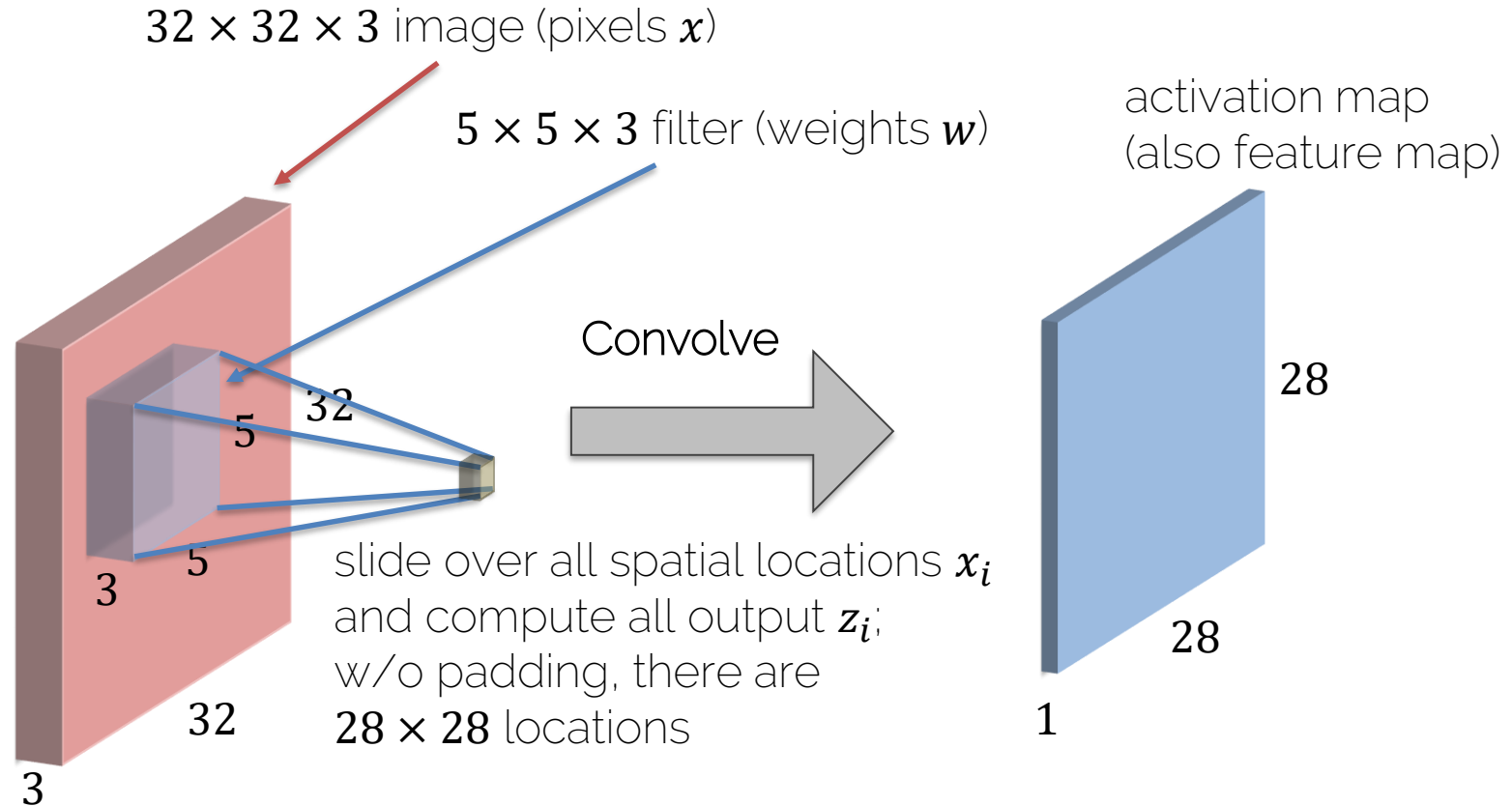
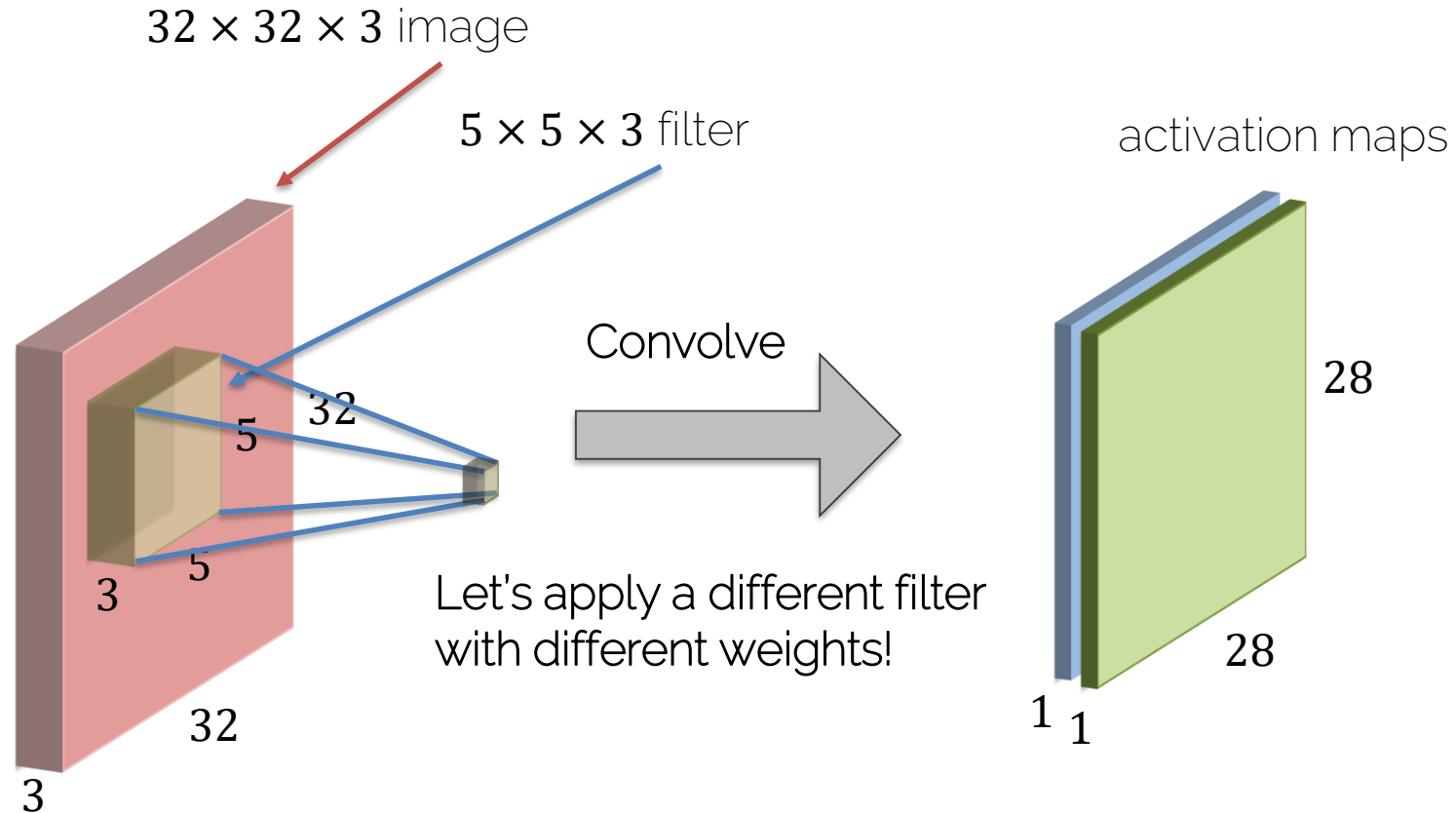


CNN Architectures

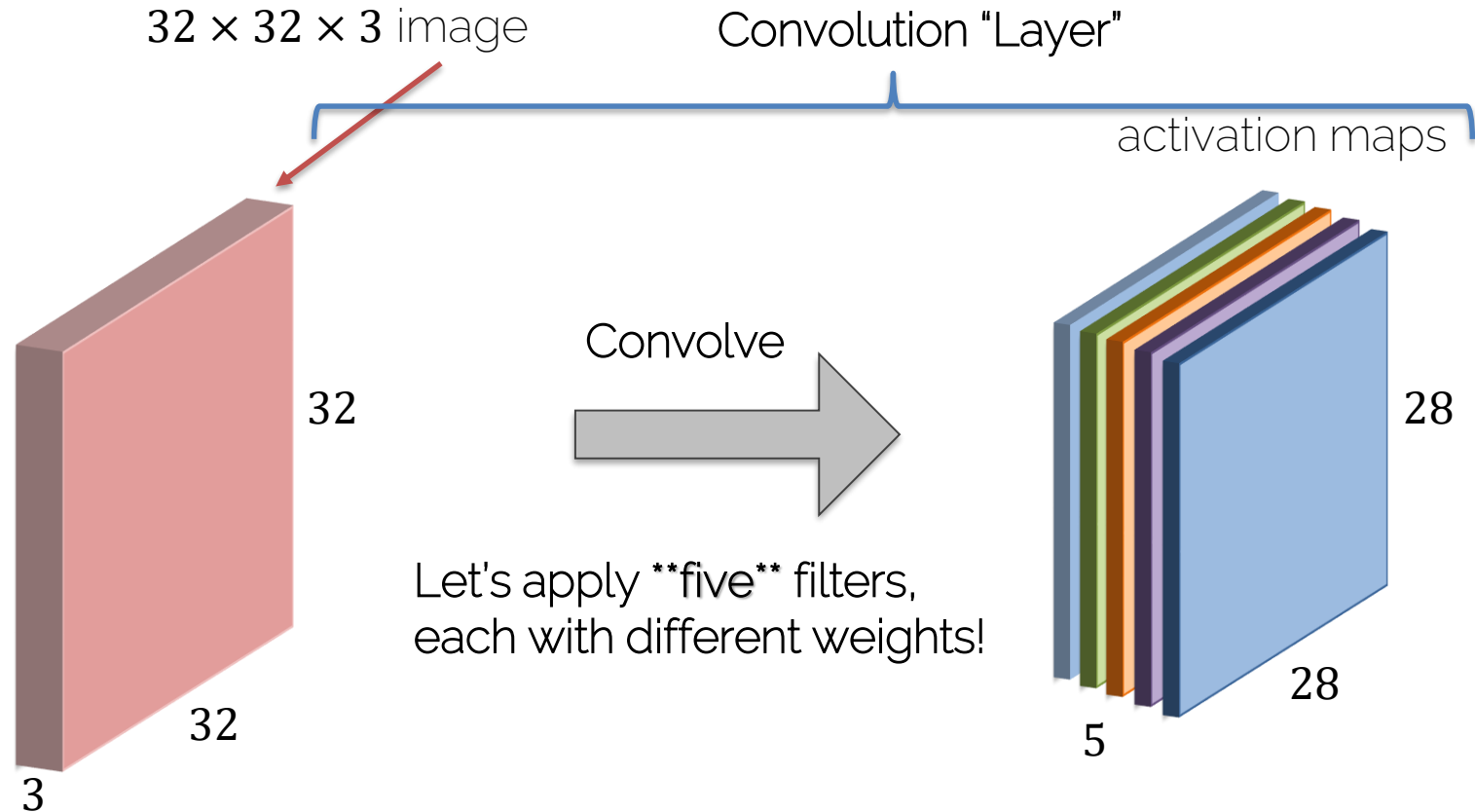
Convolutions on RGB Images



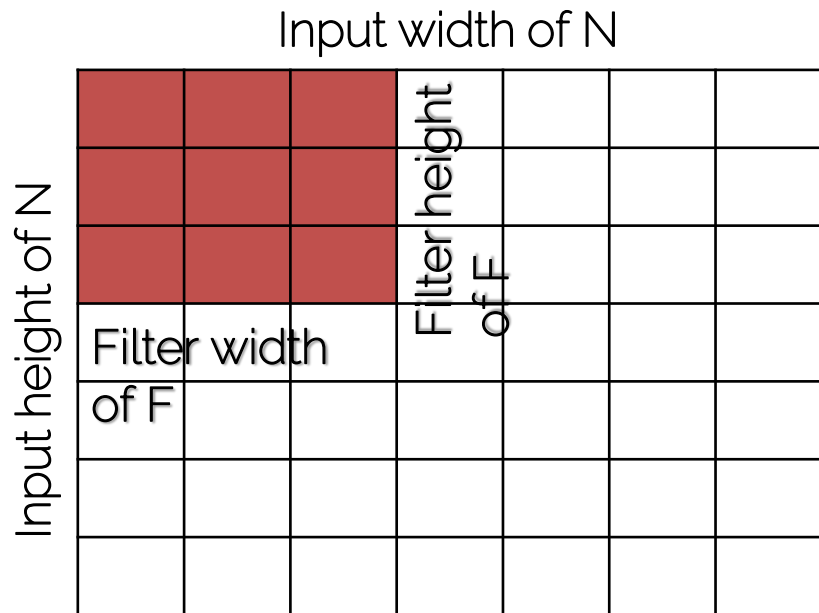
Convolution Layer



Convolution Layer



Convolution Layers: Dimensions



Input: $N \times N$

Filter: $F \times F$

Stride: S

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

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

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

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

Fractions are illegal

Convolution Layers: Padding

Image 7x7 + zero padding

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Types of convolutions:

