Introduction to Computer Vision

Instructors: Jean Ponce and Matthew Trager
jean.ponce@inria.fr, matthew.trager@cims.nyu.edu

TAs: Jiachen (Jason) Zhu and Sahar Siddiqui
jiachen.zhu@nyu.edu, ss12414@nyu.edu

Outline

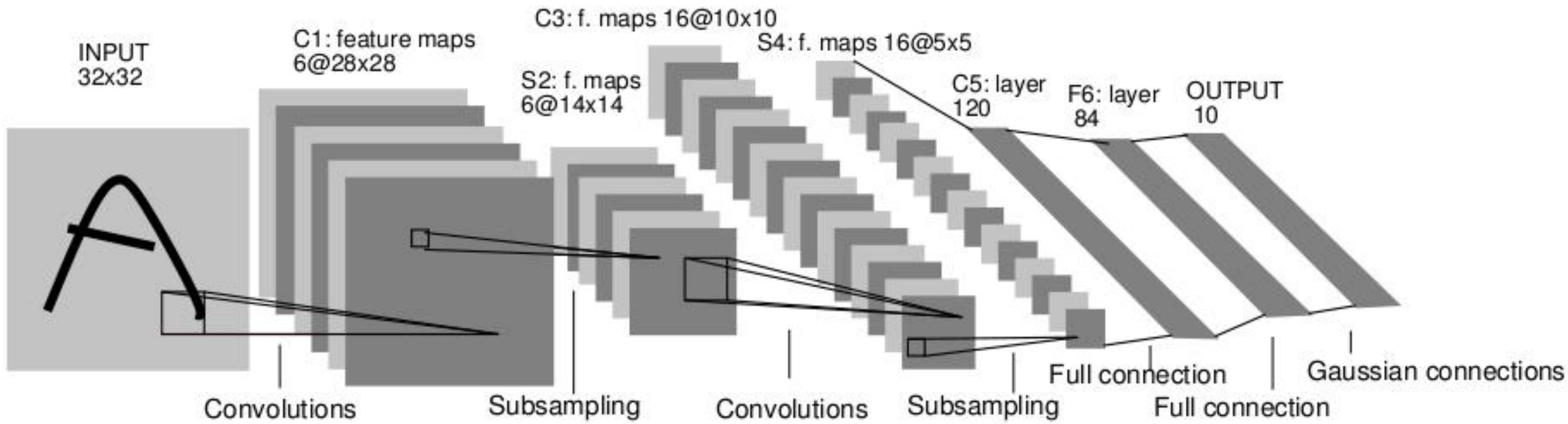
- Training neural networks
- Object detection with CNNs

Parametric supervised learning

- Training examples $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$
- A function class $\mathcal{F} = \{f_\theta : \mathcal{X} \to \mathcal{Y} \mid \theta \in \mathbb{R}^d\}$
- A loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$

- Goal: minimize the empirical risk
$$\hat{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$
- Hope that this "generalizes": if (X, Y) is a r.v., we would like to minimize
$$L(\theta) = \mathbb{E}[\ell(f_\theta(X), Y)]$$

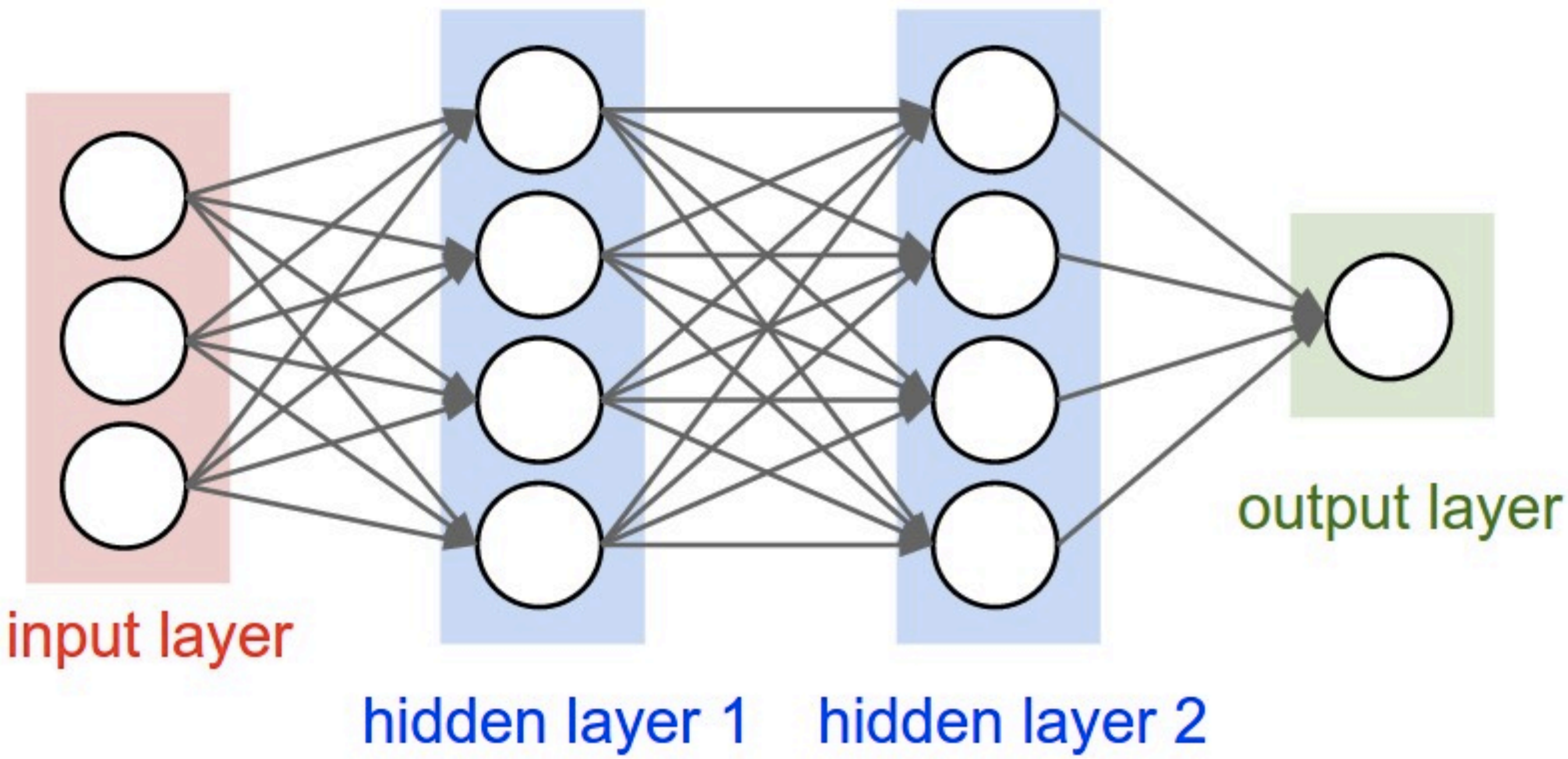
Neural networks



A neural network architecture describes a particular family of functions: $f_\theta : \mathscr{X} \to \mathscr{Y}$.

The parameters $\theta \in \mathbb{R}^d$ are the network's *weights*.

Feedforward NN (MLP)



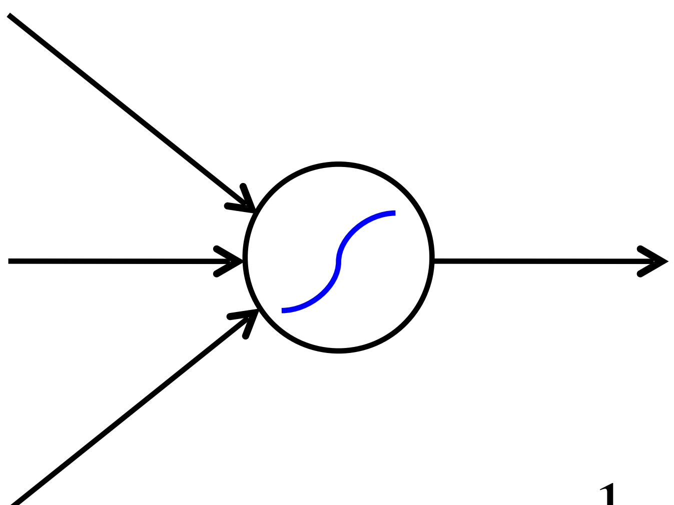
A feedforward NN is a composition of linear an non-linear functions, for example:

$$f_\theta(x) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_h}, \qquad f_\theta(x) = W_h \rho W_{h-1} \rho \ldots W_2 \rho W_1 x,$$

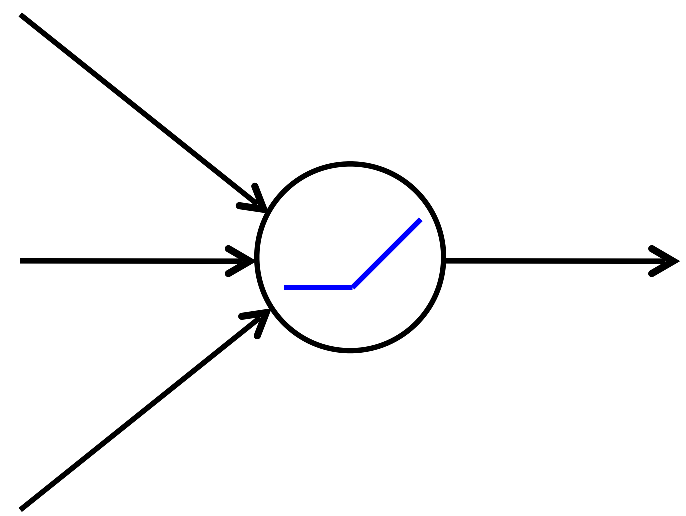
where $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $\theta = (W_h, \ldots, W_1)$, and ρ is a non-linear map acting coordinate-wise.

Non-linearity

- The nonlinearity ρ should be (almost everywhere) differentiable.

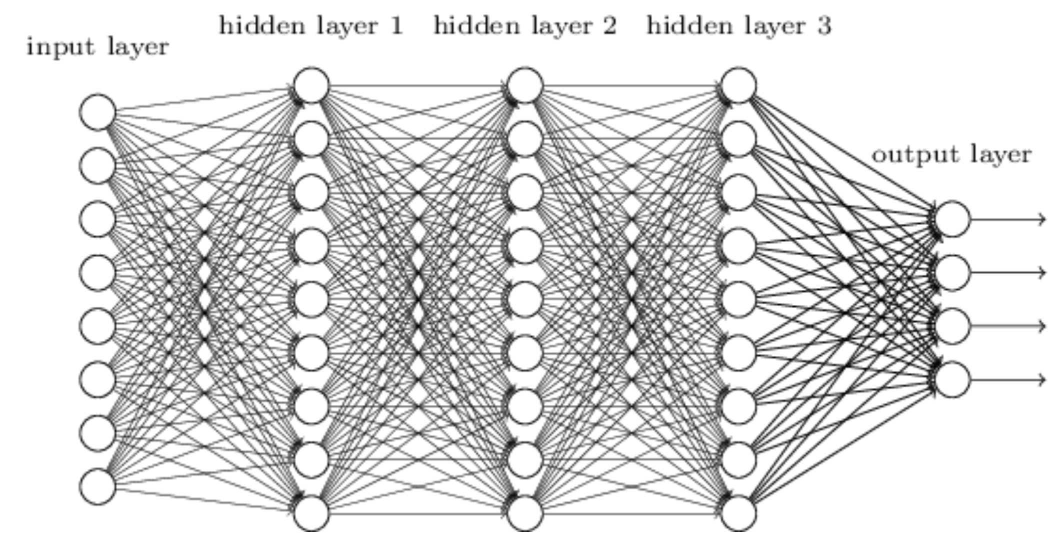


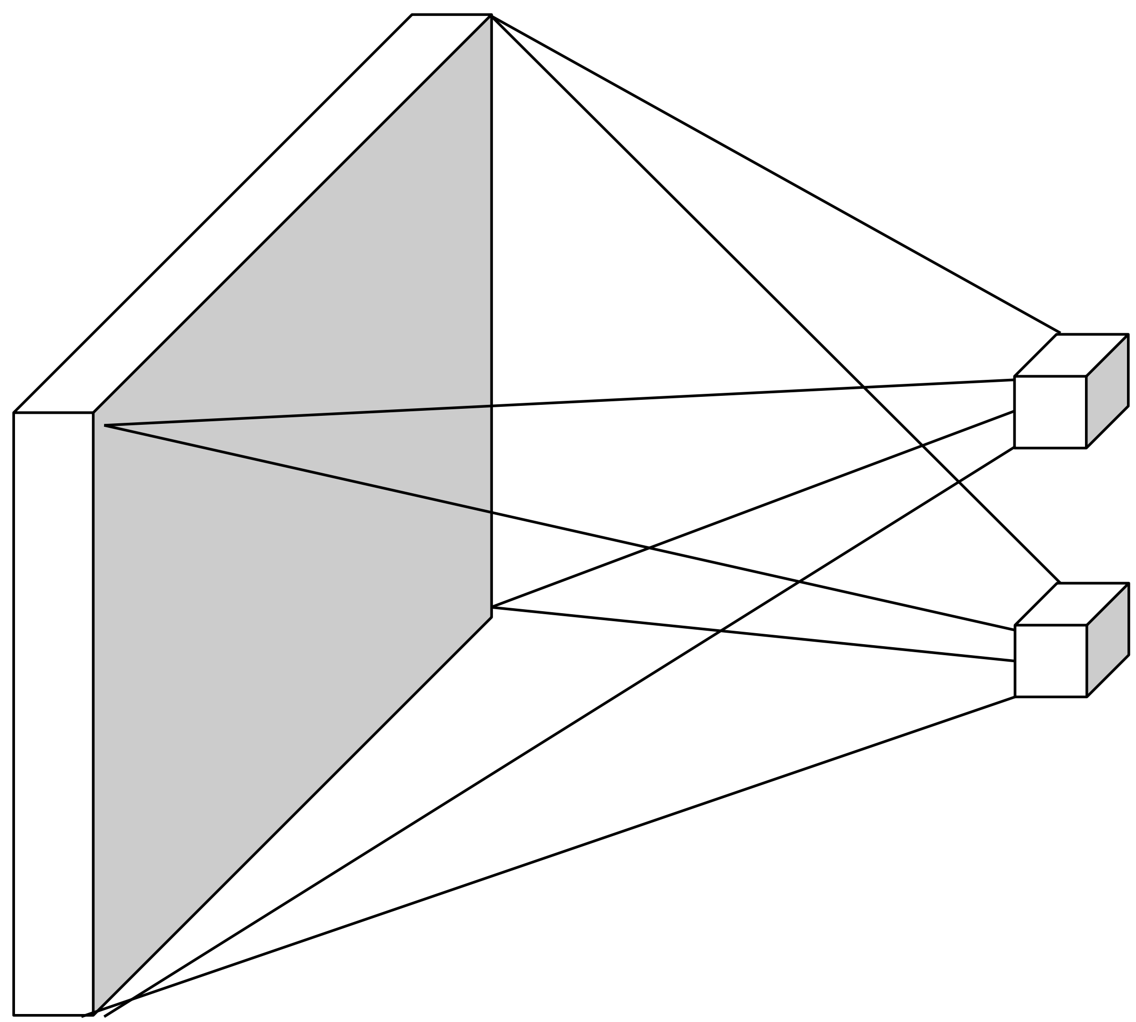
Sigmoid: $\rho(t) = \dfrac{1}{1 + e^{-t}}$



