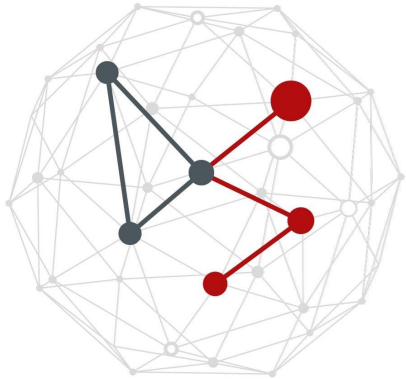
MiME
Master's degree ICT Internet Multimedia Engineering

CONVOLUTIONAL NEURAL NETWORKS (CNN)

Michele Rossi
michele.rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

UNIVERSITÀ DEGLI STUDI DI PADOVA

Outline

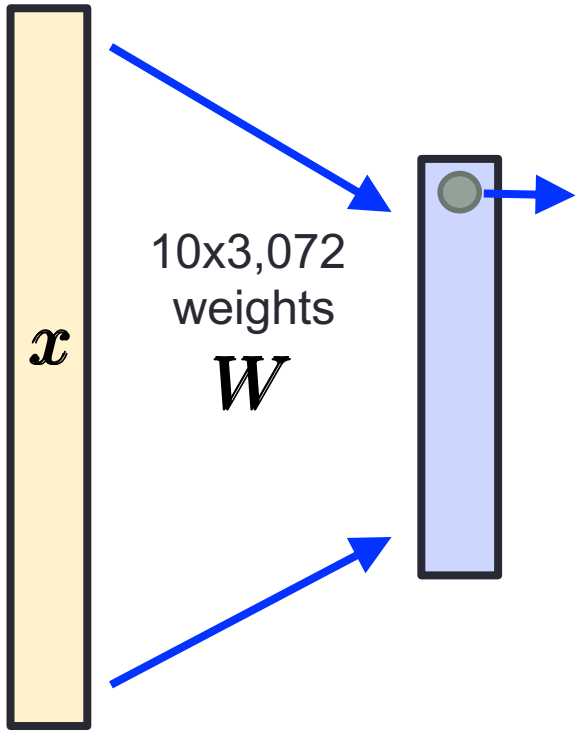
- Introduction to CNNs
- The convolution operation
  - Visual insight
  - Convolution operator
  - Input and output volumes
  - Kernels, padding
- Characteristics of CNNs
  - Sparse interactions
  - Parameter sharing
  - Equivariance (pooling)
- Backprop for CNN layers
- Bibliography

CNN - introduction

- CNNs are a specialized kind of ANN to
  - process data that has a known *grid-like topology*
  - 1D time series (grid over time)
  - 2D images (grid over space)
- CNN
  - Employ a mathematical operation called convolution
  - **Convolution**
    - is a specialized kind of linear operation
    - differs slightly from standard convolution
    - is used in place of matrix multiplication in at least one layer
  - Have been hugely successful in many applications

Fully connected (dense) layer

- Example 32x32x3 image (3 color ch.) → flatten to 3,072x1



each entry j corresponds to one number obtained taking the dot product between:

- Row j of matrix **W**
- The input vector **x**

$$Wx$$

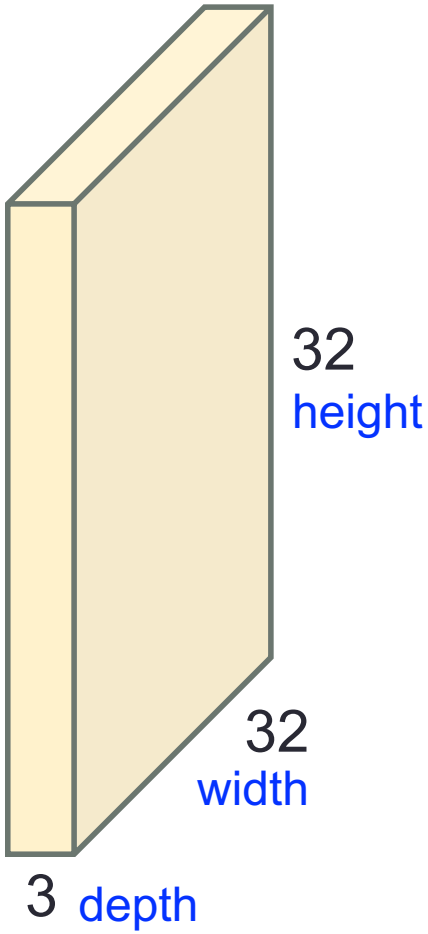
- And then applying a non-linear activation function

input vector
3,072x1

first layer's neurons
10x1

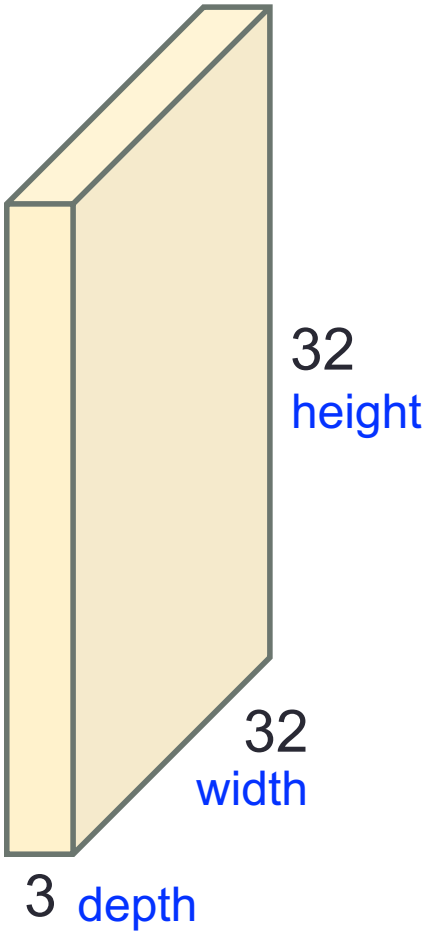
Convolutional layer

- Example 32x32x3 image → preserve spatial structure

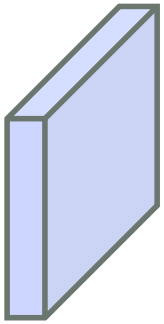


Convolutional layer

32x32x3 image



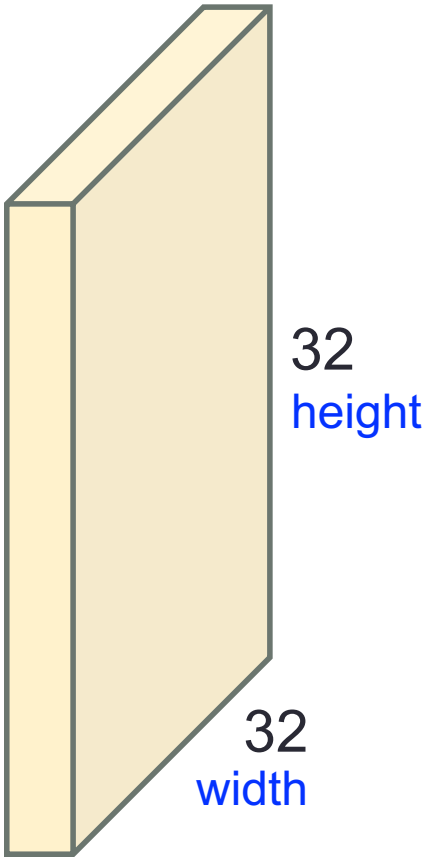
5x5x3 filter (or kernel)



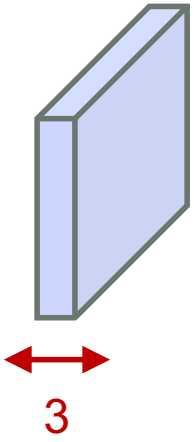
convolve the filter with the input image, i.e., *"slide over the image spatially convolving the kernel with each image patch"*

Convolutional layer

32x32x3 image



5x5x3 filter (or kernel)



Filters always extend the full depth of the input volume

1D convolution (1/2)

- 1D: operation between real valued functions
- Suppose
  - we want to track the position of a spaceship x(t)
  - using a (noisy) laser sensor
- Due to noisy measurements
  - To obtain better estimates: we average together several measurements
  - However: we would like to weigh more recent samples than past ones
  - So, we define a new weight w(a), a being the "age" of a measurement
  - s(t) provides a smoothed estimated of the spaceship position

$$s(t) = \int x(a)w(t-a)da \stackrel{\Delta}{=} (x * w)(t)$$

1D convolution (2/2)

$$s(t) = \int x(a)w(t-a)da \triangleq (x * w)(t)$$

feature map — input — kernel

- In our example
  - w(a) needs to be a valid PDF or the output is not a weighted average
  - Needs to be zero for all negative values, or it will look into the future
  - These facts hold true for this specific problem, however
  - In general, a convolution is defined for each function for which the integral is defined, and may be used for other purposes
- CNN terminology
  - **x**: referred to as the input
  - **w**: referred to as the kernel
  - **s**: referred to as the output feature map

2D convolution

- In machine learning applications
  - Time is usually discrete (quantized, data sampled at regular intervals)
  - The data is usually a multidimensional array
  - The kernel is usually a multidimensional array of parameters

- Convolution over more than one axis at a time
  - Example, 2D image I (input), we need a 2D Kernel K(m,n)

$$S(m,n) = (I * K)(m,n) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} I(i,j)K(m-i,n-j)$$

- I and K: defined over a finite set and zero outside of it

2D convolution example

- Example 3x3 kernel
  - Called "Sobel filter"
  - Used to detect edges

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

$K(m,n)$

- Input 3x3 image

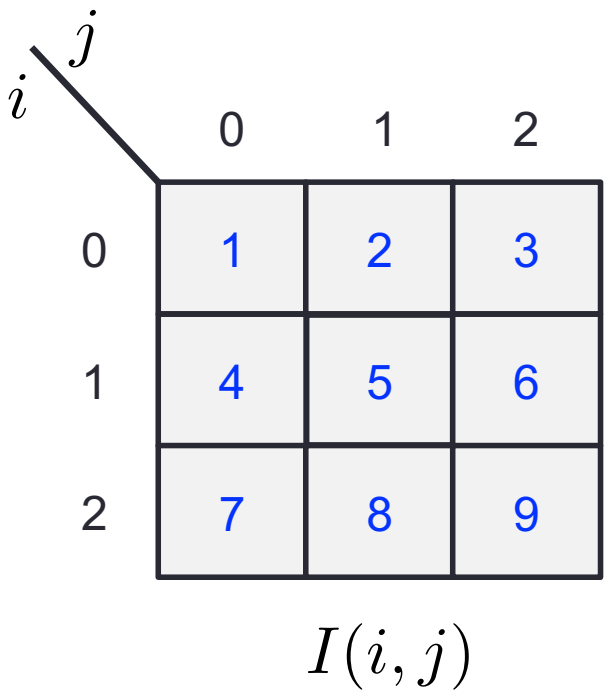
1	2	3
4	5	6
7	8	9

$I(i,j)$

Output:

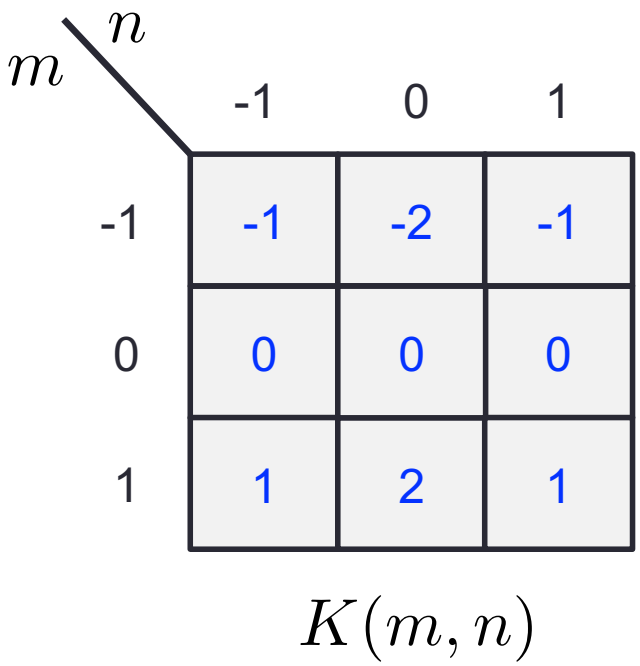
$$S(m,n) = (I * K)(m,n) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} I(i,j)K(m-i,n-j)$$

