Machine Learning – Lecture 15

Convolutional Neural Networks

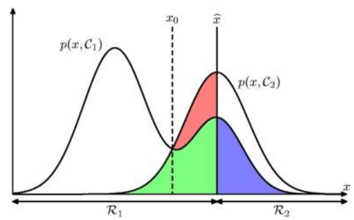
05.12.2019

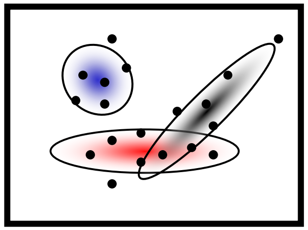
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

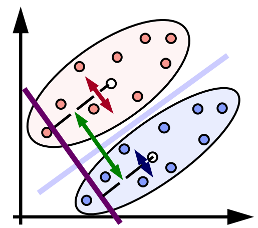
Course Outline

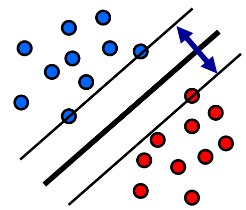
- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation

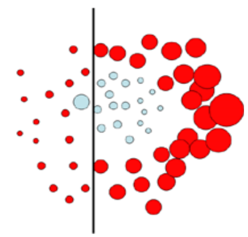


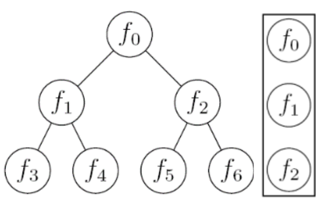


- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Random Forests

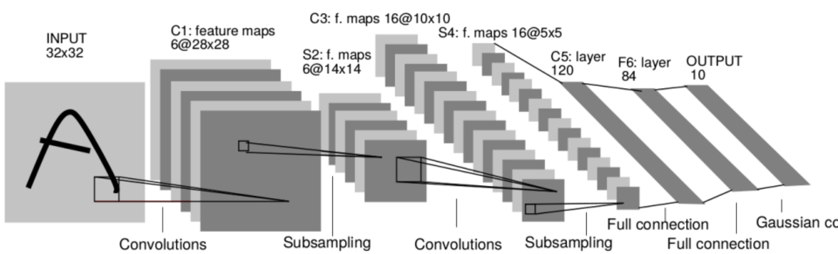








- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks

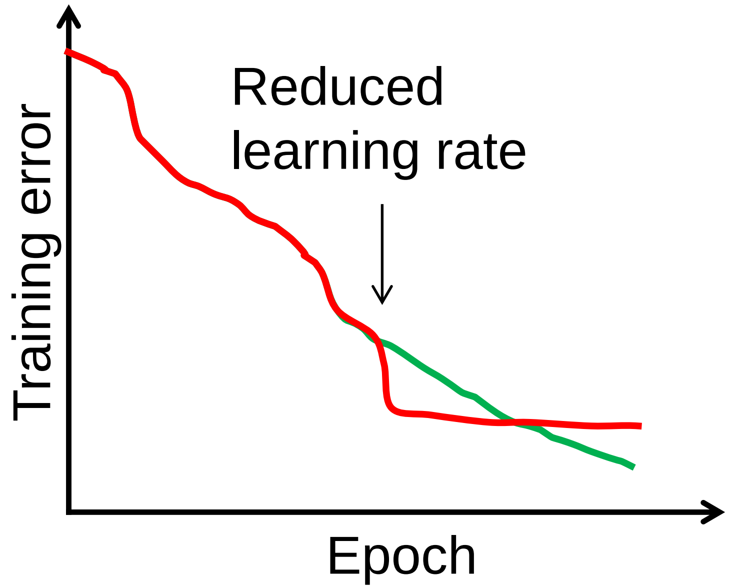


Topics of This Lecture

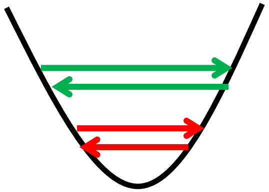
- Recap: Tricks of the Trade
  - Initialization
  - Dropout
  - Batch Normalization
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

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.



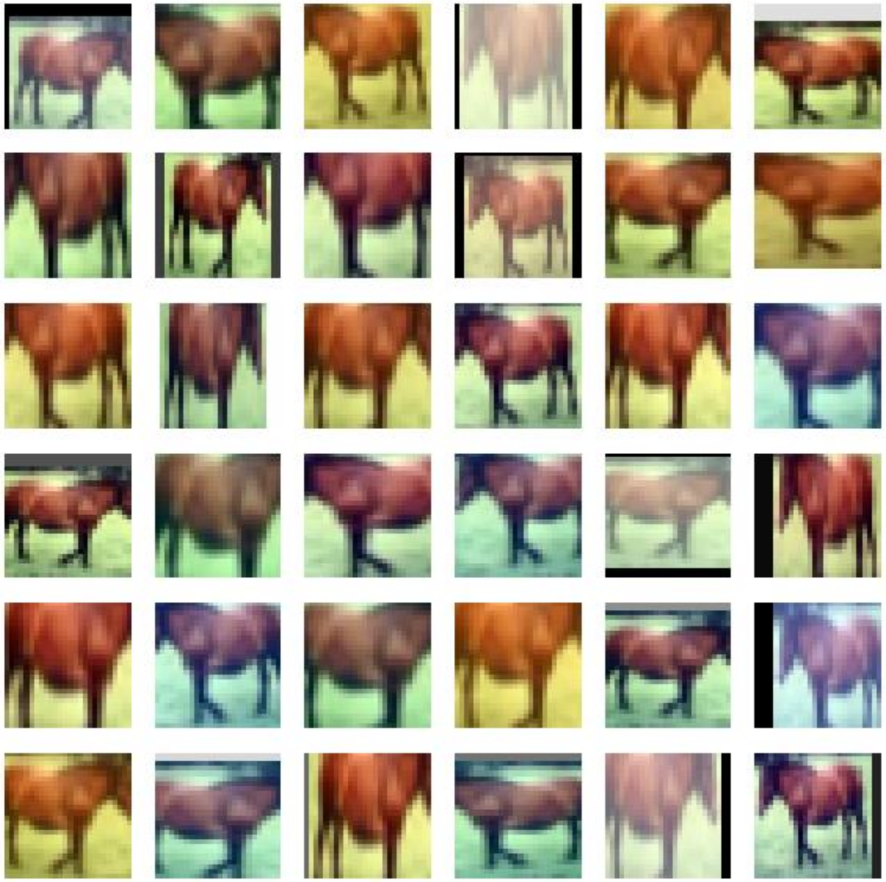
- Effect
  - Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different minibatches.



- *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.

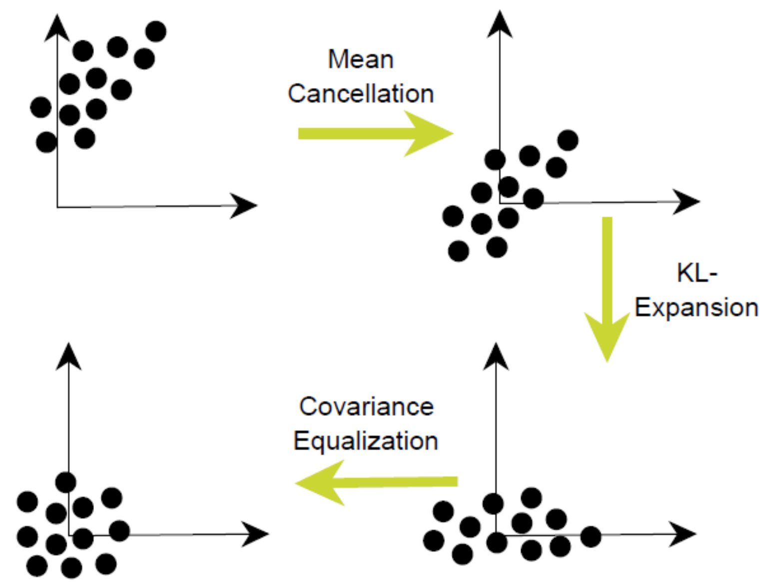


Augmented training data (from one original image)

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.



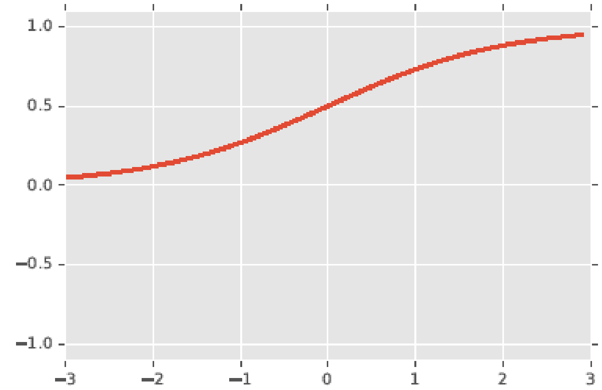
- Advisable normalization steps (for MLPs only, not for CNNs)
  - Normalize all inputs that an input unit sees to zero-mean, unit covariance.
  - If possible, try to decorrelate them using PCA (also known as Karhunen-Loeve expansion).

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

Recap: Commonly Used Nonlinearities

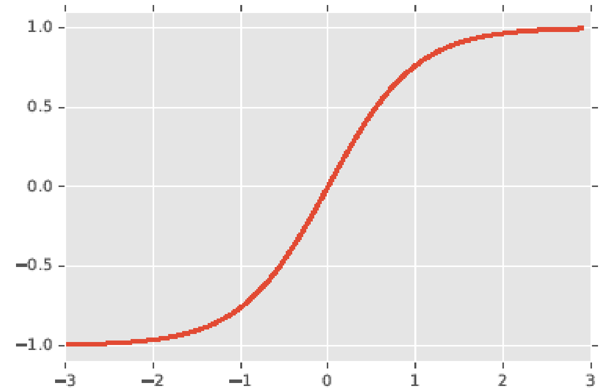
- Sigmoid

$$\begin{aligned} g(a) &= \sigma(a) \\ &= \frac{1}{1+\exp\{-a\}} \end{aligned}$$



- Hyperbolic tangent

$$\begin{aligned} g(a) &= tanh(a) \\ &= 2\sigma(2a) - 1 \end{aligned}$$



- Softmax

$$g(\mathbf{a}) = \frac{\exp\{-a_i\}}{\sum_j \exp\{-a_j\}}$$

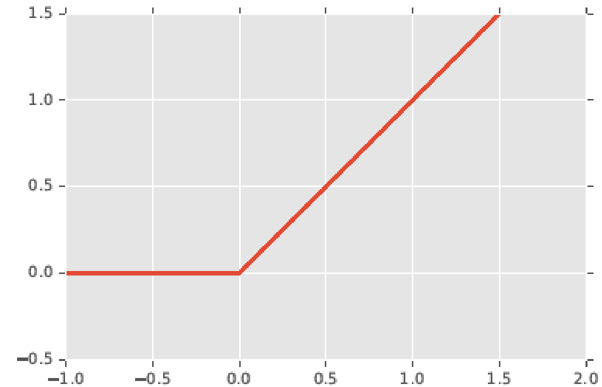
Extension: ReLU

- Another 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.*

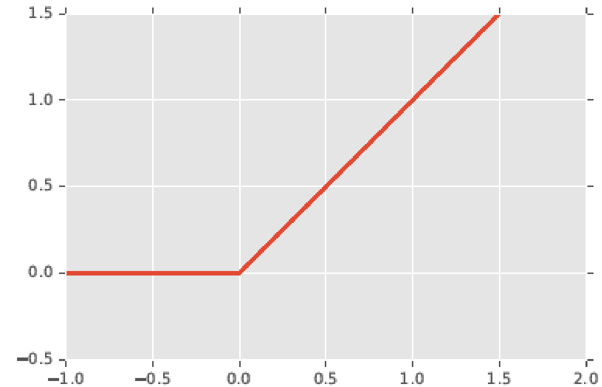
Extension: ReLU

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

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

  - Effect: gradient is propagated with a constant factor

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

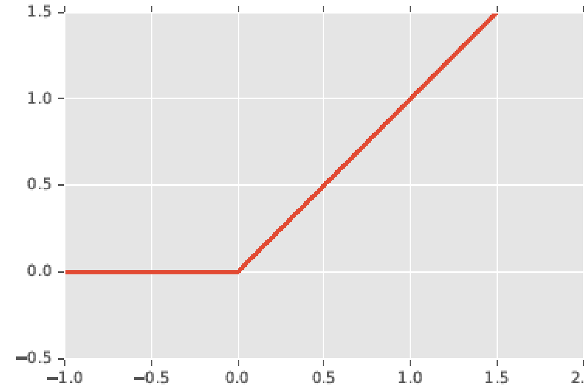


- Disadvantages / Limitations
  - A certain fraction of units will remain "stuck at zero".
    - If the initial weights are chosen such that the ReLU output is 0 for the entire training set, the unit will never pass through a gradient to change those weights.
  - ReLU has an offset bias, since its outputs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

Further Extensions

- Rectified linear unit (ReLU)

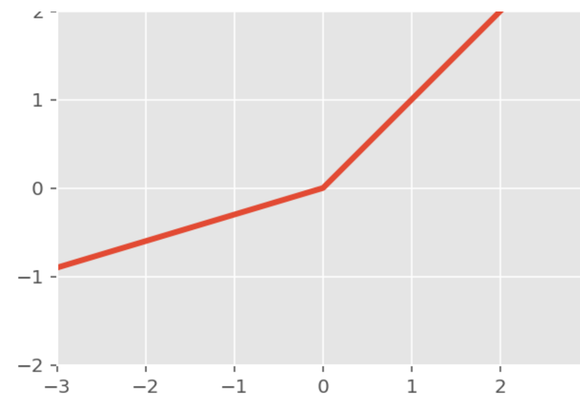
$$g(a) = \max\{0, a\}$$



- Leaky ReLU

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

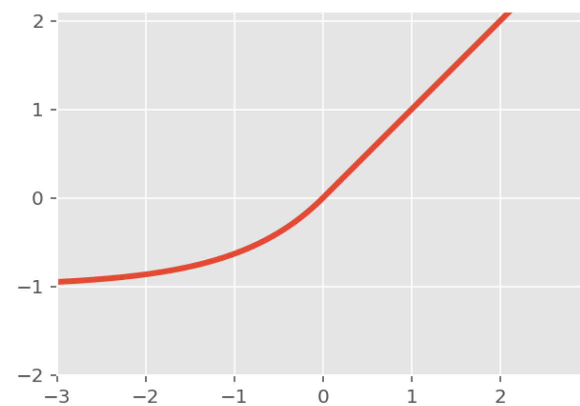
  - Avoids stuck-at-zero units
  - Weaker offset bias



