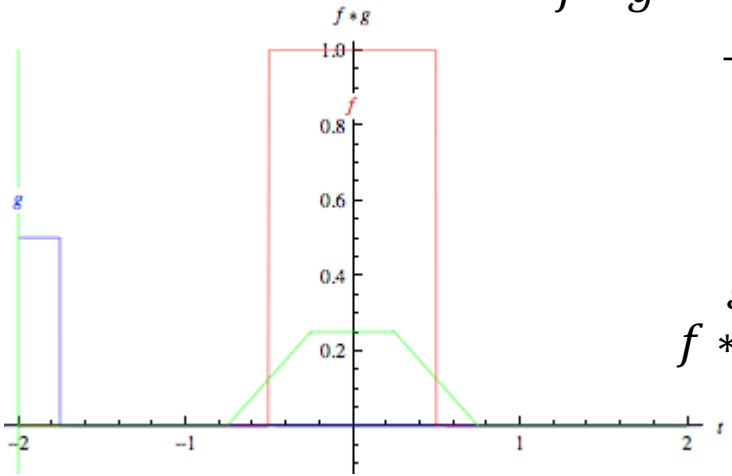
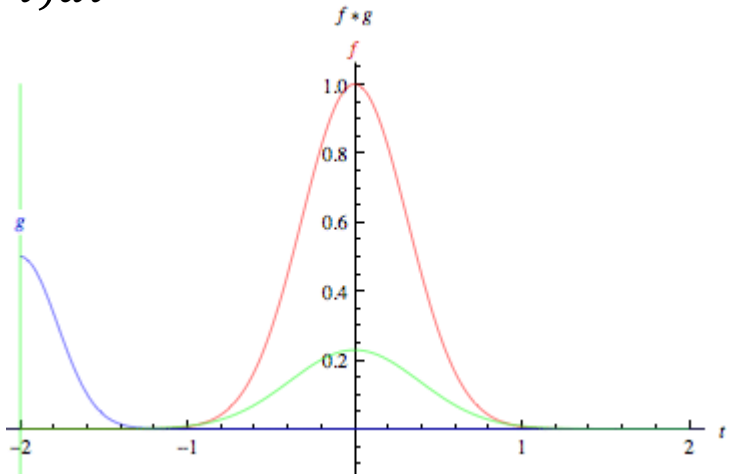
Lecture 9 Recap

What are Convolutions?

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



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



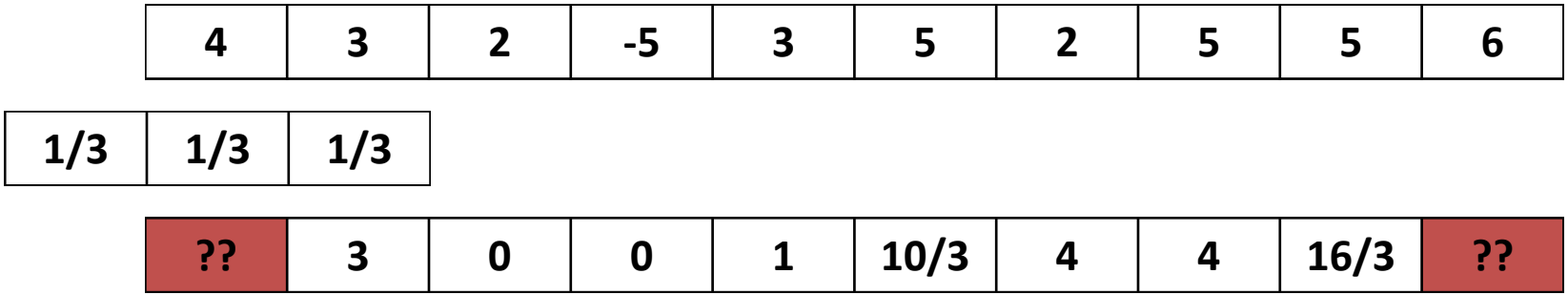
Convolution of two box functions

Convolution of two Gaussians

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

What are Convolutions?

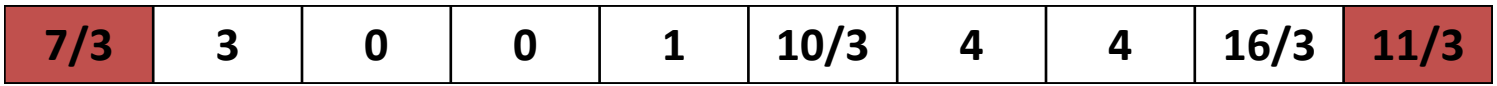
Discrete case: box filter



What to do at boundaries?

1) Shrink

2) Pad often '0'



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

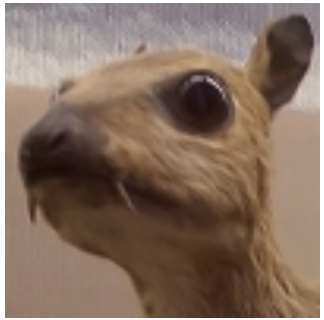
6	1	8
-7	9	2
-5	-9	3

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

Image filters

- Each kernel gives us a different image filter

Input



Edge detection

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

Box mean

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

LET'S LEARN THESE FILTERS!

