



**LG전자 Deep Learning 과정**

## Object Detection

Gunhee Kim

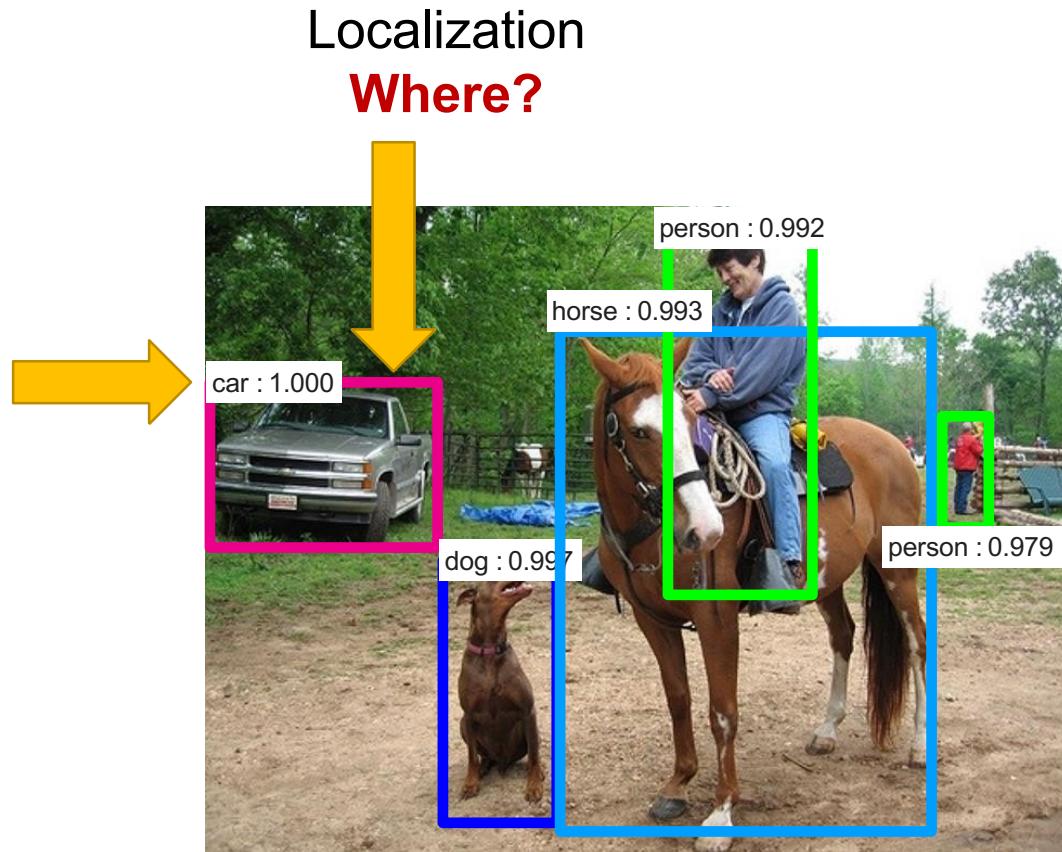
Computer Science and Engineering



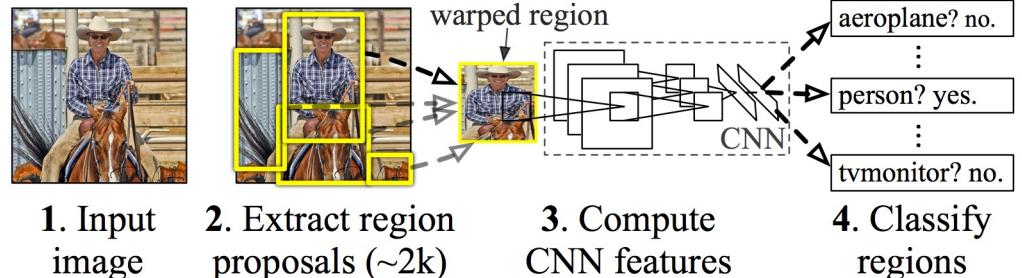
서울대학교  
SEOUL NATIONAL UNIVERSITY

# Object Detection

Recognition  
**What?**



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



# R-CNN

*Rich feature hierarchies for accurate object detection and semantic segmentation (CVPR'14)*

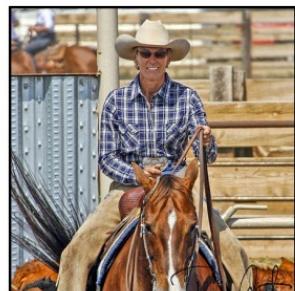
Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik

<https://github.com/rbgirshick/rcnn>

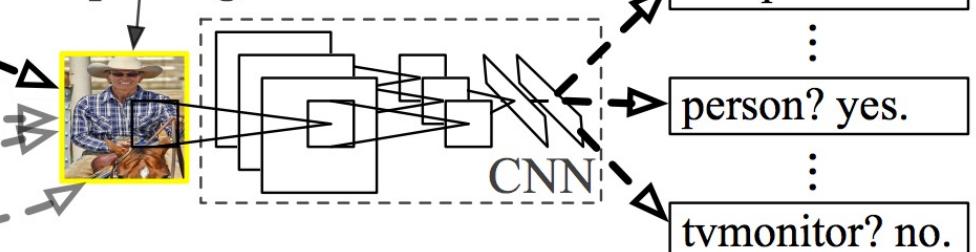
# R-CNN: We already know this!

For each region proposal (from SS or EdgeBox), compute the CNN feature of that region and classify

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



warped region



1. Input image

2. Extract region proposals (~2k)

3. Compute CNN features

4. Classify regions

Code is at <https://github.com/rbgirshick/rcnn>

# R-CNN: Slow and Naive

Depends on the external region proposal algorithm

Due to image cropping/warping, image resolution can be lost

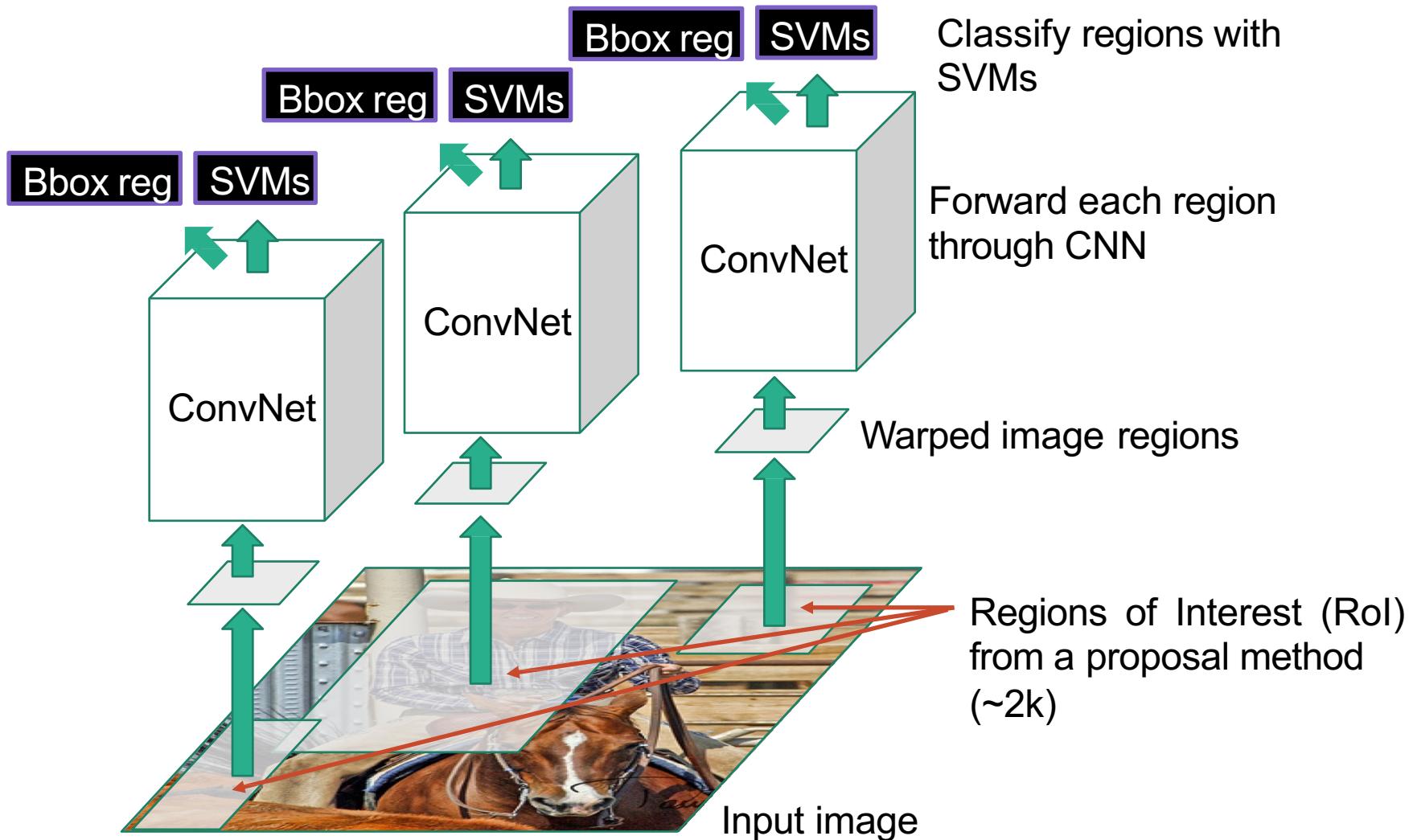
Need to feed-forward multiple times (i.e. proportional to the number of proposals, e.g.  $\sim 2K$ )

- Inference is very slow, about  $\sim 10\text{sec} / \text{image}$

External region proposals are slow, CPU only, and require a complex pipeline (no end-to-end)

- Feature caching requires a lot of disk...
- Training is very slow ( $\sim 84 \text{ hrs}$ )

# R-CNN: Slow and Naive





# SPPNET

*Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition* (ECCV'14)

Kaiming, He and Xiangyu, Zhang and Shaoqing, Ren and Jian Sun

[https://github.com/ShaoqingRen/SPP\\_net](https://github.com/ShaoqingRen/SPP_net)

# Key Problems of Previous Approaches...

1. CNN requires a fixed input image size (e.g. 224x224)

- Limit both the aspect ratio and the scale of input image
- Cropping: may not contain the entire object
- Warping: unwanted geometric distortion

2. R-CNN is time-consuming: repeatedly apply CNNs to each of thousands ROIs



crop



warp



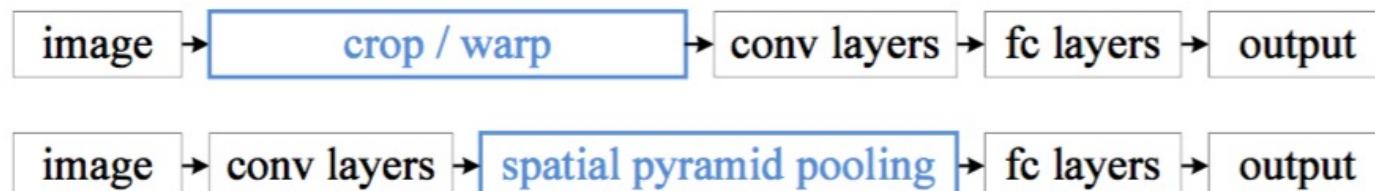
# Key Ideas of SPPNet

Use spatial pyramid pooling in the context of CNNs

1. Handle arbitrary sized input and multi-resolution images

- Later FC layer requires fixed-size input
- SPP can generate a fixed-length output regardless of input sizes

