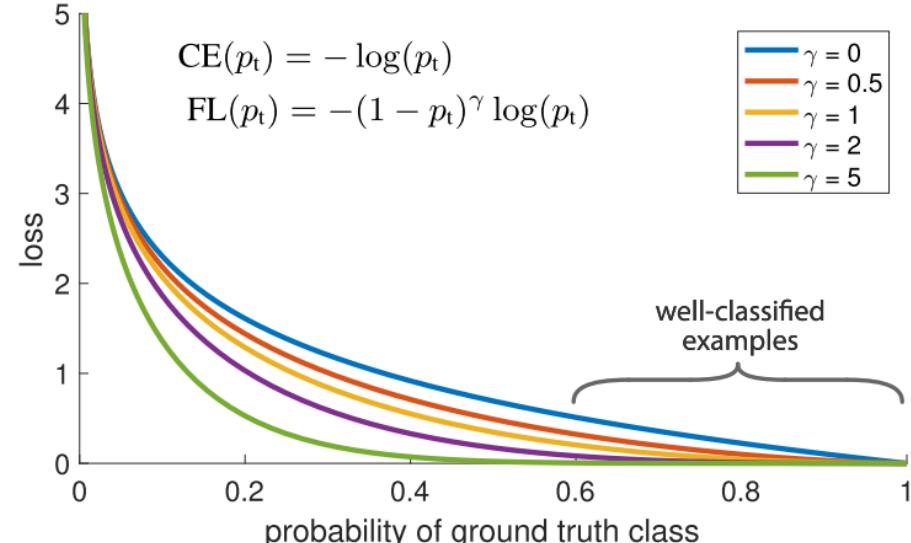
- Cross entropy 식에 class 별로 weight를 다르게 주어서 사용

$$L = - \sum_c w_c \cdot \log p$$

- Focal Loss

- Cross entropy에 비해 높은 확률의 경우에는 상대적으로 작은 loss를 나온 확률의 경우에는 높은 loss를 주도록  $(1-p)^\gamma$ 를 곱해줌

$$L = - \sum (1 - p)^\gamma \cdot \log p$$



# Experimental Results

- Base Network
  - VGG16 with fc6 and fc7 converted to conv layers and pool5 from  $2 \times 2 - s_2$  to  $3 \times 3 - s_1$  using atrous algorithm, removed fc8 and dropout
  - Fine-tuned using SGD with momentum

# PASCAL VOC2007 Detection Results

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
Fast [6]	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster [2]	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
Faster [2]	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
Faster [2]	07+12+COCO	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
SSD300	07	68.0	73.4	77.5	64.1	59.0	38.9	75.2	80.8	78.5	46.0	67.8	69.2	76.6	82.1	77.0	72.5	41.2	64.2	69.1	78.0	68.5
SSD300	07+12	74.3	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
SSD300	07+12+COCO	79.6	80.9	86.3	79.0	<b>76.2</b>	57.6	87.3	88.2	88.6	60.5	85.4	<b>76.7</b>	<b>87.5</b>	<b>89.2</b>	84.5	81.4	55.0	81.9	<b>81.5</b>	85.9	78.9
SSD512	07	71.6	75.1	81.4	69.8	60.8	46.3	82.6	84.7	84.1	48.5	75.0	67.4	82.3	83.9	79.4	76.6	44.9	69.9	69.1	78.1	71.8
SSD512	07+12	76.8	82.4	84.7	78.4	73.8	53.2	86.2	87.5	86.0	57.8	83.1	70.2	84.9	85.2	83.9	79.7	50.3	77.9	73.9	82.5	75.3
SSD512	07+12+COCO	<b>81.6</b>	<b>86.6</b>	<b>88.3</b>	<b>82.4</b>	76.0	<b>66.3</b>	<b>88.6</b>	<b>88.9</b>	<b>89.1</b>	<b>65.1</b>	<b>88.4</b>	73.6	86.5	88.9	<b>85.3</b>	<b>84.6</b>	<b>59.1</b>	<b>85.0</b>	80.4	<b>87.4</b>	<b>81.2</b>