- ELU

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

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



Topics of This Lecture

- Recap: Tricks of the Trade
  - Initialization
  - Dropout
  - Batch Normalization
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

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

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

$\Rightarrow$ 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 $n \, \mathrm{Var}(W_i)$ should be 1. This means

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

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

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

  - As a compromise, Glorot & Bengio proposed to use

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

$\Rightarrow$ 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

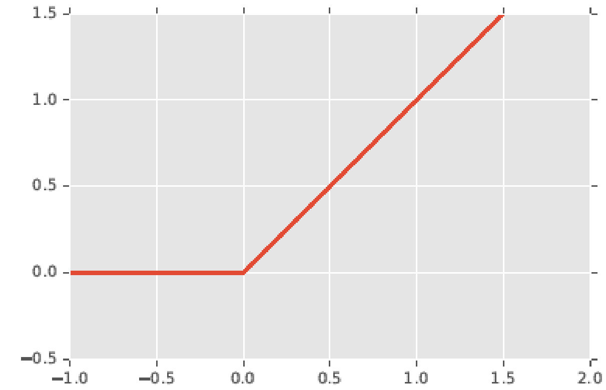
$$W \sim U\left[-\frac{\sqrt{3}}{\sqrt{n_{in}}}, \frac{\sqrt{3}}{\sqrt{n_{in}}}\right]$$

  - *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

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

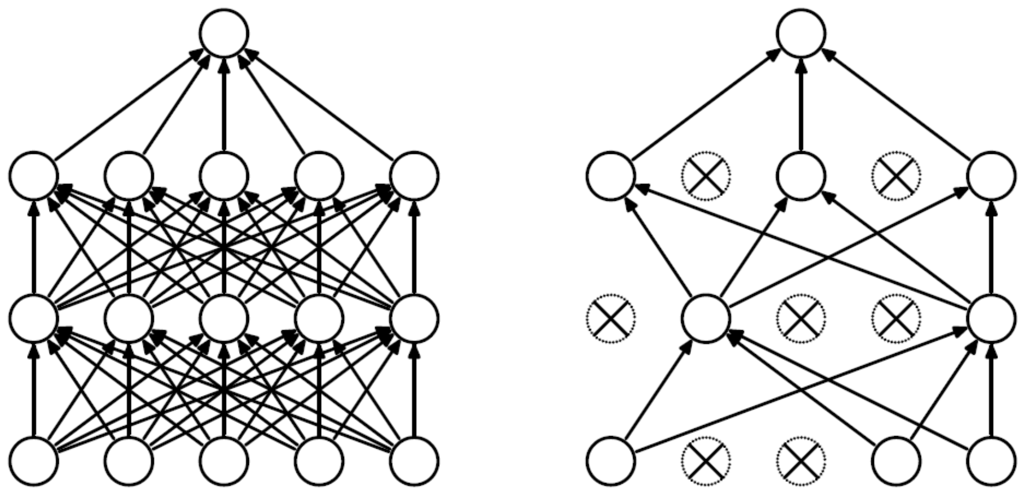
Topics of This Lecture

- Recap: Tricks of the Trade
  - Initialization
  - Dropout
  - Batch Normalization
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

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 (a form of regularization).
  - Change network architecture for each minibatch, 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 during training.

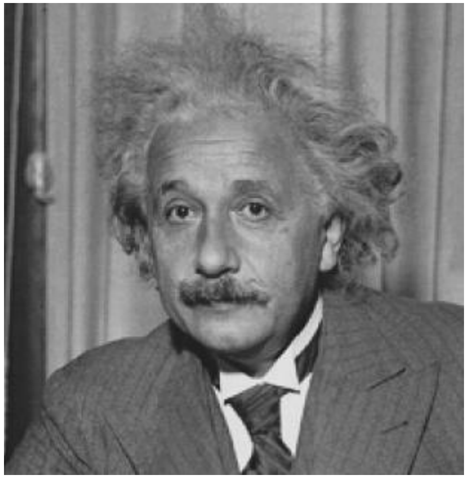
  ⇒ Greatly improved performance

Topics of This Lecture

- Recap: Tricks of the Trade
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

Neural Networks for Computer Vision

- How should we approach vision problems?



→ Face Y/N?

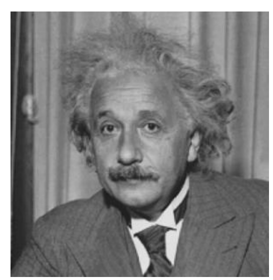
- Architectural considerations
  - Input is 2D ⇒ 2D layers of units
  - No pre-segmentation ⇒ Need robustness to misalignments
  - Vision is hierarchical ⇒ Hierarchical multi-layered structure
  - Vision is difficult ⇒ Network should be deep

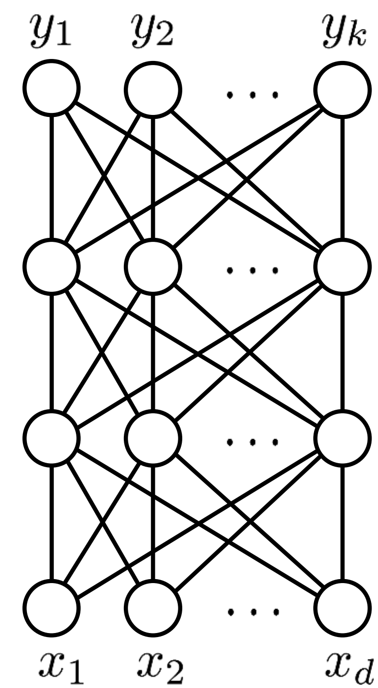
Why Hierarchical Multi-Layered Models?

- Motivation 1: Visual scenes are hierarchically organized

Object
↑
Object parts
↑
Primitive features
↑
Input image

Face
↑
Eyes, nose, ...
↑
Oriented edges
↑
Face image

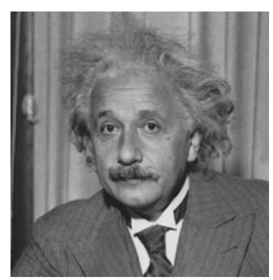




Why Hierarchical Multi-Layered Models?

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

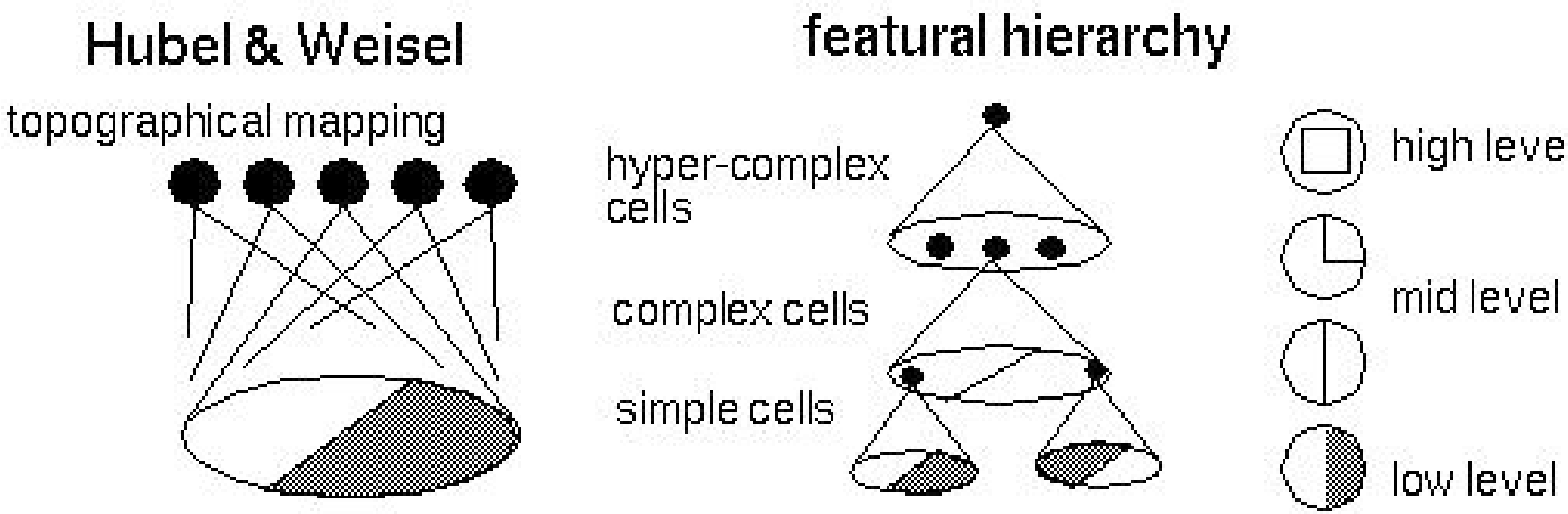
Object	Face	Inferotemporal cortex
↑	↑	
Object parts	Eyes, nose, ...	V4: different textures
↑	↑	
Primitive features	Oriented edges	V1: simple and complex cells
↑	↑	
Input image	Face image	Photoreceptors, retina



Slide adapted from Richard Turner

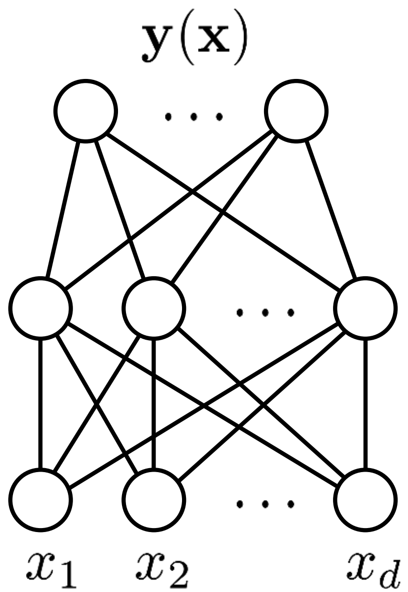
Hubel/Wiesel Architecture

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

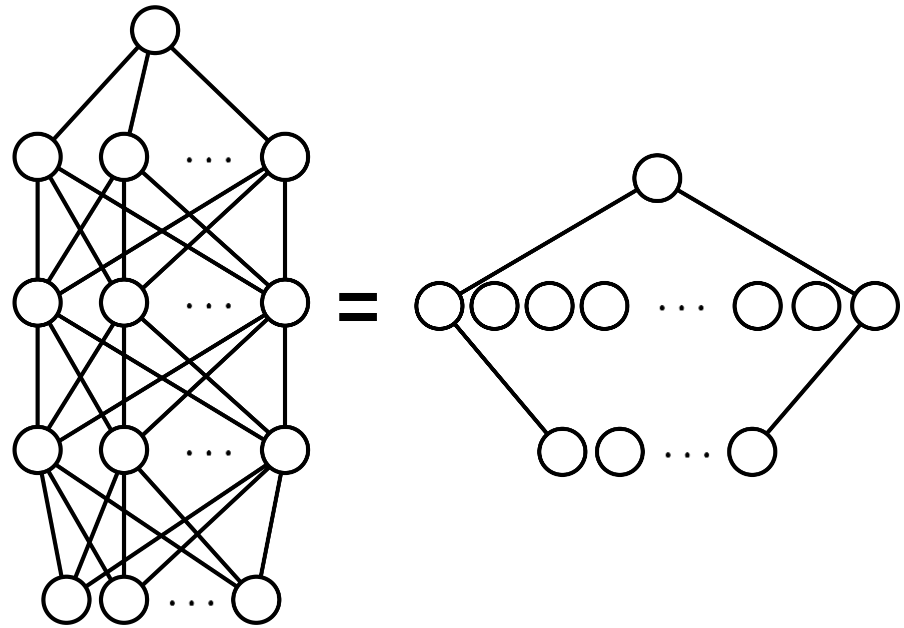


Why Hierarchical Multi-Layered Models?

- 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.

What's Wrong With Standard Neural Networks?

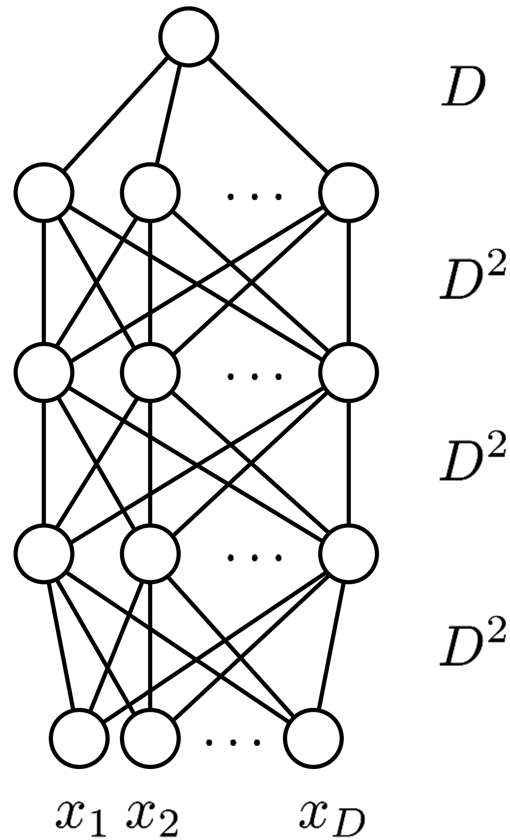
- Complexity analysis
  - How many parameters does this network have?