Rectified linear unit (ReLU): $\rho(t) = \max(0, t)$

Neural networks for images



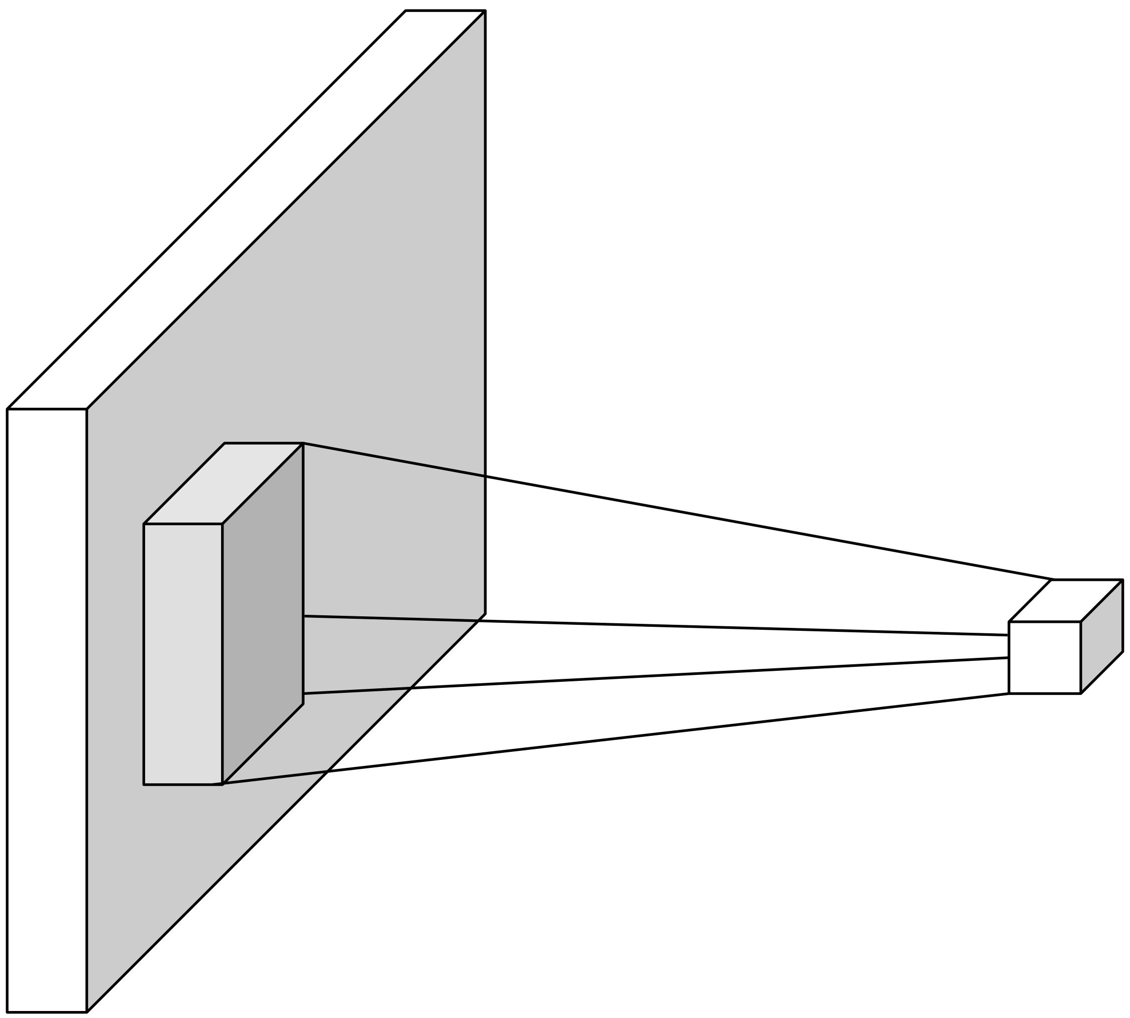


image

Fully connected layer

Source: S. Lazebnik

Neural networks for images

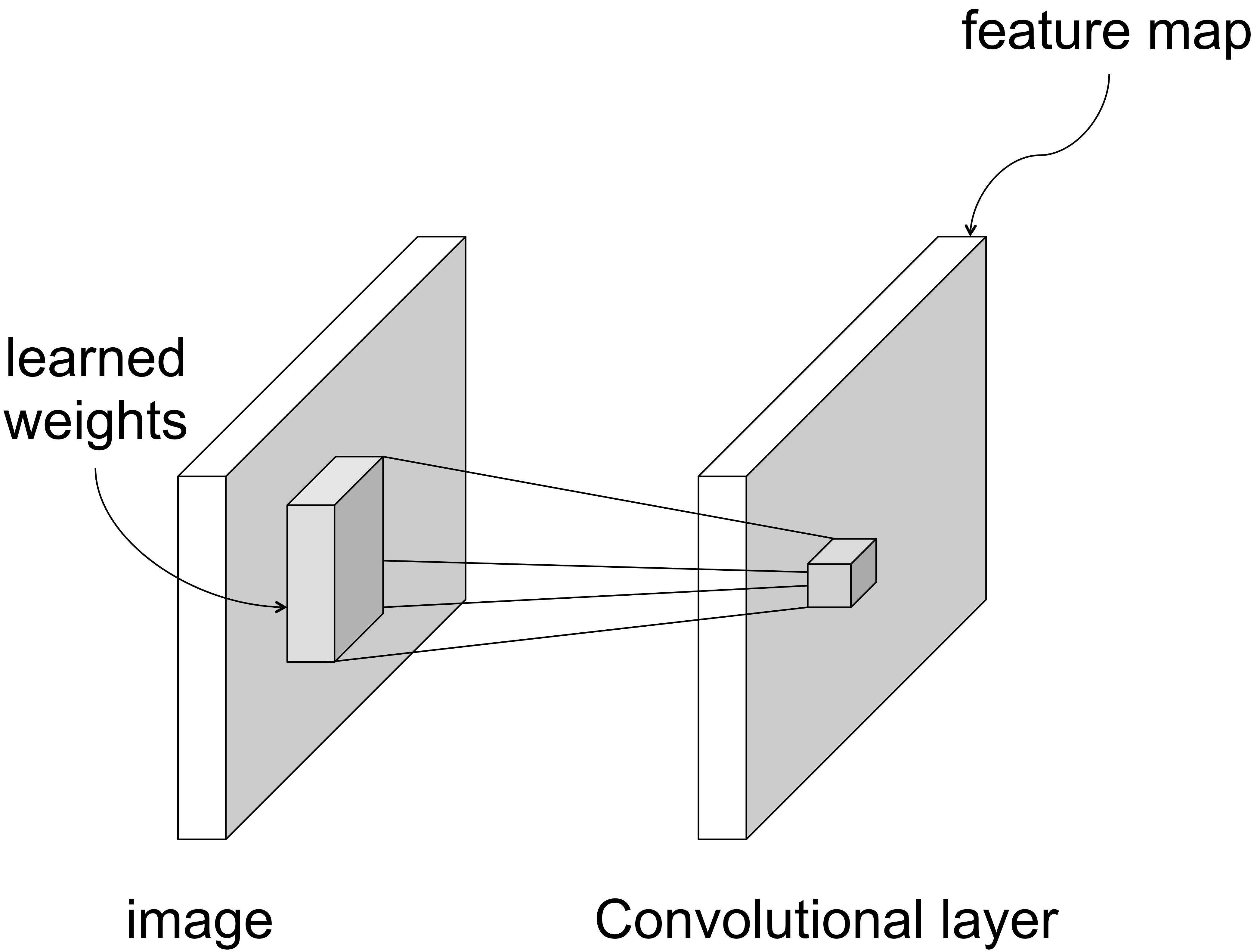


image

Convolutional layer

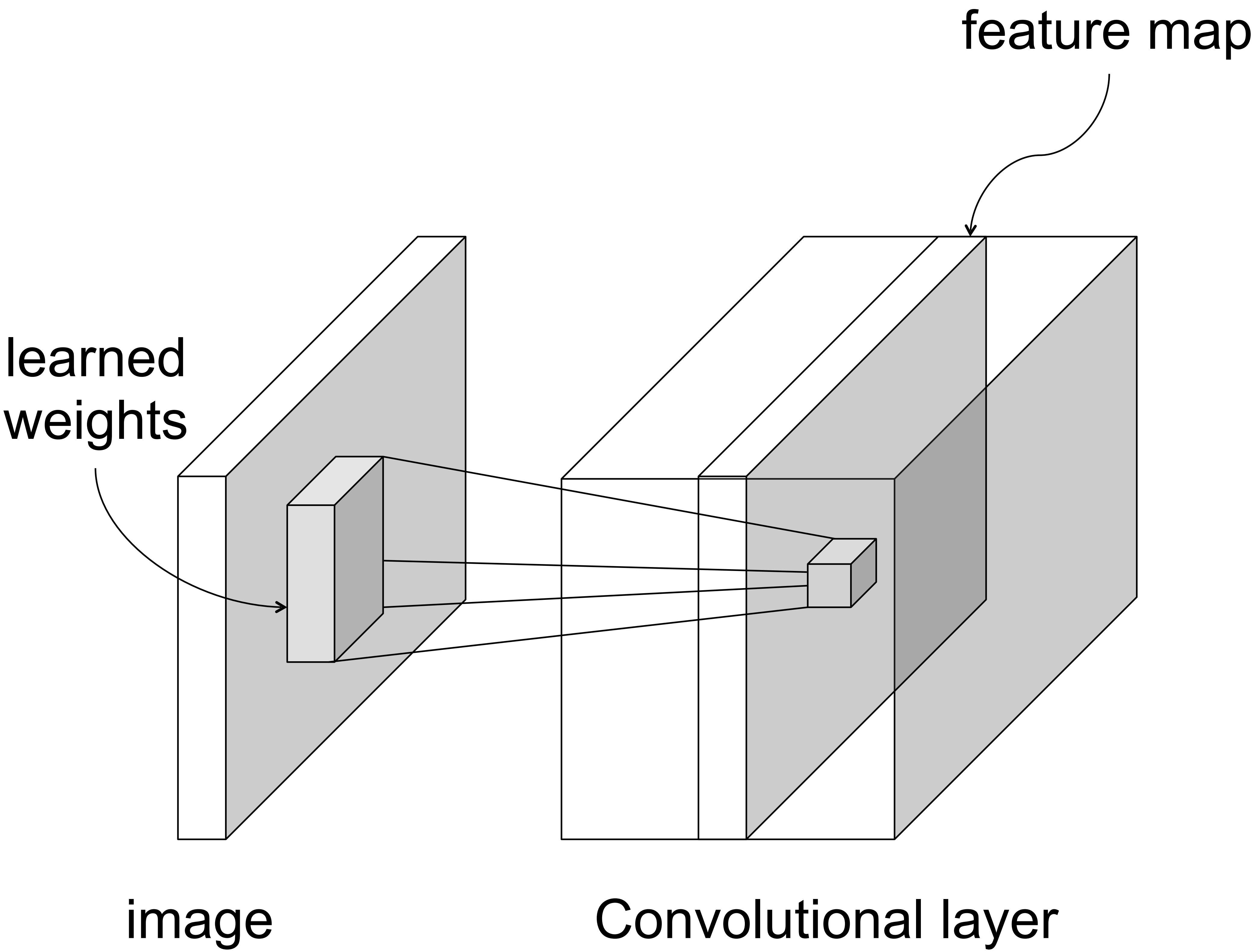
Source: S. Lazebnik

Neural networks for images



Source: S. Lazebnik

Neural networks for images

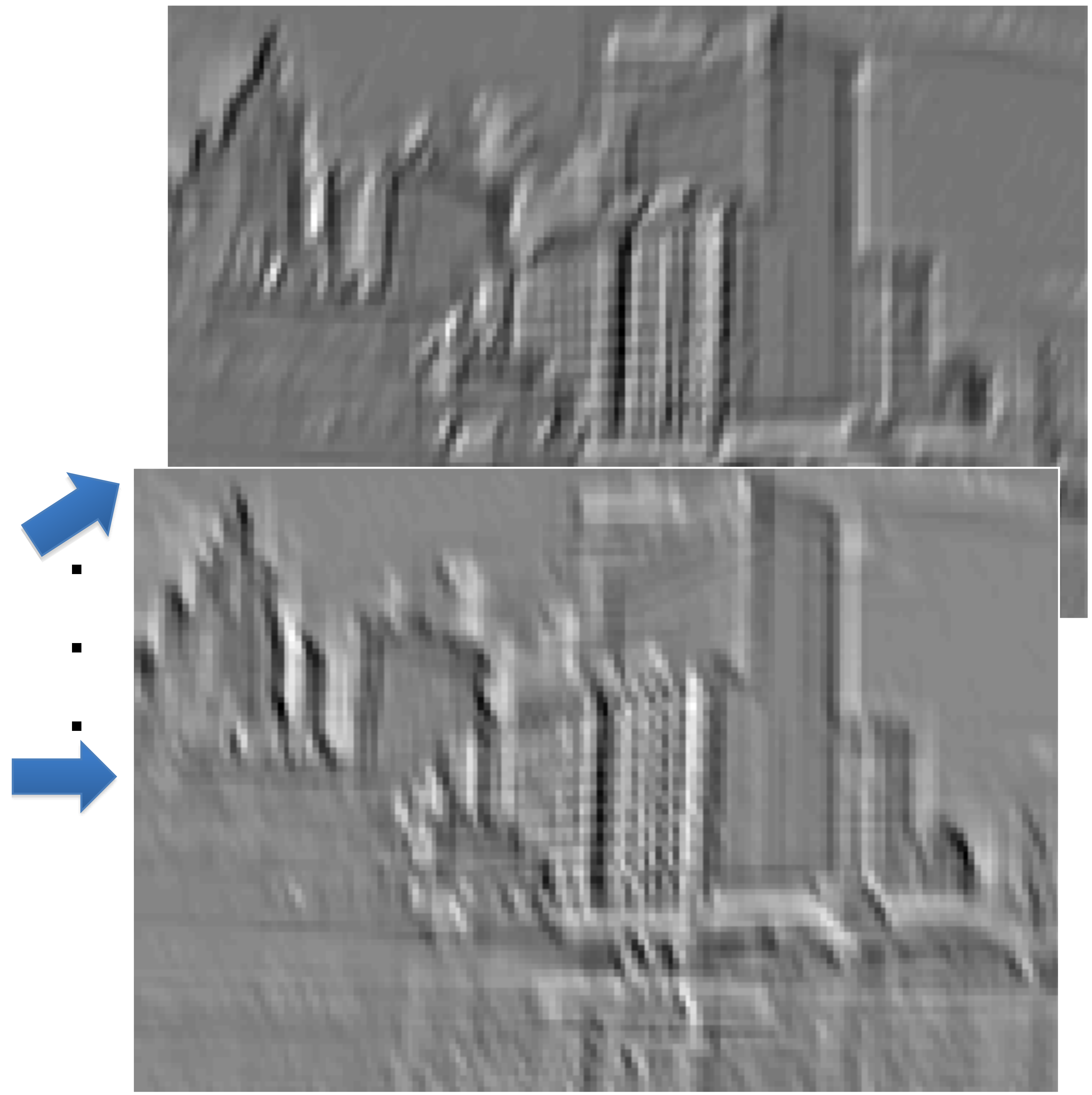


Source: S. Lazebnik

Convolution as feature extraction

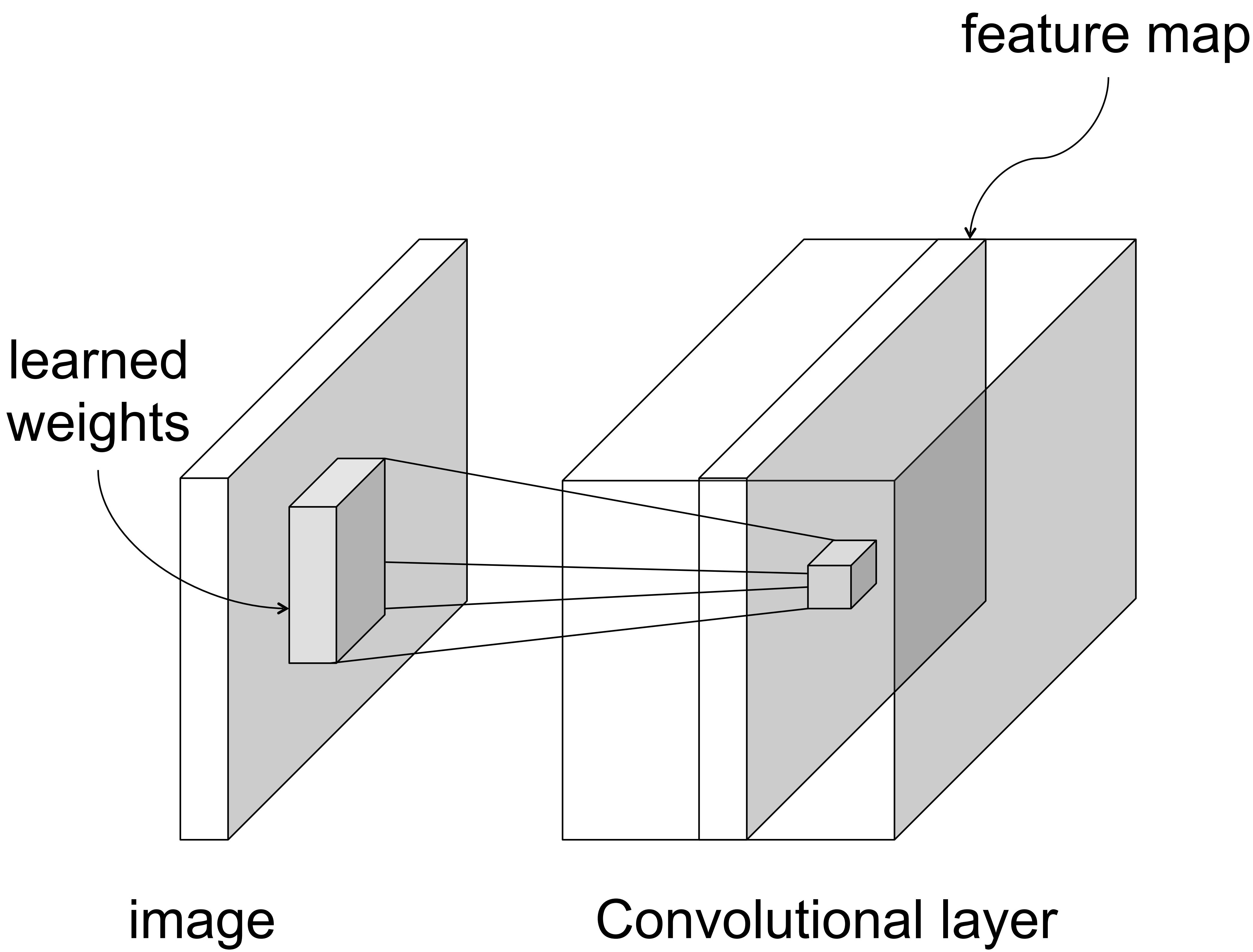
Input

Source: S. Lazebnik



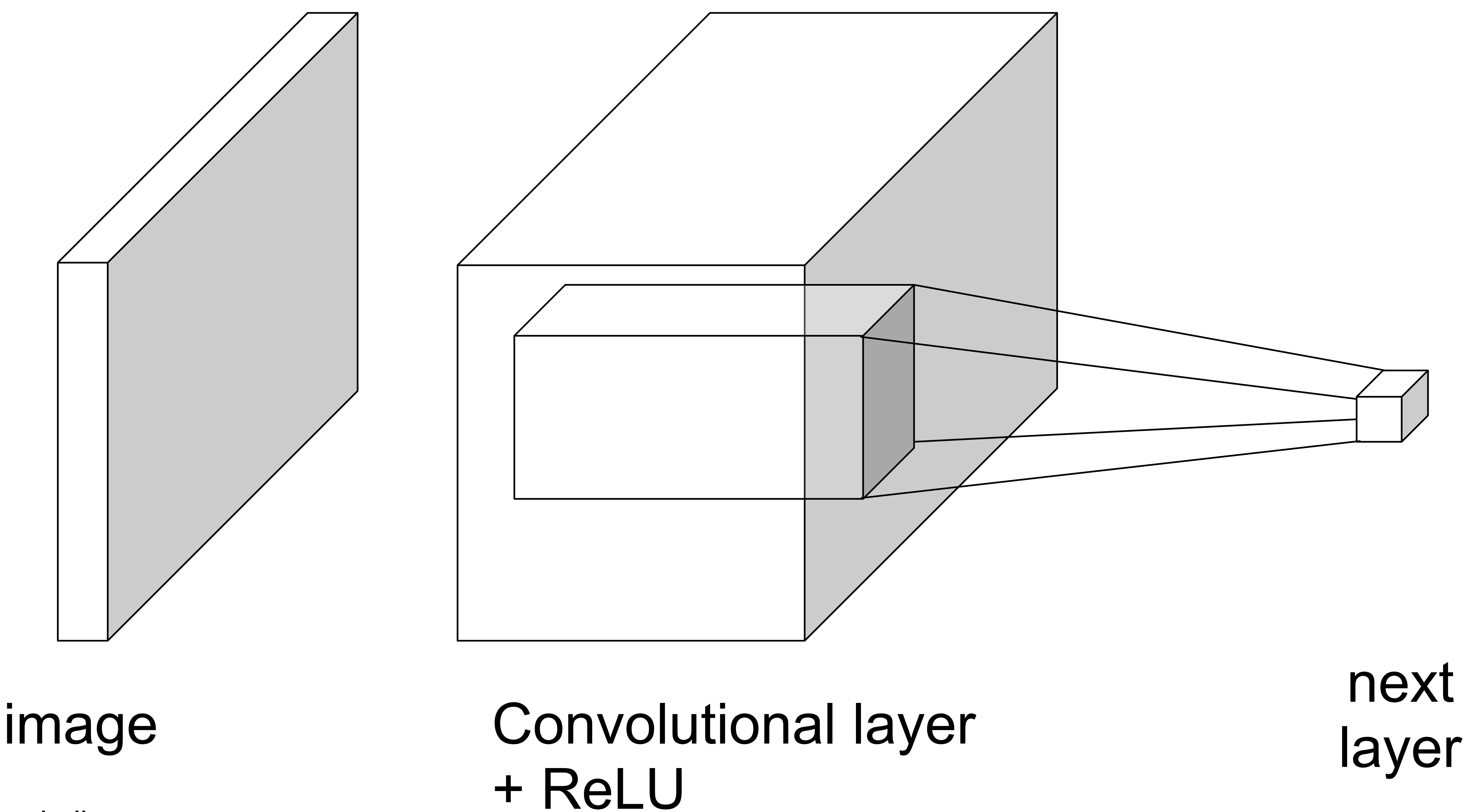
Feature Map

Neural networks for images



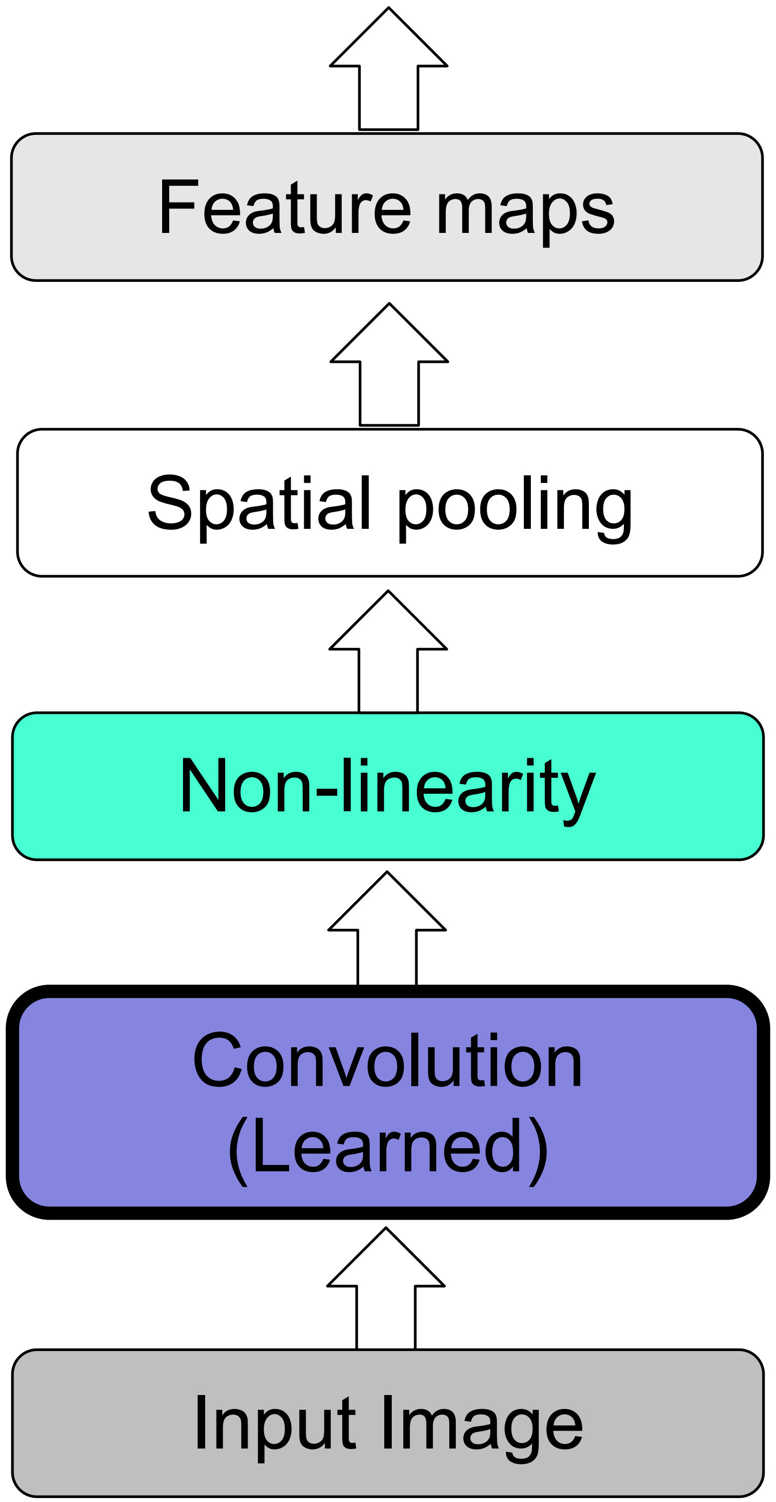
Source: S. Lazebnik

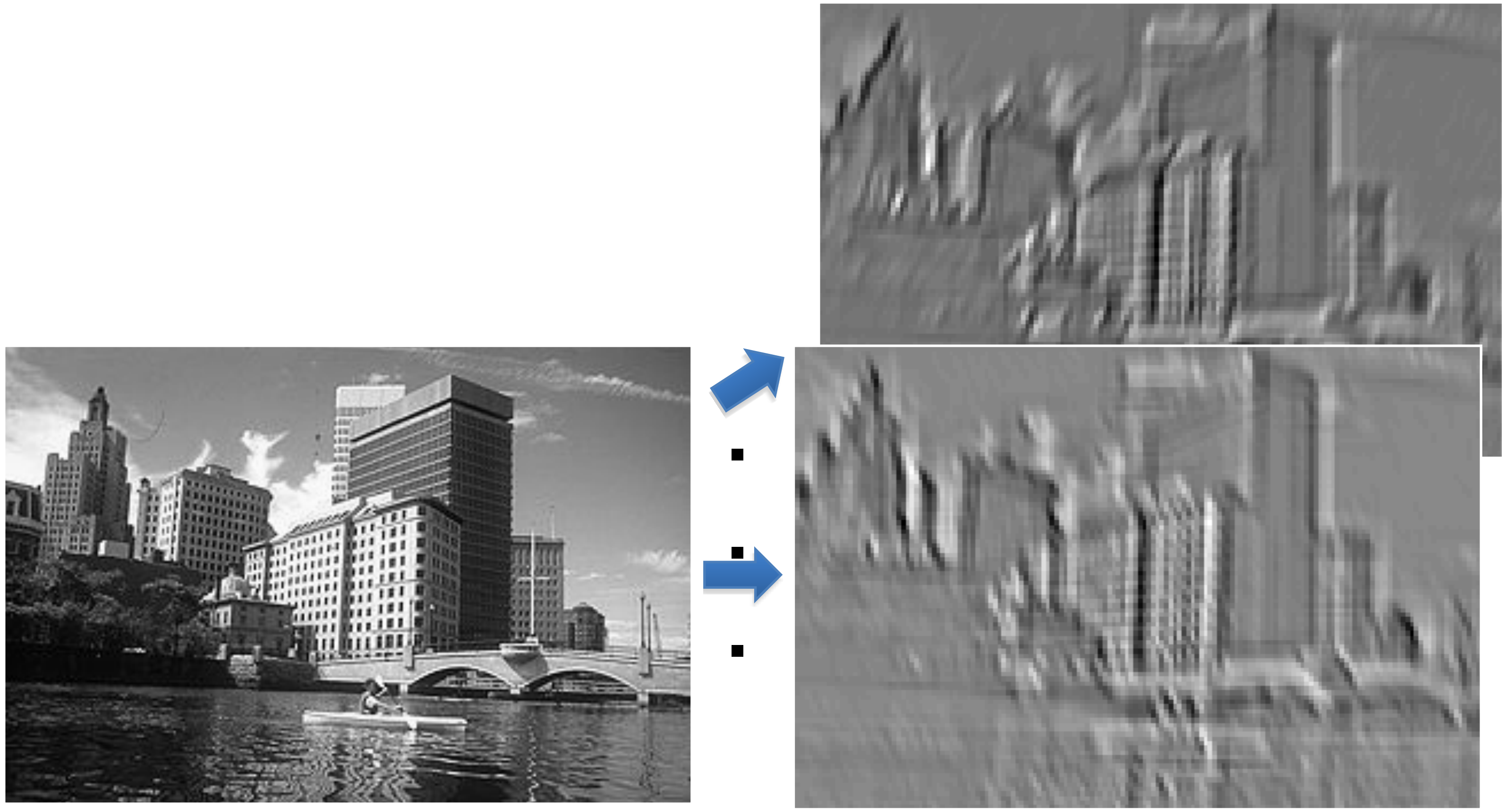
Neural networks for images



Source: S. Lazebnik

Key operations in a CNN



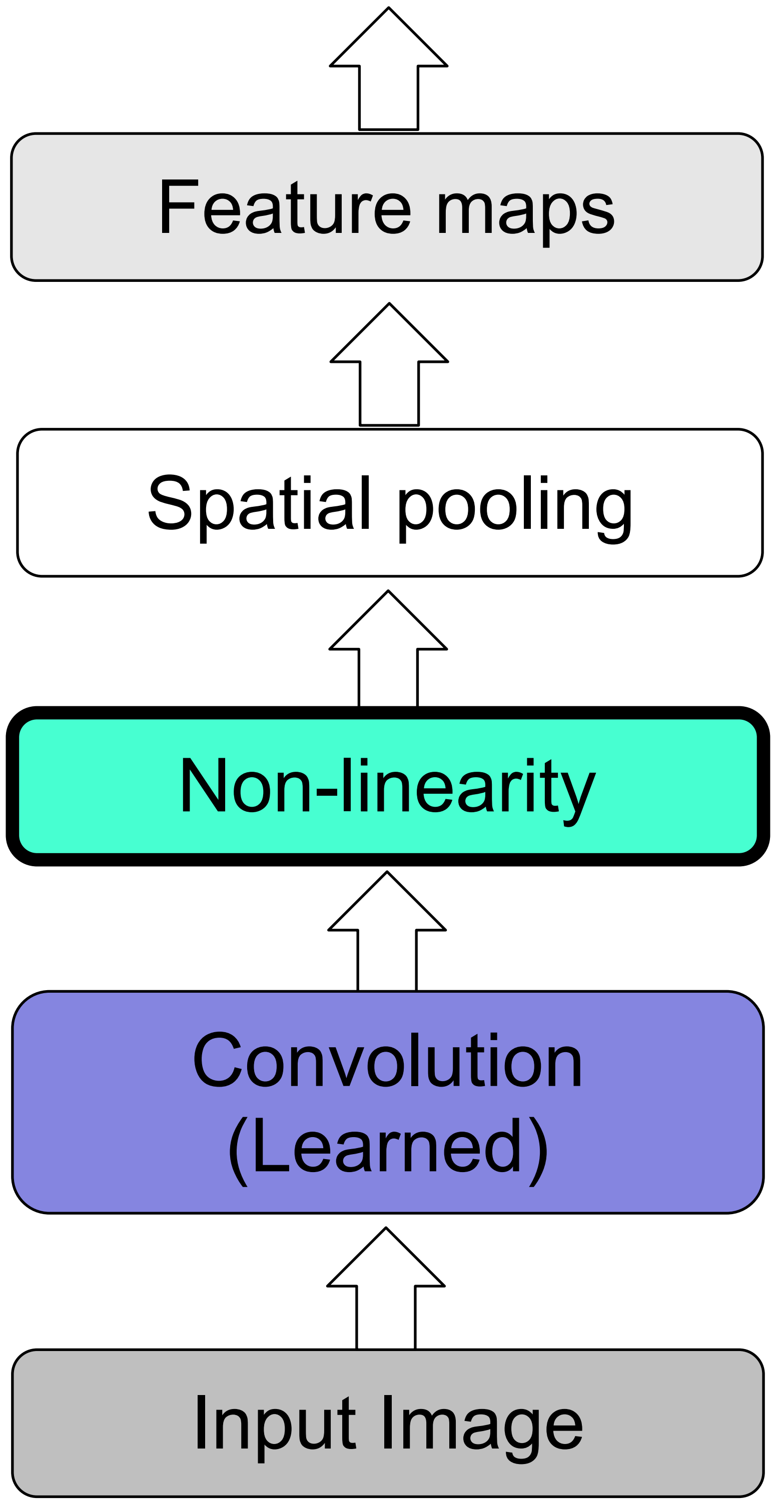


Input

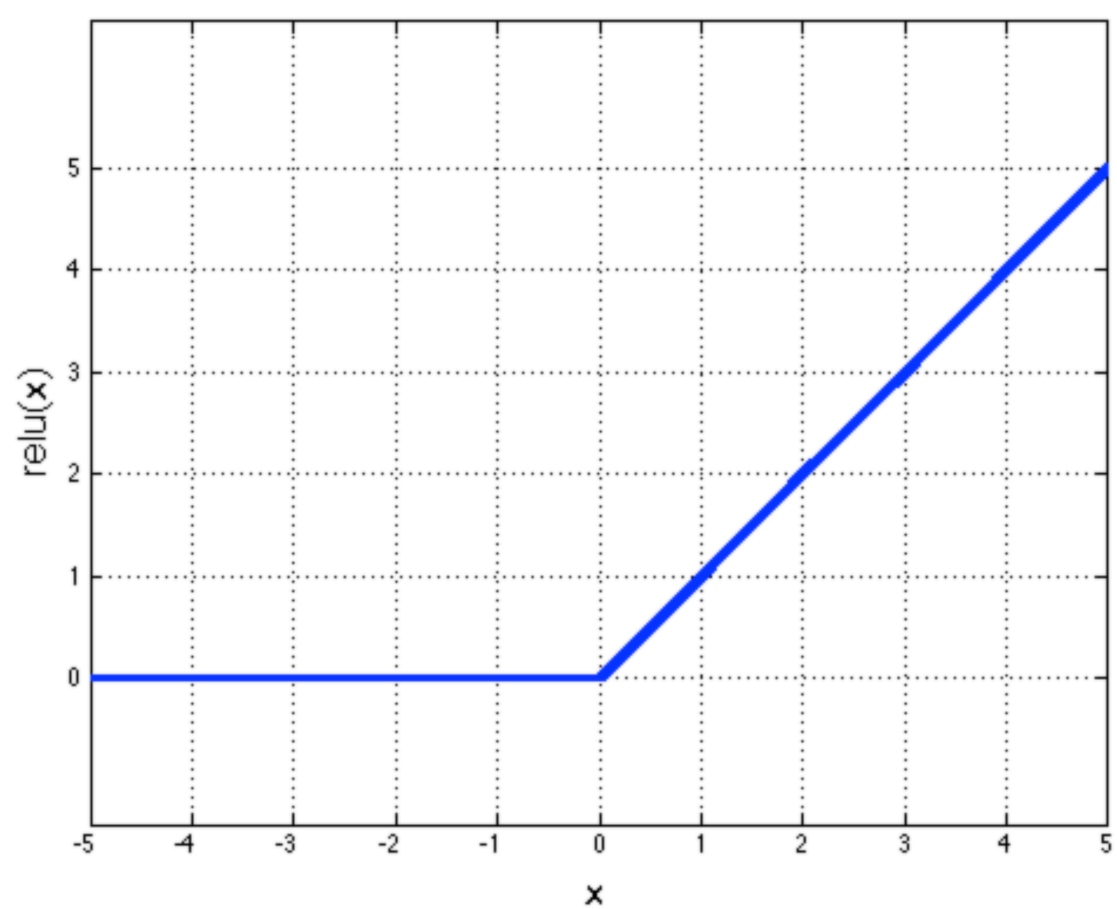
Feature Map

Source: R. Fergus, Y. LeCun

Key operations in a CNN

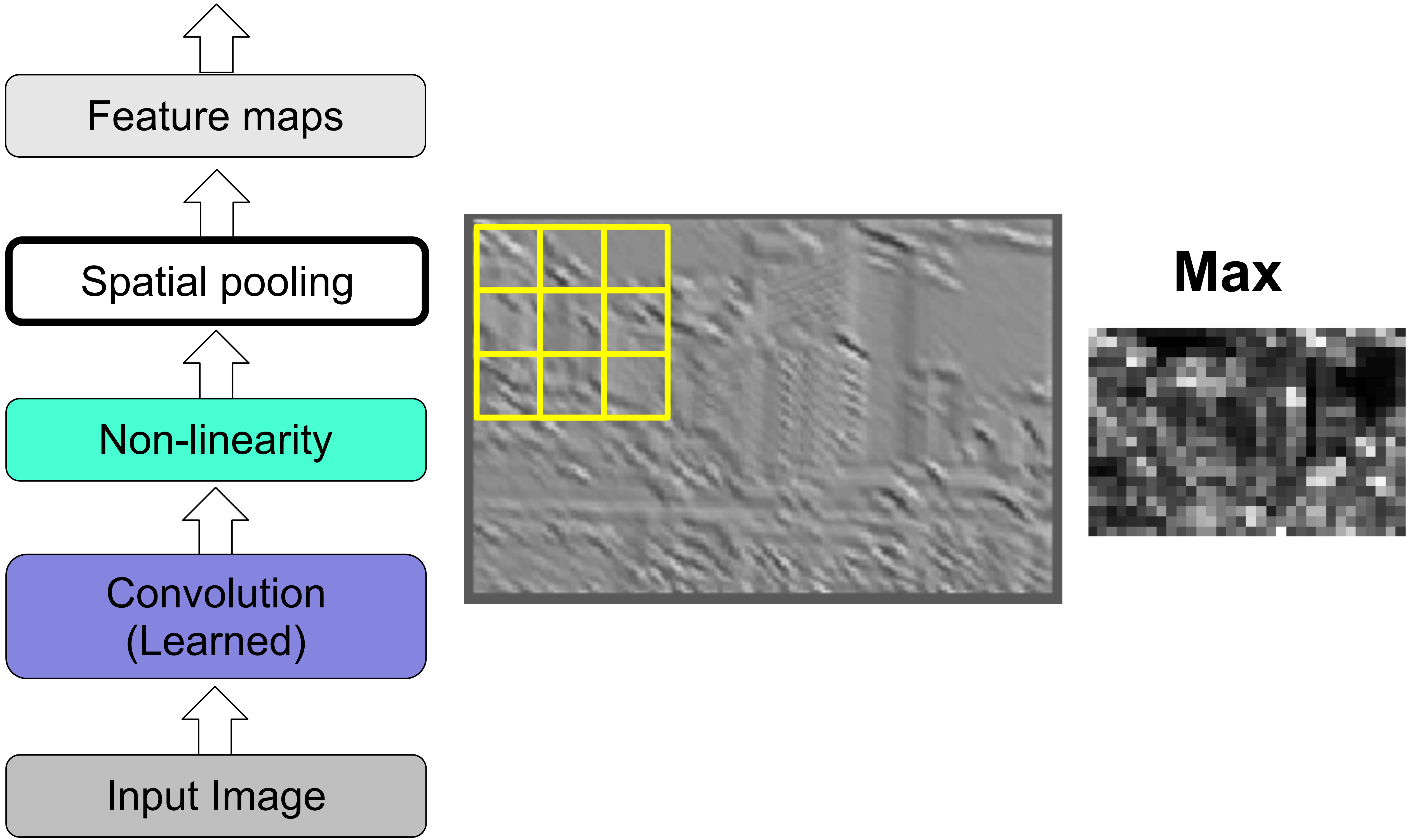


Rectified Linear Unit (ReLU)



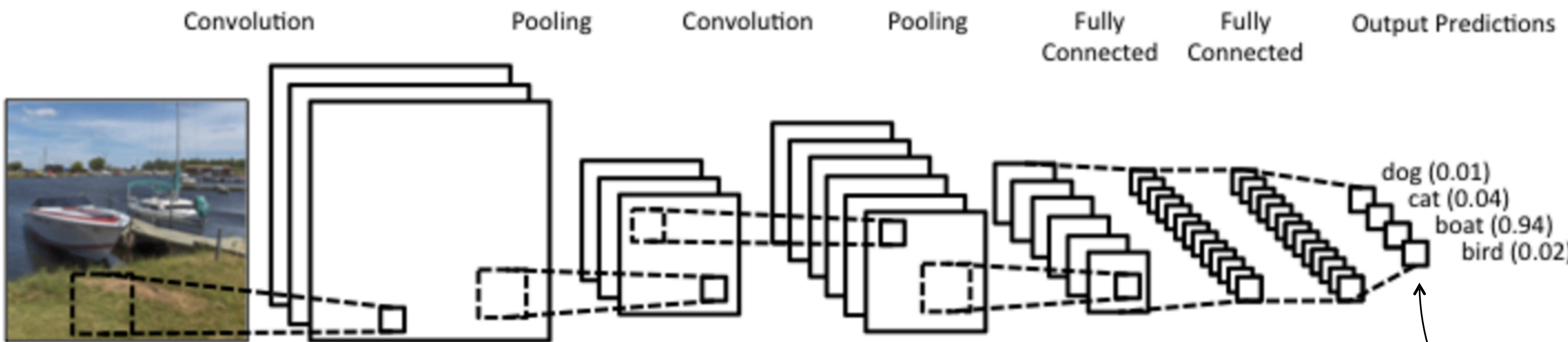
Source: R. Fergus, Y. LeCun

Key operations in a CNN



Source: R. Fergus, Y. LeCun

Key operations in a CNN



Softmax layer:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e_j^z}$$

Source: S. Lazebnik

Loss functions

The objective to minimize is of the form

$$L(\theta) = \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

Examples for ℓ:

- *Regression:*
  - Quadratic loss: $\ell(y, \hat{y}) = \|y - \hat{y}\|_2^2, \quad y, \hat{y} \in \mathbb{R}^d$
- *Classification:*
  - *Cross-entropy:* $\ell(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i), \;\; y \in \{0,1\}^d, \hat{y} \in [0,1]^d.$

Classification is usually better! (easier problem)

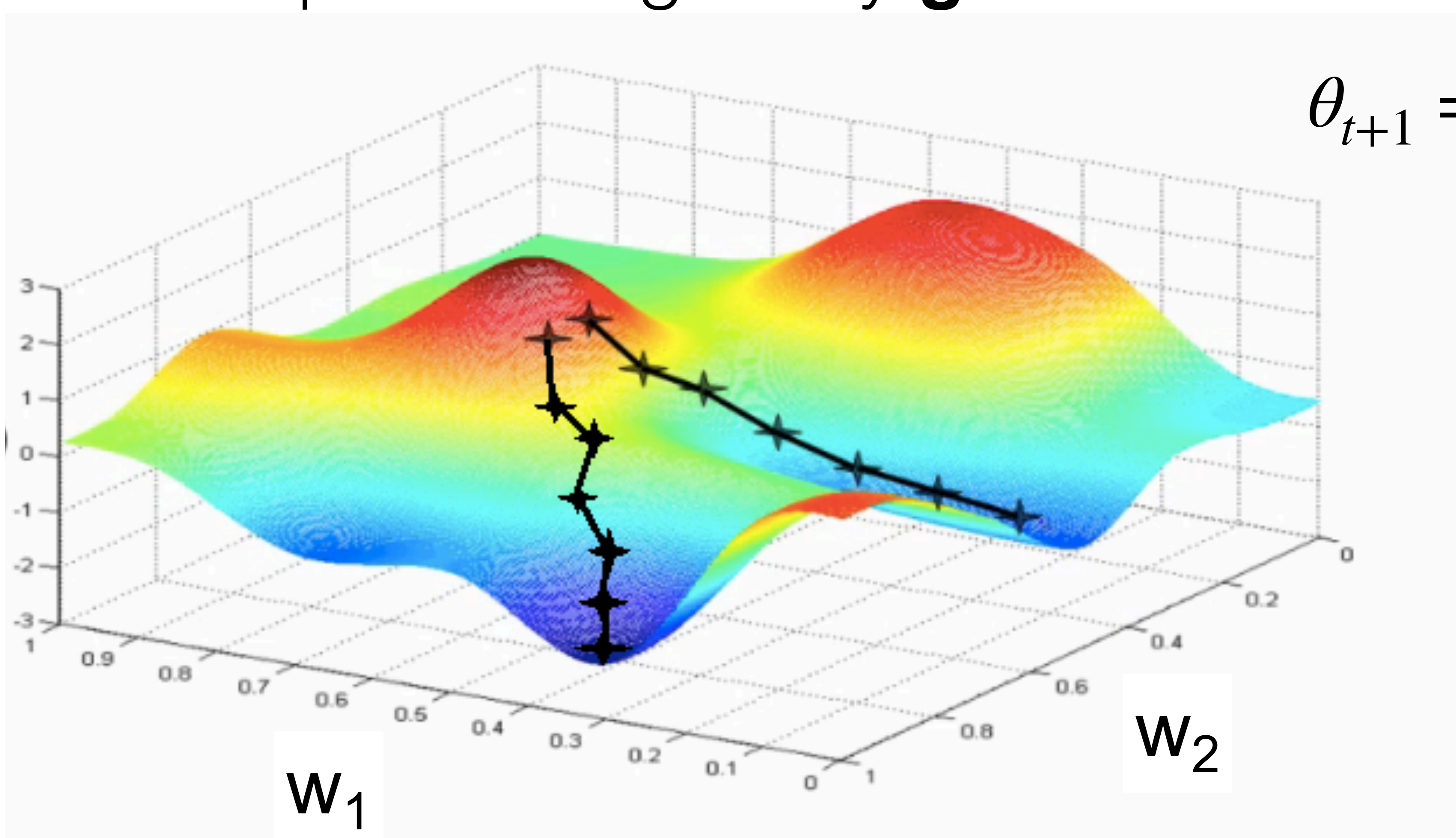
Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

- Update weights by **gradient descent:**

$$\theta_{t+1} = \theta_t - \gamma_t \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial \theta}\ell(f_\theta(x_i), y_i)$$



Gradient descent

$$\theta_{t+1} = \theta_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} \ell(f_\theta(x_i), y_i)$$

- Need to choose the learning rate policy γ_t
- Can get stuck in a local minima ($L(\theta)$ is not convex)
- Each step can be expensive to compute if the dataset is large

Stochastic gradient descent

- Idea: instead of computing the gradient, compute an approximation

$$\theta_{t+1} = \theta_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} \ell(f_\theta(x_i), y_i)$$

$$\downarrow$$

$$\theta_{t+1} = \theta_t - \gamma_t \frac{\partial}{\partial \theta} \ell(f_\theta(x_{i_t}), y_{i_t})$$

Note that in expectation

$$\mathbb{E}(\theta_{t+1} | \theta_t) = \theta_{t+1} = \theta_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} \ell(f_\theta(x_i), y_i)$$

Batch SGD

- It's faster to compute several gradients in parallel
- Some variance is good, too much can be bad

→ compute gradient estimates on a batch instead of a single sample.

