Computer Vision – Lecture 11

Deep Learning II

28.05.2019

Bastian Leibe

Visual Computing Institute
RWTH Aachen University
http://www.vision.rwth-aachen.de/

leibe@vision.rwth-aachen.de

Course Outline

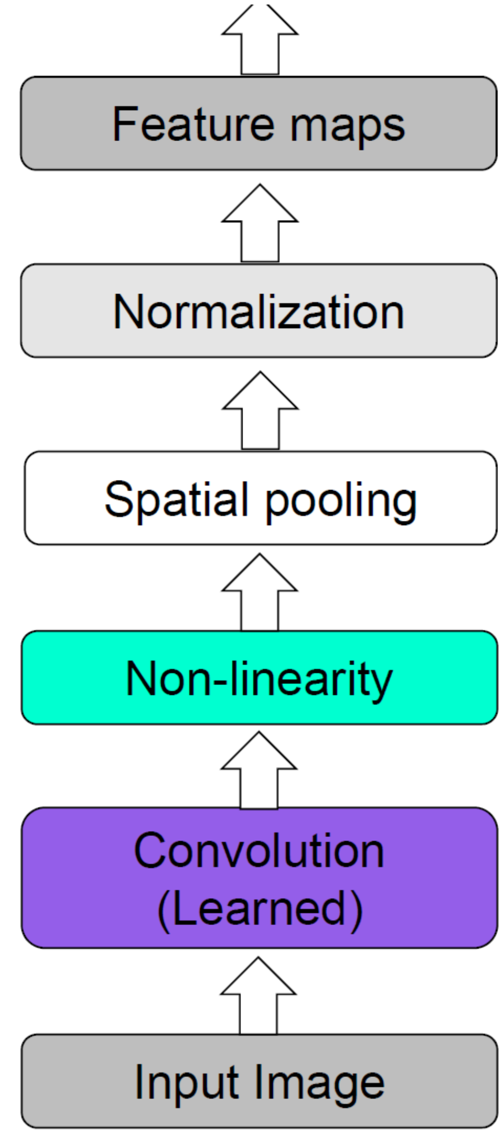
- Image Processing Basics
- Segmentation & Grouping
- Object Recognition & Categorization
  - Sliding Window based Object Detection
- Local Features & Matching
  - Local Features – Detection and Description
  - Recognition with Local Features
- Deep Learning
  - Convolutional Neural Networks
- 3D Reconstruction

Topics of This Lecture

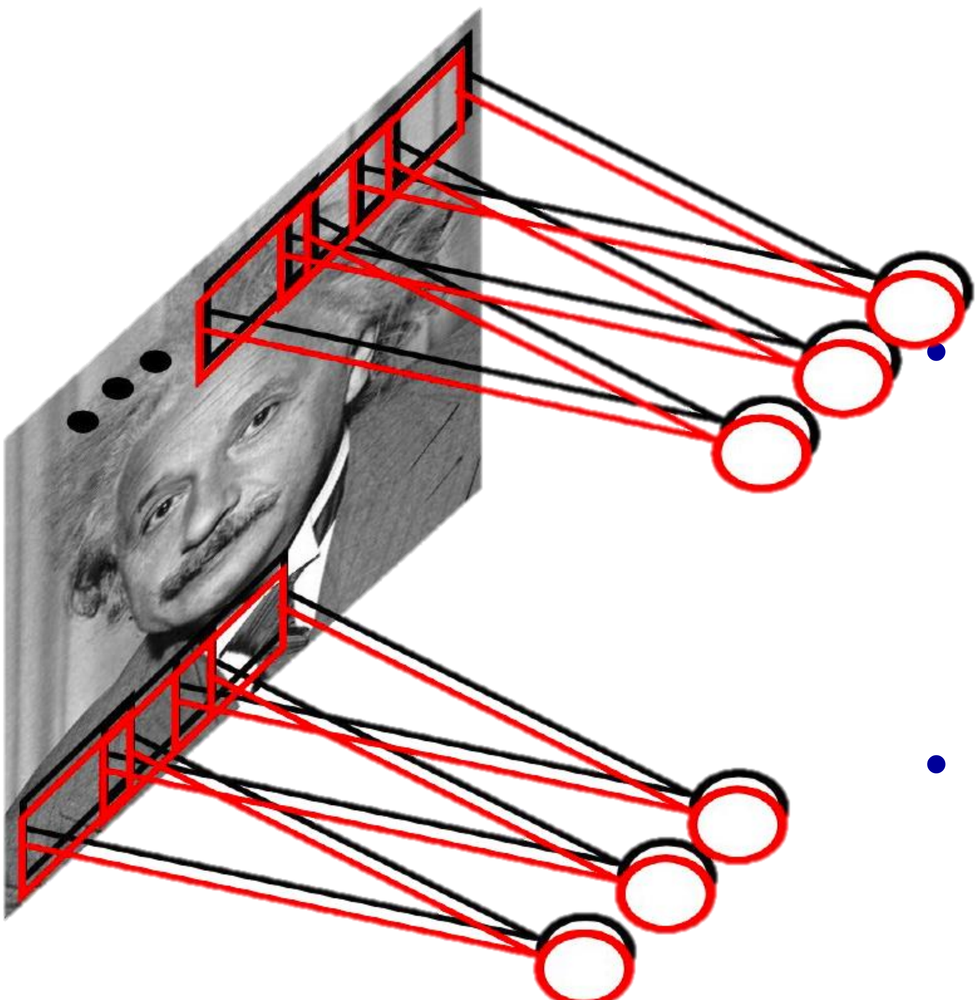
- Recap: Convolutional Neural Networks
  - Convolutional Layers
  - Pooling Layers
  - Nonlinearities
- Background: Deep Learning
  - Recap from ML lecture
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
  - ResNet

Recap: CNN Structure

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



Recap: Intuition of CNNs

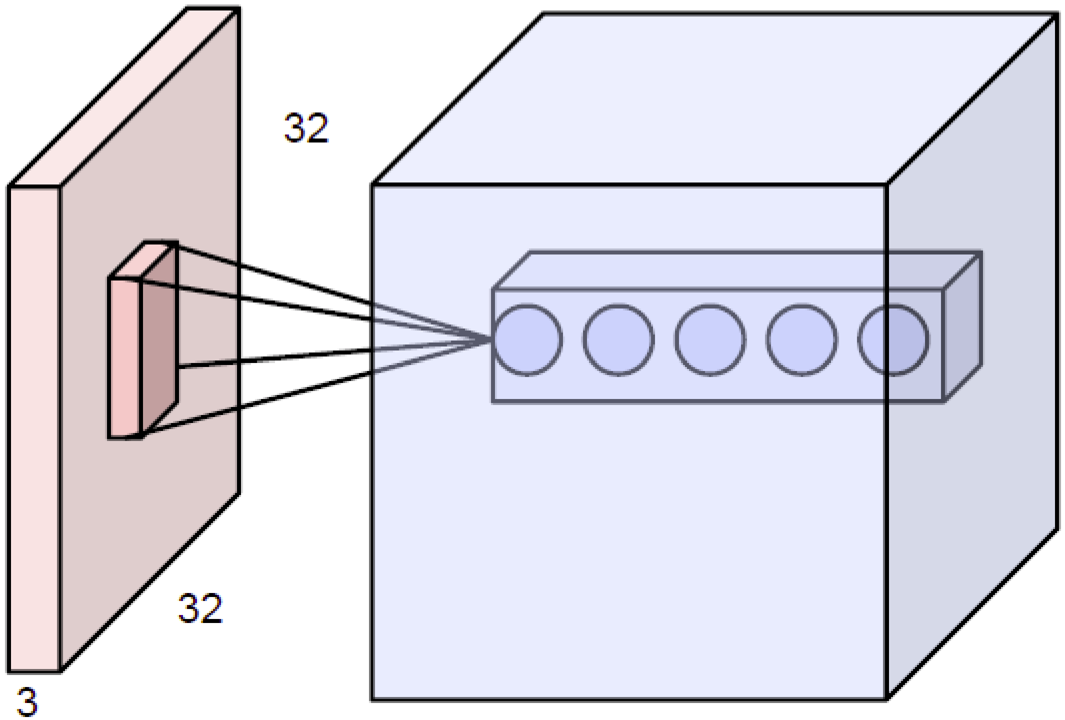


- Convolutional net
  - Share the same parameters across different locations
  - Convolutions with learned kernels
- Learn *multiple* filters
  - E.g. 1000$\times$1000 image
    100 filters
    10$\times$10 filter size

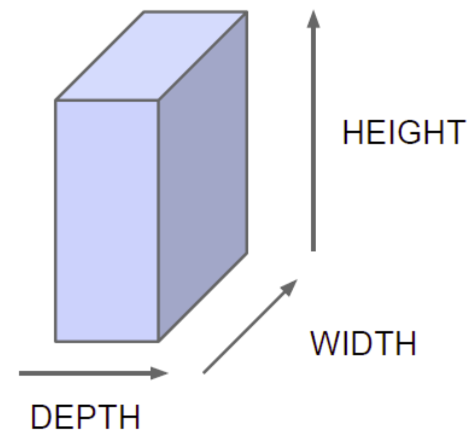
  $\Rightarrow$ 10k parameters
- Result: Response map
  - size: 1000$\times$1000$\times$100
  - Only memory, not params!

Image source: Yann LeCun

Recap: Convolution Layers



Naming convention:

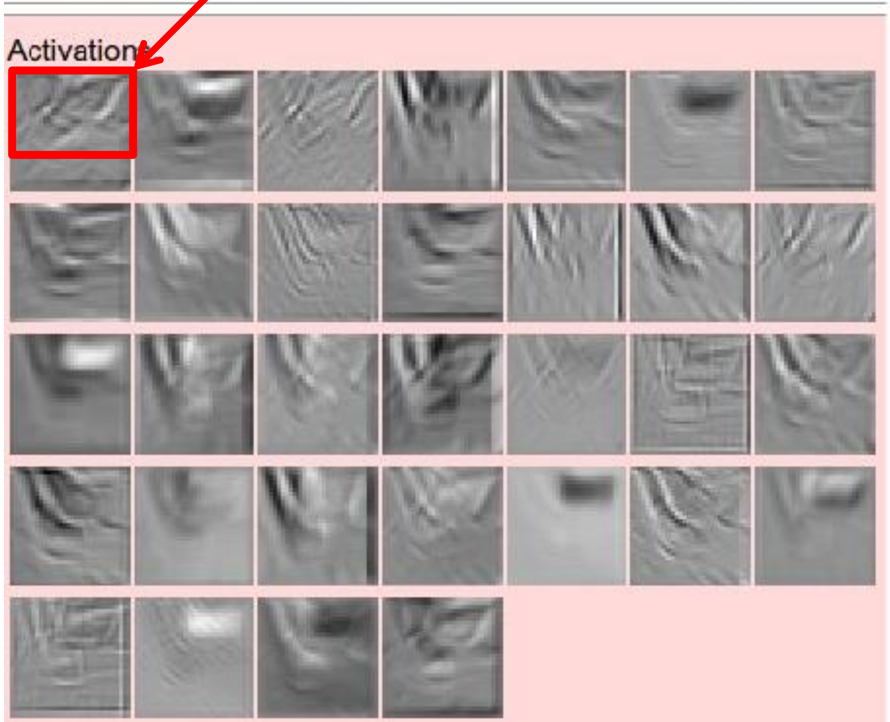


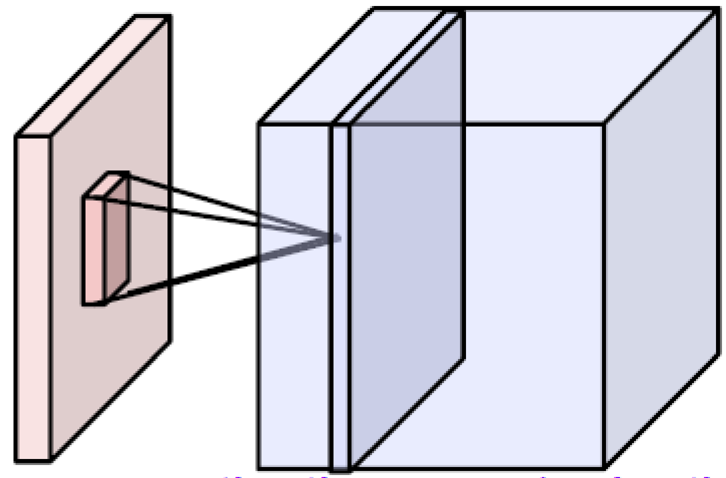
- All Neural Net activations arranged in 3 dimensions
  - Multiple neurons all looking at the same input region, stacked in depth
  - Form a single [$1\times1\times$depth] depth column in output volume.

