Machine Learning, Lecture 10

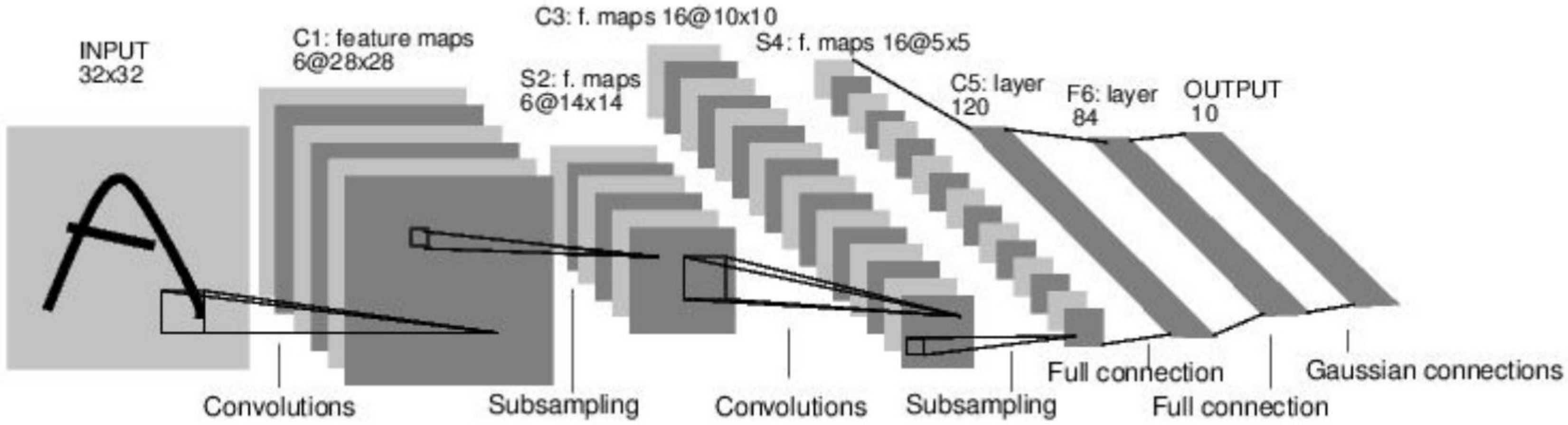
Convolutional Neural Networks

Radoslav Neychev

Outline

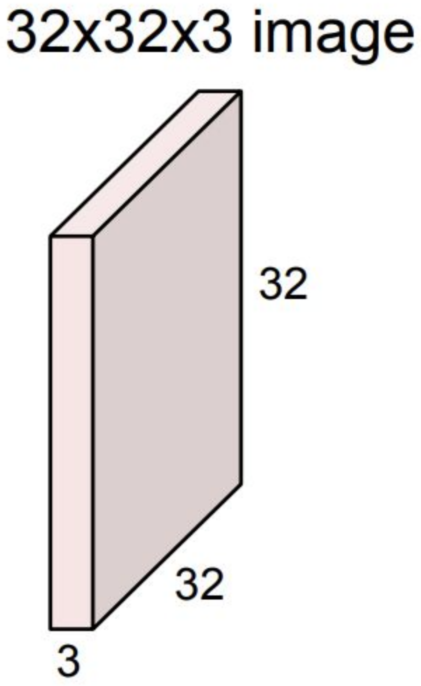
1. Convolutional layer structure.
2. Pooling layers.
3. Top architectures overview.

CNN



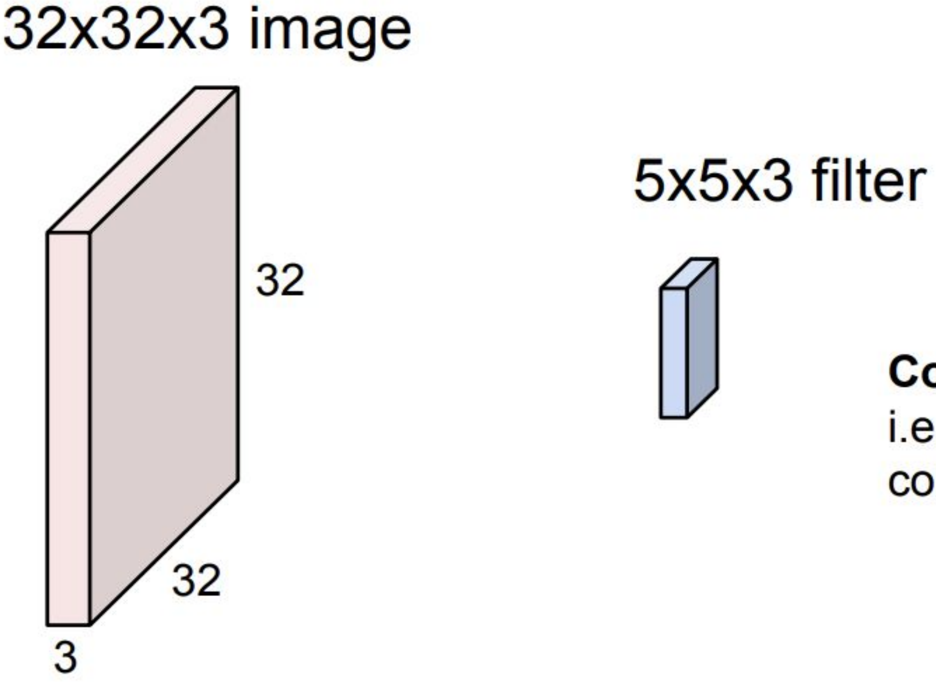
[LeNet-5, LeCun 1998]

