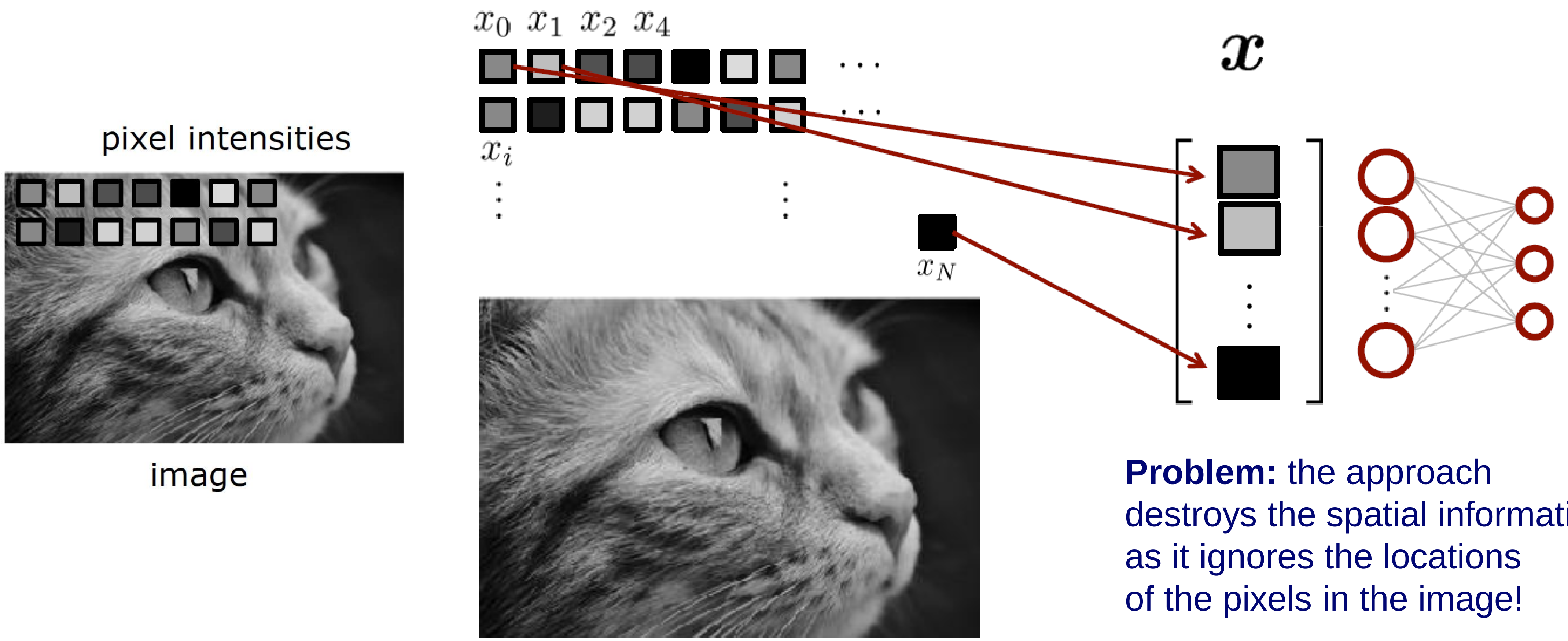
Data-intensive space engineering

Lecture 9

Carlos Sanmiguel Vila

Based on previous work of Cyrill Stachniss from University of Bonn

The Good Old MLP's Input...



Problem: the approach destroys the spatial information as it ignores the locations of the pixels in the image!

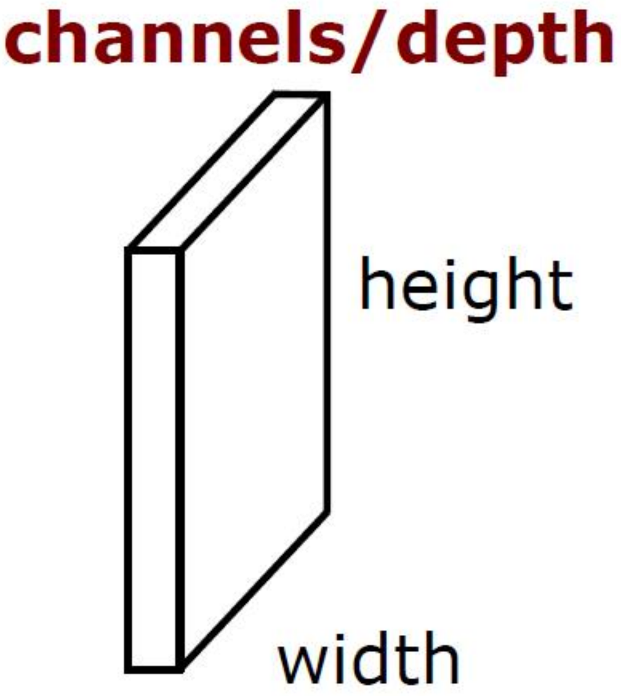
CNNs Overcome this Problem

- CNNs maintain the 2D image structure
- Neighborhoods are maintained
- Network layers can learn features that also encode spatial information
- Convolutions are local operators
- CNNs use convolutions & subsampling (called pooling)
- Thanks to the increase in computational power and the amount of available training data, convolutional neural networks (CNNs) have achieved great performance on complex visual tasks
  - Image search services, self-driving cars, video classification systems, etc.
  - Not restricted to visual applications, e.g., voice recognition

CNNs Overcome this Problem

- CNNs maintain the 2D image structure
- Neighborhoods are maintained
- Network layers can learn features that also encode spatial information
- Convolutions are local operators
- CNNs use convolutions & subsampling (called pooling)
- Thanks to the increase in computational power and the amount of available training data, convolutional neural networks (CNNs) have achieved great performance on complex visual tasks
  - Image search services, self-driving cars, video classification systems, etc.
  - Not restricted to visual applications, e.g., voice recognition

