

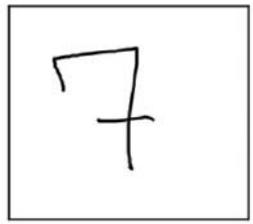


Lecture 10 – CNNs (part 2)

Classic Architectures

LeNet

- Digit recognition: 10 classes



$32 \times 32 \times 1$

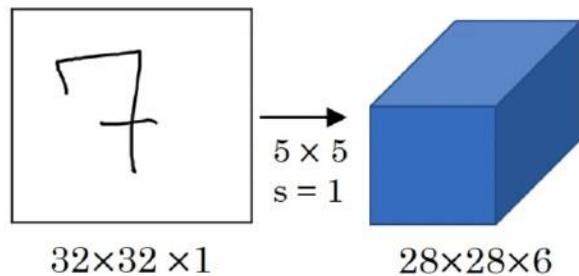
Input: 32×32 grayscale images

This one: Labeled as class "7"

[LeCun et al. '98] LeNet

LeNet

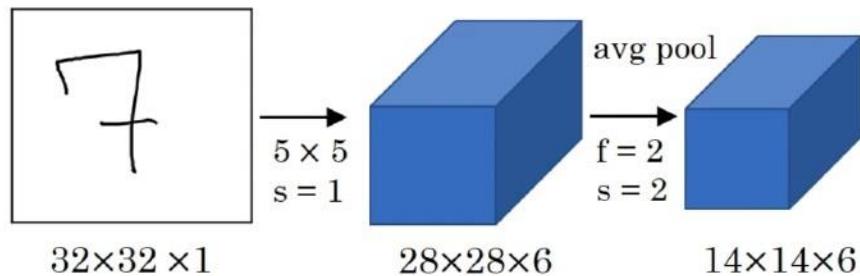
- Digit recognition: 10 classes



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

LeNet

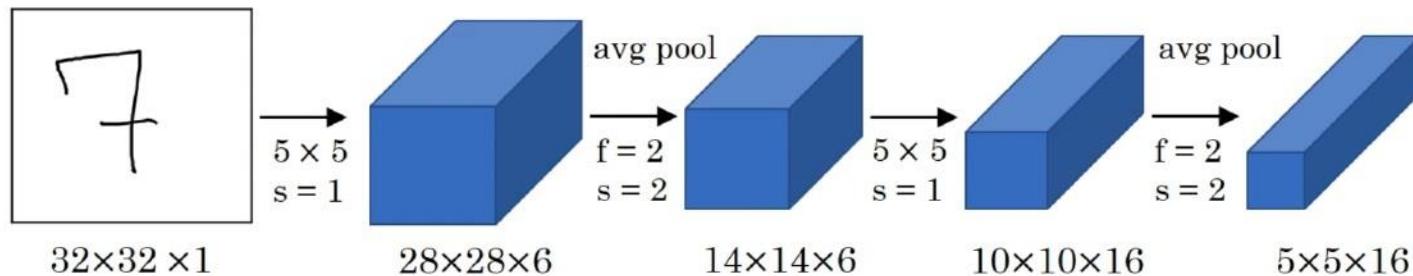
- Digit recognition: 10 classes



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

LeNet

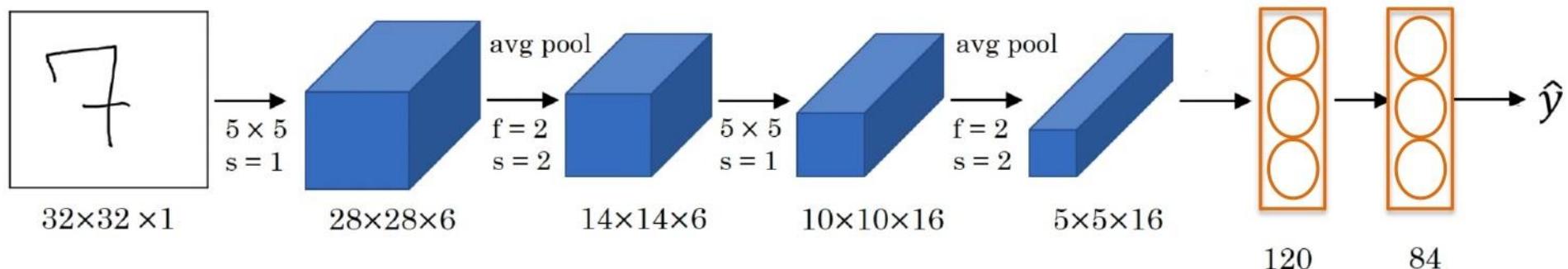
- Digit recognition: 10 classes



- Again valid convolutions, how many filters?

LeNet

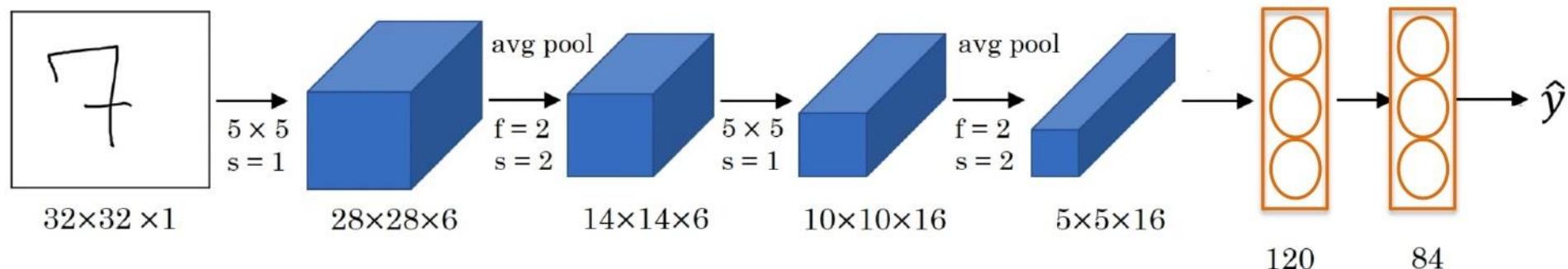
- Digit recognition: 10 classes



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

LeNet

- Digit recognition: 10 classes

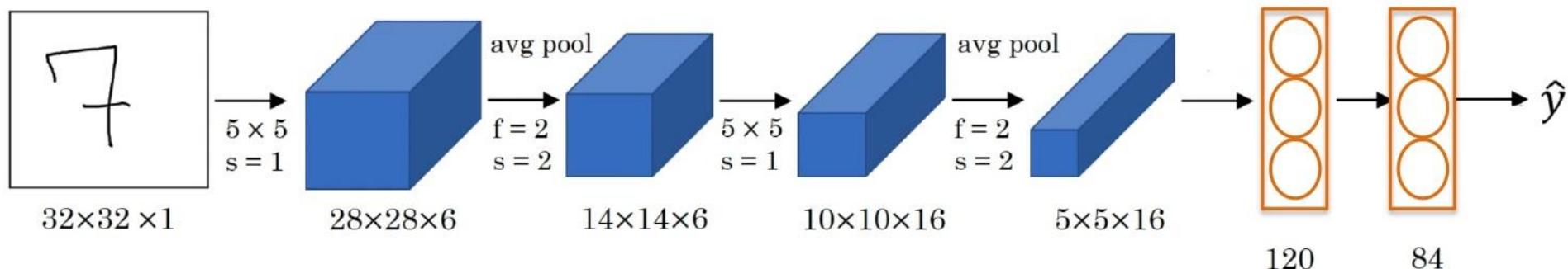


- Conv -> Pool -> Conv -> Pool -> Conv -> FC

LeNet

- Digit recognition: 10 classes

60k parameters

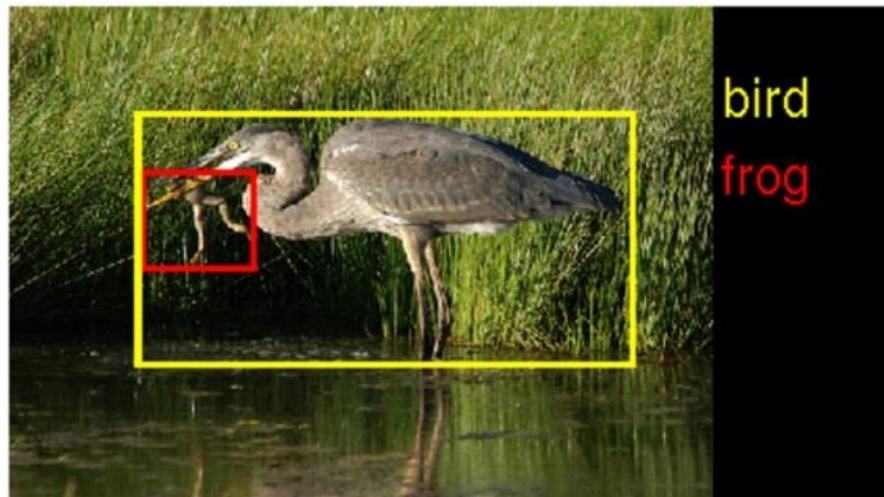


- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, Height \downarrow Number of Filters \uparrow

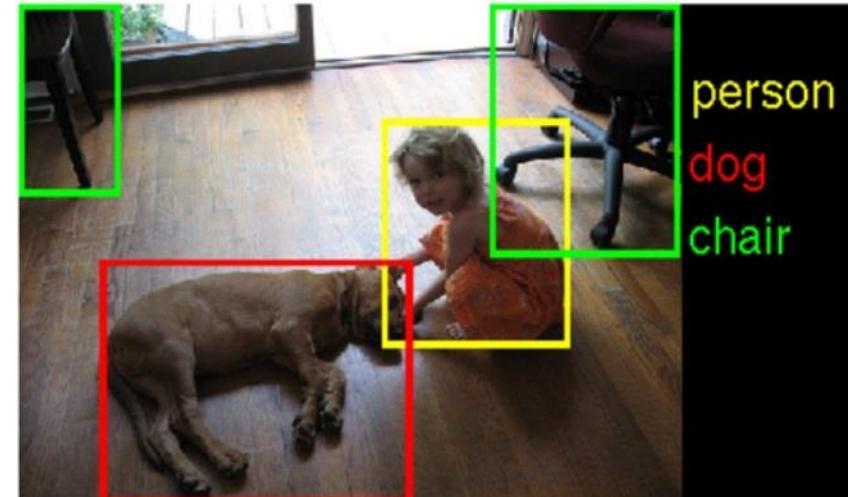
Test Benchmarks

- ImageNet Dataset:

ImageNet Large Scale Visual Recognition Competition (ILSVRC)



bird
frog



person
dog
chair

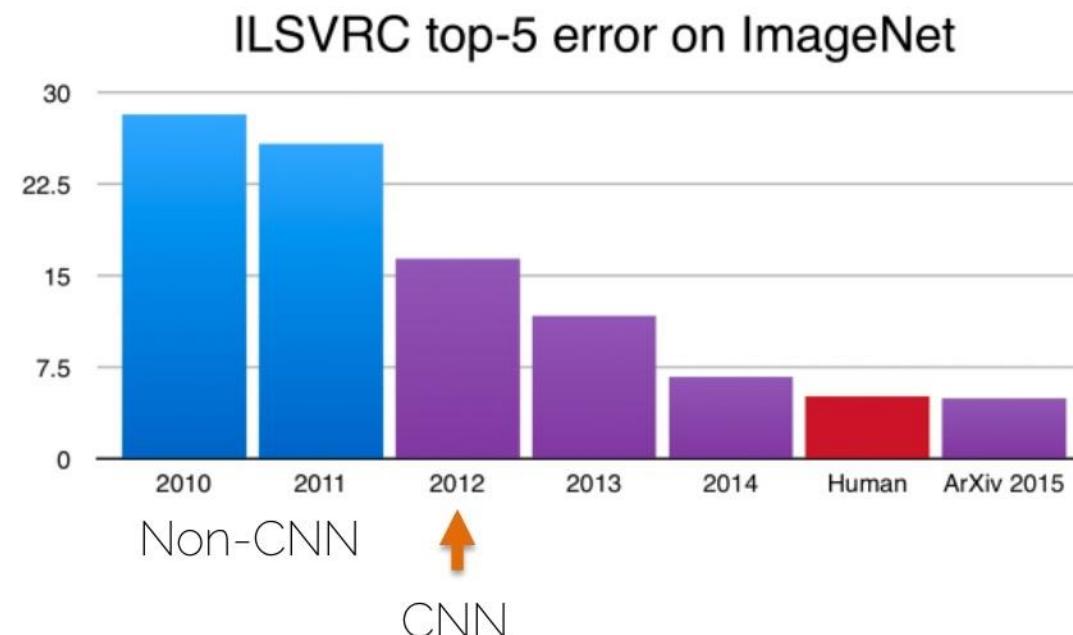
[Russakovsky et al., IJCV'15] "ImageNet Large Scale Visual Recognition Challenge."

Common Performance Metrics

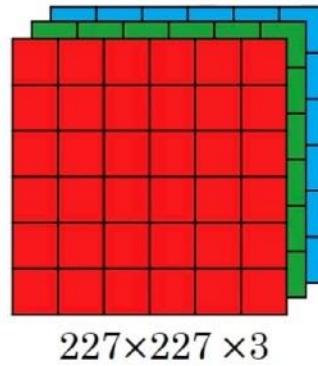
- Top-1 score: check if a sample's top class (i.e. the one with highest probability) is the same as its target label
- Top-5 score: check if your label is in your 5 first predictions (i.e. predictions with 5 highest probabilities)
- → Top-5 error: percentage of test samples for which the correct class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