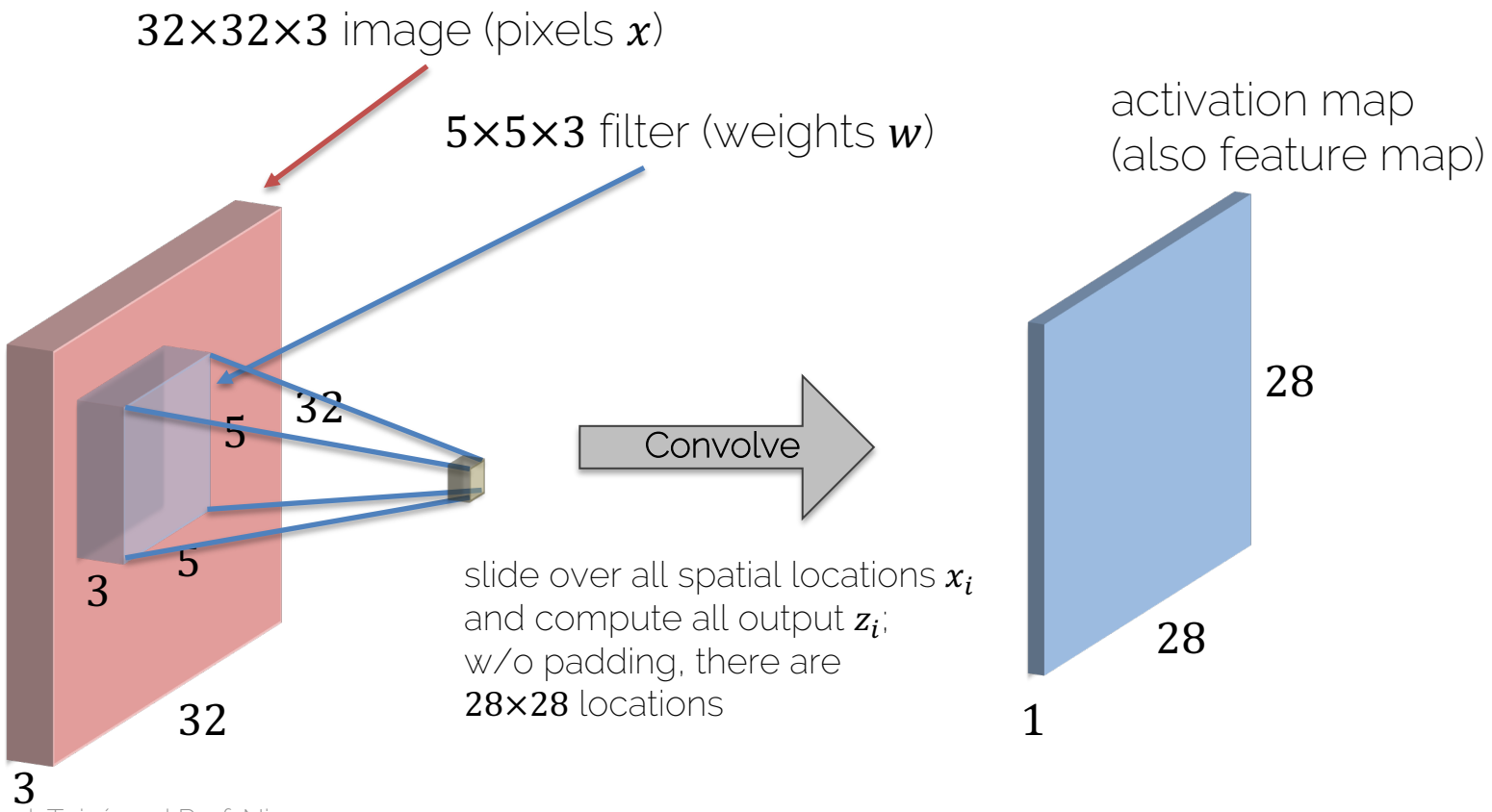
Sharpen

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

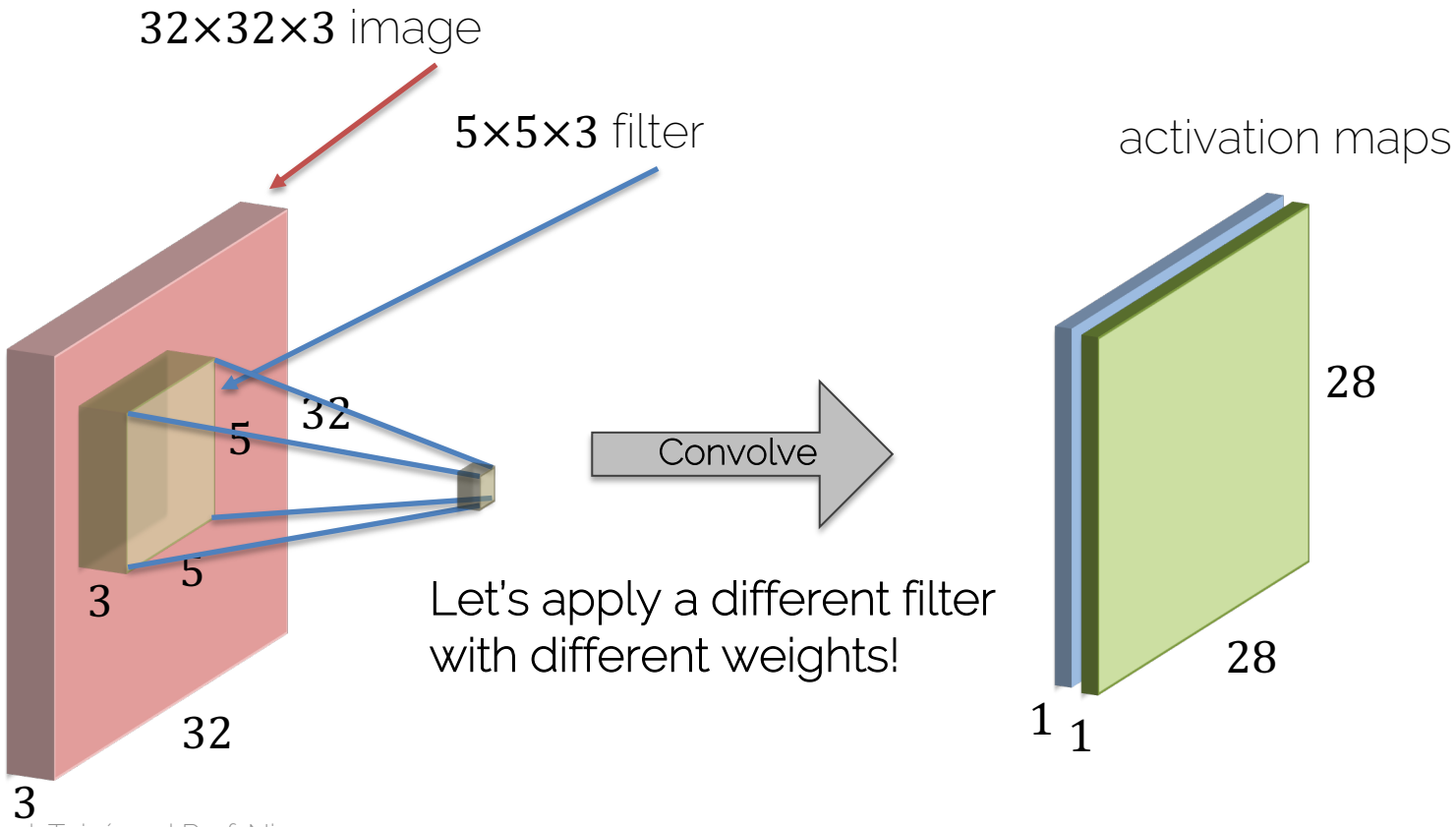
Gaussian blur

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

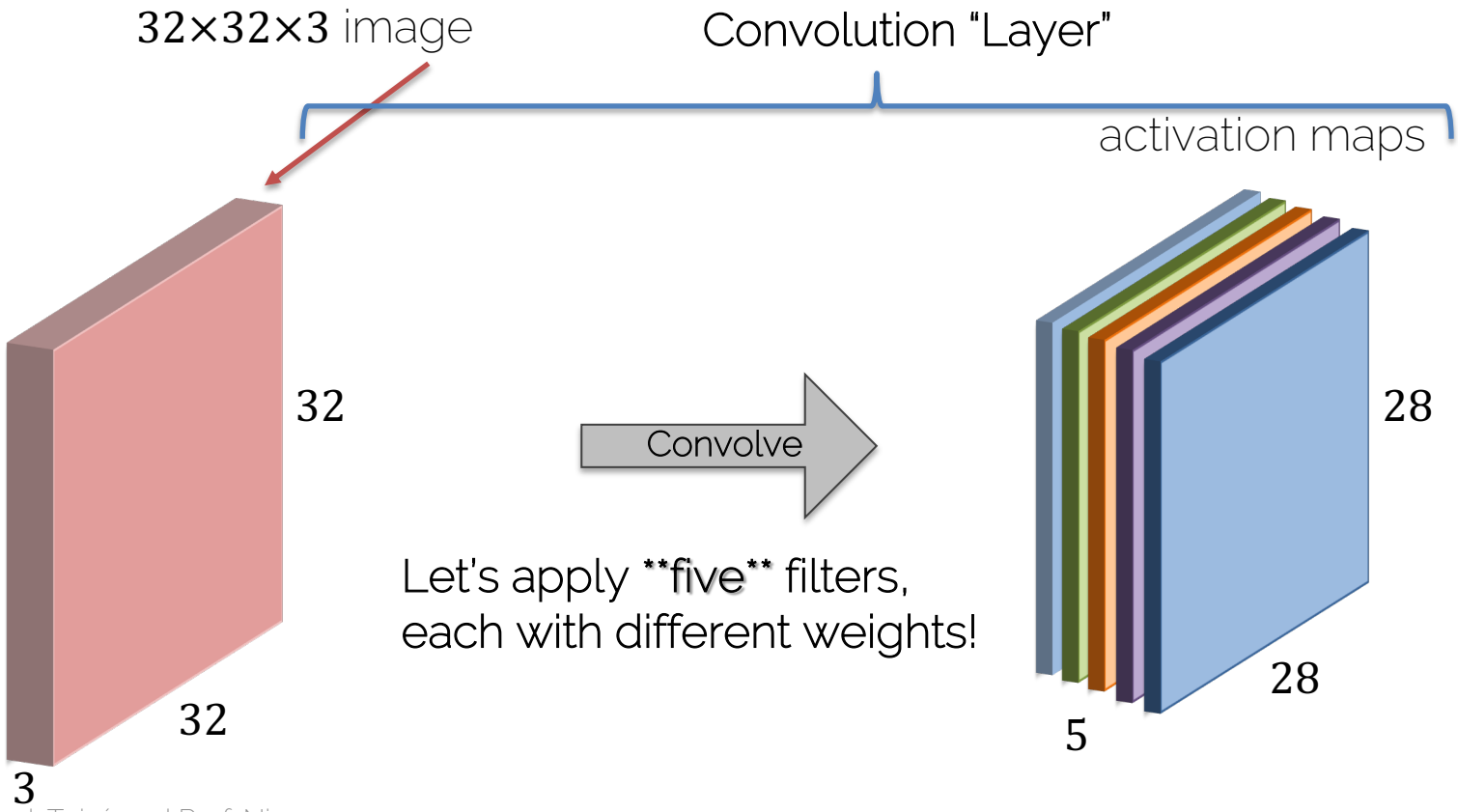
Convolutions on RGB Images



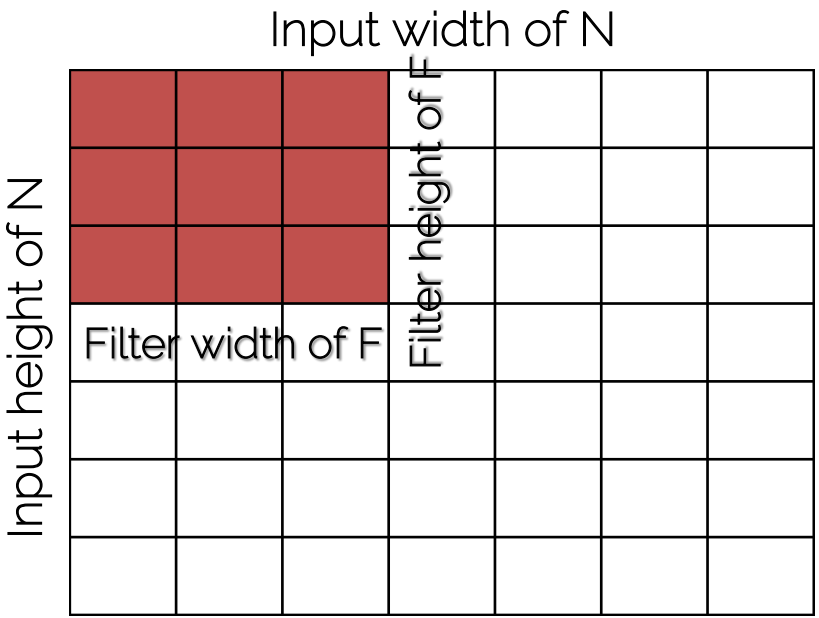
Convolution Layer



Convolution Layer



Convolution Layers: Dimensions



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

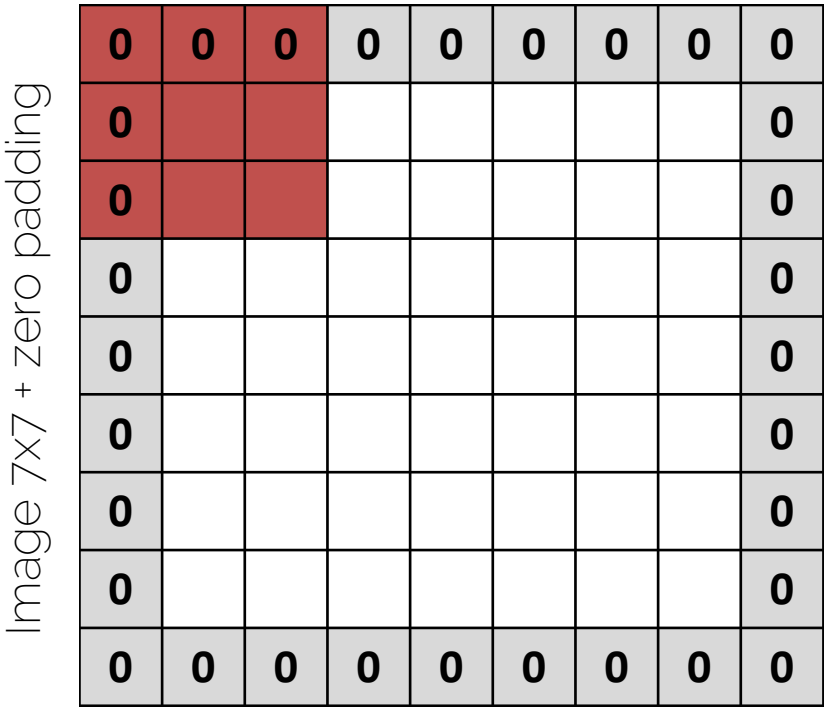
$N = 7, F = 3, S = 1$: $\frac{7-3}{1}+1 = 5$

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

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

Fractions are illegal

Convolution Layers: Padding

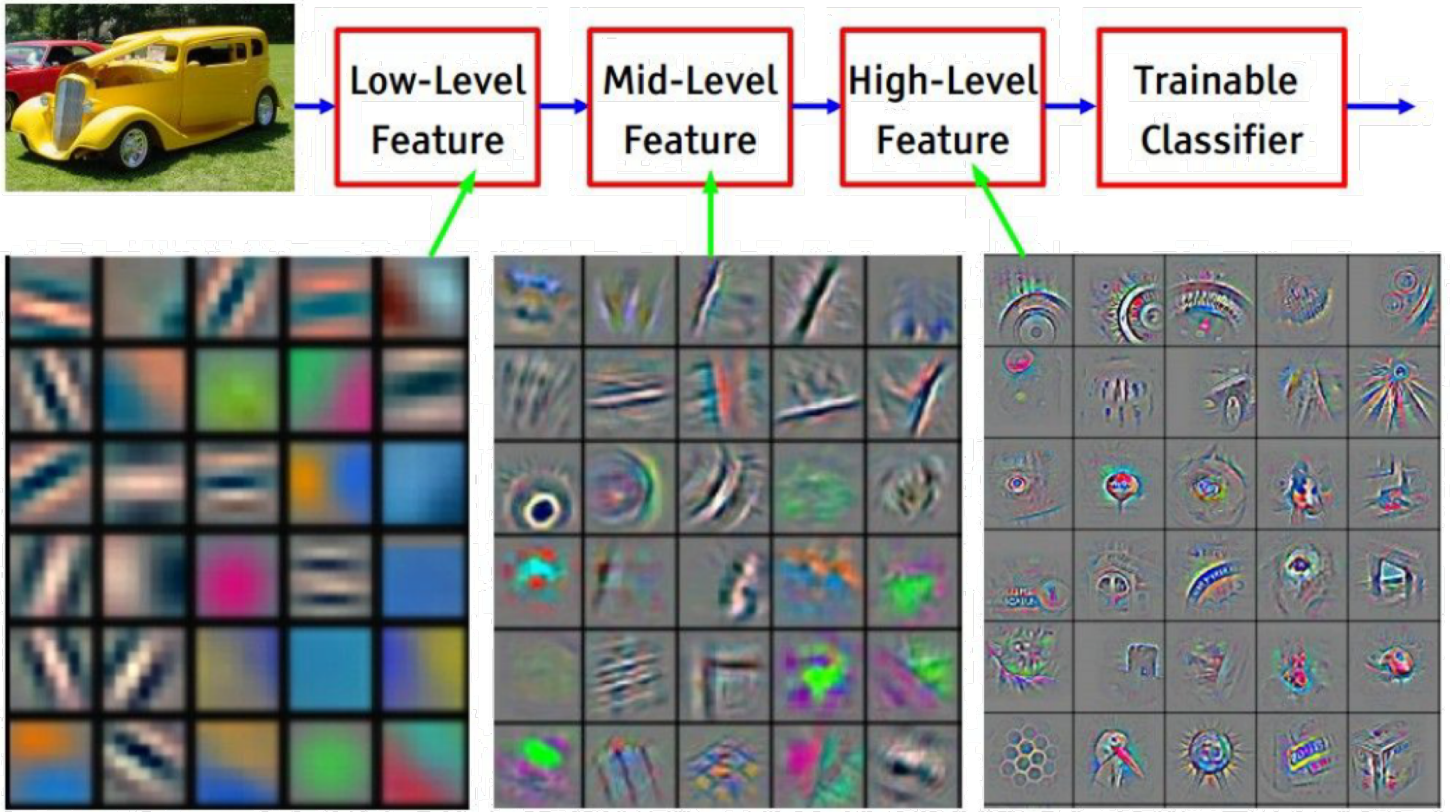


Types of convolutions:

- **Valid** convolution: using no padding
- **Same** convolution: output=input size

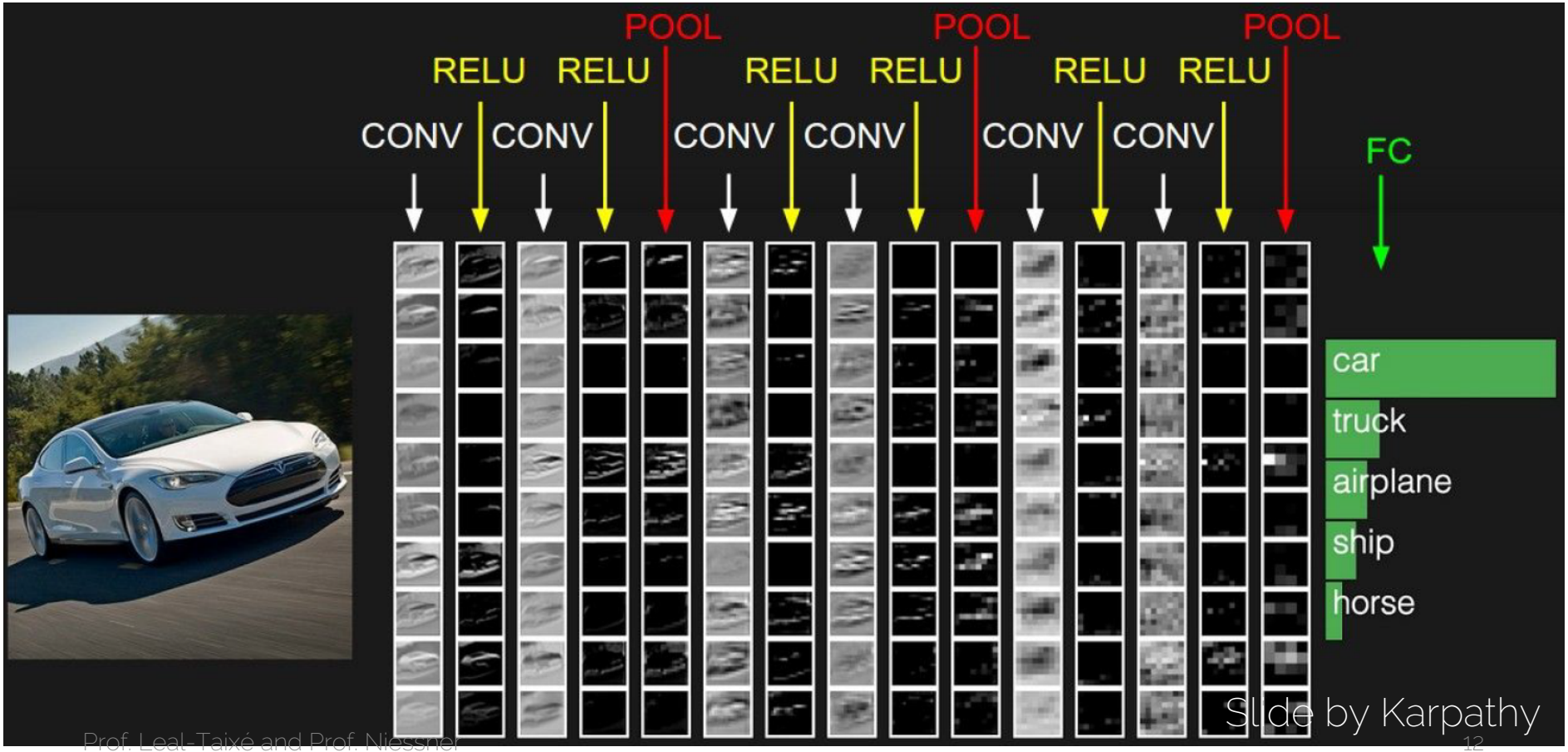
Set padding to $P = \frac{F-1}{2}$

CNN learned filters



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

CNN Prototype

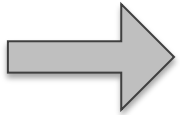


Pooling Layer: Max Pooling

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

Classic architectures

LeNet

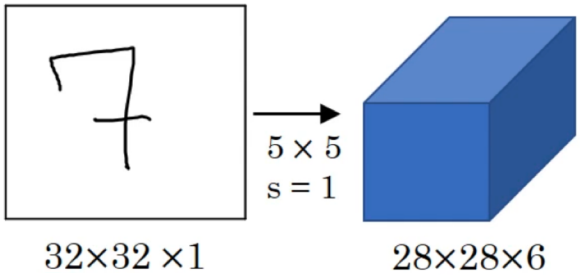
- Digit recognition: 10 classes

32×32 ×1

[LeCun et al. 1998]

LeNet

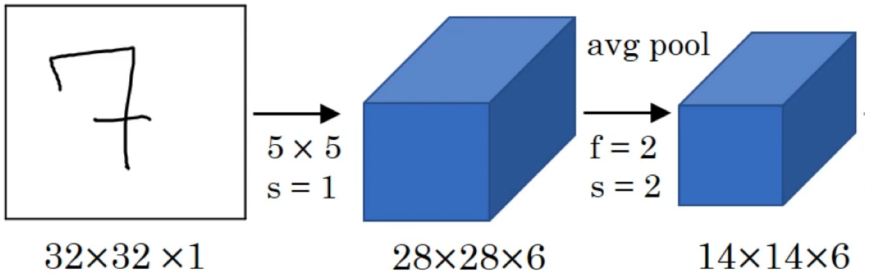
- Digit recognition: 10 classes



- Valid convolution: size shrinks
- How many conv filters are there in the first layer? 6

LeNet

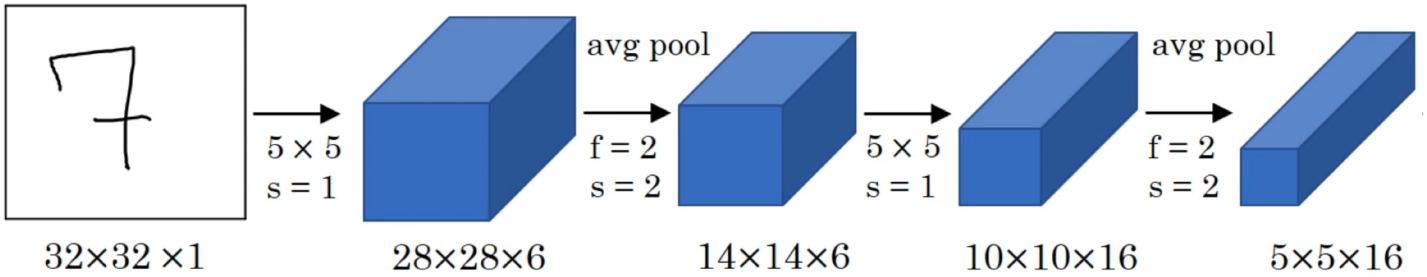
- Digit recognition: 10 classes



- At that time average pooling was used, now max pooling is much more common

LeNet

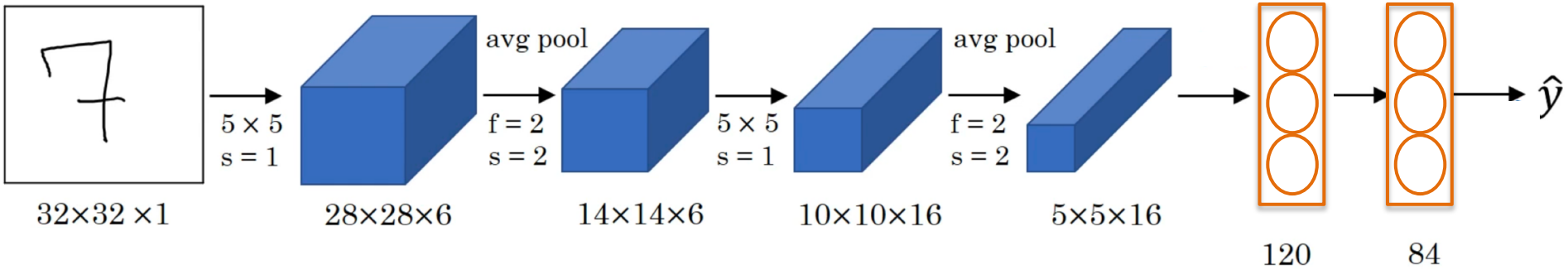
- Digit recognition: 10 classes



- Again valid convolutions, how many filters?

LeNet

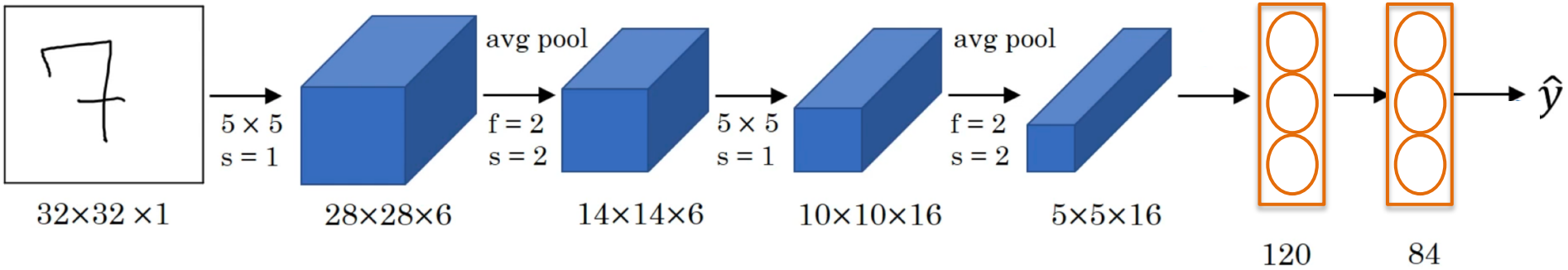
- Digit recognition: 10 classes



- Use of tanh/sigmoid activations → not common now!

