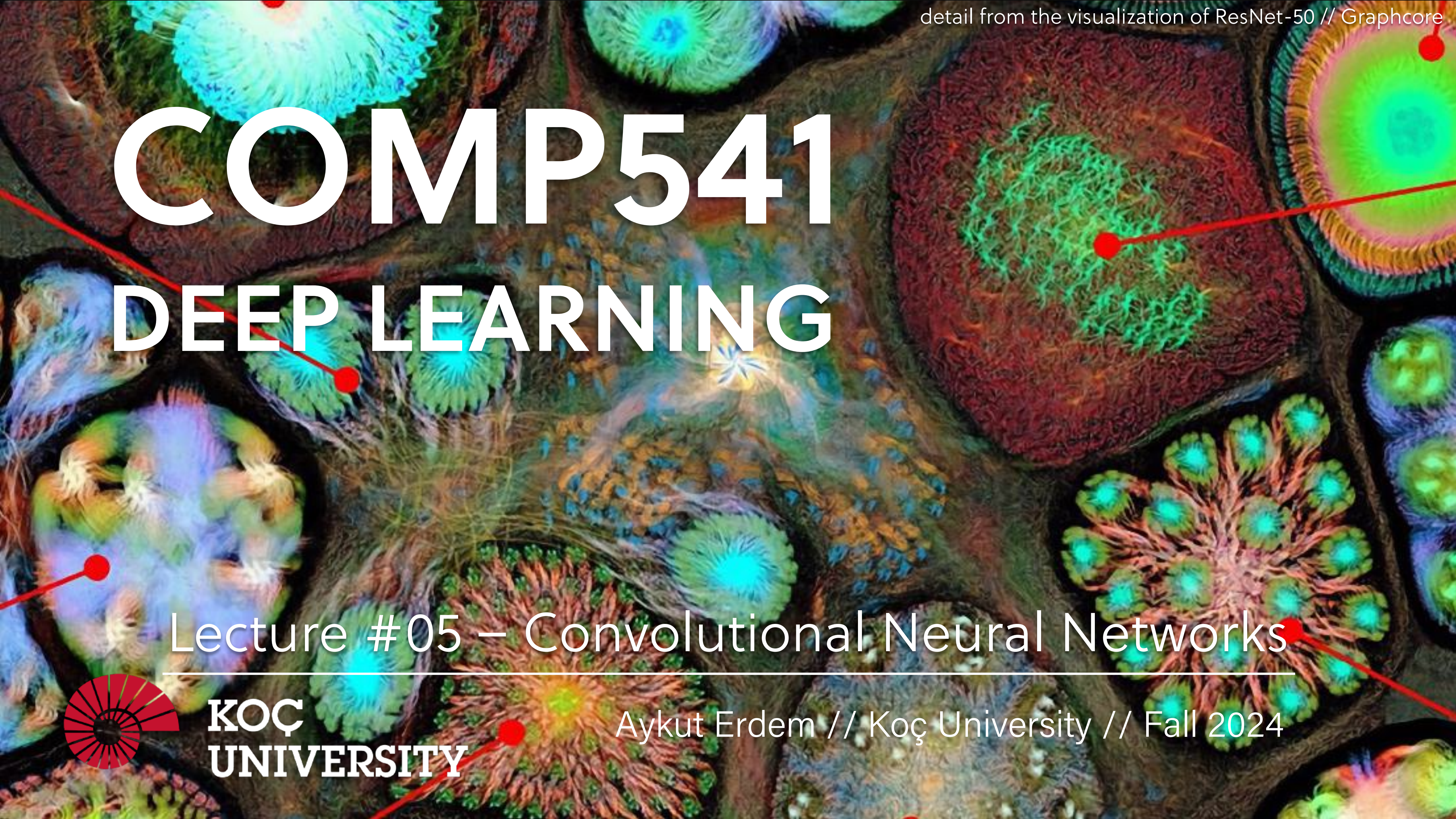

detail from the visualization of ResNet-50 // Graphcore

COMP541
DEEP LEARNING

Lecture #05 – Convolutional Neural Networks

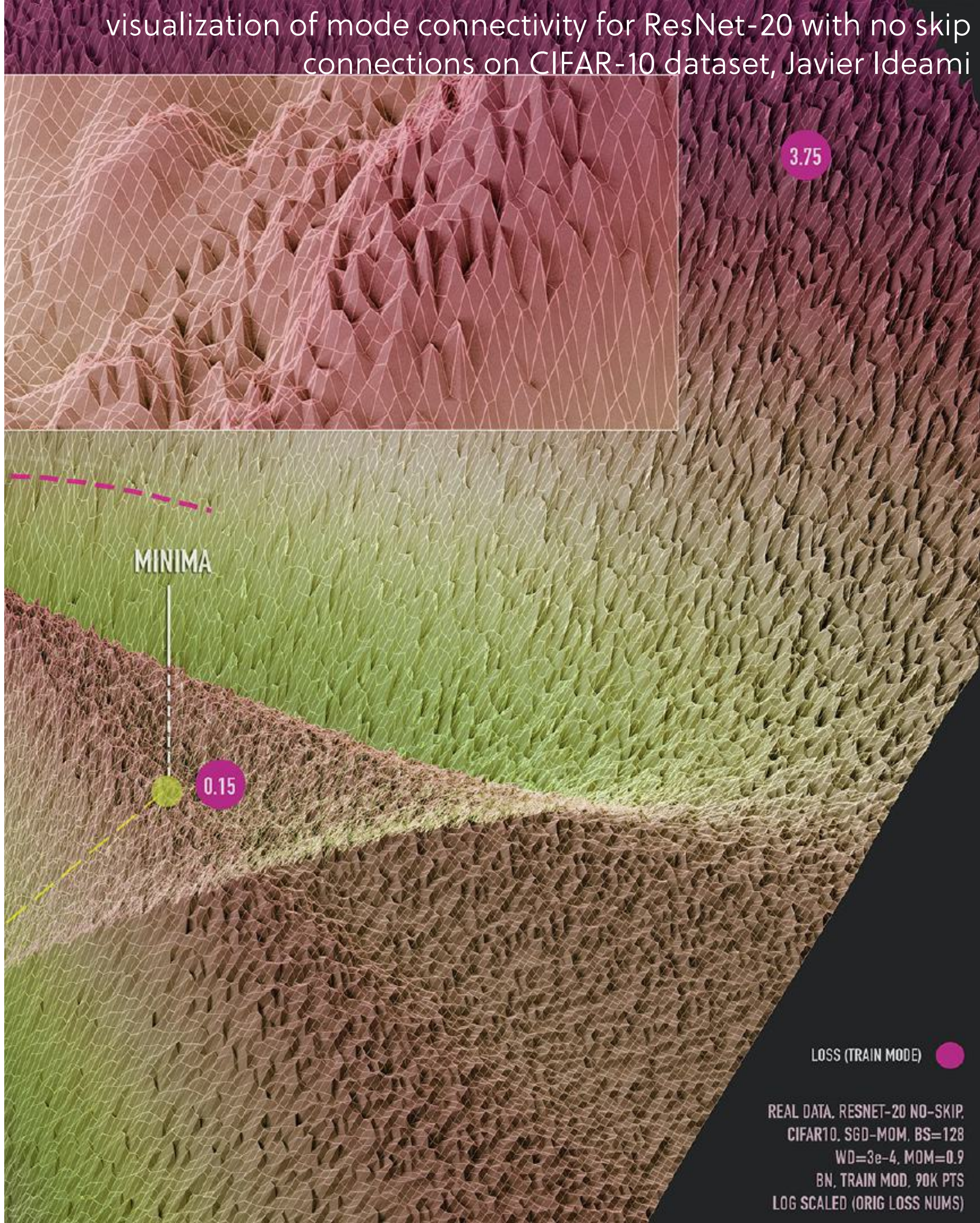
KOÇ UNIVERSITY

Aykut Erdem // Koç University // Fall 2024

Previously on COMP541

- data preprocessing and normalization
- weight initializations
- ways to improve generalization
- optimization

visualization of mode connectivity for ResNet-20 with no skip connections on CIFAR-10 dataset, Javier Ideami



Lecture Overview

- convolution layer
- pooling layer
- cnn architectures
- design guidelines
- residual connections
- semantic segmentation networks
- addressing other tasks

Disclaimer: Much of the material and slides for this lecture were borrowed from

— Andrea Vedaldi's tutorial on Convolutional Networks for Computer Vision Applications

— Kaiming He's ICML 2016 tutorial on Deep Residual Networks: Deep Learning Gets Way Deeper

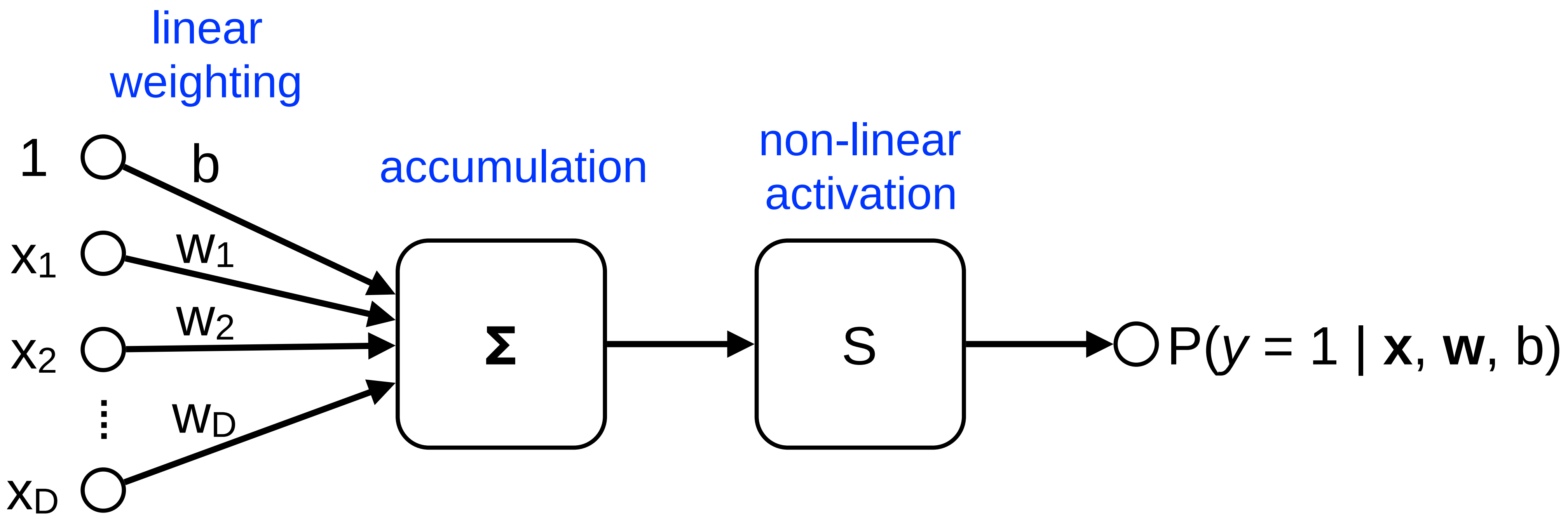
— Fei-Fei Li, Andrej Karpathy and Justin Johnson's CS231n class

— Justin Johnson's EECS 498/598 class

Perceptron

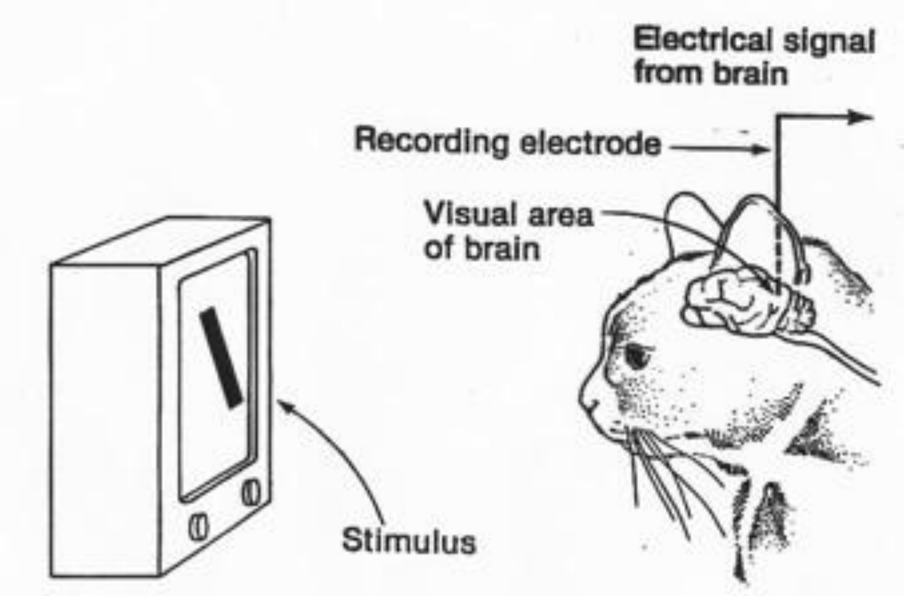
[Rosenblatt 57]

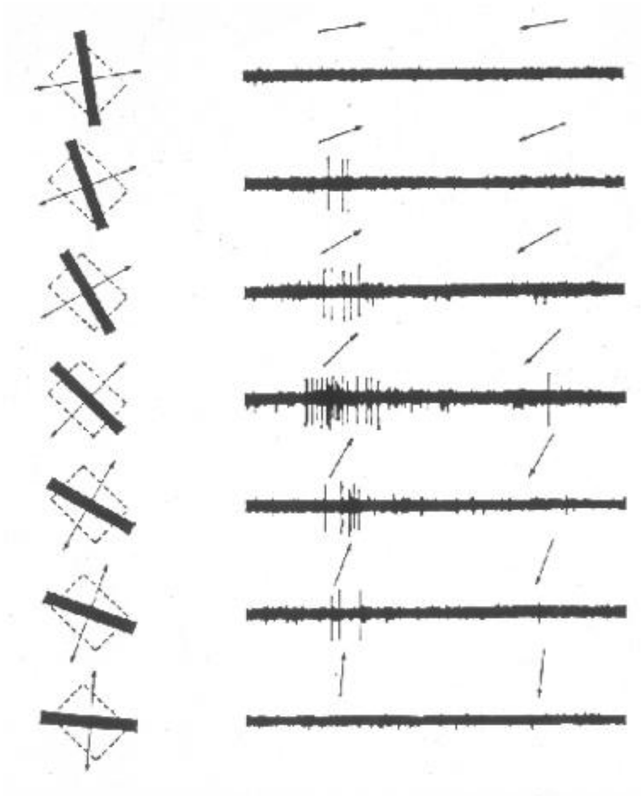
- The goal is estimating the posterior probability of the binary label y of a vector **x**:

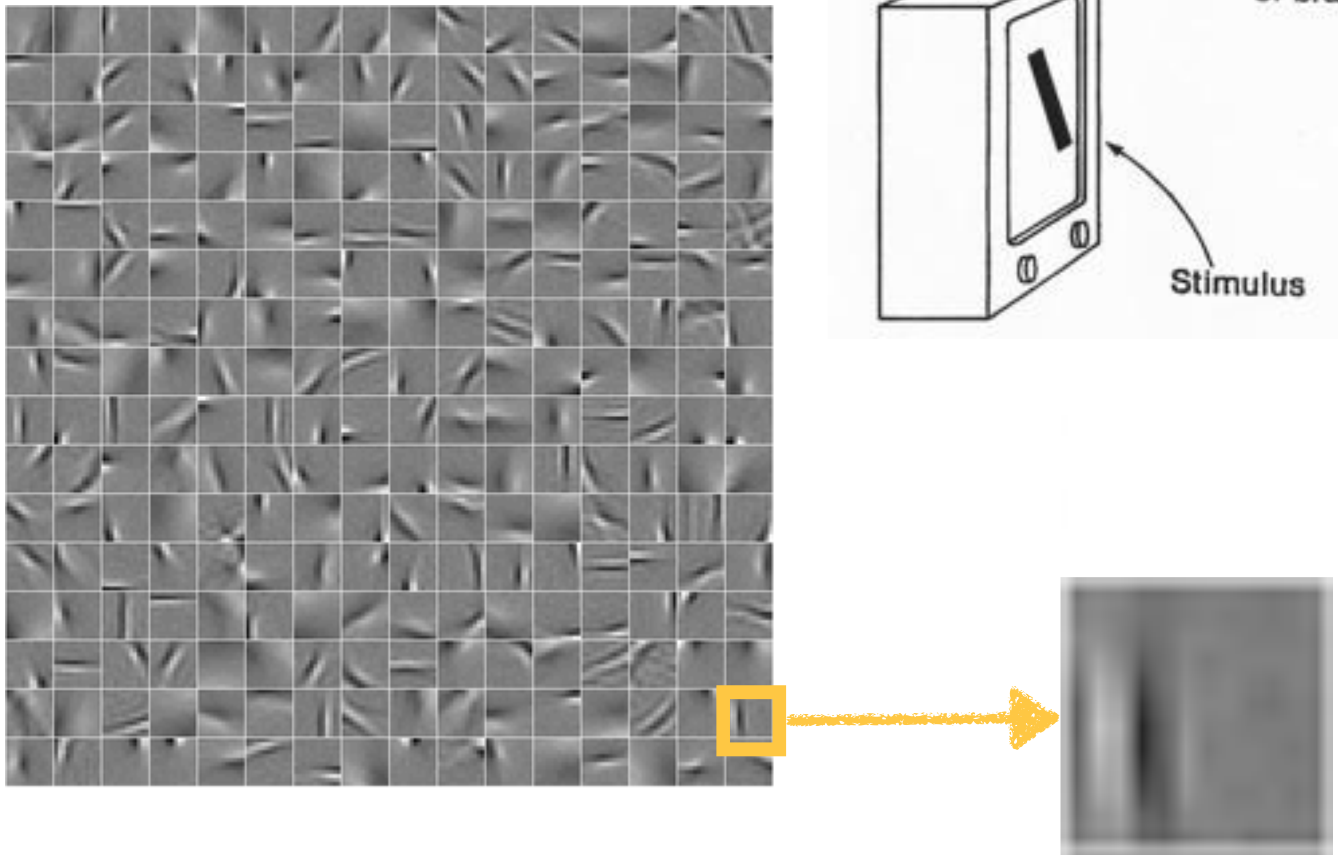


Discovery of oriented cells in the visual cortex

[Hubel and Wiesel 59]

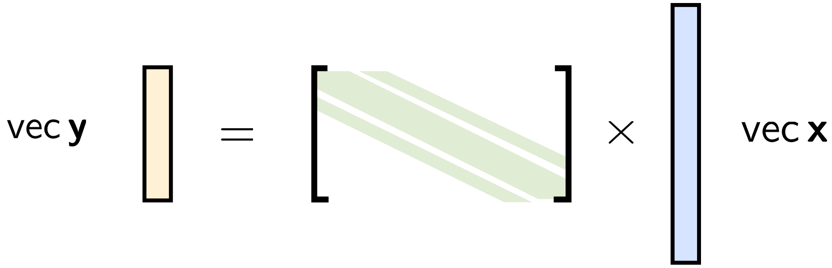






oriented filter

Convolution



- Convolution = Spatial filtering

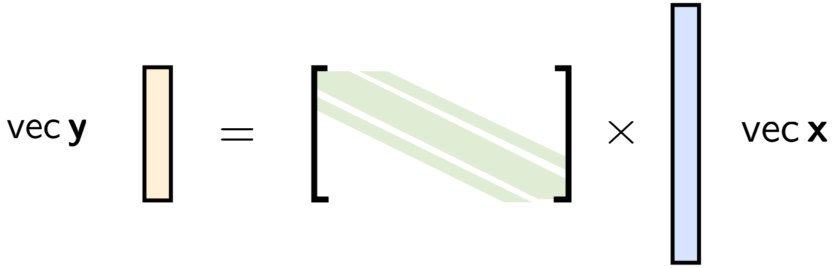
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

- Different filters (weights) reveal a different characteristics of the input.

$*^{1/8}$

0	1	0
1	4	1
0	1	0

Convolution



- Convolution = Spatial filtering

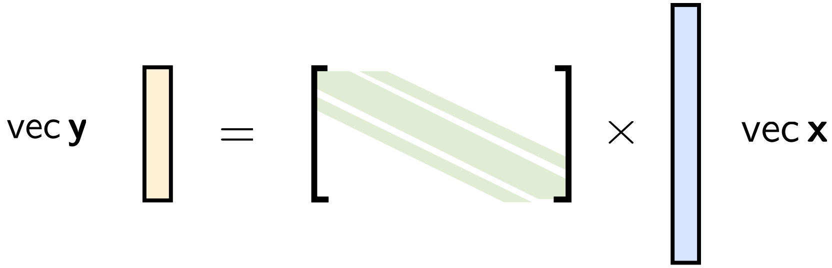
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']\,b[i-i',j-j']$$

- Different filters (weights) reveal a different characteristics of the input.

*

0	-1	0
-1	4	-1
0	-1	0

Convolution



- Convolution = Spatial filtering

$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

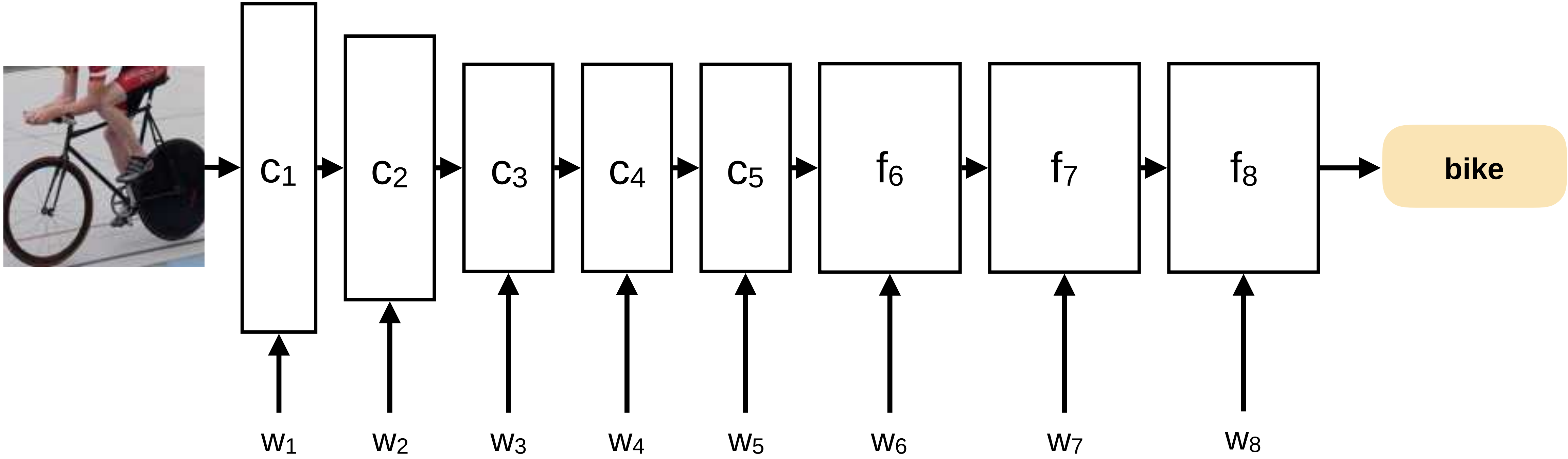
- Different filters (weights) reveal a different characteristics of the input.

*

1	0	-1
2	0	-2
1	0	-1

Convolutional Neural Networks in a Nutshell

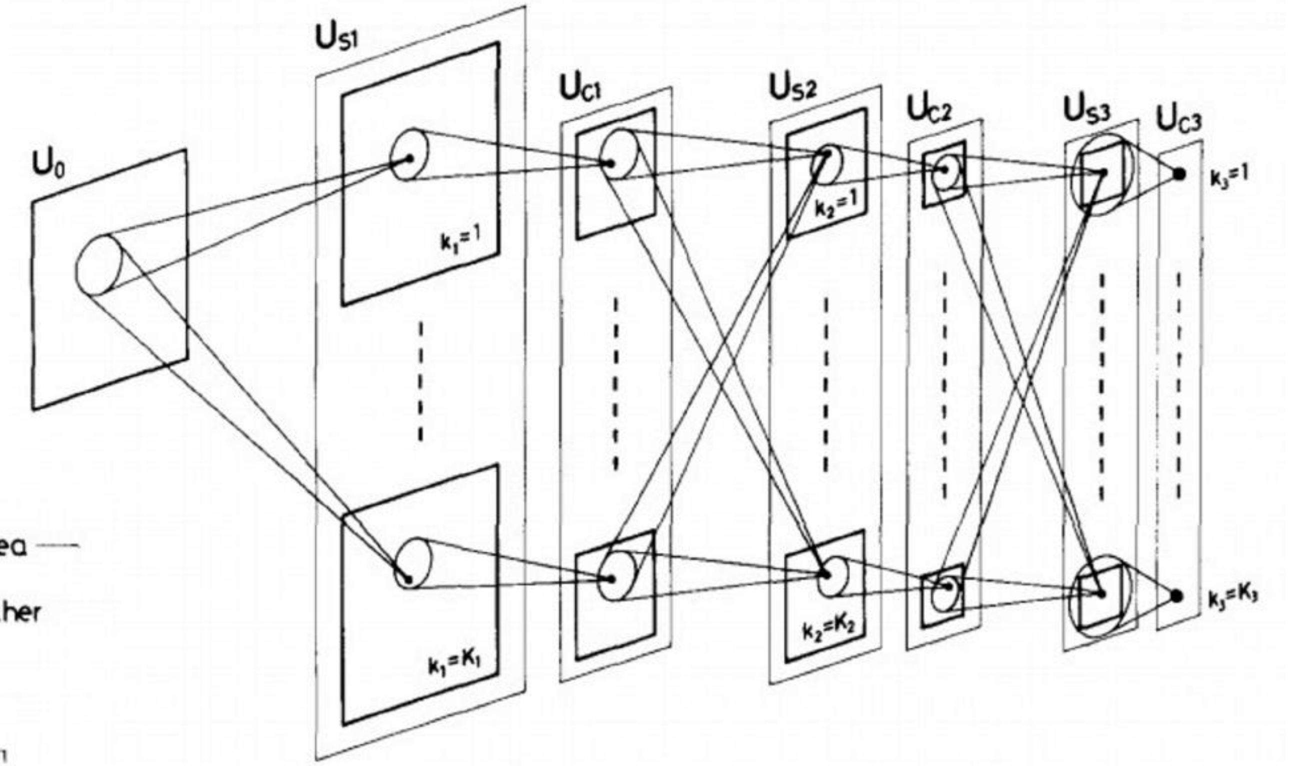
- A neural network model that consists of a sequence of local & translation-invariant layers
  - Many identical copies of the same neuron: Weight/parameter sharing
  - Hierarchical feature learning

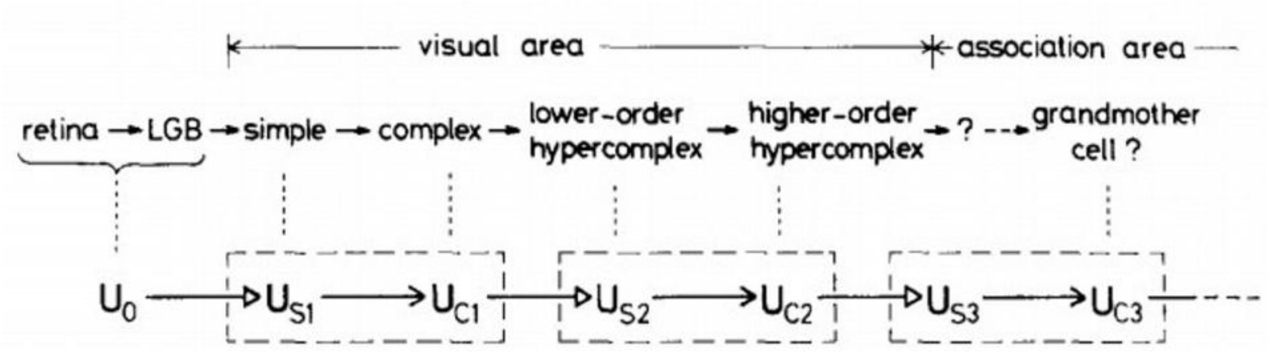


A. Krizhevsky, I. Sutskever, and G. E. Hinton. **Imagenet classification with deep convolutional neural networks**. In NIPS

A bit of history

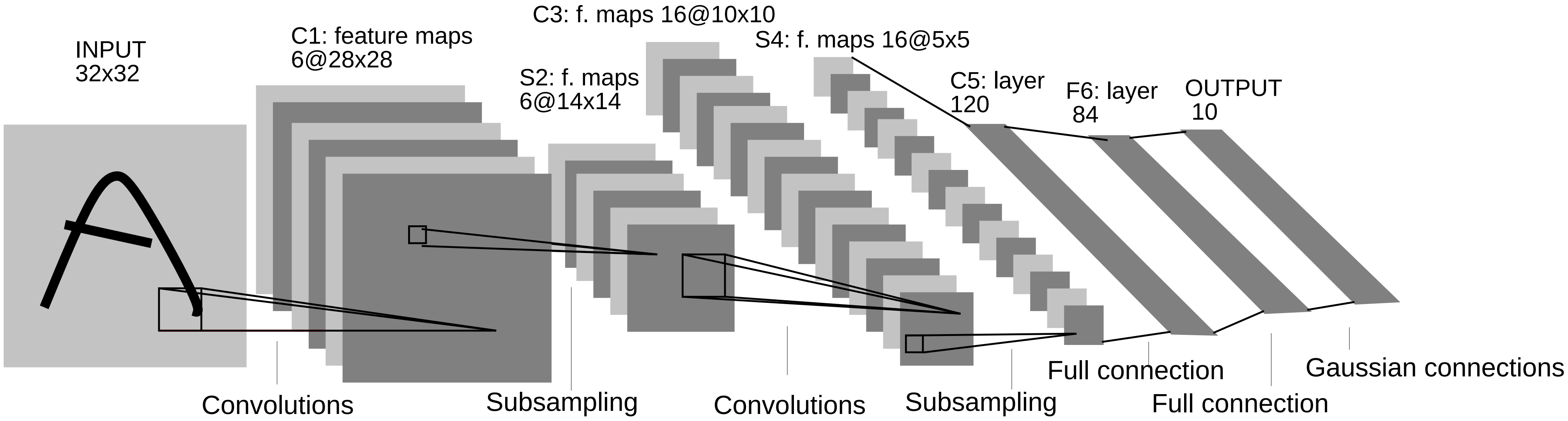
- Neocognitron model by Fukushima (1980)
- The first convolutional neural network (CNN) model
- so-called "sandwich" architecture
  - simple cells act like filters
  - complex cells perform pooling
- Difficult to train
  - No backpropagation yet





A bit of history

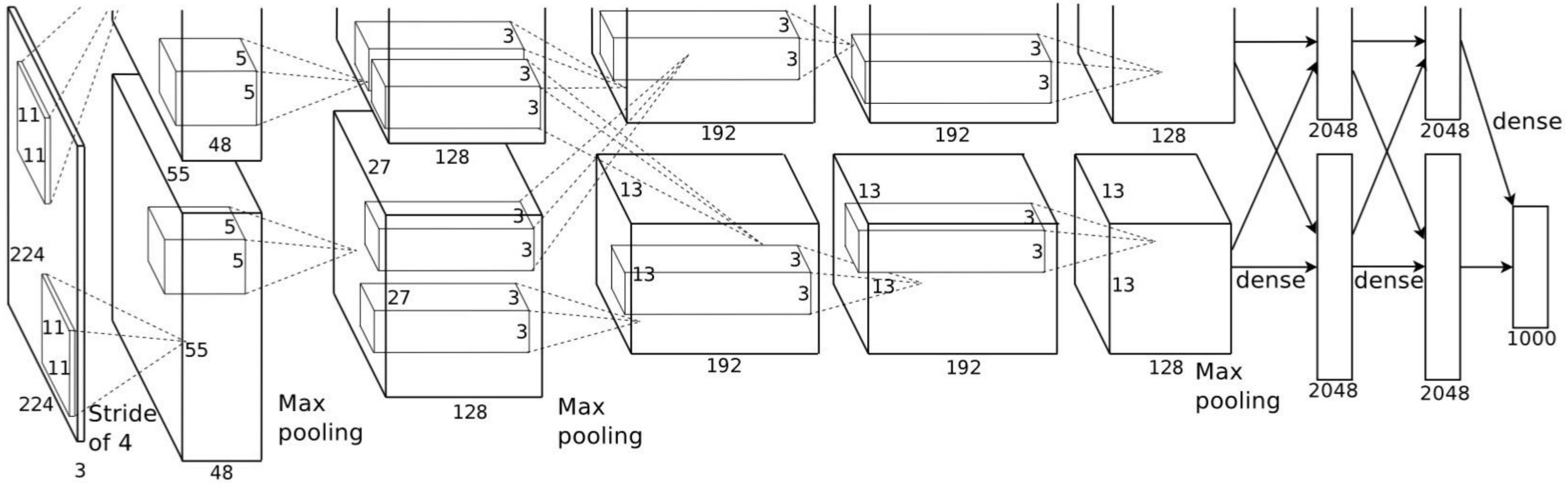
- LeNet-5 model



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE. **86** (11): 2278–2324, 1998.

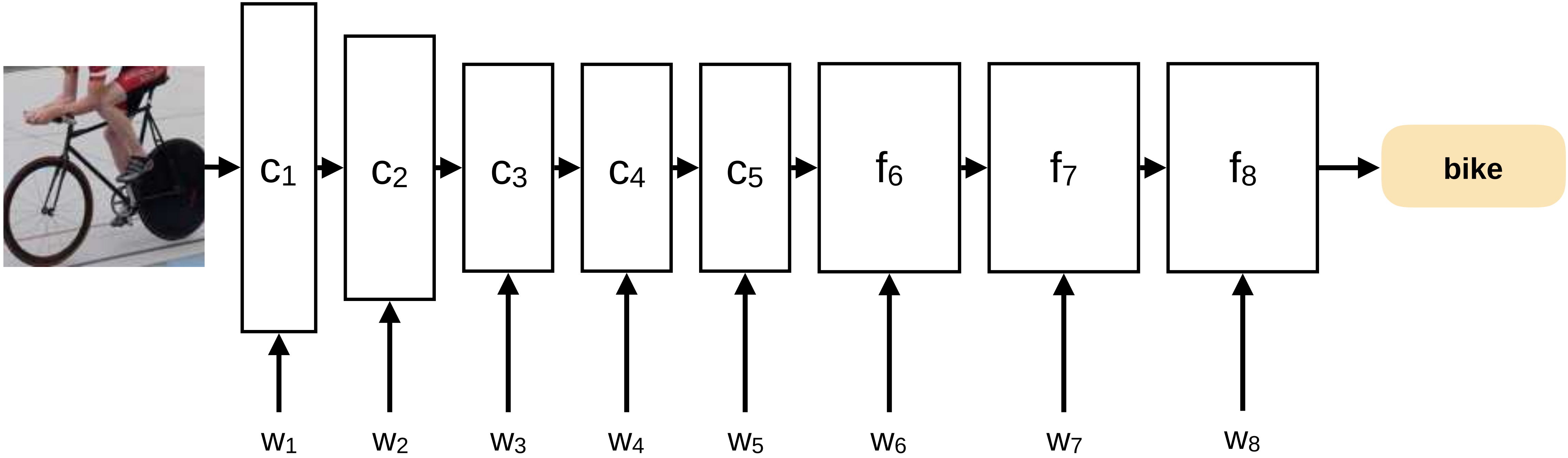
A bit of history

- AlexNet model



A. Krizhevsky, I. Sutskever, and G. E. Hinton. **Imagenet classification with deep convolutional neural networks**. In NIPS

Convolutional Neural Network

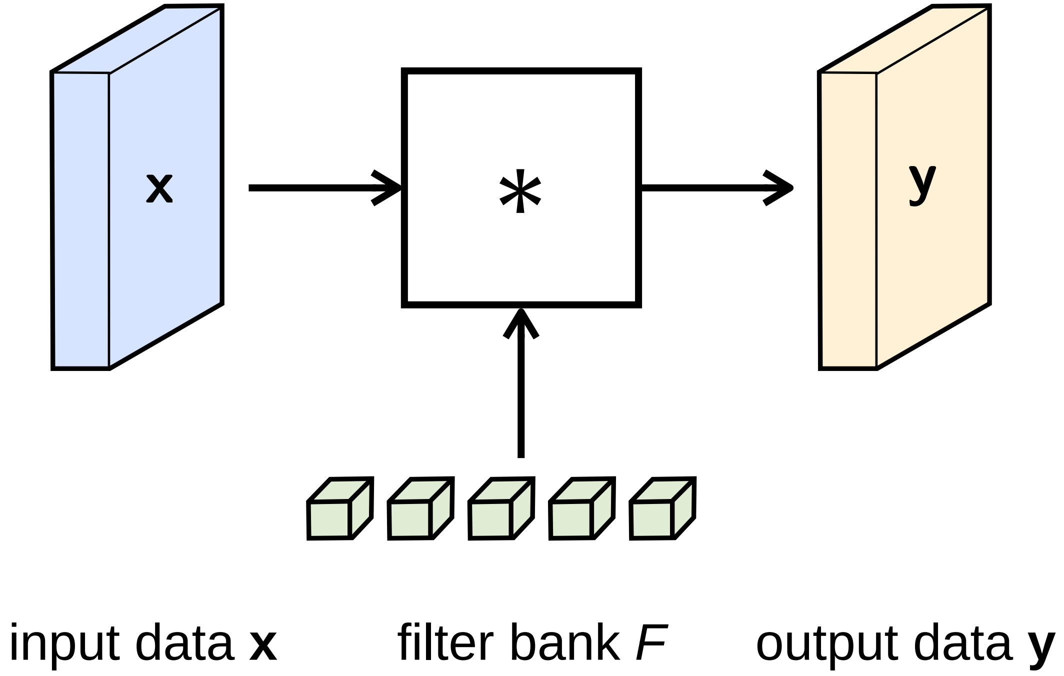


A. Krizhevsky, I. Sutskever, and G. E. Hinton. **Imagenet classification with deep convolutional neural networks**. In NIPS 2012.

Convolutional layer

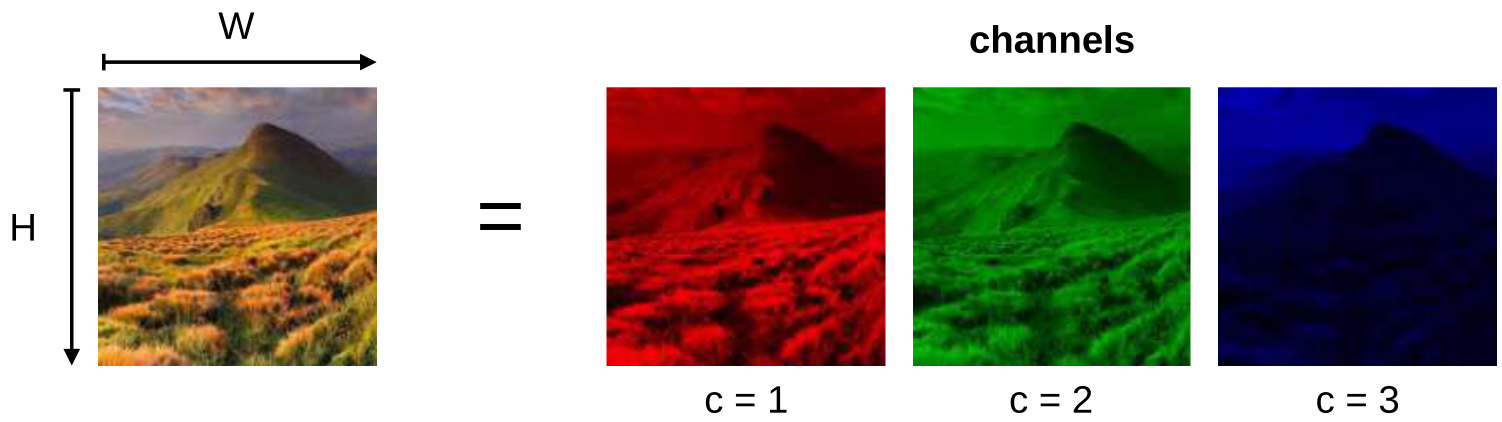
- Learn a filter bank (a set of filters) once
- Use them over the input data to extract features

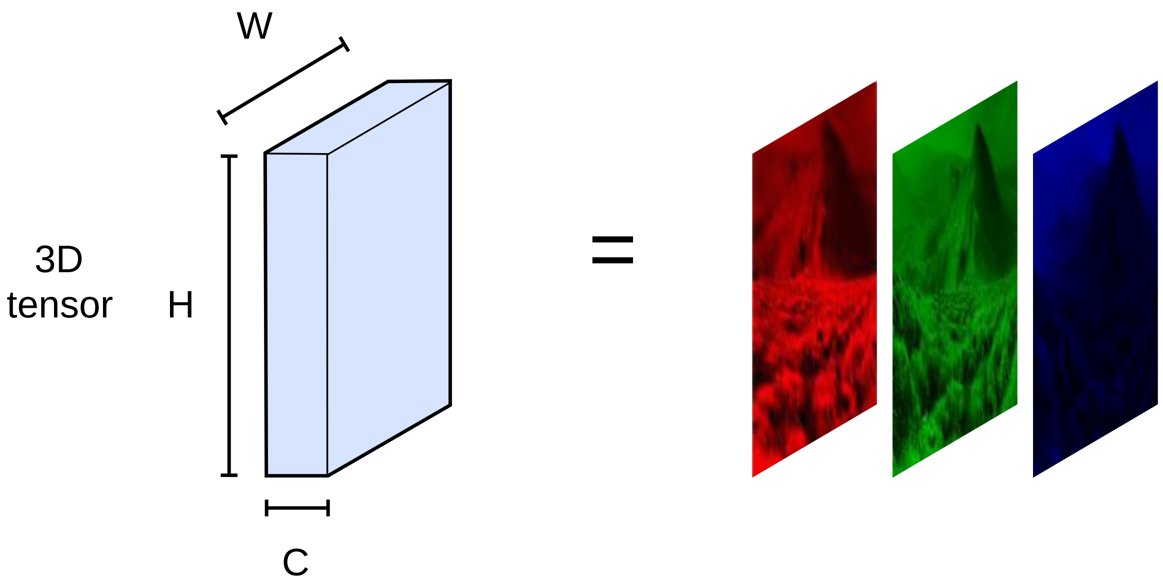
$$\mathbf{y} = F * \mathbf{x} + b$$



Data = 3D Tensors

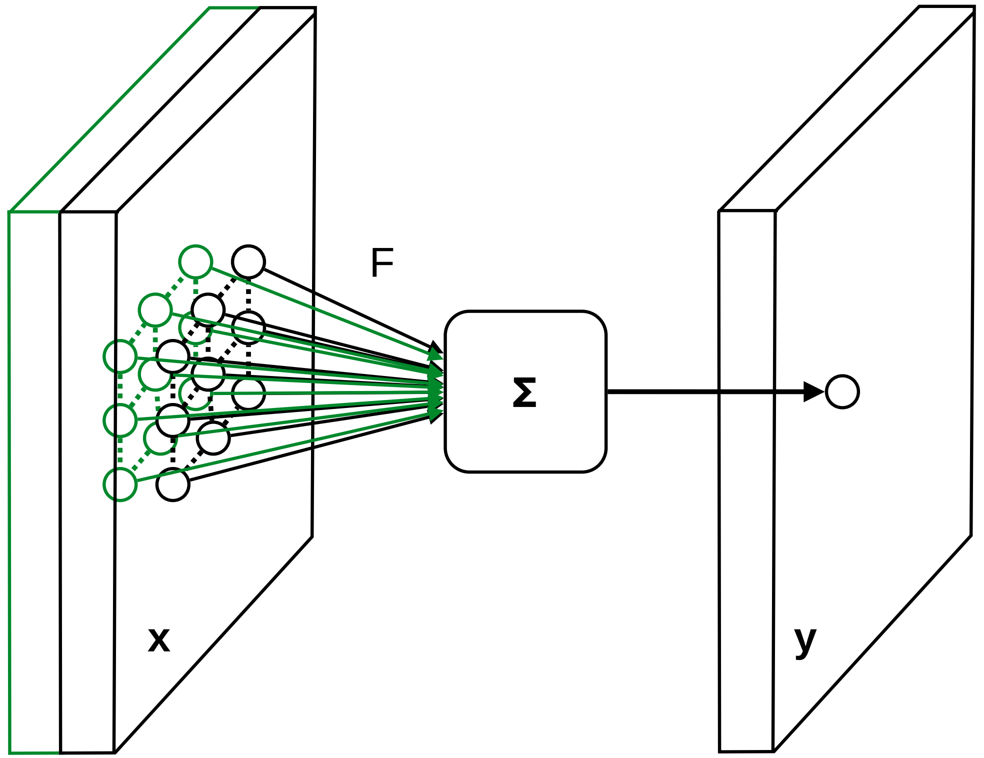
- There is a vector of feature channels (e.g. RGB) at each spatial location (pixel).

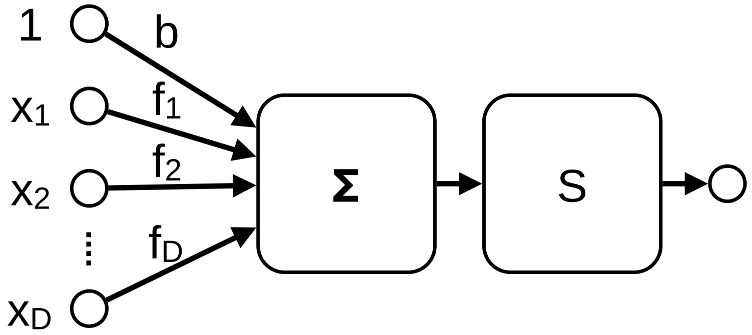


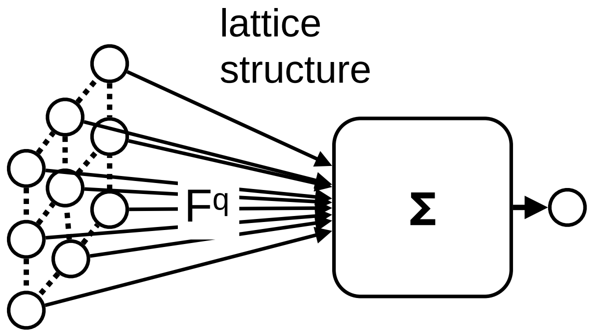


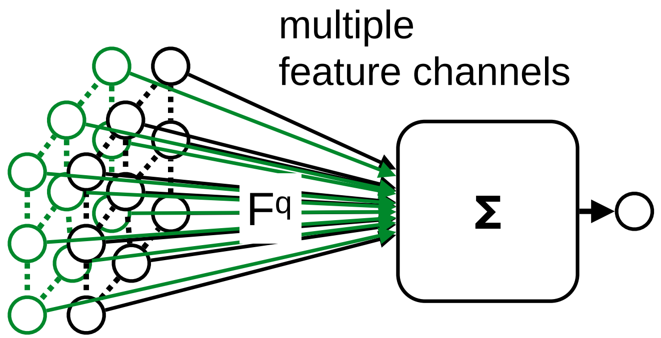
Convolutions with 3D Filters

- Each filter acts on multiple input channels
  - **Local**
    Filters look locally
  - **Translation invariant**
    Filters act the same everywhere



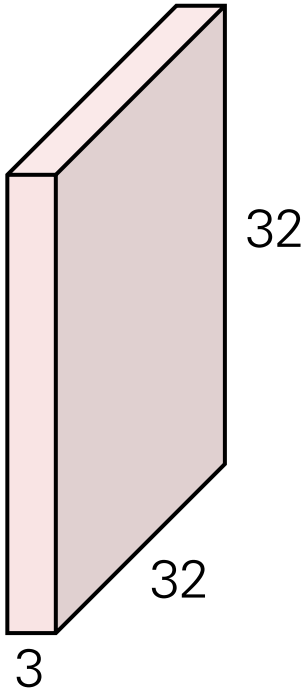




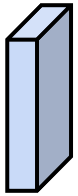


Convolutional Layer

32x32x3 input

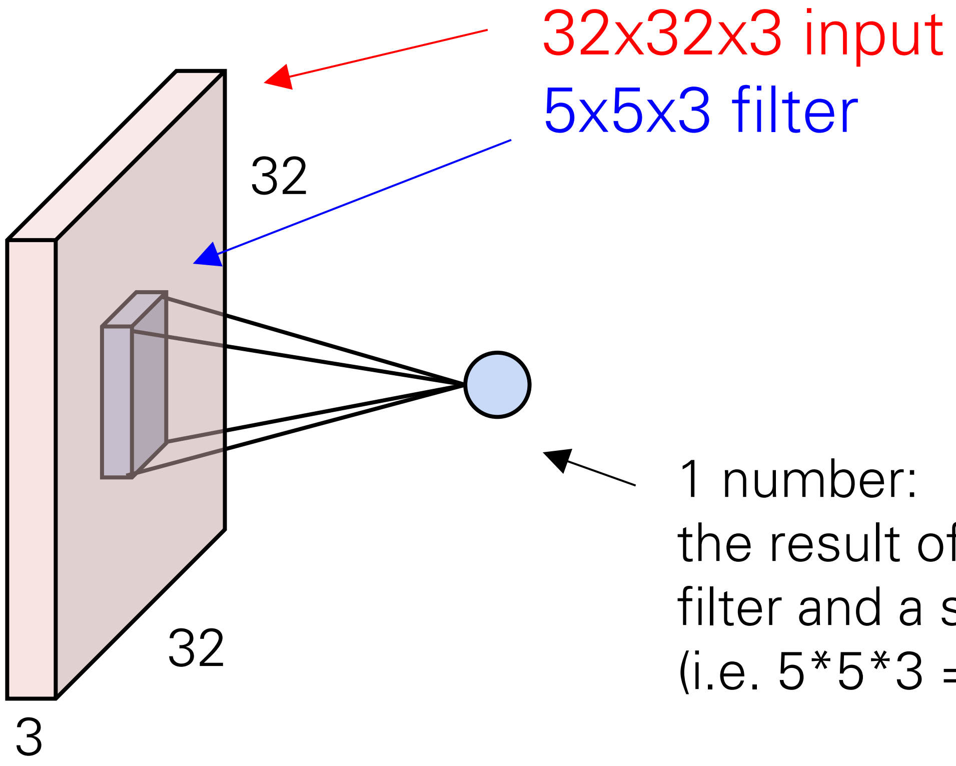


5x5x3 filter



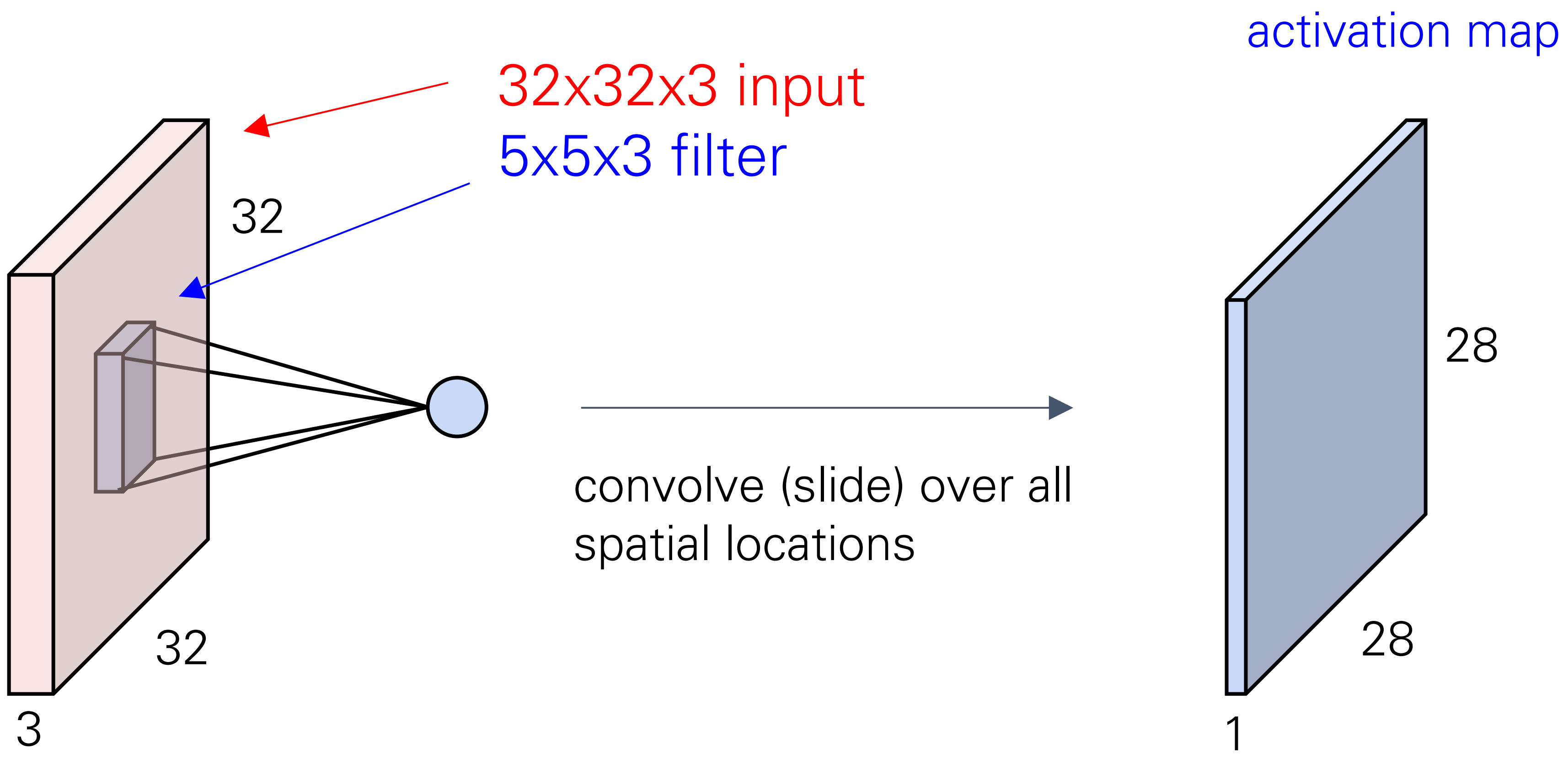
Convolve the filter with the input
i.e. “slide over the image spatially,
computing dot products”

Convolutional Layer



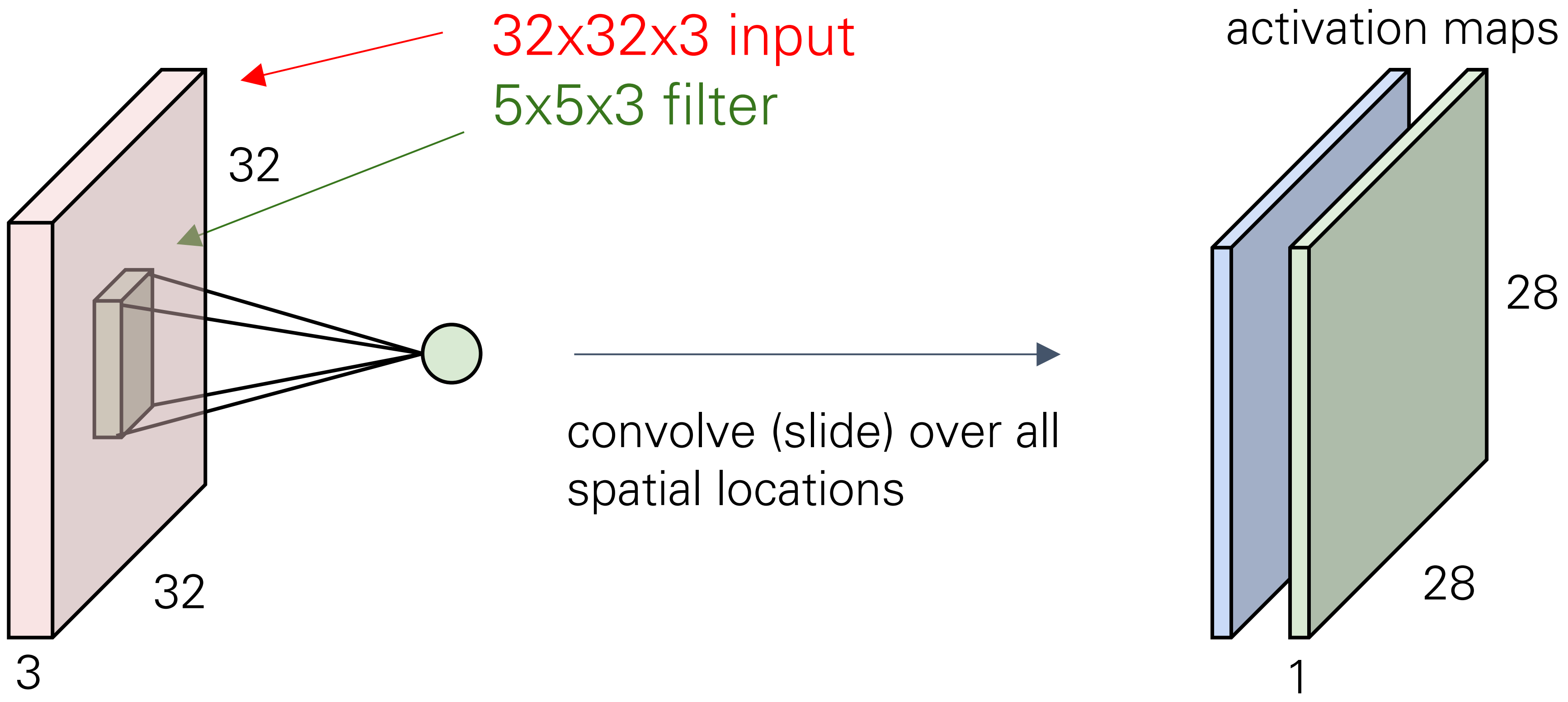
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the input
(i.e. 5*5*3 = 75-dimensional dot product + bias)

Convolutional Layer



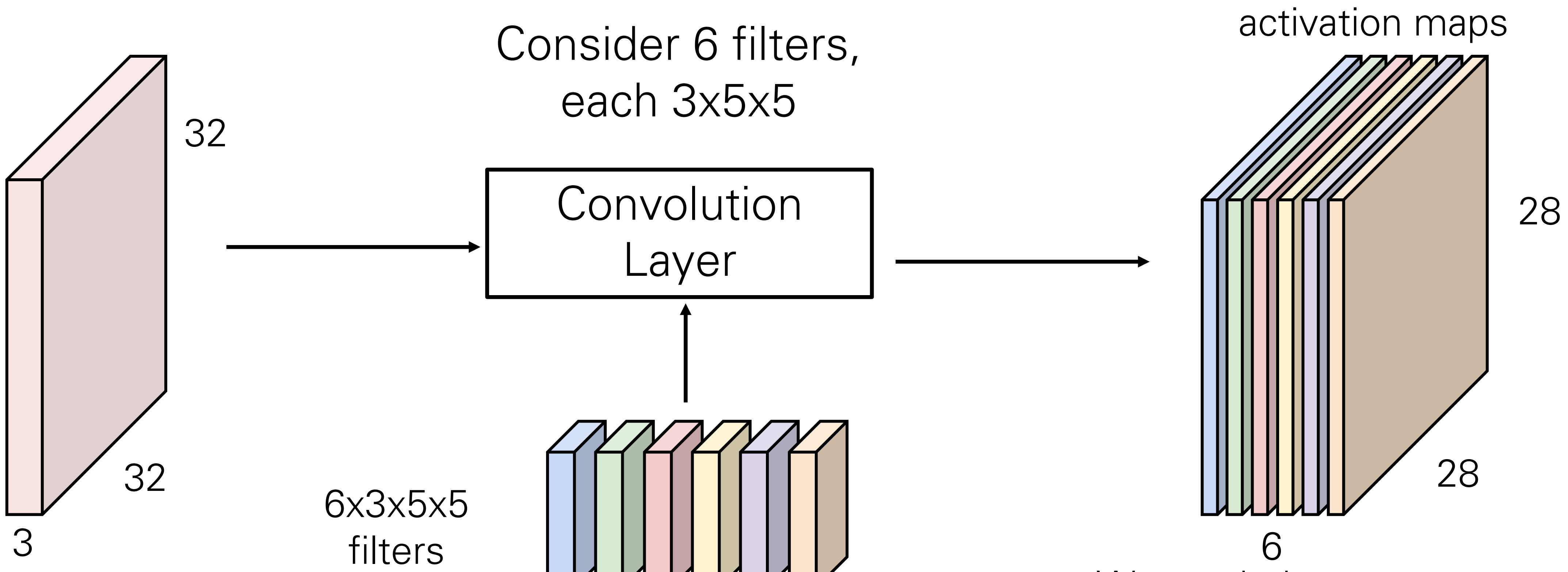
Convolutional Layer

consider a second, green filter



Convolutional Layer

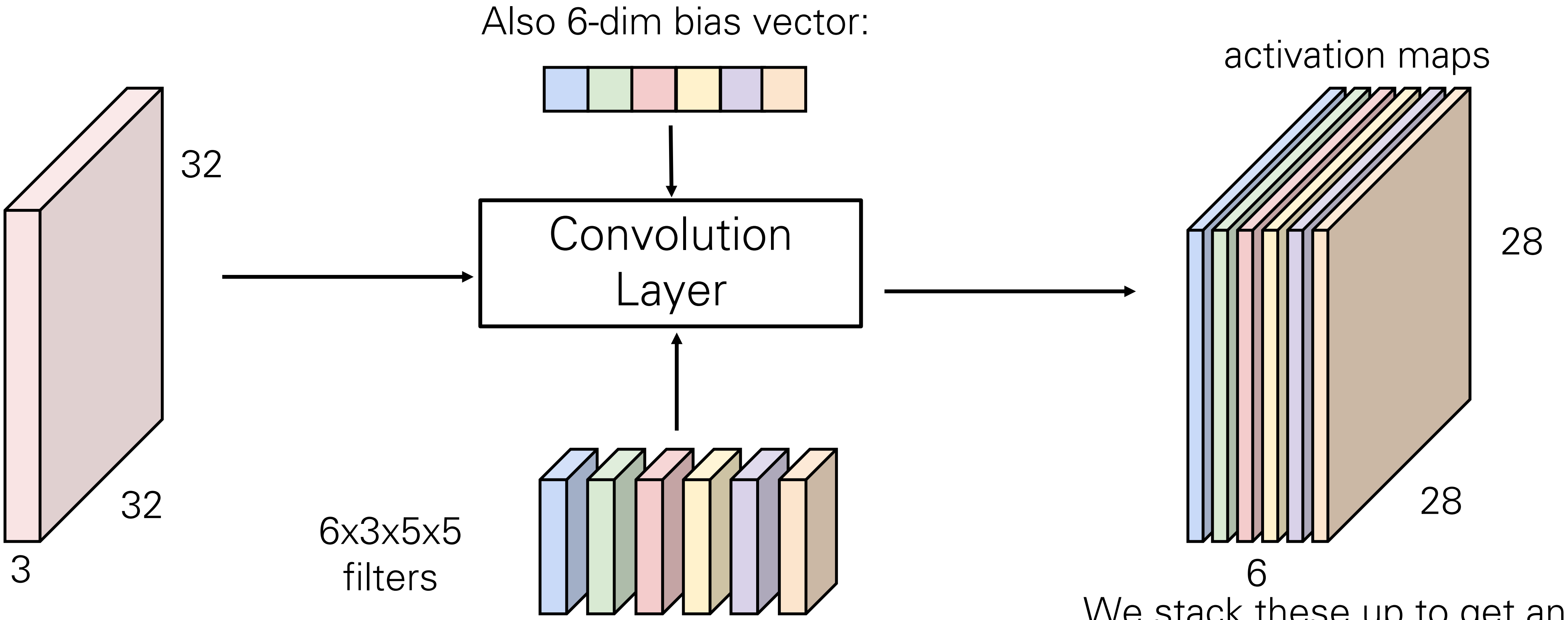
- Multiple filters produce multiple output channels



We stack these up to get an output of size 28x28x6.

Convolutional Layer

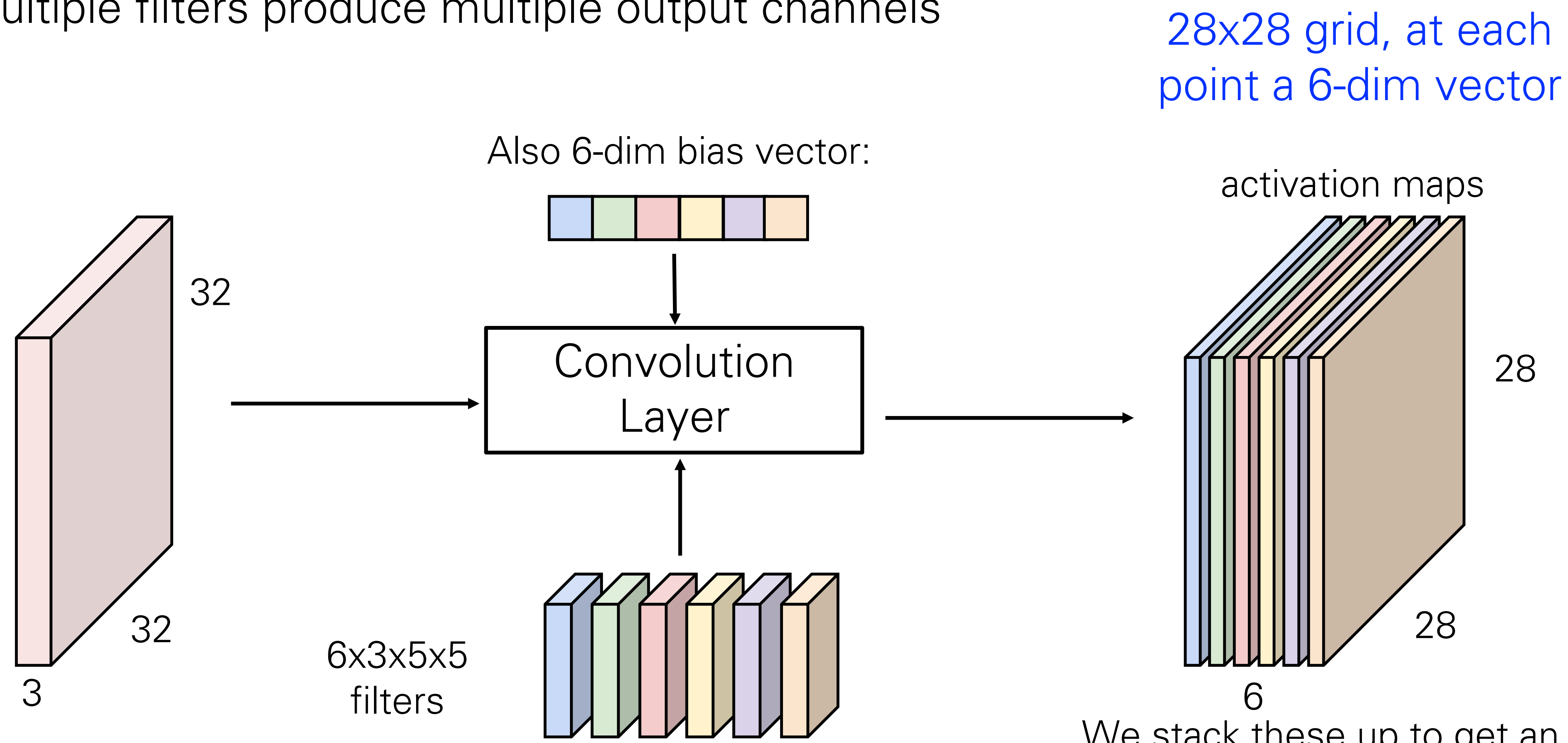
- Multiple filters produce multiple output channels



We stack these up to get an output of size 28x28x6.