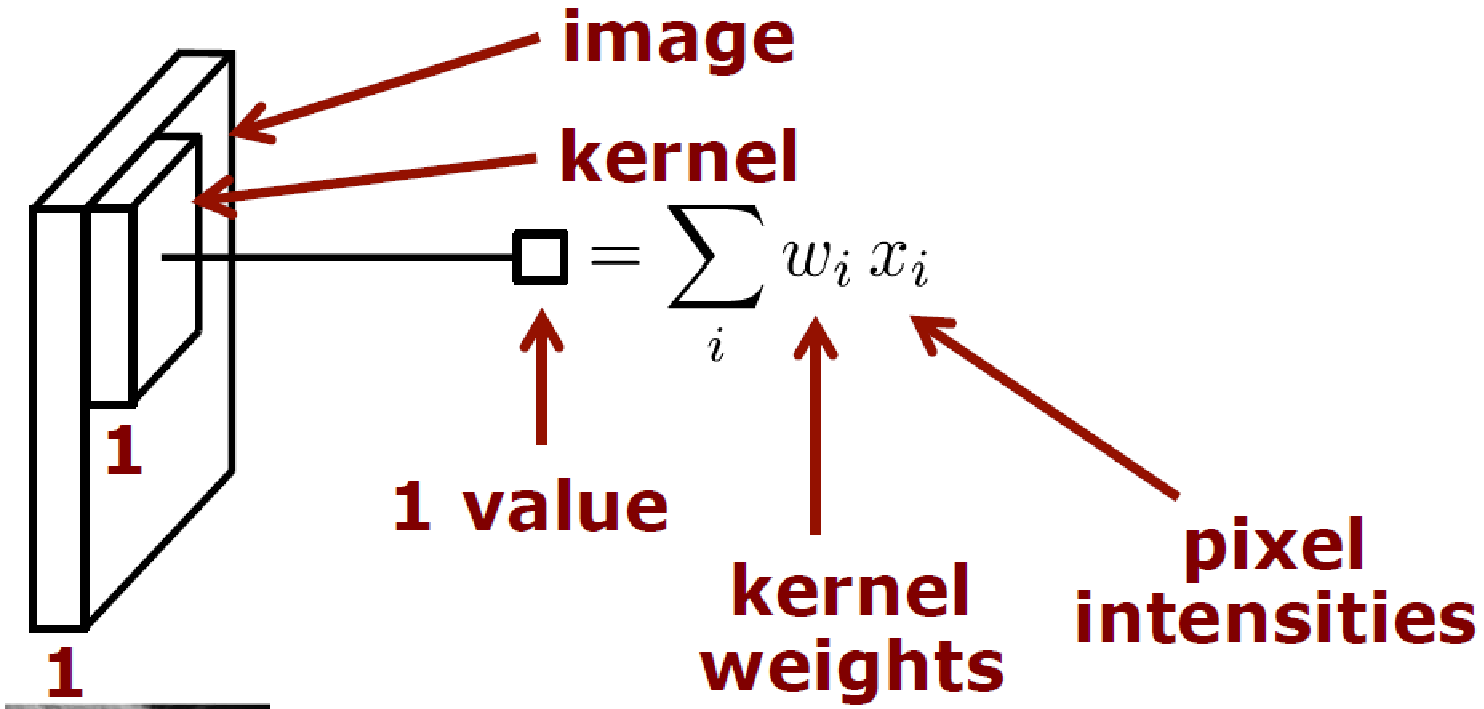
Let’s Start With the Input



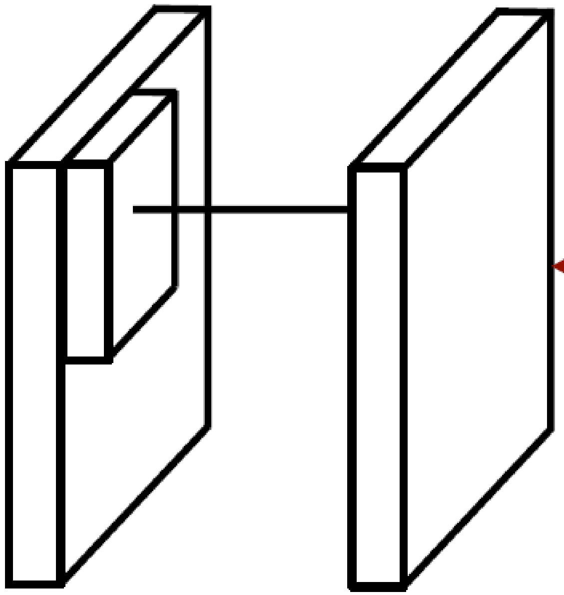
depth=1

depth=3

Convolution Using a Kernel



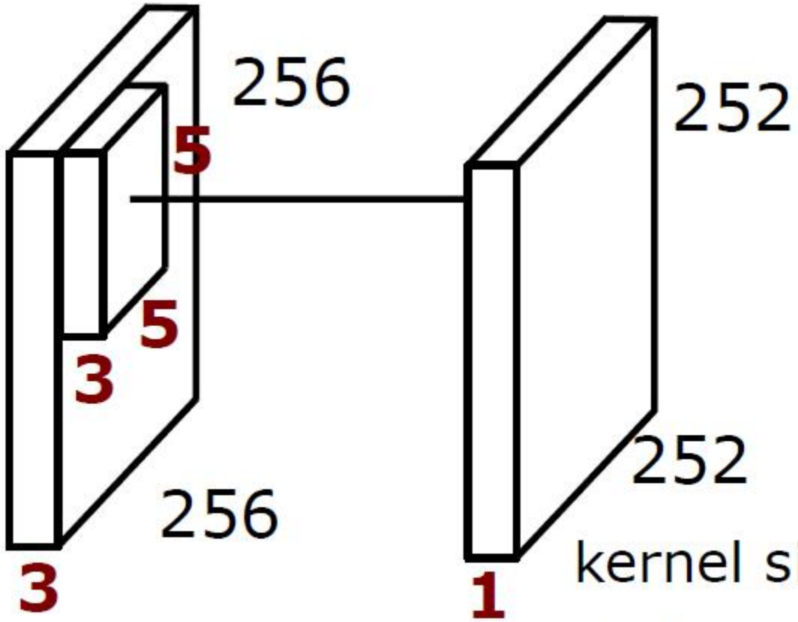
Convolution Using a Kernel



This is the output (image) of a convolution!

example for blurring through a convolution

Convolution Using a Kernel



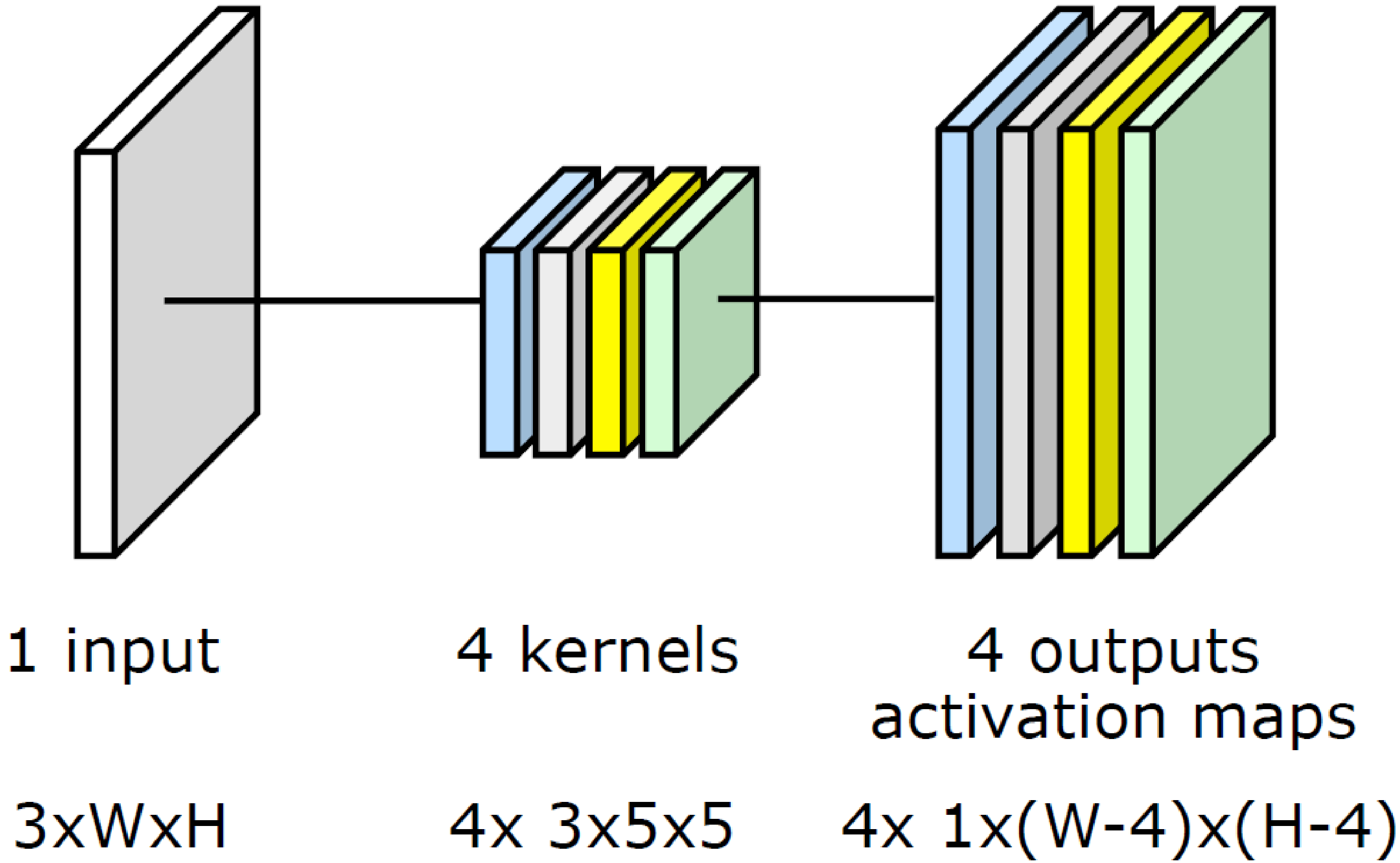
kernel size: $3 \times 5 \times 5 = 75$

→ dot product of 75 dim. vectors

number of such dot products:

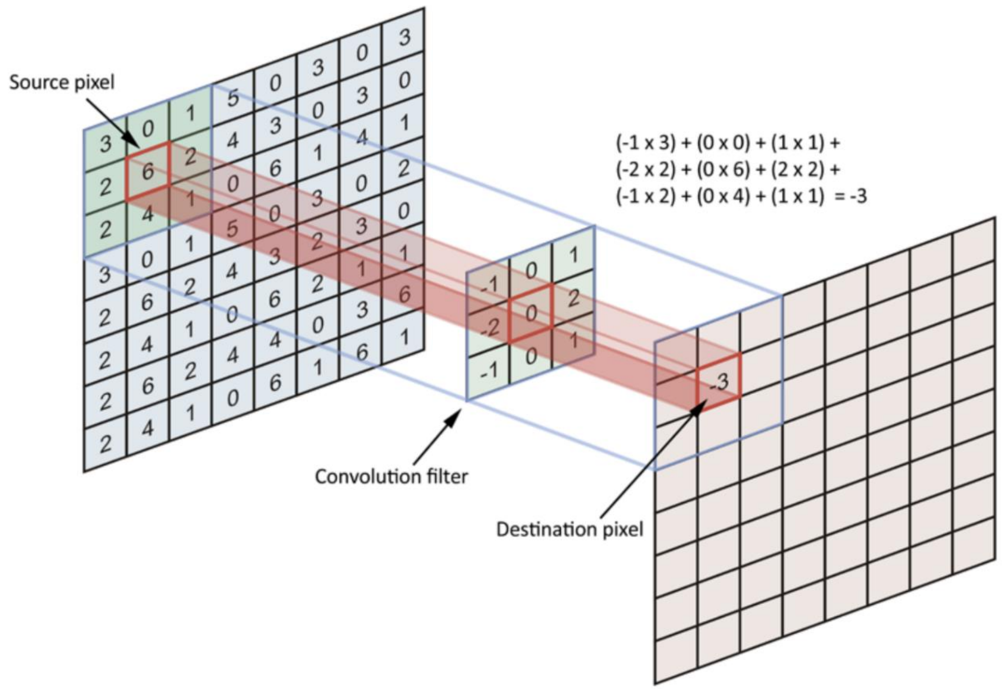
$$(256 - 4) \times (256 - 4) = 63.5k$$

Convolution Using Multiple Kernel



Convolution Kernels

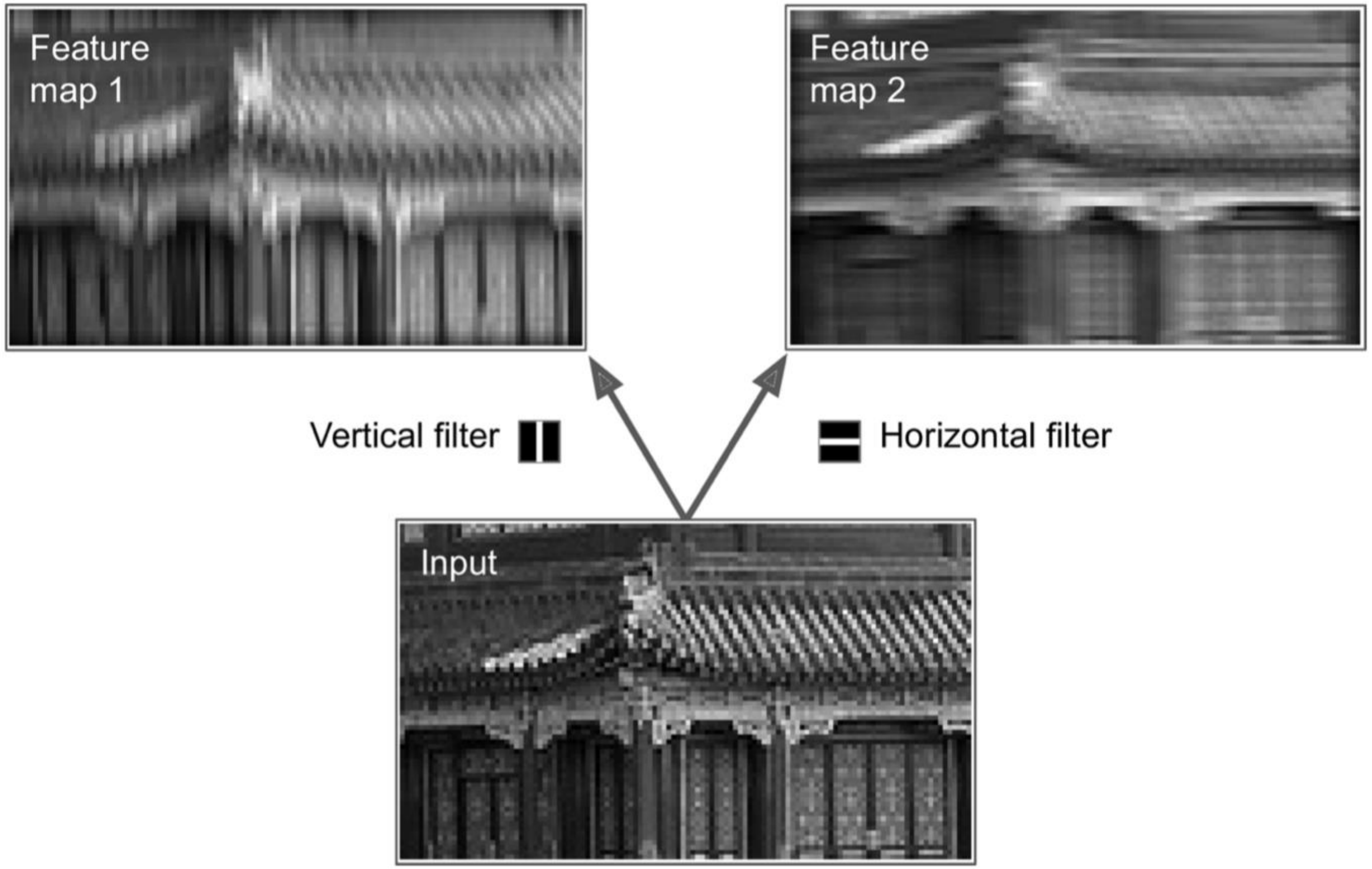
- A neuron's weight can be represented as a small image of the size of the receptive field
- We refer to sets of weights as filters or convolutional kernels



The convolution operation.

Convolution Kernels

- Let us consider two possible sets of weights, i.e., filters
- We obtain feature maps, highlighting the areas in an image that activate the filter the most



Output Sizes

Size of the activation map depends on

- Size of the input (Width, Height)
- Kernel size (K)

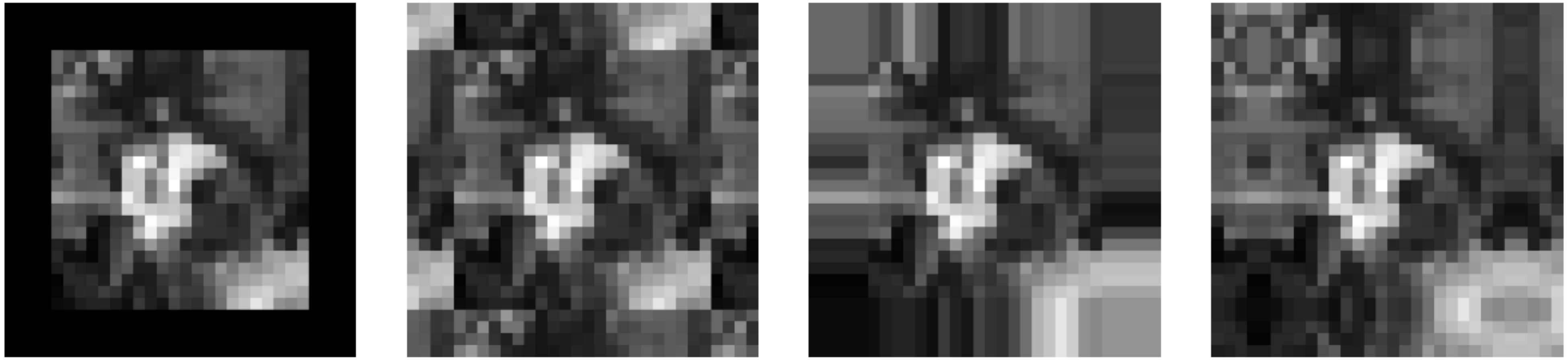
$$W' \times H' = (W - K + 1) \times (H - K + 1)$$

