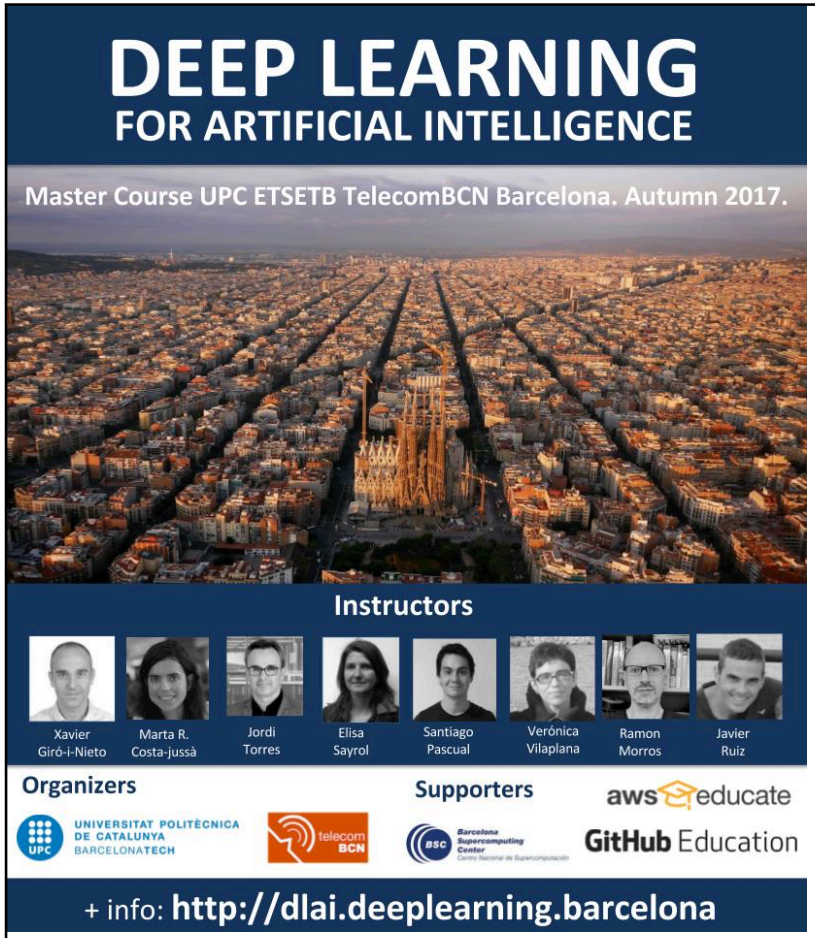


[course site]

#DLUPC

Day 5 Lecture 1

Convolutional Neural Networks

Verónica Vilaplana
veronica.vilaplana@upc.edu

Associate Professor
Universitat Politecnica de Catalunya
Technical University of Catalonia

Index

- Motivation:
  - Local connectivity
  - Parameter sharing
  - Pooling and subsampling
- Layers
  - Convolutional
  - Pooling
  - Fully connected
  - Activation functions
  - Batch normalization
  - Upsampling
- Examples

Motivation

Local connectivity

Parameter sharing

Pooling and subsampling

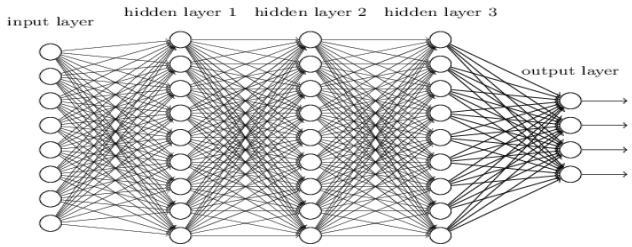
Neural networks for visual data

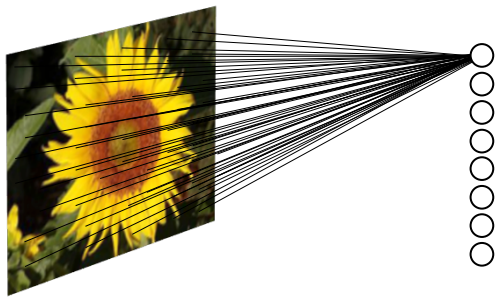
- **Example: Image recognition**
  - Given some input image, identify which object it contains

Caltech101 dataset

image size 150x112

sun flower





neurons connected to 16800 inputs

Neural networks for visual data

- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 150 x 112 pixels = 16800 inputs, or 3 x 16800 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video data)
  - can build in invariance to certain variations we can expect
    - translations, illumination, etc.

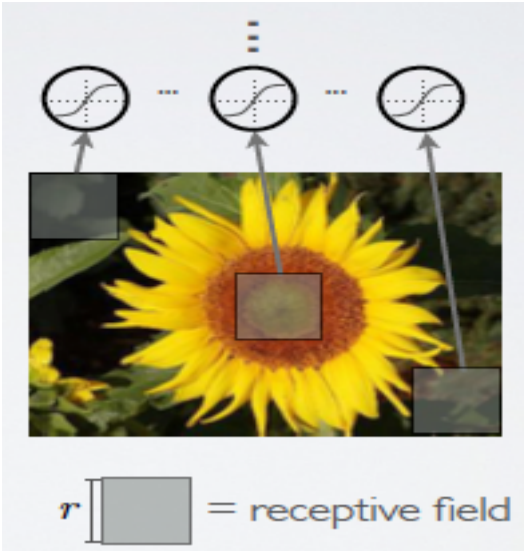
- **Convolutional networks are a specialized kind of neural network for processing data that has a known, grid-like topology. They leverage these ideas:**
  - local connectivity
  - parameter sharing
  - pooling / subsampling hidden units

S. Credit: H. Larochelle

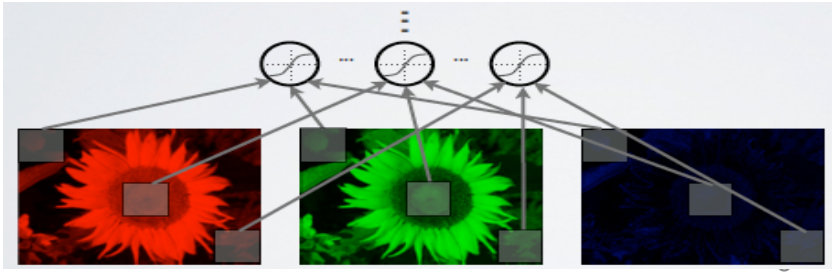
Convolutional neural networks

Local connectivity

- First idea: use a local connectivity of hidden units
  - each hidden unit is connected only to a subregion (patch) of the input image: **receptive field**
  - it is connected to all channels
    - 1 if greyscale image
    - 3 (R, G, B) for color image
    - …



- Solves the following problems:
  - fully connected hidden layer would have an unmanageable number of parameters
  - computing the linear activations of the hidden units would be very expensive

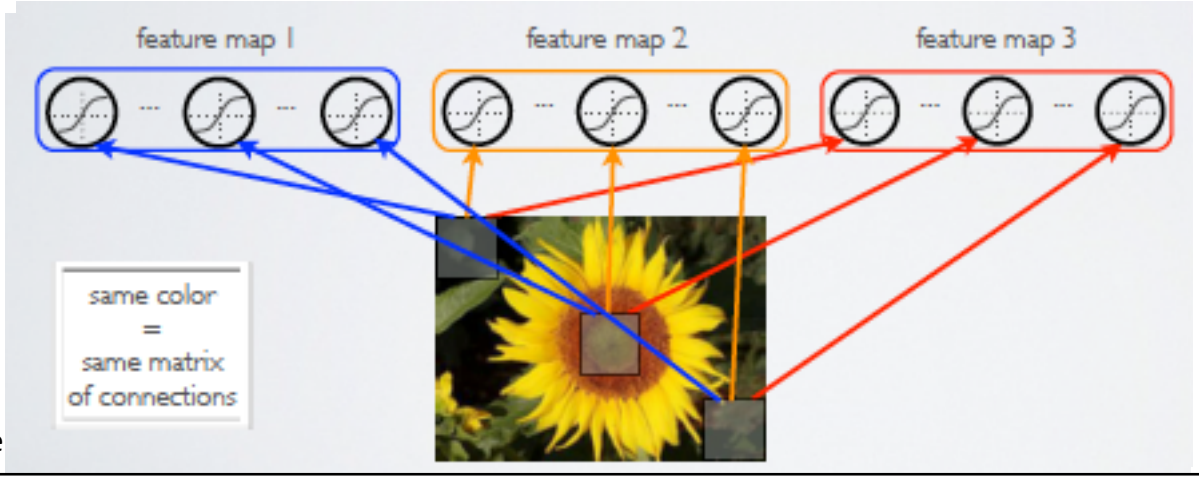


S. Credit: H. Larochelle

Convolutional neural networks

Parameter sharing

- Second idea: share matrix of parameters across certain units
  - units organized into the same ‘‘feature map’’ share parameters
  - hidden units within a feature map cover different positions in the image
- Solves the following problems:
  - reduces even more the number of parameters
  - will extract the same features at every position (features are ‘‘equivariant’’)



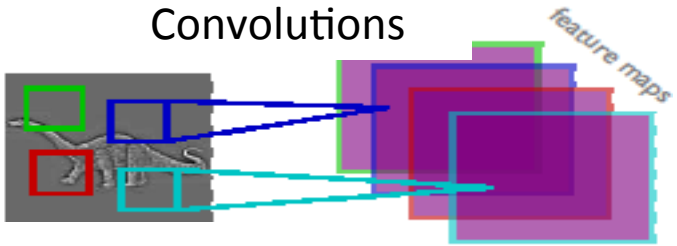
W_{ij} is the matrix connecting the ith input channel with the jth feature map

S. Credit: H. Larochelle

Convolutional neural networks

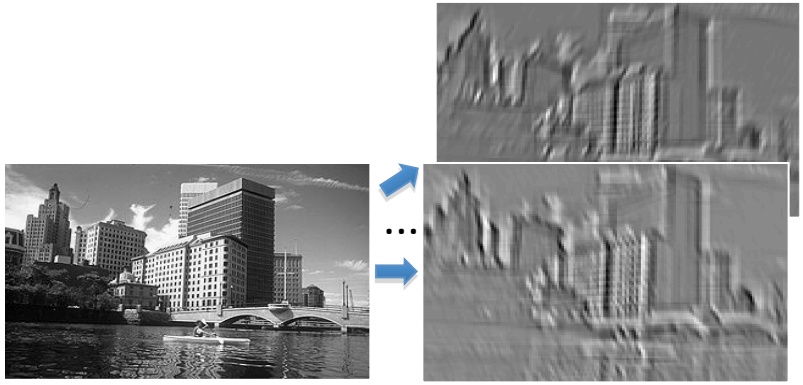
Parameter sharing

- Each feature map forms a 2D grid of features
  - can be computed with **a discrete convolution** of a kernel matrix k_{ij}
    which is the hidden weights matrix W_{ij} with its rows and columns flipped



$$y_j = f(\sum_i k_{ij} * x_i)$$

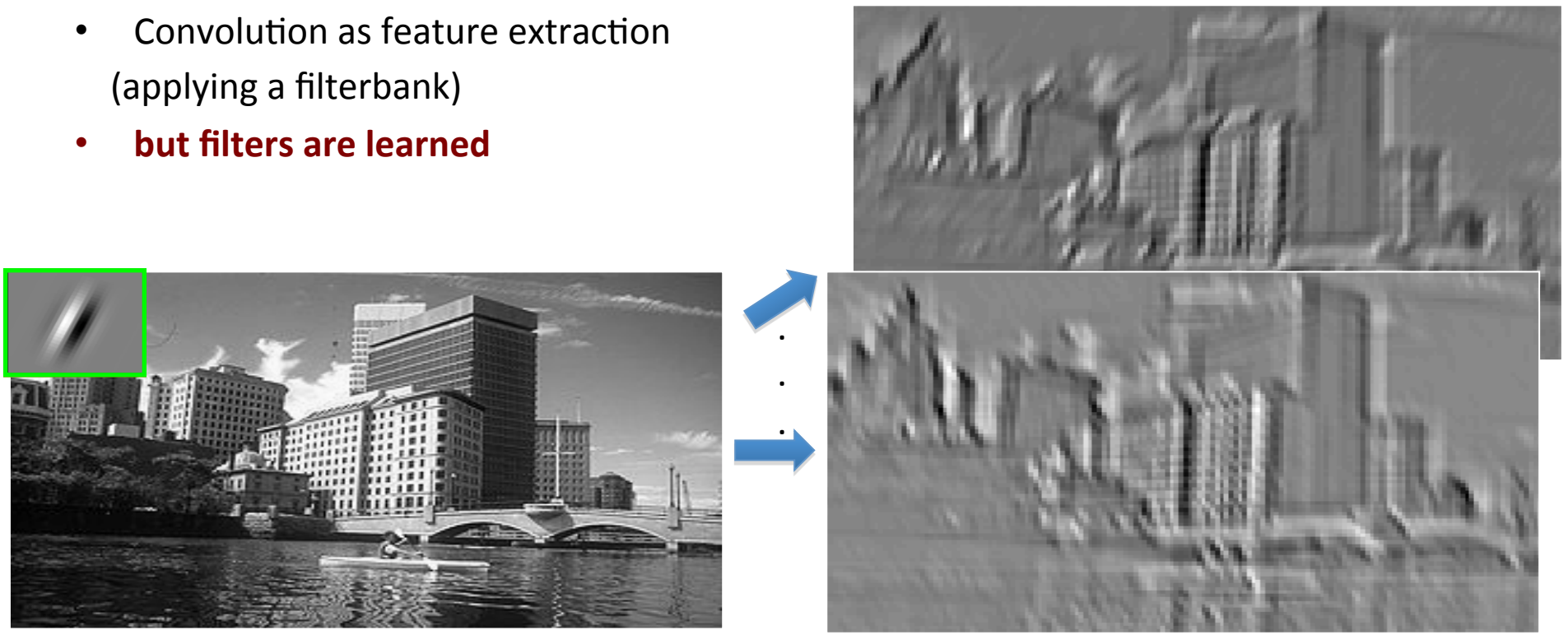
x_i ith channel of input, y_j hidden layer



S. Credit: H. Larochelle

Convolutional neural networks

- Convolution as feature extraction (applying a filterbank)
- **but filters are learned**



Input

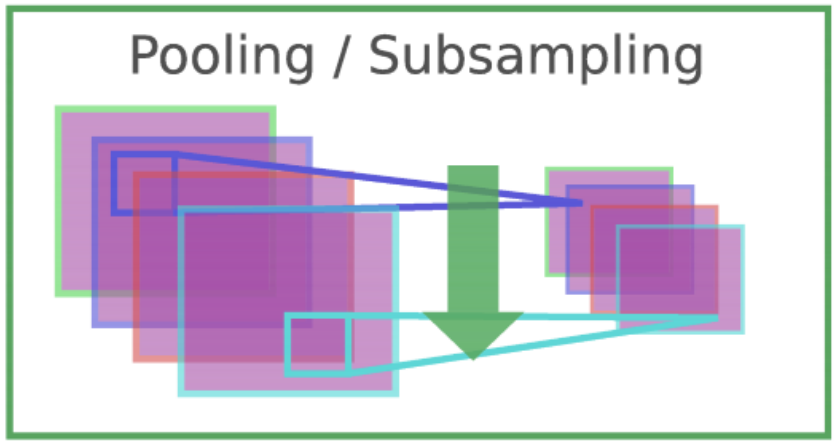
Feature Map

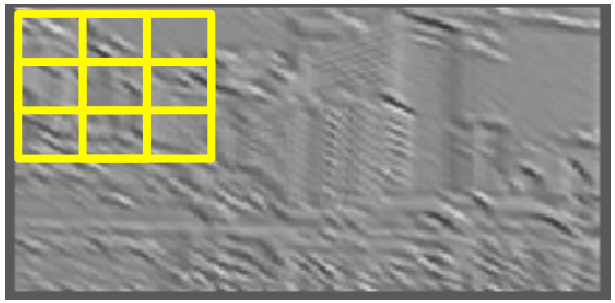
S. Credit: H. Larochelle

Convolutional neural networks

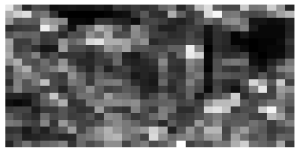
Pooling and subsampling

- Third idea: pool hidden units in same neighborhood
  - pooling is performed in non-overlapping neighborhoods (subsampling)
  - an alternative to ‘‘max’’ pooling is ‘‘average’’ pooling
  - pooling reduces dimensionality and provides invariance to small local changes





