

Machine Learning Course  
basic track

## Lecture 12: Convolutional neural networks

**Radoslav Neychev**

25.11.2019, MIPT  
Moscow, Russia

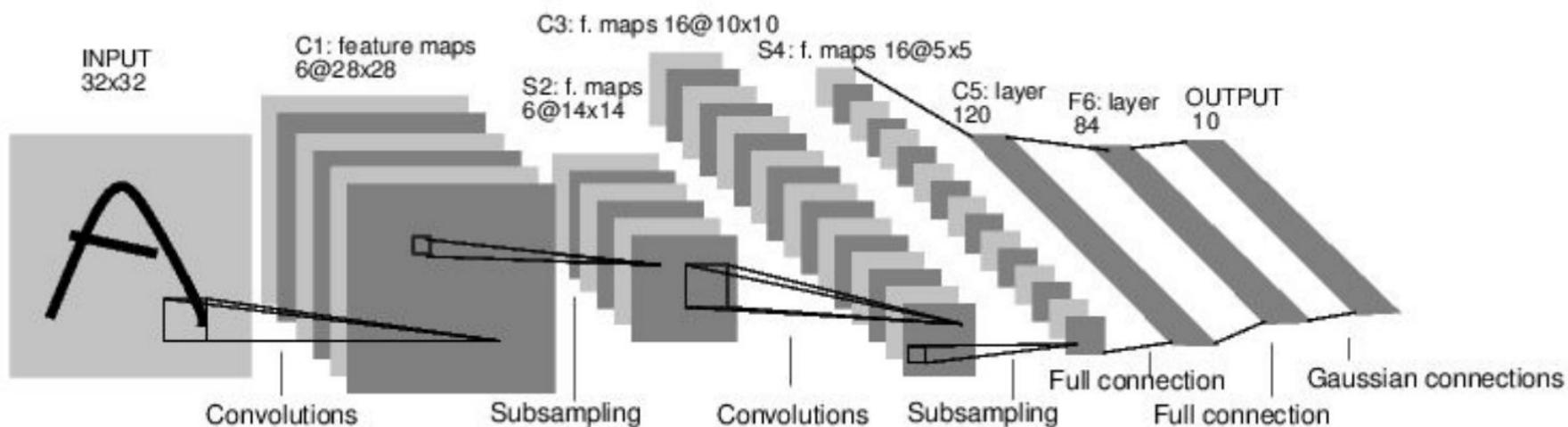
# Outline

1. Previous lecture recap.
2. Convolutional layer structure.
3. Pooling layers.
4. Top architectures overview.
5. Q & A.

This lecture is deeply based on [Stanford CS231n Lecture 7](#) materials by Fei-Fei Li & Andrej Karpathy & Justin Johnson

- Layers
  - a. Dense layer (*done*)
  - b. Convolutional layer (*this lecture*)
  - c. Pooling layer (*this lecture*)
  - d. Dropout layer (*done*)
  - e. Batchnorm layer (batch normalization) (*done*)
  - f. Embeddings (aka word2vec, GloVe) (*done*)
  - g. Recurrent layers (*done*)

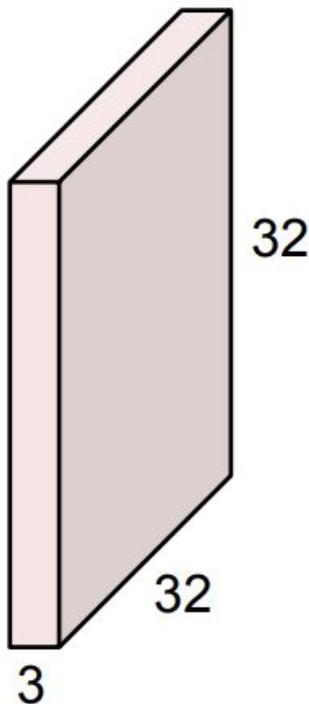
- Layers
  - a. Dense layer (*done*)
  - b. Convolutional layer (*this lecture*)
  - c. Pooling layer (*this lecture*)
  - d. Dropout layer (*done*)
  - e. Batchnorm layer (batch normalization) (*done*)
  - f. Embeddings (aka word2vec, GloVe) (*done*)
  - g. Recurrent layers (*done*)



[LeNet-5, LeCun 1998]

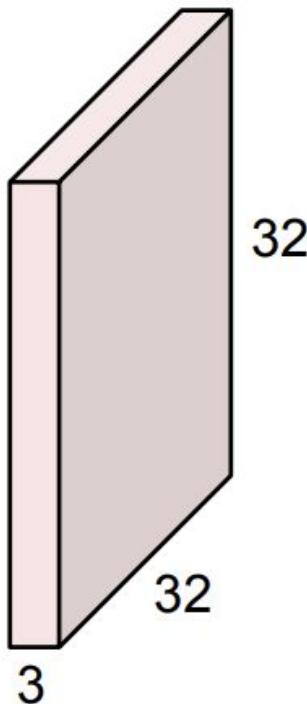
# Convolutional layer

32x32x3 image

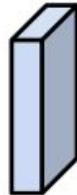


# Convolutional layer

32x32x3 image



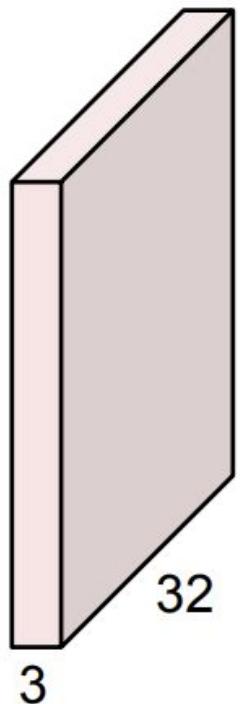
5x5x3 filter



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

# Convolutional layer

32x32x3 image



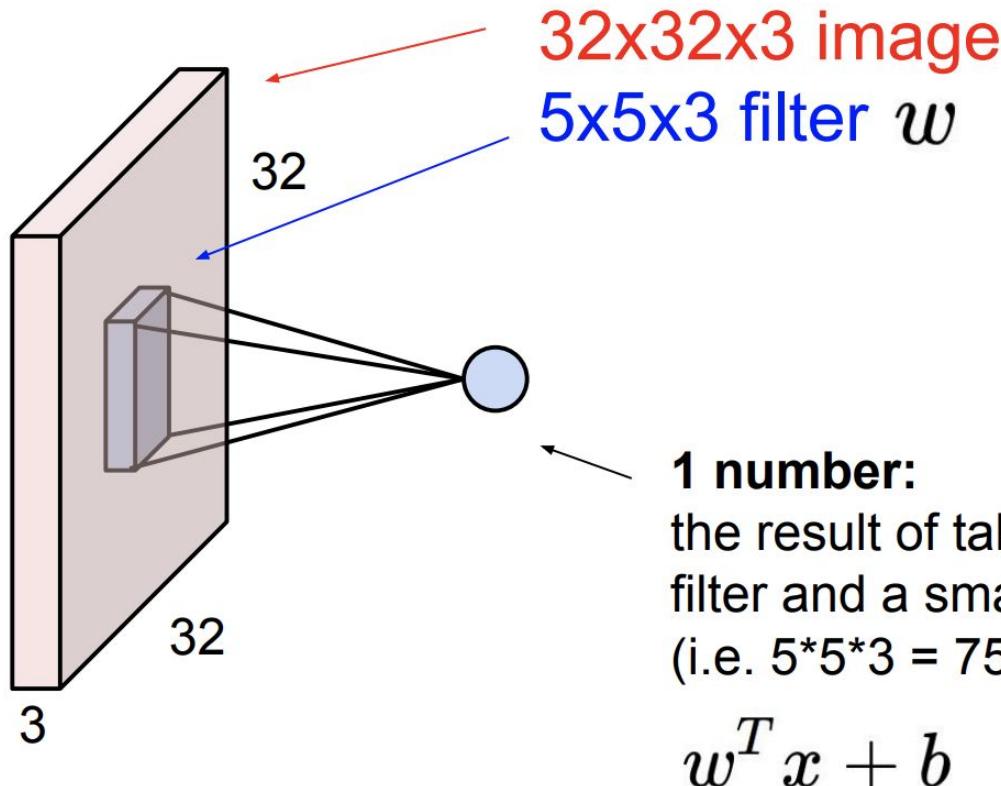
5x5x3 filter



Filters extend the *depth* of the original image

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

# Convolutional layer

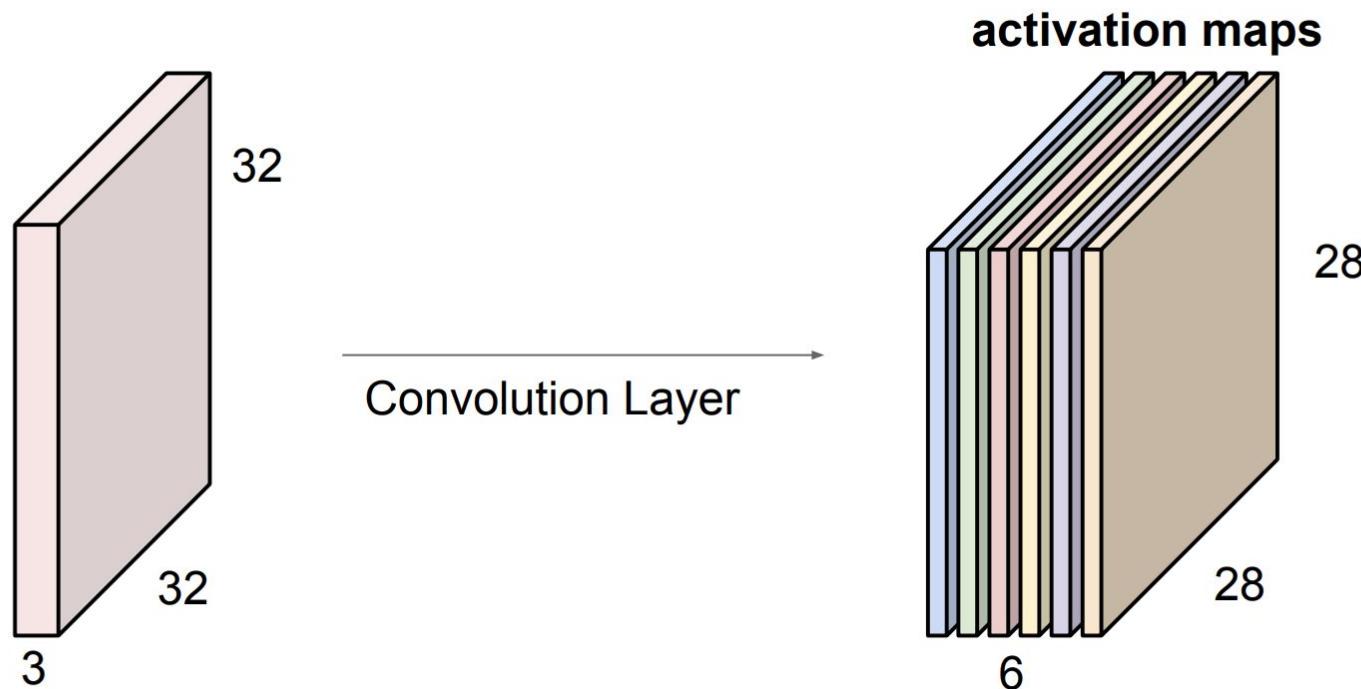


# Convolutional layer



# Convolutional layer

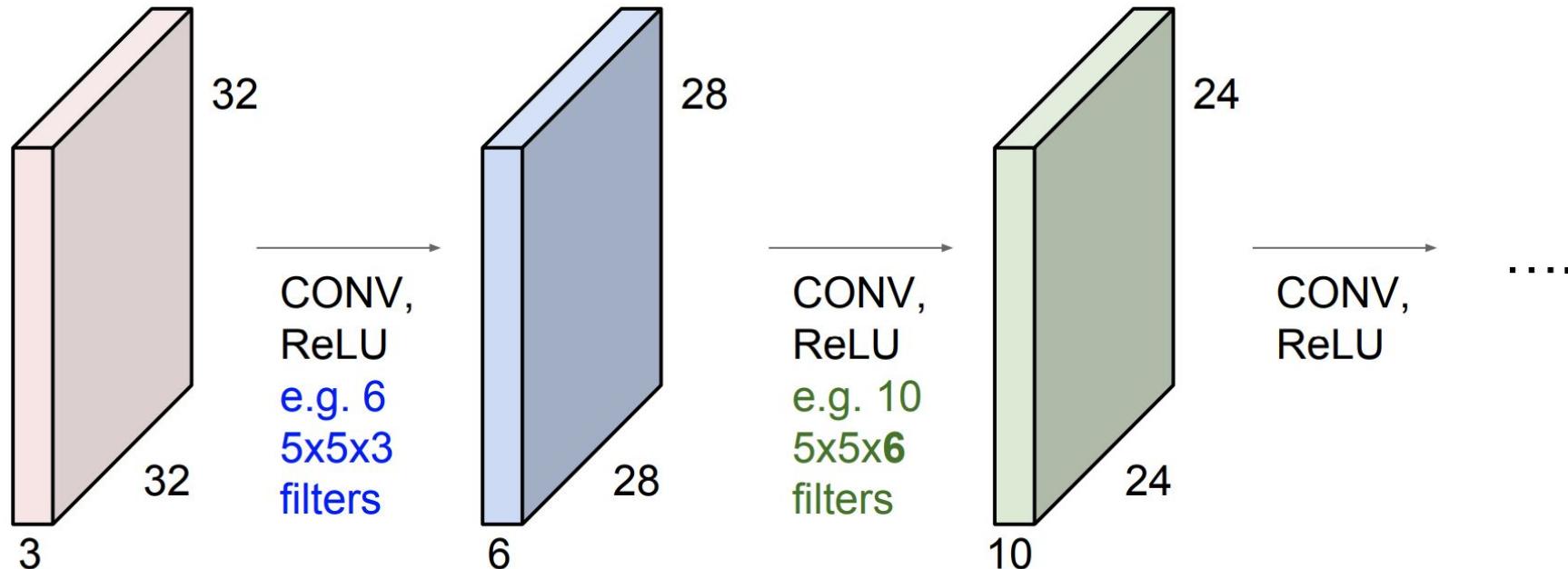
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