# Sensitivity to Different Object Characteristics

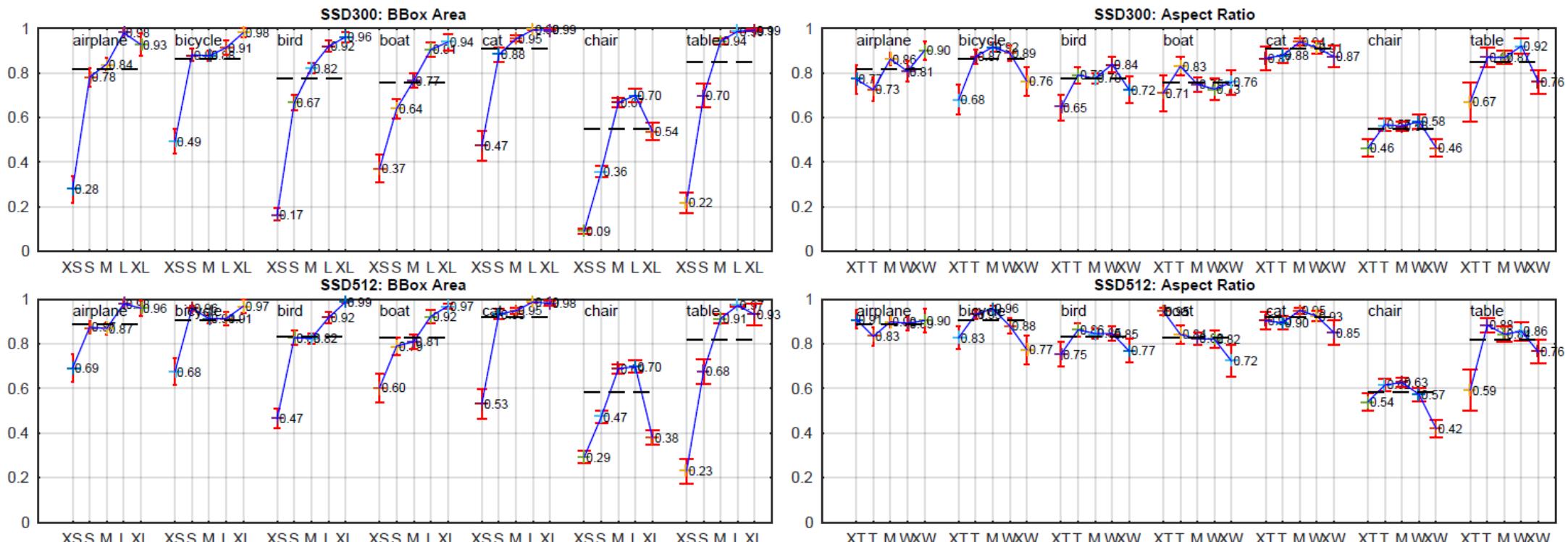
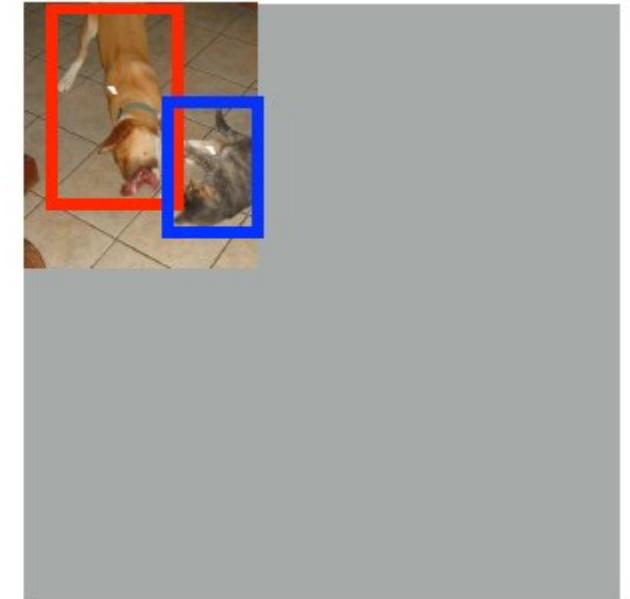
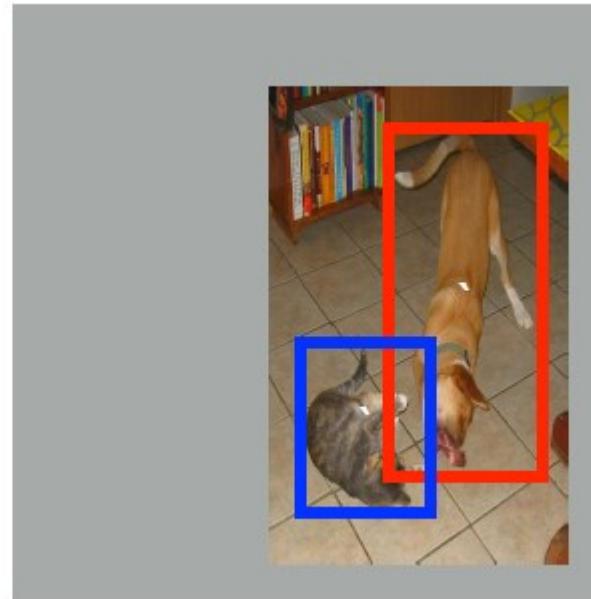
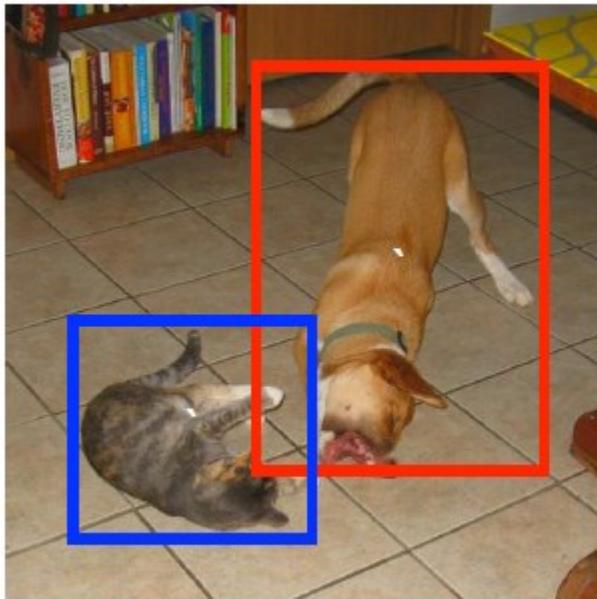


Fig. 4: **Sensitivity and impact of different object characteristics on VOC2007 test set using [21].** The plot on the left shows the effects of BBox Area per category, and the right plot shows the effect of Aspect Ratio. Key: BBox Area: XS=extra-small; S=small; M=medium; L=large; XL =extra-large. Aspect Ratio: XT=extra-tall/narrow; T=tall; M=medium; W=wide; XW =extra-wide.

