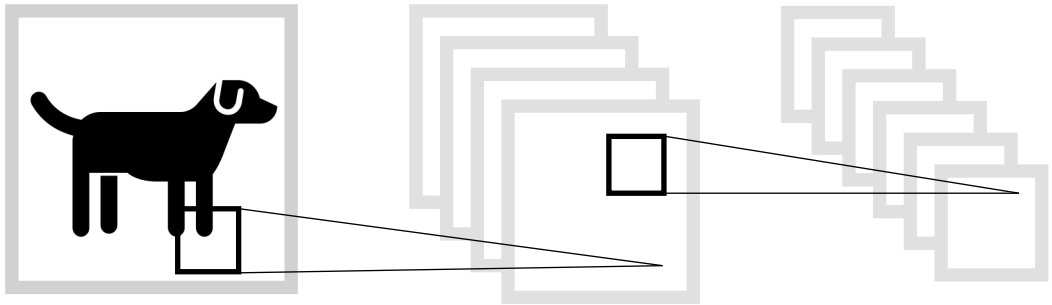
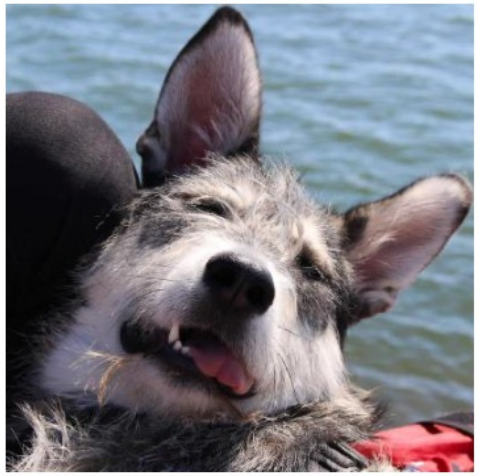
Convolutional NN



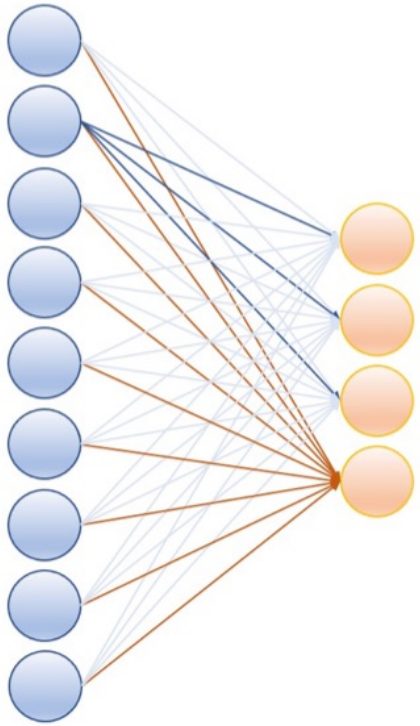
CS-EJ3311 - Deep Learning with Python
24.10.-11.12.2022
Aalto University & FiTech.io

14.11.2022 Shamsi Abdurakhmanova

Fully (densely) connected layer

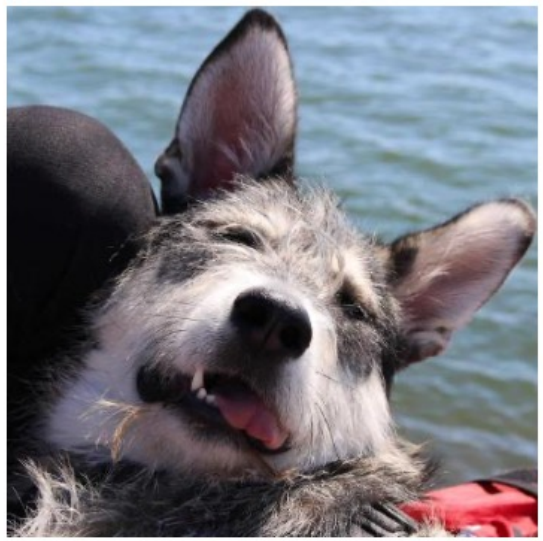


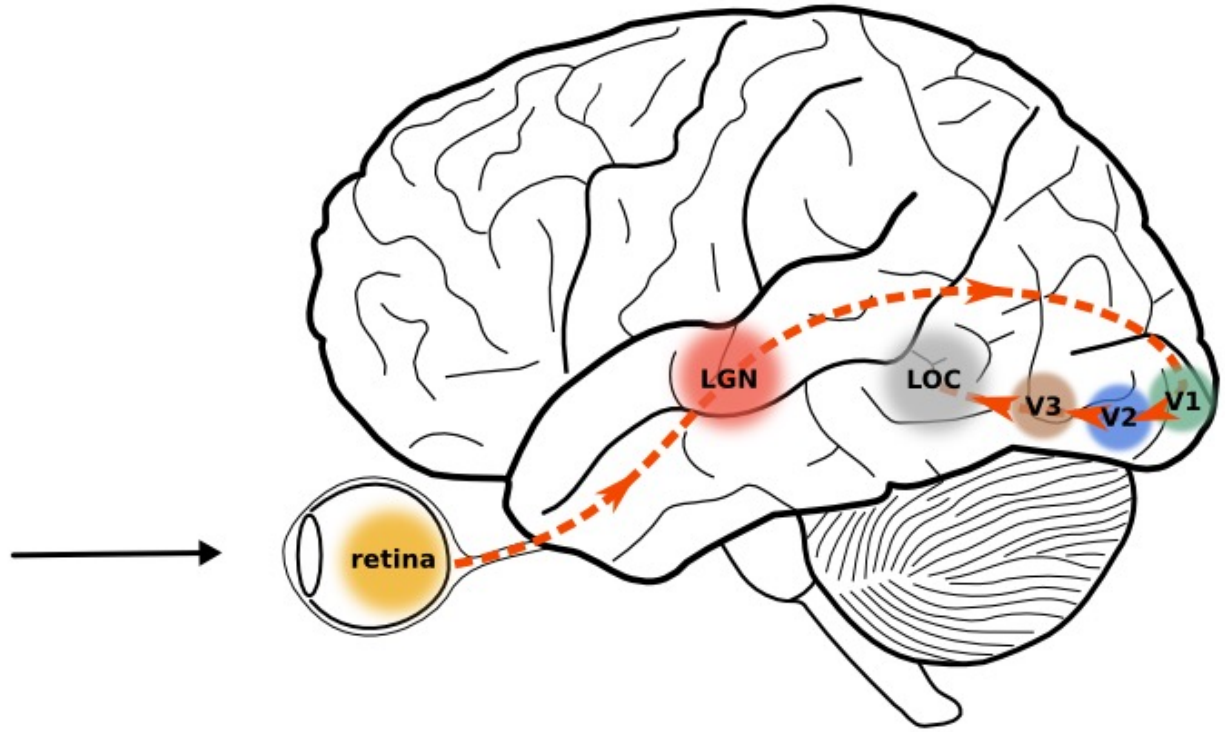
flatten



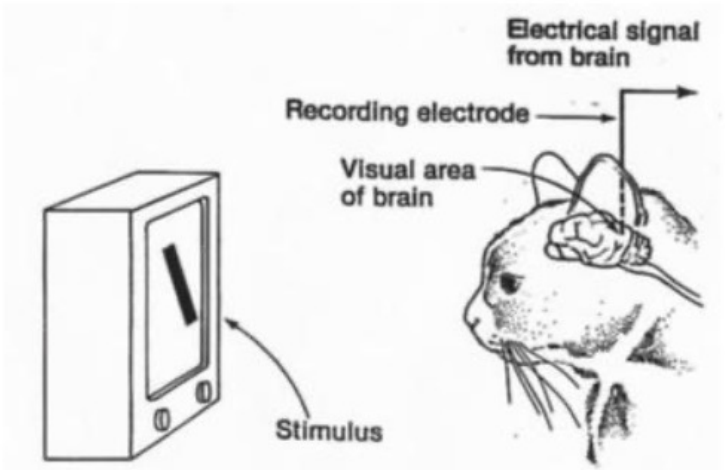
- 100 x 100 image
- 100 neurons in hidden layer

