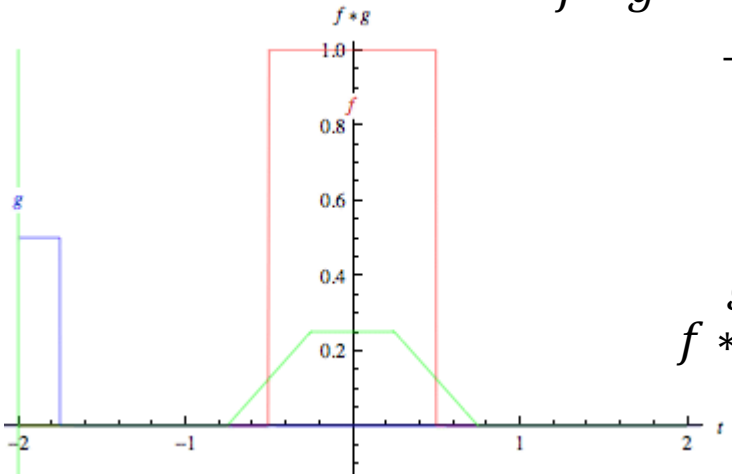
Lecture 9 Recap

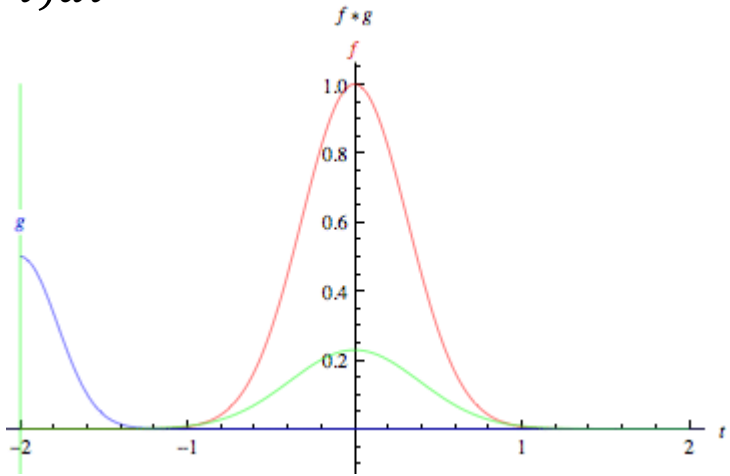
What are Convolutions?

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



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

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	1	8
-7	9	2
-5	-9	3

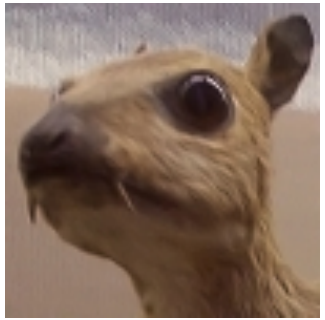
$$5 \cdot 4 + (-1) \cdot 3 + (-1) \cdot 4 + (-1) \cdot 9 + (-1) \cdot 1 =$$
$$20 - 17 = 3$$

We aim to learn those

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!

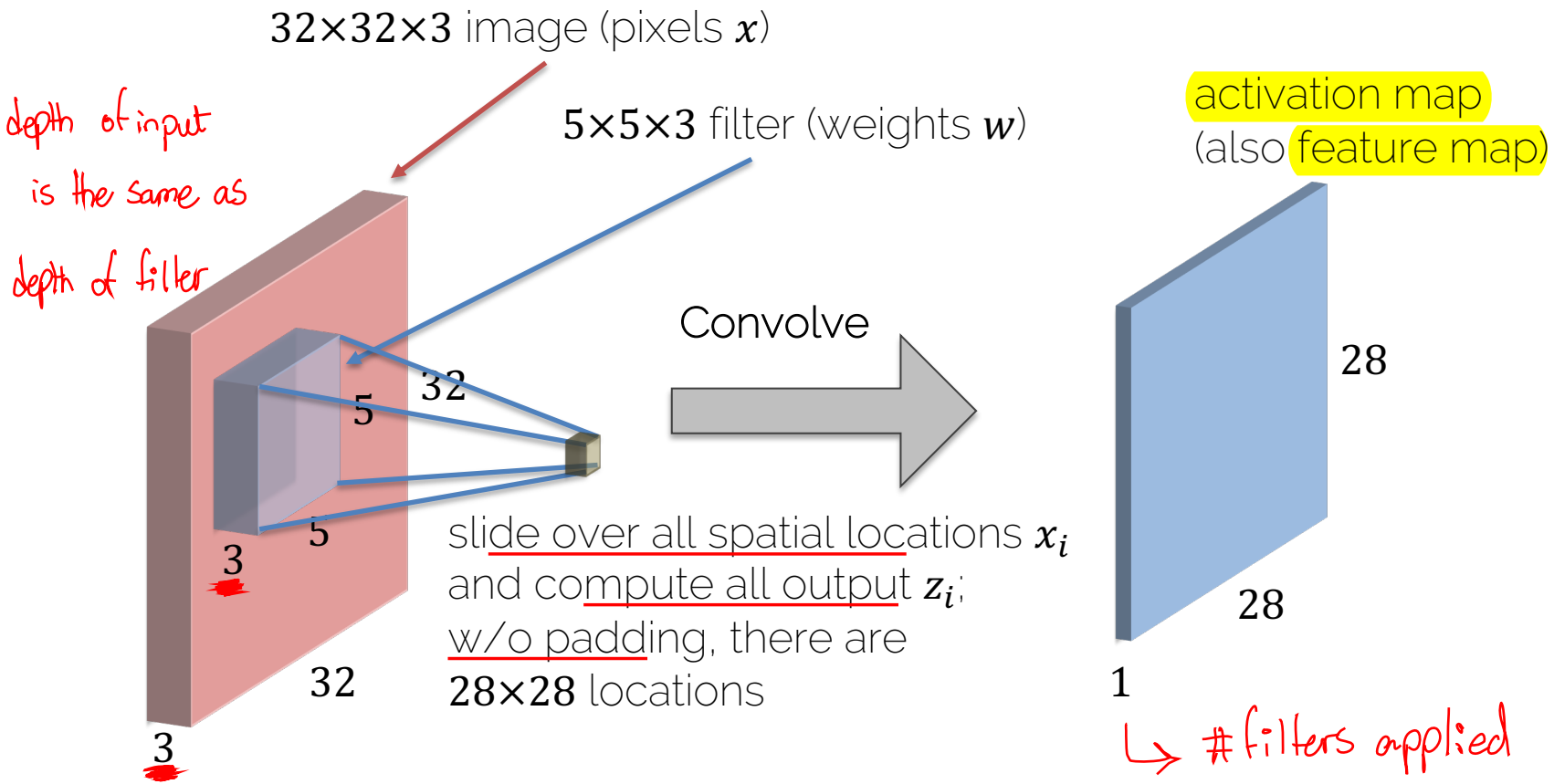
Sharpen

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

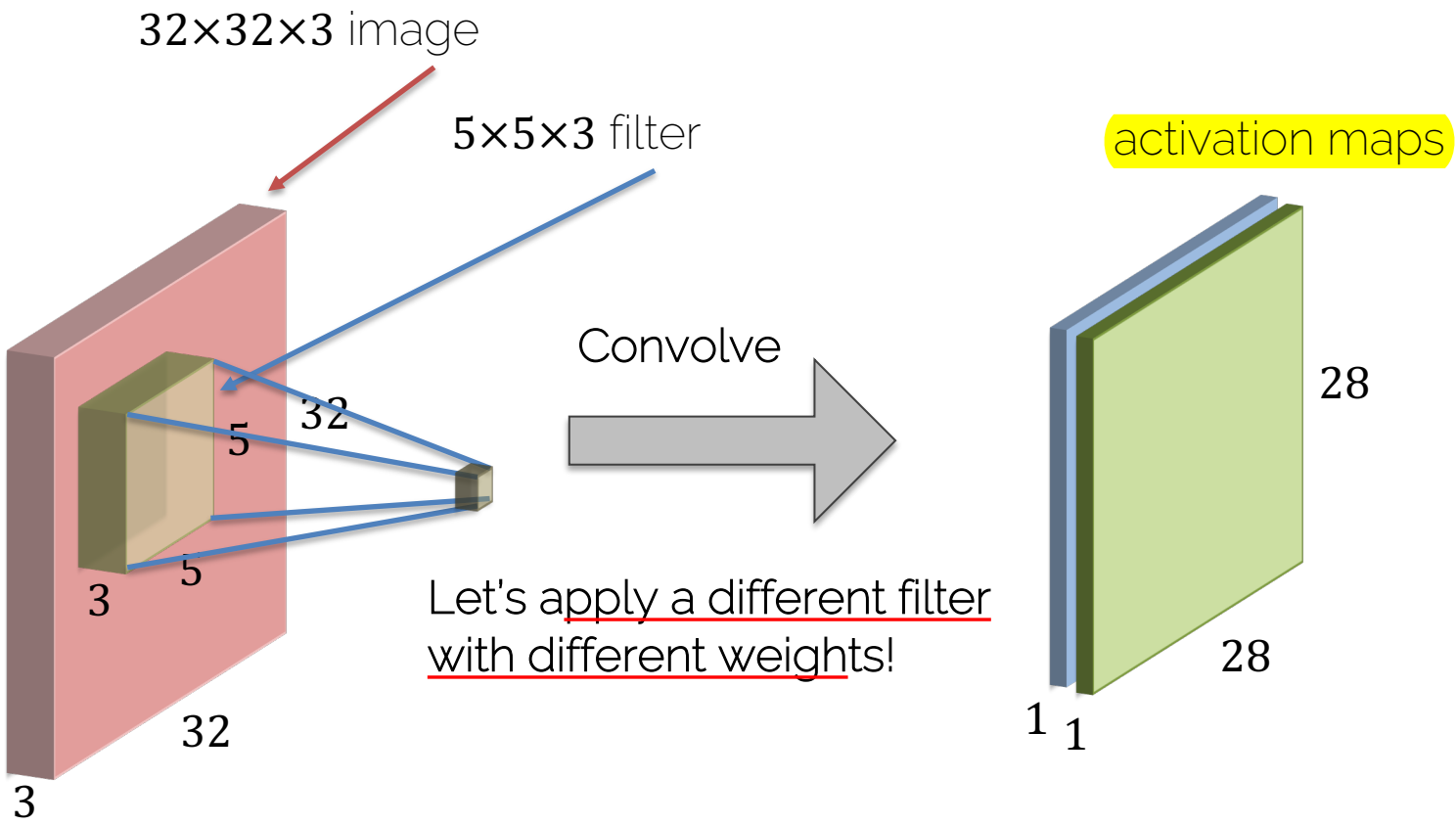
Gaussian blur

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

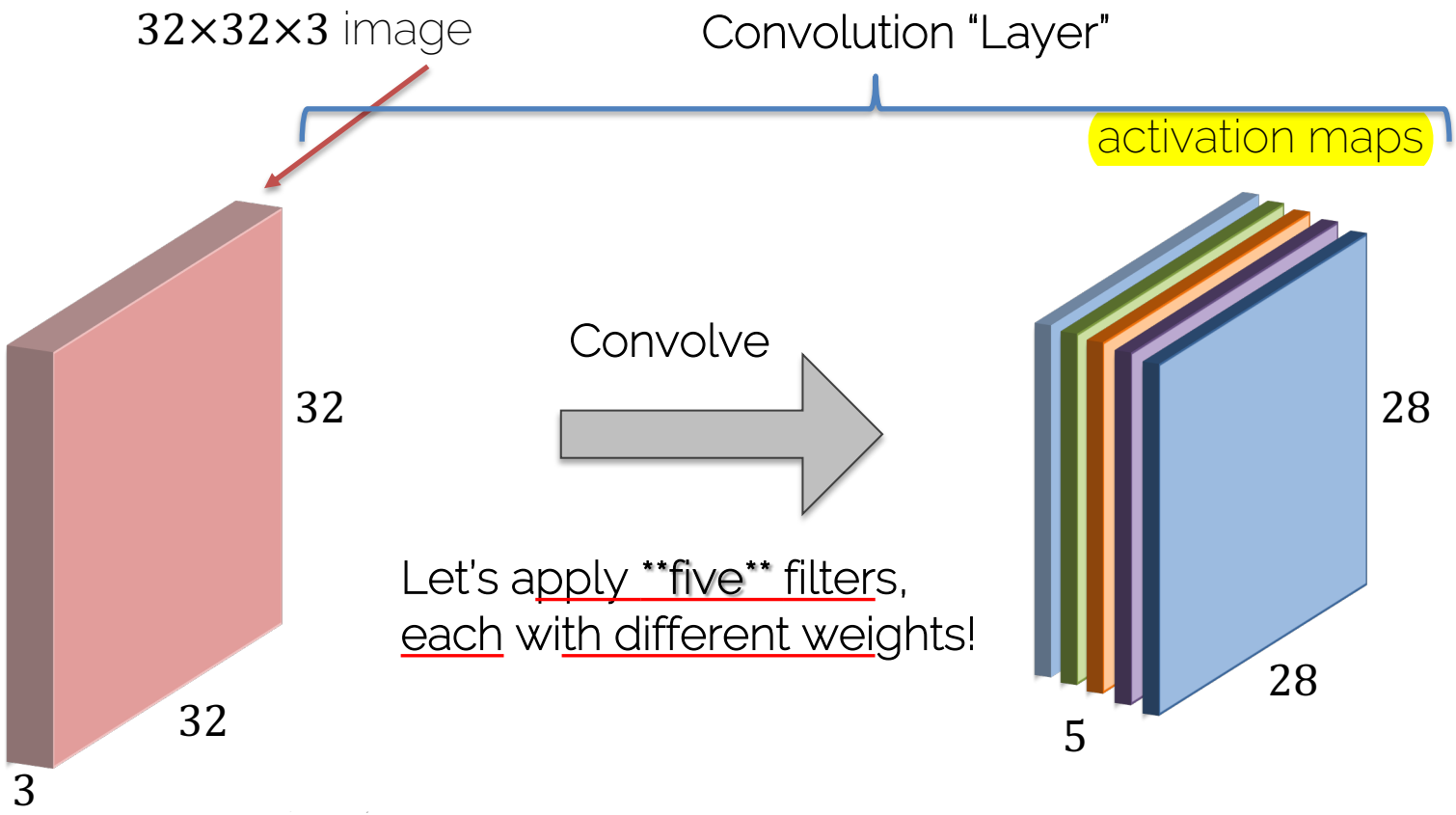
Convolutions on RGB Images



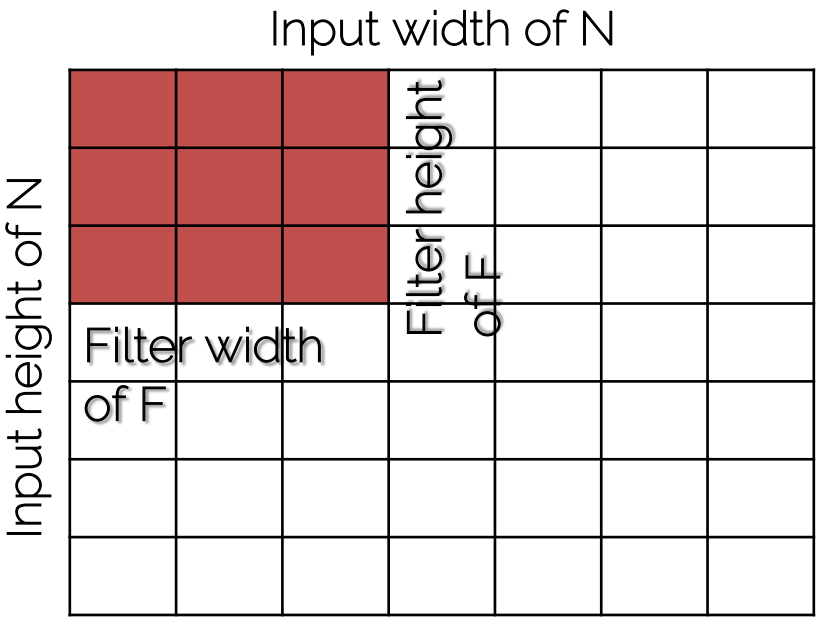
Convolution Layer



NB Features extracted are put into the depth of activation maps



Convolution Layers: Dimensions



Input: $N \times N$

Filter: $F \times F$

Stride: S

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

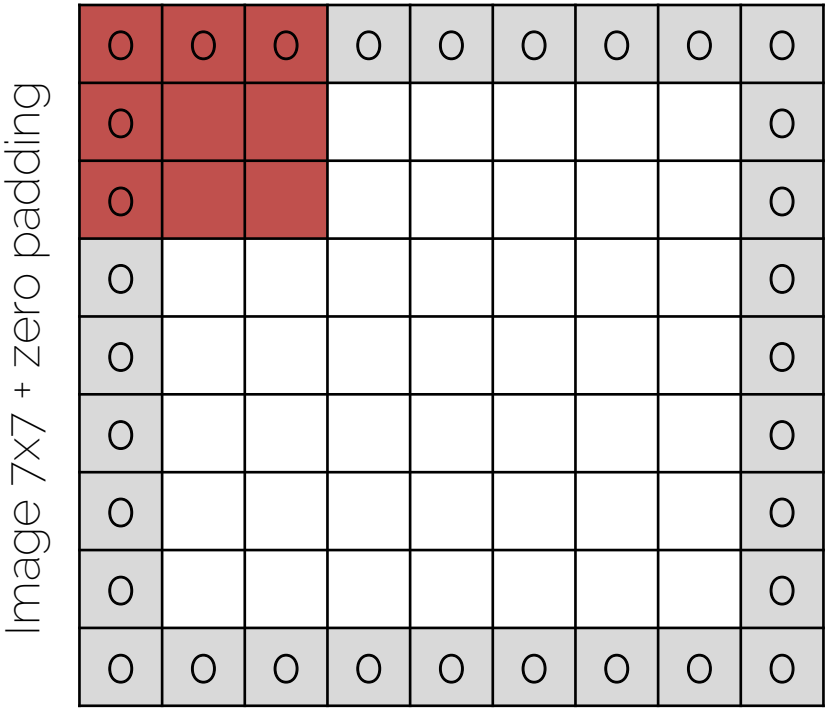
$N = 7, F = 3, S = 1$: $\frac{7-3}{1} + 1 = 5$

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

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

Fractions are illegal

Convolution Layers: Padding



Types of convolutions:

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

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

Convolution Layers: Dimensions

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

REMARK: in practice, typically integer division is used (i.e., apply the floor–operator!)

