CS 2750: Machine Learning

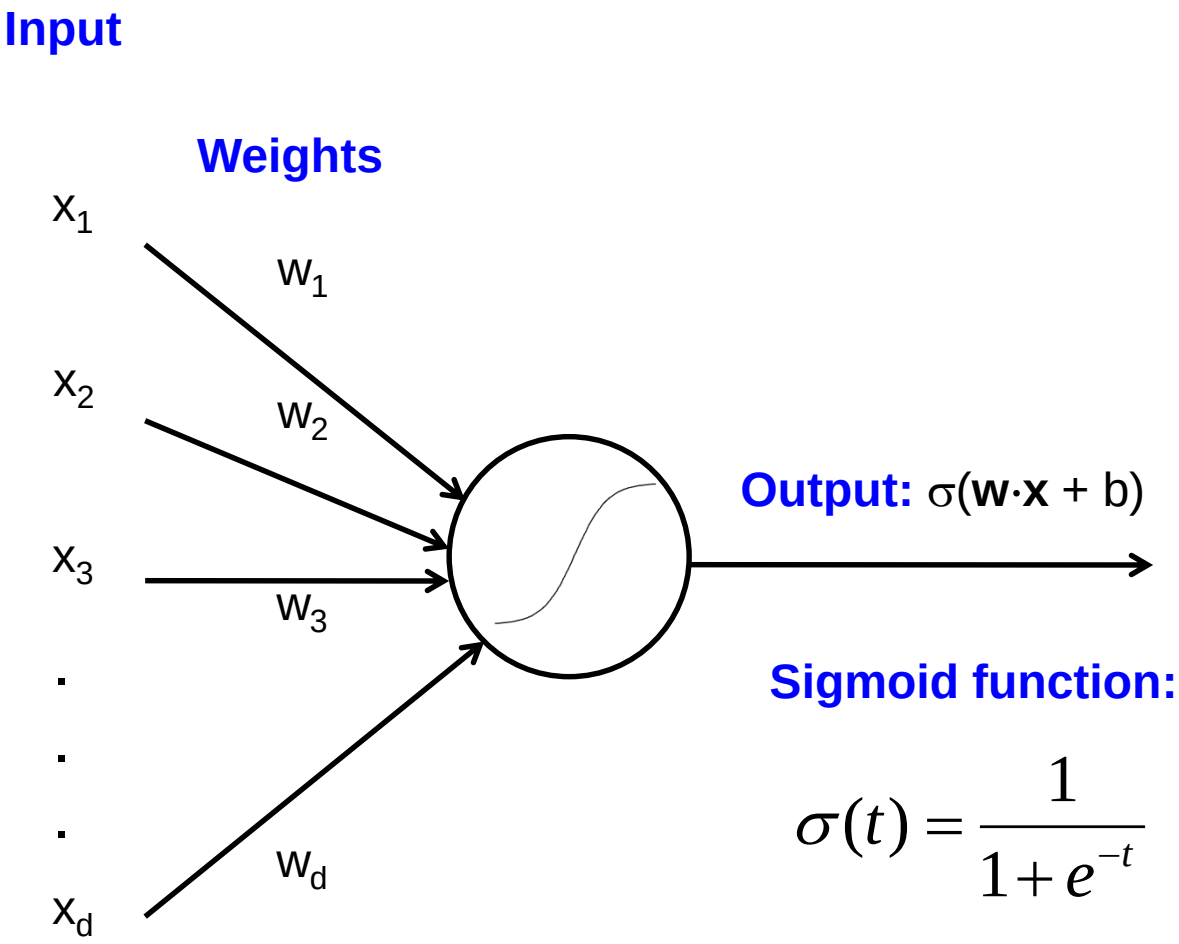
Neural Networks

Prof. Adriana Kovashka
University of Pittsburgh
April 13, 2016

Plan for today

- Neural network definition and examples
- Training neural networks (backprop)
- Convolutional neural networks
  - Architecture
  - Visualization
  - Analyzing and debugging

Background: Perceptrons



Slide credit: Lana Lazebnik

Background: Linear functions

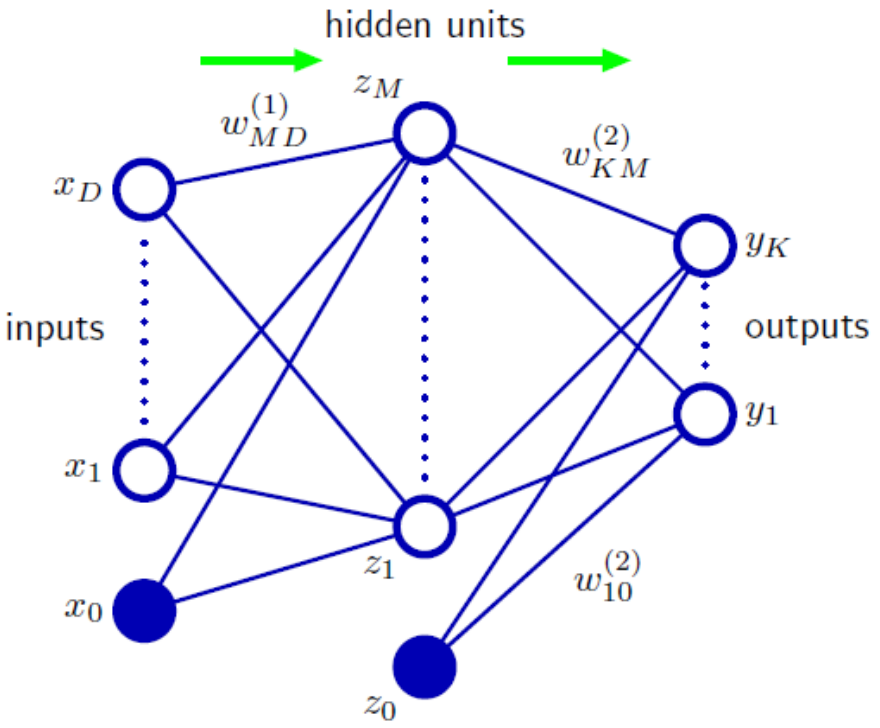
- Before: Linear function of basis functions

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

- How about *learning* as opposed to hand-coding the basis functions?

Neural network definition

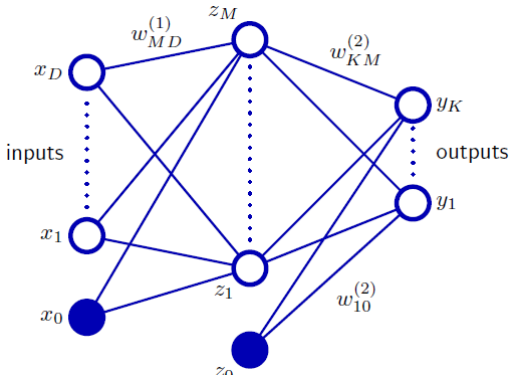
Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation.



- Activations: $$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

- Nonlinear activation function *h* (e.g. sigmoid, tanh): $$z_j = h(a_j) \qquad \tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Neural network definition



- Layer 2

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

- Layer 3 (final)

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

- Outputs

(binary) $y_k = \sigma(a_k) = \dfrac{1}{1 + \exp(-a_k)}$

(multiclass) $y_k = \dfrac{\exp(a_k)}{\sum_j \exp(a_j)}$

- Finally:

(binary)

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

Neural network definition

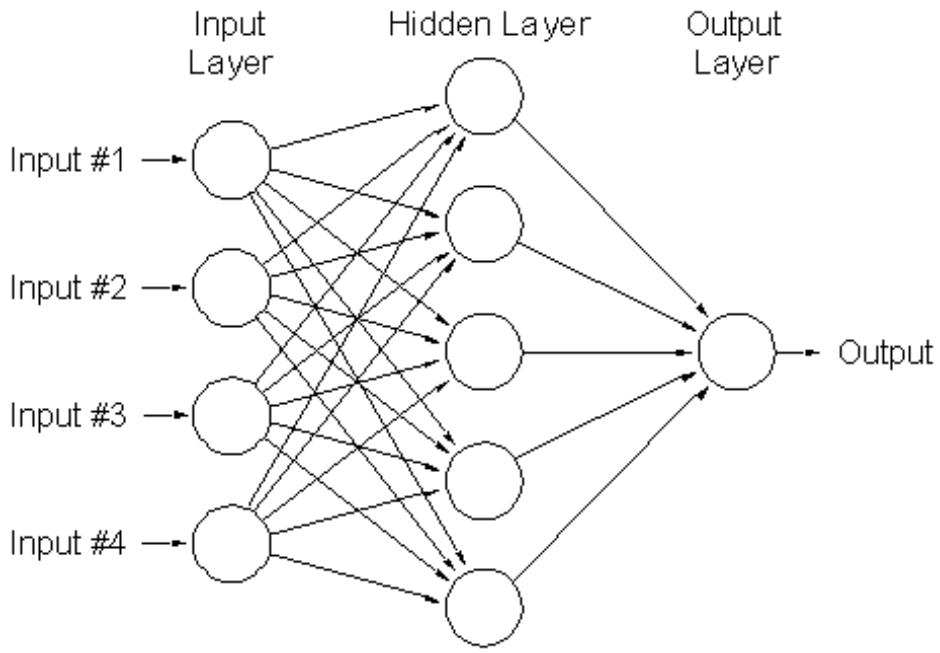
- Explicit biases:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- Absorb bias into weights:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

A multi-layer neural network

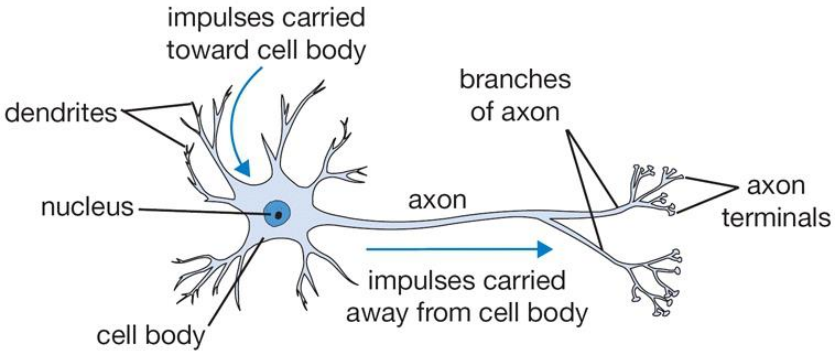


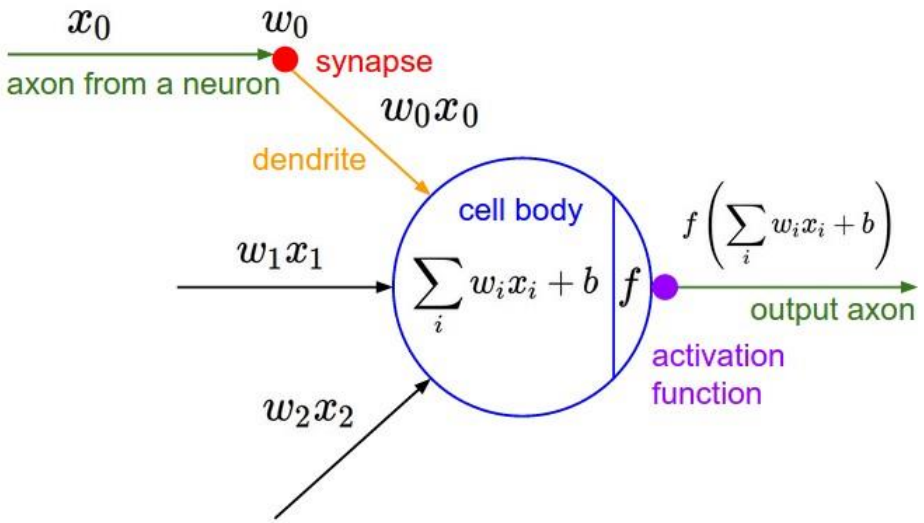
- *Nonlinear* classifier
- Can approximate any continuous function to arbitrary accuracy given sufficiently many hidden units

Slide credit: Lana Lazebnik

Inspiration: Neuron cells

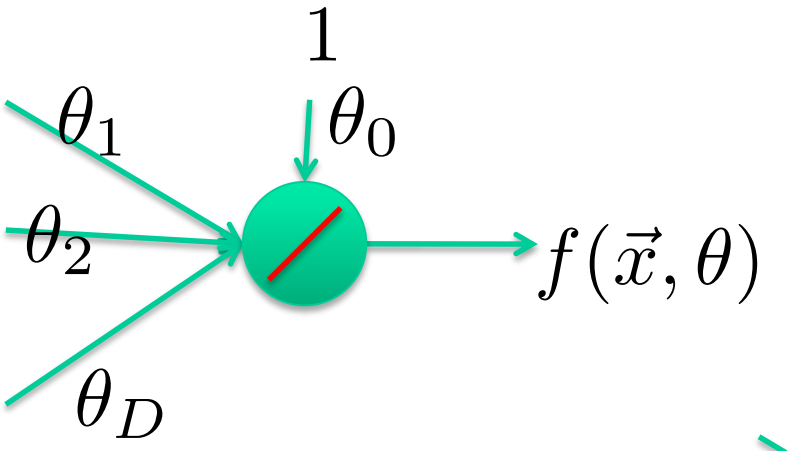
- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node
- If output of function over threshold, neuron “fires”



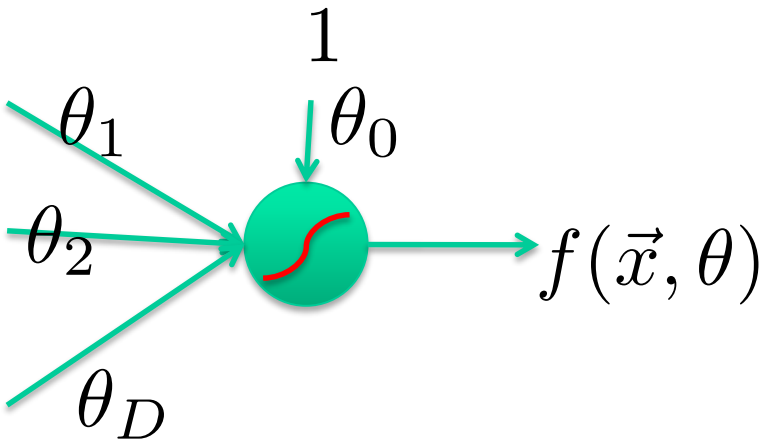


Text: HKUST, figures: Andrej Karpathy

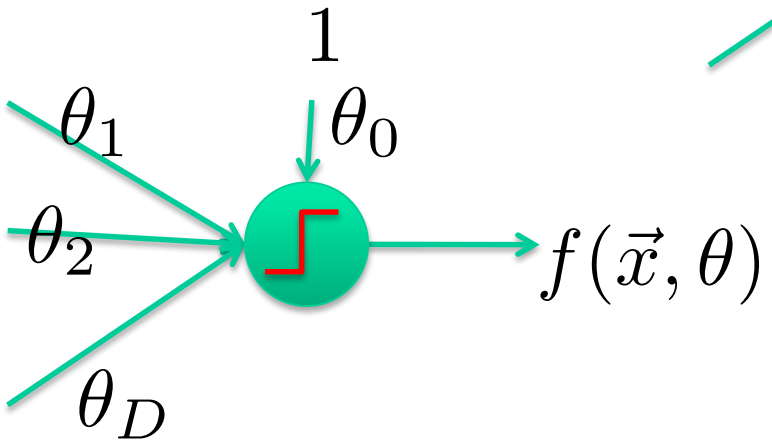
Types of neurons



Linear Neuron



Logistic Neuron

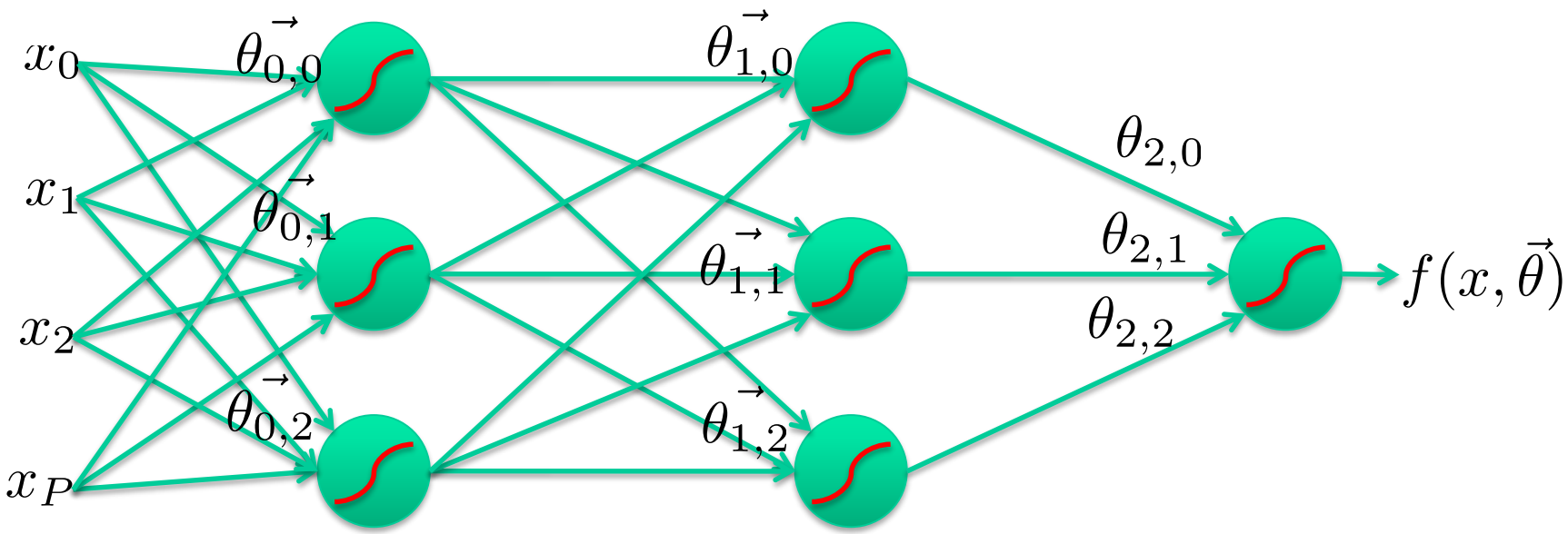


Perceptron

Slide credit: HKUST

Multilayer networks

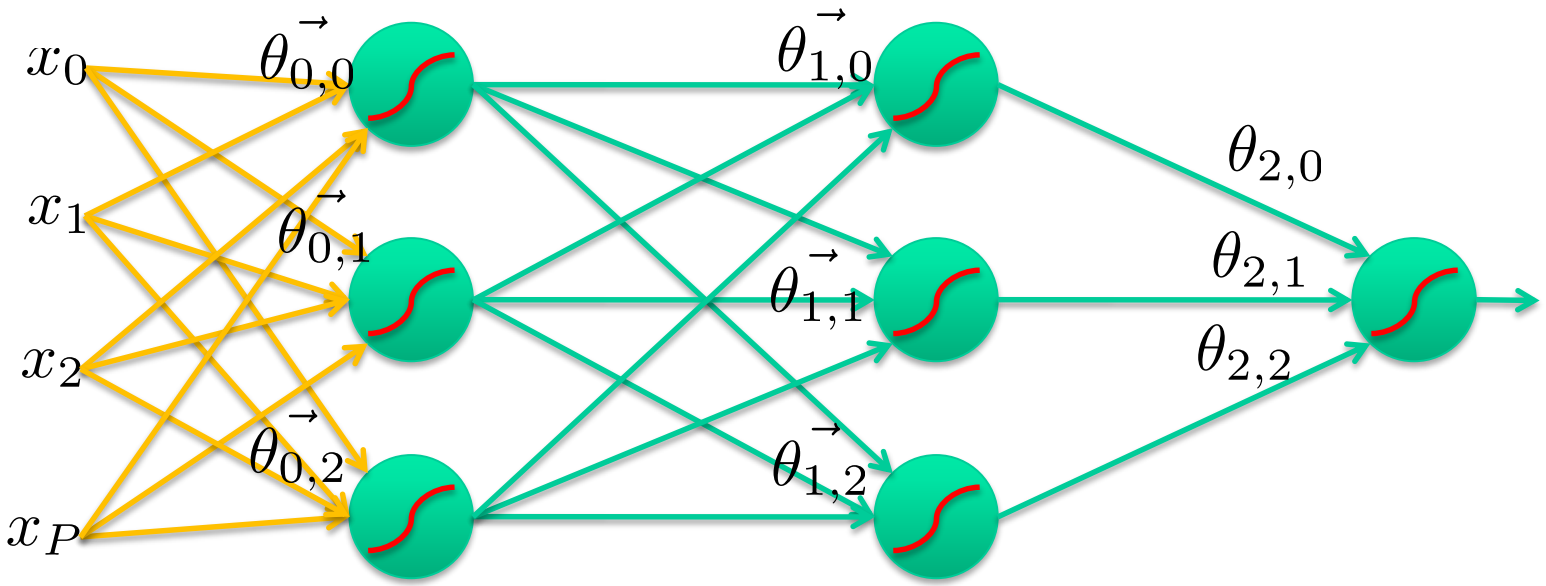
- Cascade neurons together
- Output from one layer is the input to the next
- Each layer has its own sets of weights