AlexNet

- Cut ImageNet error down in half

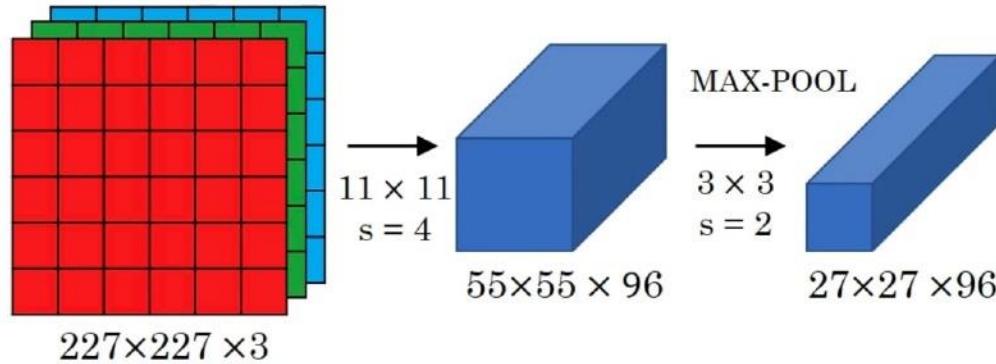


AlexNet



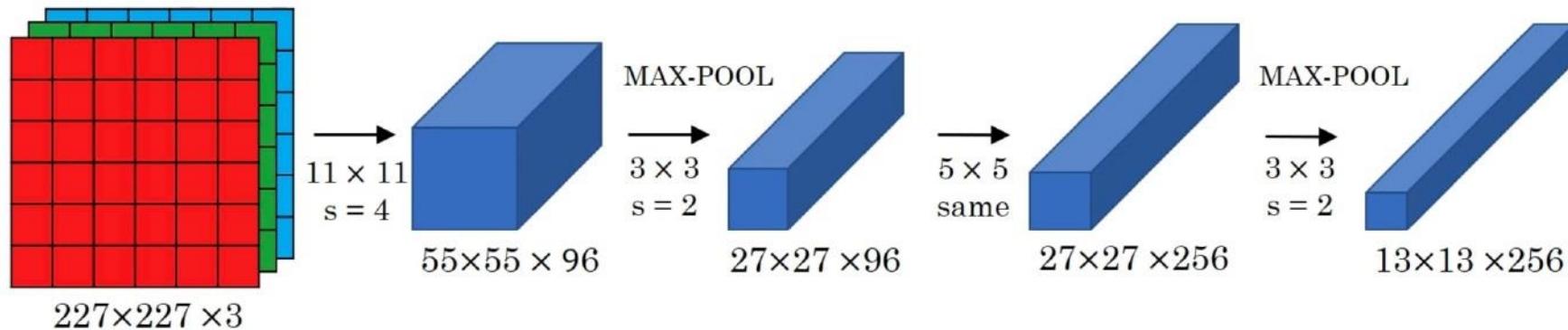
[Krizhevsky et al. NIPS'12] AlexNet

AlexNet



[Krizhevsky et al. NIPS'12] AlexNet

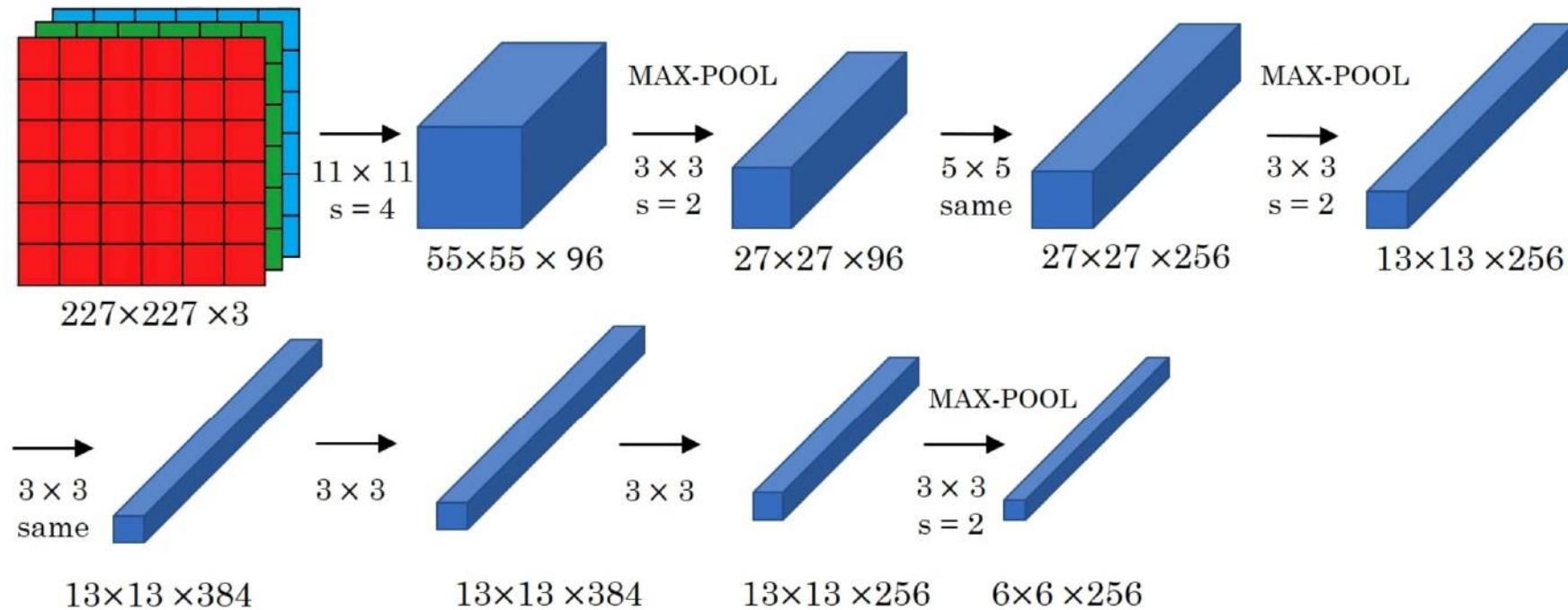
AlexNet



- Use of same convolutions
- As with LeNet: Width, Height ↓ Number of Filters ↑

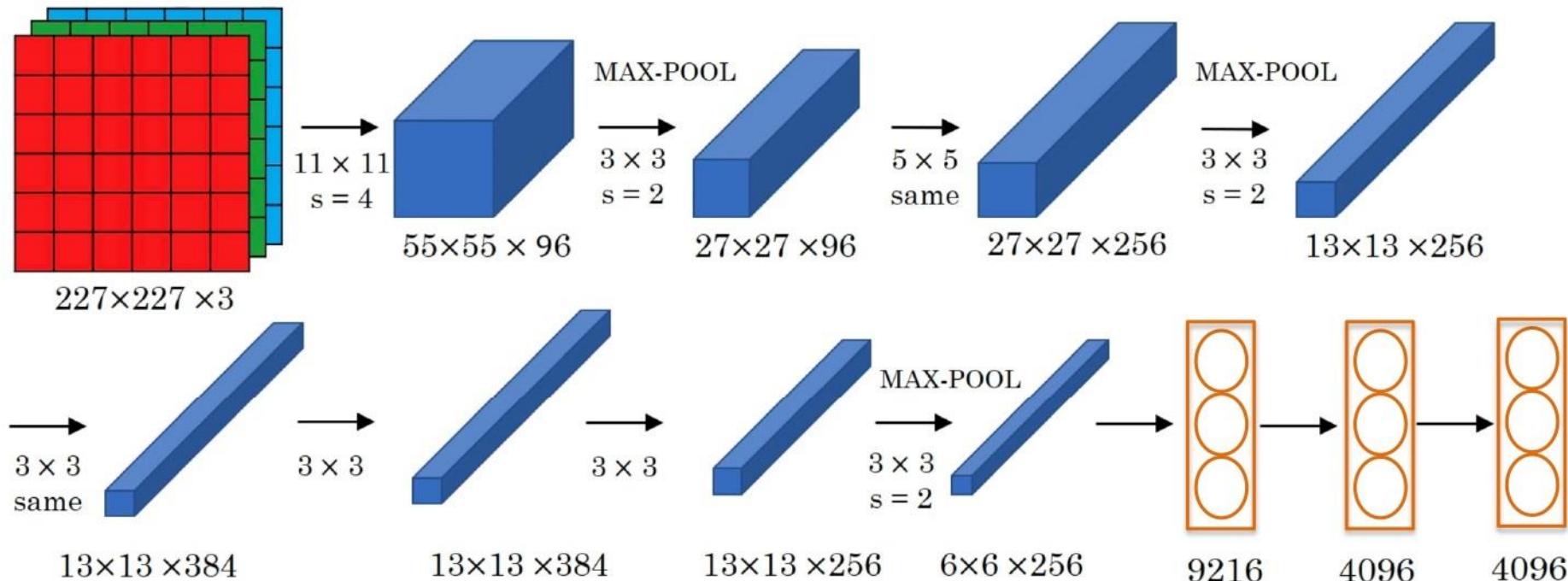
[Krizhevsky et al. NIPS'12] AlexNet

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

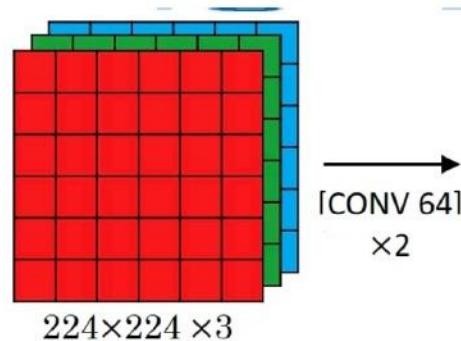
VGGNet

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

[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

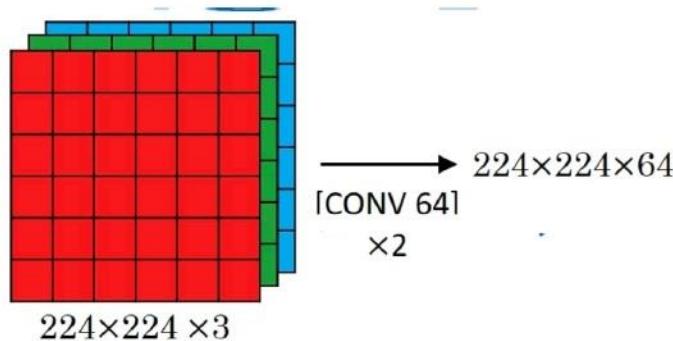
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

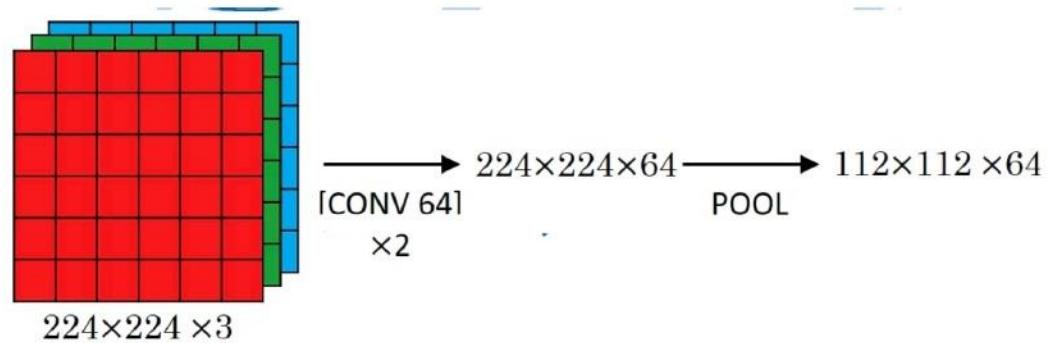
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

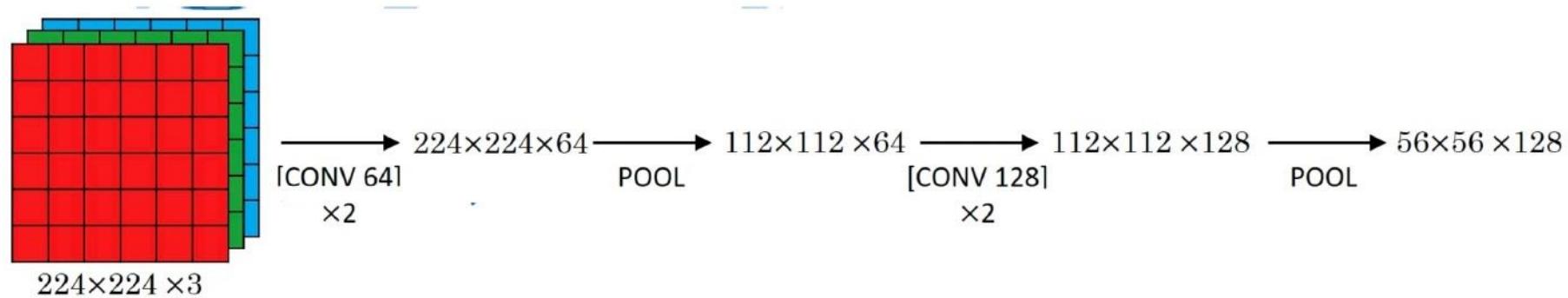
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

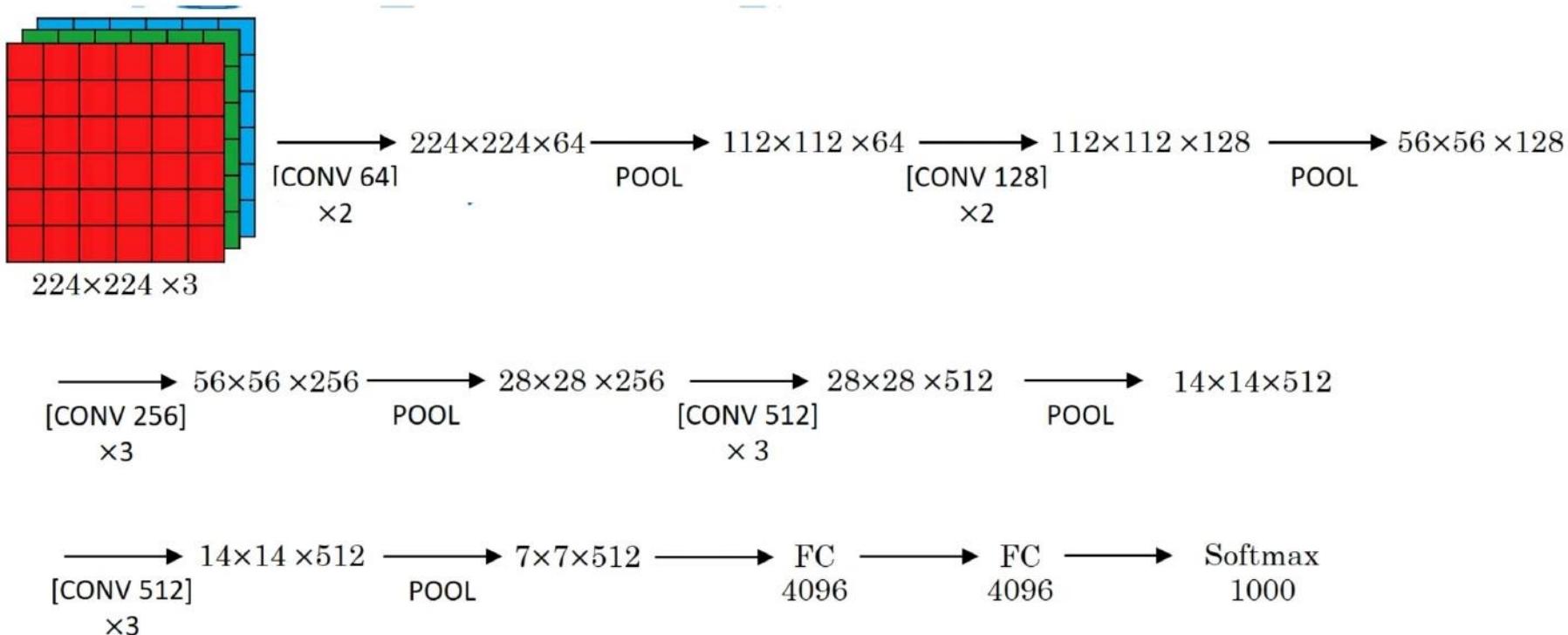
Conv=3X3,S=1,same
Maxpool=2x2,S=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

Conv=3X3,S=1,same
Maxpool=2x2,S=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

- A lot of architectures were analyzed

[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 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

