Neural Networks Layers

Layers

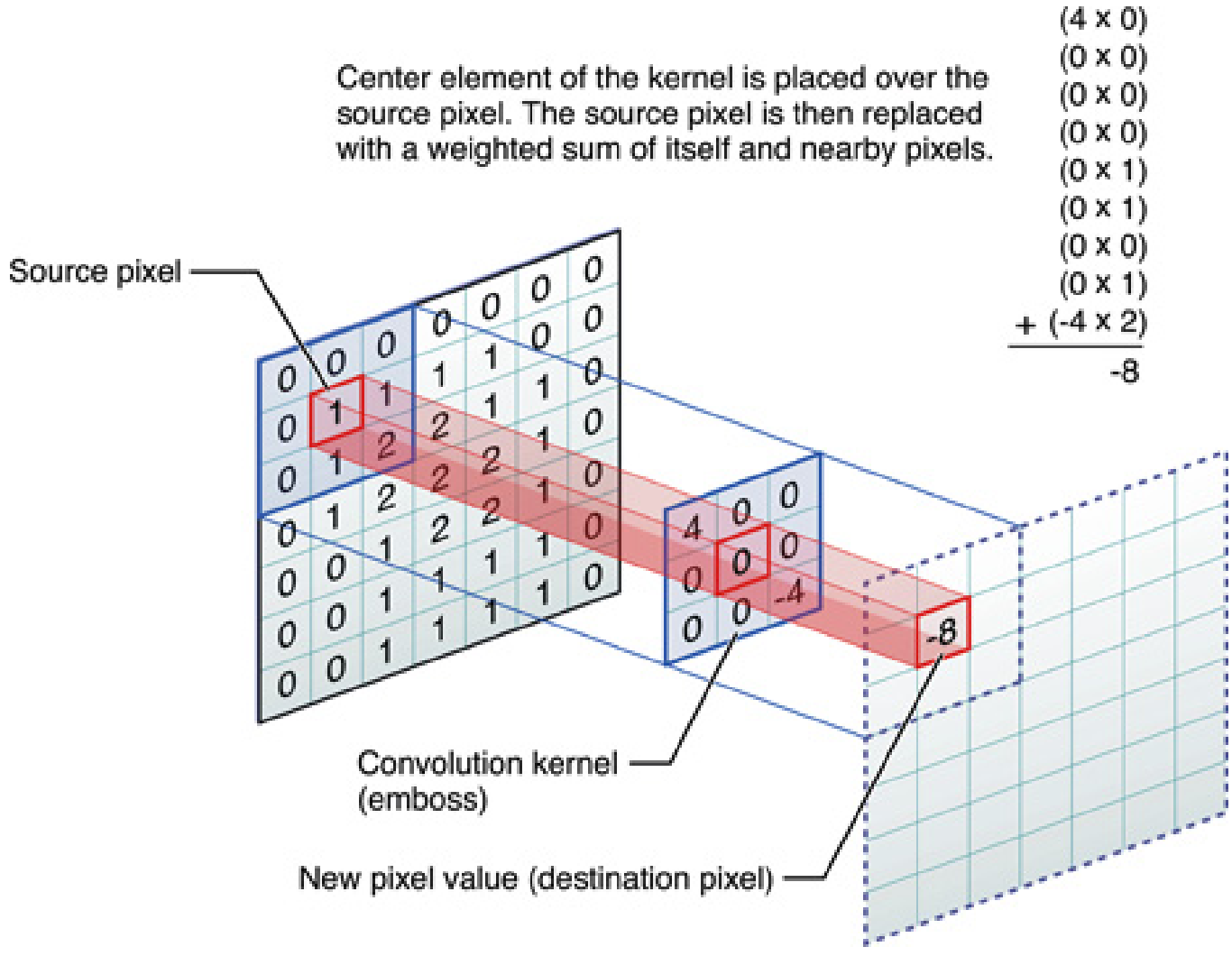
Each Layer takes in input a Tensor and returns a new Tensor, after performing some algebraic manipulation.

There are many different kind of layers, serving different purposes:

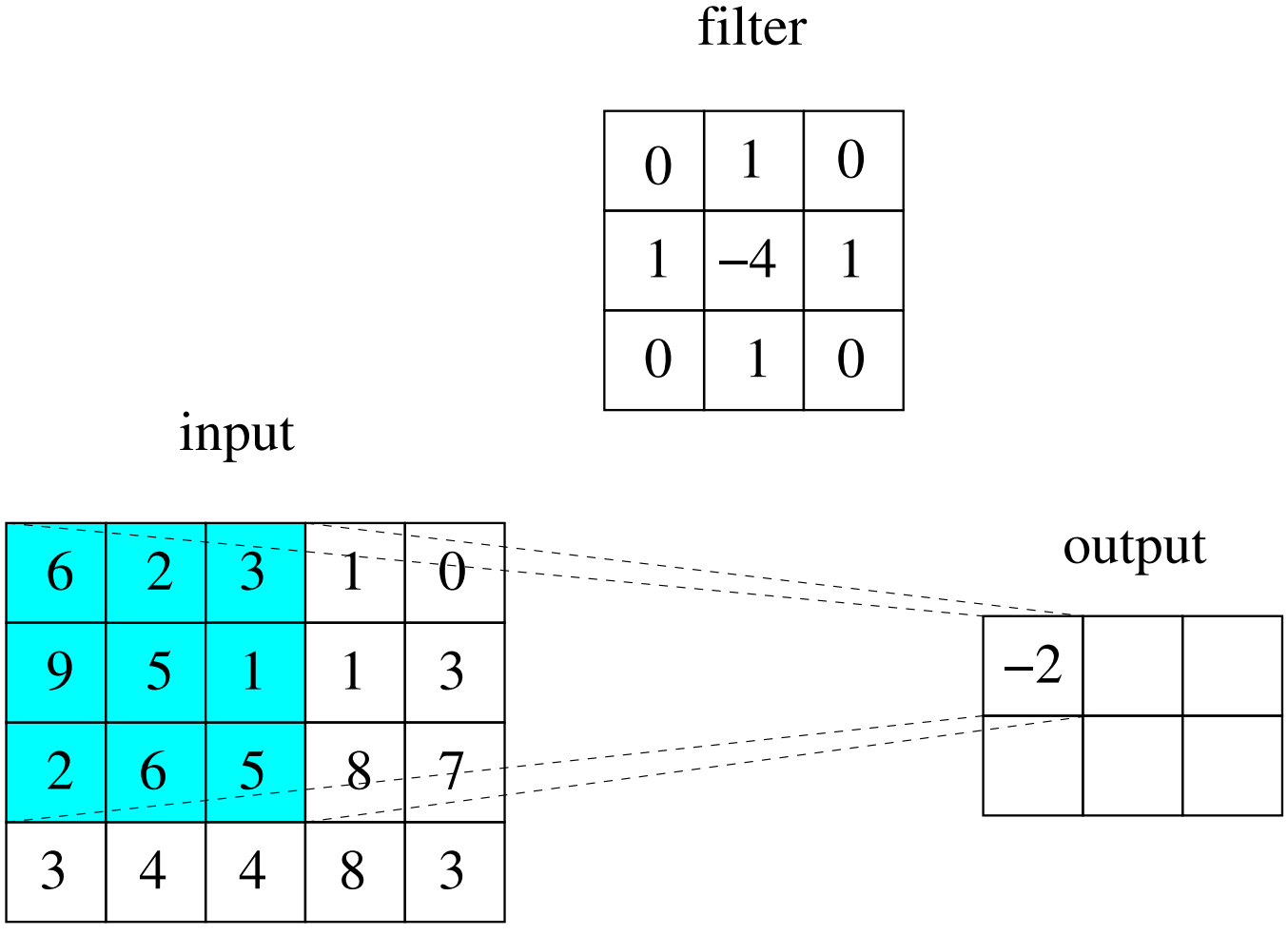
- **Processing layers**: dense (linear), convolutional, attention, ...
- **Shaping layers**: Upsampling, Dowsampling, Pooling, Reshape, ...
- **Structural layers**: concatenation (shortcuts), sum (residuality), ...
- **Normalization and regularization layers**: BatchNormalization, Dropout, ...

