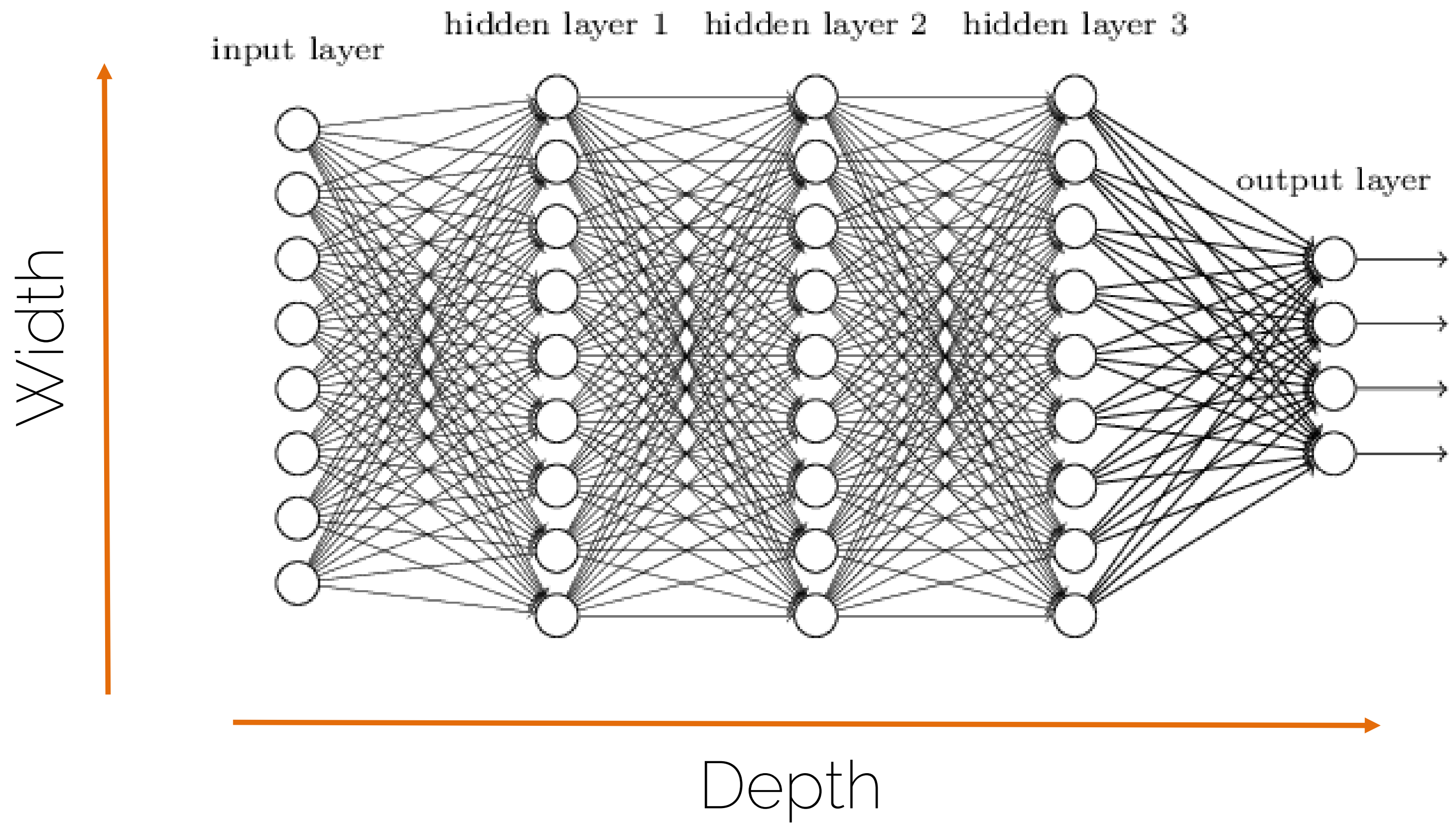
Lecture 6 Recap

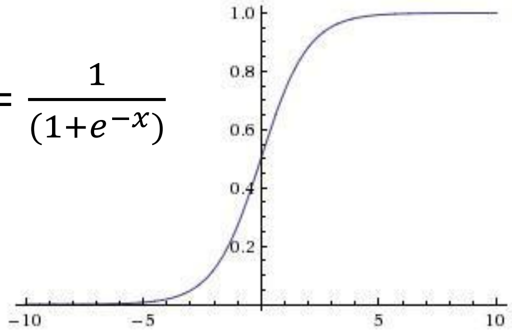
What do we know so far?



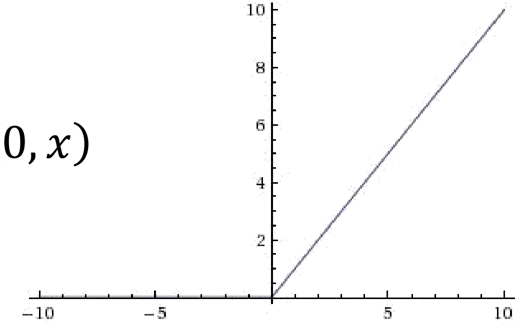
What do we know so far?

Activation Functions (non-linearities)

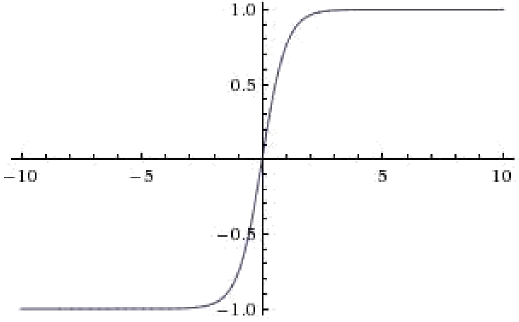
Sigmoid: $\boldsymbol{\sigma(x) = \frac{1}{(1+e^{-x})}}$



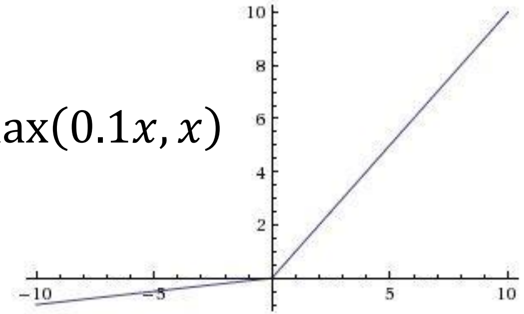
ReLU: $\mathbf{max(0,}\,x\mathbf{)}$



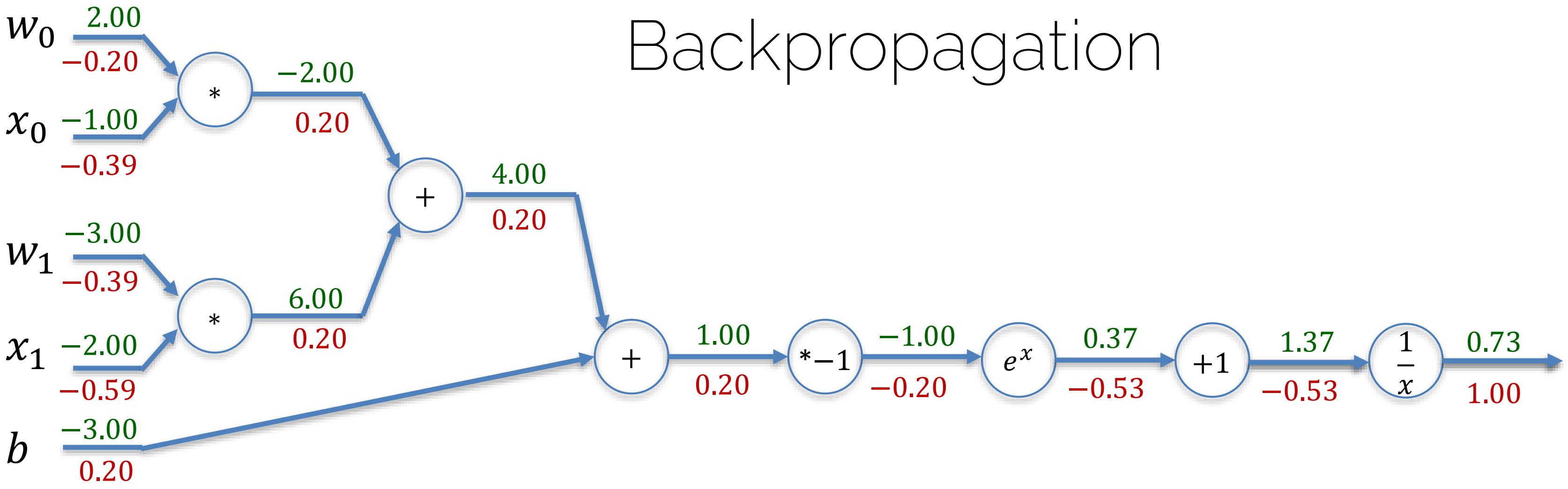
tanh: $\mathbf{tanh(}x\mathbf{)}$



Leaky ReLU: $\mathbf{max(0.1}x\mathbf{,}\,x\mathbf{)}$

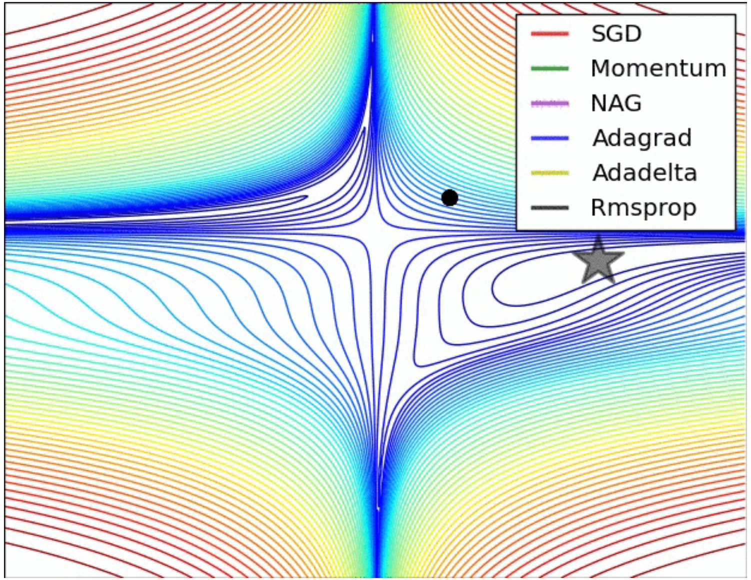


What do we know so far?

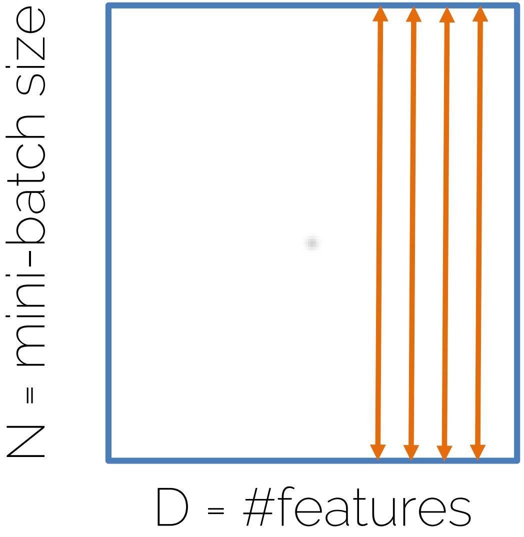


What do we know so far?

SGD Variations (Momentum, etc.)



Minibatch SGD



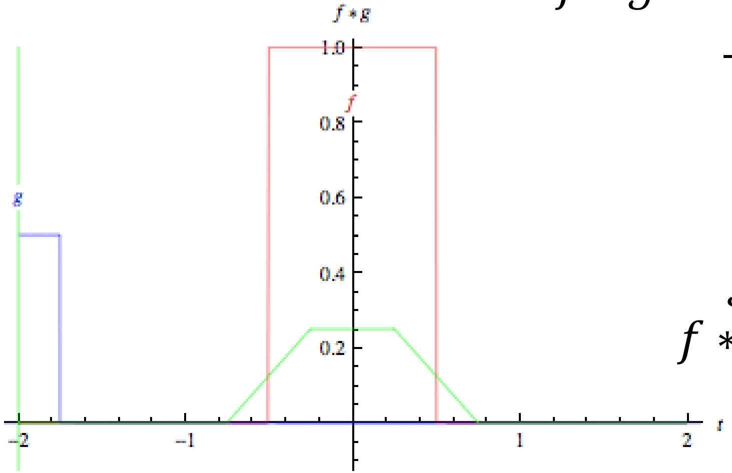
Why not only more Layers?

- We can not make networks arbitrarily complex
  - Why not just go deeper and get better?
  - No structure!!
  - It's just brute force!
  - Optimization becomes hard
  - Performance plateaus / drops!

Convolutional Neural Networks (CNNs)

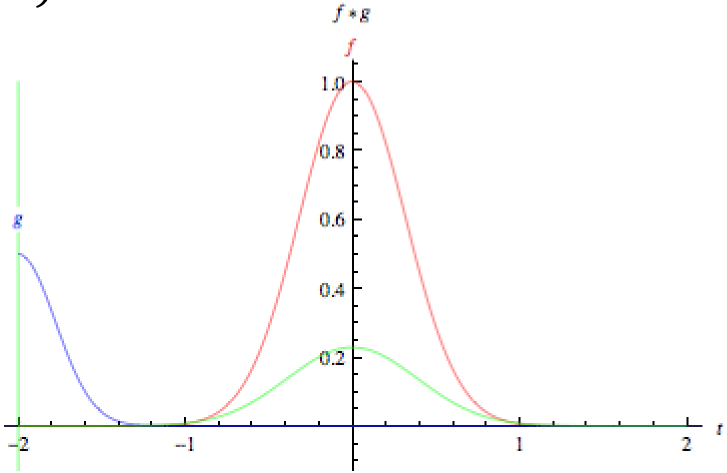
What are Convolutions?

$$f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$



Convolution of two box functions

f = red
g = blue
$f * g$ = green



Convolution of two Gaussians

application of a filter to a function
the 'smaller' one is typically called the filter kernel

What are Convolutions?

Discrete case: box filter

f	**4**	**3**	**2**	**-5**	**3**	**5**	**2**	**5**	**5**	**6**
g	**1/3**	**1/3**	**1/3**							

'Slide' filter kernel from left to right; at each position, compute a single value in the output data

What are Convolutions?

Discrete case: box filter

f	4	3	2	-5	3	5	2	5	5	6
g								1/3	1/3	1/3
$f * g$		3	0	0	1	10/3	4	4	16/3	

$$5 \cdot \frac{1}{3} + 5 \cdot \frac{1}{3} + 6 \cdot \frac{1}{3} = \frac{16}{3}$$

What are Convolutions?

Discrete case: box filter

4	**3**	**2**	**-5**	**3**	**5**	**2**	**5**	**5**	**6**

1/3	**1/3**	**1/3**

??	**3**	**0**	**0**	**1**	**10/3**	**4**	**4**	**16/3**	**??**

What to do at boundaries?

What are Convolutions?

Discrete case: box filter

4	3	2	-5	3	5	2	5	5	6

1/3	1/3	1/3

??	3	0	0	1	10/3	4	4	16/3	??

What to do at boundaries?

