Machine Learning – Lecture 15

Convolutional Neural Networks

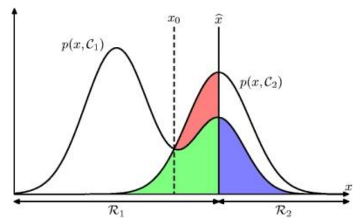
11.12.2017

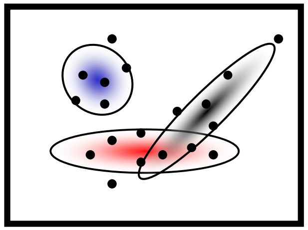
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

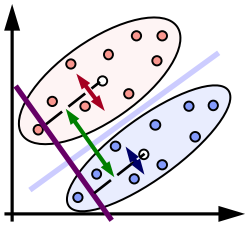
leibe@vision.rwth-aachen.de

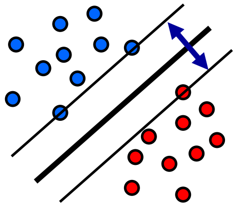
Course Outline

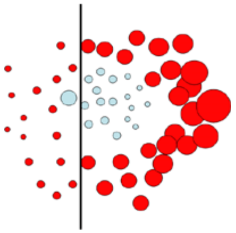
- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Random Forests
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks

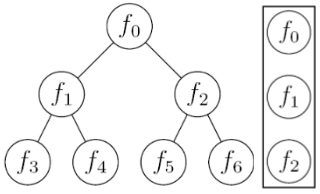


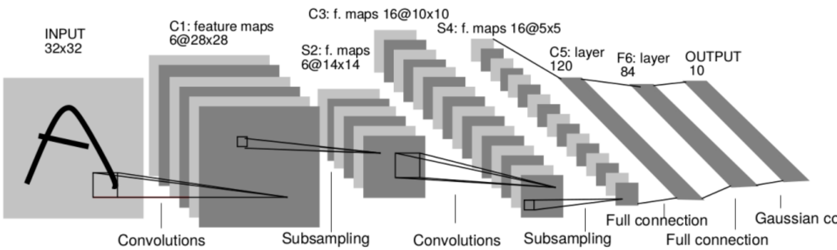










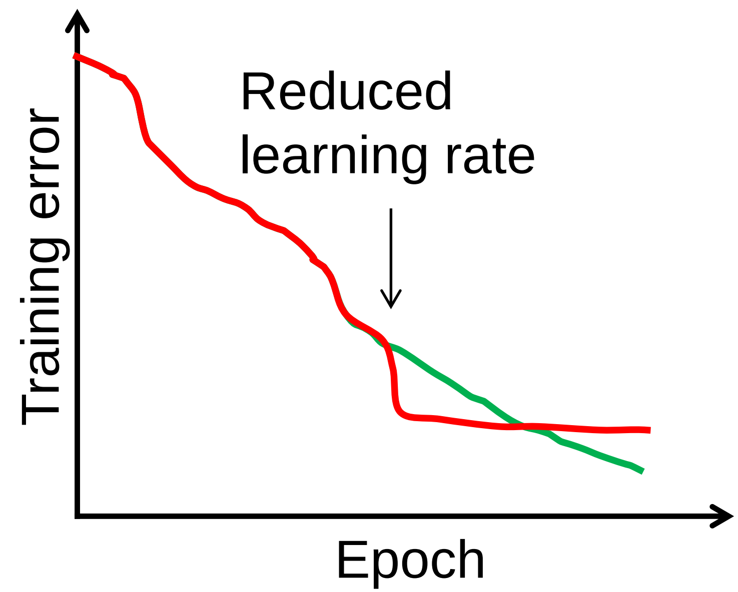


Topics of This Lecture

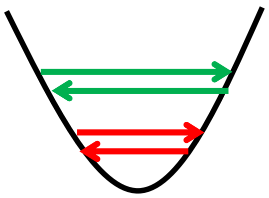
- Recap: Tricks of the Trade
- Nonlinearities
- Initialization
- Advanced techniques
  - Batch Normalization
  - Dropout
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

Recap: Reducing the Learning Rate

- Final improvement step after convergence is reached
  - Reduce learning rate by a factor of 10.
  - Continue training for a few epochs.
  - Do this 1-3 times, then stop training.



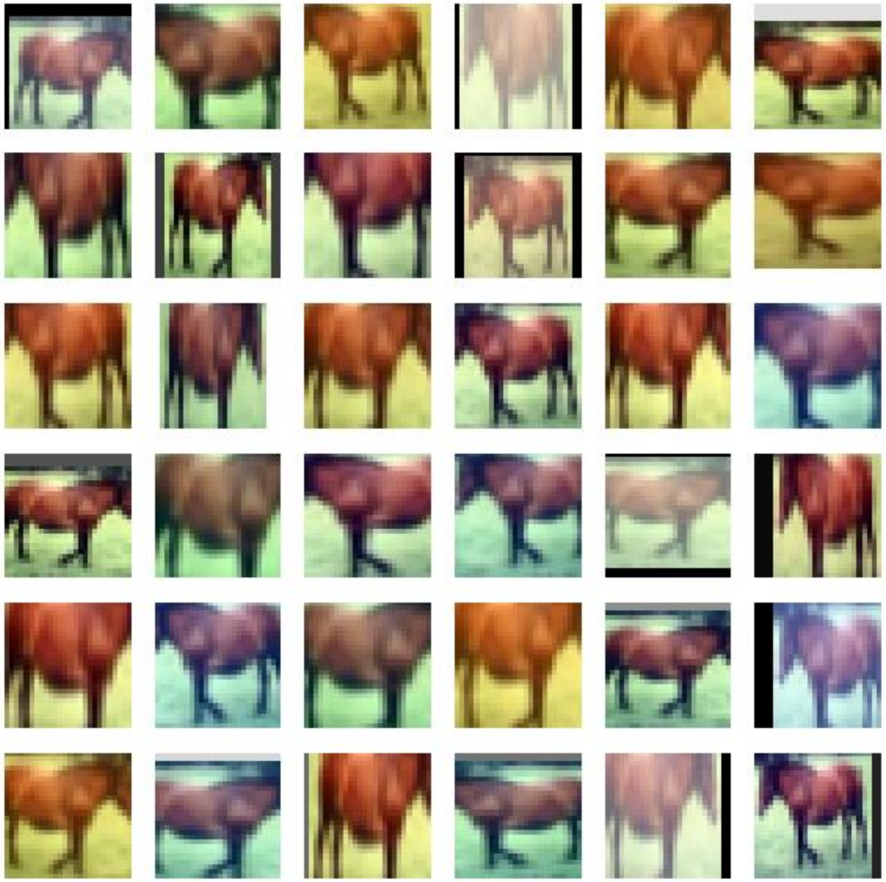
- Effect
  - Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different minibatches.



- *Be careful: Do not turn down the learning rate too soon!*
  - Further progress will be much slower/impossible after that.

Recap: Data Augmentation

- Effect
  - Much larger training set
  - Robustness against expected variations
- During testing
  - When cropping was used during training, need to again apply crops to get same image size.
  - Beneficial to also apply flipping during test.
  - Applying several ColorPCA variations can bring another ~1% improvement, but at a significantly increased runtime.

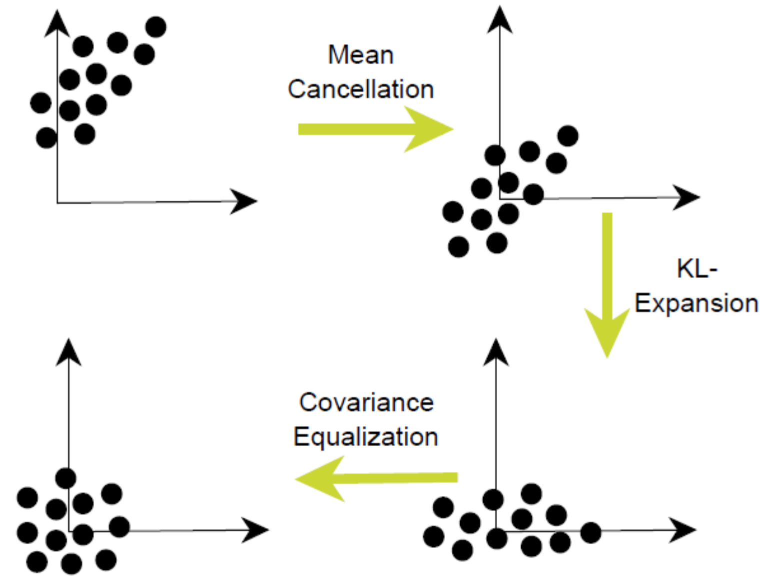


Augmented training data (from one original image)

Image source: Lucas Beyer

Recap: Normalizing the Inputs

- Convergence is fastest if
  - The mean of each input variable over the training set is zero.
  - The inputs are scaled such that all have the same covariance.
  - Input variables are uncorrelated if possible.



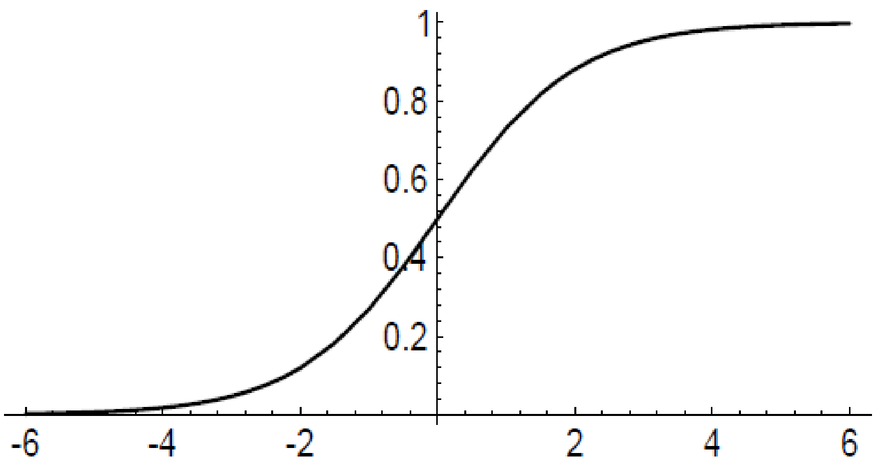
- Advisable normalization steps (for MLPs only, not for CNNs)
  - Normalize all inputs that an input unit sees to zero-mean, unit covariance.
  - If possible, try to decorrelate them using PCA (also known as Karhunen-Loeve expansion).

Image source: Yann LeCun et al., Efficient BackProp (1998)

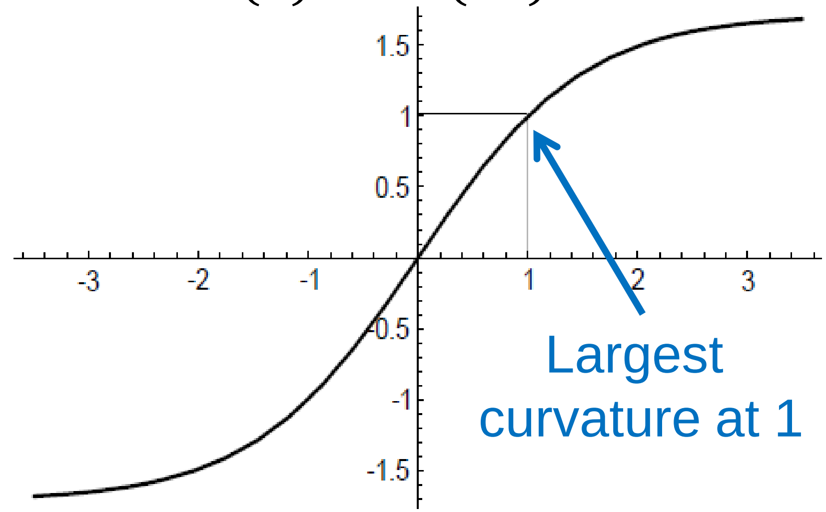
Topics of This Lecture

- Recap: Tricks of the Trade
- Nonlinearities
- Initialization
- Advanced techniques
  - Batch Normalization
  - Dropout
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

Choosing the Right Sigmoid



$\tanh(a) = 2\sigma(2a) - 1$



- Normalization is also important for intermediate layers
  - Symmetric sigmoids, such as tanh, often converge faster than the standard logistic sigmoid.
  - Recommended sigmoid:

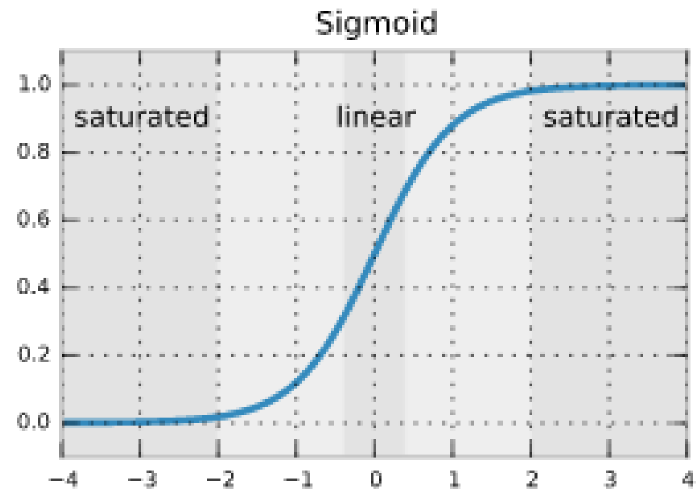
$$f(x) = 1.7159 \tanh\left(\tfrac{2}{3}x\right)$$

$\Rightarrow$ When used with normalized inputs, the variance of the outputs will be close to 1.