Convolutional Layers

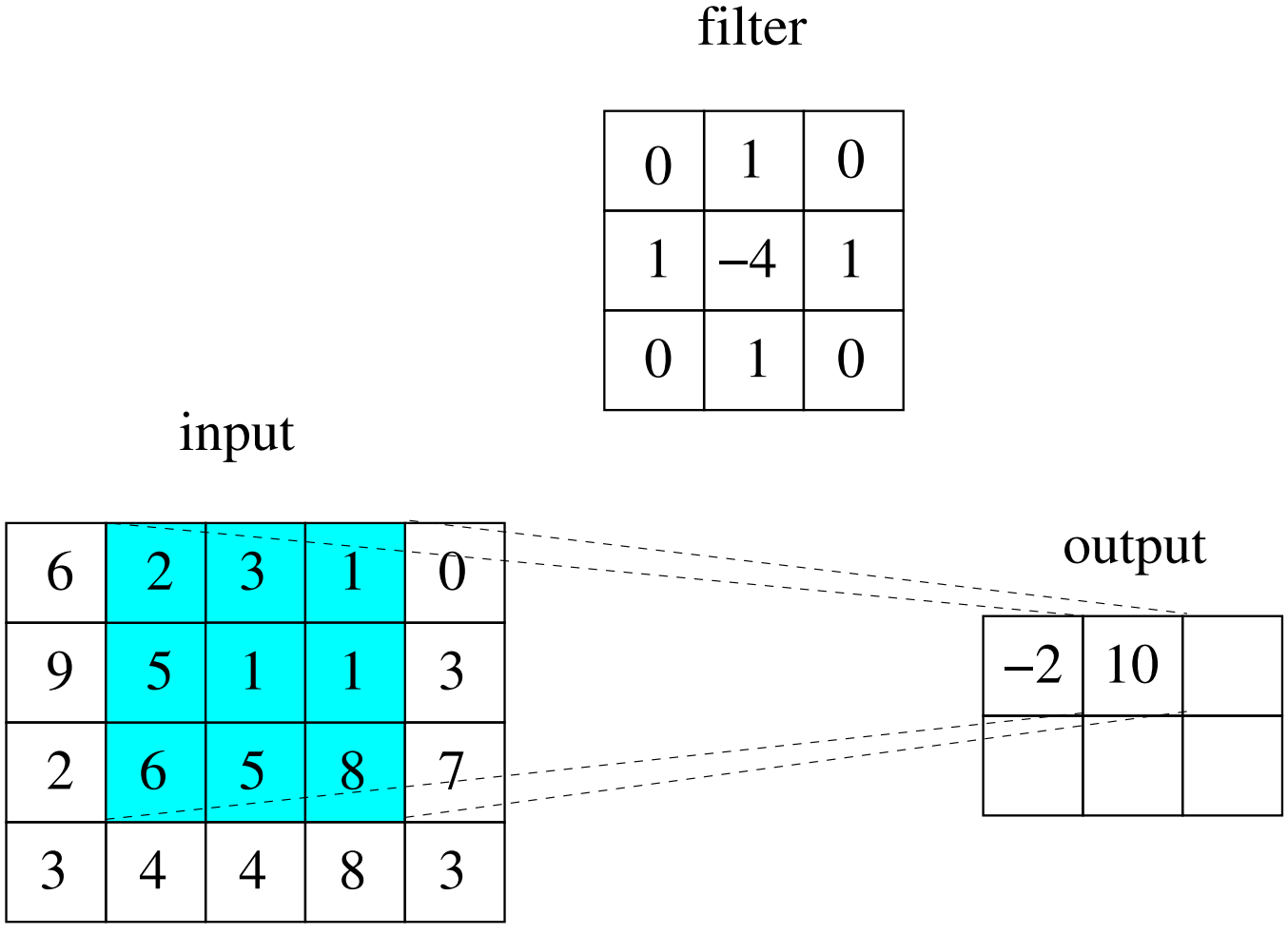
Filters and convolutions



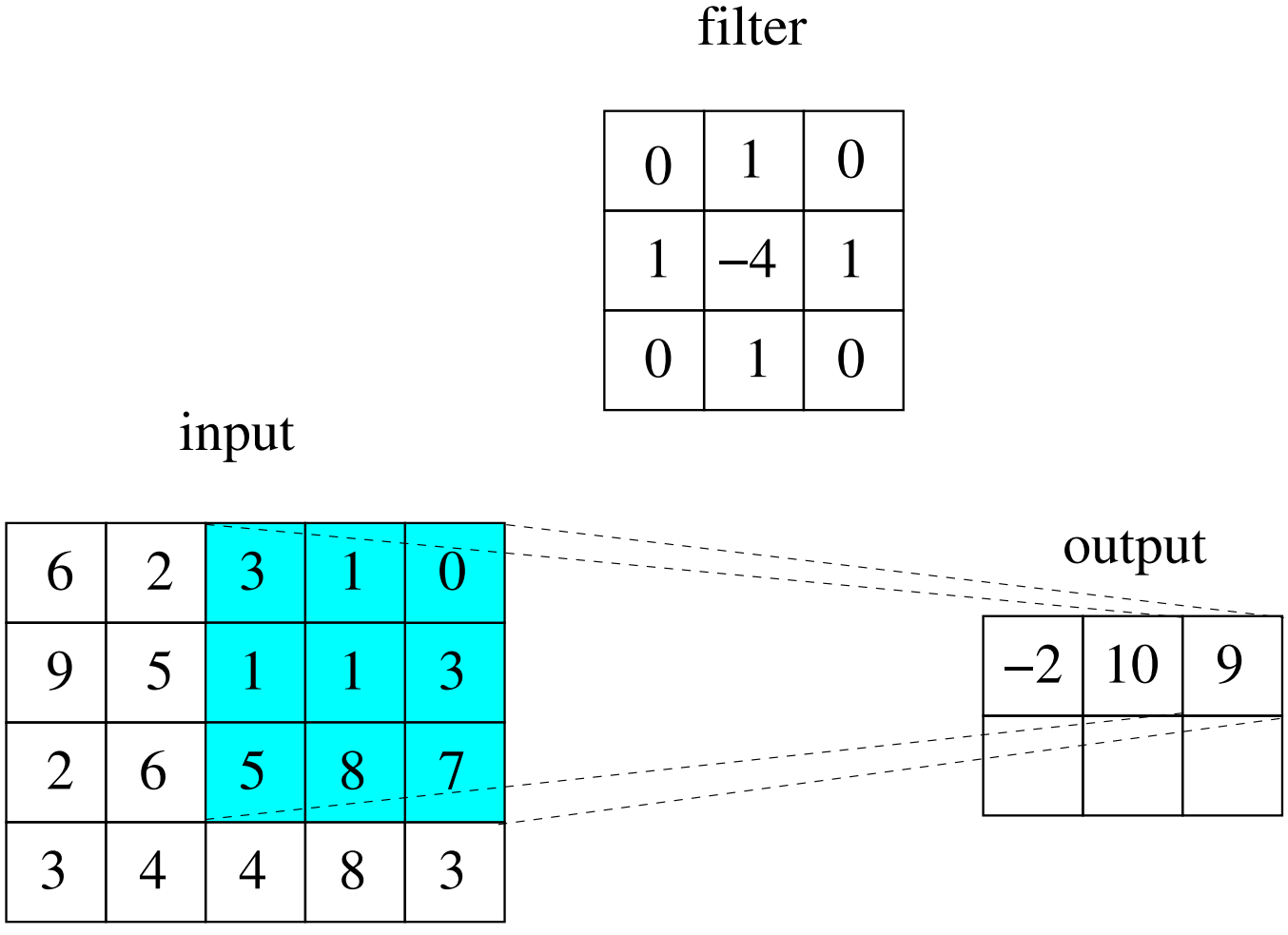
Filters and convolutions



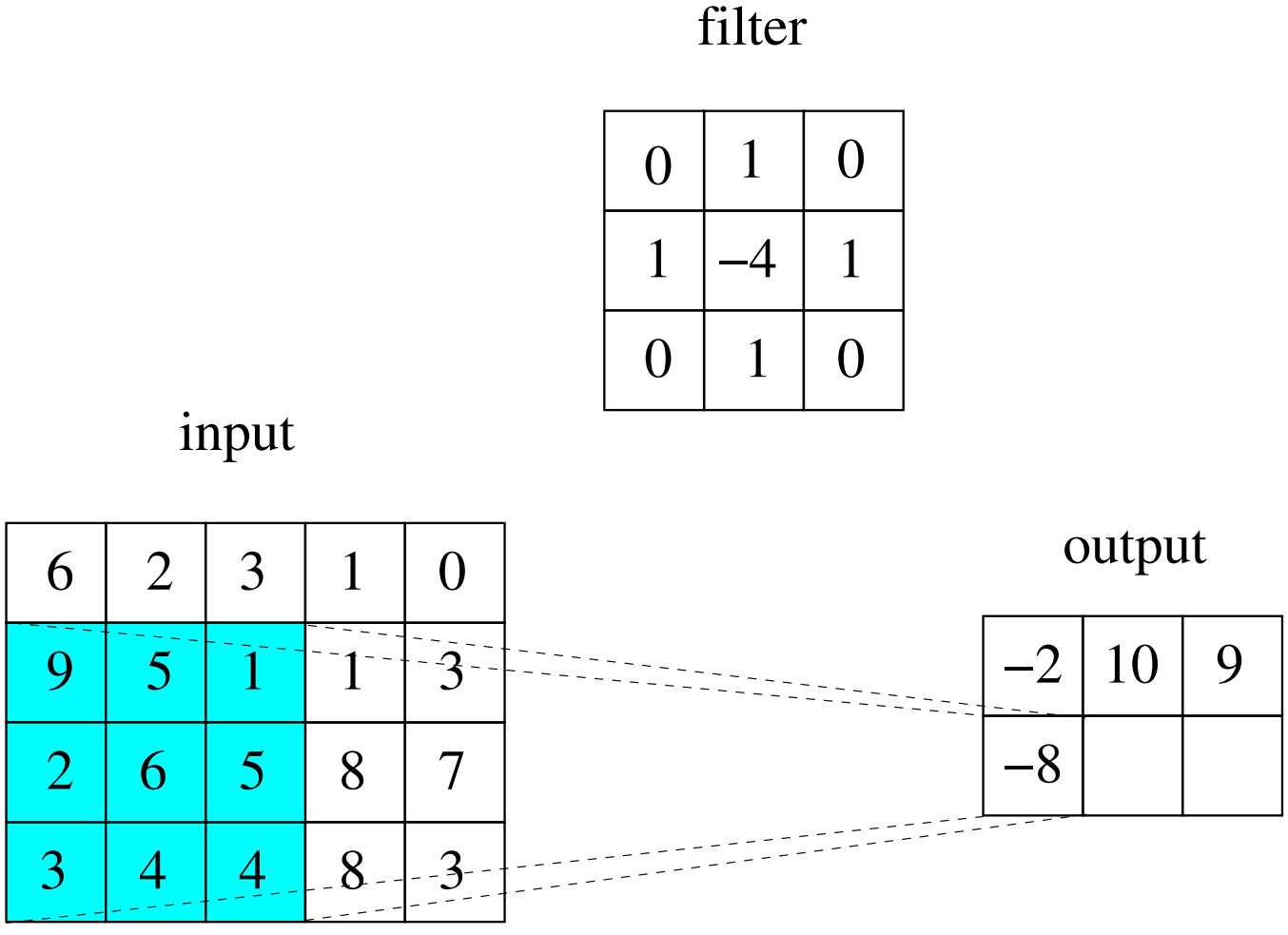
Filters and convolutions



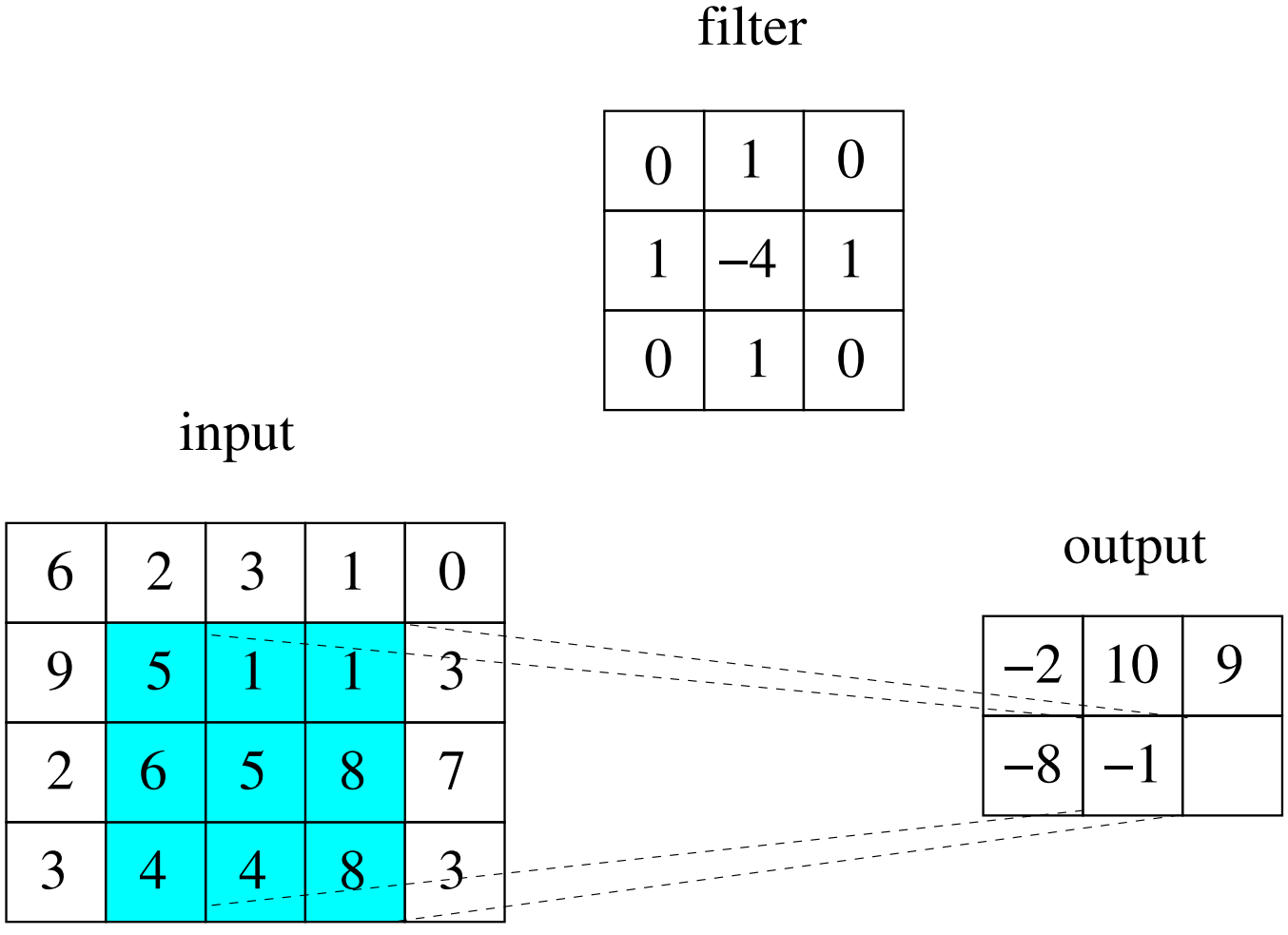
Filters and convolutions



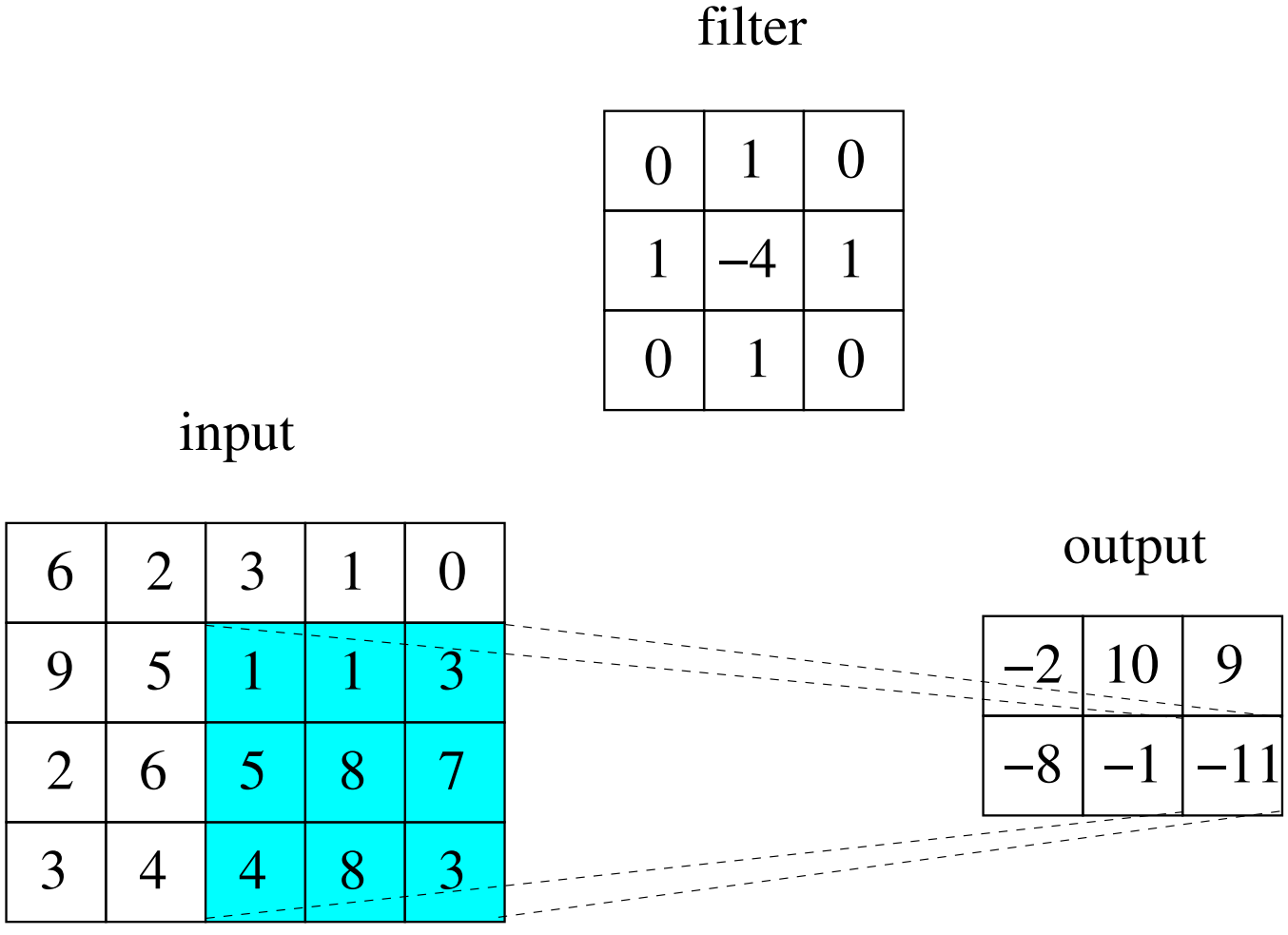
Filters and convolutions



Filters and convolutions



Filters and convolutions



Intuition

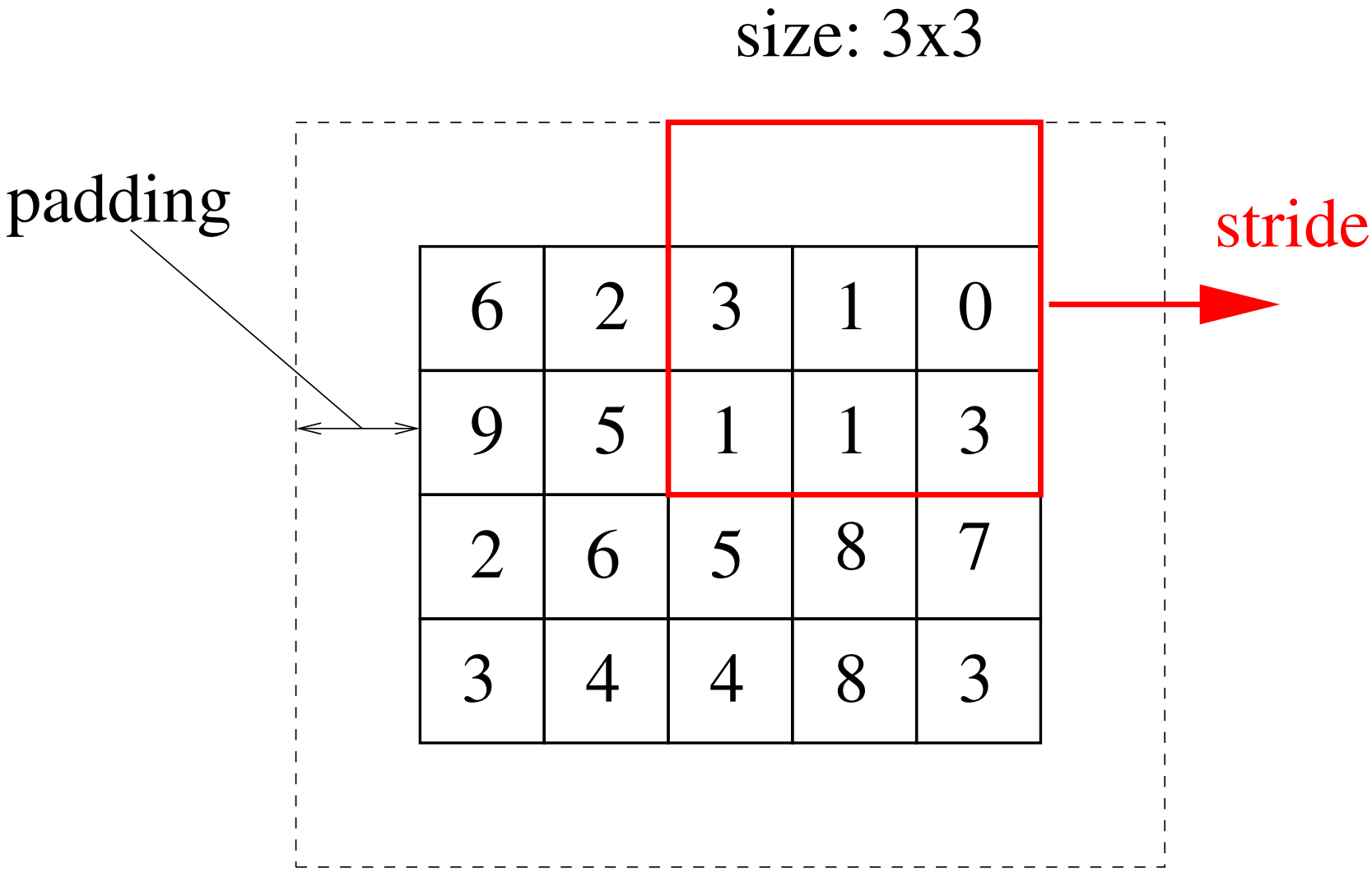
- the kernel define a pattern that we are interested to find in the input image
- the convolution consists in passing the pattern over the input as a sliding window, looking for positions matching the pattern