Indexing



$I(i, j)$

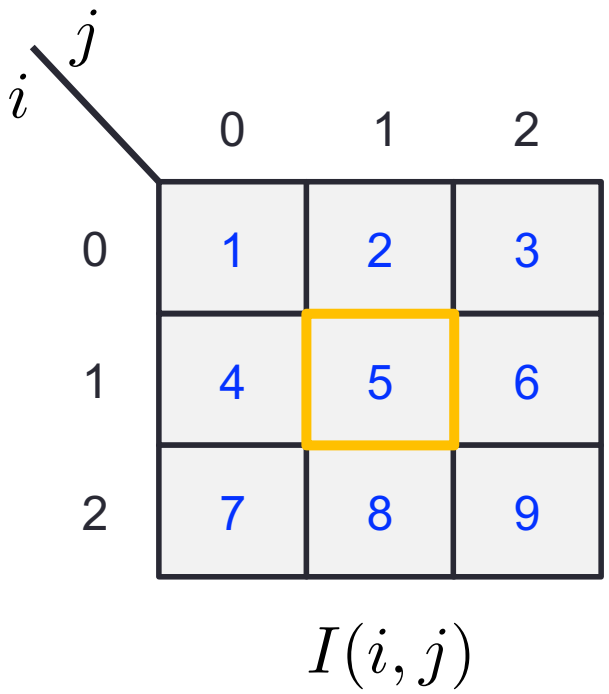
Input 2D signal



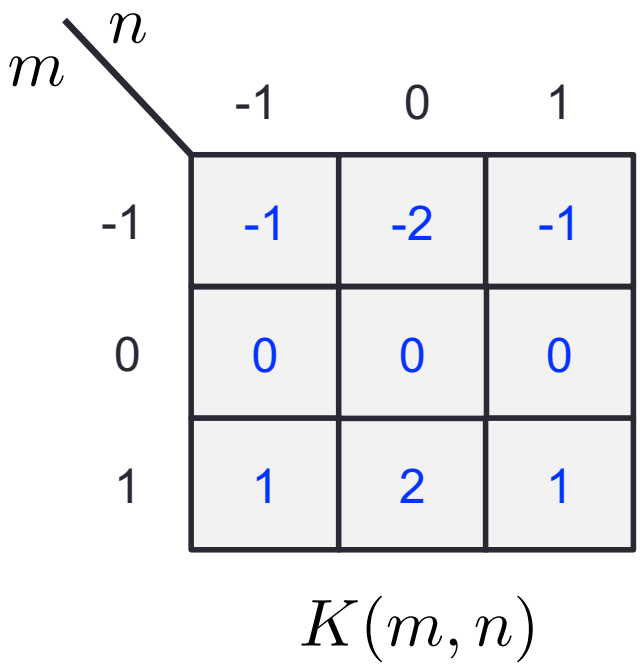
$K(m, n)$

Kernel

Compute S(1,1) (m=1, n=1)



$I(i,j)$

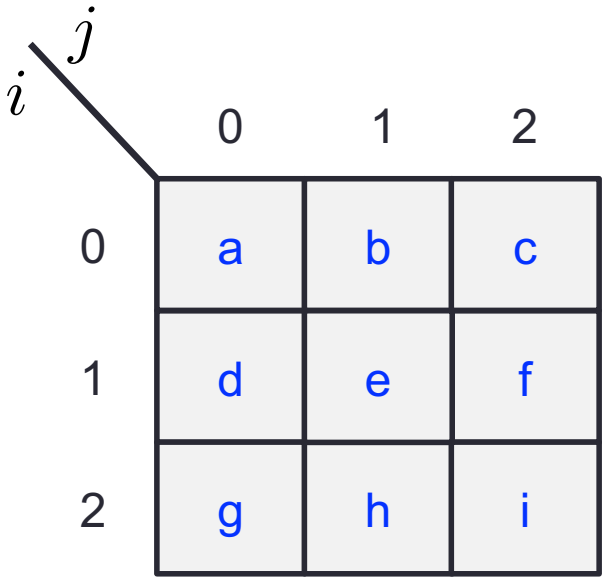


$K(m,n)$

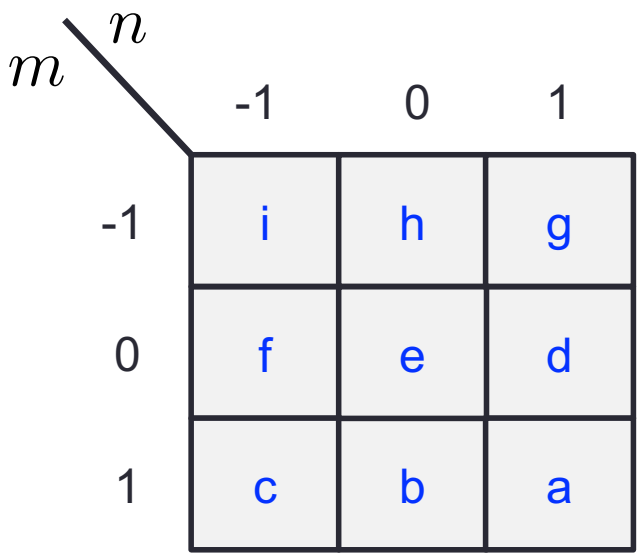
Result (applying the convolution Equation):

$$S(1,1) = I(0,0)K(1,1) + I(0,1)K(1,0) + I(0,2)K(1,-1) +$$
$$+ I(1,0)K(0,1) + I(1,1)K(0,0) + I(1,2)K(0,-1) +$$
$$+ I(2,0)K(-1,1) + I(2,1)K(-1,0) + I(2,2)K(-1,-1)$$

S(1,1) (m=1, n=1)



$I(i,j)$

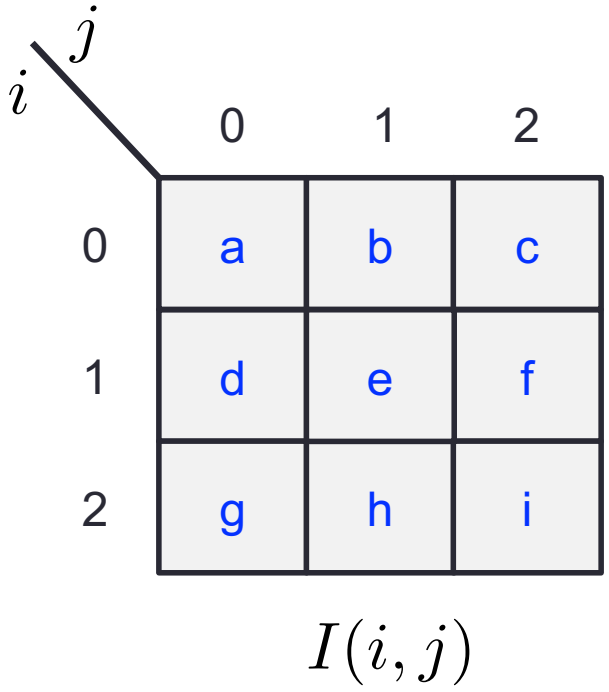


$K(m,n)$

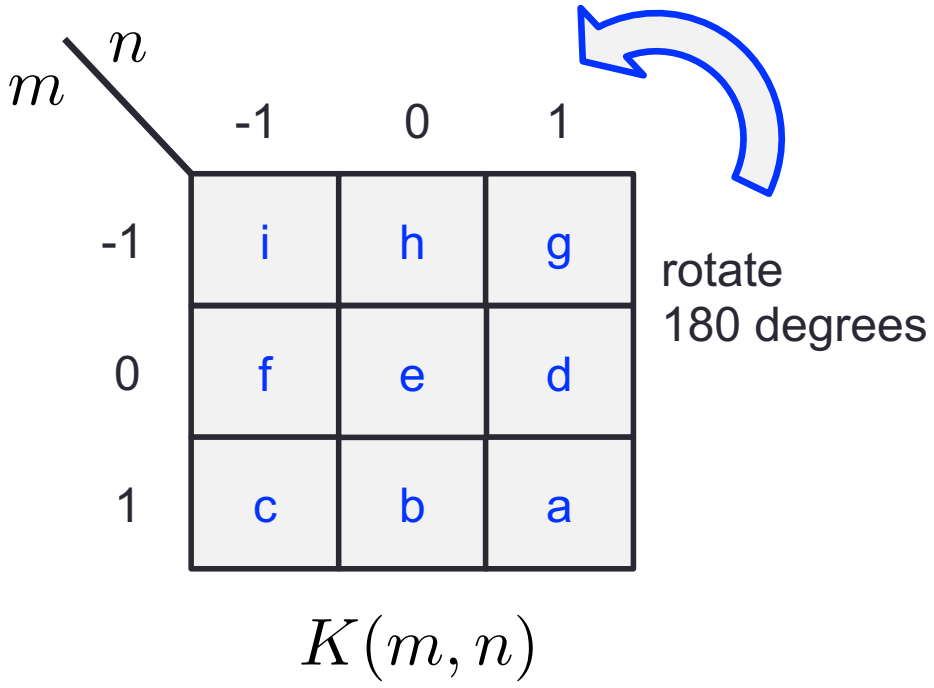
Cells with the same letter are multiplied together!

$$S(1,1) = I(0,0)K(1,1) + I(0,1)K(1,0) + I(0,2)K(1,-1) +$$
$$+ I(1,0)K(0,1) + I(1,1)K(0,0) + I(1,2)K(0,-1) +$$
$$+ I(2,0)K(-1,1) + I(2,1)K(-1,0) + I(2,2)K(-1,-1)$$