Slide credit: HKUST

Feed-forward networks

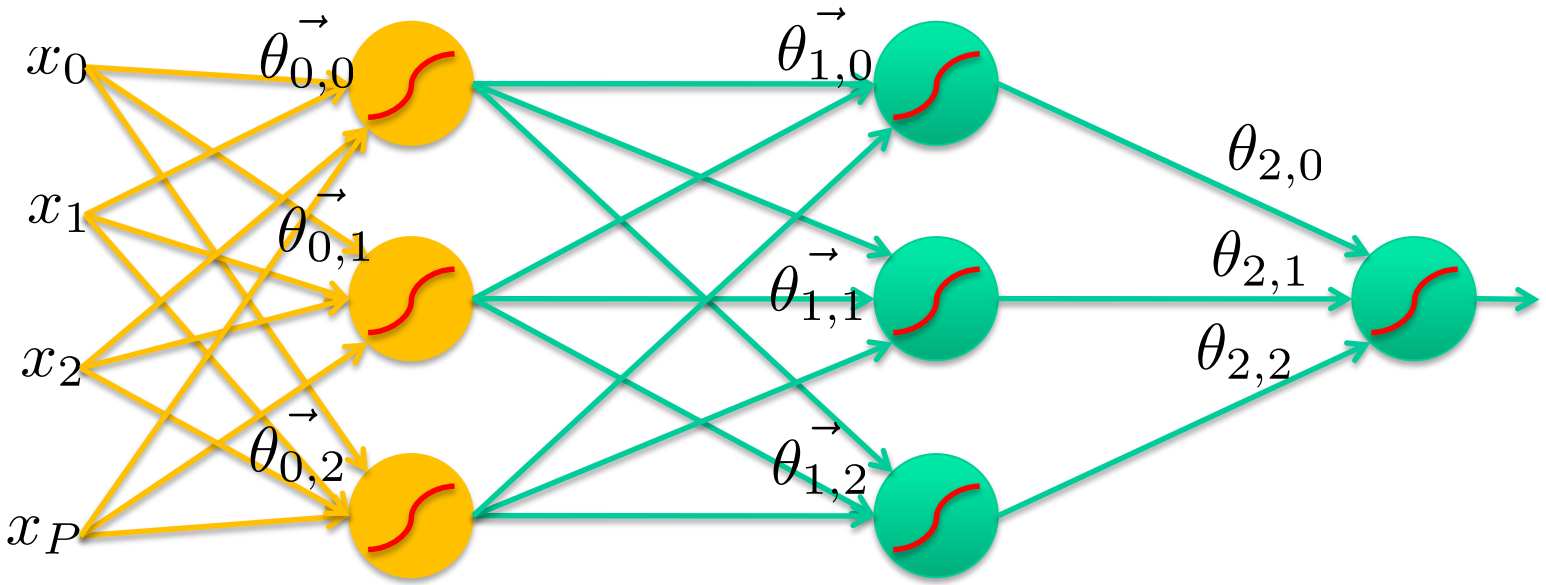
- Predictions are fed forward through the network to classify



Slide credit: HKUST

Feed-forward networks

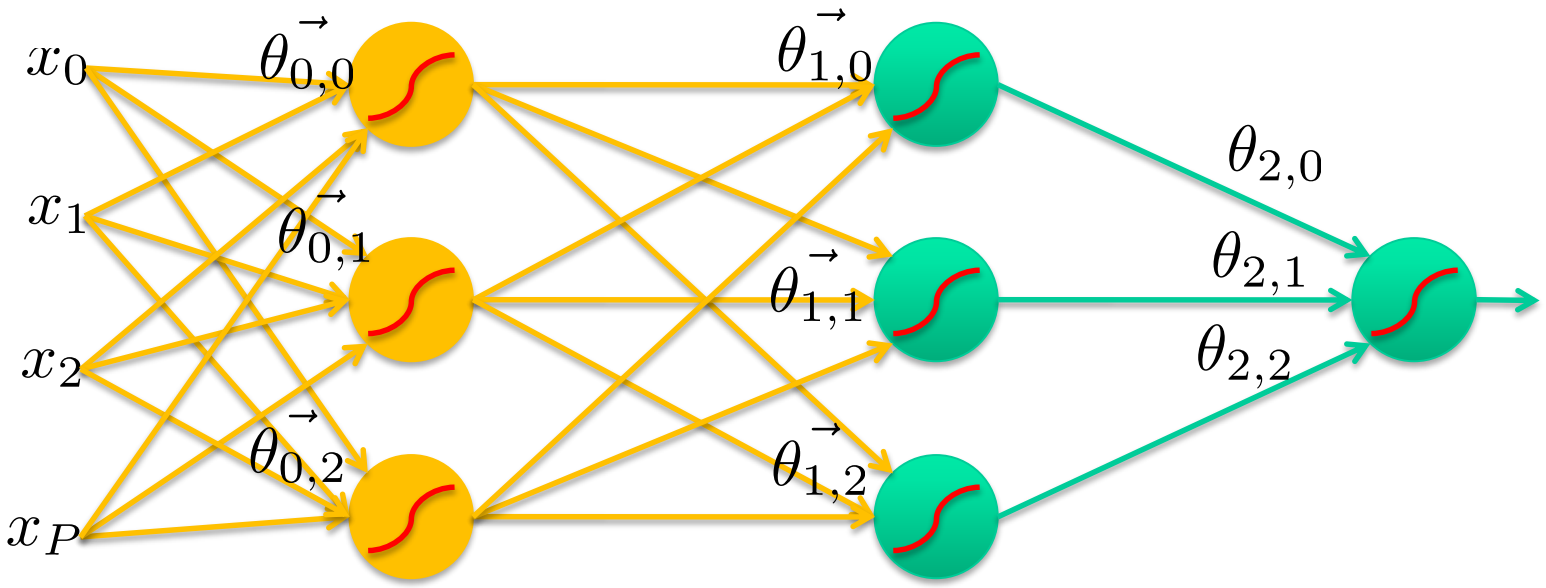
- Predictions are fed forward through the network to classify



Slide credit: HKUST

Feed-forward networks

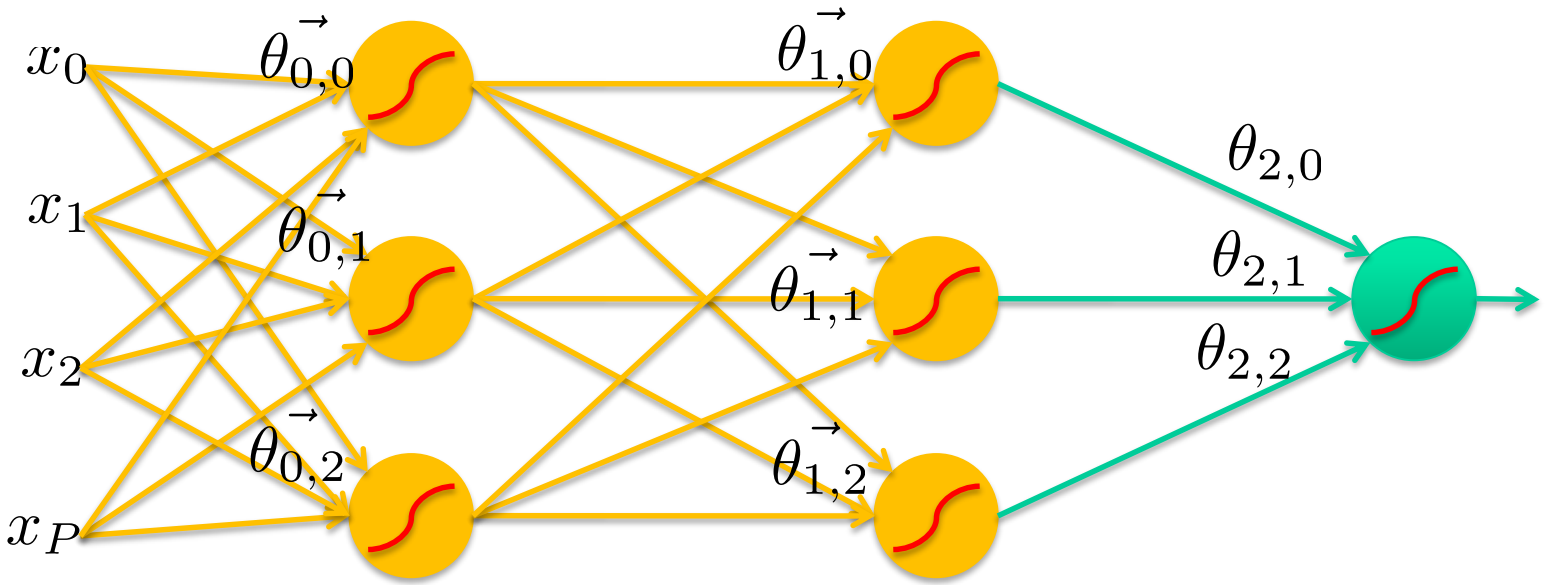
- Predictions are fed forward through the network to classify



Slide credit: HKUST

Feed-forward networks

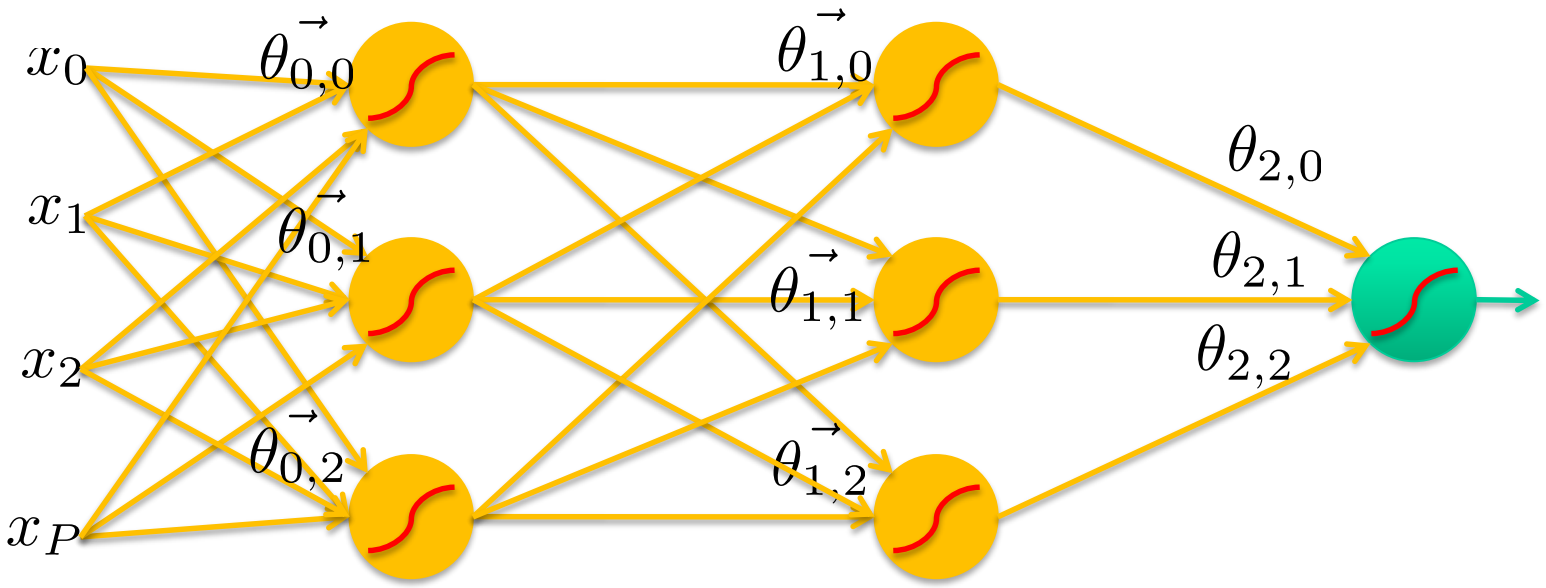
- Predictions are fed forward through the network to classify



Slide credit: HKUST

Feed-forward networks

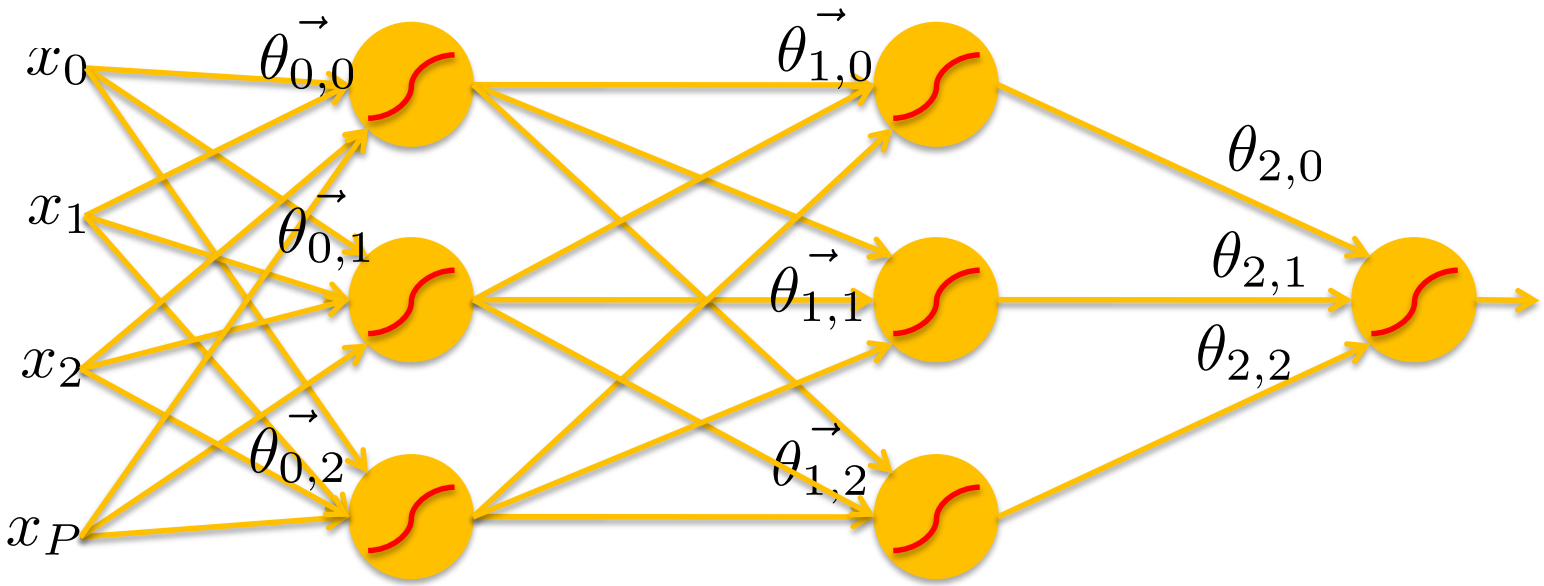
- Predictions are fed forward through the network to classify



Slide credit: HKUST

Feed-forward networks

- Predictions are fed forward through the network to classify



Slide credit: HKUST

Deep neural networks

- Lots of hidden layers
- Depth = power (usually)