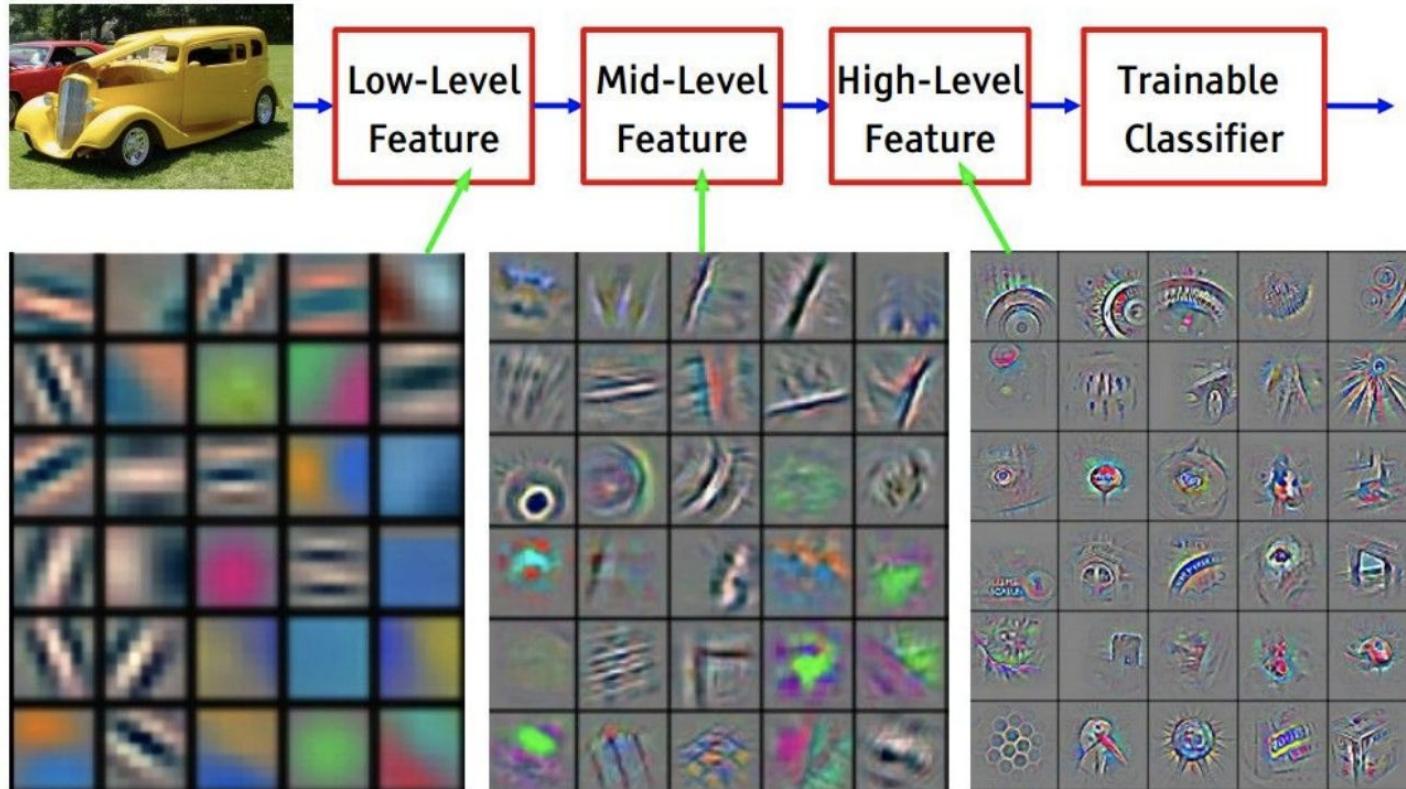
We stack these up to get a “new image” of size 28x28x6!

# Convolutional layer

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

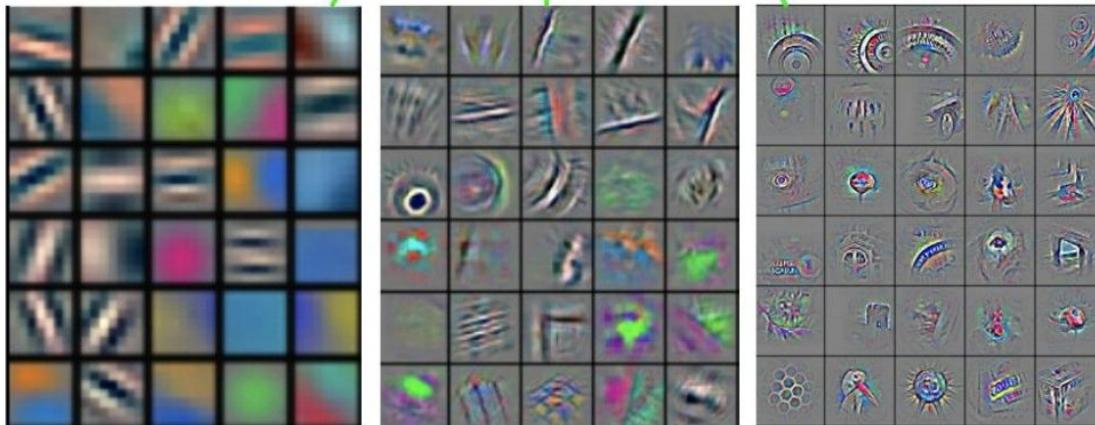


# Convolutional layer

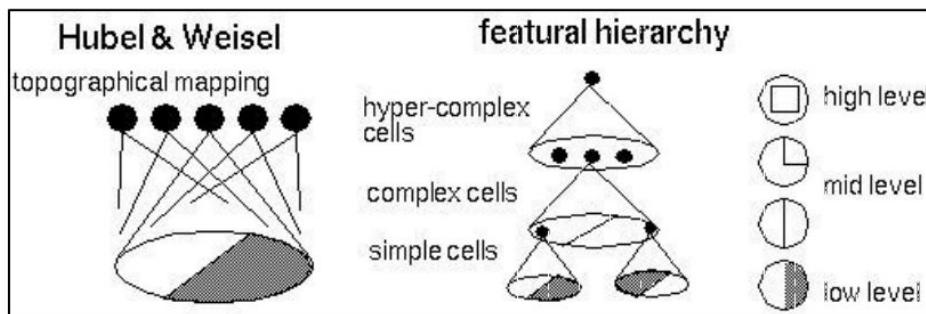


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[From Yann LeCun slides]



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



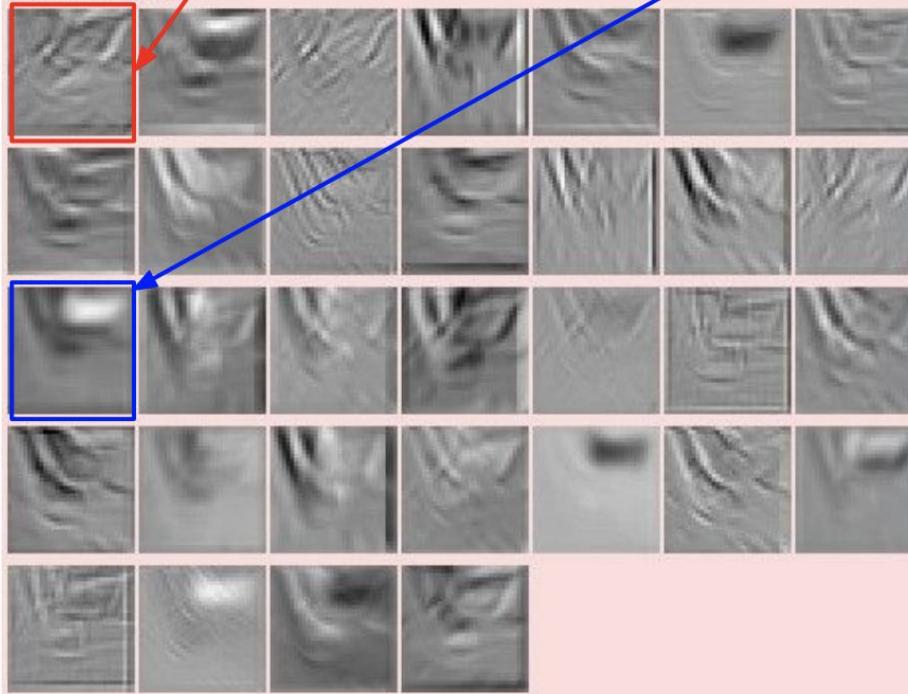
# Convolutional layer and visual cortex

[From Yann LeCun slides]



one filter =>  
one activation map

Activations:



example 5x5 filters  
(32 total)

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

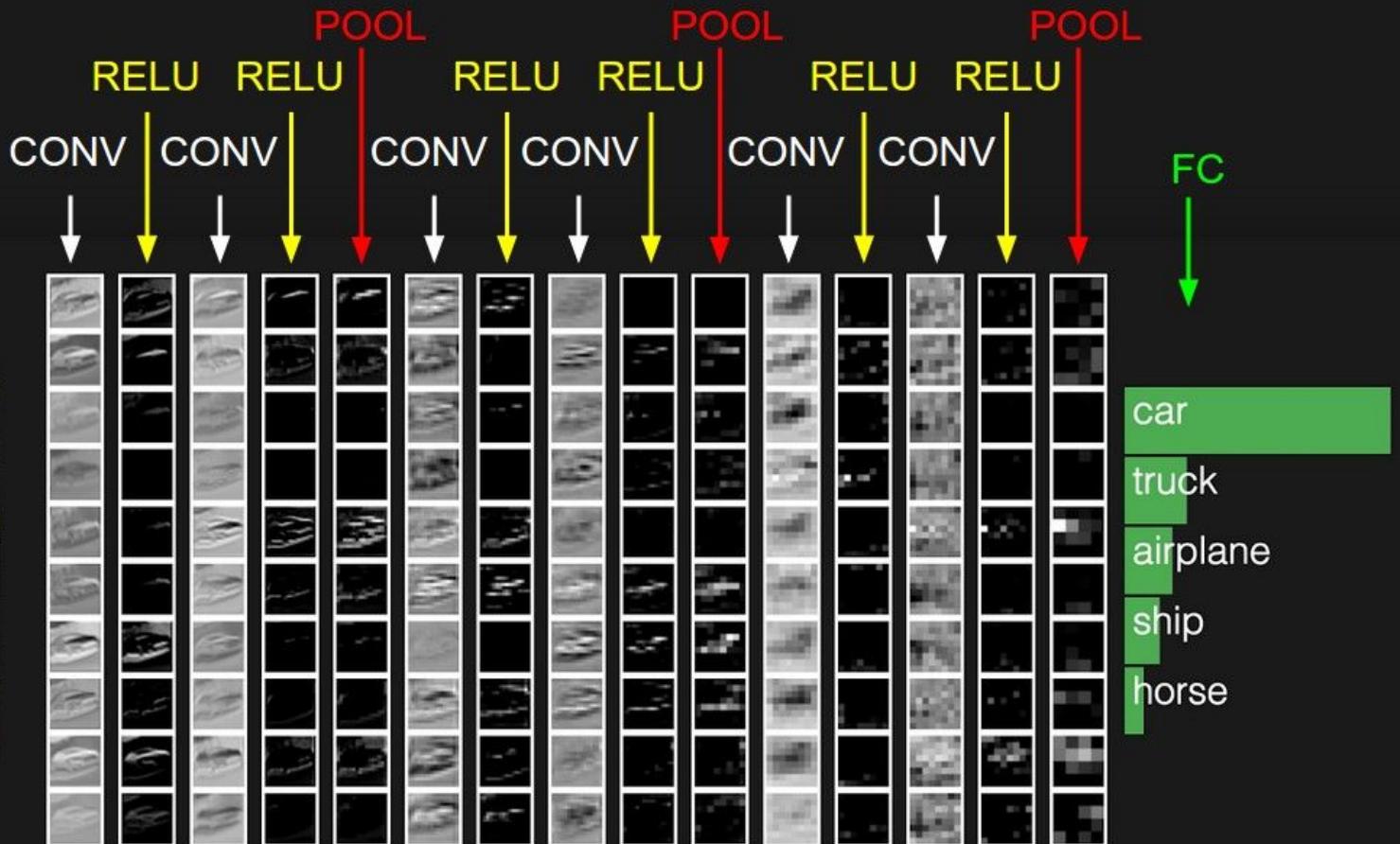
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$