1) Shrink

3	0	0	1	10/3	4	4	16/3

2) Pad
often '0'

7/3	3	0	0	1	10/3	4	4	16/3	11/3

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6		

$$5 \cdot 3 + (-1) \cdot 3 + (-1) \cdot 2 + (-1) \cdot 0 + (-1) \cdot 4 =$$
$$15 - 9 = 6$$

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6	1	

$$5 \cdot 2 + (-1) \cdot 2 + (-1) \cdot 1 + (-1) \cdot 3 + (-1) \cdot 3 =$$
$$10 - 9 = 1$$

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

$$5 \cdot 1 + (-1) \cdot (-5) + (-1) \cdot (-3) + (-1) \cdot 3 + (-1) \cdot 2 =$$
$$5 + 3 = 1$$

Output 3x3

6	1	8

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6	1	8
-7		

$$5 \cdot 0 + (-1) \cdot 3 + (-1) \cdot 0 + (-1) \cdot 1 + (-1) \cdot 3 =$$
$$0 - 7 = -7$$

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6	1	8
-7	9	

$$5 \cdot 3 + (-1) \cdot 2 + (-1) \cdot 3 + (-1) \cdot 1 + (-1) \cdot 0 =$$
$$15 - 6 = 9$$

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6	1	8
-7	9	2

$$5 \cdot 3 + (-1) \cdot 1 + (-1) \cdot 5 + (-1) \cdot 4 + (-1) \cdot 3 =$$
$$15 - 13 = 2$$

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6	1	8
-7	9	2
-4		

$$5 \cdot 0 + (-1) \cdot 0 + (-1) \cdot 1 + (-1) \cdot 6 + (-1) \cdot (-2) =$$
$$0 - 4 = -4$$

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6	1	8
-7	9	2
-4	-9	

$$5 \cdot 1 + (-1) \cdot 3 + (-1) \cdot 4 + (-1) \cdot 7 + (-1) \cdot 0 =$$
$$5 - 14 = -9$$

Convolutions on Images

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6	1	8
-7	9	2
-4	-9	3

$$5 \cdot 4 + (-1) \cdot 3 + (-1) \cdot 4 + (-1) \cdot 9 + (-1) \cdot 1 =$$
$$20 - 17 = 3$$

Convolutions on Images

Input

Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Mean (Box)

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Gaussian blur

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolutions on Images

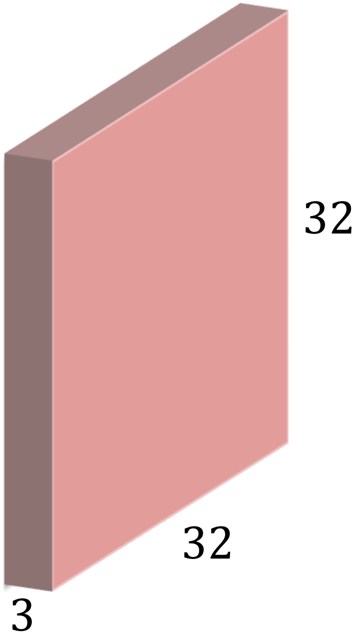
- How do we get from there to a ConvNet?
  - The idea is optimize for filter banks
  - Filters are spatially-invariant
  - Extract features at locations
  - Multiple feature banks per location (see later

Convolutions on Images

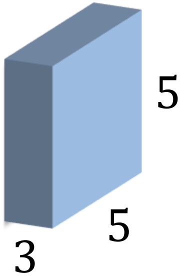
width height depth

$32 \times 32 \times \boxed{3}$ image

Depth dimension *must* match;
i.e., filter extends the full depth of the input



$5 \times 5 \times \boxed{3}$ filter



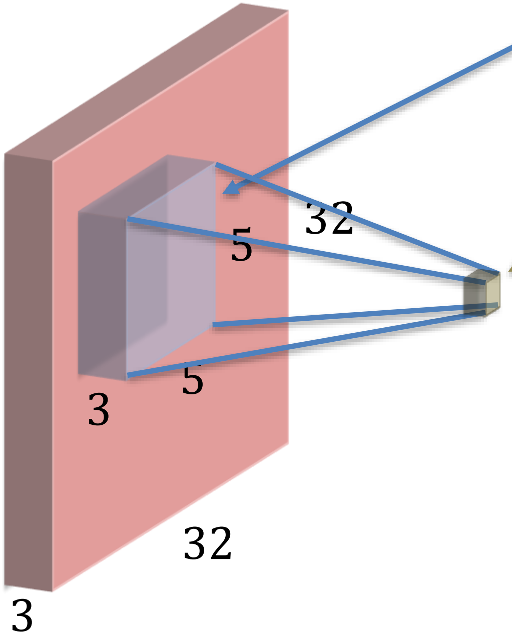
Convolve filter with image
i.e., 'slide' over it and:
-> apply filter at each location
-> dot products

Images have depth: e.g., RGB -> 3 channels

Convolutions on Images

$32 \times 32 \times 3$ image (pixels $\boldsymbol{x}$)

$5 \times 5 \times 3$ filter (weights $\boldsymbol{w}$)

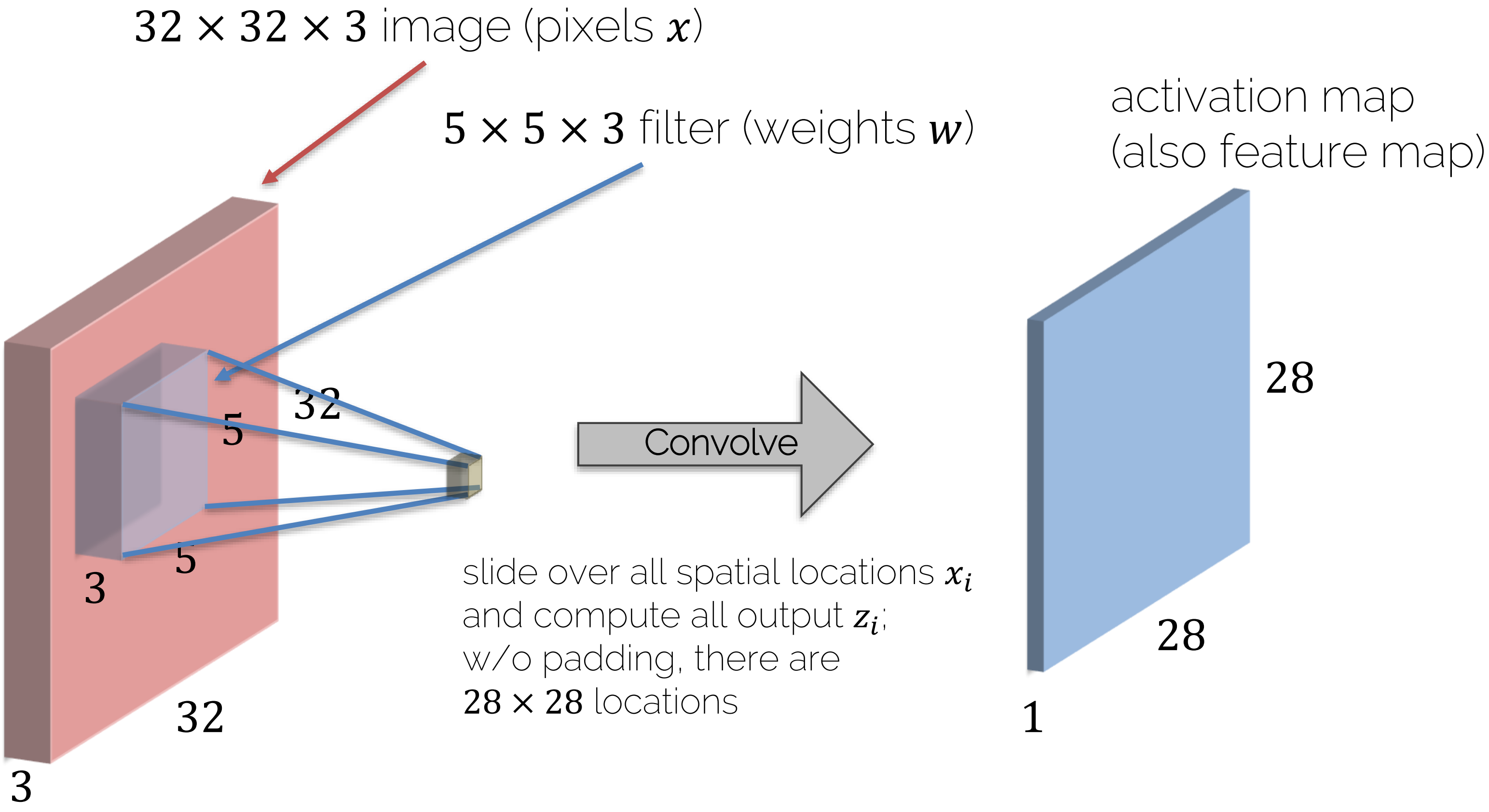


1 number:
dot product between filter weights $\boldsymbol{w}$
and $\boldsymbol{x_i - th}$ chunk of the image
Here: $5 \cdot 5 \cdot 3 = 75$-dim dot product + bias

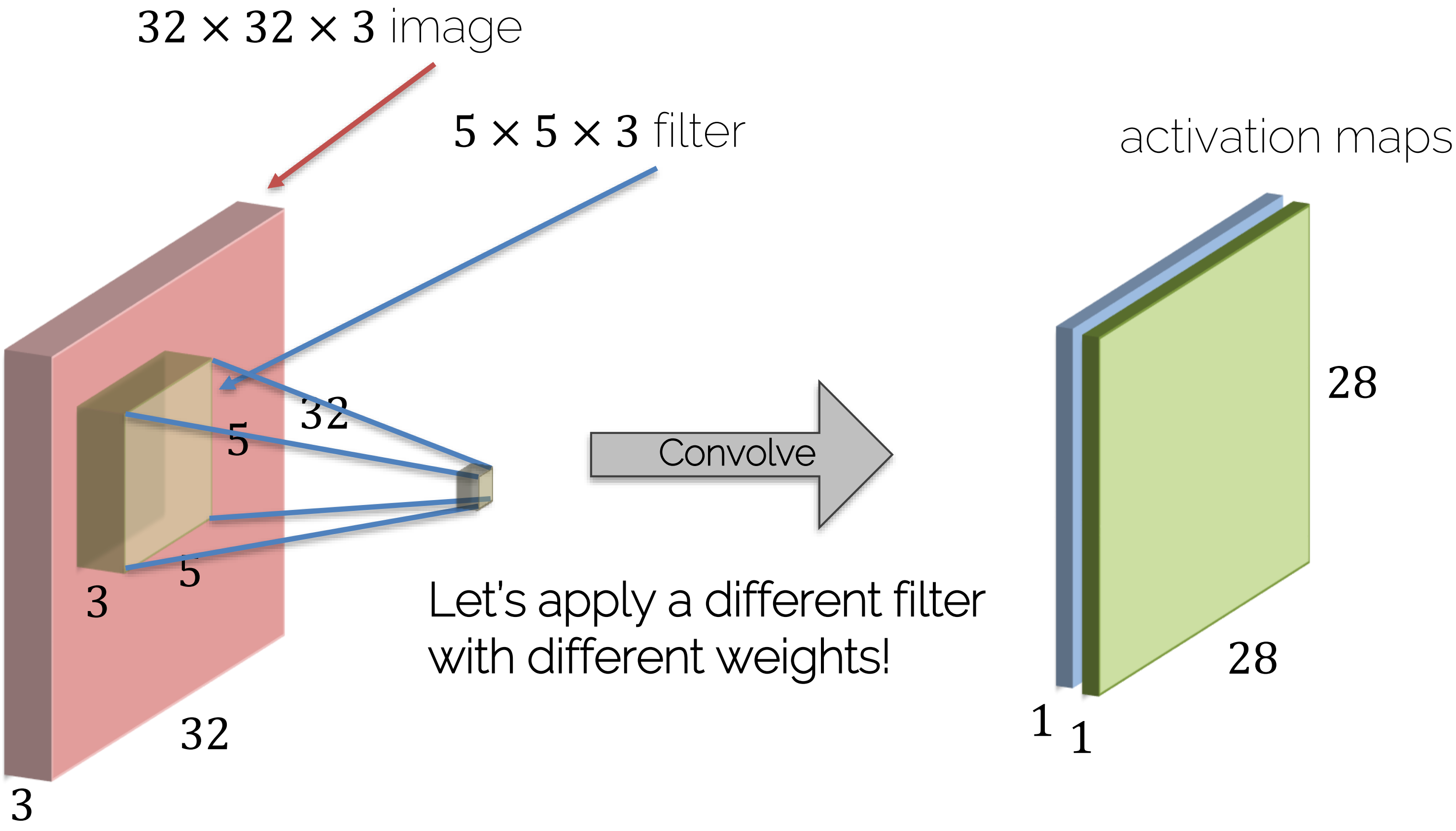
$$\boxed{z_i = w^T x_i + b}$$

$5 \times 5 \times 3$ $\quad$ $5 \times 5 \times 3$ $\quad$ 1

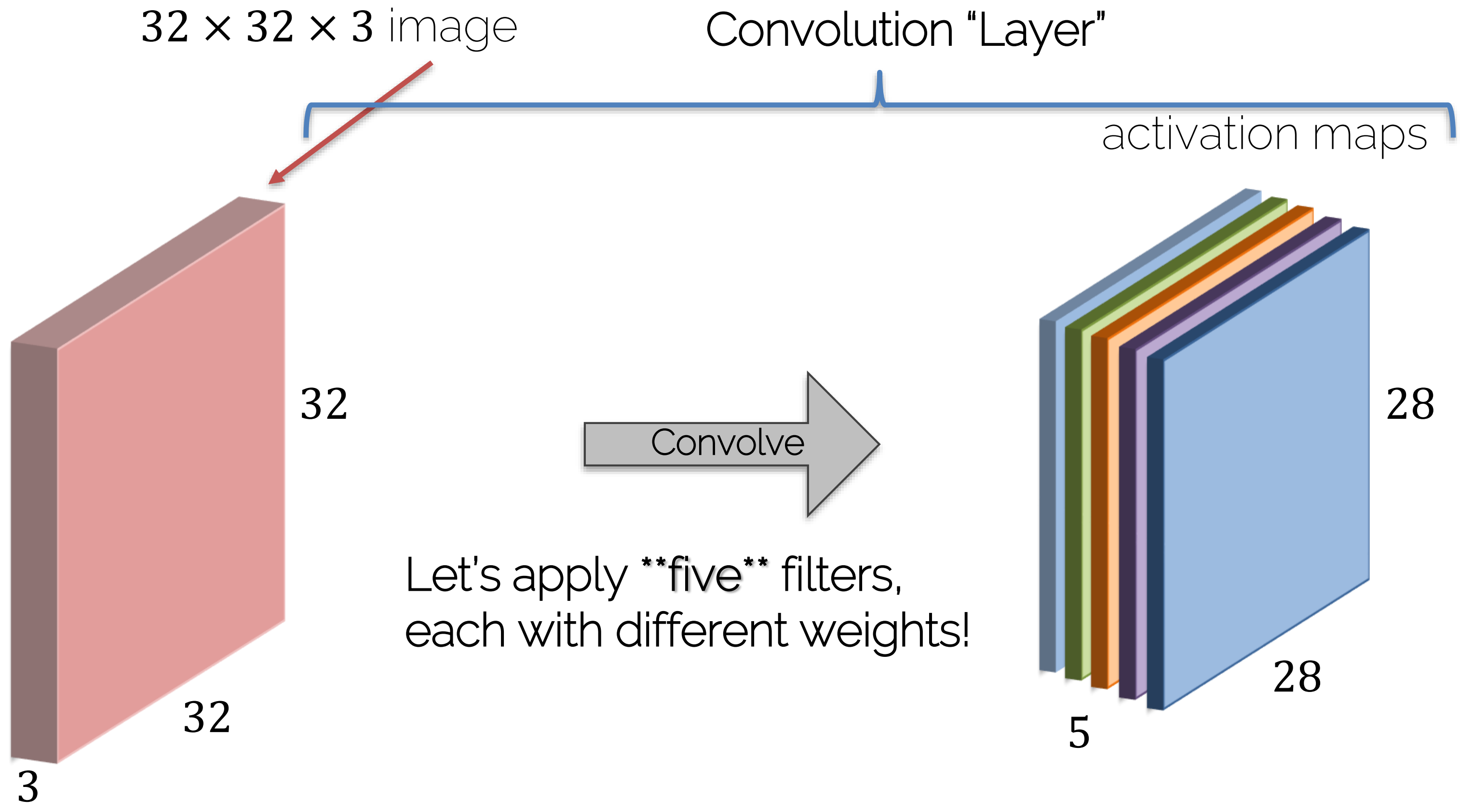
Convolutions on Images



Convolution Layer



Convolution Layer



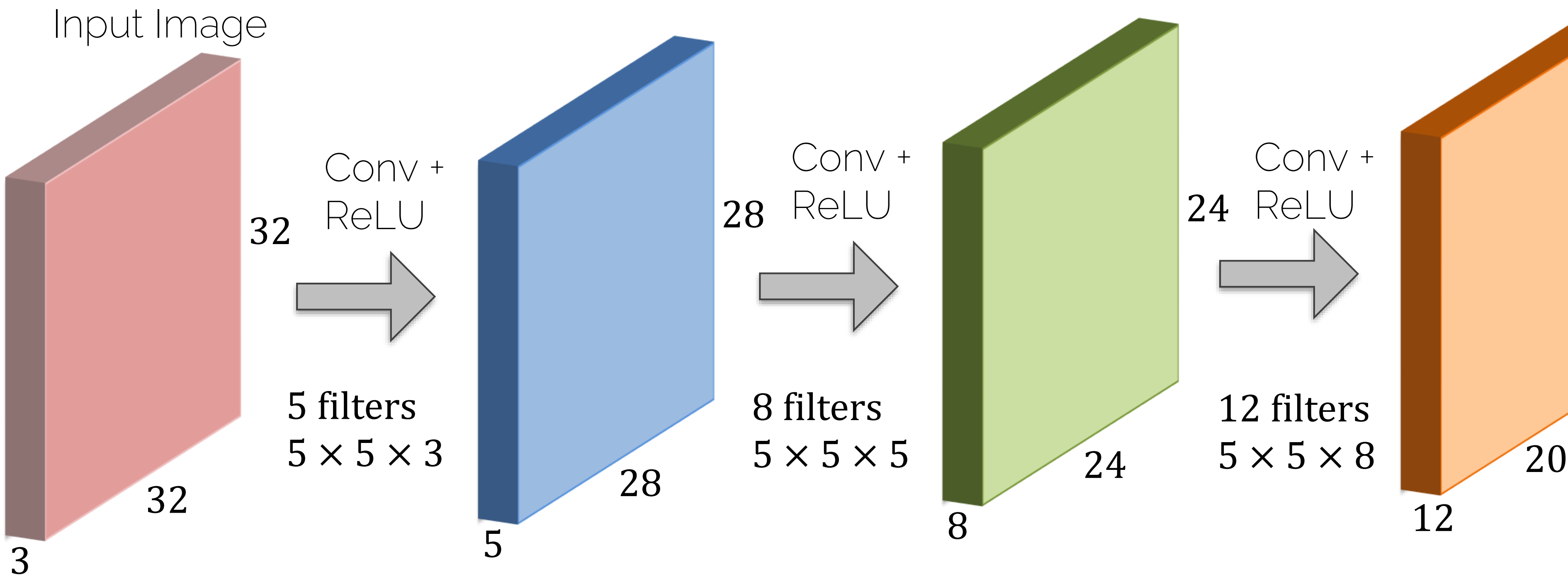
Convolution Layer

- A basic layer is defined by
  - Filter width and height (depth is implicitly given)
  - Number of different filter banks (#weight sets)

