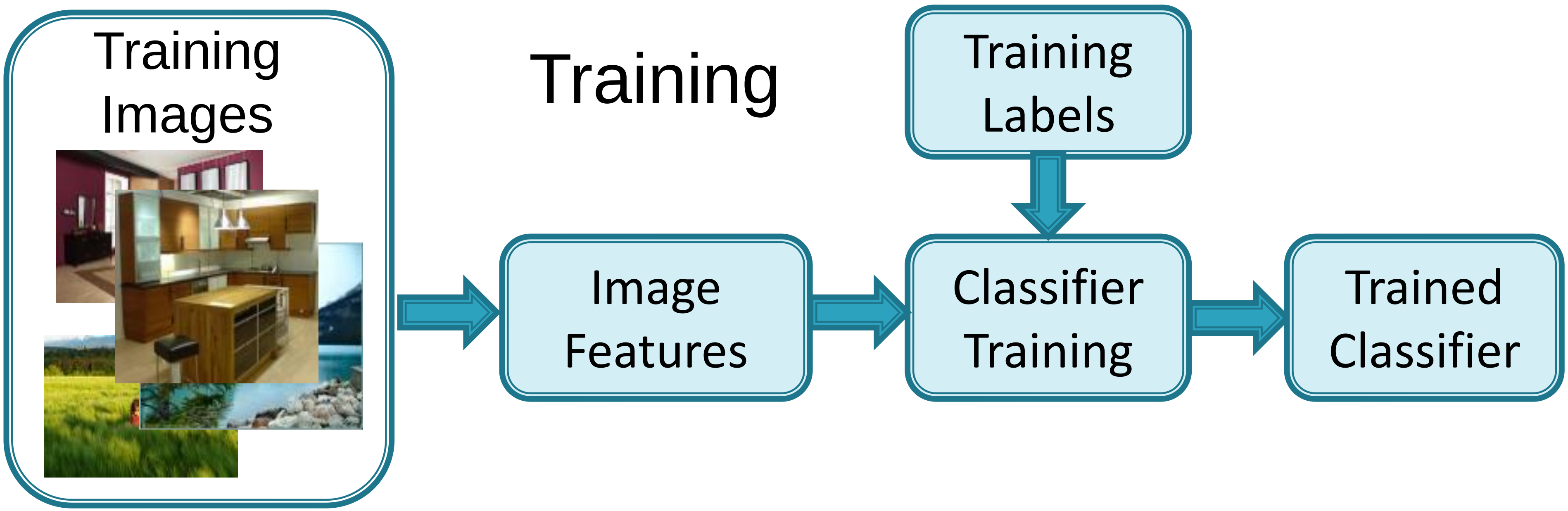
Deep Learning for Visual Recognition

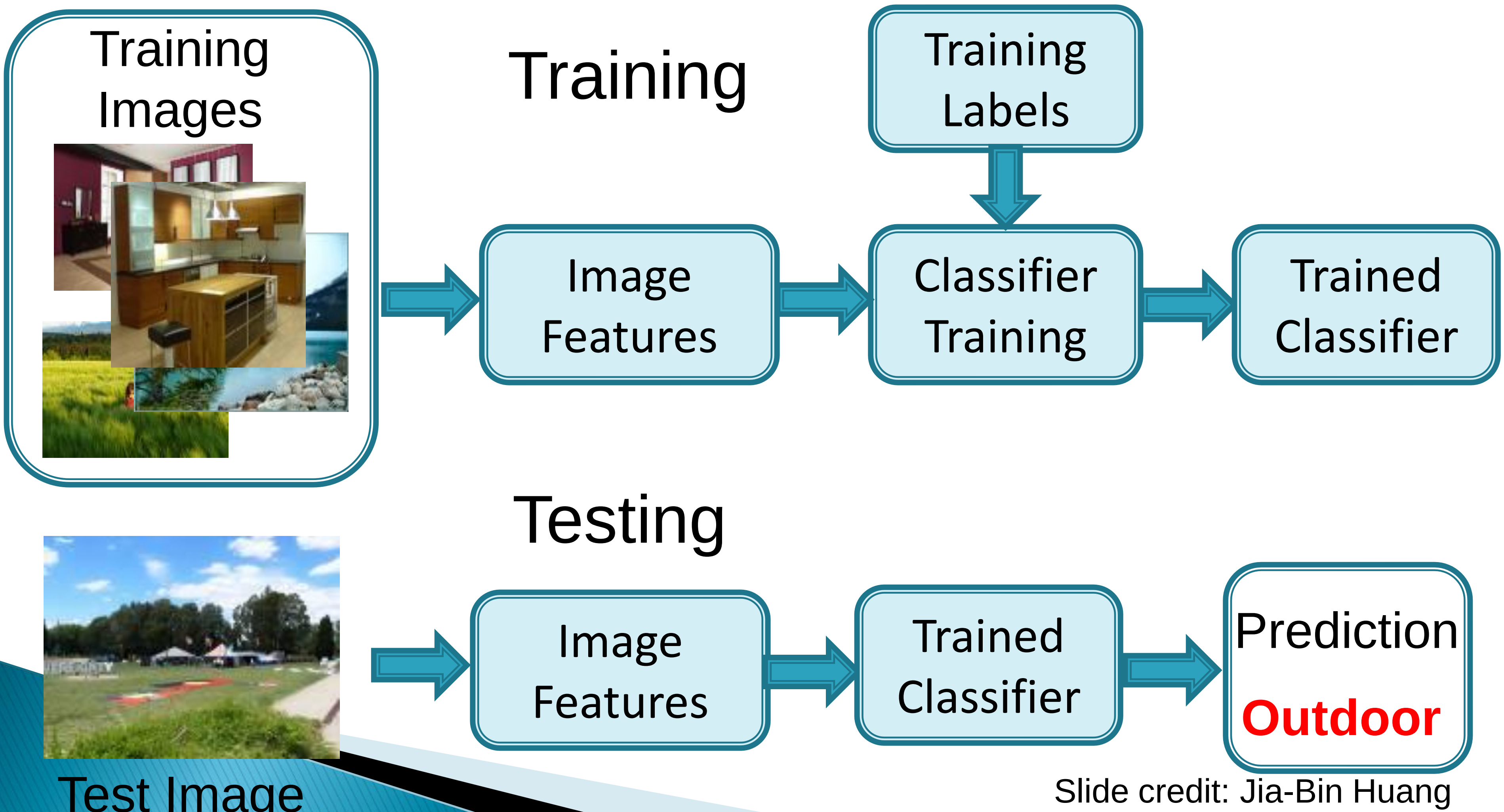
Introduction

Traditional Image Categorization: Training phase



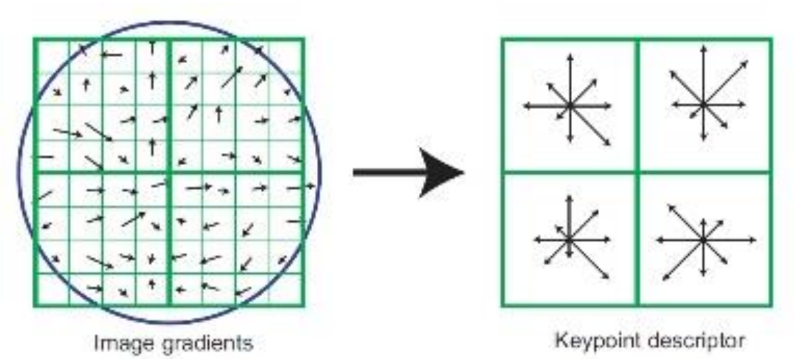
Slide credit: Jia-Bin Huang

Traditional Image Categorization: Testing phase

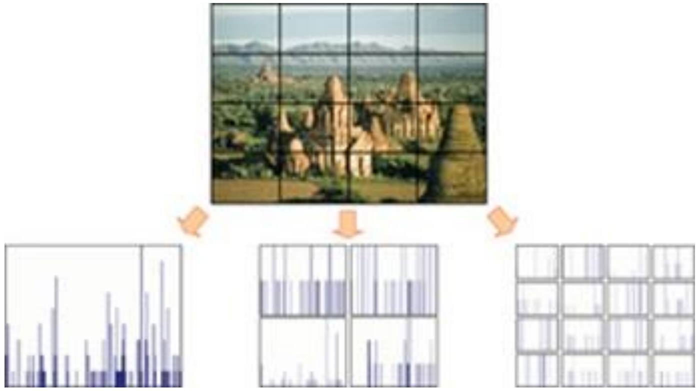


Slide credit: Jia-Bin Huang

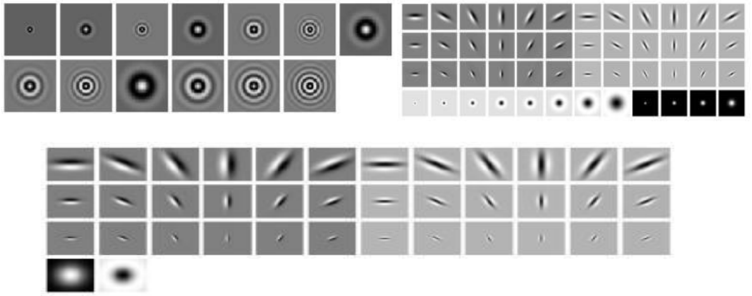
Feature Design



SIFT [Lowe IJCV 04]



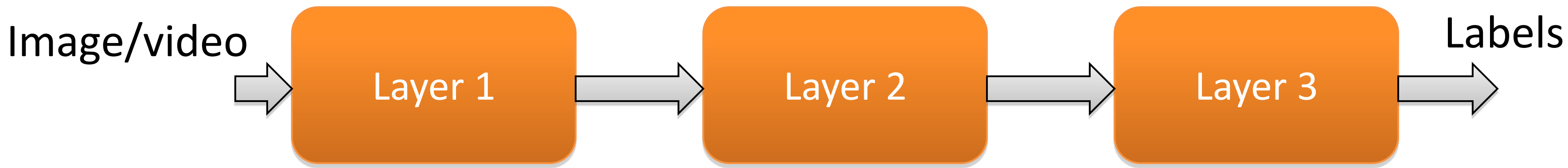
Harris Corner



Textons

Hierarchy of Feature Extractors

- Each layer of hierarchy extracts features from output of previous layer
- All the way from pixels → classifier
- Layers have the (nearly) same structure