Image source: Yann LeCun et al., Efficient BackProp (1998)

Effect of Sigmoid Nonlinearities

- Effects of sigmoid/tanh function
  - Linear behavior around 0
  - Saturation for large inputs



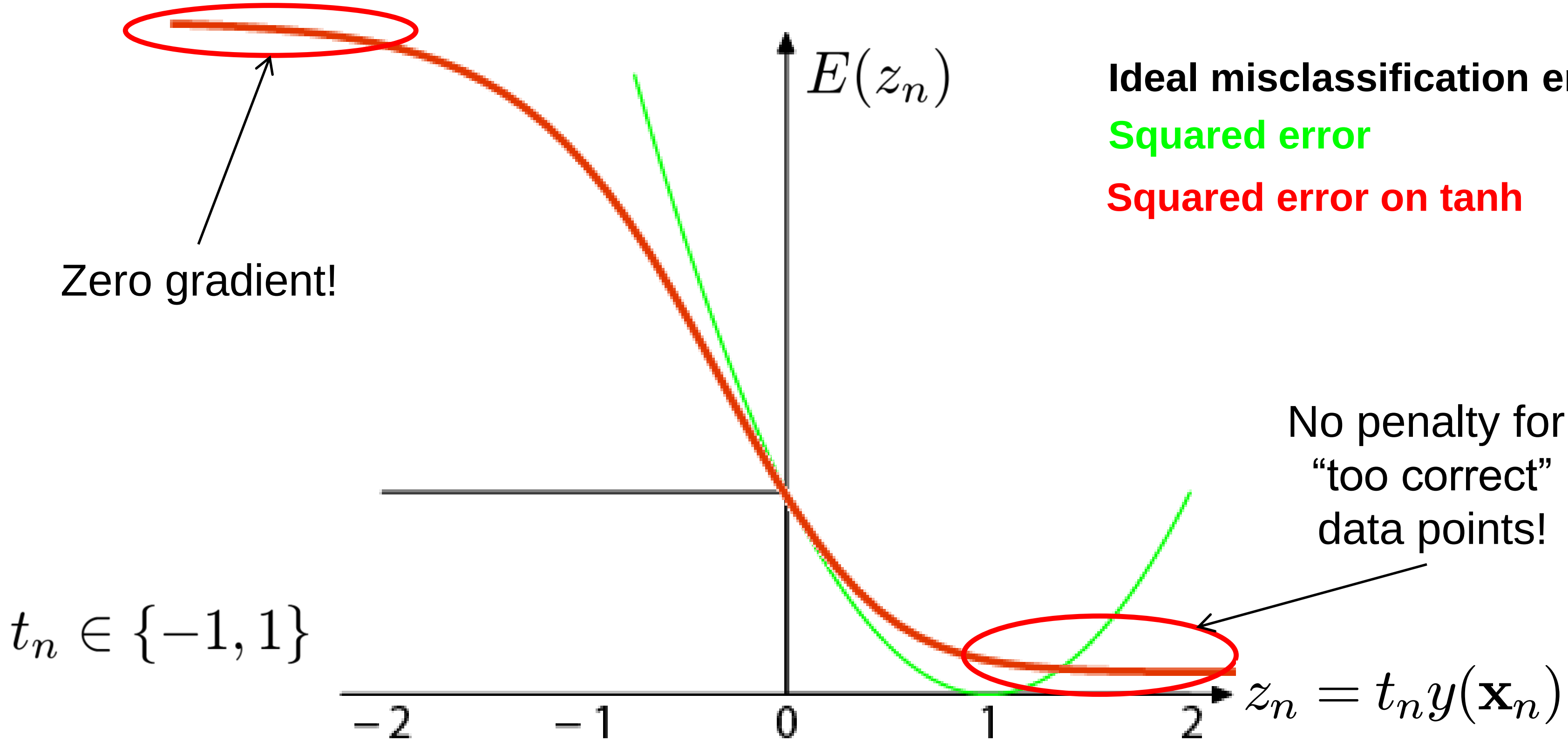
- If all parameters are too small
  - Variance of activations will drop in each layer
  - Sigmoids are approximately linear close to 0
  - Good for passing gradients through, but...
  - Gradual loss of the nonlinearity

  $\Rightarrow$ No benefit of having multiple layers

- If activations become larger and larger
  - They will saturate and gradient will become zero

Image source: http://deepdish.io/2015/02/24/network-initialization/

Another Note on Error Functions



- Squared error on sigmoid/tanh output function
  - Avoids penalizing "too correct" data points.
  - But: zero gradient for confidently incorrect classifications!

$\Rightarrow$ Do not use L_2 loss with sigmoid outputs (instead: cross-entropy)!

Image source: Bishop, 2006

Usage

- Output nodes
  - Typically, a sigmoid or tanh function is used here.
    - Sigmoid for probabilistic classification (2-class case).
    - Softmax for multi-class classification
    - tanh for regression tasks
- Internal nodes
  - Historically, tanh was most often used.
  - tanh is better than sigmoid for internal nodes, since it is already centered.
  - Internally, tanh is often implemented as piecewise linear function.
  - More recently: ReLU often used for classification tasks.

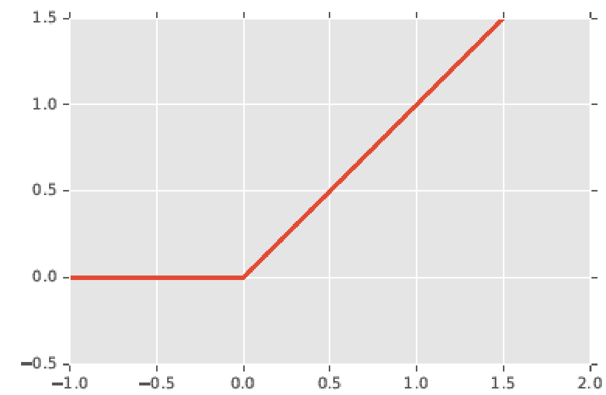
Extension: ReLU

- An improvement for learning deep models
  - Use Rectified Linear Units (ReLU)

$$g(a) = \max\{0, a\}$$

  - Effect: gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$



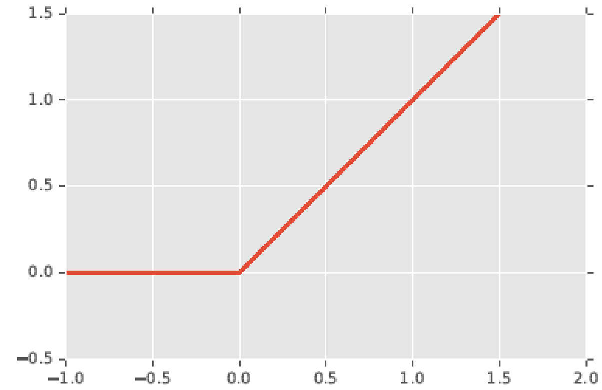
- Advantages
  - Much easier to propagate gradients through deep networks.
  - We do not need to store the ReLU output separately
    - Reduction of the required memory by half compared to tanh!

$\Rightarrow$ *ReLU has become the de-facto standard for deep networks.*

Extension: ReLU

- An improvement for learning deep models
  - Use Rectified Linear Units (ReLU)

$$g(a) = \max\{0, a\}$$



  - Effect: gradient is propagated with a constant factor

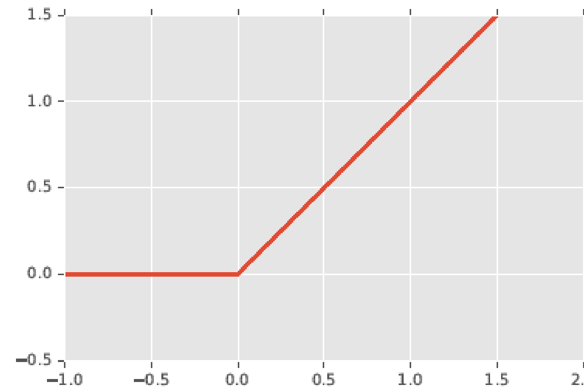
$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$

- Disadvantages / Limitations
  - A certain fraction of units will remain "stuck at zero".
    - If the initial weights are chosen such that the ReLU output is 0 for the entire training set, the unit will never pass through a gradient to change those weights.
  - ReLU has an offset bias, since its outputs will always be positive

Further Extensions

- Rectified linear unit (ReLU)

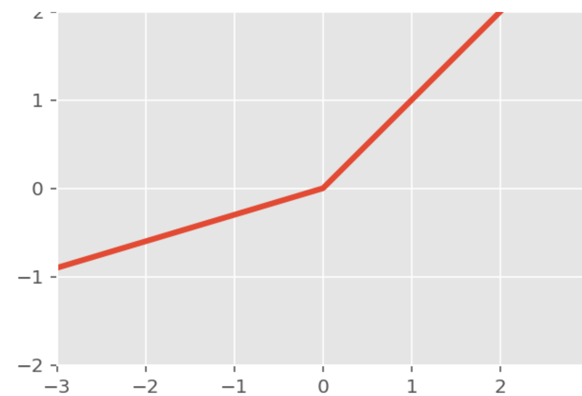
$$g(a) = \max\{0, a\}$$



- Leaky ReLU

$$g(a) = \max\{\beta a, a\} \qquad \beta \in [0.01\,, 0.3]$$

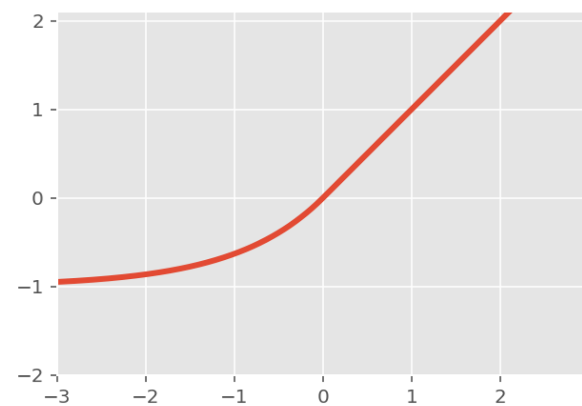
  - Avoids stuck-at-zero units
  - Weaker offset bias



- ELU

$$g(a) = \begin{cases} a, & a \geq 0 \\ e^{a} - 1, & a < 0 \end{cases}$$

  - No offset bias anymore
  - BUT: need to store activations



Topics of This Lecture

- Recap: Tricks of the Trade
- Nonlinearities
- Initialization
- Advanced techniques
  - Batch Normalization
  - Dropout
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

Initializing the Weights

- Motivation
  - The starting values of the weights can have a significant effect on the training process.
  - Weights should be chosen randomly, but in a way that the sigmoid is primarily activated in its linear region.

- Guideline (from [LeCun et al., 1998] book chapter)
  - Assuming that
    - The training set has been normalized
    - The recommended sigmoid $f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$ is used

  the initial weights should be randomly drawn from a distribution (e.g., uniform or Normal) with mean zero and variance

$$\sigma_w^2 = \frac{1}{n_{in}}$$

  where n_{in} is the fan-in (#connections into the node).

Historical Sidenote

- Apparently, this guideline was either little known or misunderstood for a long time
  - A popular heuristic (also the standard in Torch) was to use
    $$W \sim U\left[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}\right]$$
  - This looks almost like LeCun's rule. However…

- When sampling weights from a uniform distribution $[a,b]$
  - Keep in mind that the standard deviation is computed as
    $$\sigma^2 = \frac{1}{12}(b-a)^2$$
  - If we do that for the above formula, we obtain
    $$\sigma^2 = \frac{1}{12}\left(\frac{2}{\sqrt{n_{in}}}\right)^2 = \frac{1}{3}\frac{1}{n_{in}}$$

  $\Rightarrow$ Activations & gradients will be attenuated with each layer! (bad)

Glorot Initialization

- Breakthrough results
  - In 2010, Xavier Glorot published an analysis of what went wrong in the initialization and derived a more general method for automatic initialization.
  - This new initialization massively improved results and made direct learning of deep networks possible overnight.
  - Let's look at his analysis in more detail...

X. Glorot, Y. Bengio, Understanding the Difficulty of Training Deep Feedforward Neural Networks, AISTATS 2010.

Analysis

- Variance of neuron activations
  - Suppose we have an input X with n components and a linear neuron with random weights W that spits out a number Y.
  - What is the variance of Y?

$$Y = W_1X_1 + W_2X_2 + \cdots + W_nX_n$$

  - If inputs and outputs have both mean 0, the variance is

$$Var(W_iX_i) = E[X_i]^2 Var(W_i) + E[W_i]^2 Var(X_i) + Var(W_i)Var(X_i)$$

$$= Var(W_i)Var(X_i)$$

  - If the X_i and W_i are all i.i.d, then

$$Var(Y) = Var(W_1X_1 + W_2X_2 + \cdots + W_nX_n) = \mathrm{nVar}(\mathrm{W_i})\mathrm{Var}(\mathrm{X_i})$$

$\Rightarrow$ The variance of the output is the variance of the input, but scaled by $n\ \mathrm{Var}(W_i)$.