Trick to speed up computation



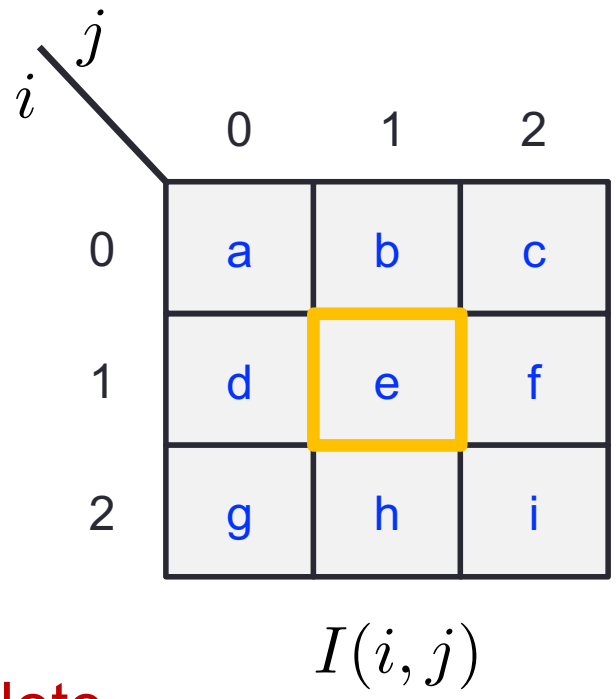
$I(i,j)$

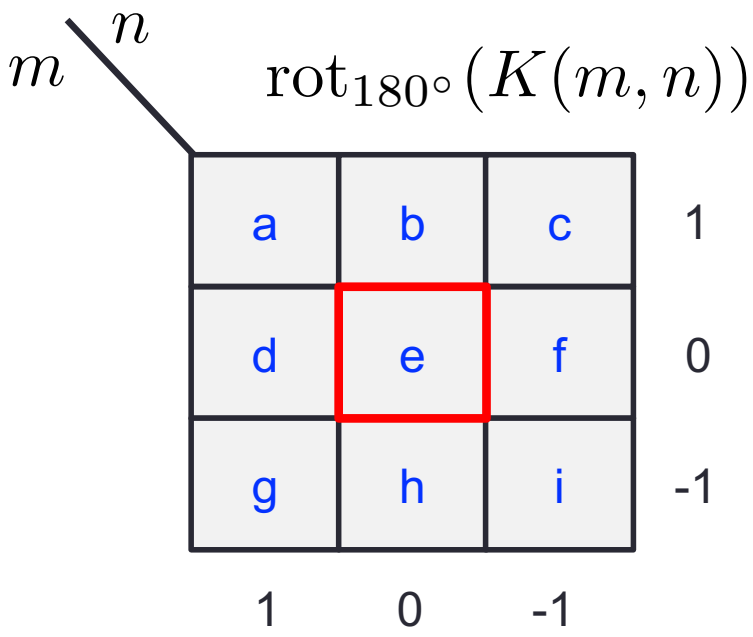


$K(m,n)$

STEP 1) Rotate the Kernel 180 degrees

Trick to speed up computation





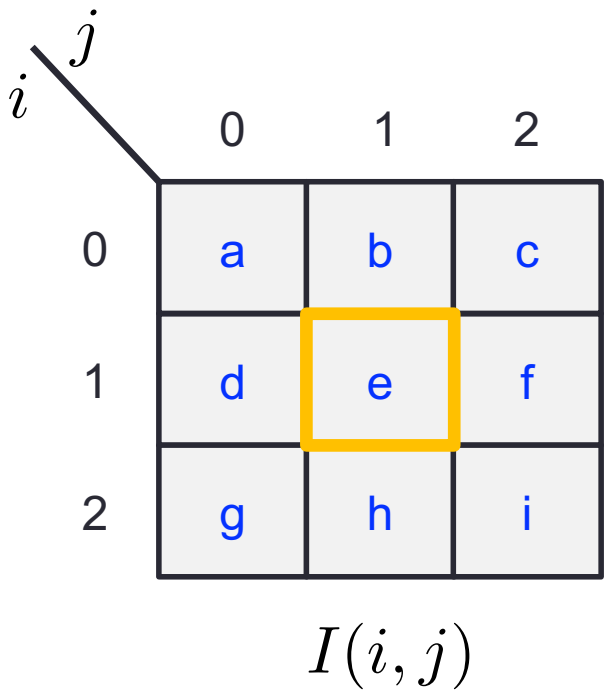
Note

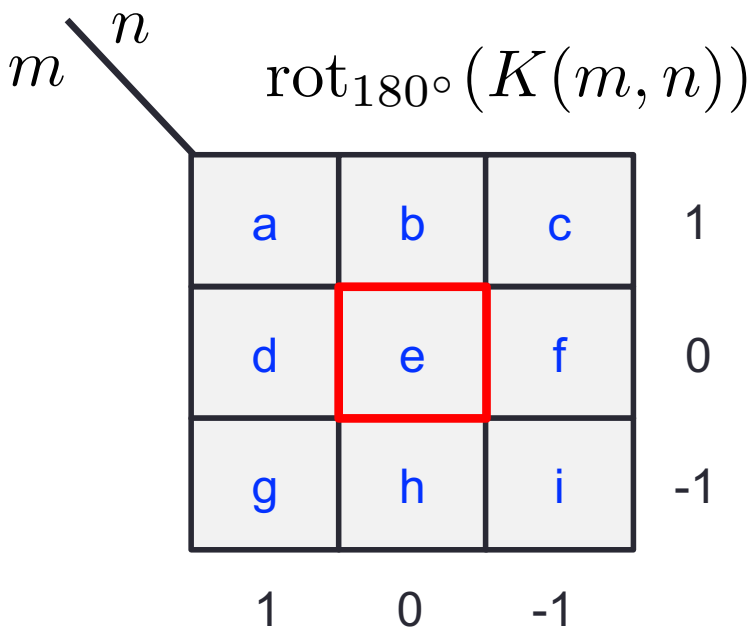
- The flipped Kernel has all the elements aligned with the input

STEP 2) Align the flipped Kernel with the input

- Positioning the Kernel cell with coordinate (0,0)
- So as it overlaps the input cell (1,1)

Trick to speed up computation

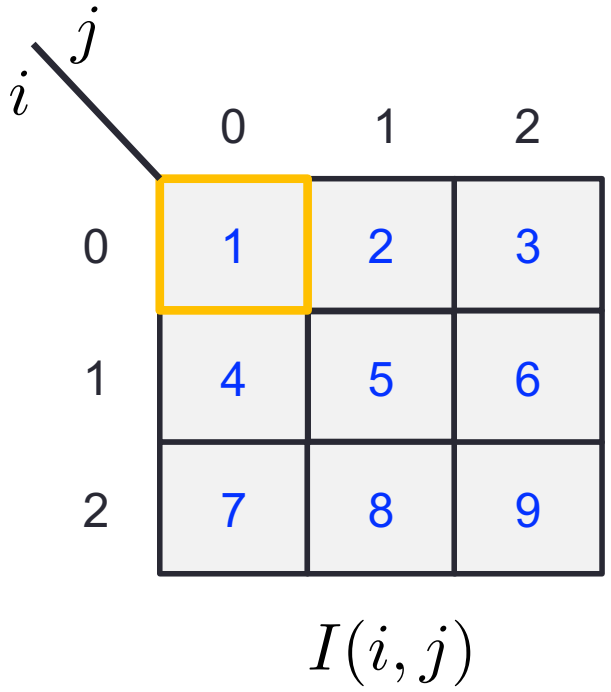




STEP 3) Products and sum

- Take product of elements in same position
- Sum those products, $S(1,1) = a^2+b^2+c^2+ \ldots +i^2$

Computing S(0,0)

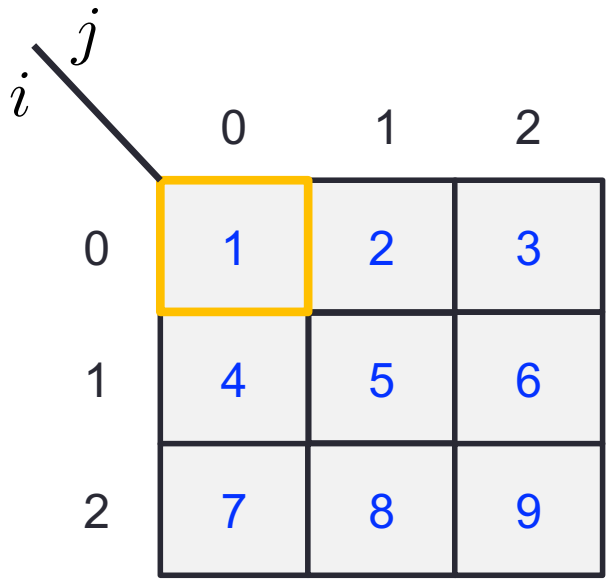


Using the convolution Equation:

$$
\begin{aligned}
S(0,0) &= I(0,0)K(0,0) + I(0,1)K(0,-1) + I(0,2)K(0,-2) + \\
&= I(1,0)K(-1,0) + I(1,1)K(-1,-1) + I(1,2)K(-1,-2) + \\
&= I(2,0)K(-2,0) + I(2,1)K(-2,-1) + I(2,2)K(-2,-2)
\end{aligned}
$$

Some terms are zero (the kernel is not defined there)

Computing S(0,0)



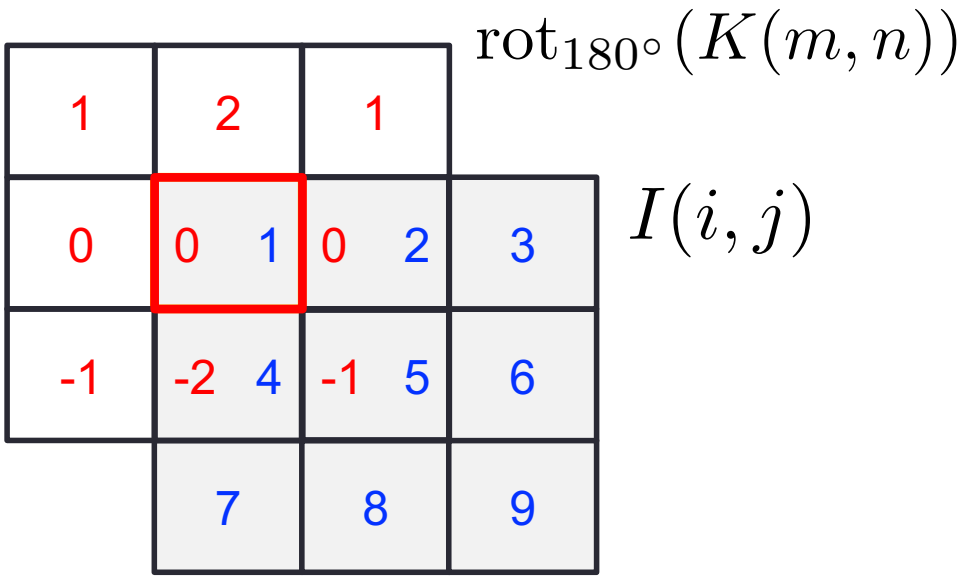
$\mathrm{rot}_{180^\circ}(K(m,n))$

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

Convolution

- Flip the Kernel

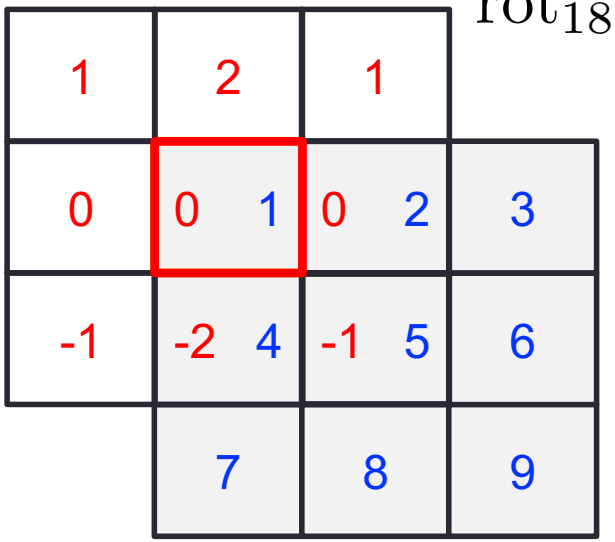
Computing S(0,0)



Convolution

- Align flipped Kernel with signal's cell (0,0)

Computing S(0,0)



$\mathrm{rot}_{180^\circ}(K(m,n))$

$I(i,j)$

Convolution

- Take element-wise products and sum

$$S(0,0) = 0 \times 1 + 0 \times 2 - 2 \times 4 - 1 \times 5 = -13$$

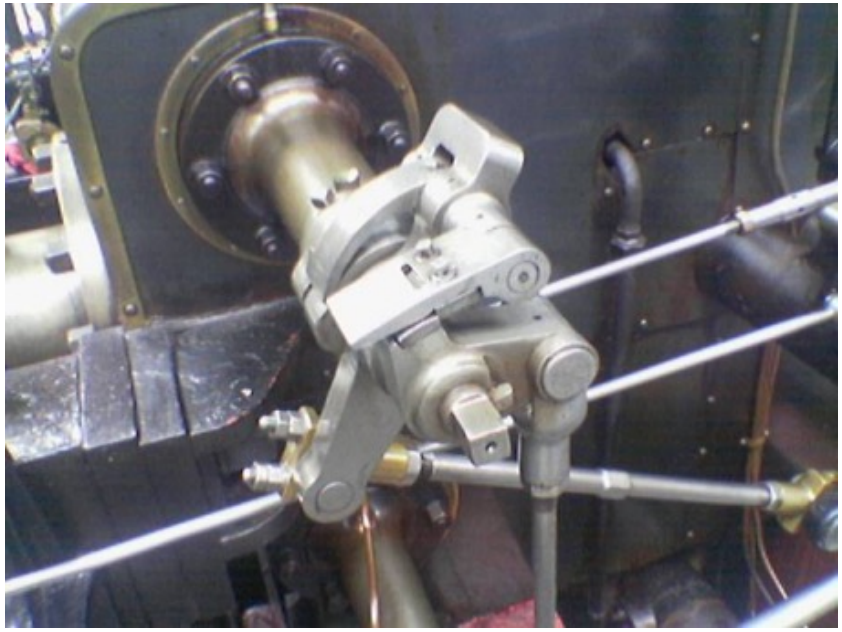
The output matrix for this example

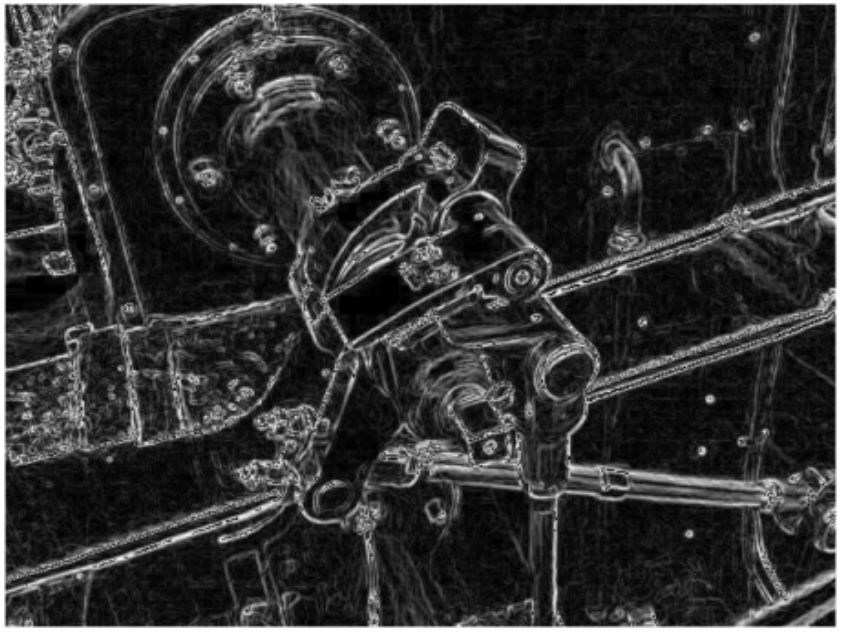
-13	-20	-17
-18	-24	-18
13	20	17

$S(i,j)$

Repeating the calculation for all entries

Applying the Sobel filter on a real image





Cross-correlation

$$(I \otimes K)(m,n) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} I(i+m, j+n) K(i,j)$$

