

# Recognition IV: Object Detection through Deep Learning and R-CNNs

Linda Shapiro

CSE 455

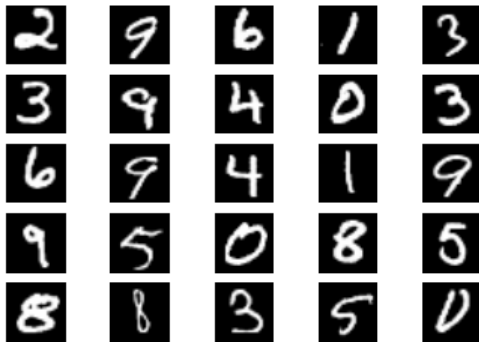
Most slides from Ross Girshick

# Outline

- Object detection
  - the task, evaluation, datasets
- Neural Net: how do they work?
- Convolutional Neural Networks (CNNs)
  - overview and history
- Region-based Convolutional Networks (R-CNNs)
- New Speedier R-CNNs

# Image classification

- $K$  classes
- Task: assign correct class label to the whole image

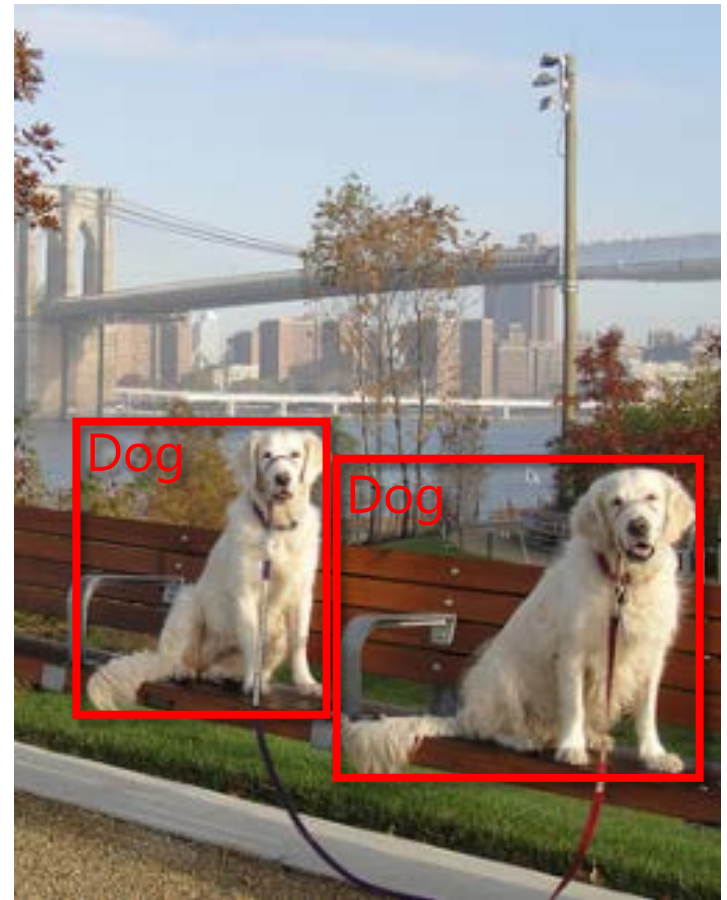


Digit classification (MNIST)



Object recognition (Caltech-101)

# Classification vs. Detection

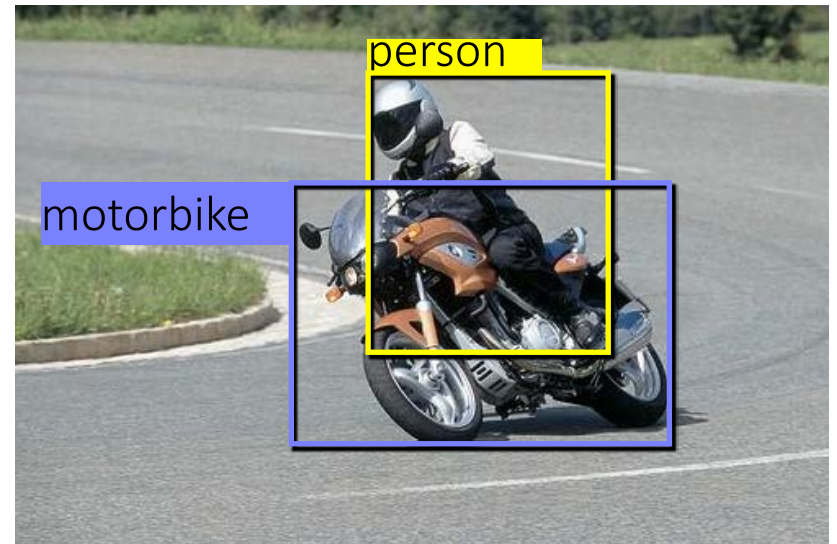


# Problem formulation

{ airplane, bird, motorbike, person, sofa }



Input



Desired output



# Evaluating a detector



Test image (previously unseen)

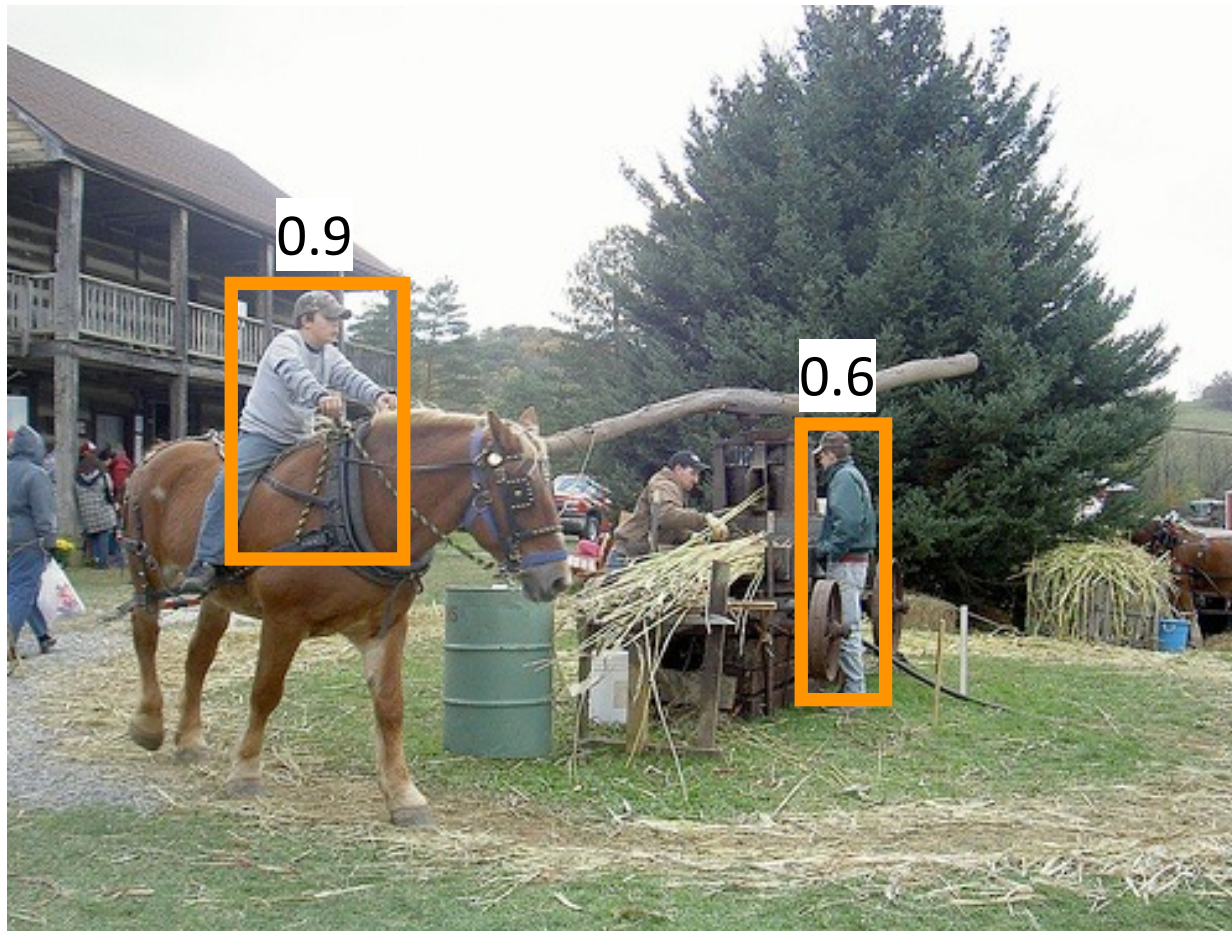
# First detection ...



 'person' detector predictions



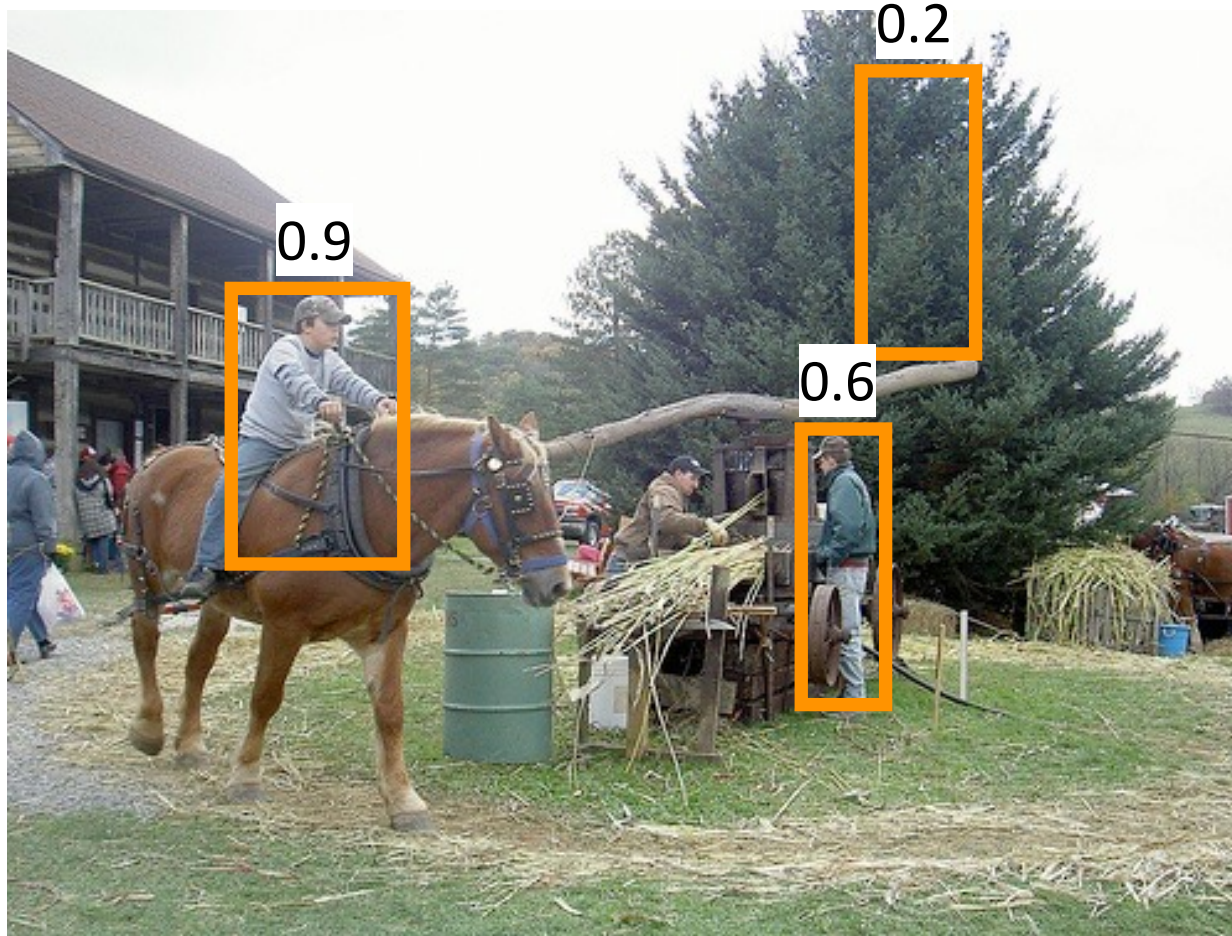
# Second detection ...



 'person' detector predictions

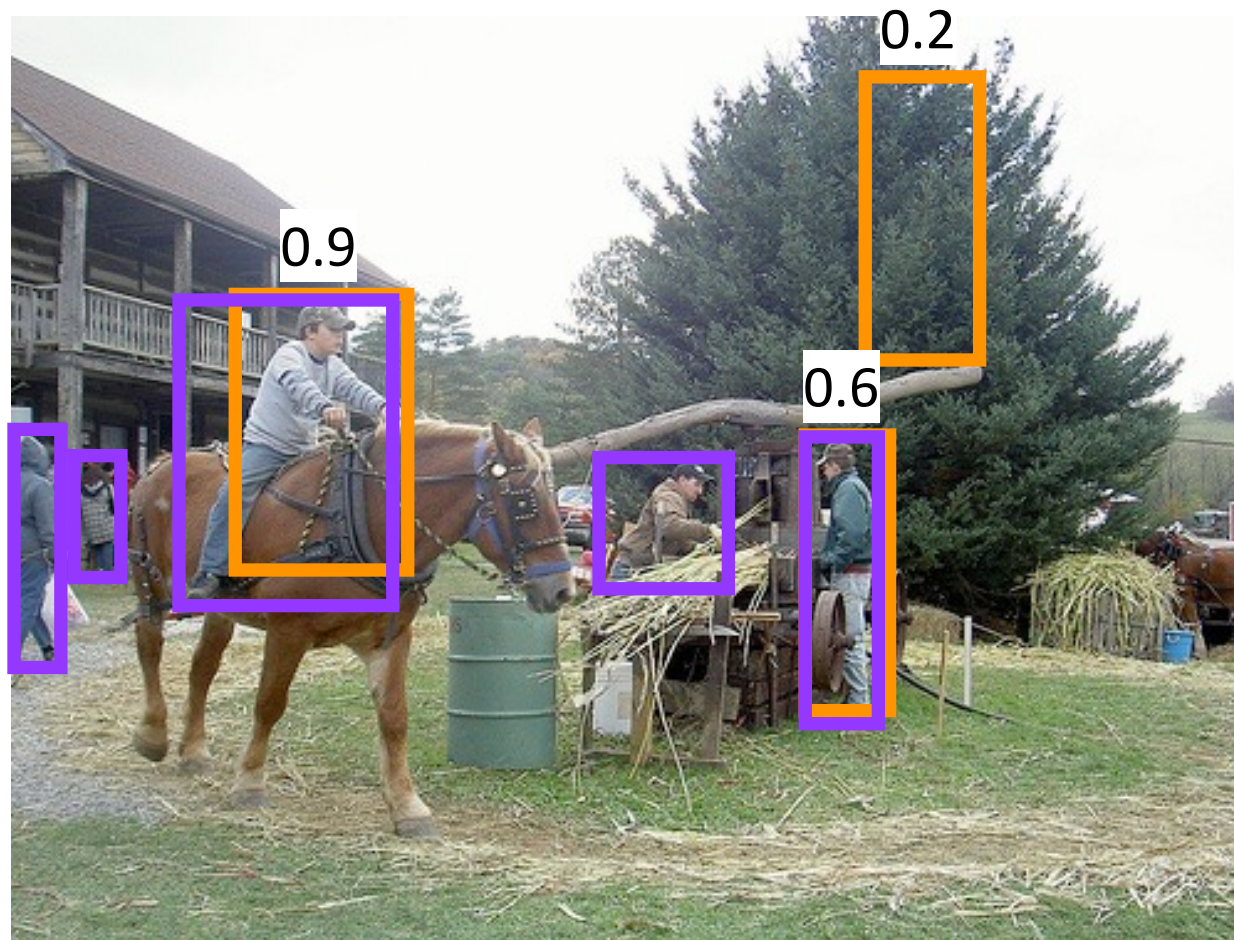




# Third detection ...



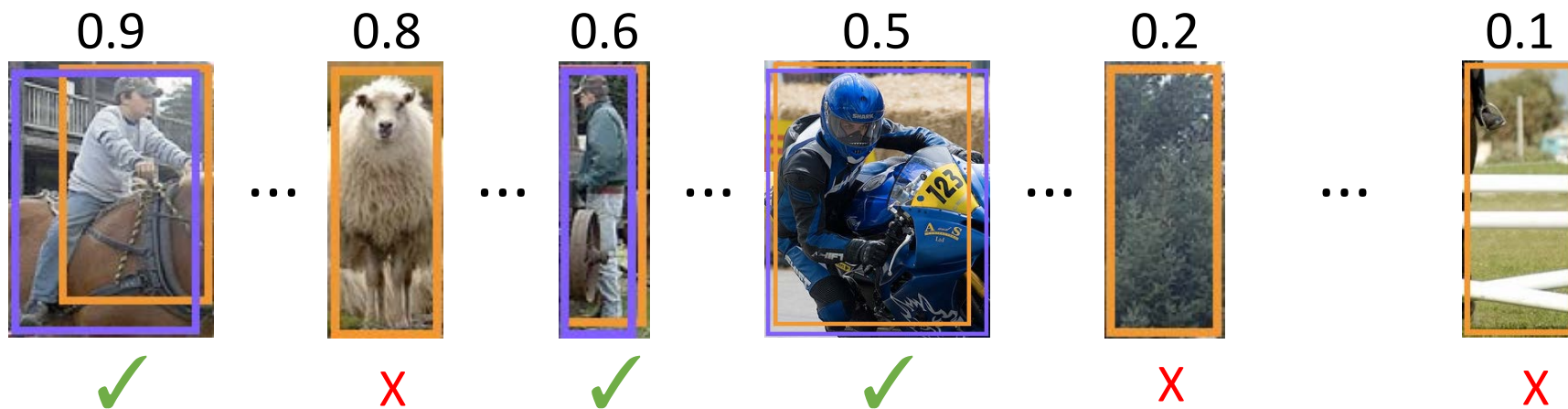
 'person' detector predictions

# Compare to ground truth



-  'person' detector predictions
-  ground truth 'person' boxes

# Sort by confidence

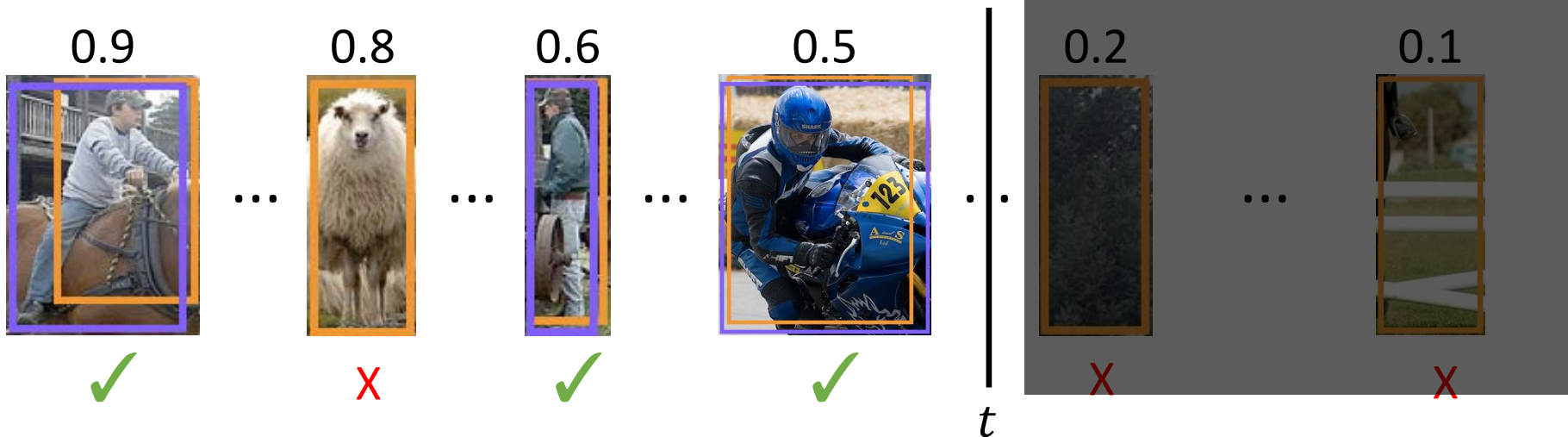


✓  
true  
positive  
(high overlap)

✗  
false  
positive  
(no overlap,  
low overlap, or  
duplicate)