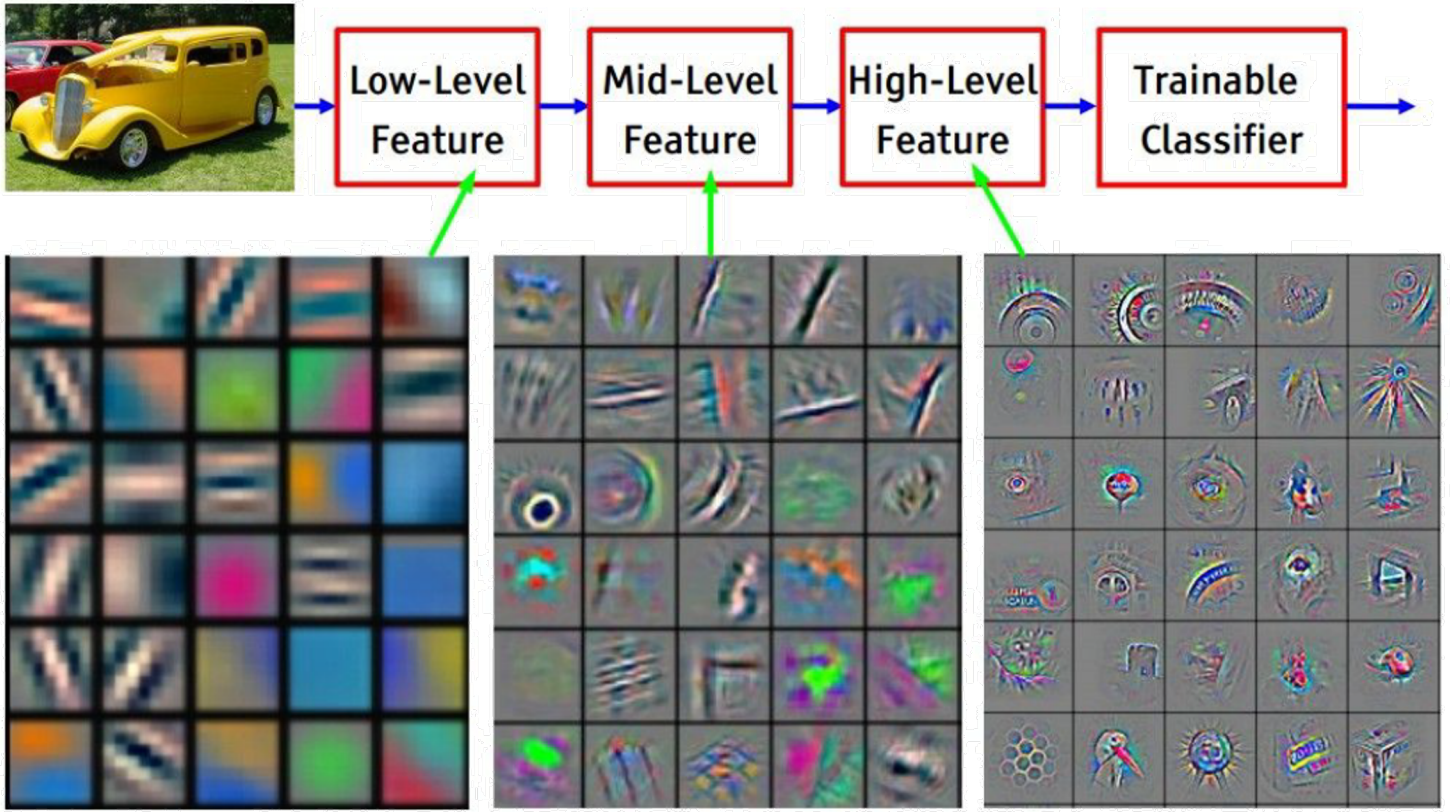
Example: 3x3 conv with same padding and strides of 2 on an 64x64 RGB image -> N = 64, F = 3, P = 1, S = 2

Output: $\left(\frac{64+2\cdot 1-3}{2}+1\right)\times\left(\frac{64+2\cdot 1-3}{2}+1\right)$

$= floor(32.5)\times floor(32.5)$

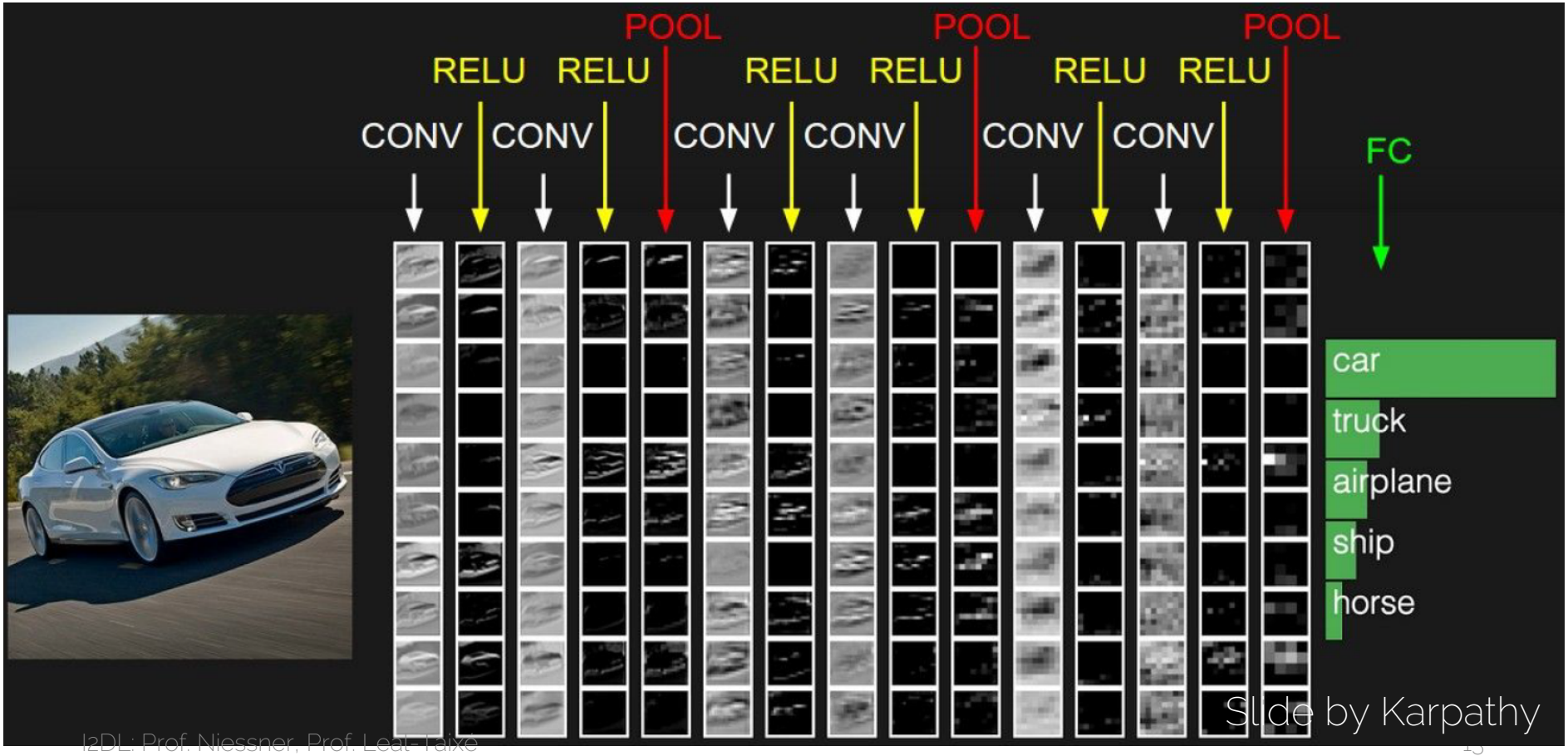
$= 32\times 32$

CNN Learned Filters



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

CNN Prototype

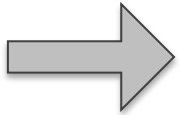


Pooling Layer: Max Pooling

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**

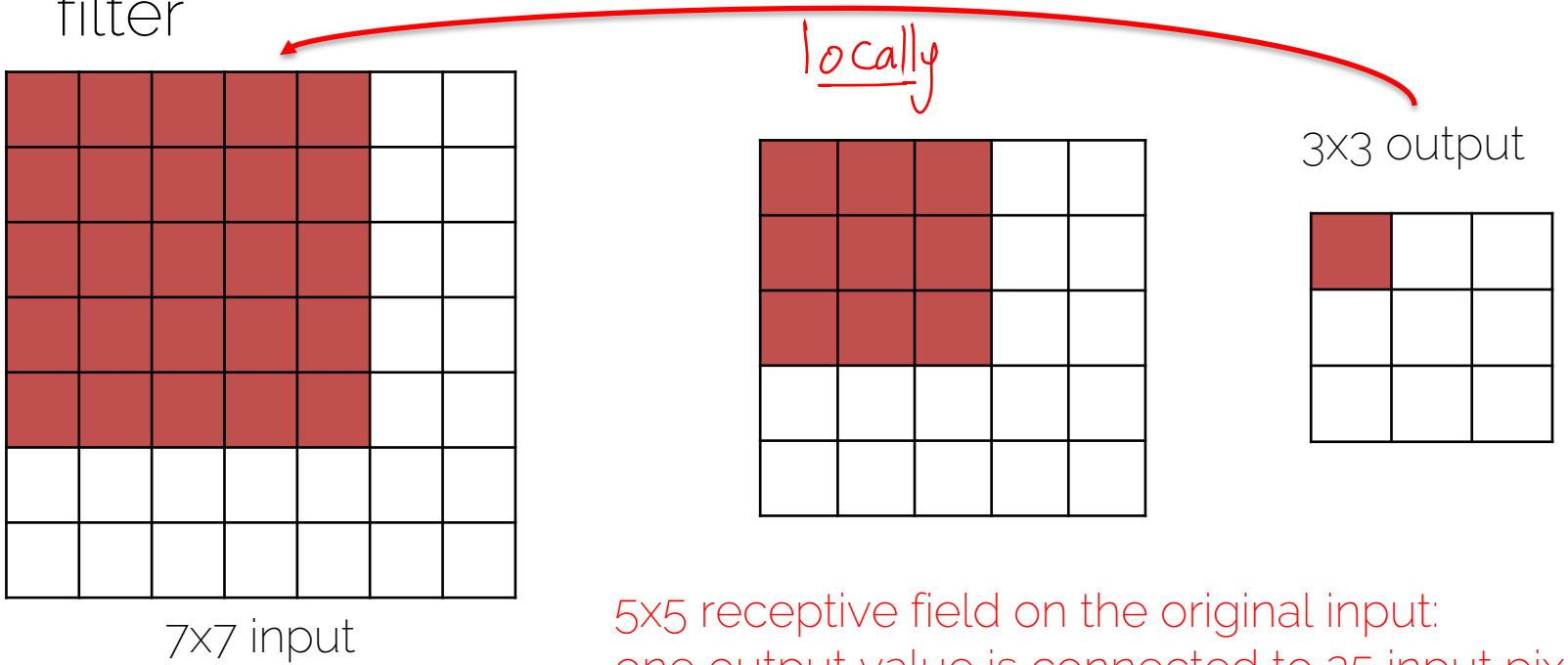


'Pooled' output

6	9
3	4

Receptive Field

- Spatial extent of the connectivity of a convolutional filter



5x5 receptive field on the original input:
one output value is connected to 25 input pixels

Lecture 10 – CNNs (part 2)

* Classic Architectures
* Skip Connections
* 1X1 Conv
* Inception layer

Classic Architectures

LeNet

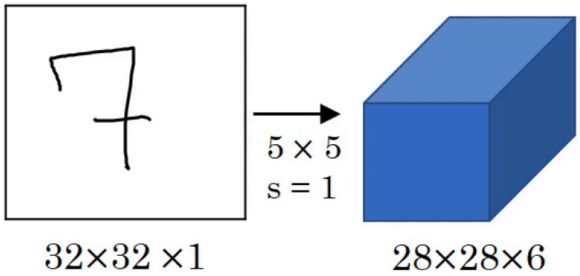
- Digit recognition: 10 classes

Input: 32×32 grayscale images
This one: Labeled as class "7"

[LeCun et al. '98] LeNet

LeNet

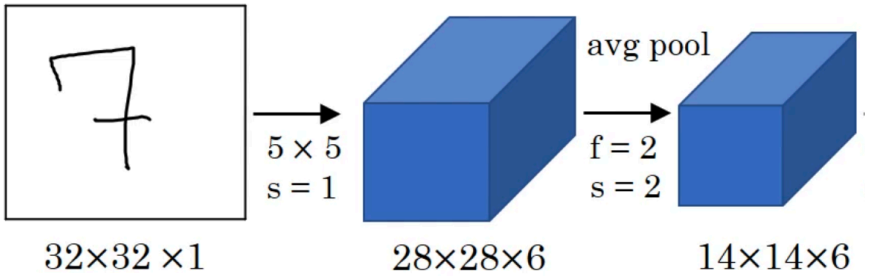
- Digit recognition: 10 classes



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

LeNet

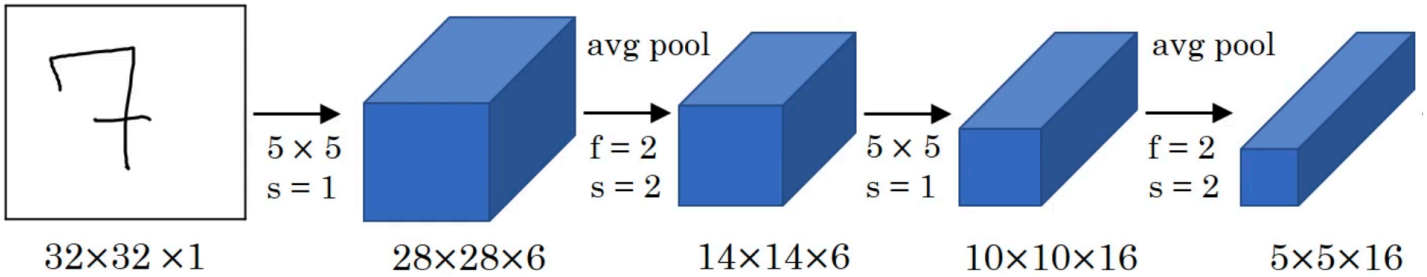
- Digit recognition: 10 classes



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

LeNet

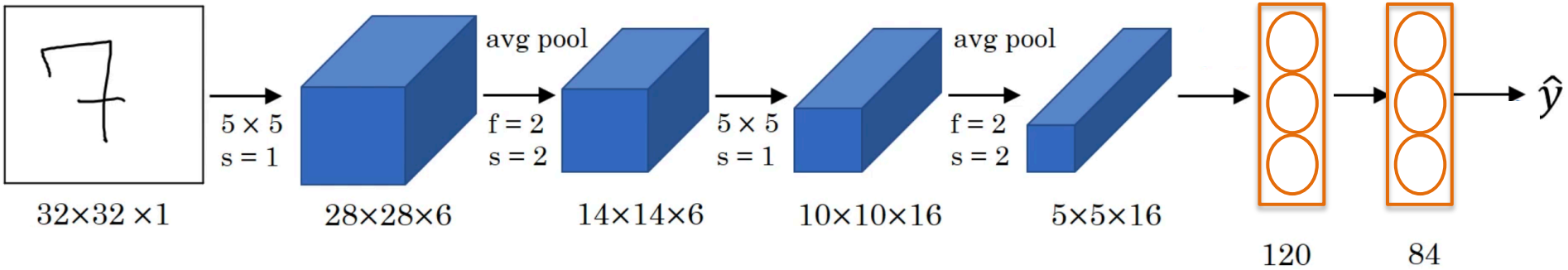
- Digit recognition: 10 classes



- Again valid convolutions, how many filters?