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

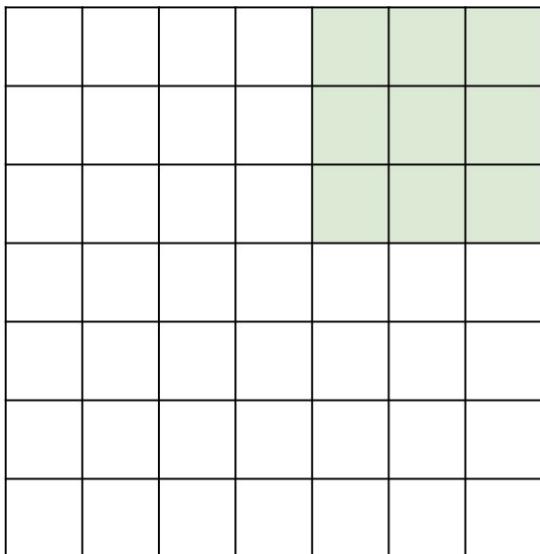
CIFAR-10 online demo:

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



## A closer look at spatial dimensions:

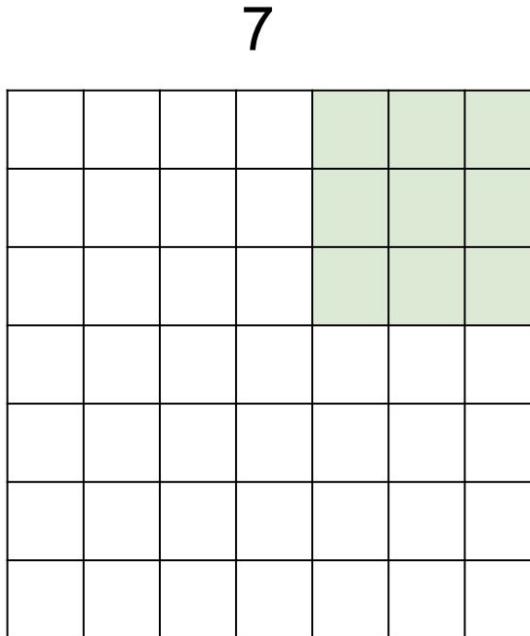
7



7x7 input (spatially)  
assume 3x3 filter

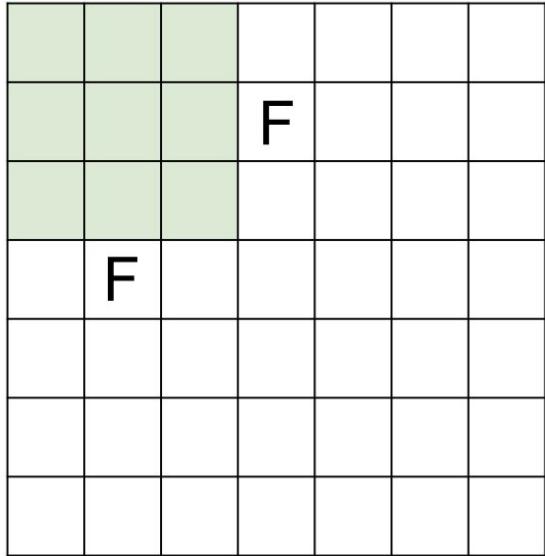
**=> 5x5 output**

## A closer look at spatial dimensions:



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

N



N

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

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

$$\text{stride } 1 \Rightarrow (7 - 3)/1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3)/2 + 1 = 3$$

$$\text{stride } 3 \Rightarrow (7 - 3)/3 + 1 = 2.33 : \backslash$$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

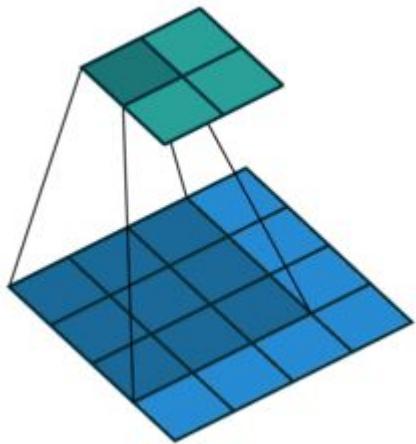
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

e.g.  $F = 3 \Rightarrow$  zero pad with 1

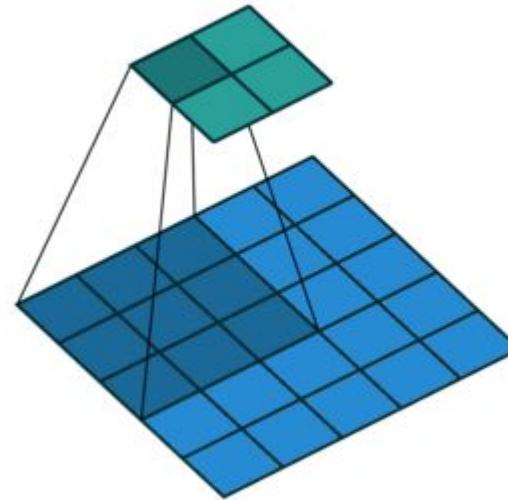
$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

# Strides, padding in convolutional layer

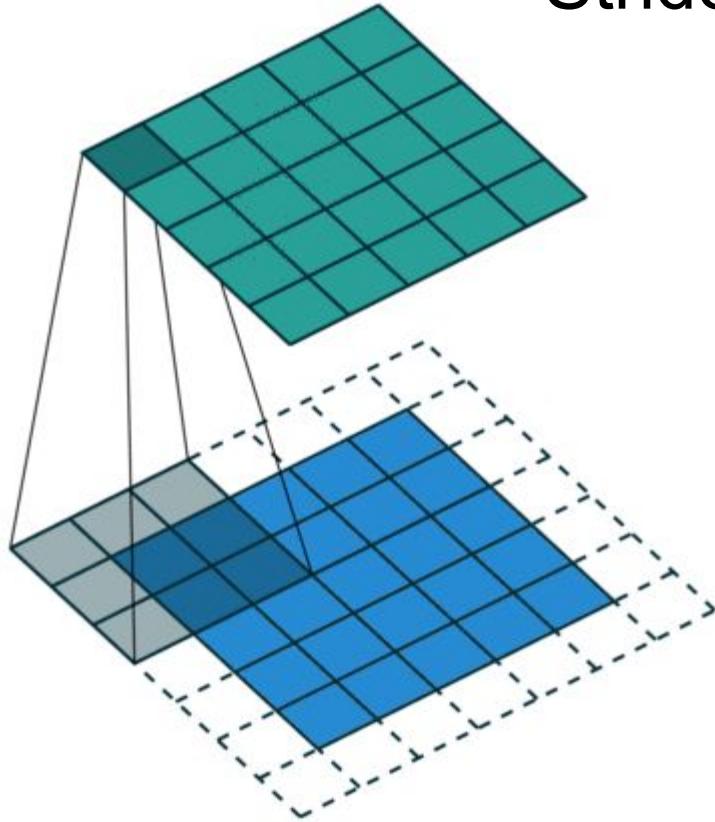


No padding,  
no strides

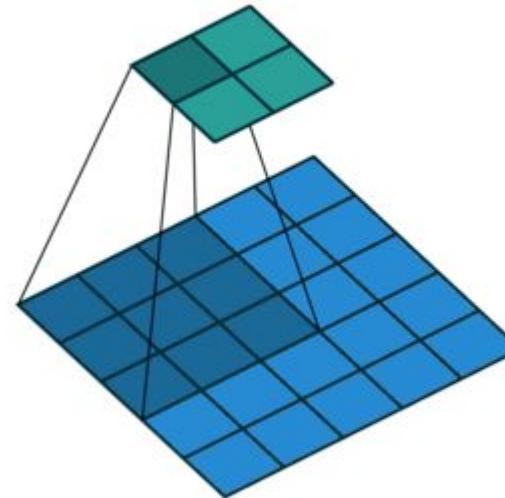


No padding,  
with strides

# Strides, padding in convolutional layer

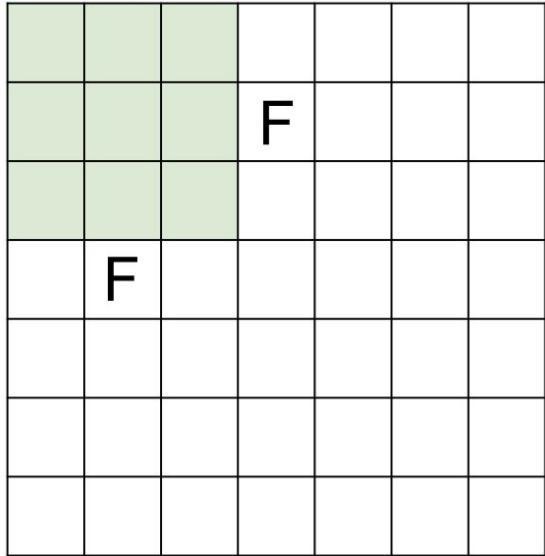


With padding,  
no strides



No padding,  
with strides

N



N

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

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

$$\text{stride } 1 \Rightarrow (7 - 3)/1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3)/2 + 1 = 3$$

$$\text{stride } 3 \Rightarrow (7 - 3)/3 + 1 = 2.33 :\backslash$$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

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.  $F = 3 \Rightarrow$  zero pad with 1

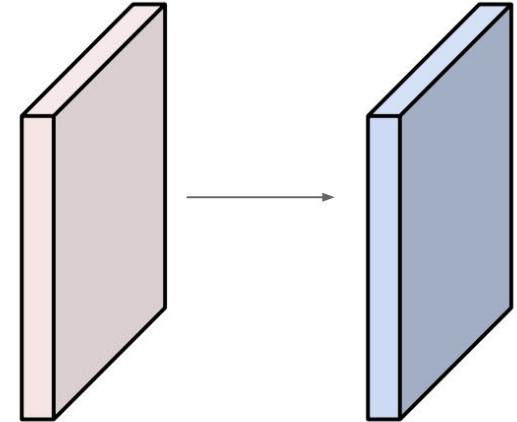
$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

# Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

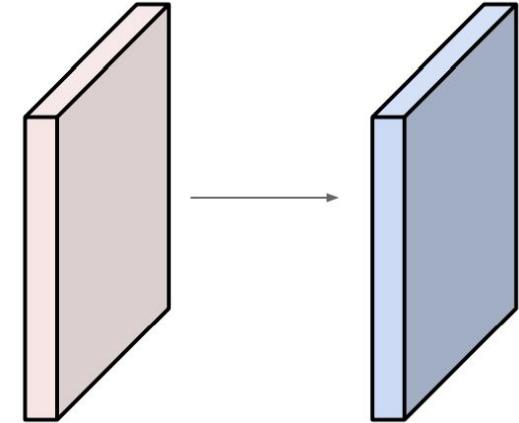


Output volume size: ?

Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2



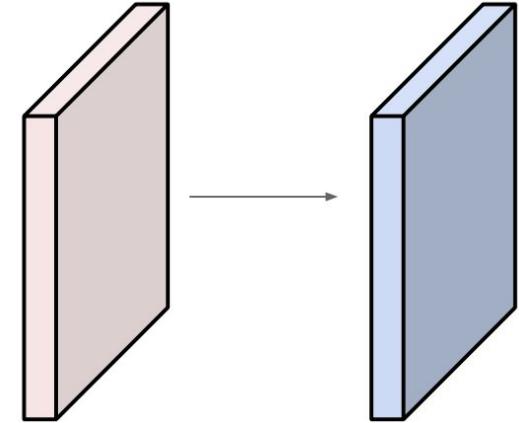
Output volume size:

$(32+2*2-5)/1+1 = 32$  spatially, so  
**32x32x10**

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

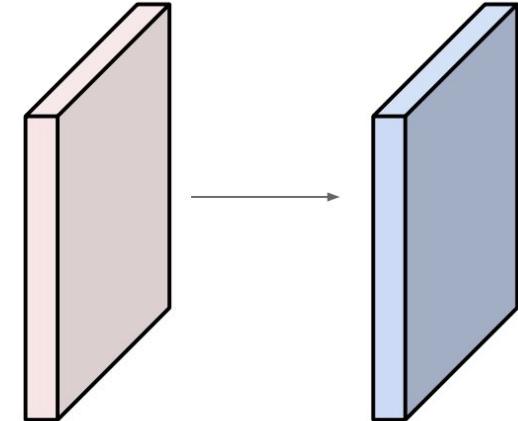


Number of parameters in this layer?

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)  
 $\Rightarrow 76*10 = 760$

**Summary.** To summarize, the Conv Layer:

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- 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  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces  $F \cdot F \cdot D_1$  weights per filter, for a total of  $(F \cdot F \cdot D_1) \cdot K$  weights and  $K$  biases.
- In the output volume, the  $d$ -th depth slice (of size  $W_2 \times H_2$ ) 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.

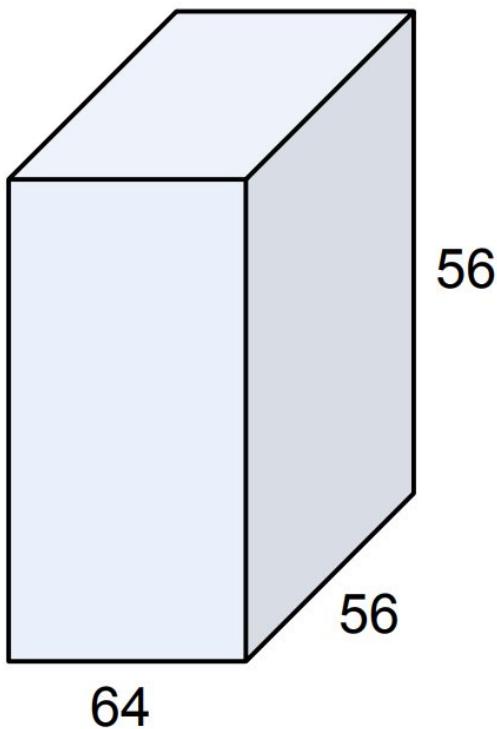
**Summary.** To summarize, the Conv Layer:

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- 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  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces  $F \cdot F \cdot D_1$  weights per filter, for a total of  $(F \cdot F \cdot D_1) \cdot K$  weights and  $K$  biases.
- In the output volume, the  $d$ -th depth slice (of size  $W_2 \times H_2$ ) 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:**

- $K = (\text{powers of 2, e.g. } 32, 64, 128, 512)$
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$  (whatever fits)
- $F = 1, S = 1, P = 0$

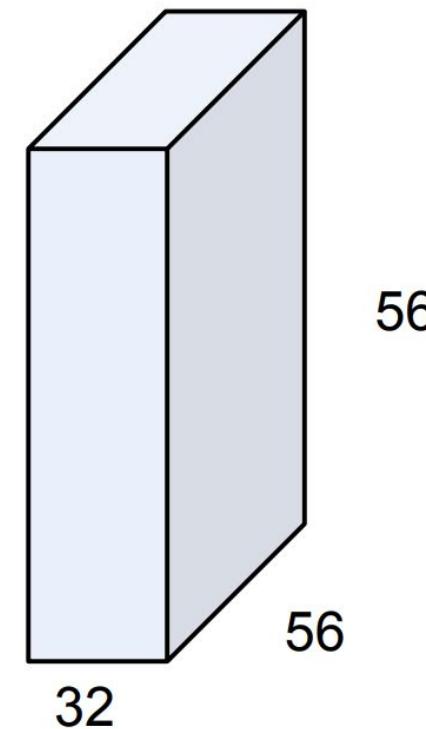
# 1x1 convolutions



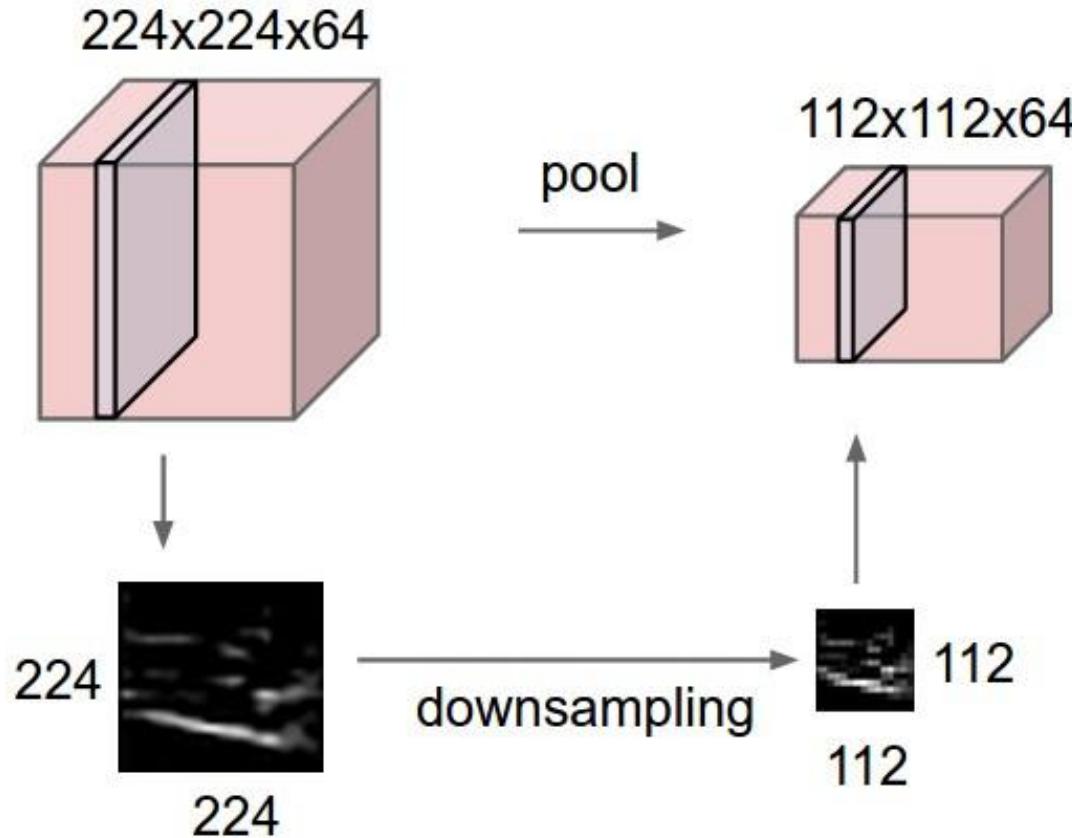
1x1 CONV  
with 32 filters

→

(each filter has size  
1x1x64, and performs a  
64-dimensional dot  
product)



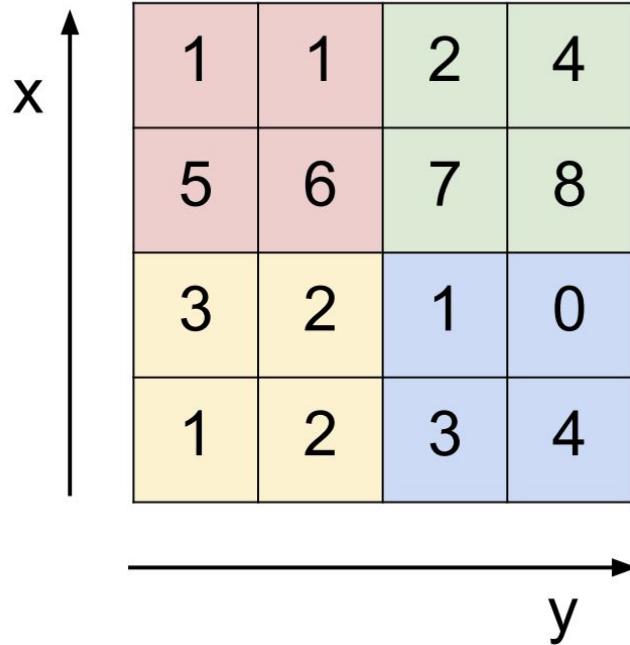
# Pooling layer



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

# Max pooling

Single depth slice



max pool with 2x2 filters  
and stride 2



6	8
3	4

# Pooling layer: sum up

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

# Pooling layer: sum up

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

Common settings:

- $F = 2, S = 2$
- $F = 3, S = 2$

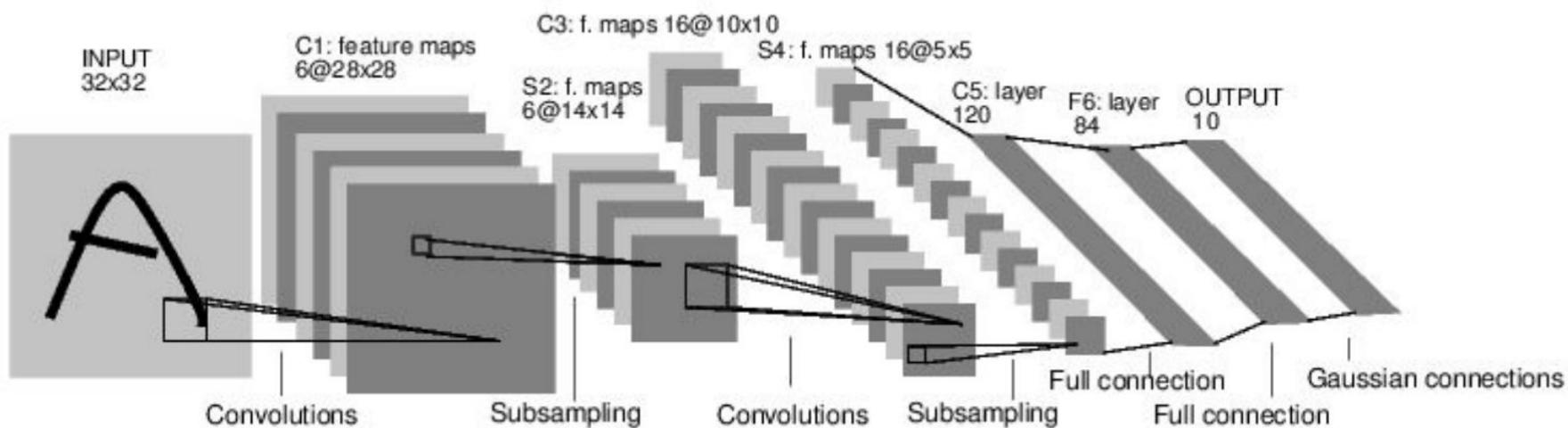
# Example time

Great in-browser CNN (using js) demo by Andrej Karpathy:

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

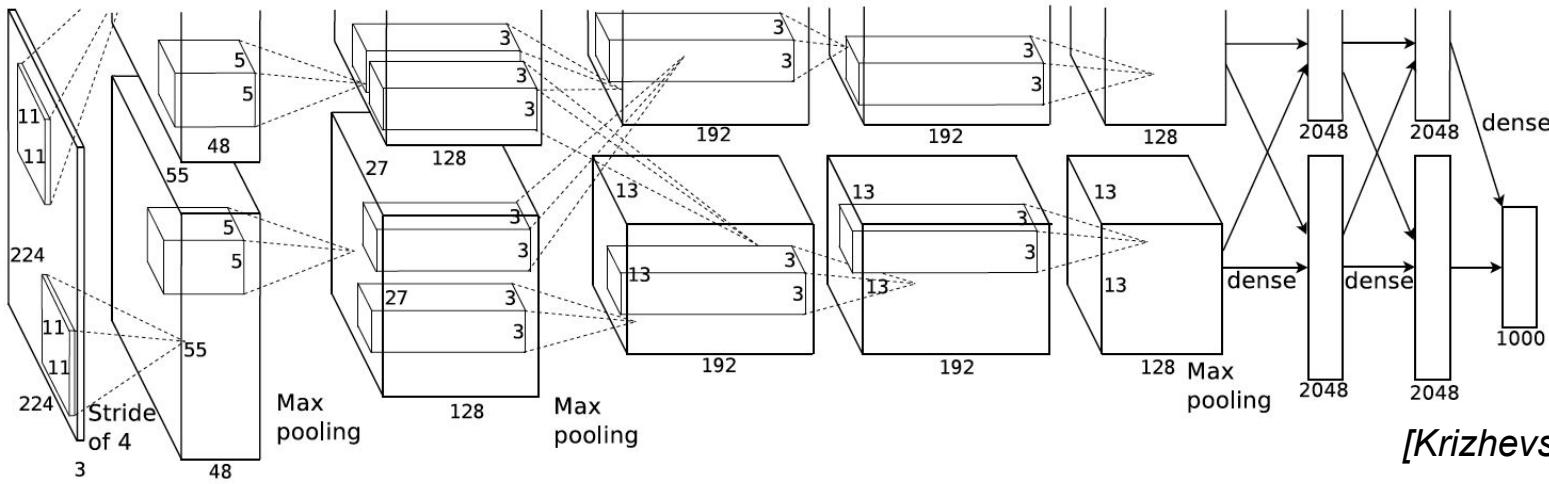
# Architectures overview

# LeNet-5

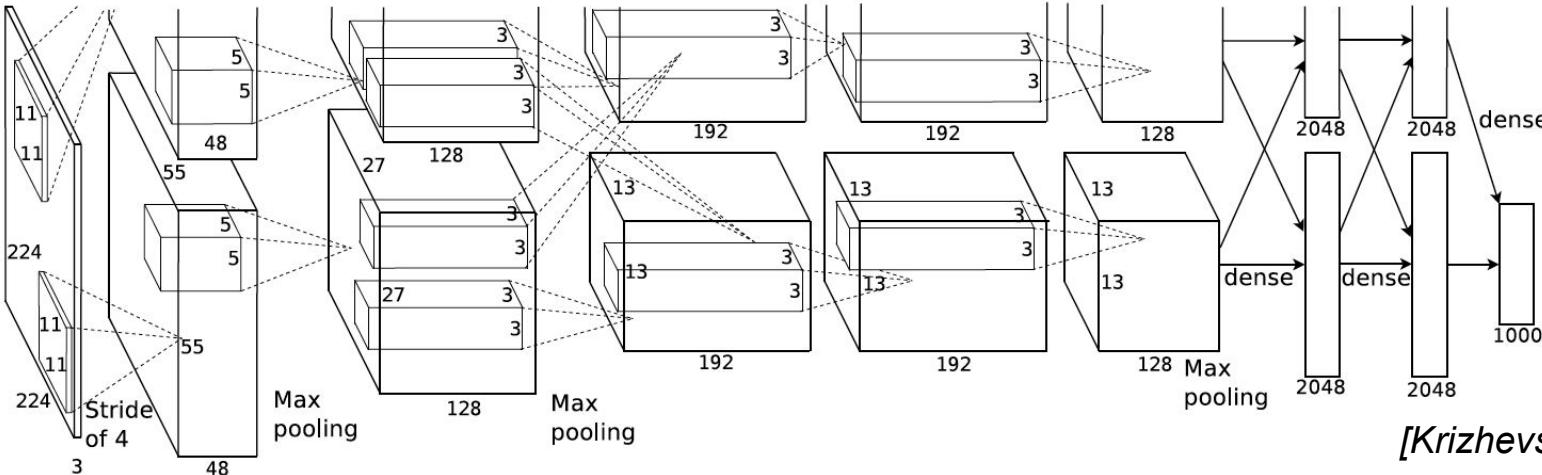


[LeNet-5, LeCun 1998]

# AlexNet



[Krizhevsky et al. 2012]

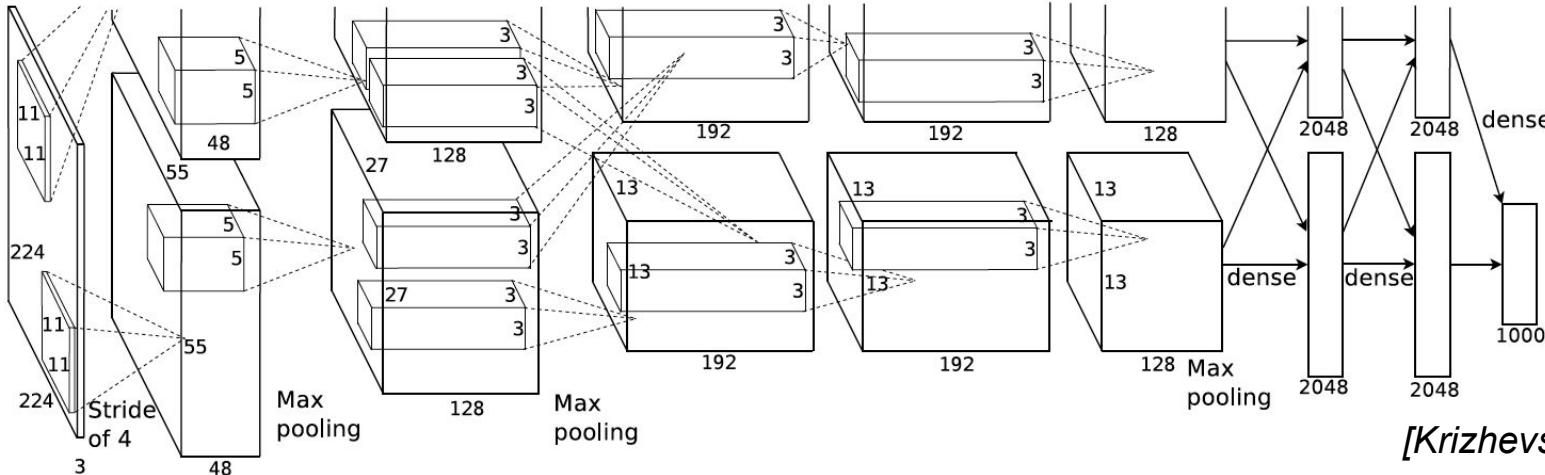


[Krizhevsky et al. 2012]

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: [http://cs231n.stanford.edu/slides/2016/winter1516\\_lecture7.pdf](http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf)



[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

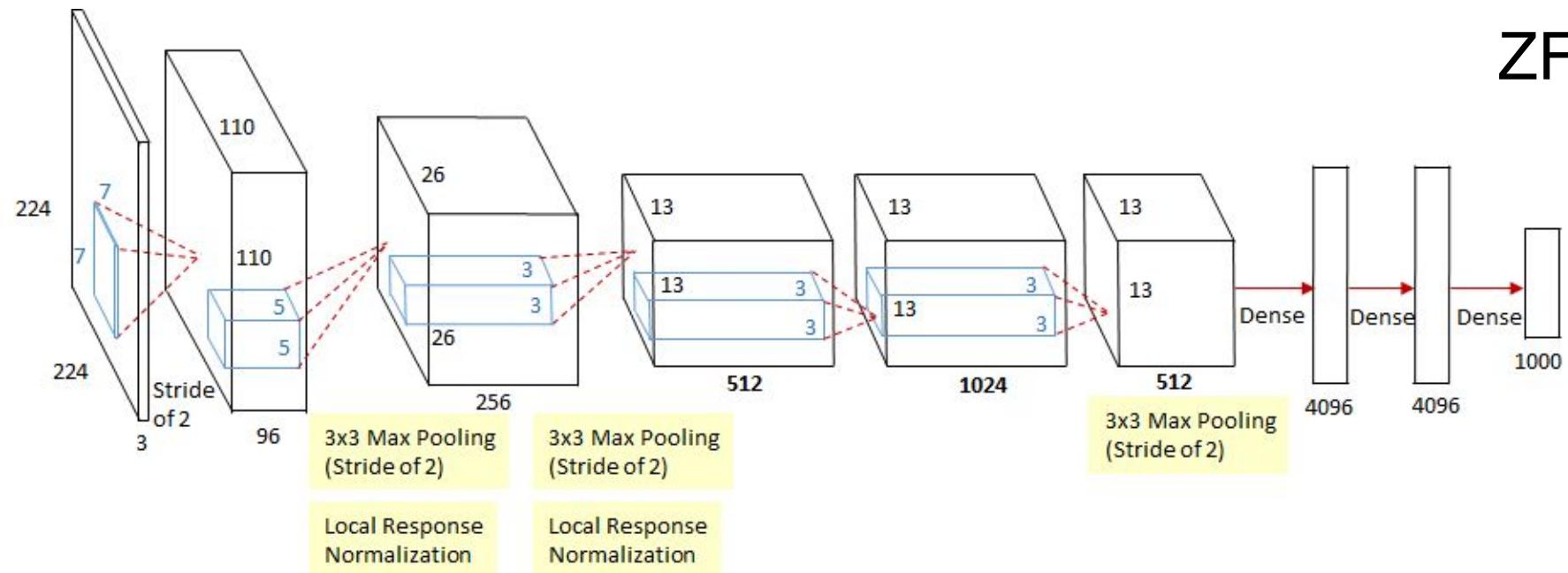
[4096] FC7: 4096 neurons

[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%

# 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%

# VGGNet

INPUT: [224x224x3] memory:  $224 \times 224 \times 3 = 150K$  params: 0 (not counting biases)

CONV3-64: [224x224x64] memory:  $224 \times 224 \times 64 = 3.2M$  params:  $(3 \times 3 \times 3) \times 64 = 1,728$

CONV3-64: [224x224x64] memory:  $224 \times 224 \times 64 = 3.2M$  params:  $(3 \times 3 \times 64) \times 64 = 36,864$

POOL2: [112x112x64] memory:  $112 \times 112 \times 64 = 800K$  params: 0

CONV3-128: [112x112x128] memory:  $112 \times 112 \times 128 = 1.6M$  params:  $(3 \times 3 \times 64) \times 128 = 73,728$

CONV3-128: [112x112x128] memory:  $112 \times 112 \times 128 = 1.6M$  params:  $(3 \times 3 \times 128) \times 128 = 147,456$

POOL2: [56x56x128] memory:  $56 \times 56 \times 128 = 400K$  params: 0

CONV3-256: [56x56x256] memory:  $56 \times 56 \times 256 = 800K$  params:  $(3 \times 3 \times 128) \times 256 = 294,912$

CONV3-256: [56x56x256] memory:  $56 \times 56 \times 256 = 800K$  params:  $(3 \times 3 \times 256) \times 256 = 589,824$

CONV3-256: [56x56x256] memory:  $56 \times 56 \times 256 = 800K$  params:  $(3 \times 3 \times 256) \times 256 = 589,824$

POOL2: [28x28x256] memory:  $28 \times 28 \times 256 = 200K$  params: 0

CONV3-512: [28x28x512] memory:  $28 \times 28 \times 512 = 400K$  params:  $(3 \times 3 \times 256) \times 512 = 1,179,648$

CONV3-512: [28x28x512] memory:  $28 \times 28 \times 512 = 400K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [28x28x512] memory:  $28 \times 28 \times 512 = 400K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params: 0

CONV3-512: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

CONV3-512: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$

POOL2: [7x7x512] memory:  $7 \times 7 \times 512 = 25K$  params: 0

FC: [1x1x4096] memory: 4096 params:  $7 \times 7 \times 512 \times 4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params:  $4096 \times 4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params:  $4096 \times 1000 = 4,096,000$

**TOTAL** memory:  $24M * 4 \text{ bytes} \approx 93\text{MB} / \text{image}$  (only forward!  $\sim 2$  for bwd)

**TOTAL** params: 138M parameters

ConvNet Configuration			
B	C	D	E
13 weight layers	16 weight layers	16 weight layers	19 weight layers
put (224 x 224 RGB image)			
conv3-64	conv3-64	conv3-64	conv3-64
<b>conv3-64</b>	conv3-64	conv3-64	conv3-64
		maxpool	
conv3-128	conv3-128	conv3-128	conv3-128
<b>conv3-128</b>	conv3-128	conv3-128	conv3-128
		maxpool	
conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256
		<b>conv1-256</b>	conv3-256
			conv3-256
		maxpool	
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
		<b>conv1-512</b>	conv3-512
			conv3-512
		maxpool	
conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512
		<b>conv1-512</b>	conv3-512
			conv3-512
		maxpool	
FC-4096			
FC-4096			
FC-1000			
soft-max			