Output size

Input size and Kernel size

Padding

- Convolutions slightly shrink the image
- We can solve this by creating a border around the input image



zero wrap clamp mirror

CNNs often use zero-padding

Image courtesy: Szelinsky

Output Sizes with Padding

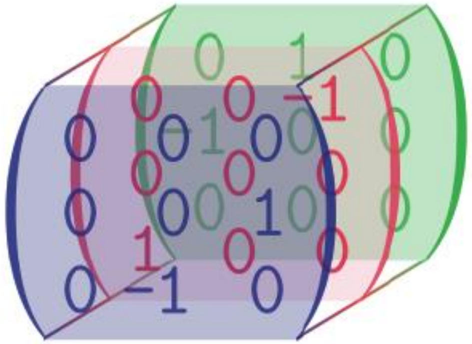
Size of the activation map depends on

- Size of the input (Width, Height)
- Kernel size (K)
- Padding (P)

$$W' \times H' = (W - K + 1 + 2P) \times (H - K + 1 + 2P)$$

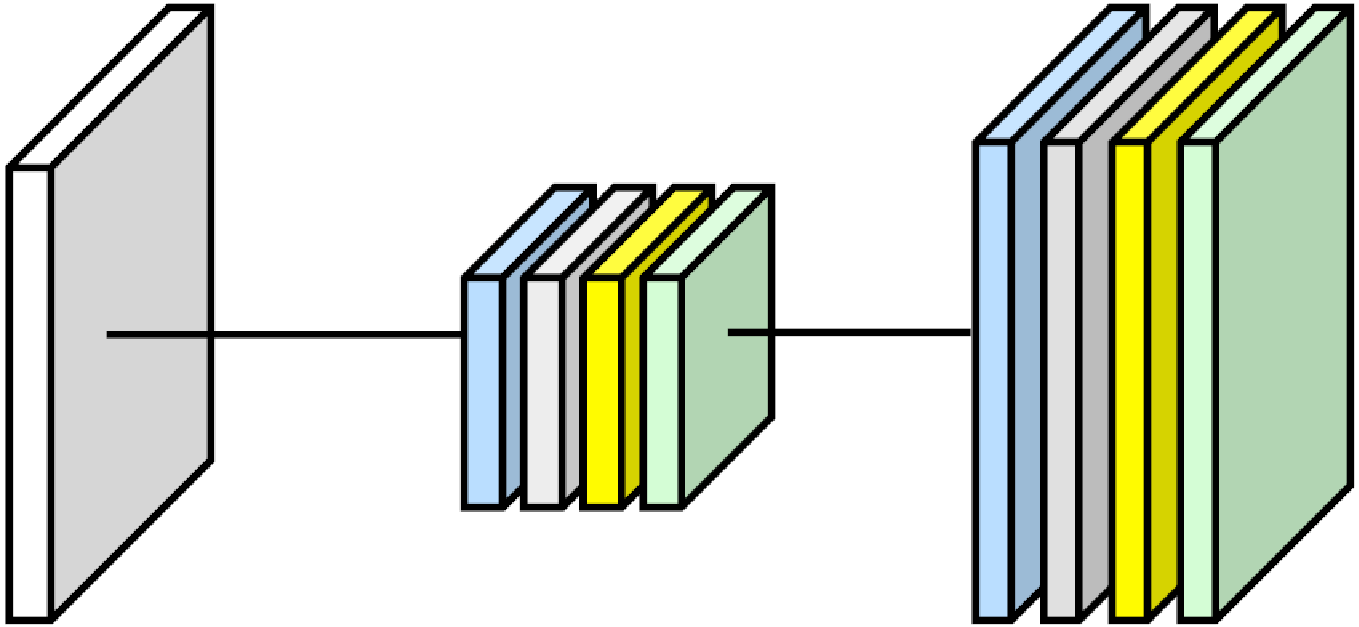
Tensor

- Vector is a 1-dimensional array
- Matrix is a 2-dimensional array
- Voxelgrid is a 3-dimensional array

$\epsilon_{ijk} =$ 

[Image courtesy: A. Kriesch]