Slide credit: FeiFei Li, Andrej Karpathy

Recap: Activation Maps

5×5 filters

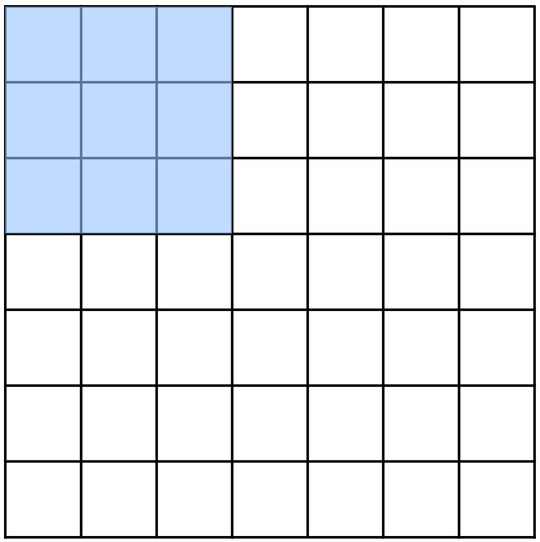




Each activation map is a depth slice through the output volume.

Activation maps

Convolution Layers

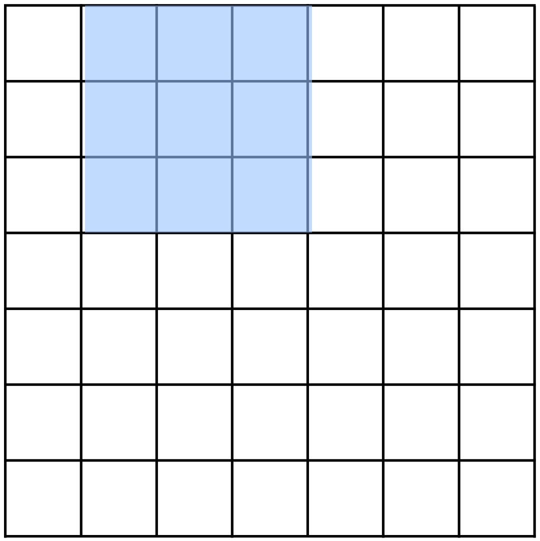


Example:
7×7 input
assume 3×3 connectivity
stride 1

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

Slide credit: FeiFei Li, Andrej Karpathy

Convolution Layers

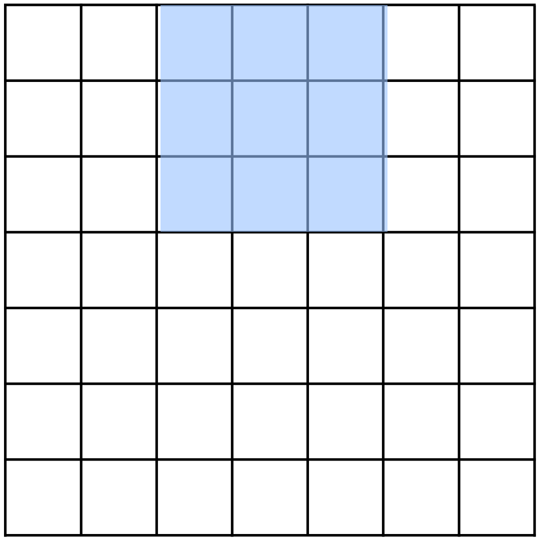


Example:
7×7 input
assume 3×3 connectivity
stride 1

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

Slide credit: FeiFei Li, Andrej Karpathy

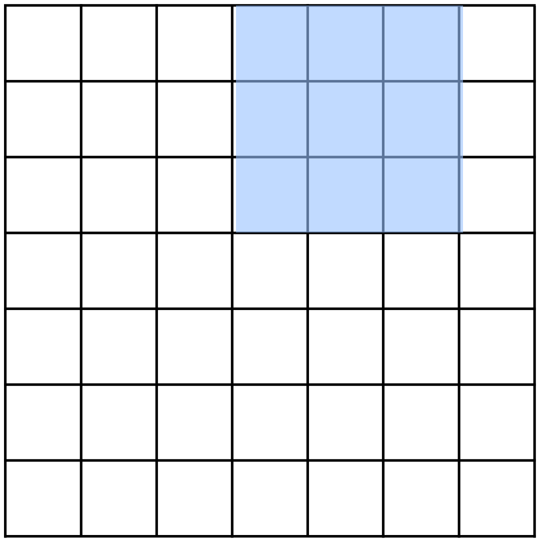
Convolution Layers



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

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

Convolution Layers

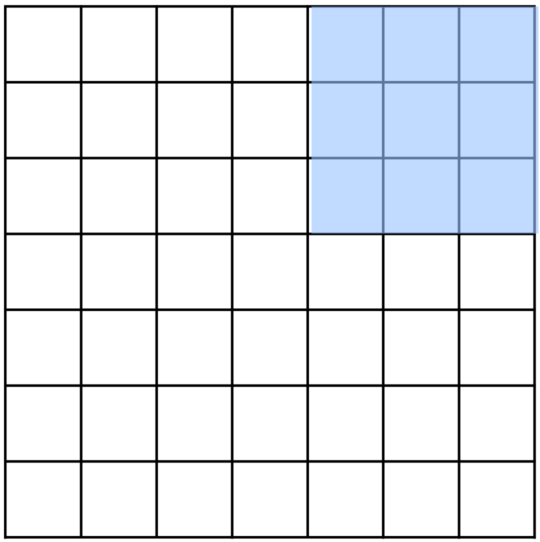


Example:
7×7 input
assume 3×3 connectivity
stride 1

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

Slide credit: FeiFei Li, Andrej Karpathy

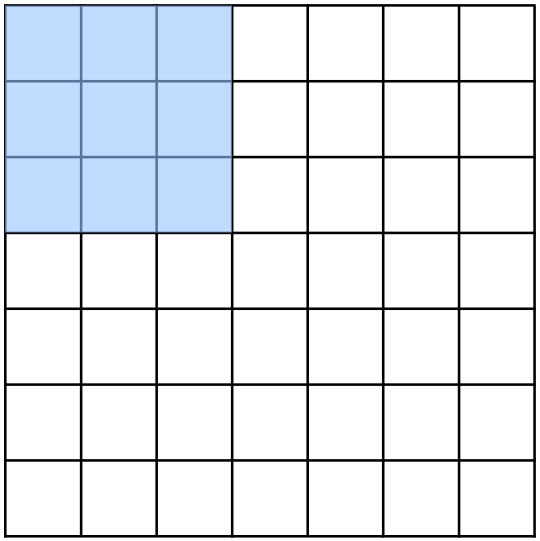
Convolution Layers



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.

Convolution Layers

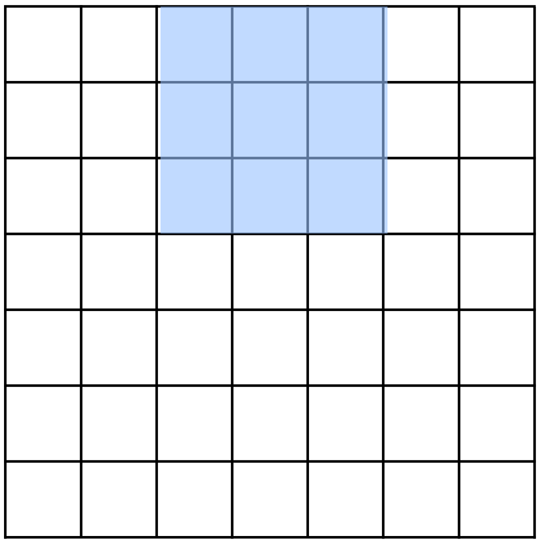


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow 5 \times 5$ output

What about stride 2?

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

Convolution Layers

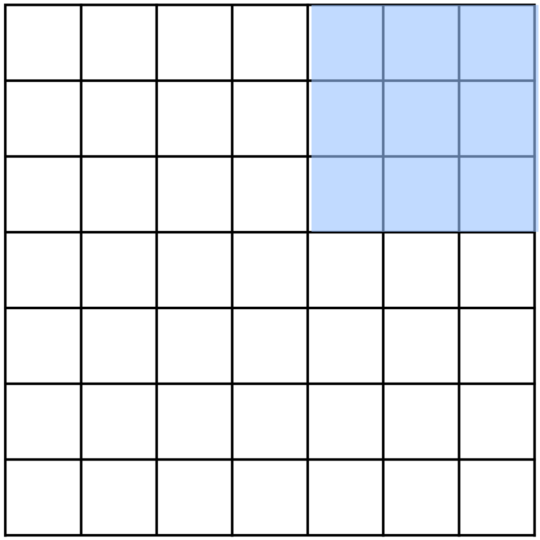


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow 5 \times 5$ output

What about stride 2?

