detail from the visualization of ResNet-18 // Graphcore

CMP784

DEEP LEARNING

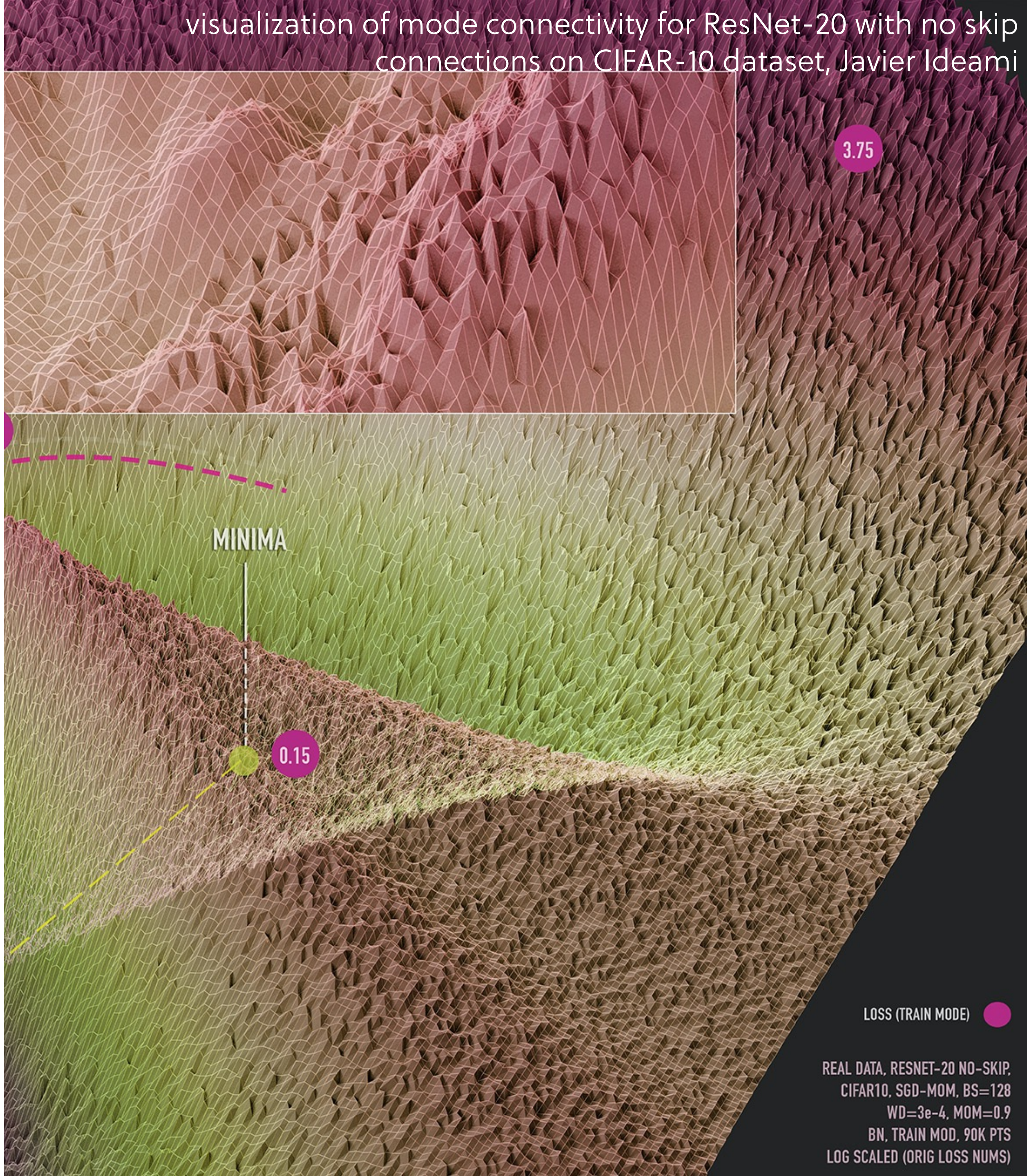
Lecture #05 – Convolutional Neural Networks (CNNs)

Erkut Erdem // Hacettepe University // Fall 2022

HACETTEPE
UNIVERSITY
COMPUTER
VISION LAB

Previously on CMP784

- data preprocessing and normalization
- weight initializations
- ways to improve generalization
- babysitting the learning process
- hyperparameter selection
- optimization



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

Lecture Overview

- convolution layer
- design guidelines for CNNs
- CNN architectures
- transfer learning
- semantic segmentation networks
- object detection networks

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

— Ross Girshick's talk on The Past, Present, and Future of Object Detection

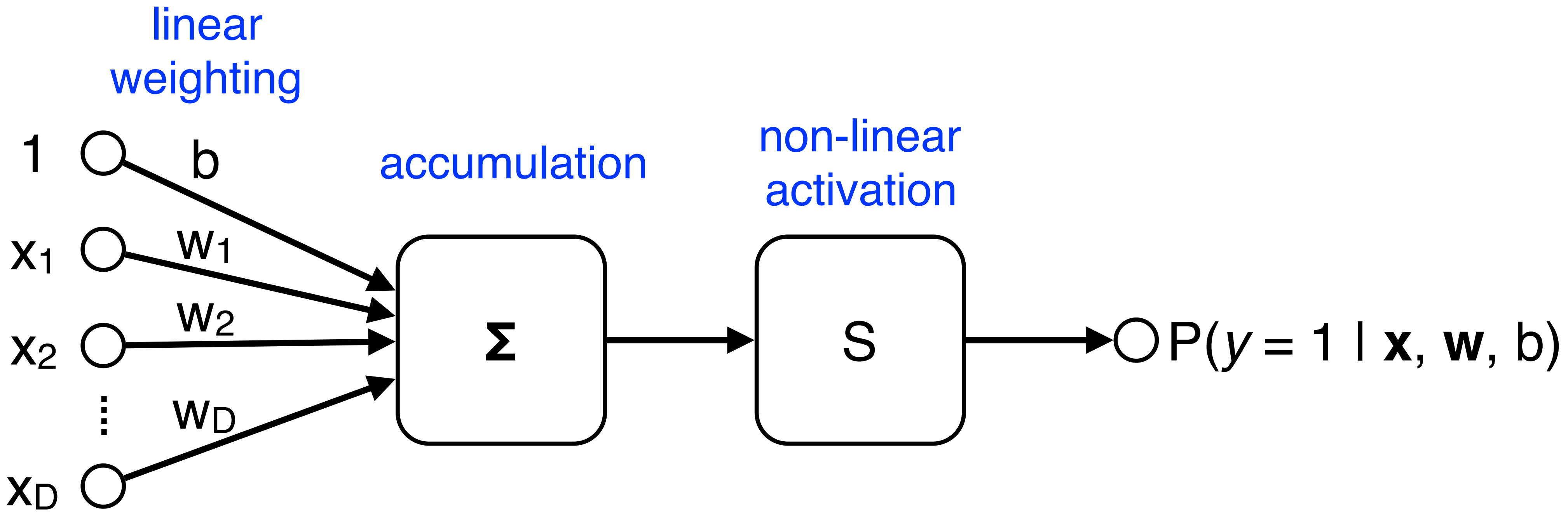
— Fei-Fei Li, Andrej Karpathy and Justin Johnson's CS231n class

— Justin Johnson's EECS 498/598 class

Perceptron

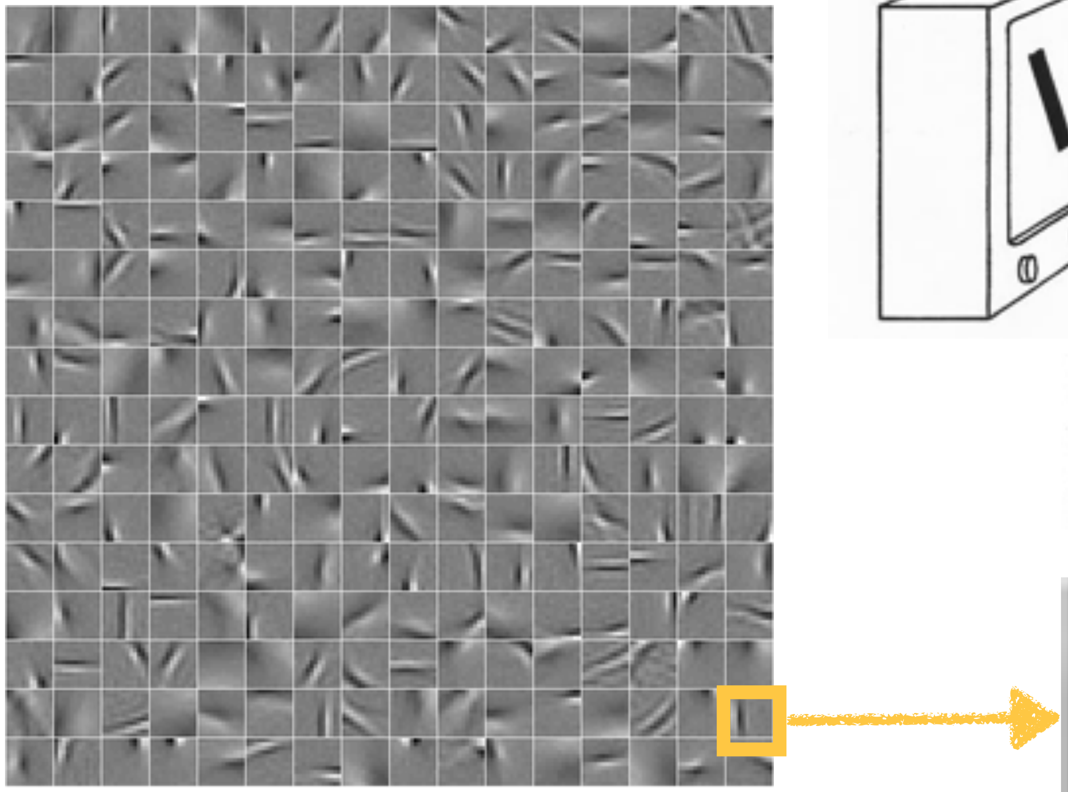
[Rosenblatt 57]

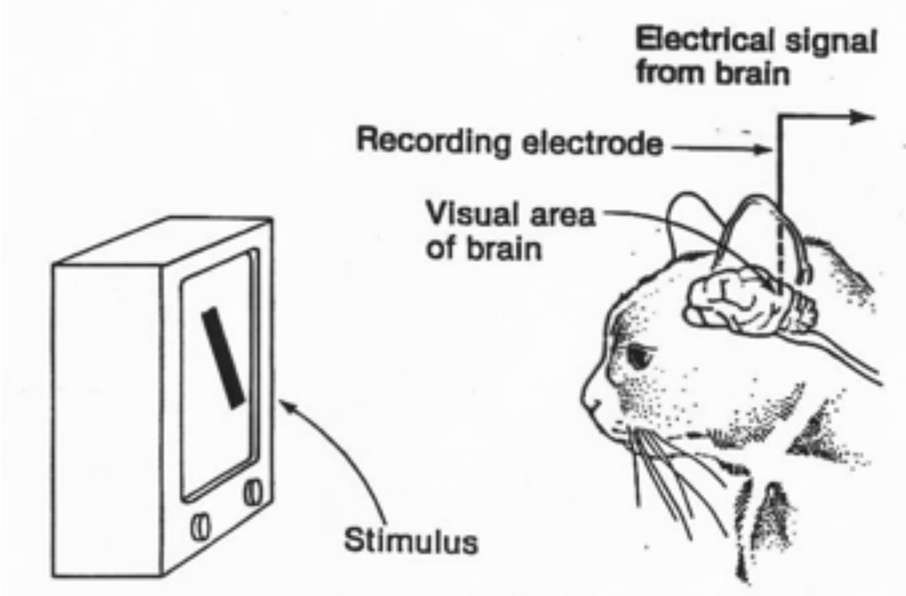
- The goal is estimating the posterior probability of the binary label y of a vector **x**:

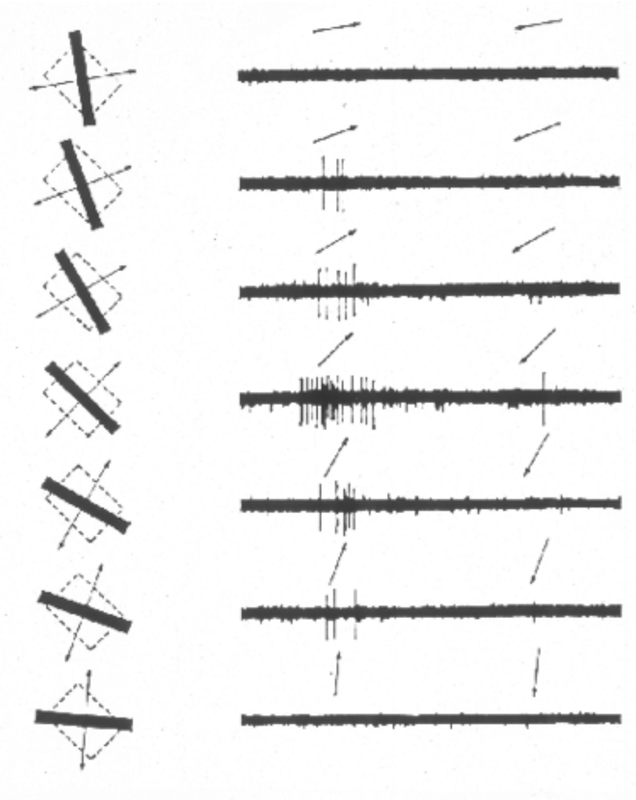


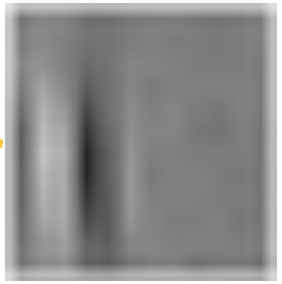
Discovery of oriented cells in the visual cortex

[Hubel and Wiesel 59]



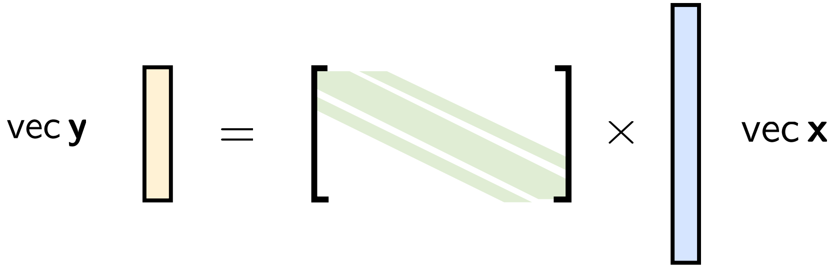






oriented filter

Convolution



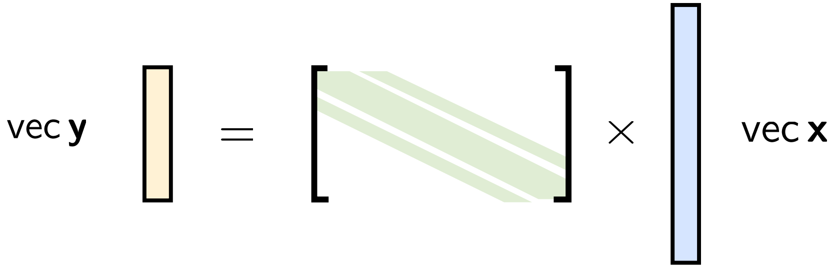
- Convolution = Spatial filtering

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

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

$$* \, ^{1/8} \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

Convolution



- Convolution = Spatial filtering

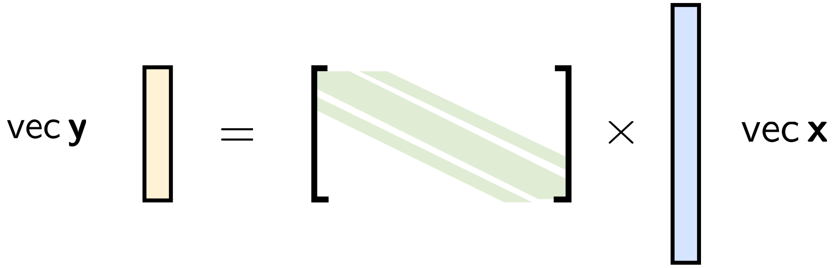
$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

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

$*$

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

Convolution



- Convolution = Spatial filtering

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

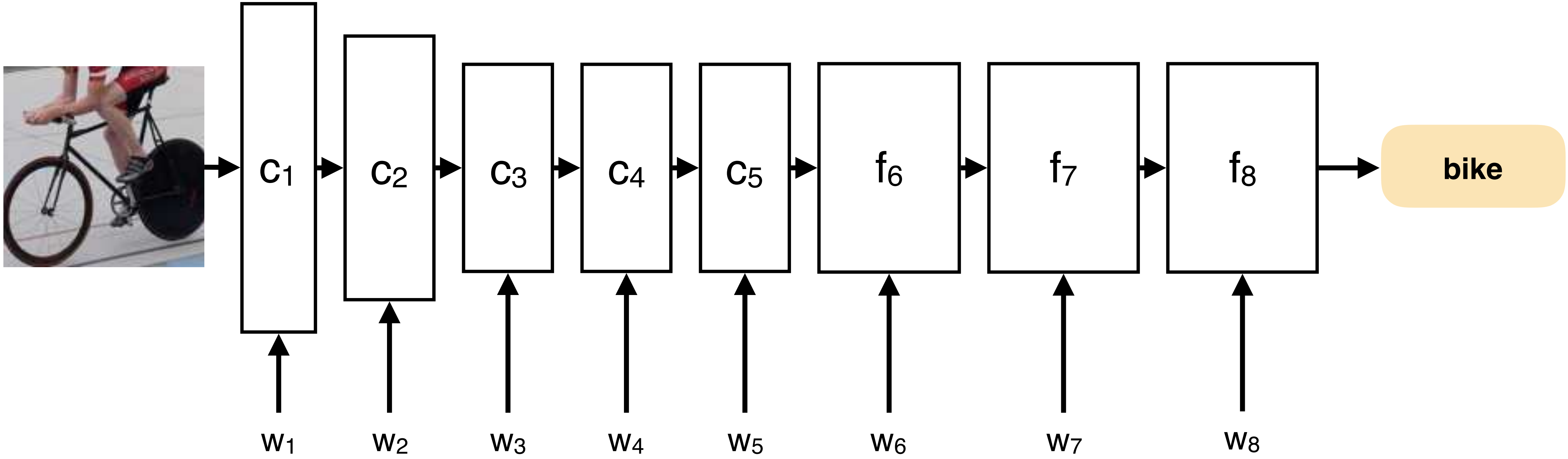
- Different filters (weights) reveal a different characteristics of the input.

$*$

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

Convolutional Neural Networks in a Nutshell

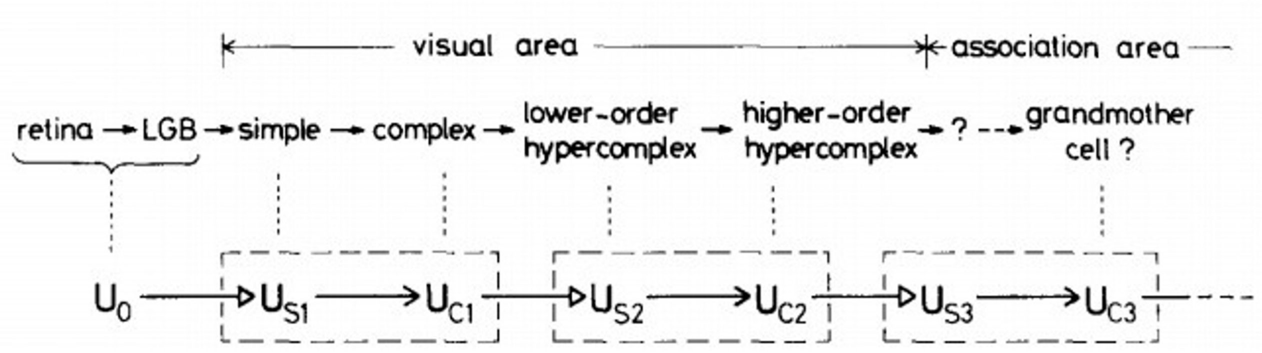
- A neural network model that consists of a sequence of local & translation invariant layers
  - Many identical copies of the same neuron: Weight/parameter sharing
  - Hierarchical feature learning

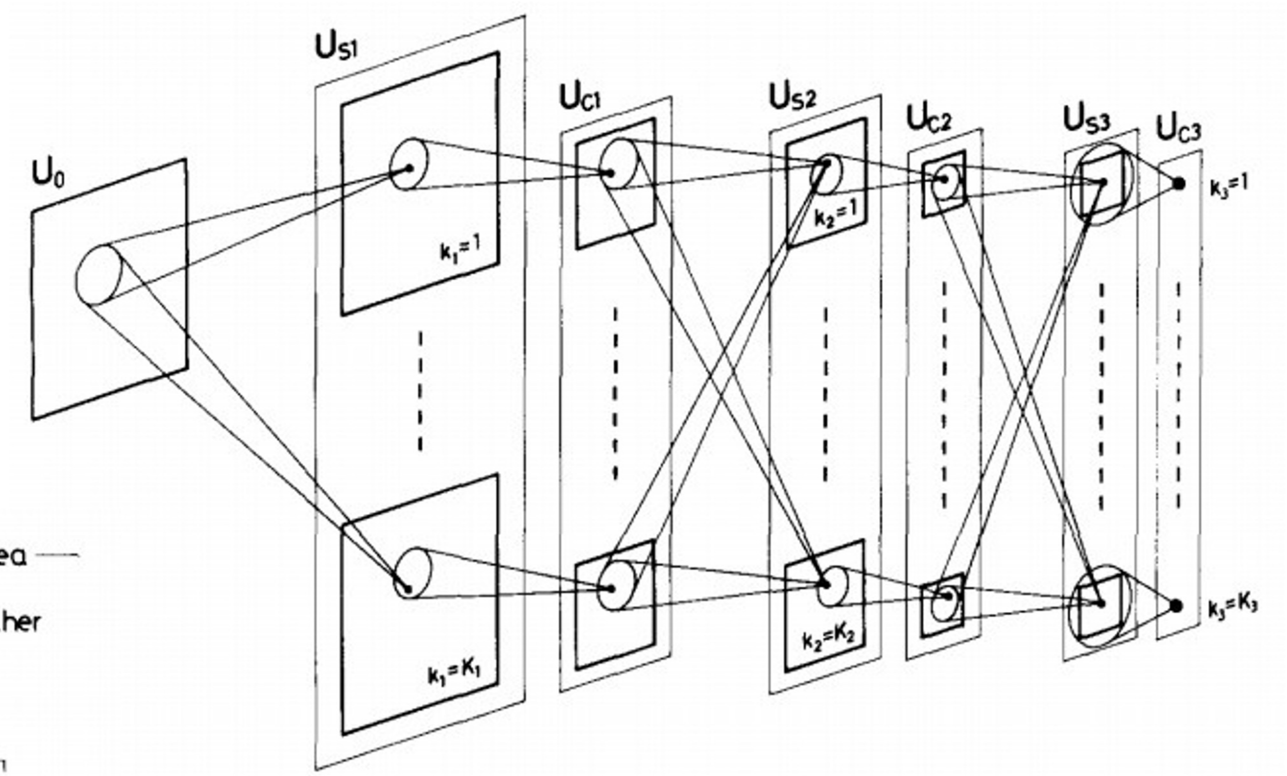


A. Krizhevsky, I. Sutskever, and G. E. Hinton. **Imagenet classification with deep convolutional neural networks**. In NIPS 2012.

A bit of history

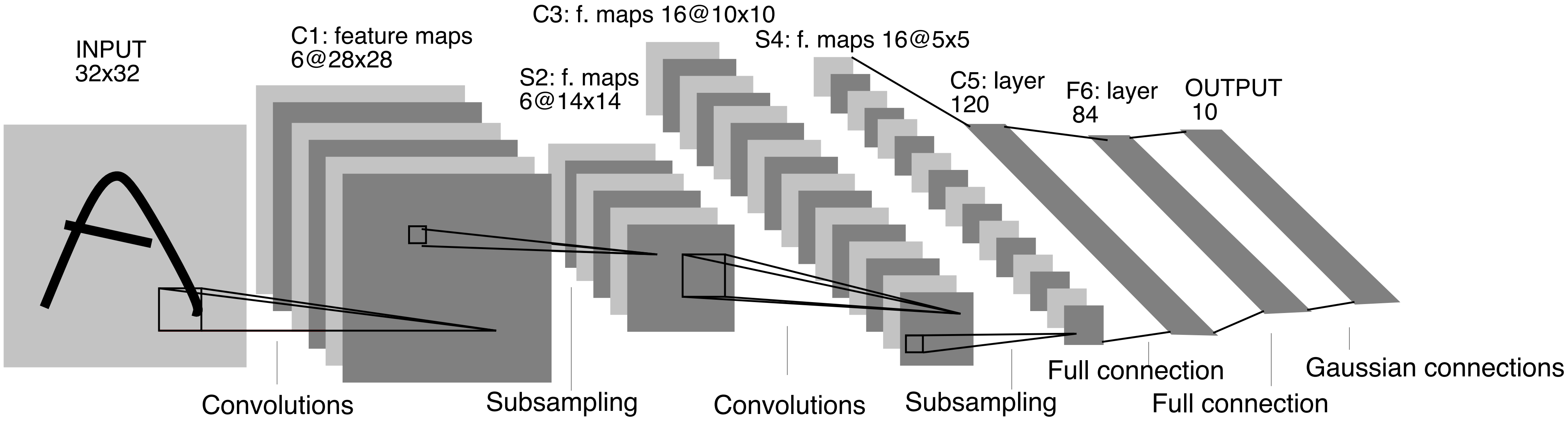
- Neocognitron model by Fukushima (1980)
- The first convolutional neural network (CNN) model
- so-called “sandwich” architecture
  - simple cells act like filters
  - complex cells perform pooling
- Difficult to train
  - No backpropagation yet





A bit of history

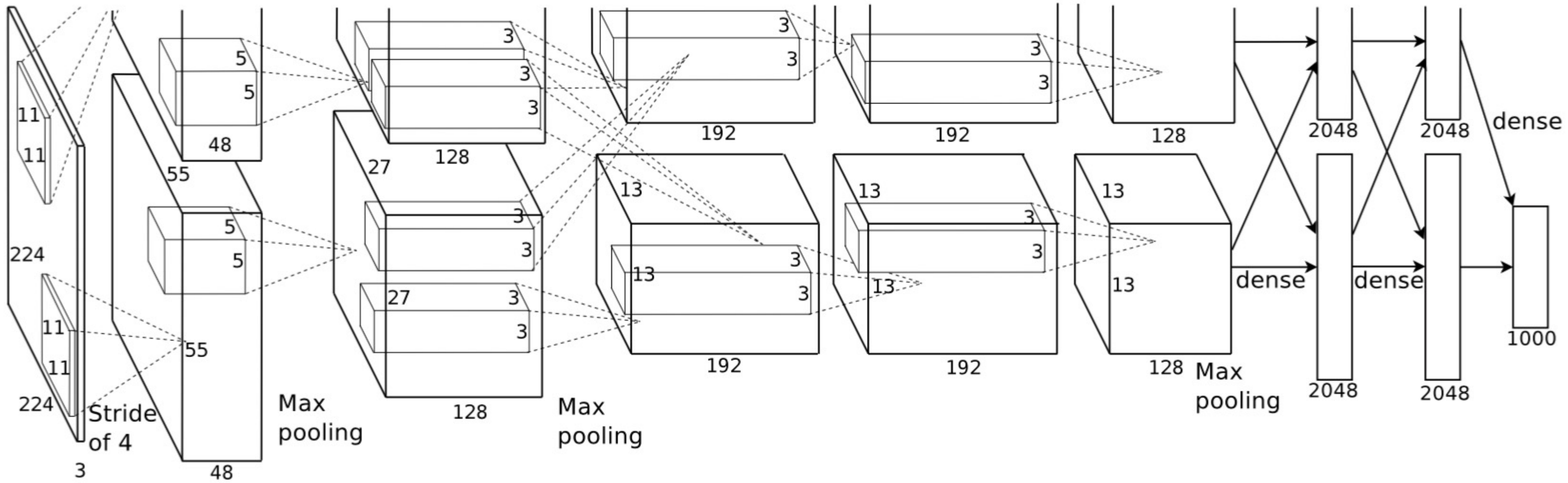
- LeNet-5 model



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.

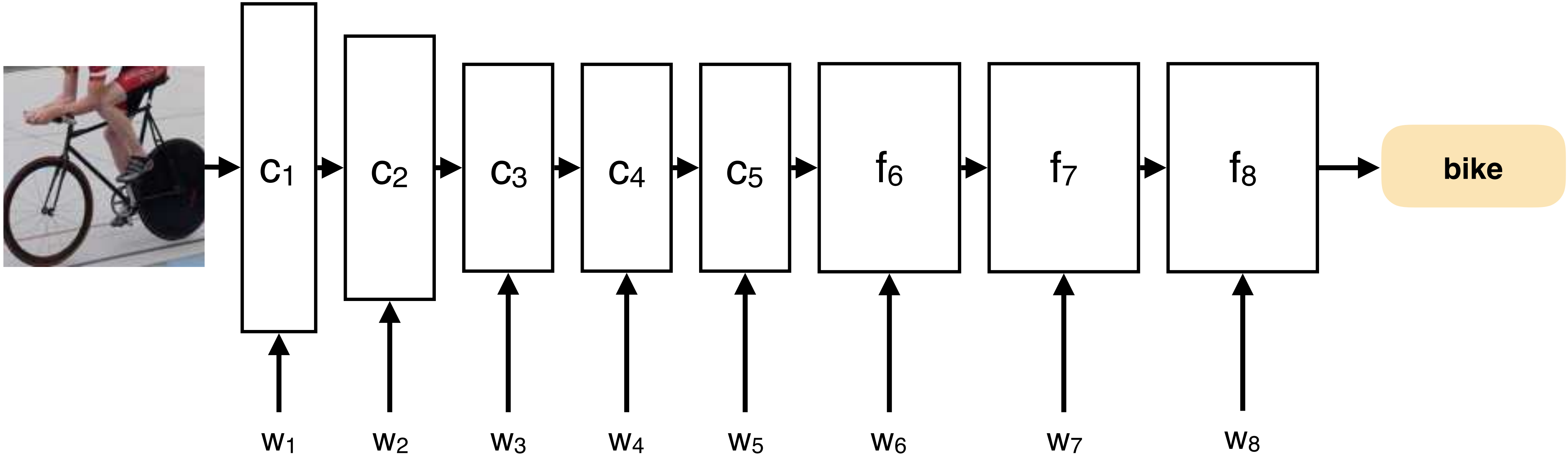
A bit of history

- AlexNet model



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

Convolutional Neural Network

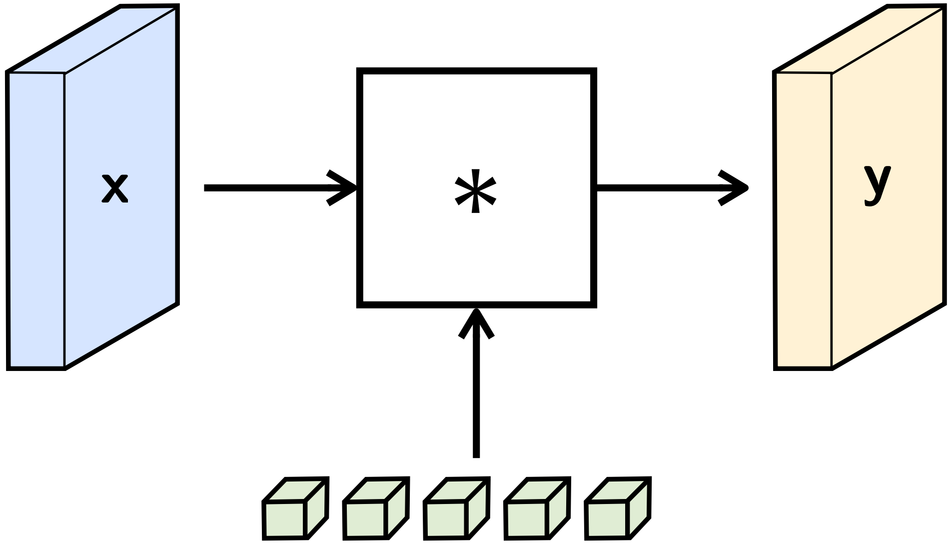


A. Krizhevsky, I. Sutskever, and G. E. Hinton. **Imagenet classification with deep convolutional neural networks**. In NIPS 2012.

Convolutional layer

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

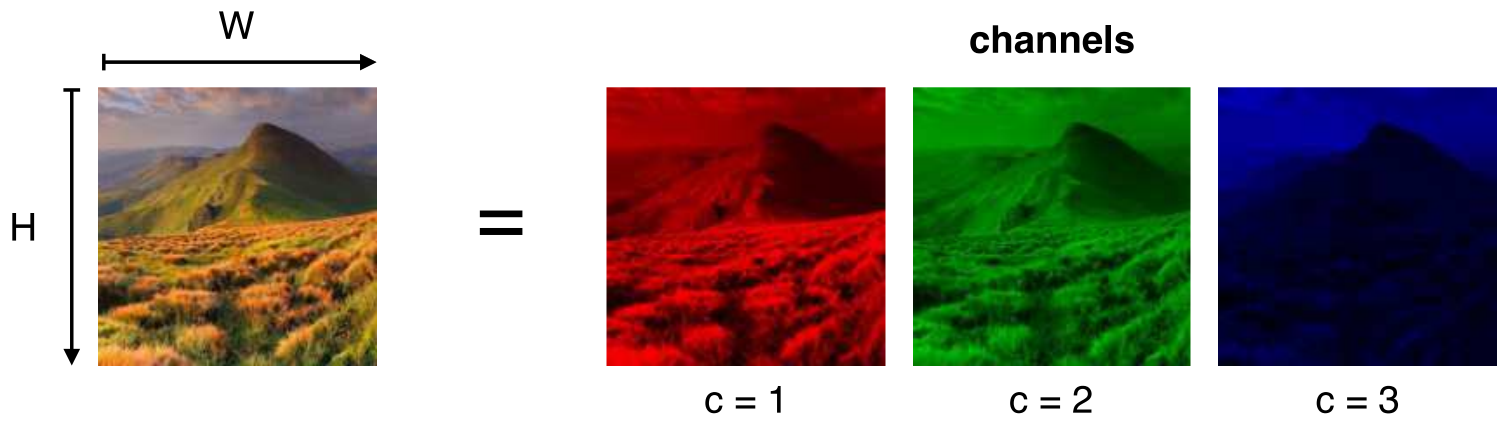
$$\mathbf{y} = F * \mathbf{x} + b$$

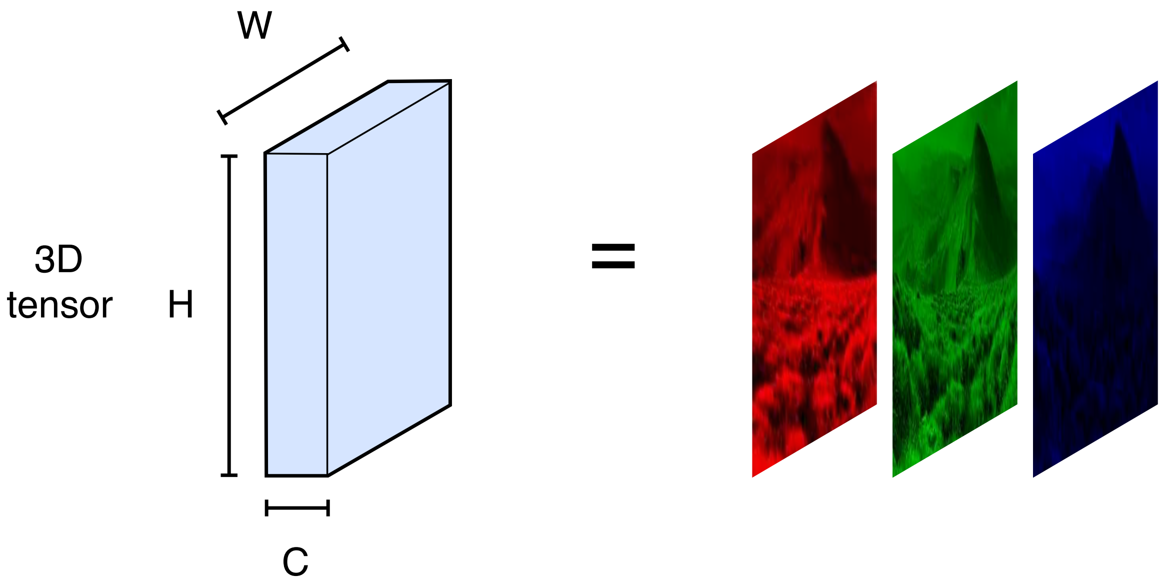


input data $\mathbf{x}$ filter bank F output data $\mathbf{y}$

Data = 3D Tensors

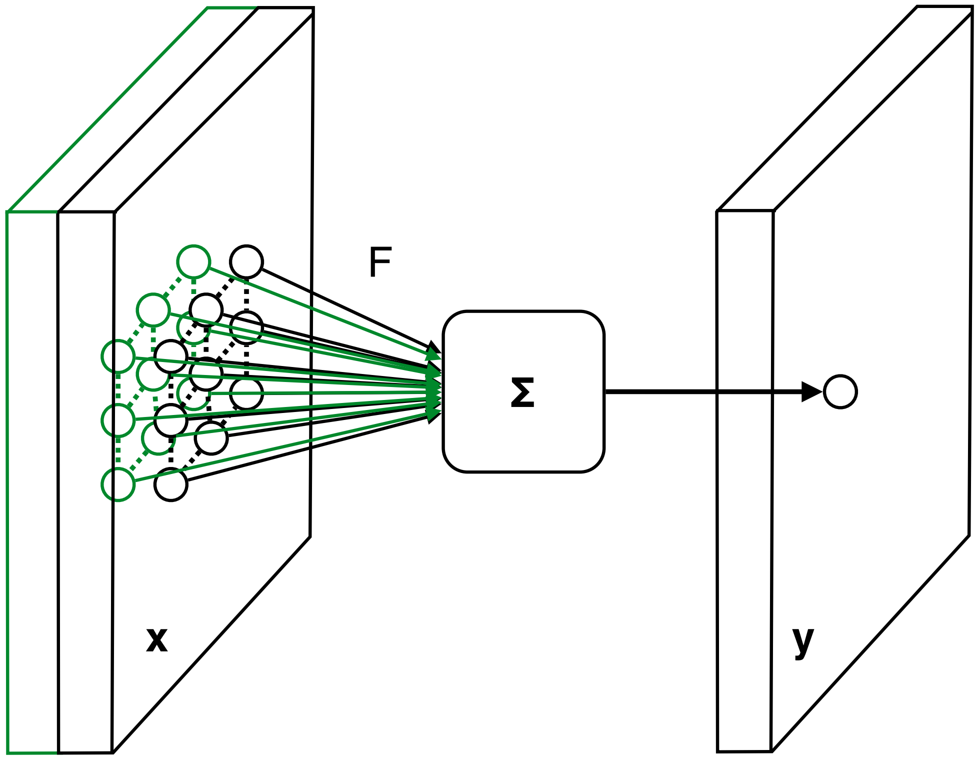
- There is a vector of feature channels (e.g. RGB) at each spatial location (pixel).

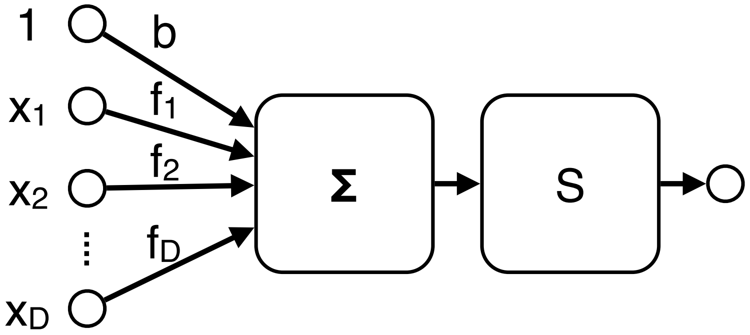


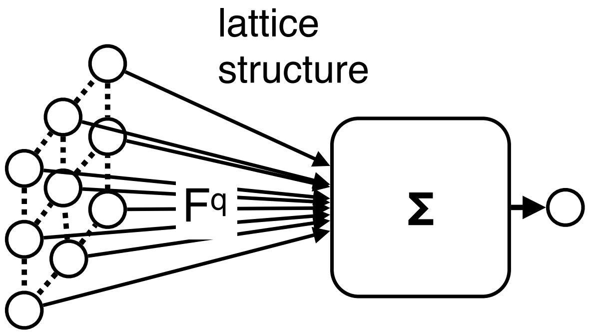


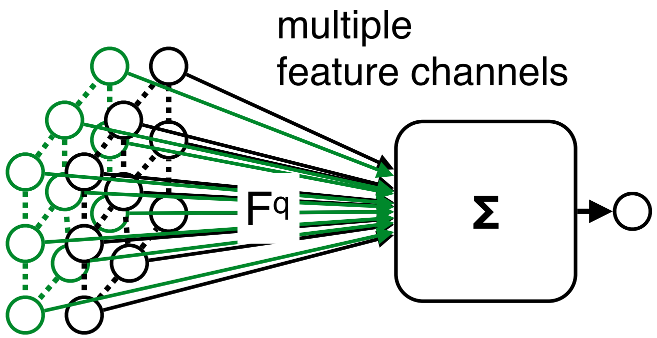
Convolutions with 3D Filters

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



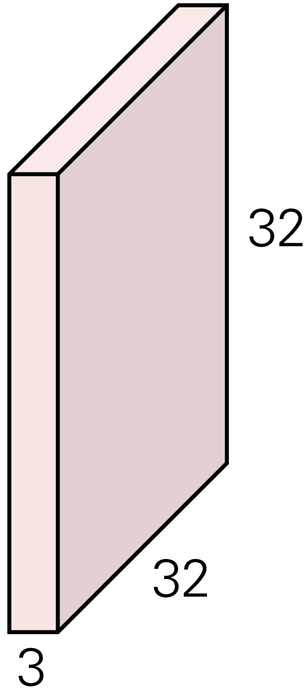




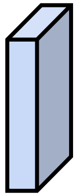


Convolutional Layer

32x32x3 input

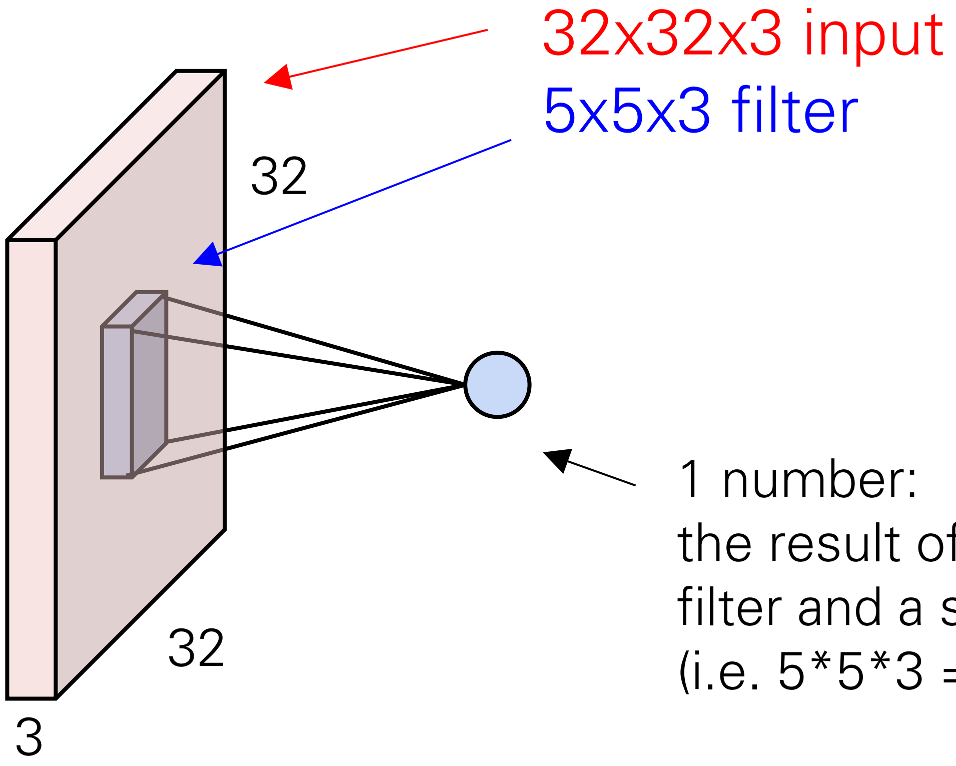


5x5x3 filter



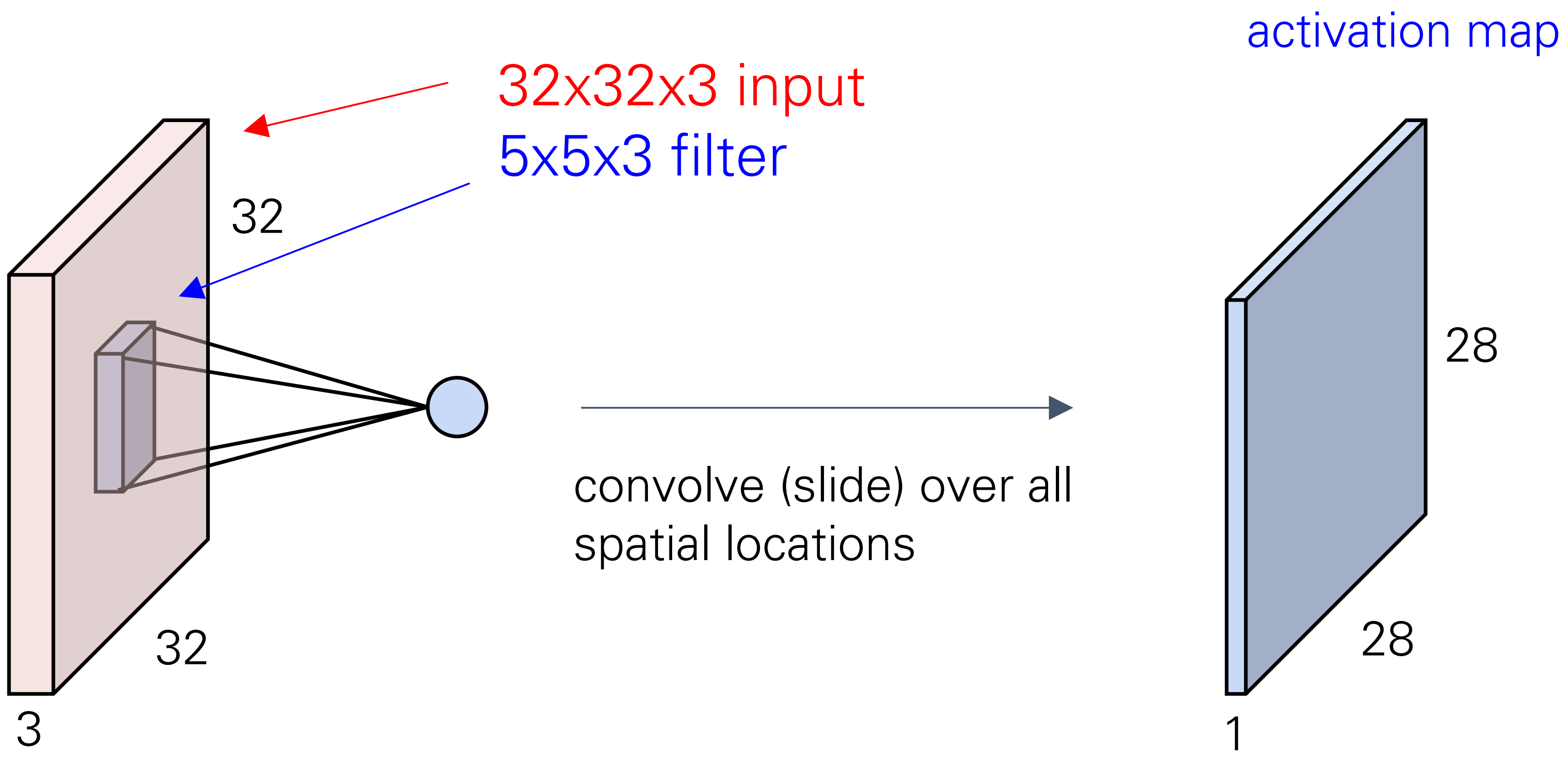
Convolve the filter with the input
i.e. “slide over the image spatially,
computing dot products”

Convolutional Layer



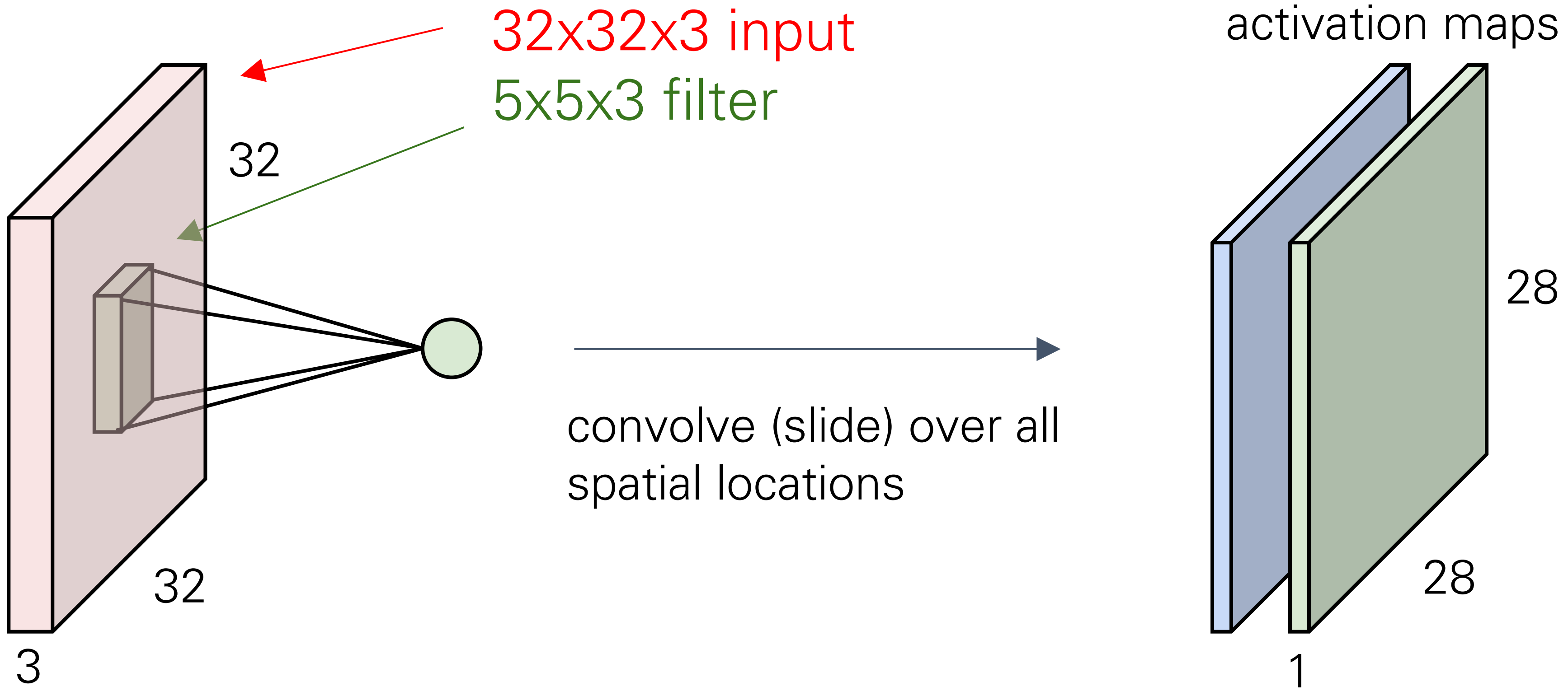
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the input
(i.e. 5*5*3 = 75-dimensional dot product + bias)

Convolutional Layer



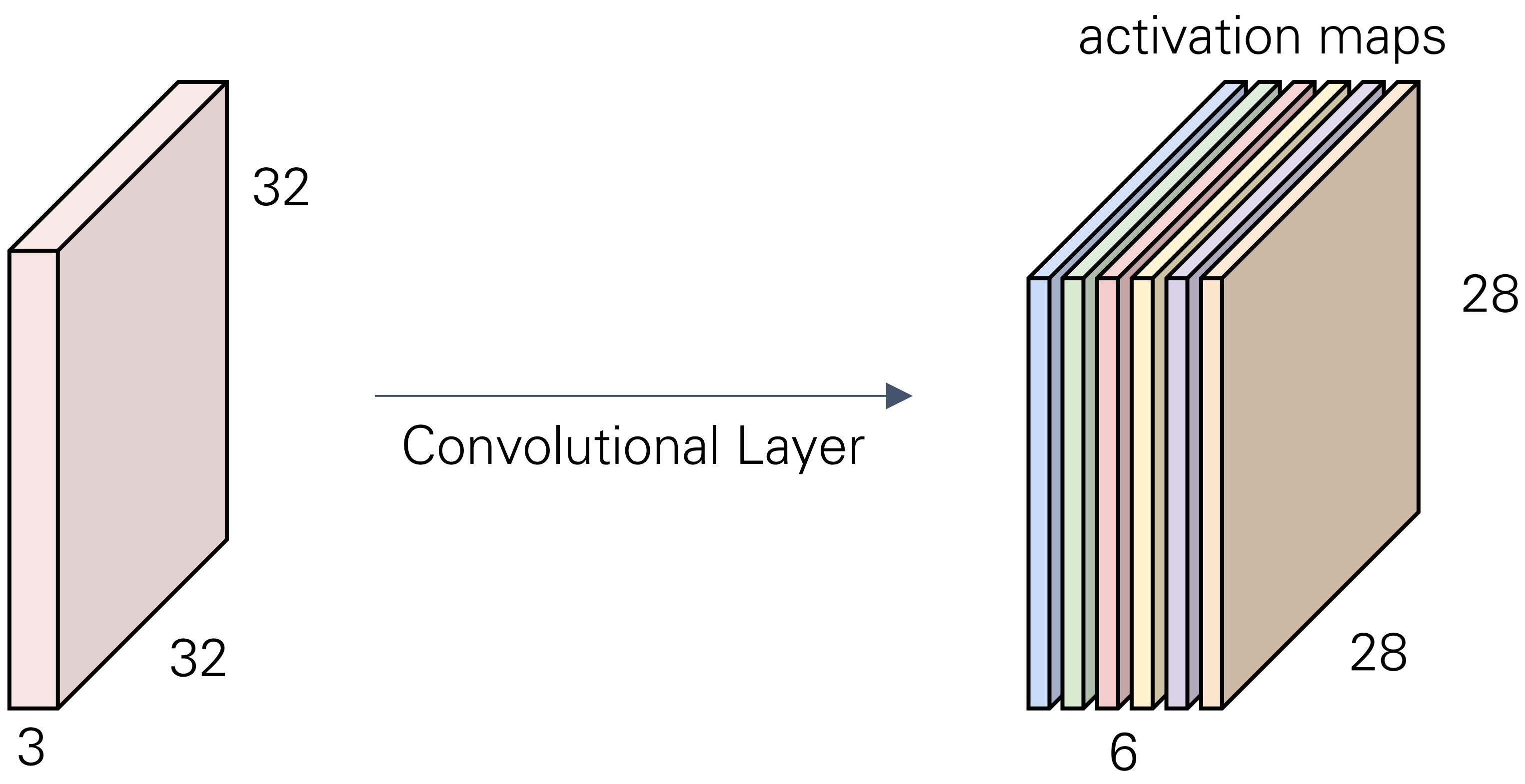
Convolutional Layer

consider a second, green filter



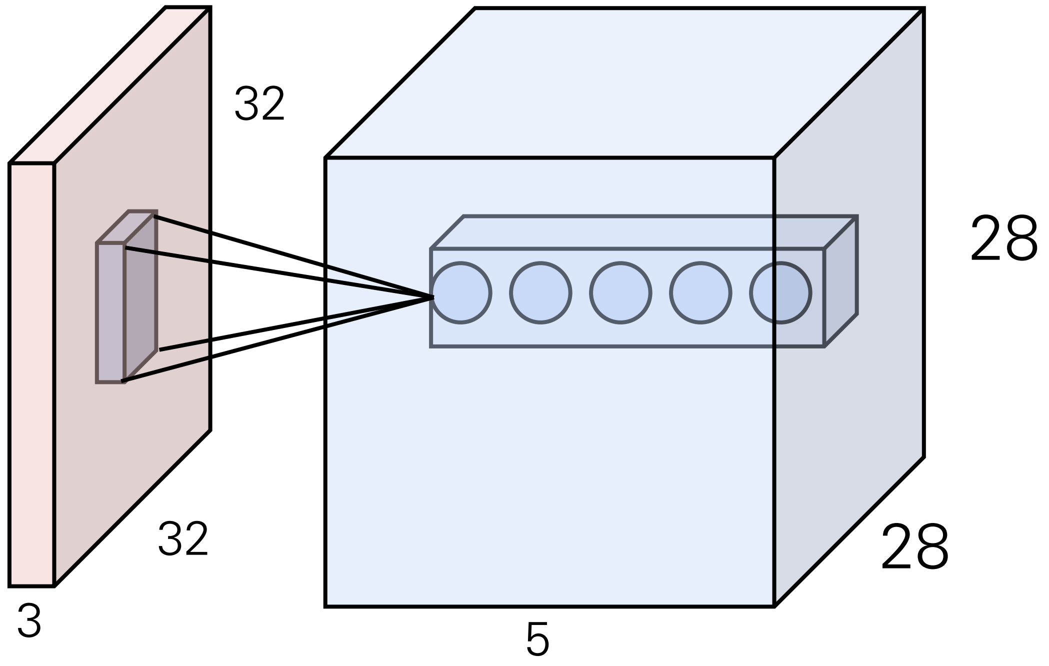
Convolutional Layer

- Multiple filters produce multiple output channels
- For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

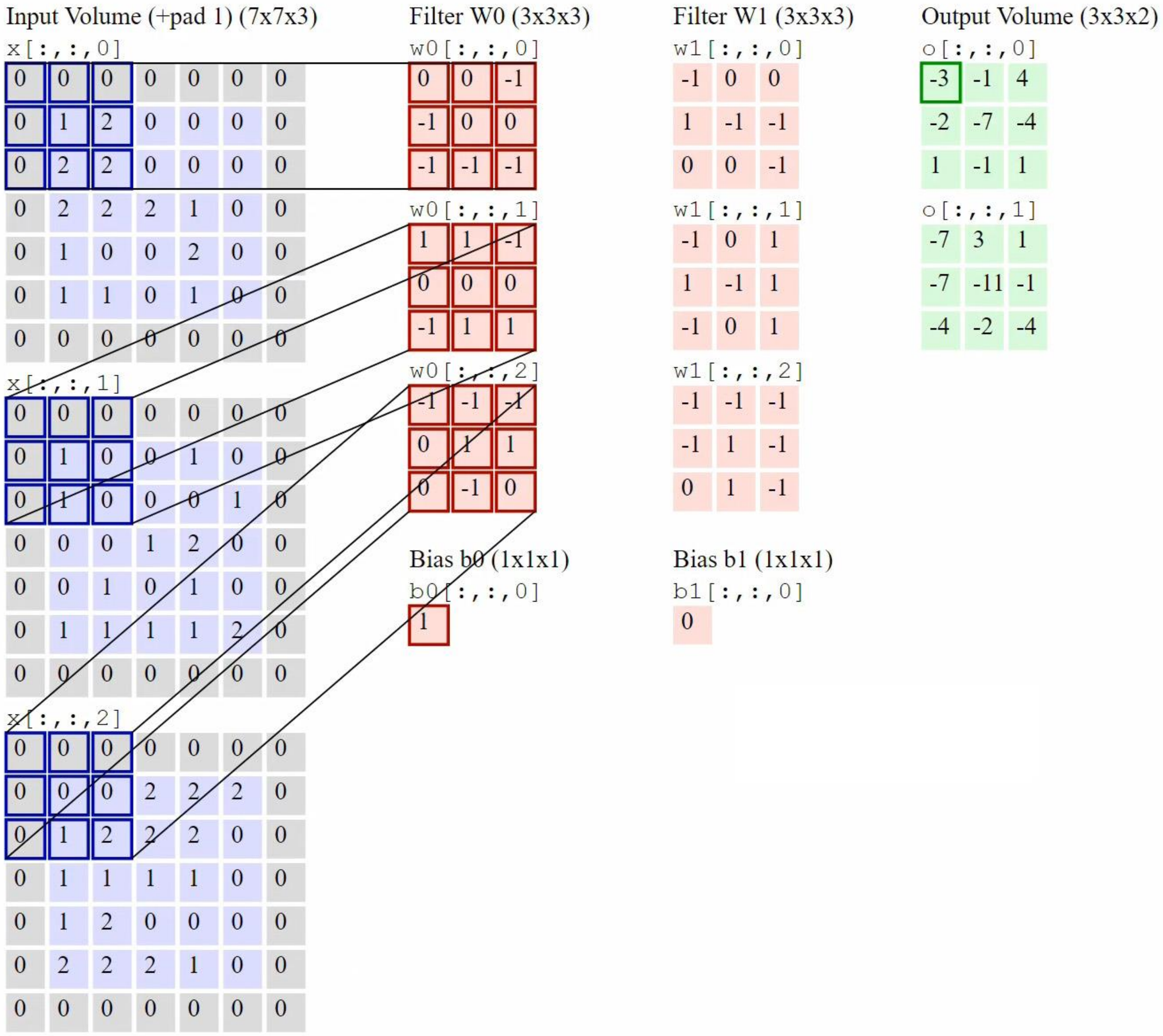


We stack these up to get an output of size 28x28x6.

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

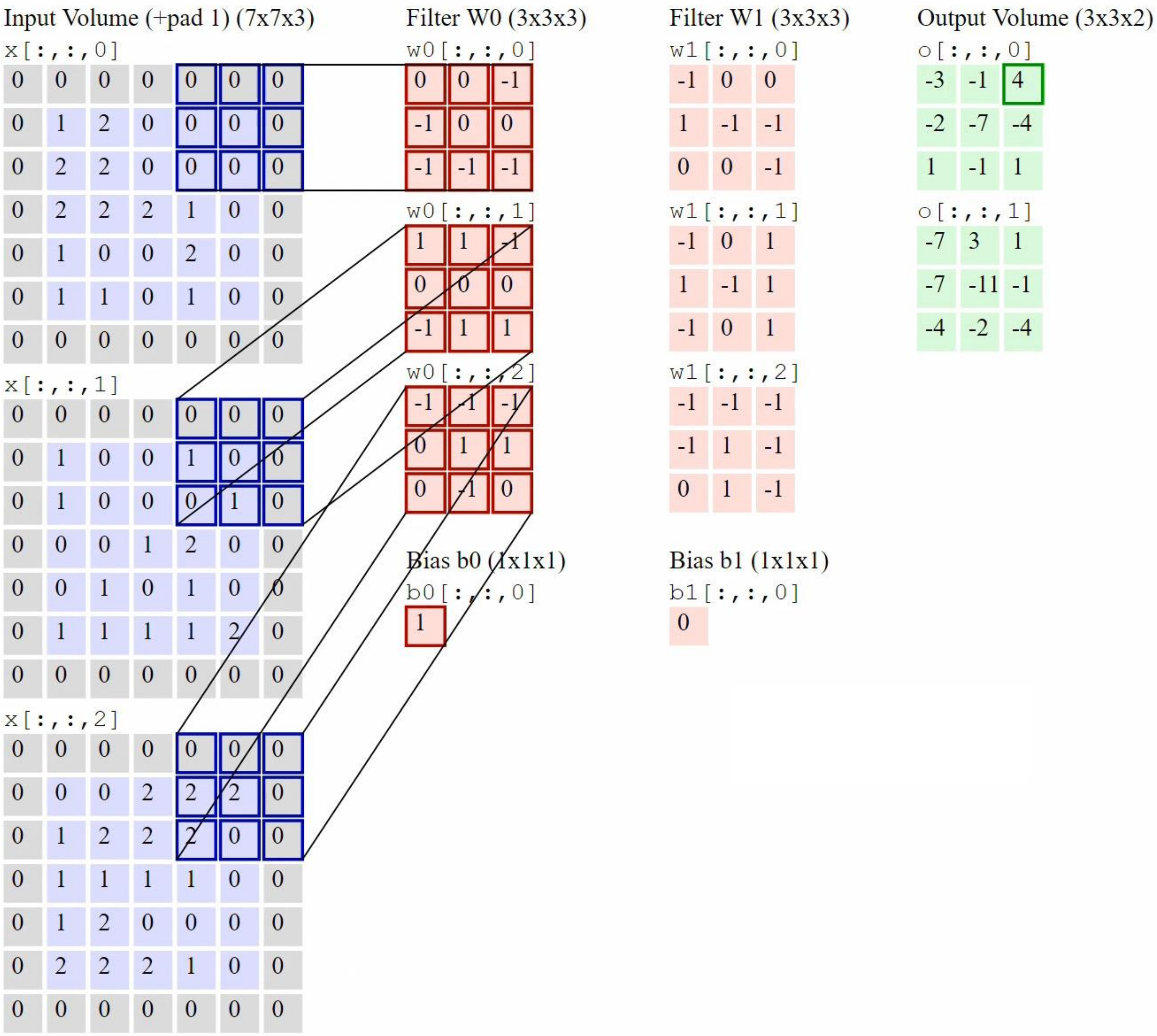
Output Volume (3x3x2)

o[:,:,0]

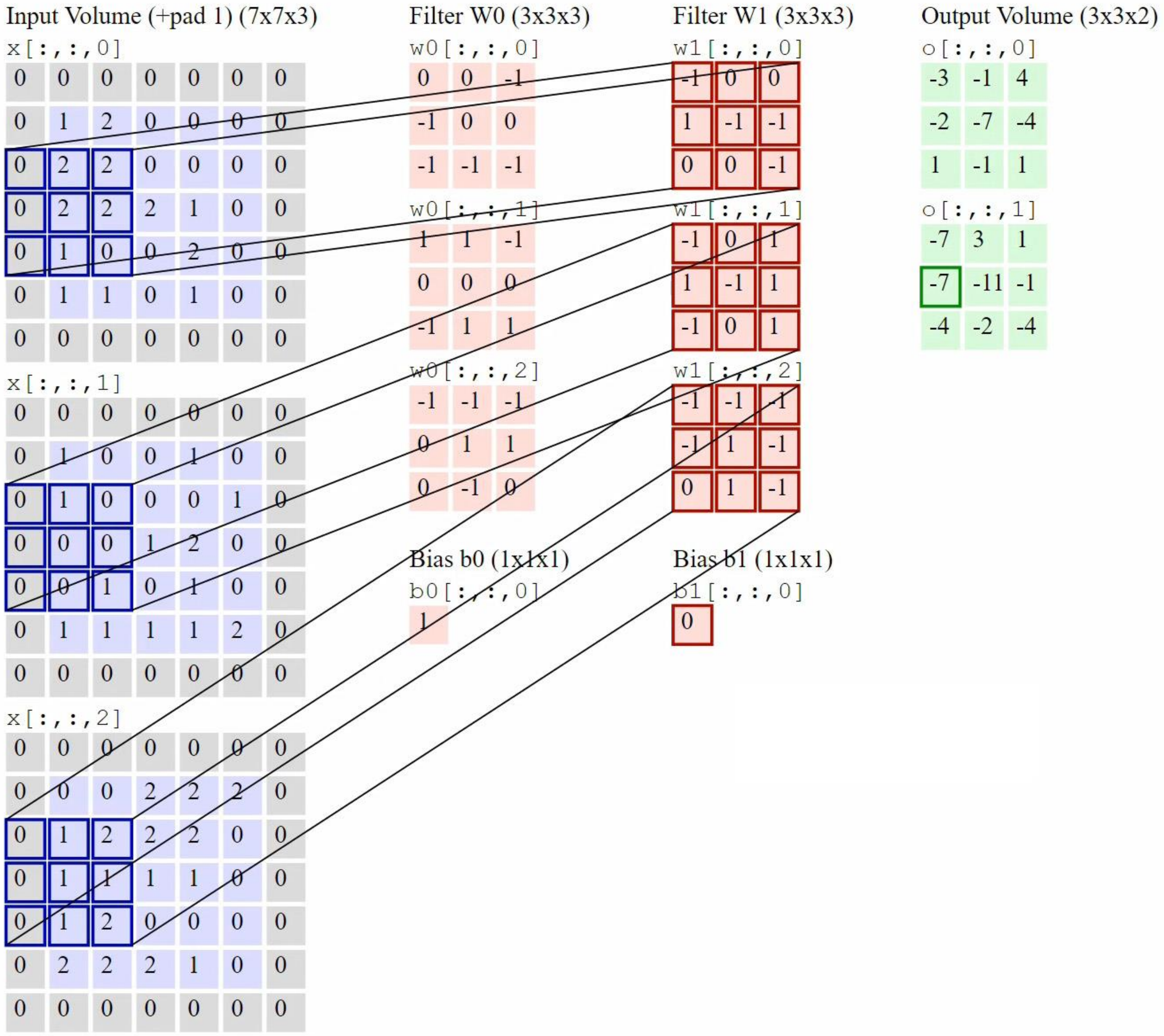
-3	-1	4
-2	-7	-4
1	-1	1

o[:,:,1]

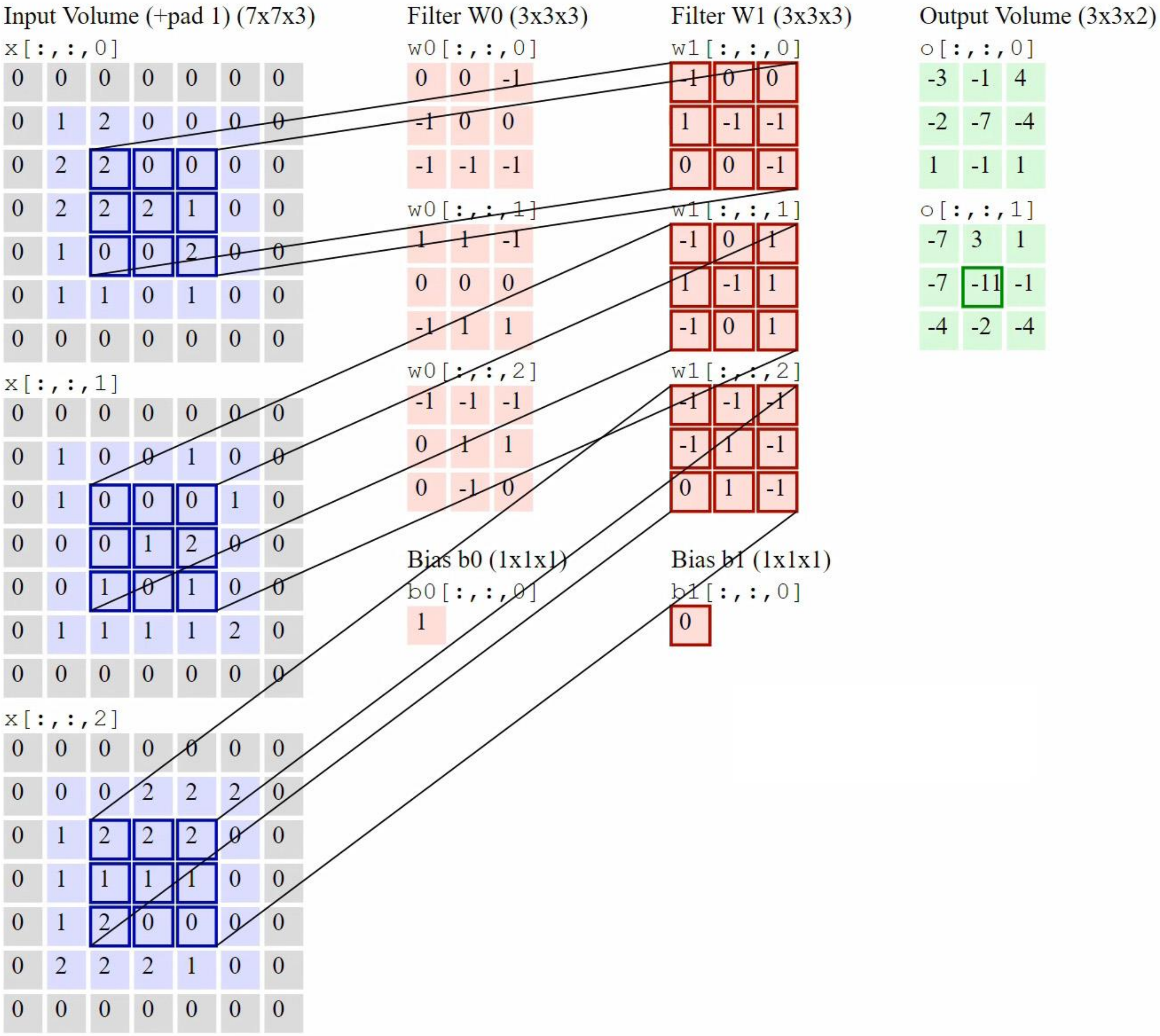
-7	3	1
-7	-11	-1
-4	-2	-4

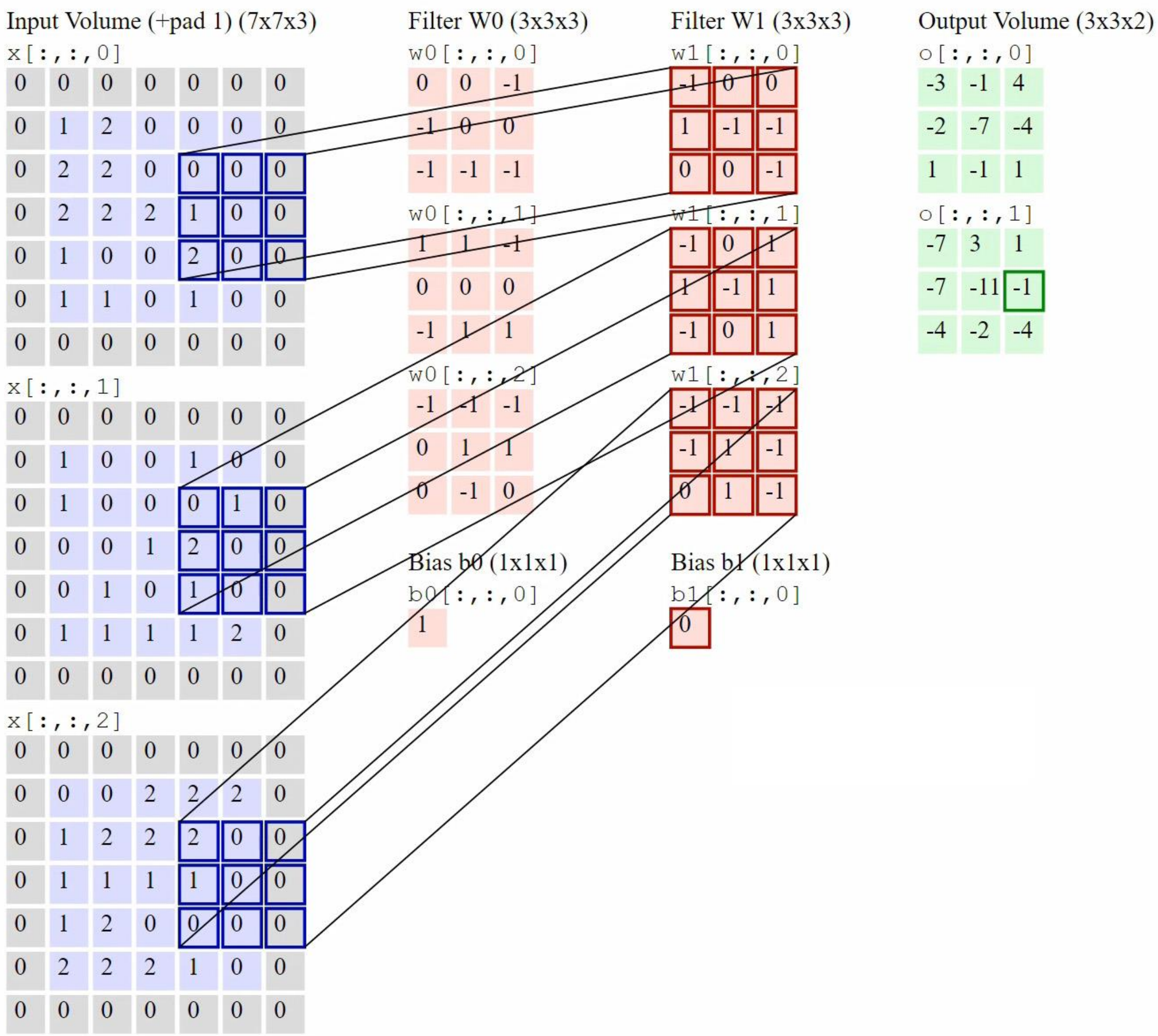


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 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