$$\theta_{t+1} = \theta_t - \gamma_t \frac{1}{K} \sum_{k=1}^{K} \frac{\partial}{\partial \theta} \ell(f_\theta(x_{i_k,t}), y_{i_k,t})$$

- In practice, using batches as large as possible so that the network fits in the GPU memory

Beyond SGD

- Many other algorithms: see https://ruder.io/optimizing-gradient-descent/

- GD with momentum: encourage directions that are coherent:

$$v_t = \eta v_{t-1} + \gamma \nabla_{\theta_t}$$
$$\theta_{t+1} = \theta_t - v_t$$

- Adagrad, Adadelta, RMSProp, ADAM…

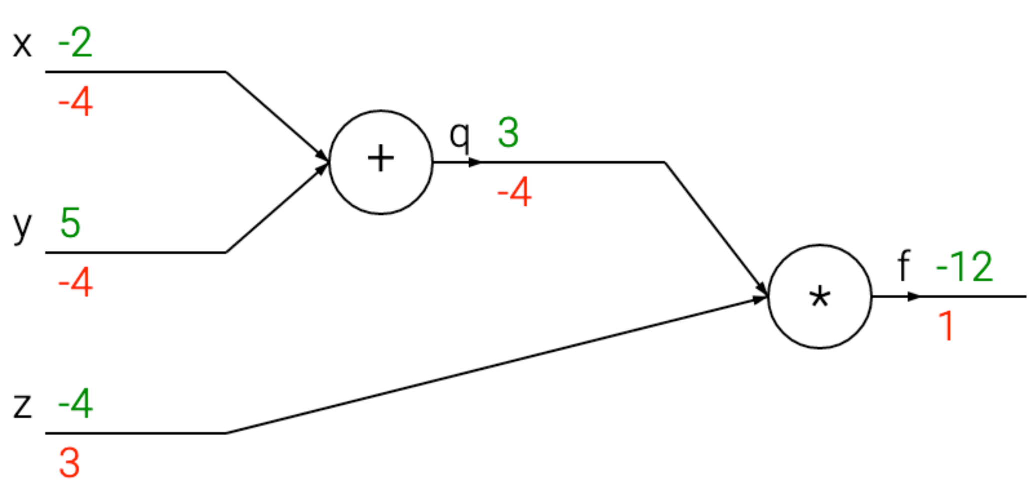
- All these optimizers are coded in standard deep learning libraries

Backpropagation

- AKA "reverse-mode automatic differentiation" ("chain rule")
  - *not* finite differences
  - *not* symbolic differentiation

Simple example: if $y = f_h(f_{h-1}(\ldots f_1(w_0)))$, $w_i = f_i(w_{i-1})$, then compute

$$\frac{dy}{dw_i} = \frac{dy}{dw_{i+1}} \frac{dw_{i+1}}{dw_i}$$



Other example: $f(x, y, z) = (x + y)z$, with $x = -2, y = 5, z = -4$

Source: http://cs231n.github.io/optimization-2/

Initialization

- Since loss is not convex, initialization is really important!
- If two neurons on the same layer are initialized with the same weight, they will stay the same.
- For ReLU activations, PyTorch uses “Kaiming Uniform”:

$$w_i \sim \mathcal{U}\left[-\sqrt{k}, \sqrt{k}\right] \quad \text{where} \quad k = \frac{1}{n_{in}}$$

Regularization

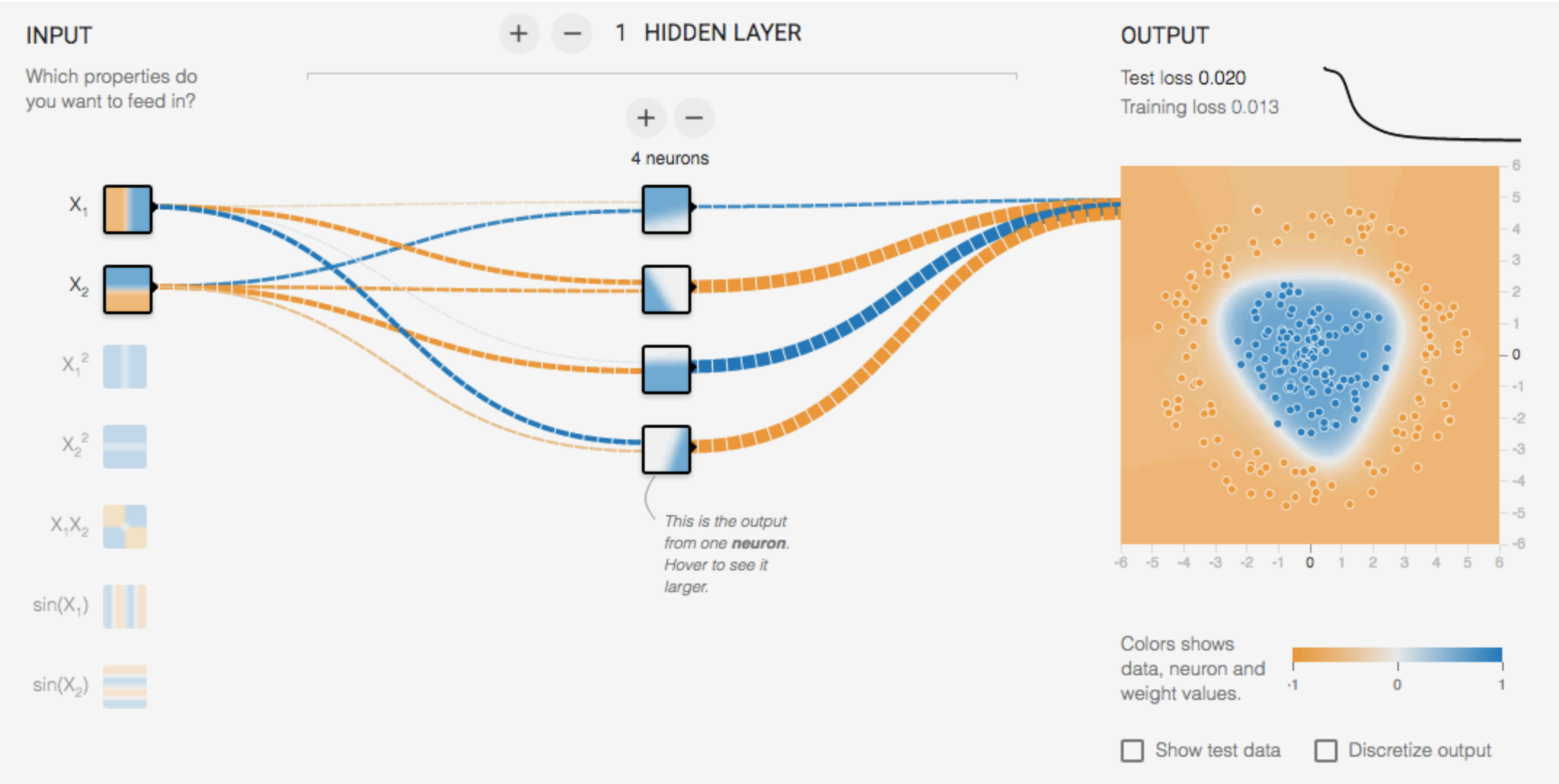
Several techniques for regularization (to avoid overfitting):

- L2 penalization on the weights (also called weight decay)

$$L(\theta) = \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + \lambda ||\theta||^2$$

- Dropout: only keep a neuron active with some probability p, or set it to zero otherwise.
- Early stopping
- Data augmentation

Interactive Demo



http://playground.tensorflow.org/

Common problem: vanishing/exploding gradients

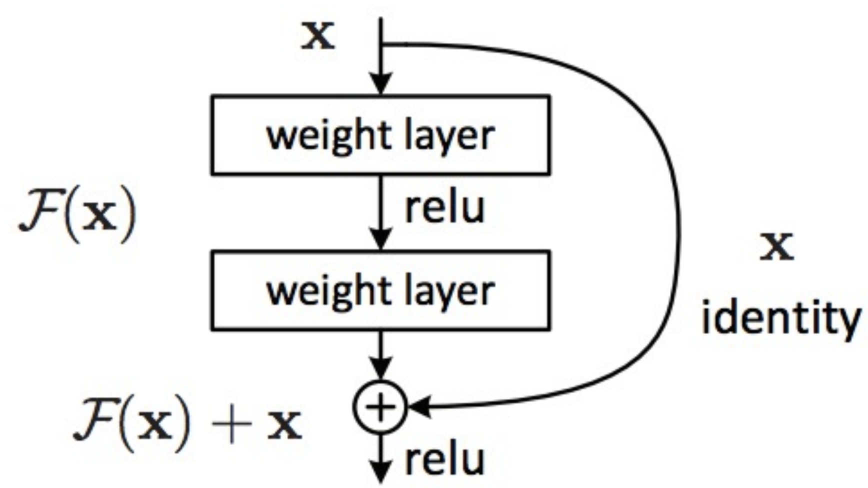
- If all linear layers have width 1, and are initialized at α:

$$\frac{\partial L}{\partial w_1} = \alpha^{N-1} \frac{\partial L}{\partial w_N}$$

- Depending on the value of α, the gradient with respect to w_1 will be huge or very small.
- Similar effects happen for more complex deep networks (even worst if non-linearities have ~0 gradients).

Vanishing/exploding gradients: solutions

- Use ReLU (non-saturating)
- Use skip-connections



- Use batch-normalization

Batch normalization

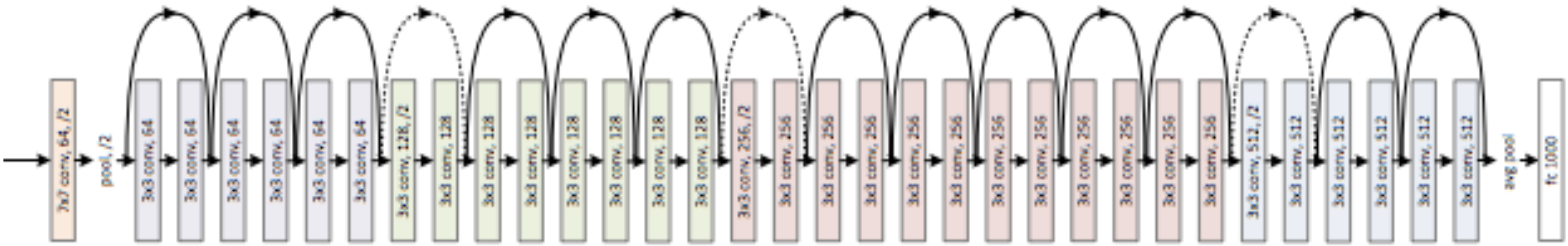
- Additional layer to avoid vanishing or exploding signal.
- Idea: normalize the data everywhere in the network using estimates of the mean/variance

$$BN_{\alpha,\beta}(x, \mu, \sigma) = \alpha \frac{x - \mu}{\sigma} + \beta$$

- $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are learned, $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are estimated over a mini-batch.
- Batch-norm layers are typically placed just before non-linearities

Typical modern network: Resnet

- Identical layers are repeated many times



- 3x3 convolution kernels + BN + ReLUs.
- Skip connections.
- Only one fully connected layer at the end.