LeNet

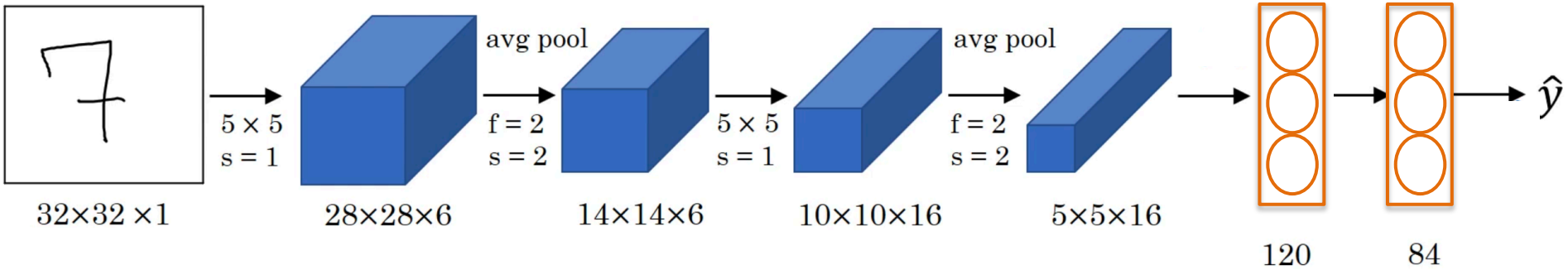
- Digit recognition: 10 classes



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

LeNet

- Digit recognition: 10 classes

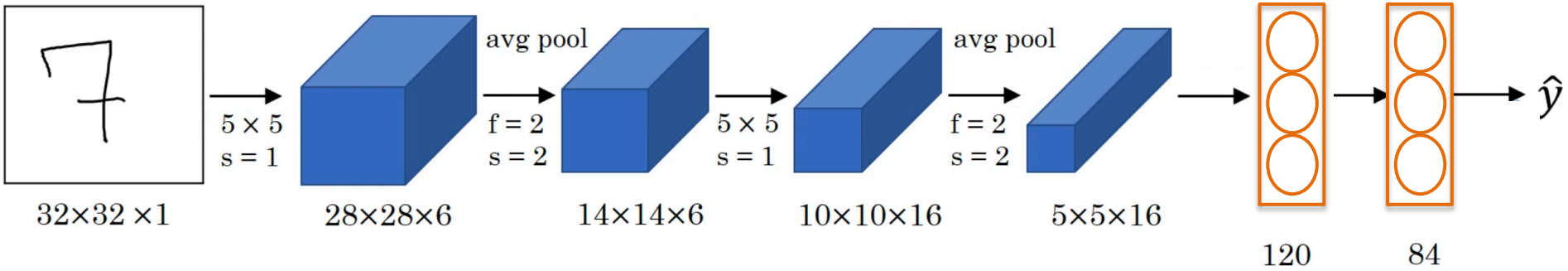


- Conv -> Pool -> Conv -> Pool -> Conv -> FC

LeNet

- Digit recognition: 10 classes

60k parameters

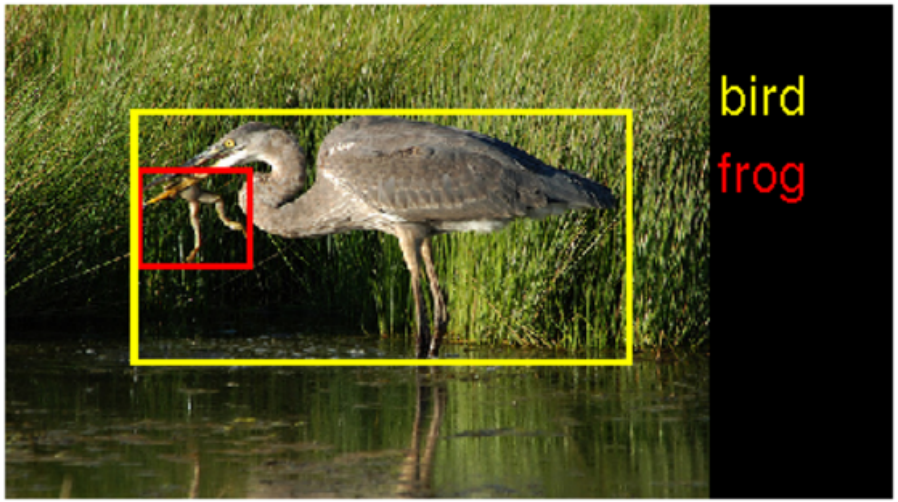


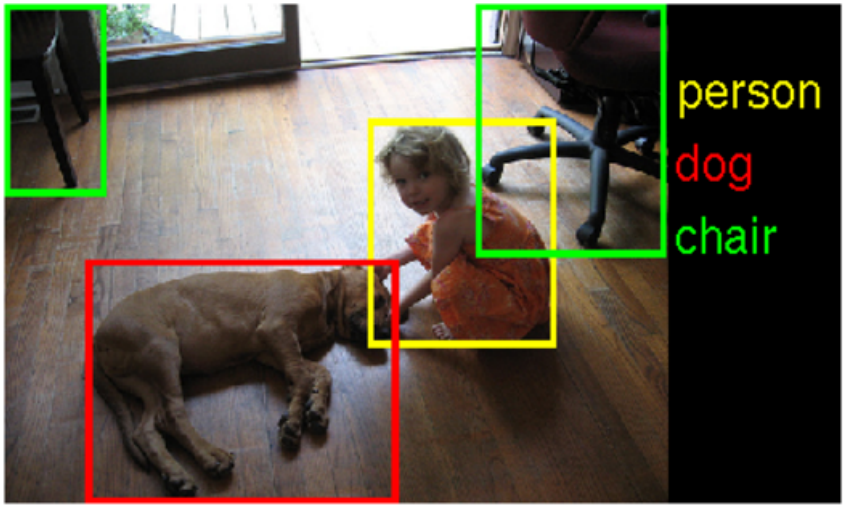
- Conv -> Pool -> Conv -> Pool -> Conv -> FC

As we go deeper: Width, Height ↓ Number of Filters ↑

Test Benchmarks

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





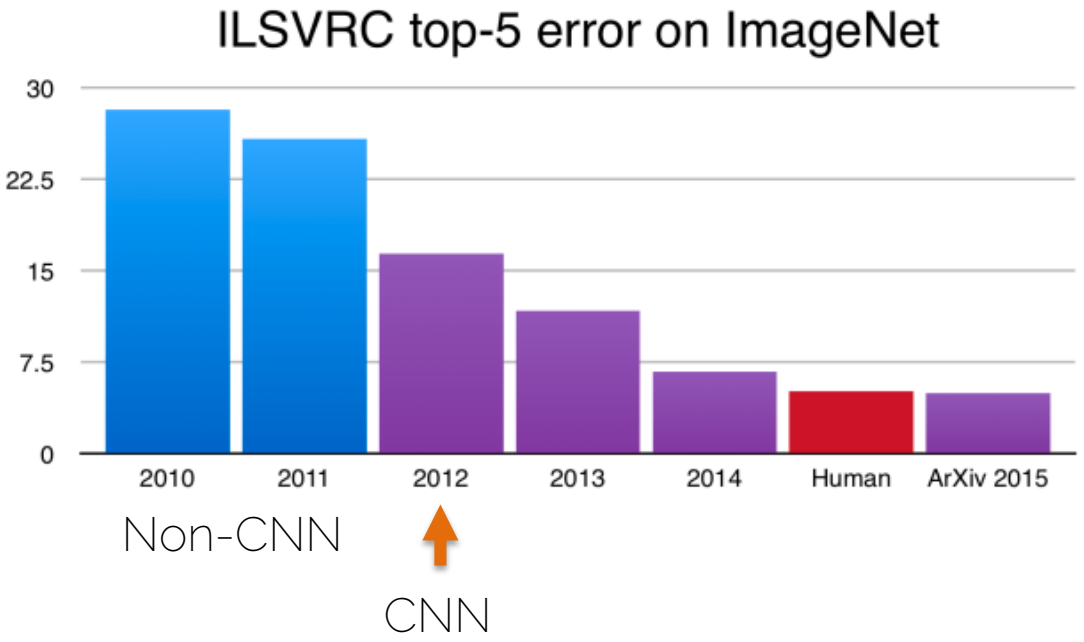
[Russakovsky et al., IJCV'15] "ImageNet Large Scale Visual Recognition Challenge."

Common Performance Metrics

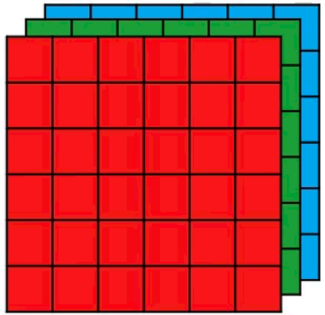
- Top-1 score: check if a sample's top class (i.e. the one with highest probability) is the same as its target label
- Top-5 score: check if your label is in your 5 first predictions (i.e. predictions with 5 highest probabilities)
- → Top-5 error: percentage of test samples for which the correct class was not in the top 5 predicted classes

AlexNet

- Cut ImageNet error down in half

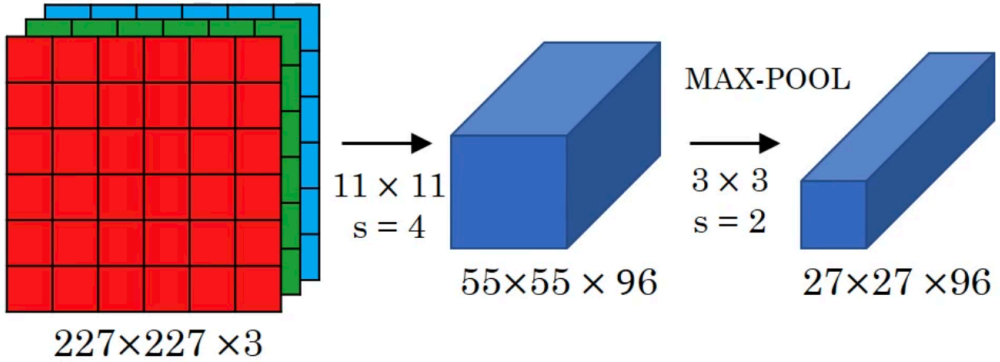


AlexNet



[Krizhevsky et al. NIPS'12] AlexNet

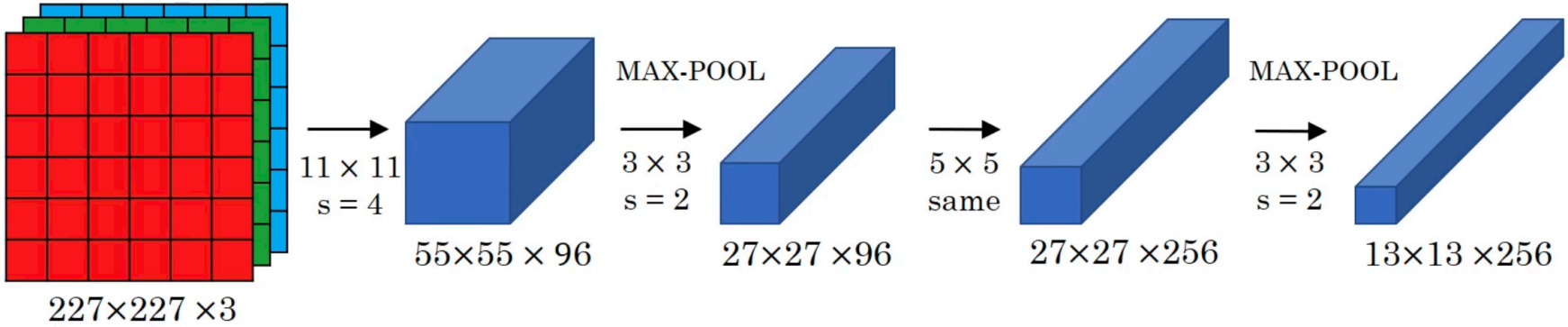
AlexNet



[Krizhevsky et al. NIPS'12] AlexNet

AlexNet

11 X 11 → too large
S=4 → very aggressive

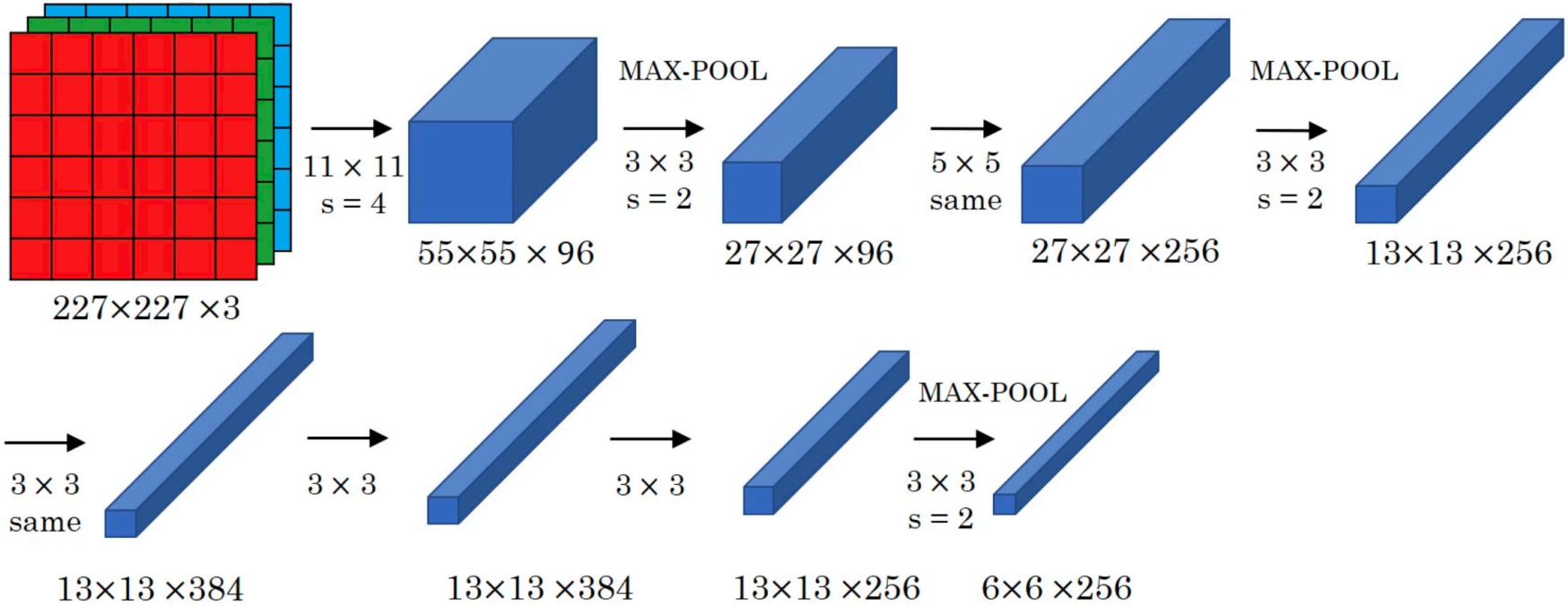


- Use of same convolutions

As with LeNet: Width, Height ↓ Number of Filters ↑

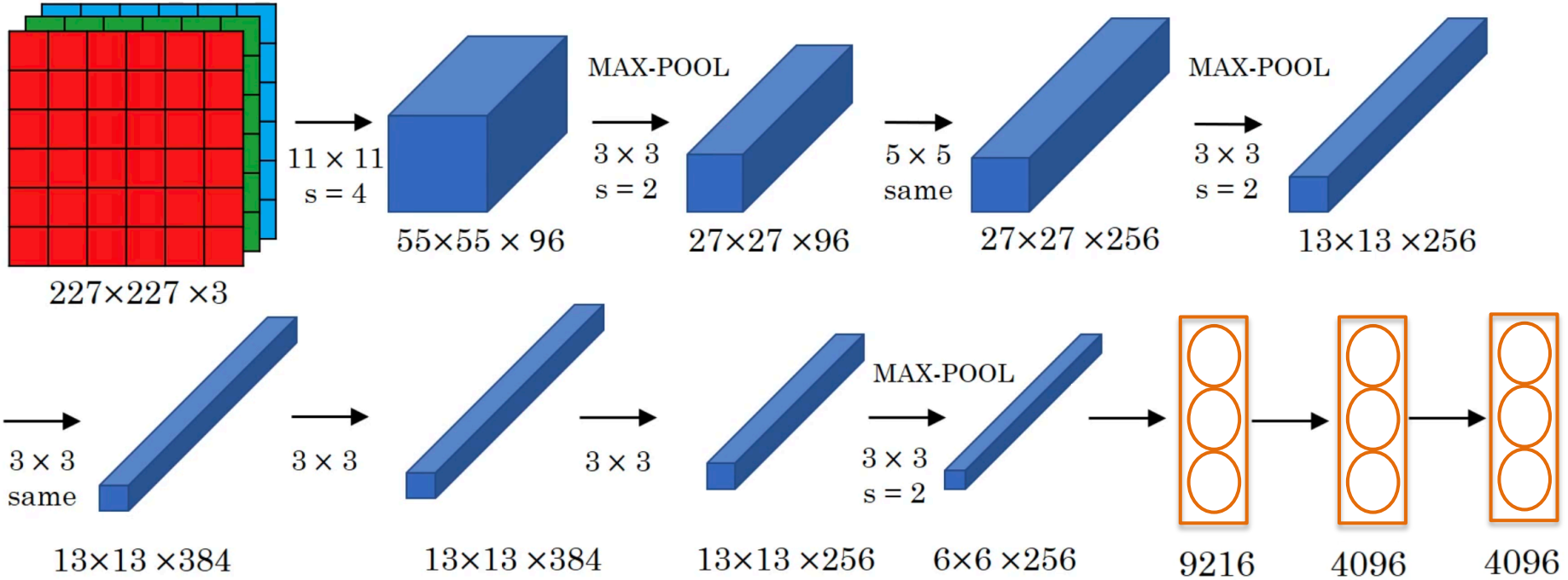
[Krizhevsky et al. NIPS'12] AlexNet

AlexNet



[Krizhevsky et al. NIPS'12] AlexNet

AlexNet



- Softmax for 1000 classes

[Krizhevsky et al. NIPS'12] AlexNet

AlexNet

- Similar to LeNet but much bigger (~1000 times)
- Use of ReLU instead of tanh/sigmoid