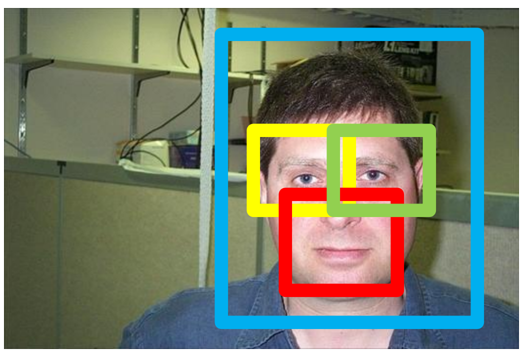


- Train all layers jointly

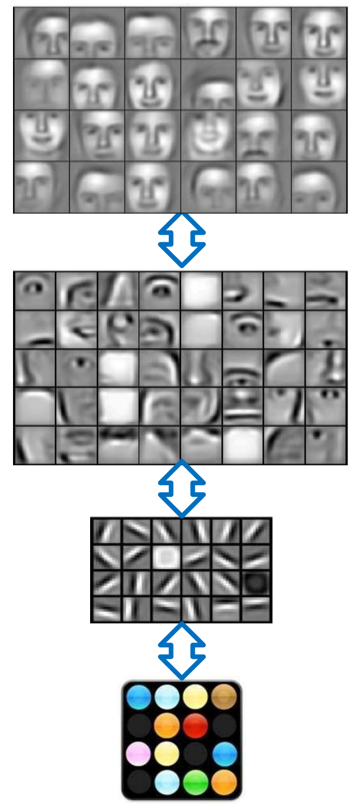
Learning Feature Hierarchy

Goal: Learn **useful higher-level features** from images

Input data



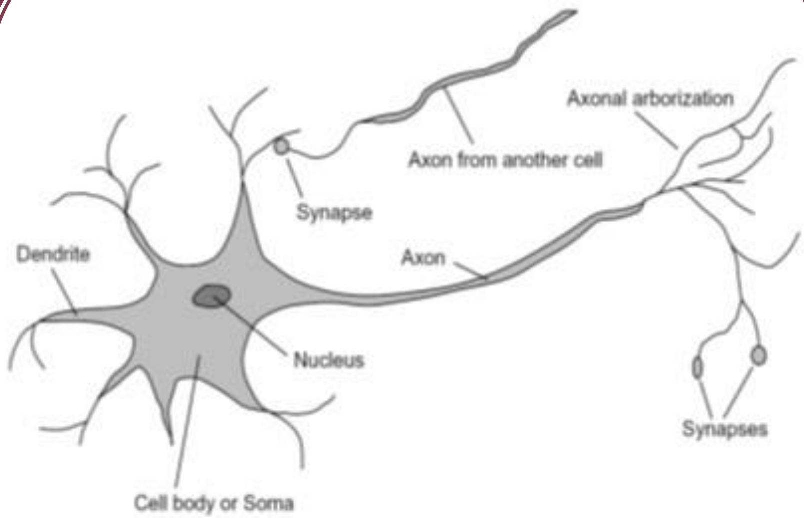
Feature representation



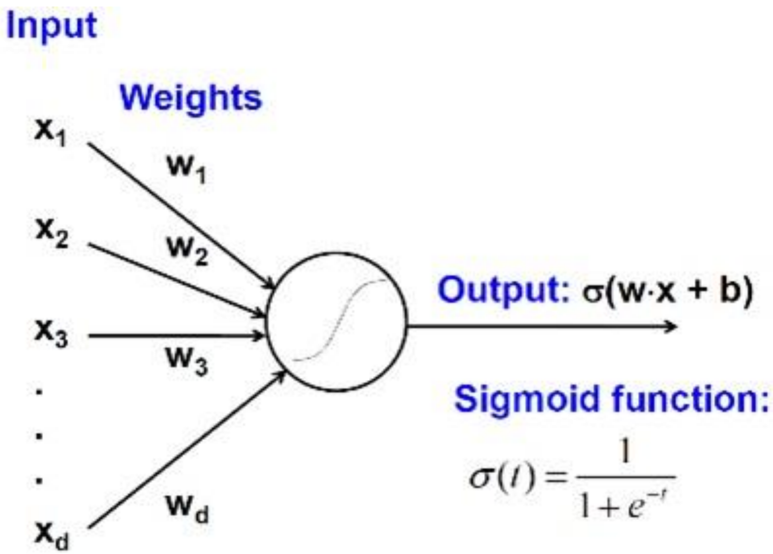
Lee et al., ICML 2009;
CACM 2011

Slide: Rob Fergus

Biological neuron and Perceptrons



A biological neuron

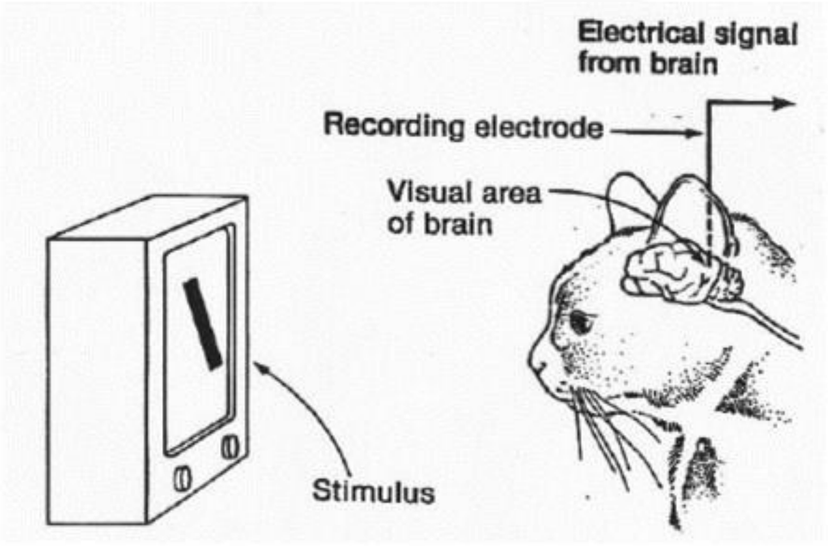


An artificial neuron (Perceptron)
- a linear classifier

Slide credit: Jia-Bin Huang

Simple, Complex and Hypercomplex cells

David H. Hubel and Torsten Wiesel

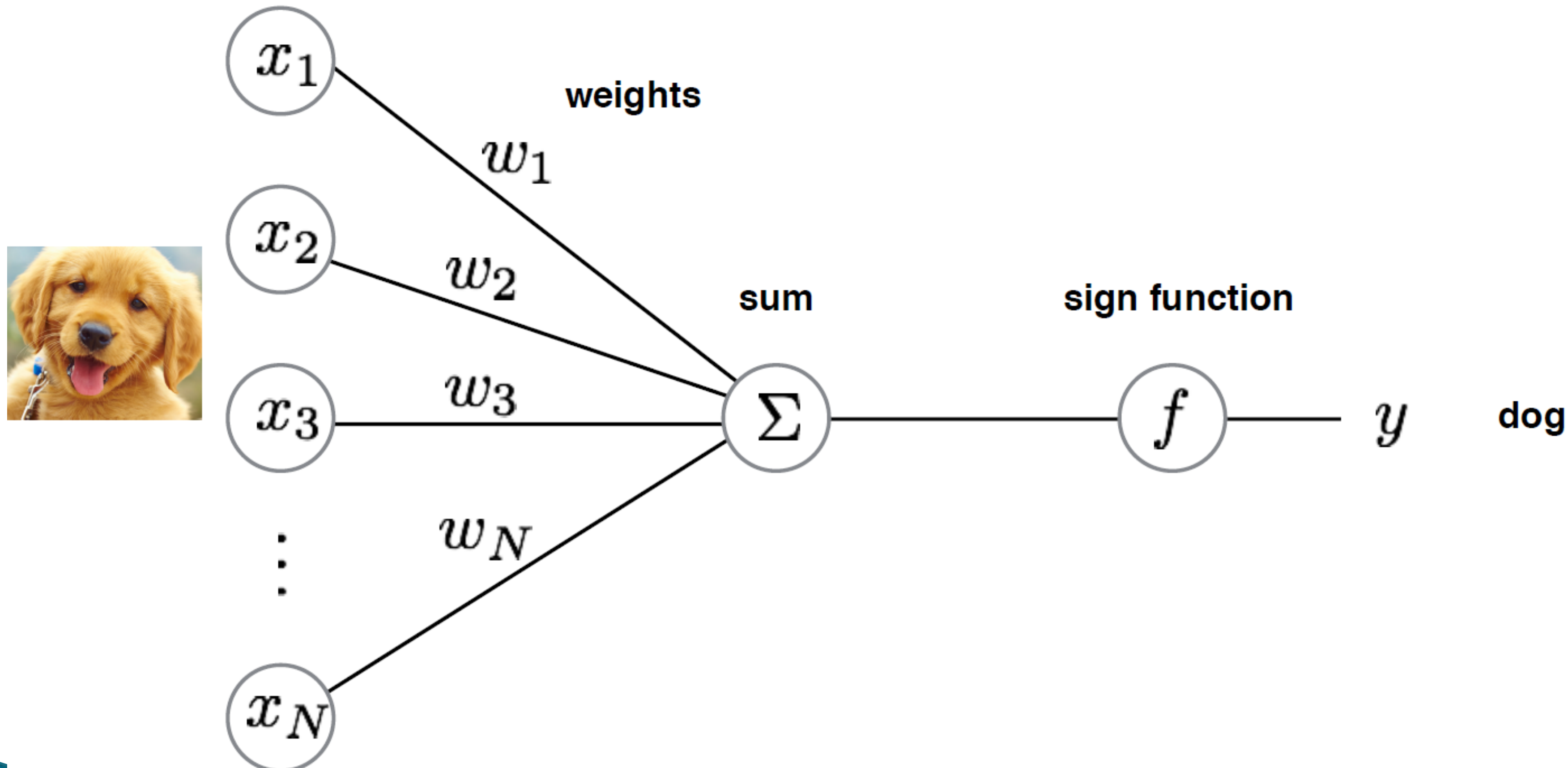


Suggested a **hierarchy** of **feature detectors** in the visual cortex, with higher level features responding to patterns of activation in lower level cells, and propagating activation upwards to still higher level cells.