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

Convolution Layers

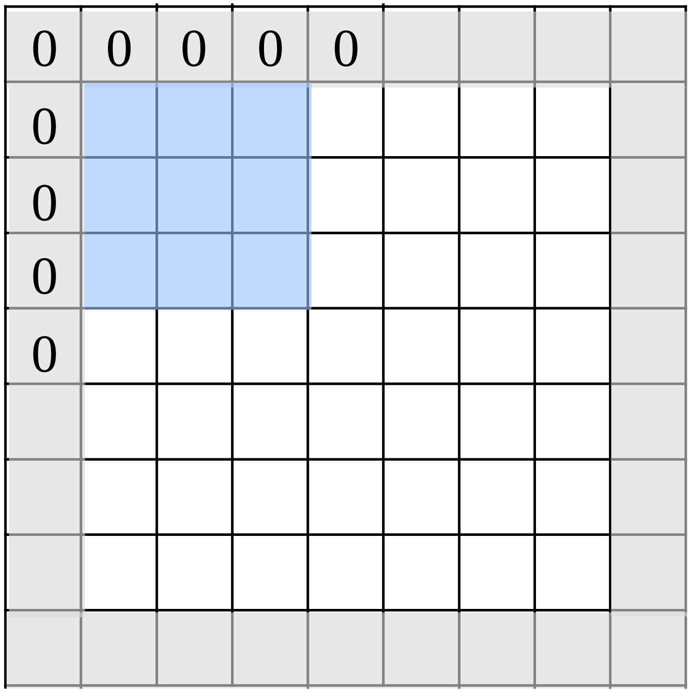


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow$ 5×5 output

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

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

Convolution Layers

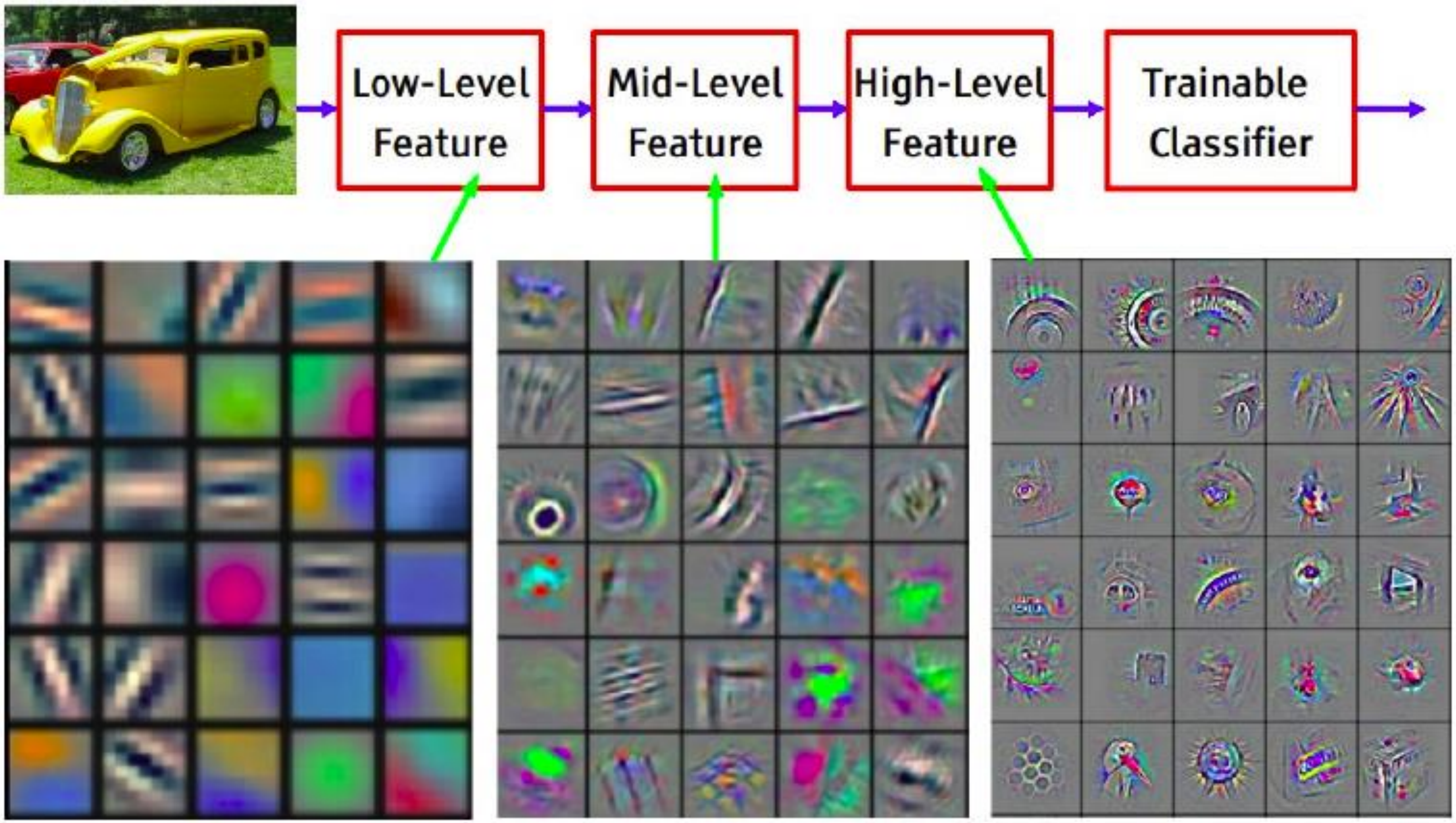


Example:
7×7 input
assume 3×3 connectivity
stride 1
$\Rightarrow 5\times5$ output

What about stride 2?
$\Rightarrow 3\times3$ 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.

Effect of Multiple Convolution Layers

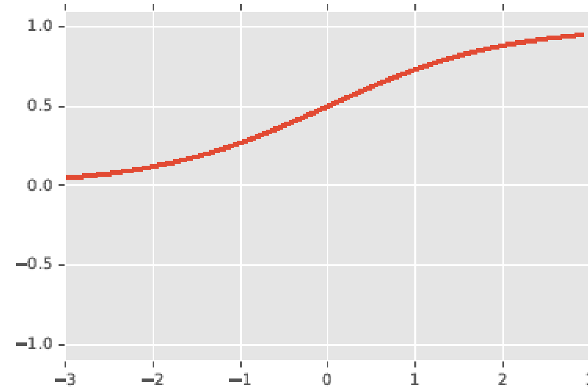


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Commonly Used Nonlinearities

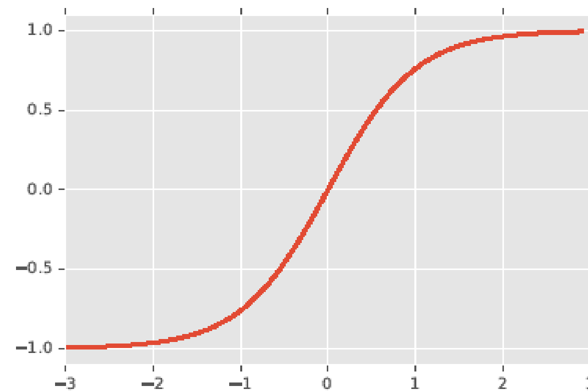
- Sigmoid

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



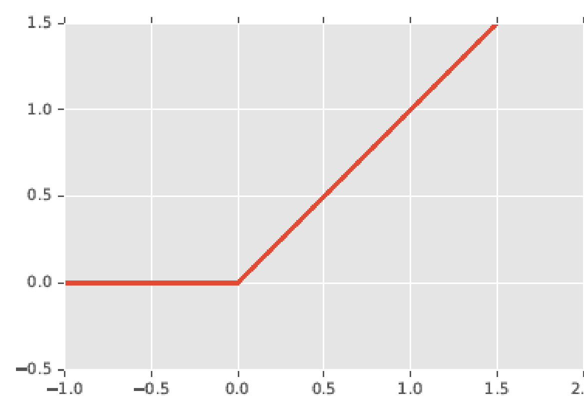
- Hyperbolic tangent

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



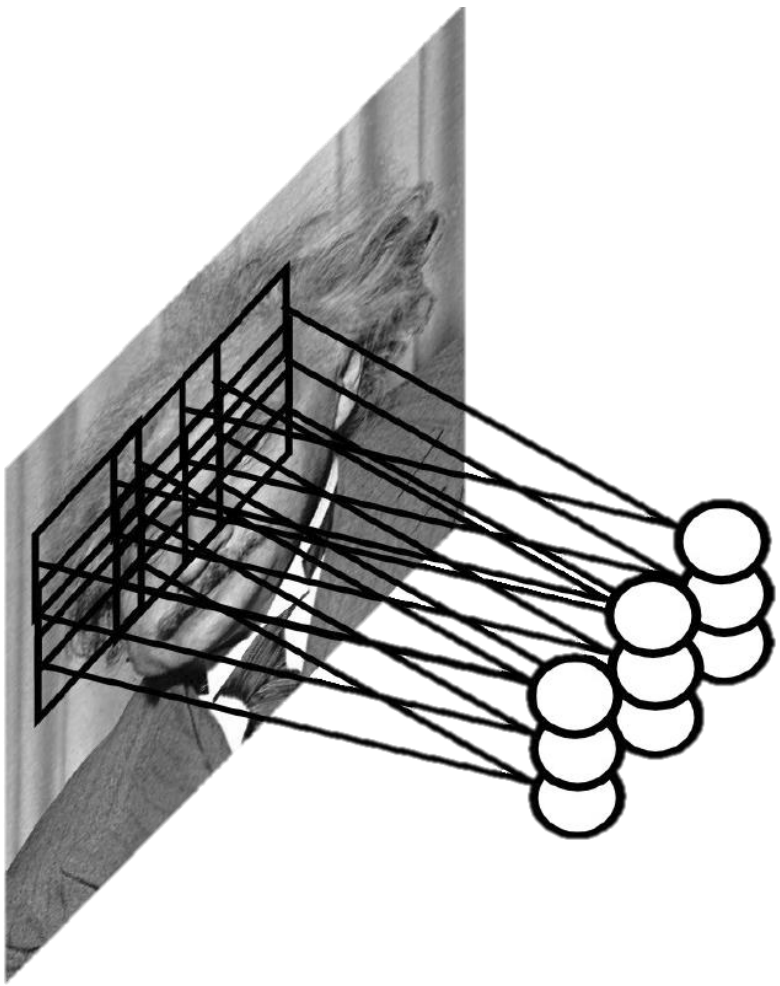
- Rectified linear unit (ReLU)