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

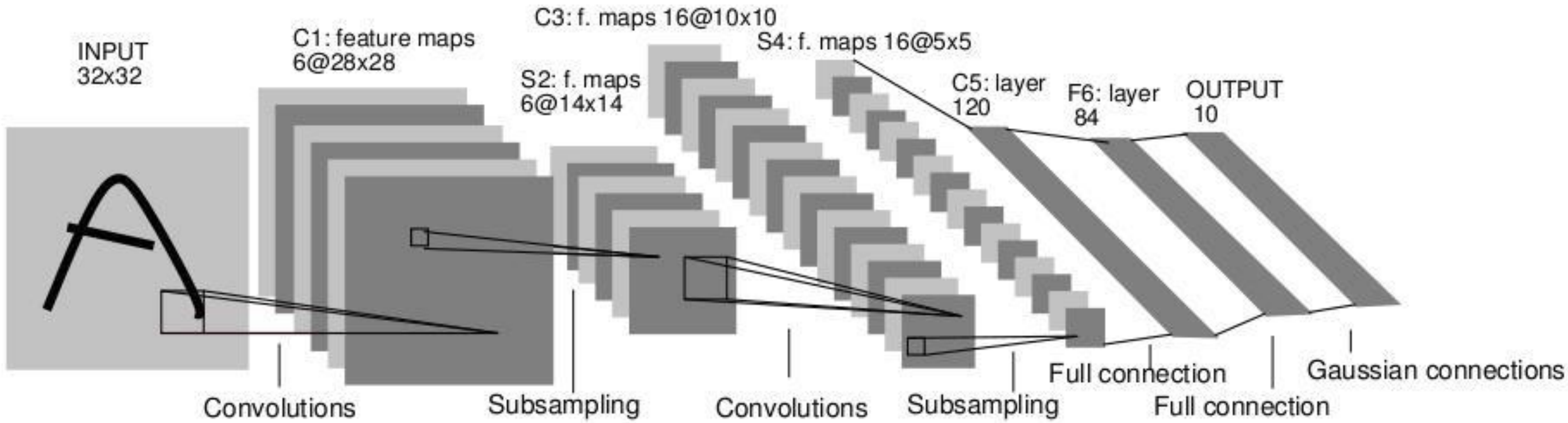
  - For a small 32×32 image

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

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



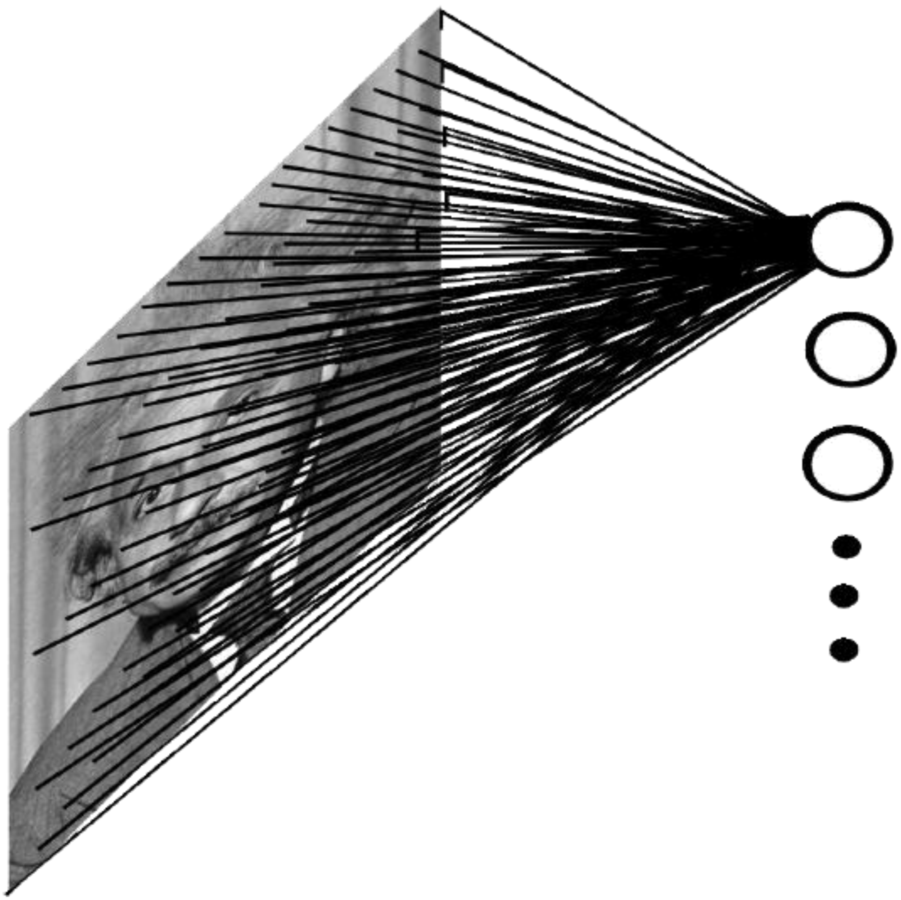
Convolutional Neural Networks (CNN, ConvNet)



- 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



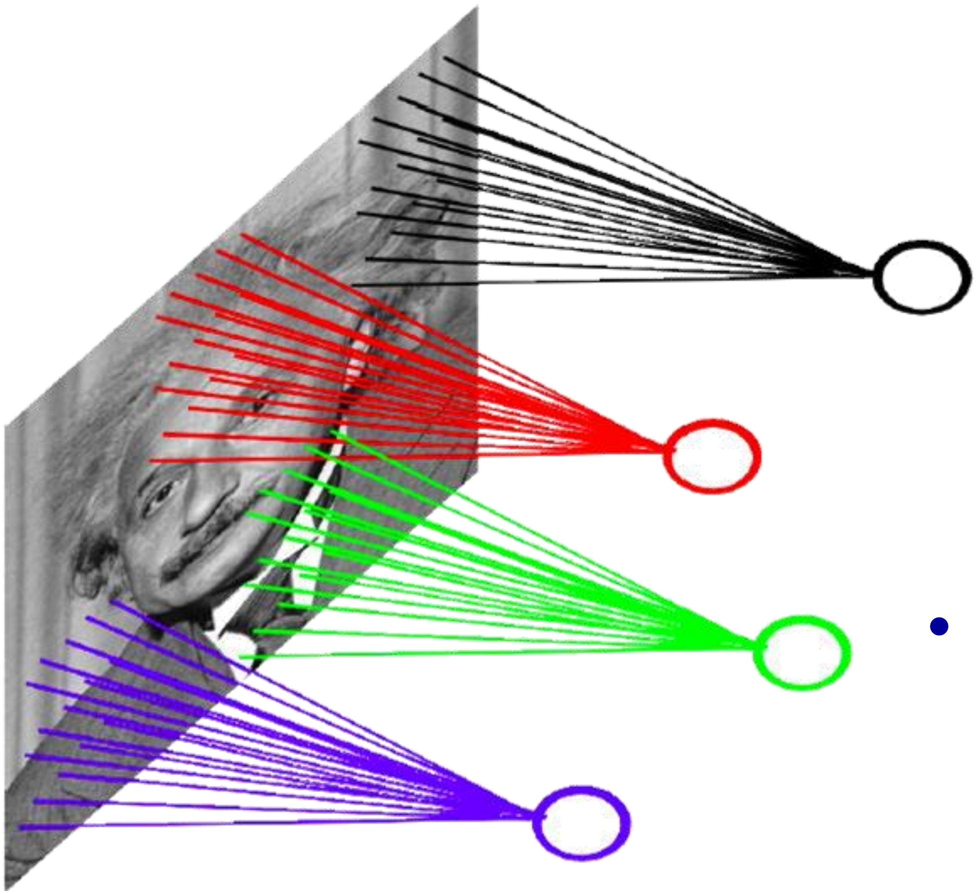
- Fully connected network
  - E.g. 1000×1000 image
    1M hidden units

  $\Rightarrow$ 1T parameters!

- Ideas to improve this
  - Spatial correlation is local

Image source: Yann LeCun

Convolutional Networks: Intuition



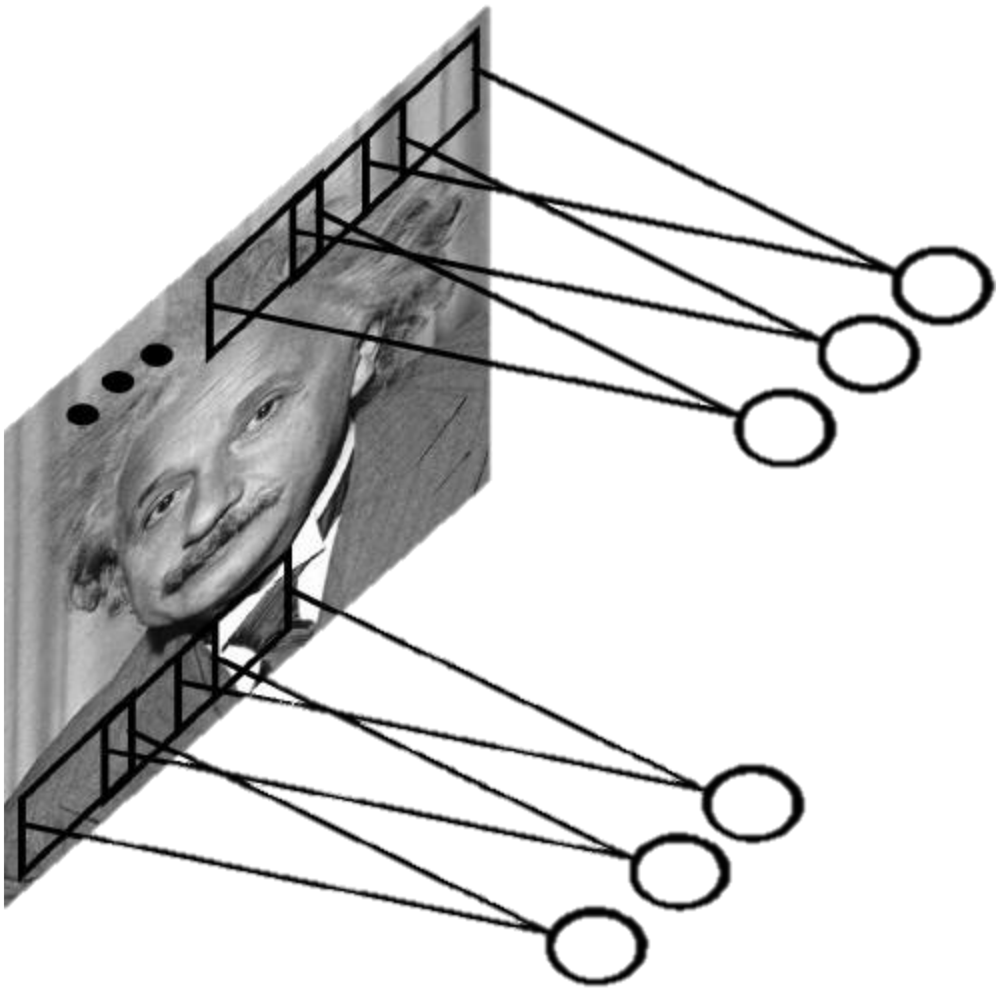
- Locally connected net
  - E.g. 1000×1000 image
    1M hidden units
    10×10 receptive fields

  ⇒ 100M parameters!

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

Image source: Yann LeCun

Convolutional Networks: Intuition

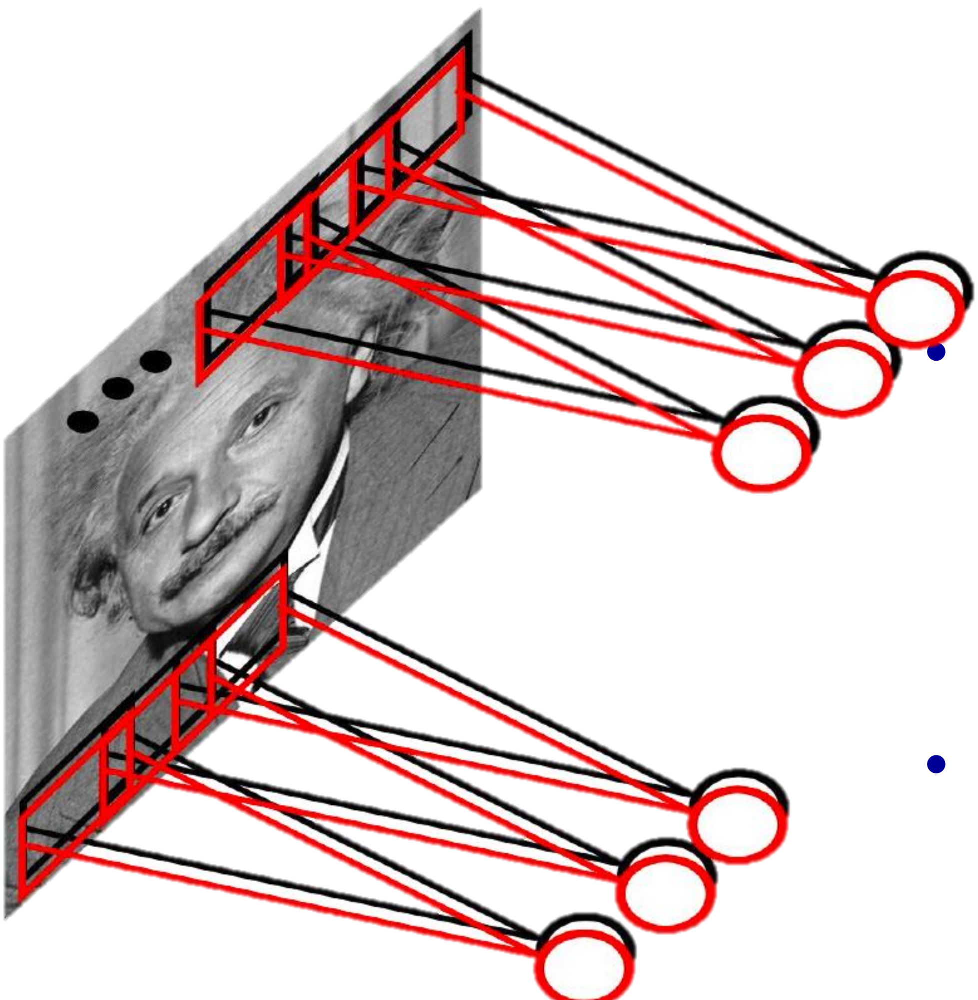


- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels

Slide adapted from Marc'Aurelio Ranzato

Image source: Yann LeCun

Convolutional Networks: Intuition



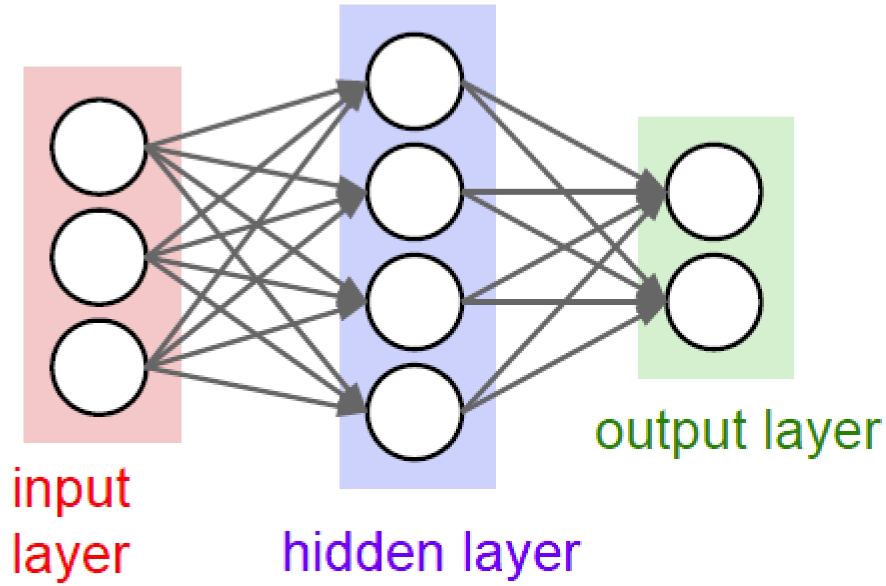
- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels
- Learn *multiple* filters
  - E.g. 1000×1000 image
    100 filters
    10×10 filter size