# Evaluation metric

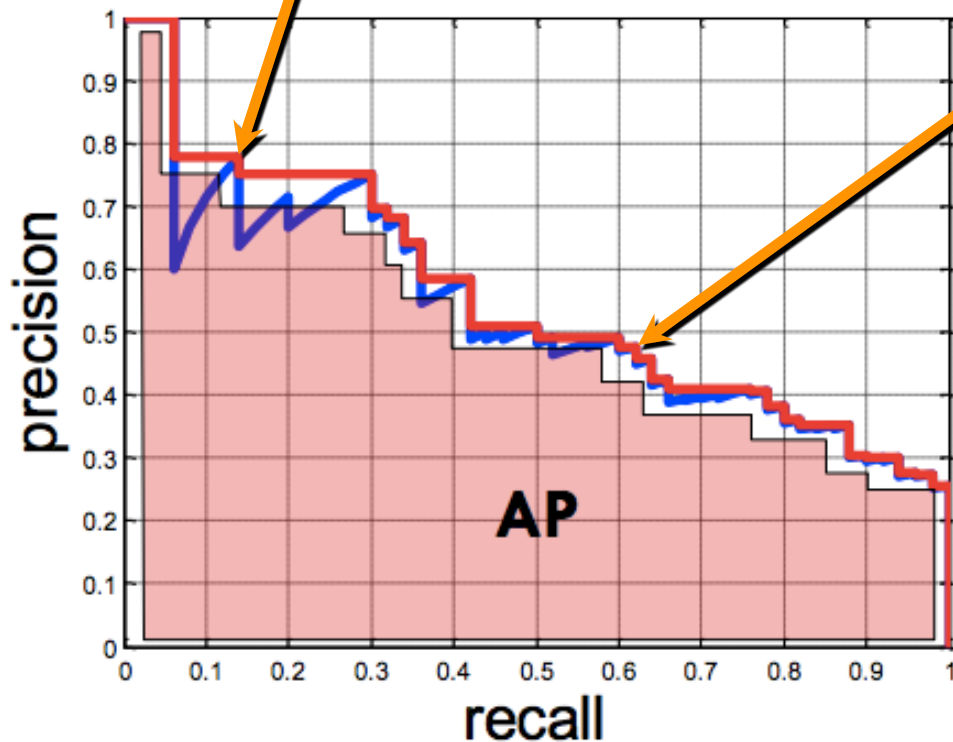
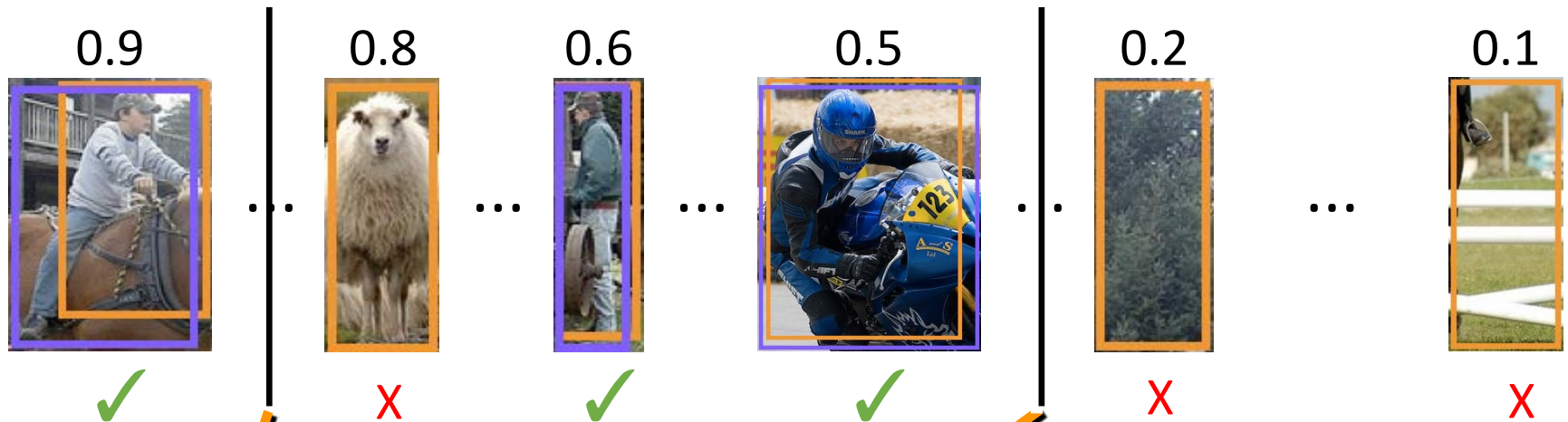


$$precision@t = \frac{\#true\ positives@t}{\#true\ positives@t + \#false\ positives@t} \quad \frac{\checkmark}{\checkmark + X}$$

$$recall@t = \frac{\#true\ positives@t}{\#ground\ truth\ objects}$$



# Evaluation metric



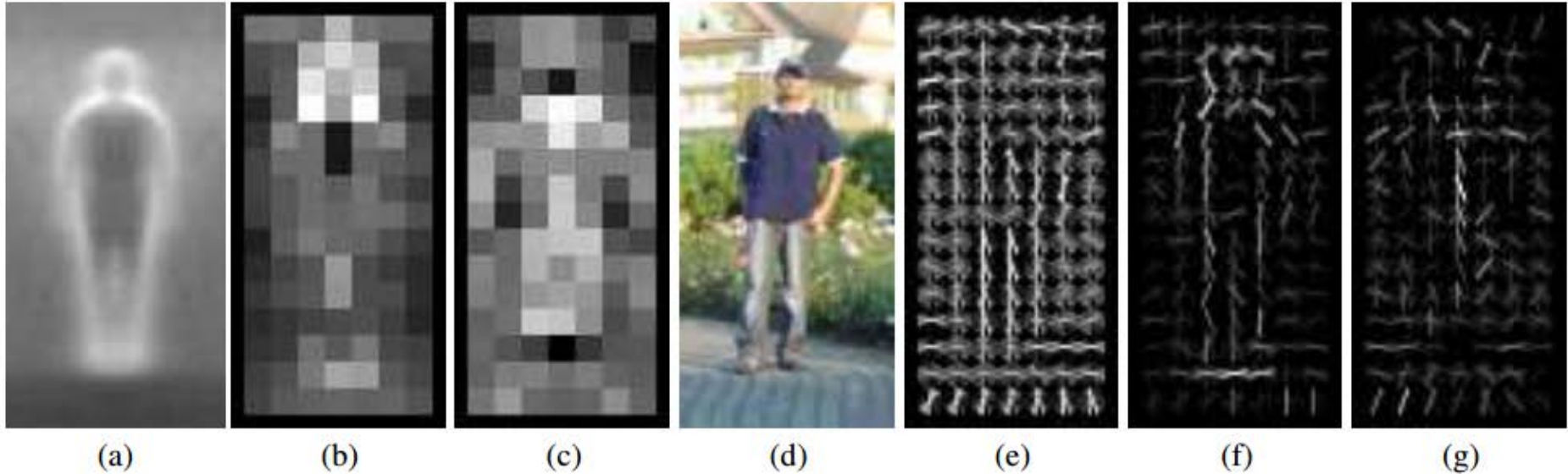
Average Precision (AP)  
0% is worst  
100% is best

mean AP over classes  
(mAP)

# Pedestrians

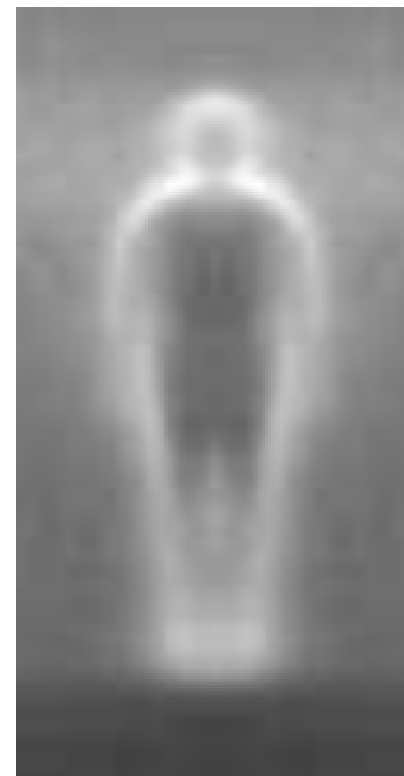
AP ~77%

More sophisticated methods: AP ~90%



- (a) average gradient image over training examples
- (b) each “pixel” shows max positive SVM weight in the block centered on that pixel
- (c) same as (b) for negative SVM weights
- (d) test image
- (e) its R-HOG descriptor
- (f) R-HOG descriptor weighted by positive SVM weights
- (g) R-HOG descriptor weighted by negative SVM weights

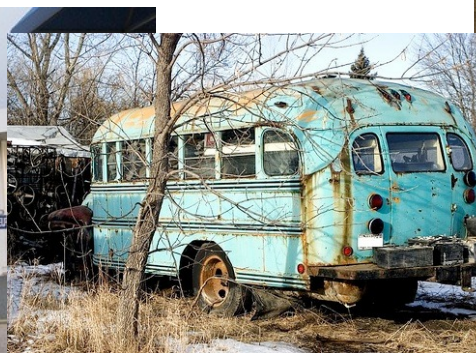
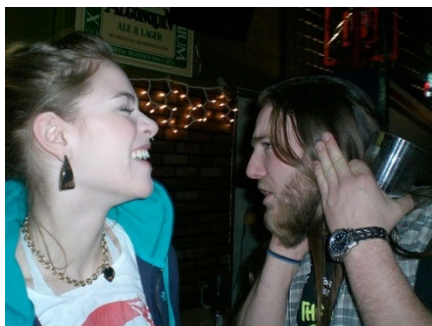
# Why did it work?



Average gradient image



# Generic categories



Can we detect people, chairs, horses, cars, dogs, buses, bottles, sheep ...?  
**PASCAL Visual Object Categories (VOC) dataset**



# Generic categories

Why doesn't this work (as well)?



Can we detect people, chairs, horses, cars, dogs, buses, bottles, sheep ...?  
**PASCAL Visual Object Categories (VOC) dataset**

Quiz time

# Warm up



This is an average image of which object class?

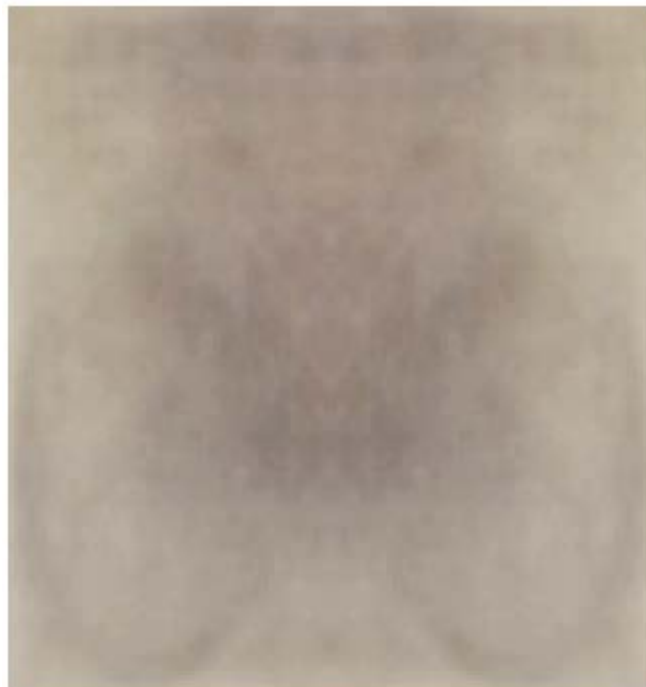
Warm up



pedestrian

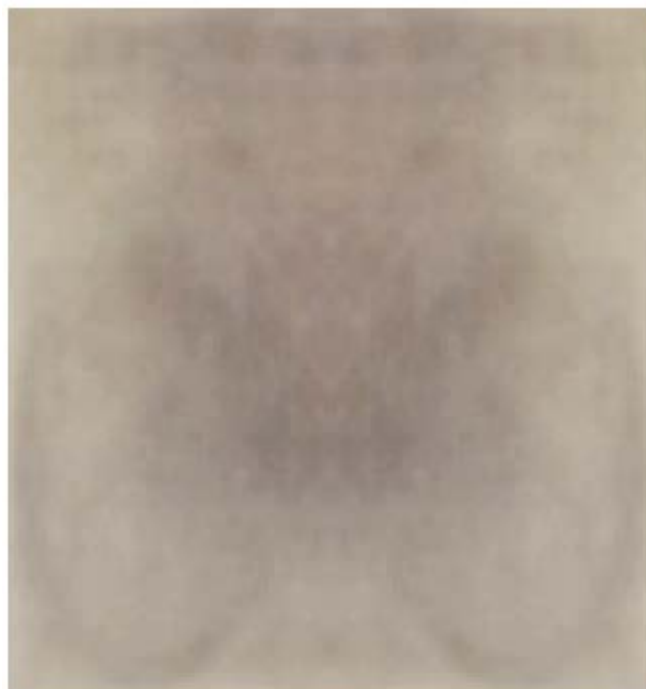


A little harder



?

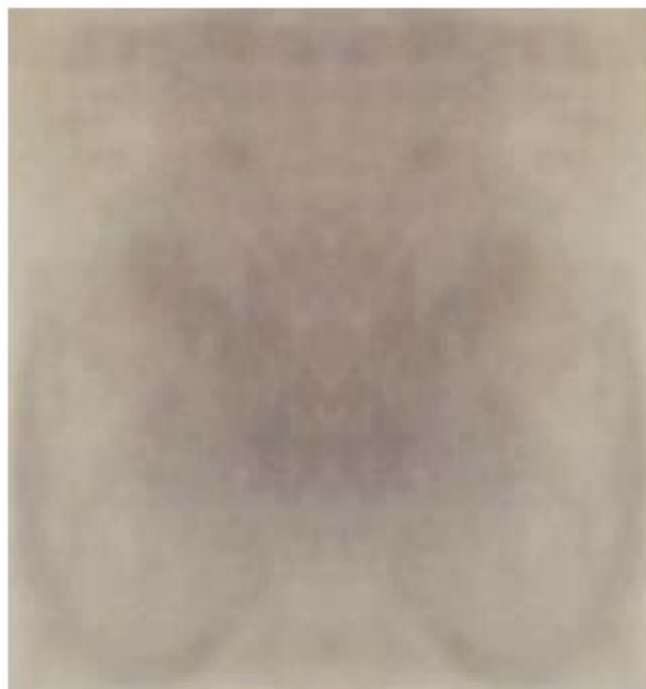
# A little harder



?

Hint: airplane, bicycle, bus, car, cat, chair, cow, dog, dining table

A little harder



bicycle (PASCAL)

A little harder, yet



?



A little harder, yet



?

Hint: white blob on a green background

A little harder, yet



sheep (PASCAL)

# Impossible?



?

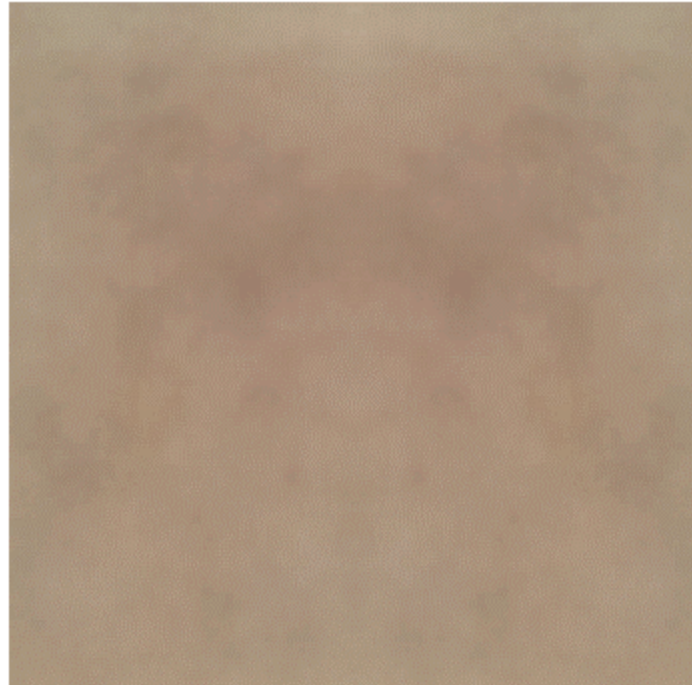
# Impossible?



dog (PASCAL)



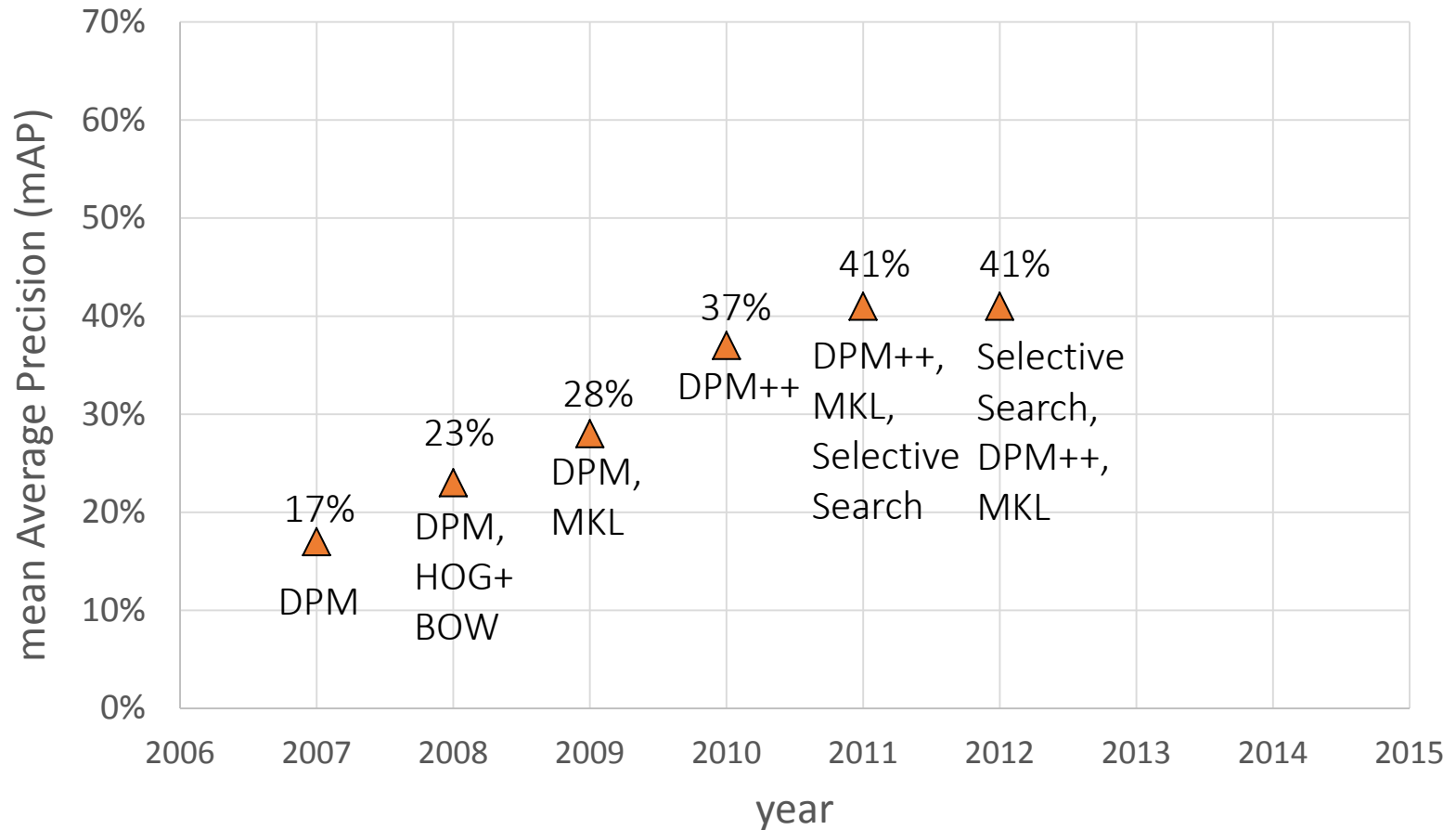
# Impossible?



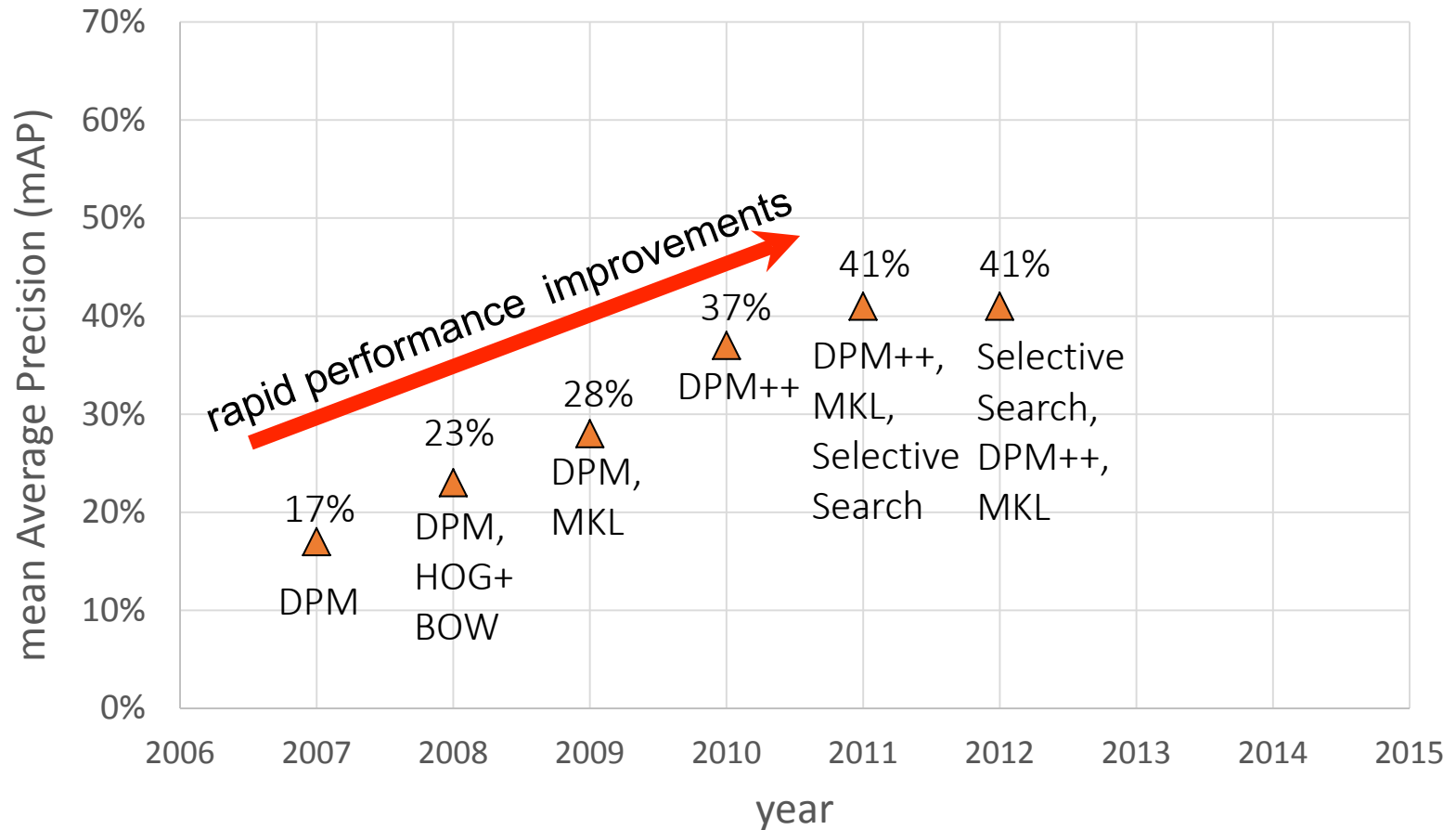
dog (PASCAL)

Why does the mean look like this?  
There's no alignment between the examples!  
How do we combat this?