Convolutional layer



source

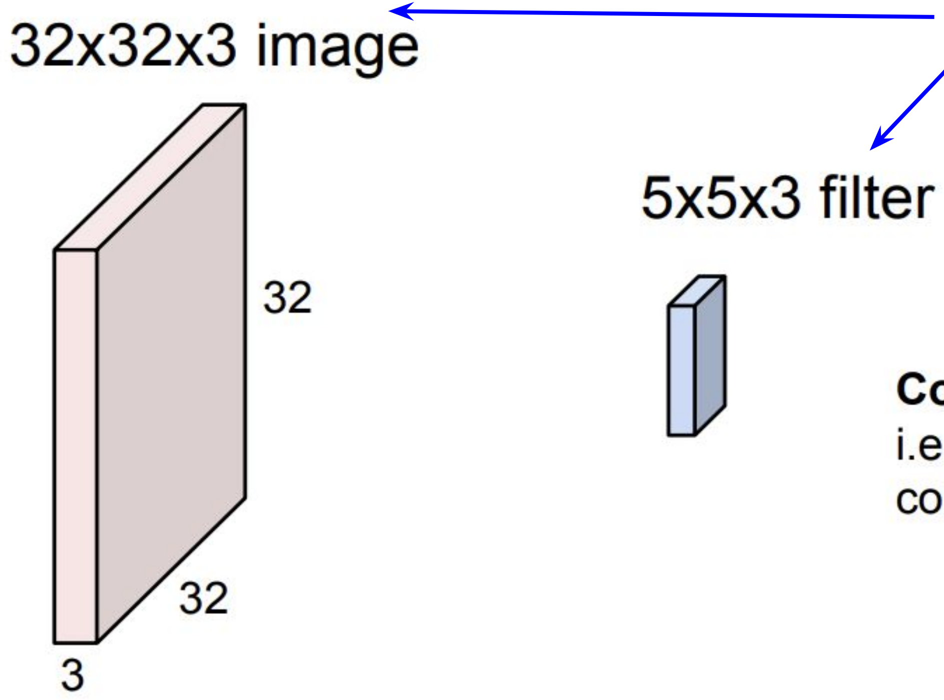
Convolutional layer



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

source

Convolutional layer

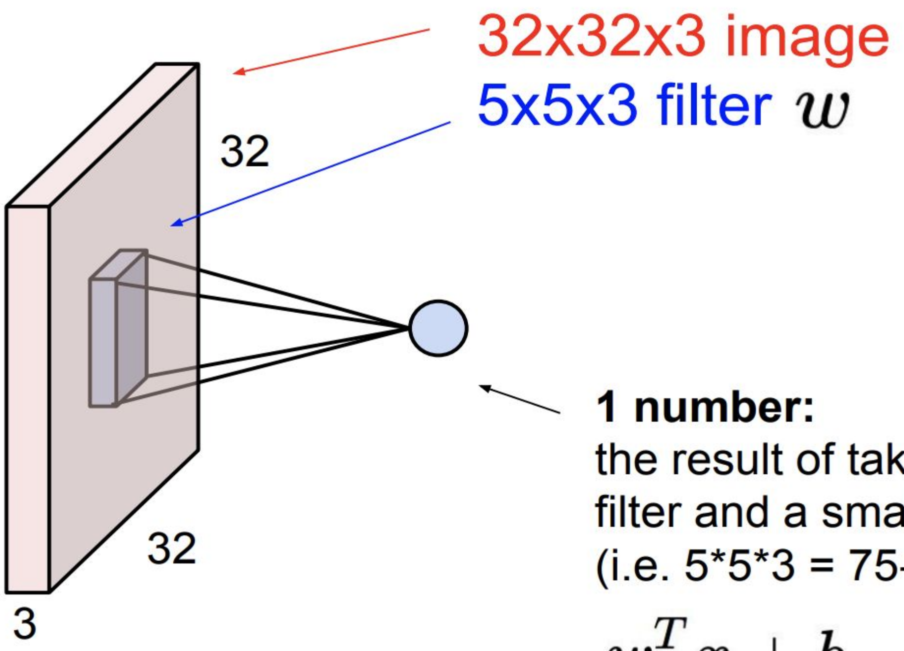


Filters extend the depth of the original image

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

source

Convolutional layer

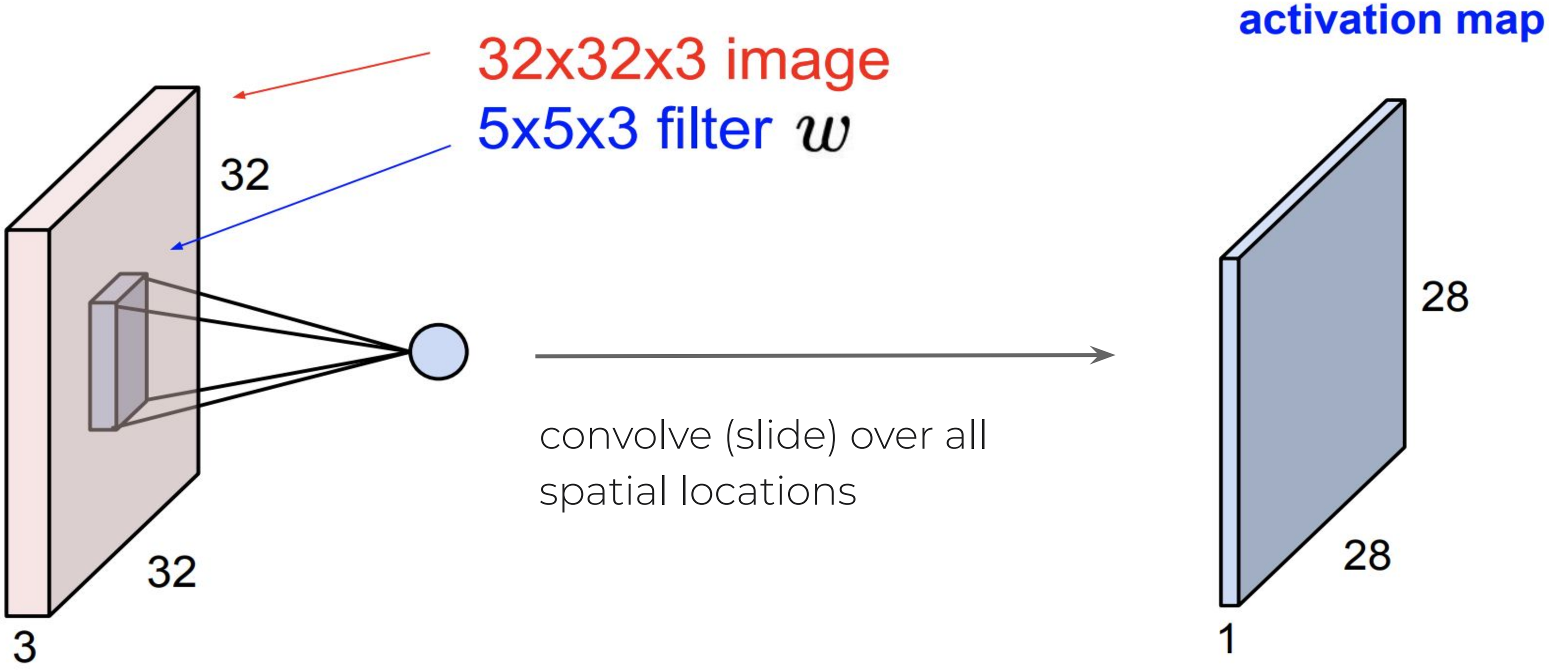


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

source

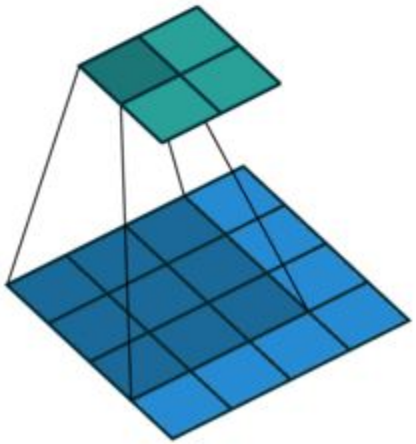
Convolutional layer



convolve (slide) over all spatial locations

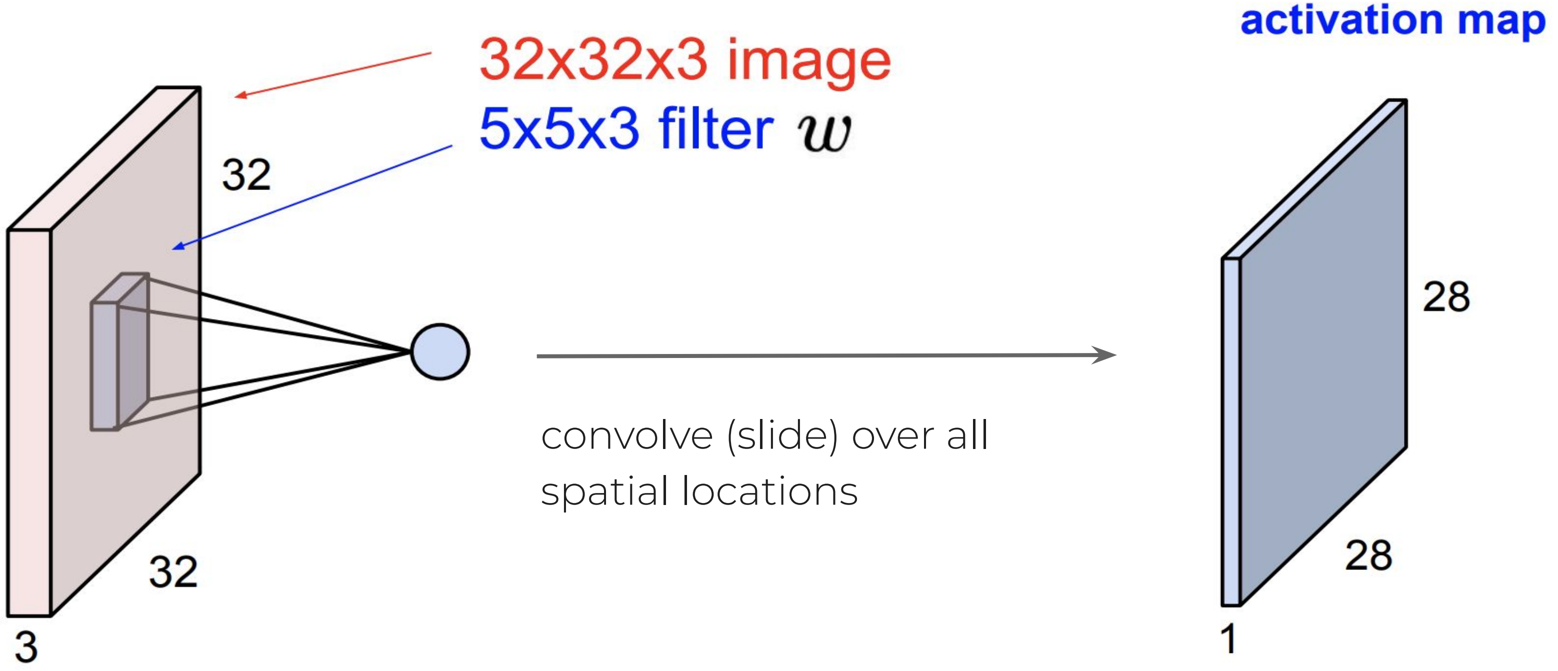
source

Convolutional layer



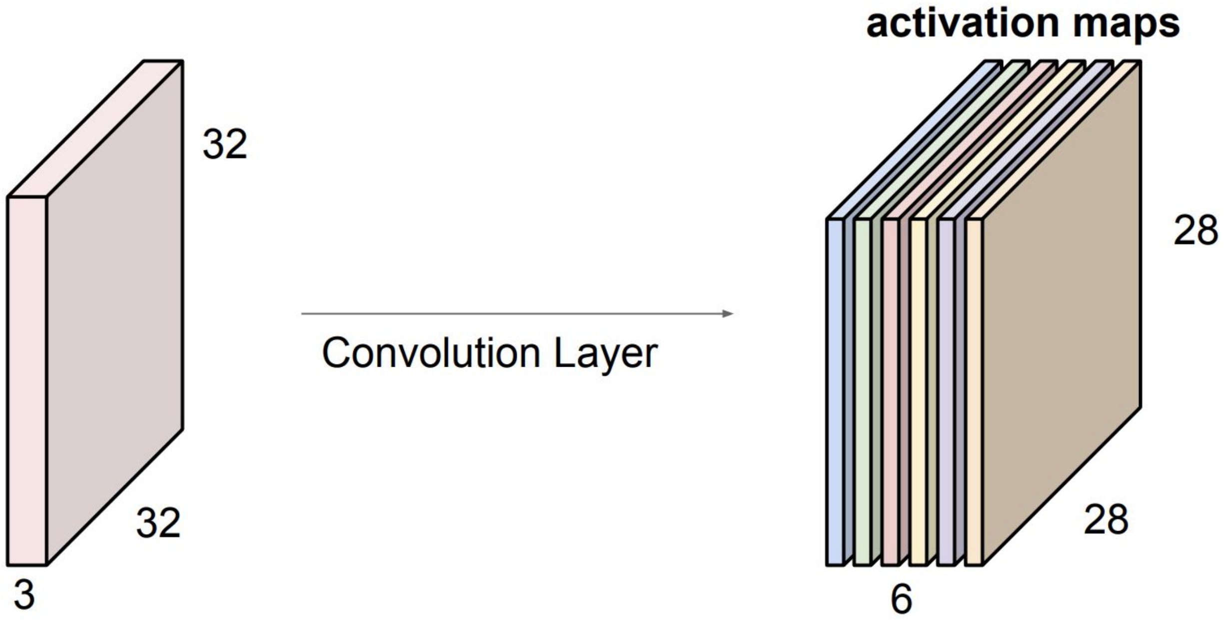
source

Convolutional layer



Convolutional layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

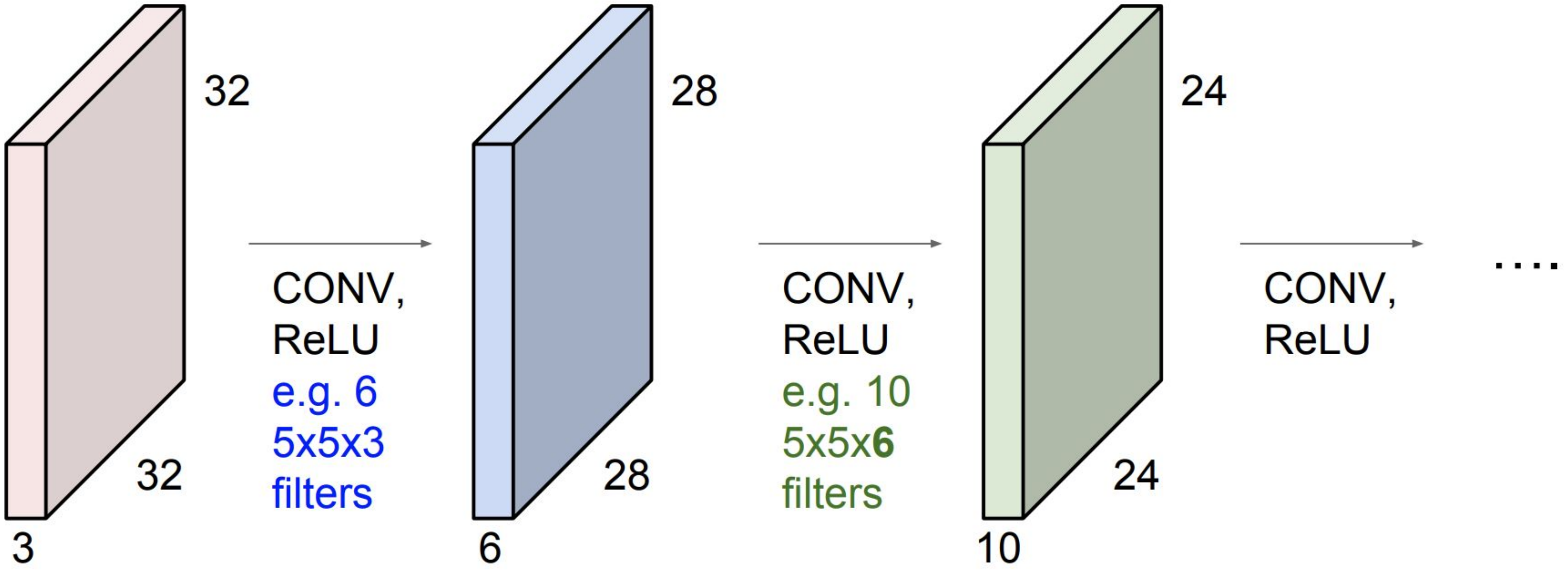


We stack these up to get a "new image" of size 28x28x6!

source

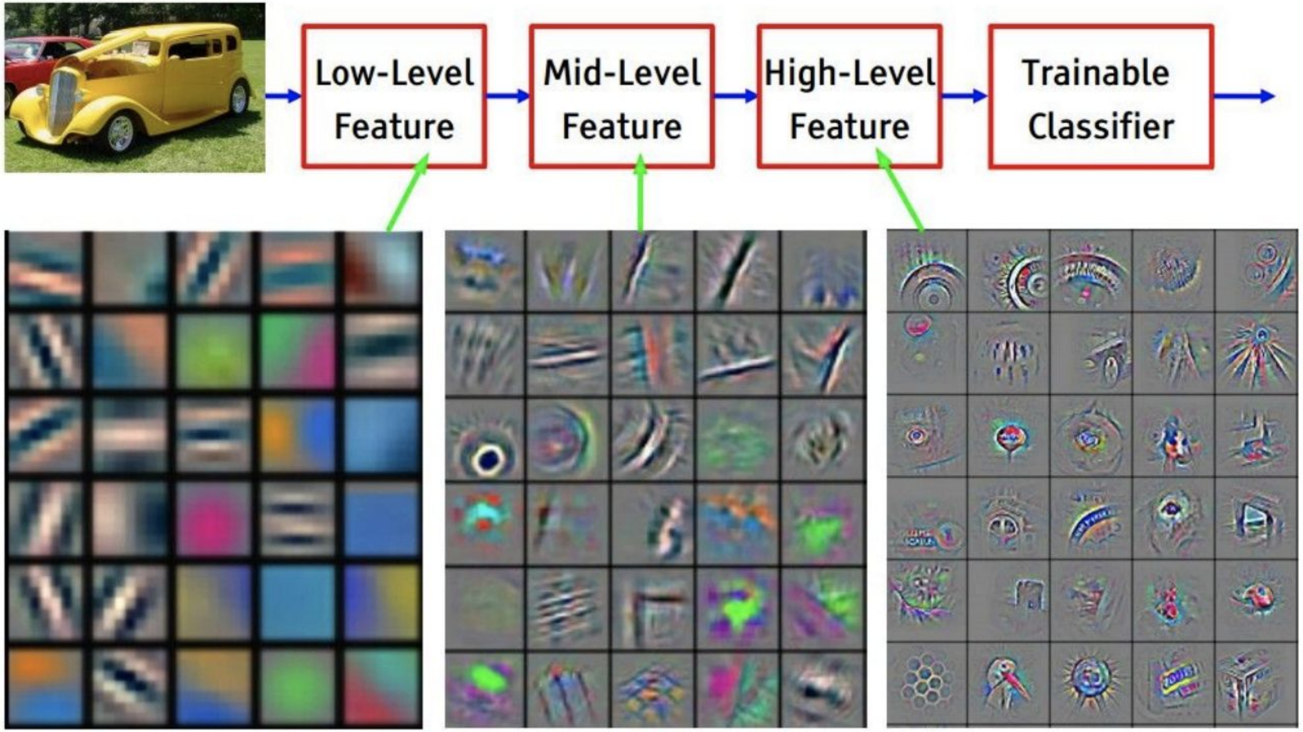
Convolutional layer

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



source

Convolutional layer

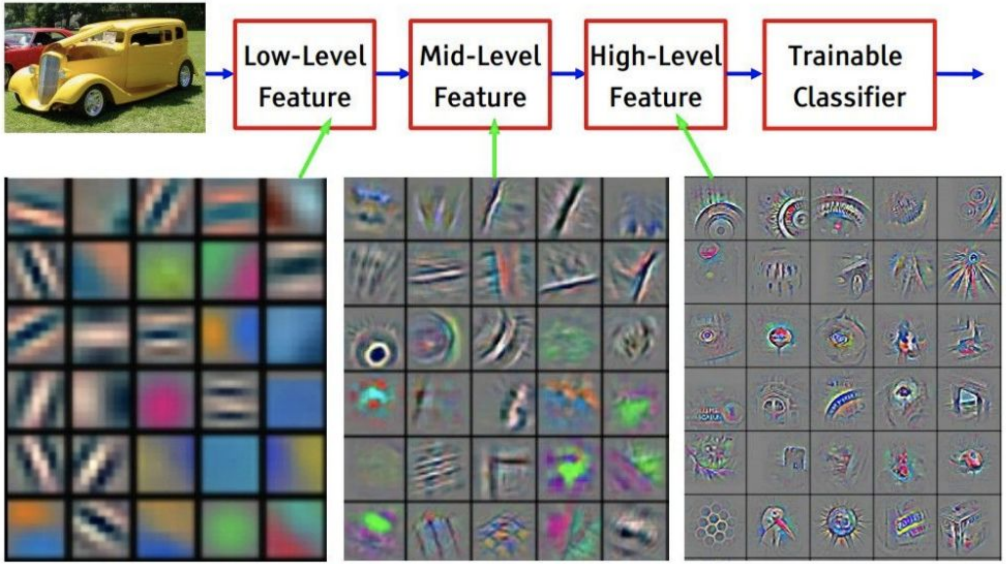


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

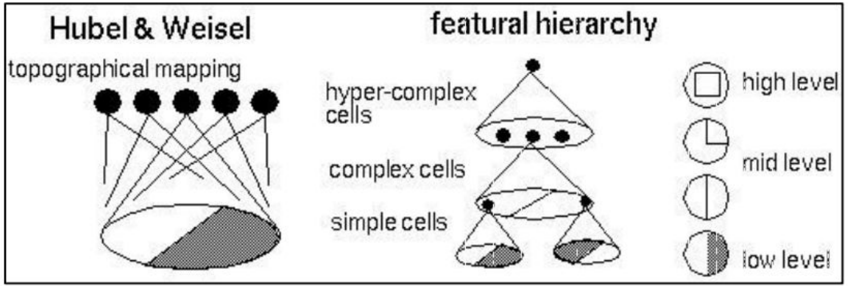
[From Yann LeCun slides]