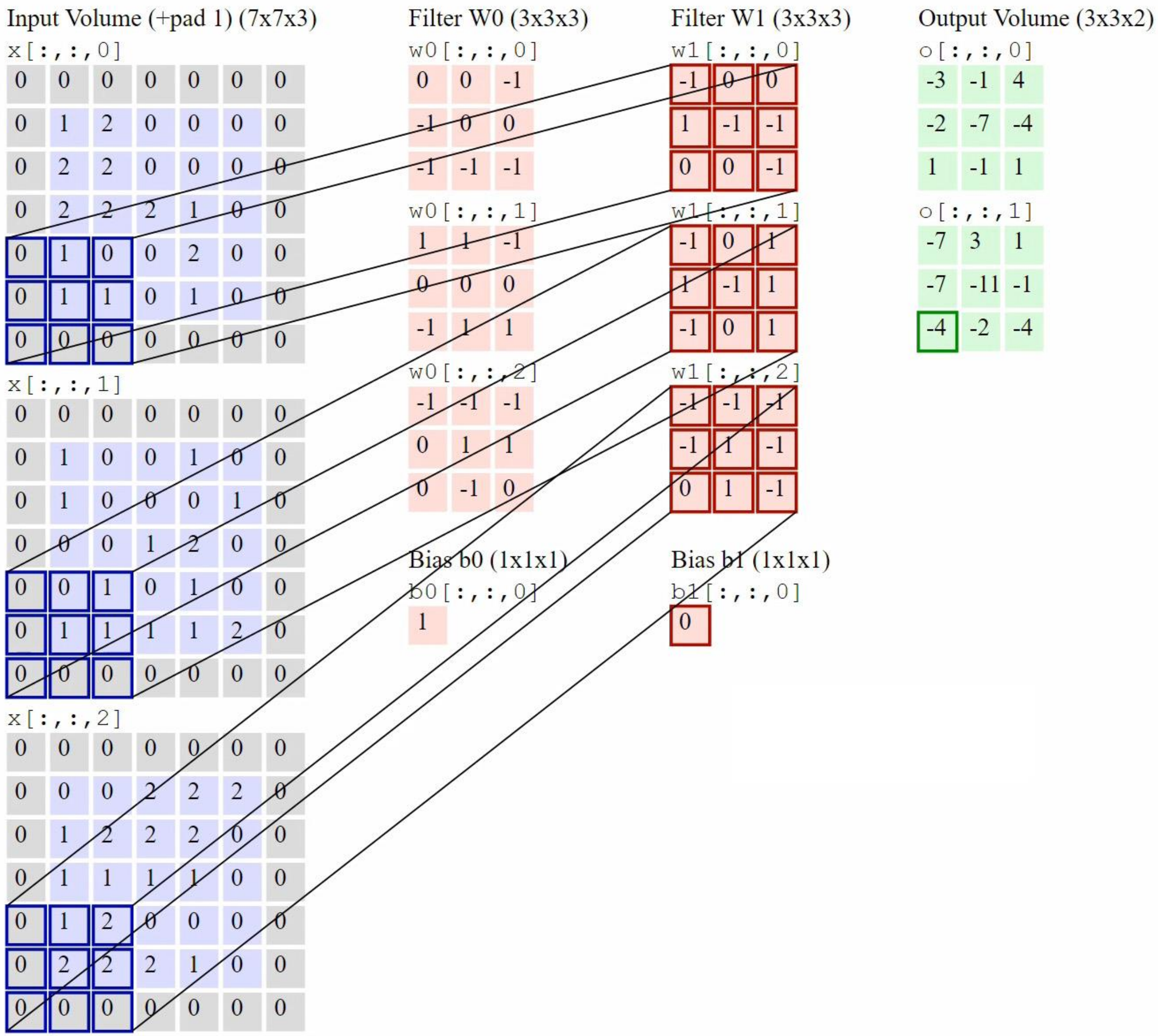


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 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

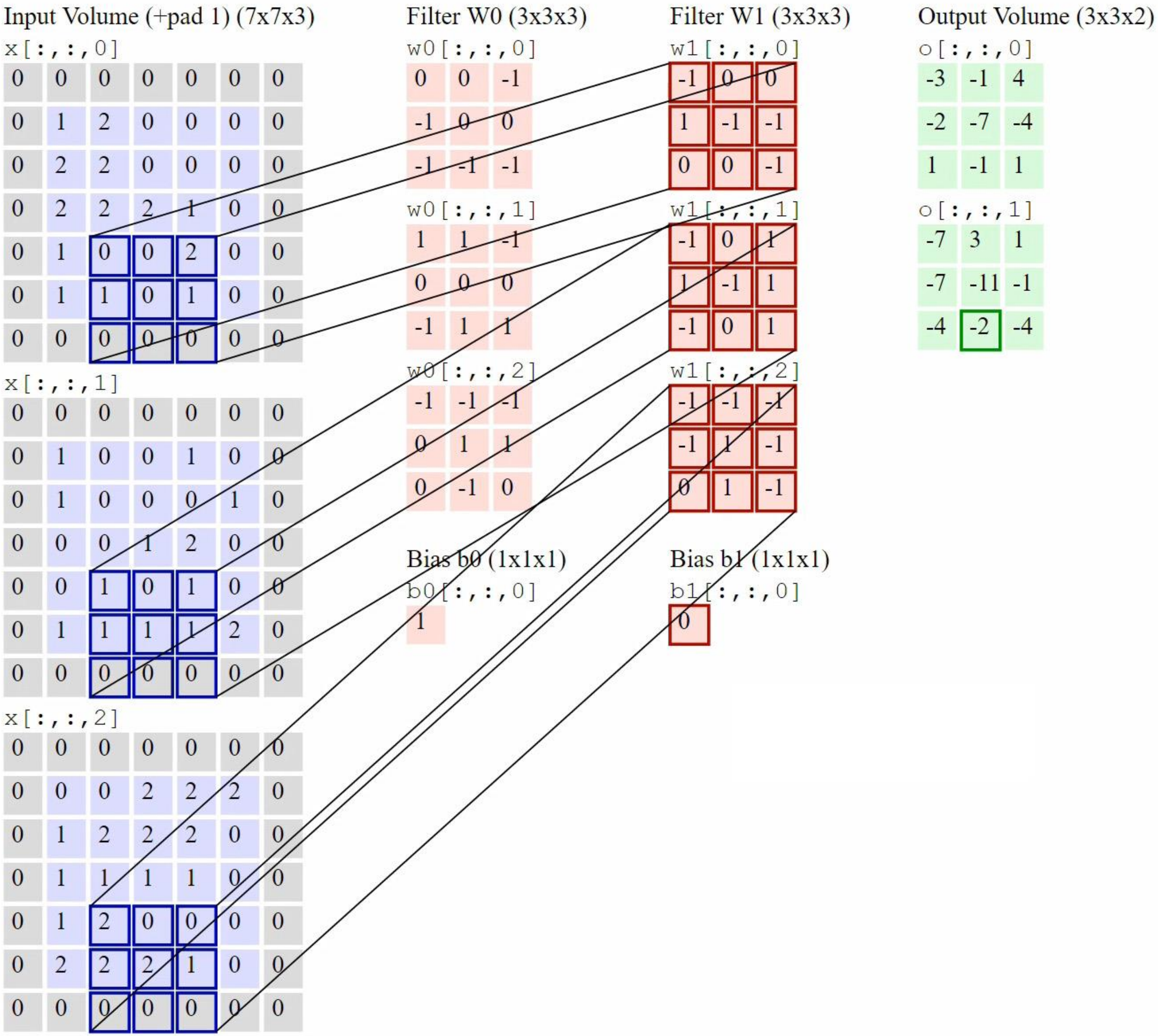

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 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

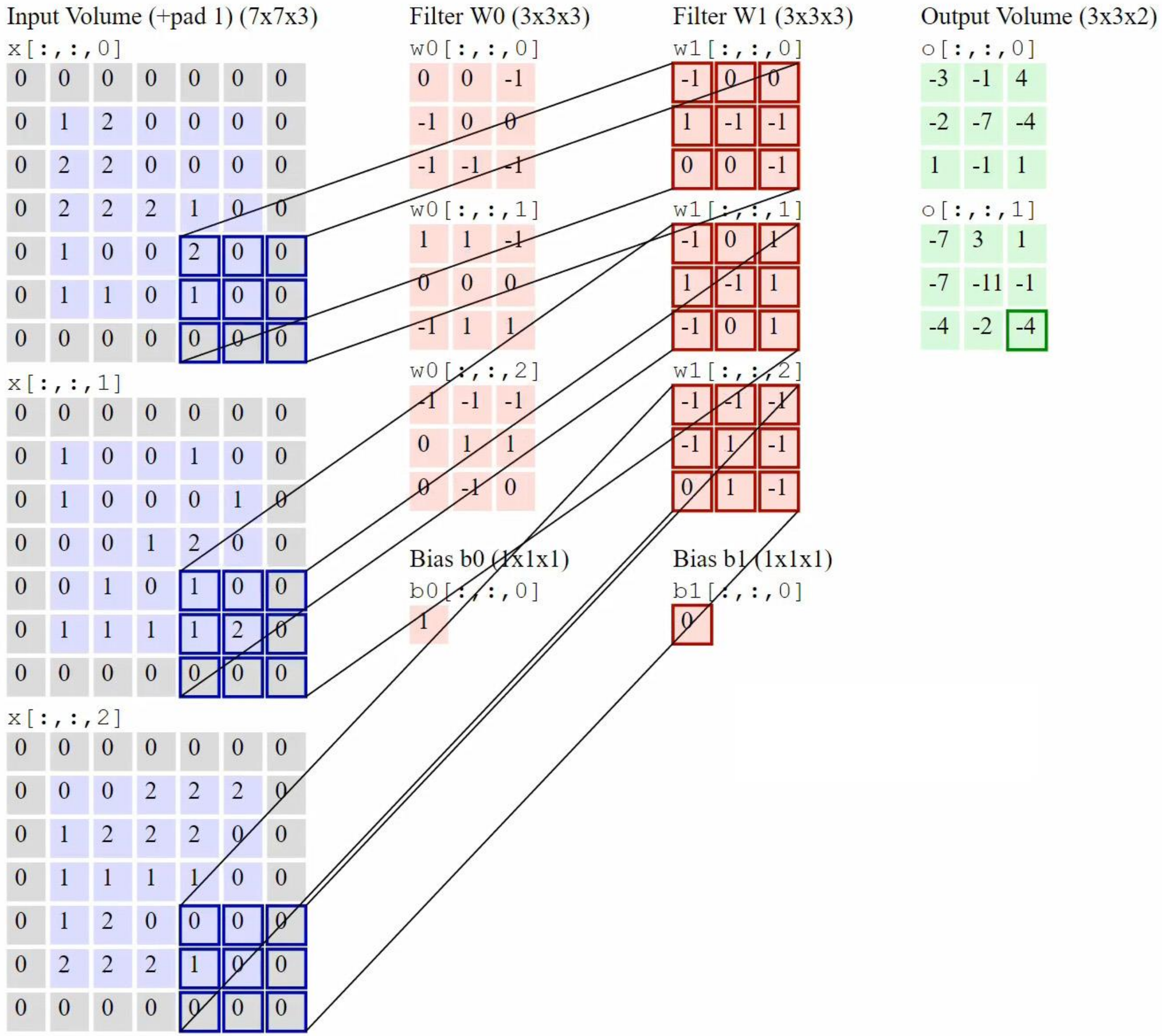


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 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



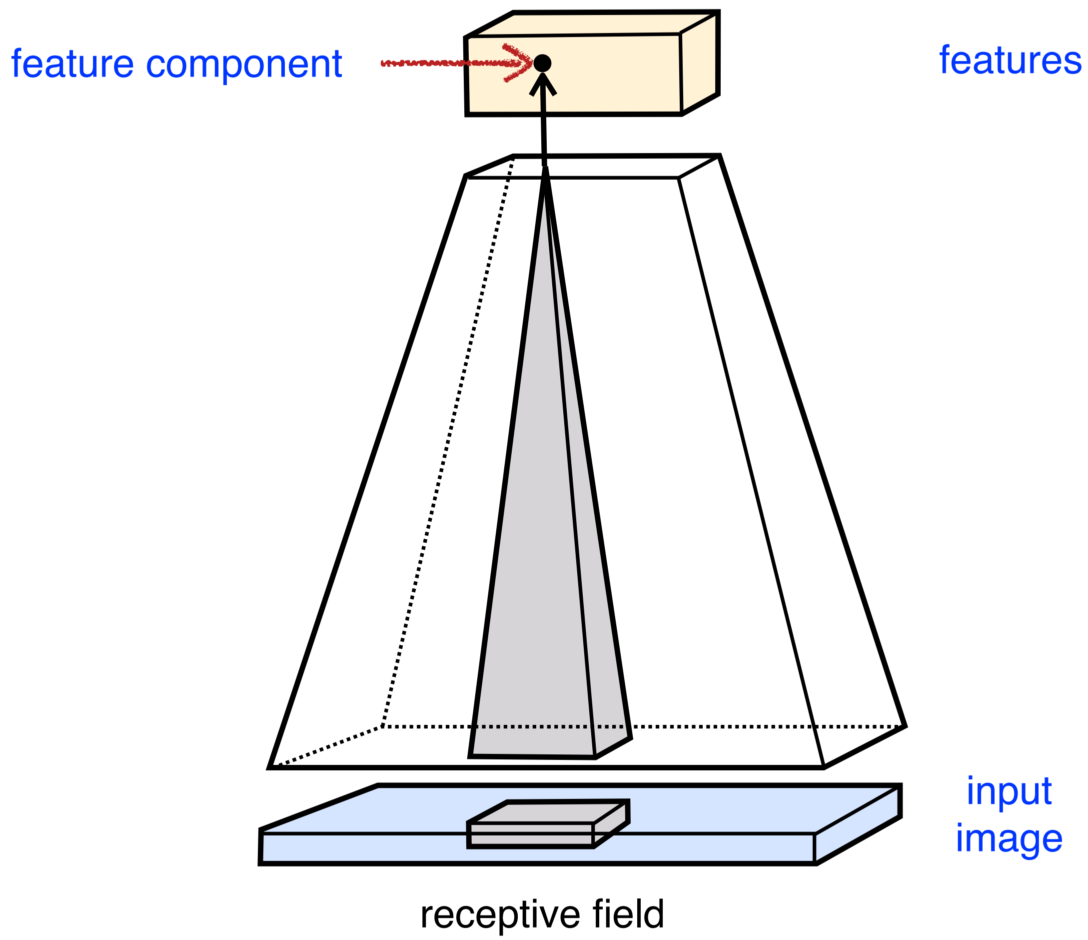
Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 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]
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

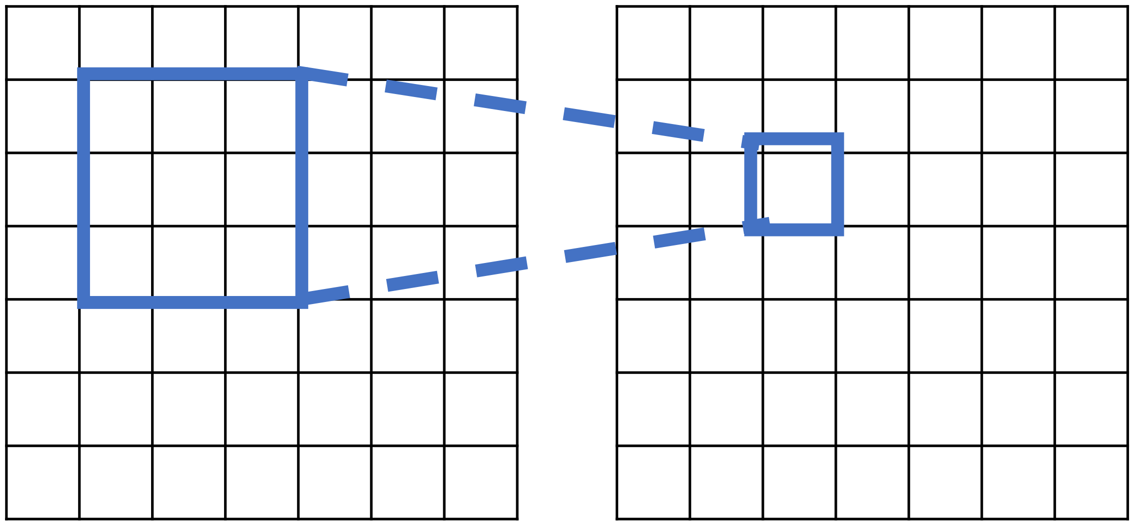
Convolutional layers

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



Receptive Fields

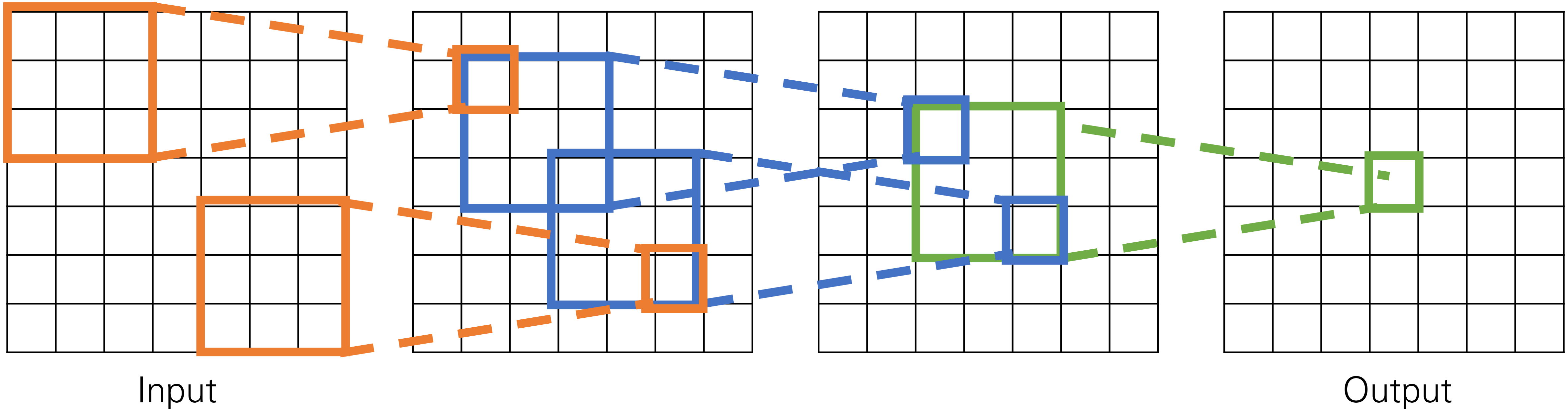
- For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input



Input   Output

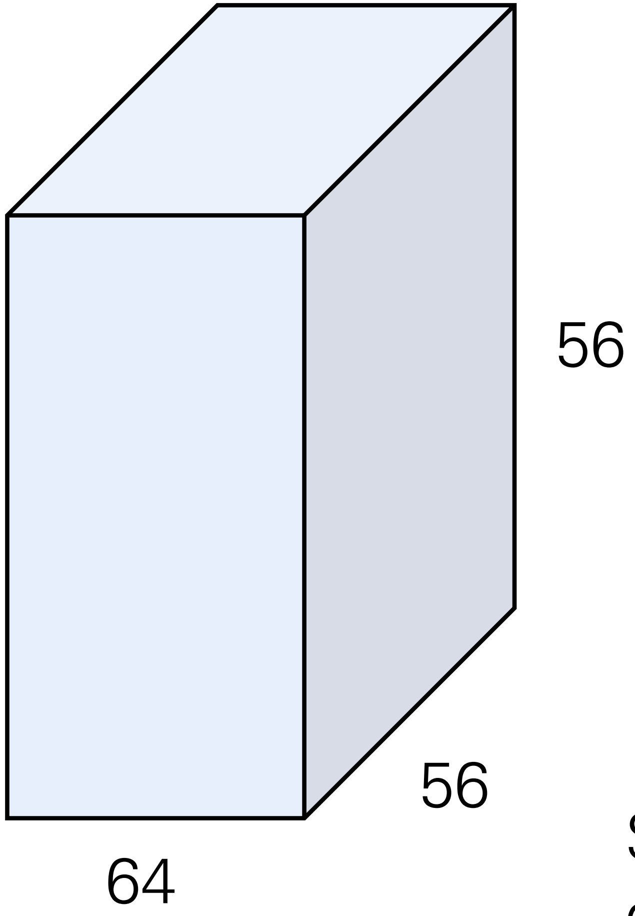
Receptive Fields

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



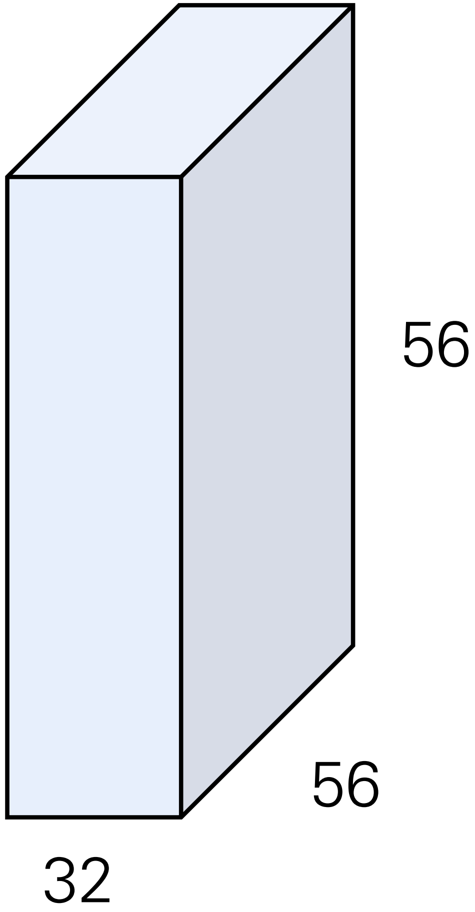
Problem: For large images we need many layers for each output to “see” the whole image image

1x1 Convolution



1x1 CONV
with 32 filters

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



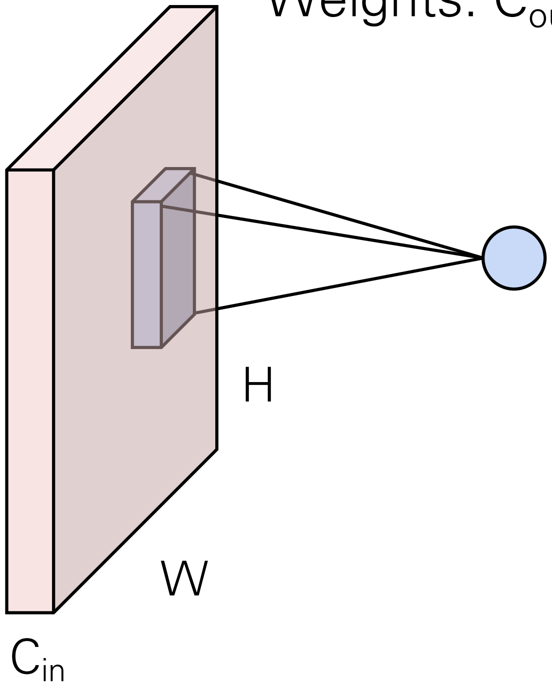
Stacking 1x1 conv layers gives MLP operating on each input position

Lin et al, “Network in Network”, ICLR 2014

Other types of convolution

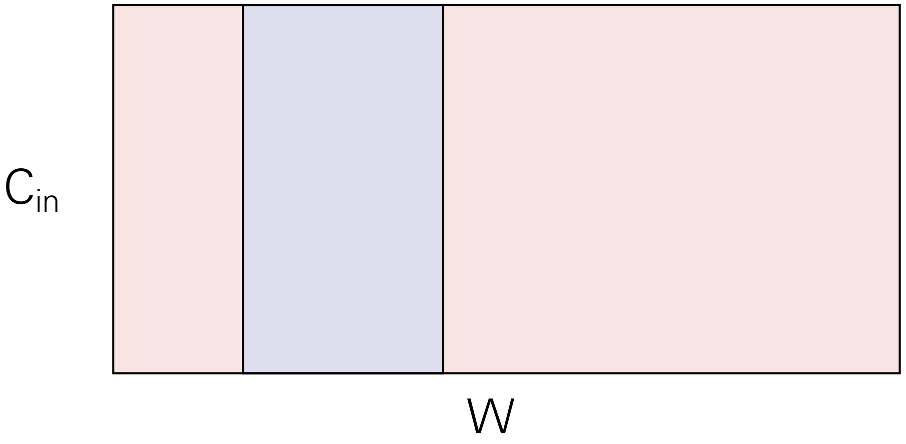
So far: 2D Convolution

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



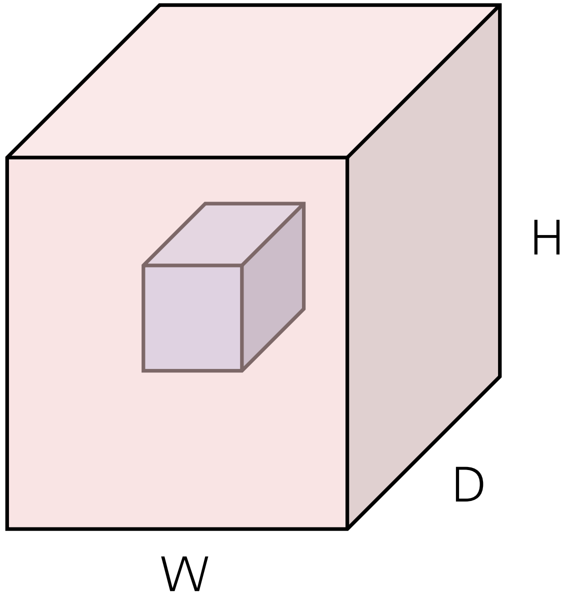
1D Convolution

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



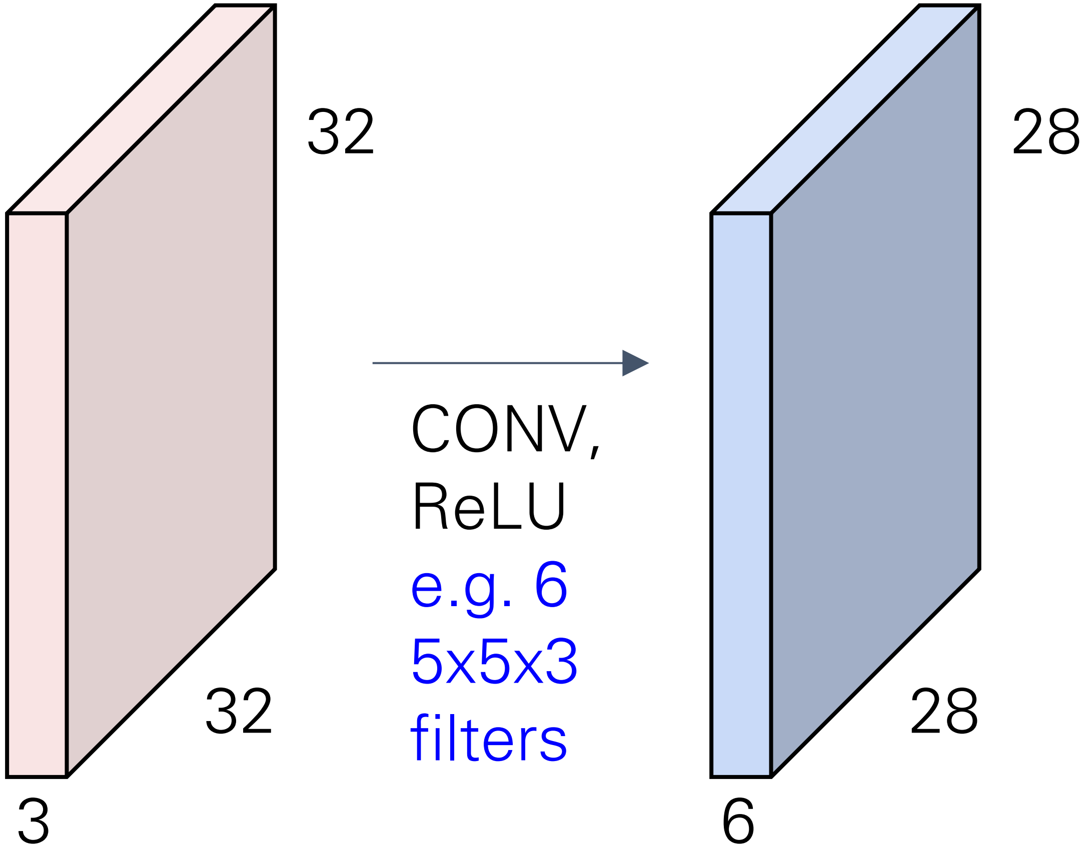
3D Convolution

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

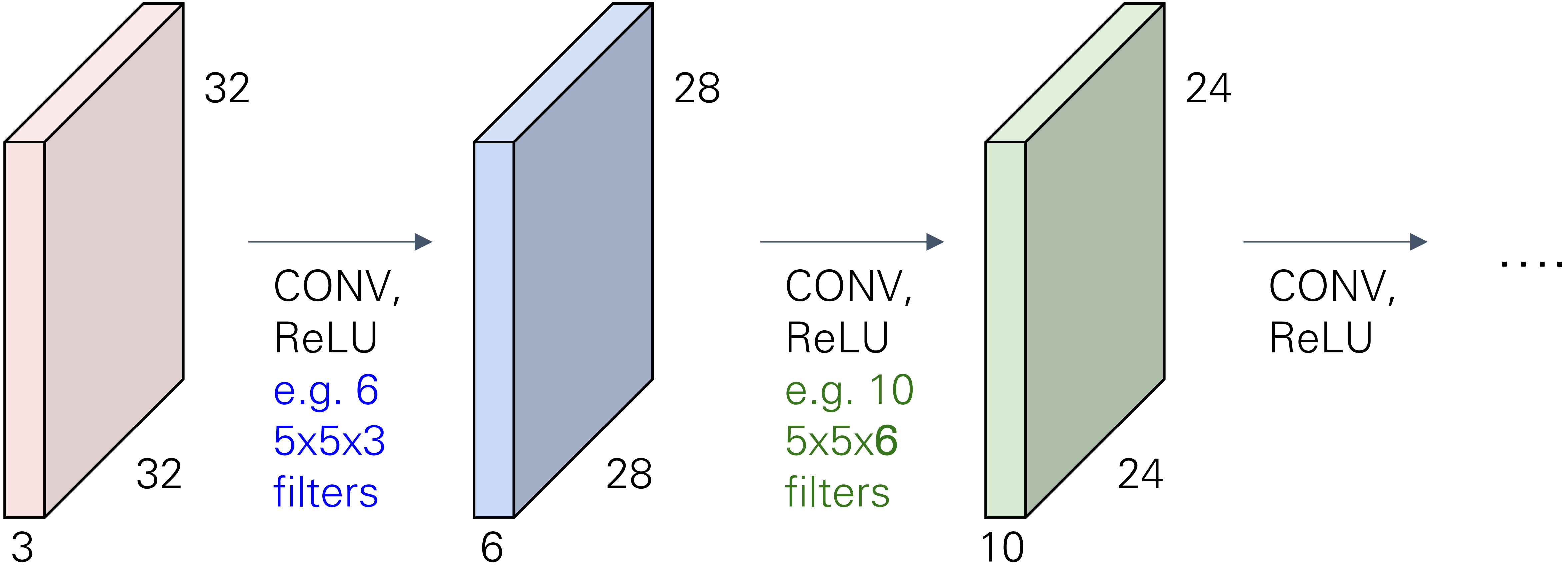


C_{in}-dim vector at each point in the volume

Convolutional layers

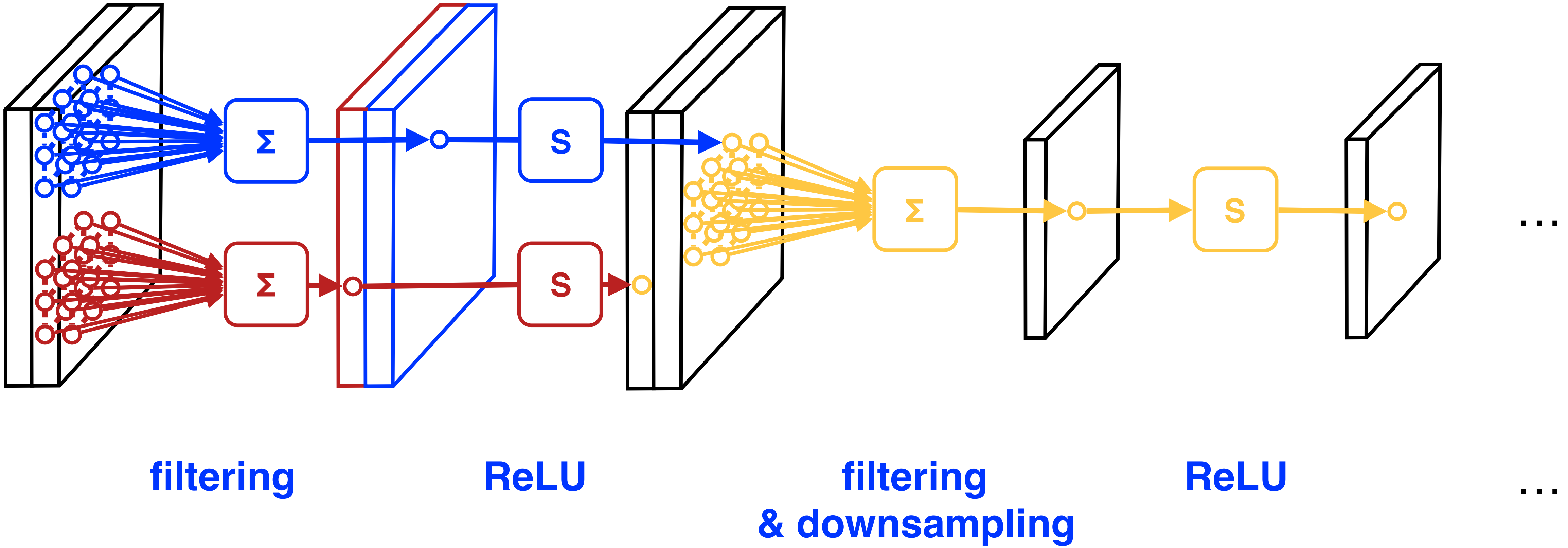


Repeat linear / non-linear operators



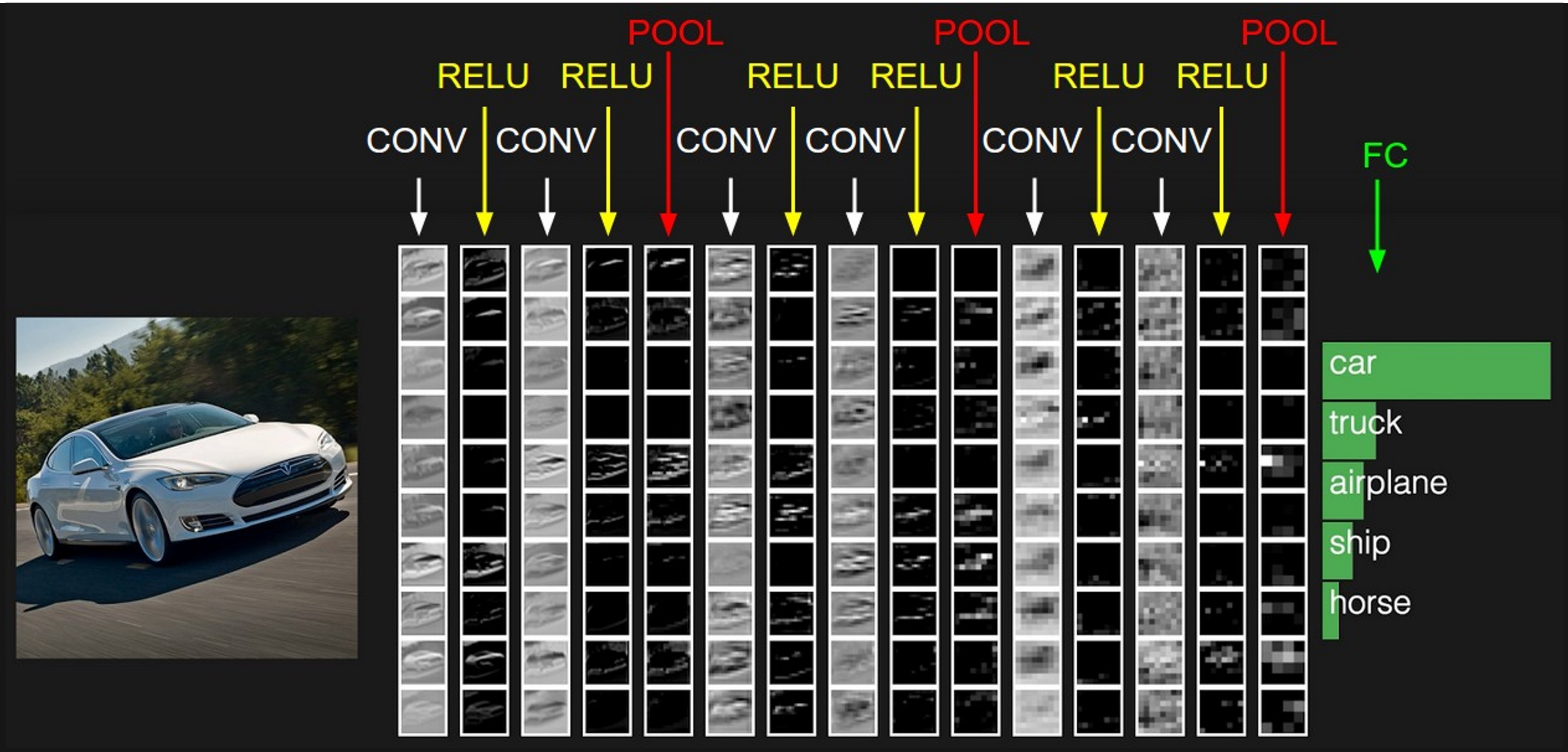
Linear/Non-linear Chains

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



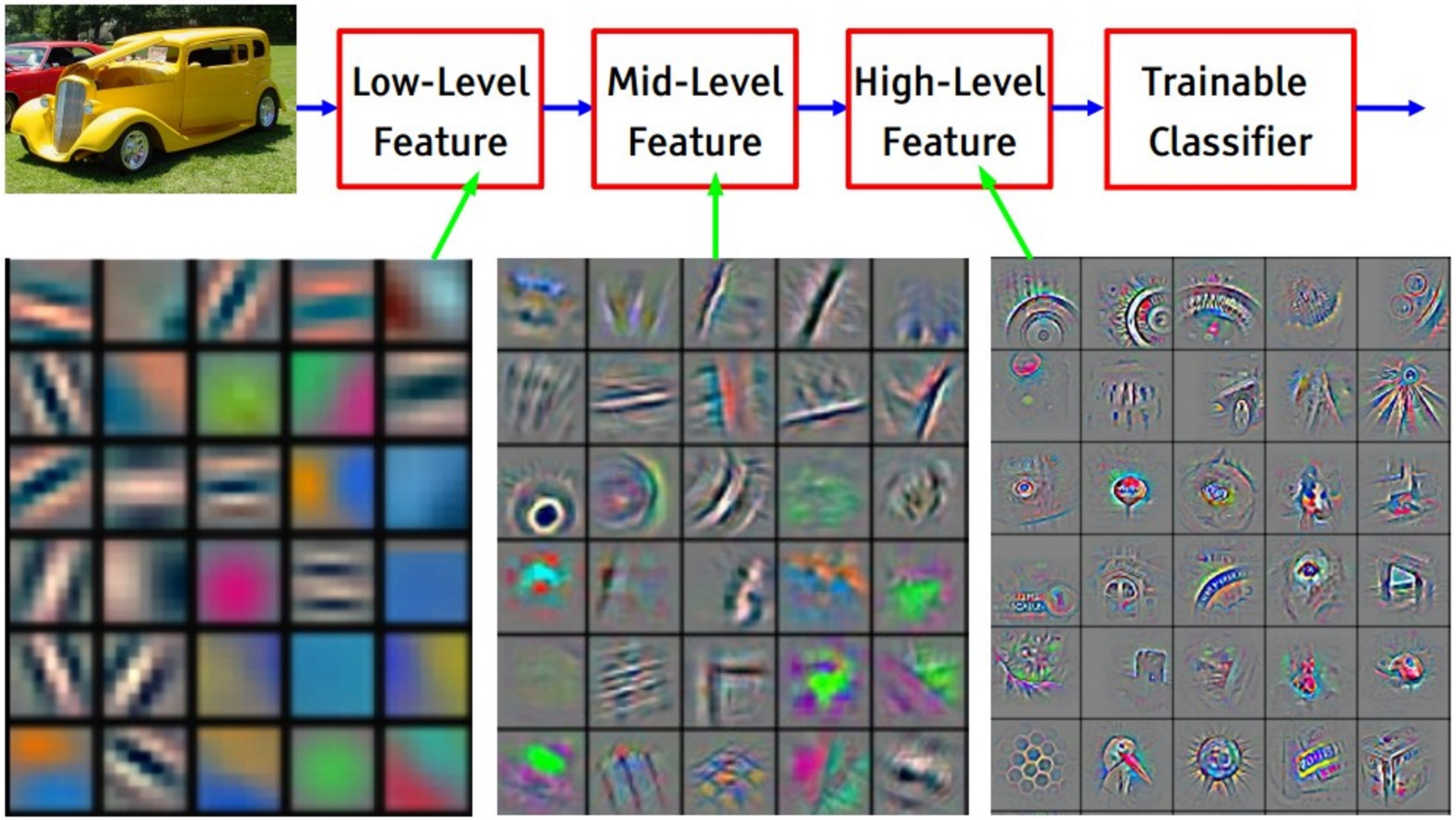
Feature Learning

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



Feature Learning

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

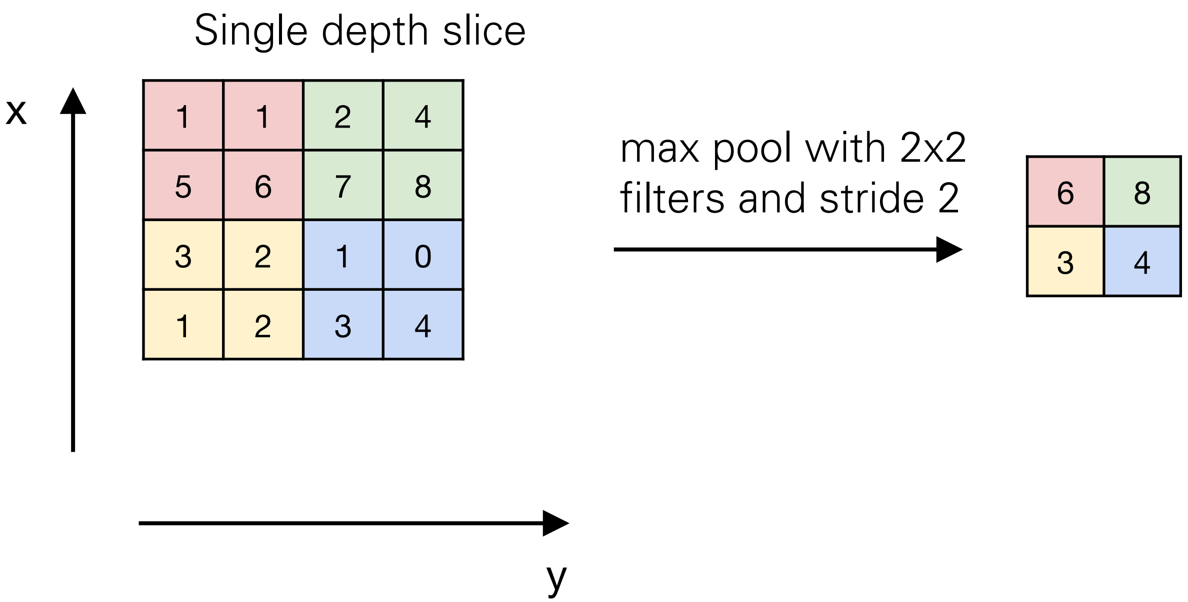


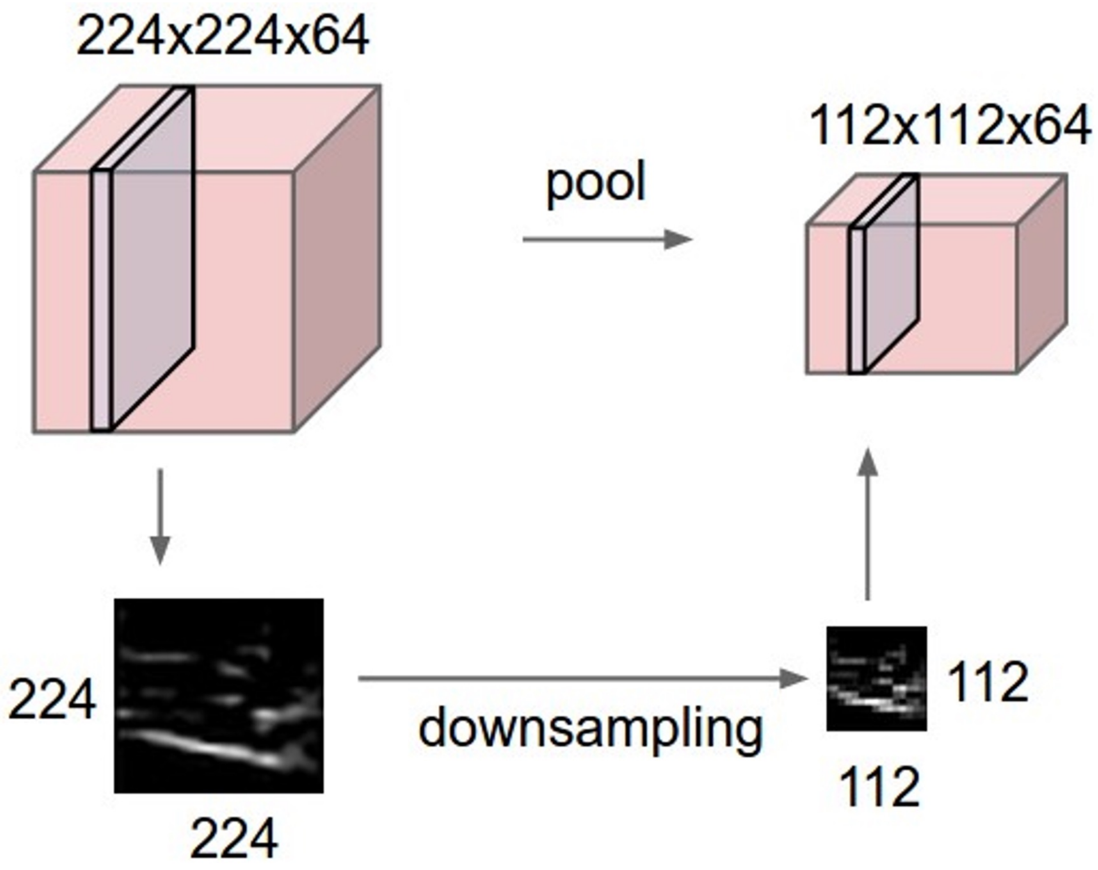
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide credit: Yann LeCun

Pooling layer

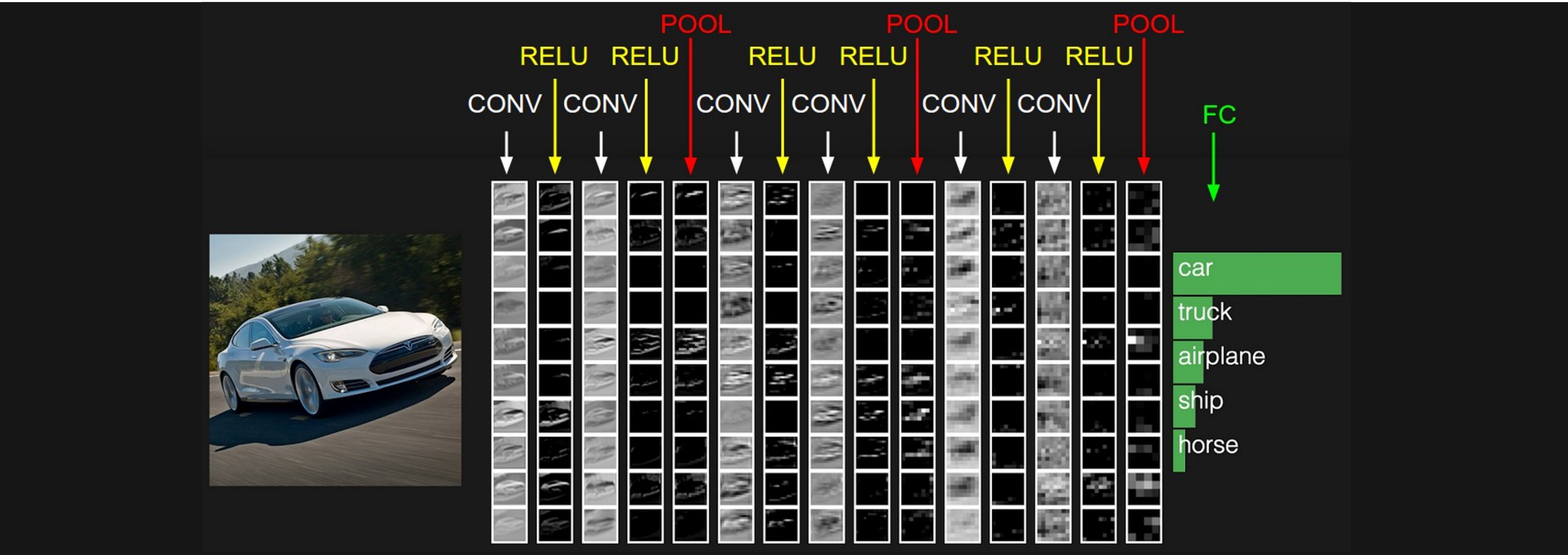
- makes the representations smaller and more manageable
- operates over each activation map independently:
- Max pooling, average pooling, etc.





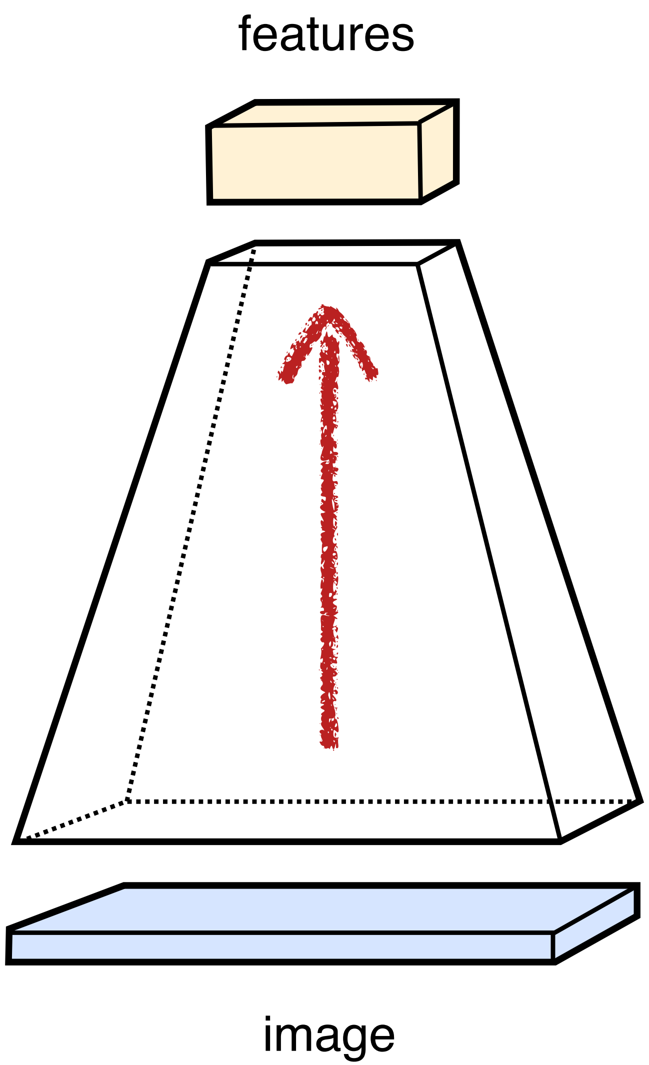
Fully connected layer

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



Design Guidelines

Design Guidelines



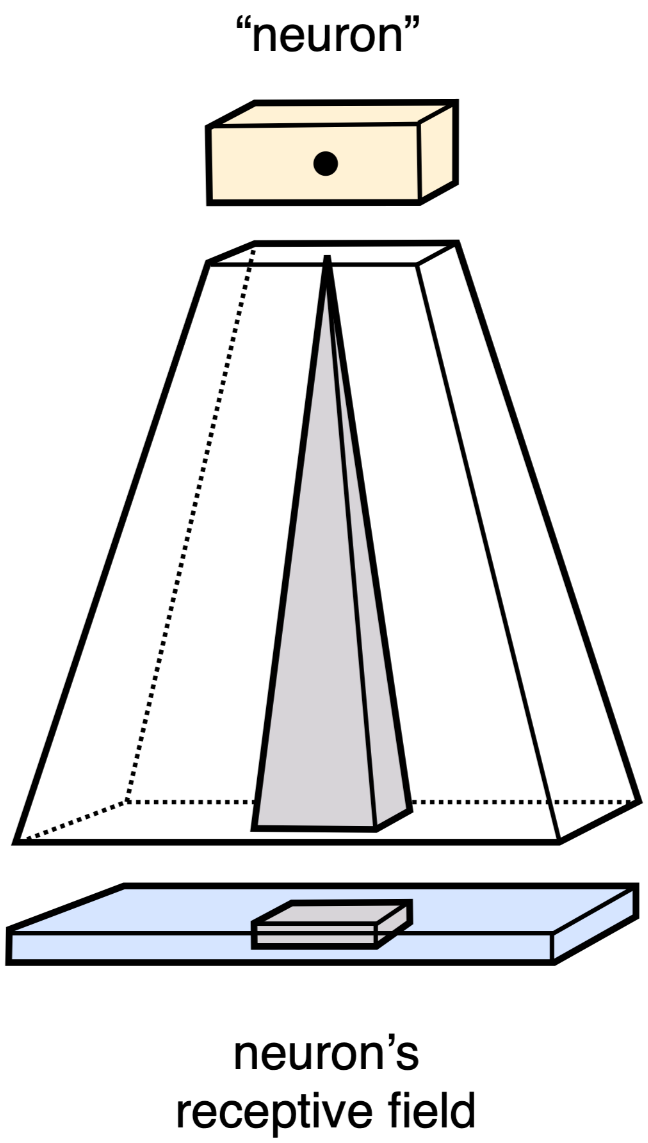
Guideline 1: Avoid tight bottlenecks

- **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

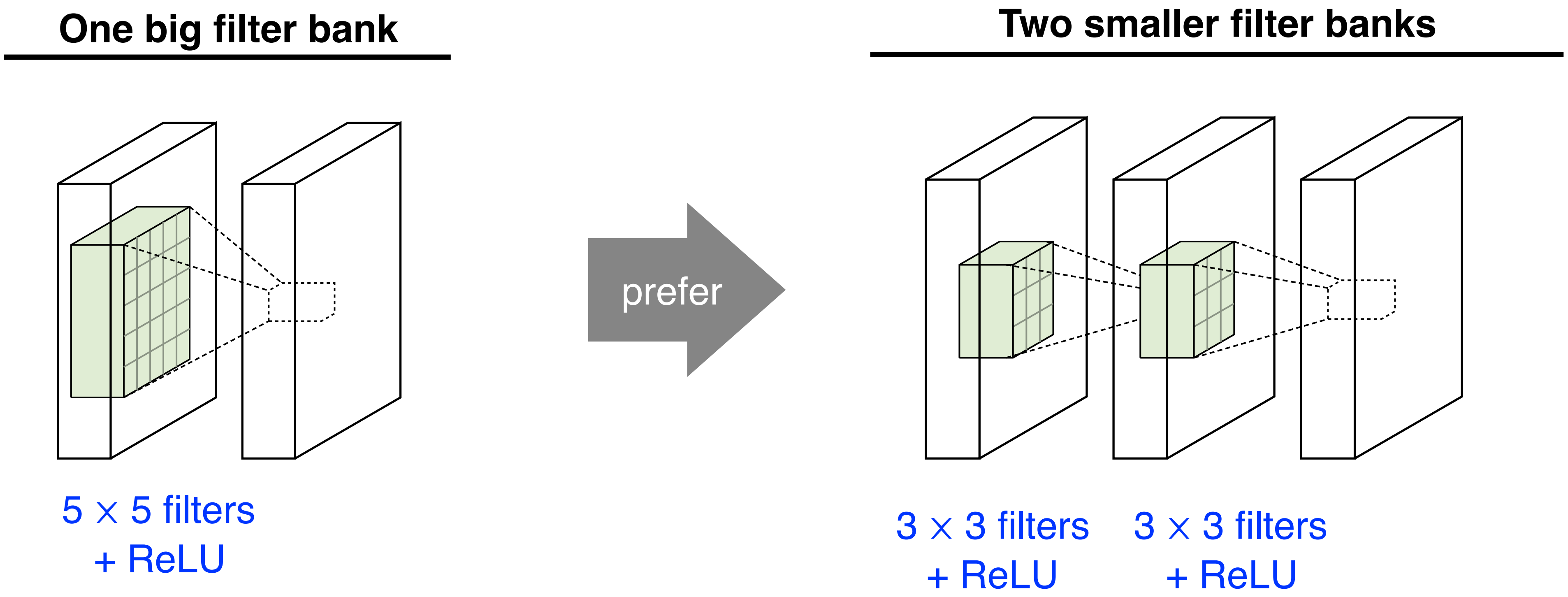


Must be large enough

- **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



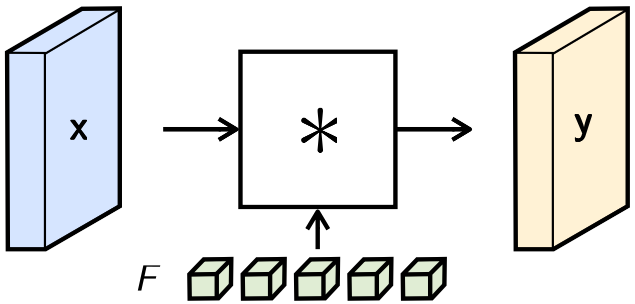
- **Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers
- **Reason:** Far fewer feature channels (quadratic speed/space gain)
- **Moral:** Optimize your architecture

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

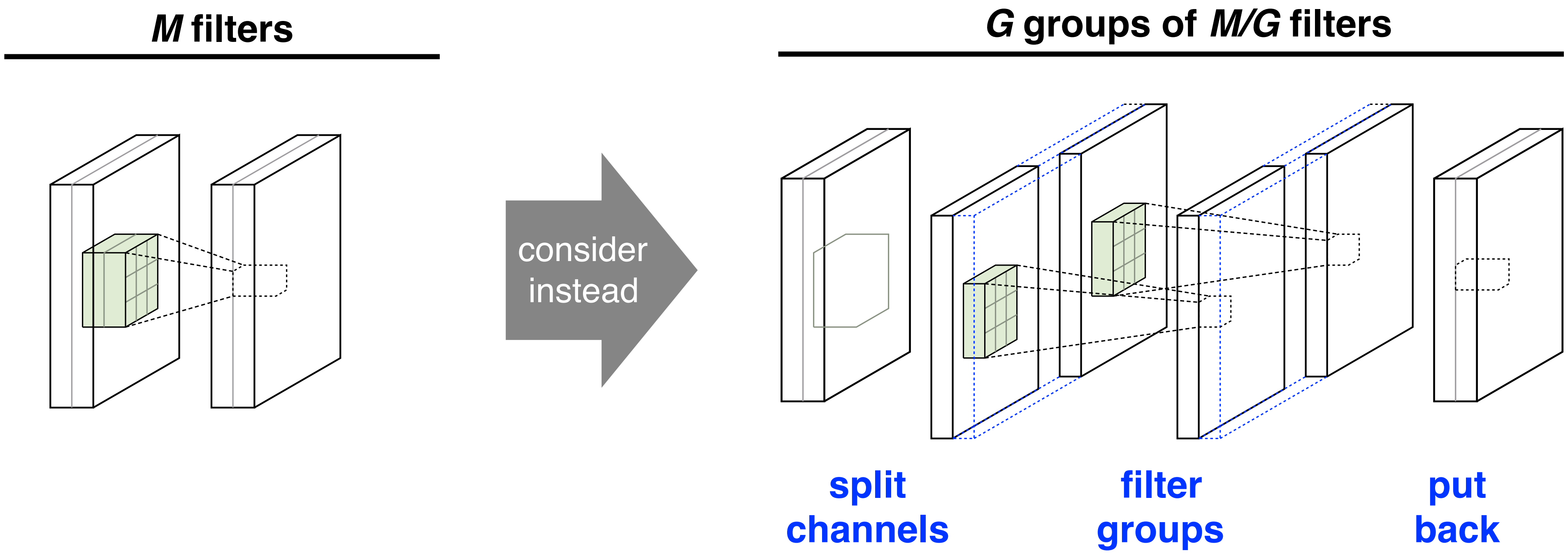
$$H_f \times W_f \times C \times K$$

$$\text{complexity} \propto C \times K$$

Design Guidelines

Guideline 4:

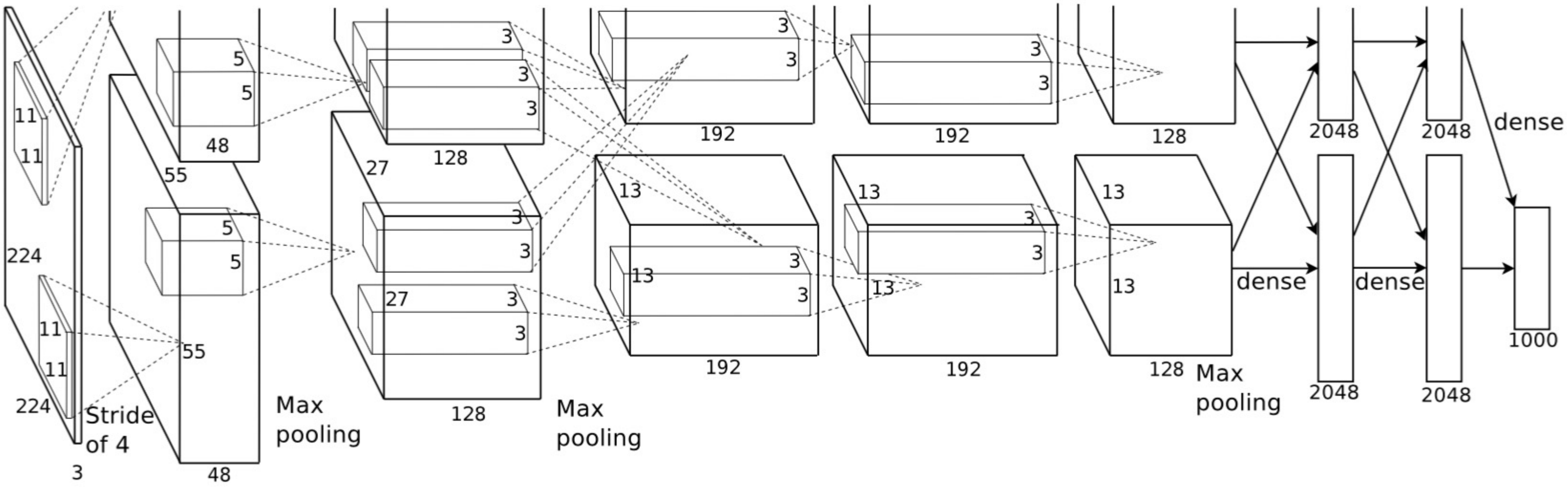
Less computations with filter groups



Did we see this before?

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

AlexNet

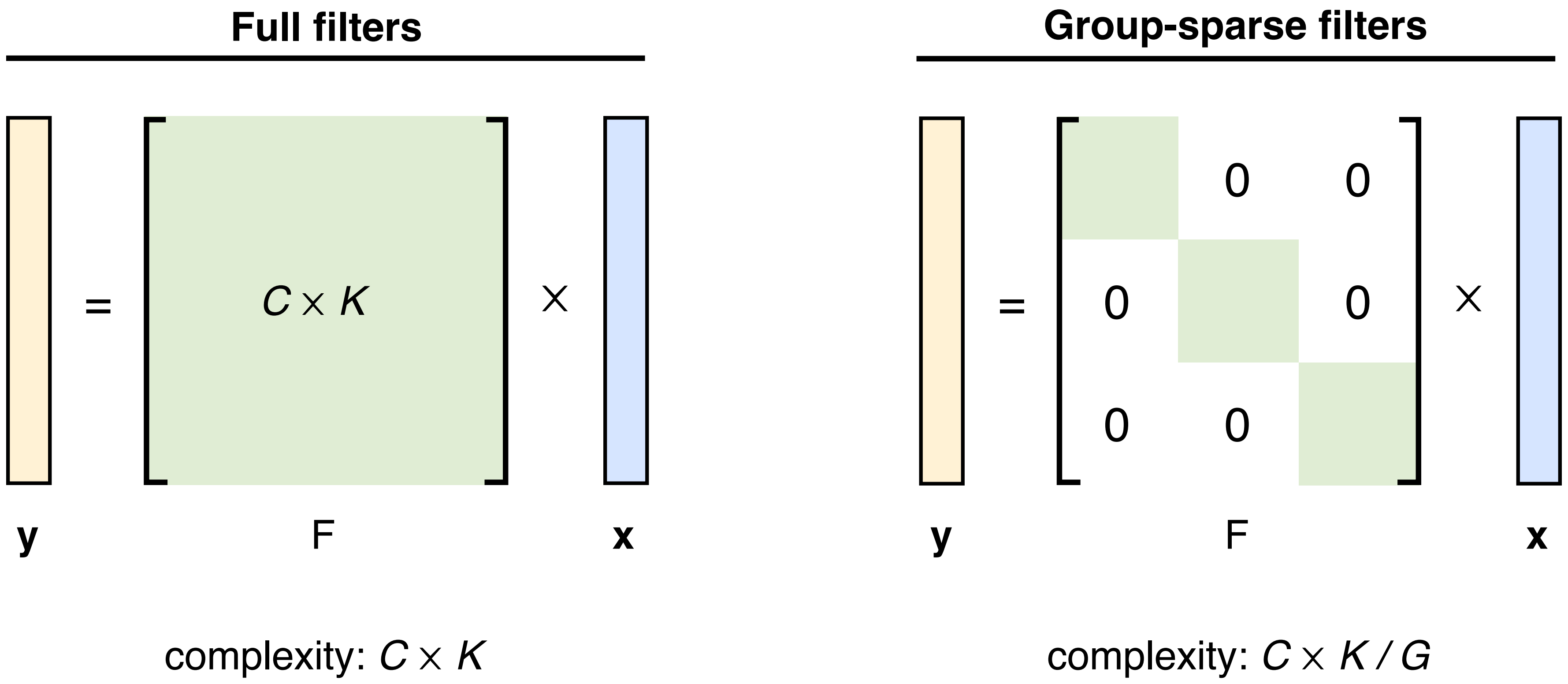


A. Krizhevsky, I. Sutskever, and G. E. Hinton. **Imagenet classification with deep convolutional neural networks**. In NIPS 2012.

Design Guidelines

Guideline 4:

Less computations with filter groups

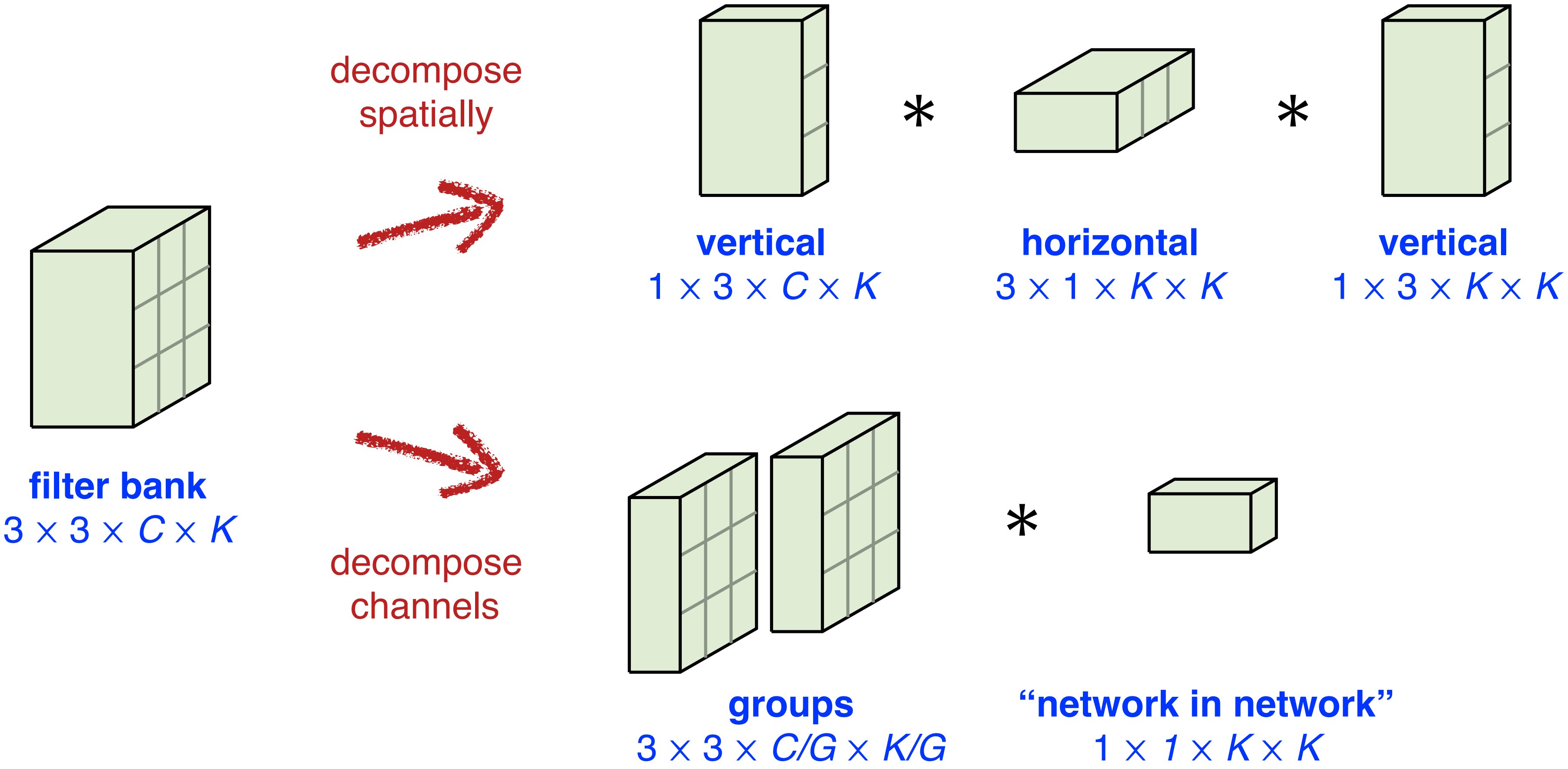


Groups = filters, seen as a matrix, have a “block” structure

Design Guidelines

Guideline 5:

Low-rank decompositions

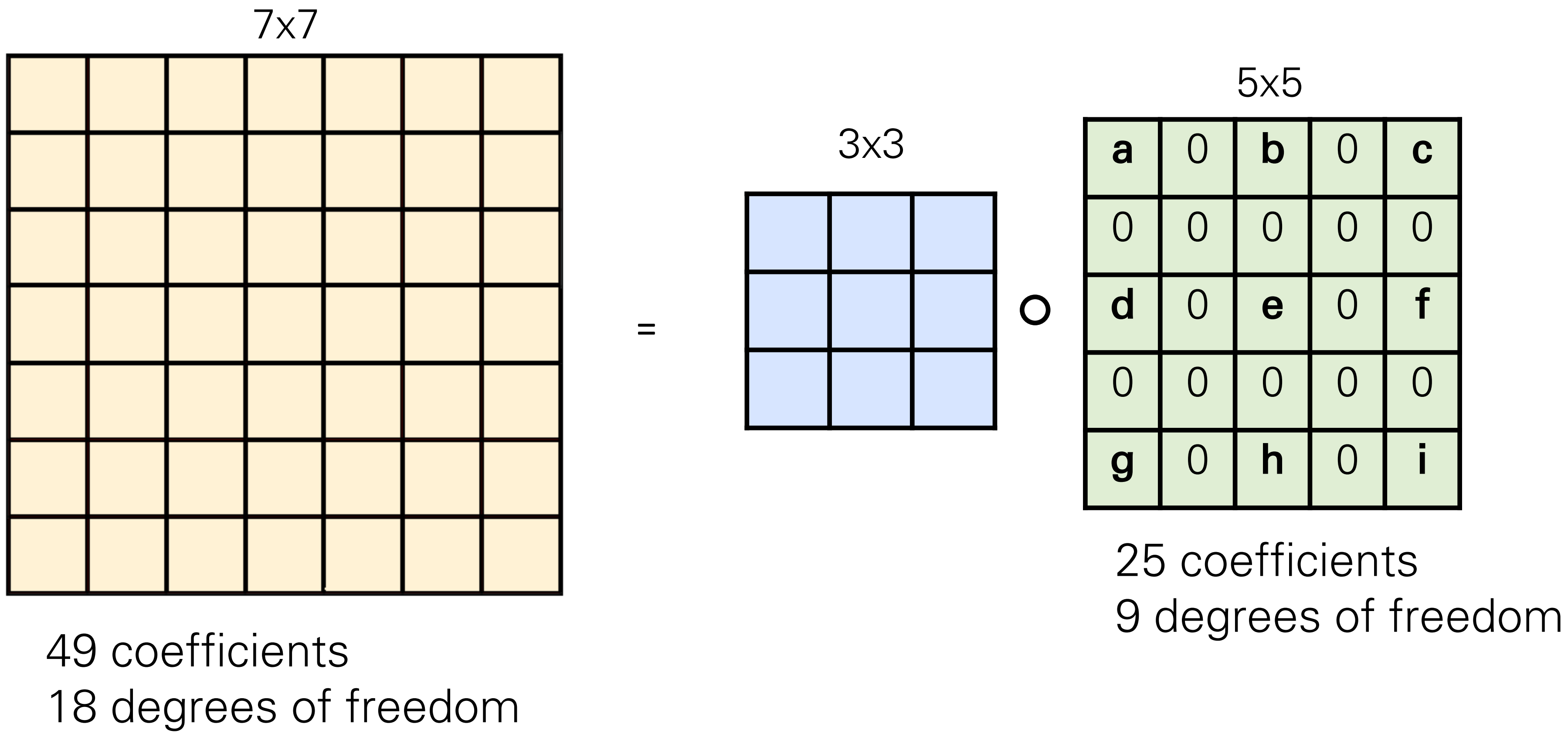


Make sure to mix the information

Design Guidelines

Guideline 6:

Dilated Convolutions



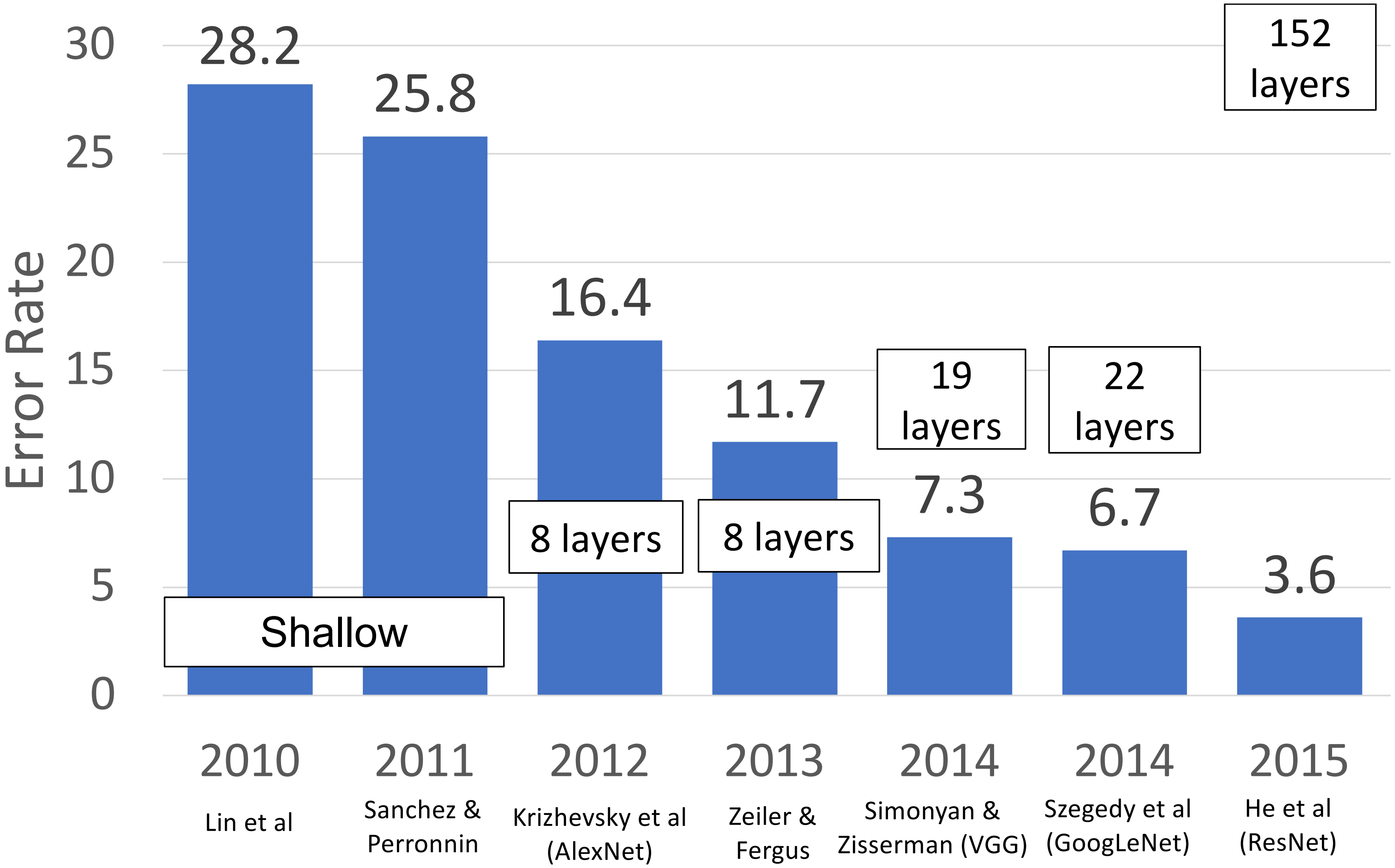
Exponential expansion of the receptive field without loss of resolution