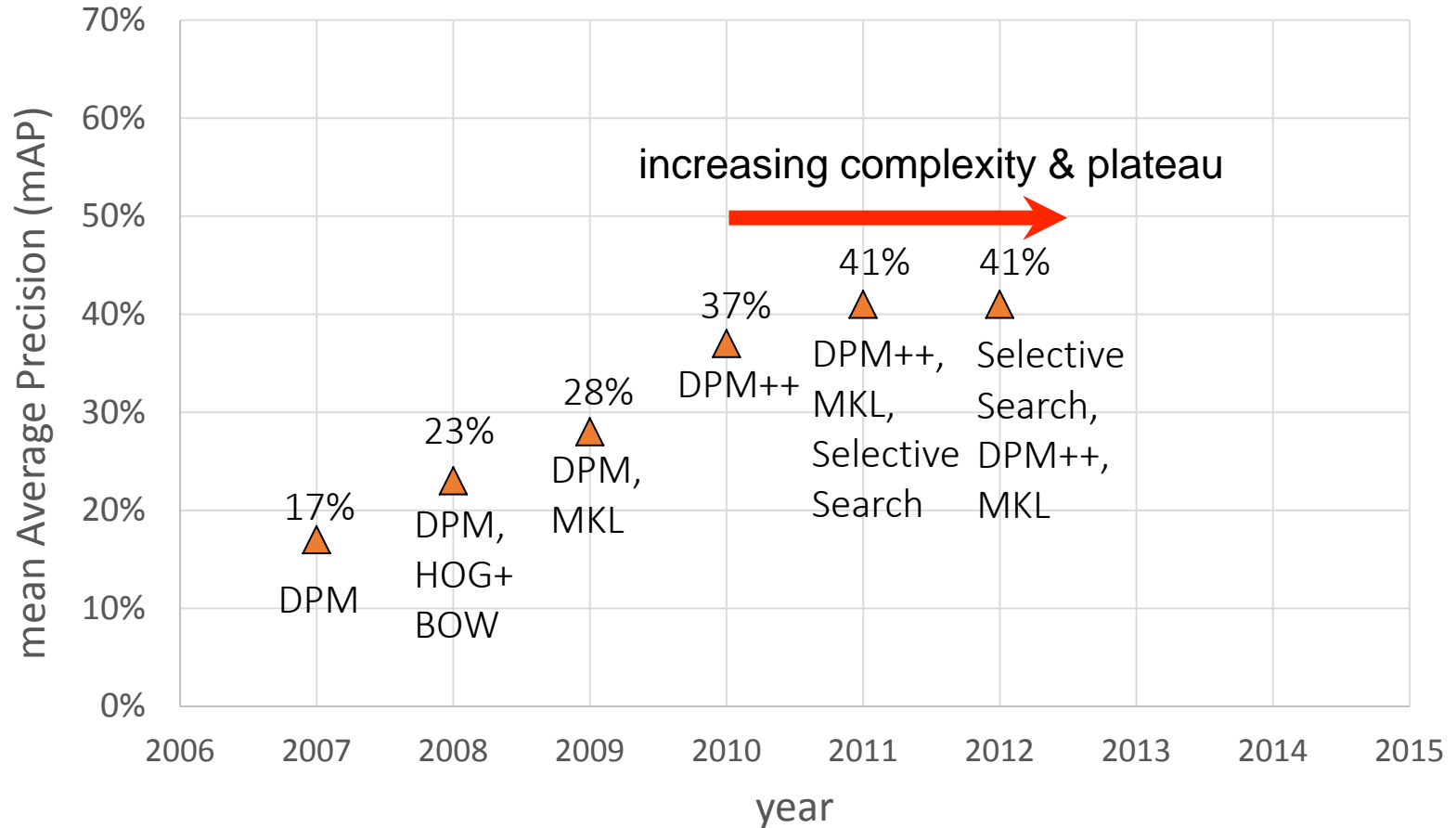
# PASCAL VOC detection history



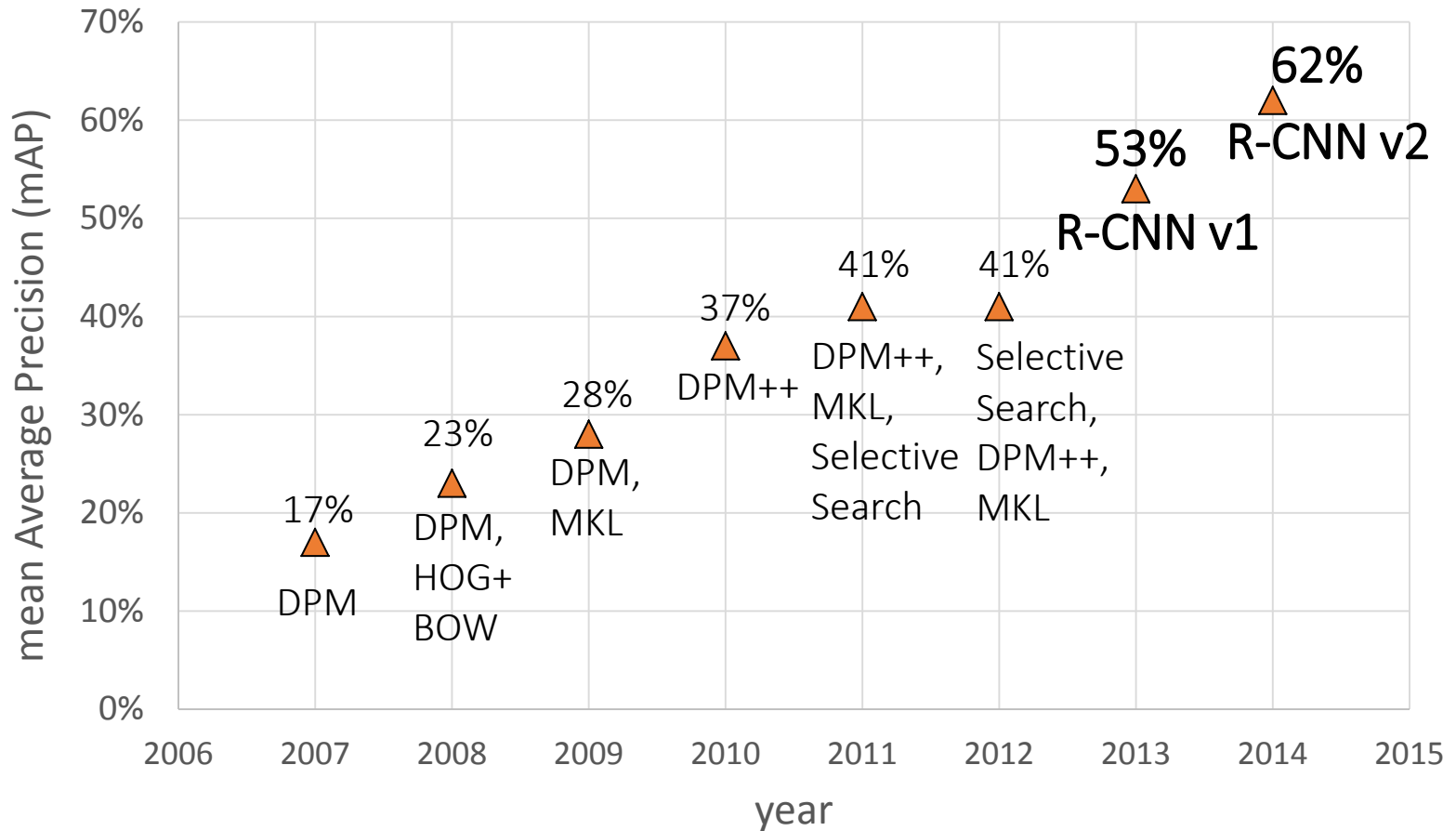
# Part-based models & multiple features (MKL)



# Kitchen-sink approaches



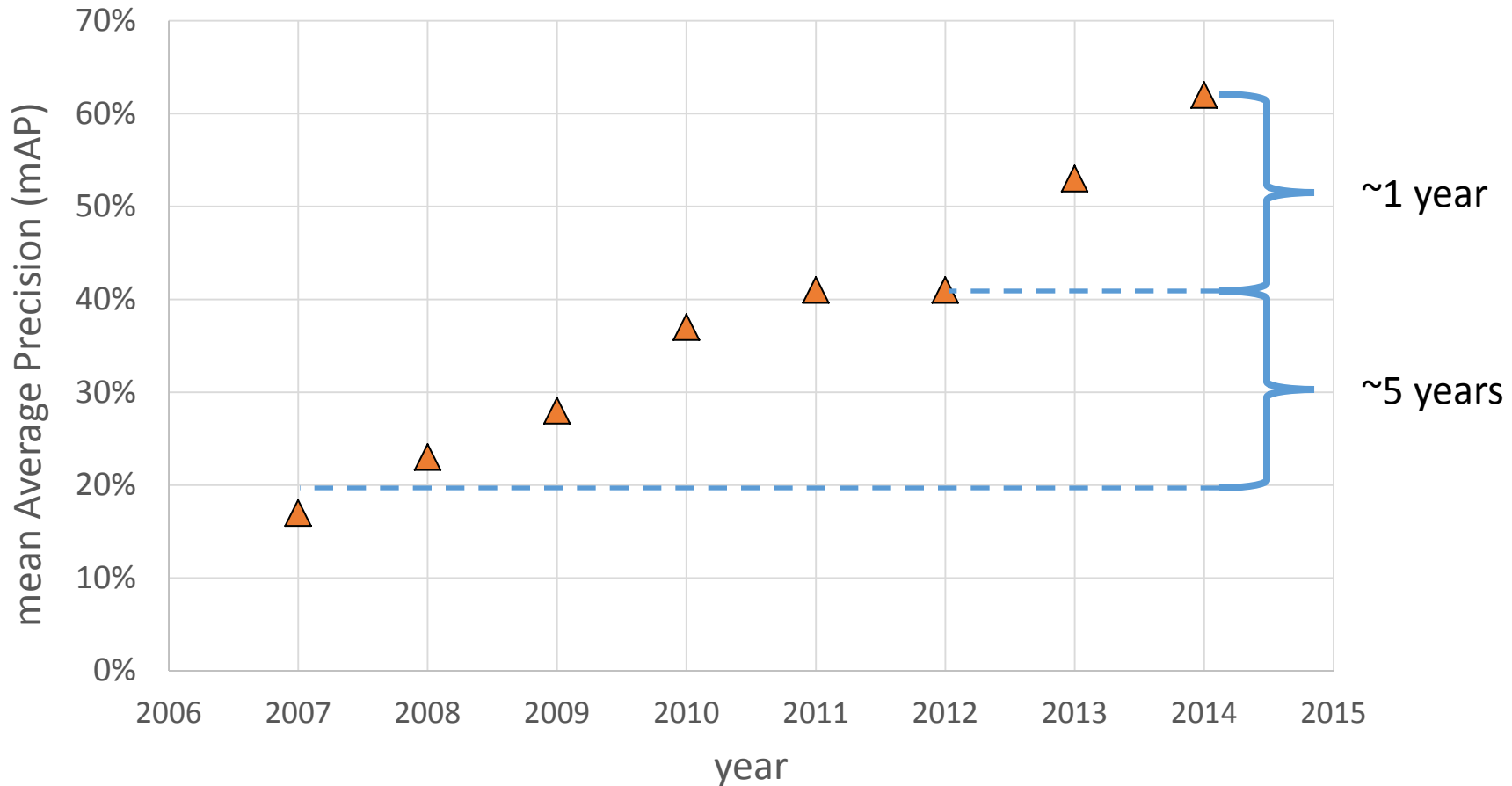
# Region-based Convolutional Networks (R-CNNs)



[R-CNN. Girshick et al. CVPR 2014]



# Region-based Convolutional Networks (R-CNNs)

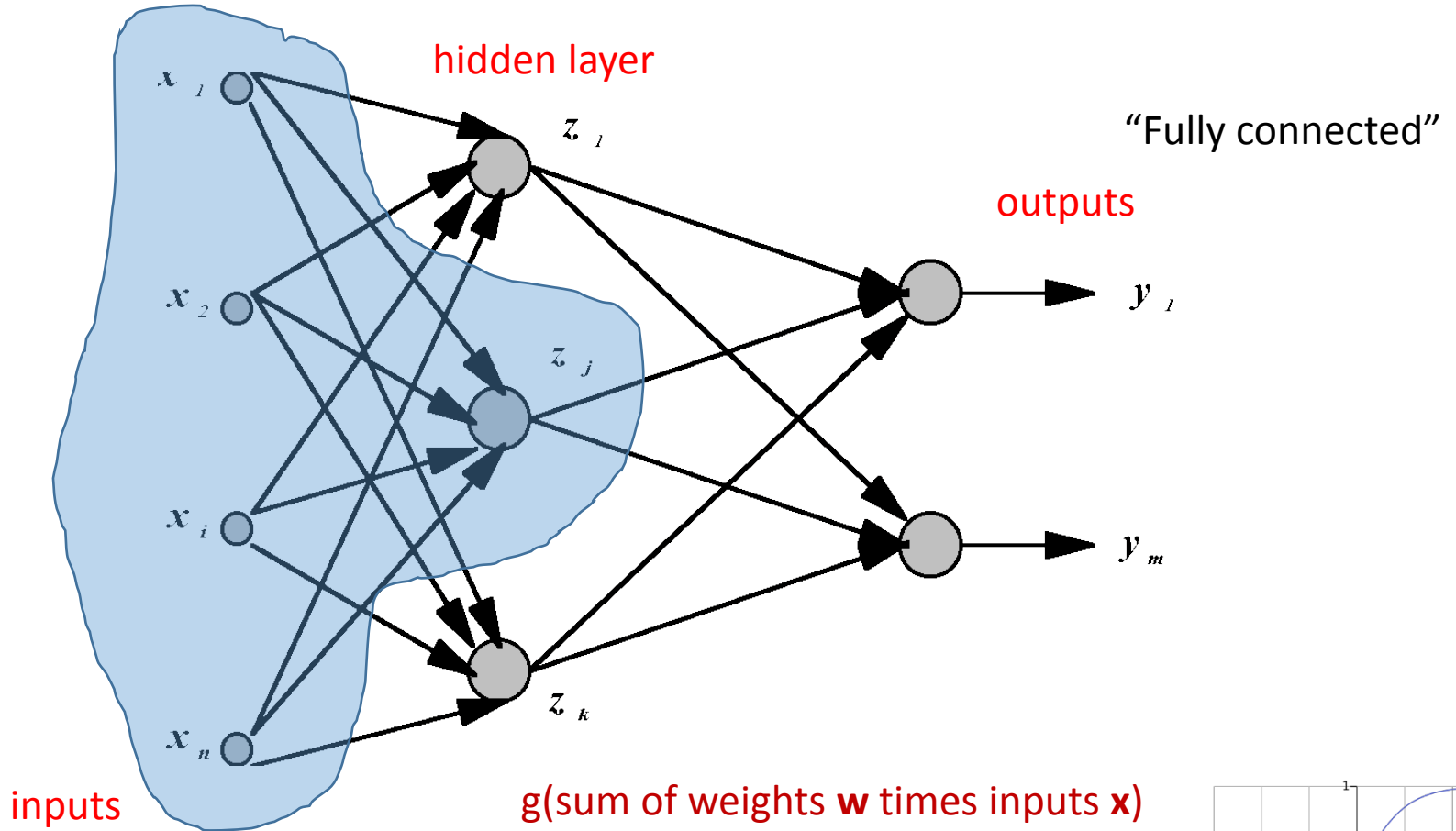


[R-CNN. Girshick et al. CVPR 2014]

# Convolutional Neural Networks

- Overview

# Standard Neural Networks



inputs

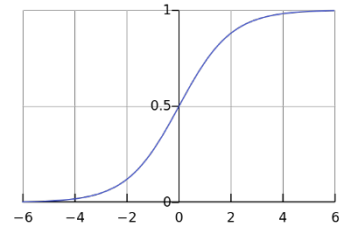
$$\mathbf{x} = (x_1, \dots, x_{784})^T$$



$g(\text{sum of weights } \mathbf{w} \text{ times inputs } \mathbf{x})$

$$z_j = g(\mathbf{w}_j^T \mathbf{x})$$

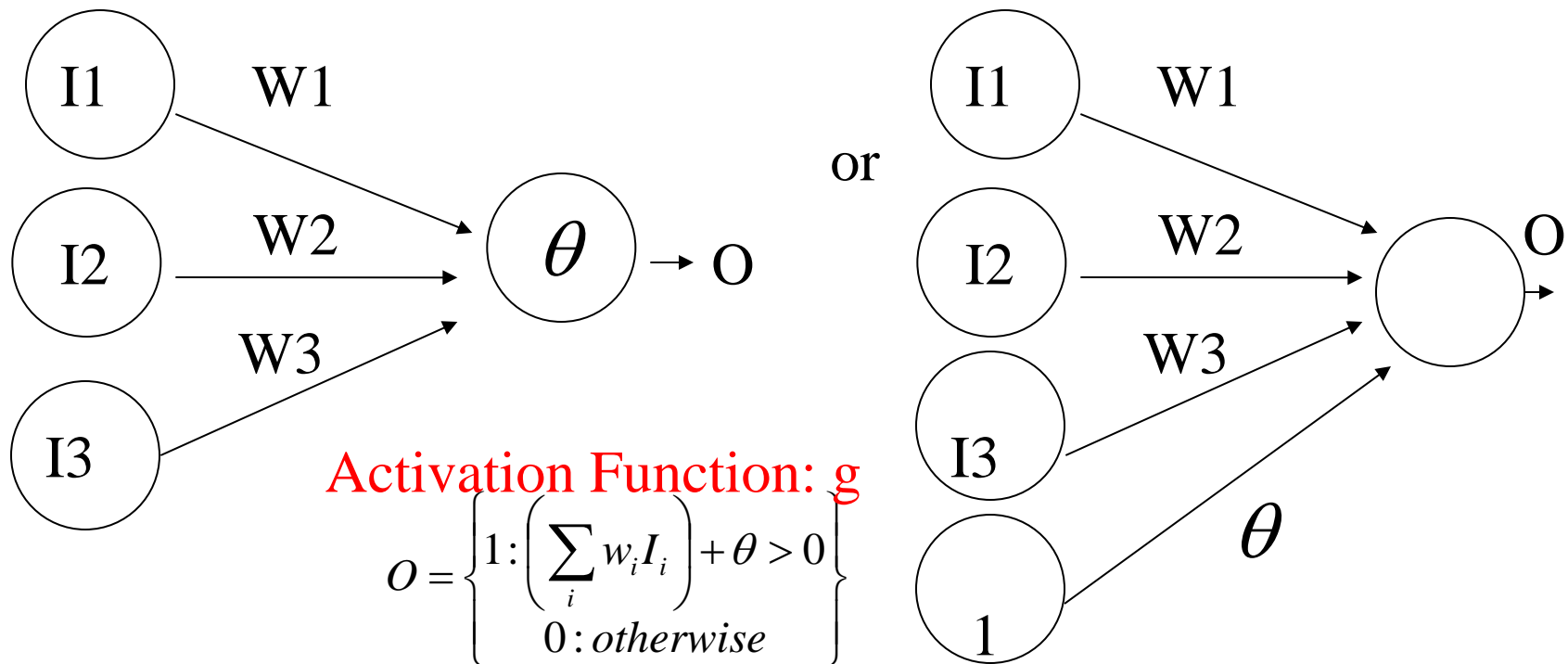
$$g(t) = \frac{1}{1 + e^{-t}}$$



Let's look at how these work

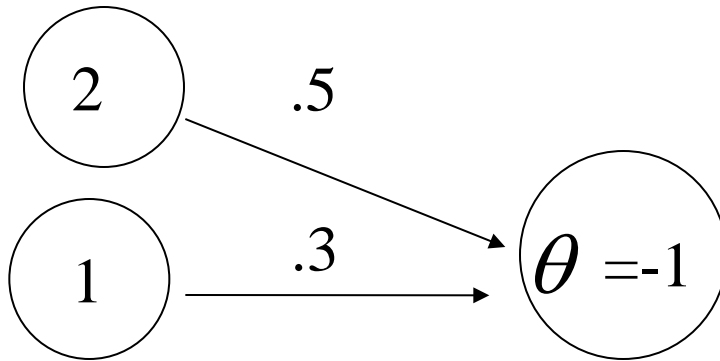
# Perceptrons

- Initial proposal of connectionist networks
- Rosenblatt, 50's and 60's
- Essentially a linear discriminant composed of nodes, weights





# Perceptron Example



$$2(0.5) + 1(0.3) + -1 = 0.3 > 0, \text{ so } \mathbf{O=1}$$

## Learning Procedure:

Randomly assign weights (between 0 and 1)

Present inputs from training data

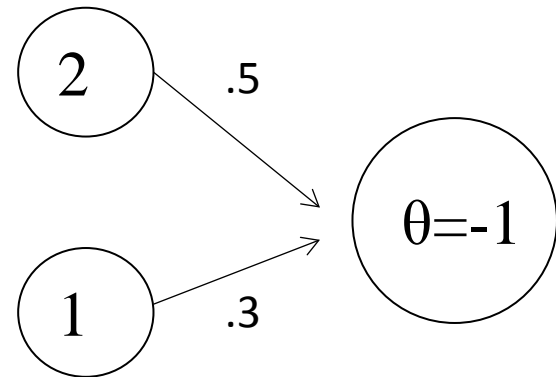
Get output O, nudge weights to give results toward our desired output T

Repeat; stop when no errors, or enough epochs completed

# Perception Training

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = (T - O)I_i$$



Weights include Threshold. T=Desired, O=Actual output.

Example: **T=0**, **O=1**,  $W_1=0.5$ ,  $W_2=0.3$ ,  $I_1=2$ ,  $I_2=1$ ,  $\theta=-1$

$$w_1(t+1) = 0.5 + (0 - 1)(2) = -1.5$$

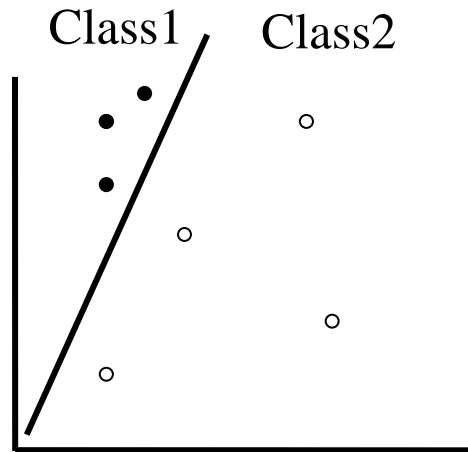
$$w_2(t+1) = 0.3 + (0 - 1)(1) = -0.7$$

$$w_\theta(t+1) = -1 + (0 - 1)(1) = -2$$

If we present this input again, we'd output 0 instead

# Perceptrons are not powerful

- Essentially a linear discriminant
- Perceptron theorem: If a linear discriminant exists that can separate the classes without error, the training procedure is guaranteed to find that line or plane.



# LMS Learning

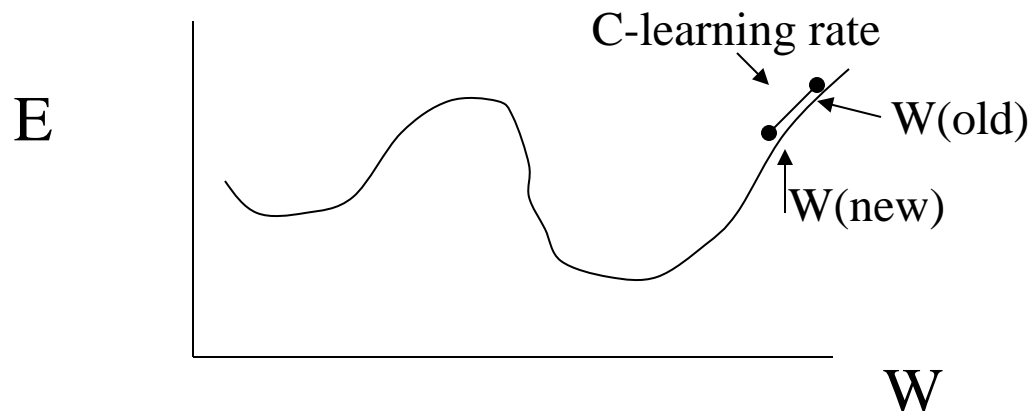
**LMS = Least Mean Square Learning Systems**, more general than the previous perceptron learning rule. The concept is to **minimize the total error, as measured over all training examples, P**.  $O$  is the raw output, as calculated by  $\sum_i w_i I_i + \theta$

$$Distance(LMS) = \frac{1}{2} \sum_P (T_P - O_P)^2$$

E.g. if we have two patterns and

$T_1=1, O_1=0.8, T_2=0, O_2=0.5$  then  $D=(0.5)[(1-0.8)^2+(0-0.5)^2]=.145$

We want to minimize the LMS:



# Activation Function

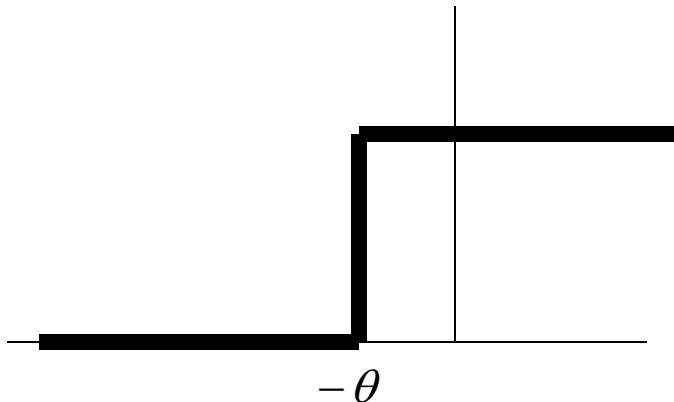
- To apply the LMS learning rule, also known as the delta rule, we need a **differentiable activation function**.

$$\Delta w_k = c I_k (T_j - O_j) f'(\text{Activation Function})$$

see next slide!