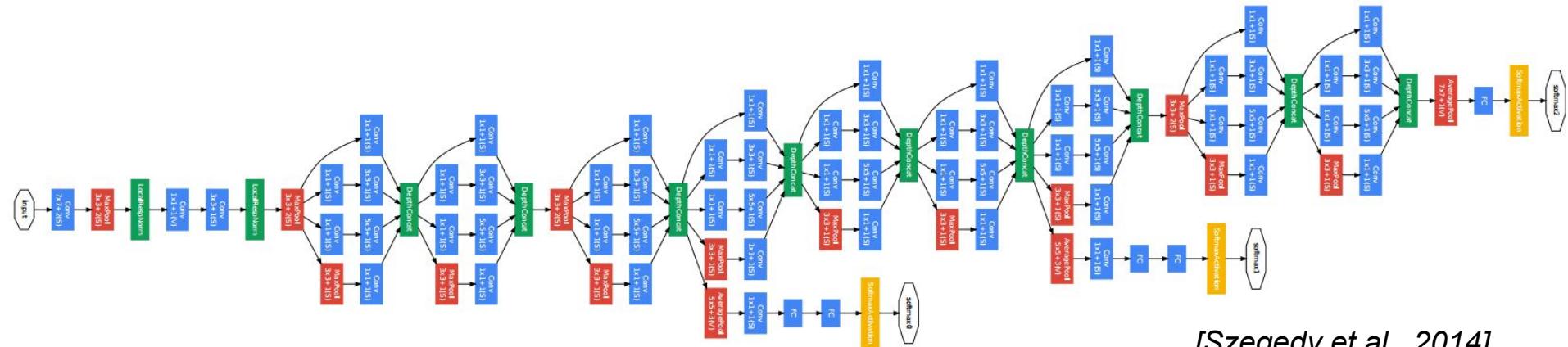
# VGGNet

7.3% top 5 error

TOTAL memory:  $24M * 4 \text{ bytes} \approx 93\text{MB} / \text{image}$  (only forward!  $\sim 2$  for bwd)  
TOTAL params: 138M parameters

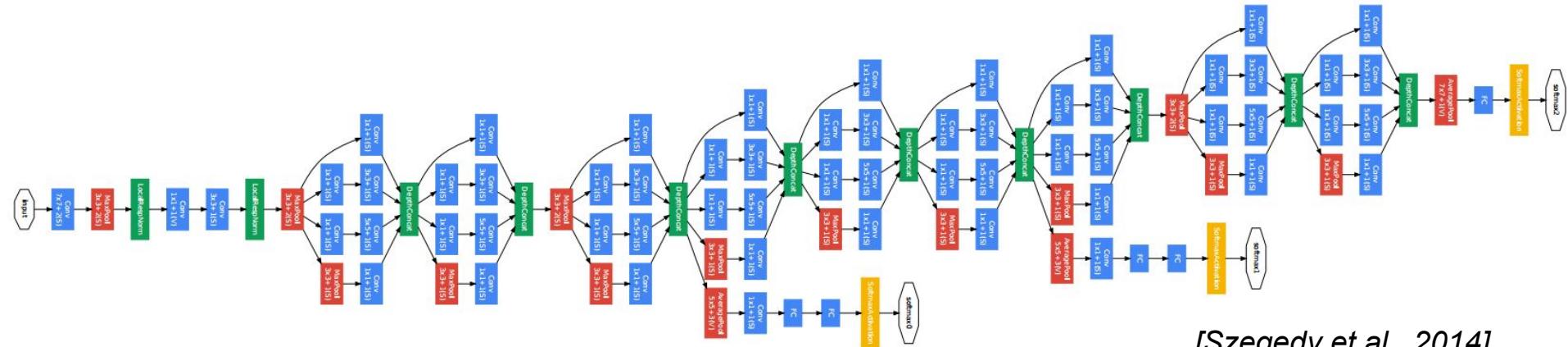
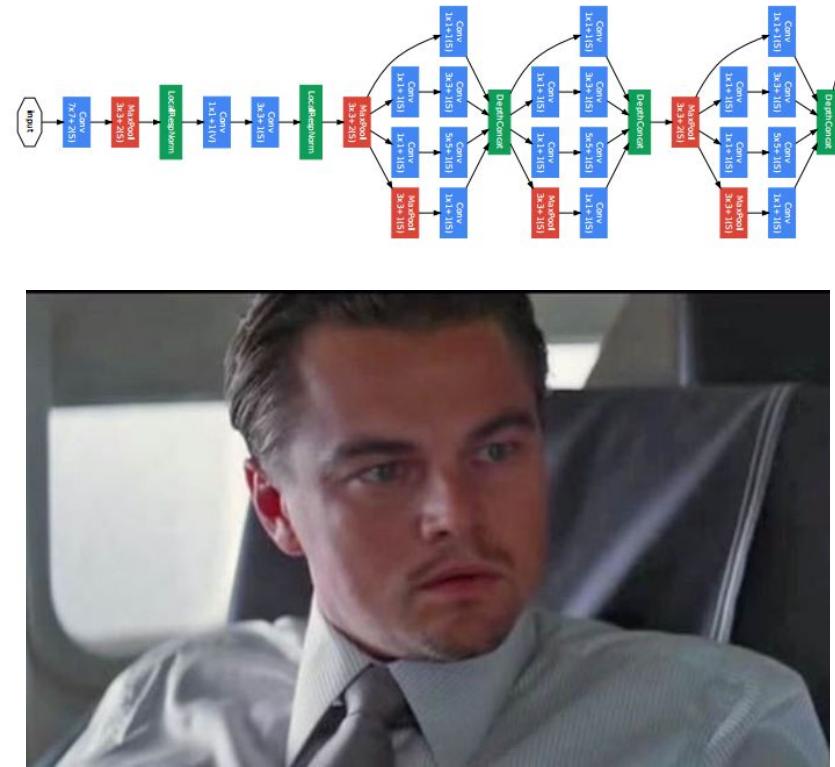
ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	cc
<b>conv3-64</b>	conv3-64	conv3-64	cc
maxpool			
conv3-128	conv3-128	conv3-128	co
<b>conv3-128</b>	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256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

# GoogLeNet



[Szegedy et al., 2014]

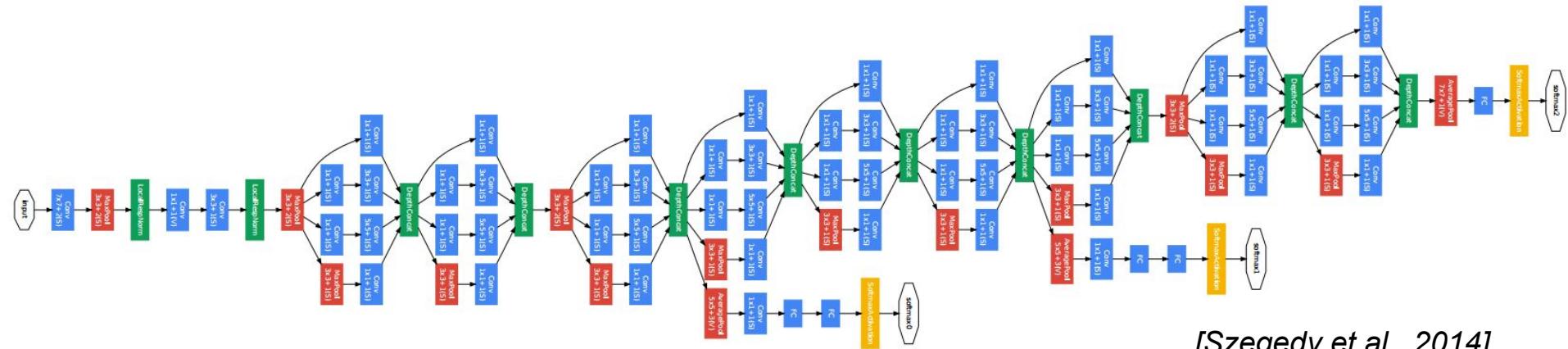
# GoogLeNet



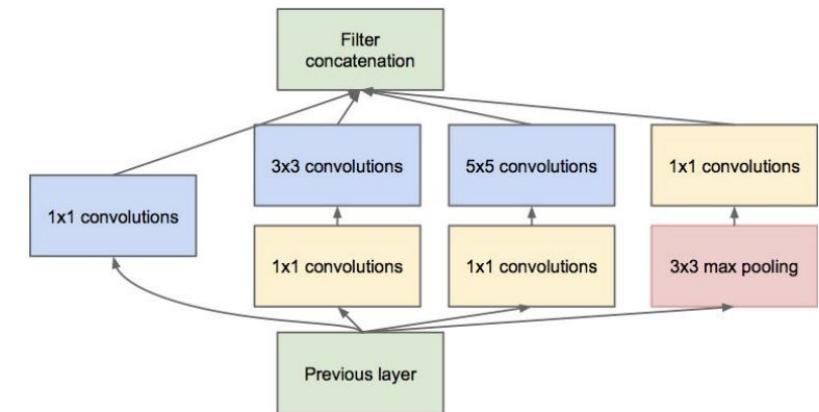
[Szegedy et al., 2014]

## Inception module

# GoogLeNet

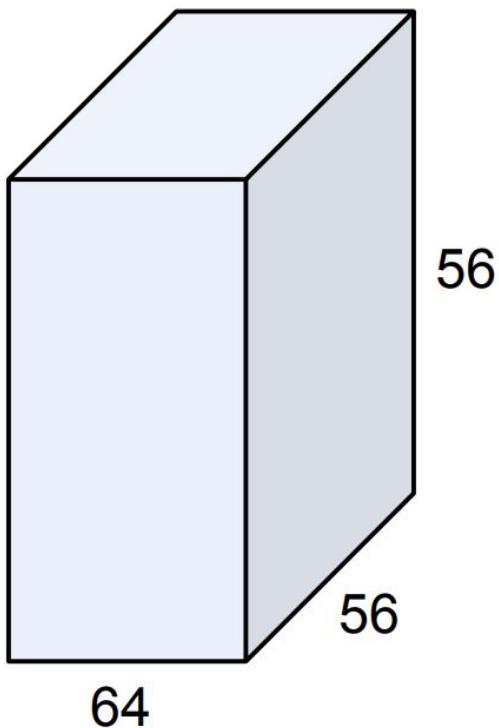


[Szegedy et al., 2014]