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



Preferred option for deep networks

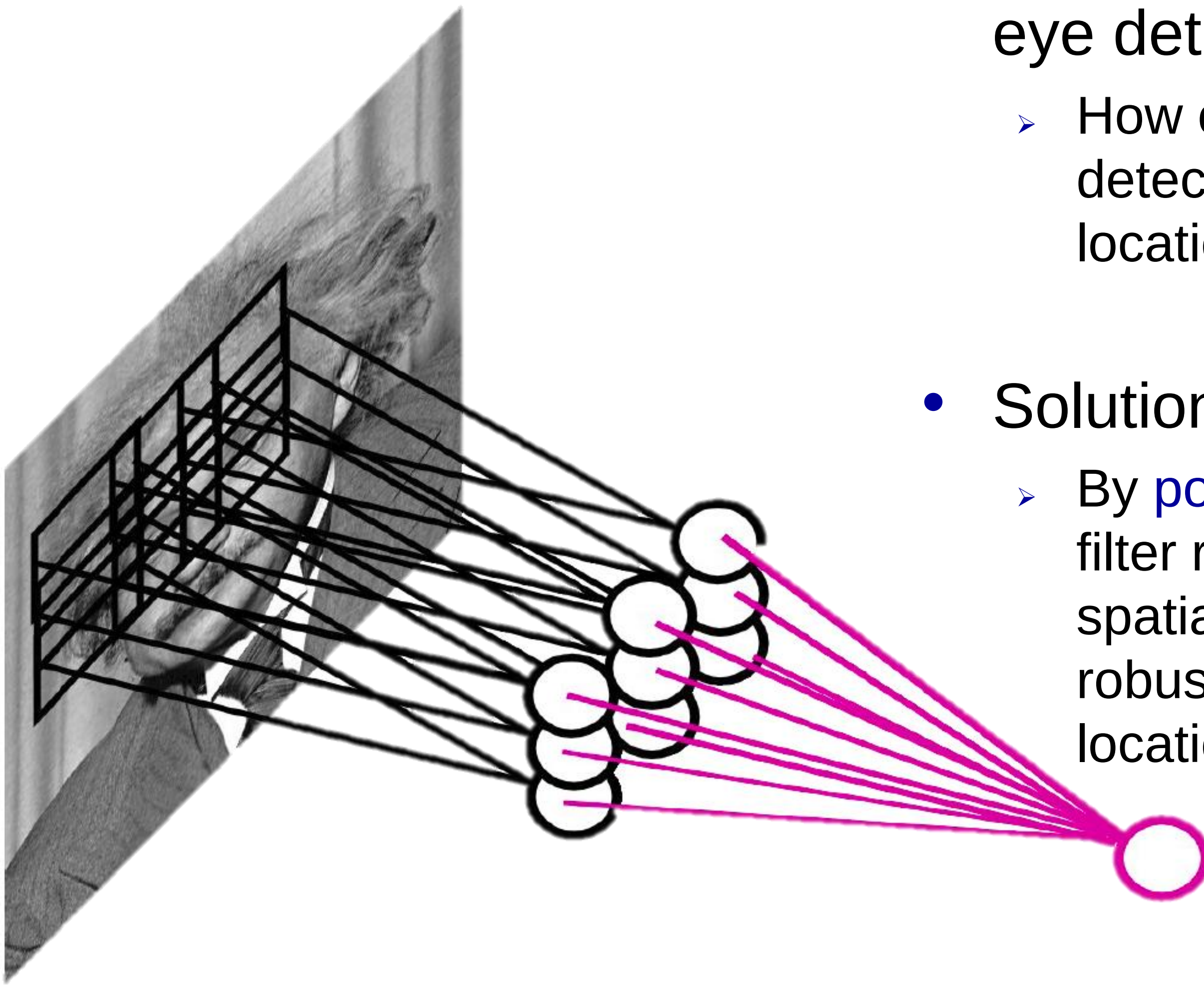
Convolutional Networks: Intuition



- 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

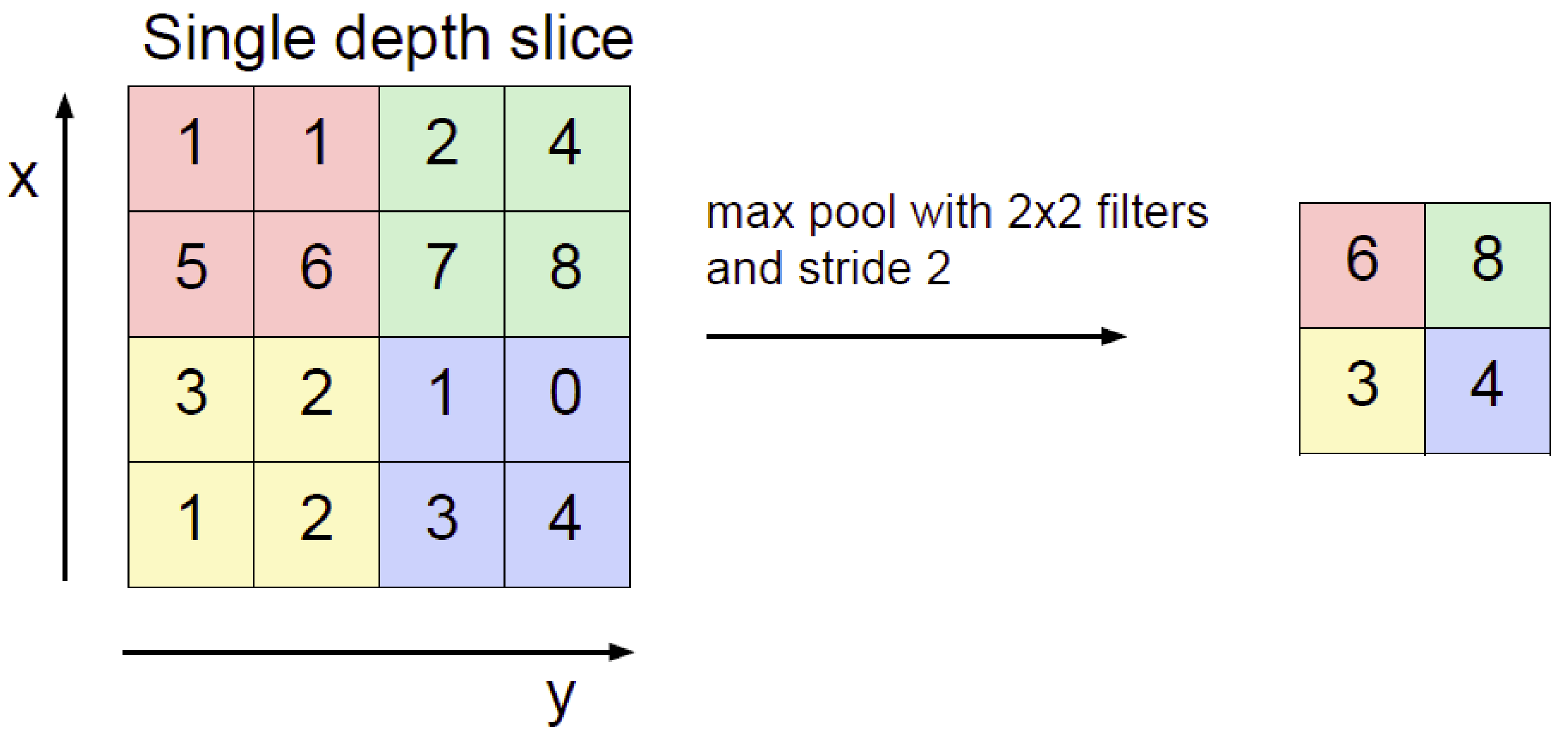
Convolutional Networks: Intuition



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

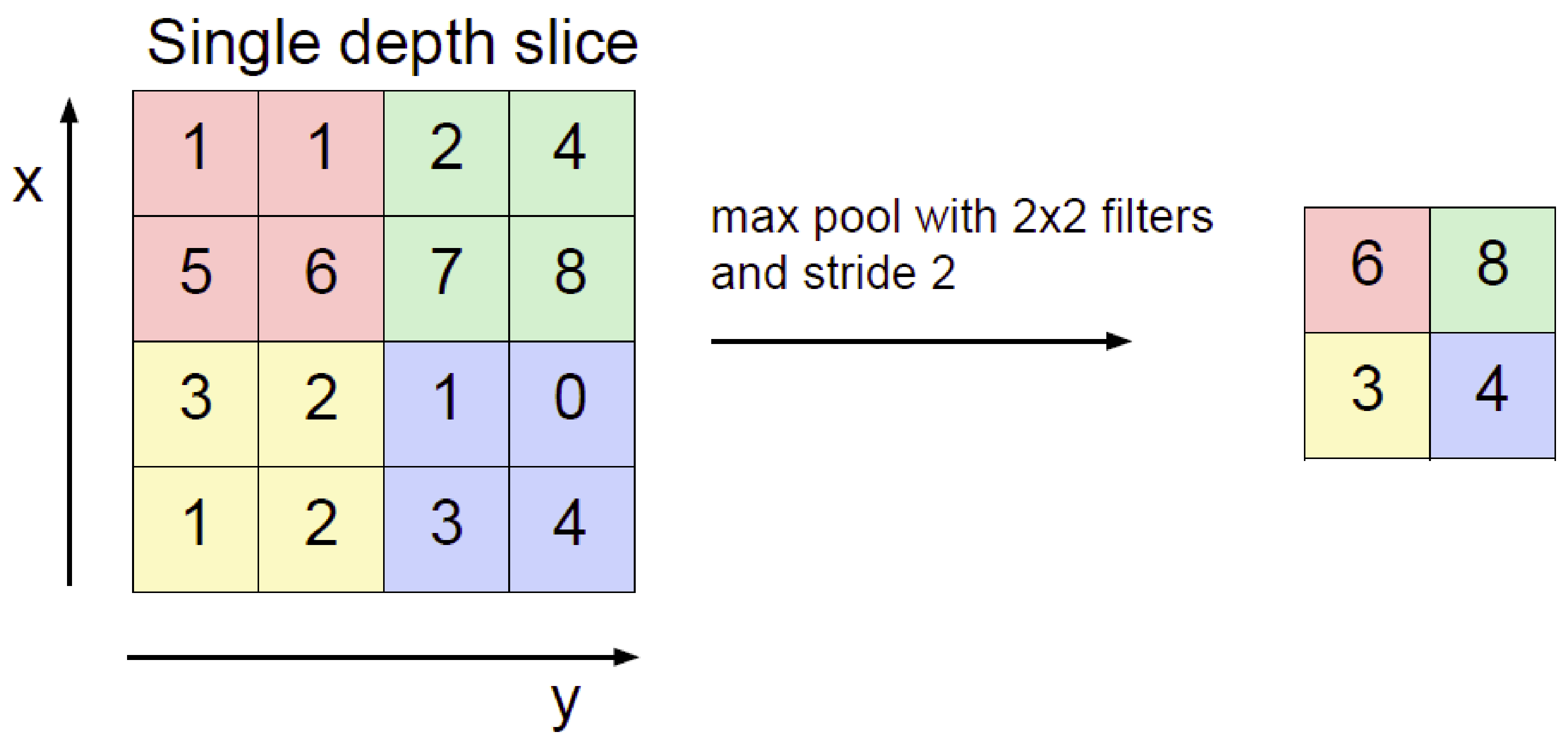
- Solution:
  - By pooling (e.g., max or avg) filter responses at different spatial locations, we gain robustness to the exact spatial location of features.

Max Pooling



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

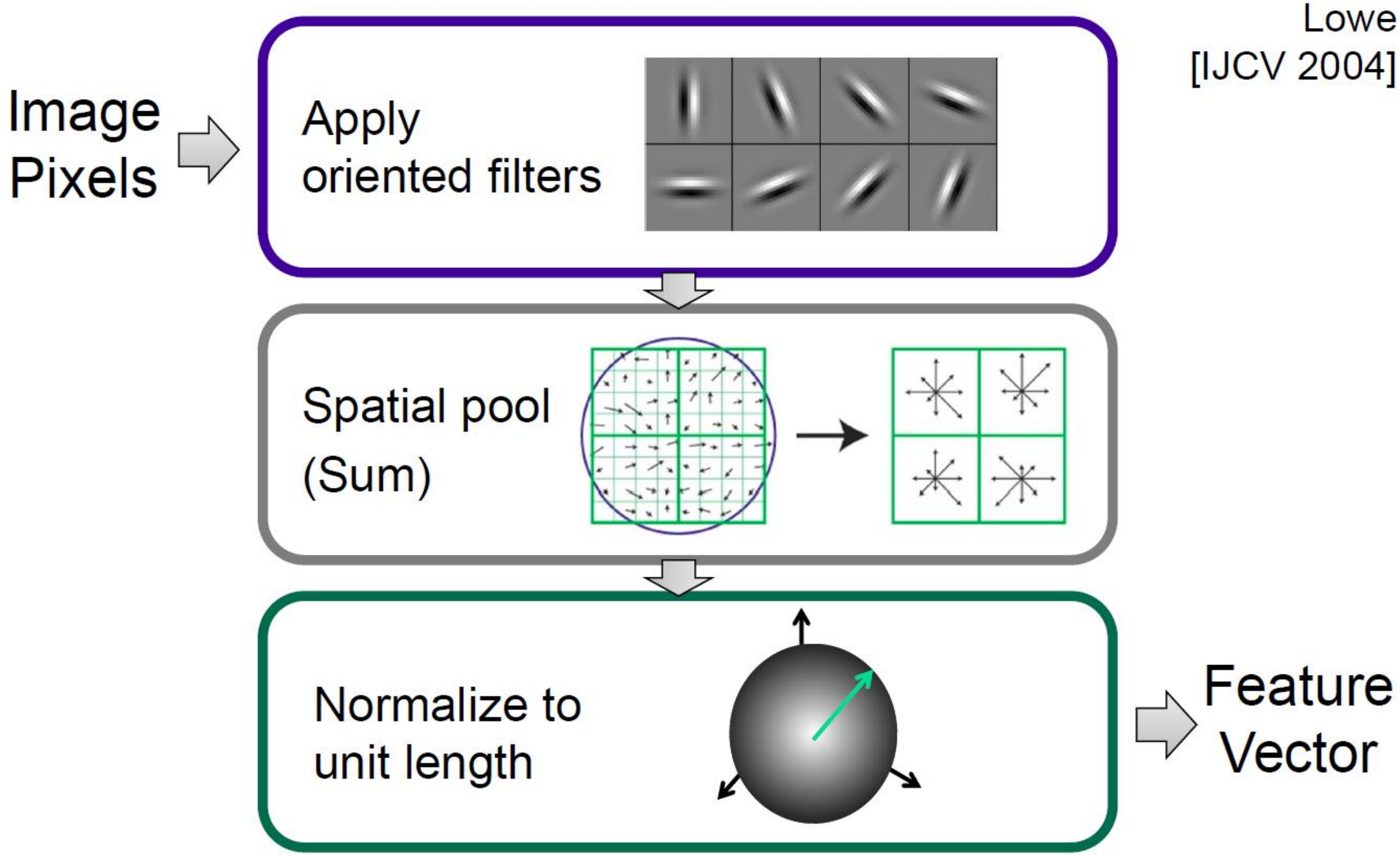
Max Pooling



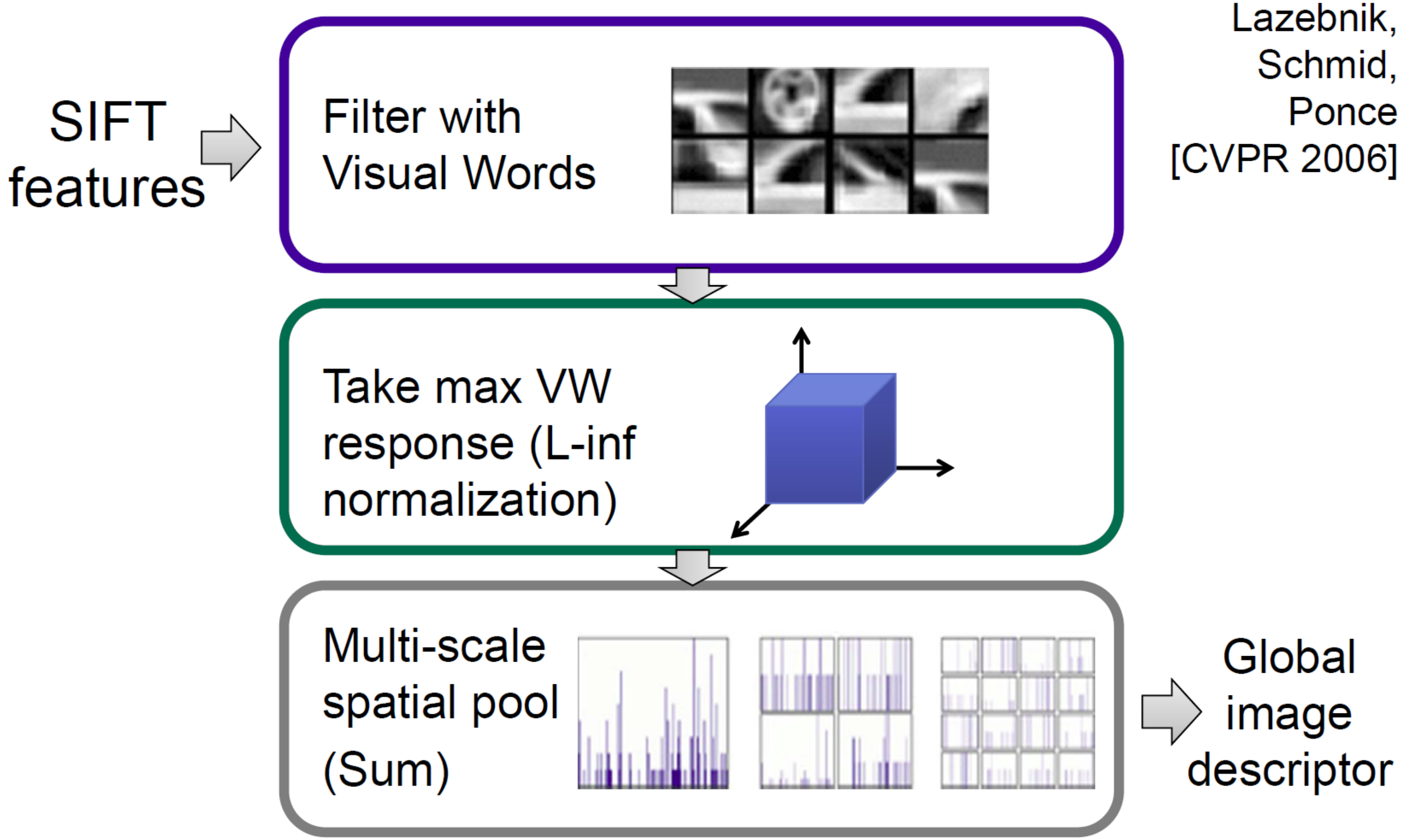
- Note
  - Pooling happens independently across each slice, preserving the number of slices.