David Hubel's [Eye, Brain, and Vision]

Slide credit: Jia-Bin Huang

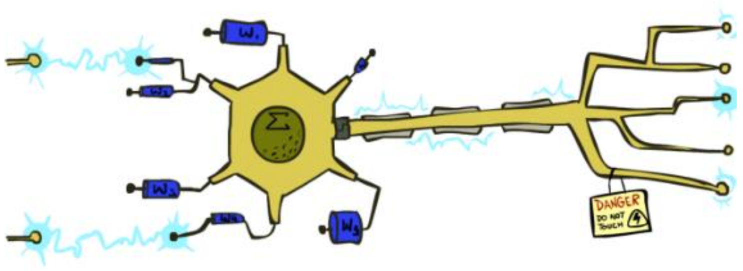
Perceptron: for image classification



Slide credit: Juan Carlos Niebles

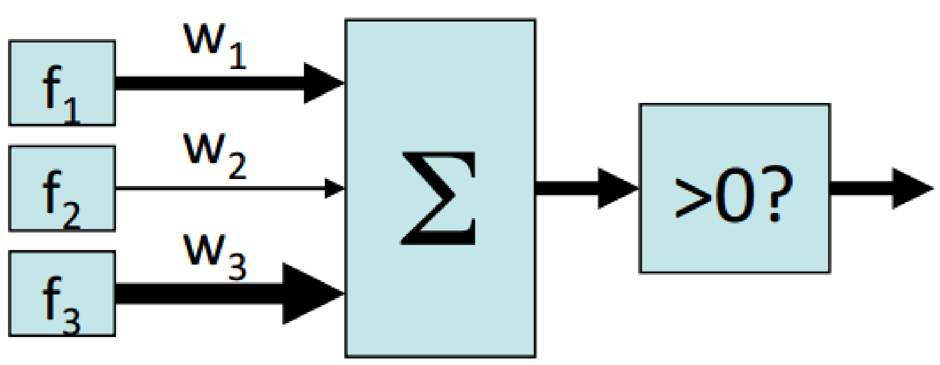
Neuron: Linear Perceptron

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output −1



Slide credit: Pieter Abeel and Dan Klein

Perceptron training algorithm

- Initialize weights **w** randomly
- Cycle through training examples in multiple pa sses (*epochs*)
- For each training example **x** with label y:
  - Classify with current weights:

$$y' = \mathrm{sgn}(\mathbf{w} \times \mathbf{x})$$

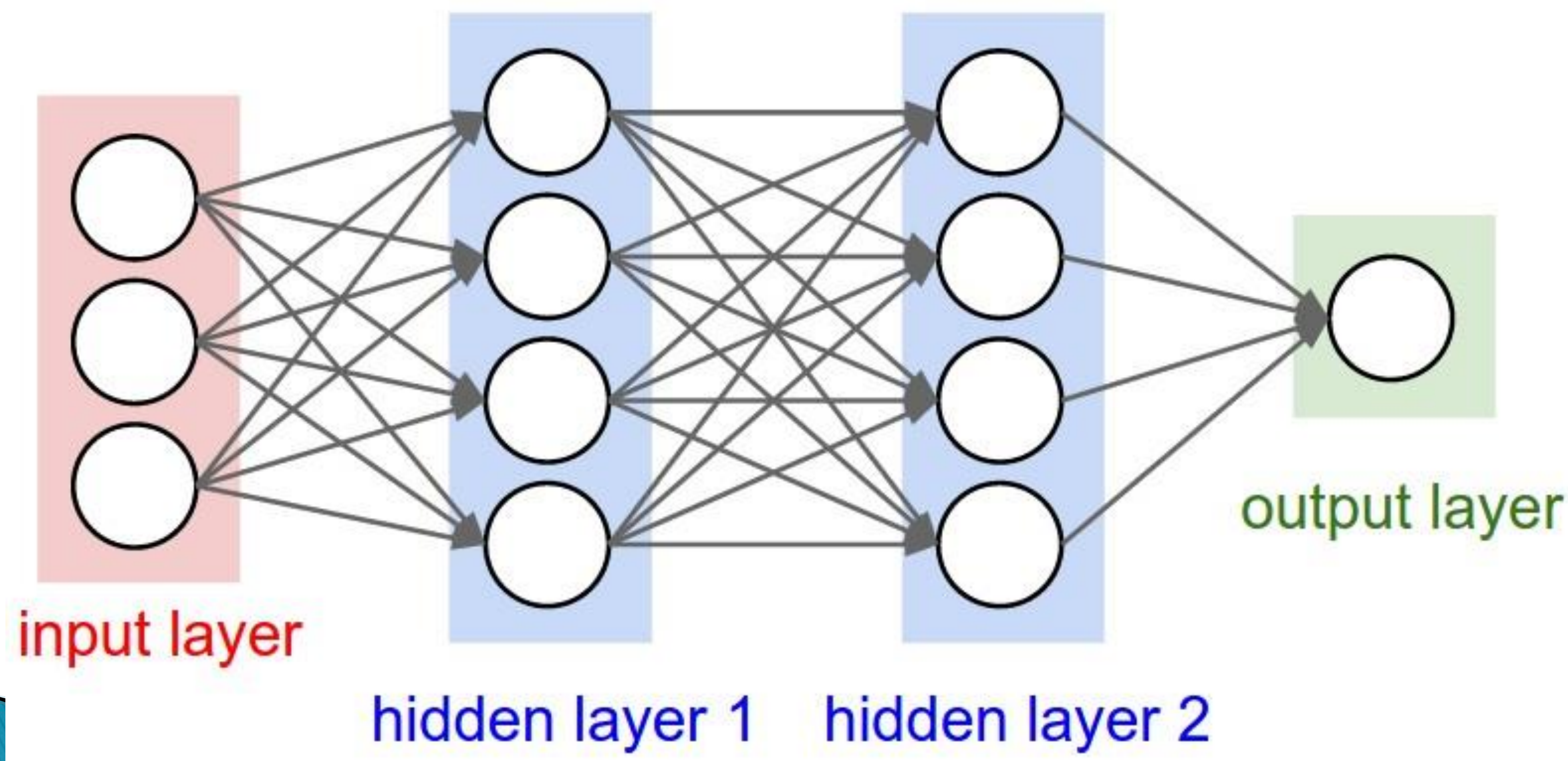
  - If classified incorrectly, update weights:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - y'\right)\mathbf{x}$$

(α is a positive *learning rate* that decays over time)

Multi-layer perceptrons

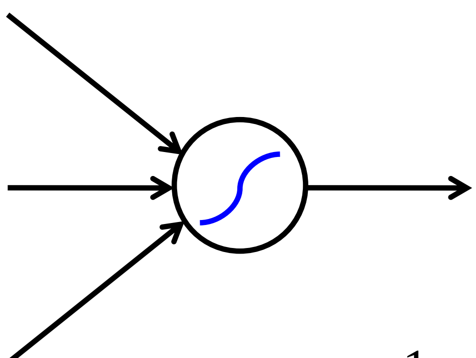
- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
  - This requires each perceptron to have a nonlinearity



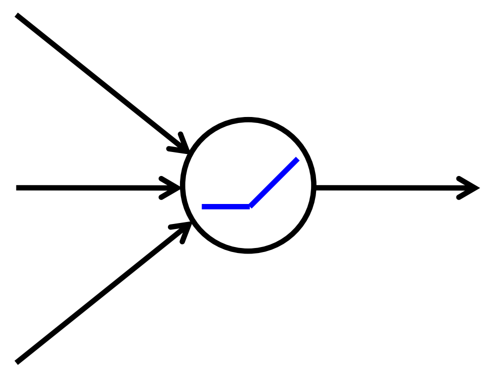
Slide credit: Svetlana Lazebnik

Multi-layer perceptrons

- To make nonlinear classifiers out of perceptrons, build a multi-layer neural network!
  - This requires each perceptron to have a nonlinearity
  - To be trainable, the nonlinearity should be differentiable



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



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

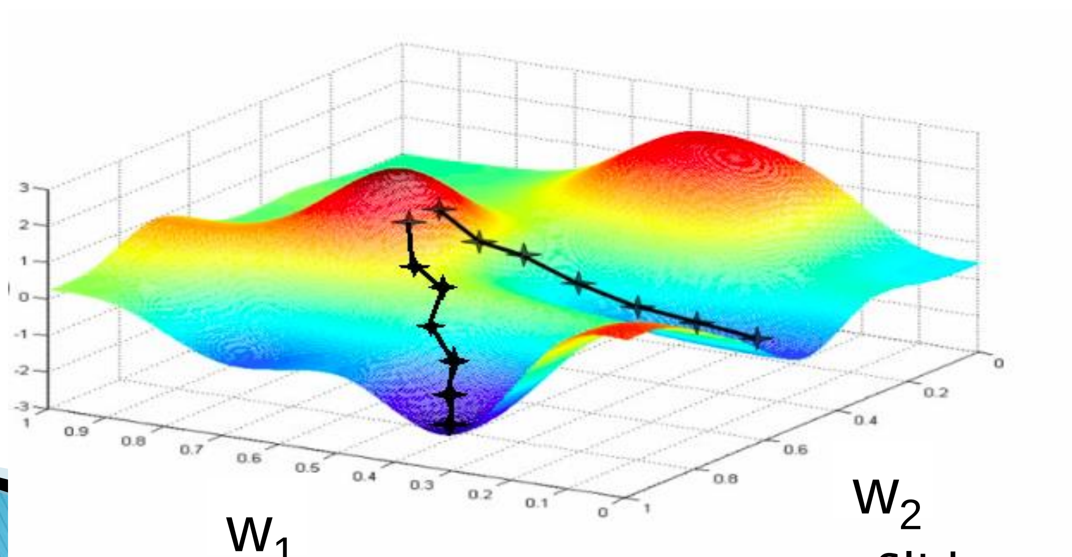
Slide credit: Svetlana Lazebnik

Training of multi-layer networks

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

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$



Slide credit: Svetlana Lazebnik

Training of multi-layer networks

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

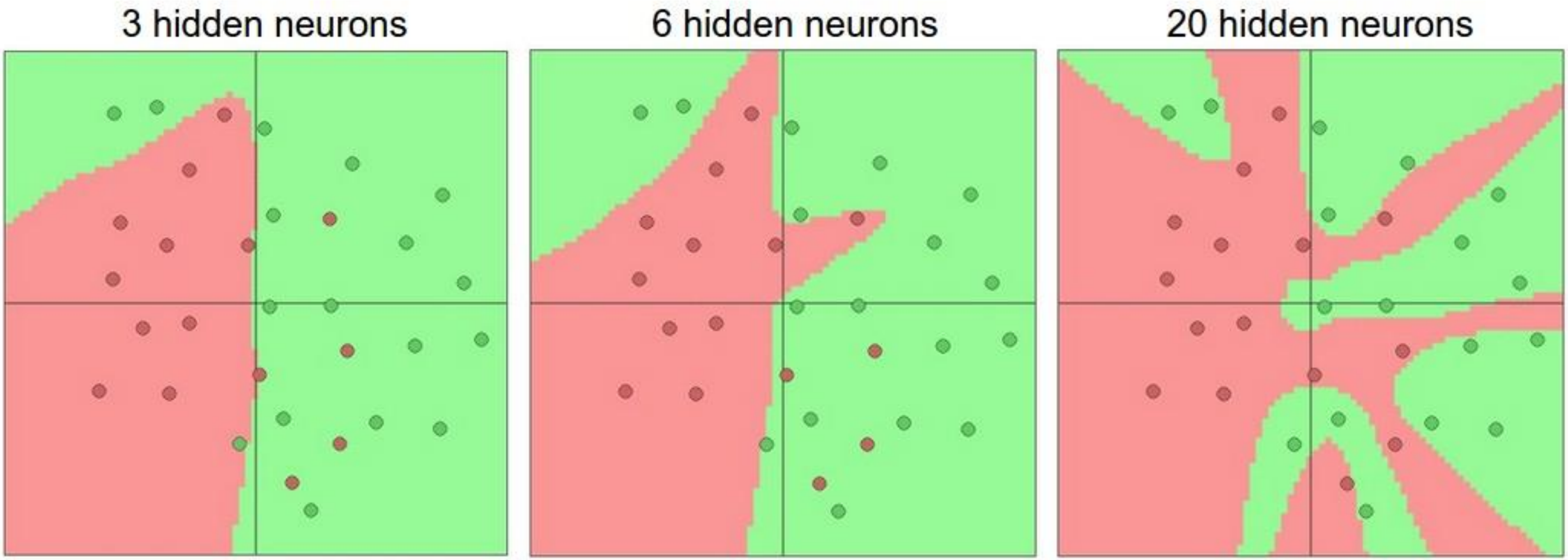
$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$
- Back-propagation: gradients are computed in the direction from output to input layers and combined using chain rule
- Stochastic gradient descent: compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Slide credit: Svetlana Lazebnik