- the result is a feature map expressing the positions of the input with a higher similarity with the pattern

Parameters for convolutional layers

A convolutional layer is defined by the following parameters

- **kernel size**: the dimension of the linear filter.
- **stride**: movement of the linear filter. With a low stride (e.g. unitary) receptive fields largely overlap. With a higher stride, we have less overlap and the dimension of the output get smaller (lower sampling rate).
- **padding** Artificial enlargement of the input to allow the application of filters on borders.
- **depth**: number of different kernels that we wish to syntesize. Each kernel will produce a different feature map with a same spatial dimension.

Layers configuration params



Dimension of the output

The spatial dimension of each output feature map depends form the spatial dimension of the input, the padding, and the stride. Along each axes the dimension of the output is given by the formula

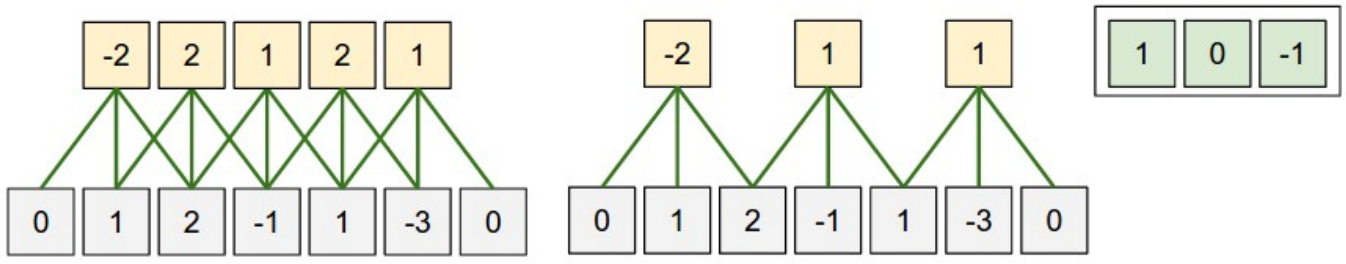
$$\frac{W + P - K}{S} + 1$$

where:

W = dimension of the input
P = padding
K = Kernel size
S = Stride

Example (unidimensional)

The width of the input (gray) is W=7.
The kernel has dimension K=3 with fixed weights $[1, 0, -1]$
Padding is zero



In the first case, the stride is S=1. We get $(W - K)/S + 1 = 5$ output values.
In the second case, the stride is S=2. We get $(W - K)/S + 1 = 3$ output values.

Example 2D