K. He, X. Zhang, S. Ren, and J. Sun, [Deep Residual Learning for Image Recognition](), CVPR 2016 (Best Paper)

AlexNet and VGGNet

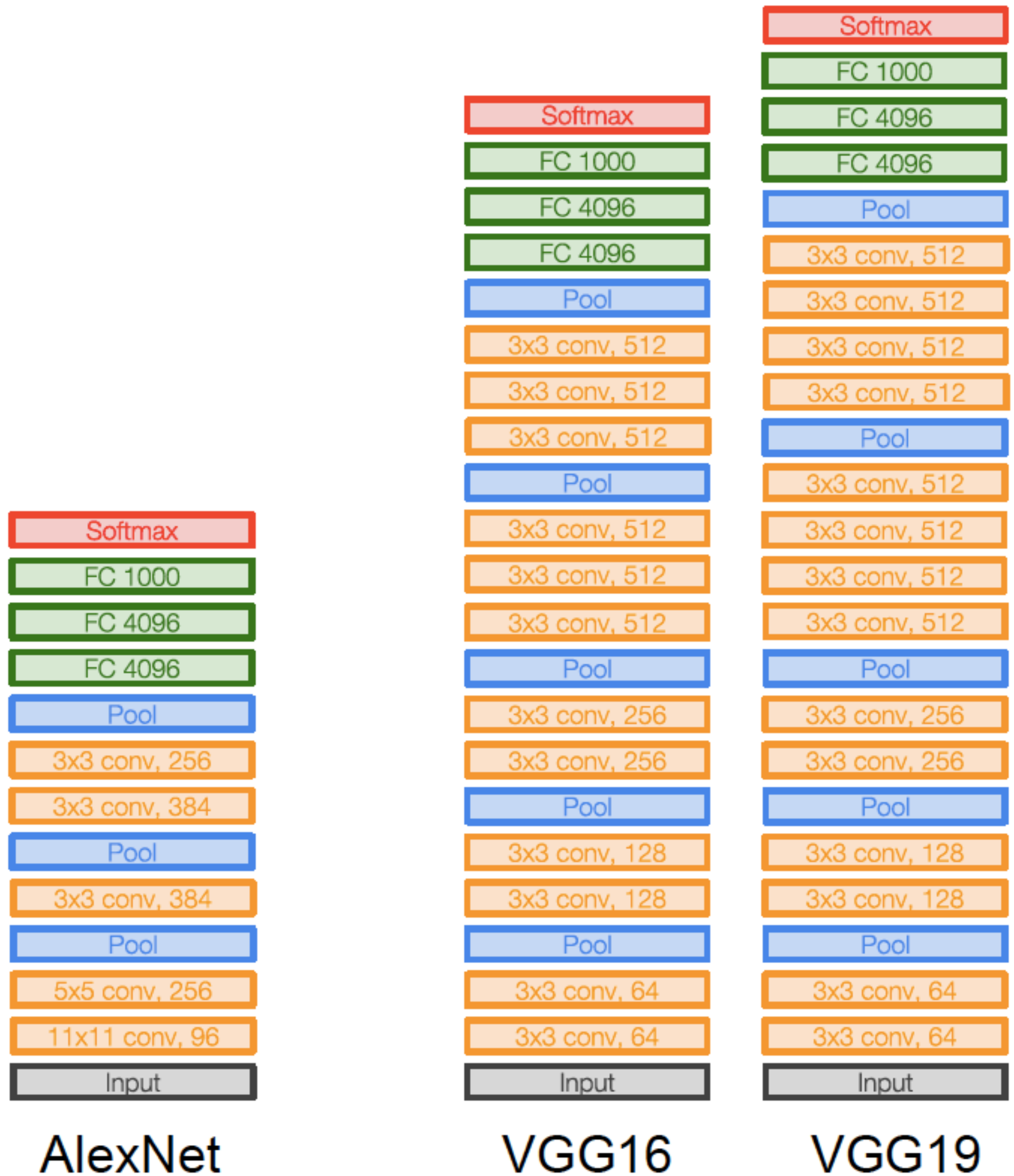
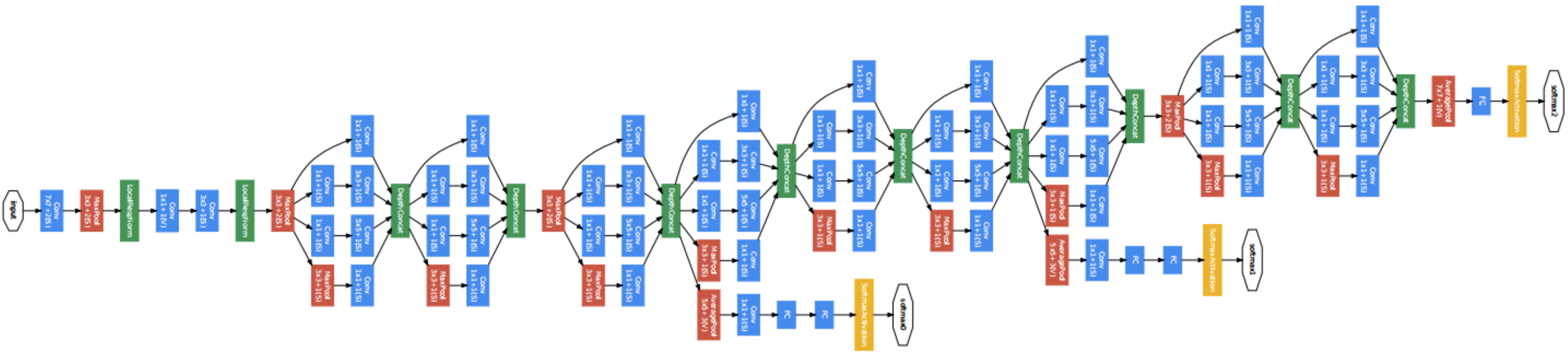


Image source

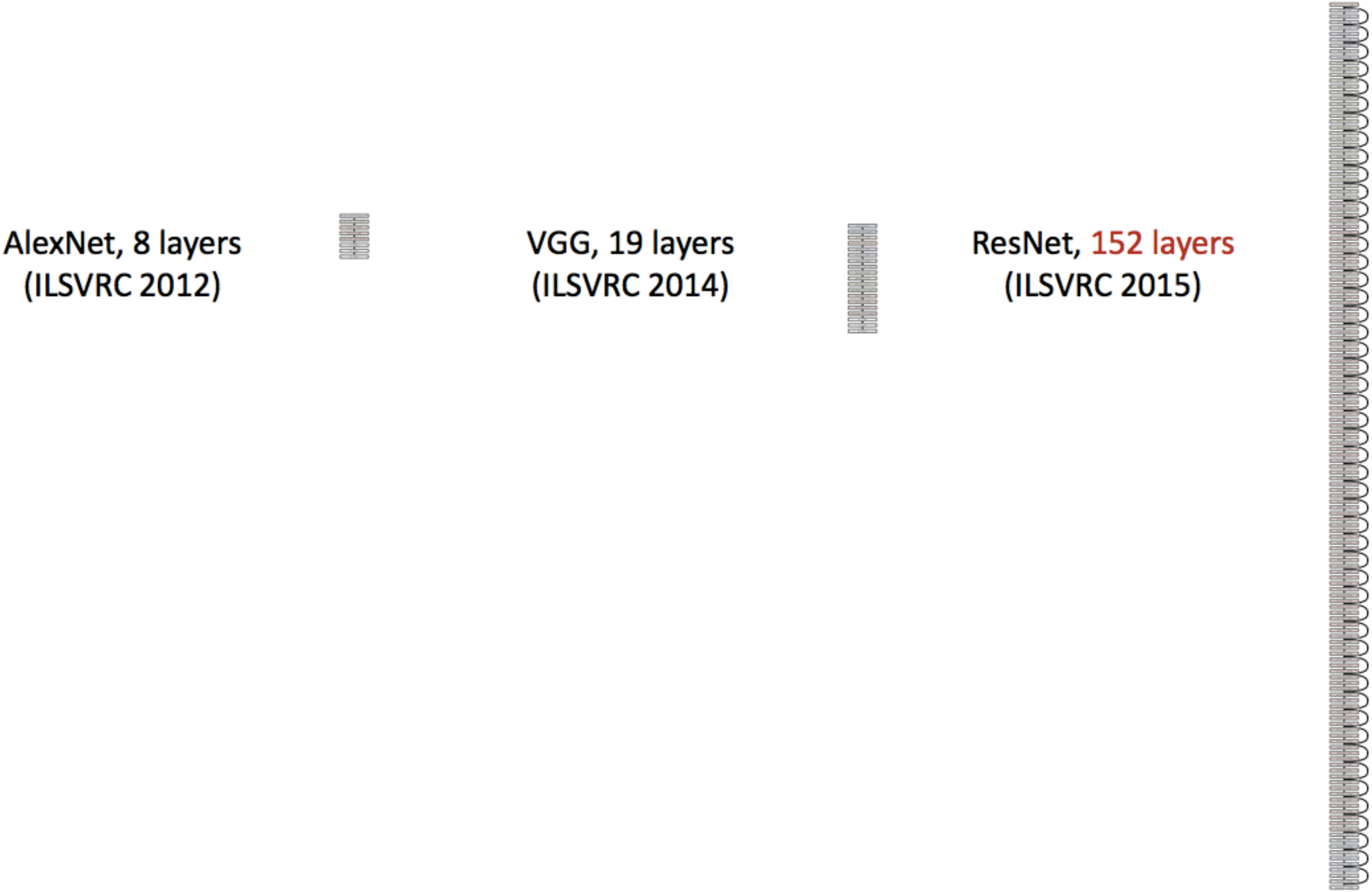
K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

GoogLeNet



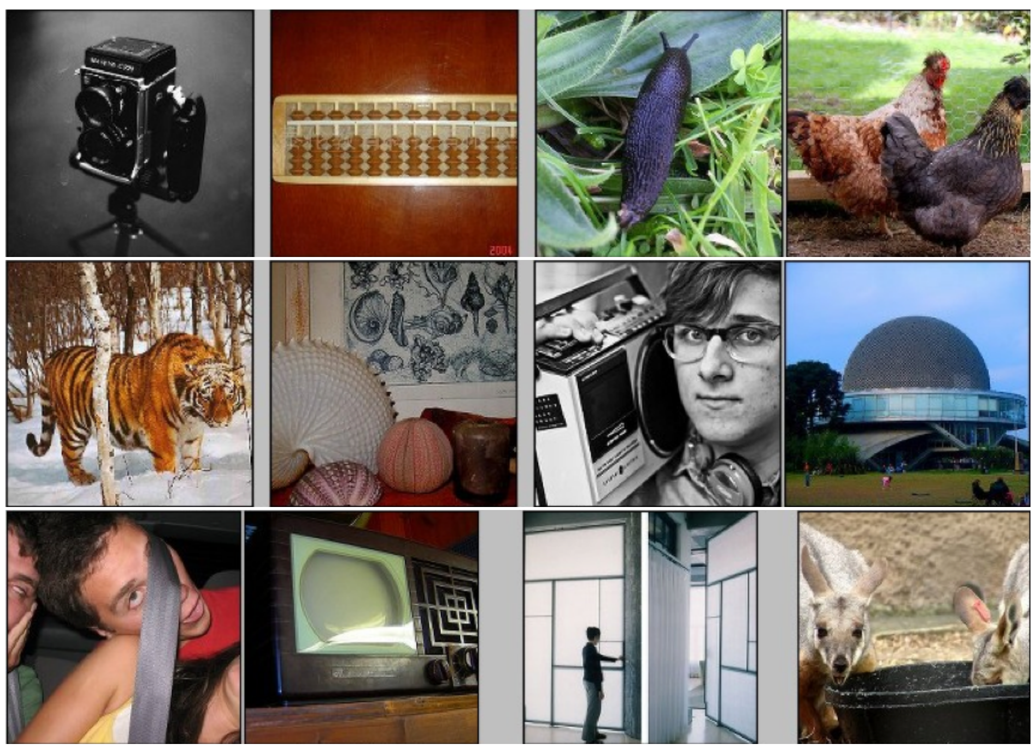
C. Szegedy et al., [Going deeper with convolutions](), CVPR 2015

ResNet



K. He, X. Zhang, S. Ren, and J. Sun, [Deep Residual Learning for Image Recognition](), CVPR 2016 (Best Paper)

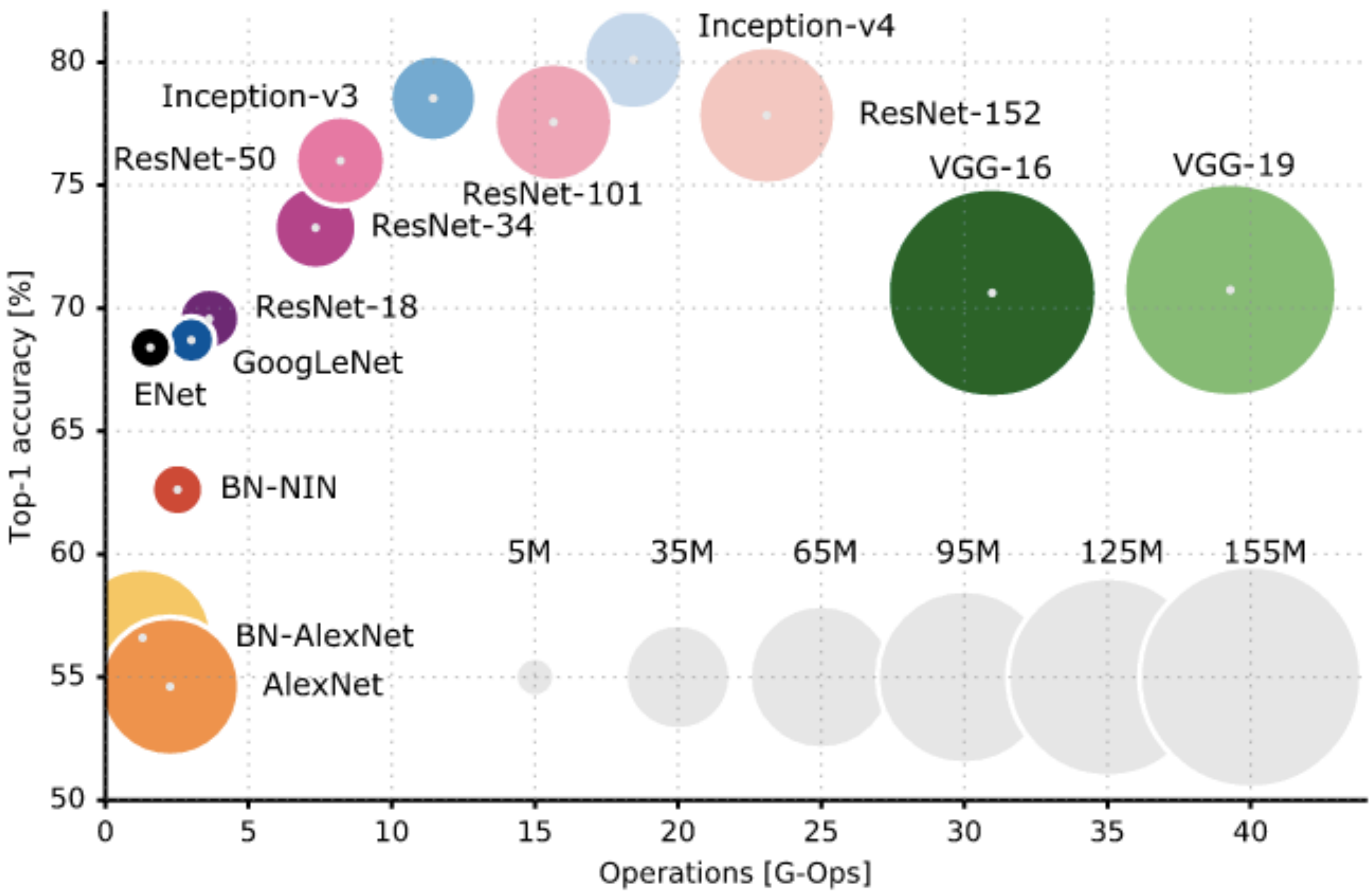
ImageNet Challenge



- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):
  1.2 million training images, 1000 classes
- Starting with 2015, image classification is not part of ILSVRC challenge (but people continue to benchmark on the data)

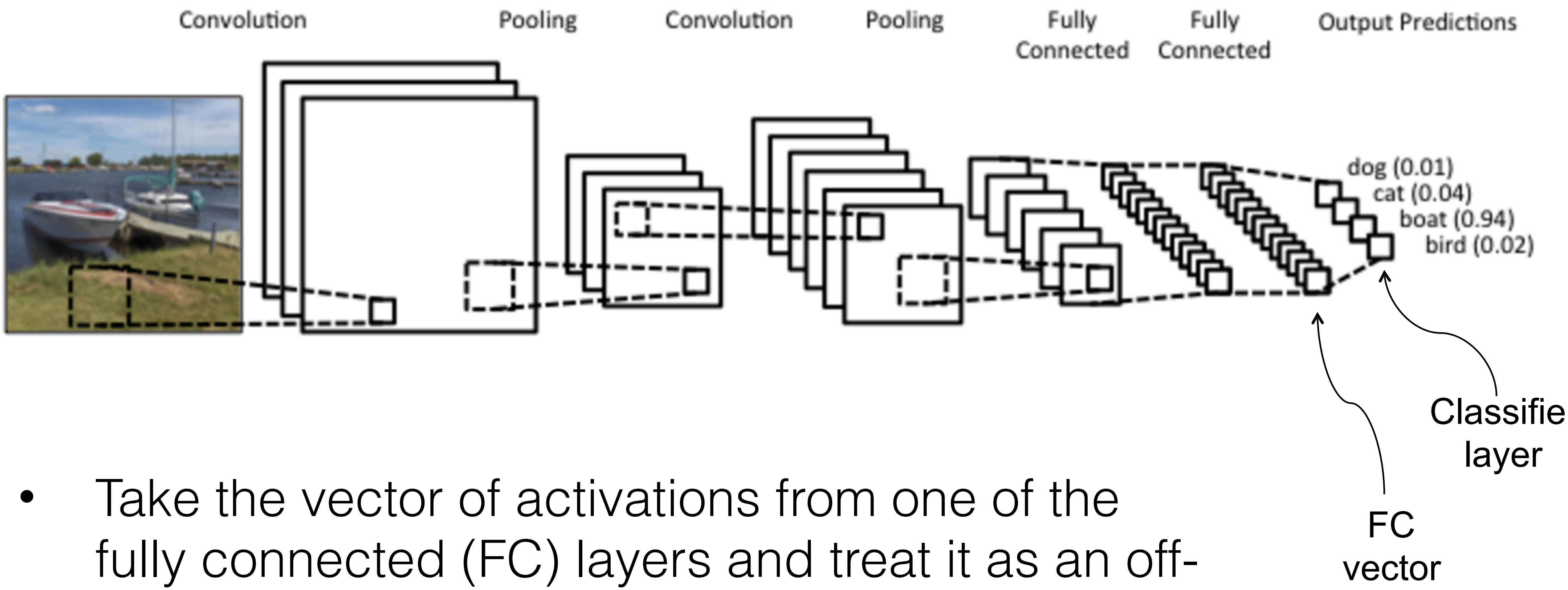
www.image-net.org/challenges/LSVRC/

Comparing architectures



https://culurciello.github.io/tech/2016/06/04/nets.html

How to use a trained network for a new task?



- Take the vector of activations from one of the fully connected (FC) layers and treat it as an off-the-shelf feature
- Train a new classifier layer on top of the FC layer
- Fine-tune the whole network.

More general pre-training than ImageNet?

- Goal: learn generic features, that can be useful for other tasks.

- Idea of “self supervised” learning: find auxiliary task which allows to learn useful features

Puzzle solving

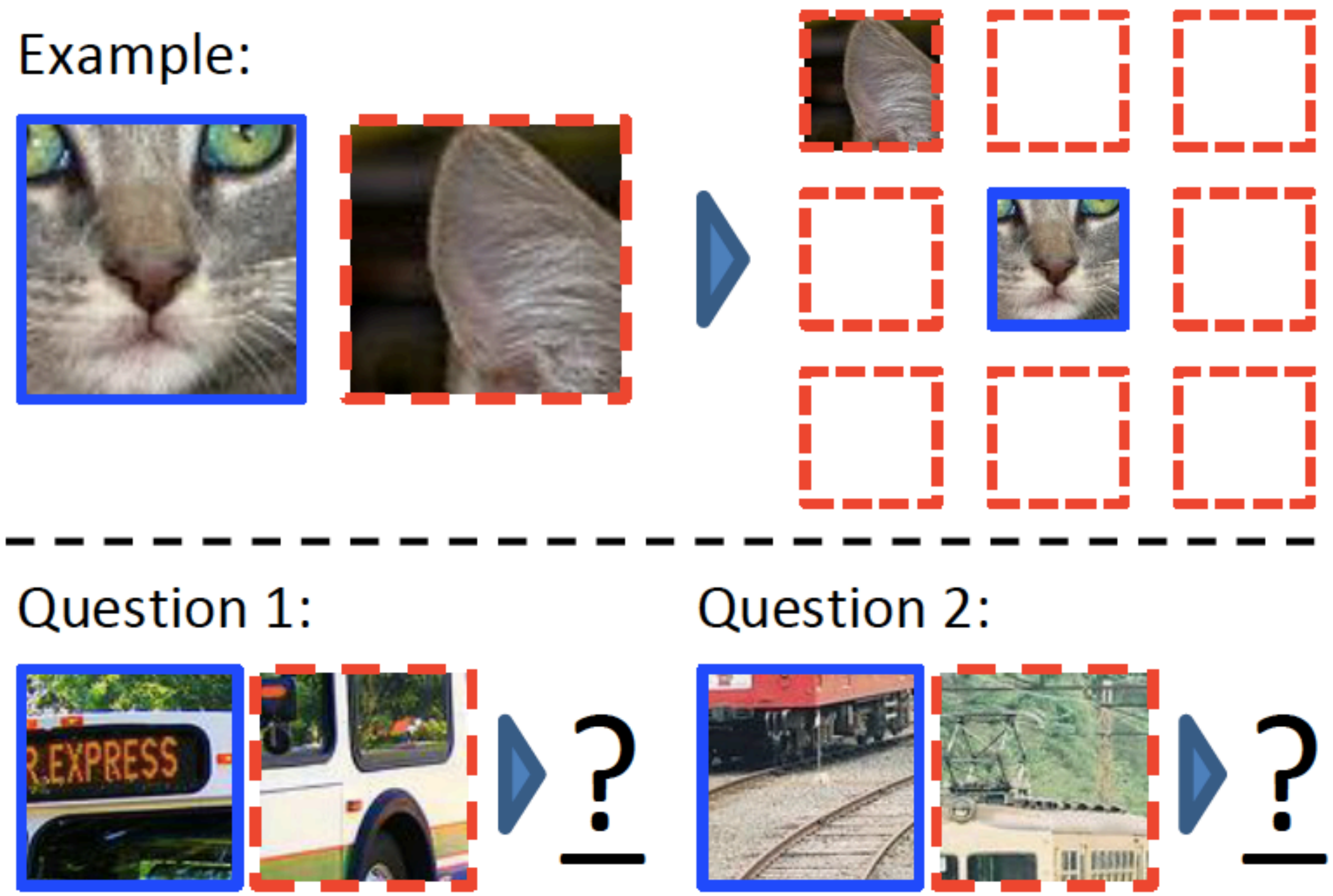
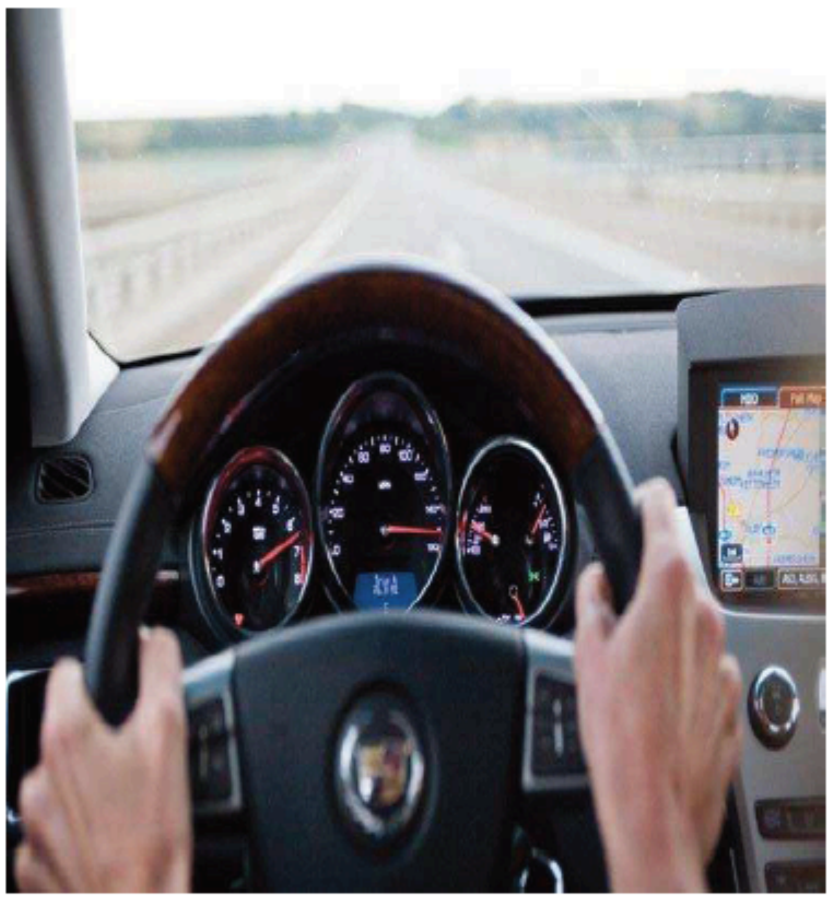
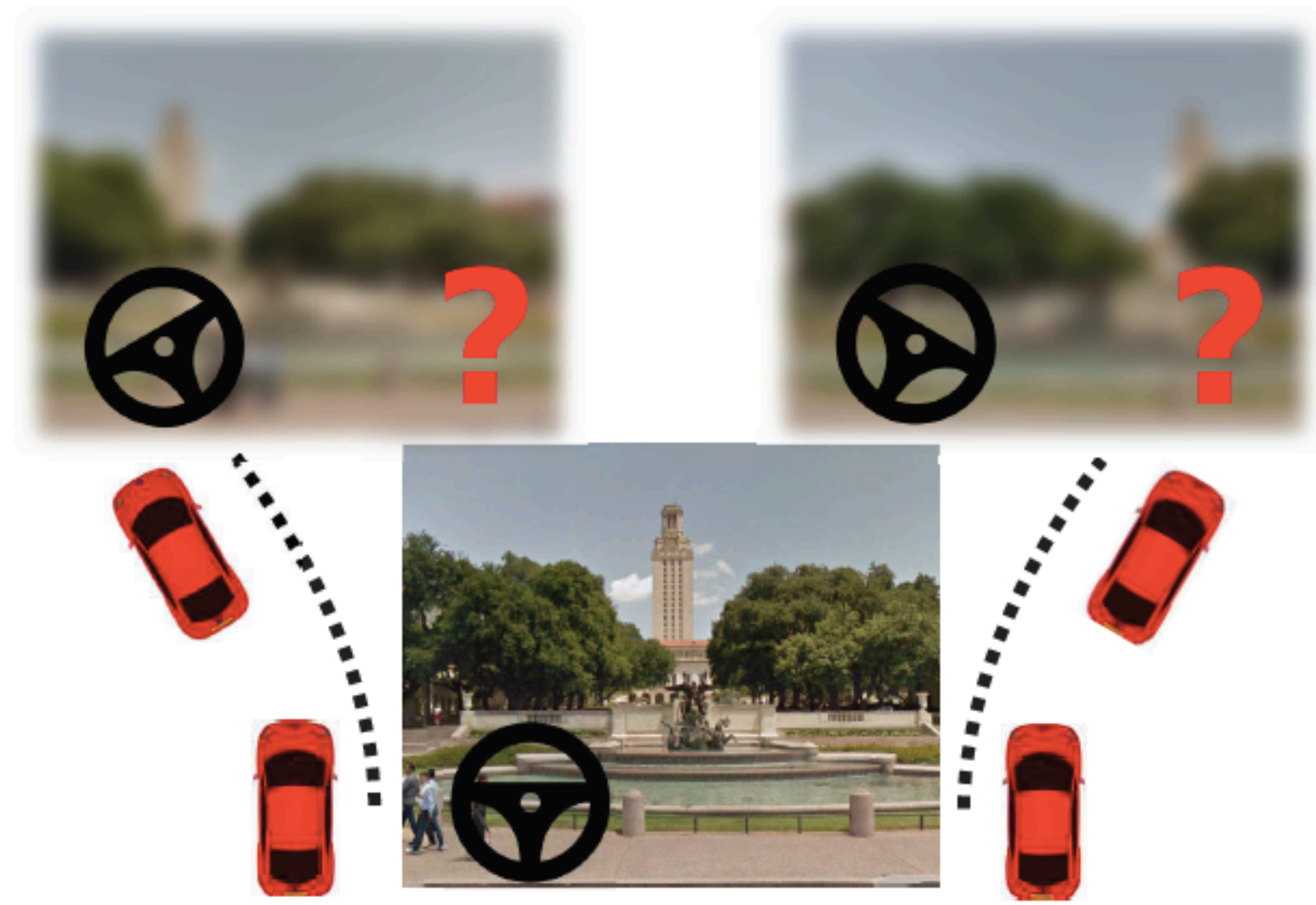