Convolutional Neural Network Demo

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

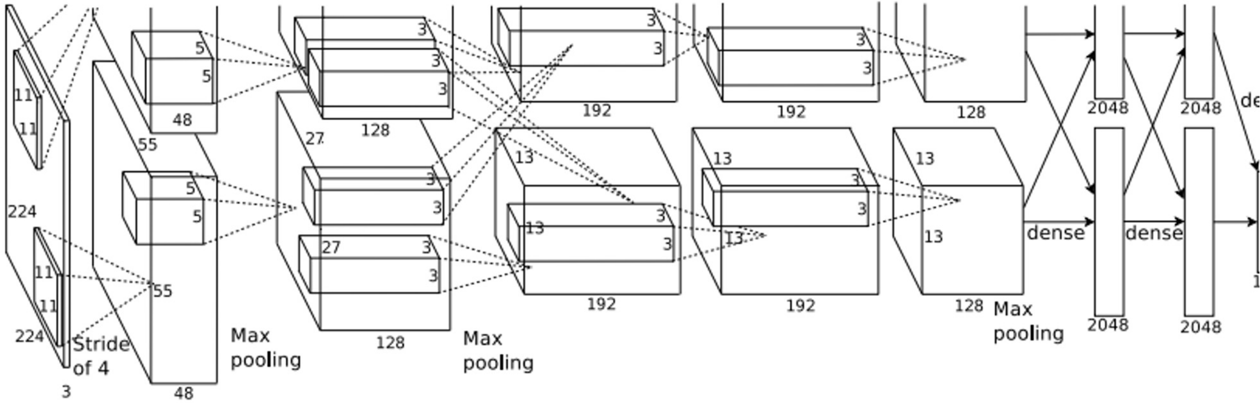
CNN Architectures

ImageNet Classification Challenge



AlexNet

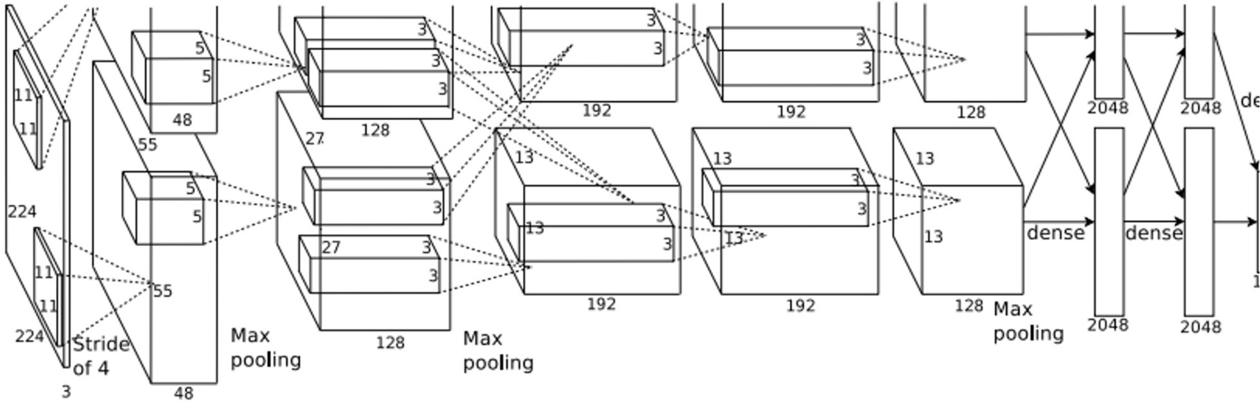
[Krizhevsky et al. 2012]



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%

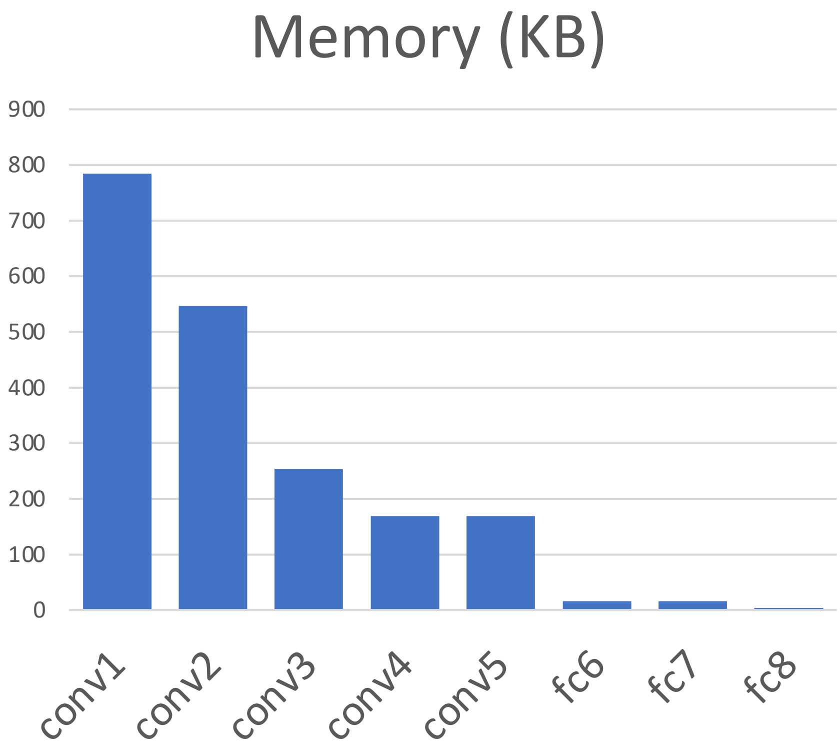
AlexNet

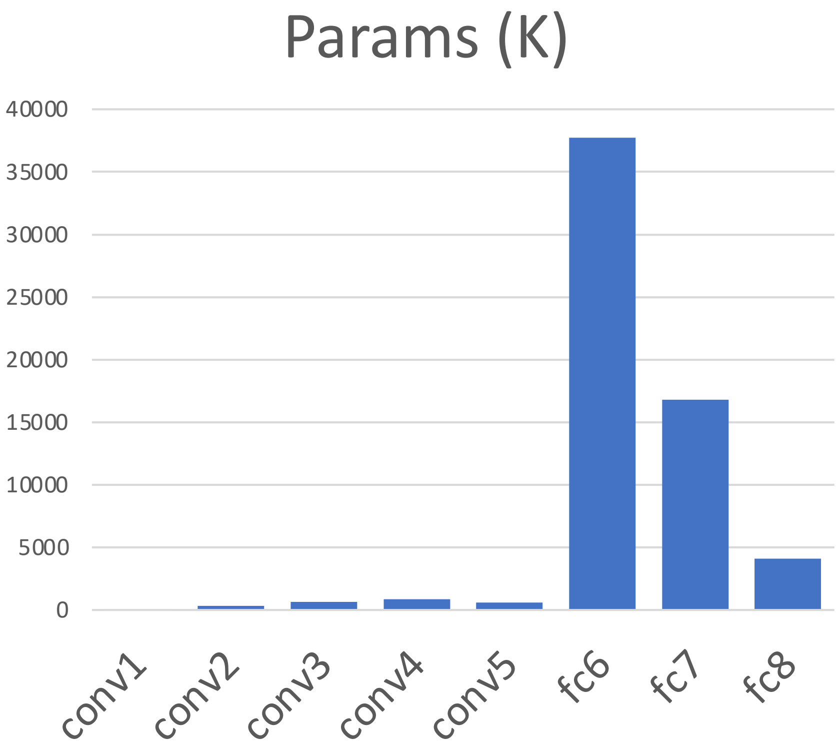


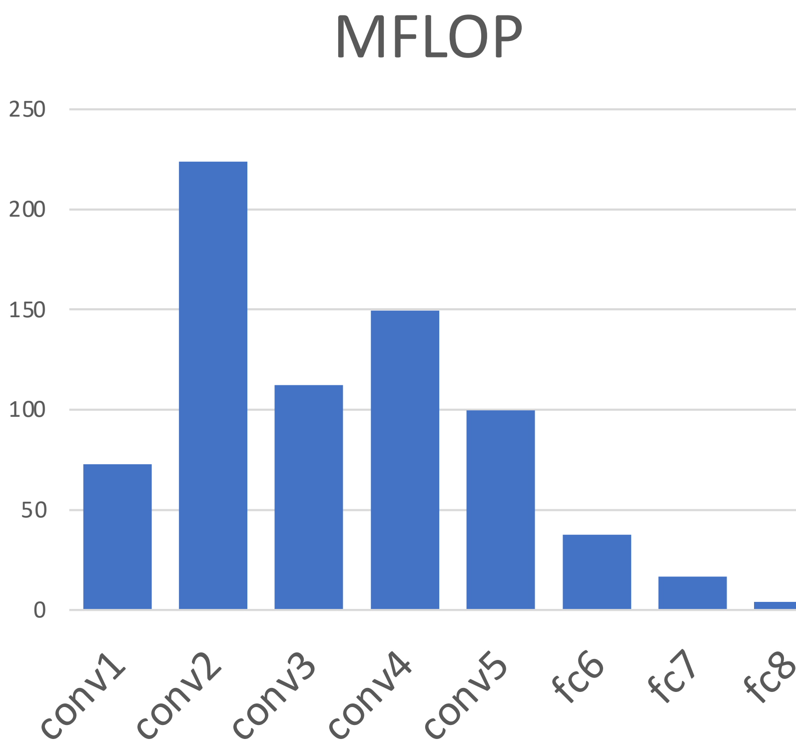
Most of the **memory usage** is in the early convolution layers

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

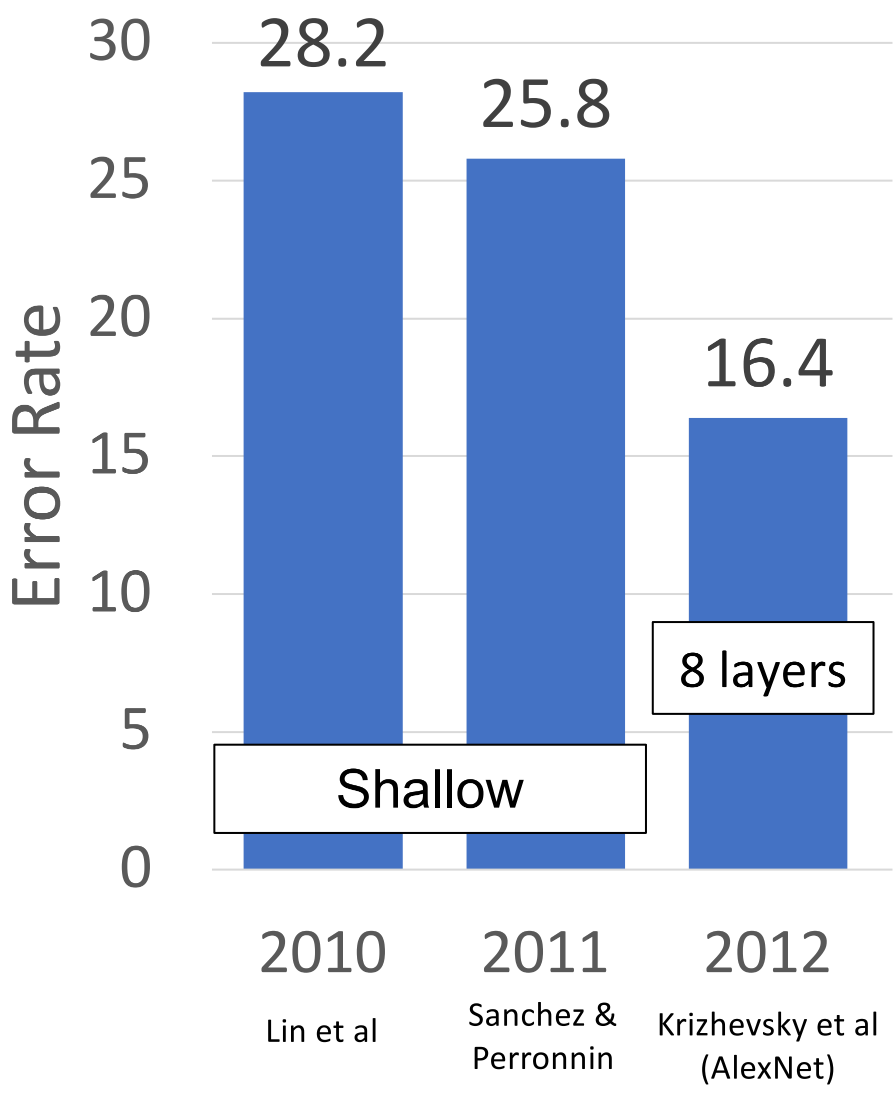
Most **floating-point ops** occur in the convolution layers



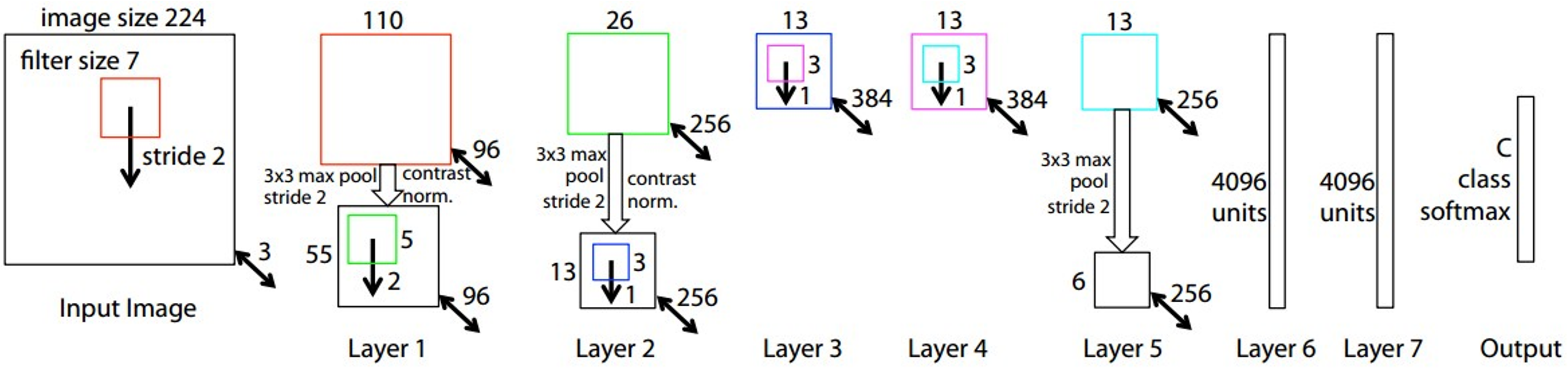




ImageNet Classification Challenge



ZFNet: A Bigger AlexNet

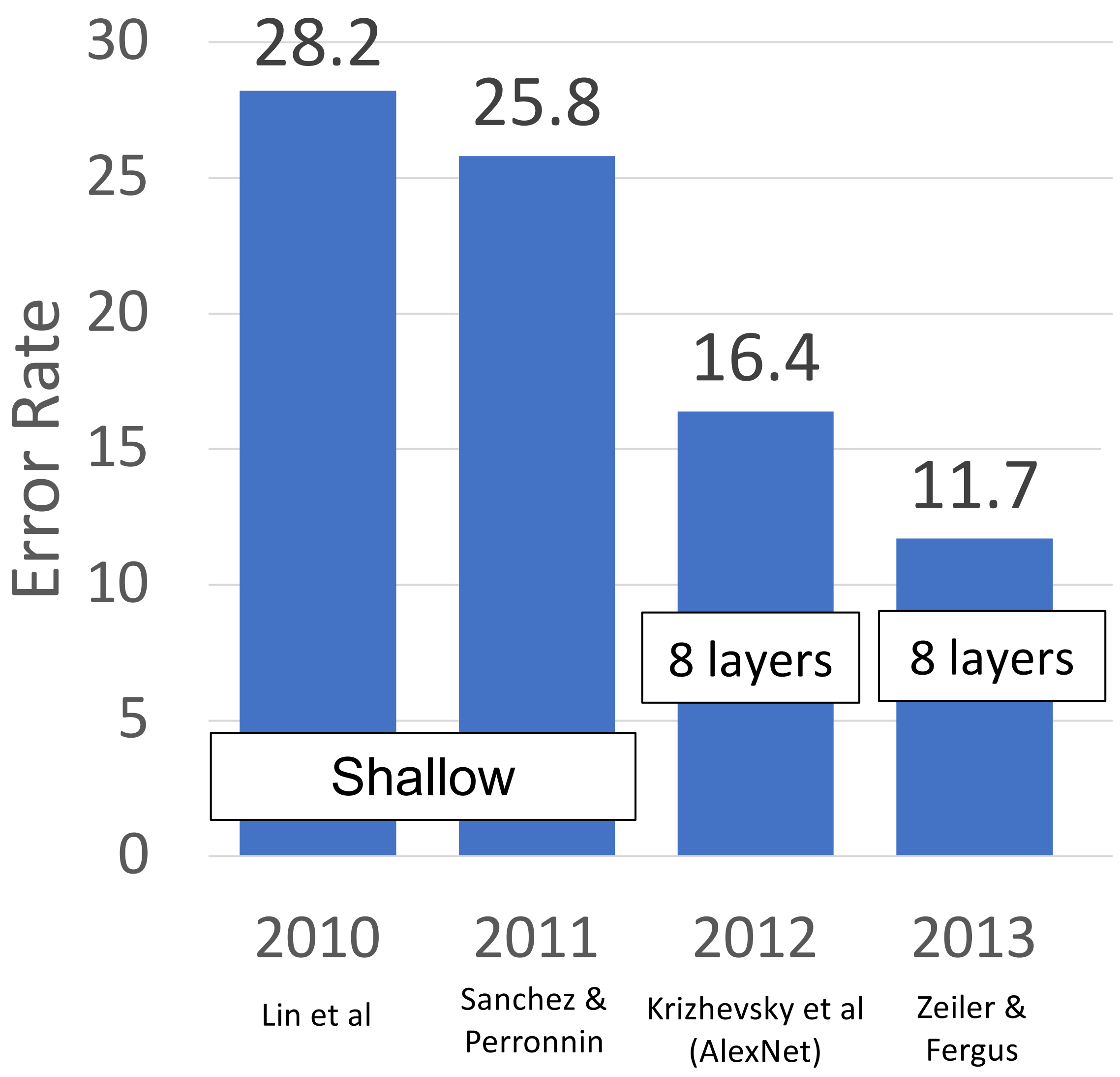


AlexNet but:

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



INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image
(only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image)		
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
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		

VGG-16 Net

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

VGG: Deeper Networks, Regular Design

VGG Design rules:

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

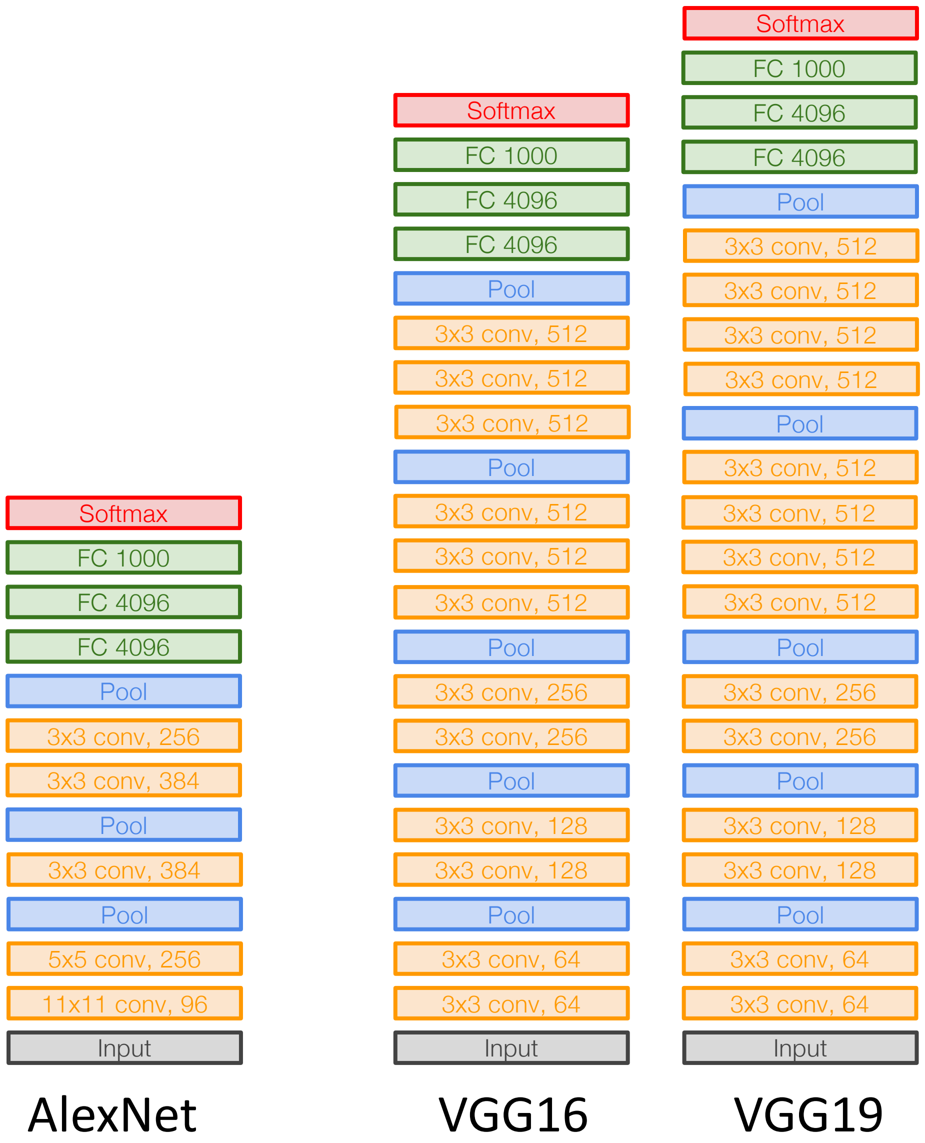
Network has 5 convolutional stages:
Stage 1: conv-conv-pool

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)



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

Network has 5 convolutional stages:
Stage 1: conv-conv-pool

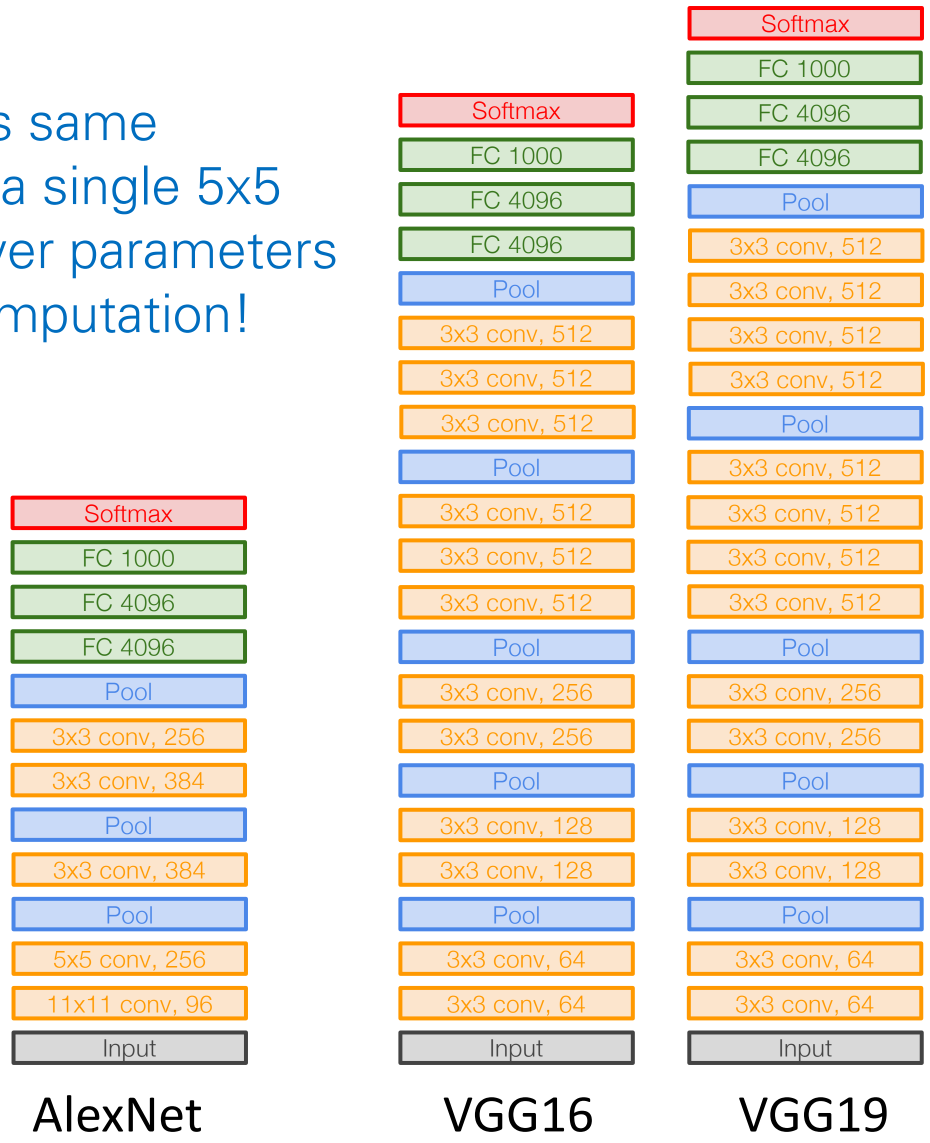
Stage 2: conv-conv-pool

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)

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



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

Network has 5 convolutional stages:
Stage 1: conv-conv-pool

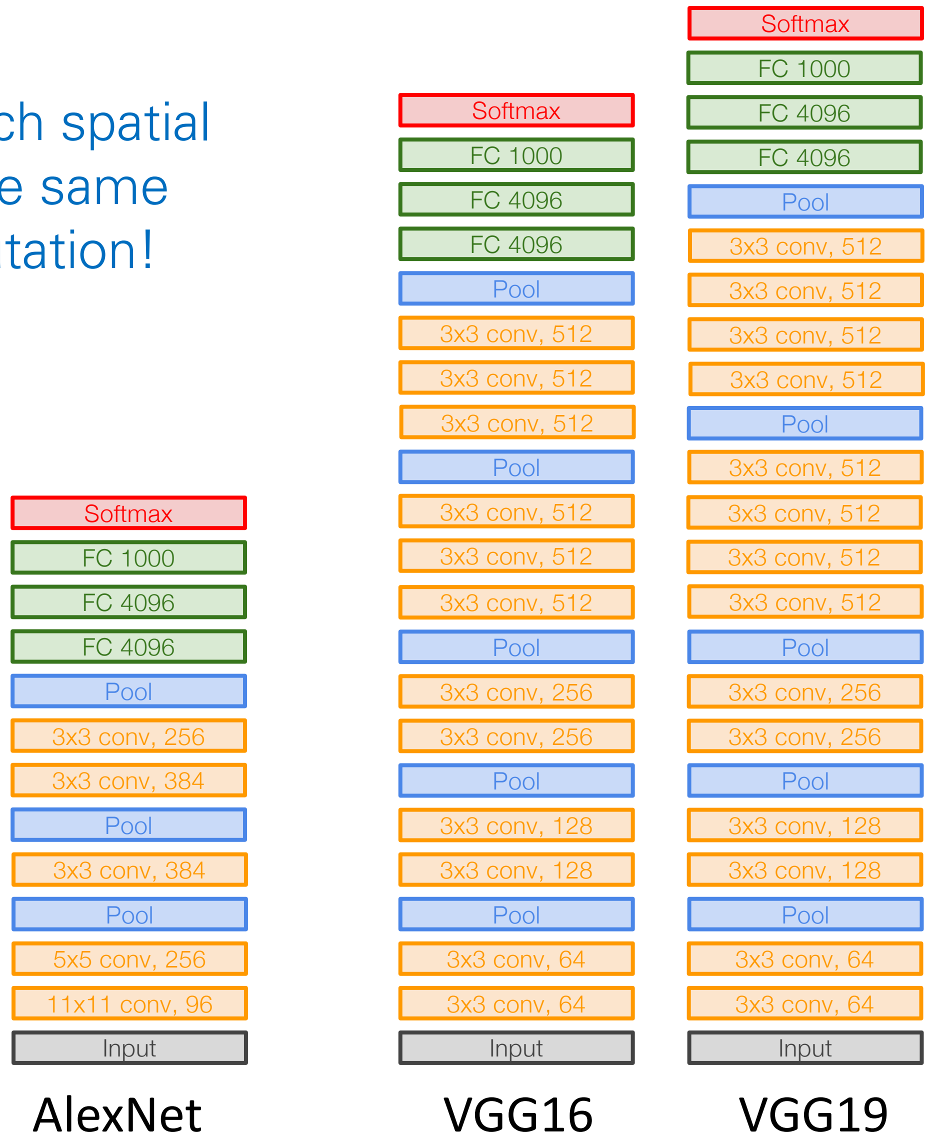
Stage 2: conv-conv-pool

Stage 3: conv-conv-pool

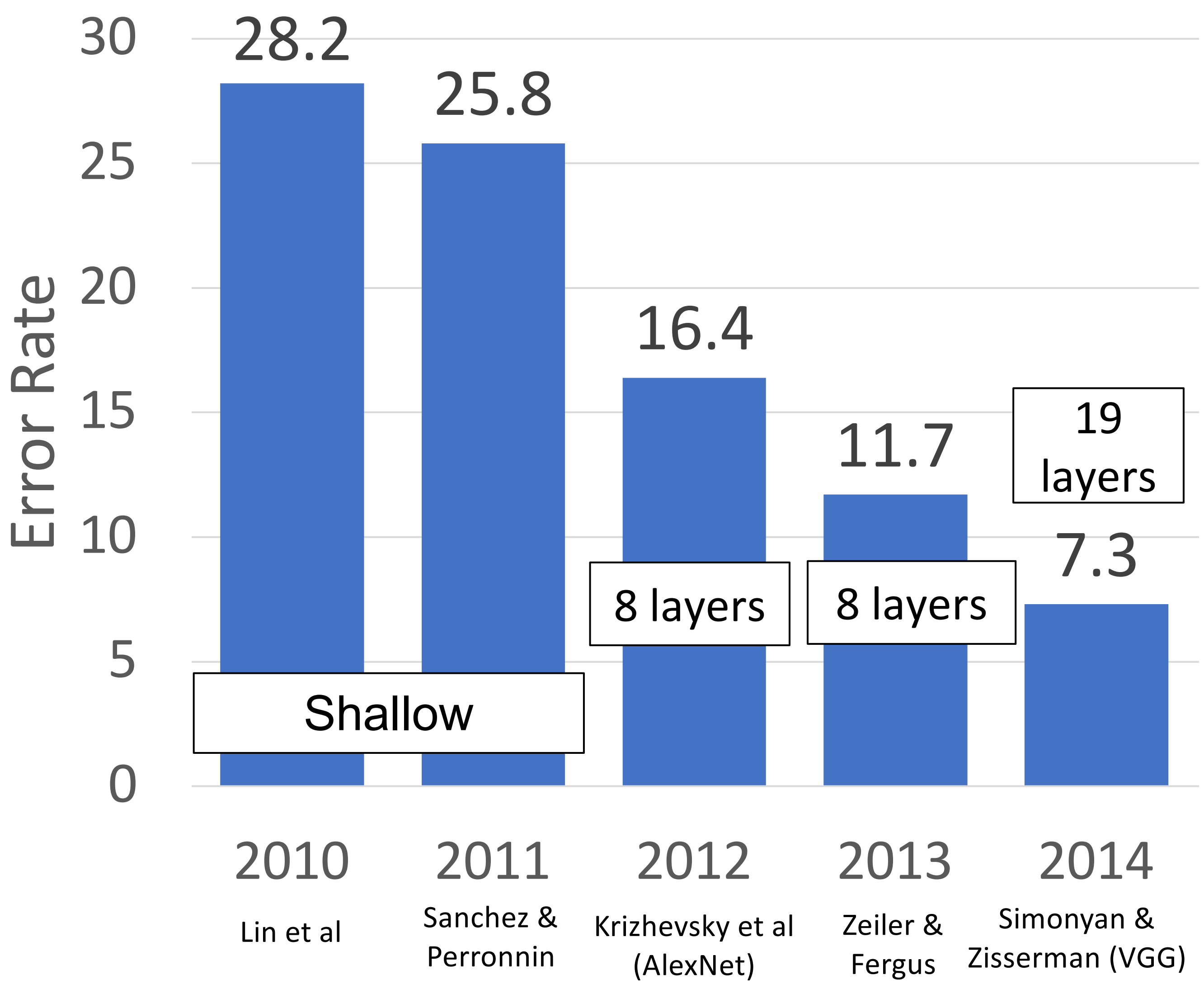
Stage 4: conv-conv-conv-[conv]-pool
Stage 5: conv-conv-conv-[conv]-pool

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

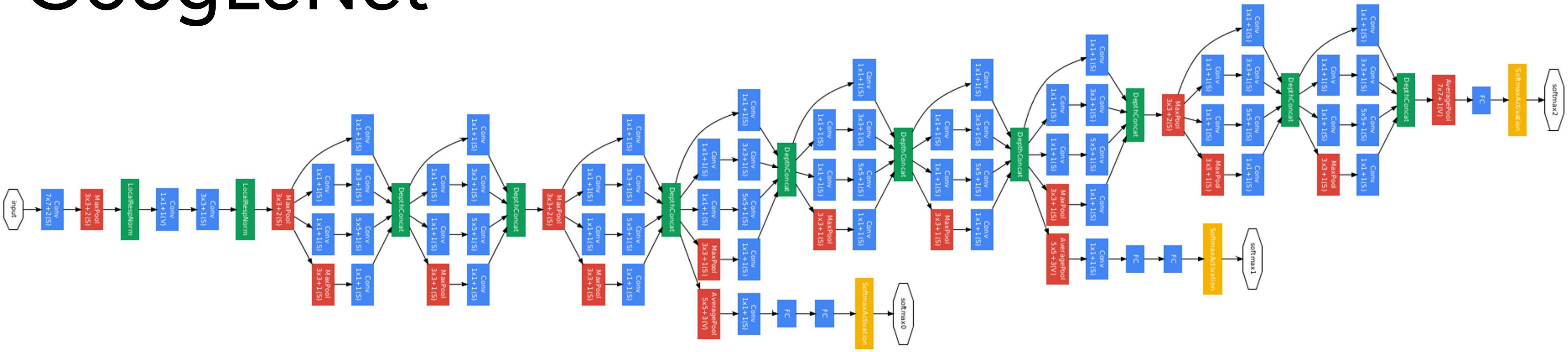
Conv layers at each spatial resolution take the same amount of computation!



ImageNet Classification Challenge



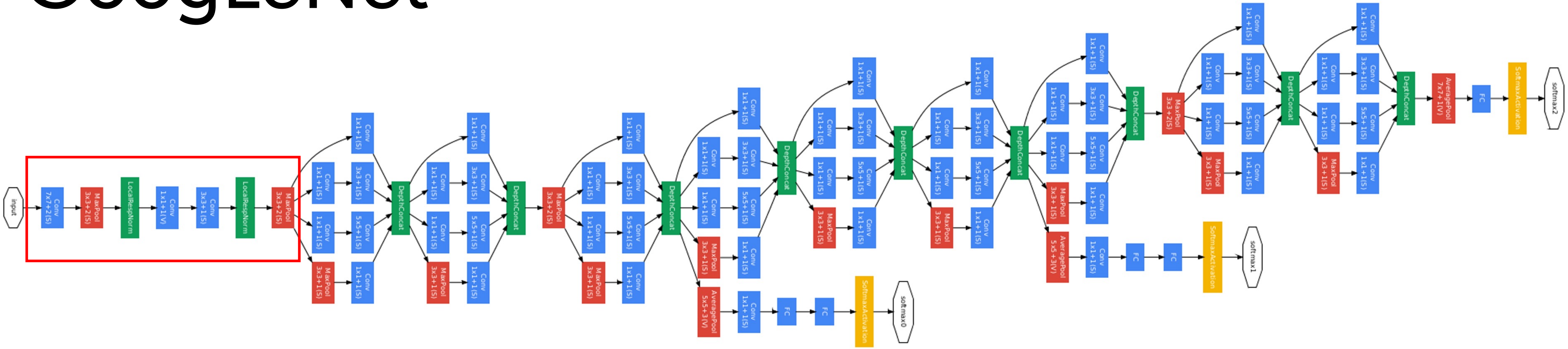
GoogLeNet



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

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

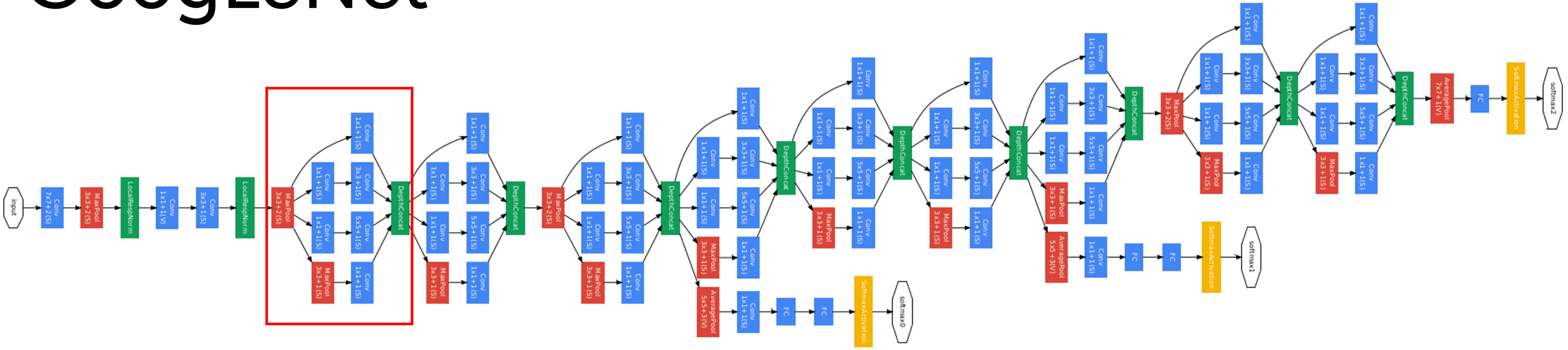
GoogLeNet

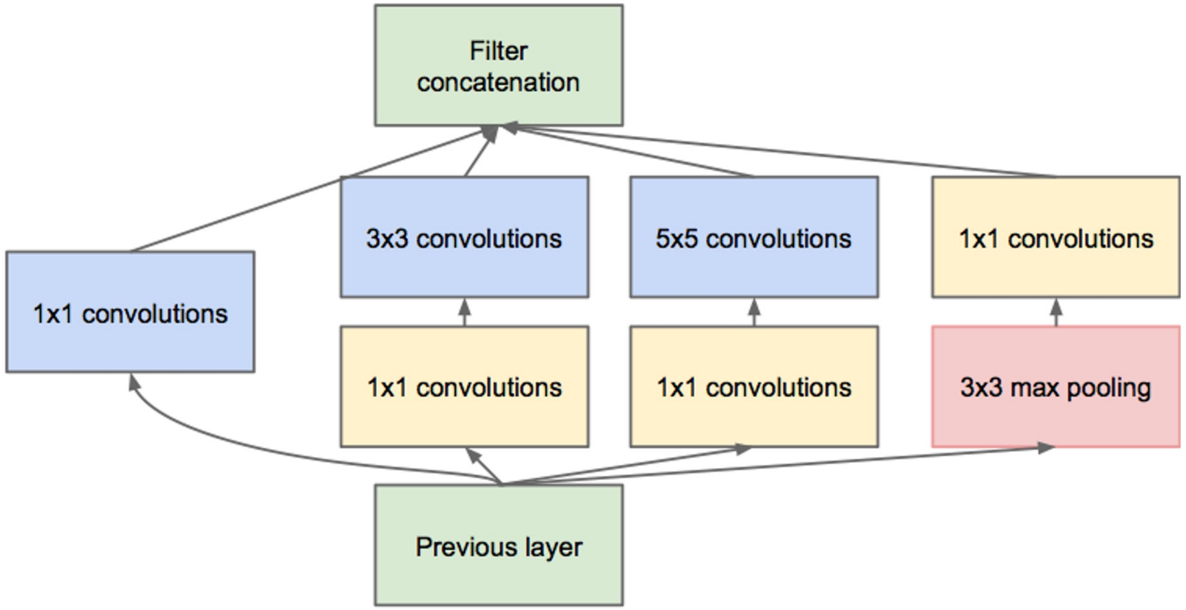


Stem network at the start aggressively downsamples input (Recall in VGG-16: Most of the compute was at the start)

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

GoogLeNet





Inception module

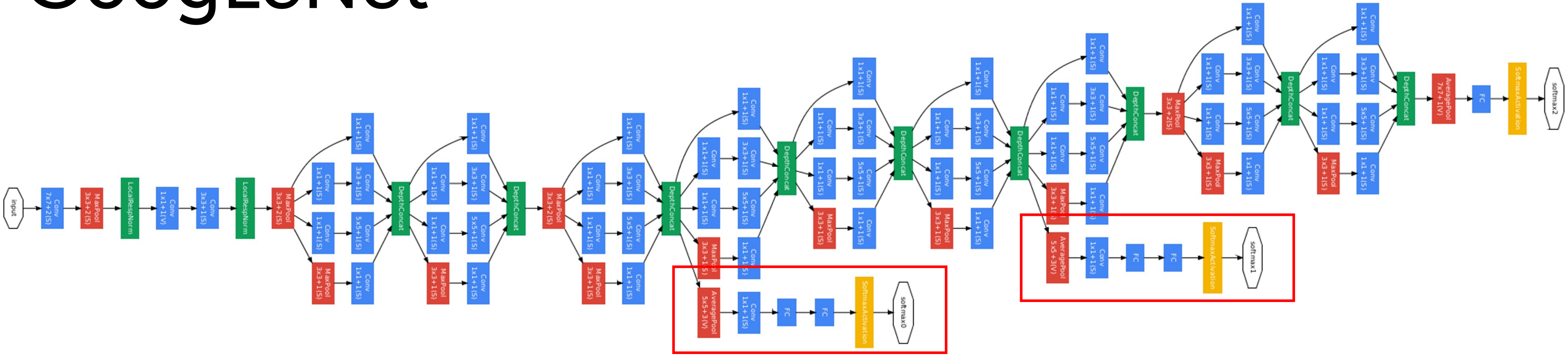
Local unit with parallel branches

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

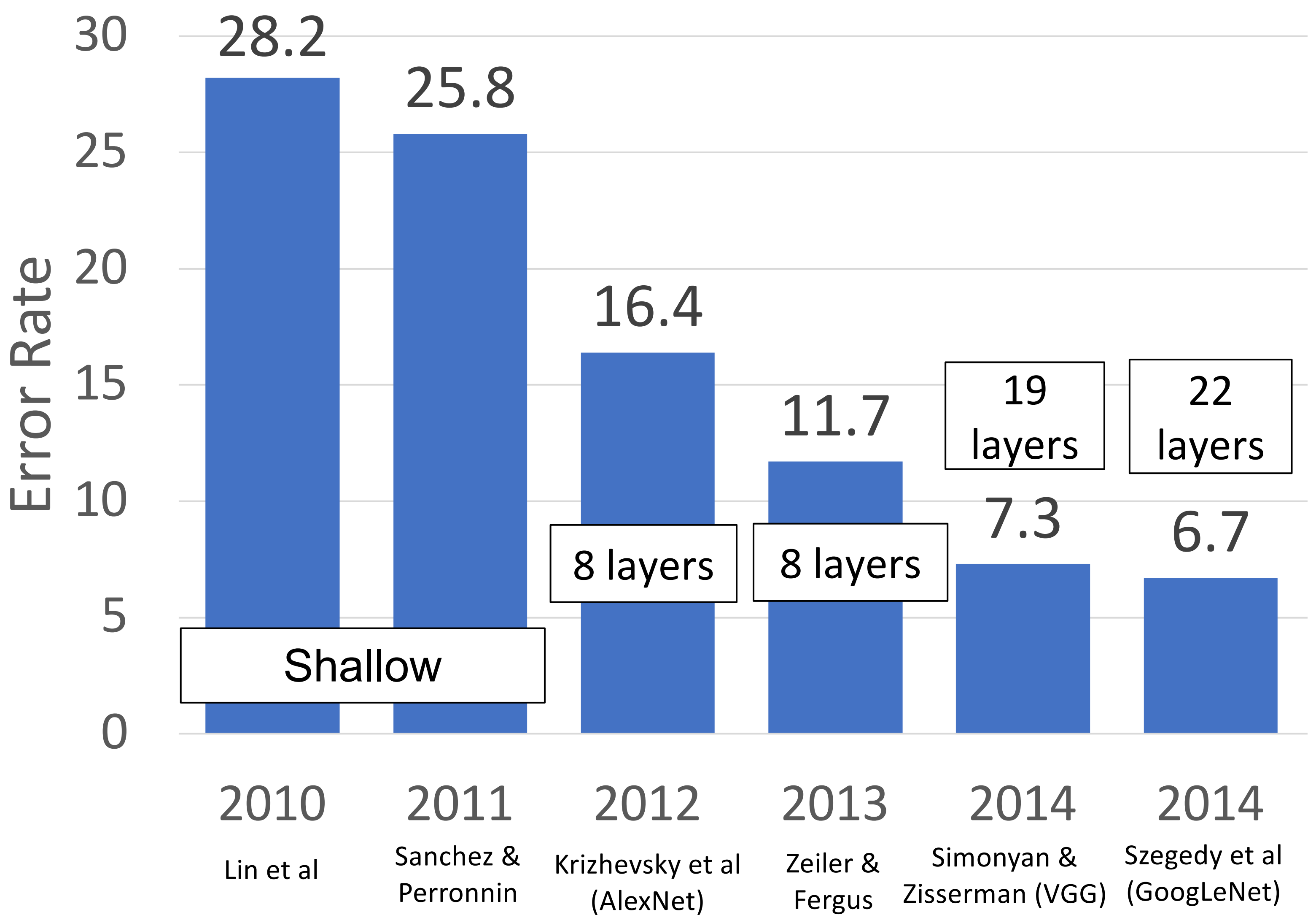
Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly

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

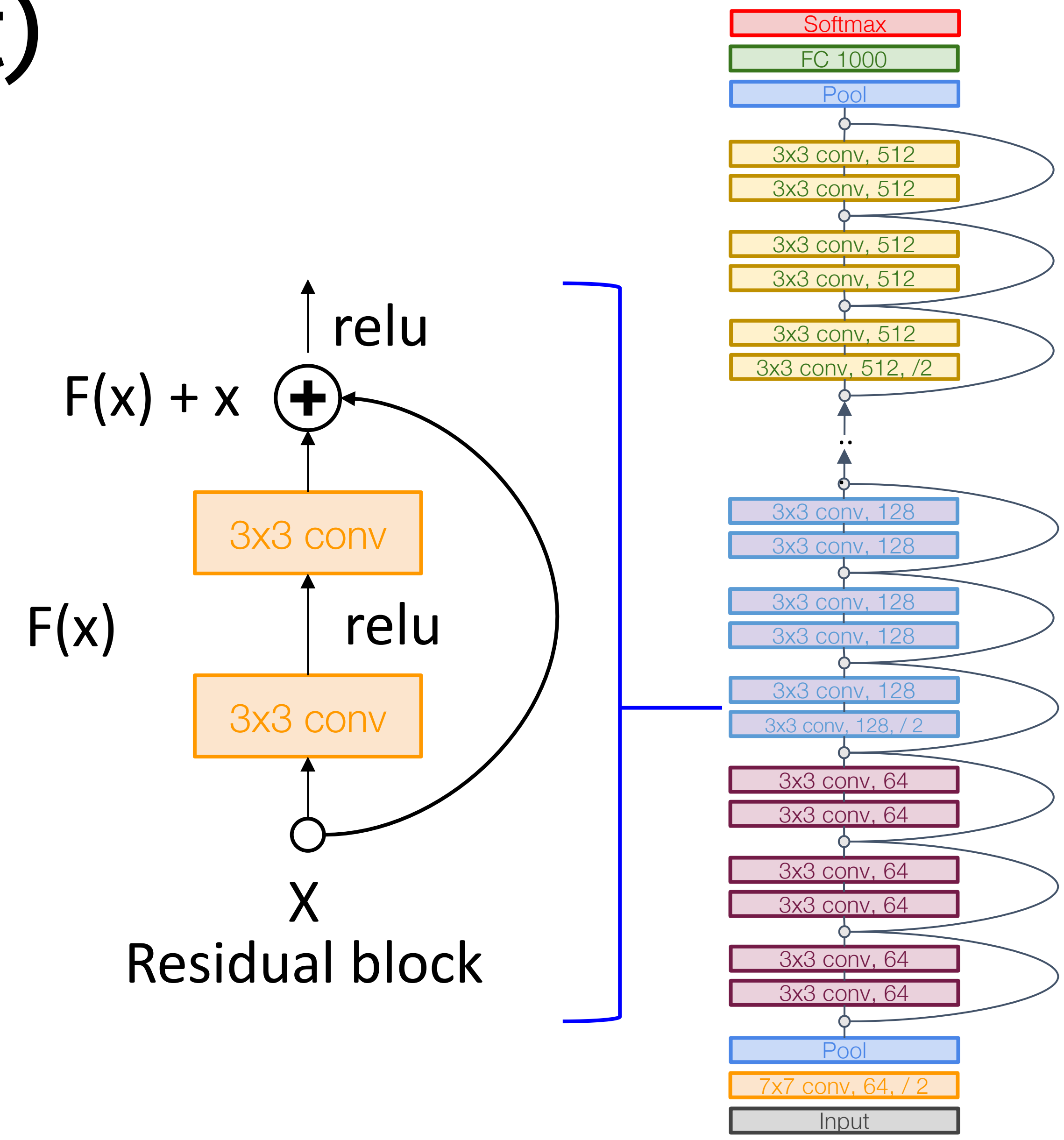


Residual Net (ResNet)

A residual network is a stack of many residual blocks

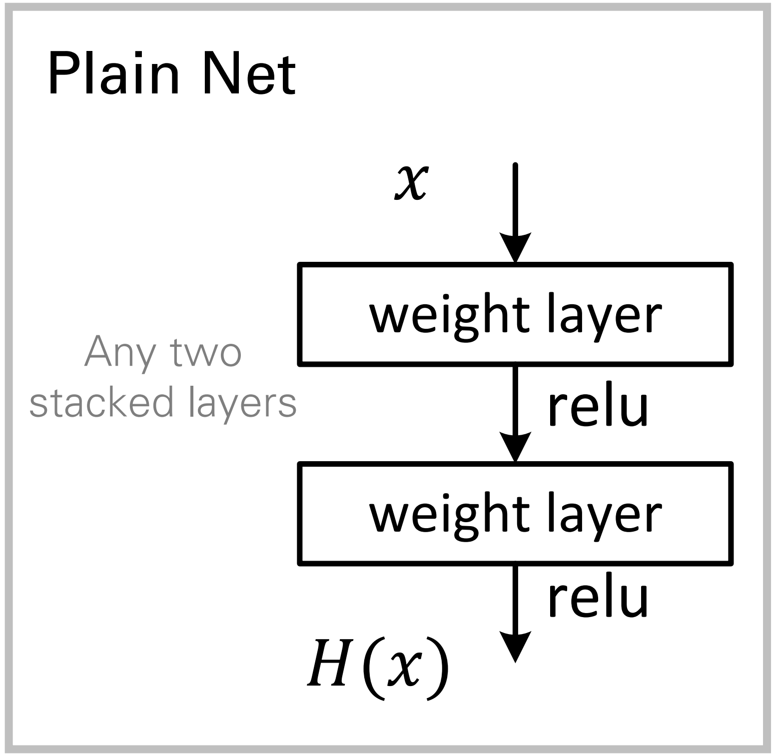
Regular design, like VGG: each residual block has two 3x3 conv

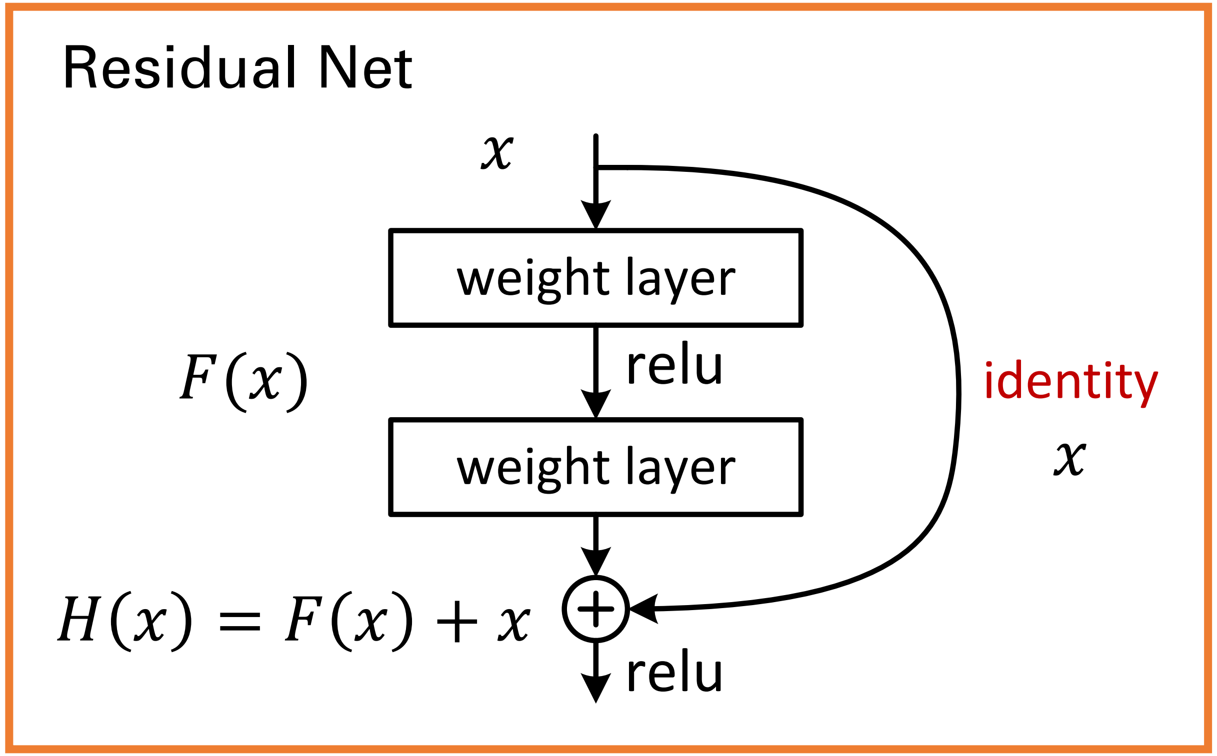
Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Net (ResNet)





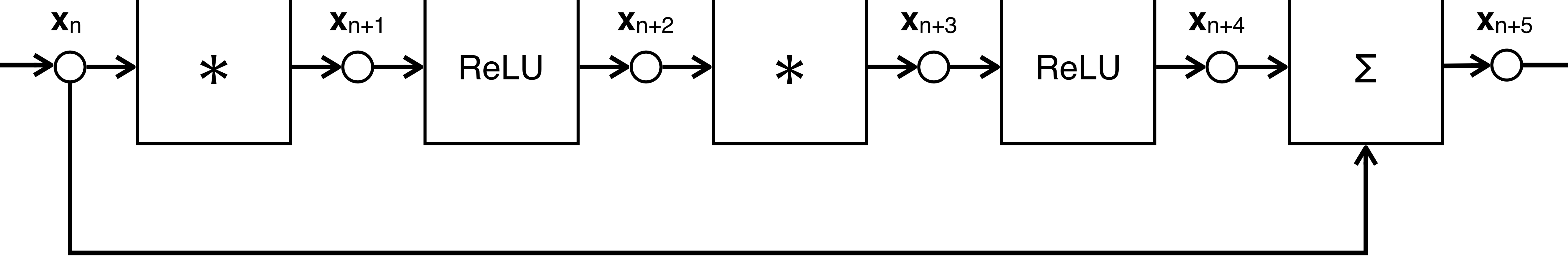
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

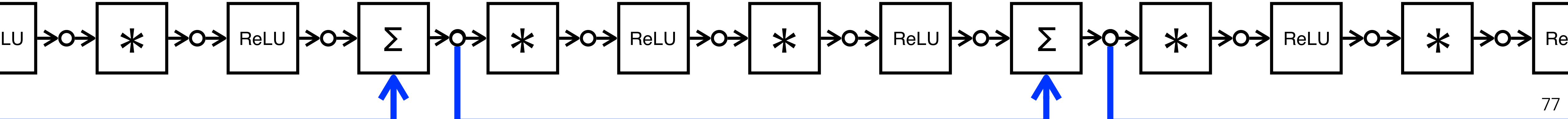
Residual Learning

Fixed identity
// learned residual

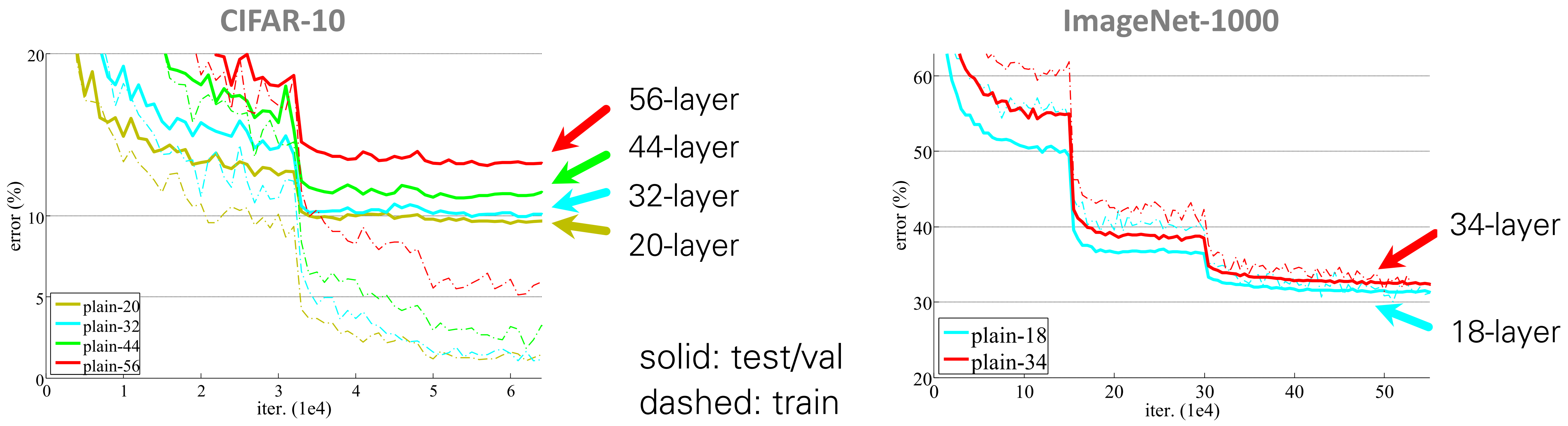
$$\underbrace{\mathbf{x}_{n+5} = \mathbf{x}_n}_{\text{identity}} + \underbrace{(\phi_{\mathrm{ReLU}} \circ \phi_* \circ \phi_{\mathrm{ReLU}} \circ \phi_*)(\mathbf{x}_n)}_{\text{residual}}$$

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





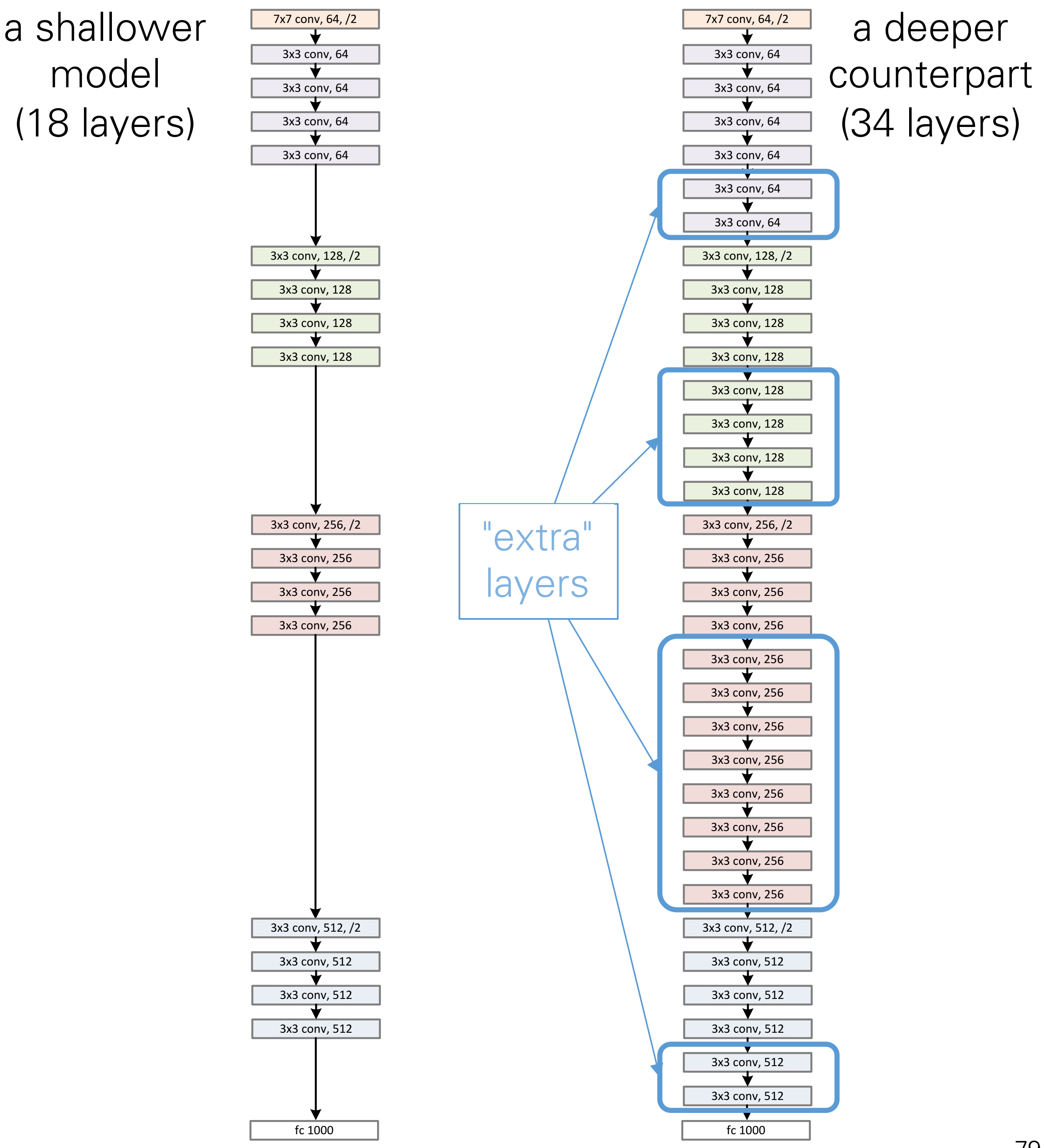
Residual Learning



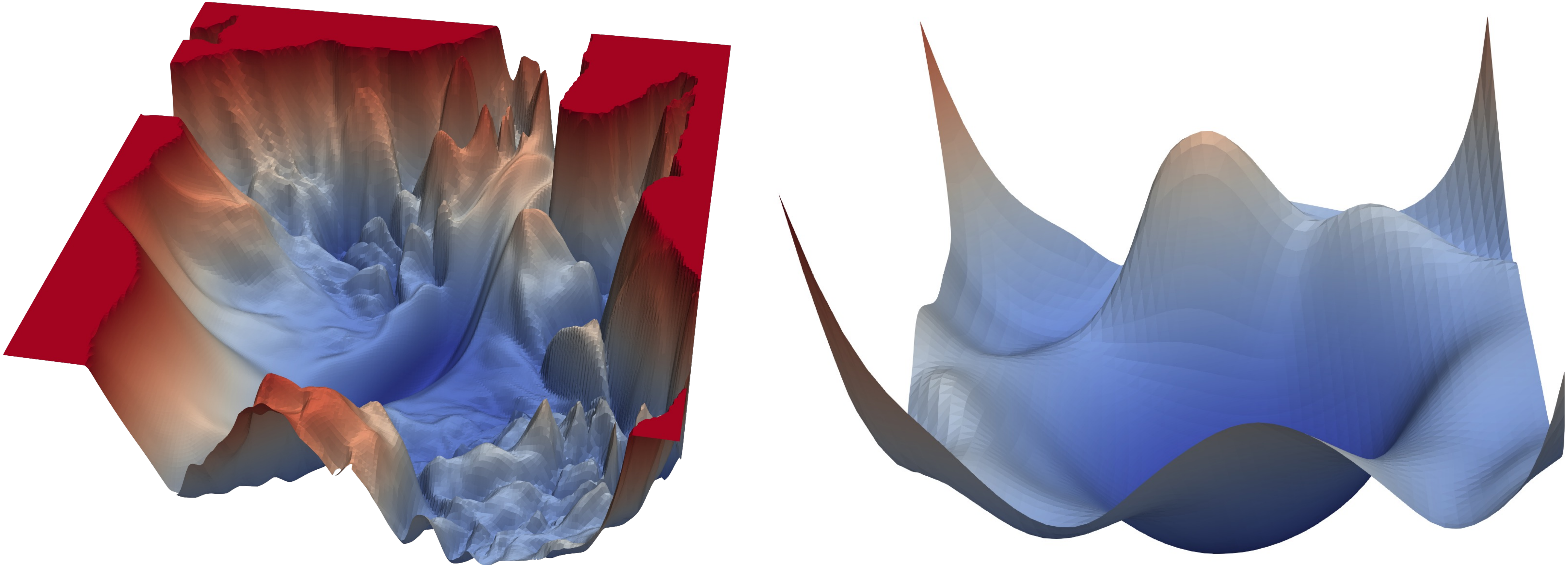
- "Overly deep" plain nets have **higher training error**
- A general phenomenon, observed in many datasets
- This is optimization issue, deeper models are harder to optimize

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



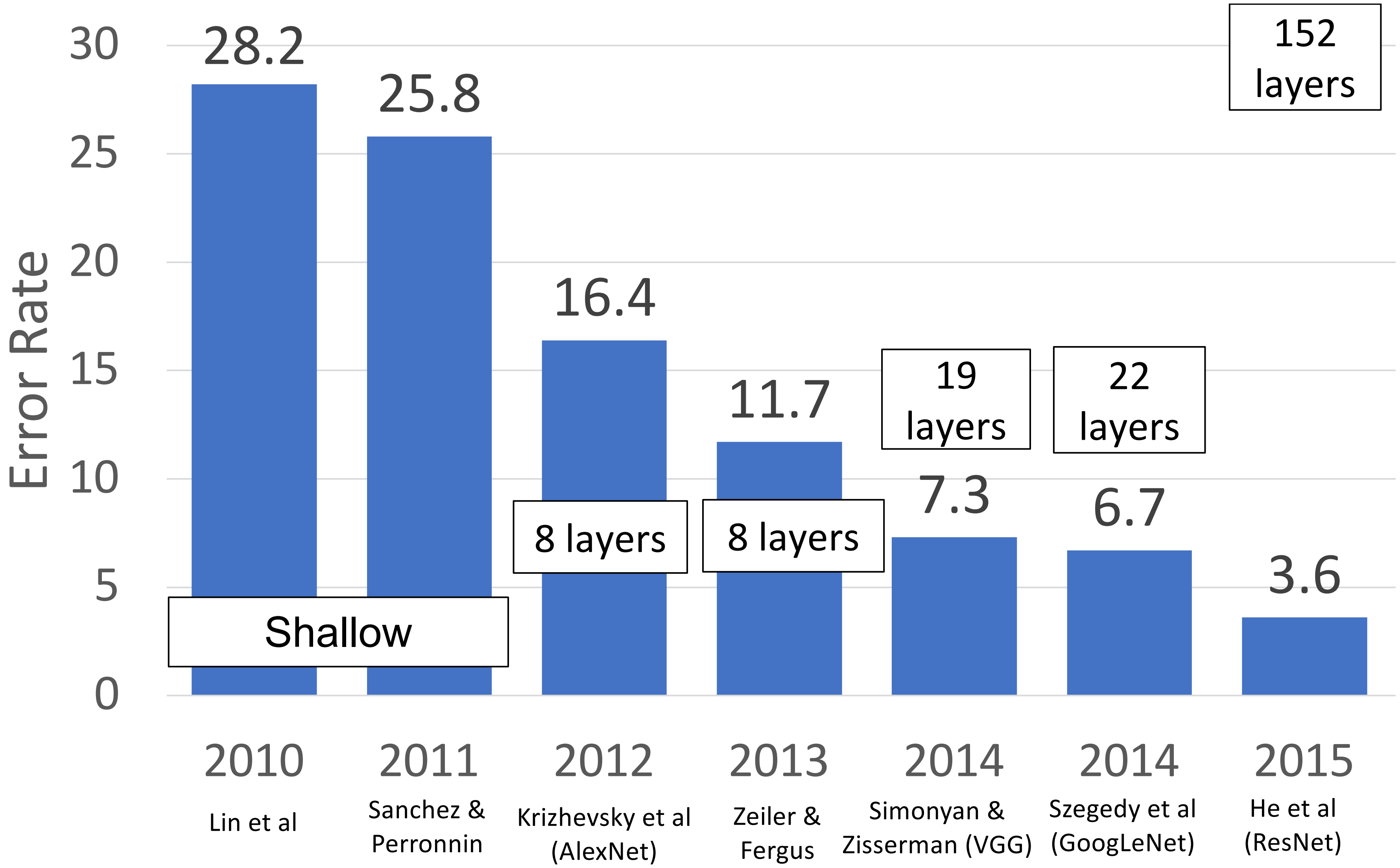
Residual Learning



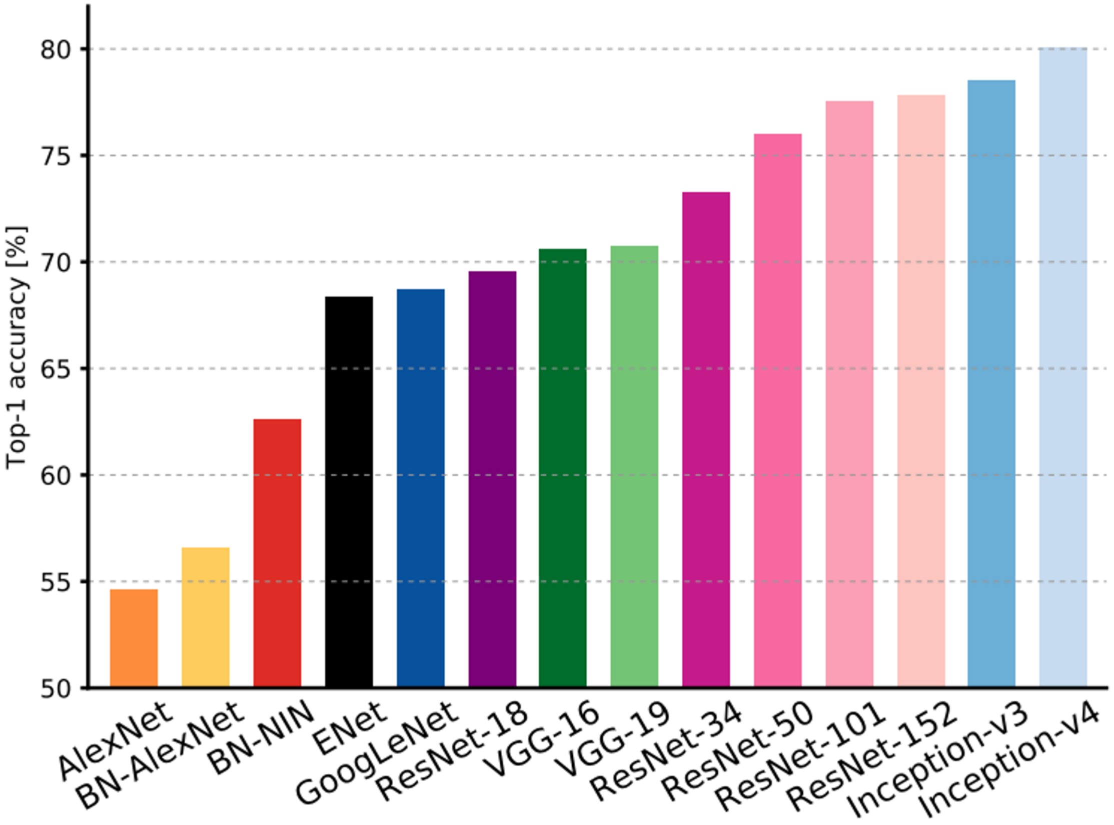
- The loss surface of a 56-layer net using the CIFAR-10 dataset, both without (left) and with (right) residual connections.

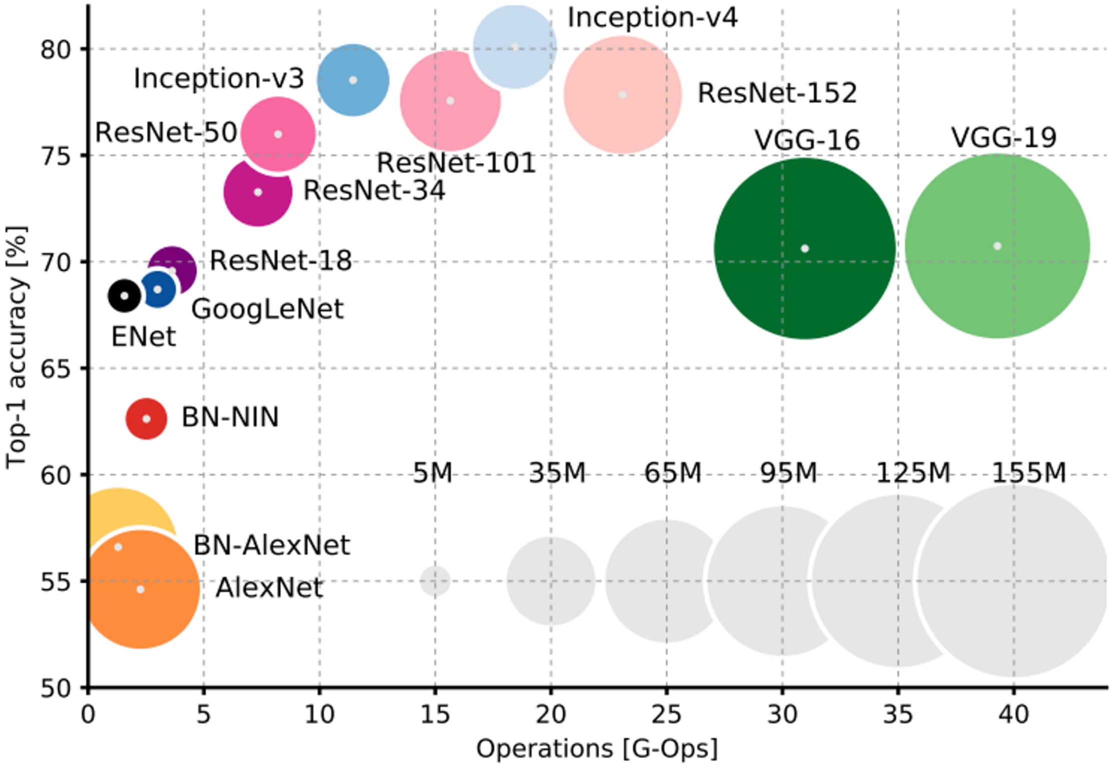
Hao Li et al., "Visualizing the Loss Landscape of Neural Nets". ICLR 2018

ImageNet Classification Challenge



Comparing Complexity

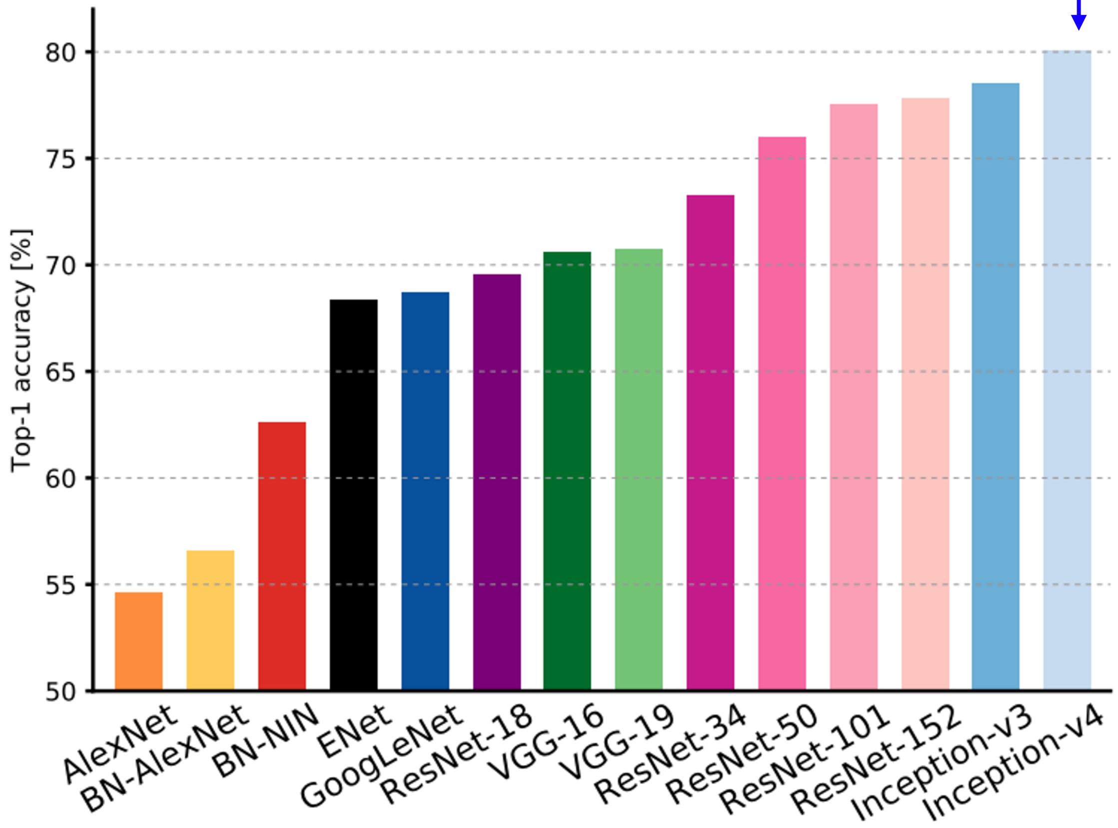


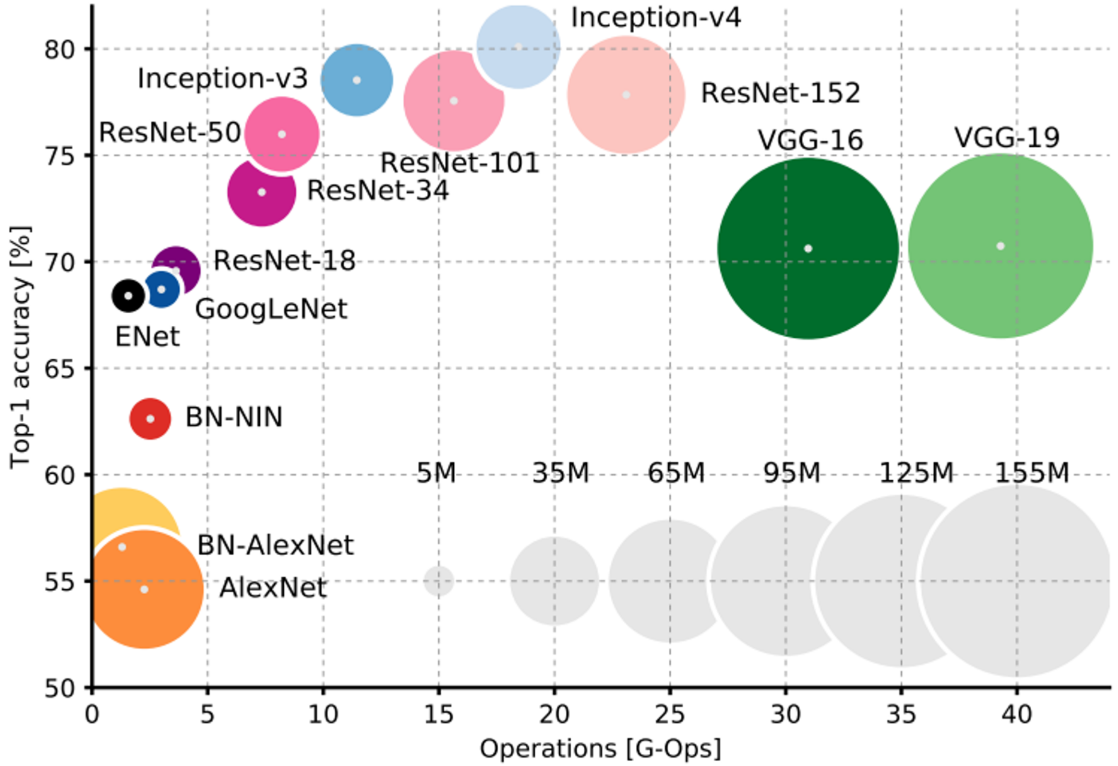


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

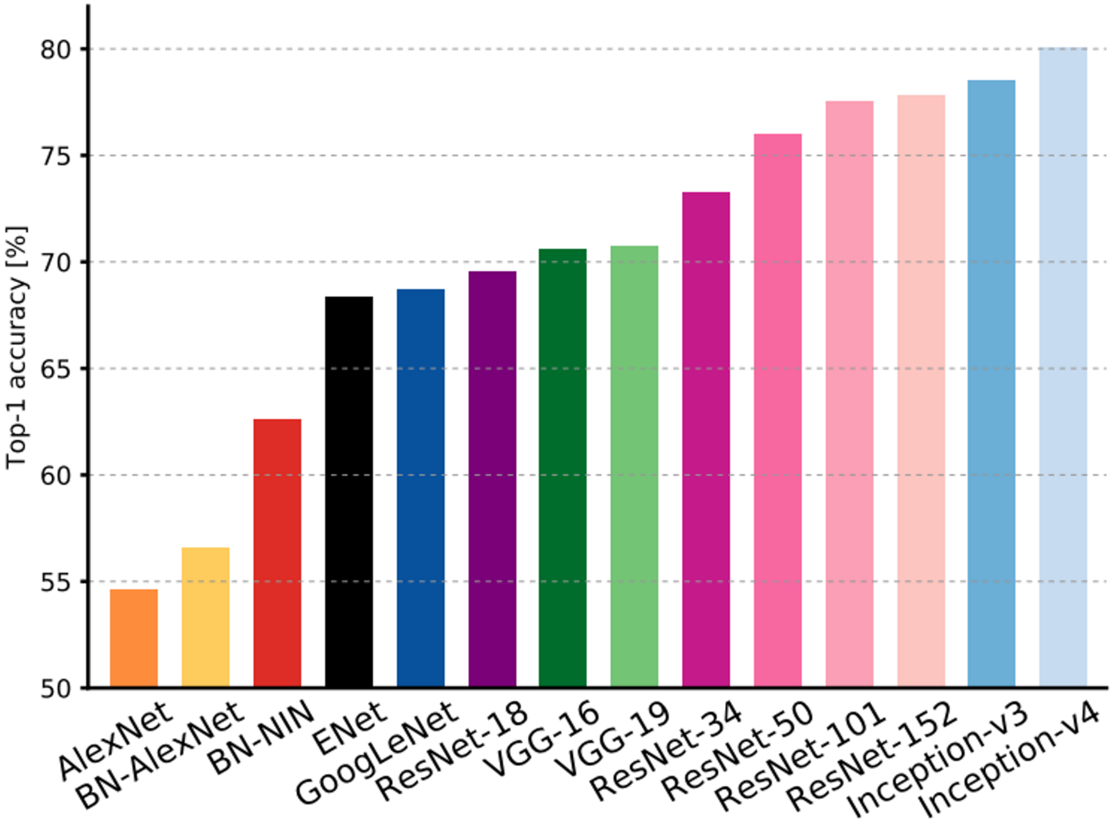
Inception-v4: Resnet + Inception!