2. Use a **single pass** of CNN on the entire image

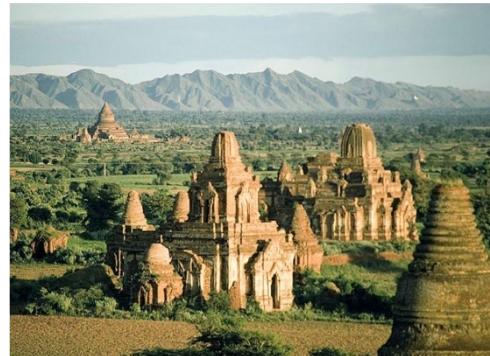


# Spatial Pyramid

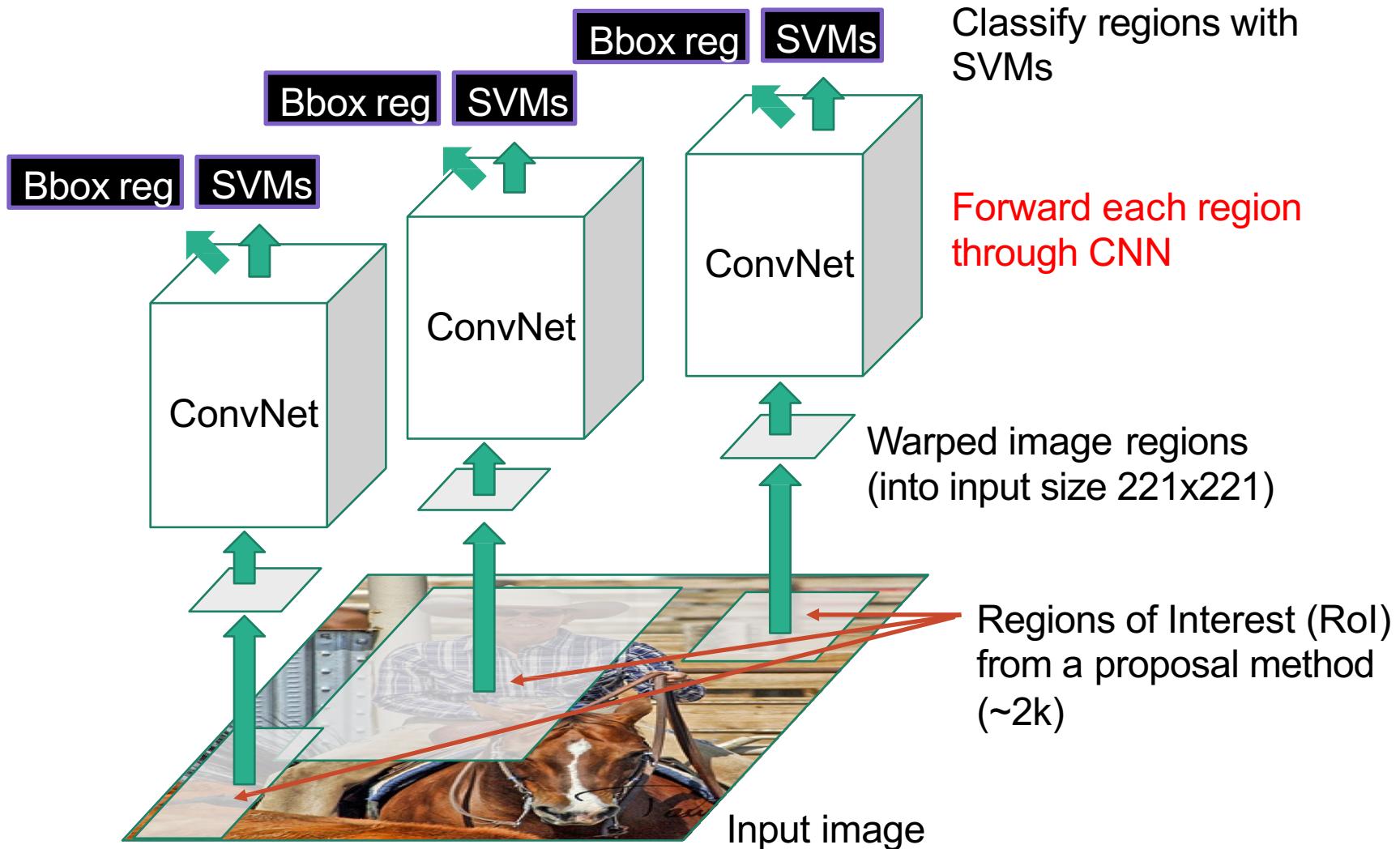
Compute histogram in each spatial bin

- Extension of a bag of features

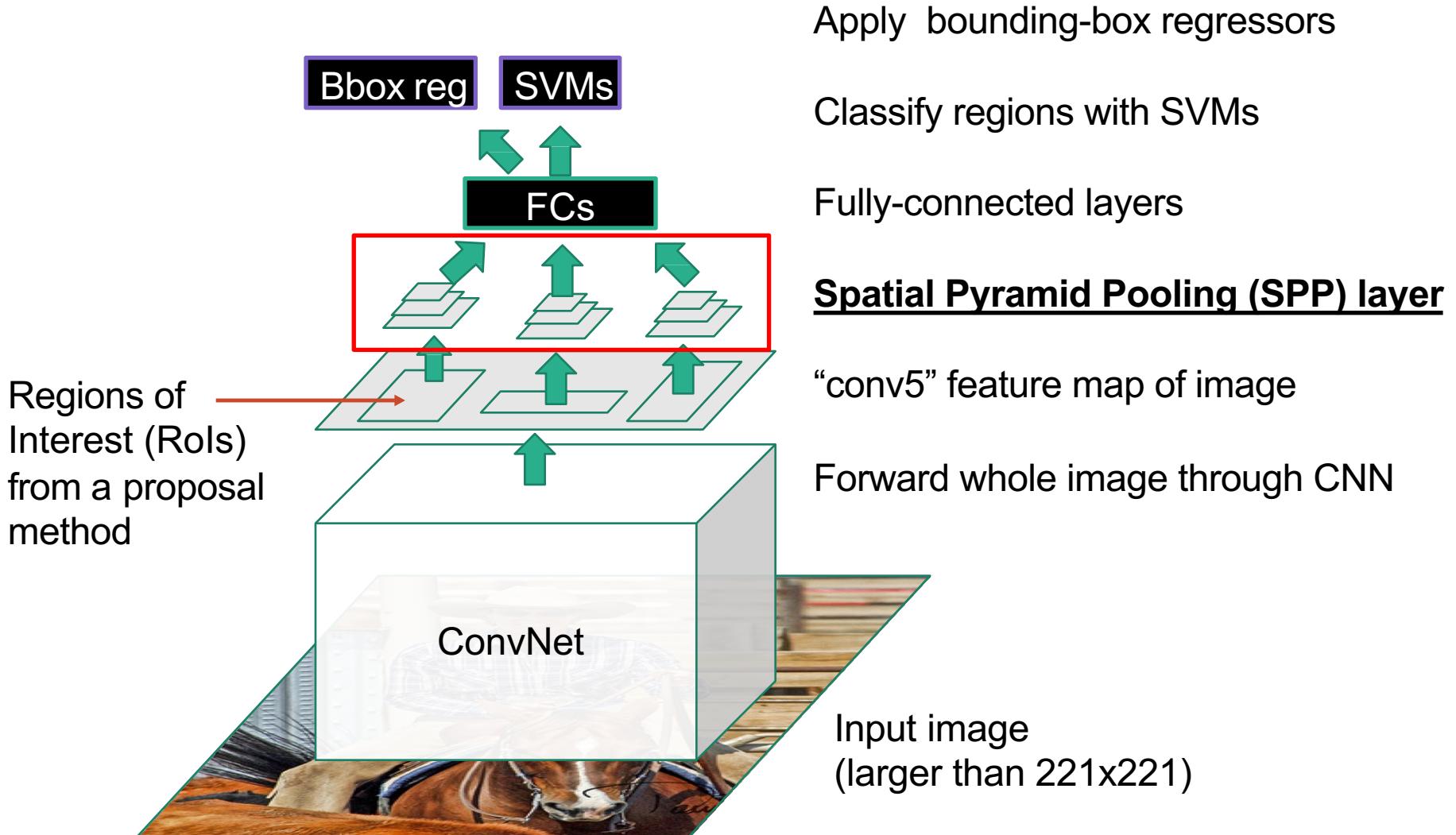
Locally orderless representation at several levels of resolution



# R-CNN



# SPPNet



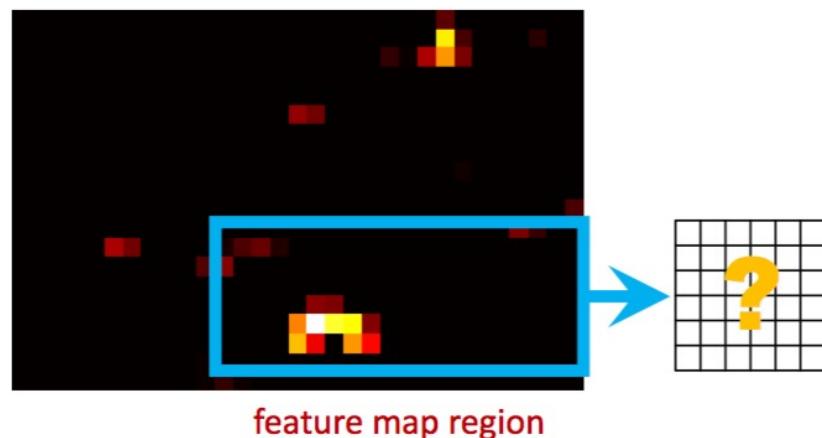
# Spatial Pyramid Pooling Layer

Replace last pooling layer with a SPP layer

- Orthogonal to any CNN designs

Input: conv feature map (e.g. conv5\_3 in VGG)

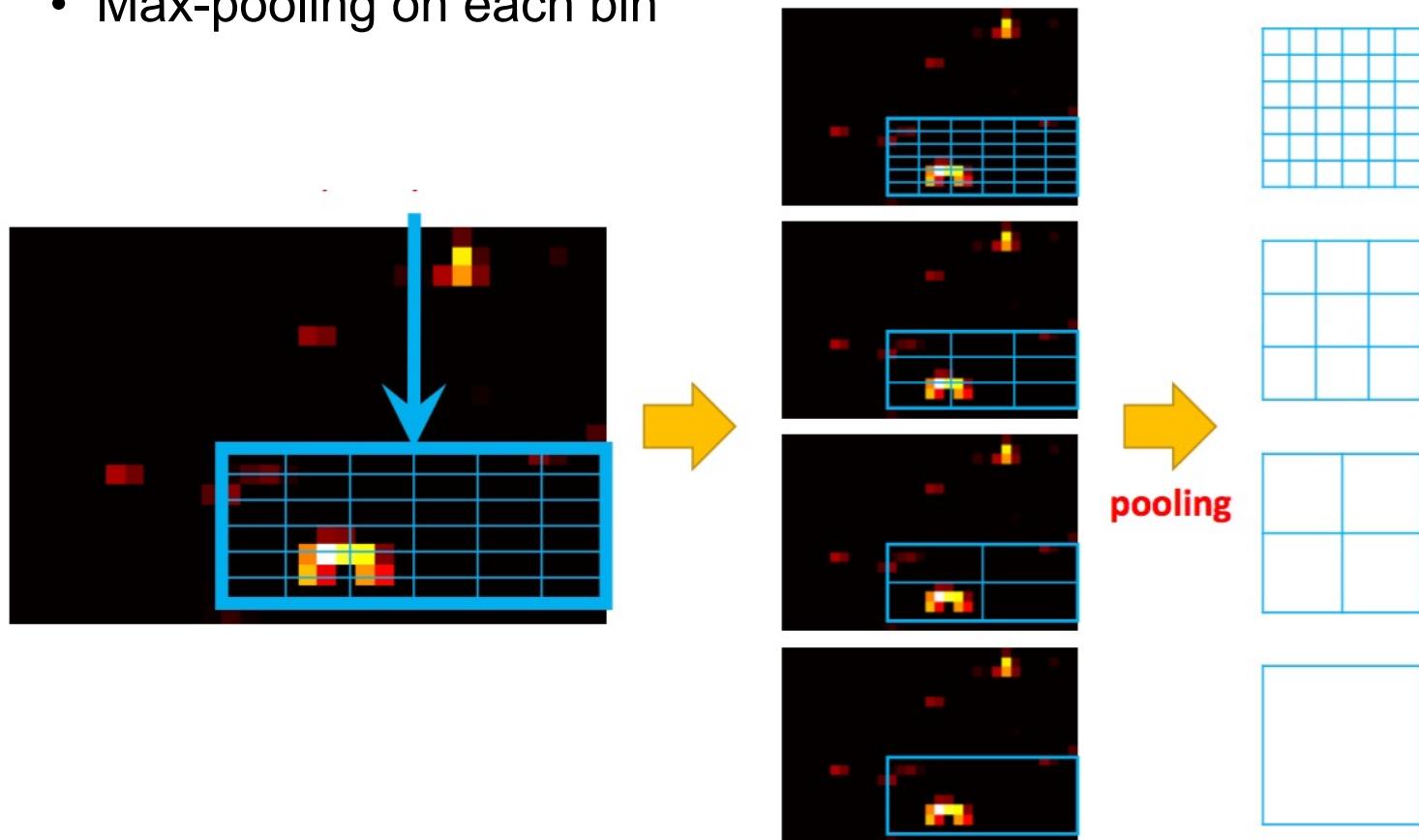
Output: corresponding feature map for a specific region of interest (RoI)