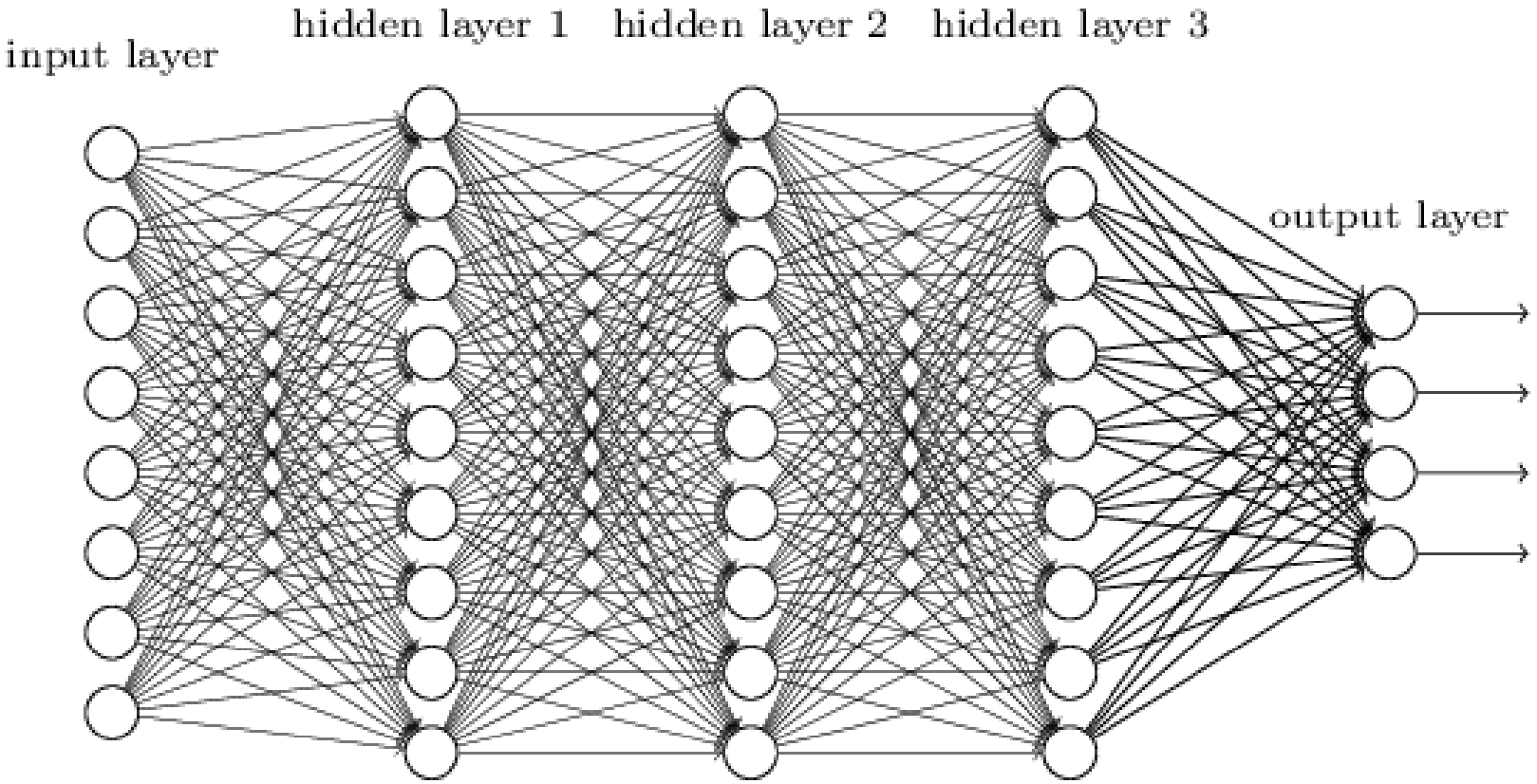
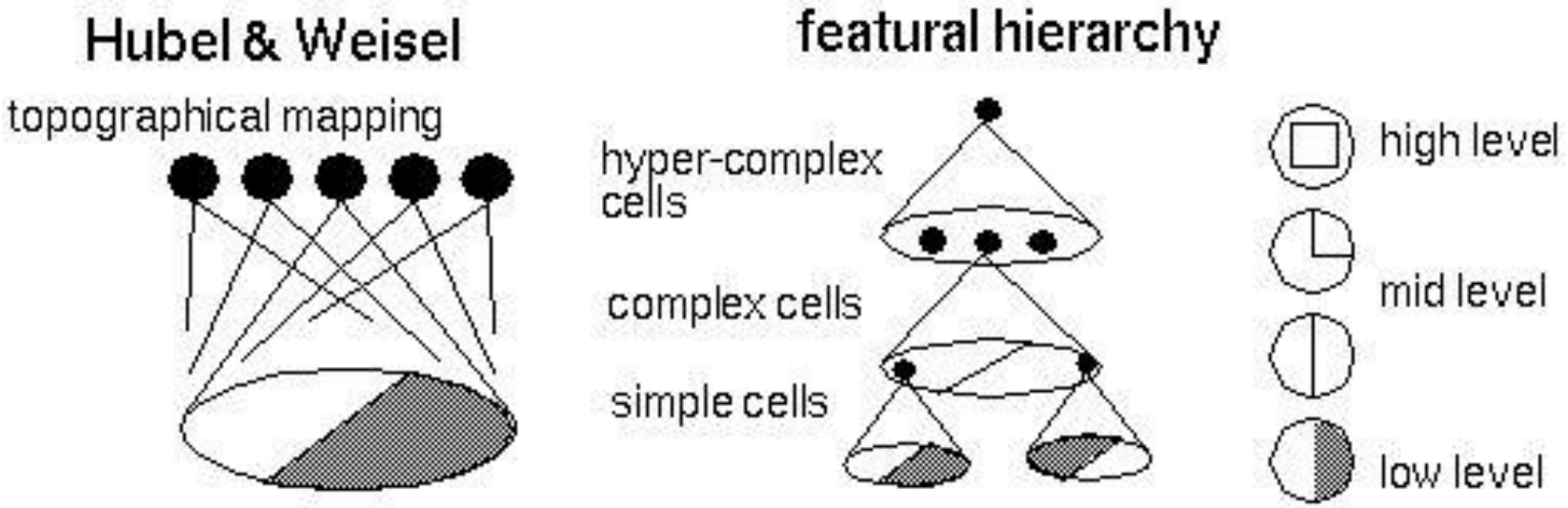


Figure from http://neuralnetworksanddeeplearning.com/chap5.html

Hubel / Wiesel architecture

- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)
  - Visual cortex consists of a hierarchy of *simple*, *complex*, and *hyper-complex* cells
  - **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



Slide credit: Lana Lazebnik, Jia-bin Huang

Plan for today

- Neural network definition and examples
- Training neural networks (backprop)
- Convolutional neural networks
  - Architecture
  - Visualization
  - Analyzing and debugging

Training: Want to minimize error

- Regression:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2$$

- Binary classification (cross-entropy):

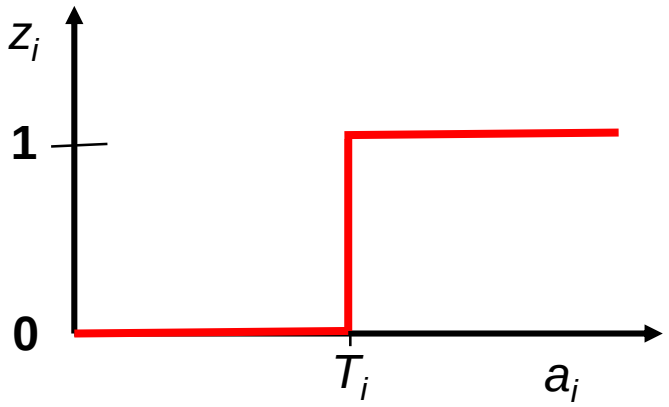
$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

- Multi-way classification:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

Gradient descent in multi-layer nets

- Error for neural networks doesn't have a closed-form solution
- Therefore, we'll use gradient descent
- However, to do gradient descent, we need the output of a unit to be a differentiable function of its input and weights
- E.g. a threshold function is not differentiable at the threshold

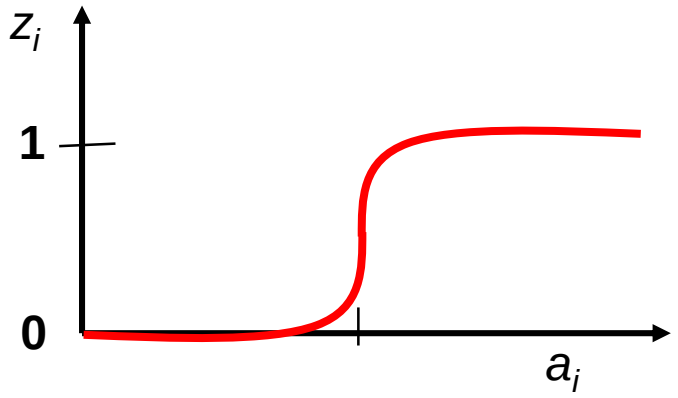


Slide credit: Ray Mooney

Differentiable output function

- Need non-linear output function to move beyond linear functions
- Standard solution is to use the non-linear, differentiable sigmoidal “logistic” function:

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$



Slide credit: Ray Mooney

Gradient descent in multi-layer nets

- We'll update weights:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

- Move in direction opposite to gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)})$$

How to compute gradient?

- Consider simple linear function:

$$y_k = \sum_i w_{ki} x_i$$

- Error is:

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

- Gradient is:

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

How to compute gradient?

- In a neural network:

$$a_j = \sum_i w_{ji} z_i \qquad z_j = h(a_j)$$

- Gradient is:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

- Denote the errors as:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

- Also:

$$\frac{\partial a_j}{\partial w_{ji}} = z_i$$

Backpropagation: Error

- For output units:

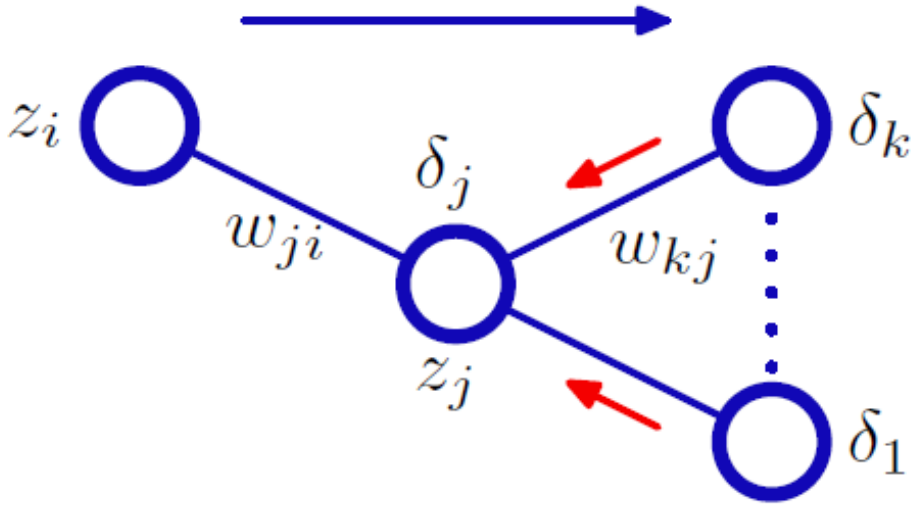
$$\delta_k = y_k - t_k$$

- For hidden units:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

- Backprop formula:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$



Example (identity output function)

- Two layer network w/ sigmoid at hidden layer:

$$h(a) \equiv \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

- Derivative: $h'(a) = 1 - h(a)^2$
- Minimize: $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$
- Forward propagation:

$$\begin{aligned} a_j &= \sum_{i=0}^{D} w_{ji}^{(1)} x_i \\ z_j &= \tanh(a_j) \\ y_k &= \sum_{j=0}^{M} w_{kj}^{(2)} z_j \end{aligned}$$

Example (identity output function)

- Errors at output:

$$\delta_k = y_k - t_k$$

- Errors at hidden units:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$$

- Derivatives wrt weights:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Graphic example

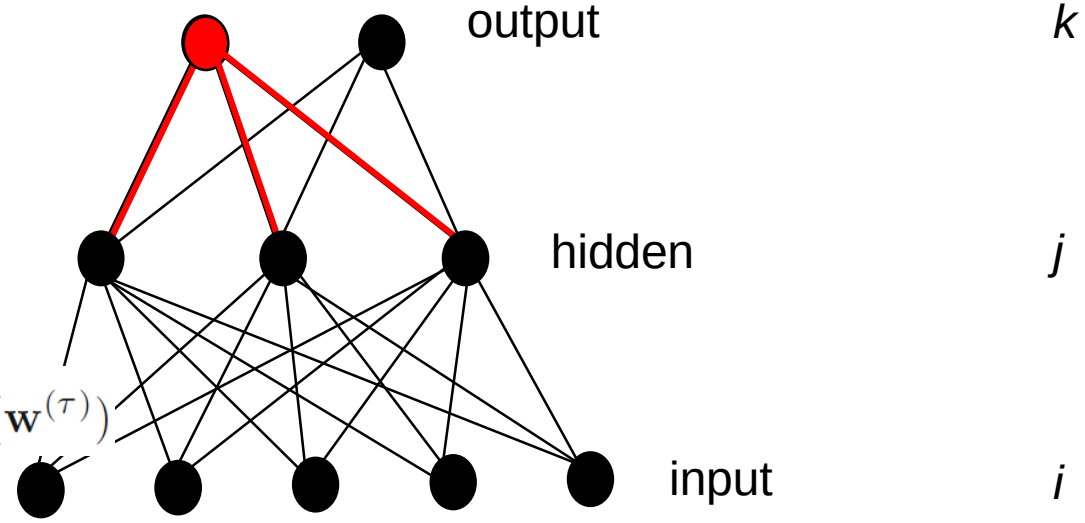
First calculate error of output units and use this to change the top layer of weights.

$$\delta_k = y_k - t_k$$

Update weights into *j*

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

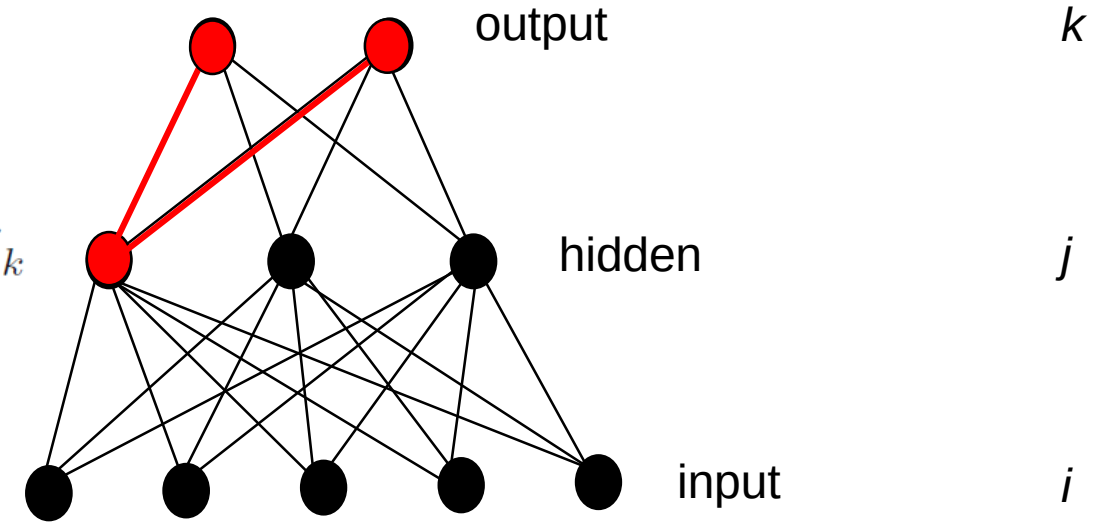
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$



Graphic example

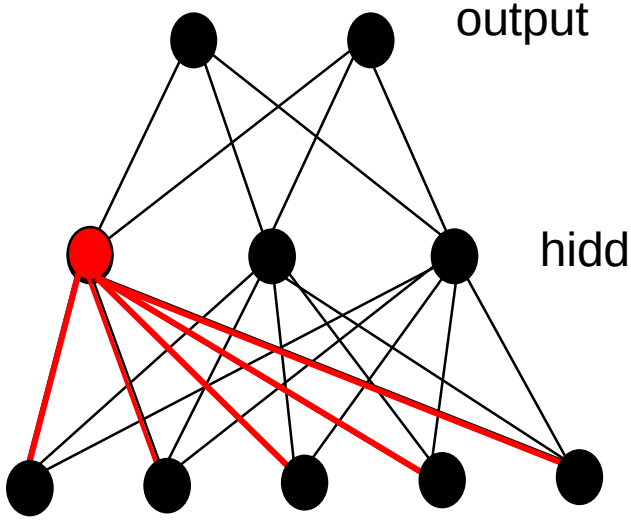
Next calculate error for hidden units based on errors on the output units it feeds into.

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$$



Graphic example

Finally update bottom layer of weights based on errors calculated for hidden units.



Update weights into *i*

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

General algorithm *for sigmoid outputs*

- Initialize all weights to small random values
- Until convergence (e.g. all training examples' error small, or error stops decreasing) repeat:
  - For each `(x, t=class(x))` in training set:
    - Calculate network outputs: y_k
    - Compute errors (gradients wrt activations) for each unit:
      - » $\delta_k := y_k (1-y_k) (t_k - y_k)$ for output units
      - » $\delta_j := y_k (1-y_k) \sum_k w_{kj} \delta_k$ for hidden units
    - Update weights:
      - » $w_{kj} \leftarrow w_{kj} - \eta \delta_k z_j$ for output units
      - » $w_{ji} \leftarrow w_{ji} - \eta \delta_j x_i$ for hidden units

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

Adapted from Rebecca Hwa and Ray Mooney

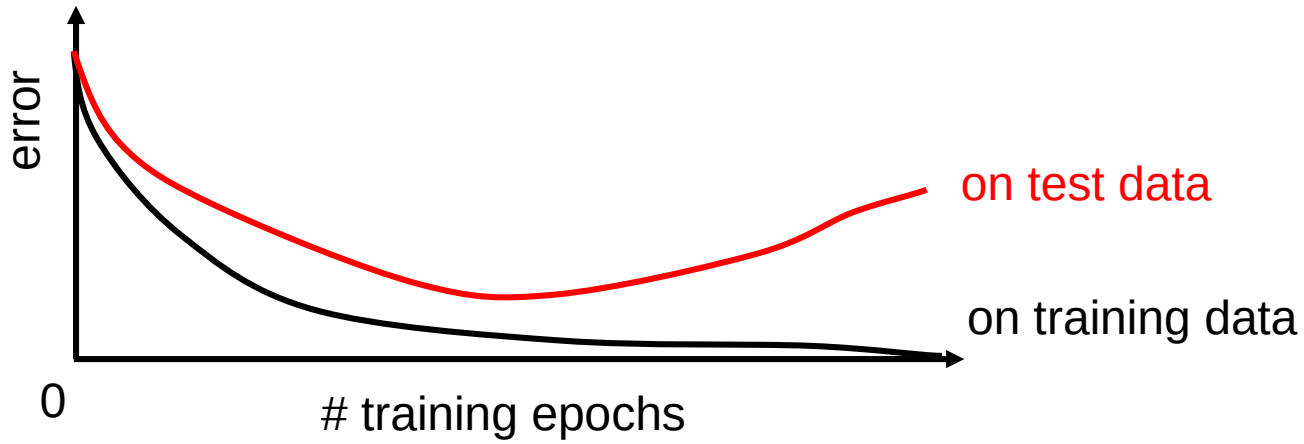
Comments on training algorithm

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data.
- Thousands of epochs (epoch = network sees all training data once) may be required, hours or days to train.
- To avoid local-minima problems, run several trials starting with different random weights (*random restarts*), and take results of trial with lowest training set error.
- May be hard to set learning rate and to select number of hidden units and layers.
- Neural networks had fallen out of fashion in 90s, early 2000s; back with a new name and significantly improved performance (deep networks trained with dropout and lots of data).