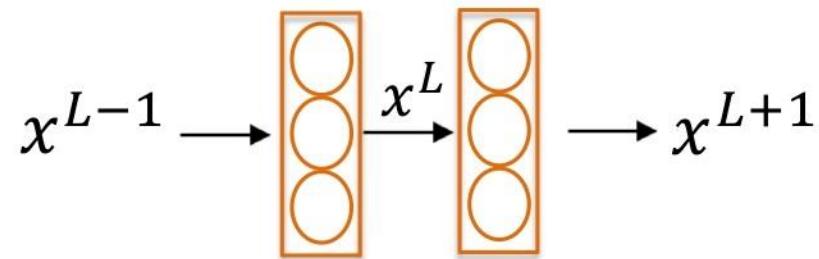
Skip Connections

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



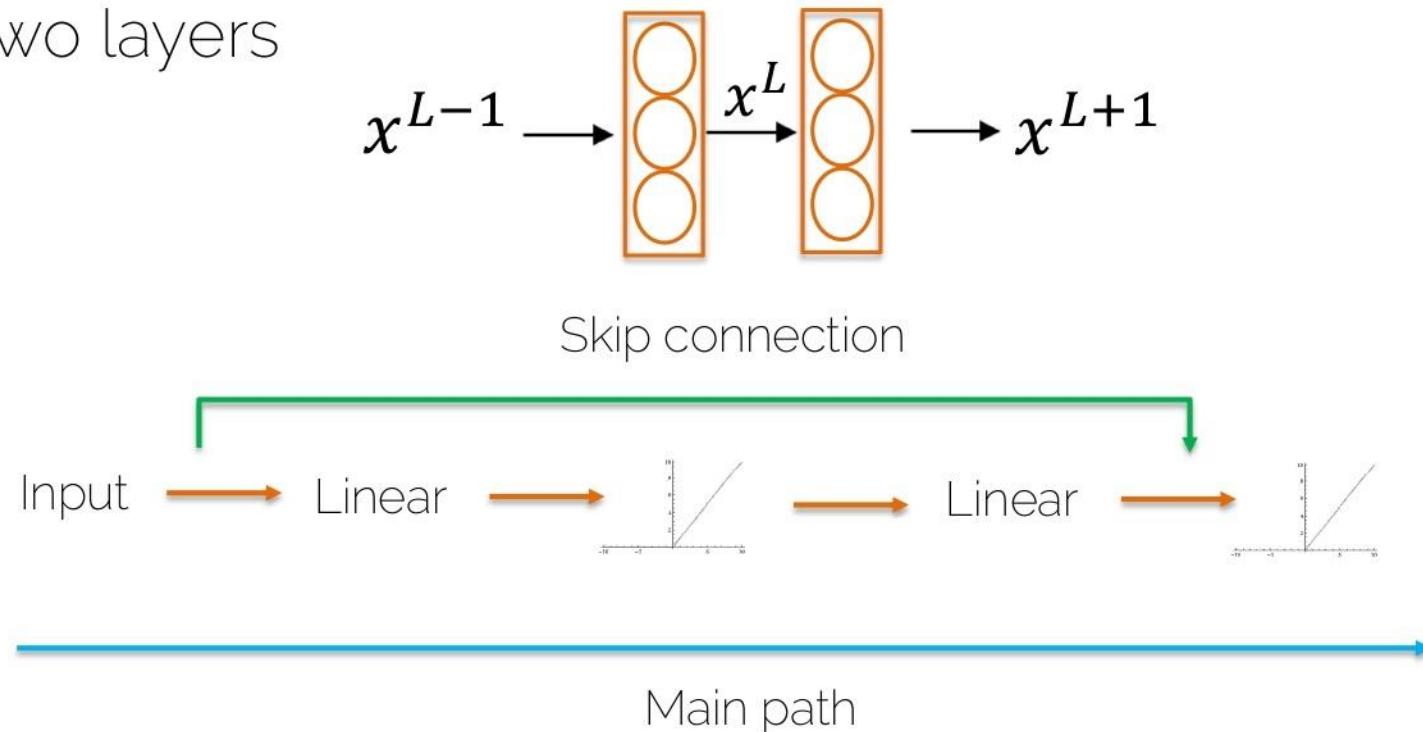
Input $\longrightarrow W^L x^{L-1} + b^L \xrightarrow{\text{Linear}} x^L = f(W^L x^{L-1} + b^L) \xrightarrow{\text{Non-linearity}}$

A small graph showing a linear function $f(x) = x$ on a coordinate plane. The x-axis ranges from -10 to 10, and the y-axis ranges from -10 to 10. A straight line passes through the origin (0,0) with a slope of 1, representing the identity function.

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

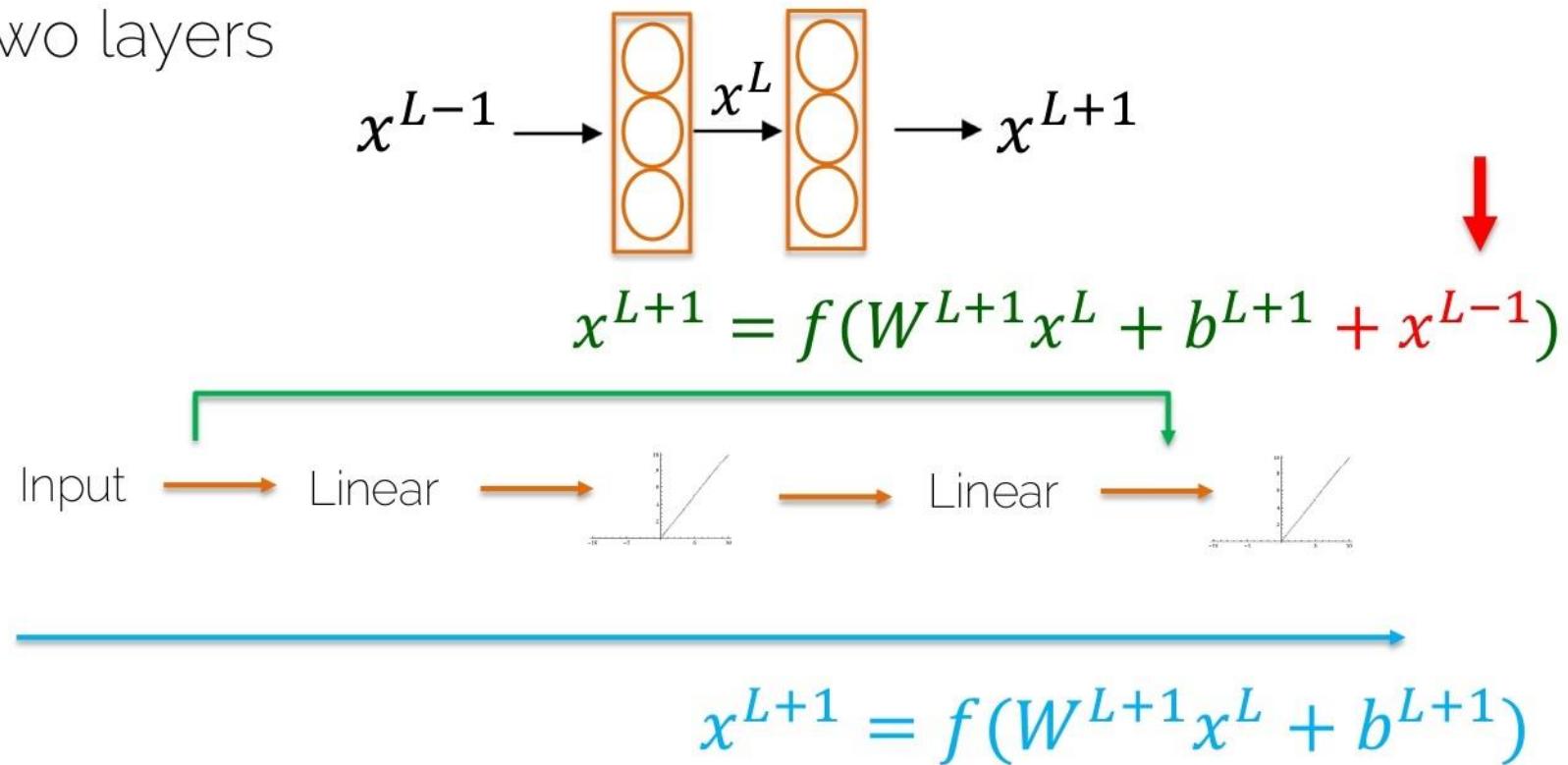
Residual Block

- Two layers



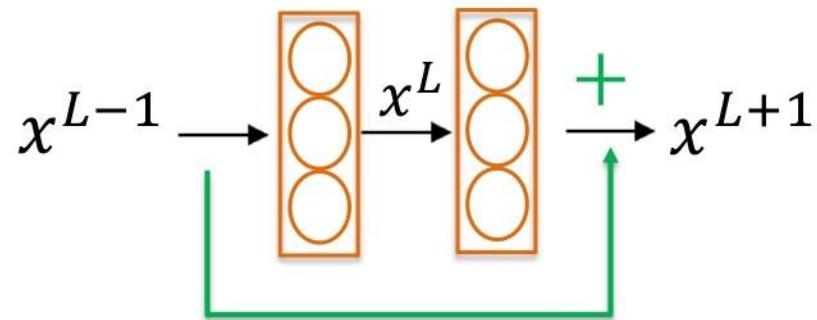
Residual Block

- Two layers



Residual Block

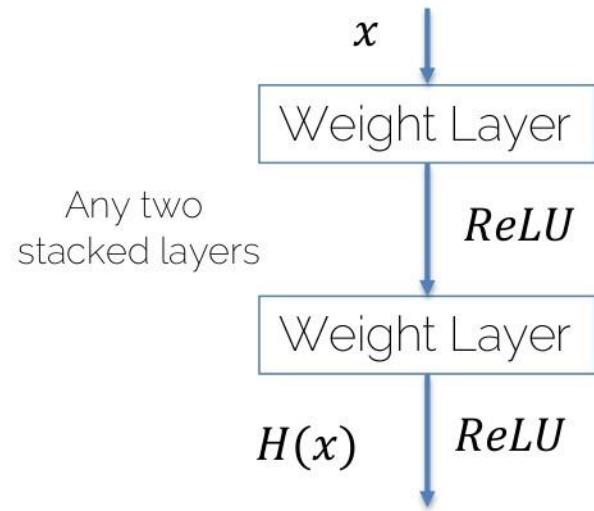
- Two layers



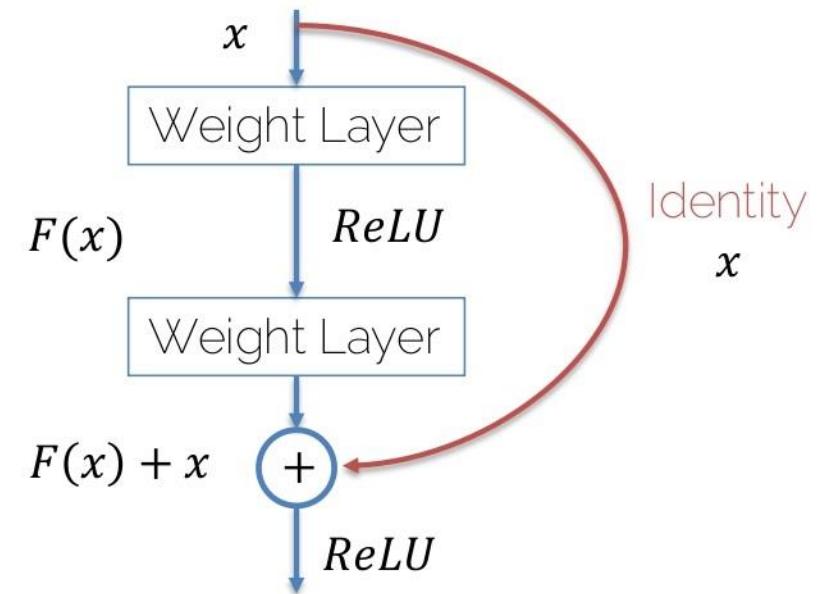
- Usually use a same convolution since we need same dimensions
- Otherwise we need to convert the dimensions with a matrix of learned weights or zero padding

ResNet Block

Plain Net

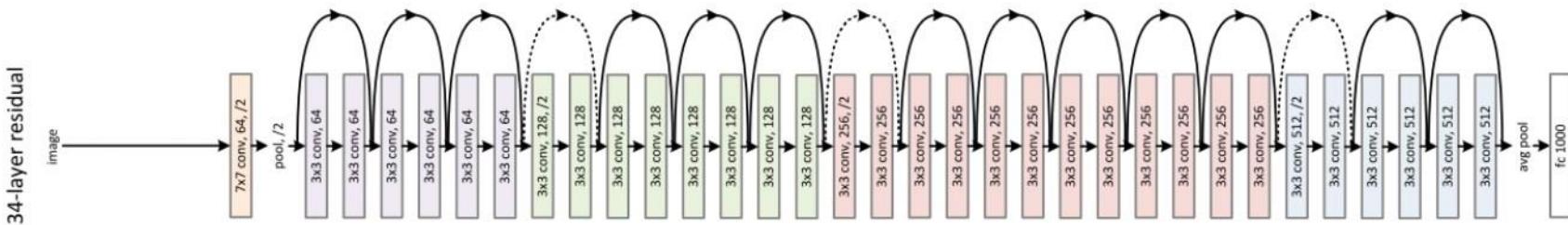


Residual Net



[He et al. CVPR'16] ResNet

ResNet



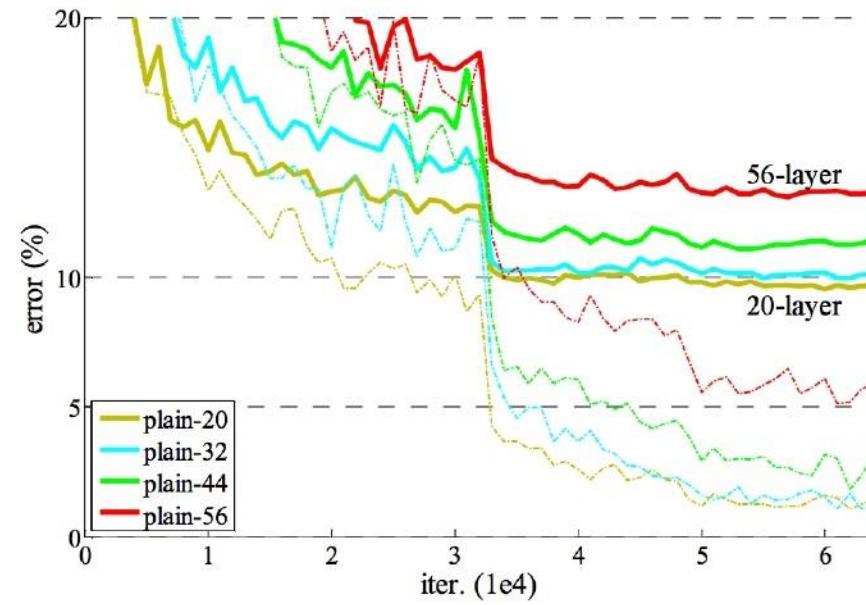
- Xavier/2 initialization
- SGD + Momentum (0.9)
- Learning rate 0.1, divided by 10 when plateau
- Mini-batch size 256
- Weight decay of $1e-5$
- No dropout