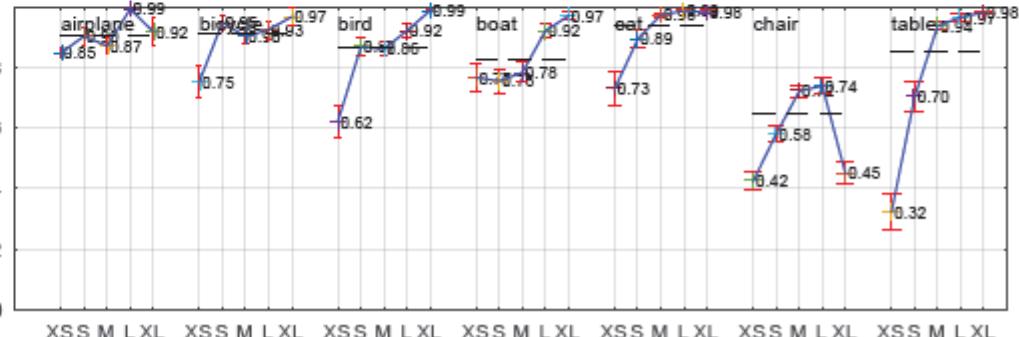
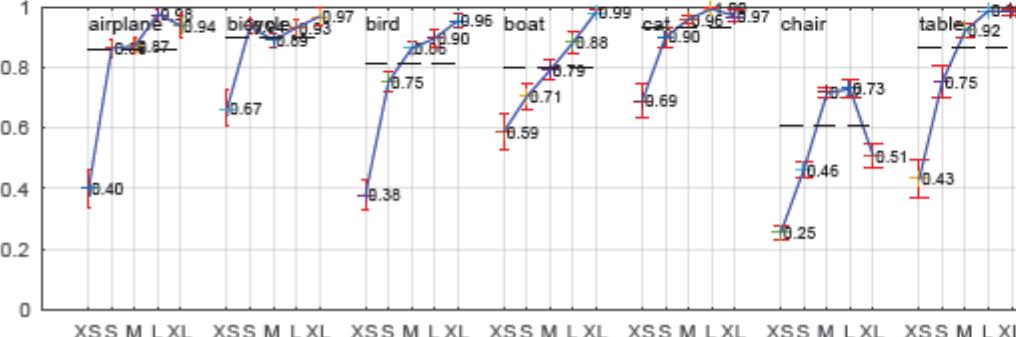
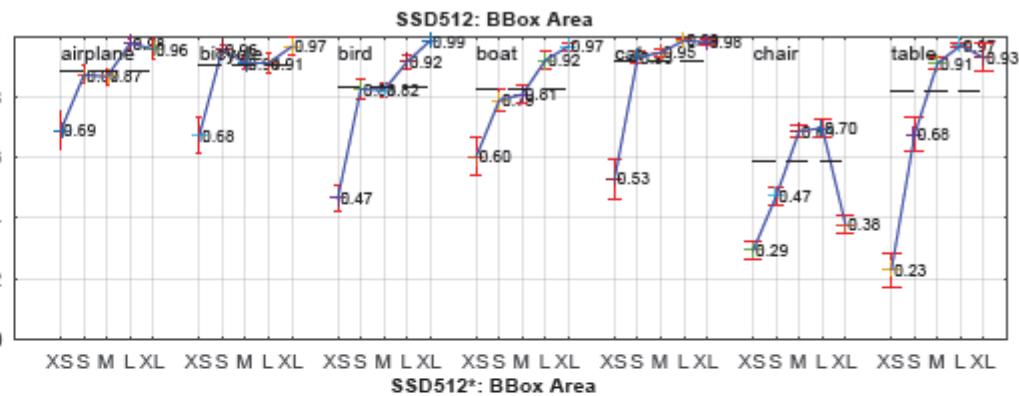
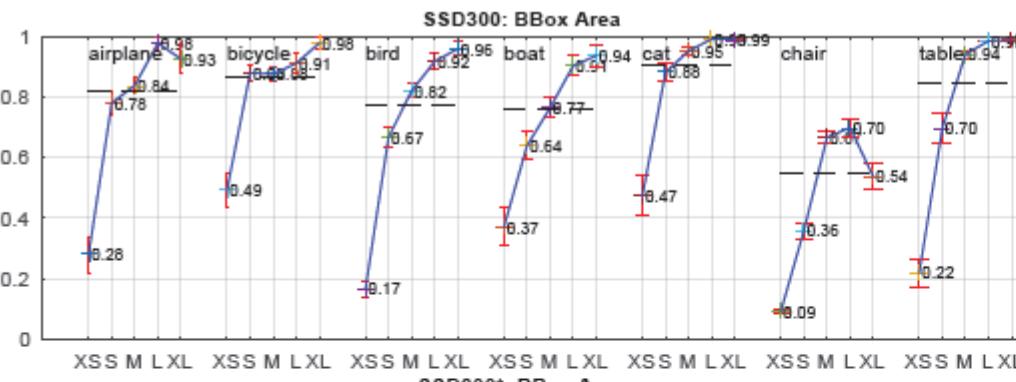
# Data Augmentation for Small Objects

- Randomly place an image on a canvas of 16 of the original image size filled with mean values before doing any random crop operation.