# Spatial Pyramid Pooling Layer

Build a pyramid of feature pools with fixed levels and sizes

- 4-level SP ( $1 \times 1$ ,  $2 \times 2$ ,  $3 \times 3$ ,  $6 \times 6$ , totally 50 bins)
- Max-pooling on each bin

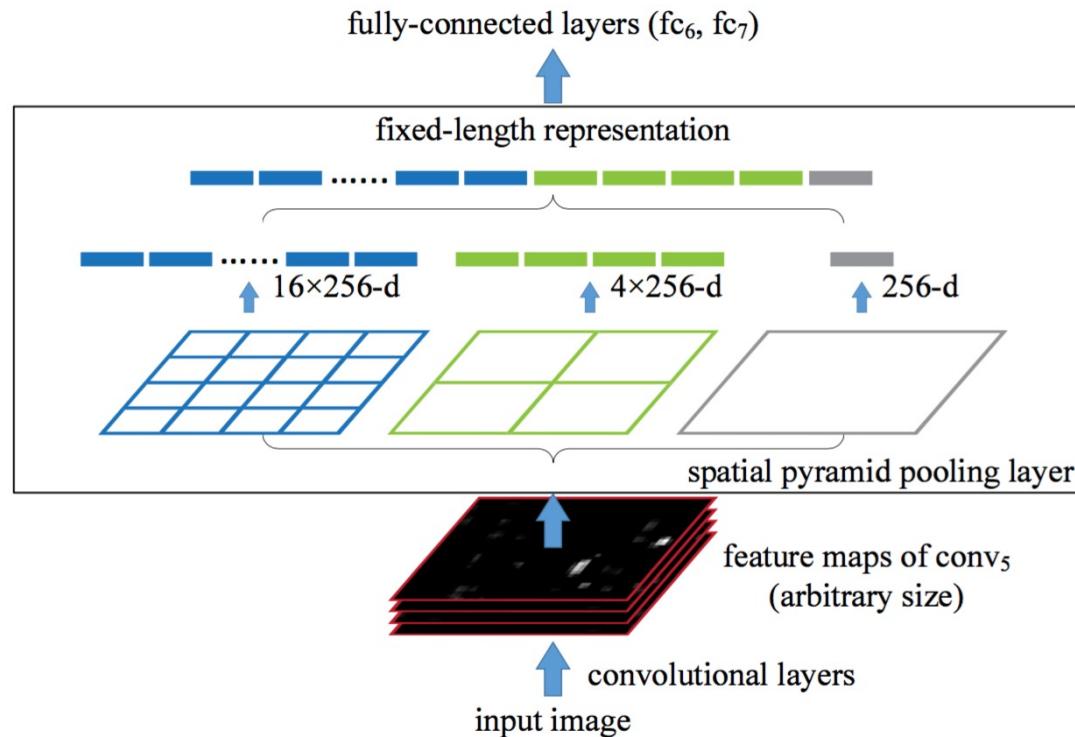


# Spatial Pyramid Pooling Layer

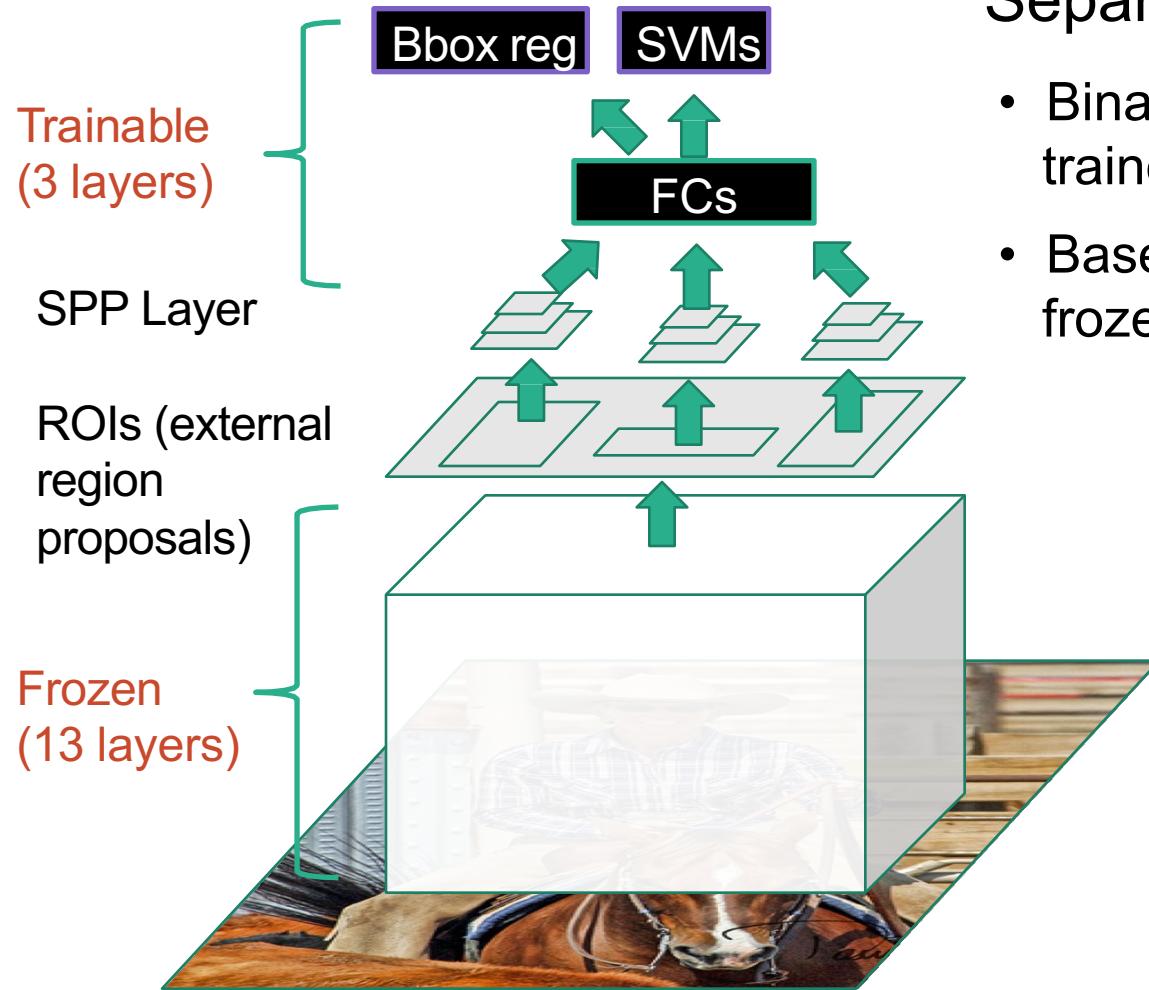
For each pyramid layer, max-pool the responses of each filter, and concatenate them to yield a fixed-length vector

The dimension : (# of conv filters)  $\times$  (# of total bins)

- e.g.  $256 \times (1^2 + 2^2 + 3^2 + 6^2) = 256 \times 50$



# SPPNet



## Separate fine-tuning

- Binary SVM classifiers are trained on the features
- Base CNN are trained and frozen

# Detection Implementation

## Fast mode of selective search

- Generate about 1K candidate windows per image
- Resize the image such that min dim = {480,576,688,864,1200}
- Run CNNs (e.g. ZFNet or VGGNet)

## Multi-scale feature extraction

- Choose a single scale such that the scaled window has similar # of pixels to 224x224

# SPPNet: Pros and Cons

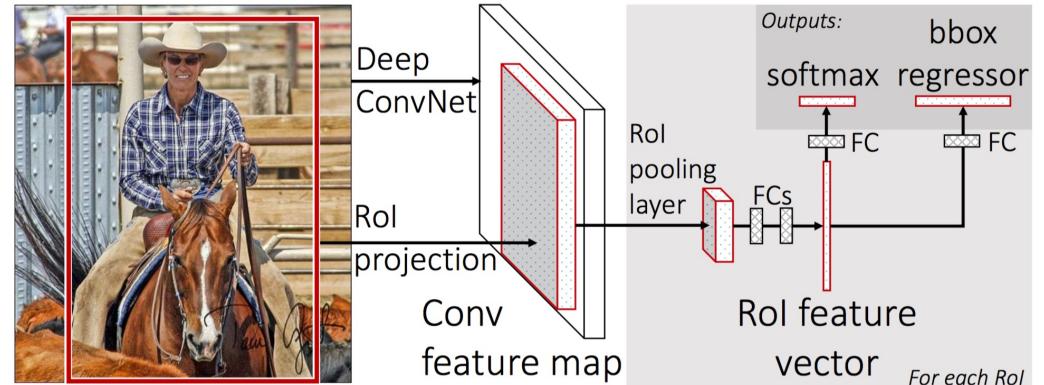
## The good

- There is only one CNN feed-forward pass per image; make inference (detection) orders-of-magnitude fast
- FC-layers and output layers are applied regionwise; can take arbitrary sized input

## The bad

- SPP Pooling is done in a separate pipeline; training is still slow (like R-CNN)
- Gradient is blocked at the SPP layer; parameters below the SPP layer can't be fine-tuned





# FAST R-CNN

*Fast R-CNN (ICCV 2015)*

Ross Girshick

<https://github.com/rbgirshick/fast-rcnn>

# Fast R-CNN

Share many features with SPPNet

Key Contribution: The RoI Pooling Layer

- A (simplified) single-level SPP layer
- Makes the whole network trainable in a single stage!

Better mAP and fast training/detection!

- vs. R-CNN: 9x faster in training and 213x faster in test
- vs. SPPNet, 3x faster in training and 10x faster in test