Image completion

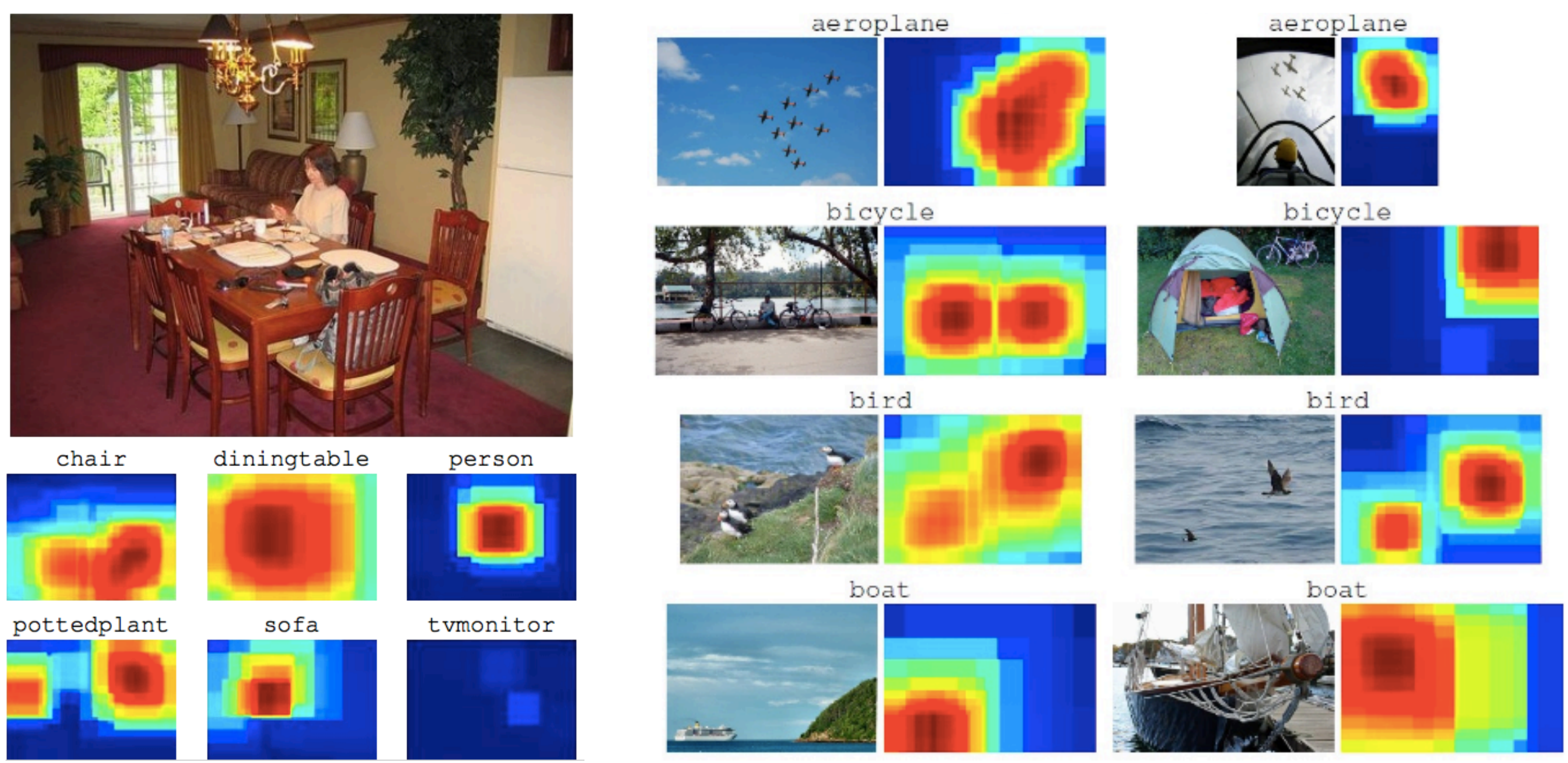
Predicting the future

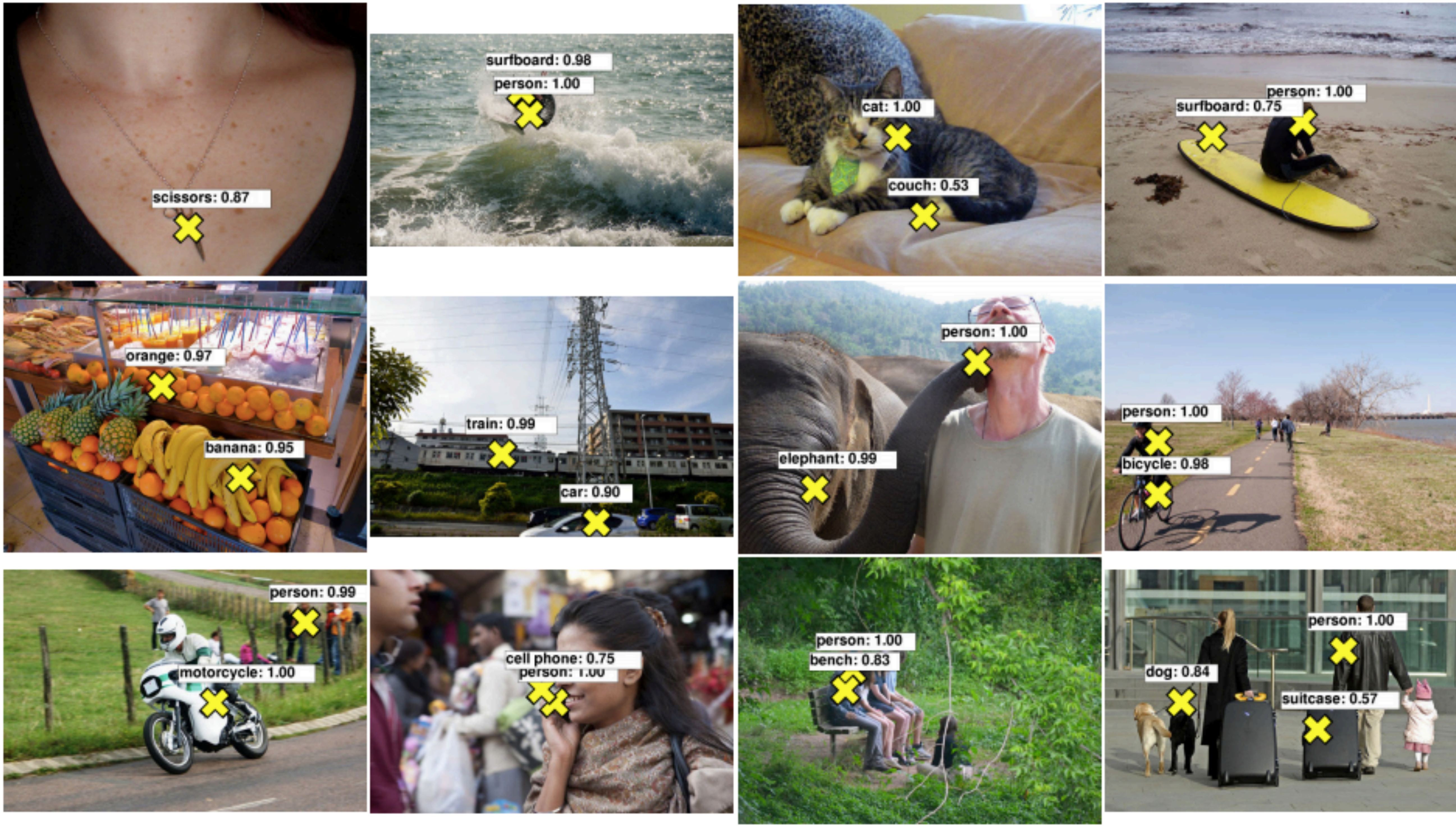




Free localization from classification

- Apply densely a classification network and look at the results:

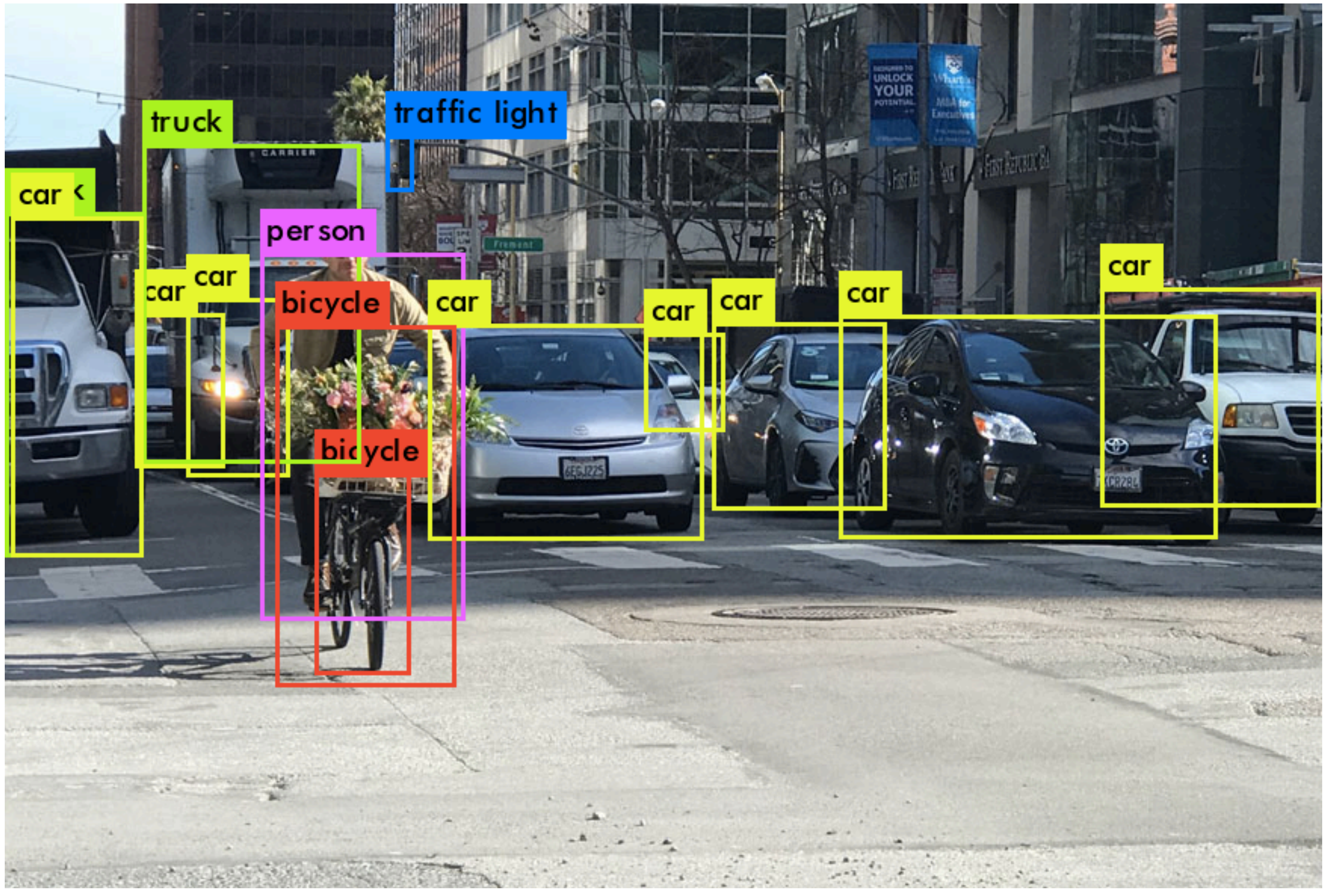



scissors: 0.87
surfboard: 0.98
person: 1.00
cat: 1.00
couch: 0.53
person: 1.00
surfboard: 0.75
orange: 0.97
banana: 0.95
train: 0.99
car: 0.90
person: 1.00
elephant: 0.99
person: 1.00
bicycle: 0.98
person: 0.99
motorcycle: 1.00
cell phone: 0.75
person: 1.00
person: 1.00
bench: 0.83
person: 1.00
dog: 0.84
suitcase: 0.57

Neural networks for object detection

What are the challenges of object detection?

- Images may contain more than one class, multiple instances from the same class
- Bounding box localization
- Evaluation

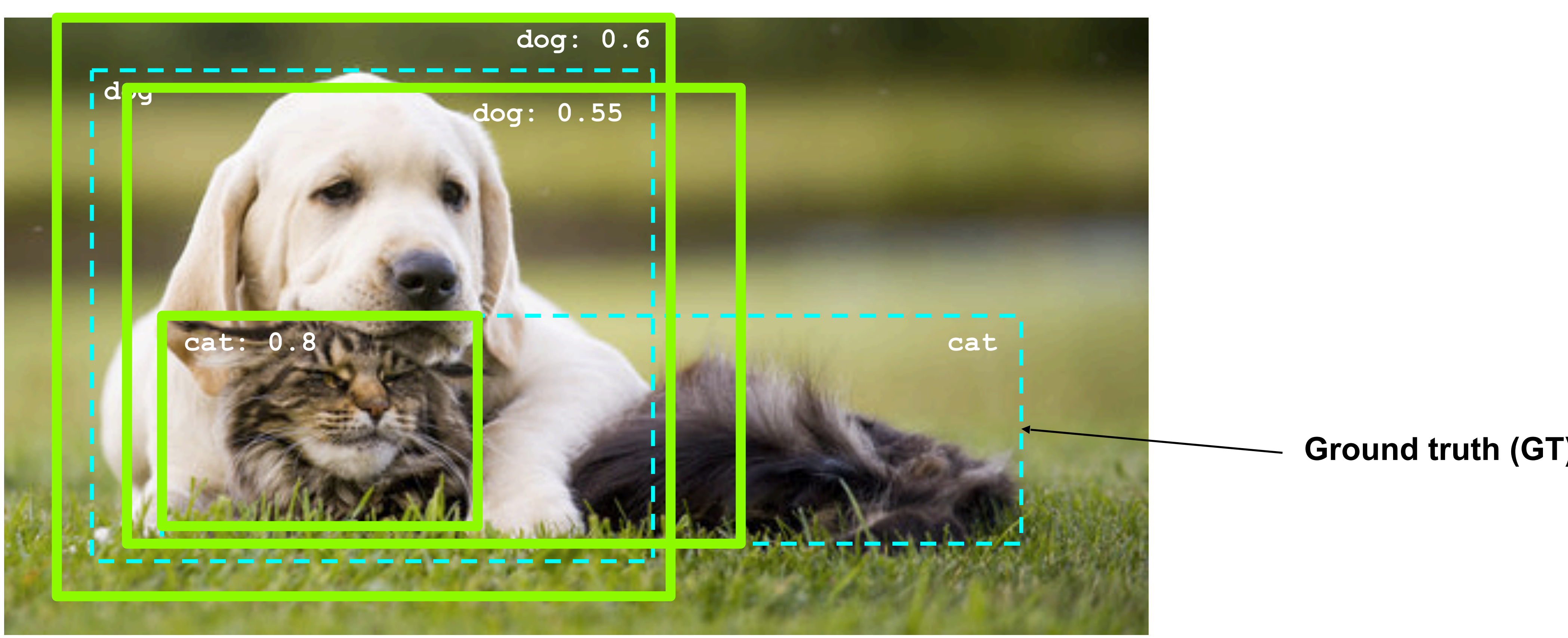


Source: S. Lazebnik

Image source

Object detection evaluation

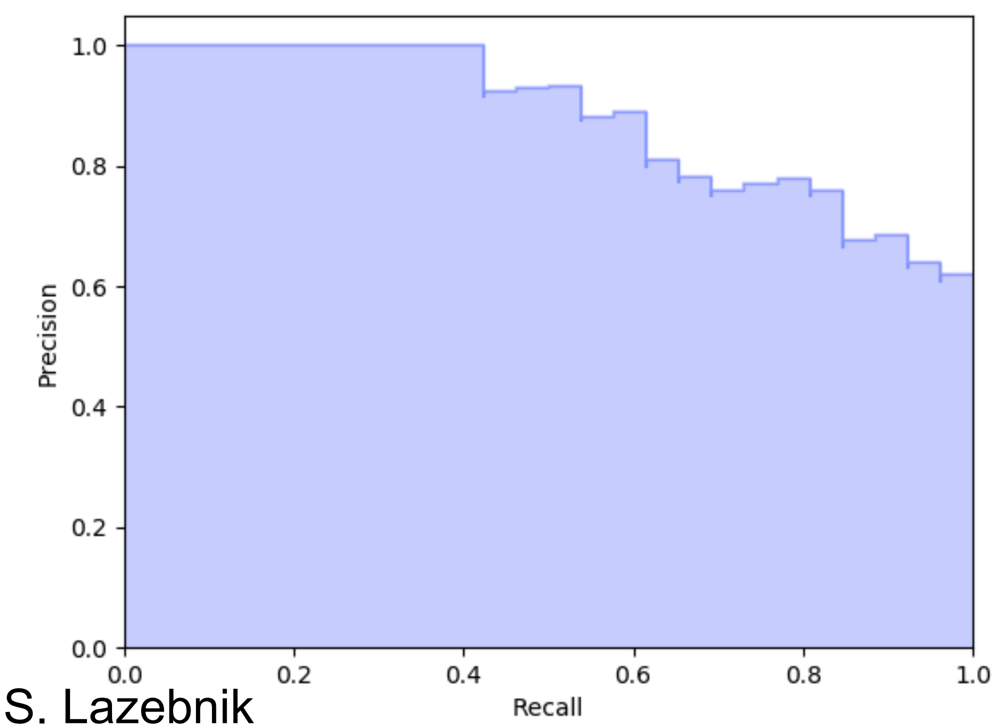
- At test time, predict bounding boxes, class labels, and confidence scores
- For each detection, determine whether it is a true or false positive
  - PASCAL criterion: Area(GT ∩ Det) / Area(GT ∪ Det) > 0.5
  - For multiple detections of the same ground truth box, only one considered a true positive



Source: S. Lazebnik

Object detection evaluation

- At test time, predict bounding boxes, class labels, and confidence scores
- For each detection, determine whether it is a true or false positive
- For each class, plot Recall-Precision curve and compute Average Precision (area under the curve)
- Take mean of AP over classes to get mAP



Precision:
true positive detections / total detections
Recall:
true positive detections / total positive test instances

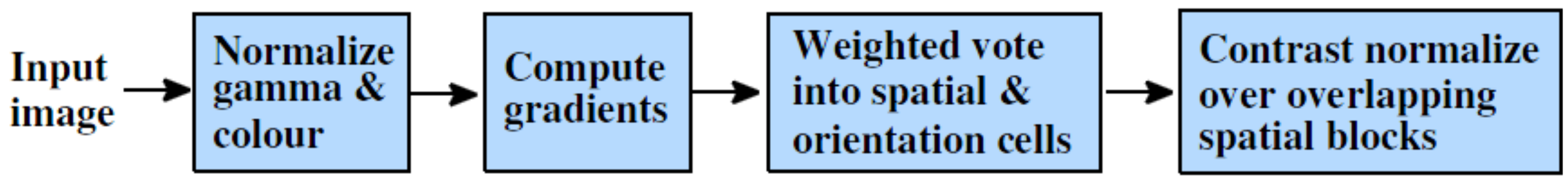
Source: S. Lazebnik

Simple approach: Sliding window detection

- Slide a window across the image and evaluate a detection model at each location
  - Thousands of windows to evaluate: efficiency and low false positive rates are essential
  - Difficult to extend to a large range of scales, aspect ratios

Histograms of oriented gradients (HOG)

- Partition image into blocks and compute histogram of gradient orientations in each block



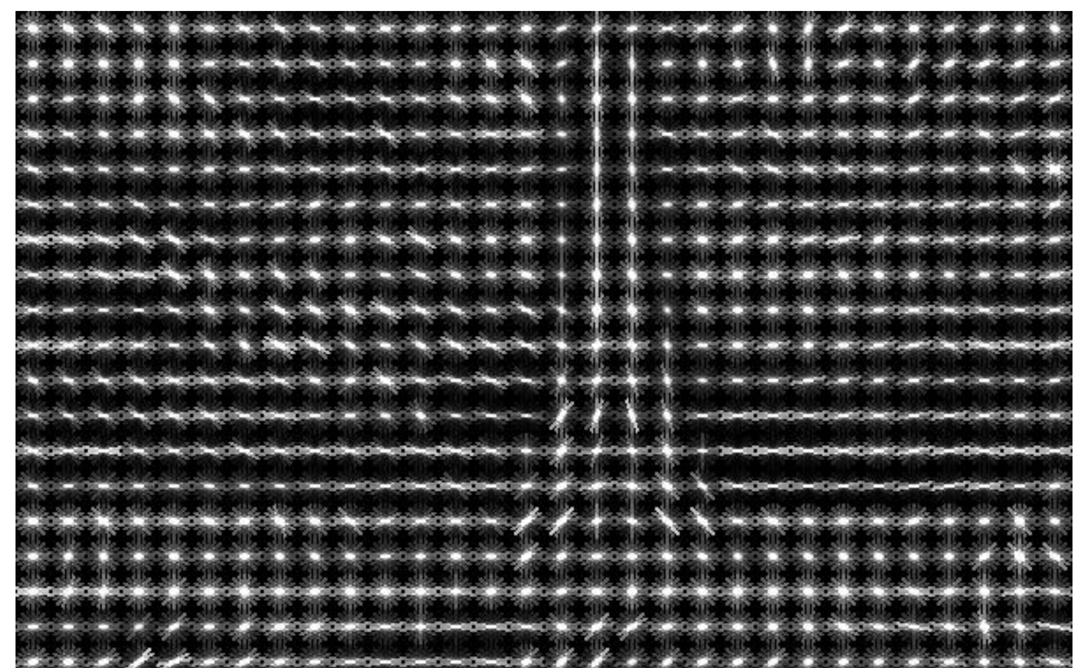


Image credit: N. Snavely

N. Dalal and B. Triggs, [Histograms of Oriented Gradients for Human Detection](), CVPR 2005

Source: S. Lazebnik

Pedestrian detection with HOG

- Train a pedestrian template using a linear support vector machine

positive training examples

negative training examples

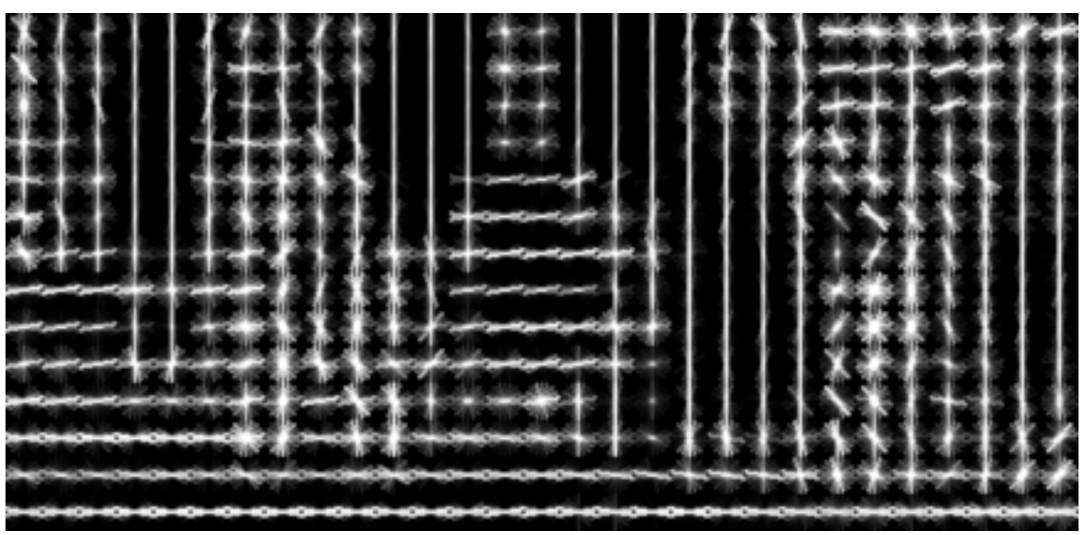
N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Source: S. Lazebnik