source

Convolutional layer and visual cortex



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



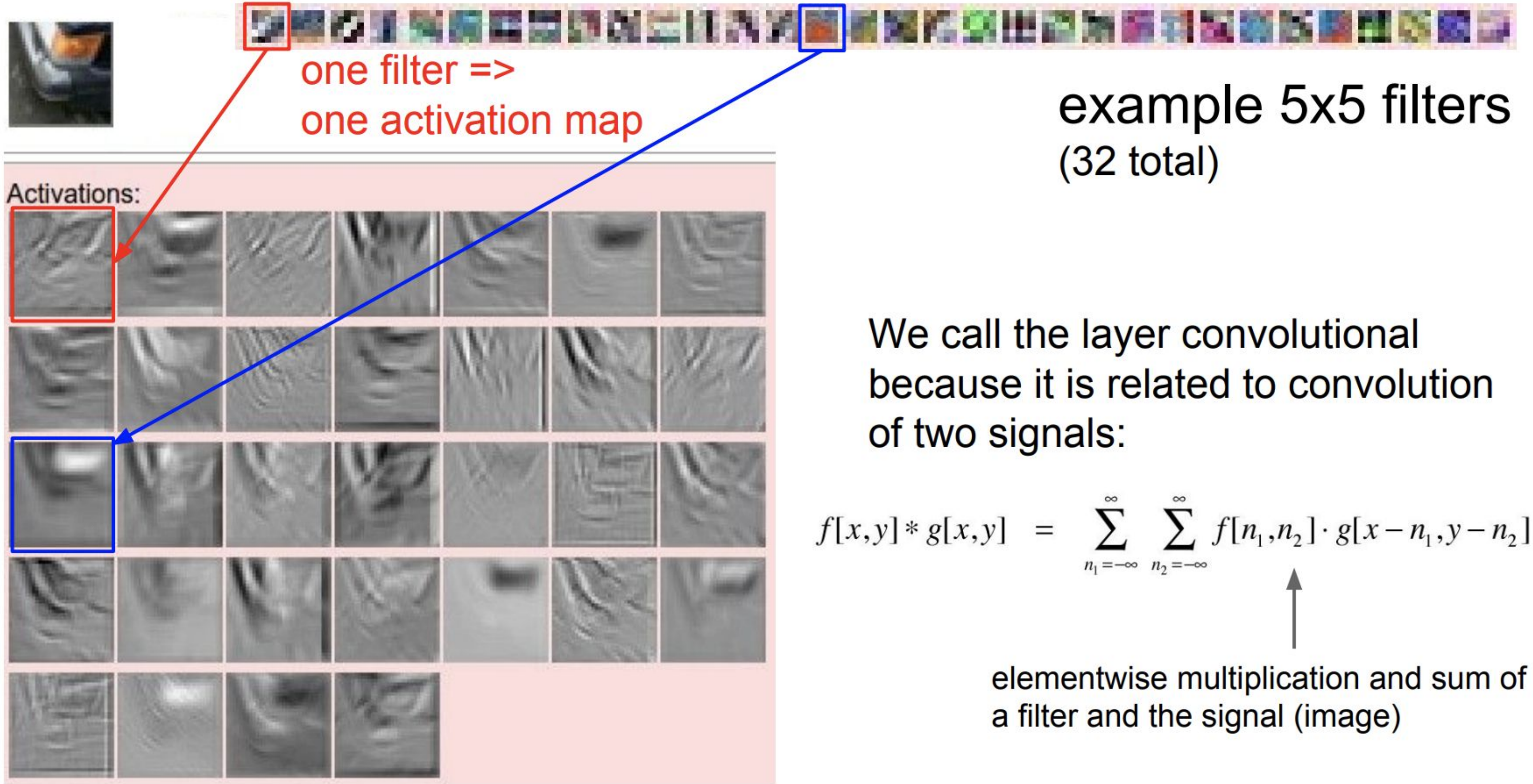
[From Yann LeCun slides]

Convolutional layer and visual cortex

source

CIFAR-10 online demo:

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



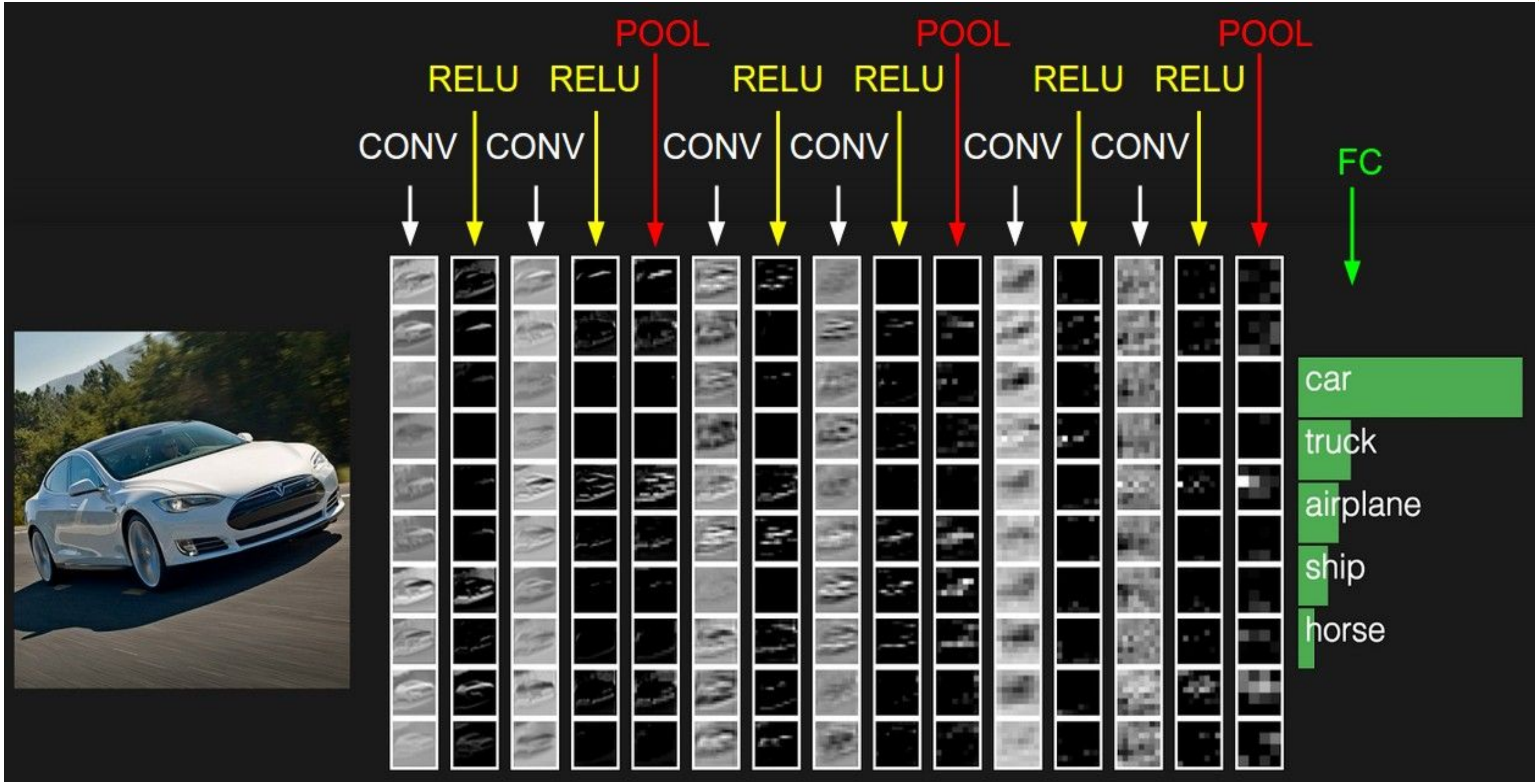
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y]*g[x,y] = \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} f[n_1,n_2]\cdot g[x-n_1,y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

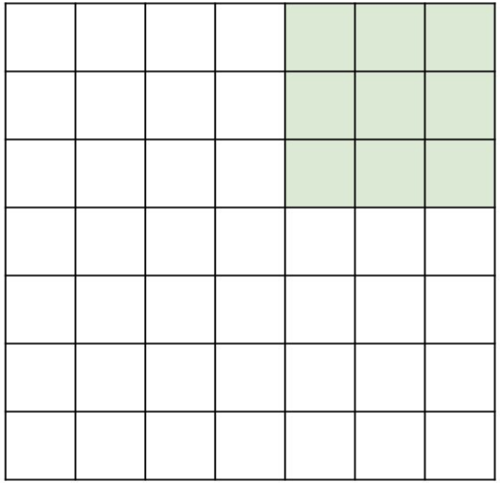
source


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

source

A closer look at spatial dimensions:

7



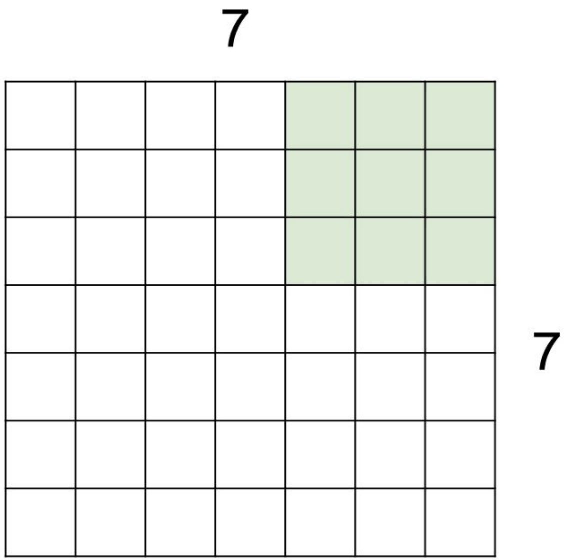
7

7x7 input (spatially)
assume 3x3 filter

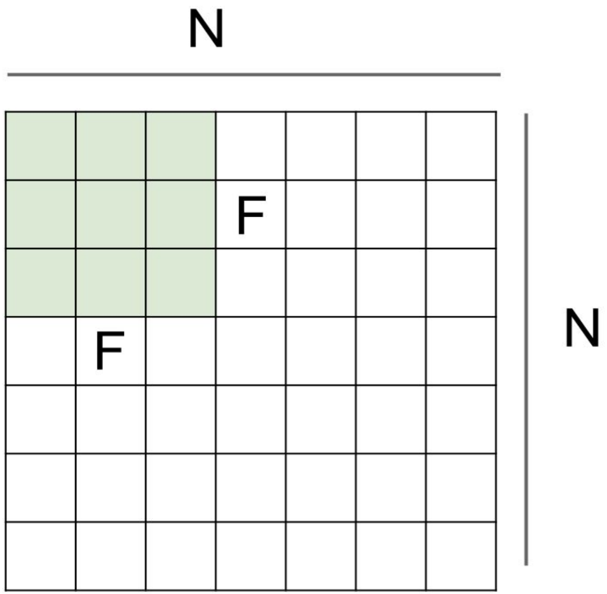
=> 5x5 output

source

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!



Output size:

(N - F) / stride + 1

e.g. N = 7, F = 3:

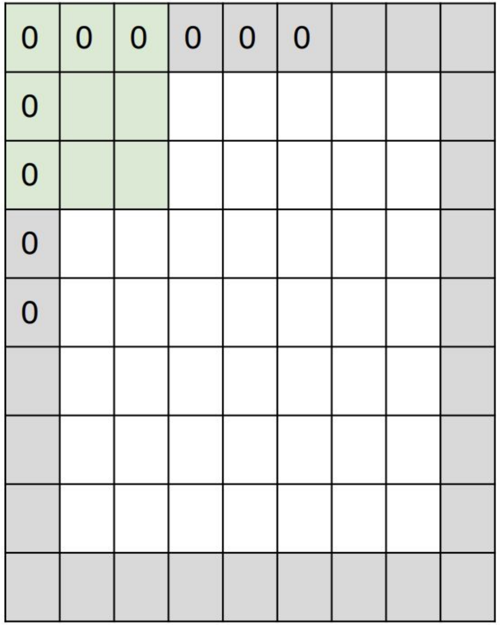
stride 1 => (7 - 3)/1 + 1 = 5

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

stride 3 => (7 - 3)/3 + 1 = 2.33 :\

source

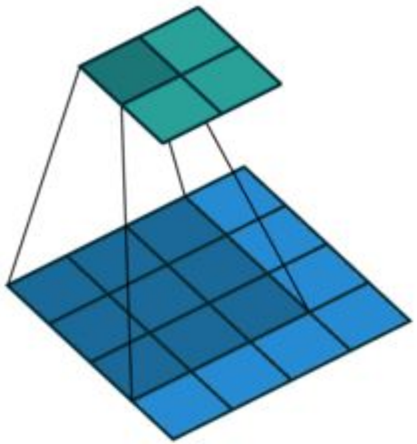
In practice: Common to zero pad the border



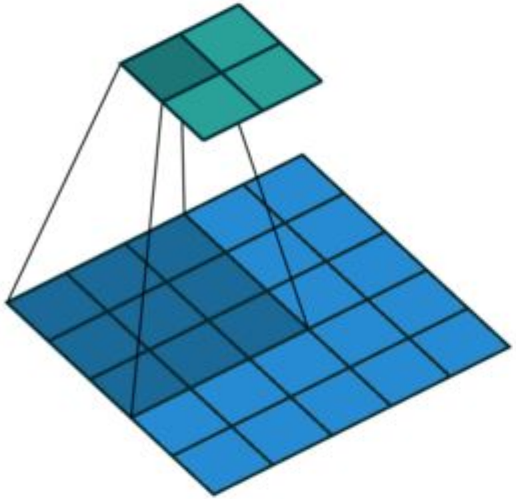
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Strides, padding in convolutional layer