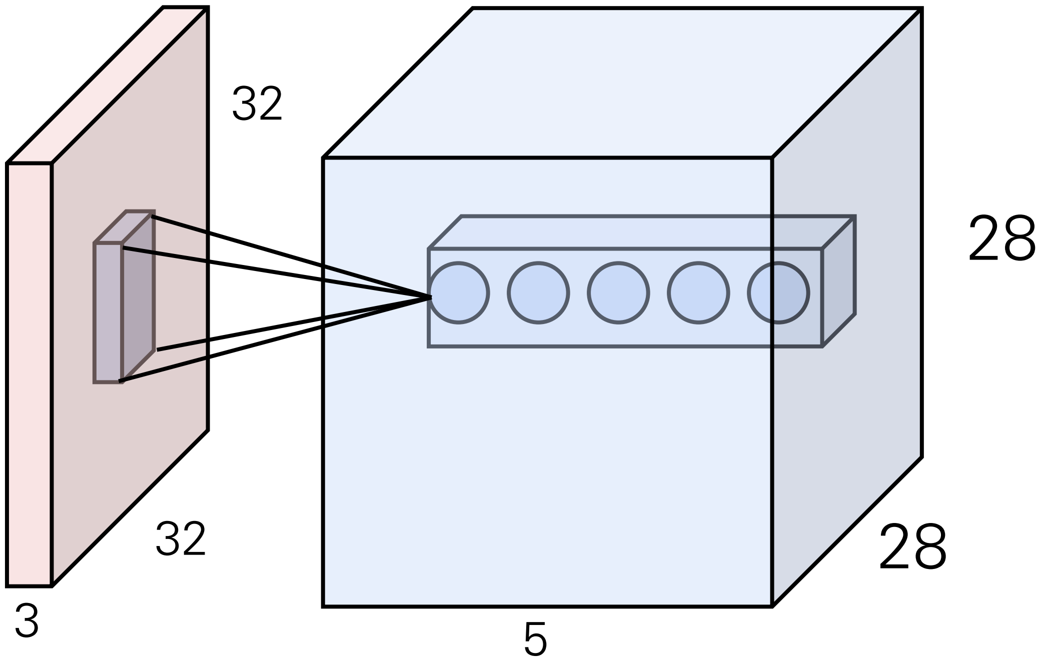
Convolutional Layer

- Multiple filters produce multiple output channels

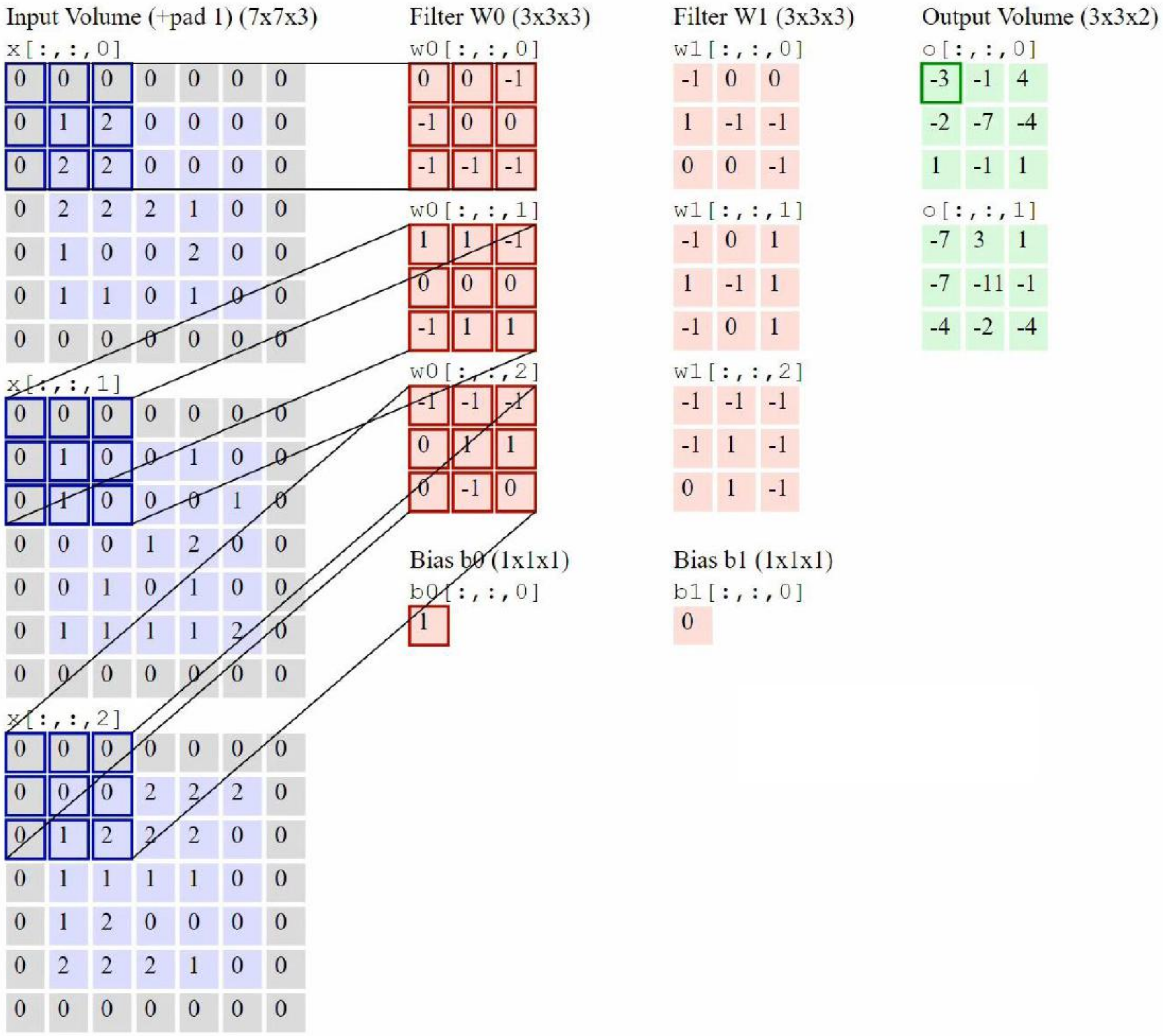


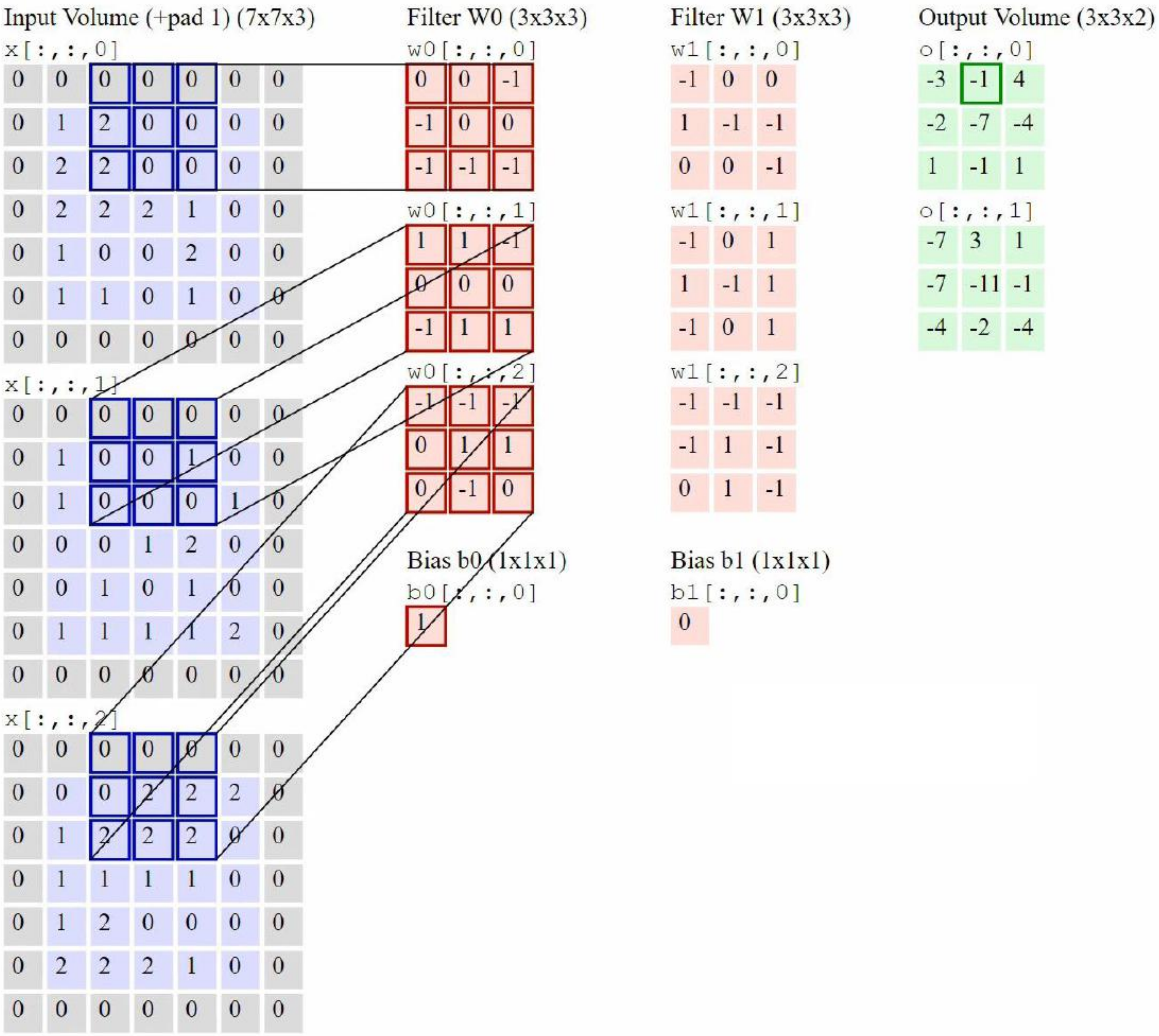
We stack these up to get an output of size 28x28x6.

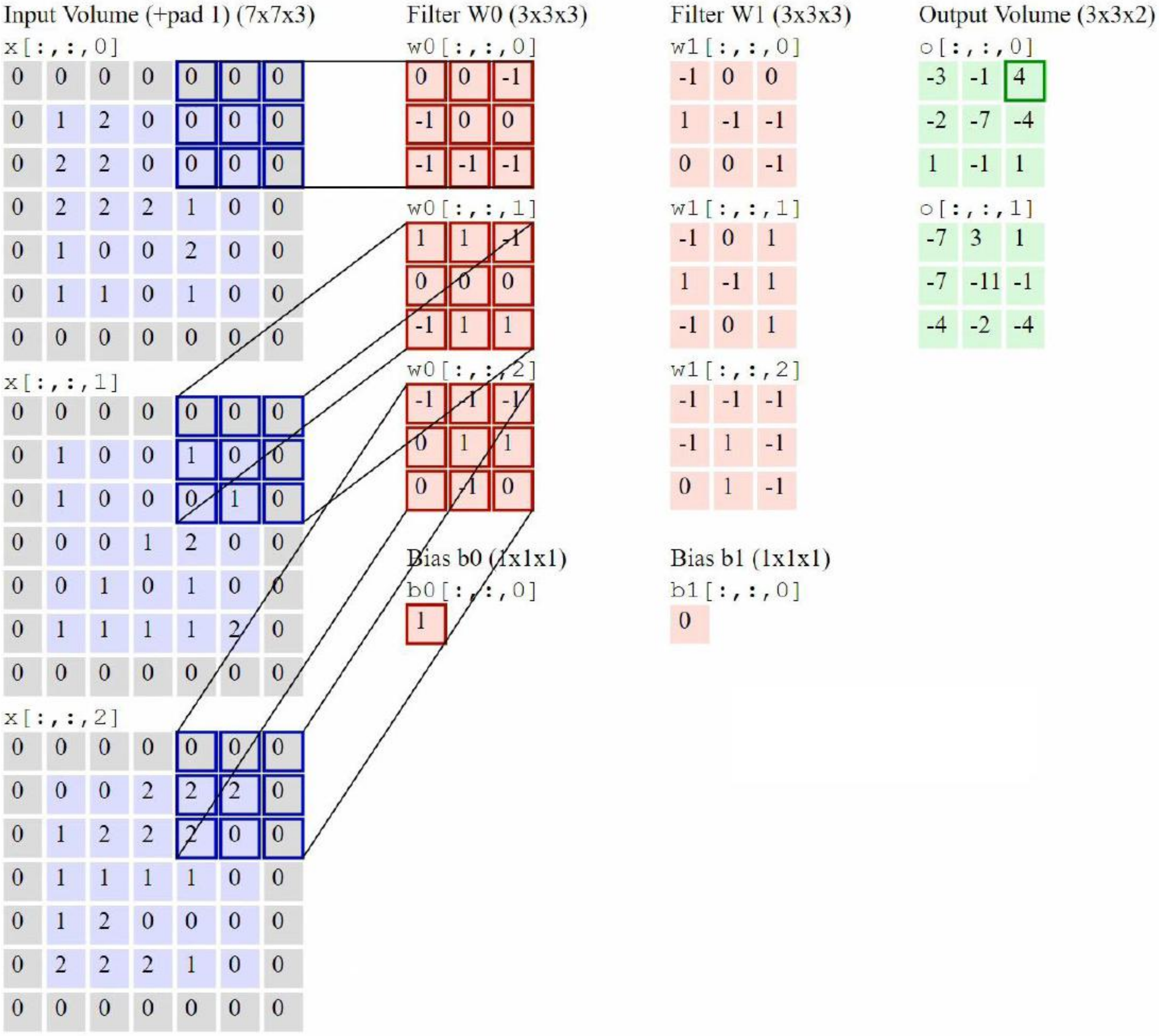
Spatial Arrangement of Output Volume

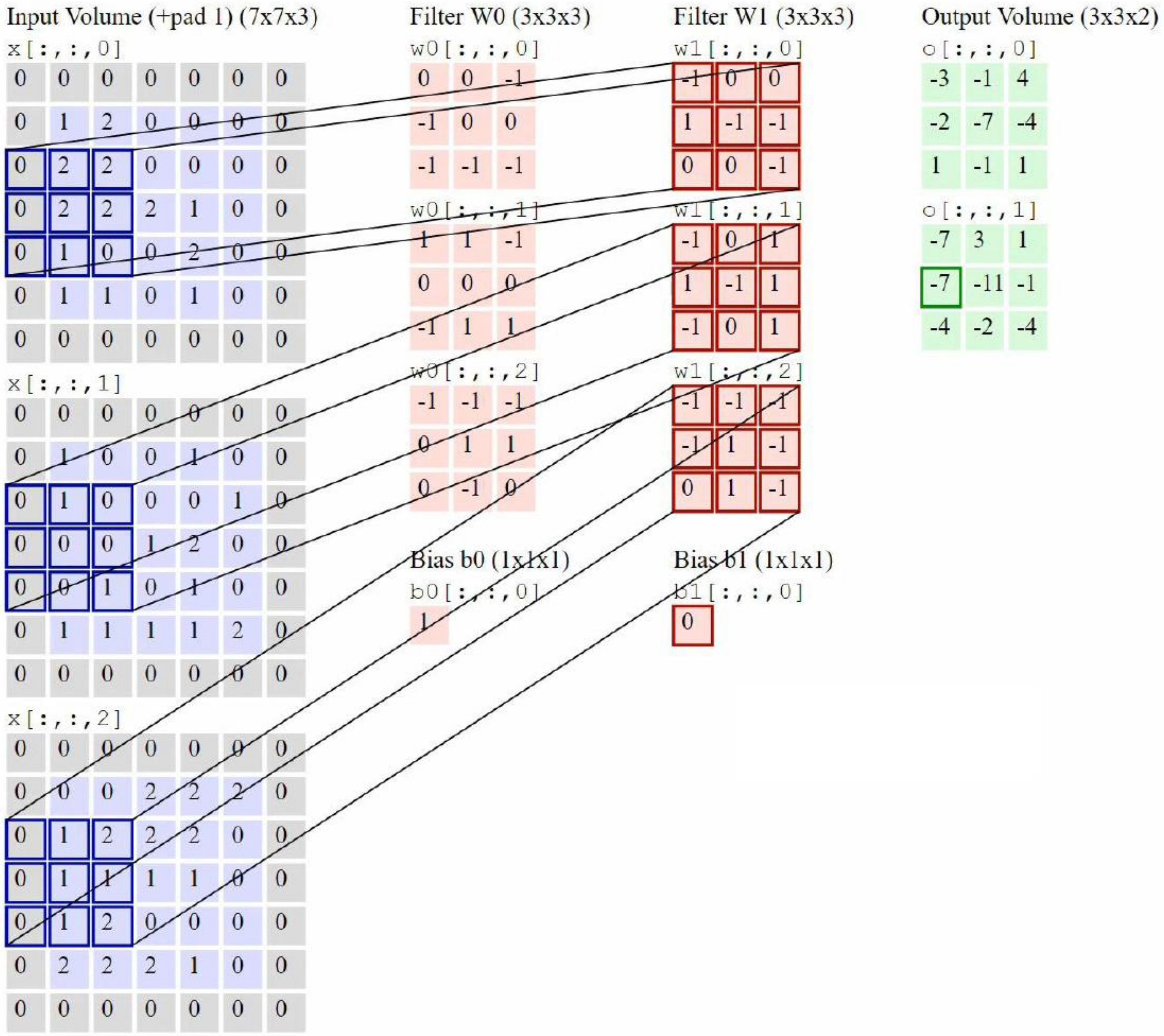


- **Depth:** number of filters
- **Stride:** filter step size (when we “slide” it)
- **Padding:** zero-pad the input

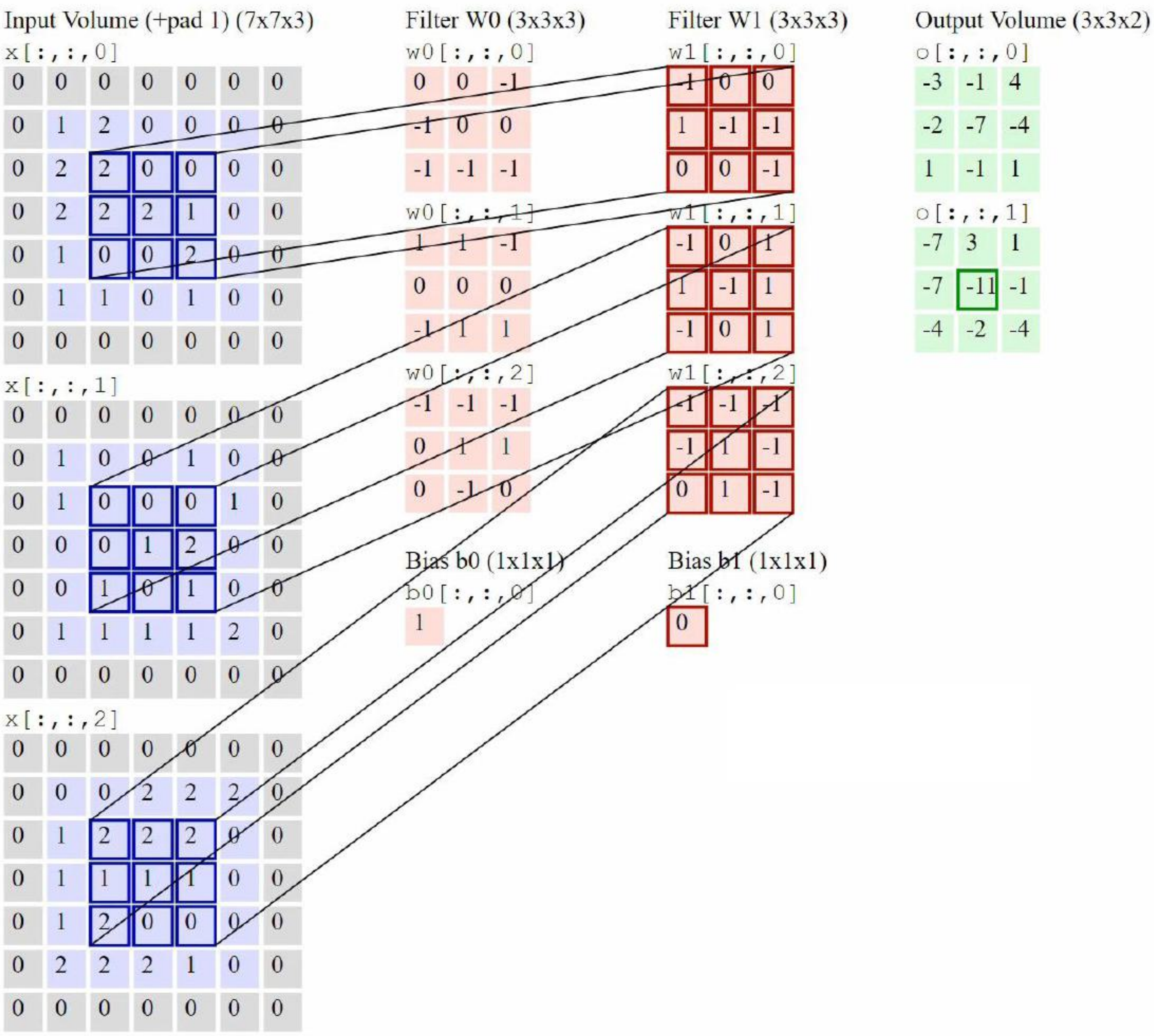

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

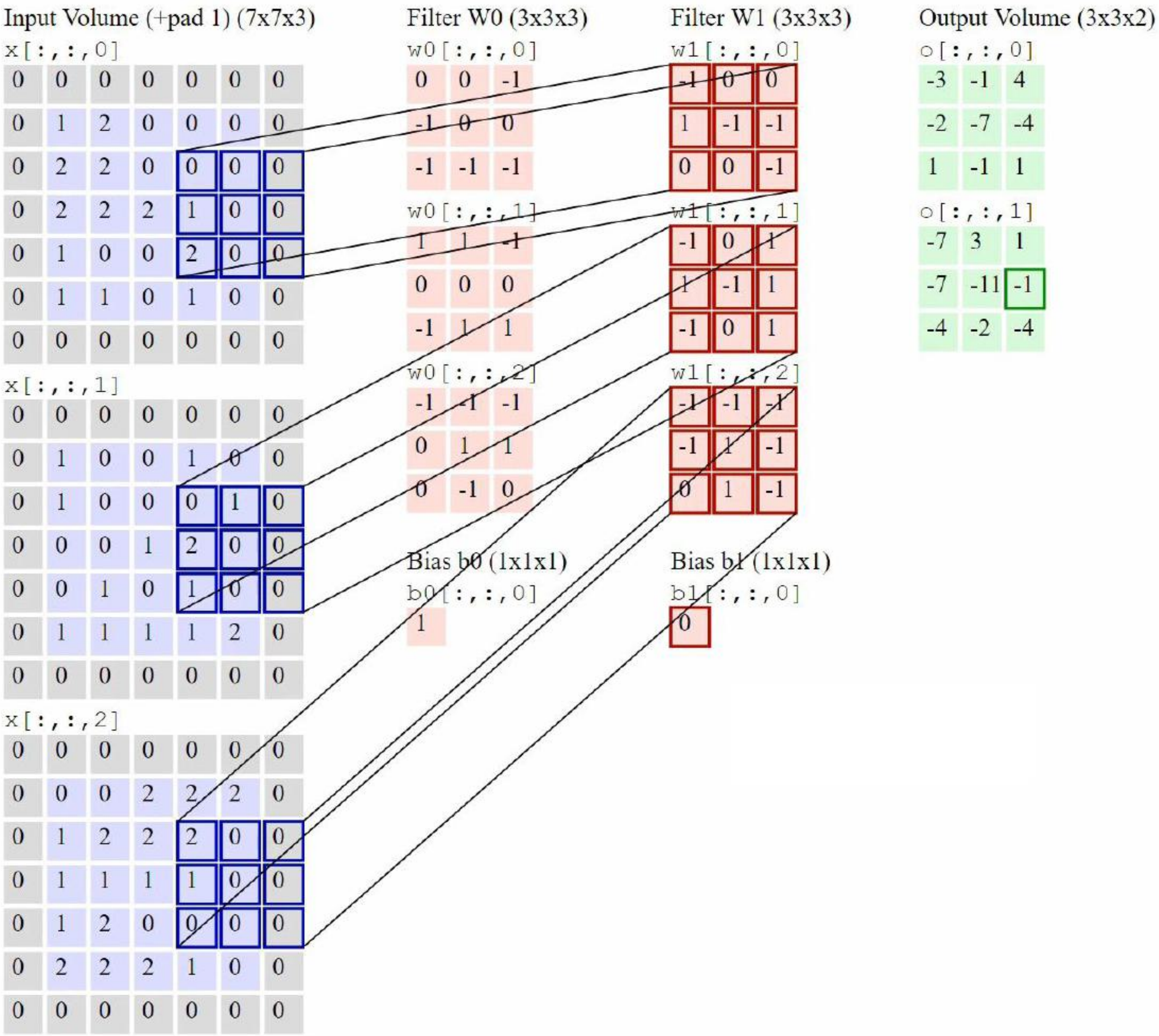

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

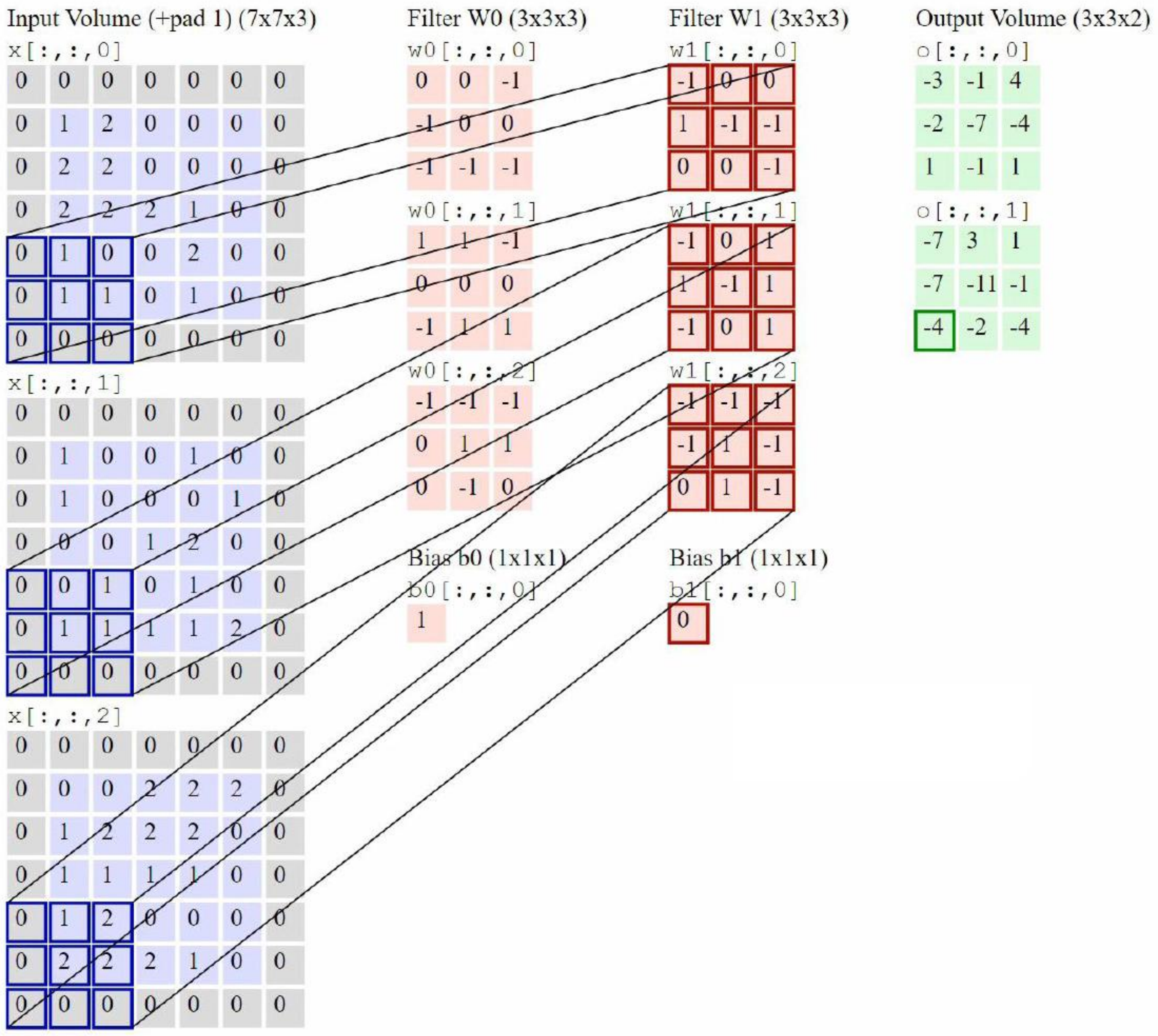

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4



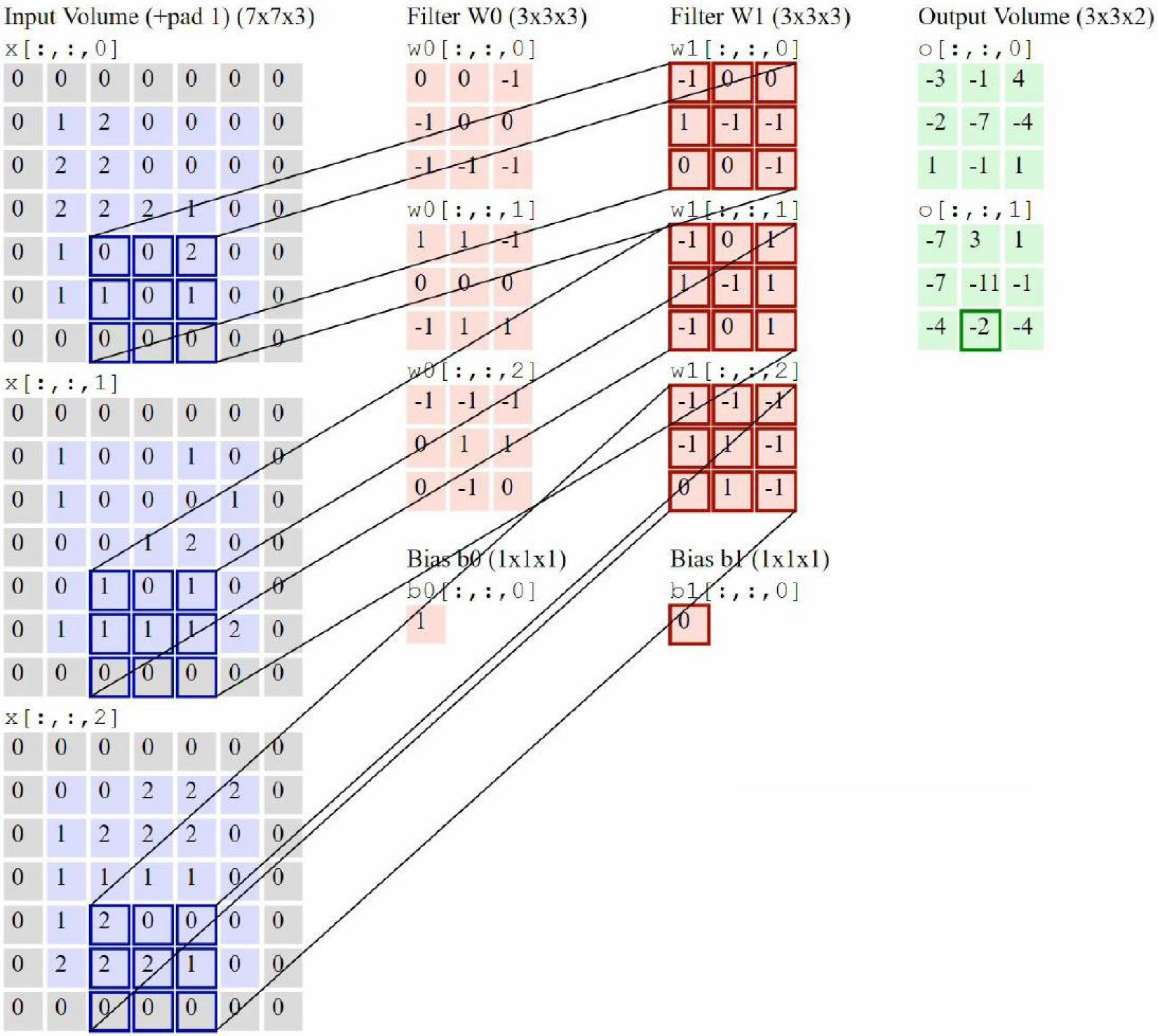
Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

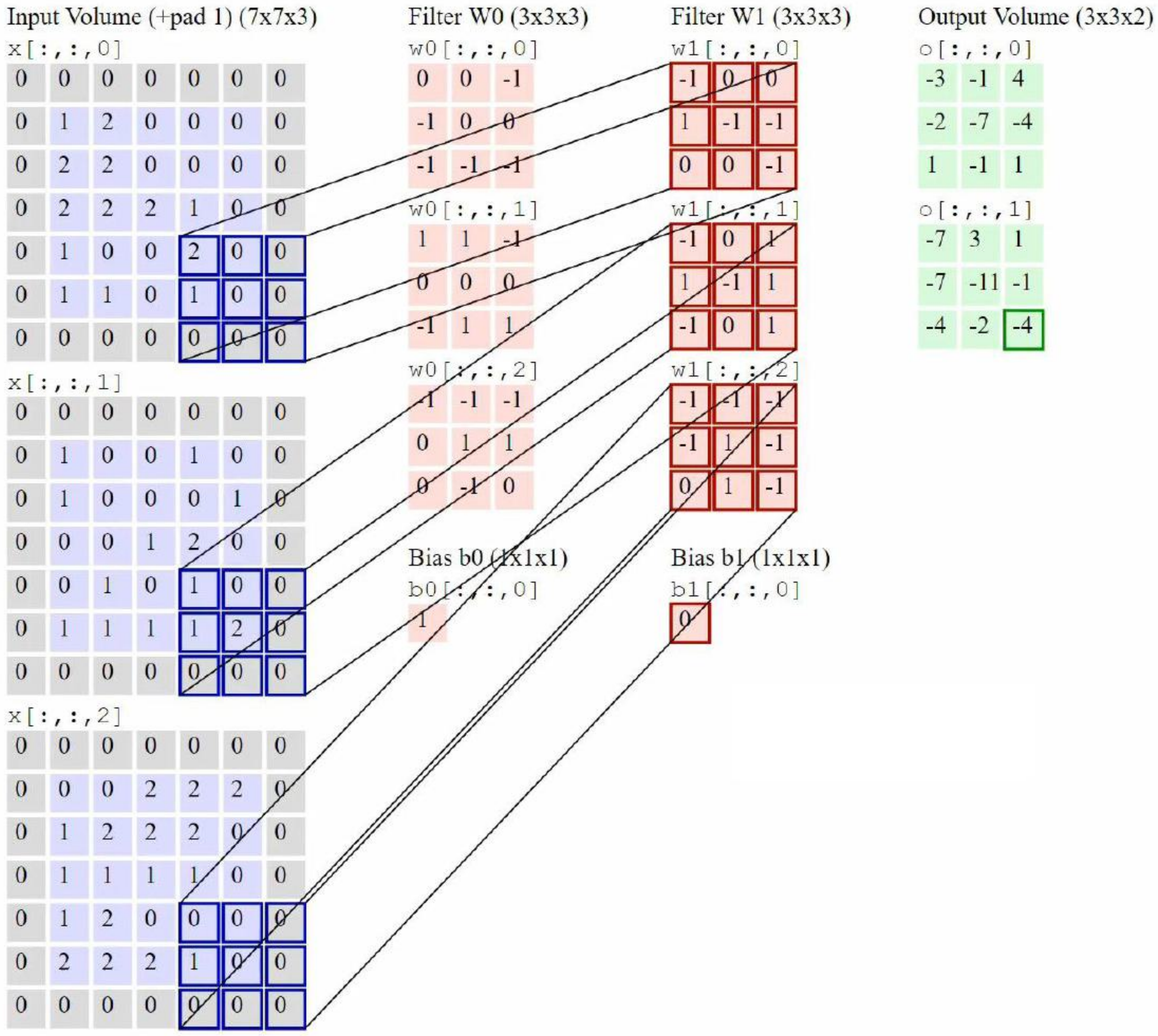

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
x[:,:,1]
x[:,:,2]
Filter W0 (3x3x3)
w0[:,:,0]
w0[:,:,1]
w0[:,:,2]
Bias b0 (1x1x1)
b0[:,:,0]
Filter W1 (3x3x3)
w1[:,:,0]
w1[:,:,1]
w1[:,:,2]
Bias b1 (1x1x1)
b1[:,:,0]
Output Volume (3x3x2)
o[:,:,0]
o[:,:,1]


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4



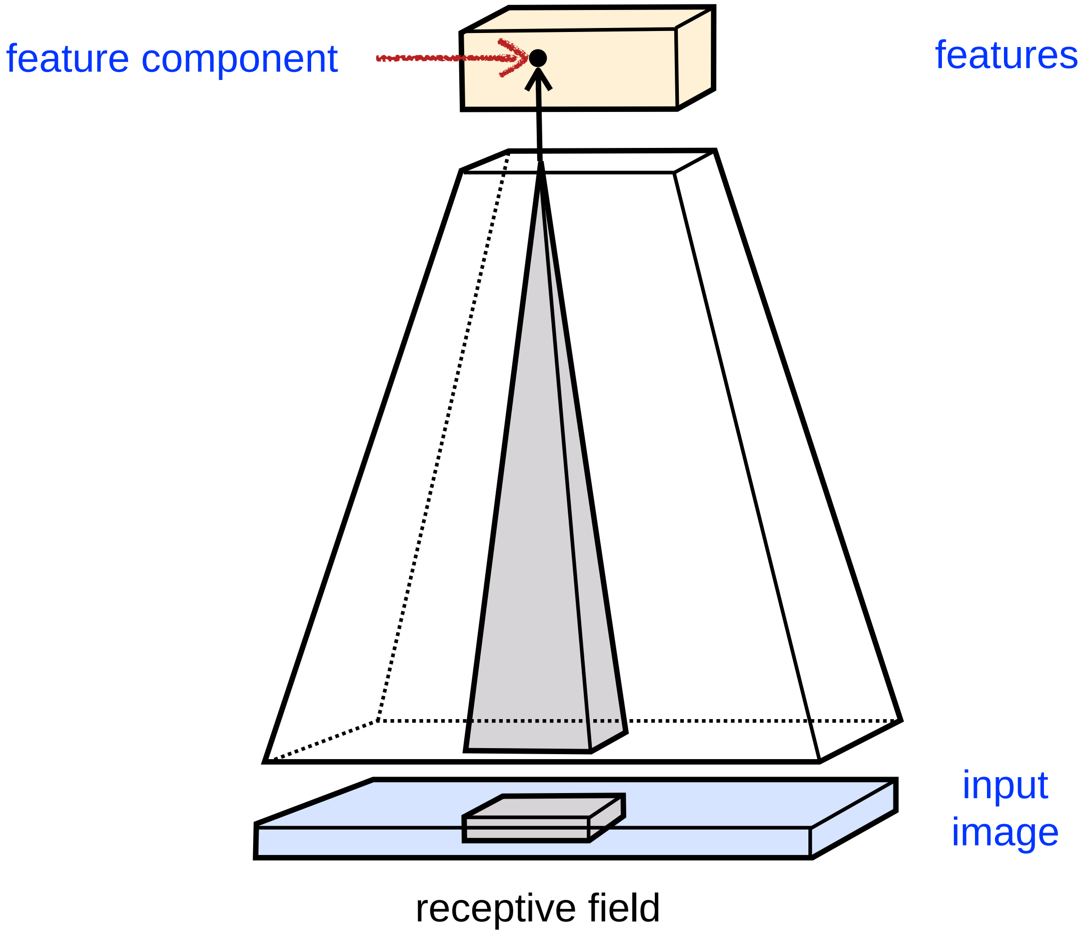
Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

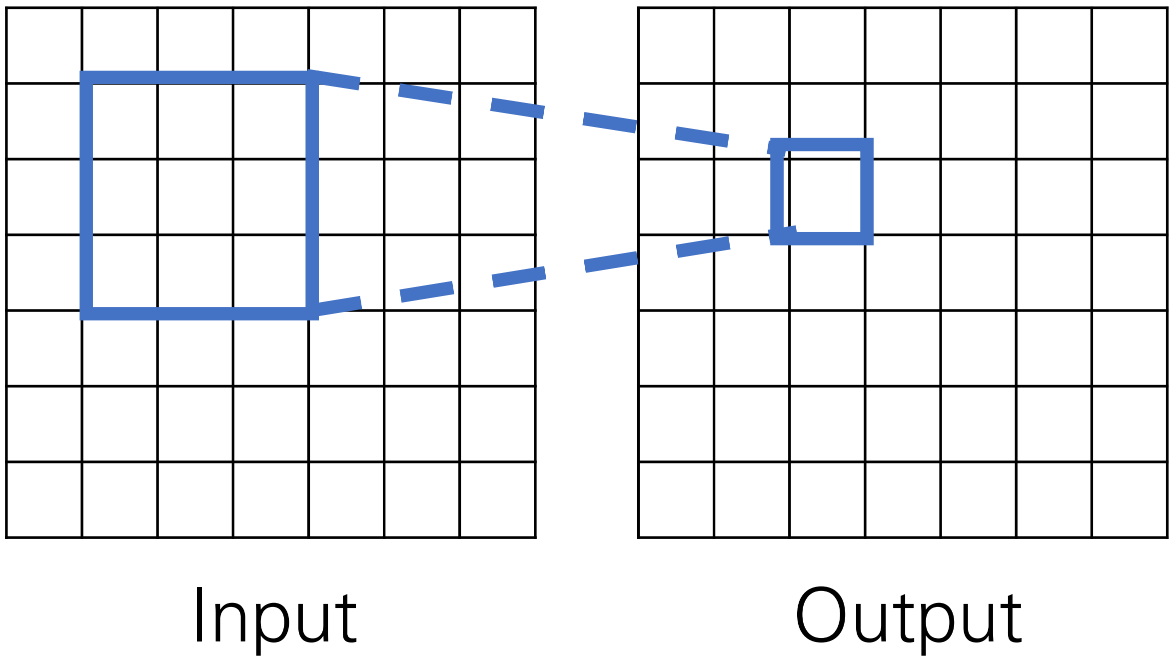
Convolutional layers

- Local receptive field
- Each column of hidden units looks at a different input patch



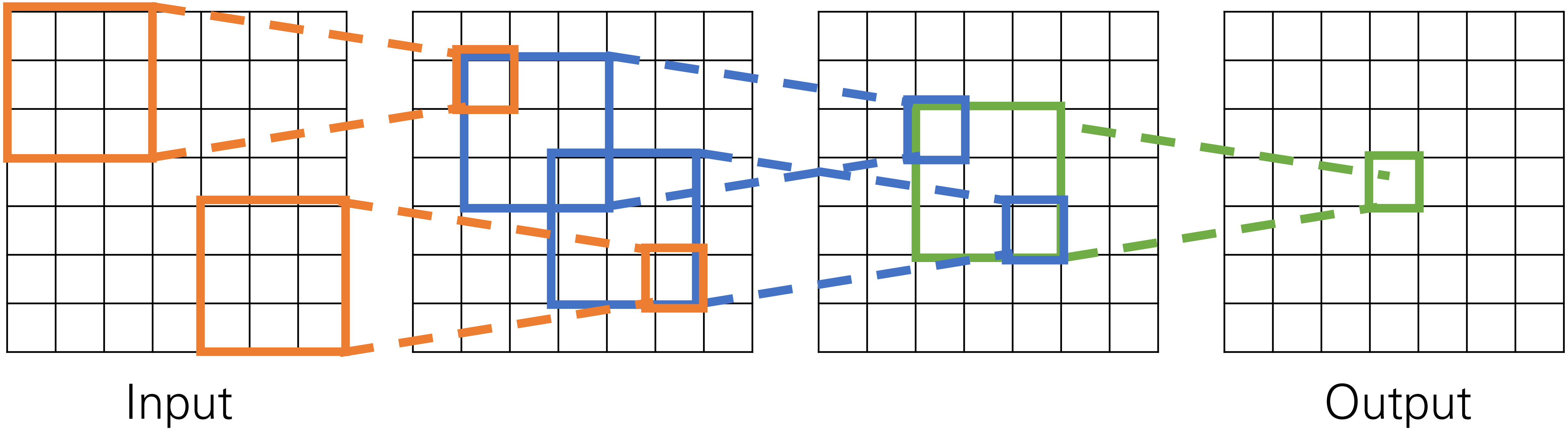
Receptive Fields

For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



Receptive Fields

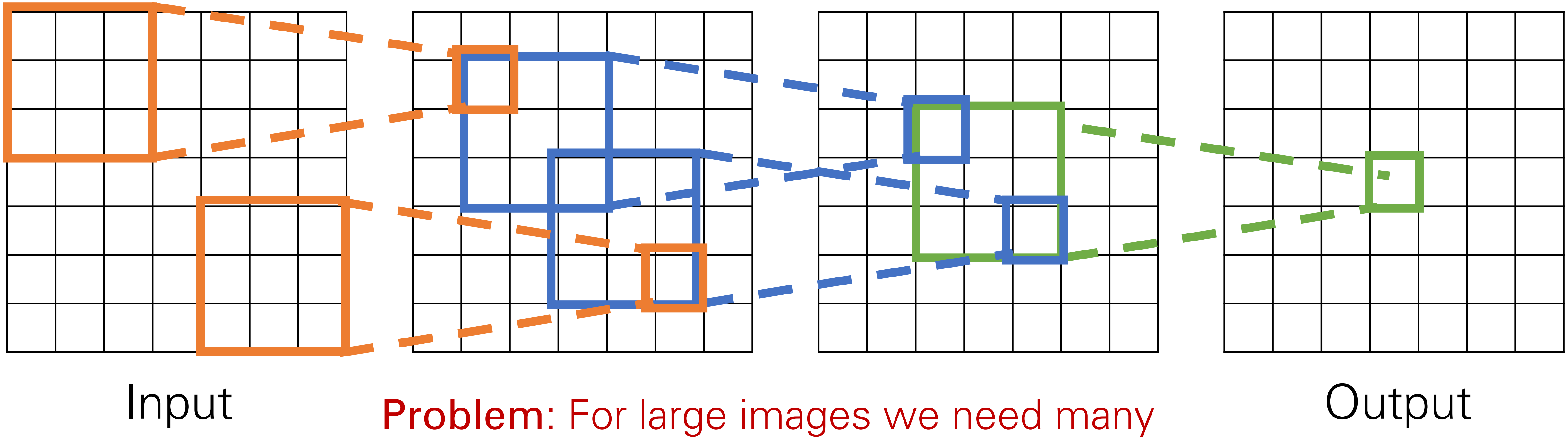
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Be careful – "receptive field in the input" vs "receptive field in the previous layer"
Hopefully clear from context!

Receptive Fields

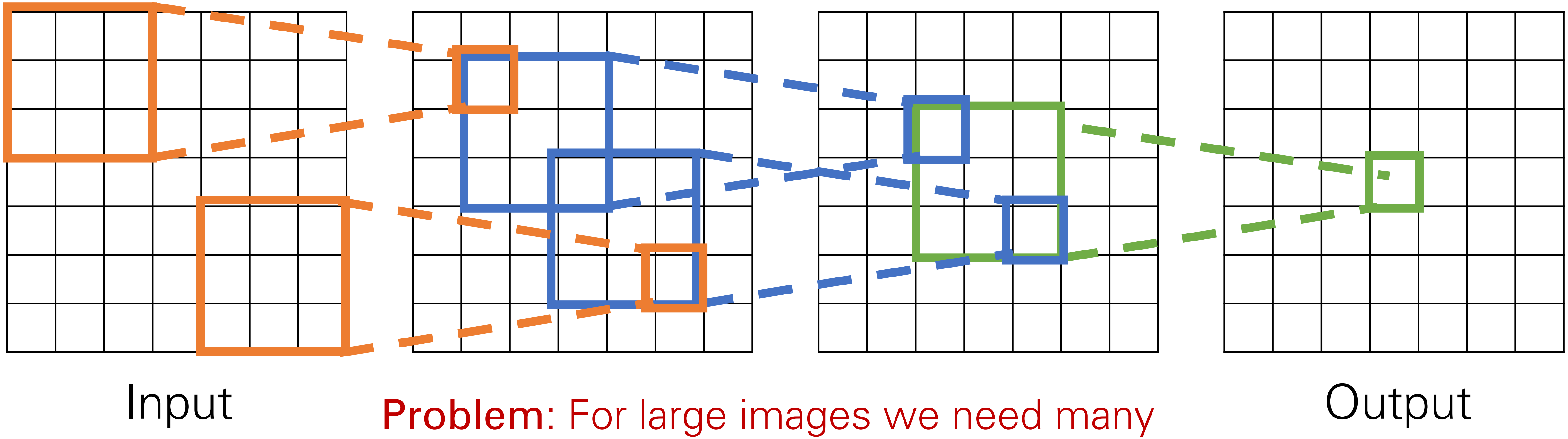
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Problem: For large images we need many layers for each output to “see” the whole image image

Receptive Fields

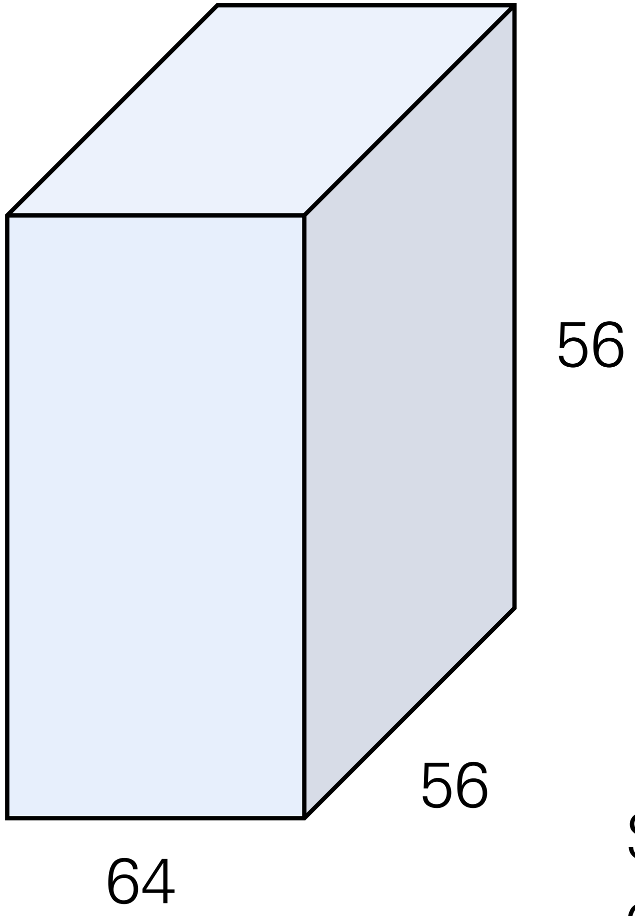
Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)



Problem: For large images we need many layers for each output to "see" the whole image image

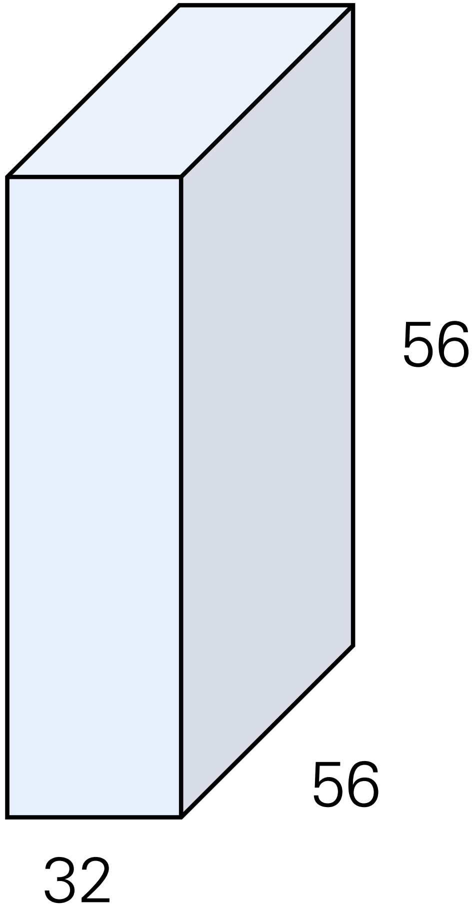
Solution: Downsample inside the network

1x1 Convolution



1x1 CONV
with 32 filters

(each filter has size 64x1x1, and performs a 64-dimensional dot product)



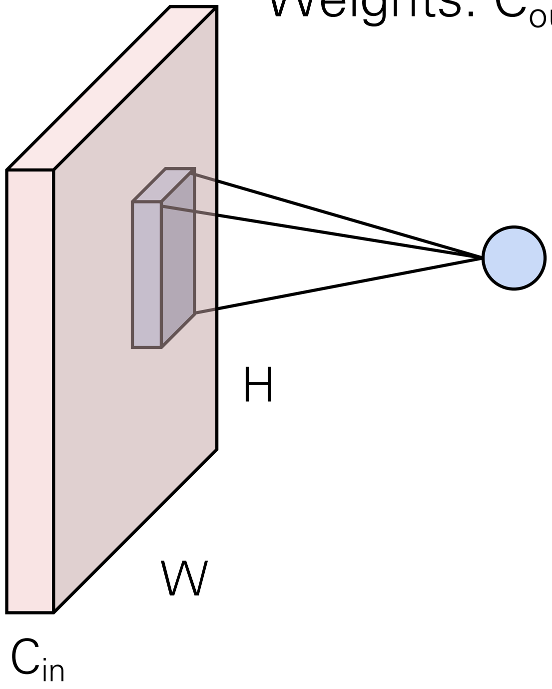
Stacking 1x1 conv layers gives MLP operating on each input position

Lin et al, "Network in Network", ICLR 2014

Other types of convolution

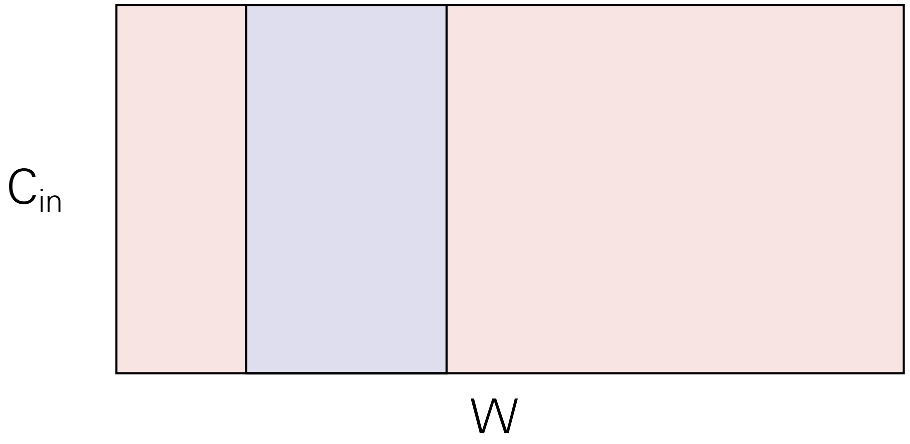
So far: 2D Convolution

Input: C_{in} x H x W
Weights: C_{out} x C_{in} x K x K



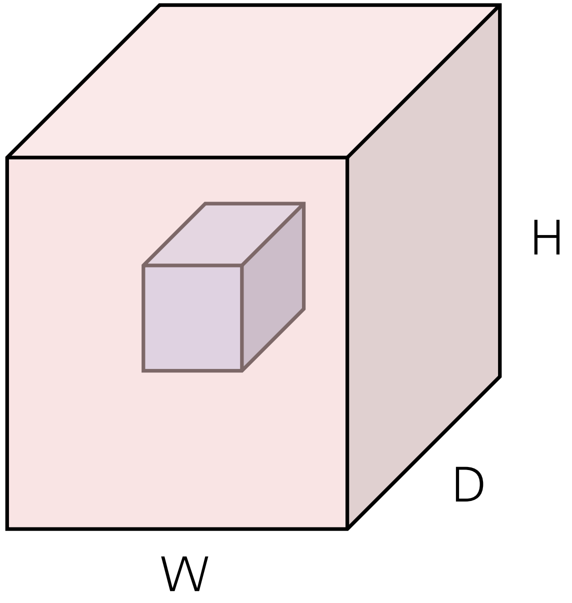
1D Convolution

Input: C_{in} x W
Weights: C_{out} x C_{in} x K



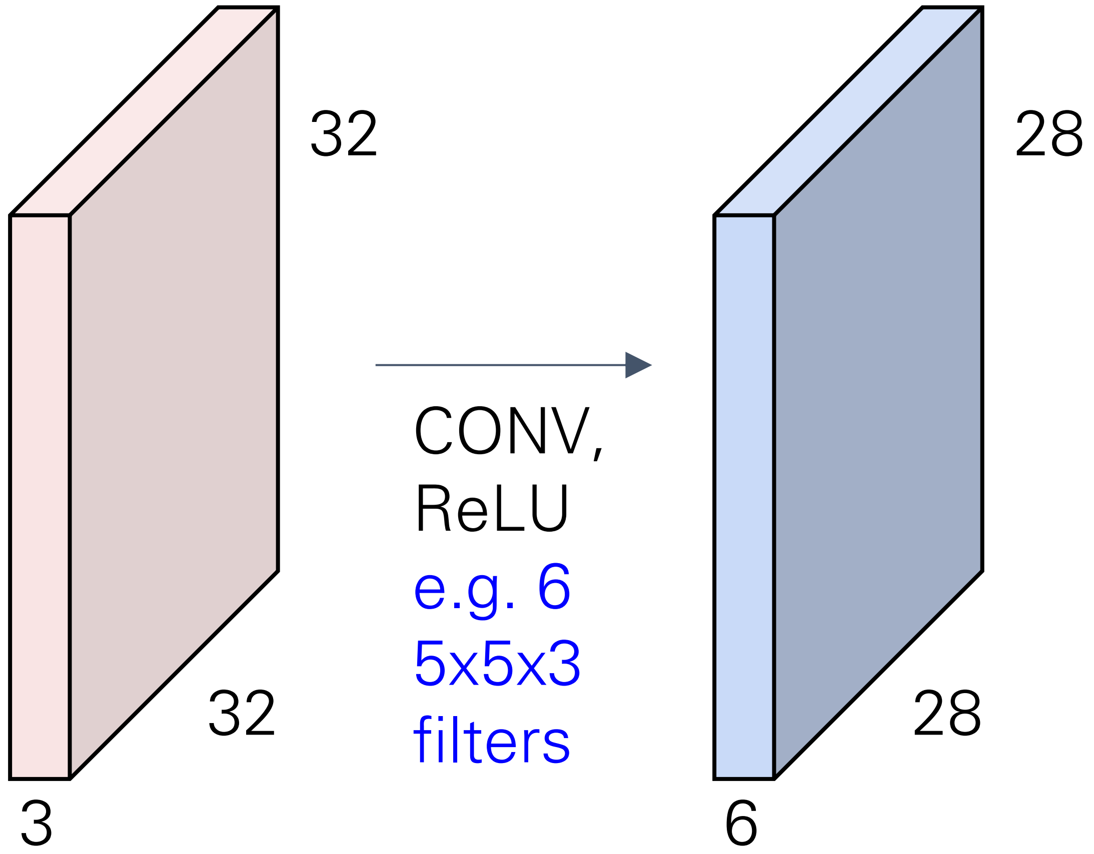
3D Convolution

Input: C_{in} x H x W x D
Weights: C_{out} x C_{in} x K x K x K

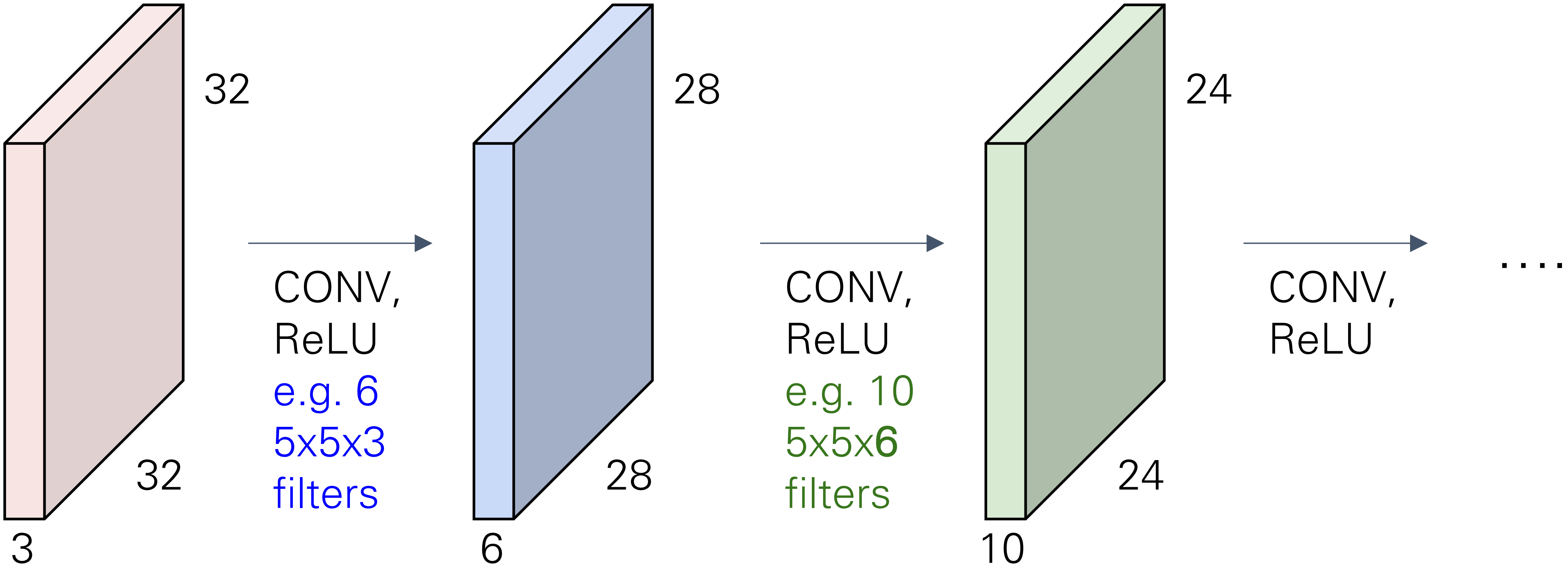


C_{in}-dim vector at each point in the volume

Convolutional layers

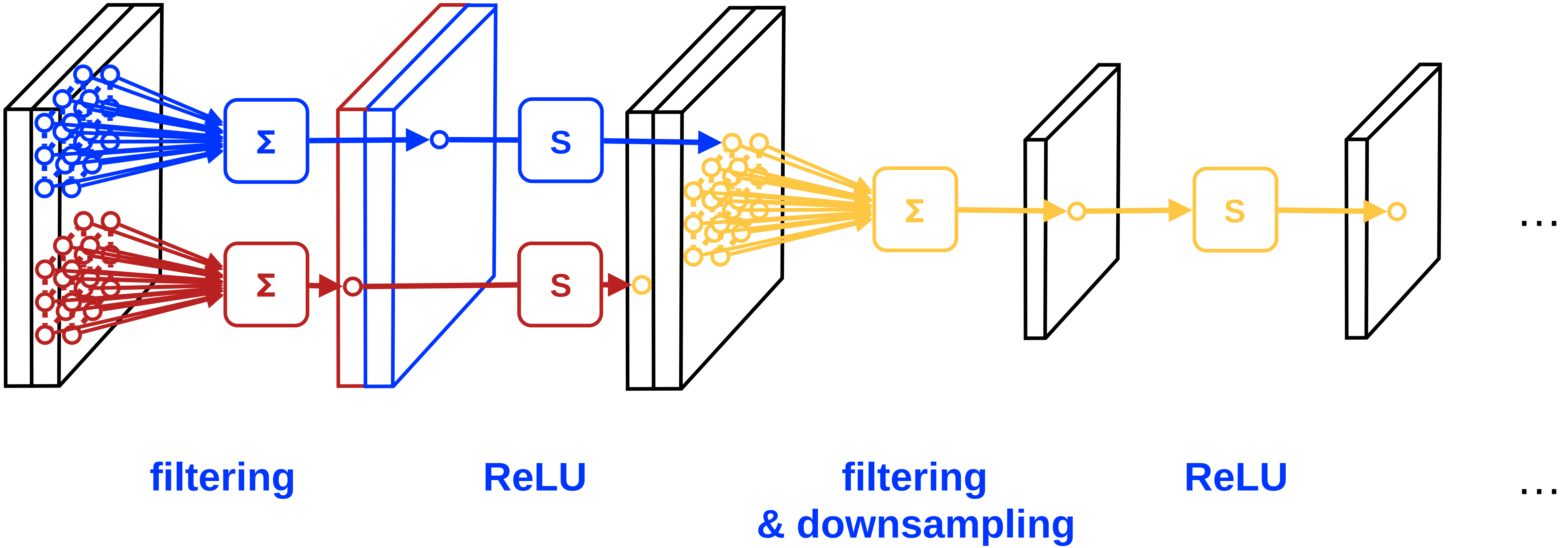


Stacking Convolutions



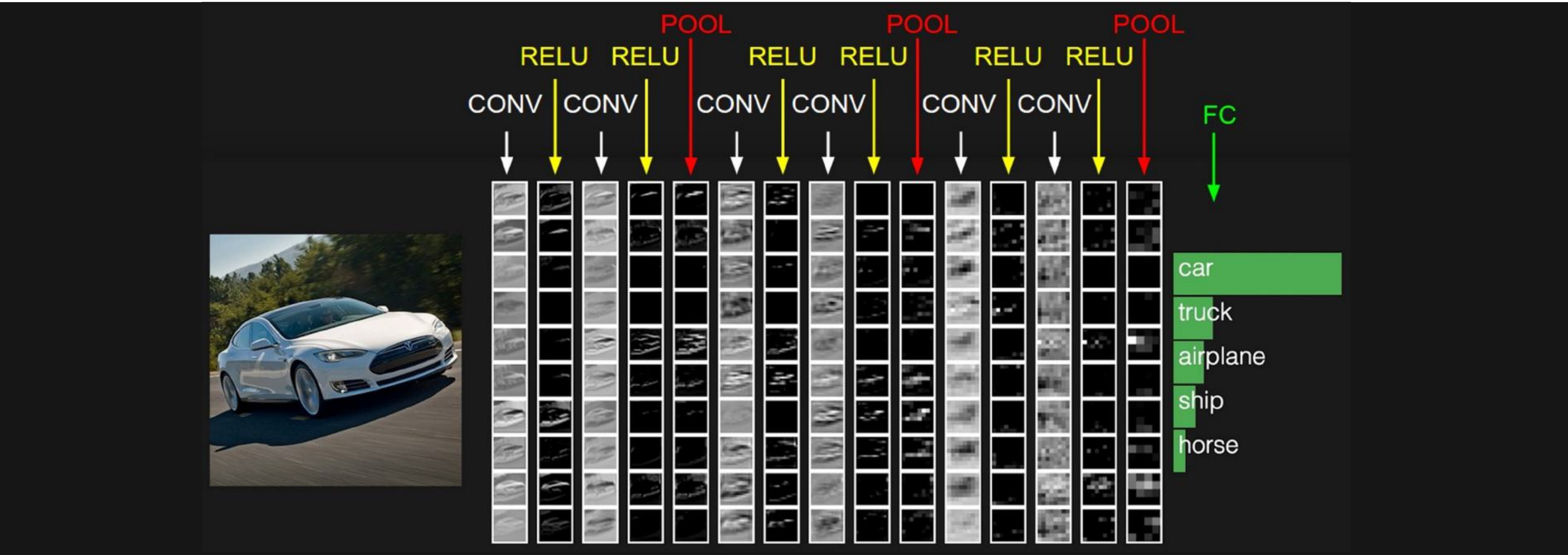
Linear/Non-linear Chains

- The basic blueprint of most architectures
- Stack multiple layers of convolutions



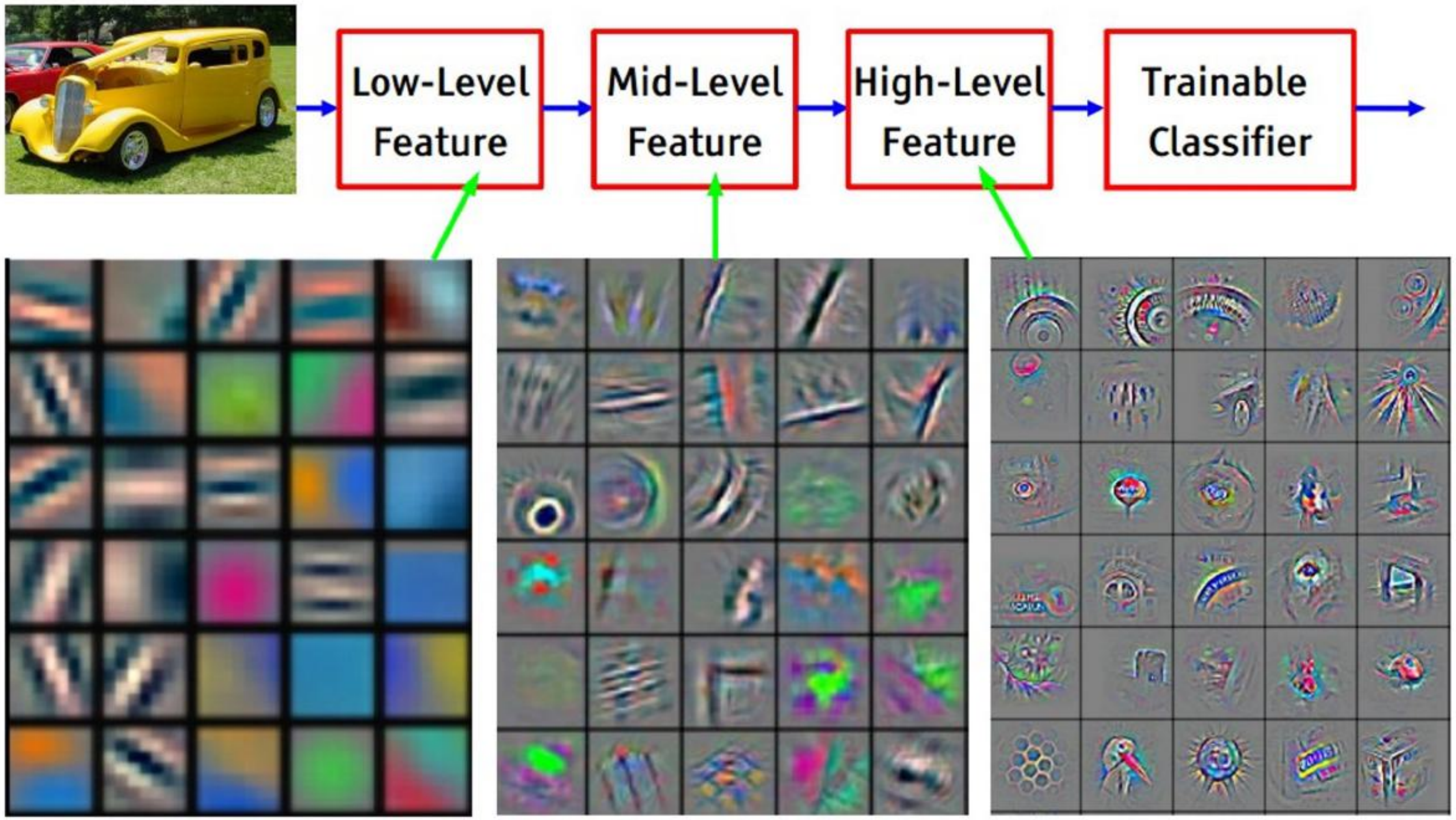
Feature Learning

- Hierarchical layer structure allows to learn hierarchical filters (features).