Compare: SIFT Descriptor

Lowe [IJCV 2004]



Compare: Spatial Pyramid Matching



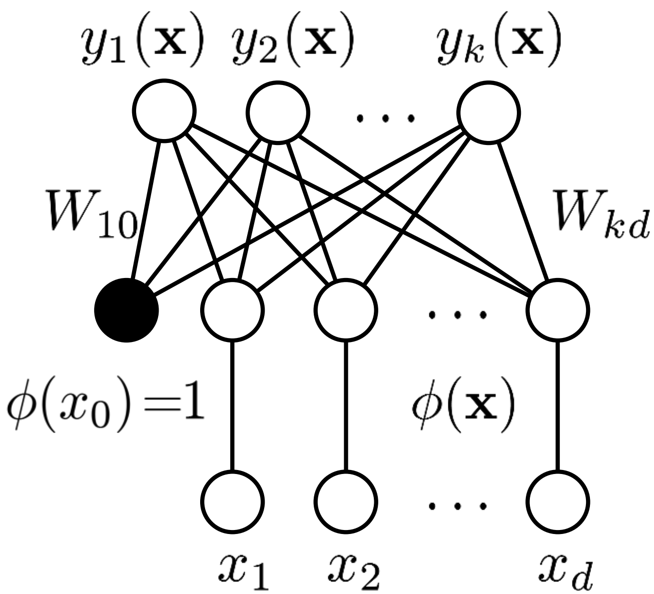
Lazebnik, Schmid, Ponce [CVPR 2006]

Topics of This Lecture

- Recap: Convolutional Neural Networks
  - Convolutional Layers
  - Pooling Layers
  - Nonlinearities
- Background: Deep Learning
  - Recap from ML lecture
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
  - ResNet

Recap: Generalized Linear Discriminants

- Linear classifiers with fixed feature transformation



Output layer

Weights

Feature layer

Mapping (fixed)

Input layer

- Outputs
  - Linear outputs

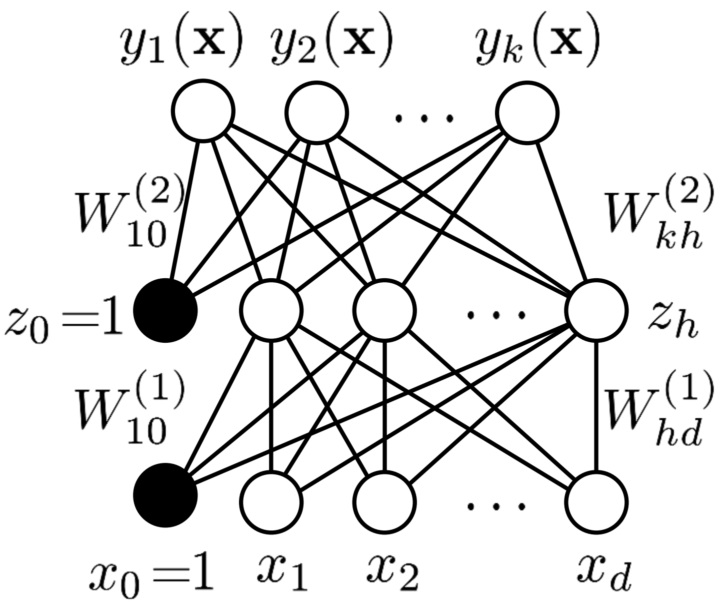
$$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki}\phi(x_i)$$

Logistic outputs

$$y_k(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} W_{ki}\phi(x_i)\right)$$

Recap: Multi-Layer Perceptrons

- Also learning the feature transformation



Output layer

Hidden layer

Mapping (learned!)

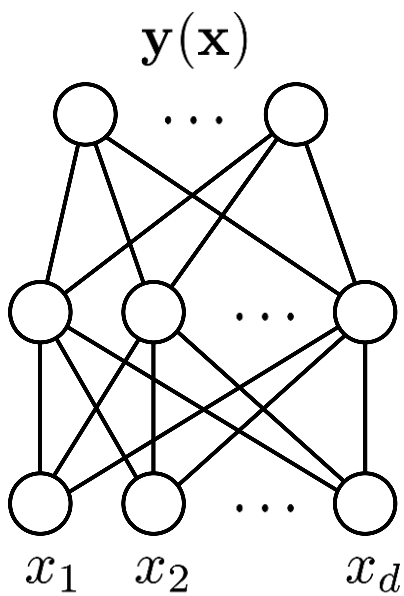
Input layer

- Output

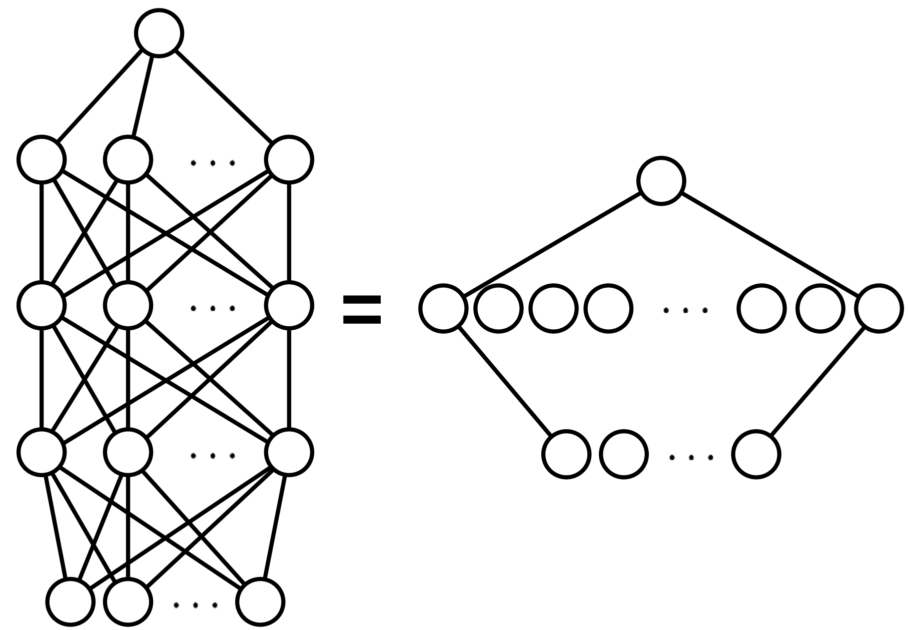
$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

Two Important Remarks

1. Why are hierarchical multi-layered models attractive?



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.

Two Important Remarks

2. Nonlinearities are essential for deep models

$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

- If we leave out the nonlinearity $g^{(1)}(\cdot)$, the two layers collapse into a single linear function

$$\widetilde{W} = W^{(1)} W^{(2)}$$

$\Rightarrow$ The nonlinearities make multi-layer representation more powerful!

Recap: Supervised Learning

- Given
  - Training data set $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^{\mathrm{T}}$
    with target labels $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$.
- Solve an optimization problem
  - Set up an error function

$$E(\mathbf{W}) = \sum_n L(t_n, y(\mathbf{x}_n; \mathbf{W})) + \lambda\Omega(\mathbf{W})$$

    with a loss $L(\cdot)$ and a regularizer $\Omega(\cdot)$.

  - E.g., $L(t, y(\mathbf{x}; \mathbf{W})) = \sum_n \left(y(\mathbf{x}_n; \mathbf{W}) - t_n\right)^2$ L$_2$ loss

$$\Omega(\mathbf{W}) = ||\mathbf{W}||_F^2$$

L$_2$ regularizer (“weight decay”)

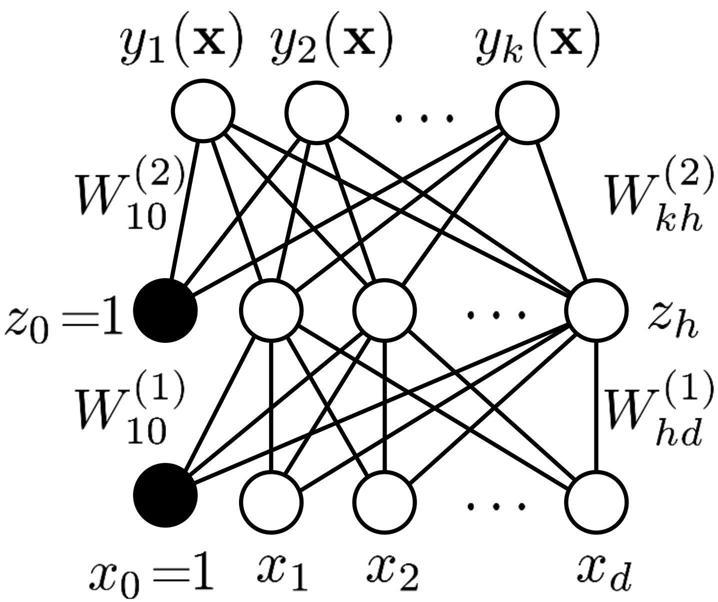
$\Rightarrow$ Update each weight $W_{ij}^{(k)}$ in the direction of the gradient $\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(k)}}$

Recap: Loss Functions

- We can now also apply other loss functions
  - L2 loss $\Rightarrow$ Least-squares regression
    $$L(t, y(\mathbf{x})) = \sum_n \left(y(\mathbf{x}_n) - t_n\right)^2$$
  - L1 loss: $\Rightarrow$ Median regression
    $$L(t, y(\mathbf{x})) = \sum_n |y(\mathbf{x}_n) - t_n|$$
  - Cross-entropy loss $\Rightarrow$ Logistic regression
    $$L(t, y(\mathbf{x})) = -\sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$
  - Hinge loss $\Rightarrow$ SVM classification
    $$L(t, y(\mathbf{x})) = \sum_n [1 - t_n y(\mathbf{x}_n)]_+$$
  - Softmax loss $\Rightarrow$ Multi-class probabilistic classification
    $$L(t, y(\mathbf{x})) = -\sum_n \sum_k \left\{ \mathbb{I}\left(t_n = k\right) \ln \frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))} \right\}$$

Recap: Obtaining the Gradients

- Approach: Incremental Analytical Differentiation



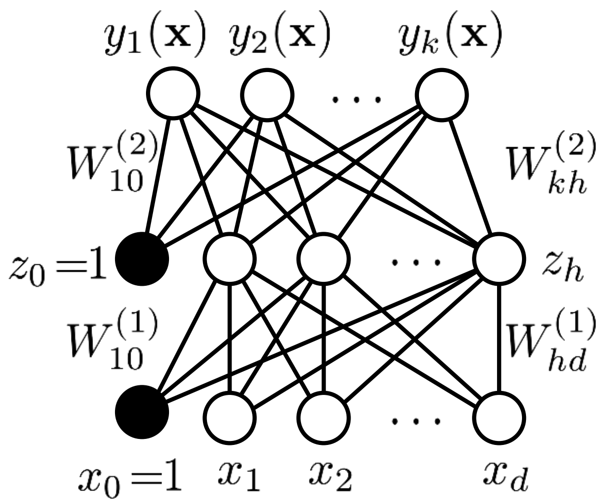
$$
\begin{tikzcd}
\frac{\partial E(\mathbf{W})}{\partial y_j} \arrow[d] \arrow[r] & \frac{\partial E(\mathbf{W})}{\partial W^{(2)}_{ij}} \\
\frac{\partial E(\mathbf{W})}{\partial z_i} \arrow[d] \arrow[r] & \frac{\partial E(\mathbf{W})}{\partial W^{(1)}_{ij}} \\
\frac{\partial E(\mathbf{W})}{\partial x_i}
\end{tikzcd}
$$

  - Idea: Compute the gradients layer by layer.
  - Each layer below builds upon the results of the layer above.