INPUT $[32 \times 32 \times 3]$ color image of 32×32 pixels. The three channels R G B define the input depth

CONV layer. Suppose we wish to compute 12 filters with kernels 6×6, stride 2 in both directions, and zero padding. Since $(32 - 6)/2 + 1 = 14$ the output dimension will be $[14 \times 14 \times 12]$

RELU layer. Adding an activation layer the output dimension does not change

Padding modes

Usually, there are two main "modes" for padding:

valid no padding is applied

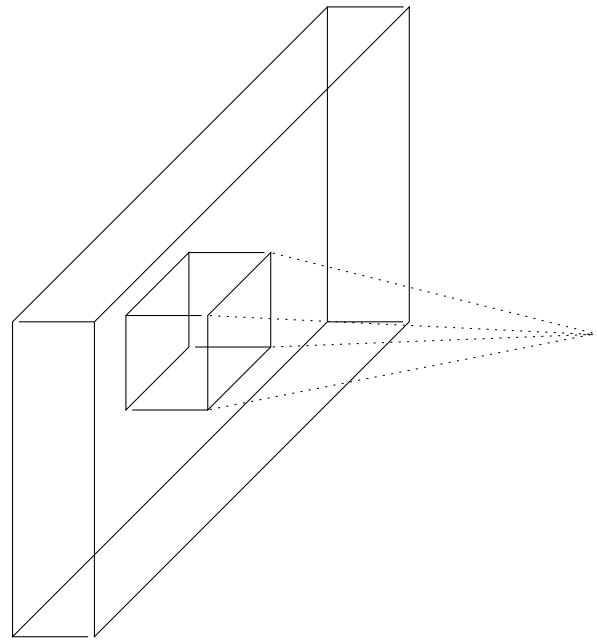
same you add a minimal padding enabling the kernel to be applied an integer number of times

Important remark

Unless stated differently (e.g. in separable convolutions), a filter operates on all input channels in parallel.