LeNet

- Digit recognition: 10 classes

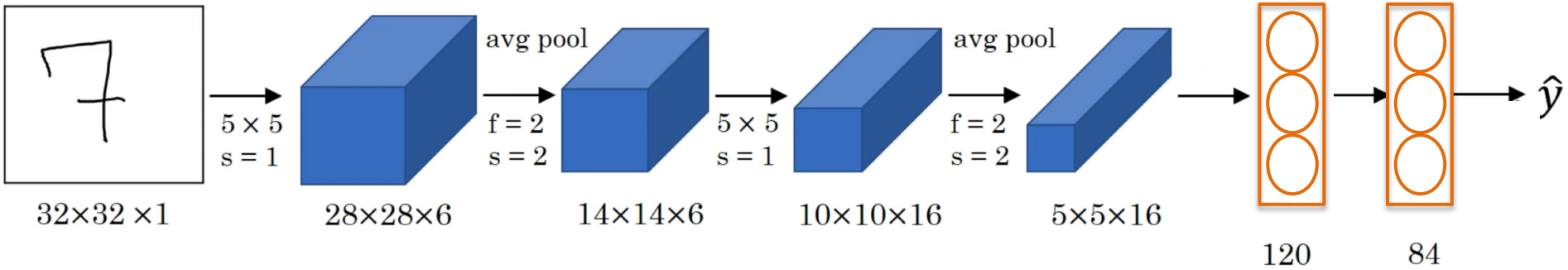


- Conv -> Pool -> Conv -> Pool -> Conv -> FC

LeNet

- Digit recognition: 10 classes

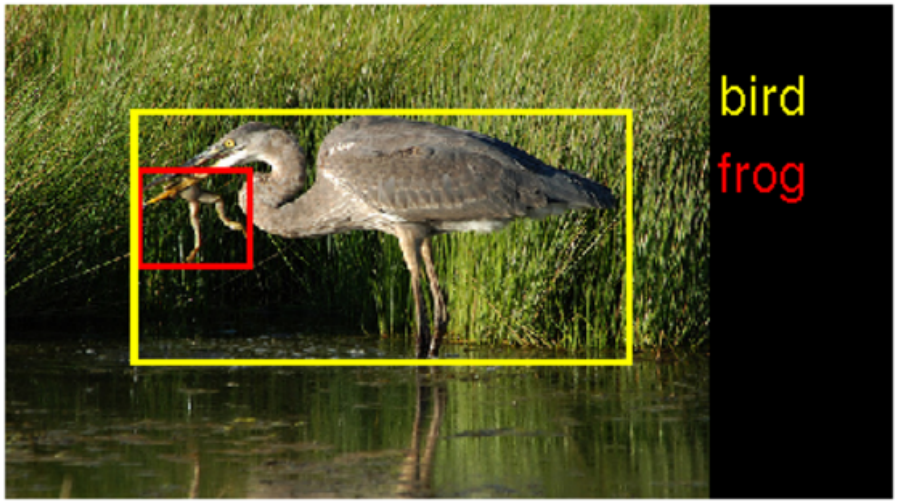
60k parameters

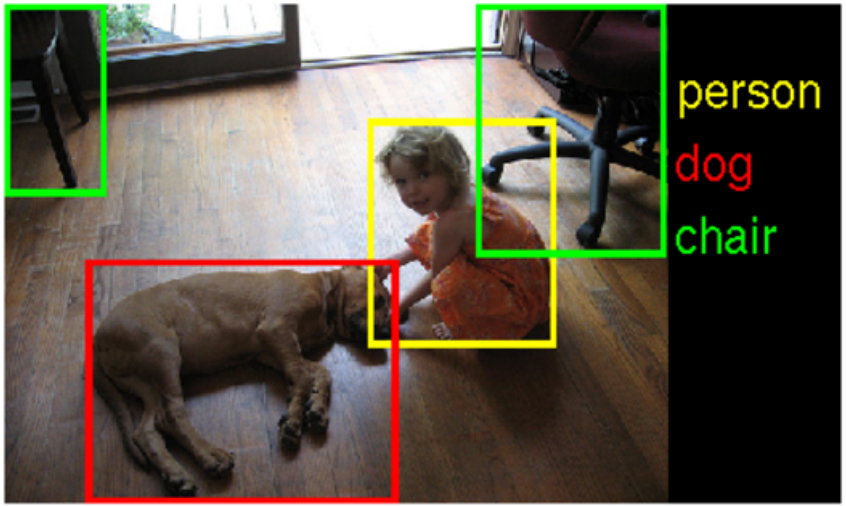


- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, height ↓ Number of filters ↑

Test Benchmarks

- Image Net Dataset:
  ImageNet Large Scale Visual Recognition Competition (ILSVRC)

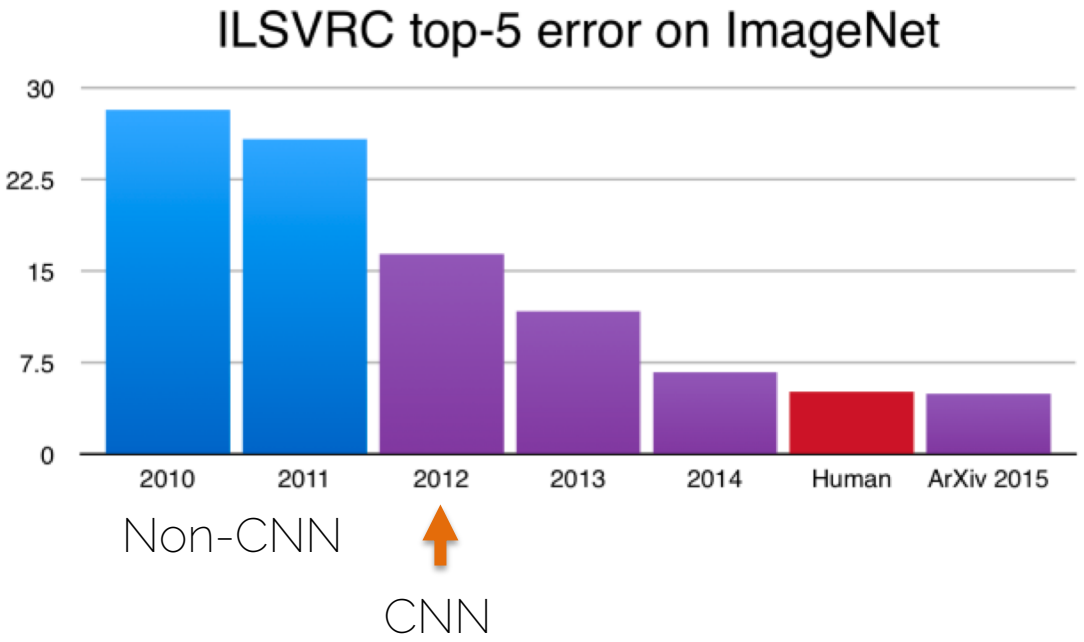




Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. IJCV 2015

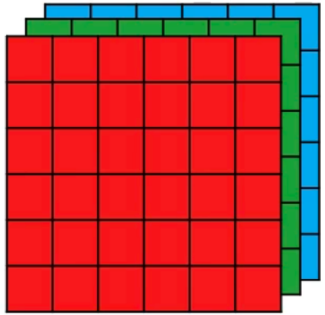
AlexNet

- Cut ImageNet error down in half



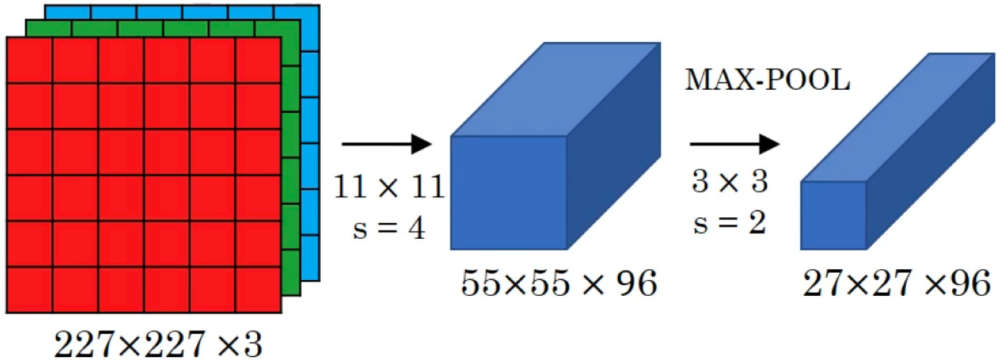
AlexNet

[Krizhevsky et al. 2012]



AlexNet

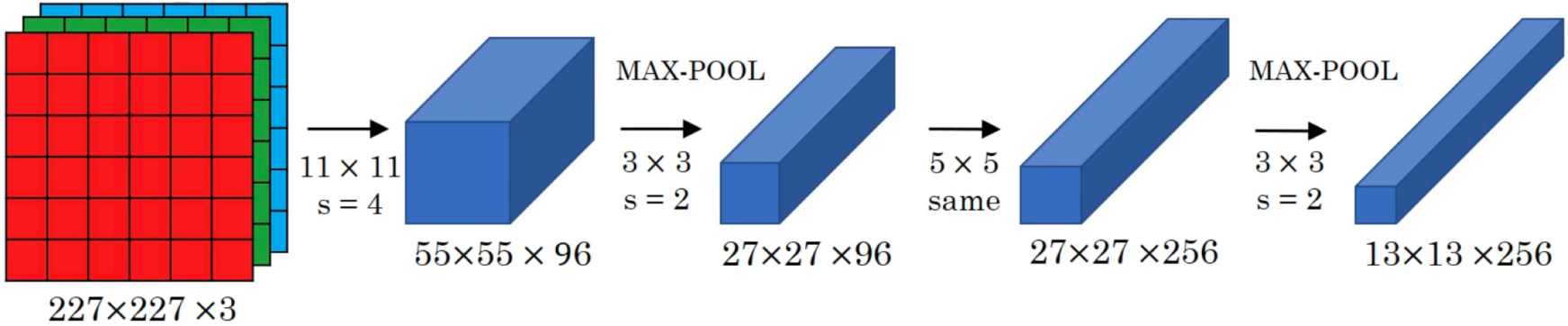
[Krizhevsky et al. 2012]



- First filter with stride 4 to reduce size significantly
- 96 filters

AlexNet

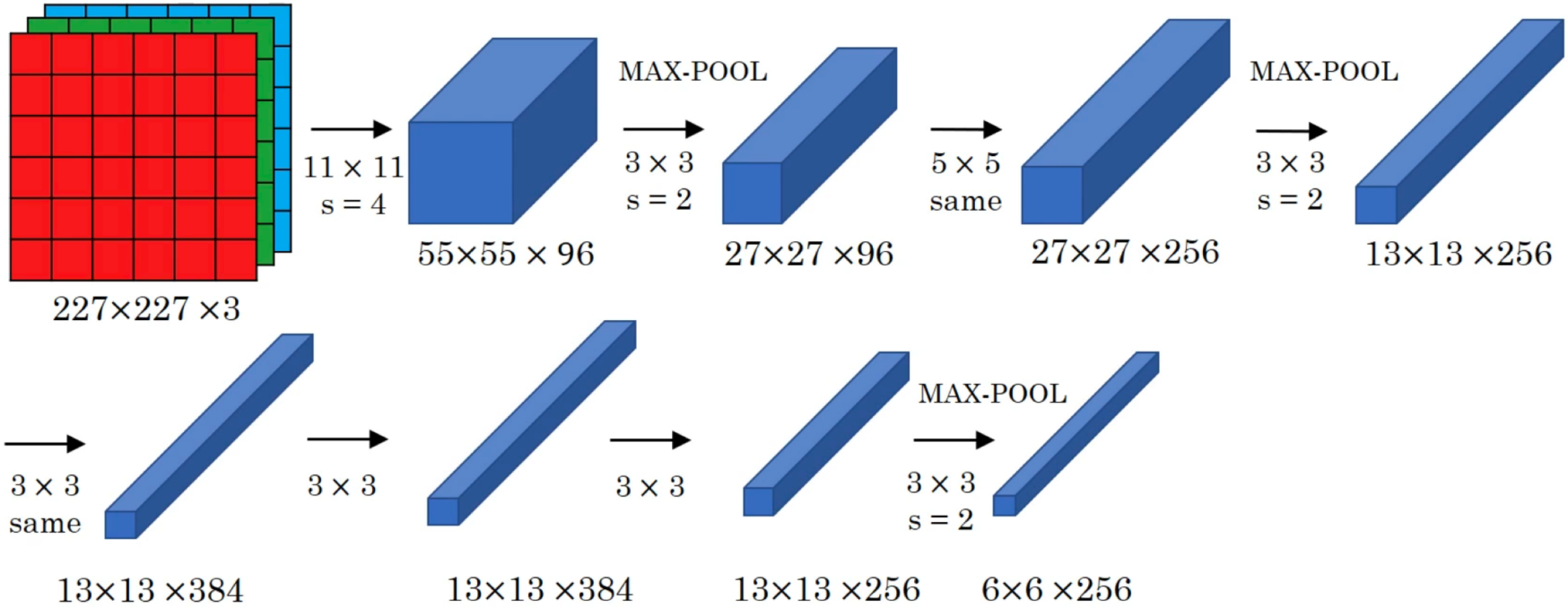
[Krizhevsky et al. 2012]



- Use of same convolutions
- As with LeNet, Width, height ↓ Number of filters ↑

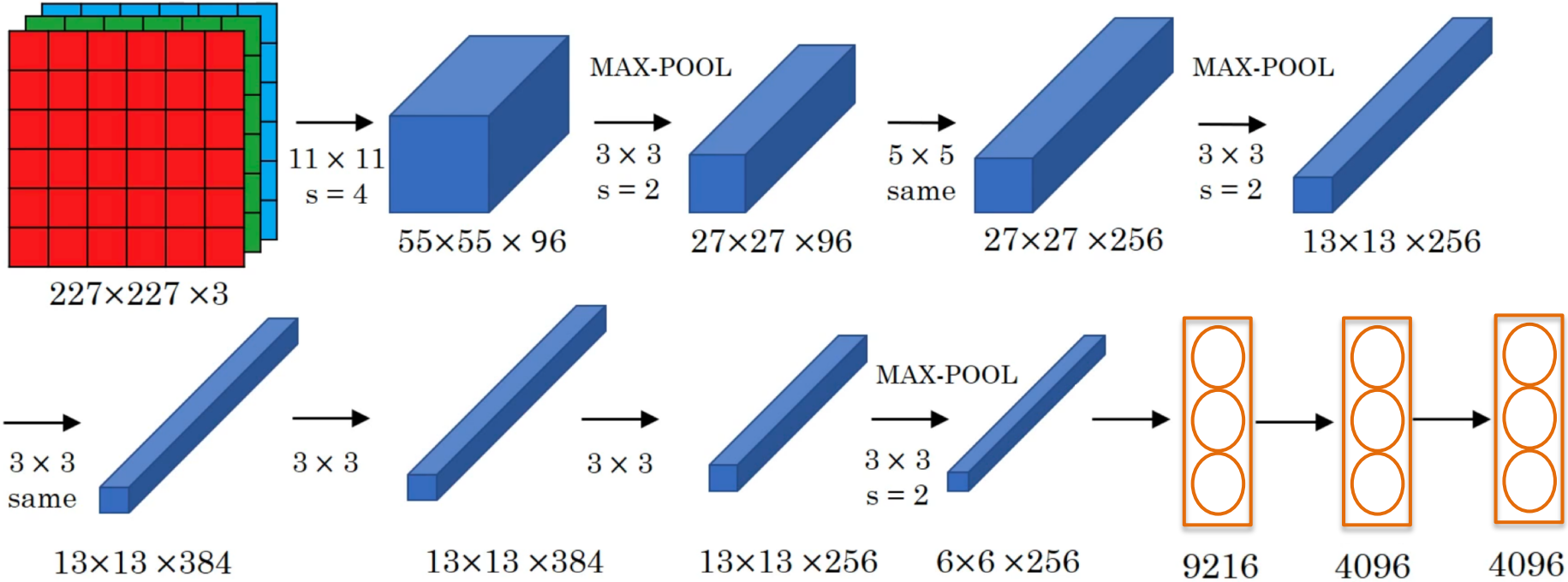
AlexNet

[Krizhevsky et al. 2012]



AlexNet

[Krizhevsky et al. 2012]



- Softmax for 1000 classes

AlexNet

[Krizhevsky et al. 2012]

- Similar to LeNet but much bigger

- Use of ReLU instead of tanh/sigmoid

60M parameters

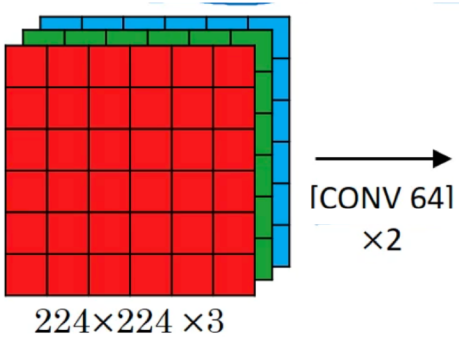
VGGNet

[Simonyan and Zisserman 2014]

- Striving for simplicity
- CONV = 3x3 filters with stride 1, same convolutions
- MAXPOOL = 2x2 filters with stride 2

VGGNet

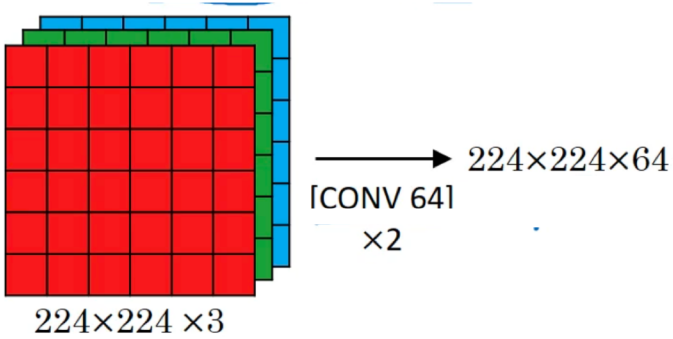
Conv=3x3,s=1,same
Maxpool=2x2,s=2



- 2 consecutive convolutional layers, each one with 64 filters
- What is the output size?