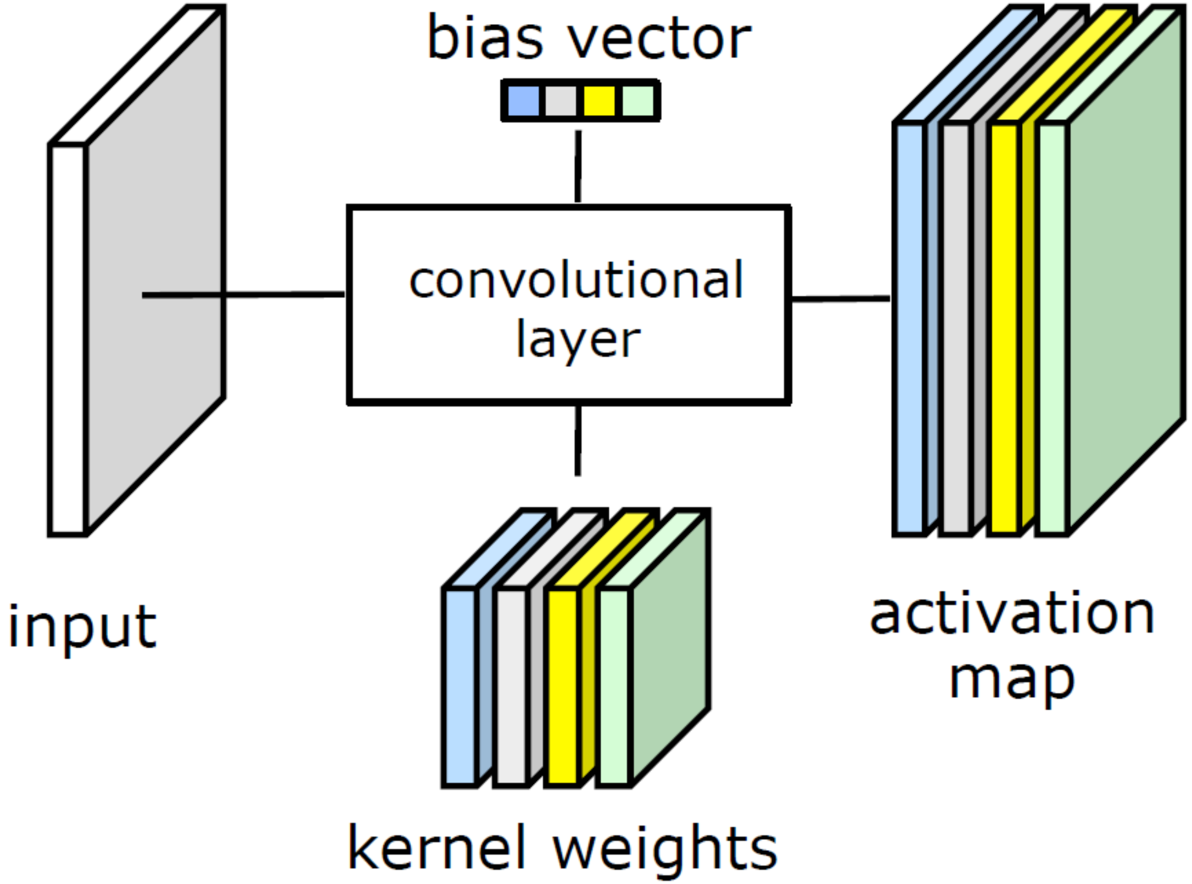
Each Layer is a Tensor



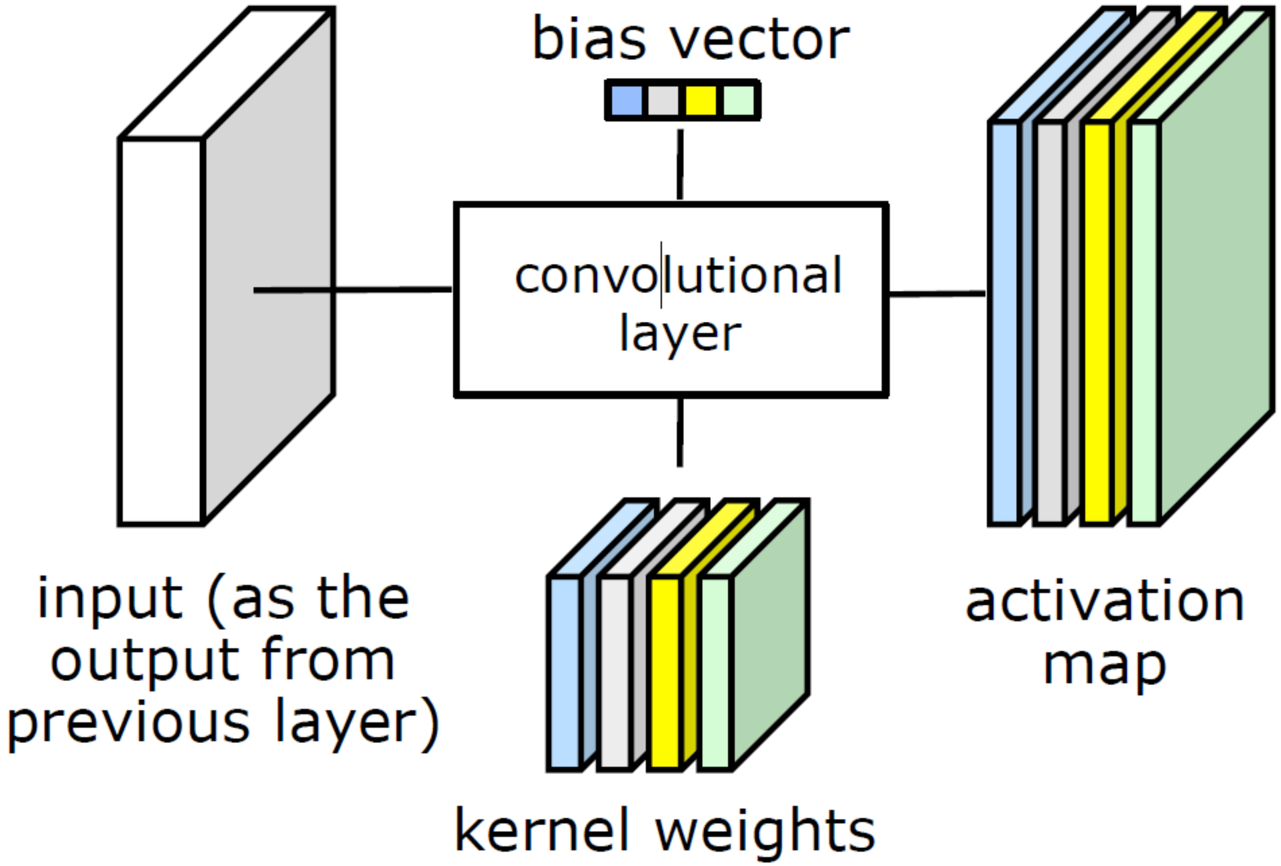
3D tensor | 4D tensor | 4D tensor

3xWxH | Nx3xKxK | Nx1xW’xH’

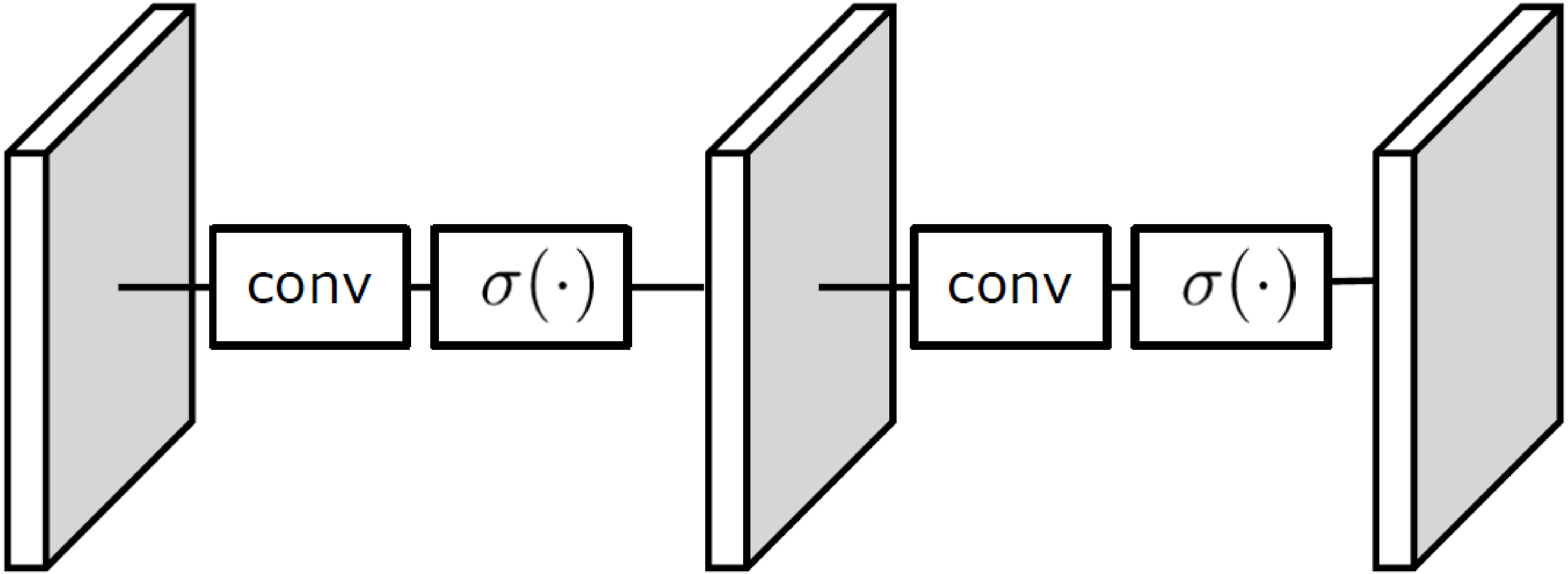
Convolutional Layer Parameters



Stacking Convolutional Layers

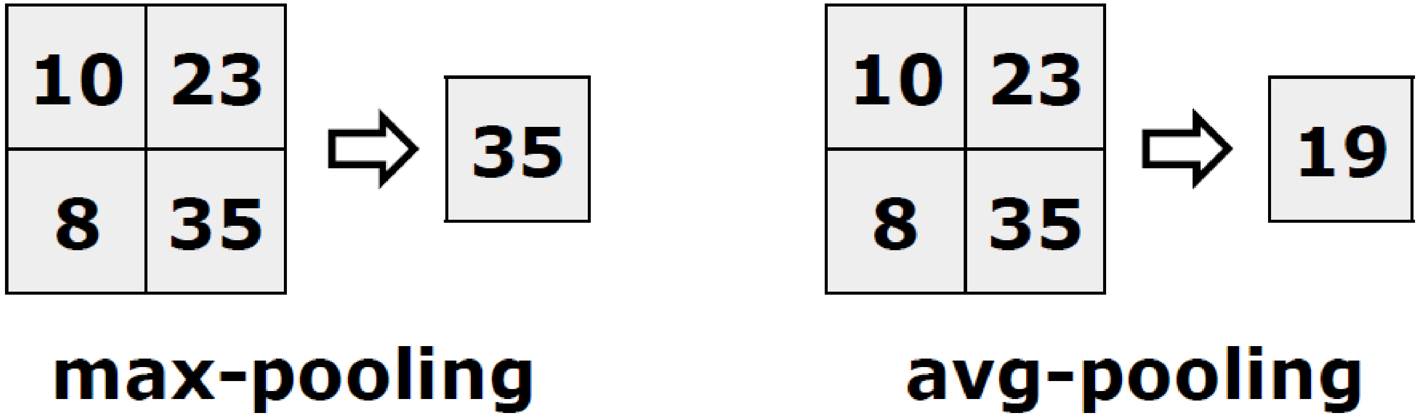


Stacking Convolutions Layers With Activation Functions



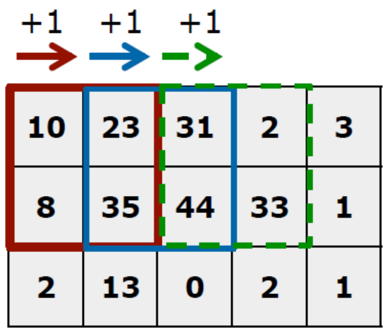
Pooling

- Besides convolutions, CNNs also use pooling layers
- Pooling combines multiple values into a single value to reduce the tensor sizes and combine information
- We want to reduce the computational load and memory usage
- Prominent examples are:

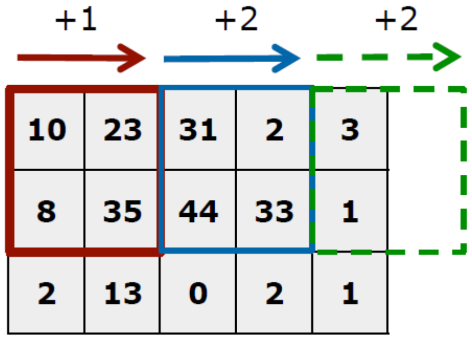


Stride

- Stride defines by how many pixels we shift the filter forward each step
- Larger stride reduces overlaps and makes the resulting image smaller



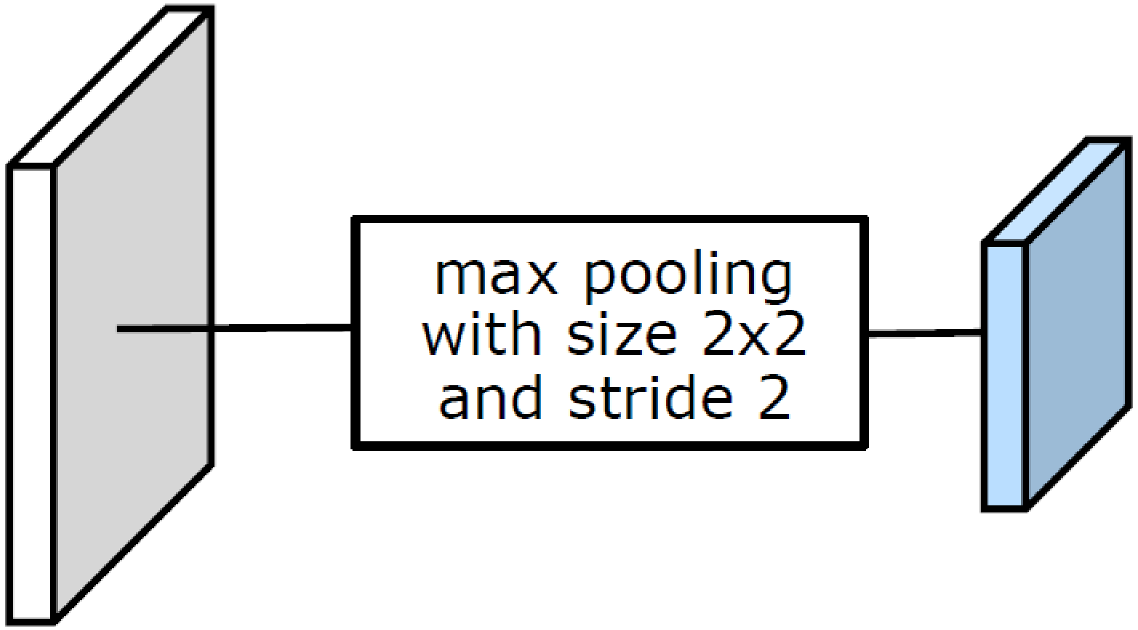
stride=1



stride=2

Max Pooling Example

Size tells us how many values to combine, and stride define by how much to shift the mask



$W \times H$ $\quad 2 \times 2$ $\quad W/2 \times H/2$

Max Pooling Example

2×2

10	23	31	2	3	34
8	35	44	33	1	45
2	13	0	2	1	7
12	3	8	22	9	88
22	88	3	0	2	0
1	9	33	3	4	4

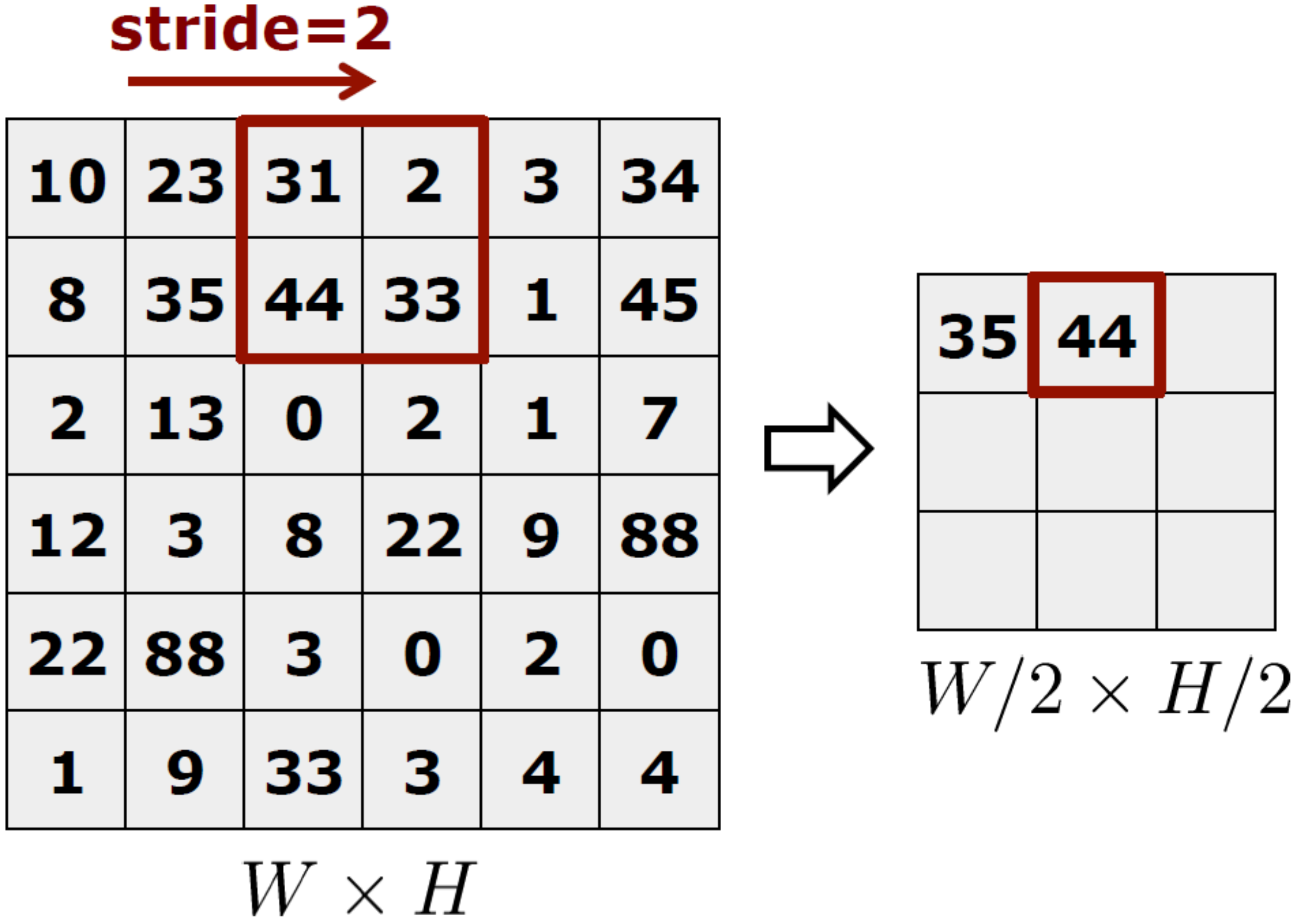
$W \times H$

$\Rightarrow$

35		

$W/2 \times H/2$

Max Pooling Example



Max Pooling Example

stride=2

10	23	31	2	3	34
8	35	44	33	1	45
2	13	0	2	1	7
12	3	8	22	9	88
22	88	3	0	2	0
1	9	33	3	4	4

$W \times H$

$\Rightarrow$

35	44	45

$W/2 \times H/2$

Max Pooling Example

10	23	31	2	3	34
8	35	44	33	1	45
2	13	0	2	1	7
12	3	8	22	9	88
22	88	3	0	2	0
1	9	33	3	4	4