Analysis (cont'd)

- Variance of neuron activations
  - if we *want the variance of the input and output of a unit to be the same*, then $n \operatorname{Var}(W_i)$ should be 1. This means

$$\operatorname{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

  - If we do the same for the backpropagated gradient, we get

$$\operatorname{Var}(W_i) = \frac{1}{n_{\text{out}}}$$

  - As a compromise, Glorot & Bengio proposed to use

$$\operatorname{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

$\Rightarrow$ Randomly sample the weights with this variance. That's it.

Sidenote

- When sampling weights from a uniform distribution $[a,b]$
  - Again keep in mind that the standard deviation is computed as
$$\sigma^2 = \frac{1}{12}(b-a)^2$$
  - Glorot initialization with uniform distribution
$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$
  - Or when only taking into account the fan-in
$$W \sim U\left[-\frac{\sqrt{3}}{\sqrt{n_{in}}}, \frac{\sqrt{3}}{\sqrt{n_{in}}}\right]$$
  - *If this had been implemented correctly in Torch from the beginning, the Deep Learning revolution might have happened a few years earlier…*

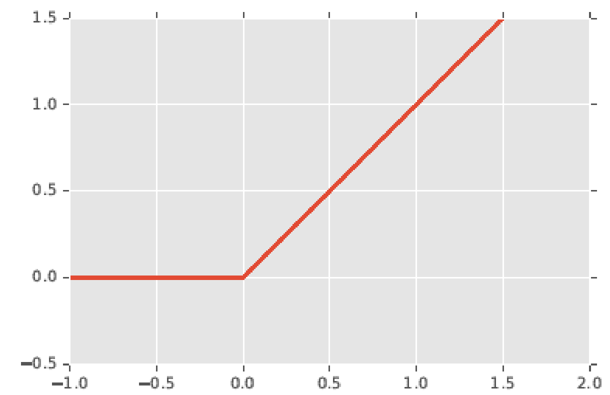
Extension to ReLU

- Important for learning deep models
  - Rectified Linear Units (ReLU)

$$g(a) = \max\{0, a\}$$

  - Effect: gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$



- We can also improve them with proper initialization
  - However, the Glorot derivation was based on tanh units, linearity assumption around zero does not hold for ReLU.
  - He et al. made the derivations, derived to use instead

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}$$

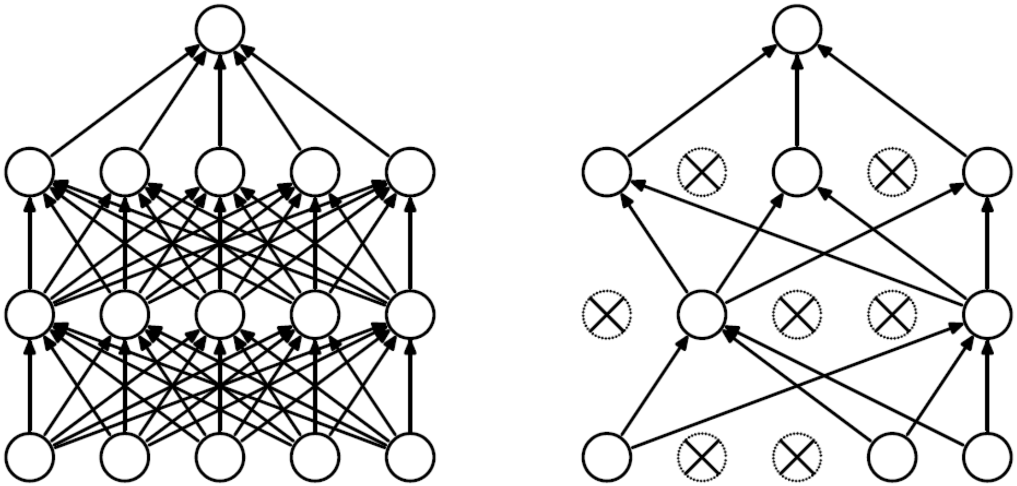
Topics of This Lecture

- Recap: Tricks of the Trade
- Nonlinearities
- Initialization
- Advanced techniques
  - Batch Normalization
  - Dropout
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

Batch Normalization [Ioffe & Szegedy '14]

- Motivation
  - Optimization works best if all inputs of a layer are normalized.
- Idea
  - Introduce intermediate layer that centers the activations of the previous layer per minibatch.
  - I.e., perform transformations on all activations and undo those transformations when backpropagating gradients
  - Complication: centering + normalization also needs to be done at test time, but minibatches are no longer available at that point.
    - Learn the normalization parameters to compensate for the expected bias of the previous layer (usually a simple moving average)
- Effect
  - Much improved convergence (but parameter values are important!)
  - Widely used in practice

Dropout [Srivastava, Hinton '12]



(a) Standard Neural Net (b) After applying dropout.

- Idea
  - Randomly switch off units during training.
  - Change network architecture for each data point, effectively training many different variants of the network.
  - When applying the trained network, multiply activations with the probability that the unit was set to zero.

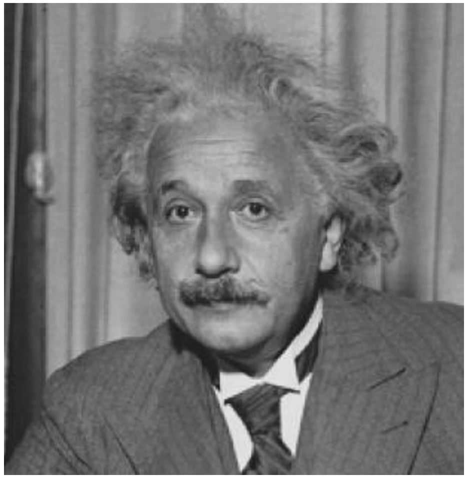
  ⇒ Greatly improved performance

Topics of This Lecture

- Recap: Tricks of the Trade
- Nonlinearities
- Initialization
- Advanced techniques
  - Batch Normalization
  - Dropout
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

Neural Networks for Computer Vision

- How should we approach vision problems?



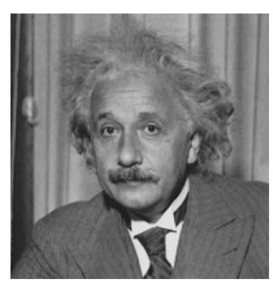
→ Face Y/N?

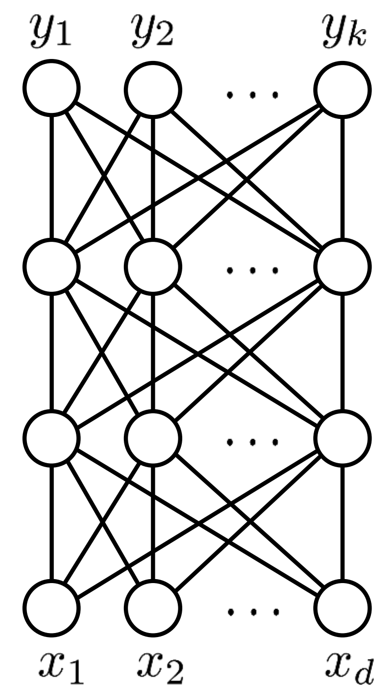
- Architectural considerations
  - Input is 2D ⇒ 2D layers of units
  - No pre-segmentation ⇒ Need robustness to misalignments
  - Vision is hierarchical ⇒ Hierarchical multi-layered structure
  - Vision is difficult ⇒ Network should be deep

Why Hierarchical Multi-Layered Models?

- Motivation 1: Visual scenes are hierarchically organized

Object	Face
↑	↑
Object parts	Eyes, nose, ...
↑	↑
Primitive features	Oriented edges
↑	↑
Input image	Face image

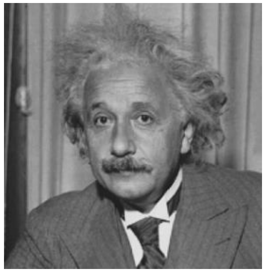




Why Hierarchical Multi-Layered Models?

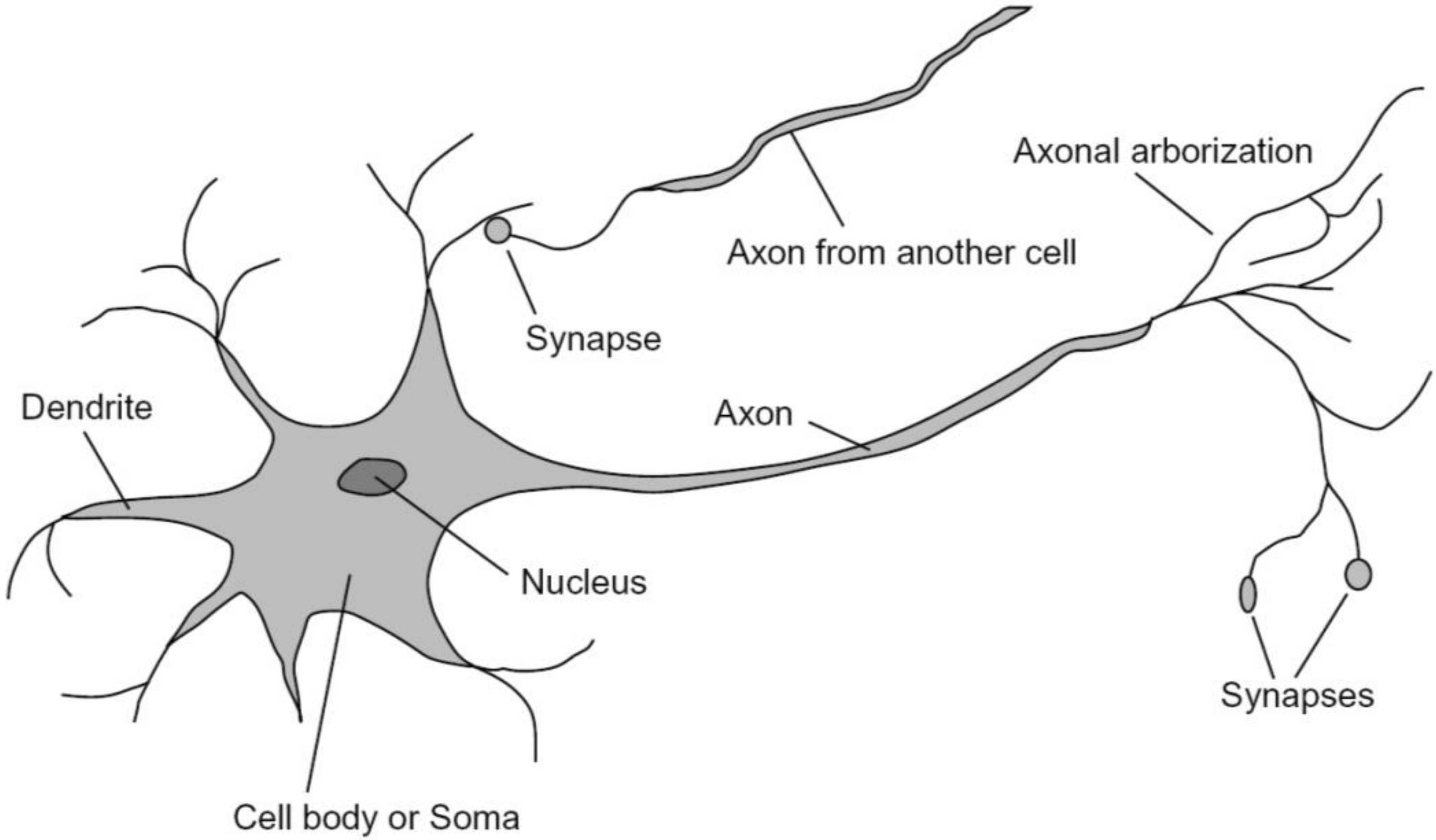
- Motivation 2: *Biological vision* is hierarchical, too

Object	Face	Inferotemporal cortex
↑	↑	
Object parts	Eyes, nose, ...	V4: different textures
↑	↑	
Primitive features	Oriented edges	V1: simple and complex cells
↑	↑	
Input image	Face image	Photoreceptors, retina