# Results on Multiple Datasets with New Data Augmentation

Method	VOC2007 test		VOC2012 test		COCO test-dev2015 trainval35k		
	07+12 0.5	07+12+COCO 0.5	07++12 0.5	07++12+COCO 0.5	0.5:0.95	0.5	0.75
SSD300	74.3	79.6	72.4	77.5	23.2	41.2	23.4
SSD512	76.8	81.6	74.9	80.0	26.8	46.5	27.8
SSD300*	77.2	81.2	75.8	79.3	25.1	43.1	25.8
SSD512*	<b>79.8</b>	<b>83.2</b>	<b>78.5</b>	<b>82.2</b>	<b>28.8</b>	<b>48.5</b>	<b>30.3</b>



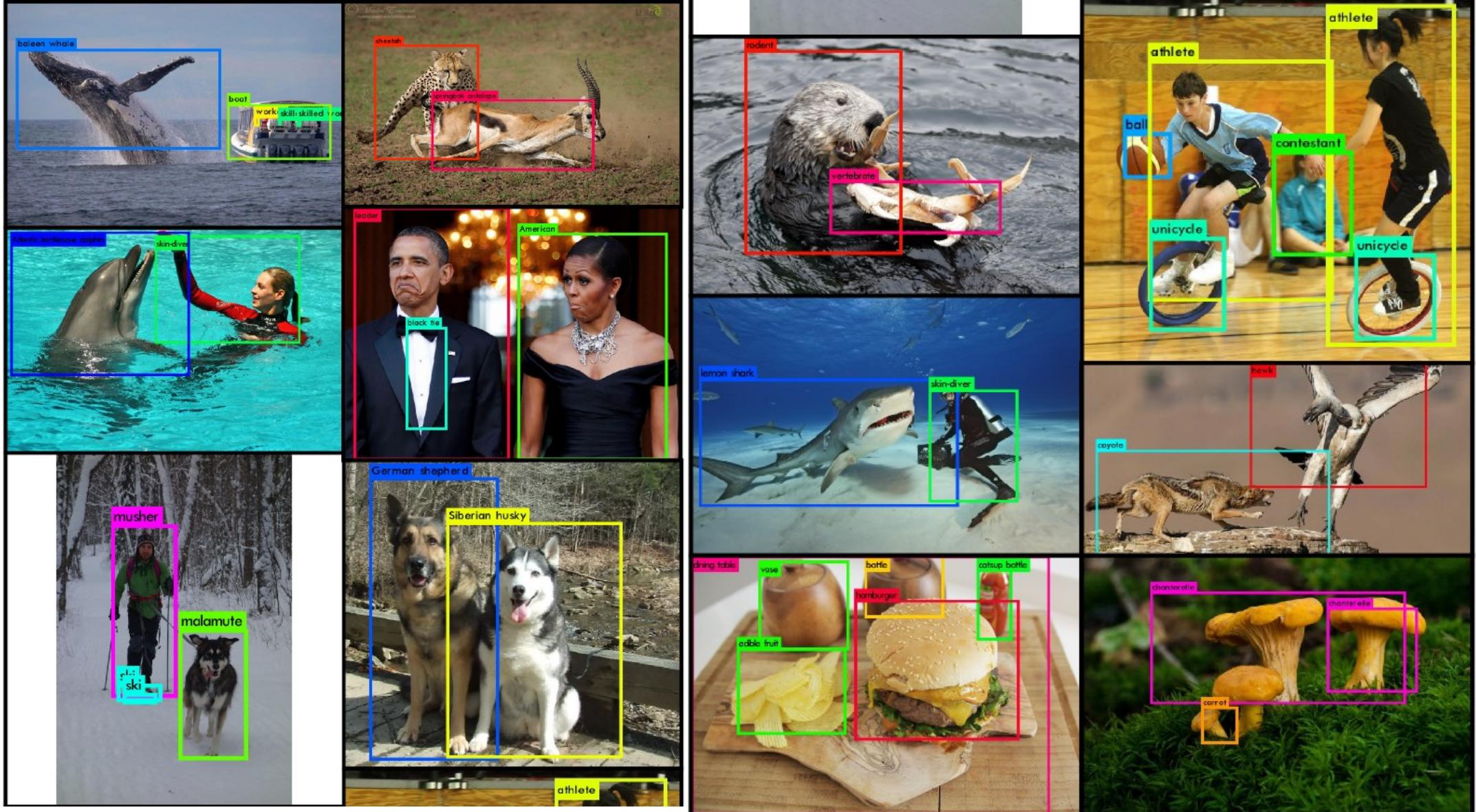
# Inference Time

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

# Let's See the Result First



# Let's See the Result First



# YOLO9000 - Better, Faster, and Stronger

- Better
  - Batch normalization
  - High resolution classifier
  - Convolution with anchor boxes
  - Dimension clusters
  - Direct location prediction
  - Fine-grained features
  - Multi-scale training
- Faster
  - Darknet-19
  - Training for classification
  - Training for detection
- Stronger
  - Hierarchical classification
  - Dataset combination with Word-tree
  - Joint classification and detection

YOLOv2

YOLO9000

# Introduction & Motivation

- Detection frameworks have become increasingly fast and accurate  
→ However, most detection methods are still constrained to a small set of objects.
- We would like detection to scale to level of object classification → However, labelling images for detection is far more expensive than labelling for classification.
- Maybe we cannot see detection datasets on the same scale as classification datasets in the near future.

# Introduction

- Authors propose a new method to expand the scope of current detection systems by using classification dataset we already have.
- Joint training algorithm is also proposed that allows to train object detectors on both detection and classification data
- First, improving upon the base YOLO detection system to produce YOLOv2 (SOTA real-time detector)
- Then, using dataset combination method and joint training algorithm to train model on more than 9000 classes!