So, if the input layer has depth D, and the kernel size is NxM, the actual dimension of the filter will be

NxMxD



The convolution kernel is tasked with simultaneously mapping cross-channel correlations and spatial correlations

Demo

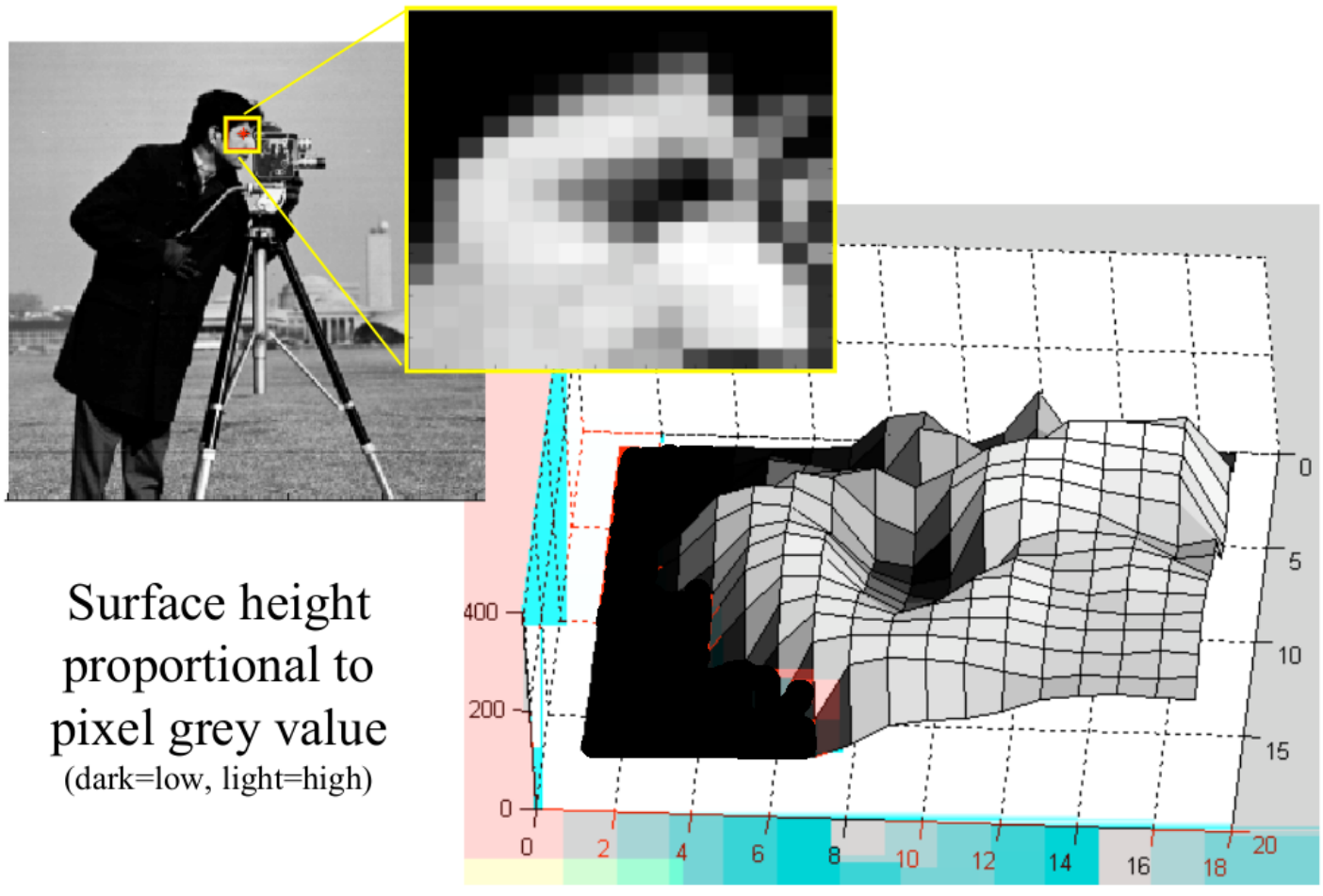
[Demo]

A parenthesis: filters in image processing

An image is coded as a numerical matrix (array)
grayscale (0-255) or rgb (triple 0-255)

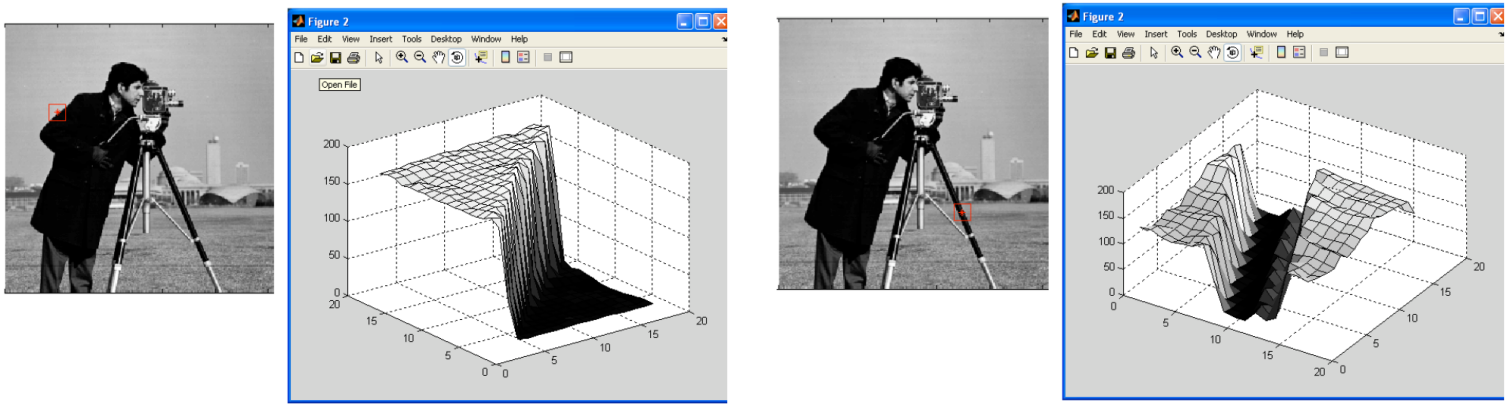
$$\begin{bmatrix} 207 & 190 & 176 & 204 & 204 & 208 \\ 110 & 108 & 114 & 112 & 123 & 142 \\ 94 & 100 & 96 & 121 & 125 & 108 \\ 95 & 86 & 81 & 84 & 88 & 88 \\ 69 & 51 & 36 & 72 & 78 & 81 \\ 74 & 97 & 107 & 116 & 128 & 133 \end{bmatrix}$$