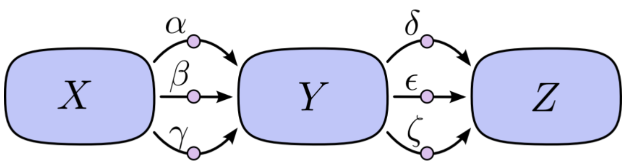
$\Rightarrow$ The gradient is propagated backwards through the layers.

$\Rightarrow$ Backpropagation algorithm

Recap: Backpropagation Algorithm



$\longrightarrow$



- More general formulation (used in deep learning packages)
  - Convert the network into a computational graph.
  - Perform reverse-mode-differentiation this graph
  - Each new layer/module just needs to specify how it affects the
    - forward pass $\quad \mathbf{y} = \text{module.}\mathbf{fprop}(\mathbf{x})$
    - backward pass $\quad \frac{\partial E}{\partial \mathbf{x}} = \text{module.}\mathbf{bprop}(\frac{\partial E}{\partial \mathbf{y}})$

$\Rightarrow$ Very general framework, *any differentiable layer* can be used.

Recap: Supervised Learning

- Two main steps
  1. Computing the gradients for each weight
  2. Adjusting the weights in the direction of the gradient

- Gradient Descent: Basic update equation

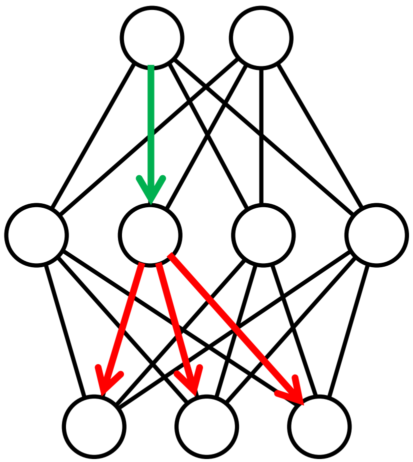
$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- Important considerations
  - On what data do we want to apply this? $\Rightarrow$ Minibatches
  - How should we choose the step size η (the learning rate)?
  - More advanced optimizers (Momentum, RMSProp, Adam, …)

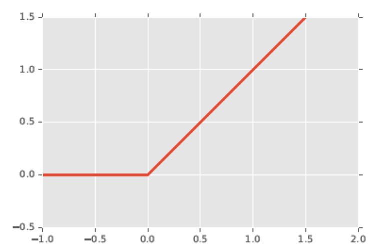
Practical Considerations

- Vanishing gradients problem
  - In multilayer nets, gradients need to be propagated through many layers
  - The magnitudes of the gradients are often very different for the different layers, especially if the initial weights are small.

  ⇒ Gradients can get very small in the early layers of deep nets.



- When designing deep networks, we need to make sure gradients can be propagated throughout the network
  - By restricting the network depth (shallow networks are easier)
  - By very careful implementation (*numerics matter*!)
  - By choosing suitable nonlinearities (e.g., ReLU)
  - By performing proper initialization (Glorot, He)

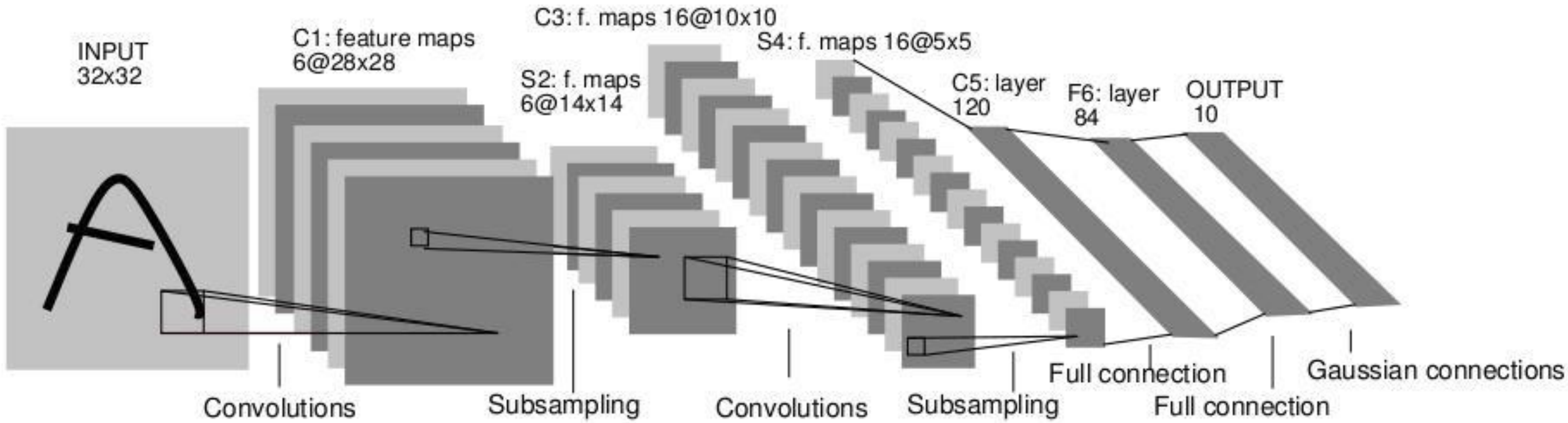


Slide adapted from Geoff Hinton

Topics of This Lecture

- Recap: Convolutional Neural Networks
  - Convolutional Layers
  - Pooling Layers
  - Nonlinearities
- Background: Deep Learning
  - Recap from ML lecture
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet
  - ResNet

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

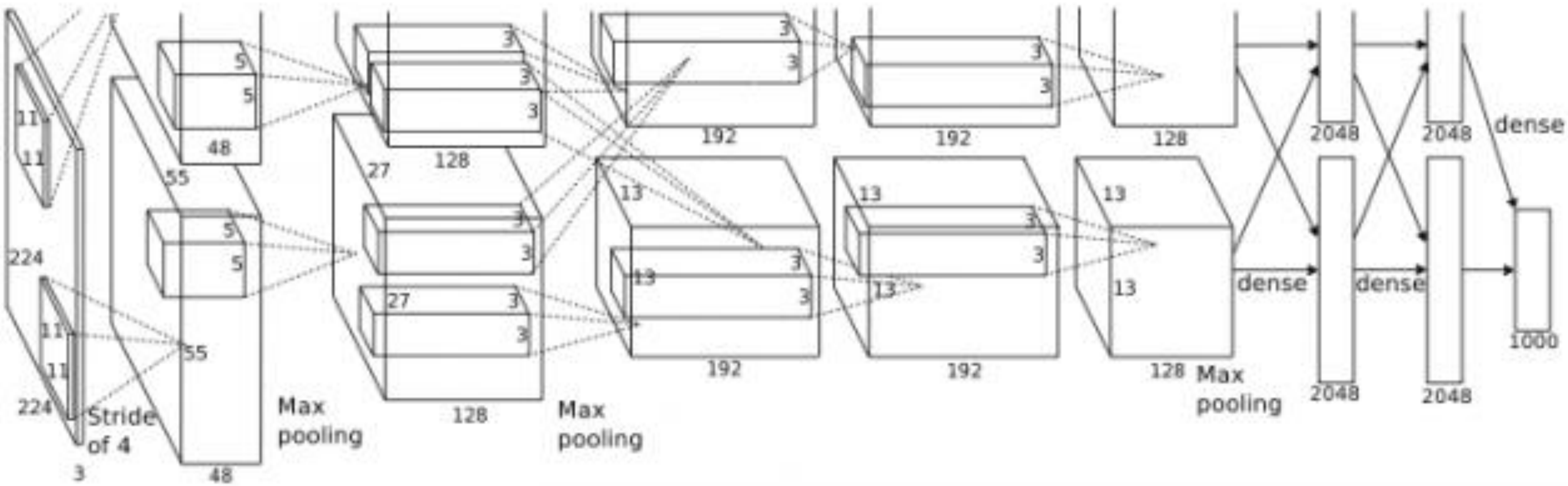
- ImageNet
  - ~14M labeled internet images
  - 20k classes
  - Human labels via Amazon Mechanical Turk

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)

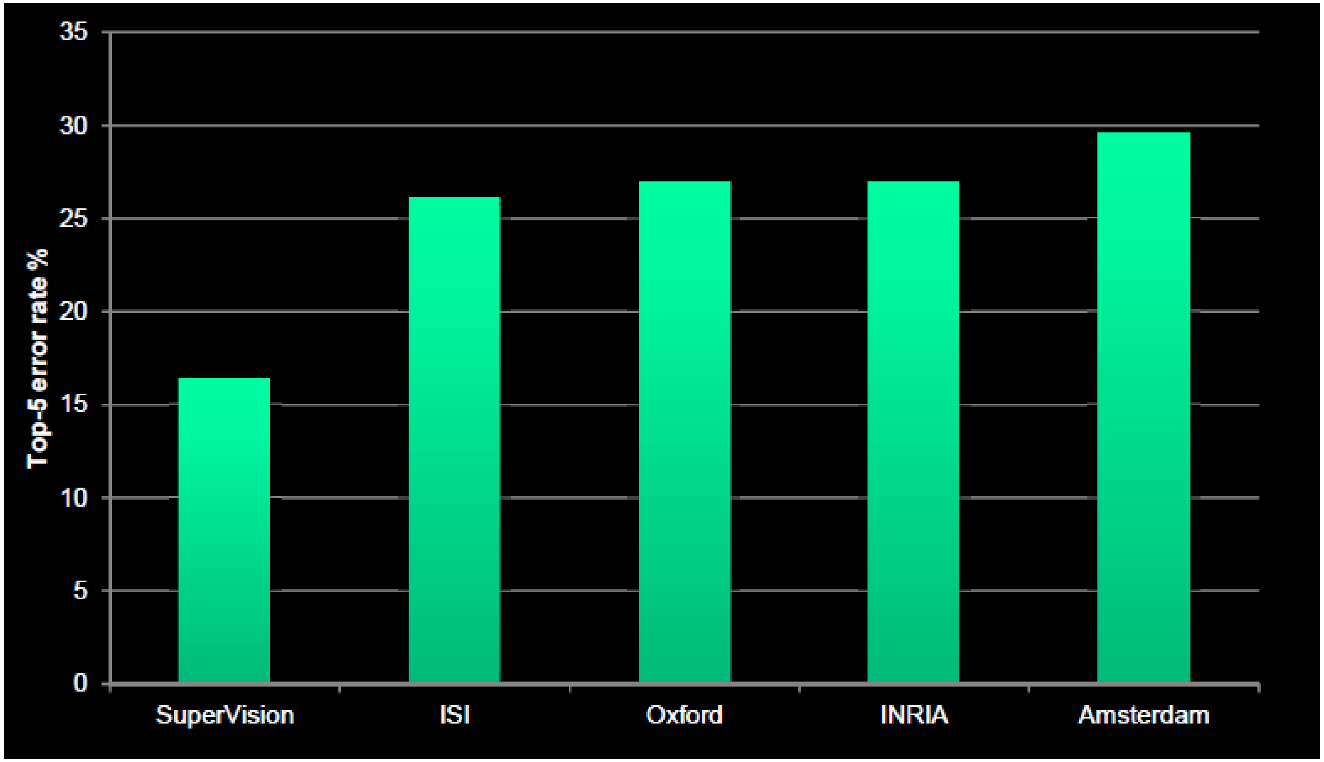


- Similar framework as LeNet, but
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data (10^6 images instead of 10^3)
  - GPU implementation
  - Better regularization and up-to-date tricks for training (Dropout)