# To Improve YOLO

- YOLO's shortcomings relative to SOTA detection systems
  - YOLO makes a **significant number of localization errors**
  - YOLO has relatively **low recall** compared to region proposal based method
- Focus mainly on **improving recall and localization while maintain classification accuracy.**
- It is still very important to detect **FAST!**

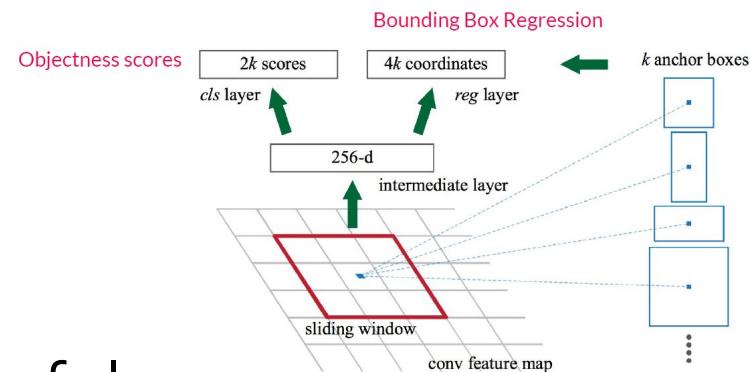
# Better

- Batch Normalization
  - Adding BN on all of conv layers in YOLO – **2% improvement in mAP**
  - **Removing dropout** without overfitting
- High Resolution Classifier
  - YOLO trains the classifier network @ $224 \times 224$  and increase resolution to  $448$  for detection → The network has to simultaneously switch to learning object detection and adjust to the new input resolution
  - **Fine tuning the classifier network @ $448 \times 448$  resolution for 10 epochs on ImageNet – almost 4% improvement in mAP**

# Better

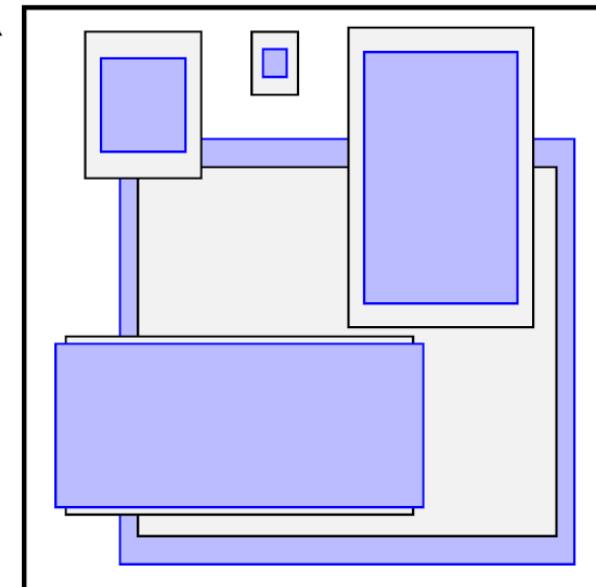
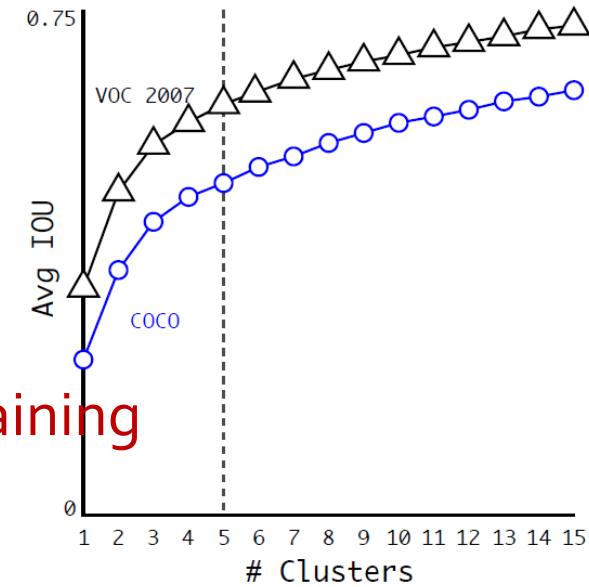
- Convolution with Anchor Boxes

- YOLO predicts the coordinates of b-boxes directly using fc layers.
- Faster R-CNN predicts b-boxes using hand-picked priors(anchor boxes).
- **YOLOv2 removes the fc layers and use anchor boxes** to predict bounding boxes
- Shrinking the network to operate on **416x416 input images instead of 448x448**
  - YOLO's CNN downsample the image by a factor of 32
  - Objects, especially large objects, **tend to occupy the center of the image**, so it is **better to have one cell in the middle than to have four cells**. → odd number of grid cells are required( $416 \times 416 \rightarrow 13 \times 13$ )
- **Predicting class and objectness for every anchor box**
- Using anchor boxes makes a small decrease in accuracy
  - **69.5mAP to 69.2mAP but 81% recall to 88% recall** → high recall means there is more room to improve



# Better

- Dimension Clusters
  - The problem with anchor boxes is that they are hand-picked
  - Running k-means clustering on the training set b-boxes to find good priors
  - To prevent larger boxes from making more error, they use a different distance metric(not Euclidian distance)
    - $d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$
  - $k=5$  is chosen as a good tradeoff between complexity and high recall



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

Table 1: Average IOU of boxes to closest priors on VOC 2007.