- It is often preferred for CNN implementations
- Reduces the variability of the indices
  - Leads to faster implementations

If we define:

$$K'(m,n) = \mathrm{rot}_{180^\circ}(K(m,n))$$

We have that:

$$(I * K)(m,n) = (I \otimes K')(m,n)$$

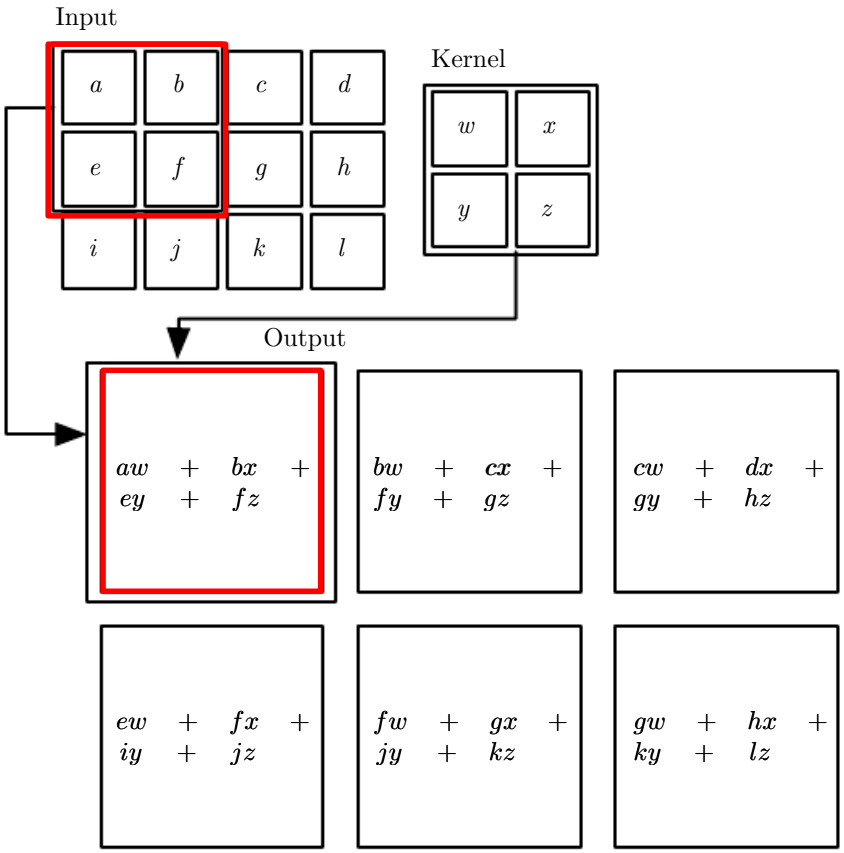
Convolution: equivalent to cross-correlation with a flipped Kernel

Convolution

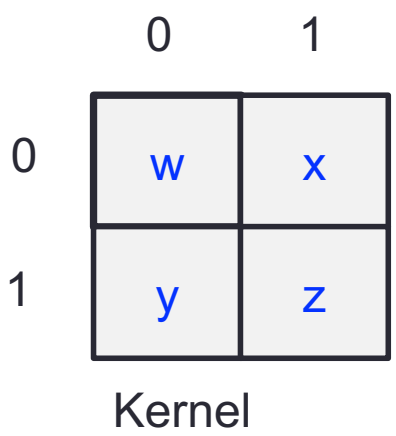
- In machine learning
  - Whether to flip the Kernel or not is irrelevant
  - The goal is to learn a convolution Kernel
- Simply
  - If flipping is considered → the learned Kernel will be flipped
  - If flipping is not used → the learned Kernel will not be flipped
- The following blocks will not be affected by this
  - The output will be the same irrespective of whether we are using flipped weight matrices or not

Cross-correlation (1/2)

- 2D cross-correlation example (without Kernel flipping)
- Input volume depth d=1 (corresponds to a 2D matrix)

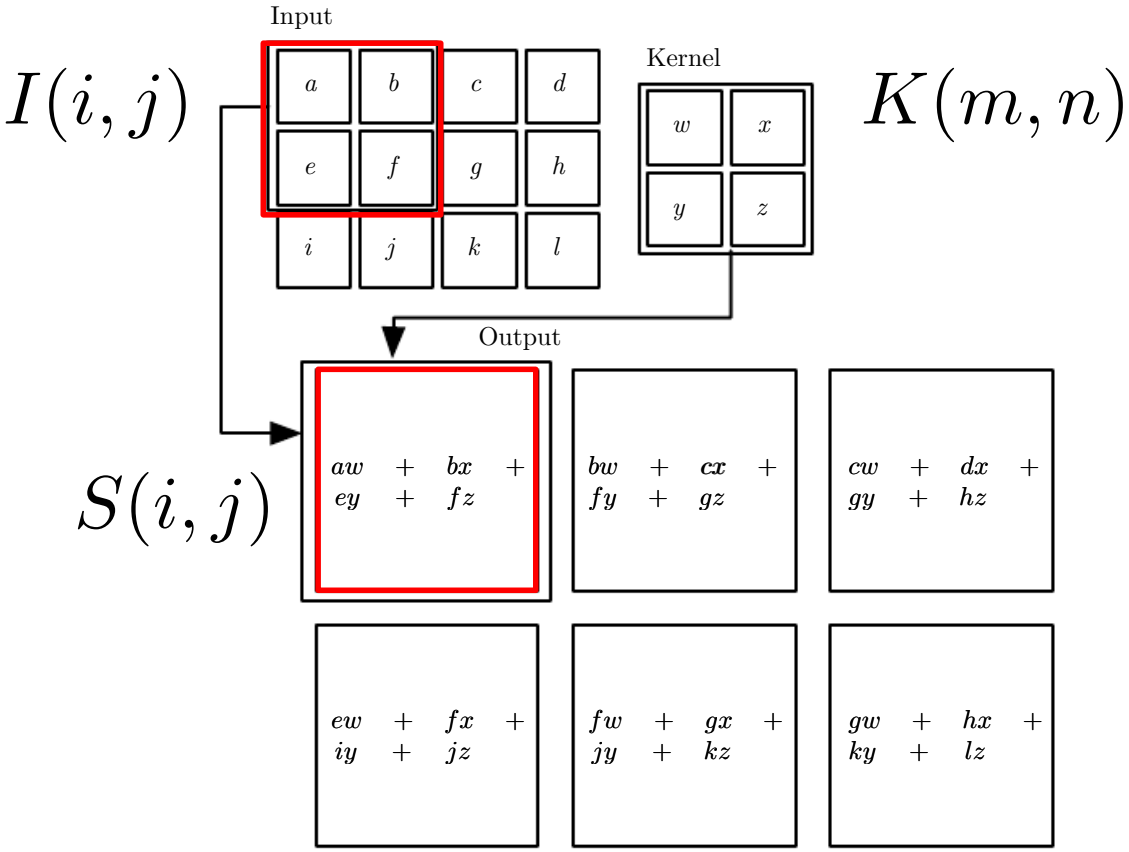


Kernel indexing

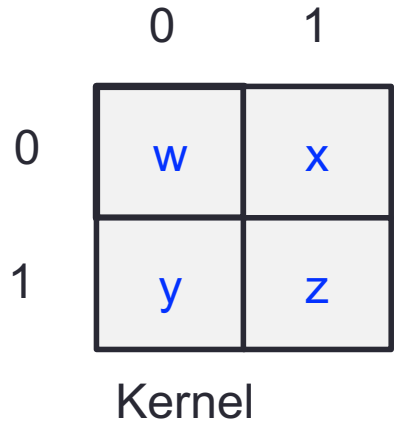


Cross-correlation (2/2)