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

AlexNet Results

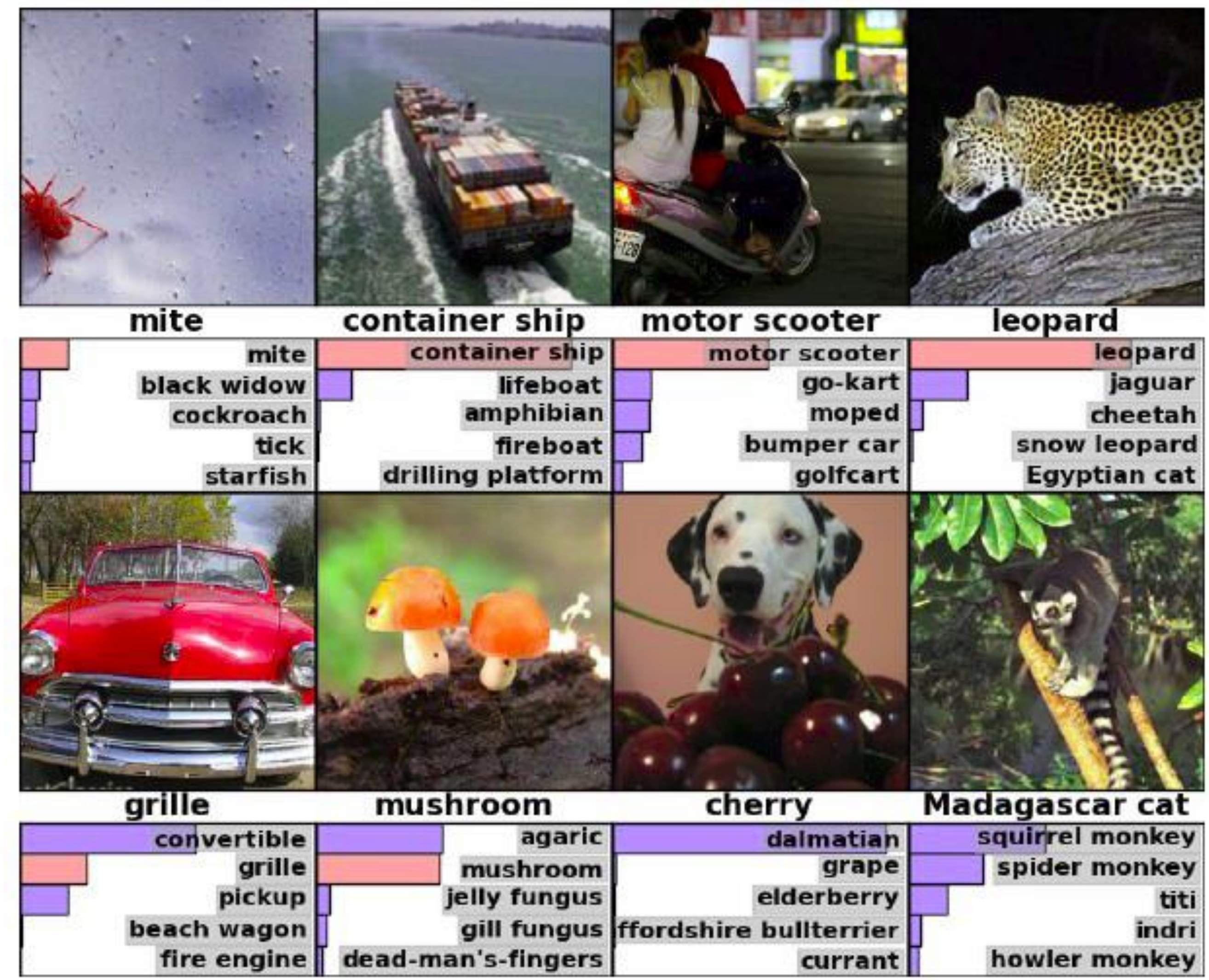


Image source: A. Krizhevsky, I. Sutskever and G.E. Hinton, NIPS 2012

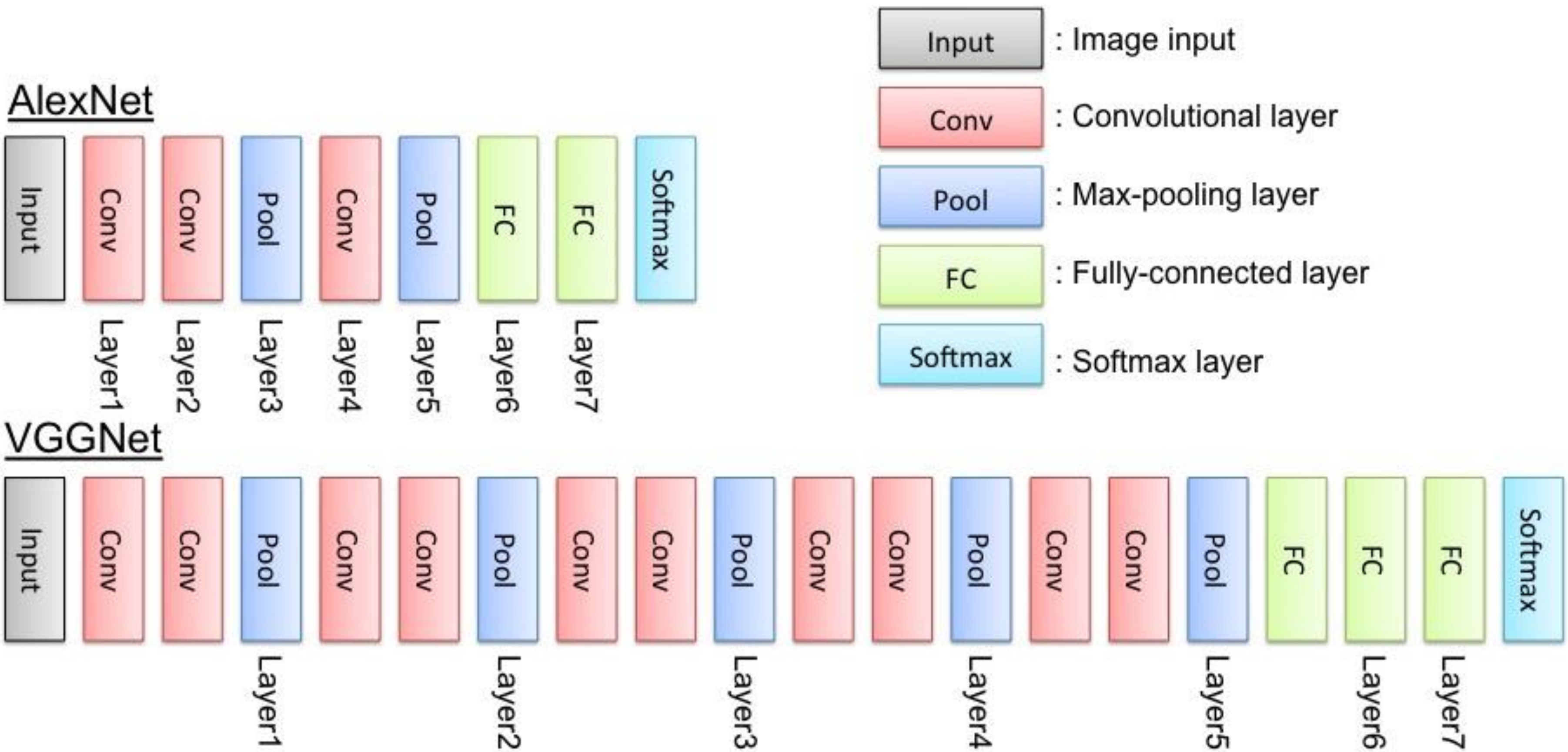
AlexNet Results

Test image

Retrieved images

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

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

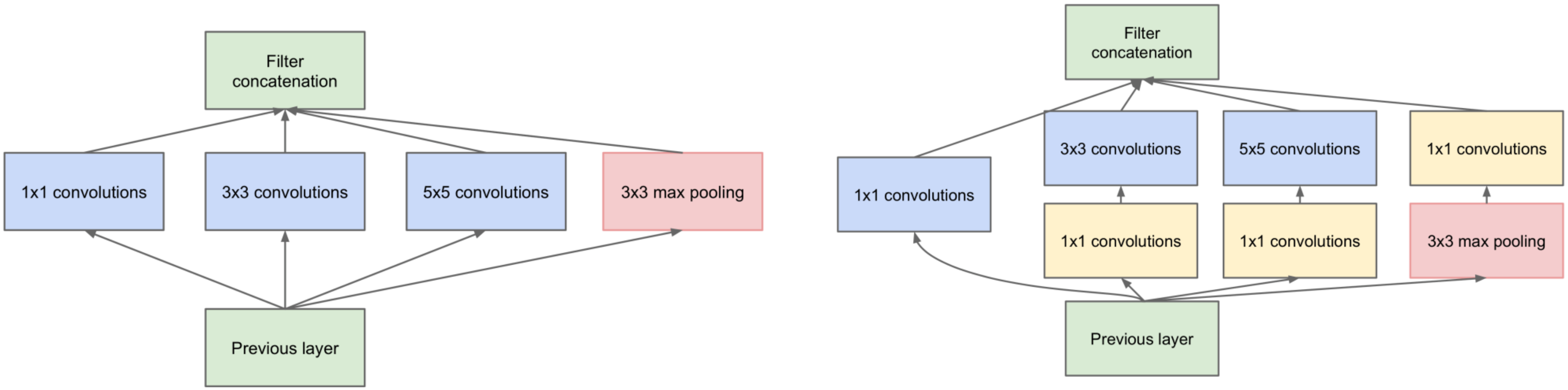
Image source: Simonyan & Zisserman

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 three 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)



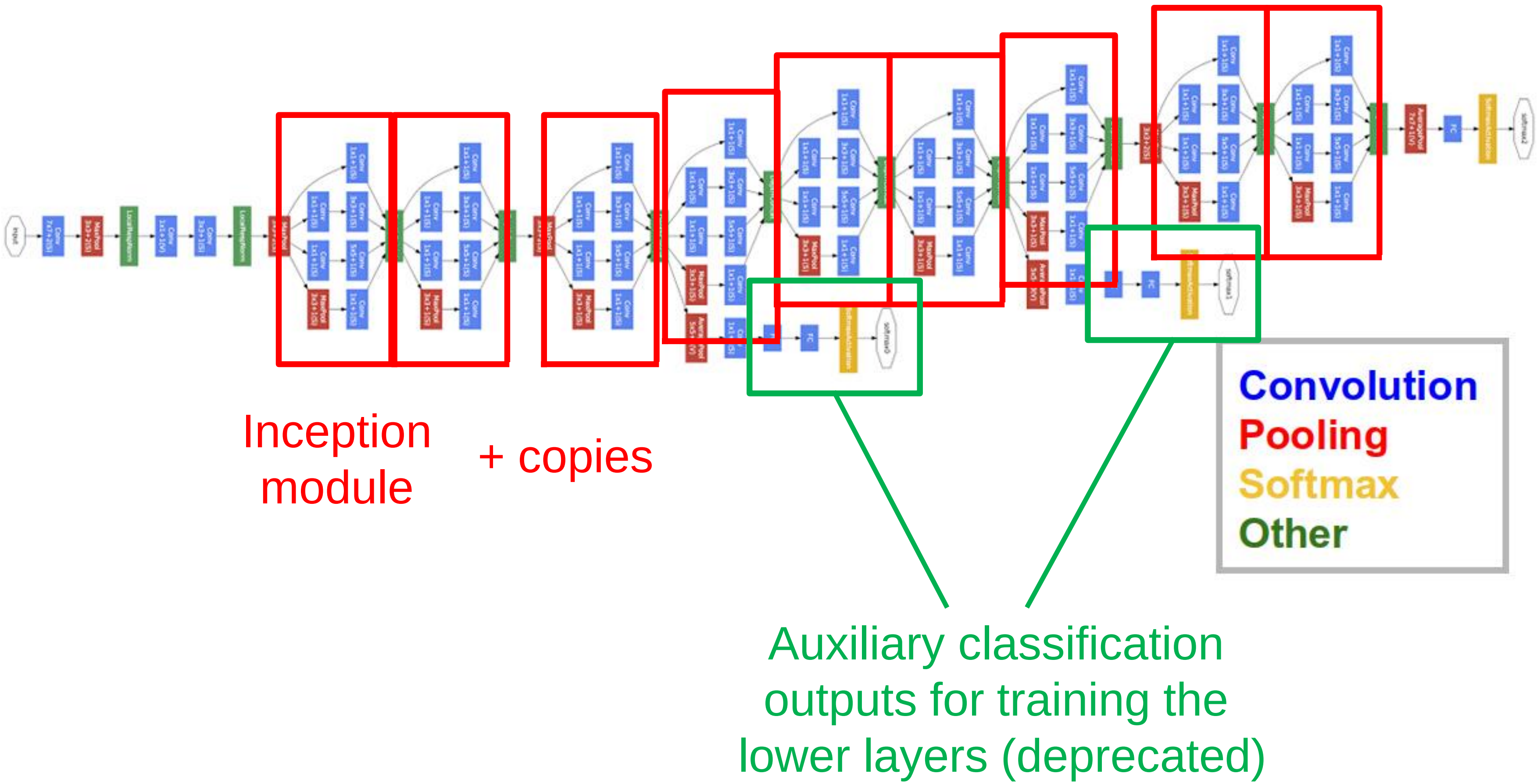
(a) Inception module, naïve version

(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.