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

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

CNN Learned Filters

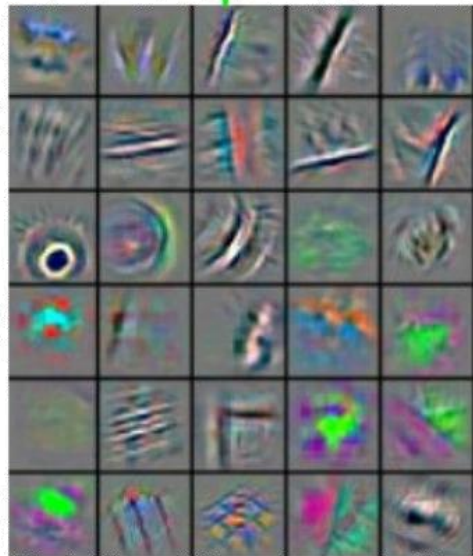


Low-Level
Feature

Mid-Level
Feature

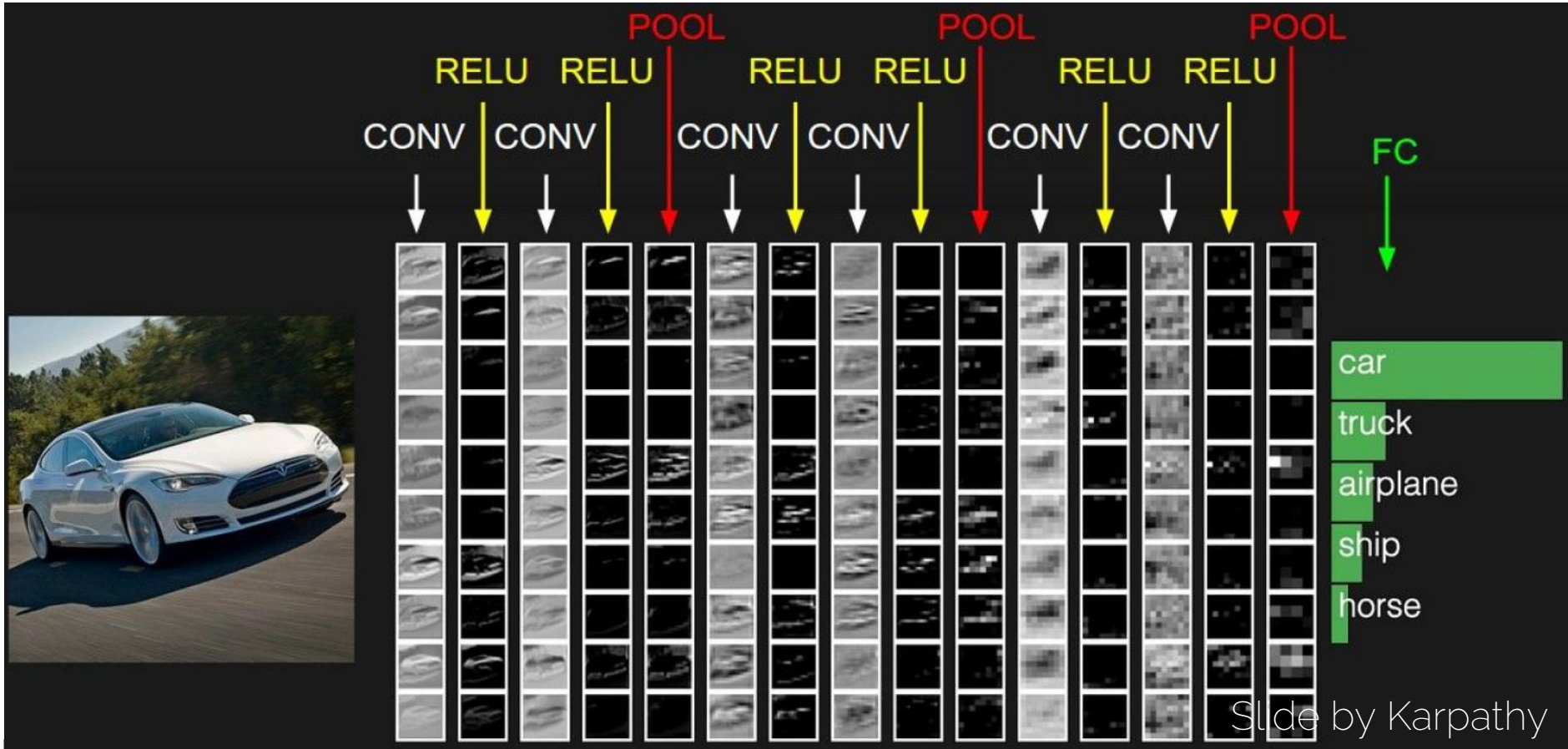
High-Level
Feature

Trainable
Classifier



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

CNN Prototype



Slide by Karpathy

Classic Architectures

LeNet

- Digit recognition: 10 classes

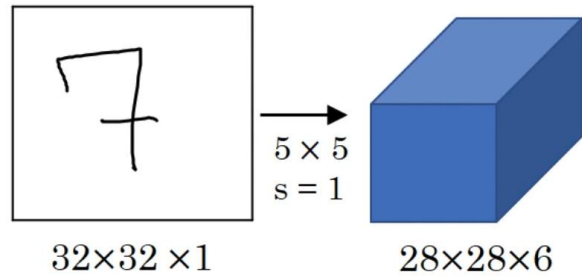


$32 \times 32 \times 1$

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

LeNet

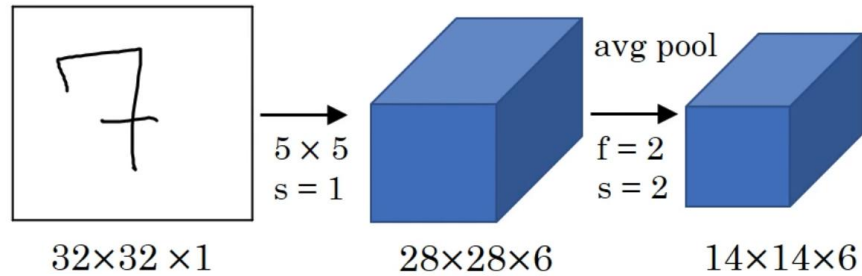
- Digit recognition: 10 classes



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

LeNet

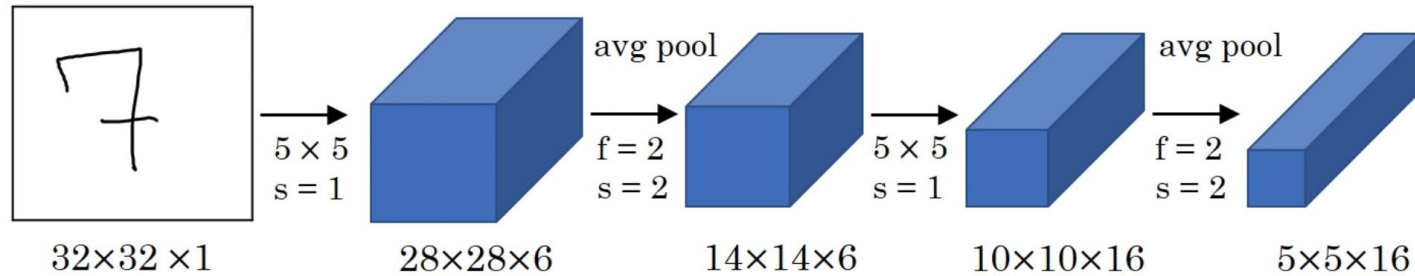
- Digit recognition: 10 classes



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

LeNet

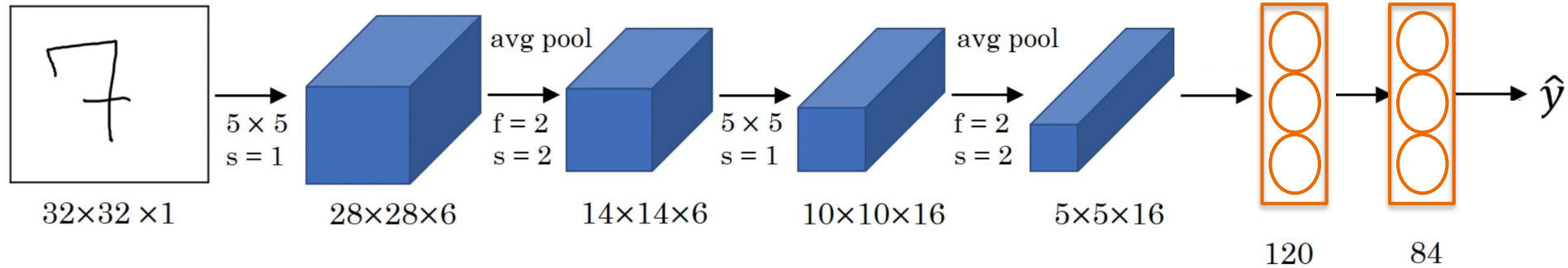
- Digit recognition: 10 classes



- Again valid convolutions, how many filters?

LeNet

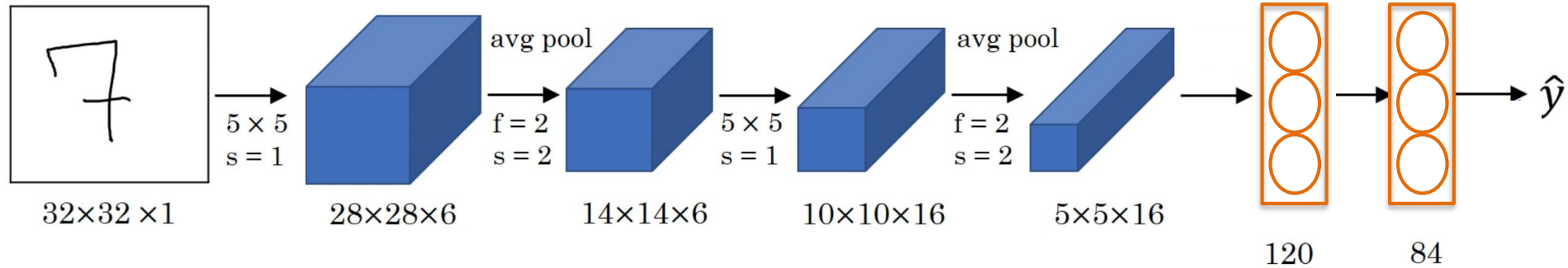
- Digit recognition: 10 classes



- Use of tanh/sigmoid activations \rightarrow not common now!

LeNet

- Digit recognition: 10 classes

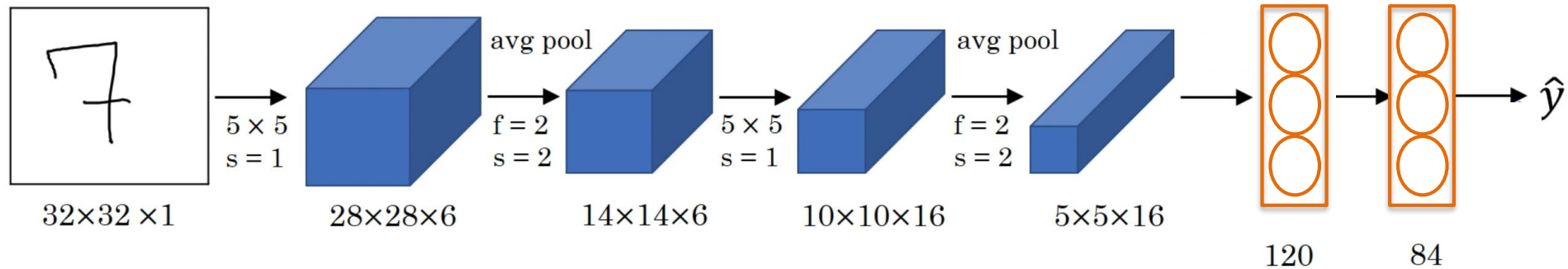


- Conv \rightarrow Pool \rightarrow Conv \rightarrow Pool \rightarrow Conv \rightarrow FC

LeNet

- Digit recognition: 10 classes

60k parameters

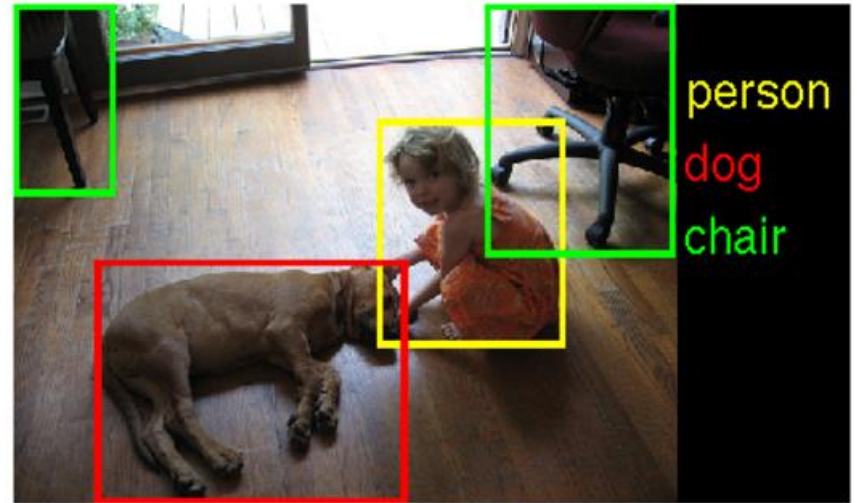
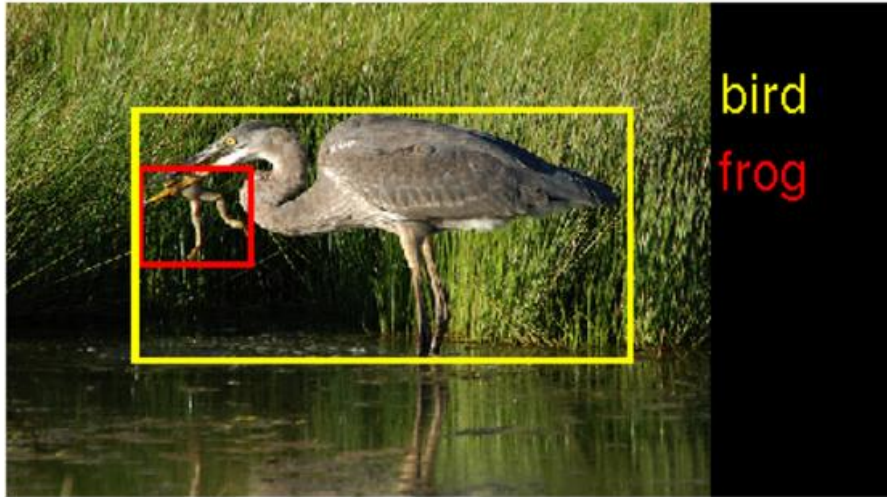


- Conv \rightarrow Pool \rightarrow Conv \rightarrow Pool \rightarrow Conv \rightarrow FC
- As we go deeper: Width, Height \downarrow Number of Filters \uparrow

Test Benchmarks

- ImageNet Dataset:

ImageNet Large Scale Visual Recognition Competition (ILSVRC)



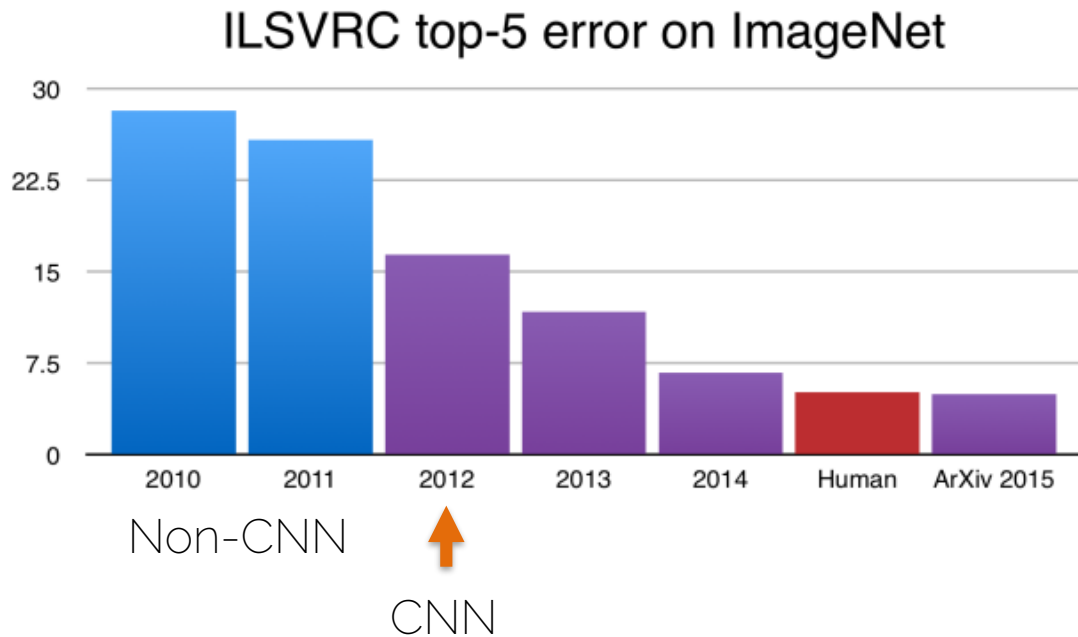
[Russakovsky et al., IJCV'15] "ImageNet Large Scale Visual Recognition Challenge."

Common Performance Metrics

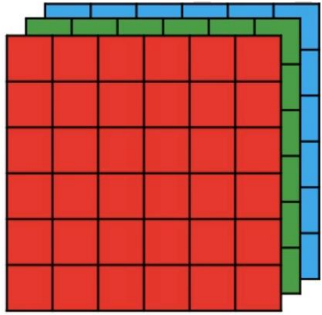
- **Top-1 score:** check if a sample's top class (i.e. the one with highest probability) is the same as its target label
- **Top-5 score:** check if your label is in your 5 first predictions (i.e. predictions with 5 highest probabilities)
- → **Top-5 error:** percentage of test samples for which the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

AlexNet

- Cut ImageNet error down in half



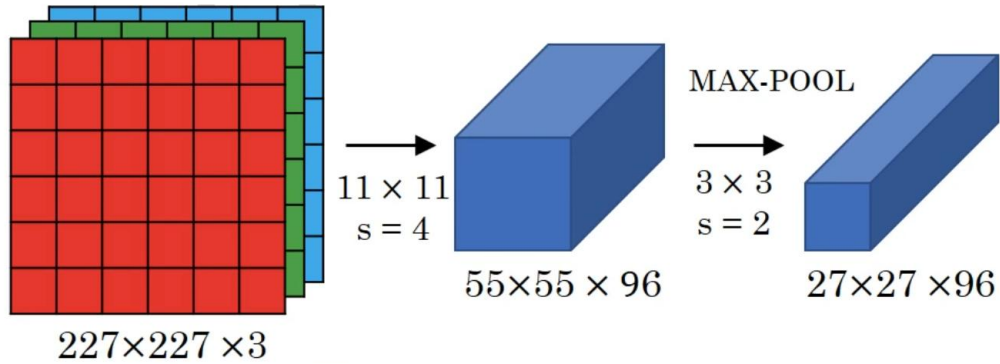
AlexNet



$227 \times 227 \times 3$

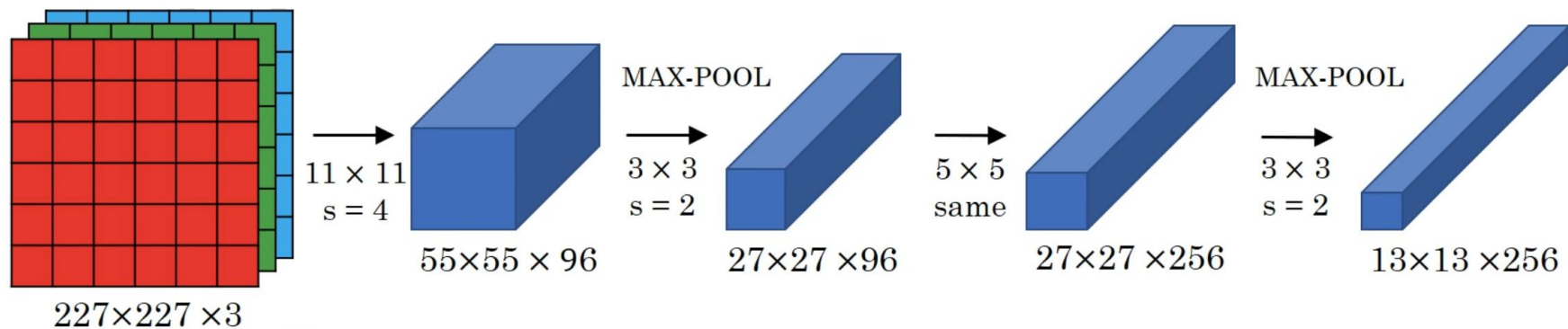
[Krizhevsky et al. NIPS'12] AlexNet

AlexNet



[Krizhevsky et al. NIPS'12] AlexNet

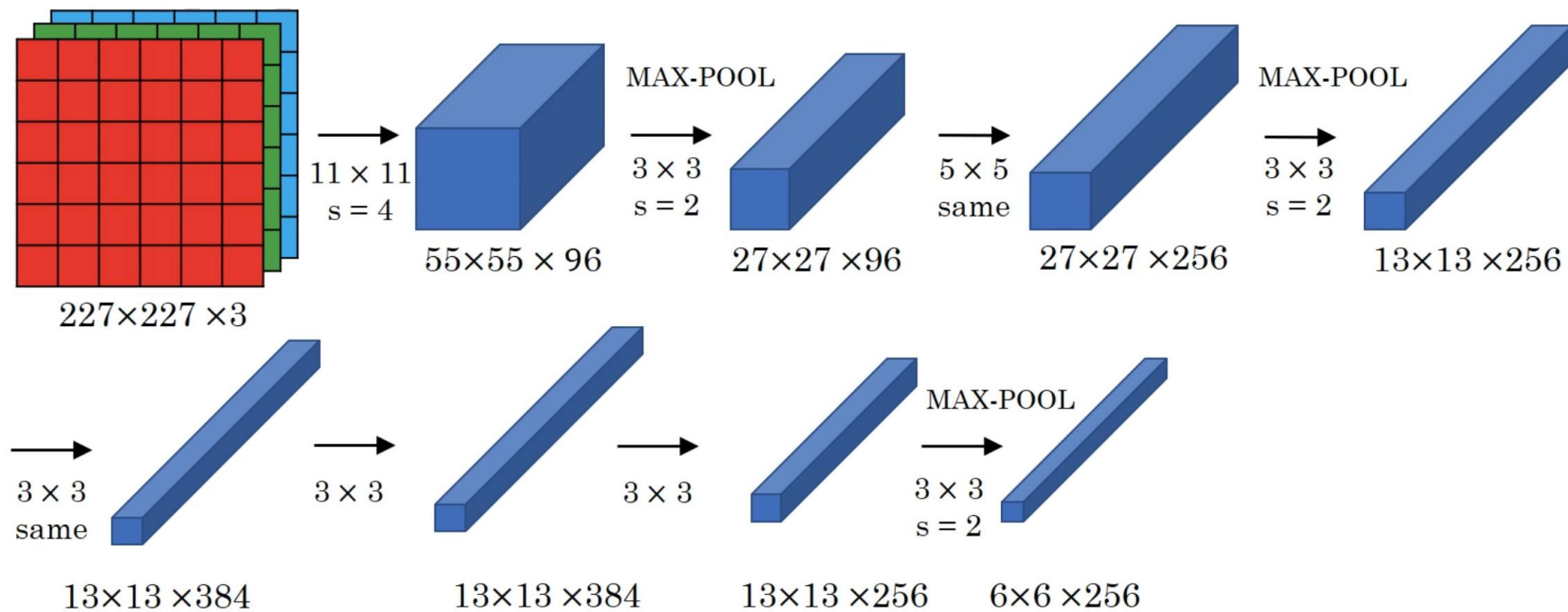
AlexNet



- Use of same convolutions
- As with LeNet: Width, Height \downarrow Number of Filters \uparrow