$$S(i,j) = (I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(m,n)$$



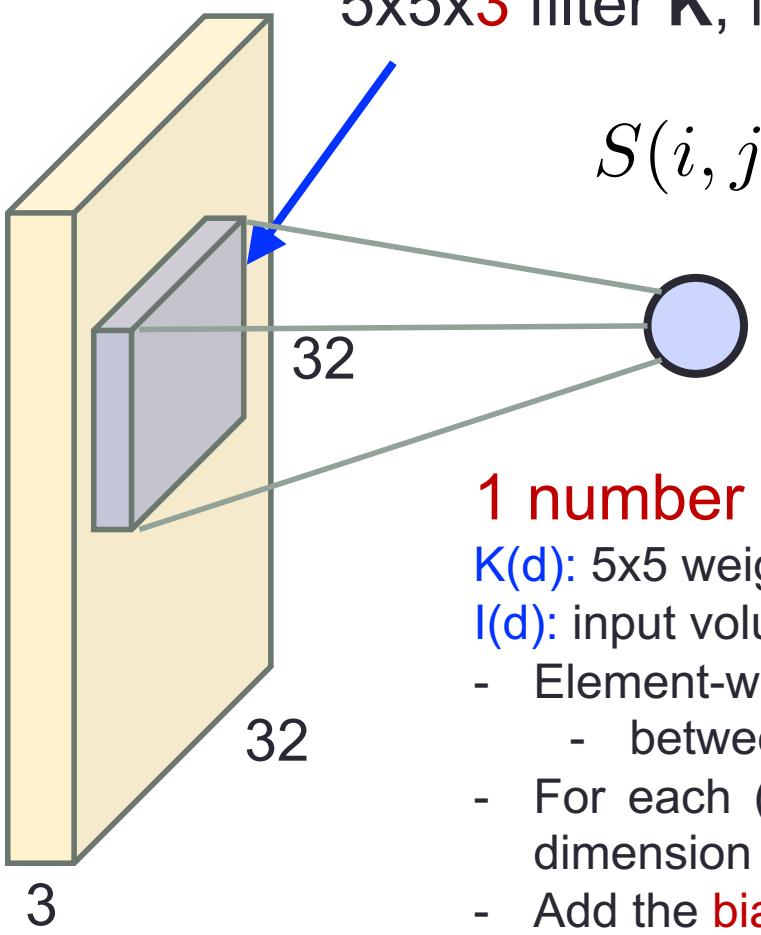
Kernel indexing



Convolution - input volume depth d=3

32x32x3 image **I**

5x5x3 filter **K**, filter depth d=3



$$S(i,j) = \sum_d (I(d) \otimes K(d))_{i,j} + b$$

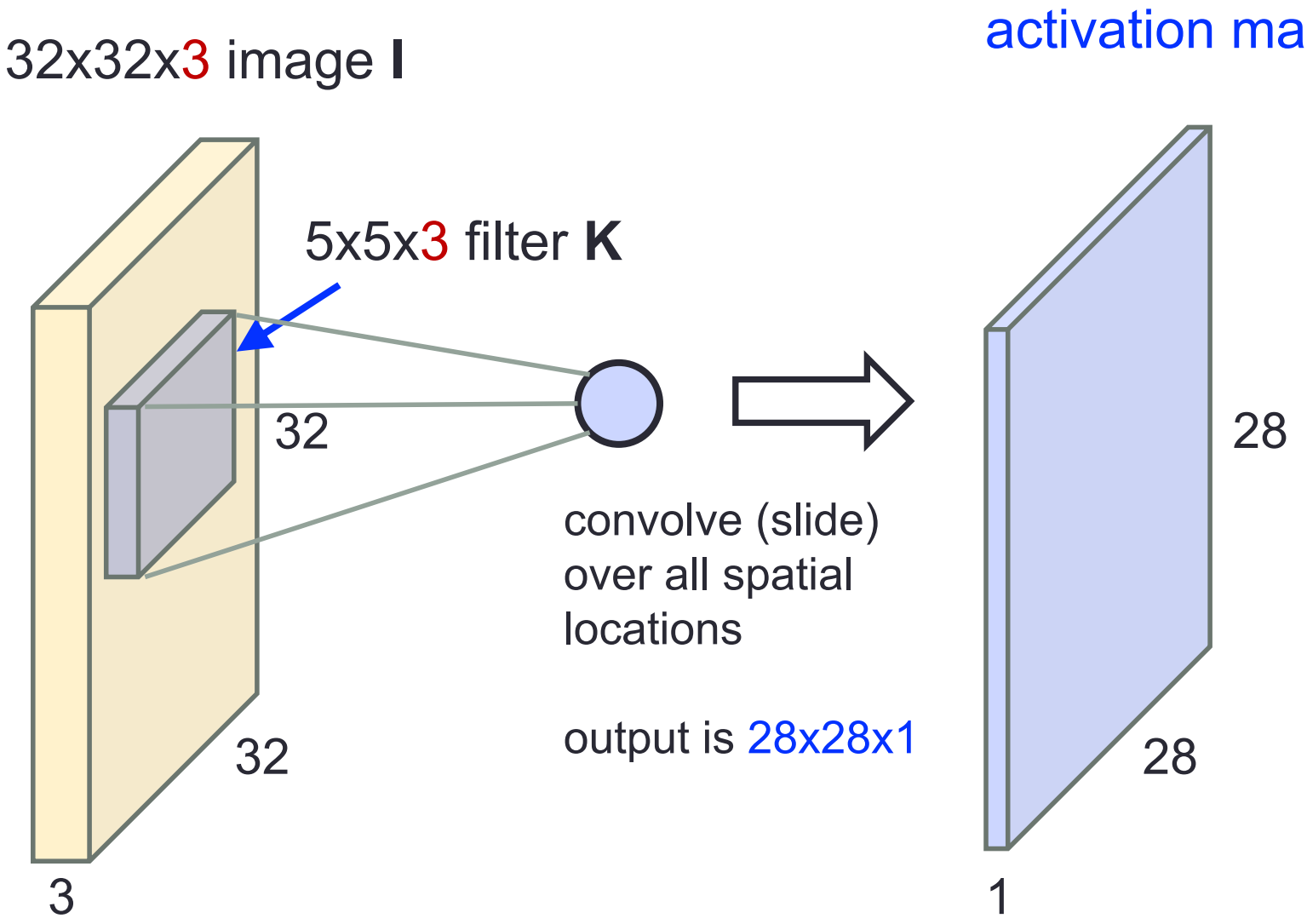
1 number - also called linear activation

K(d): 5x5 weight matrix along direction d

I(d): input volume along direction d

- Element-wise product (cross-correlation)
  - between image I(d) and K(d)
- For each (i,j) Results into a single number for each dimension d, sum over d
- Add the bias b

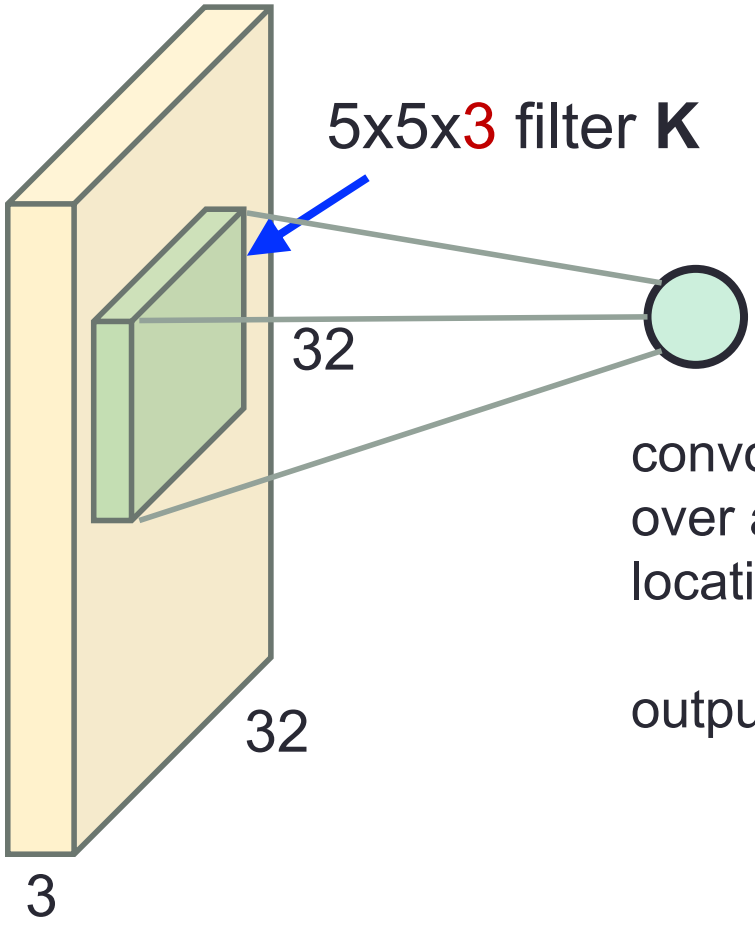
Convolution



Convolution

32x32x3 image **I**

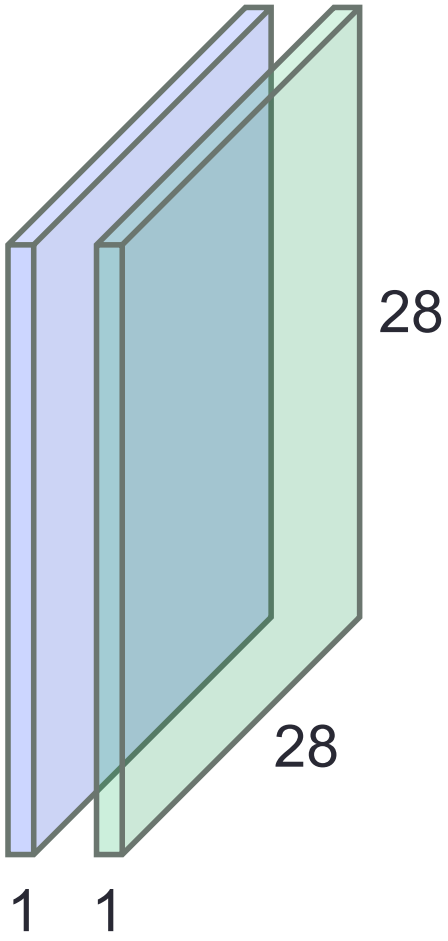
5x5x3 filter **K**



convolve (slide) over all spatial locations

output is 28x28x1

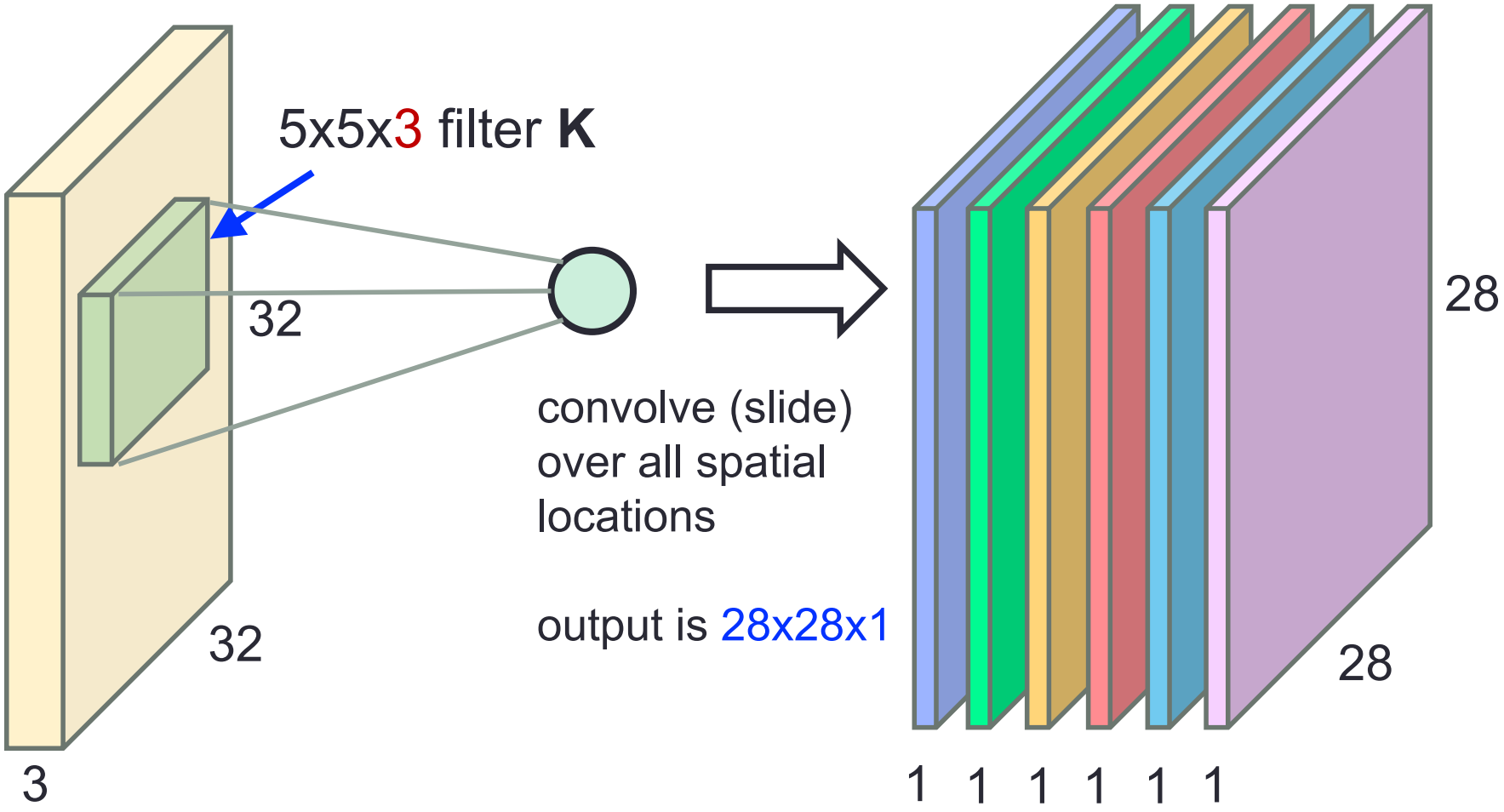
consider a second **green** filter
2 activation maps



Convolution

If we have 6, 5x5x3 filters, we get
6 activation maps of size 28x28x1

32x32x3 image **I**

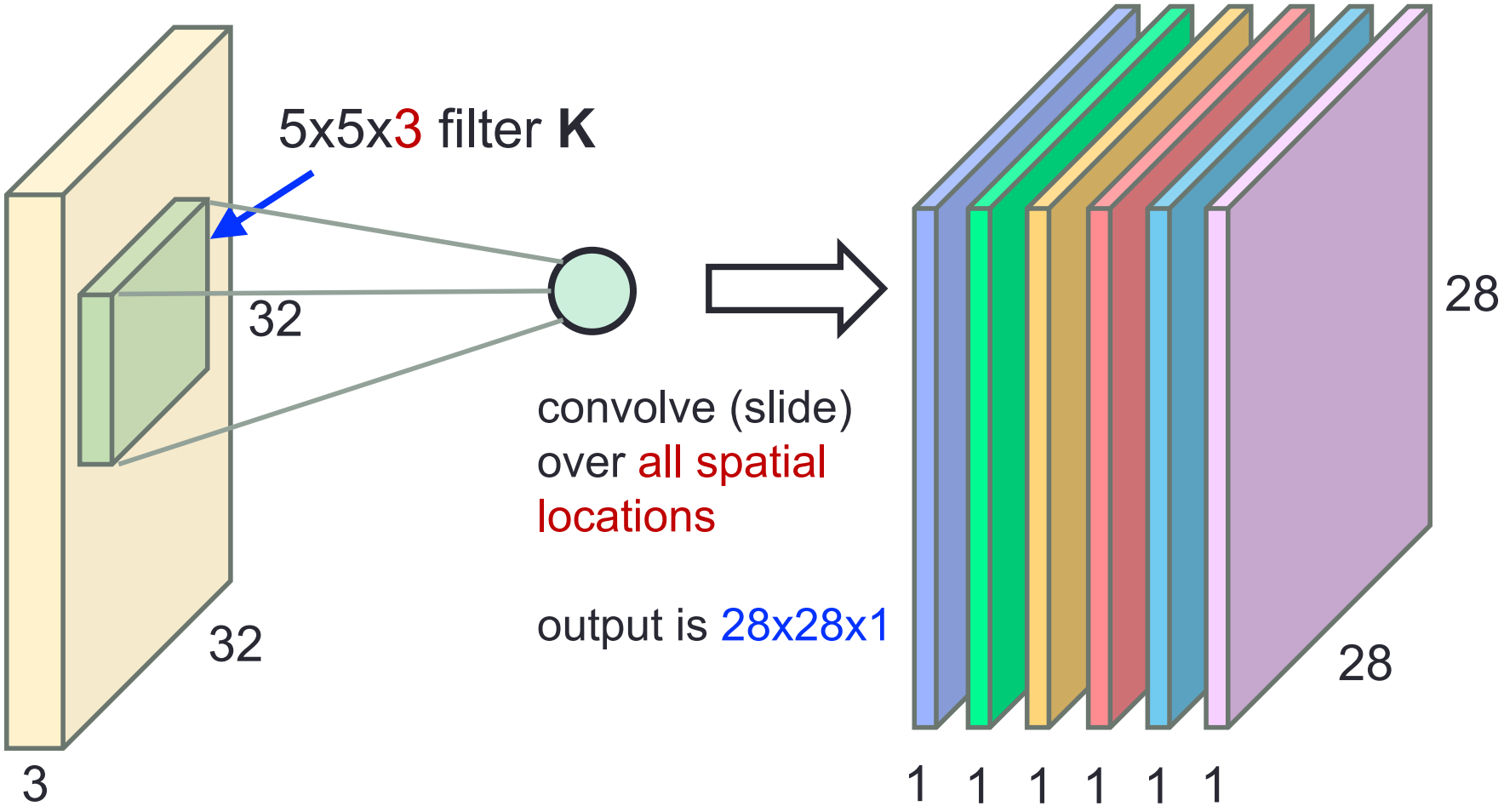


Convolution

depth of output volume is 6 (6 filters)