60M parameters

[Krizhevsky et al. NIPS'12] AlexNet

VGGNet

- Striving for simplicity

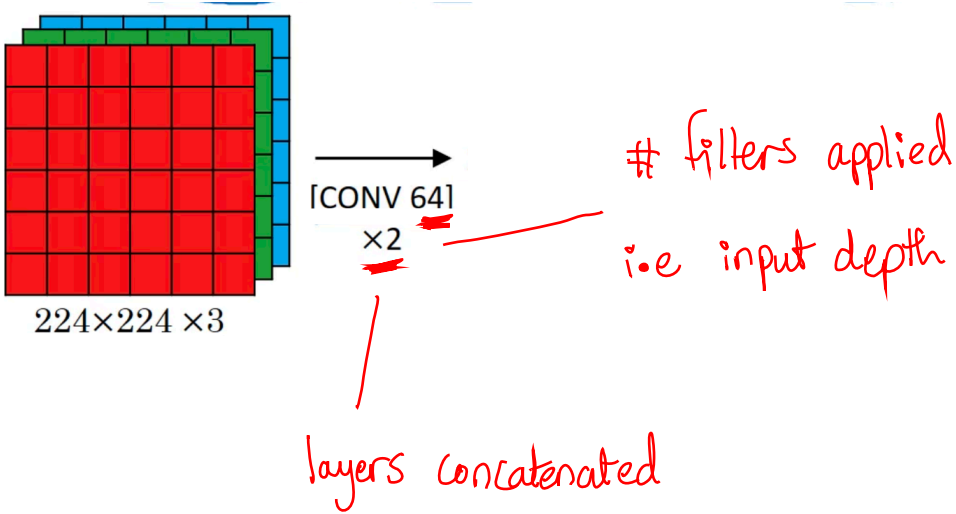
- CONV = 3x3 filters with stride 1, same convolutions

- MAXPOOL = 2x2 filters with stride 2

[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

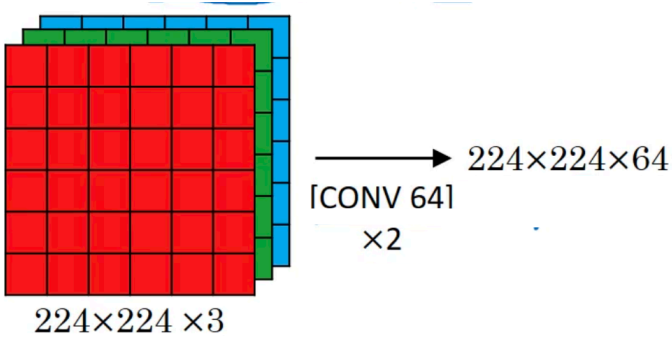
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

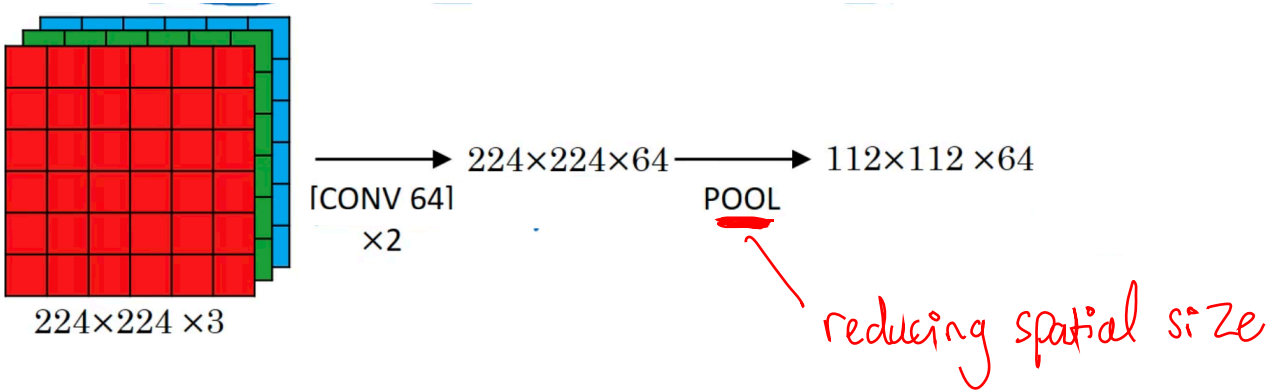
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

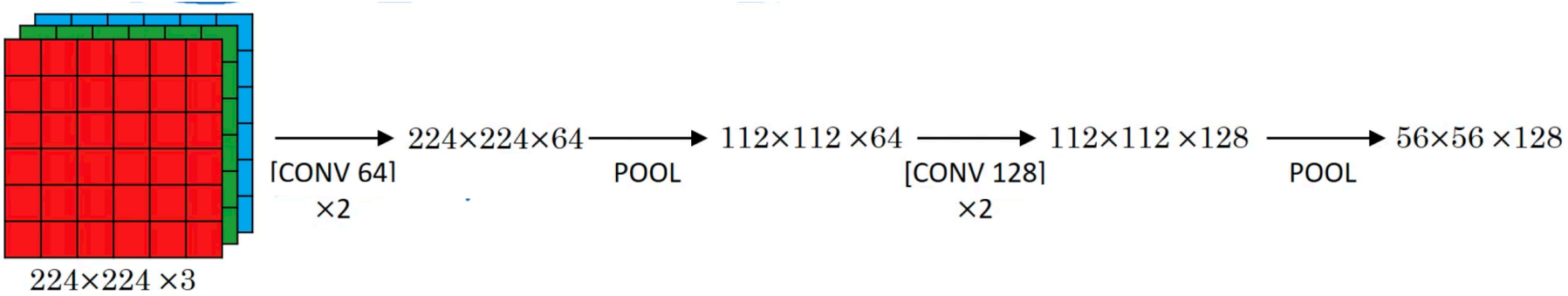
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

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

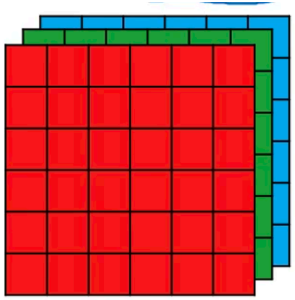


[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

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

* One Key feature: doubling #filters along the way



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

[Simonyan and Zisserman ICLR'15] VGGNet

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

[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

- A lot of architectures were analyzed

* It was obviously not their first choice.

[Simonyan and Zisserman 2014]

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

Skip Connections

* Residual Block

* ResNet

The Problem of Depth

- As we add more and more layers, training becomes harder

because of

- Vanishing and exploding gradients

⇒ In general, we aim for more expressiveness.

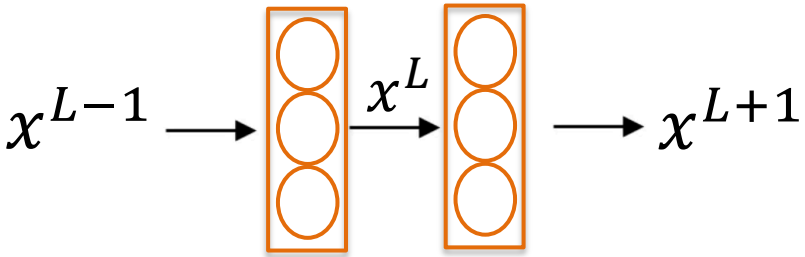
* How can we train very deep nets?

Residual Block

NB

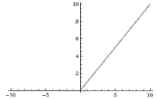
type of layer used here does not matter

- Two layers



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

Linear

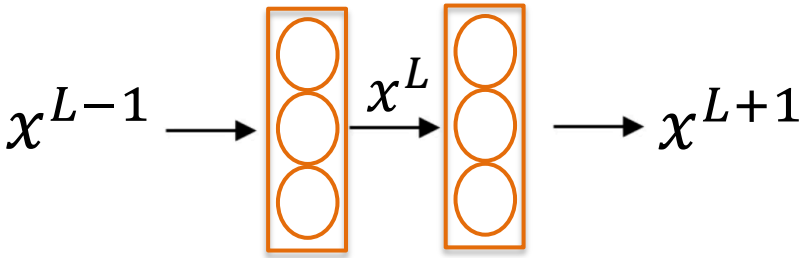


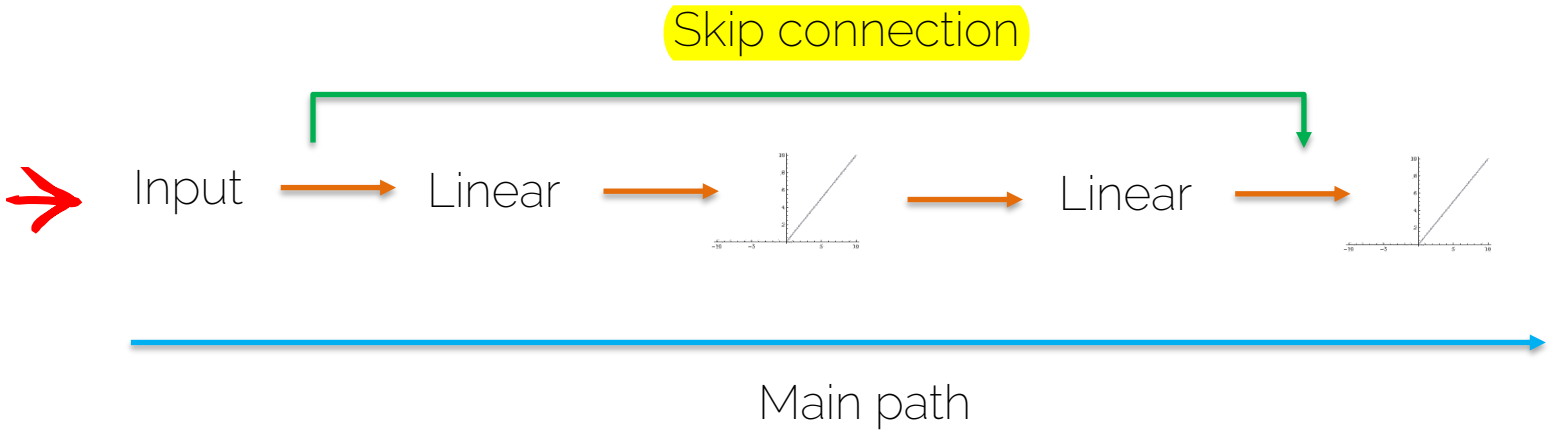
Non-linearity

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

Residual Block

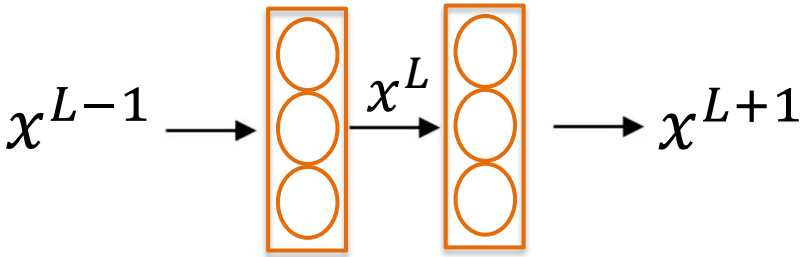
- Two layers





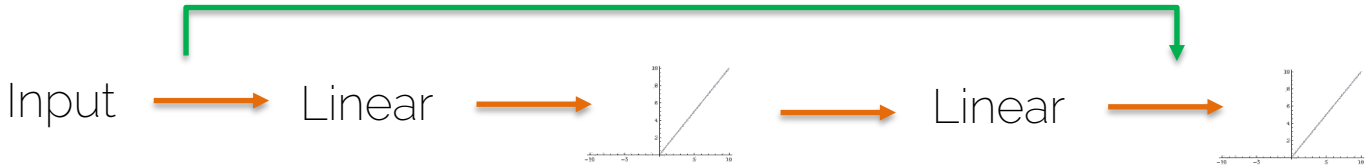
Residual Block

- Two layers



With skip connection

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

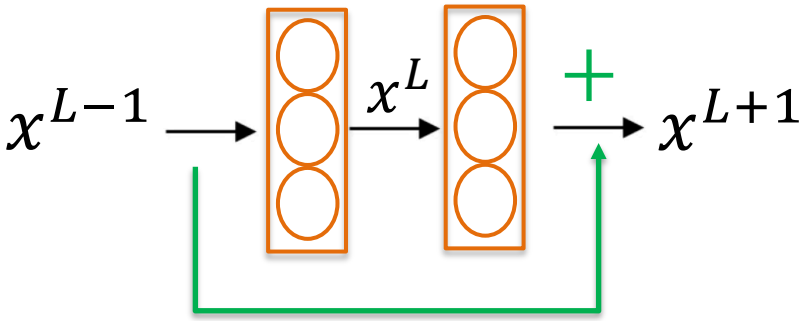


plain – w/o skip connection

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

Residual Block

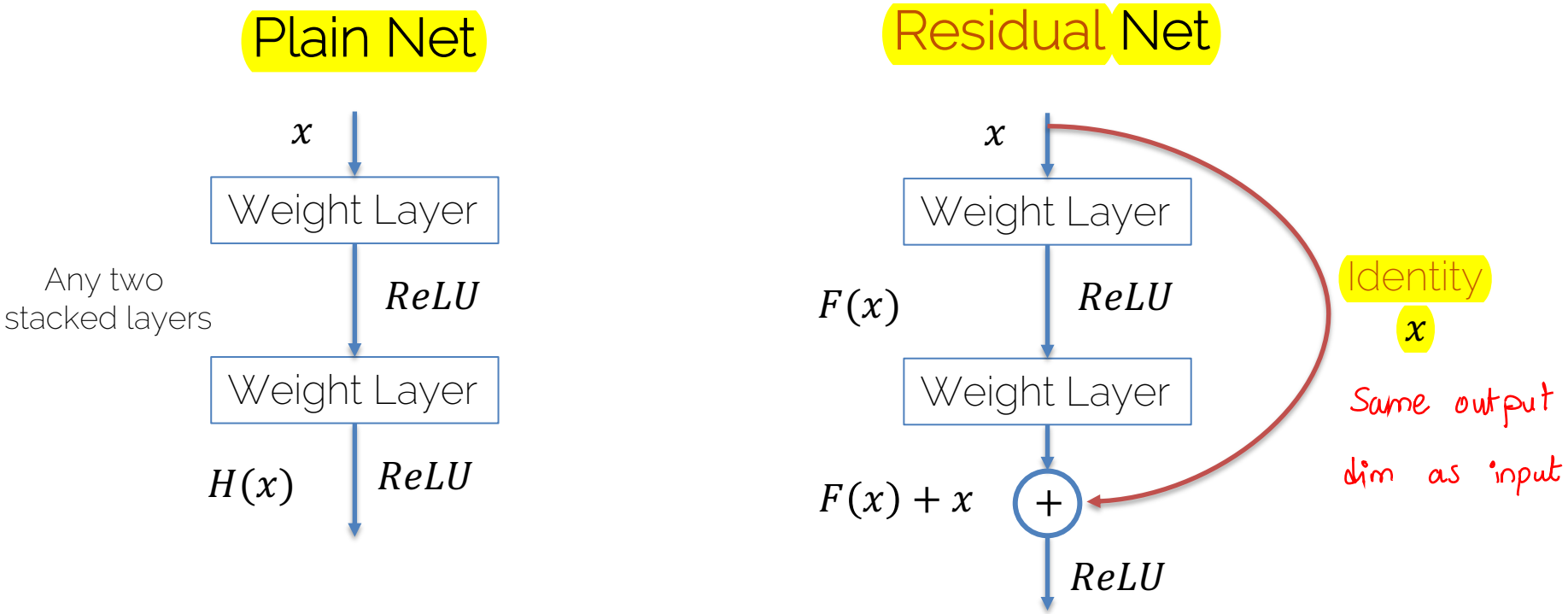
- Two layers



Remember

- 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

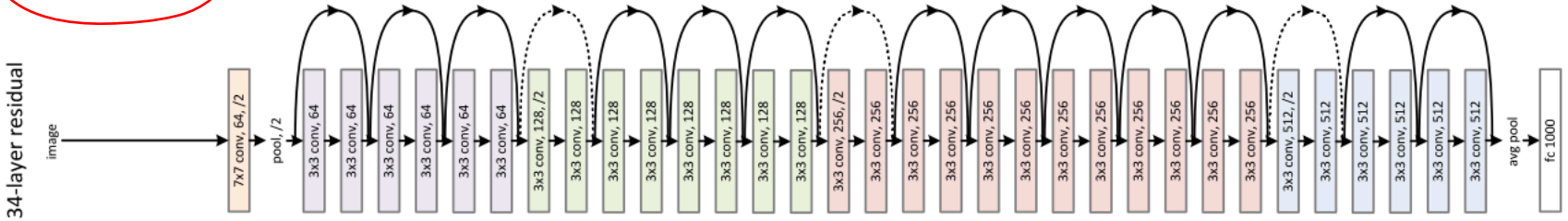
ResNet Block



Specs

ResNet

Important



- 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-152:
60M parameters

[He et al. CVPR'16] ResNet

with plain nets

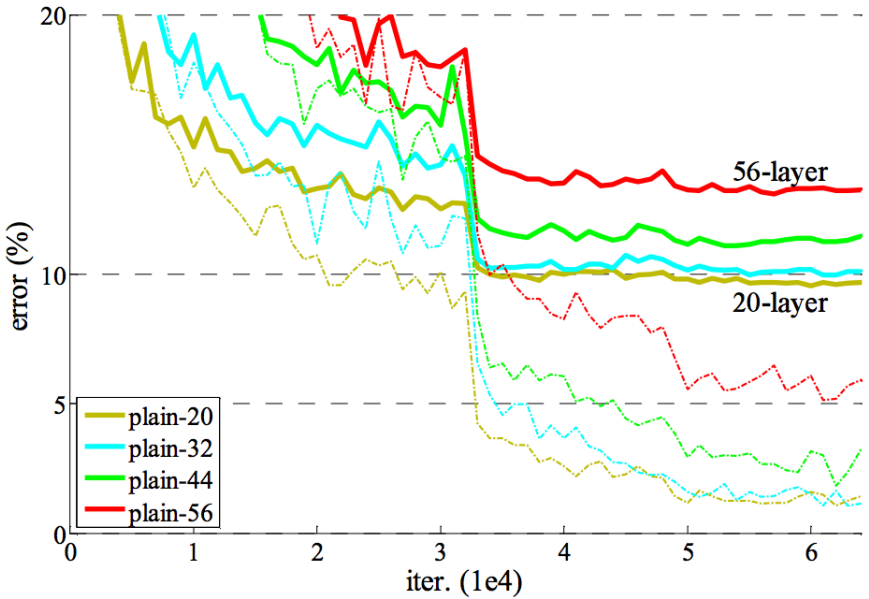
- If we make the network deeper, at some point performance starts to degrade

Notice in the first 3 iterations

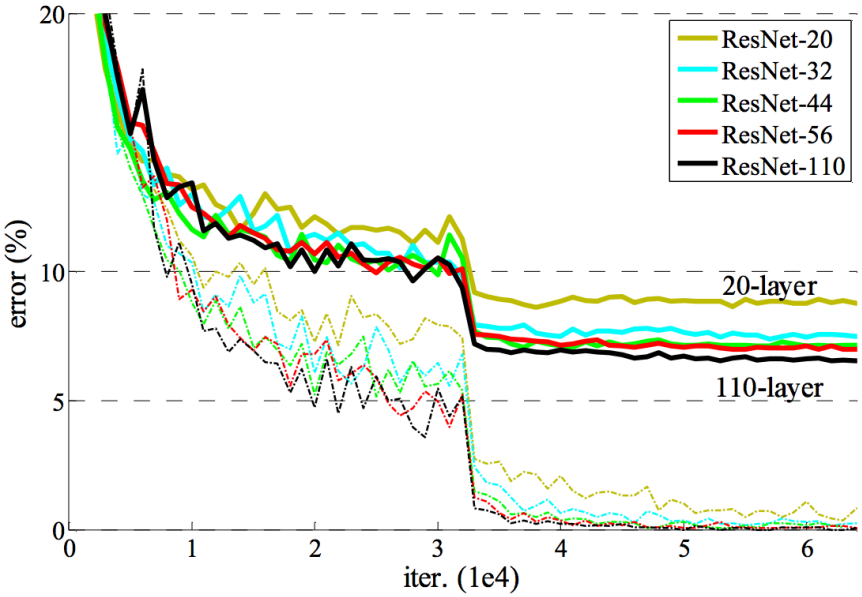
plain-20 has better error than plain-56!