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

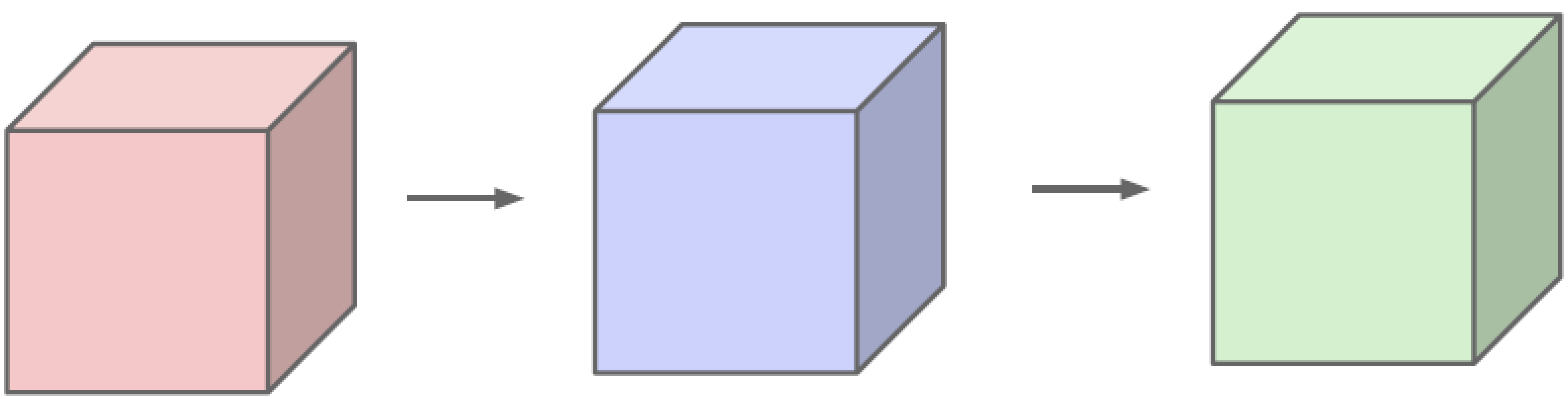
Image source: Yann LeCun

Important Conceptual Shift

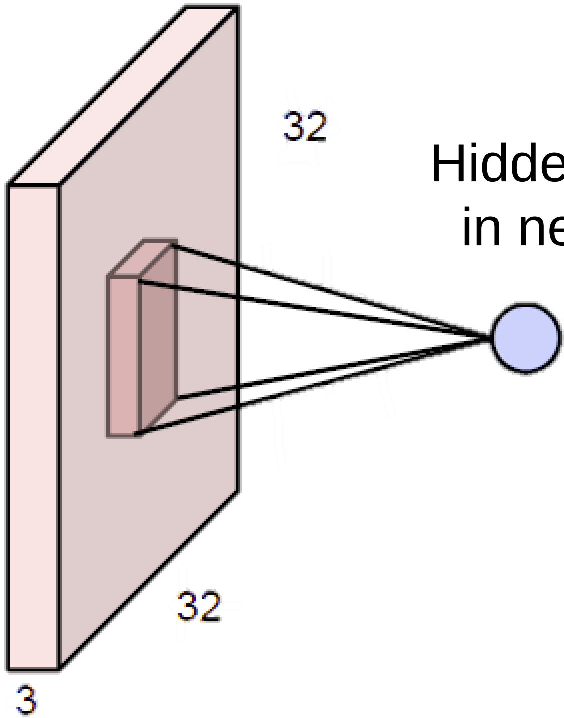
- Before



- Now:



Convolution Layers



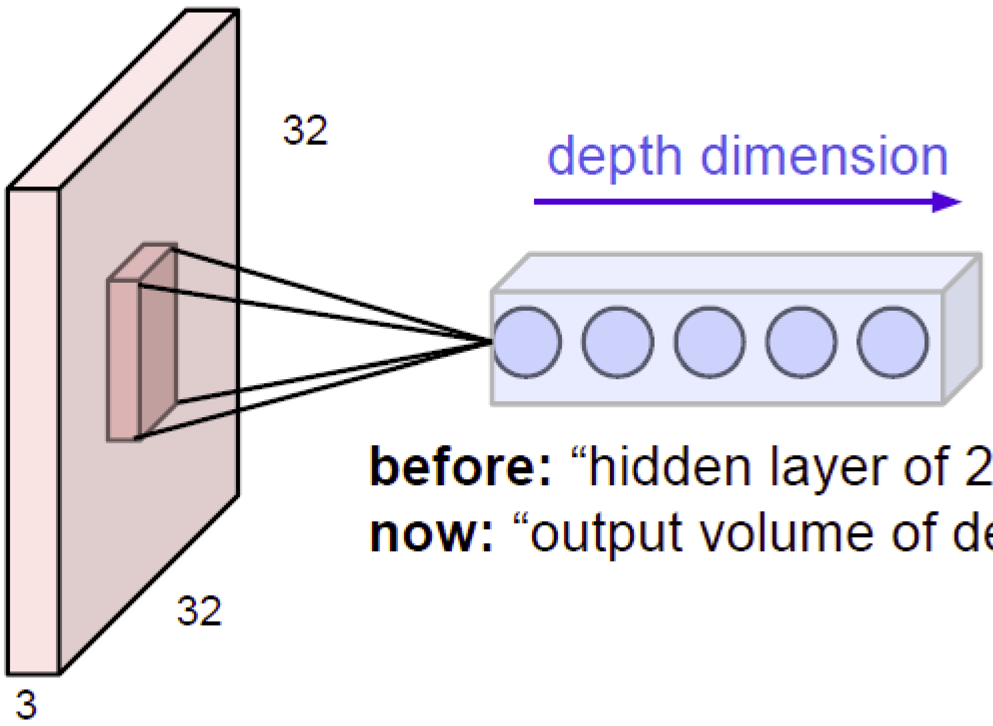
Example
image: $32 \times 32 \times 3$ volume

Before: Full connectivity
$32 \times 32 \times 3$ weights

Now: Local connectivity
One neuron connects to, e.g., $5 \times 5 \times 3$ region.
$\Rightarrow$ Only $5 \times 5 \times 3$ shared weights.

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

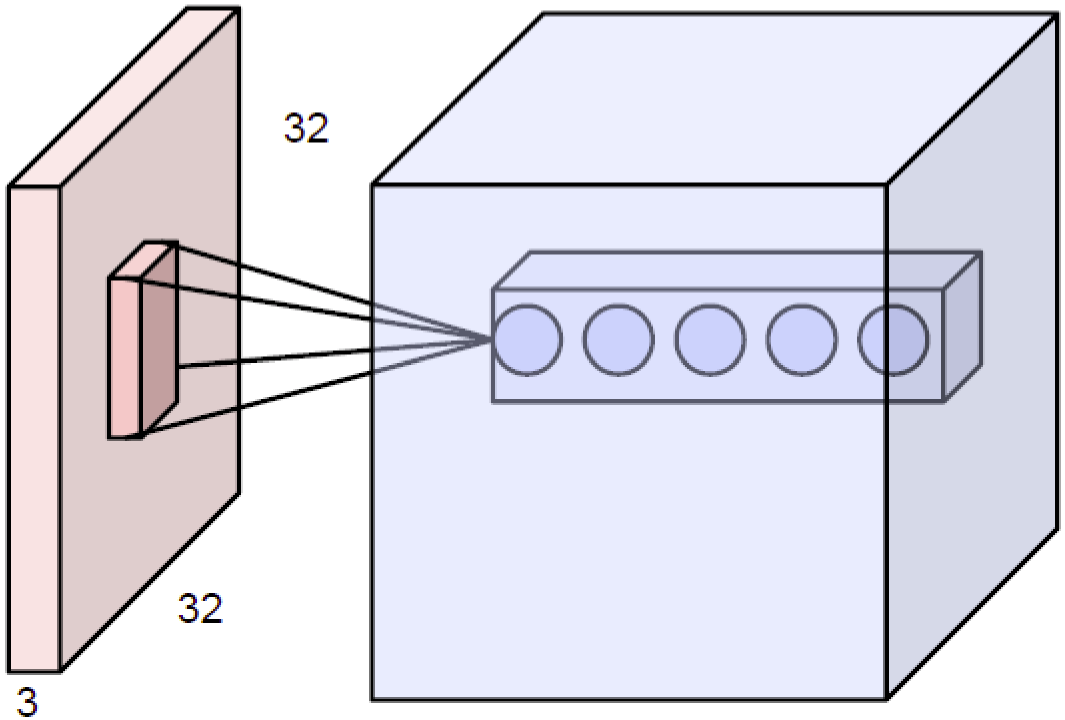
Convolution Layers



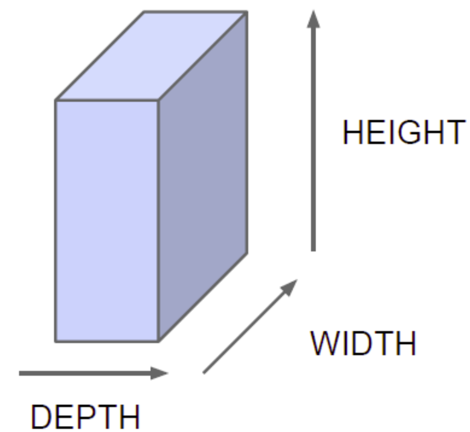
before: "hidden layer of 200 neurons"
now: "output volume of depth 200"

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

Convolution Layers

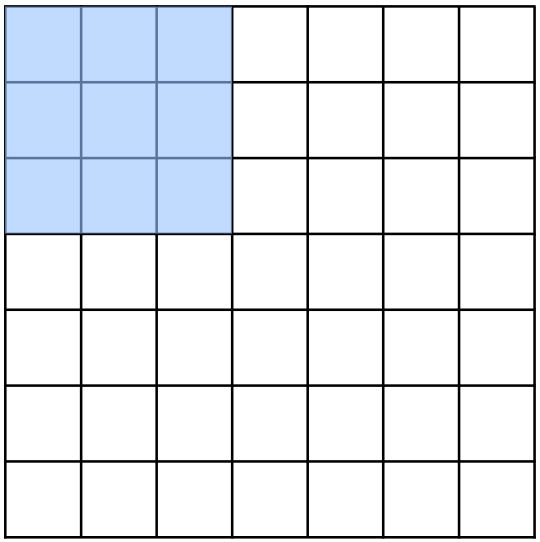


Naming convention:



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

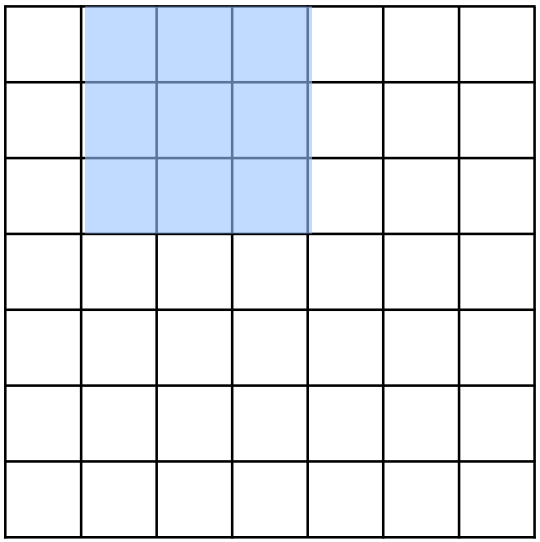
Convolution Layers



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

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

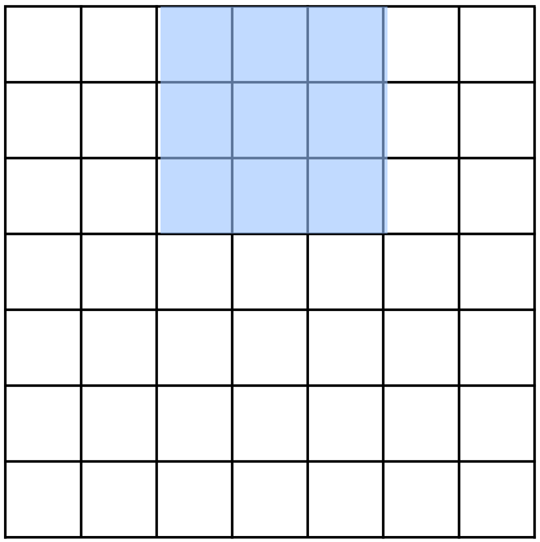
Convolution Layers



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

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

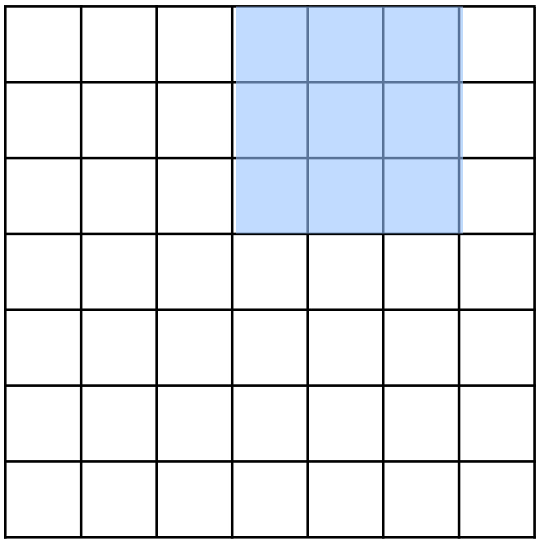
Convolution Layers



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

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

Convolution Layers

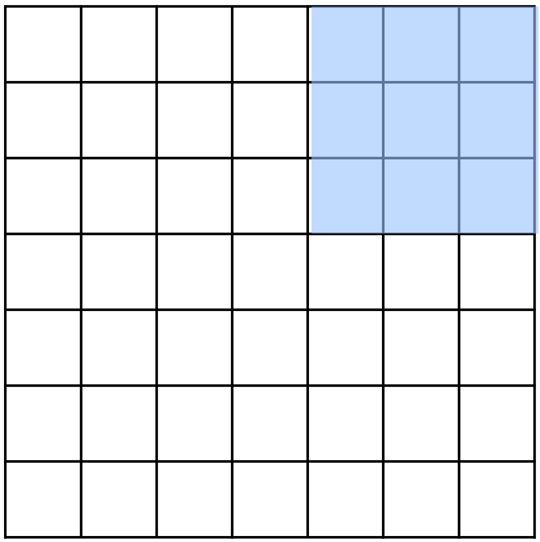


Example:
7×7 input
assume 3×3 connectivity
stride 1

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

Slide credit: FeiFei Li, Andrej Karpathy

Convolution Layers

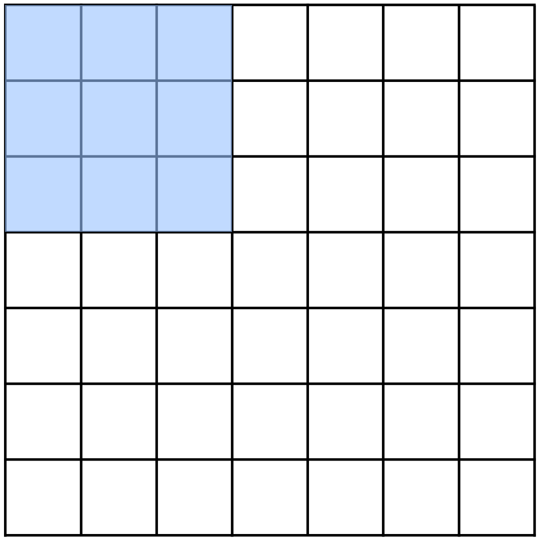


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow 5 \times 5$ output

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

Slide credit: FeiFei Li, Andrej Karpathy

Convolution Layers

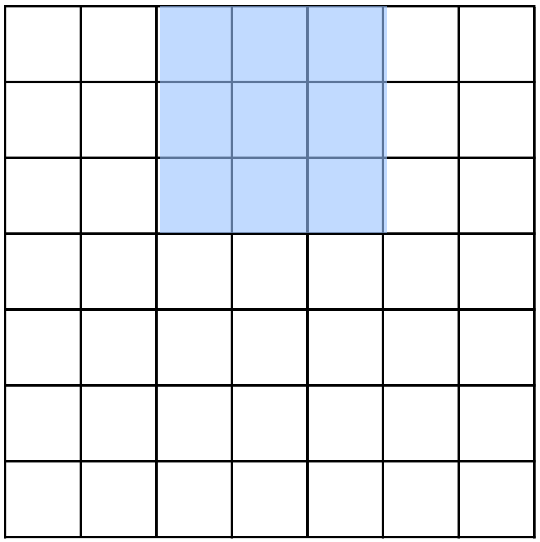


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow$ 5×5 output

What about stride 2?

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

Convolution Layers

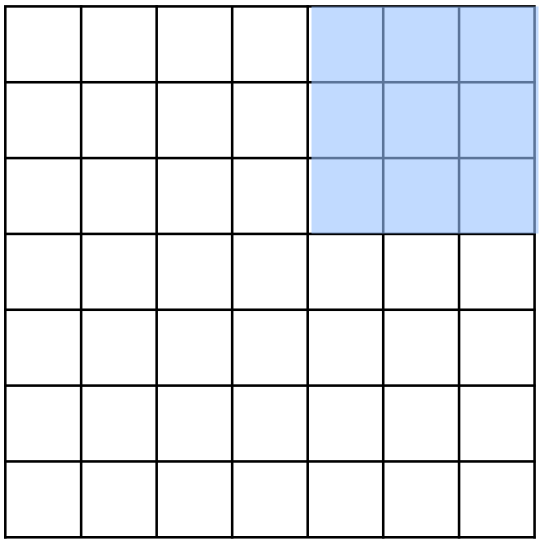


Example:
7×7 input
assume 3×3 connectivity
stride 1
⇒ 5×5 output

What about stride 2?

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

Convolution Layers

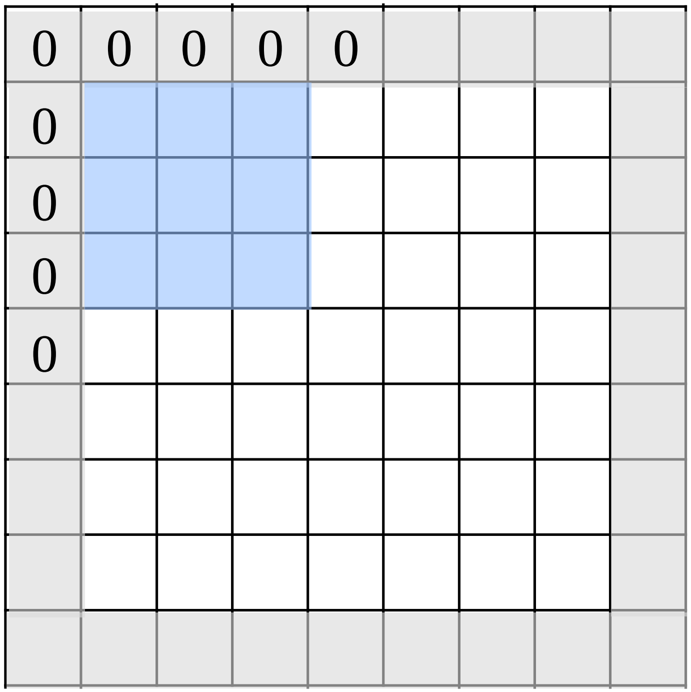


Example:
7×7 input
assume 3×3 connectivity
stride 1
⇒ 5×5 output

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

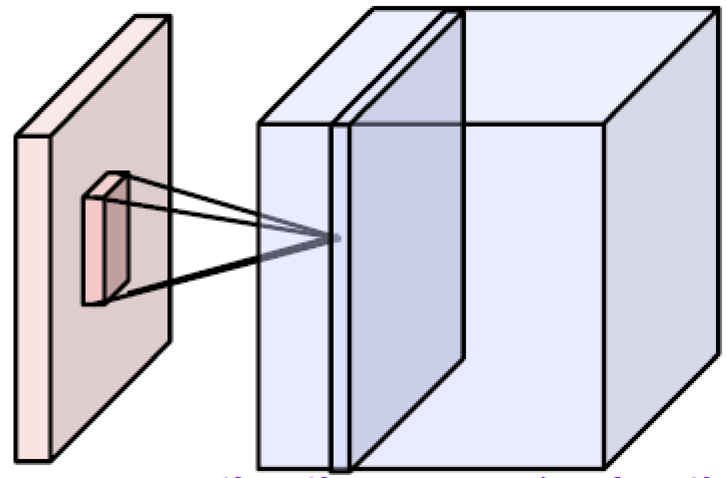
What about stride 2?
$\Rightarrow 3 \times 3$ output

- 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

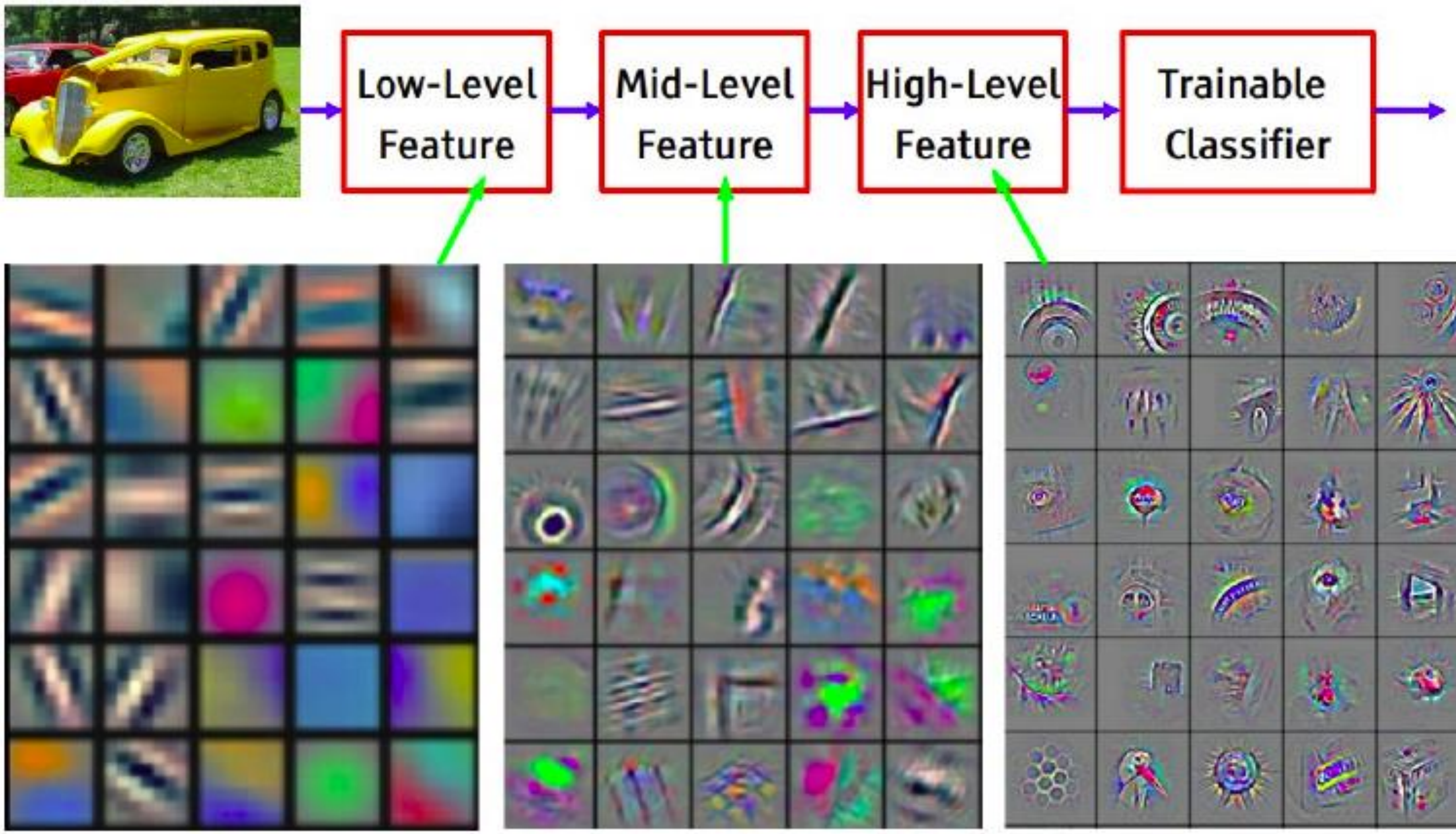
5×5 filters

Activation maps



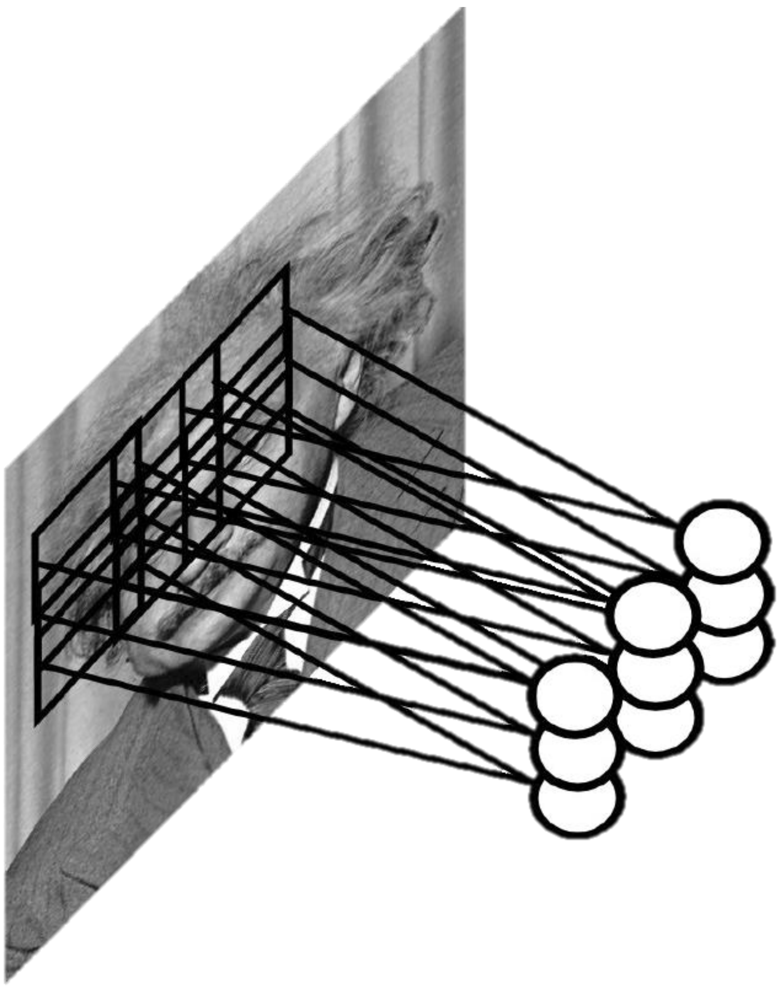
Each activation map is a depth slice through the output volume.