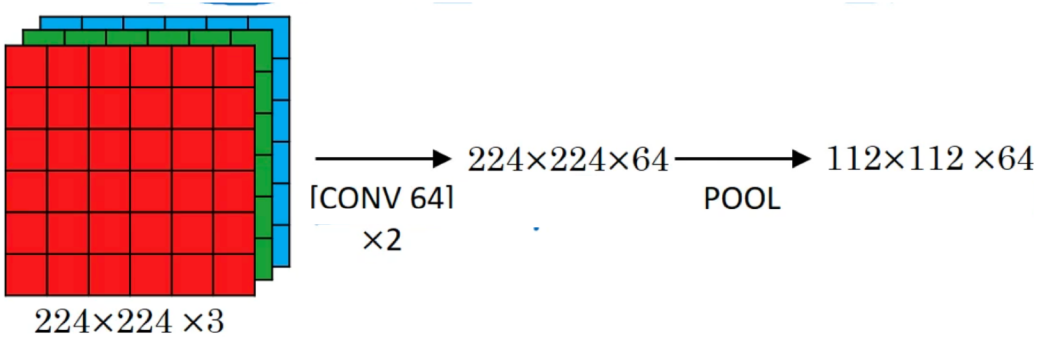
VGGNet

Conv=3x3,s=1,same
Maxpool=2x2,s=2



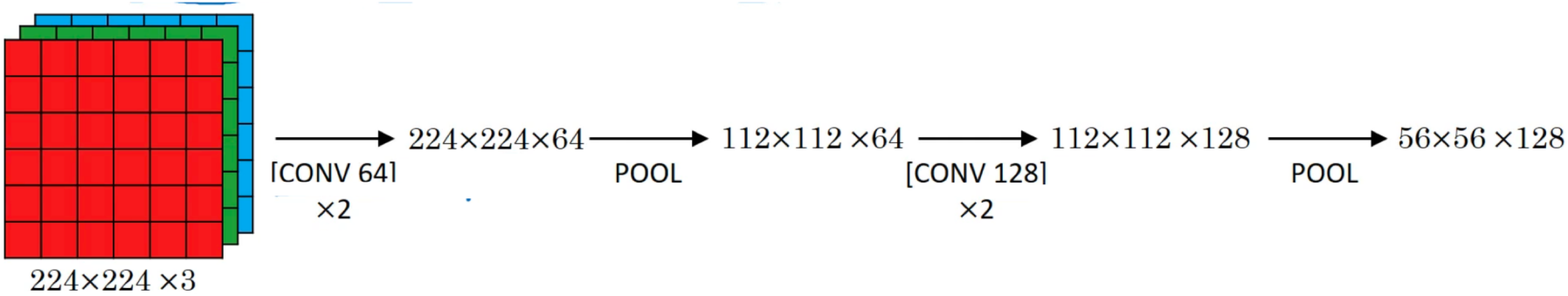
VGGNet

Conv=3x3,s=1,same
Maxpool=2x2,s=2



VGGNet

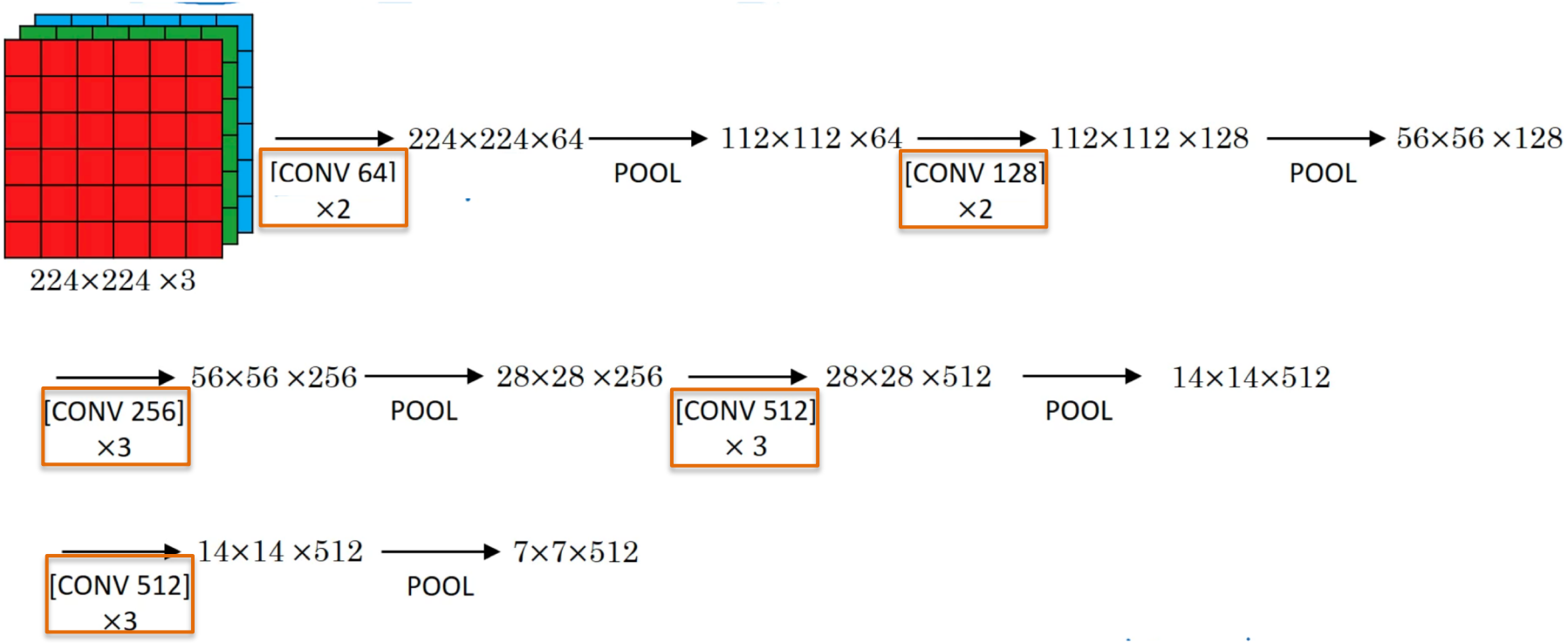
Conv=3x3,s=1,same
Maxpool=2x2,s=2



- Number of filters is multiplied by 2

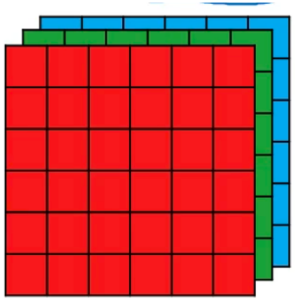
VGGNet

Conv=3x3,s=1,same
Maxpool=2x2,s=2



VGGNet

Conv=3x3,s=1,same
Maxpool=2x2,s=2



224×224 ×3

→ [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

VGGNet

- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, height ↓ Number of filters ↑

- Called VGG-16: 16 layers that have weights

138M parameters

- Large but simplicity makes it appealing

VGGNet

- A lot of architectures were analyzed

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**
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**
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**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

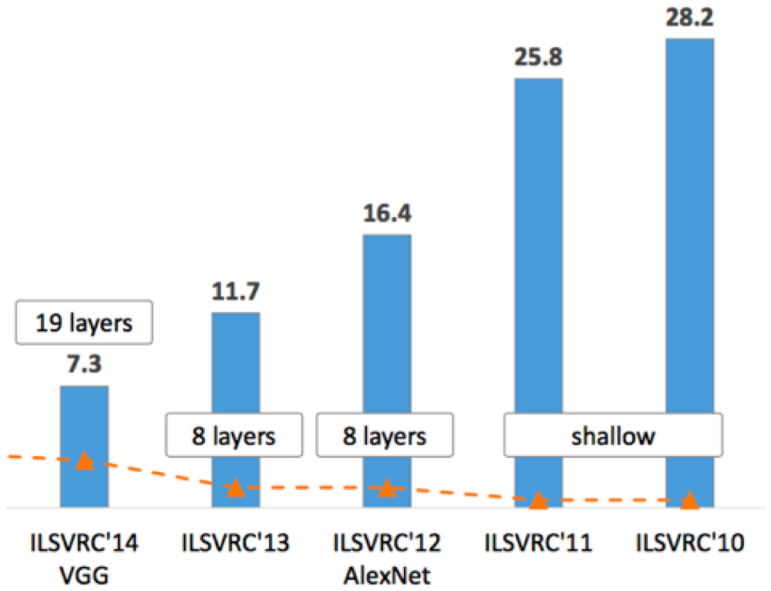
Table 2: **Number of parameters** (in millions).

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

[Simonyan and Zisserman 2014]

ImageNet results

- Top-5 error: check if your label is in your 5 first predictions



Skip connections

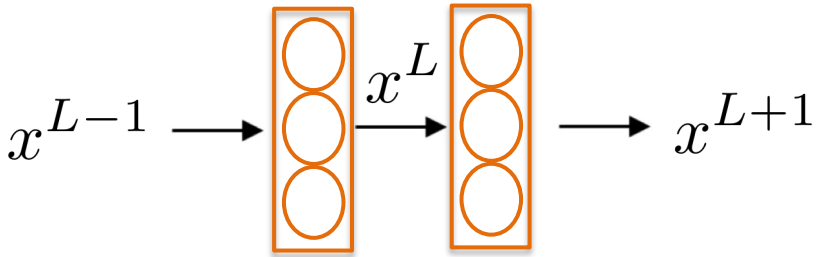
The problem of depth

- As we add more and more layers, training becomes harder
- Vanishing and exploding gradients

- How can we train very deep nets?

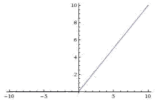
Residual block

- Two layers



Input → $W^L x^{L-1} + b^L$ → $x^L = f(W^L x^{L-1} + b^L)$ →

Linear

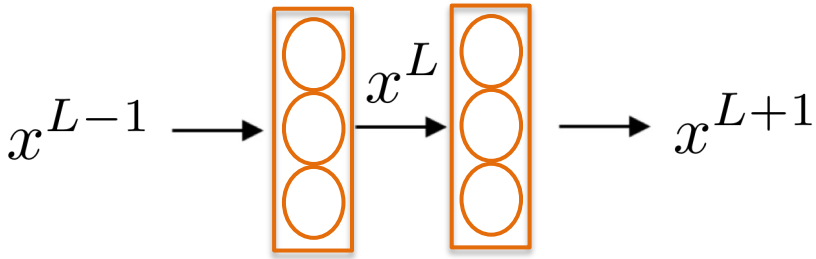


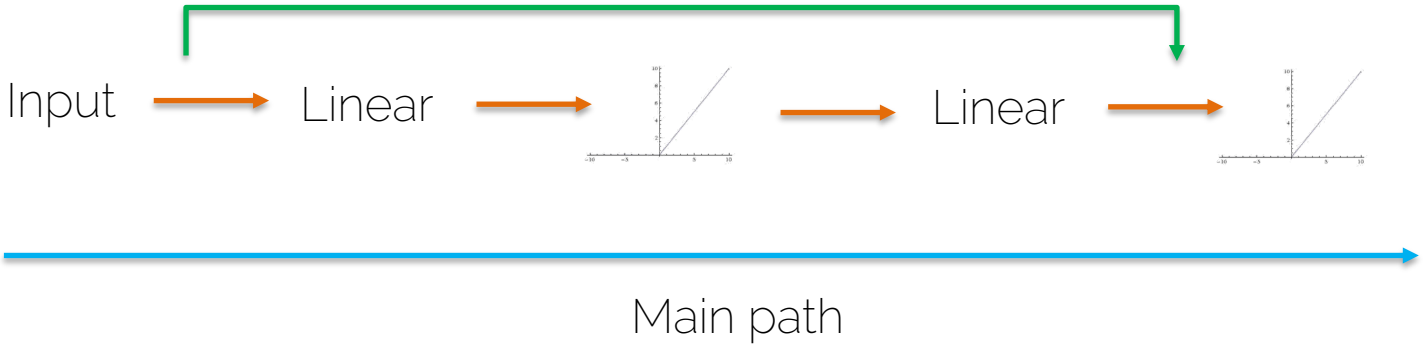
Non-linearity

→ $x^{L+1} = f(W^{L+1} x^L + b^{L+1})$

Residual block

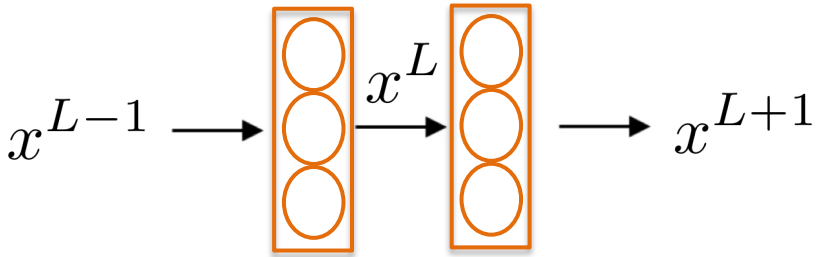
- Two layers



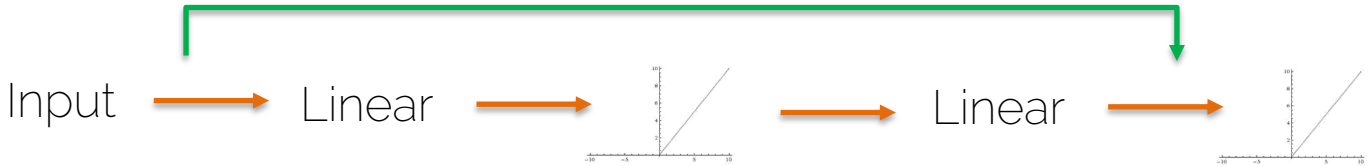


Residual block

- Two layers



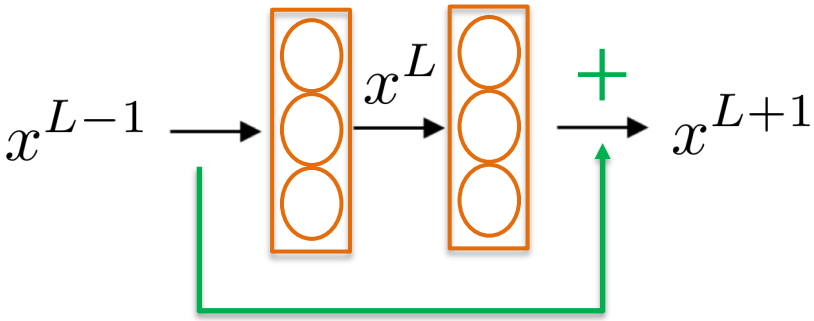
$$x^{L+1} = f(W^{L+1}x^L + b^{L+1} + x^{L-1})$$



$$x^{L+1} = f(W^{L+1}x^L + b^{L+1})$$

Residual block

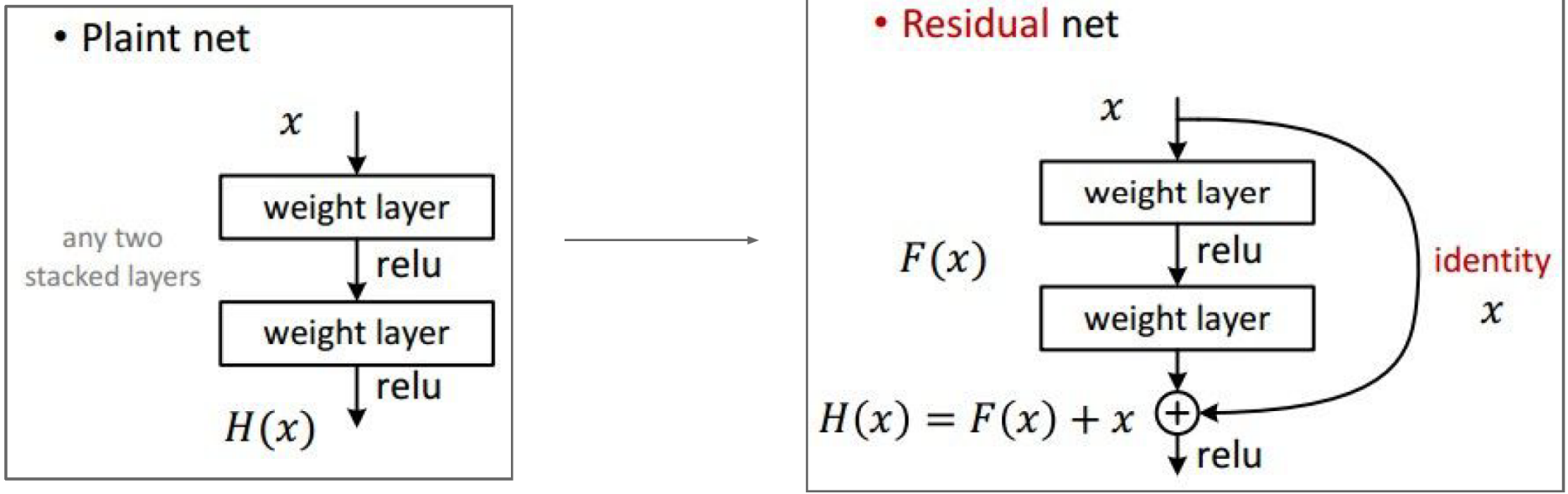
- Two layers



- Usually use a same convolution since we need same dimensions
- Otherwise we need to convert the dimensions with a matrix of learned weights or zero padding