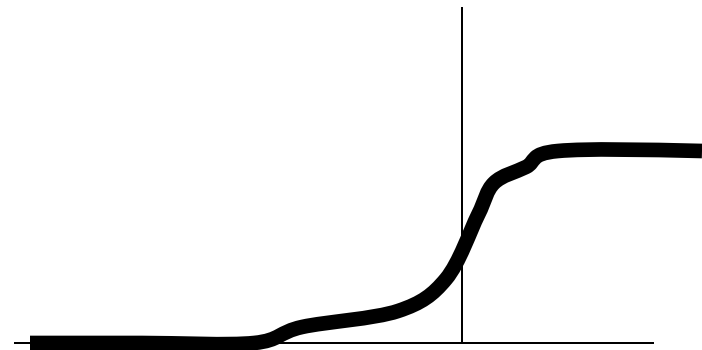
Old:

$$O = \begin{cases} 1: \sum_i w_i I_i + \theta > 0 \\ 0: \text{otherwise} \end{cases}$$



New:

$$O = \frac{1}{1 + e^{-\sum_i w_i I_i + \theta}}$$





# Gradient Descent Learning from Russell and Norvig AI Text

examples is the training set

Each input  $\mathbf{x}$  is a tuple  $x_1, \dots, x_n$  and has true output  $y$ .

Weights are in vector  $W$ ; activation function is  $g$ .

**repeat**

**for** each  $e$  in examples **do**

$$in = \sum W_j x_j[e]$$

$$Err = y[e] - g(in)$$

$$W_j = W_j + \alpha \times Err \times g'(in) \times x_j[e]$$

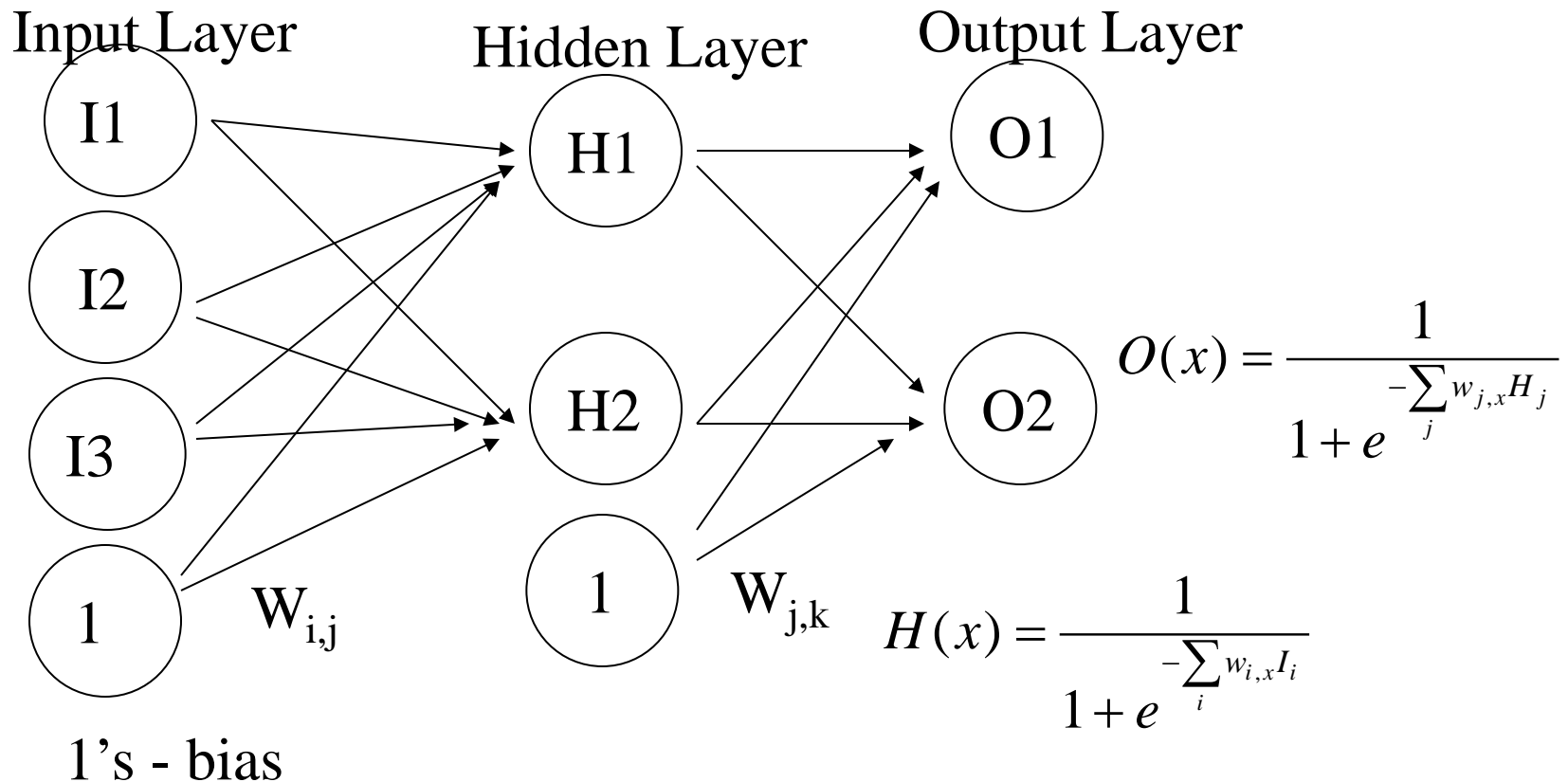
**until** some stopping criterion is satisfied

# LMS vs. Limiting Threshold

- With the new sigmoidal function that is differentiable, we can apply the delta rule toward learning.
- **Perceptron Method**
  - Forced output to 0 or 1, while LMS uses the net output
  - Guaranteed to separate, if no error and is linearly separable
    - Otherwise it may not converge
- **Gradient Descent Method:**
  - May oscillate and not converge
  - May converge to wrong answer
  - Will converge to some minimum even if the classes are not linearly separable, unlike the earlier perceptron training method

# Backpropagation Networks

- Attributed to Rumelhart and McClelland, late 70's
- To bypass the linear classification problem, we can construct *multilayer networks*. Typically we have *fully connected, feedforward* networks.



# Backprop - Learning

## Learning Procedure:

Randomly assign weights (between 0-1)

Present inputs from training data, propagate to outputs

Compute outputs  $O$ , adjust weights according to the delta rule, backpropagating the errors. The weights will be nudged closer so that the network learns to give the desired output.

Repeat; stop when no errors, or enough epochs completed

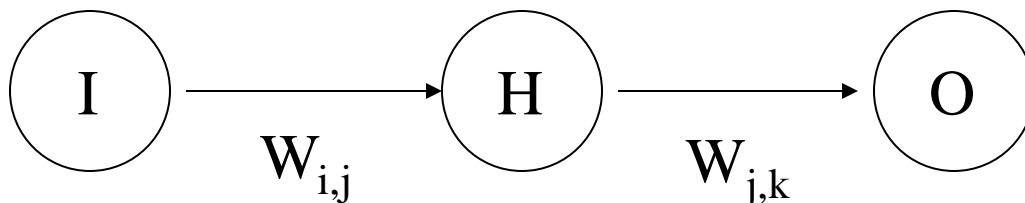
# Backprop - Modifying Weights

See Russell and Norvig algorithm in Figure 20.25 for details.

Lots of nested for loops for all the layers.

This is the idea from NN slides.

$$\Delta w_{i,j} = c H_j (1 - H_j) I_i \sum_k (T_k - O_k) O_k (1 - O_k) w_{j,k}$$



# Backprop

- Very powerful - can learn any function, given enough hidden units! With enough hidden units, we can generate any function.
- Have the same problems of Generalization vs. Memorization. With too many units, we will tend to memorize the input and not generalize well. Some schemes exist to “prune” the neural network.
- Networks require extensive training, many parameters to fiddle with. Can be extremely slow to train. May also fall into local minima.
- Inherently parallel algorithm, ideal for multiprocessor hardware.
- Despite the cons, a very powerful algorithm that has seen widespread successful deployment.

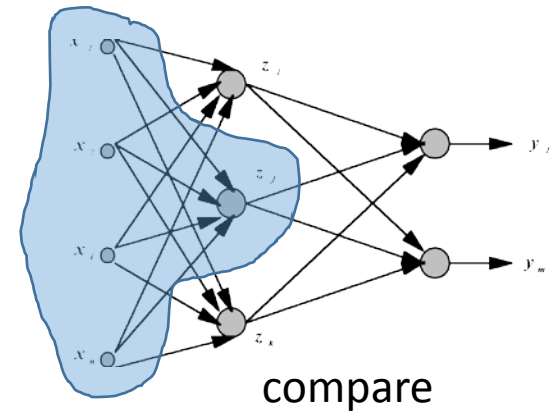
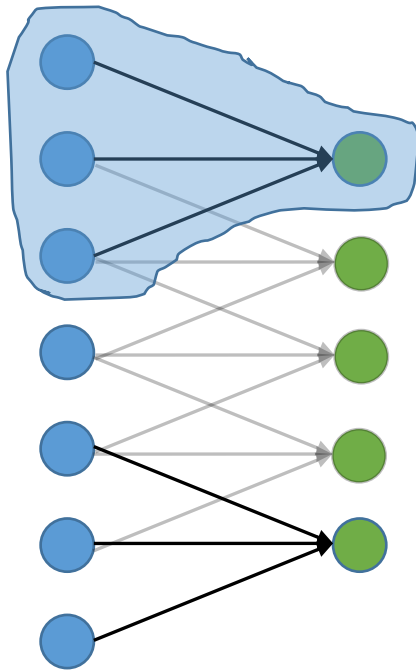


# From NNs to Convolutional NNs

- Local connectivity
- Shared (“tied”) weights
- Multiple feature maps
- Pooling

# Convolutional NNs

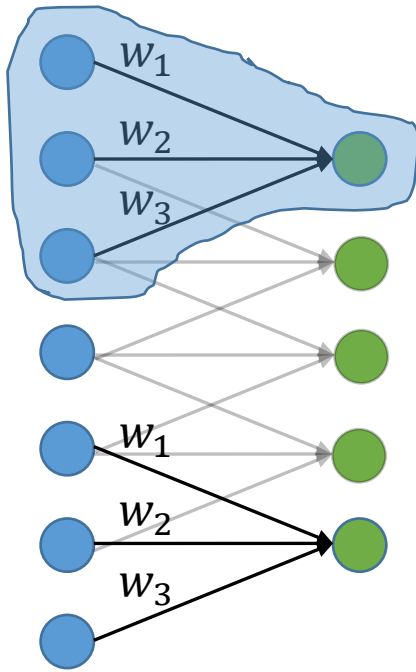
- Local connectivity



- Each green unit is only connected to (3) **neighboring** blue units

# Convolutional NNs

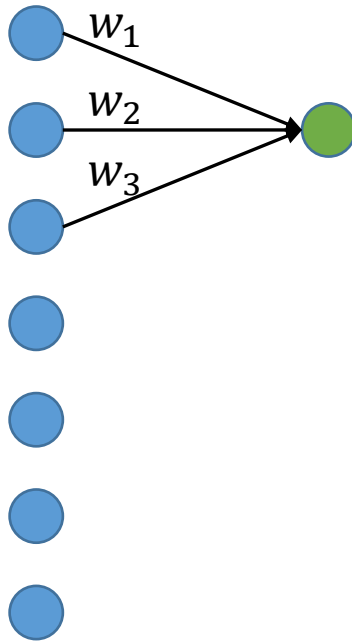
- Shared (“tied”) weights



- All green units **share** the same parameters  $w$
- Each green unit computes the **same function**, but with a **different input window**

# Convolutional NNs

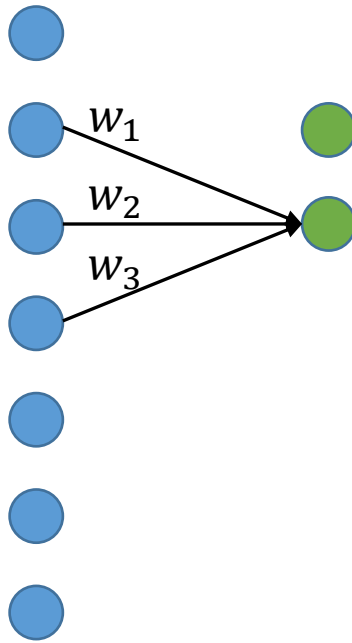
- Convolution with 1-D filter:  $[w_3, w_2, w_1]$



- All green units **share** the same parameters  $w$
- Each green unit computes the **same function**, but with a **different input window**

# Convolutional NNs

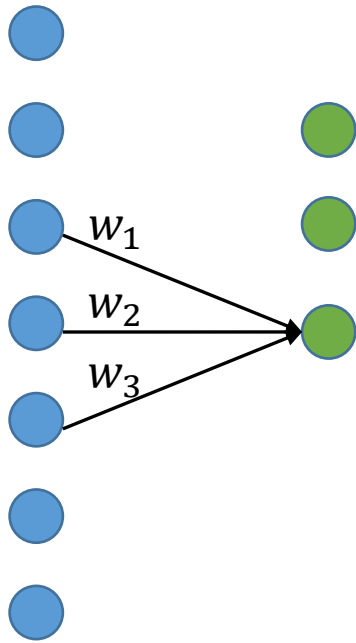
- Convolution with 1-D filter:  $[w_3, w_2, w_1]$



- All green units **share** the same parameters  $w$
- Each green unit computes the **same function**, but with a **different input window**

# Convolutional NNs

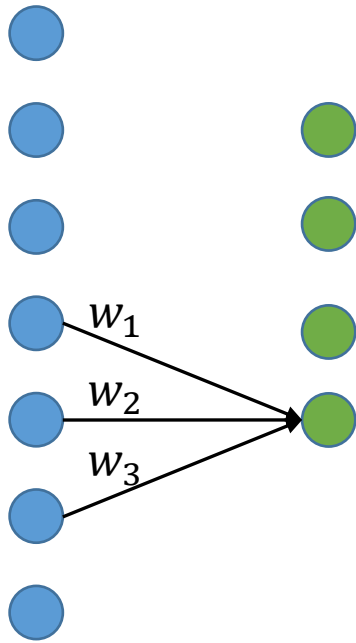
- Convolution with 1-D filter:  $[w_3, w_2, w_1]$



- All green units **share** the same parameters  $w$
- Each green unit computes the **same function**, but with a **different input window**

# Convolutional NNs

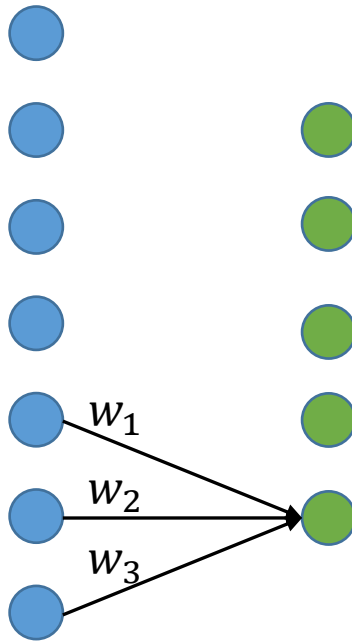
- Convolution with 1-D filter:  $[w_3, w_2, w_1]$



- All green units **share** the same parameters  $w$
- Each green unit computes the **same function**, but with a **different input window**

# Convolutional NNs

- Convolution with 1-D filter:  $[w_3, w_2, w_1]$

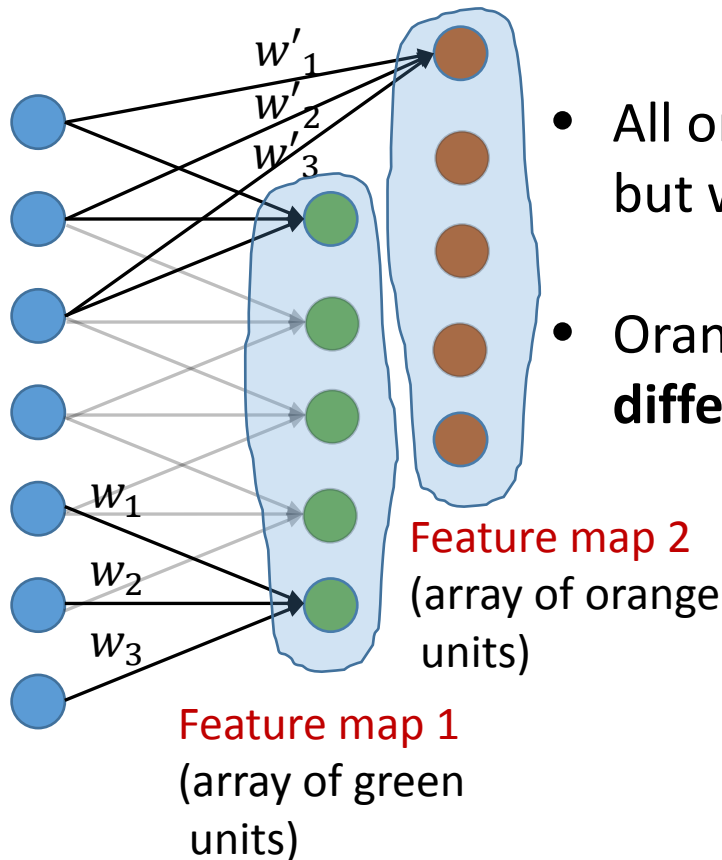


- All green units **share** the same parameters  $w$
- Each green unit computes the **same function**, but with a **different input window**



# Convolutional NNs

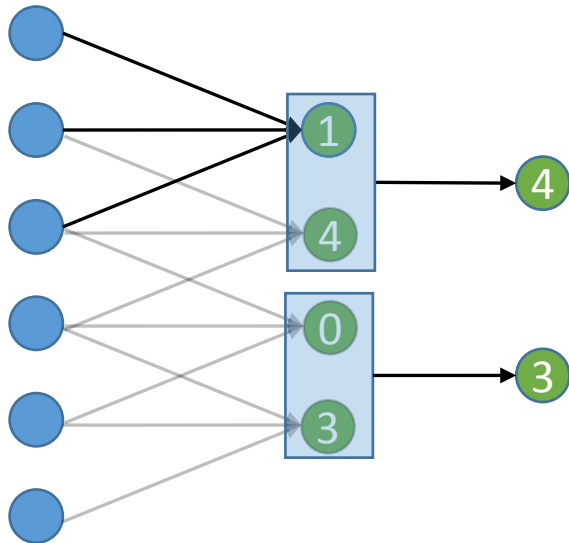
- Multiple feature maps



- All orange units compute the **same function** but with a **different input windows**
- Orange and green units **compute different functions**

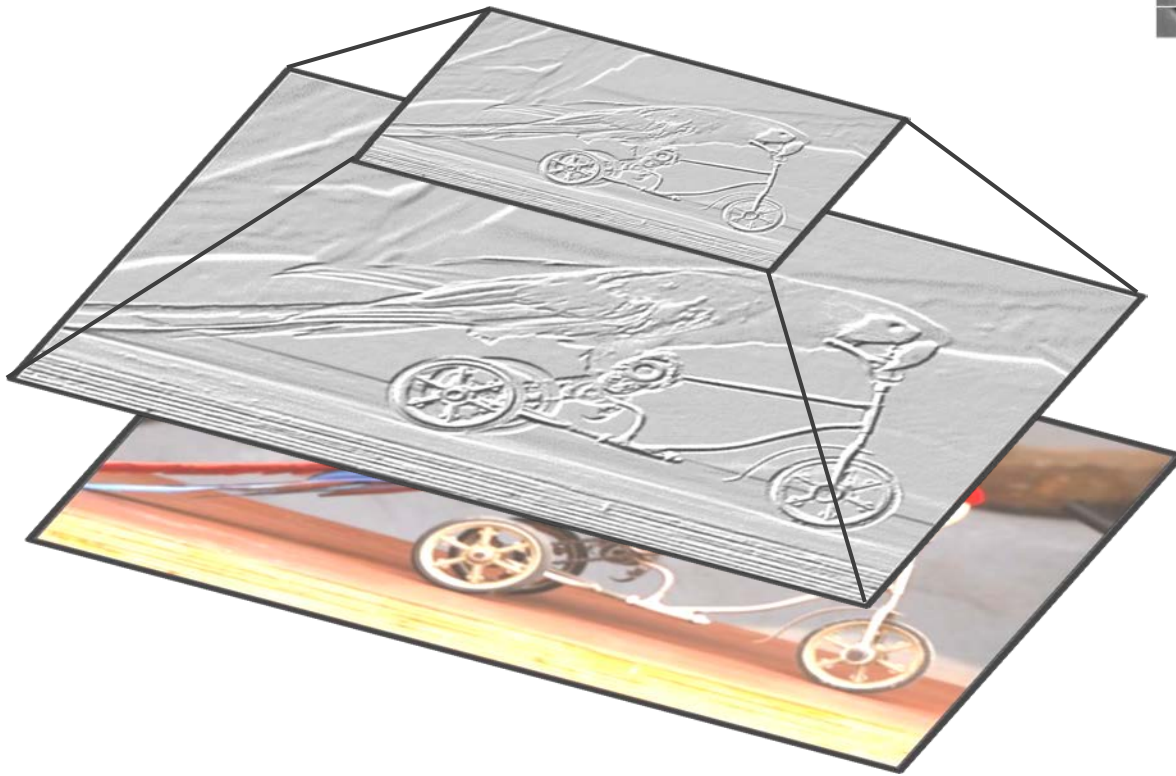
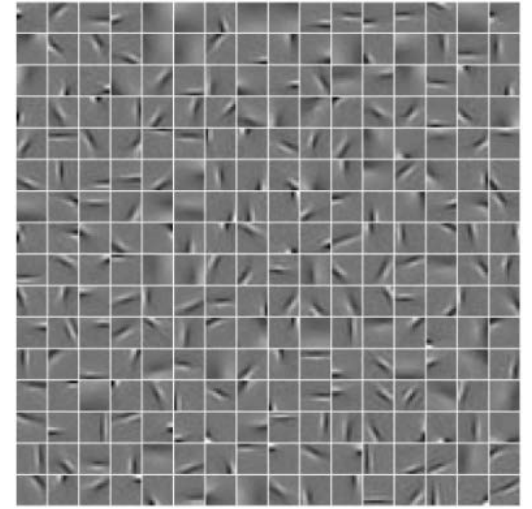
# Convolutional NNs

- Pooling (**max**, average)



- Pooling area: 2 units
- Pooling stride: 2 units
- **Subsamples** feature maps

2D input



Pooling



Convolution



Image

# Historical perspective – 1980

Biol. Cybernetics 36, 193–202 (1980)