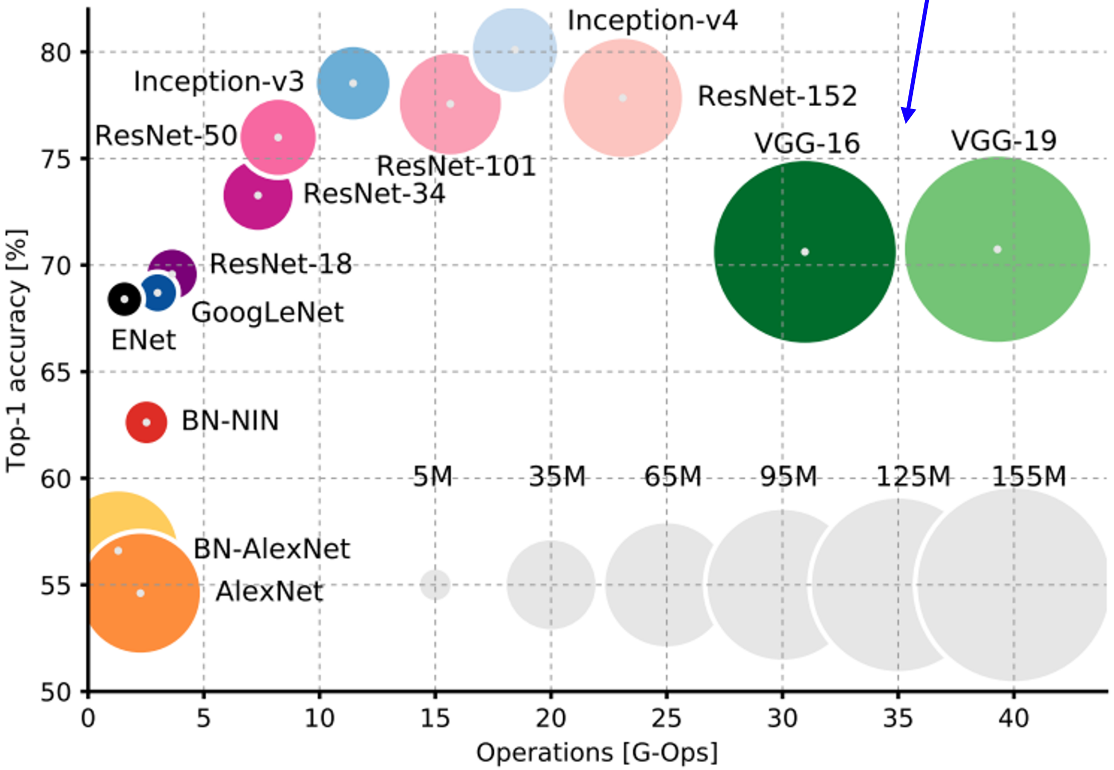




Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

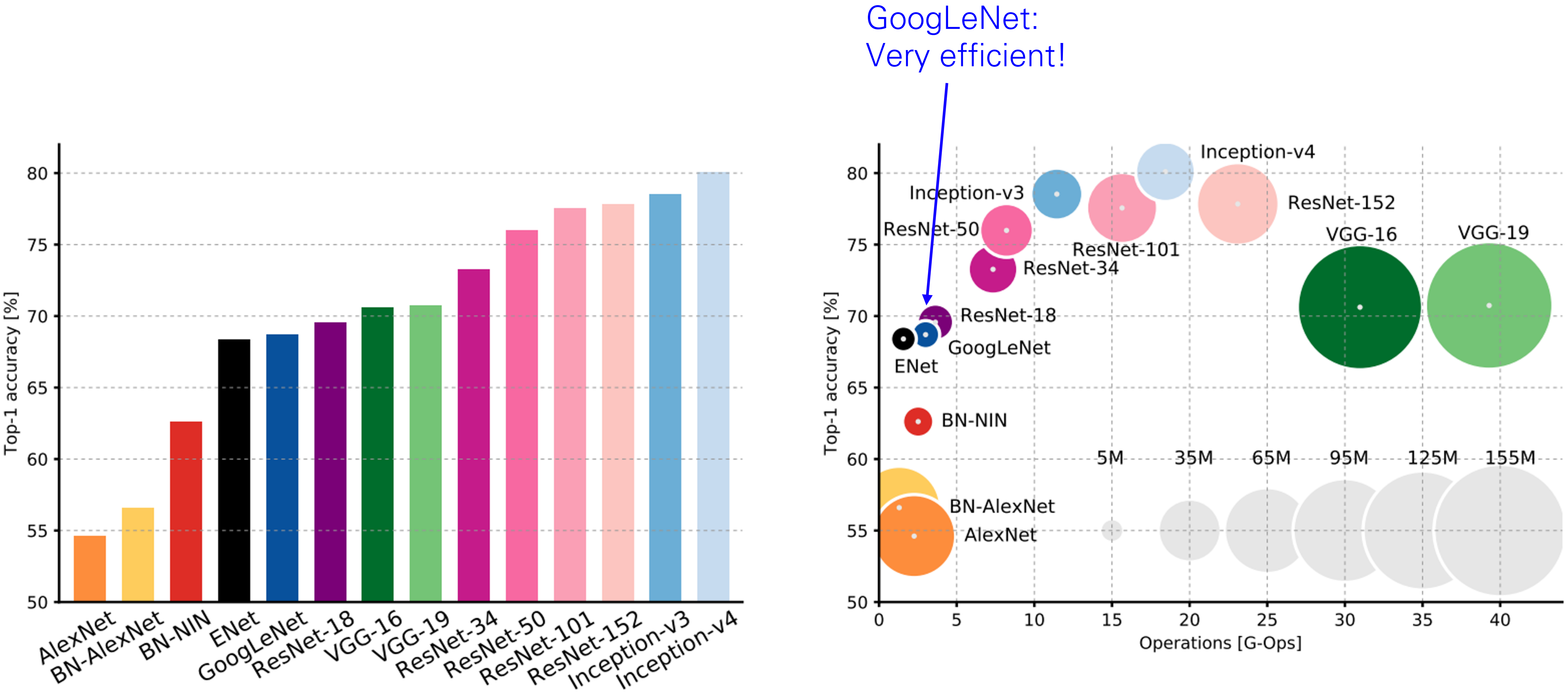




VGG: Highest memory, most operations

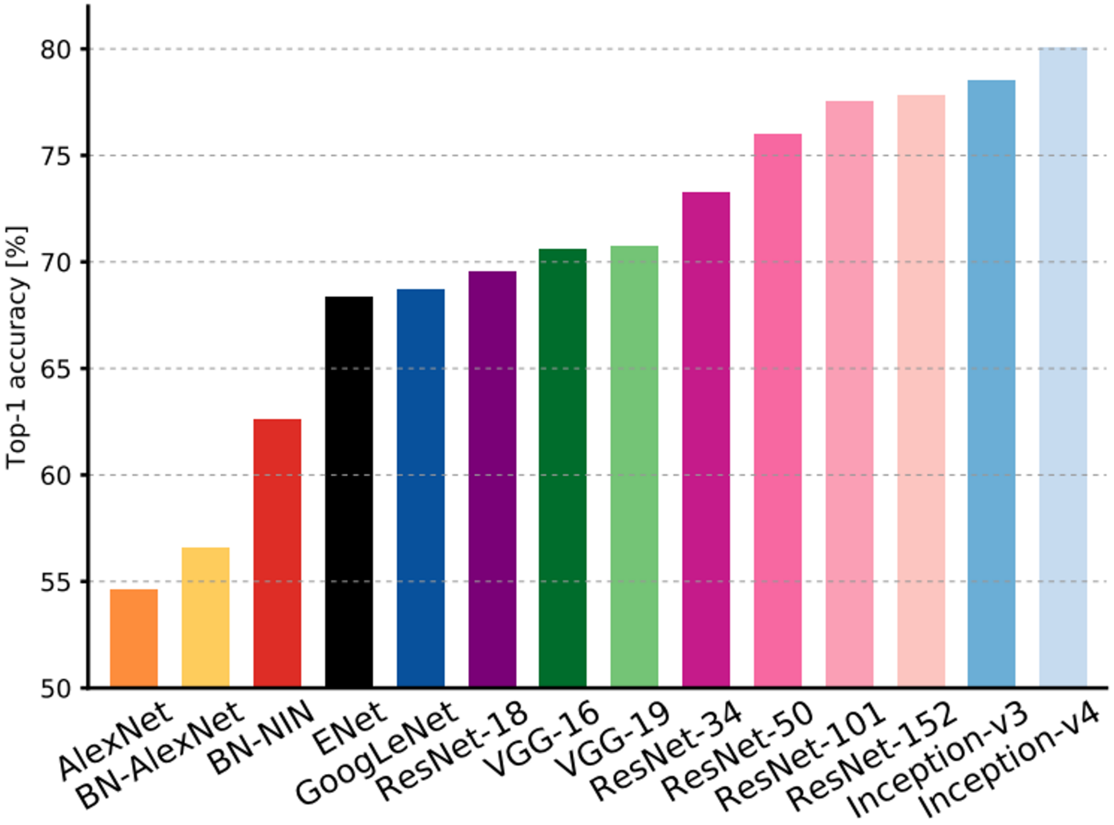
Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

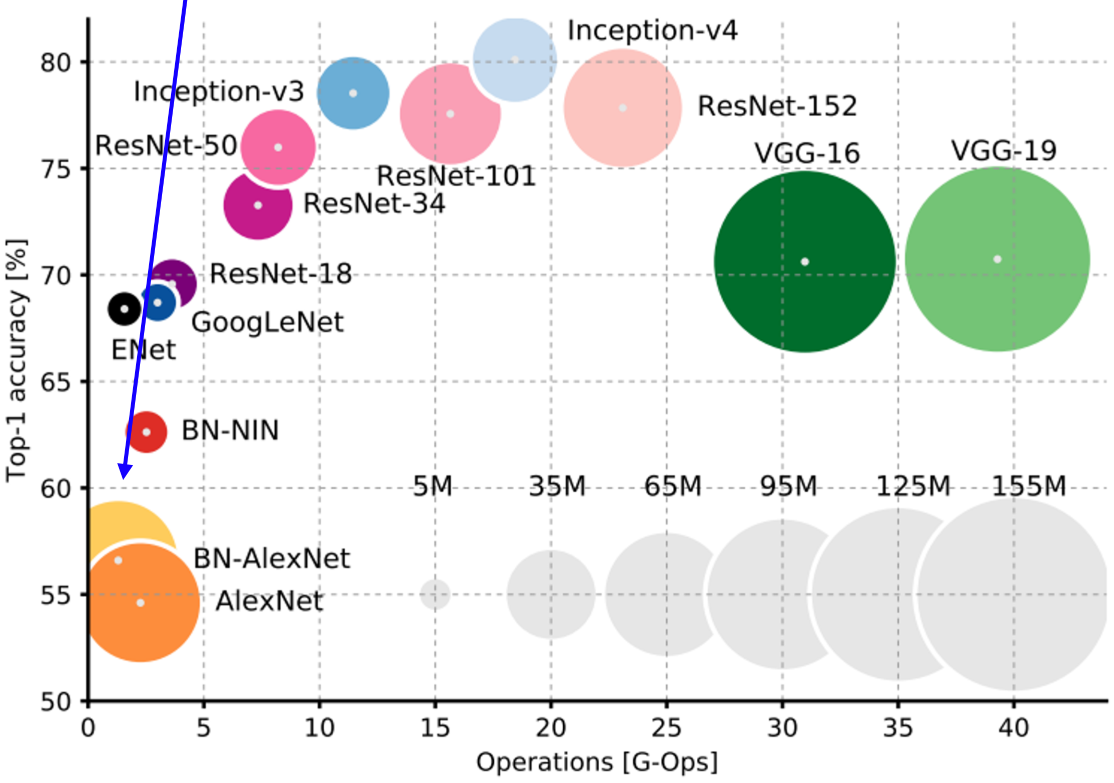


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

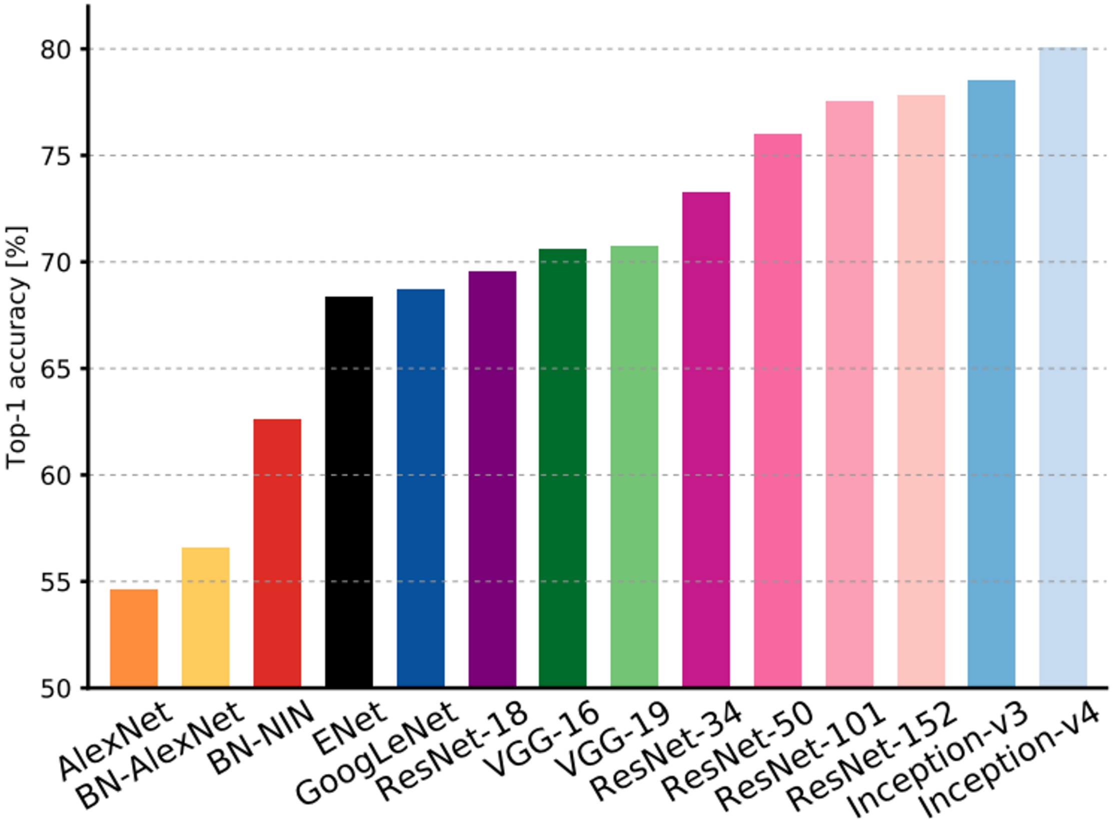


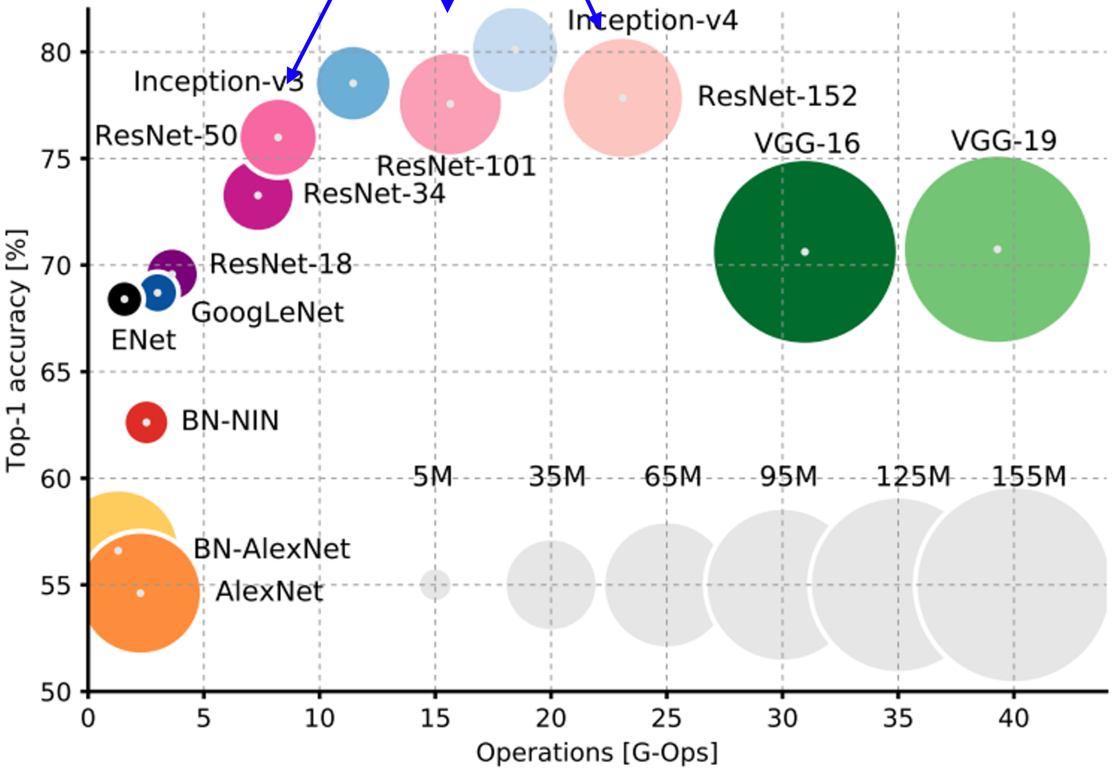
AlexNet: Low compute, lots of parameters



Comparing Complexity

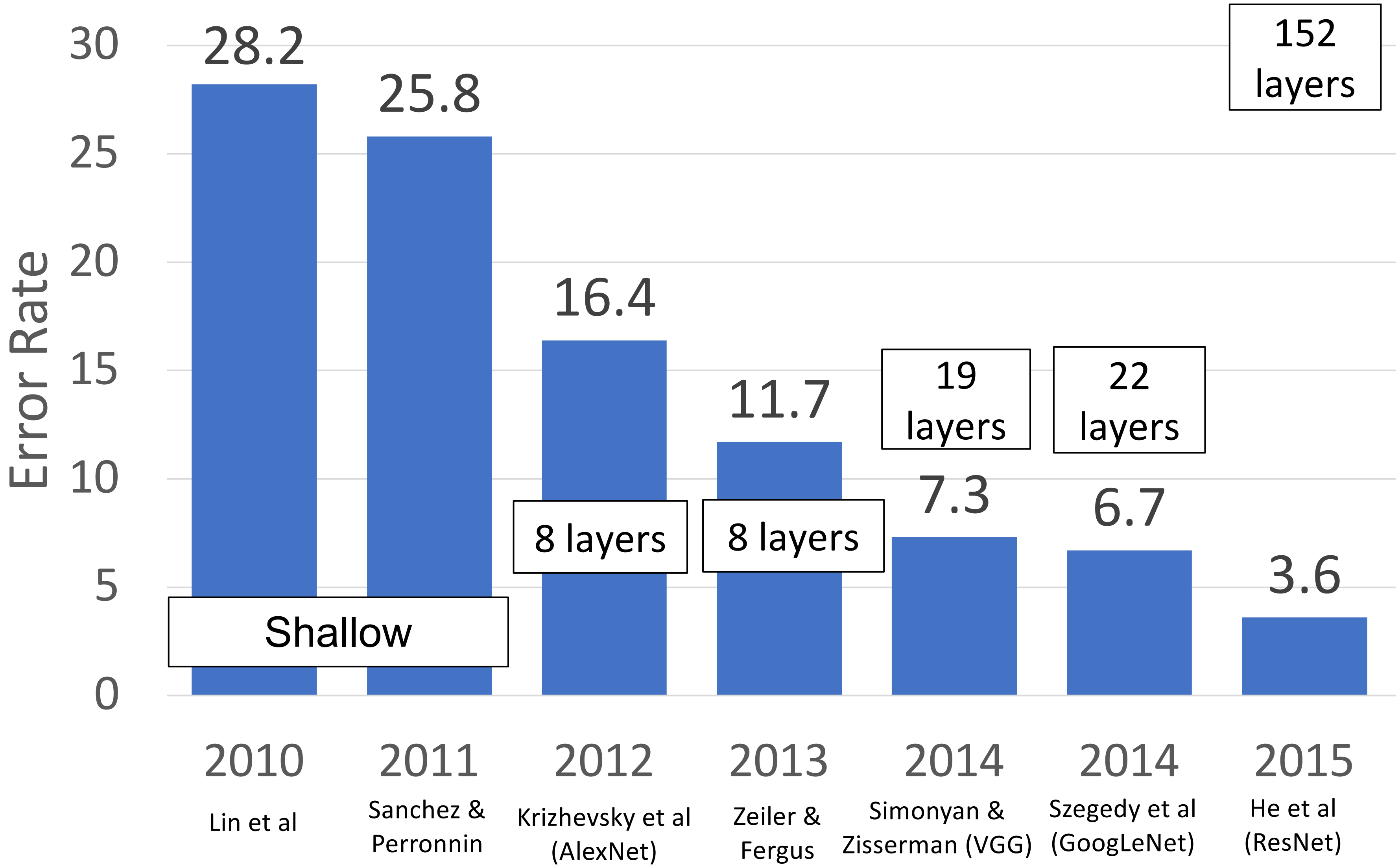
ResNet: Simple design,
moderate efficiency,
high accuracy





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

ImageNet Classification Challenge

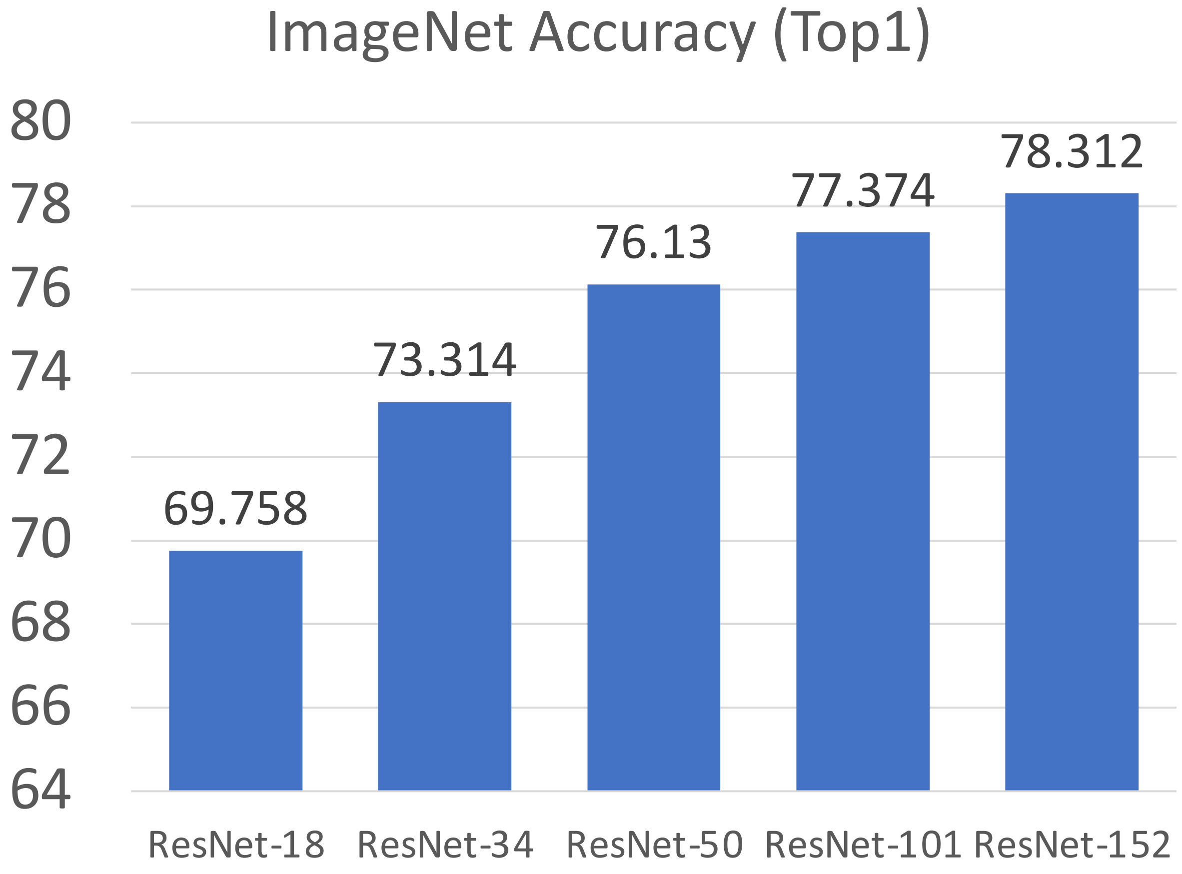


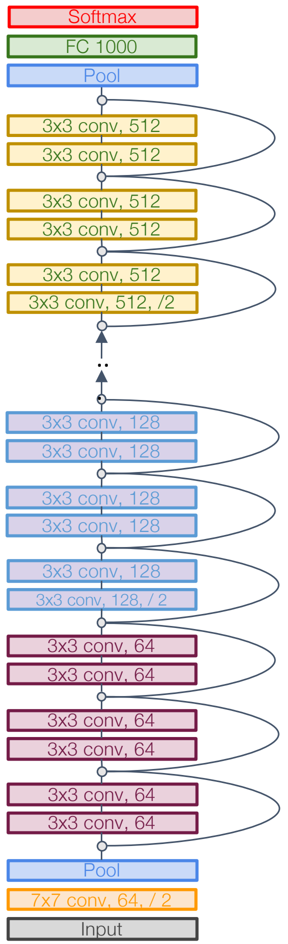
Today:
More recent CNN architectures

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?

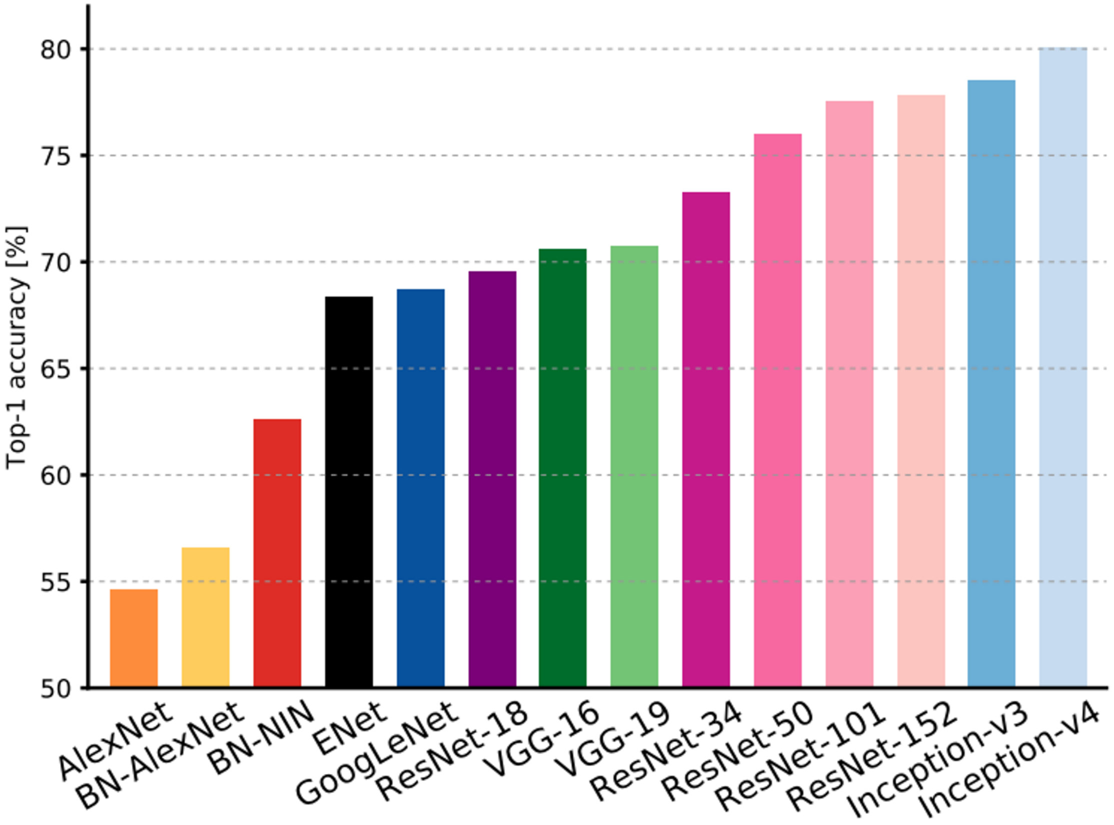
Floating Point Operations (FLOPs): How many arithmetic operations does it take to compute the forward pass of the model?

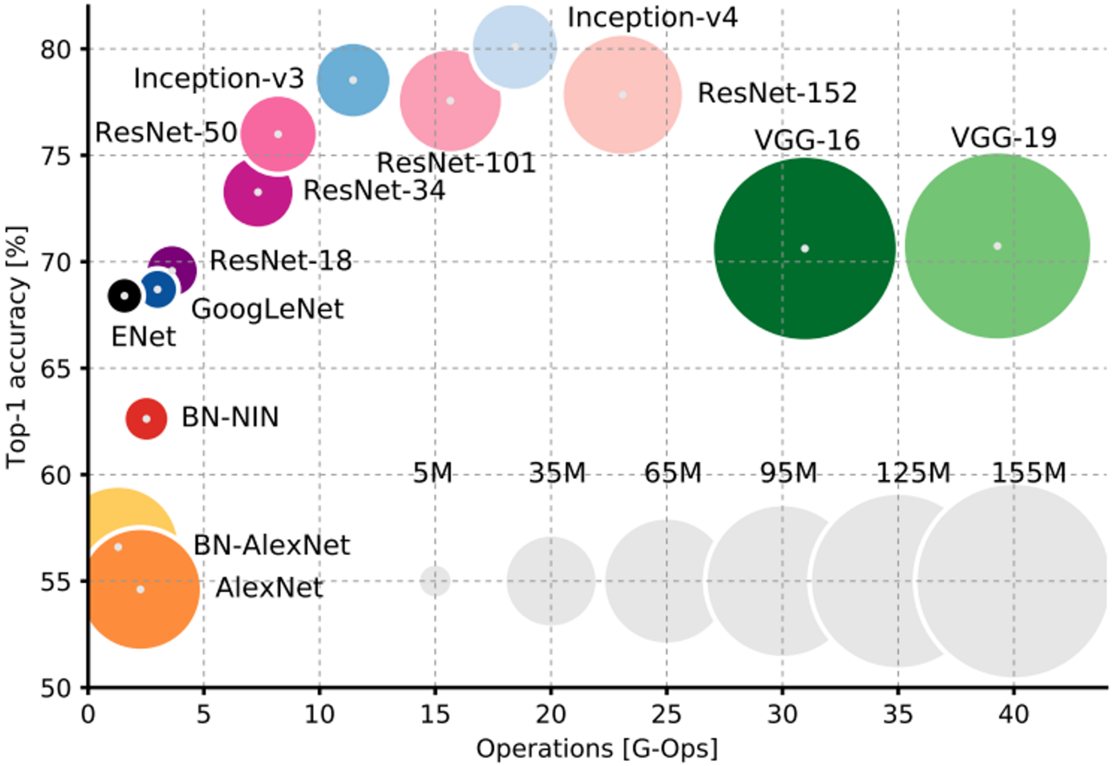
Watch out, lots of subtlety here:

- 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



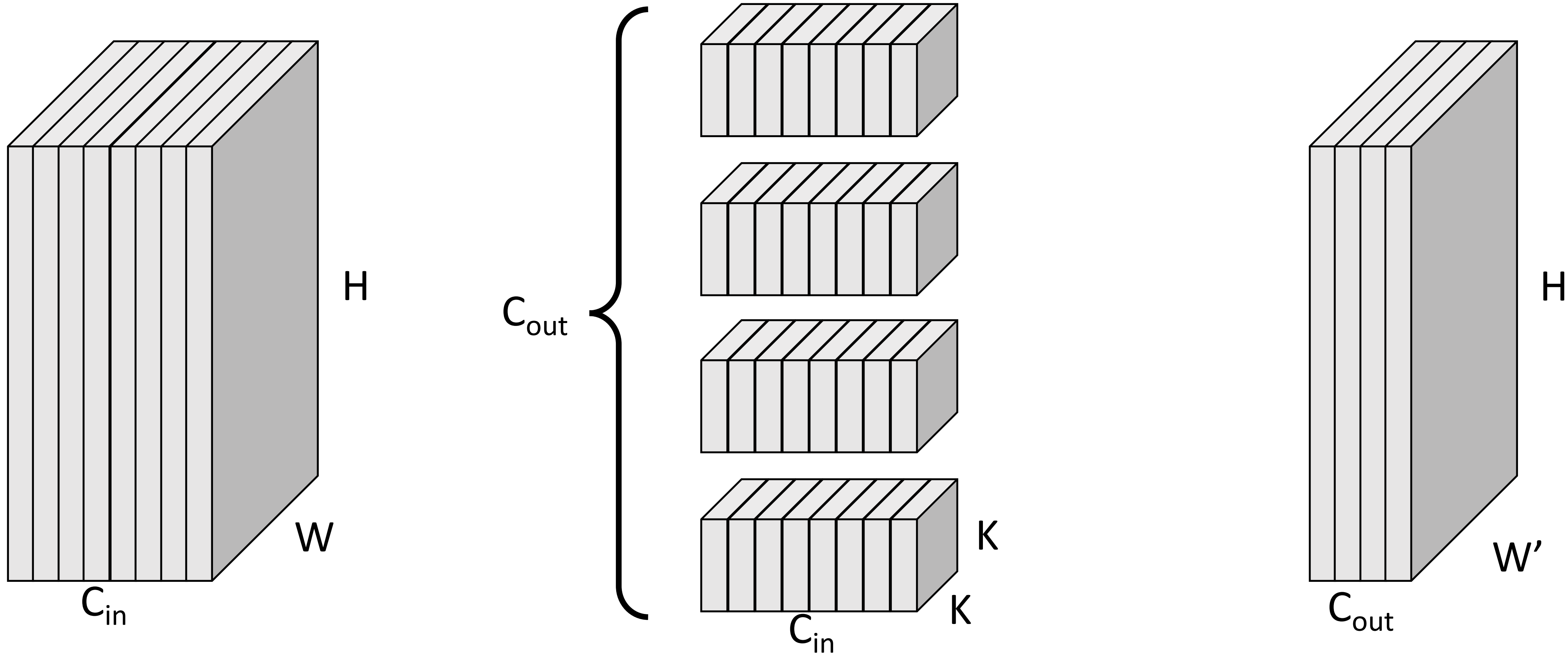


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

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

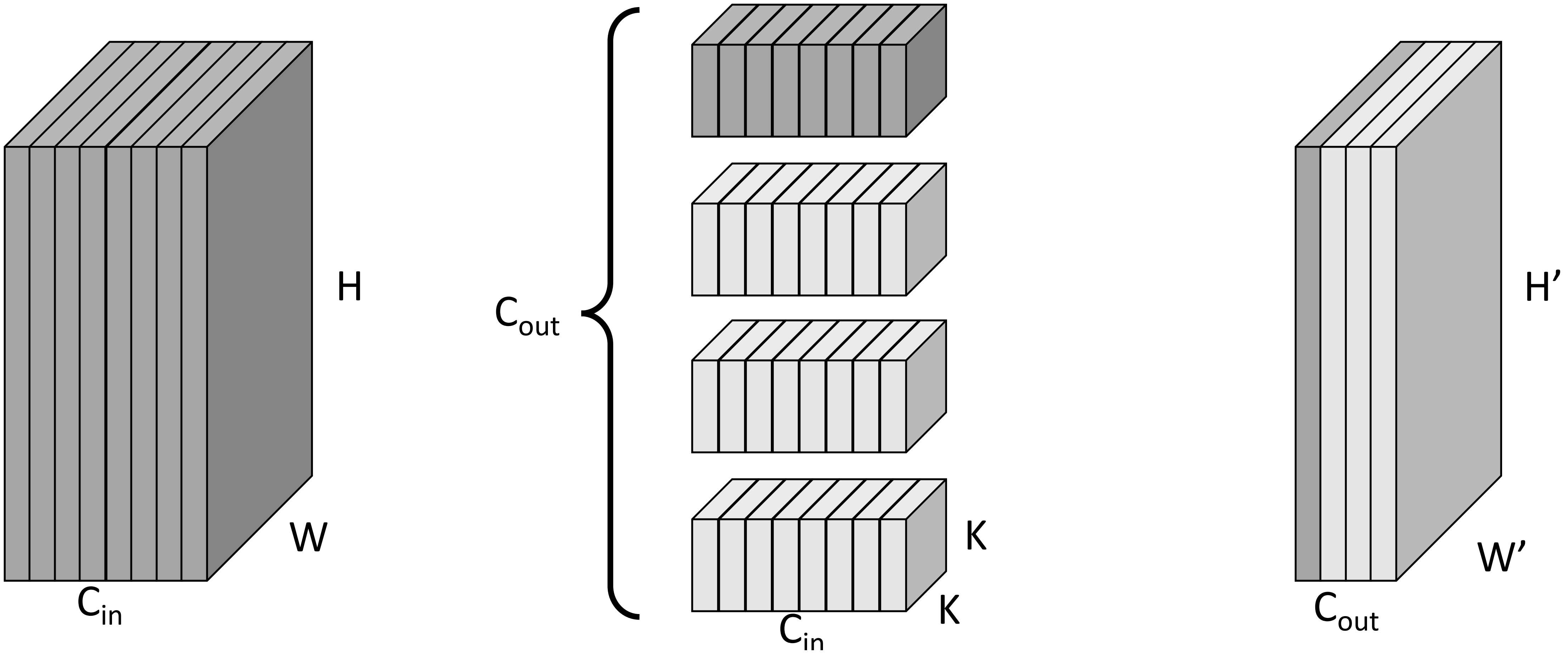
Weights: C_{out} x C_{in} x K x K

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} x H x W

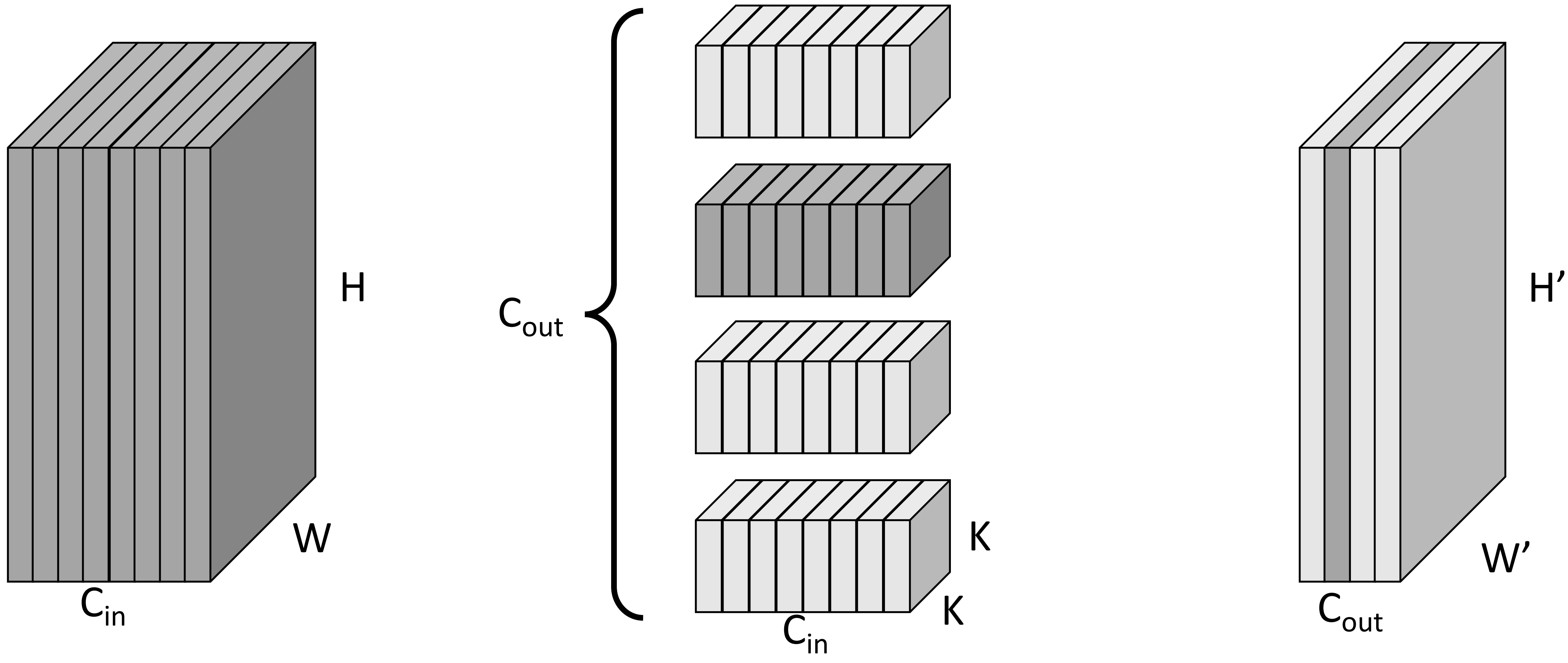
Weights: C_{out} x C_{in} x K x K

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

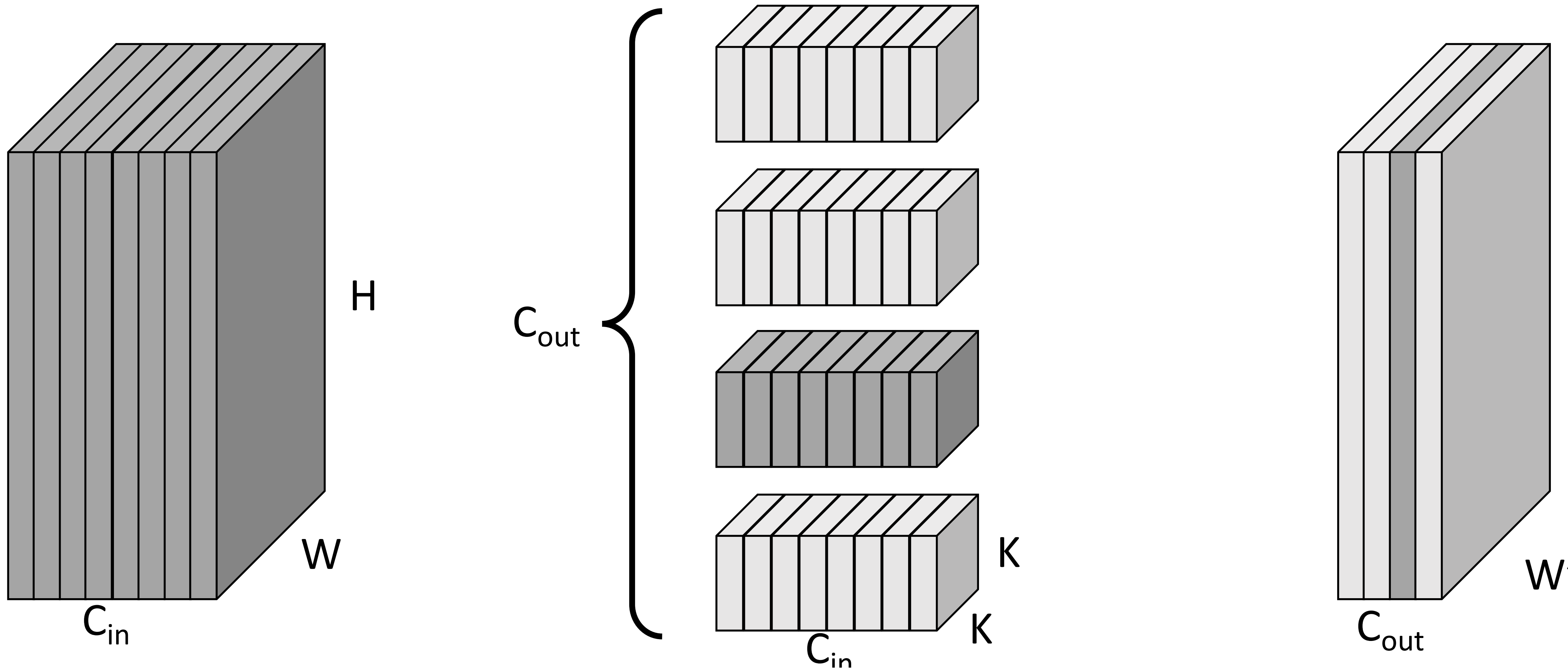
Weights: C_{out} x C_{in} x K x K

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

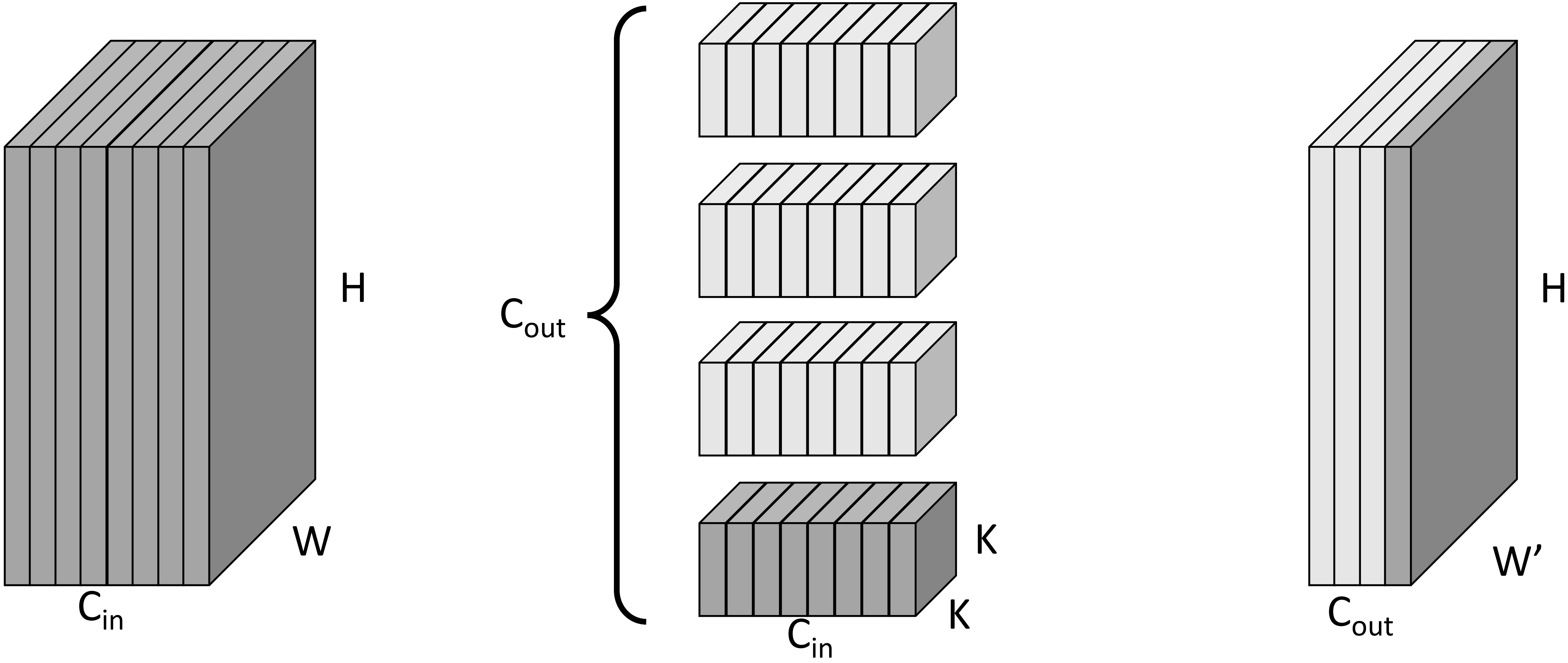


Input: C_{in} x H x W   Weights: C_{out} x C_{in} x K x K   Output: C_{out} x H' x W'

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

Grouped Convolution

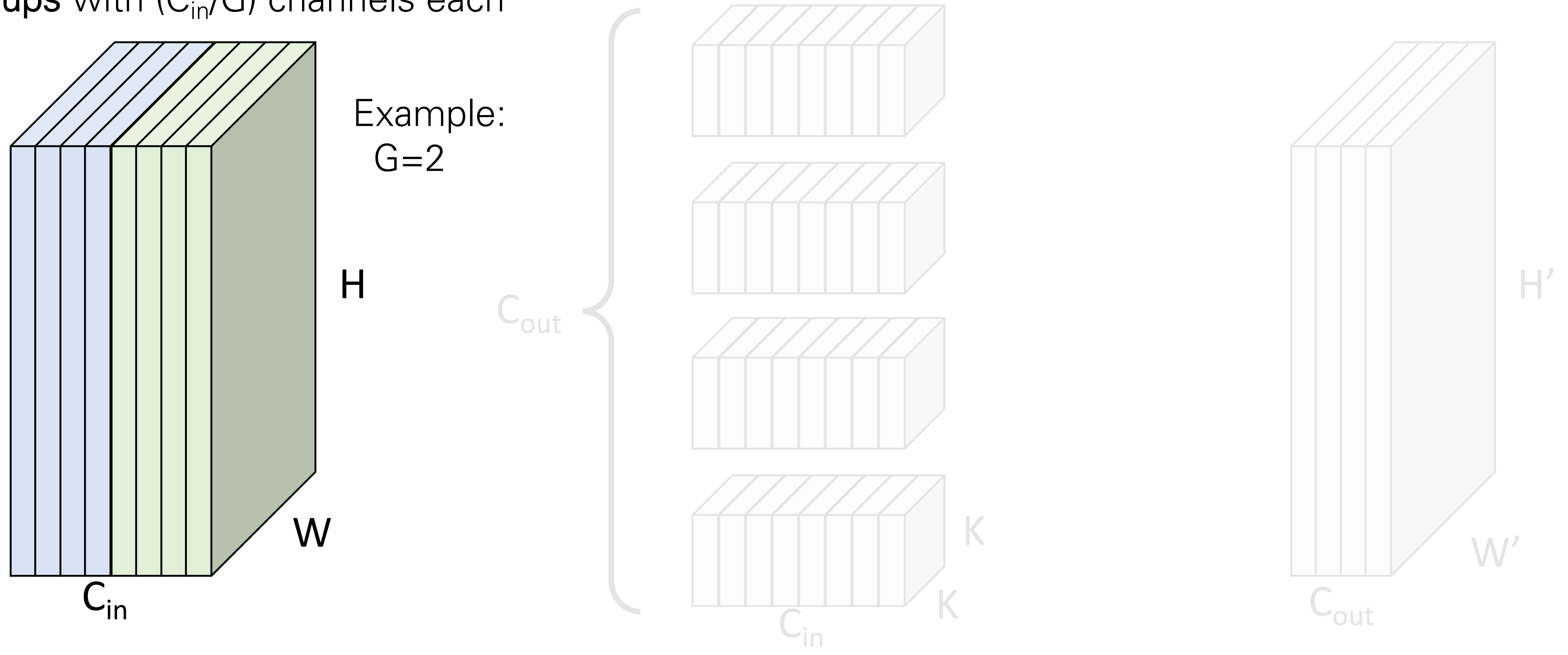
Input: C_{in} x H x W

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

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

Grouped Convolution

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



Input: C_{in} x H x W

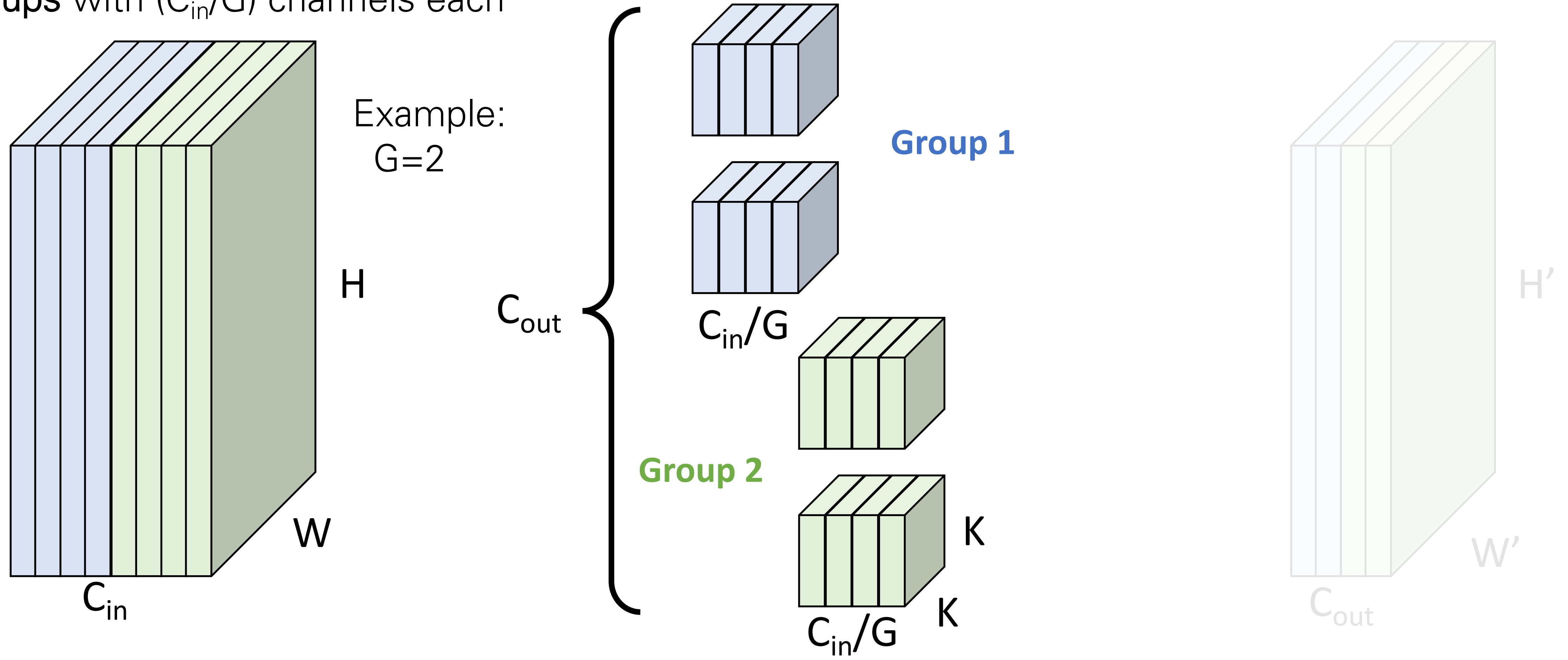
Weights: C_{out} x C_{in} x K x K

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

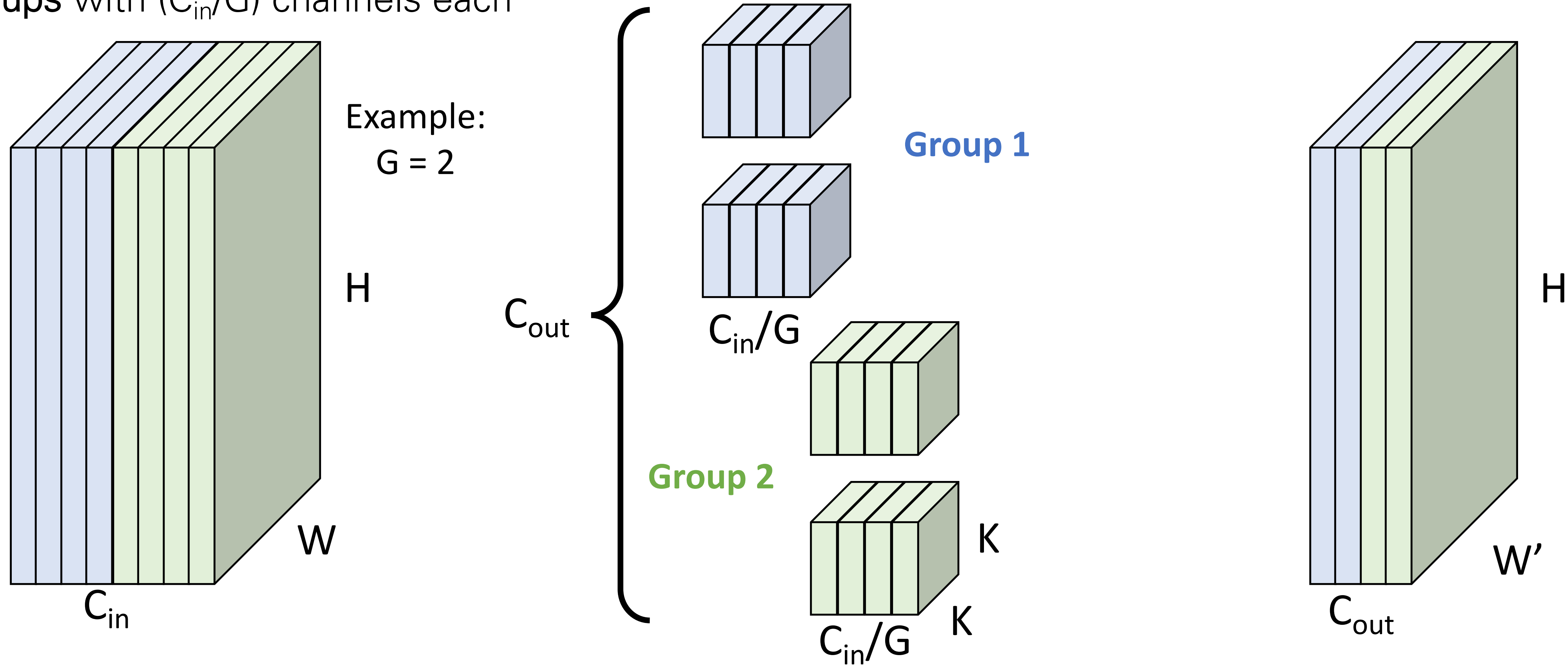
Output: C_{out} x H' x W'

Grouped Convolution

Divide 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

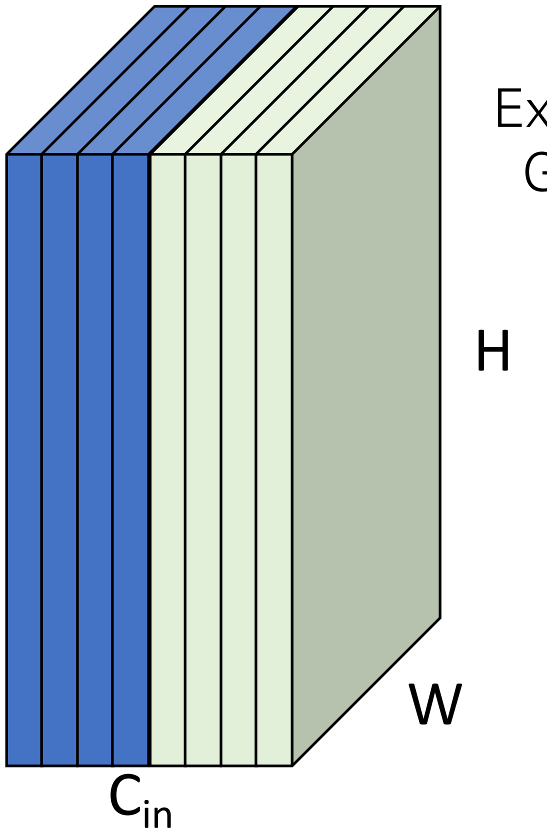
Weights: C_{out} x (C_{in} /G) x K x K

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

Group Convolution

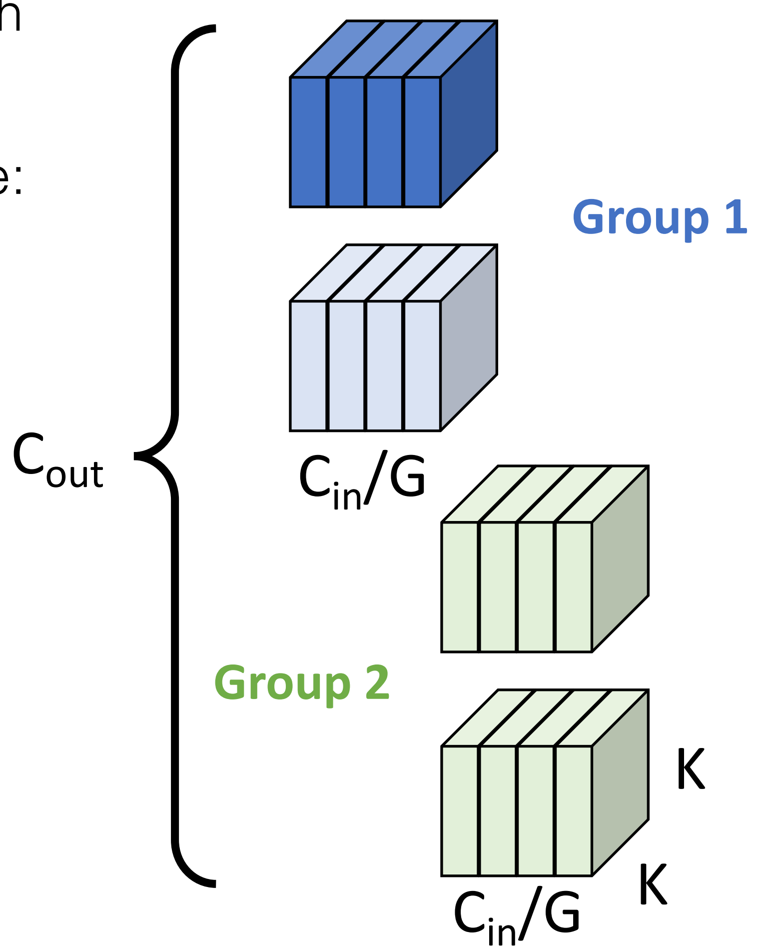
Divide channels of input into G **groups** with (C_{in}/G) channels each

Example: G=2



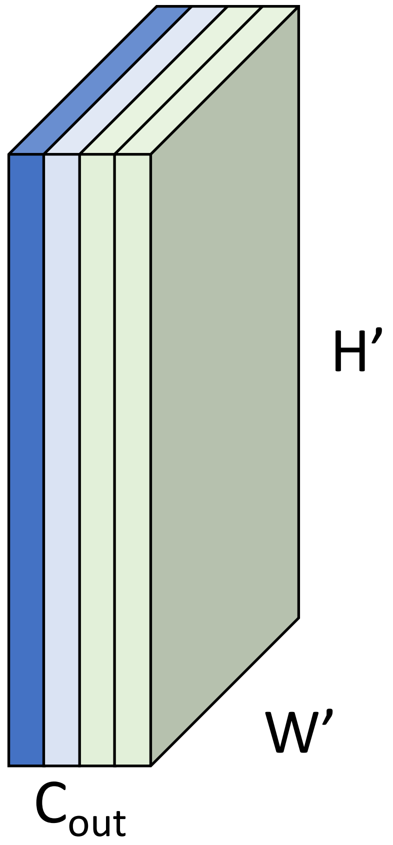
Input: C_{in} x H x W

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



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

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



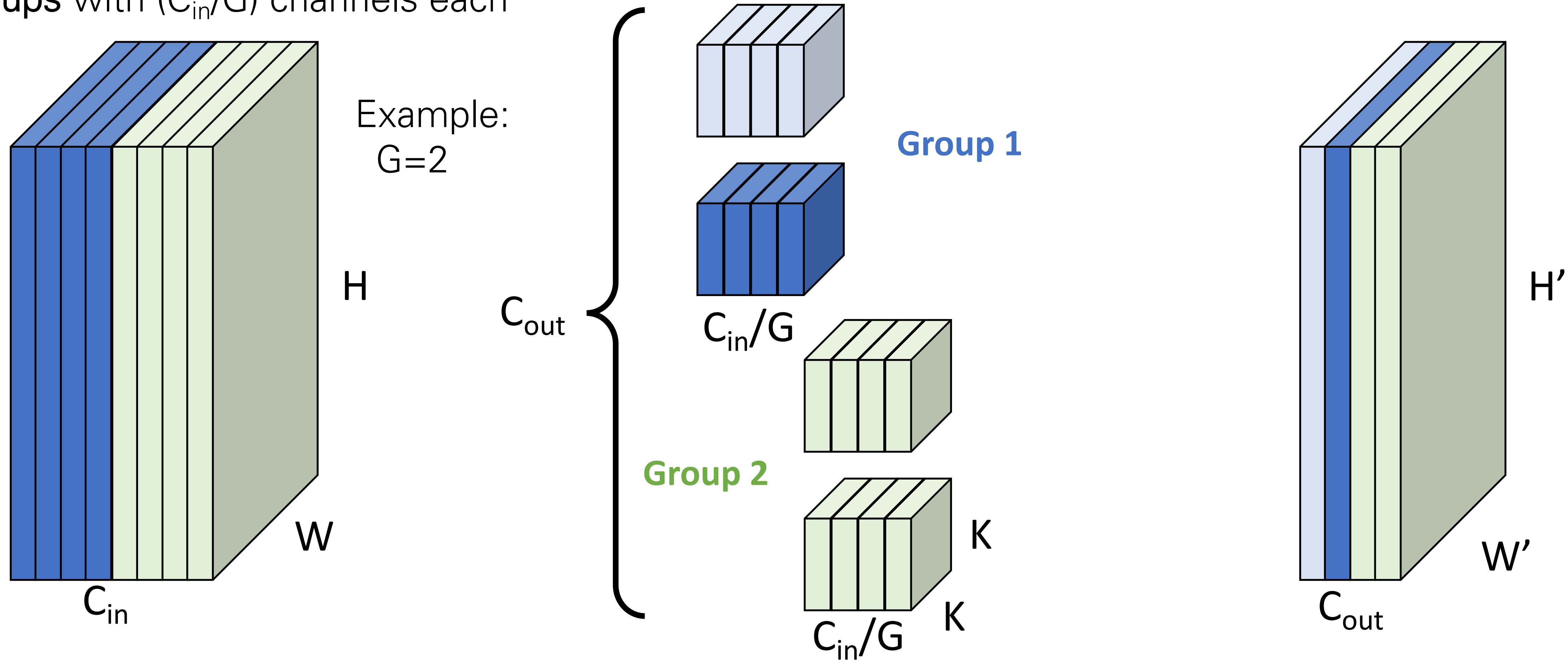
Output: C_{out} x H' x W'

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

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

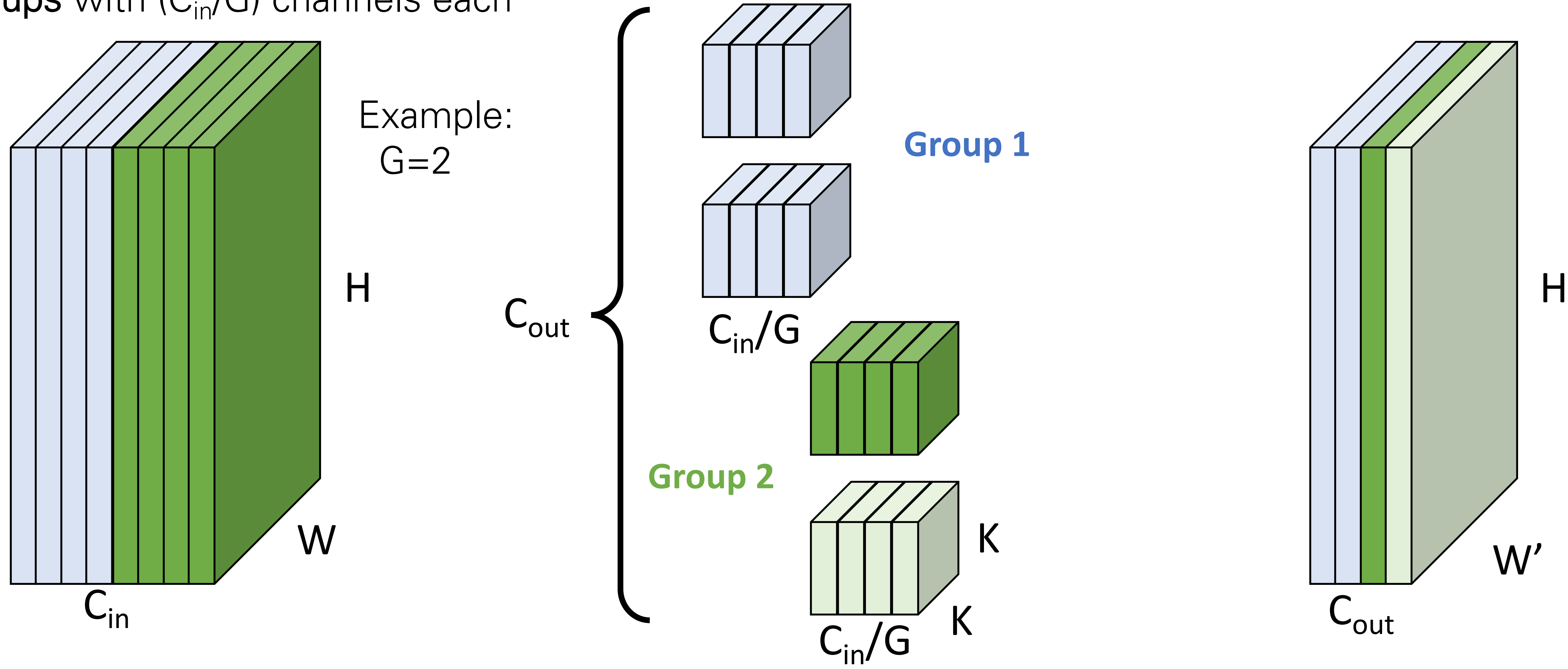
Output: $C_{out} \times H' \times W'$

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} /G) x K x K

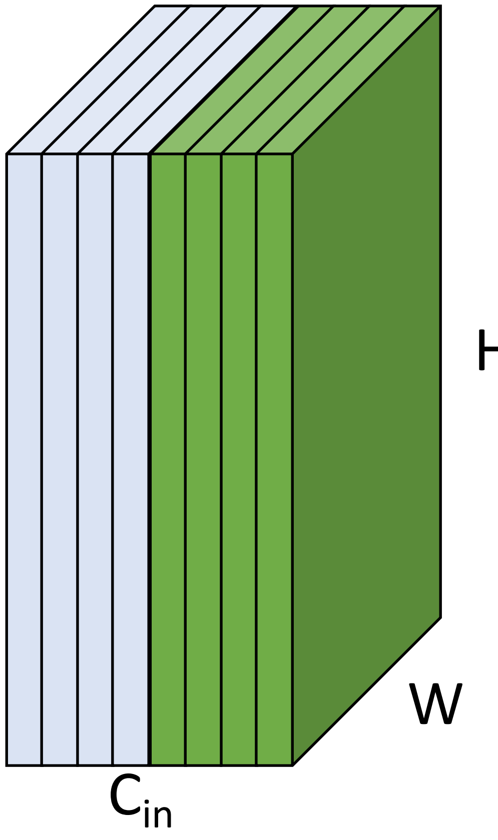
Output: C_{out} x H' x W'

Group Convolution

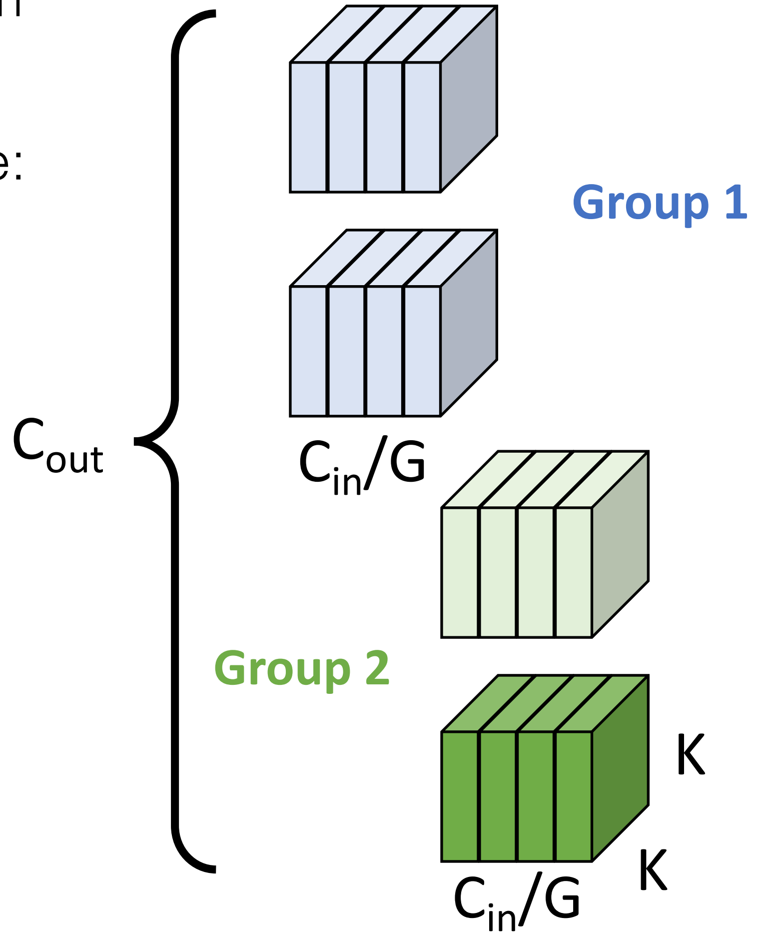
Divide filters into G groups; each group looks at a **subset** of input channels