Slide adapted from Richard Turner

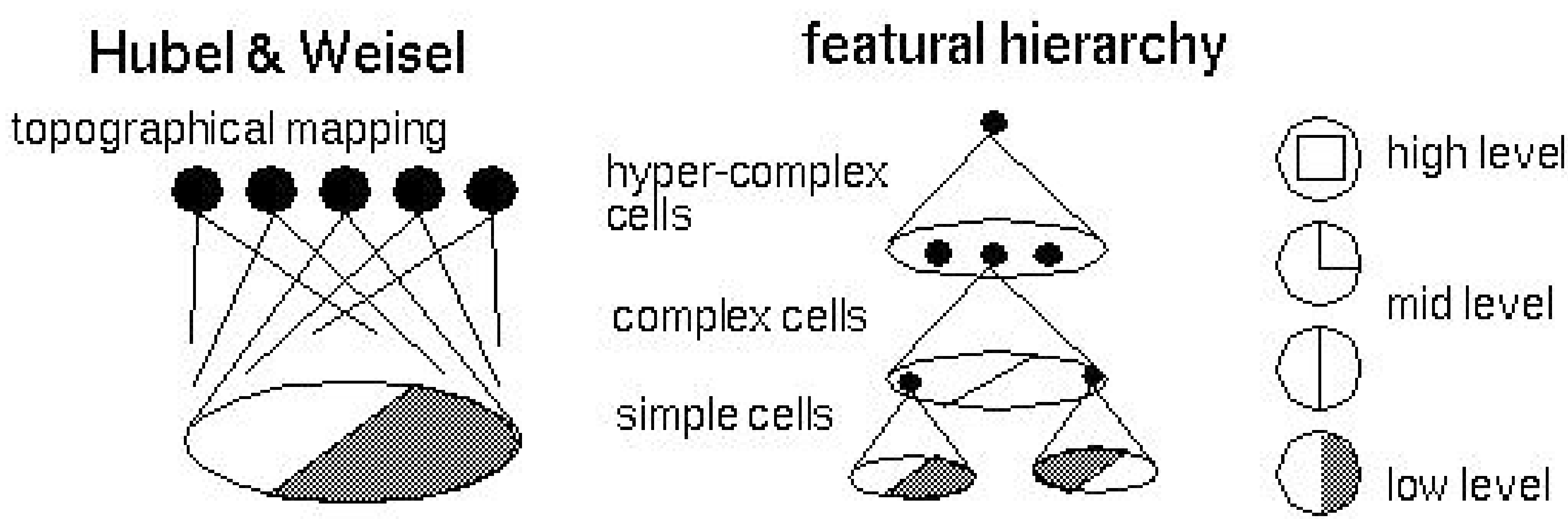
Inspiration: Neuron Cells



Slide credit: Svetlana Lazebnik, Rob Fergus

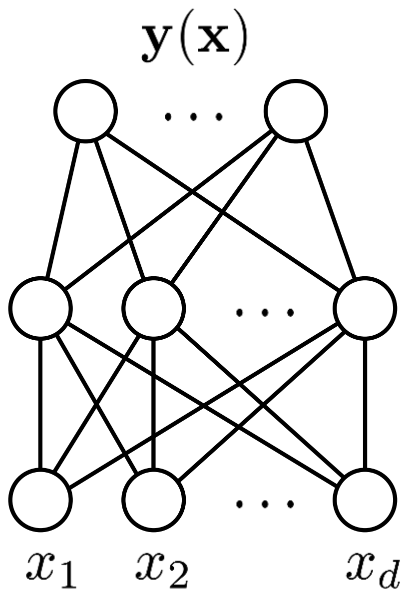
Hubel/Wiesel Architecture

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

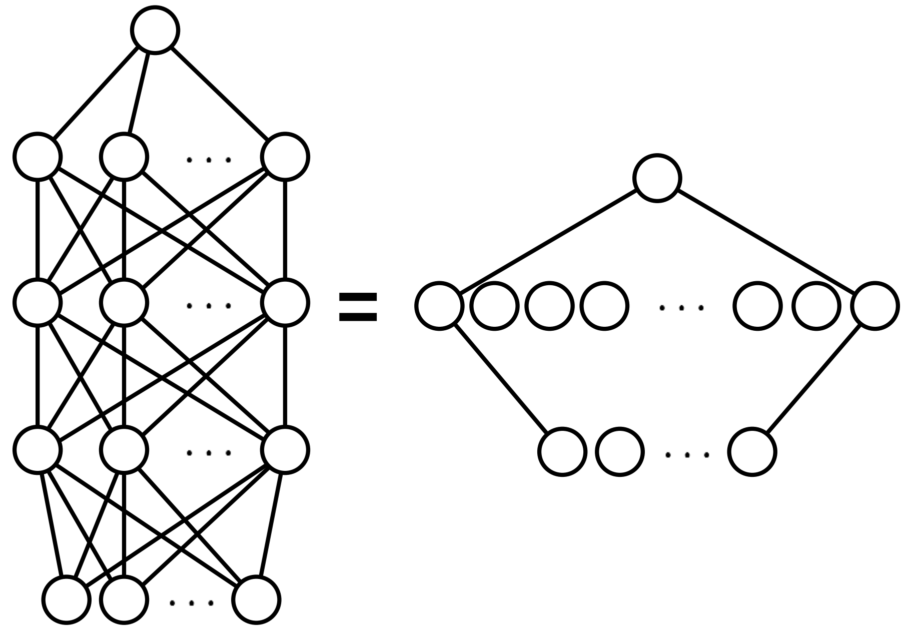


Why Hierarchical Multi-Layered Models?

- Motivation 3: Shallow architectures are inefficient at representing complex functions



An MLP with 1 hidden layer can implement *any* function (universal approximator)



However, if the function is deep, a very large hidden layer may be required.

RWTH AACHEN UNIVERSITY

What's Wrong With Standard Neural Networks?

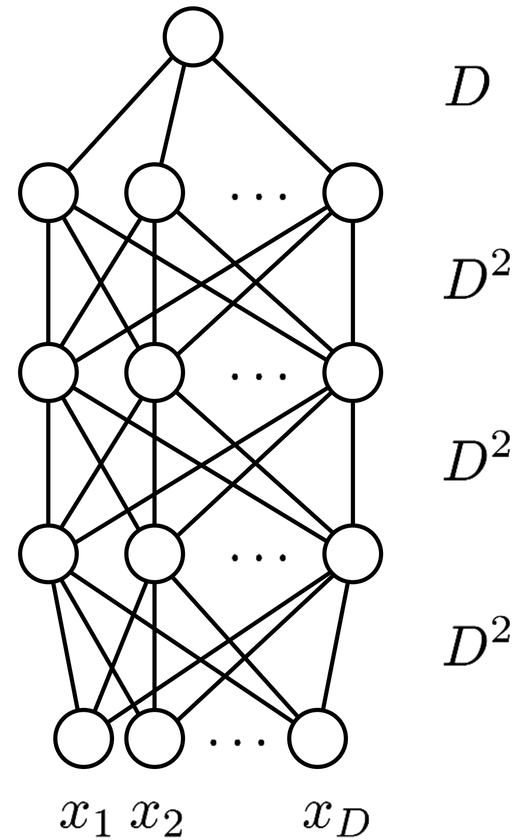
- Complexity analysis
  - How many parameters does this network have?

$$|\theta| = 3D^2 + D$$

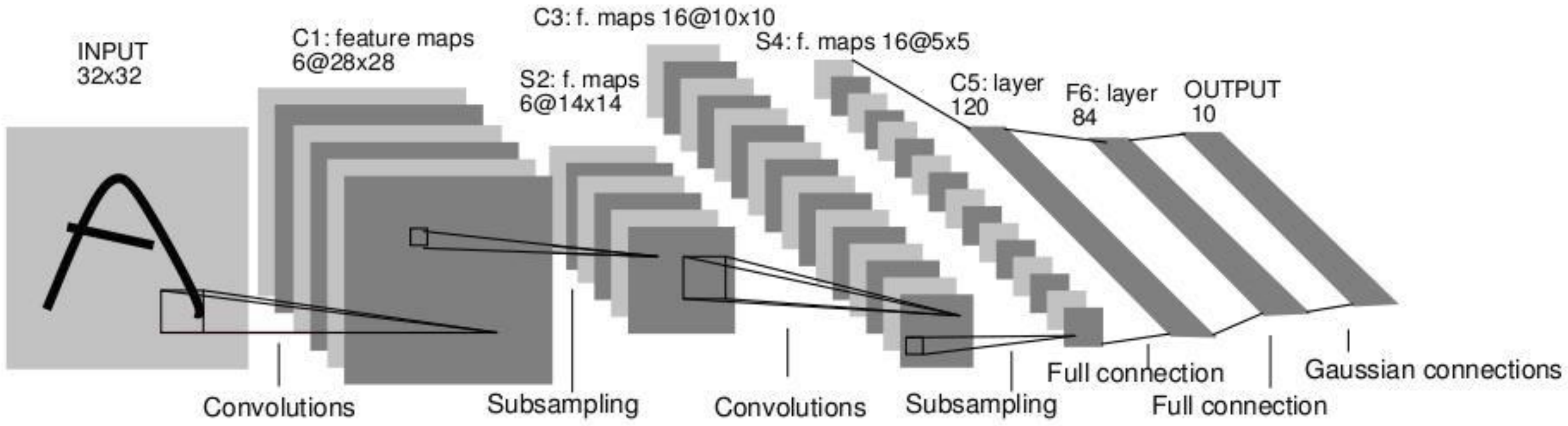
  - For a small 32×32 image

$$|\theta| = 3 \cdot 32^4 + 32^2 \approx 3 \cdot 10^6$$

- Consequences
  - Hard to train
  - Need to initialize carefully
  - *Convolutional nets reduce the number of parameters!*



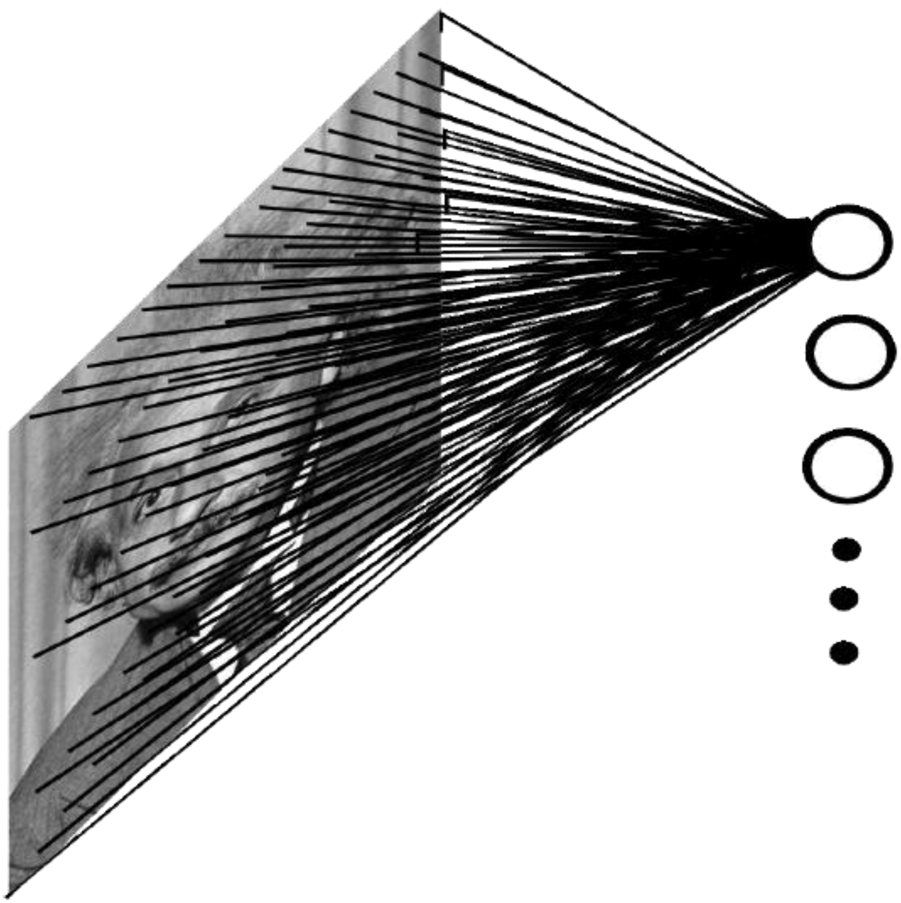
Convolutional Neural Networks (CNN, ConvNet)



- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant 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.