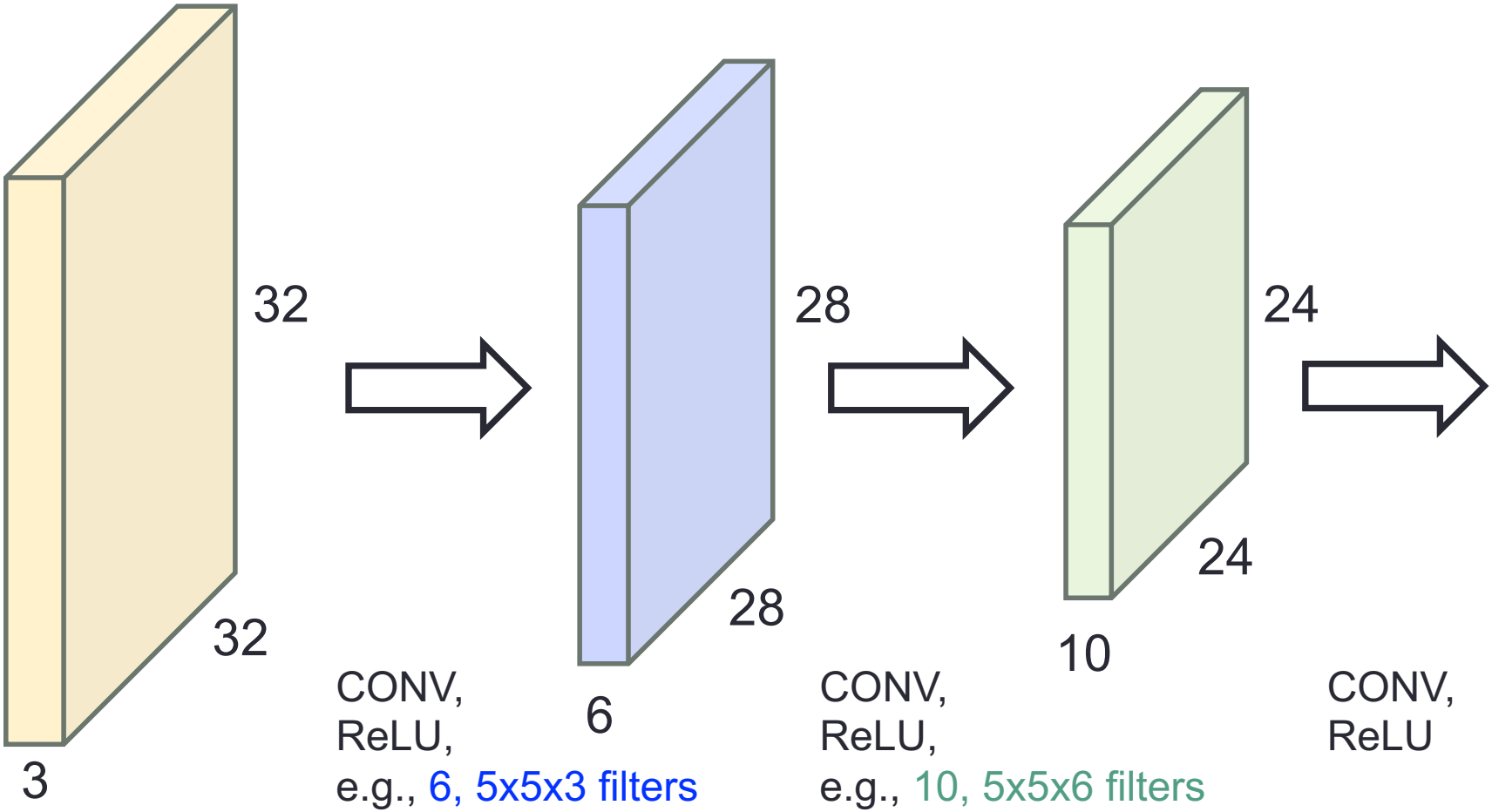
we obtain 6 2D feature maps

32x32x3 image **I**



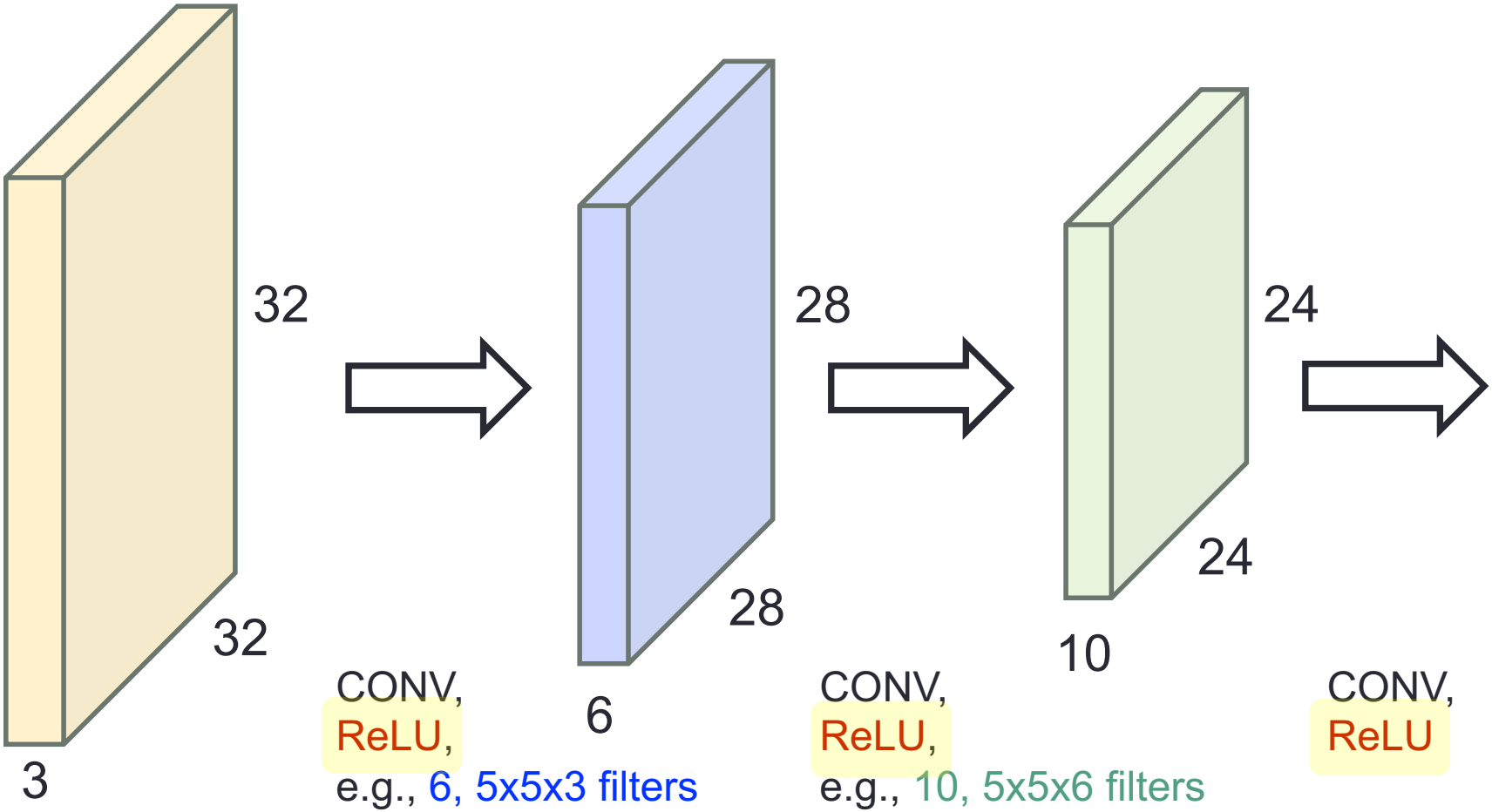
Convolutional network

It is a sequence of convolutions *interspersed* with activation functions

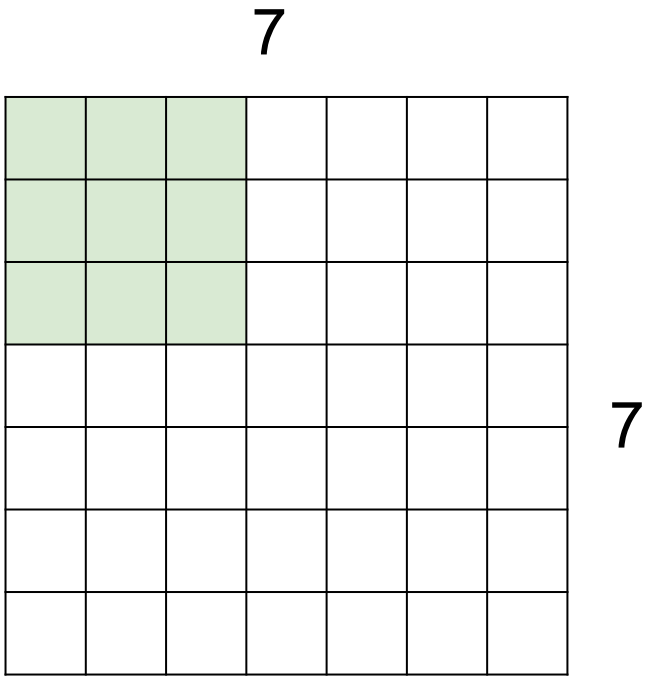


Convolutional network

It is a sequence of convolutions *interspersed* with activation functions



A closer look at spatial dimensions



STEP 1

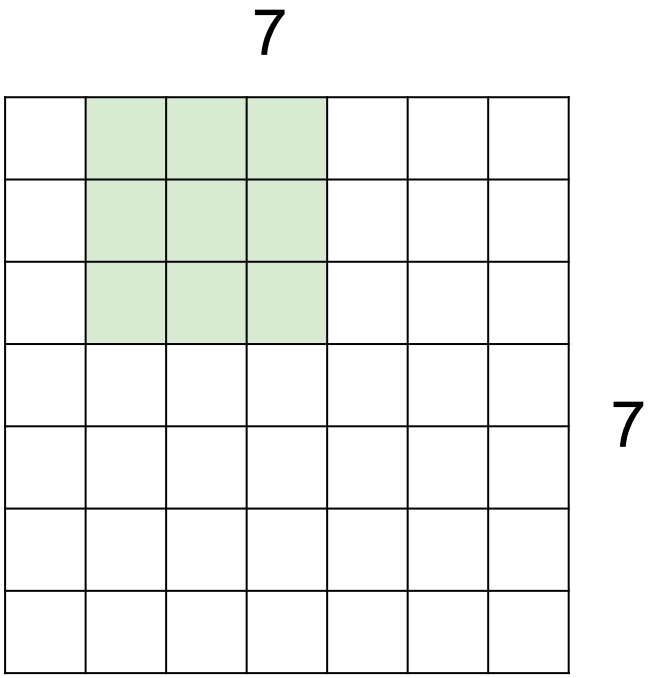
7x7(x1) input

assume a 3x3(x1) green filter

sliding step = 1

sliding step is referred to as stride

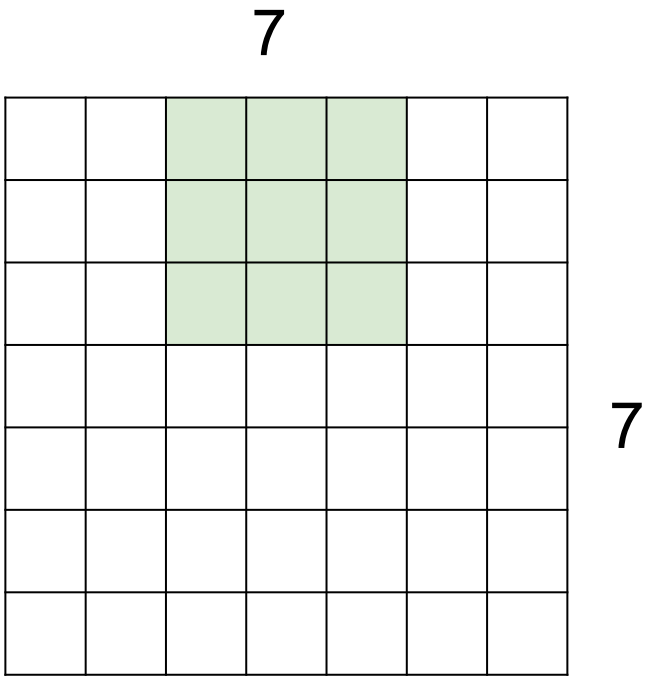
A closer look at spatial dimensions



STEP 2

move filter one step to the right

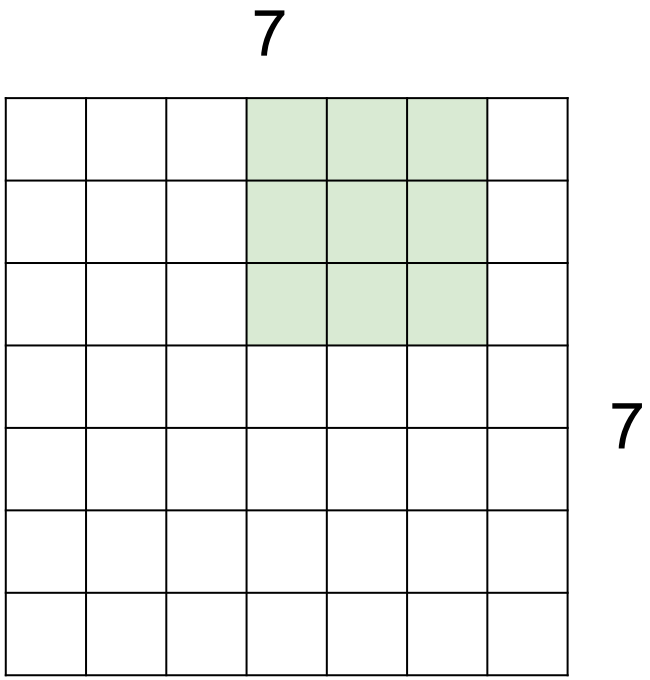
A closer look at spatial dimensions



STEP 3

move one more step to the right