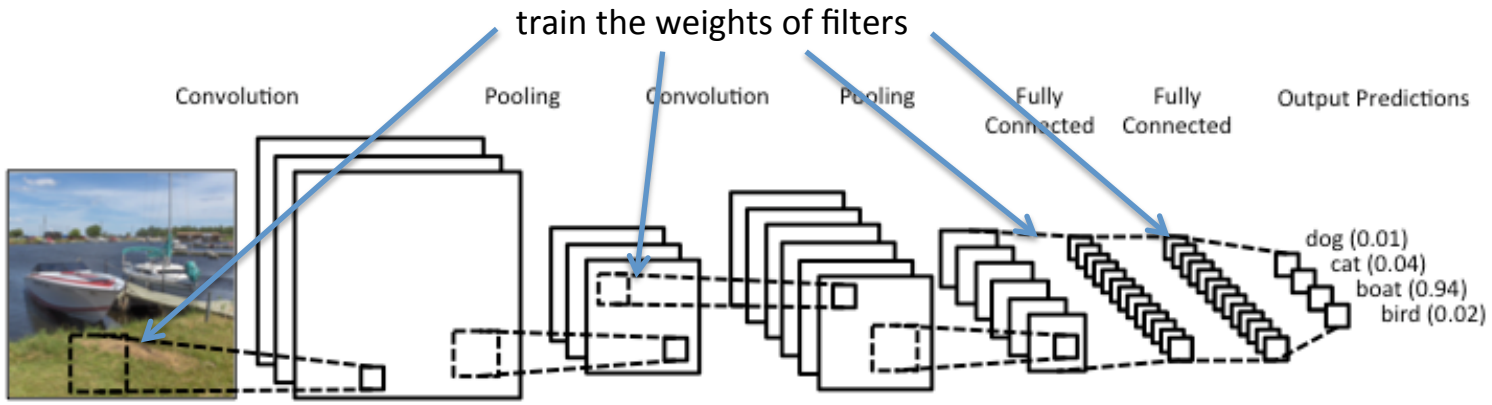
Max



S. Credit: H. Larochelle

Convolutional Neural Networks

- **Convolutional neural networks alternate between convolutional layers (followed by a nonlinearity) and pooling layers (basic architecture)**

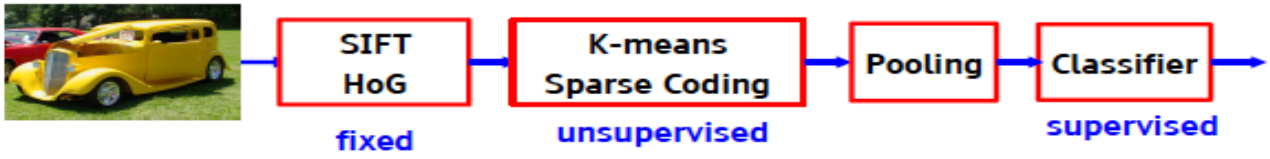


- For recognition: output layer is a regular, fully connected layer with softmax non-linearity
  - output provides an estimate of the conditional probability of each class
- The network is trained by stochastic gradient descent (& variants)
  - backpropagation is used similarly as in a fully connected network

S. Credit: H. Larochelle

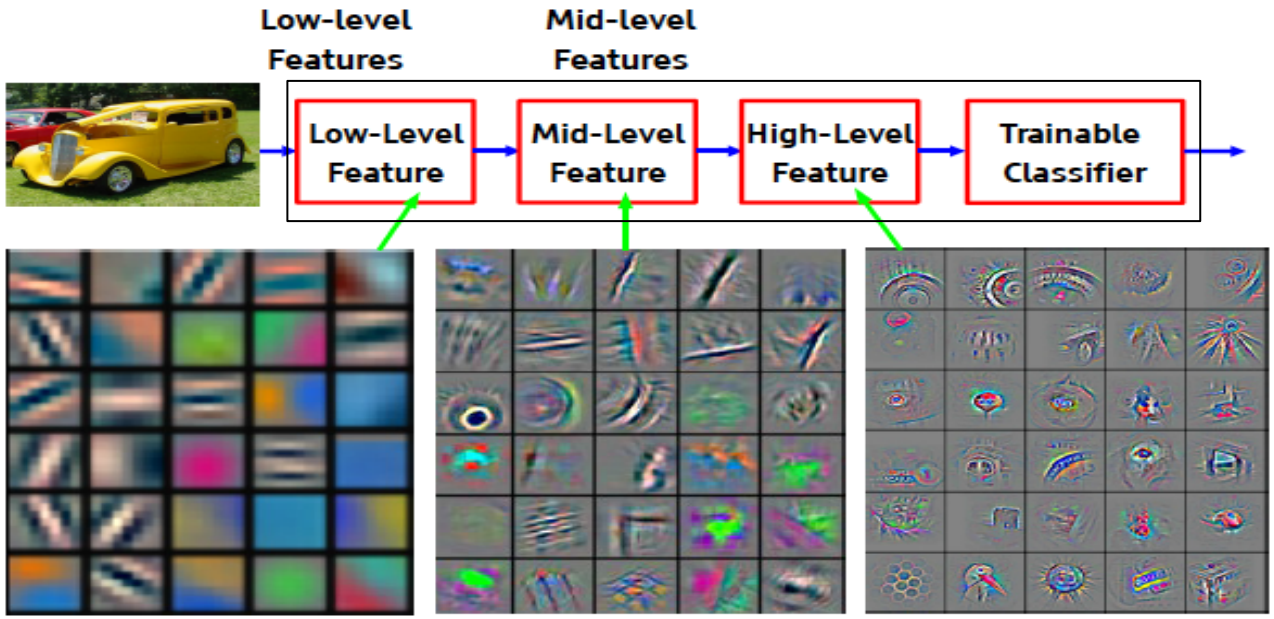
Convolutional Neural Networks

'Clasic' architecture for object recognition (BoW):



CNN= learning hierarchical representations with increasing levels of abstraction

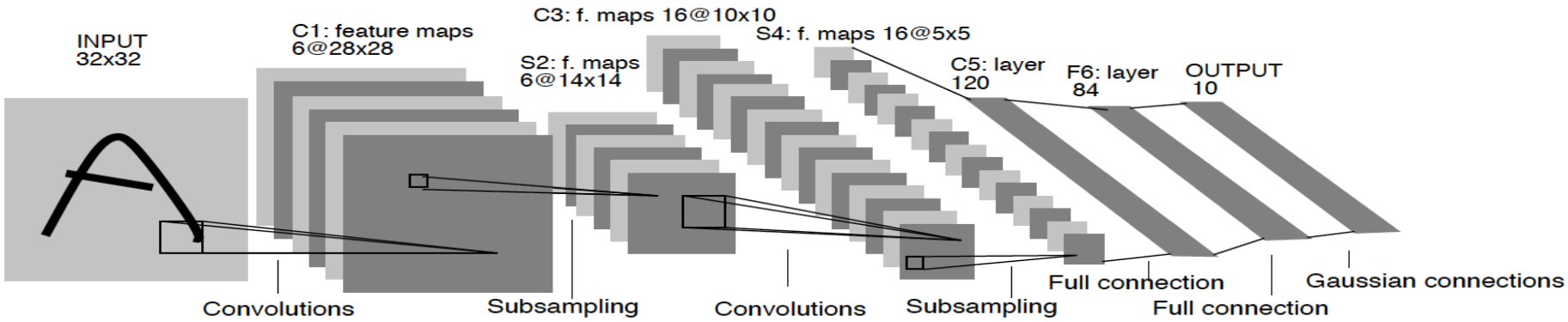
End-to-end training: joint optimization of features and classifier



Feature visualization of convolutional net trained on ImageNet (Zeiler, Fergus, 2013)

Example: LeNet-5

- LeCun et al., 1998



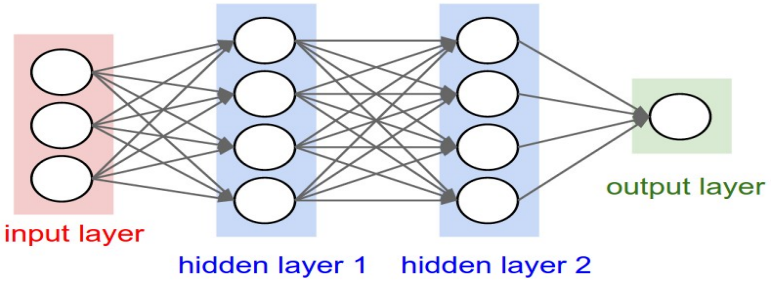
MNIST digit classification problem
handwritten digits
60,000 training examples
10,000 test samples
10 classes
28x28 grayscale images

Conv filters were 5x5, applied at stride 1
Sigmoid or tanh nonlinearity
Subsampling (average pooling) layers were 2x2 applied at stride 2
Fully connected layers at the end
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

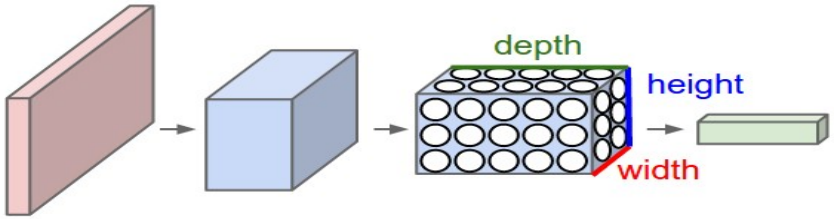
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

Layers

Convolutional Neural Networks



A regular 3-layer Neural Network



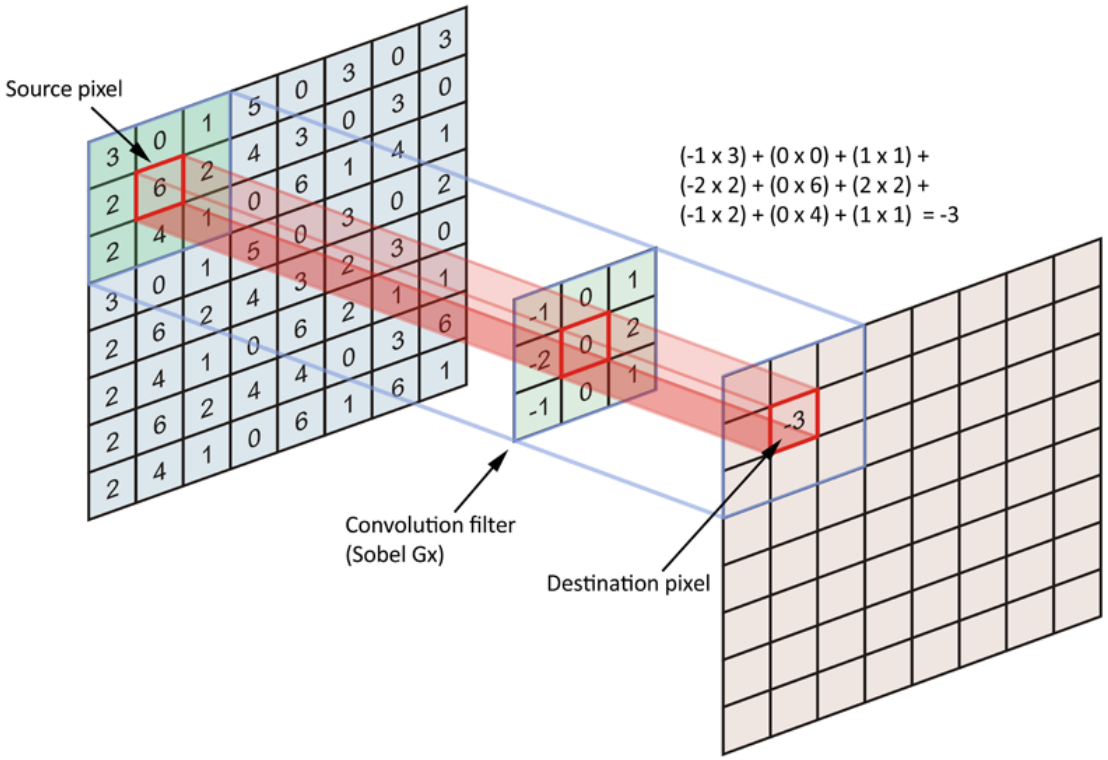
A ConvNet with 3 layers

- In ConvNets inputs are 'images' (architecture is constrained)
- A ConvNet arranges neurons in three dimensions (width, height, depth)
- Every layer transforms 3D input volume to a 3D output volume of neuron activations

S. Credit: Stanford cs231_2017

Convolutional layer

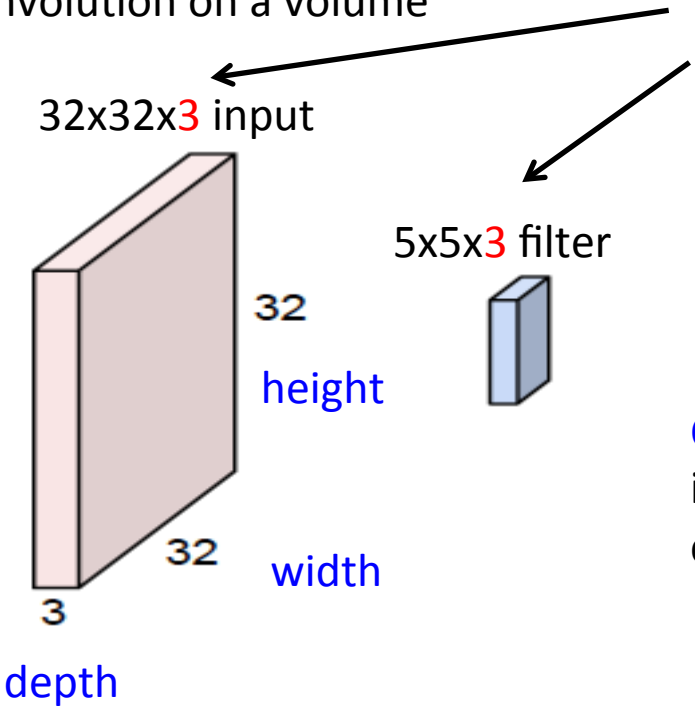
- Convolution on a 2D grid



[Image source]