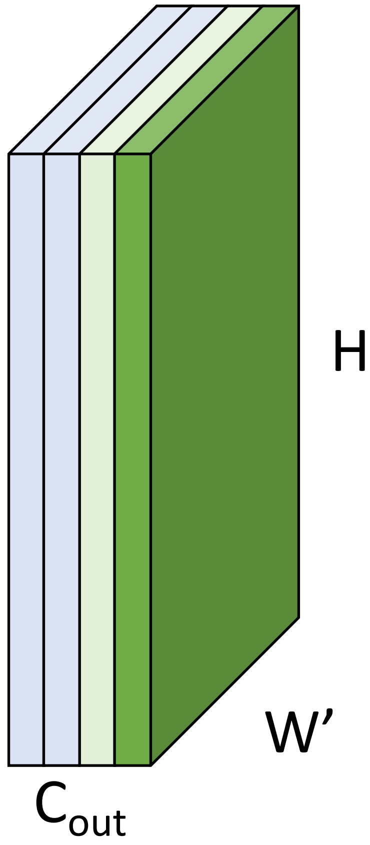
Each plane of the output depends on one filter and a **subset** of the input channels

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



Example:
G=2





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

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

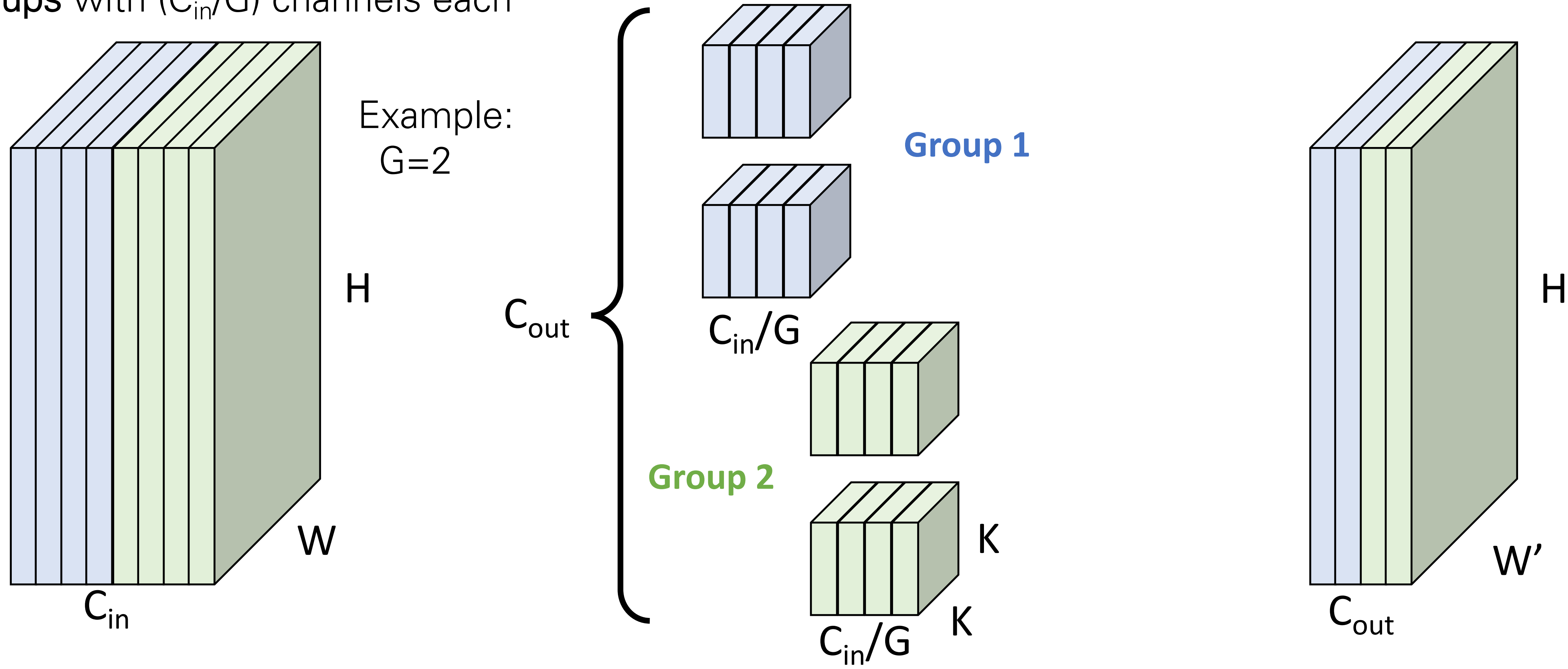
Output: $C_{out} \times H' \times W'$

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} /G) x K x K

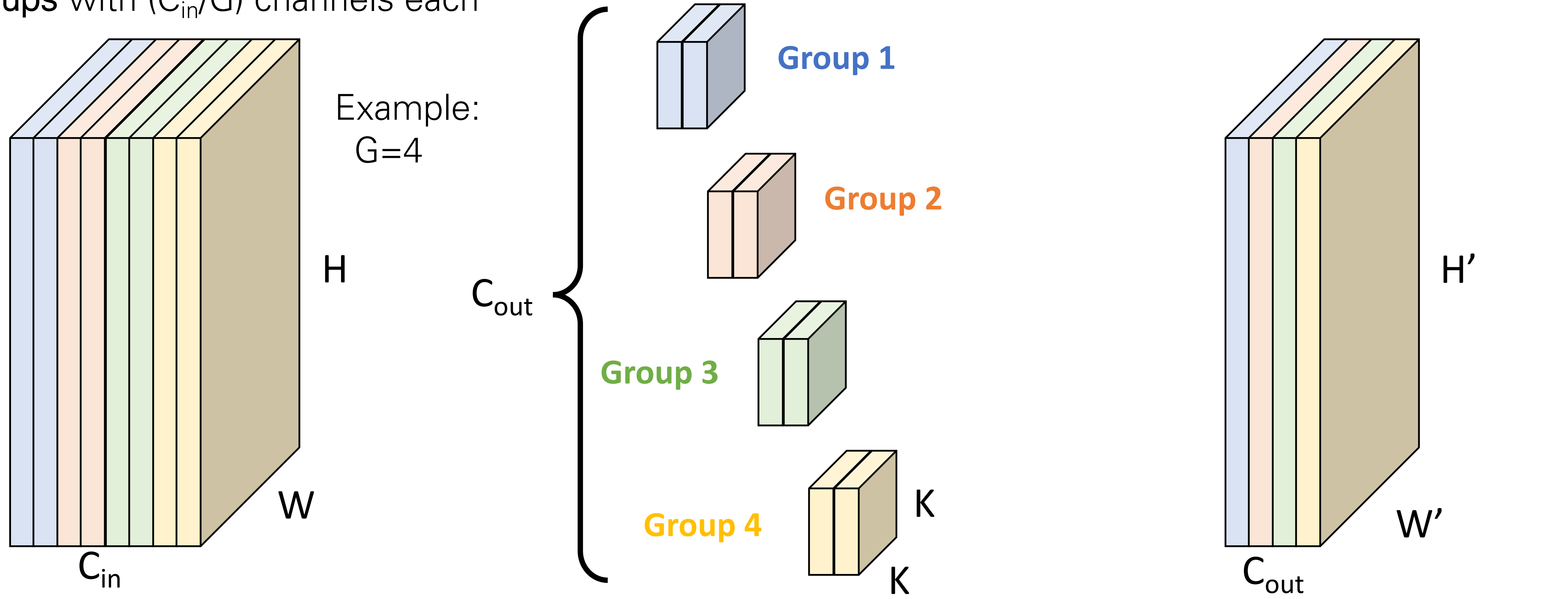
Output: C_{out} x H' x W'

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



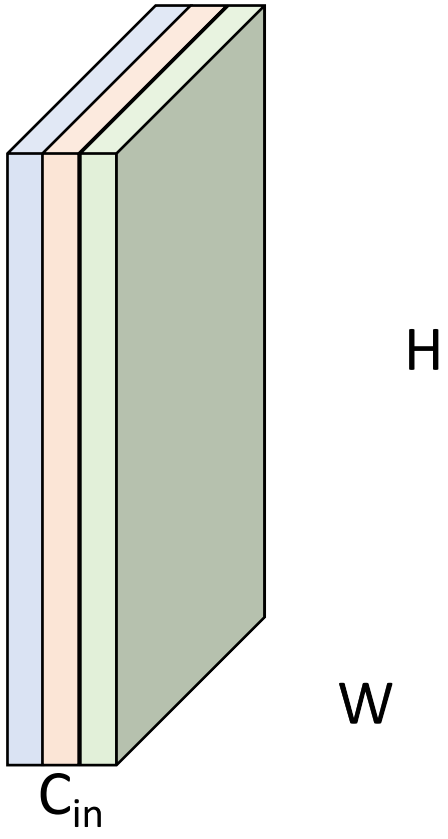
Input: C_{in} x H x W

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

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

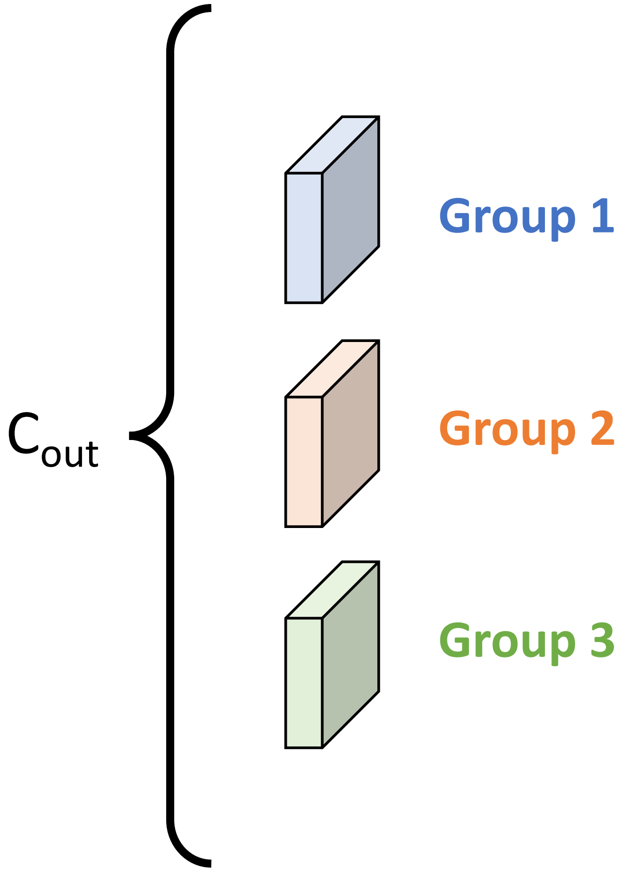
Special Case: Depthwise Convolution

Number of groups equals number of input channels



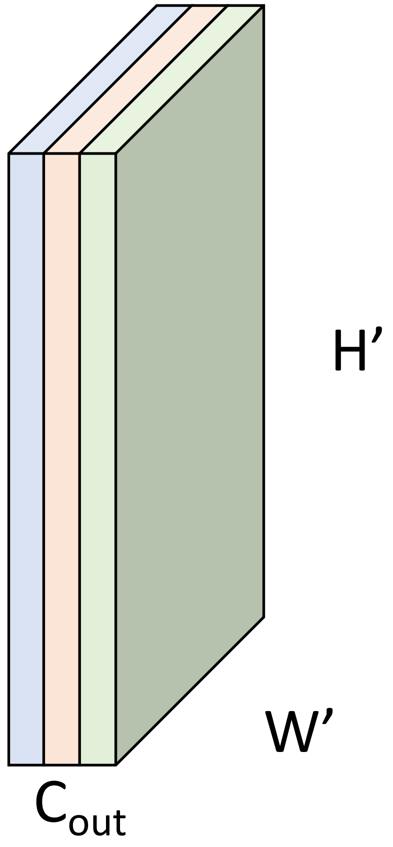
Input: C_{in} x H x W

Common to also set C_{out} = G



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

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



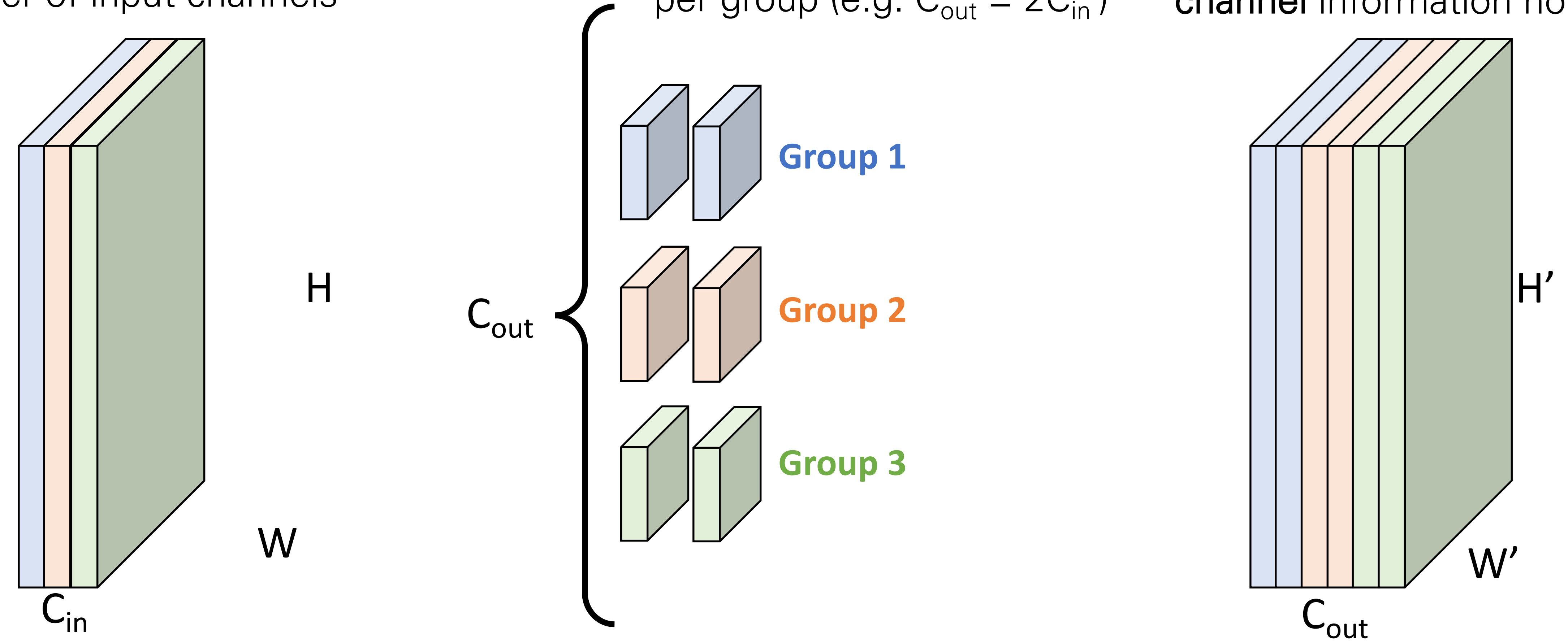
Output: C_{out} x H’ x W’

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

Weights: $C_{out} \times 1 \times K \times K$

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

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

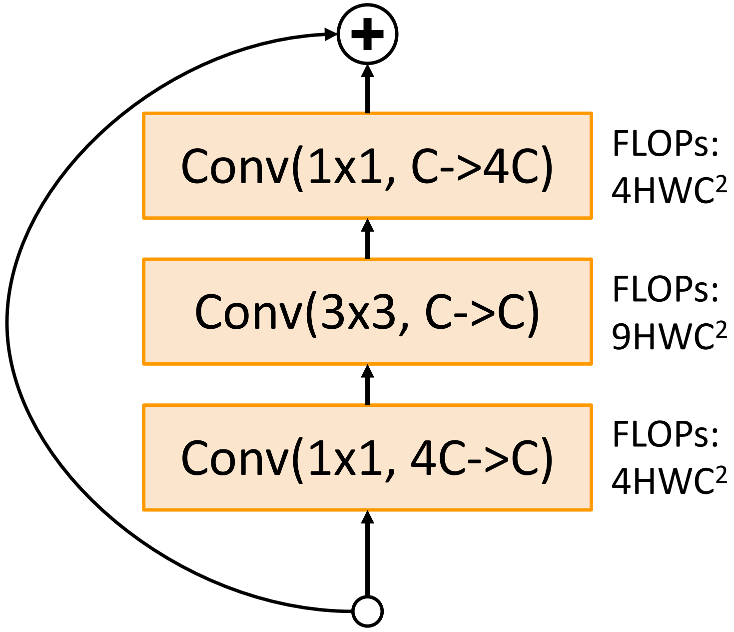
Standard Convolution (groups=1)

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

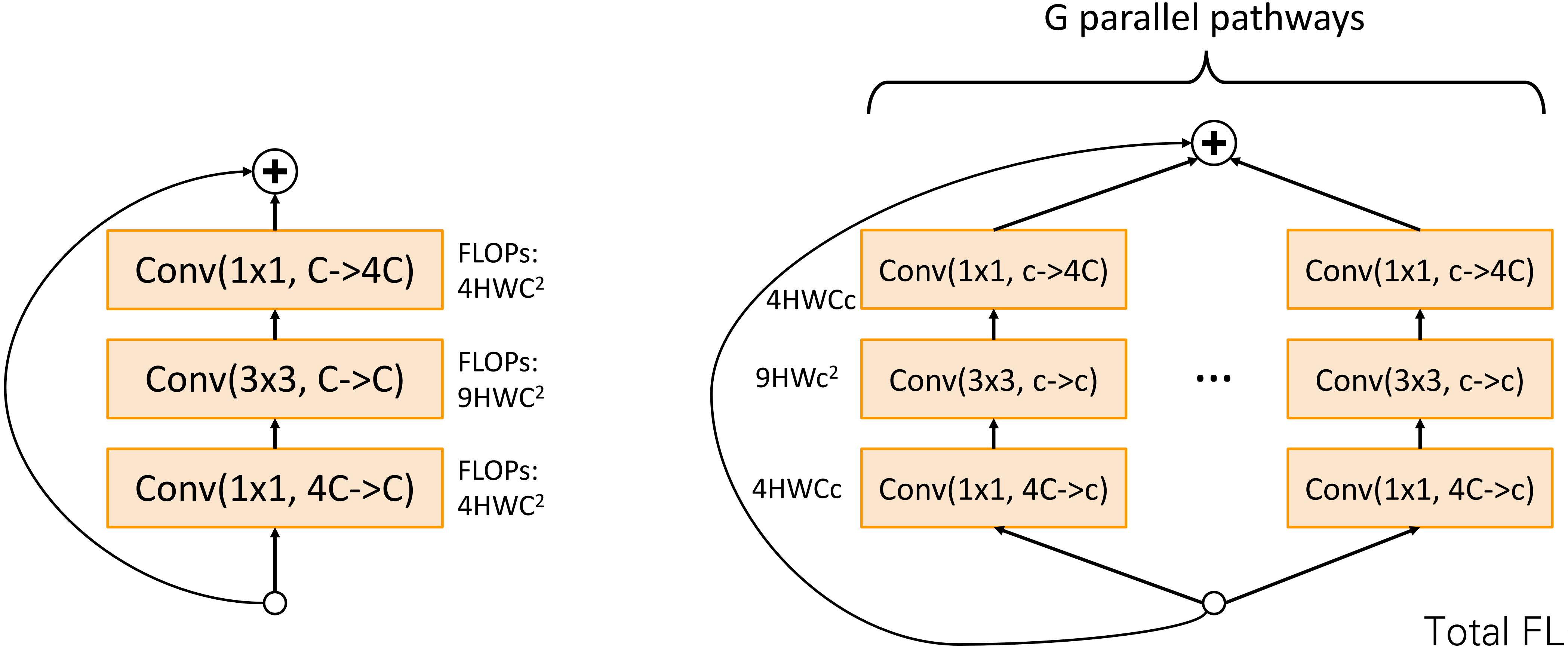


"Bottleneck"
Residual block

Total FLOPs:
$17HWC^2$

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

Improving ResNets: ResNeXt



"Bottleneck"
Residual block

Total FLOPs:
$17HWC^2$

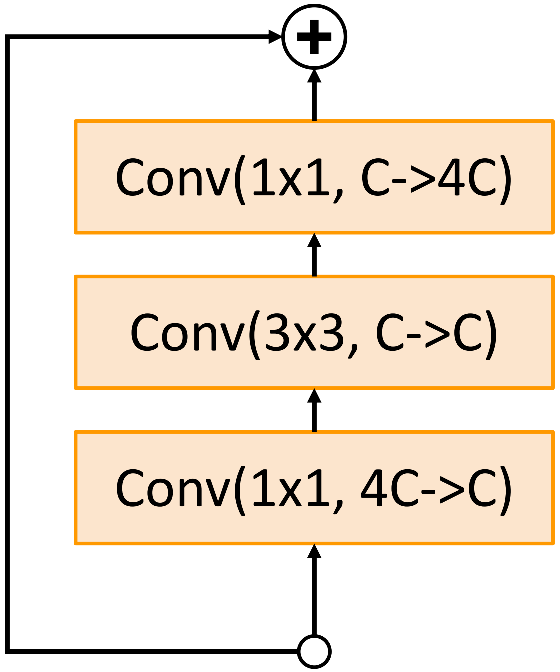
Same FLOPs when
$9Gc^2 + 8GCc - 17C^2 = 0$

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

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)

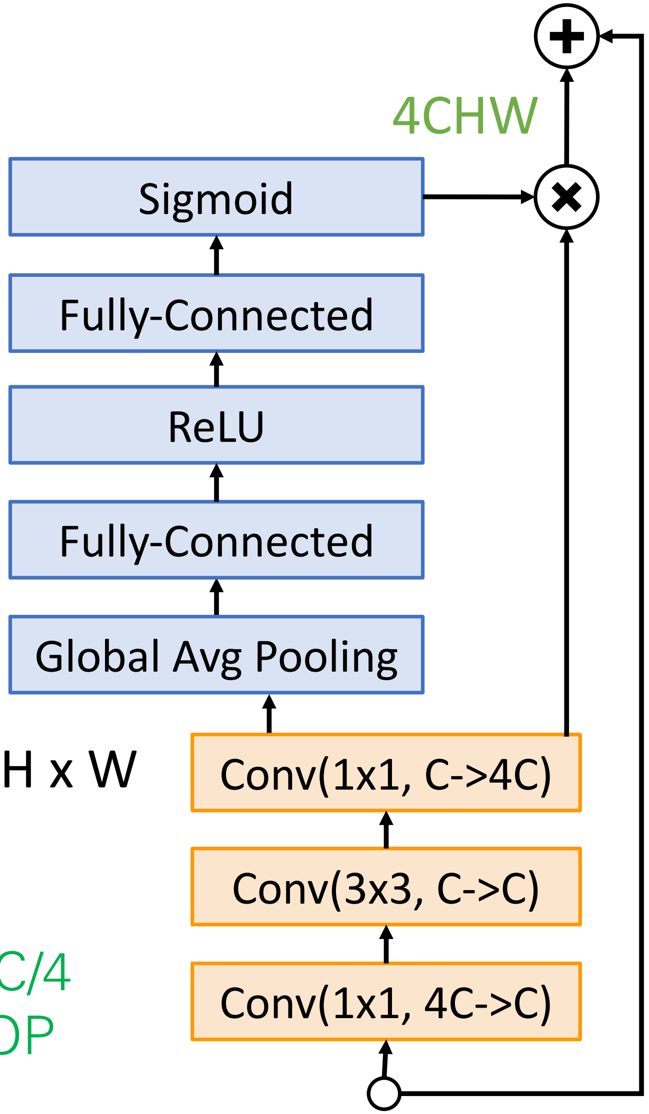


Bottleneck ResNet block
FLOPs: $17HWC^2$
H=W=56, C=64: 218 MFLOP

Adds **global context** to each ResNet block

Increases overall FLOPs by < 1%!

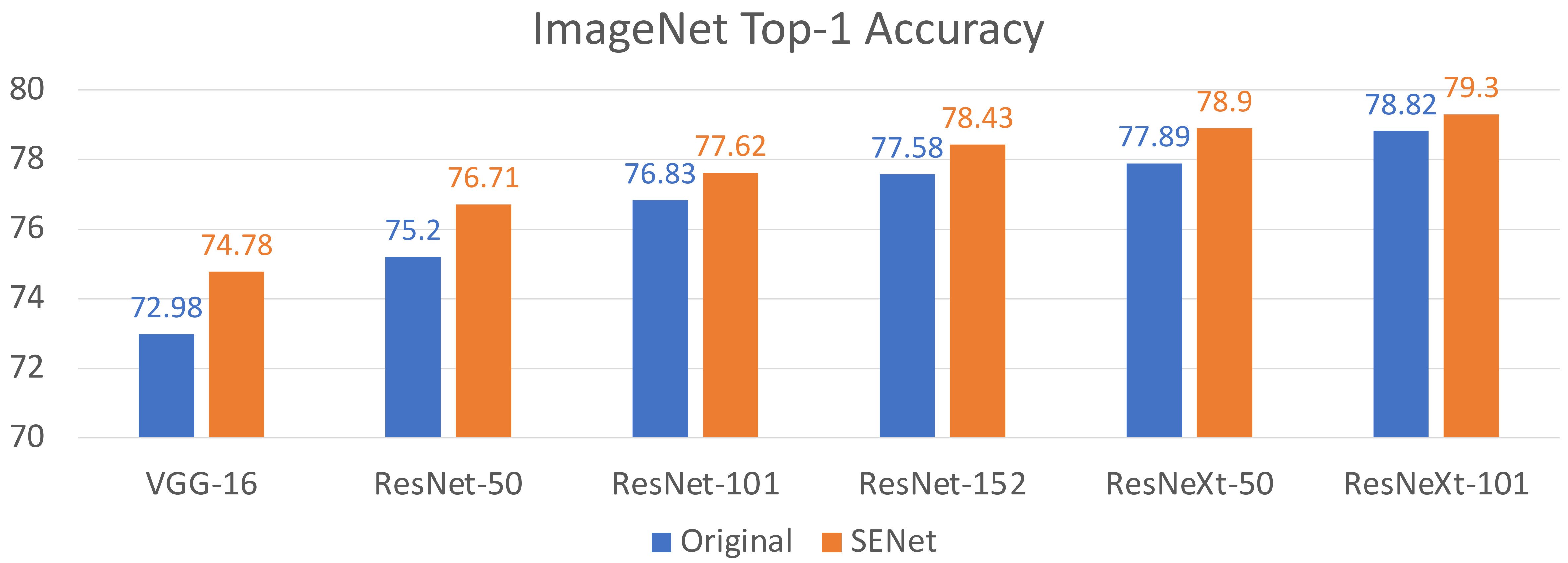
4CHW	
4C	4C x 1 x 1
C^2	4C x 1 x 1
C/4	C/4 x 1 x 1
C^2	C/4 x 1 x 1
CHW	4C x 1 x 1
	4C x H x W



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

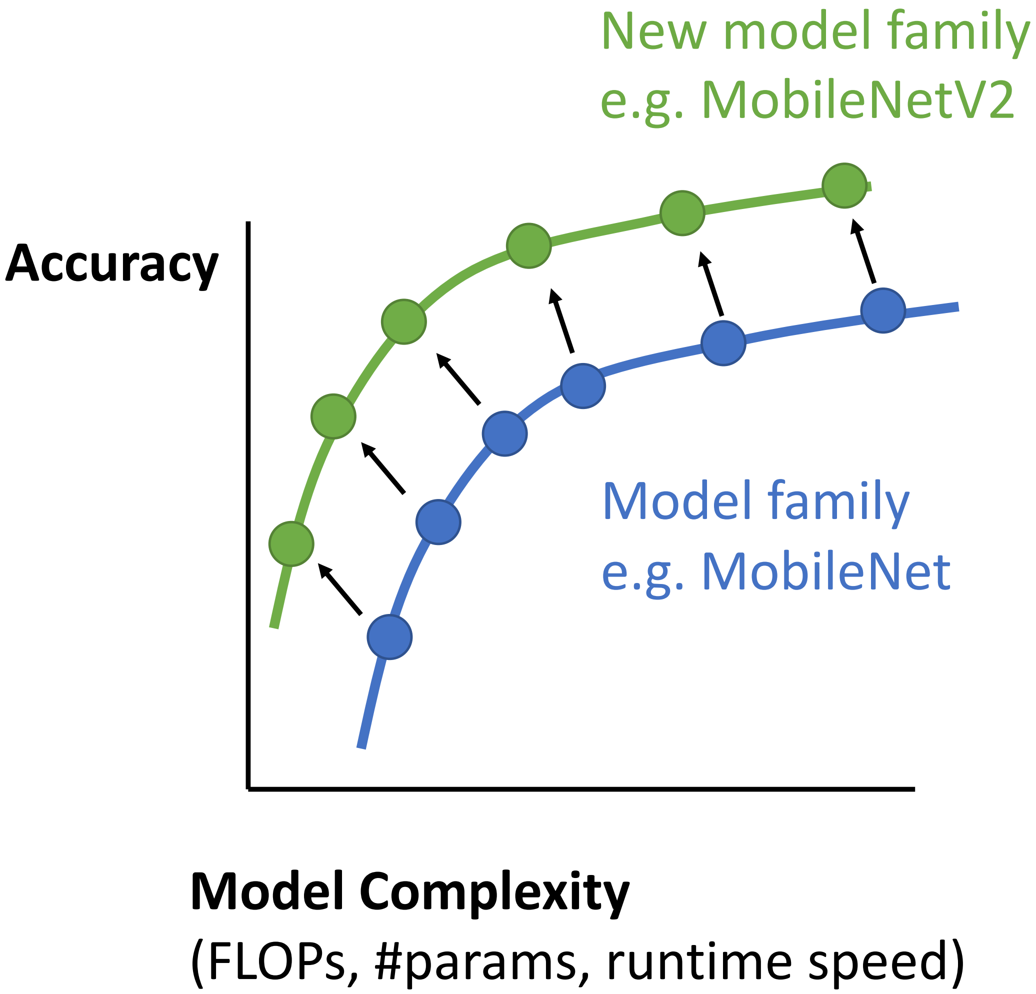
Hu et al, "Squeeze-and-Excitation networks", CVPR 2018

Recall: Convolution Layer

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

Compare **families of models**:

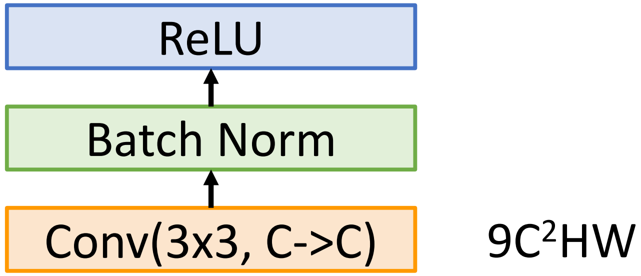
One family is better than another if it moves the whole curve up and to the left



MobileNets: Tiny Networks (For Mobile Devices)

Standard Convolution Block

Total cost: $9C^2HW$



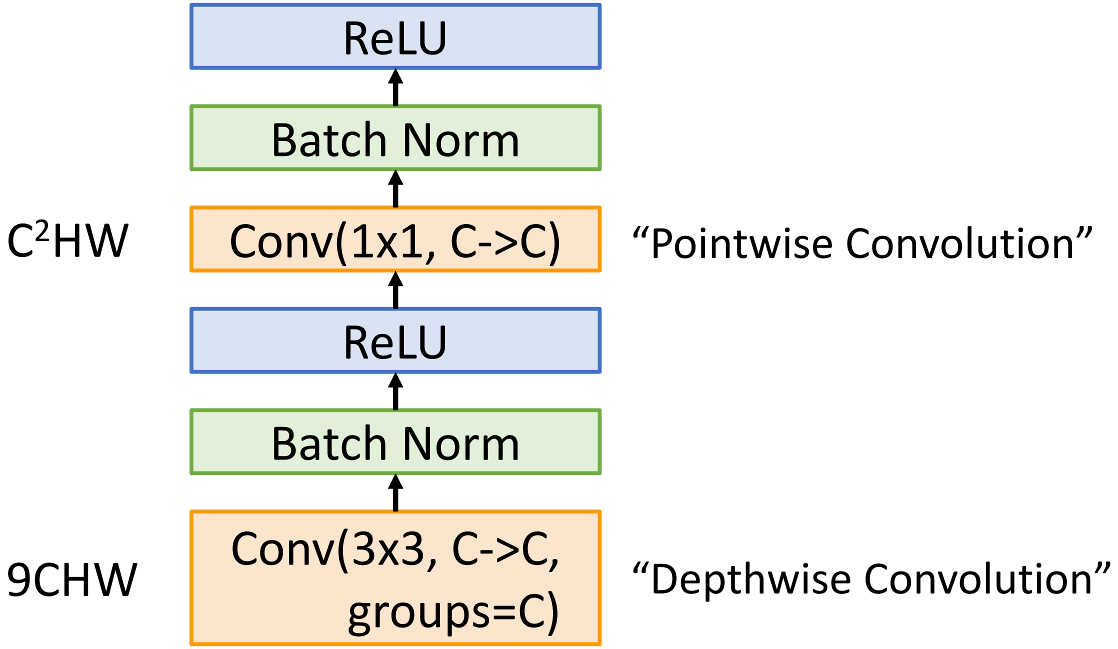
Speedup $= 9C2/(9C+C^2)$

$= 9C/(9+C)$

$=> 9$ (as C->inf)

Depthwise Separable Convolution

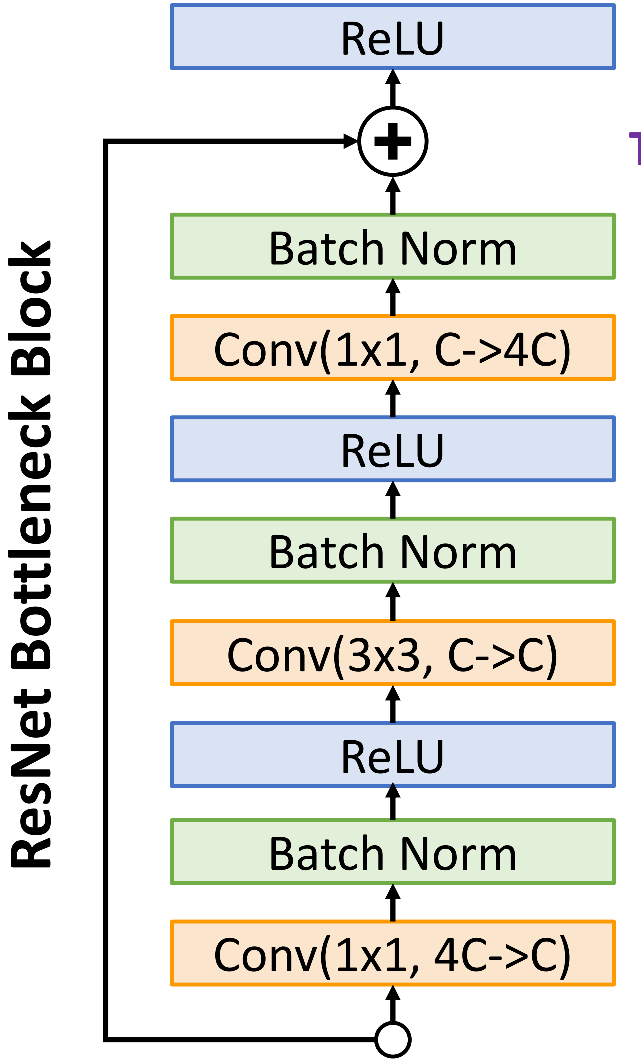
Total cost: $(9C + C^2)HW$



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: 17HWC²

1x1 conv **expands** channels output (4HWC² FLOP)

3x3 conv uses **fewer** channels than input (9HWC² FLOP)

1x1 conv **reduces** channels before 3x3 conv (4HWC² FLOP)

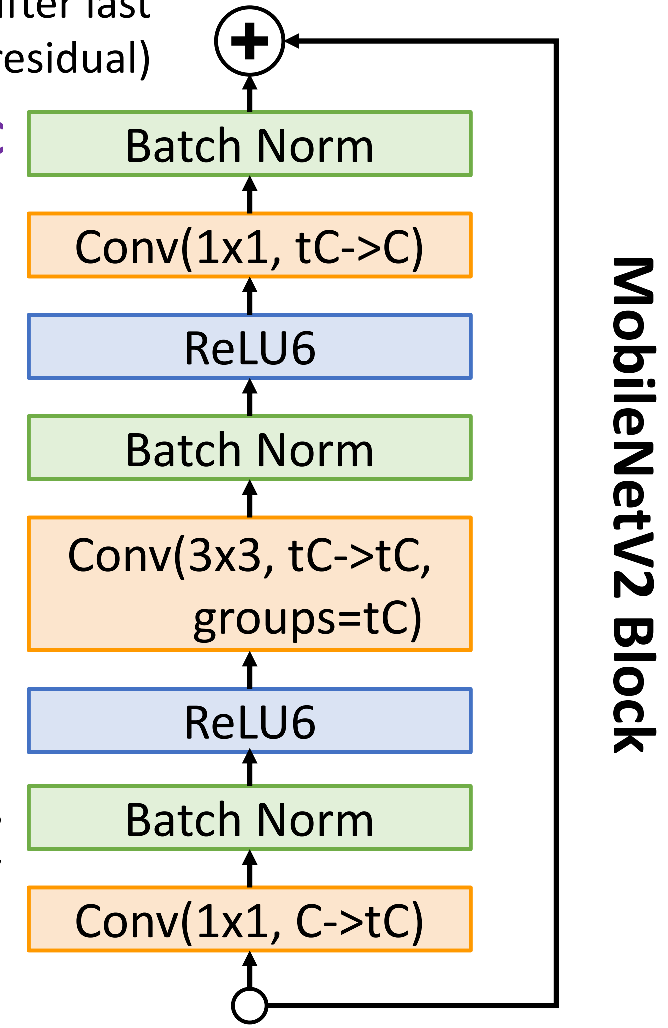
No nonlinearity after last conv! (linear residual)

Total FLOP: 2tHWC² + 9tHWC

1x1 conv **reduces** channels before output (tHWC² FLOP)

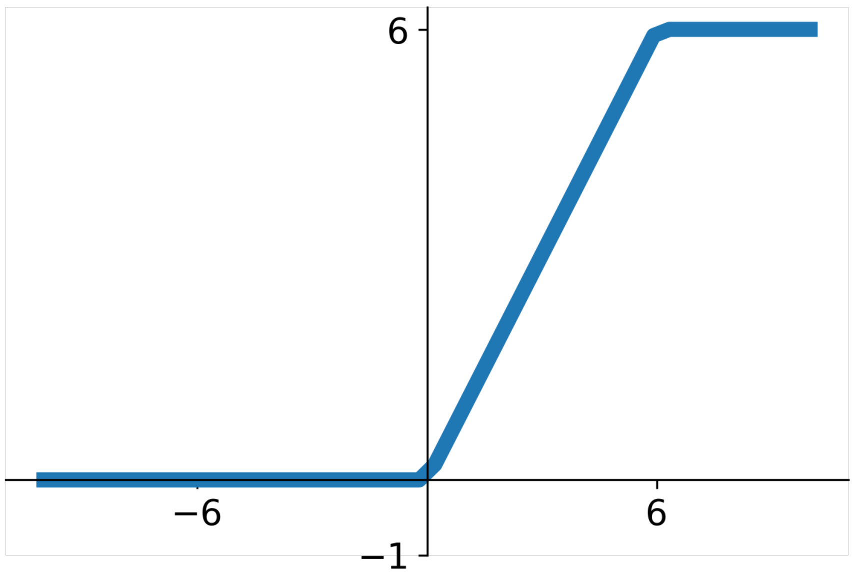
3x3 Depthwise conv with **more** channels than input (9tHWC FLOP)

1x1 conv **increases** channels before 3x3 conv (inverted bottleneck) (tHWC² 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

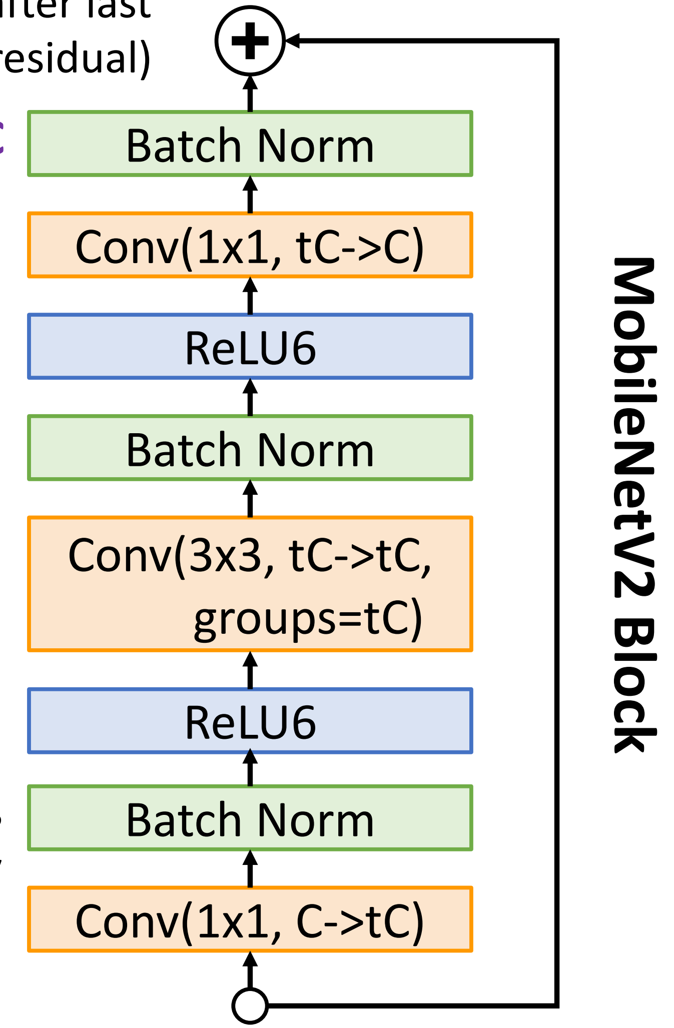
No nonlinearity after last conv! (linear residual)

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

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

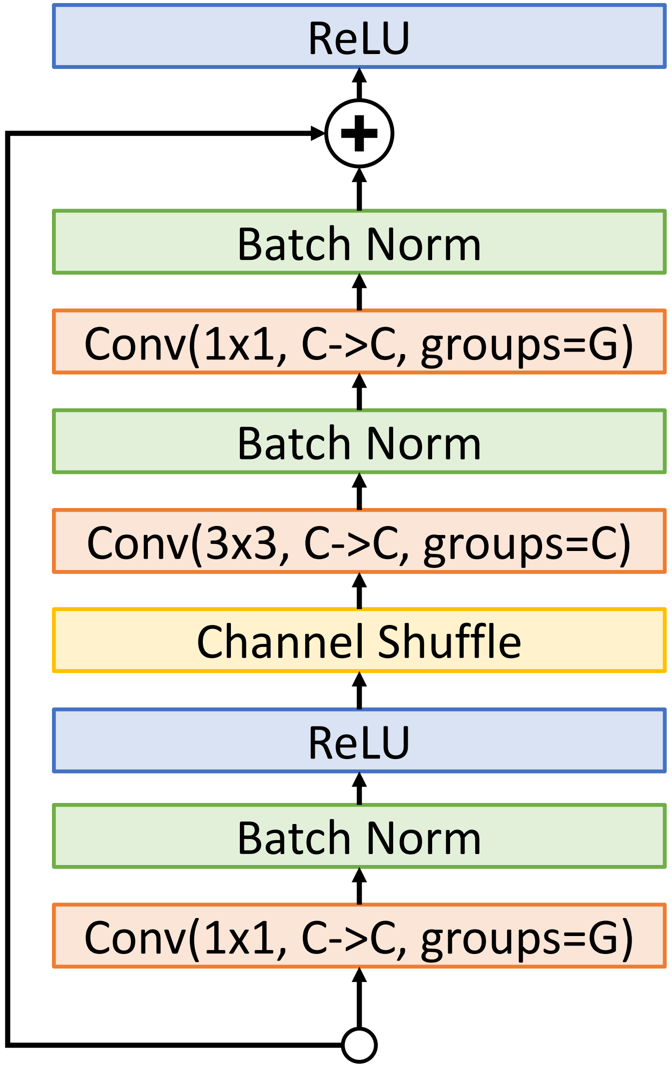
3x3 Depthwise conv with **more** channels than input ($9tHWC$ FLOP)

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



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

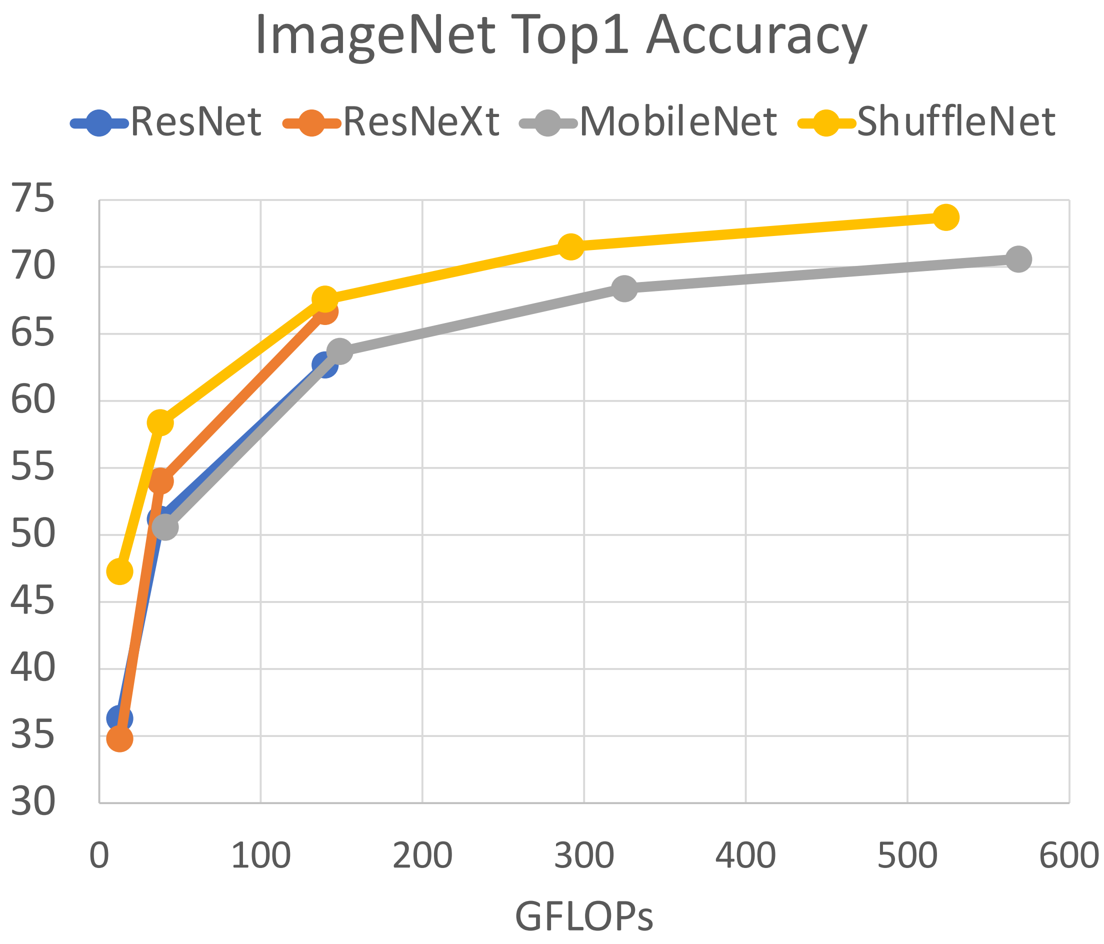
ShuffleNet



1x1 grouped conv

3x3 depthwise conv,
No nonlinearity here!

1x1 grouped conv



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

CNN Architectures Summary

- Early work (AlexNet->VGG->ResNet):**bigger networks work better**
- New focus on **efficiency**: Improve accuracy, control for network complexity
- Grouped and Depthwise 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

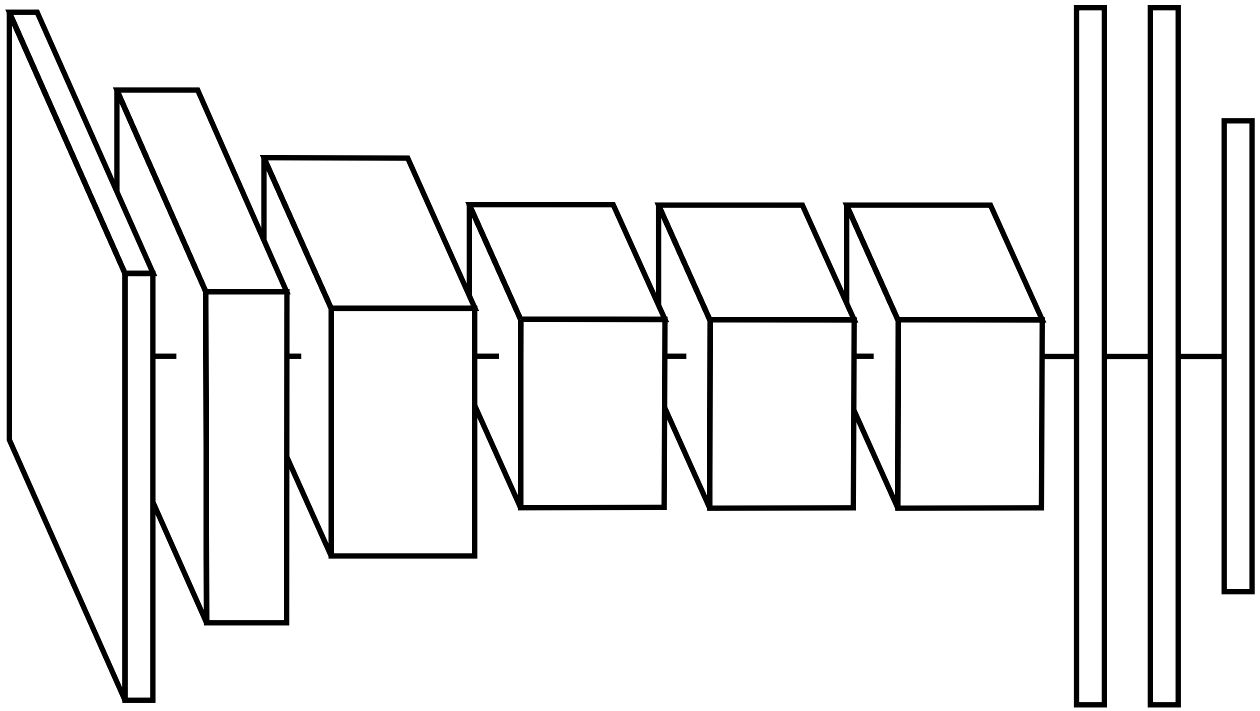
Beyond CNNs

- 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

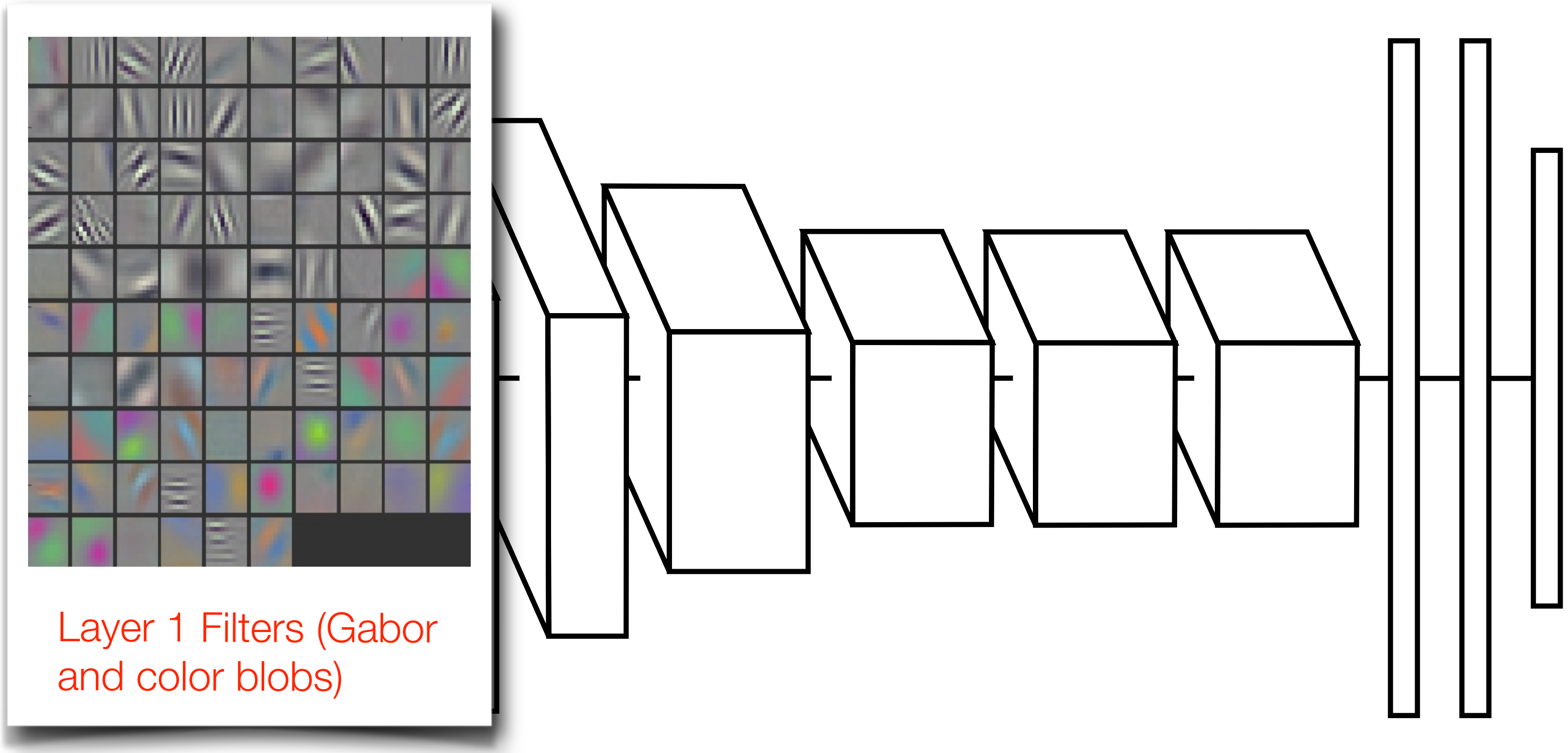
- CNNs discover effective representations. Why not to use them?



Slide credit: Jason Yosinski

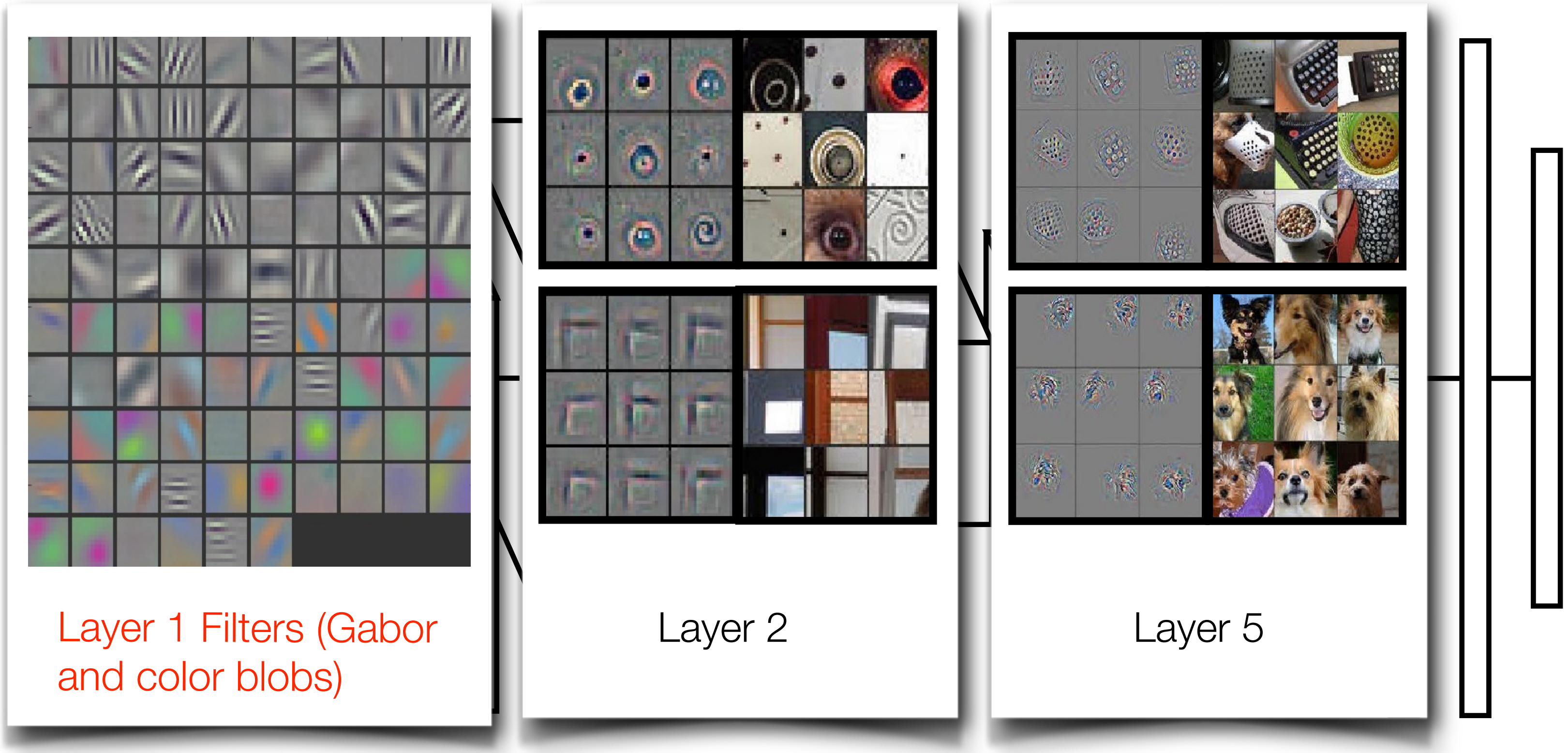
CNN activations as deep features

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



CNN activations as deep features

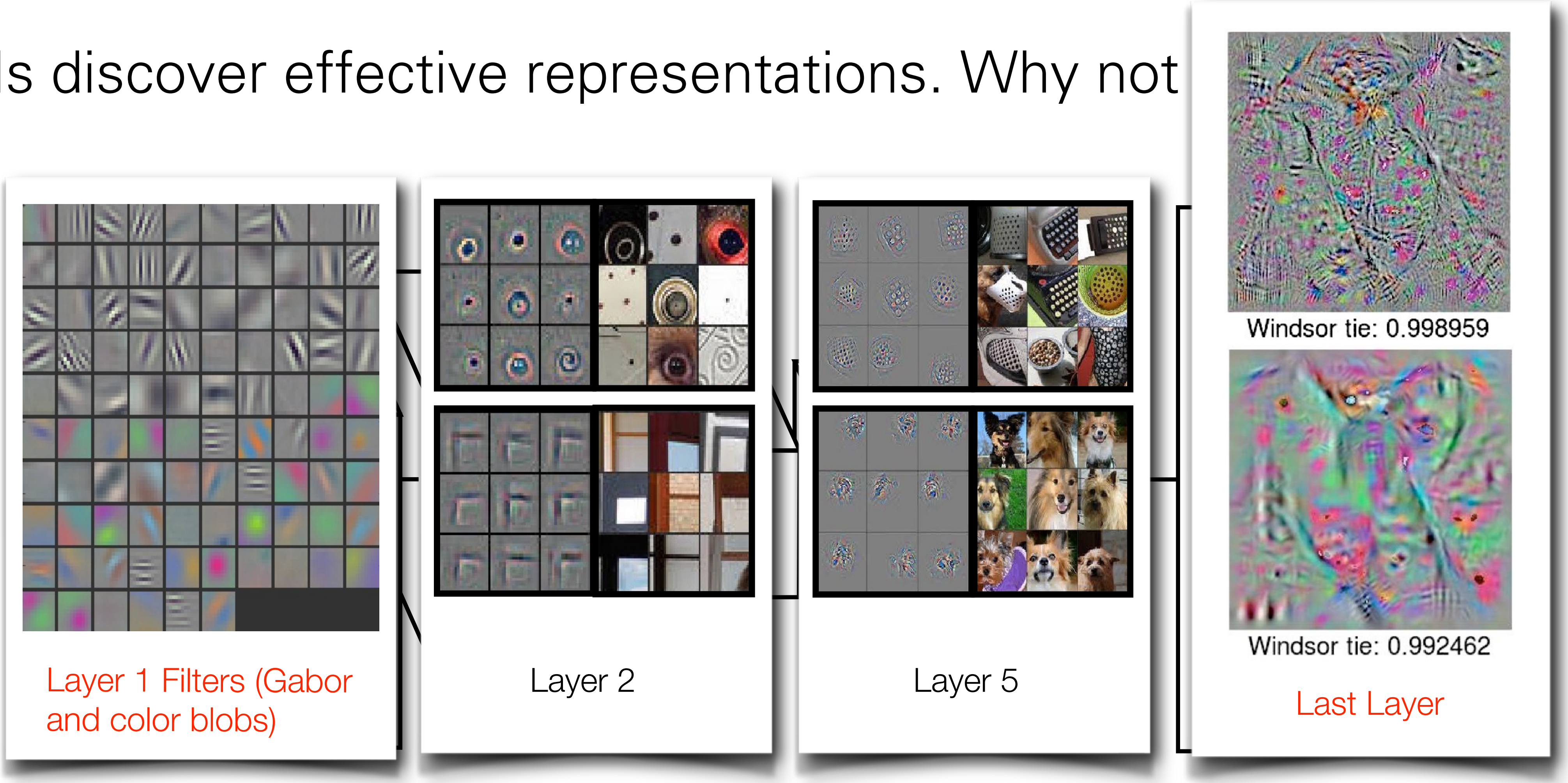
- CNNs discover effective representations. Why not to use them?



Zeiler et al., 2014

CNN activations as deep features

- CNNs discover effective representations. Why not



Zeiler et al., 2014

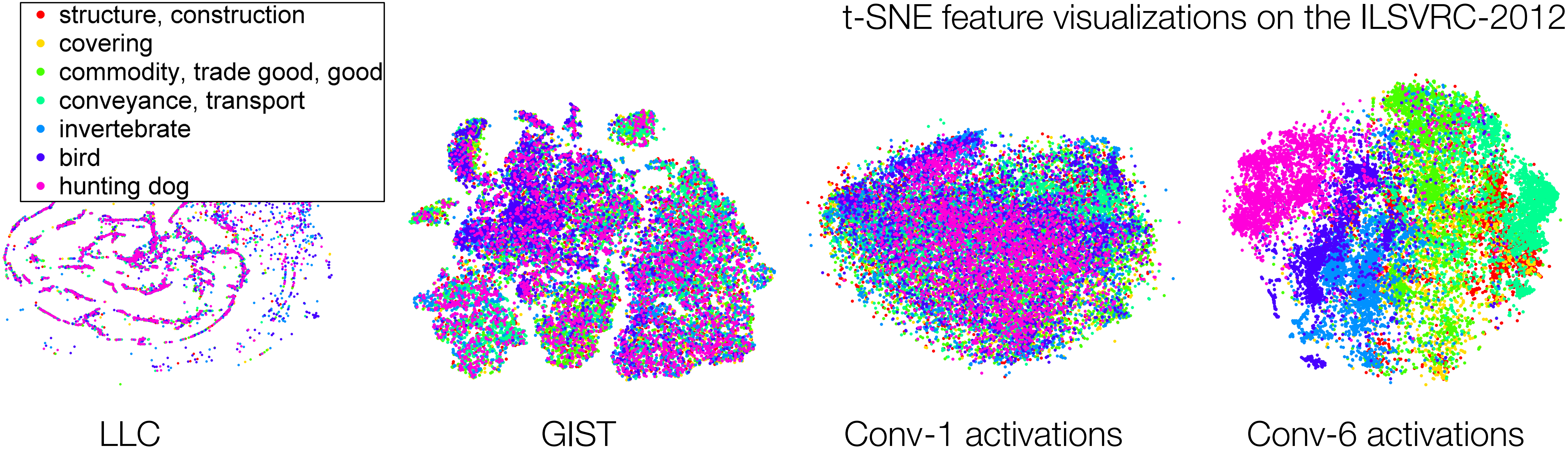
Nguyen et al., 2014

Slide credit: Jason Yosinski

CNNs as deep features

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

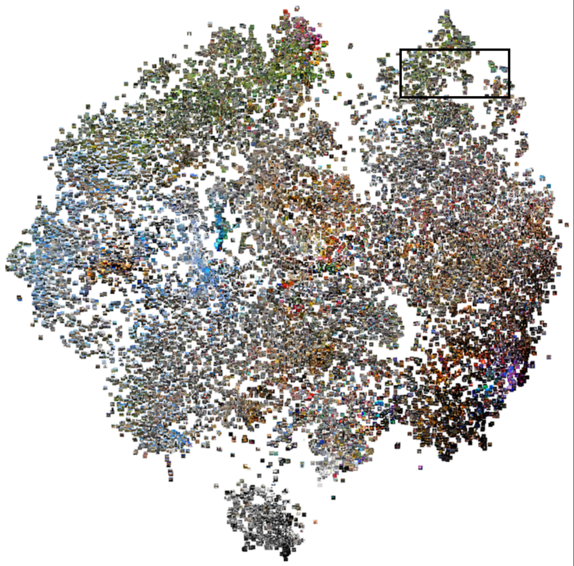
t-SNE feature visualizations on the ILSVRC-2012



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



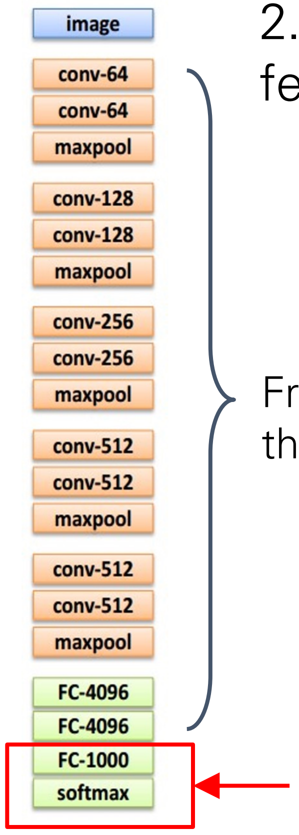
A. Joulin, L.J.P. van der Maaten, A. Jabri, and N. Vasilache **Learning visual features from Large Weakly supervised Data.** ECCV 2016

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.

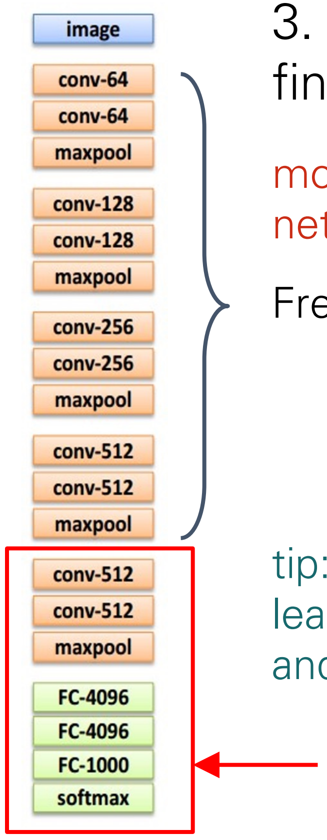
1. Train on Imagenet



2. Small dataset: feature extractor

Freeze these

Train this



3. Medium dataset: finetuning

more data = retrain more of the network (or all of it)

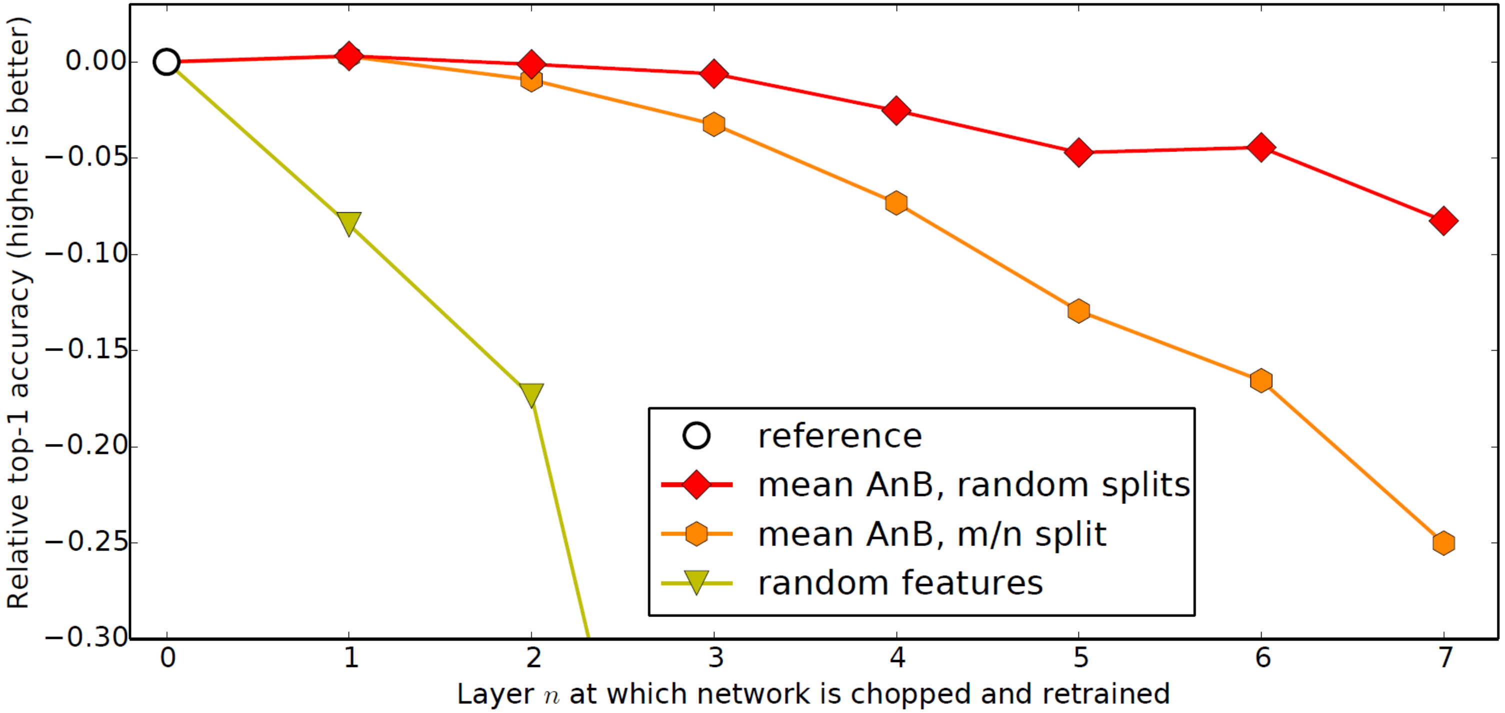
Freeze these

tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers

Train this

How transferable are features in CNN networks?

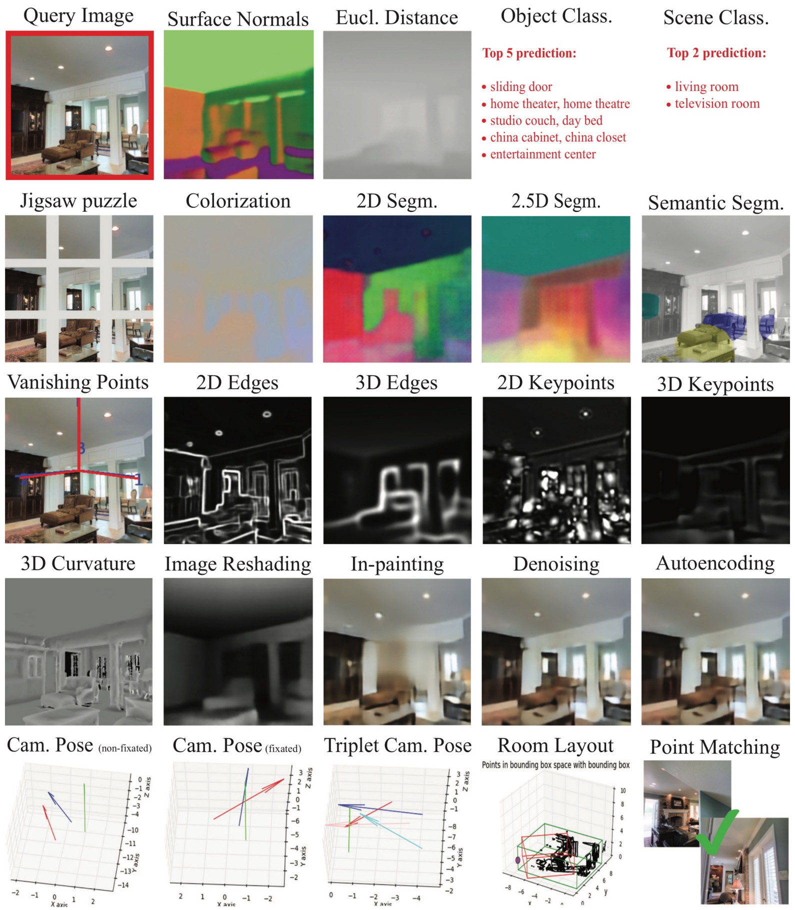
- Divide ImageNet into man-made objects A (449 classes) and natural objects B (551 classes)
- The transferability of features decreases as the distance between the base task and target task increases

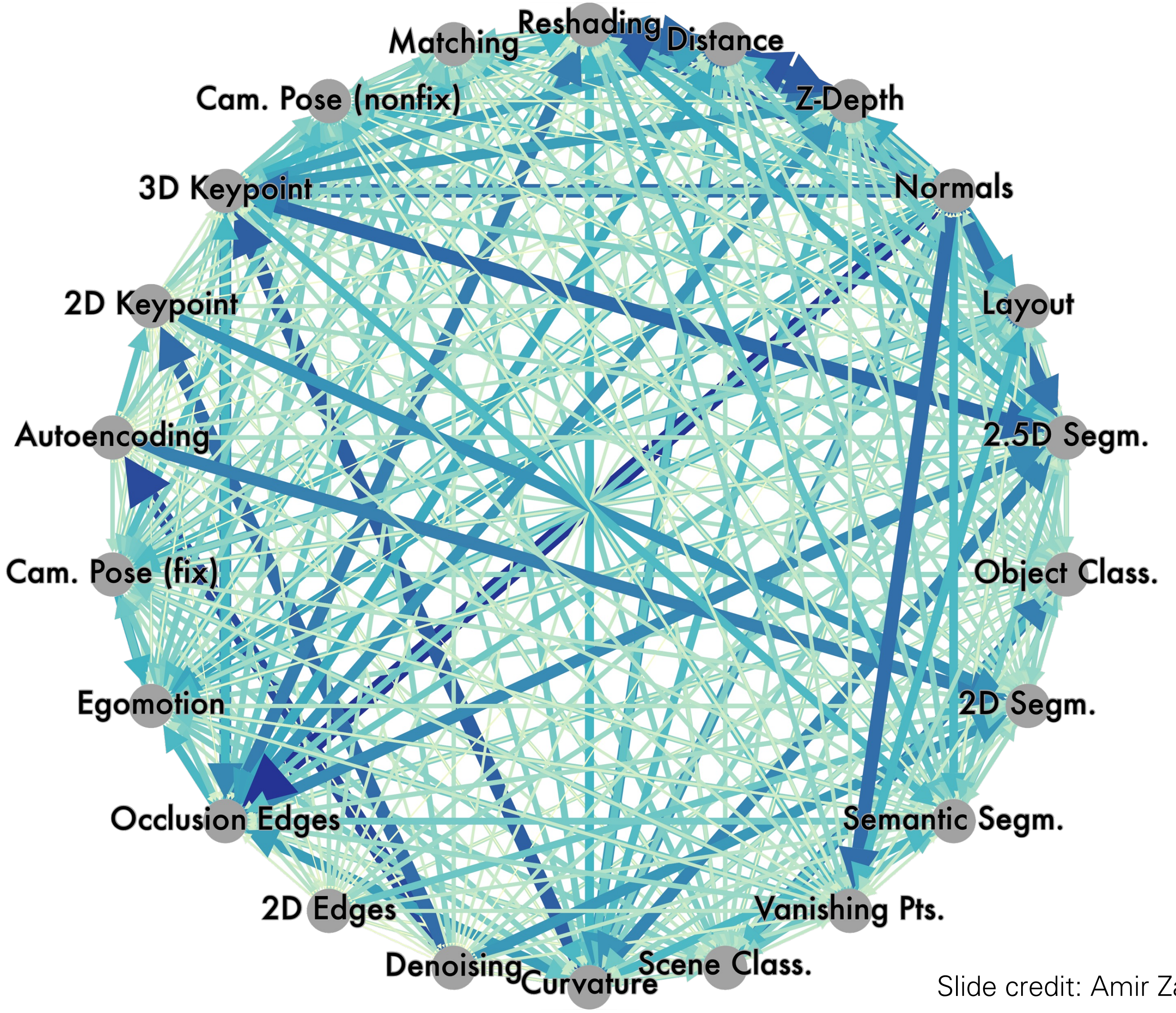


Slide credit: Xiaogan Wang

How transferable are features in CNN networks?