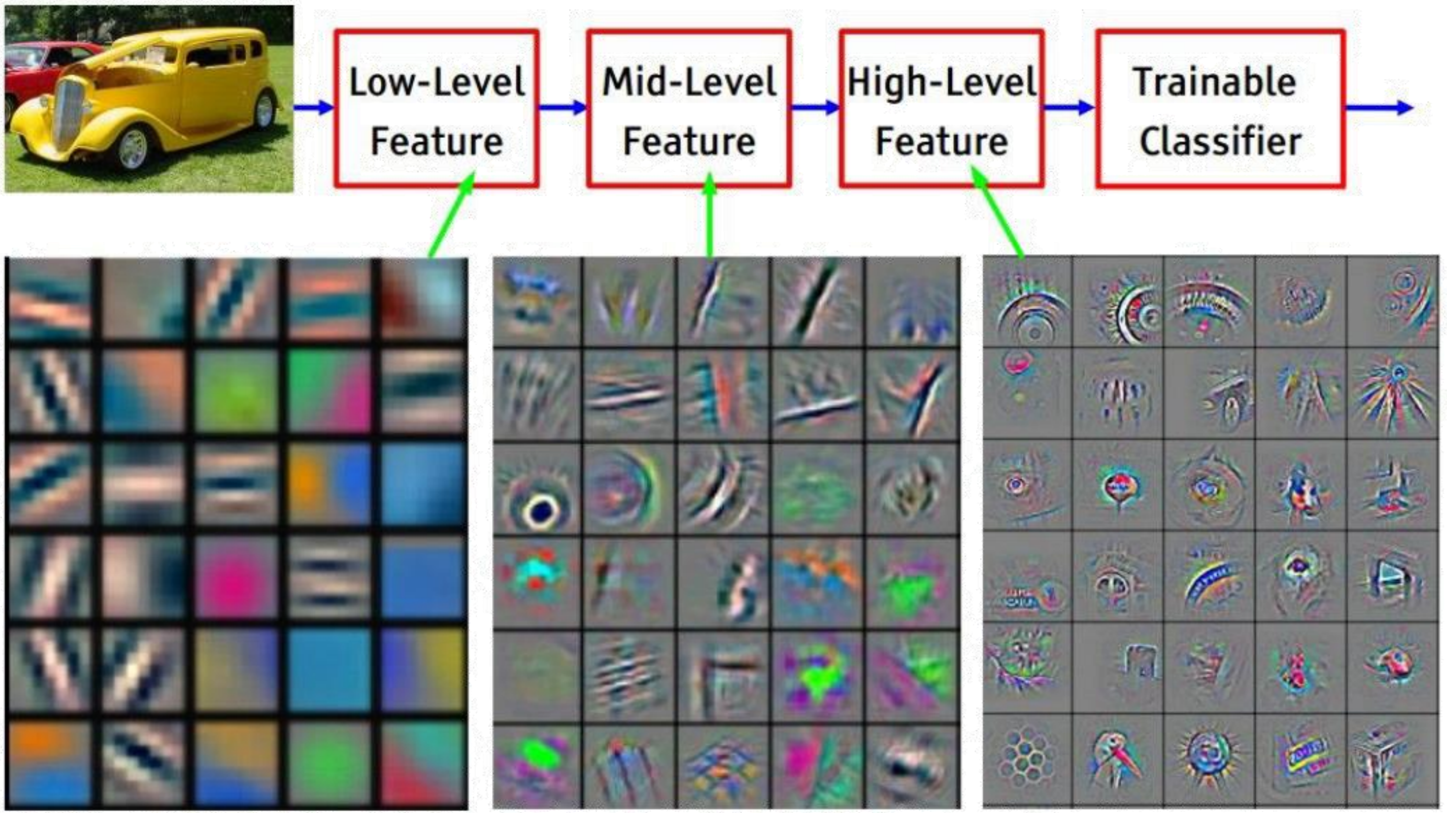
- We will also introduce stride and padding
  - Stride: specify filter locations (where?)
  - Padding: how to handle with boundaries

CNN Prototype

ConvNet is concatenation of Conv Layers and activations



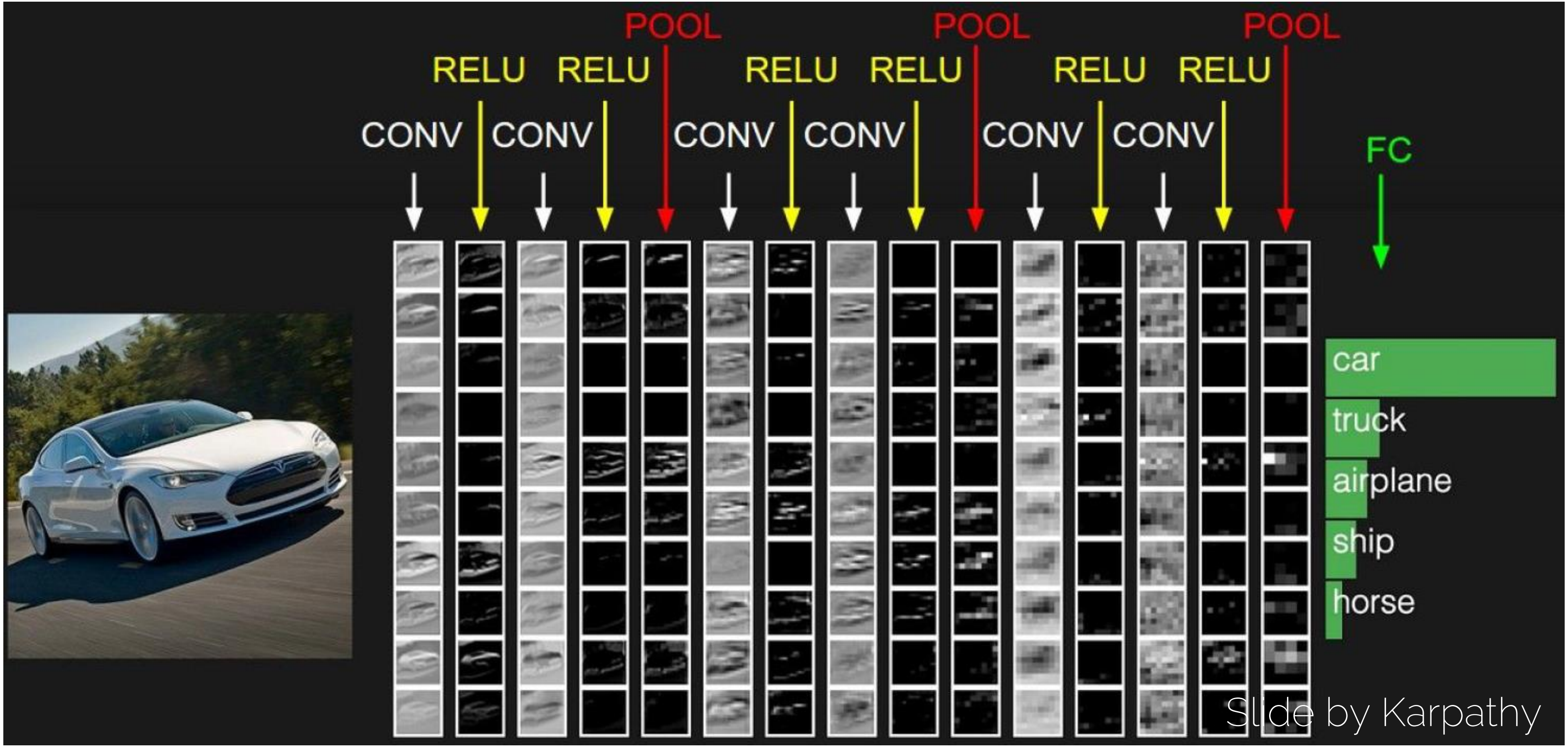
CNN Prototype



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

Slide by LeCun

CNN Prototype

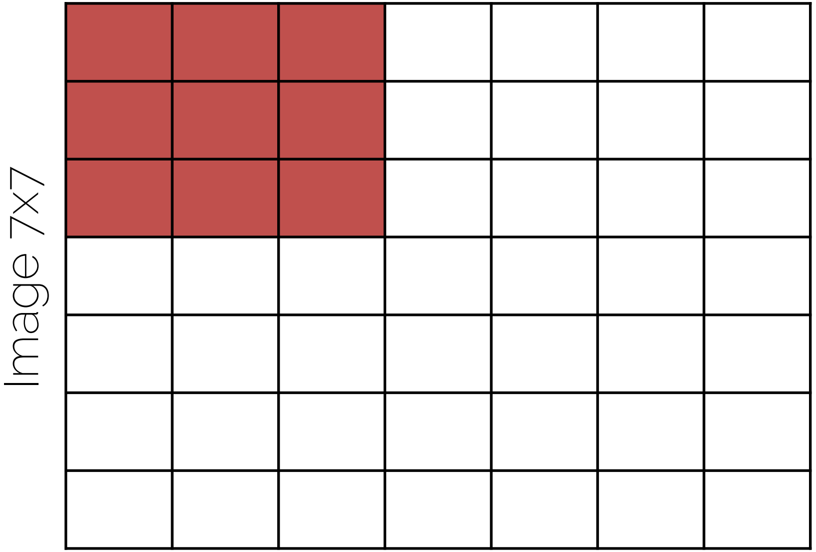


Convolution Layer

- A basic layer is defined by
  - Filter width and height (depth is implicitly given)
  - Number of different filter banks (#weight sets)

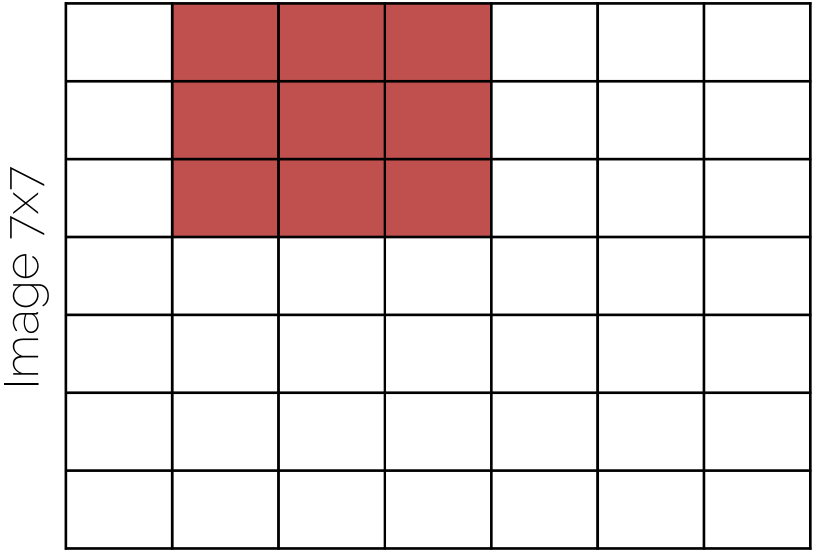
- We will also introduce stride and padding
  - Stride: specify filter locations (where?)
  - Padding: how to handle with boundaries