Slide credit: Ray Mooney, Carlos Guestrin, Dhruv Batra

Over-training prevention

- Running too many epochs can result in over-fitting.

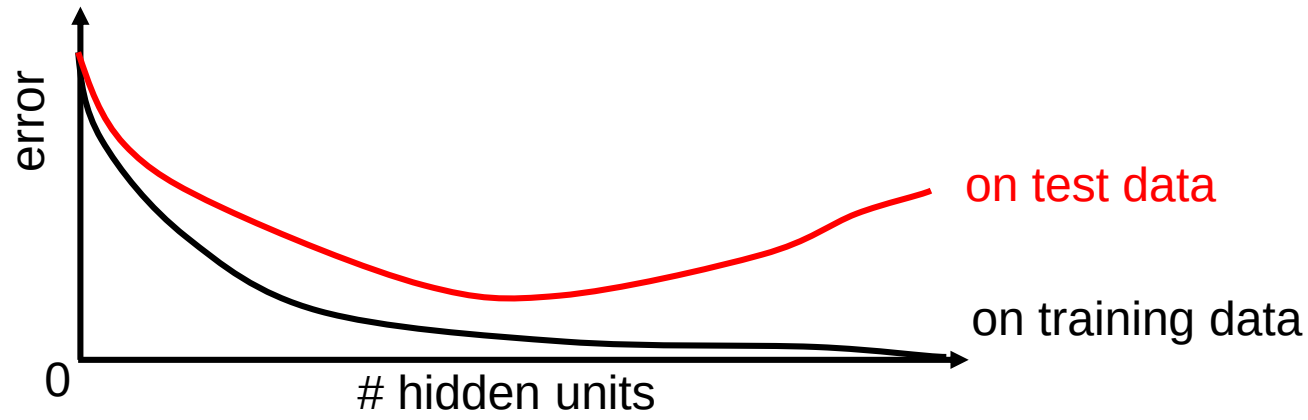


- Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.
- To avoid losing training data for validation:
  - Use internal 10-fold CV on the training set to compute avg number of epochs that maximizes generalization accuracy.
  - Train final network on complete training set for this many epochs.

Slide credit: Ray Mooney

Determining best number of hidden units

- Too few hidden units prevents the network from adequately fitting the data.
- Too many hidden units can result in over-fitting.



- Use internal cross-validation to empirically determine an optimal number of hidden units.

Slide credit: Ray Mooney

Hidden unit representations

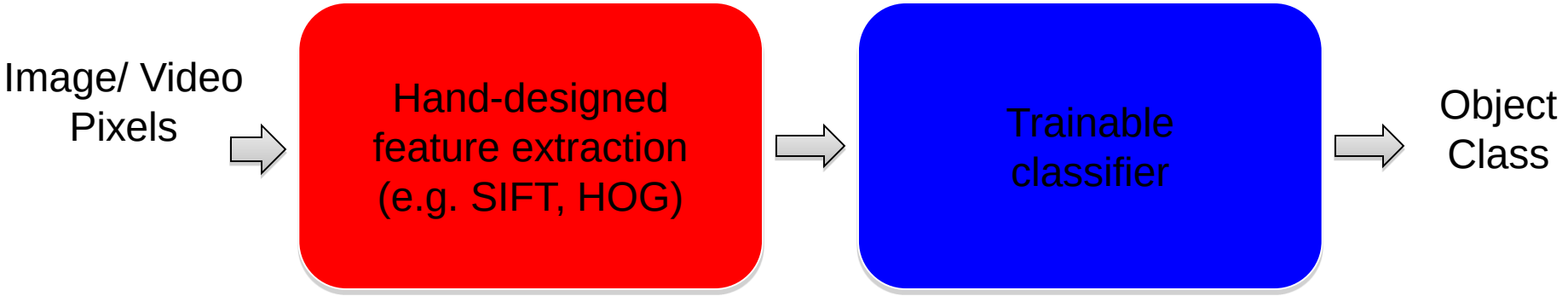
- Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
- On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc.
- However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.

Slide credit: Ray Mooney

Plan for today

- Neural network definition and examples
- Training neural networks (backprop)
- Convolutional neural networks
  - Architecture
  - Visualization
  - Analyzing and debugging

Traditional categorization in computer vision



- Features are key to recent progress in recognition, but flawed…
- Where next? Better classifiers? Or keep building more features?

Adapted from Lana Lazebnik

What about learning the features?

- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

Slide credit: Lana Lazebnik

How to use for your own classification task

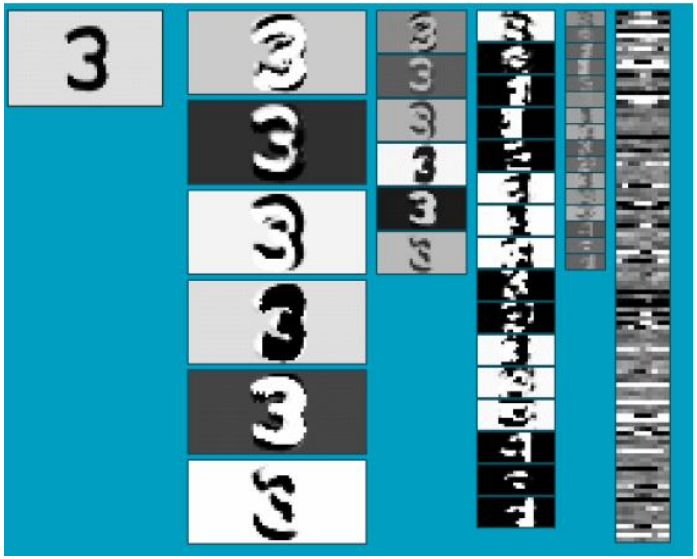
- Take model pre-trained on a large image dataset
- **Take activations from the last few layers**
- Optional: Fine-tune network on dataset for your own task
- Optional: Train network from scratch (need lots of data)
- Classify test set of new dataset

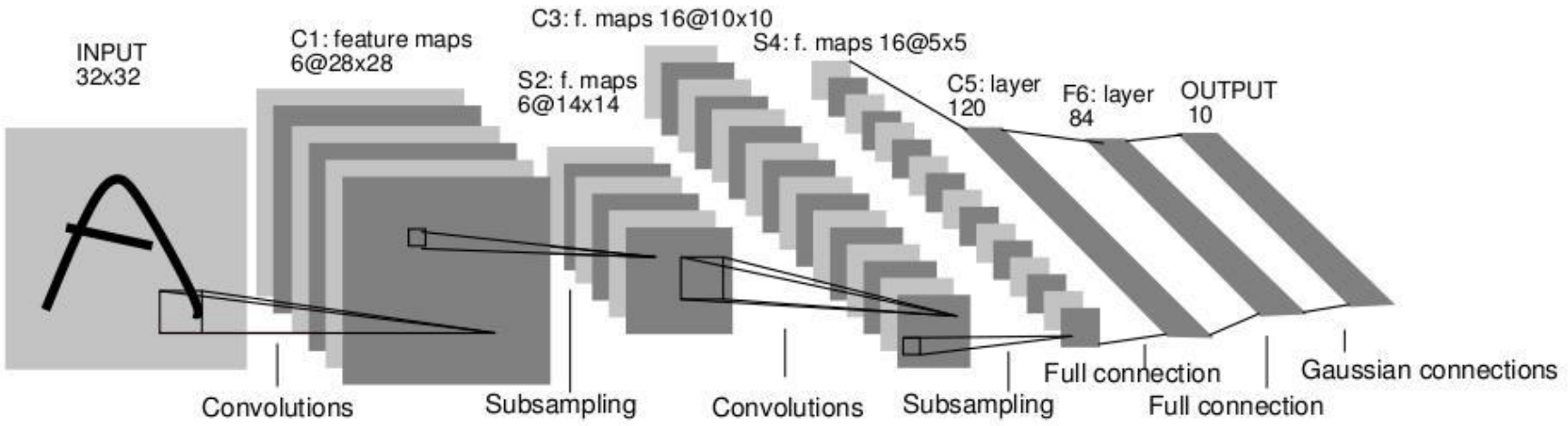
- Why now: We have lots of data, and deep nets can be trained in reasonable time with GPUs

Adapted from Lana Lazebnik

Convolutional Neural Networks (CNN)

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant, *more abstract* features
- Classification layer at the end



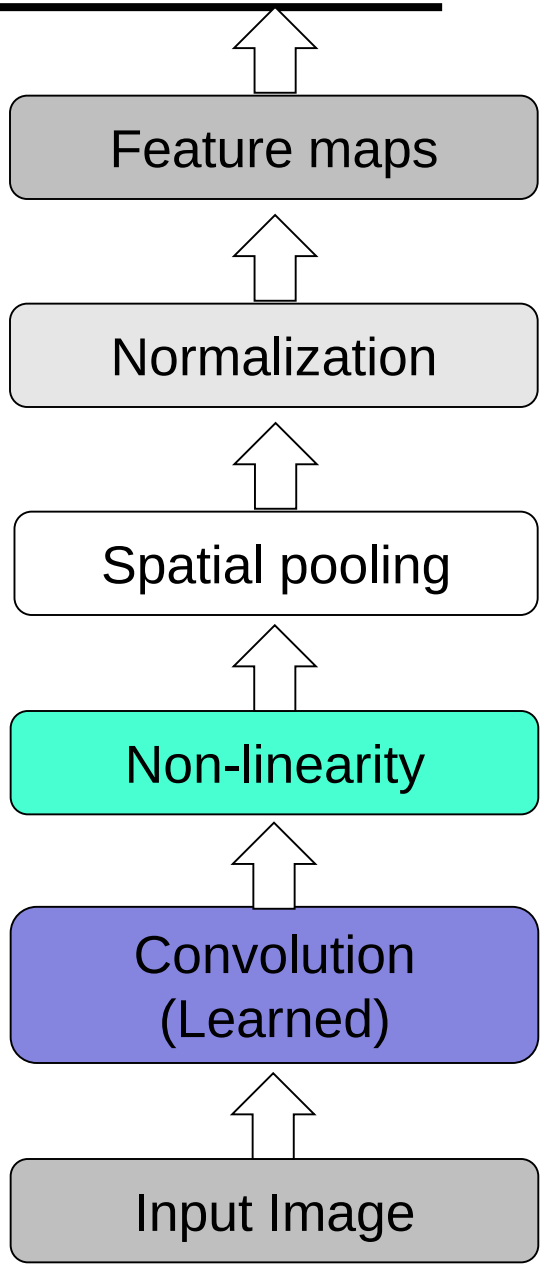


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86(11): 2278–2324, 1998.

Adapted from Rob Fergus

Convolutional Neural Networks (CNN)

- Feed-forward feature extraction:
  1. Convolve input with learned filters
  2. Apply non-linearity
  3. Spatial pooling (downsample)
  4. Normalization (optional)
- Supervised training of convolutional filters by back-propagating classification error