Network with a single hidden layer

- Hidden layer size and network capacity:



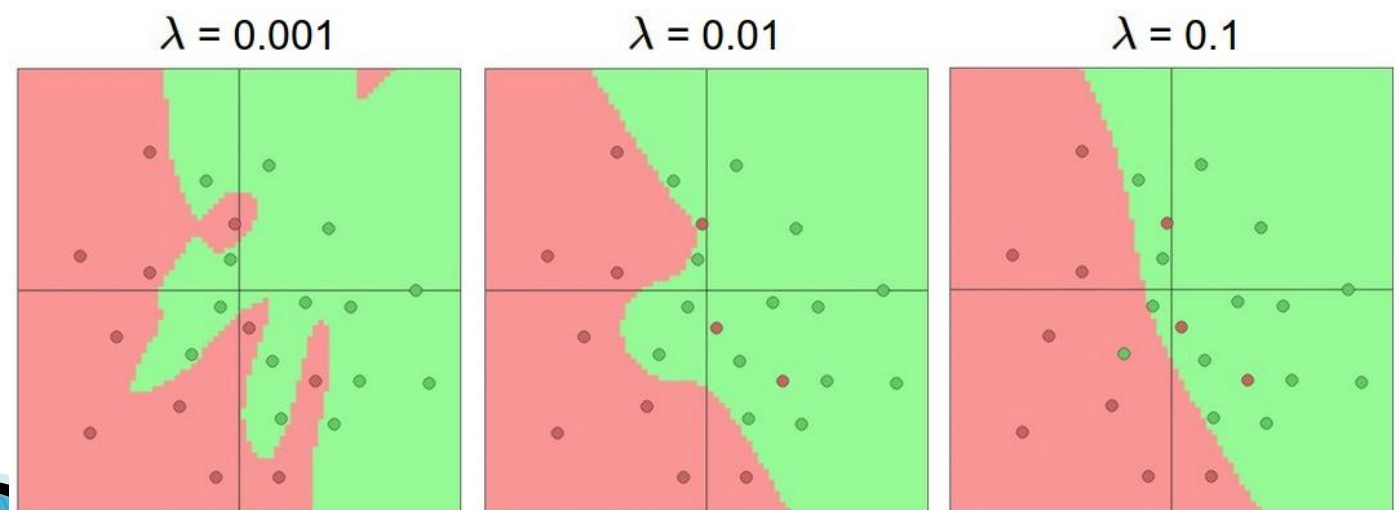
Source: http://cs231n.github.io/neural-networks-1/

Regularization

- It is common to add a penalty on weight magnitudes to the objective function:

$$E(f) = \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i)\right)^2 + \frac{\lambda}{2} \sum_j w_j^2$$

  - This encourages network to use all of its inputs “a little” rather than a few inputs “a lot”



Source: http://cs231n.github.io/neural-networks-1/

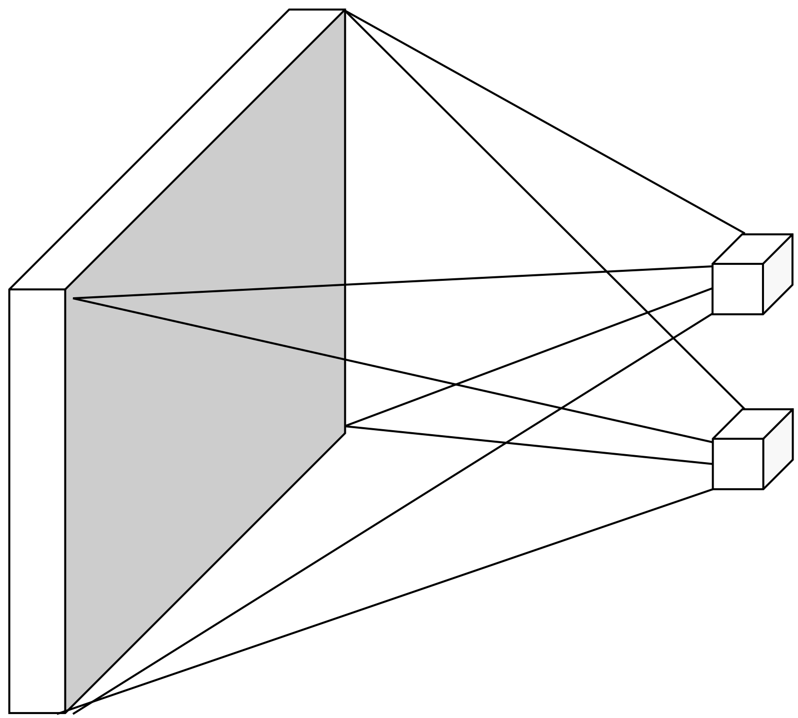
Neural networks: Pros and cons

- Pros
  - Flexible and general function approximation framework
  - Can build extremely powerful models by adding more layers
- Cons
  - Hard to analyze theoretically (e.g., training is prone to local optima)
  - Huge amount of training data, computing power may be required to get good performance
  - The space of implementation choices is huge (network architectures, parameters)