---

**Biological  
Cybernetics**  
© by Springer-Verlag 1980

---

## **Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position**

Kunihiko Fukushima

NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan

# Historical perspective – 1980

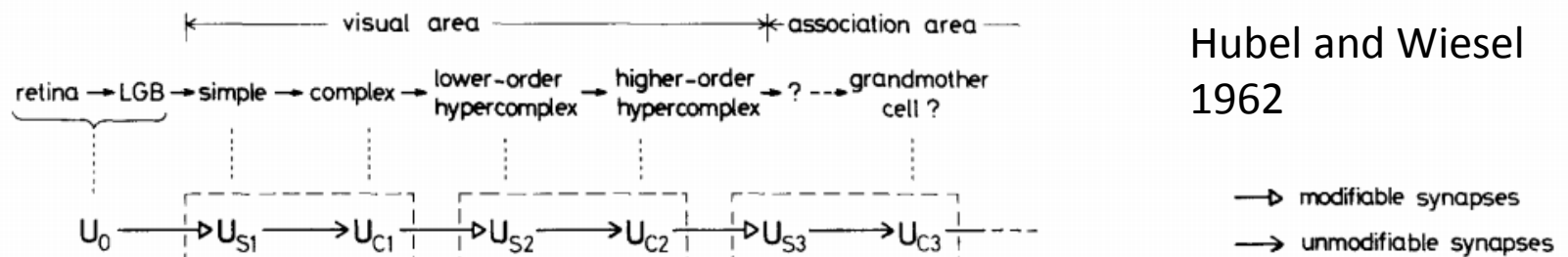


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

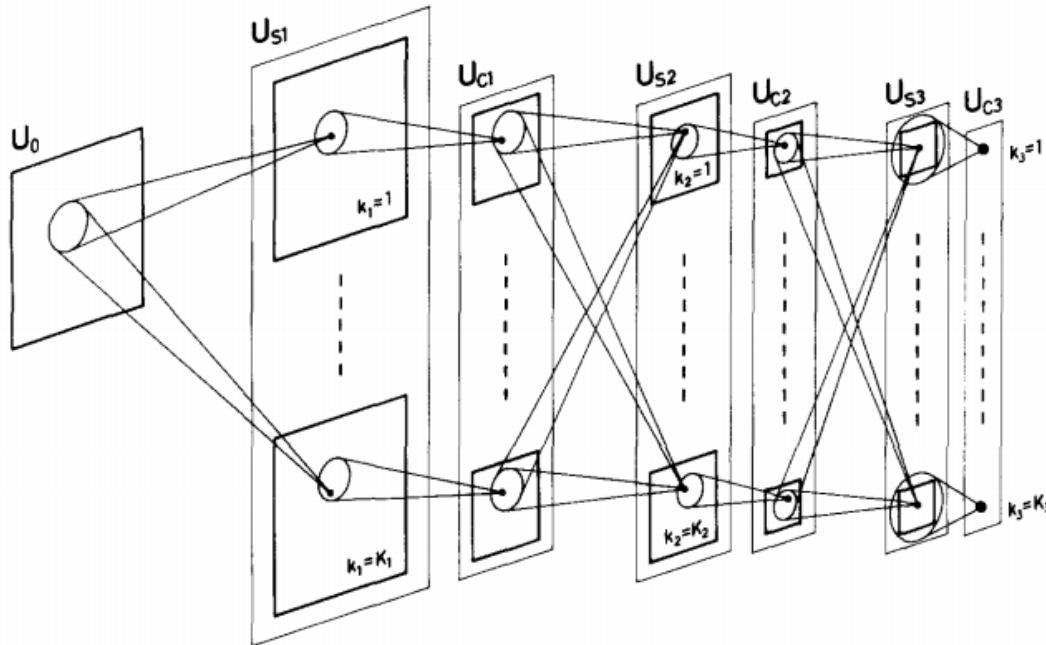


Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Included basic ingredients of ConvNets, but no supervised learning algorithm

# Supervised learning – 1986

Gradient descent training with error backpropagation

---

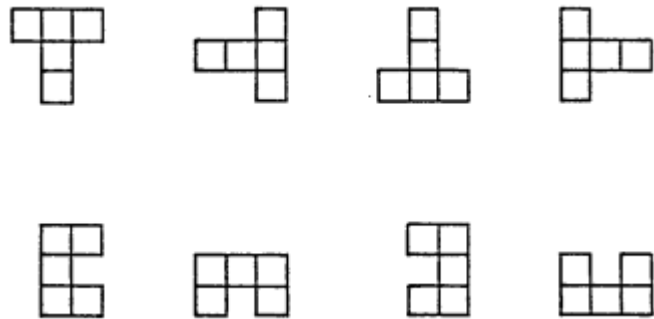
**Learning Internal Representations  
by Error Propagation**

---

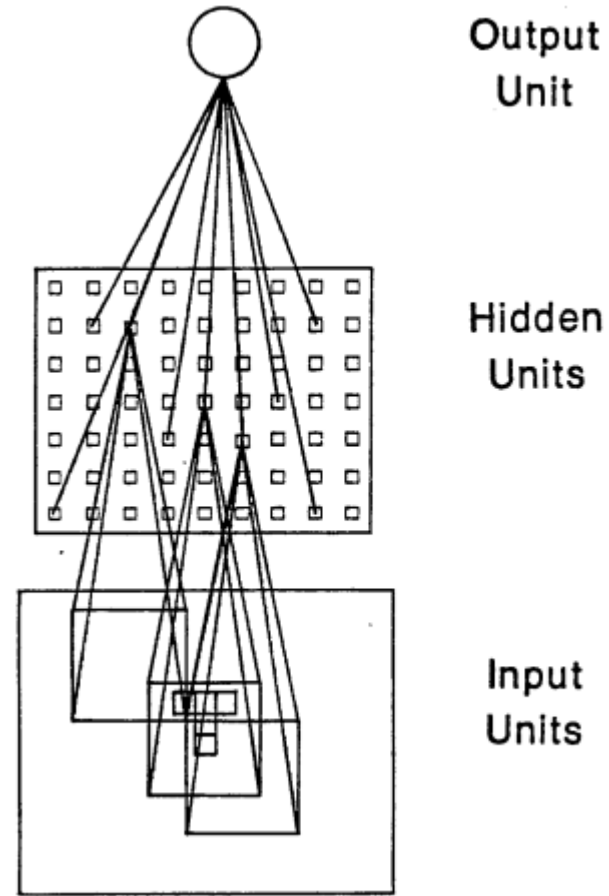
D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS

Early demonstration that error backpropagation can be used for supervised training of neural nets (including ConvNets)

# Supervised learning – 1986

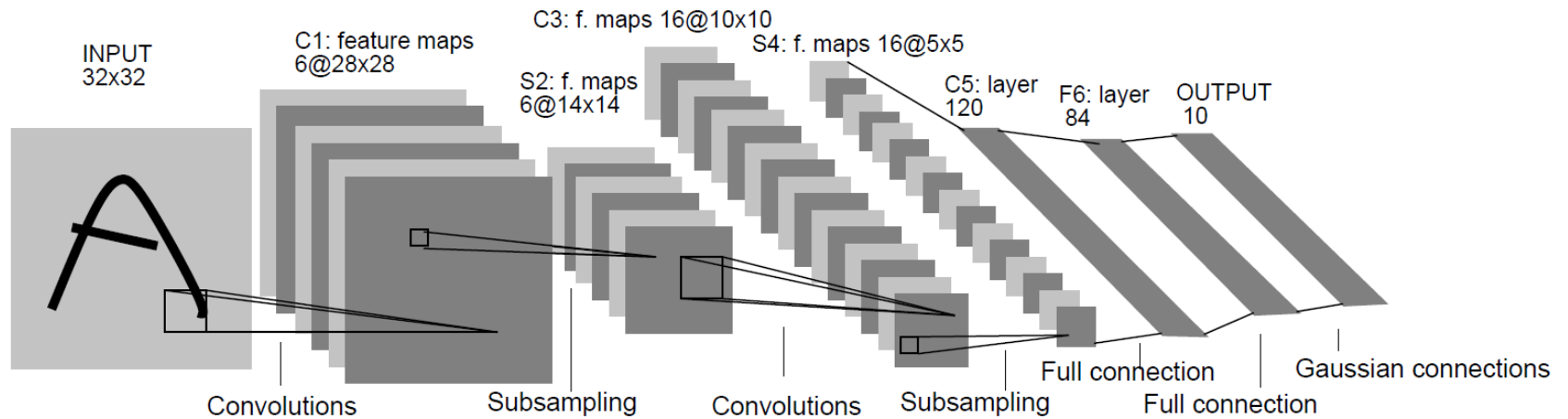


“T” vs. “C” problem



Simple ConvNet

# Practical ConvNets



**Gradient-Based Learning Applied to Document Recognition,**  
Lecun et al., 1998

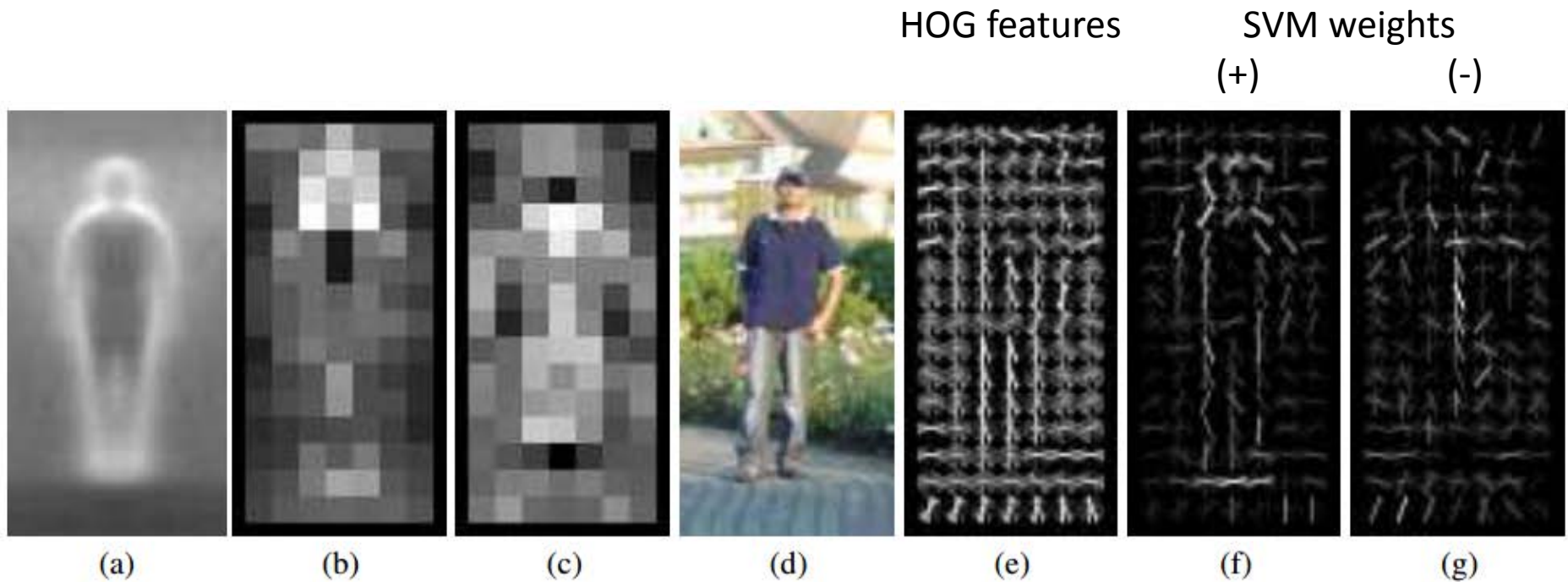


# The fall of ConvNets

- The rise of Support Vector Machines (SVMs)
- Mathematical advantages (theory, convex optimization)
- Competitive performance on tasks such as digit classification
- Neural nets became unpopular in the mid 1990s

# The key to SVMs

- *It's all about the features*



**Histograms of Oriented Gradients for Human Detection,**  
Dalal and Triggs, CVPR 2005

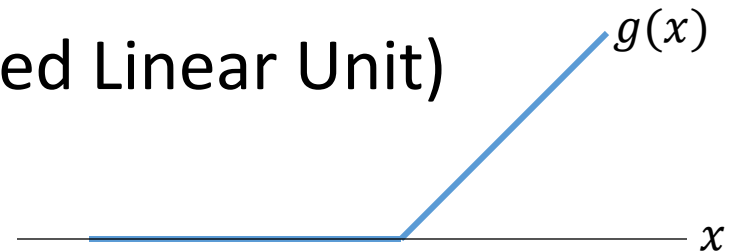
# Core idea of “deep learning”

- Input: the “*raw*” signal (image, waveform, ...)
- Features: hierarchy of features is *learned* from the raw input

- If SVMs killed neural nets, how did they come back (in computer vision)?

# What's new since the 1980s?

- More layers
  - LeNet-3 and LeNet-5 had 3 and 5 learnable layers
  - Current models have 8 – 20+
- “ReLU” non-linearities (Rectified Linear Unit)
  - $g(x) = \max(0, x)$
  - Gradient doesn't vanish
- “Dropout” regularization
- Fast GPU implementations
- More data



# Ross's Own System: Region CNNs

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

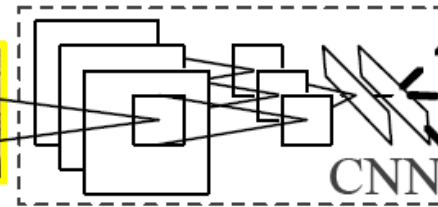


1. Input image

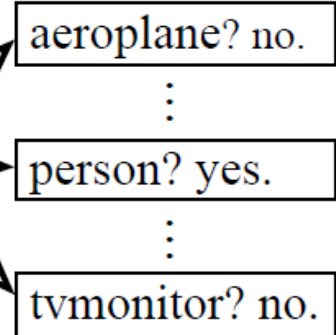


2. Extract region proposals (~2k)

warped region



3. Compute CNN features

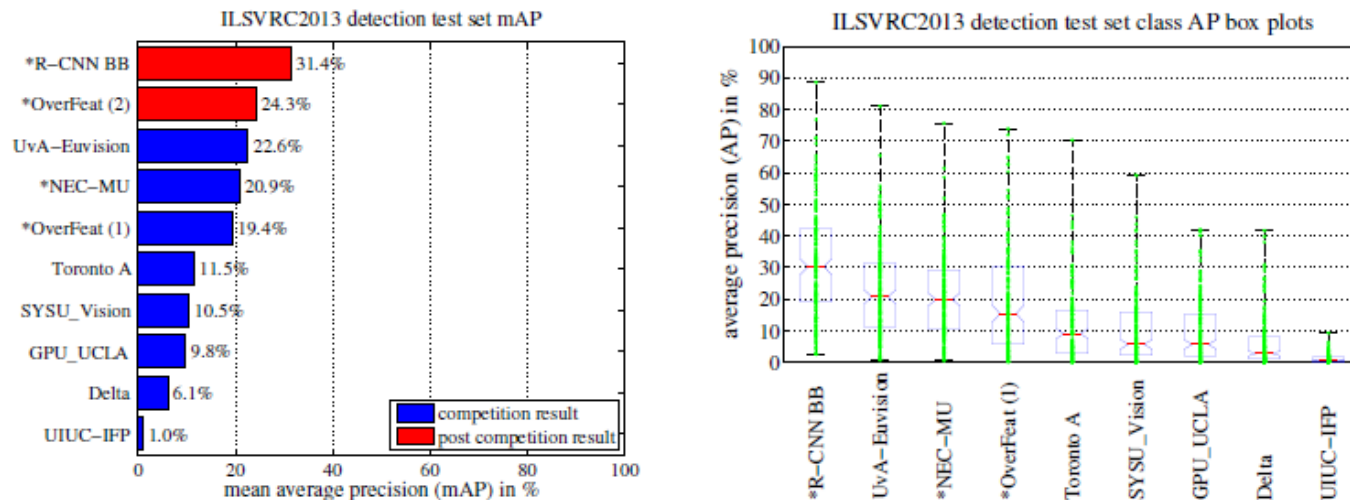


4. Classify regions

# Competitive Results

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [20] <sup>†</sup>	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [39]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [41]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [18] <sup>†</sup>	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	<b>71.8</b>	<b>65.8</b>	<b>53.0</b>	<b>36.8</b>	<b>35.9</b>	<b>59.7</b>	<b>60.0</b>	<b>69.9</b>	<b>27.9</b>	<b>50.6</b>	<b>41.4</b>	<b>70.0</b>	<b>62.0</b>	<b>69.0</b>	<b>58.1</b>	<b>29.5</b>	<b>59.4</b>	<b>39.3</b>	<b>61.2</b>	<b>52.4</b>	<b>53.7</b>

**Table 1: Detection average precision (%) on VOC 2010 test.** R-CNN is most directly comparable to UVA and Regionlets since all methods use selective search region proposals. Bounding-box regression (BB) is described in Section C. At publication time, SegDPM was the top-performer on the PASCAL VOC leaderboard. <sup>†</sup>DPM and SegDPM use context rescoring not used by the other methods.



**Figure 3: (Left) Mean average precision on the ILSVRC2013 detection test set.** Methods preceded by \* use outside training data (images and labels from the ILSVRC classification dataset in all cases). **(Right) Box plots for the 200 average precision values per method.** A box plot for the post-competition OverFeat result is not shown because per-class APs are not yet available (per-class APs for R-CNN are in Table 8 and also included in the tech report source uploaded to arXiv.org; see R-CNN-ILSVRC2013-APs.txt). The red line marks the median AP, the box bottom and top are the 25th and 75th percentiles. The whiskers extend to the min and max AP of each method. Each AP is plotted as a green dot over the whiskers (best viewed digitally with zoom).

# Top Regions for Six Object Classes

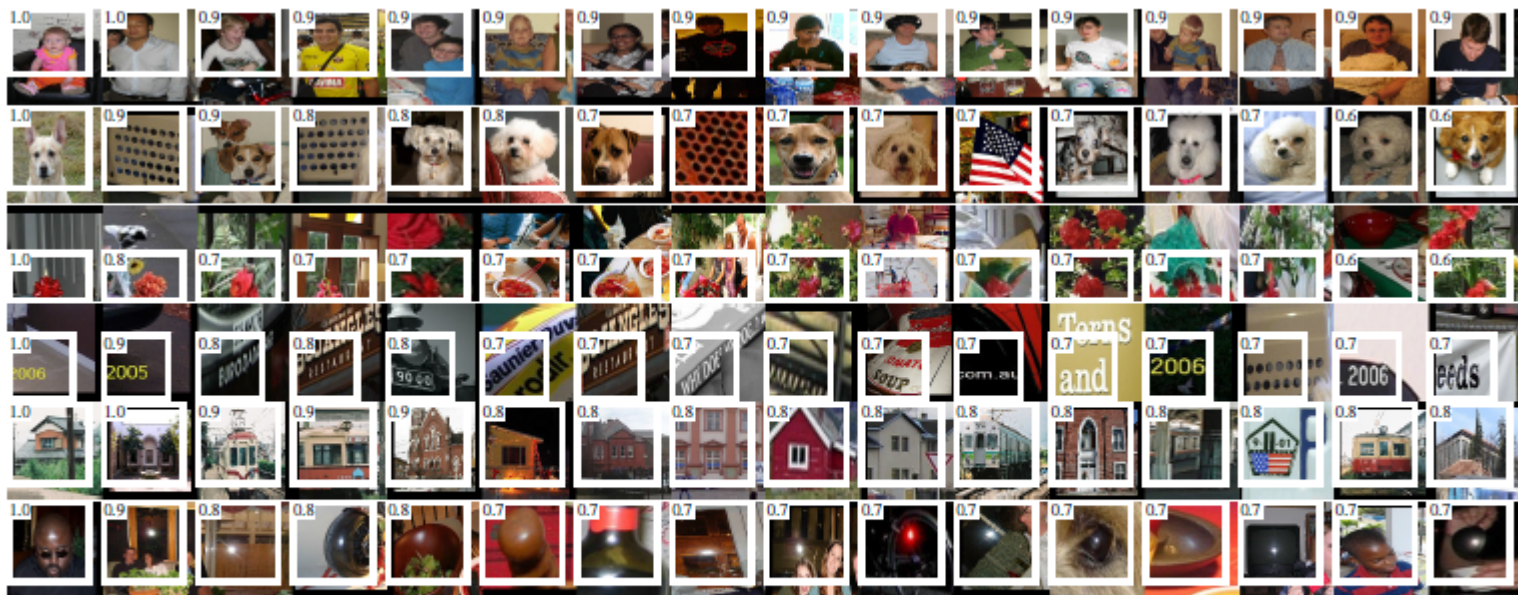
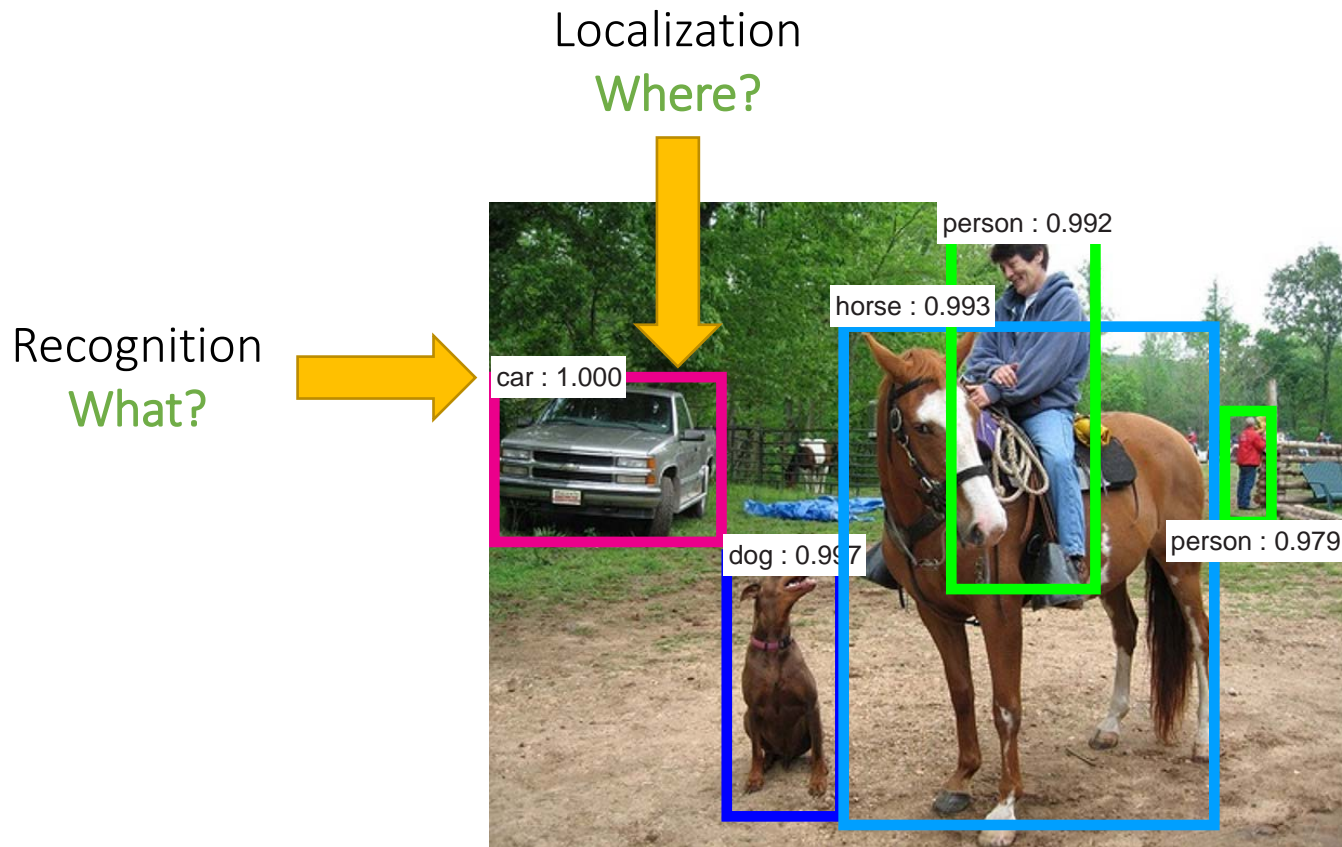


Figure 4: Top regions for six pool<sub>5</sub> units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).