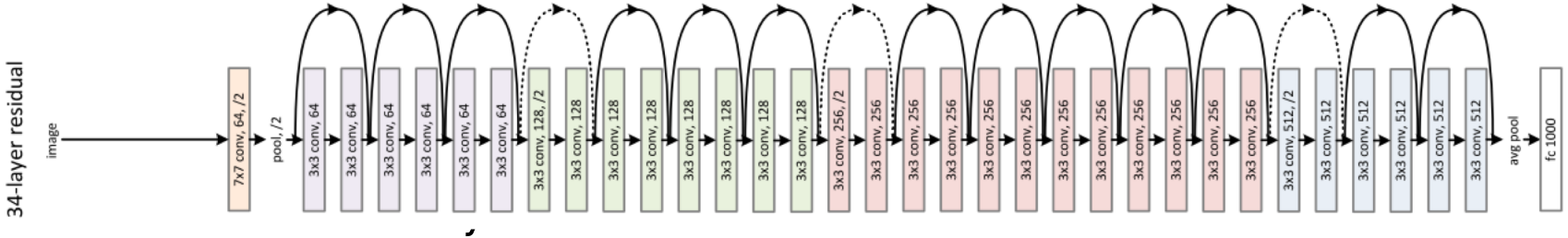
ResNet

[He et al. 2015]



ResNet

[He et al. 2015]

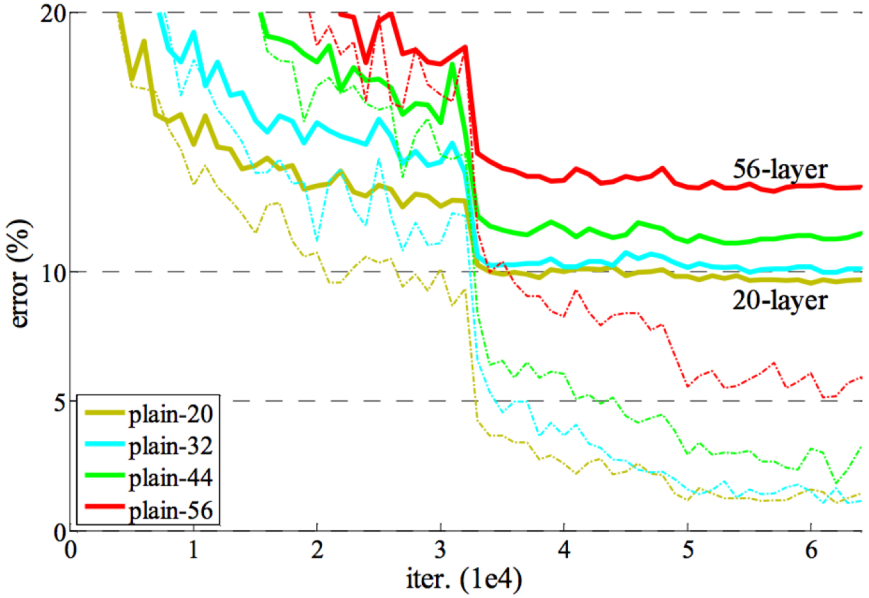


- Xavier/2 initialization
- SGD + Momentum (0.9)
- Learning rate 0.1, divided by 10 when plateau
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout

ResNet

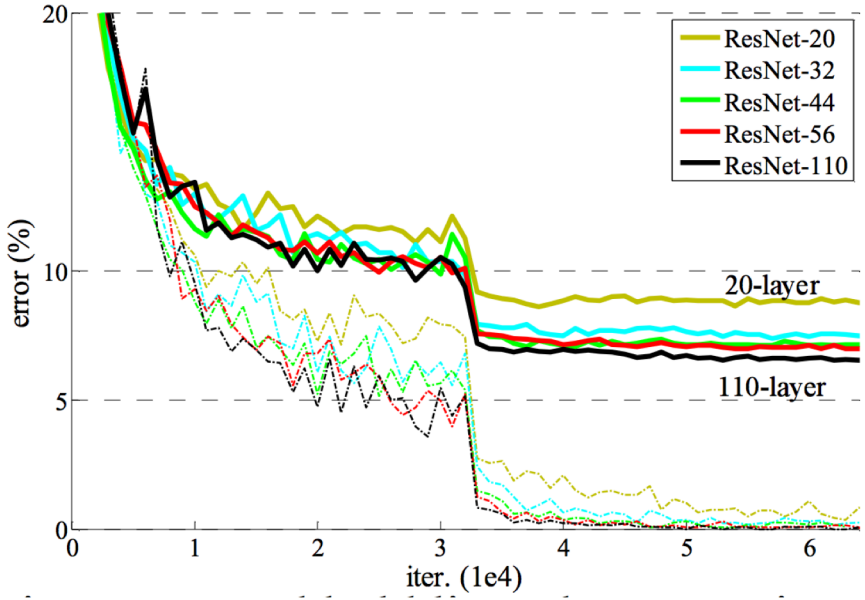
- If we make the network deeper, at some point performance starts to degrade

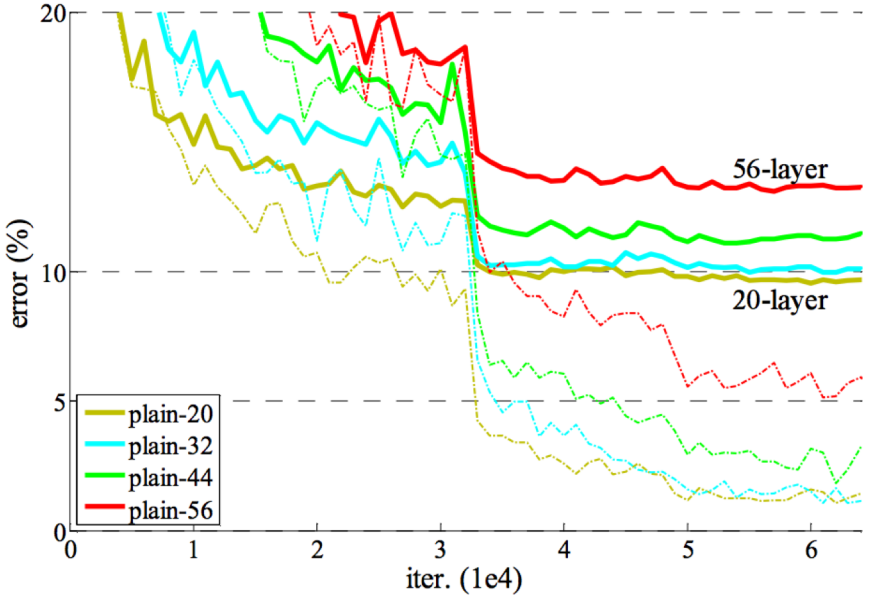
- Too many parameters, the optimizer cannot properly train the network



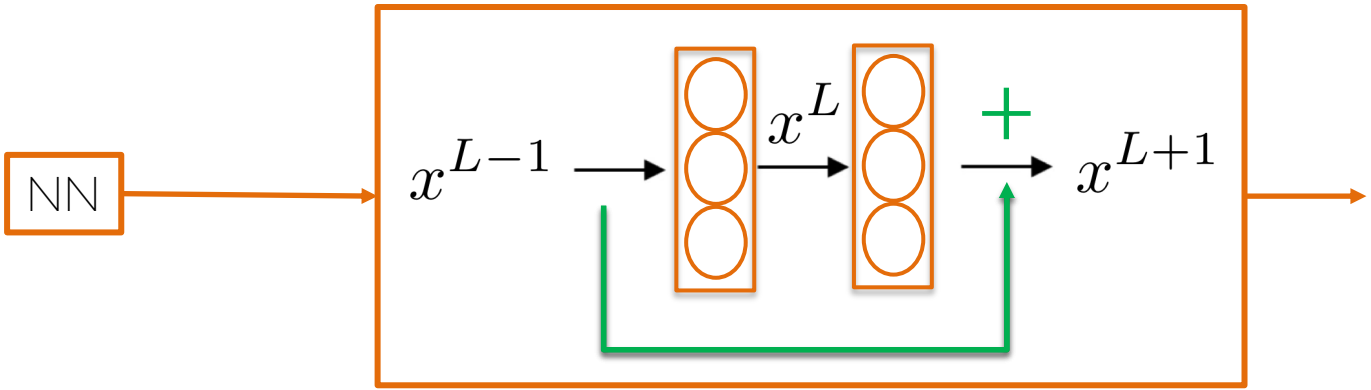
ResNet

- If we make the network deeper, at some point performance starts to degrade



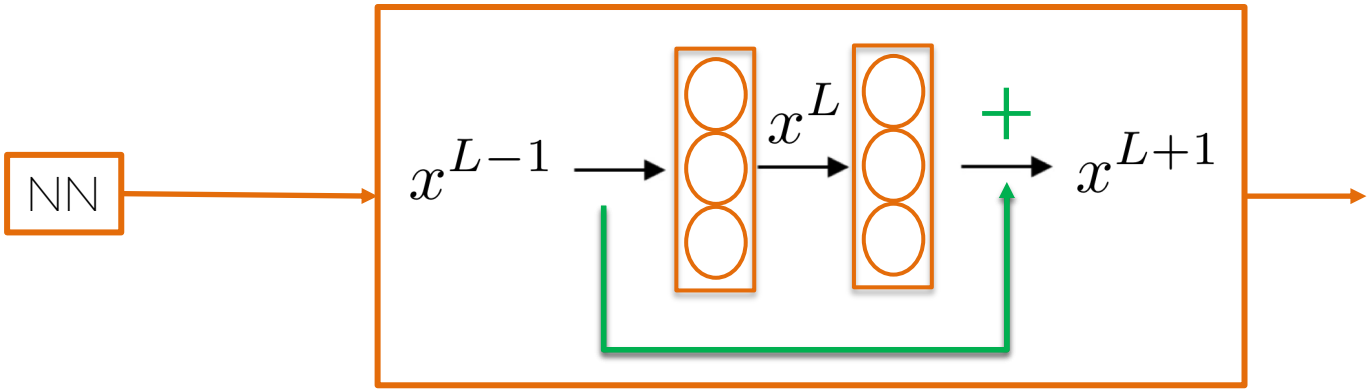


Why do ResNets work?



- How is this block really affecting me?

Why do ResNets work?

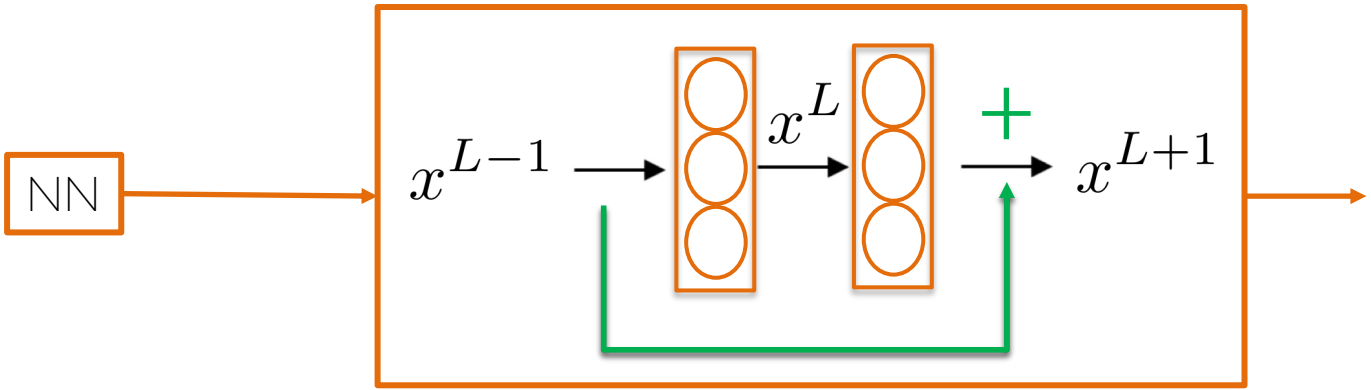


$$x^{L+1} = f(W^{L+1}x^{L} + b^{L+1} + x^{L-1})$$

~zero ~zero

$$x^{L+1} = f(x^{L-1})$$

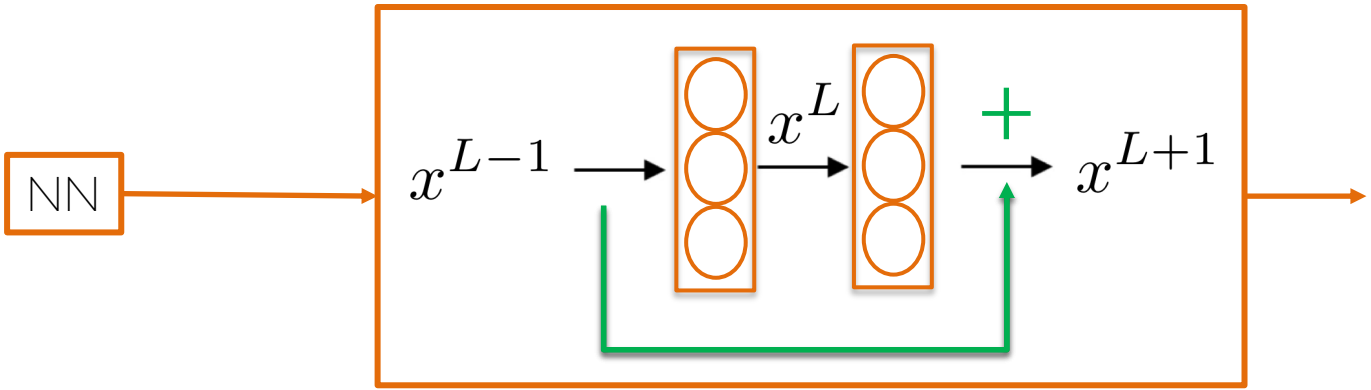
Why do ResNets work?



- We kept the same values and added a non-linearity

$$x^{L+1} = f(x^{L-1})$$

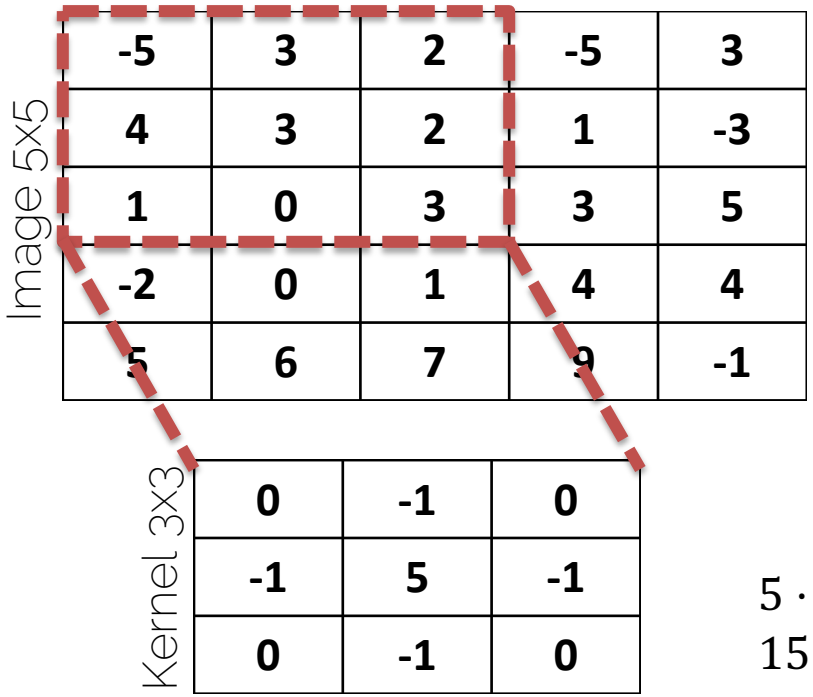
Why do ResNets work?

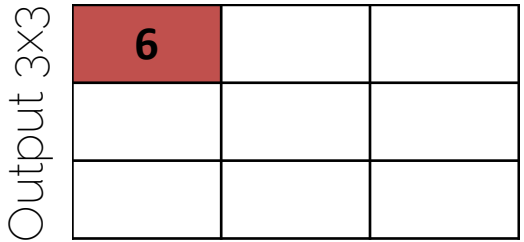


- The identity is easy for the residual block to learn
- Guaranteed it will not hurt performance, can only improve

1x1 convolutions

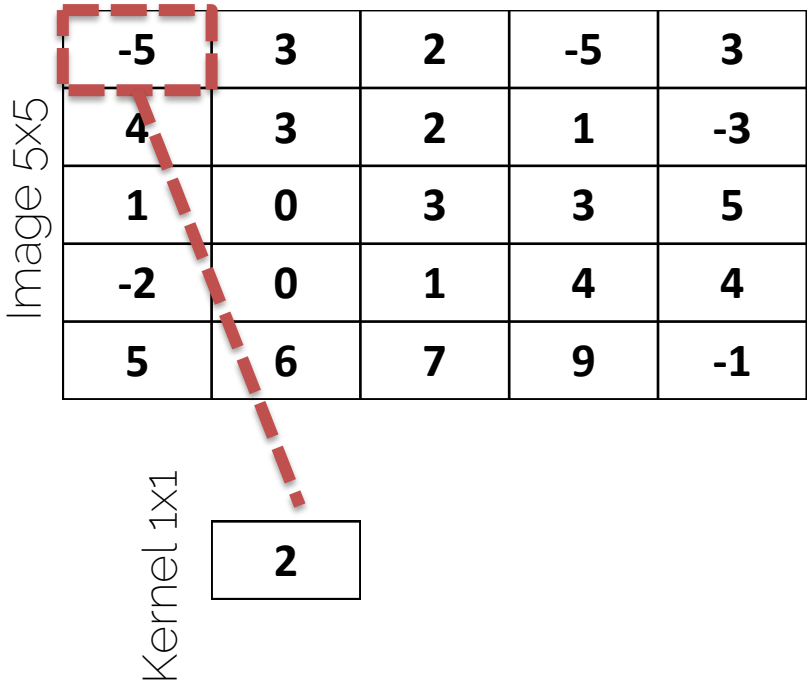
Recall: Convolutions on Images





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

1x1 convolution



What is the output size?