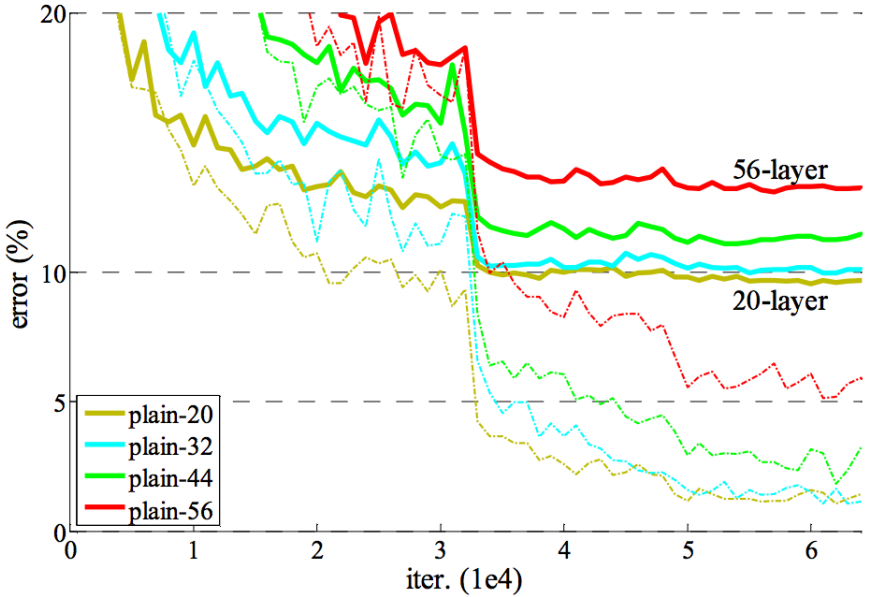
↓

is either overfitting or not trained properly



* However, using ResNet deeper is actually better!

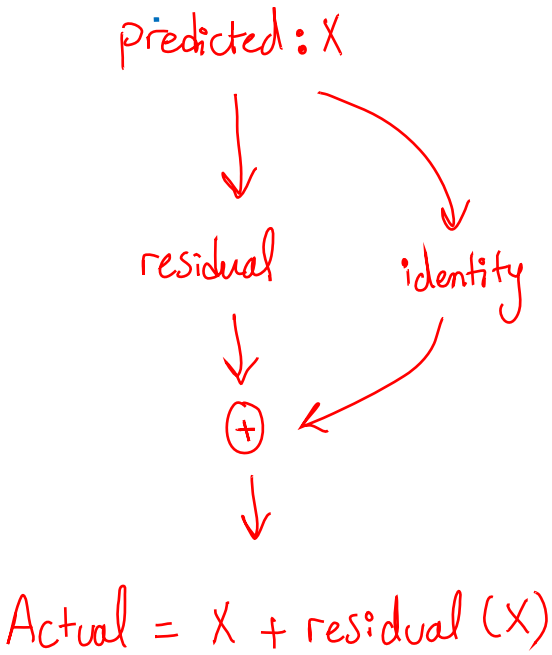




Remember residual is nothing but error.

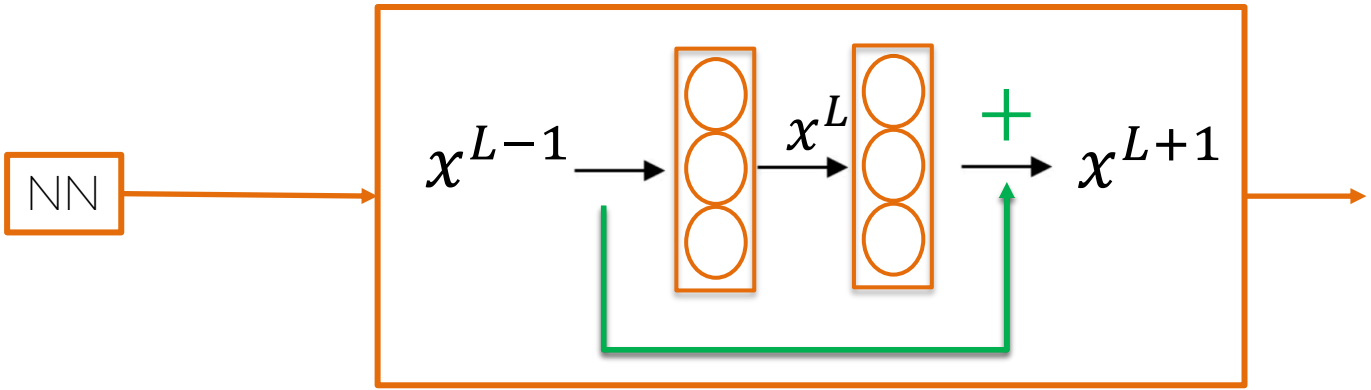
* As we go deeper, we need to make sure not to degrade the accuracy & error rate. This can be handled by identity mapping.

* Keep learning the residual to match the predicted with the actual.



- 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

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

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

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

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

It's all about learning < identity, non-linearity

1x1 Convolutions

Recall: 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

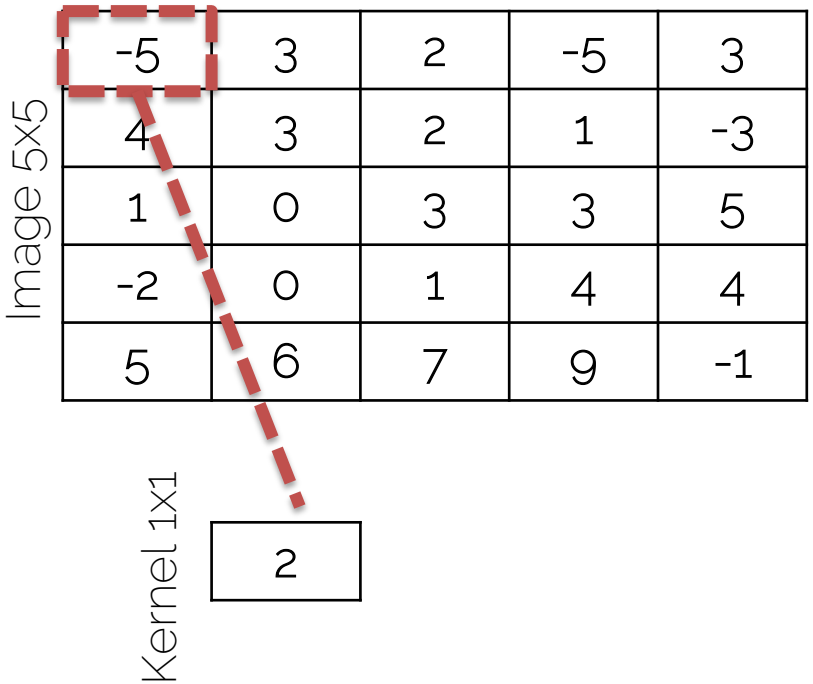
0	-1	0
-1	5	-1
0	-1	0

Output 3x3

6		

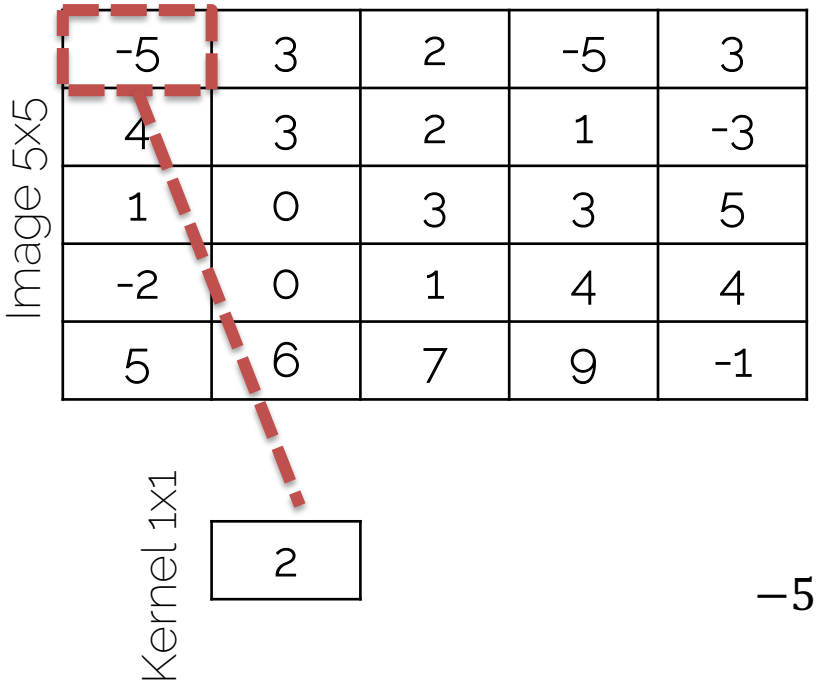
$$5 \cdot 3 + (-1) \cdot 3 + (-1) \cdot 2 + (-1) \cdot 0 + (-1) \cdot 4 =$$
$$15 - 9 = 6$$

1x1 Convolution



What is the output size?

1x1 Convolution



-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

-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

Kernel 1X1

2

$$-1 * 2 = -2$$

1x1 Convolution

Image 5x5

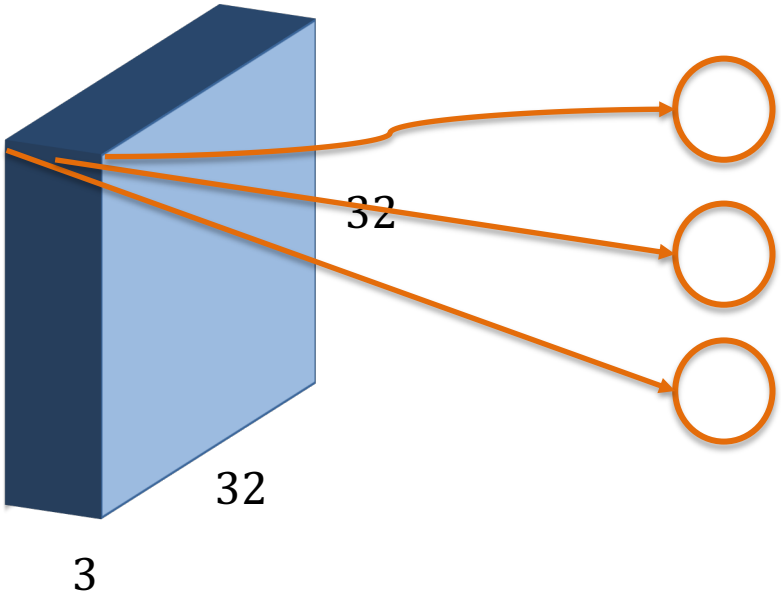
-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

-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

* 1x1 kernel: keeps the dimensions and scales input

No need for padding

1x1 Convolution

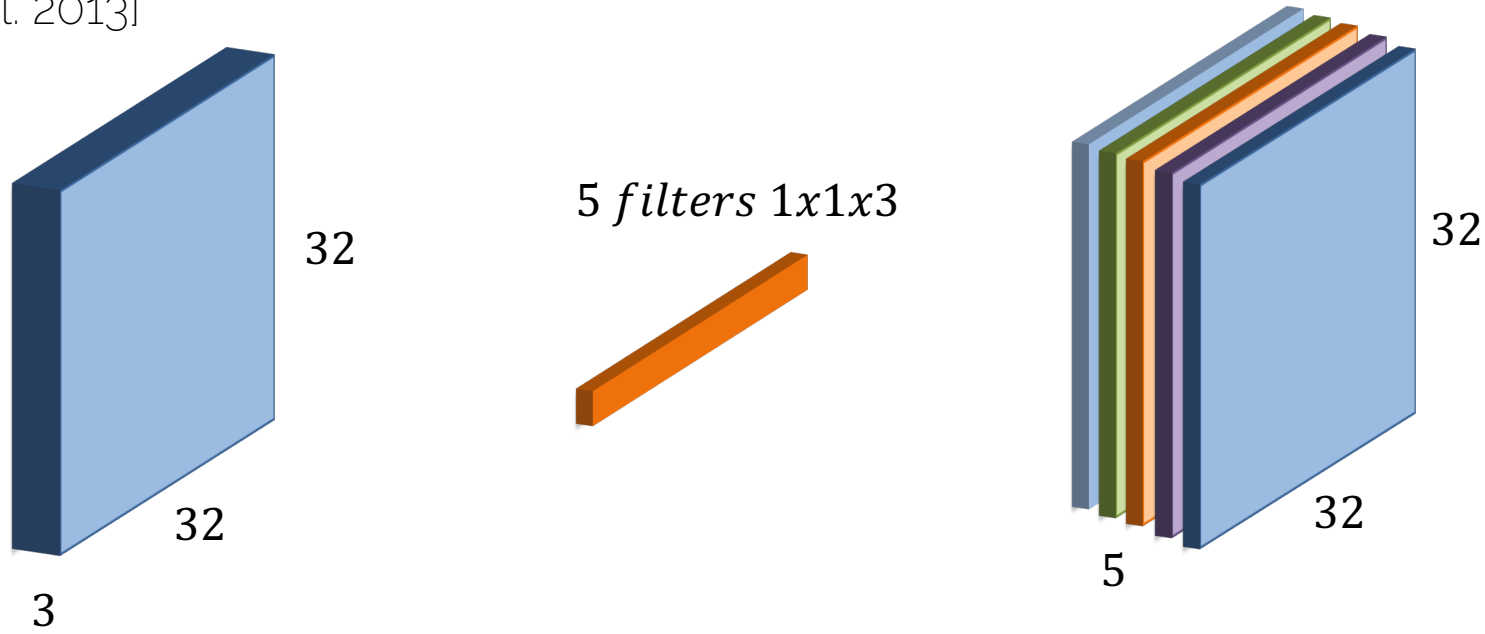


1 output

- Same as having a 3 neuron fully connected layer

1x1 Convolution

[Li et al. 2013]



- As always we use more convolutional filters

Remember

Using 1x1 Convolutions

with same depth

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

resulting in cheaper subsequent operations

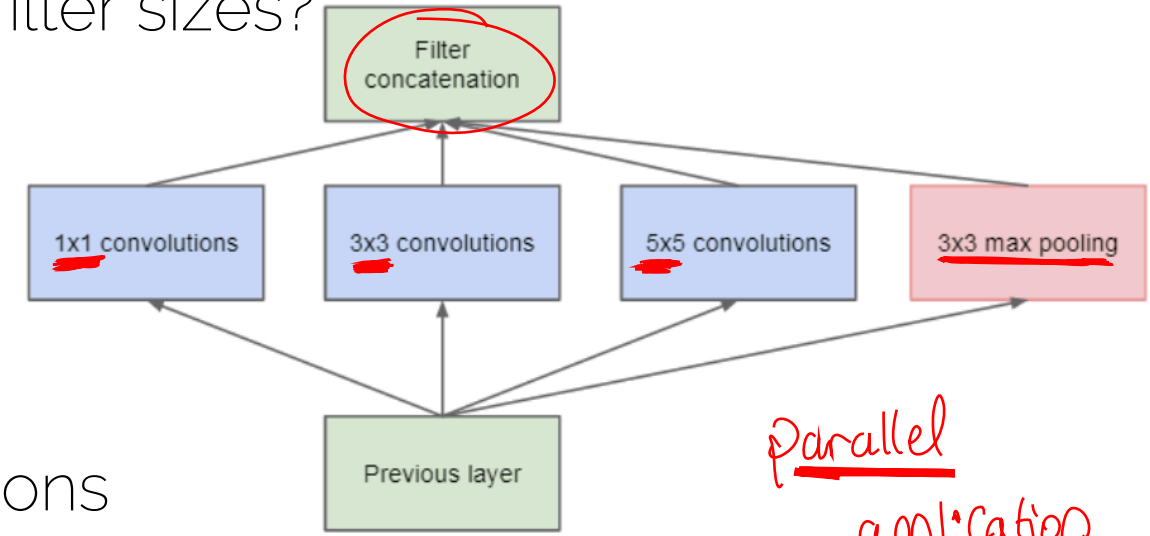
Inception Layer

Combining the benefits of both conv & pool,
allowing the net to learn whatever parameters

Inception Layer

- Tired of choosing filter sizes?