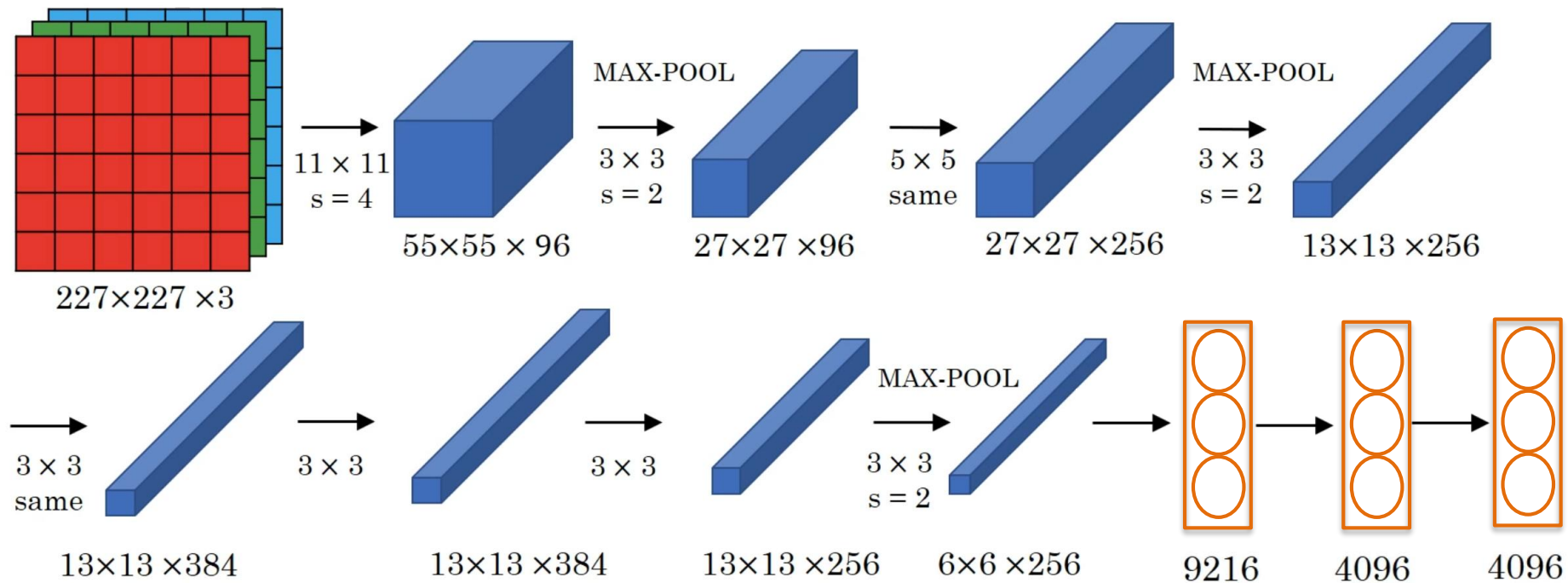
[Krizhevsky et al. NIPS'12] AlexNet

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

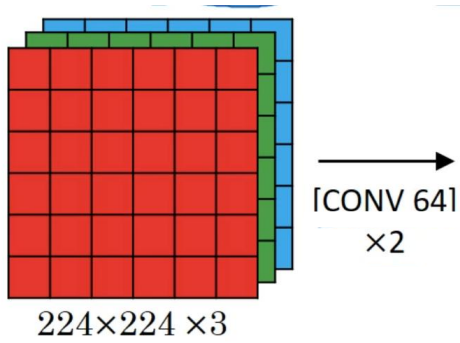
60M parameters

VGGNet

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

VGGNet

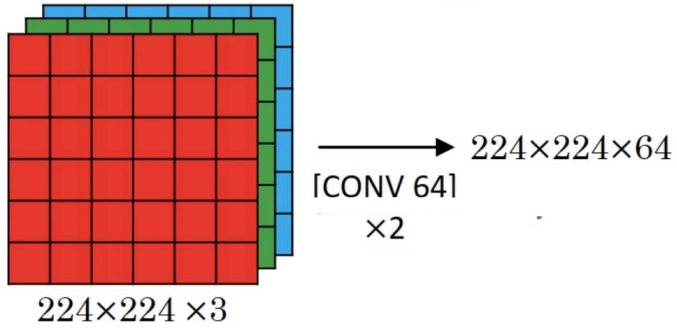
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

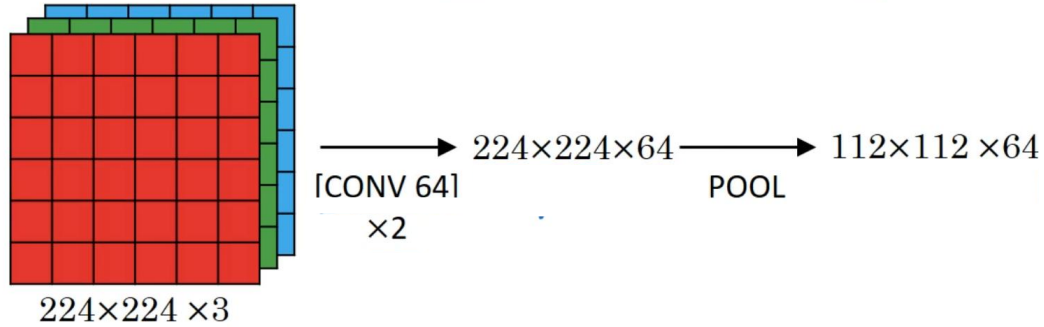
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

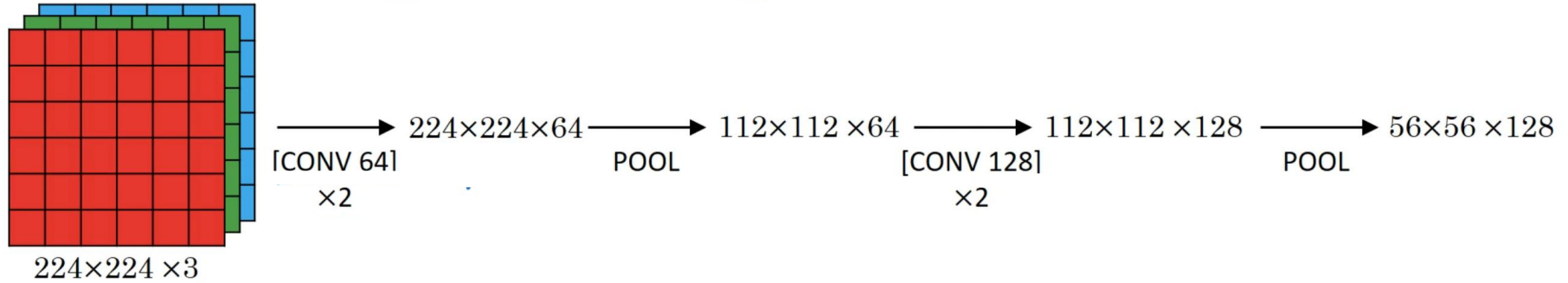
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

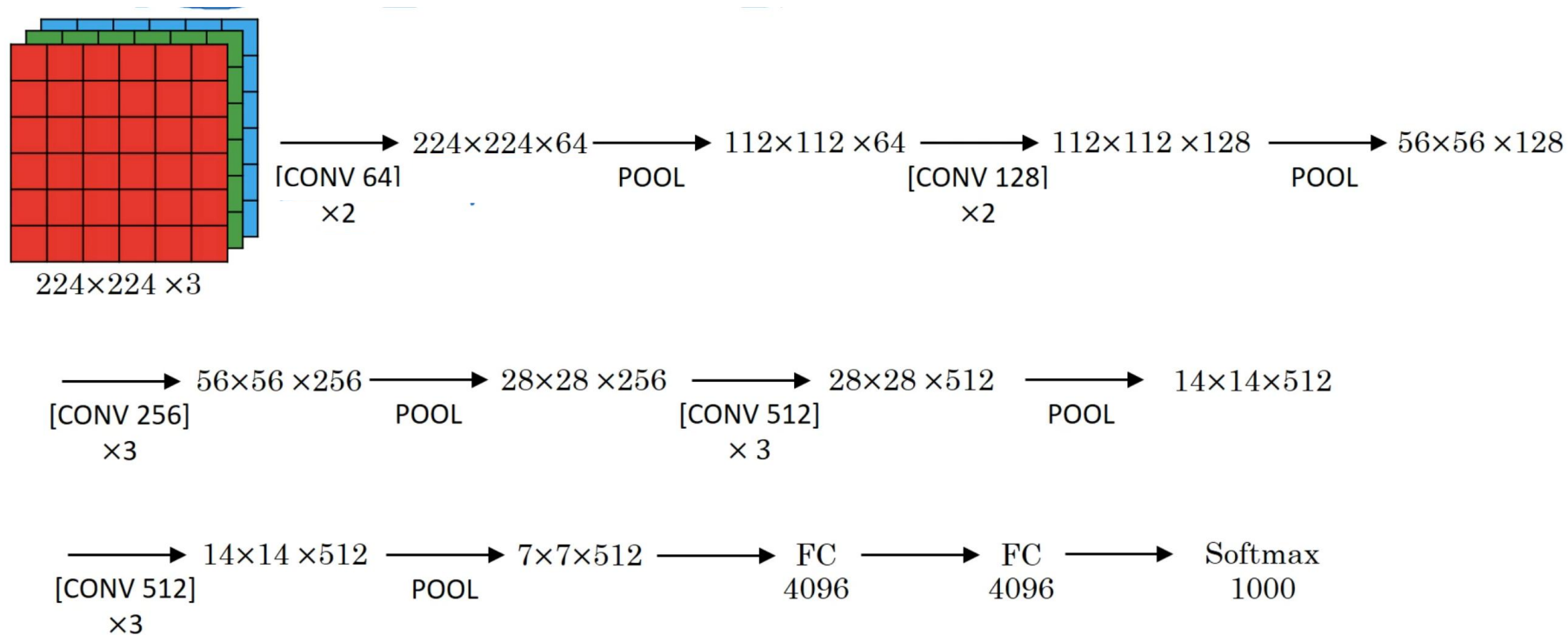
Conv=3x3,s=1,same
Maxpool=2x2,s=2



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

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



[Simonyan and Zisserman ICLR'15] VGGNet

VGGNet

- Conv -> 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 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					

[Simonyan and Zisserman 2014]

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

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

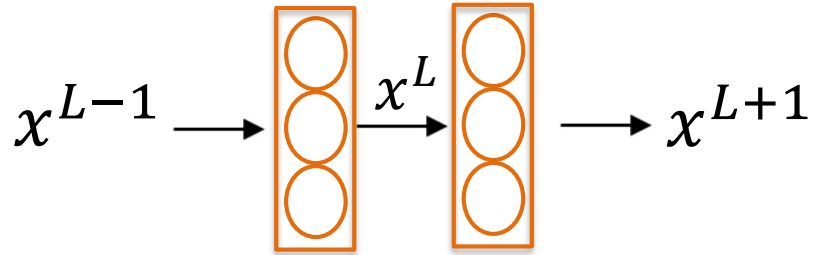
Skip Connections

The Problem of Depth

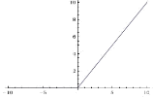
- As we add more and more layers, training becomes harder
- Vanishing and exploding gradients
- How can we train very deep nets?

Residual Block

- Two layers



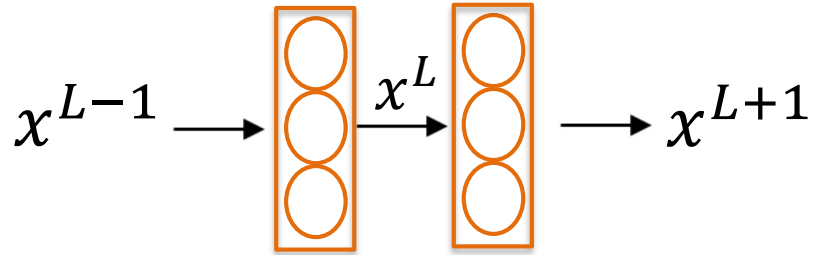
Input \longrightarrow $W^L x^{L-1} + b^L$ \longrightarrow $x^L = f(W^L x^{L-1} + b^L)$ \longrightarrow

Linear  Non-linearity

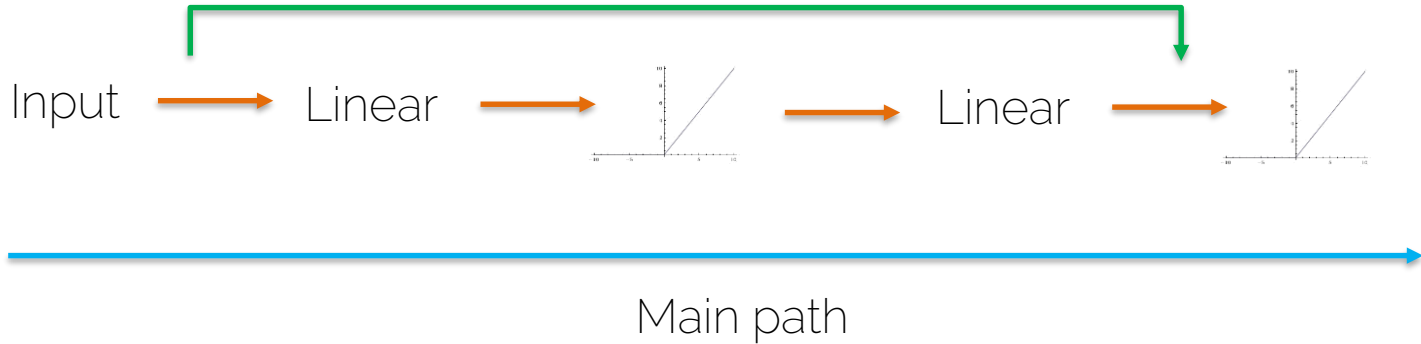
$$\longrightarrow x^{L+1} = f(W^{L+1} x^L + b^{L+1})$$

Residual Block

- Two layers

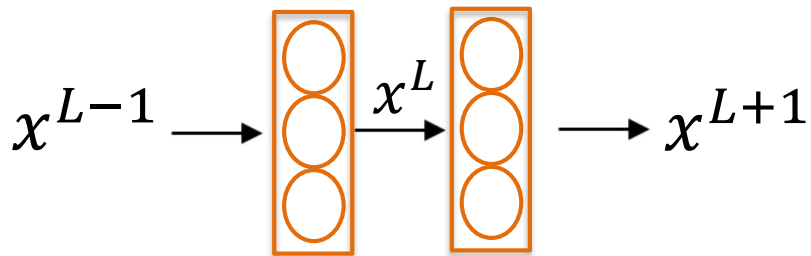


Skip connection

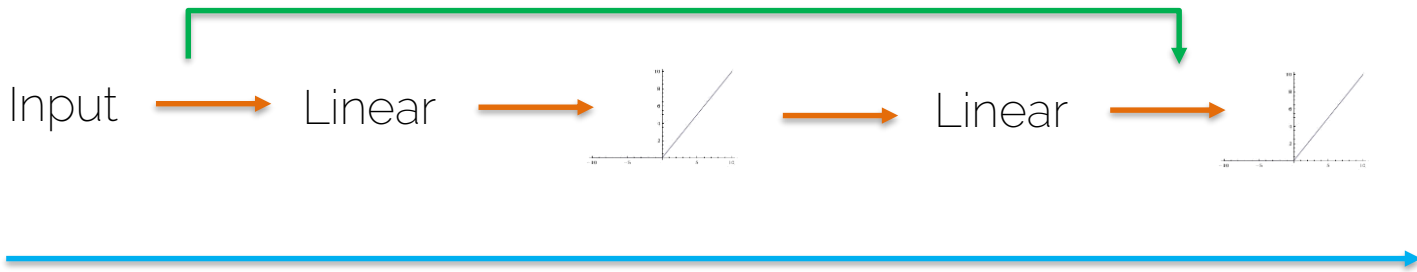


Residual Block

- Two layers



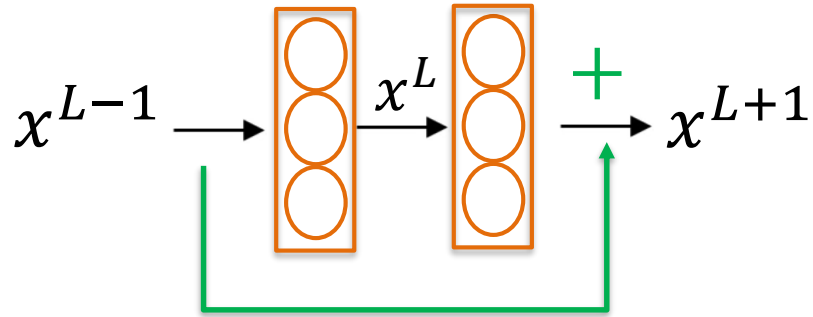
$$x^{L+1} = f(W^{L+1}x^L + b^{L+1} + x^{L-1})$$