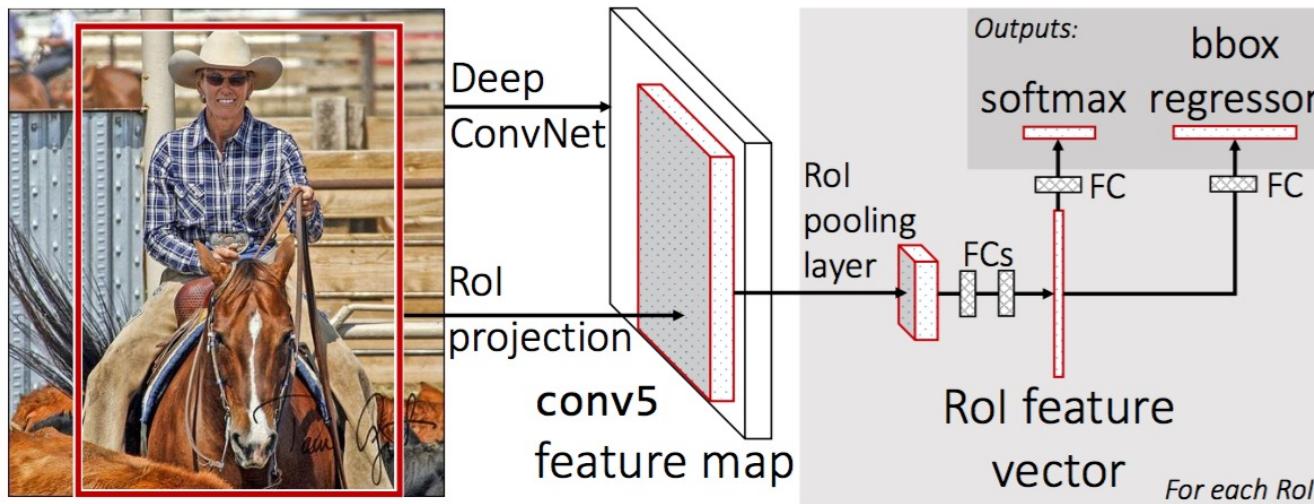
# Fast R-CNN Architecture

Input image and multiple Rois are input into Deep ConvNet

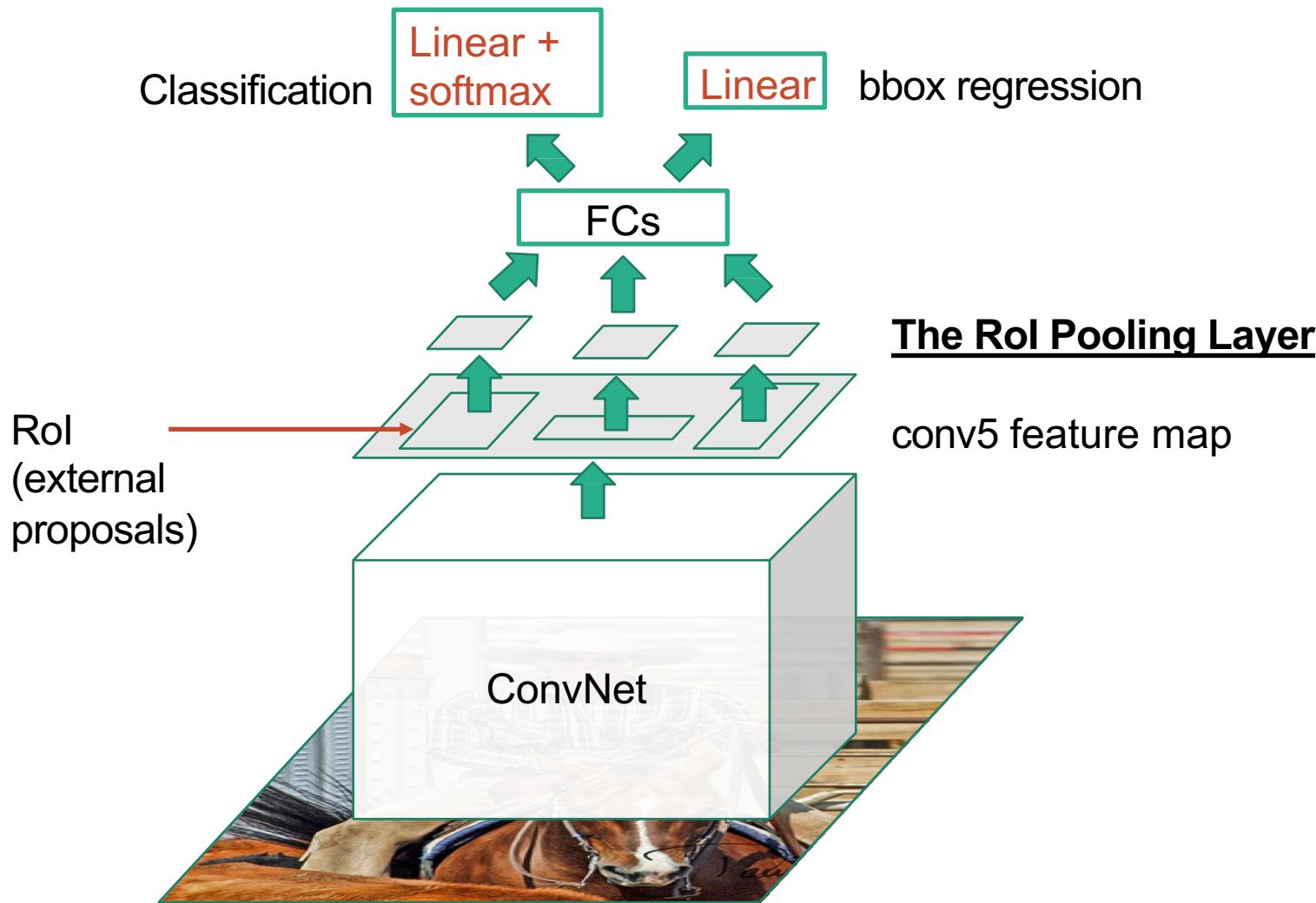
- Each ROI is pooled into a fixed size feature vector

Two output vectors per ROI

- Softmax probability over  $K+1$  classes
- BB coordinates  $[r,c,h,w]$  (top-left corner + height/width) for each of  $K$  classes



# Fast R-CNN



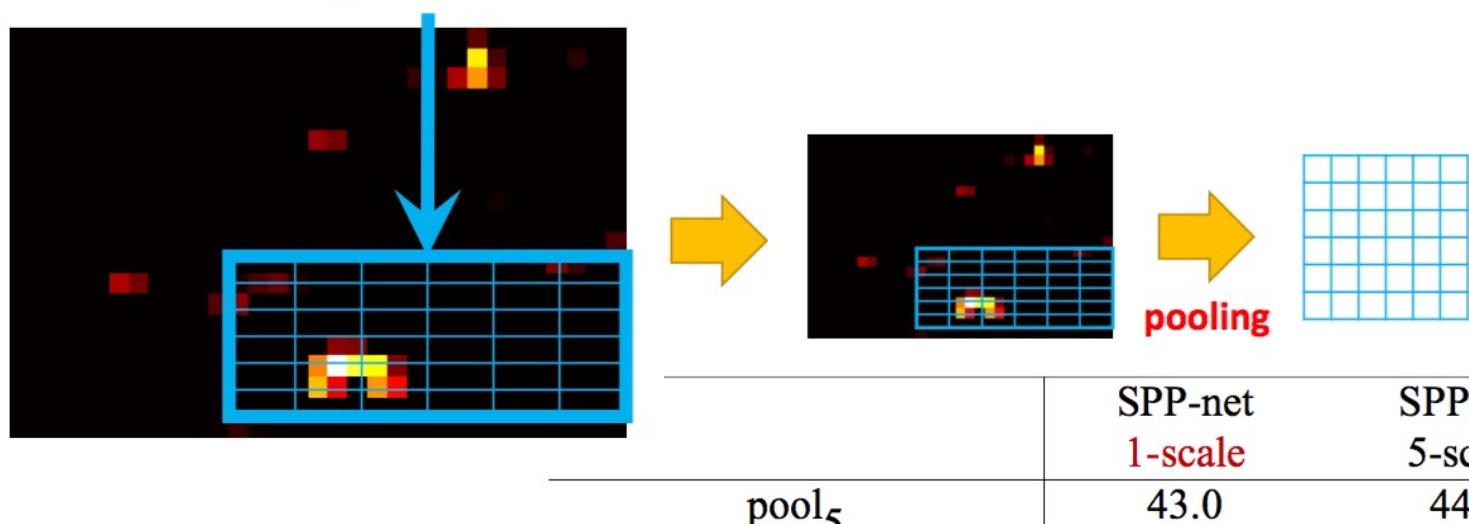
# The RoI Pooling Layer

RoI Pooling Layer = SPP layer with 1 pyramid level

- Feature map from deep convnets (e.g. VGG16) perform well **even at a single scale**

Layer input and output

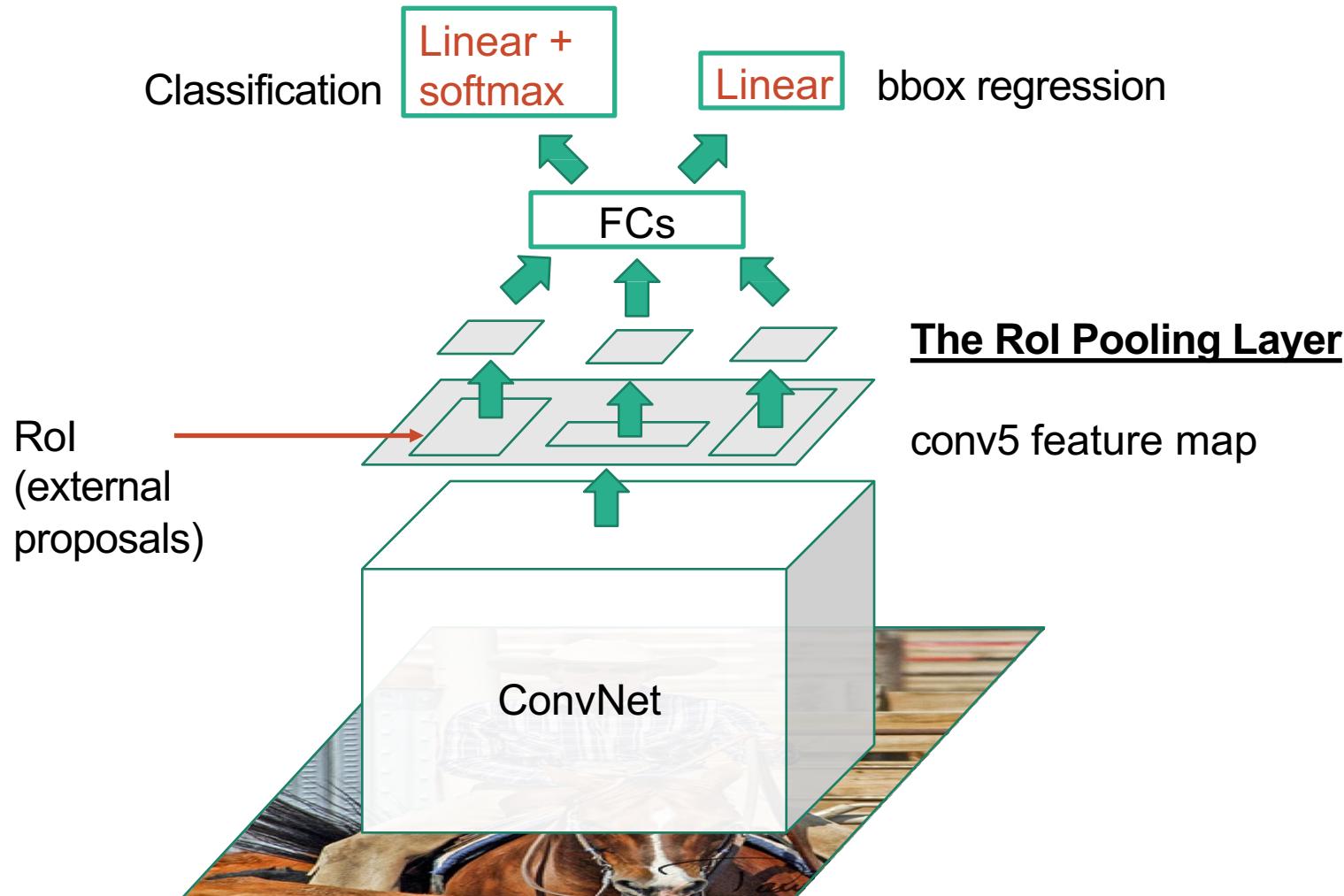
- (Shared) conv5 feature map:  $[H \times W \times 512]$  and  $R$  Rols
- Output:  $[R \times H' \times W' \times 512]$  (e.g.  $H' = W' = 7$ )