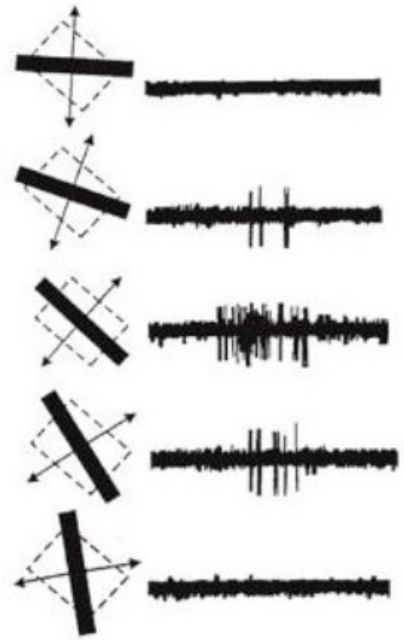
→ ≈ 1.000.000 parameters

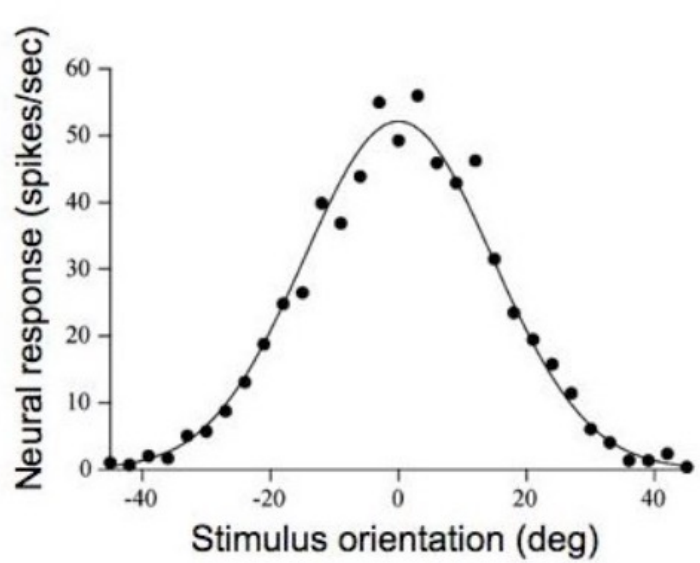



LGN
LOC
V3
V2
V1
retina

David Hubel and Torsten Wiesel ~1960
Neural Basis of Visual Perception

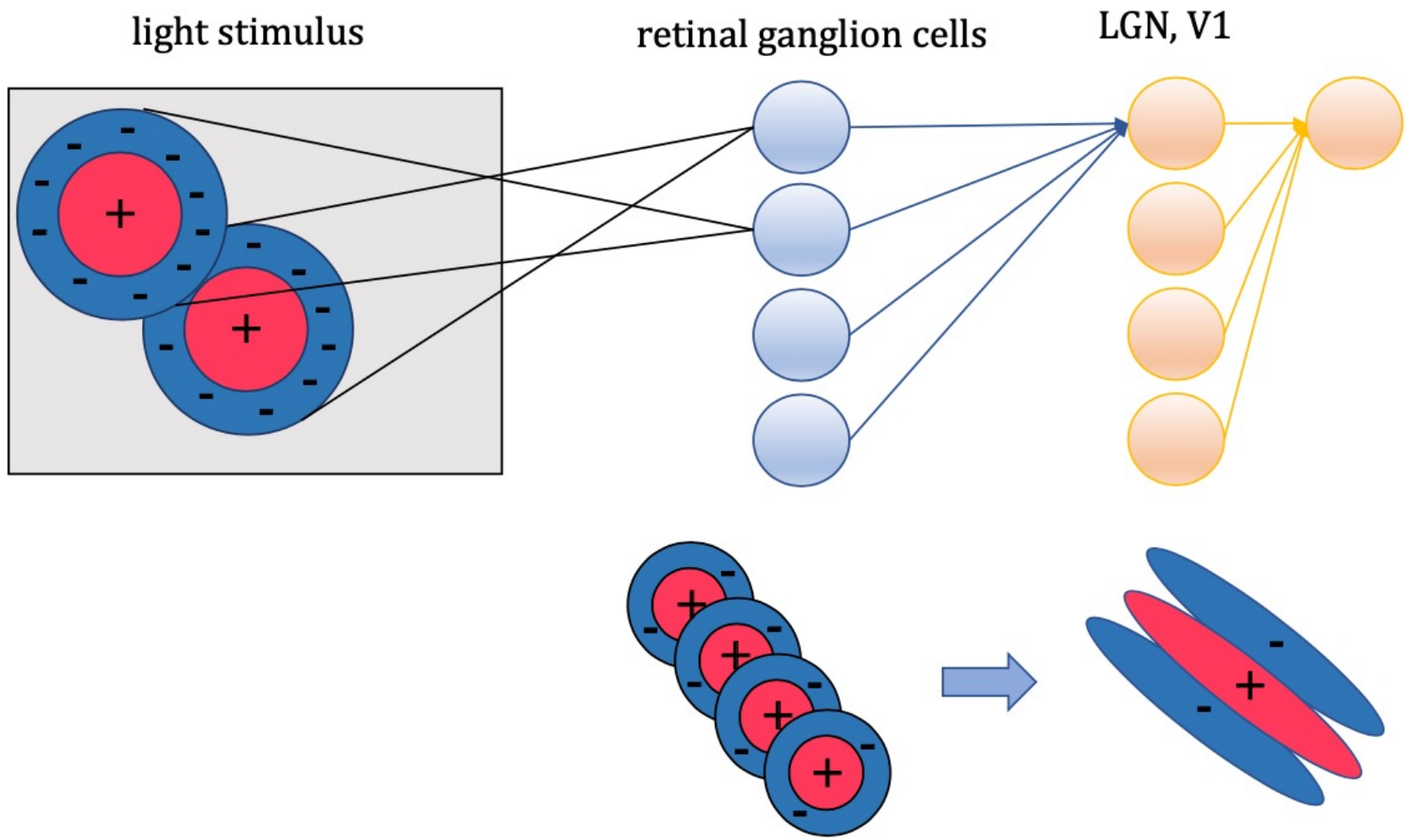




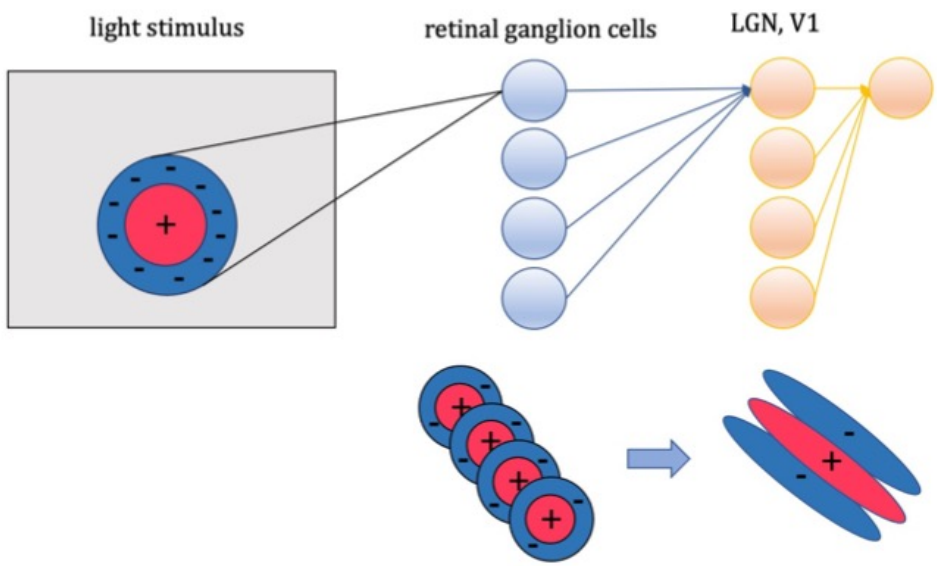


Hubel & Wiesel, 1968

Receptive field of the neurons

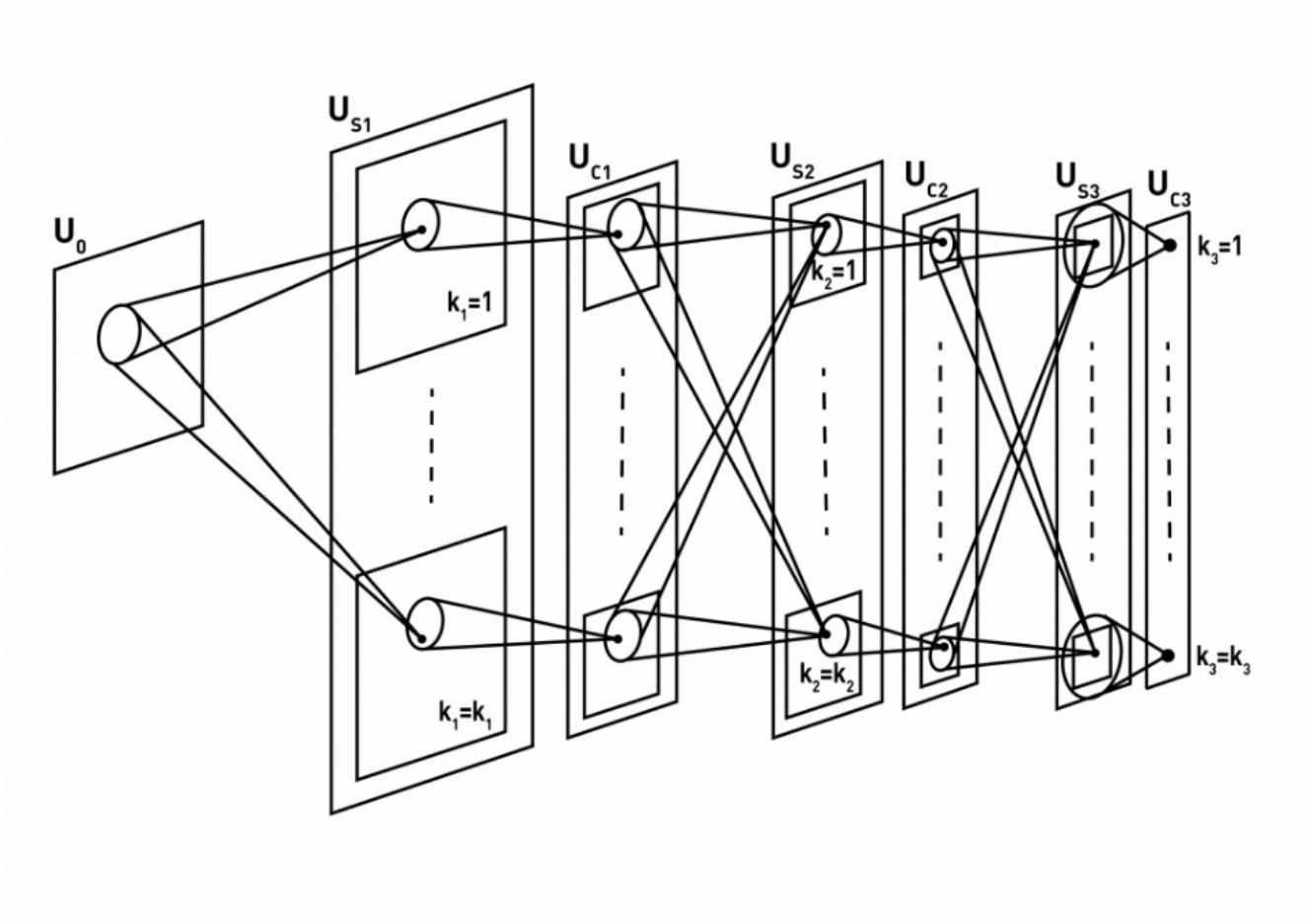


Receptive field of the neurons



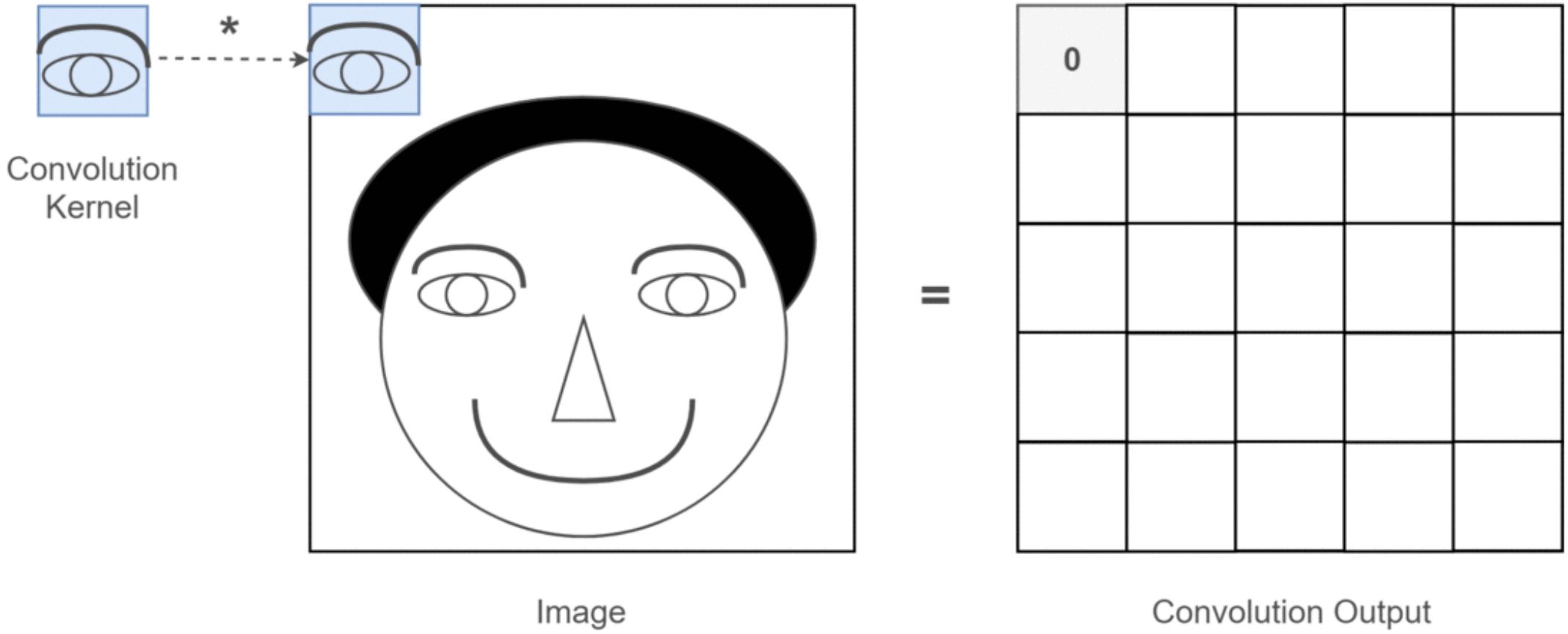
- idea of receptive field, local connections
- idea of hierarchical structure, from simple elements to complex patterns

Neocognitron - biologically inspired ANN



[Fukushima K. (1988) Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition.]

Feature detection with filters



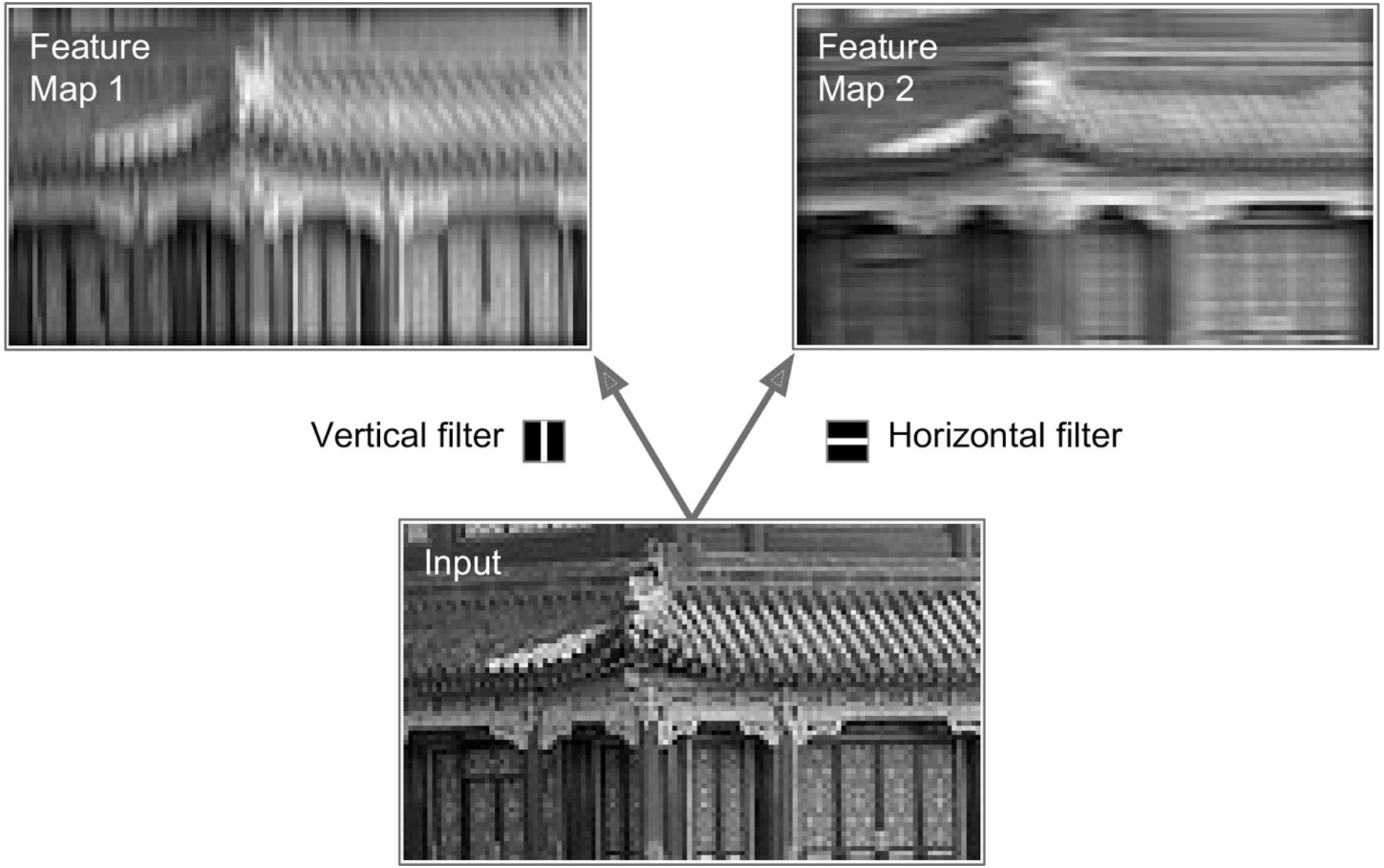
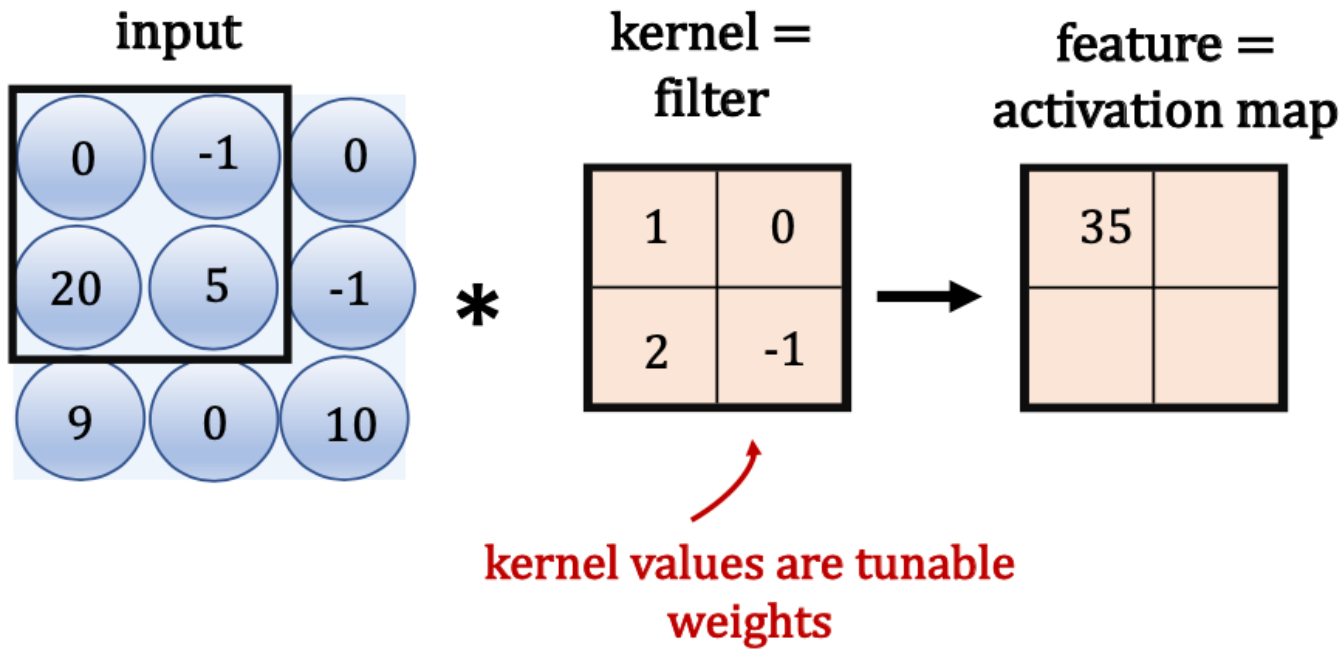


Figure 13-5. Applying two different filters to get two feature maps
"Hands-On Machine Learning …" A.Géron

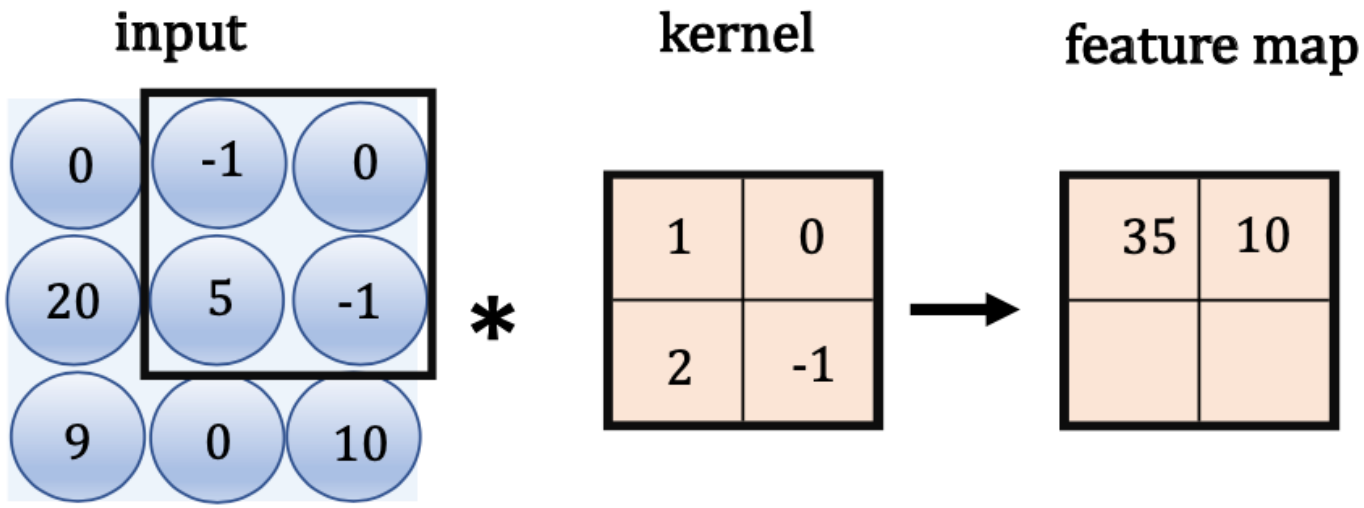
Convolution operation



$$0 * 1 + (-1) * 0 + 20 * 2 + 5 * (-1) = 35$$

(for simplicity we do not add bias)

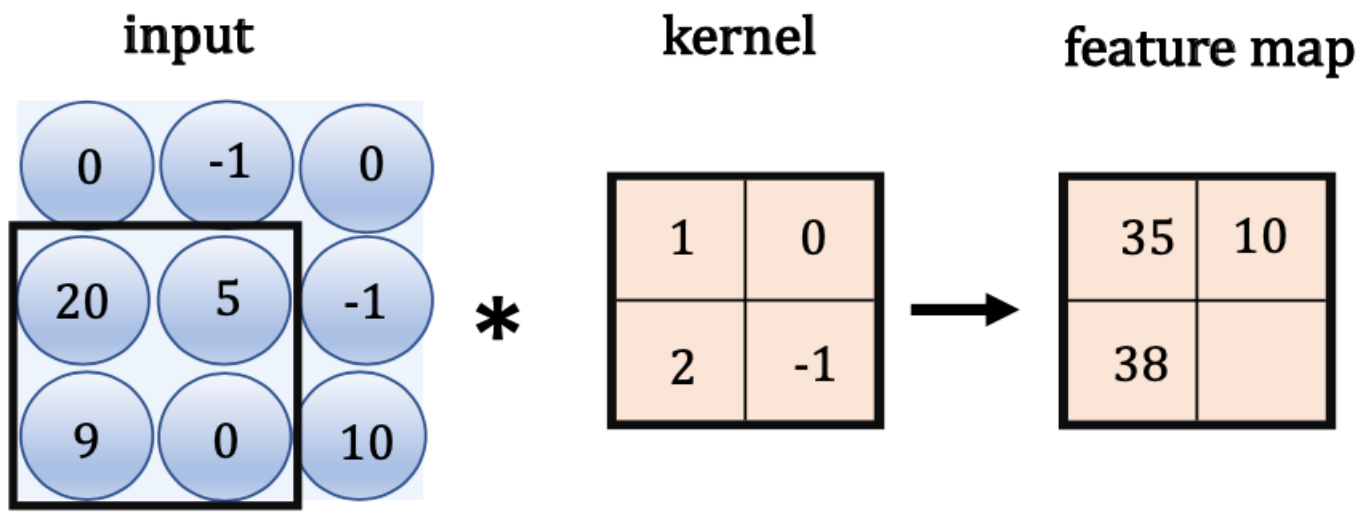
Convolution operation



$$(-1) * 1 + 0 * 0 + 5 * 2 + (-1) * (-1) = 10$$

(no bias)