* Use them all!



parallel application

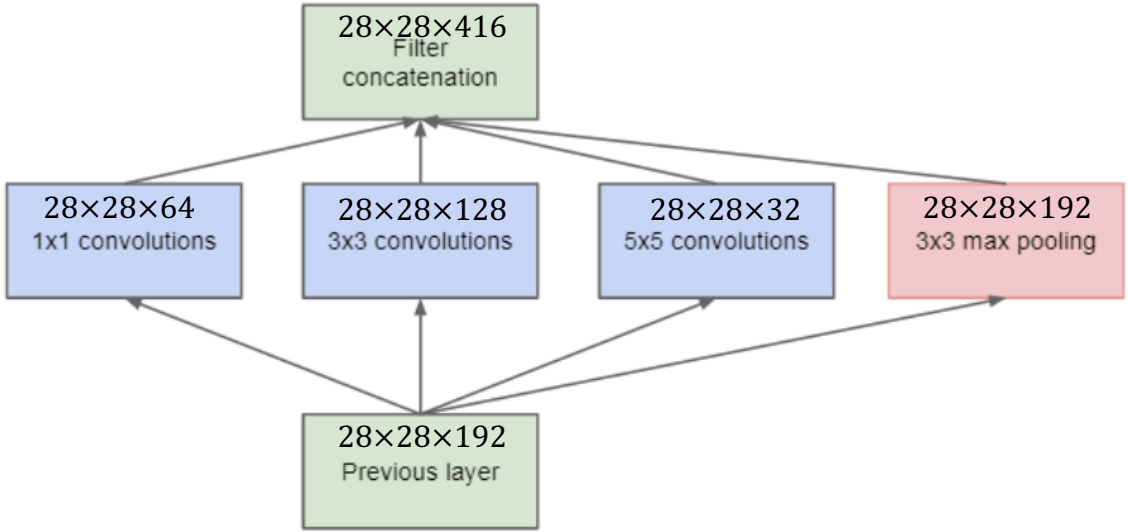
- All same convolutions

- 3x3 max pooling is with stride 1

* only depth is considered *

Naive Version of Inception

- Possible size of the output



Such naive concatination will yield in alot of multiplications!

- Not sustainable!

e.g # weights for 1x1 conv : (1x1x192 + 1) x 64

Inception Layer: Computational Cost

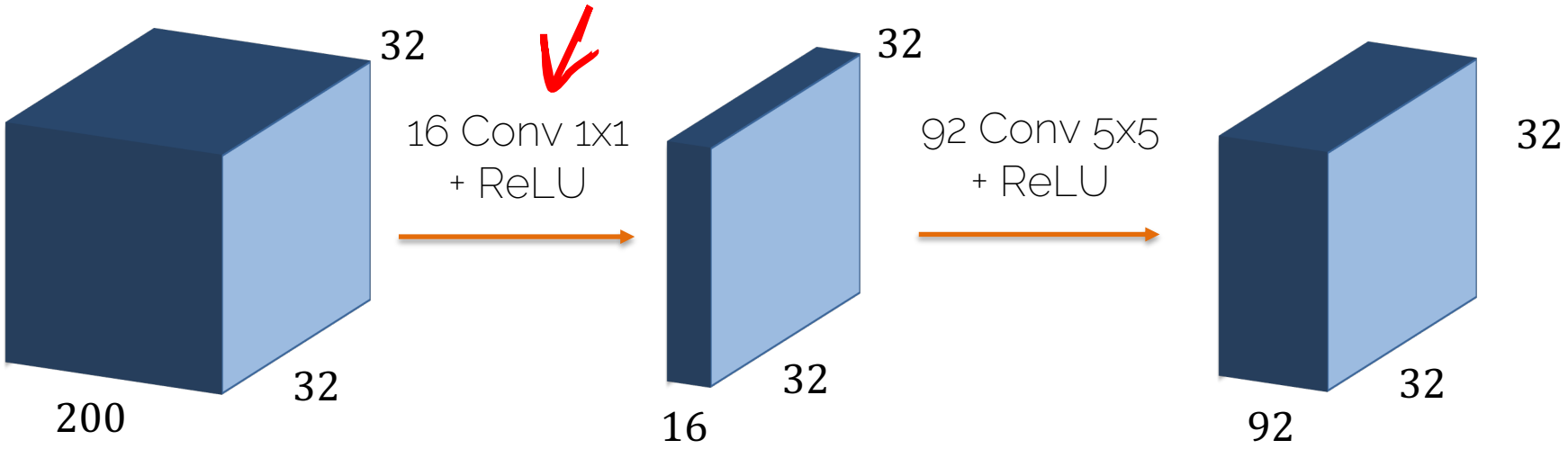
NB

Multiplications: 5x5x200 x 32x32x92 ~ 470 million

in total

1 value of the output volume

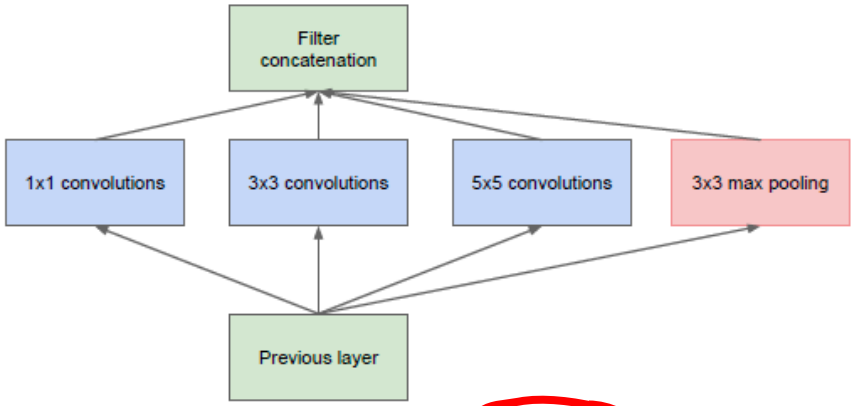
Size remains the same,
yet depth reduced drastically



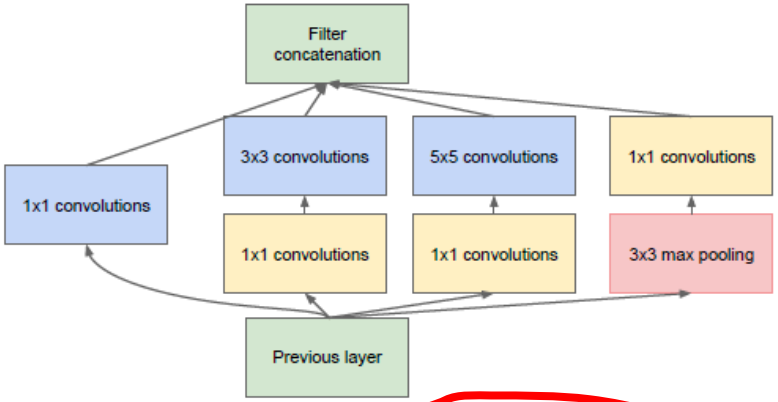
Multiplications: 1x1x200x32x32x16 5x5x16x32x32x92 ~ 40 million

Reduction of multiplications by 1/10

Inception Layer



(a) Inception module, naïve version

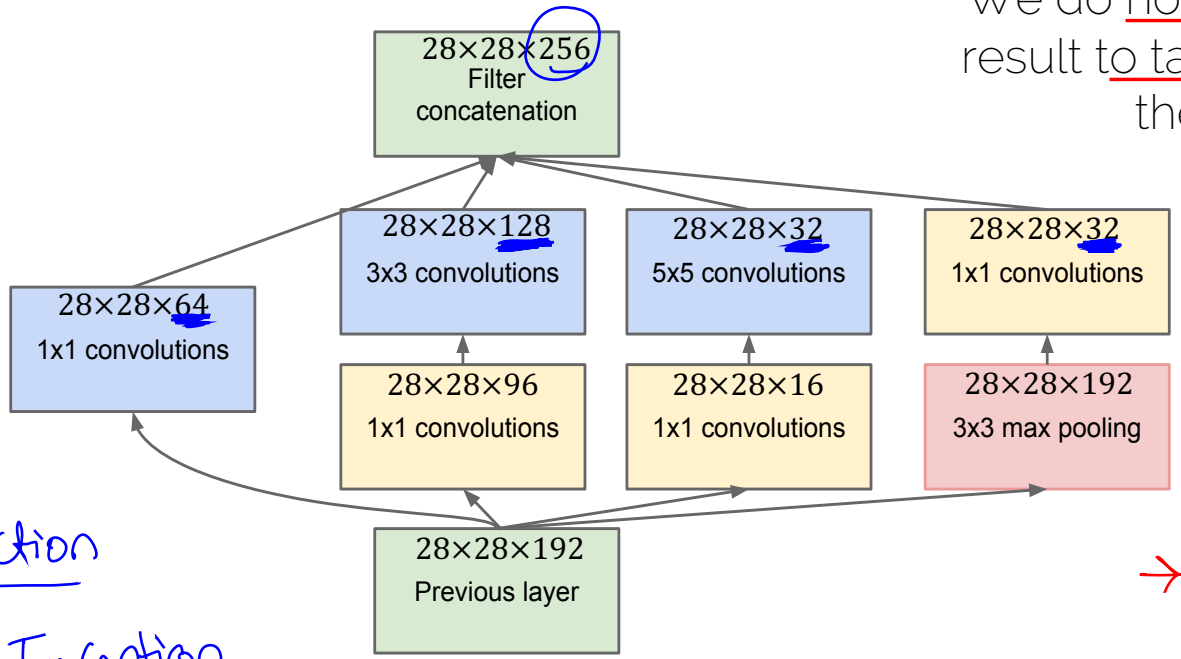


(b) Inception module with dimensionality reduction

[Szegedy et al CVPR'15] GoogLeNet

* Such architecture aimed for dimensionality reduction & subsequently # multiplications.

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



We give more weight to the conv since it extracts the features

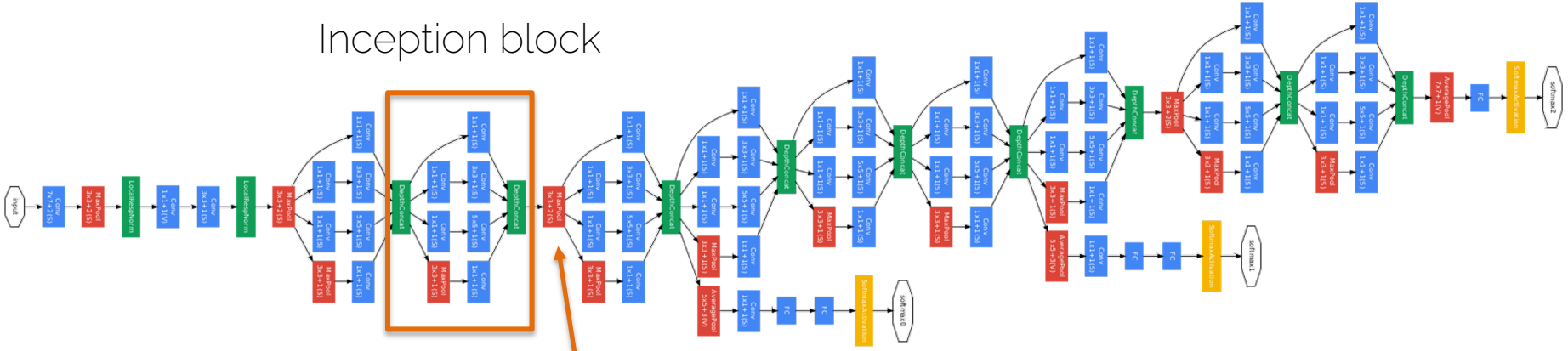
Dim Reduction Version of Inception

→ Not doing so would result in overwhelming representation!

GoogLeNet:

Using max pool with S=1 inside inception layer, we did not really reduce spatial size of activation map

Inception block



Extra max pool layers to reduce dimensionality

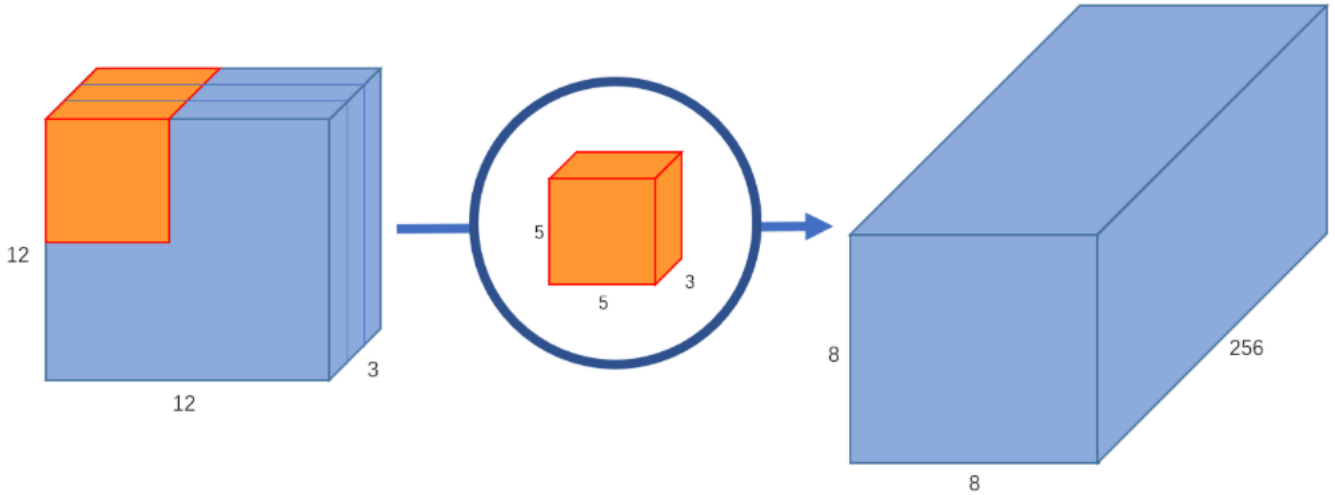
Originally, we aimed to ↓ size, ↑ depth

Xception Net

- „Extreme version of Inception“: applying (modified) Depthwise Separable Convolutions instead of normal convolutions
- 36 conv layers, structured into several modules with skip connections
- outperforms Inception Net V3

[Chollet CVPR'17] Xception

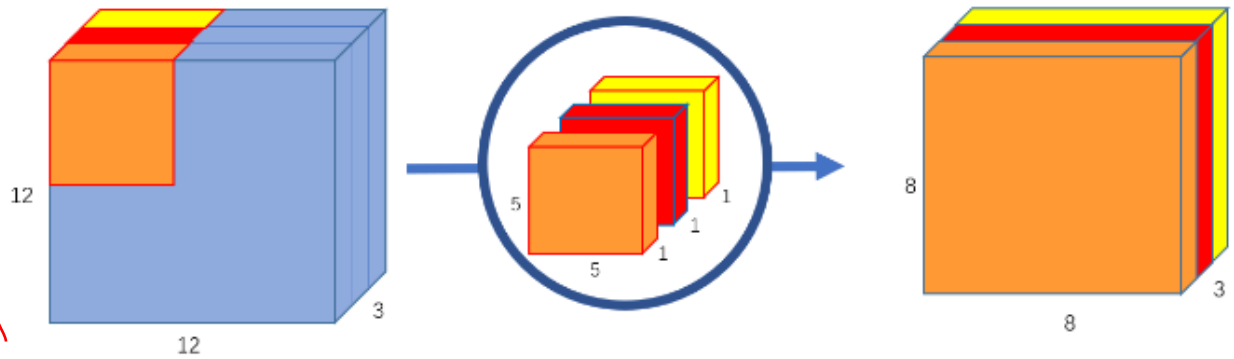
Depth-wise separable convolutions



Normal convolutions act on all channels.

Depth-wise separable convolutions

idea is that depth & spatial dim of a filter can be seperated

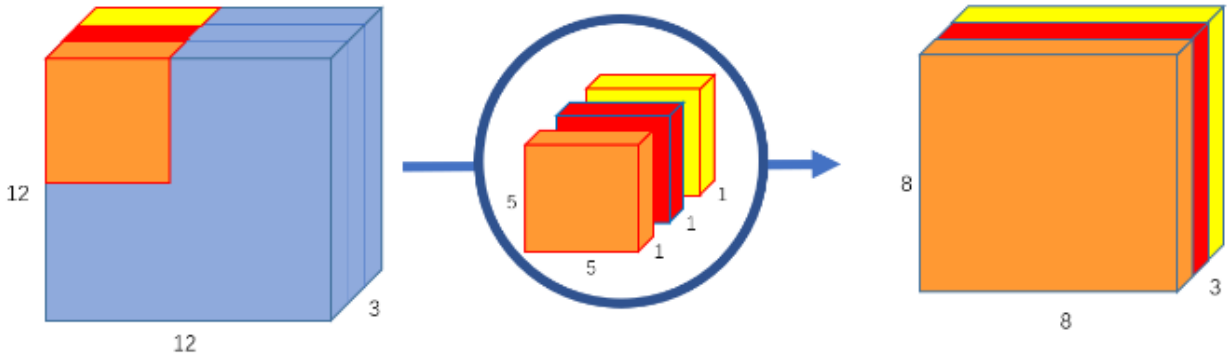


Filters are applied only at certain depths of the features.
Normal convolutions have groups set to 1, the convolutions used in this image have groups set to 3.