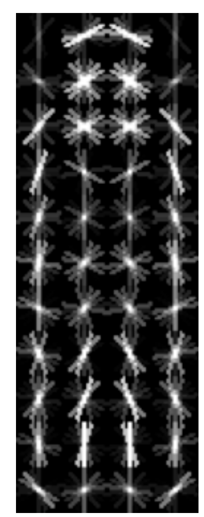
Pedestrian detection with HOG

- Train a pedestrian template using a linear support vector machine
- At test time, convolve feature map with template
- Find local maxima of response
- For multi-scale detection, repeat over multiple levels of a HOG *pyramid*

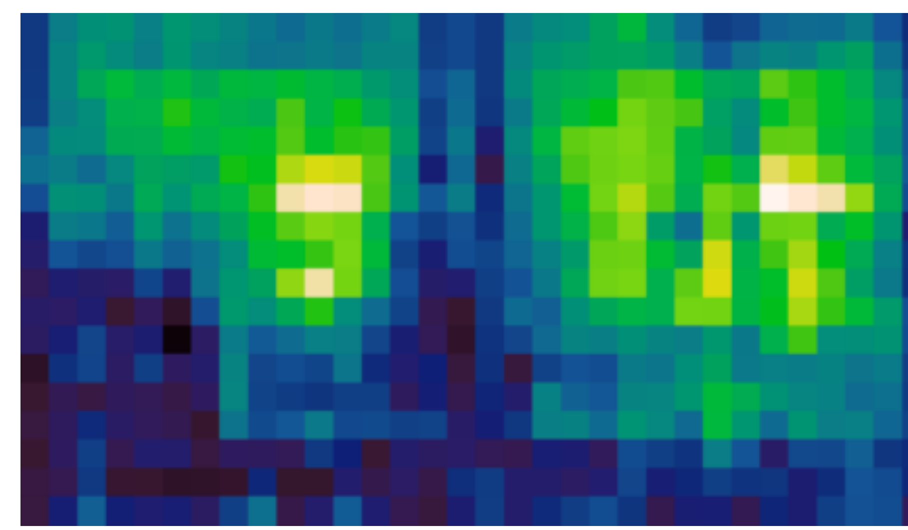
HOG feature map



Template



Detector response map



N. Dalal and B. Triggs, [Histograms of Oriented Gradients for Human Detection](), CVPR 2005

Source: S. Lazebnik

Example detections

Source: S. Lazebnik

[Dalal and Triggs, CVPR 2005]

Discriminative part-based models

- Single rigid template usually not enough to represent a category
  - Many objects (e.g. humans) are articulated, or have parts that can vary in configuration

  - Many object categories look very different from different viewpoints, or from instance to instance

Source: S. Lazebnik

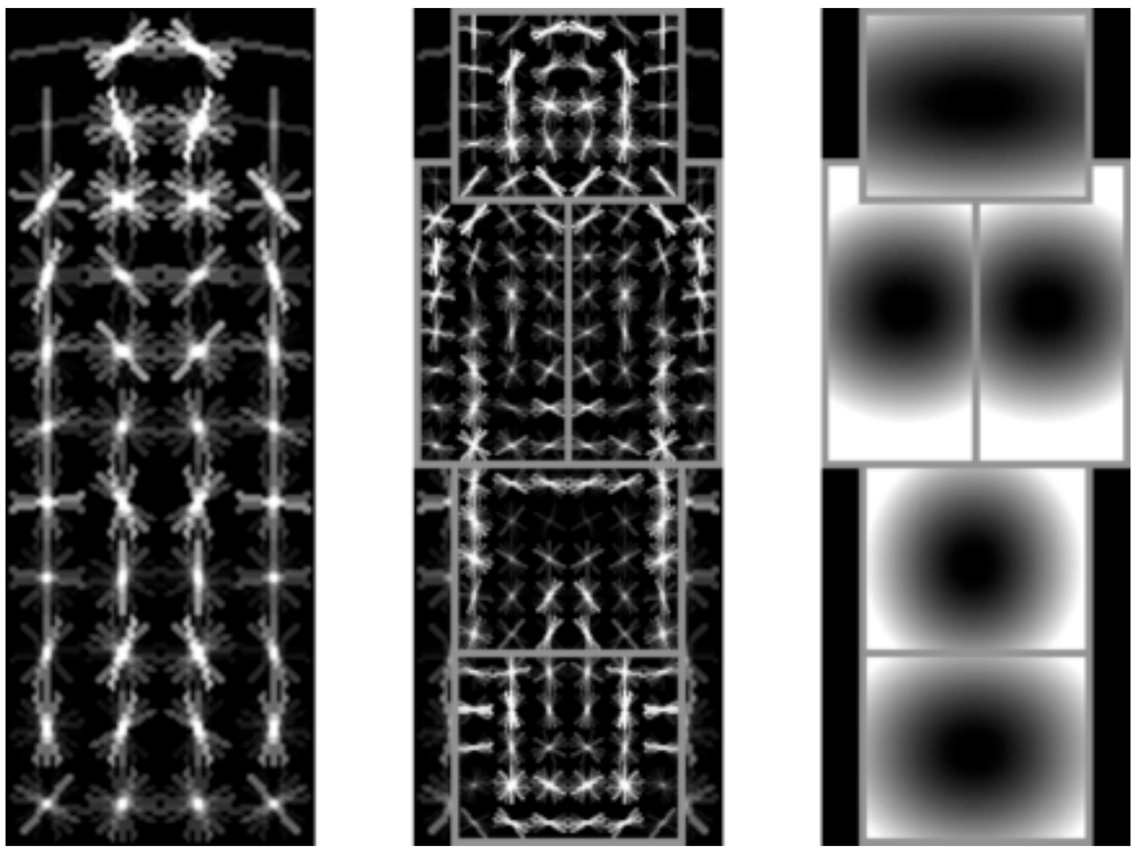
Slide by N. Snavely

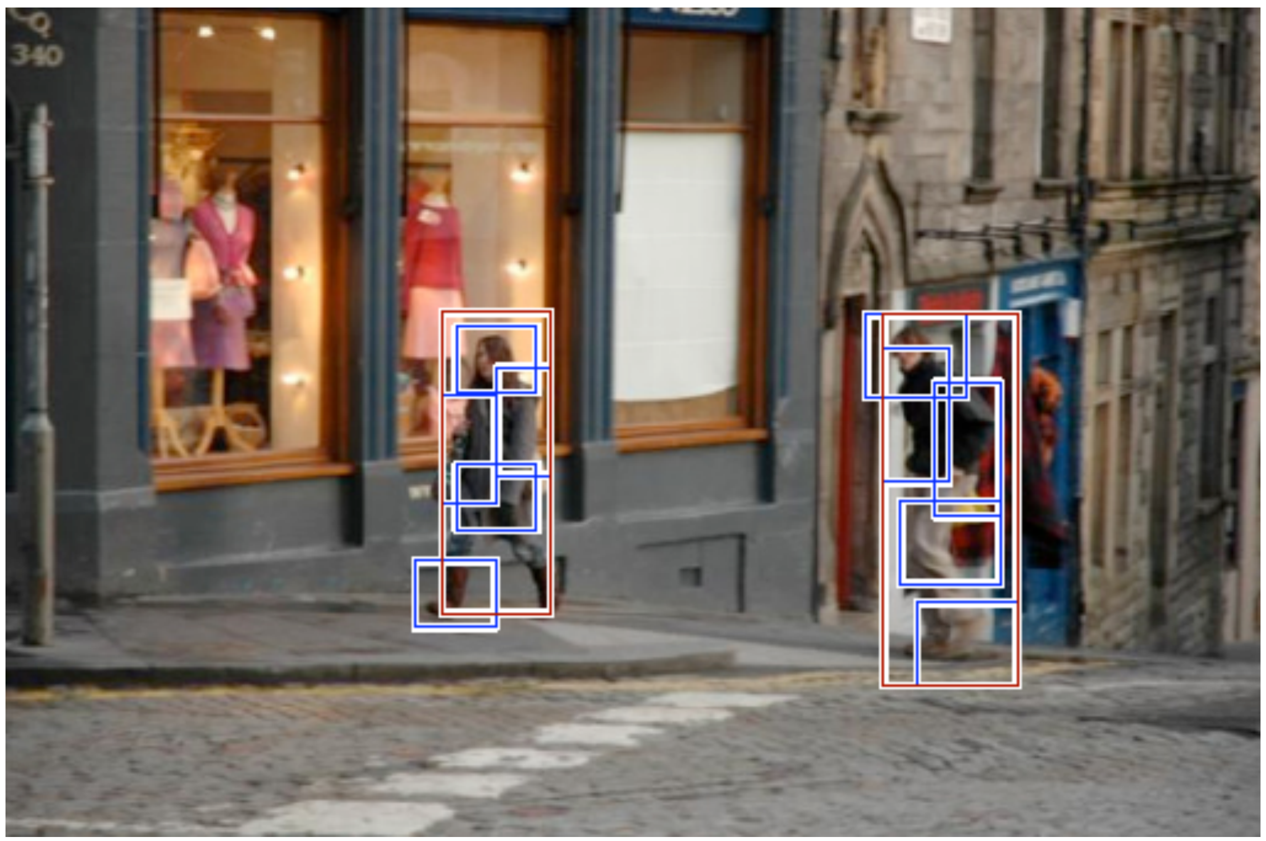
Discriminative part-based models

Root filter

Part filters

Deformation weights



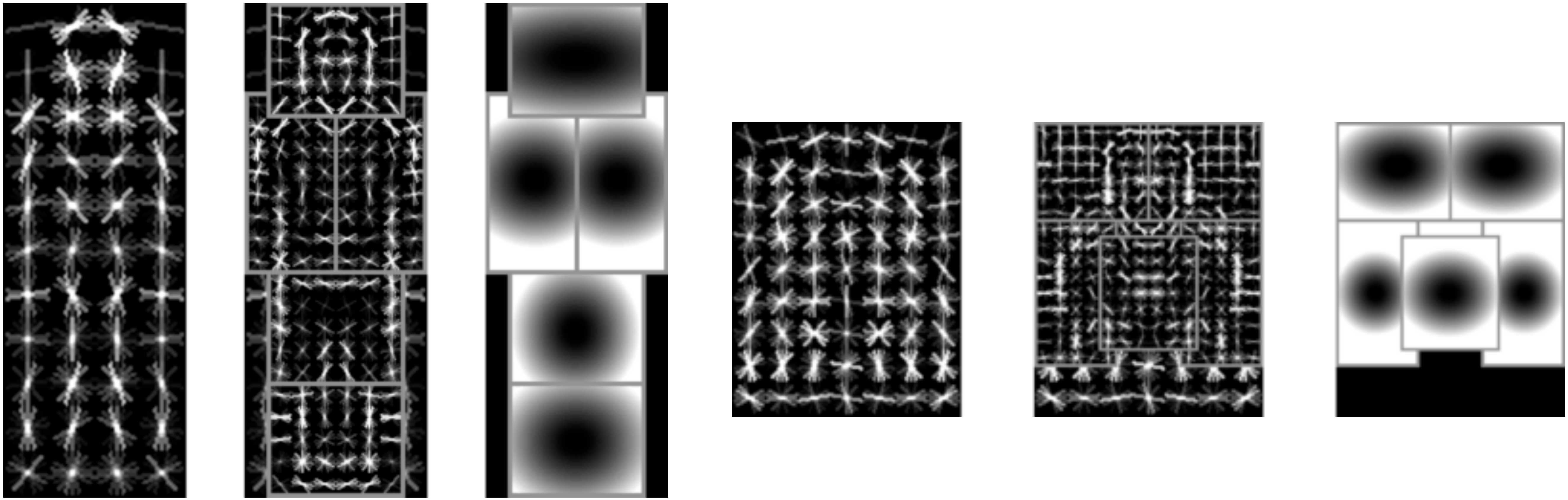


P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, [Object Detection with Discriminatively Trained Part Based Models](), PAMI 32(9), 2010

Source: S. Lazebnik

Discriminative part-based models

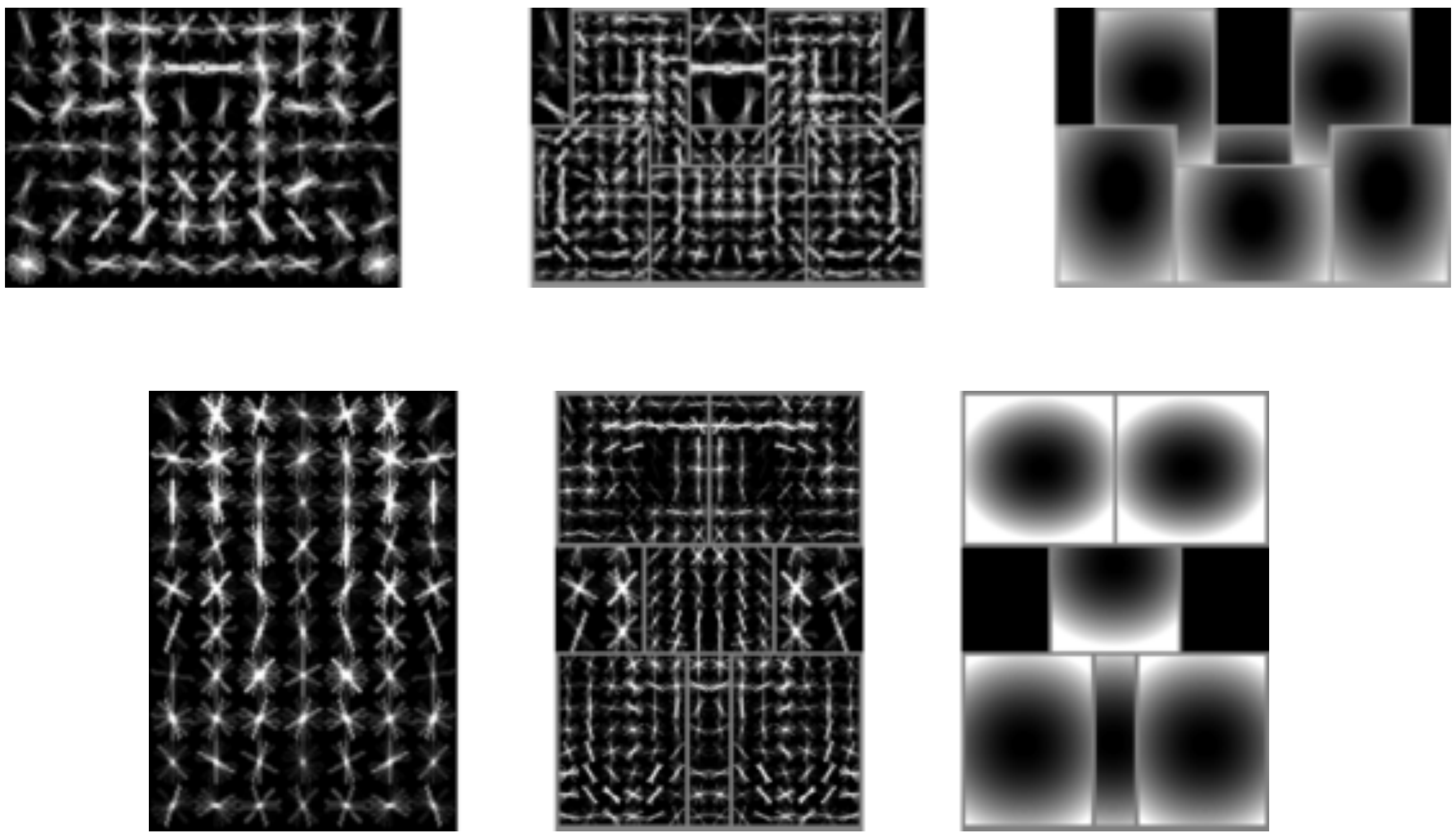
Multiple components

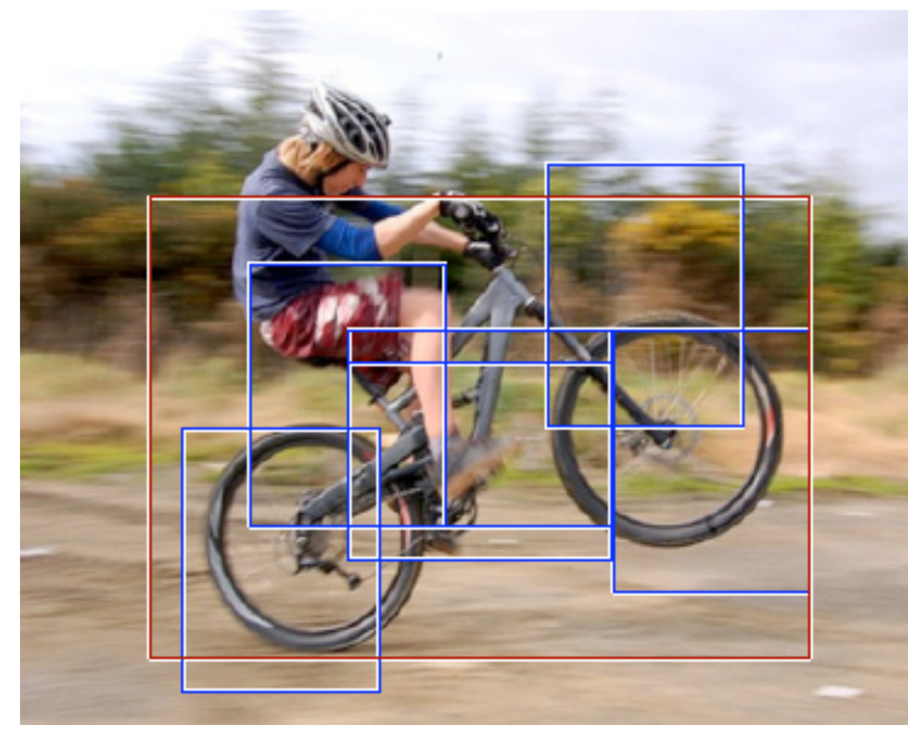


P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, [Object Detection with Discriminatively Trained Part Based Models](), PAMI 32(9), 2010

Source: S. Lazebnik

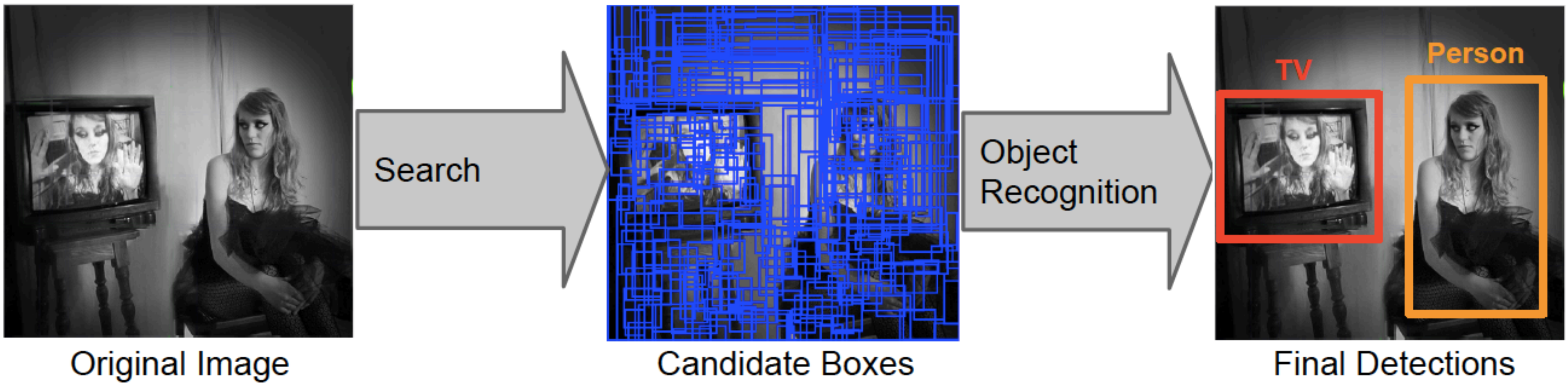
Discriminative part-based models





P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, [Object Detection with Discriminatively Trained Part Based Models](), PAMI 32(9), 2010

Conceptual approach: Proposal-driven detection



- Generate and evaluate a few hundred *region proposals*
  - Proposal mechanism can take advantage of low-level *perceptual organization* cues
  - Proposal mechanism can be category-specific or category-independent, hand-crafted or trained
  - Classifier can be slower but more powerful

Source: S. Lazebnik

Selective search for detection

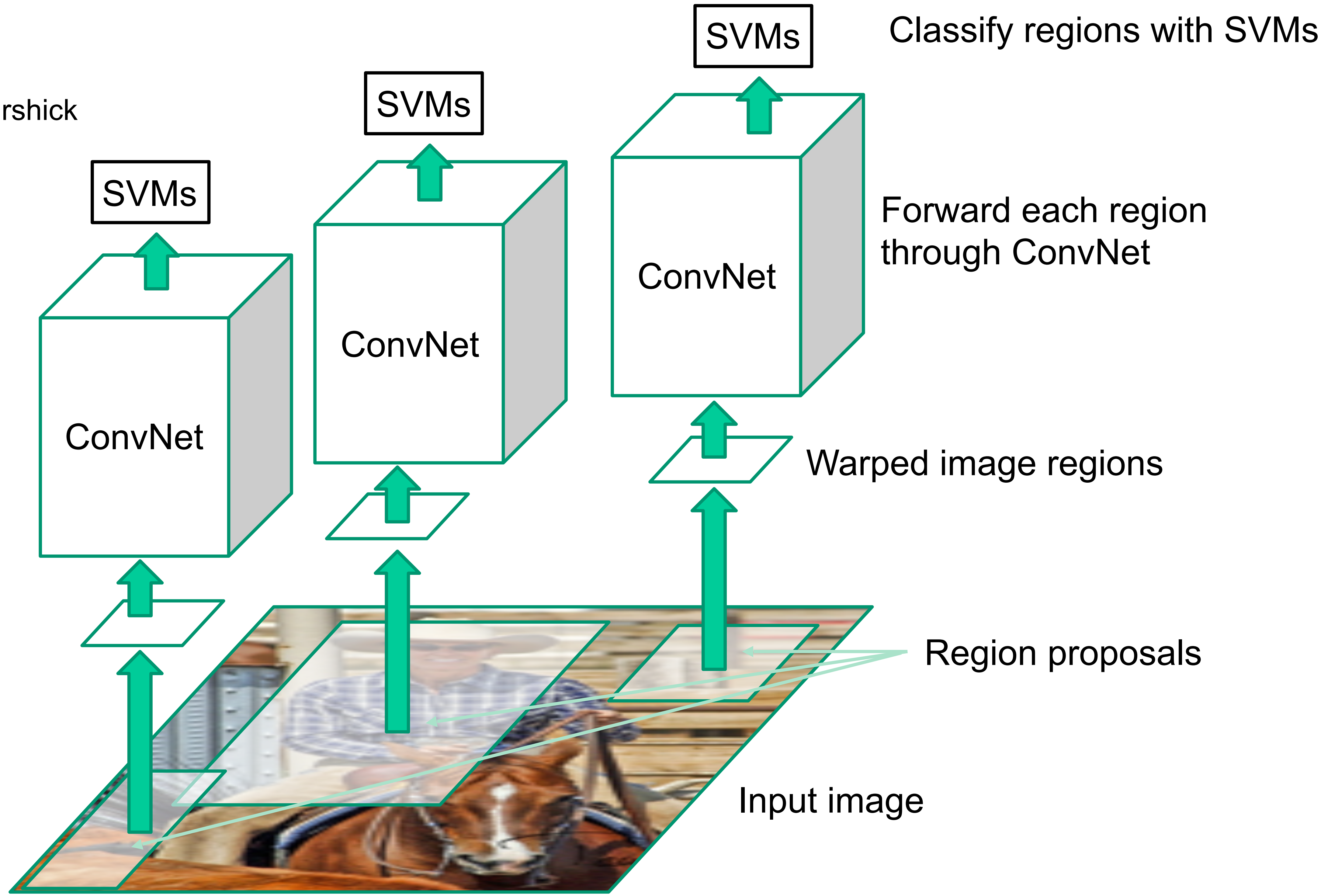
- Use hierarchical segmentation: start with small *superpixels* and merge based on diverse cues

Input Image

J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, [Selective Search for Object Recognition](), IJCV 2013

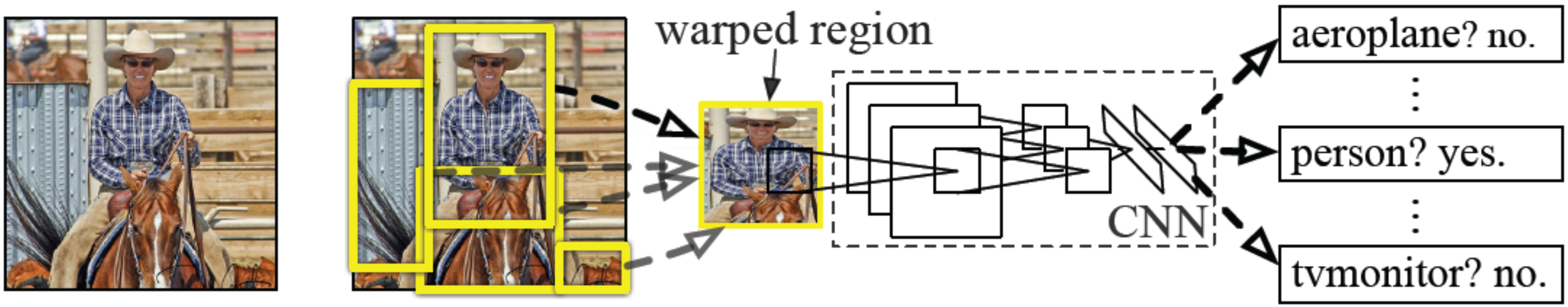
R-CNN: Region proposals + CNN features

Source: R. Girshick



R. Girshick, J. Donahue, T. Darrell, and J. Malik, [Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation](), CVPR 2014.