Convolutional Networks: Intuition



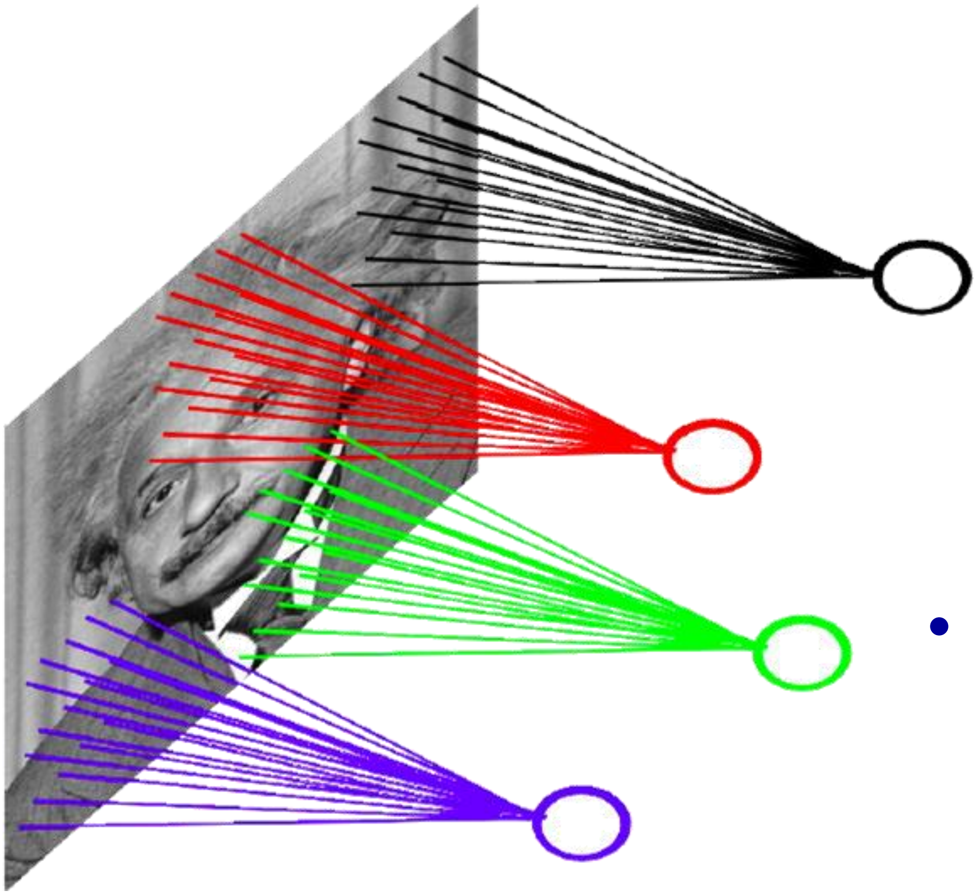
- Fully connected network
  - E.g. 1000×1000 image
    1M hidden units

  ⇒ 1T parameters!

- Ideas to improve this
  - Spatial correlation is local

Image source: Yann LeCun

Convolutional Networks: Intuition



- Locally connected net
  - E.g. 1000×1000 image
    1M hidden units
    10×10 receptive fields

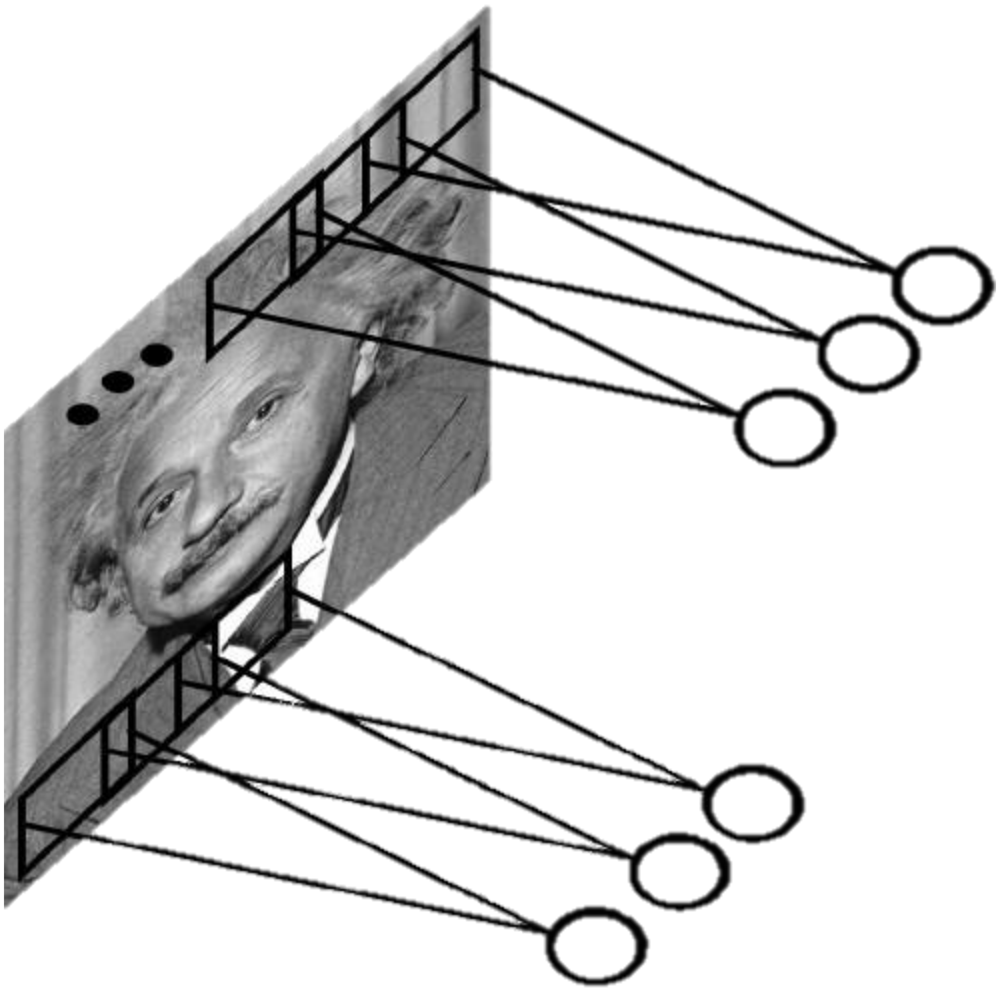
  ⇒ 100M parameters!

- Ideas to improve this
  - Spatial correlation is local
  - Want translation invariance

Slide adapted from Marc'Aurelio Ranzato

Image source: Yann LeCun

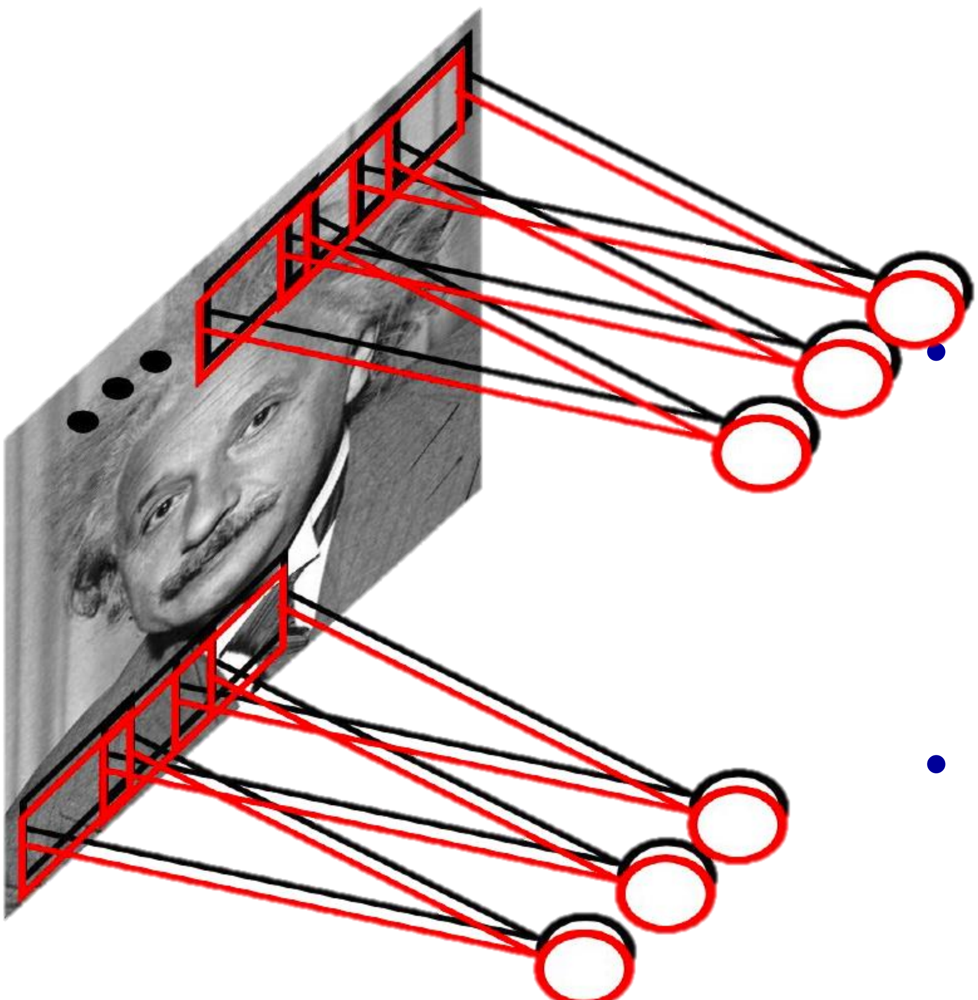
Convolutional Networks: Intuition



- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels

Image source: Yann LeCun

Convolutional Networks: Intuition



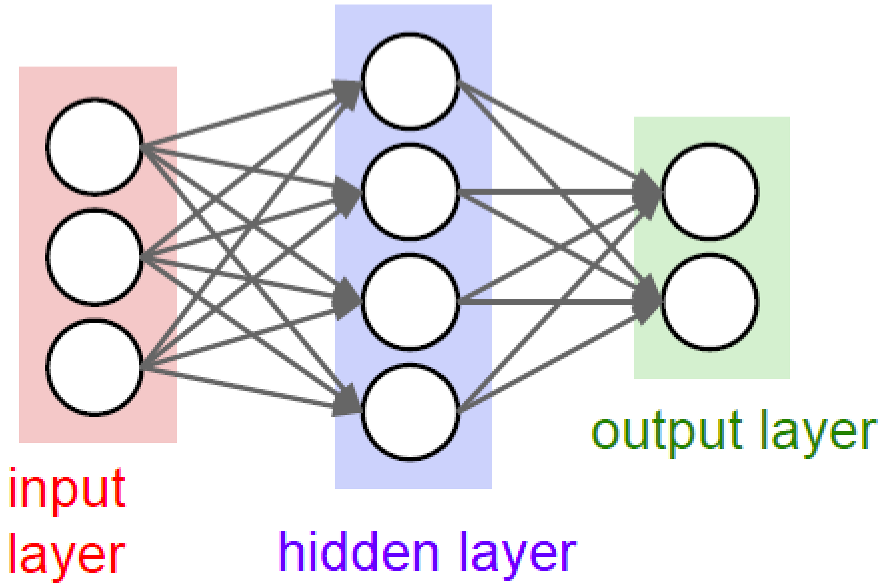
- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels
- Learn *multiple* filters
  - E.g. 1000×1000 image
    100 filters
    10×10 filter size

  $\Rightarrow$ 10k parameters
- Result: Response map
  - size: $1000 \times 1000 \times 100$
  - Only memory, not params!

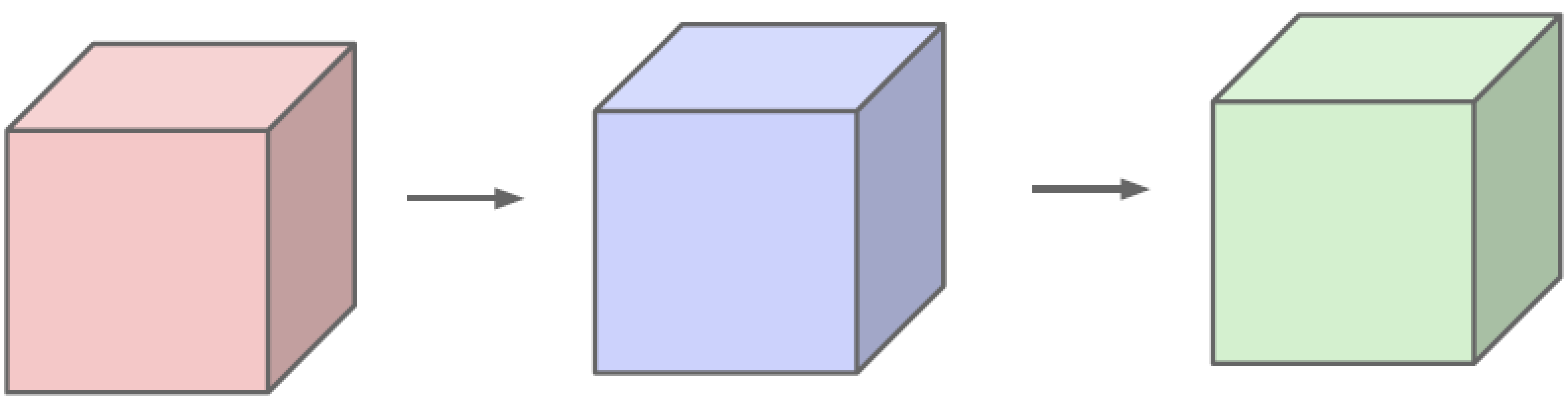
Image source: Yann LeCun

Important Conceptual Shift

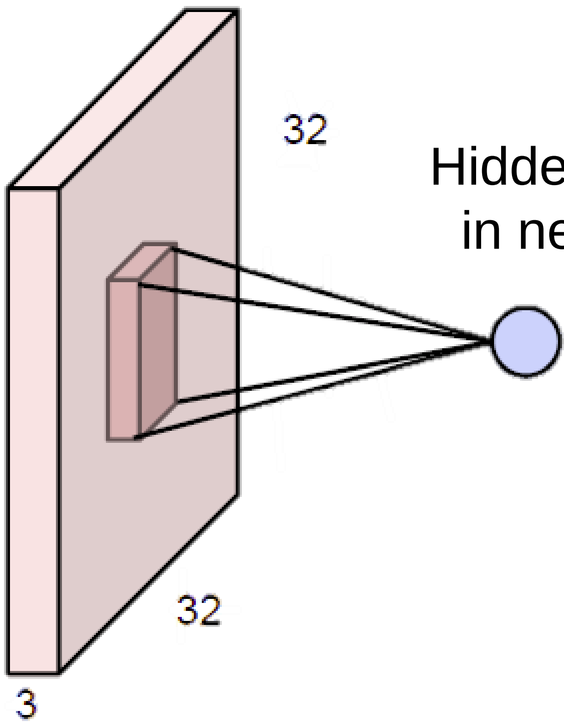
- Before



- Now:



Convolution Layers



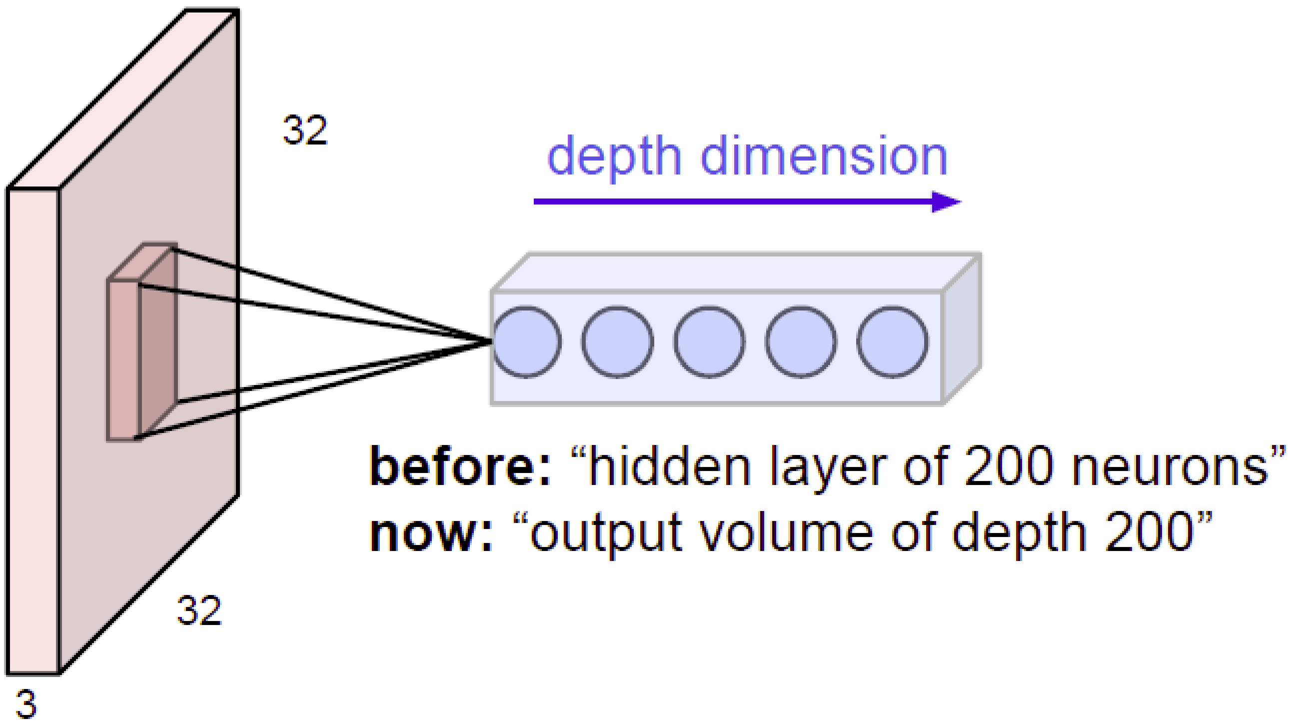
Example
image: $32\times32\times3$ volume

Before: Full connectivity
$32\times32\times3$ weights

Now: Local connectivity
One neuron connects to, e.g., $5\times5\times3$ region.
$\Rightarrow$ Only $5\times5\times3$ shared weights.

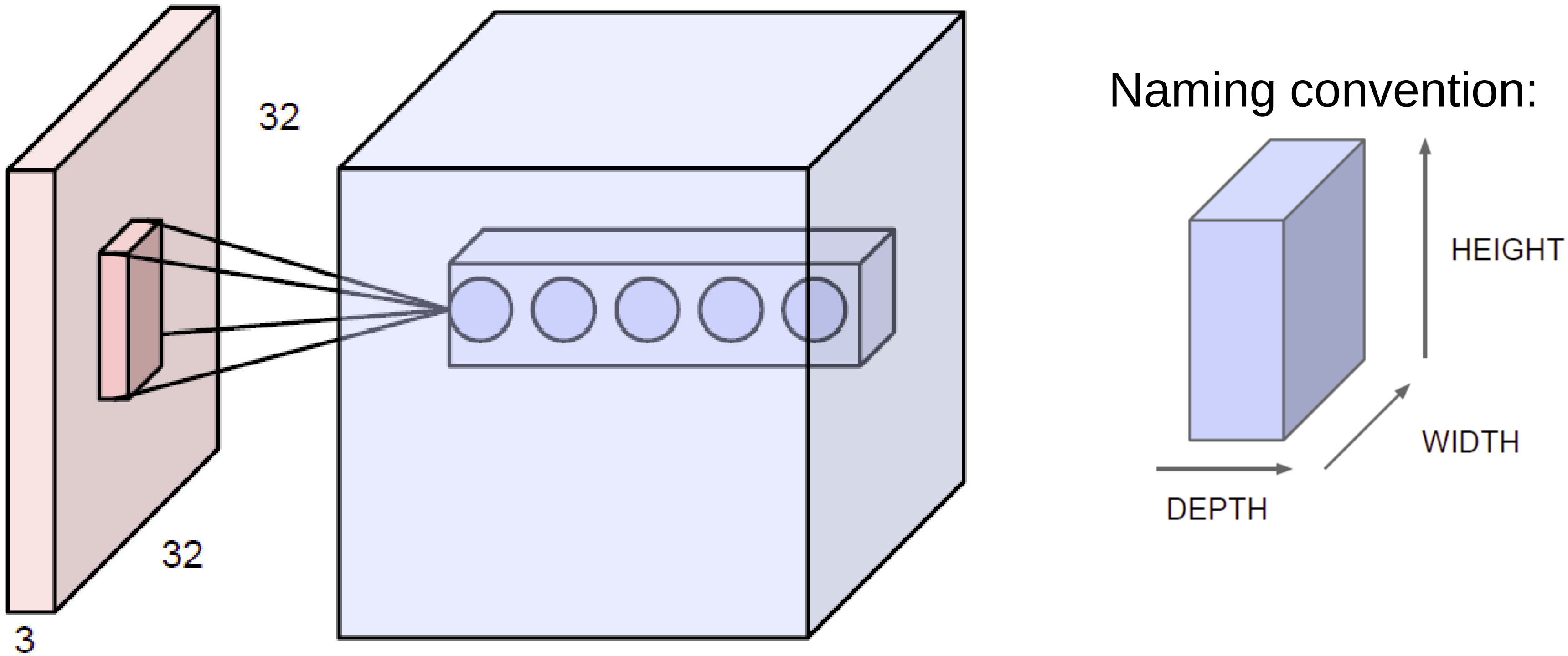
- Note: Connectivity is
  - Local in space (5×5 inside 32×32)
  - But full in depth (all 3 depth channels)

Convolution Layers



- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth

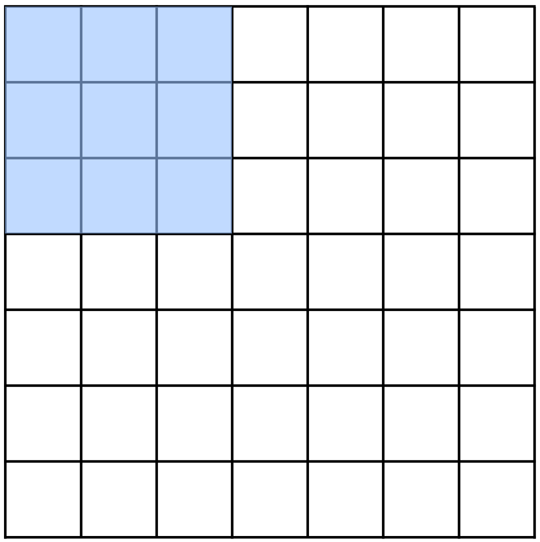
Convolution Layers



- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth
  - Form a single [$1\times1\times$depth] depth column in output volume.

Slide credit: FeiFei Li, Andrej Karpathy

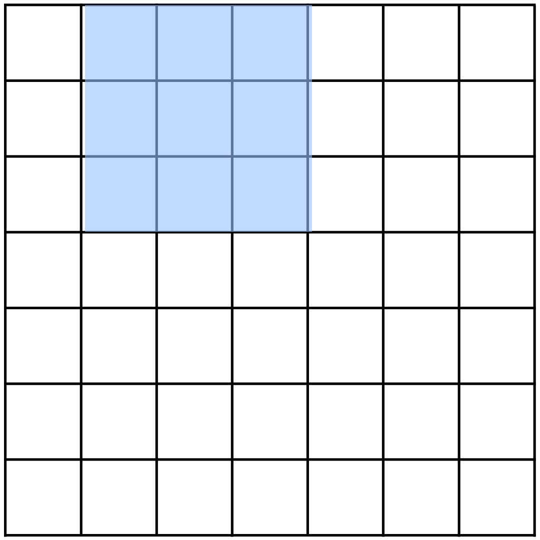
Convolution Layers



Example:
7×7 input
assume 3×3 connectivity
stride 1

- Replicate this column of hidden neurons across space, with some stride.

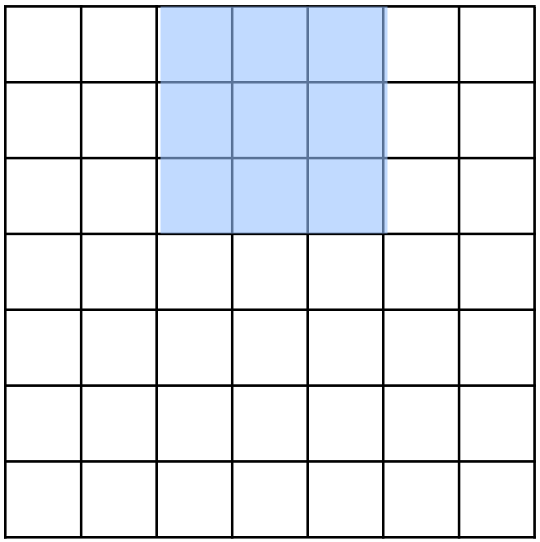
Convolution Layers



Example:
7×7 input
assume 3×3 connectivity
stride 1

- Replicate this column of hidden neurons across space, with some stride.

Convolution Layers

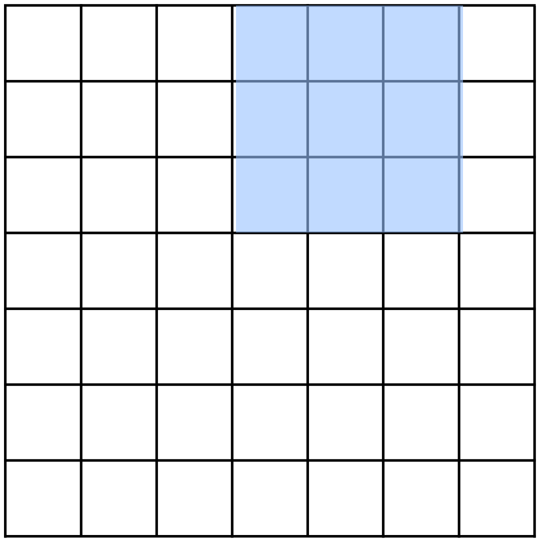


Example:
7×7 input
assume 3×3 connectivity
stride 1

- Replicate this column of hidden neurons across space, with some stride.

Slide credit: FeiFei Li, Andrej Karpathy

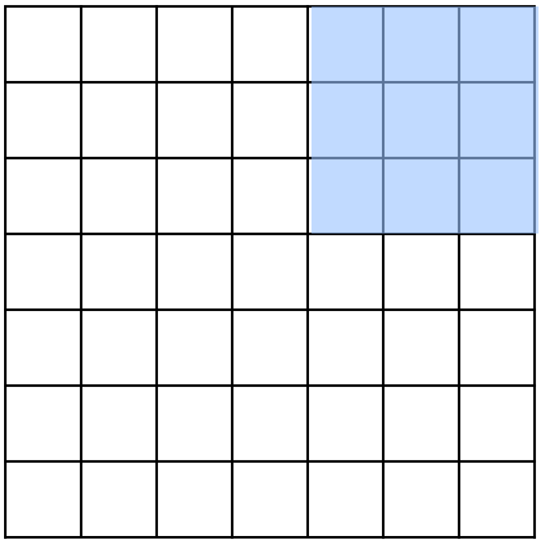
Convolution Layers



Example:
7×7 input
assume 3×3 connectivity
stride 1

- Replicate this column of hidden neurons across space, with some stride.