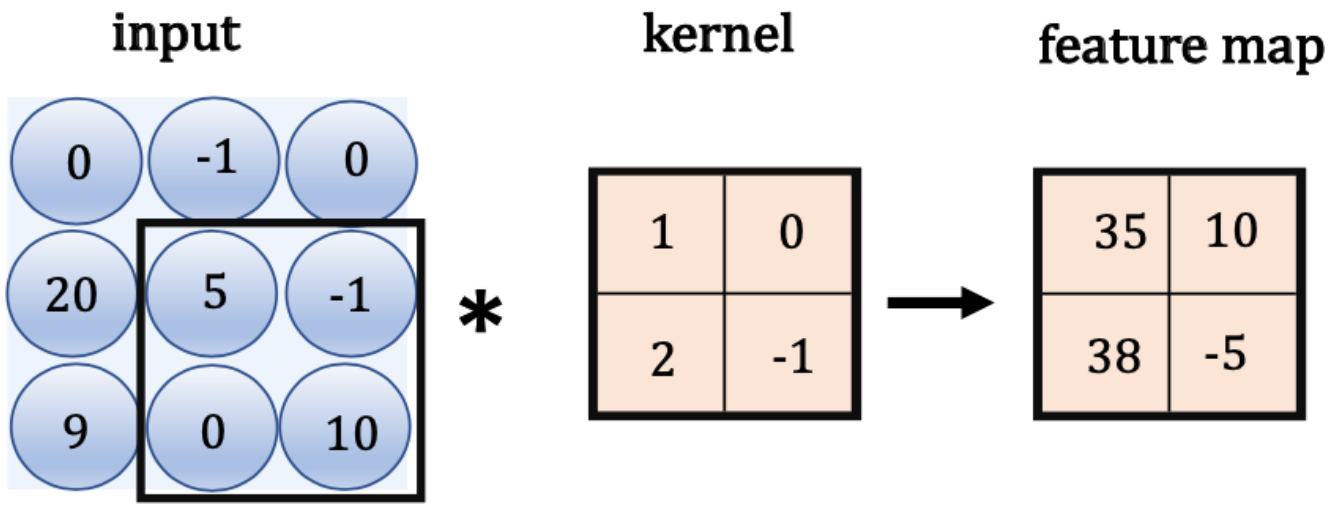
Convolution operation



$$20 * 1 + 5 * 0 + 9 * 2 + 0 * (-1) = 38$$

(no bias)

Convolution operation

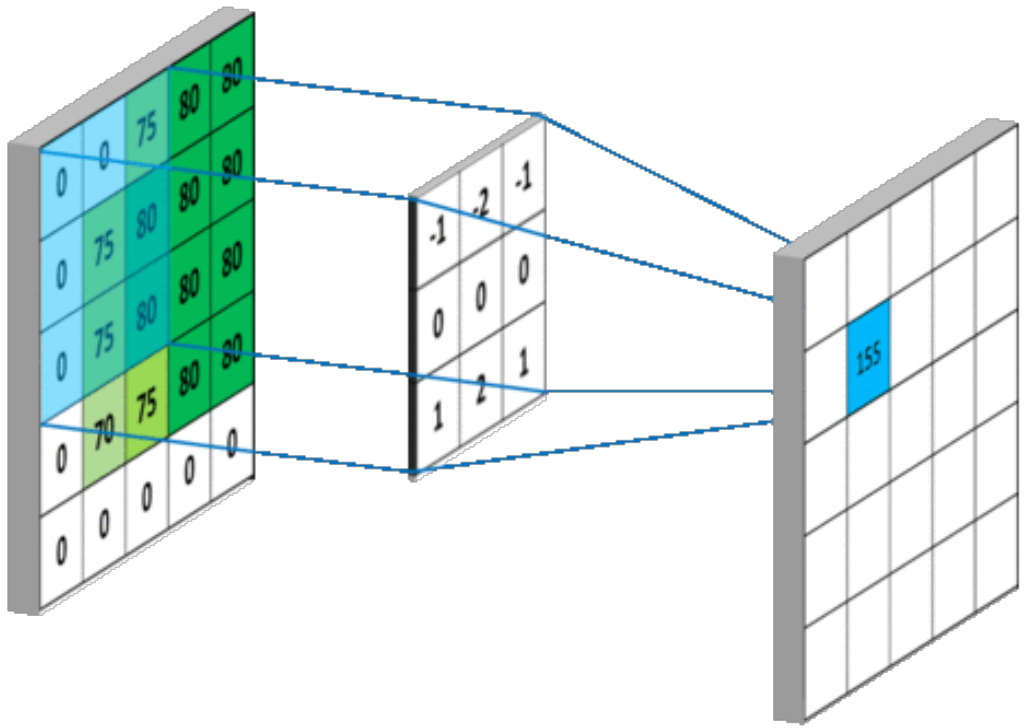


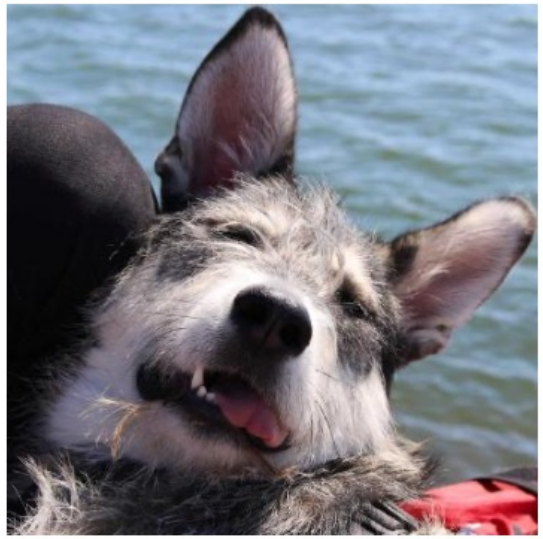
$$5 * 1 + (-1) * 0 + 0 * 2 + 10 * (-1) = -5$$

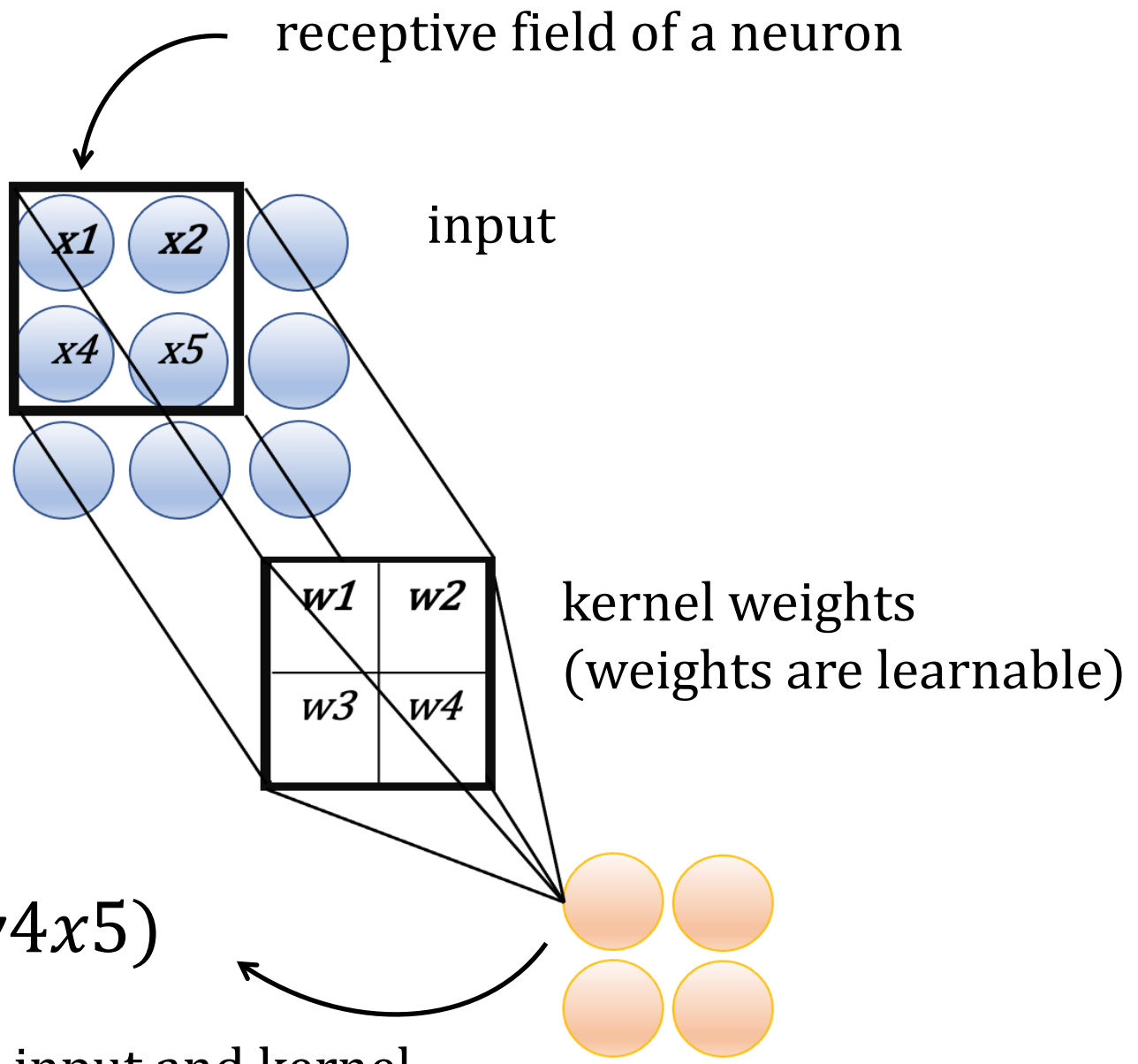
(no bias)

Note reduction in size (3,3) → (2,2)