Convolution Layers: Dimensions



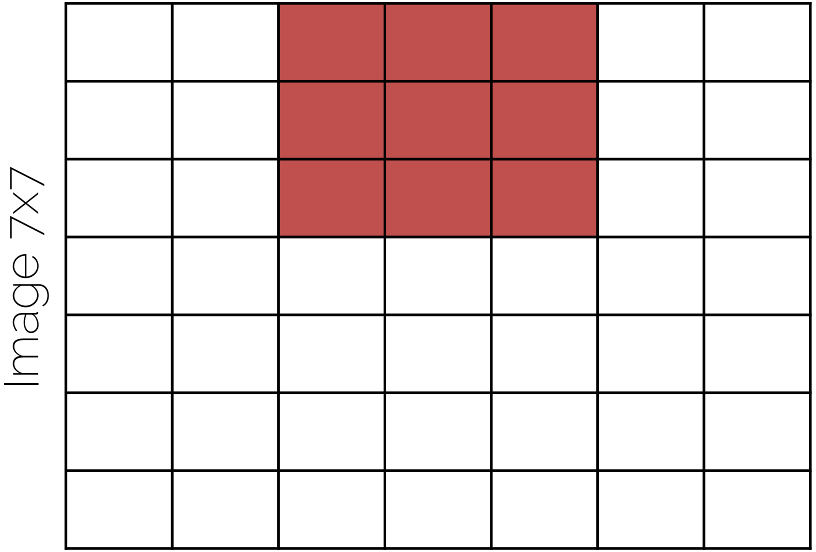
Input: 7×7
Filter: 3×3
Output: 5×5

Convolution Layers: Dimensions



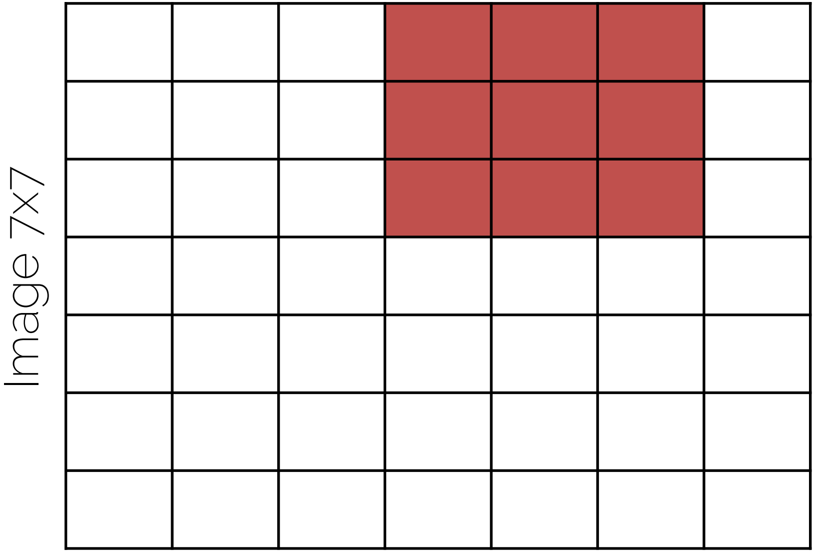
Input: 7×7
Filter: 3×3
Output: 5×5

Convolution Layers: Dimensions



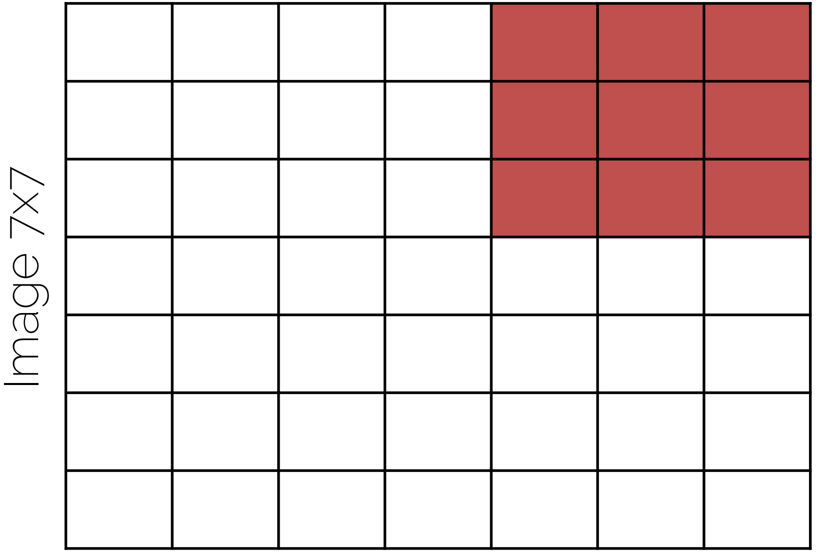
Input: 7×7
Filter: 3×3
Output: 5×5

Convolution Layers: Dimensions



Input: 7×7
Filter: 3×3
Output: 5×5

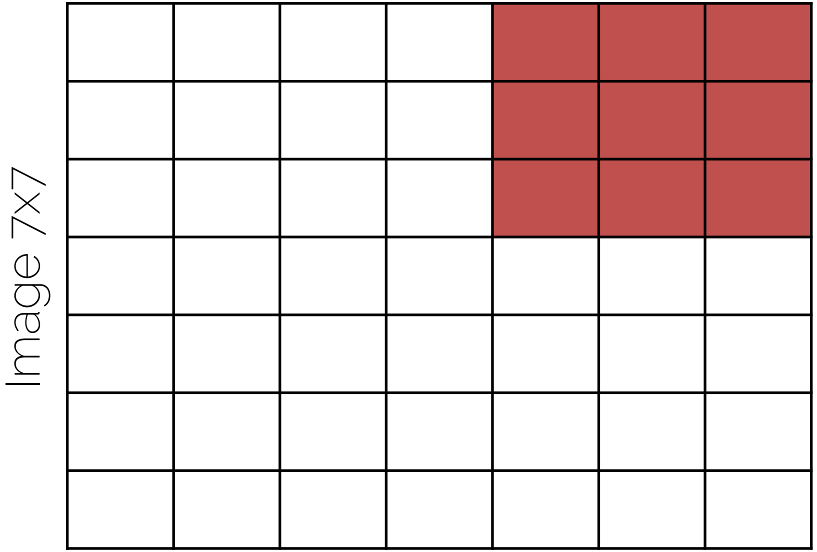
Convolution Layers: Dimensions



Input: 7×7
Filter: 3×3
Output: 5×5

Convolution Layers: Dimensions

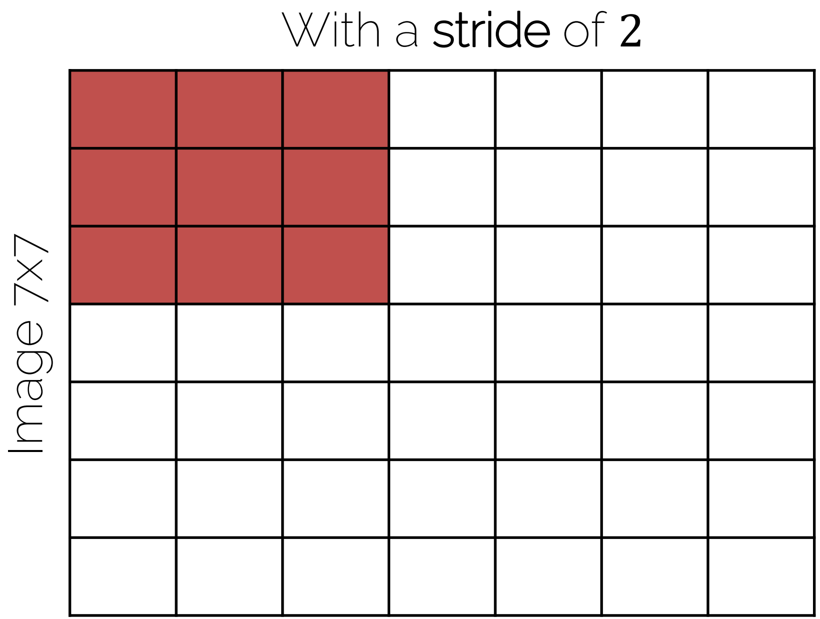
With a **stride** of 1



Input: 7×7
Filter: 3×3
Stride: 1
Output: 5×5

Stride of n: apply filter every n-th spatial location; i.e., subsample the image

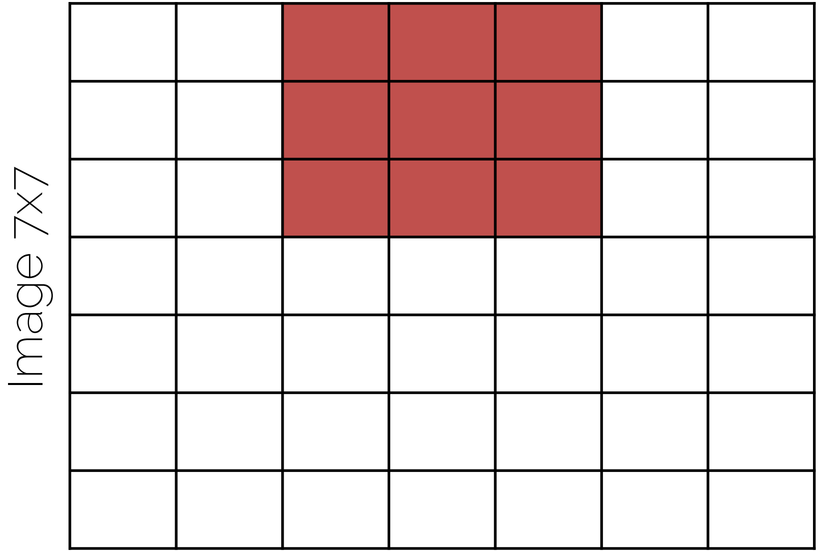
Convolution Layers: Dimensions



Input: 7×7
Filter: 3×3
Stride: 2
Output: 3×3

Convolution Layers: Dimensions

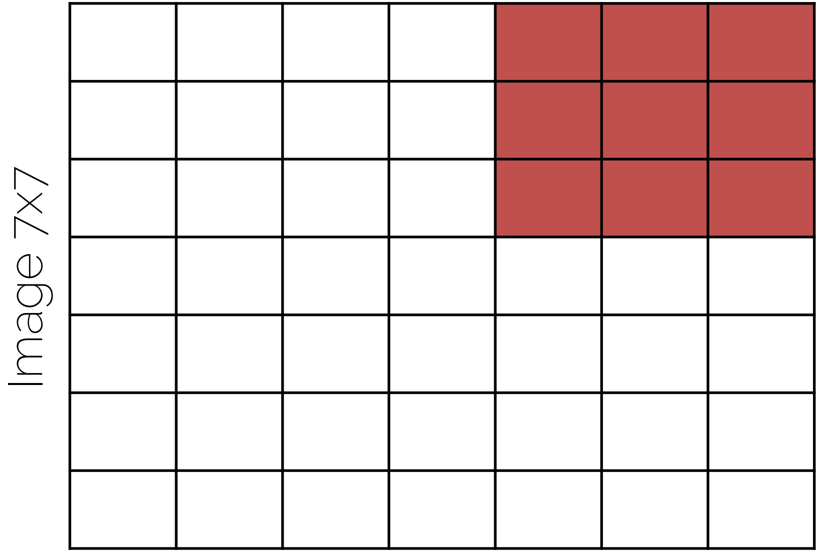
With a **stride** of **2**



Input: 7×7
Filter: 3×3
Stride: 2
Output: 3×3

Convolution Layers: Dimensions

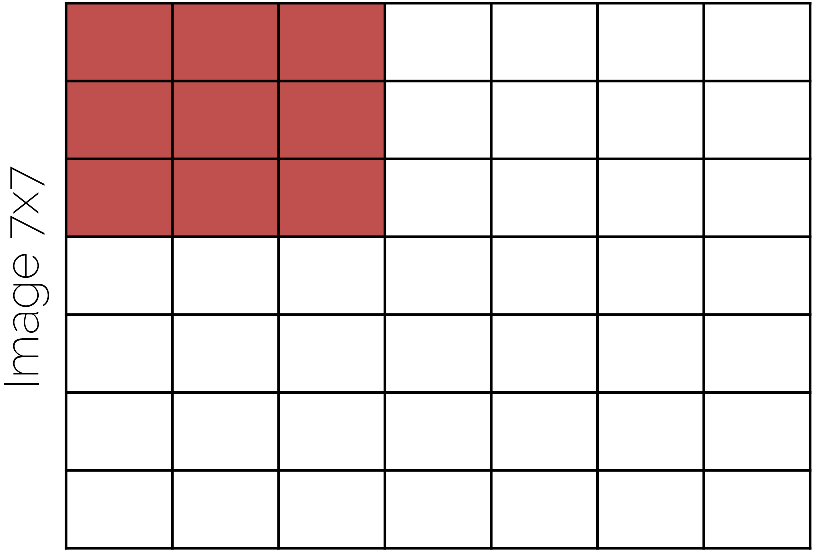
With a **stride** of **2**



Input: 7×7
Filter: 3×3
Stride: 2
Output: 3×3

Convolution Layers: Dimensions

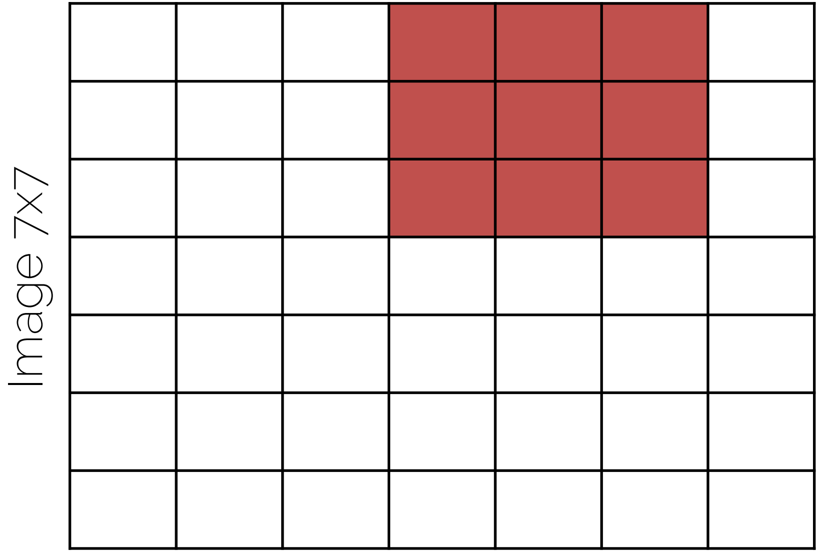
With a **stride** of **3**



Input: 7×7
Filter: 3×3
Stride: 3
Output: $?? \times ??$

Convolution Layers: Dimensions

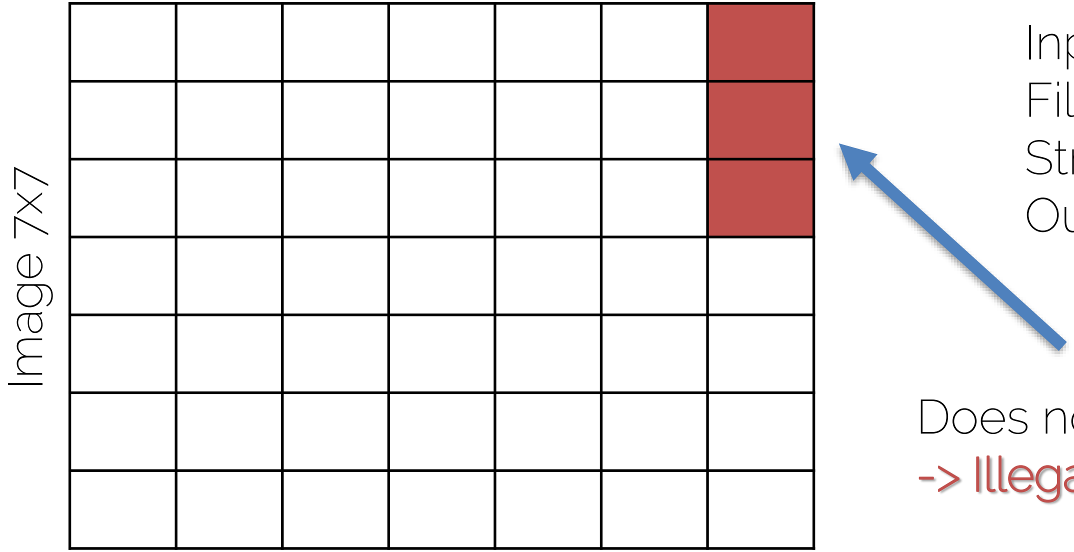
With a **stride** of 3



Input: 7×7
Filter: 3×3
Stride: 3
Output: $?? \times ??$

Convolution Layers: Dimensions

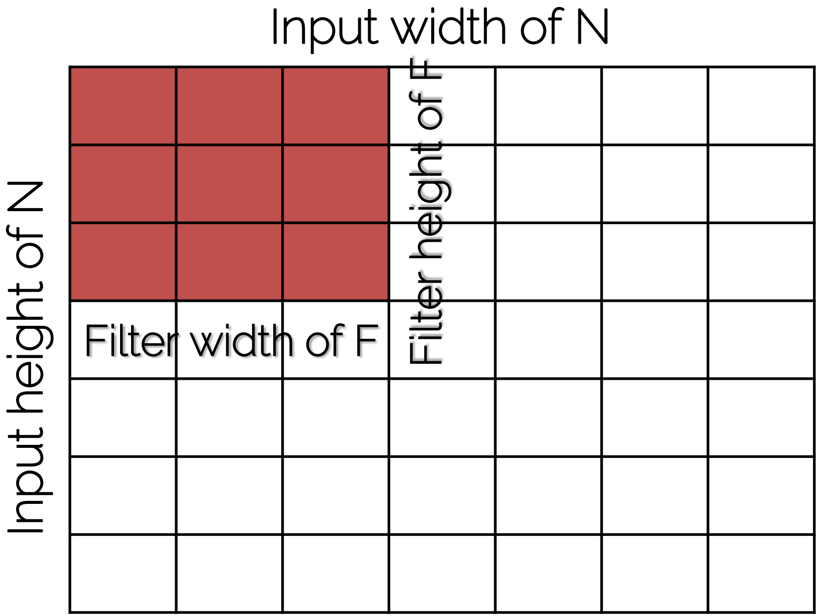
With a **stride** of **3**



Input: 7×7
Filter: 3×3
Stride: 3
Output: $?? \times ??$

Does not really fit; remainder left…
-> Illegal stride for input & filter size!

Convolution Layers: Dimensions



Input: $N \times N$
Filter: $F \times F$
Stride: S
Output: $(\frac{N-F}{S} + 1) \times (\frac{N-F}{S} + 1)$

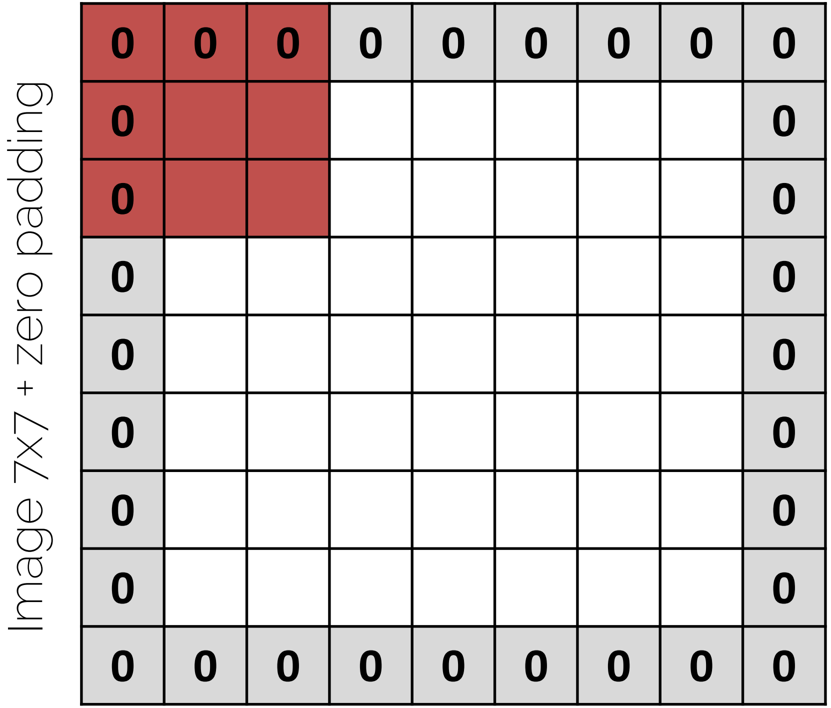
$N = 7, F = 3, S = 1$: $\frac{7-3}{1} + 1 = 5$

$N = 7, F = 3, S = 2$: $\frac{7-3}{2} + 1 = 3$

$N = 7, F = 3, S = 3$: $\frac{7-3}{3} + 1 = 2.3333$

Fractions are illegal

Convolution Layers: Dimensions



Input: 7×7
Filter: 3×3
Padding: 1
Stride: 1
Output: 7×7

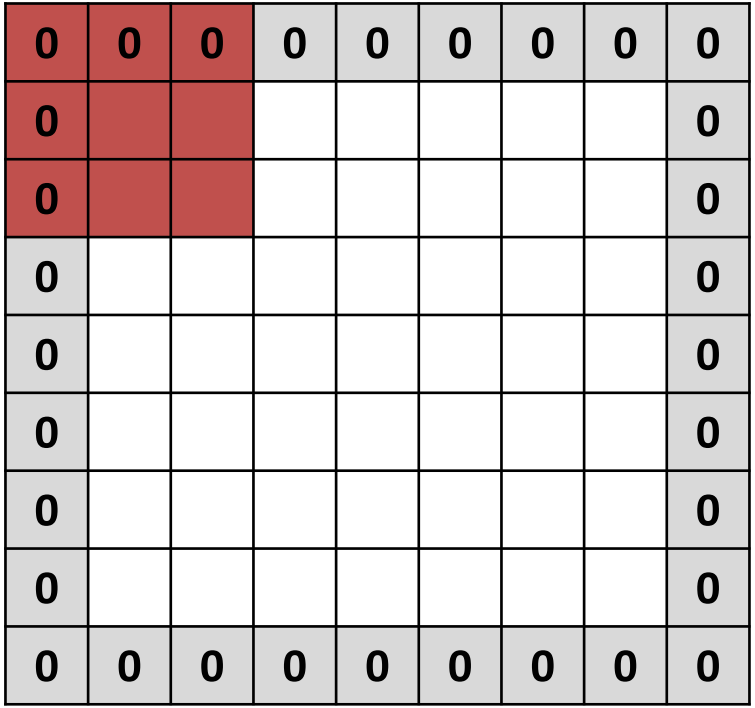
To preserve (spatial) size,
set padding to $P = \frac{F-1}{2}$

Most common is 'zero' padding

Output Size: $(\frac{N+2 \cdot P-F}{S} + 1) \times (\frac{N+2 \cdot P-F}{S} + 1)$

Convolution Layers: Dimensions

Image 7x7 + zero padding



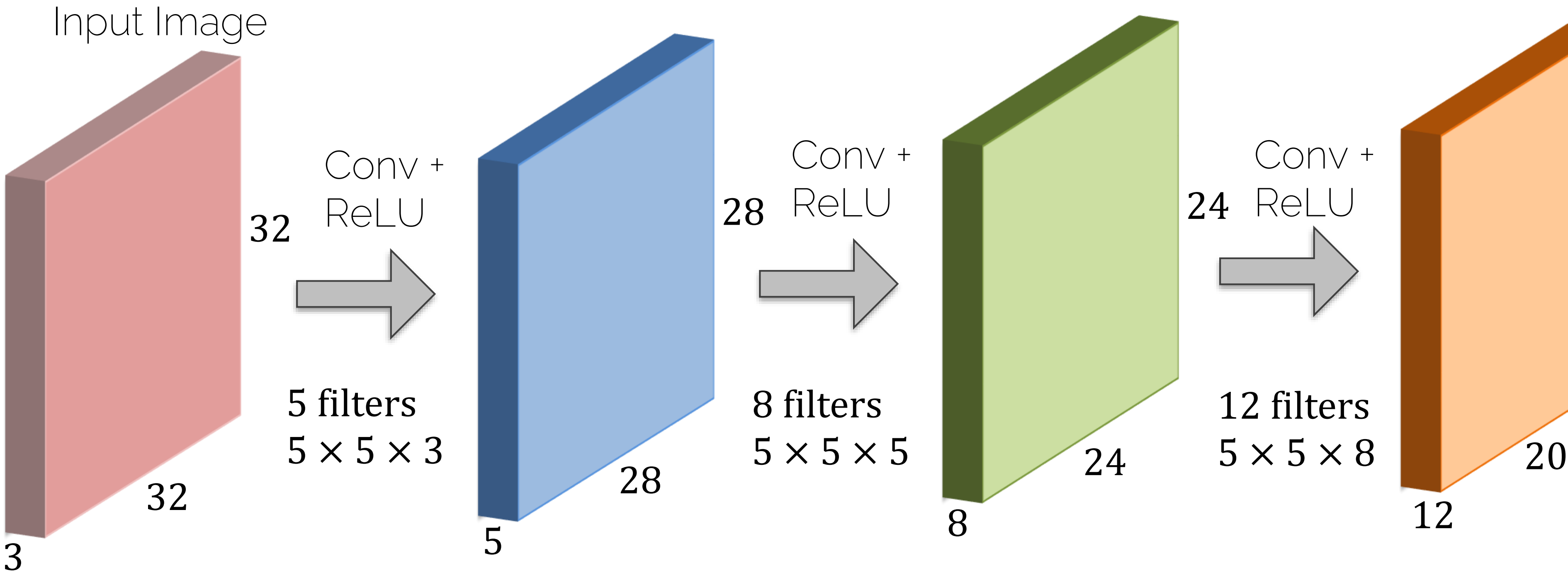
Input: 7×7
Filter: 3×3
Padding: 1
Stride: 1
Output: 7×7

To preserve (spatial) size,
set padding to $P = \frac{F-1}{2}$

Most common is ‘zero’ padding

What is the output if we set $P = 2$?