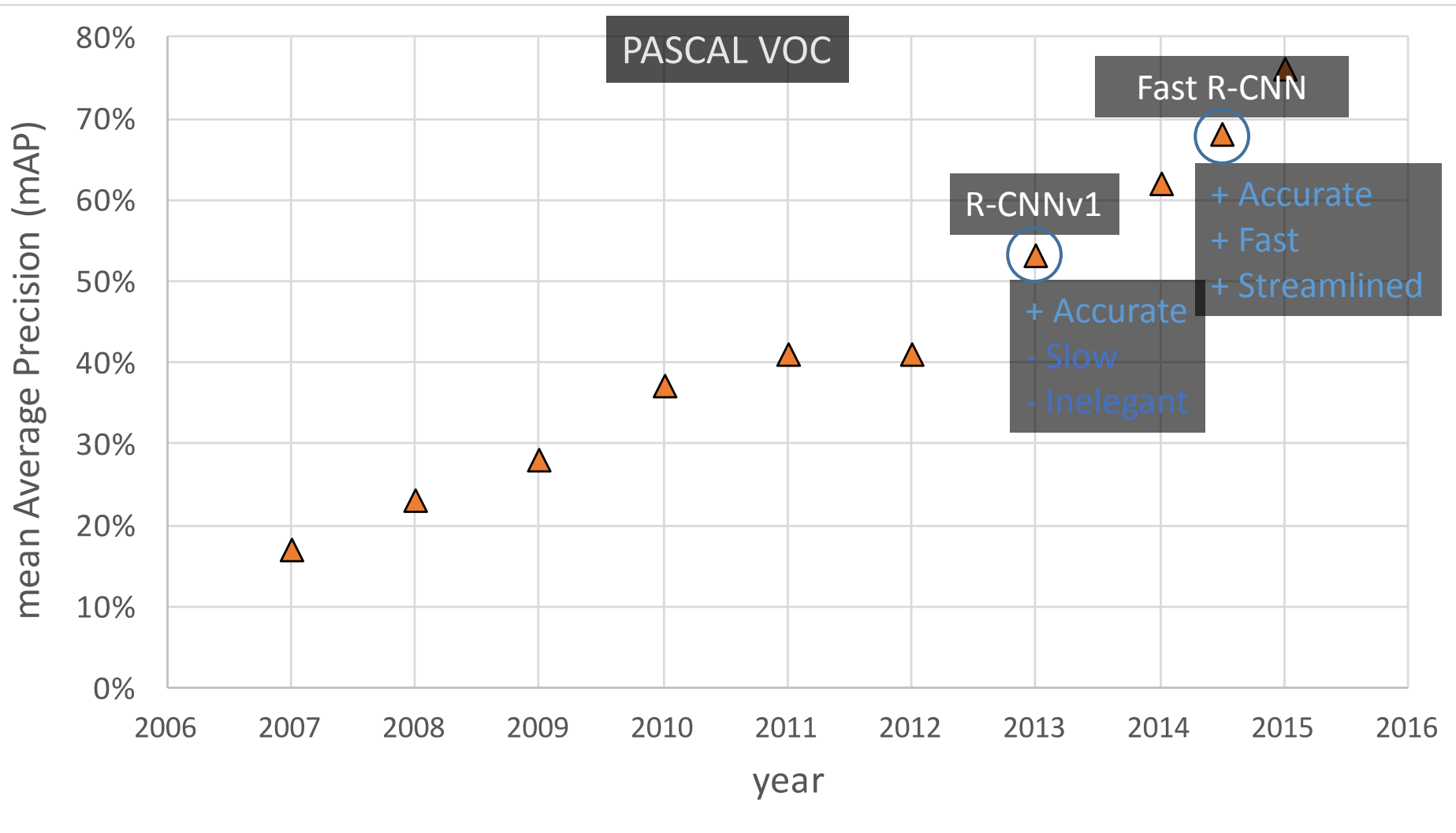




# But it wasn't fast enough! So we have: Fast Region-based ConvNets (R-CNNs) for Object Detection



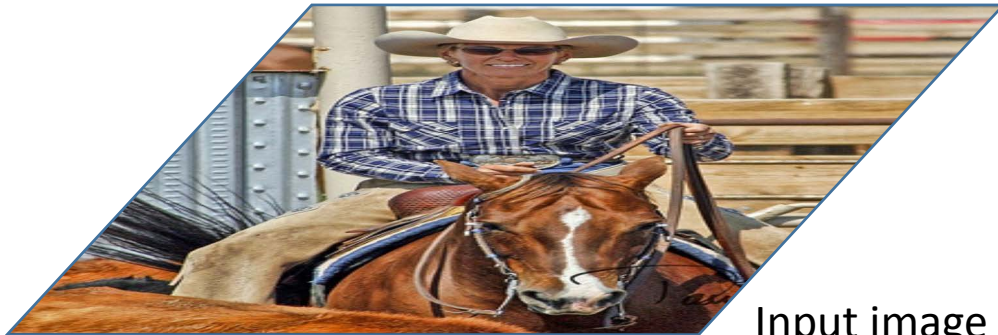
# Object detection renaissance (2013-present)



# Region-based convnets (R-CNNs)

- R-CNN (aka “slow R-CNN”) [Girshick et al. CVPR14]
- SPP-net [He et al. ECCV14]

# Slow R-CNN



Input image

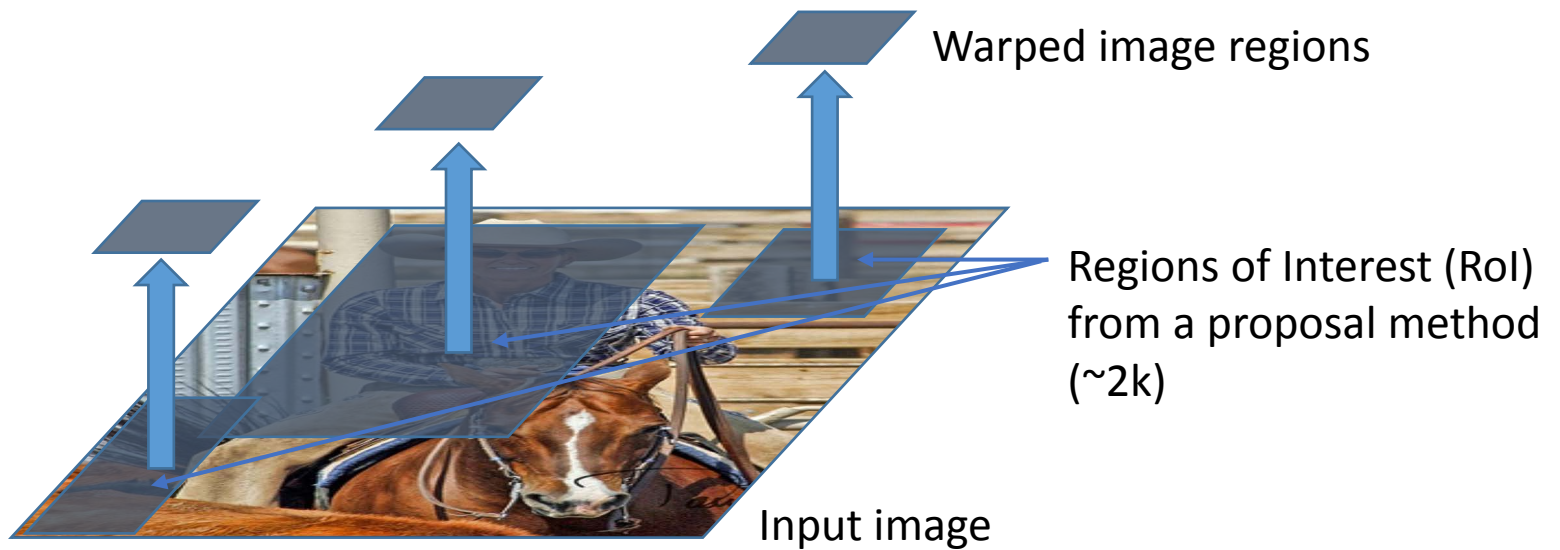
# Slow R-CNN



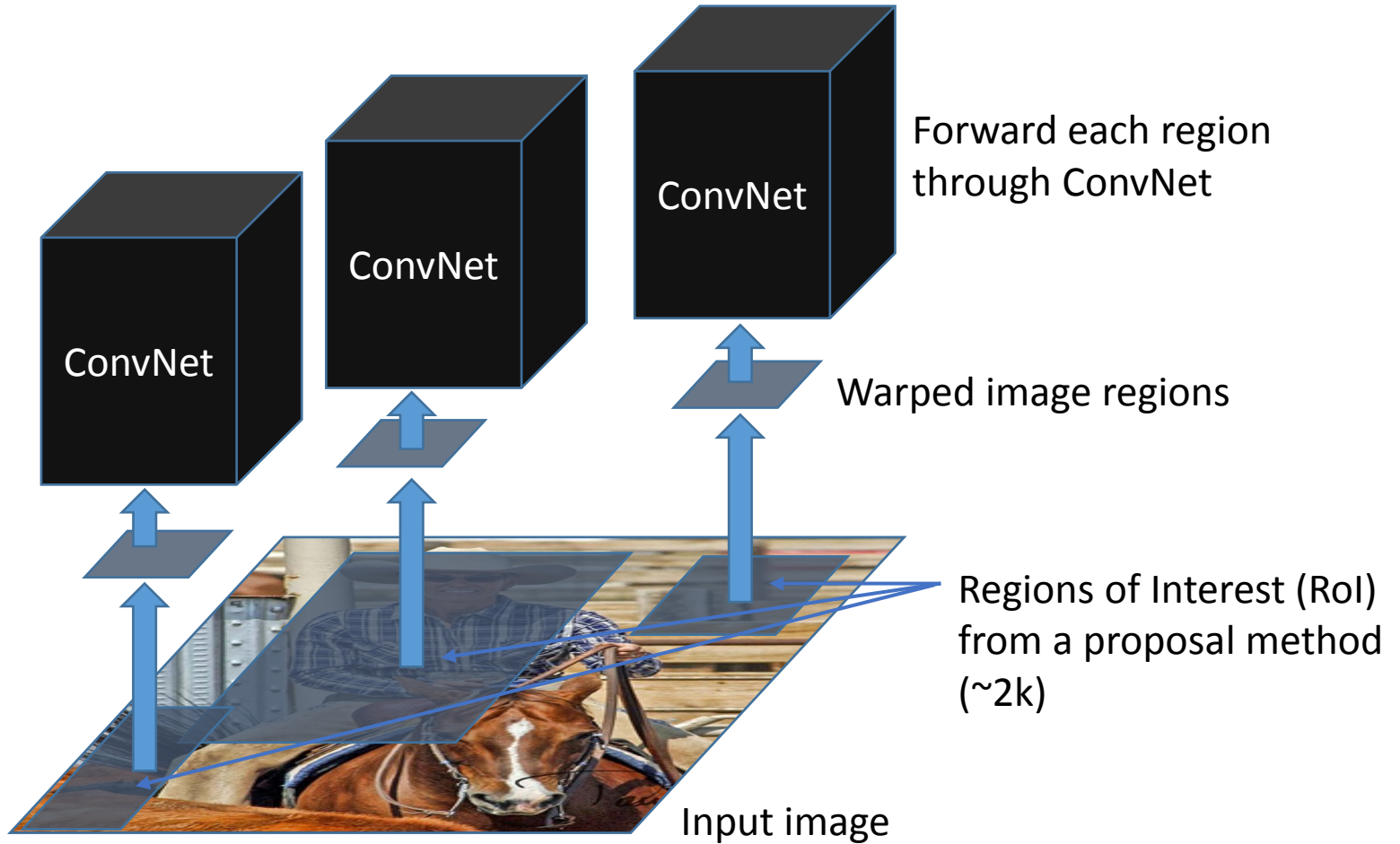
Regions of Interest (RoI)  
from a proposal method  
(~2k)

Input image

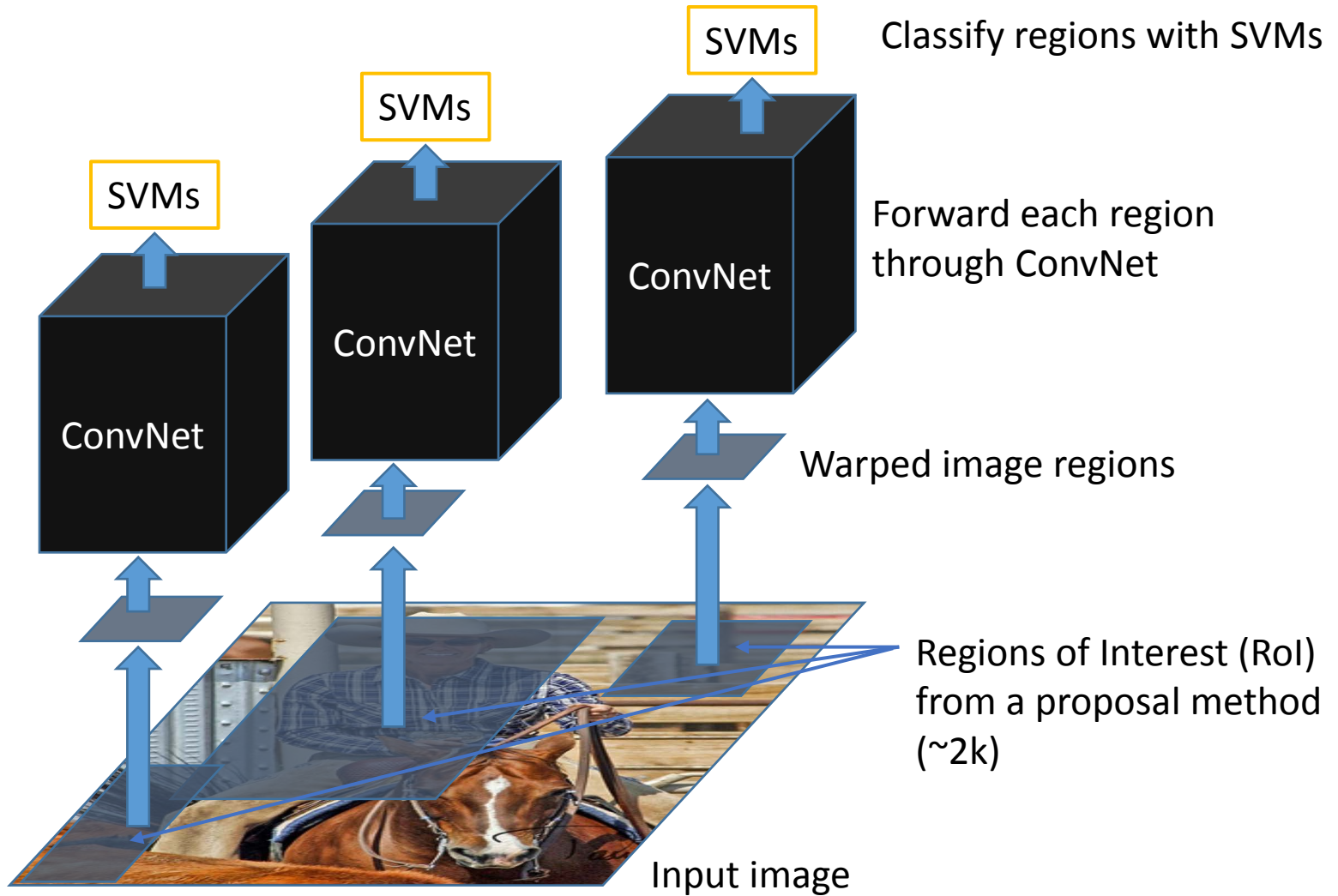
# Slow R-CNN



# Slow R-CNN

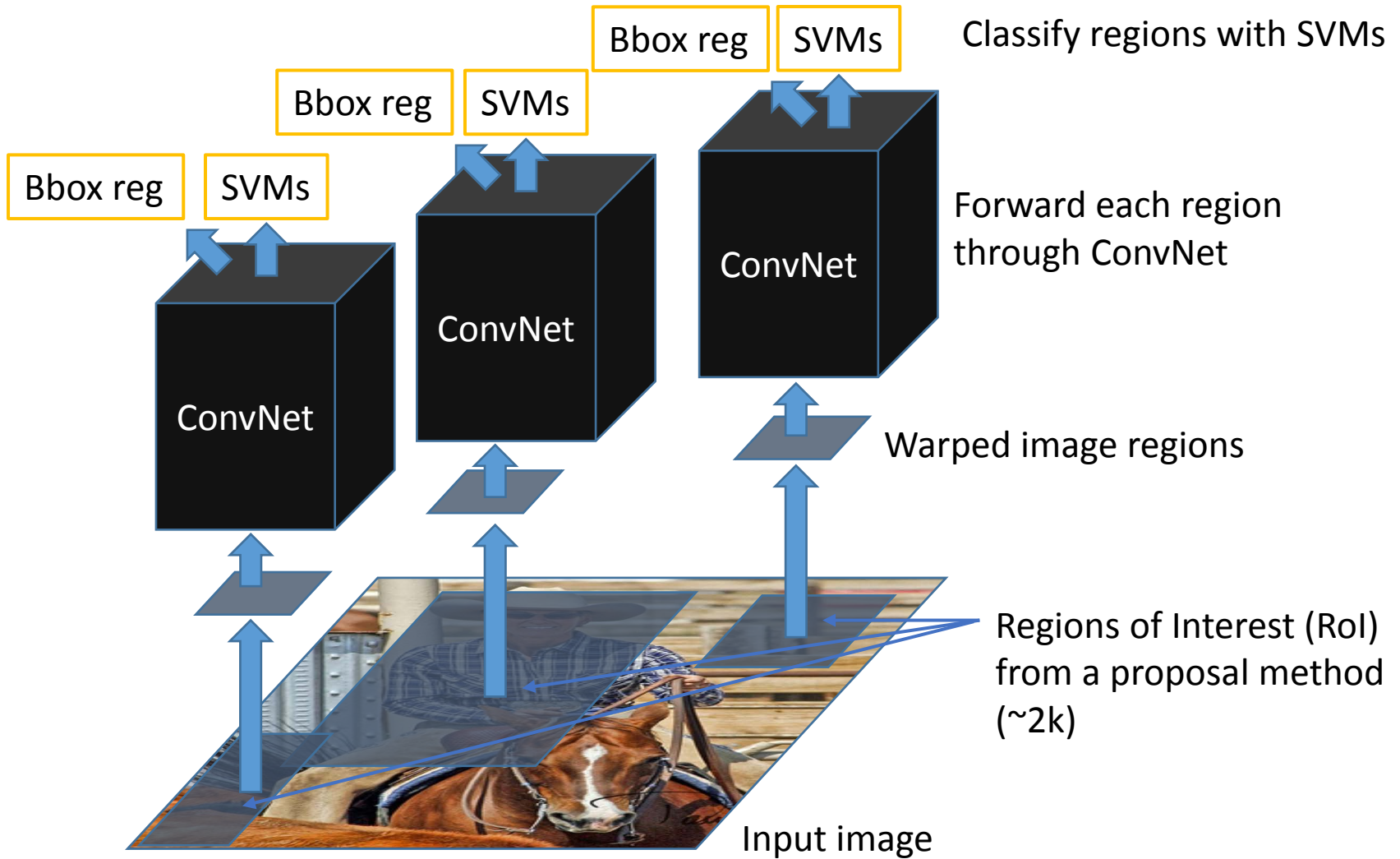


# Slow R-CNN





# Slow R-CNN



Post hoc component

What's wrong with slow R-CNN?

# What's wrong with slow R-CNN?

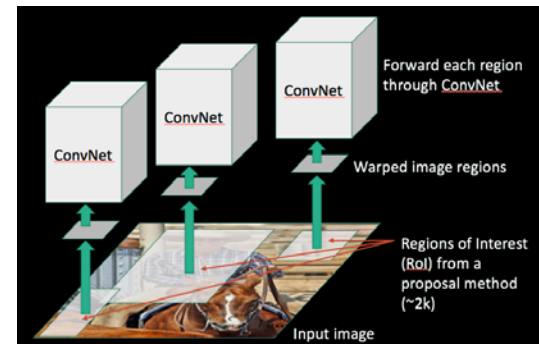
- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressors (squared loss)

# What's wrong with slow R-CNN?

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressors (squared loss)
- Training is slow (84h), takes a lot of disk space

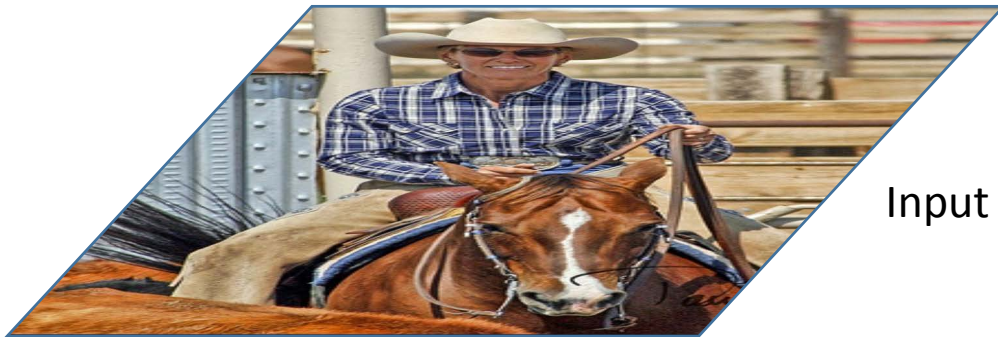
# What's wrong with slow R-CNN?