Effect of Multiple Convolution Layers



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

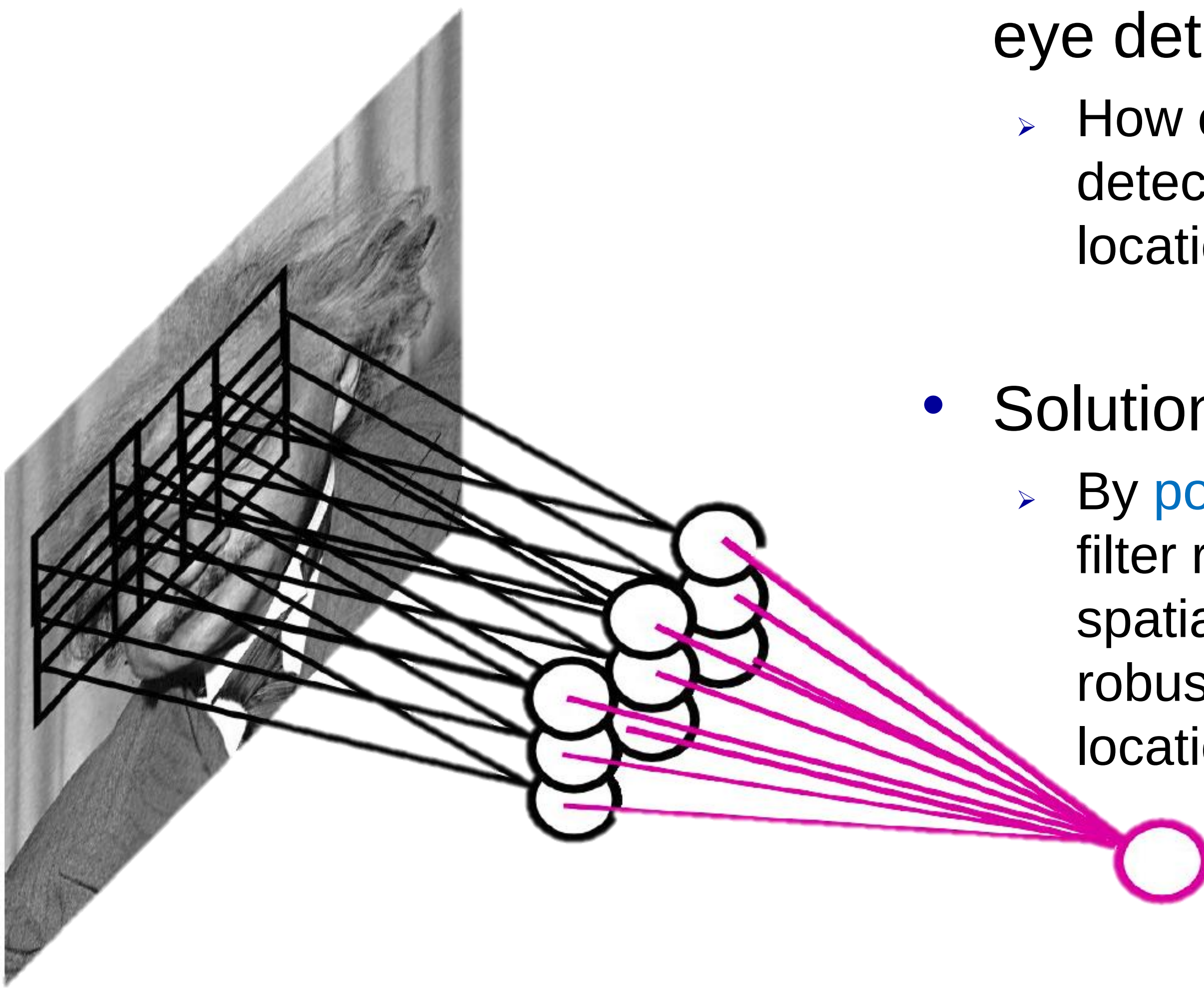
Convolutional Networks: Intuition



- 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

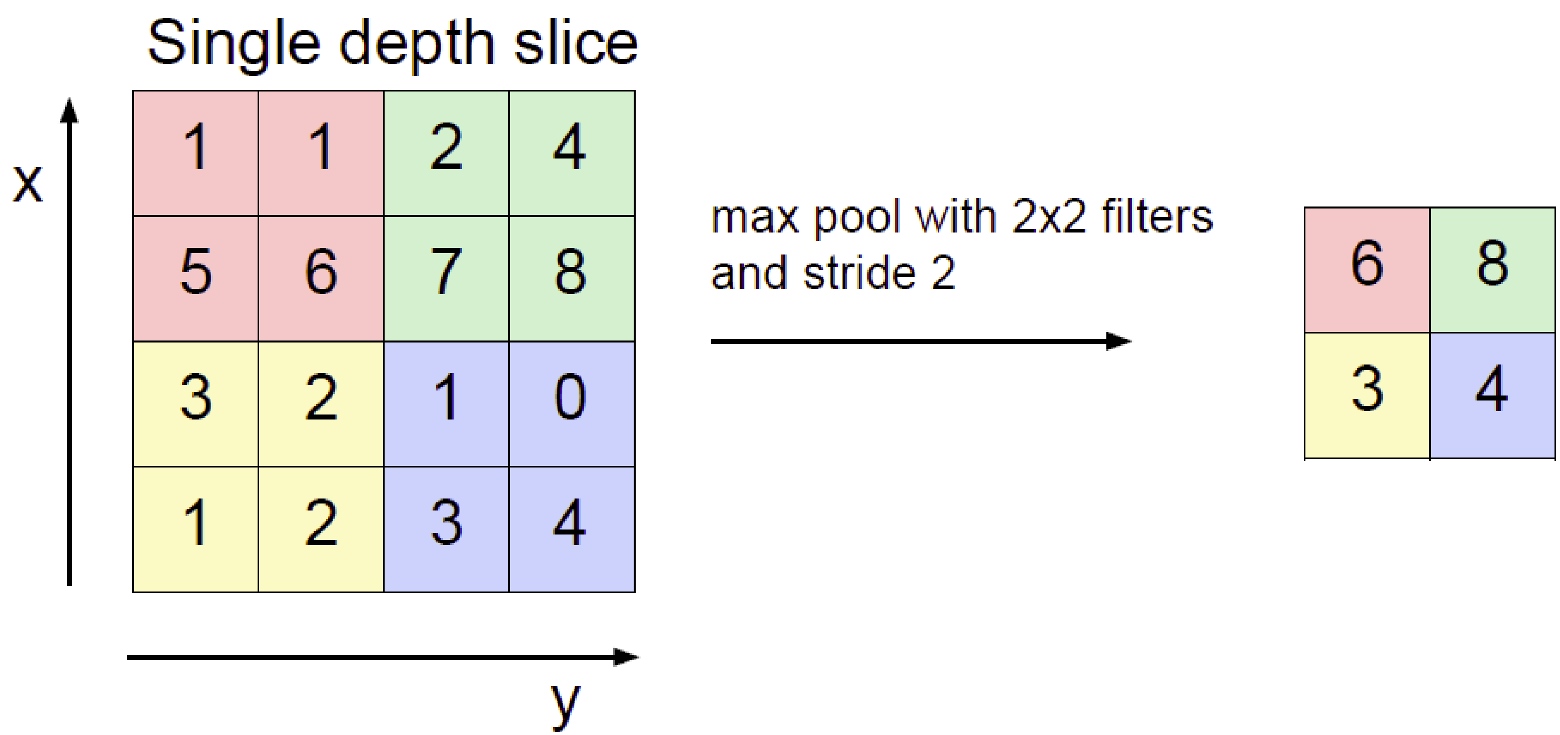


- Let’s assume the filter is an eye detector
  - How can we make the detection robust to the exact location of the eye?

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

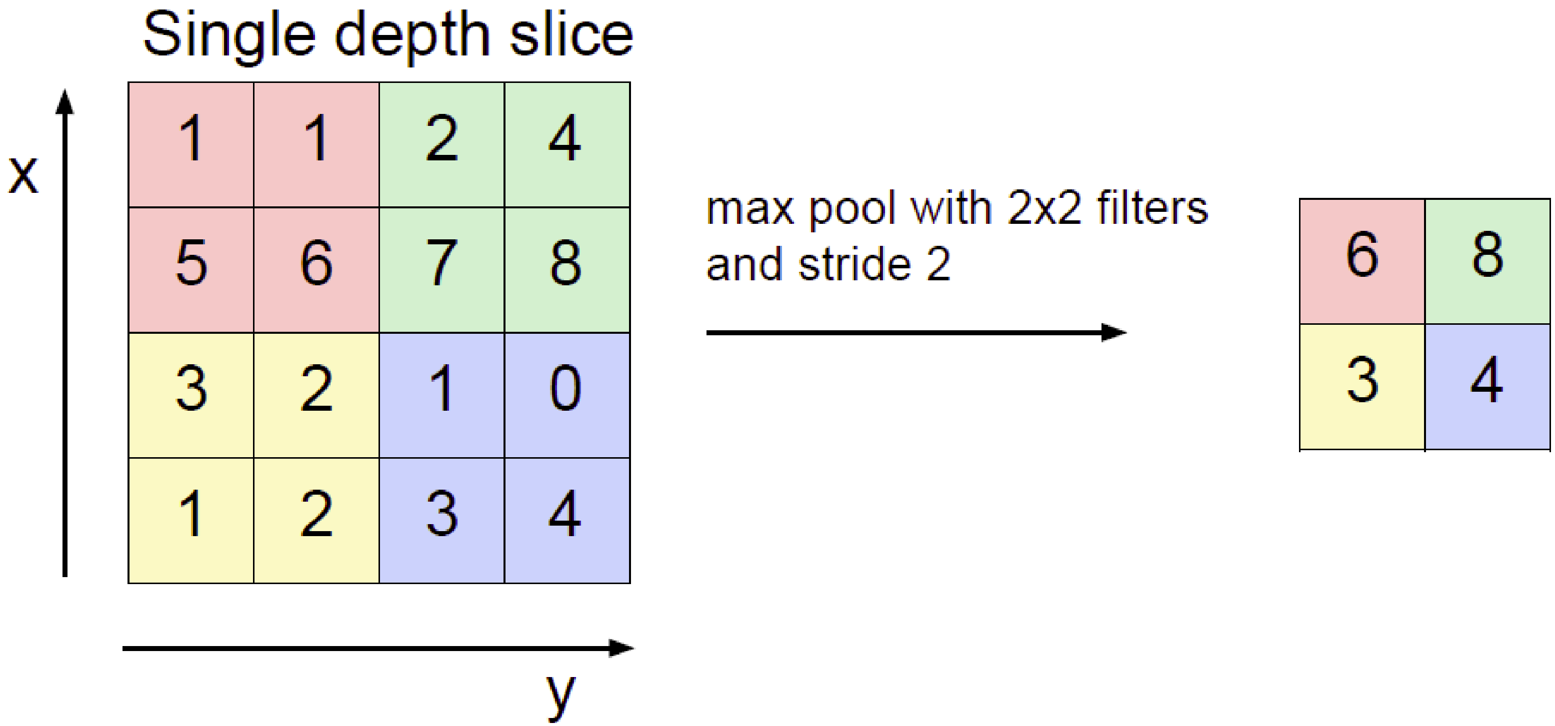
Image source: Yann LeCun

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

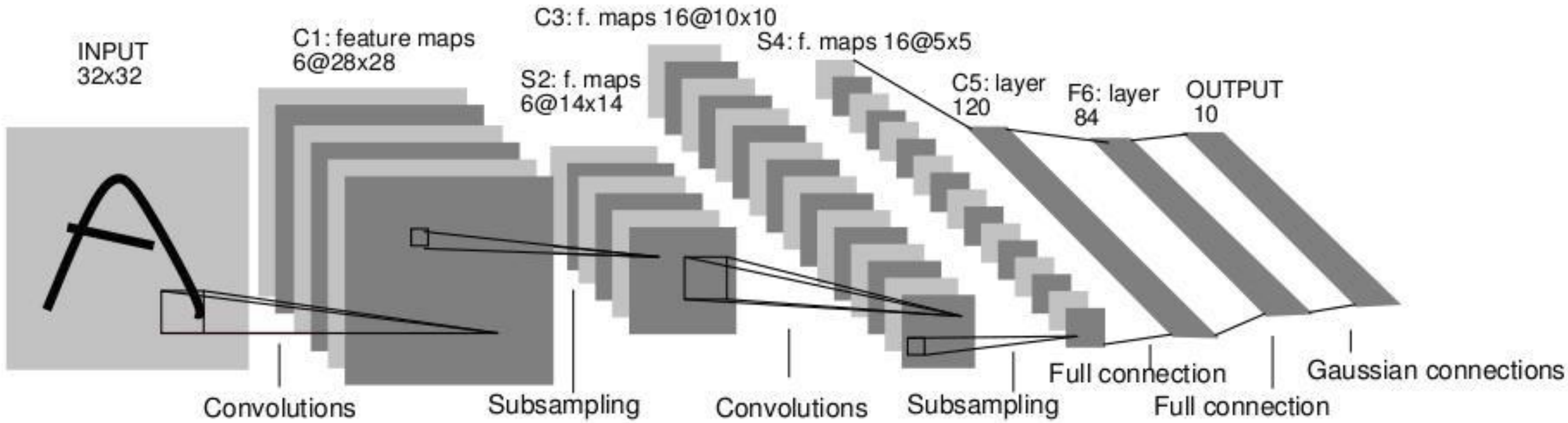
- Convolutional layers
  - Filter weights are shared between locations

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

Topics of This Lecture

- Recap: Tricks of the Trade
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

CNN Architectures: LeNet (1998)



- Early convolutional architecture
  - 2 Convolutional layers, 2 pooling layers
  - Fully-connected NN layers for classification
  - Successfully used for handwritten digit recognition (MNIST)

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.