# Better

- Direct Location Prediction
  - Another problem with anchor boxes is **instability**, in RPNs the **anchor box can be anywhere in the image**, regardless of what location predicted the box
  - Instead of predicting offsets, YOLOv2 predicts **locations relative to the location of the grid cells**
  - 5 bounding boxes for each cell, and **5 values for each bounding box** → almost 5% improvement over the version with anchor boxes

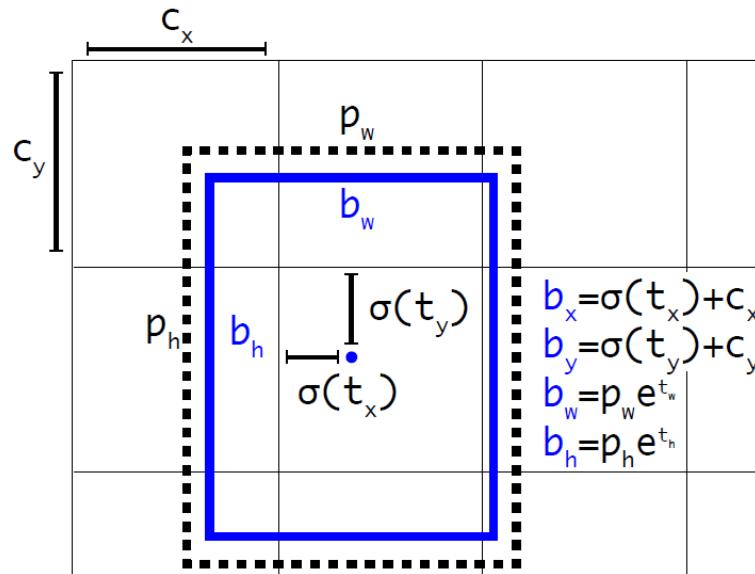
RCNN's b-box regression

$$\hat{G}_x = P_w d_x(P) + P_x$$

$$\hat{G}_y = P_h d_y(P) + P_y$$

$$\hat{G}_w = P_w \exp(d_w(P))$$

$$\hat{G}_h = P_h \exp(d_h(P)).$$



Different equations  
– direct prediction

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

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

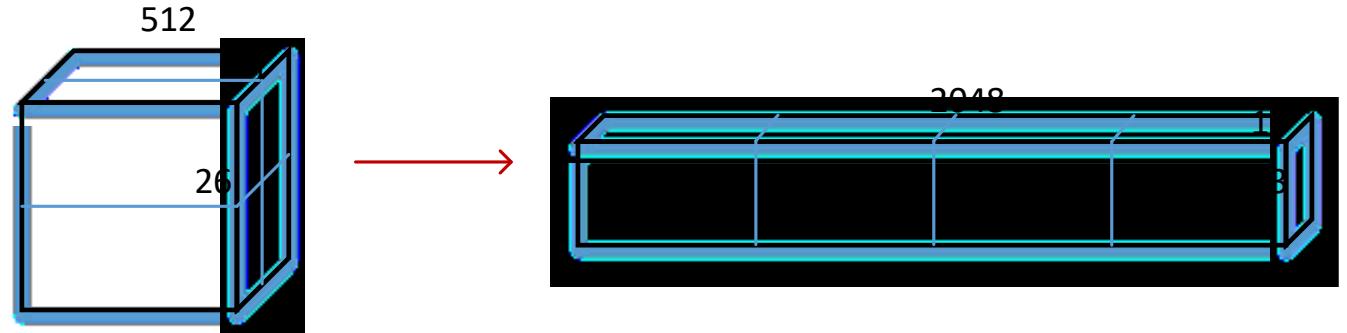
$$b_w = p_w e^{t_w}$$

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

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

# Better

- Fine-Grained Features



- Modified YOLO uses  $13 \times 13$  feature maps to predict detections
  - Good at large object, finer grained features are required for small objects
- Simply **adding a passthrough layer** that brings features from an earlier layer at  $26 \times 26$  resolution
- Turn  **$26 \times 26 \times 512$  feature map into a  $13 \times 13 \times 2048$  feature map** which can be concatenated with original features
- Detector runs on top of this expanded features
- **1% performance increase**

# Better

- Multi-Scale Training

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	<b>78.6</b>	40

- Removing fc layers → can be resized on the fly
- Every 10 batches YOLOv2 randomly choose a new image dimension size in {320, 352, ..., 608}
- Effect of the input size
  - At low resolution, fair detection performance but runs fast
  - At high resolution, SOTA detection and slower but still above real time speeds

# Further Experiments

- Pascal VOC2012 results

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

- COCO2015 results

			0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [5]	train		19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[1]	train		20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[15]	trainval		21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train		23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[10]	trainval		24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [11]	trainval35k		23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [11]	trainval35k		<b>26.8</b>	<b>46.5</b>	<b>27.8</b>	<b>9.0</b>	<b>28.9</b>	<b>41.9</b>	<b>24.8</b>	<b>37.5</b>	<b>39.8</b>	<b>14.0</b>	<b>43.5</b>	<b>59.0</b>
YOLOv2 [11]	trainval35k		21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

# Faster

- Darknet-19
  - mostly **3x3 filters** (similar to VGG)
  - Following the work on Network in Network (NIN), **use global average pooling** to make predictions as well as **1x1 filters to compress the feature representation** between 3x3 convolutions
  - Darknet-19: **19 convolutional layers & 5 max-pooling layers**
  - **5.58 billion operations: 72.9% top-1 & 91.2% top-5 accuracy** (30.69 billion operations & 90.0% top-5 accuracy for VGG-16)