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
  - Fixed by SPP-net [He et al. ECCV14]



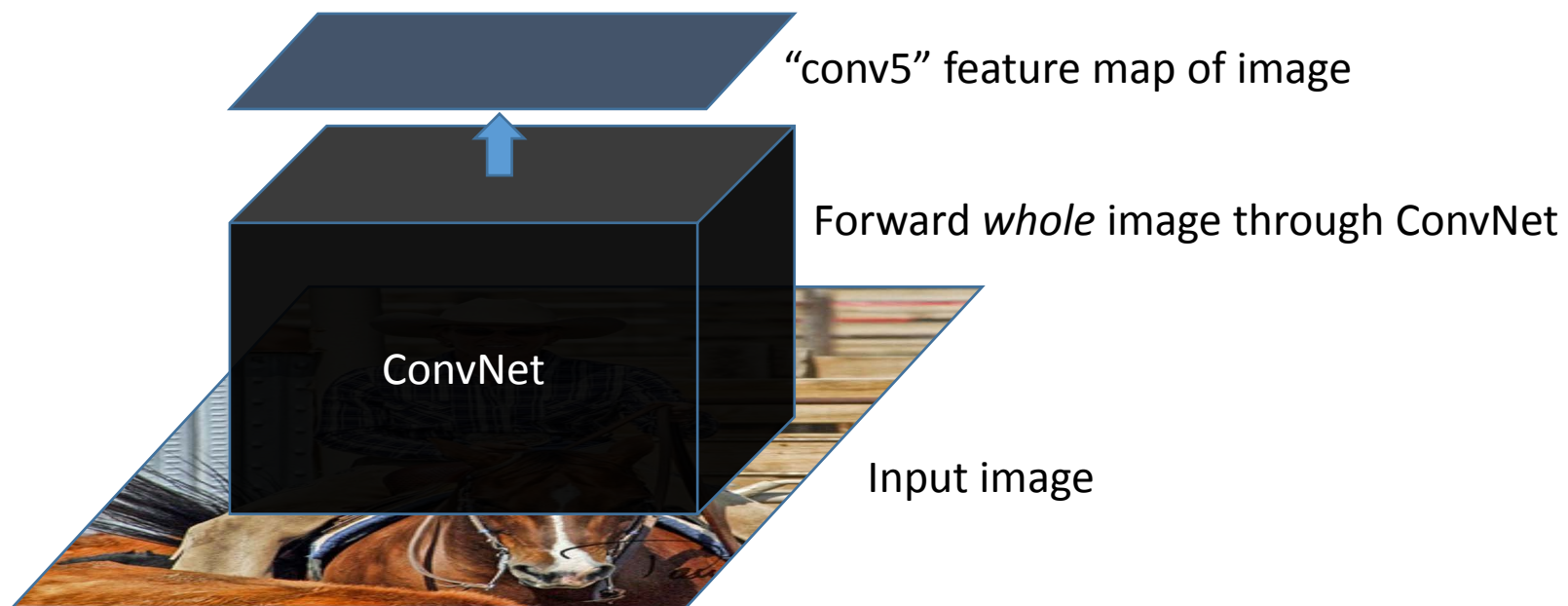
~2000 ConvNet forward passes per image

# SPP-net

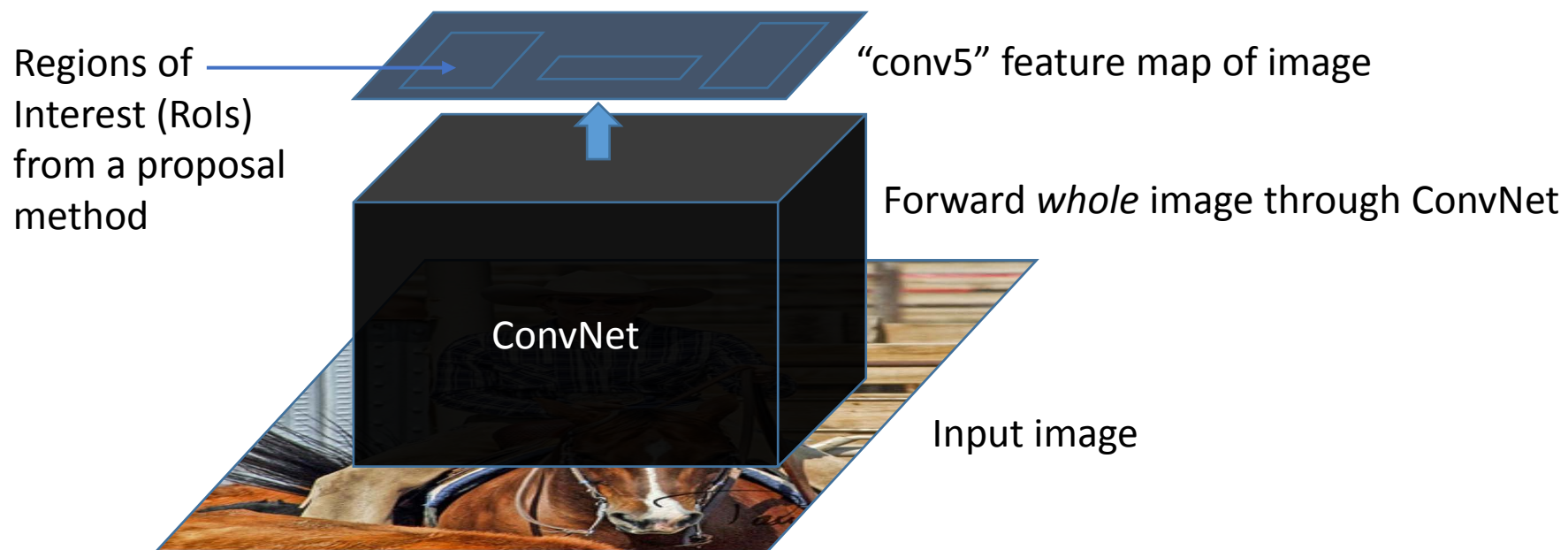


Input image

# SPP-net

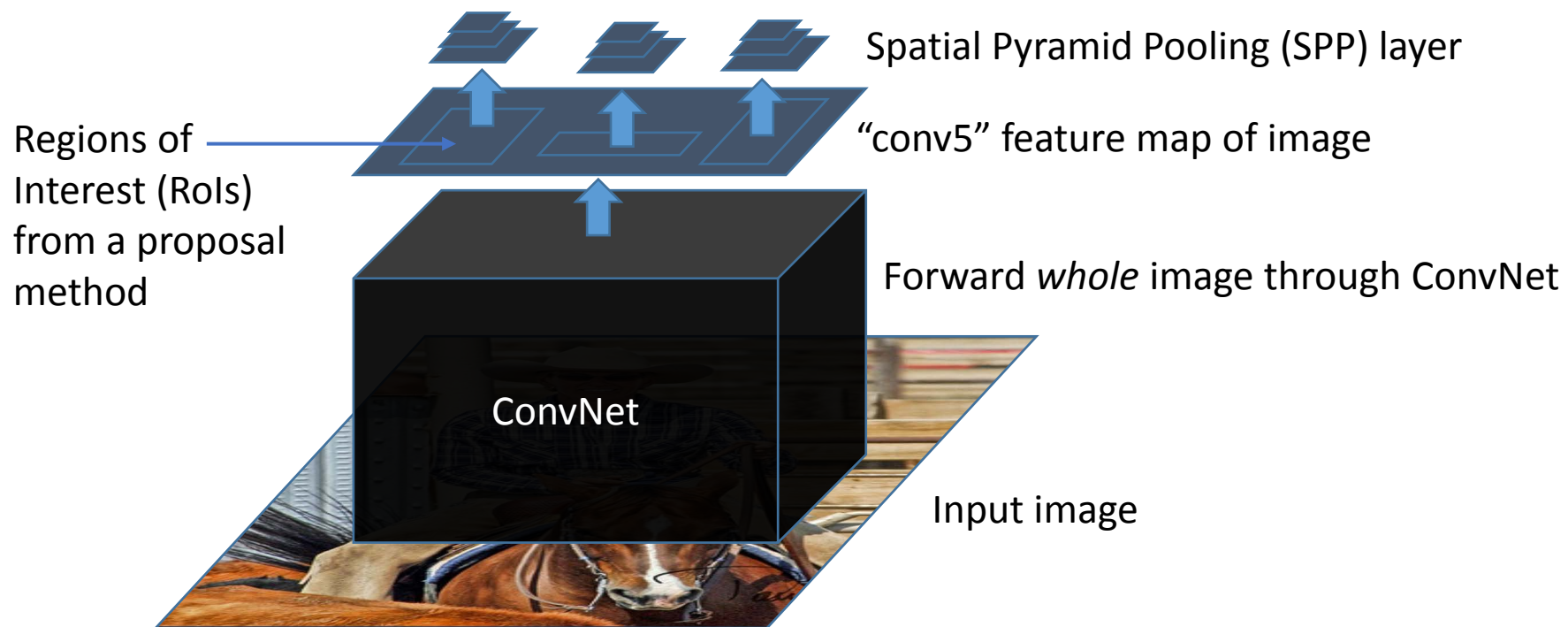


# SPP-net

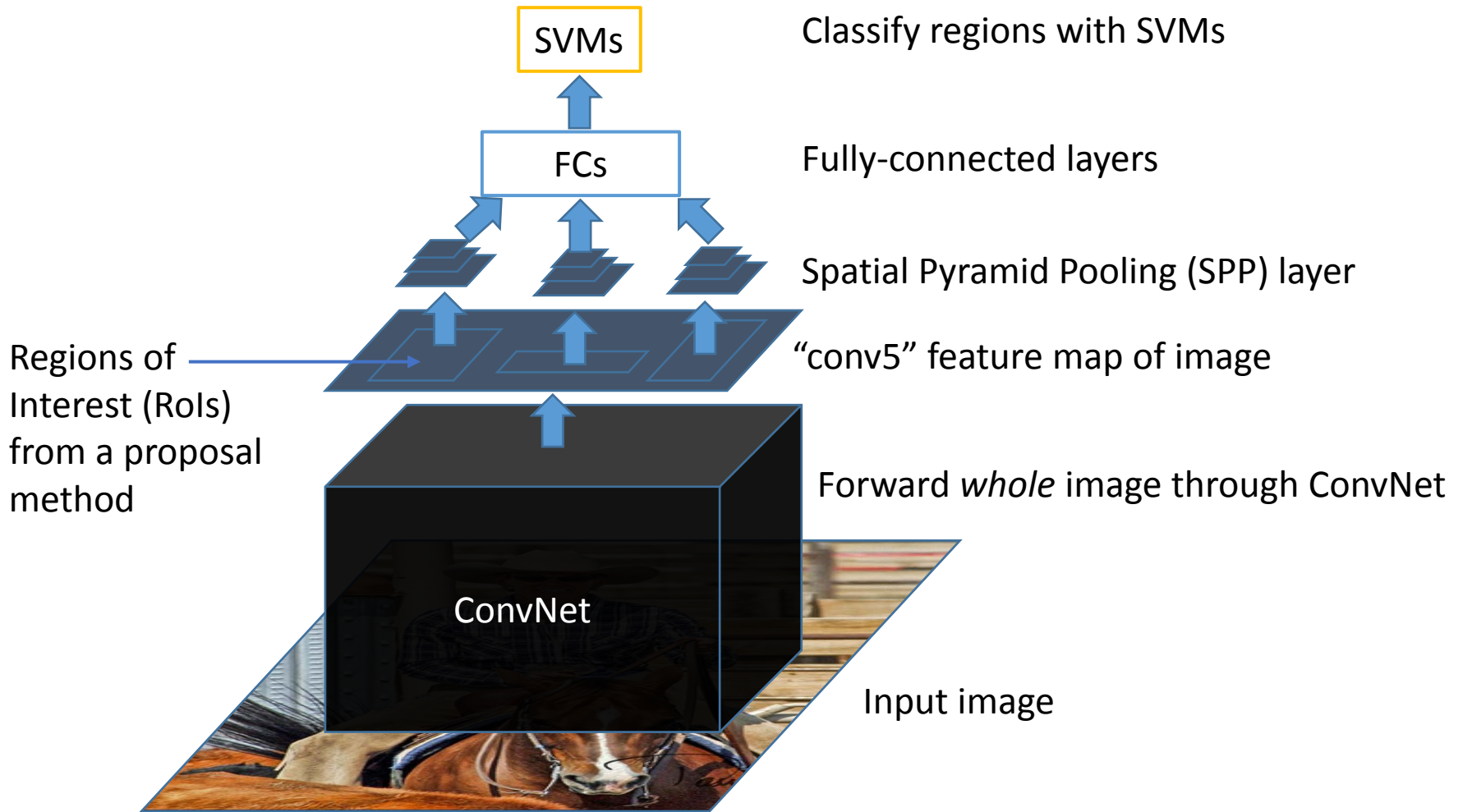




# SPP-net

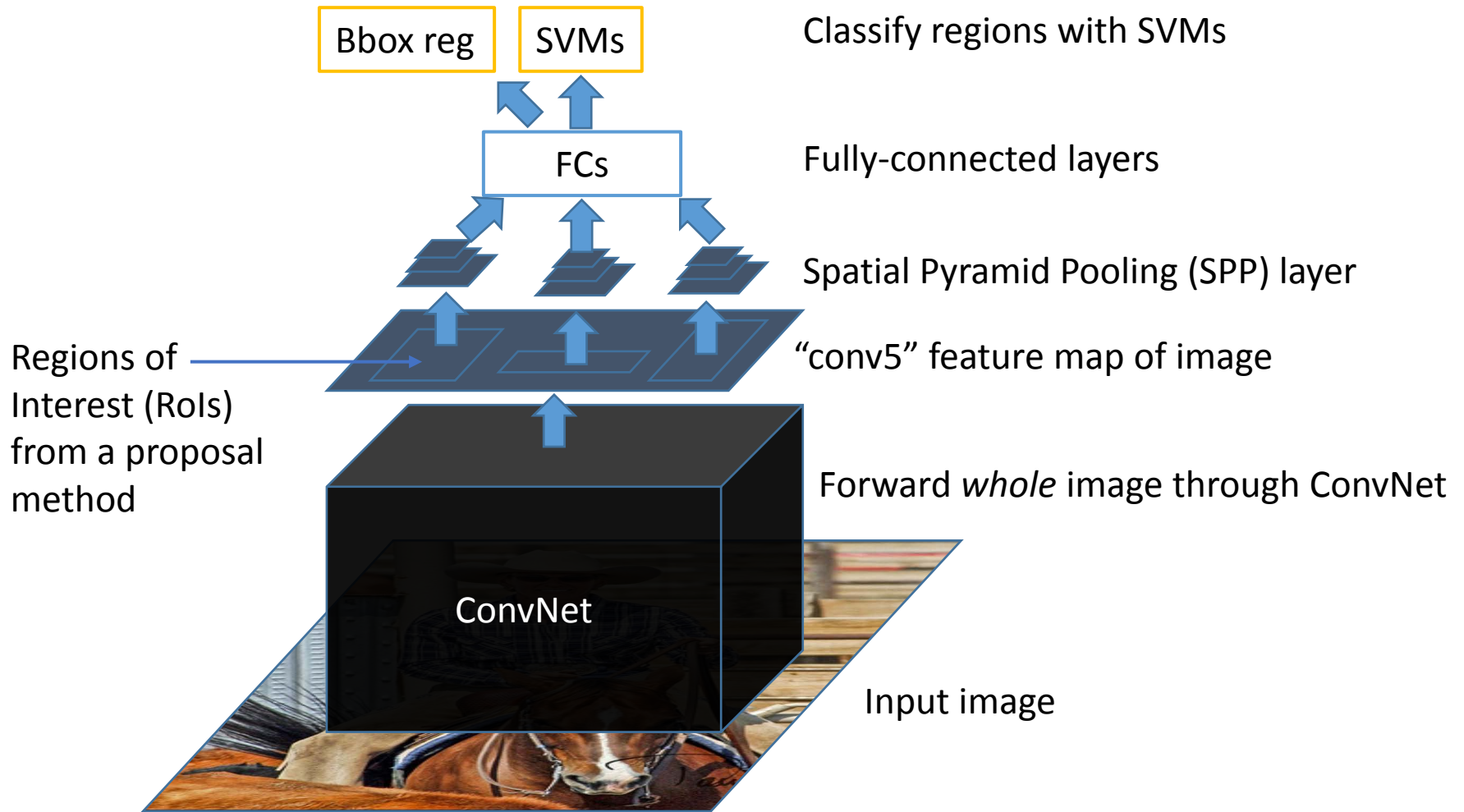


# SPP-net



Post hoc component

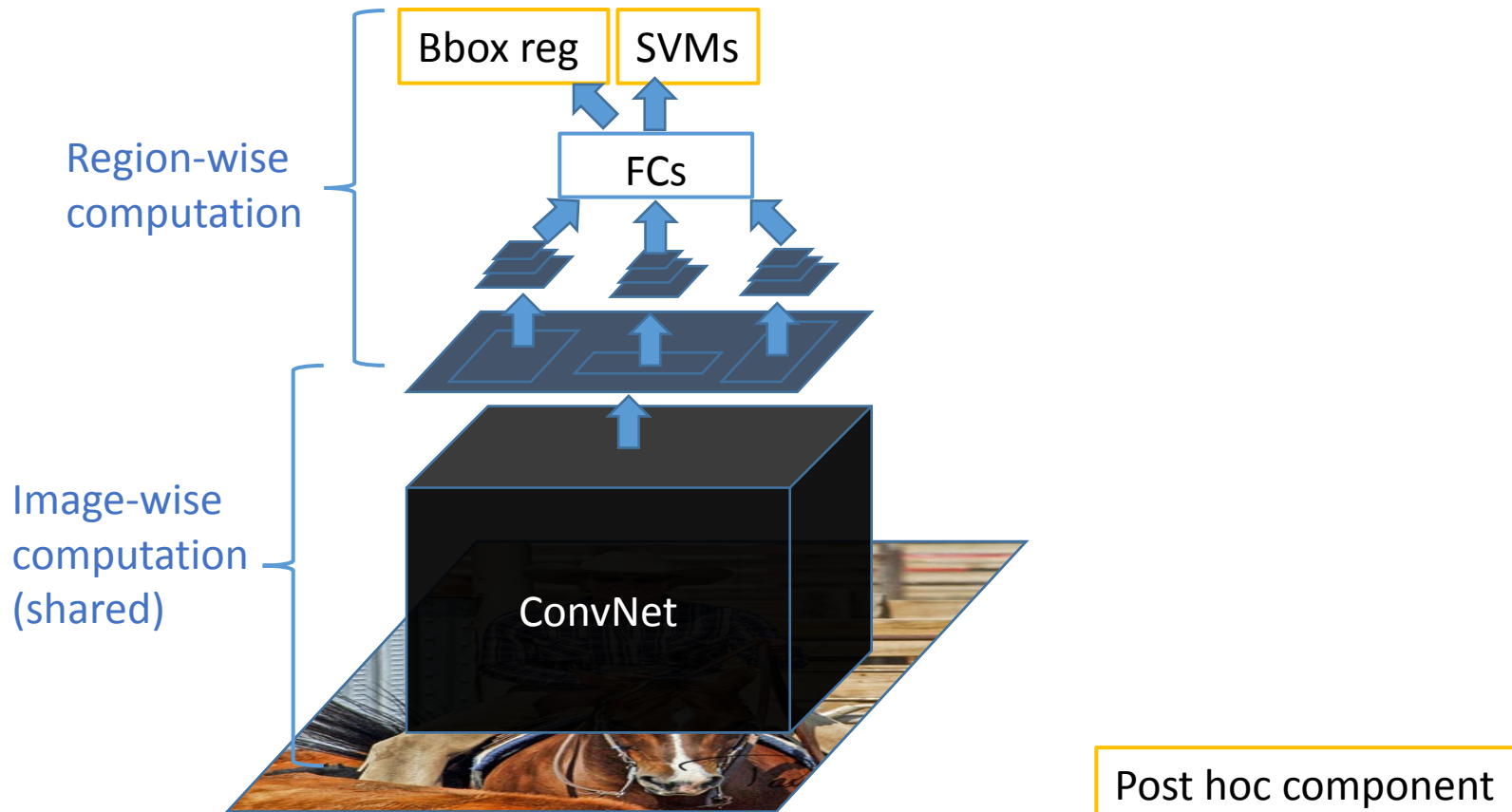
# SPP-net



Post hoc component

# What's good about SPP-net?

- Fixes one issue with R-CNN: makes testing fast



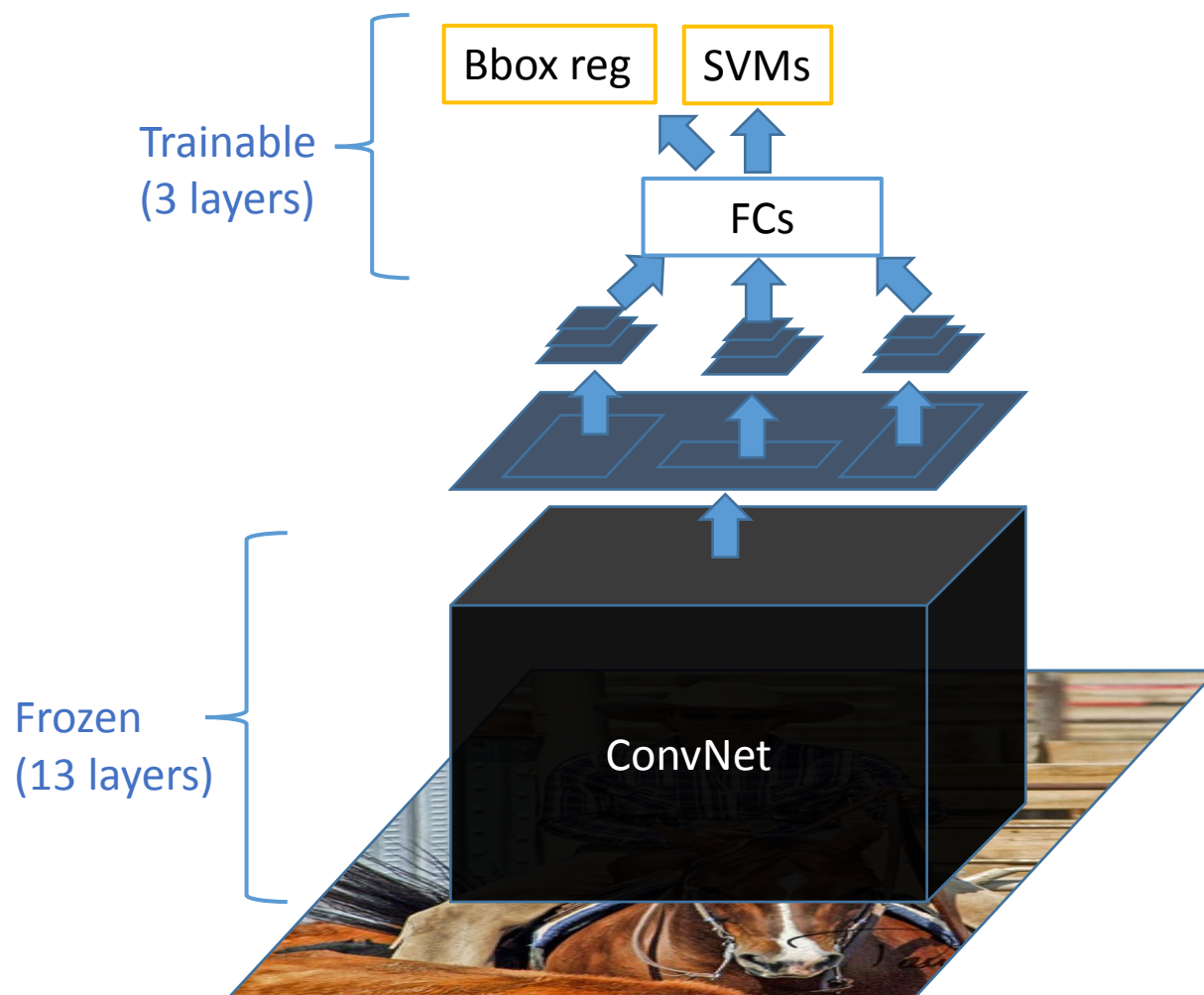
# What's wrong with SPP-net?