ImageNet Challenge 2012

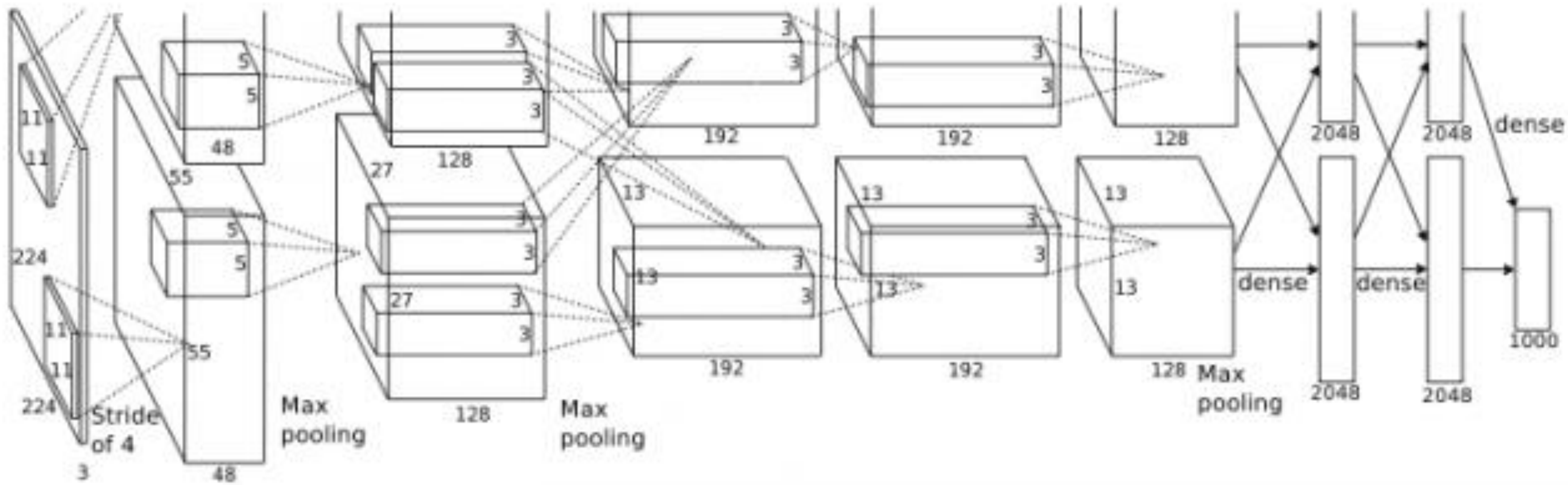
- ImageNet
  - ~14M labeled internet images
  - 20k classes
  - Human labels via Amazon Mechanical Turk

IMAGENET

[Deng et al., CVPR'09]

- Challenge (ILSVRC)
  - 1.2 million training images
  - 1000 classes
  - Goal: Predict ground-truth class within top-5 responses
  - Currently one of the top benchmarks in Computer Vision

CNN Architectures: AlexNet (2012)

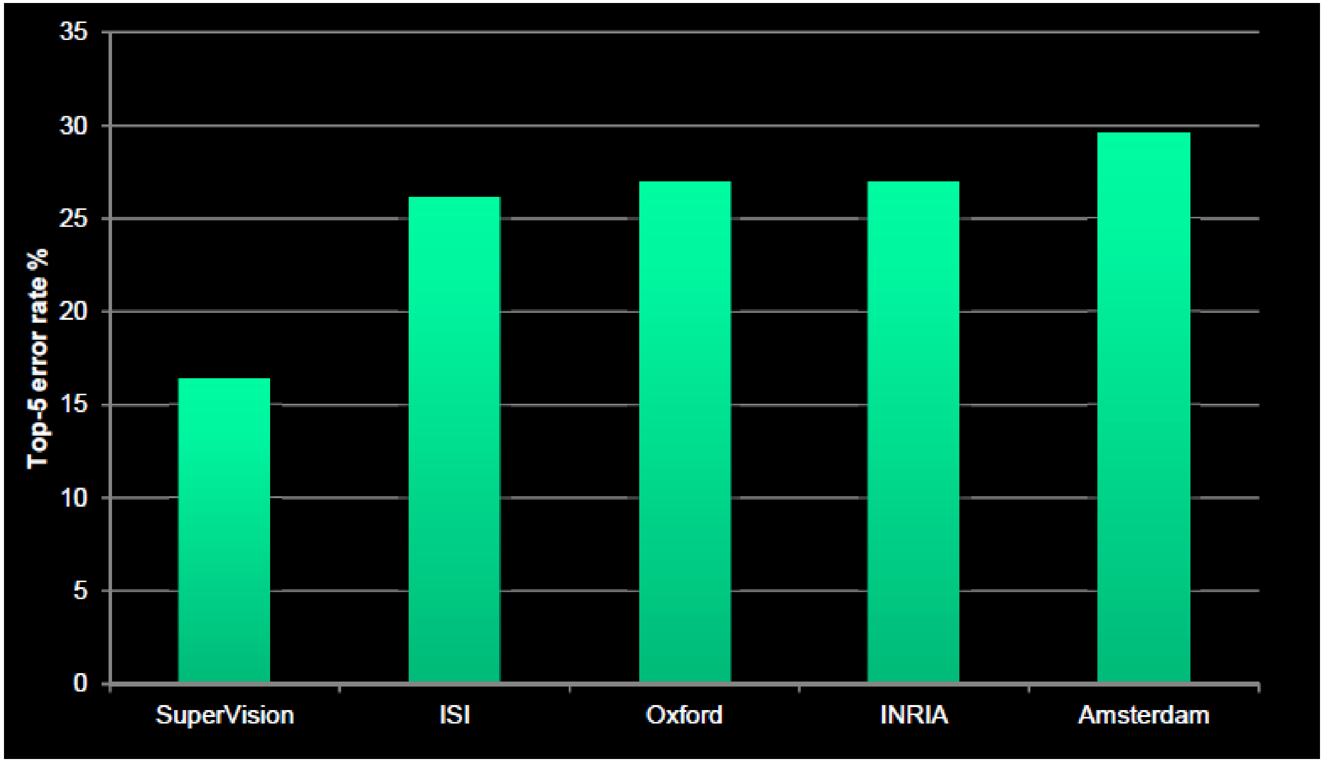


- Similar framework as LeNet, but
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data (10^6 images instead of 10^3)
  - GPU implementation
  - Better regularization and up-to-date tricks for training (Dropout)

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

Image source: A. Krizhevsky, I. Sutskever and G.E. Hinton, NIPS 2012

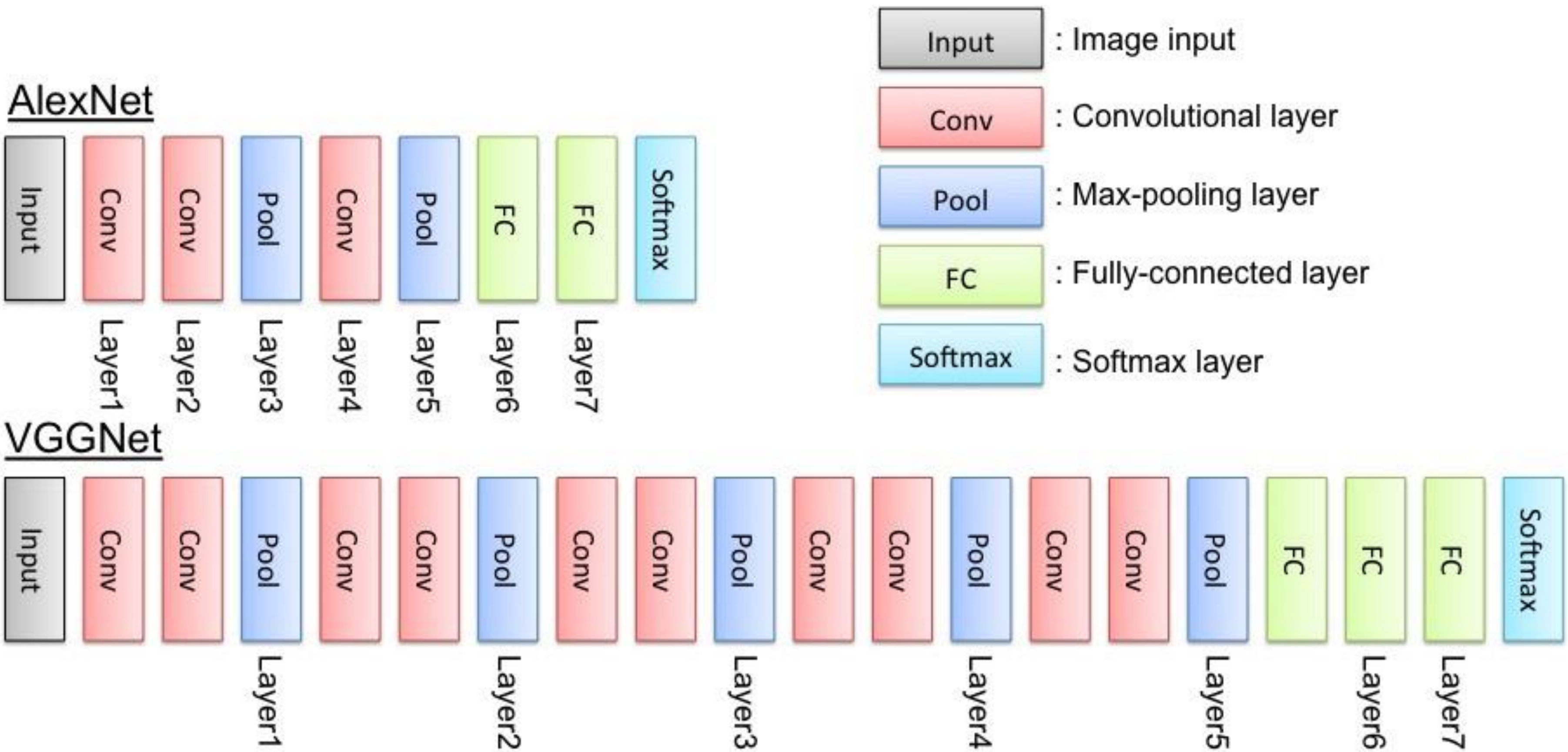
ILSVRC 2012 Results



- AlexNet almost halved the error rate
  - 16.4% error (top-5) vs. 26.2% for the next best approach

    ⇒ A revolution in Computer Vision
  - Acquired by Google in Jan '13, deployed in Google+ in May '13

CNN Architectures: VGGNet (2014/15)



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

Image source: Hirokatsu Kataoka

CNN Architectures: VGGNet (2014/15)

- Main ideas
  - Deeper network
  - Stacked convolutional layers with smaller filters (+ nonlinearity)
  - Detailed evaluation of all components
- Results
  - Improved ILSVRC top-5 error rate to 6.7%.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

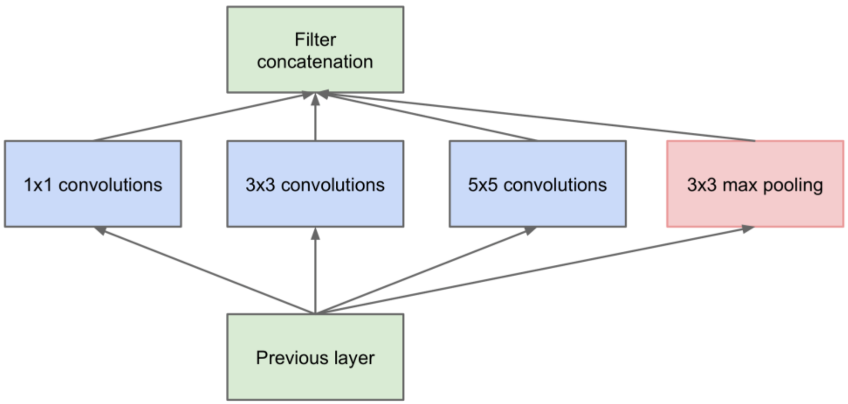
Mainly used

Image source: Simonyan & Zisserman

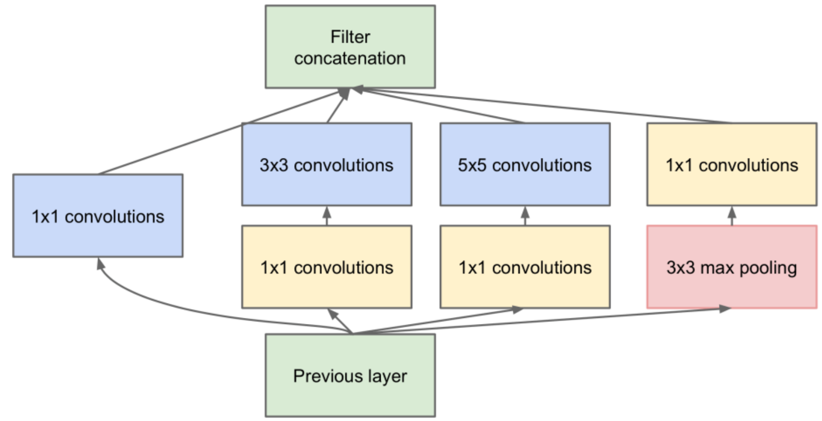
Comparison: AlexNet vs. VGGNet

- Receptive fields in the first layer
  - AlexNet: 11×11, stride 4
  - Zeiler & Fergus: 7×7, stride 2
  - VGGNet: 3×3, stride 1
- Why that?
  - If you stack a 3×3 on top of another 3×3 layer, you effectively get a 5×5 receptive field.
  - With three 3×3 layers, the receptive field is already 7×7.
  - But much fewer parameters: $3 \cdot 3^2 = 27$ instead of $7^2 = 49$.
  - In addition, non-linearities in-between 3×3 layers for additional discriminativity.

CNN Architectures: GoogLeNet (2014/2015)



(a) Inception module, naïve version

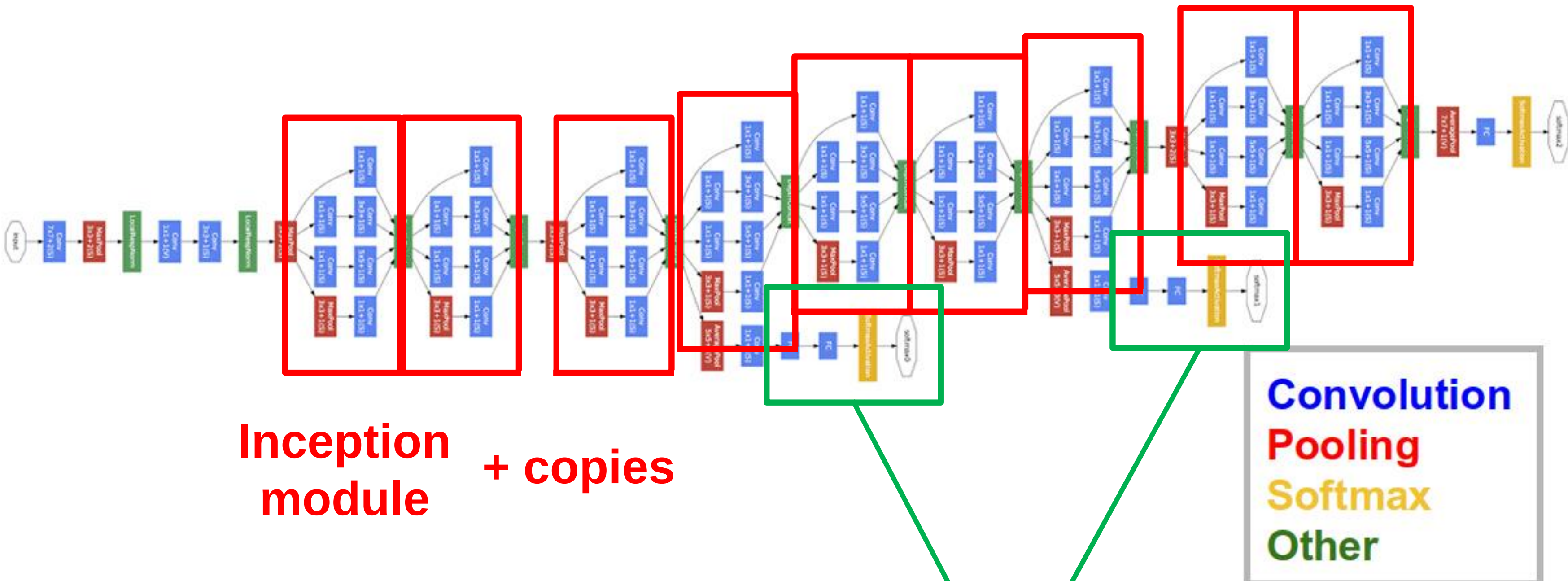


(b) Inception module with dimension reductions

- Main ideas
  - "Inception" module as modular component
  - Learns filters at several scales within each module