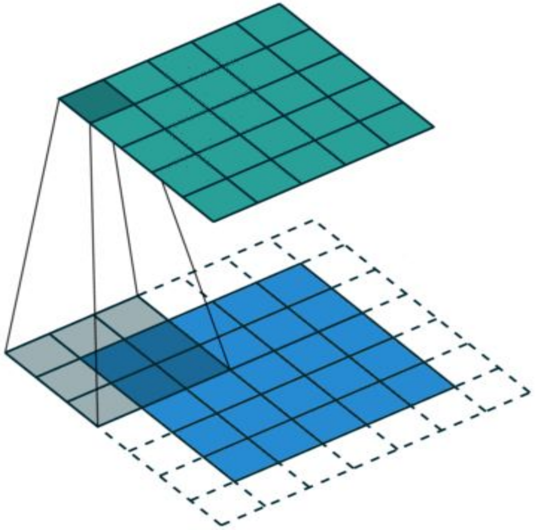
No padding, no strides



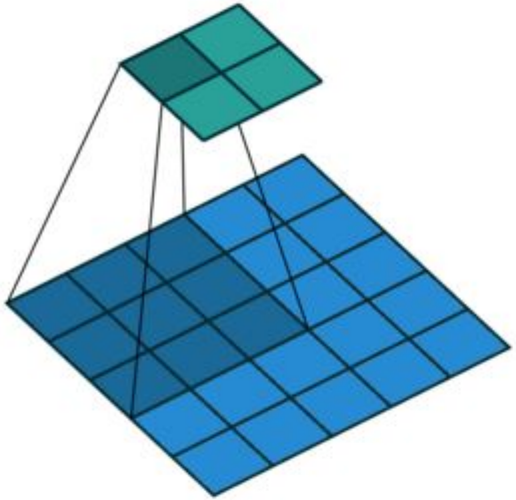
No padding, with strides

source

Strides, padding in convolutional layer

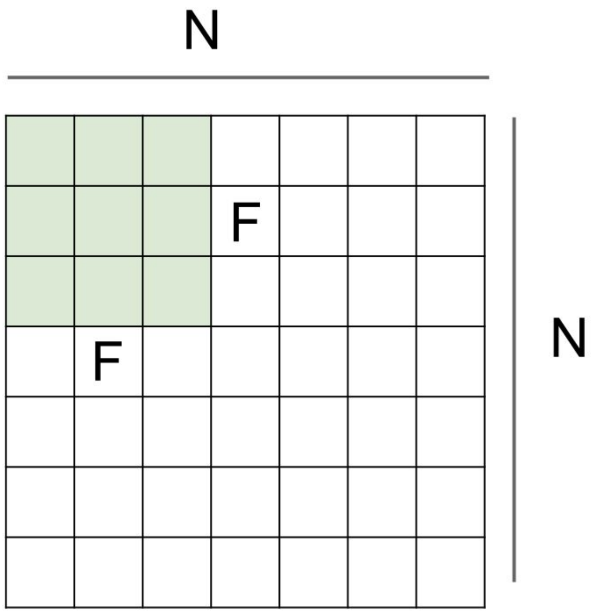


With padding, no strides



No padding, with strides

source

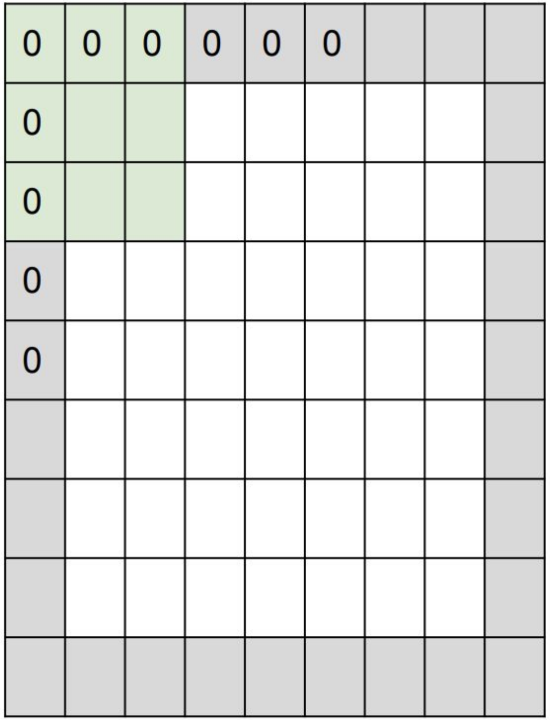


Output size:
(N - F) / stride + 1

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 :\

source

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

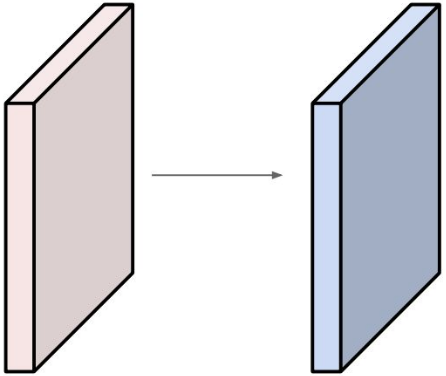
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

source

Examples time:

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

Output volume size: ?

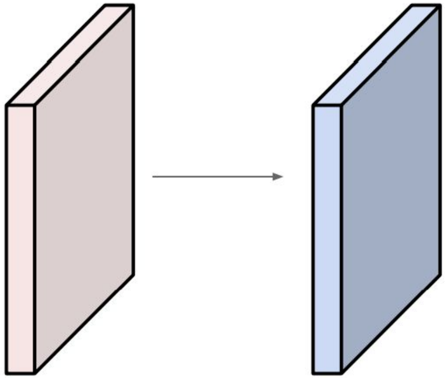


source

Examples time:

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

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

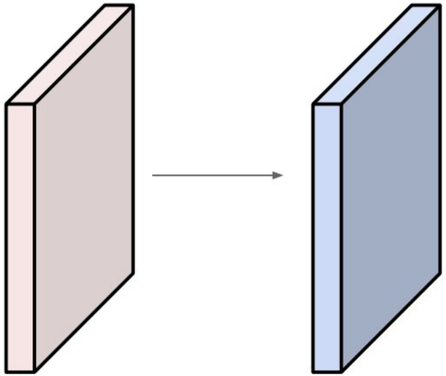


source

Examples time:

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

Number of parameters in this layer?

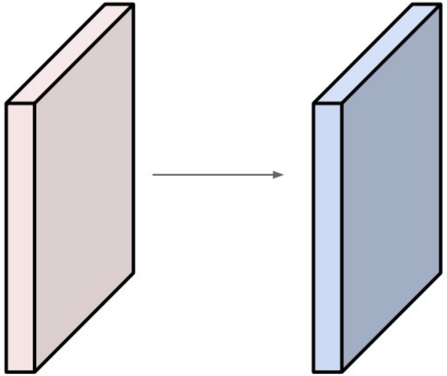


source

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



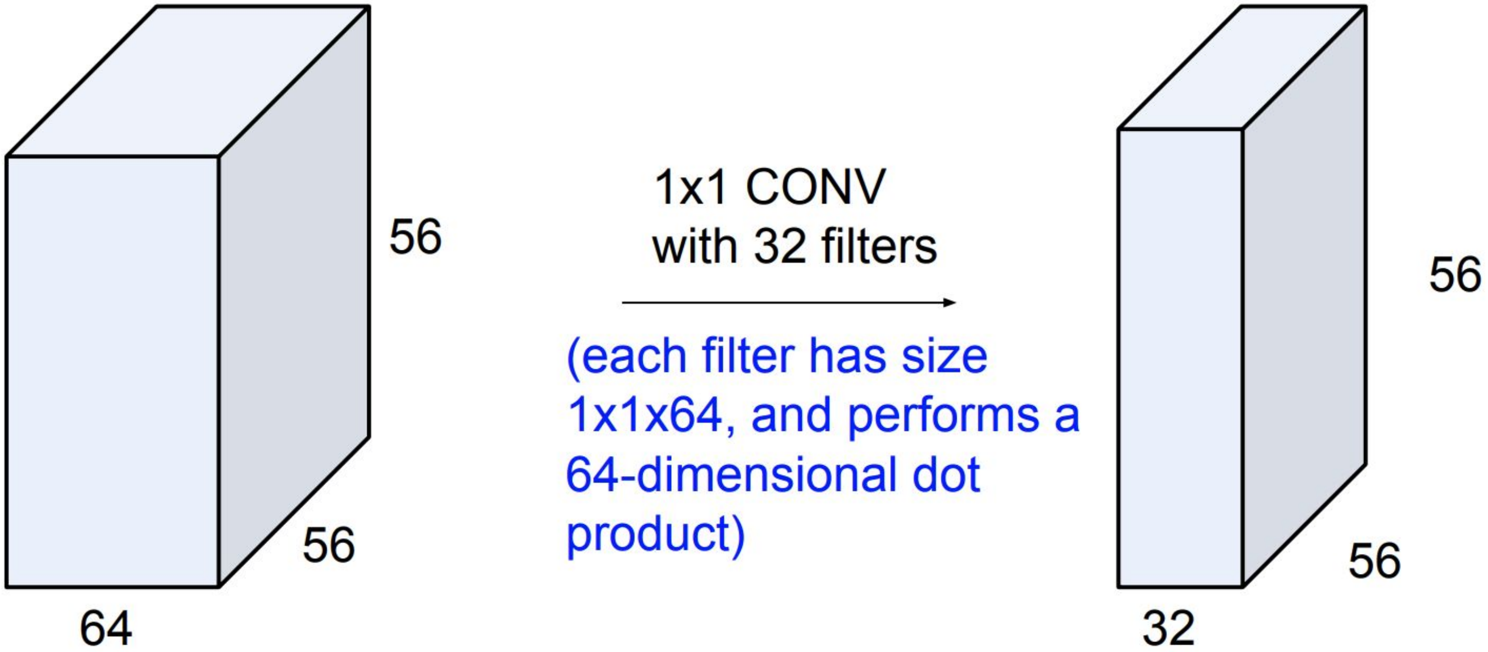
Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