Convolutional layer

- Convolution on a volume

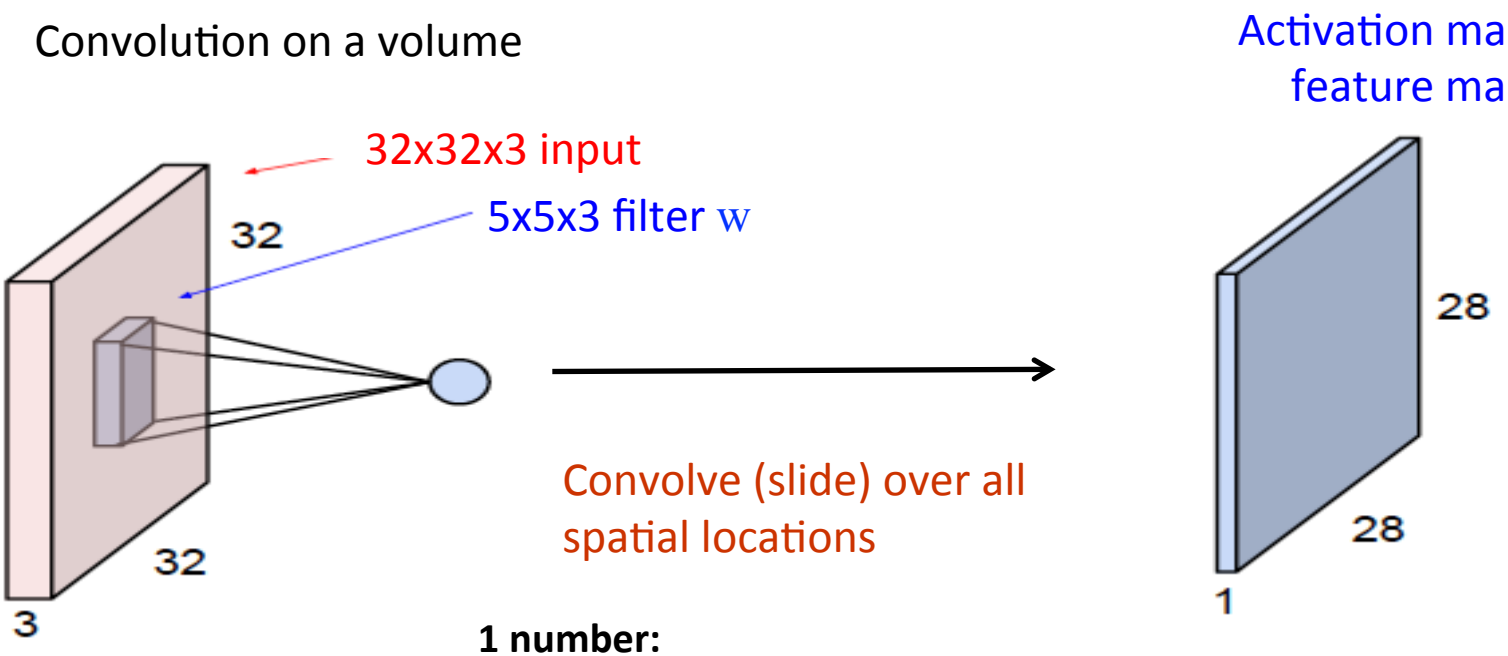


Convolve the filter with the input
i.e. slide over the input spatially,
computing dot products

S. Credit: Stanford cs231_2017

Convolutional layer

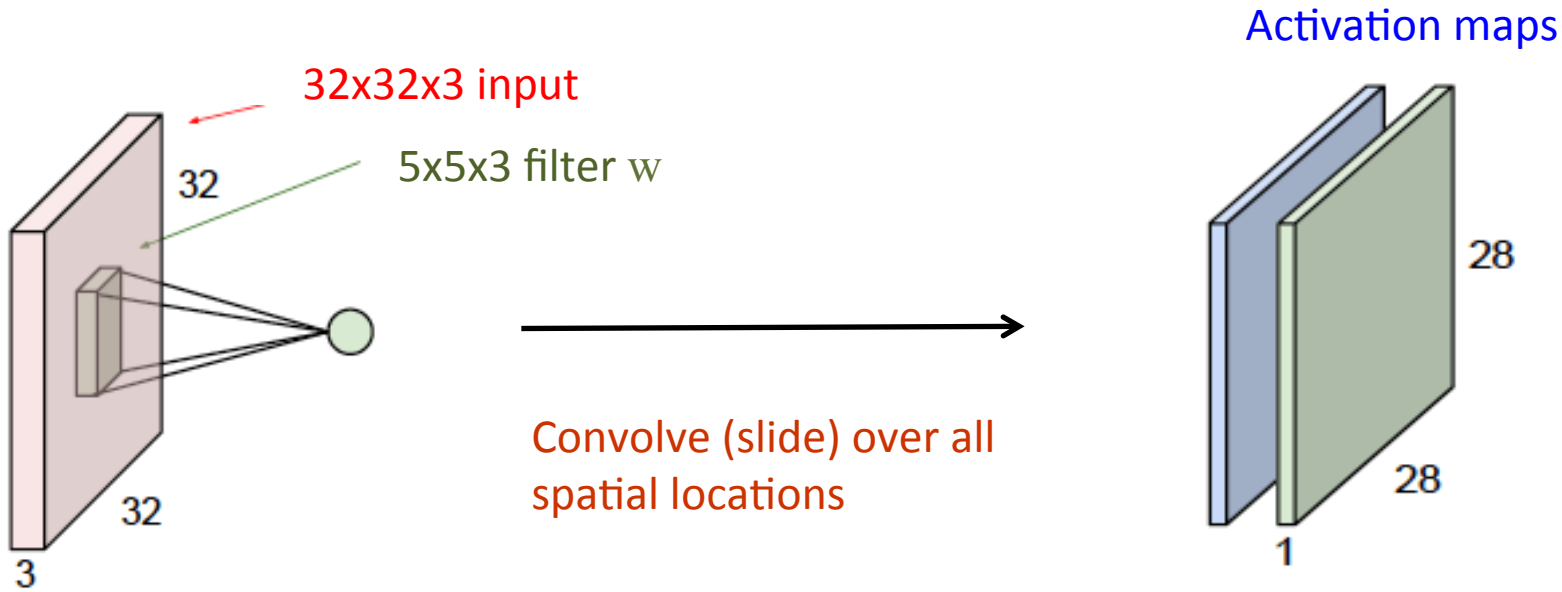
- Convolution on a volume



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the input: 5x5x3=75-dim dot product + bias $w^t x+b$

S. Credit: Stanford cs231_2017

Convolutional layer

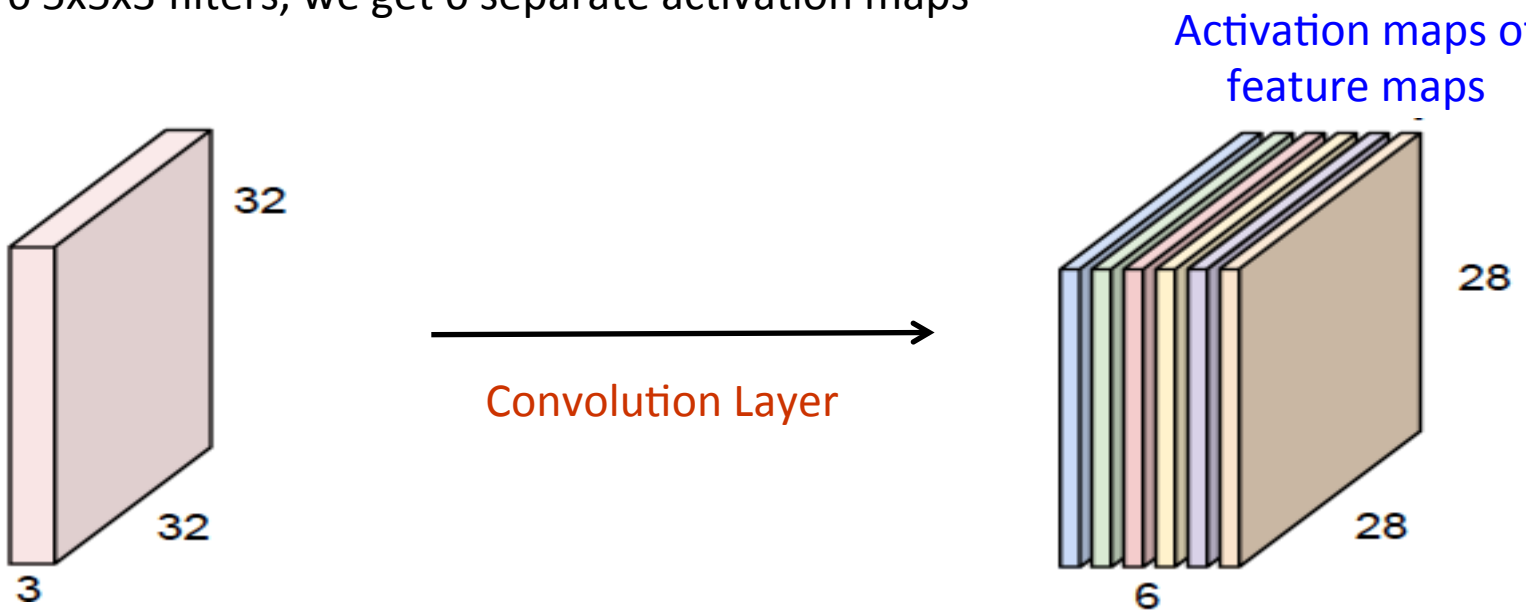


Consider a second, green filter

S. Credit: Stanford cs231_2017

Convolutional layer

If we have 6 5x5x3 filters, we get 6 separate activation maps



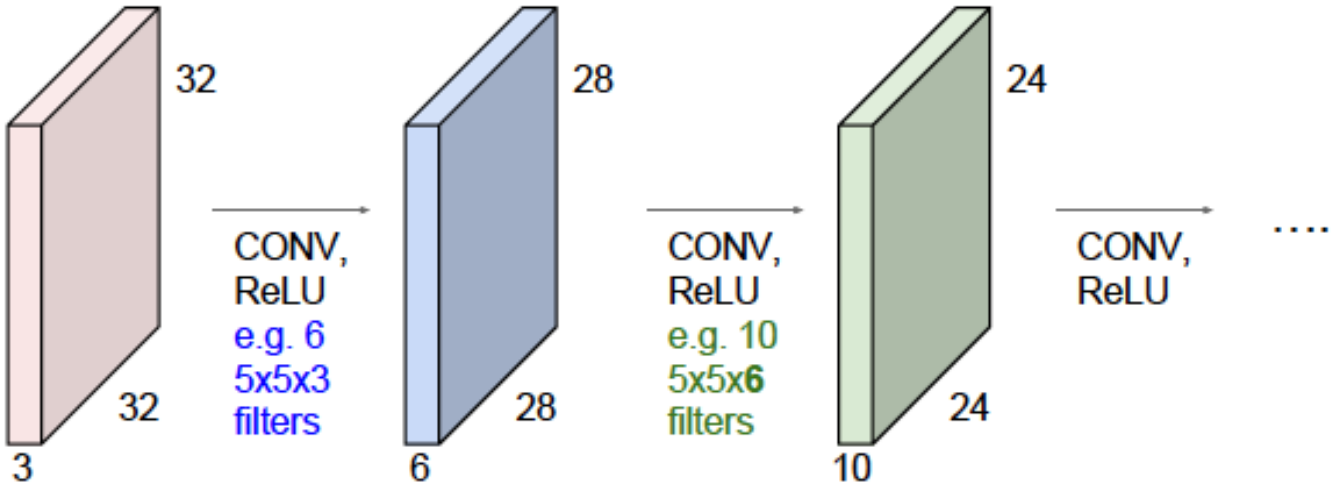
We stack the maps up to get a 'new volume' of size 28x28x6

S. Credit: Stanford cs231_2017

So applying a filterbank to an input (3D matrix) yields a cube-like output, a 3D matrix in which each slice is an output of convolution with one filter.

Convolutional layer

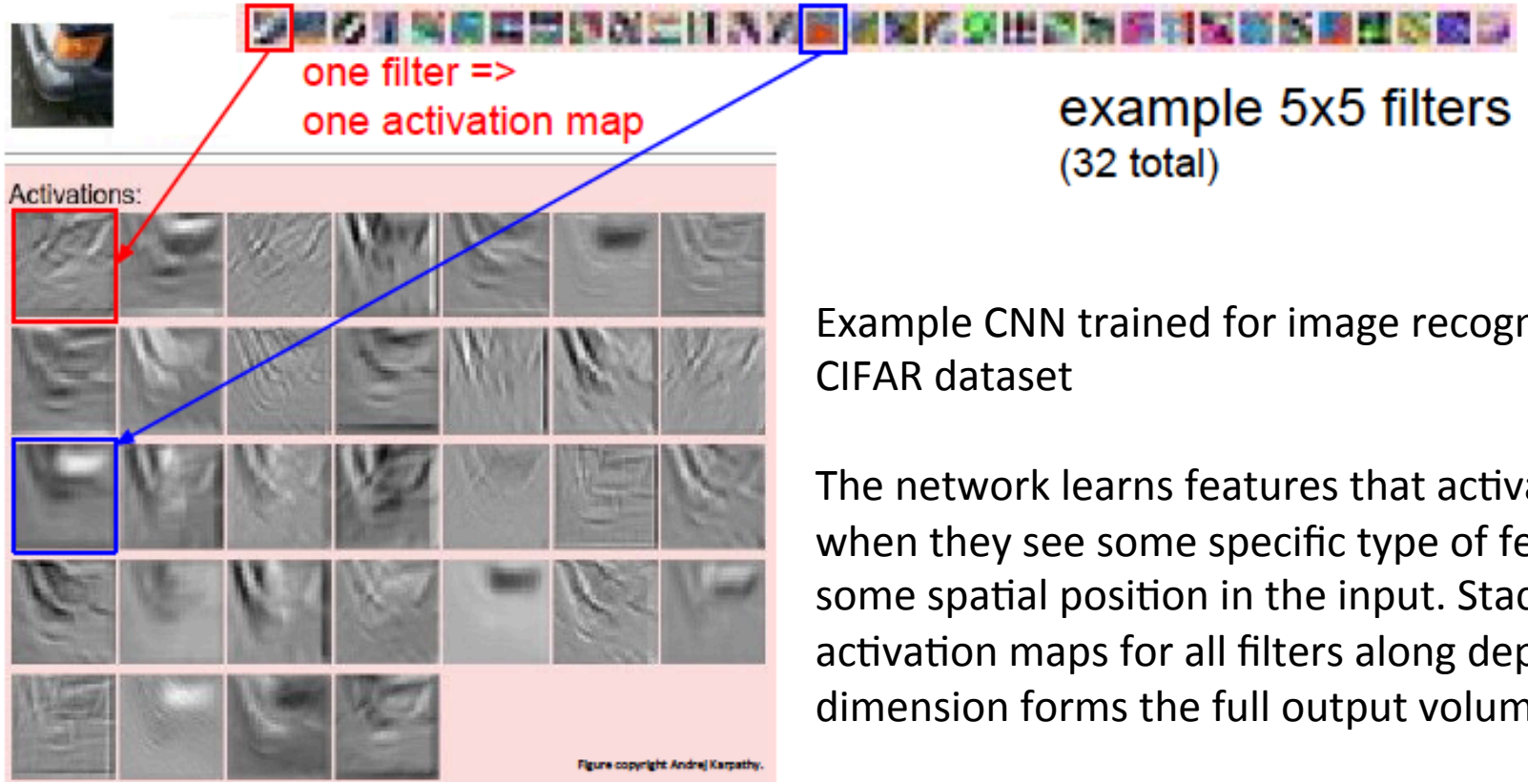
ConvNet is a sequence of Convolutional Layers, interspersed with activation functions and pooling layers (and a small number of fully connected layers)



We add more layers of filters. We apply filters (convolutions) to the output volume of the previous layer. The result of each convolution is a slice in the new volume.

S. Credit: Stanford cs231_2017

Example: filters and activation maps



Example CNN trained for image recognition on CIFAR dataset

The network learns features that activate when they see some specific type of feature at some spatial position in the input. Stacking activation maps for all filters along depth dimension forms the full output volume

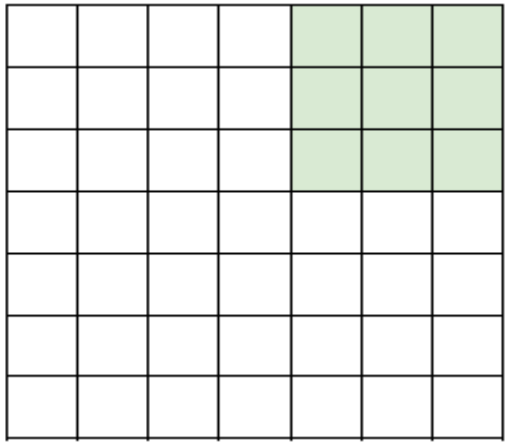
http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Convolution

Hyperparameters: filter spatial extent F, stride S, padding P

Stride is the number of pixels by which we slide the filter matrix over the input matrix.
Larger stride will produce smaller feature maps.