- An open research problem





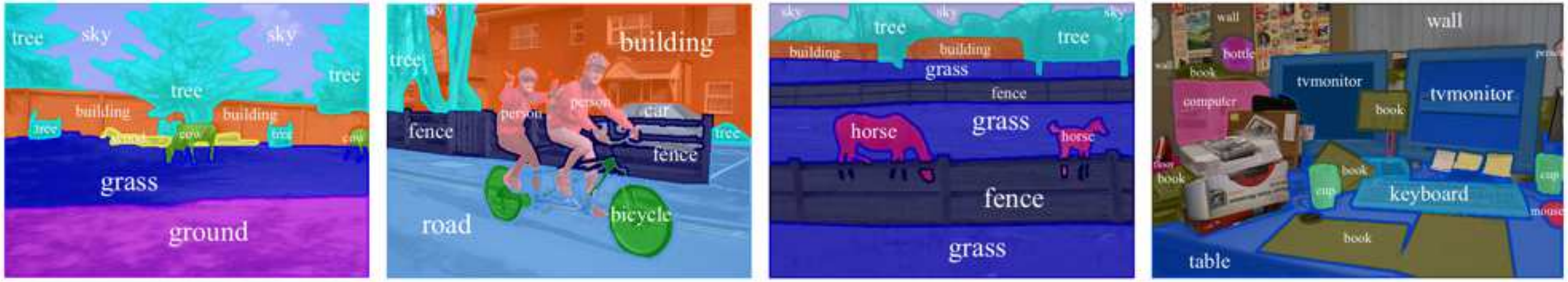
Slide credit: Amir Zamir

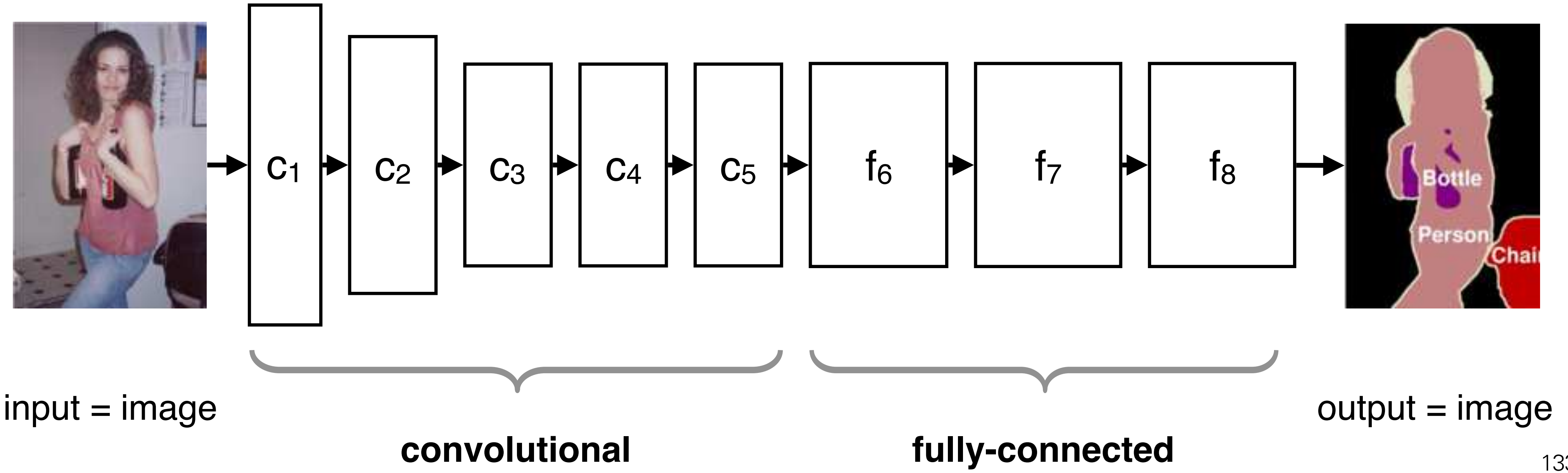
A. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, S. Savarese. **Taskonomy: Disentangling Task Transfer Learning**. CVPR 2018.

Semantic Segmentation

Semantic Image Segmentation

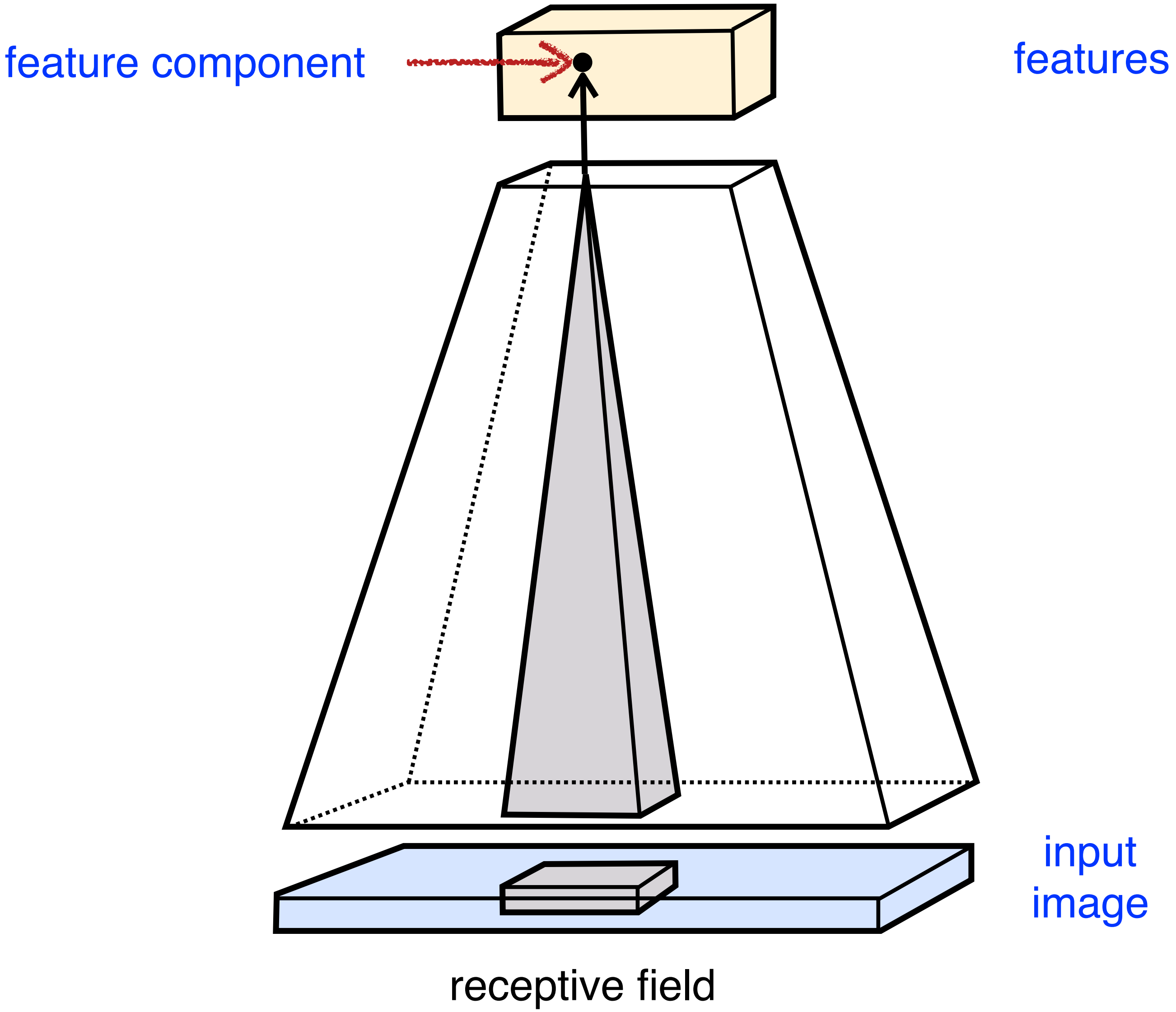
- Label individual pixels





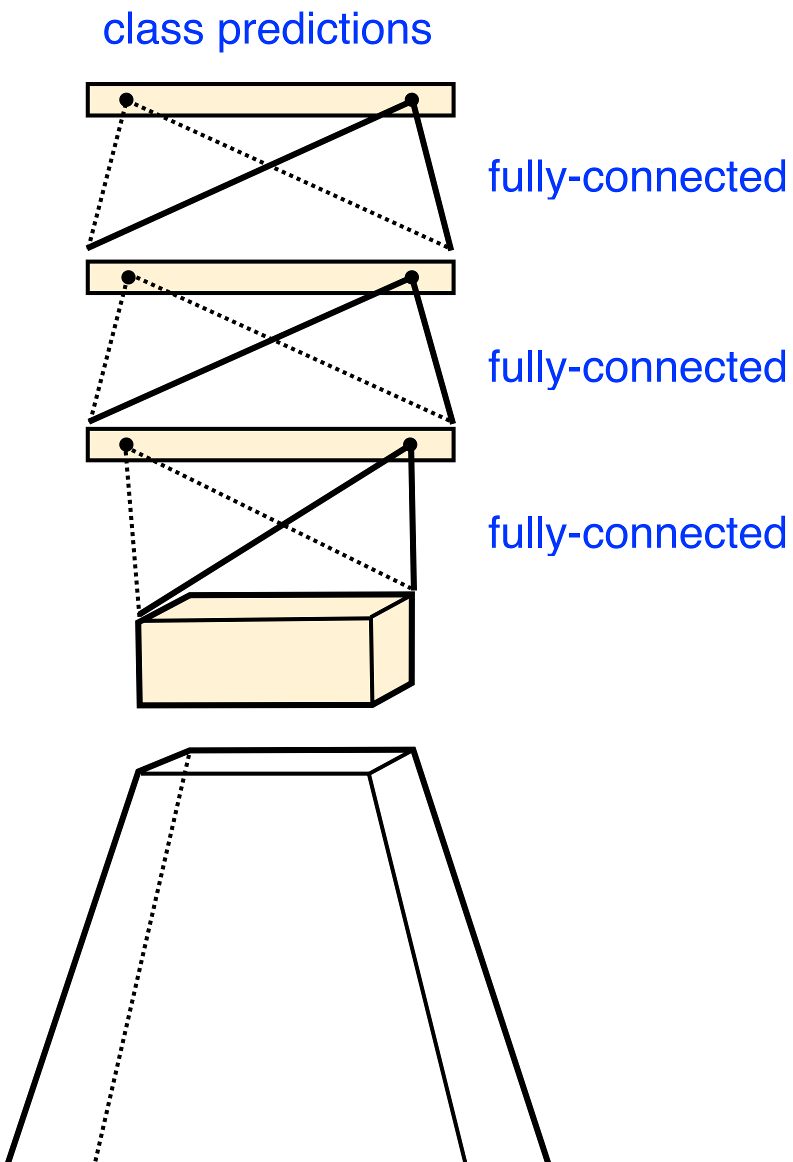
Convolutional Layers

- Local receptive field



Fully Connected Layers

- Global receptive field

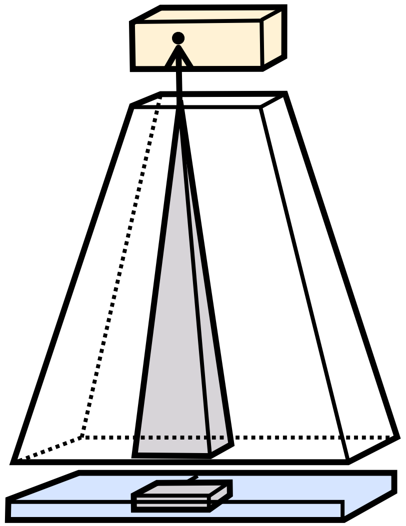


Convolutional vs. Fully Connected

- Comparing the receptive fields

Downsampling filters

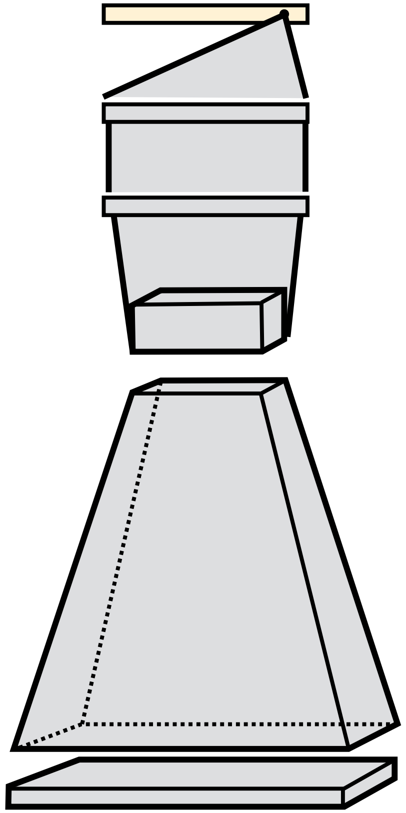
Responses are spatially selective, can be used to localize things.



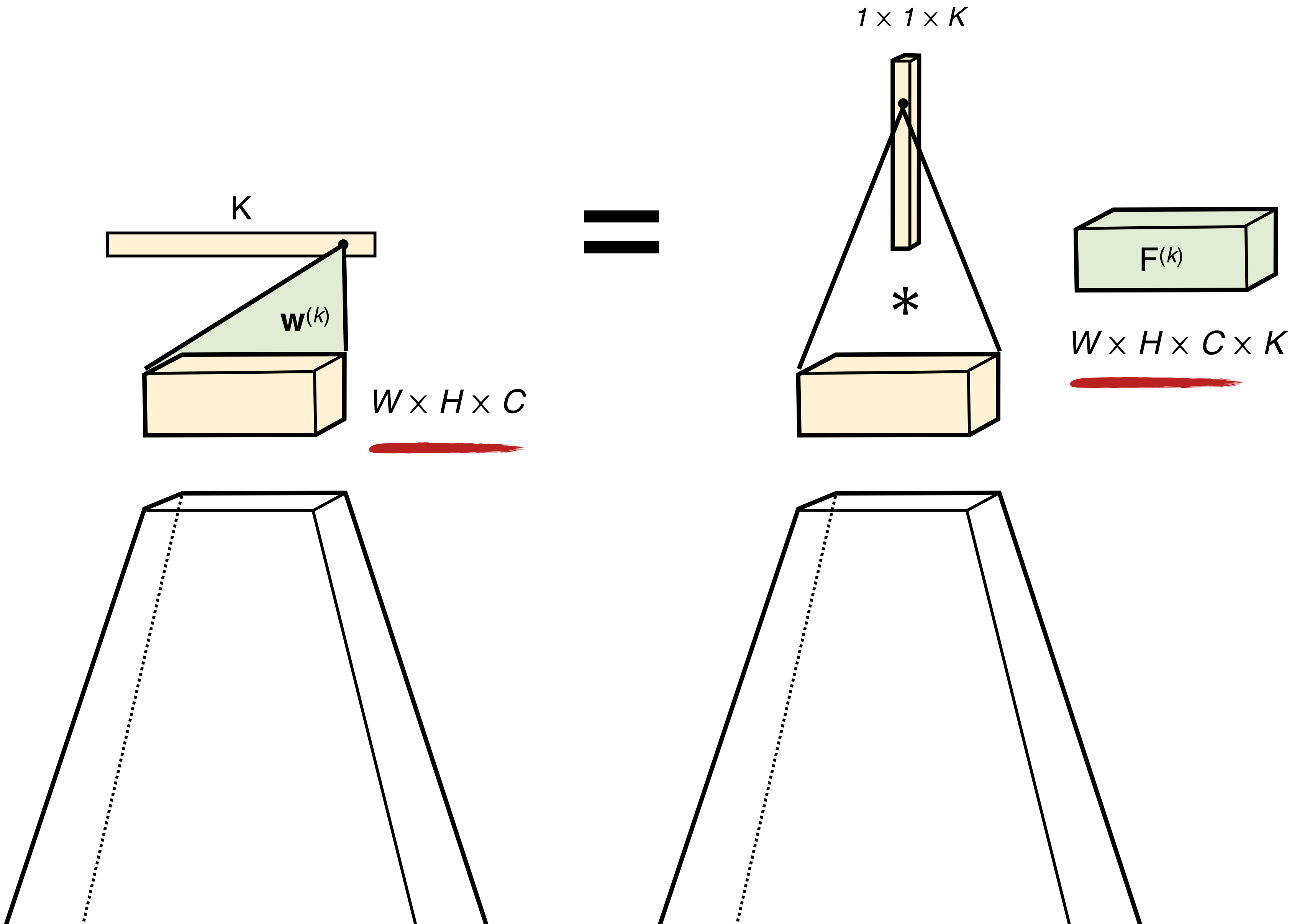
Upsampling filters

Responses are global, do not characterize well position

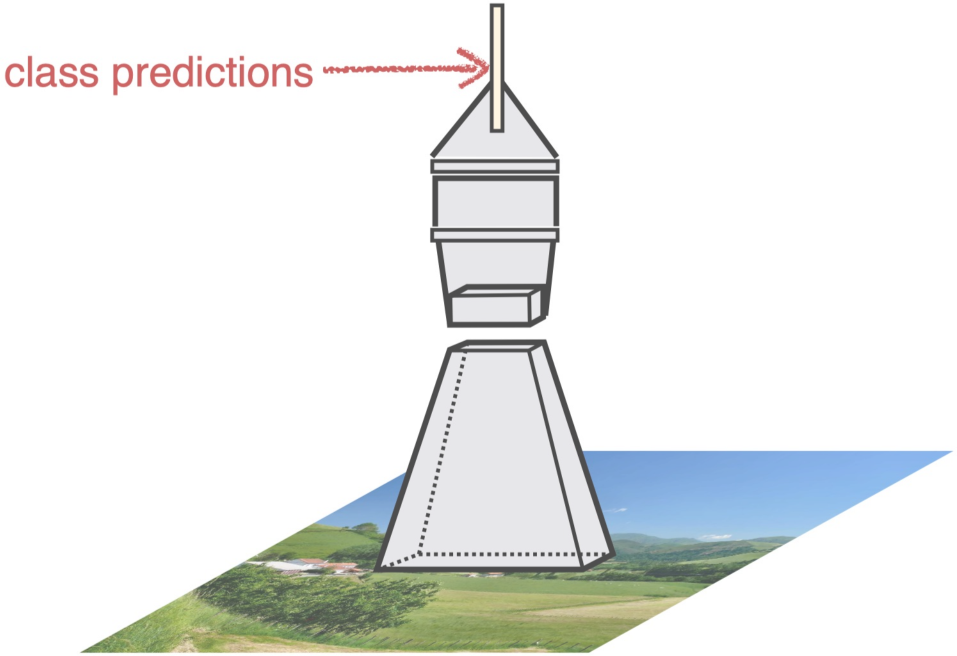
Which one is more useful for pixel level labelling?



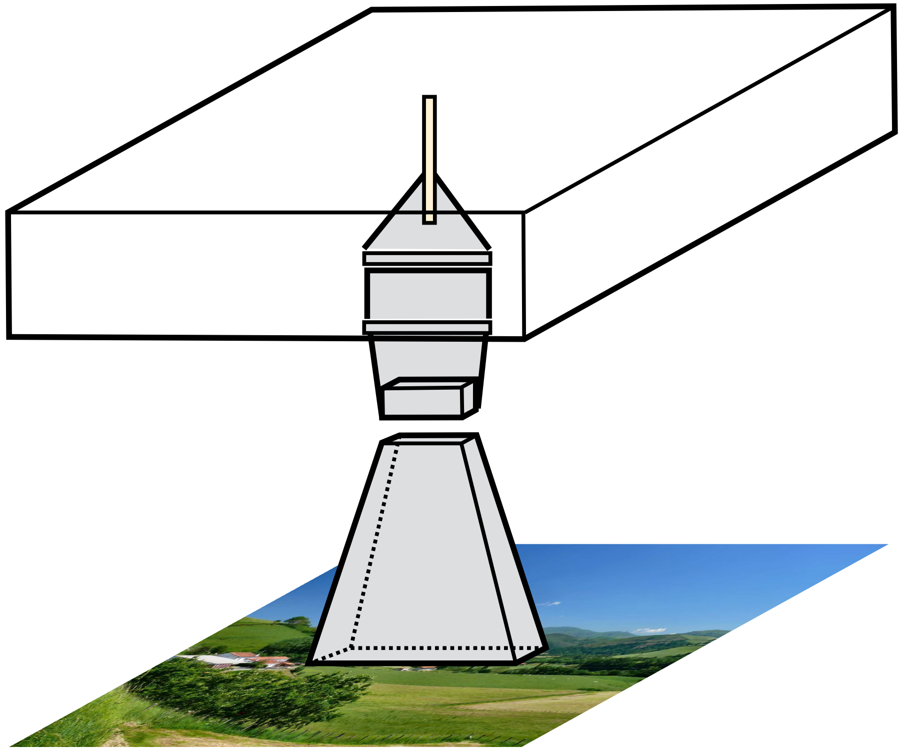
Fully-Connected Layer = Large Filter



Fully-Convolutional Neural Networks



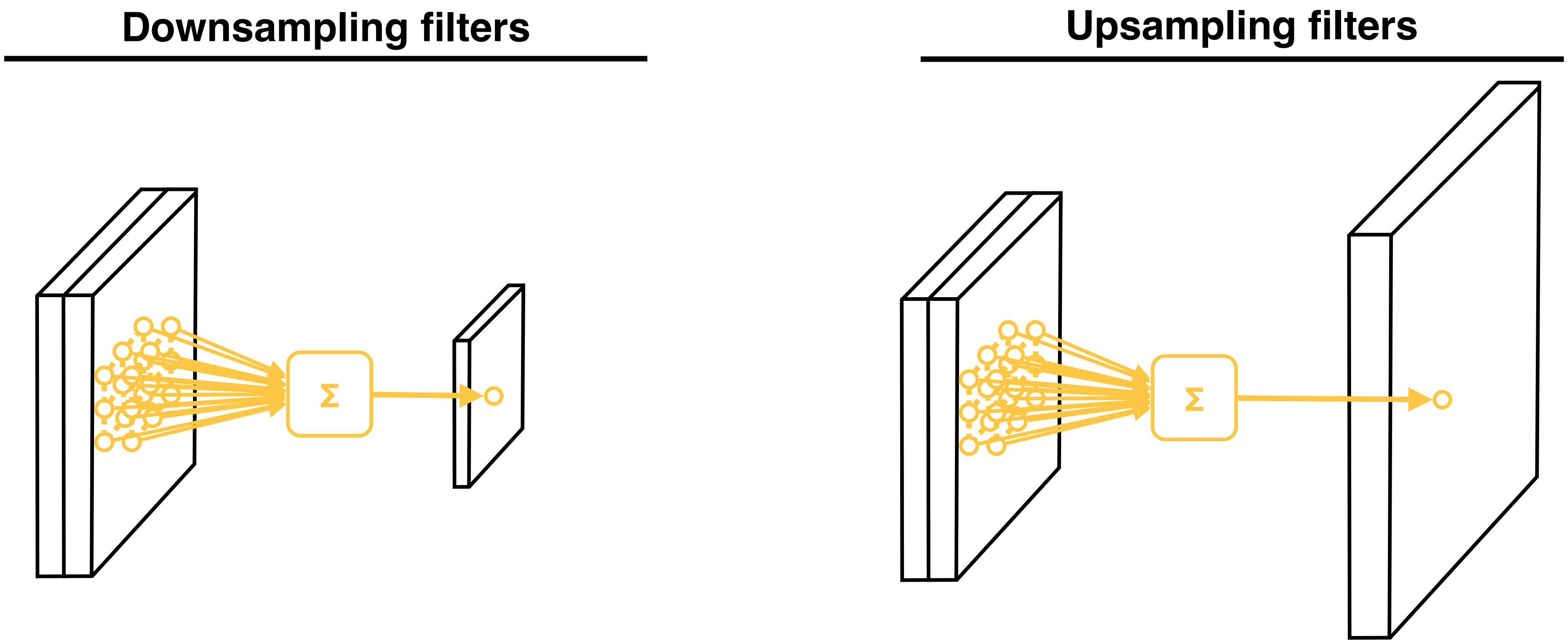
Fully-Convolutional Neural Networks



- **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

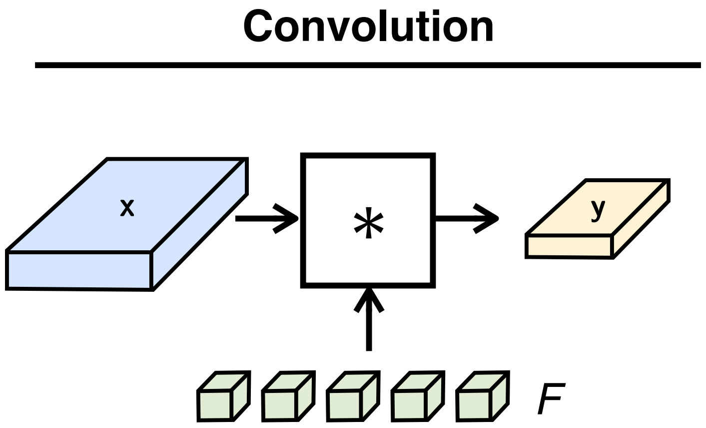


Upsampling filters allow to increase the resolution of the output

Very useful to get full-resolution segmentation results

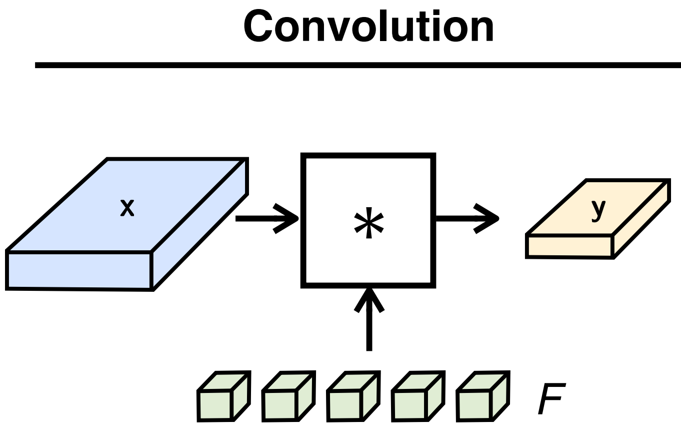
Deconvolution Layer

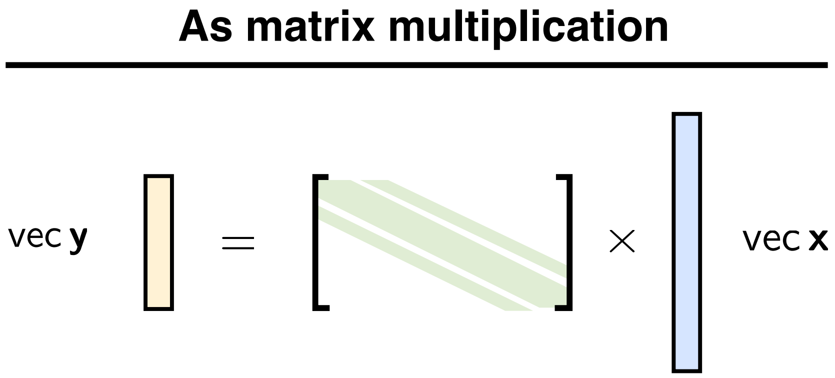
- Or convolution transpose



Deconvolution Layer

- Or convolution transpose

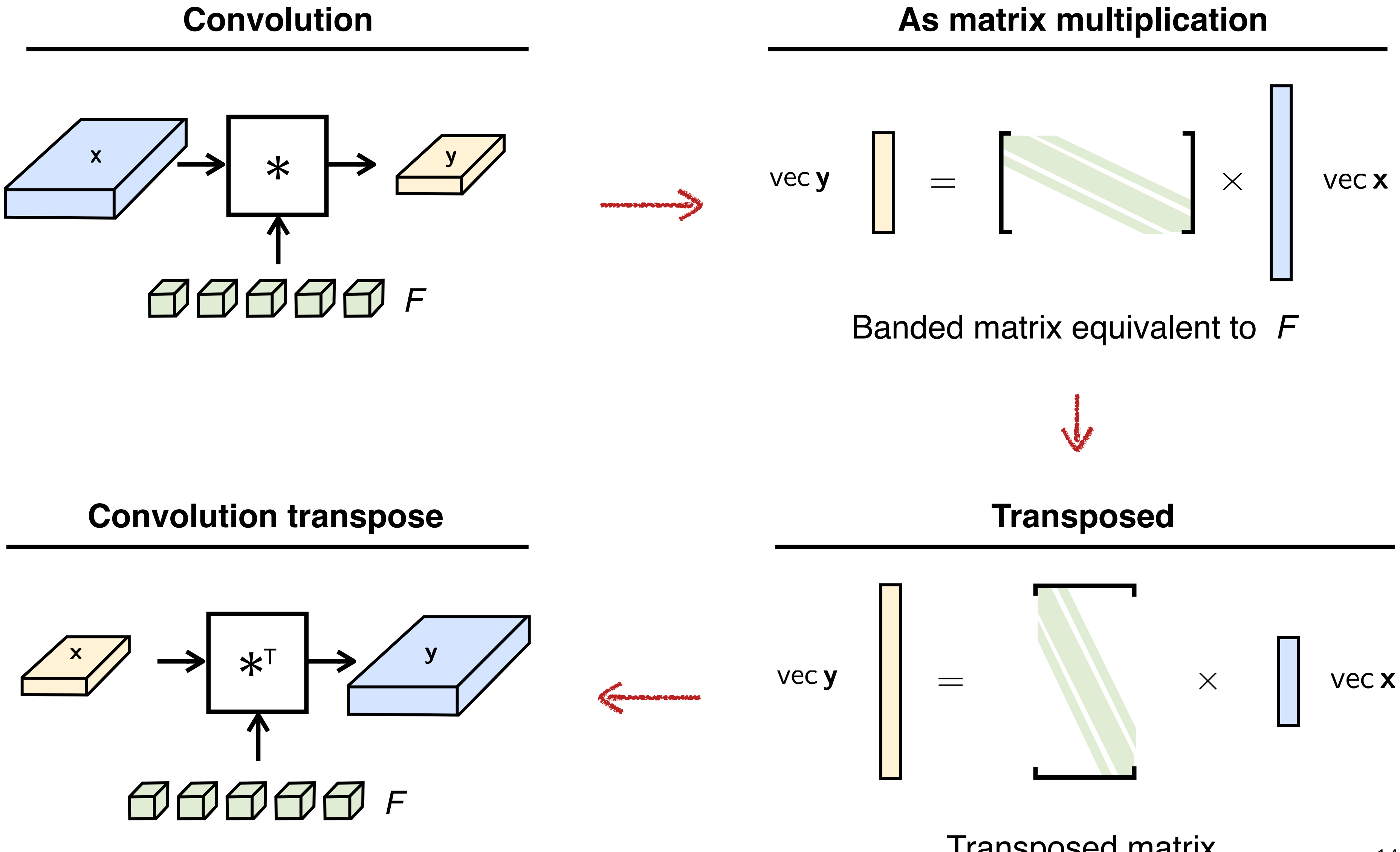




Banded matrix equivalent to F

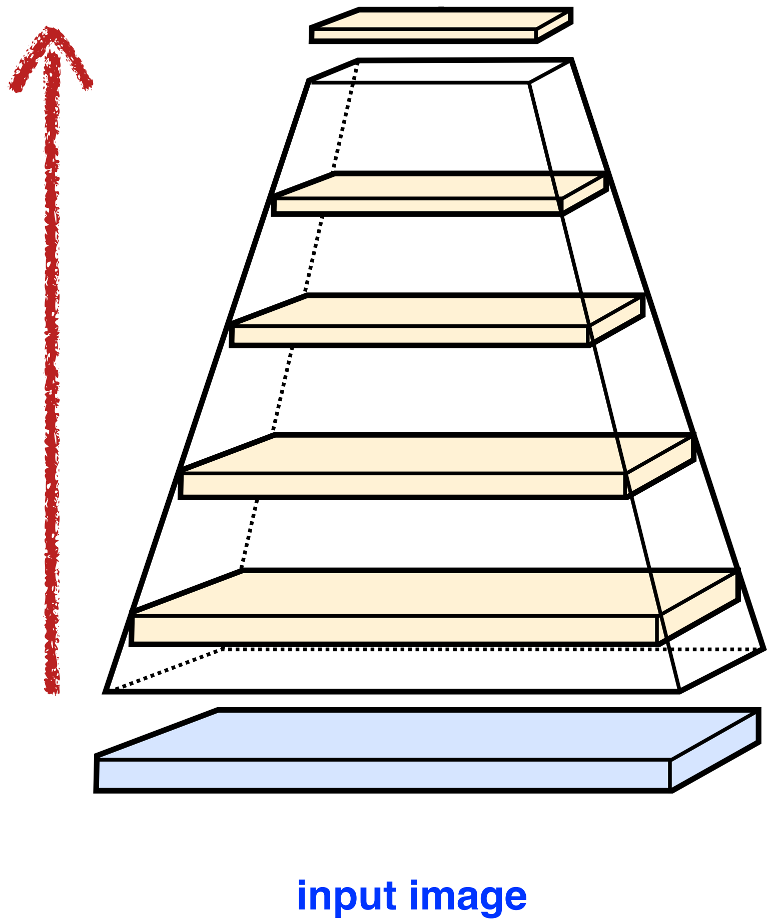
Deconvolution Layer

- Or convolution transpose



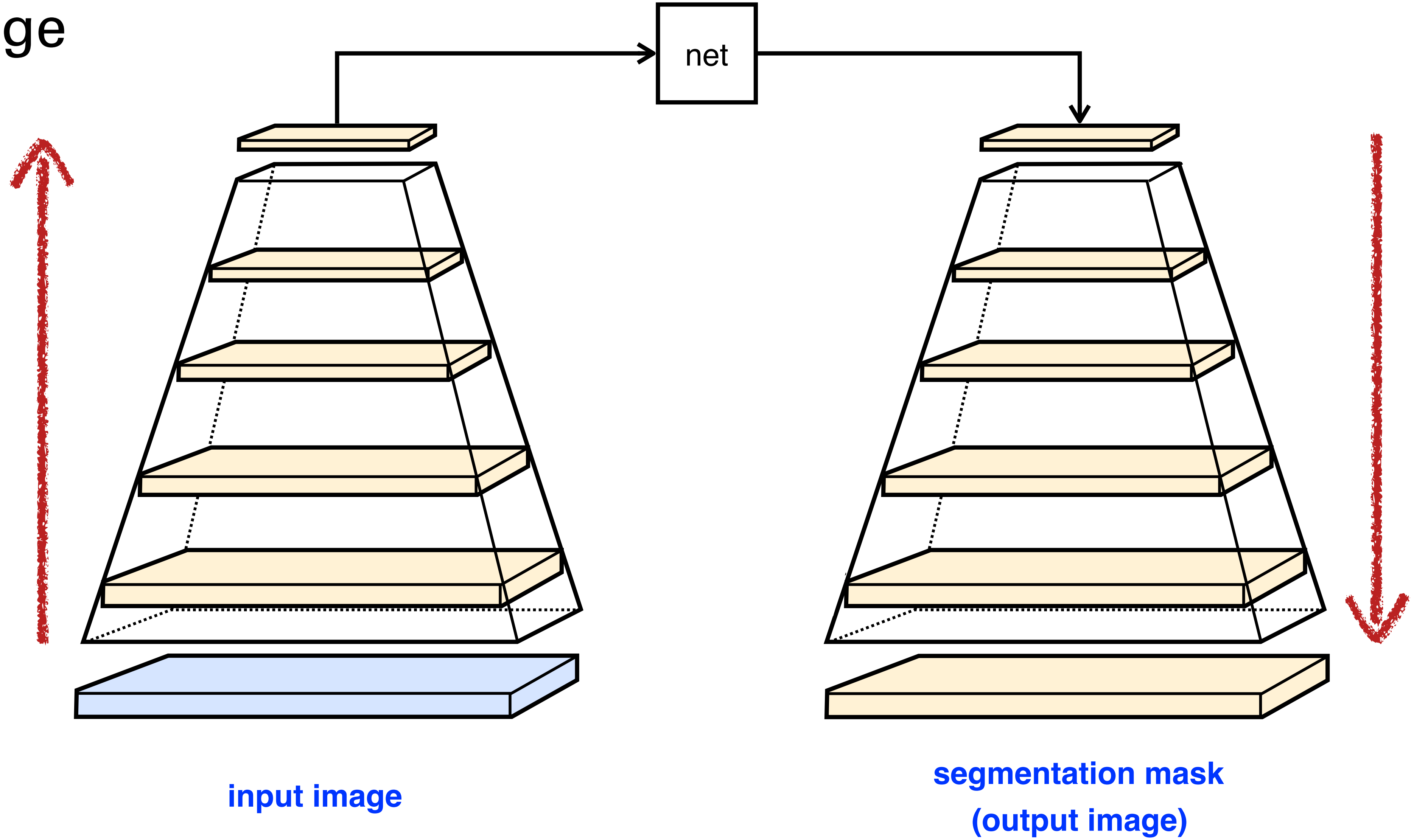
U-Architectures

- Image to image



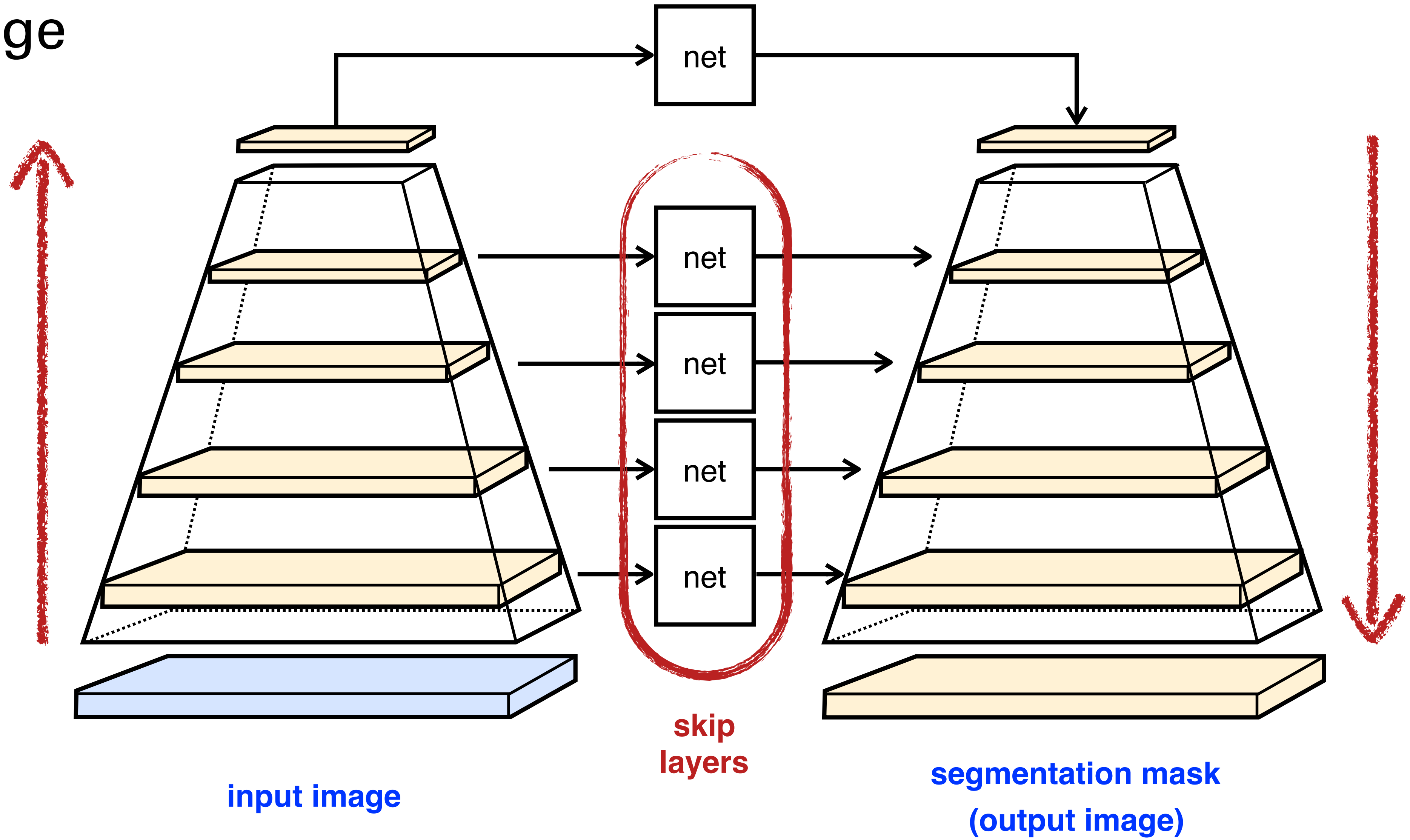
U-Architectures

- Image to image



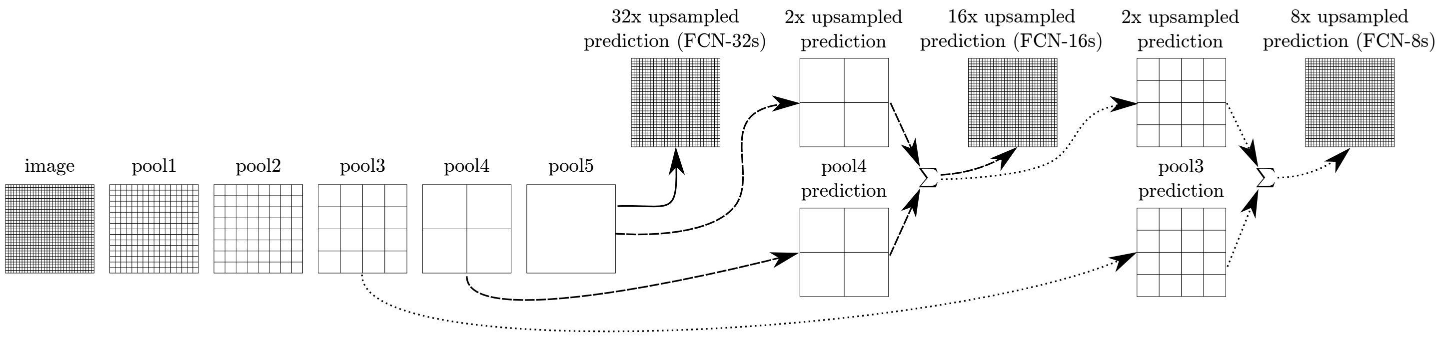
U-Architectures

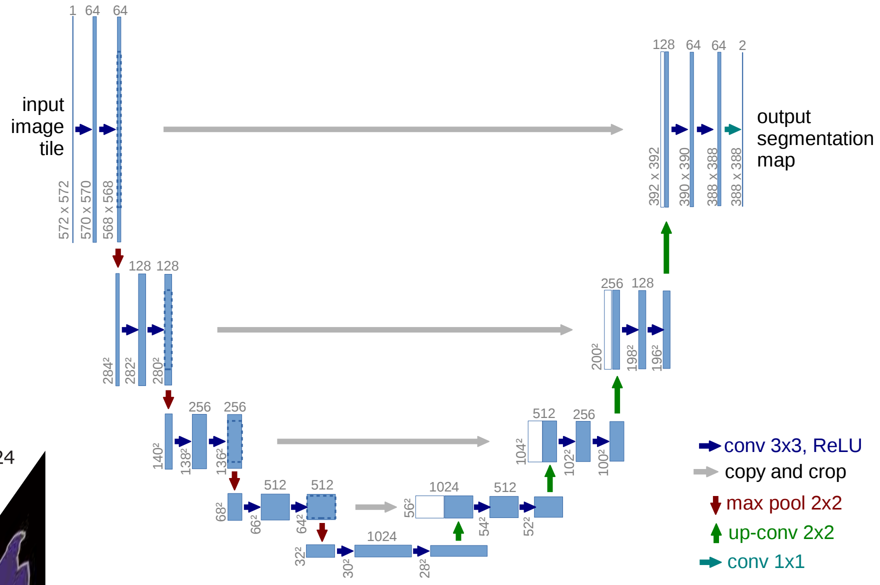
- Image to image

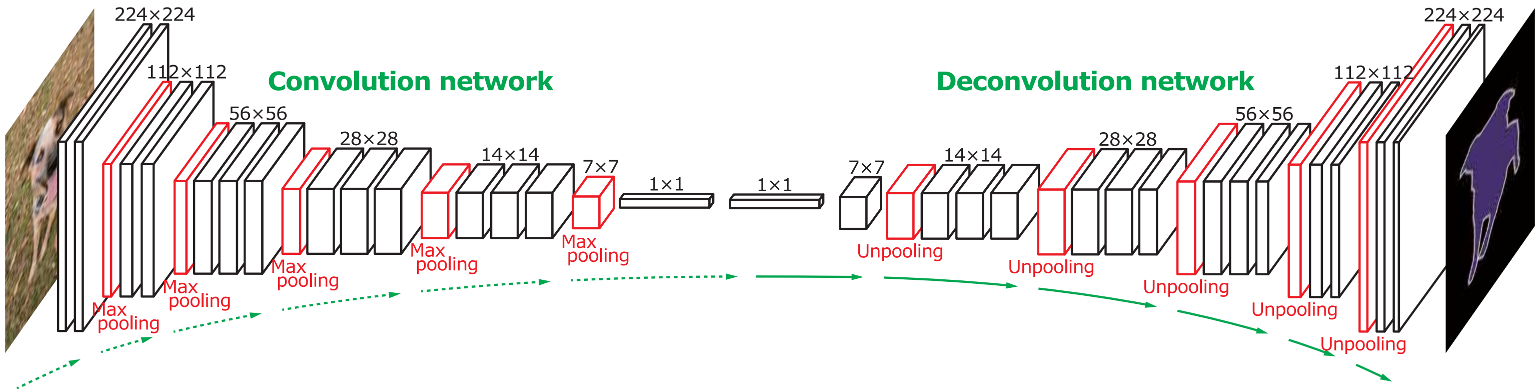


U-Architectures

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

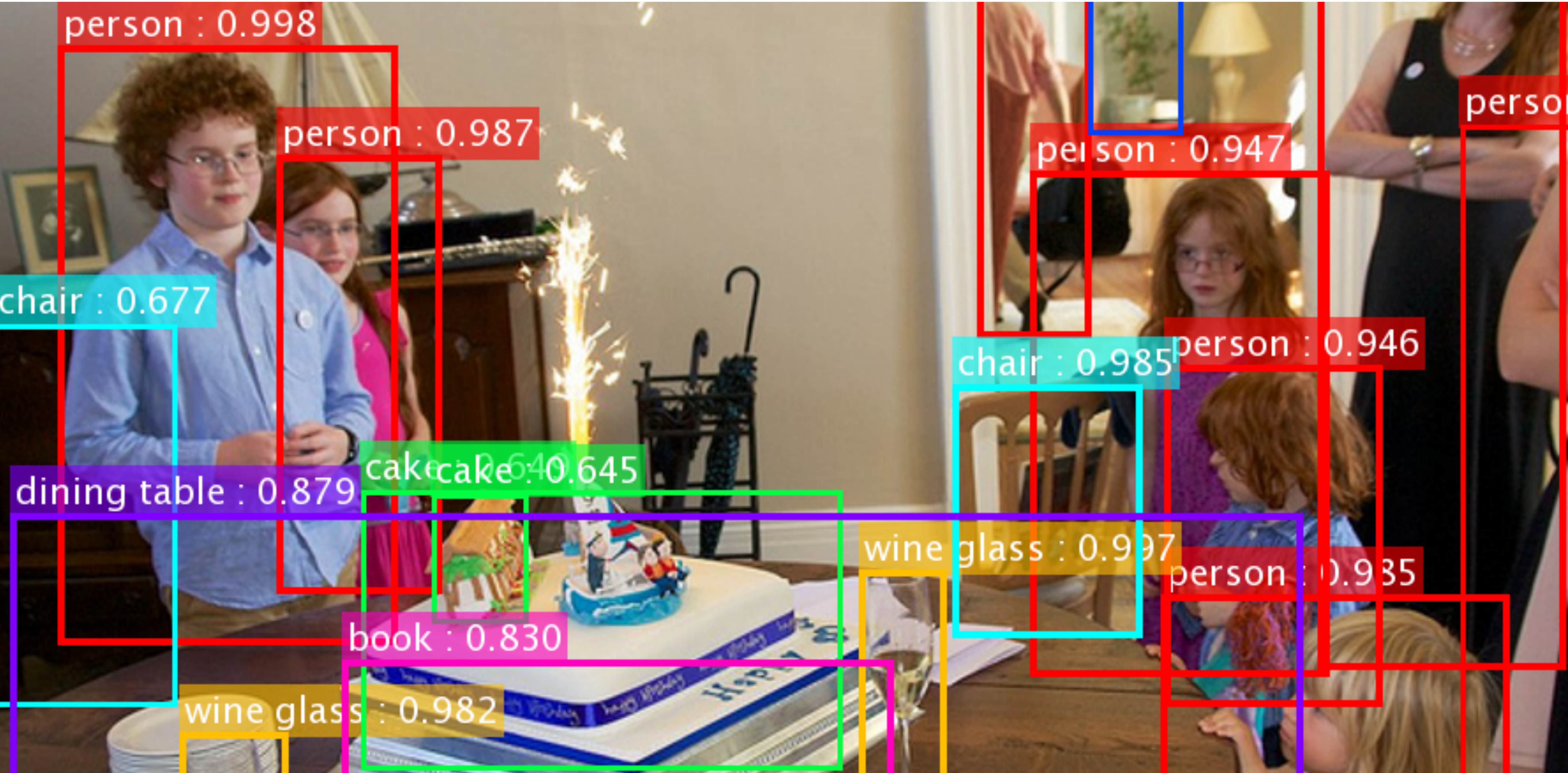






J. Long, E. Shelhamer, and T. Darrell. **Fully convolutional models for semantic segmentation.** In CVPR 2015
H. Noh, S. Hong, and B. Han. **Learning deconvolution network for semantic segmentation.** In ICCV 2015
O. Ronneberger, P. Fischer, and T. Brox. **U-net: Convolutional networks for biomedical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

Object Detection



MS COCO Dataset Images

MS COCO Annotations

- 80 different categories

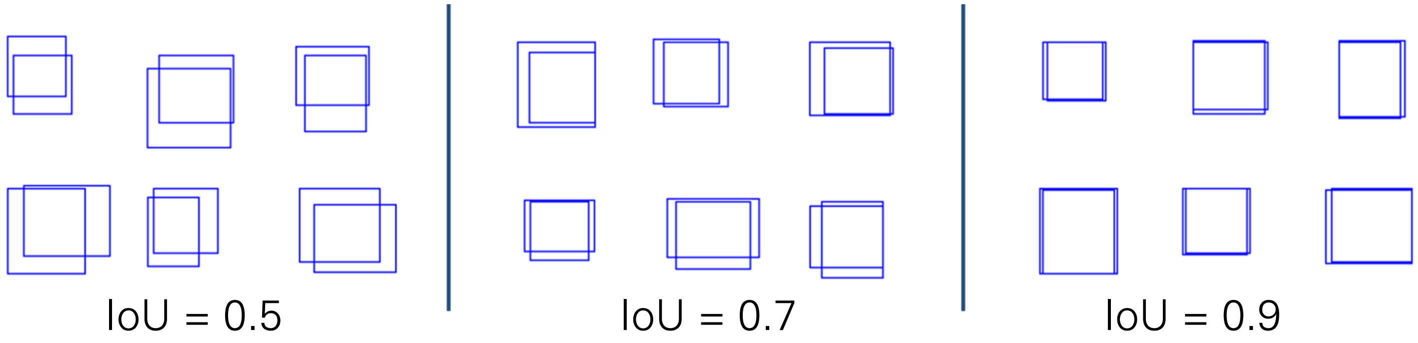
MS COCO

Dataset Images
+
Annotations

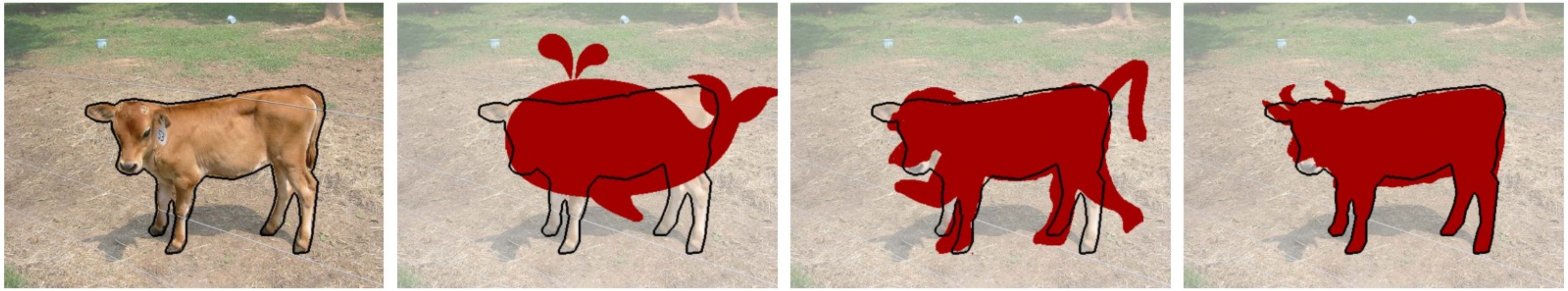
COCO Object Detection Average Precision (%)

- Area under a detector's precision-recall curve, averaged over…
  - Object categories
  - **True positive overlap requirement** (IoU from 0.5 to 0.95; see below)

boxes



masks



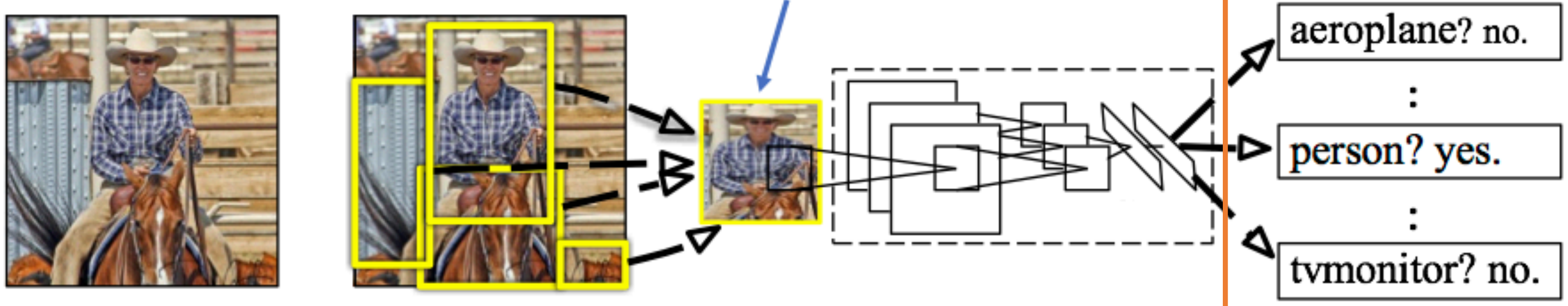
Ground truth | IoU = 0.55 | IoU = 0.70 | IoU = 0.91

Figure credits: Dollár and Zitnick (top), Krähenbühl and Kulton (bottom)

More than one "stage" (≈proposal based; but doesn't require proposals)

classification of reduced output space elements

RoI transformation



Cascade-like reduction in output space

Input image

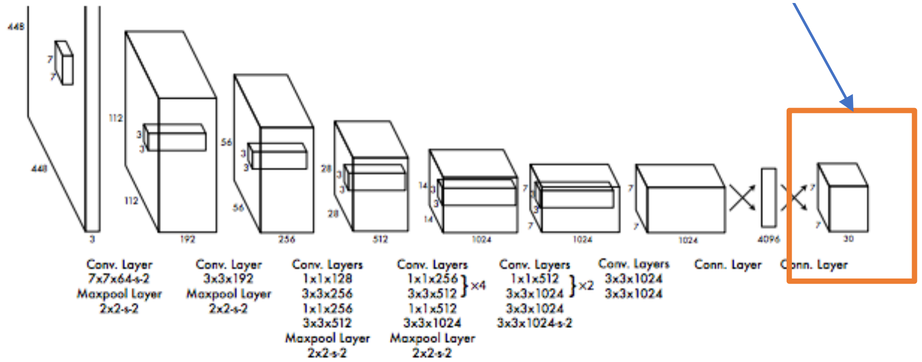
Object / region proposals

Deep Learning region classifier

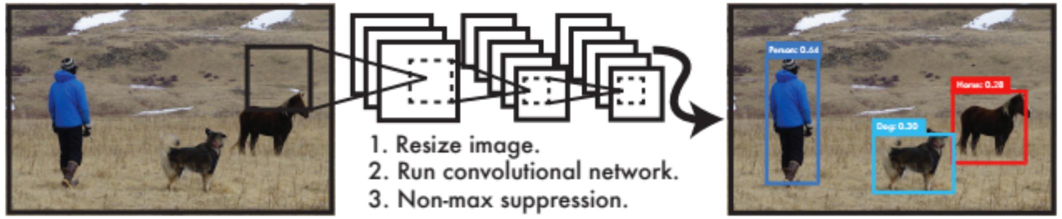
Region classification, box regression

One stage

Direct classification
Of all output space elements



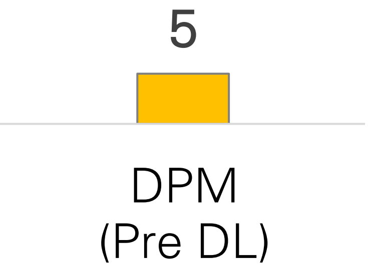
Redmond et al. You Only Look Once:
Unified Real-time Object Detection. In CVPR 2016



"You only look once»
"Single shot"

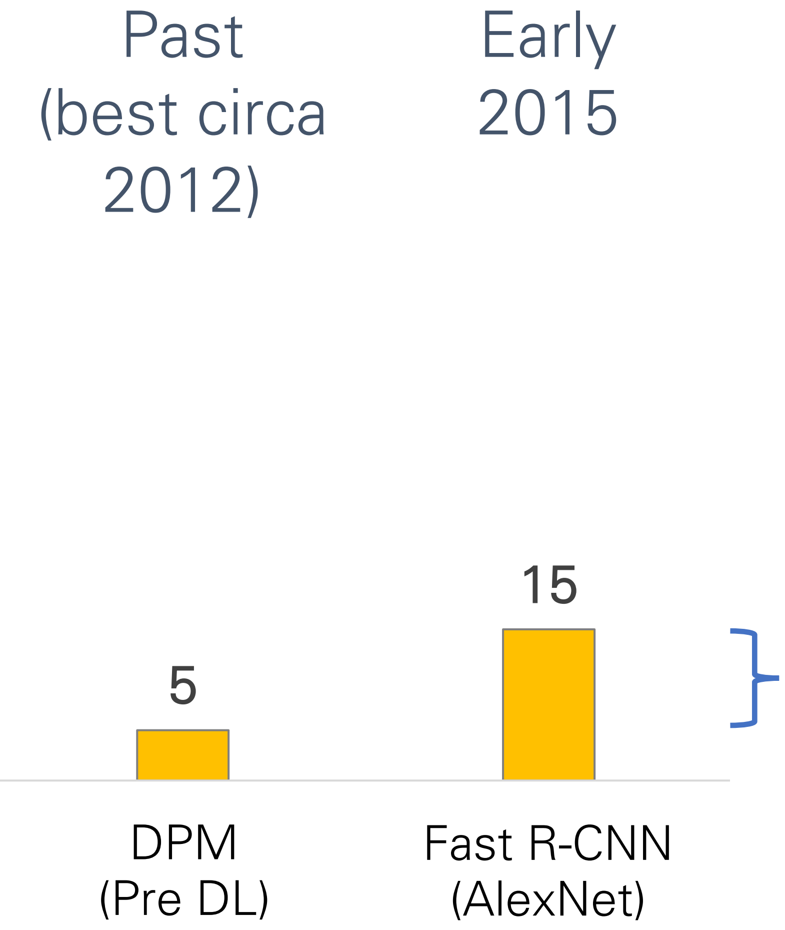
COCO Object Detection Average Precision (%)

Past
(best circa
2012)



Felzenszwalb, Girshick, McAllester, Ramanan. Object Detection with Discriminatively Trained Part Based Models. PAMI 2010.

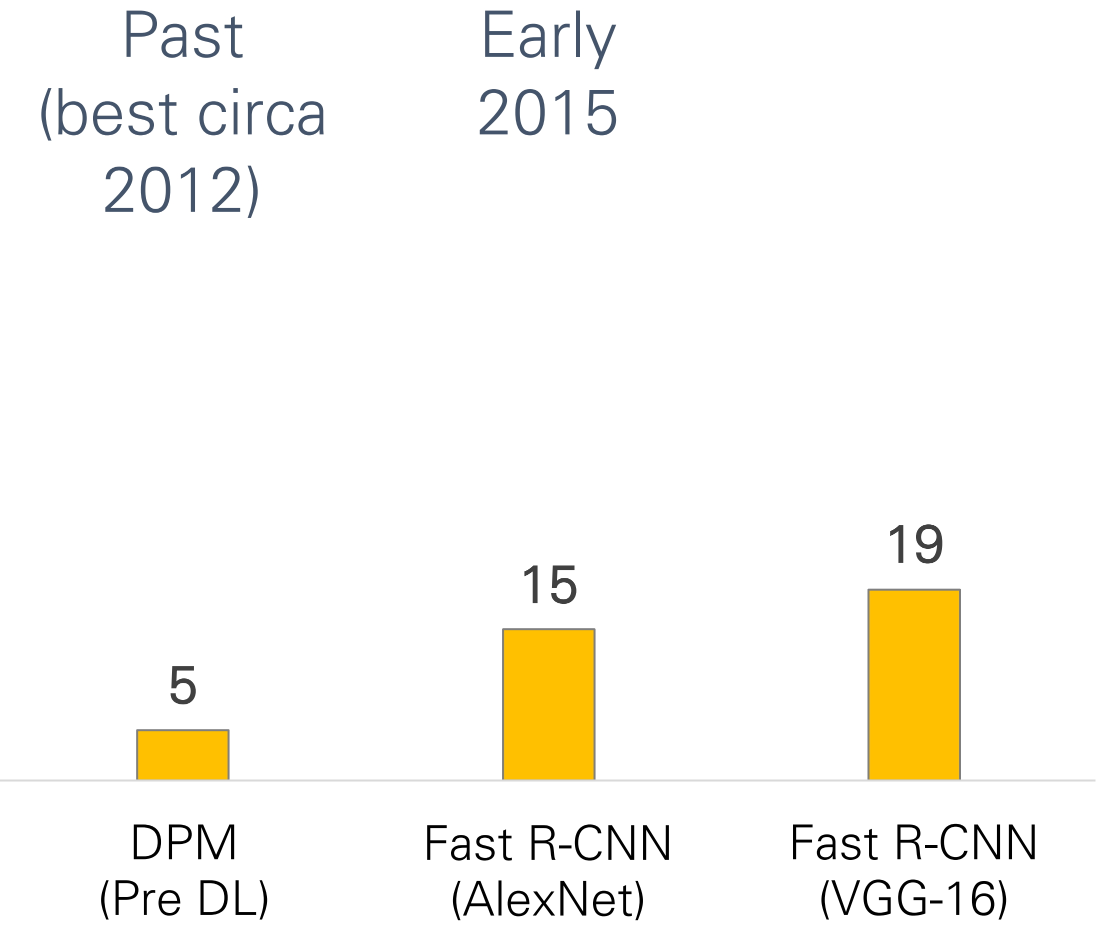
COCO Object Detection Average Precision (%)



Movement to DL methods

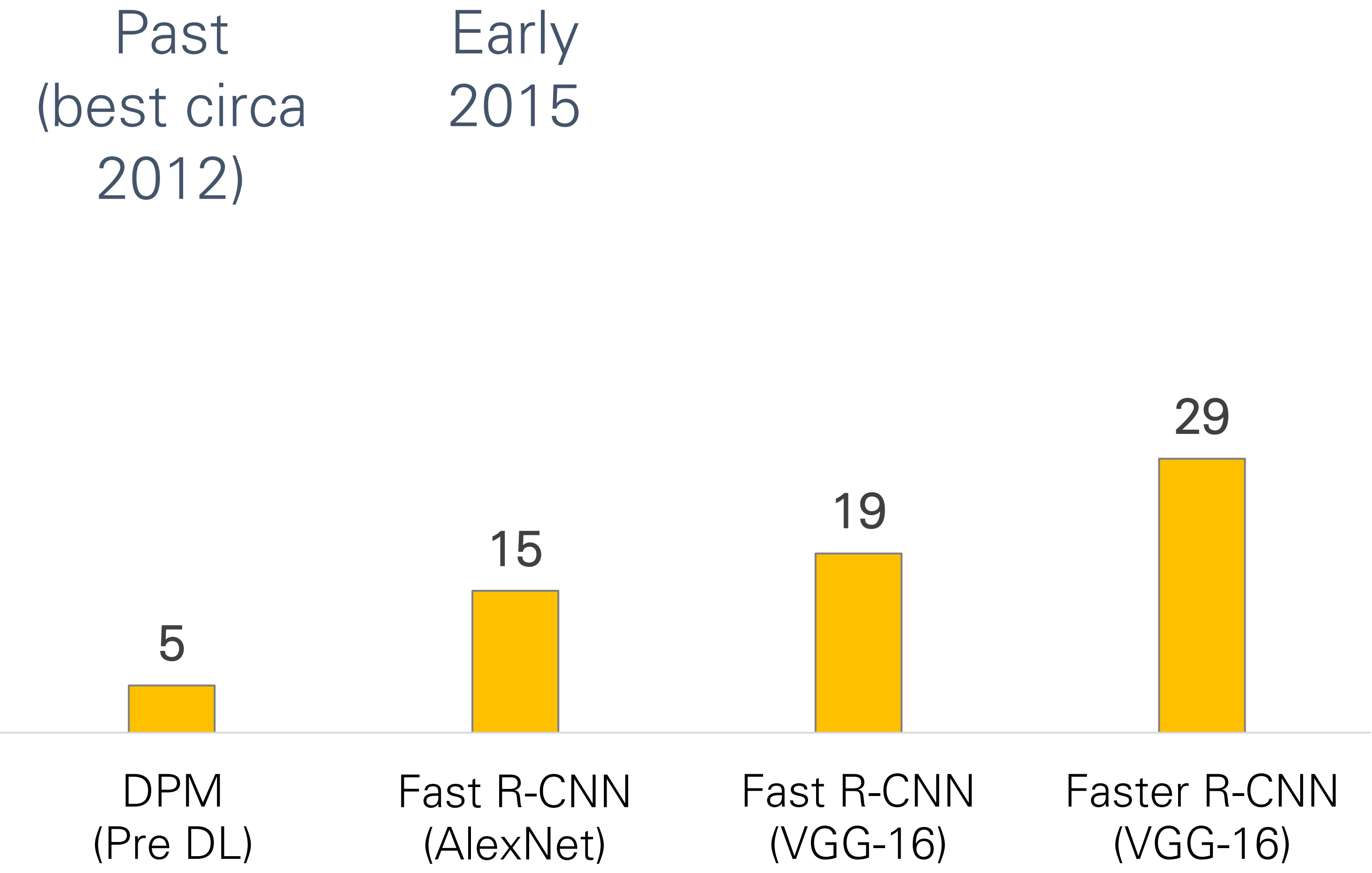
Girshick. Fast R-CNN. ICCV 2015.

COCO Object Detection Average Precision (%)



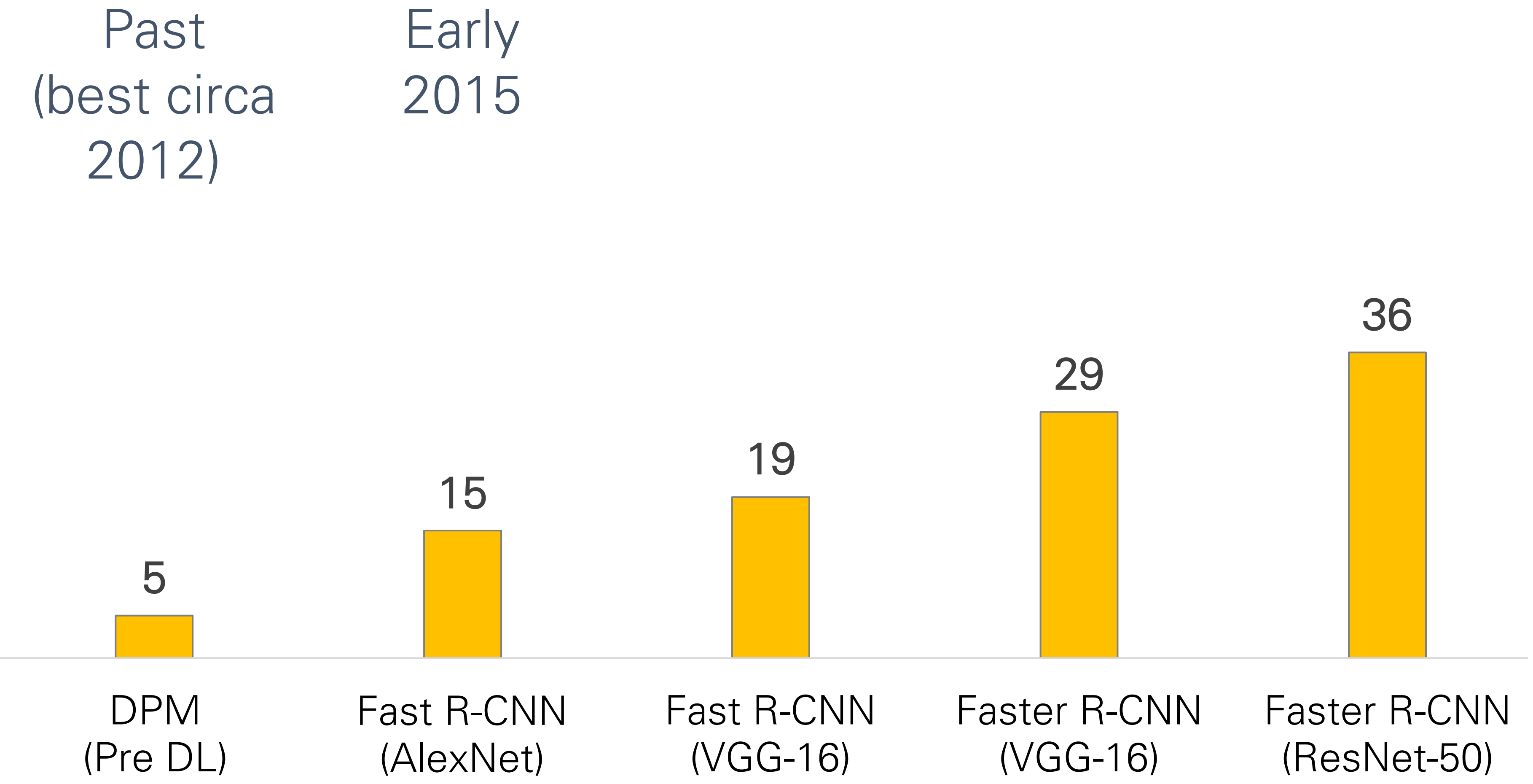
Girshick. Fast R-CNN. ICCV 2015.

COCO Object Detection Average Precision (%)



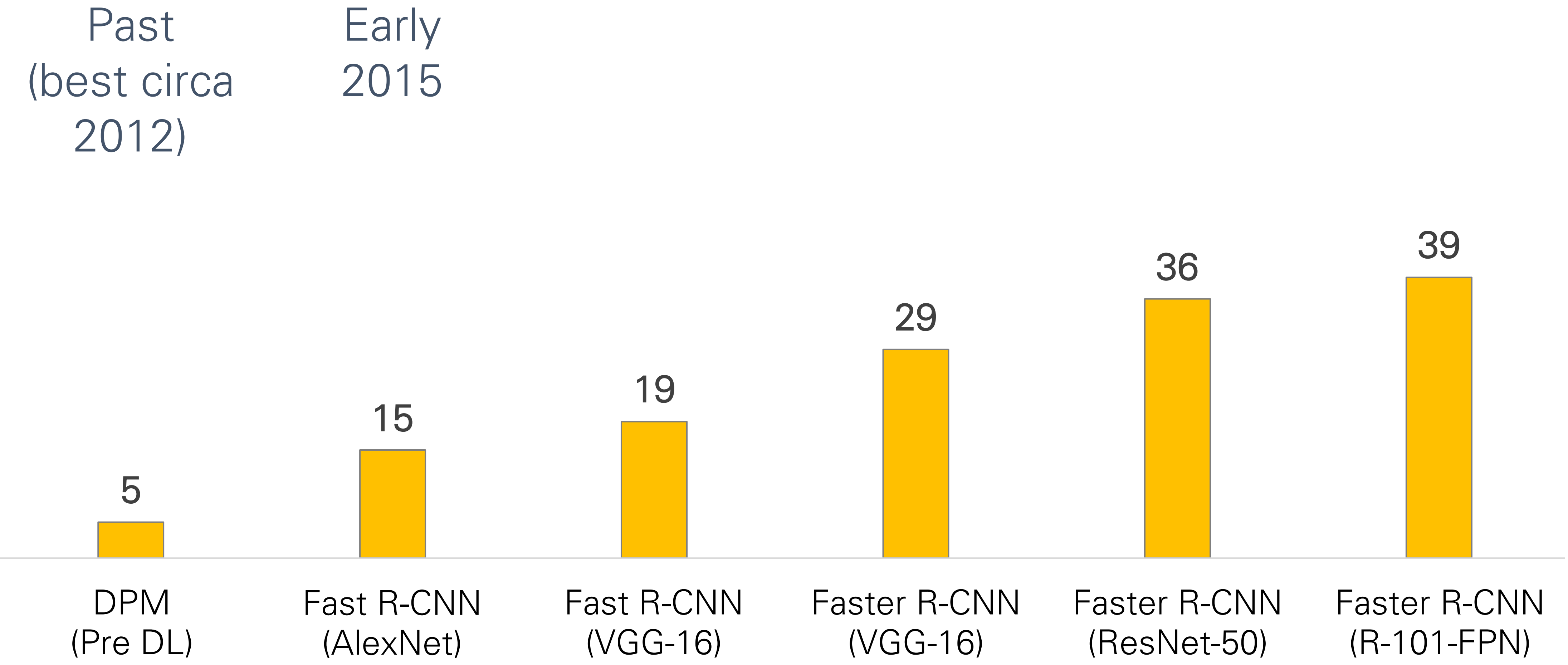
Ren, He, Girshick, Sun. Faster R-CNN: Towards Real-Time Object Detection. NIPS 2015.

COCO Object Detection Average Precision (%)



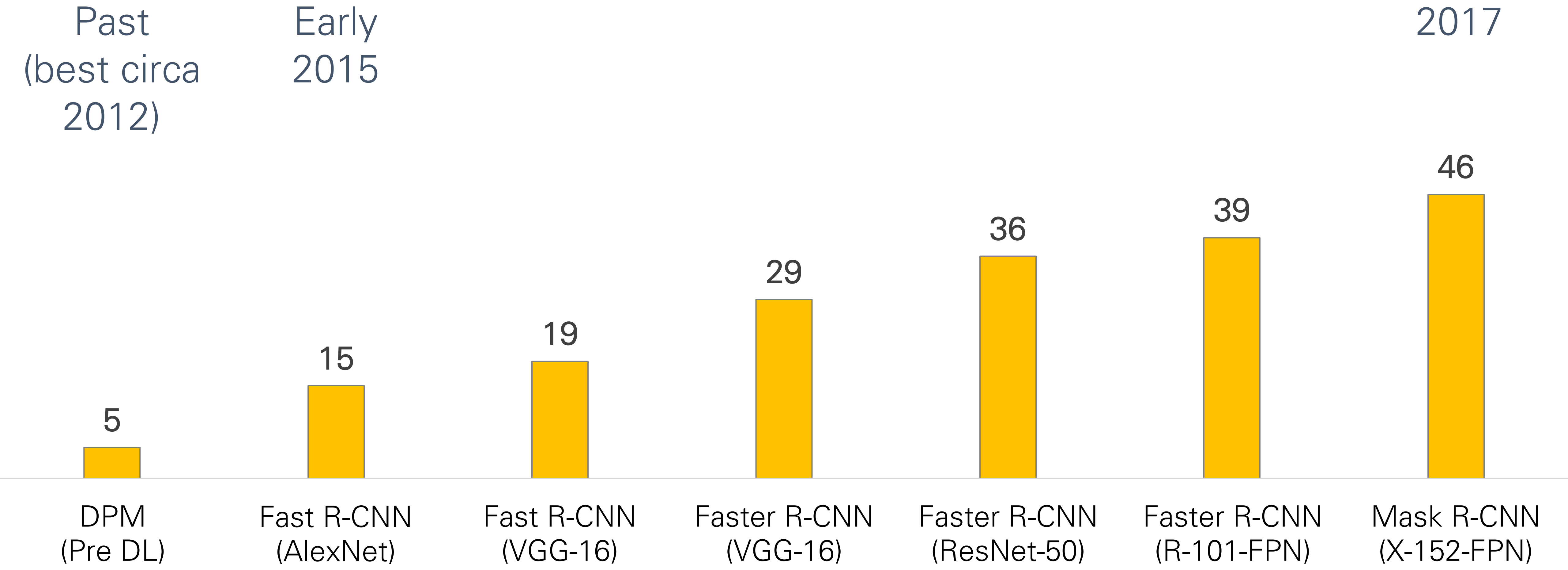
Ren, He, Girshick, Sun. Faster R-CNN: Towards Real-Time Object Detection. NIPS 2015.