Images as surfaces



Interesting points

Edges, angles, ...: points where there is a discontinuity, i.e. a fast variation of the intensity



More generally, are interested to identify patterns inside the image.

The key idea is that the kernel of the convolution expresses the pattern we are looking for.

Example: finite derivative

Suppose we want to find the positions inside the image where there is a sudden horizontal passage from a dark region to a bright one. The pattern we are looking for is

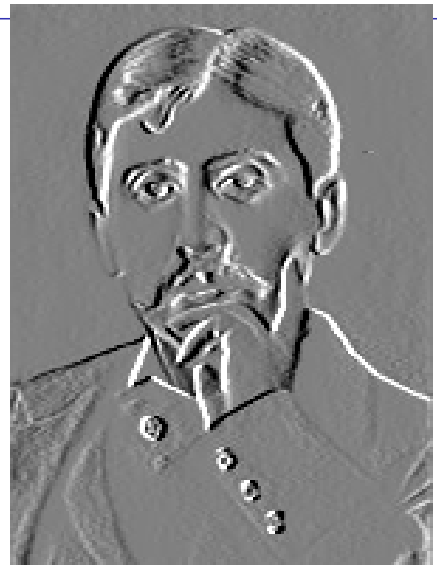
$$\begin{bmatrix} -1 & 1 \end{bmatrix}$$

or, varying the distance between pixels:

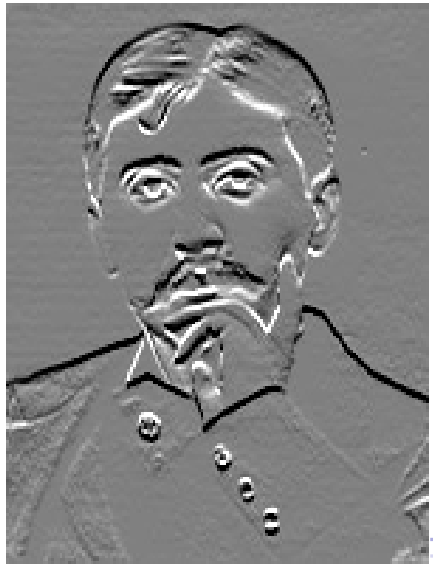
$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

The finite derivative at work

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \longrightarrow$$



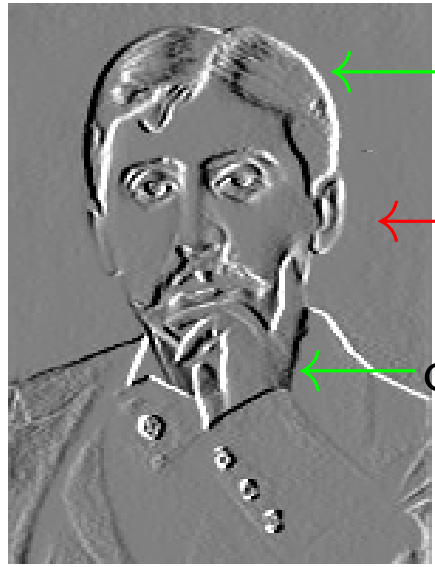
$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \longrightarrow$$



Recognizing Patterns

Each neuron in a convolutional layer gets activated by specific patterns in the input image.

$$\text{pattern} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



Another example: the finite laplacian

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Back to CNN: discovering patterns

So far we did examples with handcrafted kernels.
But how to find good patterns?

Usual idea:

instead of using human designed pre-defined patterns, let the net learn them.

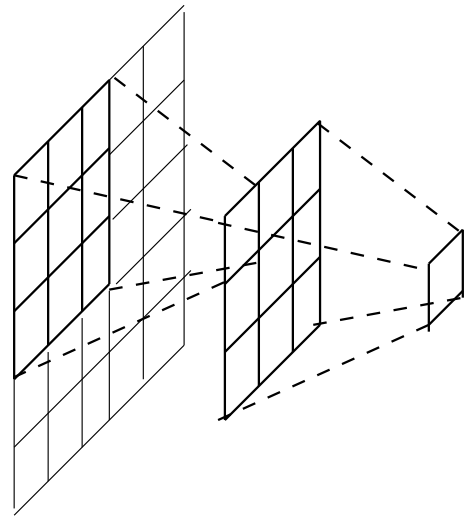
Particularly important in deep architectures, because:

- stacking kernels we **enlarge their receptive fields** (see next slide)
- adding non-linear activations we synthesize complex, **non-linear kernels**

Receptive field

The receptive field of a (deep, hidden) neuron is the dimension of the input region influencing it.

It is equal to the dimension of an input image producing (without padding) an output with dimension 1.



A neuron cannot see anything outside its receptive field!

We may also rapidly enlarge the receptive fields by means of **downsampling** layers, e.g. pooling layers or convolutional layers with non-unitarian stride

Deep filters

If you are curious to see what kind of filters are synthesized by CNNs, and how CNNs "see" the world, check this blog: An exploration of convnet filter with keras