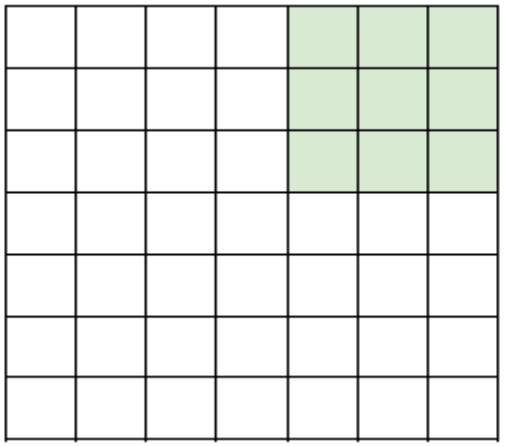
7



7

7x7 input (spatially)
assume 3x3 filter: 5x5 output
(stride 1)

7



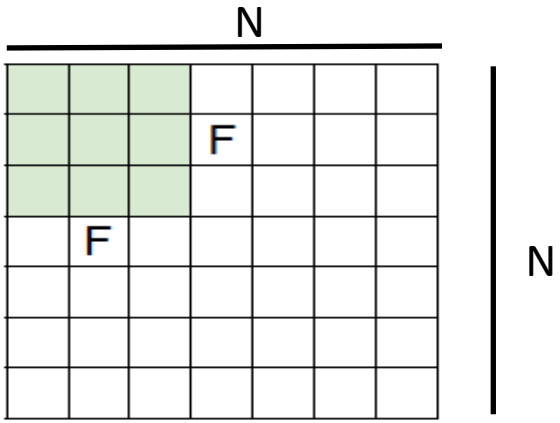
7

7x7 input (spatially)
assume 3x3 filter
applied with stride 2: 3x3 output

S. Credit: Stanford cs231_2017

Convolution

Hyperparameters: filter spatial extent F, stride S, padding P



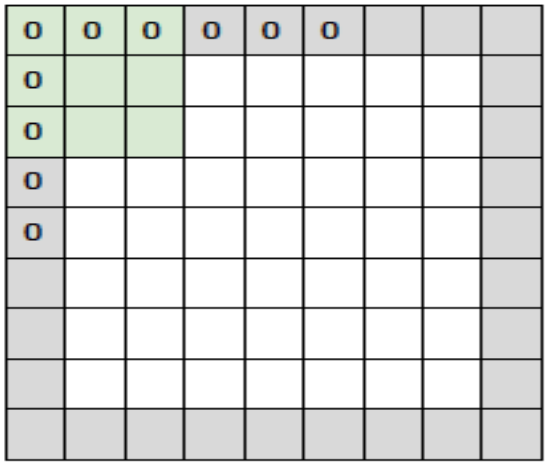
Output size: **(N-F)/ stride + 1**

e.g. N=7, F=3

stride 1: (7-3)/1+1 = 5

stride 2: (7-3)/2+1 = 3

stride 3: (7-3)/3+1 = 2.33 not applied



common: zero-padding in the border

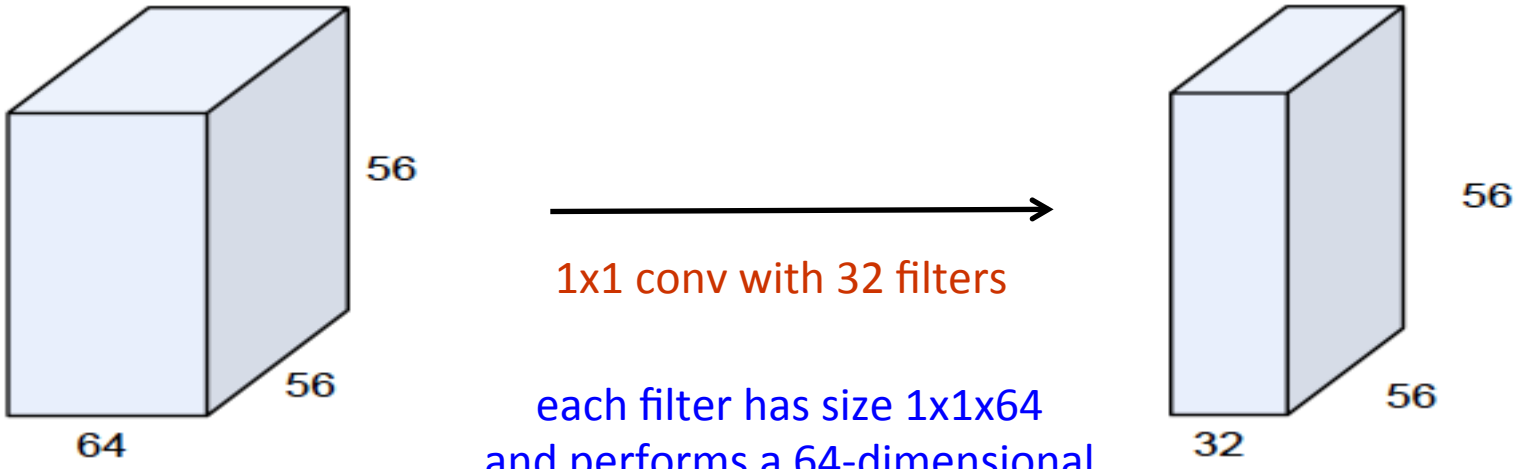
E.g. N=7, F=3 , stride 1, pad with 1 pixel border: output size 7x7

In general, common to see CONV layers with stride 1, filters FxF, zero-padding with P= (F-1)/2: **preserve size spatially**

S. Credit: Stanford cs231_2017

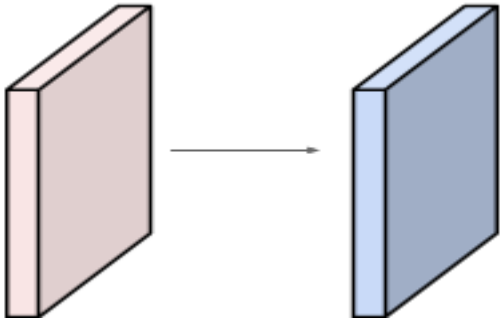
1x1 convolutions

1x1 convolution layers: used to reduce dimensionality (number of feature maps)



S. Credit: Stanford cs231_2017

Example: size, parameters



Input volume: 32x32x3
10 5x5 filters with stride 1, pad 2

Output volume size:
(N + 2P – F) / S + 1
(32+2*2-5)/1+1= 32 spatially so 32x32x10

Number of parameters in this layer:
each filter has 5x5x3 + 1 =76 params (+1 for bias)
-> 76x10 = 760 parameters

S. Credit: Stanford cs231_2017

Summary: conv layer

To summarize, the Conv Layer

- Accepts a volume of size W1 x H1 x D1
- Requires four hyperparameters
  - **number of filters K**
  - **their spatial extent F**
  - **the stride S**
  - **the amount of zero padding P**
- Produces a volume of size W2 x H2 x D2
  - W2 = (W1 – F + 2P) / S + 1
  - H2 = (H1 – F + 2P) / S + 1
  - D2 = K
- With parameter sharing, it introduces F.F.D1 weights per filter, for a total of **(F.F.D1).K weights and K biases**
- In the output volume, the d^{th} depth slice (of size W2xH2) is the result of performing a valid convolution of the d^{th} filter over the input volume with a stride of S, and then offset by d^{th} bias

Common settings:
K = powers of 2 (32,64,128,256)
F=3, S=1, P=1
F=5, S=1, P=2
F=5, S=2, P=? (whatever fits)
F=1, S=1, P=0

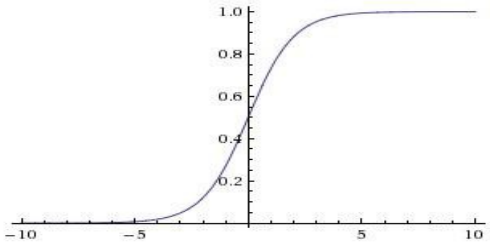
S. Credit: Stanford cs231_2017

Activation functions

- **Desirable properties:** mostly smooth, continuous, differentiable, fairly linear

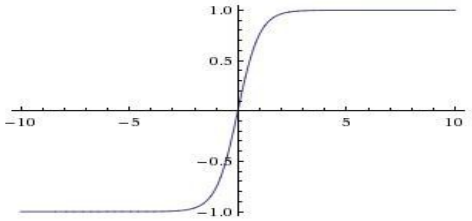
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

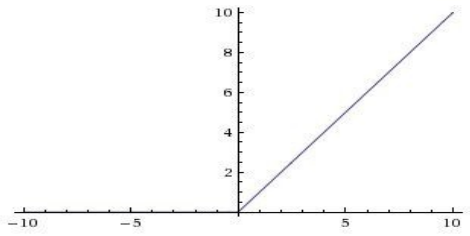


tanh

$$\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

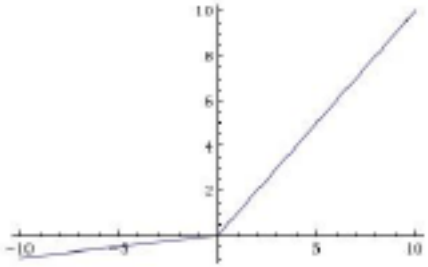


ReLu $\max(0, x)$



Leaky ReLu

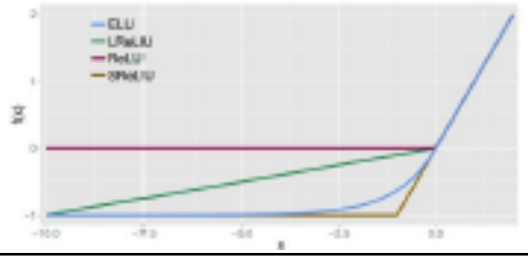
$\max(0.1x, x)$



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

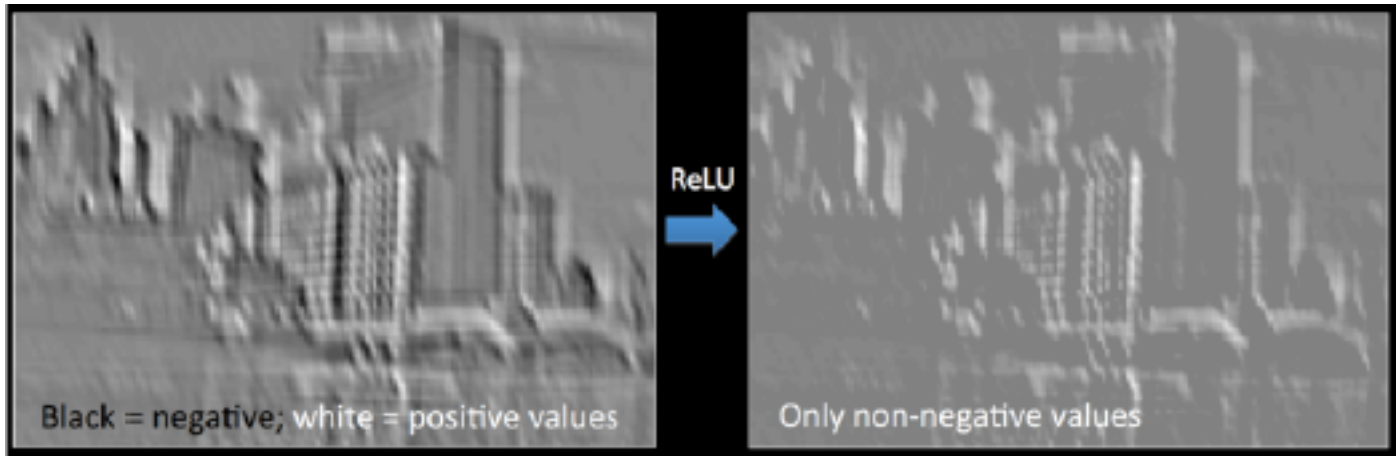
ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



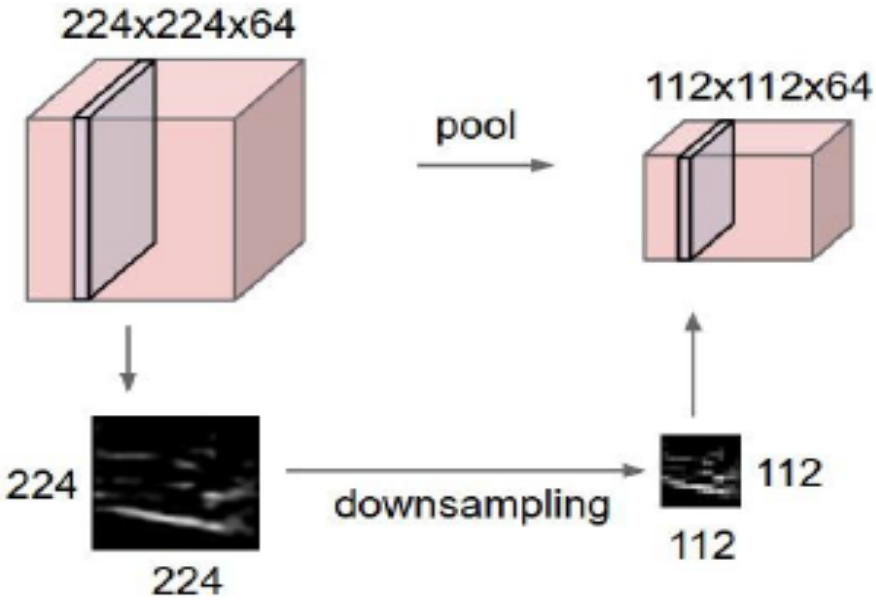
Activation functions

- Example



Pooling layer

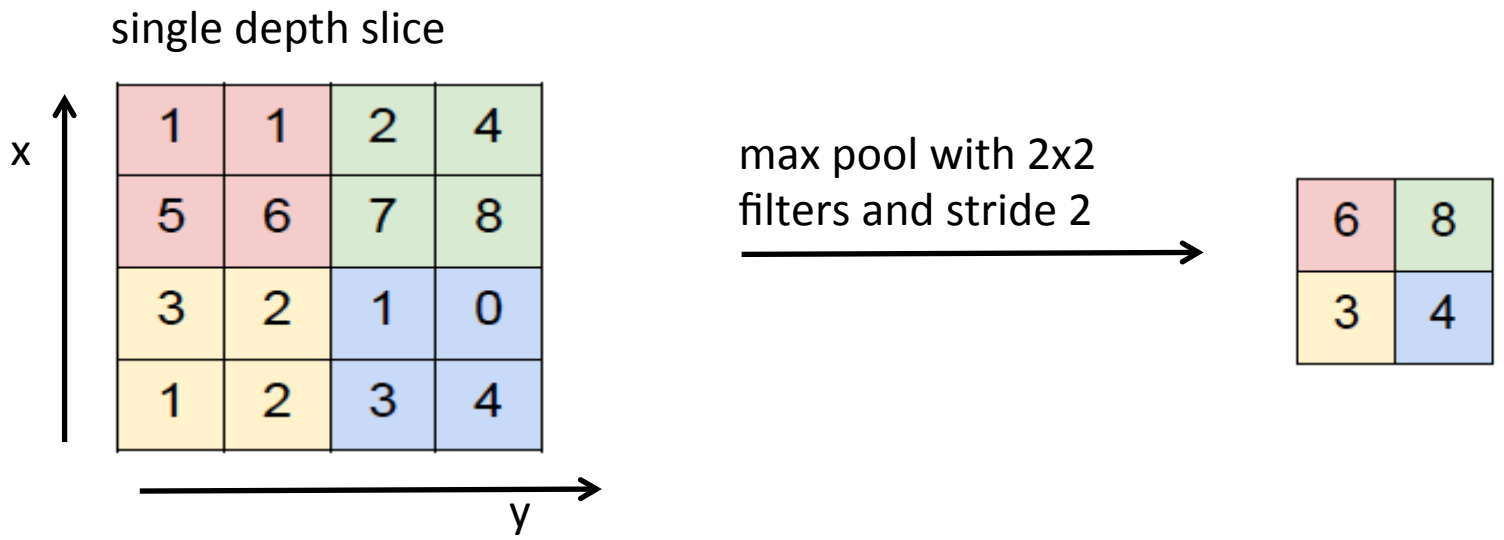
- Makes the representations smaller and more manageable for later layers
- Useful to get **invariance to small local changes**
- Operates over each activation map independently