Convolution Layers

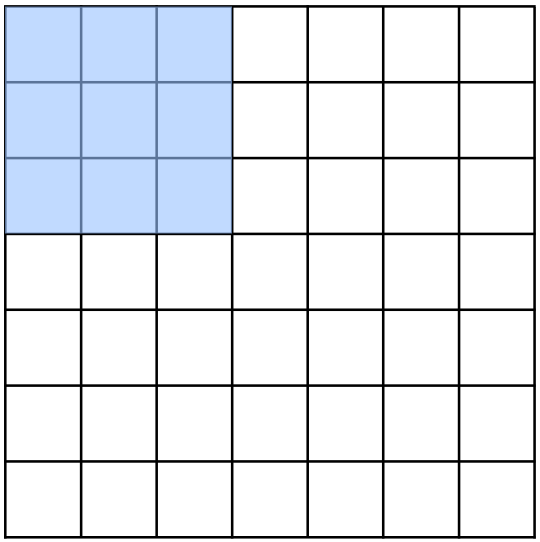


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow 5 \times 5$ output

- Replicate this column of hidden neurons across space, with some stride.

Slide credit: FeiFei Li, Andrej Karpathy

Convolution Layers

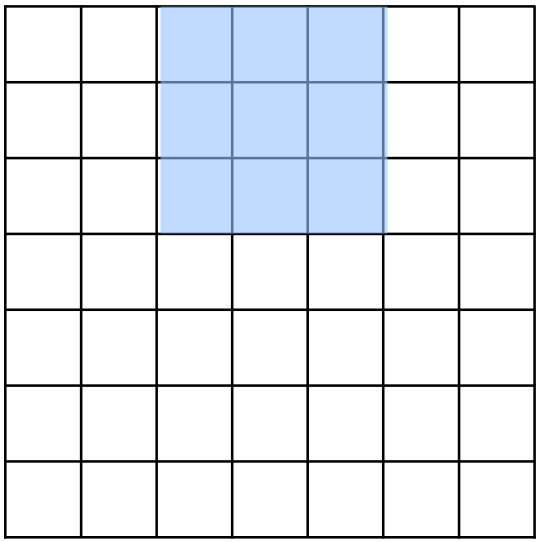


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow$ 5×5 output

What about stride 2?

- Replicate this column of hidden neurons across space, with some stride.

Convolution Layers

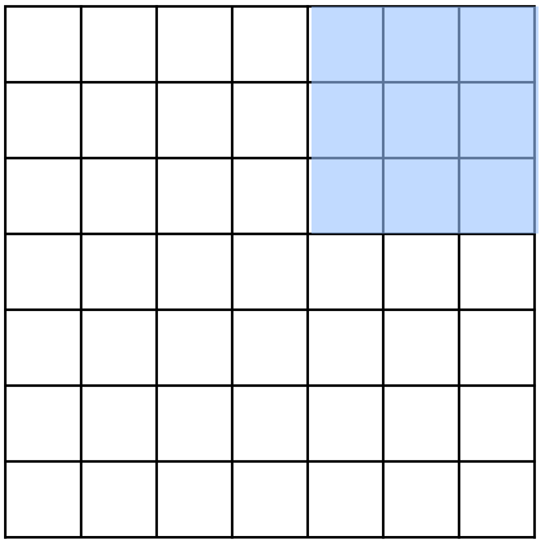


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow 5 \times 5$ output

What about stride 2?

- Replicate this column of hidden neurons across space, with some stride.

Convolution Layers

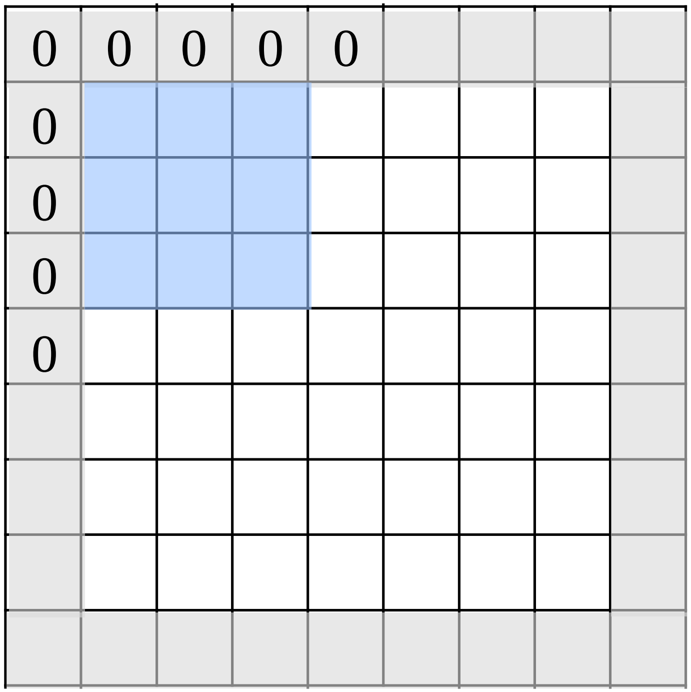


Example:
7×7 input
assume 3×3 connectivity
stride 1
⇒ 5×5 output

What about stride 2?
⇒ 3×3 output

- Replicate this column of hidden neurons across space, with some stride.

Convolution Layers



Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow 5 \times 5$ output

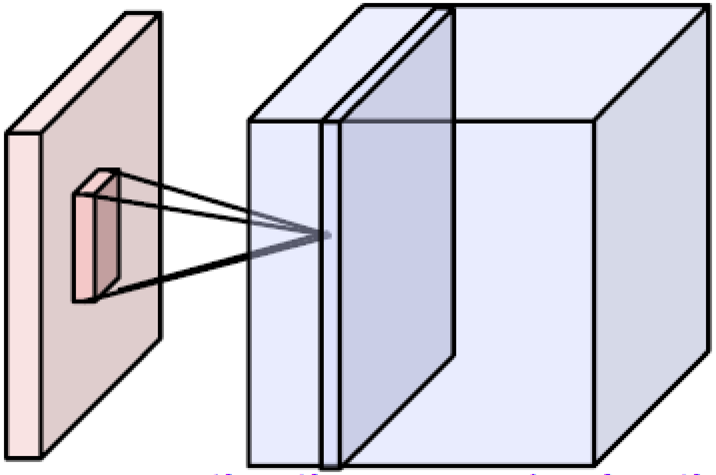
What about stride 2?
$\Rightarrow 3 \times 3$ output

- Replicate this column of hidden neurons across space, with some stride.
- In practice, common to zero-pad the border.
  - Preserves the size of the input spatially.

Activation Maps of Convolutional Filters

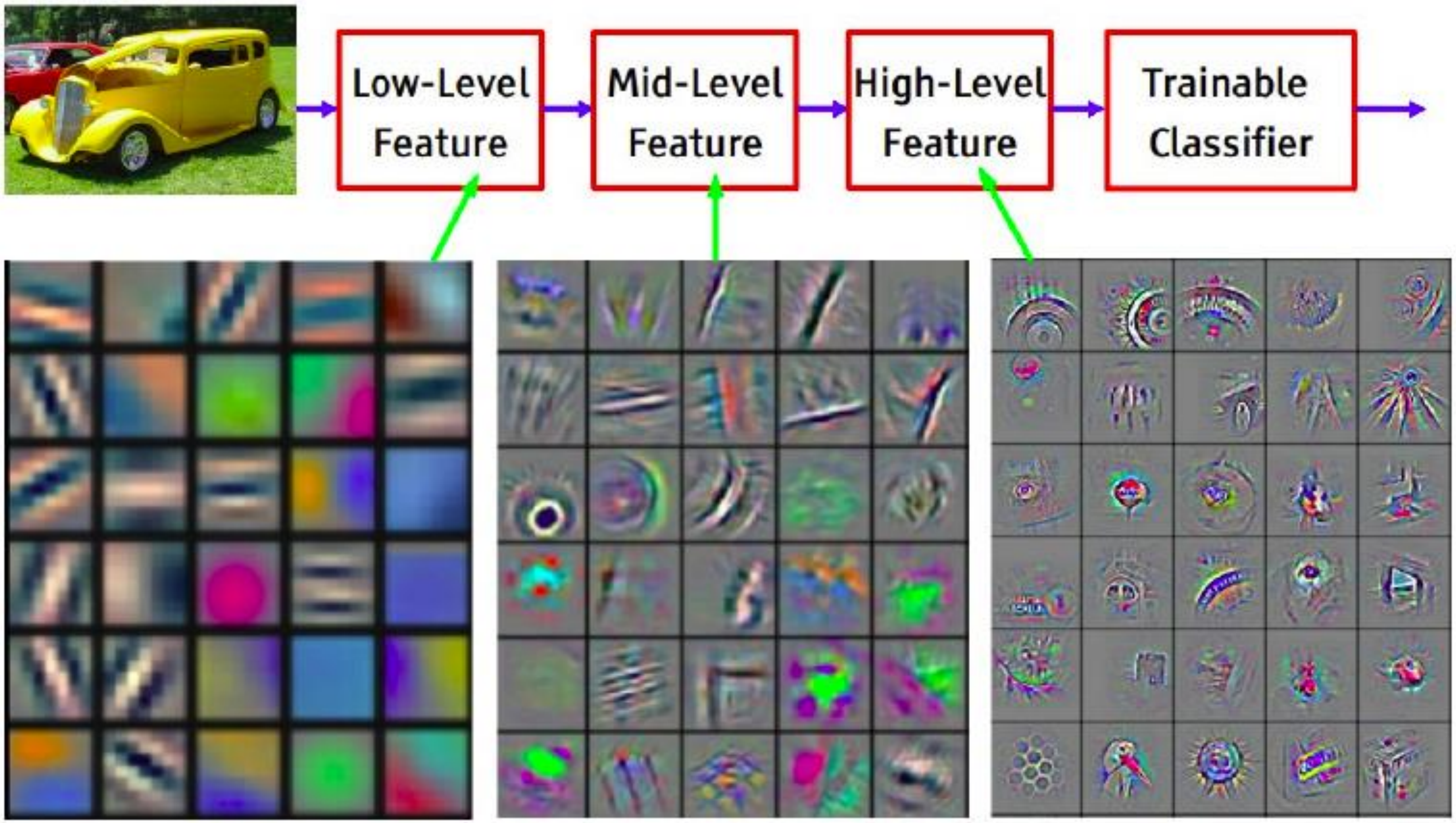
5×5 filters

Activation maps



Each activation map is a depth slice through the output volume.