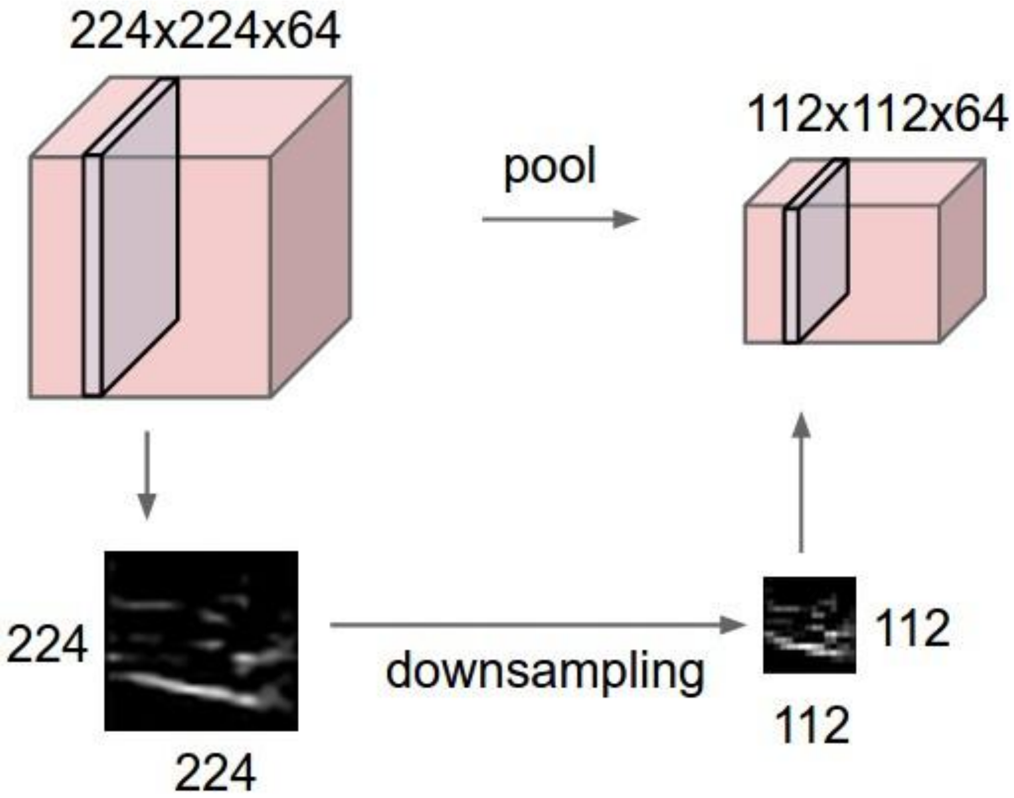
source

1x1 convolutions



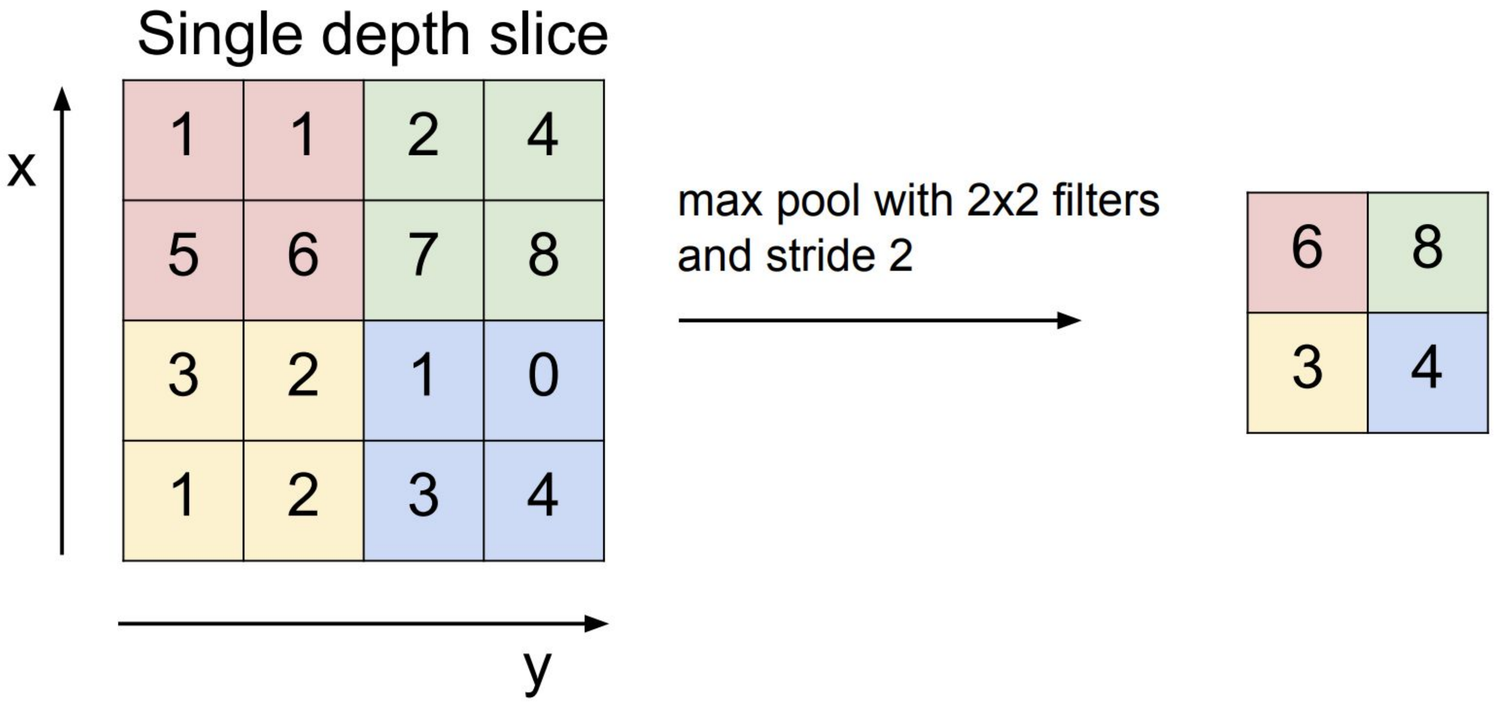
source

Pooling layer



- Makes the representations smaller and more manageable
- Operates over each activation map independently

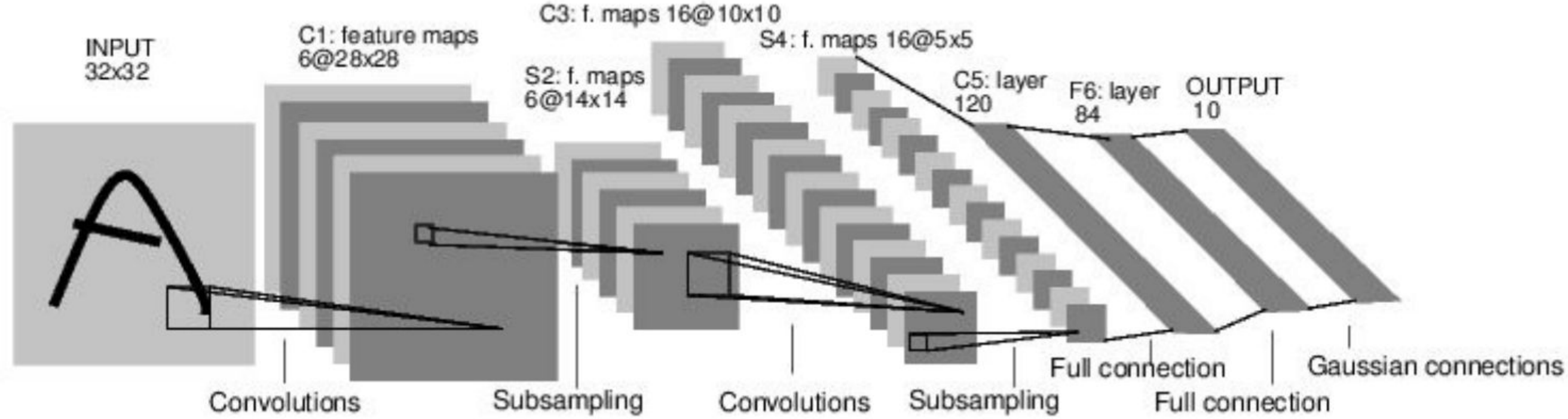
source


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

source

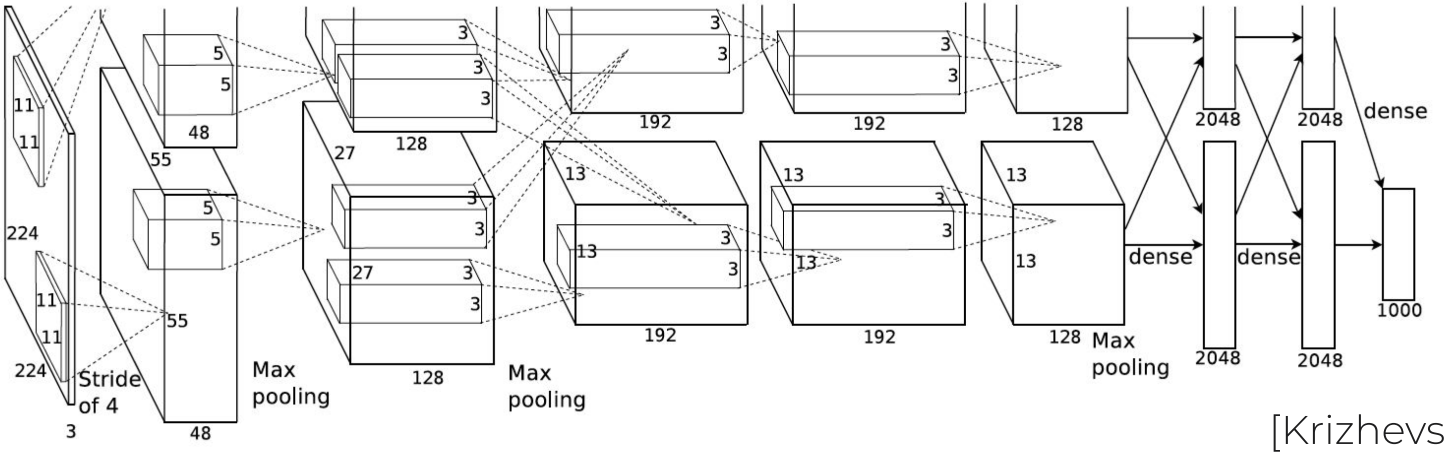
Architectures overview

LeNet-5



[LeNet-5, LeCun 1998]

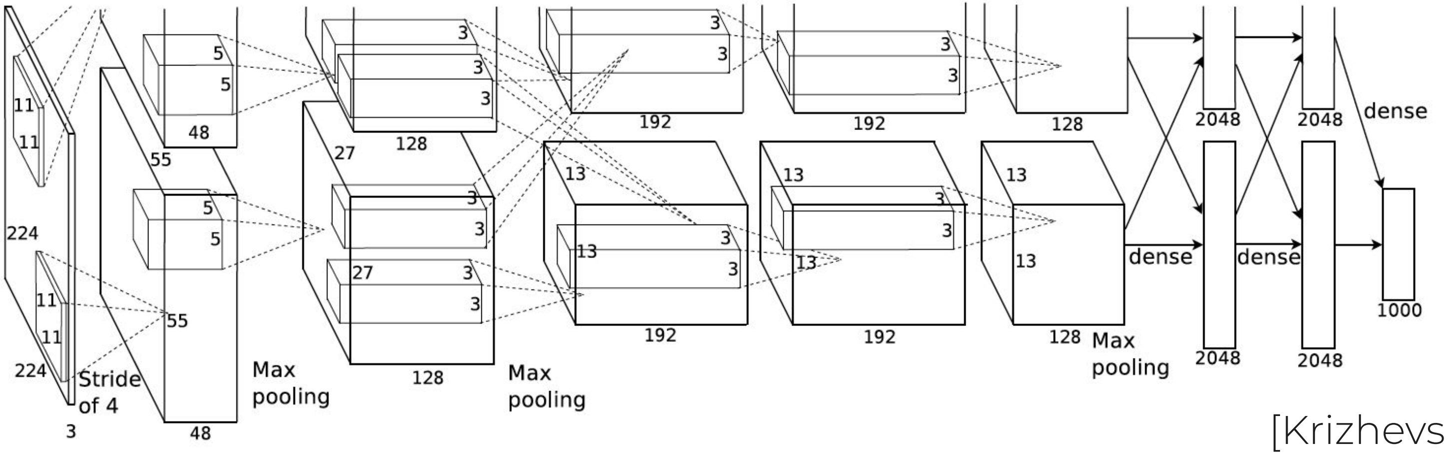
AlexNet



[Krizhevsky et al. 2012]

source

AlexNet

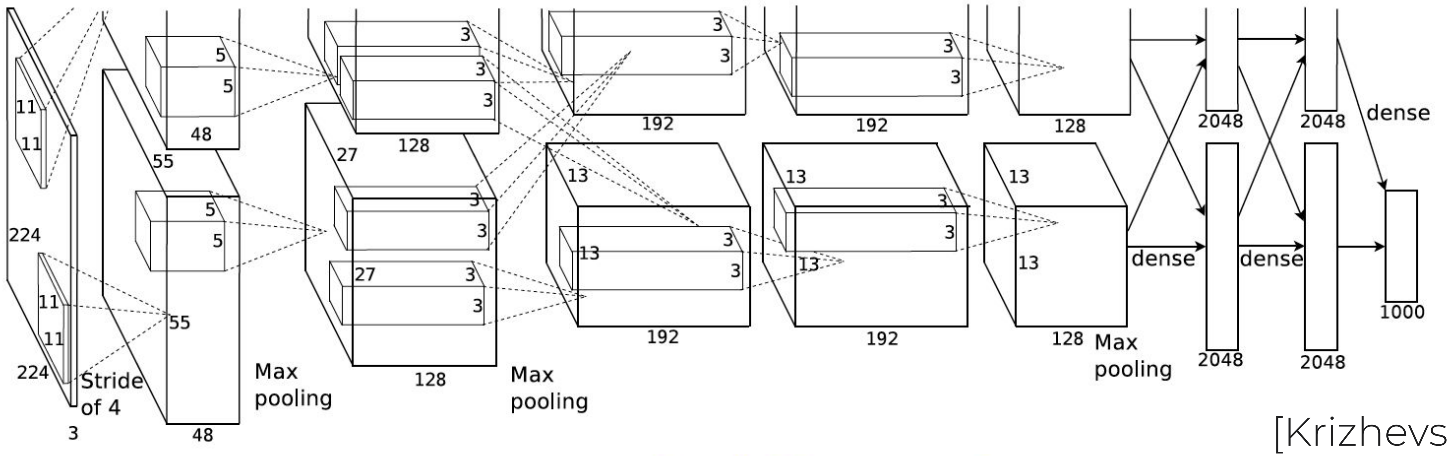


[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[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)

source

AlexNet



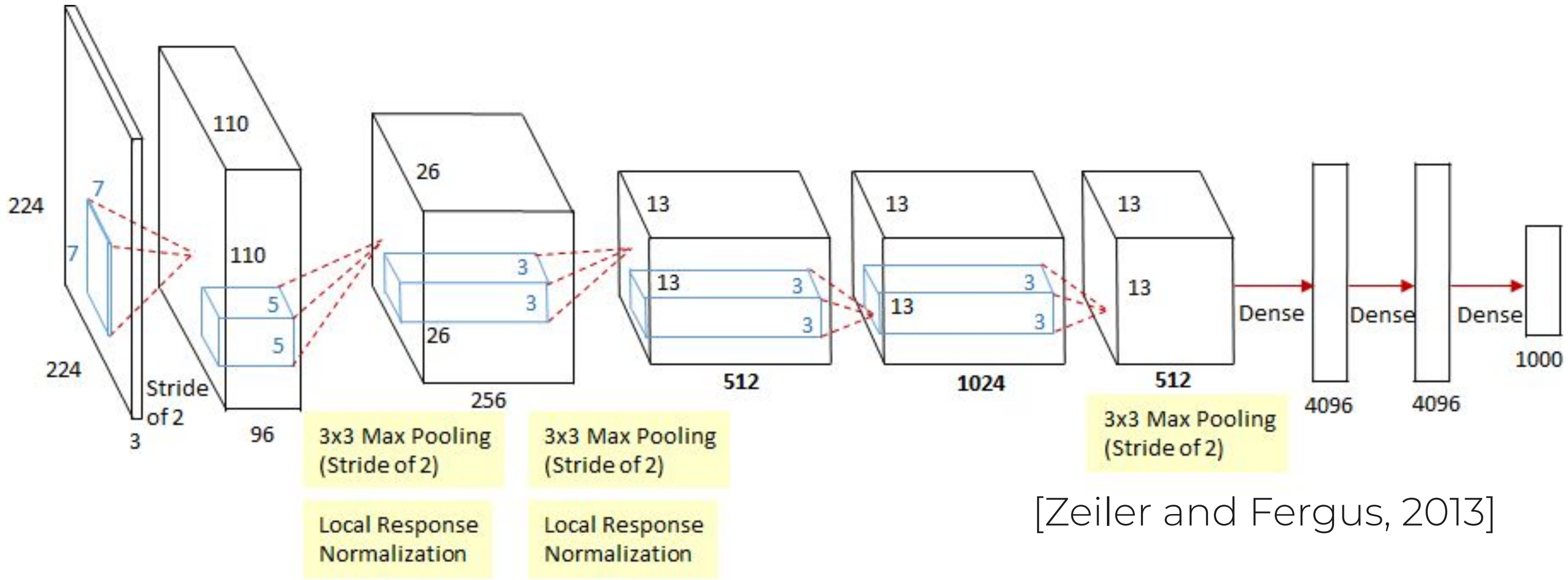
[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[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)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- 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
- 7 CNN ensemble: 18.2% -> 15.4%

source

ZFNet



[Zeiler and Fergus, 2013]