## Inception module

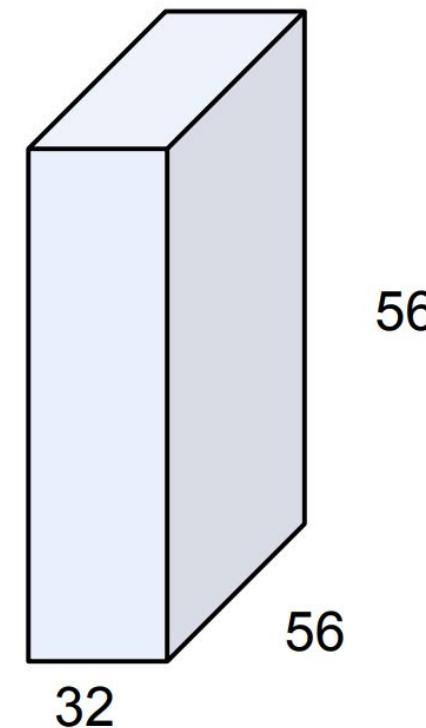
# Once again: 1x1 convolutions



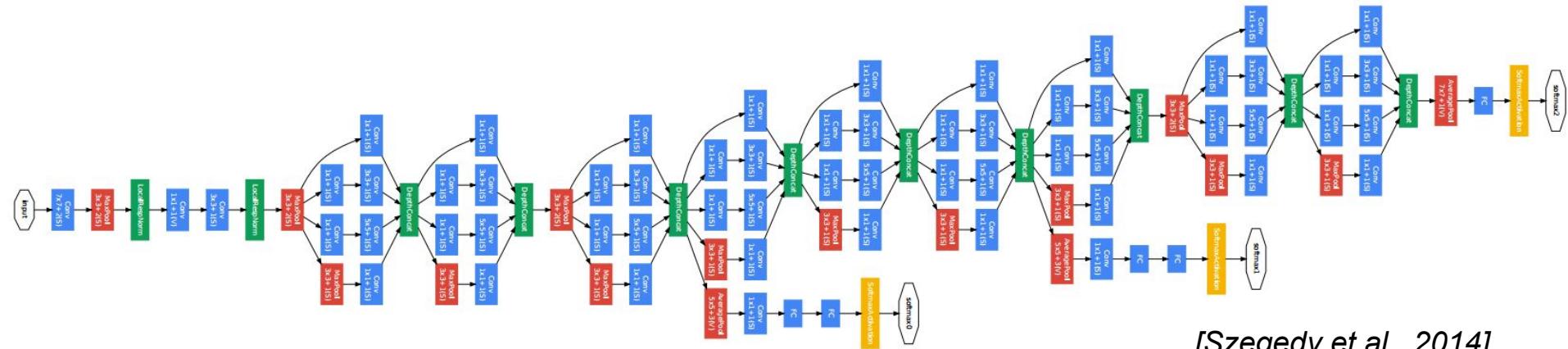
1x1 CONV  
with 32 filters

→

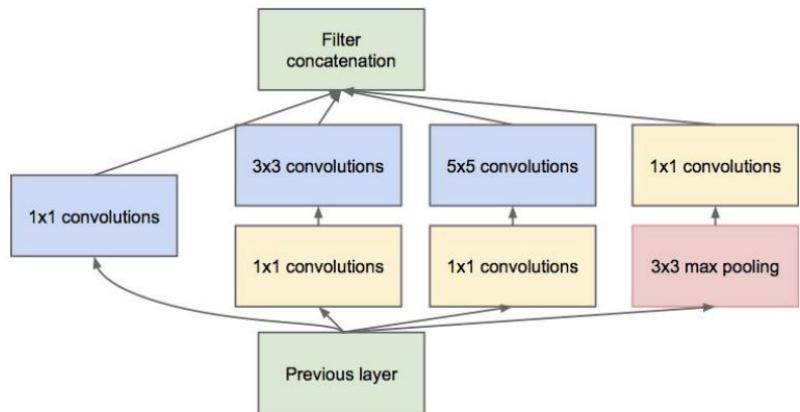
(each filter has size  
1x1x64, and performs a  
64-dimensional dot  
product)



# GoogLeNet



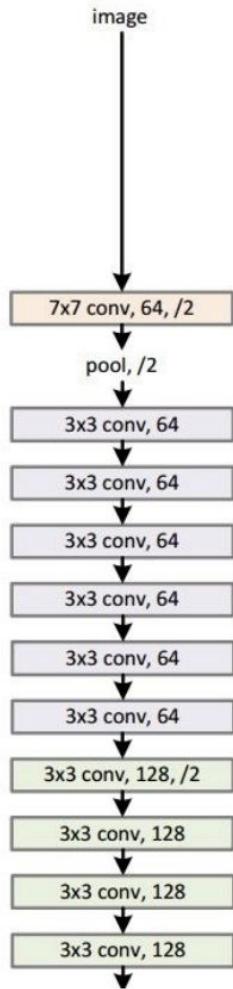
[Szegedy et al., 2014]



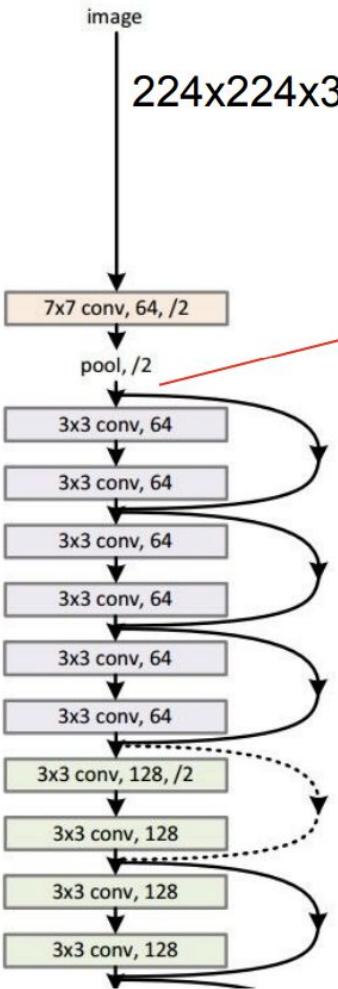
## Inception module

ILSVRC 2014 winner (6.7% top 5 error)

34-layer plain



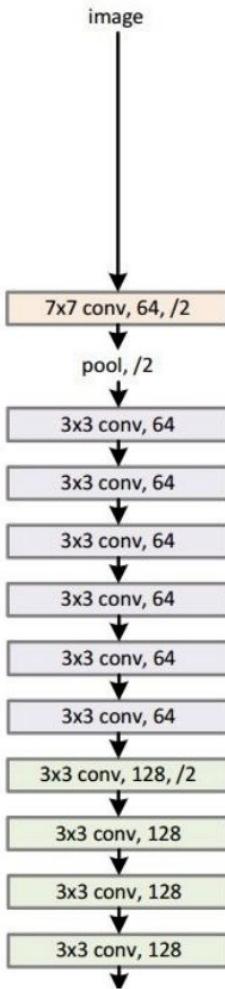
34-layer residual



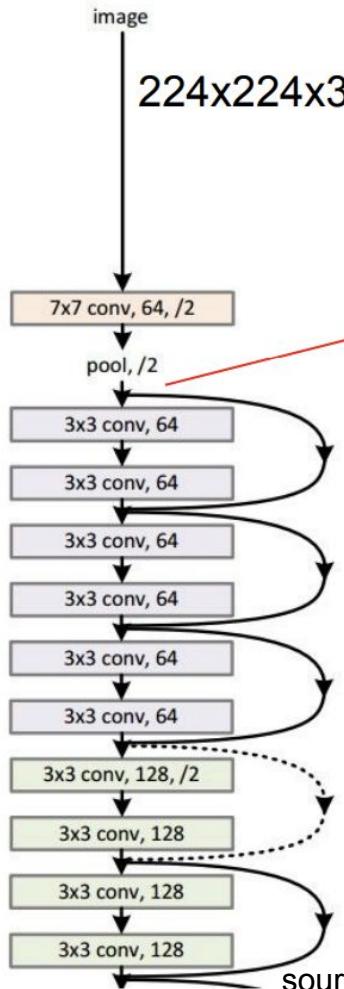
spatial dimension  
only 56x56!

ILSVRC 2015 winner (3.6% top 5 error)

34-layer plain



34-layer residual



[He et al., 2015]

spatial dimension  
only 56x56!

- 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: [http://cs231n.stanford.edu/slides/2016/winter1516\\_lecture7.pdf](http://cs231n.stanford.edu/slides/2016/winter1516_lecture7.pdf)

# MSRA @ ILSVRC & COCO 2015 Competitions

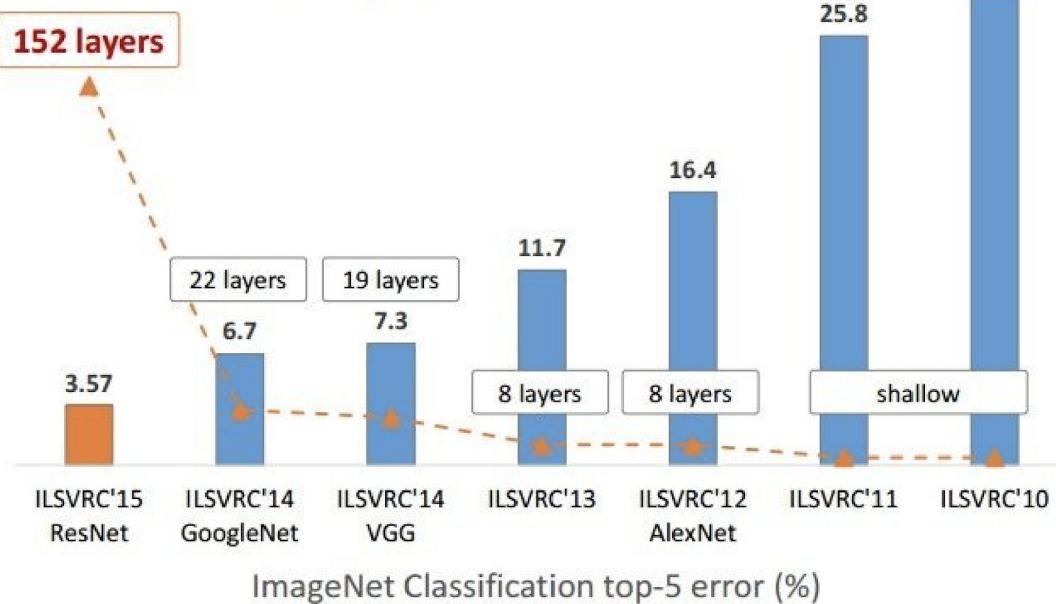
- **1st places in all five main tracks**

- ImageNet Classification: “*Ultra-deep*” (quote Yann) **152-layer nets**
- ImageNet Detection: **16%** better than 2nd
- ImageNet Localization: **27%** better than 2nd
- COCO Detection: **11%** better than 2nd
- COCO Segmentation: **12%** better than 2nd

\*improvements are relative numbers

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

# Revolution of Depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

(slide from Kaiming He's recent presentation)

# Batch normalization

*It's usually a good idea to normalize linear model inputs.*

*(c) Every machine learning lecturer, ever*

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

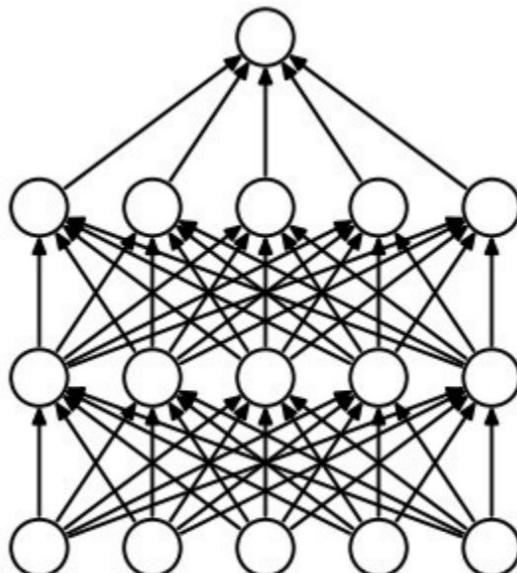
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

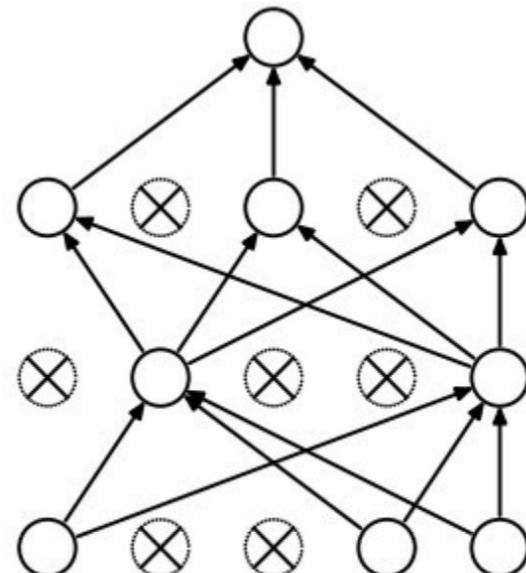
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

# Regularization: Dropout

Some neurons are “dropped” during training.



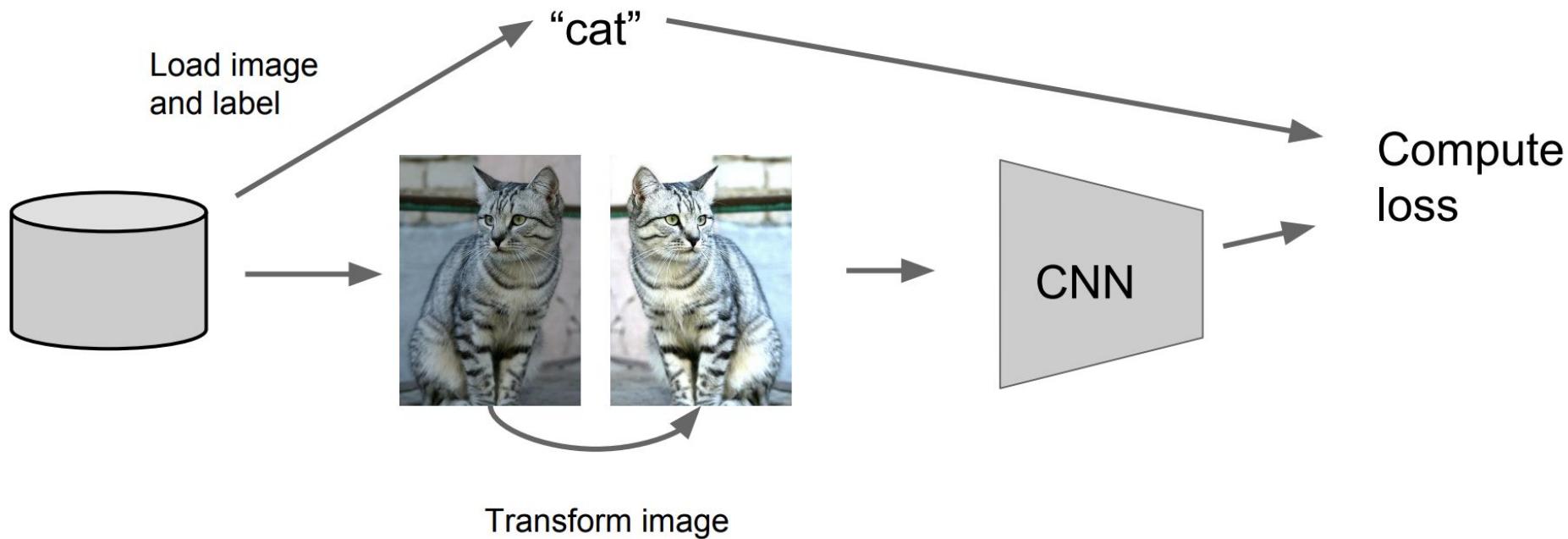
(a) Standard Neural Net



(b) After applying dropout.