Slide credit: Svetlana Lazebnik

Convolutional Neural Network

Neural networks for images

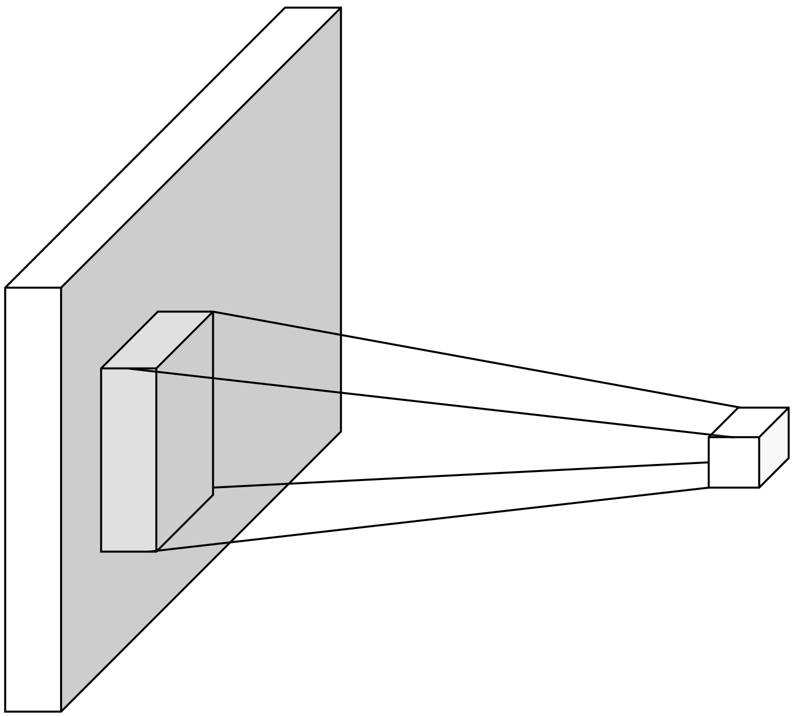


image

Fully connected layer

Slide credit: Svetlana Lazebnik

Neural networks for images

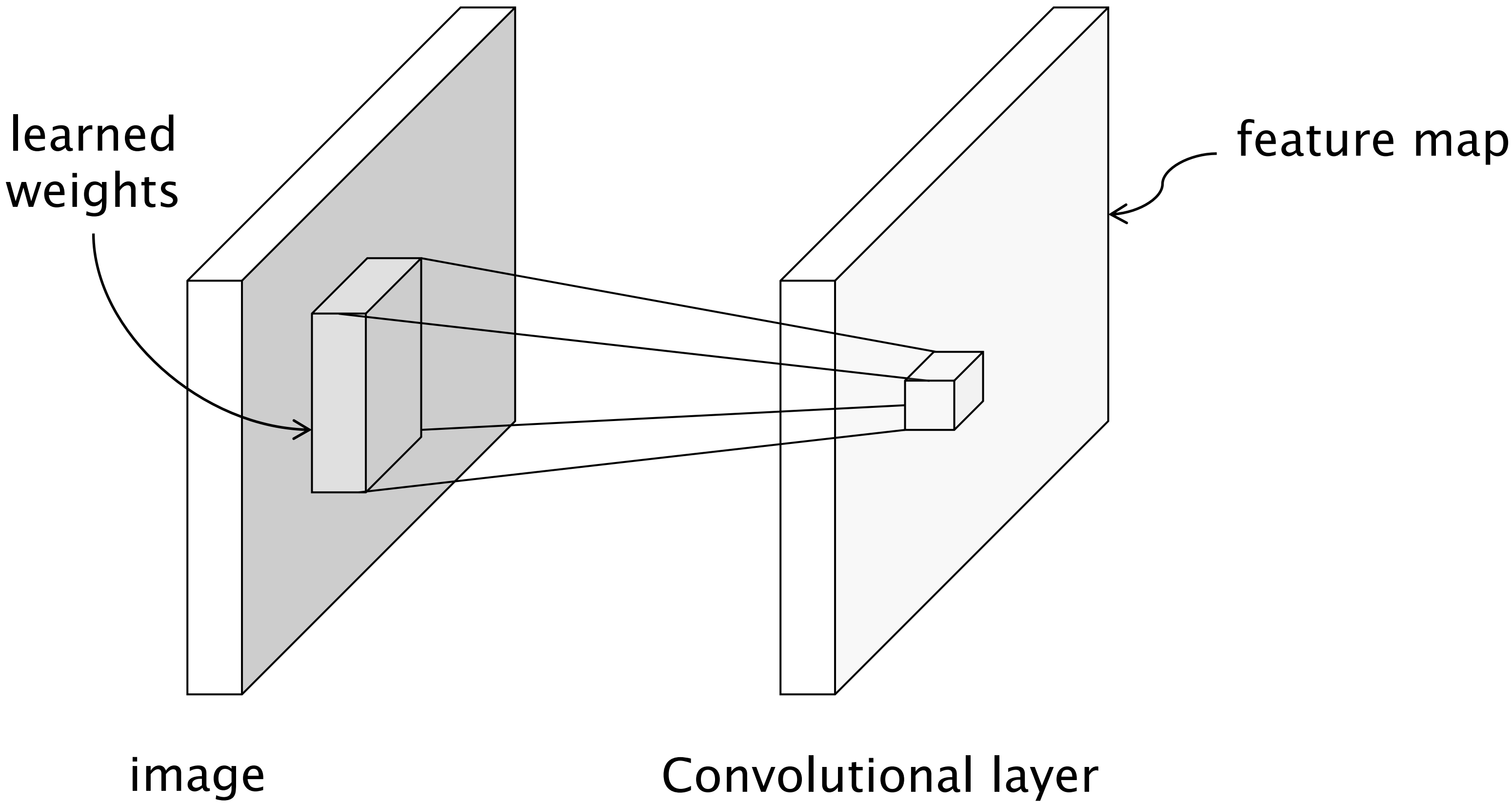


image

Convolutional layer

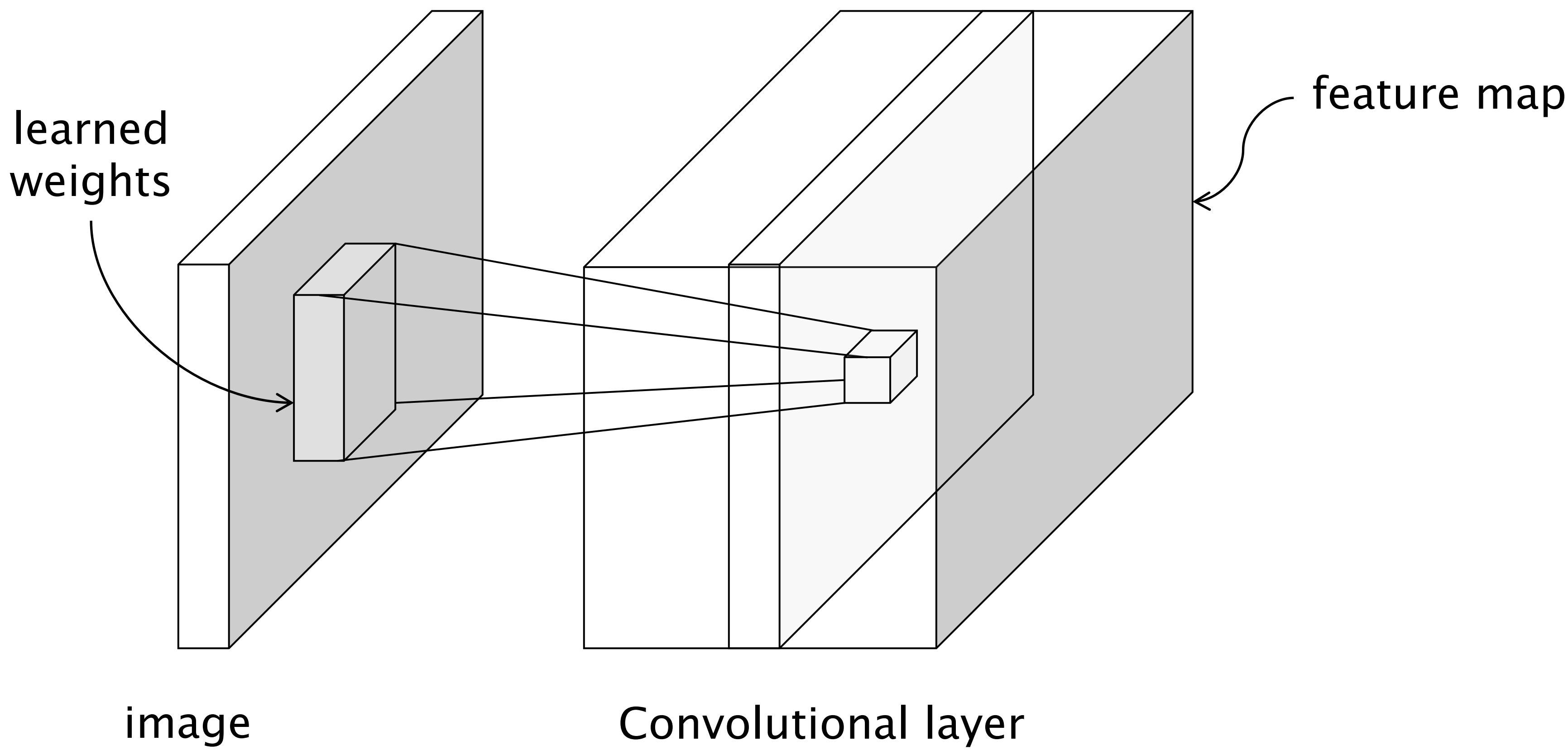
Slide credit: Svetlana Lazebnik

Neural networks for images



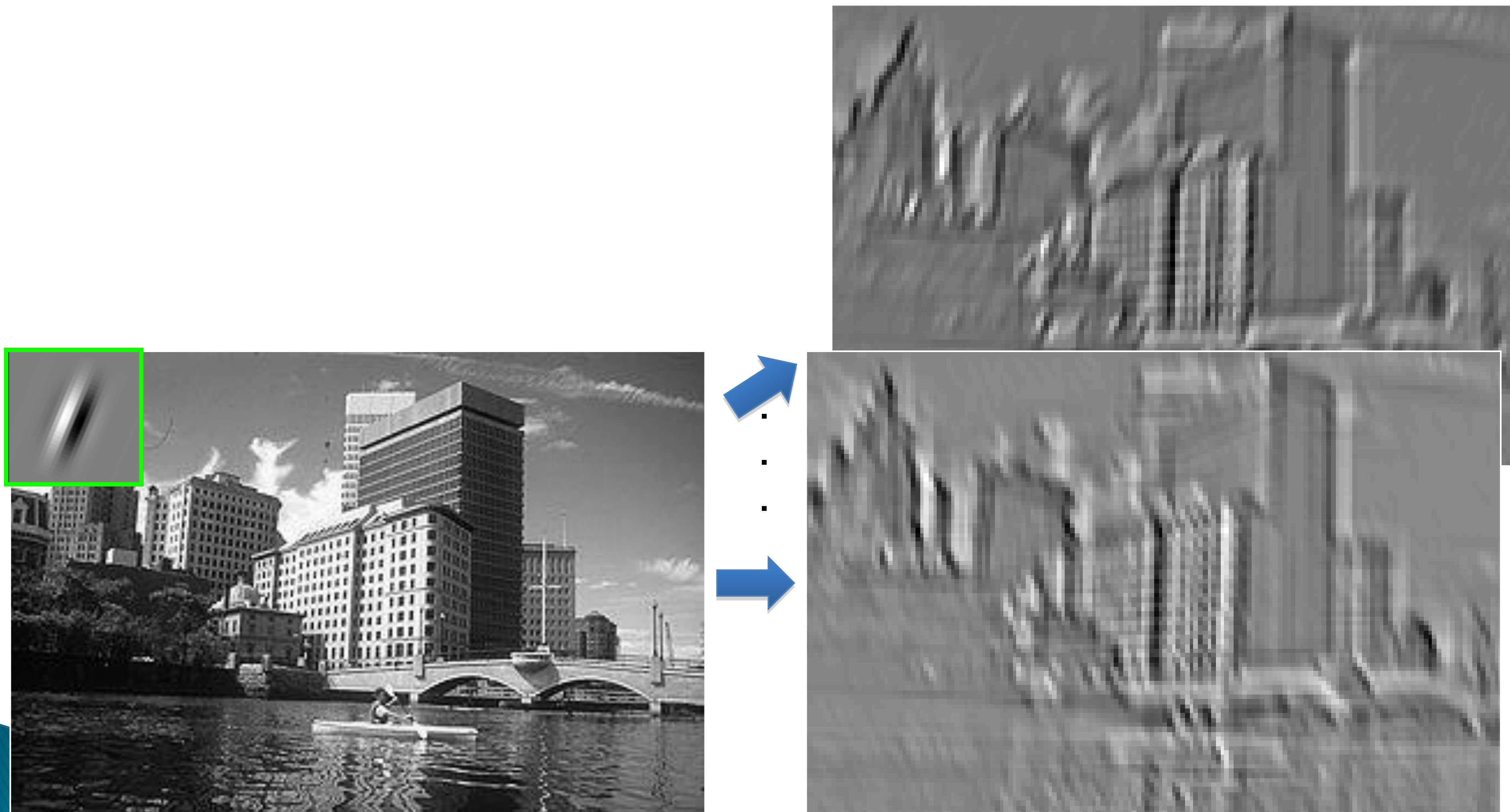
Slide credit: Svetlana Lazebnik

Neural networks for images



Slide credit: Svetlana Lazebnik