Convolution Layers: Dimensions



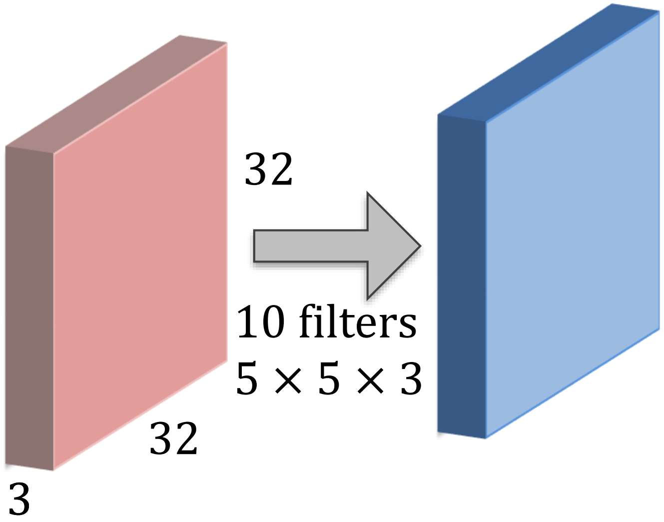
Shrinking down so quickly (32->28->24->20) is typically not a good idea…

Convolution Layers: Dimensions

Example

Input image: $32 \times 32 \times 3$
10 filters 5×5
Stride 1
Pad 2

Depth of 3 is implicitly given



Output size is:

$$\frac{32 + 2 \cdot 2 - 5}{1} + 1 = 32$$

I.e., $32 \times 32 \times 10$

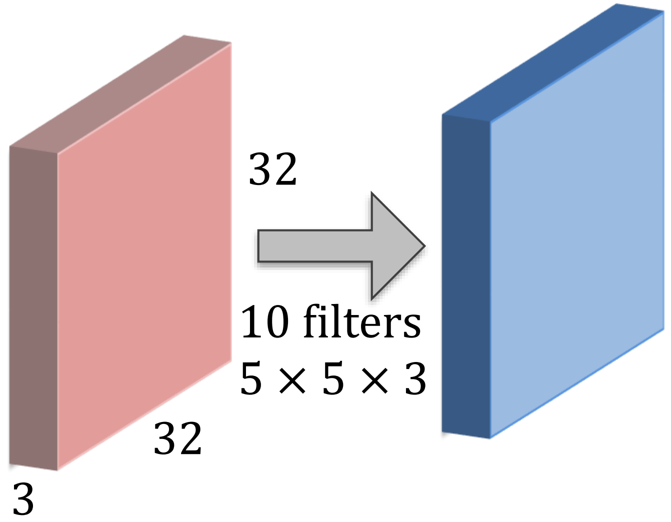
Remember

Output: $\left(\frac{N+2 \cdot P-F}{S} + 1\right) \times \left(\frac{N+2 \cdot P-F}{S} + 1\right)$

Convolution Layers: Dimensions

Example

Input image: $\boxed{32} \times 32 \times 3$
$\boxed{10}$ filters $\boxed{5} \times 5$
Stride $\boxed{1}$
Pad $\boxed{2}$



Output size is:

$$\frac{\boxed{32} + 2 \cdot \boxed{2} - \boxed{5}}{\boxed{1}} + 1 = 32$$

I.e., $32 \times 32 \times \boxed{10}$

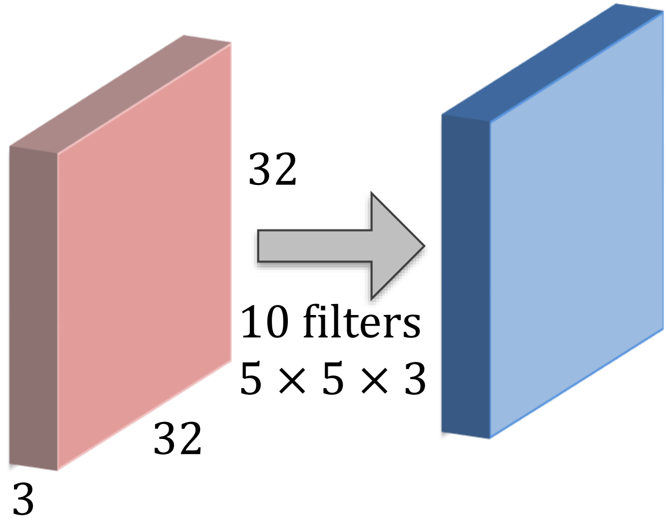
Remember

Output: $\left(\frac{N + 2 \cdot P - F}{S} + 1\right) \times \left(\frac{N + 2 \cdot P - F}{S} + 1\right)$

Convolution Layers: Dimensions

Example

Input image: $32 \times 32 \times 3$
10 filters 5×5
Stride 1
Pad 2



Number of parameters (weights):
Each filter has $5 \times 5 \times 3 + 1 = 76$ params (+1 for bias)
-> $76 \cdot 10 = 760$ params in layer

Convolution Layers: Dimensions

- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Four hyperparameters
  - Number of filters K
  - Spatial filter extent F
  - Stride S
  - Zero padding P
- Output volume is of size $W_{out} \times H_{out} \times D_{out}$
  - $W_{out} = \frac{W_{in} - F + 2 \cdot P}{S} + 1$
  - $H_{out} = \frac{H_{in} - F + 2 \cdot P}{S} + 1$
  - $D_{out} = K$

Common settings:

$K =' \text{powers of } 2', \text{e.g.}, 32, 64, 128, 512$

$F = 3,\ S = 1,\ P = 1$

$F = 5,\ S = 1,\ P = 2$

$F = 5,\ S = 2,\ P = (whatever\ fits)$

$F = 1,\ S = 1,\ P = 0$

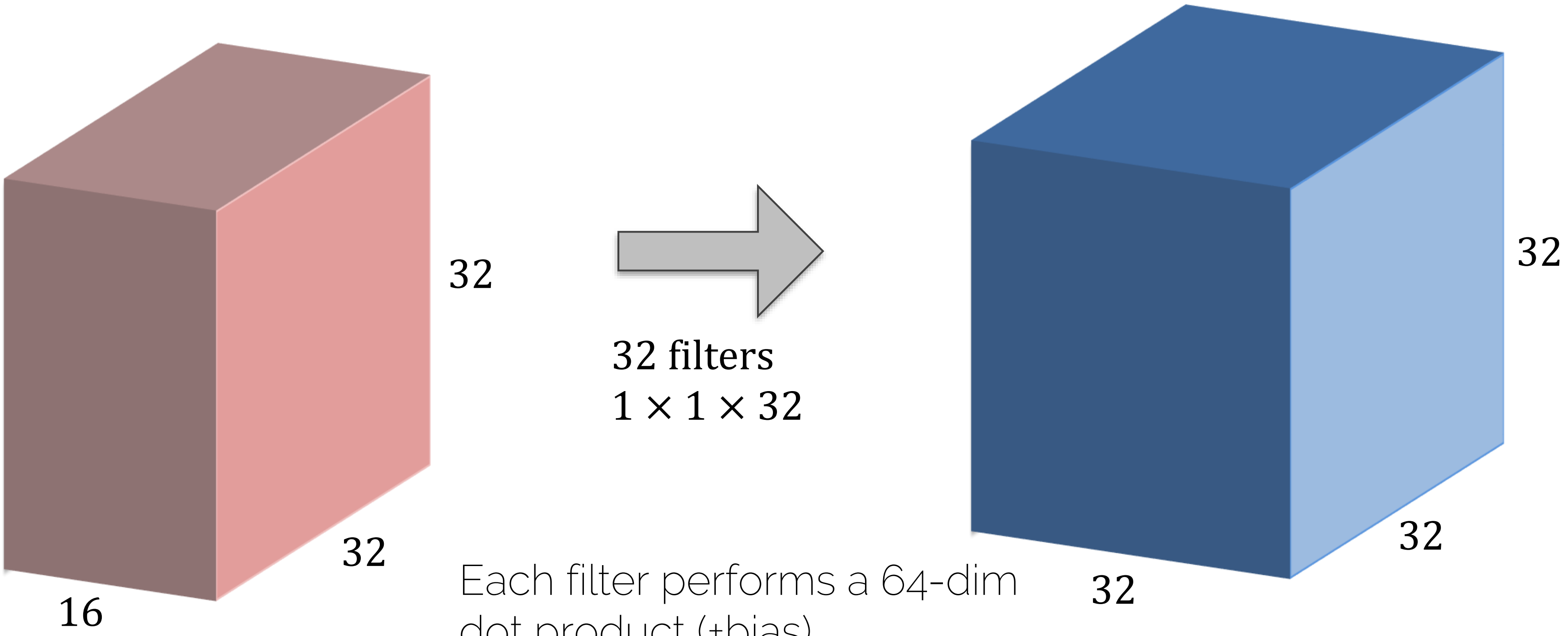
- There are $F \cdot F \cdot D_{in}$ weights per filter; i.e., a total of $(F \cdot F \cdot D_{in}) \cdot K$ weights and K biases

- In the output volume, the D-th depth slice of size $(W_{out} \times H_{out})$ is the result of the convolution of the D-th over the input volume with a stride of S, and offset by its bias

Slide by Li/Karpathy/Johnson

Convolution Layers: Dimensions

- 1×1 Convolution is actually pretty common



Each filter performs a 64-dim dot product (+bias)

Conv Layer in Torch

Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
Four hyperparameters
- Number of filters K
- Spatial filter extent F
- Stride S
- Zero padding P

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

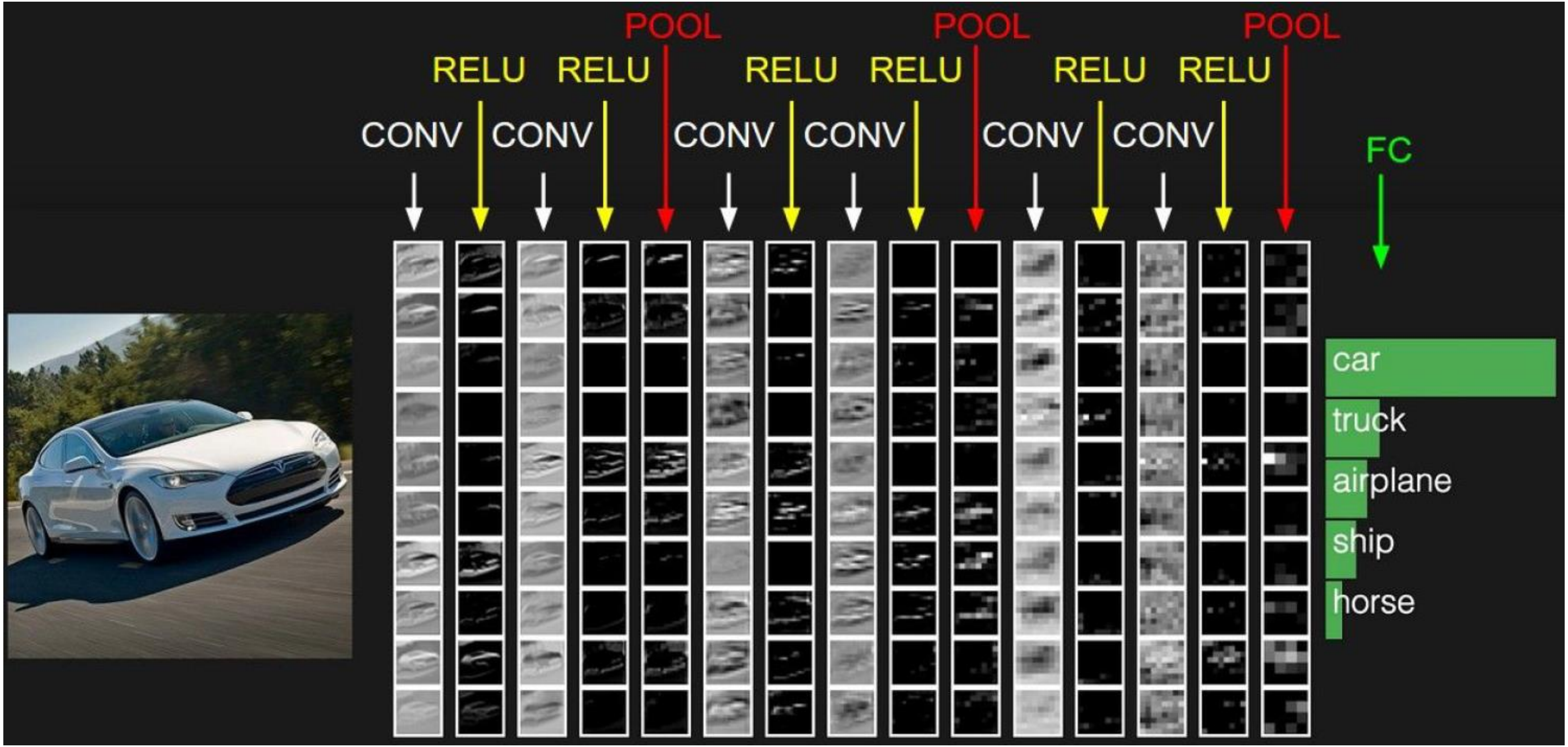
- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

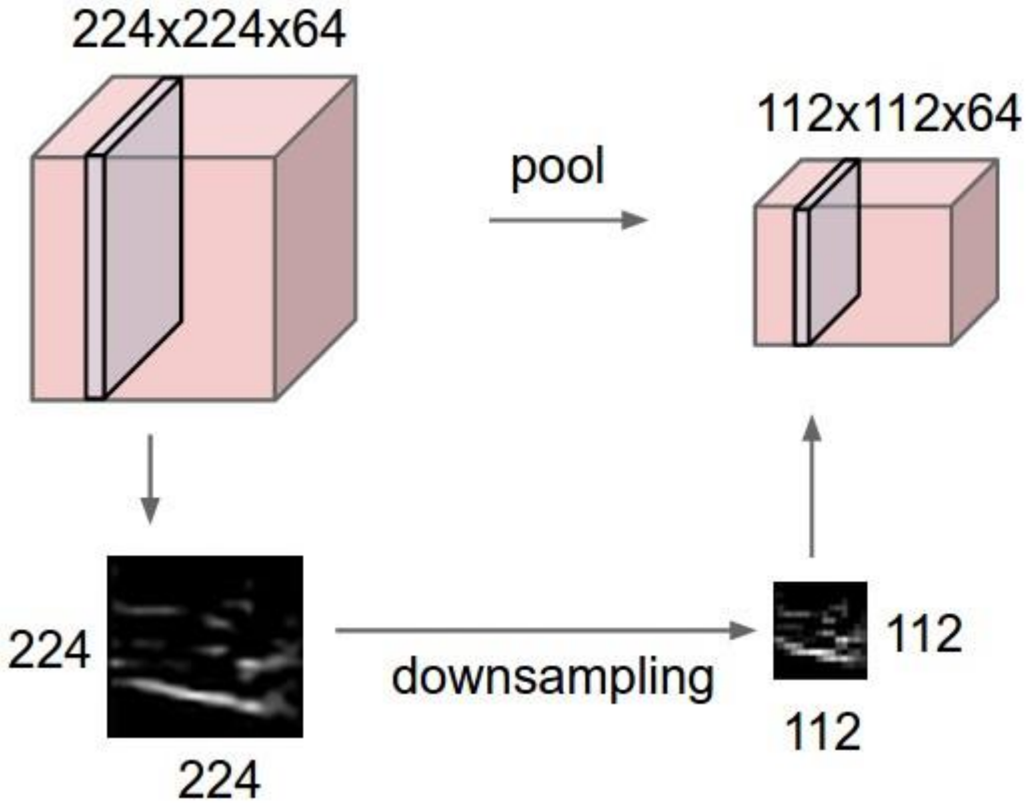
If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

Convolutional Neural Network



Pooling Layer



Slide by Li/Karpathy/Johnson

Pooling Layer: Max Pooling

Single depth slice of input

3	1	3	5
6	0	7	9
3	2	1	4
0	2	4	3

Max pool with
2×2 filters and stride 2

'Pooled' output

6	9
3	4

Pooling Layer

- Conv Layer = 'Feature Extraction'
  - Computes a feature in a given region
- Pooling Layer = 'Feature Selection'
  - Picks the strongest activation in a region