S. Credit: Stanford cs231_2017

Pooling layer

- **Max pooling**

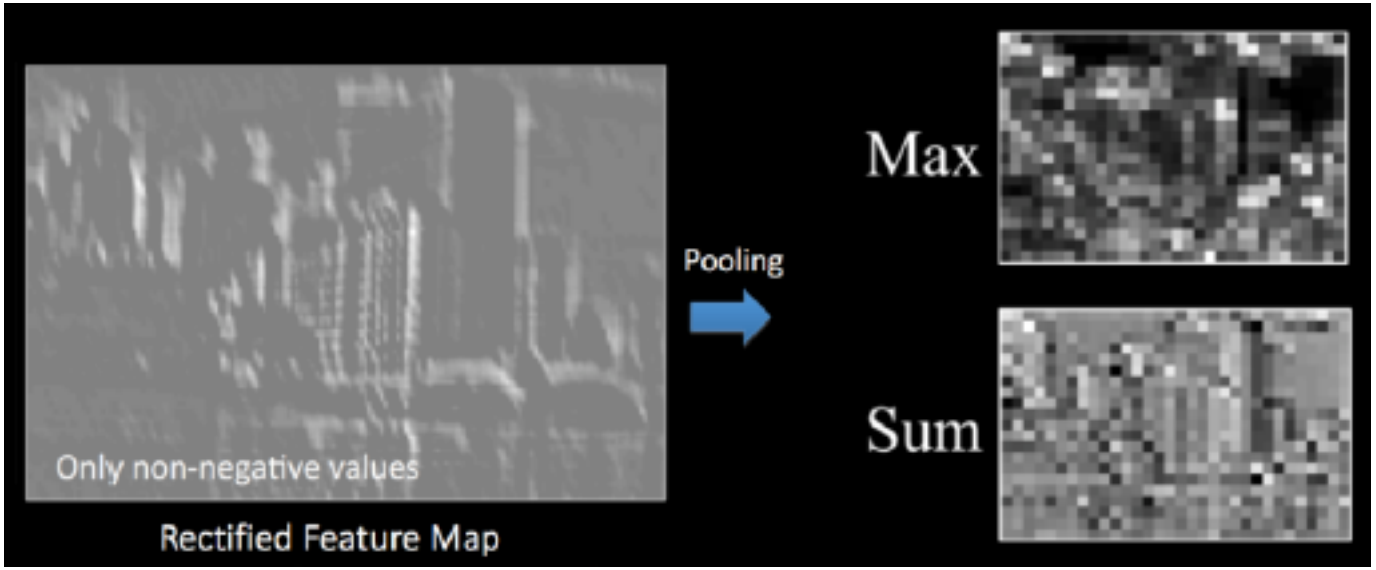


- Other pooling functions**: average pooling or L2-norm pooling**

S. Credit: Stanford cs231_2017

Pooling layer

- Example



Summary: pooling layer

To summarize, the pooling layer

- Accepts a volume of size W1xH1xD1
- Requires two hyperparameters
  - **the spatial extent F**
  - **the stride S**
- Produces a volume of size W2xH2xD2, where
  - W2 = (W1-F) / S + 1
  - H2 = (H1-F) / S + 1
  - D2 = D1
- Introduces zero parameters since it computes a fixed function of the input

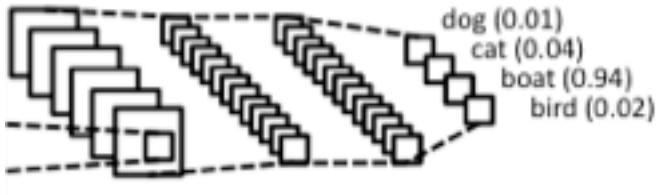
Common settings:
F=2, S=2
F=3, S=2

S. Credit: Stanford cs231_2017

Fully connected layer

- In the end it is common to add one or more fully (or densely) connected layers.
- Every neuron in the previous layer is connected to every neuron in the next layer (as in regular neural networks). Activation is computed as matrix multiplication plus bias
- At the output, softmax activation for classification

connections and weights
not shown here



4 possible
outputs

Fully connected layers and Convolutional layers

- A convolutional layer can be implemented as a fully connected layer
- The weight matrix is a large matrix that is mostly zero except for certain blocks (due to local connectivity)

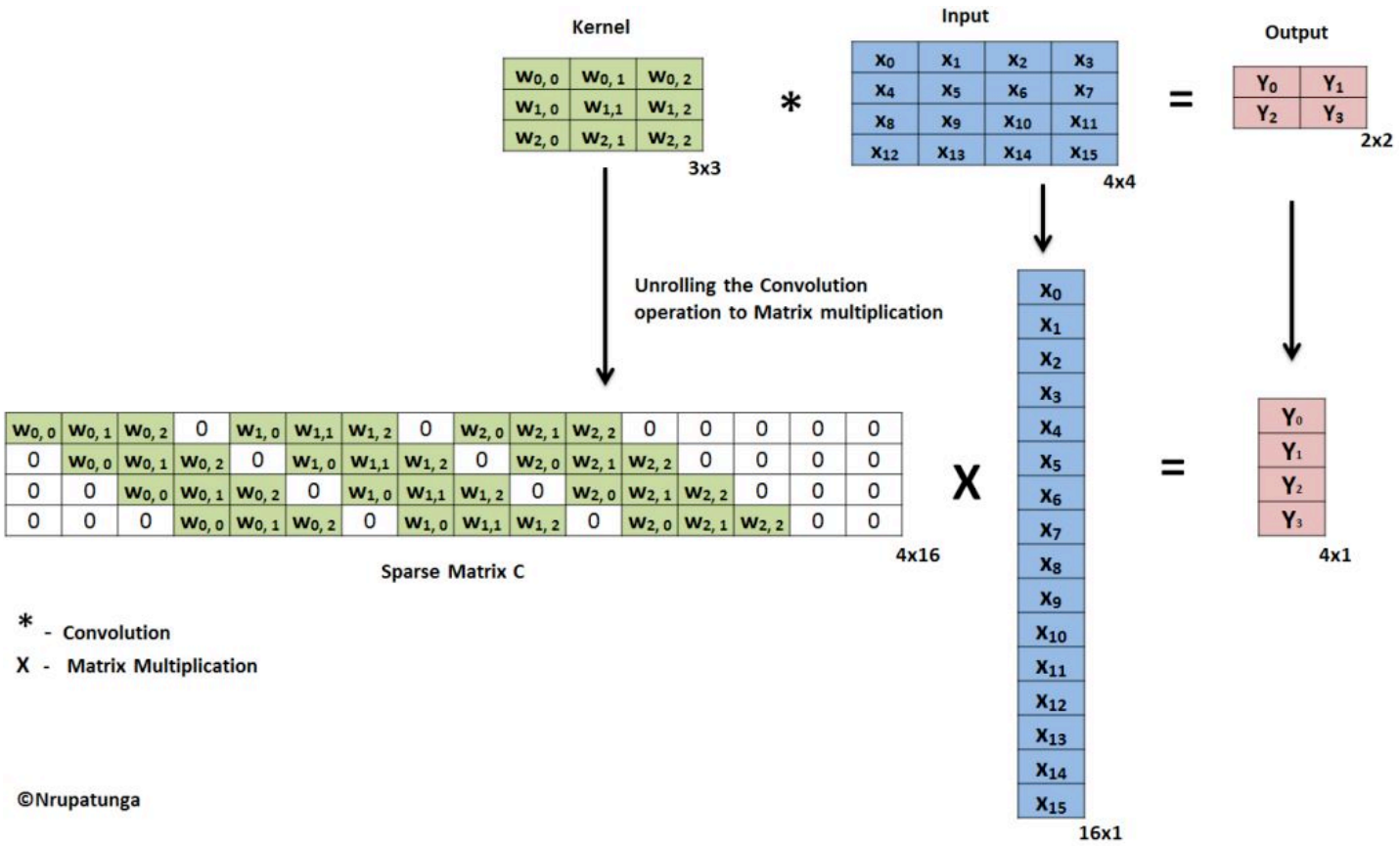
$$O = I * h = C.I$$

I input image 4x4 vectorized to 16x1
O output image 4x1 (later reshaped 2x2)
h 3x3 kernel; C 16x4 (weights)

$$h = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Fully connected layers and Convolutional layers



Fully connected layers and Convolutional layers

- Fully connected layers can also be viewed as convolutions with kernels that cover the entire input region

- Example:

  A fully convolutional layer with K=4096 neurons, input volume 32x32x512
  can be expressed as a convolutional layer with F=32 (size of kernel), P=0, S=1
  K=4096 (number of filters)
  the filter size is exactly the size of the input volume
  the output is 1x1x4096

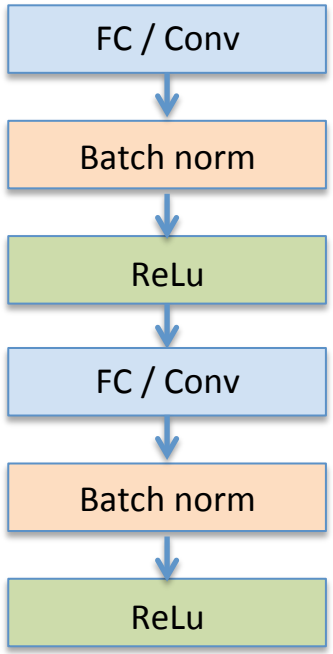
Batch normalization layer

- As learning progresses, the distribution of the layer inputs changes due to parameter updates (internal covariate shift)
- This can result in most inputs being in the non-linear regime of the activation function, slowing down learning
- Bach normalization is a technique to reduce this effect
  - Explicitly force the layer activations to have zero mean and unit variance w.r.t running batch estimates

$$\hat{x}^{(k)} = \frac{x^{(k)} - E(x^{(k)})}{\sqrt{\text{var}(x^{(k)})}}$$

  - Adds a learnable scale and bias term to allow the network to still use the nonlinearity

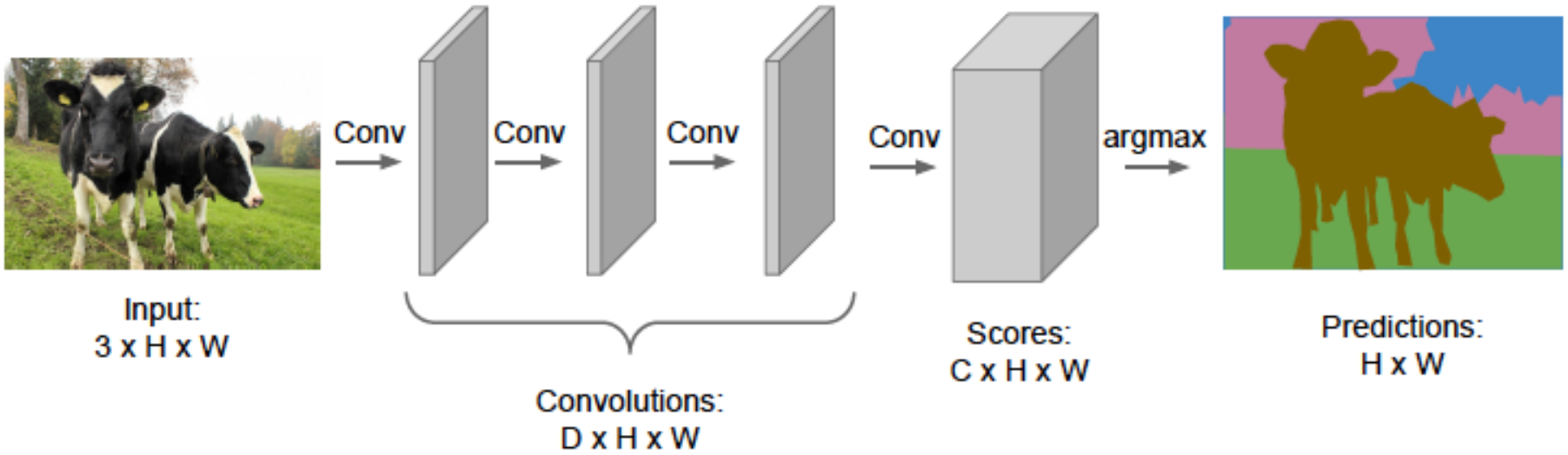
$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$



Ioffe and Szegedy, 2015. "Batch normalization: accelerating deep network training by reducing internal covariate shift"

Upsampling layers: recovering spatial shape

- Motivation: semantic segmentation. Make predictions for all pixels at once

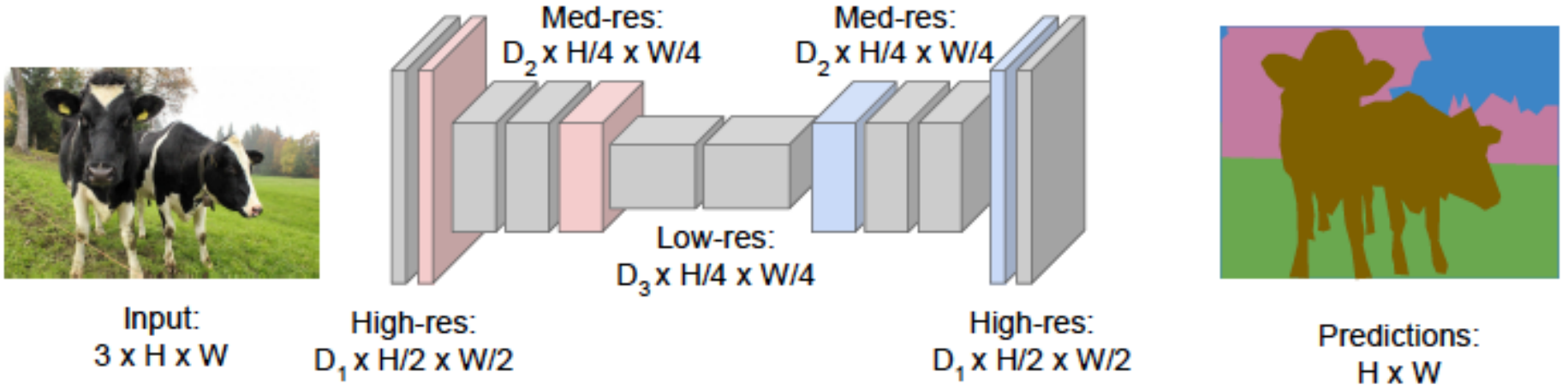


Problem: convolutions at original image resolution will be very expensive

S. Credit: Stanford cs231_2017

Upsampling layers: recovering spatial shape

- Motivation: semantic segmentation. Make predictions for all pixels at once



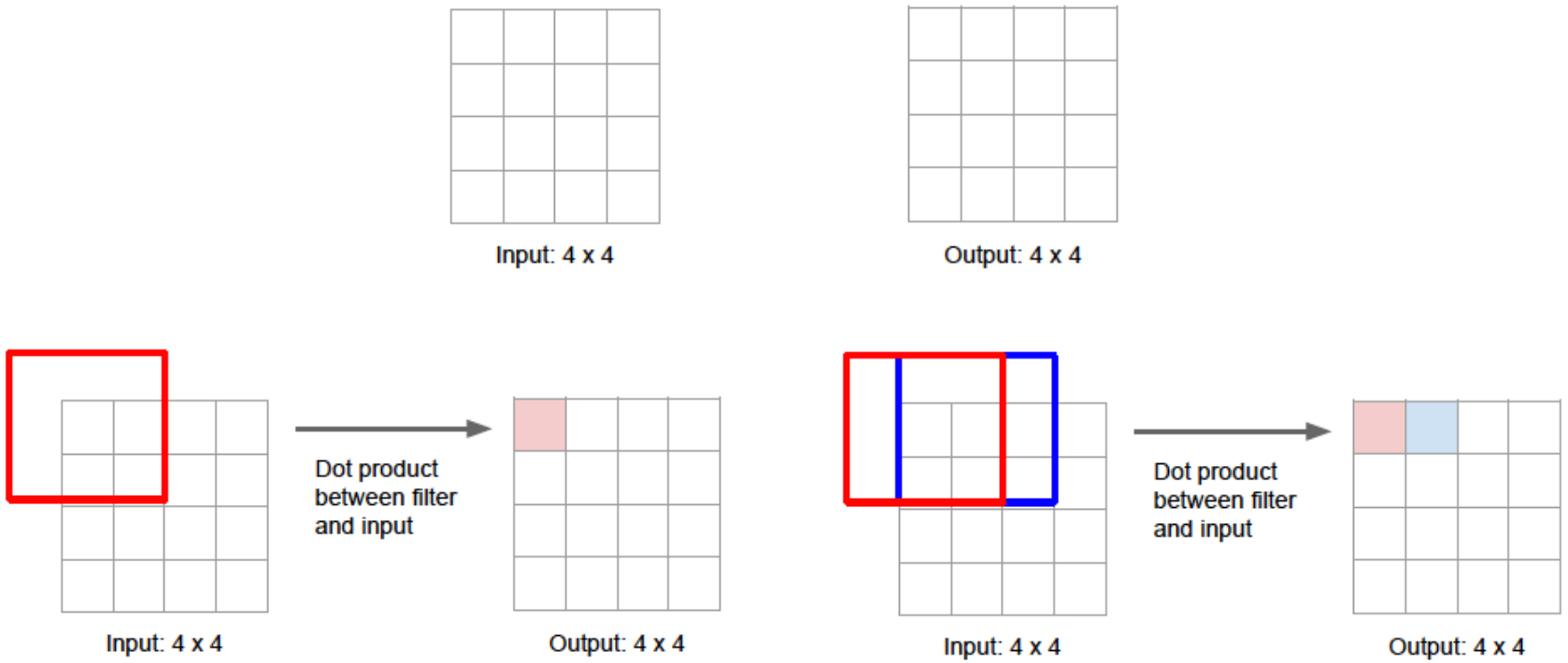
Design a network as a sequence of convolutional layers **with downsampling and upsampling**