COCO Object Detection Average Precision (%)



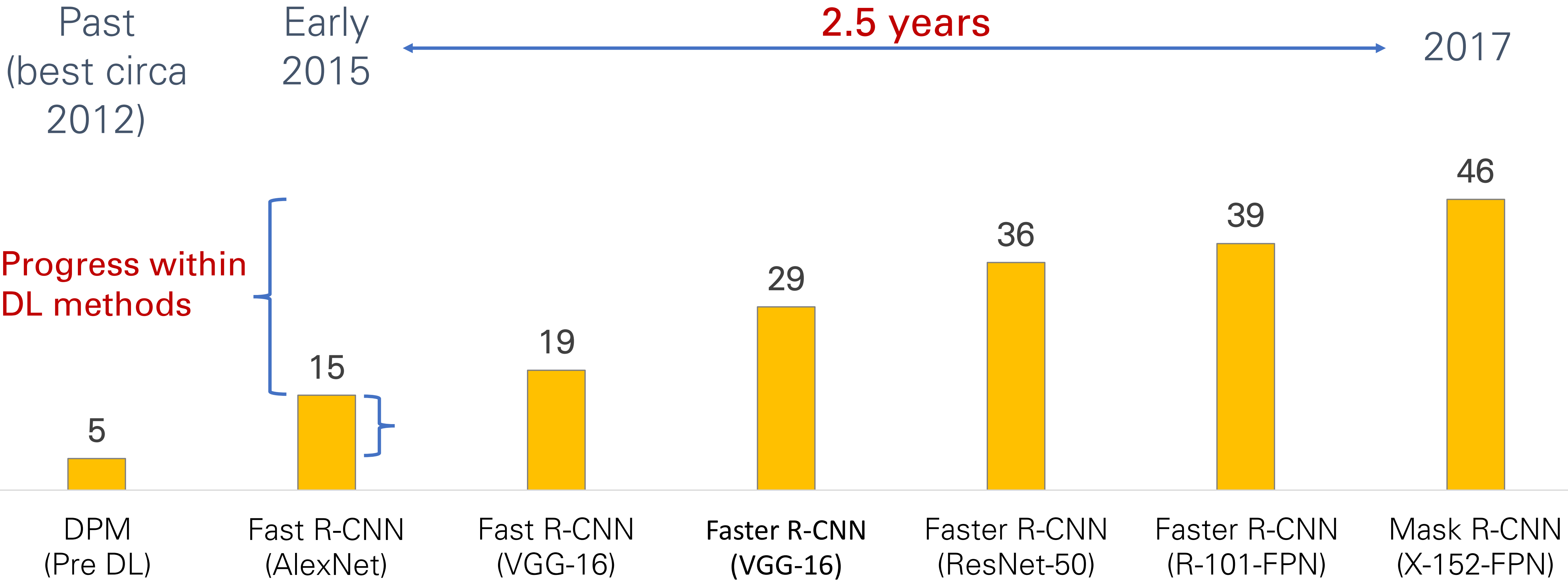
Lin et al. Feature Pyramid Networks. CVPR 2017.

COCO Object Detection Average Precision (%)



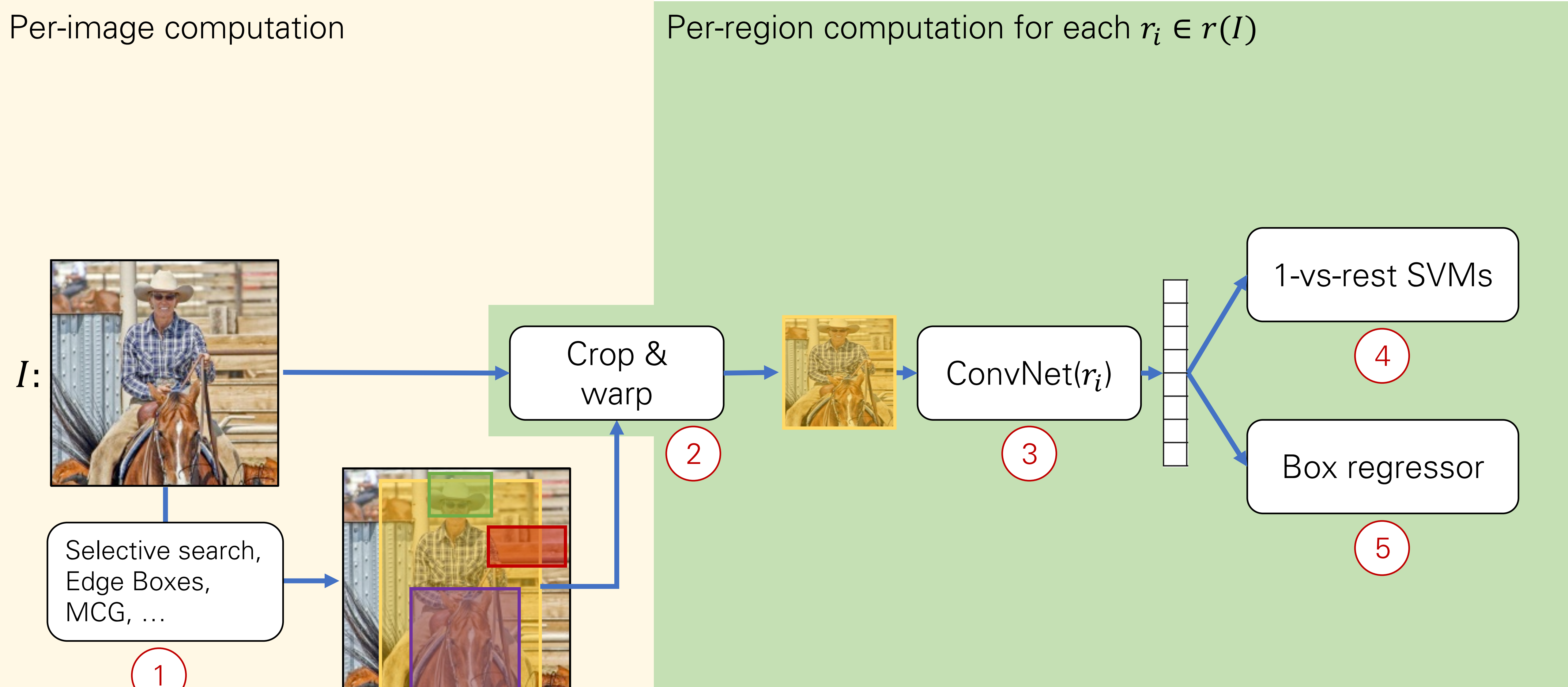
He, Gkioxari, Dollár, Girshick. Mask R-CNN. ICCV 2017.

COCO Object Detection Average Precision (%)



"Slow" R-CNN

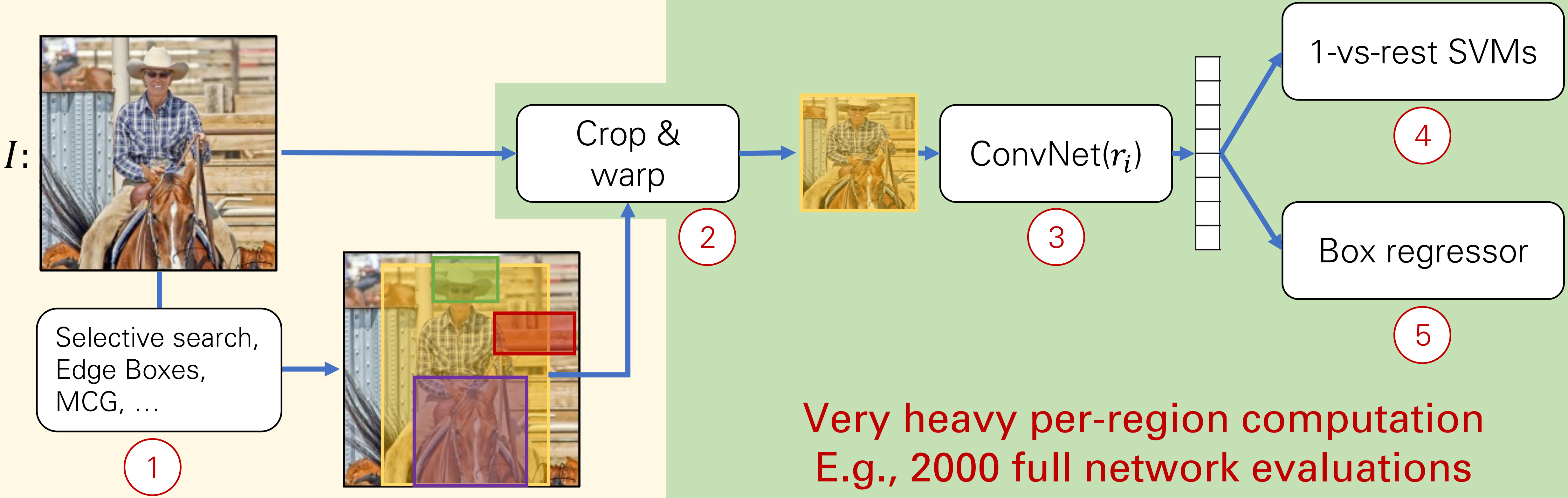
Girshick, Donahue, Darrell, Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. CVPR 2014.



"Slow" R-CNN

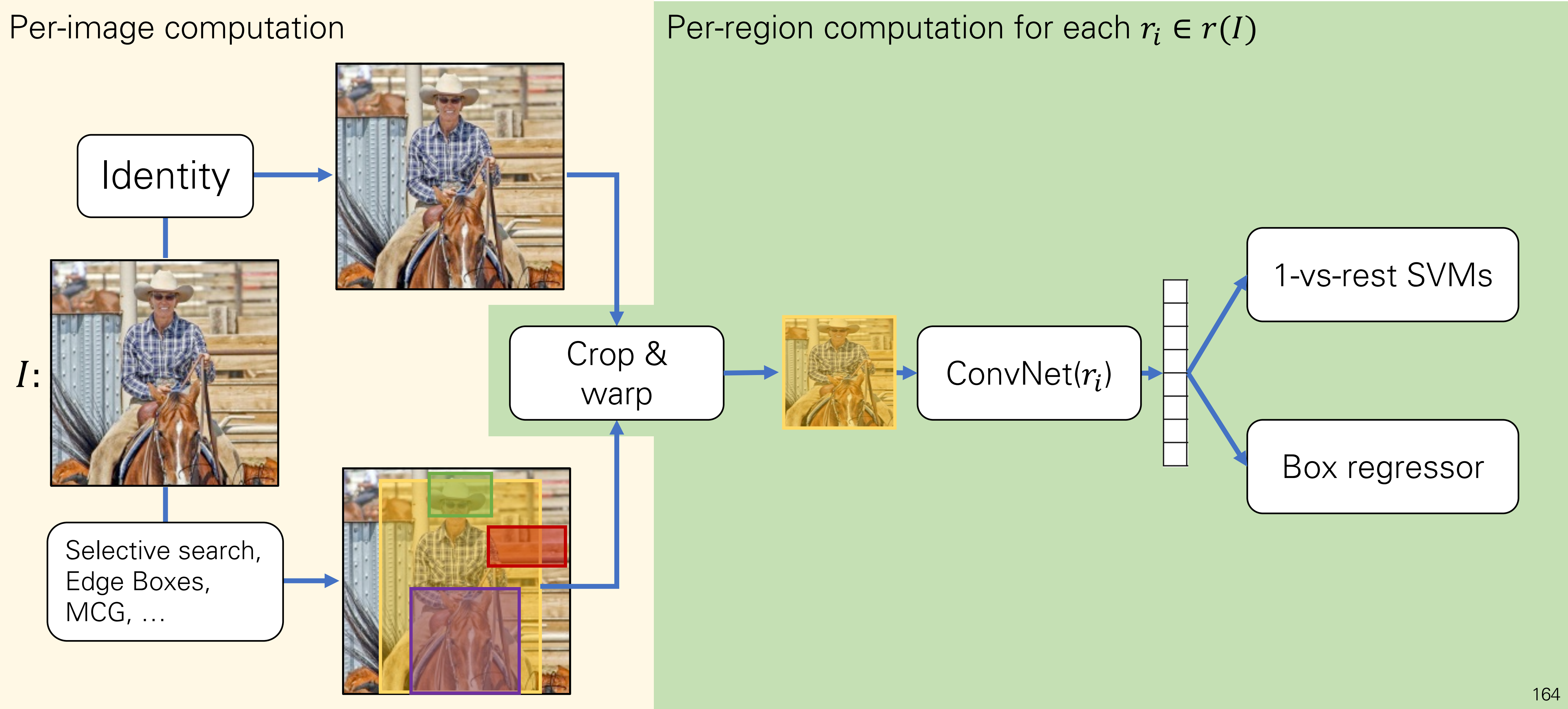
Per-image computation

Per-region computation for each $r_i \in r(I)$

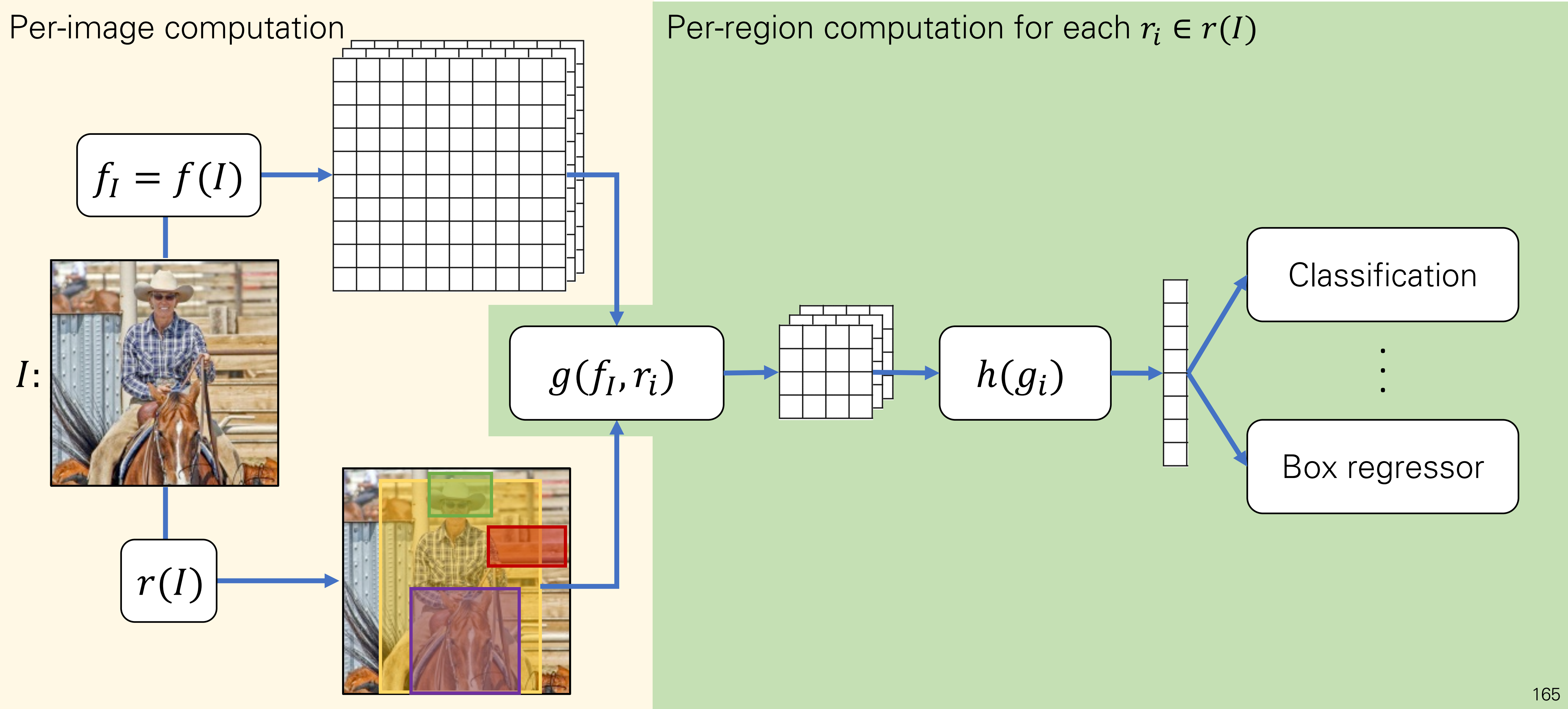


Very heavy per-region computation
E.g., 2000 full network evaluations

"Slow" R-CNN

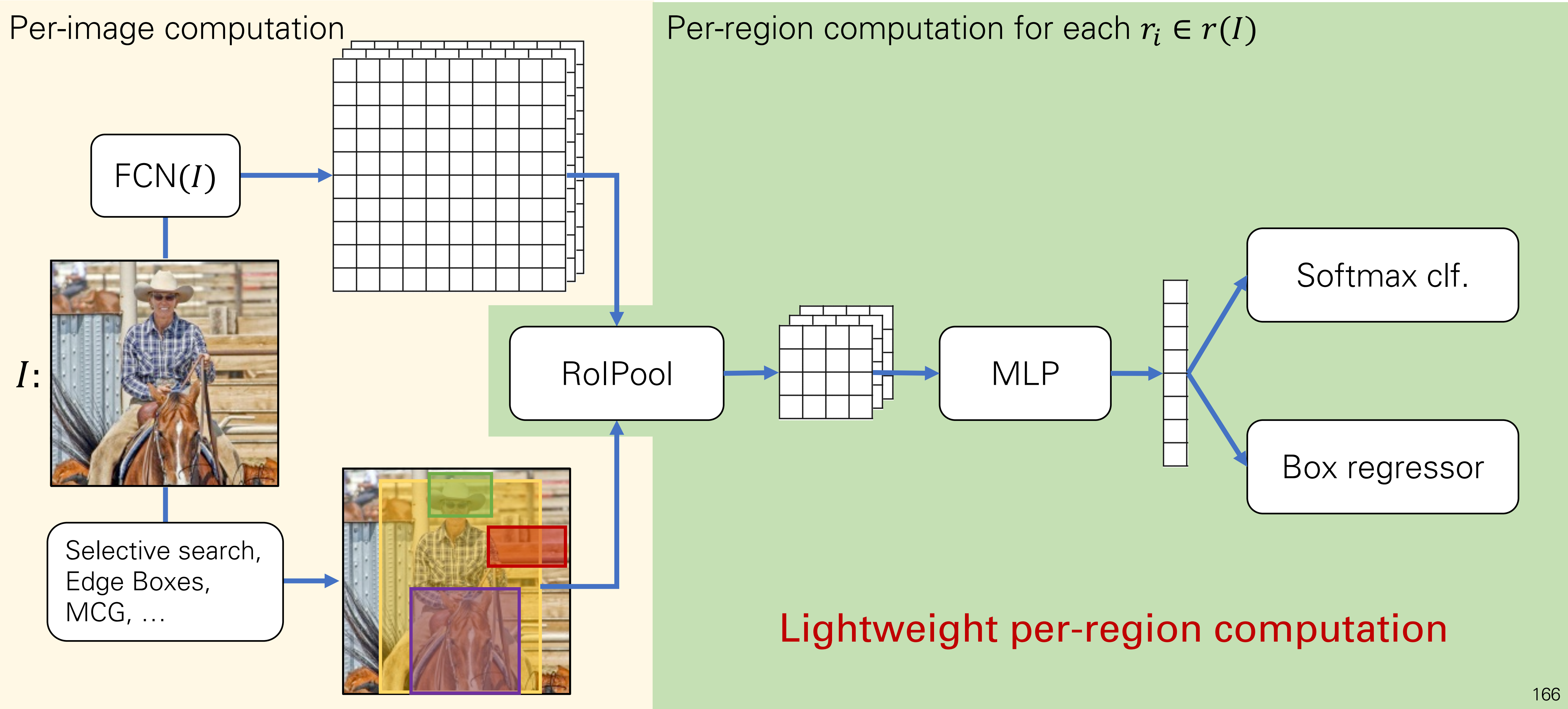

Per-image computation
Per-region computation for each $r_i \in r(I)$
Identity
I:
Selective search,
Edge Boxes,
MCG, …
Crop &
warp
ConvNet(r_i)
1-vs-rest SVMs
Box regressor

Generalized R-CNN Approach to Detection

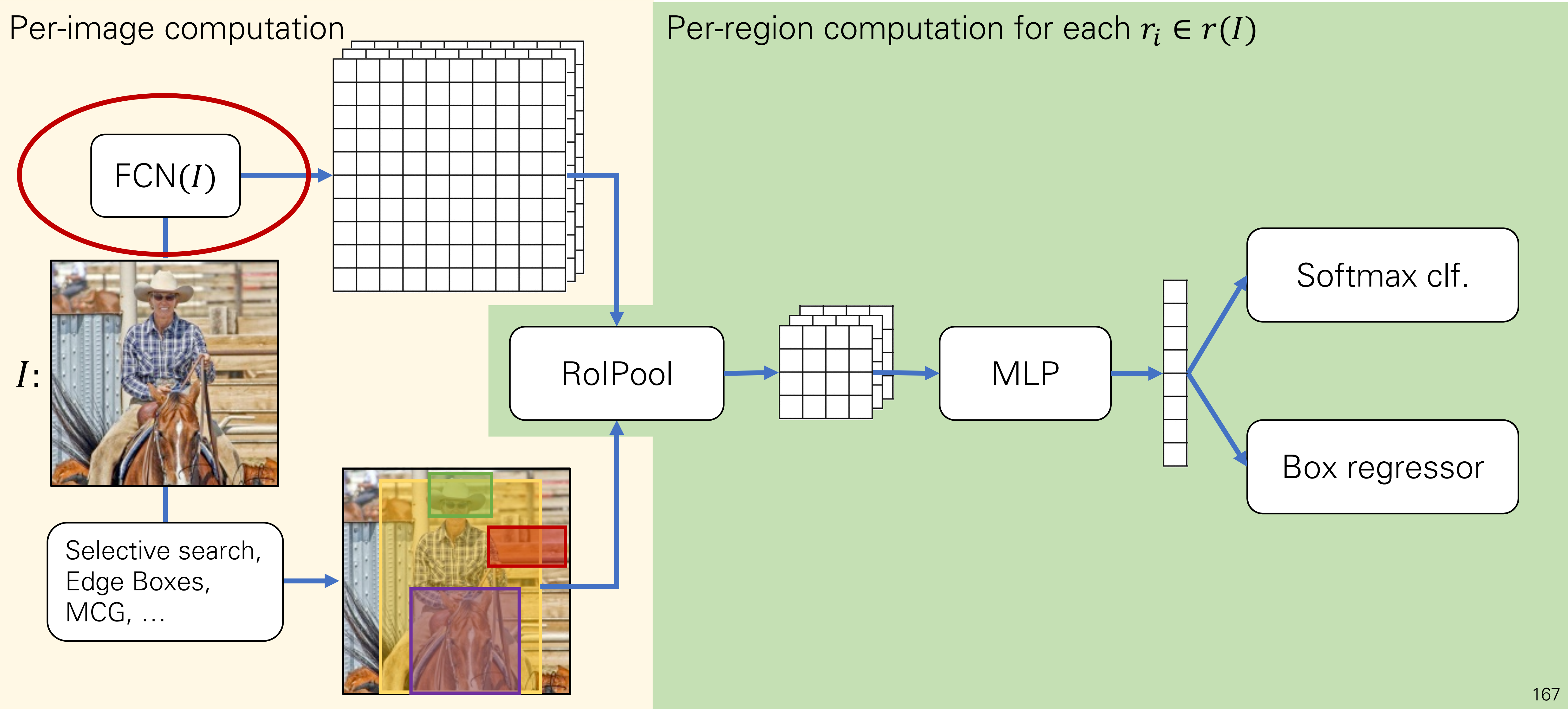


Girshick. Fast R-CNN. ICCV 2015.

Fast R-CNN



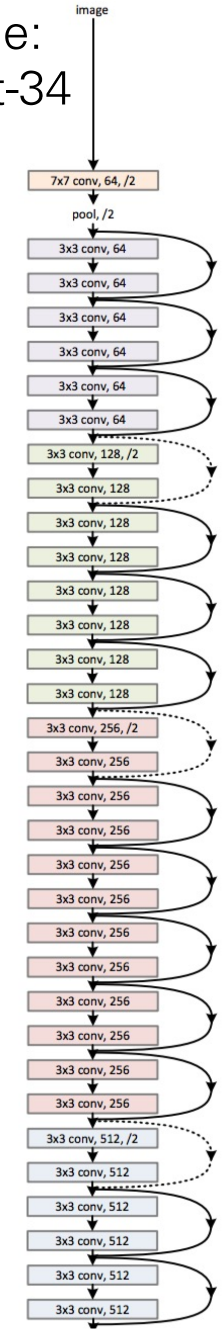
Fast R-CNN



Per-image computation
Per-region computation for each $r_i \in r(I)$
FCN(I)
I:
Selective search,
Edge Boxes,
MCG, …
RoIPool
MLP
Softmax clf.
Box regressor

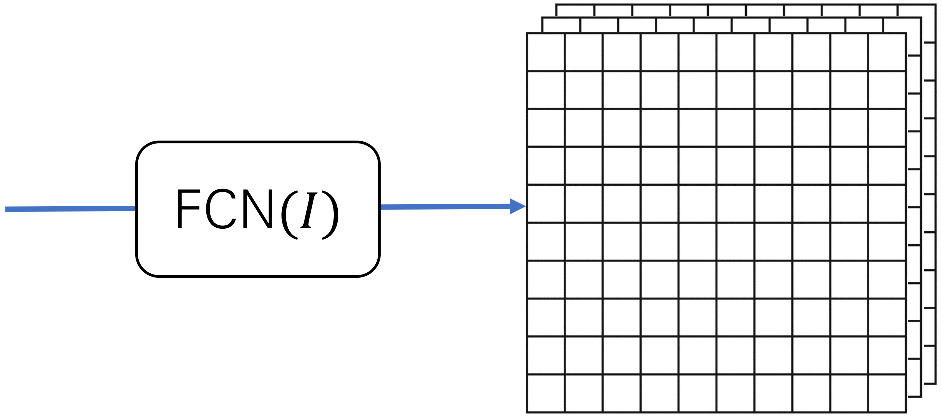
Whole-image FCN

Example:
ResNet-34



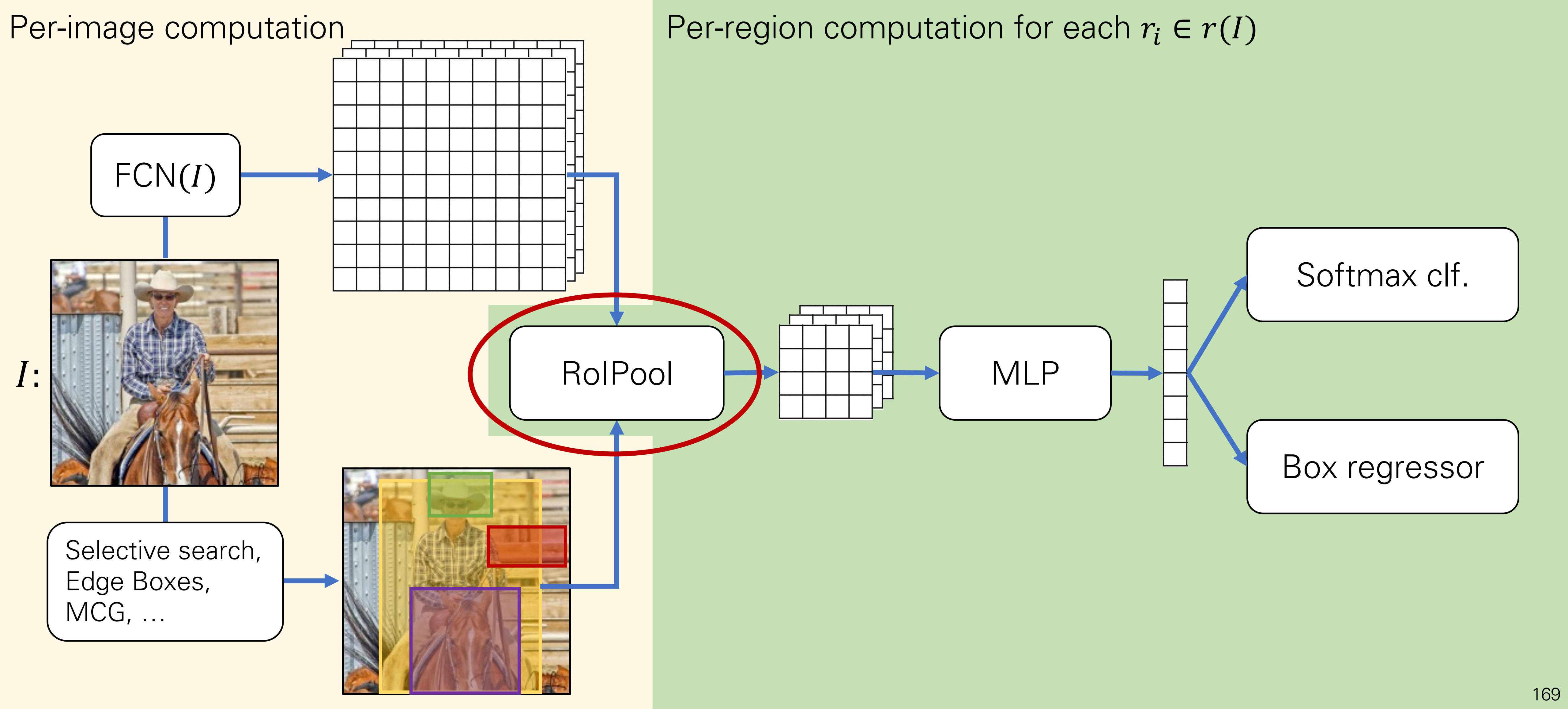
- Use **any standard ConvNet** as the "**backbone architecture**"
  - AlexNet, VGG, ResNet, Inception, Inception-ResNet, ResNeXt, DenseNet, …
  - Use the first N layers with spatial extent (e.g., up to "conv5")

I:



Example feature map dimensions:
(512, H/16, W/16)

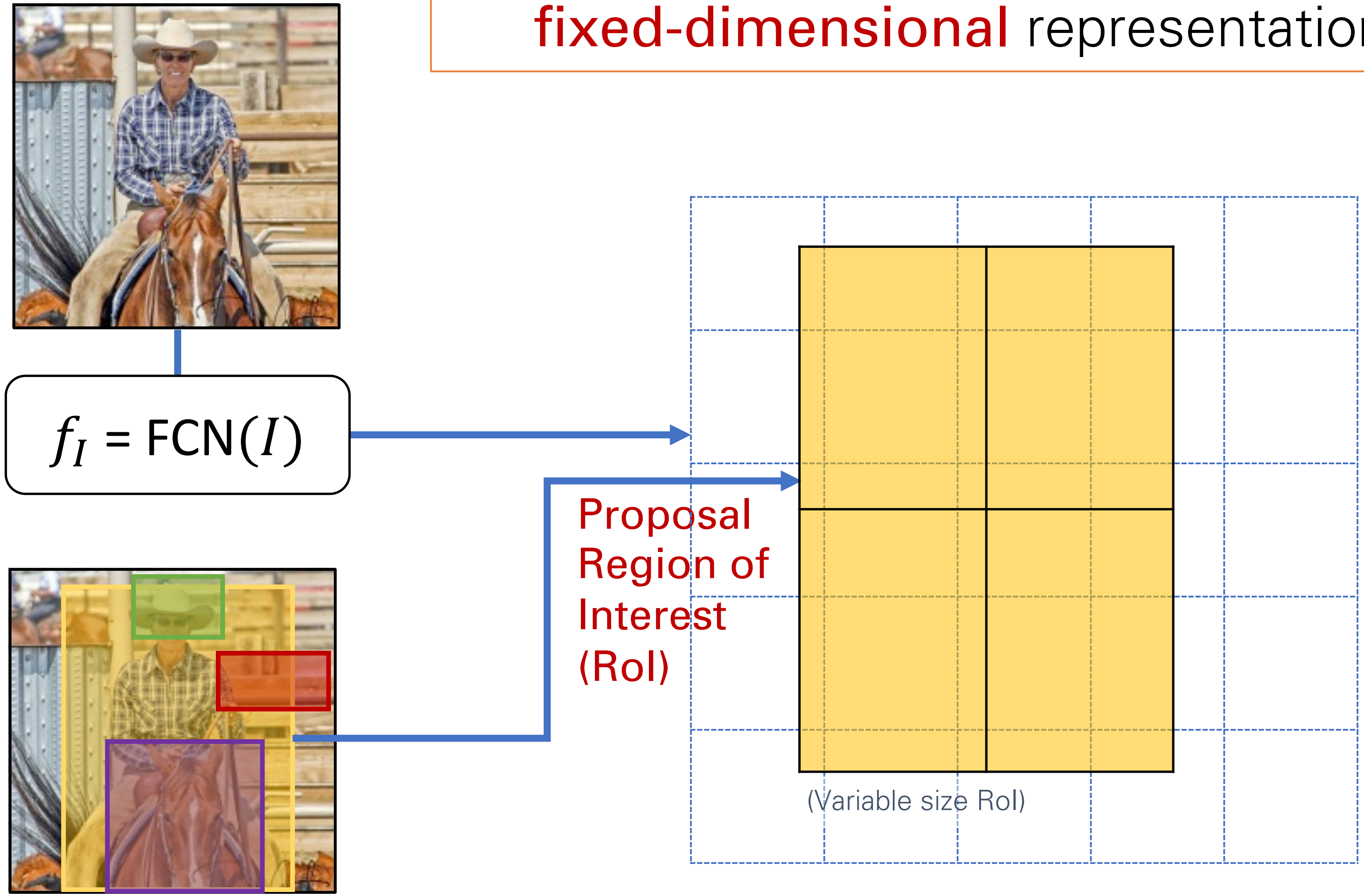
Fast R-CNN



Per-image computation
Per-region computation for each $r_i \in r(I)$
FCN(I)
I:
Selective search,
Edge Boxes,
MCG, …
RoIPool
MLP
Softmax clf.
Box regressor

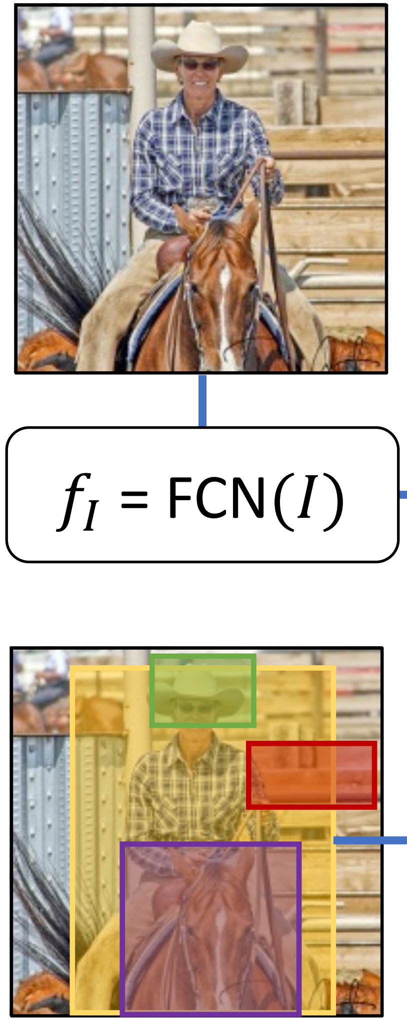
RoIPool (on each Proposal)

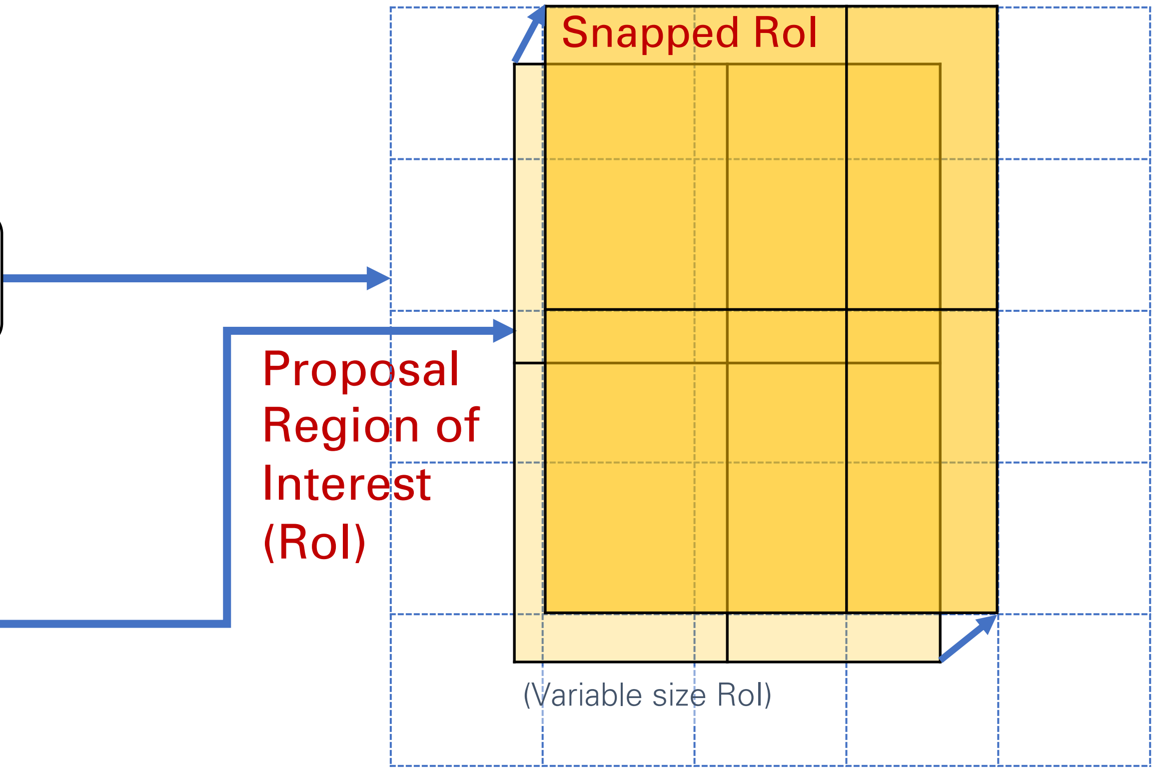
Transform **arbitrary size proposal** into a **fixed-dimensional** representation (e.g., 2x2)



RoIPool (on each Proposal)

Transform **arbitrary size proposal** into a **fixed-dimensional** representation (e.g., 2x2)

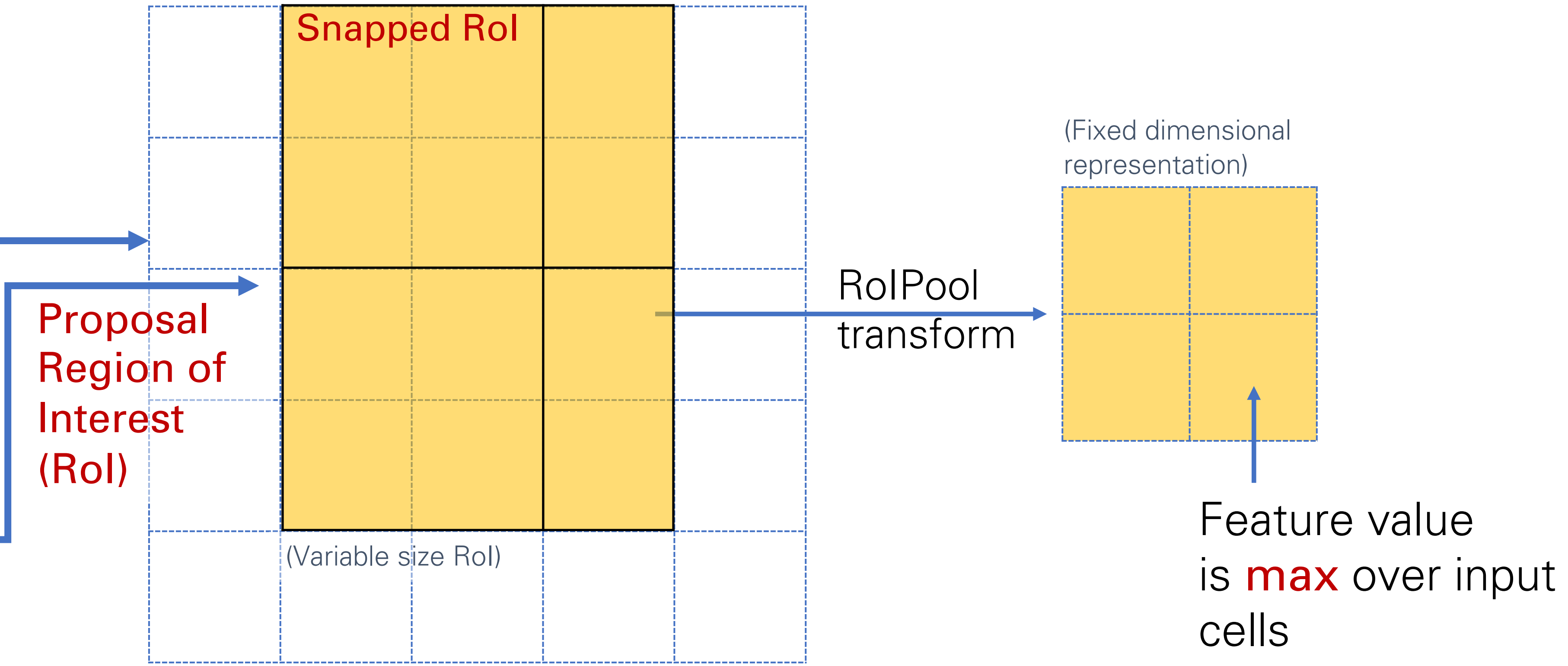




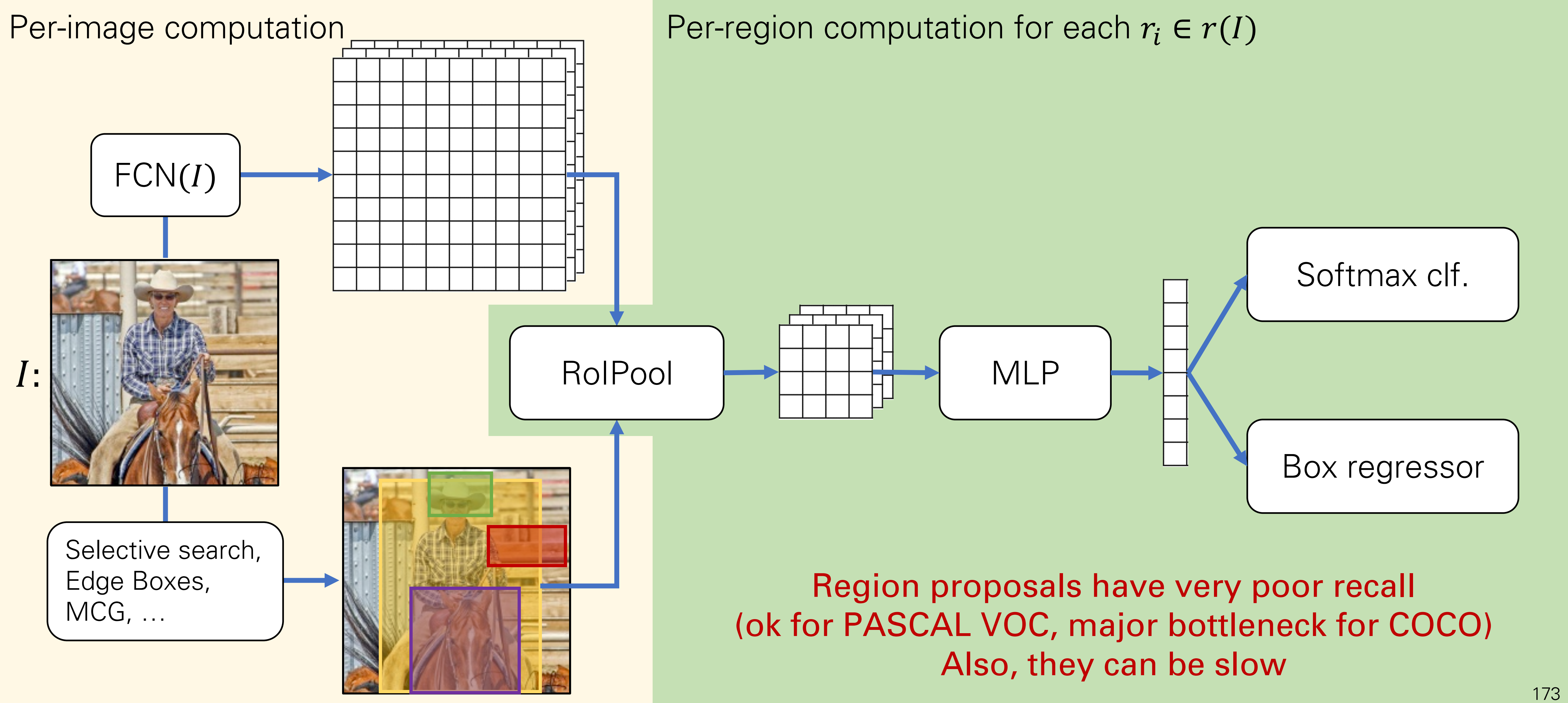
RoIPool (on each Proposal)

Transform **arbitrary size proposal** into a **fixed-dimensional** representation (e.g., 2x2)

f_I = FCN(I)



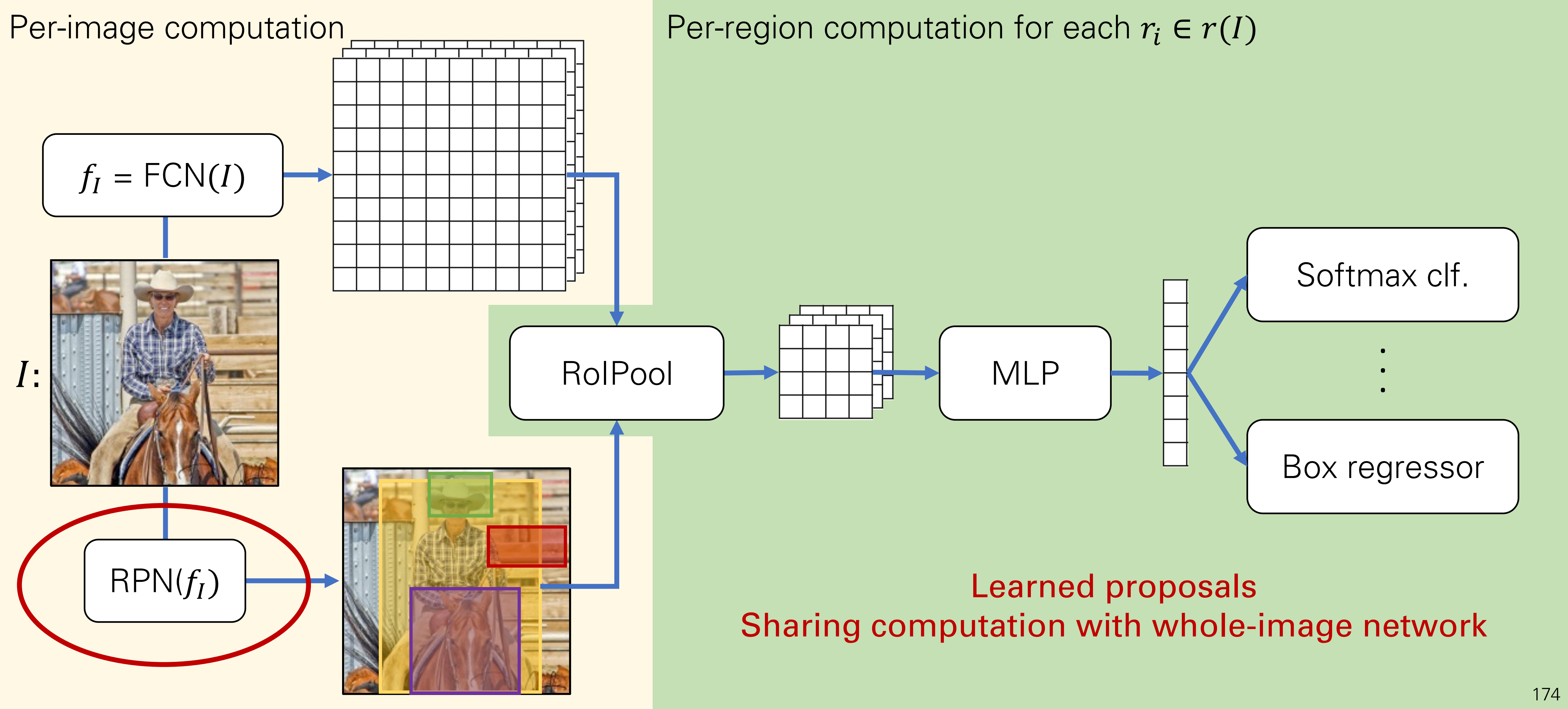
Fast R-CNN



Region proposals have very poor recall
(ok for PASCAL VOC, major bottleneck for COCO)
Also, they can be slow

Faster R-CNN

Ren, He, Girshick, Sun. Faster R-CNN: Towards Real-Time Object Detection. NIPS 2015.

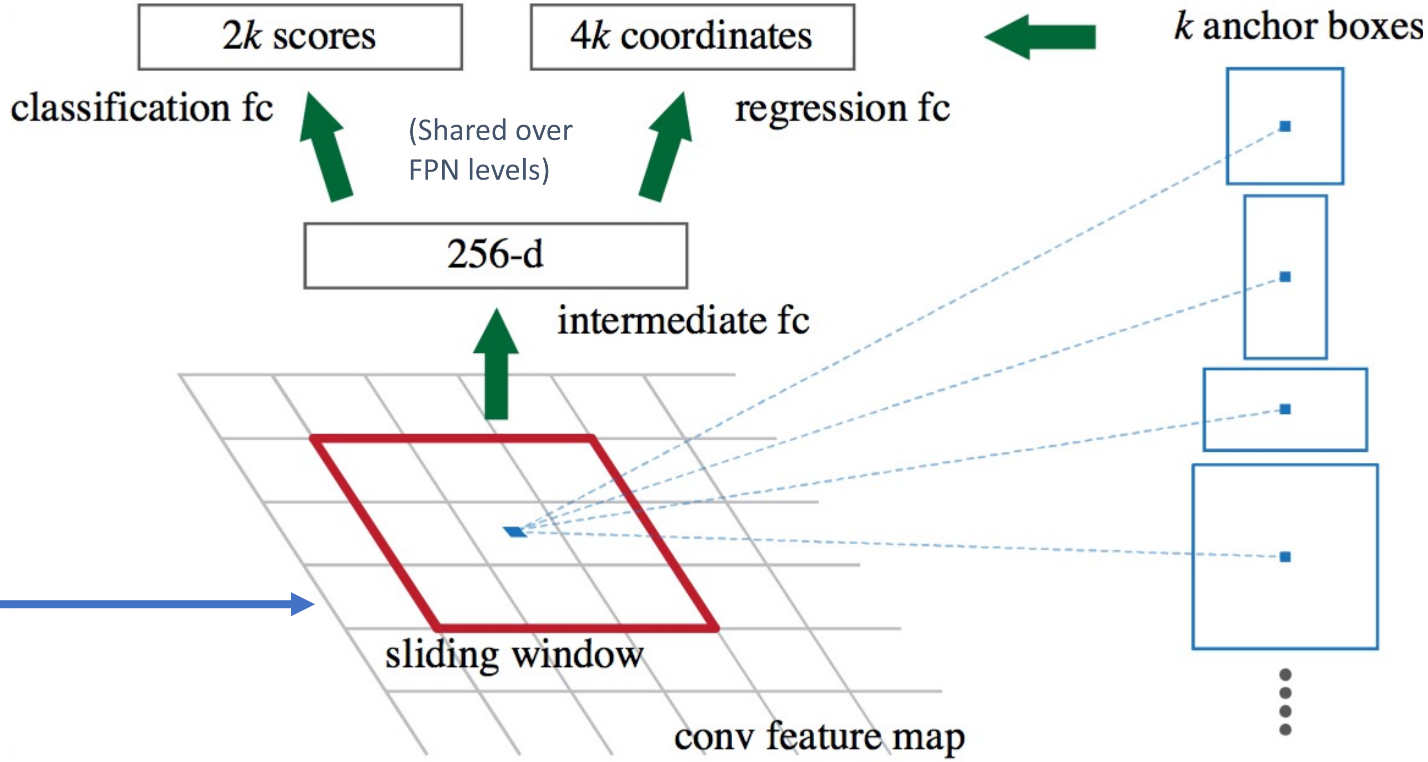


Learned proposals
Sharing computation with whole-image network

Region Proposal Network (RPN)

Proposals = sliding window object/not-object classifier + box regression **inside the same network**

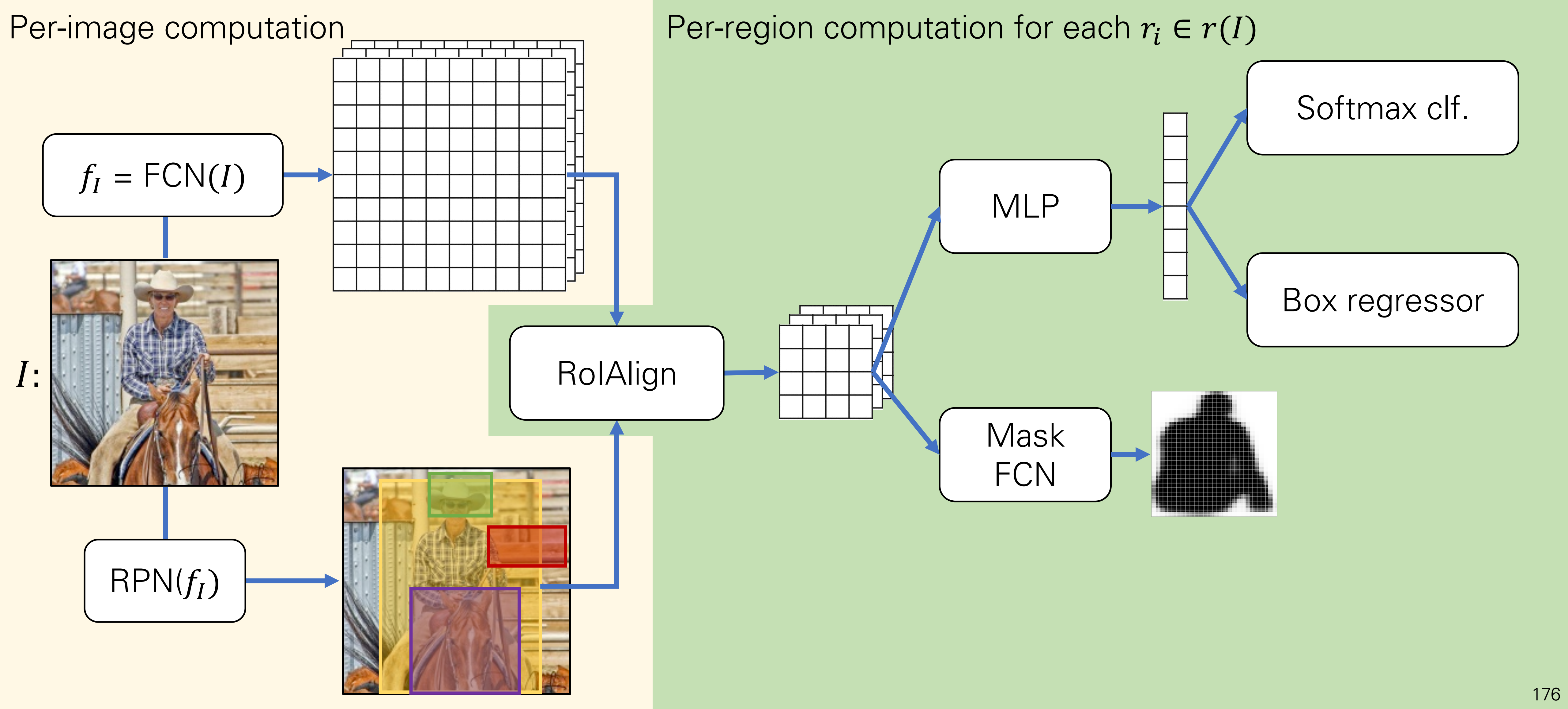
f_I = FCN(I)



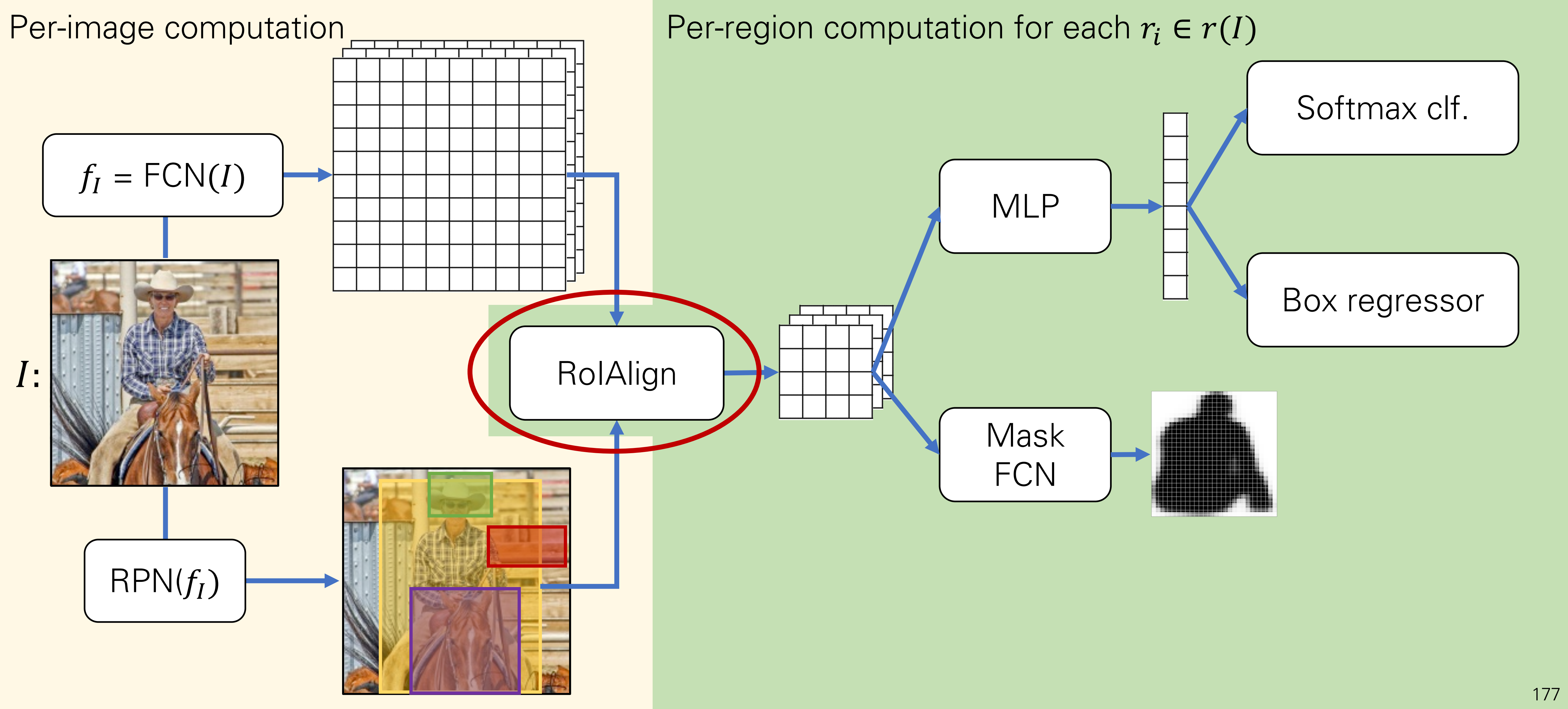
Anchors are prototypical object boxes

Mask R-CNN

He, Gkioxari, Dollár, Girshick. Mask R-CNN. ICCV 2017.

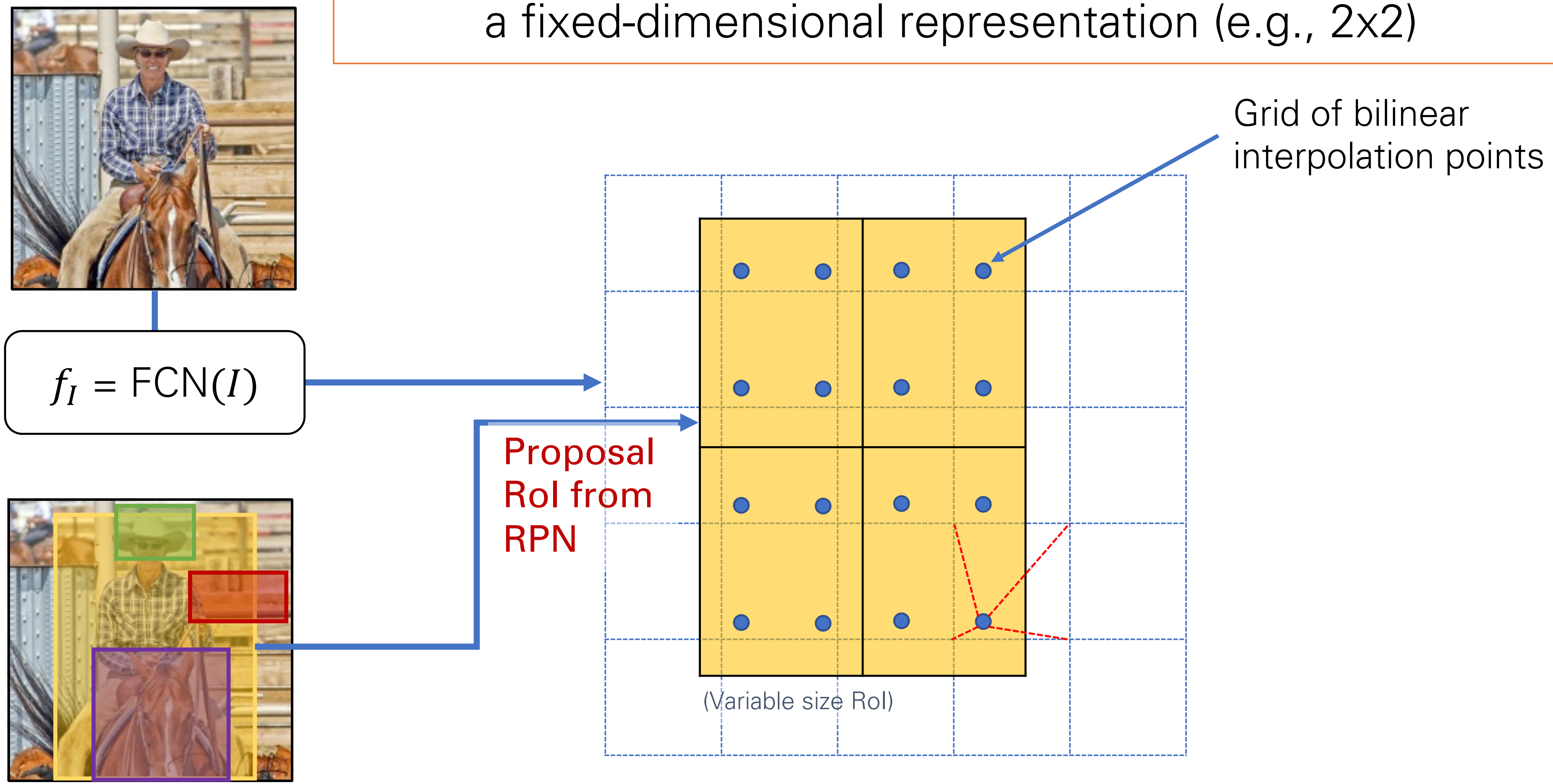


Mask R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
f_I = FCN(I)
I:
RPN(f_I)
RoIAlign
MLP
Mask FCN
Softmax clf.
Box regressor

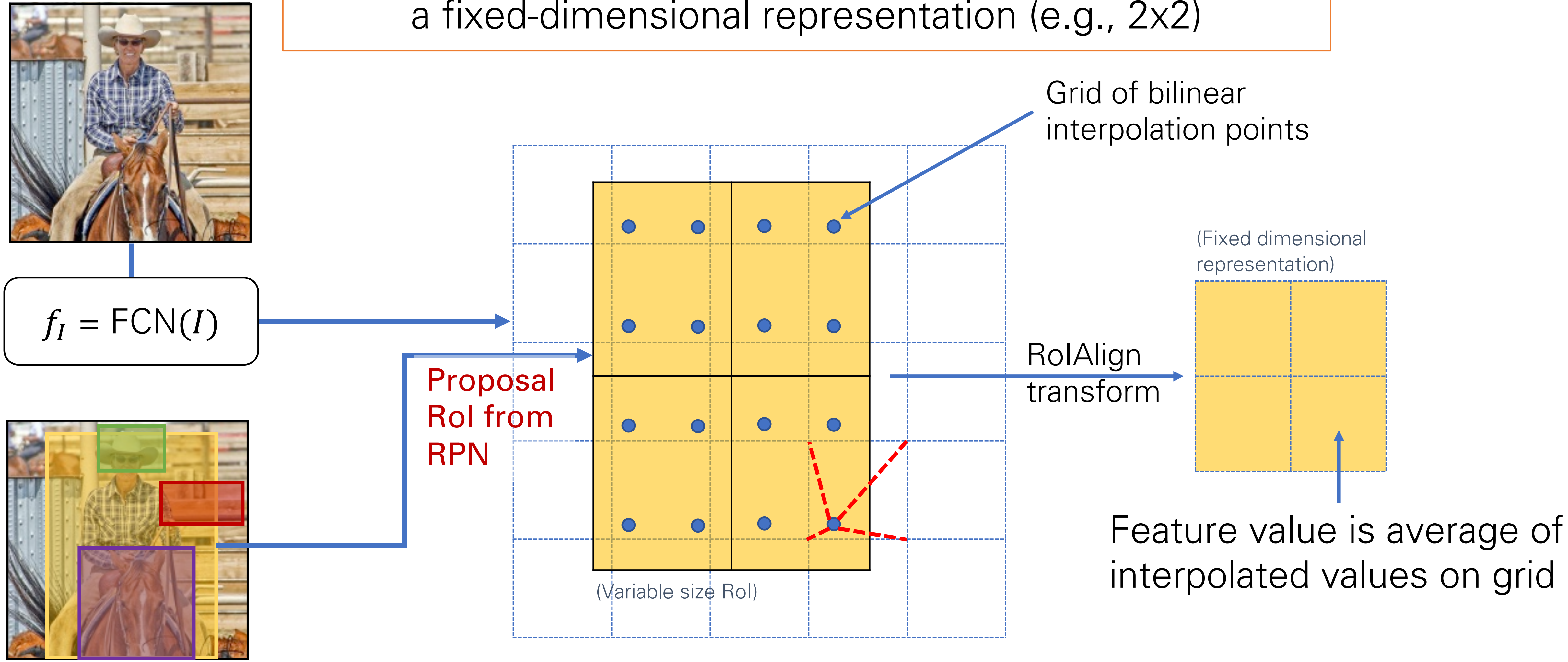
RoIAlign (on each Proposal)

Smoothly transform RoI features into
a fixed-dimensional representation (e.g., 2x2)



RoIAlign (on each Proposal)

Smoothly transform RoI features into
a fixed-dimensional representation (e.g., 2x2)



Compare to RoIPool

Preserve alignment or not?

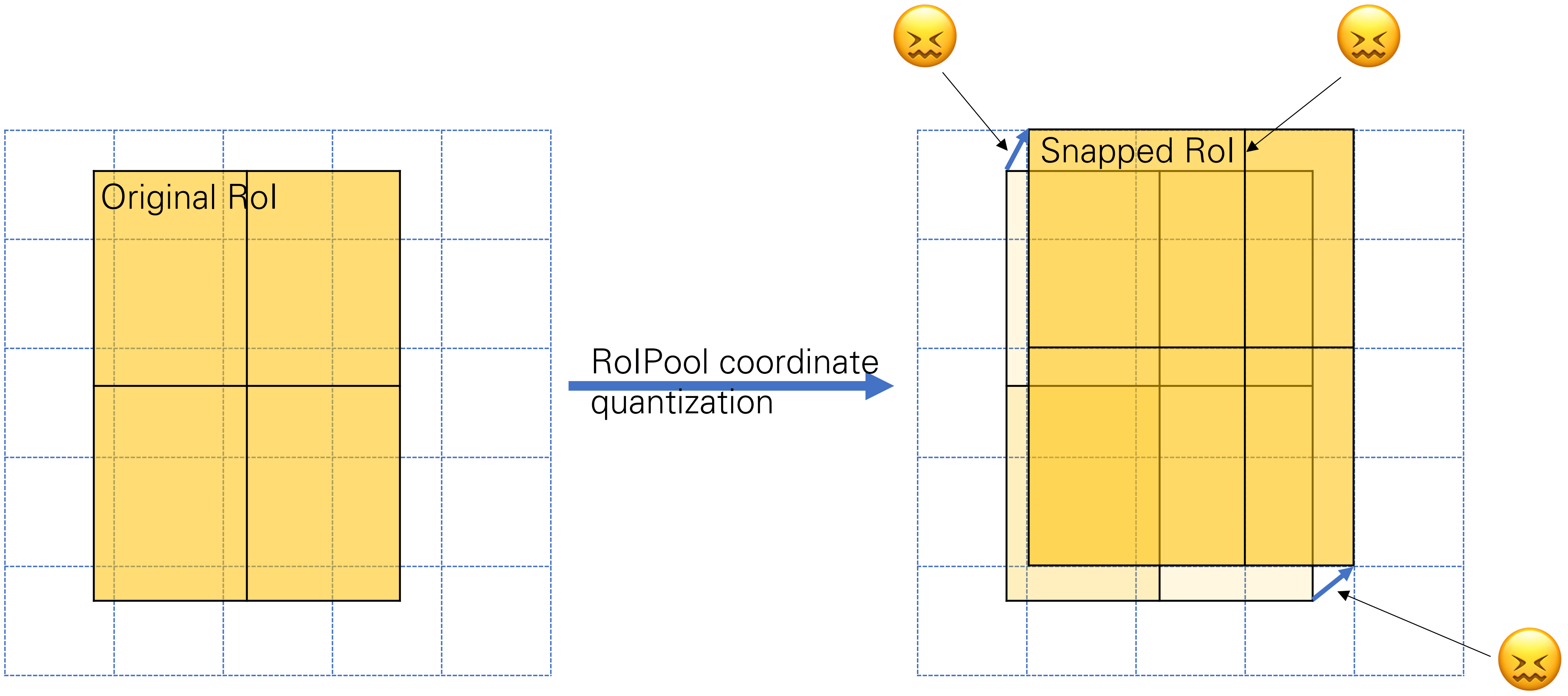
	align?	bilinear?	agg.	AP	AP_{50}	AP_{75}
RoIPool [12]			max	26.9	48.8	26.4
RoIWarp [10]		✓	max	27.2	49.2	27.1
		✓	ave	27.1	48.9	27.1
RoIAlign	✓	✓	max	**30.2**	**51.0**	**31.8**
	✓	✓	ave	**30.3**	**51.2**	**31.5**

+20% relative at high IoU

(c) **RoIAlign** (ResNet-50-C4): Mask results with various RoI layers. Our RoIAlign layer improves AP by ∼3 points and AP_{75} by ∼5 points. Using proper alignment is the only factor that contributes to the large gap between RoI layers.