1x1 convolution

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 1X1

2

-10				

$$-5 * 2 = -10$$

1x1 convolution

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 1X1

2

-10	6	4	-10	6
8	6	4	2	-6
2	0	6	6	10
-4	0	2	8	8
10	12	14	18	-2

$$-1 * 2 = -2$$

1x1 convolution

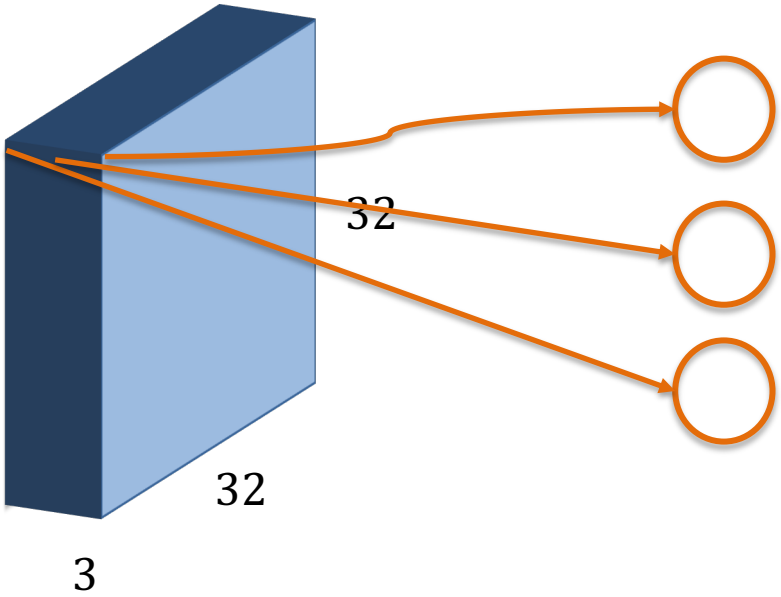
Image 5x5

-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

-10	6	4	-10	6
8	6	4	2	-6
2	0	6	6	10
-4	0	2	8	8
10	12	14	18	-2

- For 1 kernel or filter, it keeps the dimensions and just scales the input with a number

1x1 convolution

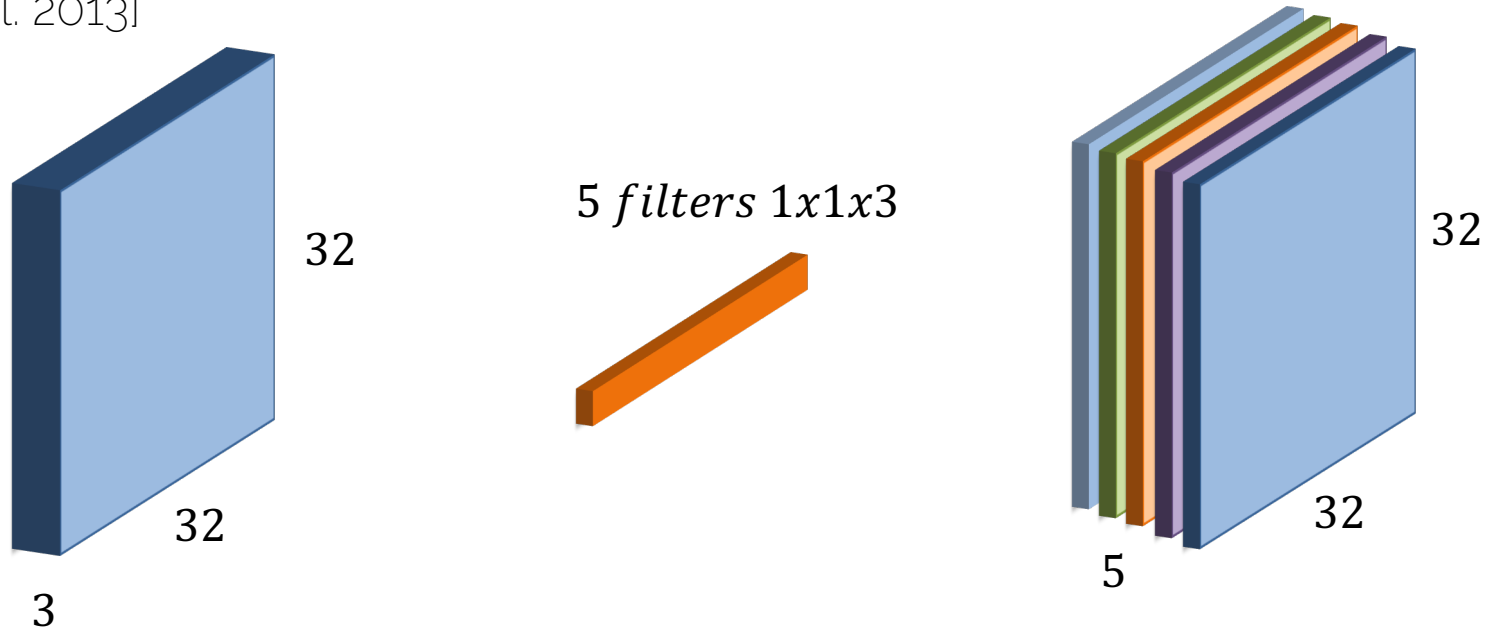


1 output

- Same as having a 3 neuron fully connected layer

1x1 convolution

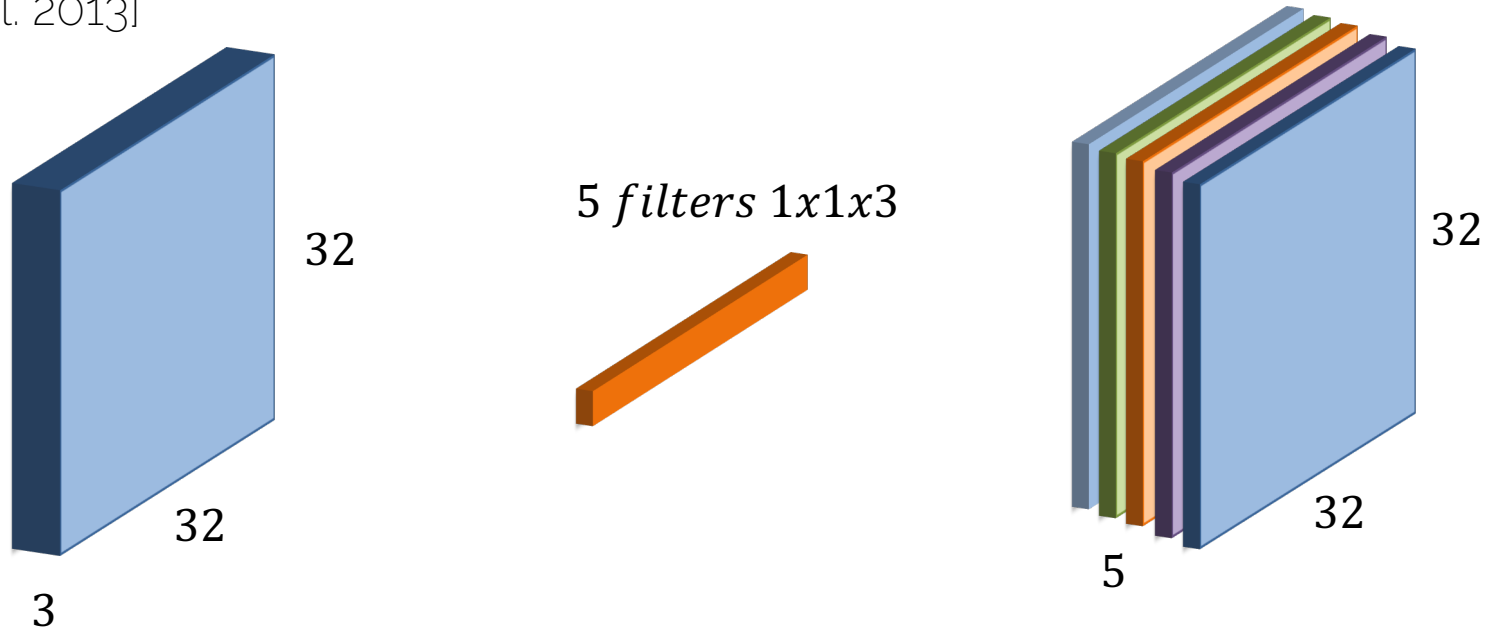
[Li et al. 2013]



- As always we use more convolutional filters

Network in network

[Li et al. 2013]



- As always we use more convolutional filters

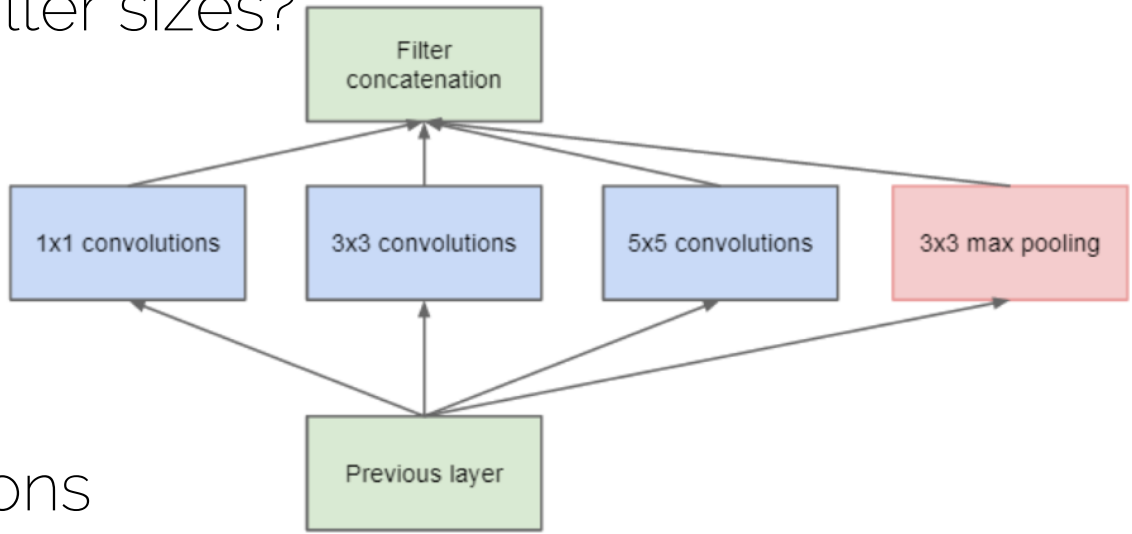
Using 1x1 convolutions

- Use it to shrink the number of channels
- Further adds a non-linearity → one can learn more complex functions

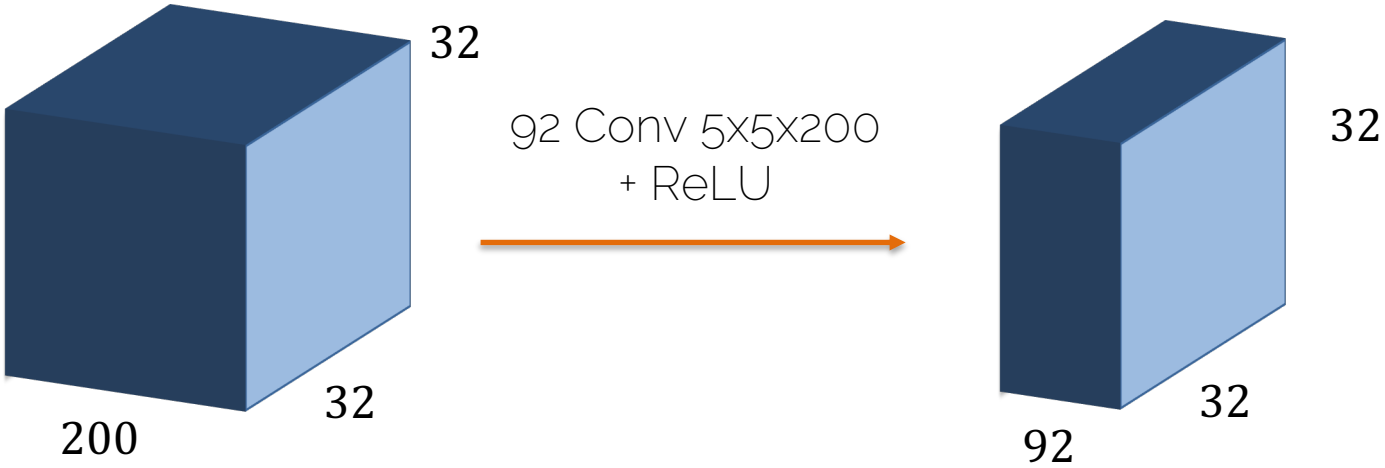
Inception layer

Inception layer

- Tired of choosing filter sizes?
- Use them all!
- All same convolutions
- 3x3 max pooling is with stride 1



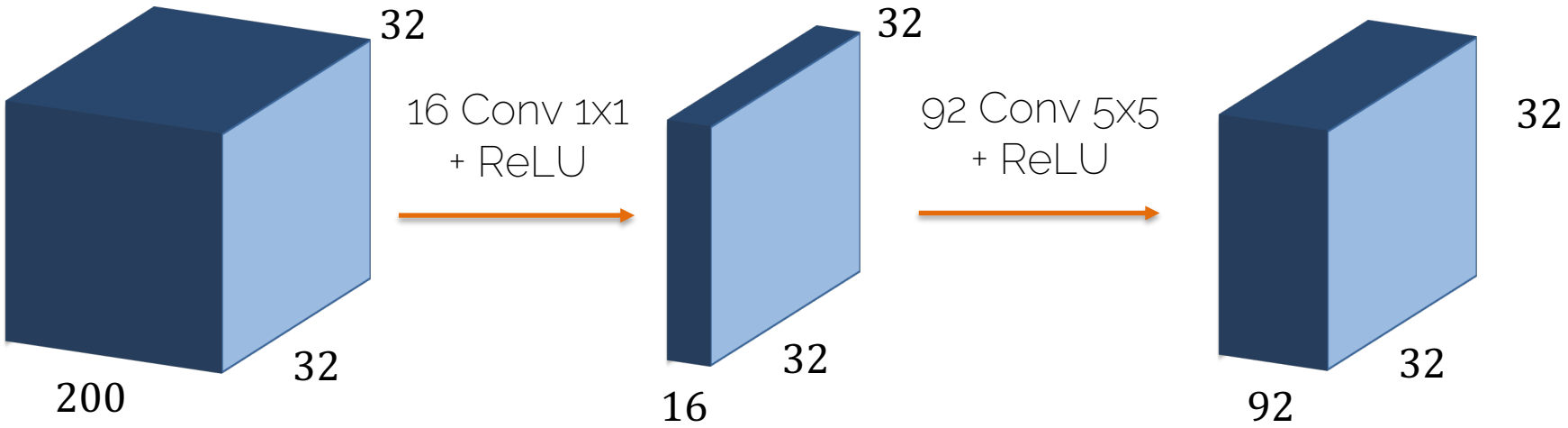
Inception layer: computational cost



Multiplications: 5x5x200 x 32x32x92 ~ 470 million

1 value of the output volume

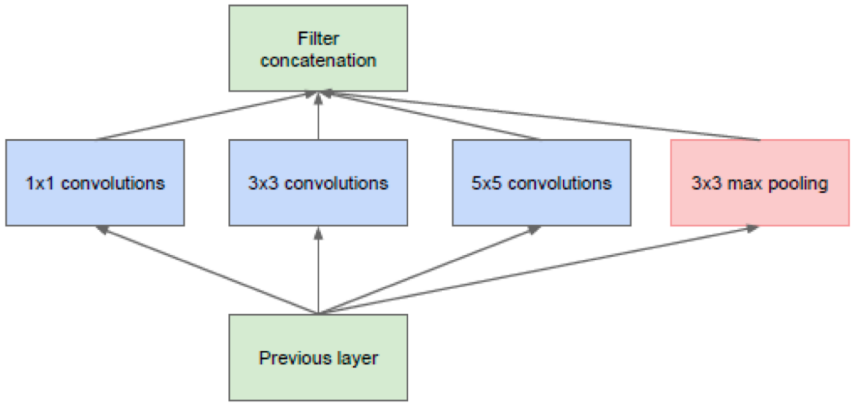
Inception layer: computational cost



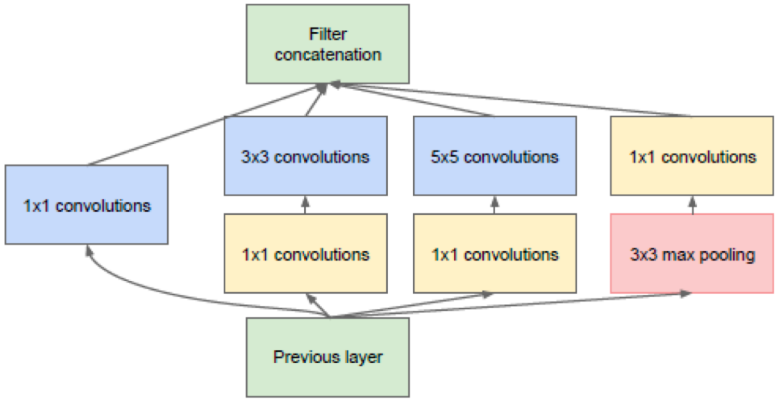
Multiplications: 1x1x200x32x32x16 5x5x16x32x32x92 ~ 40 million