Other applications: super-resolution, flow estimation, generative modeling

S. Credit: Stanford cs231_2017

Learnable upsampling

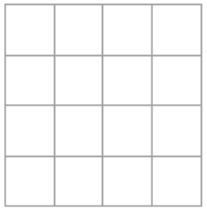
- Recall: 3x3 convolution, stride 1, pad 1



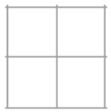
S. Credit: Stanford cs231_2017

Learnable upsampling

- Recall: 3x3 convolution, stride 2, pad 1



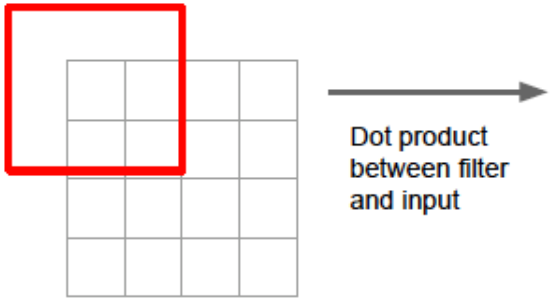
Input: 4 x 4



Output: 2 x 2

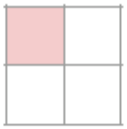
Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

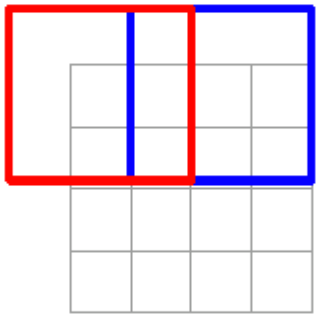


Input: 4 x 4

Dot product between filter and input



Output: 2 x 2



Input: 4 x 4

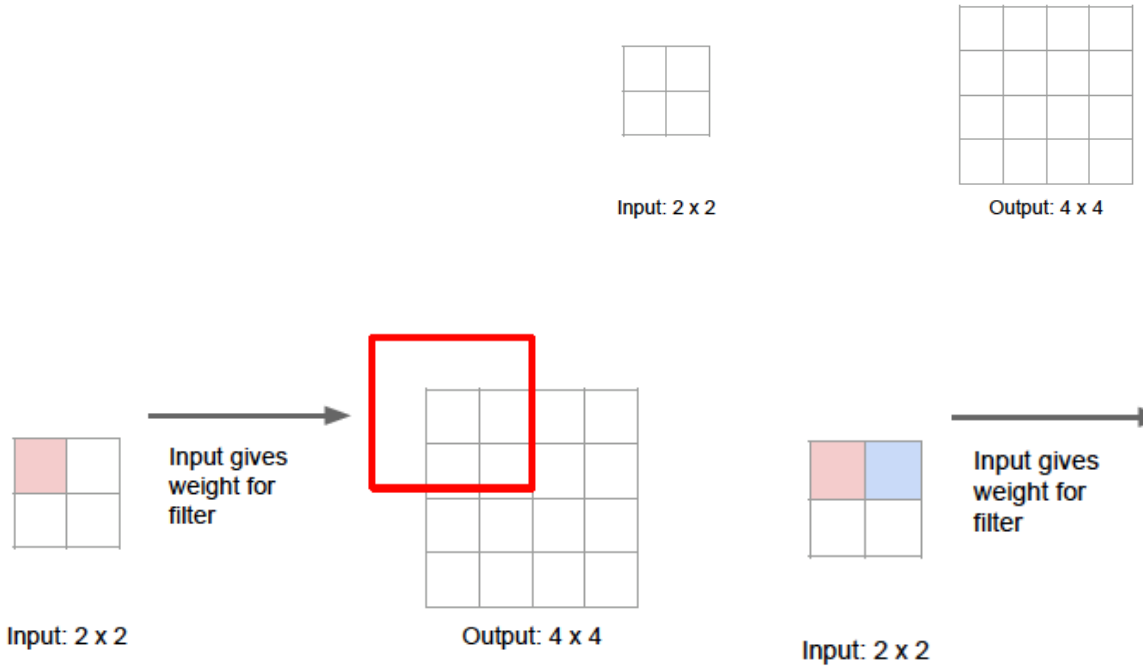
Dot product between filter and input

Output: 2 x 2

S. Credit: Stanford cs231_2017

Learnable upsampling: transposed convolution

- 3x3 transposed convolution, stride 2, pad1



Filter moves 2 pixels **in the output** for every one pixel in the input

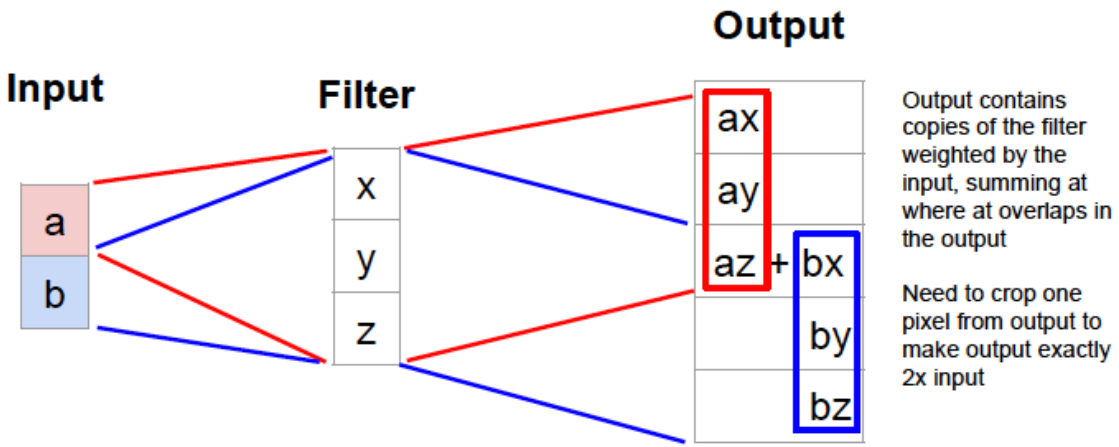
Stride gives ratio between movement in input and output

Sum where output overlaps

S. Credit: Stanford cs231_2017

Learnable upsampling: transposed convolution

- **1D example**



Other names

-transposed convolution
-backward strided convolution,
-fractionally strided convolution,
-upconvolution,
-"deconvolution"

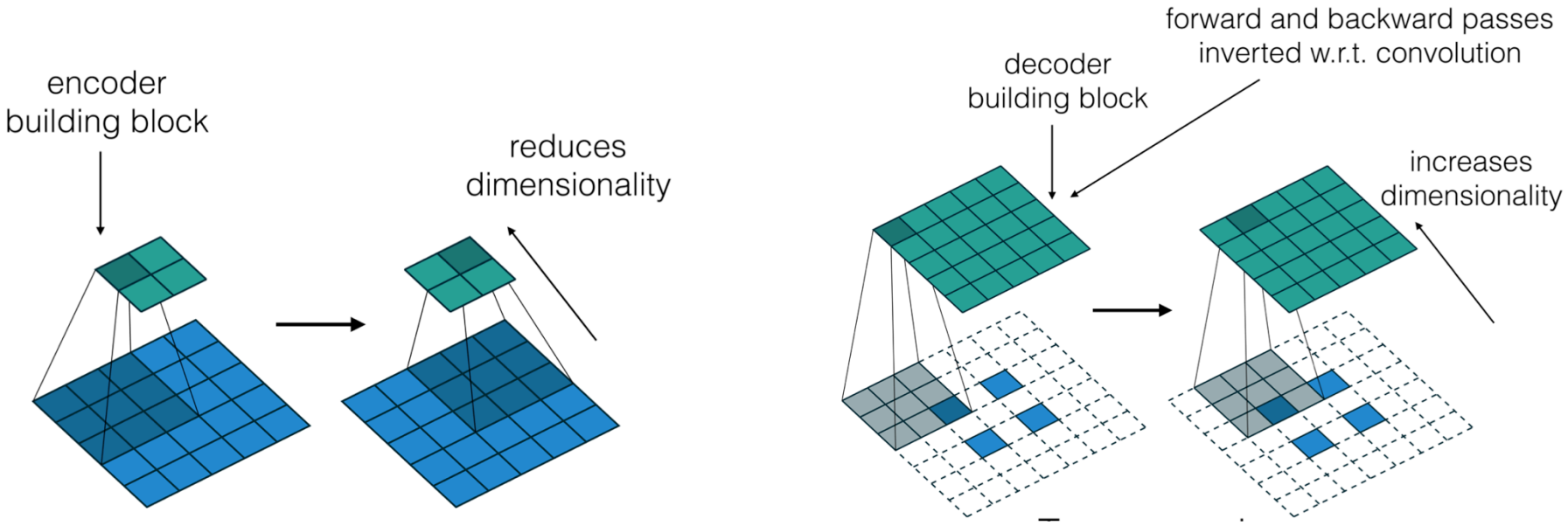
More info:

Dumoulin et al, A guide to convolution arithmetic for deep learning, 2016
https://github.com/vdumoulin/conv_arithmetic
Shi, Is the deconvolution layer the same as a convolutional layer?, 2016

S. Credit: Stanford cs231_2017

Learnable upsampling

It is always possible to emulate a transposed convolution with a direct convolution (with fractional stride)

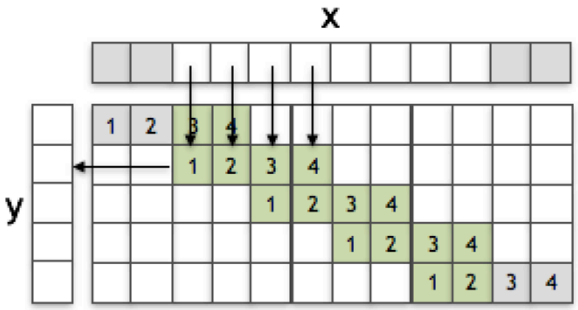


More info:
Dumoulin et al, A guide to convolution arithmetic for deep learning, 2016
https://github.com/vdumoulin/conv_arithmetic

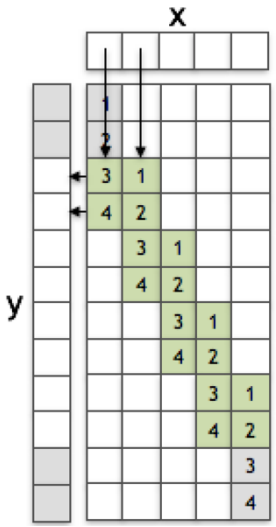
Learnable upsampling

Transposed convolution vs fractional strided convolution: the two operators can achieve the same result if the filters are learned.

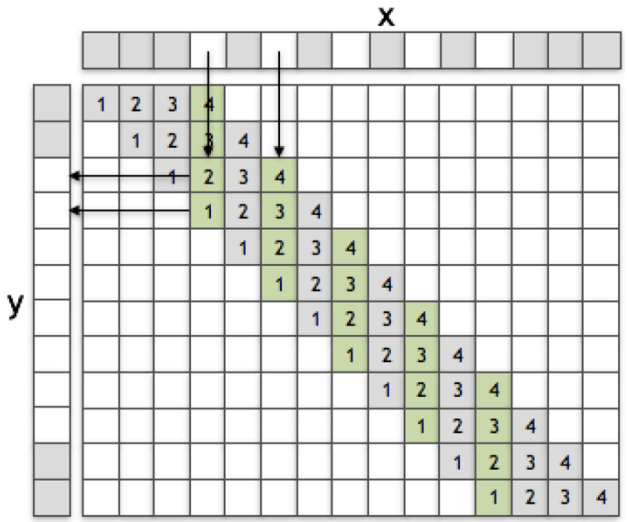
1D example:



1D Convolution with stride 2
x input size 8
y output size 5
filter size 4



1D Transposed Convolution with stride 2



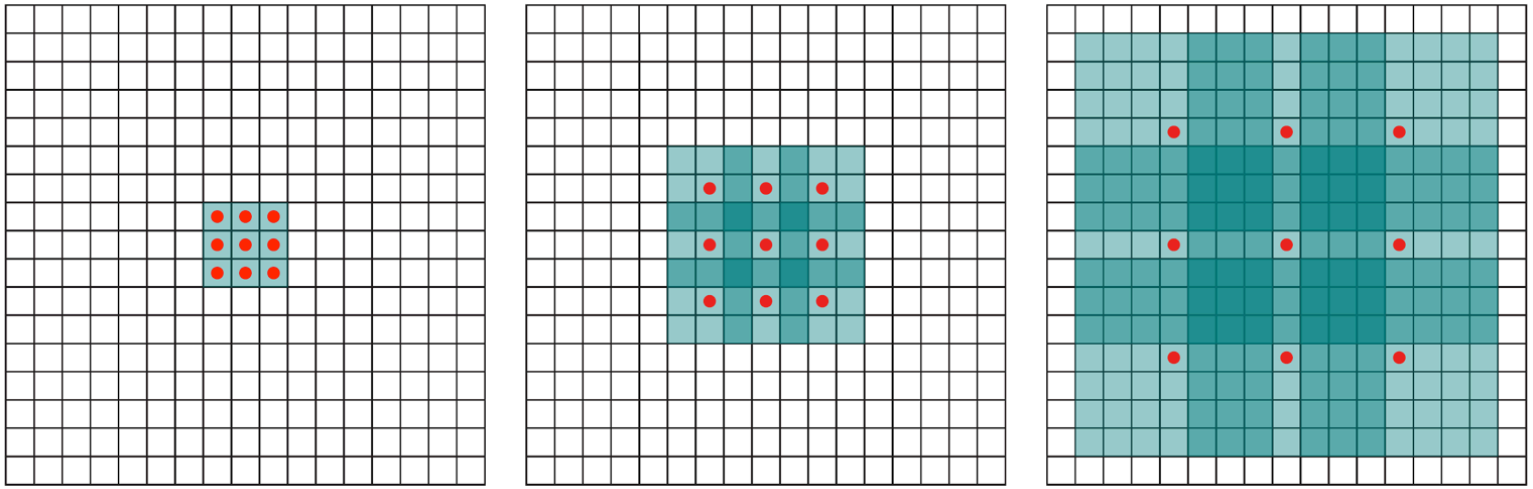
1D Subpixel convolution with stride 1/2

Shi, Is the deconvolution layer the same as a convolutional layer?, 2016

Dilated convolutions

- Systematically aggregate multiscale contextual information without losing resolution

Usual convolution $(F * k)(p) = \sum_t F(p-t)k(t)$ Dilated convolution $(F *_l k)(p) = \sum_t F(p-lt)k(t)$



The receptive field grows exponentially as you add more layers $F_{i+1} = F_i *_{2^i} k_i \quad i = 0,1,2,\ldots n-2$