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

Residual Block

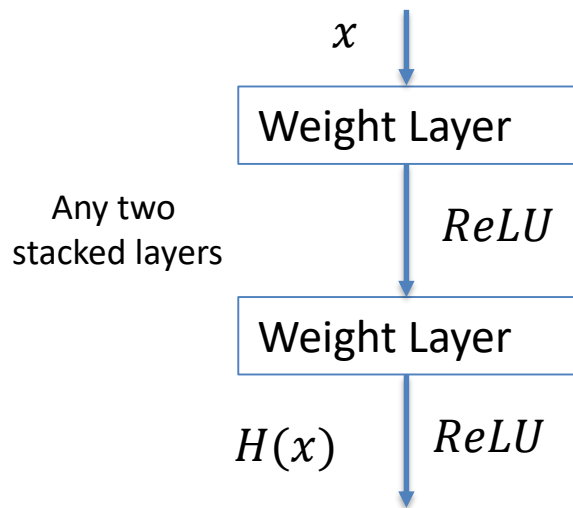
- Two layers



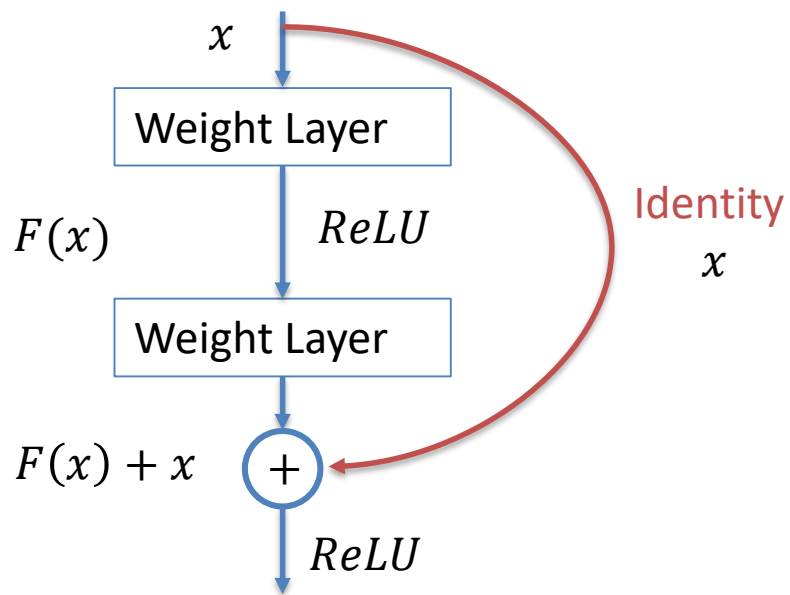
- Usually use a same convolution since we need same dimensions
- Otherwise we need to convert the dimensions with a matrix of learned weights or zero padding

ResNet Block

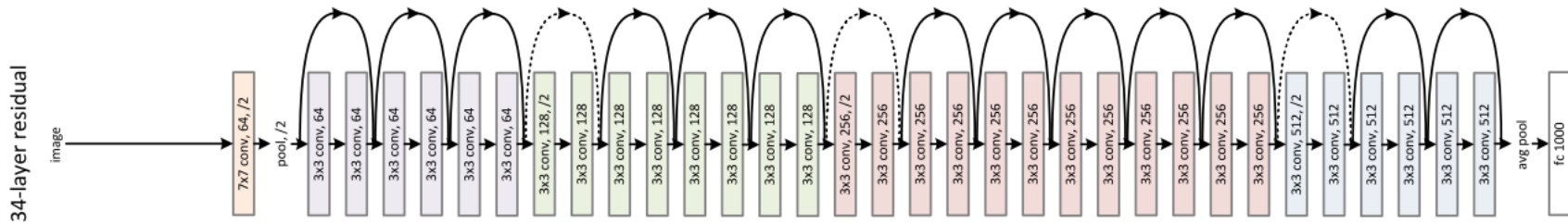
Plain Net



Residual Net



ResNet

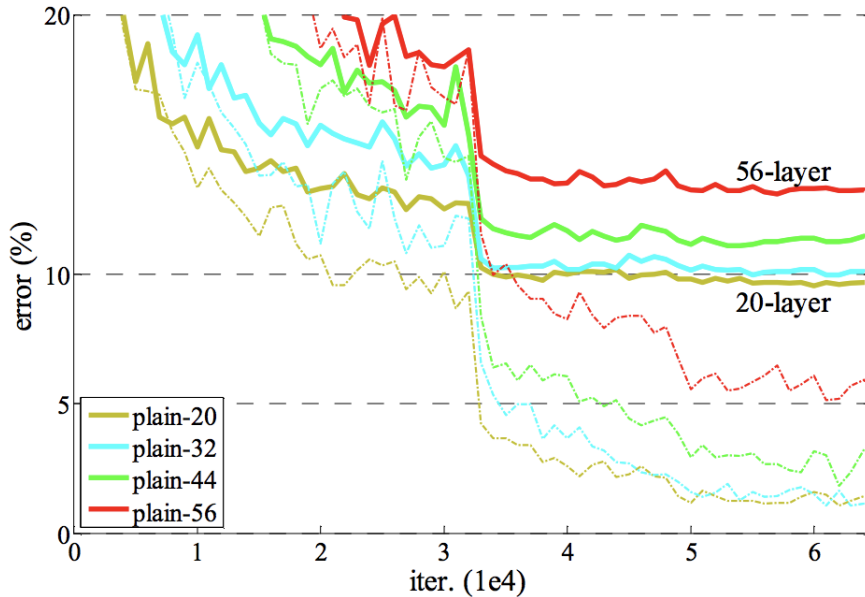


- Xavier/2 initialization
- SGD + Momentum (0.9)
- Learning rate 0.1, divided by 10 when plateau
- Mini-batch size 256
- Weight decay of $1e-5$
- No dropout

ResNet-152:
60M parameters

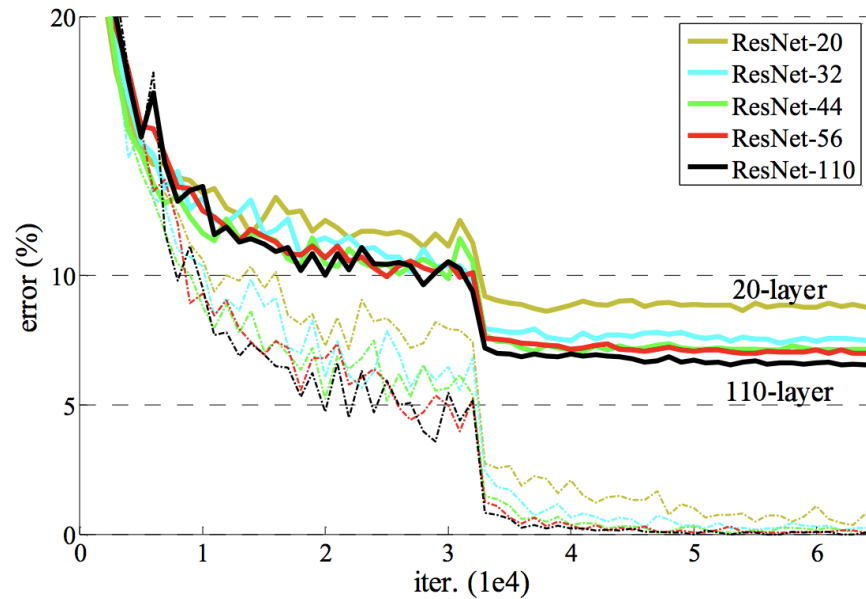
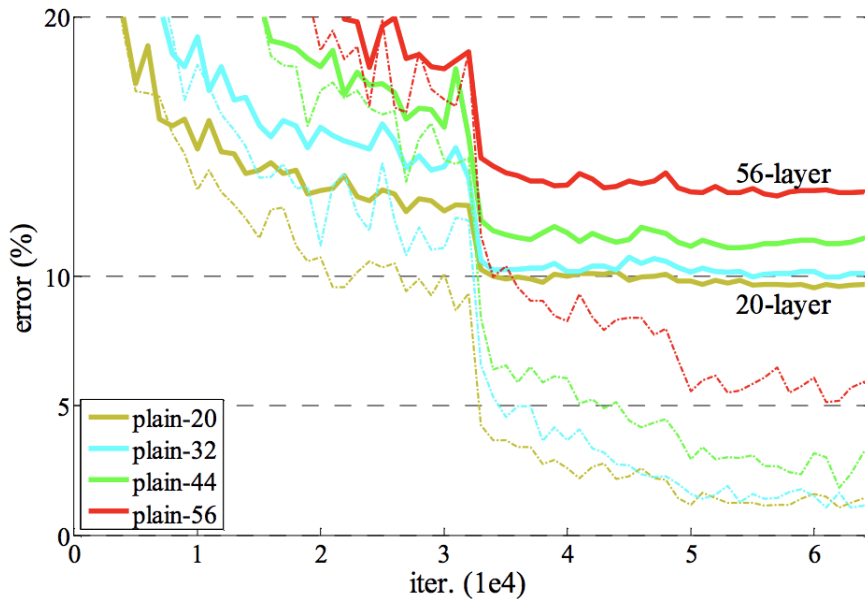
ResNet

- Without ResNet, if we make the network deeper, at some point performance starts to degrade:

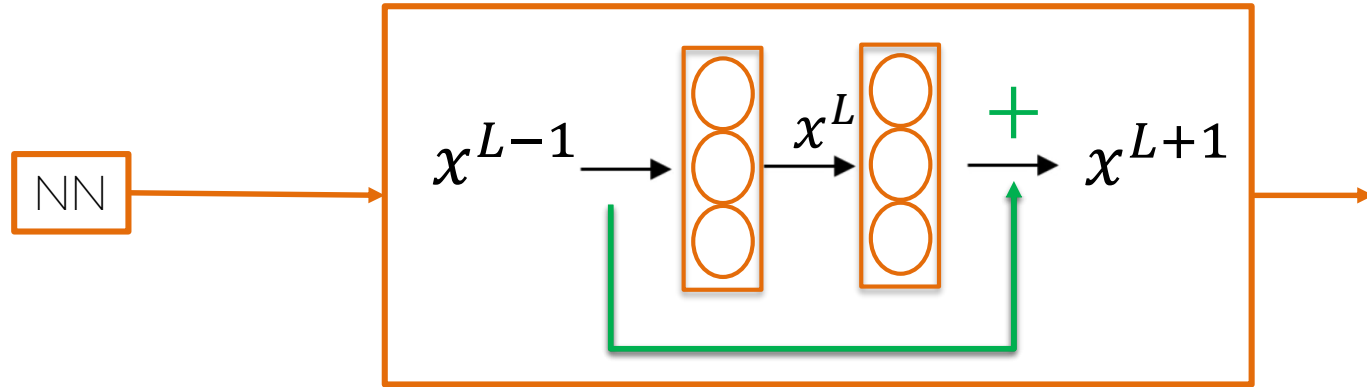


ResNet

- With Residual Blocks, performance gets better with deeper network:

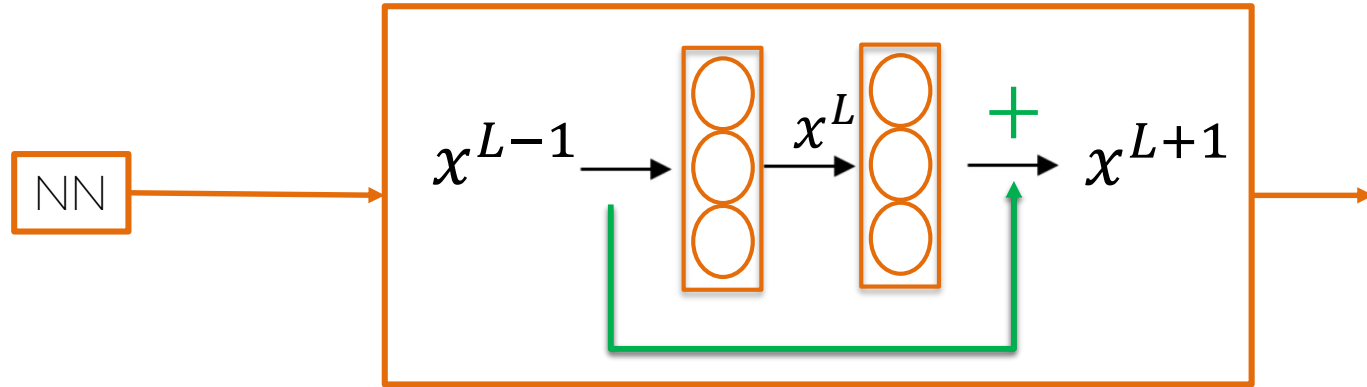


Why do ResNets Work?



- How is this block really affecting me?

Why do ResNets Work?



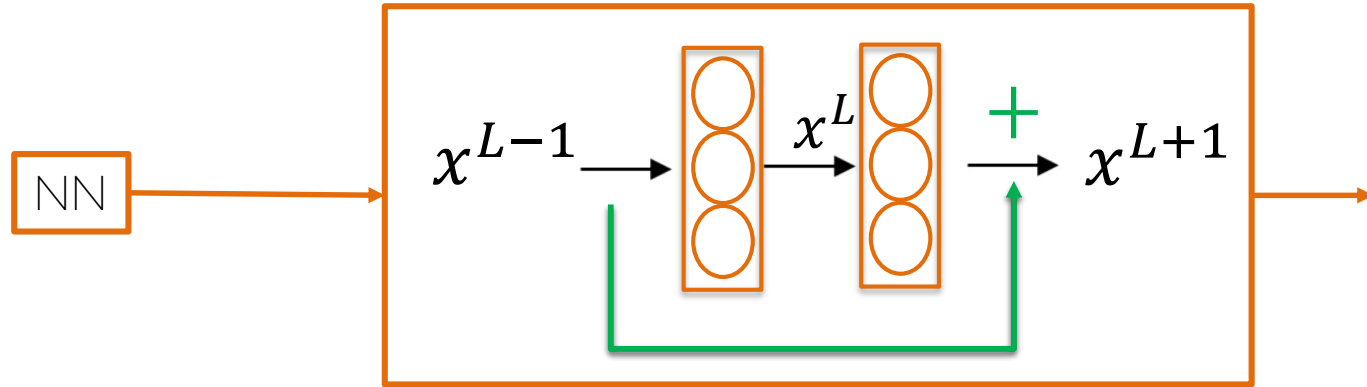
$$x^{L+1} = f(W^{L+1}x^L + b^{L+1} + x^{L-1})$$

\sim zero

\sim zero

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

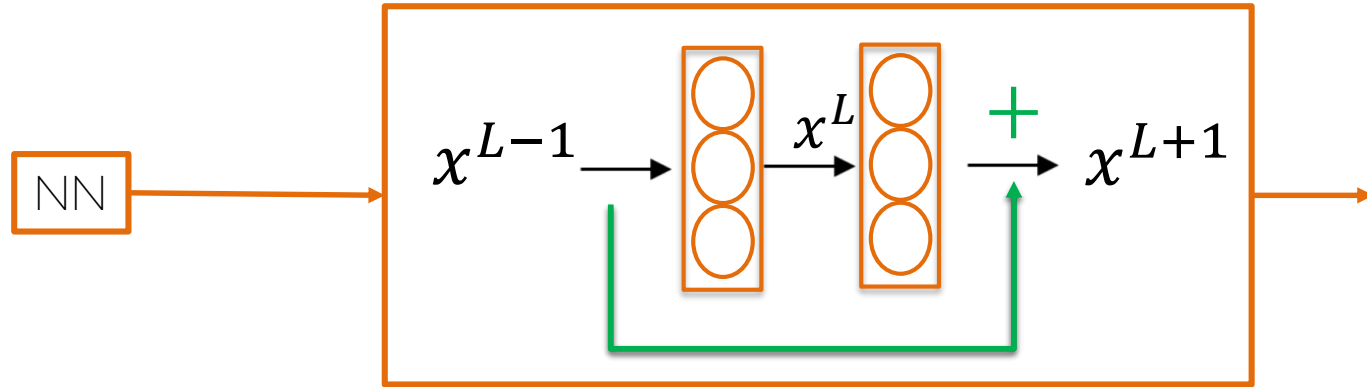
Why do ResNets Work?



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

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

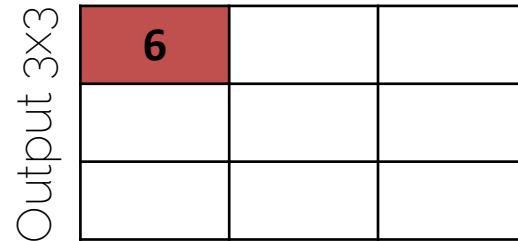
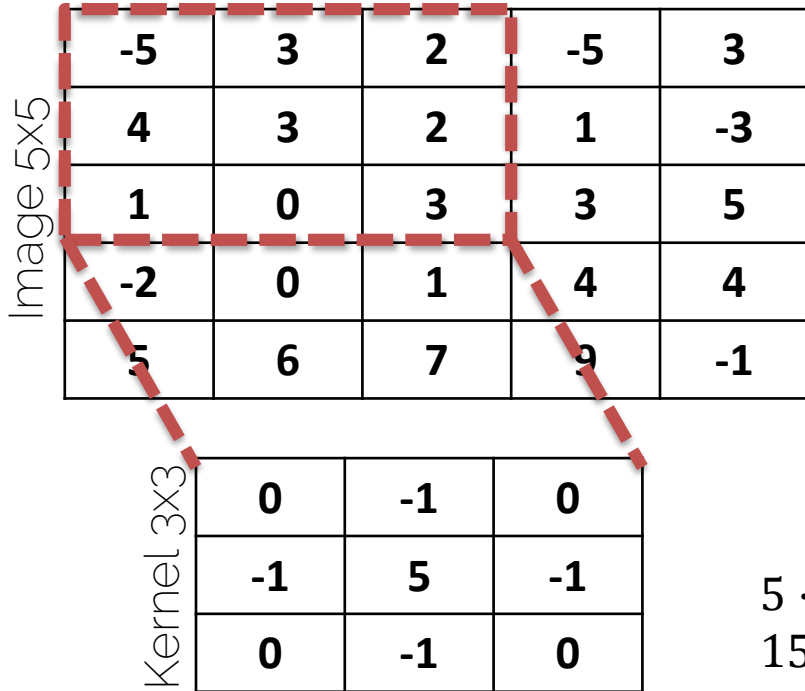
Why do ResNets Work?