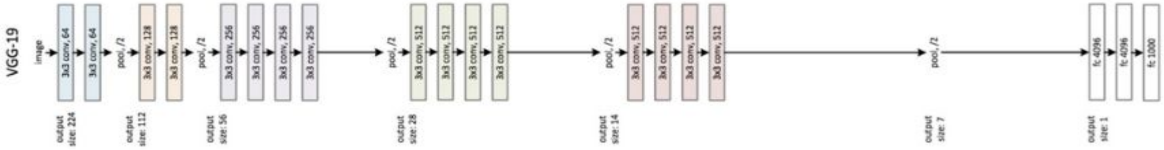
AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

source

VGGNet

7.3% top 5 error



ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

source

VGGNet

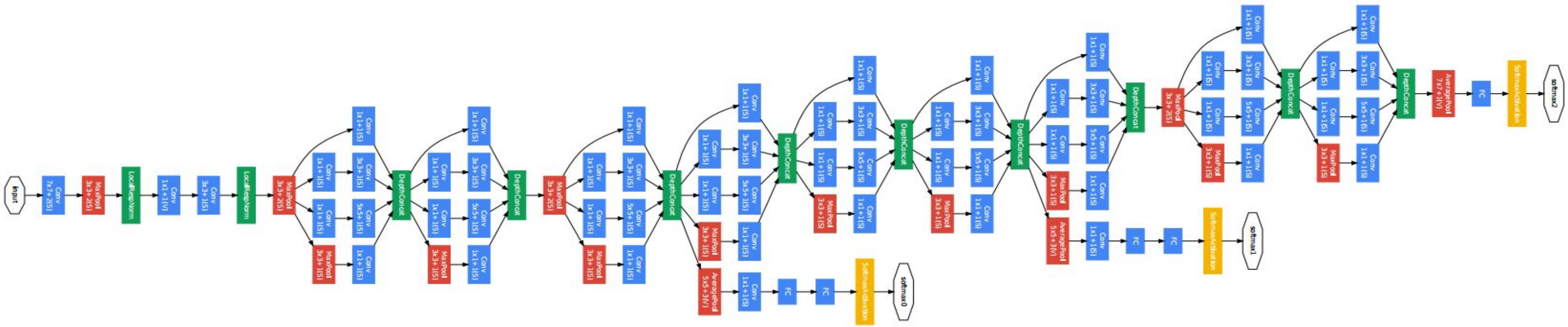
INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

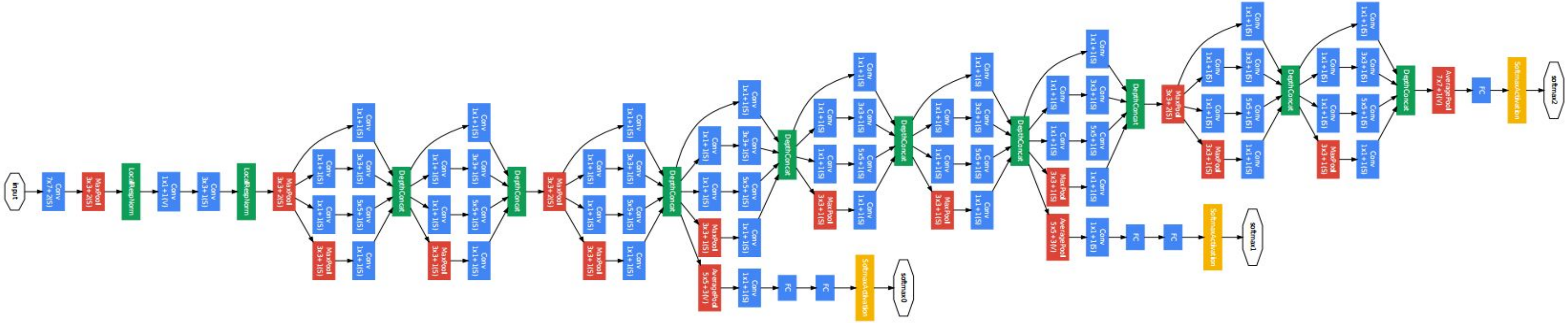
source

GoogLeNet



[Szegedy et al., 2014]

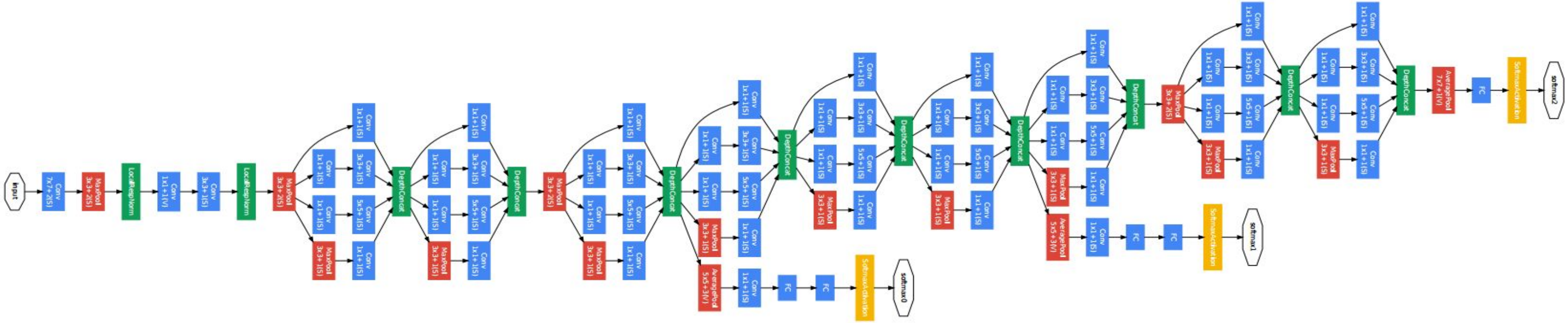
GoogLeNet



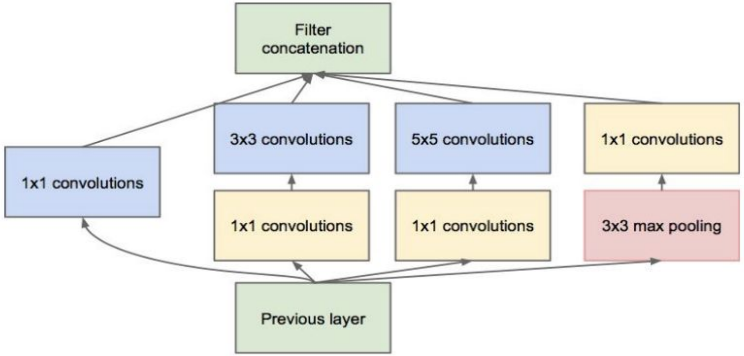
[Szegedy et al., 2014]

Inception module

GoogLeNet

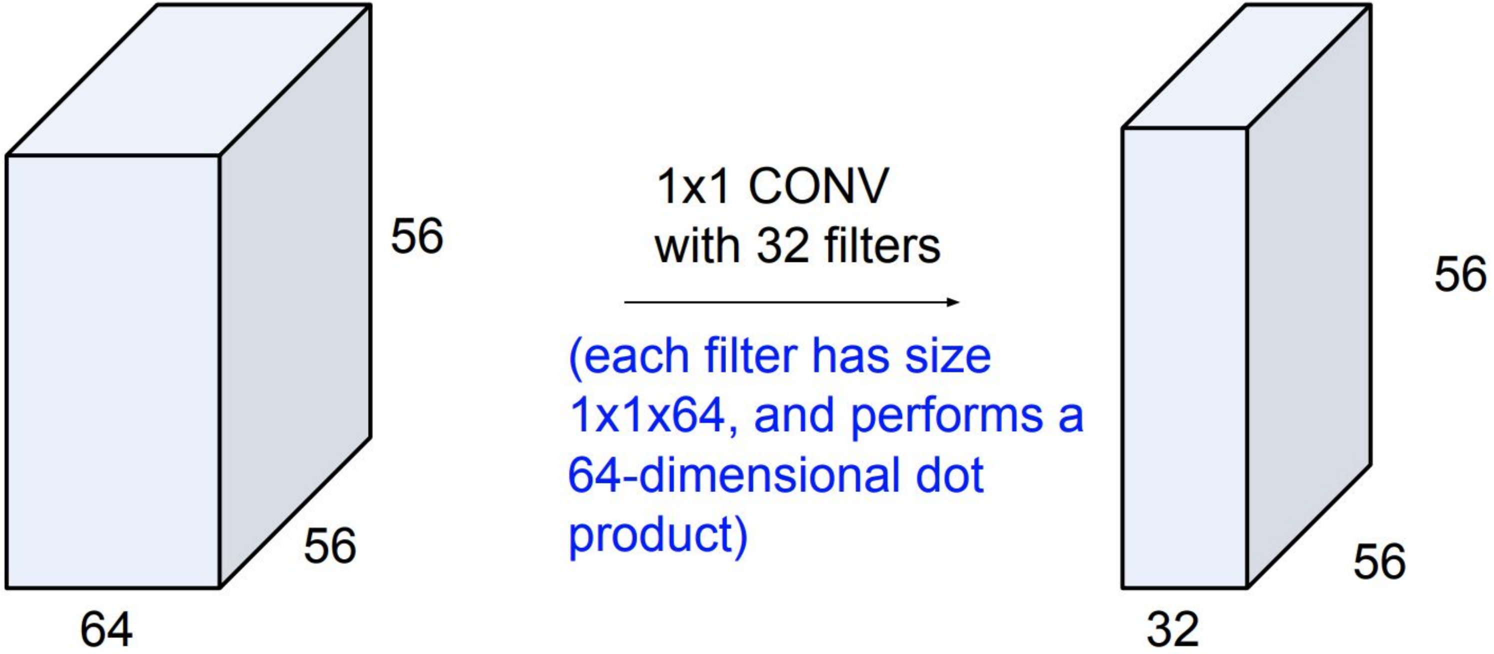


[Szegedy et al., 2014]