ResNet-152:
60M parameters

[He et al. CVPR'16] ResNet

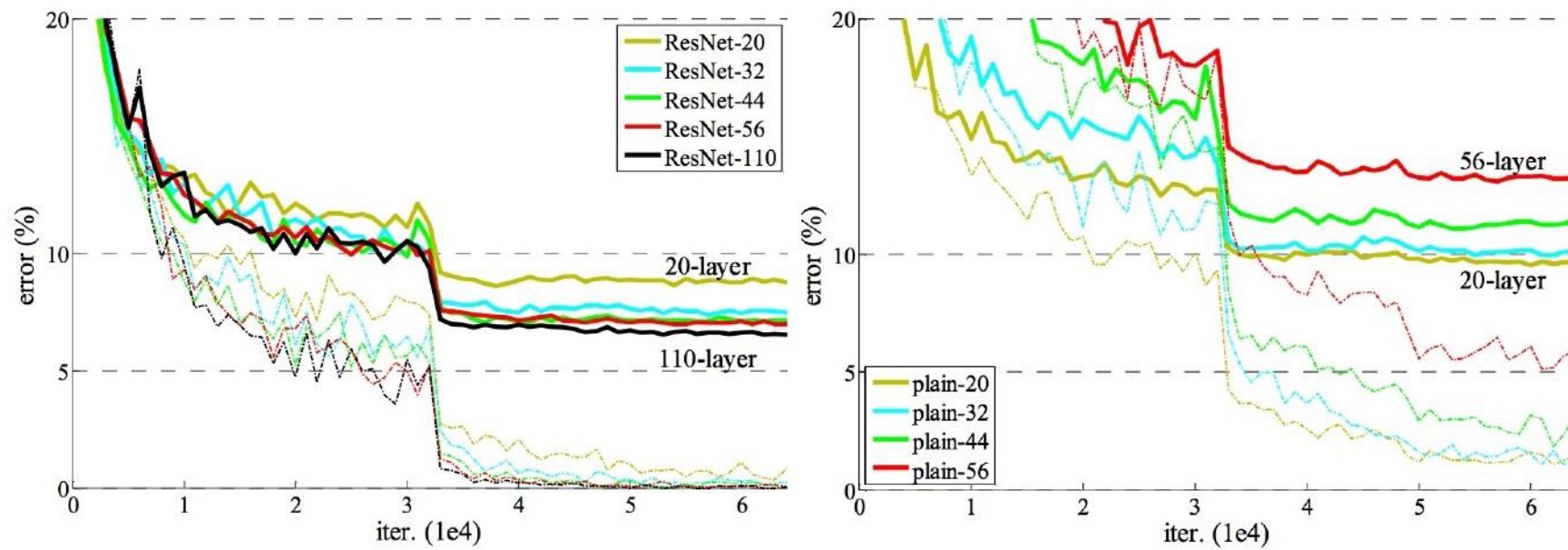
ResNet

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

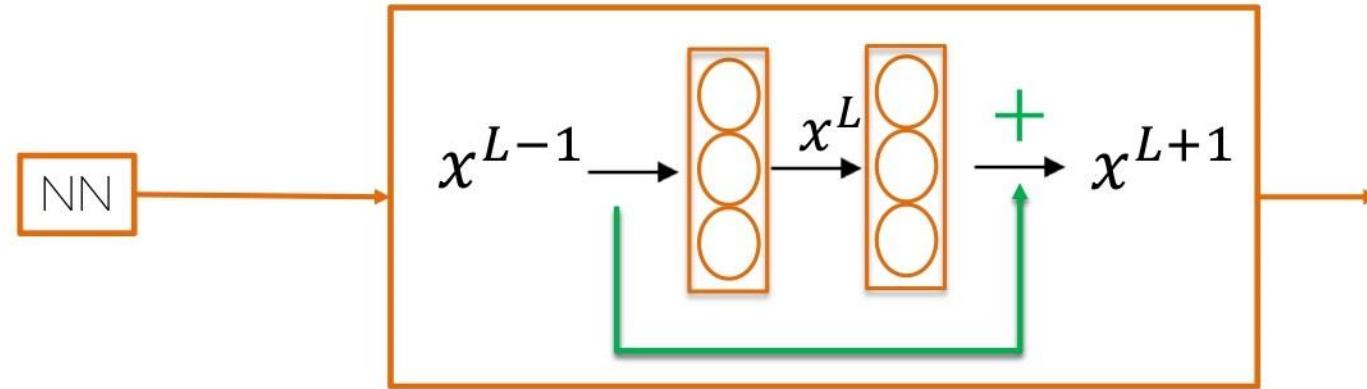


ResNet

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

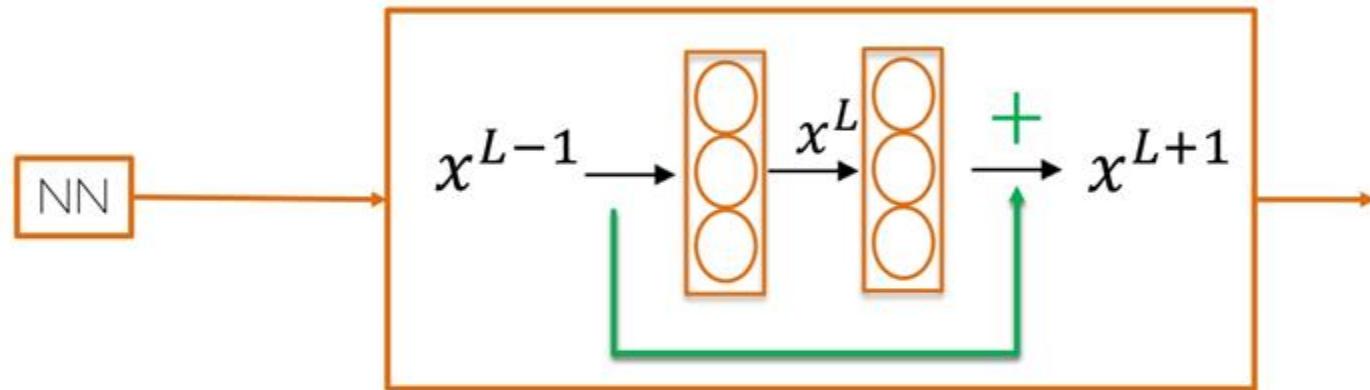


Why do ResNets Work?



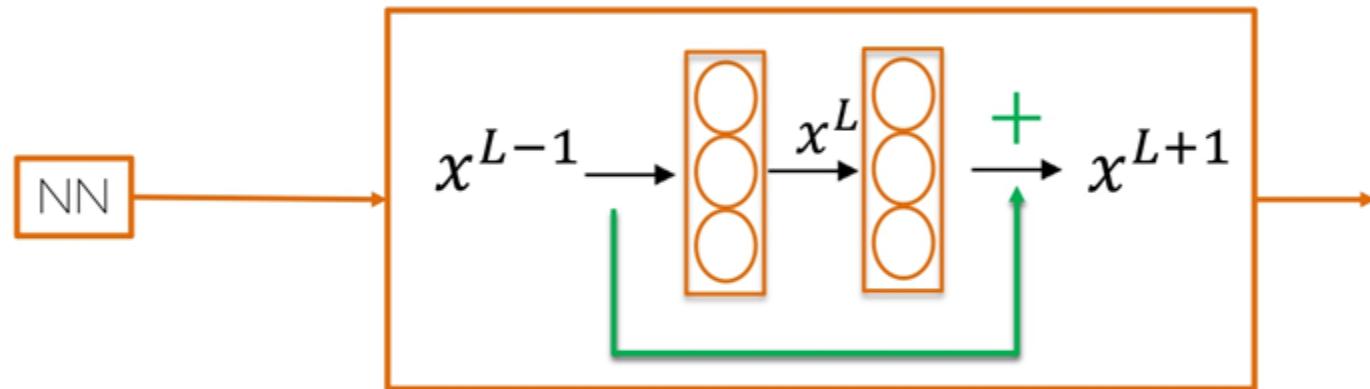
- How is this block really affecting me?

Why do ResNets Work?



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

Why do ResNets Work?

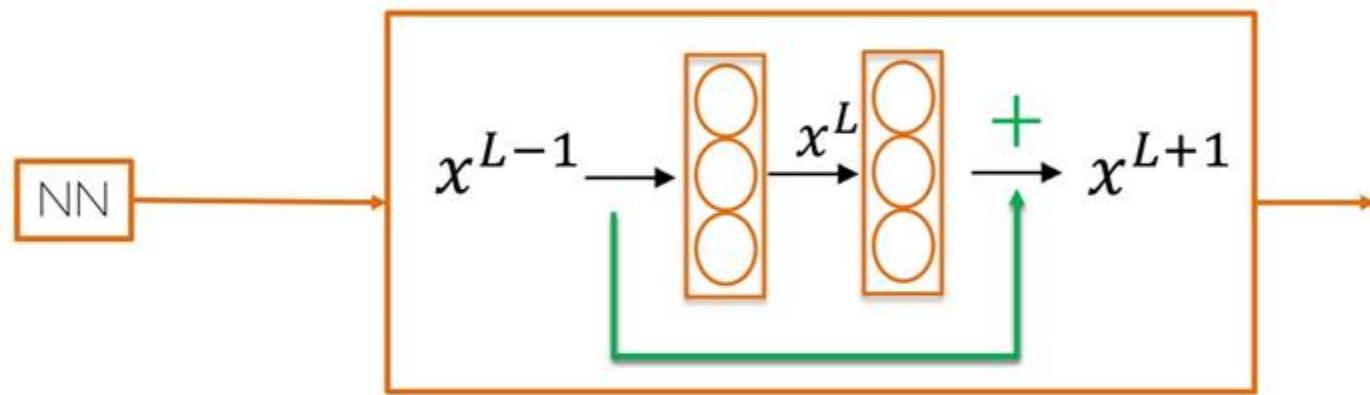


$$x^{L+1} = f(W^{L+1}x^L + b^{L+1} + x^{L-1})$$

~zero ~zero

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

Why do ResNets Work?