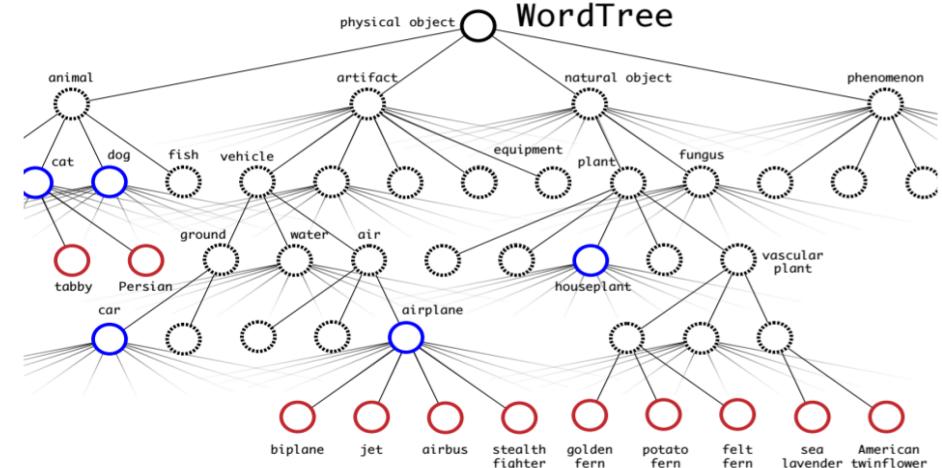
Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional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

# Faster

- Training for classification
  - ImageNet 1000 classes for 160 epochs
  - Standard data augmentation: random crops, rotations, hue, saturation, and exposure shifts
  - Initial training : 224x224 → 448x448 fine-tuning for 10 epochs
    - Higher resolution achieves a top-5 accuracy of 93.3%
- Training for detection
  - Adding 3x3 conv layers with 1024 filters each followed by a final 1x1 conv layer
  - For VOC, predicting 5 boxes with 5 coordinates each and 20 classes per box, so 125 filters
  - 160 epochs with a start learning rate of  $10^{-3}$  dividing it by 10 at 60 and 90 epochs

# Stronger

- Hierarchical classification
  - **ImageNet labels are pulled from WordNet, a language database that structures concepts and how they relate**
    - “Norfolk terrier” and “Yorkshire terrier” are both hyponyms of “terrier” which is a type of “hunting dog”, which is a type of “dog”, which is a “canine”, etc.
  - **WordNet is structured as a directed graph, not a tree, because language is complex**
    - To make a tree not a graph, If a concept has tow paths to the root, **choose the shorter path**
  - Root note is a “Physical Object”



# Stronger

- Hierarchical classification
  - To perform classification with WordTree, predicting conditional probabilities at every node for the probability of each hyponym of that synset given that synset
$$Pr(\text{Norfolk terrier}|\text{terrier})$$
$$Pr(\text{Yorkshire terrier}|\text{terrier})$$
$$Pr(\text{Bedlington terrier}|\text{terrier})$$
$$\dots$$
  - Then if a picture of a Norfolk terrier is encountered, its probability is calculated as below

$$Pr(\text{Norfolk terrier}) = Pr(\text{Norfolk terrier}|\text{terrier})$$

$$*Pr(\text{terrier}|\text{hunting dog})$$

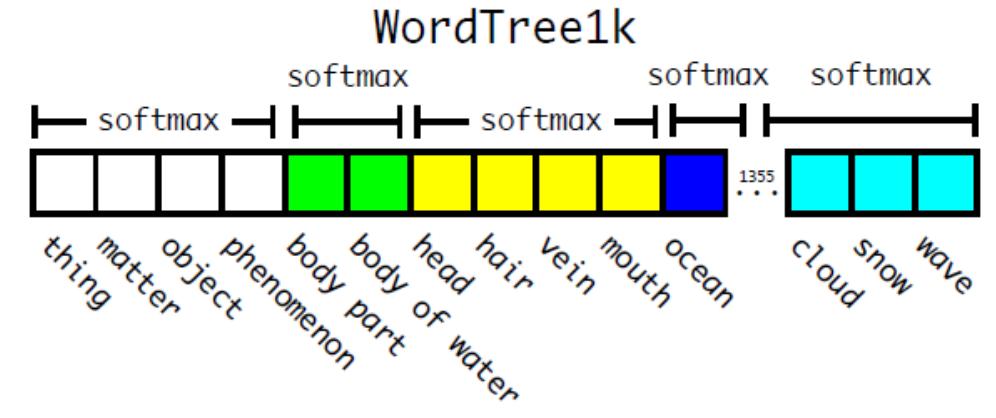
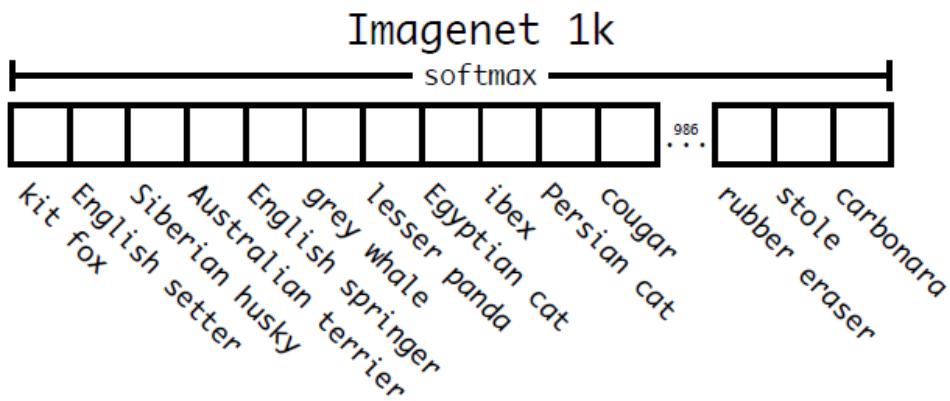
$* \dots *$