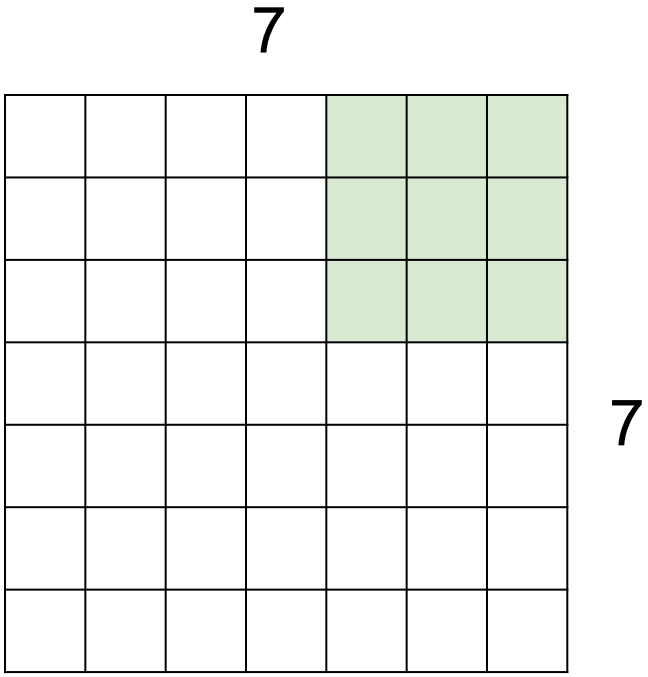
A closer look at spatial dimensions



STEP 4

move one more step to the right

A closer look at spatial dimensions

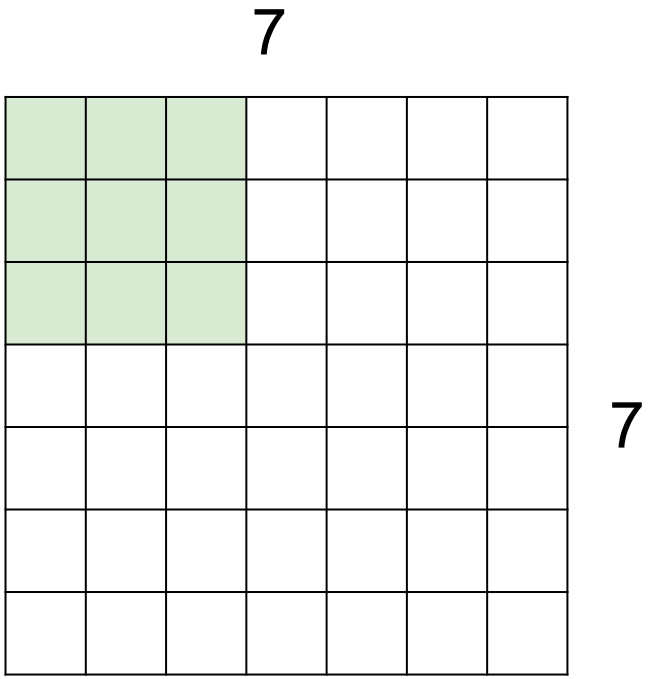


STEP 5

move one more step to the right

5x5 output

A closer look at spatial dimensions

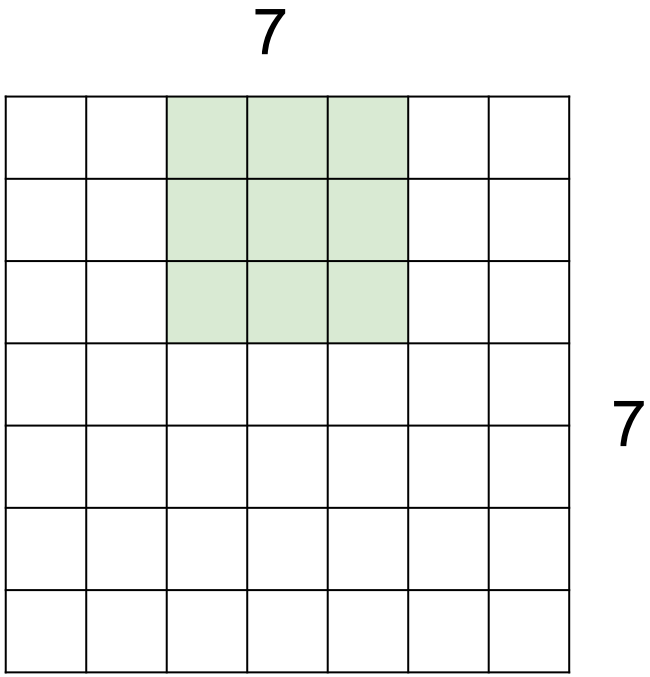


STEP 1

7x7 spatial input
assume a 3x3 green filter

apply a stride of 2

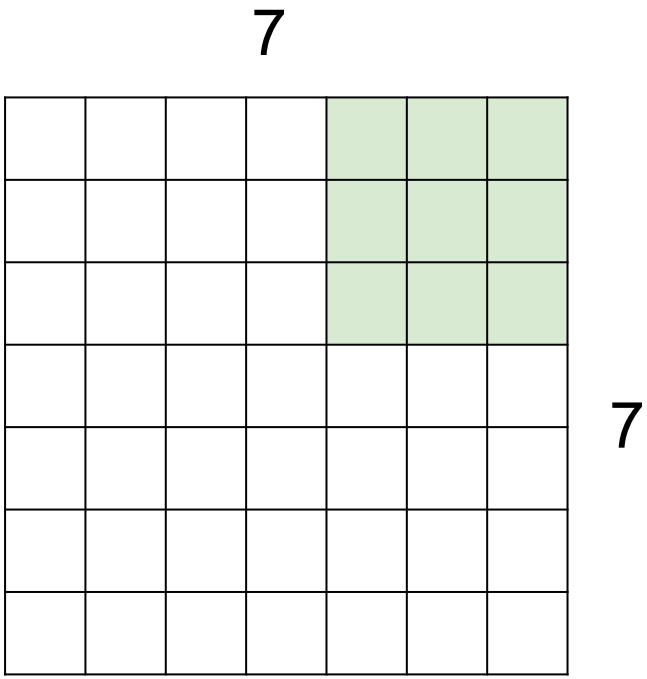
A closer look at spatial dimensions



STEP 2

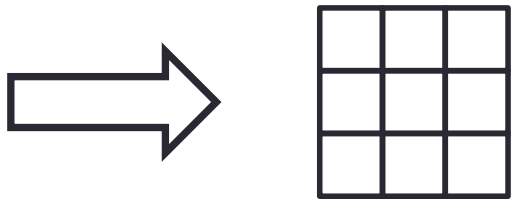
move 2 steps to the right

A closer look at spatial dimensions



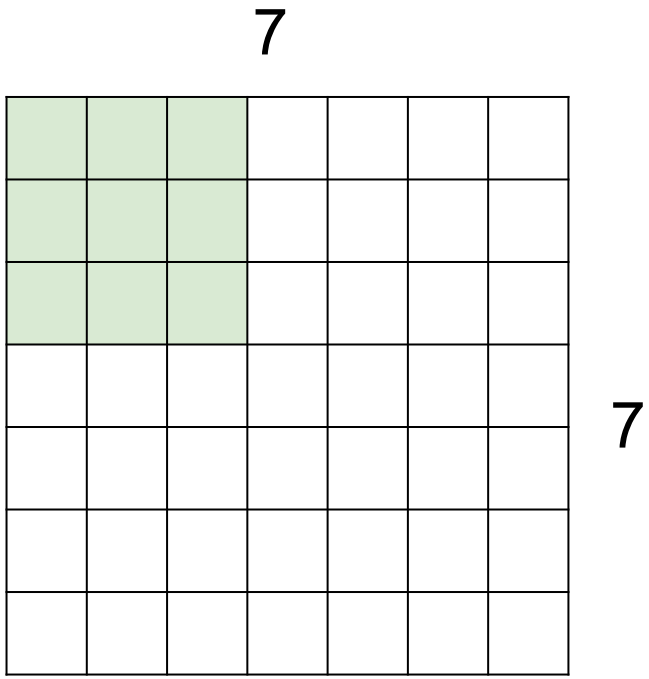
STEP 3

move 2 more steps to the right



3x3 output

Spatial dimensions: stride = 3

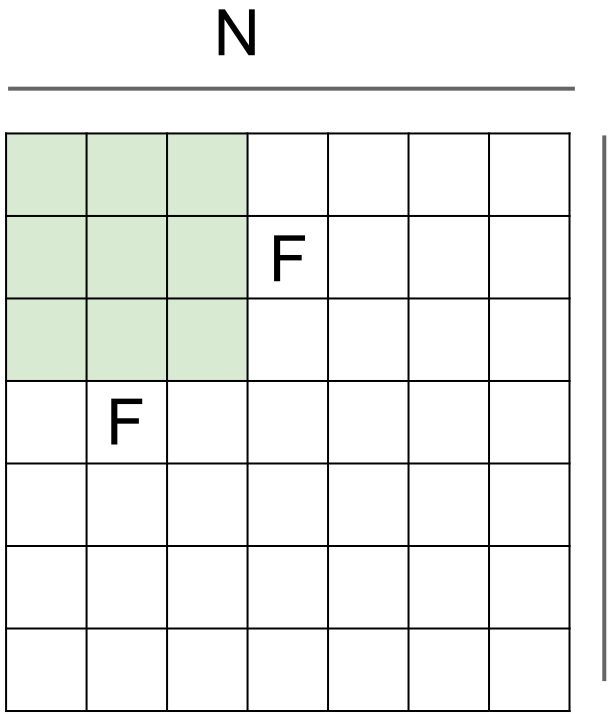


We now try with a stride of 3

Does not fit!