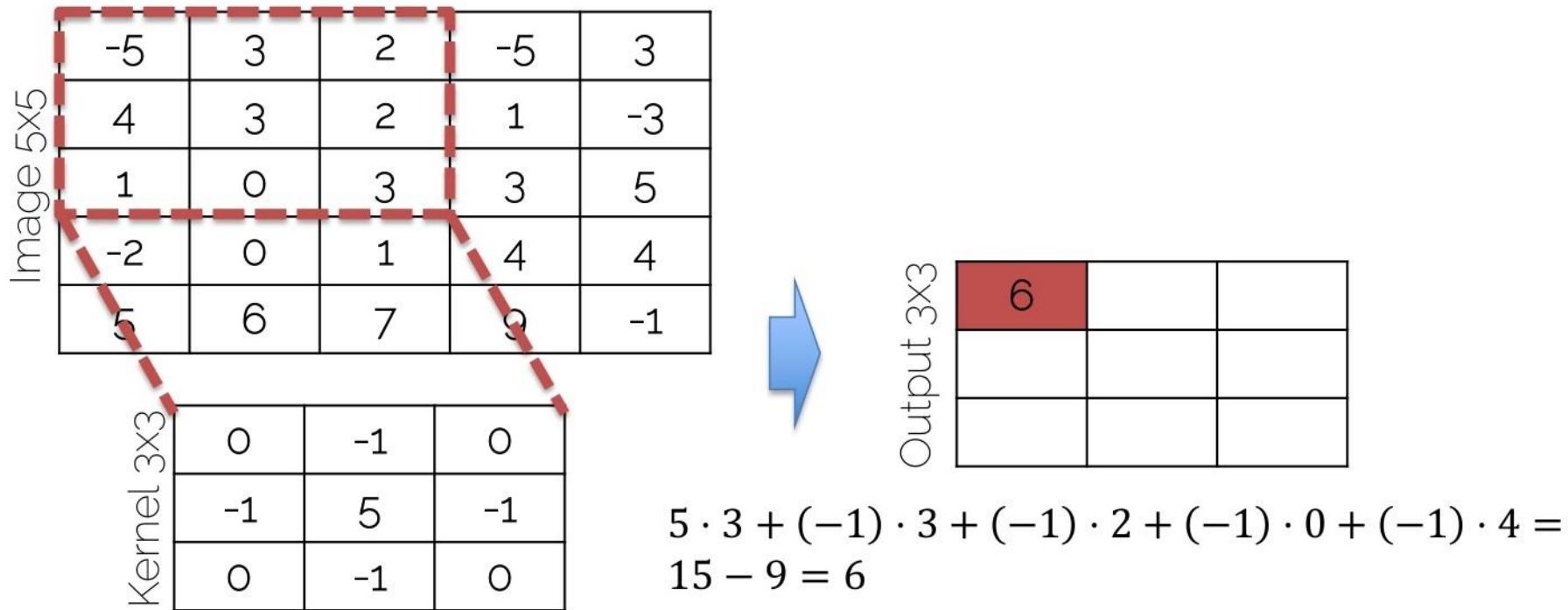


- We kept the same values and added a non-linearity

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

1x1 Convolutions

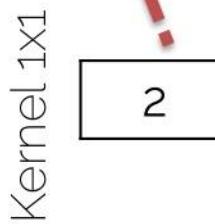
Recall: Convolutions on Images



1x1 Convolution

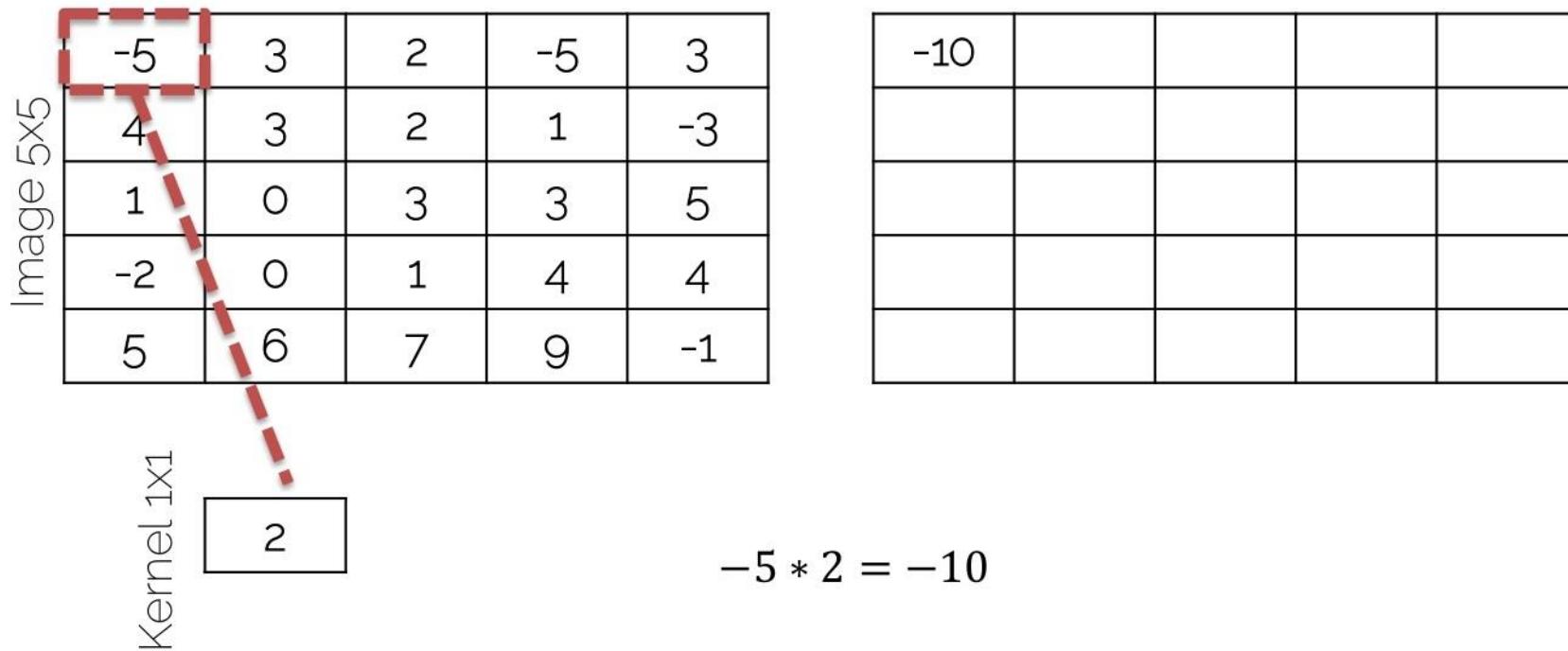
Image 5x5

-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

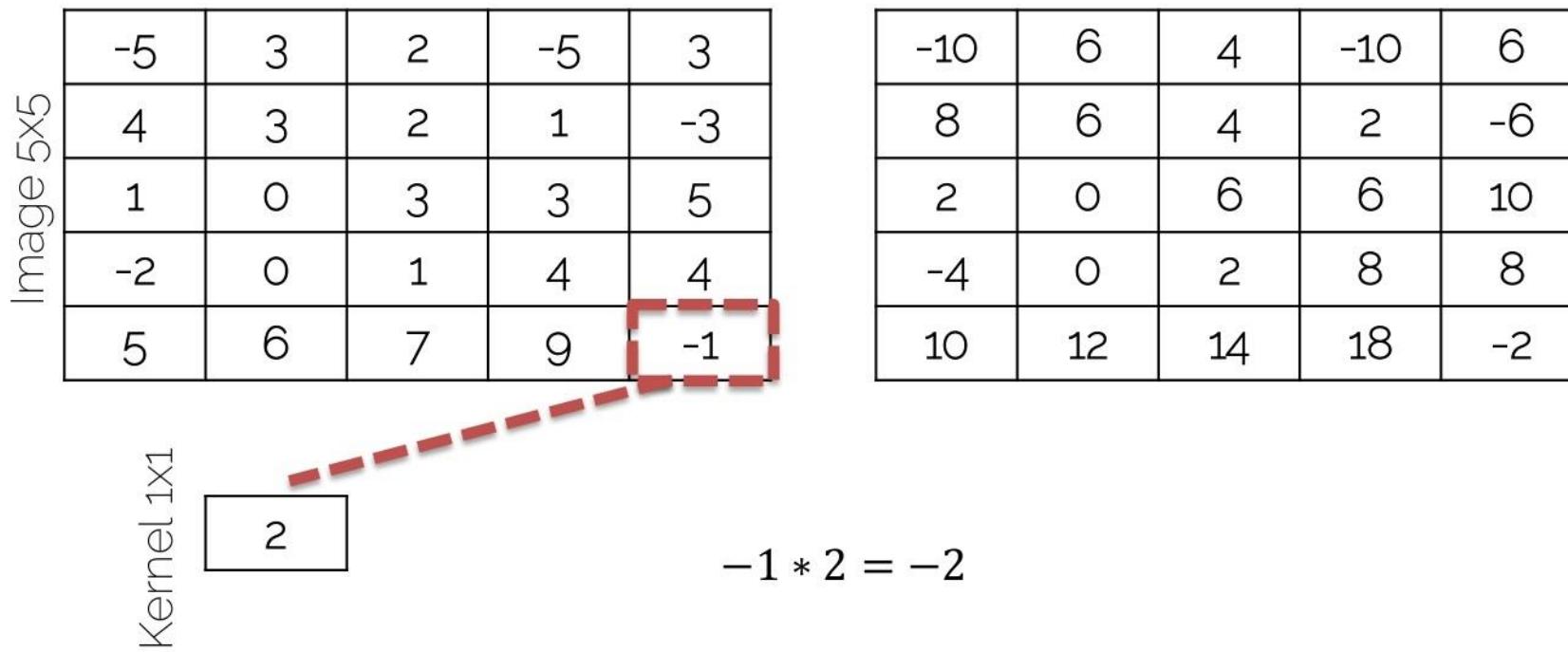


What is the output size?

1x1 Convolution



1x1 Convolution



1x1 Convolution

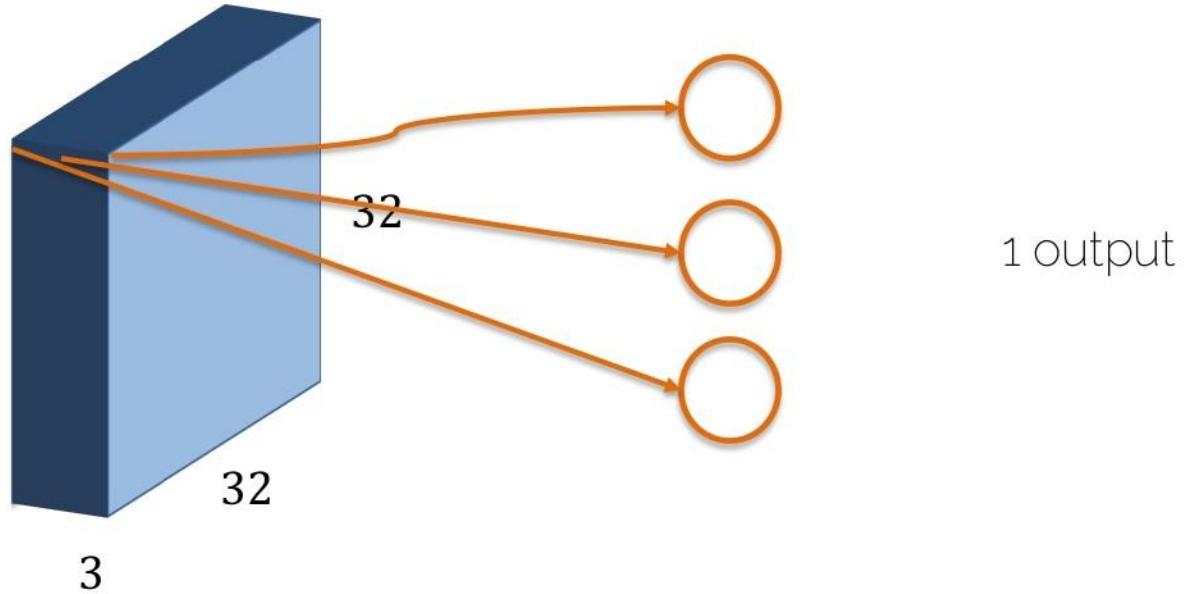
Image 5x5

-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

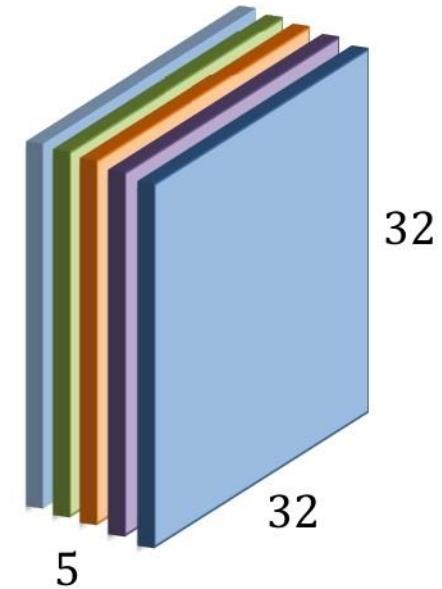
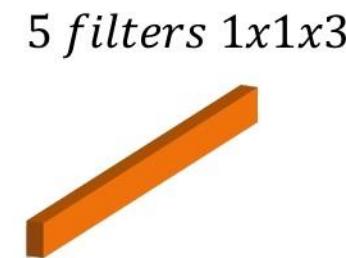
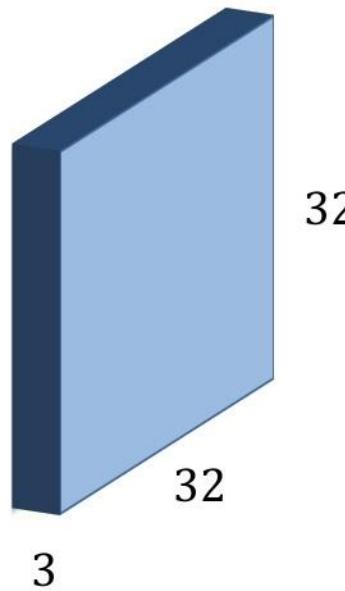
1x1 Convolution



- Same as having a 3 neuron fully connected layer

1x1 Convolution

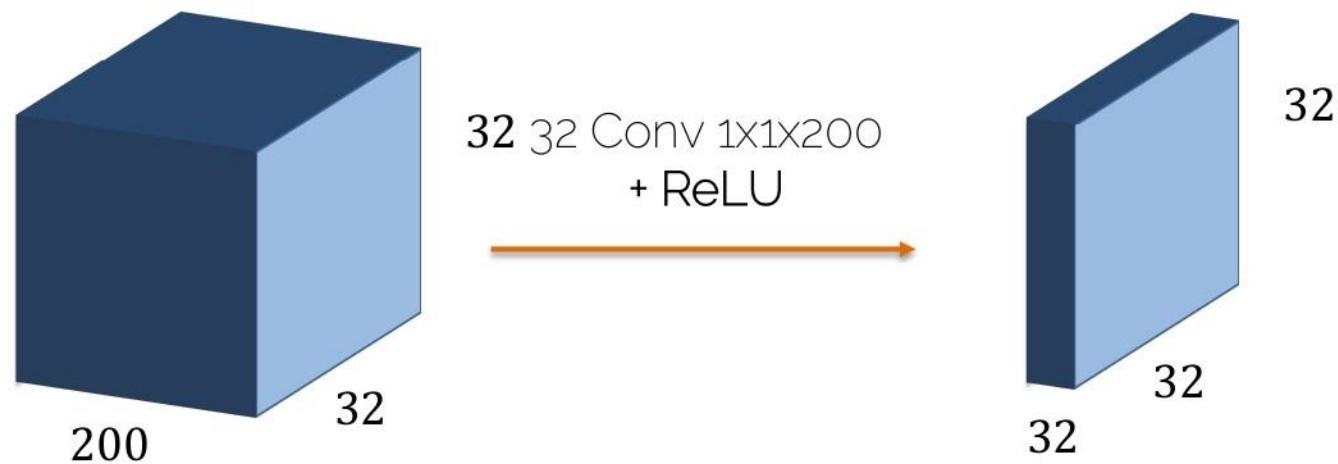
[Li et al. 2013]



- As always we use more convolutional filters

Using 1x1 Convolutions

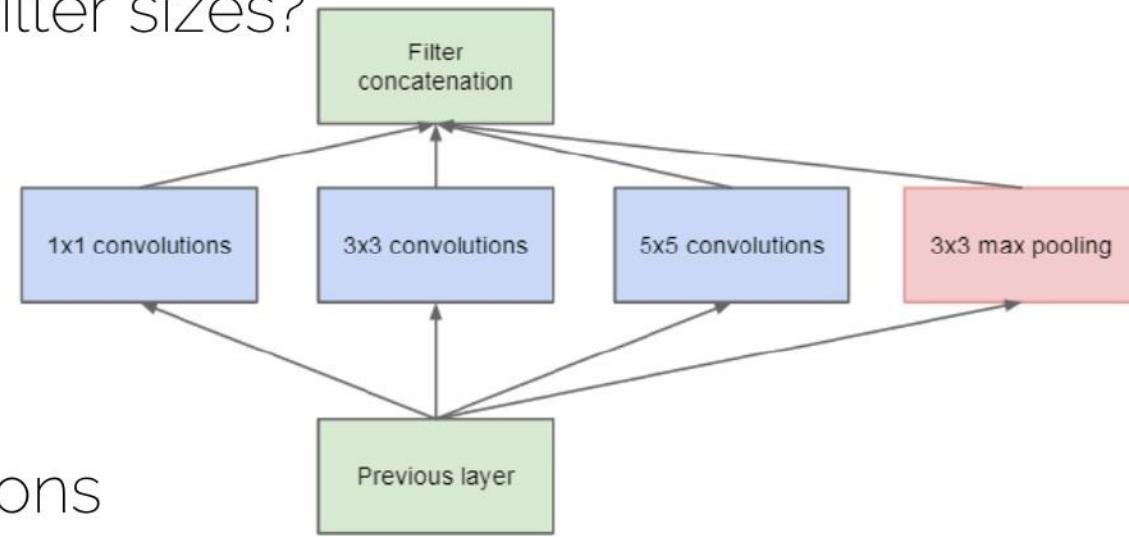
- Use it to shrink the number of channels
- Further adds a non-linearity → one can learn more complex functions



Inception Layer

Inception Layer

- Tired of choosing filter sizes?



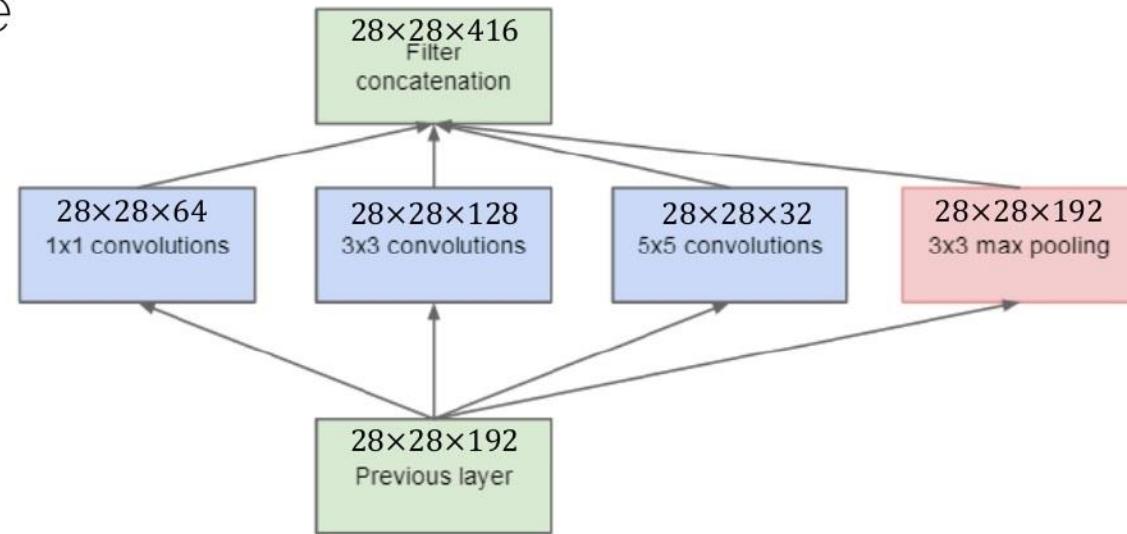
- Use them all!

- All same convolutions

- 3×3 max pooling is with stride 1

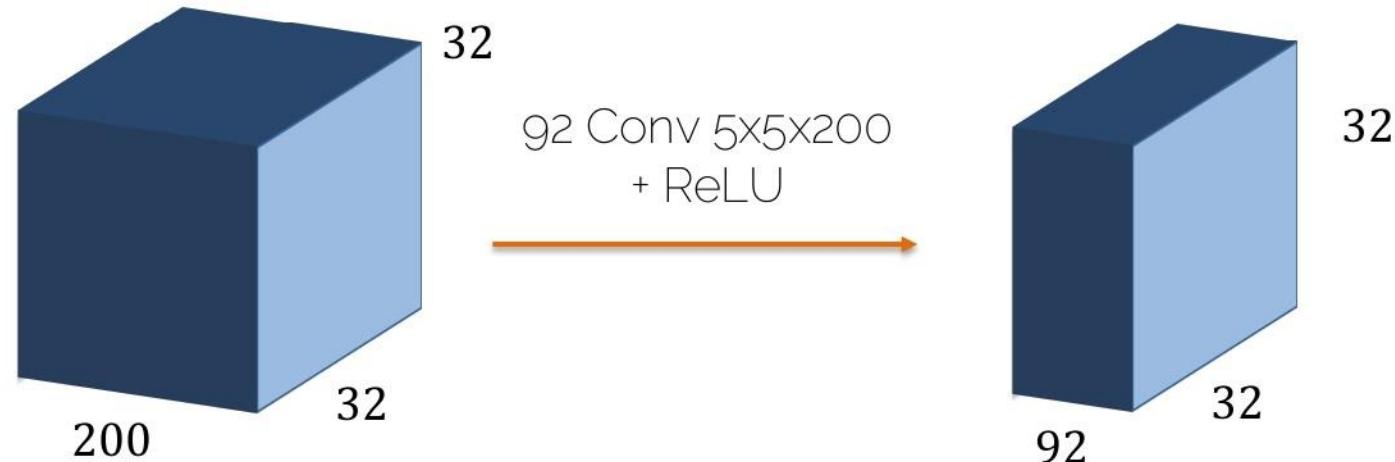
Inception Layer

- Possible size of the output



- Not sustainable!