Convolution

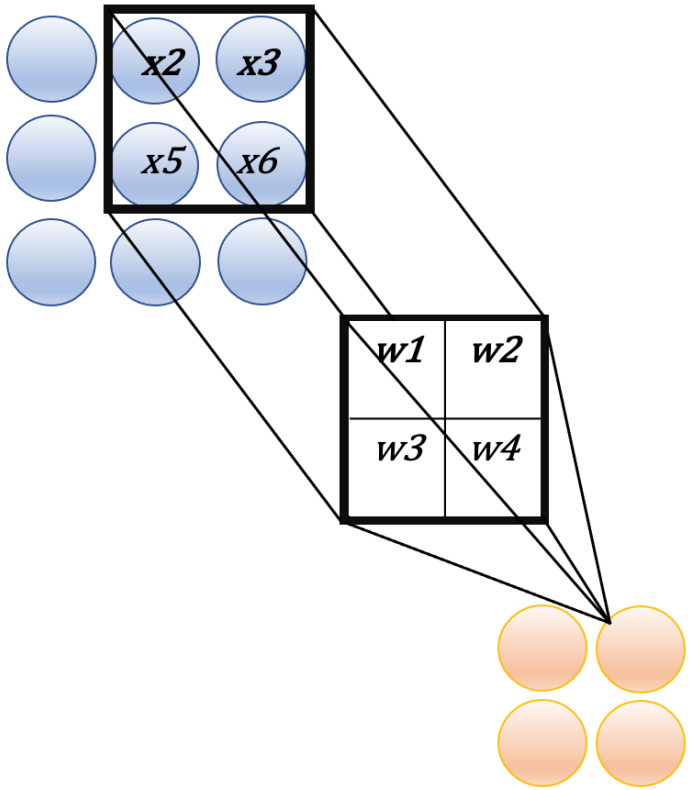


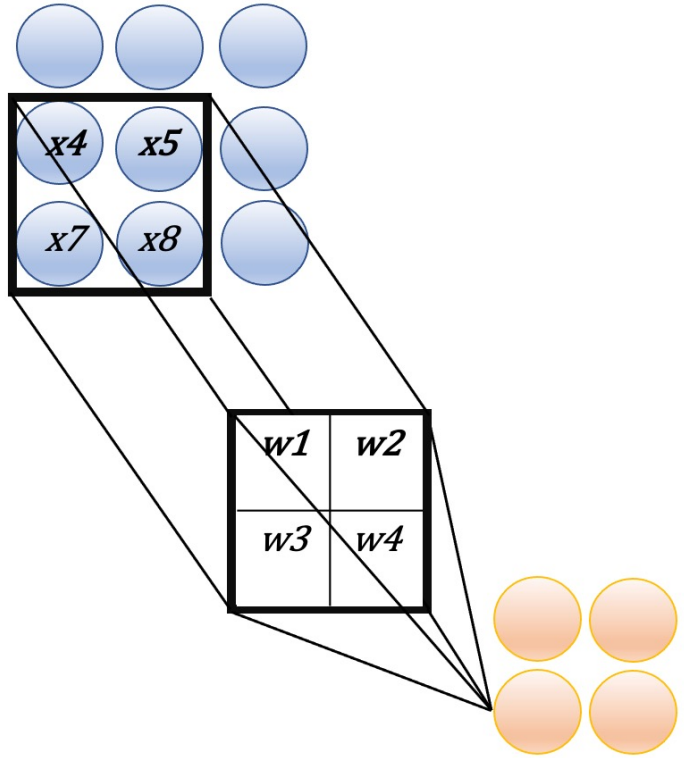


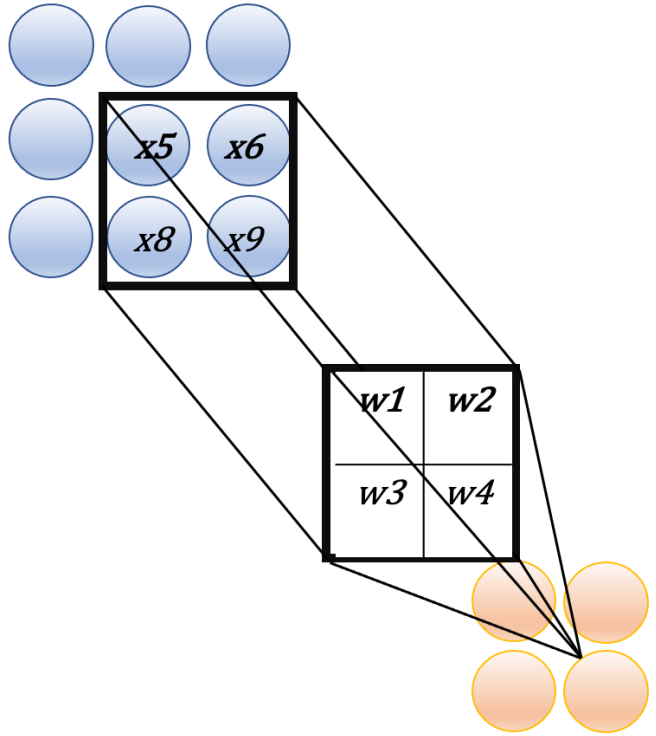


$$\sigma(w1x1 + w2x2 + w3x4 + w4x5)$$

output (activation) = convolution between input and kernel





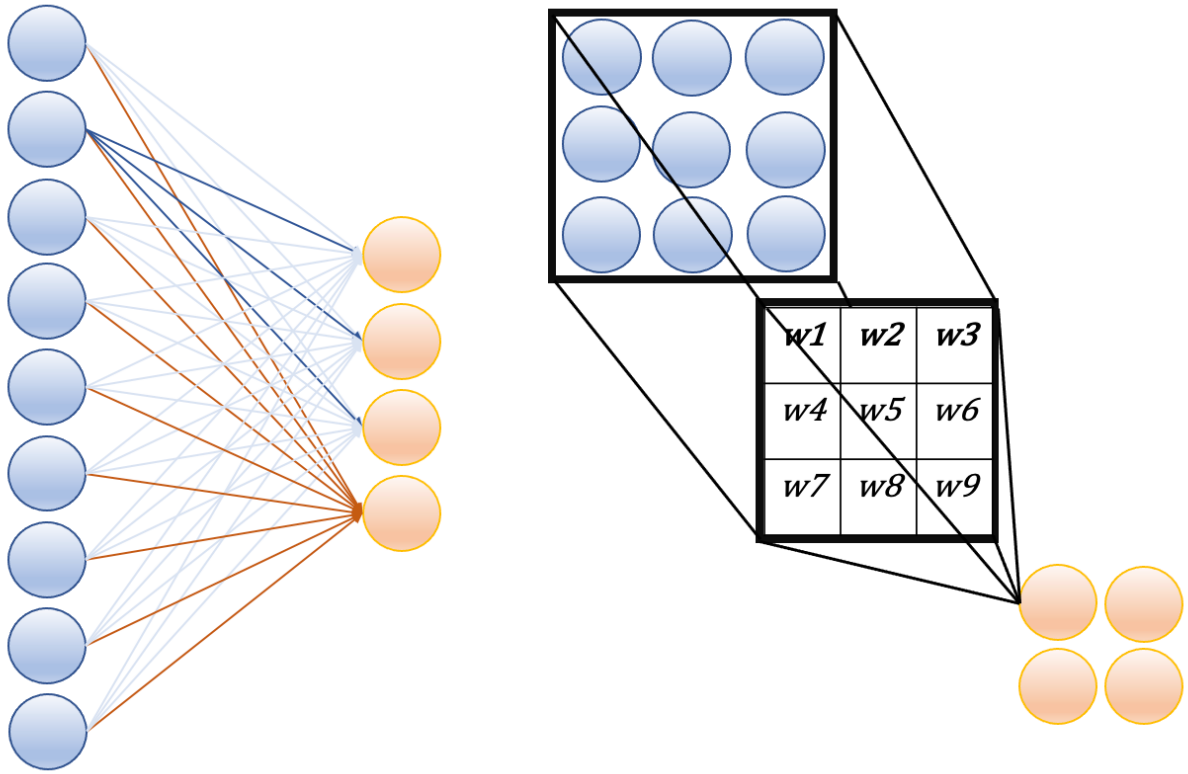


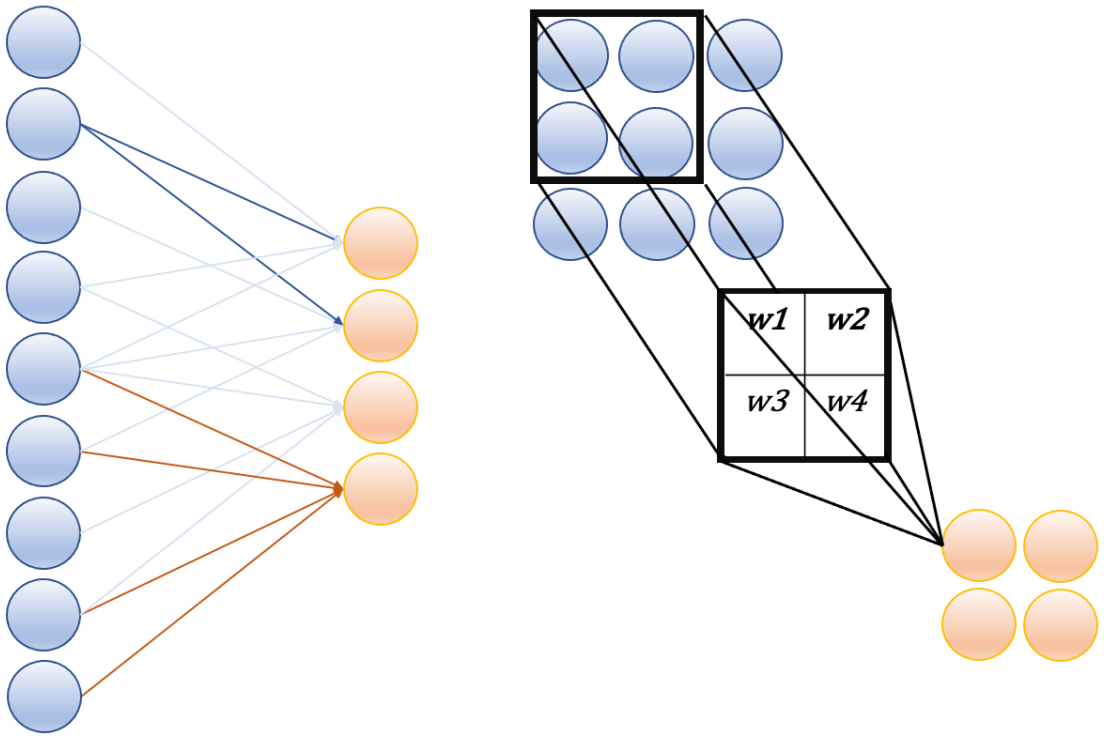
$\sigma(w1x2 + w2x3 + w3x5 + w4x6)$

$\sigma(w1x4 + w2x5 + w3x7 + w4x8)$

$\sigma(w1x5 + w2x6 + w3x8 + w4x9)$

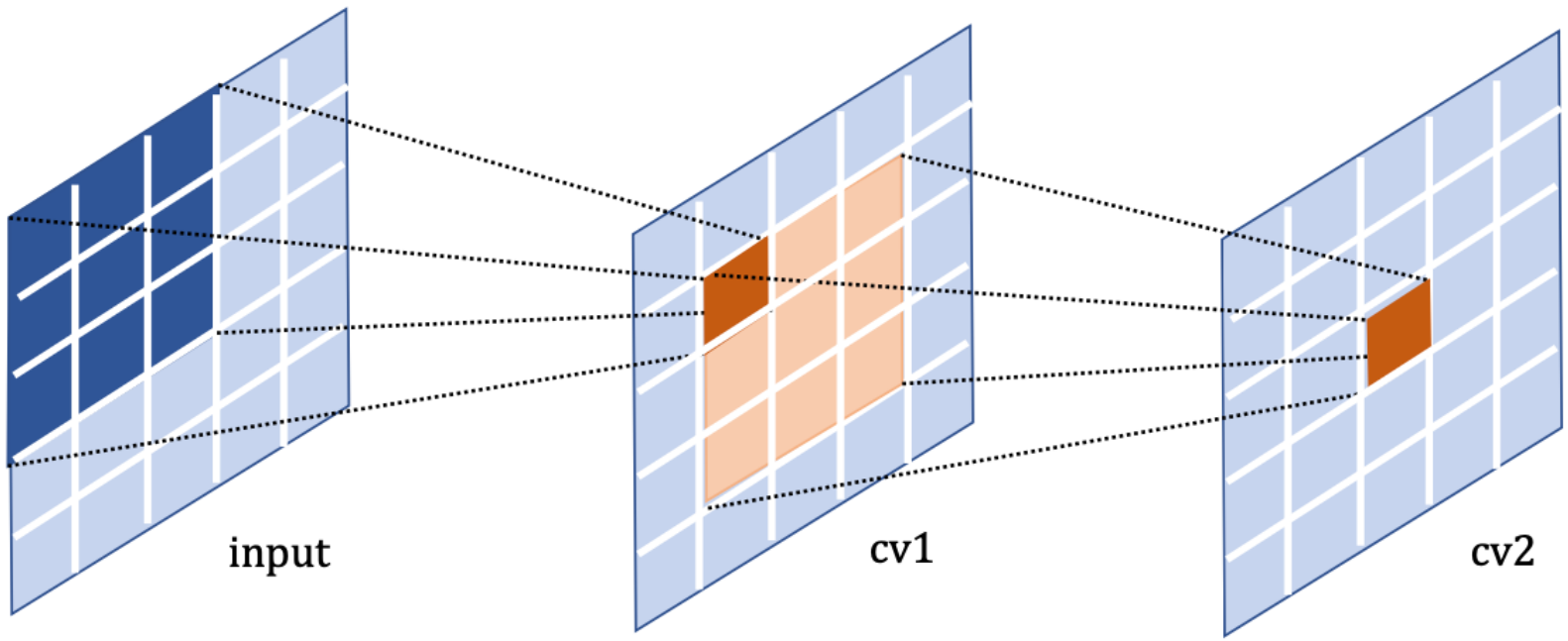
Dense vs Convolutional layers





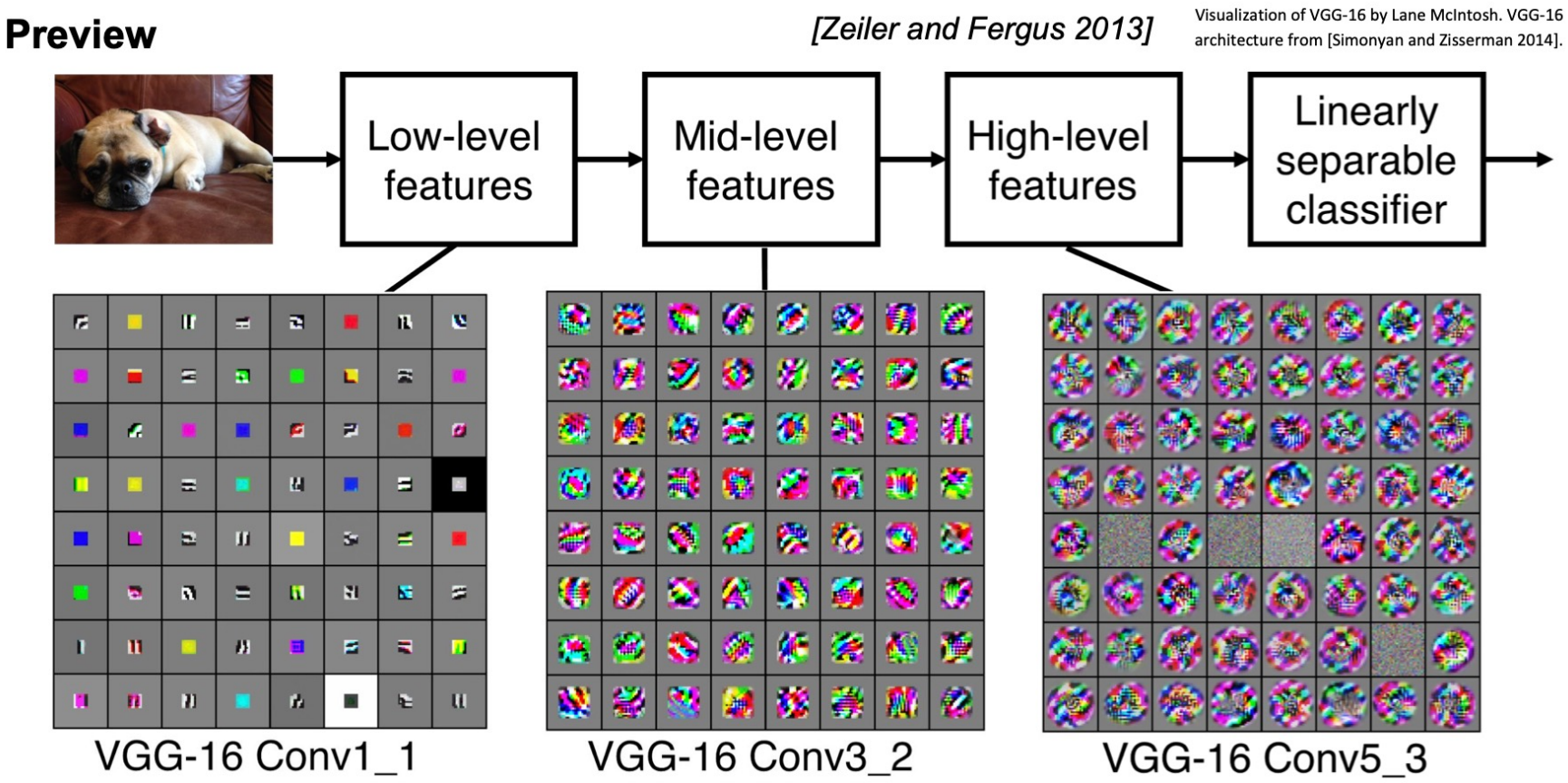
- local connections
- shared weights → lower # of parameters
- spactial structure is preserved

Stack of convolutional layers



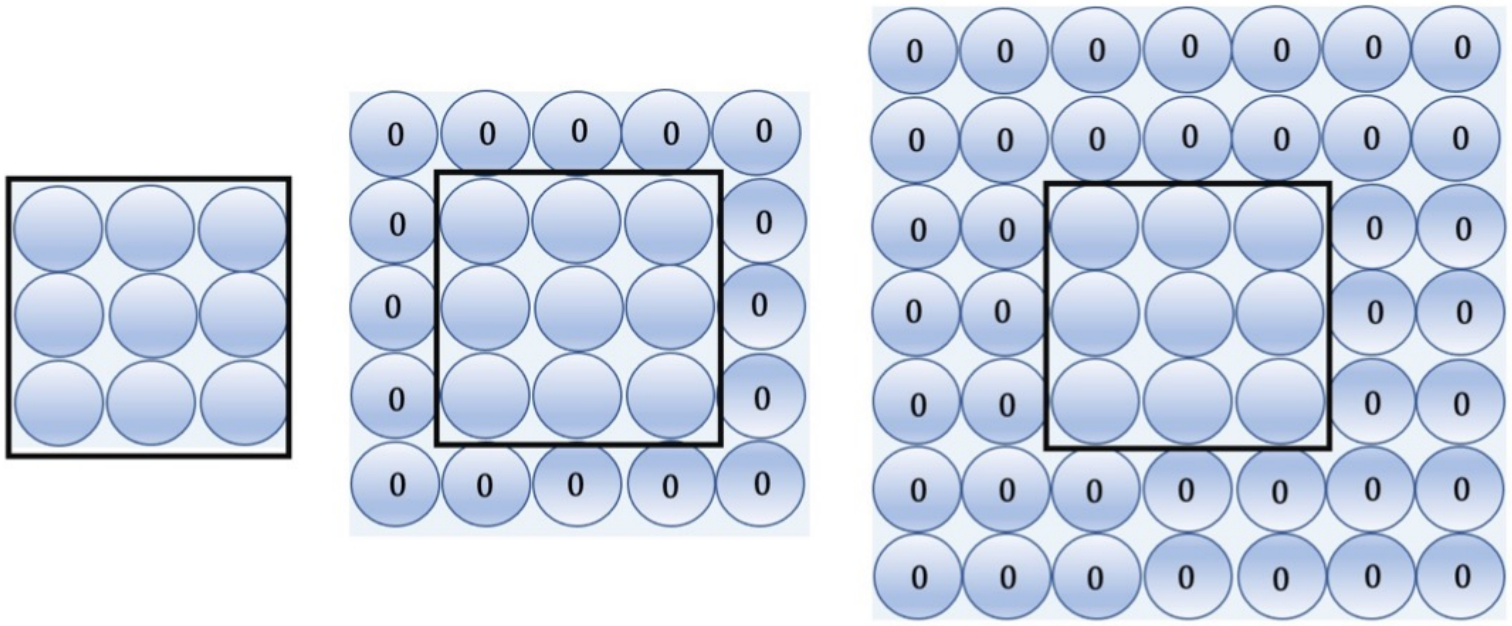
Note, that the size of receptive field becomes larger for deep layers (hierarchical structure)

Activation maps



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

Padding

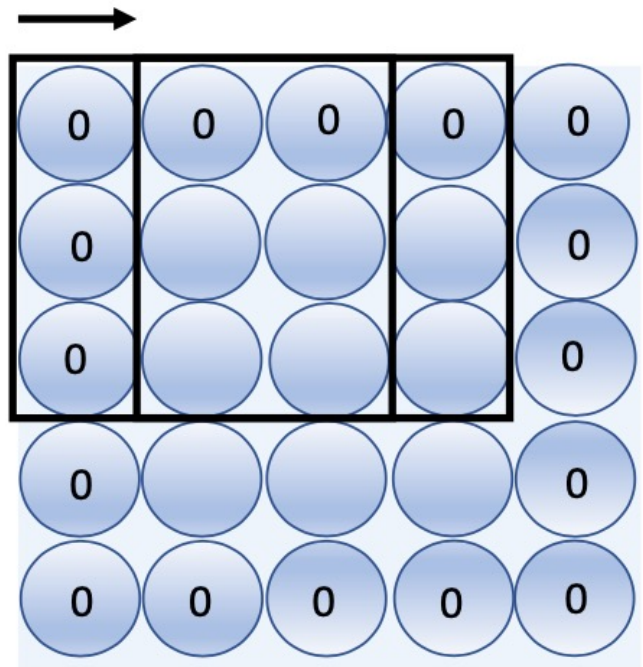


no padding
(3,3)

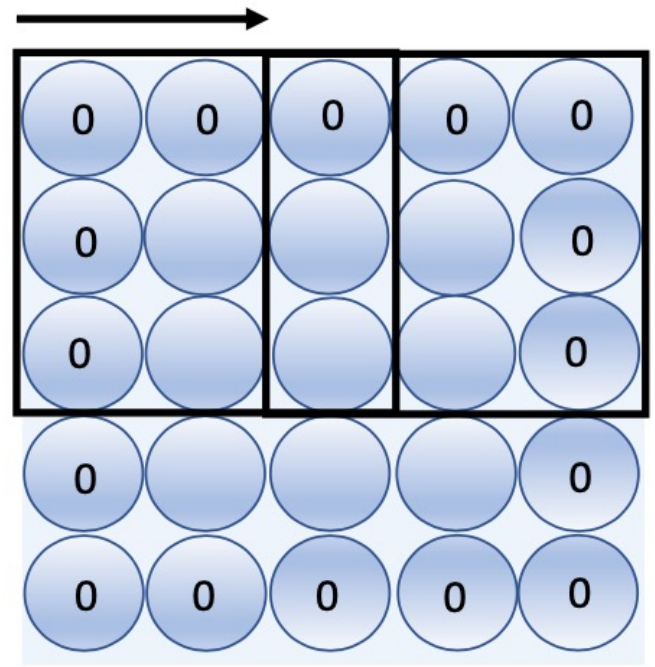
padding = 1
(5,5)

padding = 2
(7,7)