Number of parameters increases linearly as you add more layers

Yu, Koltun, Multi-scale context aggregation by dilated convolutions, 2016

Some architectures

for Visual Recognition

ImageNet: ILSVRC

IMAGENET

Large Scale Visual Recognition Challenge

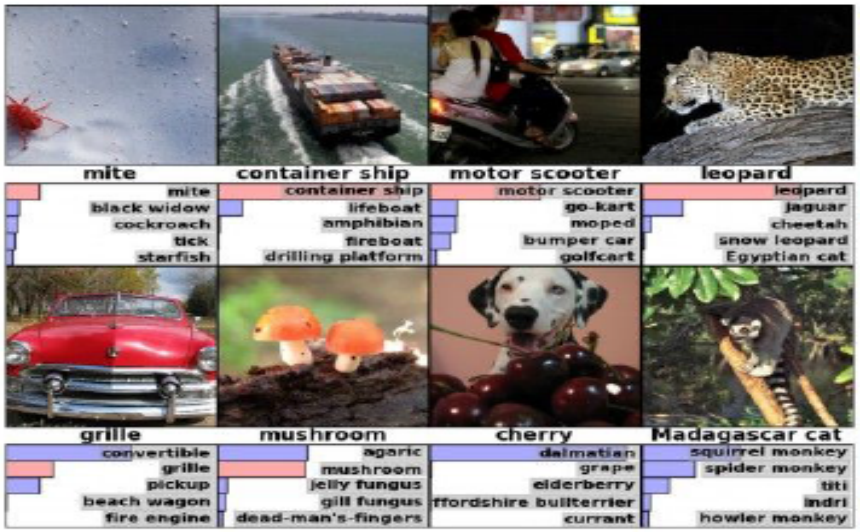
Image Classification

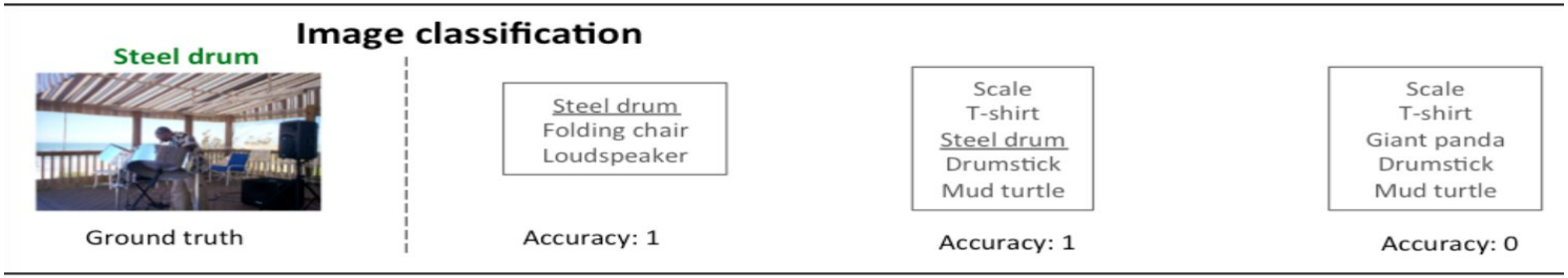
1000 object classes (categories)

Images:

- 1.2 M train
- 100k test.

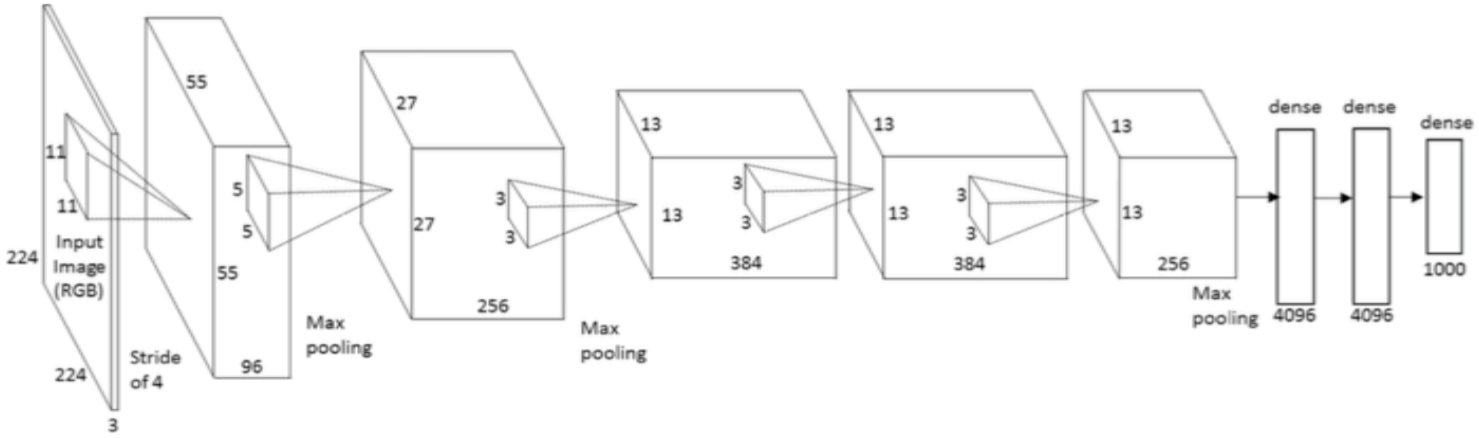
Metric: top 5 error rate (predict 5 classes)





www.image-net.org/challenges/LSVRC/

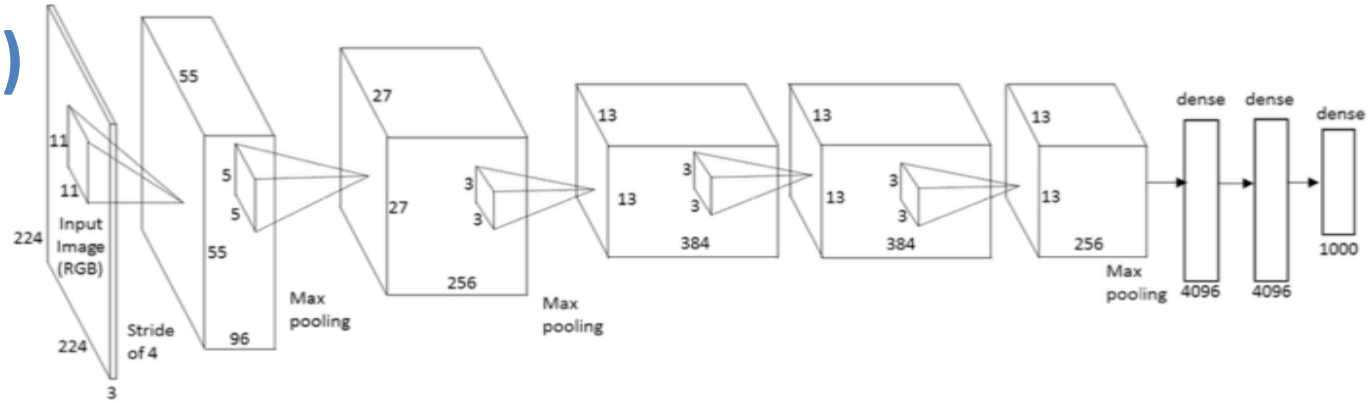
AlexNet (2012)



- Similar framework to LeNet:
  - 8 parameter layers (5 convolutional, 3 fully connected)
  - Max pooling, ReLu nonlinearities
  - 650,000 units, 60 million parameters)
  - trained on two GPUs for a week
  - Dropout regularization

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

AlexNet (2012)



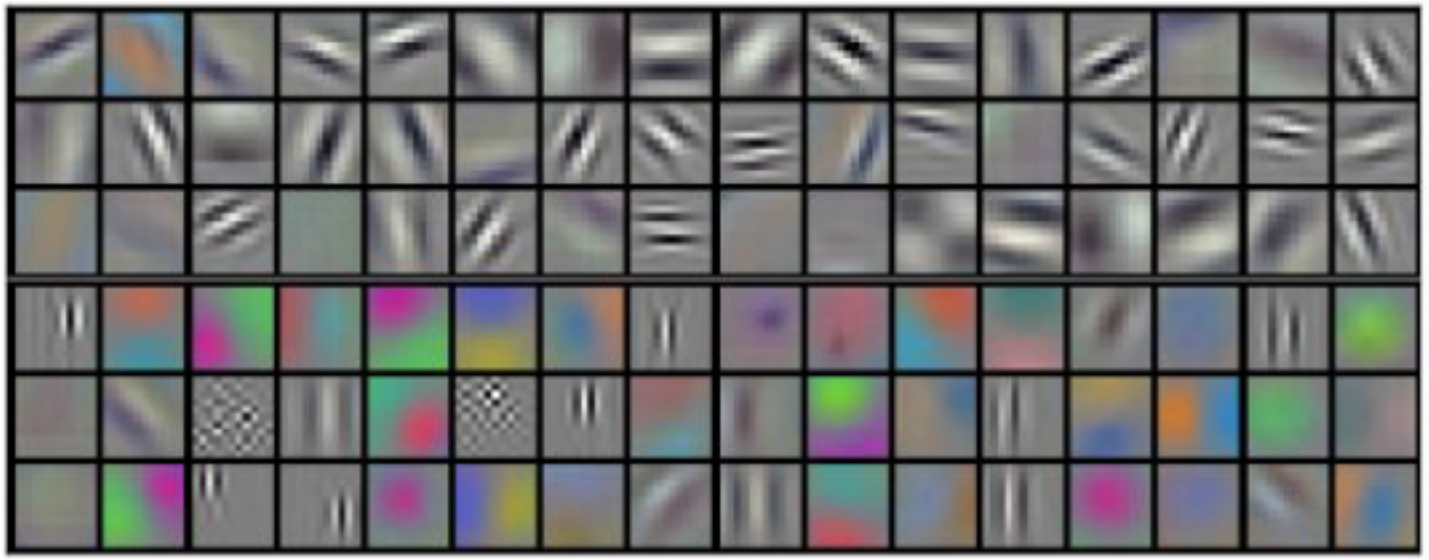
Full AlexNet architecture:
8 layers
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

ILSVRC 2012 winner
7 CNN ensemble: 18.2% -> 15.4%

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4

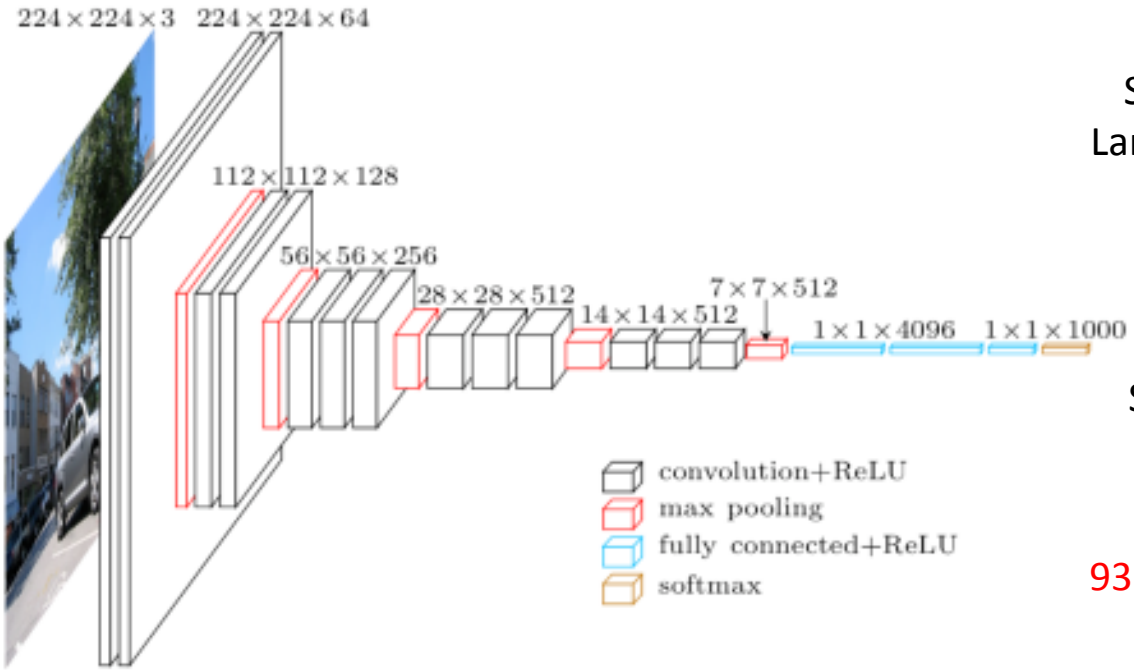
AlexNet (2012)

- Visualization of the 96 11x11 filters learned by the first layer



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

VGGNet-16 (2014)



Seq. of deeper nets trained progressively
Large receptive fields replaced by 3x3 conv

Only:
3x3 CONV stride 1, pad 1 and
2x2 MAX POOL stride 2

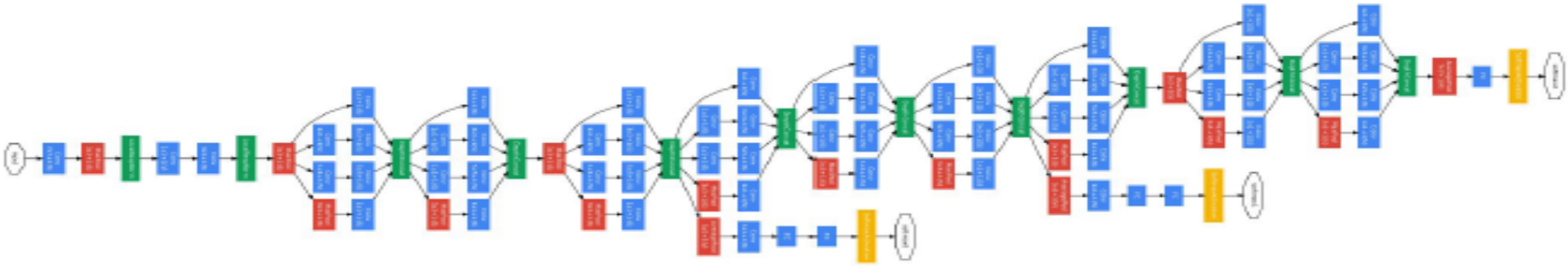
Shows that depth is a critical component for good performance

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
most memory is in the early CONV

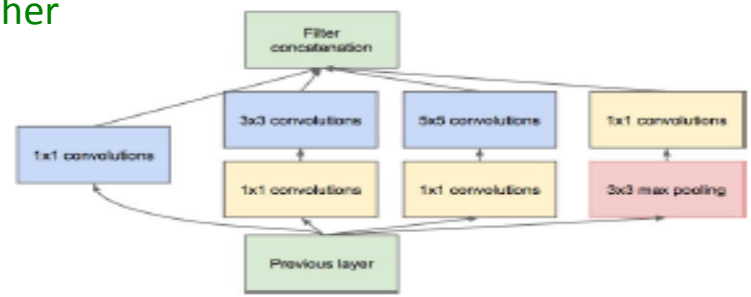
TOTAL params: 138M parameters
most parameters are in late FC

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

GoogLeNet (2014)



Convolution
Pooling
Softmax
Other



9 inception modules: Network in a network…

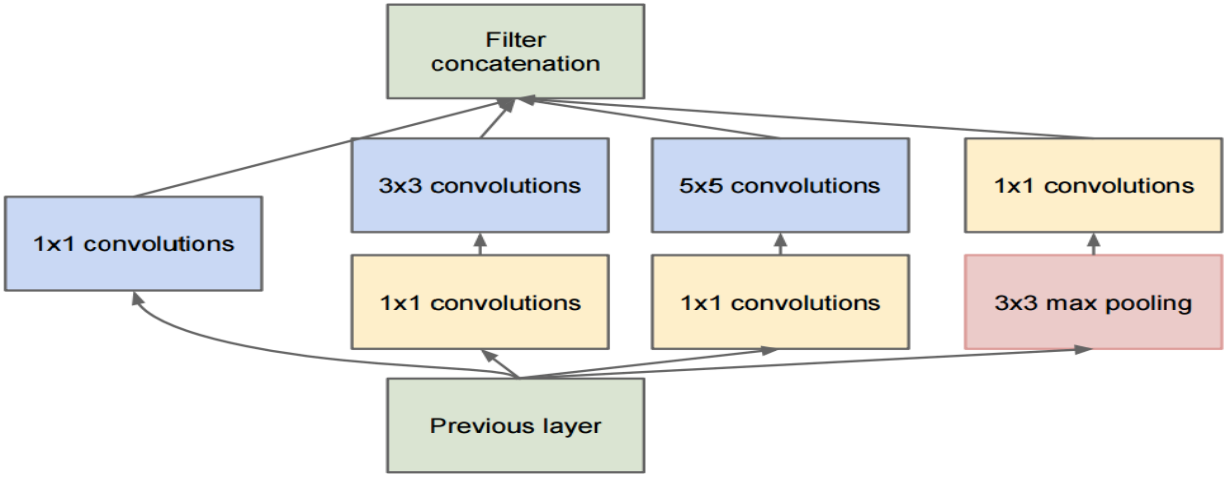
- Inception module that dramatically reduced the number of parameters
- Uses average pooling instead of FC