Pooling Layer

- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Four hyperparameters
  - Spatial filter extent $\boldsymbol{F}$
  - Stride $\boldsymbol{S}$

Filter count and padding make no sense here

- Output volume is of size $W_{out} \times H_{out} \times D_{out}$
  - $W_{out} = \frac{W_{in} - F}{S} + 1$
  - $H_{out} = \frac{H_{in} - F}{S} + 1$
  - $D_{out} = D_{in}$
- Does not contain parameters; e.g., its fixed function

Pooling Layer

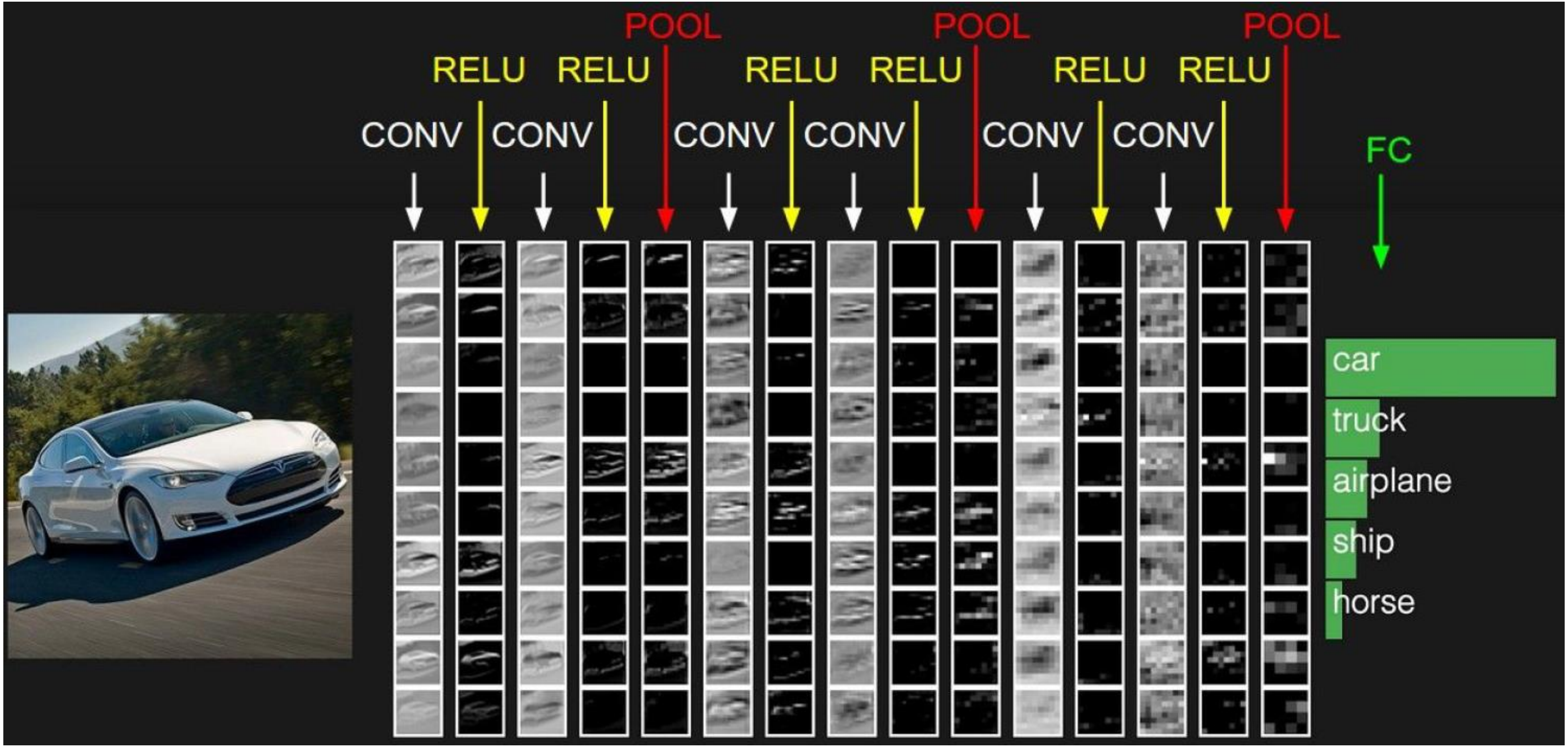
- Input is a volume of size $W_{in} \times H_{in} \times D_{in}$
- Four hyperparameters
  - Spatial filter extent F
  - Stride S
- Output volume is of size $W_{out} \times H_{out} \times D_{out}$
  - $W_{out} = \frac{W_{in} - F}{S} + 1$
  - $H_{out} = \frac{H_{in} - F}{S} + 1$
  - $D_{out} = D_{in}$
- Does not contain parameters; e.g., its fixed function

Common settings:

$F = 2, S = 2$

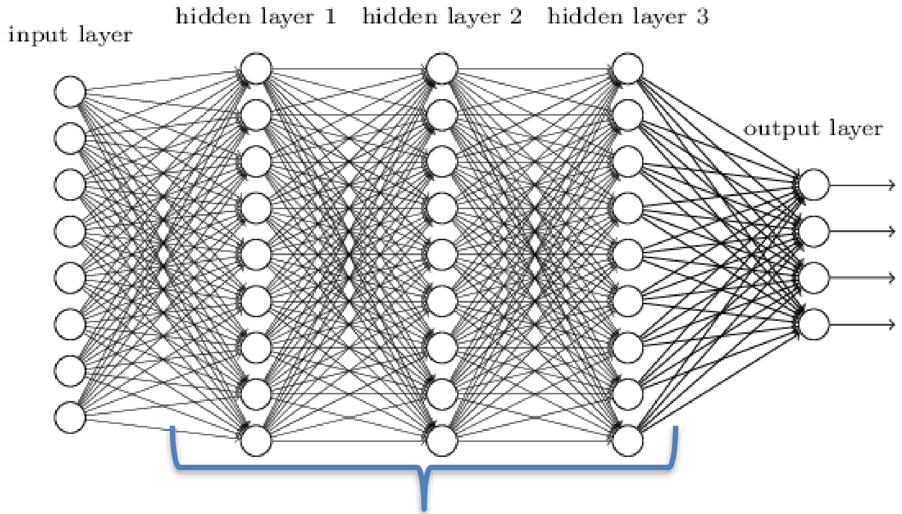
$F = 3, S = 2$

Convolutional Neural Network



Fully-Connected Layer (FC)

- Same as what we had in 'ordinary' Neural Networks
  - i.e., set of hidden layers
  - Brute-force connections (everything with everything)



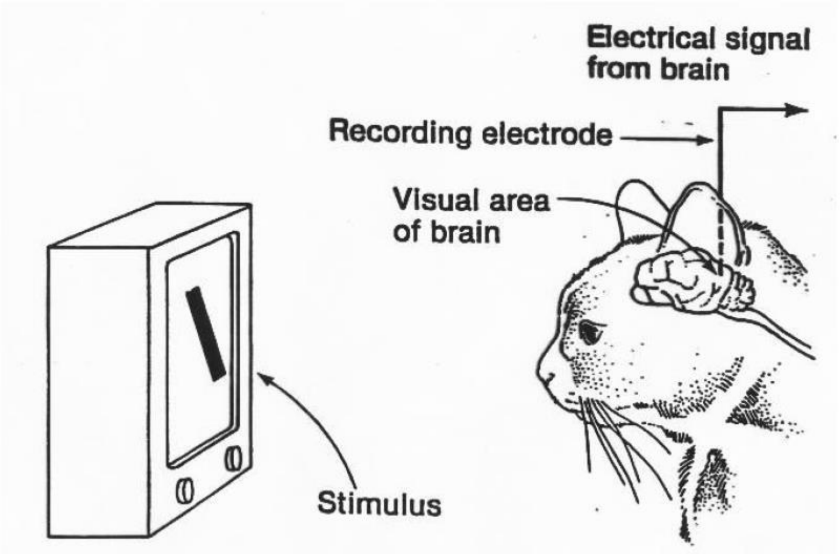
3 fully-connected layers

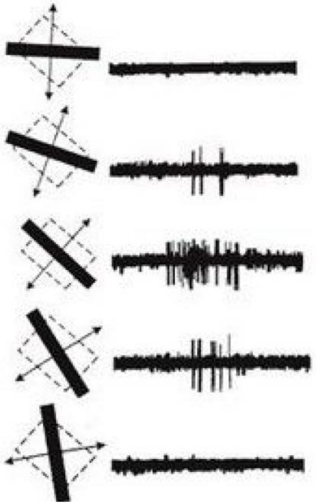
Convolutions vs Fully-Connected

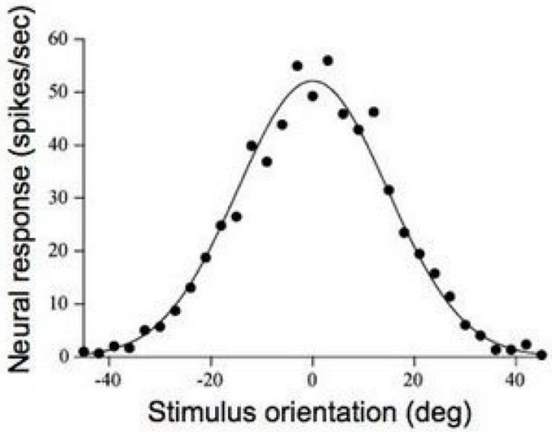
- In contrast to fully-connected layers, we want to restrict the degrees of freedom
  - FC is somewhat brute force
  - Convolutions are structured
- Sliding window to with the same filter parameters to extract image features
  - Concept of weight sharing
  - Extract same features independent of location

Convolutional Neural Network

- Turns out that CNNs are similar to the visual cortex:





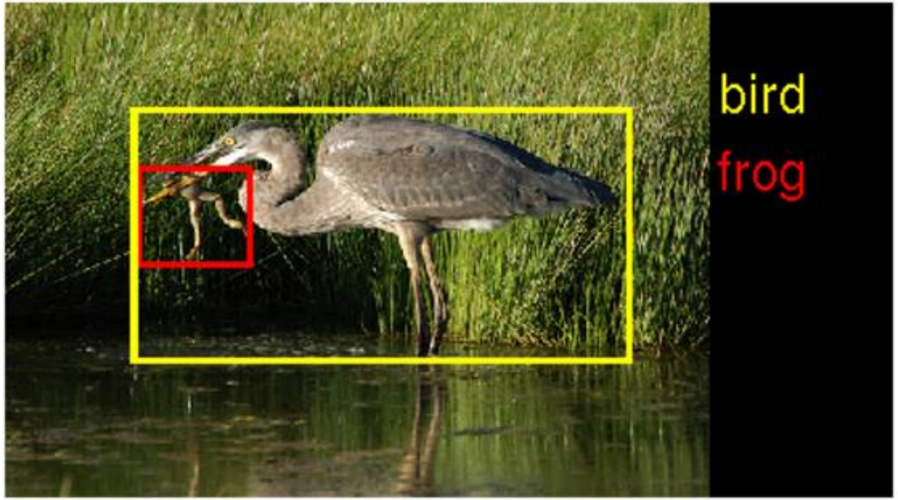


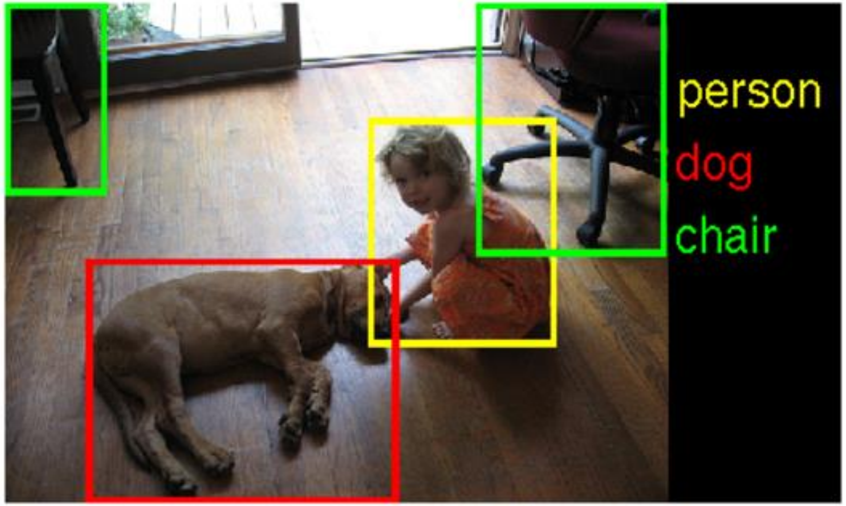
[Hubel & Wiesel, 59, 62, 68, ...]

Test Benchmarks

- Image Net Dataset:

  ImageNet Large Scale Visual Recognition Competition (ILSVRC)

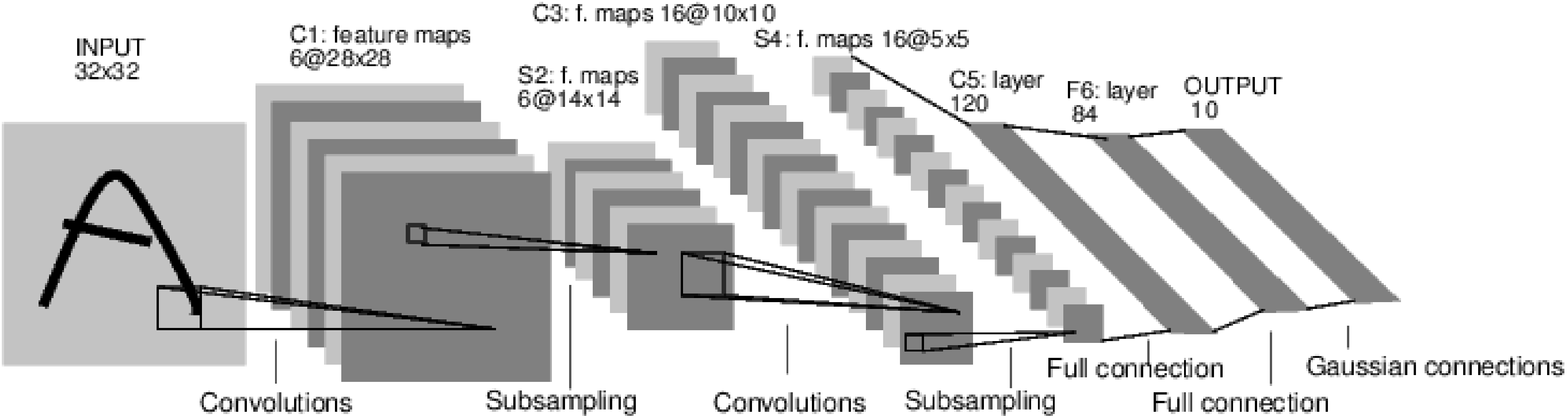




Russakovsky et al. (out of FeiFei Li's lab)

CNN Architectures: LeNet-5

[LeCun et al. 1998]

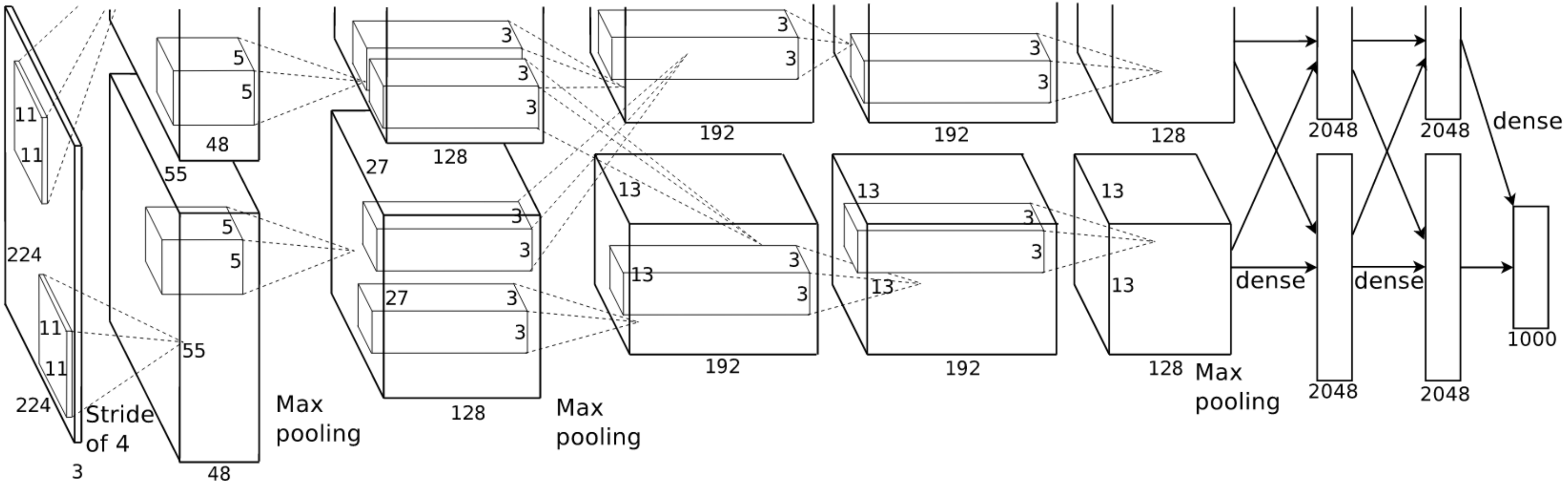


Conv filters of 5×5, stride of 1
Subsampling (i.e., pooling) of 2×2 with stride of 2

CNN Architectures: Conv -> Pool -> Conv -> Pool -> Conv -> FC

CNN Architectures: AlexNet

[Krizhevskv et al. 2012]