Stride

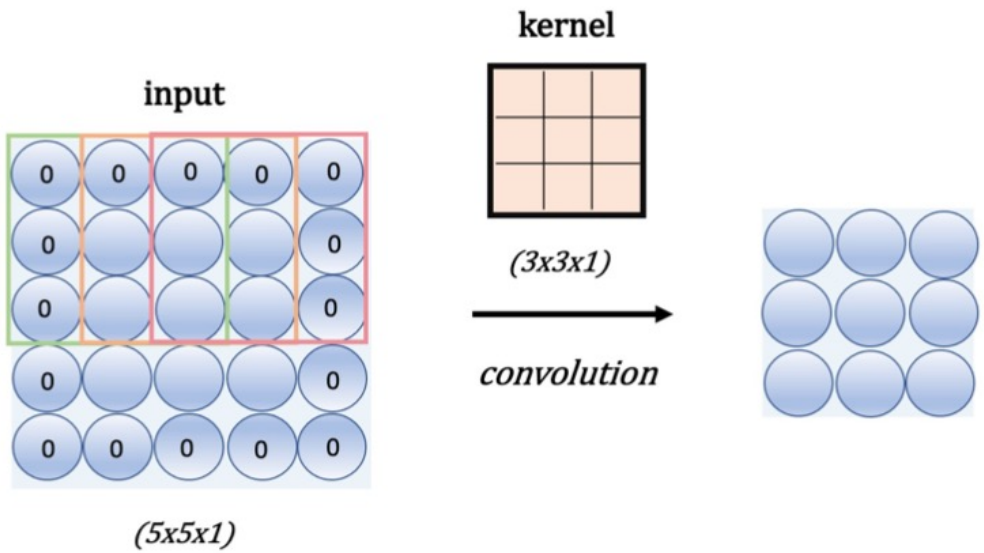


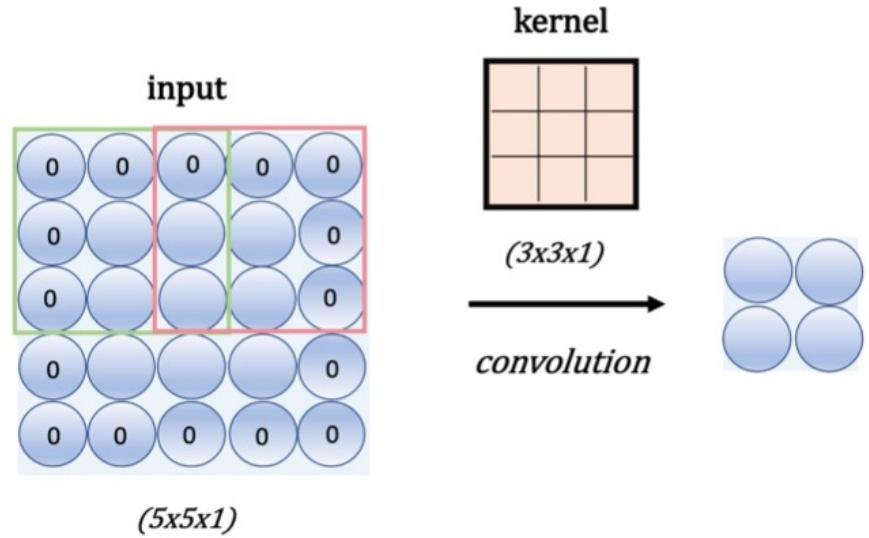
stride = 1



stride = 2

Padding & stride

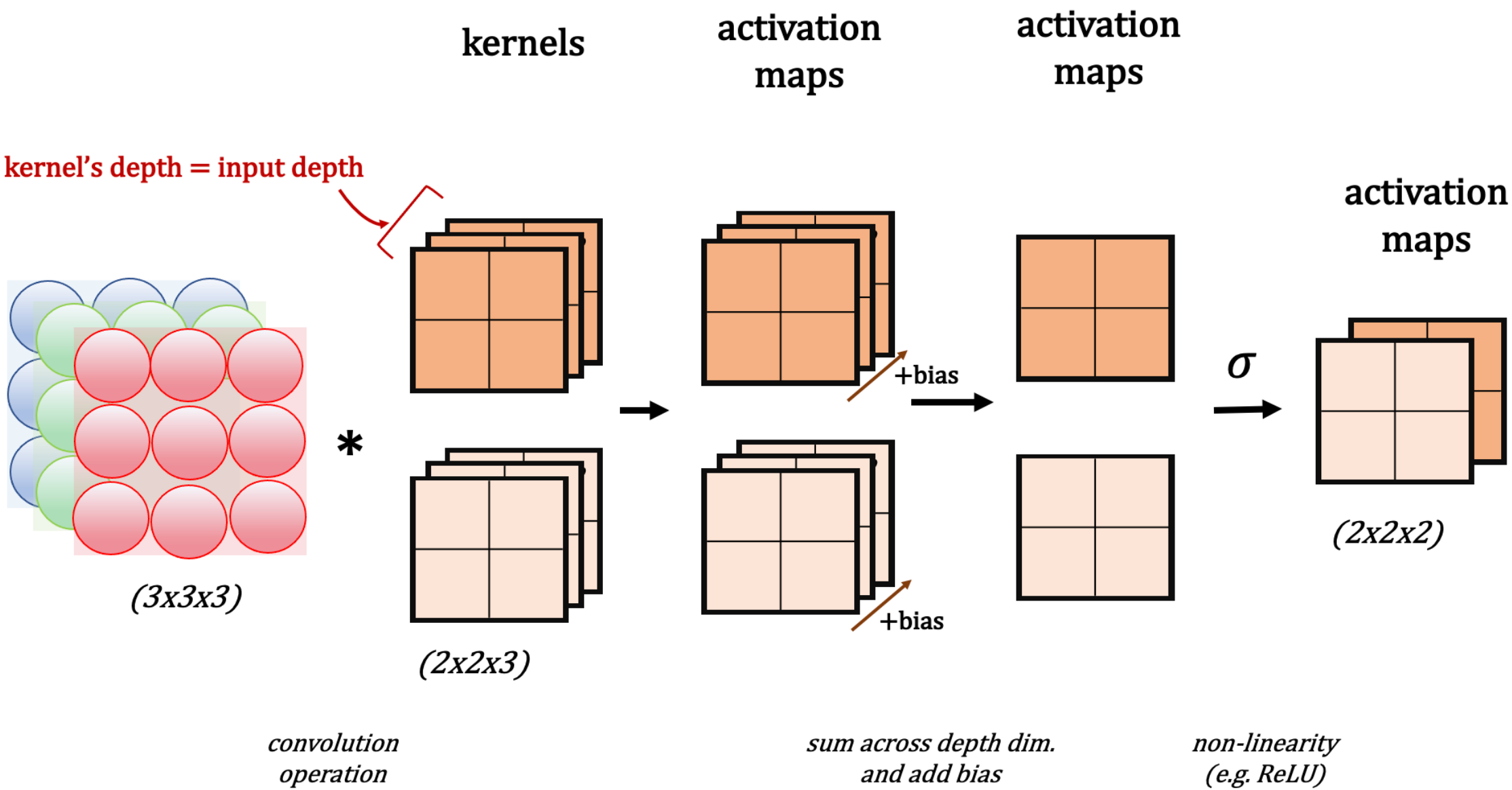


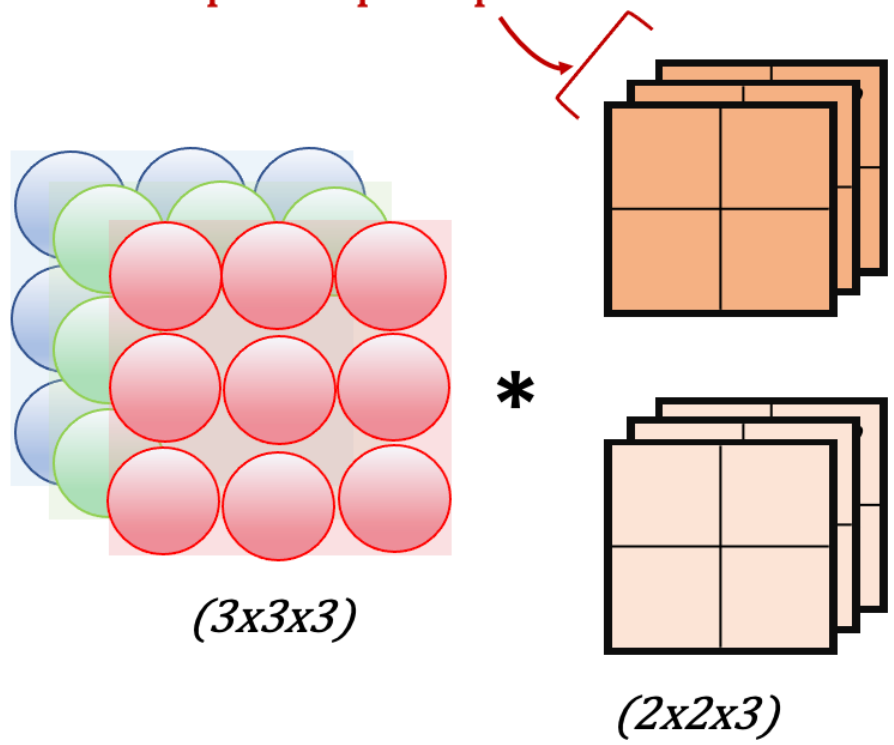


$$output\ size = \frac{input\ size + 2 * padding - kernel\ size}{stride} + 1$$

$$output\ size = \frac{3 + 2 * 1 - 3}{1} + 1 = 3$$

$$output\ size = \frac{3 + 2 * 1 - 3}{2} + 1 = 2$$


kernels
activation maps
activation maps
activation maps
kernel's depth = input depth
+bias
+bias
σ
(3x3x3)
(2x2x3)
(2x2x2)
convolution operation
sum across depth dim. and add bias
non-linearity (e.g. ReLU)



n.o. parameters:

- 2*(2*2*3 + 1)
- K_num * (K_h*K_w*K_ch + bias)

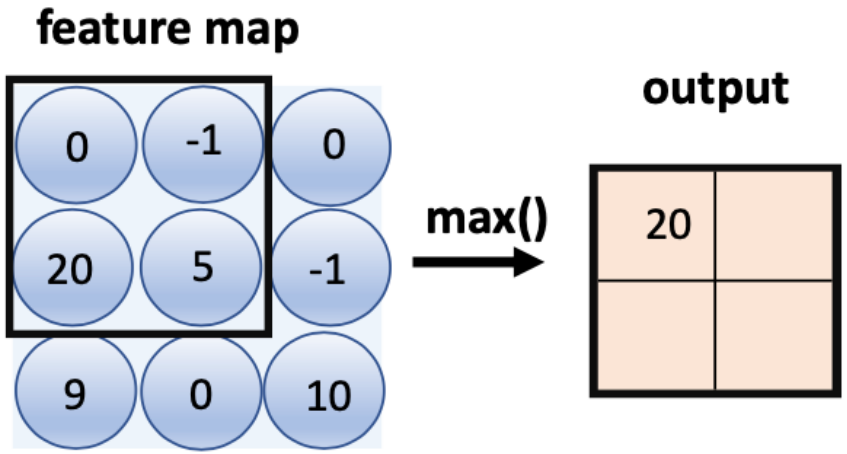
Memory requirements

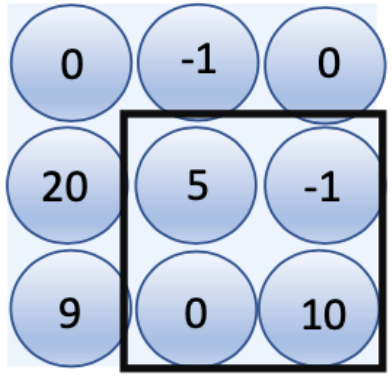
- input (150x100x3)
- 200 kernels (5x5x3)
- output (actiavtion maps) size (150x100)

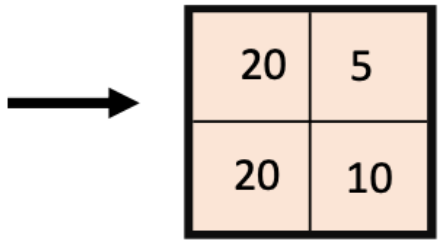
- n.o. of multiplications:
  200*(150*100)*(5*5*3) = 225*10^6

- have to store 200*(150*100) values, for 32-bit floats:
  200*(150*100)*32 = 96*10^6 bits = 12MB
  for one data point, one conv. layer

Max pooling layer







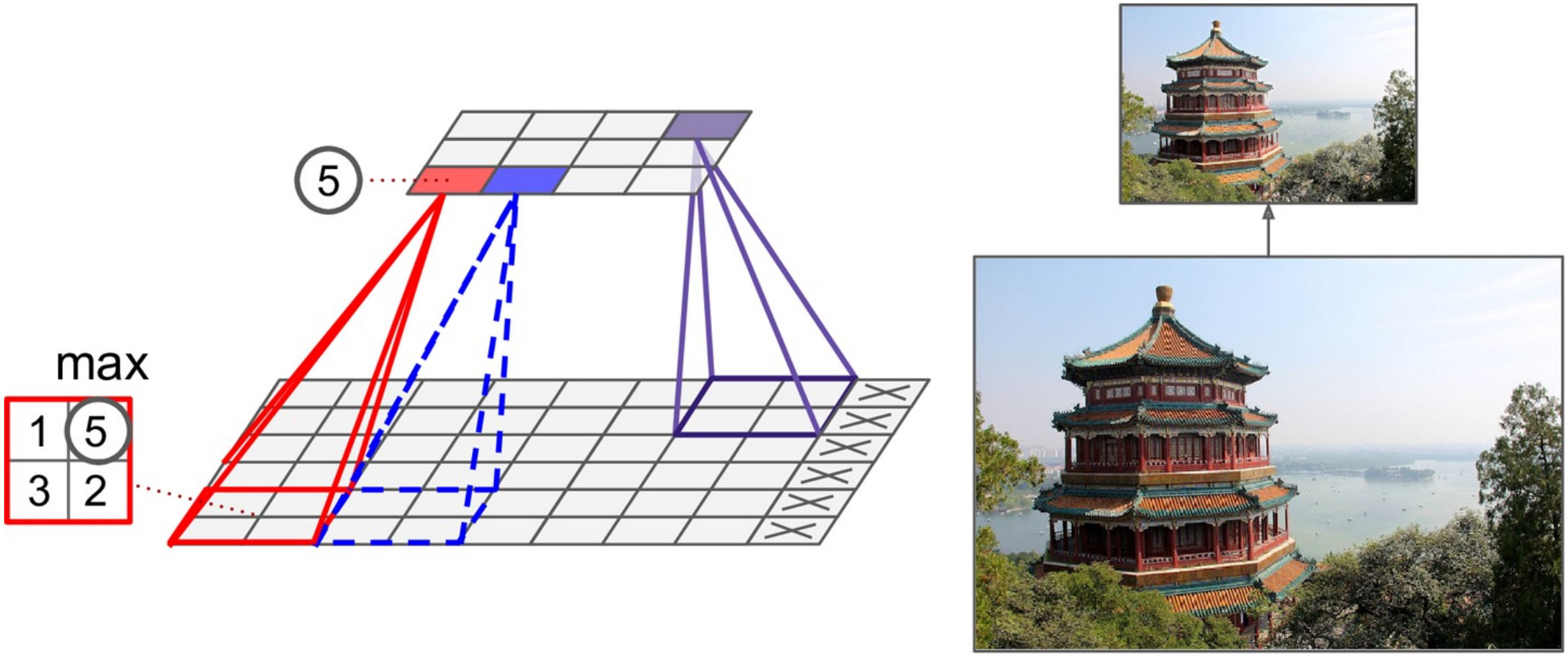


Figure 13-8. Max pooling layer (2 × 2 pooling kernel, stride 2, no padding)
"Hands-On Machine Learning …" A.Géron