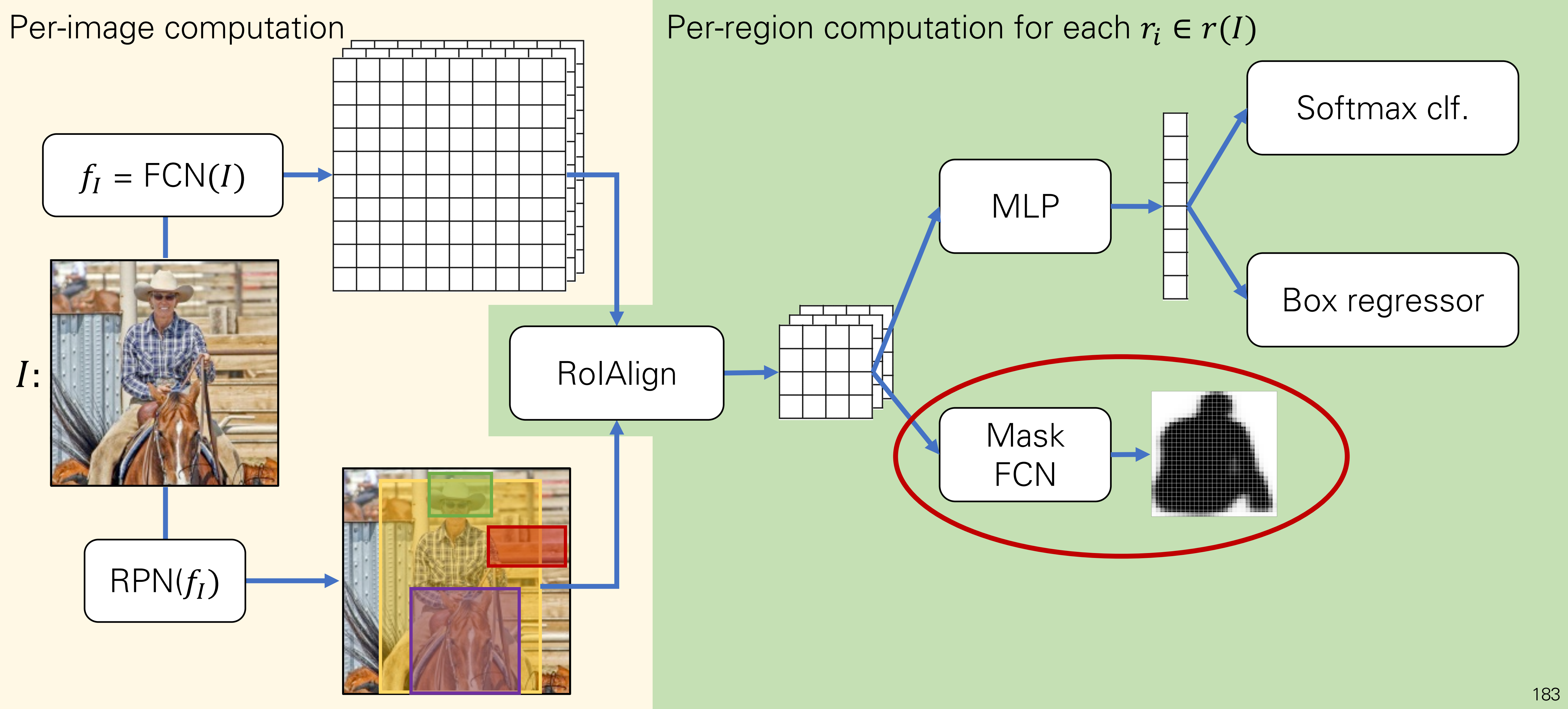
Compare to RoIPool

Quantization breaks pixel-to-pixel alignment



Instance Segmentation

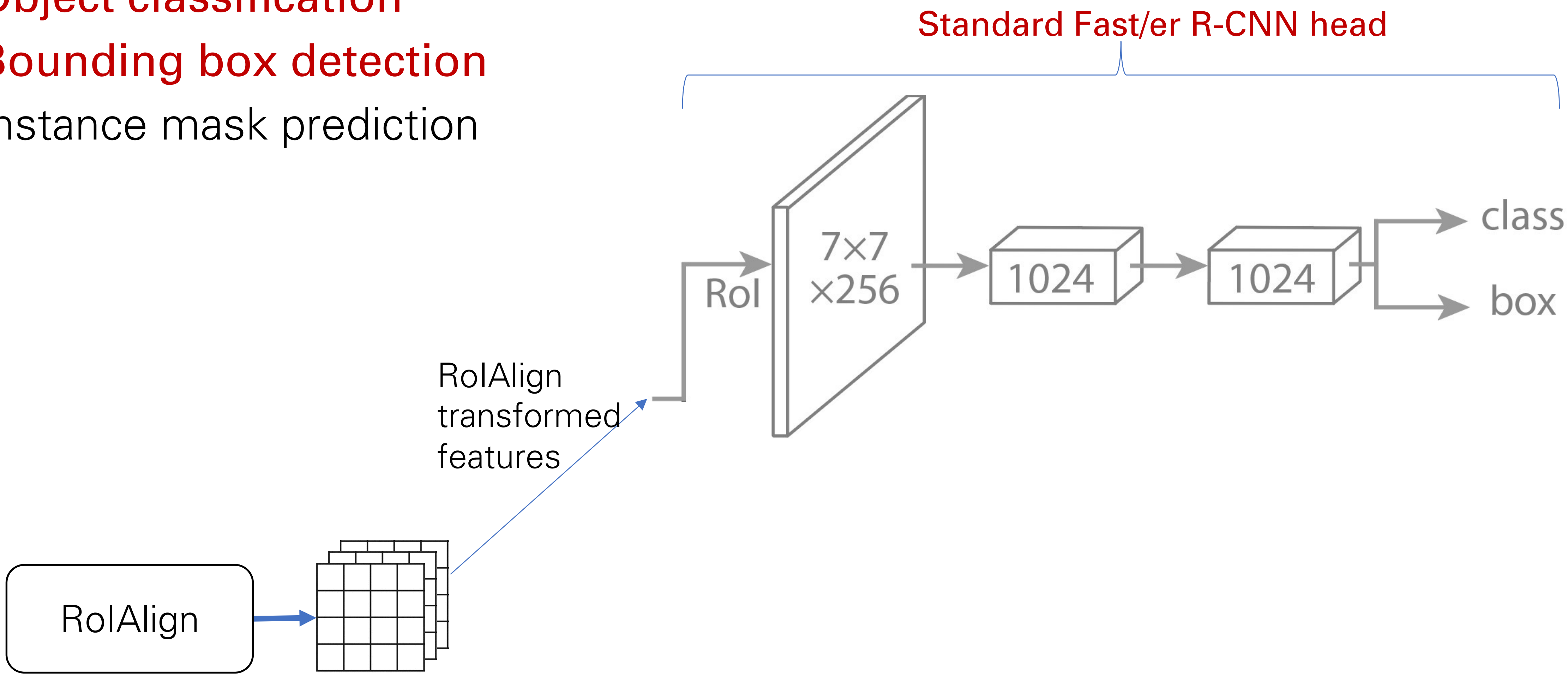
Mask R-CNN



Per-image computation
Per-region computation for each $r_i \in r(I)$
f_I = FCN(I)
I:
RPN(f_I)
RoIAlign
MLP
Softmax clf.
Box regressor
Mask FCN

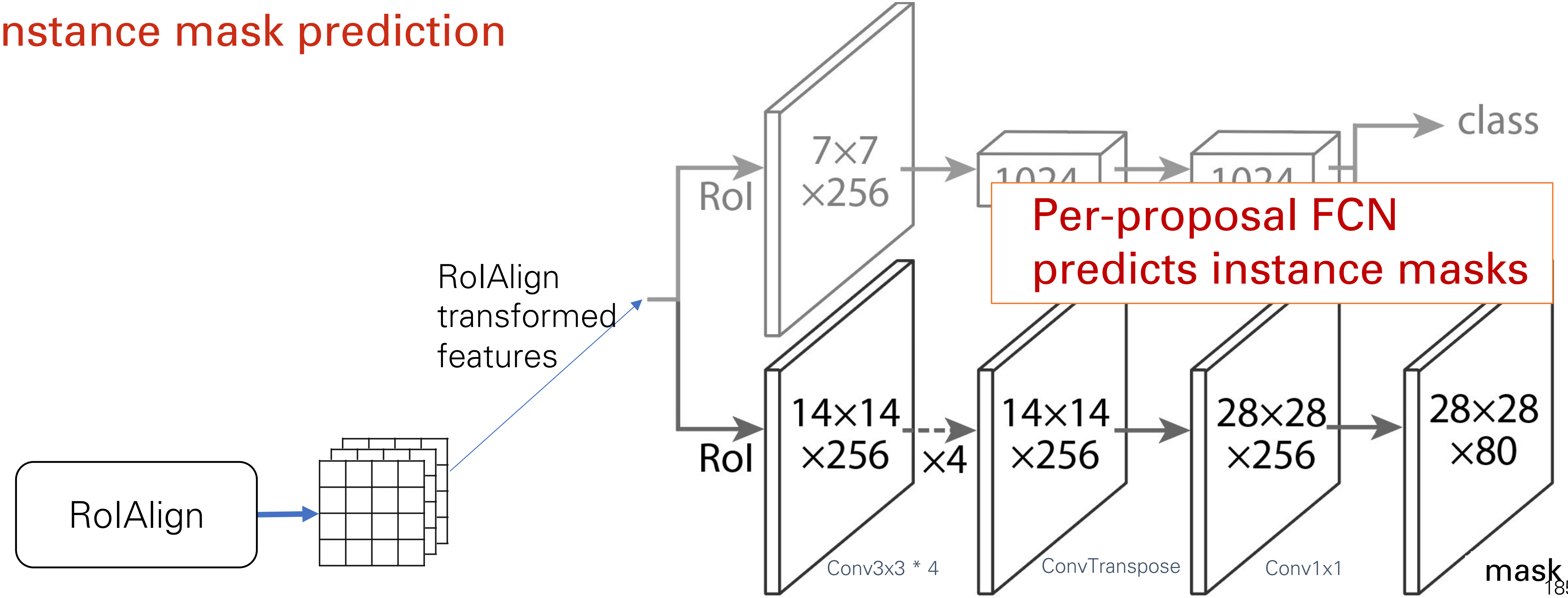
Mask Head (on each Proposal)

- Task specific heads for …
  - Object classification
  - Bounding box detection
  - Instance mask prediction

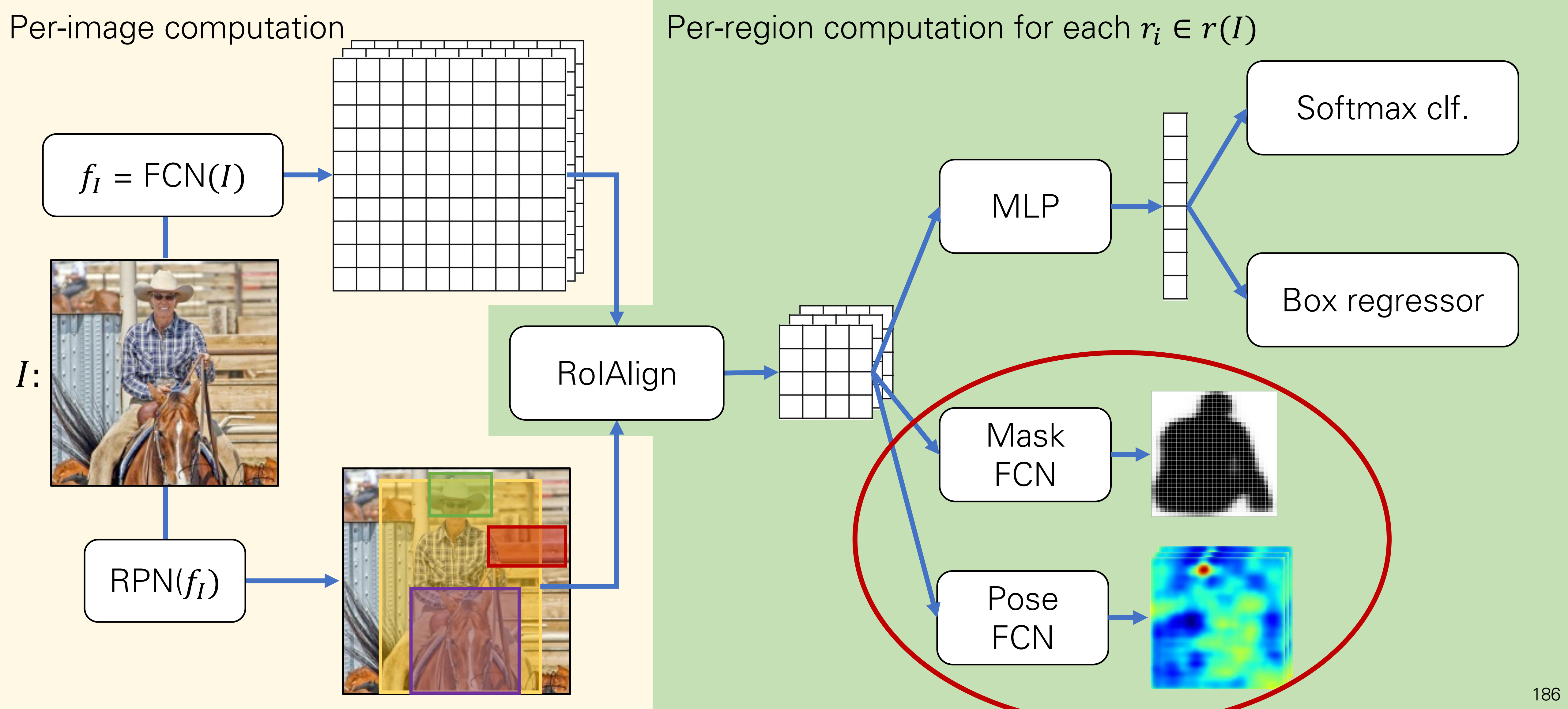


Mask Head (on each Proposal)

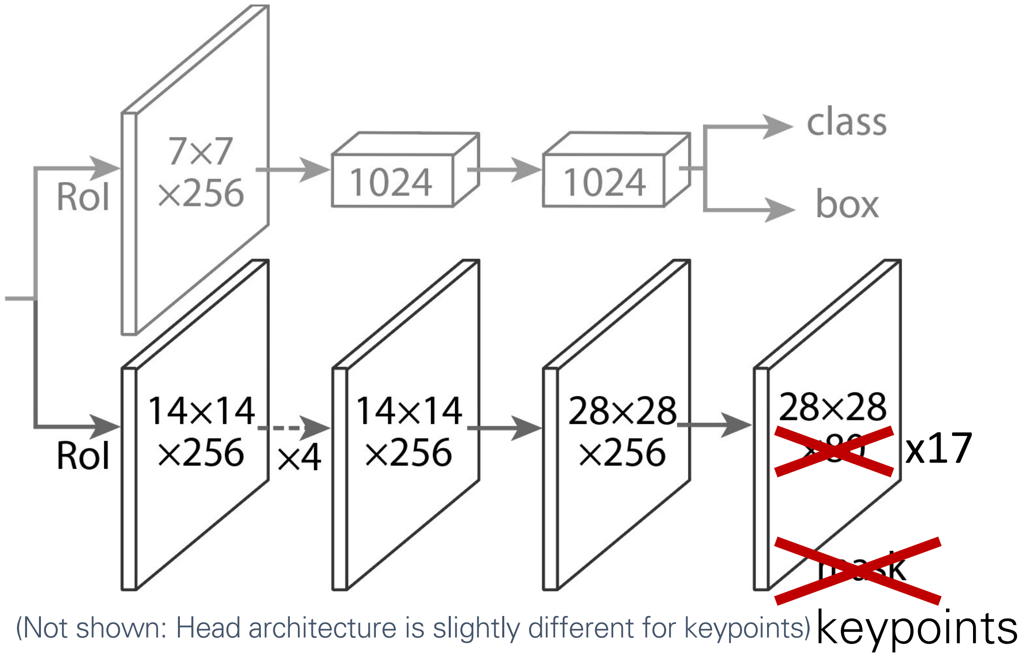
- Task specific heads for …
  - Object classification
  - Bounding box detection
  - Instance mask prediction



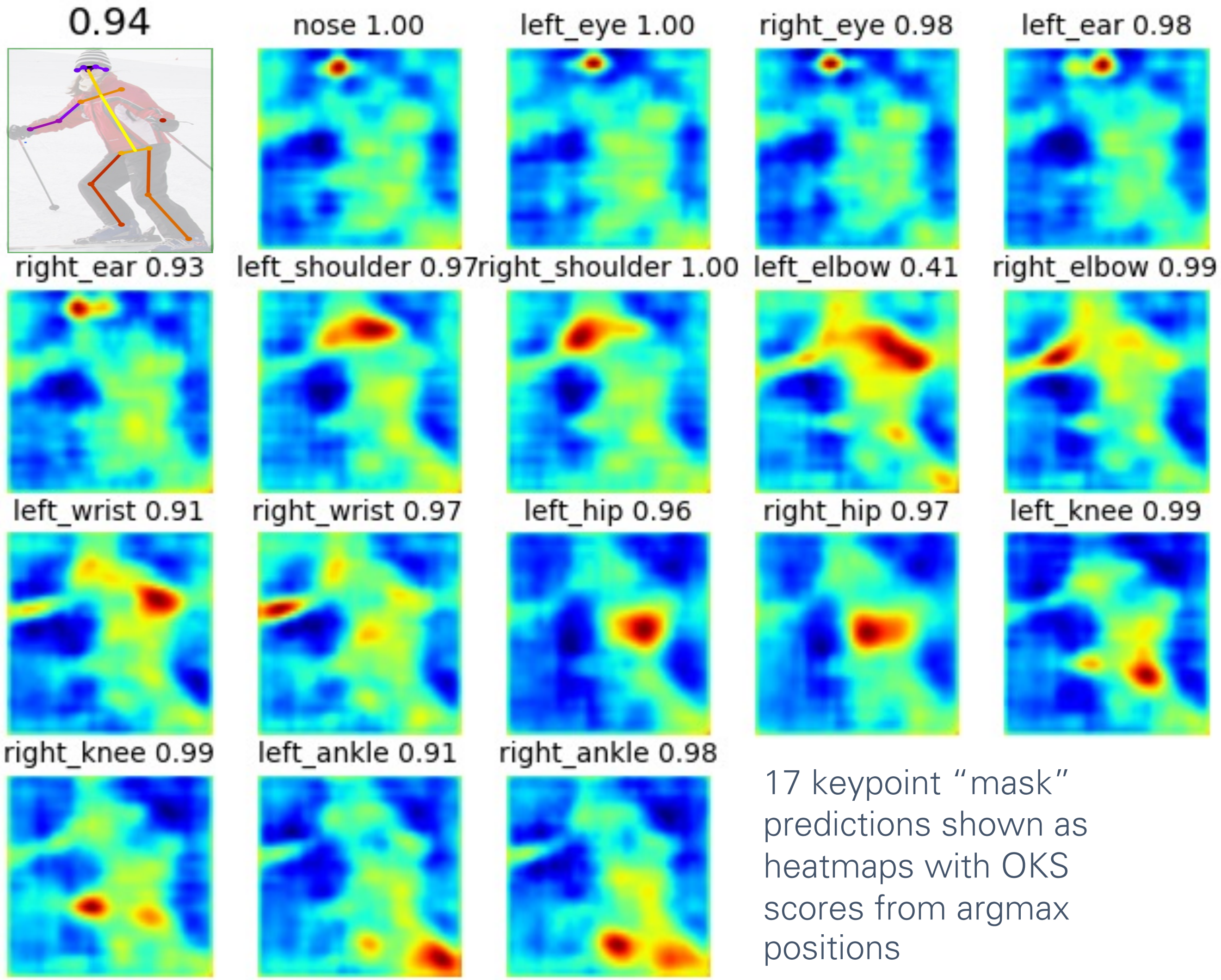
Mask R-CNN: Extension to 2D Human Pose



Pose Head



(Not shown: Head architecture is slightly different for keypoints)



17 keypoint "mask" predictions shown as heatmaps with OKS scores from argmax positions

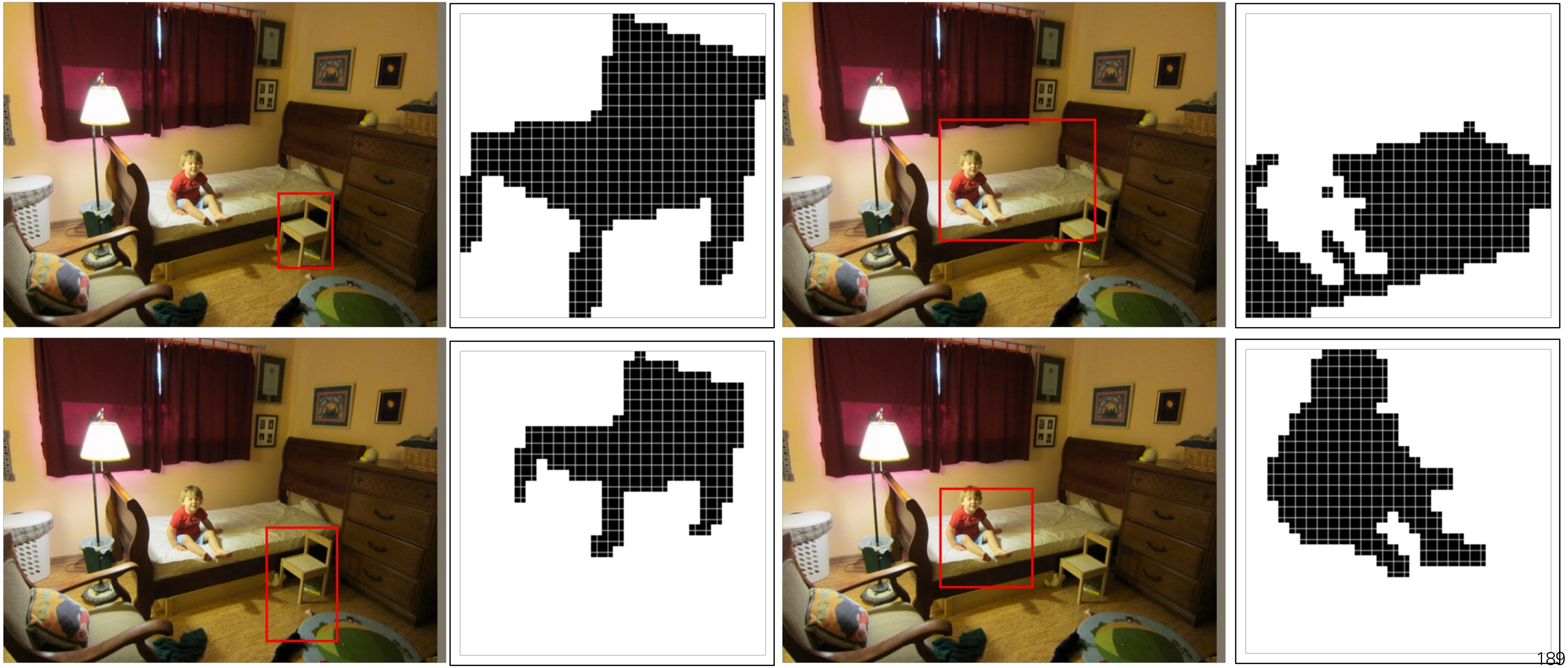
- Add keypoint head (28x28x17)
- Predict one "mask" for each keypoint
- Softmax over spatial locations (encodes one keypoint per mask "prior")

Mask R-CNN: Training

- Same as “image centric” Fast/er R-CNN training
- But with training targets for masks

Example Mask Training Targets

Image with training proposal | 28x28 mask target | Image with training proposal | 28x28 mask target



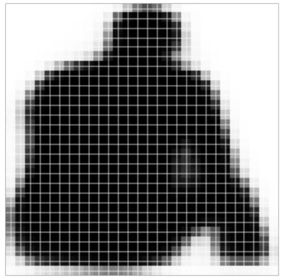
Mask R-CNN: Inference

1. Perform Faster R-CNN inference
   - Run backbone FCN
   - Generate proposals with RPN
   - Score the proposals with clf. head
   - Refine proposals with box regressor
   - Apply NMS and take the top K (= 100, e.g.)

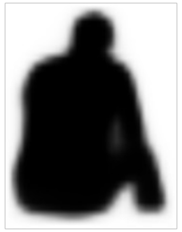
2. Run RoIAlign and mask head on top-*K* refined, post-NMS boxes
   - Fast (only compute masks for top-K detections)
   - Improves accuracy (uses refined detection boxes, not proposals)

Mask Prediction

28x28 soft prediction from Mask R-CNN
(enlarged)



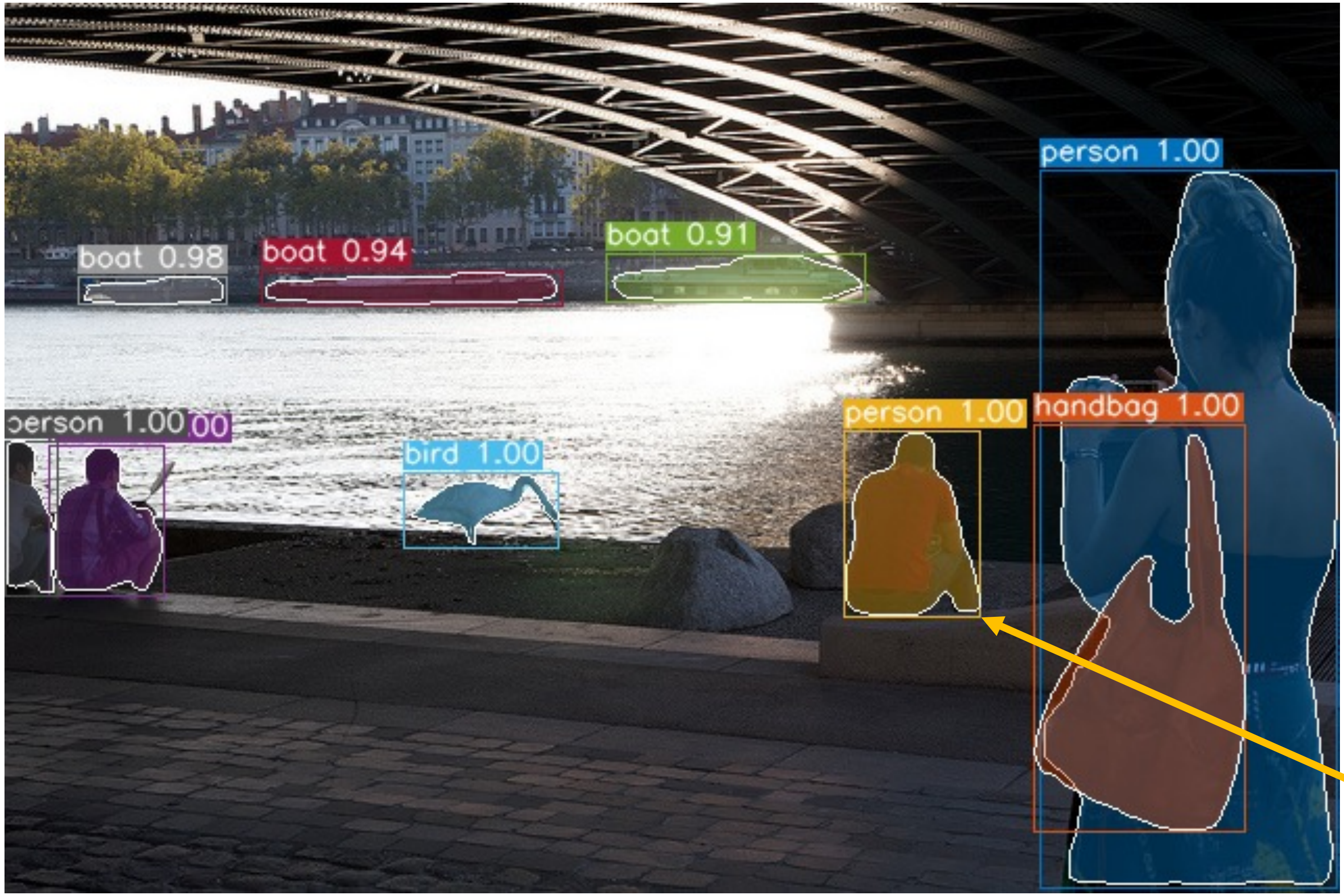
Soft prediction **resampled to image coordinates**
(bilinear and bicubic interpolation work equally well)



Final prediction (threshold at 0.5)

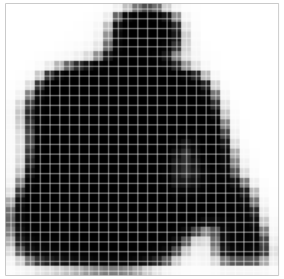
Validation image with box detection shown in red

Mask Prediction

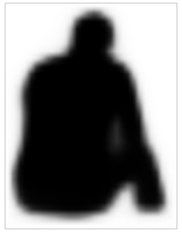


Validation image with box detection shown in red

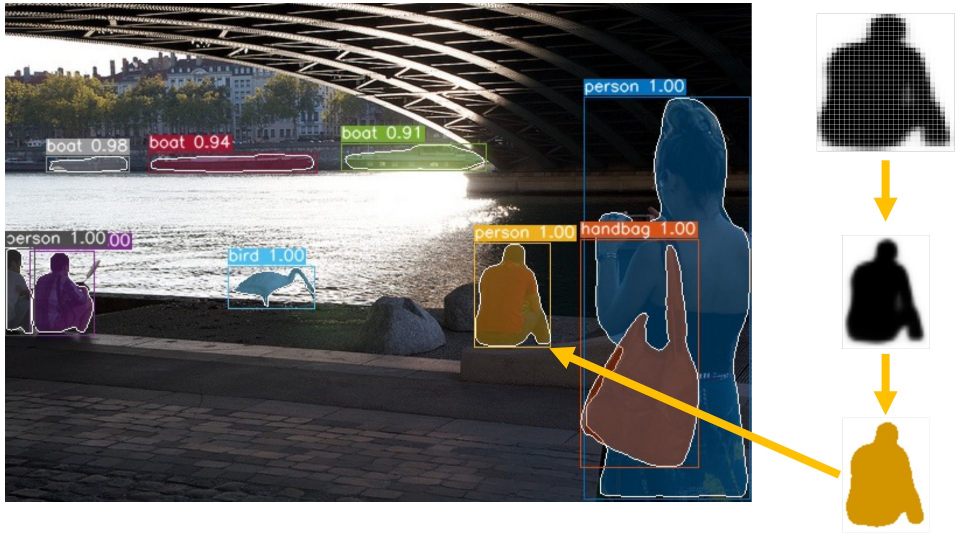
28x28 soft prediction from Mask R-CNN
(enlarged)



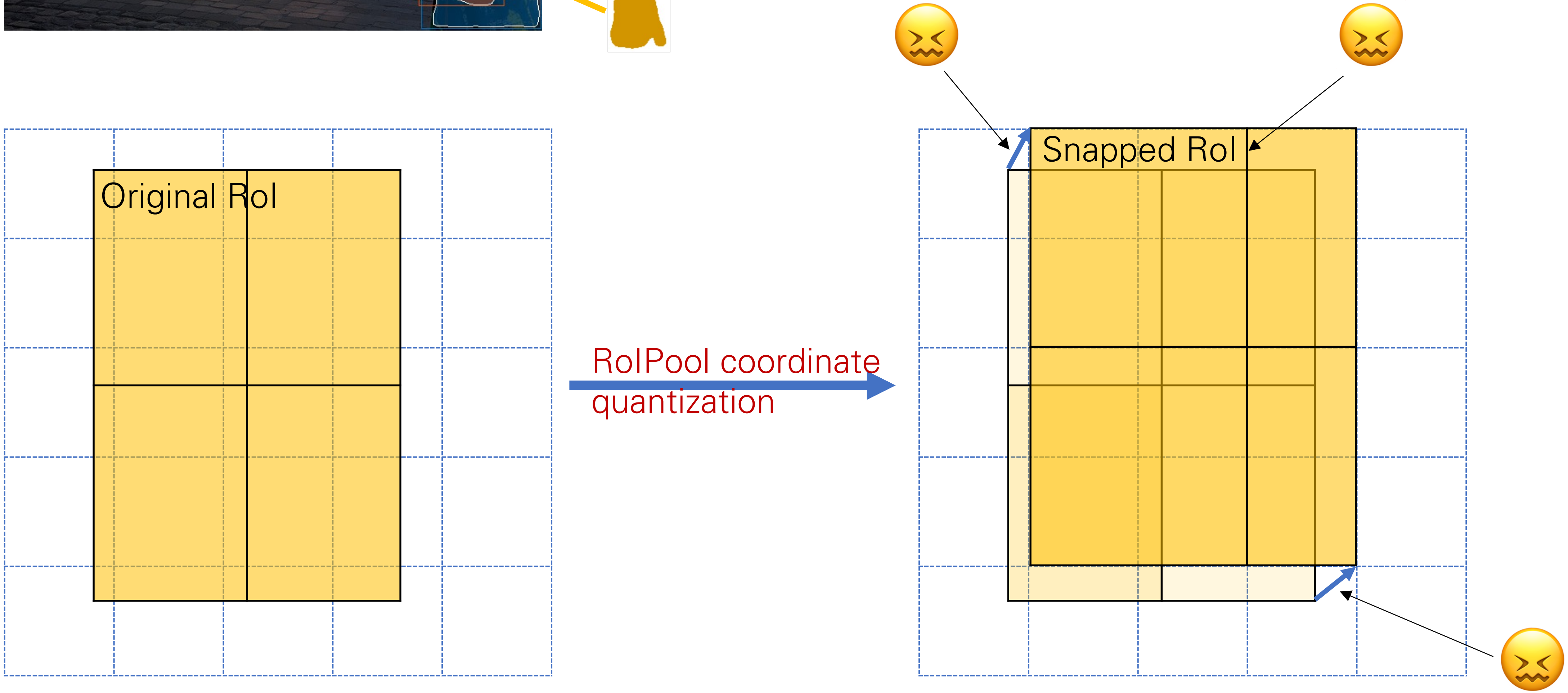
Soft prediction **resampled to image coordinates**
(bilinear and bicubic interpolation work equally well)



Final prediction (threshold at 0.5)

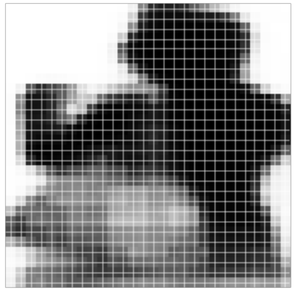


Quantization breaks pixel-to-pixel alignment



Mask Prediction

28x28 soft prediction



Validation image with box detection shown in red

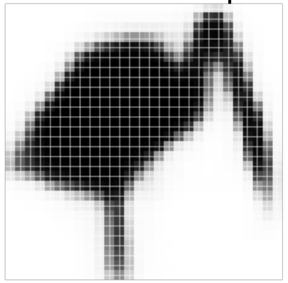
Resized soft prediction

Final mask

Mask Prediction

Validation image with box detection shown in red

28x28 soft prediction

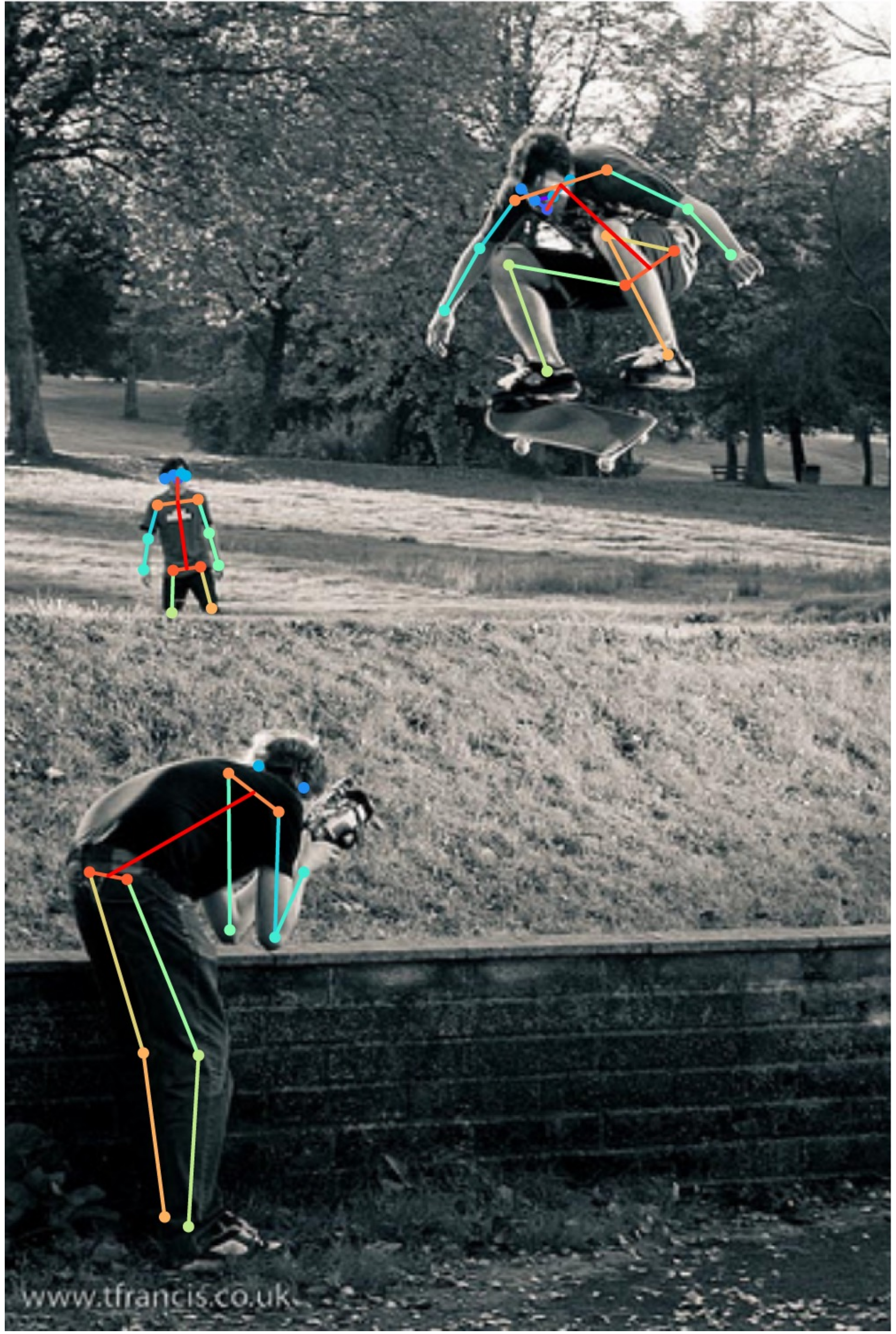


Resized Soft prediction

Final mask

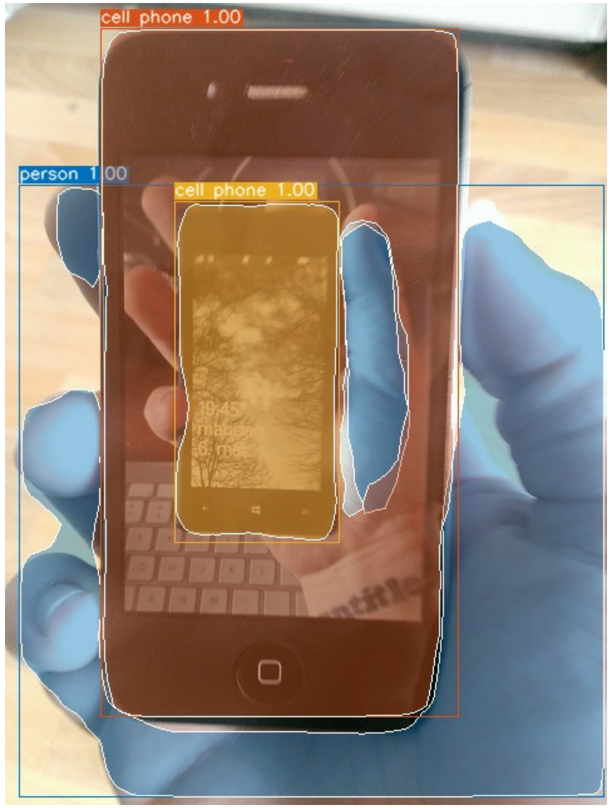

person1.00
person1.00
person.91
surfboard1.00
surfboard1.00
person1.00
surfboard.98
person1.00
surfboard1.00
person.98
surfboard1.00
person.74


person.88
person1.00
tv.98
tv.84
person1.00
person1.00
bottle.97
wine glass.99
dining table.95
wine glass1.00
wine glass1.00


www.tfrancis.co.uk

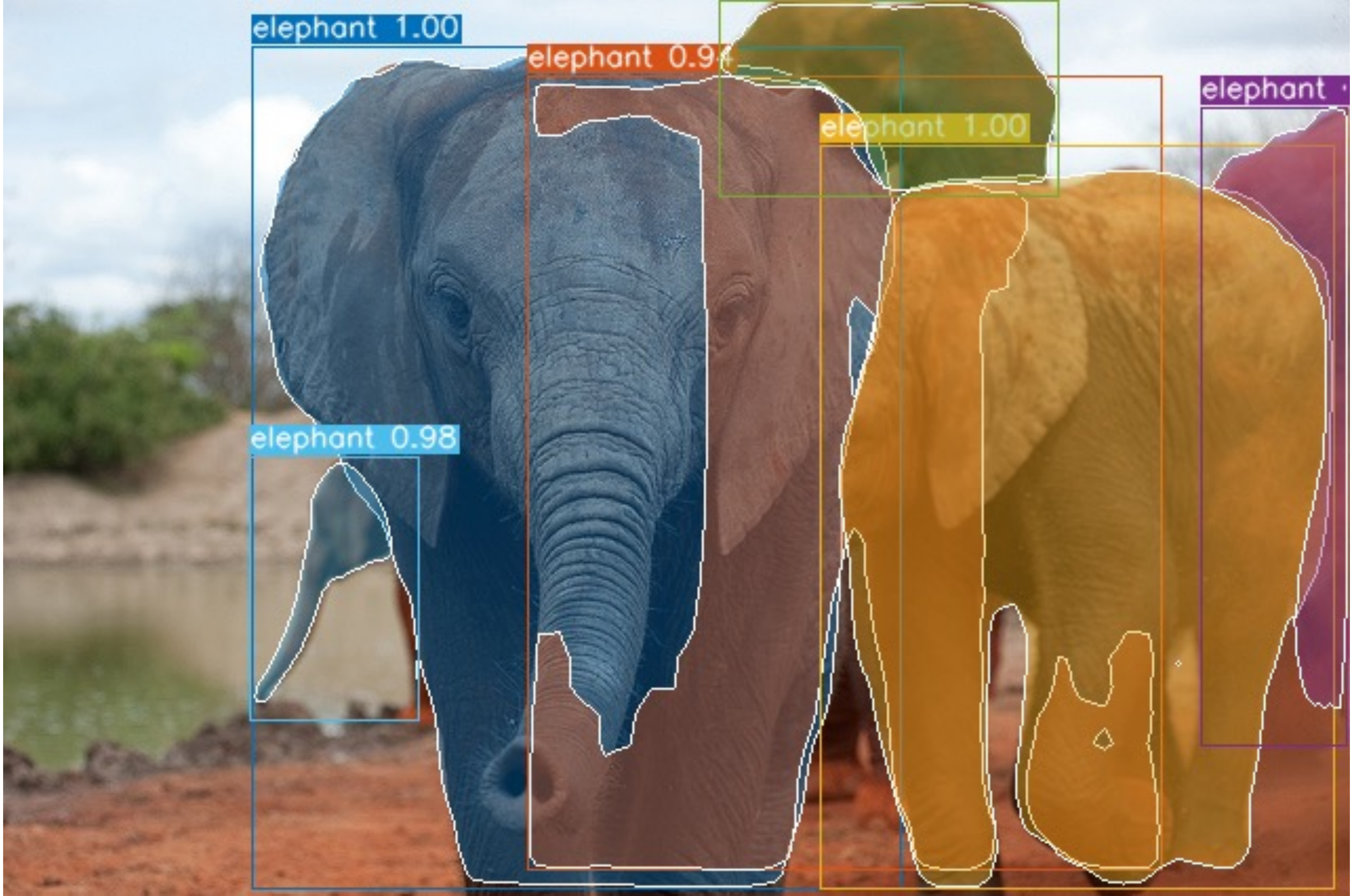
Is Object Detection Solved?

- Obviously no; there are **frequently silly errors**
- But it is getting frustratingly good
- The errors are often reasonable
- The bottlenecks are raw recognition and "reasoning"



tv 1.00
person 1.00
bottle 0.92
bottle 0.93
sink 0.99
sink 0.99
sink 0.96
bed 0.98

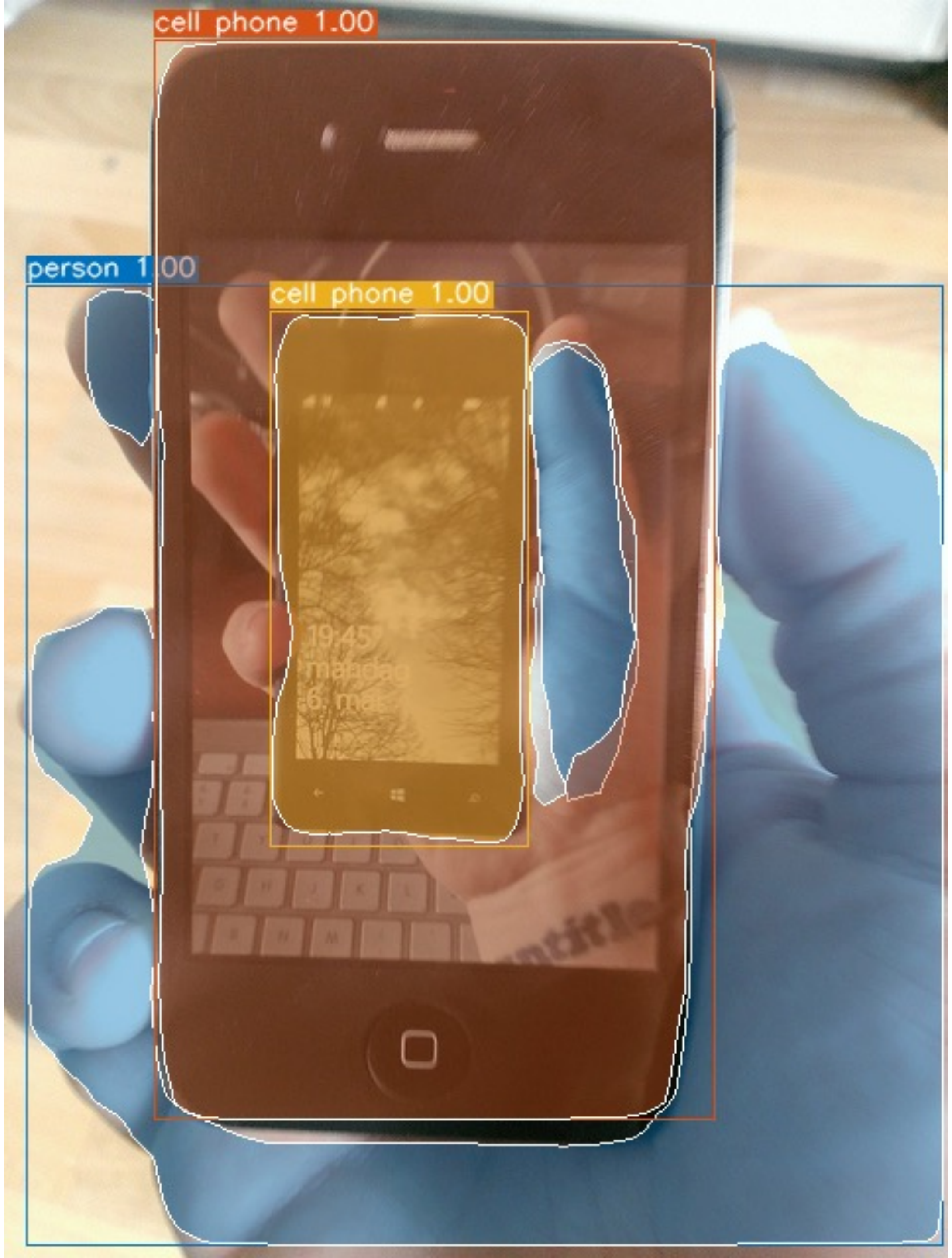

bed 1.00
bed 0.98
bed 0.90
bed 0.94

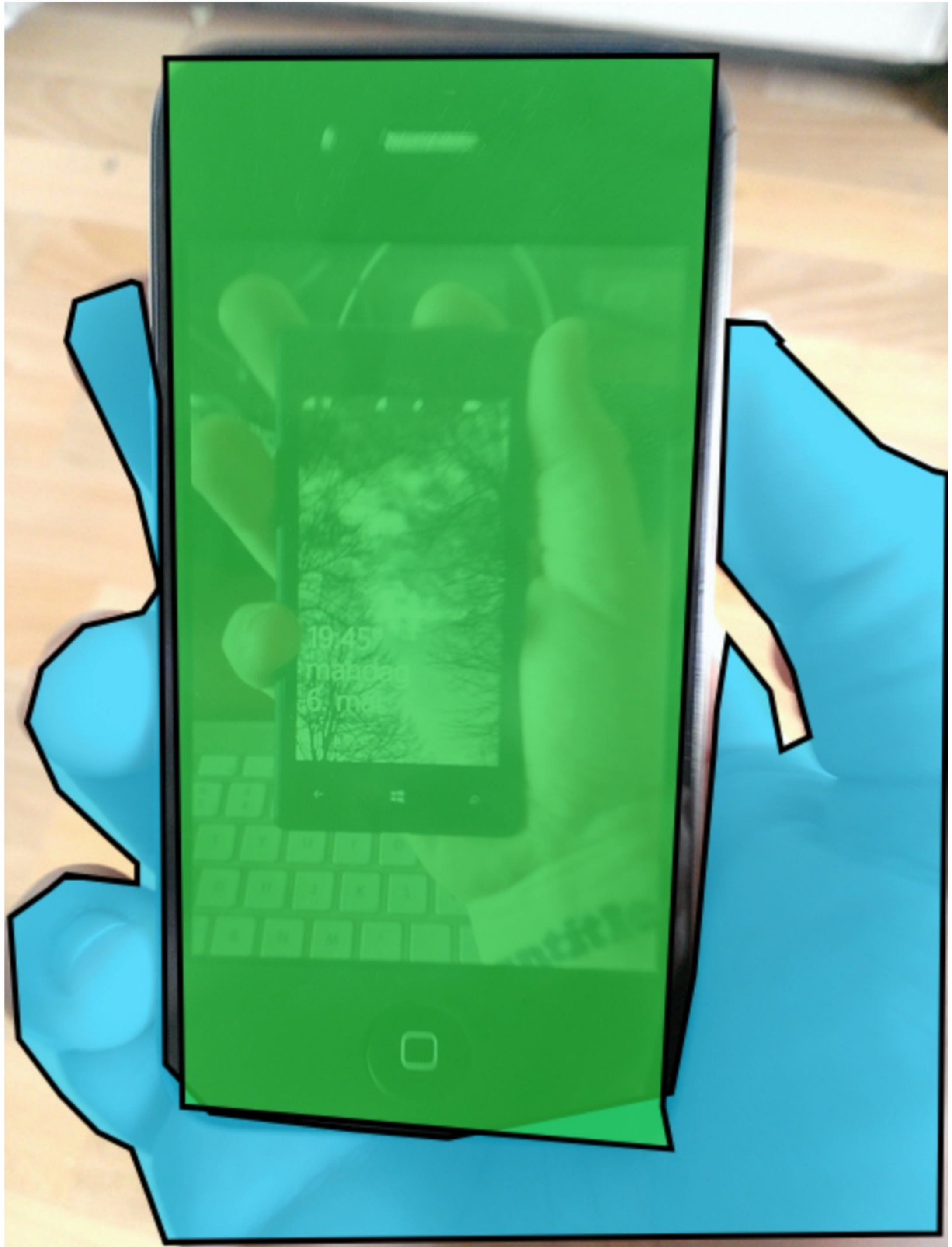

elephant 1.00
elephant 0.94
elephant
elephant 1.00
elephant 0.98


person 0.99
person 0.95
person 0.94
person 0.93
person 1.00
person 0.97
bottle 0.94
bottle 0.99
bottle 0.95


person 1.00
tie 0.96

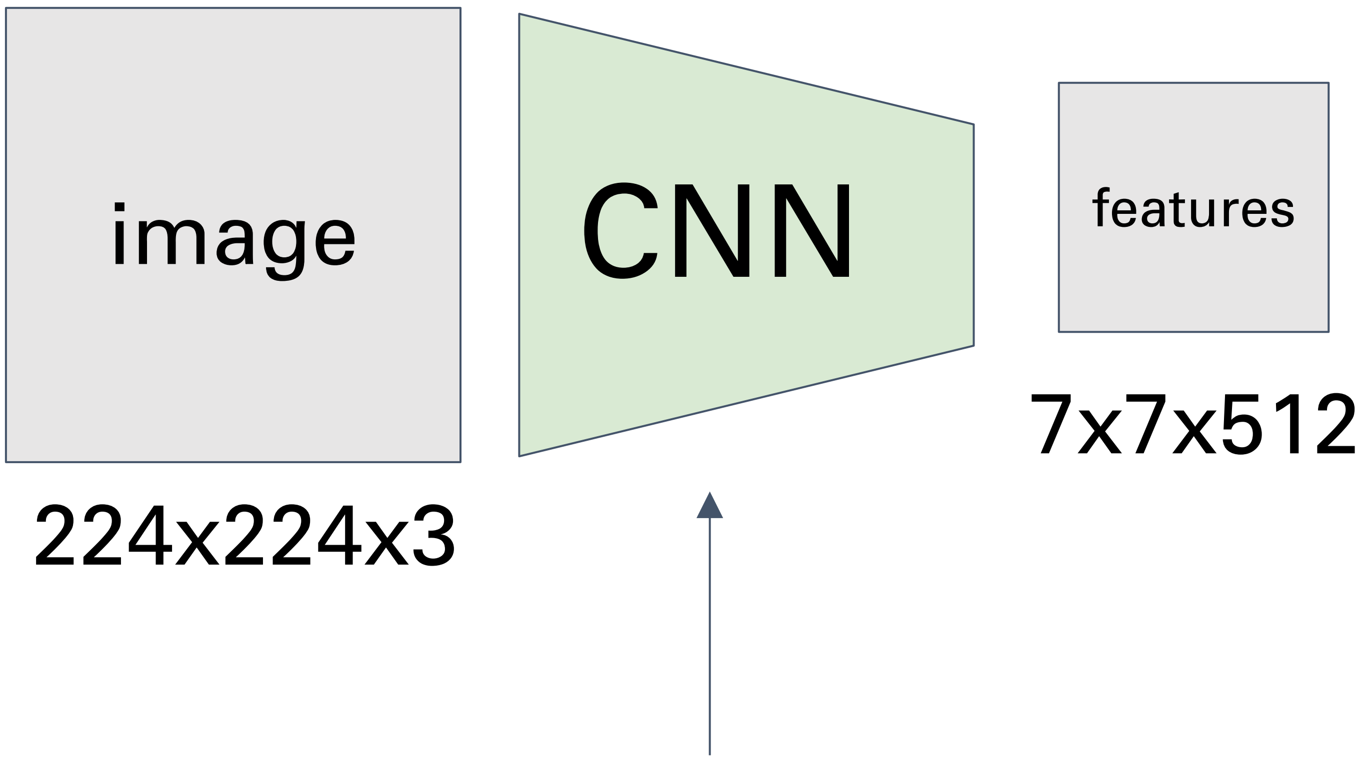
person 0.95
vase 0.97
orange 0.96
orange 0.97
book 0.


cell phone 1.00
person 1.00
cell phone 1.00



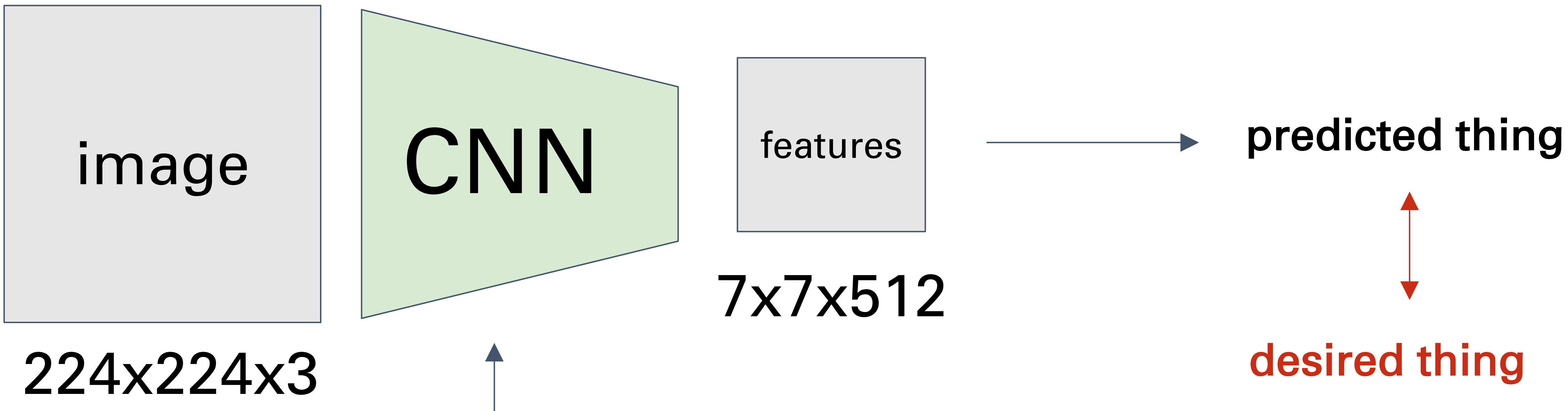
Addressing other tasks...

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...

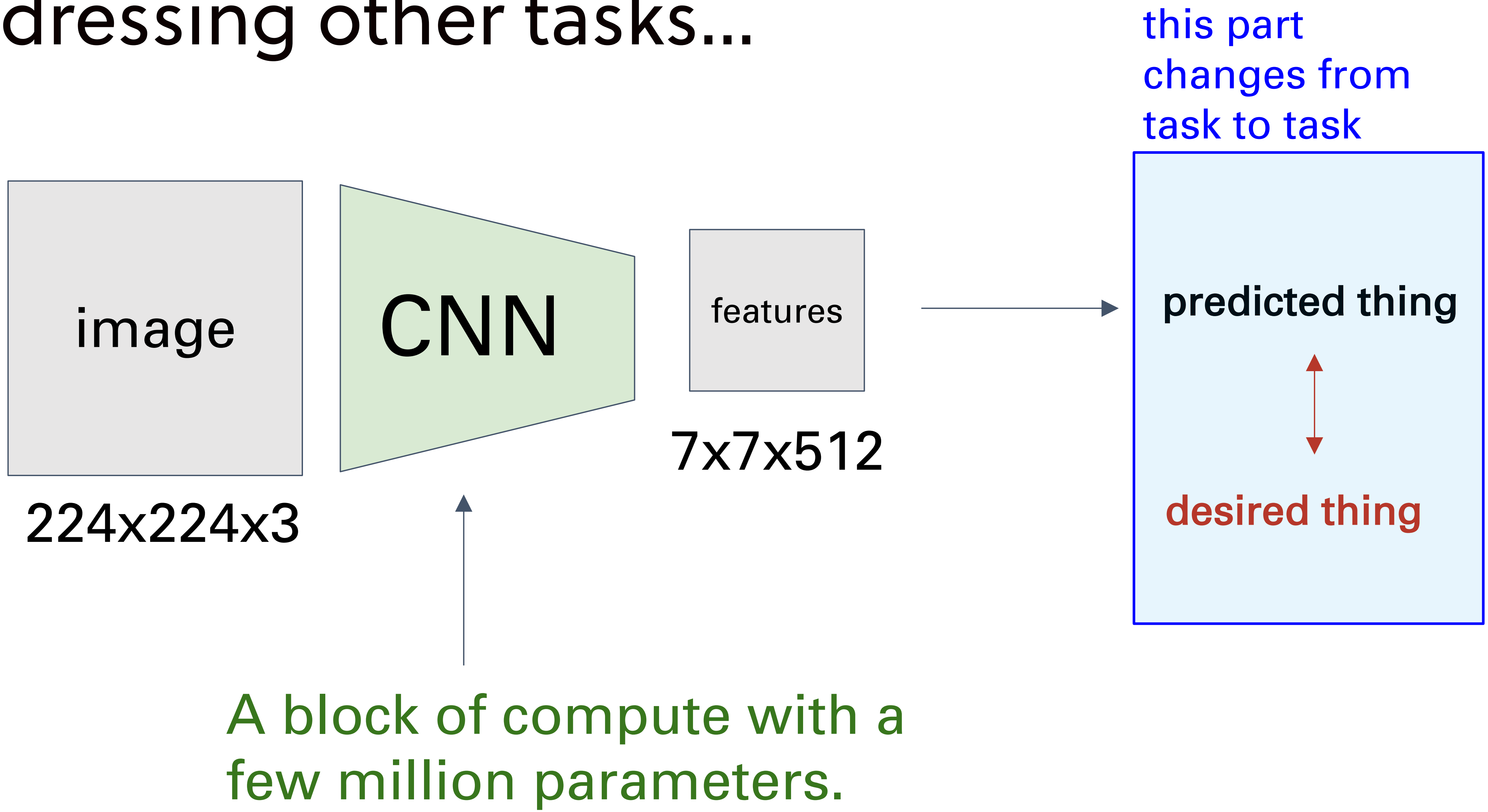
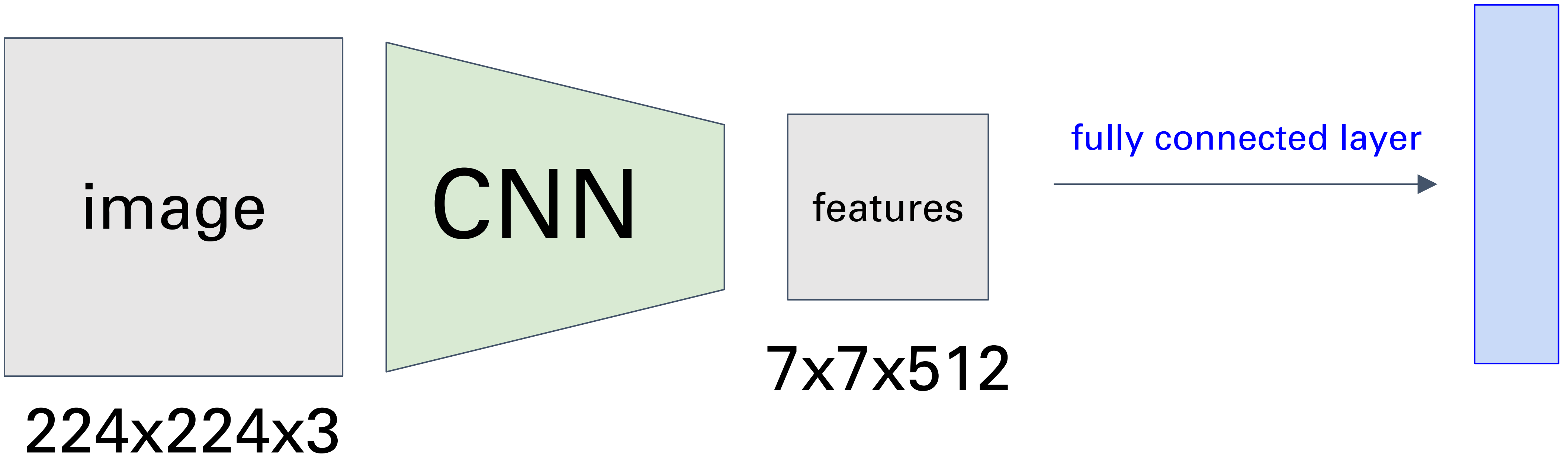


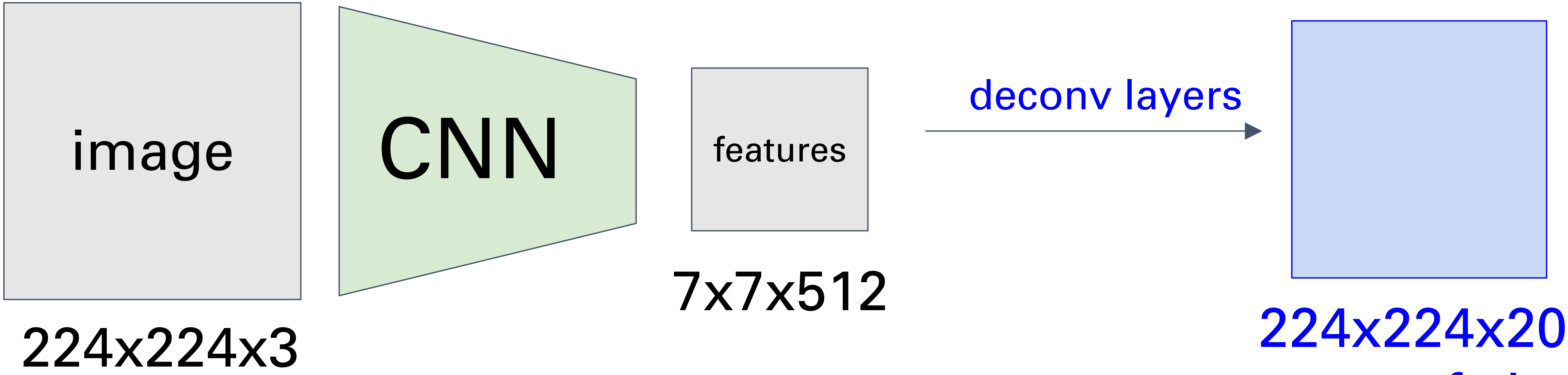
Image Classification

thing = a vector of probabilities for different classes



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

Segmentation



224x224x20
array of class probabilities at each pixel.

Localization

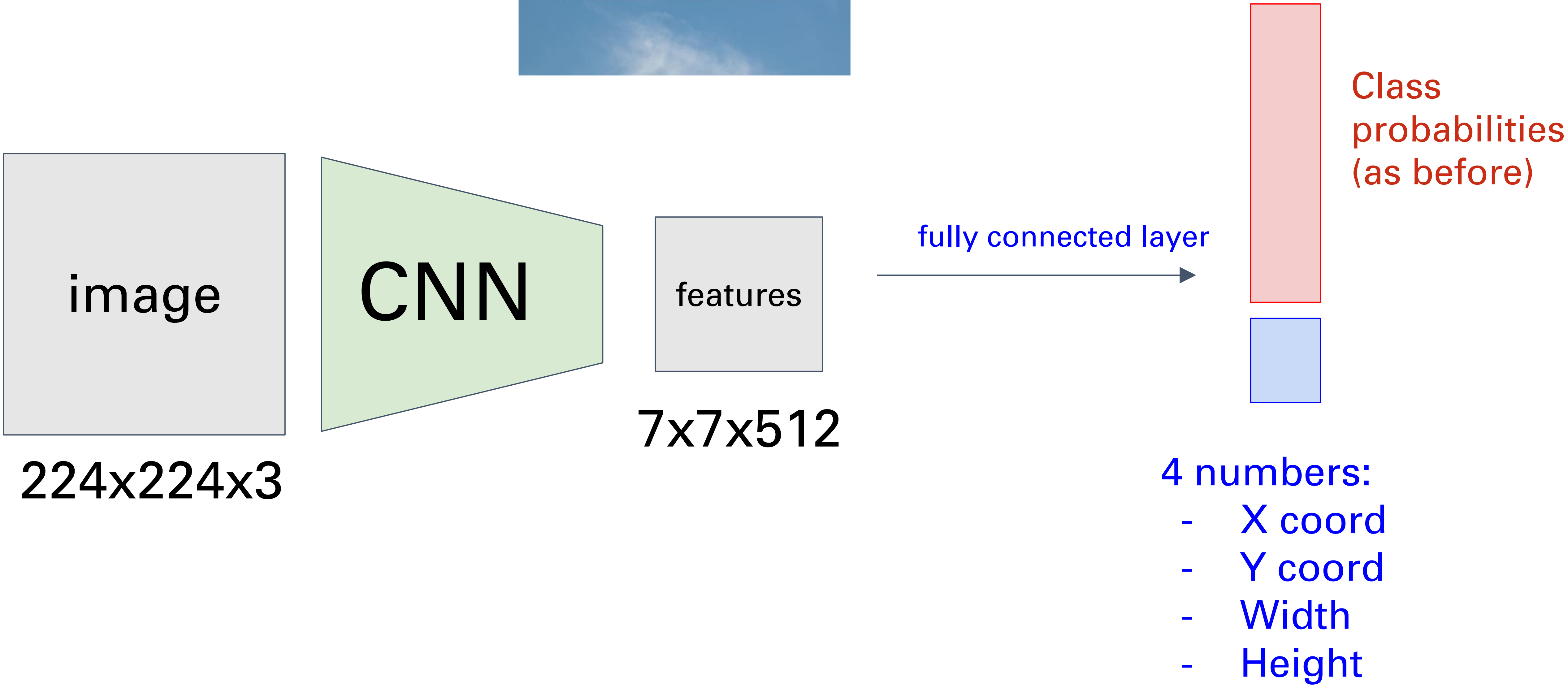
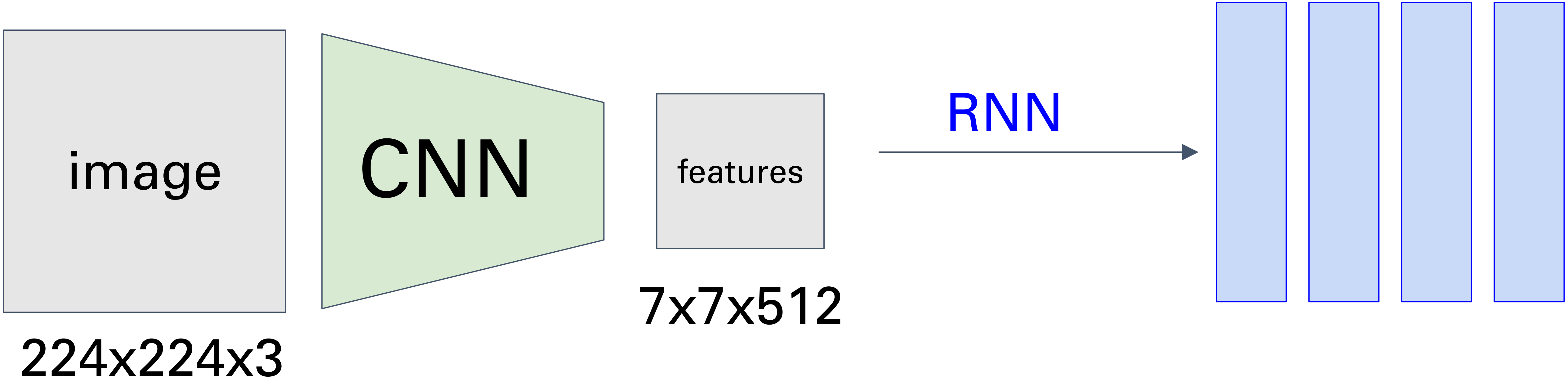
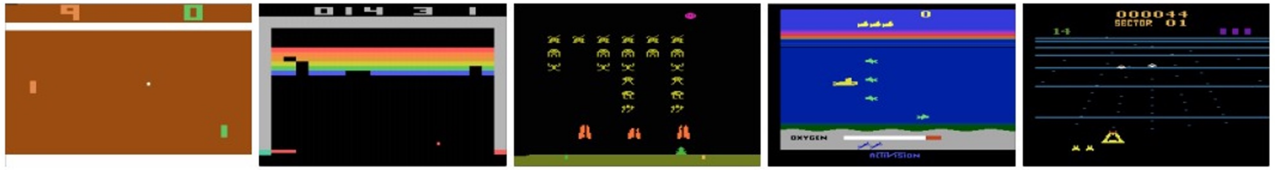


Image Captioning

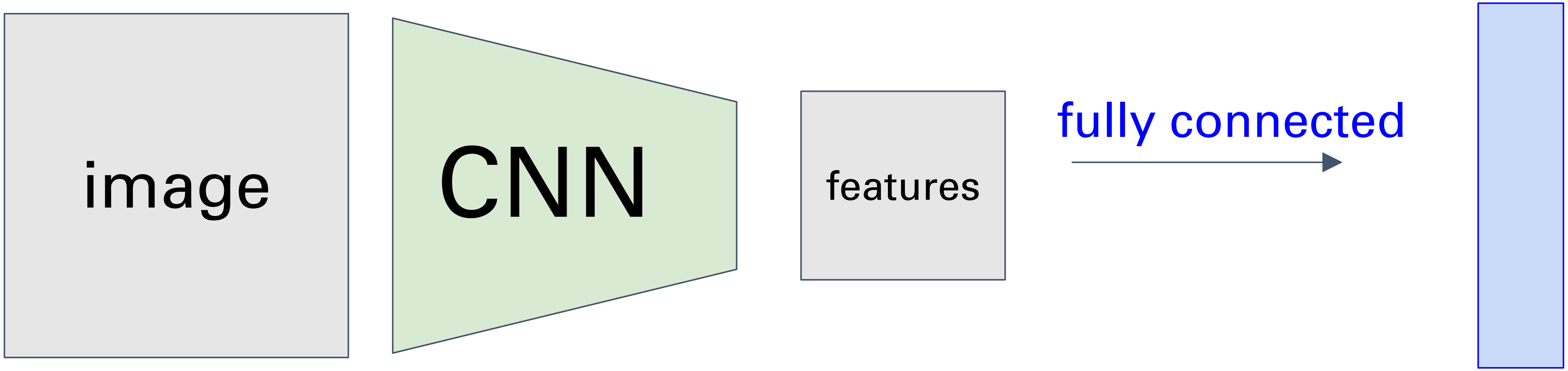


A sequence of 10,000-dimensional vectors giving probabilities of different words in the caption.

Reinforcement Learning



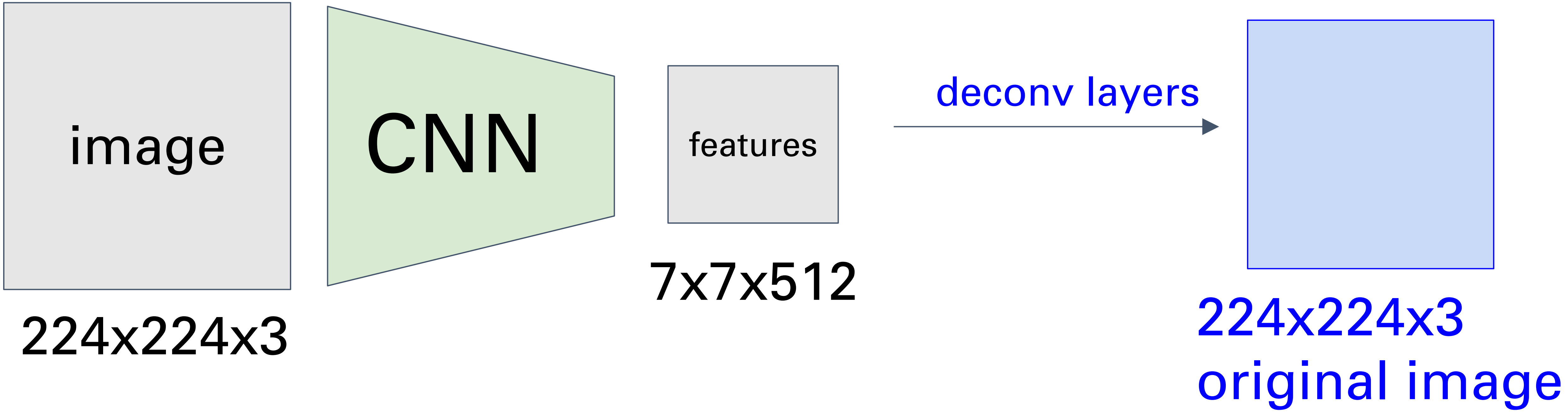
Mnih et al. 2015



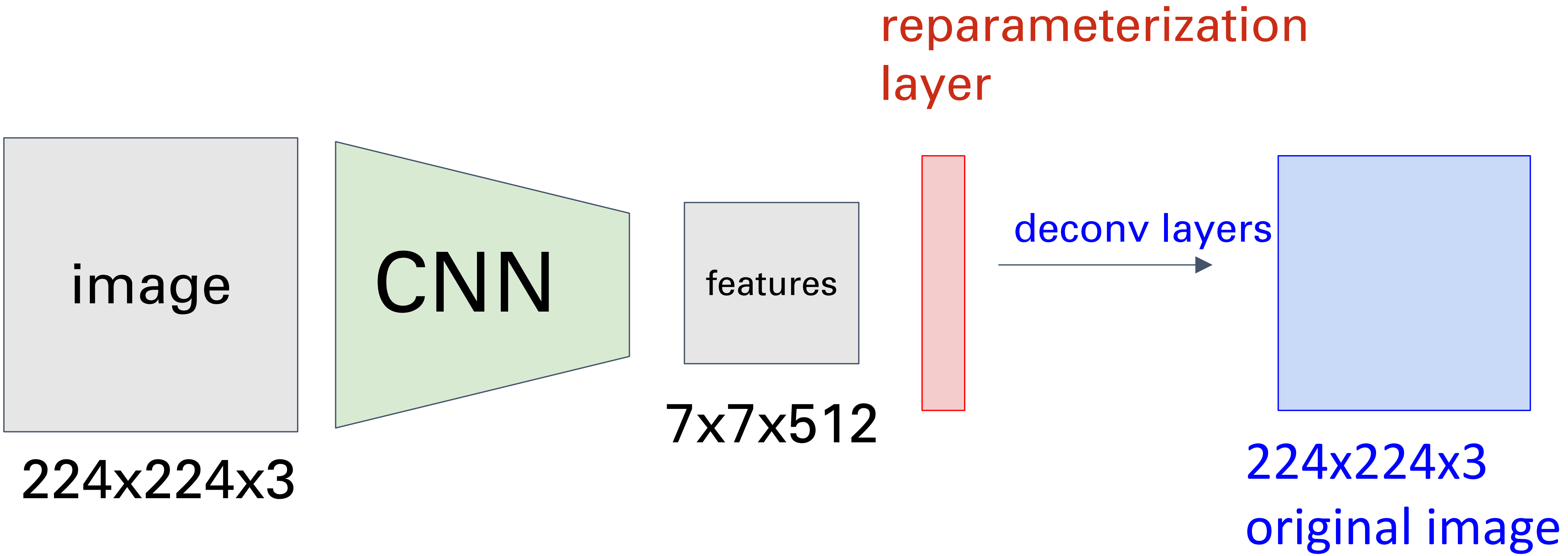
160x210x3

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

Autoencoders

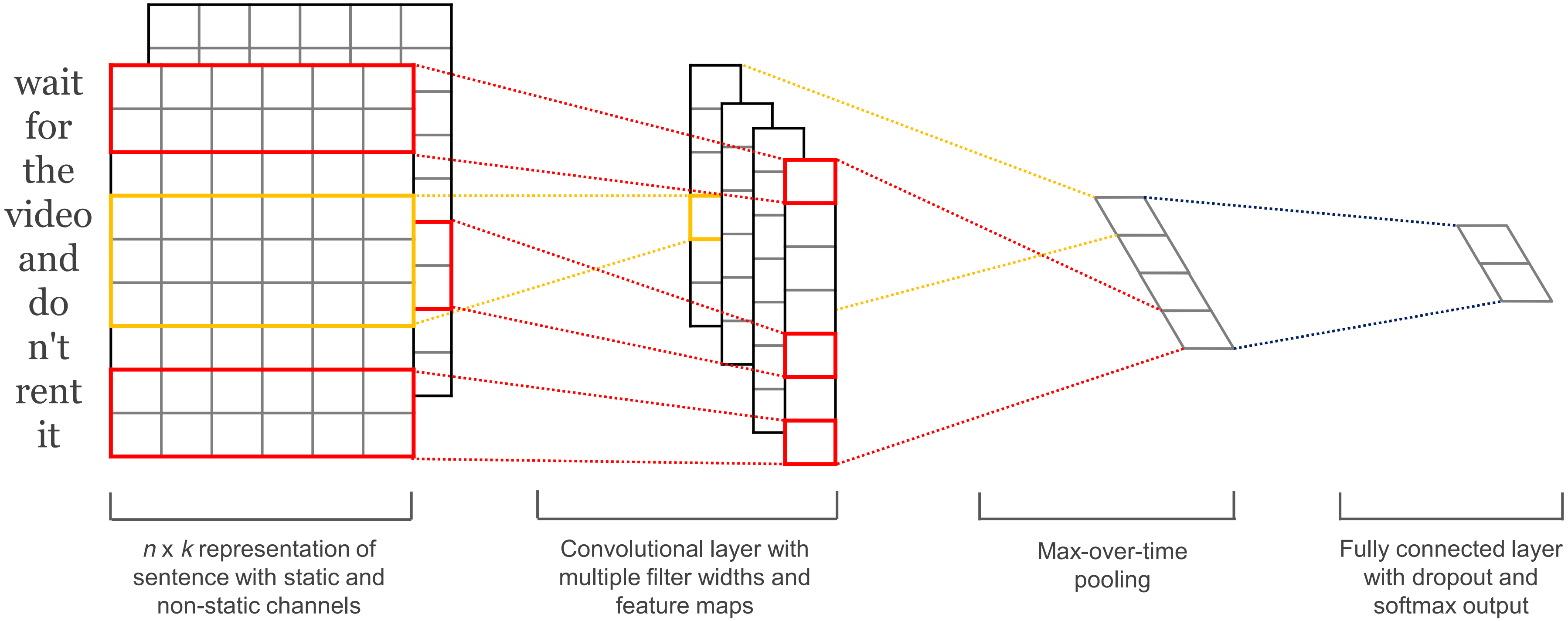


Variational Autoencoders



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

Addressing other tasks...

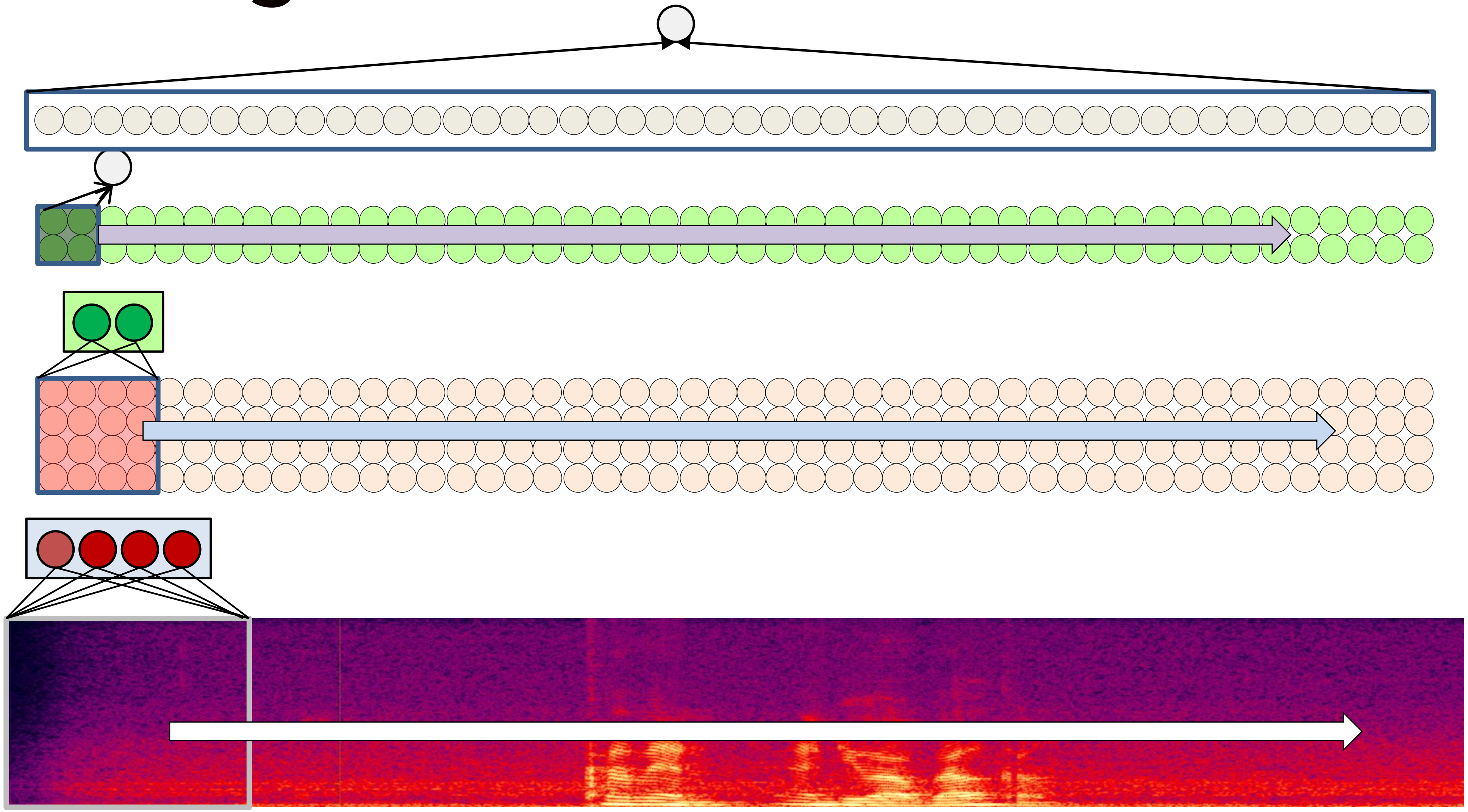


- 1D convolution ≈ Time Delay Neural Networks (Waibel et al. 1989, Collobert and Weston 2011)
- Two main paradigms:
  - **Context window modeling:** For tagging, etc. get the surrounding context before tagging
  - **Sentence modeling:** Do convolution to extract n-grams, pooling to combine over whole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