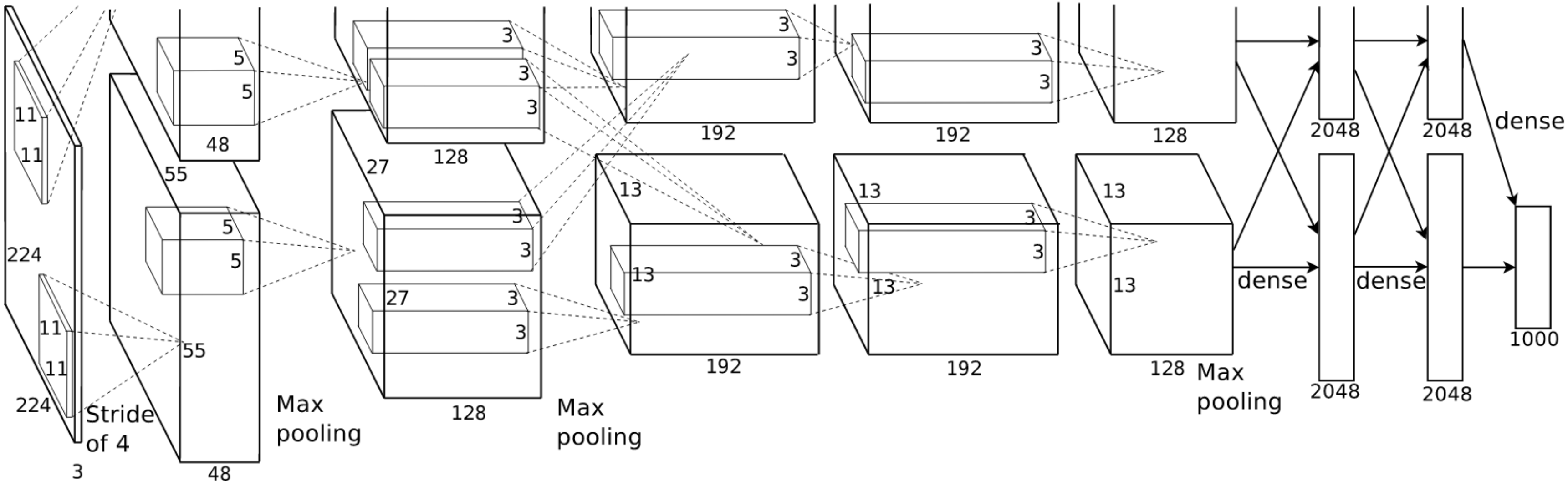
Input: $227 \times 227 \times 3$ images

Conv1 -> MaxPool1 -> Norm1 -> Conv2 -> MaxPool2 -> Norm2 ->
-> Conv3 -> Conv4 -> Conv5 -> Maxpool3 -> FC6 -> FC7 -> FC8

First use of ReLU!

CNN Architectures: AlexNet

[Krizhevskv et al. 2012]



Input: $227 \times 227 \times 3$ images

First layer:

- 96 filters of 11×11 applied at stride 4
- Output: $55 \times 55 \times 96$
- Parameters: $(11 \cdot 11 \cdot 3 + 1) \cdot 96 = 35K$

CNN Architectures: AlexNet

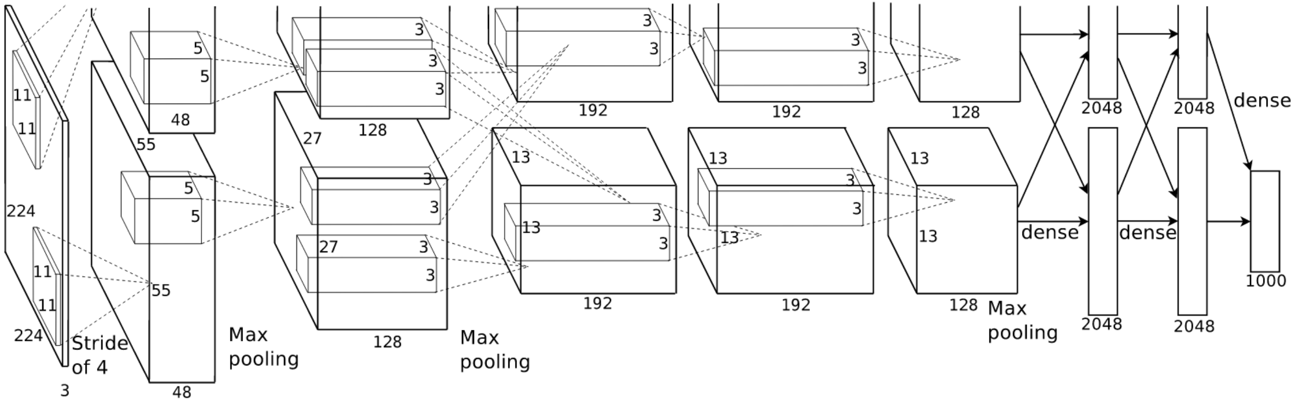
[Krizhevsky et al. 2012]

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)

- First use of ReLu
- Norm layers (not used today)
- Heavy data augmentation
- Dropout 0.5
- Batch size of 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 when accuracy plateaus
- L2 weight decay 5e-4

CNN Architectures AlexNet

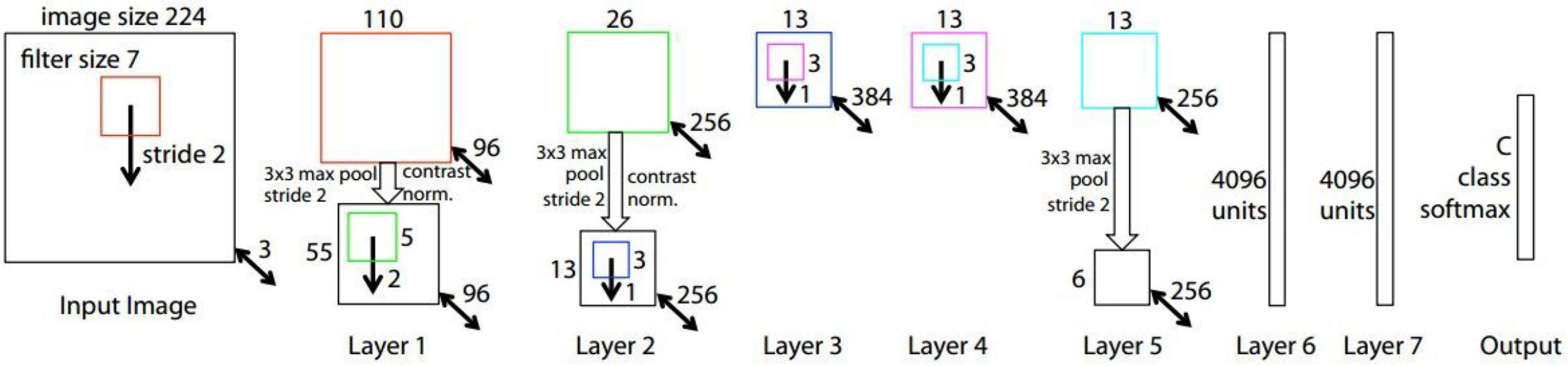
[Krizhevsky et al. 2012]



7CNN Ensemble
ImageNet top 5 error: 18.2% -> 15.4%

CNN Architectures: ZFNet

[Zeiler and Fergus 2013]



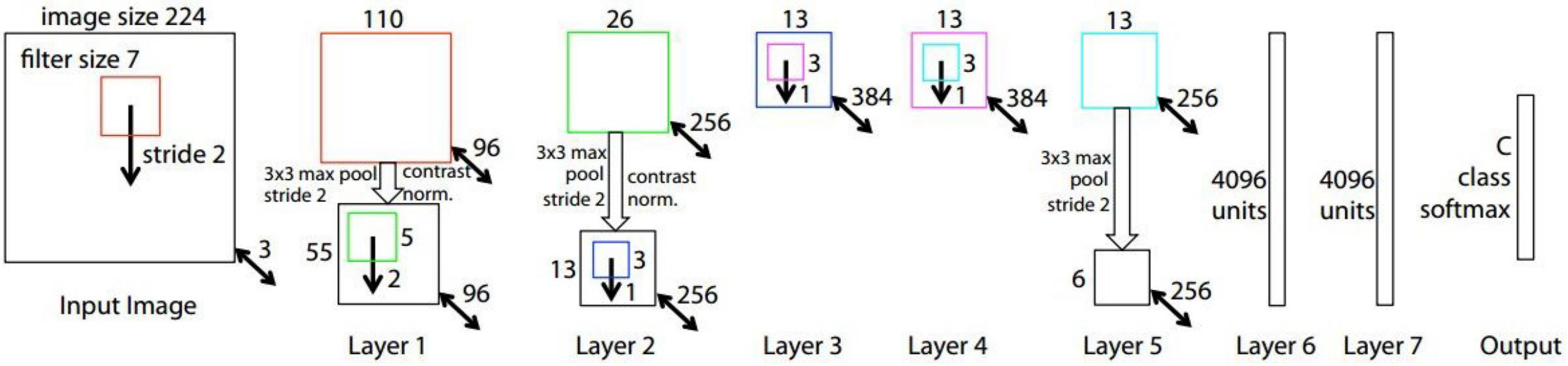
Similar to AlexNet

Conv1: 11×11 stride of 4 changed to 7×7 stride of 2

Conv3,4,5: instead of $384, 384, 256$ filters use $512, 1024, 512$

CNN Architectures: ZFNet

[Zeiler and Fergus 2013]



Ensemble

ImageNet top 5 error: 15.4% -> 14.8%

CNN Architectures: VGGNet

[Simonyan and Zisserman 2014]

Analyze different architectures!

Best model:

Ensemble
ImageNet top 5 error: 11.2% -> 7.3%

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

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

CNN Architectures: VGGNet

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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

Note:

Most memory is in early CONV

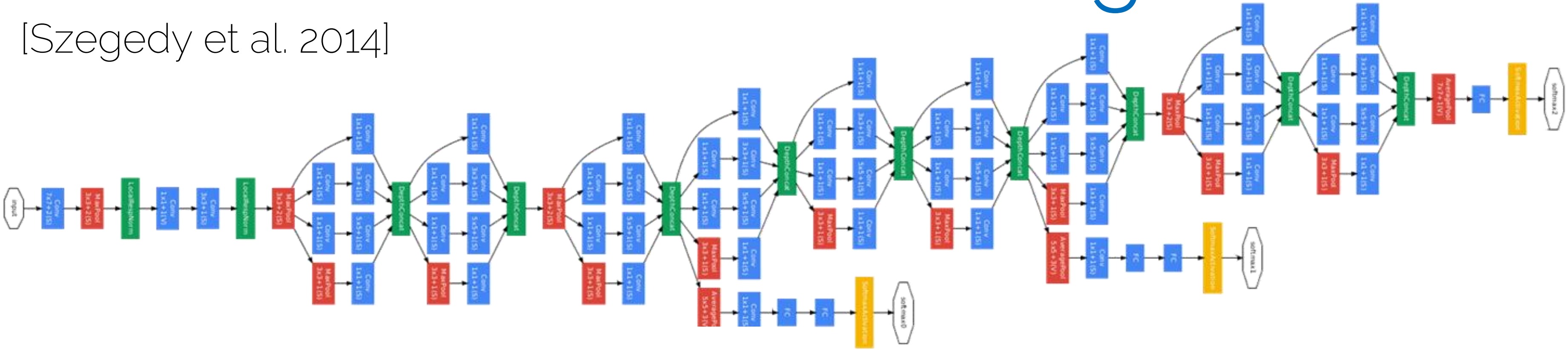
Most params are in late FC

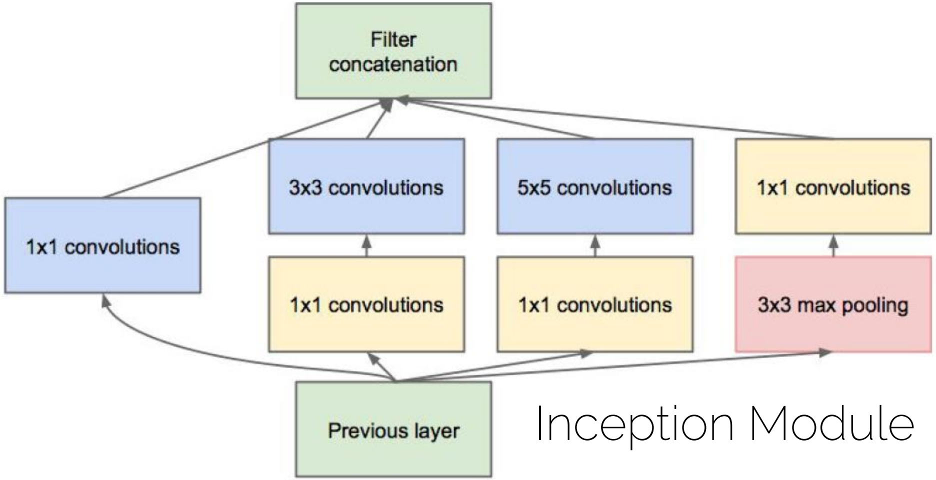
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Slide by Li/Karpathy/Johnson

CNN Architectures: GoogLeNet

[Szegedy et al. 2014]





Inception Module

22 Layers + Inception module

Ensemble
ImageNet top 5 error: 6.7%

CNN Architectures: GoogLeNet

[Szegedy et al. 2014]

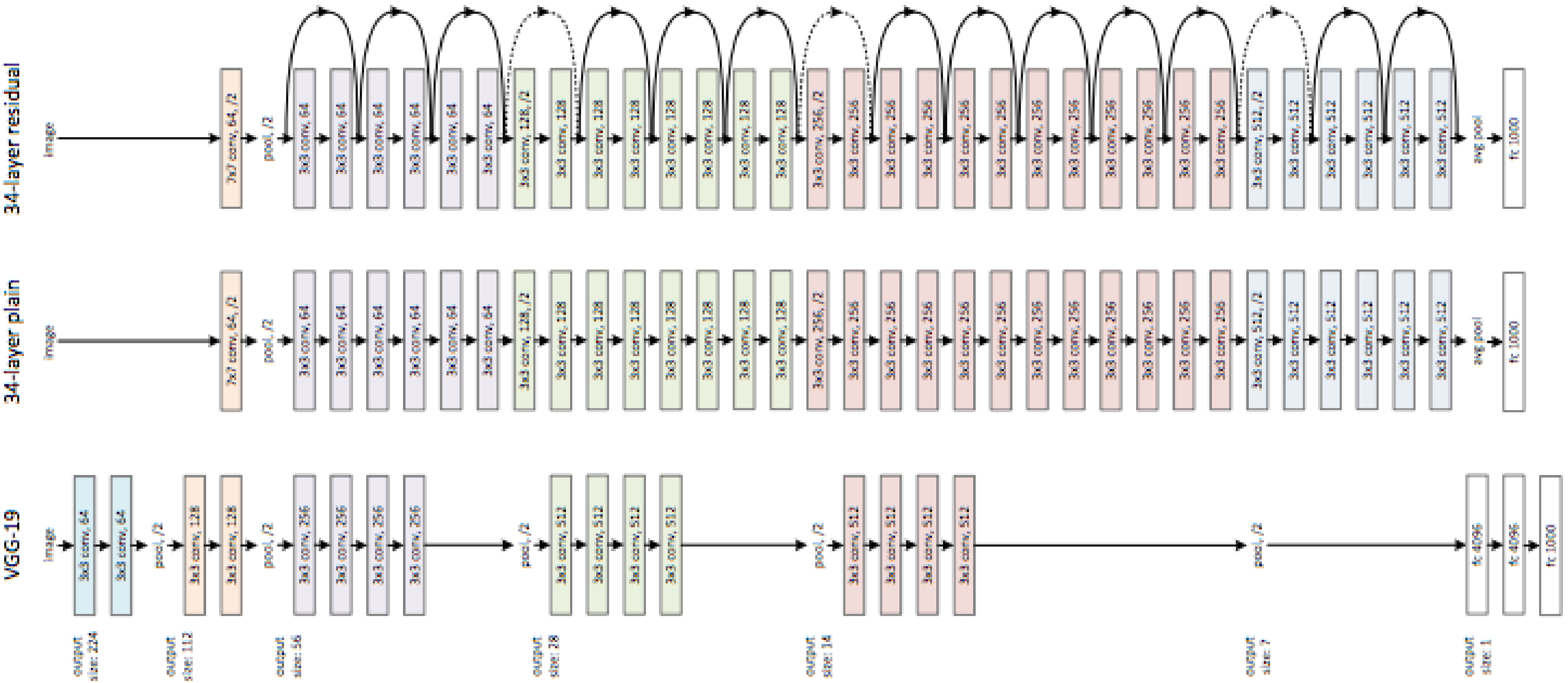
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Only 5 mio params!
No FC Layers