Feature Learning

- Hierarchical layer structure allows to learn hierarchical filters (features).

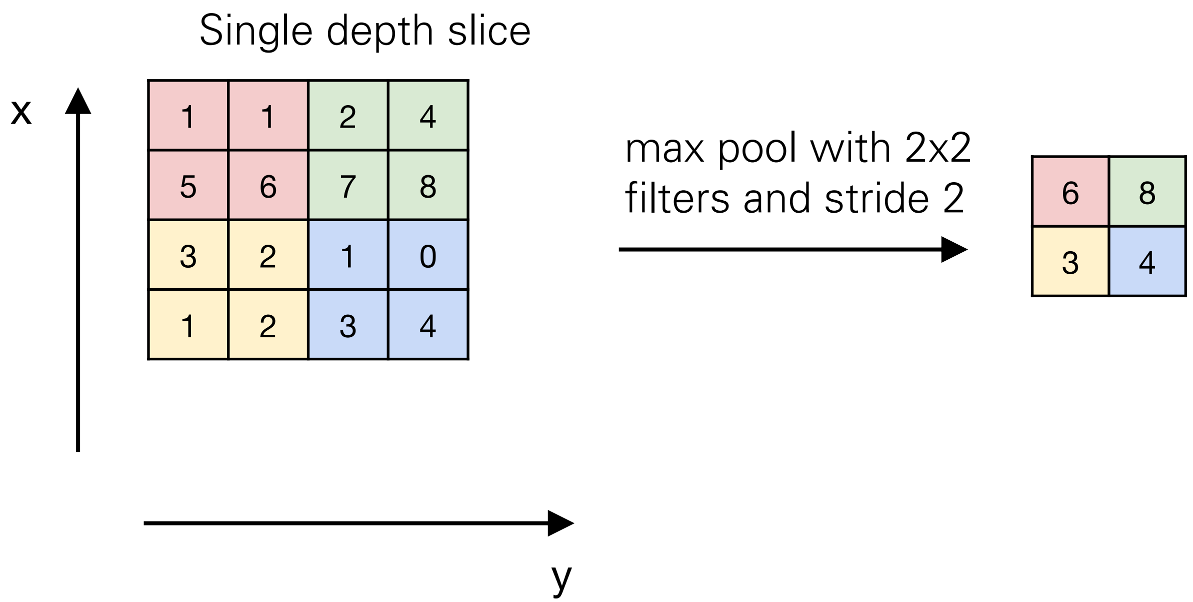


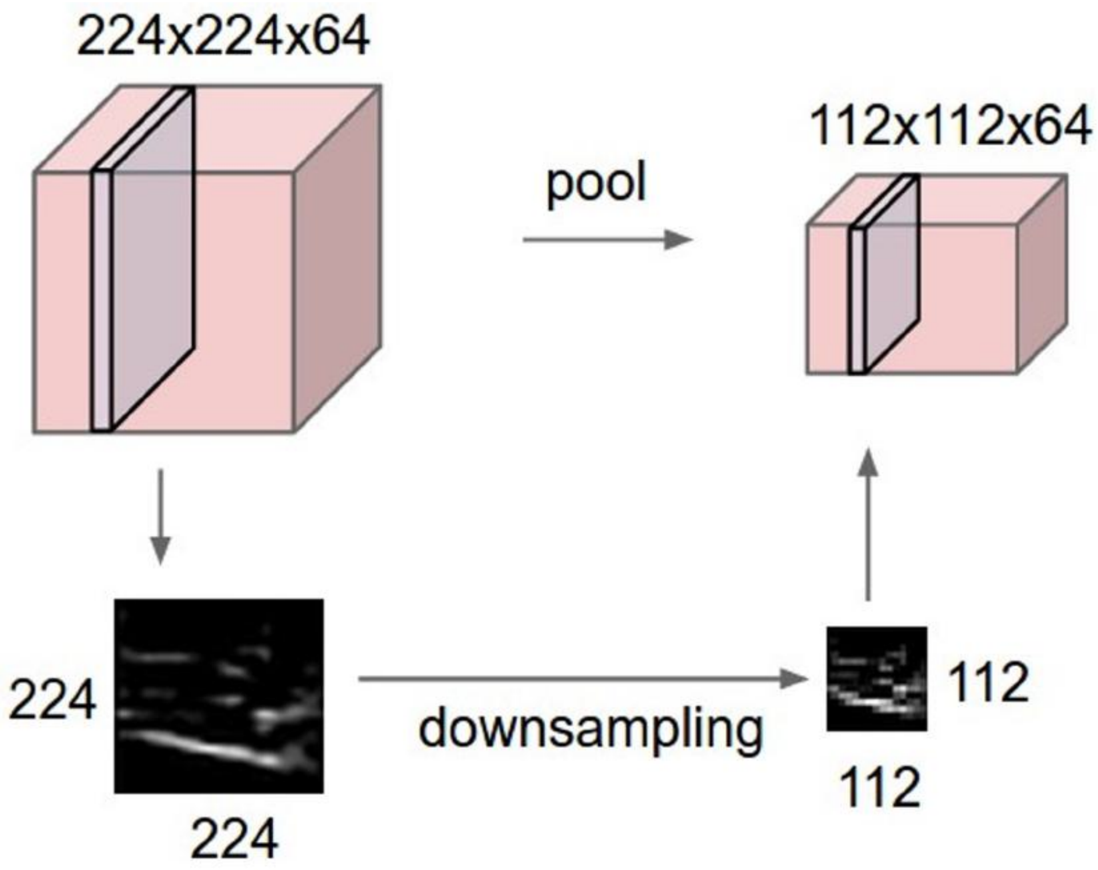
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Yann LeCun

Pooling layer

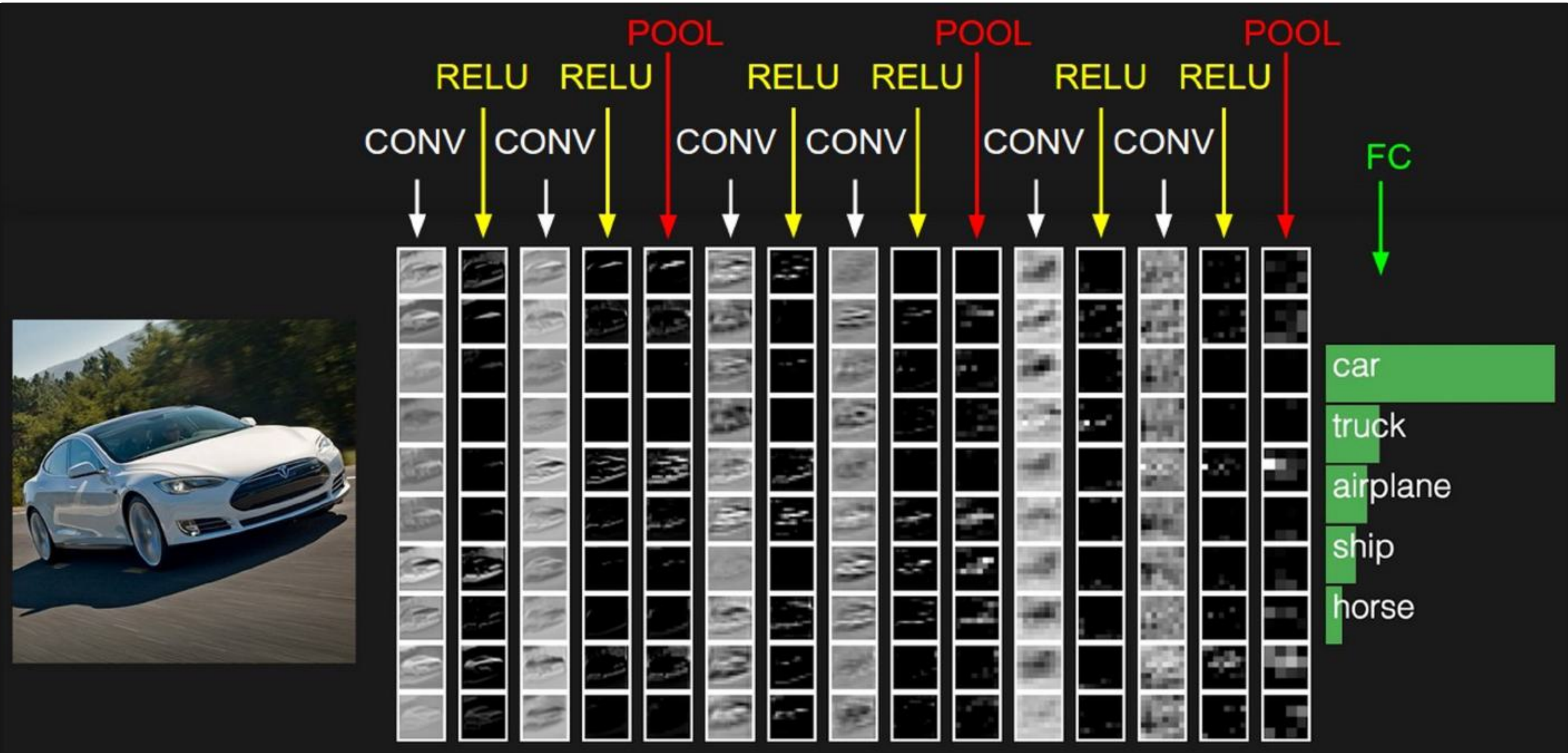
- makes the representations smaller and more manageable
- operates over each activation map independently:
- Max pooling, average pooling, etc.





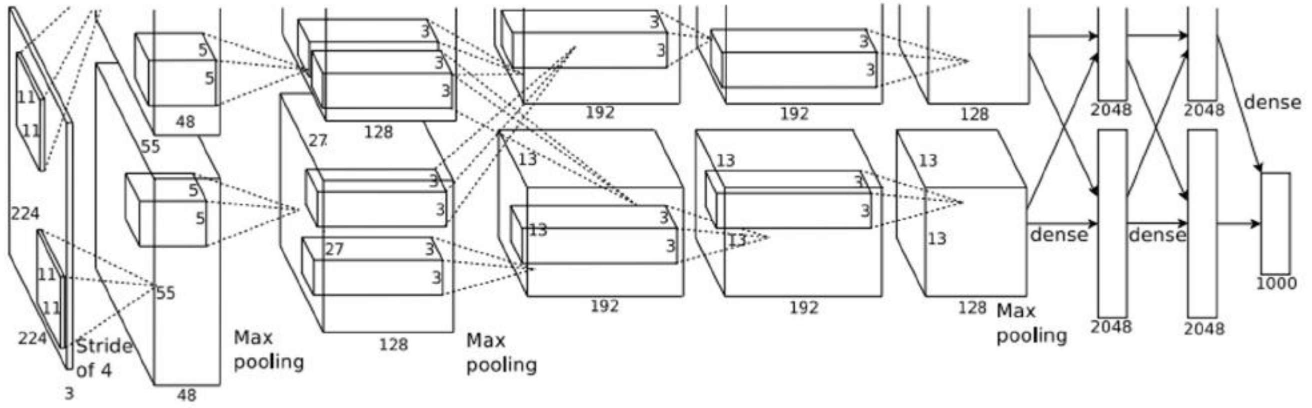
Fully connected layer

- contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

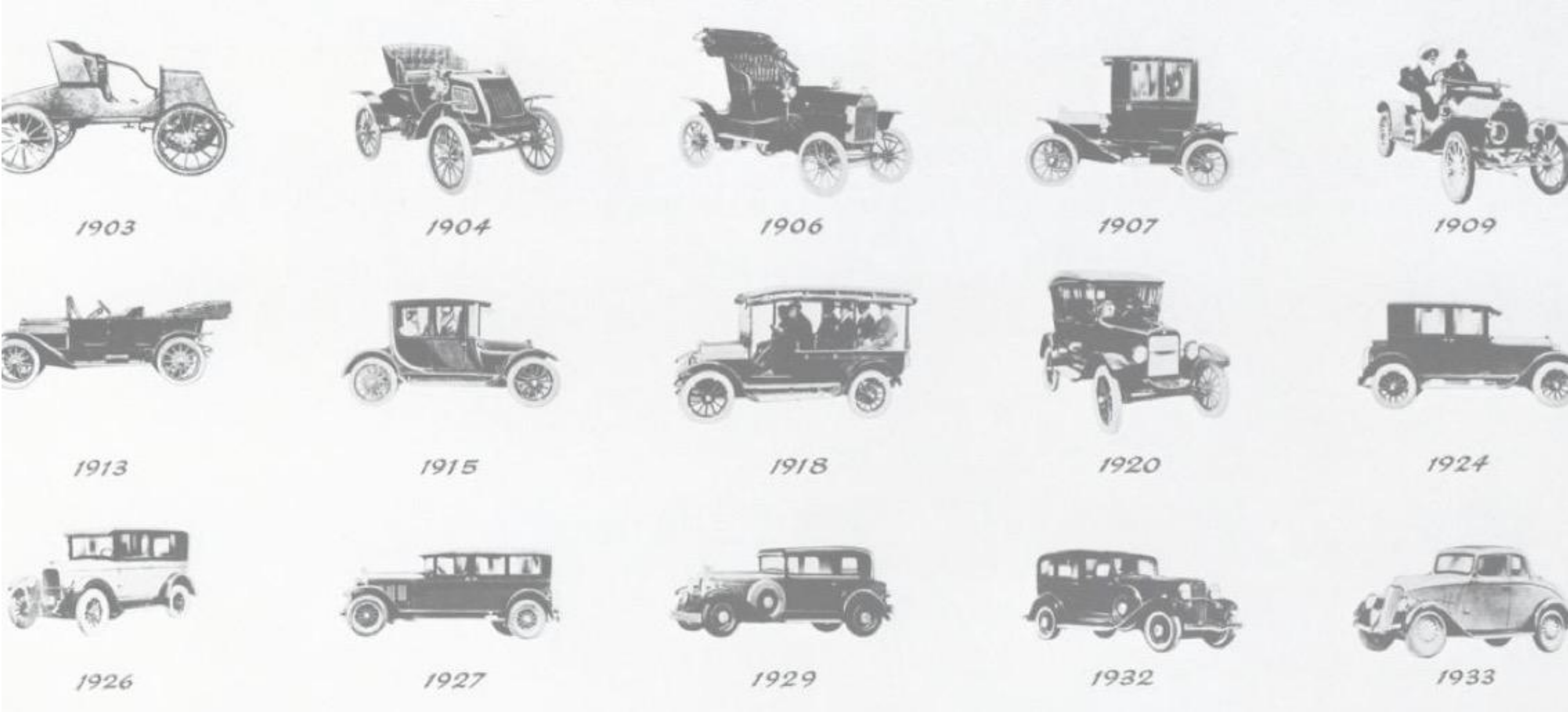
Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Convolutional Neural Network Demo

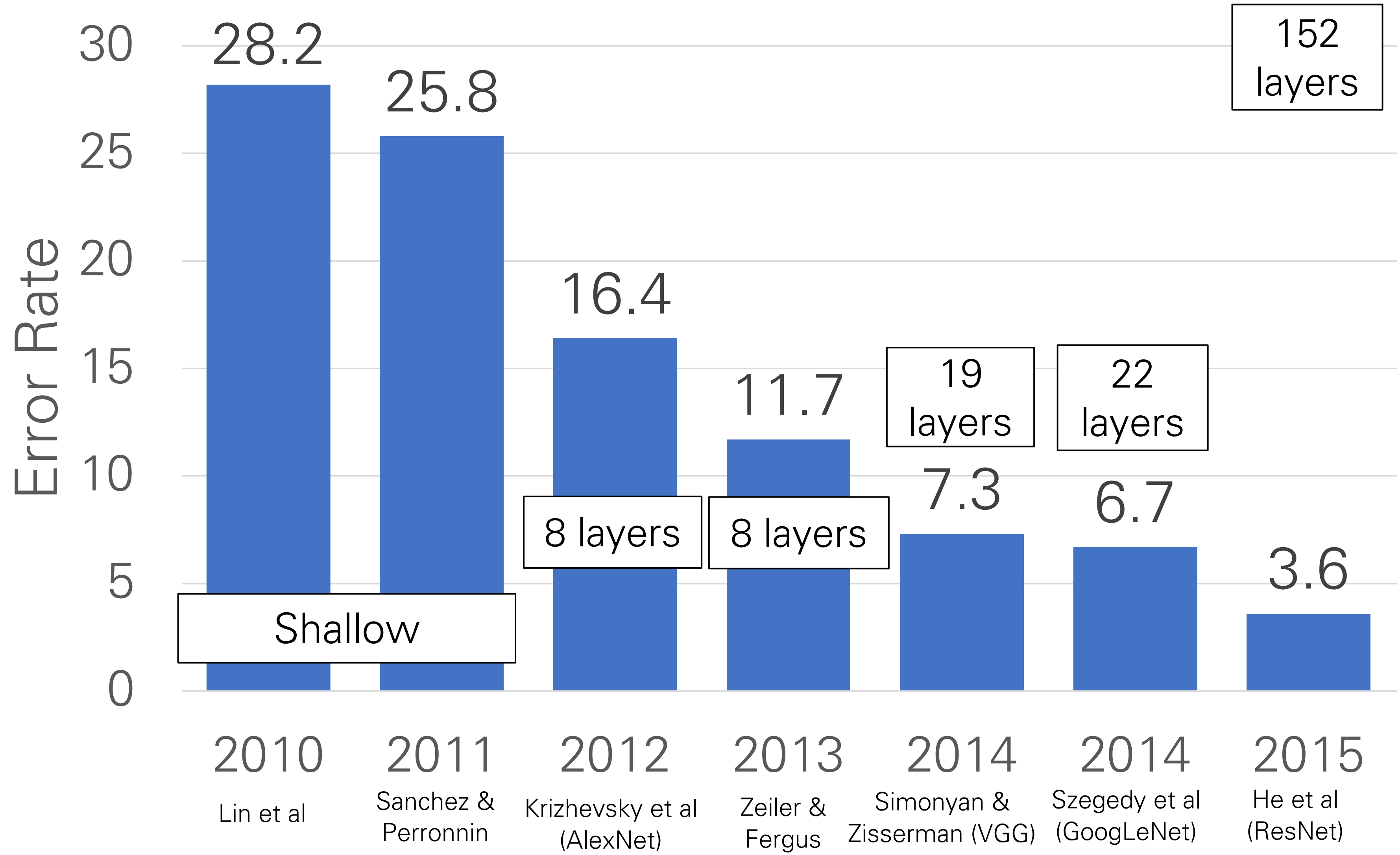
- ConvNetJS demo: training on CIFAR-10
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Three Years of Progress

From AlexNet (2012) to ResNet (2015)



ImageNet Classification Challenge



Revolution of Depth

AlexNet, 8 layers

(ILSVRC 2012)

11x11 conv, 96, /4, pool/2

5x5 conv, 256, pool/2

3x3 conv, 384

3x3 conv, 384

3x3 conv, 256, pool/2

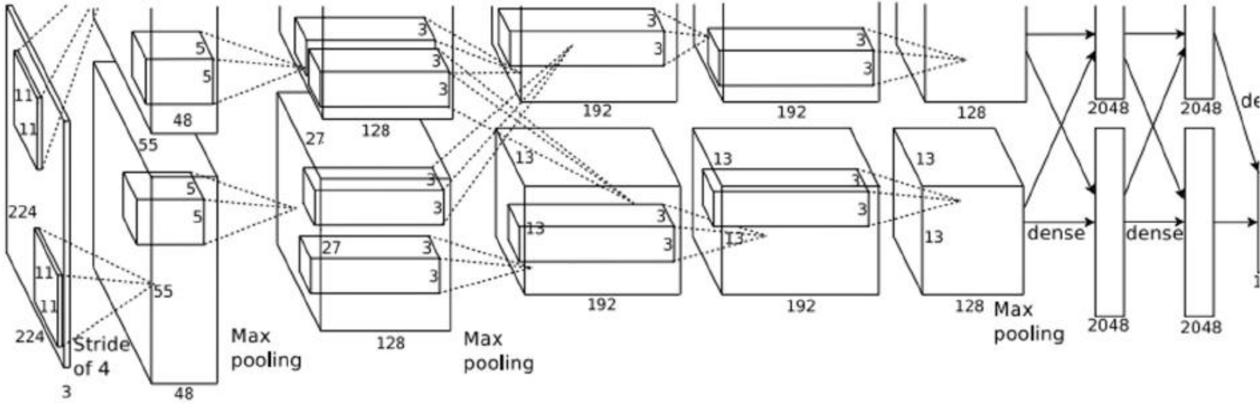
fc, 4096

fc, 4096

fc, 1000

- 5 convolutional layers
- 3 fully connected layers
- ReLU
- End-to-end (no pre-training)
- Data augmentation

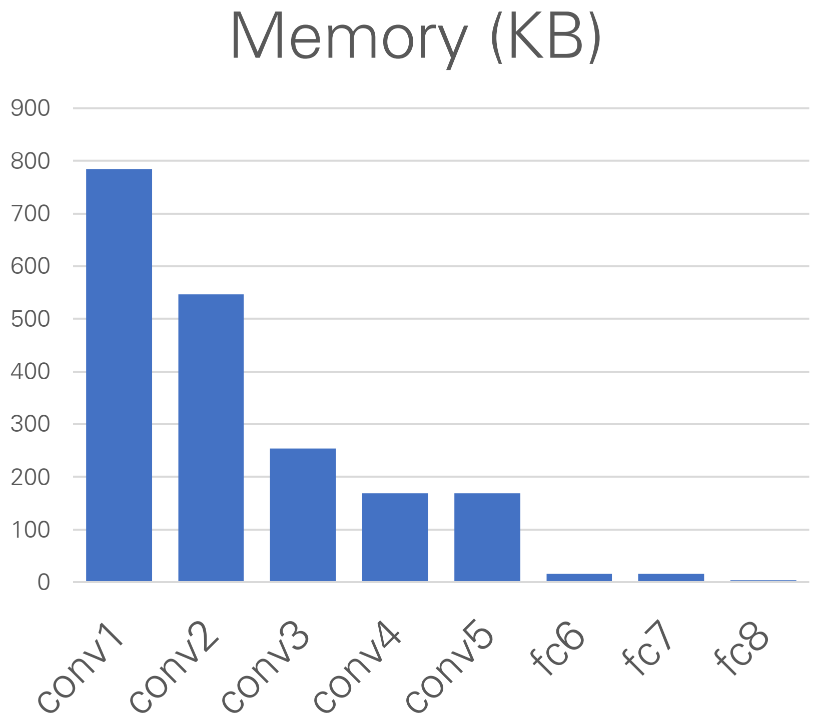
AlexNet

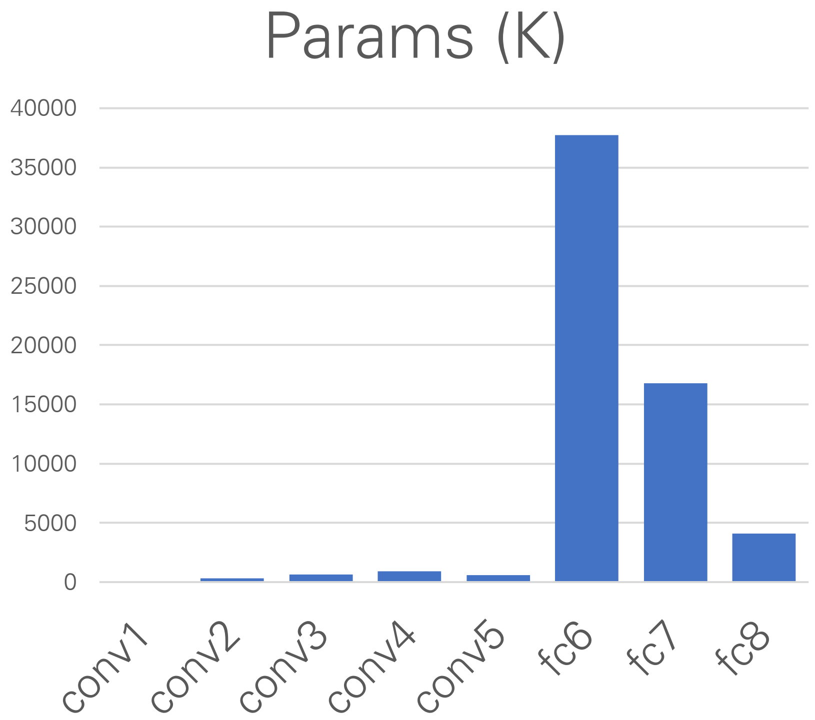


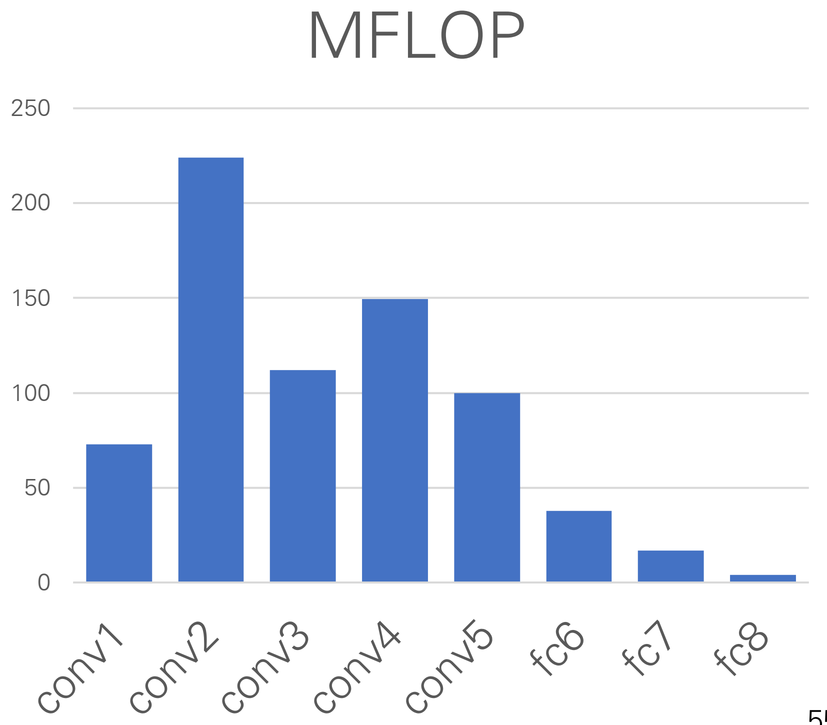
Most of the **memory usage** is in the early convolution layers

Nearly all **parameters** are in the fully-connected layers

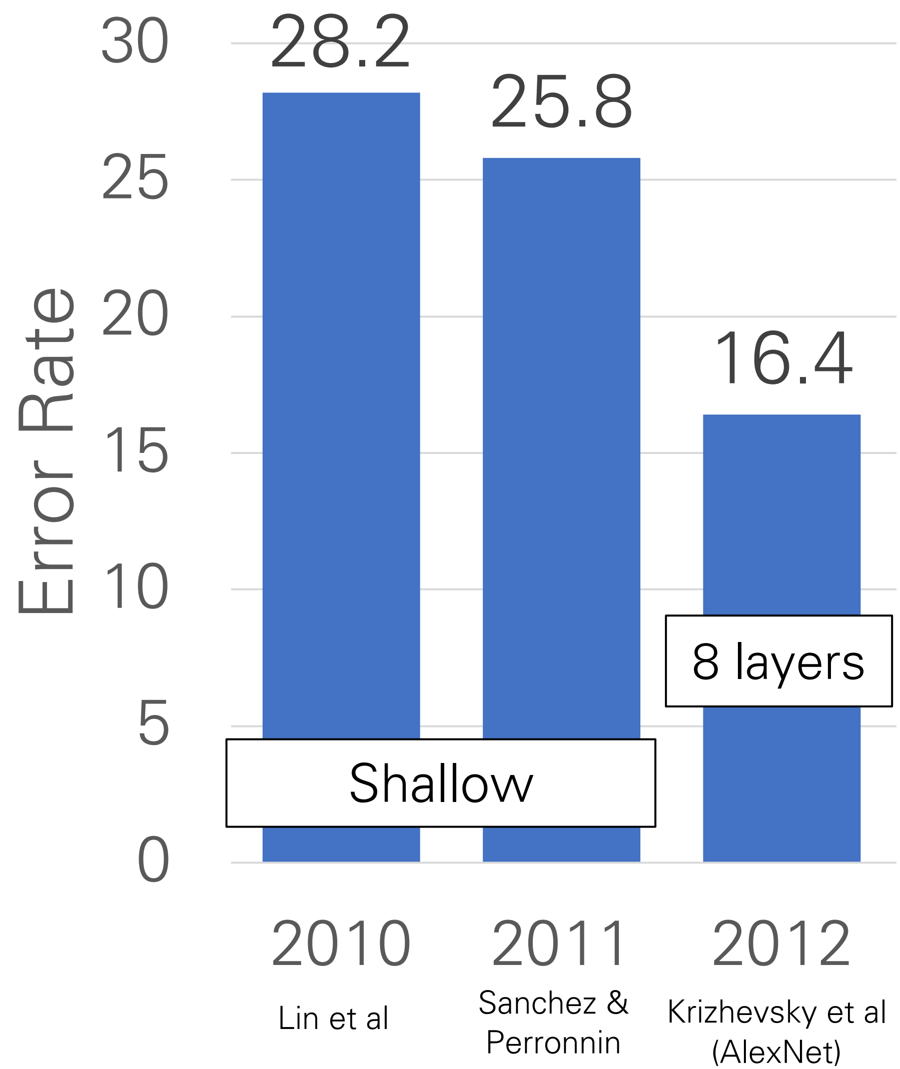
Most **floating-point ops** occur in the convolution layers



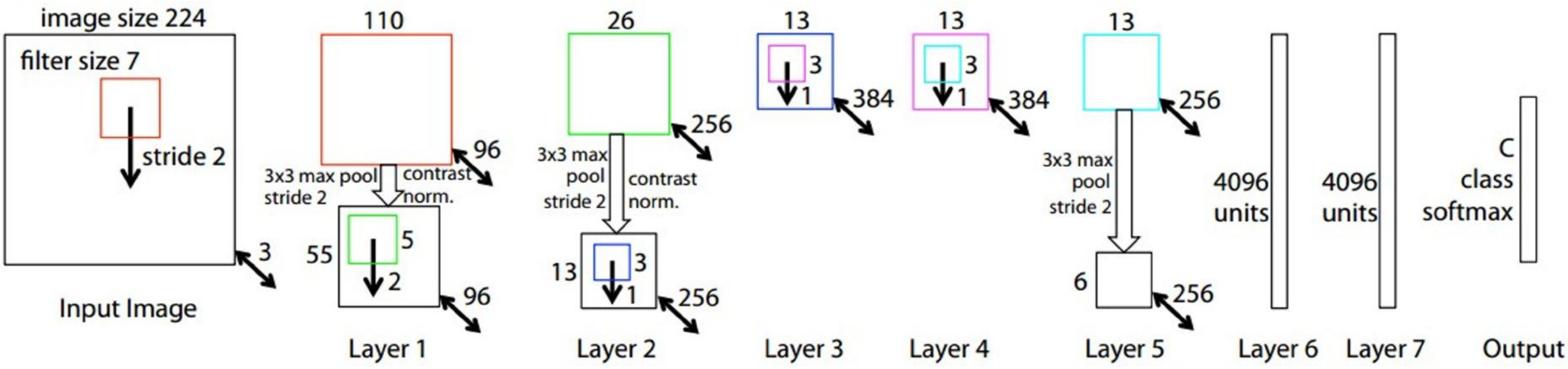




ImageNet Classification Challenge



ZFNet: A Bigger AlexNet

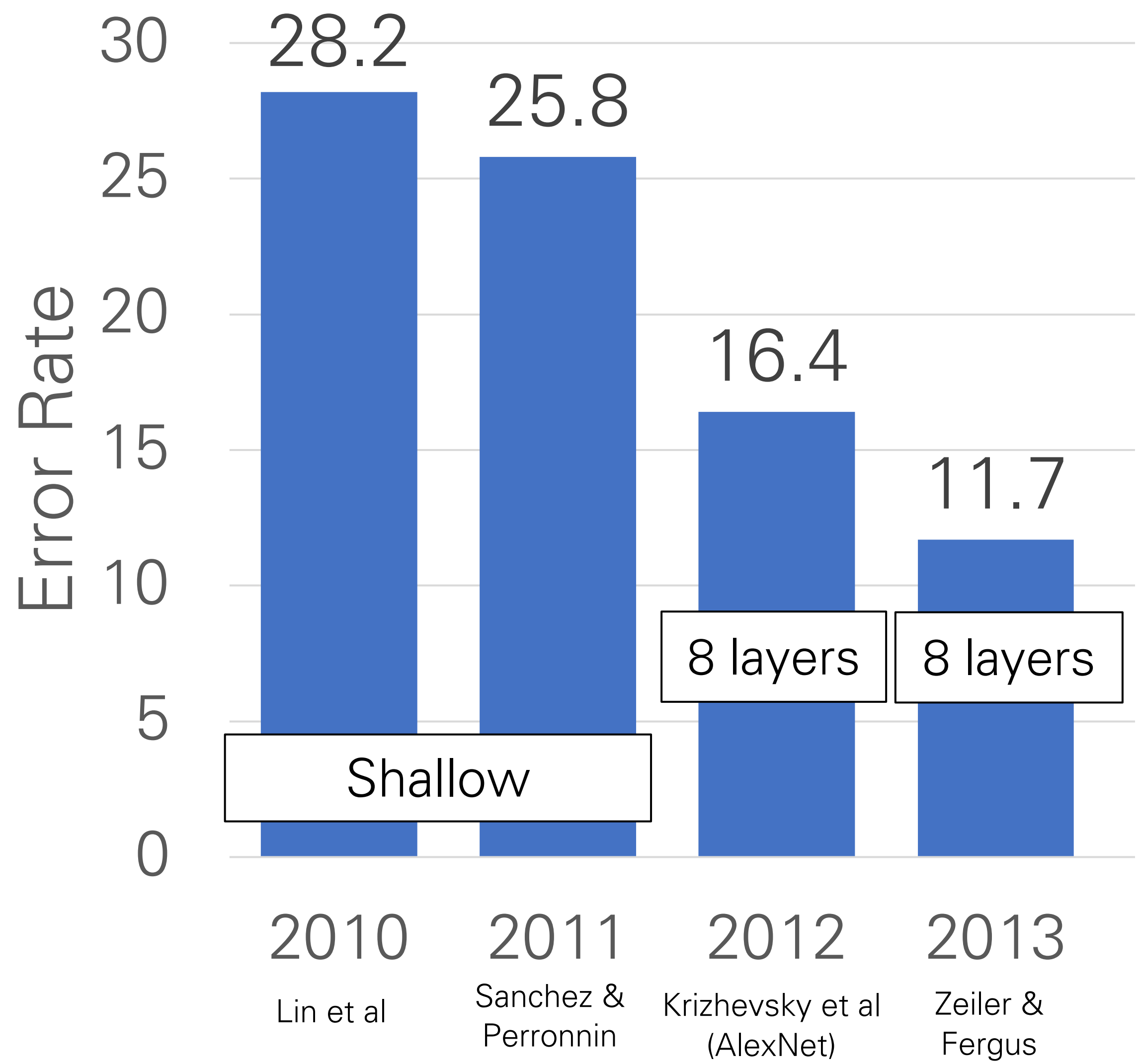


AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512
More trial and error

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

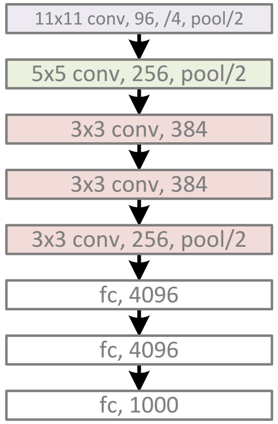
ImageNet Classification Challenge



Revolution of Depth

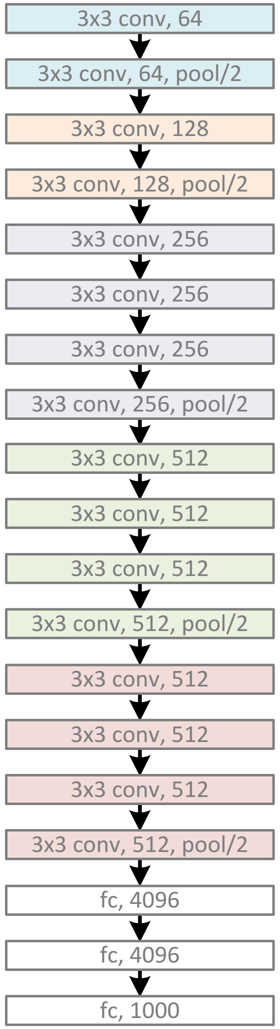
AlexNet, 8 layers

(ILSVRC 2012)



VGG, 19 layers

(ILSVRC 2014)



VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Network has 5 convolutional **stages**:
Stage 1: conv-conv-pool
Stage 2: conv-conv-pool
Stage 3: conv-conv-pool
Stage 4: conv-conv-conv-[conv]-pool
Stage 5: conv-conv-conv-[conv]-pool

(VGG-19 has 4 conv in stages 4 and 5

- Very deep
- Simply deep

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Two 3x3 conv has same receptive field as a single 5x5 conv, but has fewer parameters and takes less computation!

<u>Option 1:</u>

Conv(5x5, C -> C)

Params: $25C^2$

FLOPs: $25C^2HW$

<u>Option 2:</u>

Conv(3x3, C -> C)

Conv(3x3, C -> C)

Params: $18C^2$

FLOPs: $18C^2HW$

AlexNet VGG16 VGG19

Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Conv layers at each spatial resolution take the same amount of computation!

Input: C x 2H x 2W
Layer: Conv(3x3, C->C)

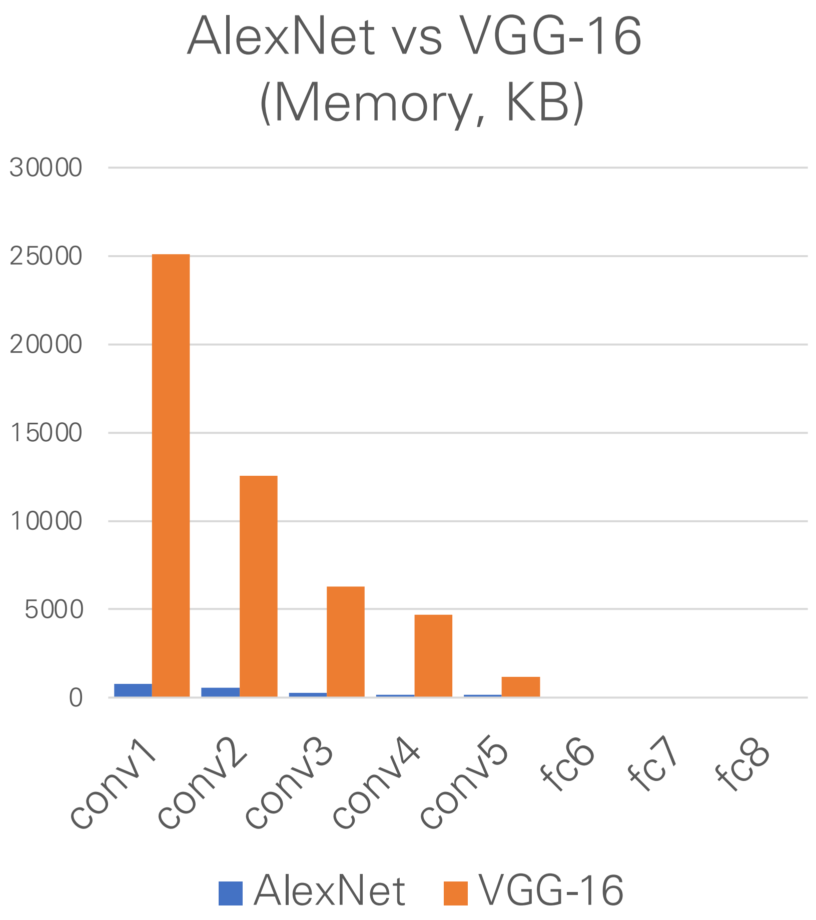
Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$

Input: 2C x H x W
Conv(3x3, 2C -> 2C)

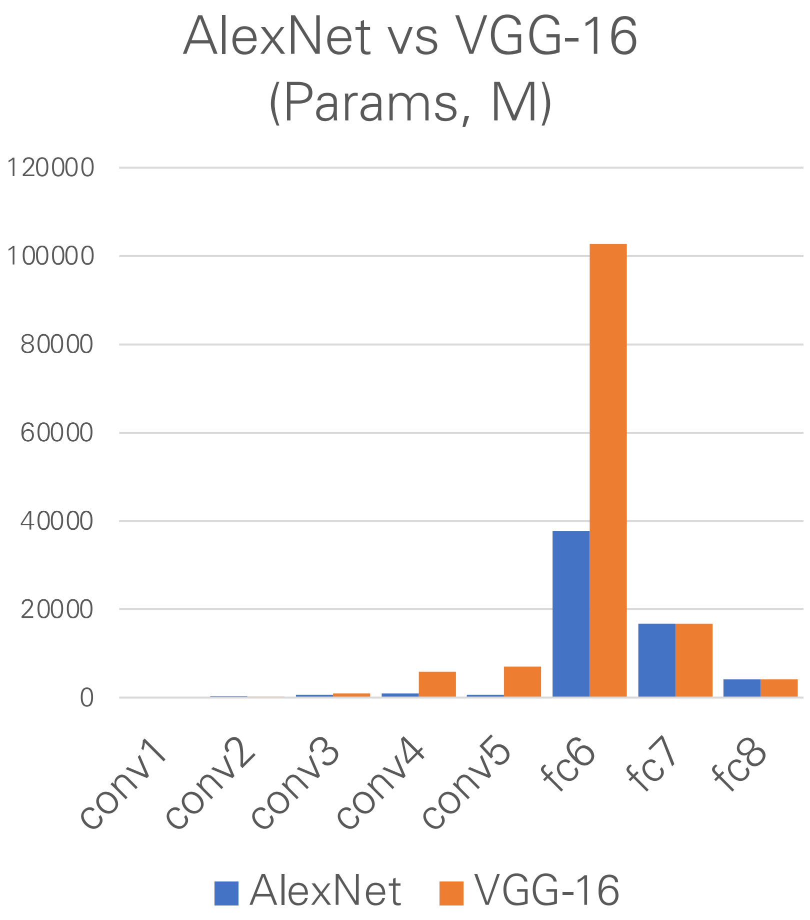
Memory: 2HWC
Params: $36C^2$
FLOPs: $36HWC^2$

Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

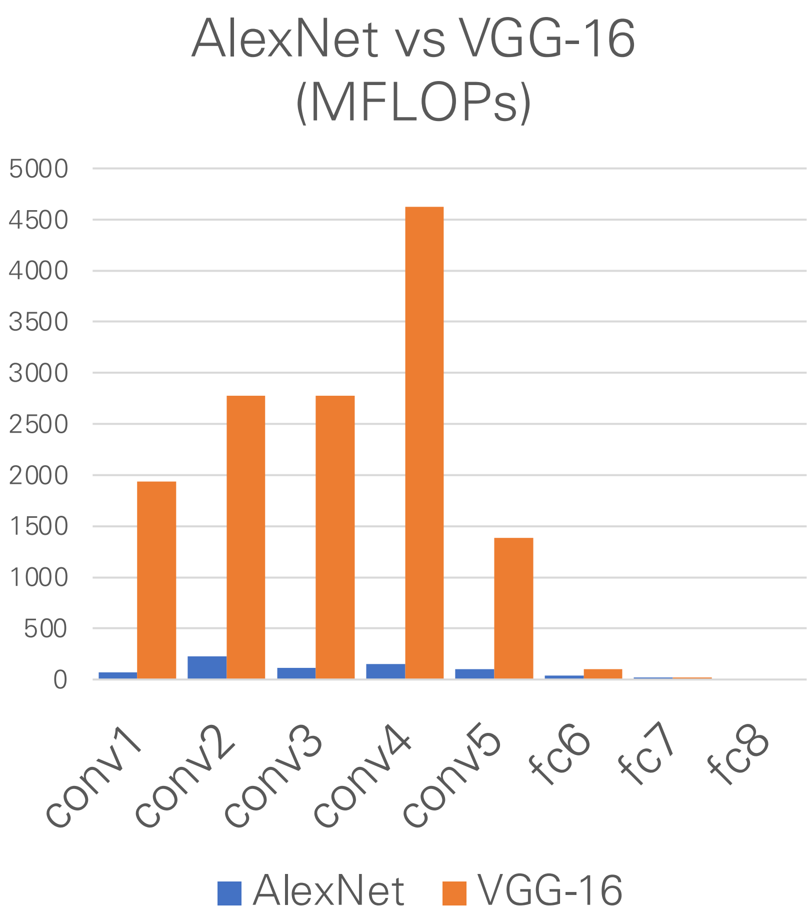
AlexNet vs VGG-16: Much bigger network!



AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

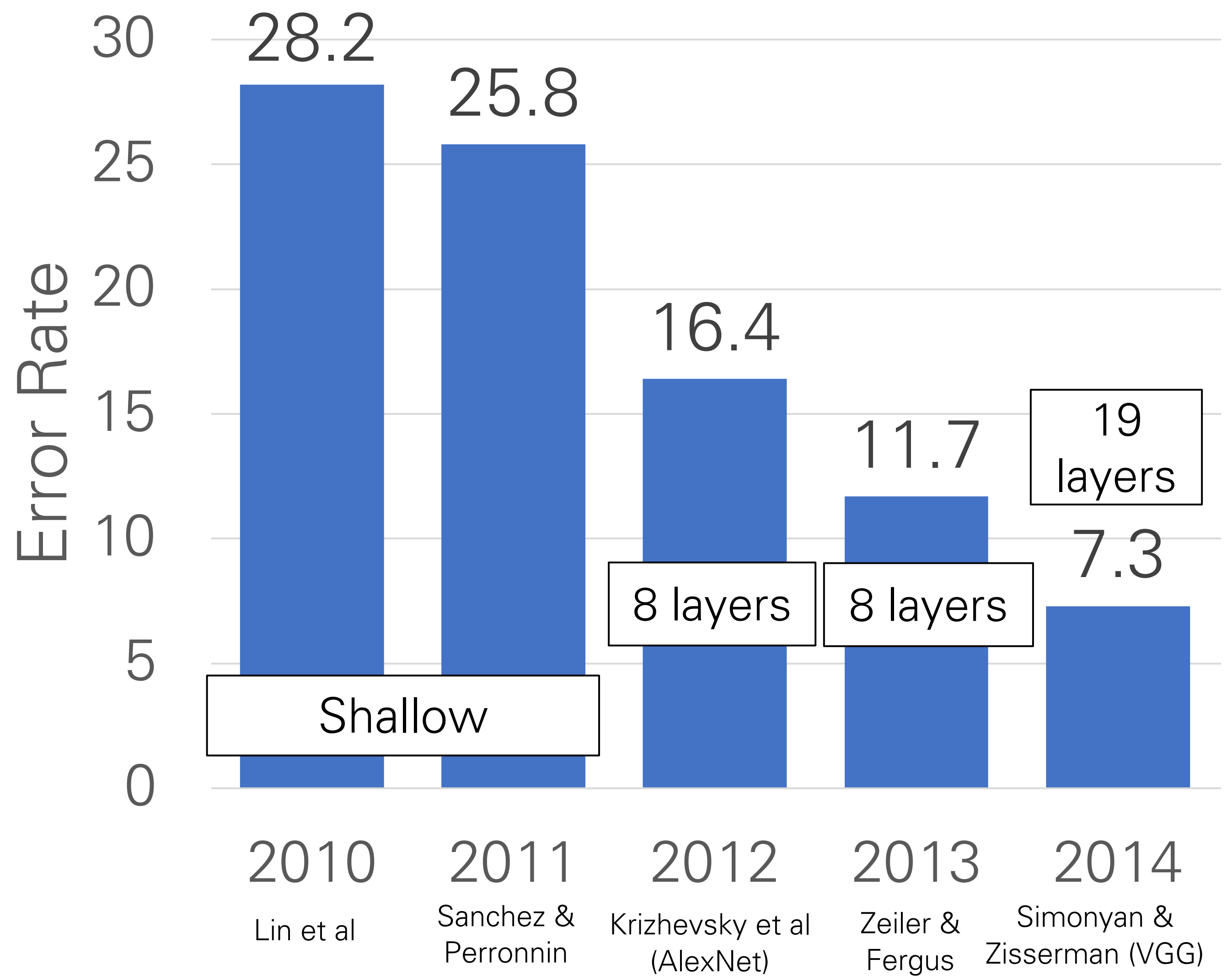


AlexNet total: 61M
VGG-16 total: 138M (2.3x)



AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

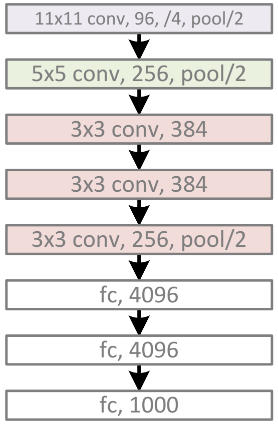
ImageNet Classification Challenge



Revolution of Depth

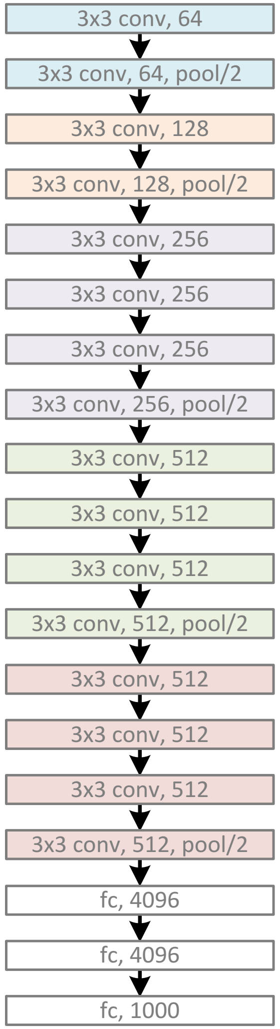
AlexNet, 8 layers

(ILSVRC 2012)



VGG, 19 layers

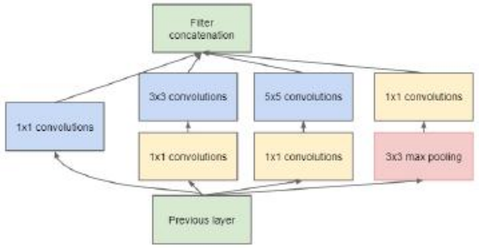
(ILSVRC 2014)

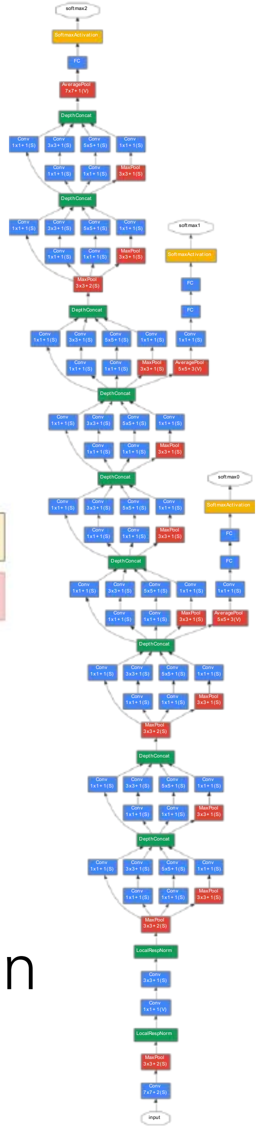


- Very deep
- Simply deep

GoogLeNet,
22 layers

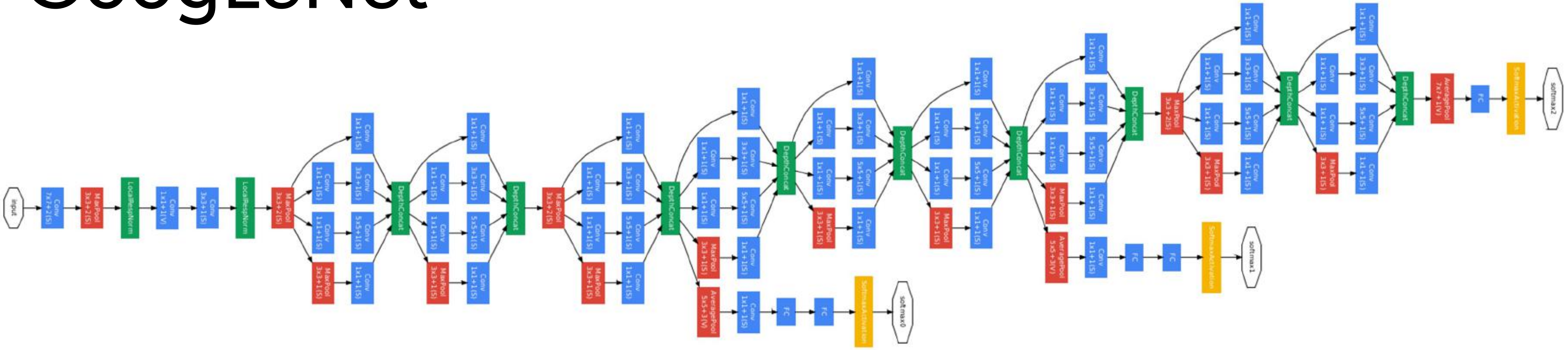
(ILSVRC 2014)





- Branching
- Bootleneck
- Skip connection

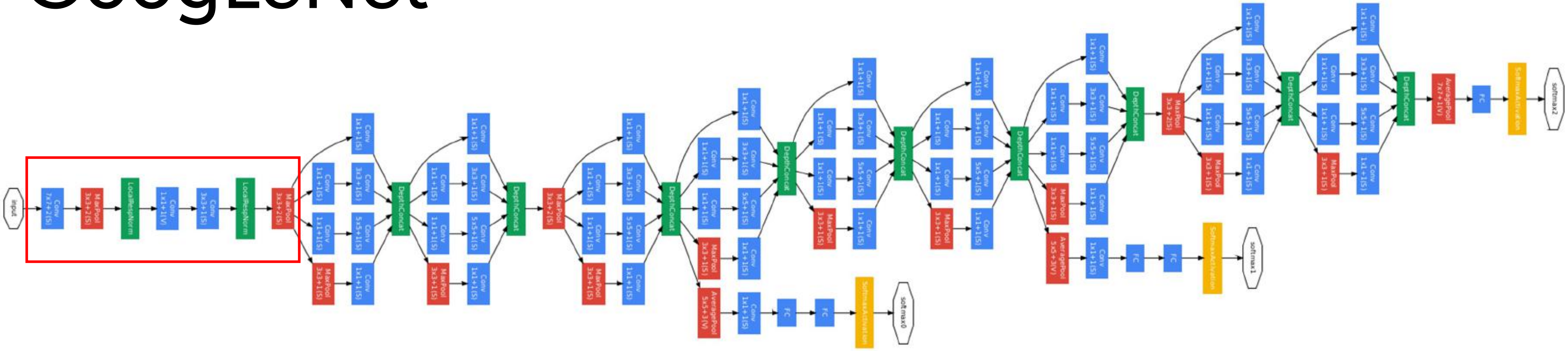
GoogLeNet



Many innovations for efficiency: reduce parameter count, memory usage, and computation

Szegedy et al, "Going deeper with convolutions", CVPR 2015

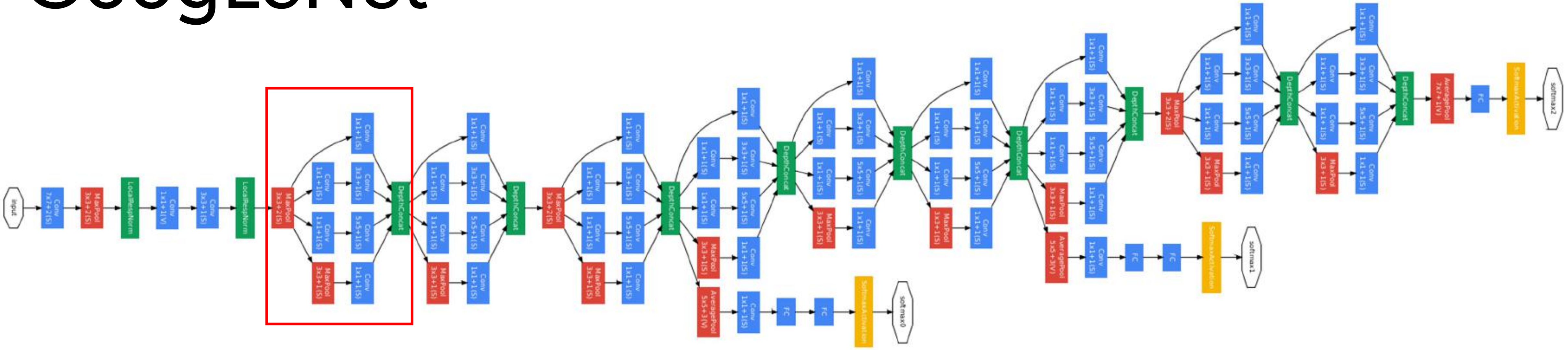
GoogLeNet

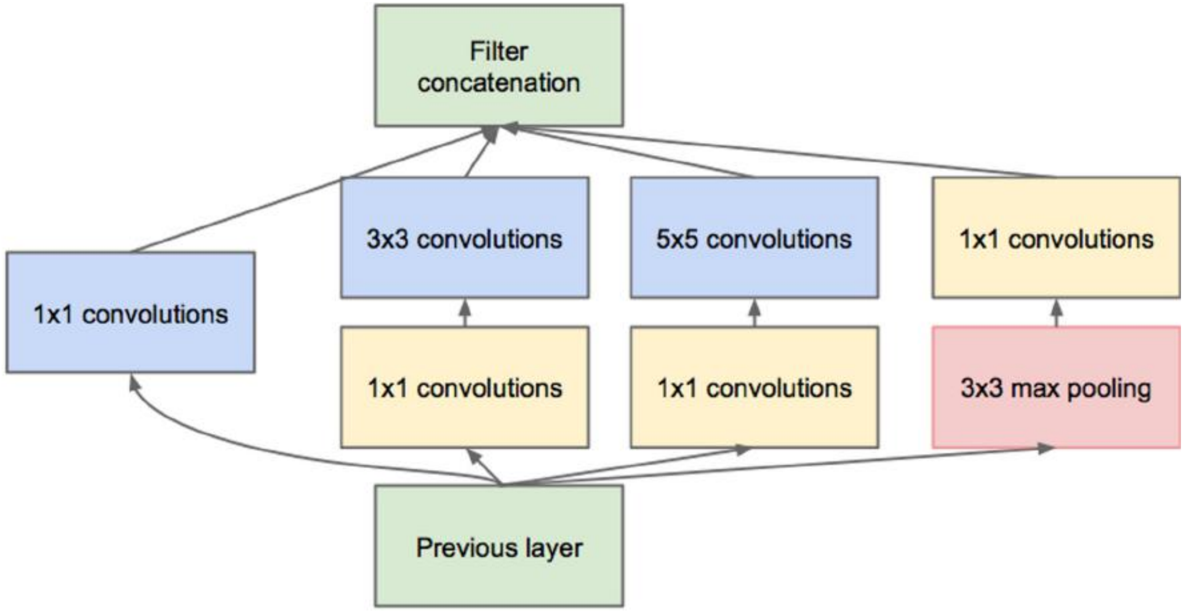


Stem network at the start aggressively downsamples input (Recall in VGG-16: Most of the compute was at the start)

Szegedy et al, "Going deeper with convolutions", CVPR 2015

GoogLeNet





Inception module

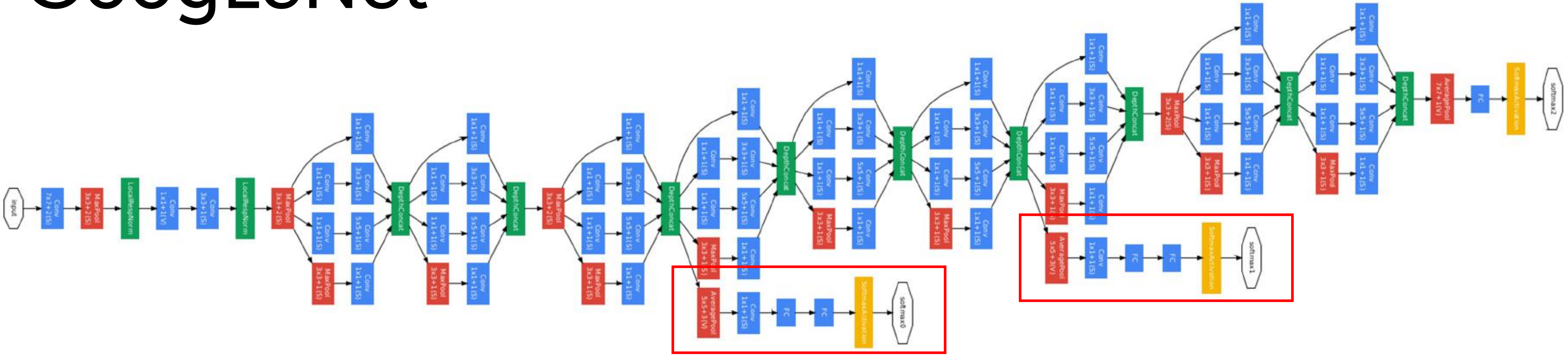
Local unit with parallel branches

Local structure repeated many times throughout the network

Uses 1x1 "Bottleneck" layers to reduce channel dimension before expensive conv

Szegedy et al, "Going deeper with convolutions", CVPR 2015

GoogLeNet



Auxiliary Classifiers

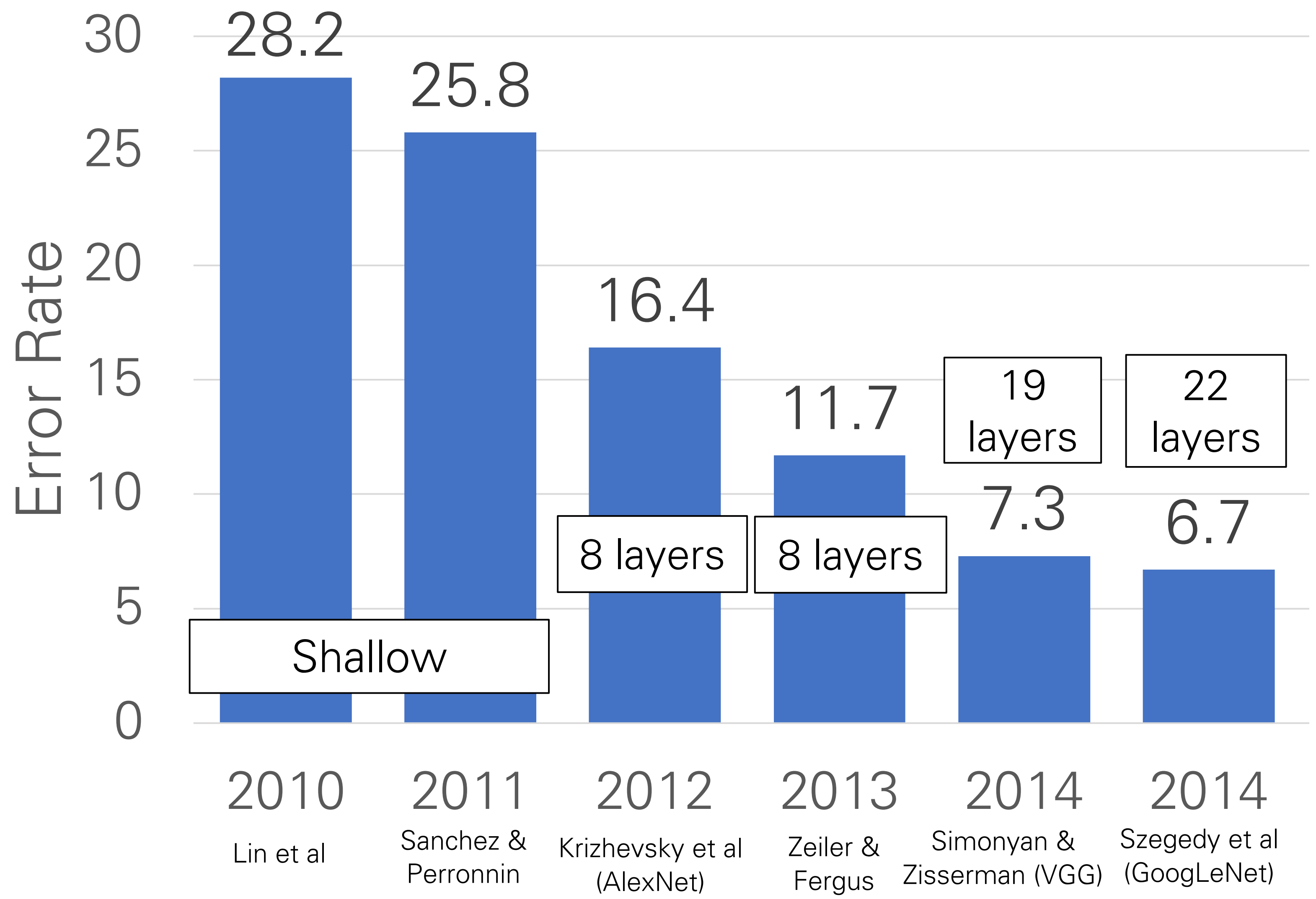
Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly

As a hack, attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss

GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick

Szegedy et al, "Going deeper with convolutions", CVPR 2015

ImageNet Classification Challenge



GoogLeNet

[Szegedy et al., 2014]

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

Fun features:

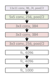
- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

Revolution of Depth

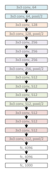
AlexNet, 8 layers

(ILSVRC 2012)



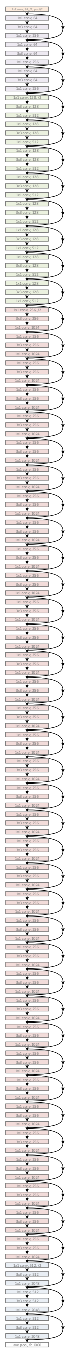
VGG, 19 layers

(ILSVRC 2014)



ResNet,
152 layers

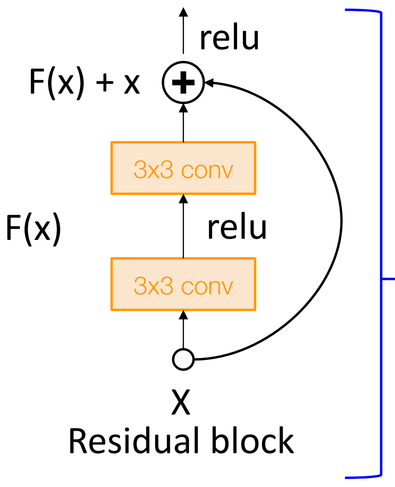
(ILSVRC 2015)

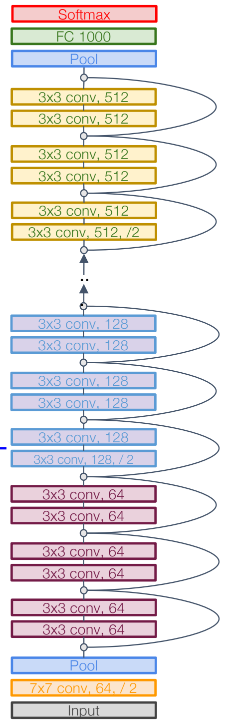


A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels





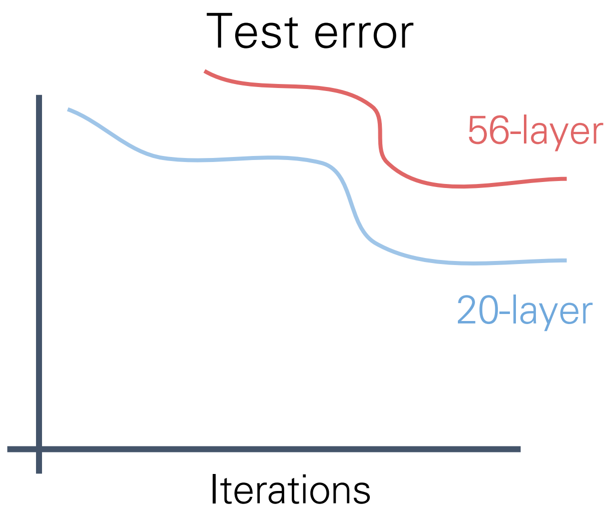
Residual Networks (ResNet)

[He et al., 2015]

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?