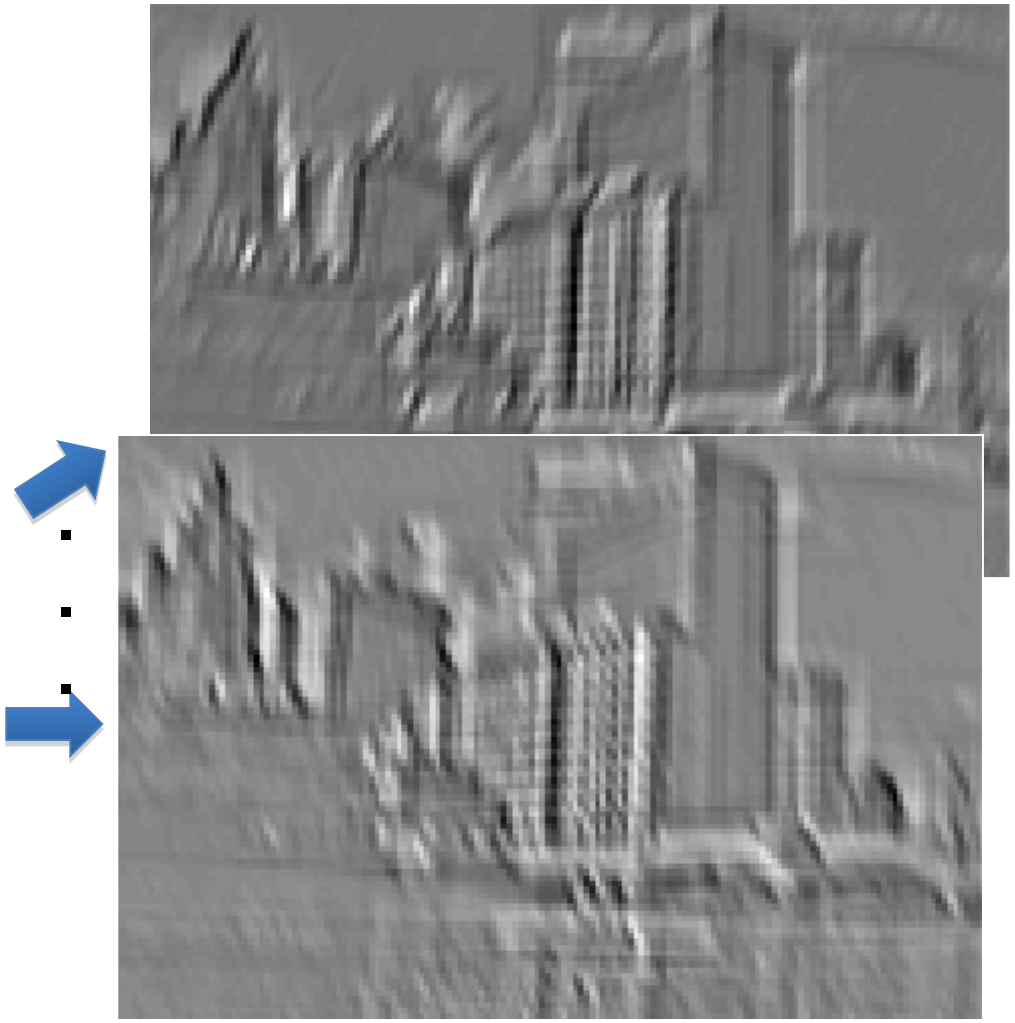


Adapted from Lana Lazebnik

1. Convolution

- Apply learned filter weights across the image
- One feature map per filter

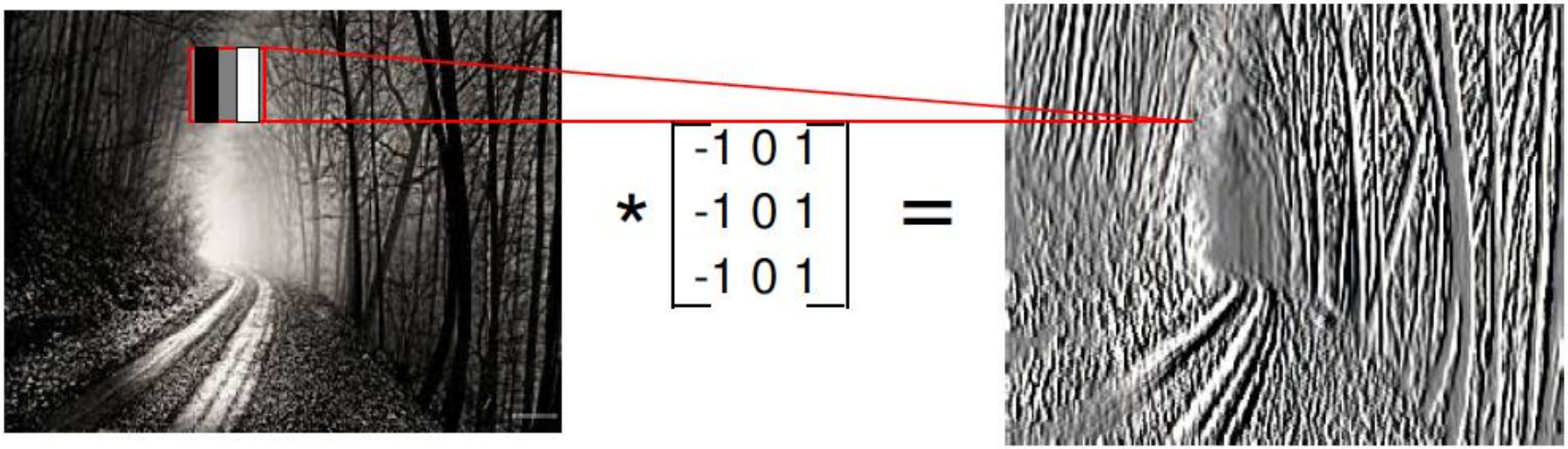
Input



Feature Map

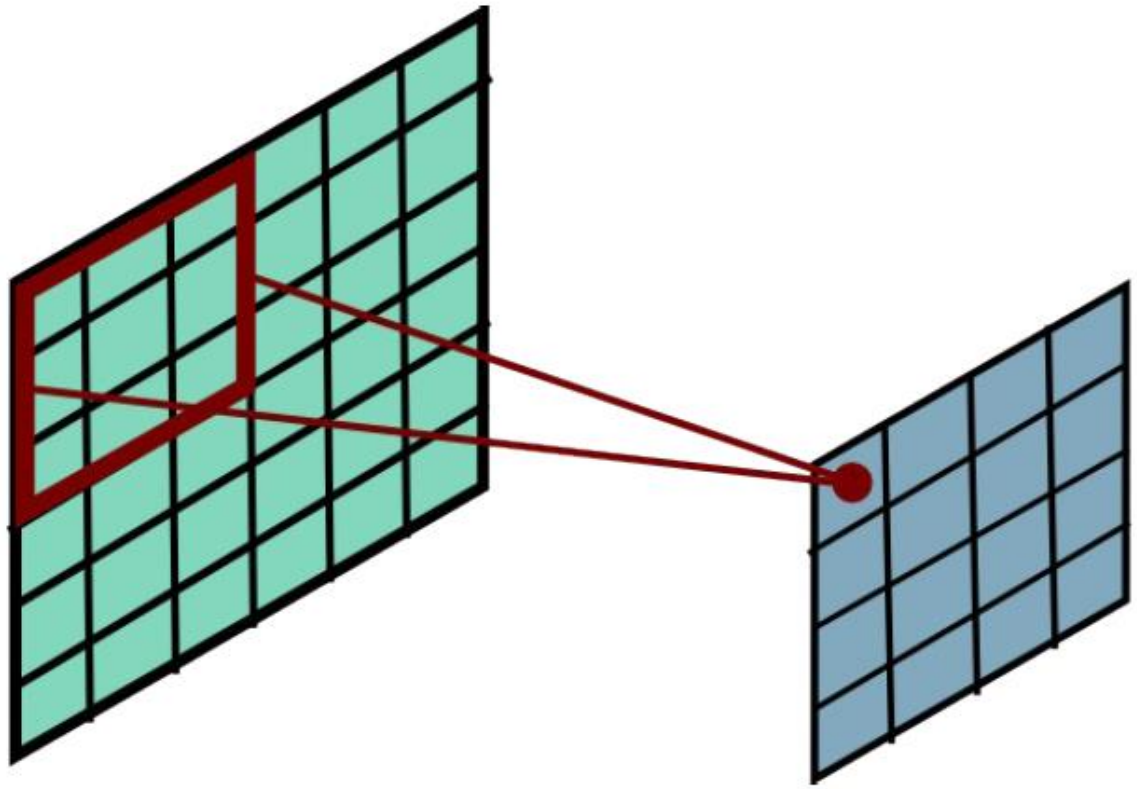
Adapted from Rob Fergus

1. Convolution



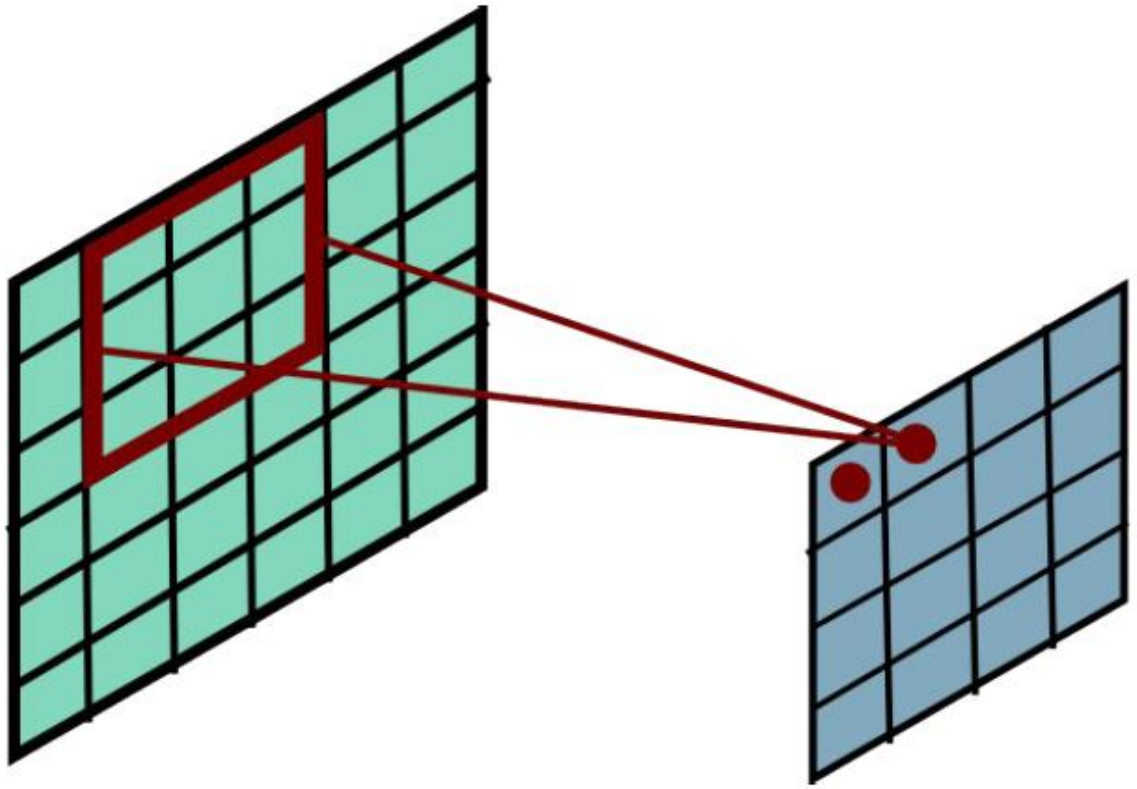
Slide credit: Marc'Aurelio Ranzato

1. Convolution



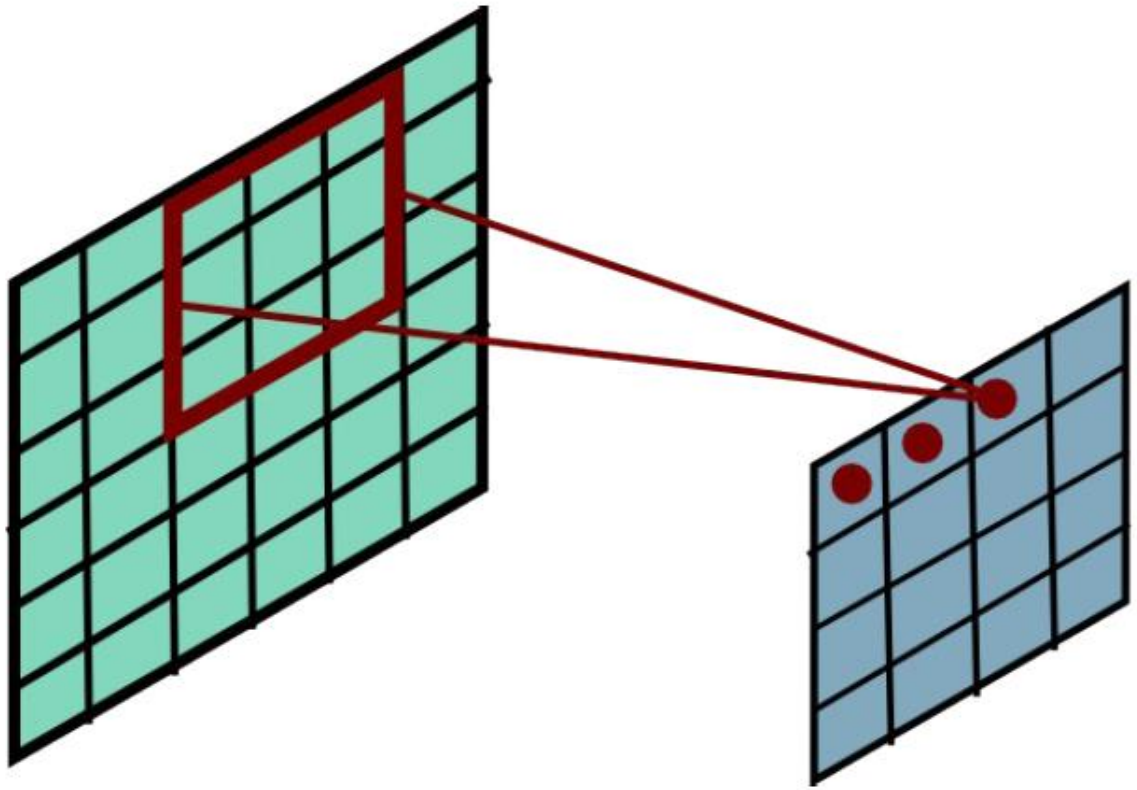
Slide credit: Marc'Aurelio Ranzato

1. Convolution



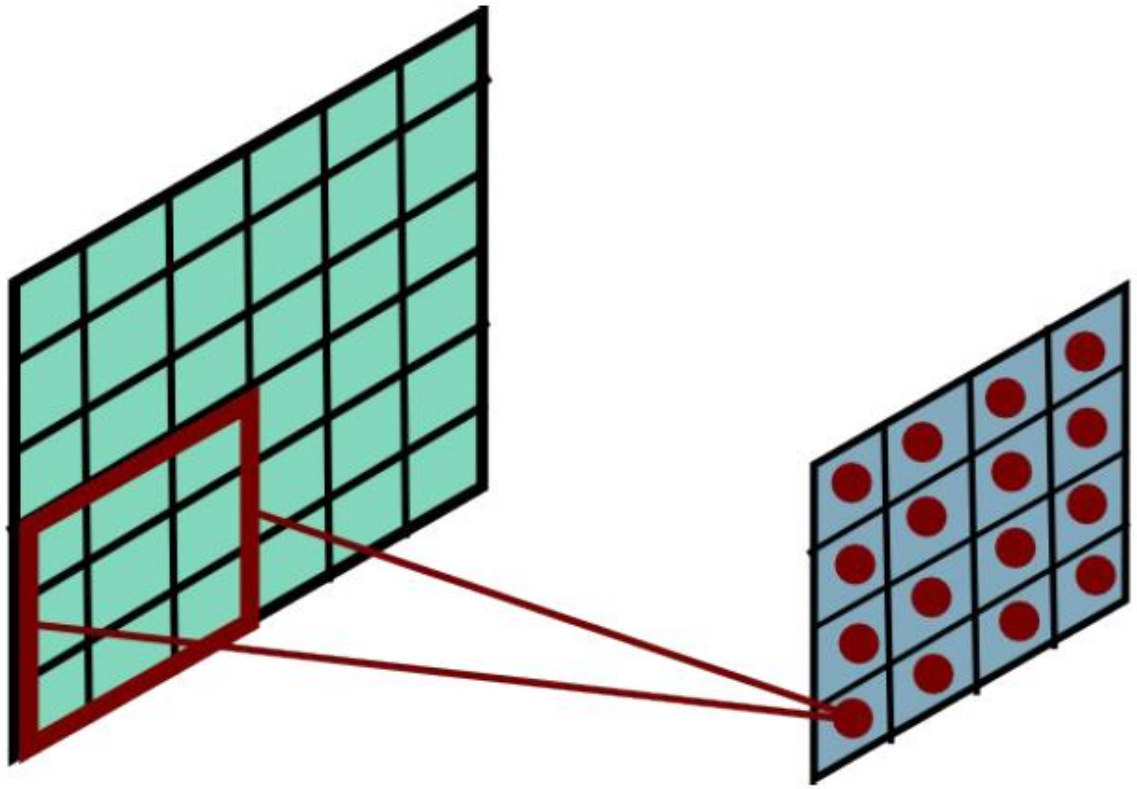
Slide credit: Marc'Aurelio Ranzato

1. Convolution



Slide credit: Marc'Aurelio Ranzato

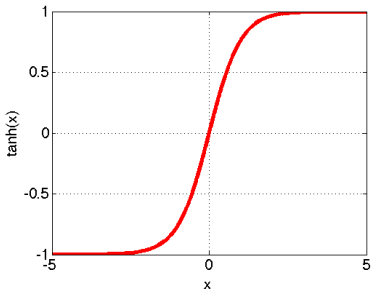
1. Convolution

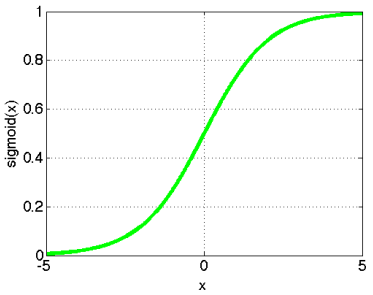


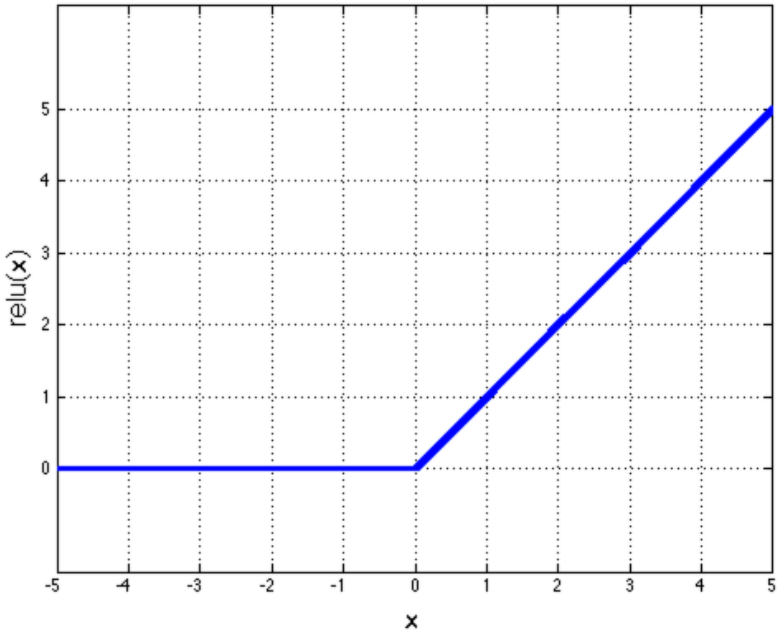
Slide credit: Marc'Aurelio Ranzato

2. Non-Linearity

- Per-element (independent)
- Options:
  - Tanh
  - Sigmoid: 1/(1+exp(-x))
  - Rectified linear unit (ReLU)
    - Simplifies backpropagation
    - Makes learning faster
    - Avoids saturation issues
    - Preferred option (works well)



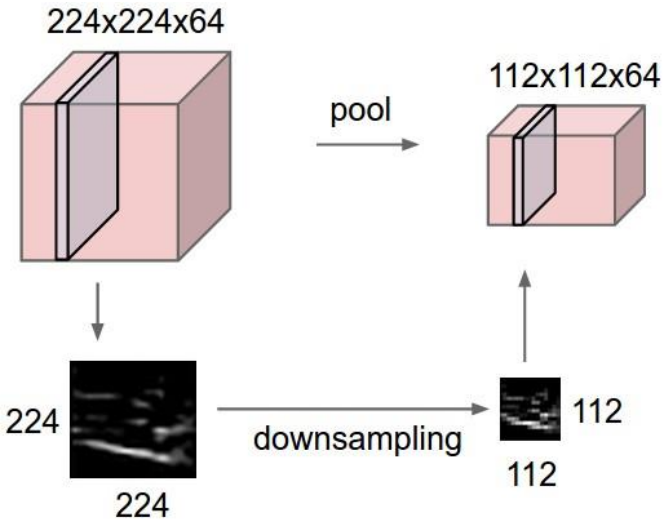


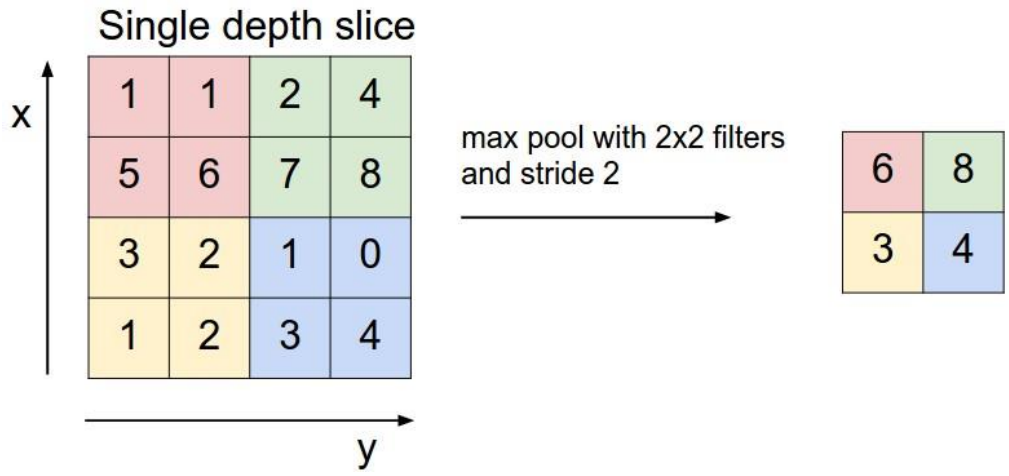


Slide credit: Rob Fergus

3. Spatial Pooling

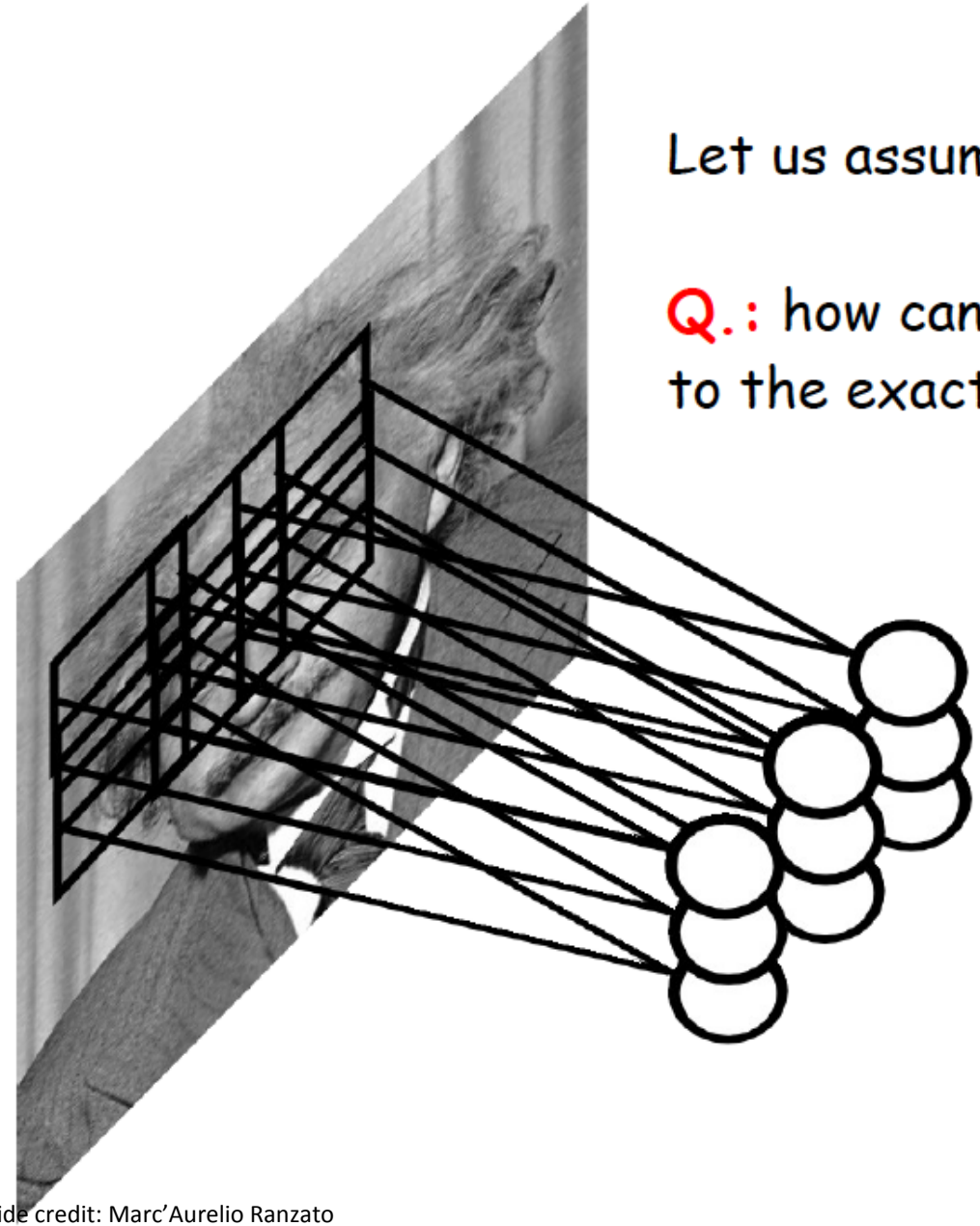
- Sum or max
- Non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (see more of input)