About 12x less param than AlexNet; 2x more compute
6.7% vs 16.4%

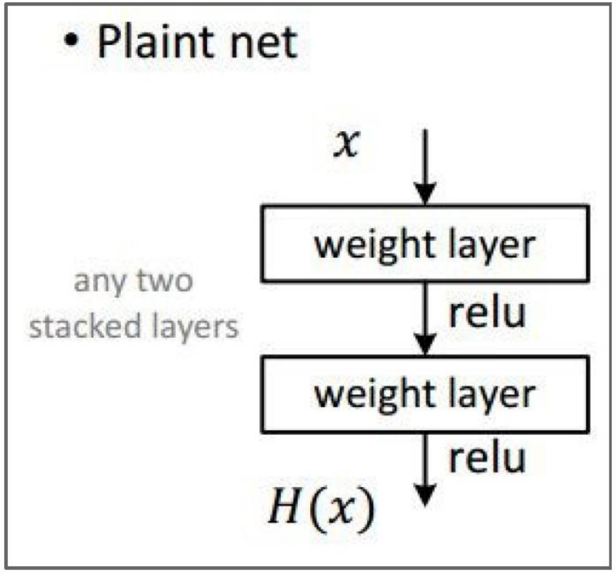
CNN Architectures: ResNet

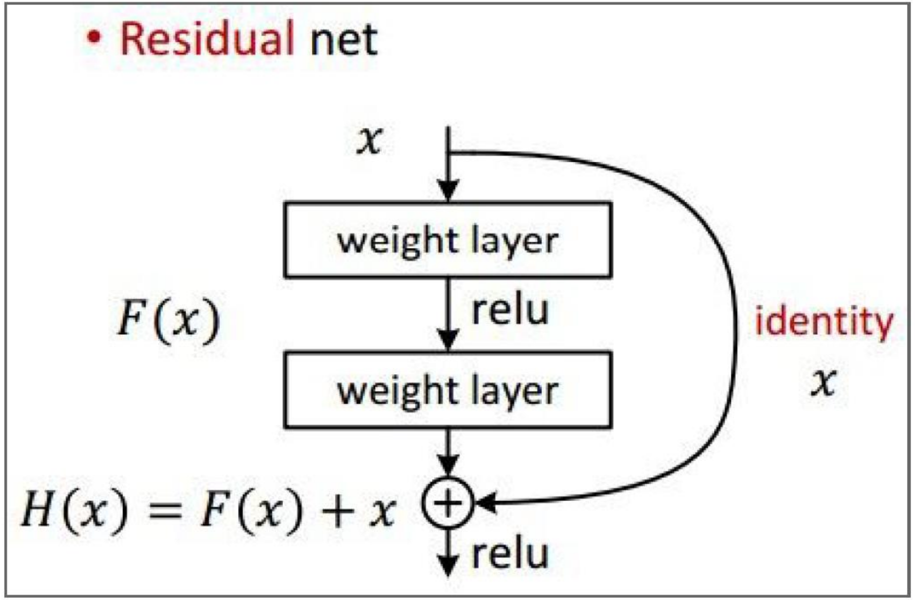
[He et al. 2015]



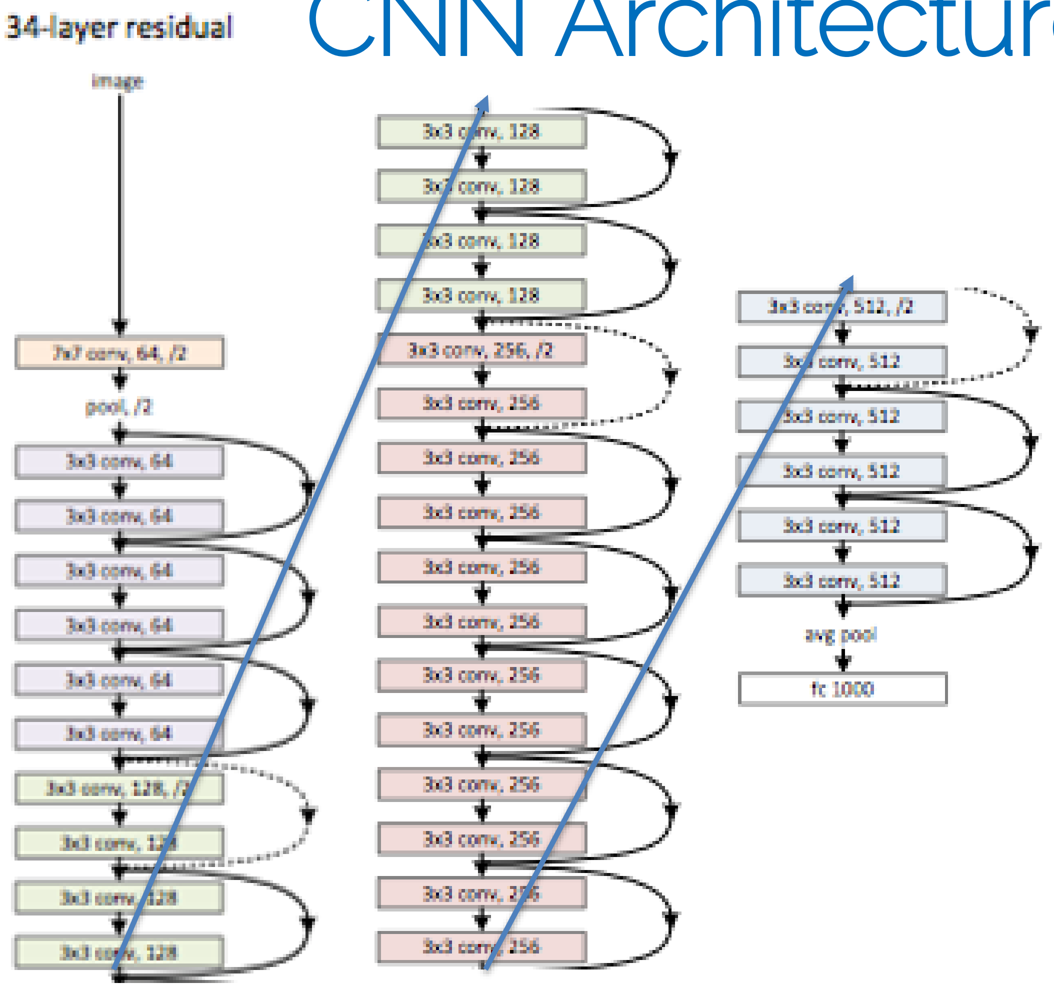
CNN Architectures: ResNet

[He et al. 2015]



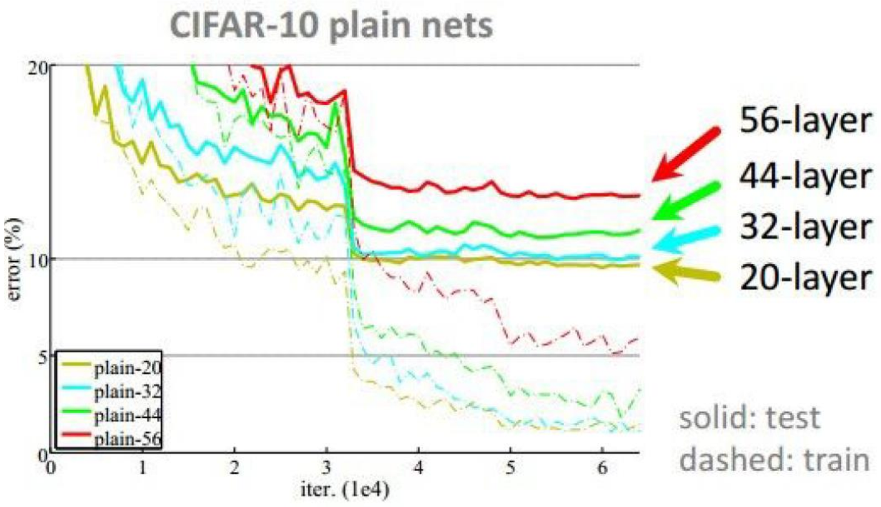


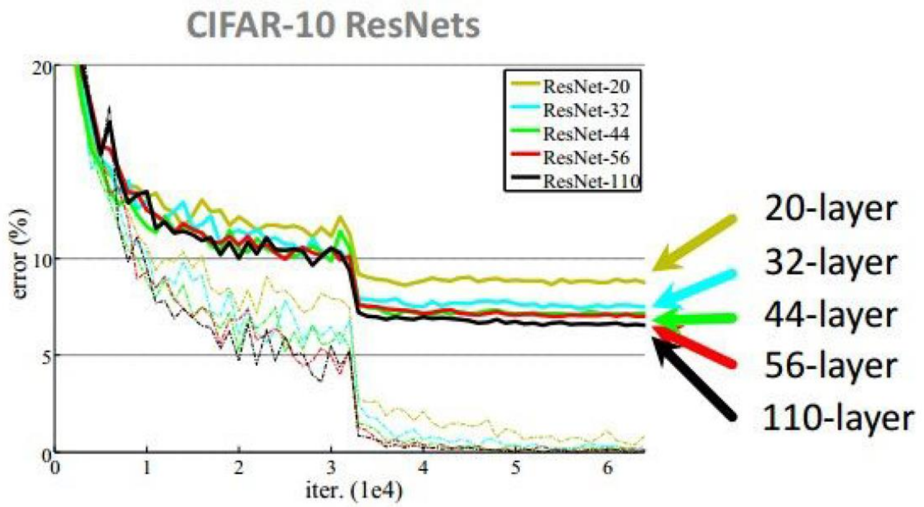
CNN Architectures: ResNet



- Batch norm after every Conv Layer
- Xavier/2 init by He et al.
- SGD + Momentum (0.9)
- Learning rate 0.1, divided by 10 when plateau
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout!

CNN Architectures





Slide by Li/Karpathy/Johnson

CNN Architectures: ResNet

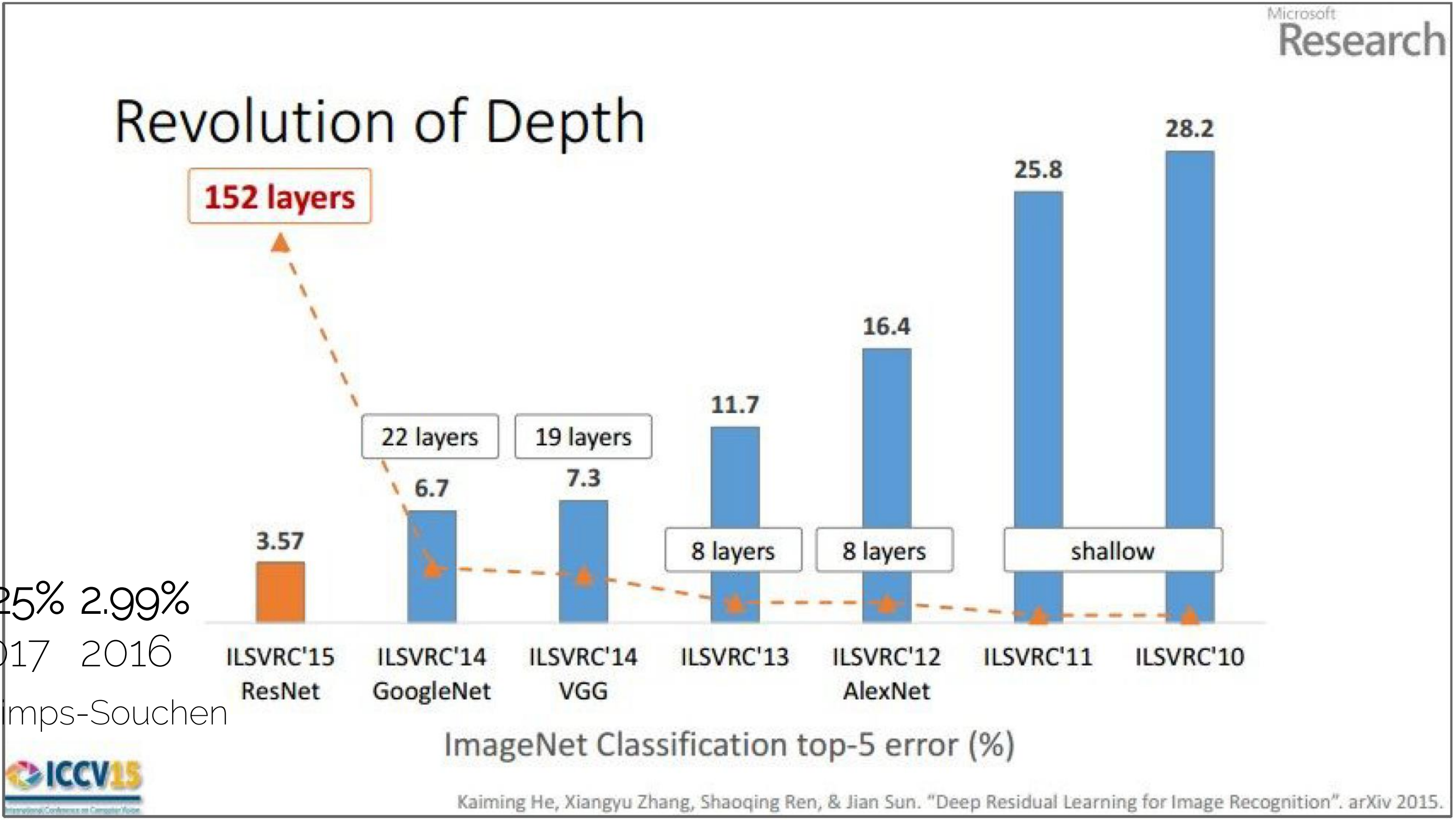
[He et al. 2015]

- What Conv Layers do spatially, ResNet and Inception models do across layers (kind of)

MSRA @ ILSVRC & COCO 2015 Competitions

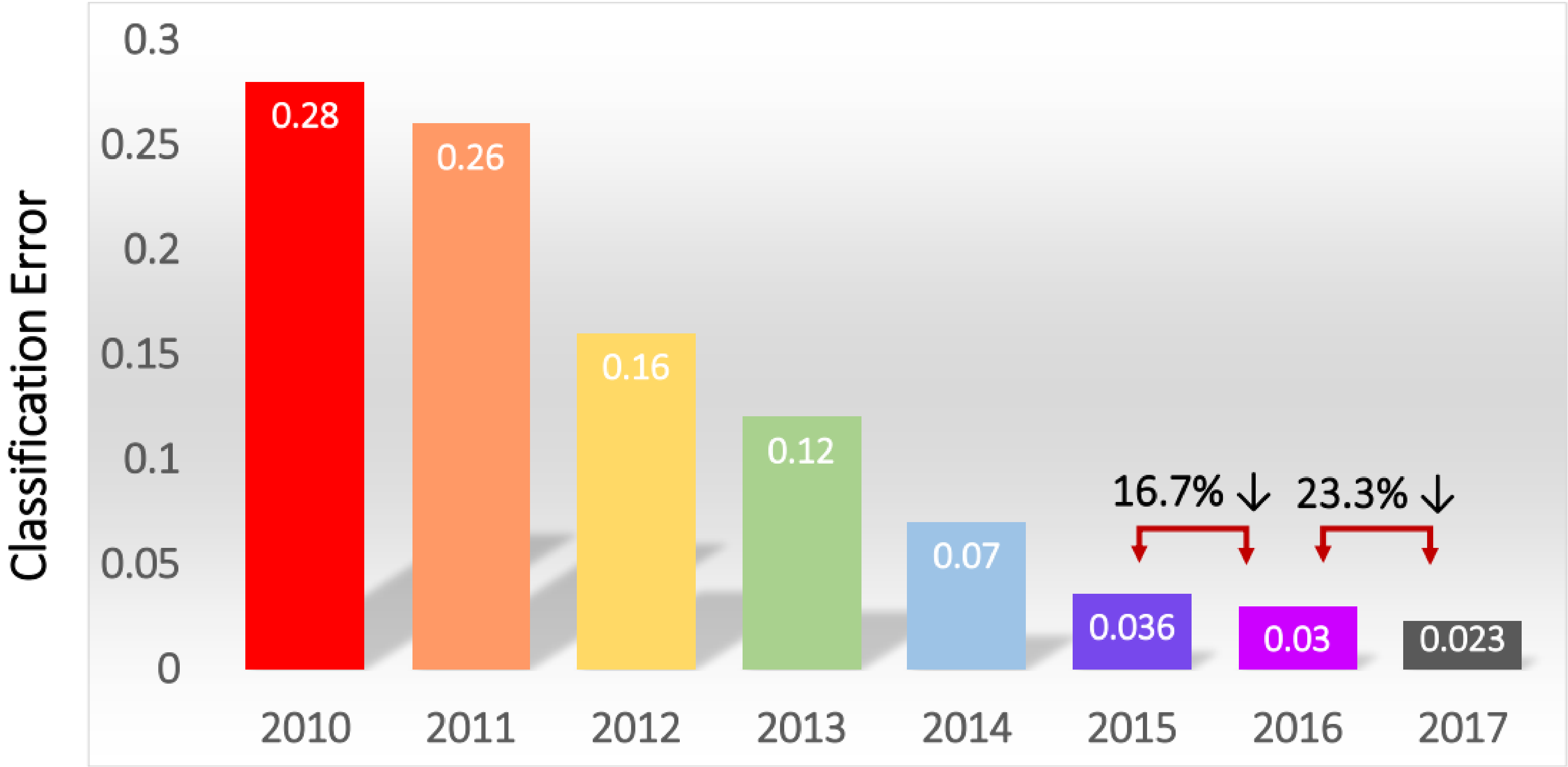
- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

CNN Architectures



2.25% 2.99%
2017 2016
WMW / Trimps-Souchen

CNN Architectures



http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf

History of Conv Nets

- LeNet-5 [LeCun et al. 98]
- AlexNet [Krishevsky et al. 12]
- ZFNet [Zeiler and Fergus 13]
- VGGNet [Simonyan and Zisserman 14]

- ‘Advanced’ Architectures
  - GoogLeNet [Szegedy et al. 14]
  - ResNet [He et al. 15]
  - XceptionNet [Chollet 17]

CNN Architectures

- Summary:
  - ConvNets stack Conv, Pool, FC
  - Trend towards smaller filters and deeper
  - Trend towards removing Pool and FC
    - I.e., only conv -> ‘fully-convolutional’

  - ResNet and InceptionNet architectures crush all!
    - Need to fast forward gradients ☺

Administrative Things

- Evaluation starts!
- Next Tuesday Dec 12th: More on ConvNets!
- This Thursday Dec 7th: No Tutorial (Dies Academicus)