Reduction of multiplications by 1/10

Inception layer

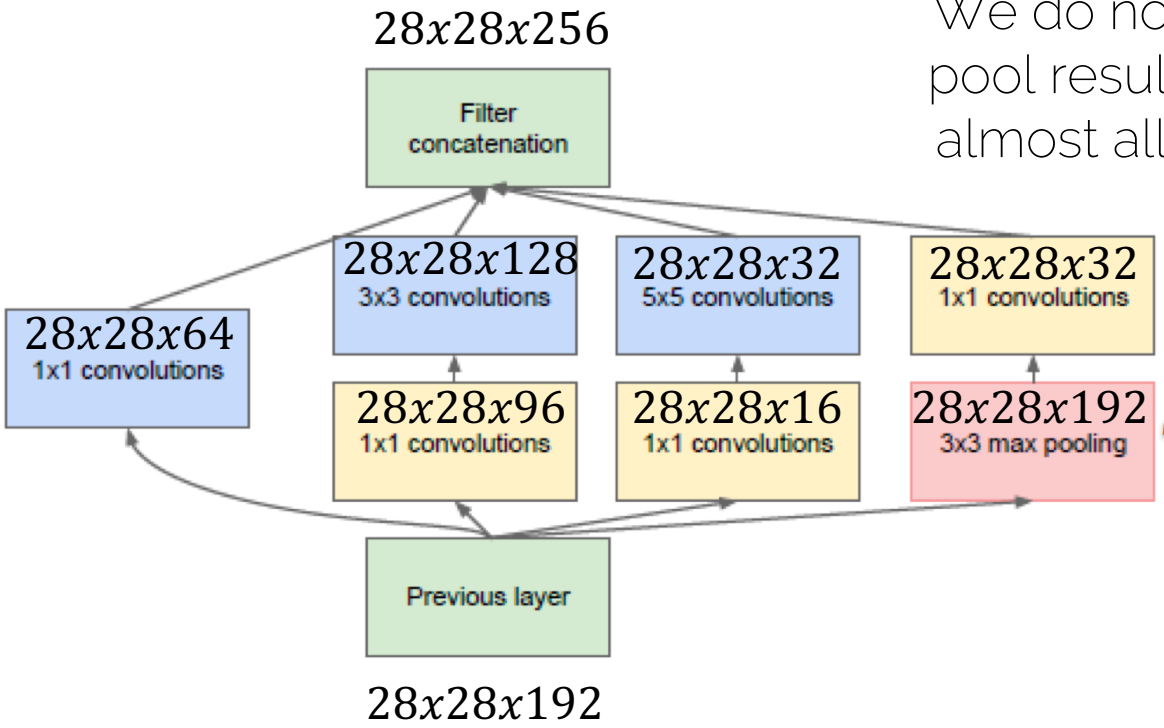


(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

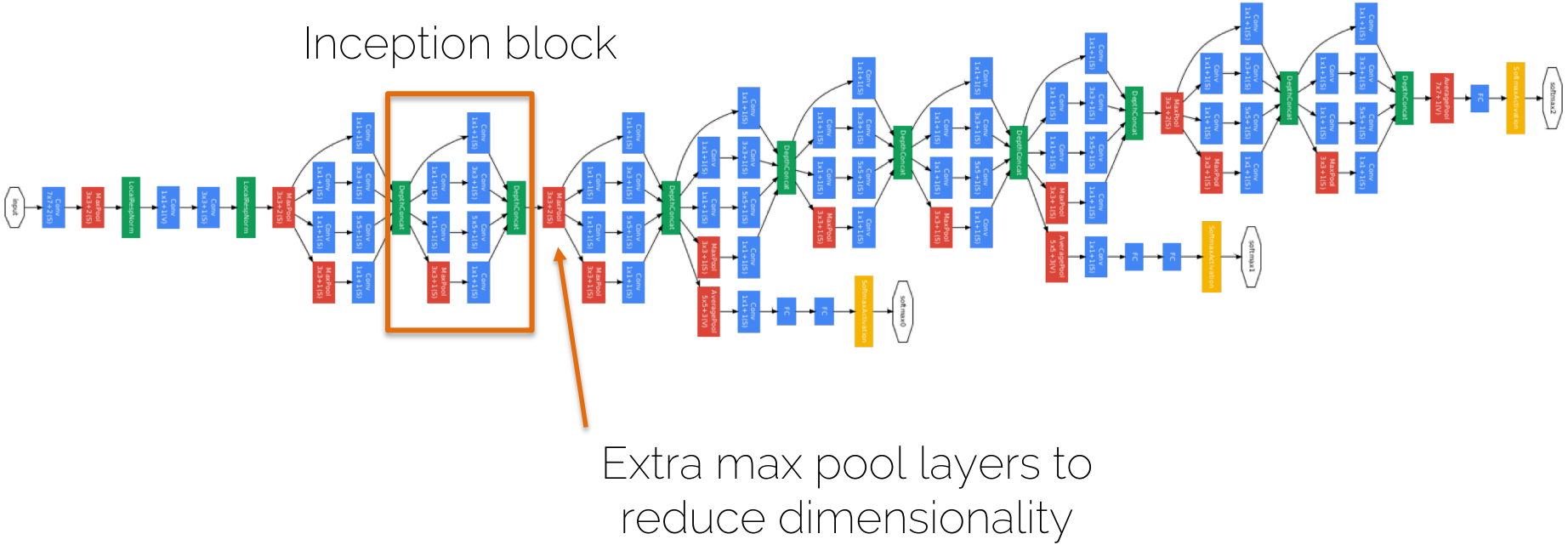
Inception layer: dimensions



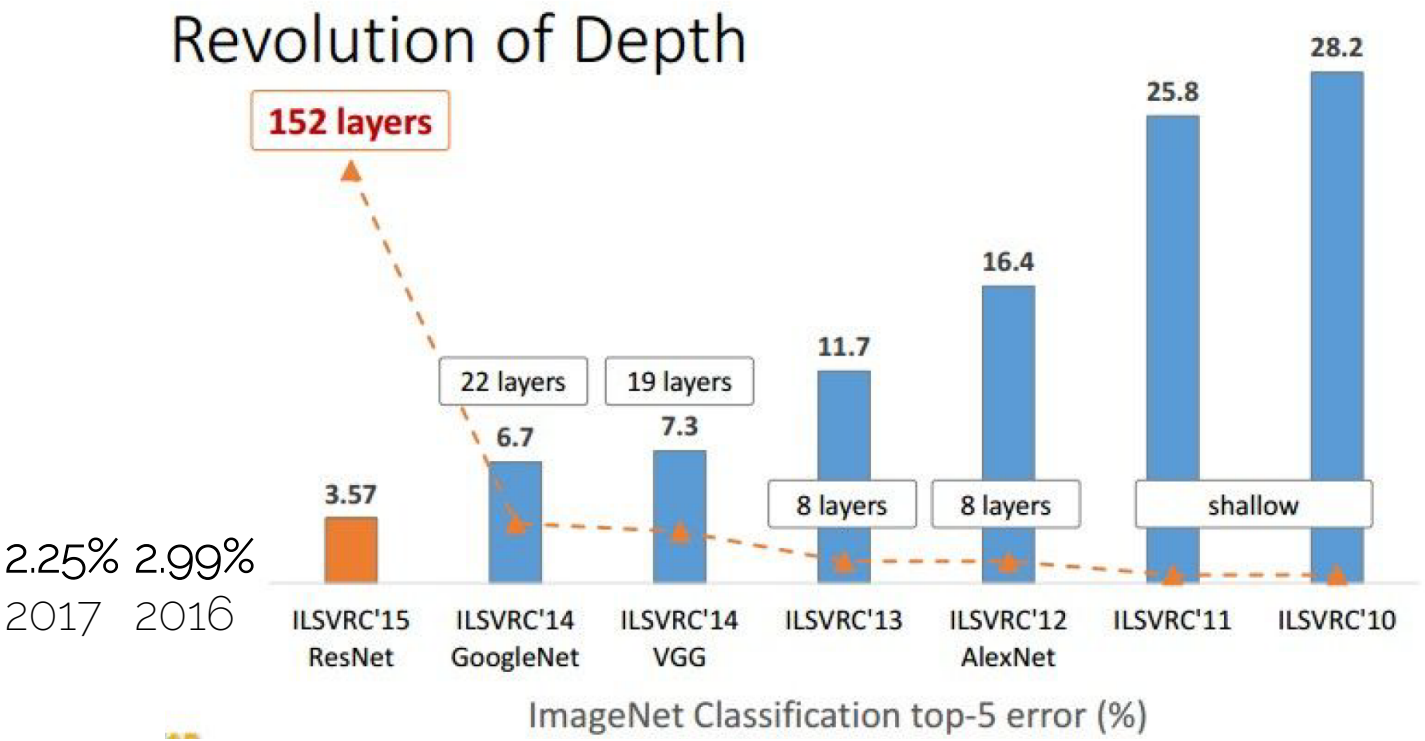
We do not want max pool result to take up almost all the output

GoogLeNet: using the inception layer

[Szegedy et al. 2014]



CNN Architectures



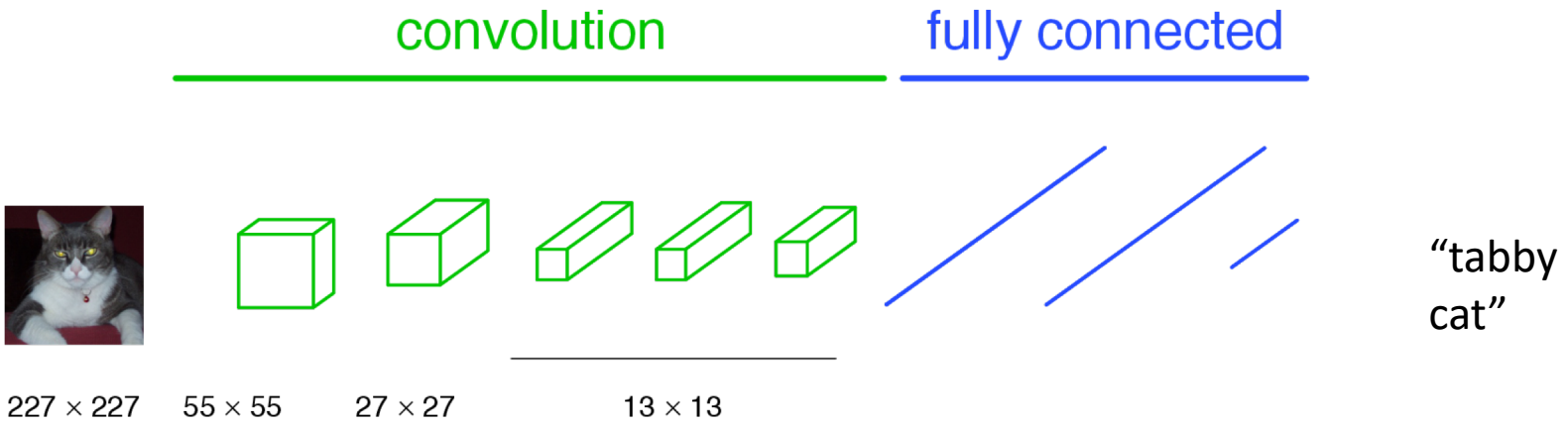
History of Conv Nets

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

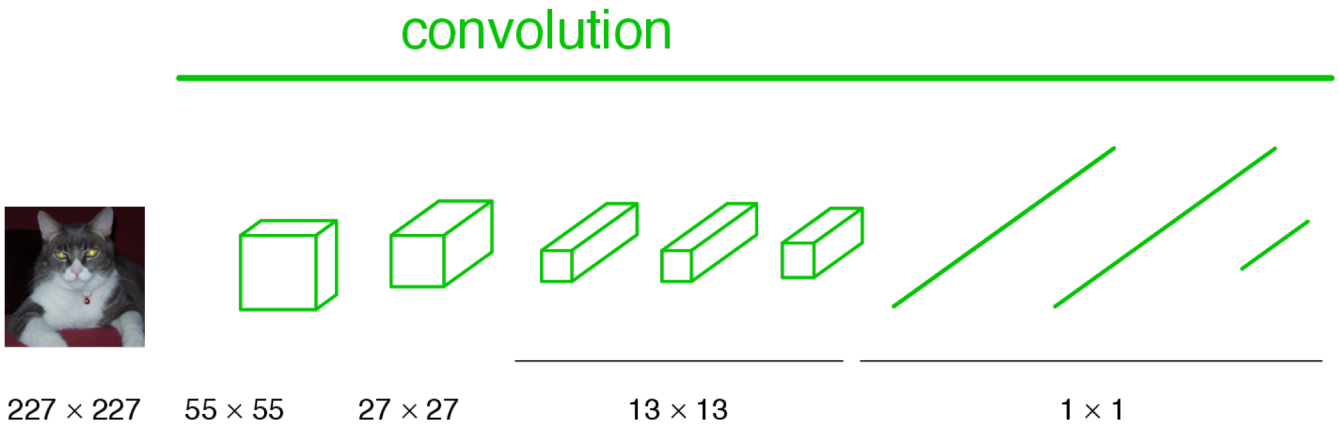
- 'Advanced' Architectures
  - GoogLeNet [Szegedy et al. 14]
  - ResNet [He et al. 15]
  - XceptionNet [Chollet 17]

Fully convolutional network

Classification network

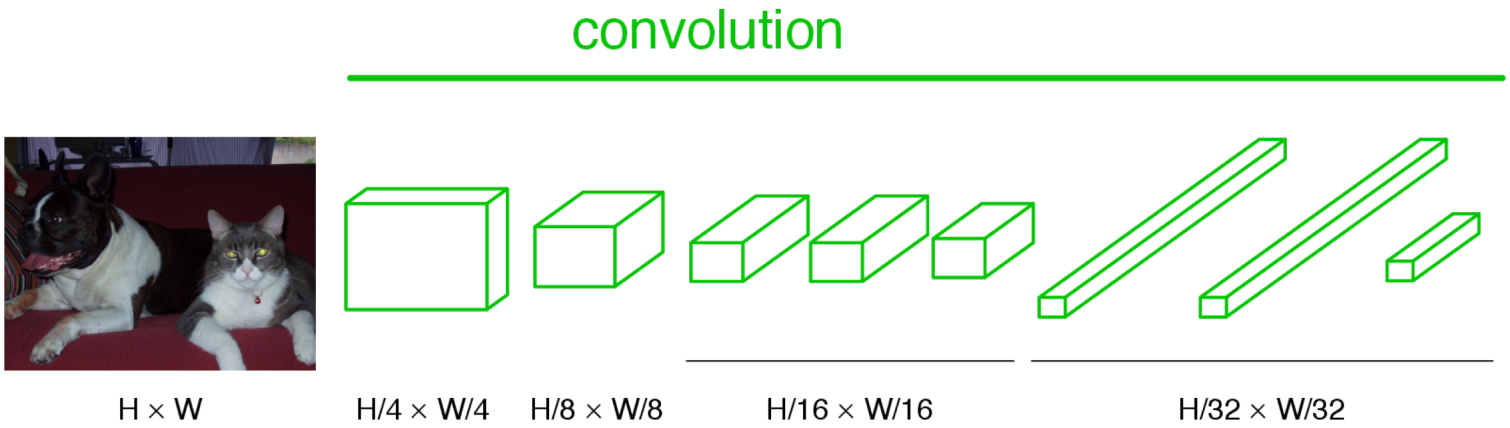


FCN: Becoming Fully Convolutional

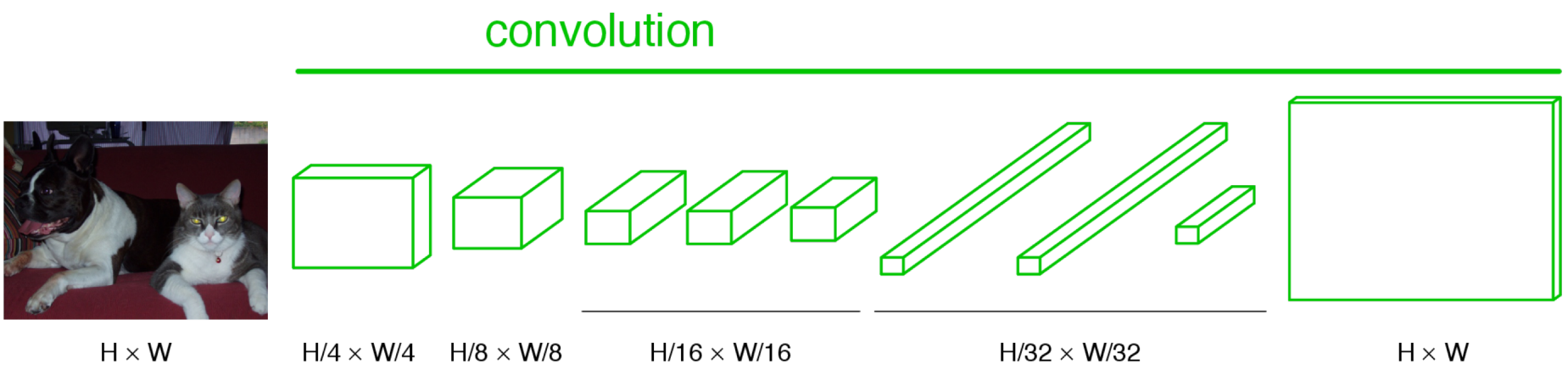


Convert fully connected layers to convolutional layers!