Deeper model does worse than shallow model!

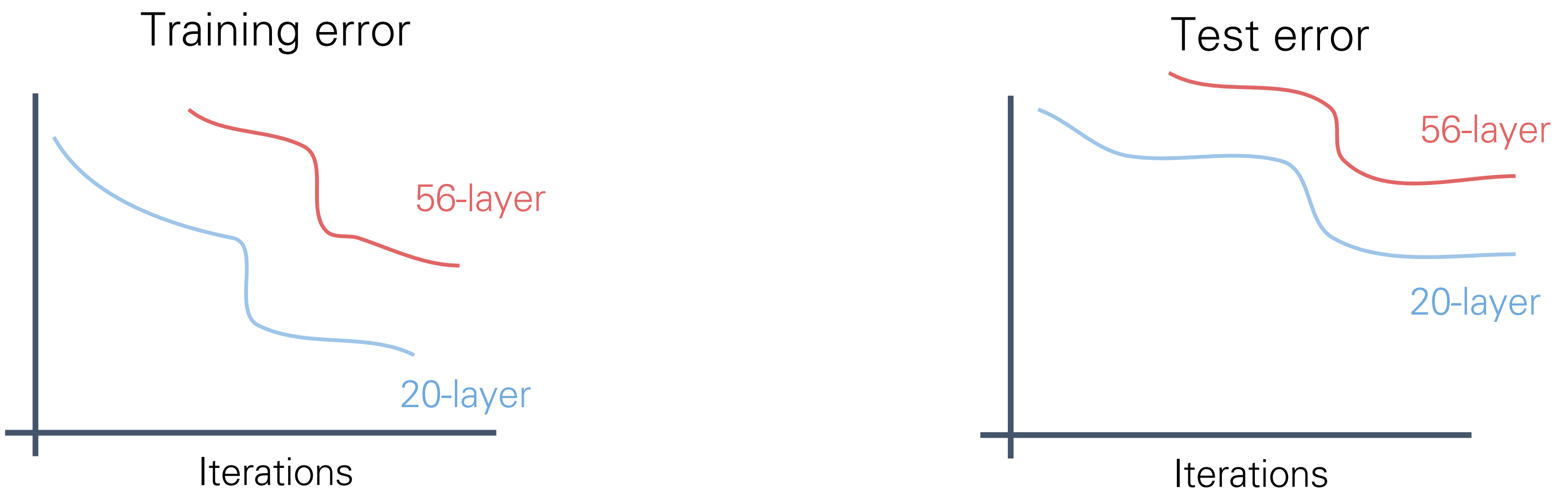
Initial guess: Deep model is **overfitting** since it is much bigger than the other model



Residual Networks (ResNet)

[He et al., 2015]

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?



In fact, the deep model seems to be **underfitting** since it also performs worse than the shallow model on the training set! It is actually **underfitting**

Residual Networks (ResNet)

[He et al., 2015]

- A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

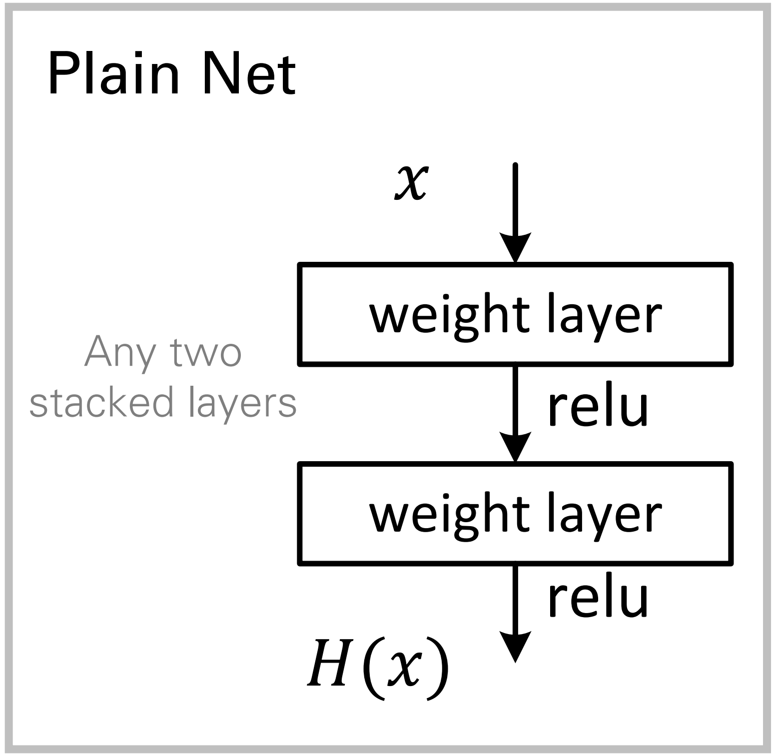
- Thus deeper models should do at least as good as shallow models

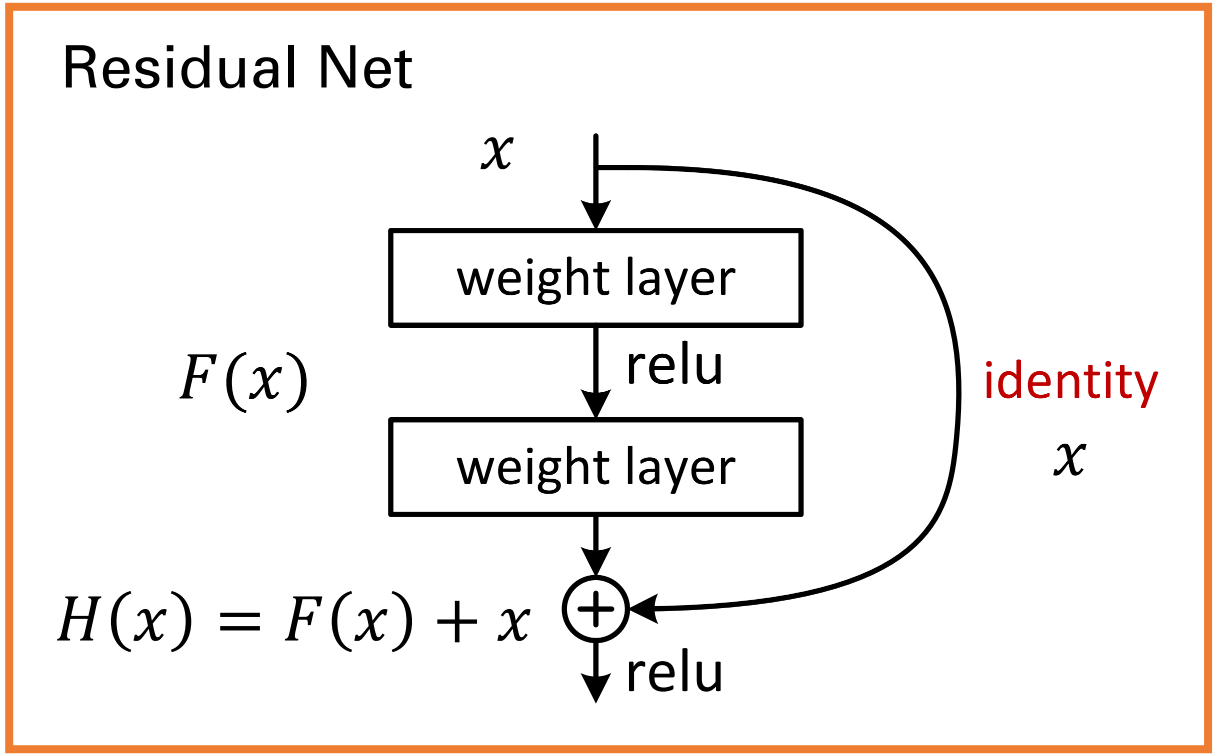
- **Hypothesis:** This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

Solution: Change the network so learning identity functions with extra layers is easy!

Residual Networks (ResNet)

[He et al., 2015]

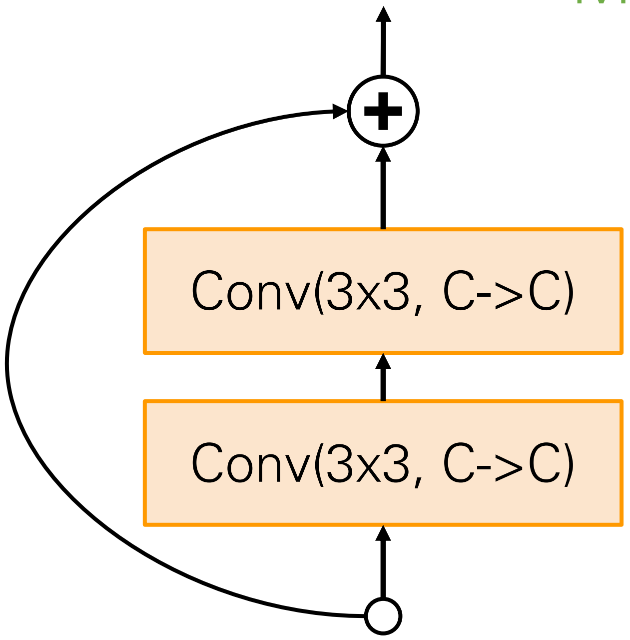




Residual Networks (ResNet)

[He et al., 2015]

More layers, less computational cost!

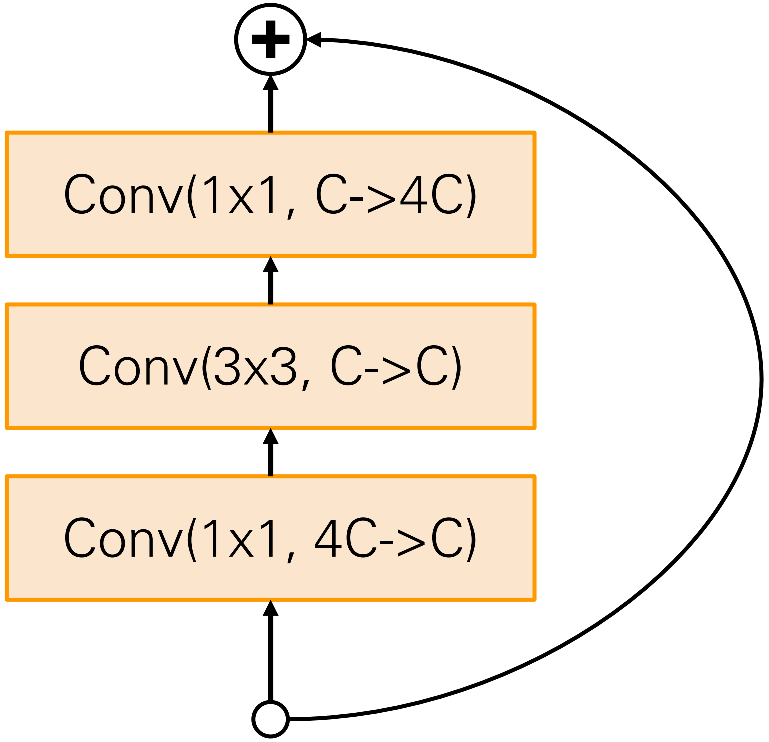


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic" Residual block

Total FLOPs: $18HWC^2$



FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

Total FLOPs: $17HWC^2$

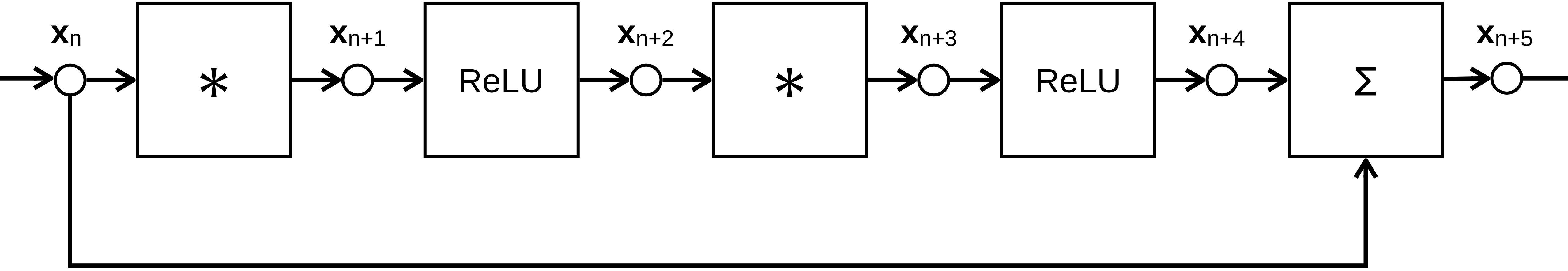
"Bottleneck" Residual block

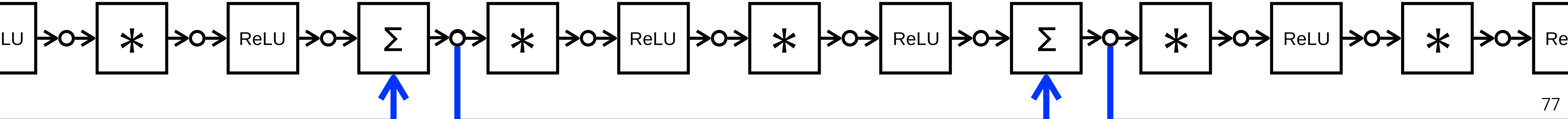
Residual Learning

Fixed identity
// learned residual

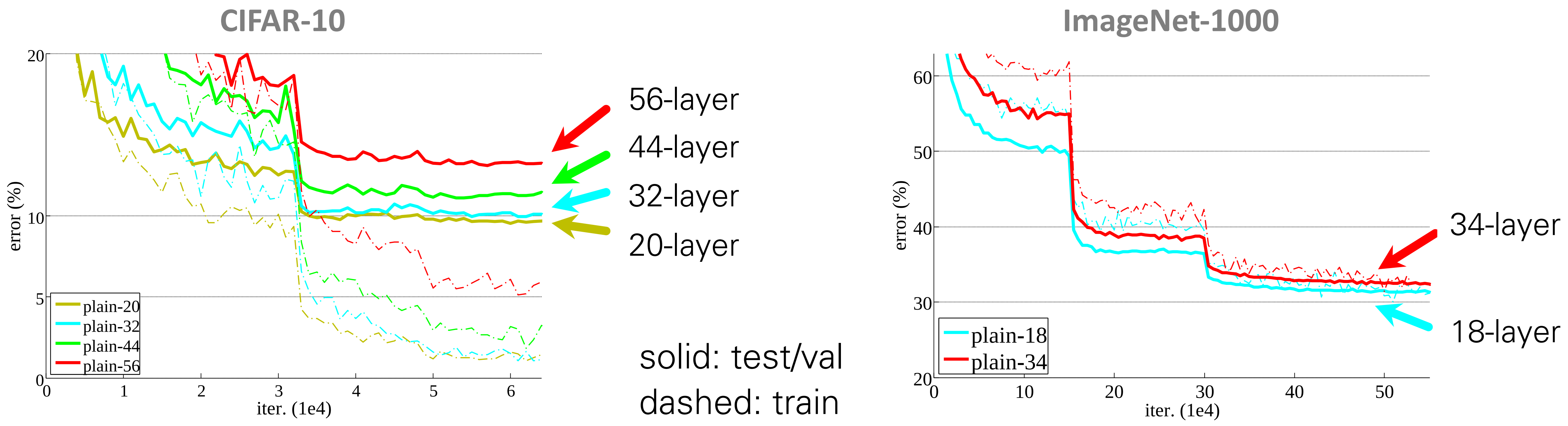
$$\underbrace{\mathbf{x}_{n+5} = \mathbf{x}_n}_{\text{identity}} + \underbrace{(\phi_{\mathrm{ReLU}} \circ \phi_* \circ \phi_{\mathrm{ReLU}} \circ \phi_*)(\mathbf{x}_n)}_{\text{residual}}$$

K. He, X. Zhang, S. Ren, and J. Sun. **Deep residual learning for image recognition**. In CVPR 2016.





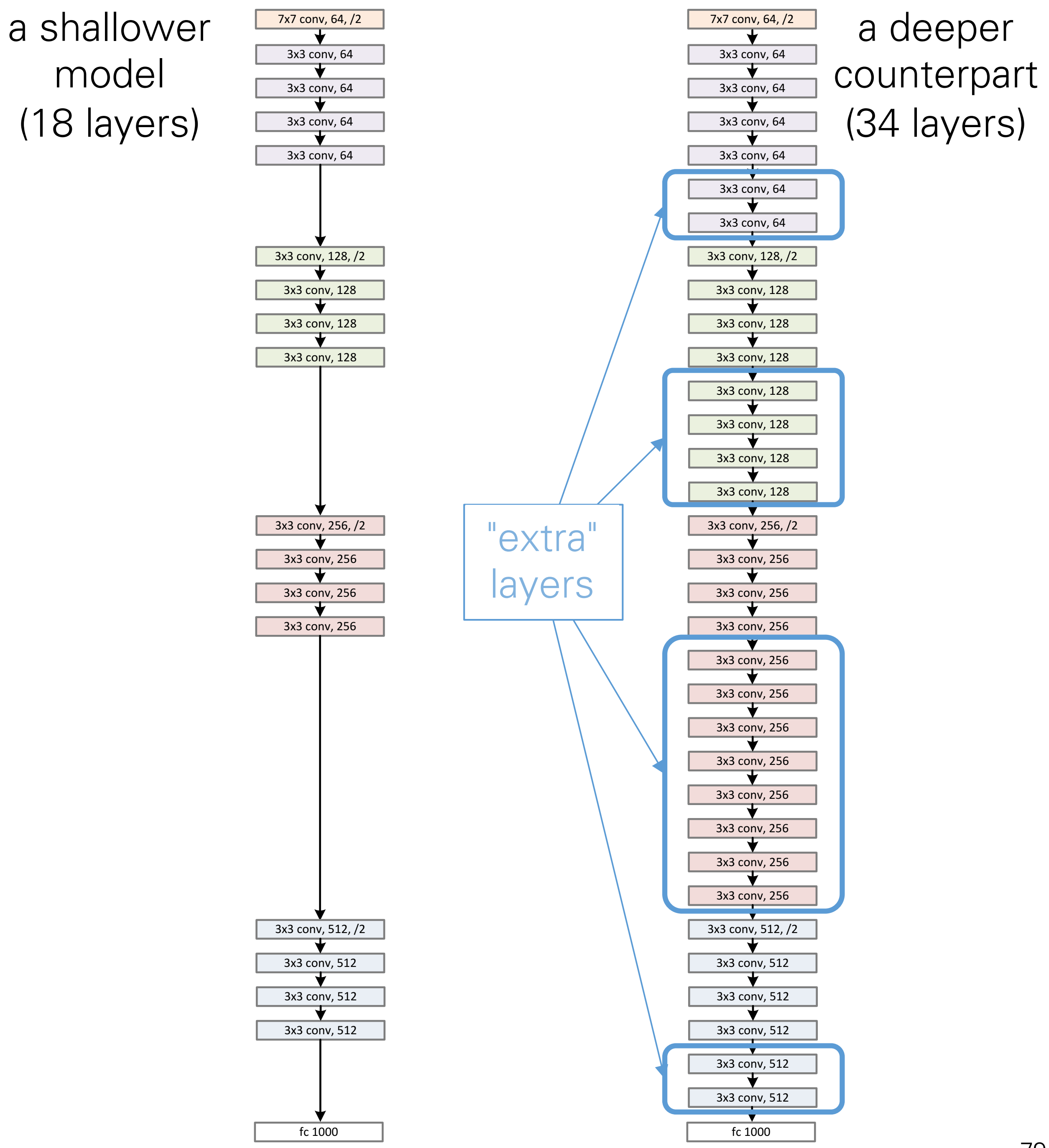
Residual Learning



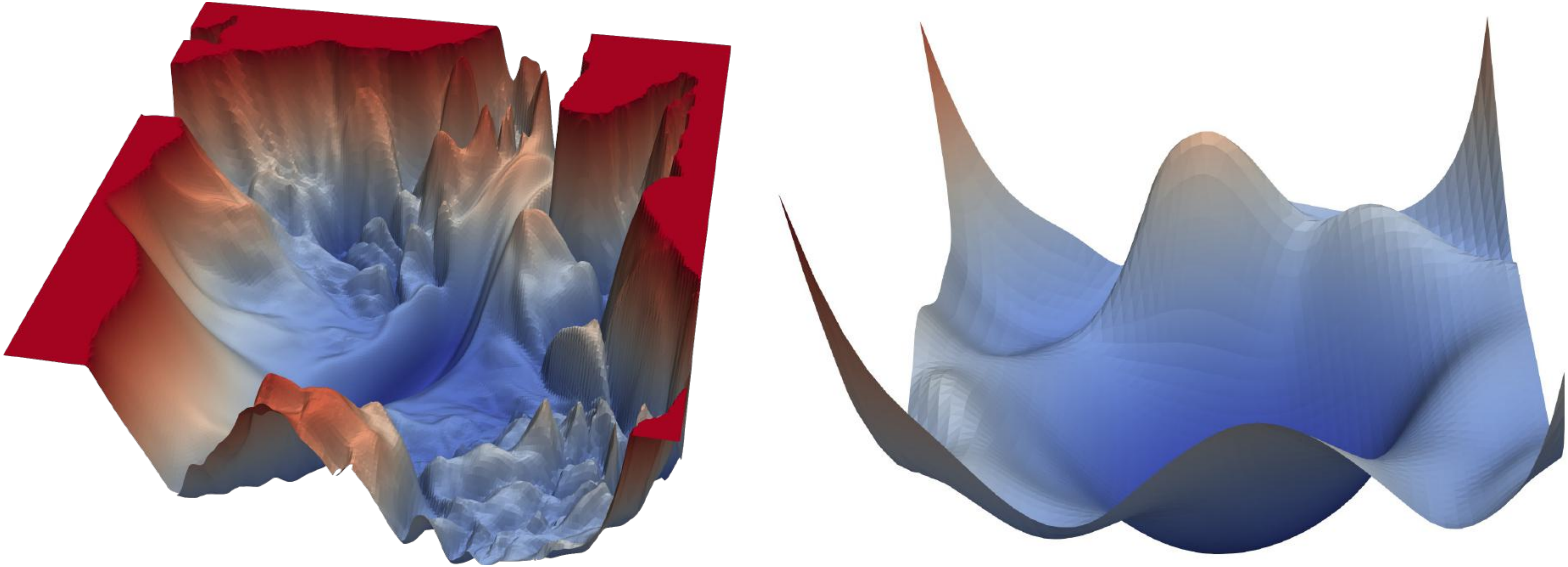
- "Overly deep" plain nets have **higher training error**
- A general phenomenon, observed in many datasets
- This is optimization issue, deeper models are harder to optimize

Residual Learning

- Richer solution space
- A deeper model should not have **higher training error**
- A solution by construction:
  - original layers: copied from a
  - learned shallower model
  - extra layers: set as identity
  - at least the same training error



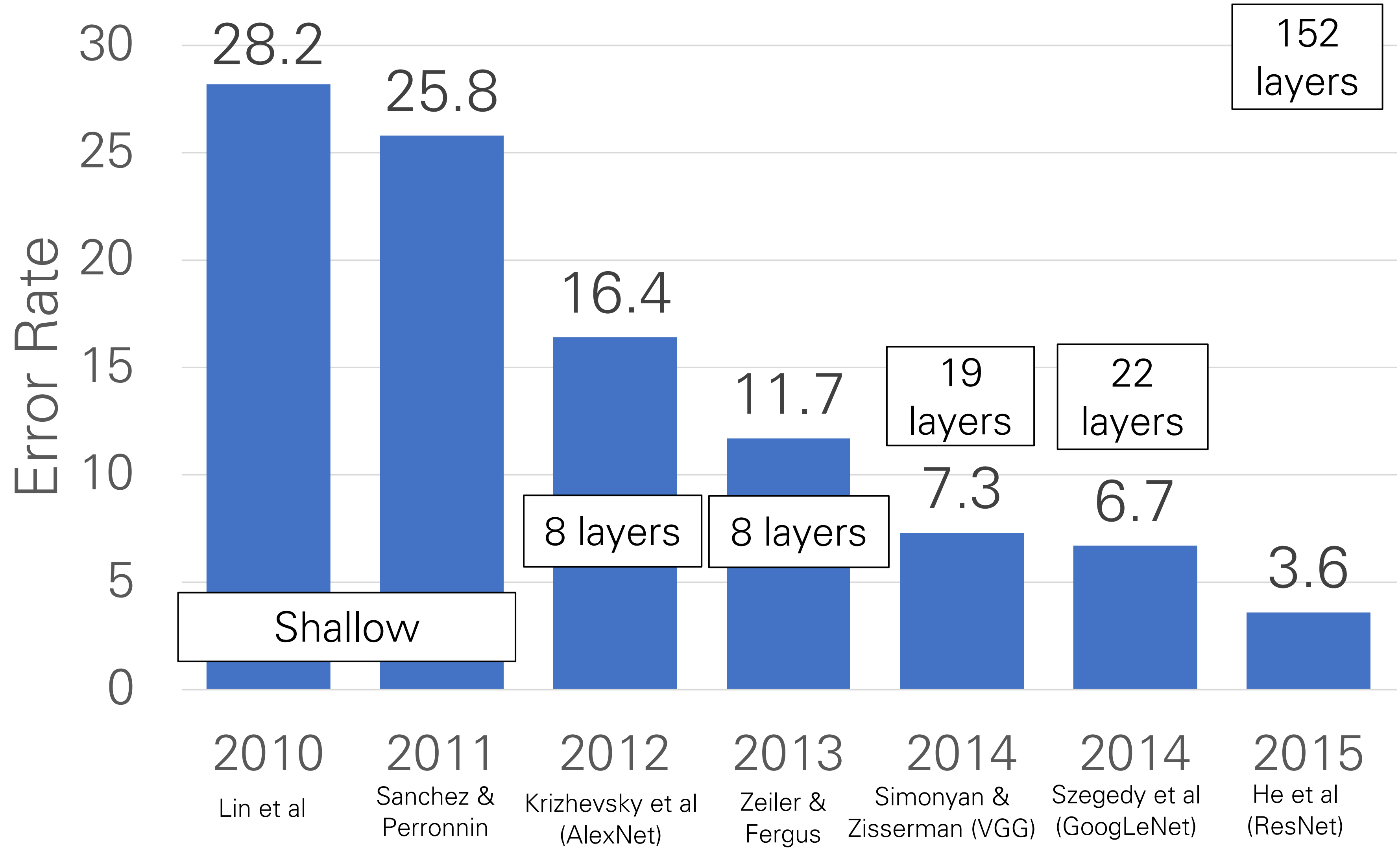
Residual Learning



- The loss surface of a 56-layer net using the CIFAR-10 dataset, both without (left) and with (right) residual connections.

Hao Li et al., "Visualizing the Loss Landscape of Neural Nets". ICLR 2018

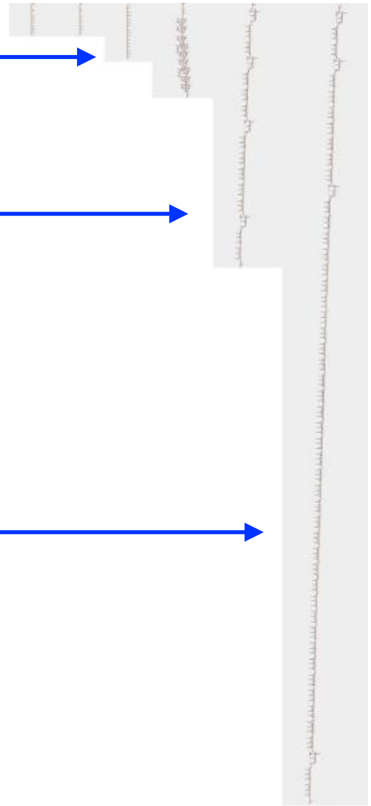
ImageNet Classification Challenge



How deep is enough?

GoogLeNet (2014)
VGG-VD-16 (2014)
VGG-M (2013)
AlexNet (2012)

ResNet 50 (2015)
ResNet 152 (2015)



16 convolutional layers

50 convolutional layers

152 convolutional layers

Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

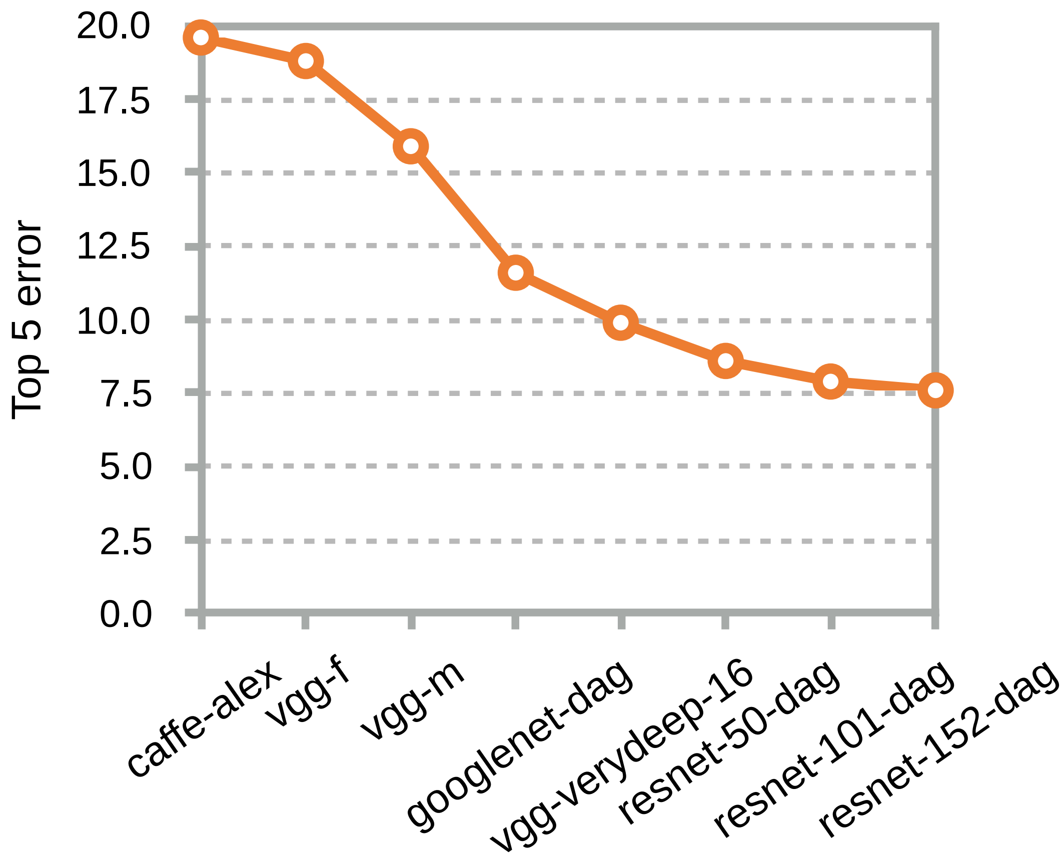
C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

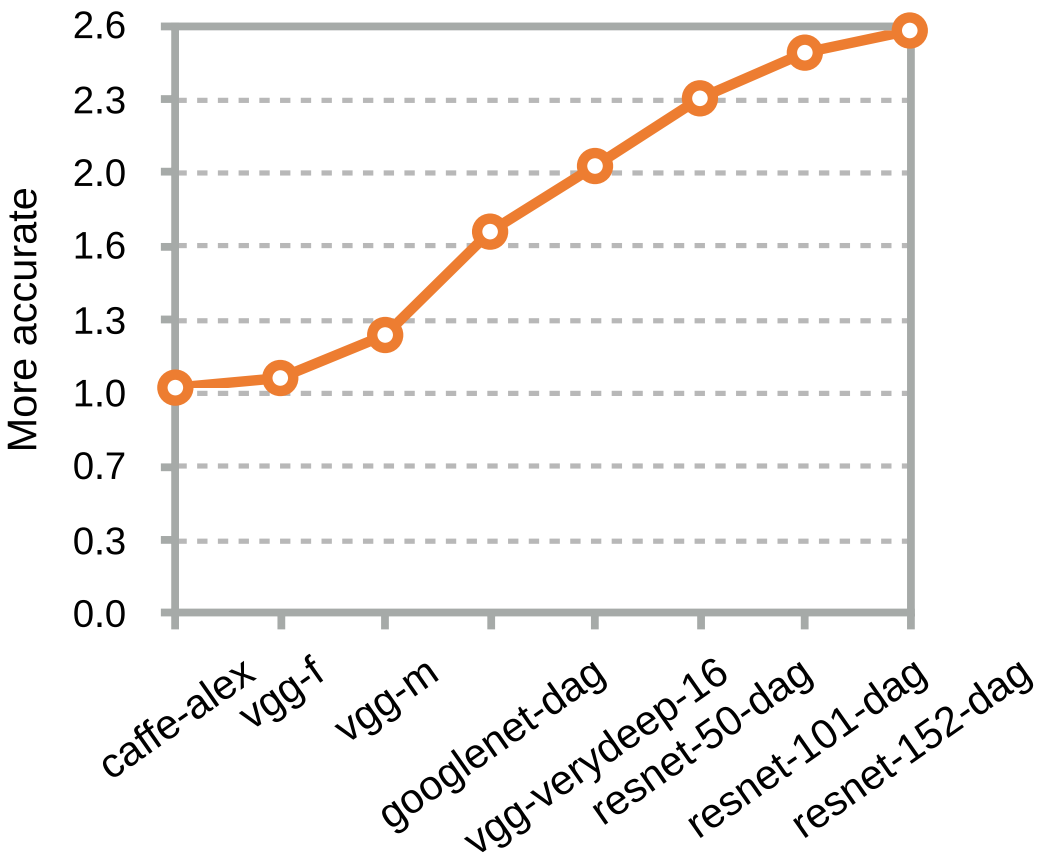
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

How deep is enough?

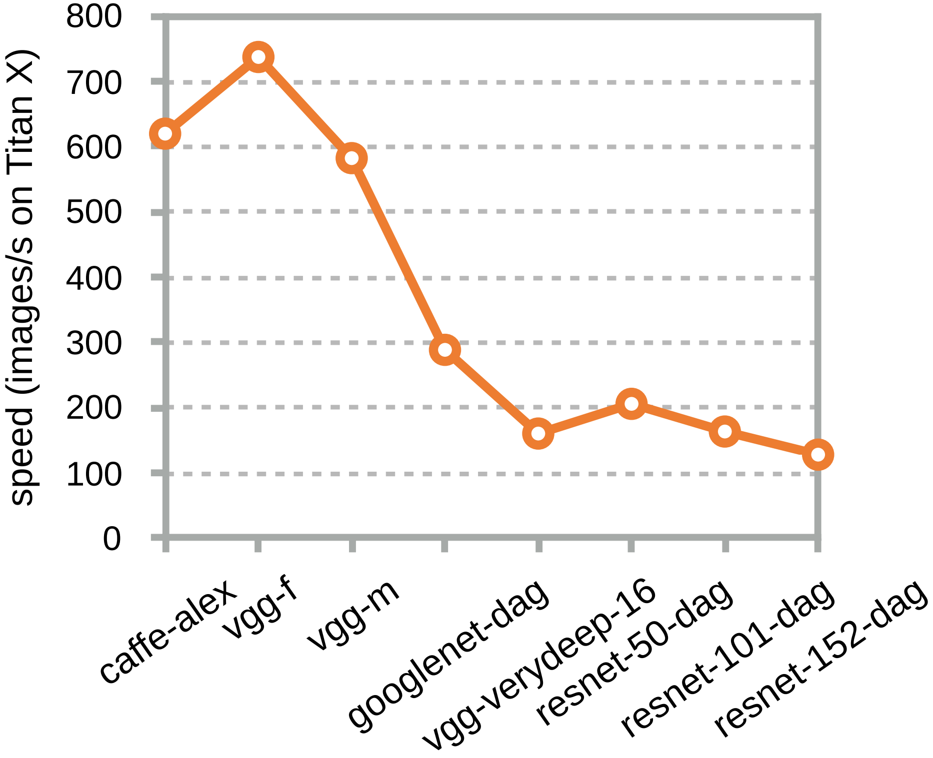
- 3 × more accurate in 3 years

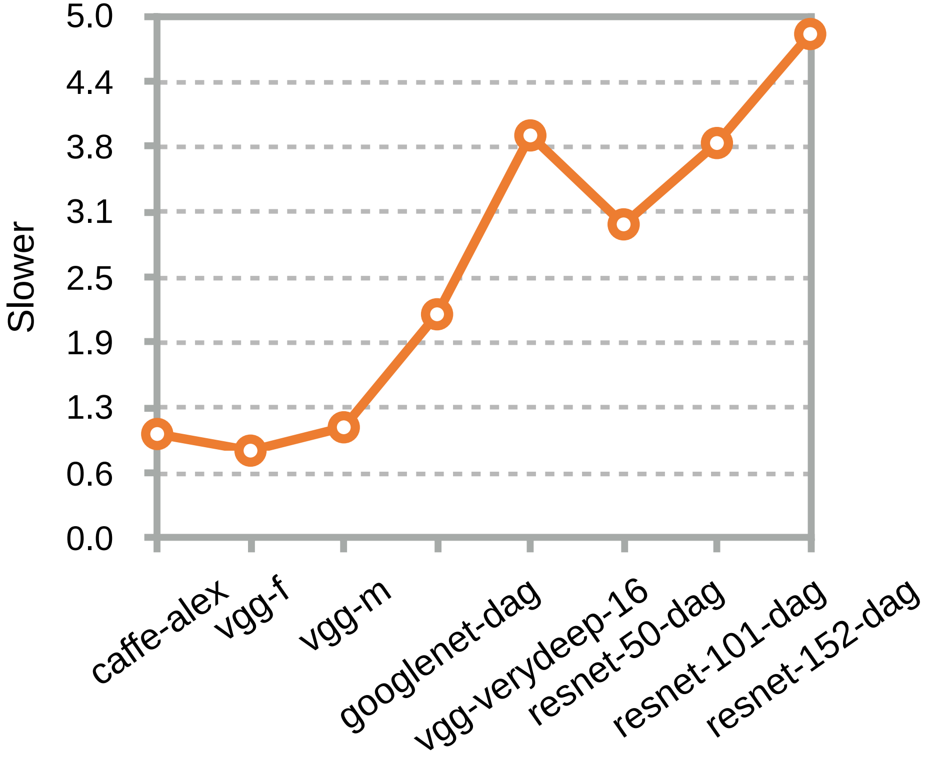




Speed

- 5 × slower

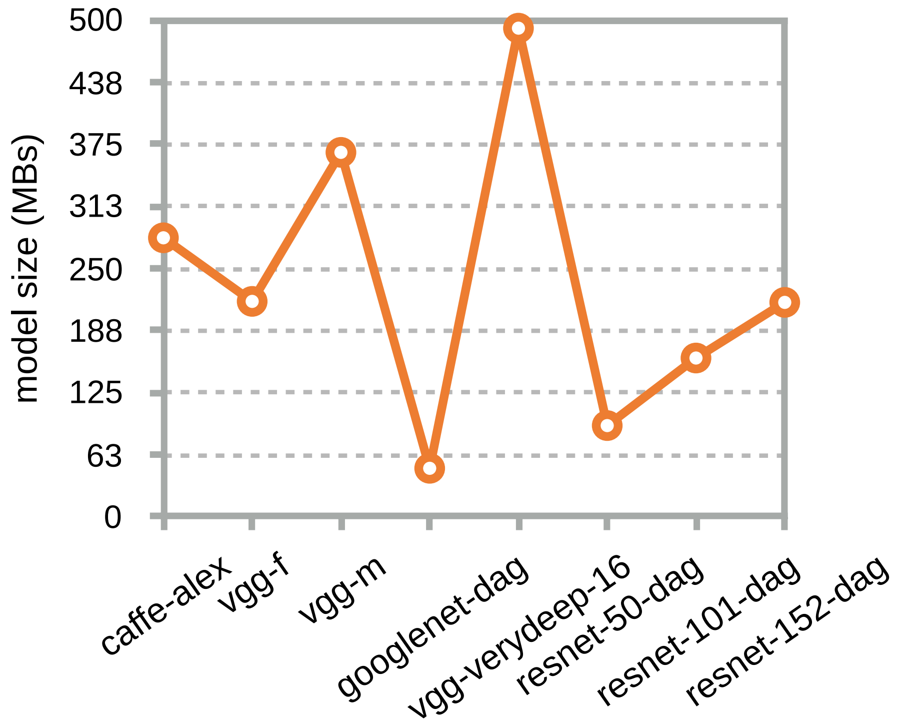


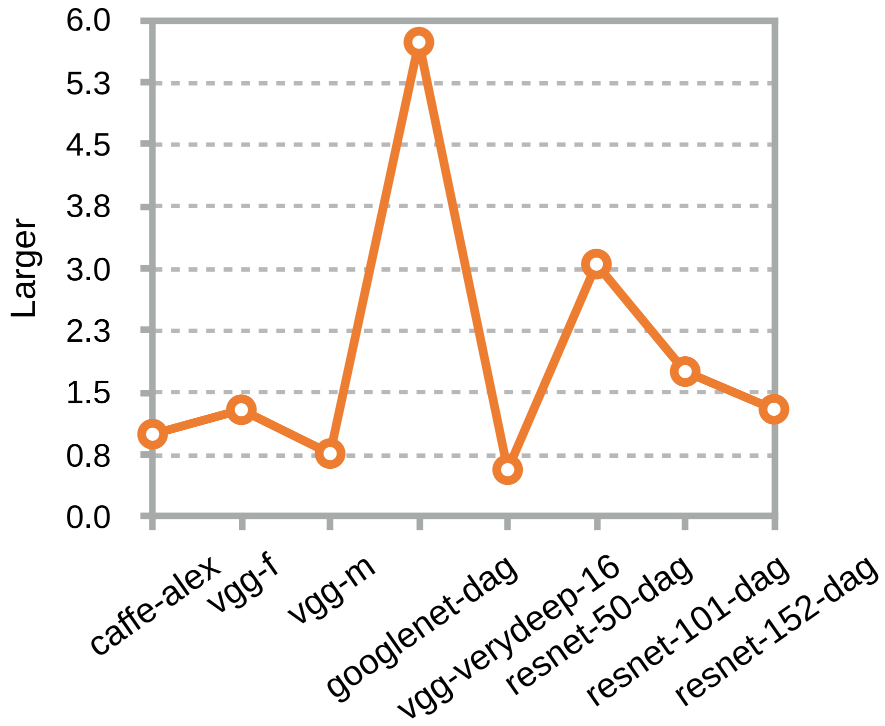


- **Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers
- **Reason:** Far fewer feature channels (quadratic speed/space gain)
- **Moral:** Optimize your architecture

Model Size

- Num. of parameters is about the same

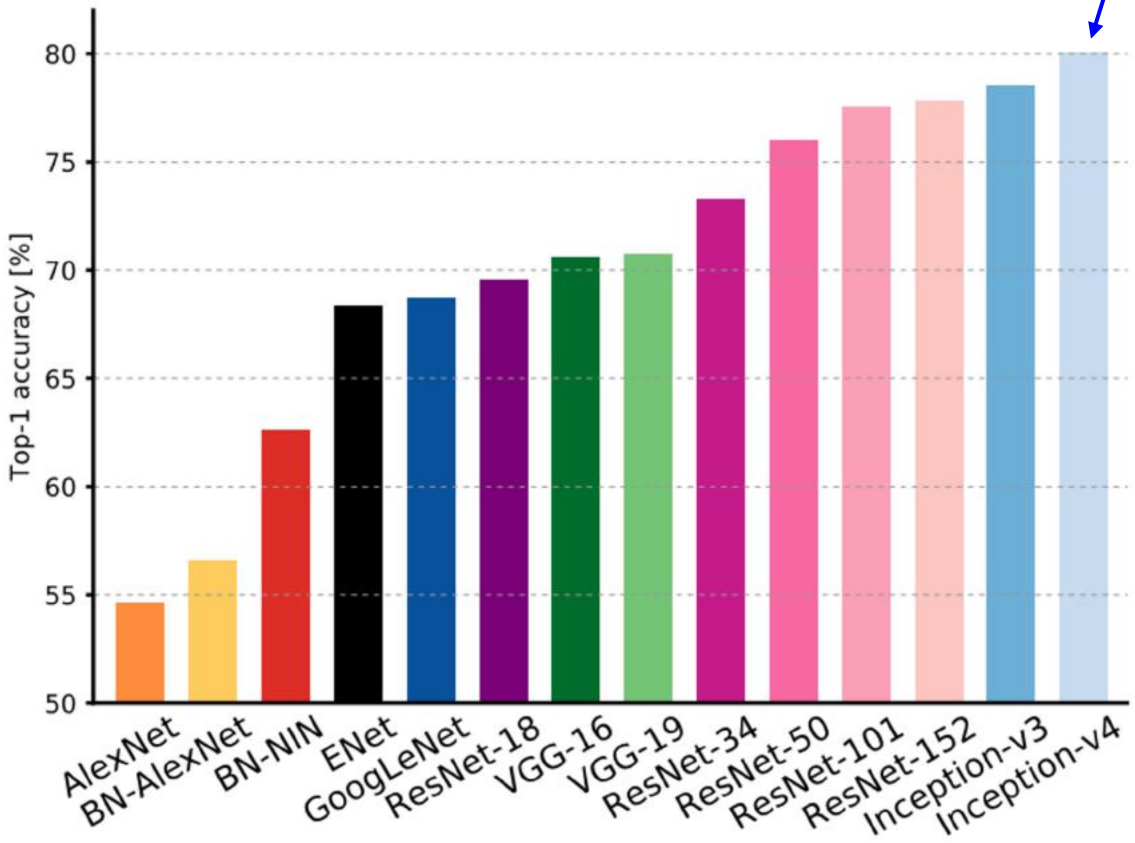


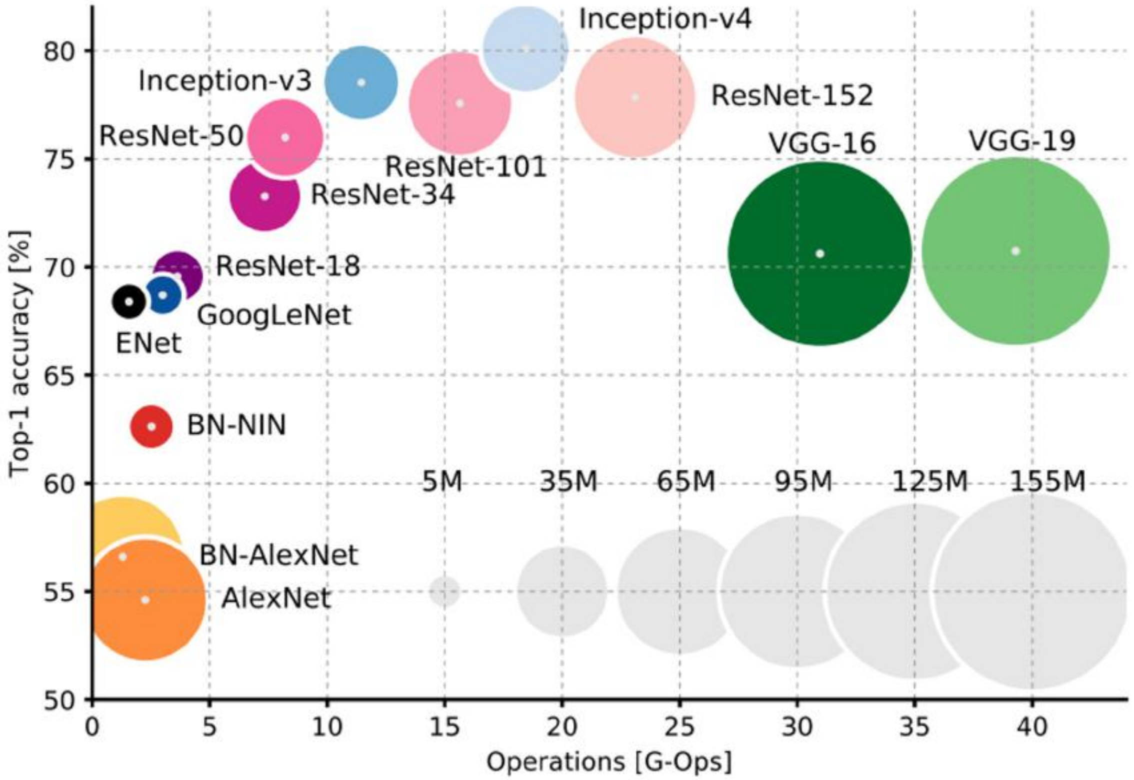


- **Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers
- **Reason:** Far fewer feature channels (quadratic speed/space gain)
- **Moral:** Optimize your architecture

Comparing Complexity

Inception-v4: Resnet + Inception!

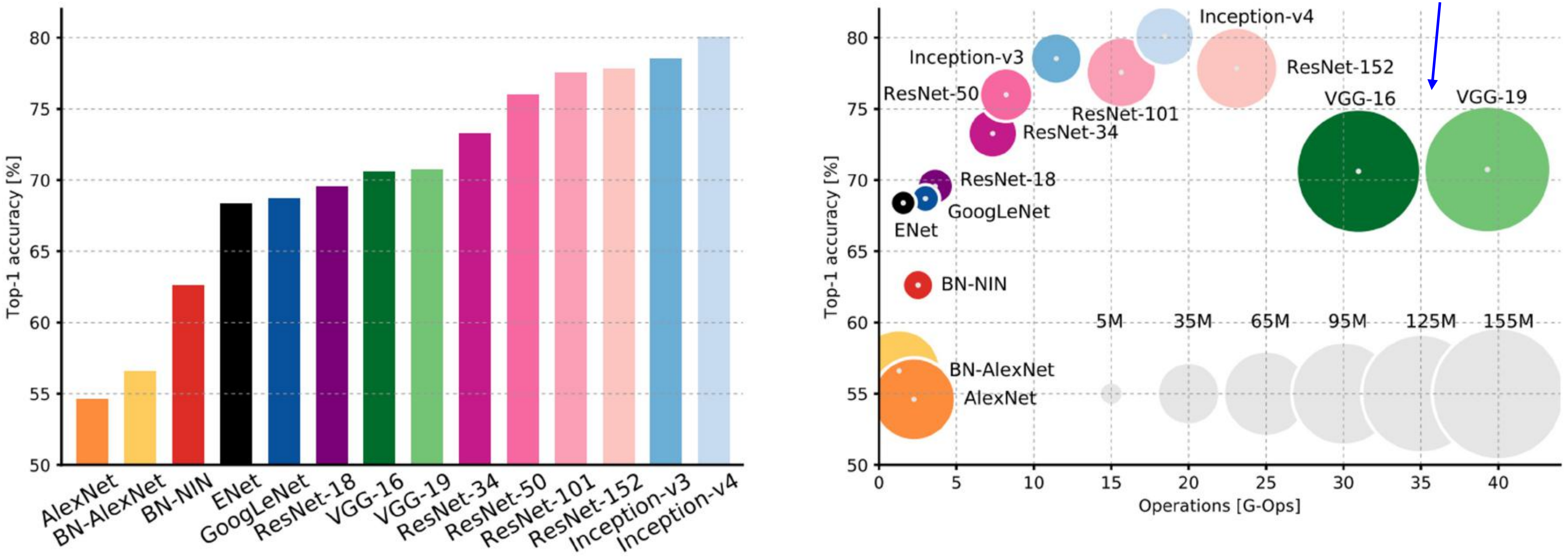




Canziani et al, "An analysis of deep neural network models for practical applications", 2017

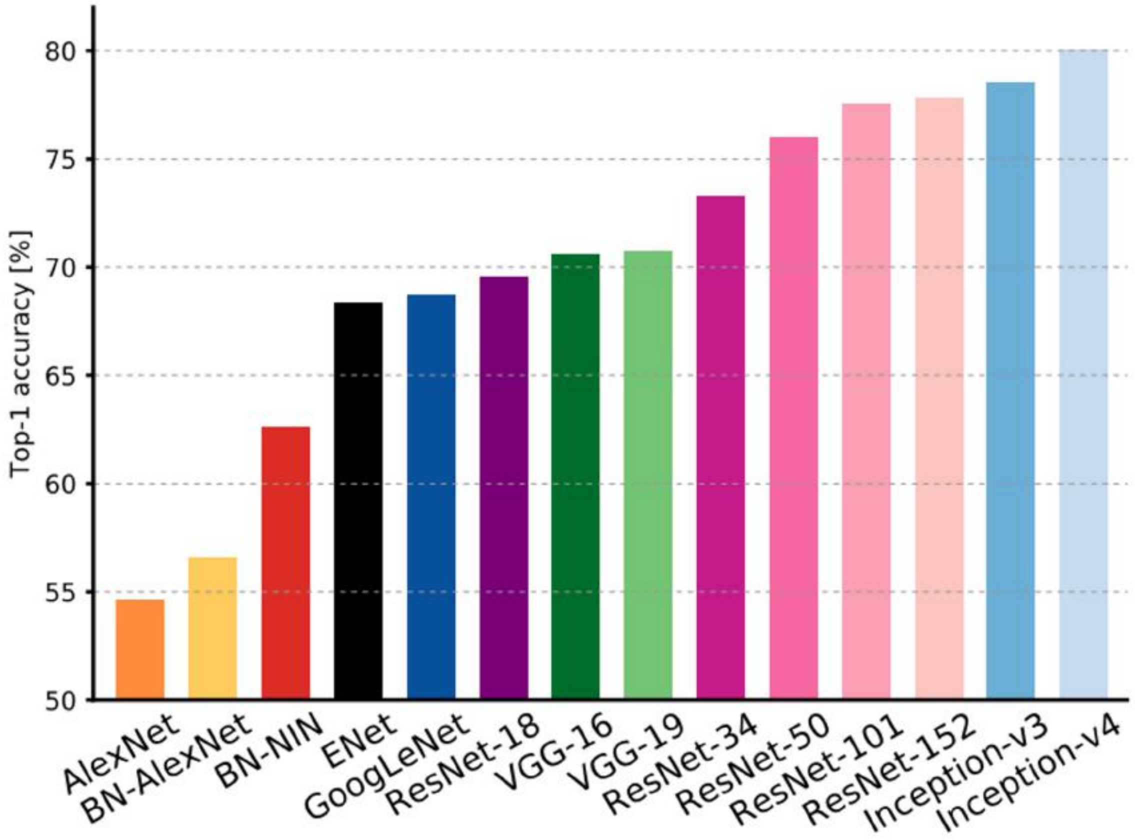
Comparing Complexity

VGG: Highest memory, most operations

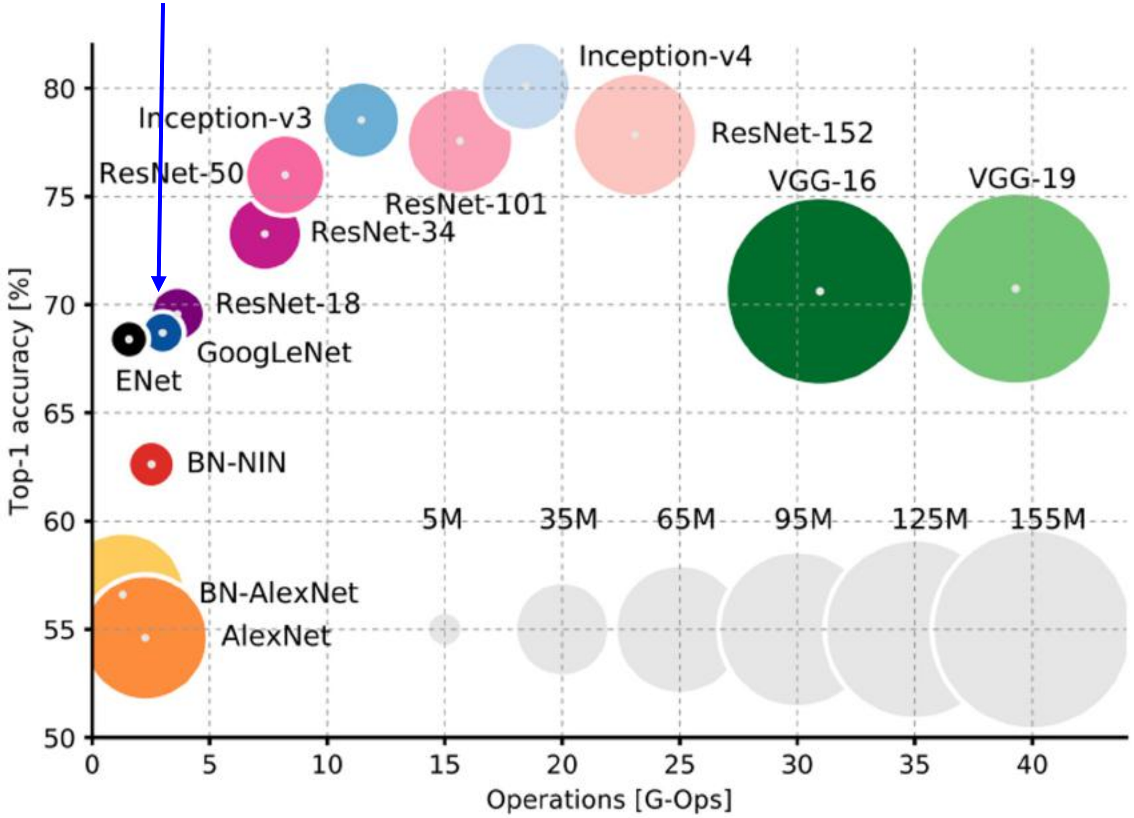


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity



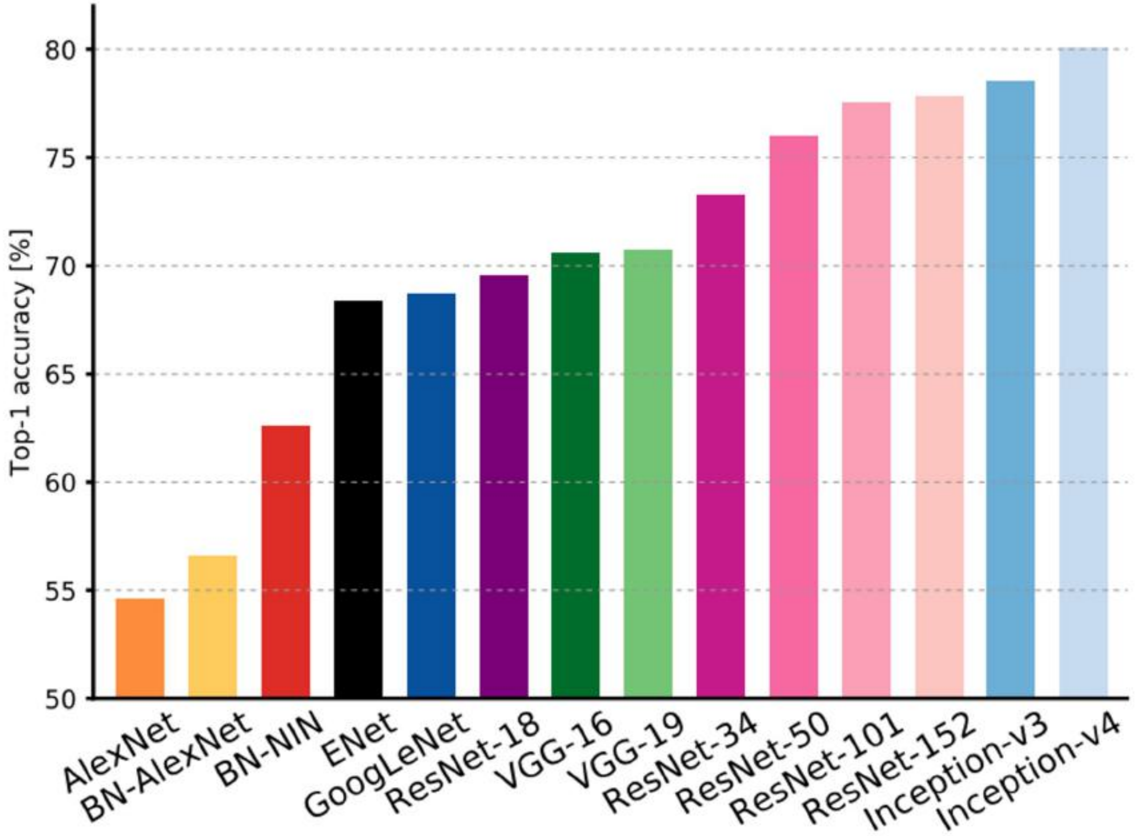
GoogLeNet:
Very efficient!