`class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)`

`class torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)`

Depth-wise separable convolutions

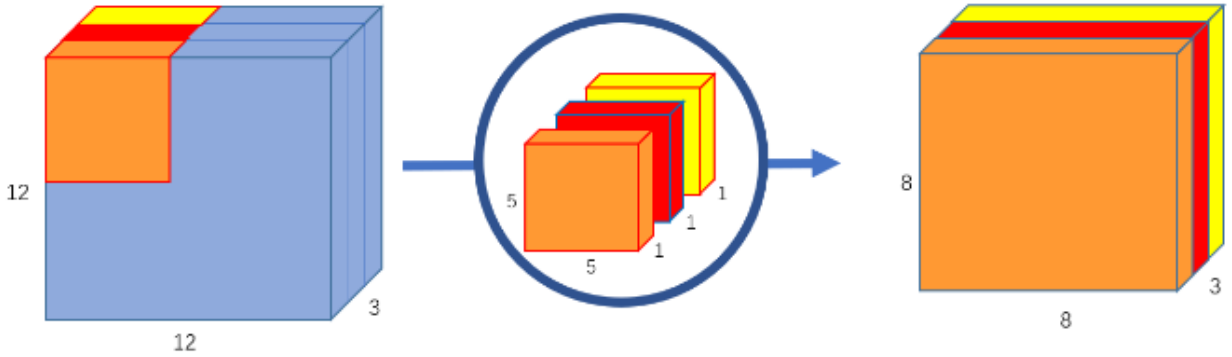


But the depth size is always the same!

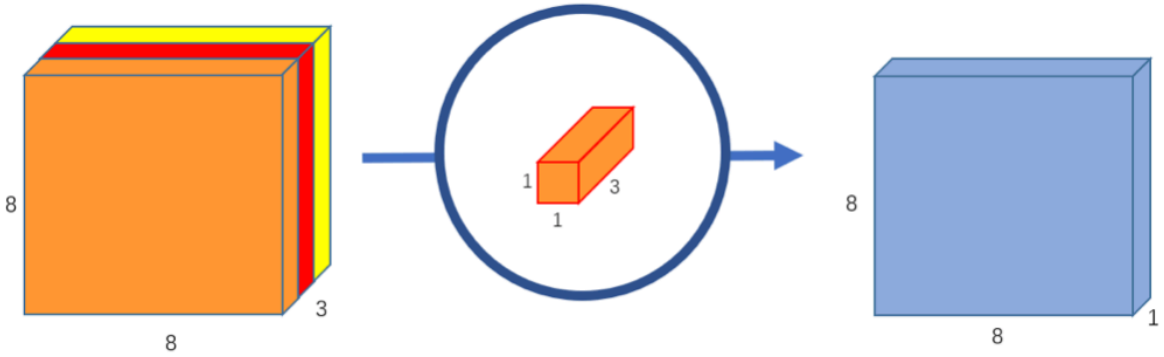
class*`torch.nn.Conv2d`*(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)

class*`torch.nn.ConvTranspose2d`*(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)

Depth-wise separable convolutions

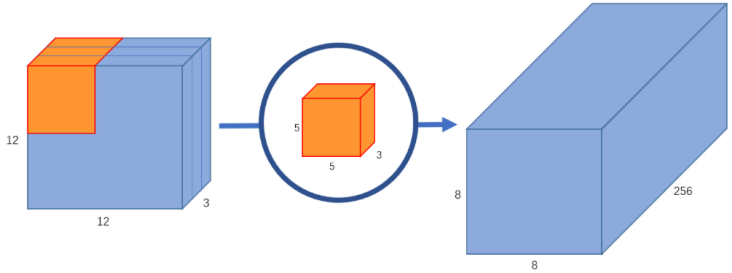


not required
Since depth
is not reduced



Solution:
1x1 convs!

But why?



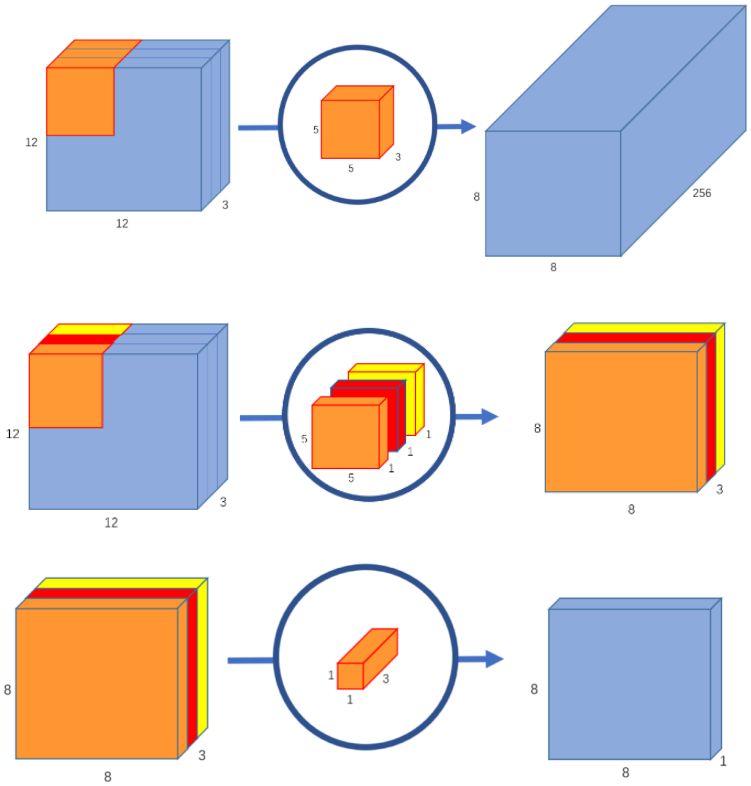
Original convolution
256 kernels of size 5x5x3

Multiplications:
256x5x5x3 x (8x8 locations) = 1.228.800

* Calc # multiplications : F x F x depth X Out x out x depth

→ Reducing # parameters to output same # channels

But why?



Original convolution
256 kernels of size 5x5x3

Multiplications:
256x5x5x3 x (8x8 locations) = 1.228.800

Depth-wise convolution
3 kernels of size 5x5x1

Multiplications:
5x5x3 x (8x8 locations) = 4800

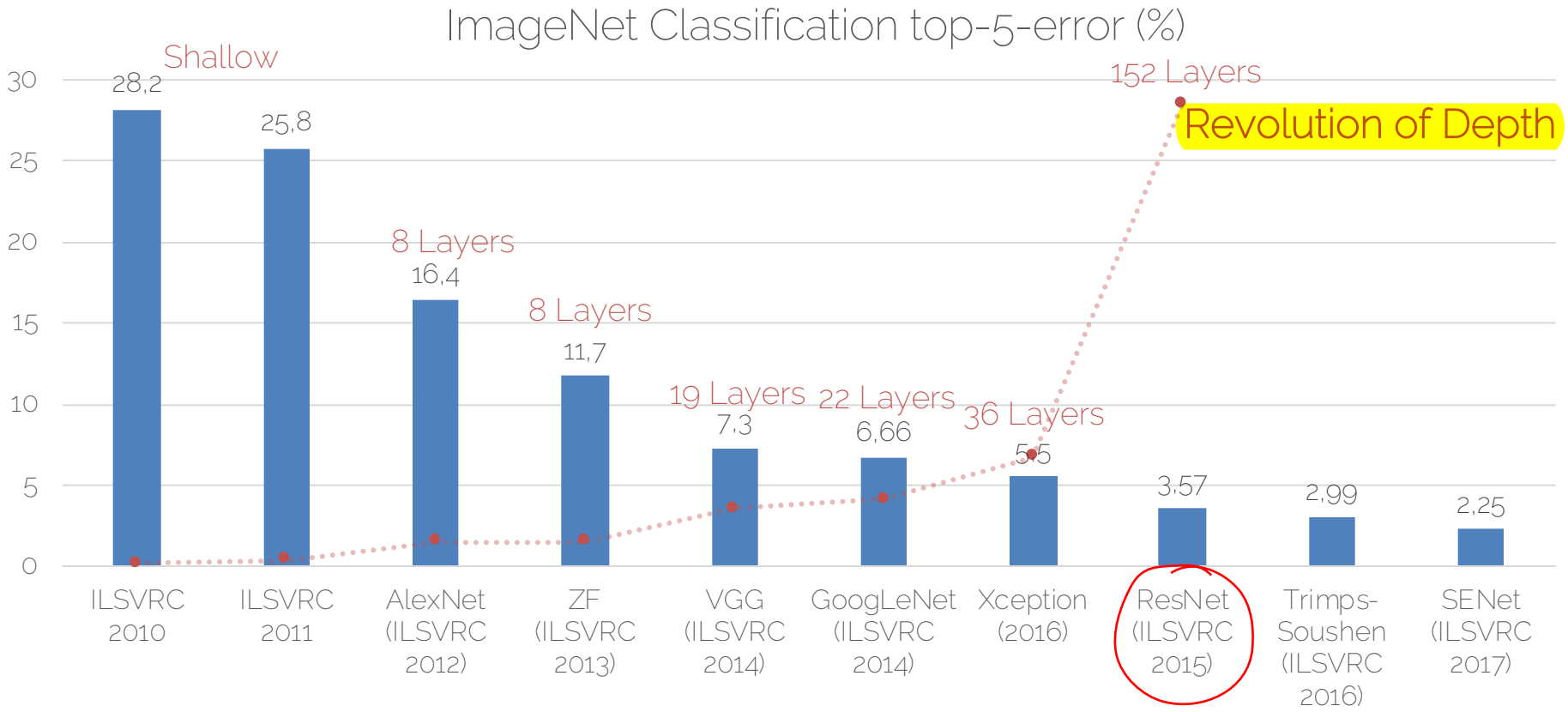
NB

Less computations!

1x1 convolution
256 kernels of size 1x1x3

Multiplications:
256x1x1x3x (8x8 locations) = 49152

ImageNet Benchmark

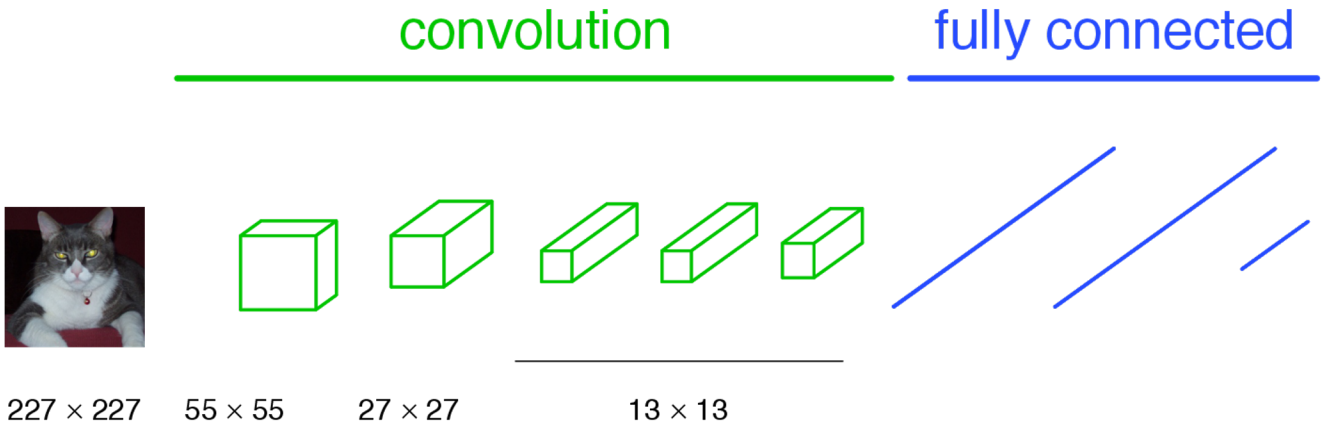


NB The convolution operation reduces the spatial dimensions as we go deeper down the network and creates an abstract representation of the input image.

Fully Convolutional Network

FCN

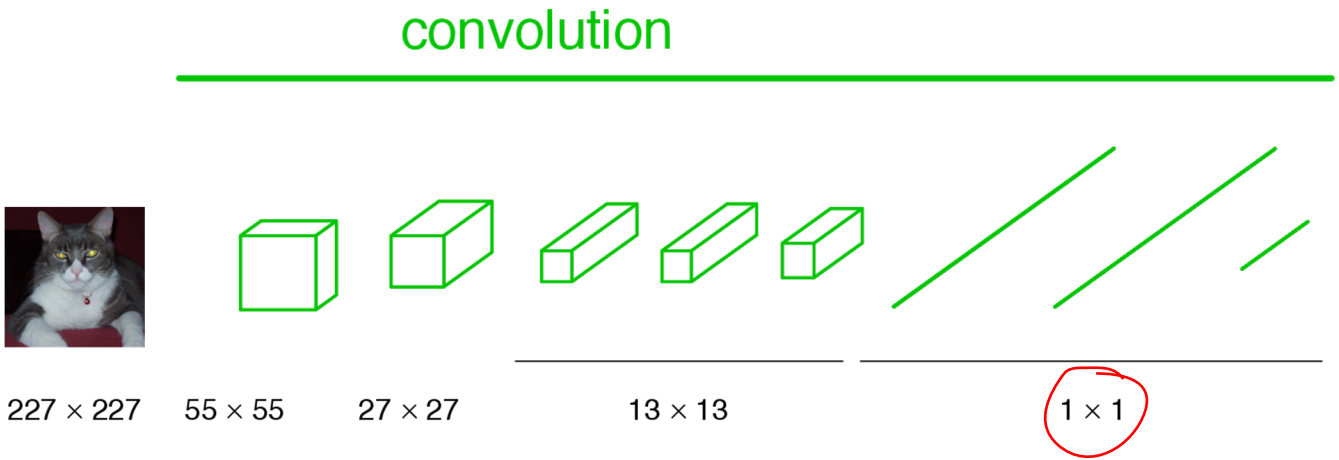
Classification Network



* Fully connected huge disadvantage was taking fixed input size.

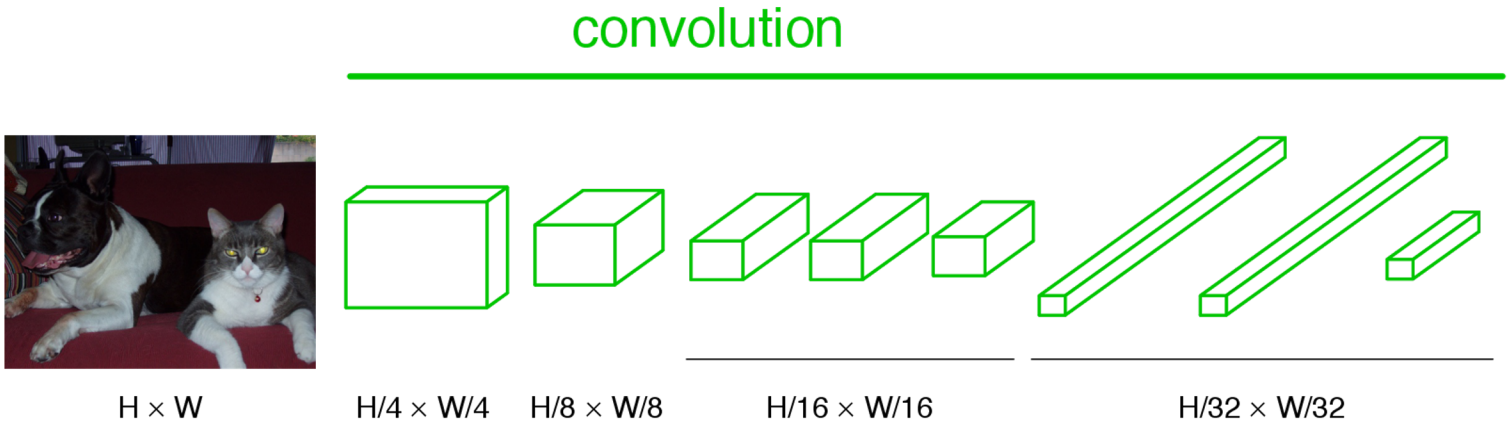
It has no structure, just brute force!