# Fast R-CNN: Training

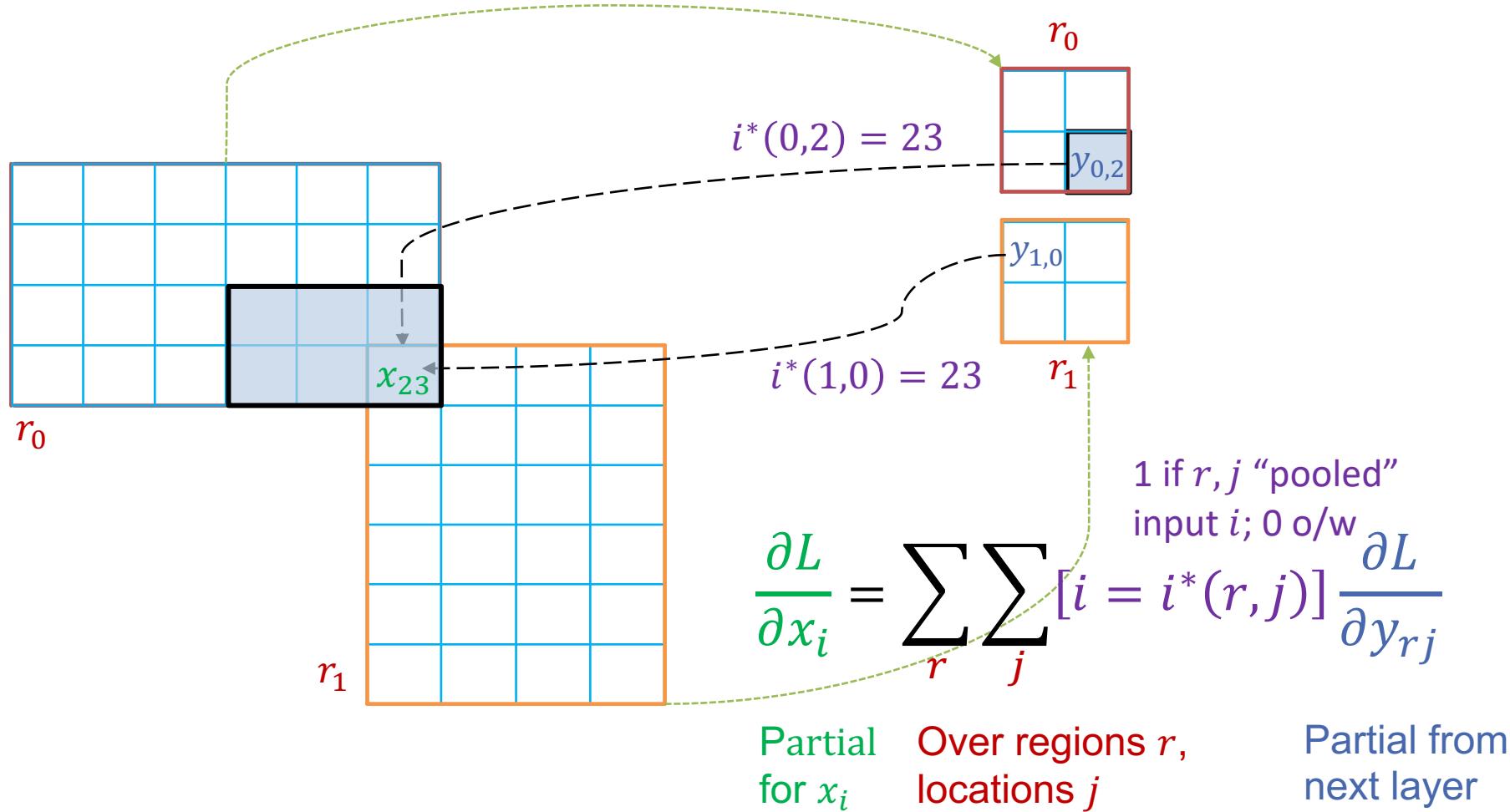
Network could be trained in a single training stage

- The ROI pooling layer operation should be differentiable



# 1. Backpropagation through ROI Pooling

Almost same as max-pooling (but regions may overlap)



## 2. Multi-task Loss

Multi-task loss: classification and bbox regression  
(per ROI)

- The ROI pooling layer operation should be differentiable

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

- $L_{cls}(p, u) = -\log p_u$ : classification cross-entropy loss
- $L_{loc}(t, t^*)$ : sum of smooth-L1 loss for regression
- $u$ : true class,  $p$ : class probability
- $t^u$ : bbox of true class  $v$ : predicted bbox,

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

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

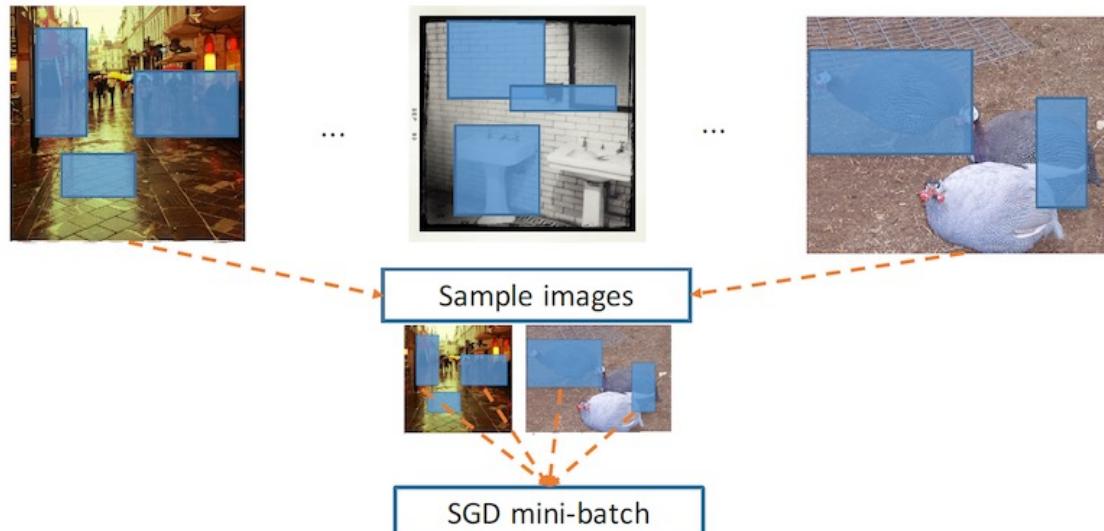
### 3. Hierarchical Sampling to Build mini-batch

If a mini-batch includes Rols from many different training images, SGD steps are inefficient

- e.g. What if we sample 128 Rols uniformly at random?

Hierarchical sampling

- Sample a small number of images (e.g. N=2)
- Then sample many examples from each image (e.g. R/N = 64)



### 3. Hierarchical Sampling to Build mini-batch

Pros: Share computation between overlapping examples from the same image

Cons: Examples from the sample image may be too correlated

- Not seriously problematic in practice, but significantly reduce computation time

Better training time and testing time with better accuracy

- R-CNN / SPPNet / Fast R-CNN
- Training time: 84 hours / 25.5 hours / 8.75 hours
- VOC07 test mAP: 66.0% / 63.1% / 68.1%
- Test time per image: 47s / 2.3s / 0.32s

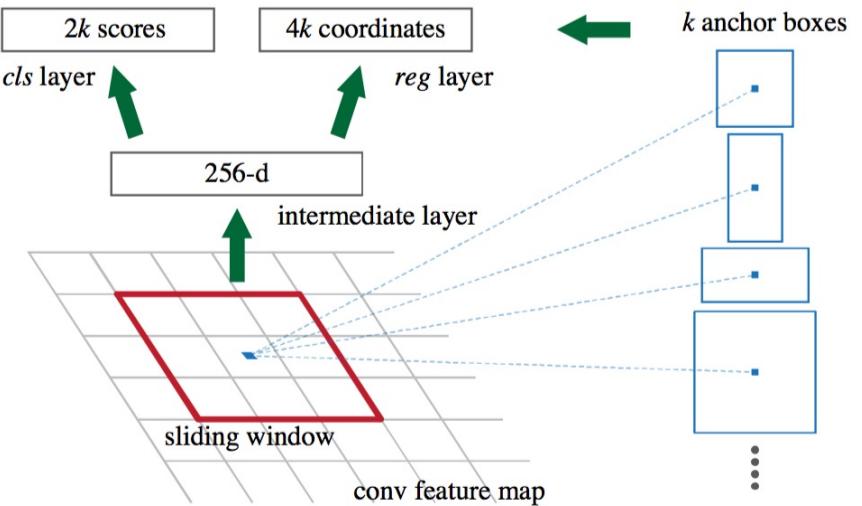
# Summary of Fast R-CNN

## The good

- Inference is single-forward, and fast (~160x faster than R-CNN)
- Can handle multi-scale input sizes

## The bad

- Still depends on the external region proposals (which is eliminated by Faster R-CNN!)



# FASTER R-CNN

*Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (NIPS 2015)*

Shaoqing Ren and Kaiming He and Ross Girshick and Jian Sun

<https://github.com/rbgirshick/py-faster-rcnn>

# Faster R-CNN

Eliminates the only drawback of Fast R-CNN:  
region proposals are jointly done in a single network

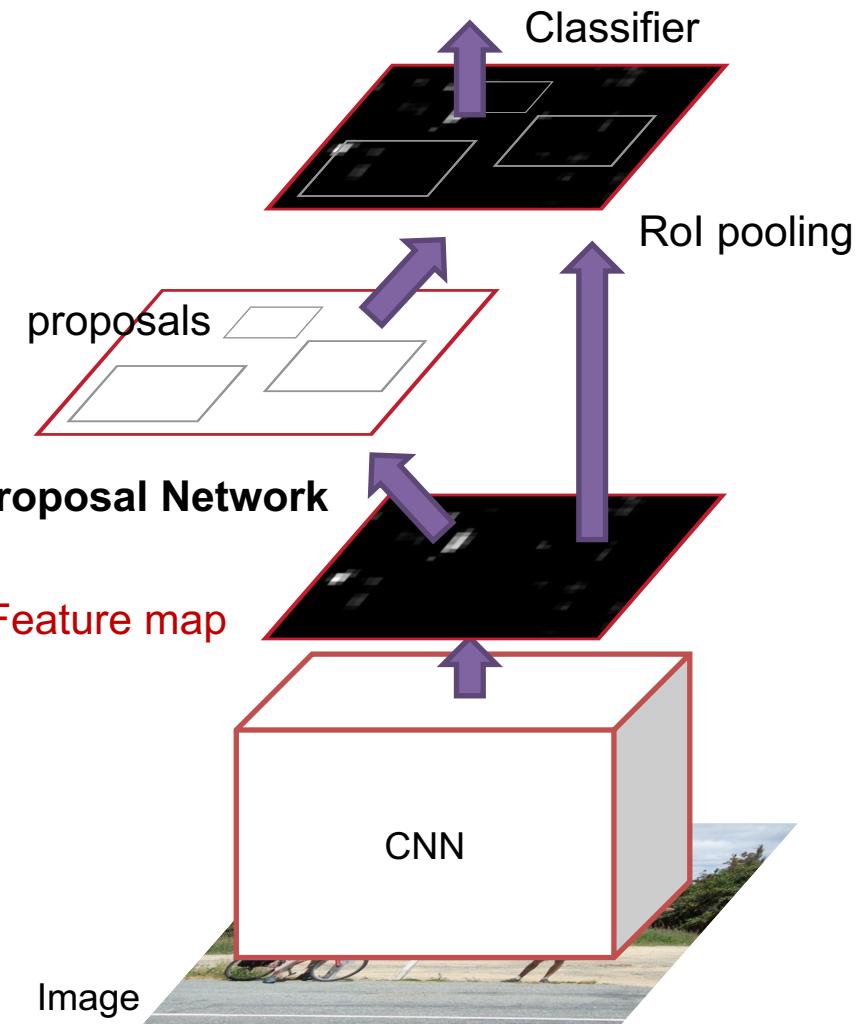
RPN (Region Proposal Network): Let the CNN do region proposals too!!