Convolution as feature extraction

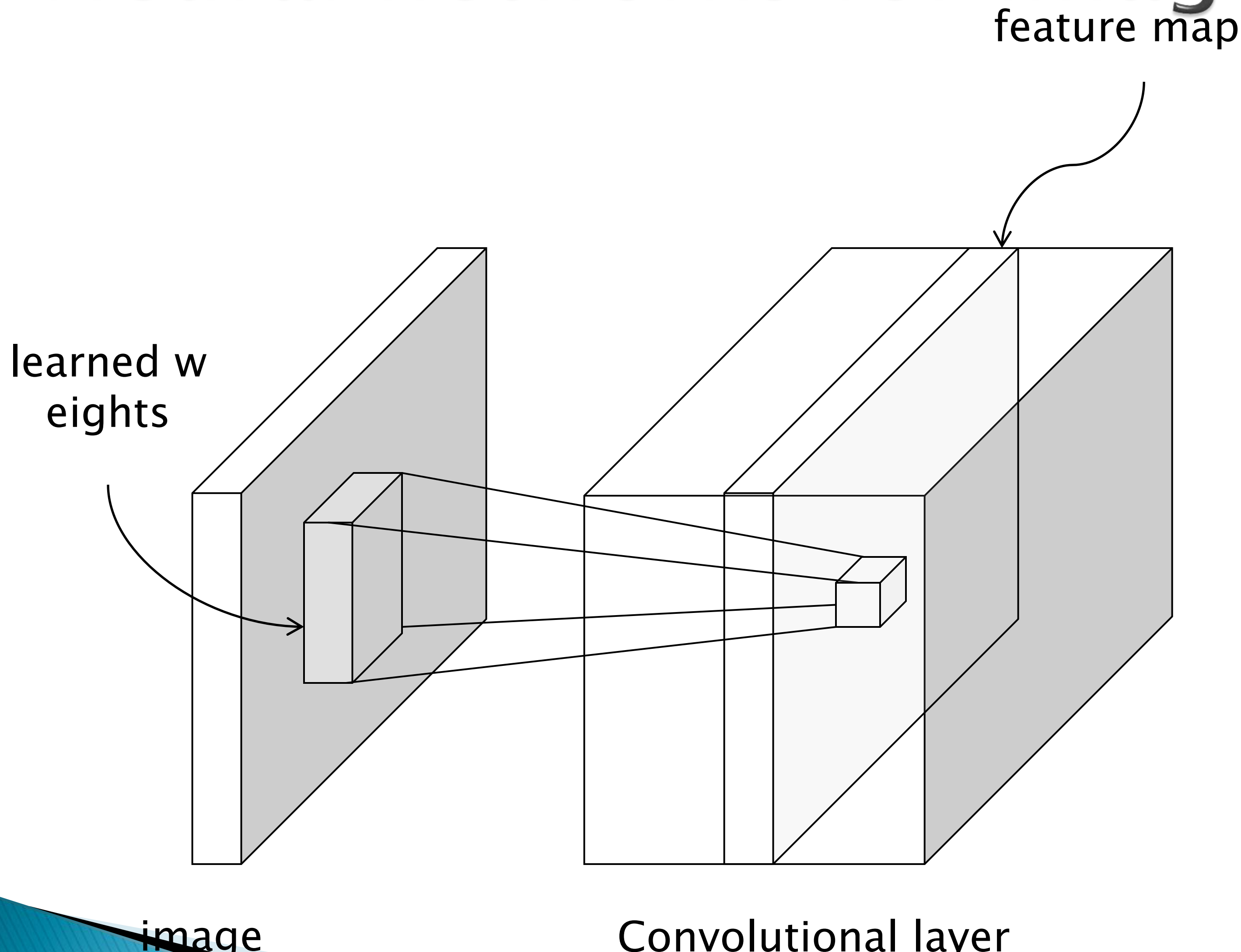


Input

Feature Map

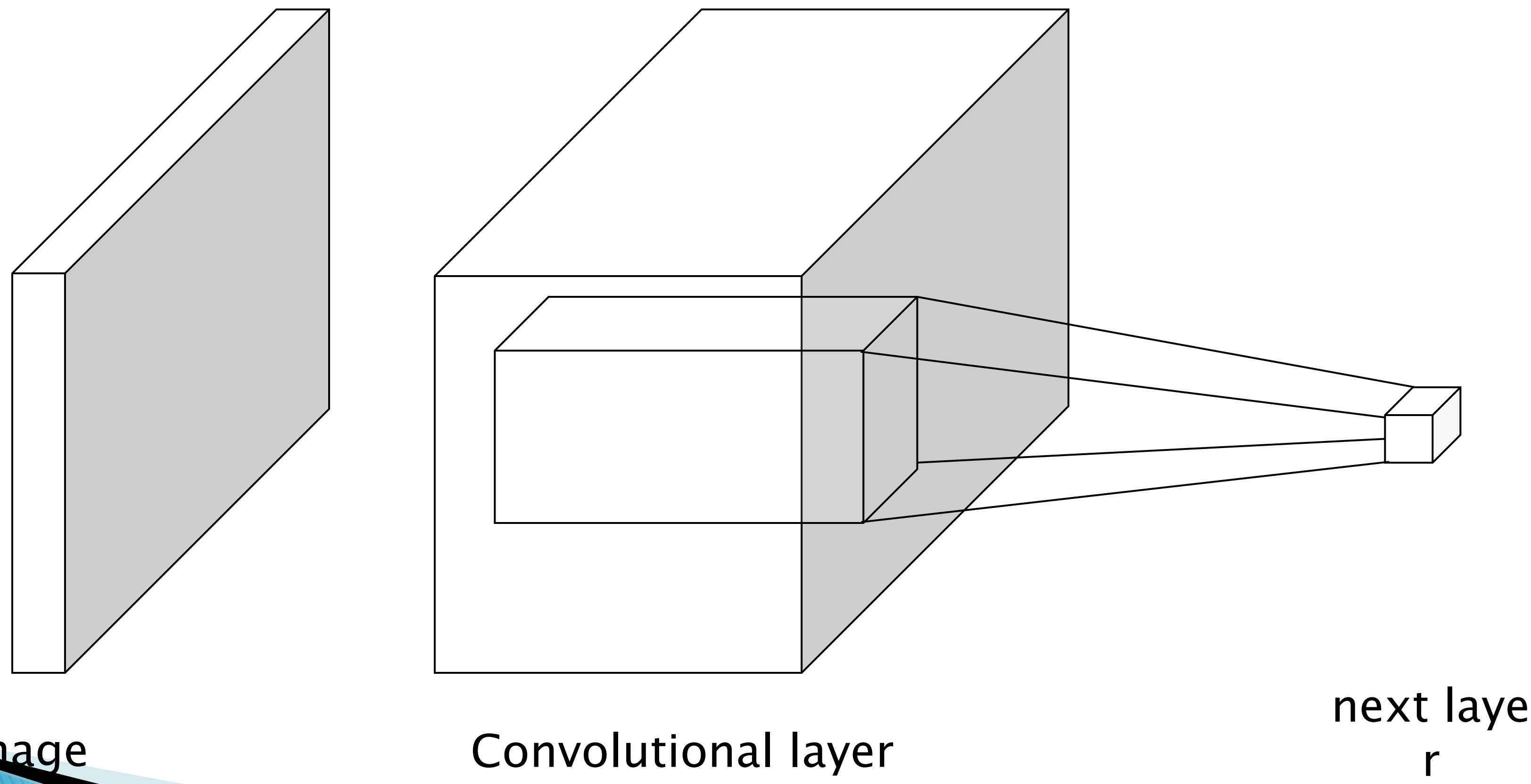
Slide credit: Svetlana Lazebnik

Neural networks for images



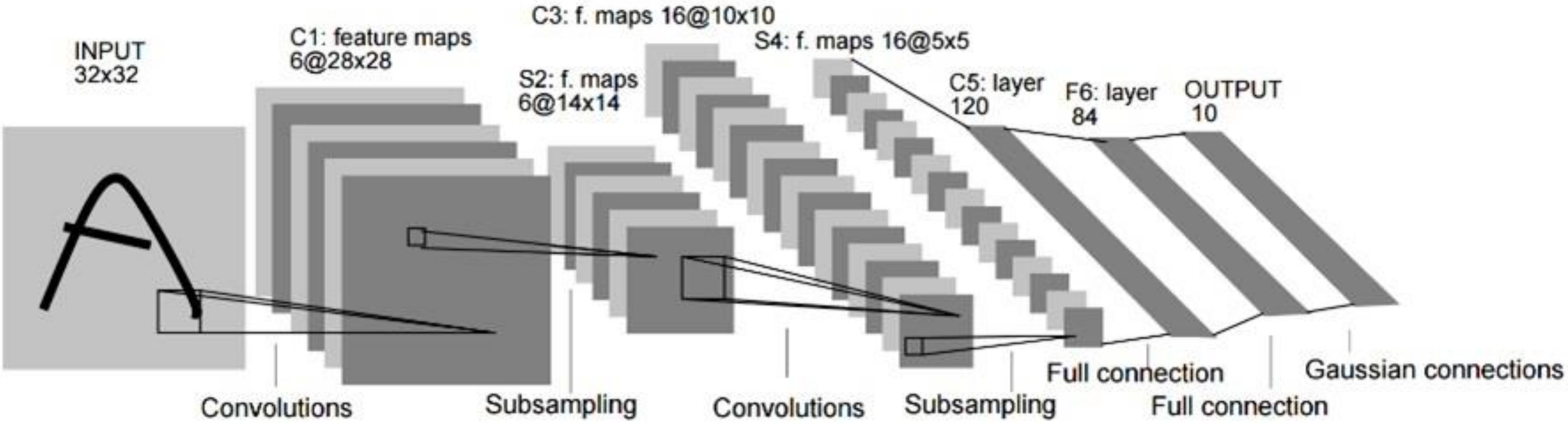
Slide credit: Svetlana Lazebnik

Neural networks for images



Slide credit: Svetlana Lazebnik

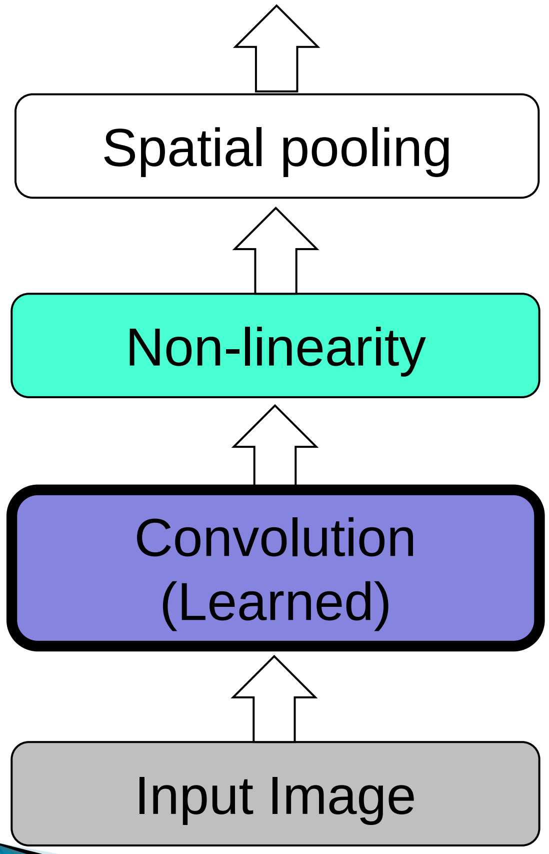
LeNet [LeCun et al. 1998]

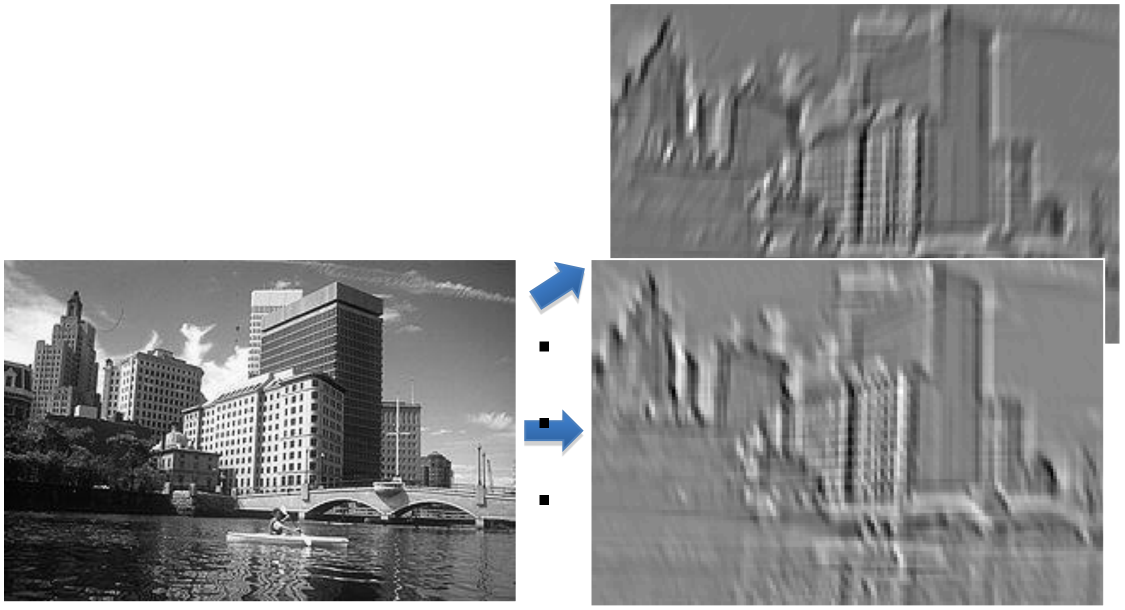


LeNet-1 from 1993

Gradient-based learning applied to document recognition [LeCun, Bottou, Bengio, Haffner 1998]