Inception Layer: Computational Cost

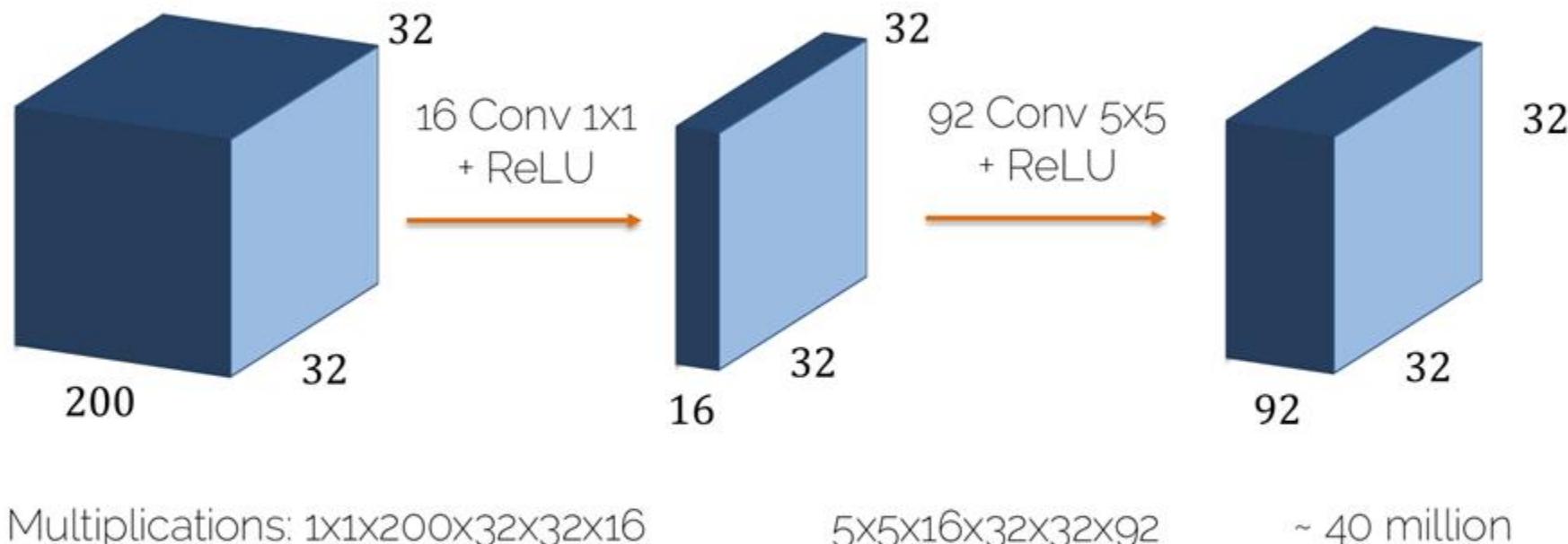


Multiplications: $5 \times 5 \times 200 \times 32 \times 32 \times 92 \sim 470 \text{ million}$



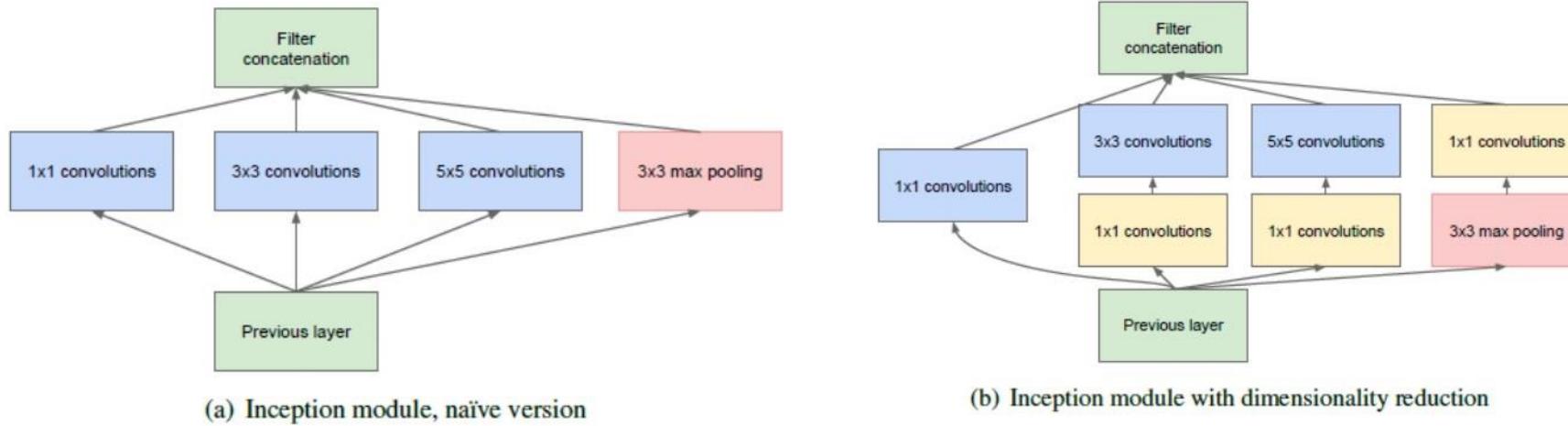
1 value of the output volume

Inception Layer: Computational Cost



Reduction of multiplications by 1/10

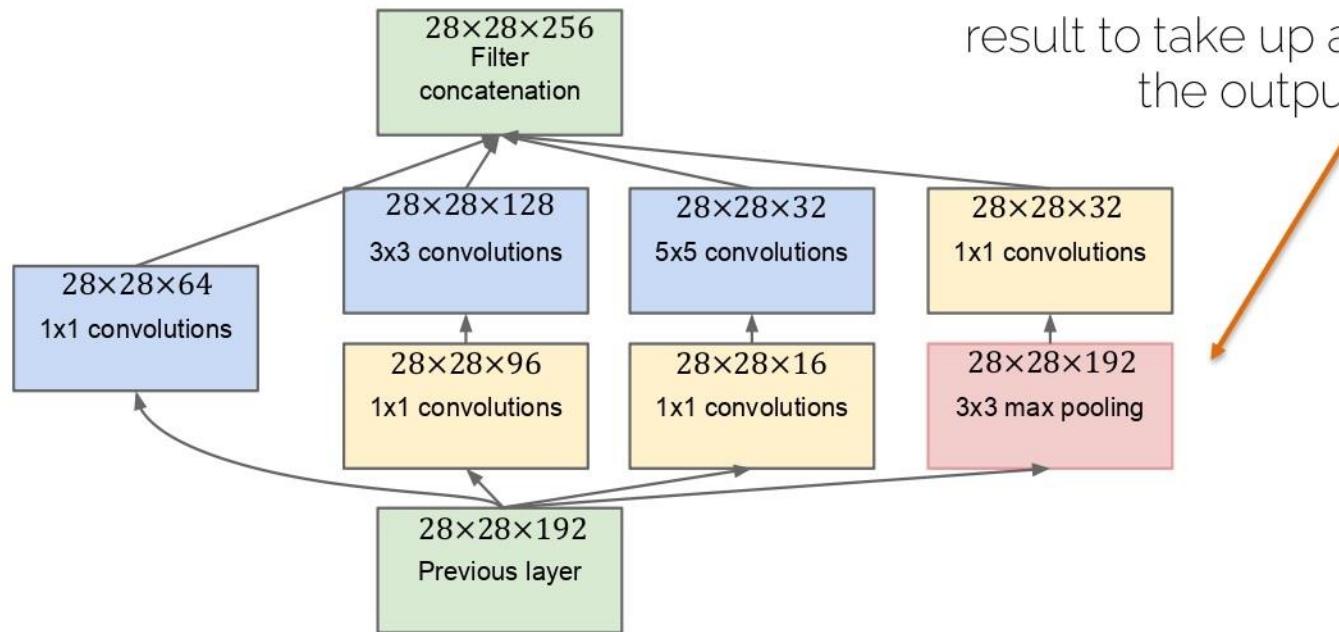
Inception Layer



[Szegedy et al CVPR'15] GoogLeNet

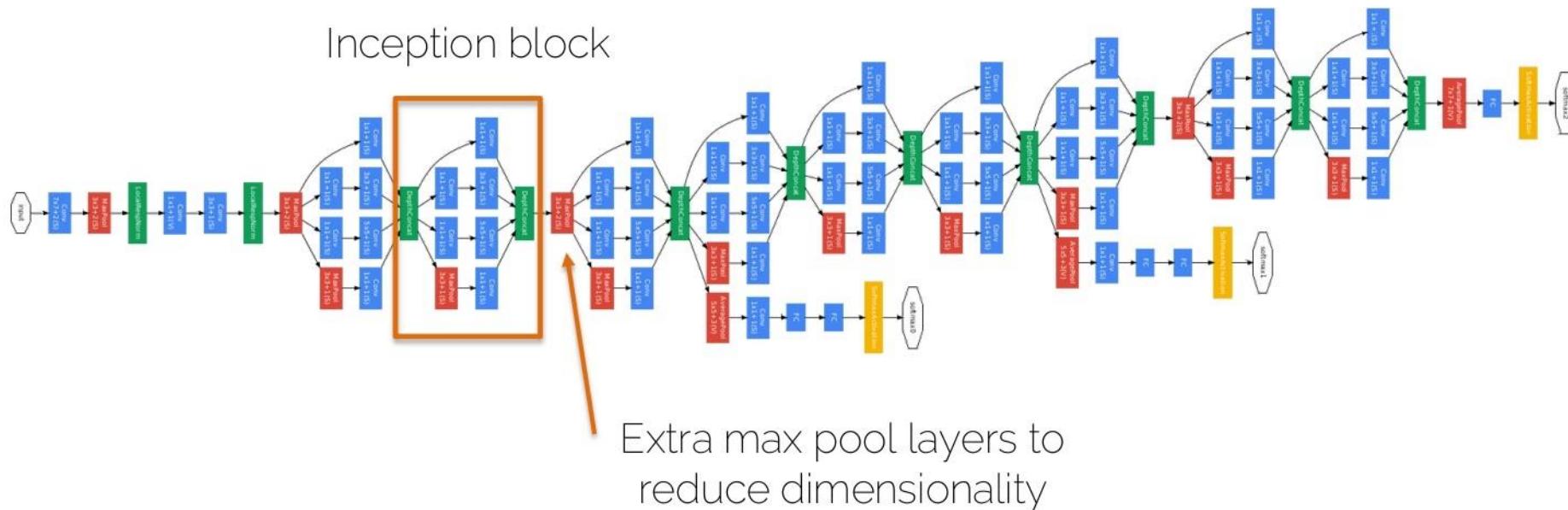
Inception Layer: Dimensions

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



[Szegedy et al CVPR'15] GoogLeNet

GoogLeNet: Using the Inception Layer



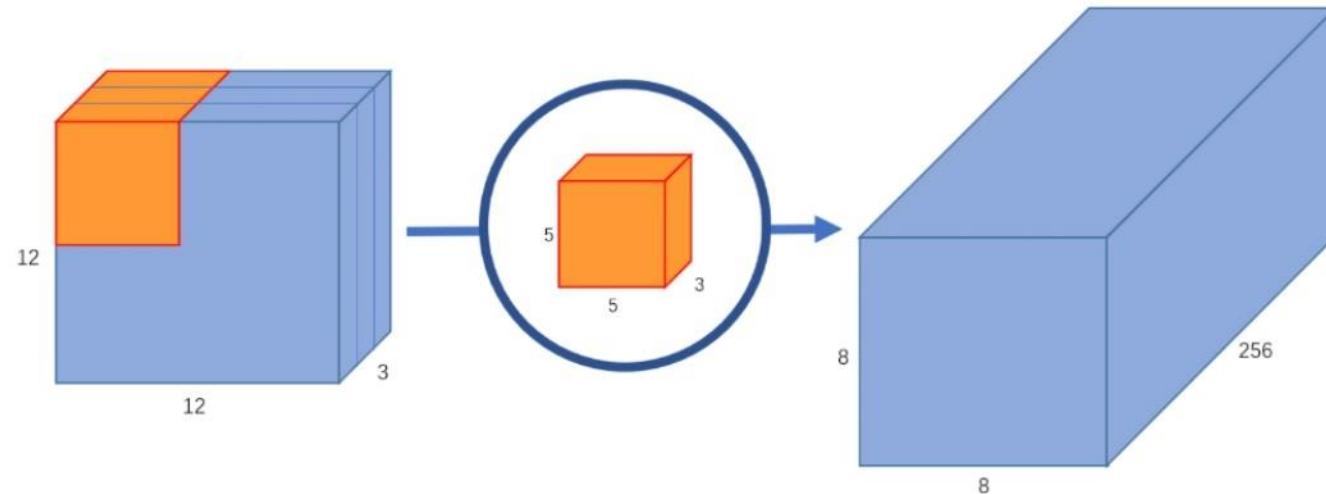
[Szegedy et al CVPR'15] GoogLeNet

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

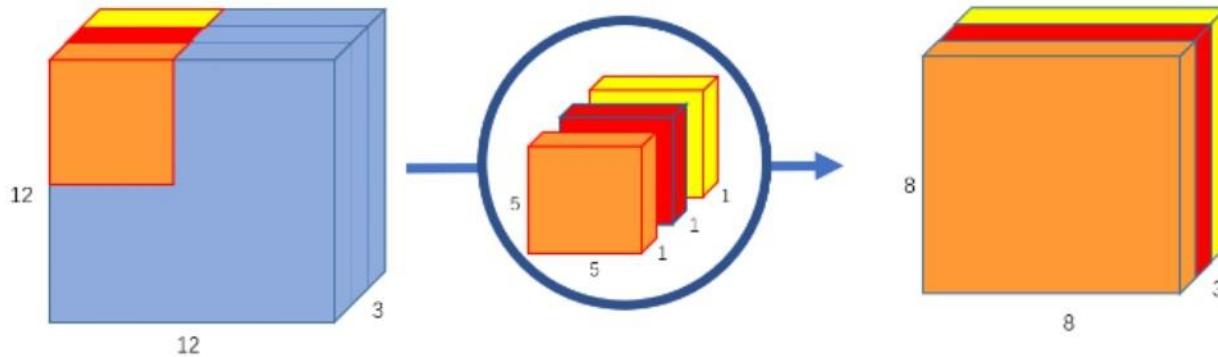
[Chollet CVPR'17] Xception

Depth-wise separable convolutions



Normal convolutions act on all channels.