FCN: Becoming Fully Convolutional



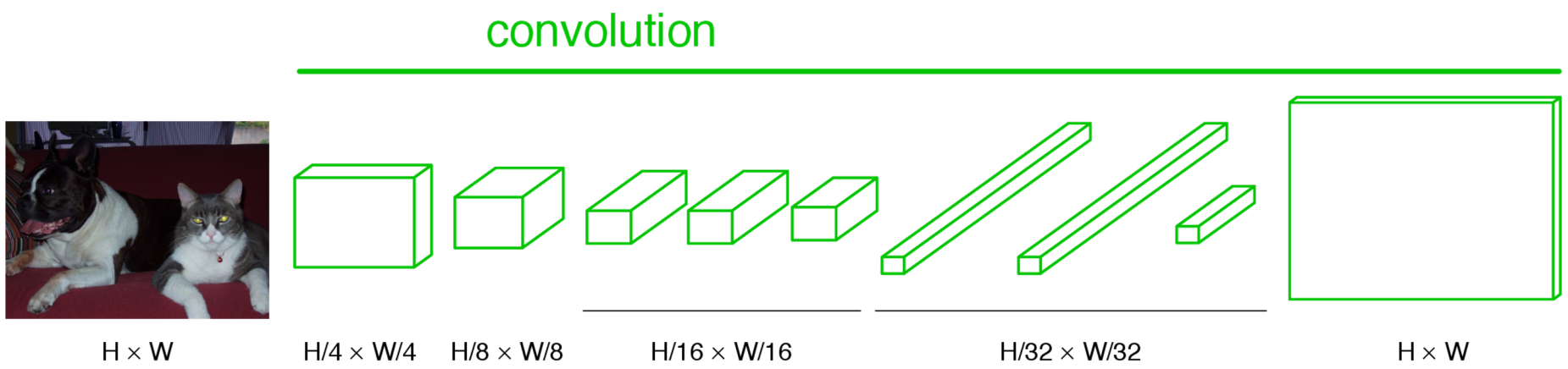
Convert fully connected layers to convolutional layers!

* CNN even yields better flexibility working with any input size.



FCN: Upsampling Output

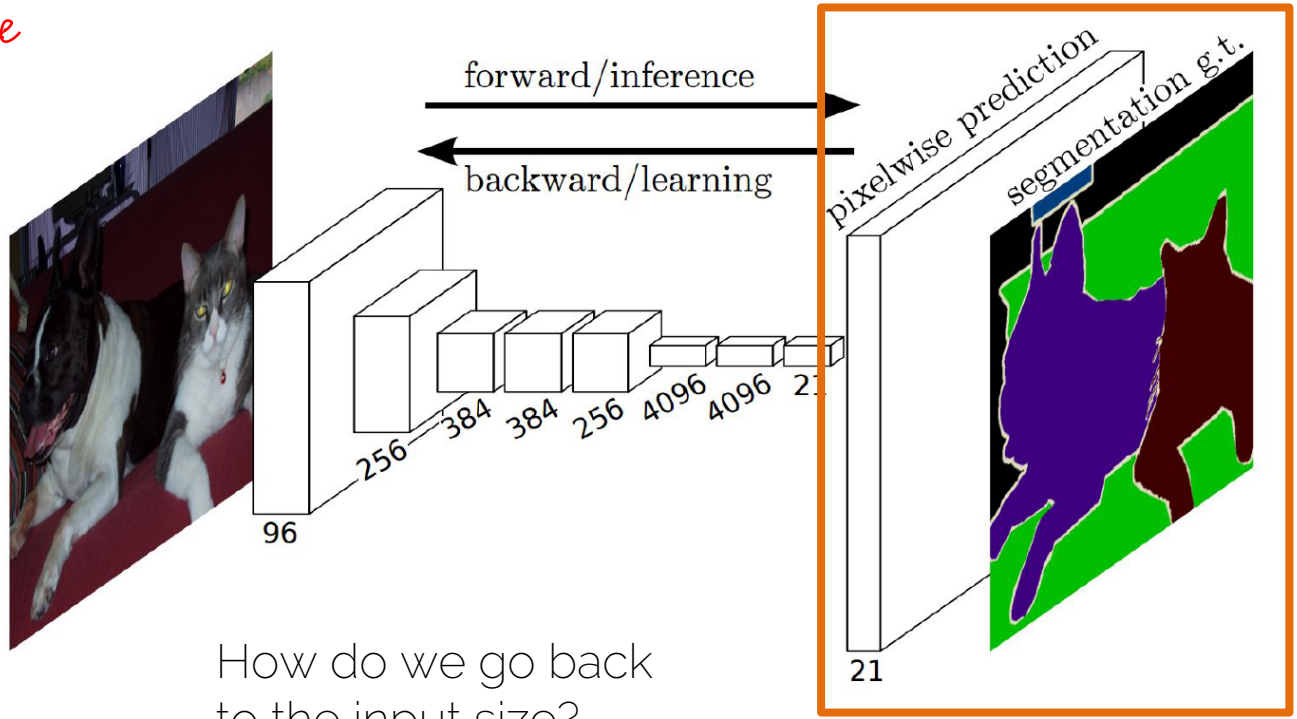
↖ to bring back the resoln of prev layers



* The Sole purpose of down/up sampling is to reduce computations in each layer, while keeping the dim of I/O as before.

Semantic Segmentation (FCN)

input size shall be equal output size

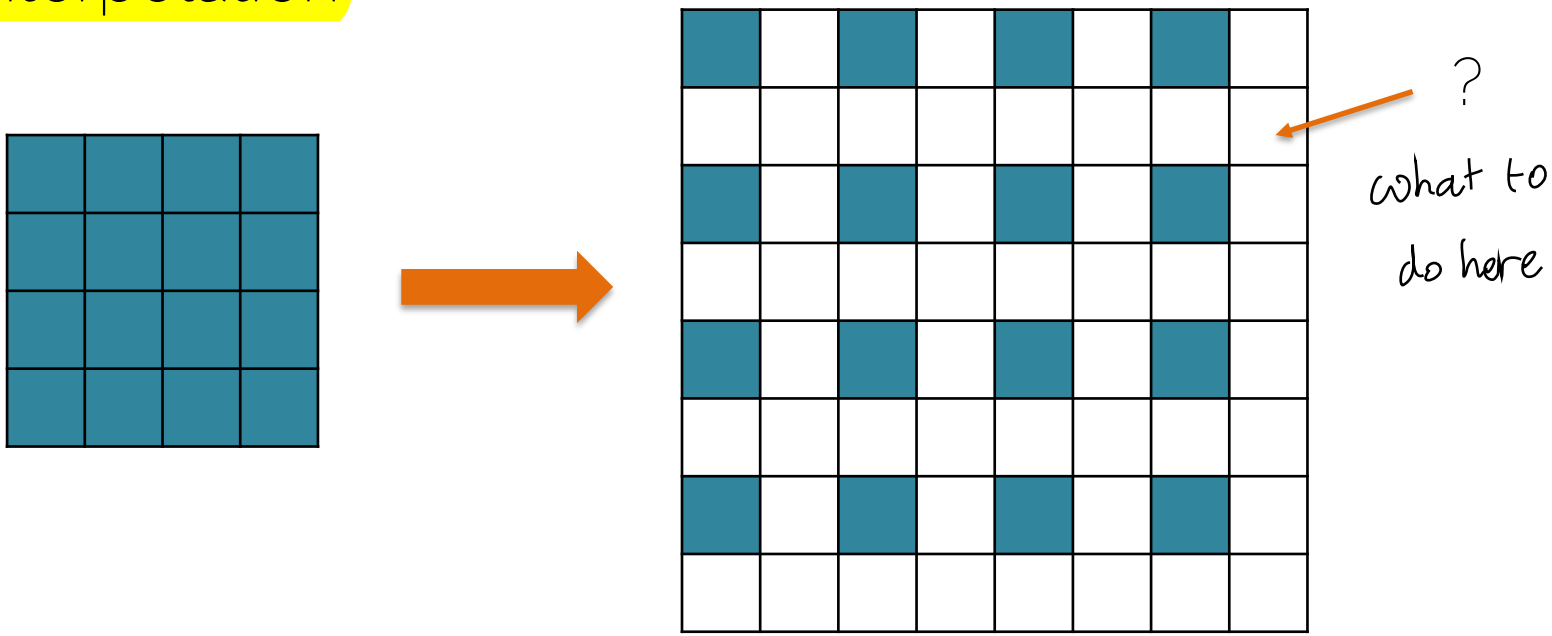


How do we go back to the input size?

[Long and Shelhamer. 15] FCN

Types of Upsampling

- 1. Interpolation



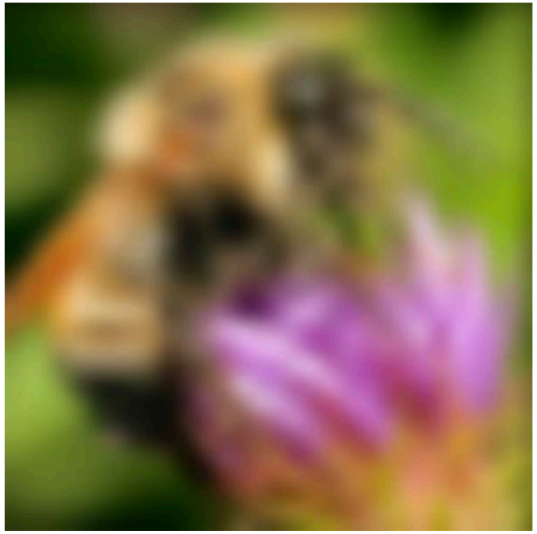
How to handle no-value pixels ?

- 1. Interpolation

Original image  **x 10**

Nearest neighbor interpolation

Bilinear interpolation



Bicubic interpolation

Image: Michael Guerzhoy

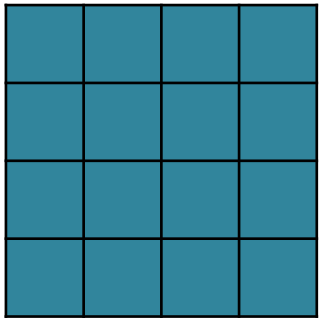
- 1. Interpolation

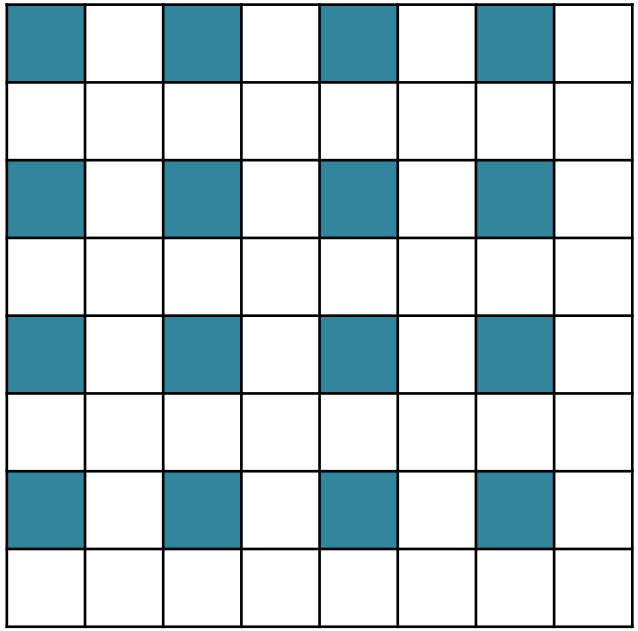
  Few artifacts

NB Upsampling doesn't (and cannot) reconstruct any lost info!

Is used to upsample the input feature map to a desired output feature map using some learnable parameters.

- 2. Transposed conv





efficient

+ CONVS

mainly helps refining representation

[A. Dosovitskiy, TPAMI 2017] "Learning to Generate Chairs, Tables and Cars with Convolutional Networks"

- 2. Transposed convolution
  - Unpooling
  - Convolution filter (learned)
  - Also called up-convolution

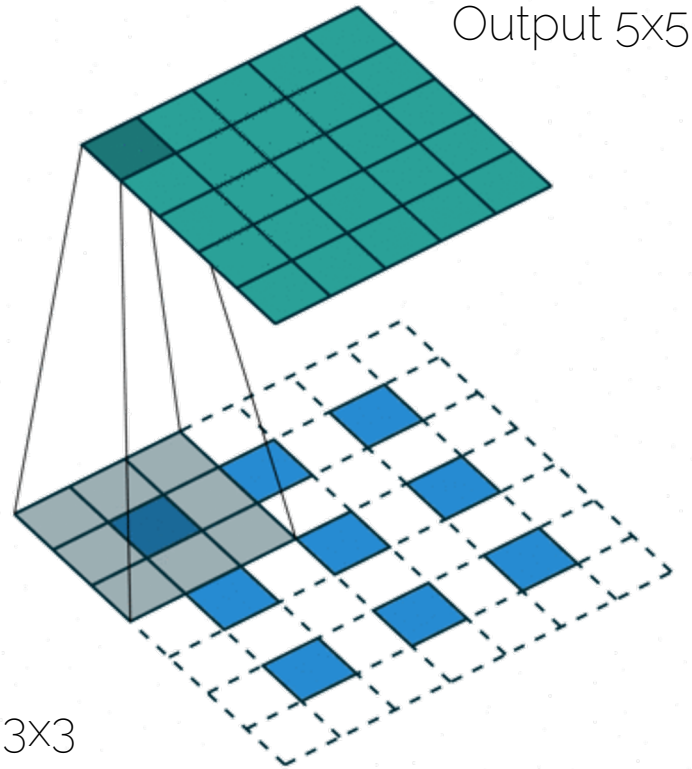
(never deconvolution)

mainly increase size

filling empty pixels

implies removing the effect of convolution

which we are not aiming to achieve

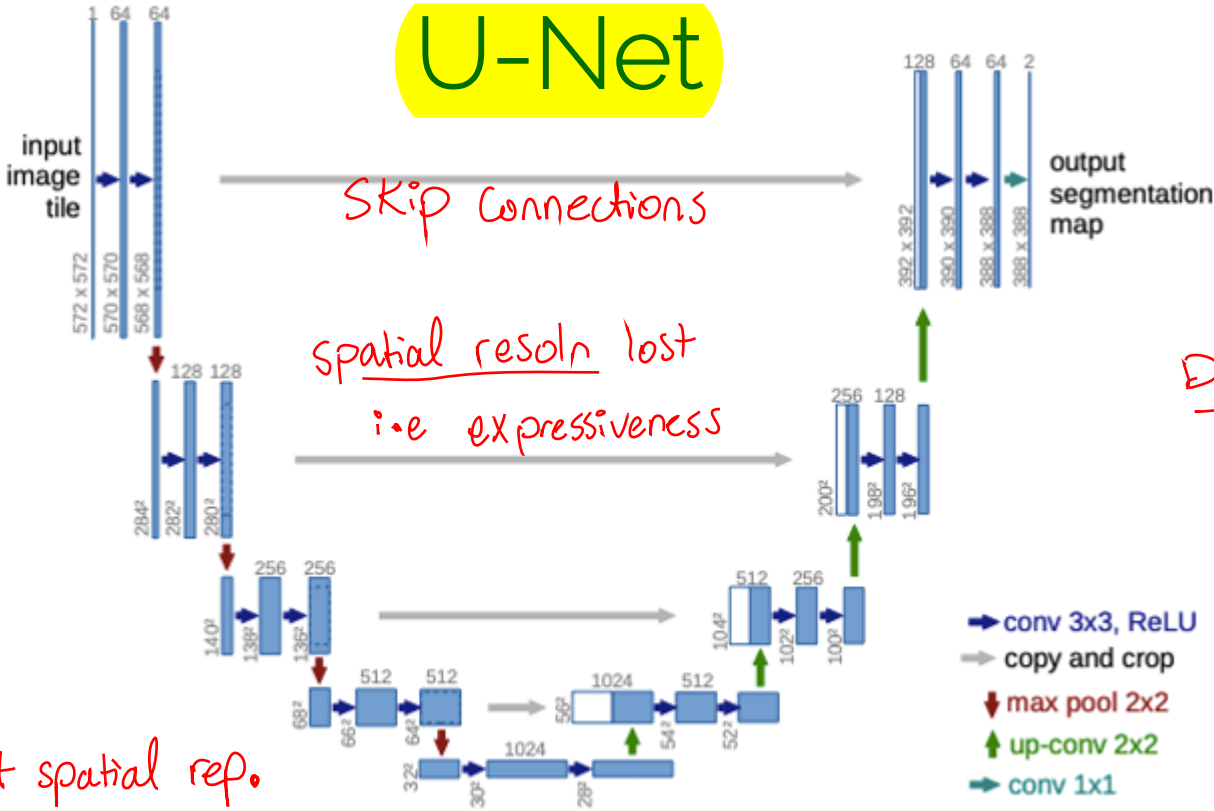


Refined Outputs

- If one does a cascade of unpooling + conv operations, we get to the encoder-decoder architecture

- Even more refined: Autoencoders with skip connections (aka U-Net)

U-Net



U-Net architecture: Each blue box is a multichannel feature map. Number of channels denoted at the top of the box . Dimensions at the side of the box. White boxes are the copied feature maps.

U-Net: Encoder

Left side: **Contraction Path** (Encoder)

- Captures context of the image
- Follows typical architecture of a CNN:
  - Repeated application of 2 unpadded 3x3 convolutions
  - Each followed by ReLU activation
  - 2x2 maxpooling operation with stride 2 for downsampling
  - At each downsampling step, # of channels is doubled

as before: Height, Width ↓, Depth: ↑

[Ronneberger et al. MICCAI'15] U-Net

U-Net: Decoder

Right Side: **Expansion Path** (Decoder):

- Upsampling to recover spatial locations for assigning class labels to each pixel
  - 2x2 up-convolution that halves number of input channels
  - **Skip Connections**: outputs of up-convolutions are concatenated with feature maps from encoder
  - Followed by 2 ordinary 3x3 convs — to process the info
  - final layer: 1x1 conv to map 64 channels to # classes

Height, Width: ↑ Depth: ↓

inverse conv decoder
transpose conv

[Ronneberger et al. MICCAI'15] U-Net

See you next time!

References

We highly recommend to read through these papers!

- AlexNet [Krizhevsky et al. 2012]
- VGGNet [Simonyan & Zisserman 2014]
- ResNet [He et al. 2015]
- GoogLeNet [Szegedy et al. 2014]
- Xception [Chollet 2016]
- Fast R-CNN [Girshick 2015]
- U-Net [Ronneberger et al. 2015]
- EfficientNet [Tan & Le 2019]