FCN: Becoming Fully Convolutional

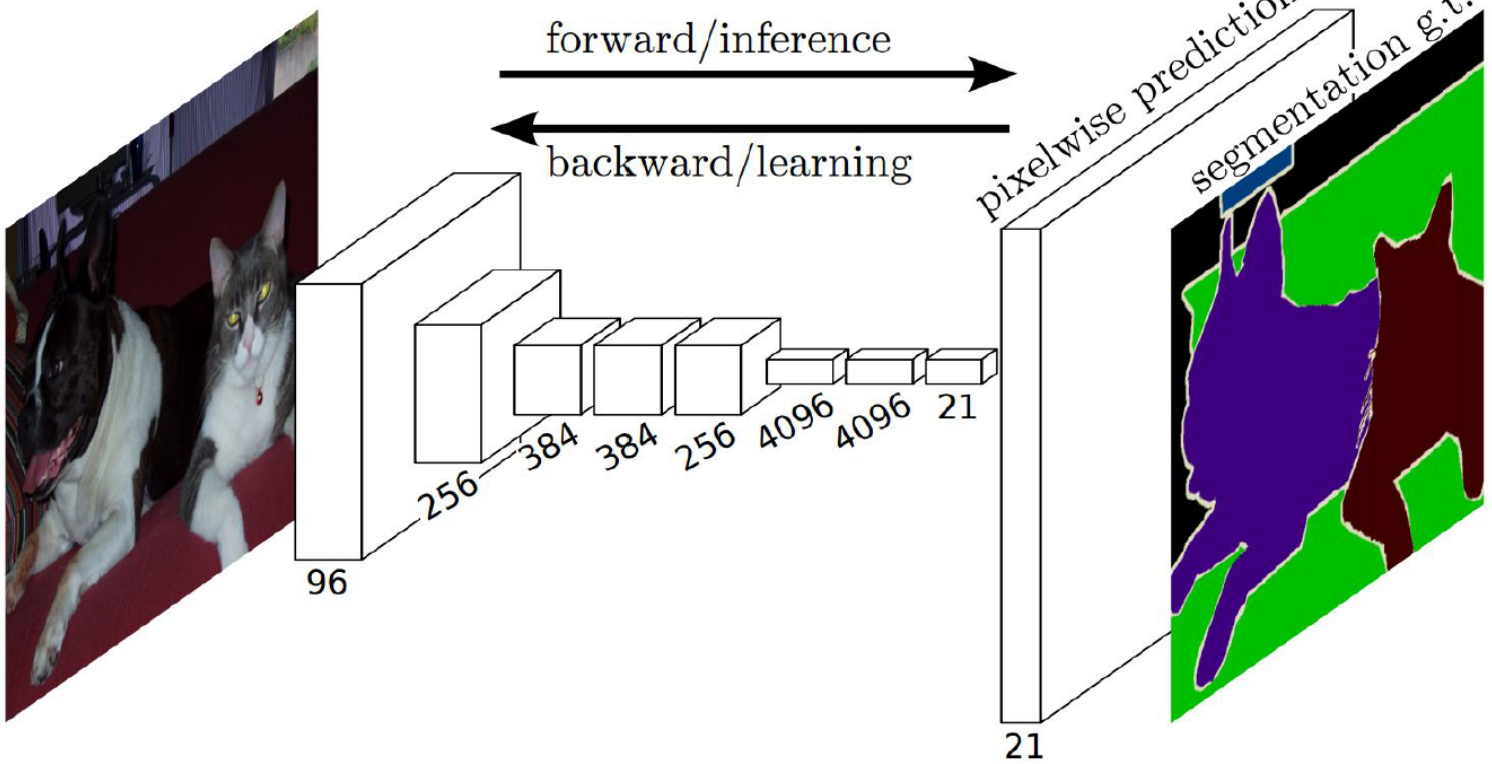


FCN: Upsampling Output



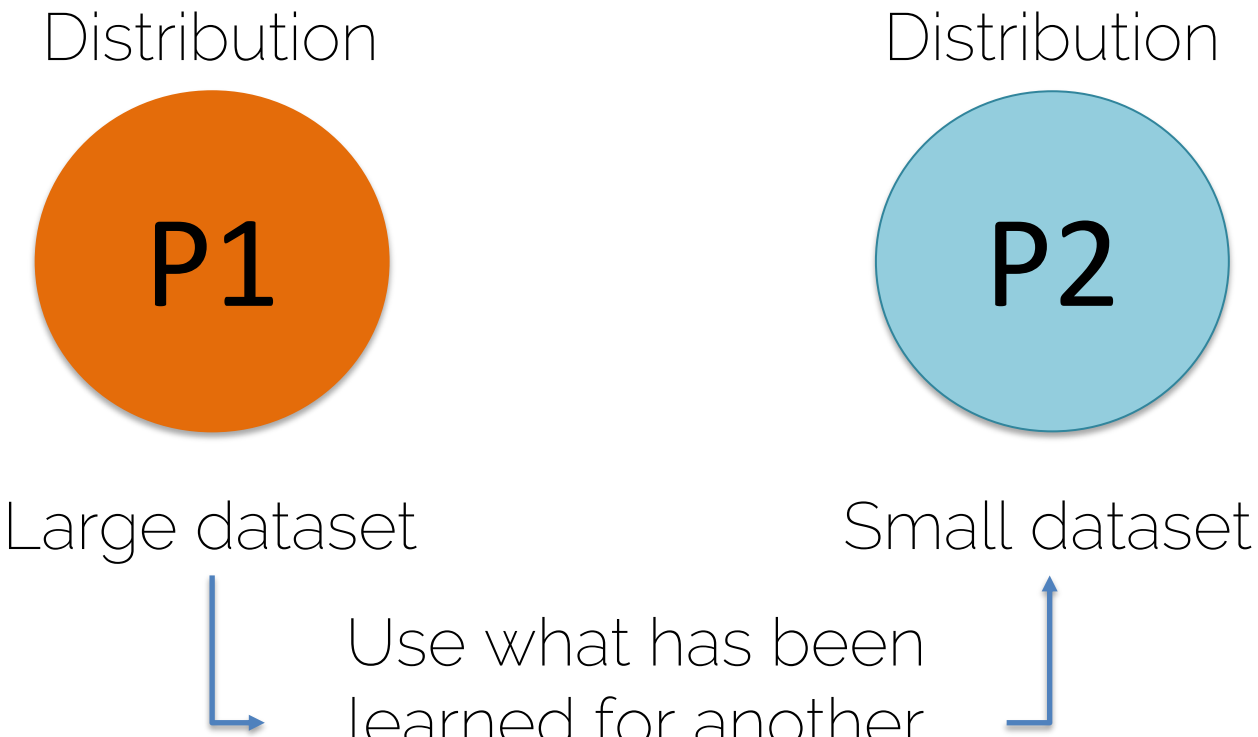
Semantic Segmentation (FCN)

[Long et al. 15] Fully Convolutional Networks for Semantic Segmetnation (FCN)

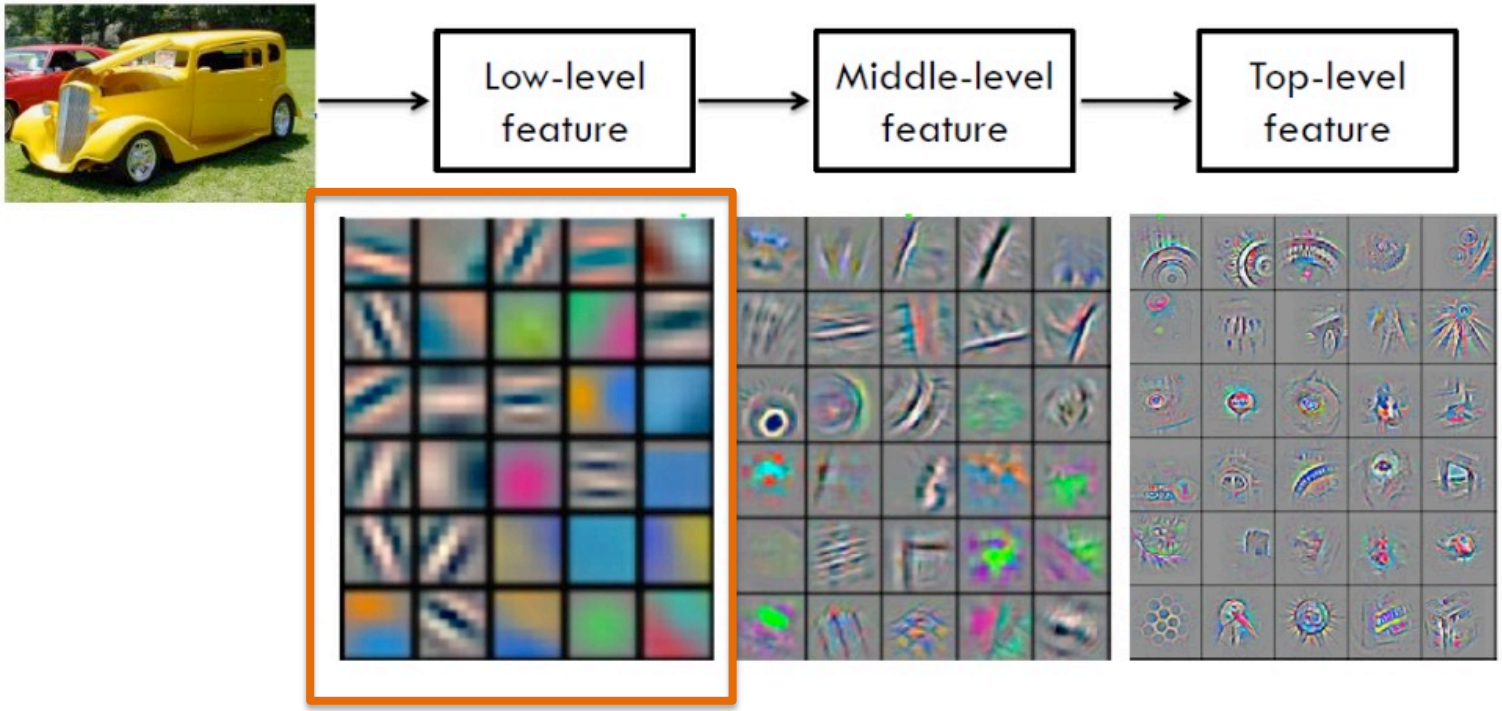


Transfer learning

Transfer learning



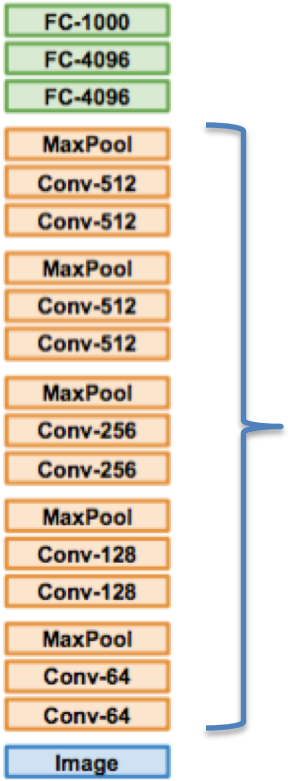
Transfer learning for images



Zeiler and Fergus 2013

Transfer learning

Trained on ImageNet

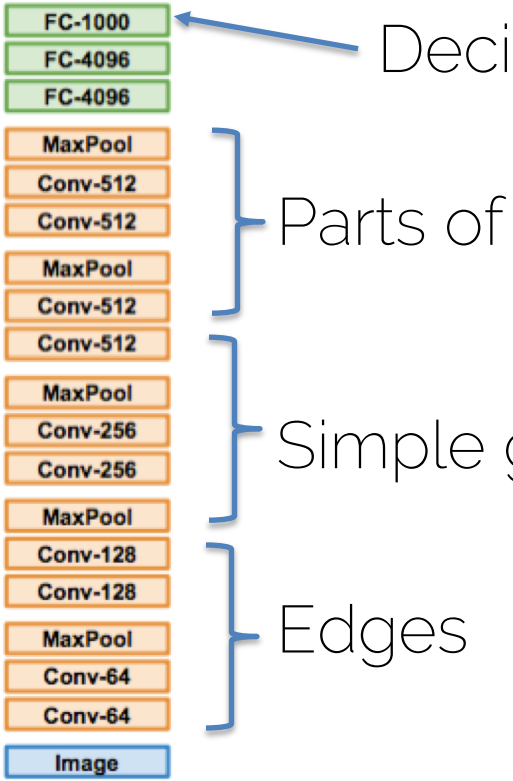


Feature extraction

Donahue 2014, Razavian 2014

Transfer learning

Trained on ImageNet



Decision layers

Parts of an object (wheel, window)

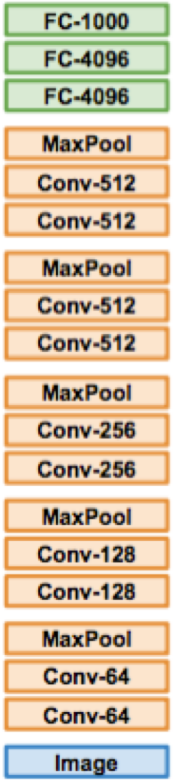
Simple geometrical shapes (circles, etc)

Edges

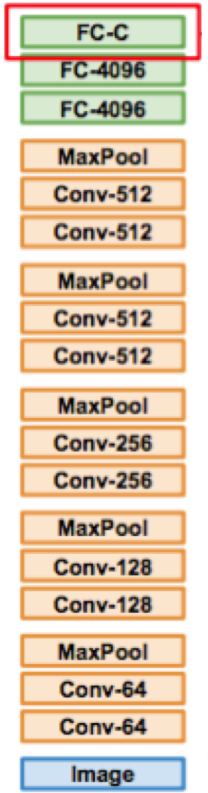
Donahue 2014, Razavian 2014

Transfer learning

Trained on ImageNet



TRAIN



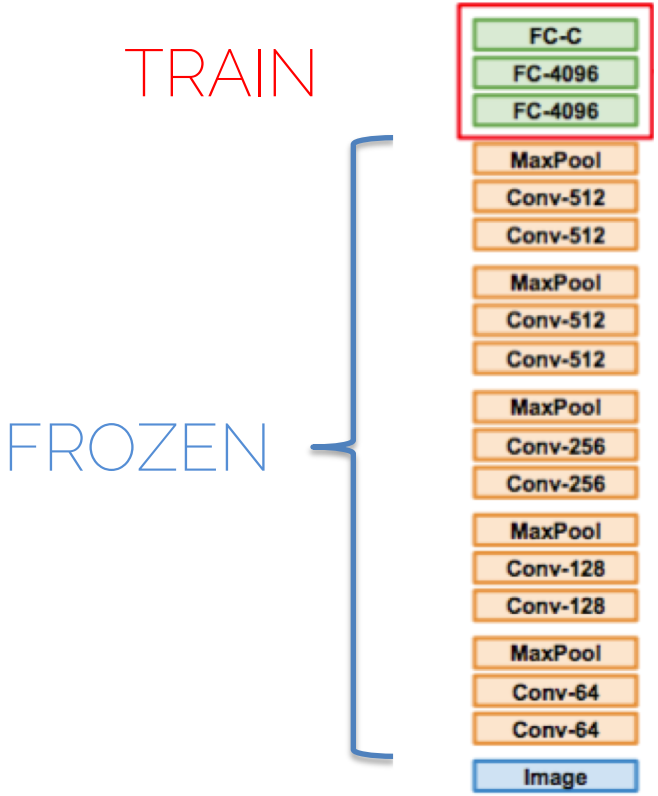
New dataset with C classes

FROZEN

Donahue 2014, Razavian 2014

Transfer learning

If the dataset is big enough train more layers with a low learning rate



Donahue 2014, Razavian 2014

When Transfer Learning makes sense

- When task P1 and P2 have the same input (e.g. an RGB image)
- When you have more data for task P1 than for task P2
- When the low-level features for P1 could be useful to learn P2

Administrative Things

- Next Thursday: solutions for exercise 3 plus some guidance for exercise 4
- Deadline for exercise 4: **04.07.2018, 23.59**
- Next Monday, 2nd July: Guest lecture!
- 9th July: last lecture, Recurrent Neural Networks