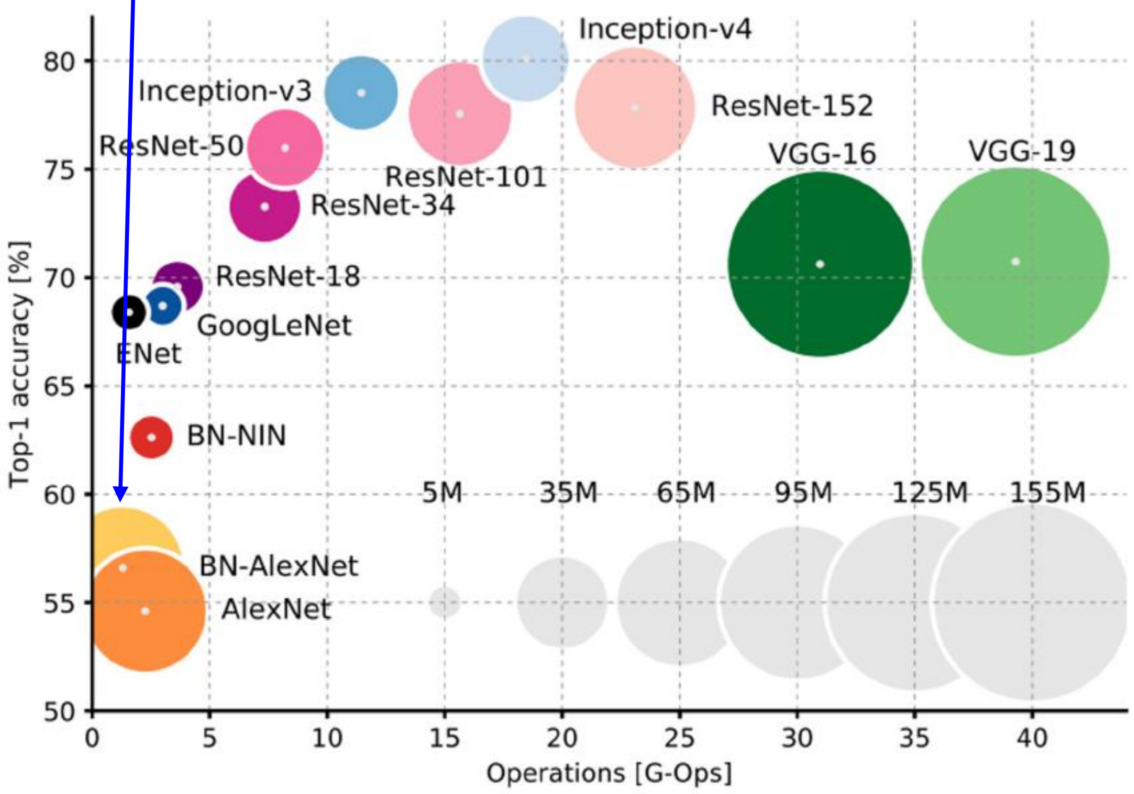


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

AlexNet: Low compute, lots of parameters

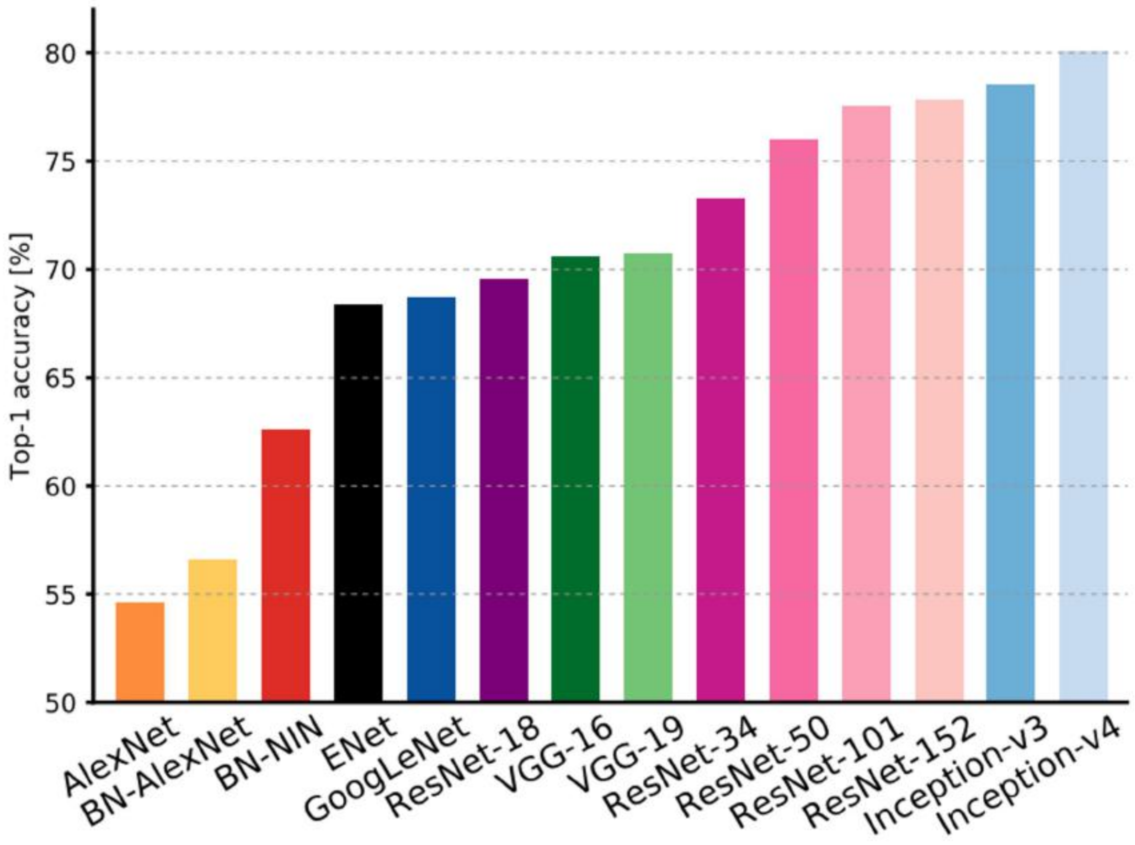


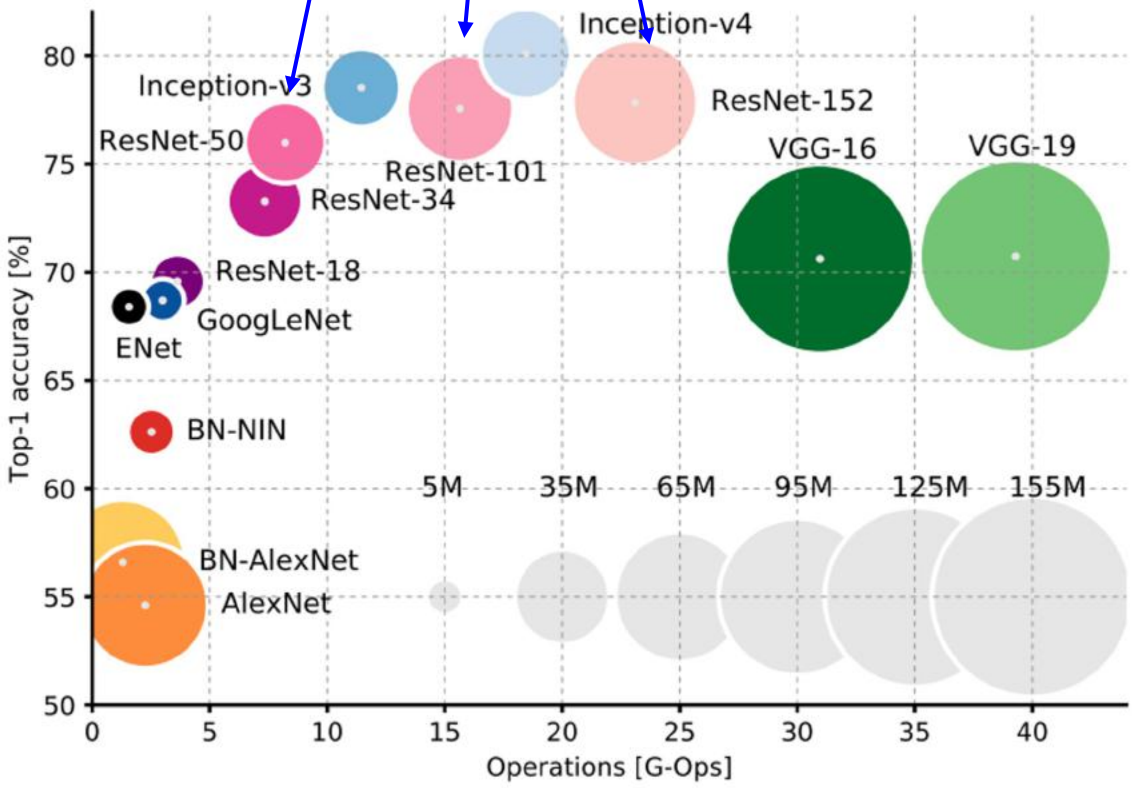


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

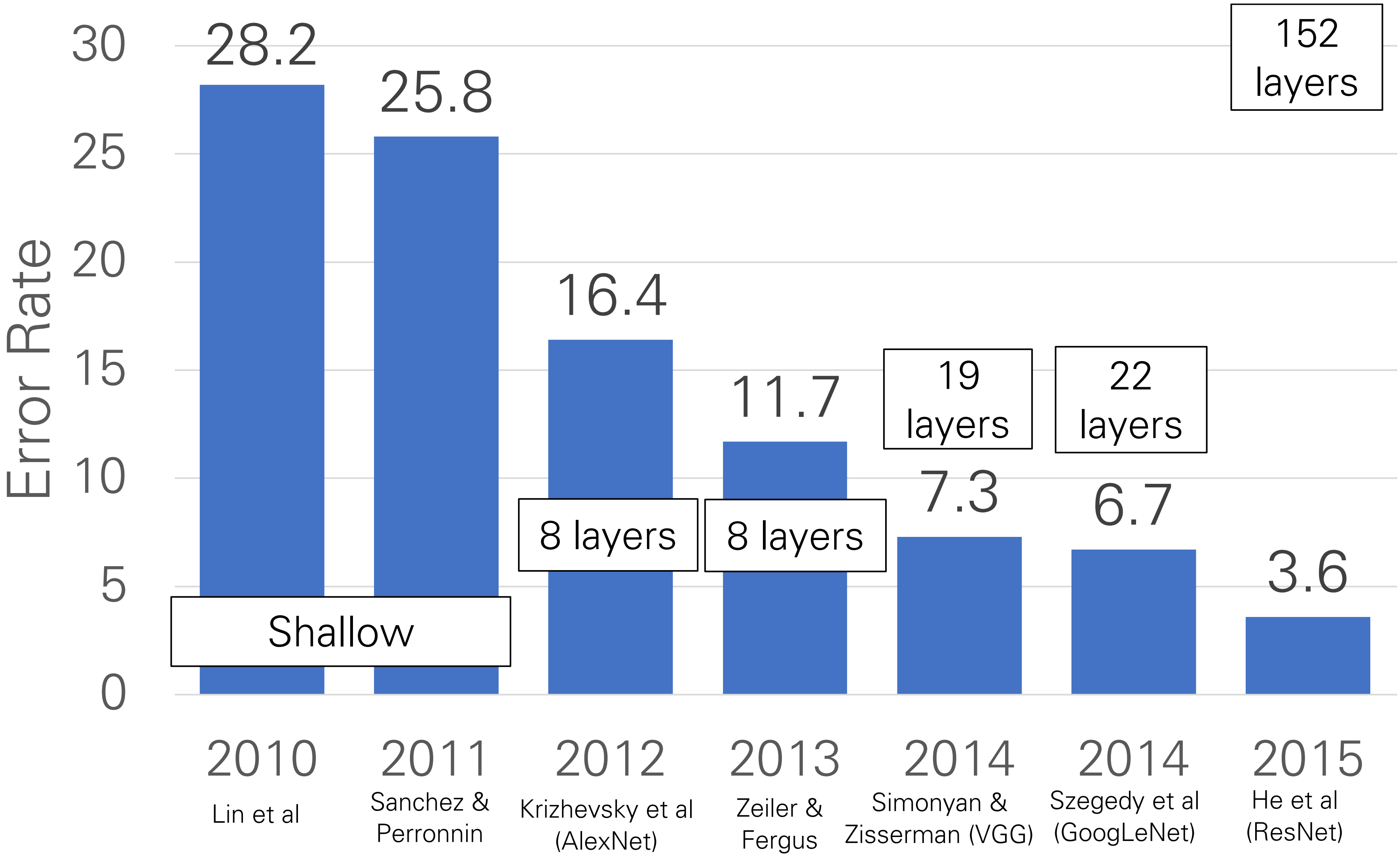
ResNet: Simple design,
moderate efficiency,
high accuracy





Canziani et al, "An analysis of deep neural network models for practical applications", 2017

ImageNet Classification Challenge

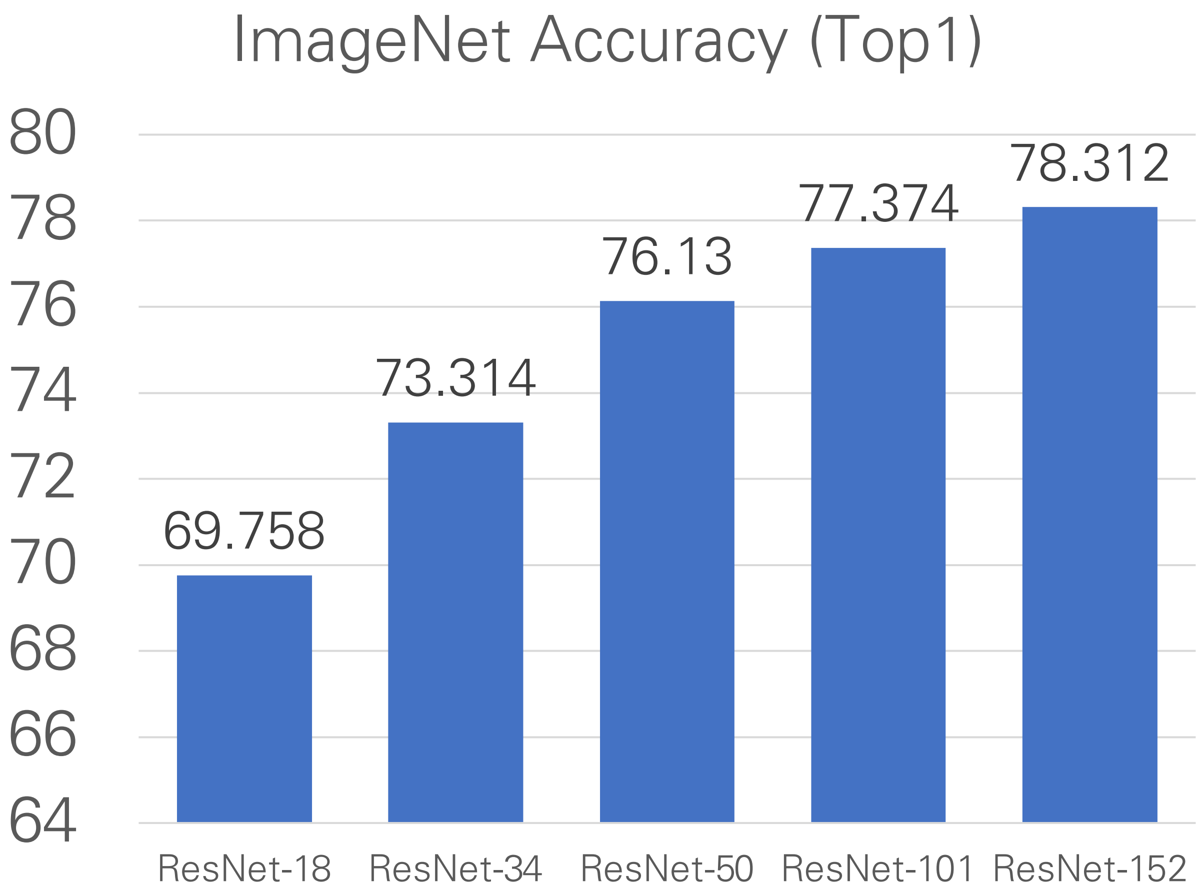


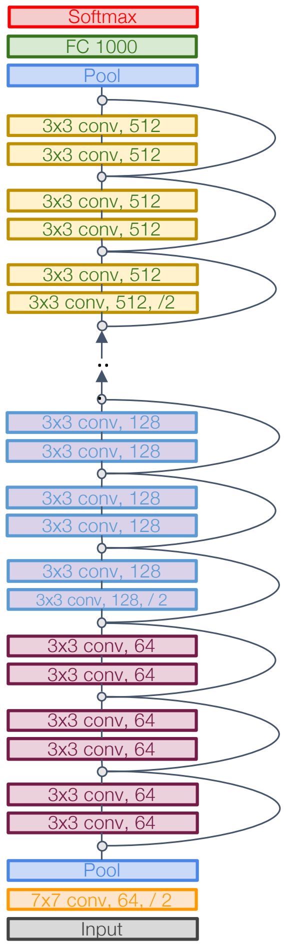
Today:
More recent CNN architectures

Post-ResNet Architectures

ResNet made it possible to increase accuracy with larger, deeper models

Many followup architectures emphasize **efficiency**: can we improve accuracy while controlling for model "complexity"?





Measures of Model Complexity

Parameters: How many learnable parameters does the model have?

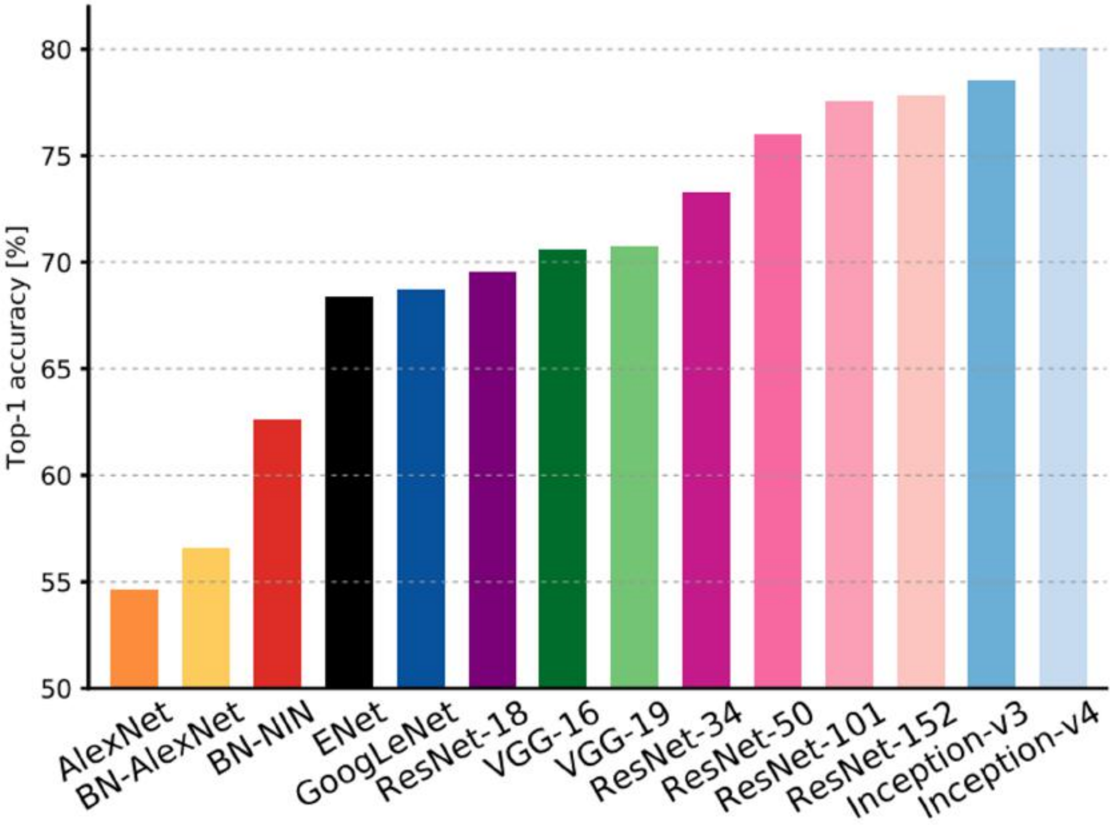
Floating Point Operations (FLOPs): How many arithmetic operations does it take to compute the forward pass of the model?

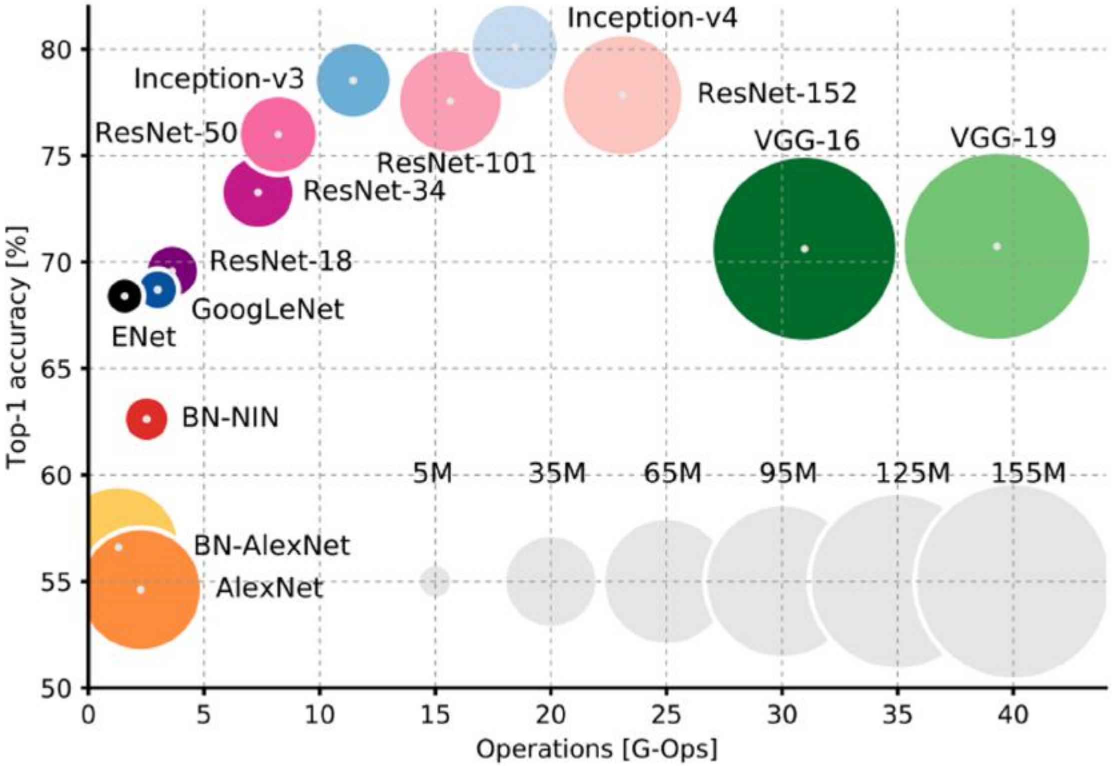
Watch out, lots of subtlety here:

- Many papers only count operations in conv layers (ignore ReLU, pooling, BatchNorm). Most papers use "1 FLOP" = "1 multiply and 1 addition" so dot product of two N-dim vectors takes N FLOPs; some papers say MADD or MACC instead of FLOP
- Other sources (e.g. NVIDIA marketing material) count "1 multiply and one addition" = 2 FLOPs, so dot product of two N-dim vectors takes 2N FLOPs

Network Runtime: How long does a forward pass of the model take on real hardware?

Comparing Complexity



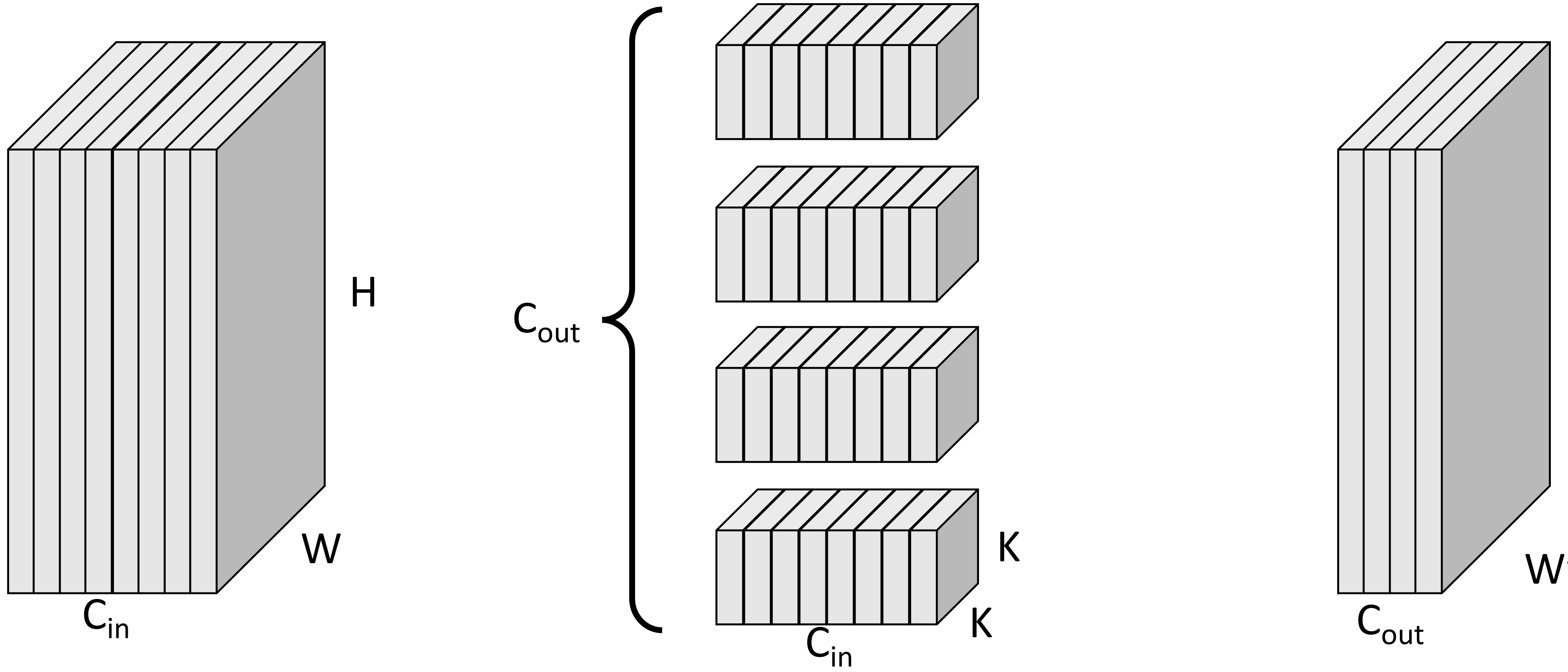


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Key ingredient:
Grouped / Separable convolution

Recall: Convolution Layer

Each filter has the same number of channels as the input



Input: C_{in} x H x W

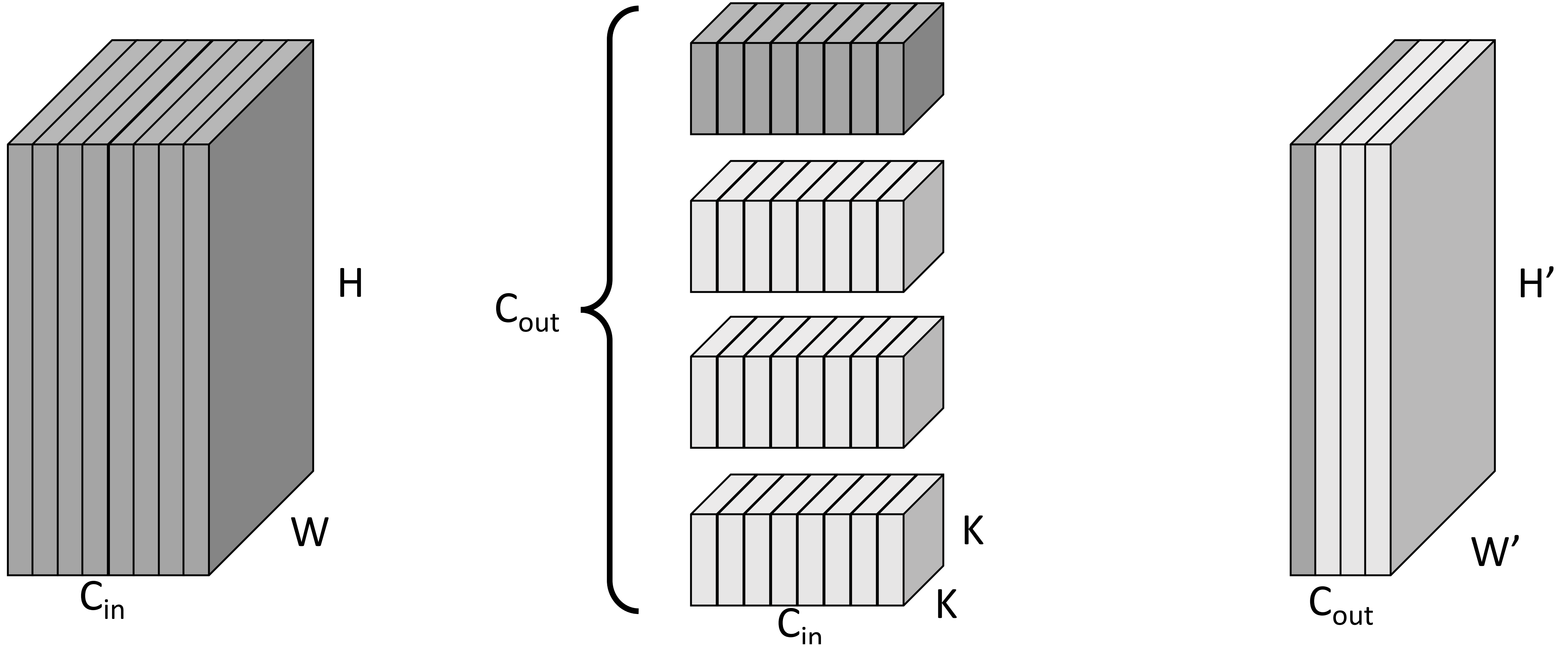
Weights: C_{out} x C_{in} x K x K

Output: C_{out} x H' x W'

Recall: Convolution Layer

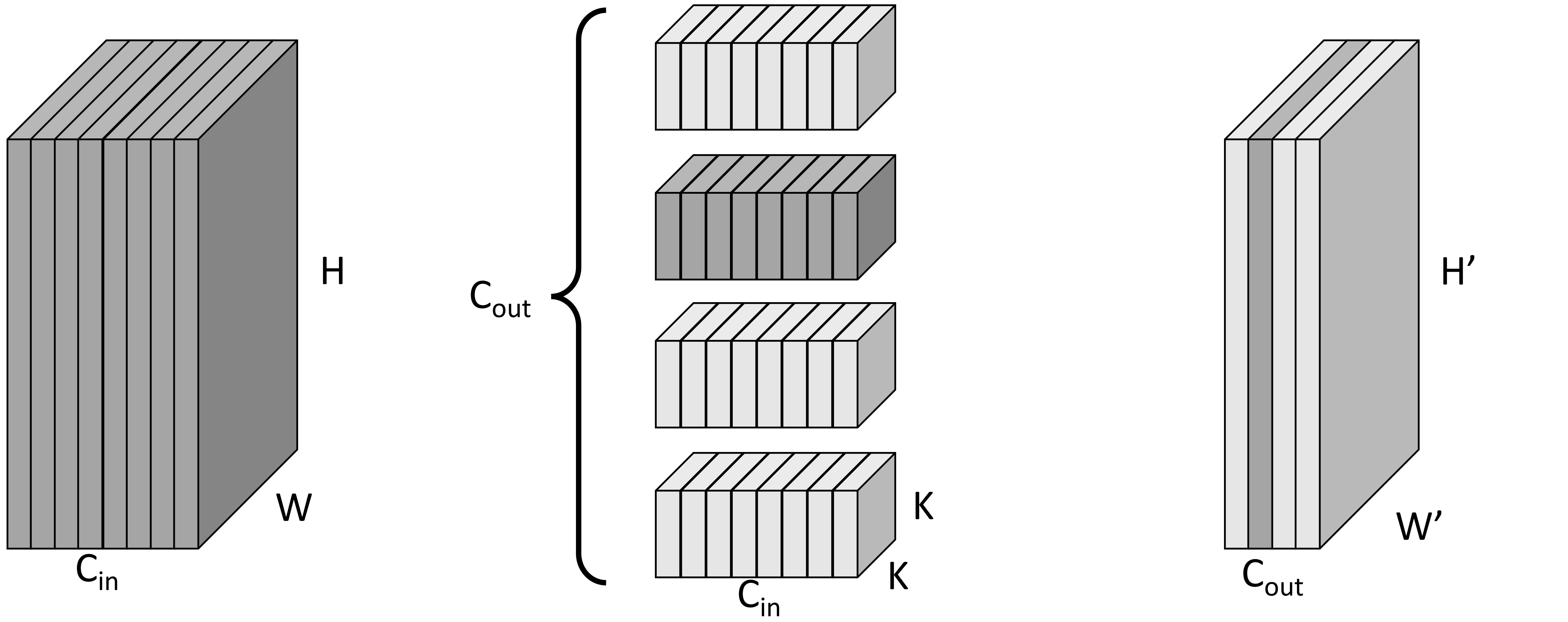
Each filter has the same number of channels as the input

Each plane of the output depends on the full input and one filter



Input: $C_{in} \times H \times W$ Weights: $C_{out} \times C_{in} \times K \times K$ Output: $C_{out} \times H' \times W'$

Recall: Convolution Layer



Input: C_{in} x H x W

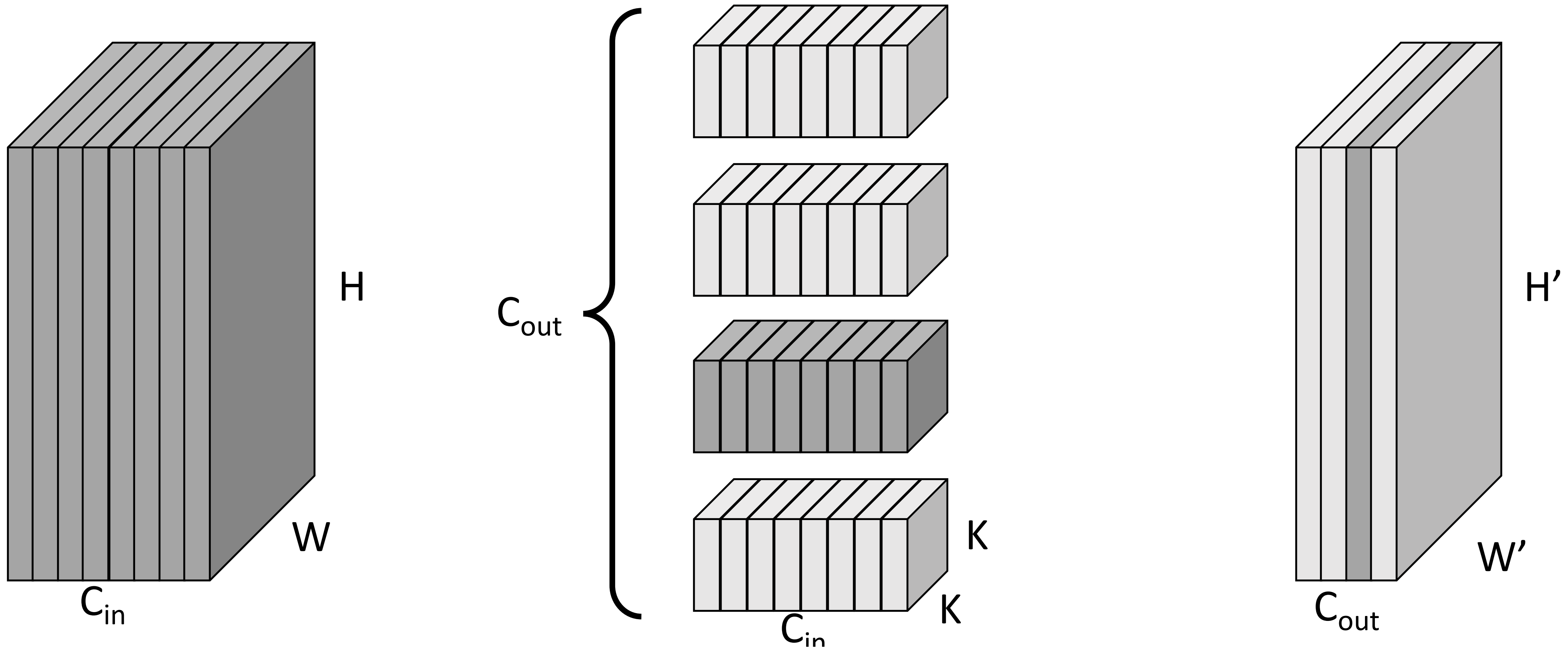
Weights: C_{out} x C_{in} x K x K

Output: C_{out} x H' x W'

Recall: Convolution Layer

Each filter has the same number of channels as the input

Each plane of the output depends on the full input and one filter



Input: C_{in} x H x W

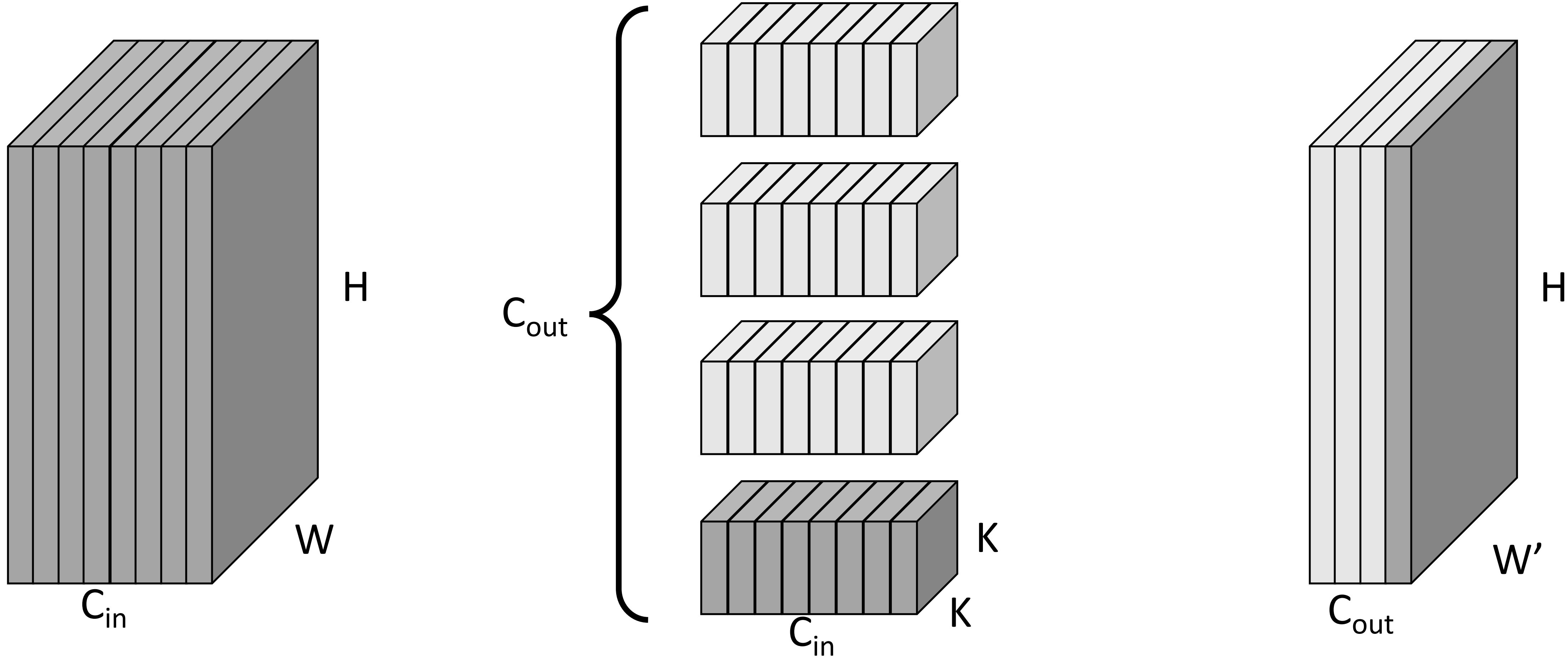
Weights: C_{out} x C_{in} x K x K

Output: C_{out} x H' x W'

Recall: Convolution Layer

Each filter has the same number of channels as the input

Each plane of the output depends on the full input and one filter



Input: C_{in} x H x W

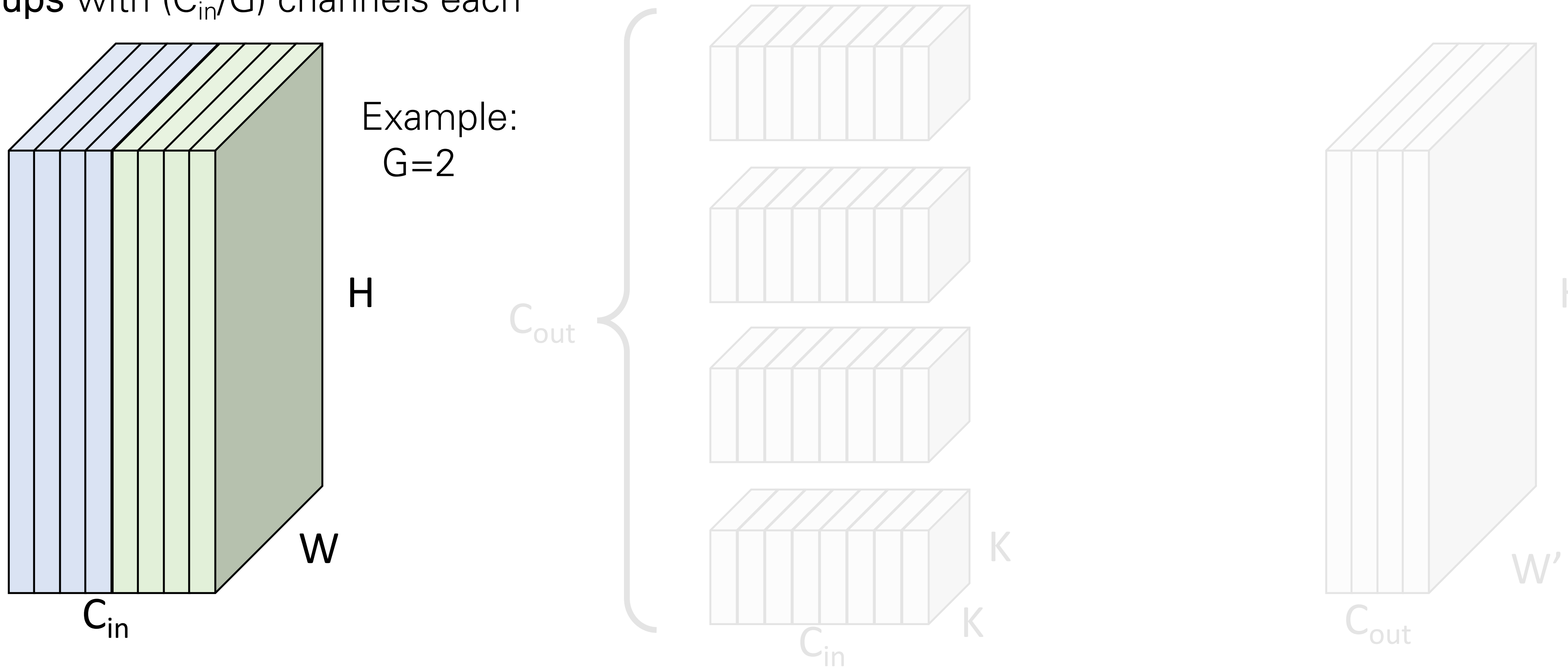
Weights: C_{out} x C_{in} x K x K

Output: C_{out} x H' x W'

Grouped Convolution

Grouped Convolution

Divide channels of input into G **groups** with (C_{in}/G) channels each



Input: C_{in} x H x W

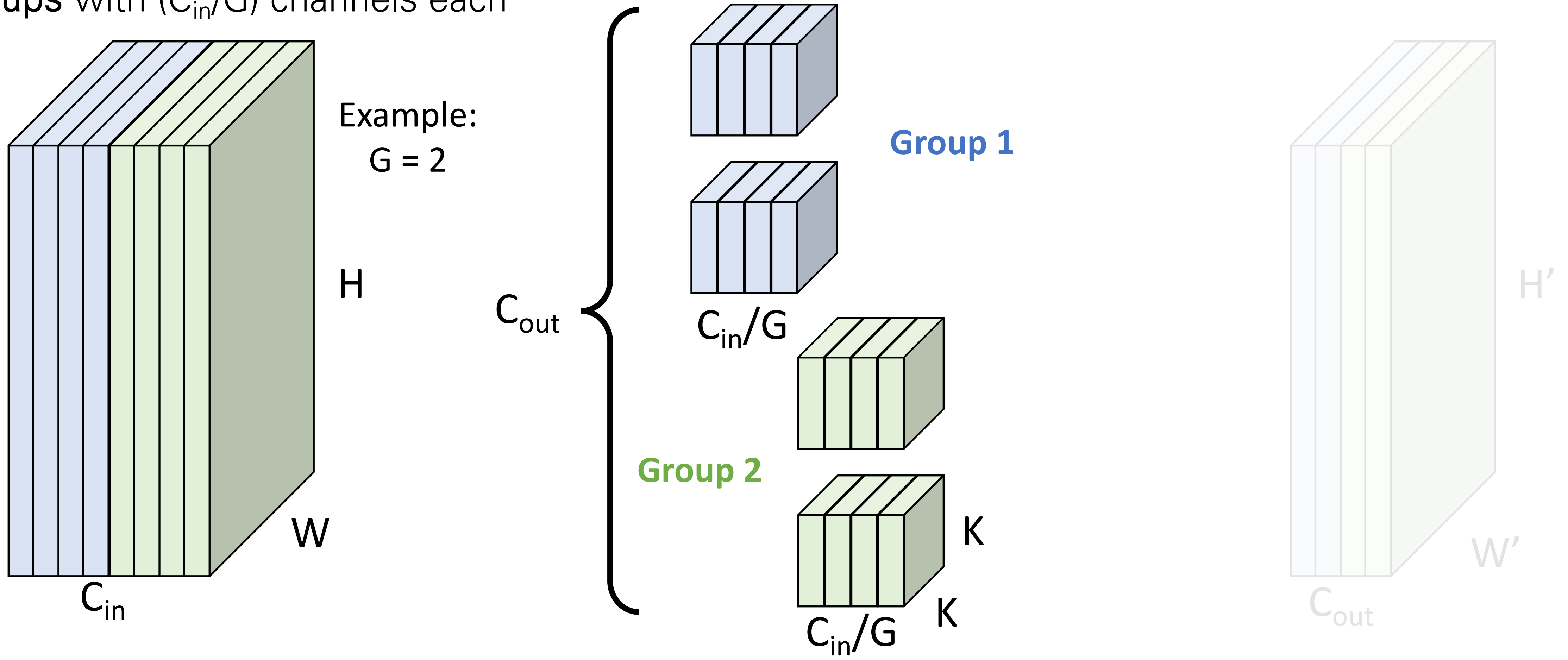
Weights: C_{out} x C_{in} x K x K

Output: C_{out} x H' x W'

Grouped Convolution

Divide filters into G groups; each group looks at a **subset** of input channels

Divide channels of input into G **groups** with (C_{in}/G) channels each



Input: C_{in} x H x W

Weights: C_{out} x (C_{in} /G) x K x K

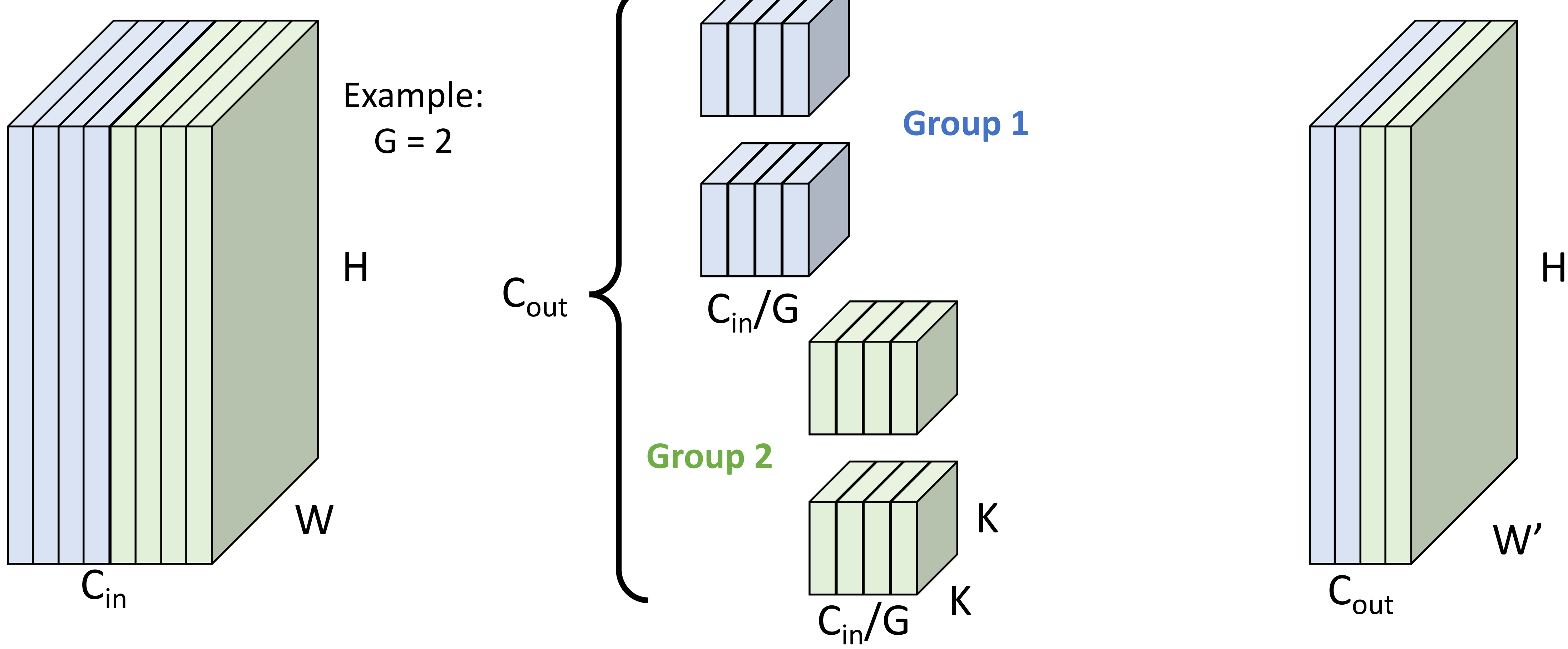
Output: C_{out} x H' x W'

Grouped Convolution

Divide filters into G groups; each group looks at a **subset** of input channels

Each plane of the output depends on one filter and a **subset** of the input channels

Divide channels of input into G **groups** with (C_{in}/G) channels each



Input: C_{in} x H x W

Weights: C_{out} x (C_{in} /G) x K x K

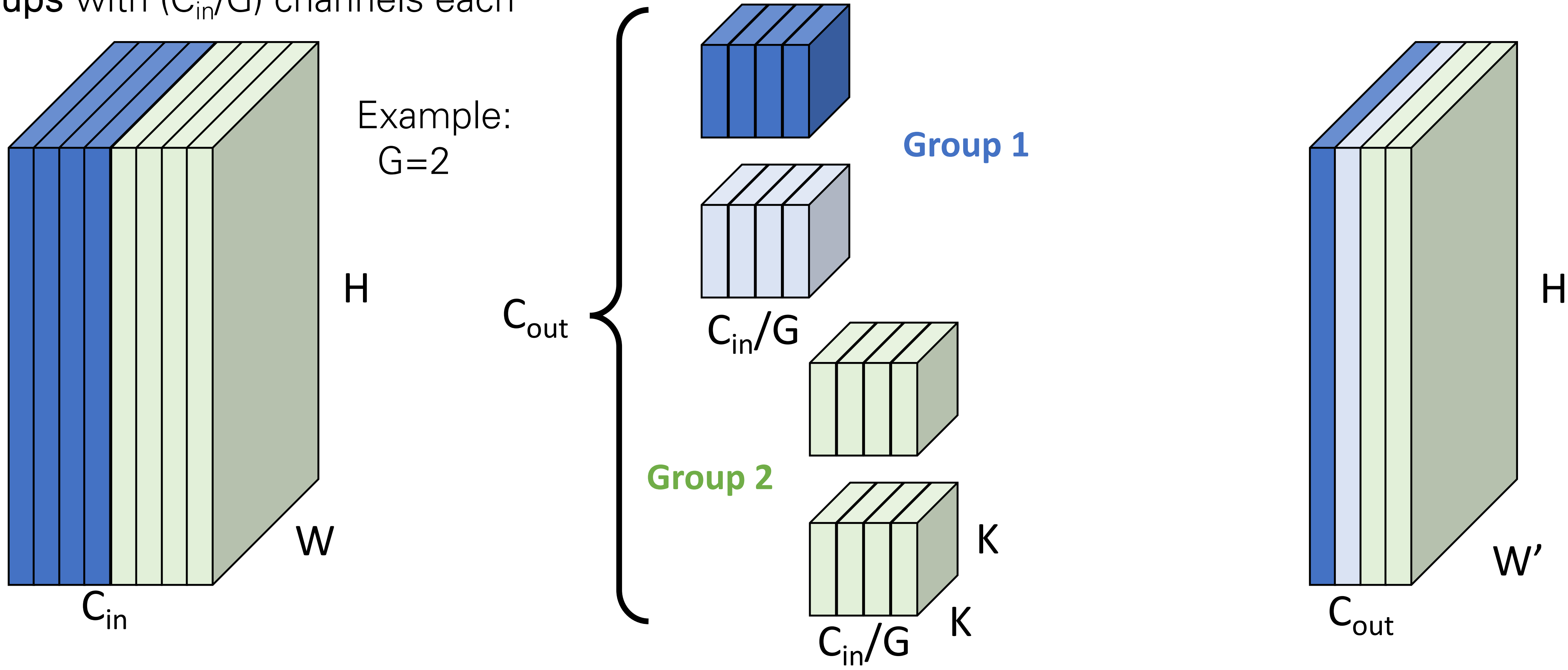
Output: C_{out} x H' x W'

Group Convolution

Divide channels of input into G **groups** with (C_{in}/G) channels each

Divide filters into G groups; each group looks at a **subset** of input channels

Each plane of the output depends on one filter and a **subset** of the input channels



Input: C_{in} x H x W

Weights: C_{out} x C_{in} x K x K

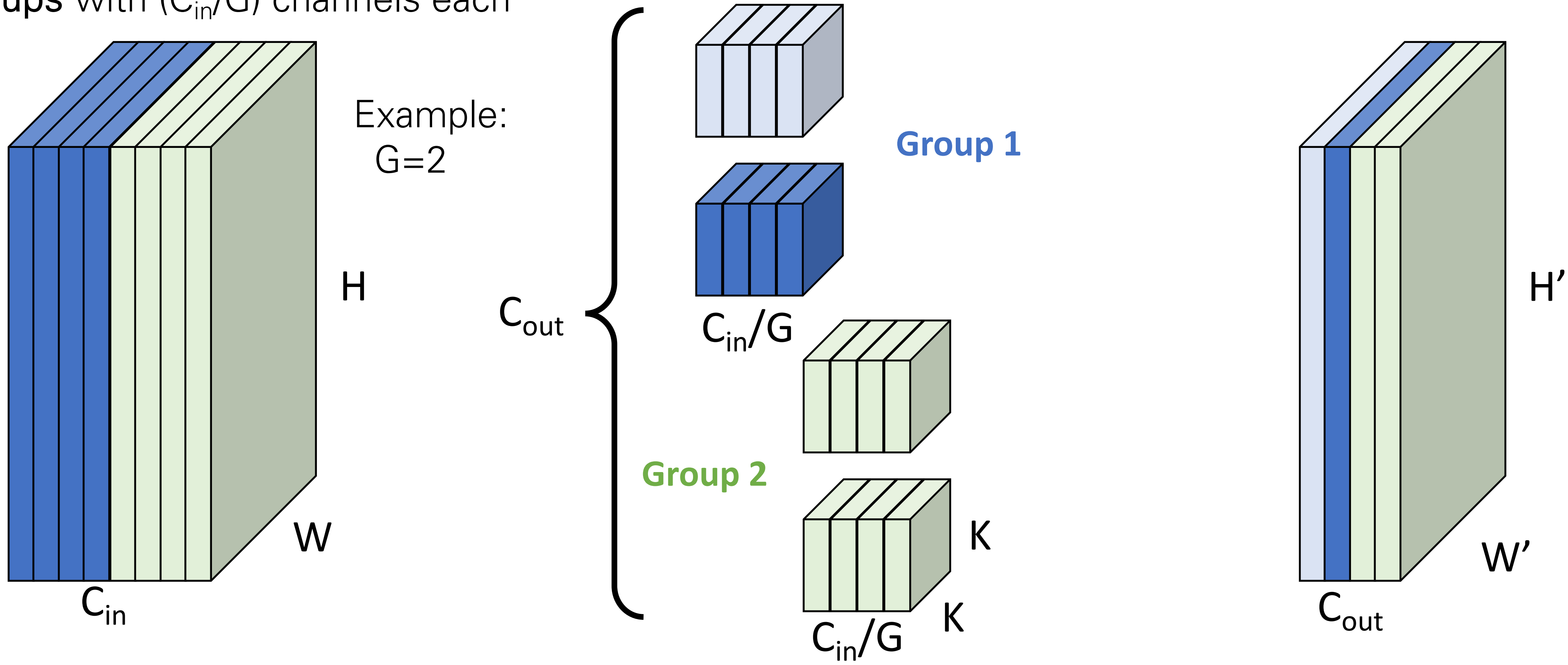
Output: C_{out} x H' x W'

Group Convolution

Divide channels of input into G **groups** with (C_{in}/G) channels each

Divide filters into G groups; each group looks at a **subset** of input channels

Each plane of the output depends on one filter and a **subset** of the input channels



Input: C_{in} x H x W

Weights: C_{out} x C_{in} x K x K

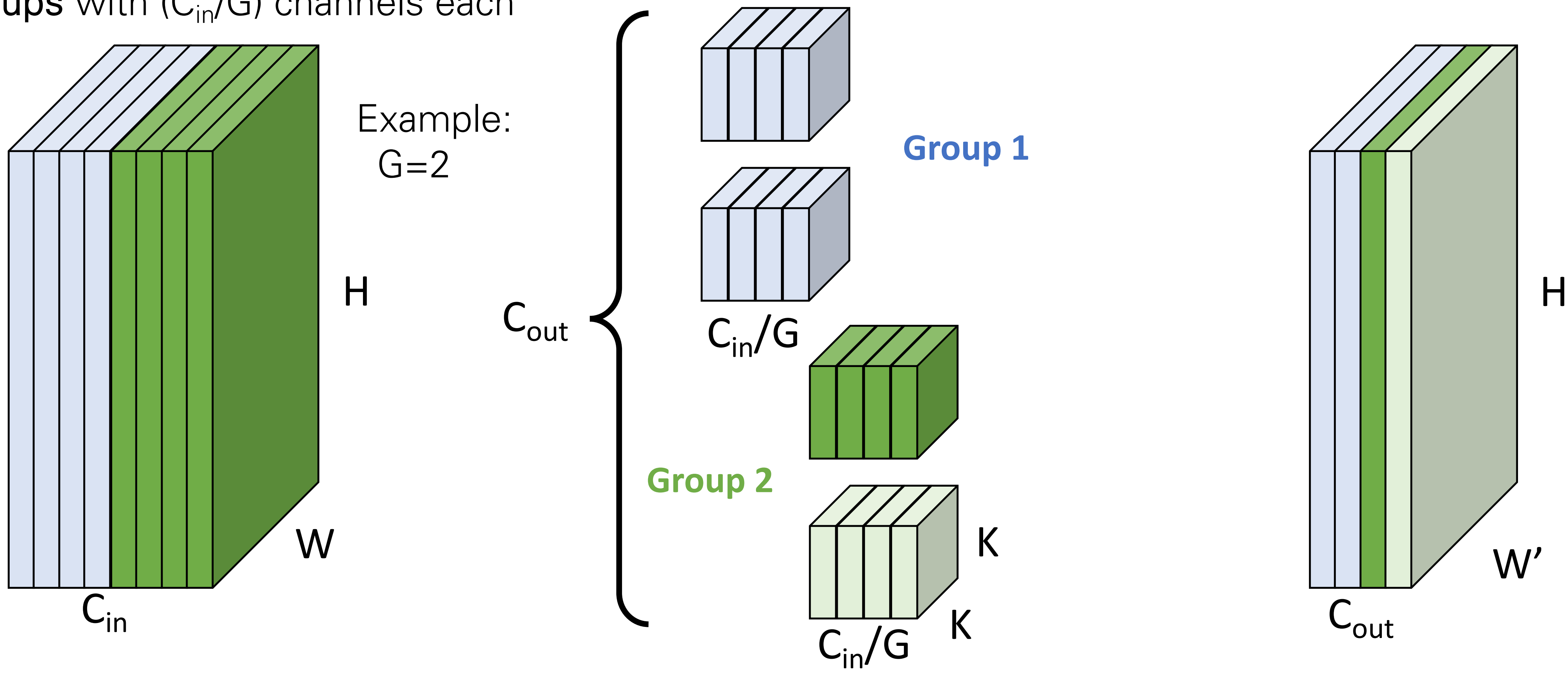
Output: C_{out} x H' x W'

Group Convolution

Divide filters into G groups; each group looks at a **subset** of input channels

Each plane of the output depends on one filter and a **subset** of the input channels

Divide channels of input into G **groups** with (C_{in}/G) channels each



Input: C_{in} x H x W

Weights: C_{out} x C_{in} x K x K

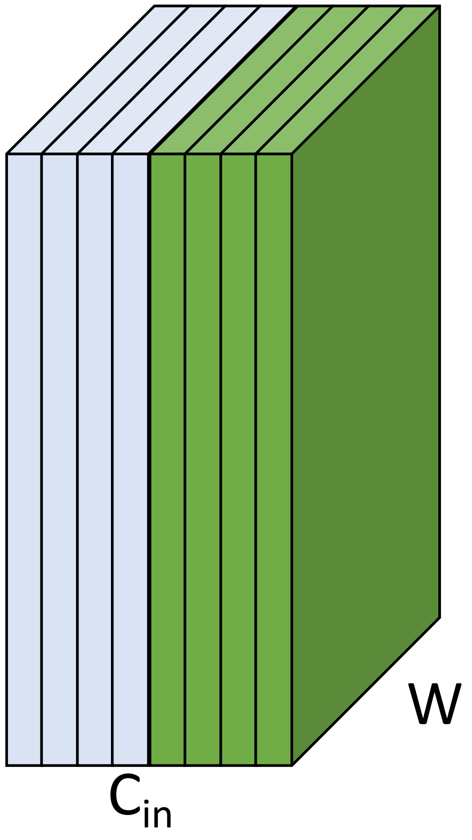
Output: C_{out} x H' x W'

Group Convolution

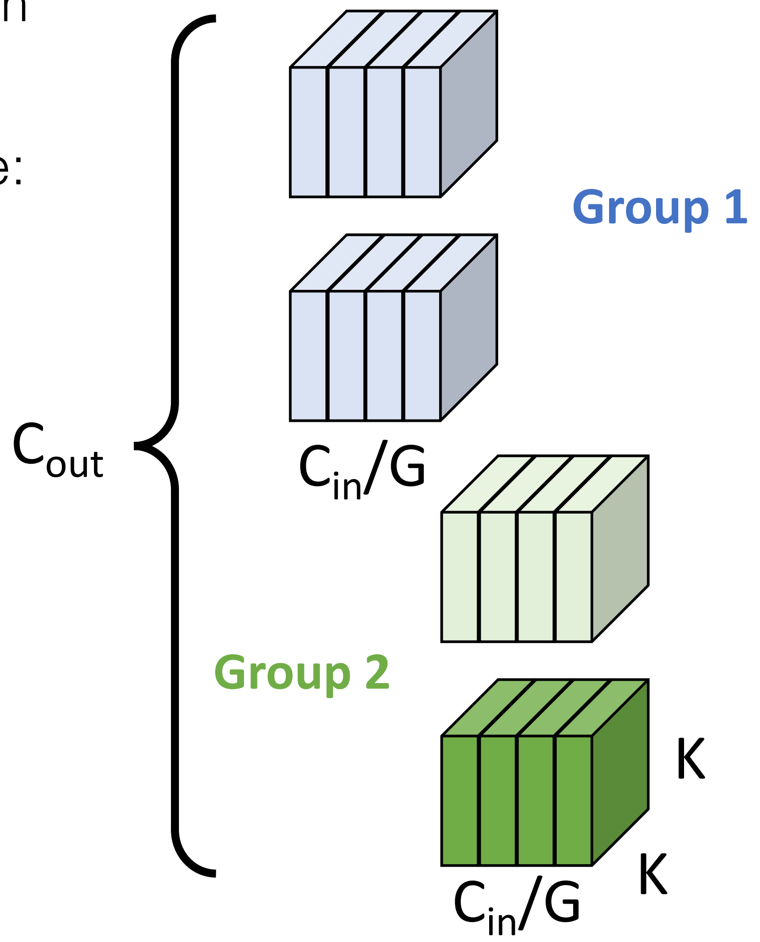
Divide filters into G groups; each group looks at a **subset** of input channels

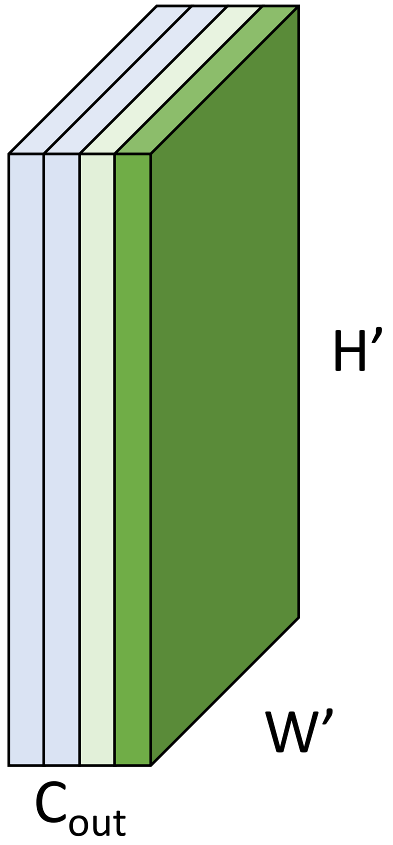
Each plane of the output depends on one filter and a **subset** of the input channels