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

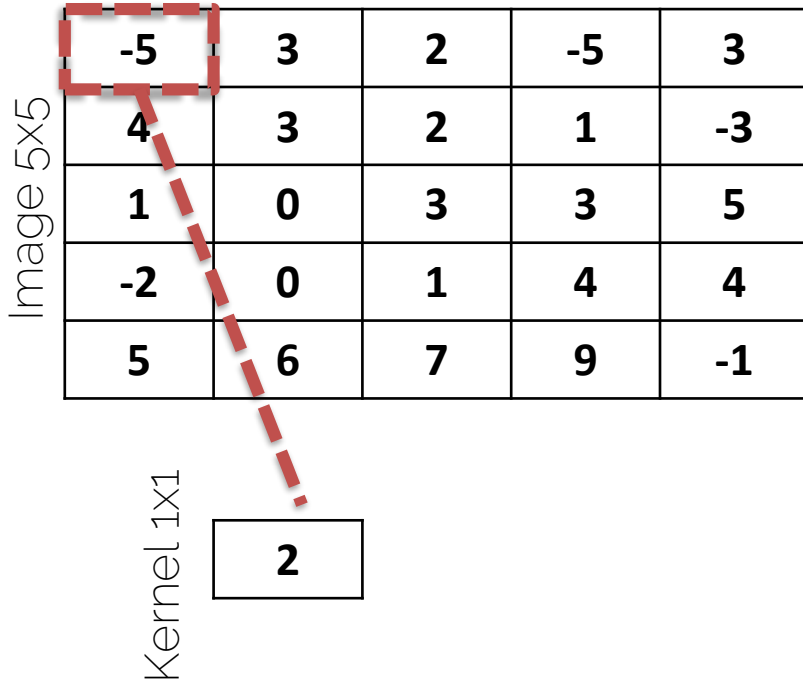
1x1 Convolutions

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

-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

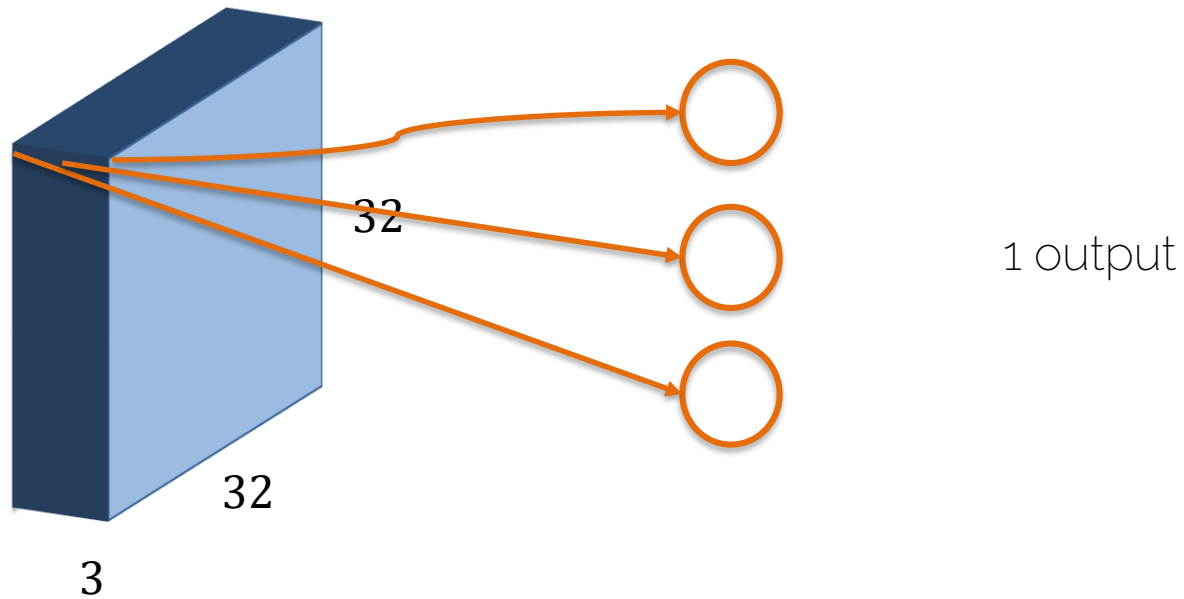
Image 5x5

-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

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

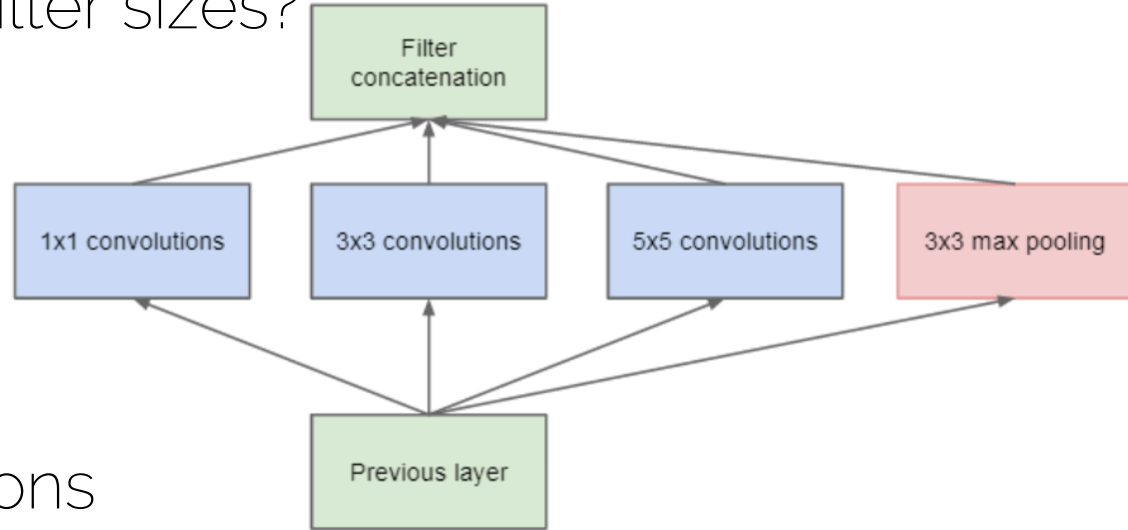


Inception Layer

Inception Layer

- Tired of choosing filter sizes?

- Use them all!

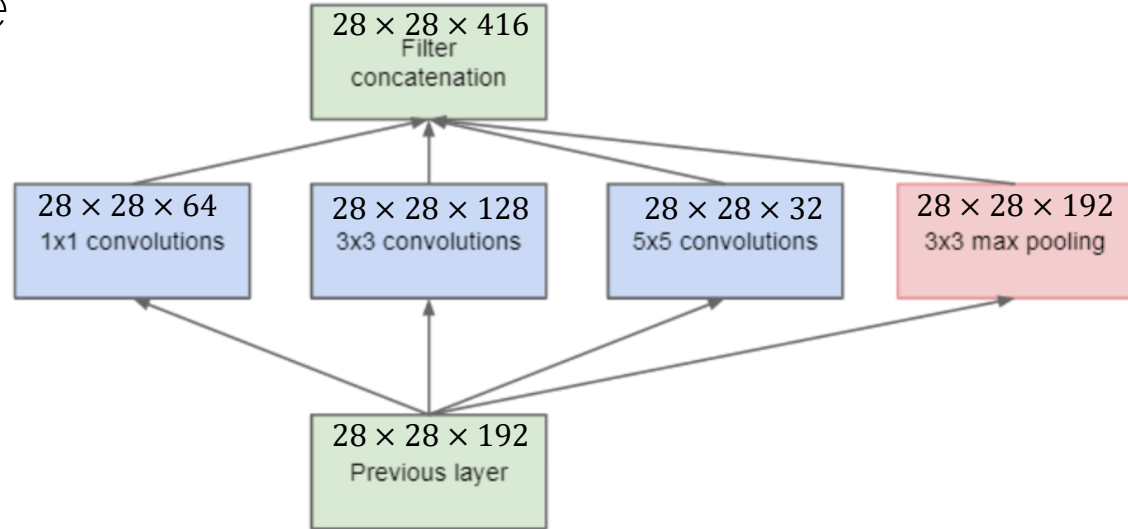


- All same convolutions

- 3x3 max pooling is with stride 1

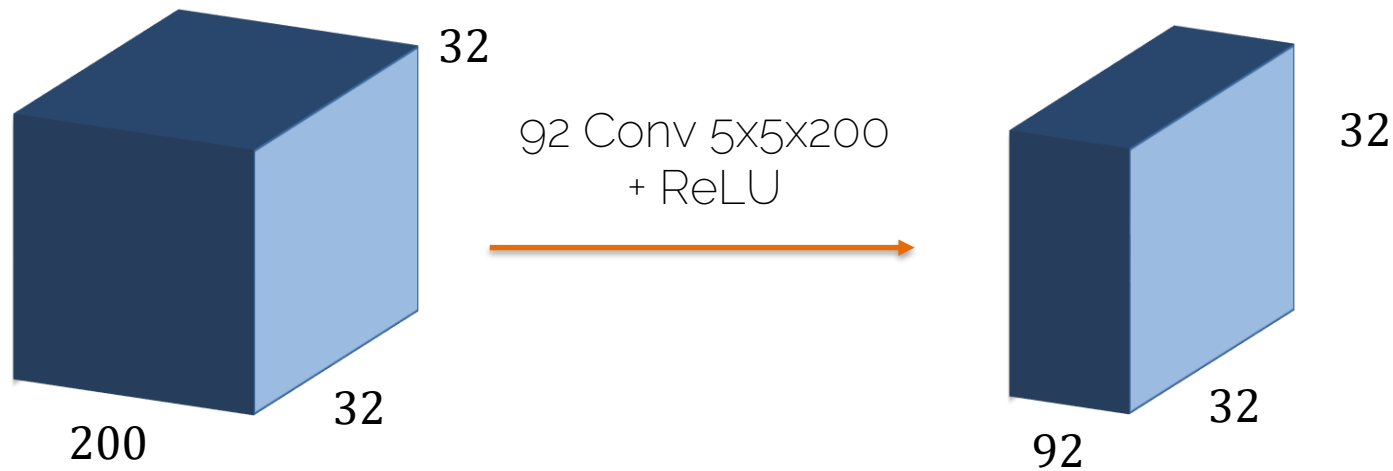
Inception Layer

- Possible size of the output



- Not sustainable!

Inception Layer: Computational Cost

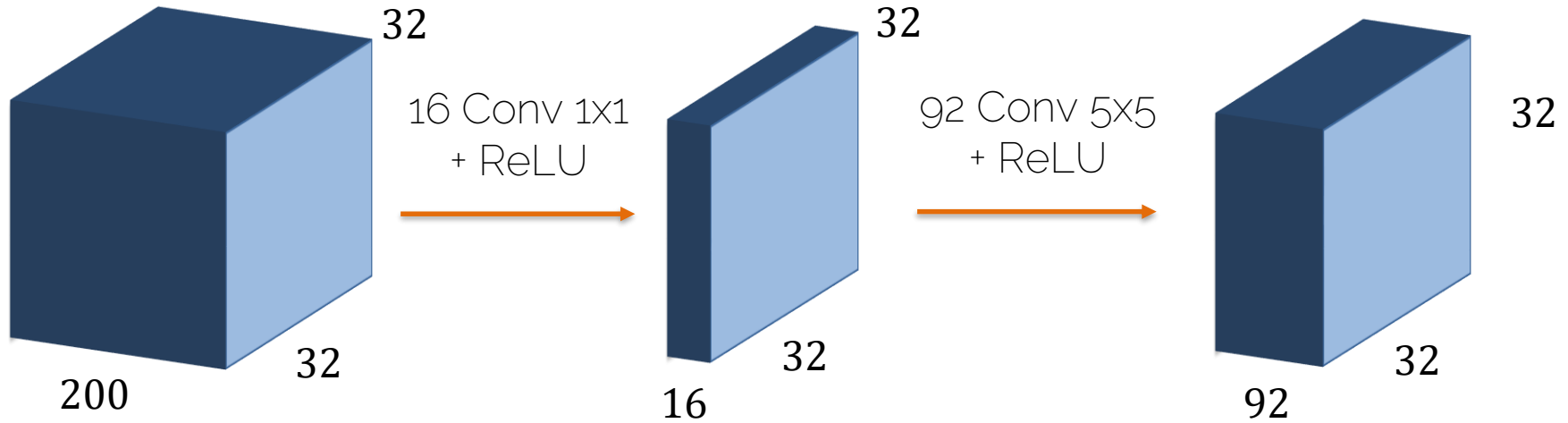


Multiplications: $5 \times 5 \times 200 \times 32 \times 32 \times 92 \sim 470$ million



1 value of the output volume

Inception Layer: Computational Cost



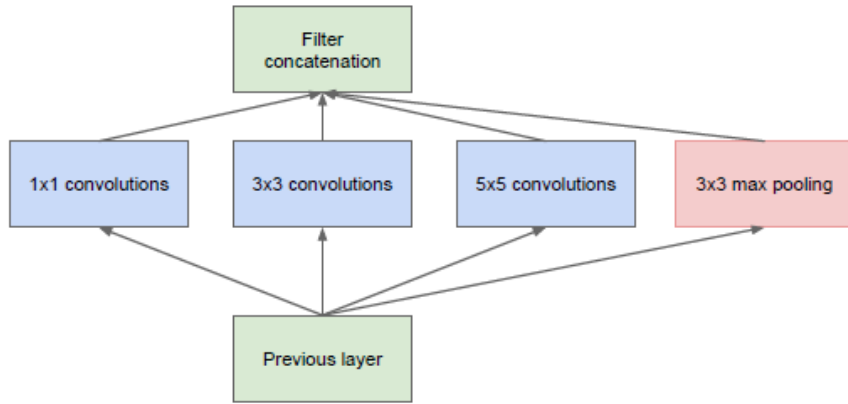
Multiplications: $1 \times 1 \times 200 \times 32 \times 32 \times 16$

$5 \times 5 \times 16 \times 32 \times 32 \times 92$

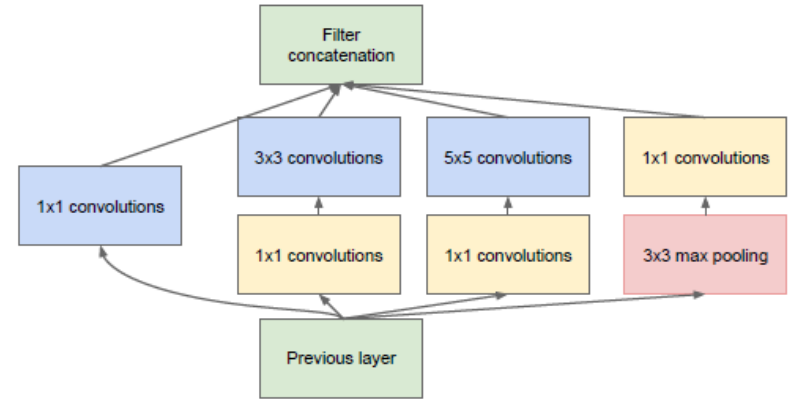
~ 40 million

Reduction of multiplications by 1/10

Inception Layer

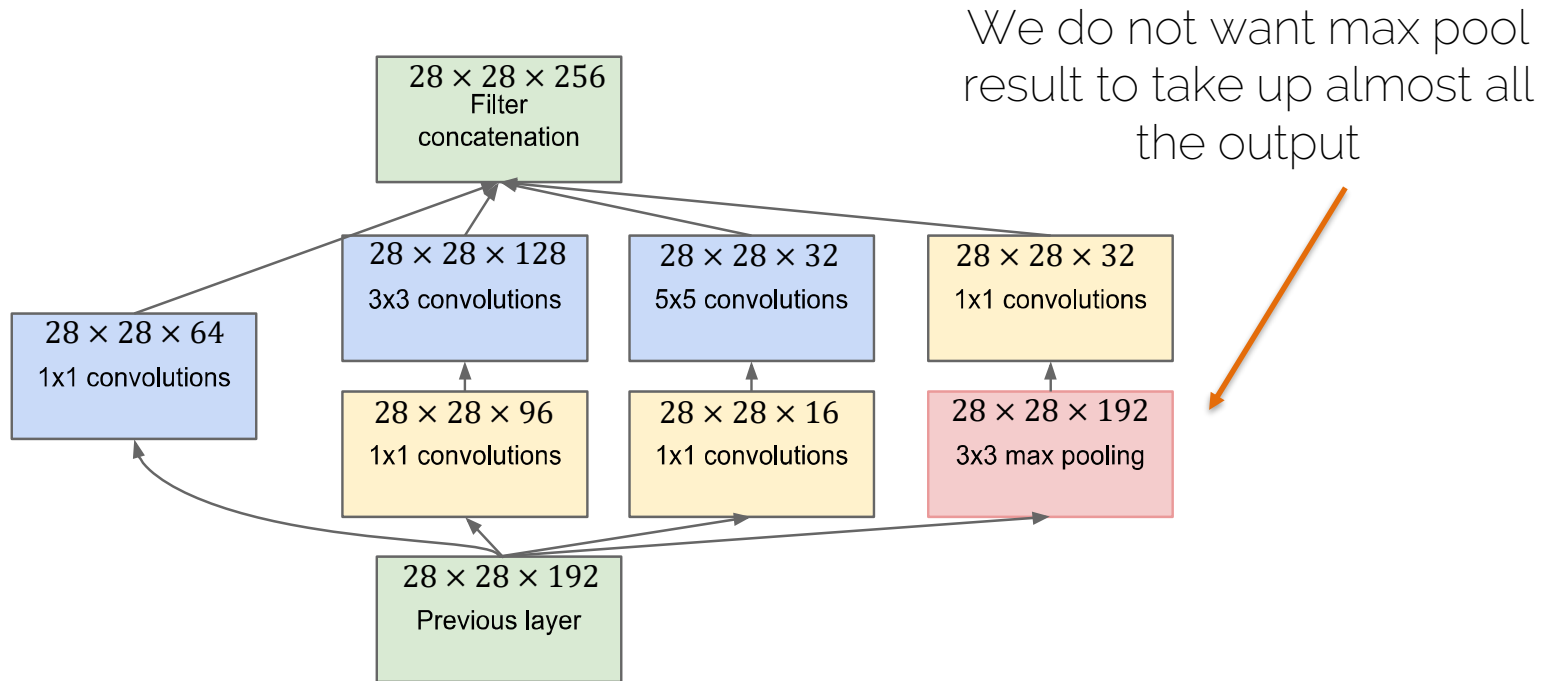


(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

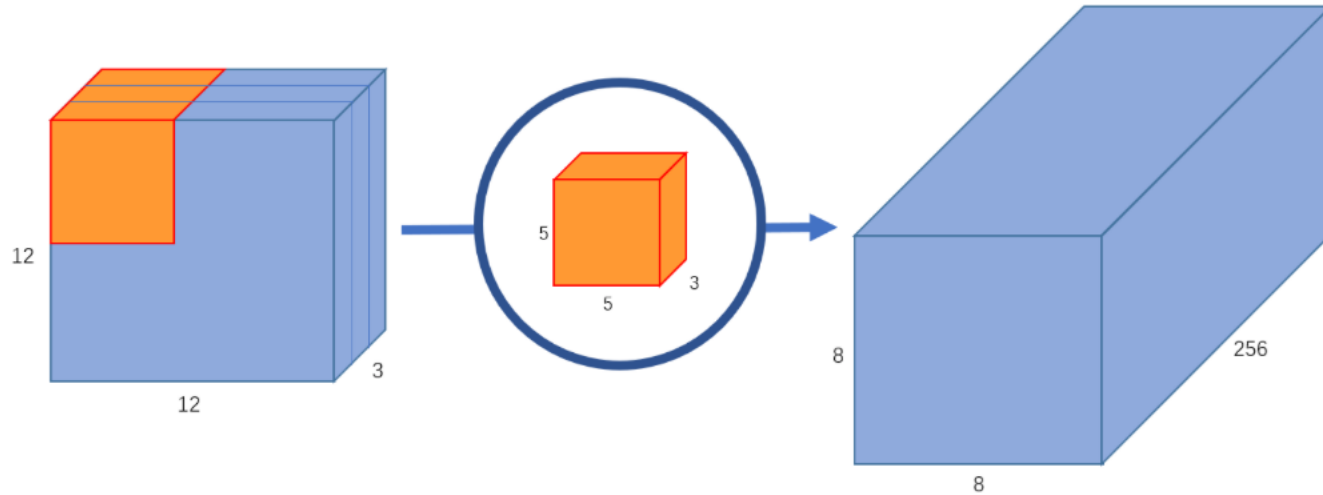
Inception Layer: Dimensions



Xception Net

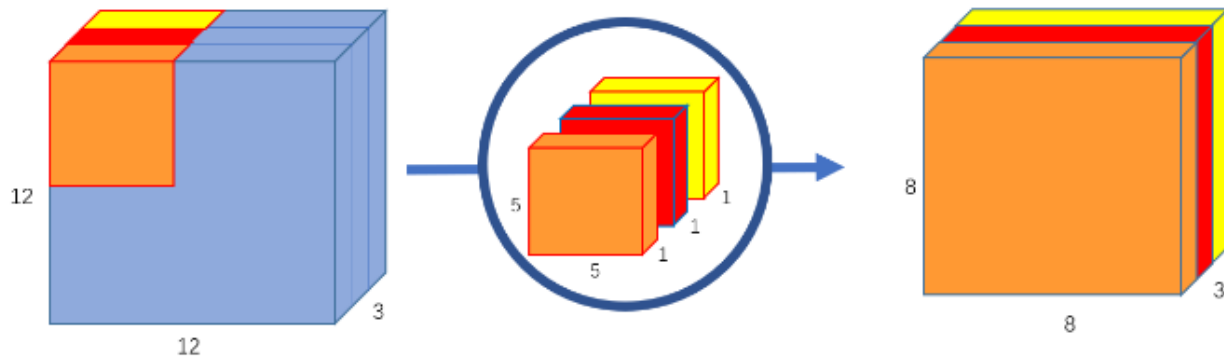
- “Extreme version of Inception”: applying (modified) **Depthwise Separable Convolutions** instead of normal convolutions
- 36 conv layers, structured into several modules with **skip connections**
- outperforms Inception Net V3

Depth-wise separable convolutions



Normal convolutions act on all channels.