Depth-wise separable convolutions

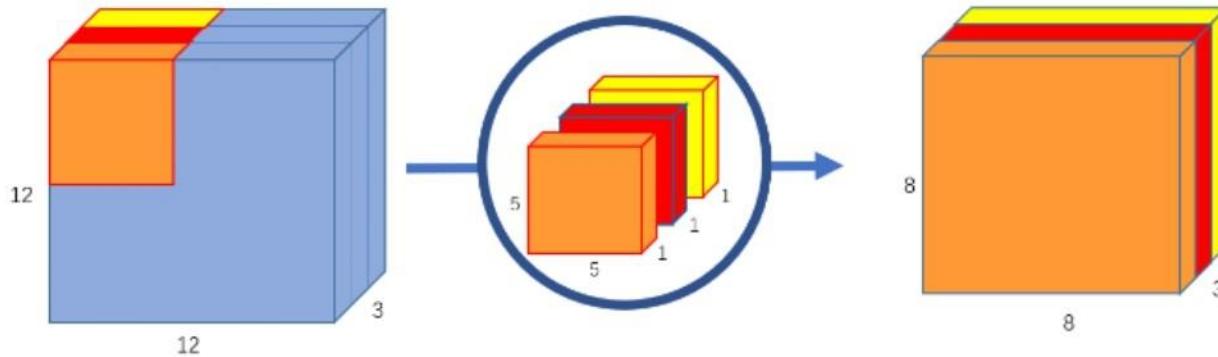


Filters are applied only at certain depths of the features.
Normal convolutions have groups set to 1, the convolutions used in this image have groups set to 3.

```
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

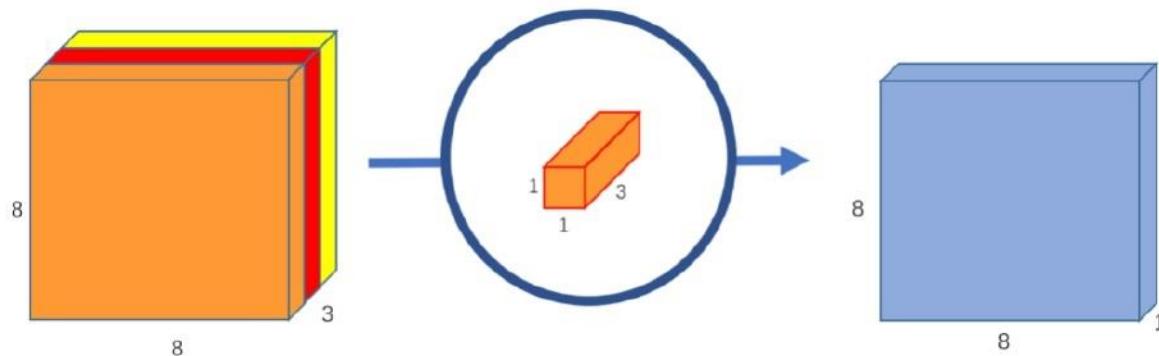
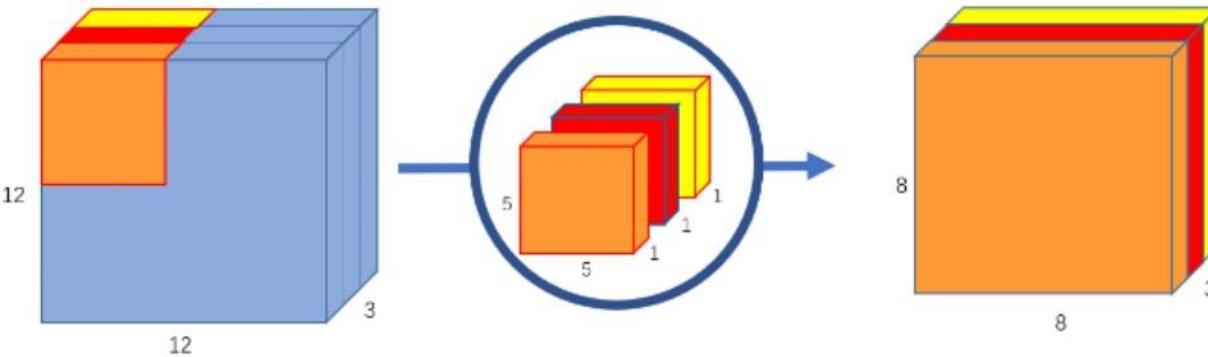


But the depth size is always the same!

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

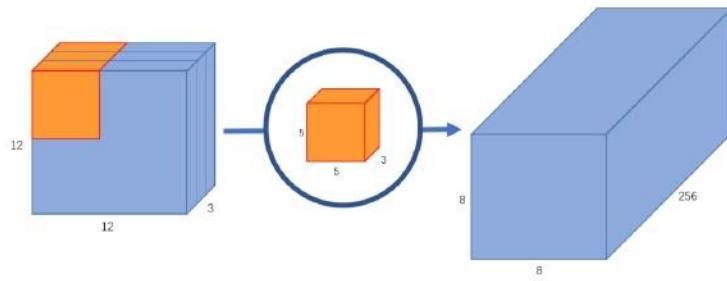
```
class torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)
```

Depth-wise separable convolutions



Solution:
1x1 convs!

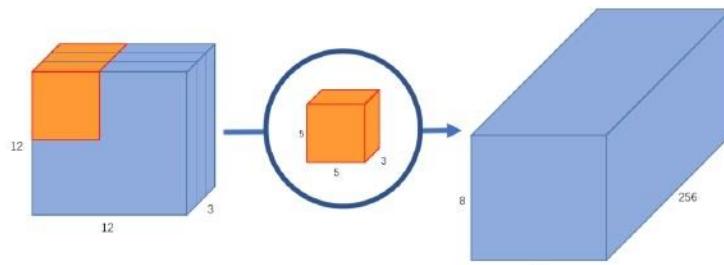
But why?



Original convolution
256 kernels of size 5x5x3

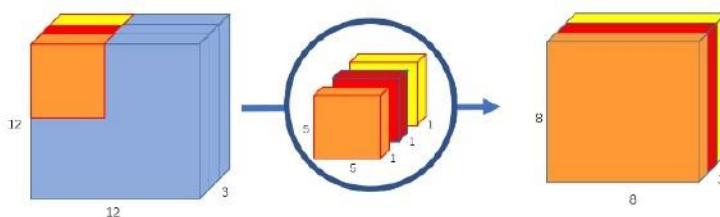
Multiplications:
 $256 \times 5 \times 5 \times 3 \times (8 \times 8 \text{ locations}) = 1.228.800$

But why?



Original convolution
256 kernels of size 5x5x3

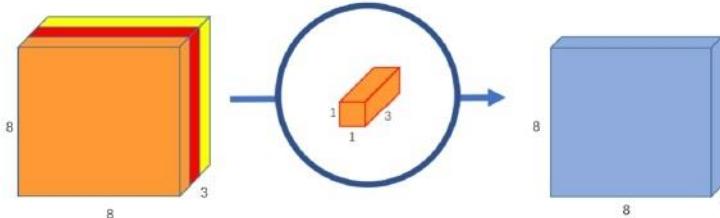
Multiplications:
 $256 \times 5 \times 5 \times 3 \times (8 \times 8 \text{ locations}) = 1.228.800$



Depth-wise convolution
3 kernels of size 5x5x1

Multiplications:
 $5 \times 5 \times 3 \times (8 \times 8 \text{ locations}) = 4800$

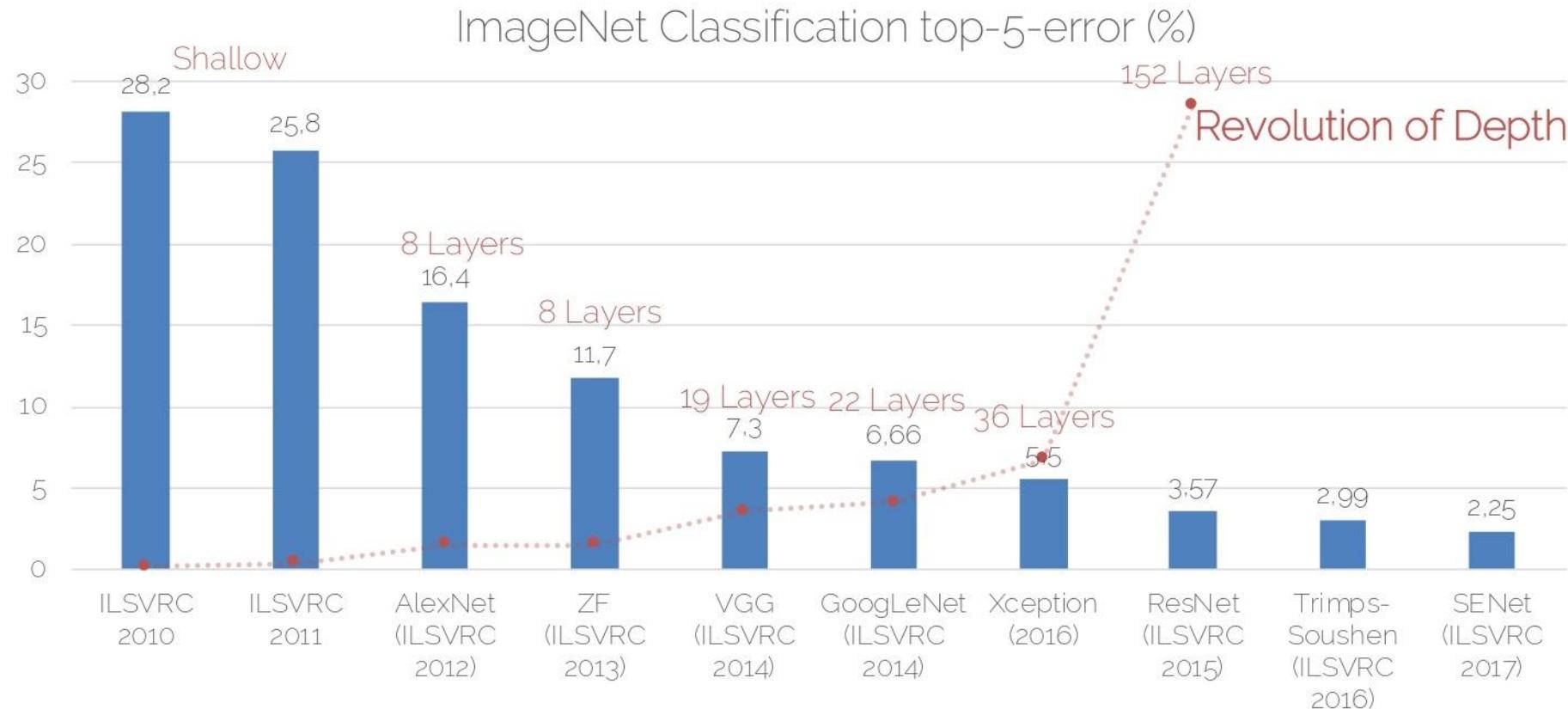
Less computations!



1x1 convolution
256 kernels of size 1x1x3

Multiplications:
 $256 \times 1 \times 1 \times 3 \times (8 \times 8 \text{ locations}) = 49152$

ImageNet Benchmark

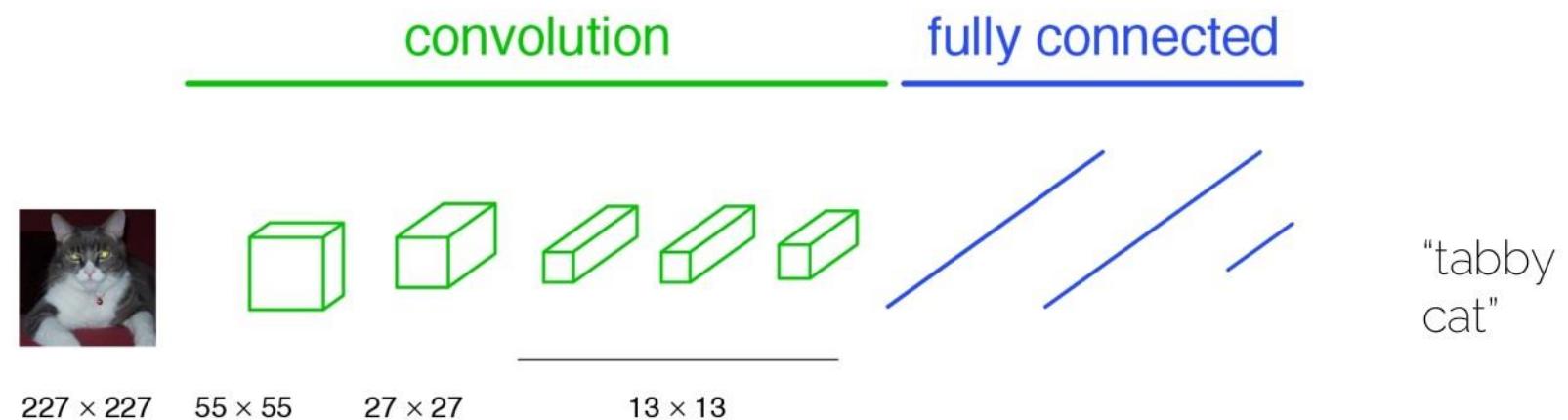




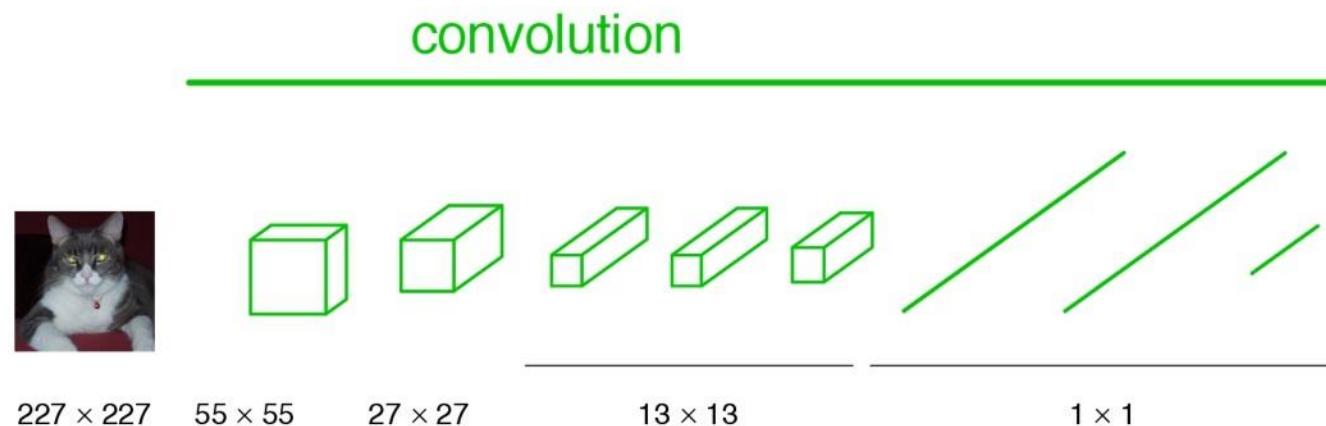