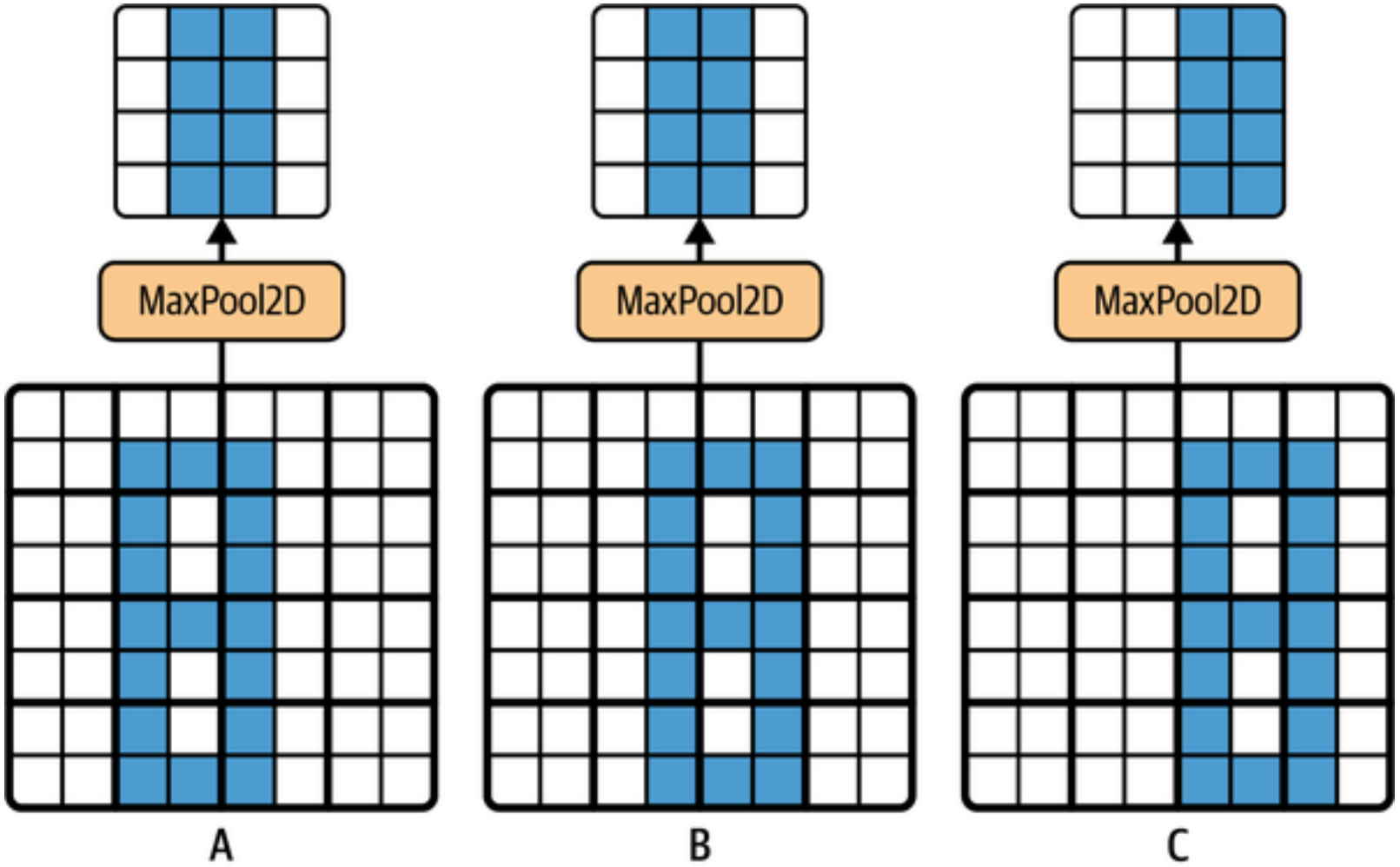
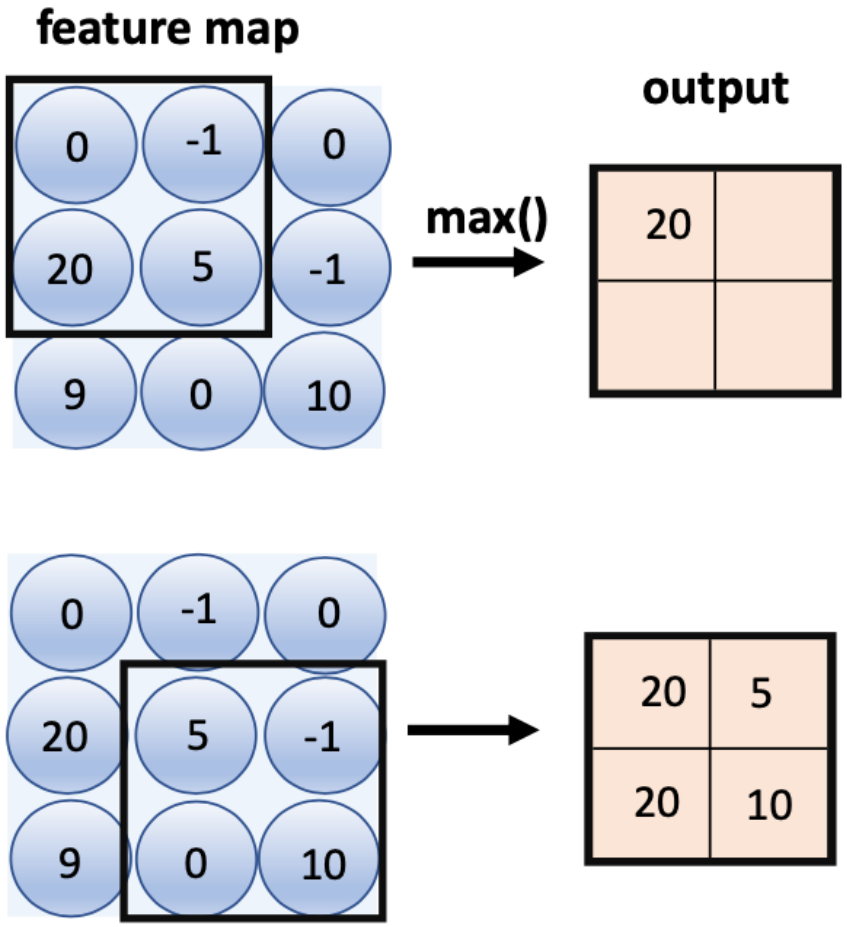


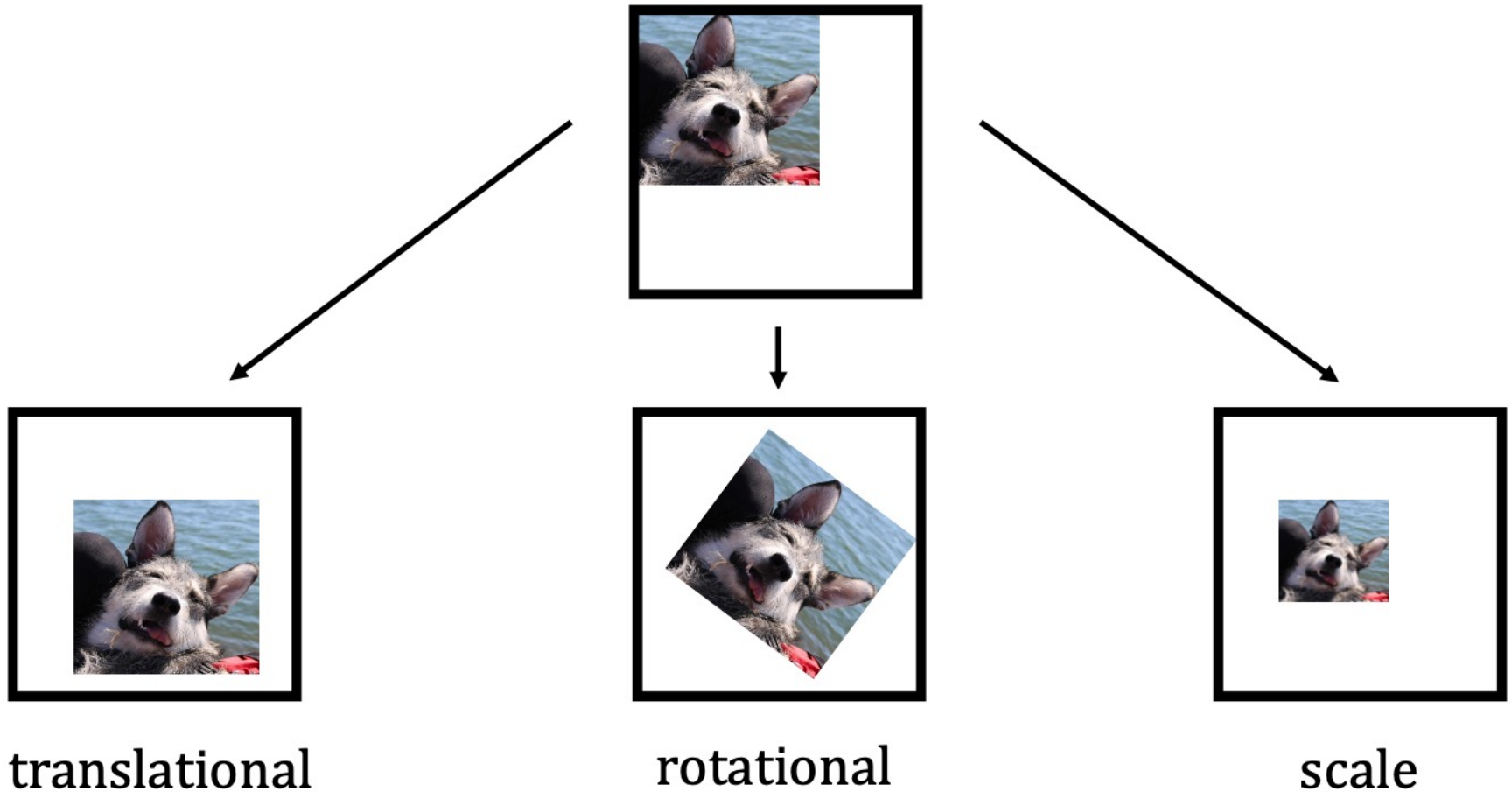
Figure 14-10. Invariance to small translations
“Hands-On Machine Learning …” A.Géron

Max pooling layer



- reduces the number of parameters
- provides (some degree of) invariance

Conv + Max pooling layers → invariance



Common CNN architecture

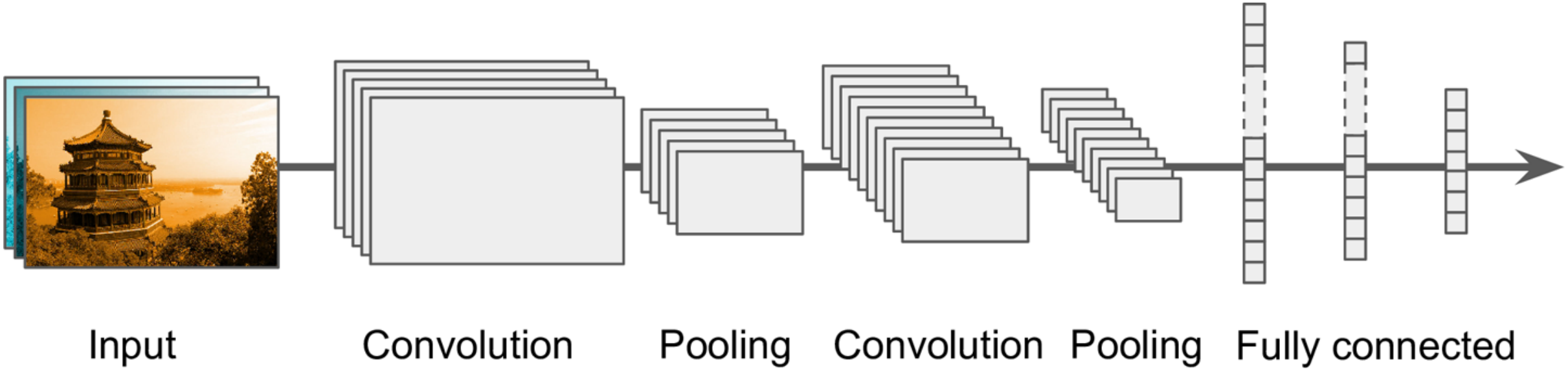


Figure 13-9. Typical CNN architecture
"Hands-On Machine Learning ..." A.Géron

INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

Usually N<=3, K<3

For example:

```
1) INPUT -> CONV -> RELU -> FC
   [N=1, M=1, no POOL, K=0]

2) INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC
   [N=1, M=2, K=1]

3) INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC ->
RELU]*2 -> FC
   [N=2, M=3, K=2]
```

CNN hyperparams – some tips

- larger kernels in the 1st layer (for large img)
- going deeper to the CNN → input getting smaller, n.o. kernels is increasing
- use stack of smaller kernels than one large kernel (allows to express more powerful features)
- keep input size same (stride=1&padding), size reduction only at pooling layers

https://cs231n.github.io/convolutional-networks/#convert

Conv layer in Keras

```
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid',
    data_format=None, dilation_rate=(1, 1), groups=1, activation=None,
    use_bias=True, kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None,
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,
    bias_constraint=None, **kwargs
)
```

```
tf.keras.layers.Conv2D(filters=16,
                       kernel_size=3,
                       padding="same",
                       activation="relu",
                       input_shape=[28,28,1])
```

Conv layer in Keras

Padding: one of "valid" or "same" (case-insensitive).

- "valid" means no padding.
- "same" results in padding with zeros evenly to the left/right or up/down of the input:

    - When padding="same" and strides=1, the output has the same size as the input.

    - When padding="same" and strides is not =1, the output has size in_size/stride rounded up.

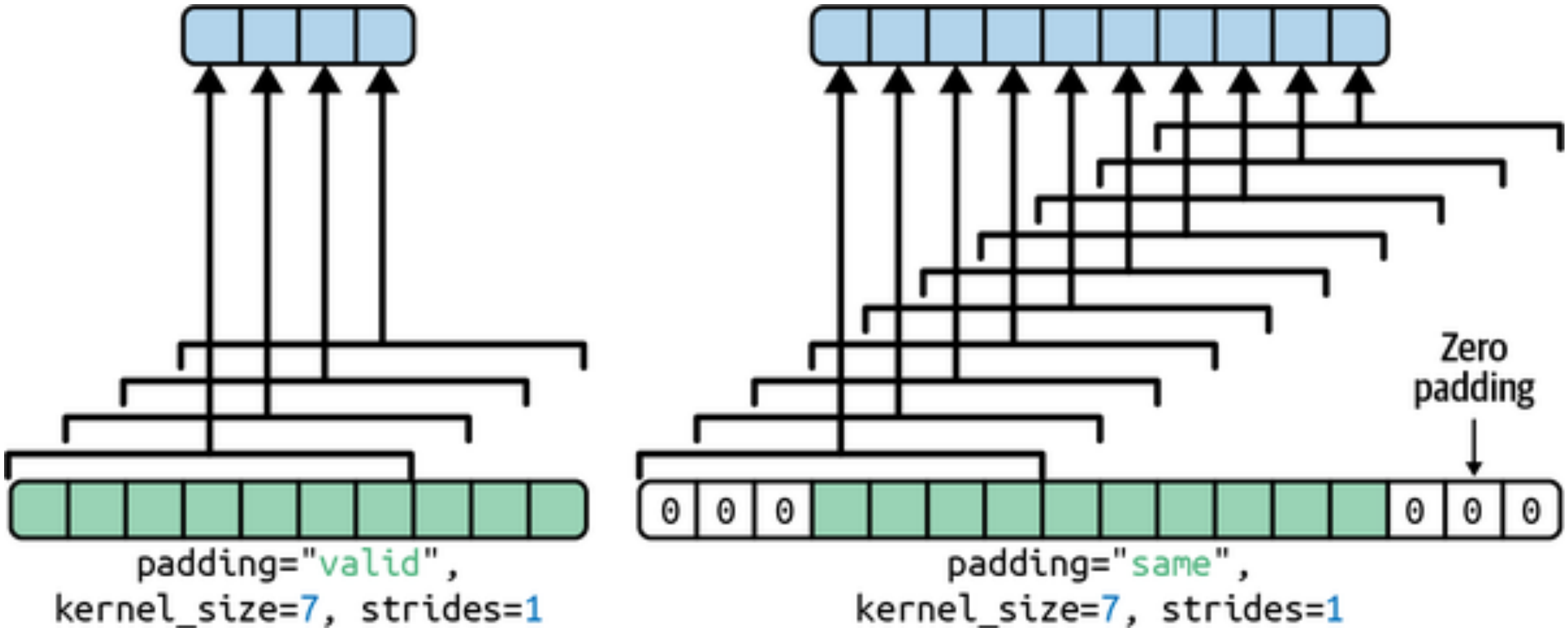


Figure 14-7. The two padding options, when strides=1
“Hands-On Machine Learning …” A.Géron

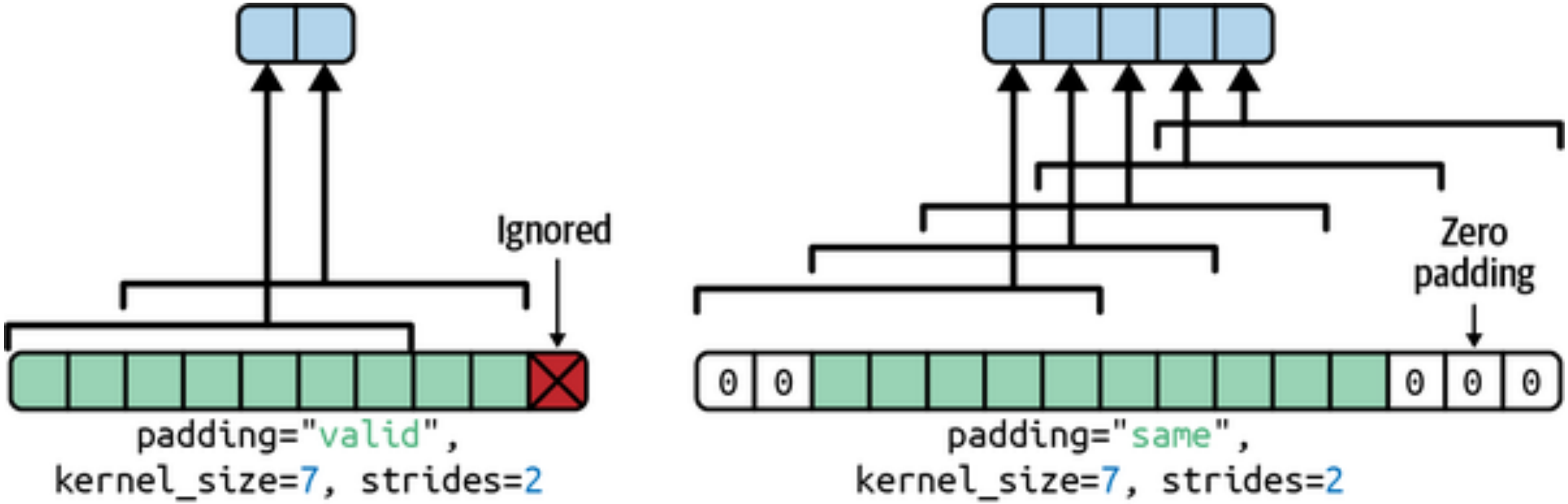


Figure 14-8. With strides greater than 1, the output is much smaller even when using "same" padding (and "valid" padding may ignore some inputs)
“Hands-On Machine Learning …” A.Géron