- Inherits the rest of R-CNN's problems
  - Ad hoc training objectives
  - Training is slow (25h), takes a lot of disk space

# What's wrong with SPP-net?

- Inherits the rest of R-CNN's problems
  - Ad hoc training objectives
  - Training is slow (though faster), takes a lot of disk space
- Introduces a new problem: **cannot update parameters below SPP layer during training**

# SPP-net: the main limitation



# Fast R-CNN

- Fast test-time, like SPP-net



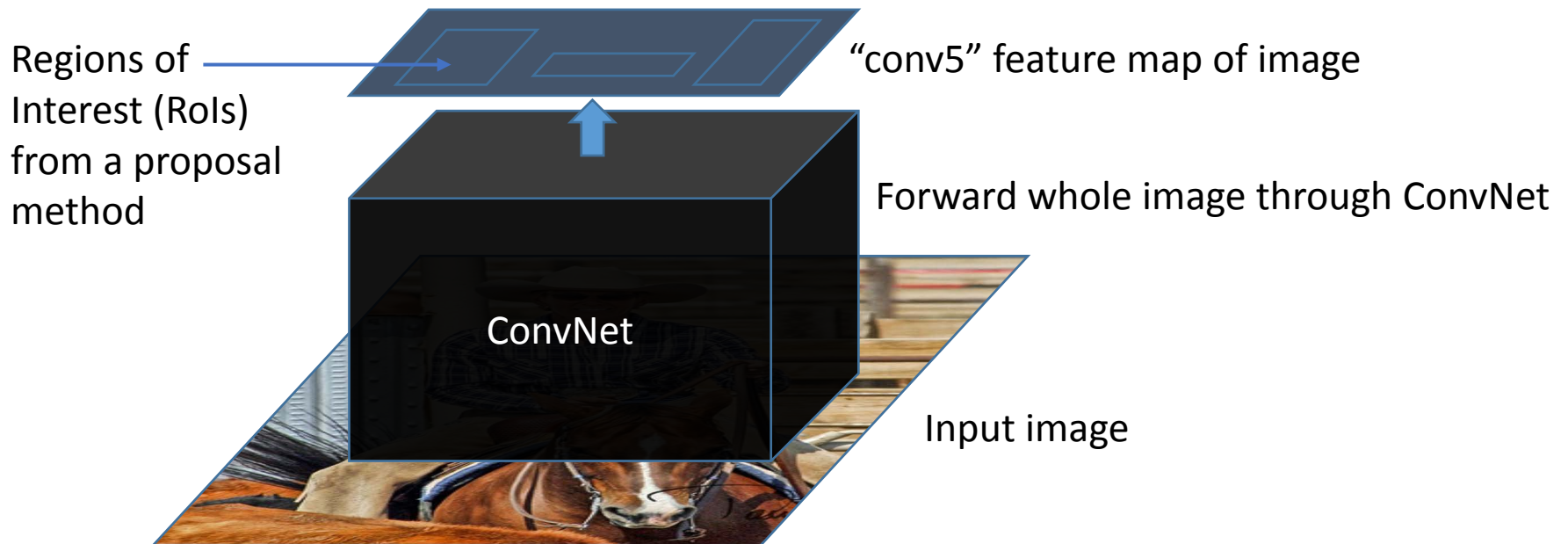
# Fast R-CNN

- Fast test-time, like SPP-net
- One network, trained in one stage

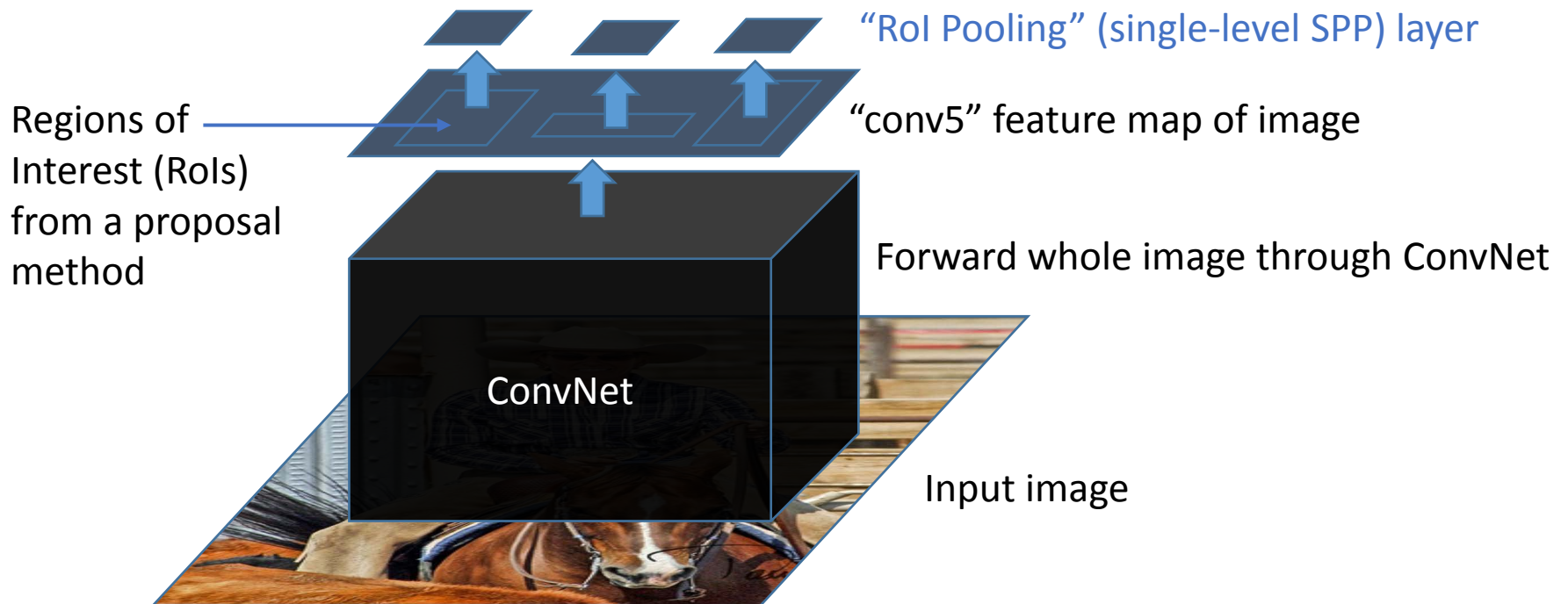
# Fast R-CNN

- Fast test-time, like SPP-net
- One network, trained in one stage
- Higher mean average precision than slow R-CNN and SPP-net

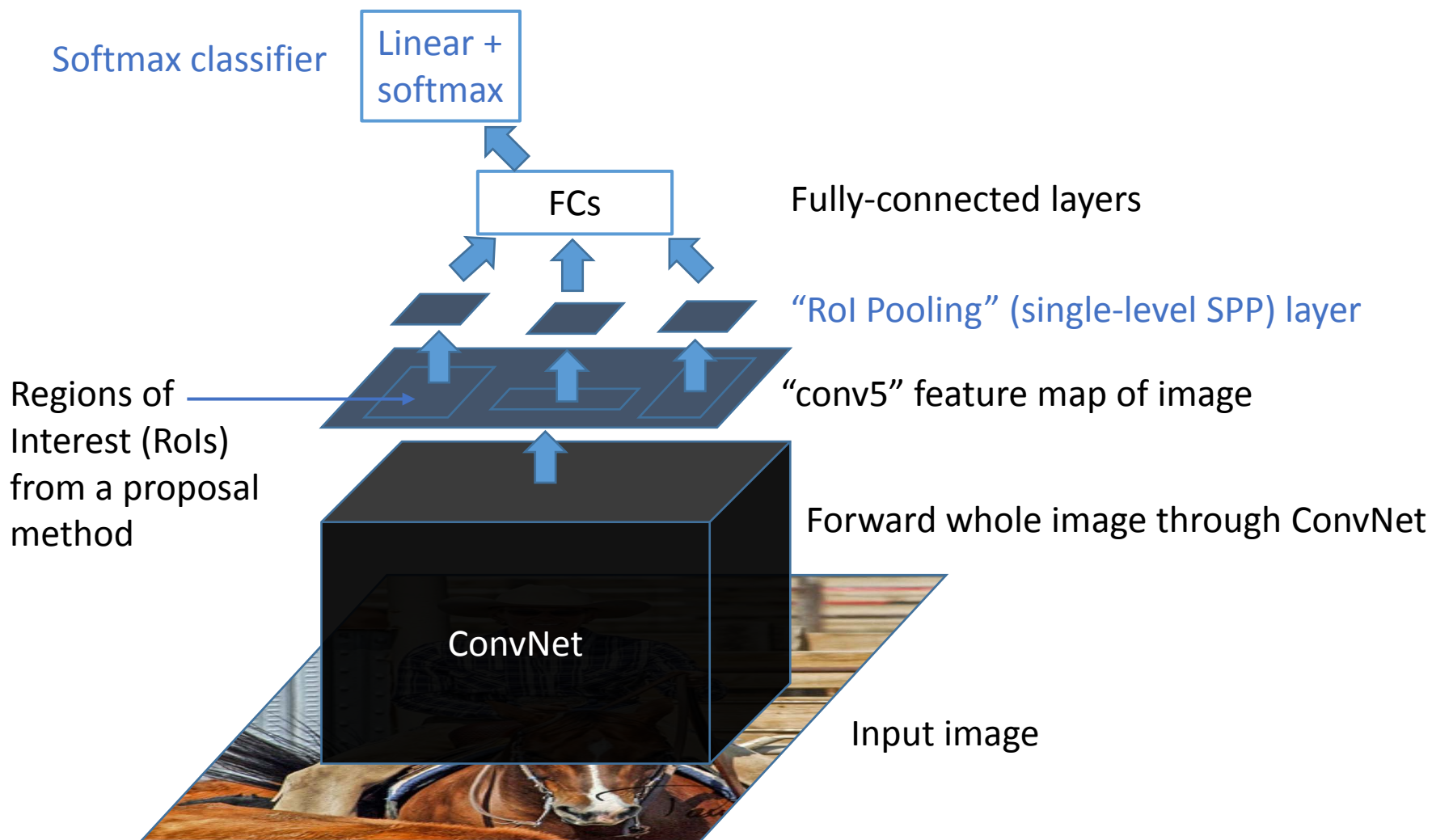
# Fast R-CNN (test time)



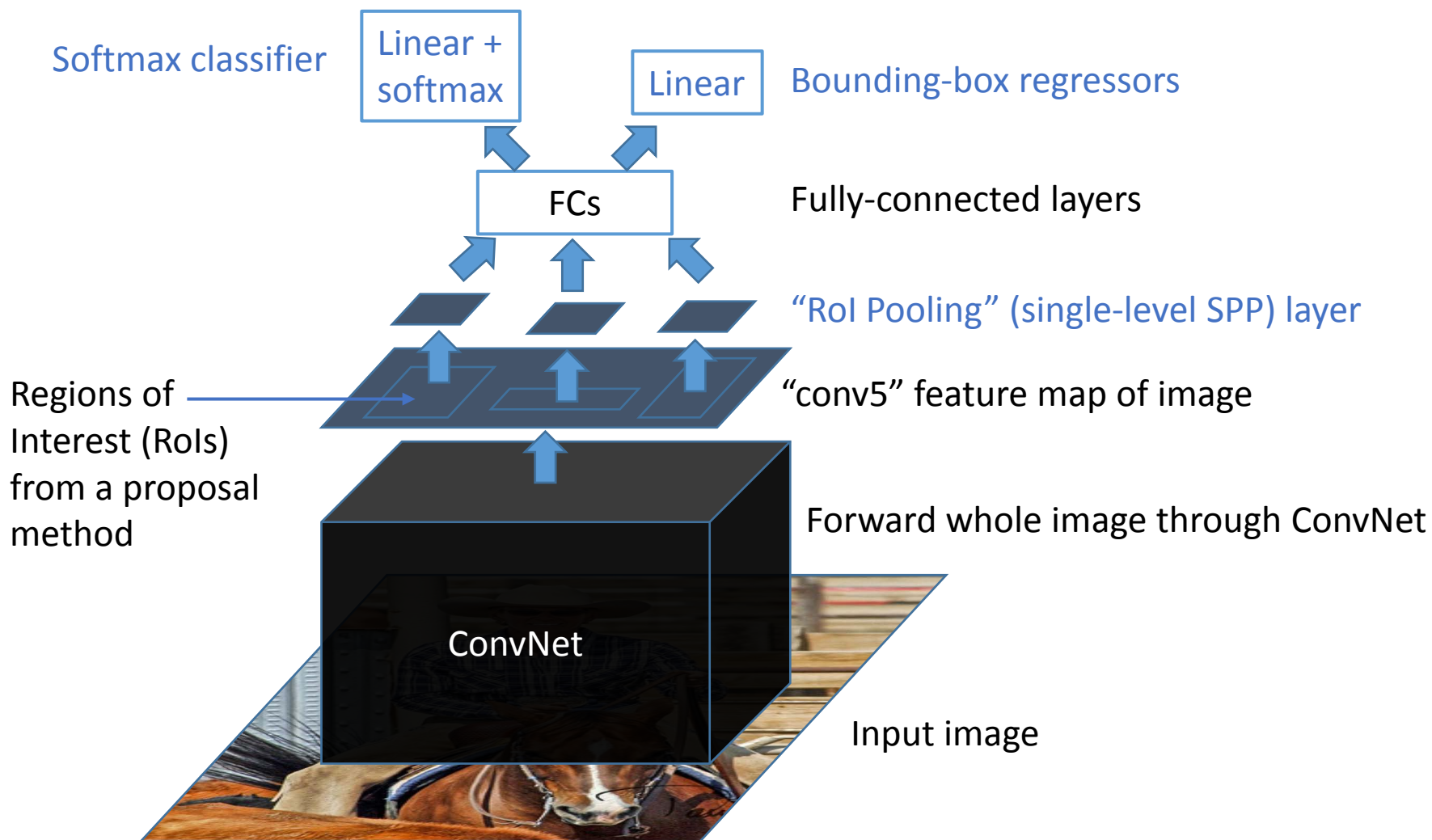
# Fast R-CNN (test time)



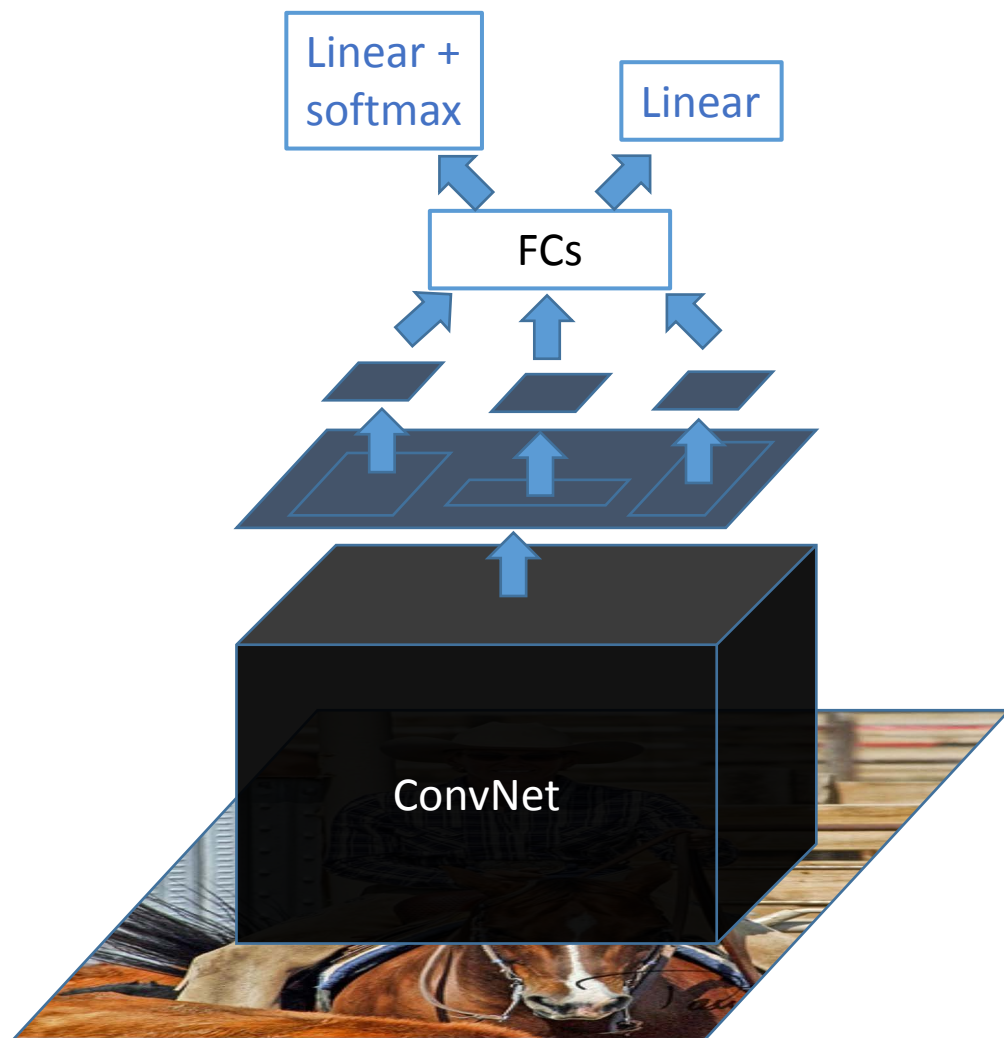
# Fast R-CNN (test time)



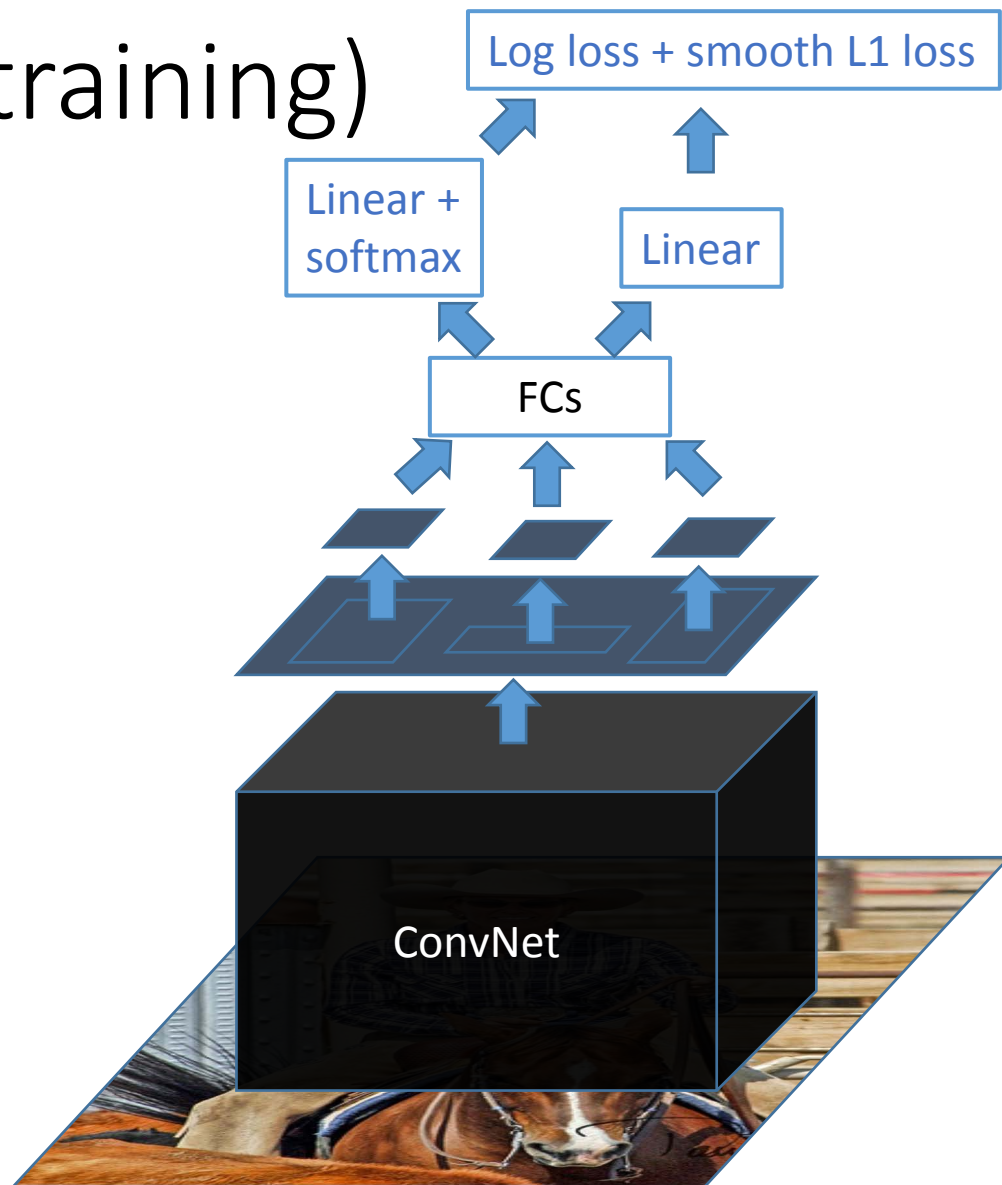
# Fast R-CNN (test time)



# Fast R-CNN (training)



# Fast R-CNN (training)



Multi-task loss



# Main results

	Fast R-CNN	R-CNN [1]	SPP-net [2]
Train time (h)	<b>9.5</b>	84	25
- Speedup	<b>8.8x</b>	1x	3.4x
Test time / image	<b>0.32s</b>	47.0s	2.3s
Test speedup	<b>146x</b>	1x	20x
mAP	<b>66.9%</b>	66.0%	63.1%

Timings exclude object proposal time, which is equal for all methods.  
All methods use VGG16 from Simonyan and Zisserman.

[1] Girshick et al. CVPR14.

[2] He et al. ECCV14.

# Main results

	Fast R-CNN	R-CNN [1]	SPP-net [2]
Train time (h)	9.5	84	25
- Speedup	8.8x	1x	3.4x
Test time / image	<b>0.32s</b>	47.0s	2.3s
Test speedup	<b>146x</b>	1x	20x
mAP	66.9%	66.0%	63.1%

Timings exclude object proposal time, which is equal for all methods.  
All methods use VGG16 from Simonyan and Zisserman.

[1] Girshick et al. CVPR14.

[2] He et al. ECCV14.

# Main results

	Fast R-CNN	R-CNN [1]	SPP-net [2]
Train time (h)	9.5	84	25
- Speedup	8.8x	1x	3.4x
Test time / image	0.32s	47.0s	2.3s
Test speedup	146x	1x	20x
mAP	<b>66.9%</b>	66.0%	63.1%

Timings exclude object proposal time, which is equal for all methods.  
All methods use VGG16 from Simonyan and Zisserman.

[1] Girshick et al. CVPR14.

[2] He et al. ECCV14.