Fully Convolutional Network



Classification Network

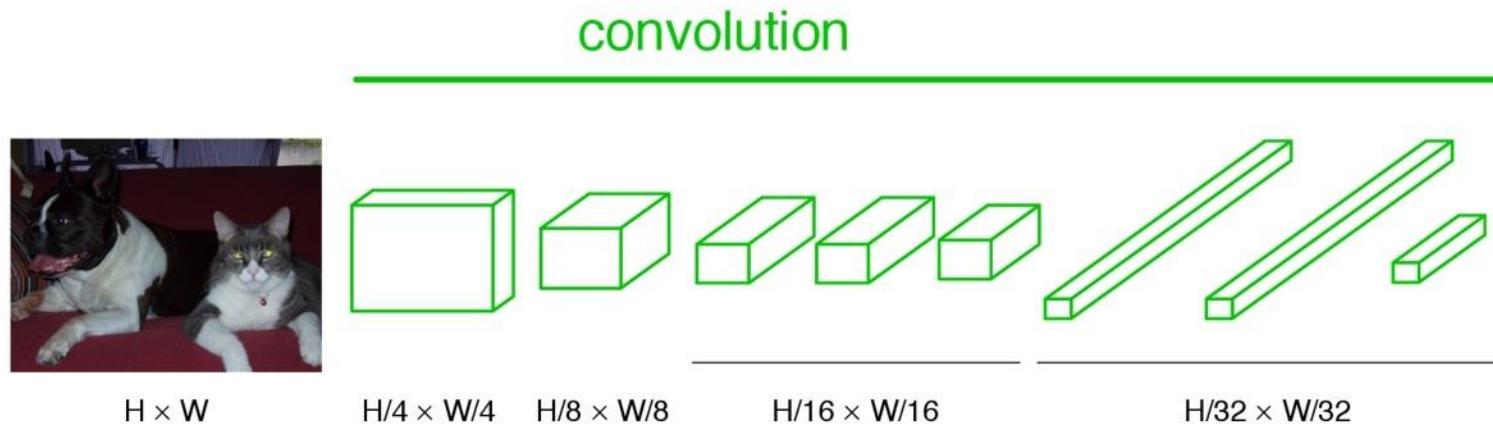


FCN: Becoming Fully Convolutional

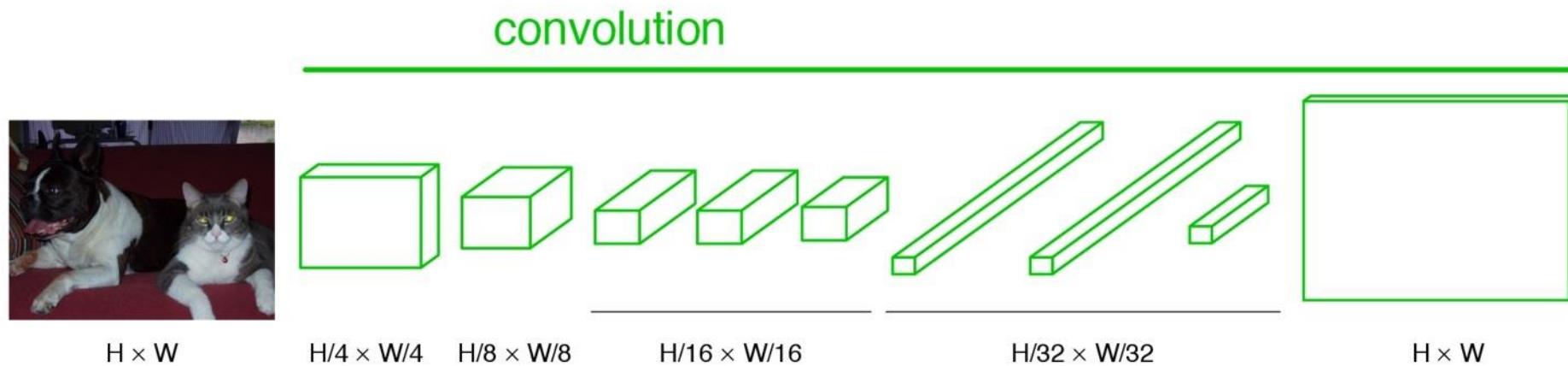


Convert fully connected layers to convolutional layers!

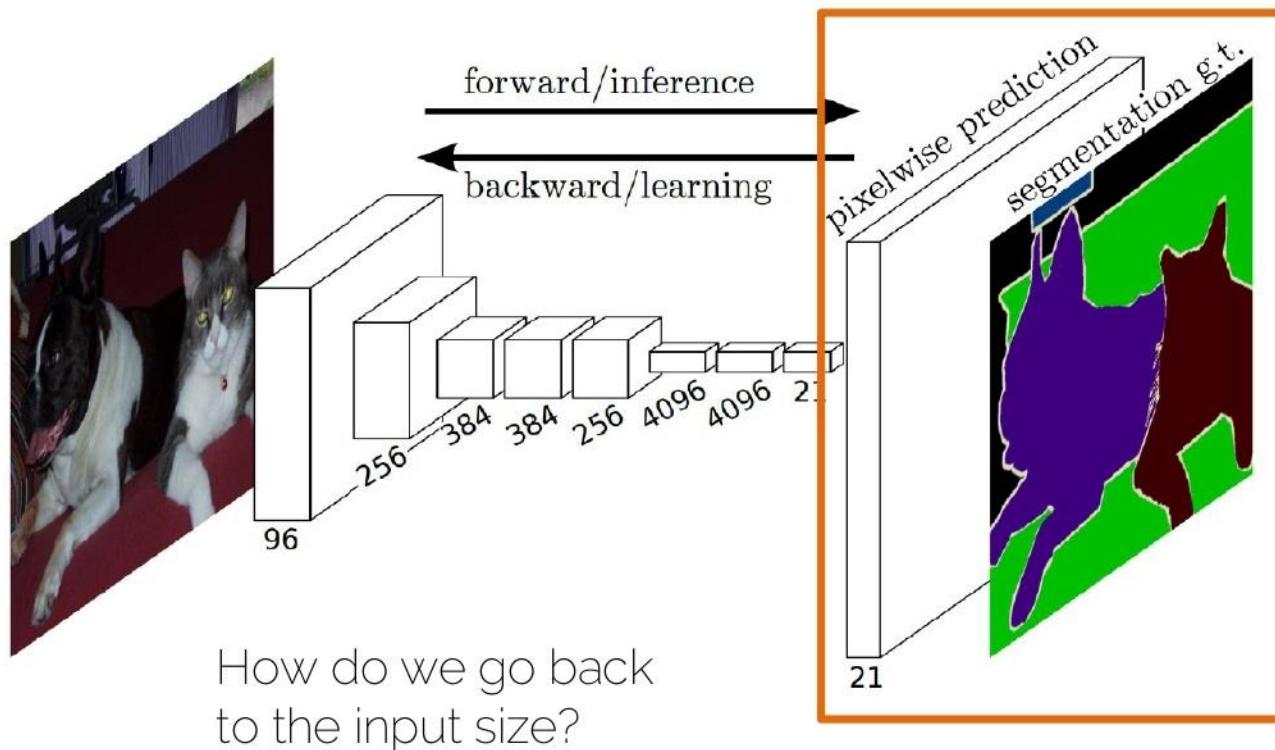
FCN: Becoming Fully Convolutional



FCN: Upsampling Output



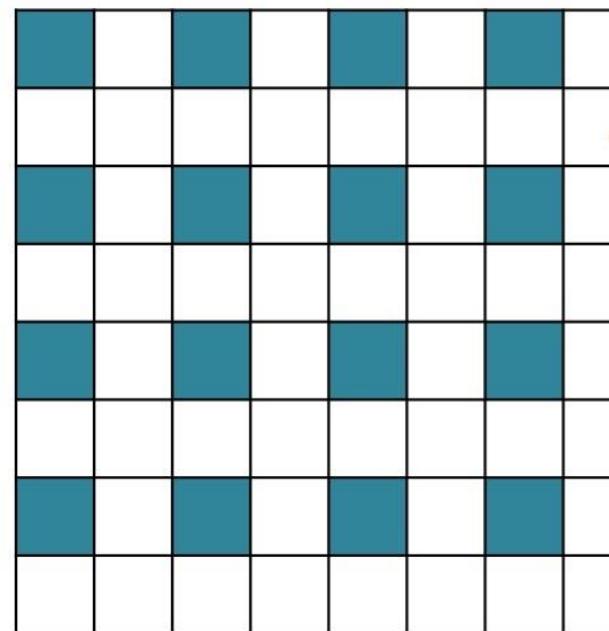
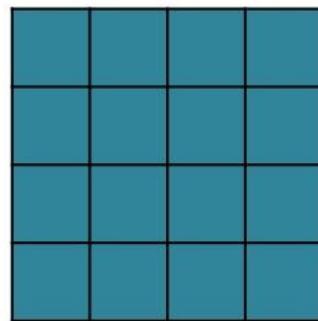
Semantic Segmentation (FCN)



[Long and Shelhamer. 15] FCN

Types of Upsampling

- 1. Interpolation



?

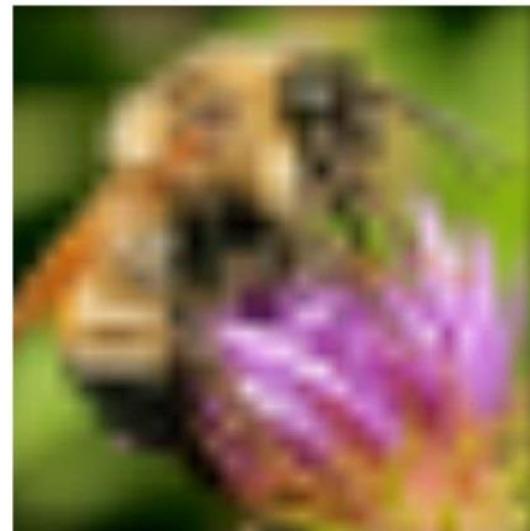
Types of Upsampling

- 1. Interpolation

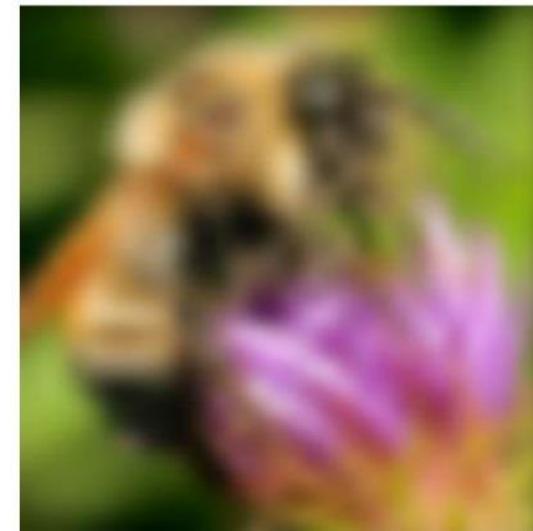
Original image  x 10



Nearest neighbor interpolation



Bilinear interpolation



Bicubic interpolation

Image: Michael Guerzhoy

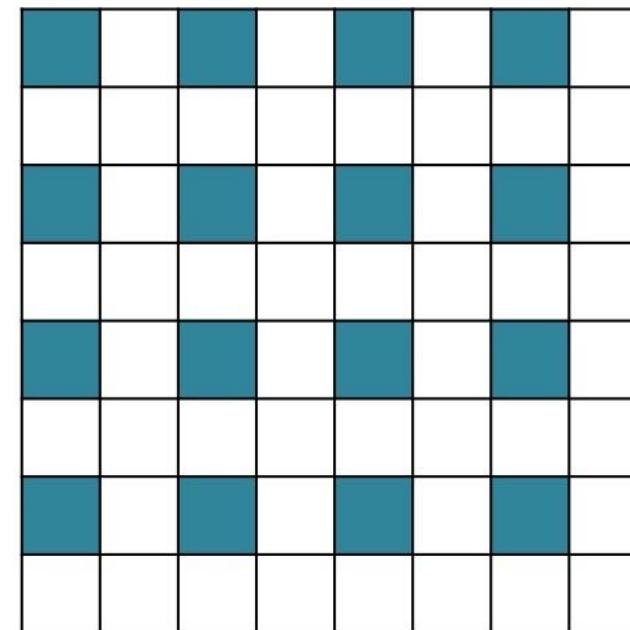
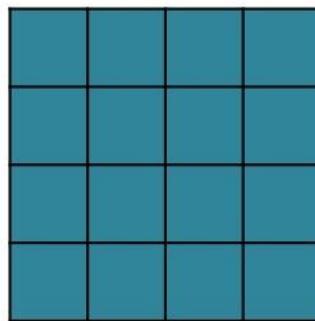
Types of Upsampling

- 1. Interpolation

Few artifacts

Types of Upsampling

- 2. Transposed conv



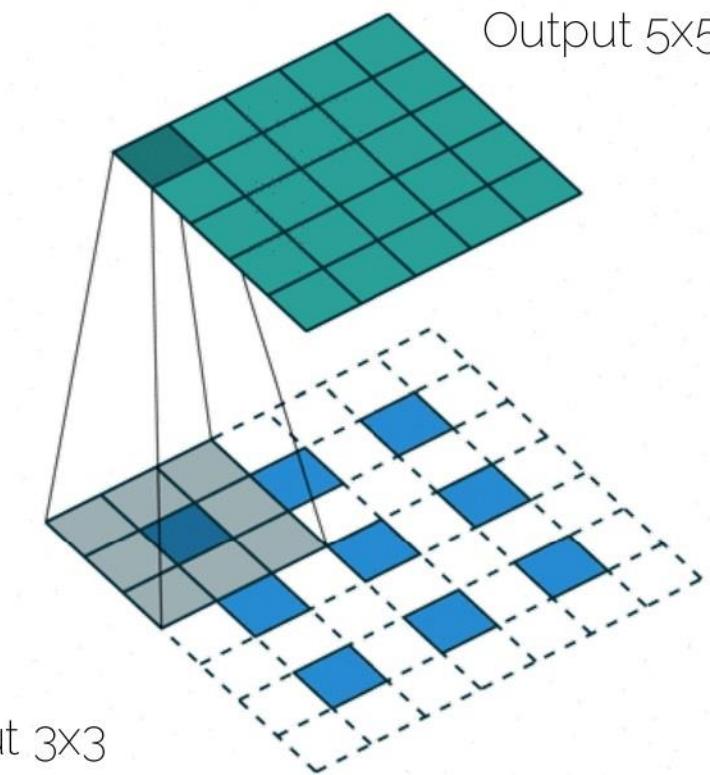
efficient

+ CONVS

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

Types of Upsampling

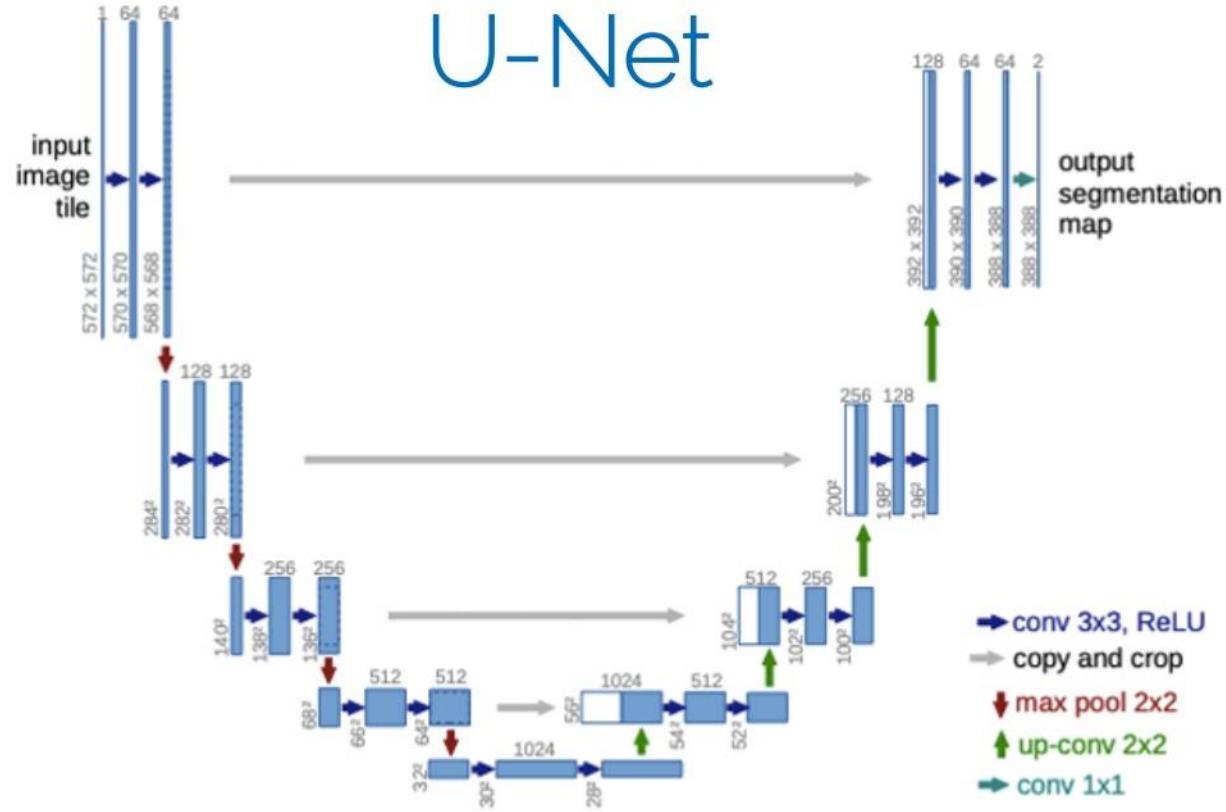
- 2. Transposed convolution
 - Unpooling
 - Convolution filter (learned)
 - Also called up-convolution
(**never** deconvolution)



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 top of the box. White boxes are the copied feature maps.

[Ronneberger et al. MICCAI'15] U-Net 96

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
 - final layer: 1x1 conv to map 64 channels to # classes
- Height, Width:  Depth: 