GoogLeNet Visualization



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	-

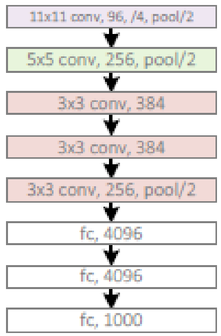
- VGGNet and GoogLeNet perform at similar level
  - Comparison: human performance ~5% [Karpathy]

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

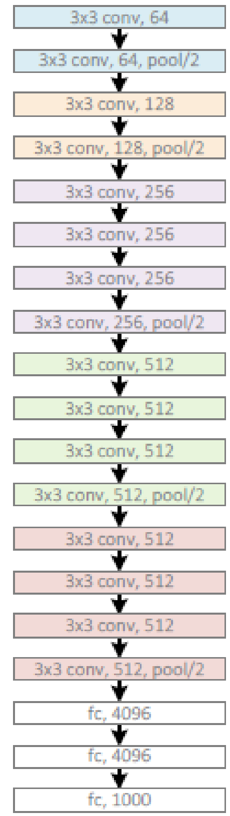
Image source: Simonyan & Zisserman

Newest Development: Residual Networks

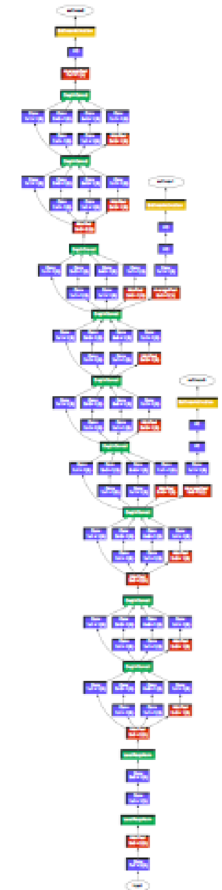
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)



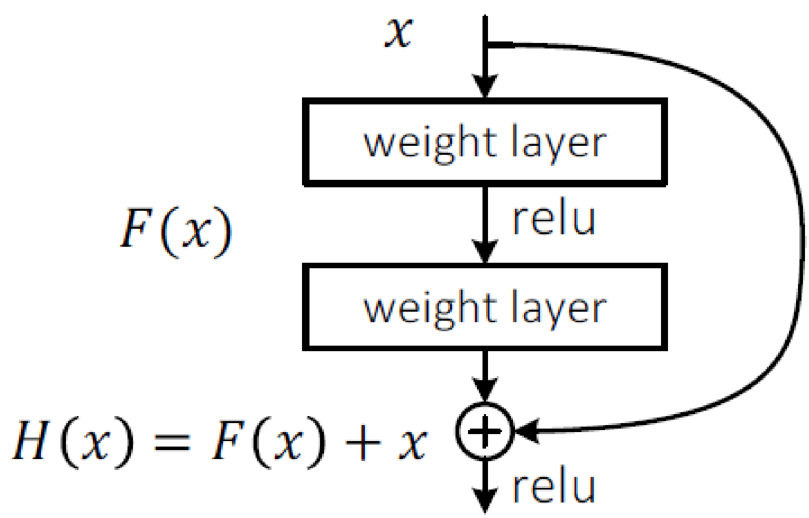
Newest Development: Residual Networks

AlexNet, 8 layers
(ILSVRC 2012)

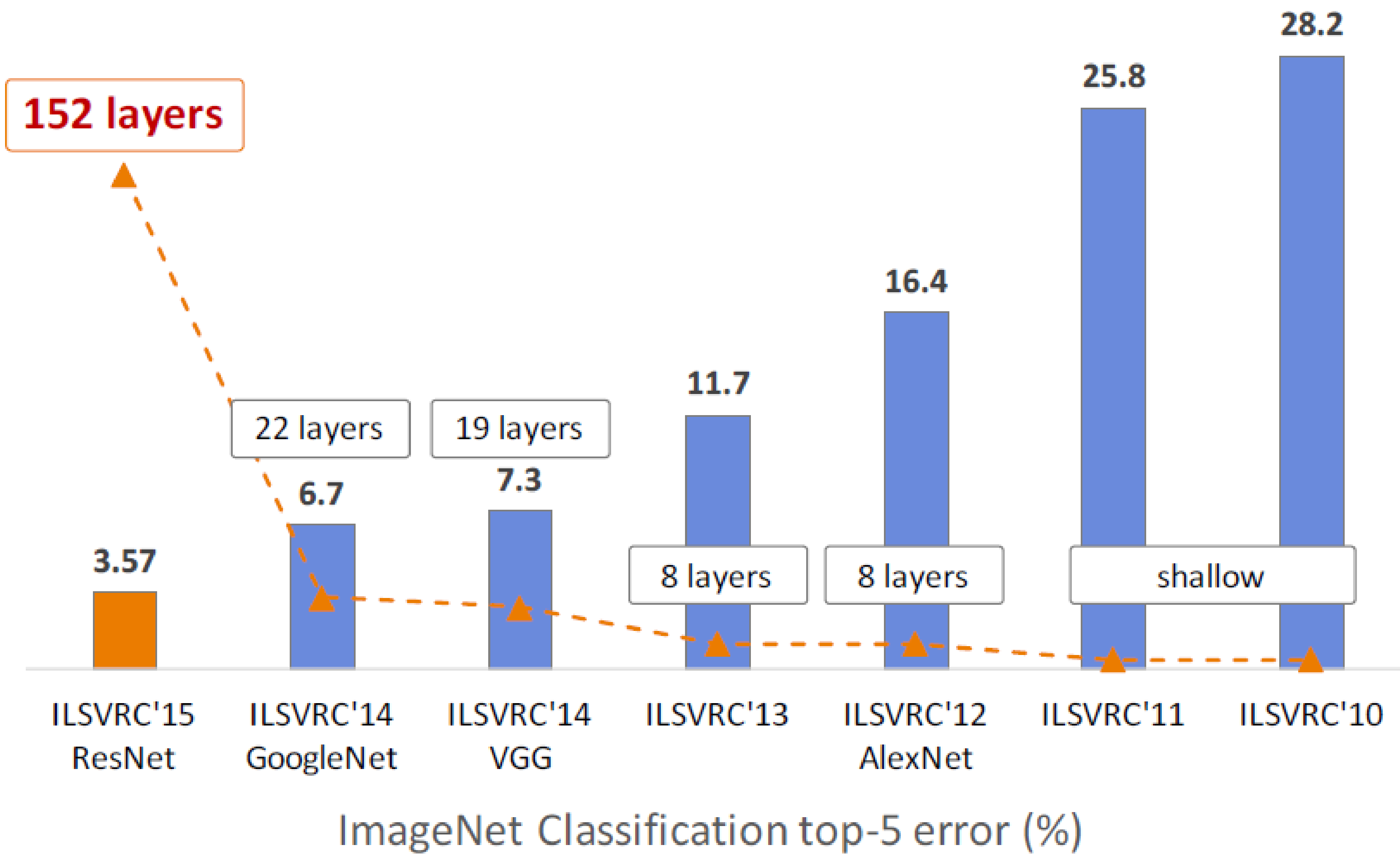
VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

- Core component
  - Skip connections bypassing each layer
  - Better propagation of gradients to the deeper layers

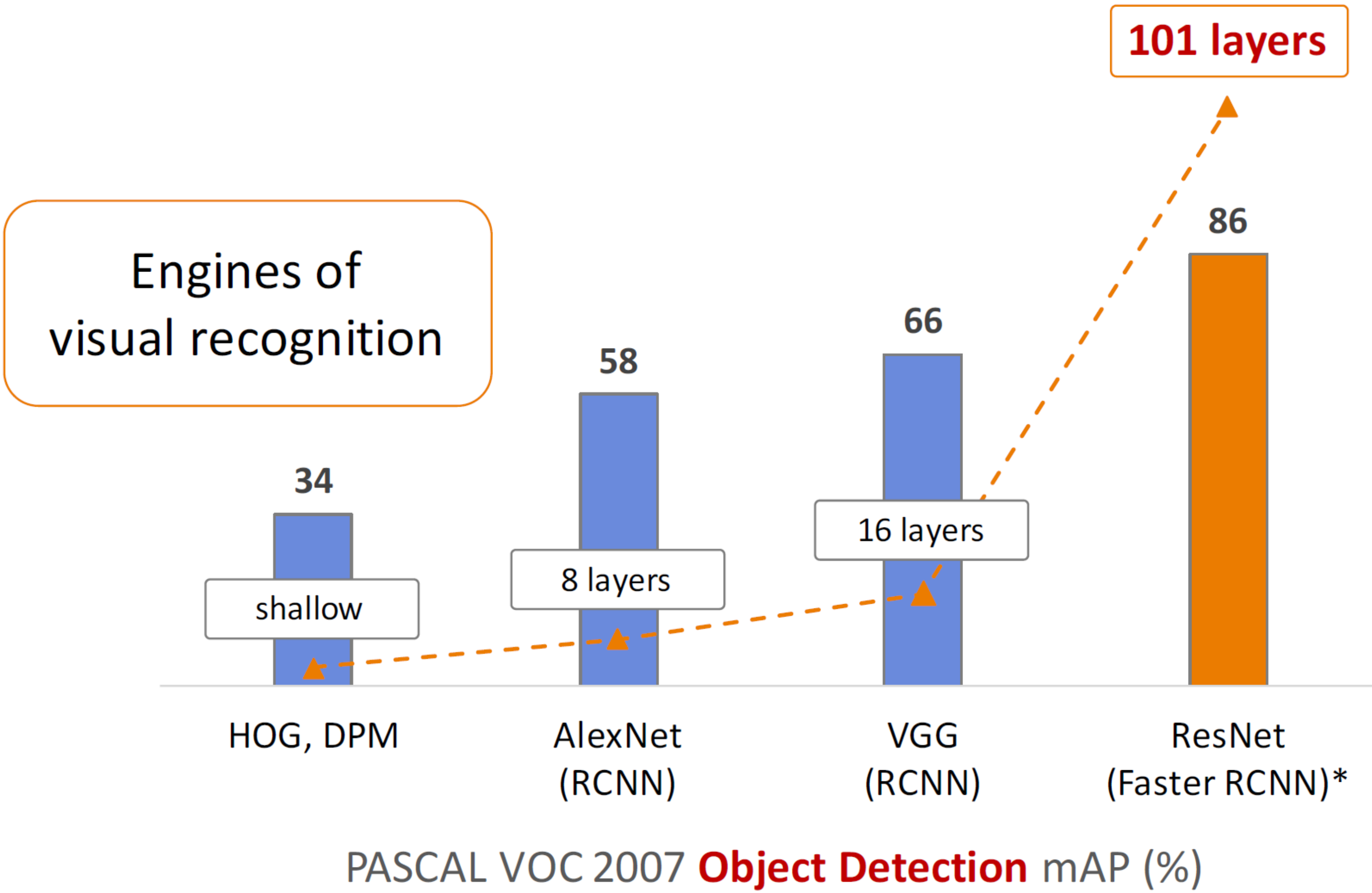


ImageNet Performance



Slide credit: Kaiming He

PASCAL VOC Object Detection Performance



Slide credit: Kaiming He

References and Further Reading

- More information on Deep Learning and CNNs can be found in Chapters 6 and 9 of the Goodfellow & Bengio book

I. Goodfellow, Y. Bengio, A. Courville
Deep Learning
MIT Press, 2016
http://www.deeplearningbook.org/