R-CNN details



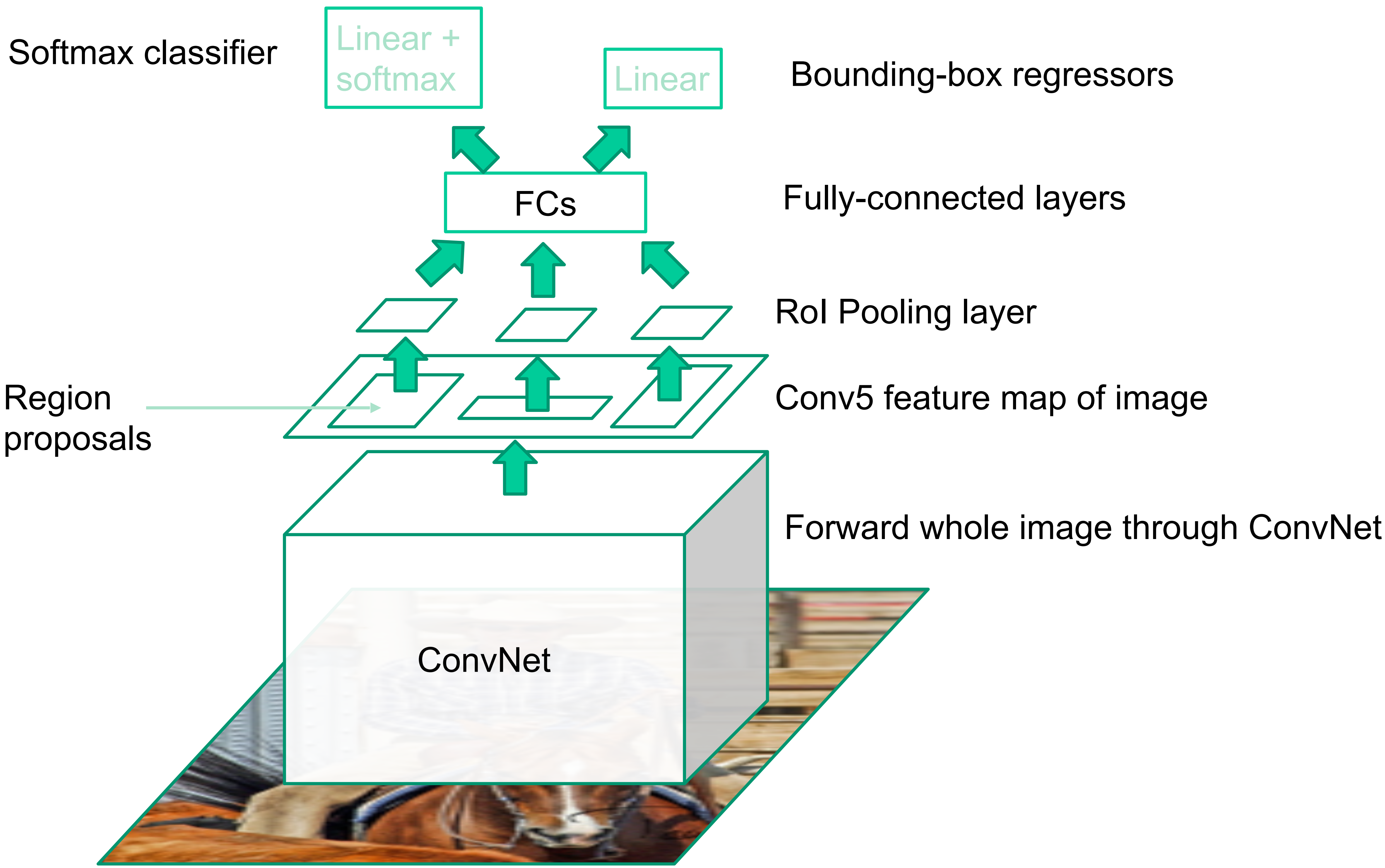
- **Regions**: ~2000 Selective Search proposals
- **Network**: AlexNet *pre-trained* on ImageNet (1000 classes), *fine-tuned* on PASCAL (21 classes)
- **Final detector**: warp proposal regions, extract fc7 network activations (4096 dimensions), classify with linear SVM
- **Bounding box regression** to refine box locations
- **Performance:** mAP of **53.7%** on PASCAL 2010 (vs. **35.1%** for Selective Search and **33.4%** for Deformable Part Models)

Source: S. Lazebnik

R-CNN pros and cons

- Pros
  - Accurate!
  - Any deep architecture can immediately be “plugged in”
- Cons
  - Not a single end-to-end system
    - 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
    - 2000 CNN passes per image
  - Inference (detection) is slow (47s / image with VGG16)

Fast R-CNN

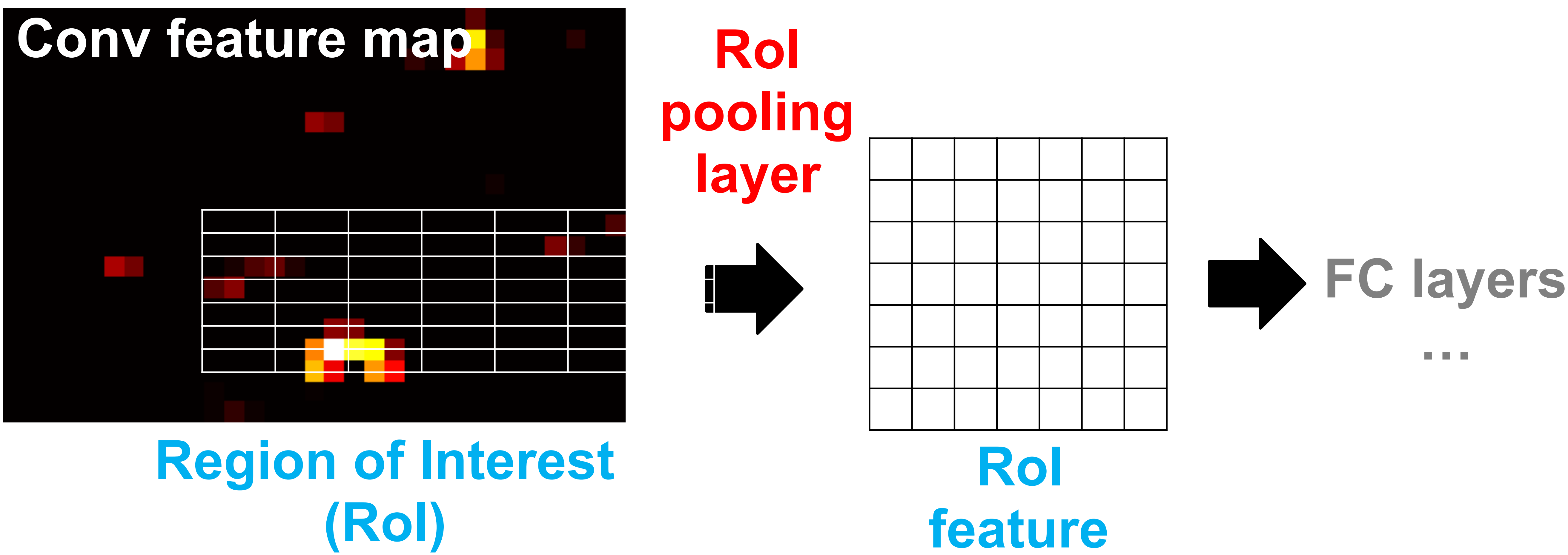


Source: R. Girshick

R. Girshick, [Fast R-CNN](), ICCV 2015

RoI pooling

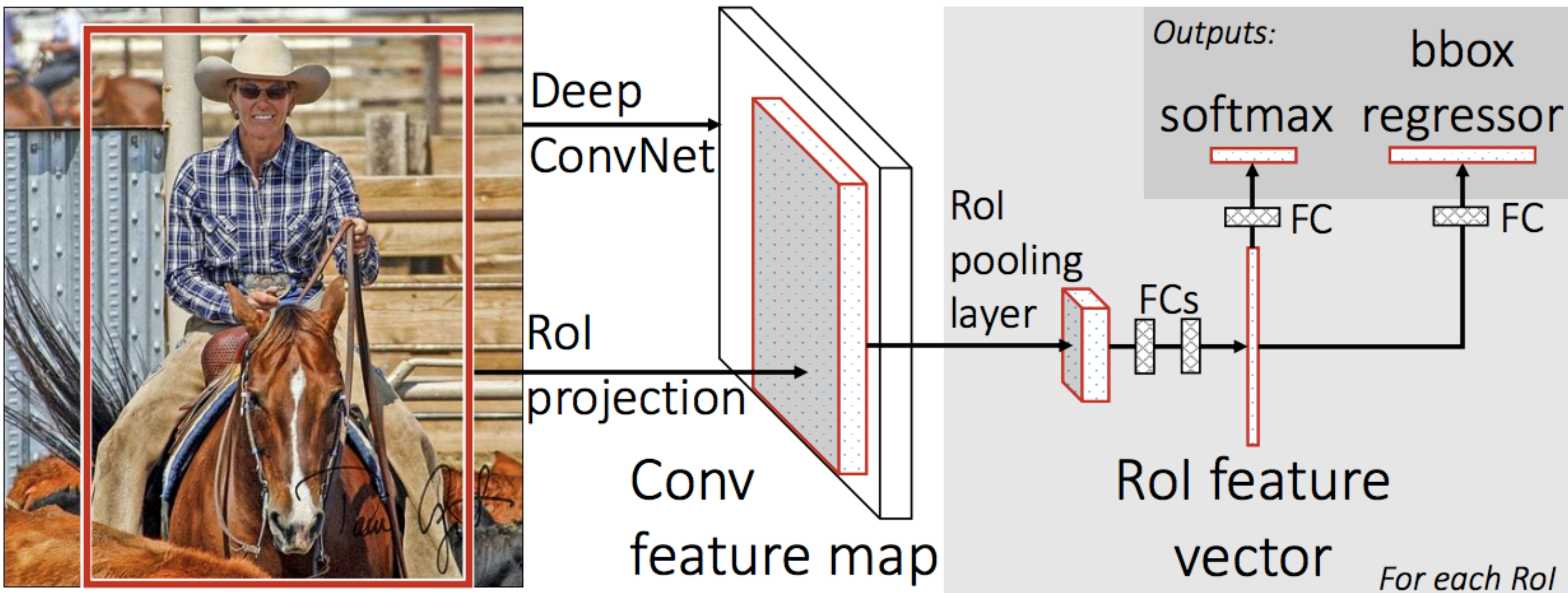
- "Crop and resample" a fixed-size feature representing a region of interest out of the outputs of the last conv layer
  - Use nearest-neighbor interpolation of coordinates, max pooling



Source: R. Girshick, K. He

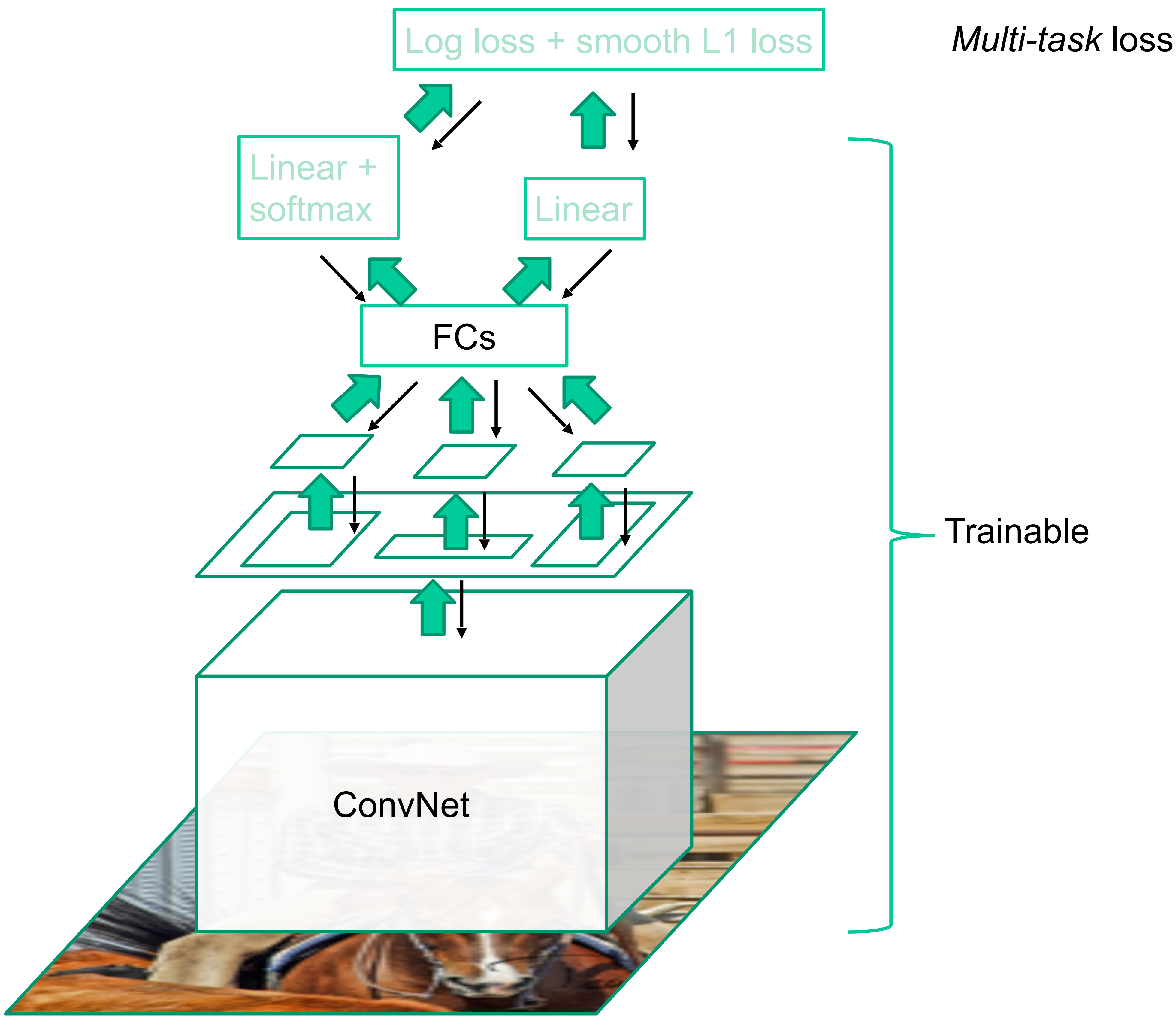
Prediction

- For each RoI, network predicts probabilities for C+1 classes (class 0 is background) and four bounding box offsets for C classes



R. Girshick, [Fast R-CNN](), ICCV 2015

Fast R-CNN training



Source: R. Girshick

R. Girshick, [Fast R-CNN](), ICCV 2015

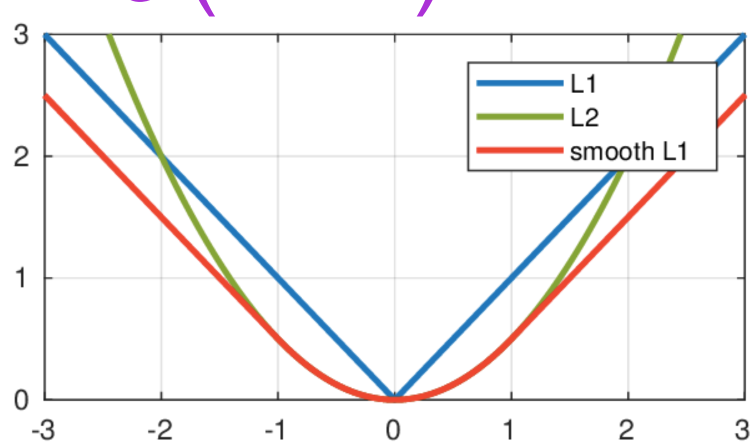
Multi-task loss

- Loss for ground truth class y, predicted class probabilities $P(y)$, ground truth box b, and predicted box $\hat{b}$:

$$L\left(y,\ P,\ b, \hat{b}\right) = \underbrace{-\log P(y)}_{\text{softmax loss}} + \lambda \mathbb{1}[y \geq 1] \underbrace{L_{\text{reg}}(b, \hat{b})}_{\text{regression loss}}$$

- Regression loss: ***smooth L1 loss*** on top of log space offsets relative to proposal

$$L_{\text{reg}}\left(b, \hat{b}\right) = \sum \text{smooth}_{L_1}(b_i - \hat{b}_i)$$



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

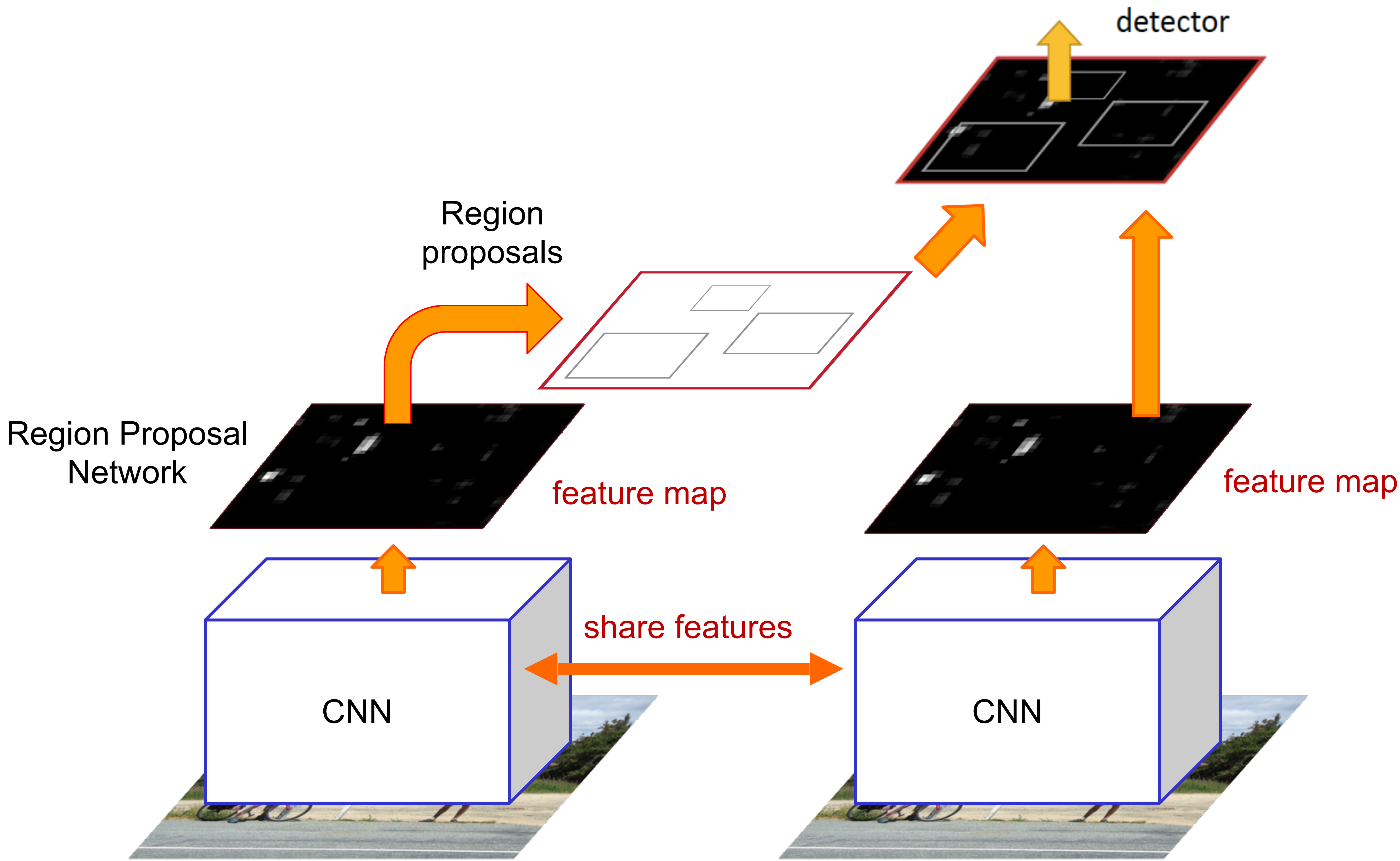
Fast R-CNN results

	Fast R-CNN	**R-CNN**
Train time (h)	**9.5**	84
- Speedup	**8.8x**	1x
Test time / image	**0.32s**	47.0s
Test speedup	**146x**	1x
mAP	**66.9%**	66.0%

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

Source: R. Girshick

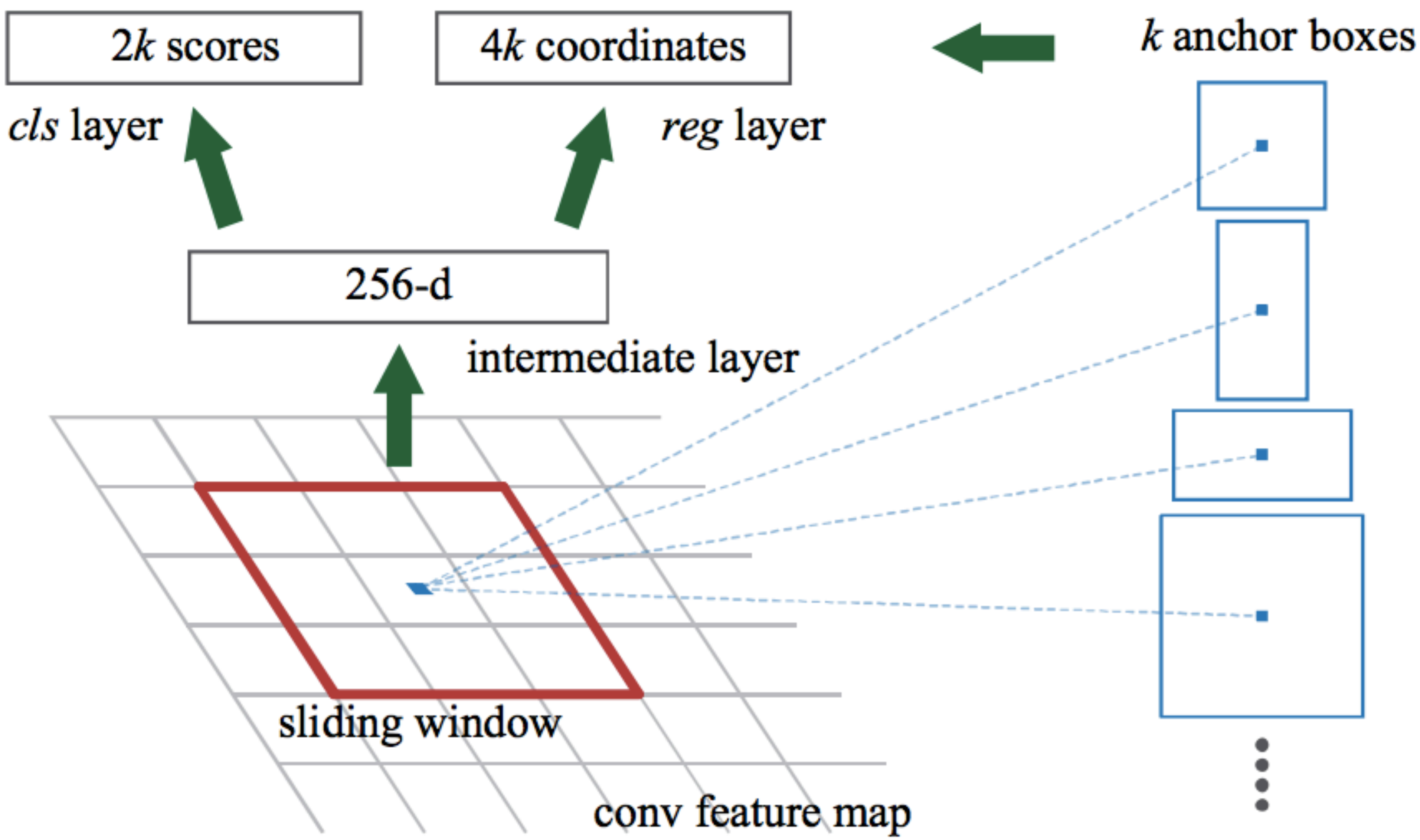
Faster R-CNN



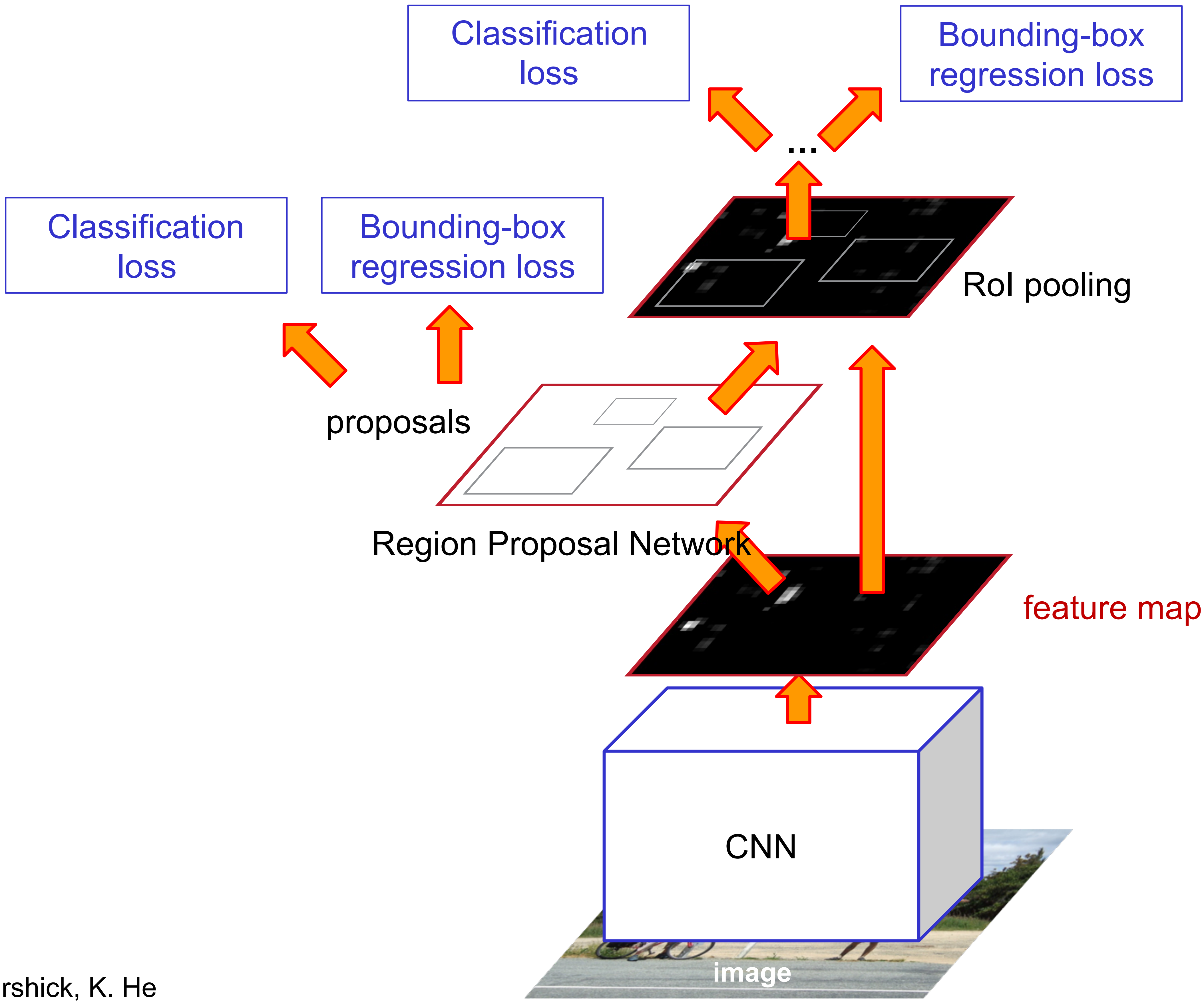
S. Ren, K. He, R. Girshick, and J. Sun, [Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks](), NIPS 2015