Slide credit: Rob Fergus, figure from Andrej Karpathy

CONVOLUTIONAL NET

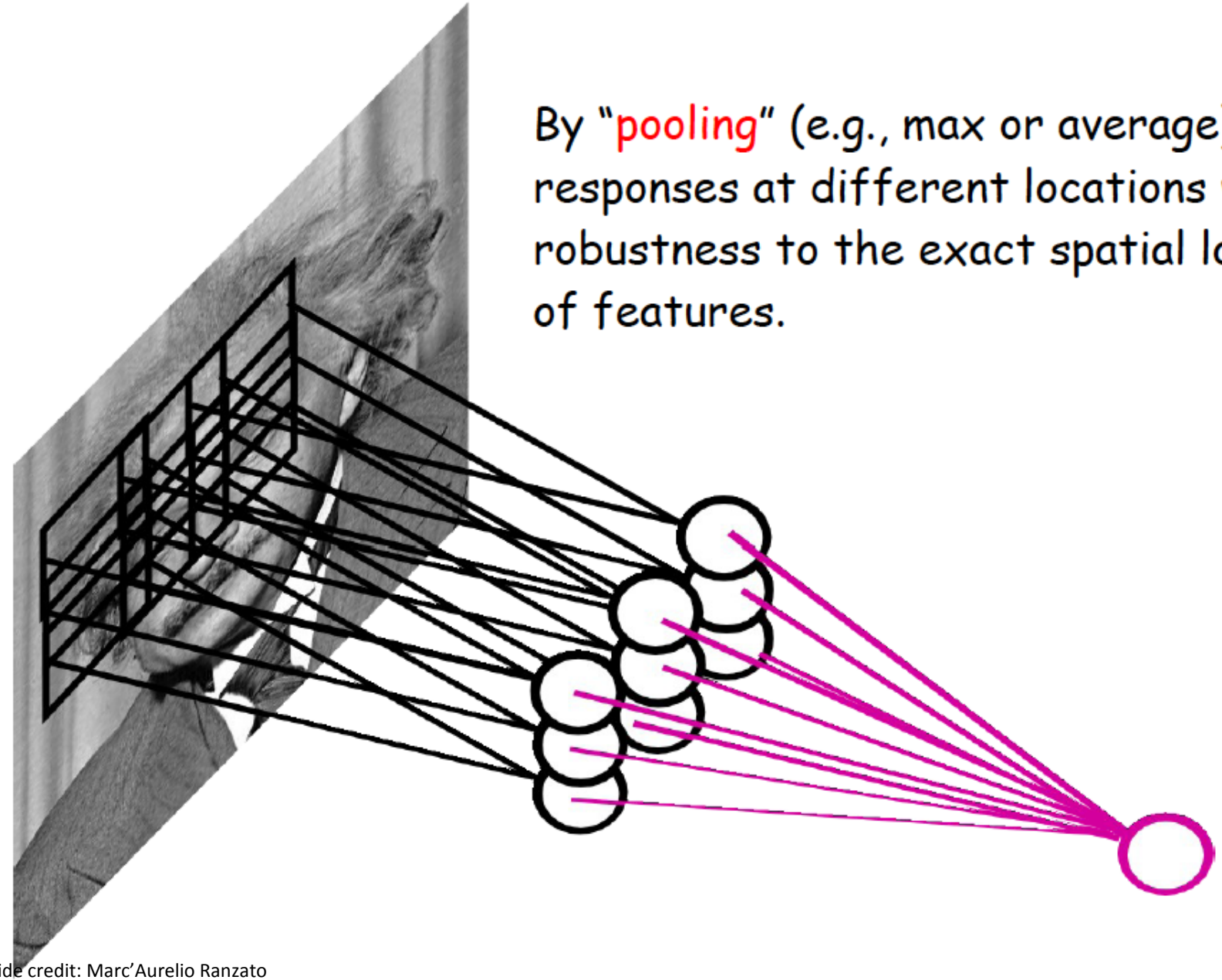


Let us assume filter is an "eye" detector.

Q.: how can we make the detection robust to the exact location of the eye?

Slide credit: Marc'Aurelio Ranzato

CONVOLUTIONAL NET

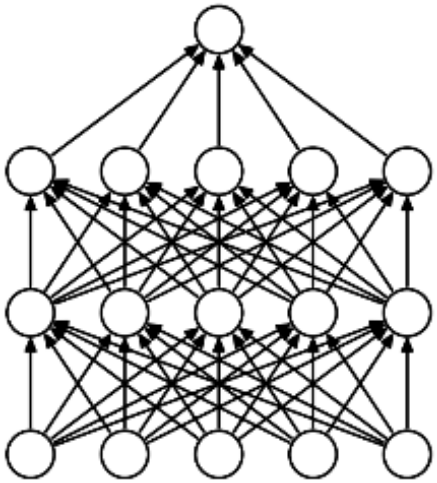


By "pooling" (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features.

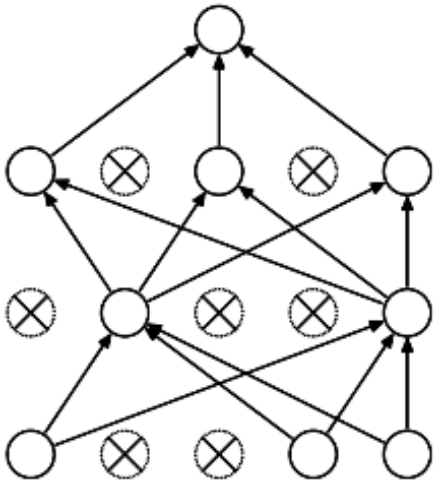
Slide credit: Marc'Aurelio Ranzato

Training trick 1: Dropout

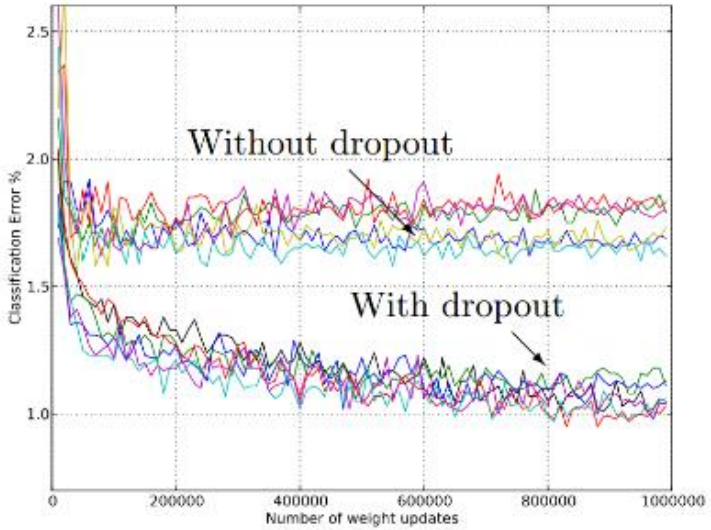
- Randomly turn off some neurons
- Allows individual neurons to independently be responsible for performance



(a) Standard Neural Net



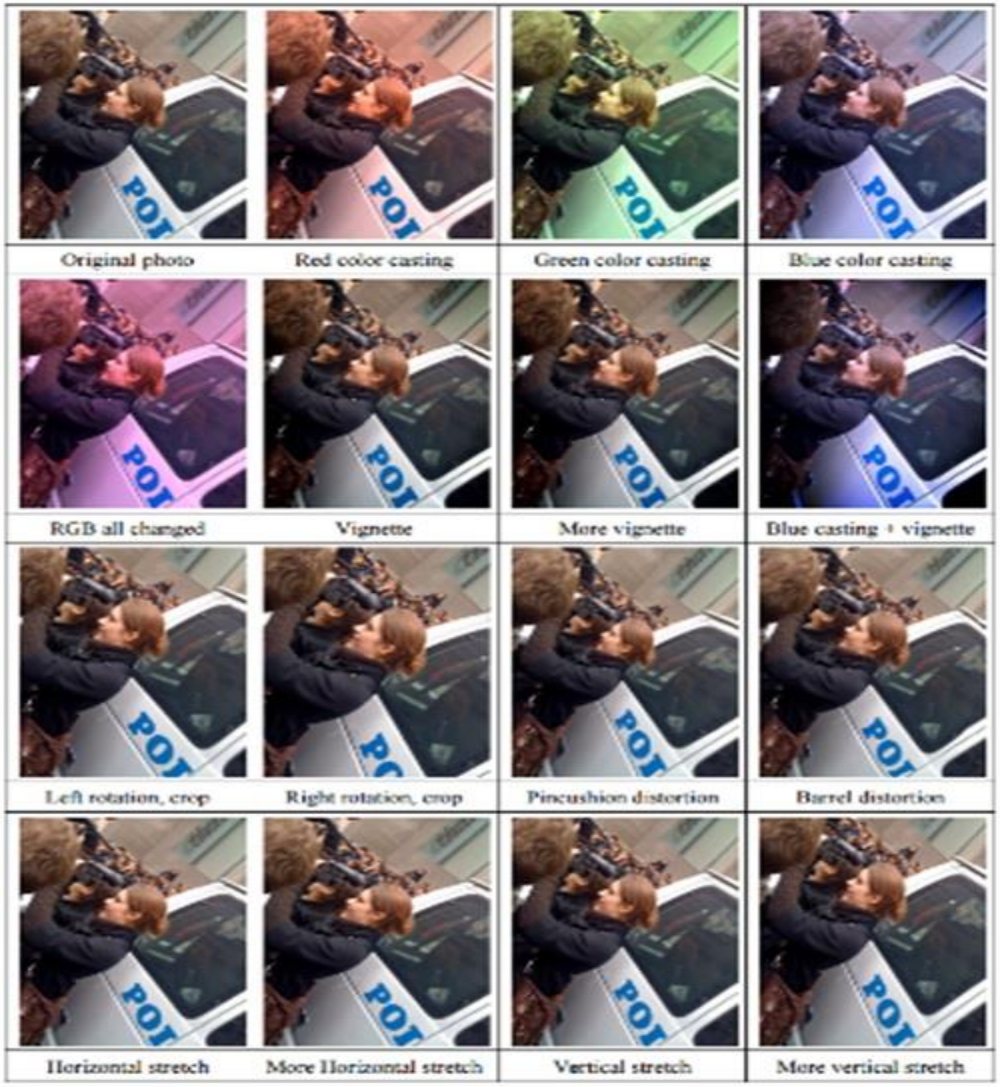
(b) After applying dropout.



Adapted from Jia-bin Huang

Training trick 2: Data augmentation

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion



Slide credit: Jia-bin Huang

ConvNet Successes

- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]
- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition
    - 0.56% error vs 1.16% for humans [Ciresan et al. 2011]
- But until recently, less good at more complex datasets
  - Caltech-101/256 (few training examples)

Slide credit: Lana Lazebnik

ImageNet Challenge 2012

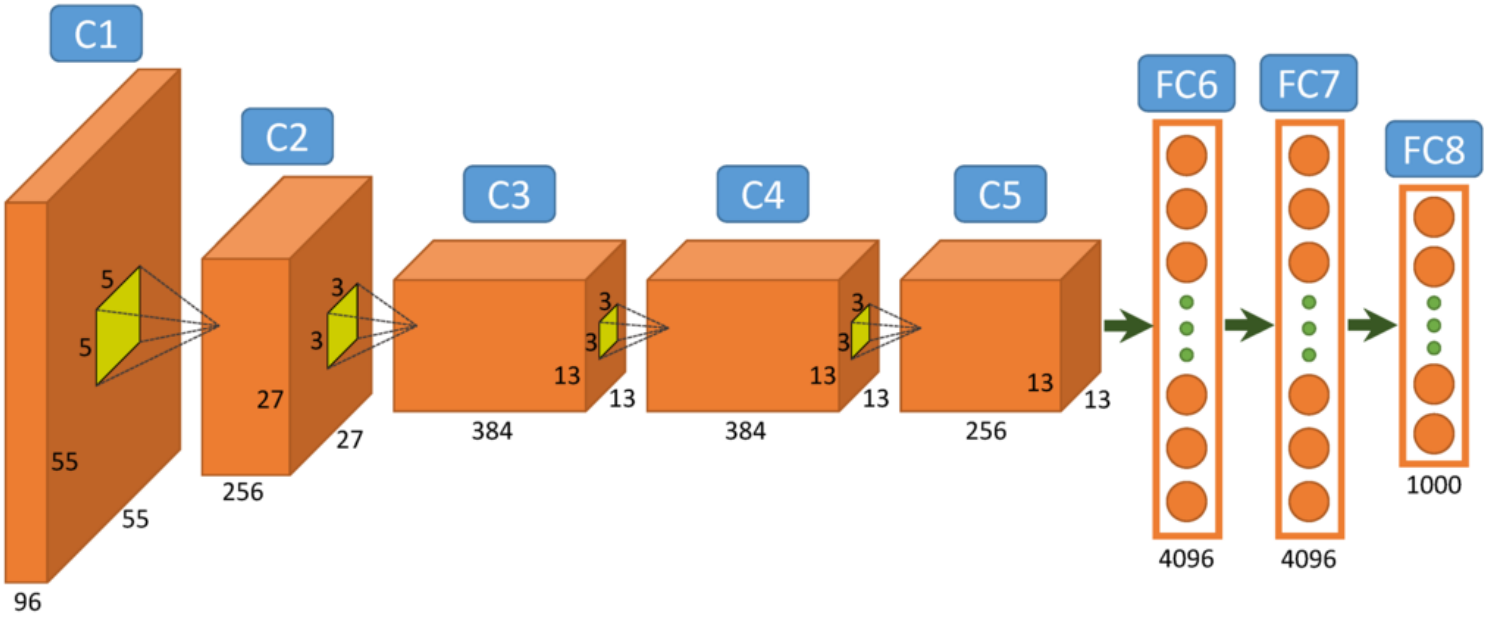
[Deng et al. CVPR 2009]

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk
- Challenge: 1.2 million training images, 1000 classes

Slide credit: Lana Lazebnik

ImageNet Challenge 2012

- AlexNet: Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650k units, 60mil params)
  - More data (10^6 vs. 10^3 images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Better regularization for training (DropOut)



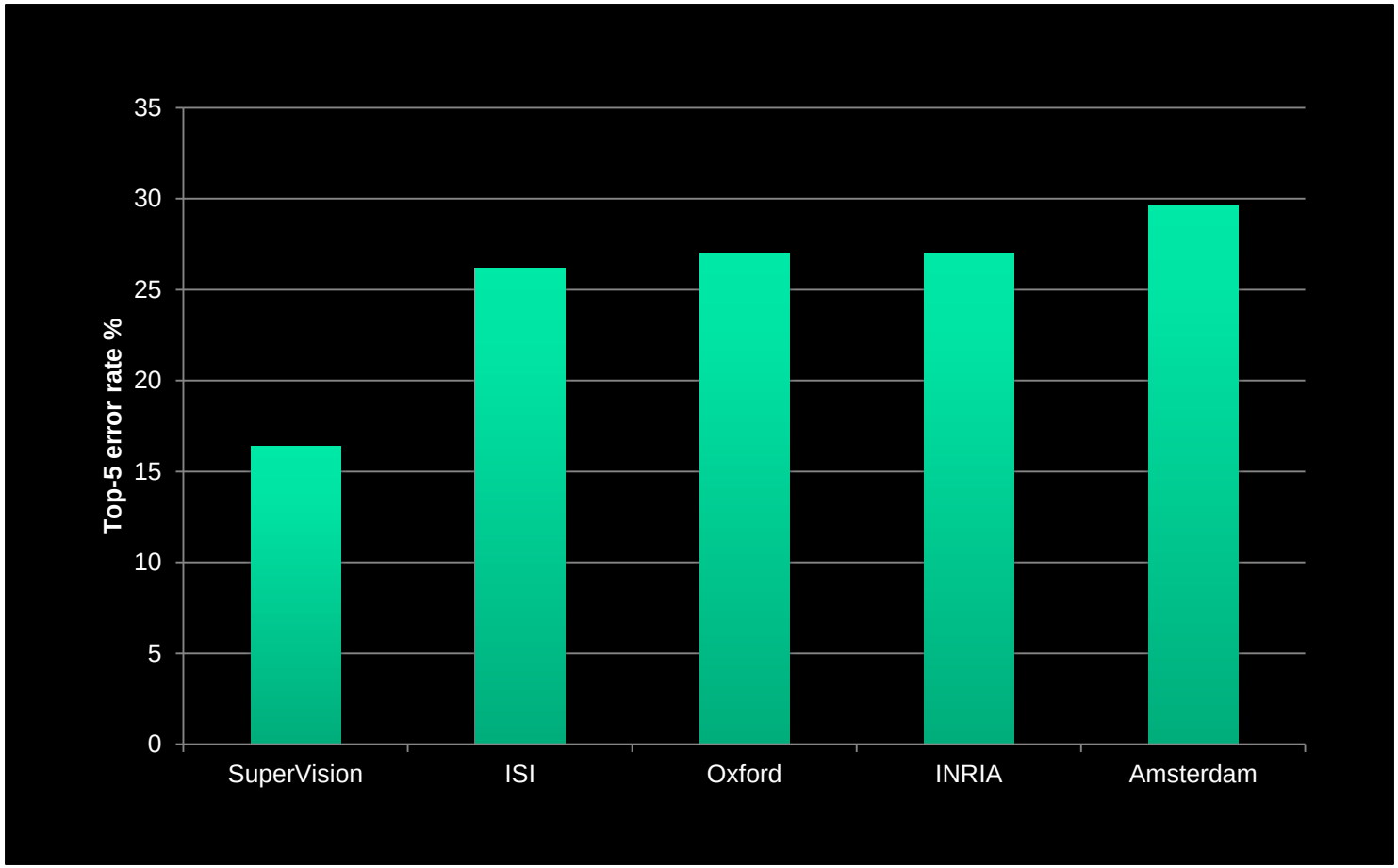
A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Slide credit: Lana Lazebnik