Depth-wise separable convolutions

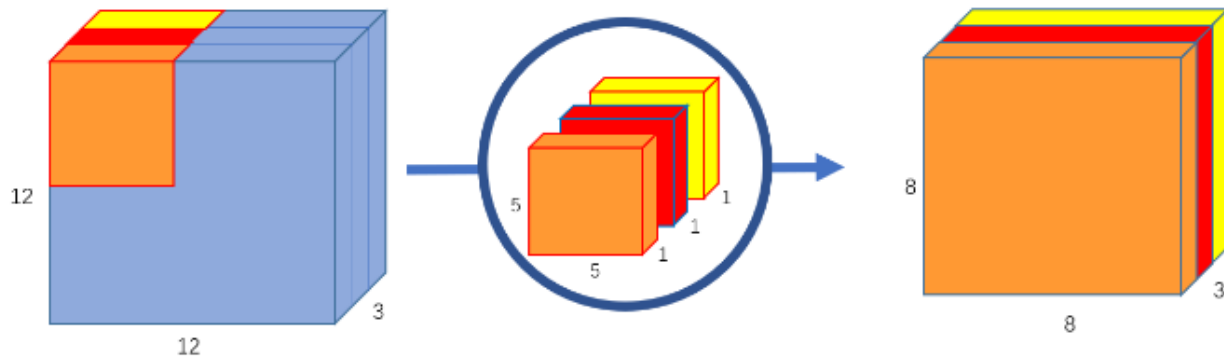


Filters are applied only at certain depths of the features. Normal convolutions have groups set to 1, the convolutions used in this image have groups set to 3.

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

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

Depth-wise separable convolutions

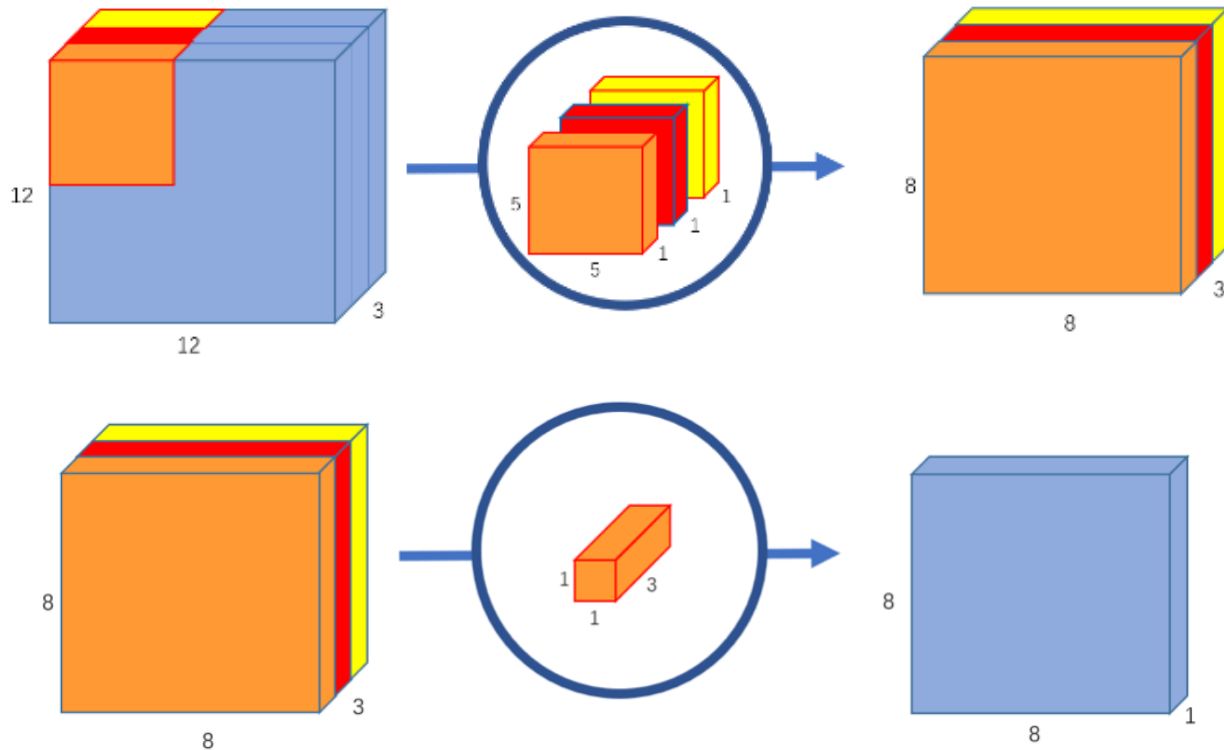


But the depth size is always the same!

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

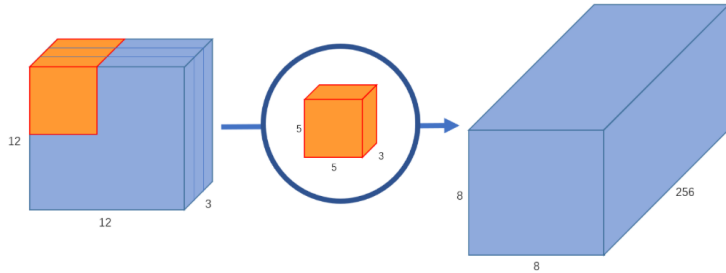
```
classtorch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)
```

Depth-wise separable convolutions



Solution:
1x1 convs!

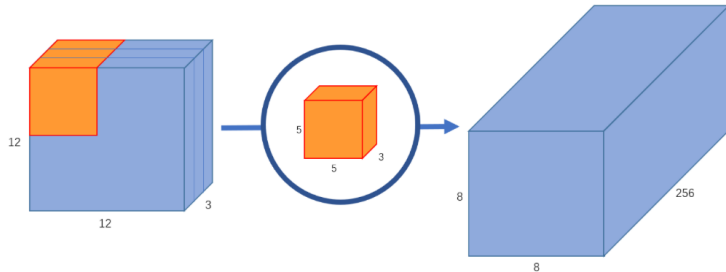
But why?



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?

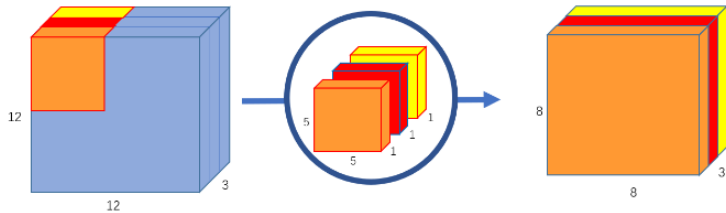


Original convolution

256 kernels of size 5x5x3

Multiplications:

$256 \times 5 \times 5 \times 3 \times (8 \times 8 \text{ locations}) = 1,228,800$

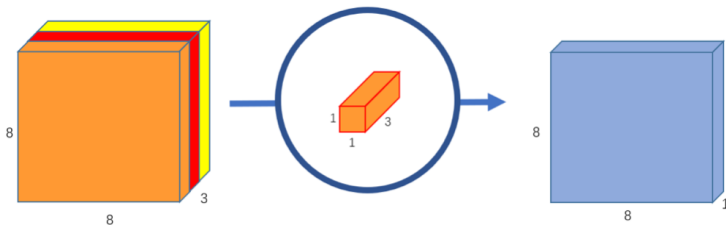


Depth-wise convolution

3 kernels of size 5x5x1

Multiplications:

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



1x1 convolution

256 kernels of size 1x1x3

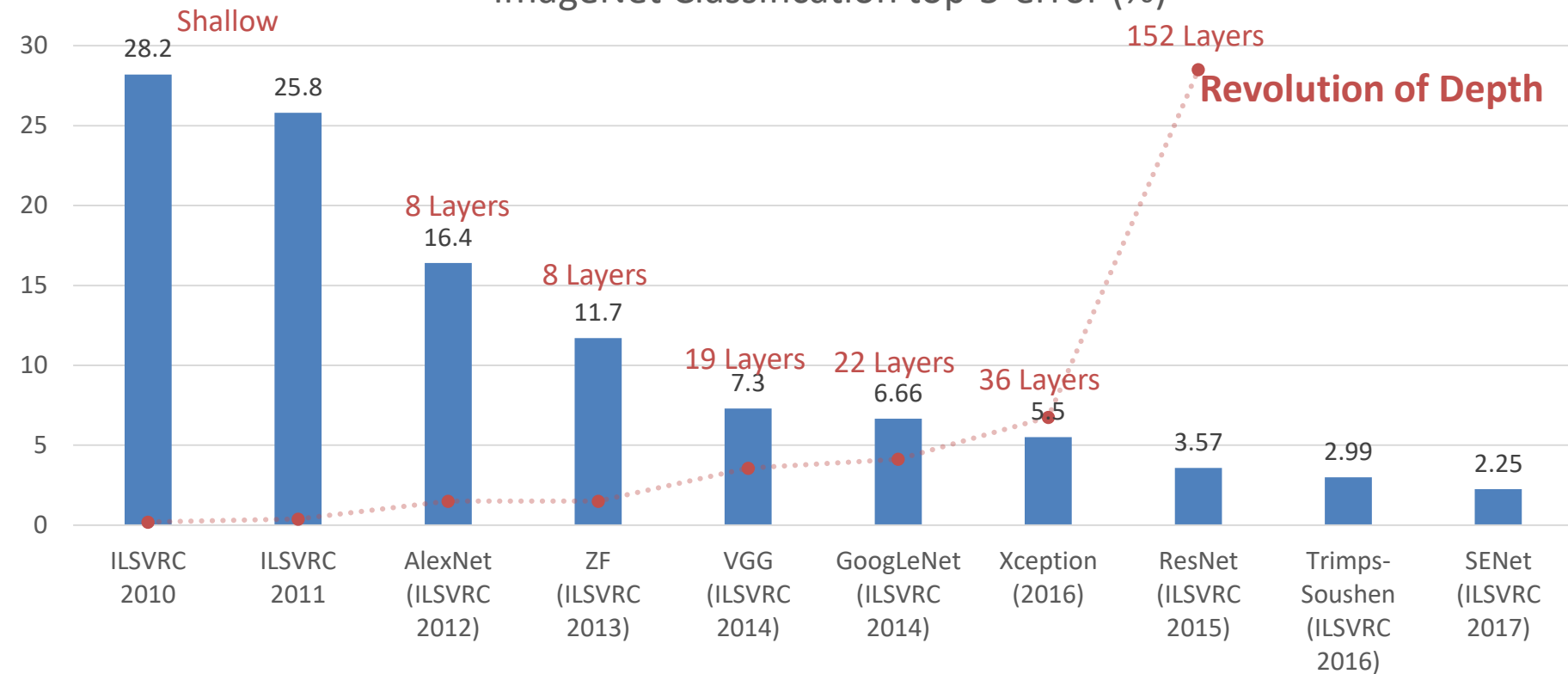
Multiplications:

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

Less
computation!

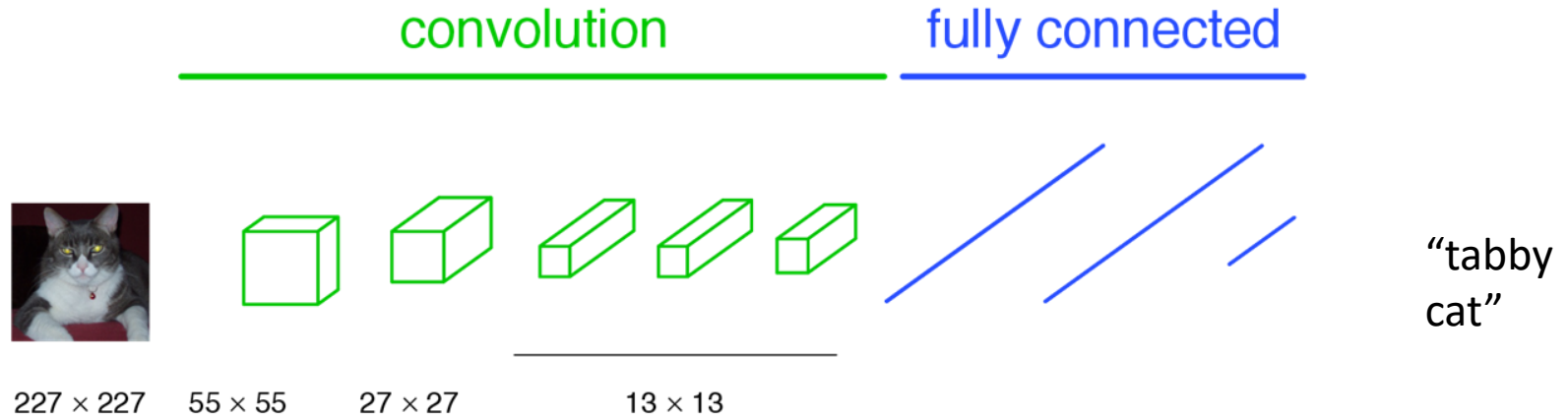
ImageNet Benchmark

ImageNet Classification top-5-error (%)

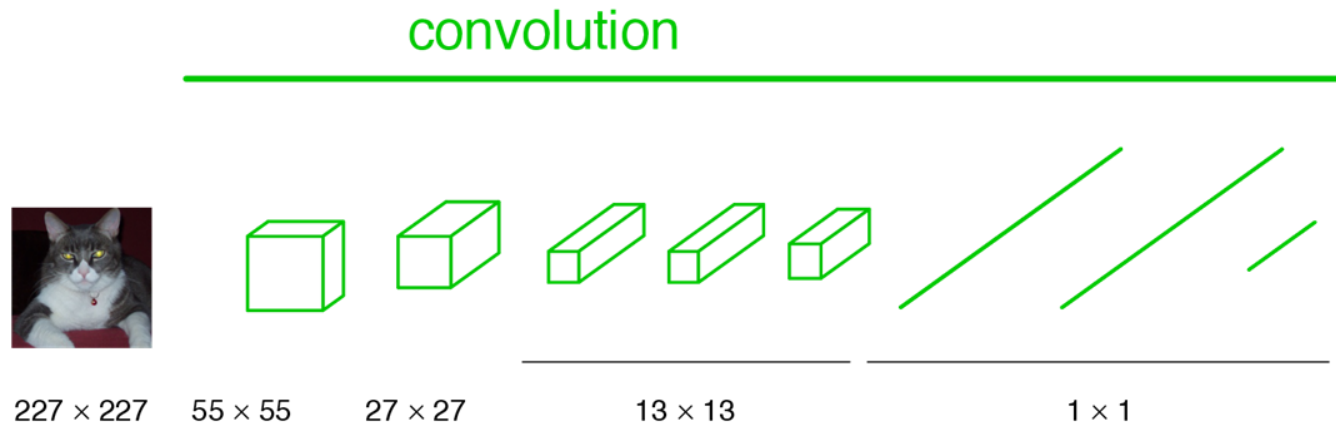


Fully Convolutional Network

Classification Network



FCN: Becoming Fully Convolutional



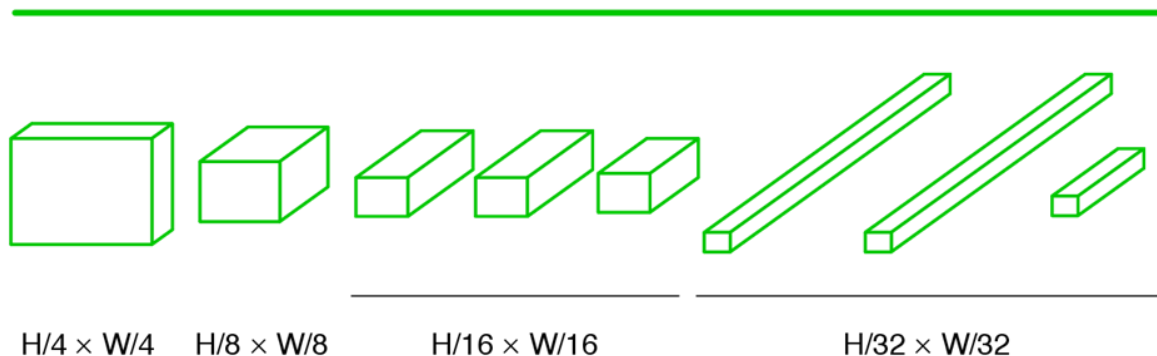
Convert fully connected layers to convolutional layers!