$W \times H$

35	44	45
13		

$W/2 \times H/2$

Max Pooling Example

10	23	31	2	3	34
8	35	44	33	1	45
2	13	0	2	1	7
12	3	8	22	9	88
22	88	3	0	2	0
1	9	33	3	4	4

$W \times H$

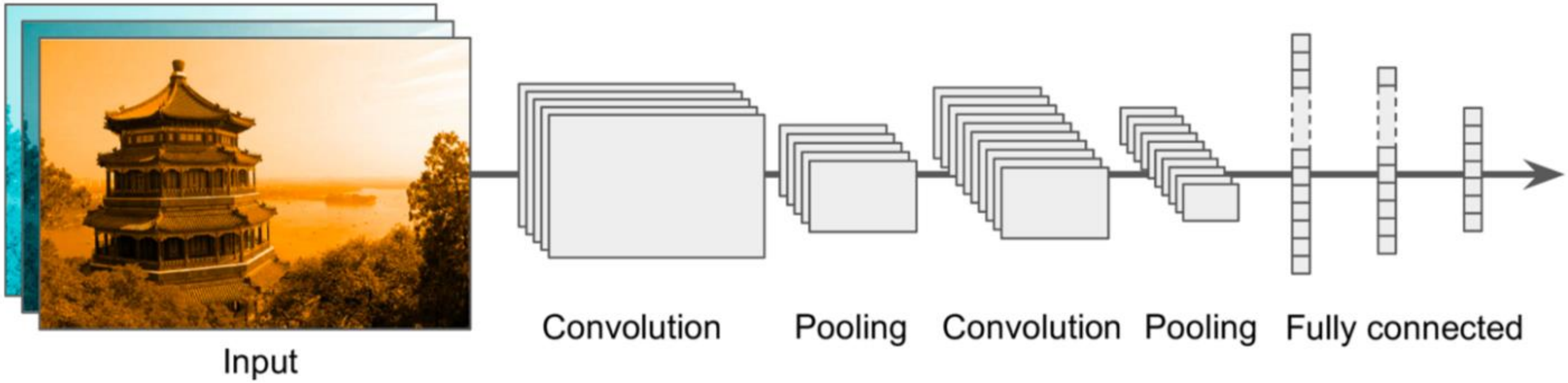
35	44	45
13	22	88
88	33	4

$W/2 \times H/2$

stride and size determine the output size

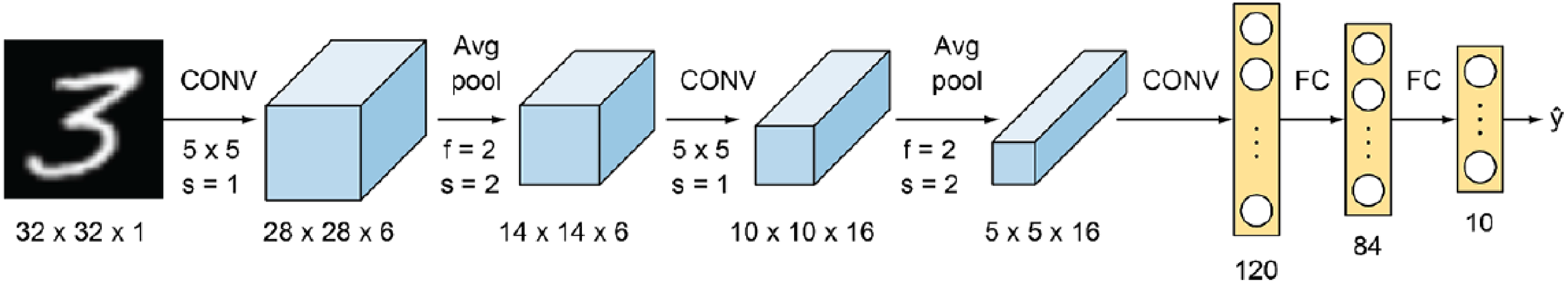
Training CNNs

- Like MLPs, CNNs are trained using SDG and backpropagation
- Large number of parameters need to be determined
- Fairly large training sets are needed (end-to-end vs. given features)
- A regular feedforward neural network is added composed of fully connected layers
- Final layer outputs the prediction (e.g., a softmax layer that outputs estimated class probabilities)

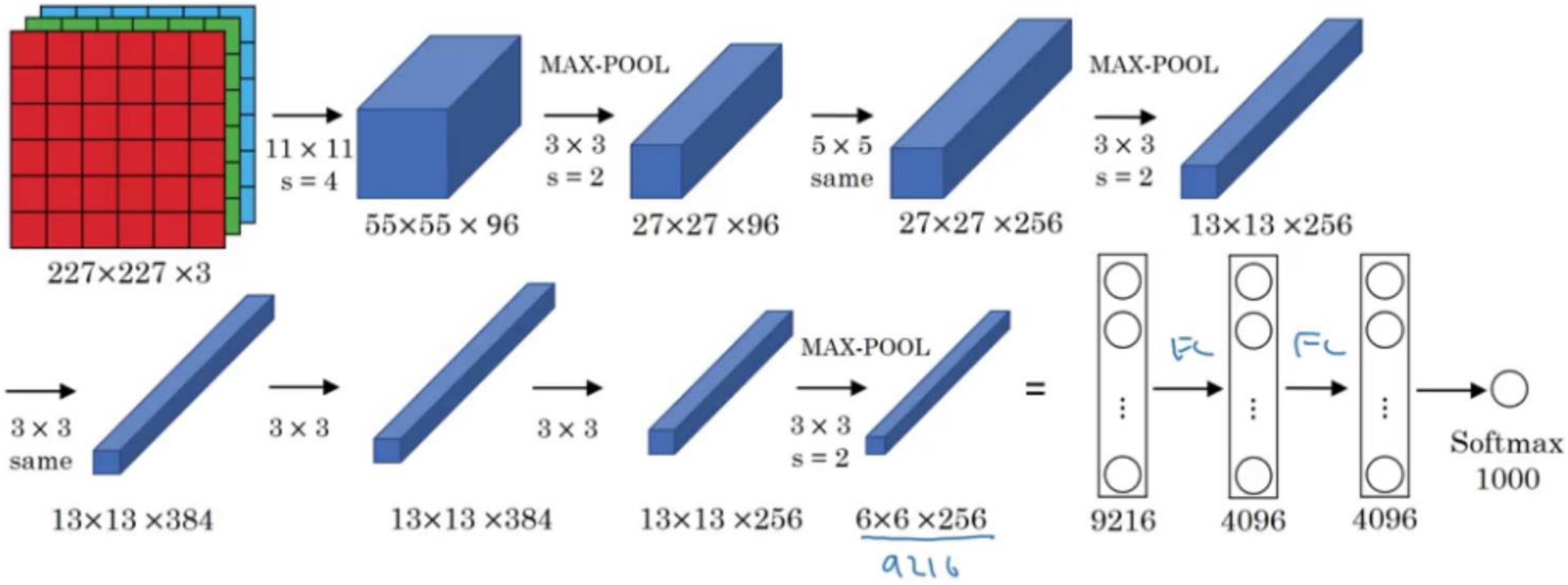


LeNet-5 by LeCun et al., 1989

- First convolutional network proposed by Yann LeCun et al.
- Recognition of handwritten digits
- Outperformed all other networks (at that time)



AlexNet (2012)