Divide channels of input into G **groups** with (C_{in}/G) channels each



Example:
G=2





Input: C_{in} x H x W

Weights: C_{out} x C_{in} x K x K

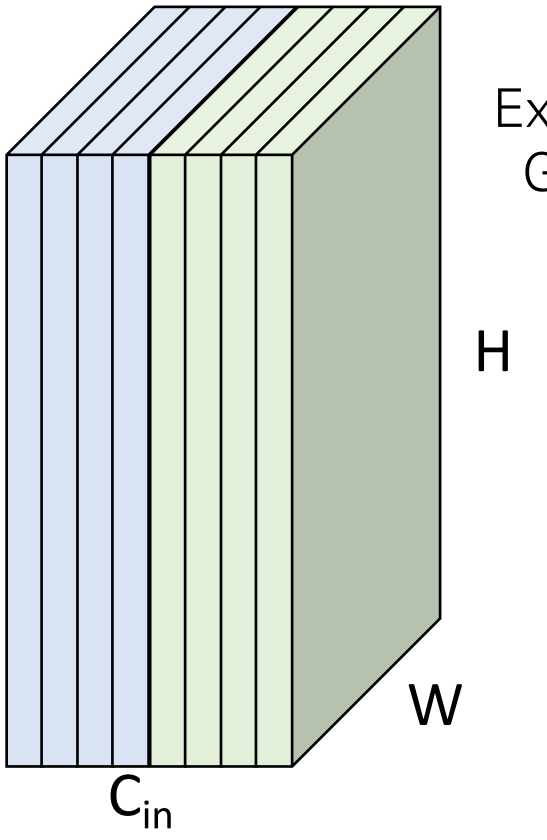
Output: C_{out} x H’ x W’

Group Convolution

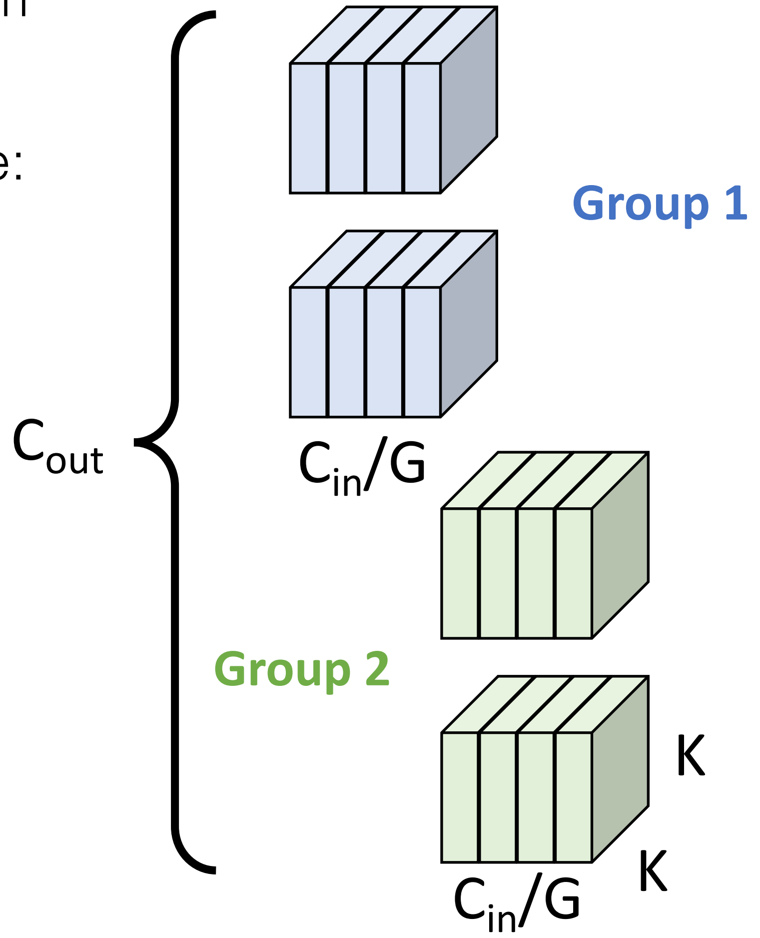
Divide filters into G groups; each group looks at a **subset** of input channels

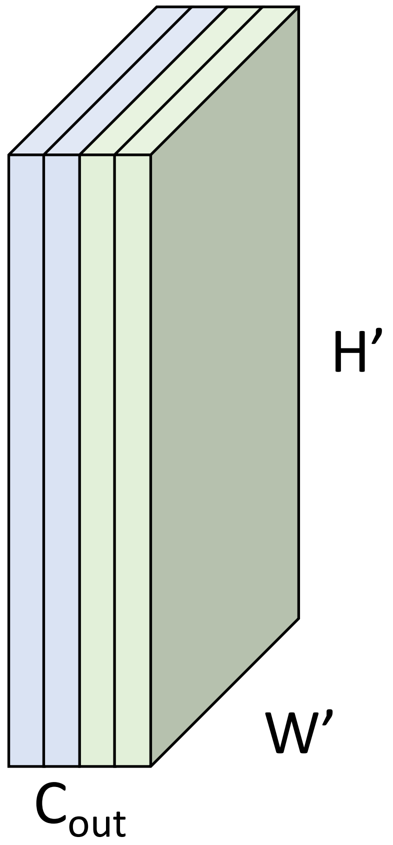
Each plane of the output depends on one filter and a **subset** of the input channels

Divide channels of input into G **groups** with (C_{in}/G) channels each



Example:
G=2





Input: C_{in} x H x W

Weights: C_{out} x C_{in} x K x K

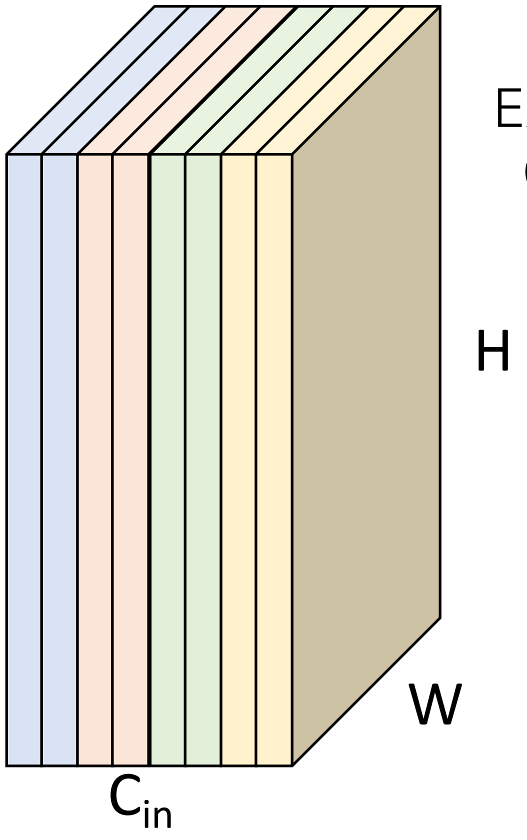
Output: C_{out} x H' x W'

Group Convolution

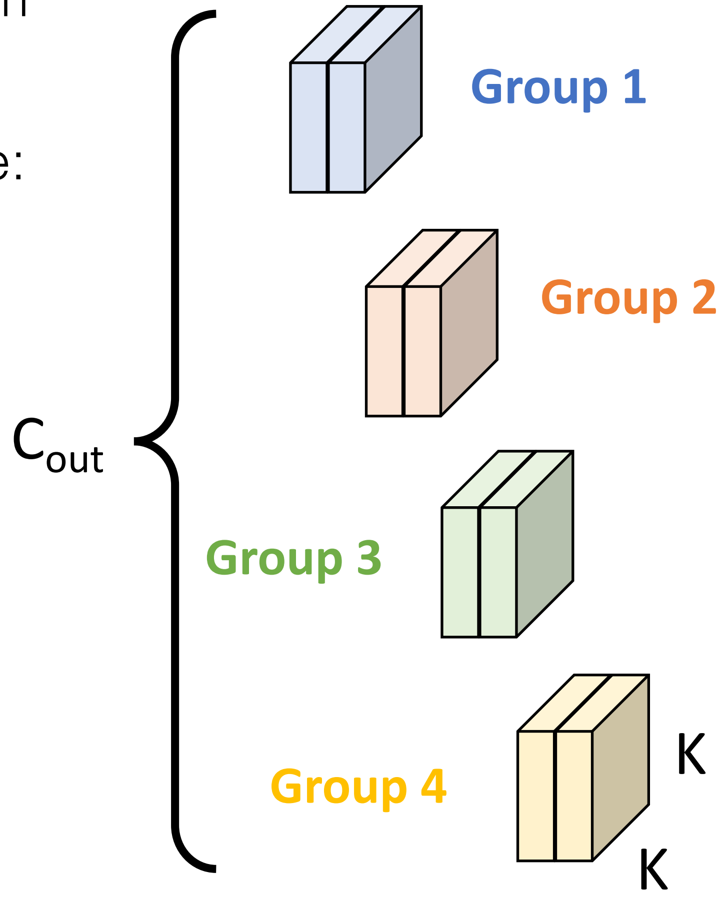
Divide channels of input into G **groups** with (C_{in}/G) channels each

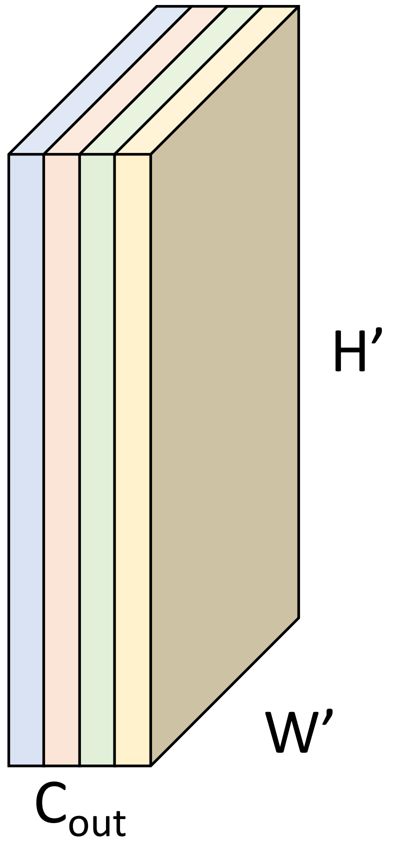
Divide filters into G groups; each group looks at a **subset** of input channels

Each plane of the output depends on one filter and a **subset** of the input channels



Example:
G=4





Input: C_{in} x H x W

Weights: C_{out} x C_{in} x K x K

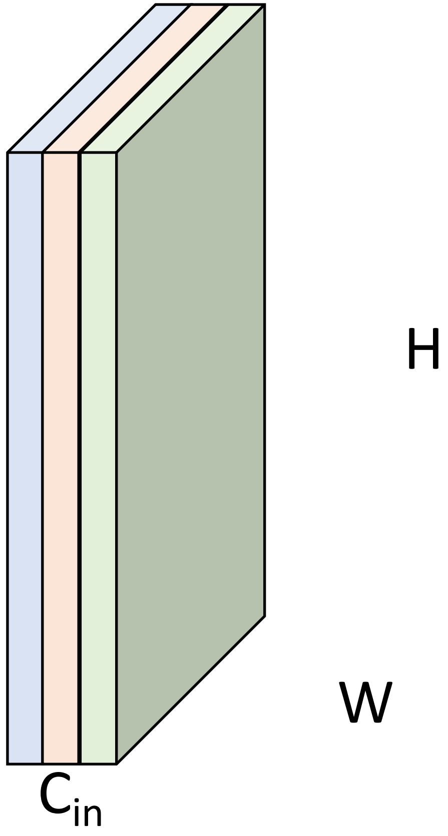
Output: C_{out} x H’ x W’

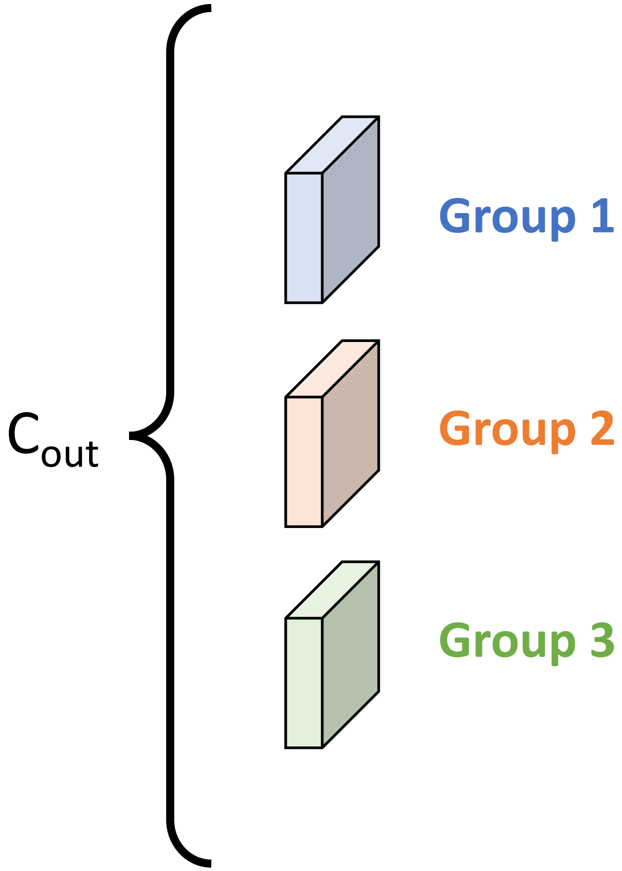
Special Case: Depthwise Convolution

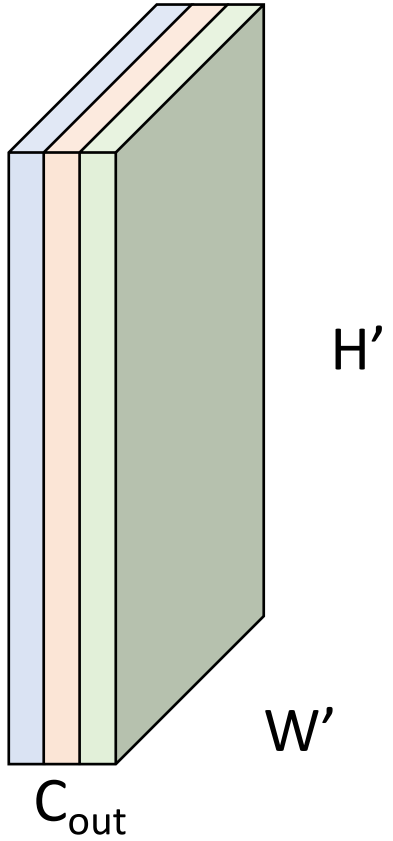
Number of groups equals number of input channels

Common to also set C_{out} = G

Output only mixes ***spatial*** information from input; **channel** information not mixed







Input: C_{in} x H x W

Weights: C_{out} x 1 x K x K

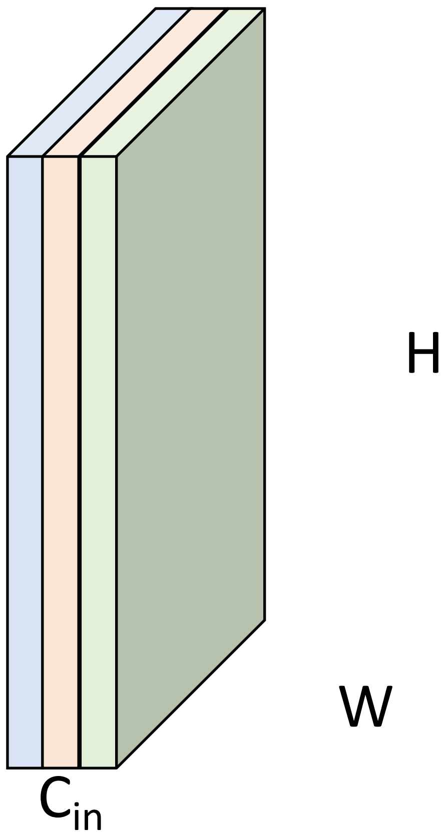
Output: C_{out} x H' x W'

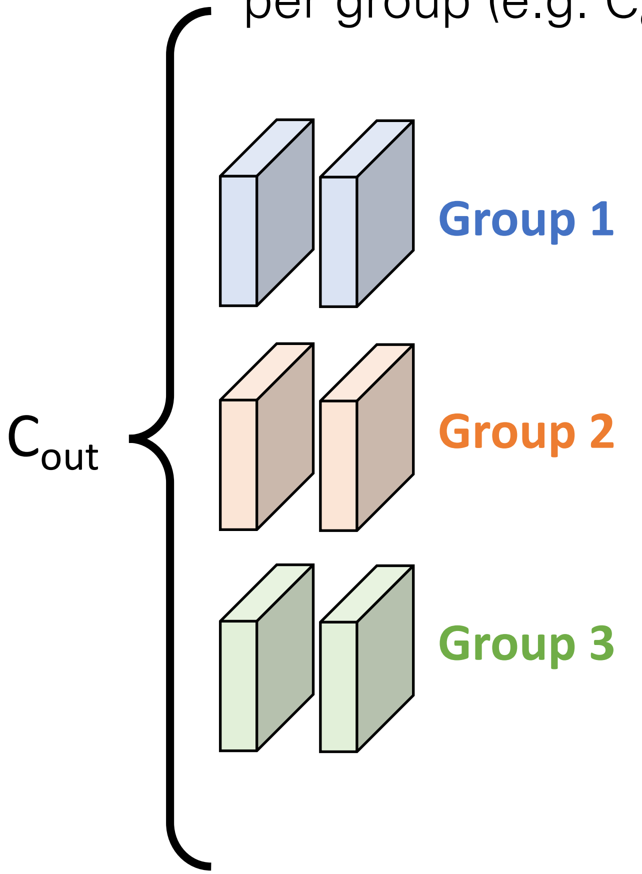
Special Case: Depthwise Convolution

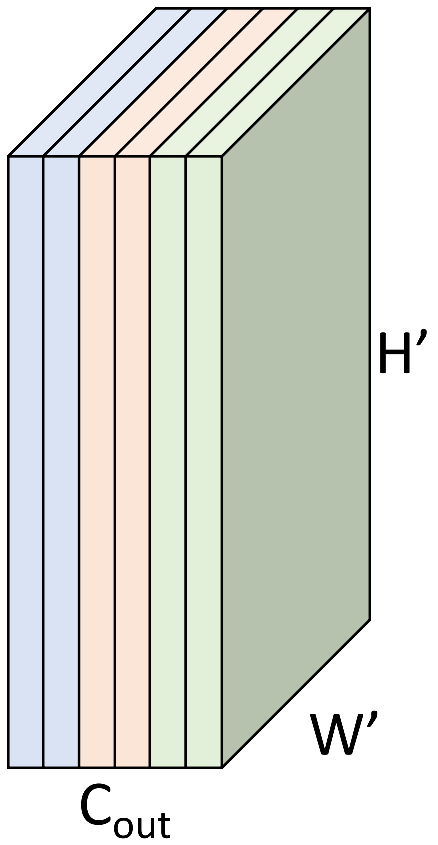
Number of groups equals number of input channels

Can still have multiple filters per group (e.g. $C_{out} = 2C_{in}$)

Output only mixes ***spatial*** information from input; **channel** information not mixed







Input: C_{in} x H x W

Weights: C_{out} x 1 x K x K

Output: C_{out} x H' x W'

Grouped Convolution vs Standard Convolution

Grouped Convolution (G groups):

G parallel conv layers; each "sees" C_{in}/G input channels and produces C_{out}/G output channels

Input: $C_{in} \times H \times W$

Split to $G \times [(C_{in} / G) \times H \times W]$

Weight: $G \times (C_{out} / G) \times (C_{in} / G) \times K \times K$

G parallel convolutions

Output: $G \times [(C_{out} / G) \times H' \times W']$

Concat to $C_{out} \times H' \times W'$

FLOPs: $C_{out}C_{in}K^2HW/G$

Standard Convolution (groups=1)

Input: $C_{in} \times H \times W$

Weight: $C_{out} \times C_{in} \times K \times K$

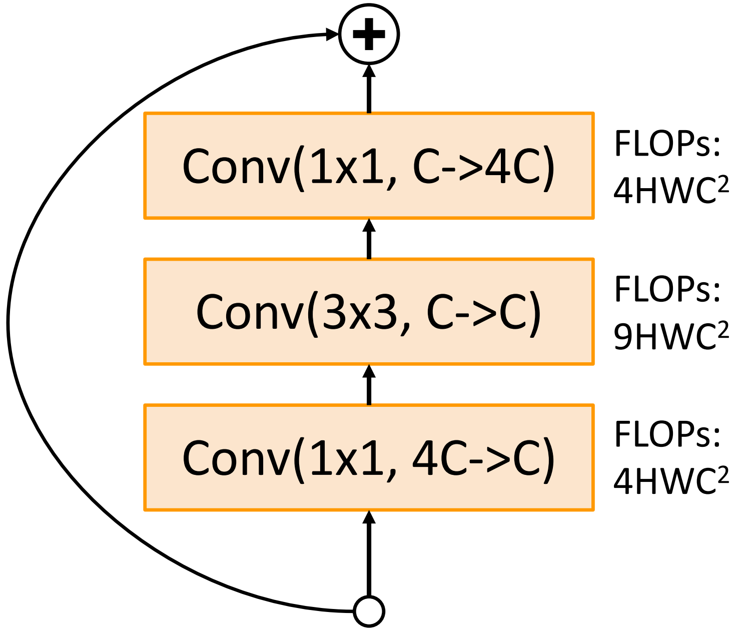
Output: $C_{out} \times H' \times W'$

FLOPs: $C_{out}C_{in}K^2HW$

All convolutional kernels touch all C_{in} channels of the input

Using G groups reduces FLOPs by a factor of G!

Improving ResNets

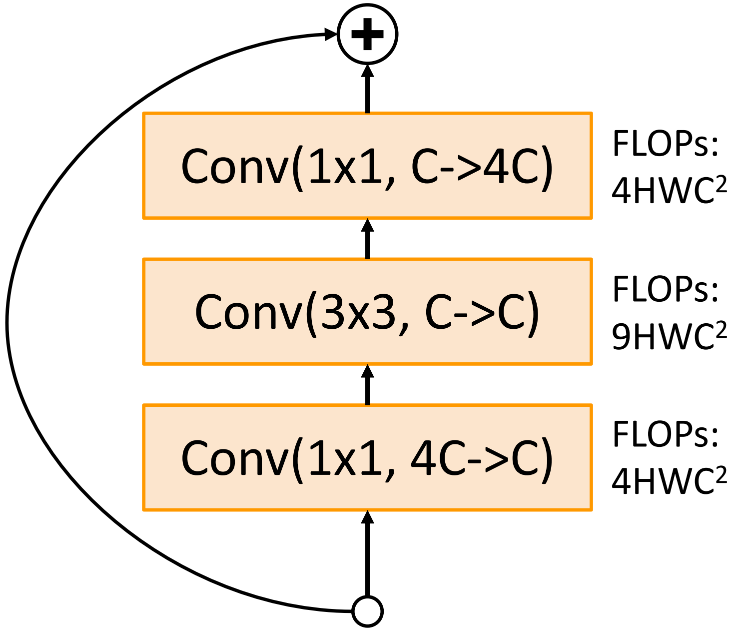


"Bottleneck" Residual block

Total FLOPs: $17HWC^2$

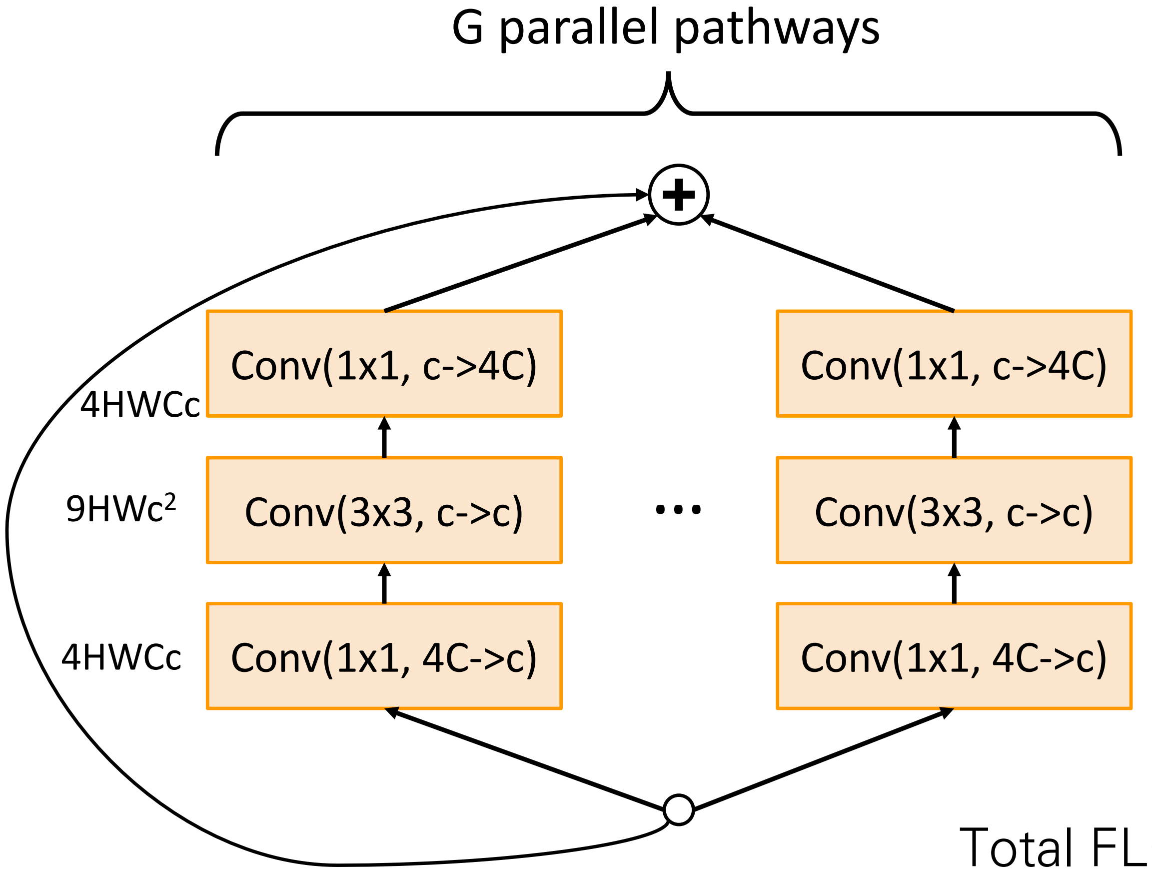
Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

Improving ResNets: ResNeXt



"Bottleneck" Residual block

Total FLOPs: $17HWC^2$



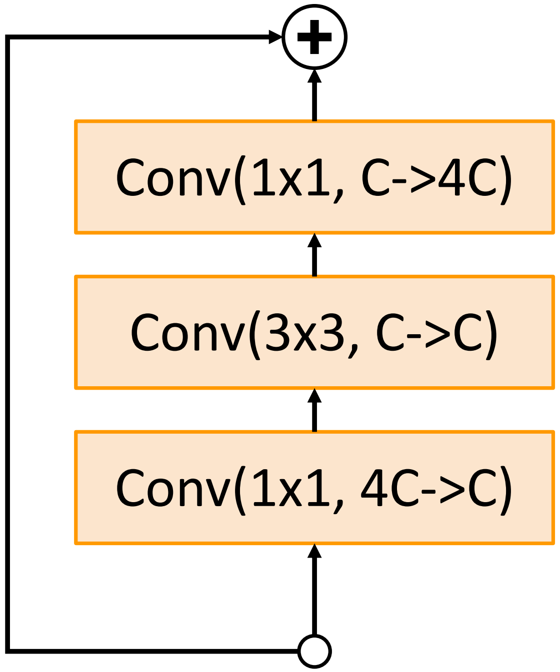
Total FLOPs: $(8Cc + 9c^2)*HWG$

Same FLOPs when $9Gc^2 + 8GCc - 17C^2 = 0$

Example: C=64, G=4, c=24; C=64, G=32, c=4

Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

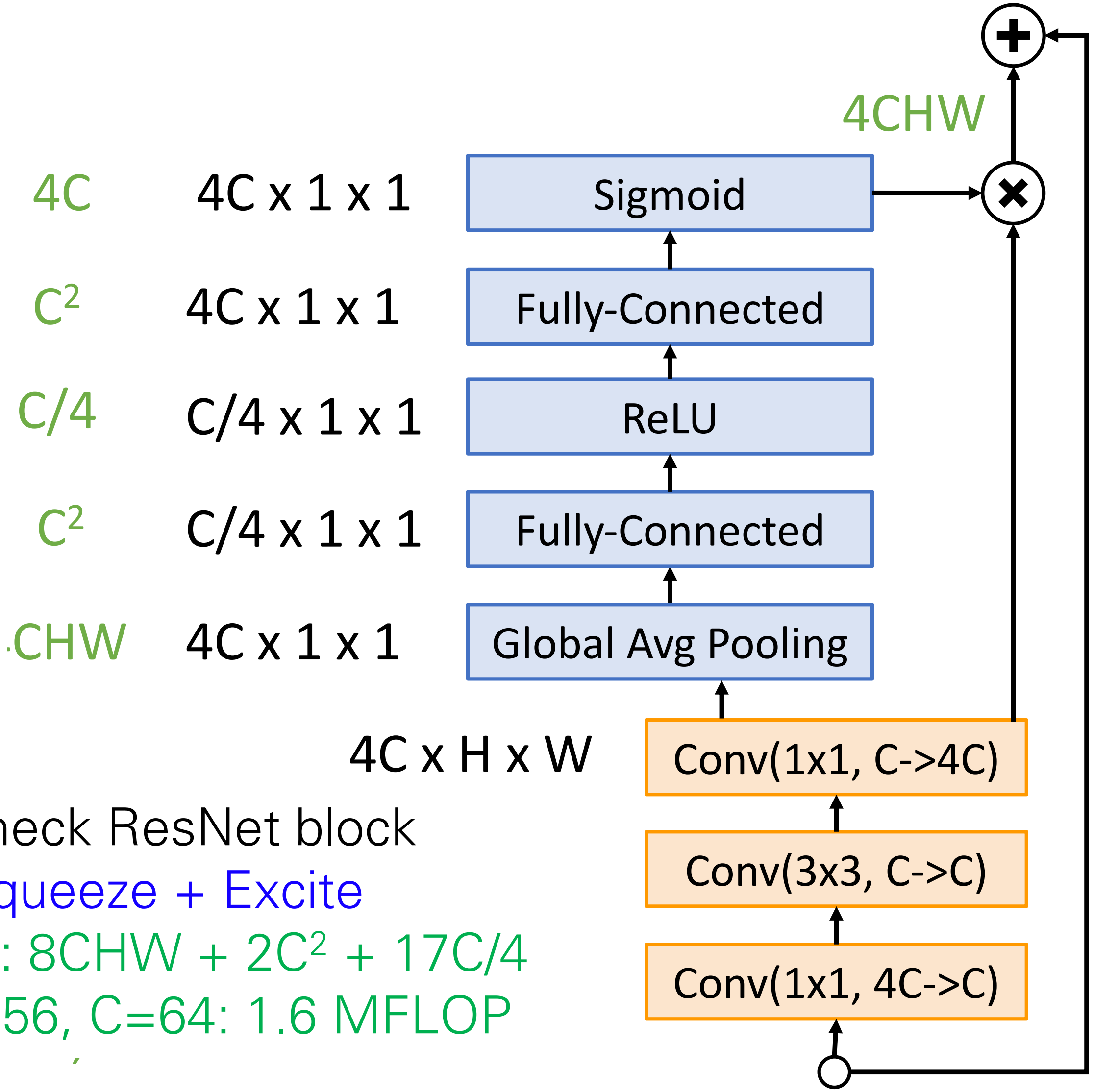
Squeeze-and-Excitation Networks (SENet)



Bottleneck ResNet block
FLOPs: $17HWC^2$
H=W=56, C=64: 218 MFLOP

Adds **global context** to each ResNet block

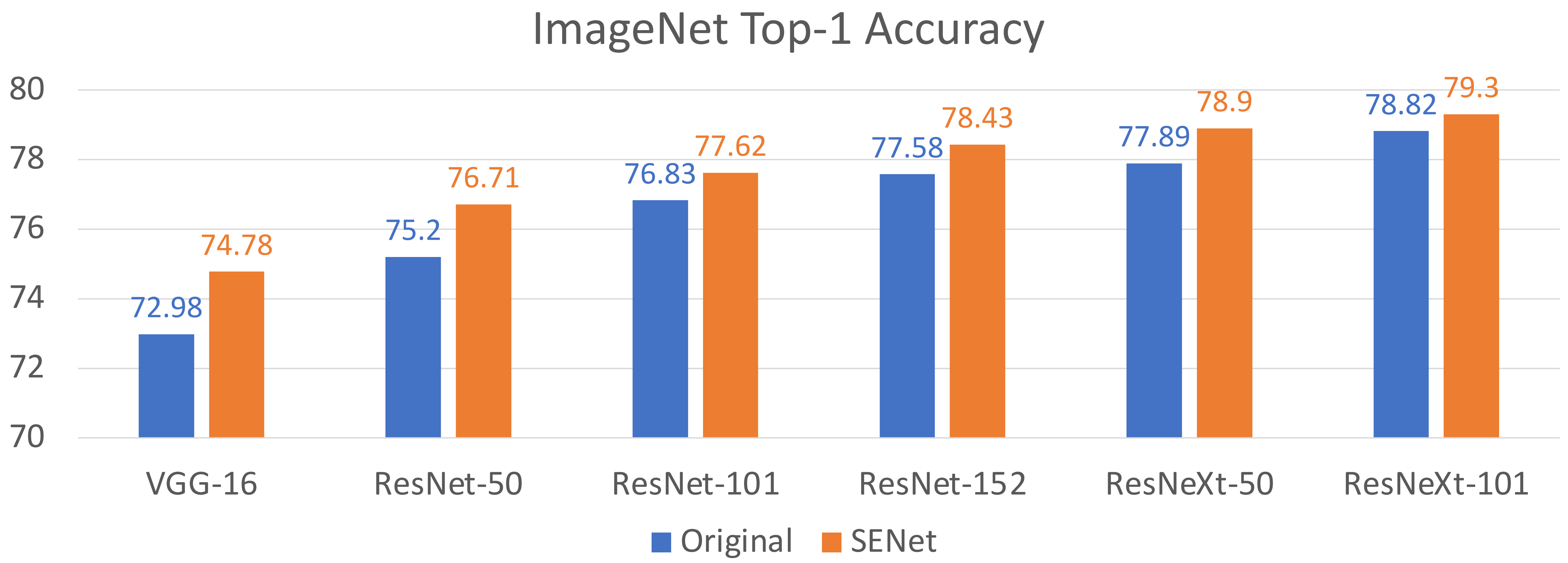
Increases overall FLOPs by < 1%!



Bottleneck ResNet block
with Squeeze + Excite
FLOPs: $8CHW + 2C^2 + 17C/4$
H=W=56, C=64: 1.6 MFLOP

Hu et al, "Squeeze-and-Excitation networks", CVPR 2018

Squeeze-and-Excitation Networks (SENet)



Add SE to any architecture, enjoy 1-2% boost in accuracy

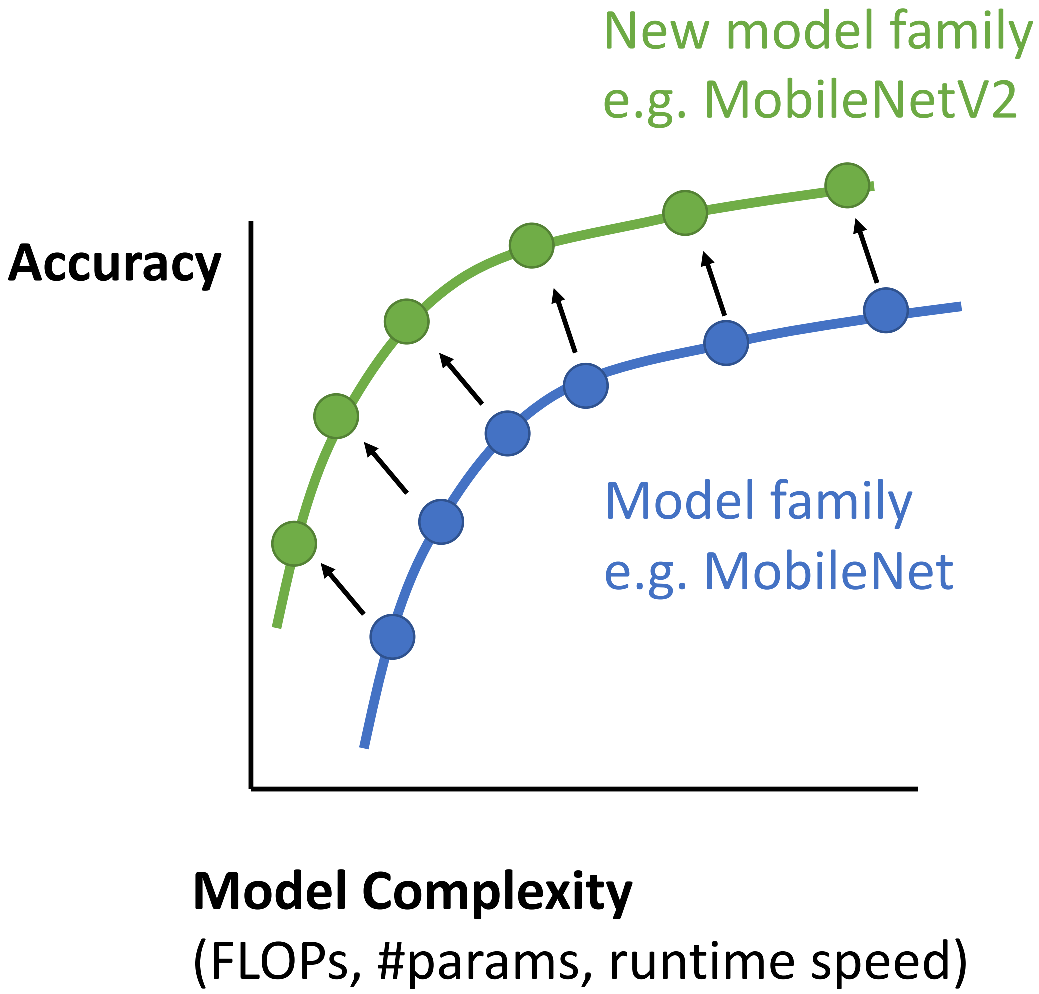
Hu et al, "Squeeze-and-Excitation networks", CVPR 2018

Recall: Convolution Layer

Instead of pushing for the largest network with biggest accuracy, consider tiny networks and accuracy / complexity tradeoff

Compare **families of models**:

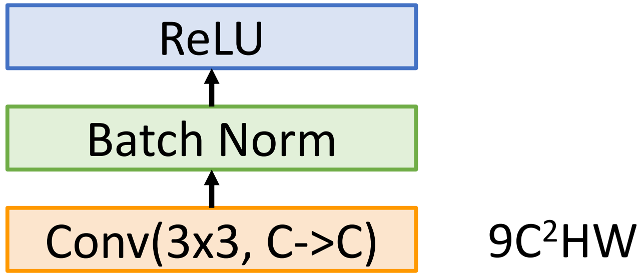
One family is better than another if it moves the whole curve up and to the left



MobileNets: Tiny Networks (For Mobile Devices)

Standard Convolution Block

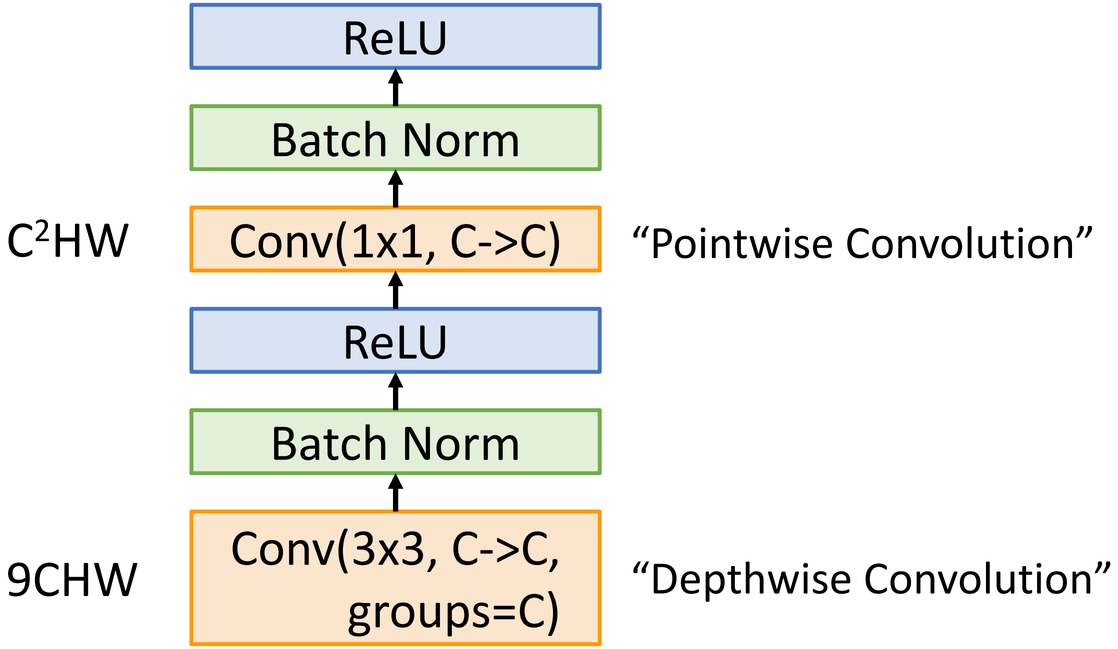
Total cost: $9C^2HW$



Speedup $= 9C2/(9C+C^2)$
$= 9C/(9+C)$
$=> 9$ (as C->inf)

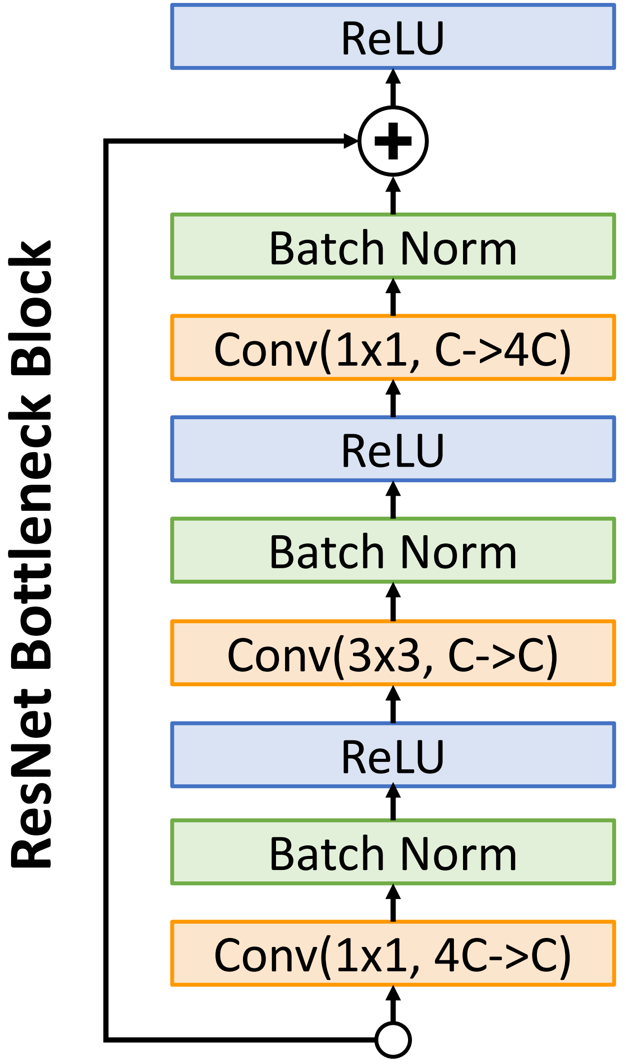
Depthwise Separable Convolution

Total cost: $(9C + C^2)HW$



Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017
Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions", CVPR 2017

MobileNetV2: Inverted Bottleneck, Linear Residual



Nonlinearity outside residual

Total FLOP: $17HWC^2$

1x1 conv **expands** channels output ($4HWC^2$ FLOP)

3x3 conv uses **fewer** channels than input ($9HWC^2$ FLOP)

1x1 conv **reduces** channels before 3x3 conv ($4HWC^2$ FLOP)

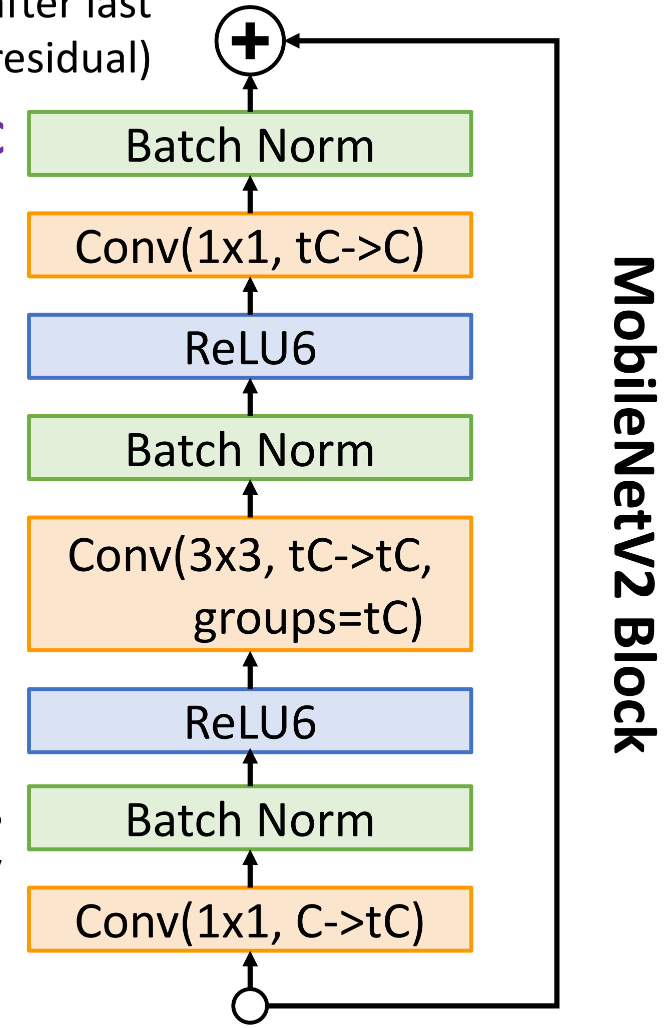
No nonlinearity after last conv! (linear residual)

Total FLOP: $2tHWC^2 + 9tHWC$

1x1 conv **reduces** channels before output ($tHWC^2$ FLOP)

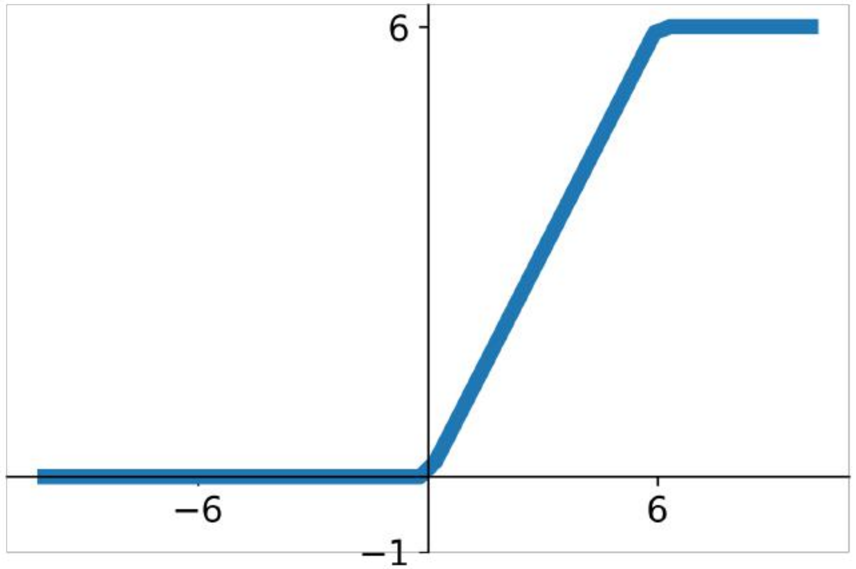
3x3 Depthwise conv with **more** channels than input ($9tHWC$ FLOP)

1x1 conv **increases** channels before 3x3 conv (inverted bottleneck) ($tHWC^2$ FLOP)



Sandler et al, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", CVPR 2018

MobileNetV2: Inverted Bottleneck, Linear Residual



$$ReLU6(x) = \begin{cases} 0 & if\ x \le 0 \\ x & if\ 0 < x < 6 \\ 6 & if\ x \ge 6 \end{cases}$$

Keeps activations in reasonable range when running inference in low precision

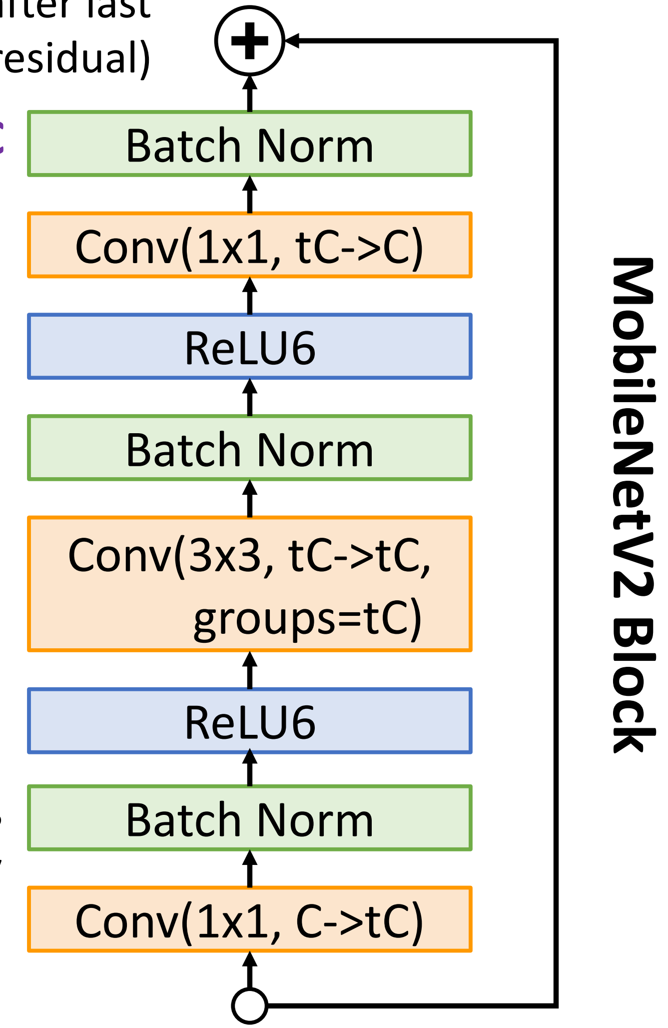
No nonlinearity after last conv! (linear residual)

Total FLOP: $2tHWC^2 + 9tHWC$

1x1 conv **reduces** channels before output ($tHWC^2$ FLOP)

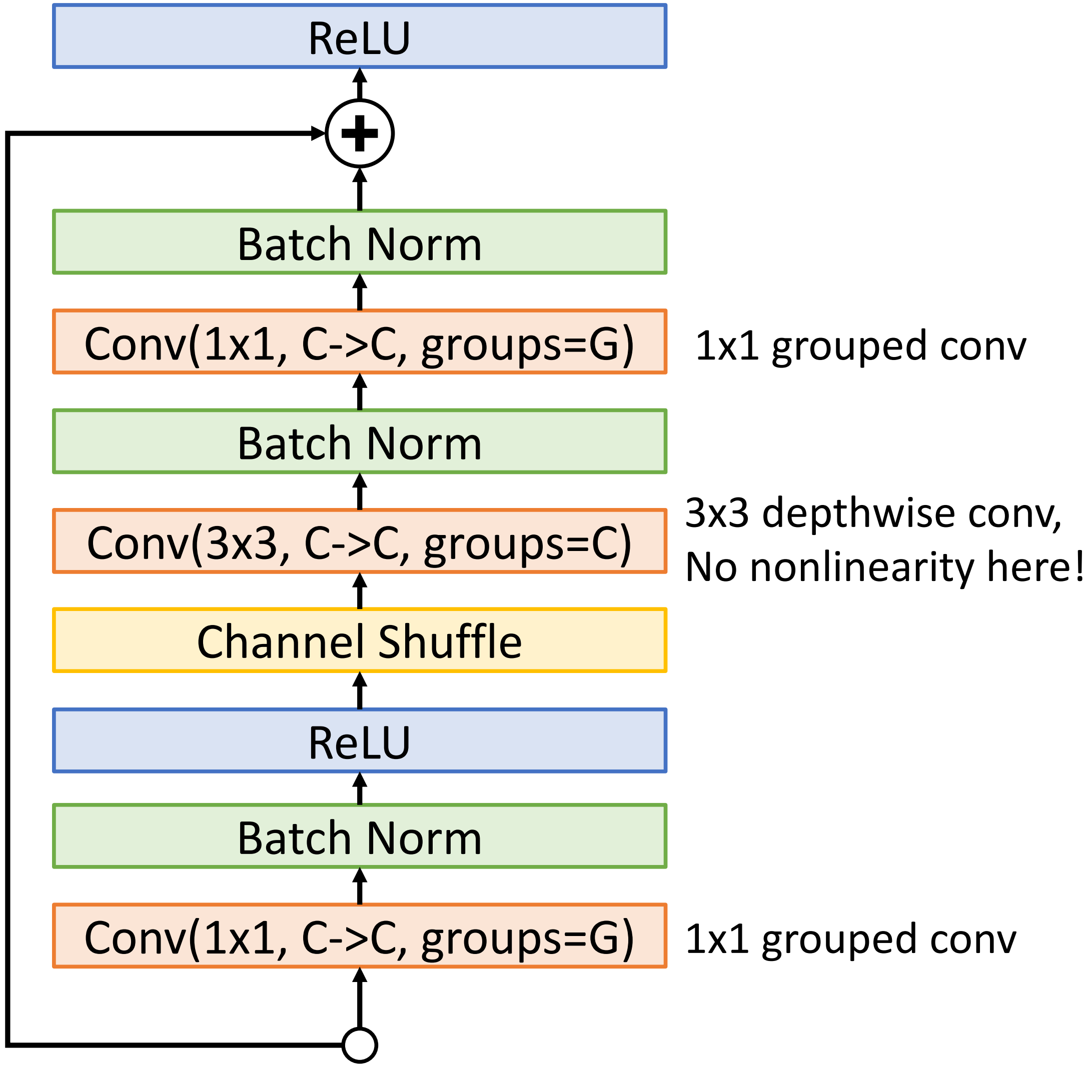
3x3 Depthwise conv with **more** channels than input (9tHWC FLOP)

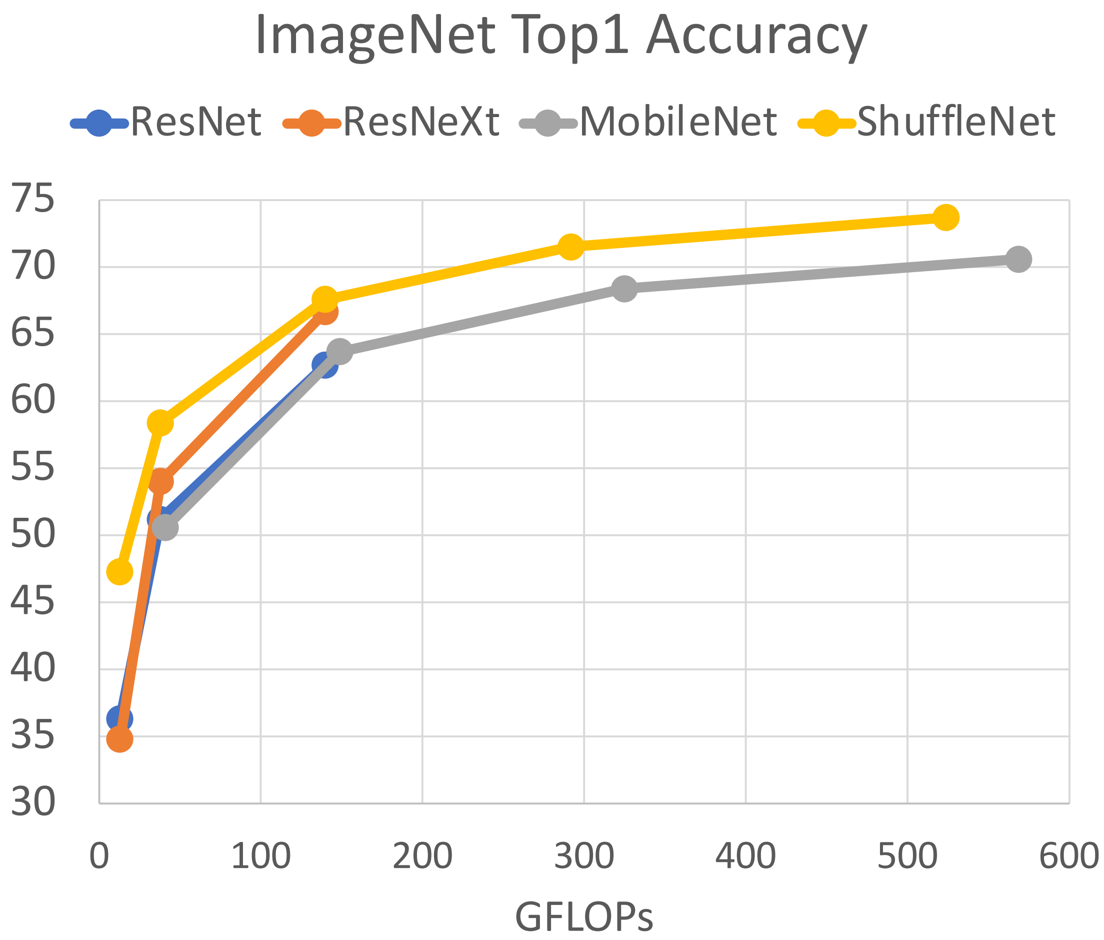
1x1 conv **increases** channels before 3x3 conv (inverted bottleneck) ($tHWC^2$ FLOP)



Sandler et al, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", CVPR 2018

ShuffleNet

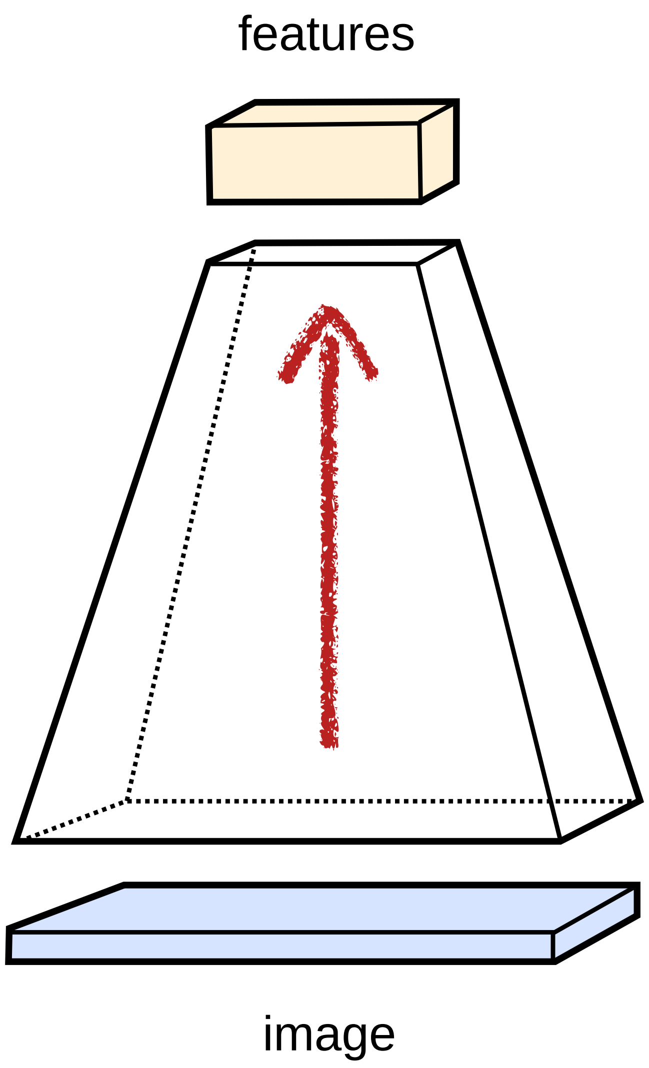




Zhang et al, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices", CVPR 2018

Design Guidelines

Design Guidelines



Guideline 1: Avoid tight bottlenecks

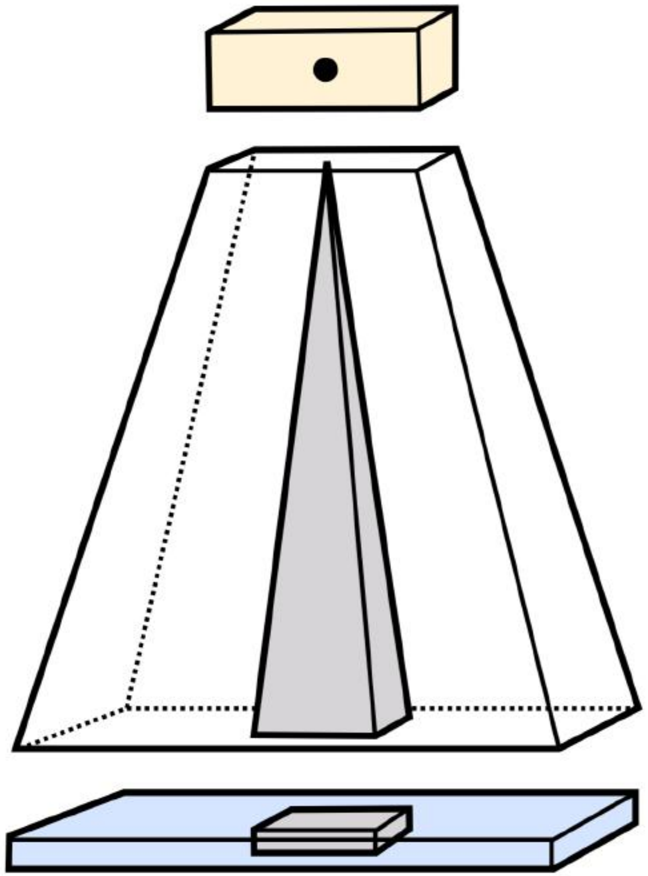
- **From bottom to top**
  - The spatial resolution H×W decreases
  - The number of channels C increases

- **Guideline**
  - Avoid tight information bottleneck
  - Decrease the data volume H × W × C slowly

K. Simonyan and A. Zisserman. **Very deep convolutional networks for large-scale image recognition**. In ICLR 2015.

C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens. **Rethinking the inception architecture for computer vision**. In CVPR 2016.