FCN: Becoming Fully Convolutional



$H \times W$

convolution



$H/4 \times W/4$

$H/8 \times W/8$

$H/16 \times W/16$

$H/32 \times W/32$

FCN: Upsampling Output

convolution



$H \times W$



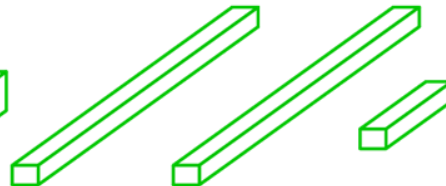
$H/4 \times W/4$



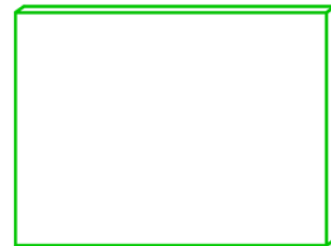
$H/8 \times W/8$



$H/16 \times W/16$

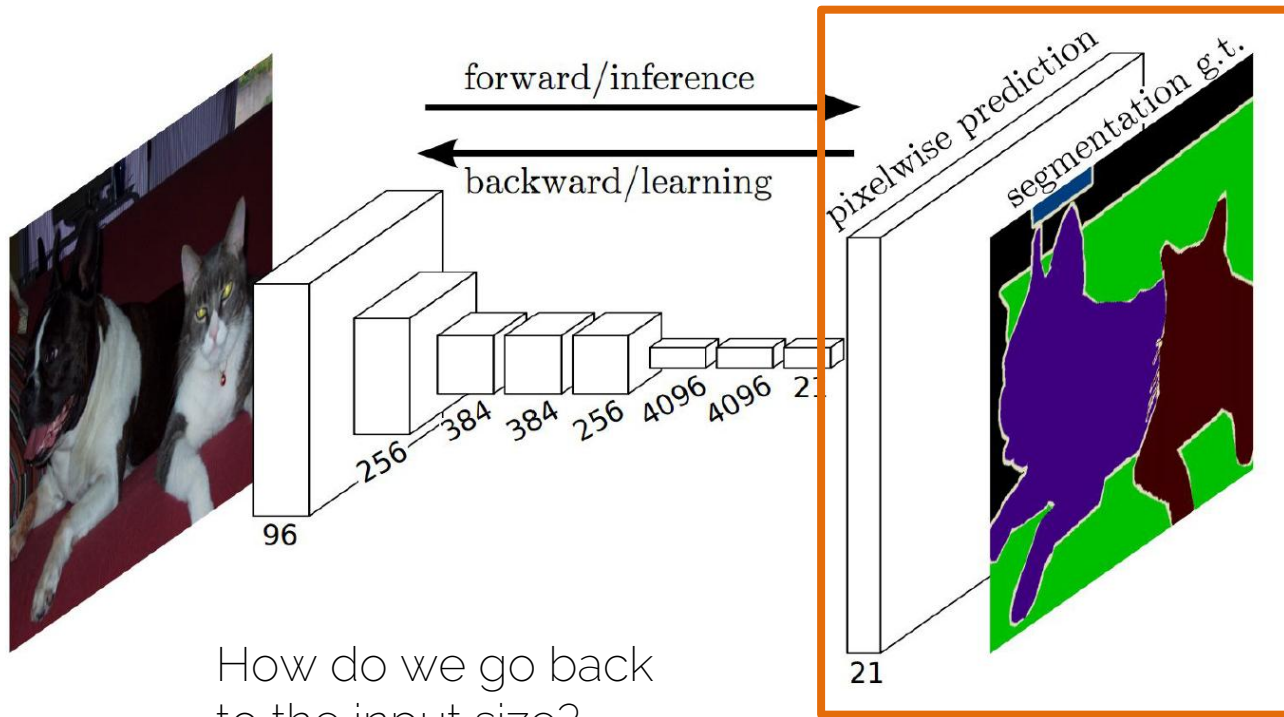


$H/32 \times W/32$



$H \times W$

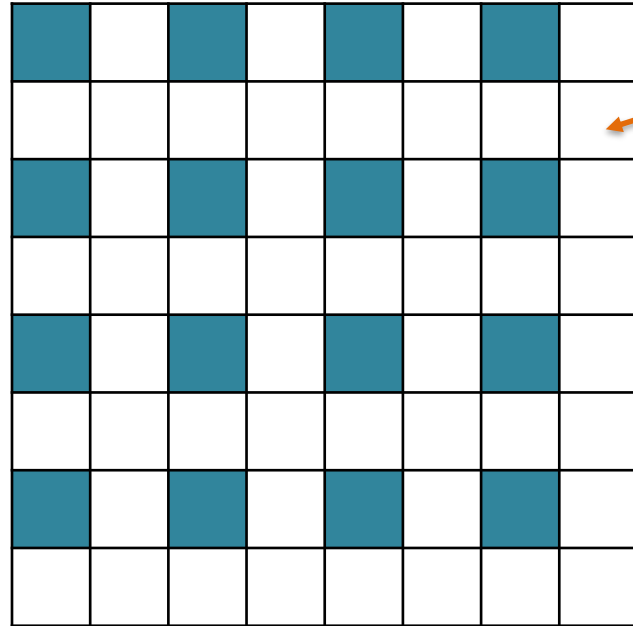
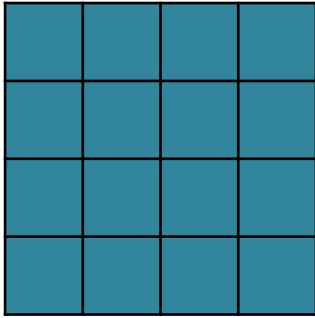
Semantic Segmentation (FCN)



[Long and Shelhamer. 15] FCN

Types of Upsampling

- 1. Interpolation



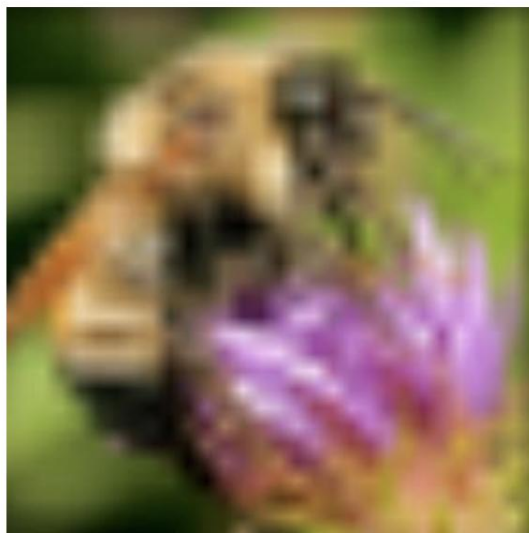
Types of Upsampling

- 1. Interpolation

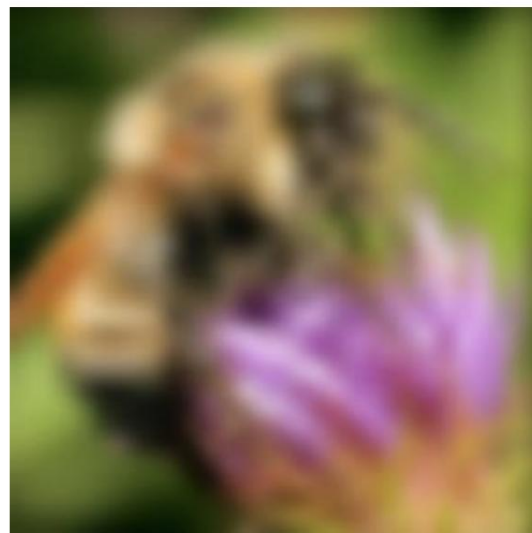
Original image  x 10



Nearest neighbor interpolation



Bilinear interpolation



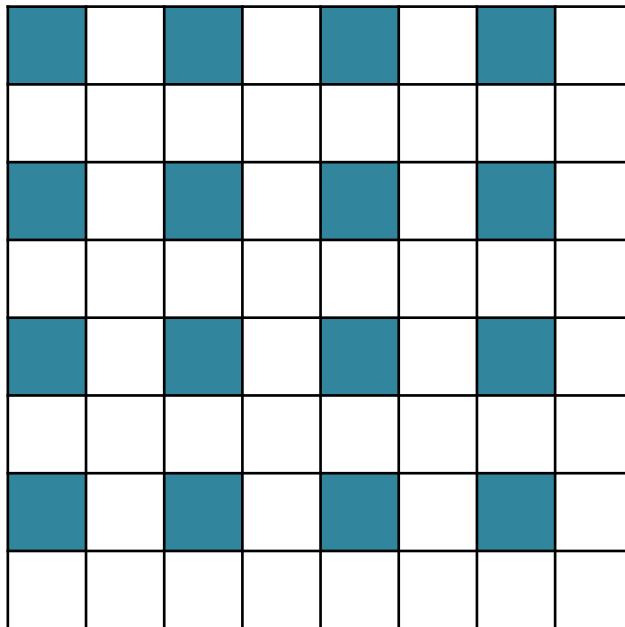
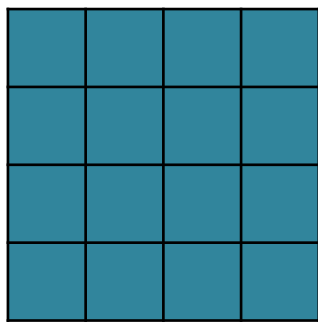
Bicubic interpolation

Types of Upsampling

- 1. Interpolation
 - ✓ Few artifacts

Types of Upsampling

- 2. Transposed conv



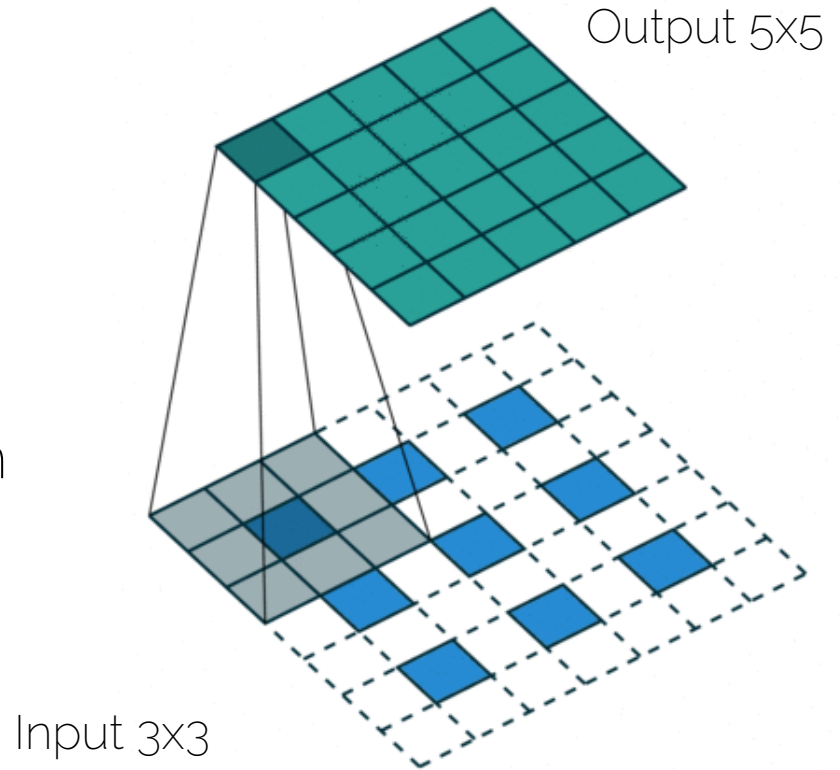
+ CONVS

✓ efficient

[Dosovitskiy, Springenberg, Brox, "Learning to Generate Chairs with Convolutional Networks", CVPR 2015]

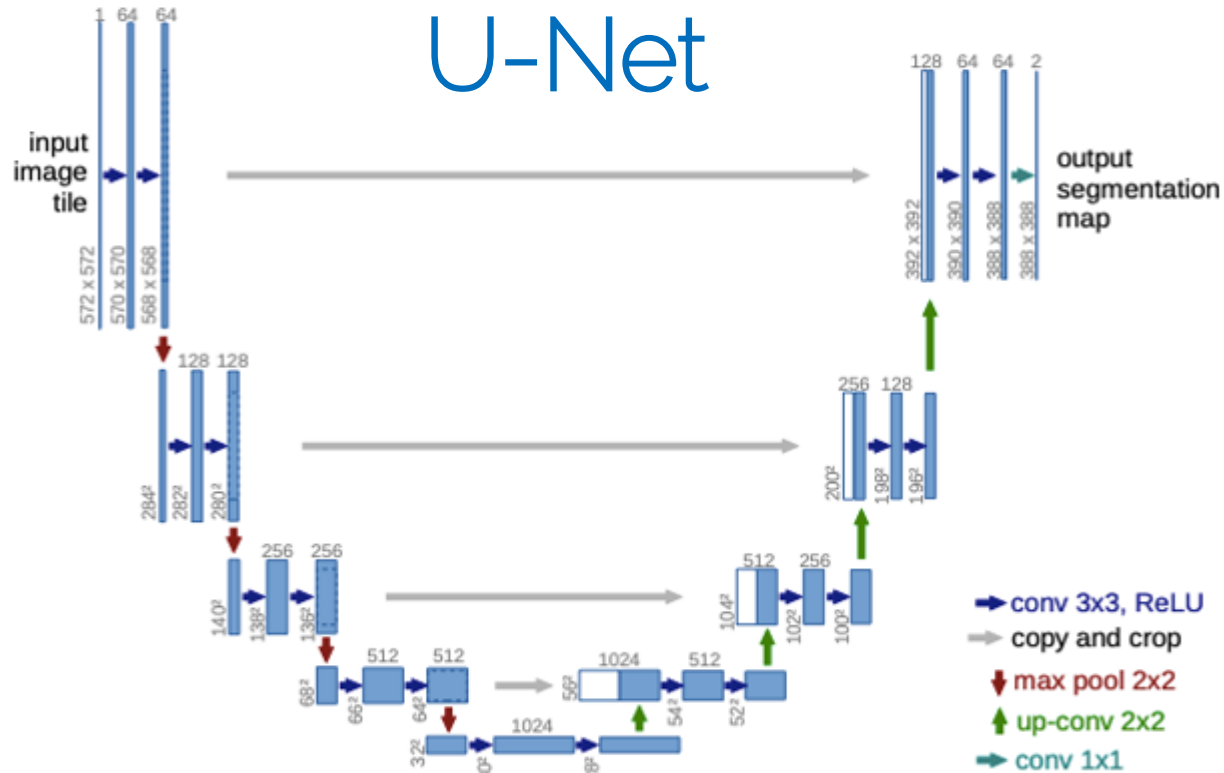
Types of Upsampling

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

[Ronneberger, Fischer, Brox, "U-net: Convolutional networks for biomedical image segmentation", MICCAI'15]

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, Fischer, Brox, "U-net: Convolutional networks for biomedical image segmentation", MICCAI'15]

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: 

[Ronneberger, Fischer, Brox, "U-net: Convolutional networks for biomedical image segmentation", MICCAI'15]

Object Detection

What is Object Detection?

Classification



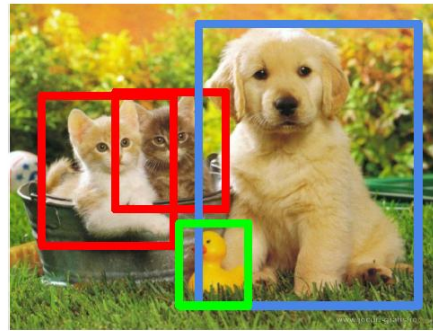
CAT

Classification
+ Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance
Segmentation



CAT, DOG, DUCK

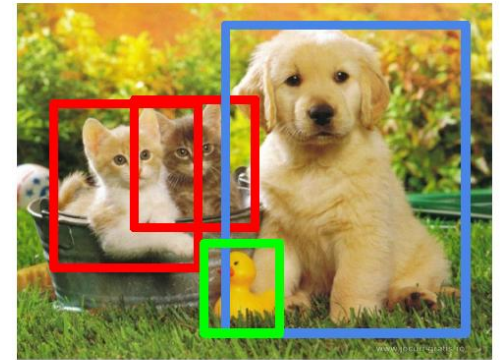
Single object

Multiple objects

What is Object Detection?