Compared to AlexNet
12x less parameters (5M vs 60M)
2x more compute 6.67% (vs 16,4%)
22 layers

C. Szegedy et al., Going deeper with convolutions, CVPR 2015

GoogLeNet (2014)

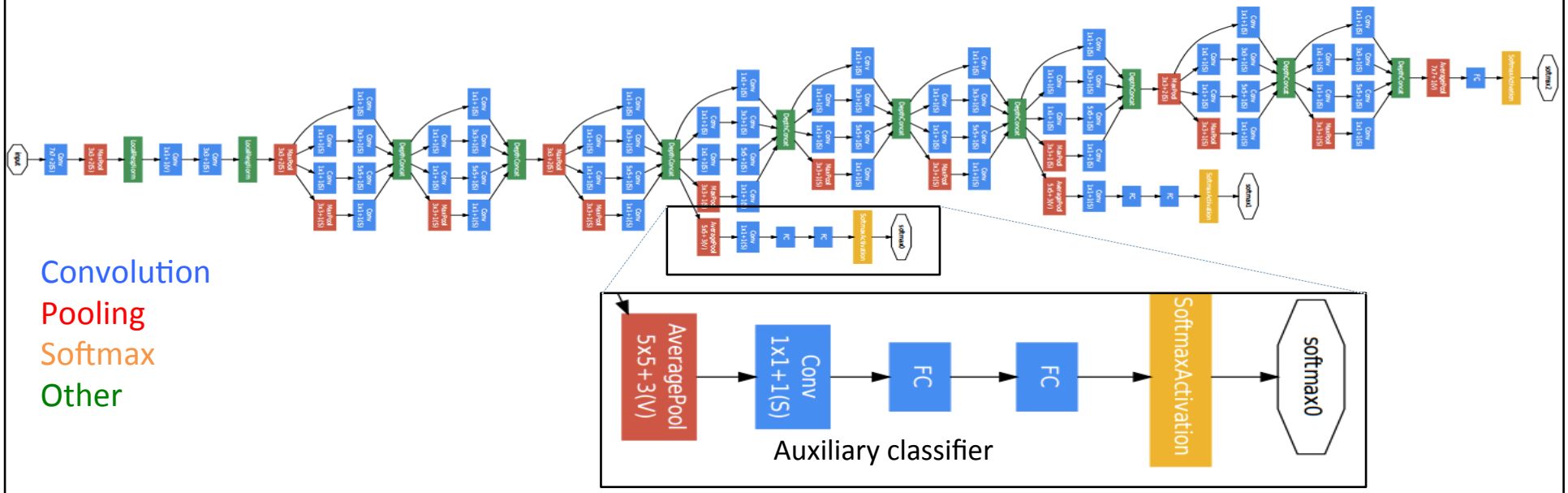
- The Inception Module
  - Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps
  - Use 1x1 convolutions for dimensionality reduction before expensive convolutions



C. Szegedy et al., Going deeper with convolutions, CVPR 2015

GoogLeNet (2014)

- Two Softmax Classifiers at intermediate layers combat the vanishing gradient while providing regularization at training time.

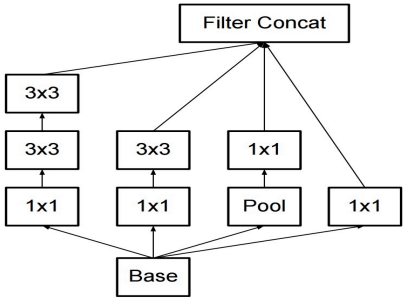


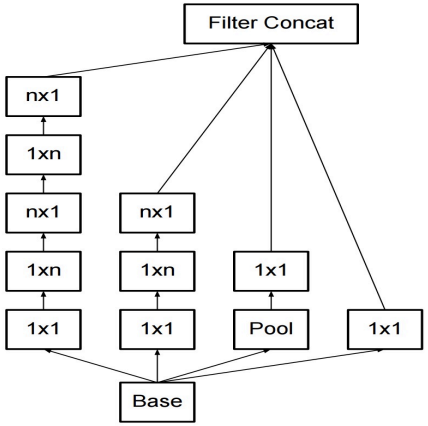
...and no fully connected layers needed !

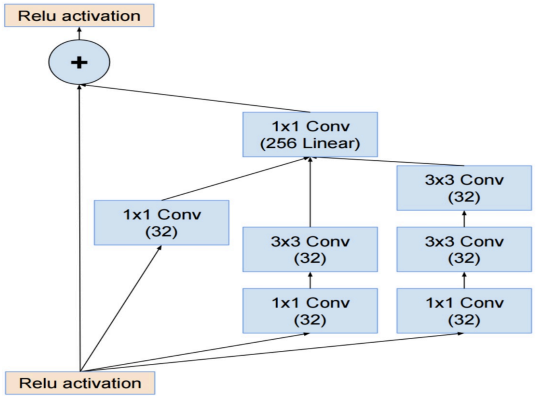
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

Inception v2, v3, v4

- Regularize training with batch normalization, reducing importance of auxiliary classifiers
- More variants of inception modules with aggressive factorization of filters







C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016

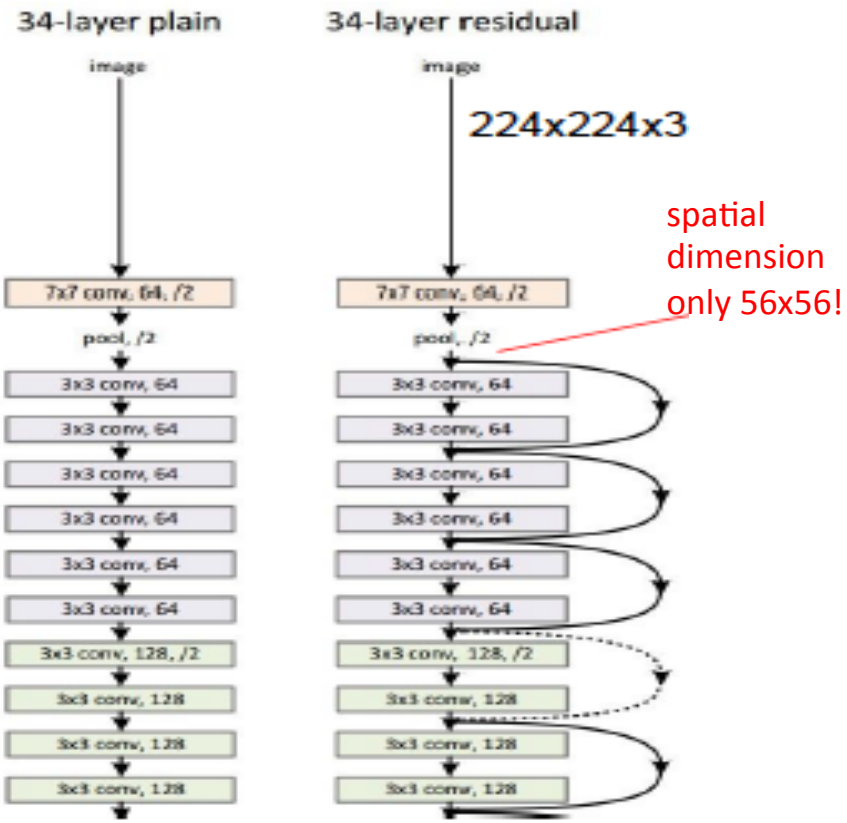
C. Szegedy et al., Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

ResNet (2015)

ILSVRC 2015 winner (3.6% top 5 error)

MSRA: ILSVRC & COCO 2015 competitions
- ImageNet Classification: “ultra deep”, 152-leyers
- ImageNet Detection: 16% better than 2nd
- ImageNet Localization: 27% better than 2nd
- COCO Detection: 11% better than 2nd
- COCO Segmentation: 12% better than 2nd

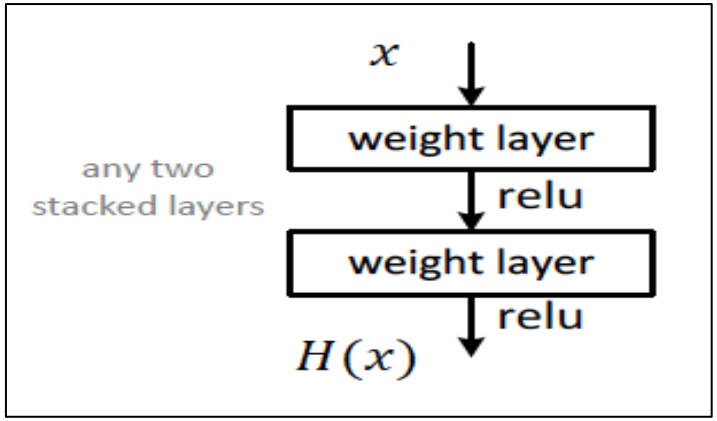
2-3 weeks of training on 8 GPU machine
at runtime: faster than a VGGNet!
(even though it has 8x more layers)



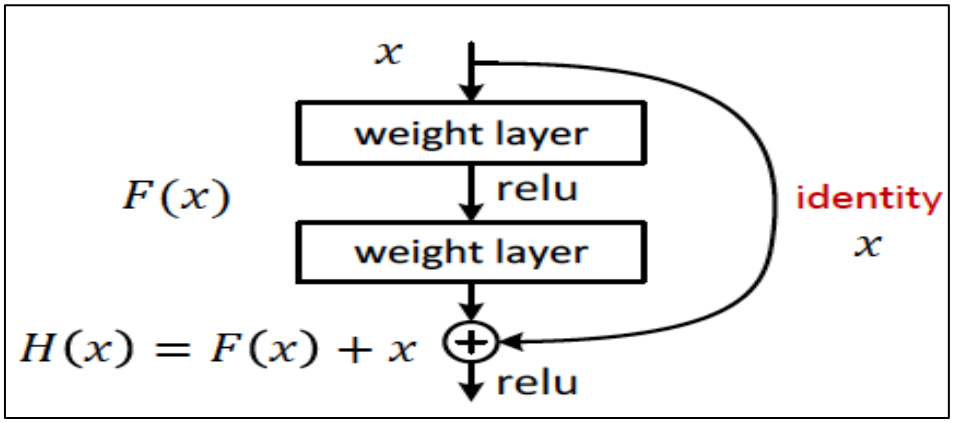
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet (2015)

Plain net



Residual net



Residual learning: reformulate the layers as **learning residual functions** with reference to the layer inputs, instead of learning unreferenced functions

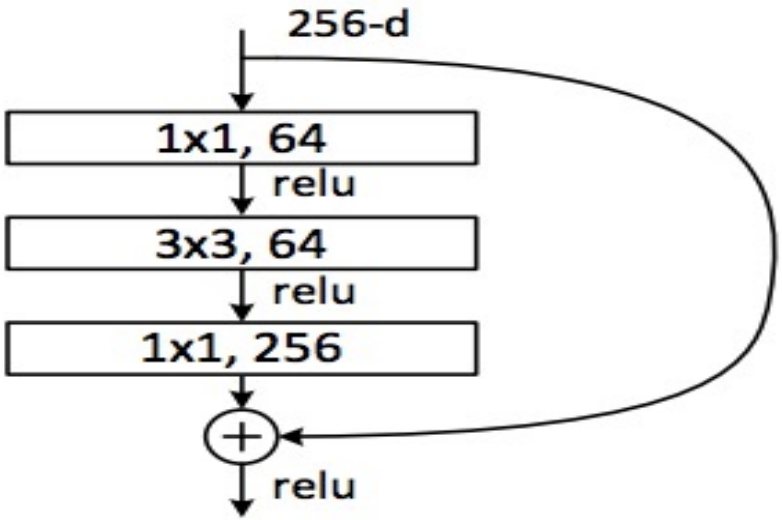
- Introduces skip or shortcut connections (existing before in the literature)
- Make it easy for net layers to represent the identity mapping

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

ResNet (2015)

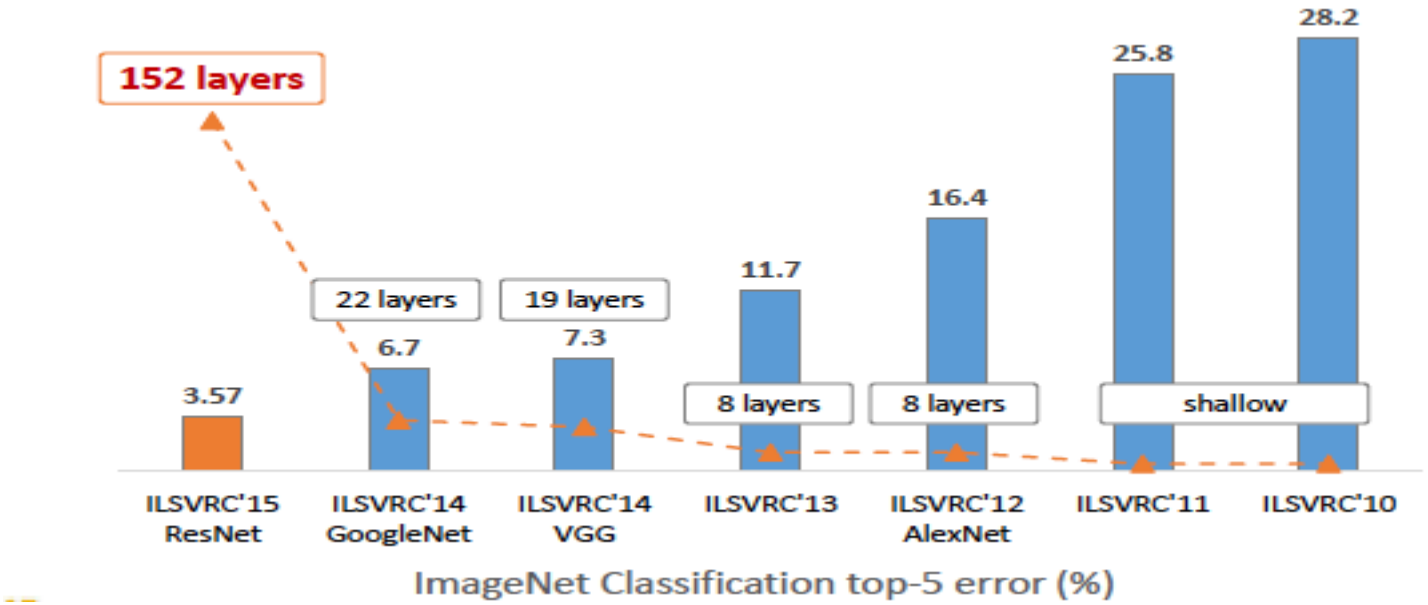
- Deeper residual module (bottleneck)



- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

Deeper models



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016

ILSVRC 2012-2015 summary

Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (**AlexNet**, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
ResNet (152 layers)	2015	1st	3.57%	
Human expert*			5.1%	

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

Summary

- Convolutional neural networks are a specialized kind of neural network for processing data that has a known, grid-like topology
- CNNs leverage these ideas:
  - local connectivity
  - parameter sharing
  - pooling / subsampling hidden units
- Layers: convolutional, non-linear activation, pooling, upsampling, batch normalization
- Architectures for object recognition in images