Max pooling layer in Keras

```
tf.keras.layers.MaxPool2D(
    pool_size=(2, 2), strides=None, padding='valid', data_format=None,
    **kwargs
)
```

```
tf.keras.layers.MaxPool2D(pool_size=2)
```

Conv Net

```
model = tf.keras.models.Sequential([
    # conv block
    tf.keras.layers.Conv2D(16, 3, padding="same", activation="relu",
input_shape=[28,28,1])
    tf.keras.layers.Conv2D(16, 3, padding="same", activation="relu"),
    tf.keras.layers.MaxPool2D(pool_size=2),
    # Classification head
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64,"relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

CNN architectures – LeNet-5

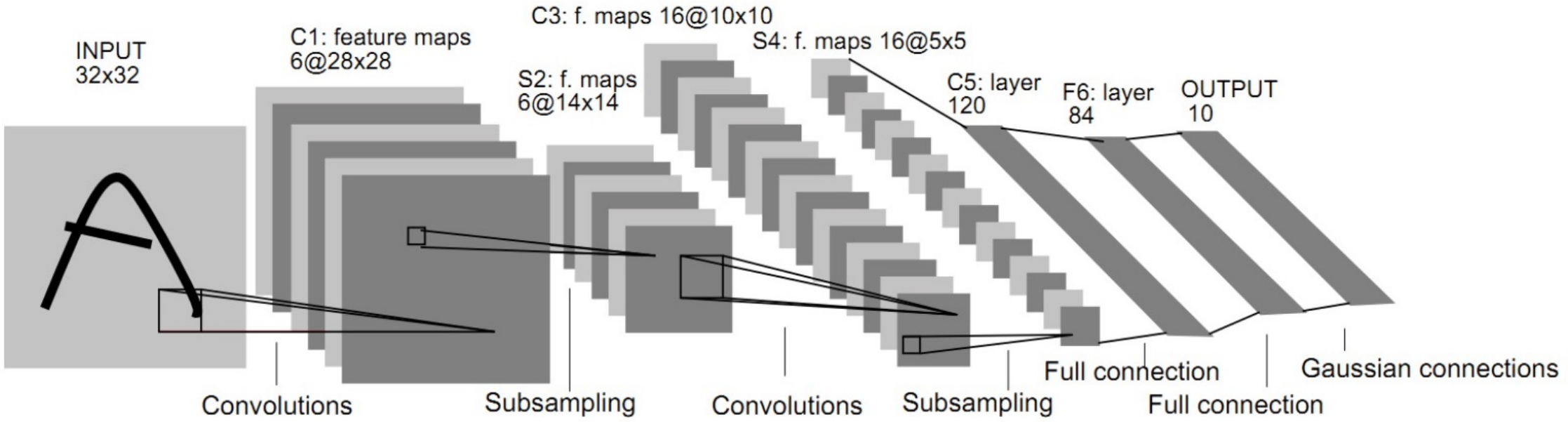


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

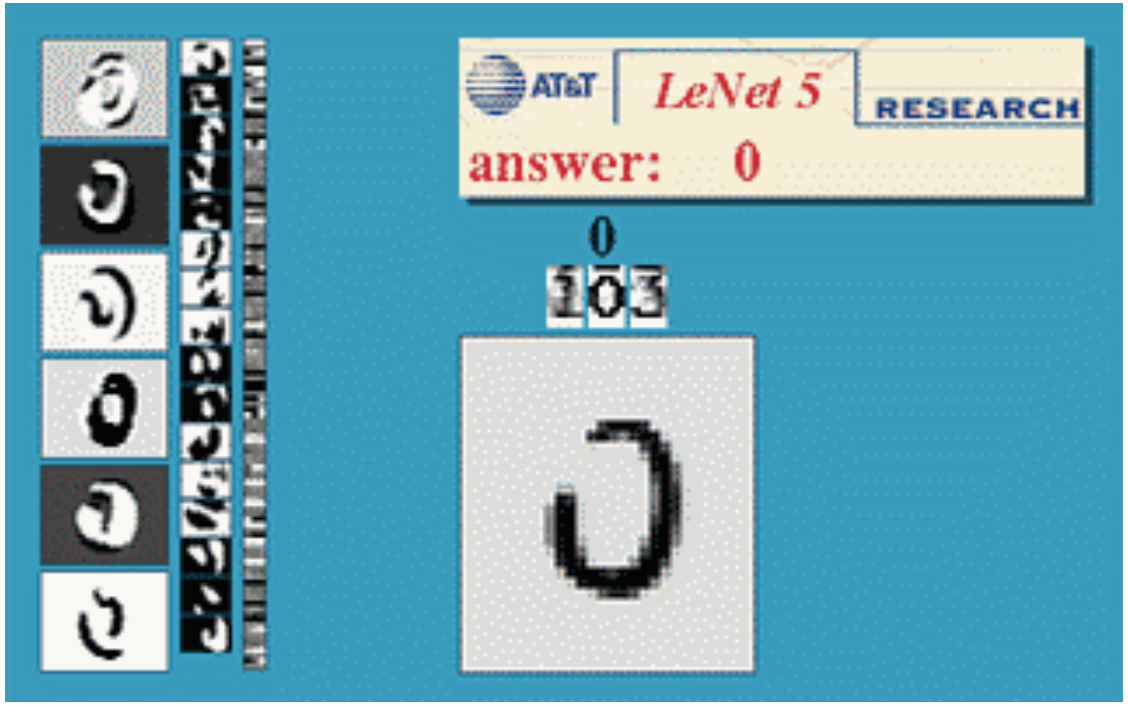
[CONV →POOLING → CONV →POOLING → FC →FC]

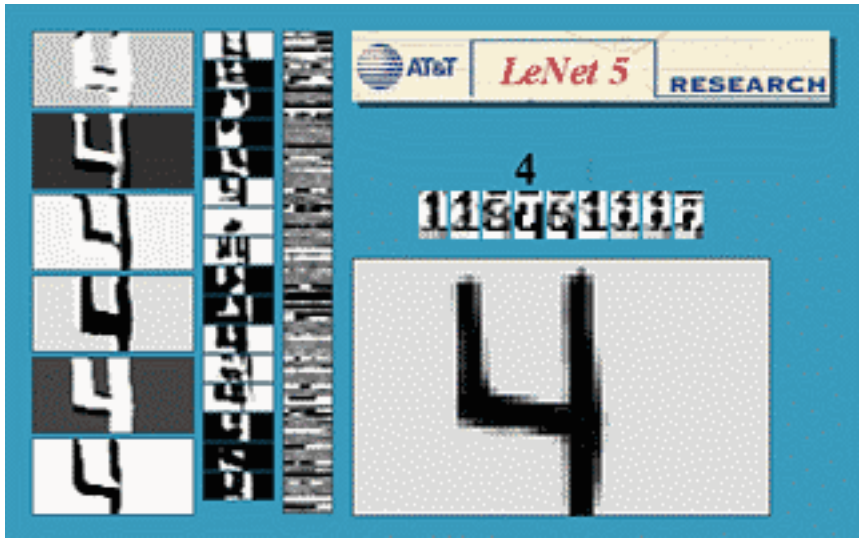
[LeCun, Y. et al(1998). Gradient-based learning applied to document recognition.]

CNN architectures – LeNet-5

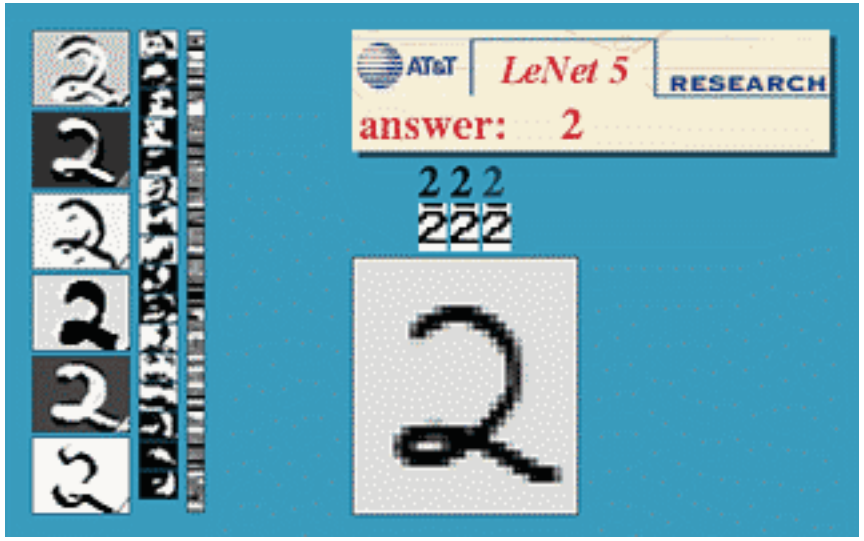
Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

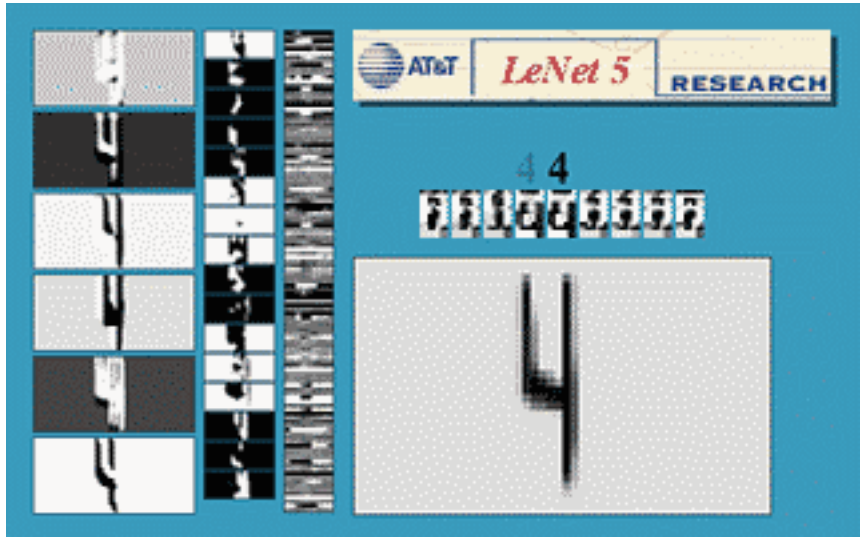
CNN architectures – LeNet-5



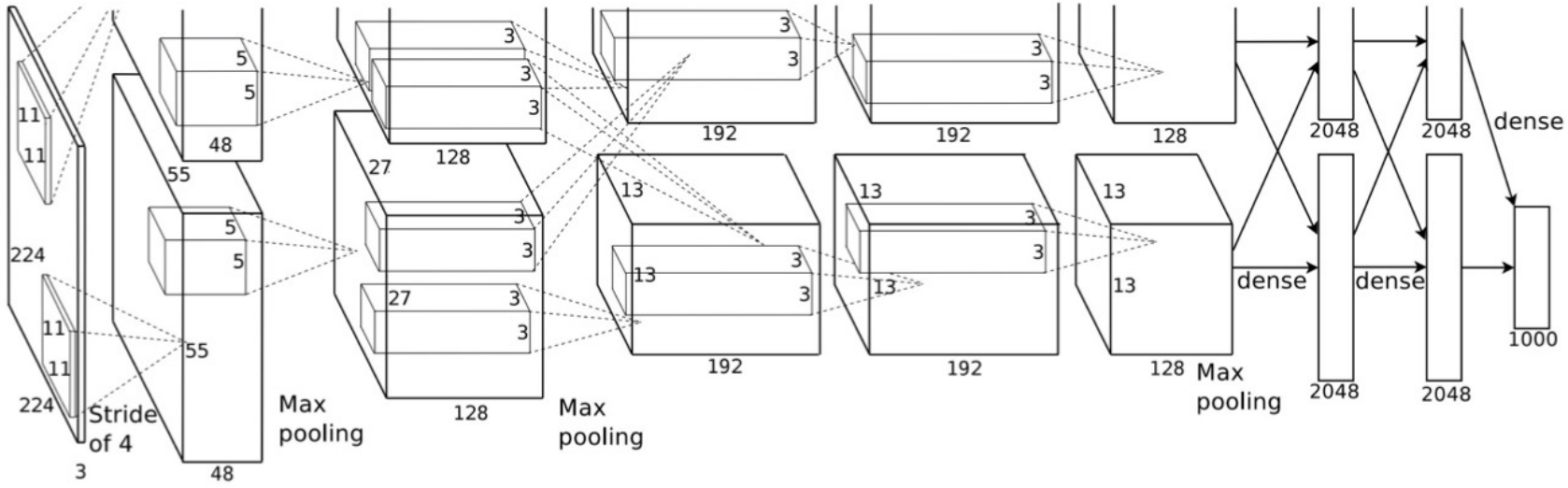

AT&T
LeNet 5
RESEARCH
4


AT&T
LeNet 5
RESEARCH


AT&T
LeNet 5
RESEARCH
answer: 2
2 2 2


AT&T
LeNet 5
RESEARCH
4 4

CNN architectures – AlexNet



[CONV1 → POOLING1 → NORM1 →
CONV2 → POOLING2 → NORM2 →
CONV3 → CONV4 → CONV5 → POOLING3
→ FC1 →FC2 →FC3]

[Krizhevsky, A. et al (**2012**). Imagenet classification with deep convolutional neural networks.]