Receptive Field

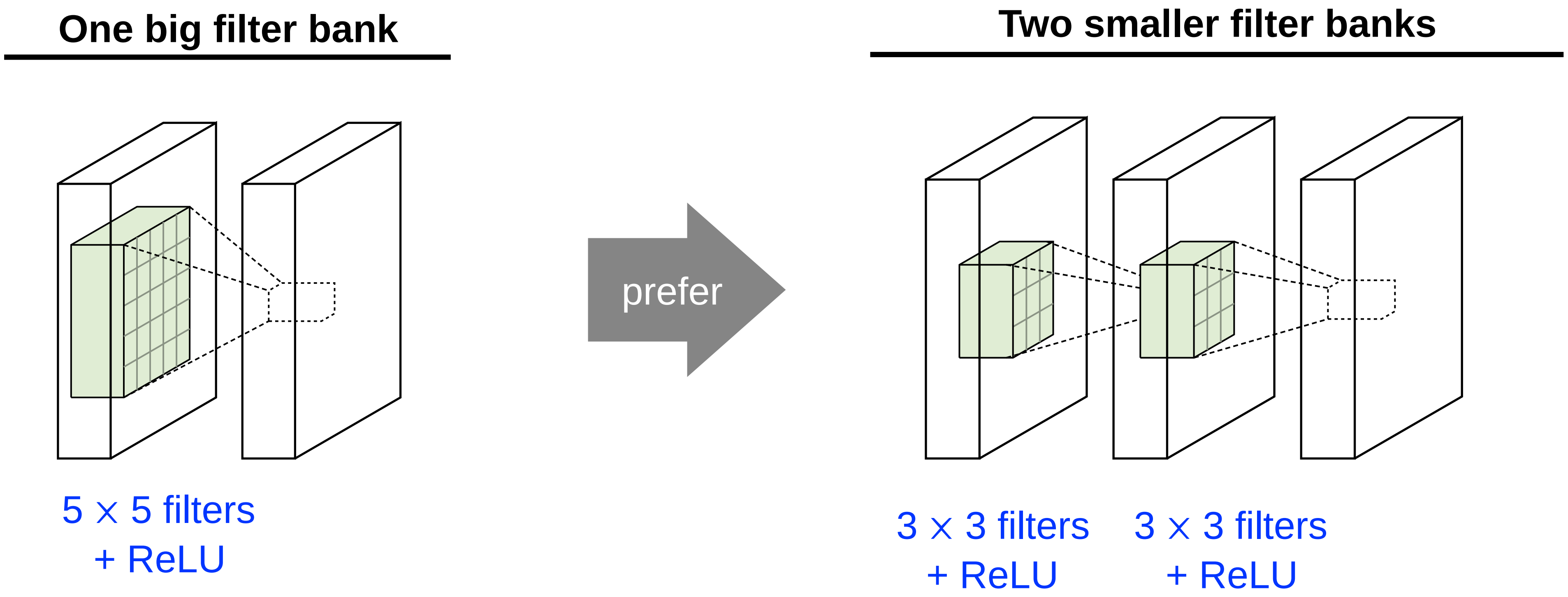


Must be large enough

- **Receptive field of a neuron**
  - The image region influencing a neuron
  - Anything happening outside is invisible to the neuron
- **Importance**
  - Large image structures cannot be detected by neurons with small receptive fields
- **Enlarging the receptive field**
  - Large filters
  - Chains of small filters

Design Guidelines

Guideline 2: Prefer small filter chains



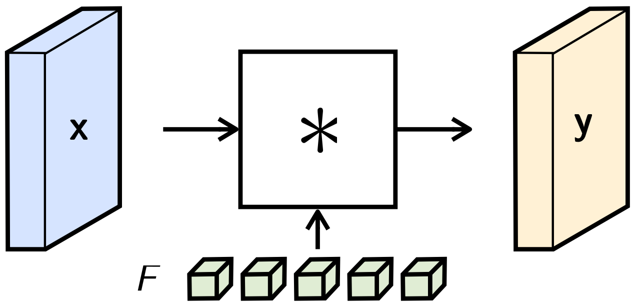
- **Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers
- **Reason:** Far fewer feature channels (quadratic speed/space gain)
- **Moral:** Optimize your architecture

Design Guidelines

Guideline 3:

Keep the number of channels at bay

$H \times W \times C$



$H_f \times W_f \times C \times K$

C = num. input channels

K = num. output channels

Num. of operations

$$\frac{H \times H_f}{\text{stride}} \times \frac{W \times W_f}{\text{stride}} \times C \times K$$

Num. of parameters

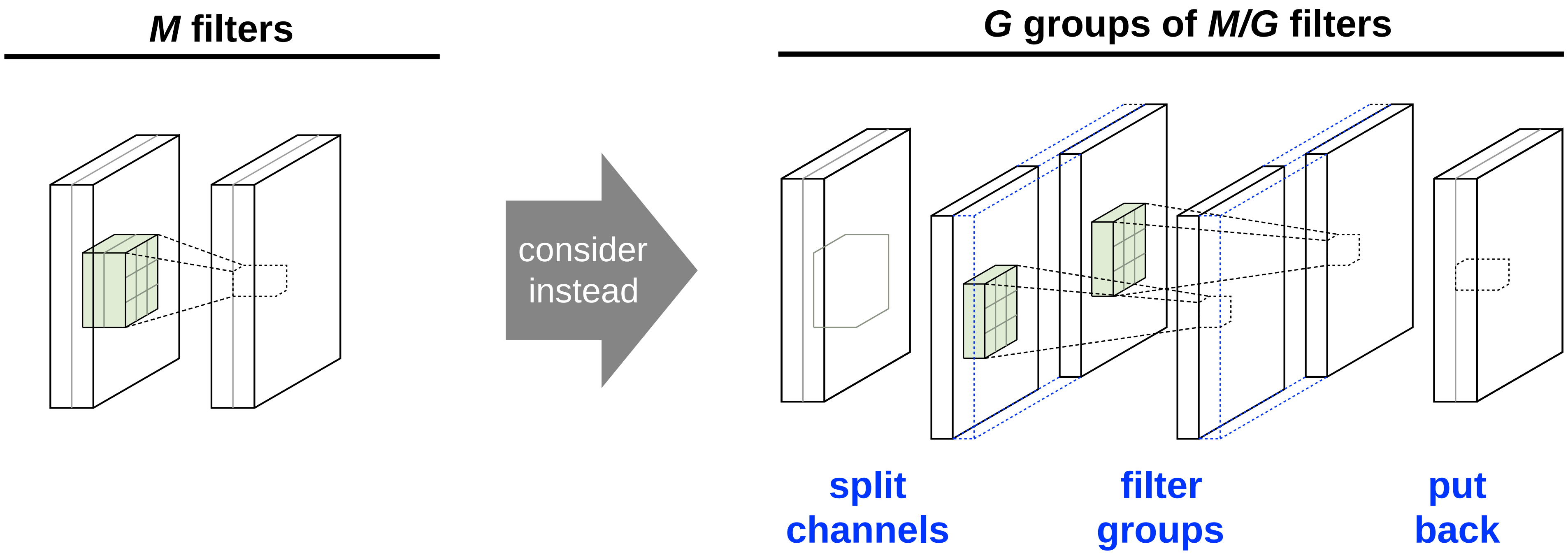
$$H_f \times W_f \times C \times K$$

complexity $\propto C \times K$

Design Guidelines

Guideline 4:

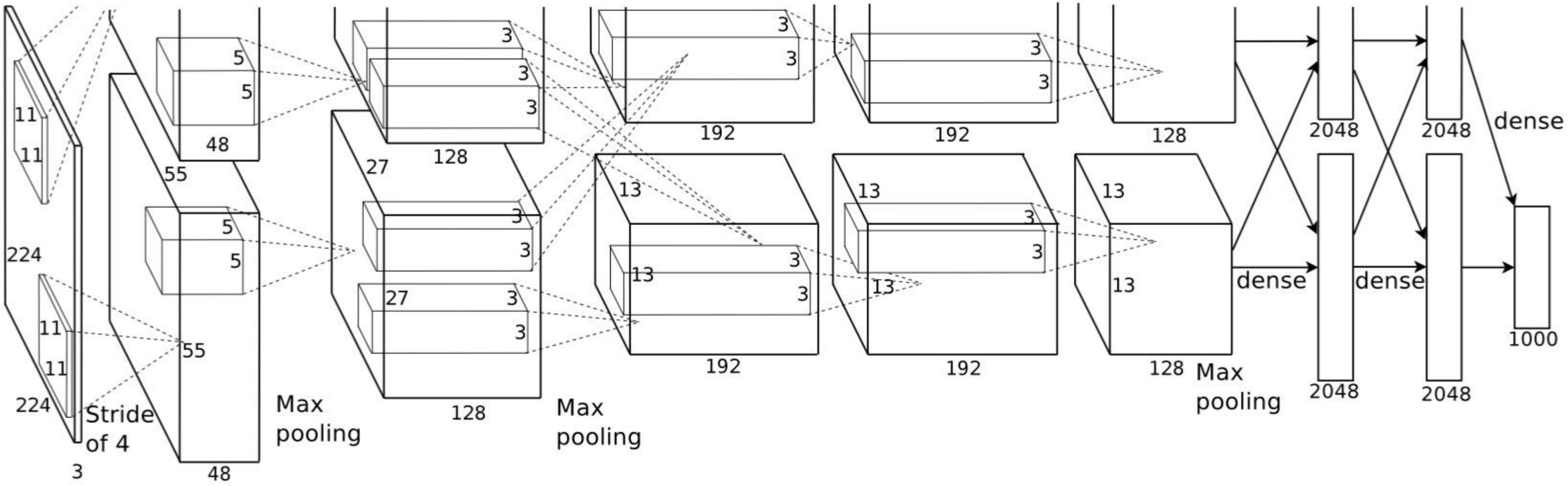
Less computations with filter groups



Did we see this before?

complexity $\propto (C \times K) / G$

AlexNet



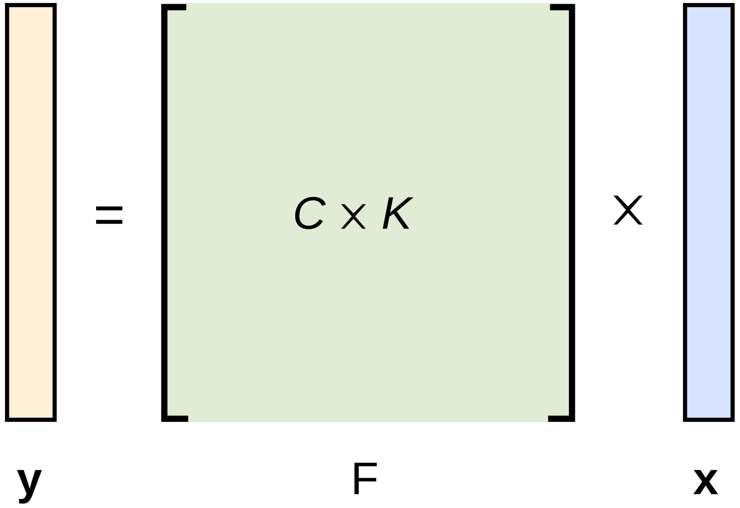
A. Krizhevsky, I. Sutskever, and G. E. Hinton. **Imagenet classification with deep convolutional neural networks**. In NIPS

Design Guidelines

Guideline 4:

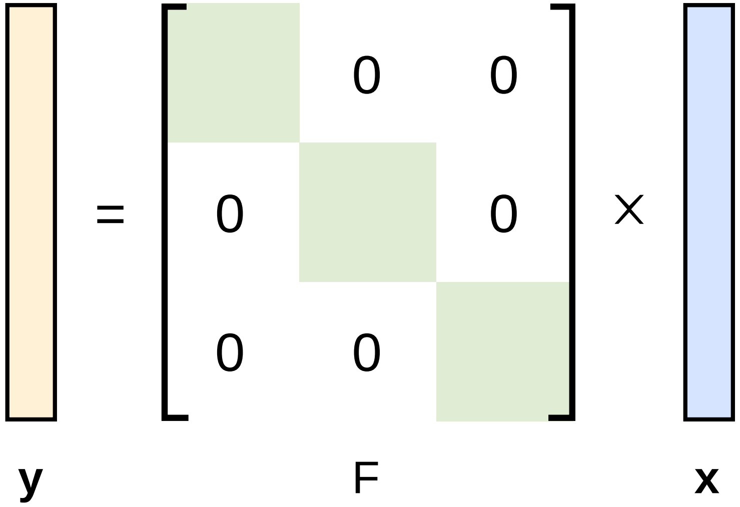
Less computations with filter groups

Full filters



complexity: $C \times K$

Group-sparse filters



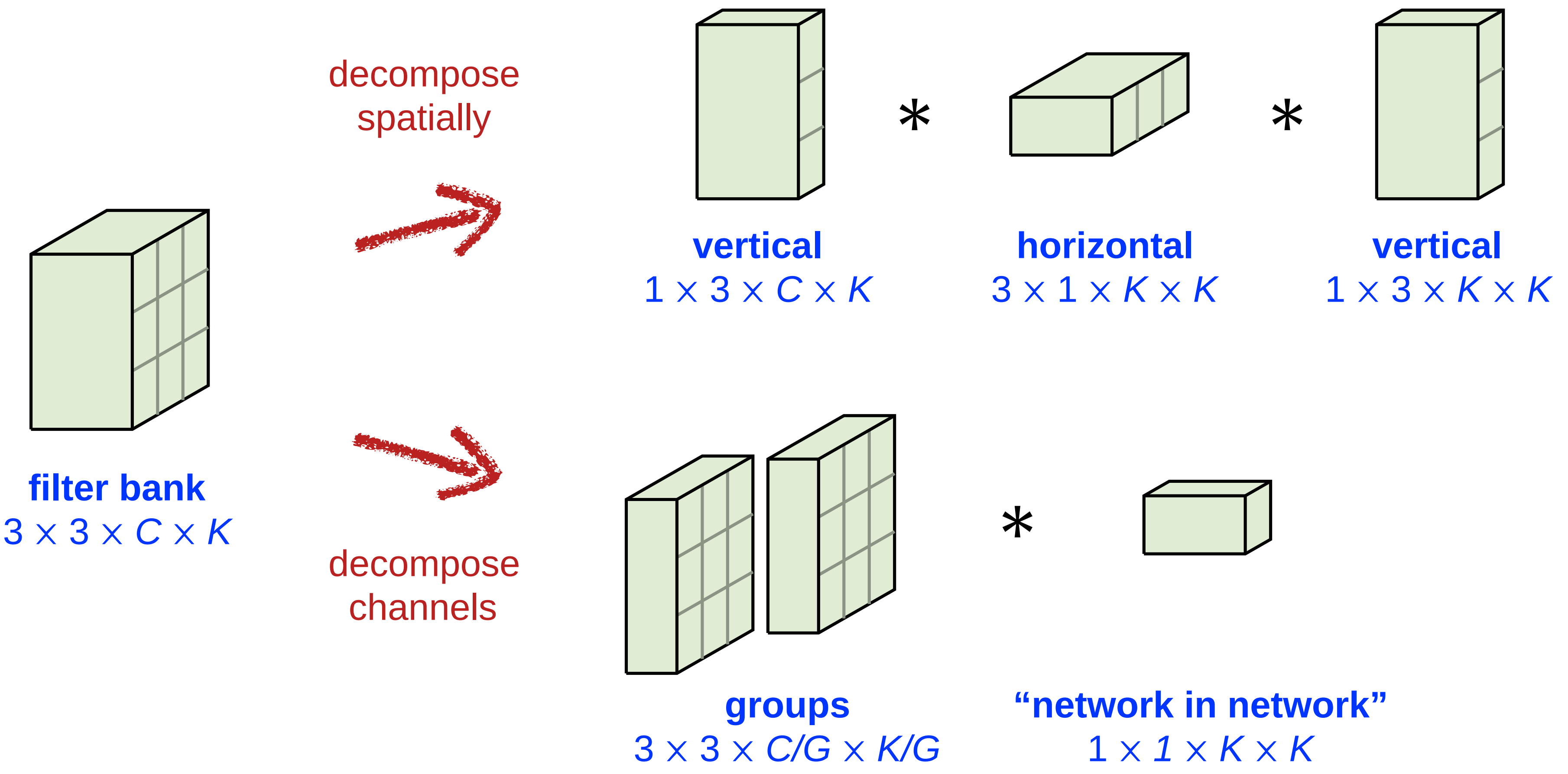
complexity: $C \times K / G$

Groups = filters, seen as a matrix, have a "block" structure

Design Guidelines

Guideline 5:

Low-rank decompositions

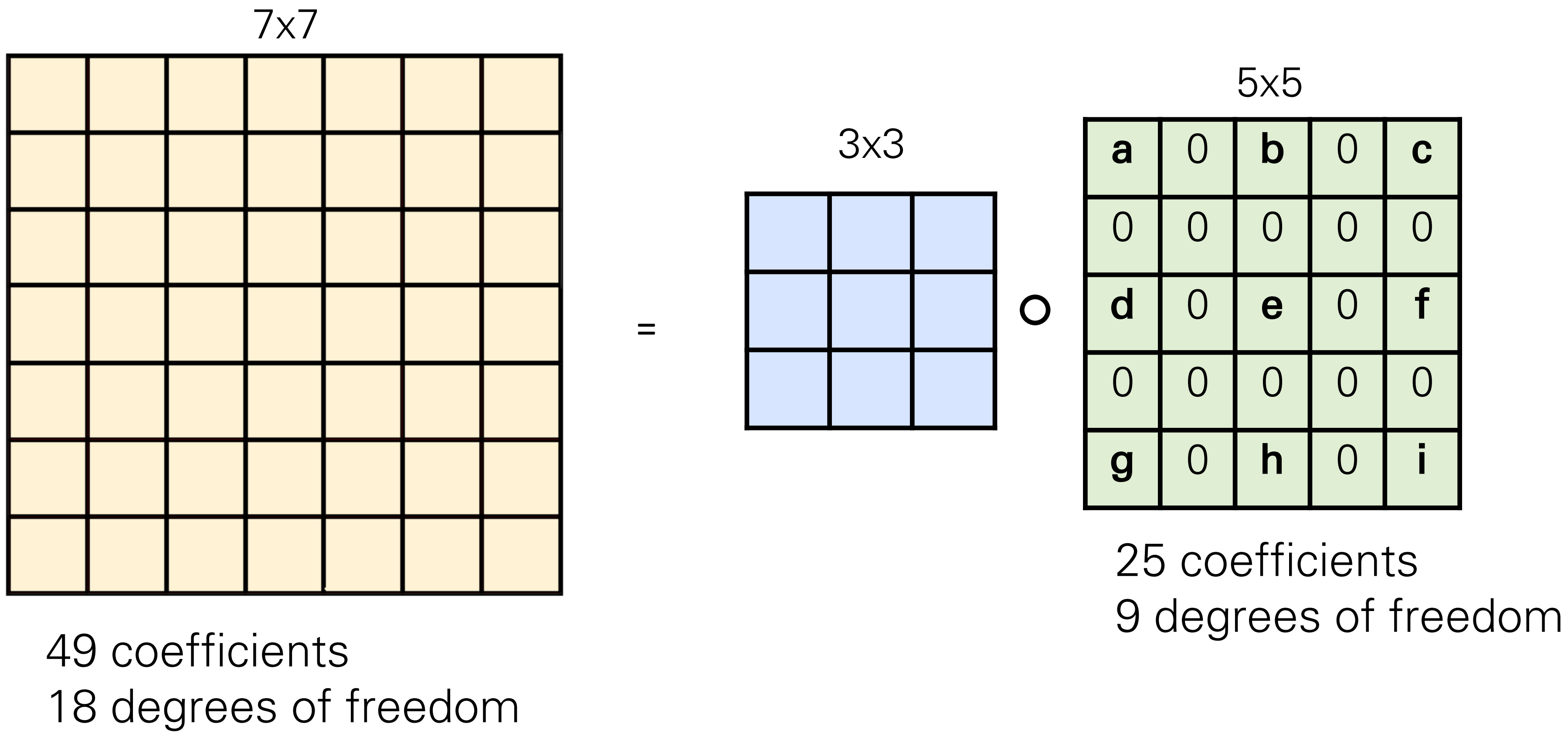


Make sure to mix the information

Design Guidelines

Guideline 6:

Dilated Convolutions



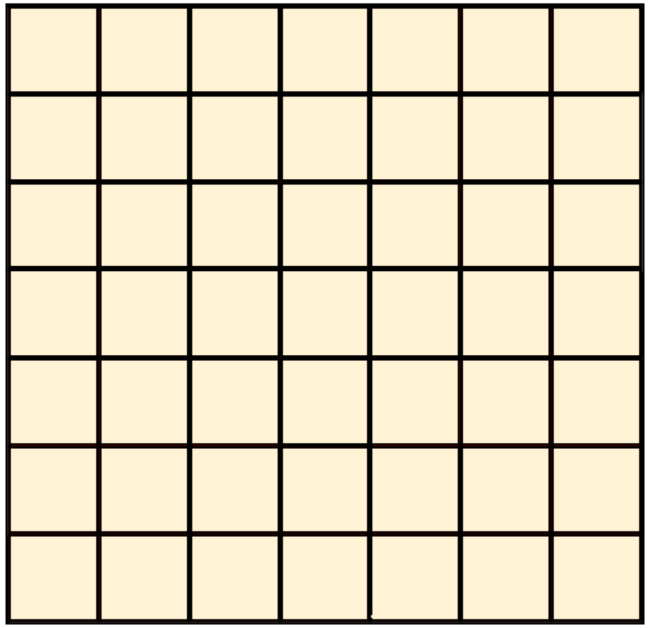
Exponential expansion of the receptive field without loss of resolution

Design Guidelines

Guideline 6:

Dilated Convolutions

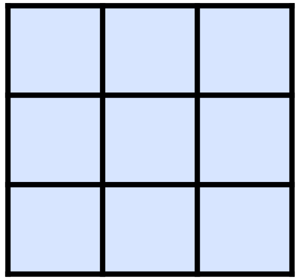
7x7



49 coefficients
18 degrees of freedom

=

3x3



∘

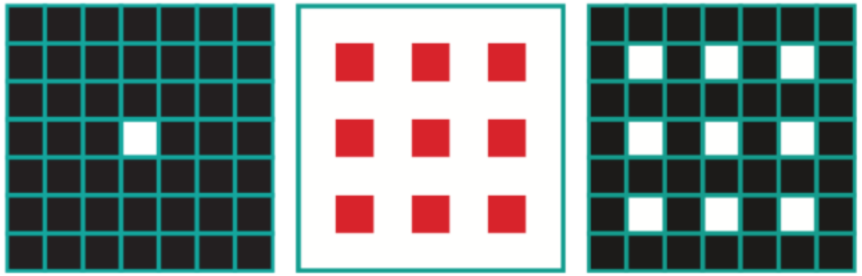
5x5

a	0	**b**	0	**c**
0	0	0	0	0
d	0	**e**	0	**f**
0	0	0	0	0
g	0	**h**	0	**i**

25 coefficients
9 degrees of freedom

Exponential expansion of the receptive field without loss of resolution

What is lost?



(a) Input (b) Dilation 2 (c) Output

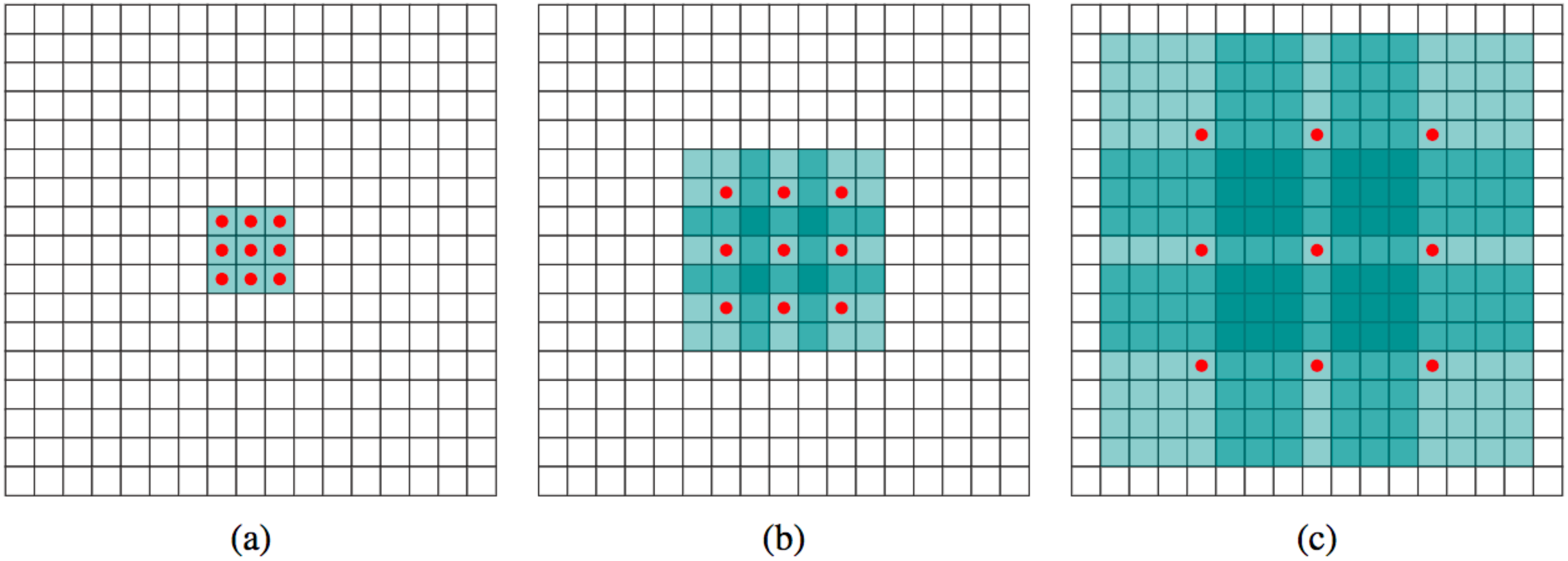


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

[https://arxiv.org/pdf/1511.07122.pdf]

CNN Architectures Summary

- Early work (AlexNet->VGG->ResNet): **bigger networks work better**
- New focus on **efficiency**: Improve accuracy, control for network complexity
- Grouped, Depthwise, Dilated convolution appear in many modern architectures
- **Squeeze-and-Excite** adds accuracy boost to just about any architecture while only adding a tiny amount of FLOPs and runtime
- Tiny networks for **mobile devices** (MobileNet, ShuffleNet)
- **Neural Architecture Search (NAS)** promised to automate architecture design
- More recent work has moved towards **careful improvements to ResNet-like architectures**
- ResNet and ResNeXt are still surprisingly strong and popular architectures!

Transfer Learning with Convolutional Neural Networks

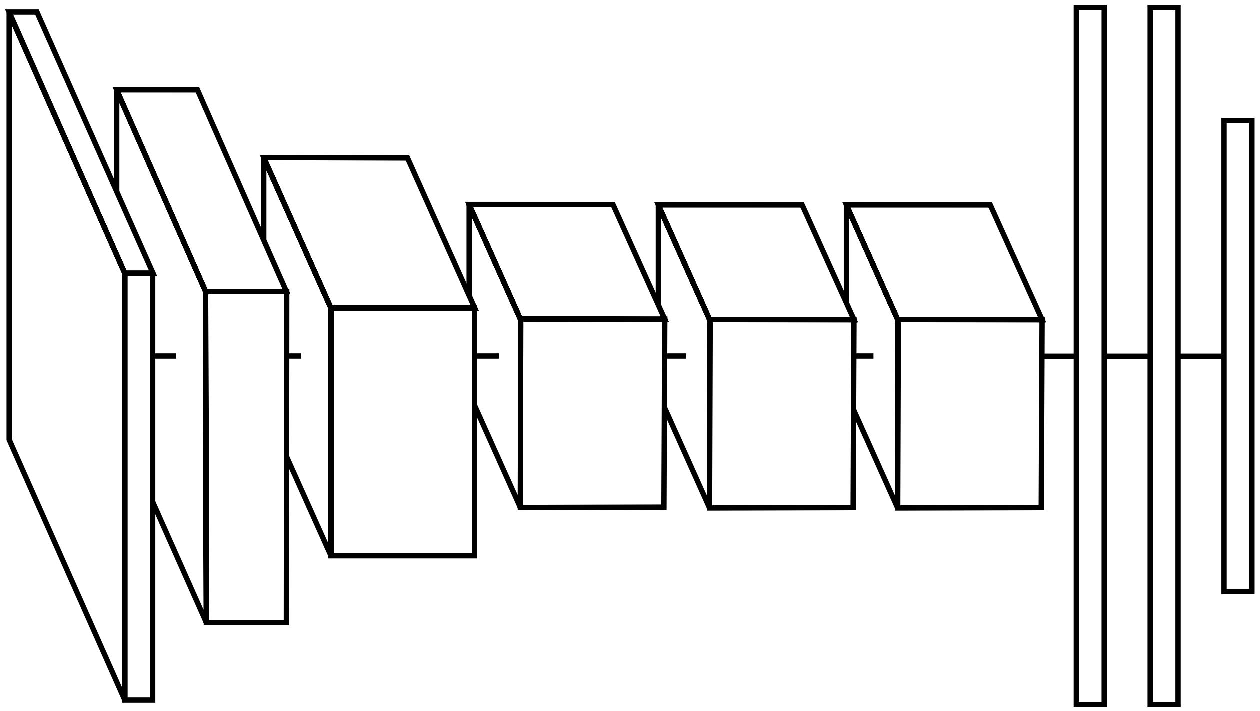
Beyond CNNs

- Do features extracted from the CNN generalize other tasks and datasets?
  - Donahue et al. (2013), Chatfield et al. (2014), Razavian et al. (2014), Yosinski et al. (2014), etc.

- CNN activations as deep features
- Finetuning CNNs

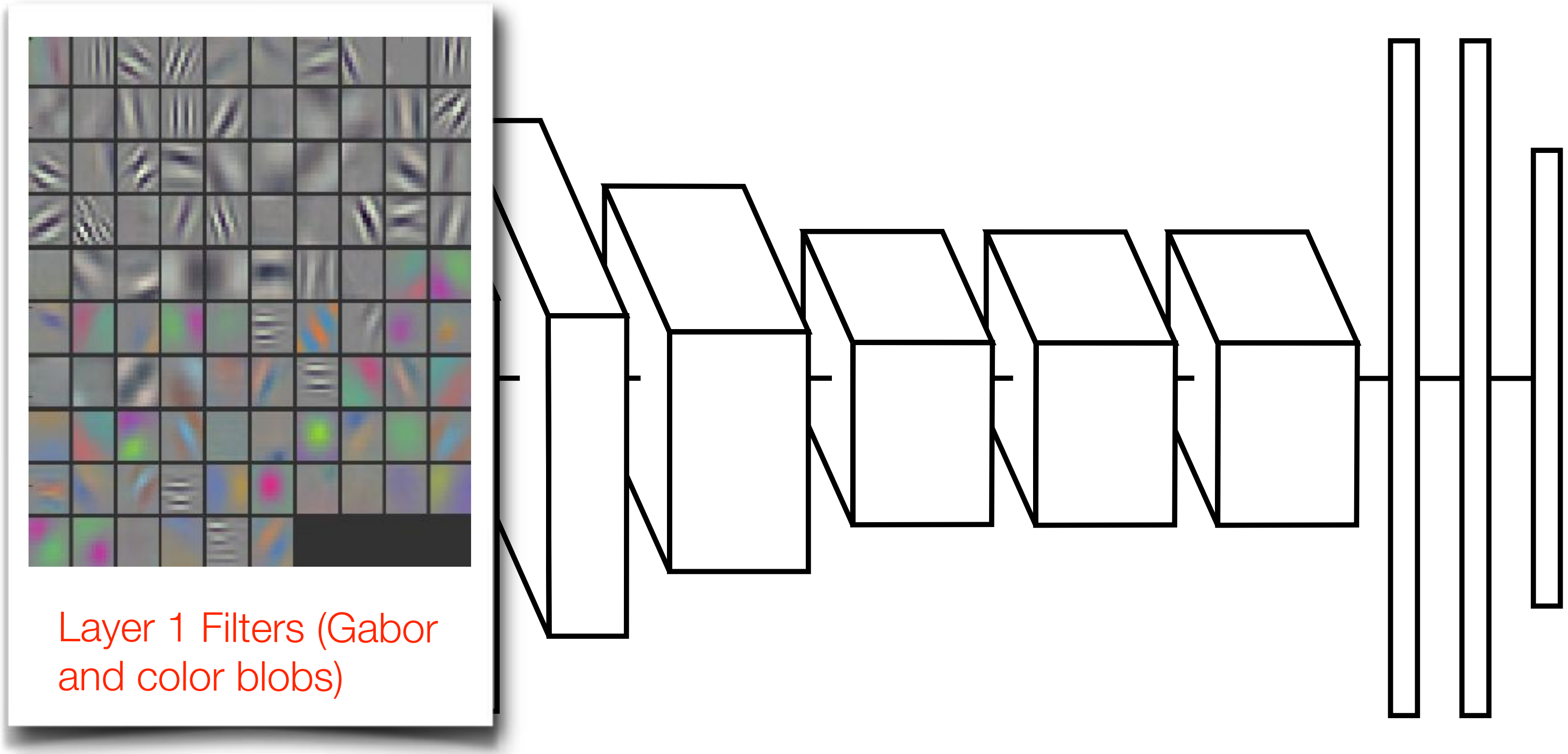
CNN activations as deep features

- CNNs discover effective representations. Why not to use them?



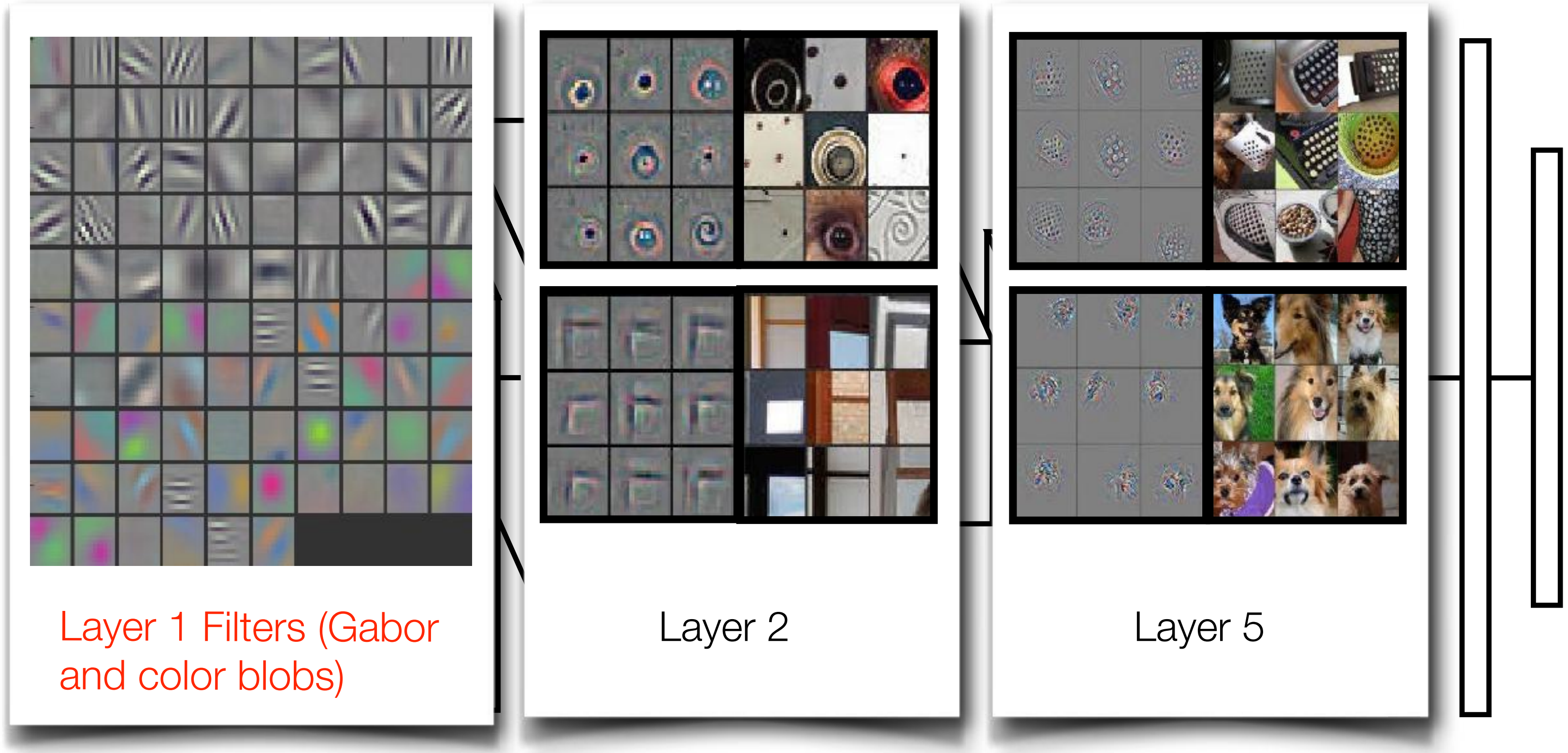
CNN activations as deep features

- CNNs discover effective representations. Why not to use them?



CNN activations as deep features

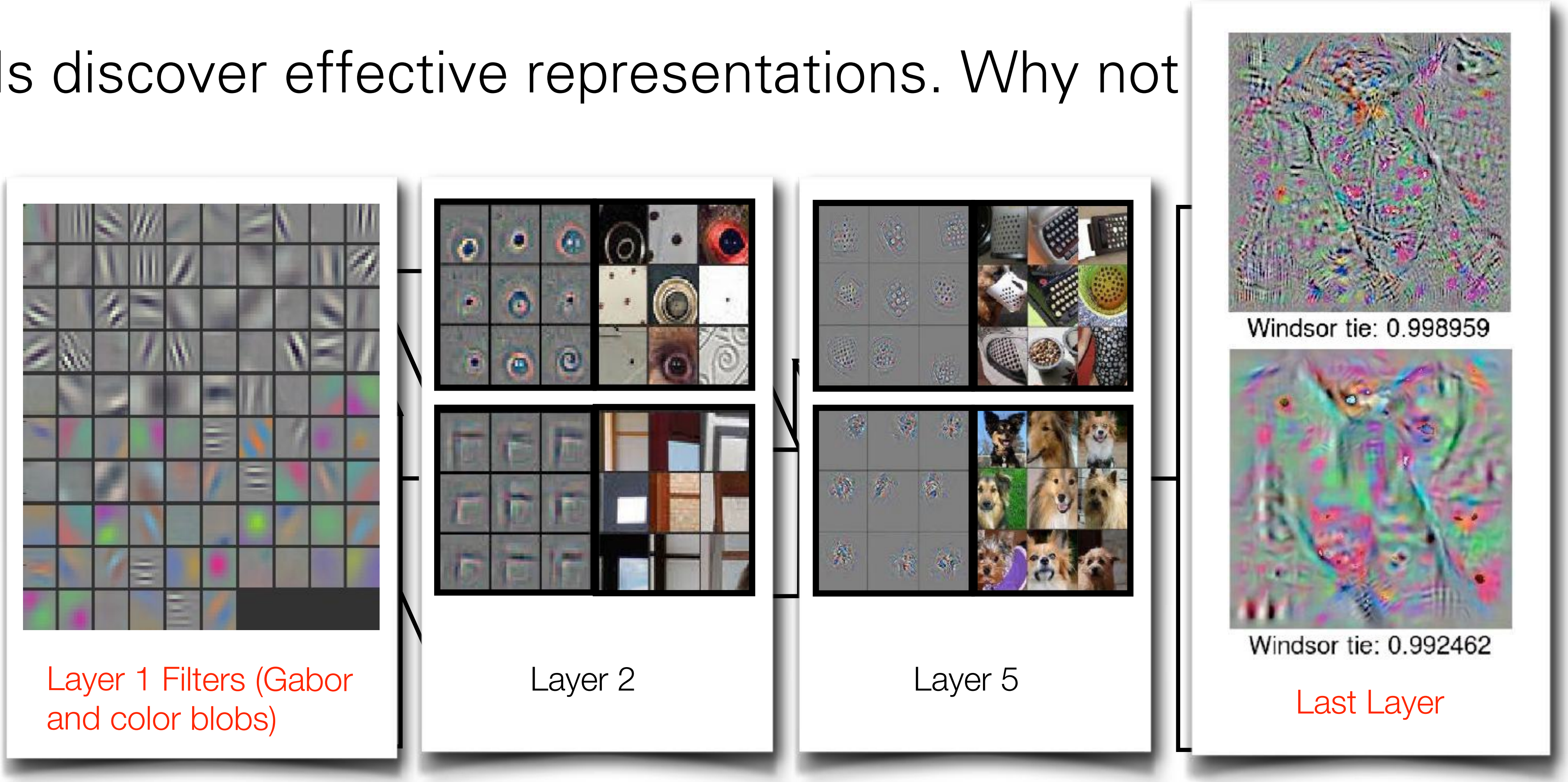
- CNNs discover effective representations. Why not to use them?



Zeiler et al., 2014

CNN activations as deep features

- CNNs discover effective representations. Why not



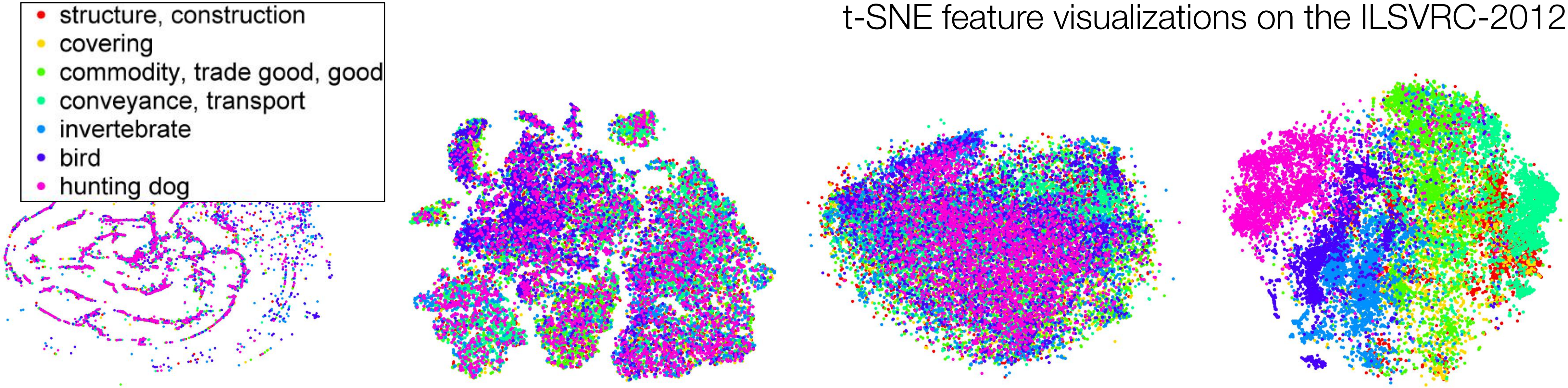
Zeiler et al., 2014

Nguyen et al., 2014

CNNs as deep features

• CNNs discover effective representations. Why not to use them?

t-SNE feature visualizations on the ILSVRC-2012

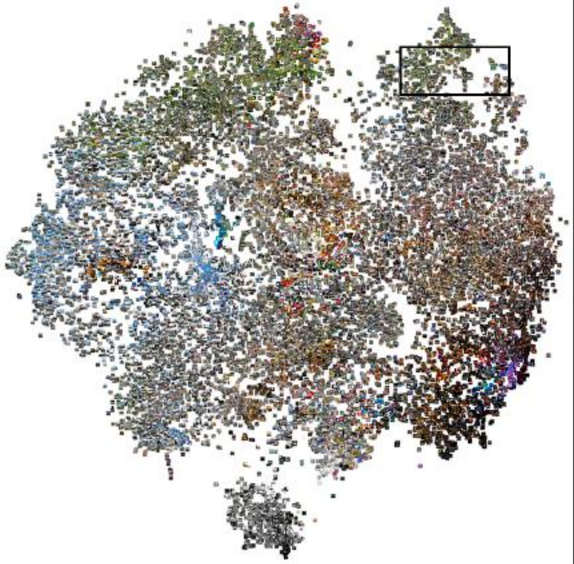


LLC GIST Conv-1 activations Conv-6 activations

Donahue et al. **DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition**, 2014

Transfer Learning with CNNs

- A CNN trained on a (large enough) dataset generalizes to other visual tasks



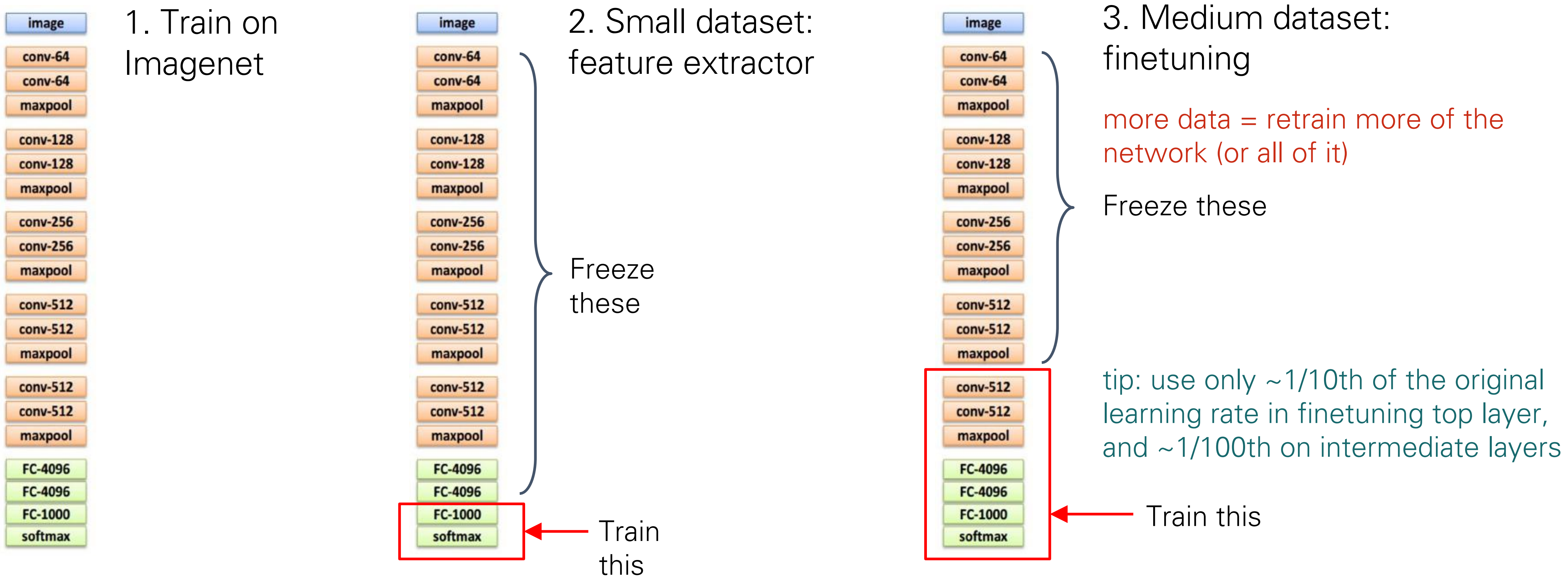
20, 000 Flickr test images

A. Joulin, L.J.P. van der Maaten, A. Jabri, and N. Vasilache **Learning visual features from Large Weakly supervised Data.** ECCV 2016

Slide credit: Joan Bruna

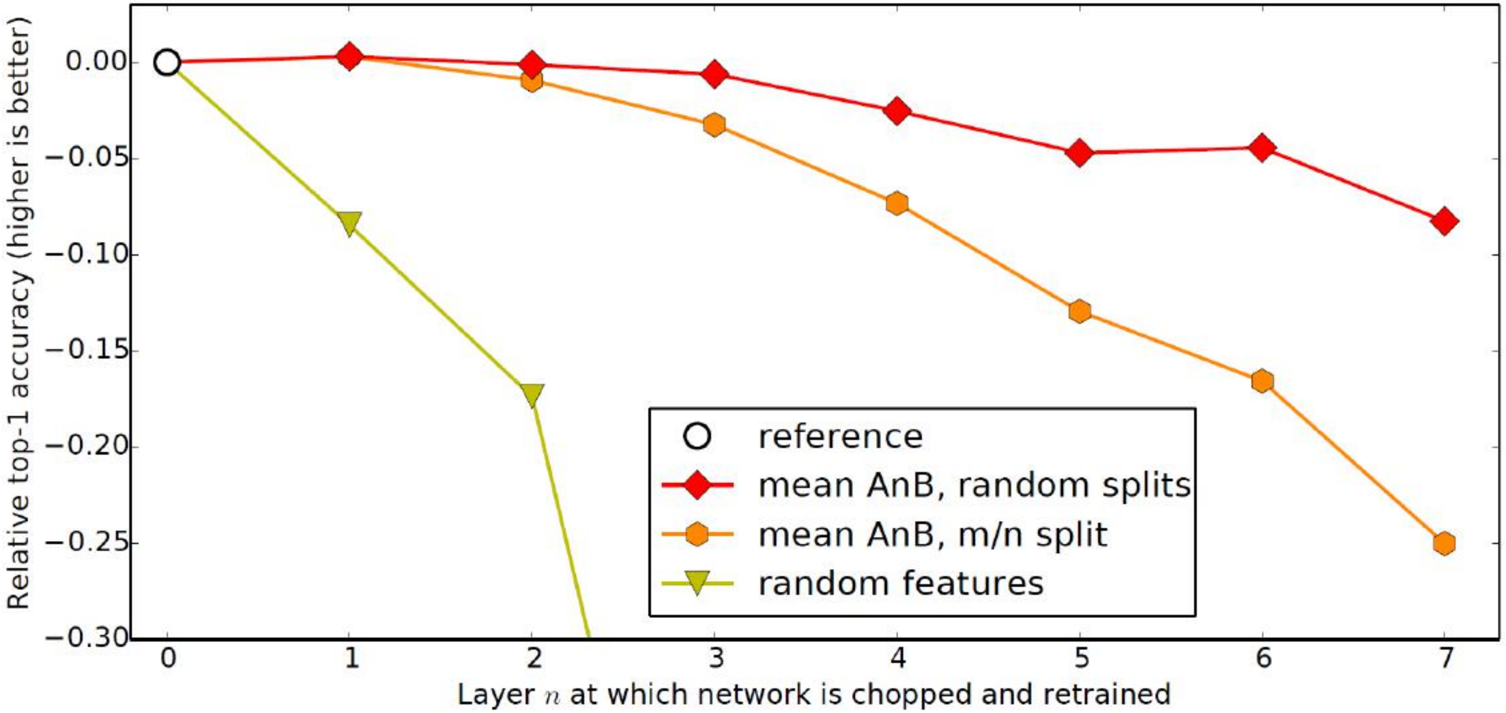
Transfer Learning with CNNs

- Keep layers 1-7 of our ImageNet-trained model fixed
- Train a new softmax classifier on top using the training images of the new dataset.



How transferable are features in CNN networks?

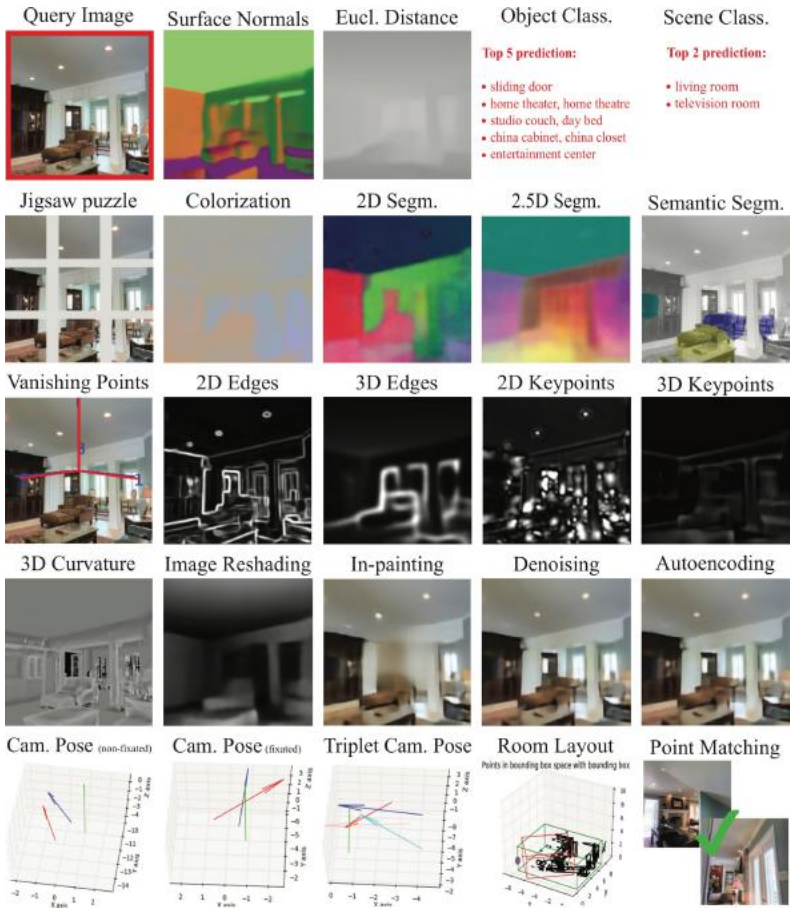
- Divide ImageNet into man-made objects A (449 classes) and natural objects B (551 classes)
- The transferability of features decreases as the distance between the base task and target task increases

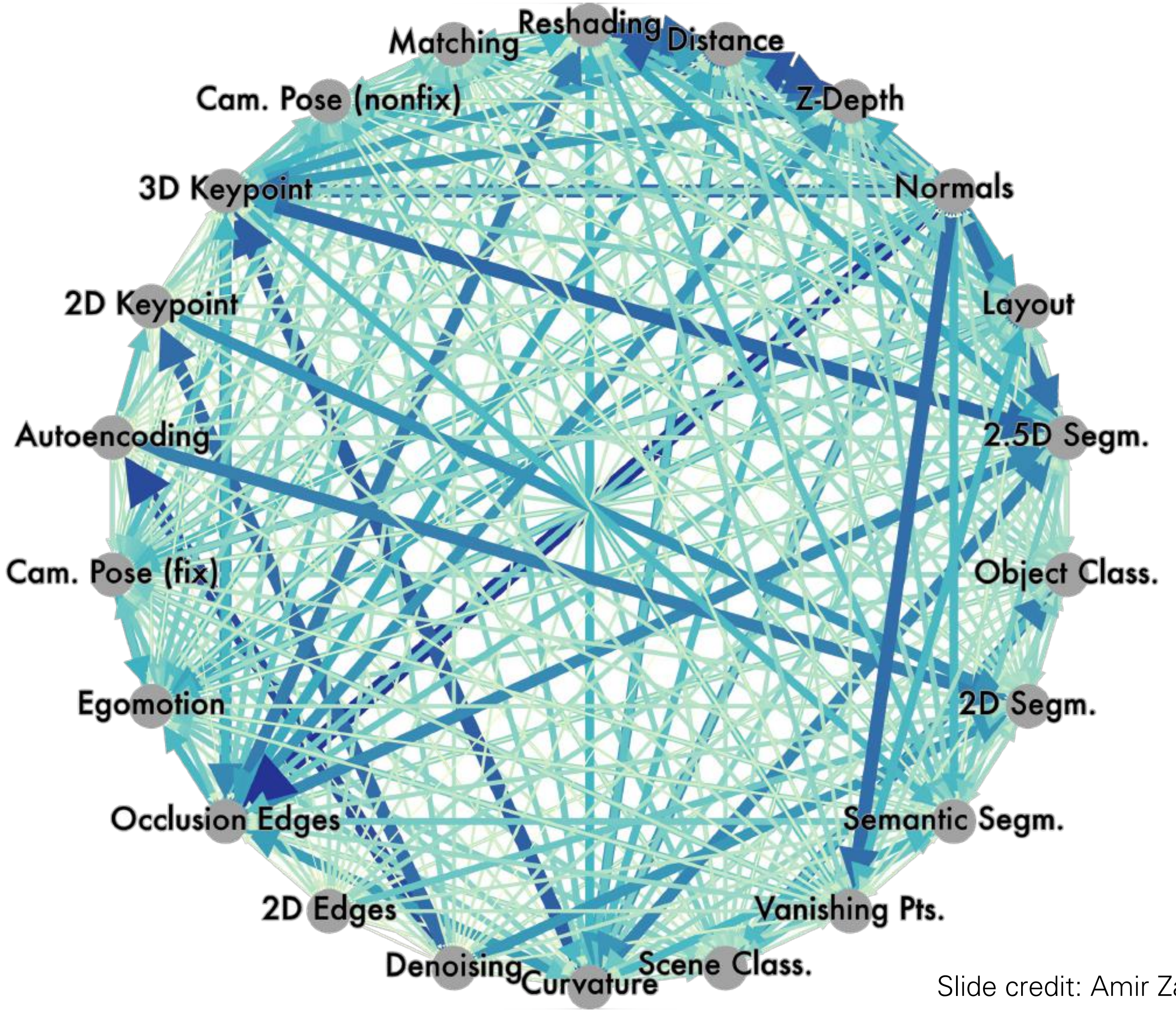


Slide credit: Xiaogan Wang

How transferable are features in CNN networks?

- An open research problem





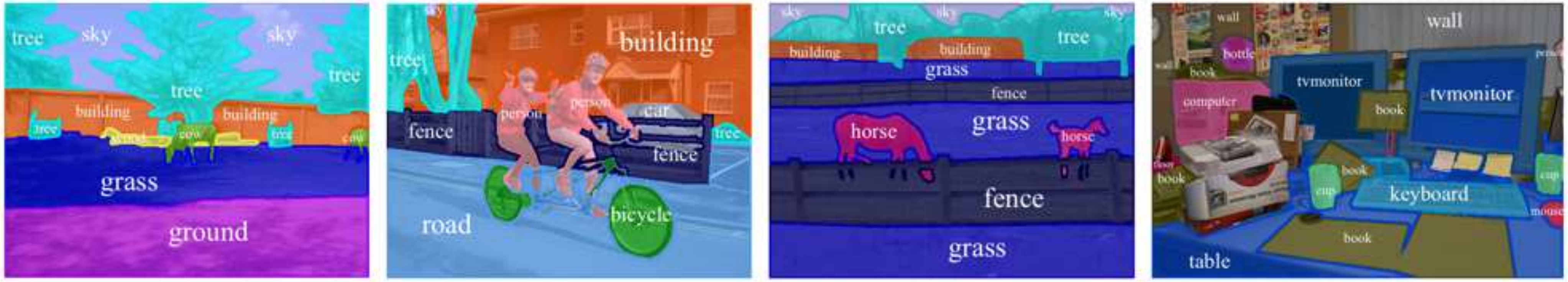
Slide credit: Amir Zamir

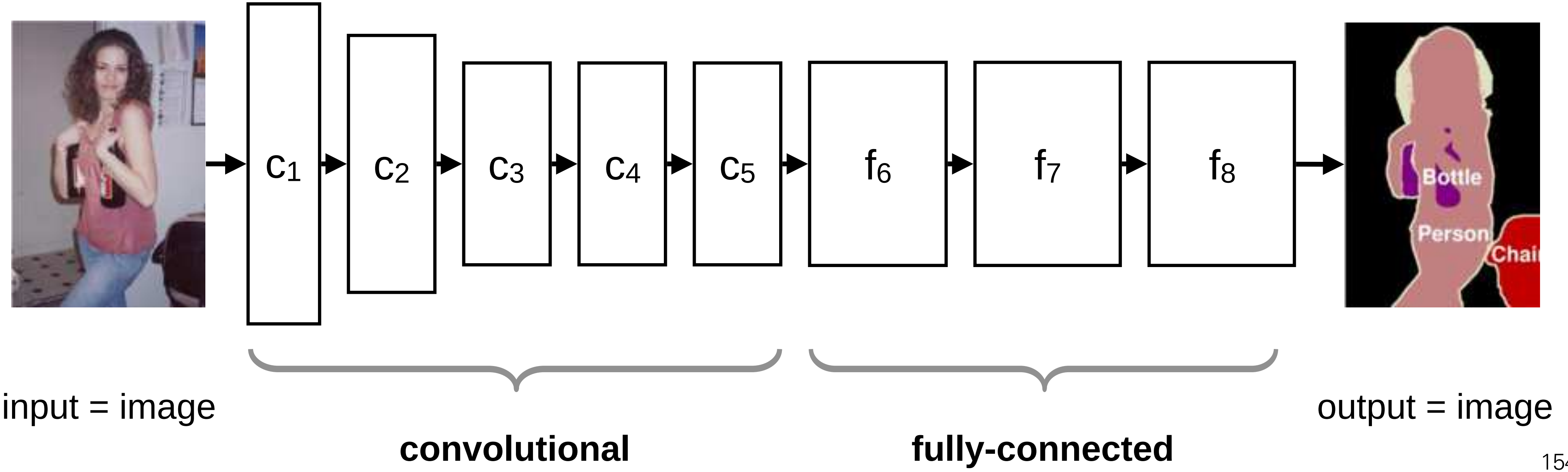
A. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, S. Savarese. **Taskonomy: Disentangling Task Transfer Learning**. CVPR

Semantic Segmentation

Semantic Image Segmentation

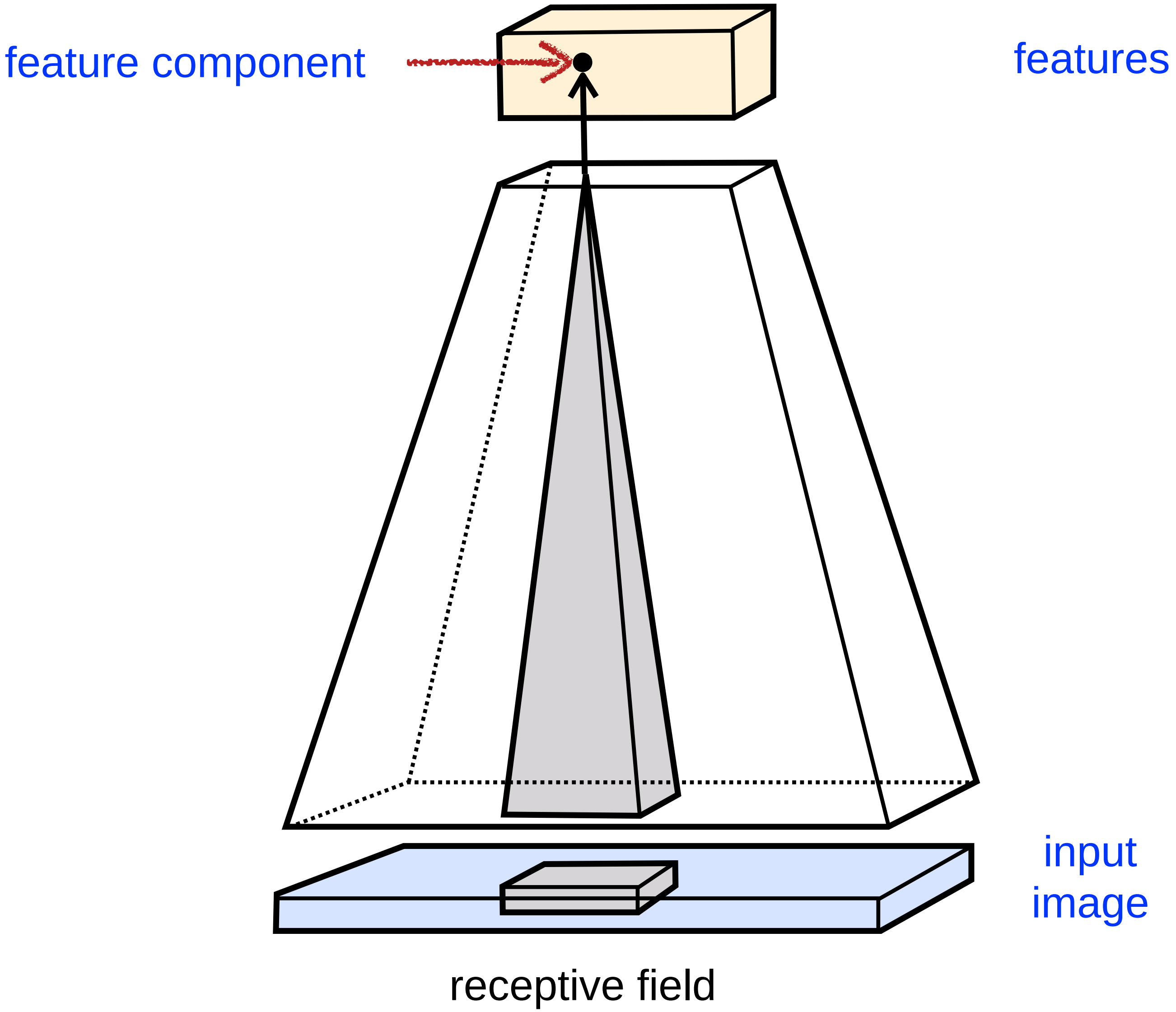
- Label individual pixels





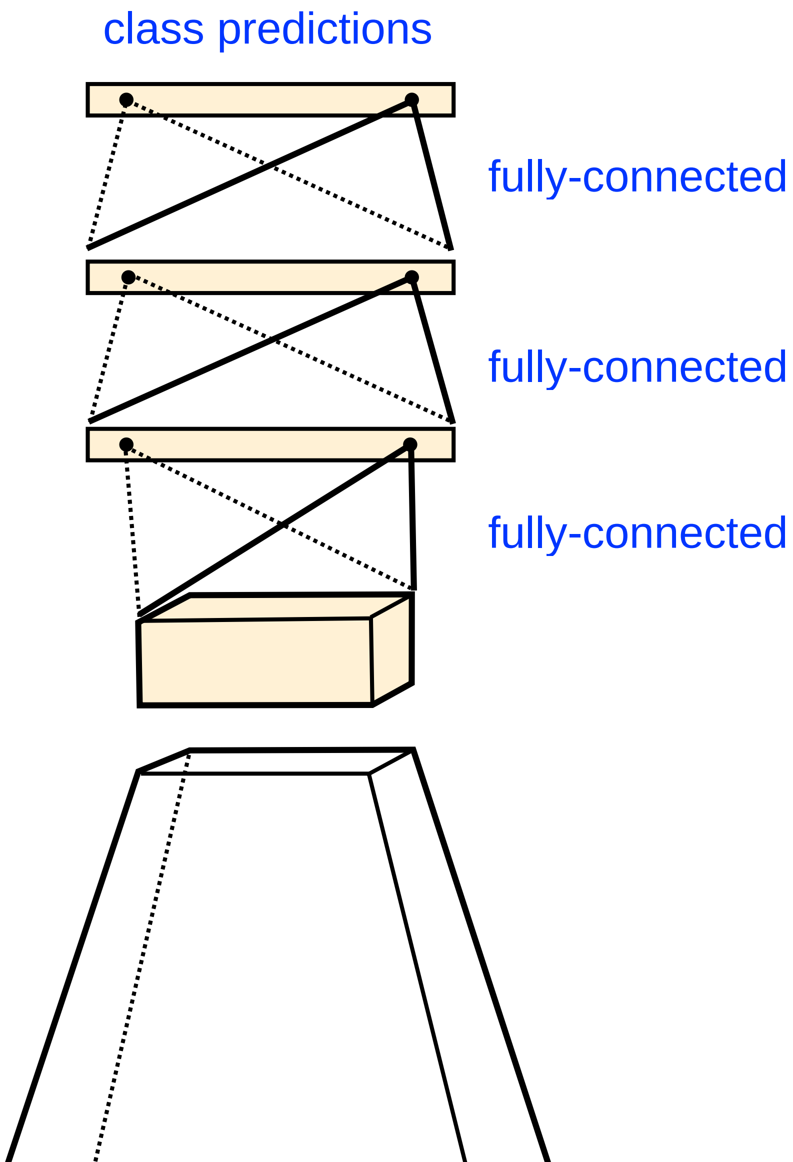
Convolutional Layers

- Local receptive field



Fully Connected Layers

- Global receptive field

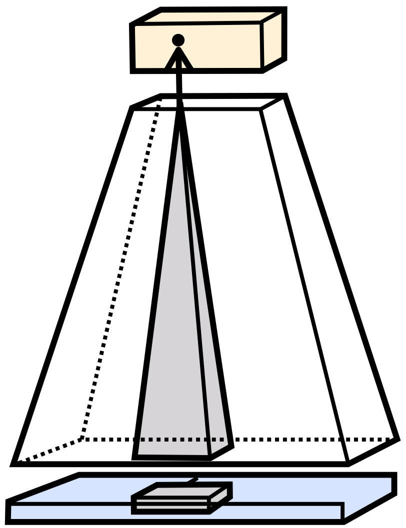


Convolutional vs. Fully Connected

- Comparing the receptive fields

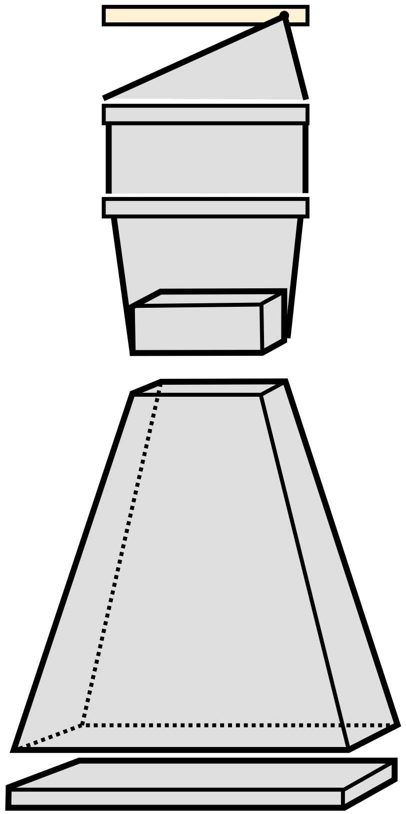
Downsampling filters

Responses are spatially selective, can be used to localize things.