C. Szegedy, W. Liu, Y. Jia, et al, Going Deeper with Convolutions, arXiv:1409.4842, 2014, CVPR'15, 2015.

GoogLeNet Visualization



Inception module

+ copies

Auxiliary classification outputs for training the lower layers (deprecated)

Results on ILSVRC

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

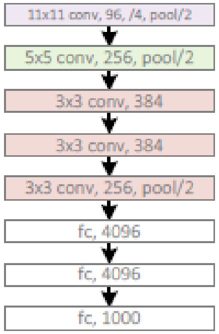
- VGGNet and GoogLeNet perform at similar level
  - Comparison: human performance ~5% [Karpathy]

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

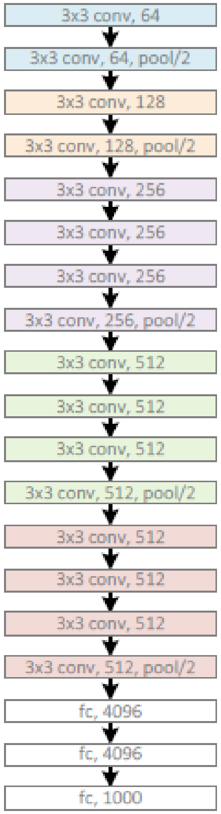
Image source: Simonyan & Zisserman

Newer Developments: Residual Networks

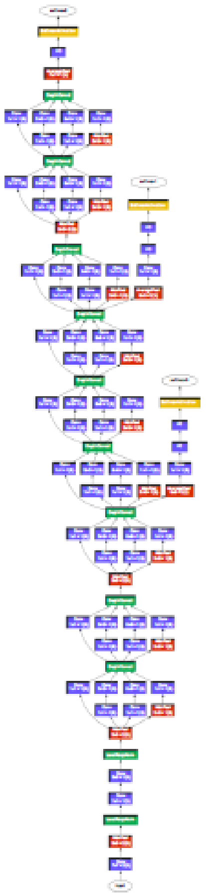
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

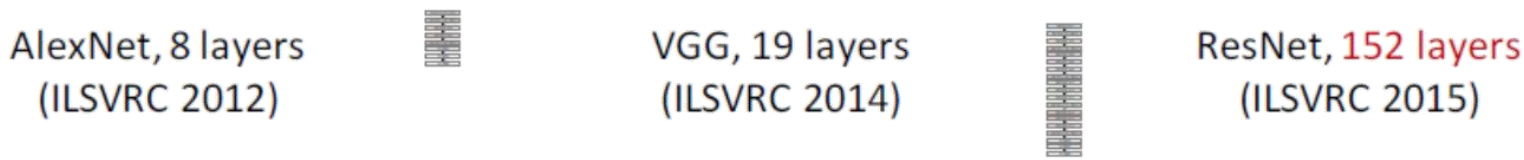


GoogleNet, 22 layers
(ILSVRC 2014)

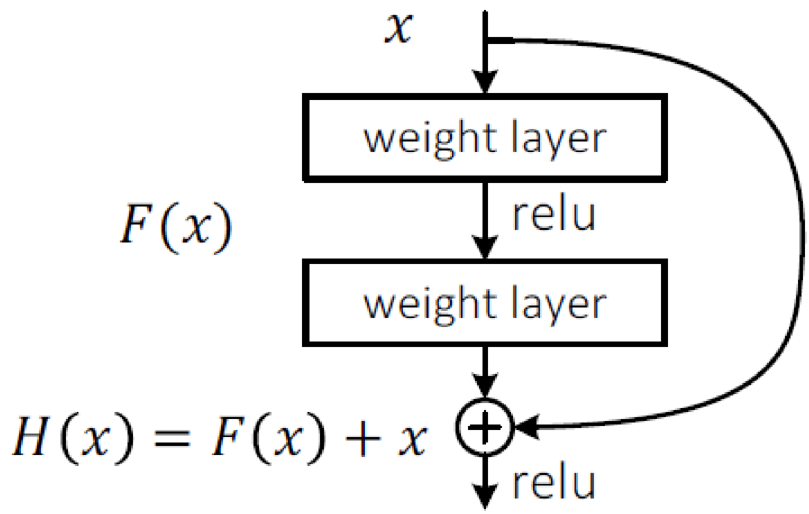


RWTH AACHEN UNIVERSITY

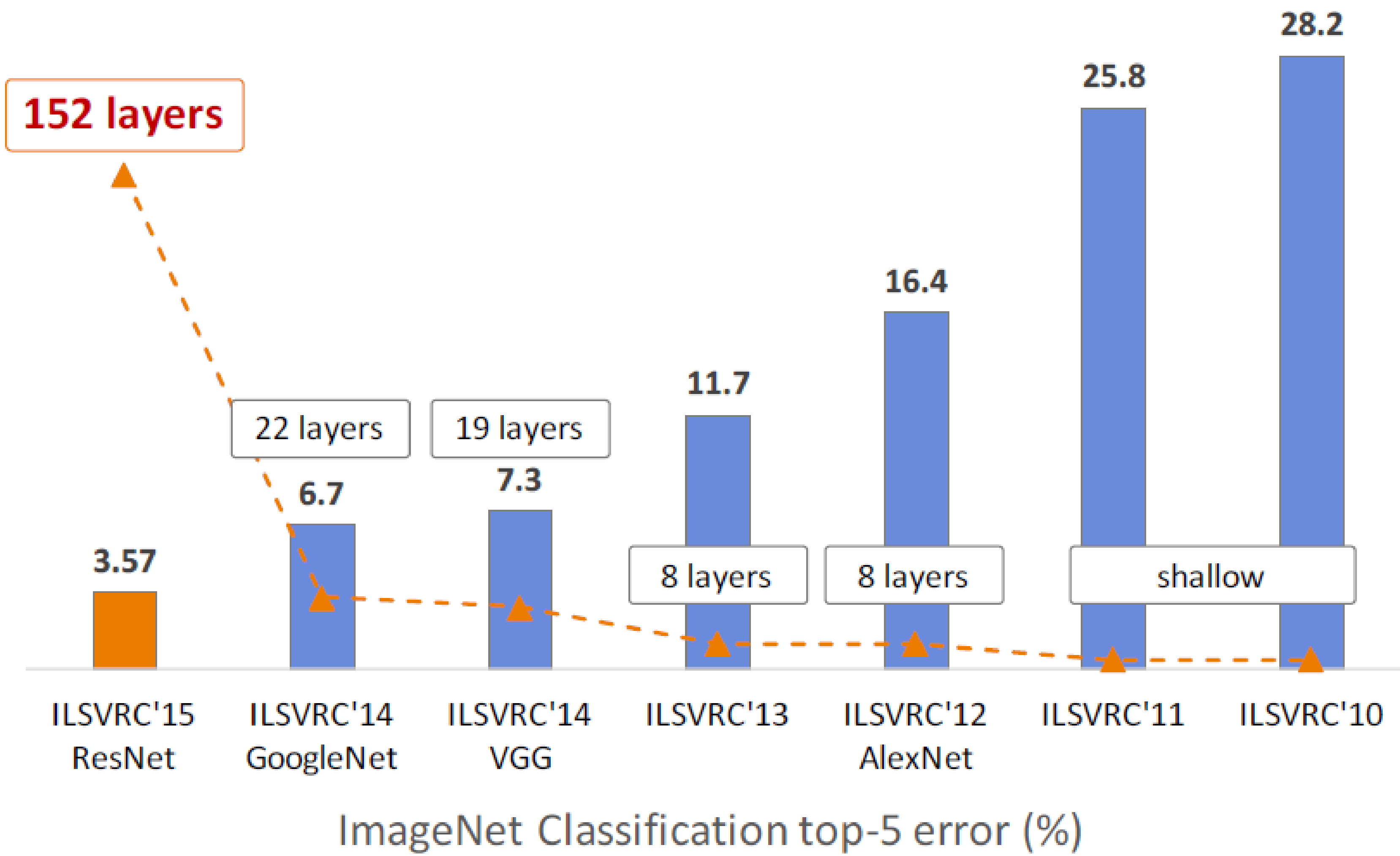
Newer Developments: Residual Networks



- Core component
  - Skip connections bypassing each layer
  - Better propagation of gradients to the deeper layers
  - We'll analyze this mechanism in more detail later…



ImageNet Performance



Understanding the ILSVRC Challenge

- Imagine the scope of the problem!
  - 1000 categories
  - 1.2M training images
  - 50k validation images

- This means...
  - Speaking out the list of category names at 1 word/s...
    ...takes 15mins.
  - Watching a slideshow of the validation images at 2s/image...
    ...takes a full day (24h+).
  - Watching a slideshow of the training images at 2s/image...
    ...takes a full month.

rier, Airedale, airliner, airship, albatross, alligator lizard, alp, altar, ambulance, American alligator, American black bear, American chameleon, American coot, American egret, American lobster, American Staffordshire terrier, amphibian, analog clock, anemone fish, Angora, ant, apiary, Appenzeller, apron, Arabian camel, Arctic fox, armadillo, artichoke, ashcan, assault rifle, Australian terrier, axolotl, baboon, backpack, badger, bagel, bakery, balance beam, bald eagle, balloon, ballplayer, ballpoint, banana, Band Aid, banded gecko, banjo, bannister, barbell, barber chair, barbershop, barn, barn spider, barometer, barracouta, barrel, barrow, baseball, basenji, basketball, basset, bassinet, bassoon, bath towel, bathing cap, bathtub, beach wagon, beacon, beagle, beaker, bearskin, beaver, Bedlington terrier, bee, bee eater, beer bottle, beer glass, bell cote, bell pepper, Bernese mountain dog, bib, bicycle-built-for-two, bighorn, bikini, binder, binoculars, birdhouse, bison, bittern, black and gold garden spider, black grouse, black stork, black swan, black widow, black-and-tan coonhound, black-footed ferret, Blenheim spaniel, bloodhound, bluetick, boa constrictor, boathouse, bobsled, bolete, bolo tie, bonnet, book jacket, bookcase, bookshop, Border collie, Border terrier, borzoi, Boston bull, bottlecap, Bouvier des Flandres, bow, bow tie, box turtle, boxer, Brabancon griffon, brain coral, brambling, brass, brassiere, breakwater, breastplate, briard, Brittany spaniel, broccoli, broom, brown bear, bubble, bucket, buckeye, buckle, bulbul, bull mastiff, bullet train, bulletproof vest, bullfrog, burrito, bustard, butcher shop, butternut squash, cab, cabbage butterfly, cairn, caldron, can opener, candle, cannon, canoe, capuchin, car mirror, car wheel, carbonara, Cardigan, cardigan, cardoon, carousel, carpenter's kit, carton, cash machine, cassette, cassette player, castle, catamaran, cauliflower, CD player, cello, cellular telephone, centipede, chain, chain mail, chain saw, chain-link fence, chambered nautilus, cheeseburger, cheetah, Chesapeake Bay retriever, chest, chickadee, chiffonier, Chihuahua, chime, chimpanzee, china cabinet, chiton, chocolate sauce, chow, Christmas stocking, church, cicada, cinema, cleaver, cliff, cliff dwelling, cloak, clog, clumber, cock, cocker spaniel, cockroach, cocktail shaker, coffee mug, coffeepot, coho, coil, collie, colobus, combination lock, comic book, common iguana, common newt, computer keyboard, conch, confectionery, consomme, container ship, convertible, coral fungus, coral reef, corkscrew, corn, cornet, coucal, cougar, cowboy boot, cowboy hat, coyote, cradle, crane, crane, crash helmet, crate, crayfish, crib, cricket, Crock Pot, croquet ball, crossword puzzle, crutch, cucumber, cuirass, cup, curly-coated retriever, custard apple, daisy, dalmatian, dam, damselfly, Dandie Dinmont, desk, desktop computer, dhole, dial telephone, diamondback, diaper, digital clock, digital watch, dingo, dining table, dishrag, dishwasher, disk brake, Doberman, dock, dogsled, dome, doormat, dough, dowitcher, dragonfly, drake, drilling platform, drum, drumstick, dugong, dumbbell, dung beetle, Dungeness crab, Dutch oven, ear, earthstar, echidna, eel, eft, eggnog, Egyptian cat, electric fan, electric guitar, electric locomotive, electric ray, English foxhound, English setter, English springer, entertainment center, EntleBucher, envelope, Eskimo dog, espresso, espresso maker, European fire salamander, European gallinule, face powder, feather boa, fiddler crab, fig, file, fire engine, fire screen, fireboat, flagpole, flamingo, flat-coated retriever, flatworm, flute, fly, folding chair, football helmet, forklift, fountain, fountain pen, four-poster, fox squirrel, freight car, French bulldog, French horn, French loaf, frilled lizard, frying pan, fur coat, gar, garbage truck, garden spider, garter snake, gas pump, gasmask, gazelle, German shepherd, German short-haired pointer, geyser, giant panda, giant schnauzer, gibbon, Gila monster, go-kart, goblet, golden retriever, goldfinch, goldfish, golf ball, golfcart, gondola, gong, goose, Gordon setter, gorilla, gown, grand piano, Granny Smith, grasshopper, Great Dane, great grey owl, Great Pyrenees, great white shark,

More Finegrained Classes

Image source: O. Russakovsky et al.

Quirks and Limitations of the Data Set

- Generated from WordNet ontology
  - Some animal categories are overrepresented
  - E.g., 120 subcategories of dog breeds

⇒ 6.7% top-5 error looks all the more impressive

Image source: A. Karpathy

References and Further Reading

- LeNet
  - 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.
- AlexNet
  - A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.
- VGGNet
  - K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015
- GoogLeNet
  - C. Szegedy, W. Liu, Y. Jia, et al, Going Deeper with Convolutions, arXiv:1409.4842, 2014.

References and Further Reading

- ResNet
  - K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016.