- Goal: localize + classify objects in an image
- Input: image
- Output: list of bounding boxes, class labels (+ confidences)

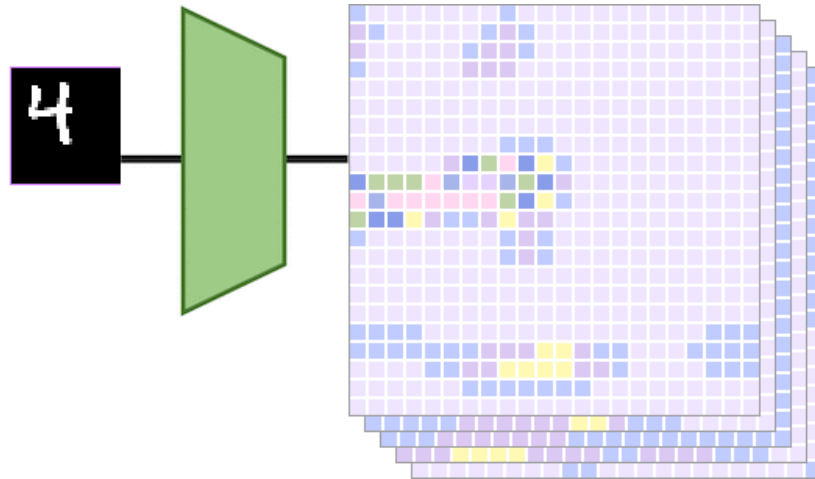
Object Detection



CAT, DOG, DUCK

Challenges in Detection

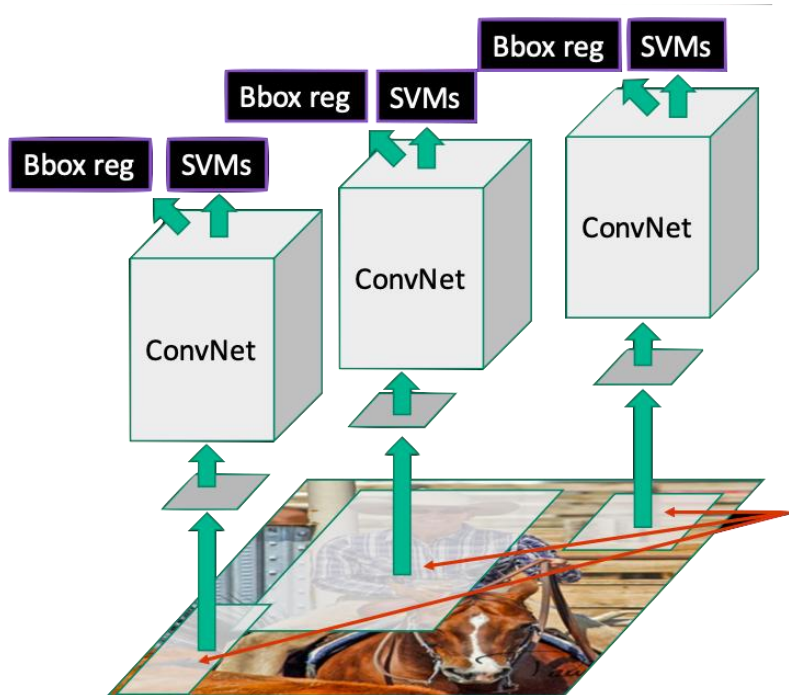
- Convolutions are translationally equivariant
 - Convolutions share weights: same filters slide across all positions
 - Shifting input image \rightarrow output feature maps will shift by same amount



Challenges in Detection

- Convolutions are translationally equivariant
- Good for identifying “what” (e.g., classification), makes “where” challenging
- Detection: need to detect multiple objects, likely different sizes/places

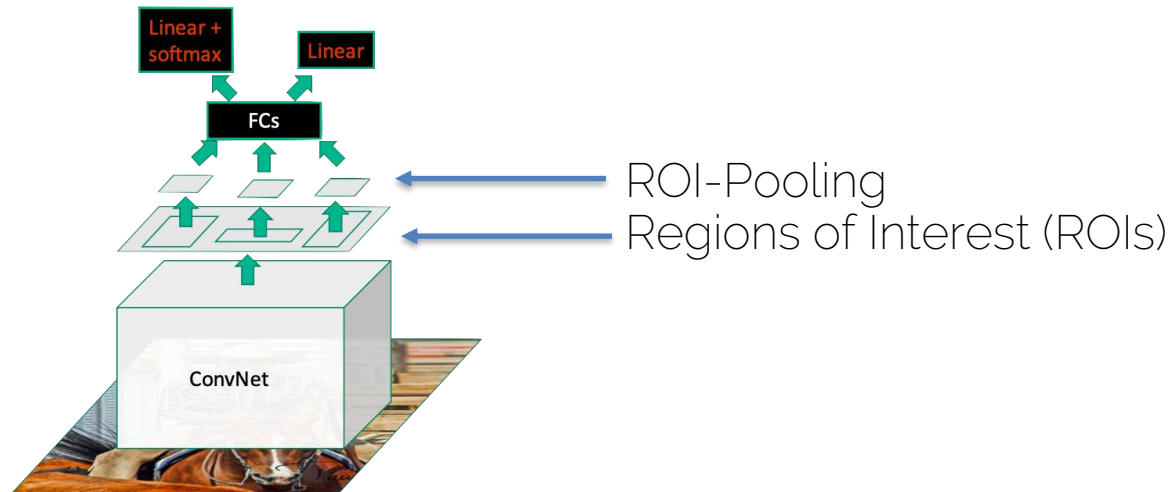
Region-Based CNNs (R-CNNs)



- Use selective search to generate candidate regions
- Warp/rescale each region to fixed image size, pass through a CNN
- CNN extracts features for predicting class + box offsets
- Slow: runs CNN separately for each region

Fast R-CNN

- Instead of running CNN once per region, run CNN once for the entire image
- Use region of interest (ROI) pooling to extract fixed-size feature maps for each region

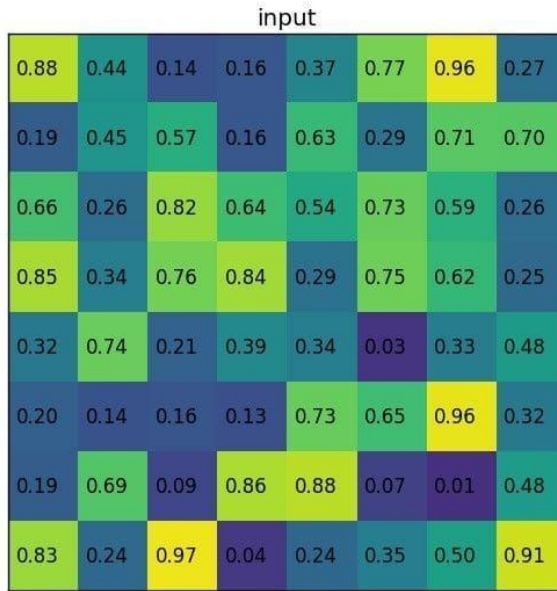


Region of Interest Pooling

- Used to handle variable-sized object proposals efficiently
 - Proposed regions have different sizes and aspect ratios
 - Fully-connected layers expect fixed-size inputs
- Convert each variable-sized region of interest (ROI) from the feature map and rescale it into a fixed-sized output using max pooling

Region of Interest Pooling

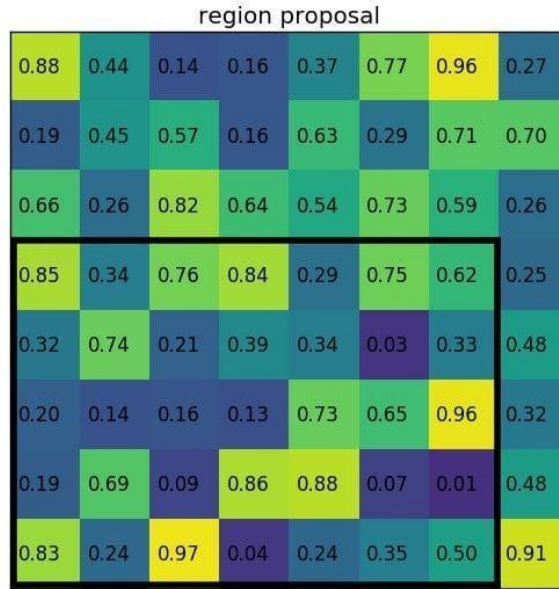
- Input: 8x8 feature map
- Output size: 2x2



Source: <https://deepsense.ai/region-of-interest-pooling-explained/>

Region of Interest Pooling

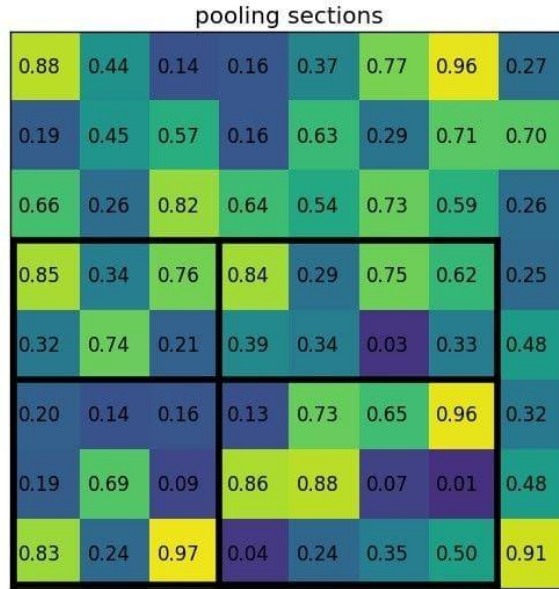
- Input: 8x8 feature map
- Output size: 2x2



Source: <https://deepsense.ai/region-of-interest-pooling-explained/>

Region of Interest Pooling

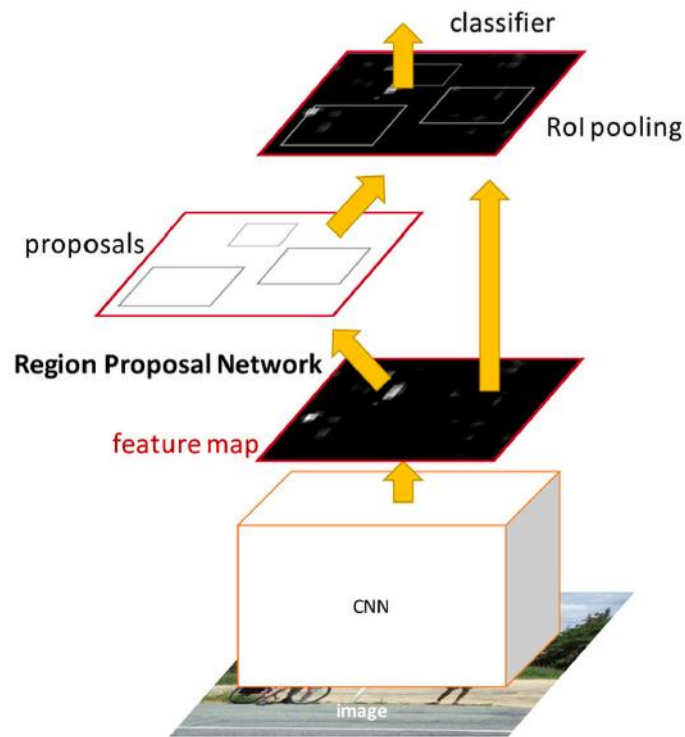
- Input: 8x8 feature map
- Output size: 2x2



Source: <https://deepsense.ai/region-of-interest-pooling-explained/>

Faster R-CNN

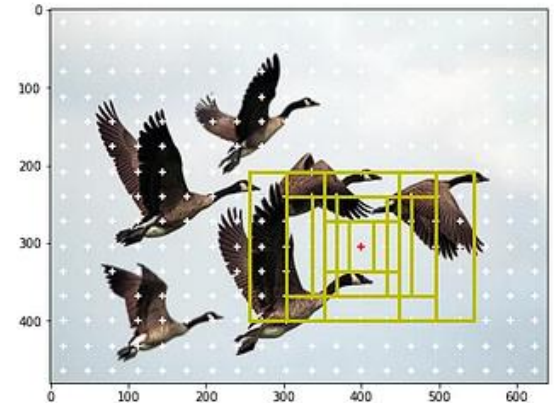
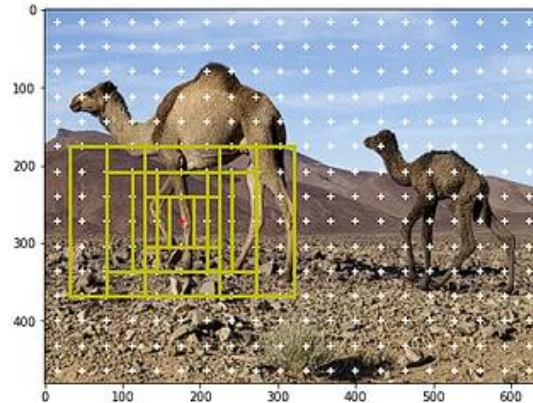
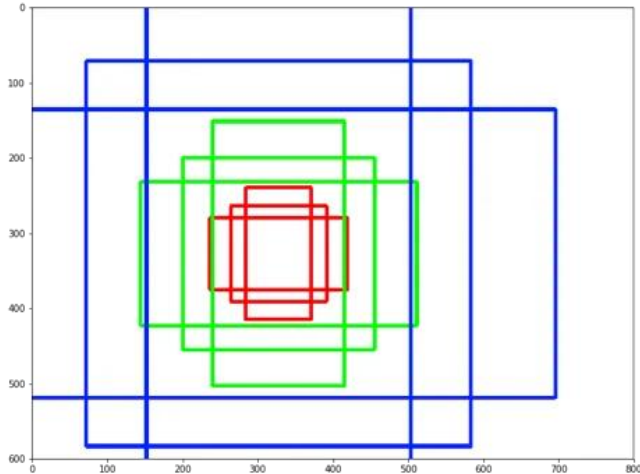
- Use convolutions to get sliding window effect
- Replace selective search with a region proposal network (RPN)
 - Small CNN that slides over feature map
 - Produces region proposals based on anchors



[Ren et al. NeurIPS'15] Faster R-CNN

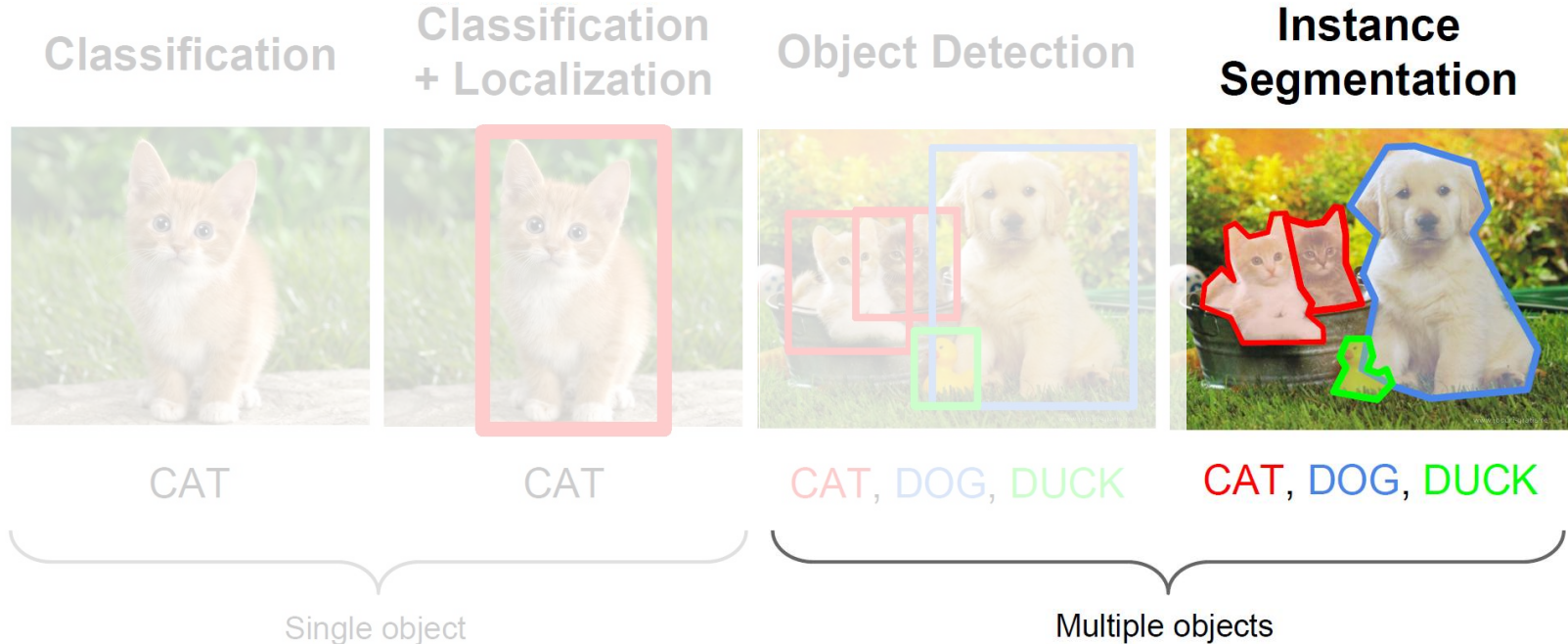
Anchors

- Potential bounding box candidates where an object can be detected



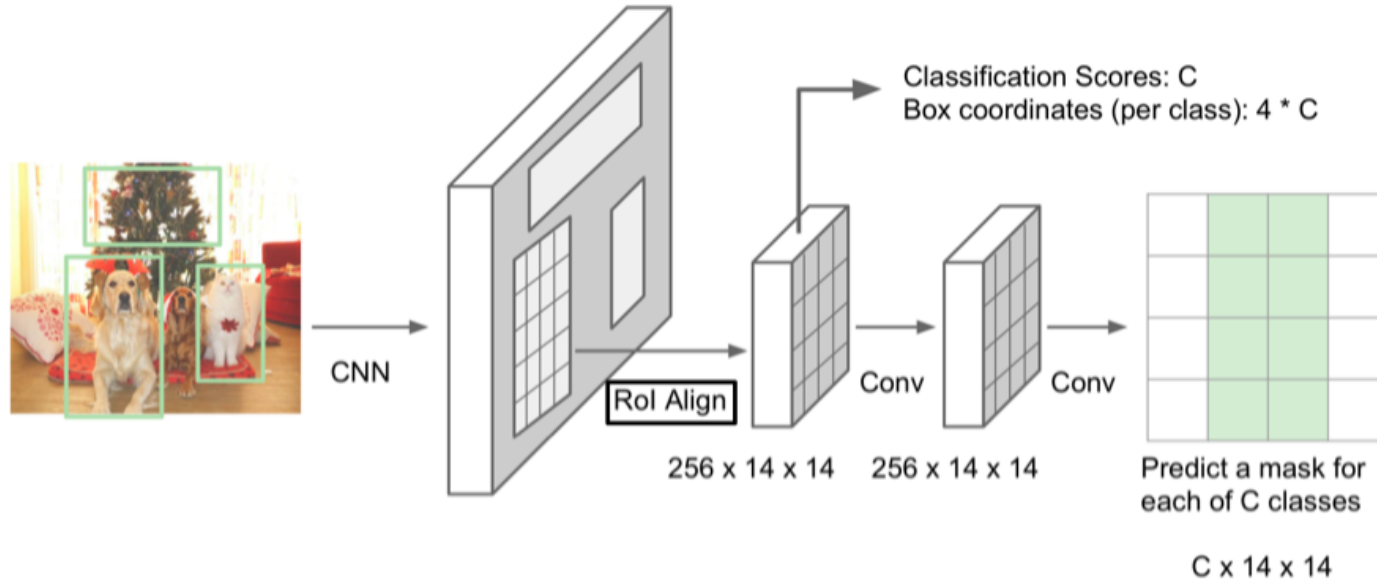
Instance Segmentation

- Predict list of object bounding boxes, classes, and pixel-wise masks



Mask R-CNN

- Extend Faster R-CNN to also predict pixel-wise mask for each object



R-CNNs for Detection + Segmentation

- R-CNN, Fast R-CNN, Faster R-CNN, Mask-RCNN: two-stage detectors
- First generate region proposals, then refine boxes + predict class labels (+ masks)
- Today: end-to-end, unified approach with transformers

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]