ImageNet Challenge 2012

Krizhevsky et al. – **16.4% error** (top-5)

Next best (non-convnet) – **26.2% error**



Slide credit: Lana Lazebnik

ImageNet Challenge 2012-2014

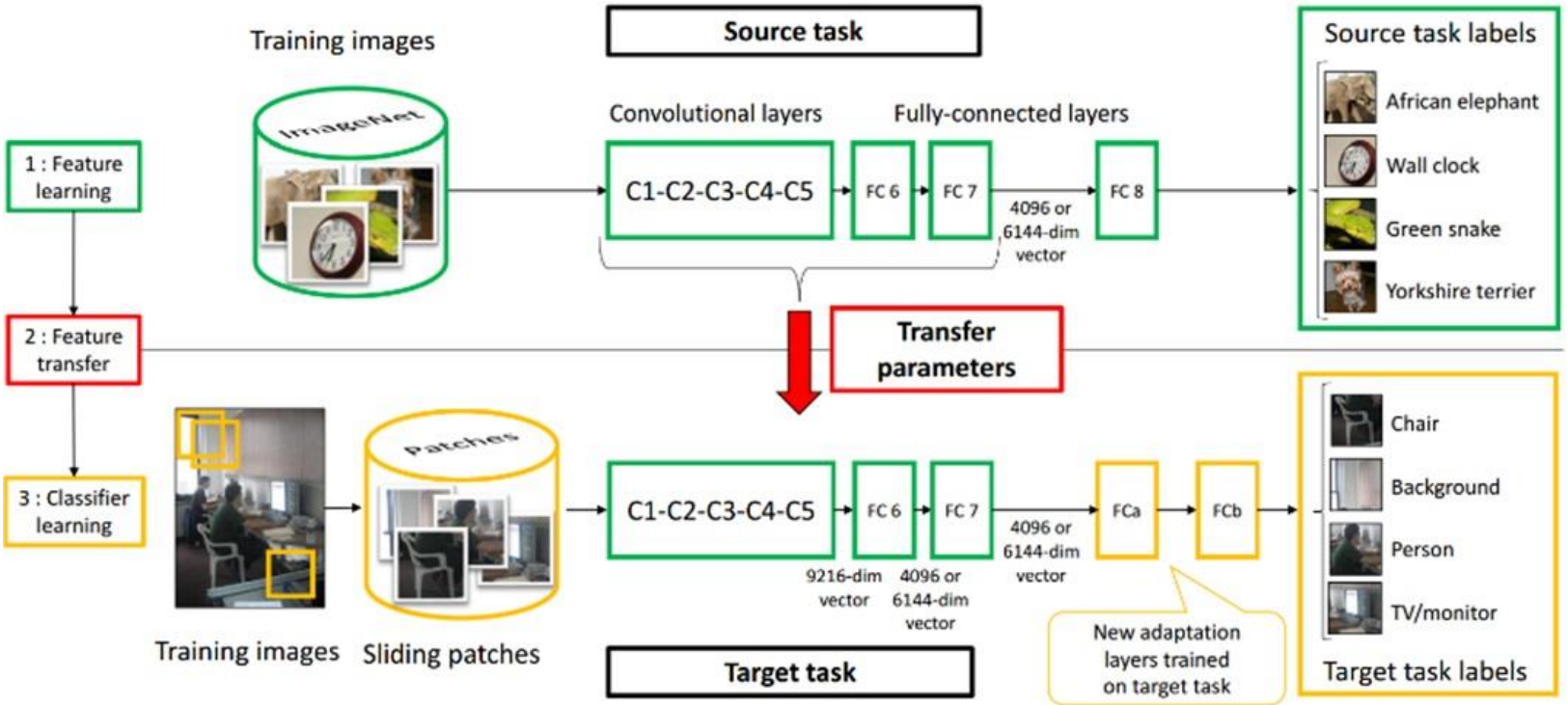
Best non-convnet in 2012: 26.2%

Team	Year	Place	Error (top-5)
SuperVision – Toronto (7 layers)	2012	-	16.4%
Clarifai – NYU (7 layers)	2013	-	11.7%
VGG – Oxford (16 layers)	2014	2nd	7.32%
GoogLeNet (19 layers)	2014	1st	6.67%
Human expert*			5.1%

Slide credit: Jia-bin Huang

Transfer learning with CNNs

- Improve learning of a **new** task through the *transfer of knowledge* from a **related** task that has already been learned



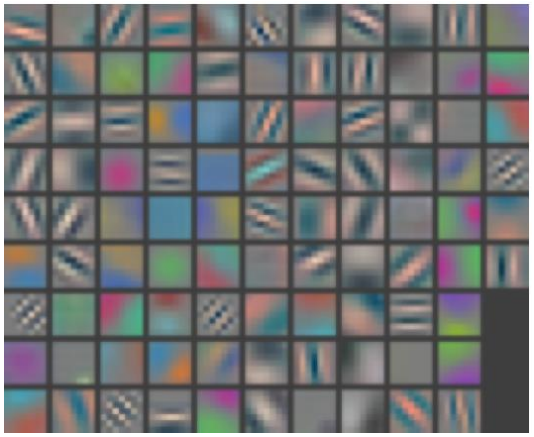
Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks [Oquab et al. CVPR 2014]

Slide credit: Jia-bin Huang

Plan for today

- Neural network definition and examples
- Training neural networks (backprop)
- Convolutional neural networks
  - Architecture
  - Visualization
  - Analyzing and debugging

Layer 1

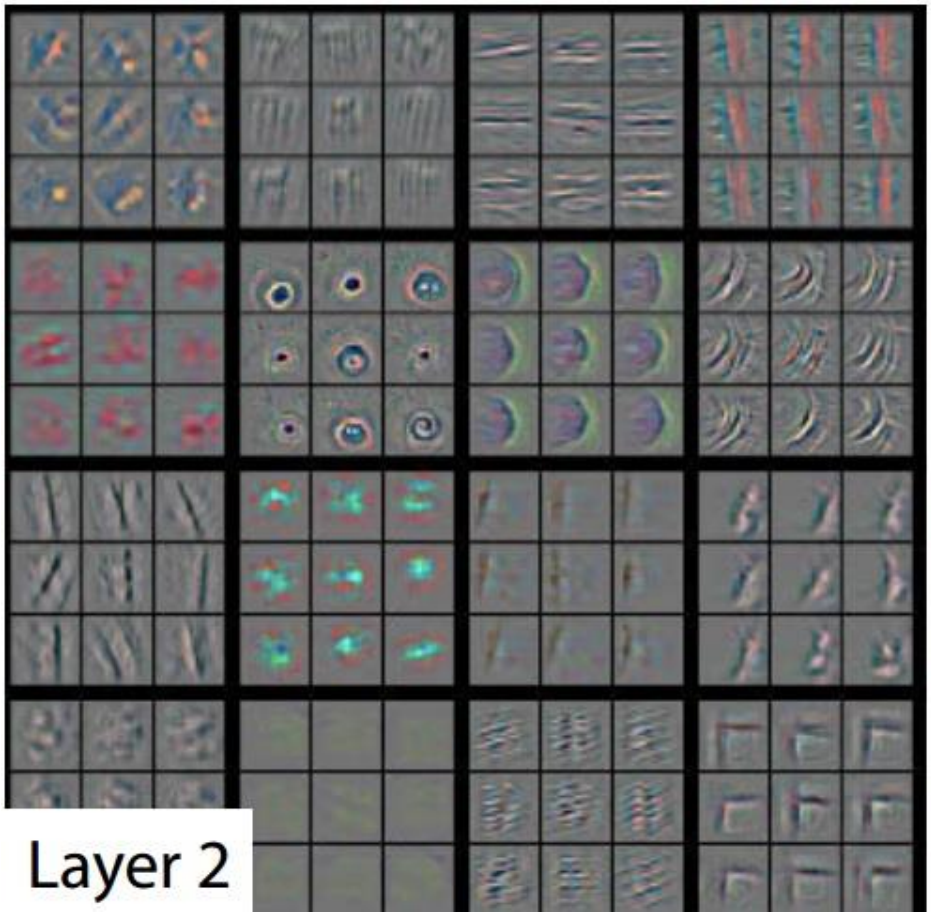


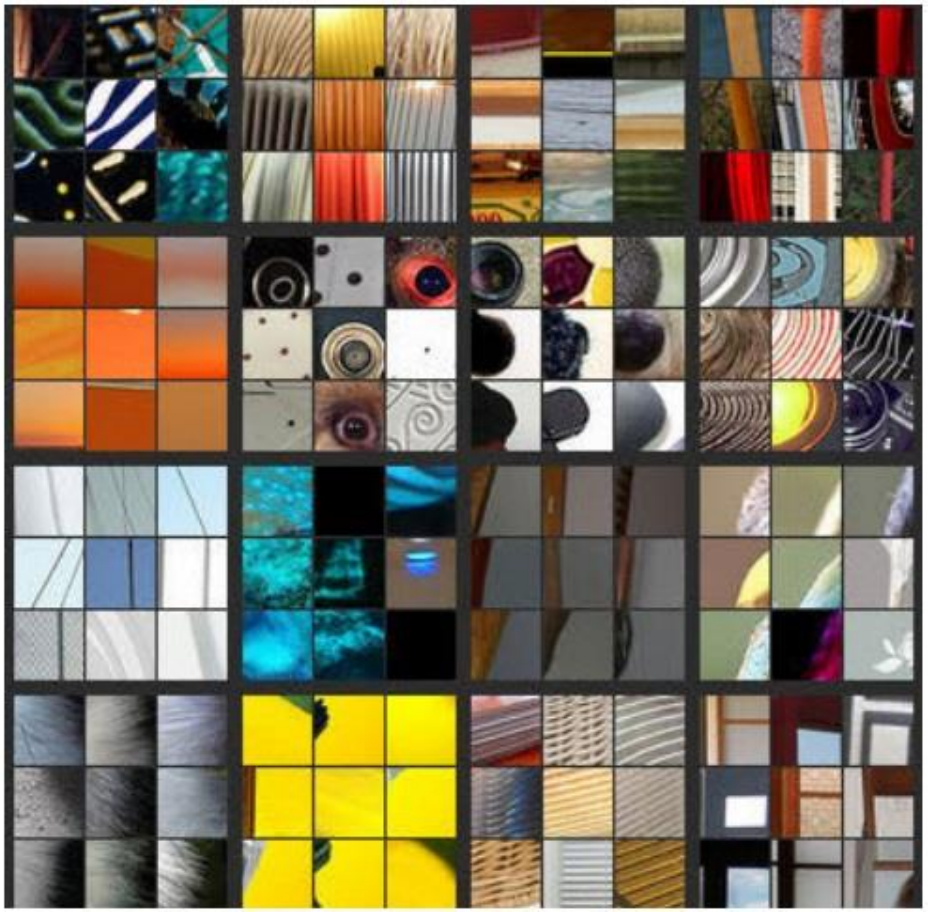
- Weights

- Patches from validation images that give maximal activation of a given feature map

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 2

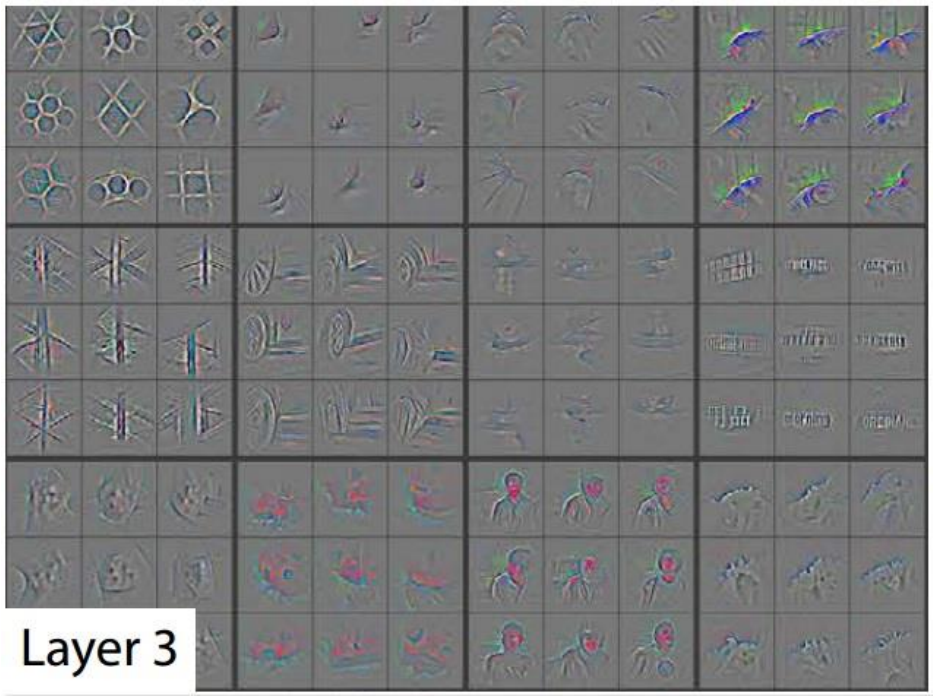




- Weights
- Patches from validation images that give maximal activation of a given feature map

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

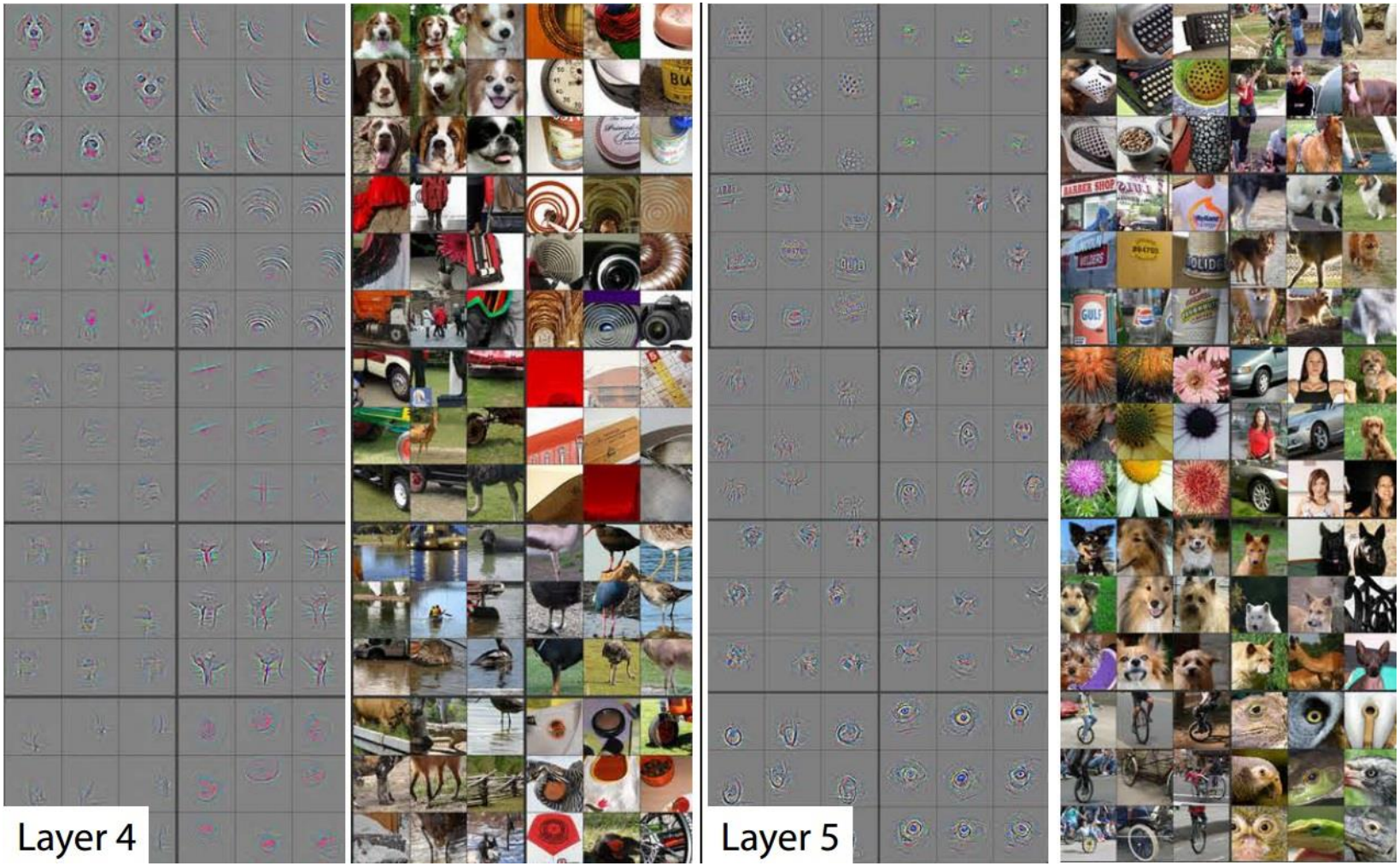
Layer 3



- Weights
- Patches from validation images that give maximal activation of a given feature map

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 4 and 5

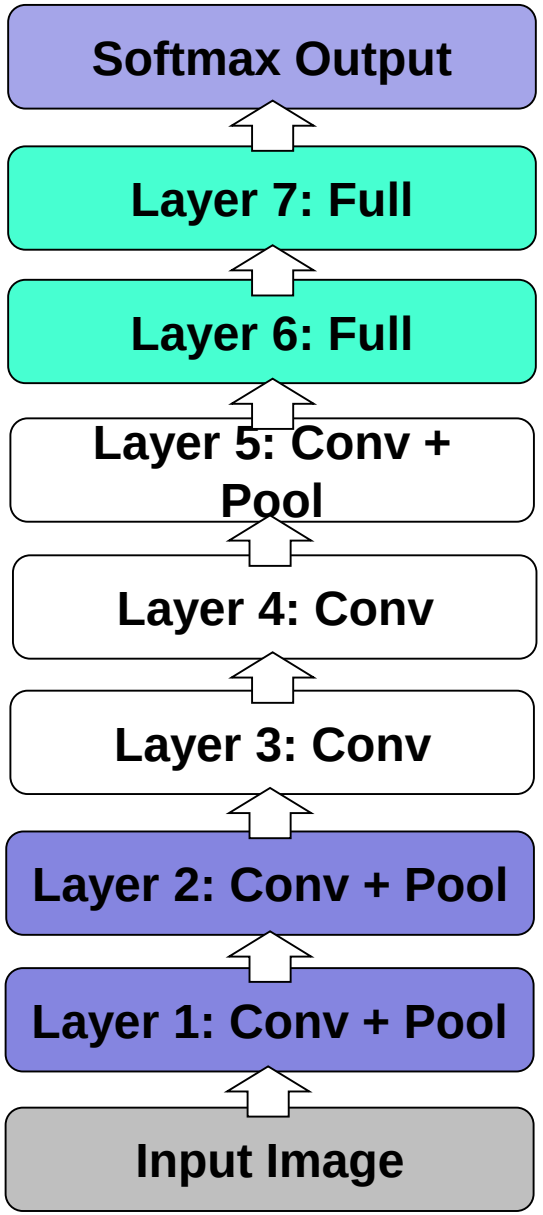


Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

How important is depth?