Faster R-CNN = Fast R-CNN + RPN

# Faster R-CNN

Insert a RPN after  
the last convolutional layer

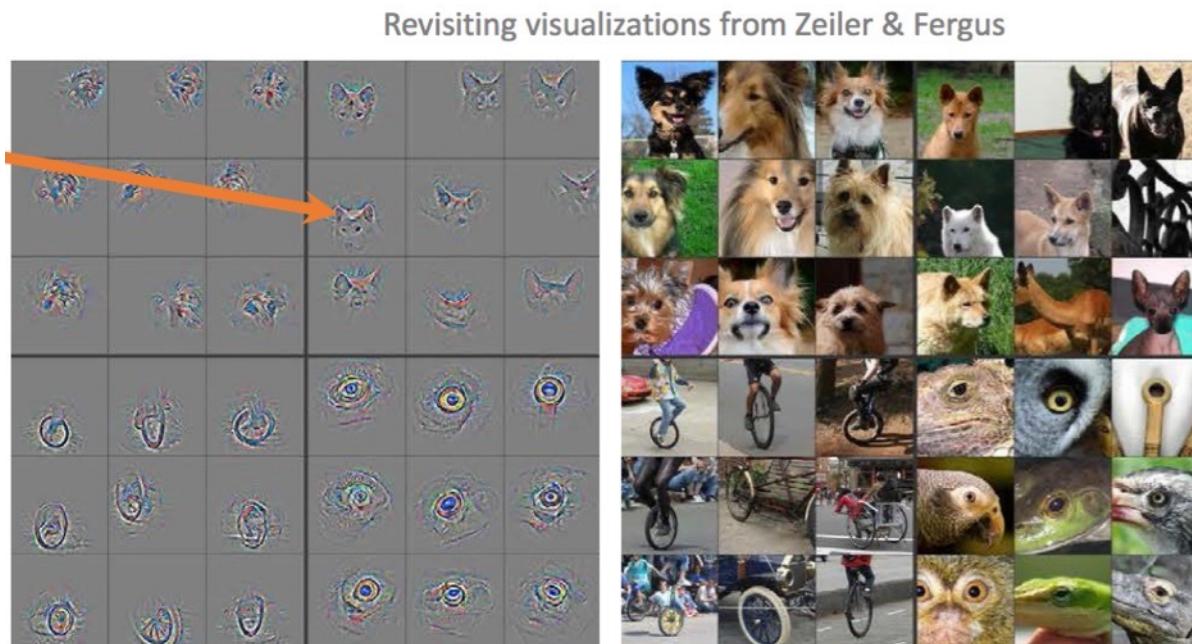
The proposed regions are fed  
into the RoI Pooling Layer  
(same as Fast R-CNN)



# Region Proposal from Feature Maps

Intuition from visualizations from (Zeiler & Fergus)

- Convolutional feature map contains an encoded fine-grained localization information!



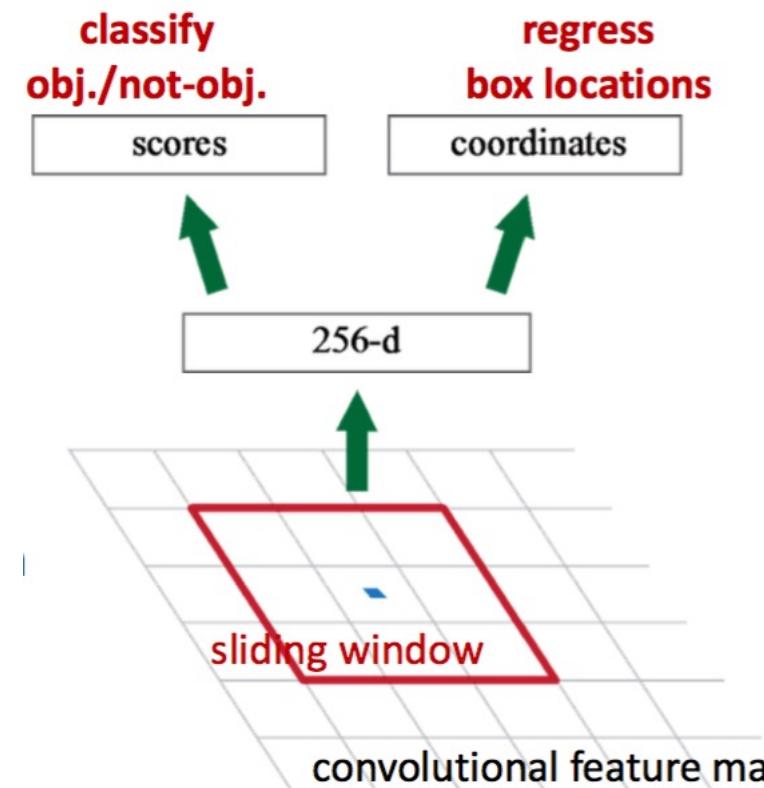
# Region Proposal Network

Maintain a sliding-window on the conv feature map, and build a small prediction network for

- Classifying object or not-object
- Bounding-box regressions

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

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



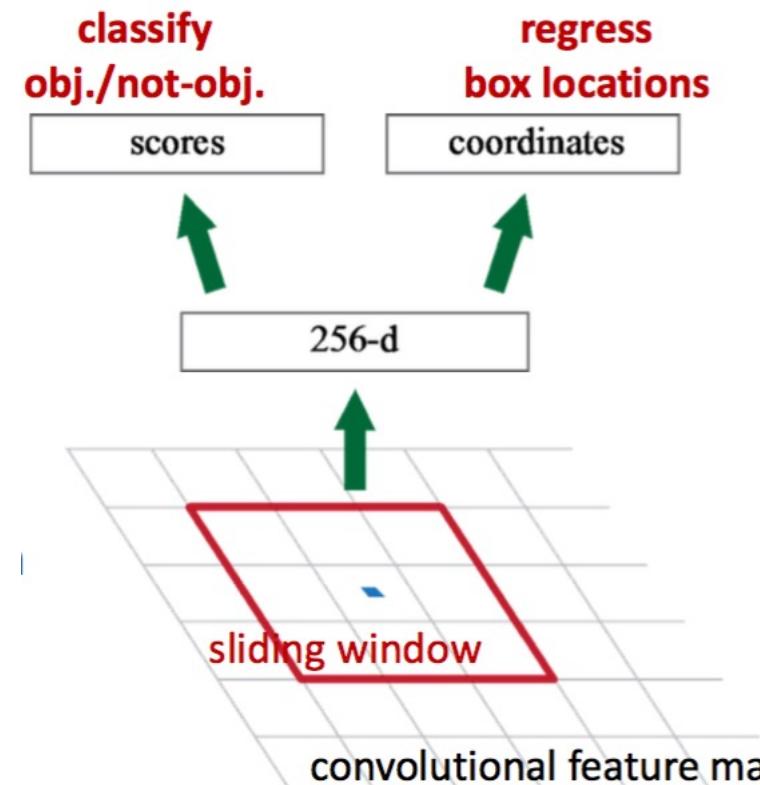
# Region Proposal Network

Use  $N$  anchors (pre-defined reference boxes) at each sliding window location

Bounding box regression gives offsets from these anchors

Object scores: regress to 1.0 (positive) or 0.0 (negative)

- Positive:  $\text{IoU} > 0.7$
- Negative:  $\text{IoU} < 0.3$



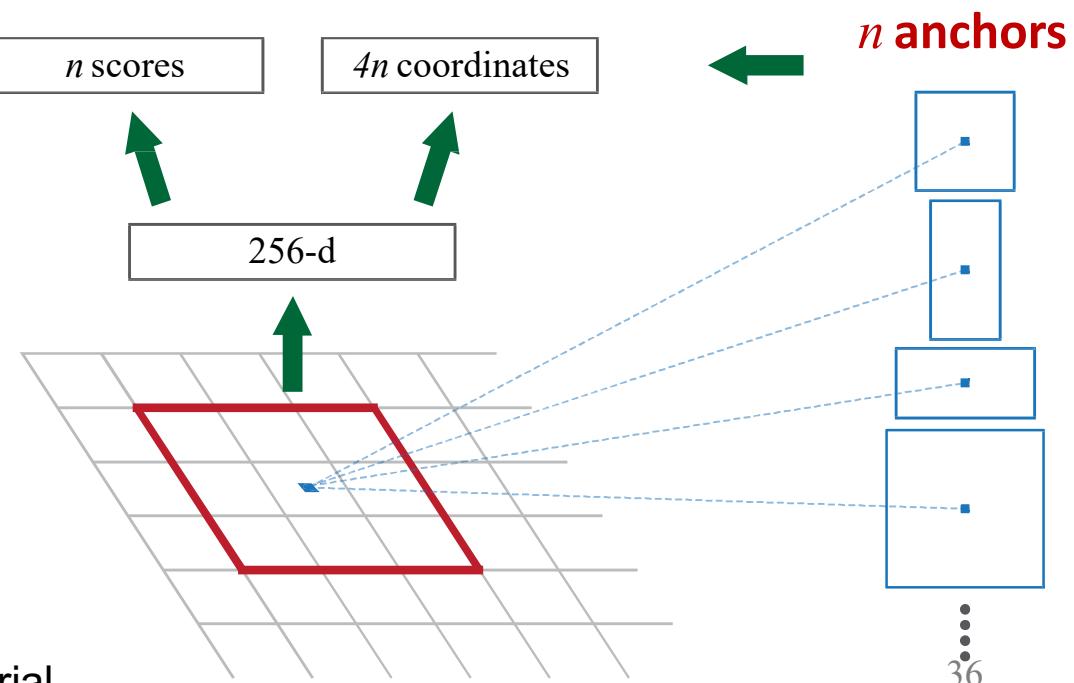
# Region Proposal Network

Anchors: multiple anchors are used at each position

- e.g. 3 scales (128, 256, 512)  $\times$  3 aspect ratios (2:1, 1:1, 1:2)

Anchors are translation-invariant;  
the same set of anchors are used at each window location

The number of region  
proposals per image:  
 $W \times H \times n$  ( $\sim 2400 \times 9$ )



# RPN: Training

Two training methods

(#1) Alternating optimization (published in NIPS paper)

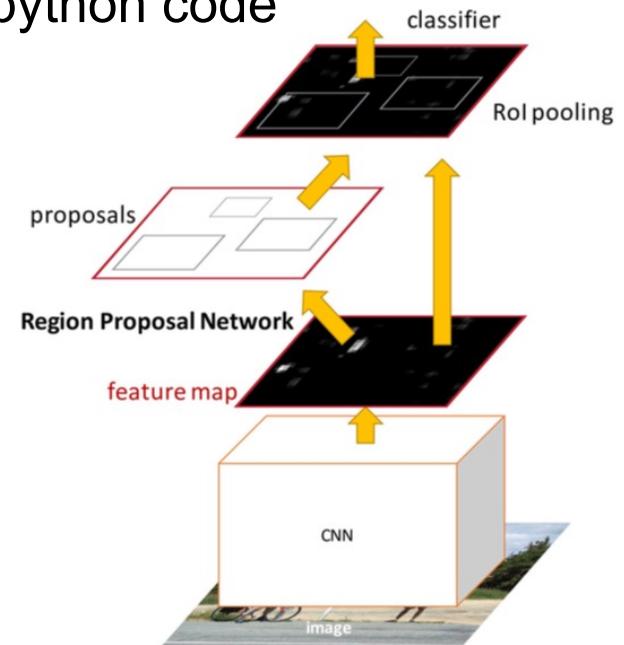
- Train RPN, FRCNN, RPN, FRCNN, and so on ... ?!

(#2) Approximate joint optimization (SGD)

- Later updated in PAMI paper and the python code
- Multi-task loss optimization

Goal:

- RPN and Fast-RCNN should share the ‘same convolutional layers’



# Performance!

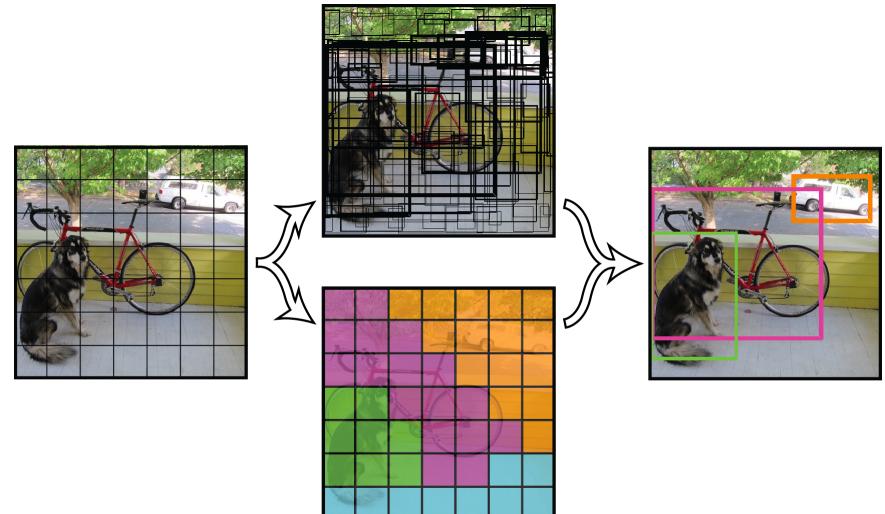
mAP ~ 66.9 (with VOC07) / mAP ~ 73.2 (with VOC07+12)

	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>66.9</b>

Current (2016) state-of-the-art is on the collaboration with RPN (Faster R-CNN) and ResNet-101 !