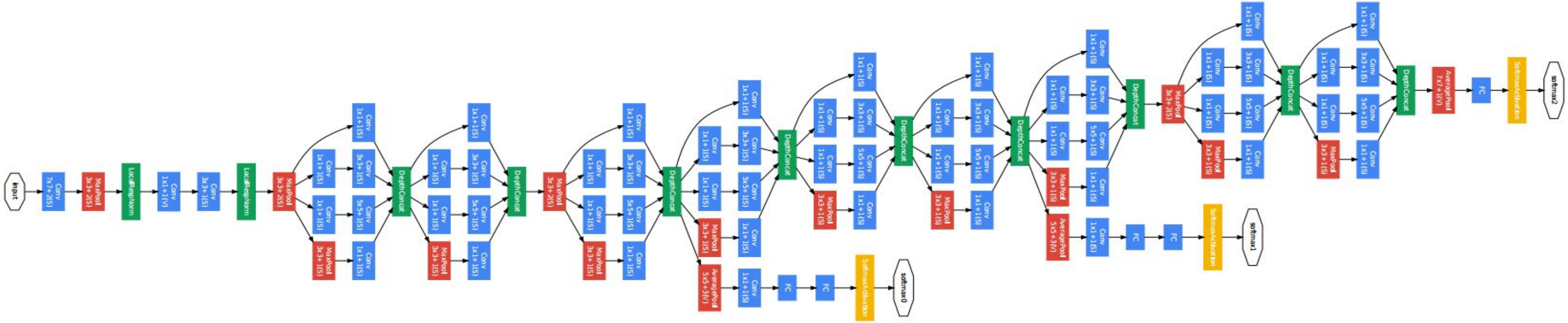
Inception module

Once again: 1x1 convolutions

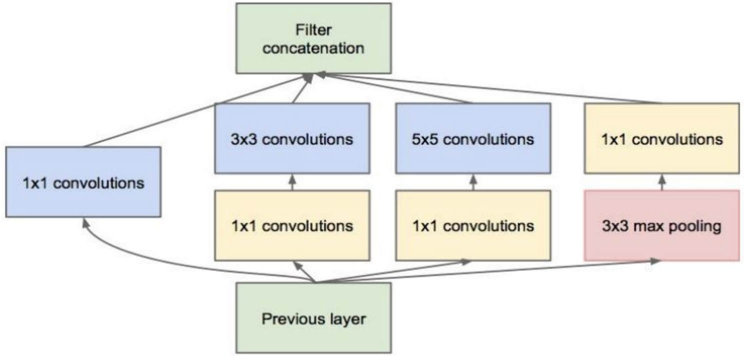


source

GoogLeNet

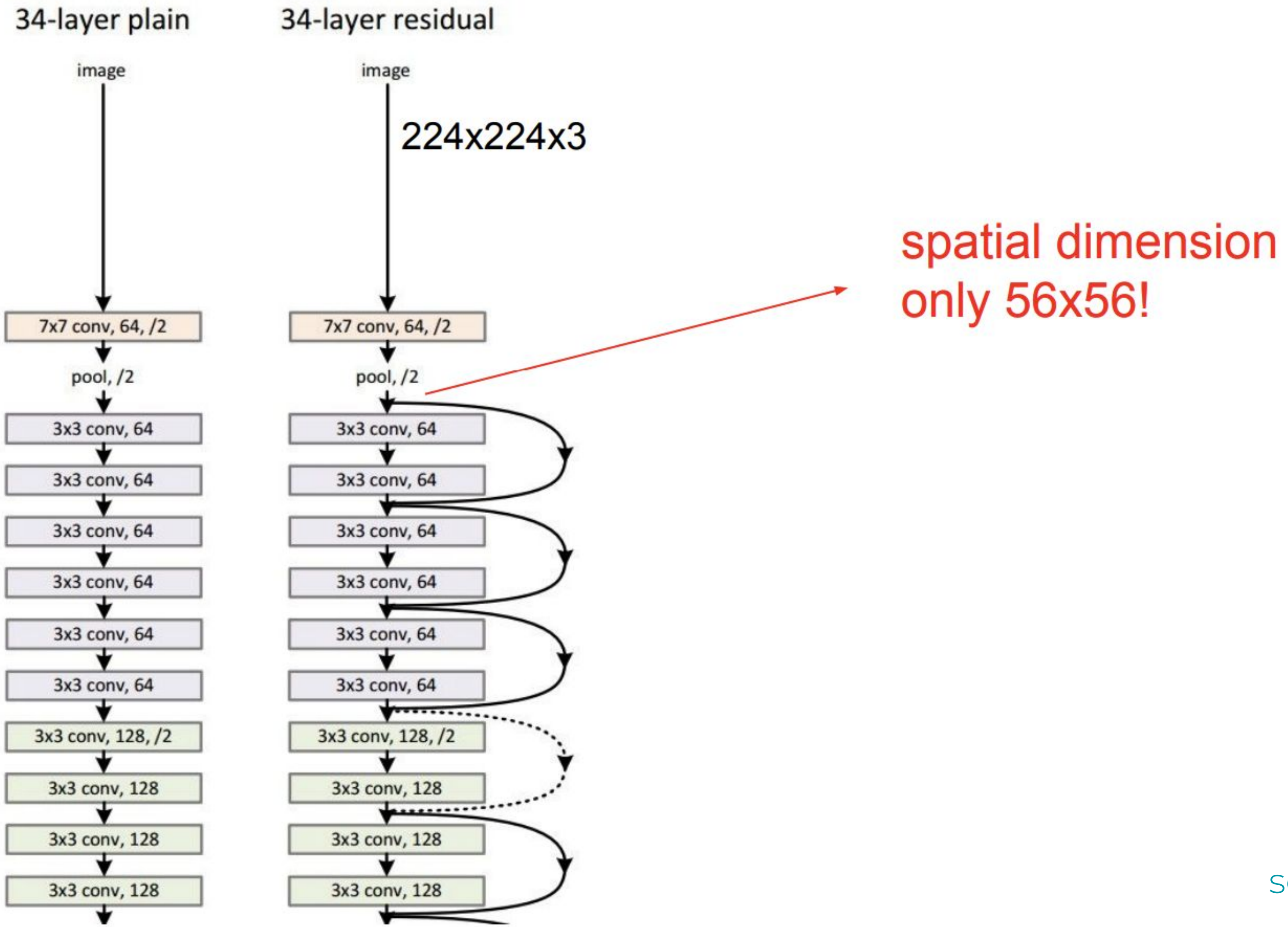


[Szegedy et al., 2014]



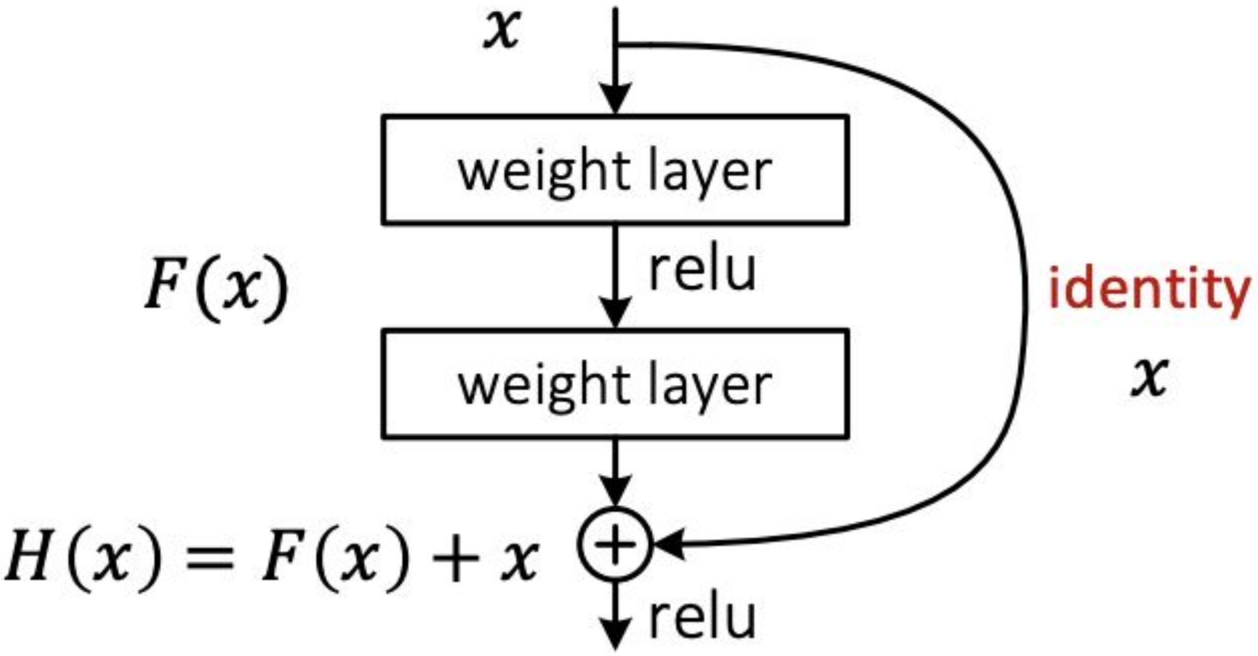
Inception module

ILSVRC 2014 winner (6.7% top 5 error)


34-layer plain
34-layer residual
image
image
224x224x3
7x7 conv, 64, /2
7x7 conv, 64, /2
pool, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
spatial dimension
only 56x56!