Region proposal network (RPN)

- Slide a small window (3x3) over the conv5 layer
  - Predict object/no object
  - Regress bounding box coordinates with reference to *anchors* (3 scales x 3 aspect ratios)



One network, four losses



Source: R. Girshick, K. He

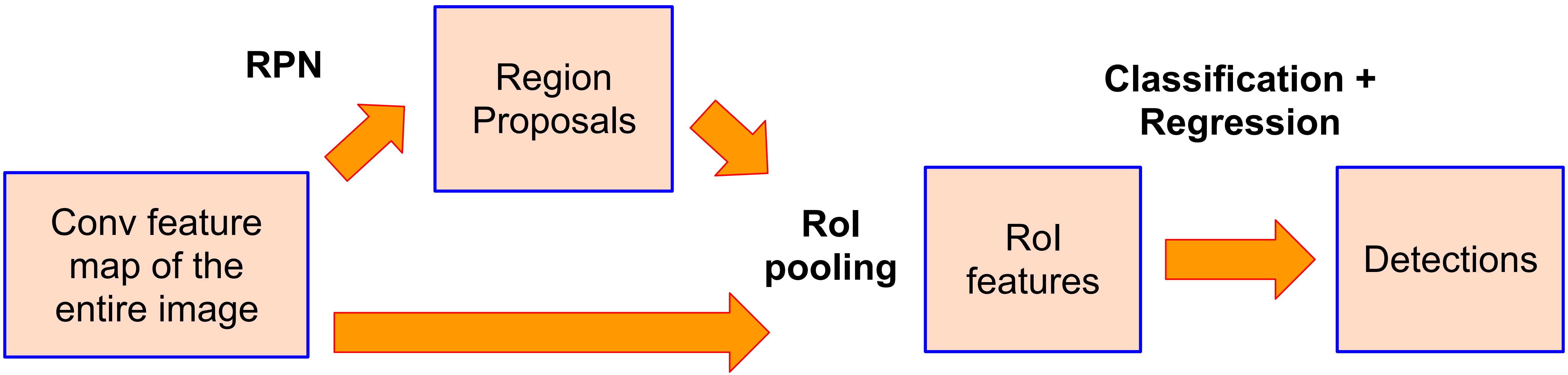
Faster R-CNN results

system	time	07 data	07+12 data
R-CNN	~50s	66.0	-
Fast R-CNN	~2s	66.9	70.0
Faster R-CNN	198ms	**69.9**	**73.2**

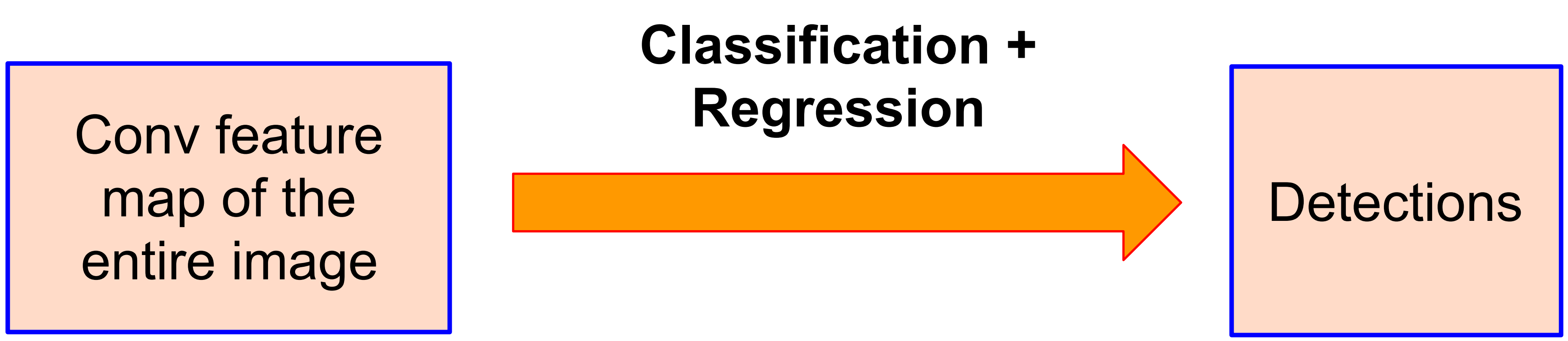
detection mAP on PASCAL VOC 2007, with VGG-16 pre-trained on ImageNet

Streamlined detection architectures

- The Faster R-CNN pipeline separates proposal generation and region classification:



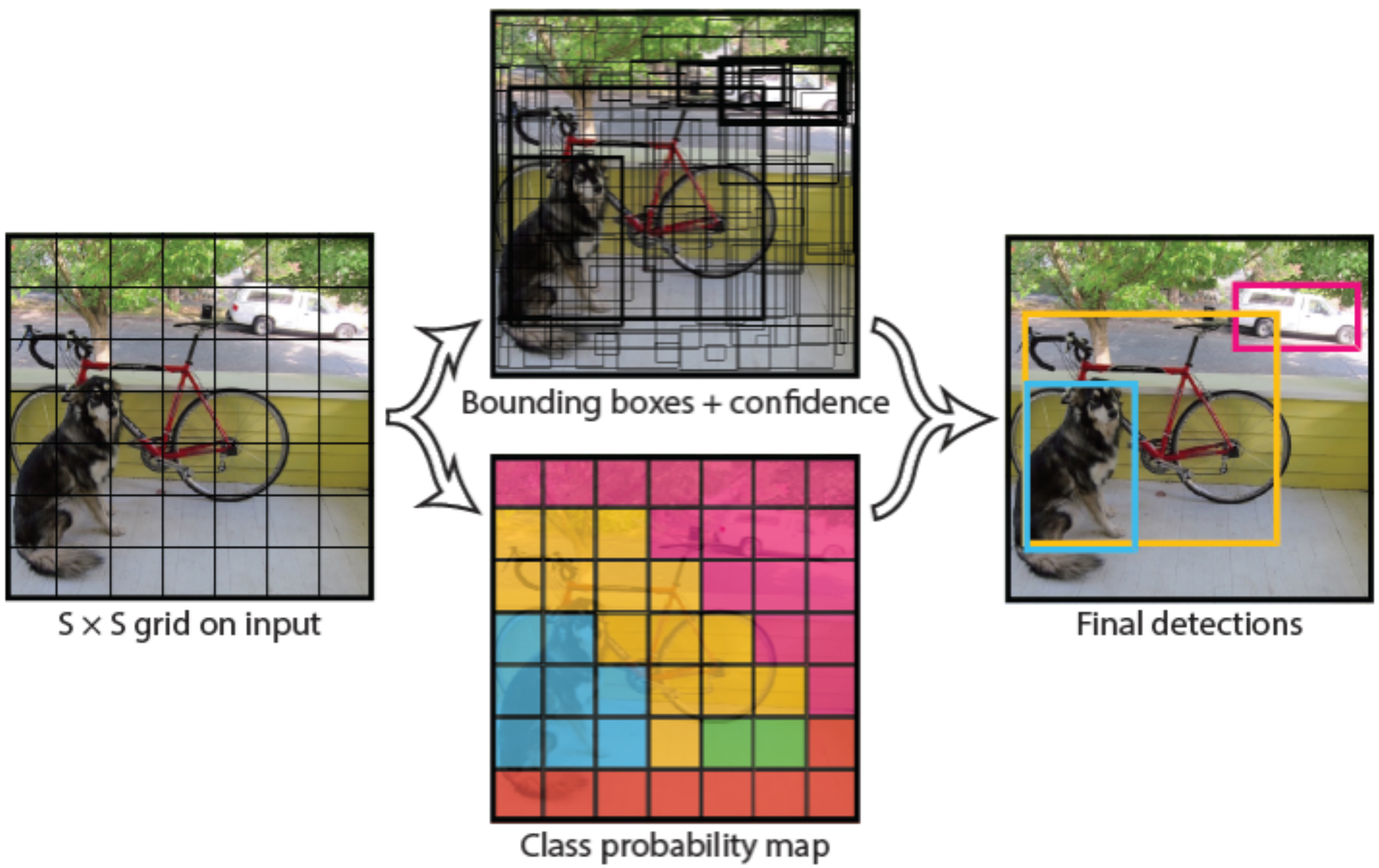
- Is it possible do detection in one shot?



Source: S. Lazebnik

YOLO

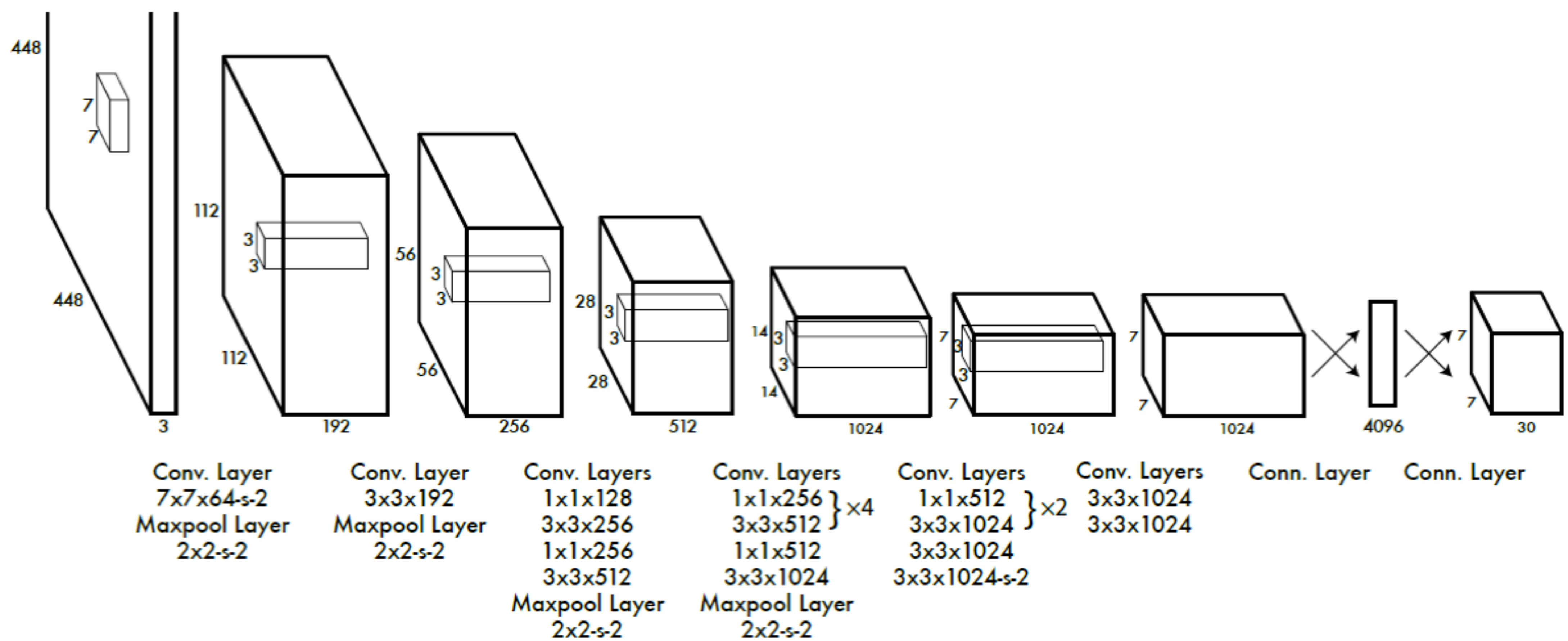
- Divide the image into a coarse grid and directly predict class label and a few candidate boxes for each grid cell



J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, [You Only Look Once: Unified, Real-Time Object Detection](), CVPR 2016

YOLO

1. Take conv feature maps at 7x7 resolution
2. Add two FC layers to predict, at each location, a score for each class and 2 bboxes w/ confidences
   - For PASCAL, output is 7x7x30 (30 = 20 + 2*(4+1))



J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, [You Only Look Once: Unified, Real-Time Object Detection](), CVPR 2016

Source: S. Lazebnik

YOLO

- Objective function:

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

Regression

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

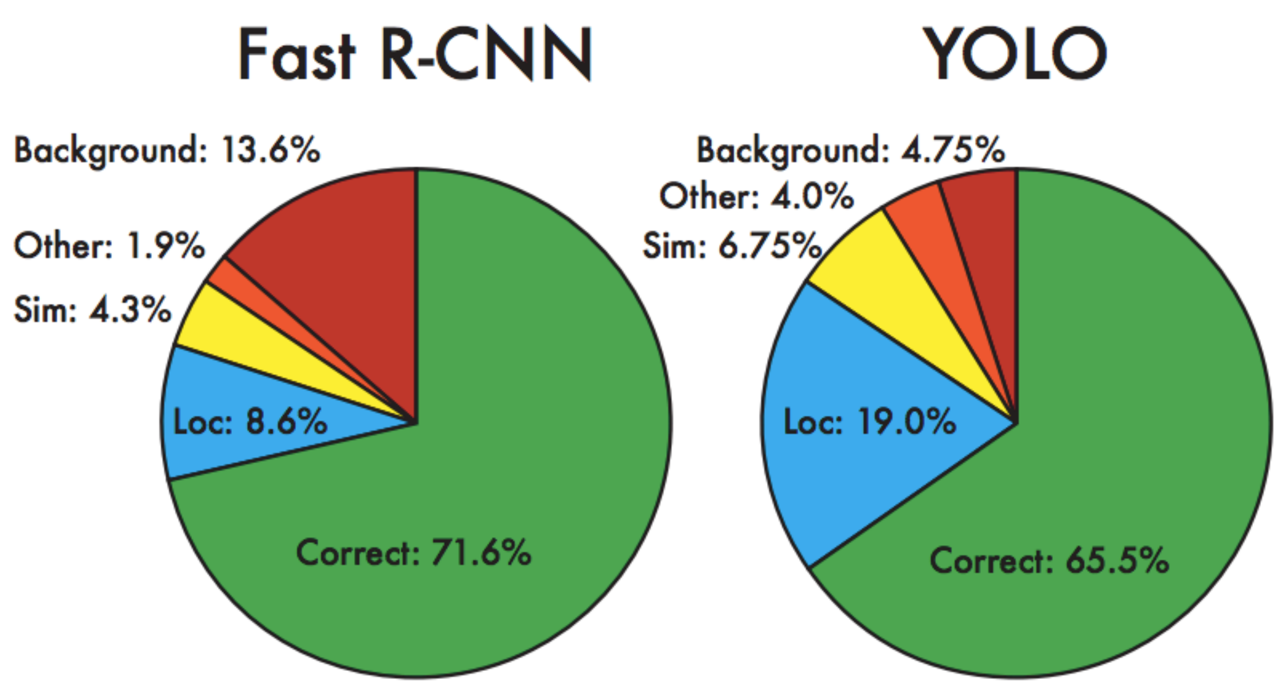
Object/no object confidence

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

Class prediction

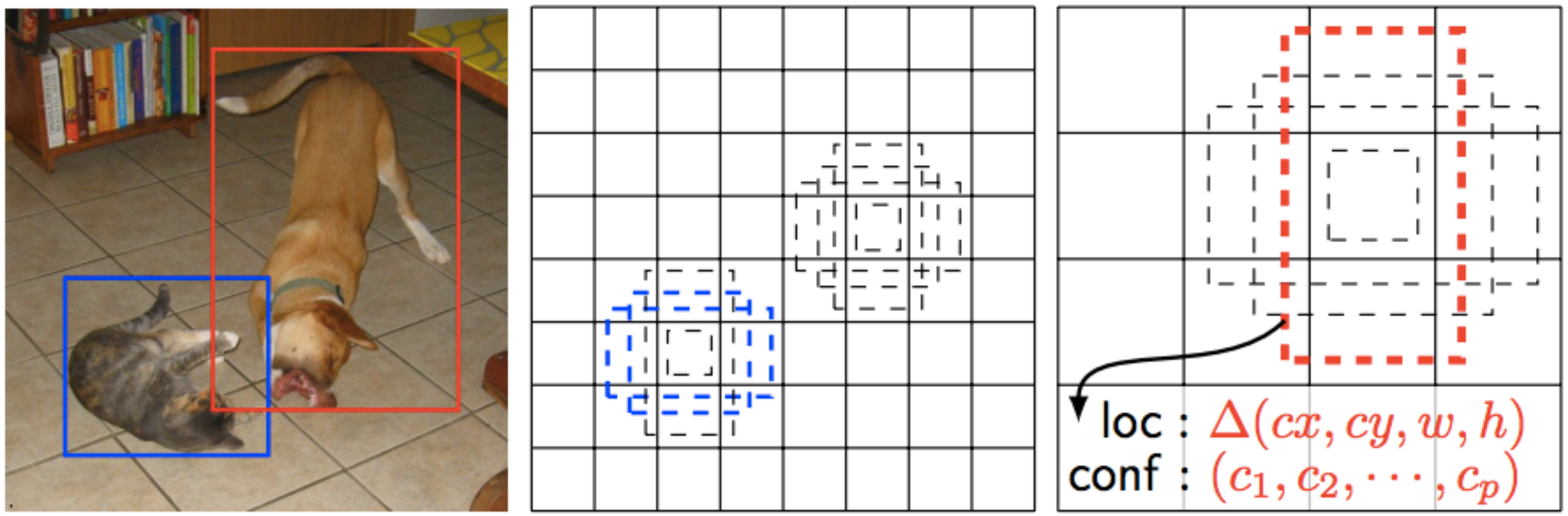
YOLO: Results

- Each grid cell predicts only two boxes and can only have one class – this limits the number of nearby objects that can be predicted
- Localization accuracy suffers compared to Fast(er) R-CNN due to coarser features, errors on small boxes
- 7x speedup over Faster R-CNN (45-155 FPS vs. 7-18 FPS)



Performance on PASCAL 2007

SSD

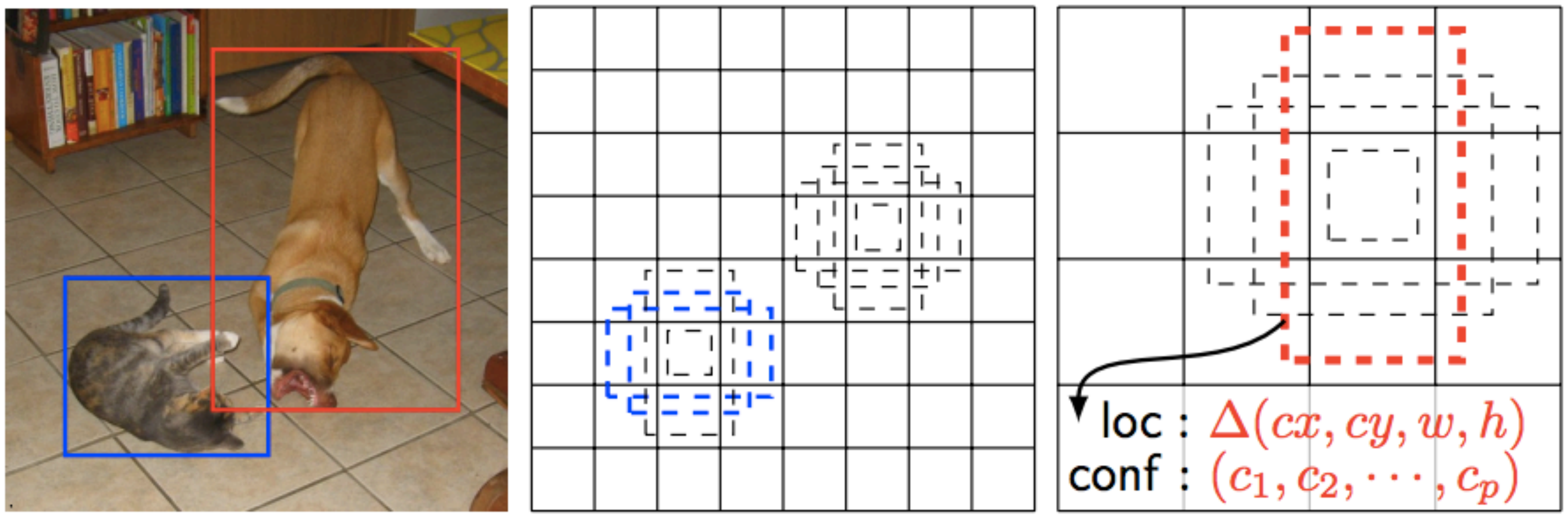


(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

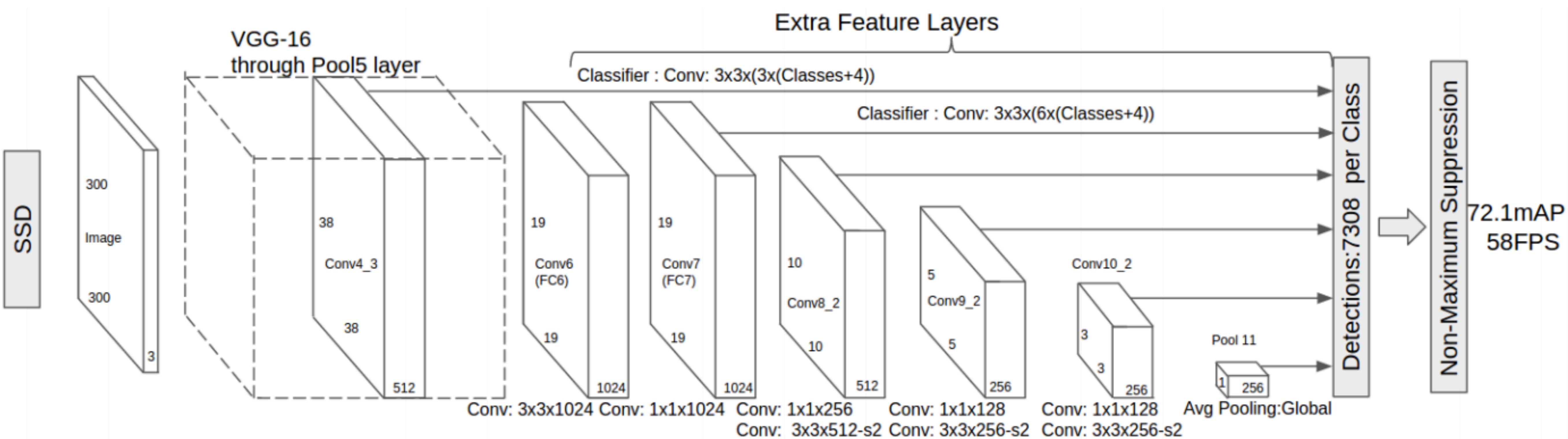
- Similarly to YOLO, predict bounding boxes directly from conv maps
- Unlike YOLO, do not use FC layers and predict different size boxes from conv maps at different resolutions
- Similarly to RPN, use anchors

W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Berg, SSD: Single Shot MultiBox Detector, ECCV 2016.

SSD



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



W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. Berg, [SSD: Single Shot MultiBox Detector](), ECCV 2016.

Object detection overview

- Different choices for “base network”:
  - VGG, ResNet…
- Object detection architecture:
  - Region-based: R-CNN, Fast R-CNN, Faster R-CNN
  - Single shot: YOLO, SSD
- Faster R-CNN is slower but more accurate.
- YOLO/SSD are faster but less accurate.

Huang et al. “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017