# Outline

- Two-stage Models
  - R-CNN (CVPR'14), SPPNet (ECCV'14), Fast R-CNN (ICCV'15), Faster R-CNN (NIPS'15)
- Single-stage Models
  - YOLO (CVPR' 16), YOLO v2/9000 (CVPR'17), SSD (ECCV' 16), DSSD (ArXiv'17)
- Application: OCR



# YOLO

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

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

<http://pjreddie.com/darknet/yolo/>

# YOLO (You Only Look Once)

A single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes

- A full image is processed in one evaluation in an end-to-end manner
- Common post-processing steps such as non-maximum suppression are unnecessary
- Framed as a single regression problem: from image pixels to bbox coordinates and class probabilities

Extremely fast: 45 frames per second on Titan X GPU

- Faster version: 155 frames per second
- Performance is not as good as state-of-the-art detector

# YOLO (You Only Look Once)

Divides the input image into a SxS grid (here, S=7)

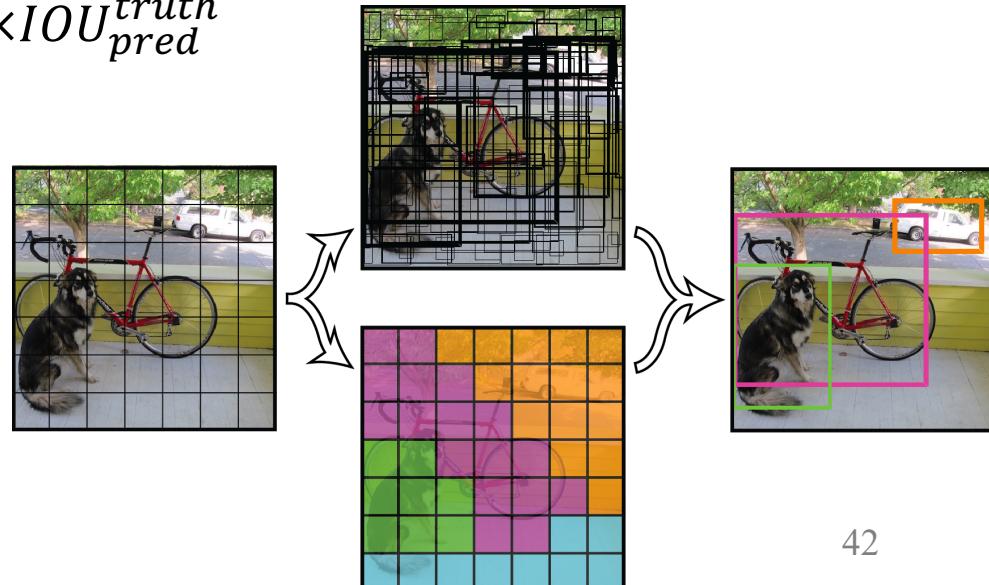
- If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object

Each grid cell predicts (1) B bboxes, (2) C conditional class probabilities

- Each bbox consists of 5 predictions: x, y, w, h, and confidence
- Confidence =  $P(\text{Object}) \times \text{IOU}_{\text{pred}}^{\text{truth}}$
- CCP =  $P(\text{Class}_i | \text{Object})$

Prediction is encoded a SxSx(Bx5+C) tensor

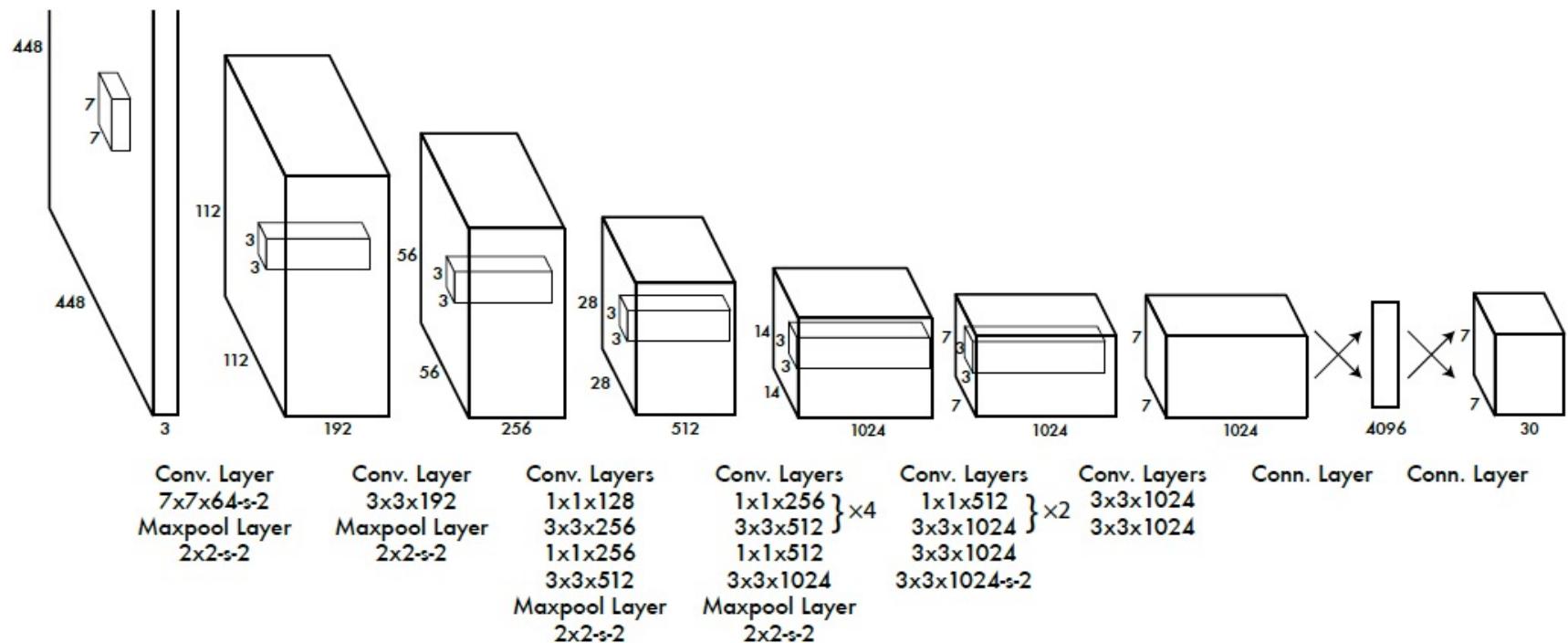
- e.g. PASCAL VOC:  
 $7 \times 7 \times (2 \times 5 + 20)$   
 $= 7 \times 7 \times 30$  tensor



# Network Design

Based on GoogLeNet model

- 24 conv layers + 2 FC layers
- Faster version: 9 conv layers



# Loss Function

Multi-part loss function

j-th bbox predictor in cell i is  
responsible for that prediction

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

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

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

Objectness confidence penalty

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

Objectness confidence penalty  
(for recall)

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

Class prediction  
penalty

# Performance

Fast YOLO is the fastest detector on PASCAL

Significantly better than other real-time systems, but not as good as the state-of-the-art methods

- Faster RCNN with the best accuracy: 7 fps

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45

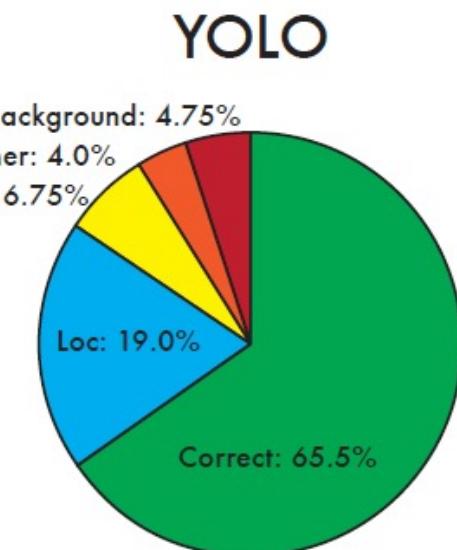
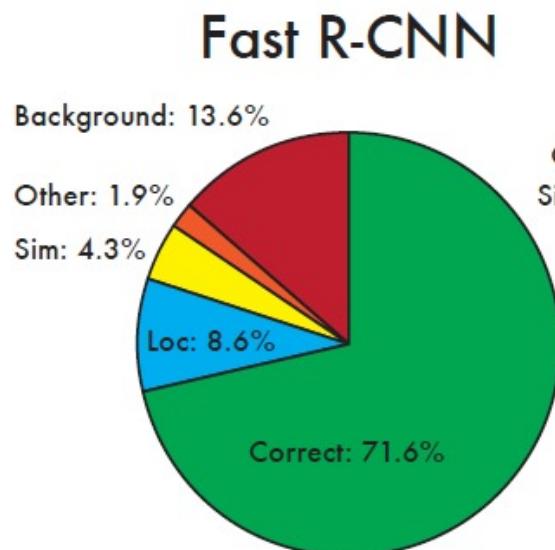
  

Less Than Real-Time	Train	mAP	FPS
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

# Performance

YOLO struggles to localize object correctly

R-CNN makes many false positives



	mAP	Combined	Gain
Fast R-CNN	-	71.8	-
Fast R-CNN (2007 data)	<b>66.9</b>	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
Ensemble of Fast R-CNN			
YOLO + Fast R-CNN	YOLO	63.4	<b>75.0</b> <i>3.2</i>

# YOLOV2/YOLO9000

*YOLO9000: Better, Faster, Stronger* (CVPR 2017)

Joseph Redmon and Ali Farhadi

<https://github.com/philipperemy/yolo-9000>

# YOLOv2: Better

A list of heuristics to make more accurate without scaling up the network

## 1. Batch normalization

- Able to remove dropout from the model without overfitting

## 2. High resolution classifier

- The original YOLO used 224x224 for pre-training and 448x448 for fine-tuning. YOLOv2 used 448x448 from the beginning

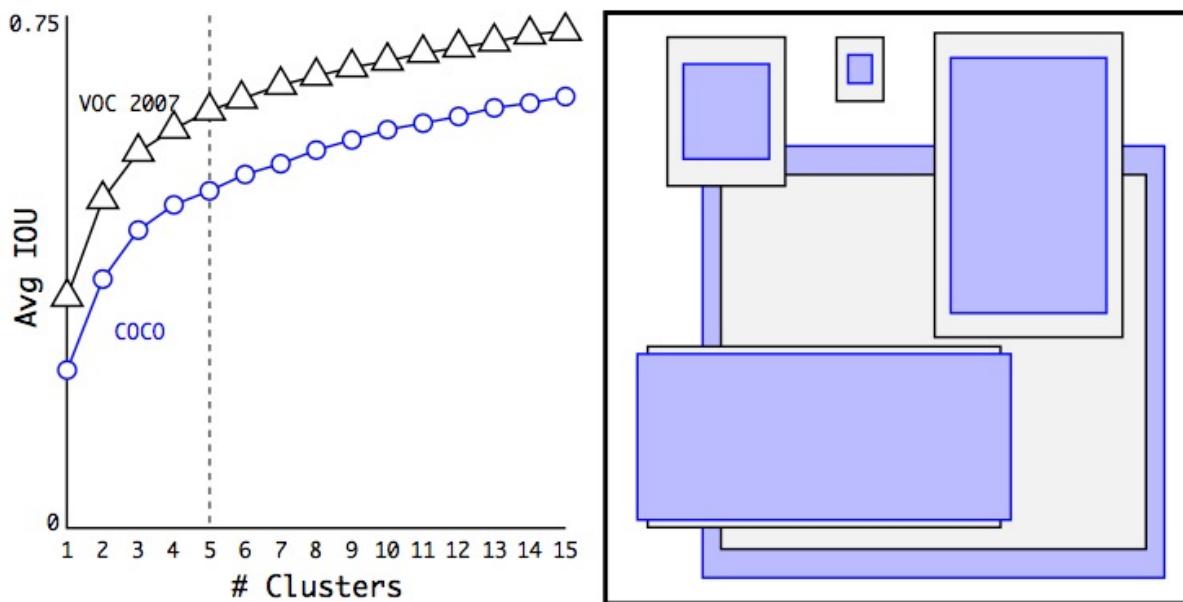
## 3. Convolutional with anchor boxes

- Remove the FC layers from YOLO and use anchor boxes to predict bounding boxes

# YOLOv2: Better

## 4. Dimension clusters

- In YOLO, anchor box dimensions are hand-picked
- Run the k-means clustering and find out k anchors (per cell of the SxS grid)