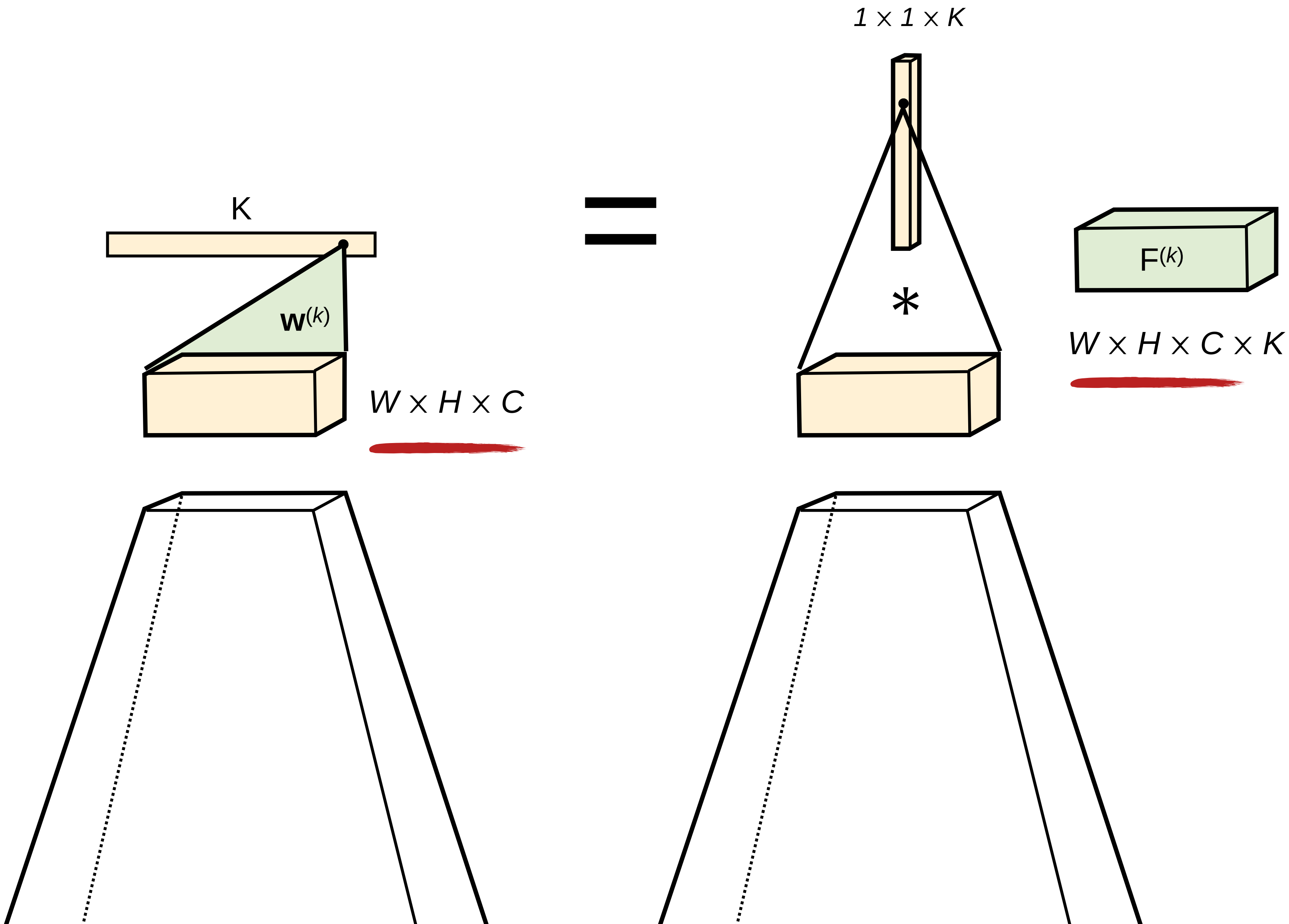
Upsampling filters

Responses are global, do not characterize well position

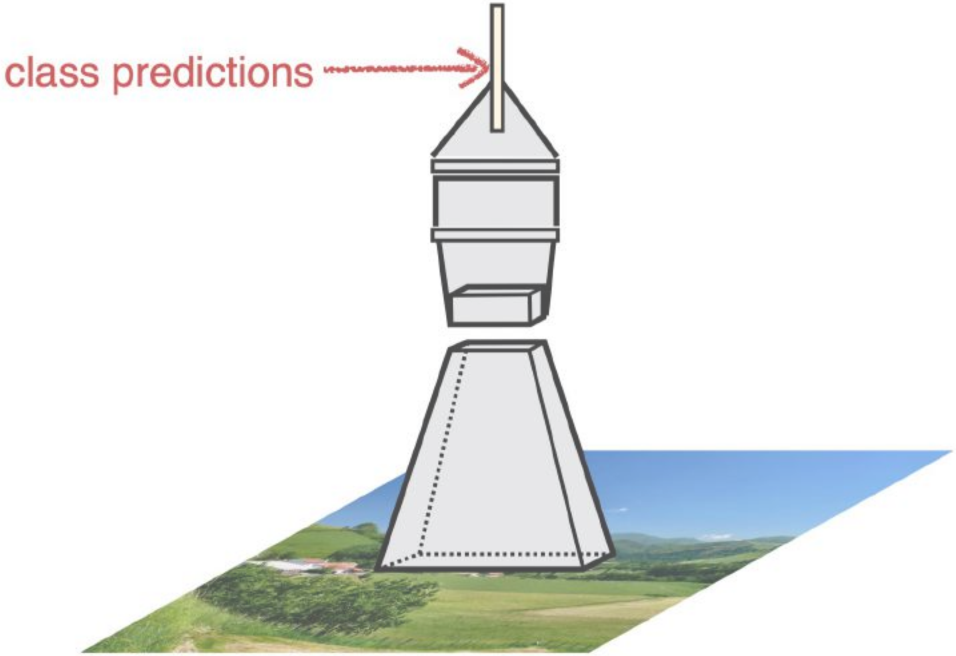


Which one is more useful for pixel level labelling?

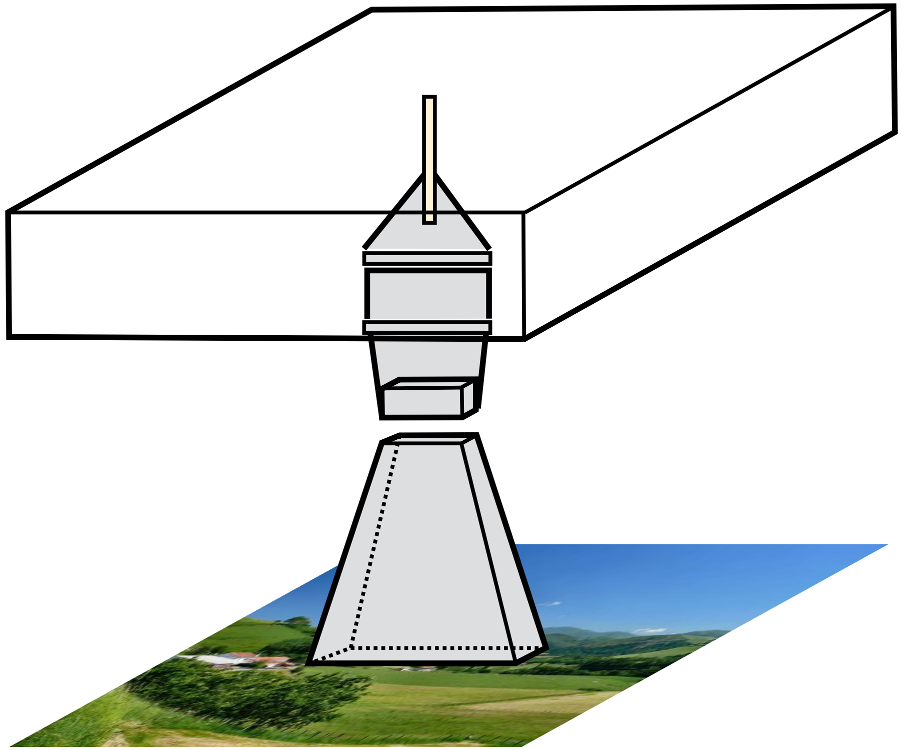
Fully-Connected Layer = Large Filter



Fully-Convolutional Neural Networks



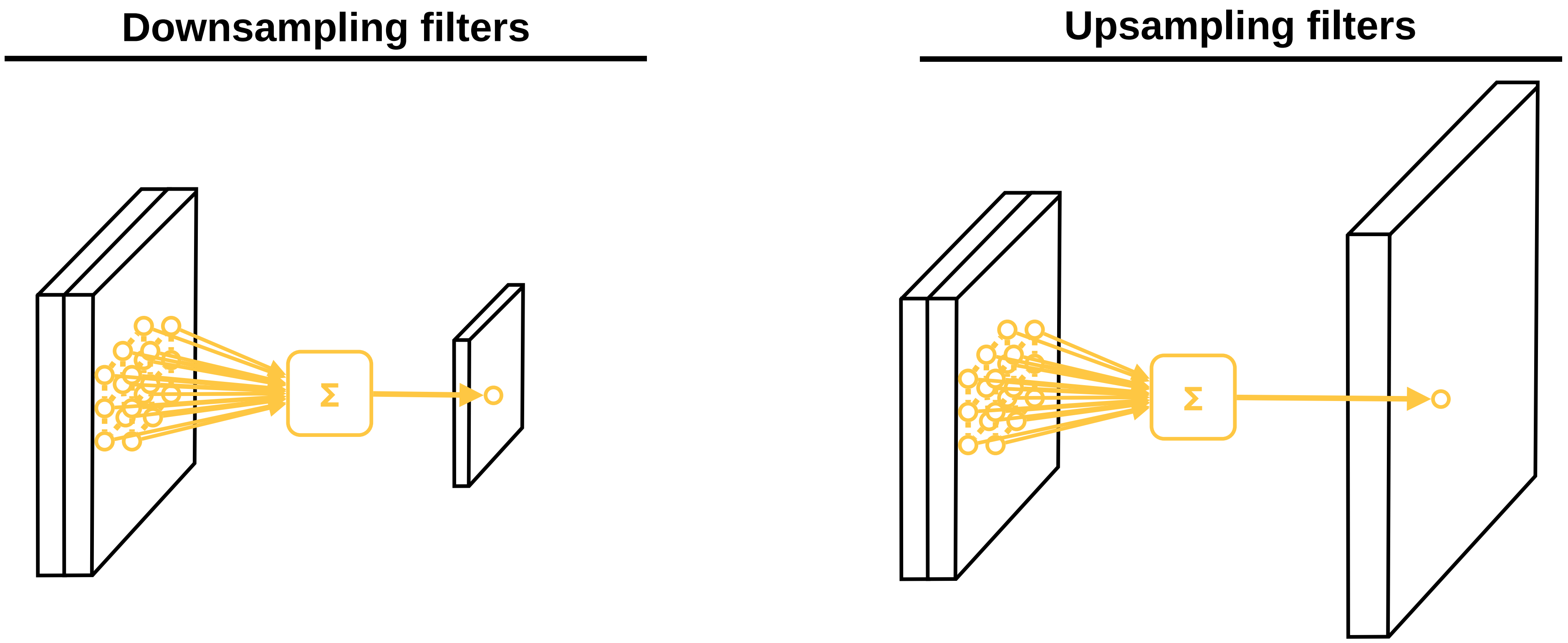
Fully-Convolutional Neural Networks



- **Dense evaluation**
  - Apply the whole network convolutional
  - Estimates a vector of class probabilities at each pixel
- **Downsampling**
  - In practice most network downsample the data fast
  - The output is very low resolution (e.g. 1/32 of original)

Upsampling The Resolution

- Interpolating filter

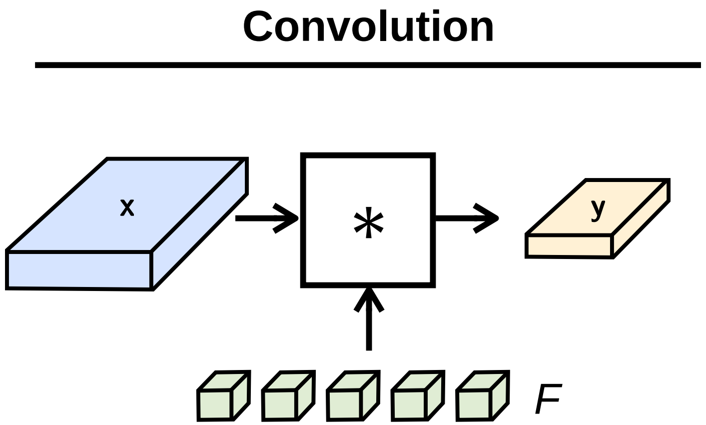


Upsampling filters allow to increase the resolution of the output

Very useful to get full-resolution segmentation results

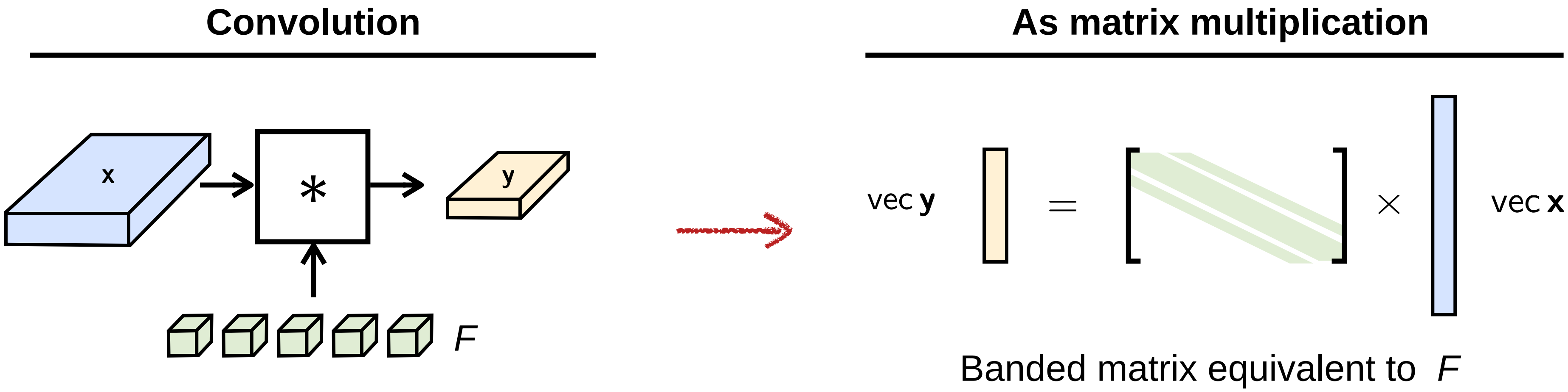
Deconvolution Layer

- Or convolution transpose



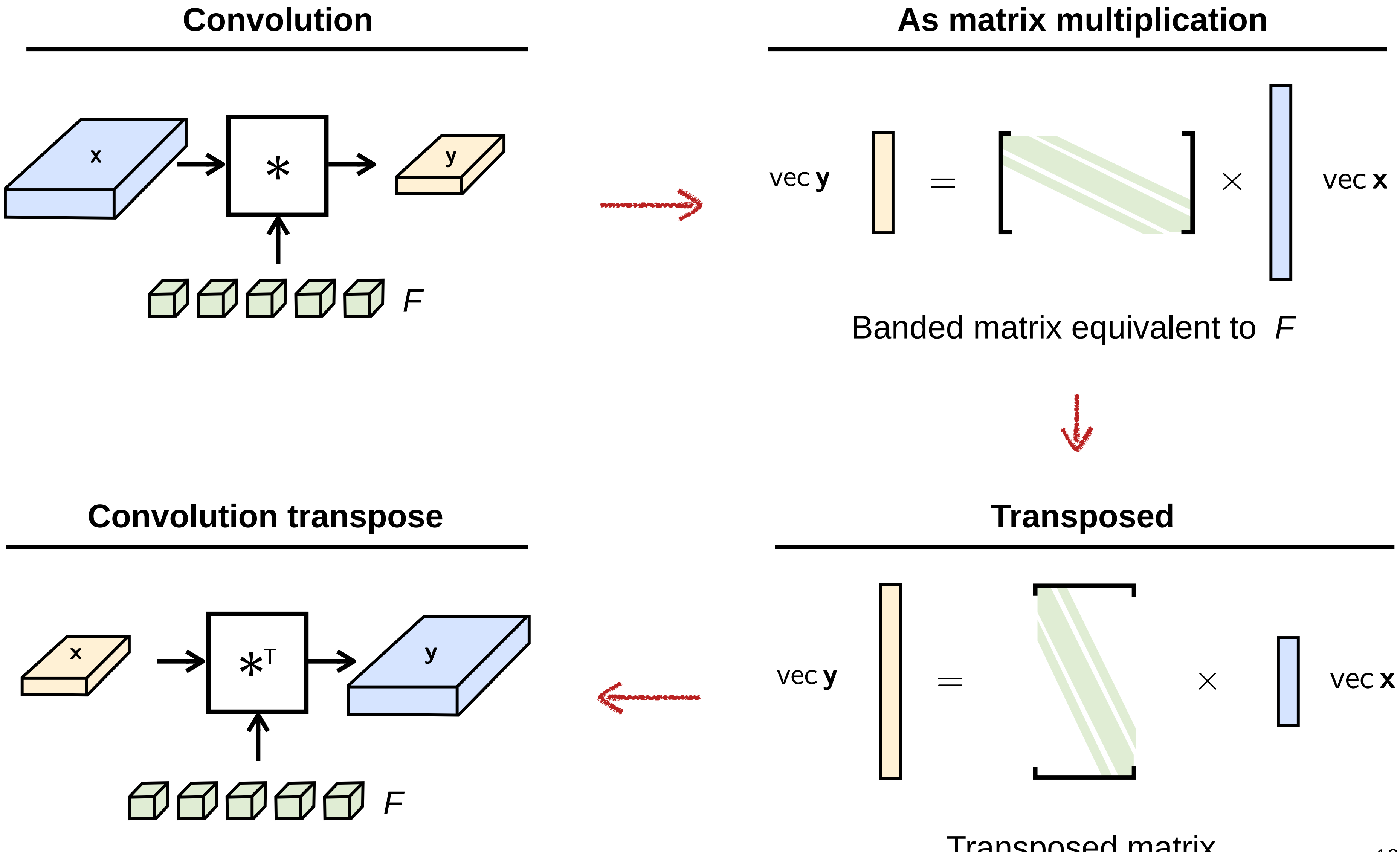
Deconvolution Layer

- Or convolution transpose



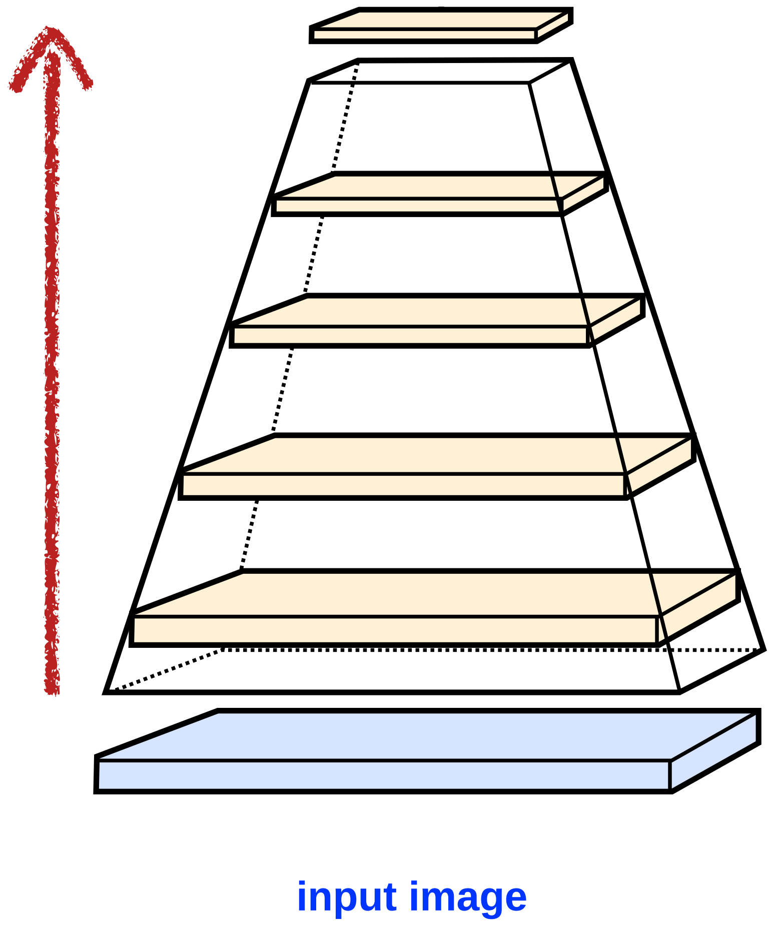
Deconvolution Layer

- Or convolution transpose



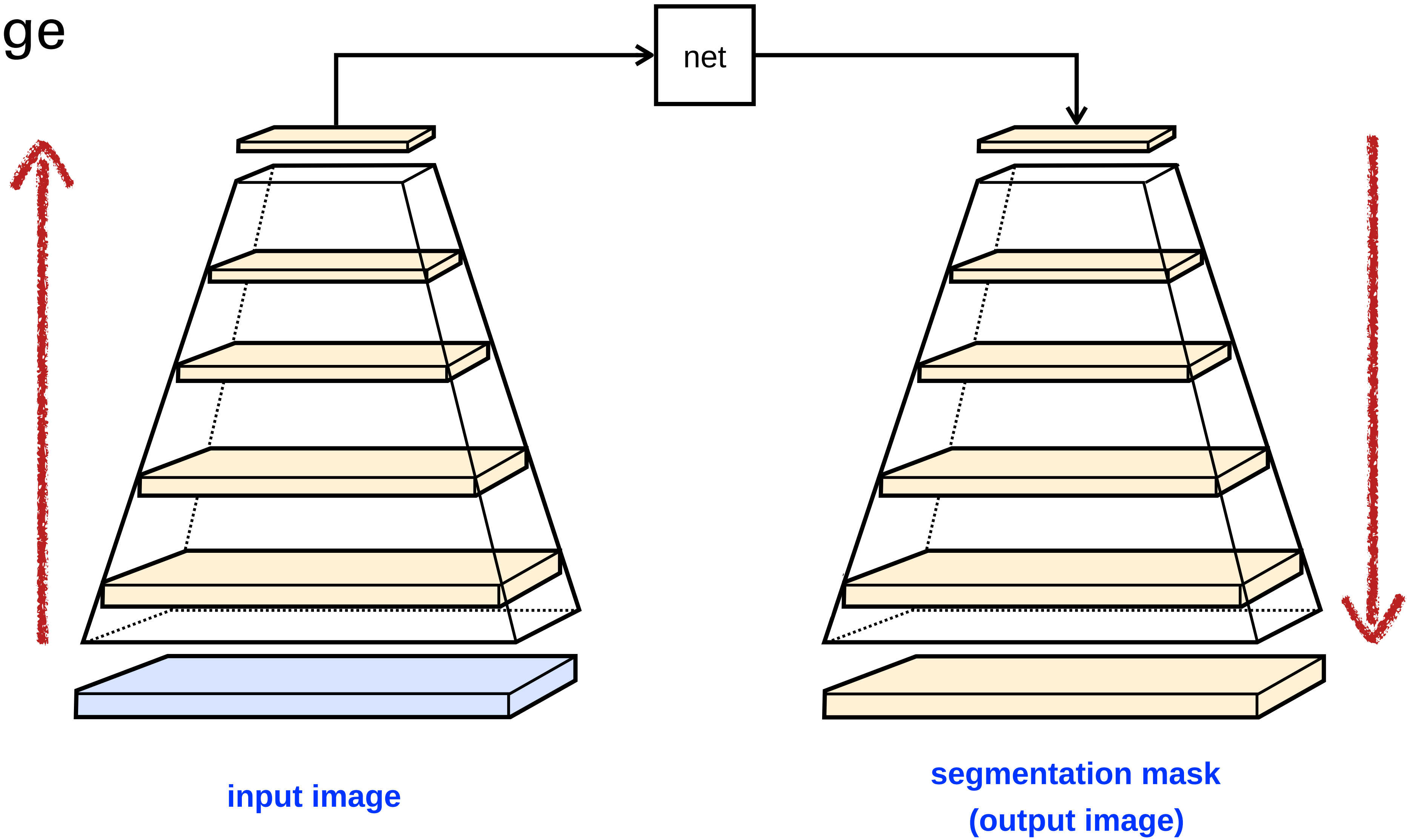
U-Architectures

- Image to image



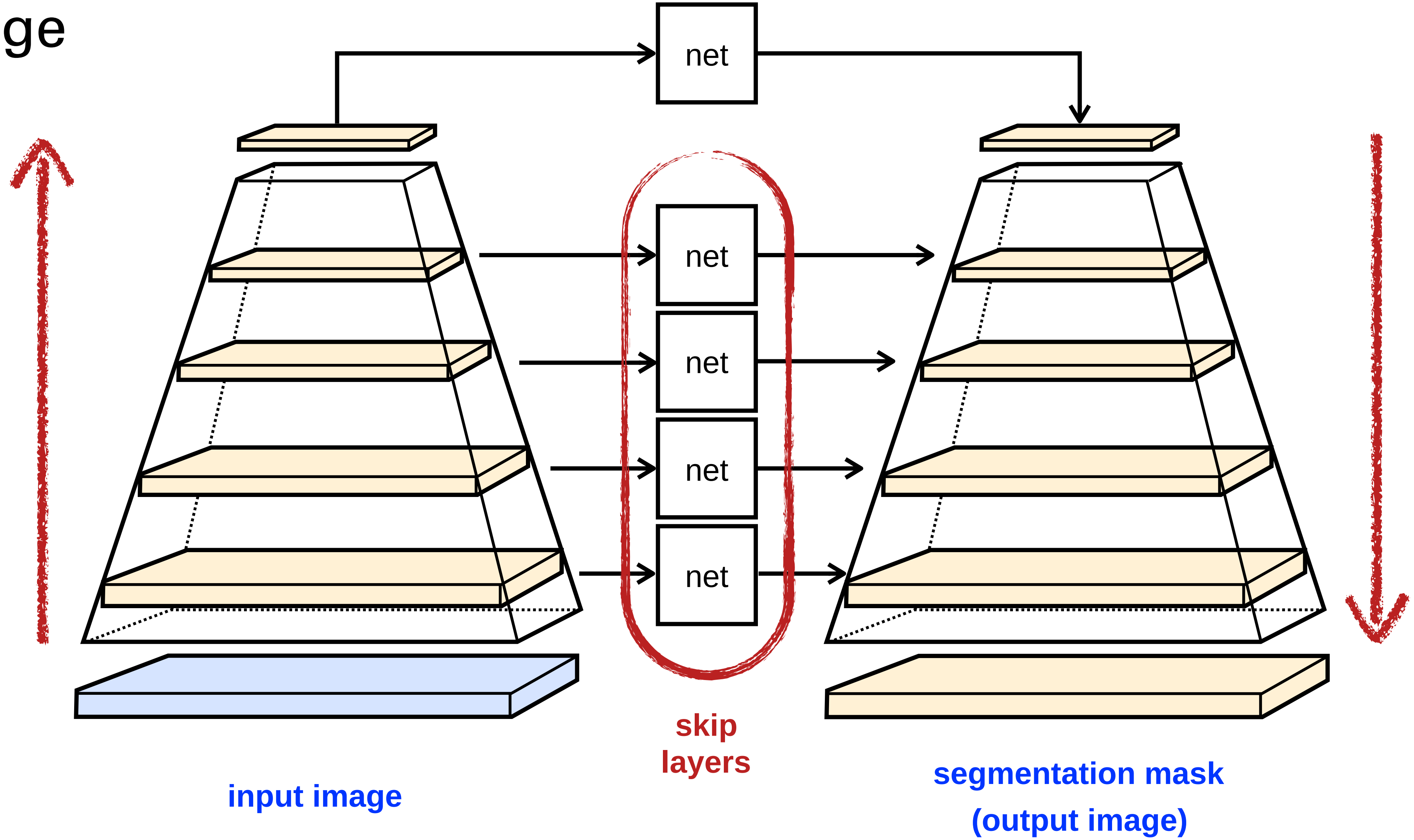
U-Architectures

- Image to image



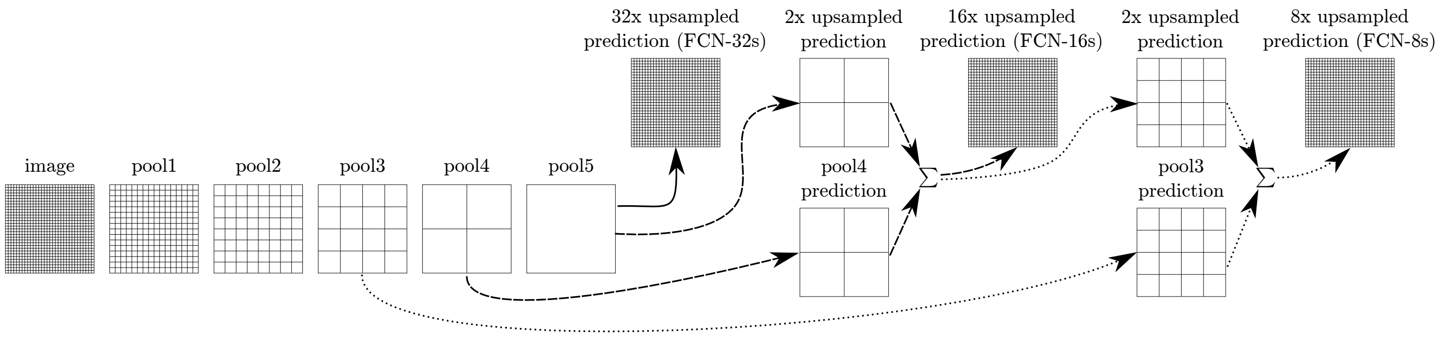
U-Architectures

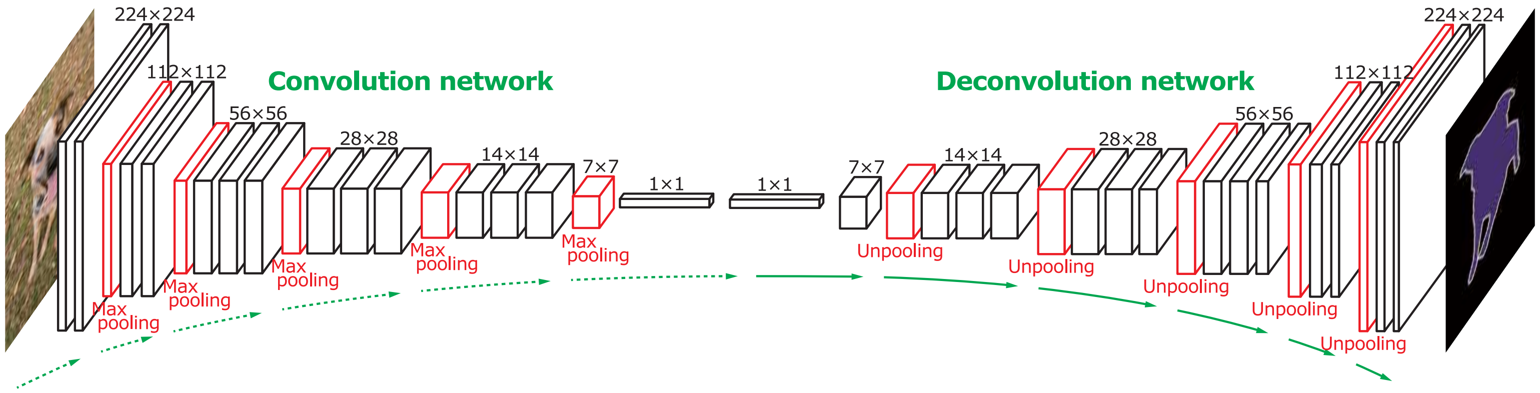
- Image to image

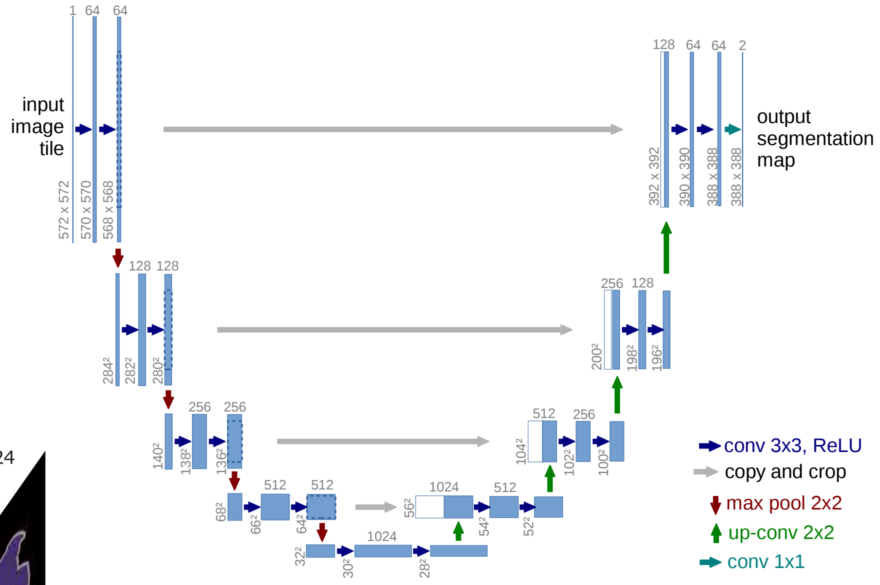


U-Architectures

- Several variants: FCN, U-arch, deconvolution, ...



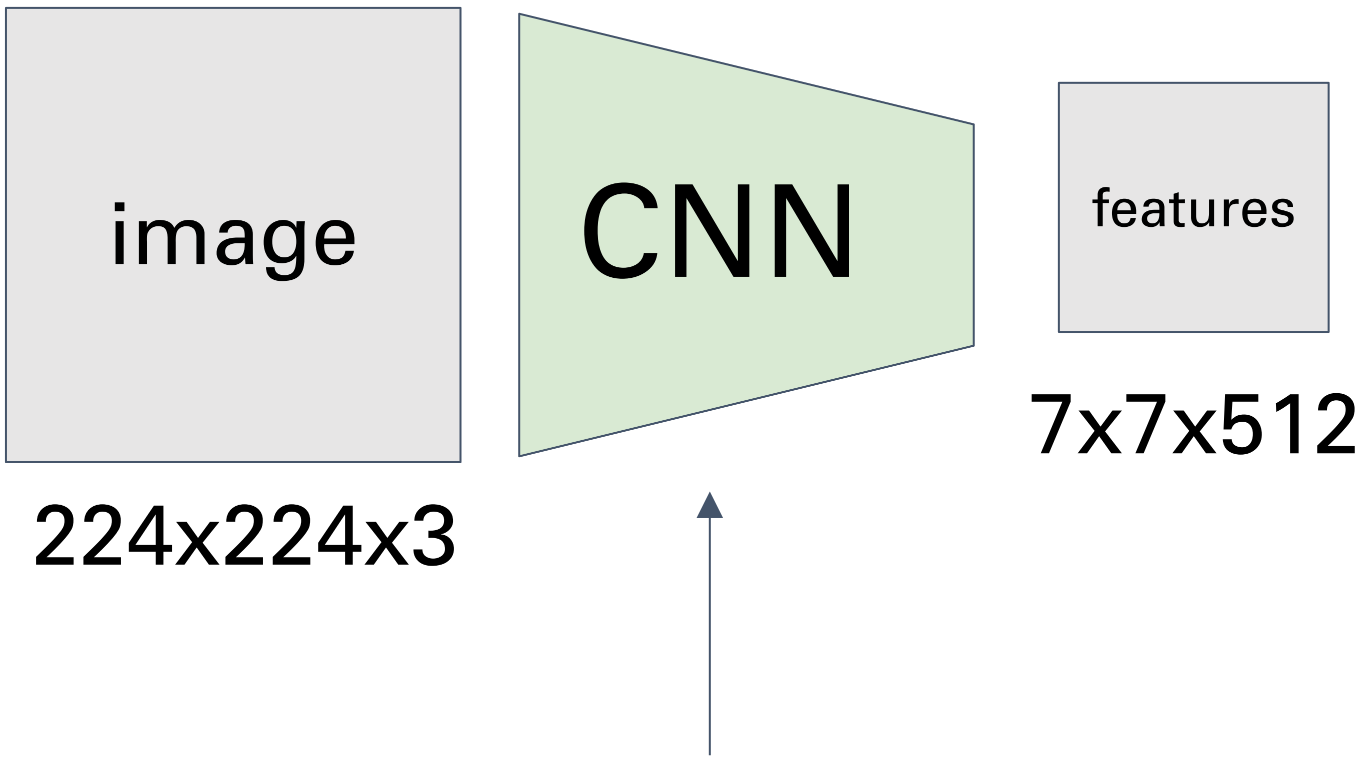




J. Long, E. Shelhamer, and T. Darrell. **Fully convolutional models for semantic segmentation.** In CVPR 2015
H. Noh, S. Hong, and B. Han. **Learning deconvolution network for semantic segmentation.** In ICCV 2015
O. Ronneberger, P. Fischer, and T. Brox. **U-net: Convolutional networks for biomedical image segmentation.** In MICCAI 2015

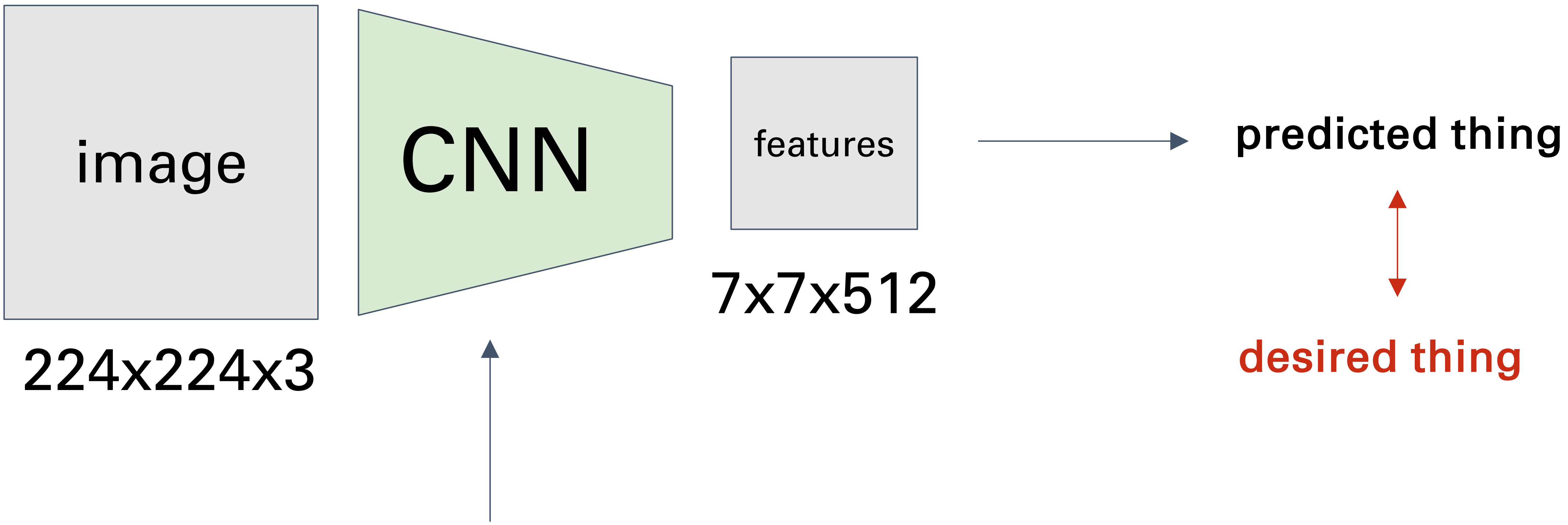
Addressing other tasks...

Addressing other tasks...



A block of compute with a few million parameters.

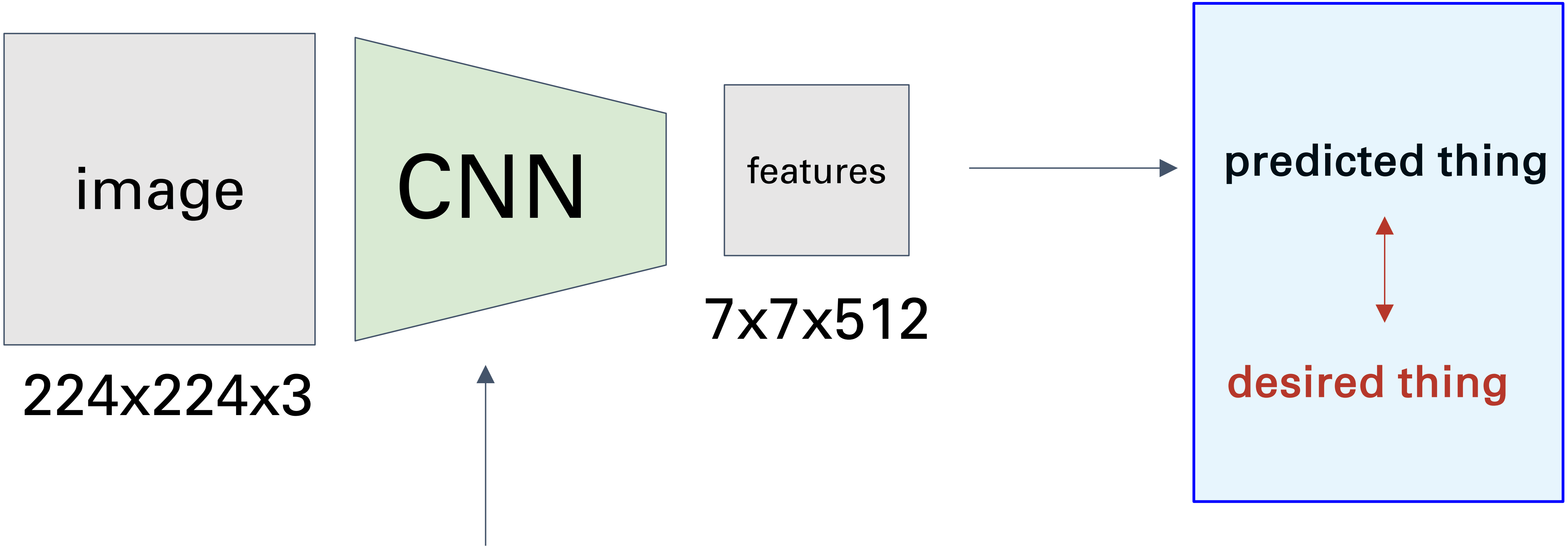
Addressing other tasks...



A block of compute with a few million parameters.

Addressing other tasks...

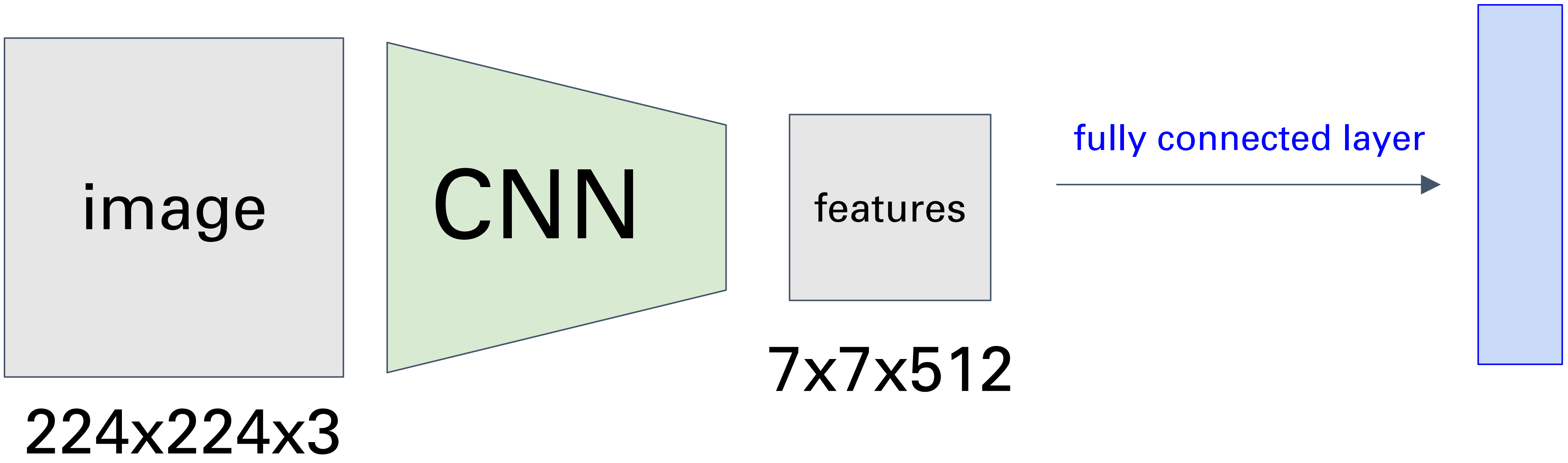
this part changes from task to task



A block of compute with a few million parameters.

Image Classification

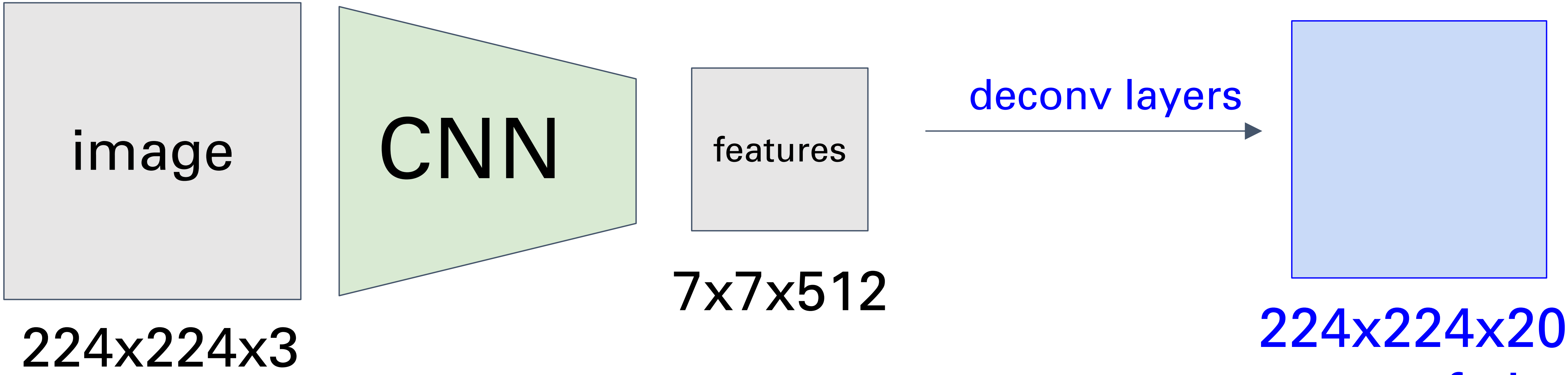
thing = a vector of probabilities for different classes



e.g. vector of 1000 numbers giving probabilities for different classes.

Segmentation

image class “map”



array of class probabilities at each pixel.

Localization

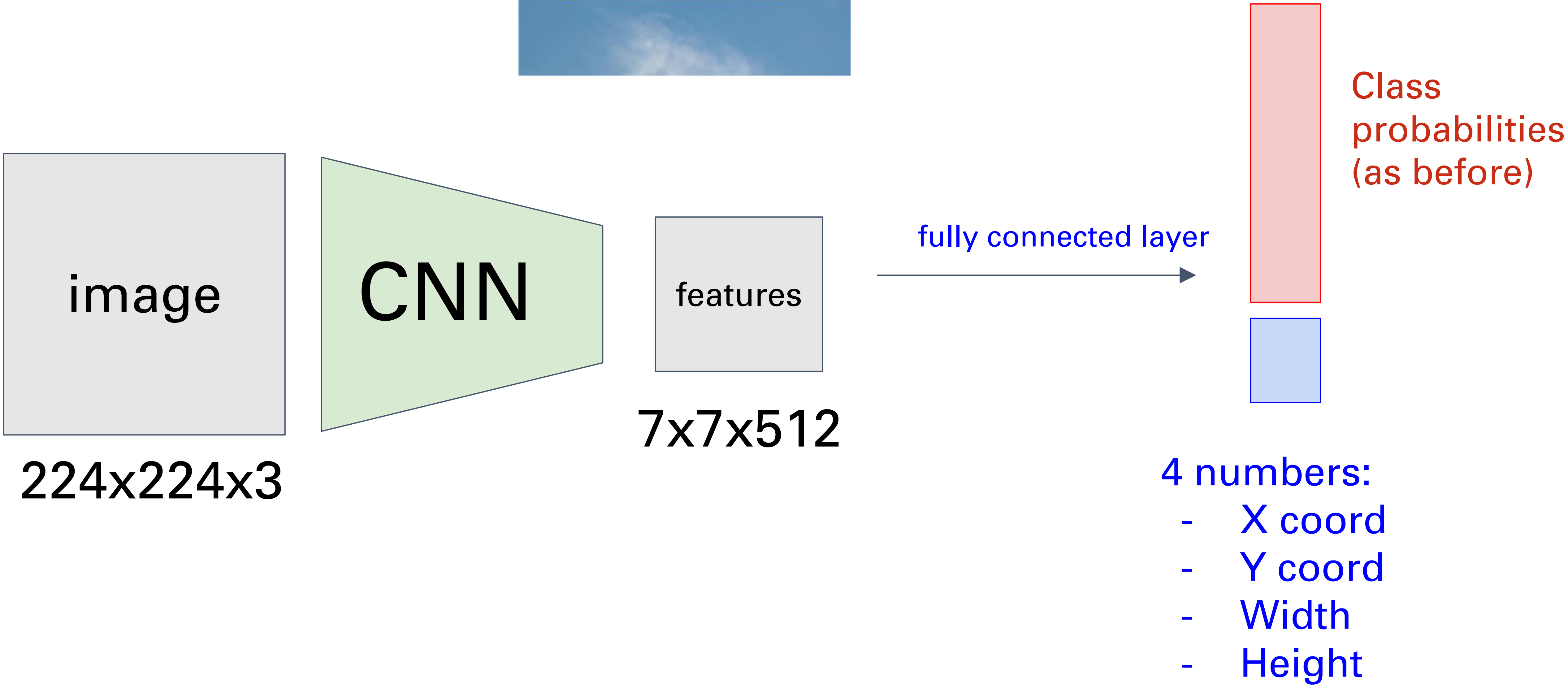
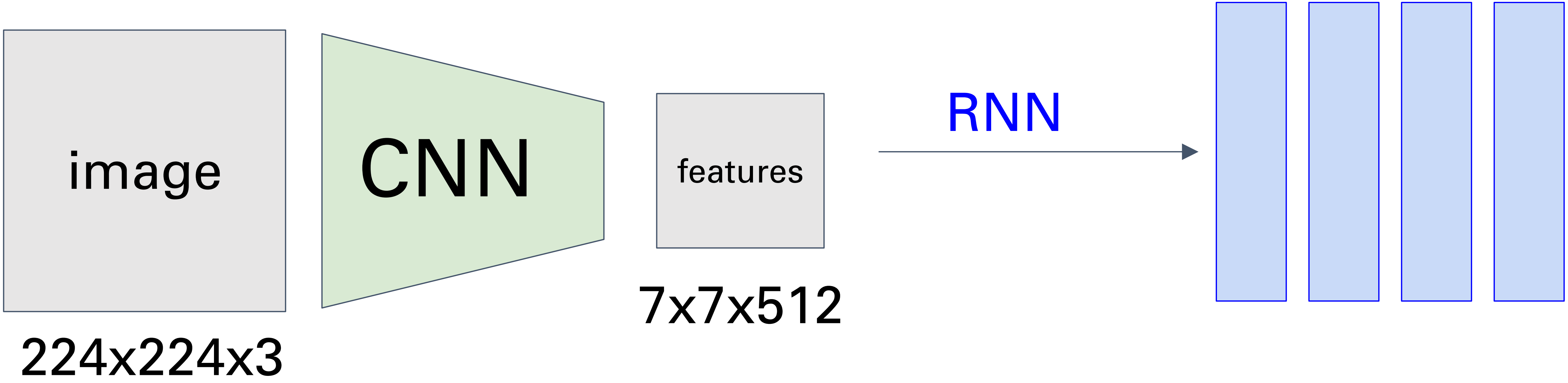
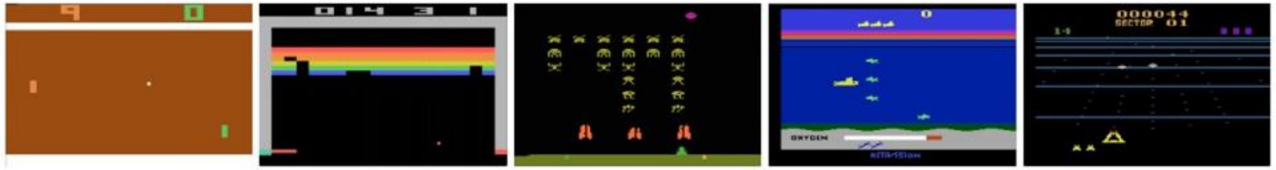


Image Captioning

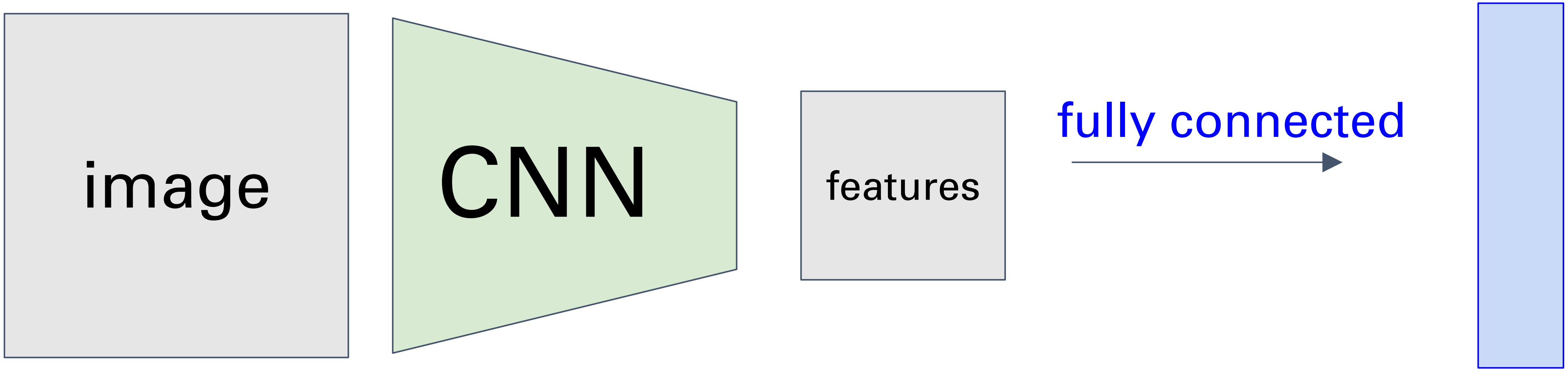


A sequence of 10,000-dimensional vectors giving probabilities of different words in the caption.

Reinforcement Learning



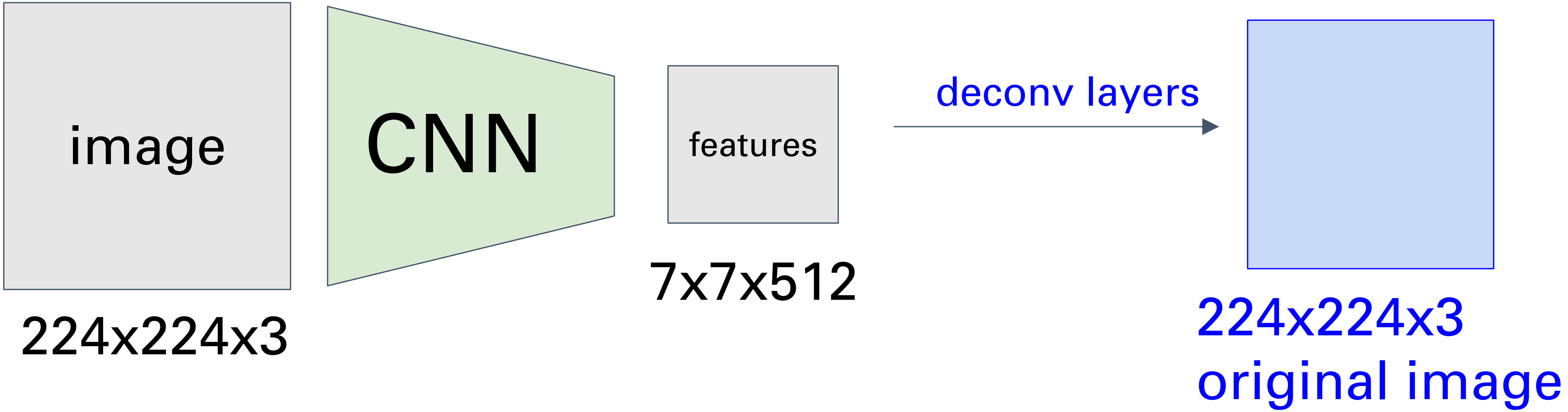
Mnih et al. 2015



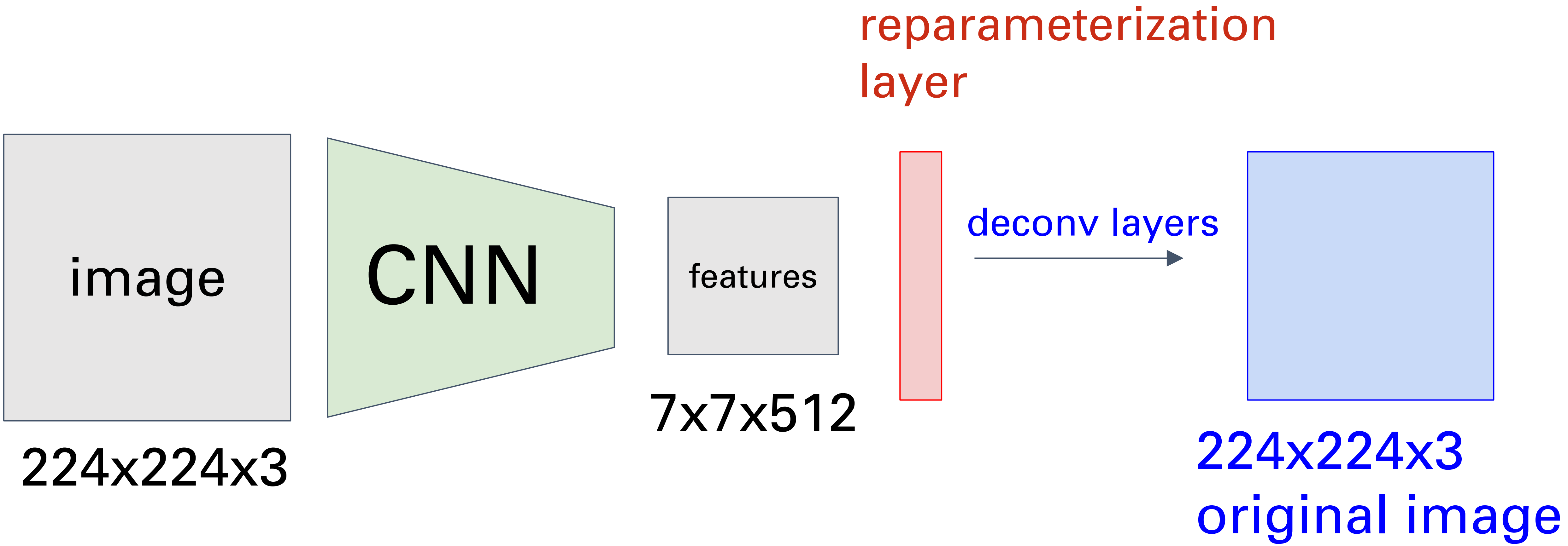
160x210x3

e.g. vector of 8 numbers giving probability of wanting to take any of the 8 possible ATARI actions.

Autoencoders

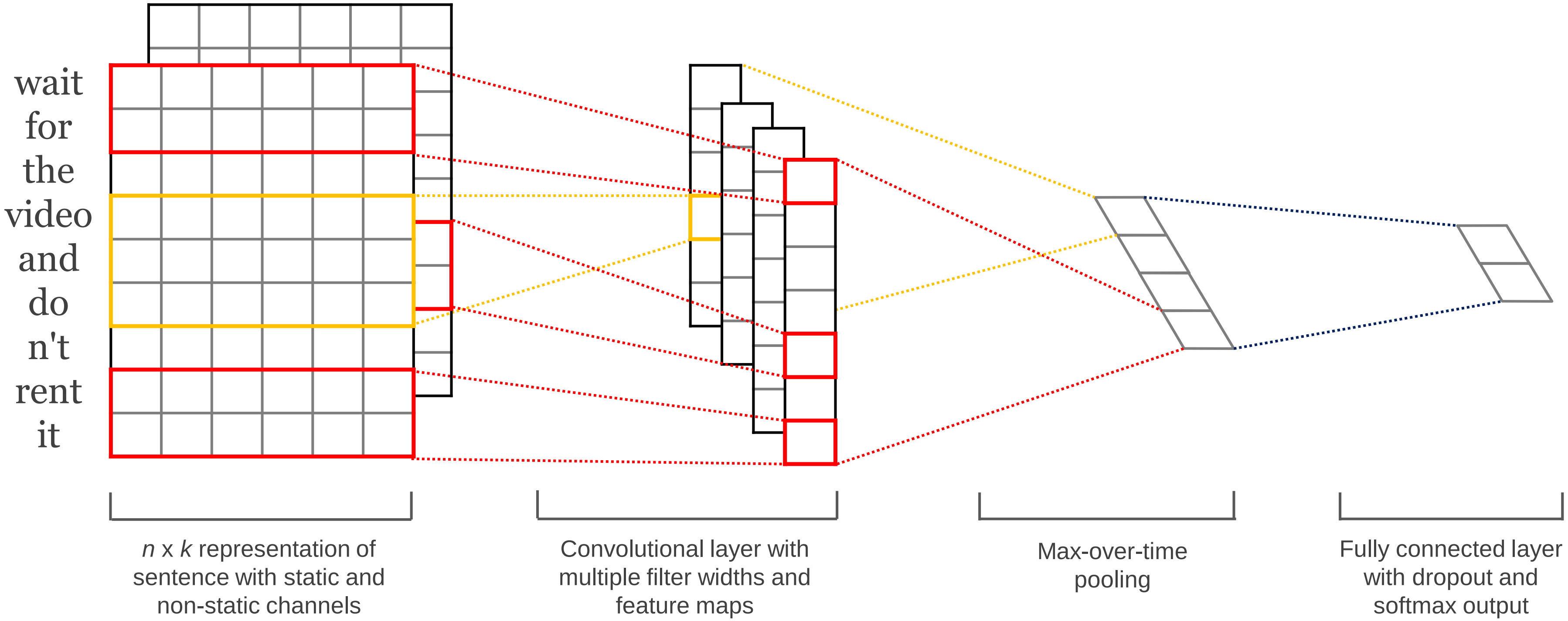


Variational Autoencoders



[Kingma et al.], [Rezende et al.], [Salimans et al.]

Addressing other tasks…

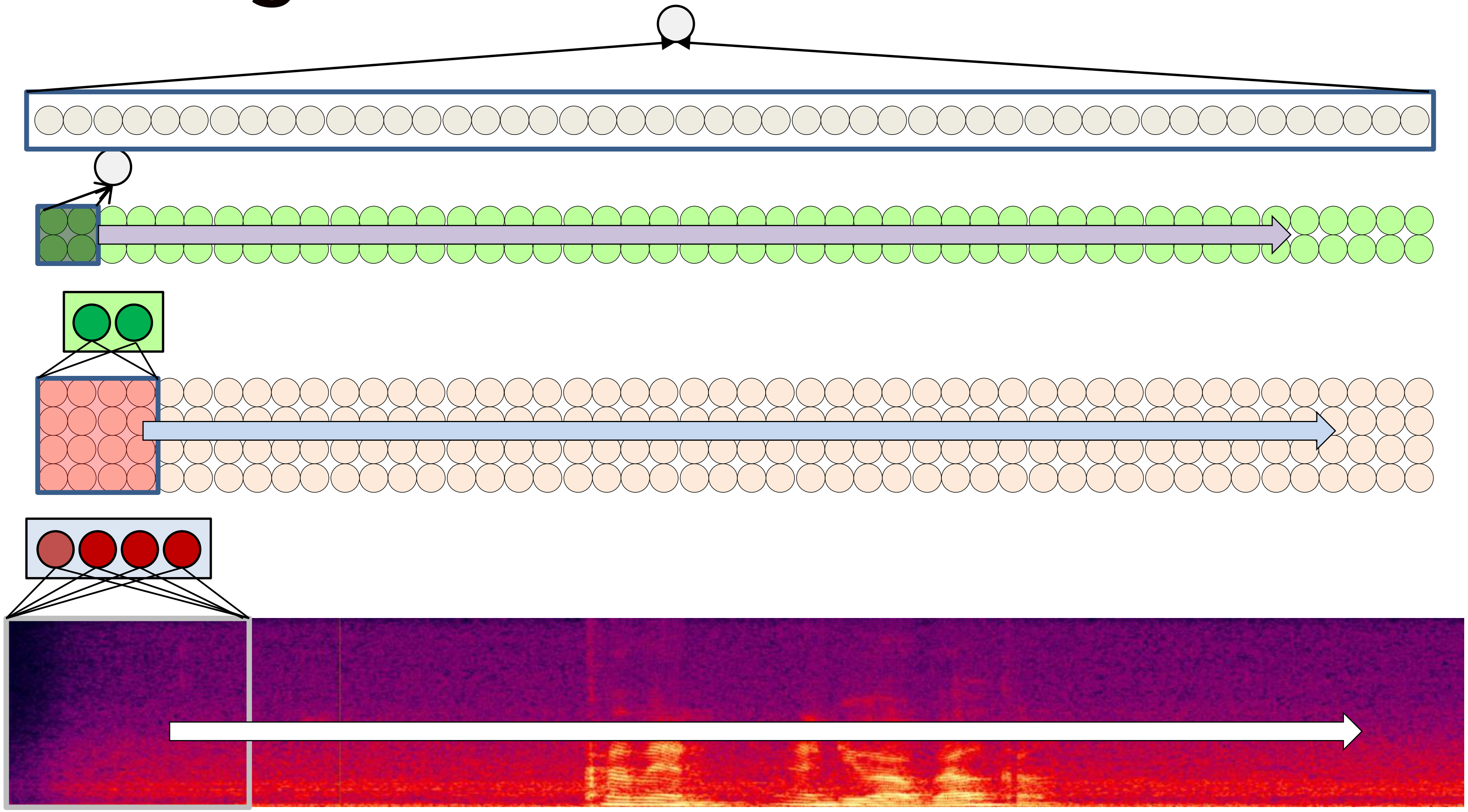


- 1D convolution ≈ Time Delay Neural Networks (Waibel et al. 1989, Collobert and Weston 2011)
- Two main paradigms:
  - **Context window modeling:** For tagging, etc. get the surrounding context before tagging
  - **Sentence modeling:** Do convolution to extract n-grams, pooling to combine over whole sentence

Figure credit: Yoon Kim

Slide adapted from Graham Neubig

Addressing other tasks...



- **CNNs for audio processing:** MFCC features + Time Delay Neural Networks

Figure credit: Bhiksha Raj

Next lecture:
Understanding and Visualizing ConvNets