CNN architectures – AlexNet

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	valid	–
C7	Convolution	256	13 × 13	3 × 3	1	same	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	same	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	same	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	valid	–
C3	Convolution	256	27 × 27	5 × 5	1	same	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	valid	–
C1	Convolution	96	55 × 55	11 × 11	4	valid	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

CNN architectures – GoogLeNet

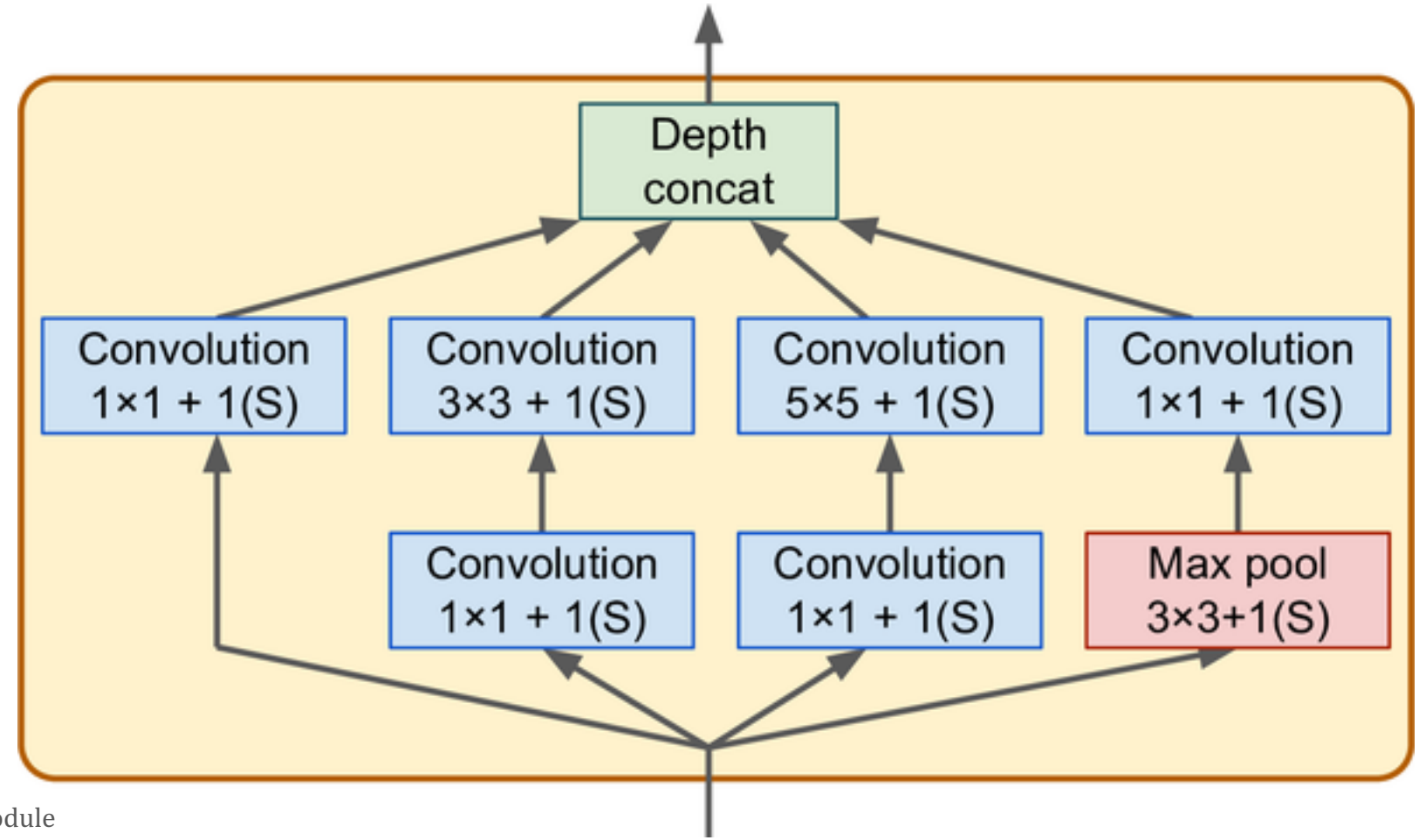


Figure 14-14. Inception module
"Hands-On Machine Learning …" A.Géron

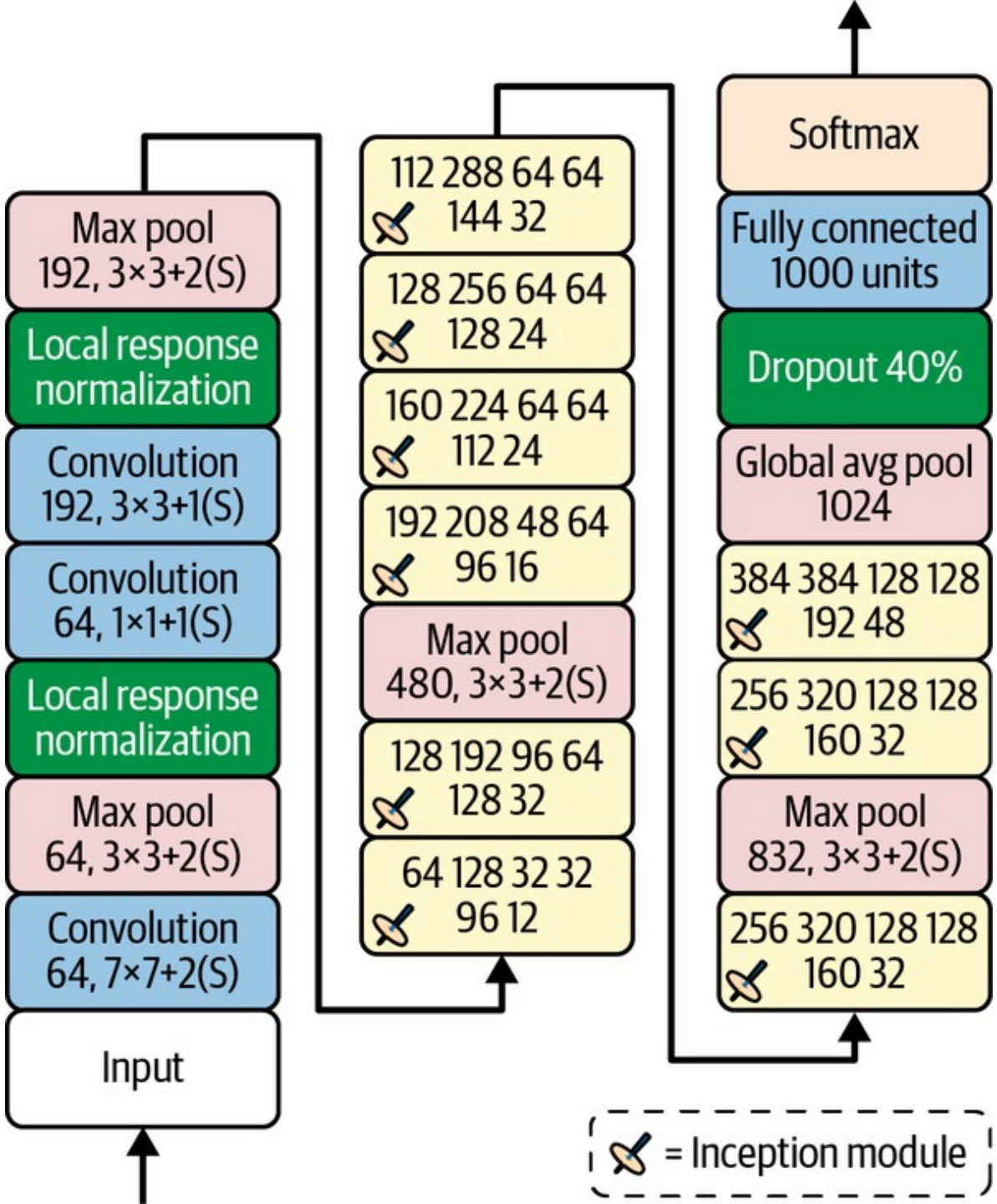


Figure 14-15. GoogLeNet architecture
"Hands-On Machine Learning ..." A.Géron

CNN architectures – ResNet

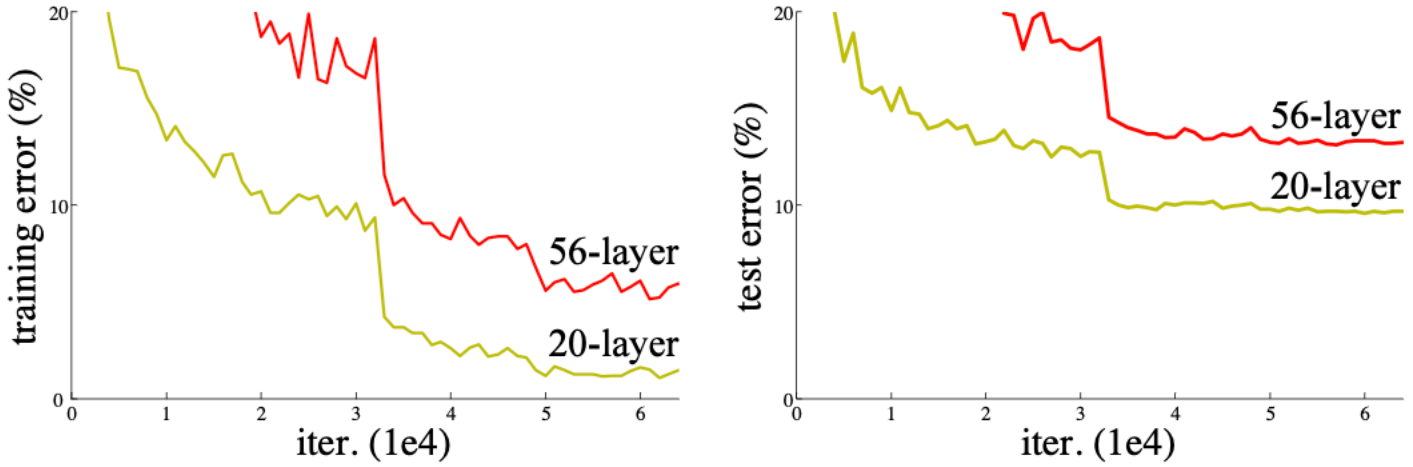


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

"Deep Residual Learning for Image Recognition" K. He

CNN architectures – ResNet

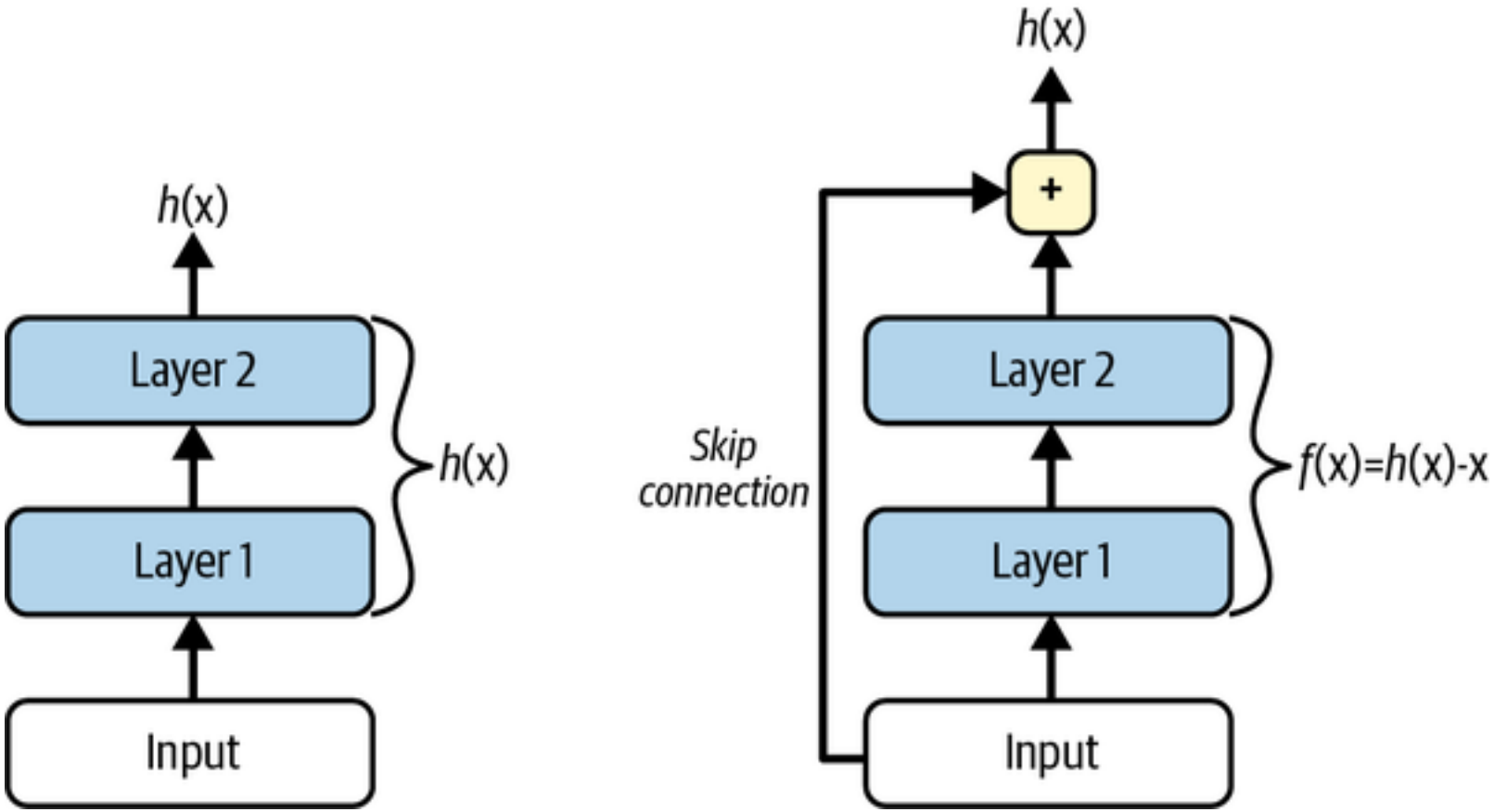


Figure 14-16. Residual learning
"Hands-On Machine Learning …" A.Géron

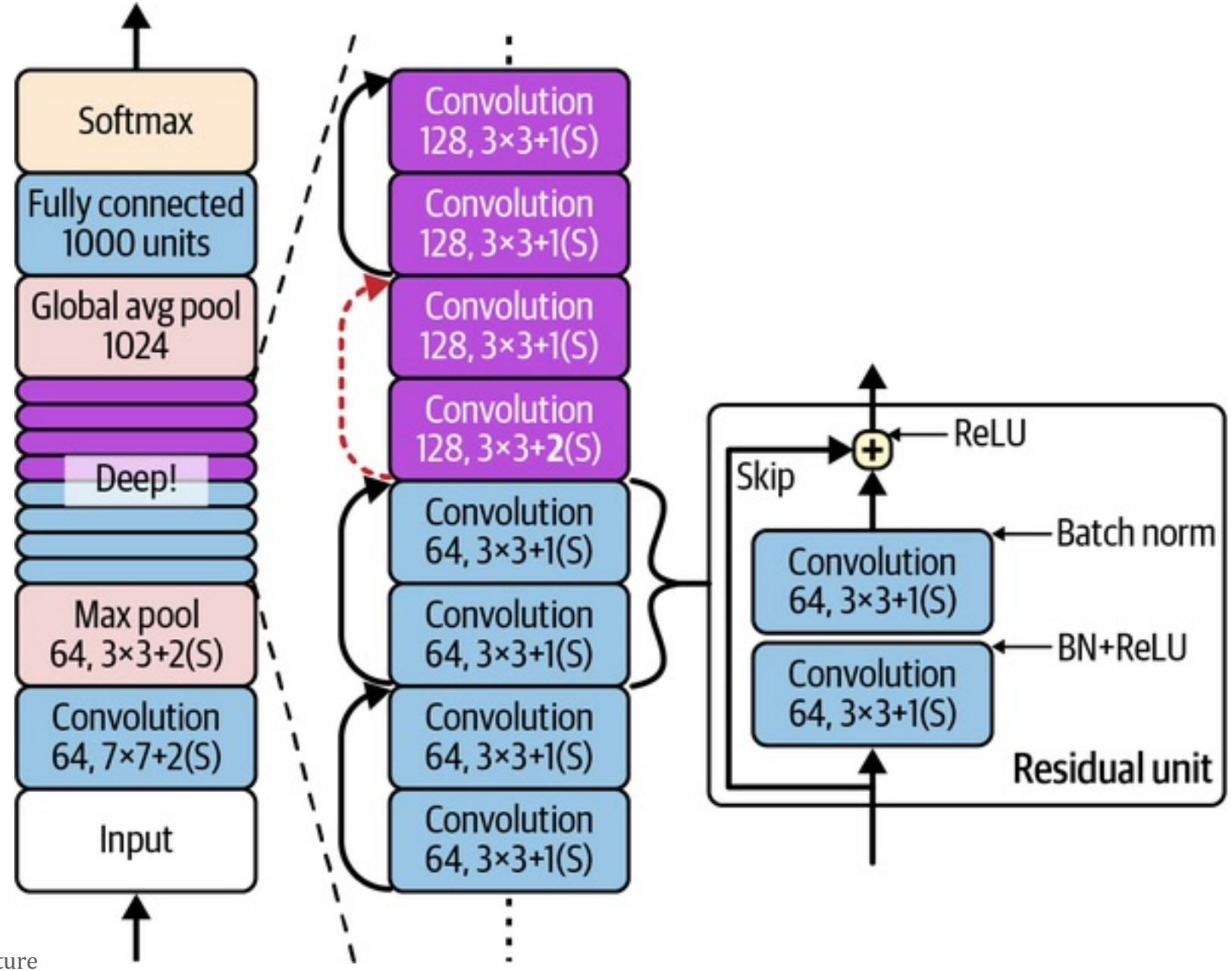
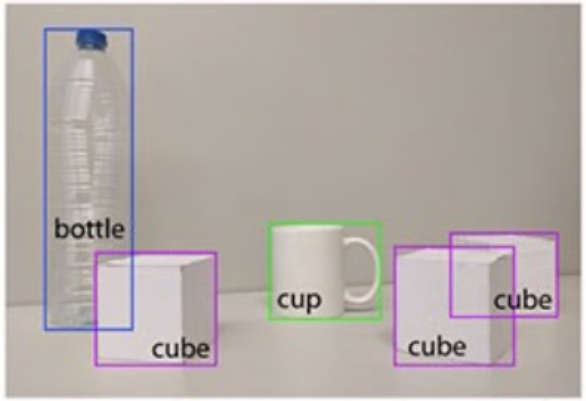


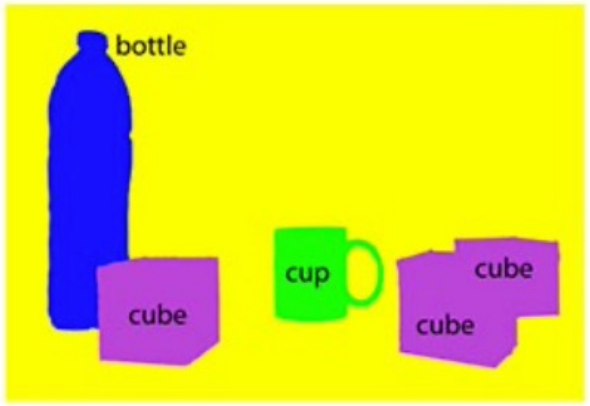
Figure 14-18. ResNet architecture
"Hands-On Machine Learning …" A.Géron

CNN tasks

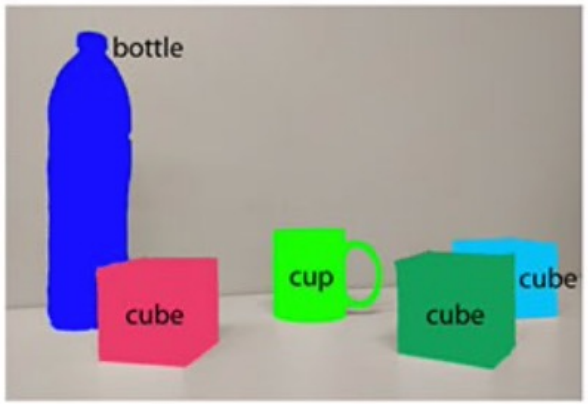
(a) Image classification



(b) Object localization



(c) Semantic segmentation



(d) Instance segmentation