Key operations in a CNN

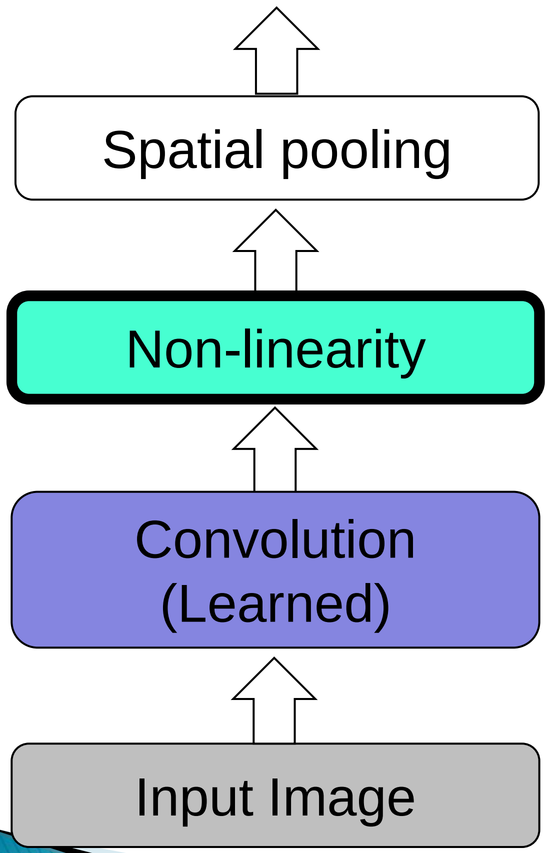




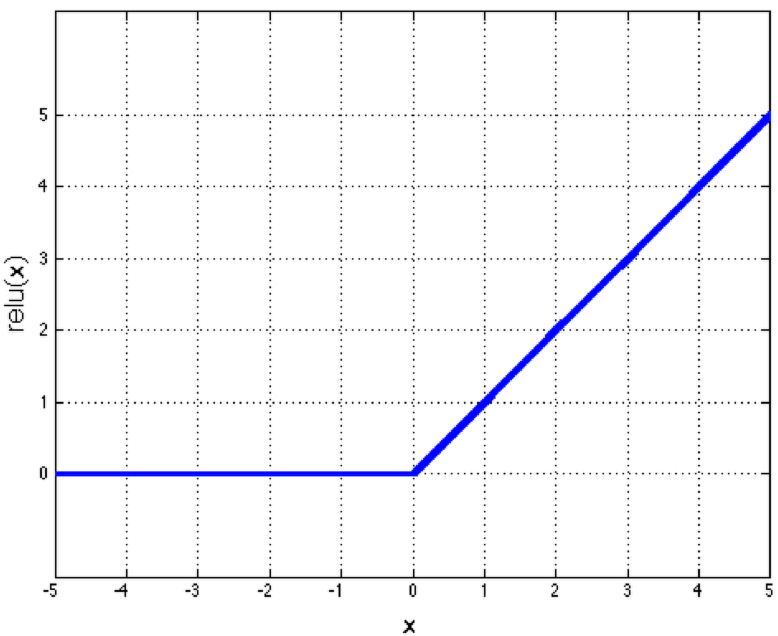
Input Feature Map

Source: R. Fergus, Y. LeCun

Key operations in a CNN

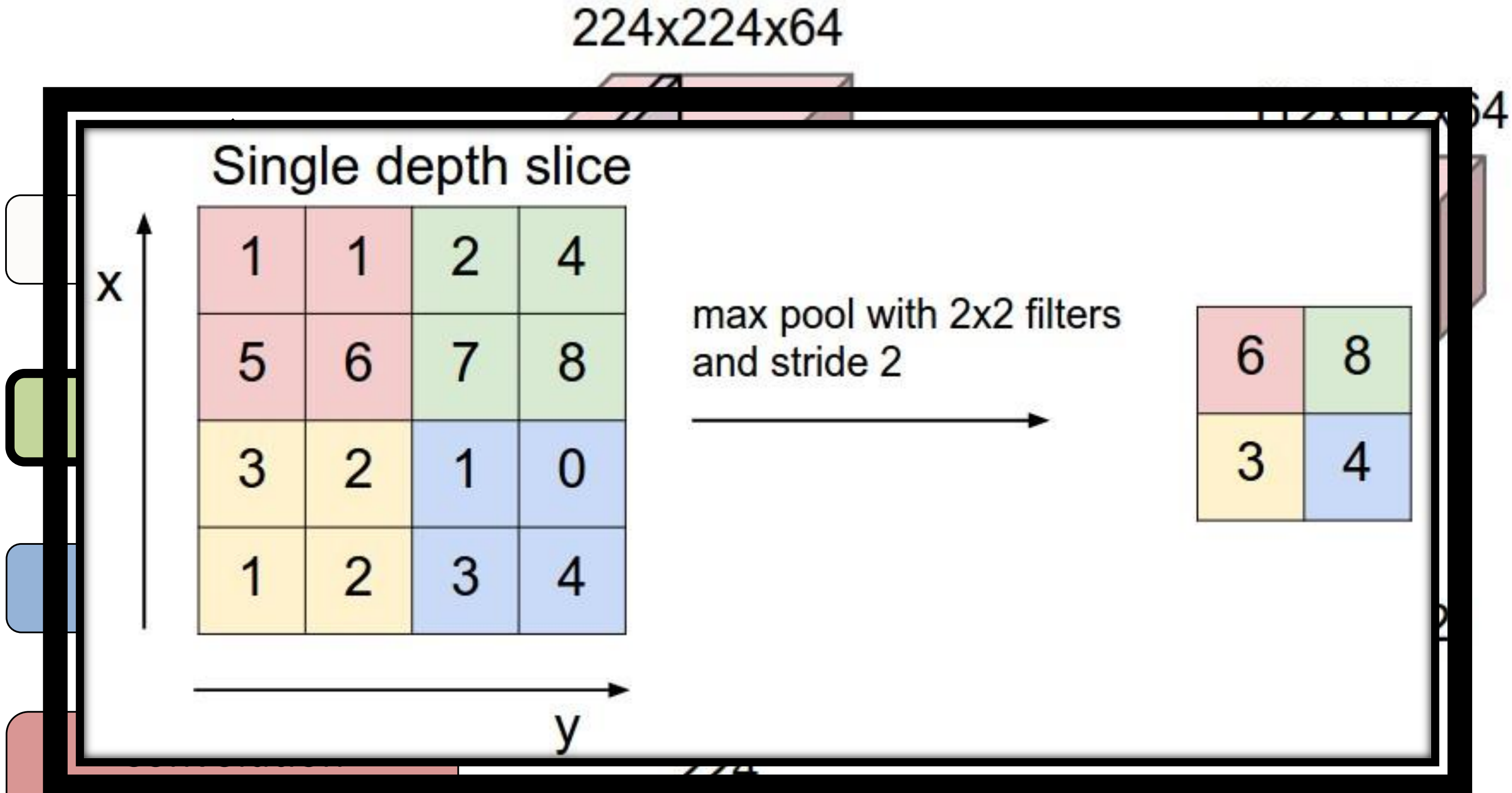


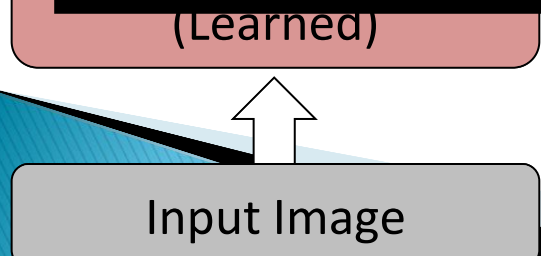
Rectified Linear Unit (ReLU)



Source: R. Fergus, Y. LeCun

Key operations in a CNN



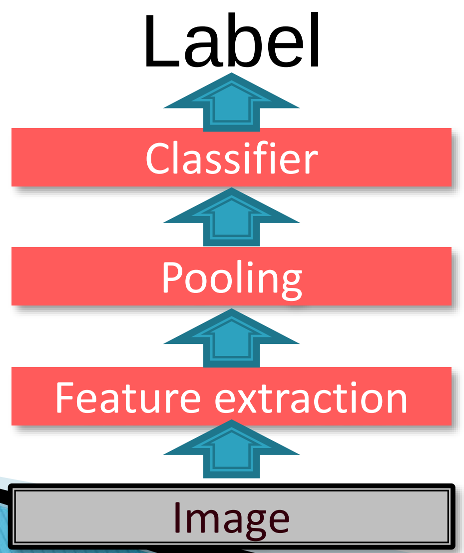


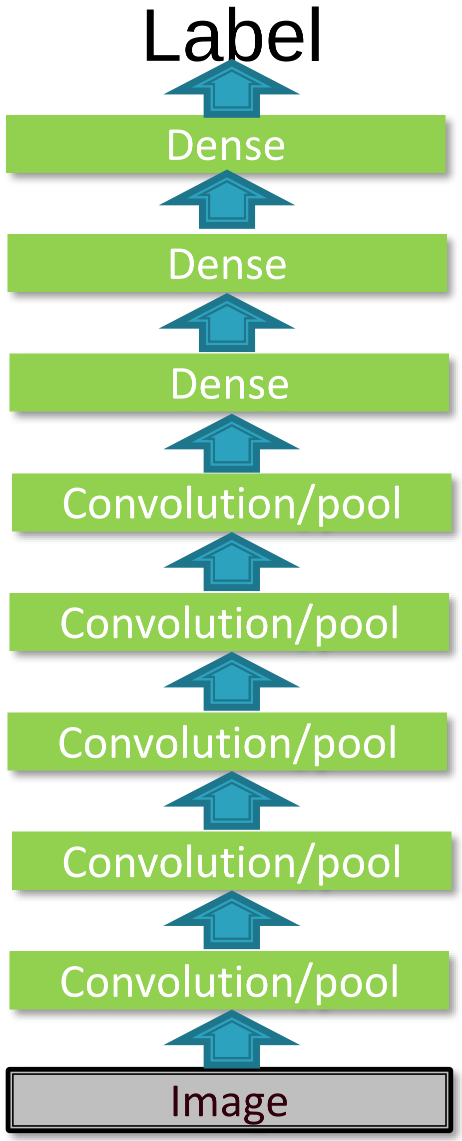
Provide *translation invariance*

slide credit: S. Lazebnik

Engineered vs. learned features

Convolutional filters are trained in a supervised manner by back-propagating classification error