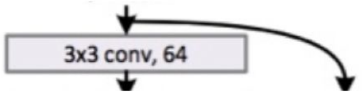
source

Residual Block

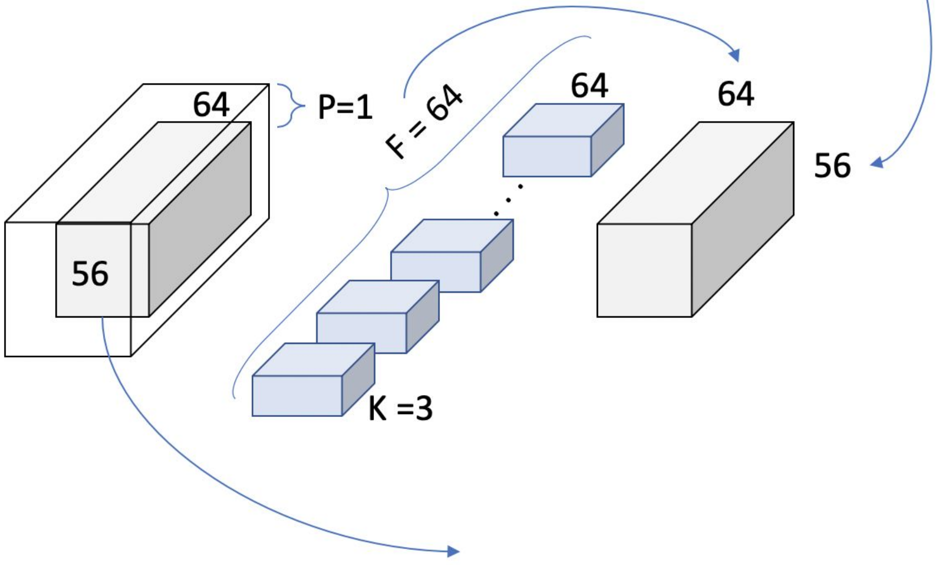


source

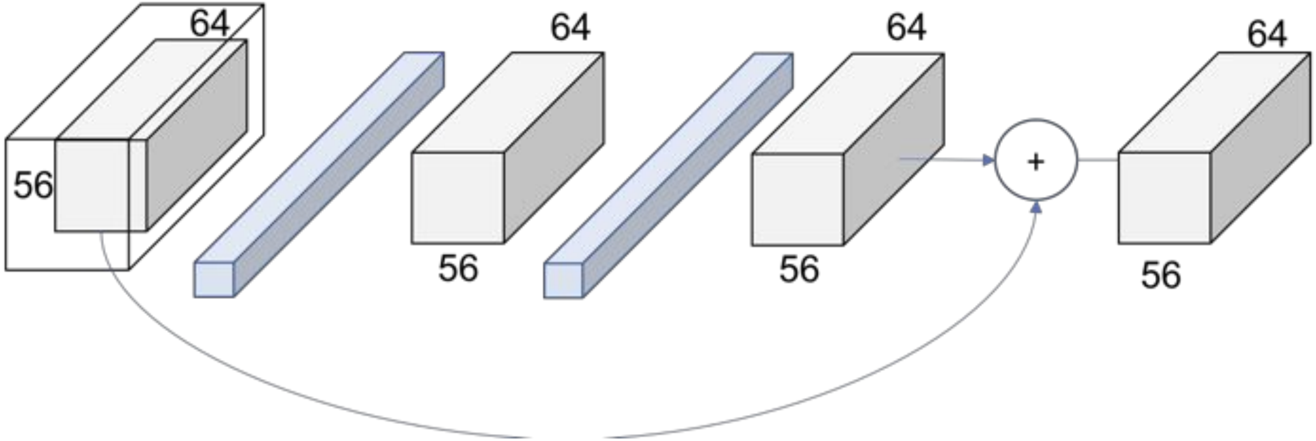
Residual Block

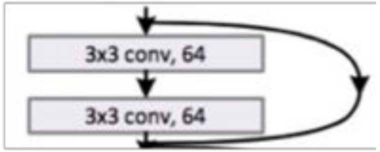


$$O = \frac{I + 2P\ - K}{S} + 1 = \frac{56\ - 3 + 2}{1} + 1 = 56$$



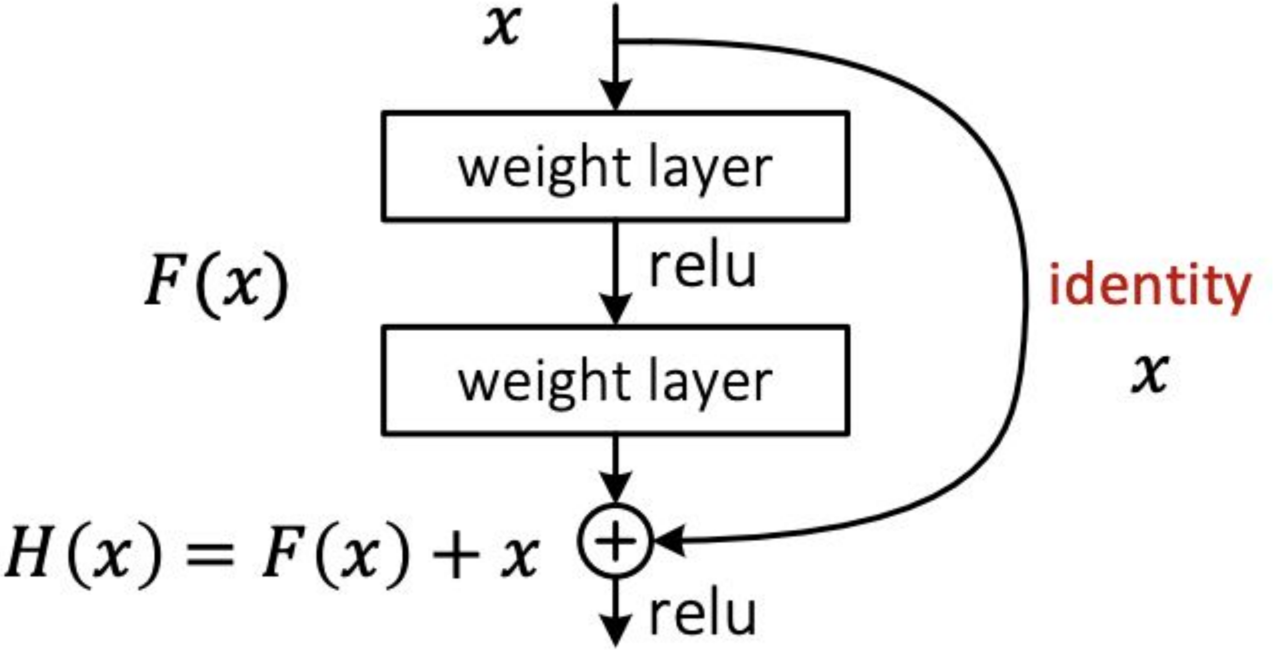
Residual Block





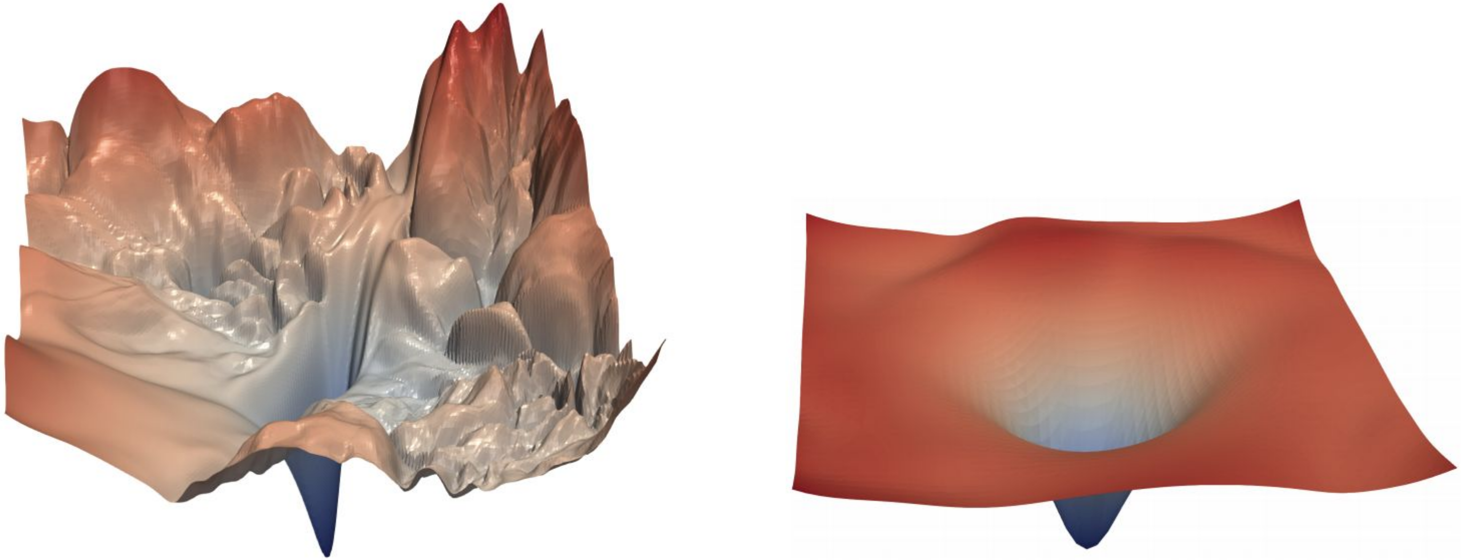
source

Residual Block



source

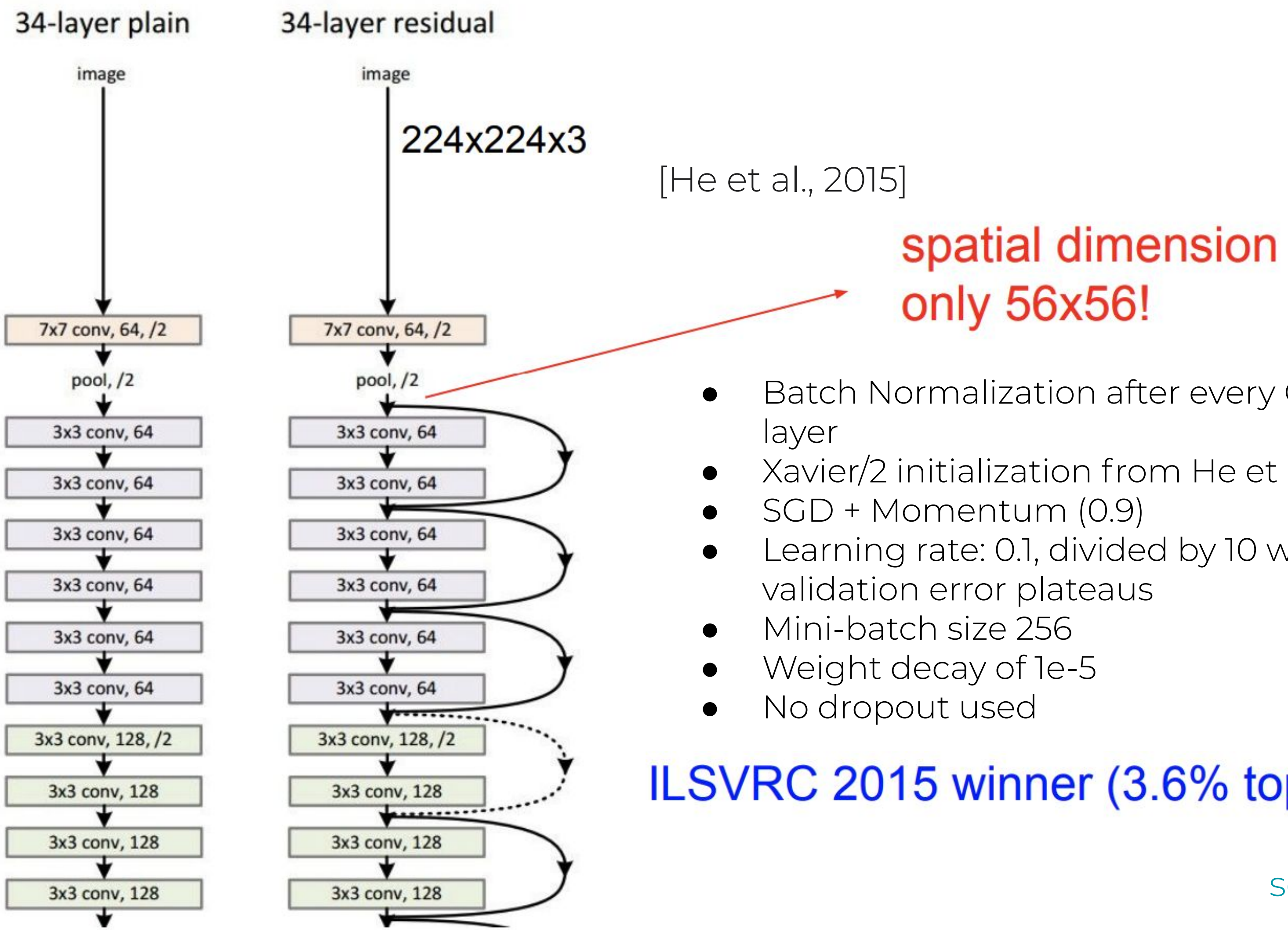
Residual Block



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

source NeurIPS 2018



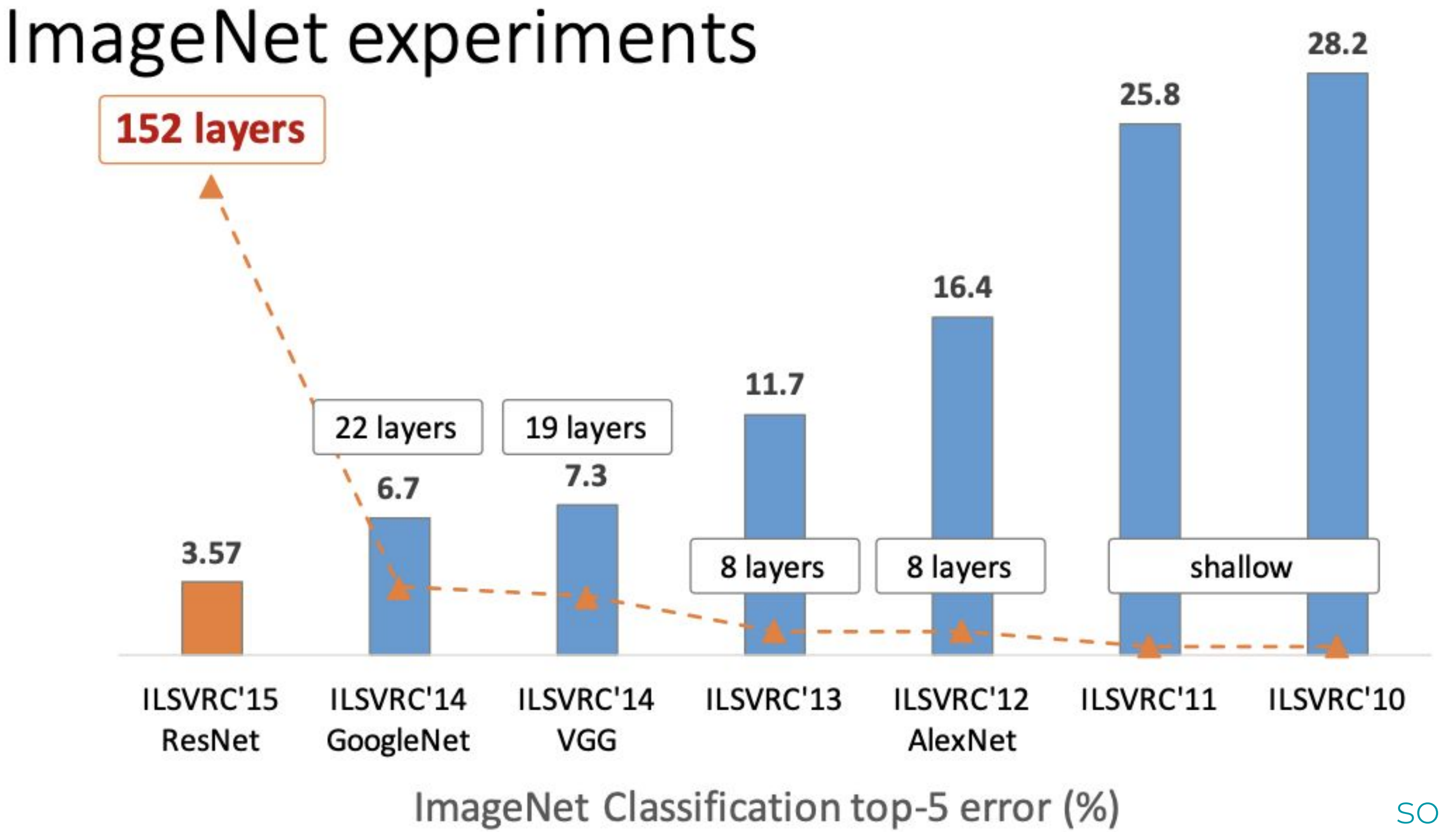
[He et al., 2015]

- 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

ILSVRC 2015 winner (3.6% top 5 error)

source

ResNet



source

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- Potential solution: direct (or skip-) connections (just like in ResNet)

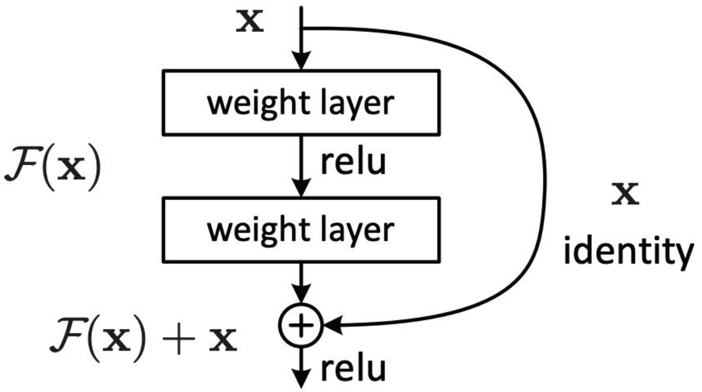


Figure 2. Residual learning: a building block.

image source

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural network architectures.

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- Potential solution: dense connections (just like in DenseNet)

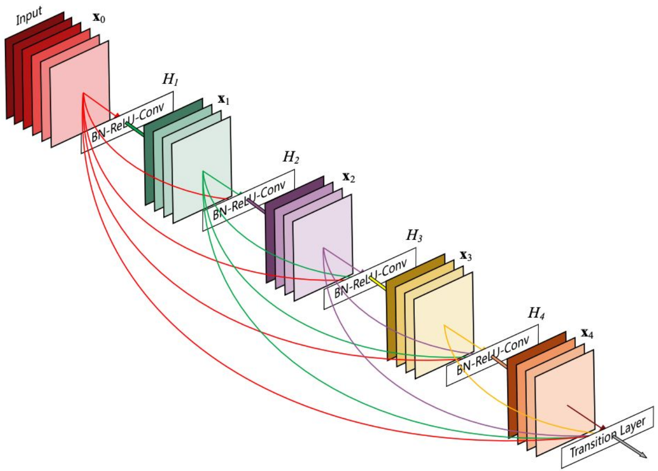
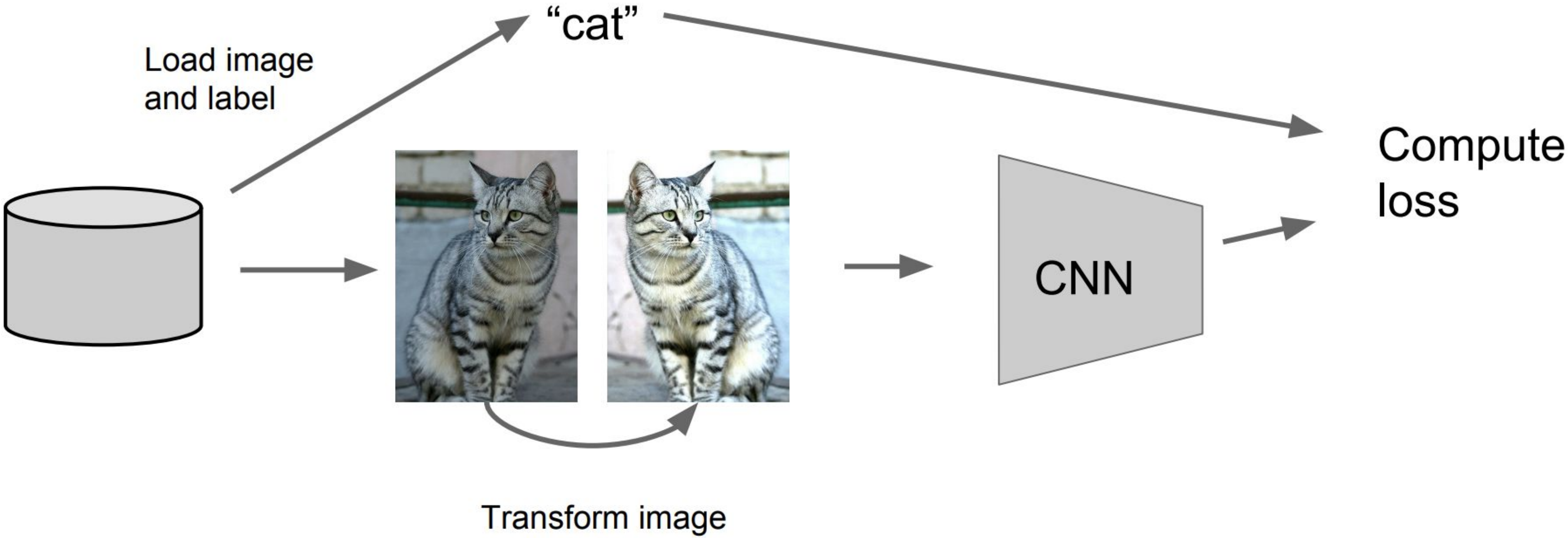


image source

Recap: data augmentation



source

Summary

- ConvNets stack convolutional, pooling and dense layers
- Trend towards smaller filters and deeper architectures
- 1x1 convolutions are meaningful
- Humanity is already beaten on ImageNet.

Revise

1. Convolutional layer structure.
2. Pooling layers.
3. Top architectures overview.

Thanks for attention!

Questions?

girafe ai