$$*Pr(\text{mammal}|Pr(\text{animal}))$$

$$*Pr(\text{animal}|\text{physical object})$$

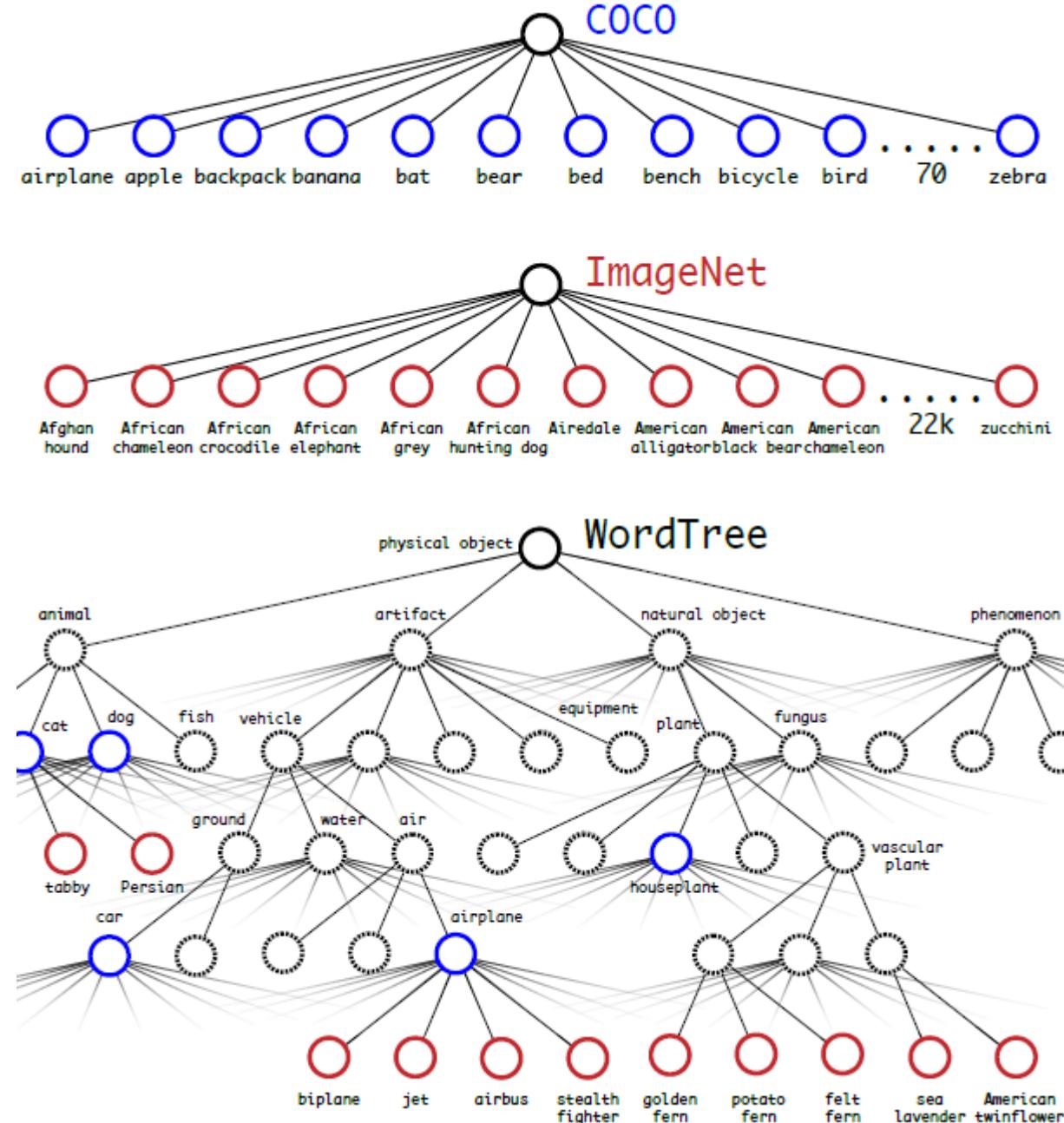
# Stronger

- Hierarchical classification
  - To build WordTree1k, adding in all of the intermediate nodes which expands the label space **from 1000 to 1369**
  - During training, **ground truth labels are propagated up**
    - If an image is labelled as a “Norfolk terrier” it also gets labelled as a “dog” and a “mammal”, etc.
  - To compute the conditional probabilities the model predicts a vector of 1369 values and **computes the softmax over all synsets that are hyponyms of the same concept** → 90.4% top-5 accuracy



# Stronger

- Dataset Combination with WordTree
  - Example of using WordTree to combine the labels from ImageNet and COCO.
    - COCO – **general concepts**(higher nodes)
    - ImageNet – **specific concepts**(lower nodes and leaves)



# Stronger

- Joint Classification and Detection
  - New dataset created using the **COCO detection dataset and the top 9000 classes from the full ImageNet release**. ImageNet detection challenge dataset is also added for evaluation
  - Using YOLOv2 but only **3 priors** instead of 5 to limit output size
  - The network sees a **detection image, backpropagate loss as normal**. For classification loss, only **backpropagate loss at or above the corresponding level and only backpropagate classification loss**
    - Simply find the b-box that predicts the highest probability for that class and compute the loss on just its predicted tree

# Stronger

- Joint Classification and Detection
  - Evaluating YOLO9000 on the ImageNet detection task
    - ImageNet only share 44 object categories with COCO
    - 19.7mAP overall with 16.0mAP on the disjoint 156 object classes that it has never seen
  - YOLO9000 learns new species of animals well but struggles with learning categories like clothing and equipment

diaper	0.0
horizontal bar	0.0
rubber eraser	0.0
sunglasses	0.0
swimming trunks	0.0
...	
red panda	50.7
fox	52.1
koala bear	54.3
tiger	61.0
armadillo	61.7