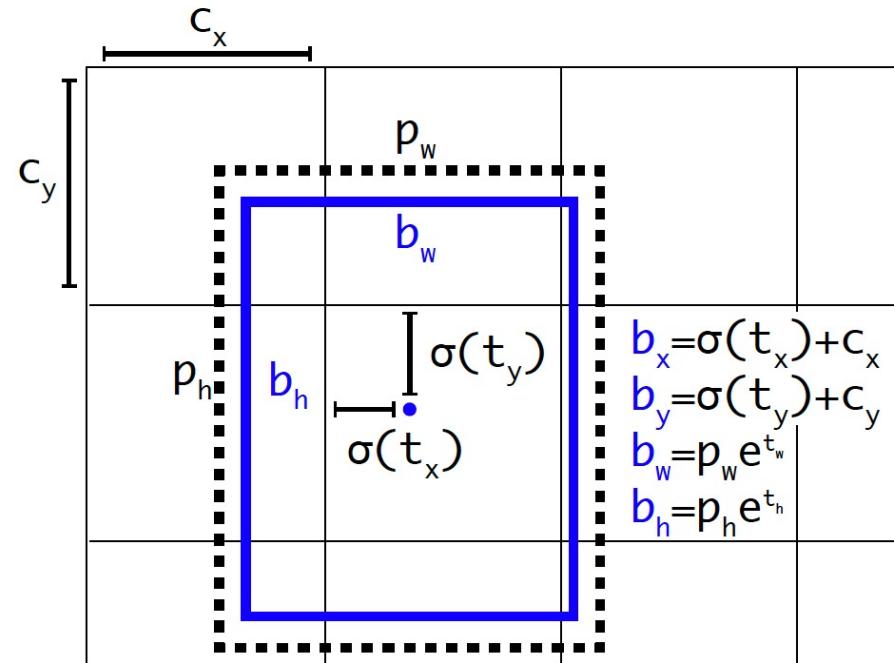
# YOLOv2: Better

## 5. Direct location prediction

- Related to model stability
- Predict 5 coordinates for each bbox:  $t_x, t_y, t_w, t_h, t_o$
- bbox prior by k-means:  $p_w, p_h$

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$

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



# YOLOv2: Better

## 6. Fine-grained features

- The original YOLO used only the last layer of feature map,
- YOLOv2 uses both conv16 (26x26x512) conv24 (13x13x1024)

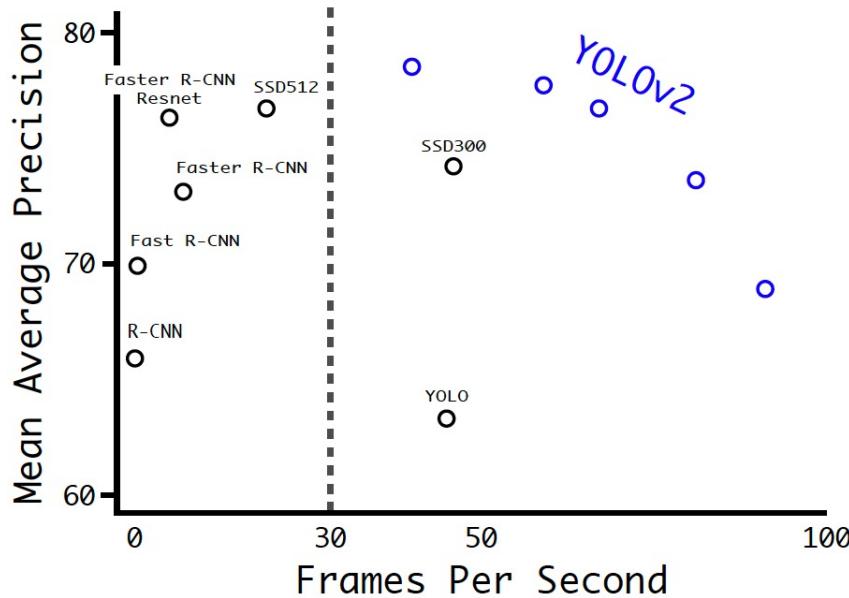
## 7. Multi-scale training

- The original YOLO used 448x448 input images
- Instead of fixing the input image size, randomly choose a new image dimension every 10 batches
- Pull from the following multiples of 32: {320, 352, ..., 608}, since the model downsamples by a factor of 32

# YOLOv2: Faster

YOLOv2 uses DarkNet-19

- YOLO uses GoogLeNet instead of VGG-16, although its accuracy is slightly worse than VGG-16
- 19 Conv + 5 Max-pooling



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	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax			

# YOLOv2: Results

## Path from YOLO to YOLOv2

	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		<b>78.6</b>	

# YOLOv2: Results

Tradeoff between speed and accuracy at different resolutions

- PASCAL VOC 2007

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

# YOLOv2: Results

## PASCAL VOC 2012 test detection

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

## COCO test-dev 2015

		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

# YOLOv2: Stronger

## Joint classification and detection

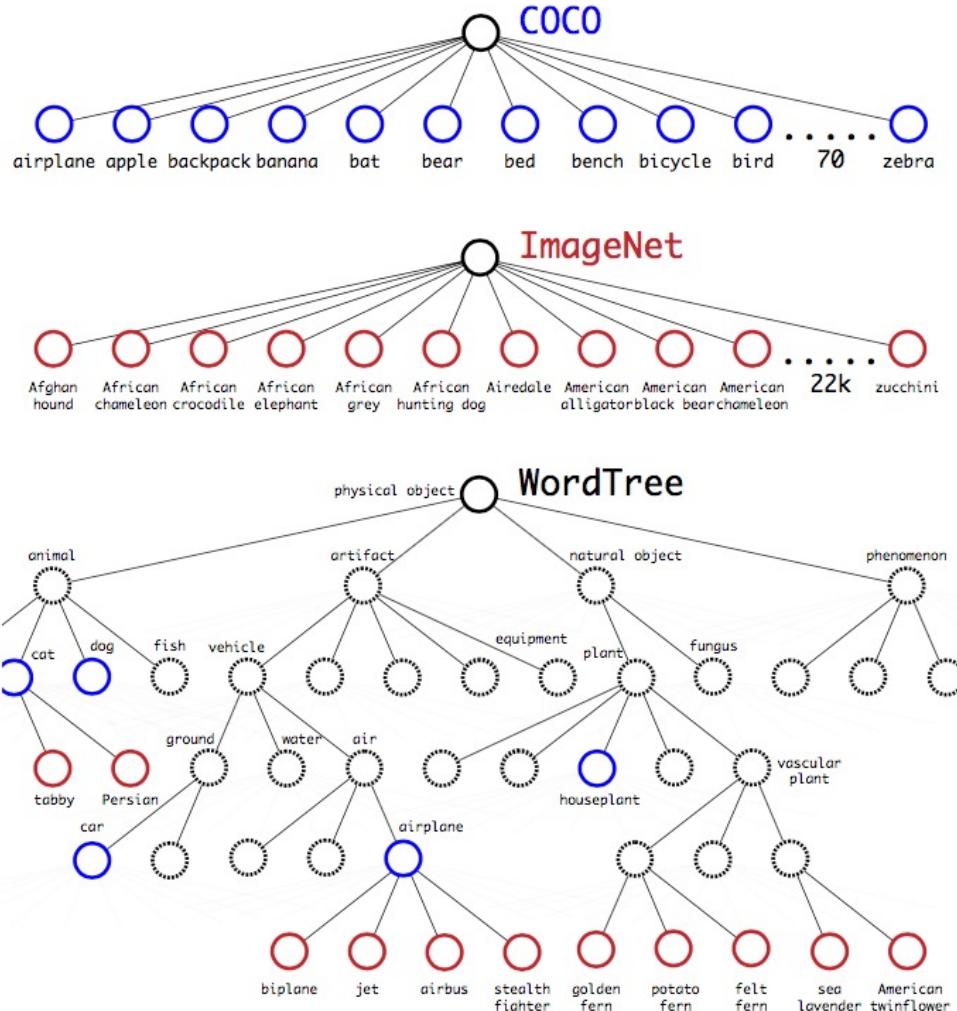
- Use labeled images for detection, to learn detection-specific information (e.g. bbox coordinates, objectness, common object prediction)
- Use labeled images for *classification*, to expand the number of categories it can detect

However, it is not straightforward ...

- Classification datasets have a much wider and deeper range of labels
- e.g. ImageNet has more than a hundred breeds of dog, such as *Norfolk terrier*, *Yorkshire terrier*, *Bedlington terrier*

# YOLOv2: Stronger

Build a WordTree (a hierarchical model of visual concepts)



- ImageNet labels are pulled from WordNet (a directed graph)
- Build a tree using some heuristics from WordNet
- Combine multiple datasets together

# YOLOv2: Stronger

## Hierarchical classification

- Perform multiple softmax operations over co-hyponyms
- Then we can compute the following conditional probabilities

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

$$Pr(\text{Yorkshire terrier}|\text{terrier})$$

$$Pr(\text{Bedlington terrier}|\text{terrier})$$

...

- We can also compute absolute probability for any node by simply following the path through the tree to the root

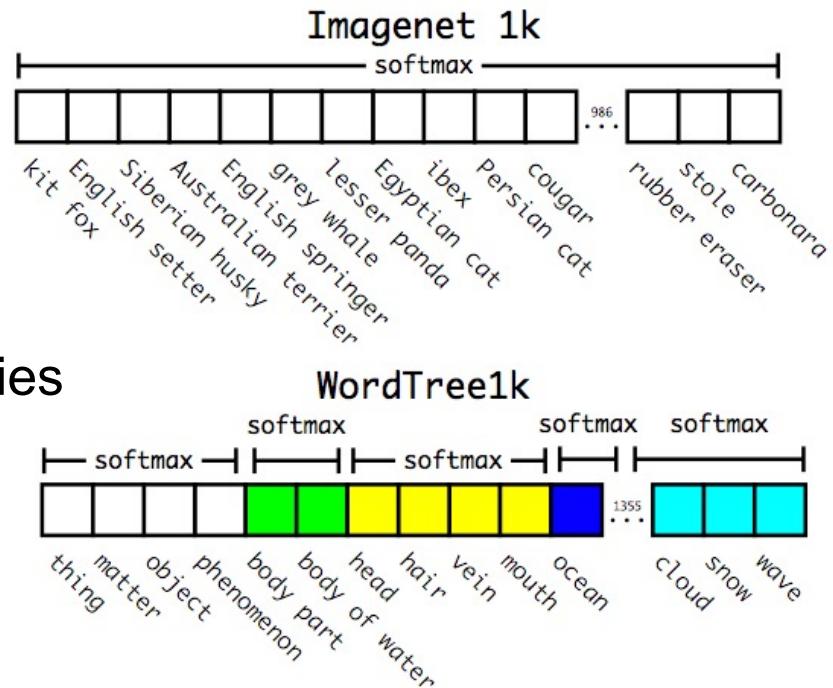
$$Pr(\text{Norfolk terrier}) = Pr(\text{Norfolk terrier}|\text{terrier})$$

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

\* . . \*

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

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



# YOLOv2: Stronger

## Joint classification and detection

- When the network sees a *detection* image we backpropagate loss as normal
- When it sees a *classification* image we only backpropagate classification loss
- To do this they simply find the bbox that predicts the highest probability for that class and we compute the loss on its predicted tree

## For ImageNet detection task

- YOLO9000 gets 19.7 mAP overall with 16.0 mAP on the disjoint 156 object classes that it has never seen any labelled detection data for

# YOLOv2: Demo



YOLO v2

<http://pureddie.com/yolo>