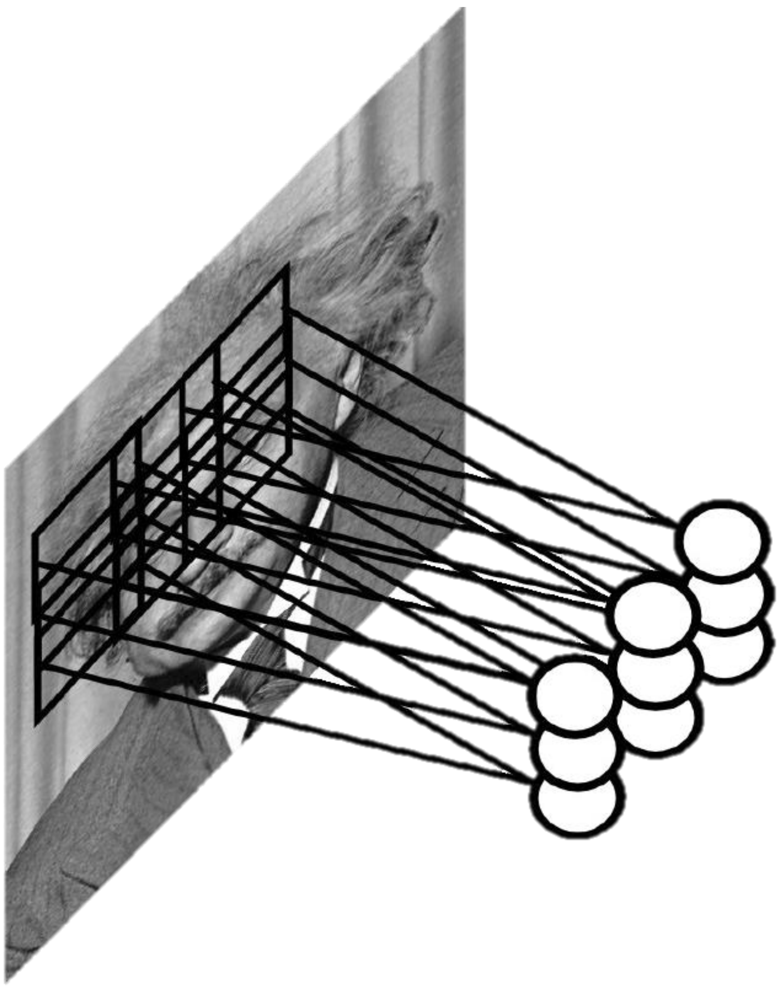
Effect of Multiple Convolution Layers



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Yann LeCun

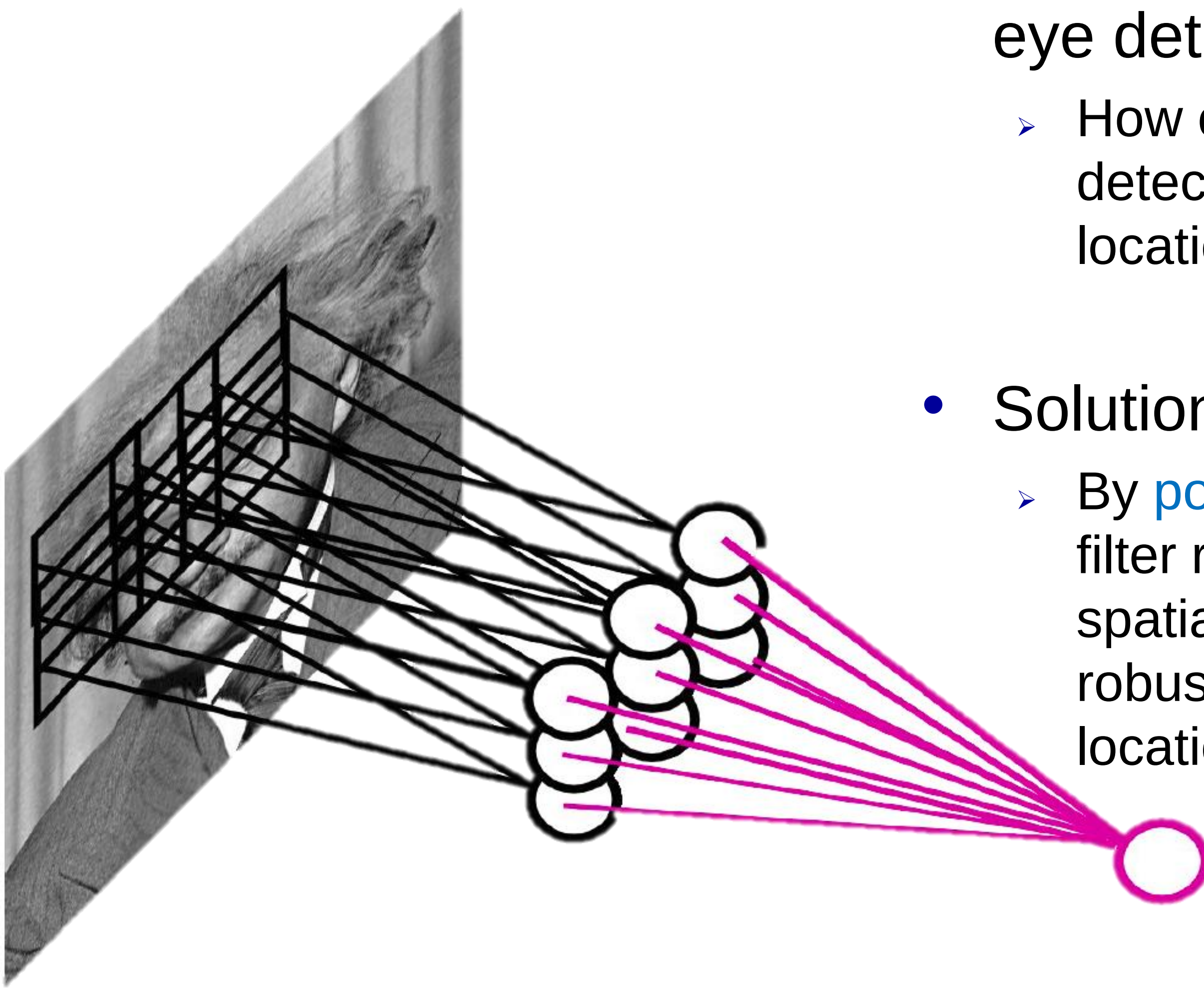
Convolutional Networks: Intuition



- Let's assume the filter is an eye detector
  - How can we make the detection robust to the exact location of the eye?

Image source: Yann LeCun

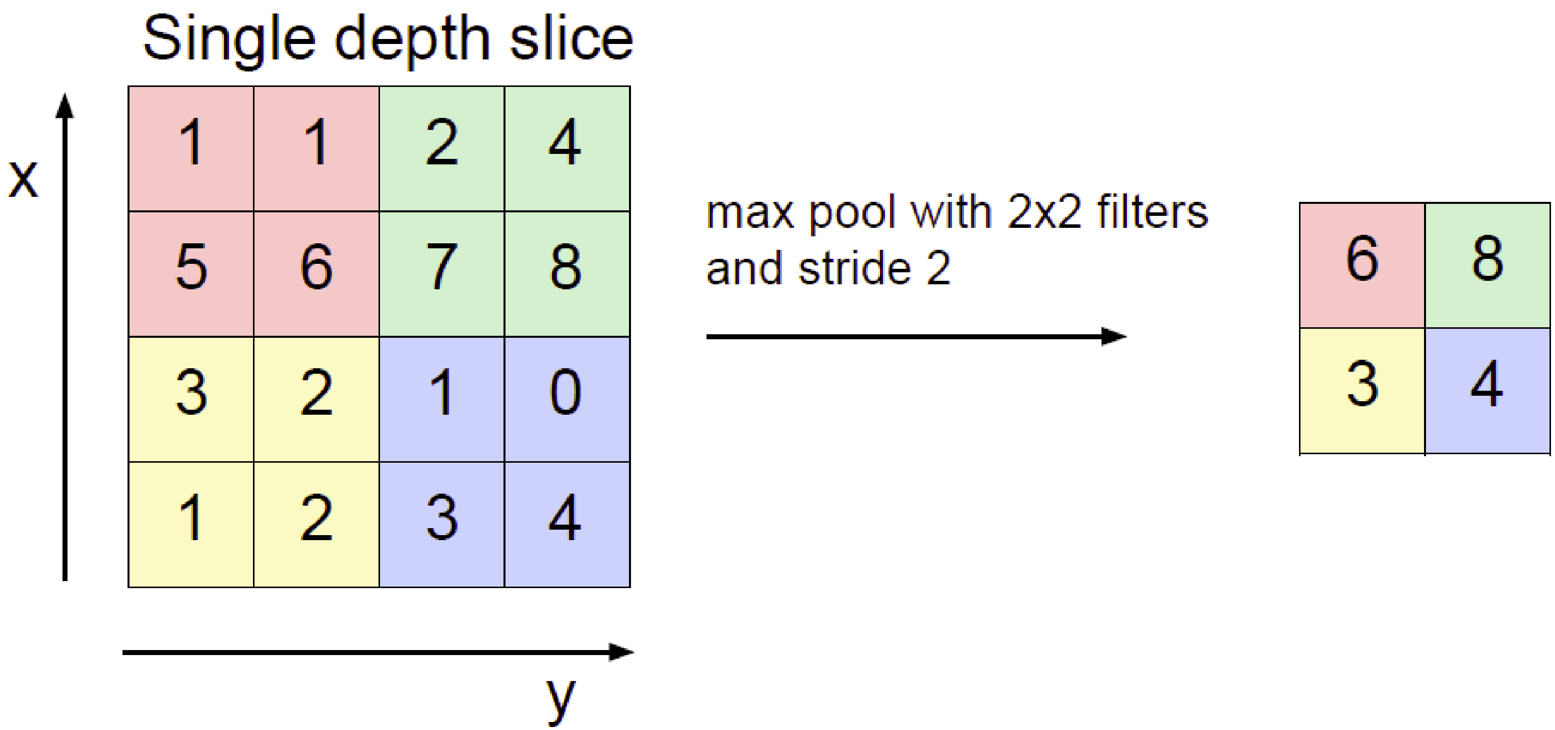
Convolutional Networks: Intuition



- Let’s assume the filter is an eye detector
  - How can we make the detection robust to the exact location of the eye?
- Solution:
  - By pooling (e.g., max or avg) filter responses at different spatial locations, we gain robustness to the exact spatial location of features.

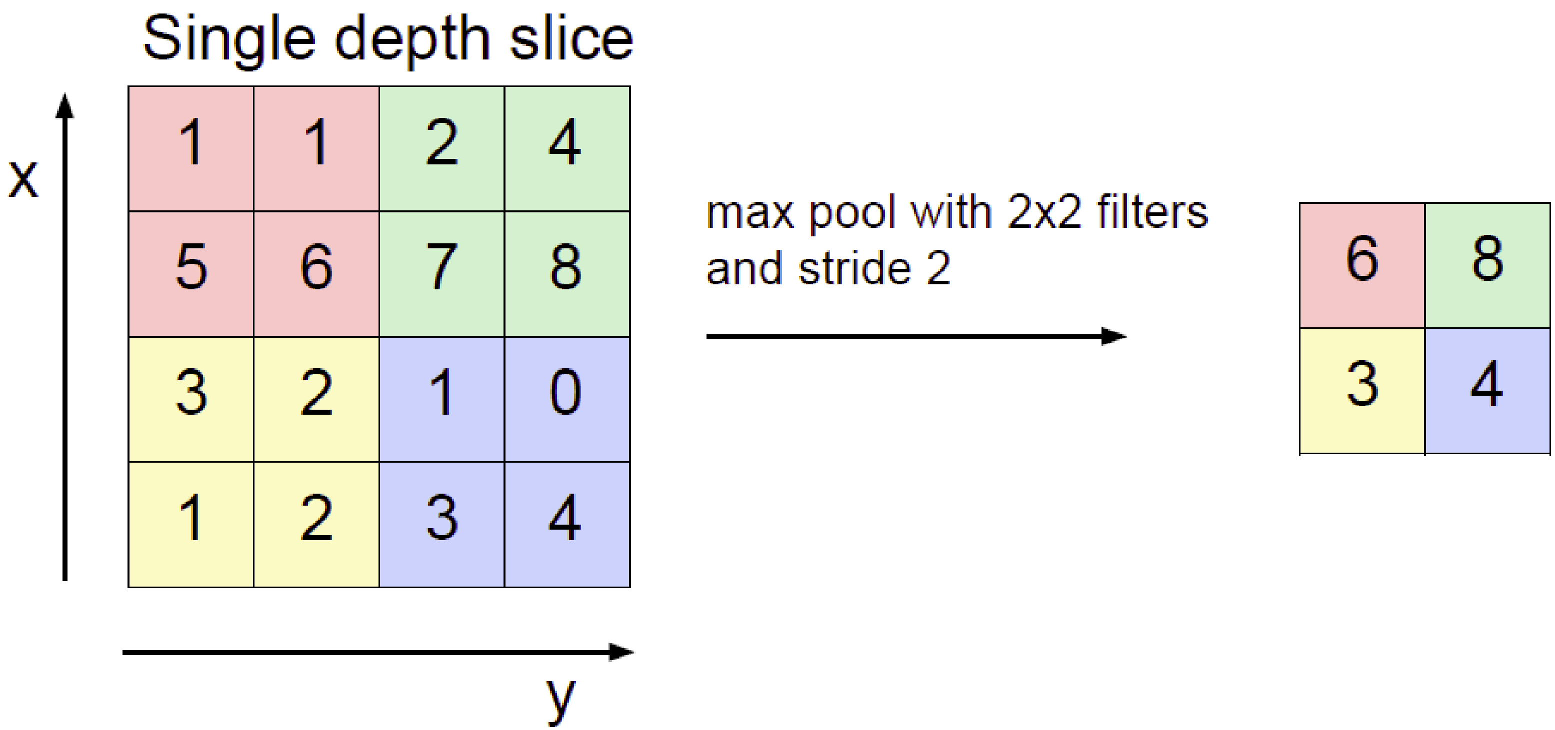
Image source: Yann LeCun

Max Pooling



- Effect:
  - Make the representation smaller without losing too much information
  - Achieve robustness to translations

Max Pooling



- Note
  - Pooling happens independently across each slice, preserving the number of slices.

CNNs: Implication for Back-Propagation

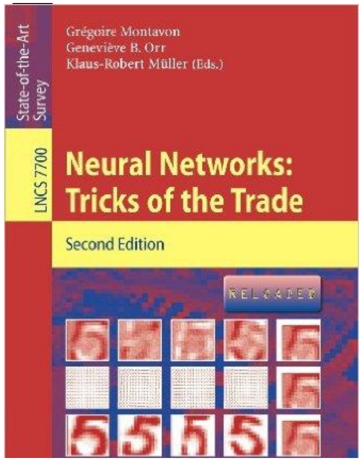
- Convolutional layers
  - Filter weights are shared between locations

  $\Rightarrow$ Gradients are added for each filter location.

References and Further Reading

- More information on many practical tricks can be found in Chapter 1 of the book

G. Montavon, G. B. Orr, K-R Mueller (Eds.)
Neural Networks: Tricks of the Trade
Springer, 1998, 2012



Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Mueller
Efficient BackProp, Ch.1 of the above book., 1998.

References

- ReLu
  - X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, AISTATS 2011.
- Initialization
  - X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, AISTATS 2010.
  - K. He, X.Y. Zhang, S.Q. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, ArXiV 1502.01852v1, 2015.
  - A.M. Saxe, J.L. McClelland, S. Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, ArXiV 1312.6120v3, 2014.

References and Further Reading

- Batch Normalization
  - S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ArXiV 1502.03167, 2015.
- Dropout
  - N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR, Vol. 15:1929-1958, 2014.