Architecture of Krizhevsky et al.

- 8 layers total
- Trained on ImageNet
- 18.1% top-5 error



Slide credit: Lana Lazebnik

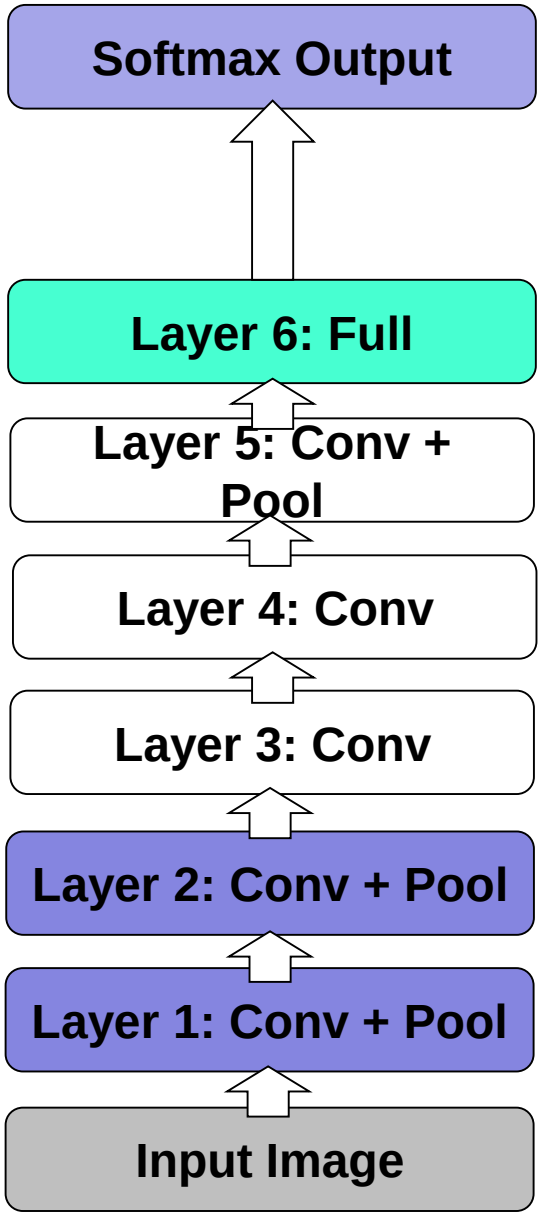
How important is depth?

Remove top fully connected layer

- Layer 7

Drop 16 million parameters

Only 1.1% drop in performance!



Slide credit: Lana Lazebnik

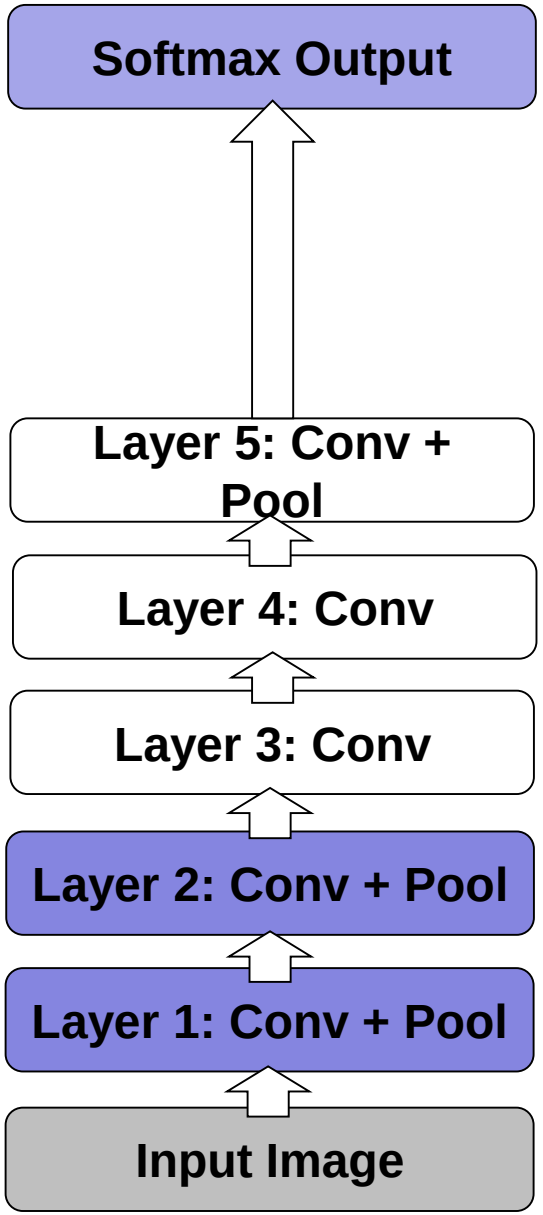
How important is depth?

Remove both fully connected layers

- Layer 6 & 7

Drop ~50 million parameters

5.7% drop in performance



Slide credit: Lana Lazebnik

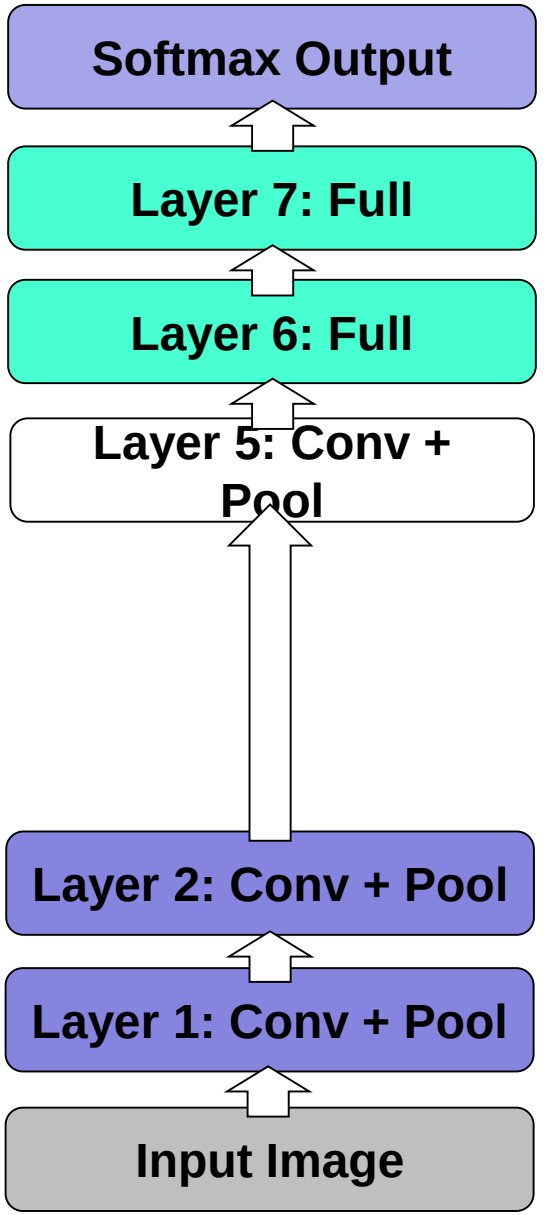
How important is depth?

Now try removing upper feature extractor layers:

- Layers 3 & 4

Drop ~1 million parameters

3.0% drop in performance



Slide credit: Lana Lazebnik

How important is depth?

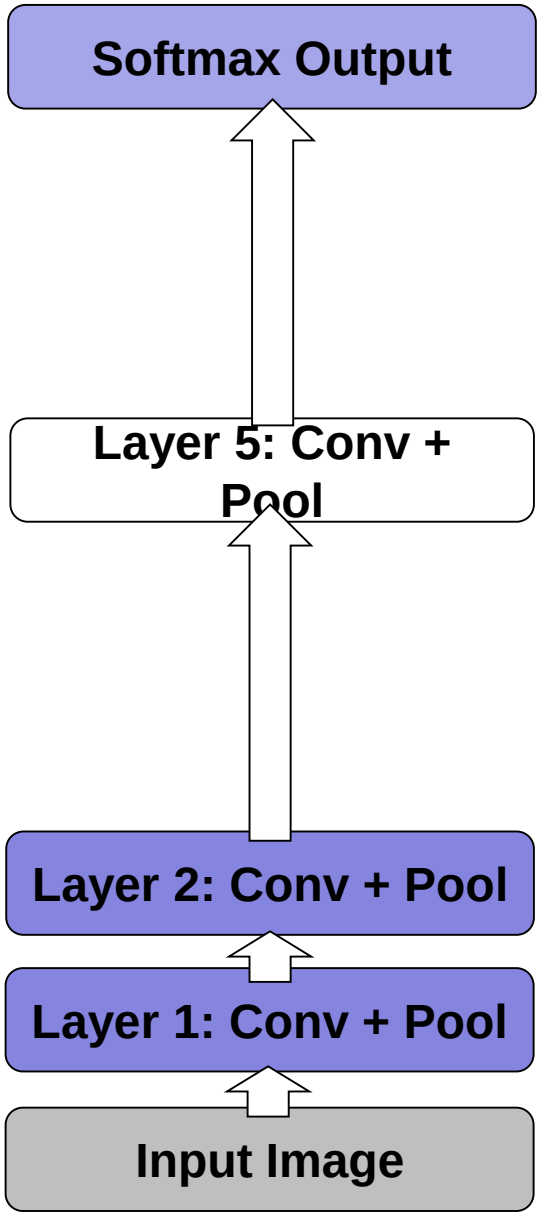
Now try removing upper feature extractor layers & fully connected:

- Layers 3, 4, 6 ,7

Now only 4 layers

33.5% drop in performance

→Depth of network is key



Slide credit: Lana Lazebnik

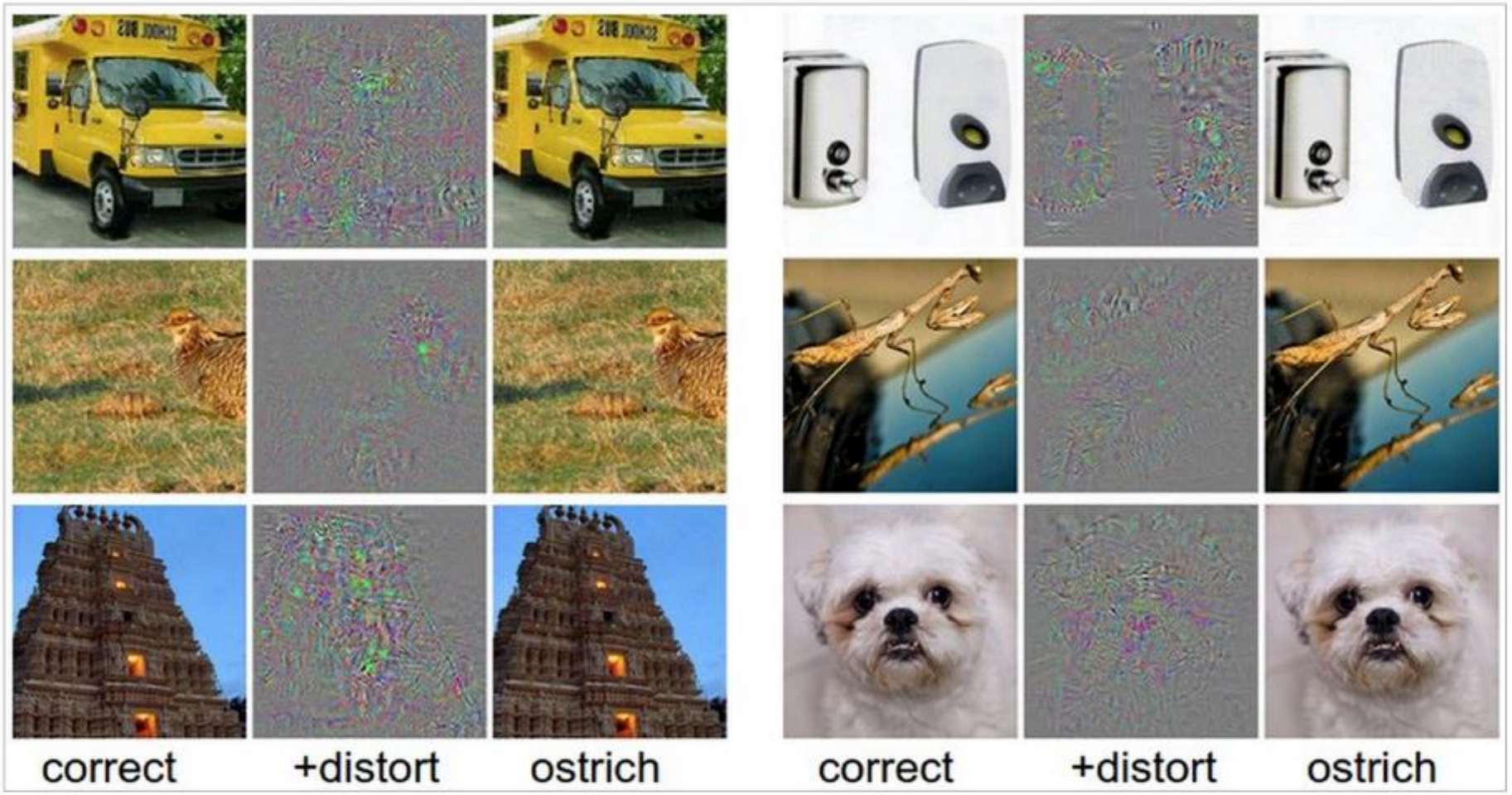
Tapping off features at each Layer

- Plug features from each layer into linear SVM
- Features are neuron activations at that level

	Cal-101 (30/class)	Cal-256 (60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	$\mathbf{86.2 \pm 0.8}$	65.6 ± 0.3
SVM (7)	$\mathbf{85.5 \pm 0.4}$	$\mathbf{71.7 \pm 0.2}$

Adapted from Matt Zeiler

Breaking CNNs

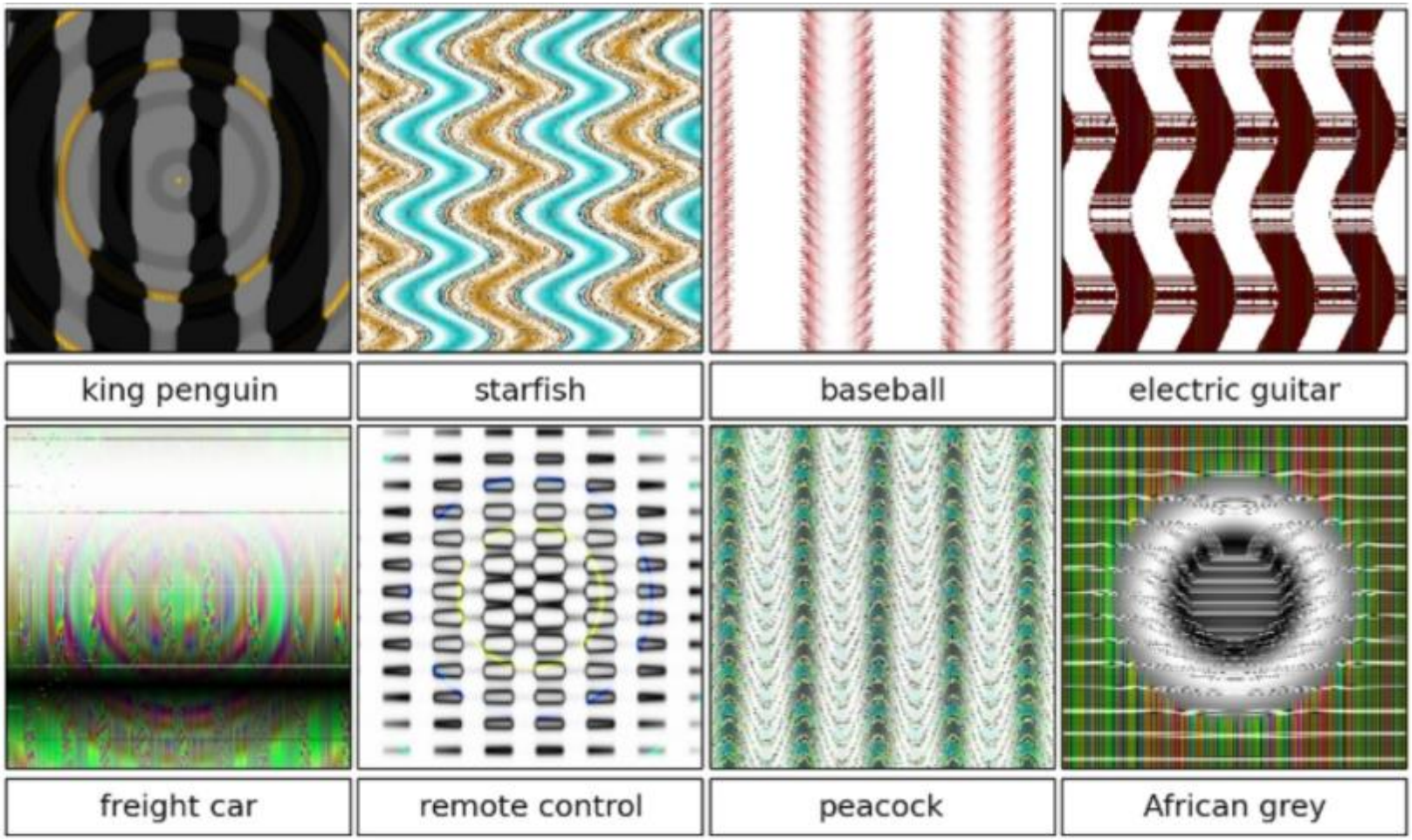


Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

Intriguing properties of neural networks [Szegedy ICLR 2014]

Slide credit: Jia-bin Huang

Breaking CNNs



Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [Nguyen et al. CVPR 2015]

Slide credit: Jia-bin Huang

Convolutional neural network packages

Caffe

cuda-convnet2

Torch

Theano

MatConvNet

Pylearn2

Overfeat