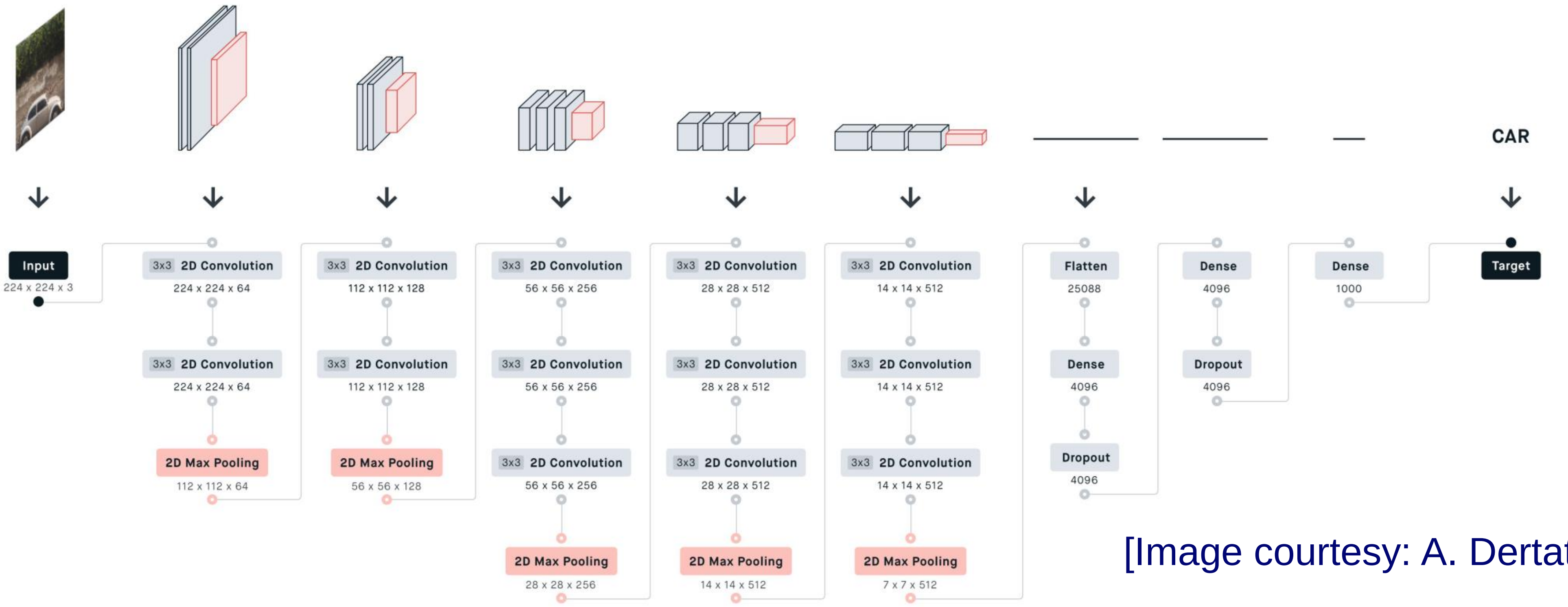


This network is similar to LeNet-5 with just more convolution and pooling layers:

- **Parameters:** 60 million
- **Activation function:** ReLu

VGG-16/19 (2014)

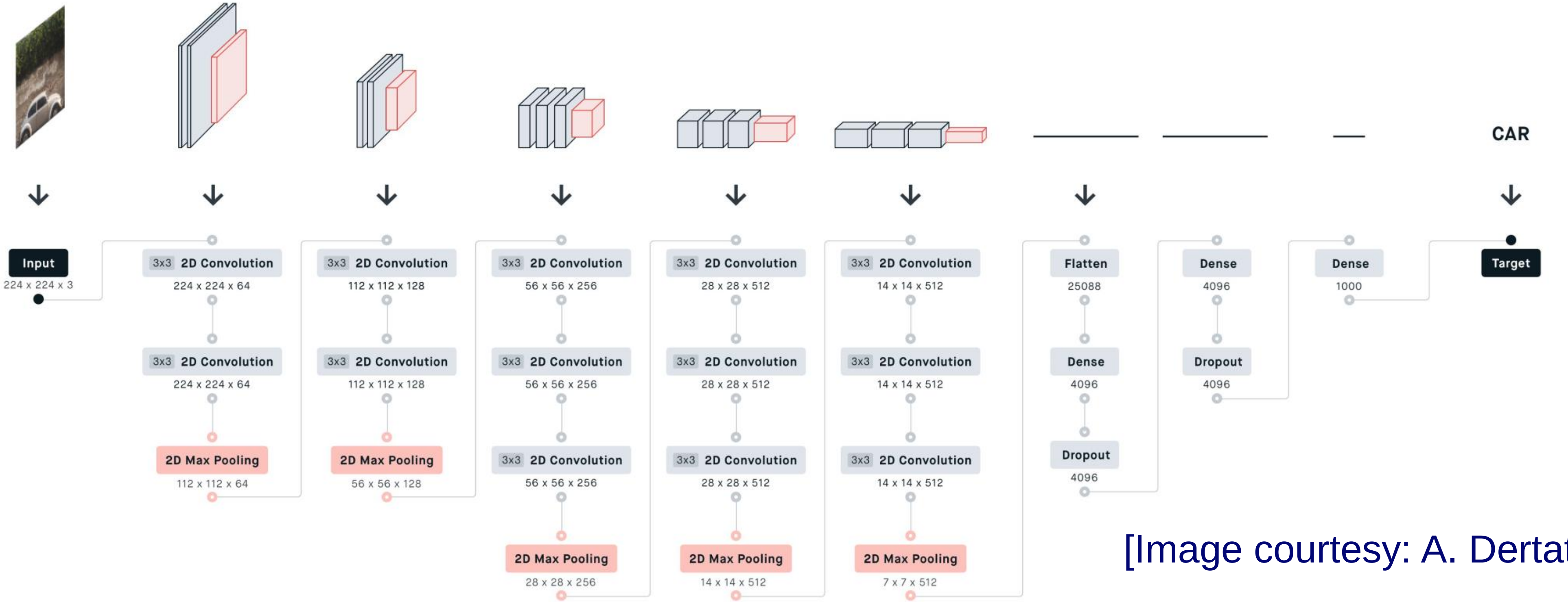
- Stacks elements from AlexNet using smaller filters
- 16/19 layers
- 138M parameters (16 layer version)



[Image courtesy: A. Dertat]

VGG-16/19 (2014)

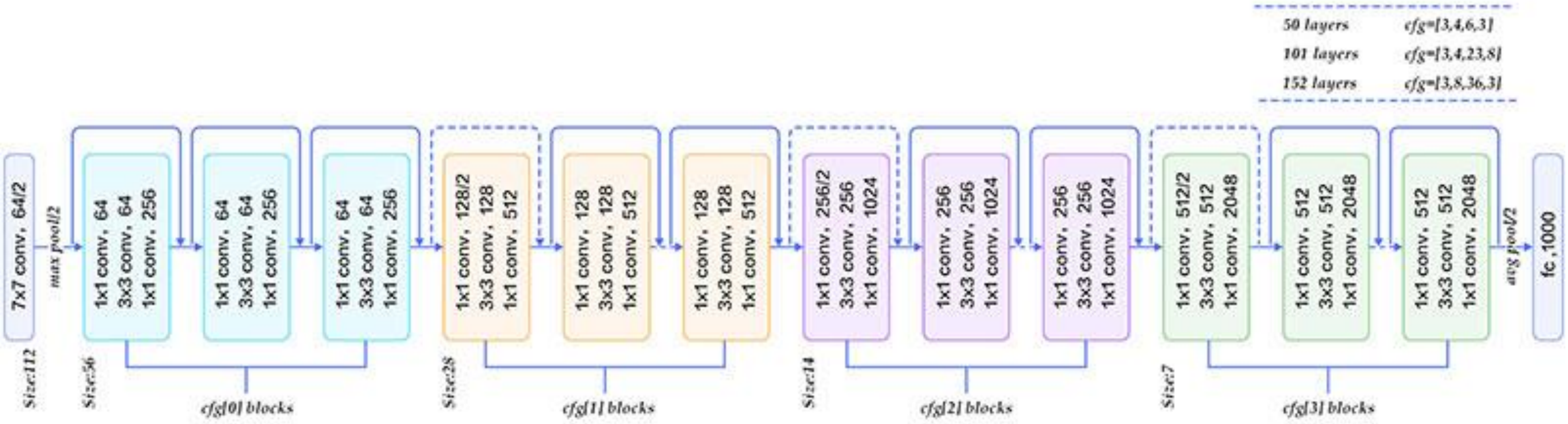
- Stacks elements from AlexNet using smaller filters
- 16/19 layers
- 138M parameters (16 layer version)



[Image courtesy: A. Dertat]

VGG-16/19 (2014)

- Very deep network:152 layers
- Consists of residual blocks



[Image courtesy: A. Dertat]