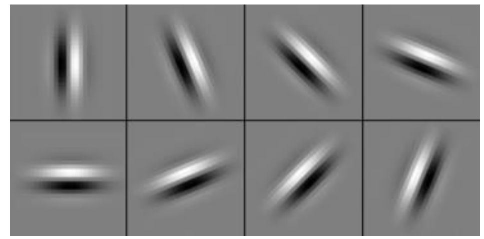


SIFT Descriptor

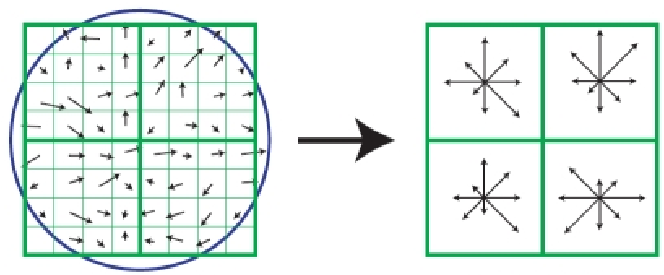
Lowe [IJCV 2004]

Image Pixels

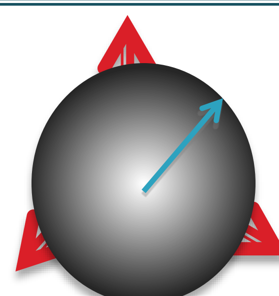
Apply oriented filters



Spatial pool (Sum)



Normalize to unit length



Feature Vector

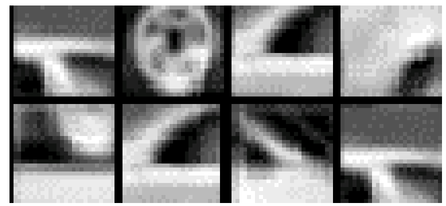
slide credit: R. Fergus

Spatial Pyramid Matching

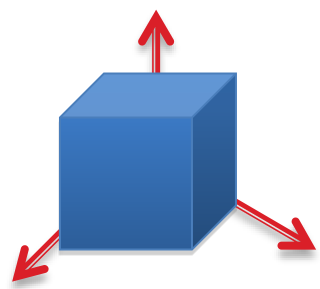
Lazebnik, Schmid, Ponce [CVPR 2006]

SIFT Features

Filter with Visual Words



Take max VW response (L-inf normalization)



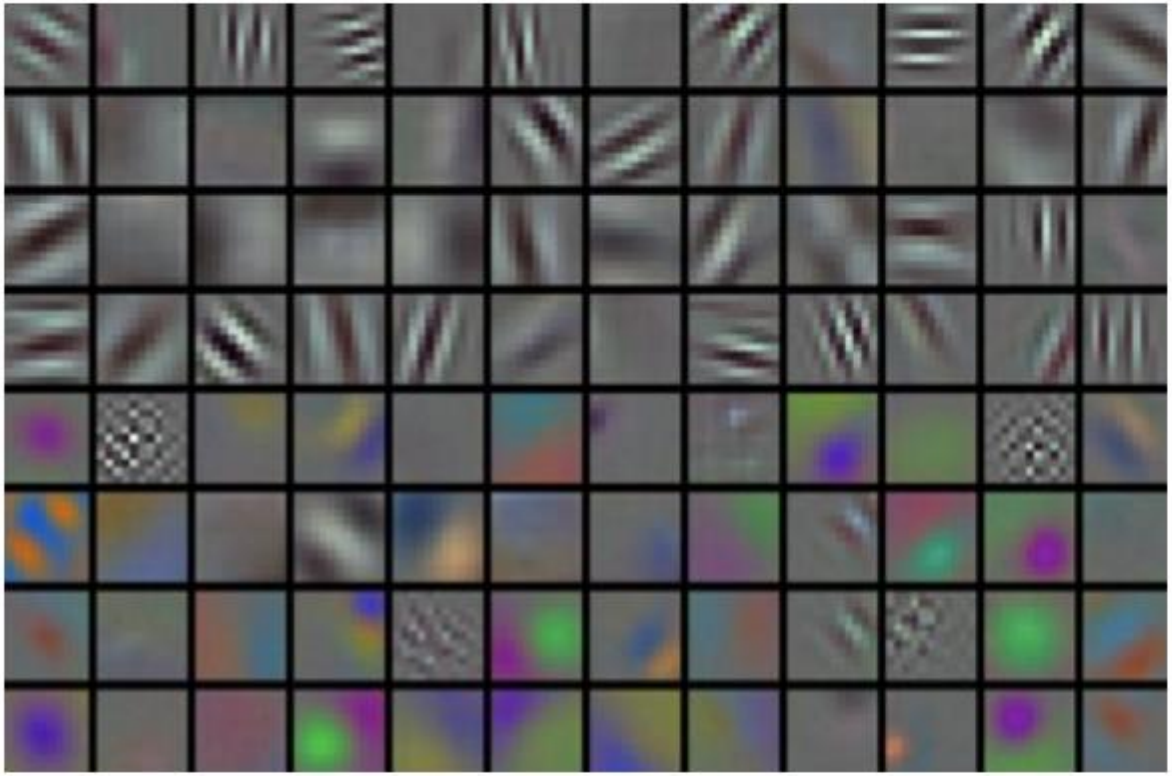
Multi-scale spatial pool (Sum)

Classifier

slide credit: R. Fergus

Visualizing what was learned

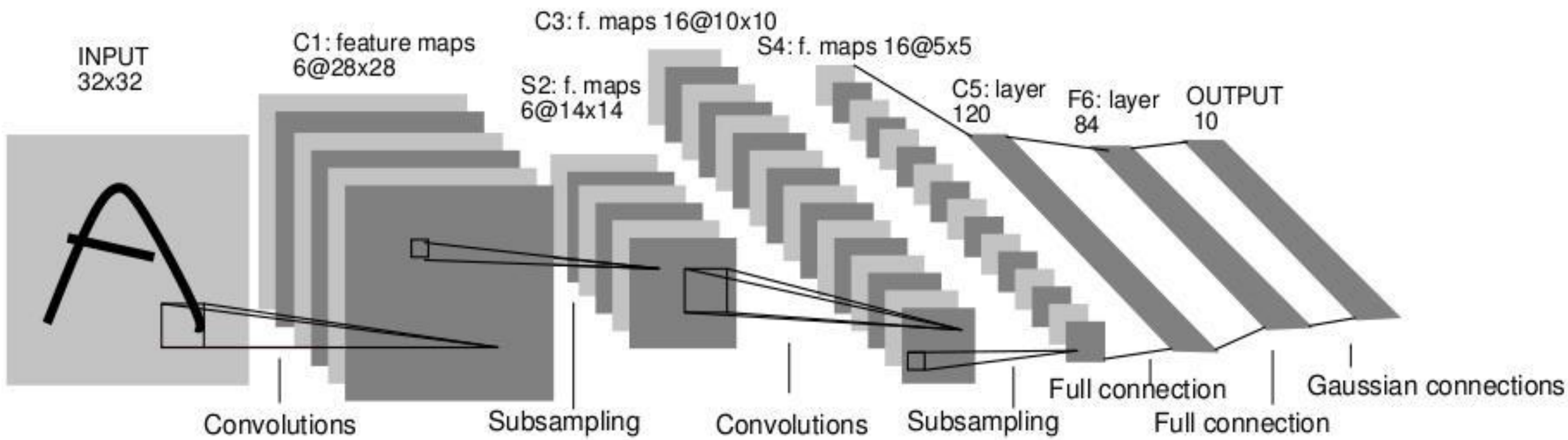
- What do the learned filters look like?



Typical first layer filters

Slide credit: Kristen Grauman

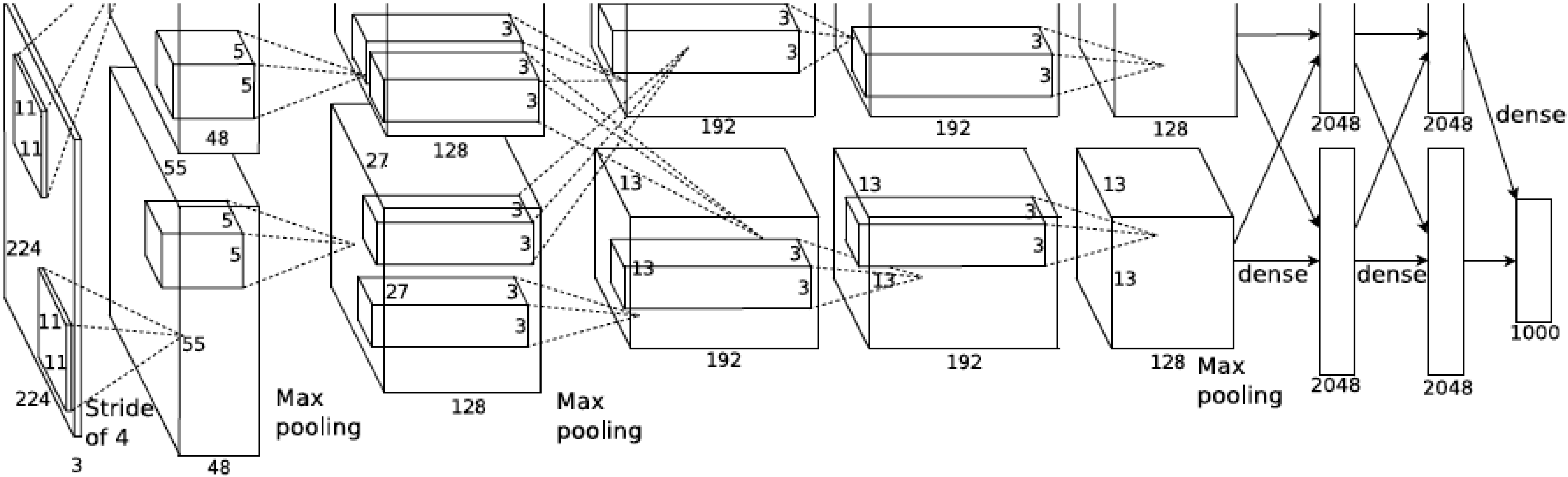
History of CNNs: LeNet-5



- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training exa mples

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to docu ment recognition, Proc. IEEE 86(11): 2278–2324, 1998.

AlexNet: ILSVRC 2012 winner



- Similar framework to LeNet but:
  - Max pooling, ReLU nonlinearity
  - More data and bigger model (7 hidden layers, 650K units, 60M params)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012