Actually, on test case output should be normalized. See sources for more info.

# Regularization: data augmentation



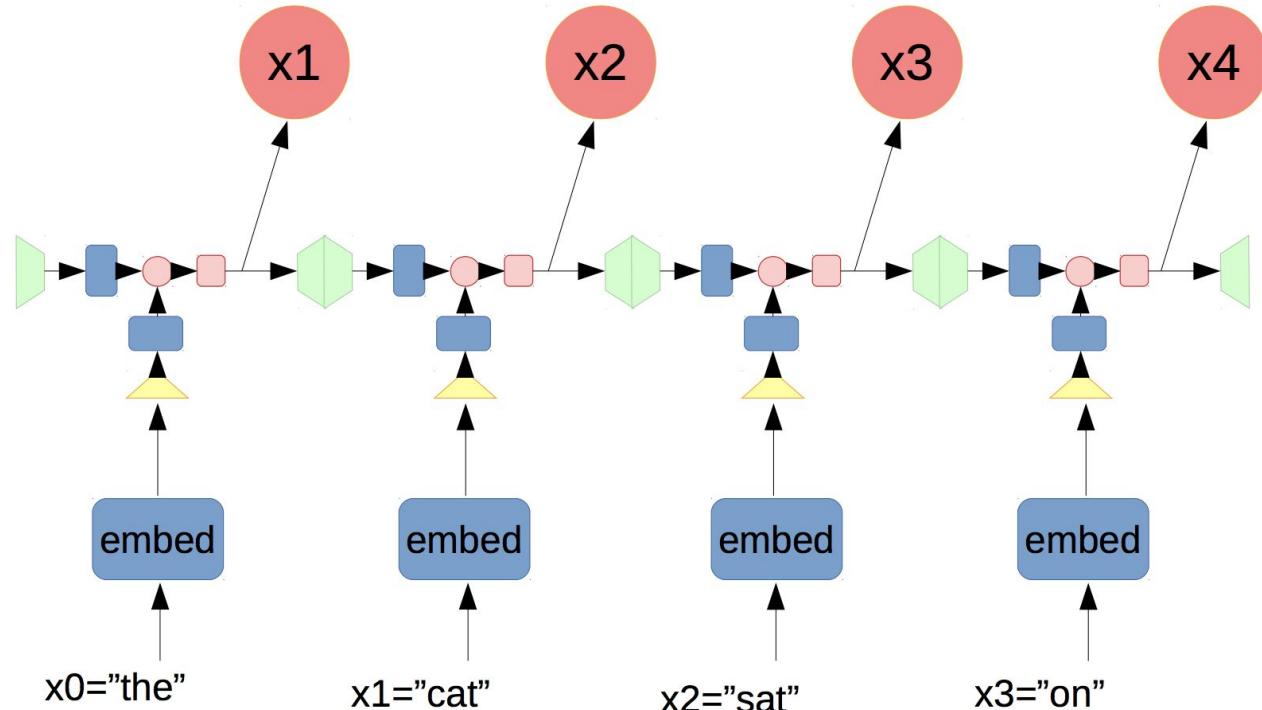
# Summary

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

That's all. Feel free to ask any questions.

# Backup

# Recap: RNN

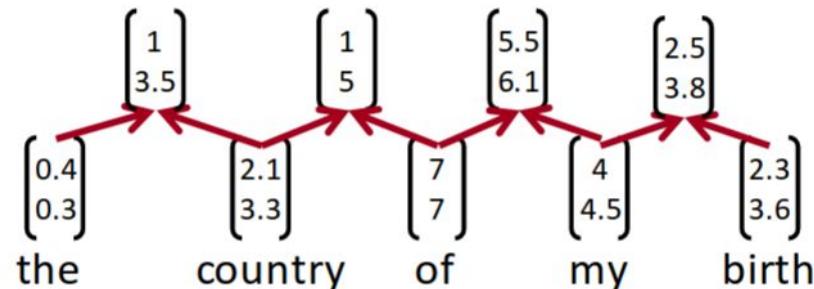


# From RNN to CNN

- RNN: Get compositional vectors for grammatical phrases only
- CNN: What if we compute vectors for every possible phrase?
  - Example: “*the country of my birth*” computes vectors for:
    - *the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth*
- Regardless of whether it is grammatical
- Wouldn’t need parser
- Not very linguistically or cognitively plausible

# From RNN to CNN

- Imagine using only bigrams



- Same operation as in RNN, but for every pair

$$p = \tanh \left( W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right)$$

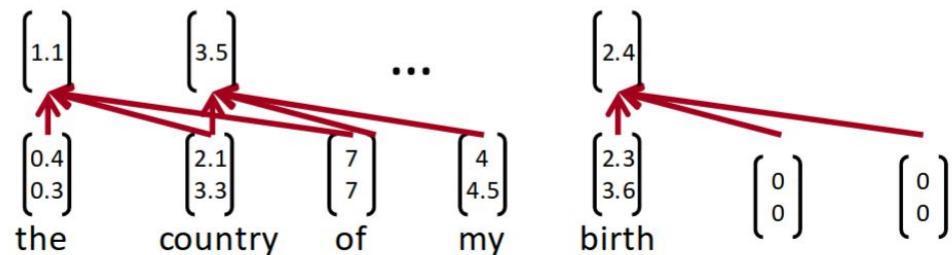
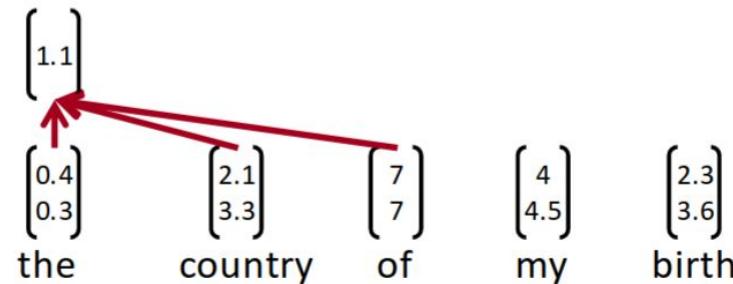
- Can be interpreted as convolution over the word vectors

# From RNN to CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

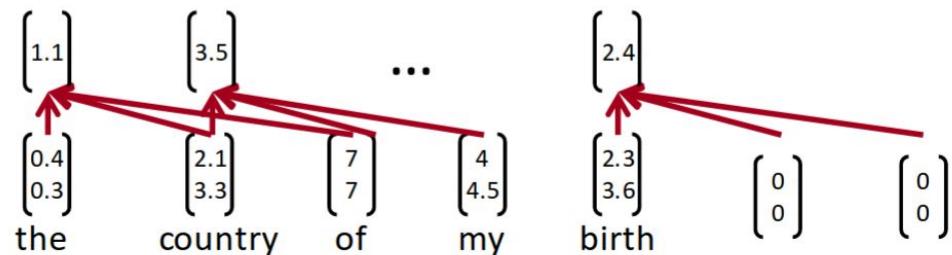
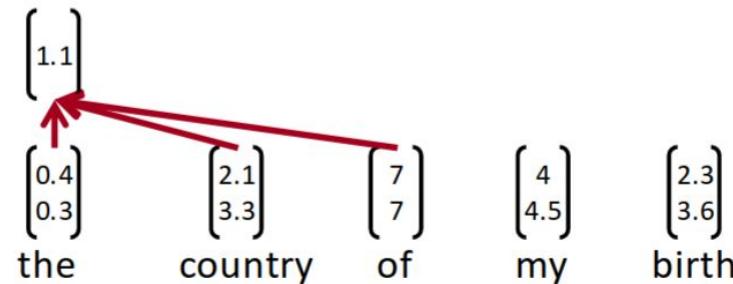


# One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



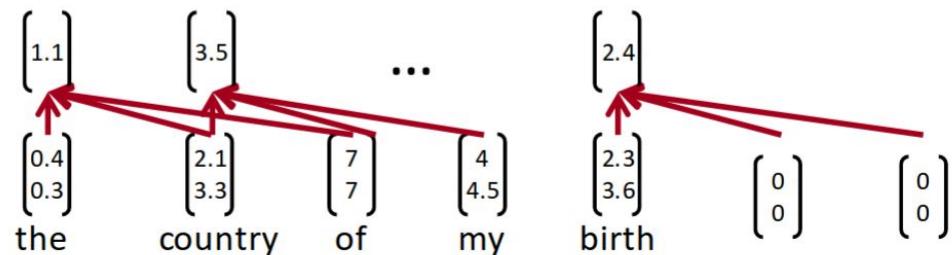
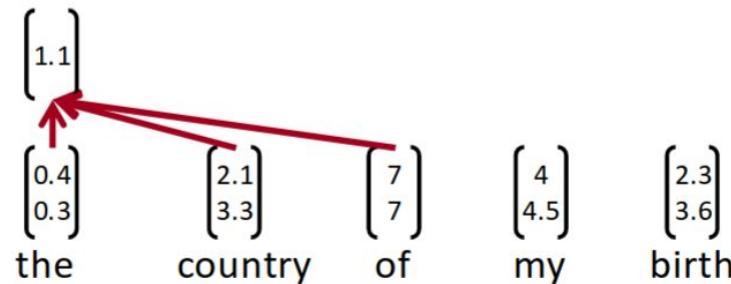
# One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

What's next?

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



# One layer CNN

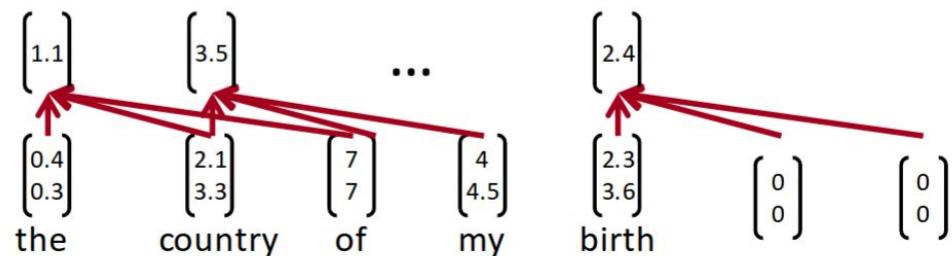
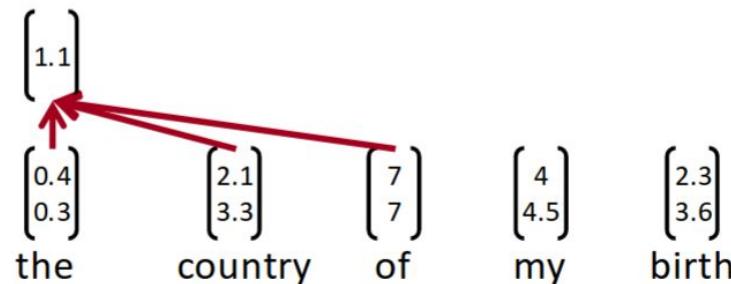
- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

What's next?

We need more features!

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



- Feature representation is based on some applied filter:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

- Let's use pooling over the time axis:  $\hat{c} = \max\{\mathbf{c}\}$
- Now the length of  $\mathbf{c}$  is irrelevant!
- So we can use filters based on unigrams, bigrams, tri-grams, 4-grams, etc.

# Another example from Kim (2014) paper