Cannot apply a 3x3 filter with a 7x7 input and stride = 3

Output size *vs* filter size & stride

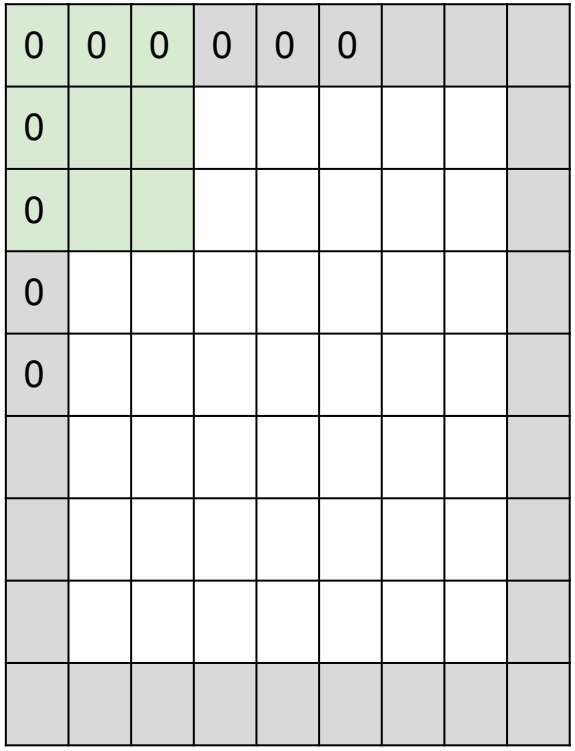


Output volume (OxO):
O = (N-F) / stride + 1

e.g., N=7, F=3
stride=1, O = (7-3)/1 + 1 = 5
stride=2, O = (7-3)/2 + 1 = 3
stride=3, O = (7-3)/3 + 1 = 2.33 :-(

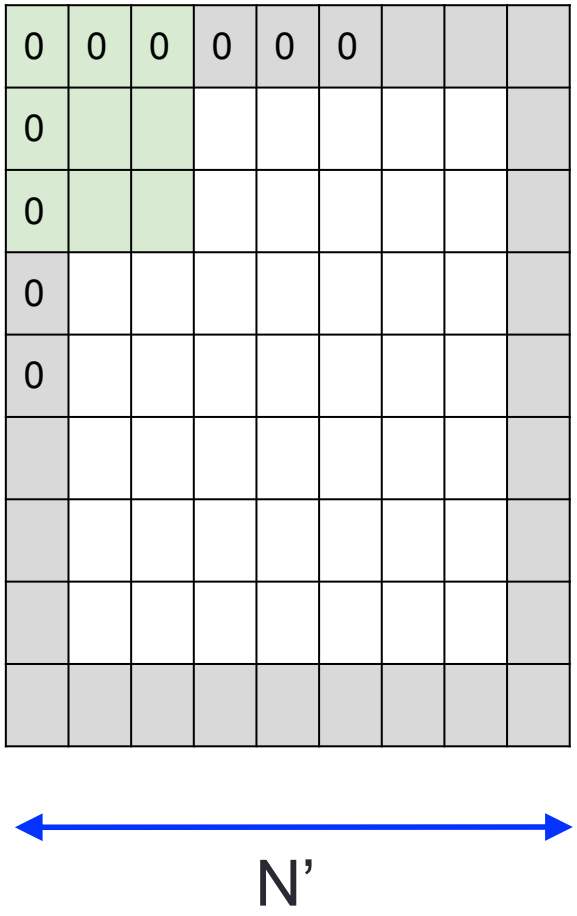
NOTE: output volume O is generally smaller than input volume and it also decreases with an increasing stride

Solution



It is common to zero-pad the border

Solution



If stride = 1, FxF filters and zero padding width (F-1)/2

We have that:

$N' = N + 2((F - 1)/2) = N + F - 1$

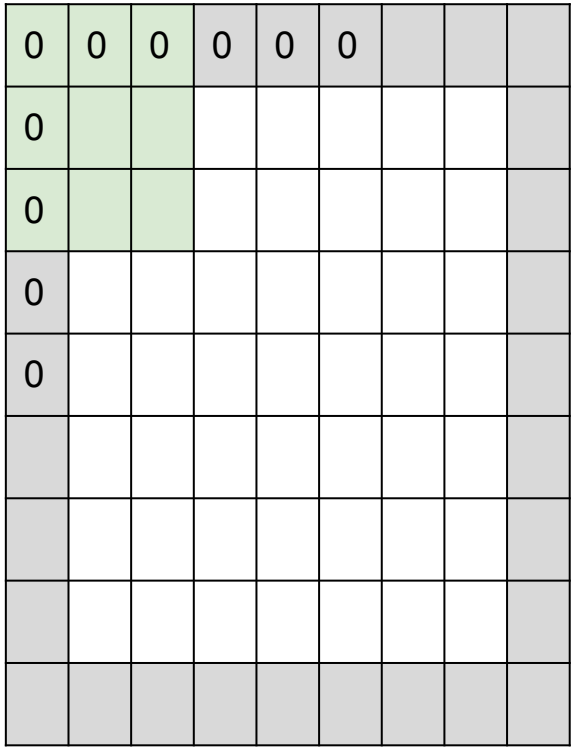
$O = (N' - F) / \text{stride} + 1$

$= (N + F - 1 - F) / 1 + 1 = N$

This means that the output size O is equal to the input size N

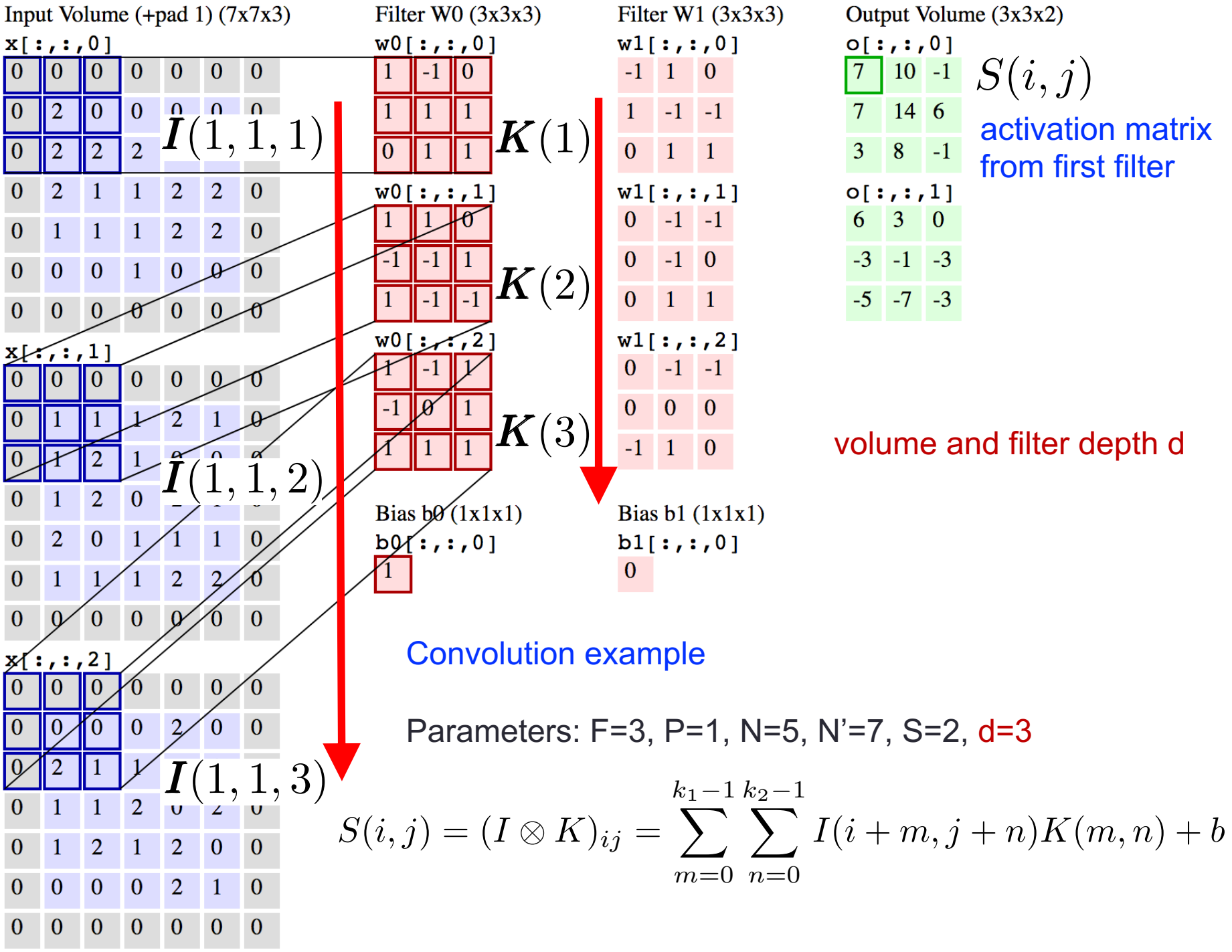
From input to output volume

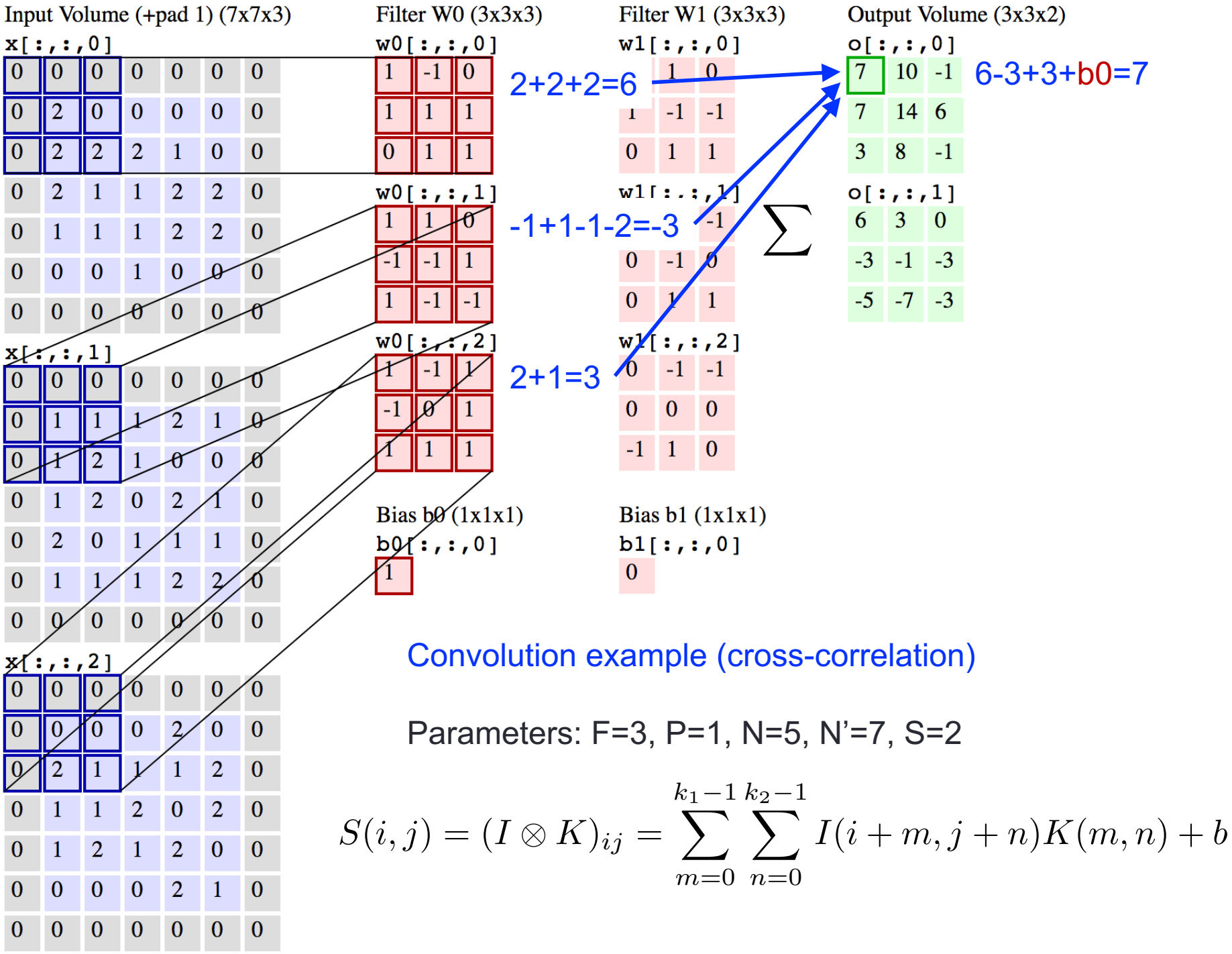
input volume



N = input width/height
F = filter size
P = padding size
S = stride
O = output width/height

$$O = \frac{N - F + 2P}{S} + 1$$

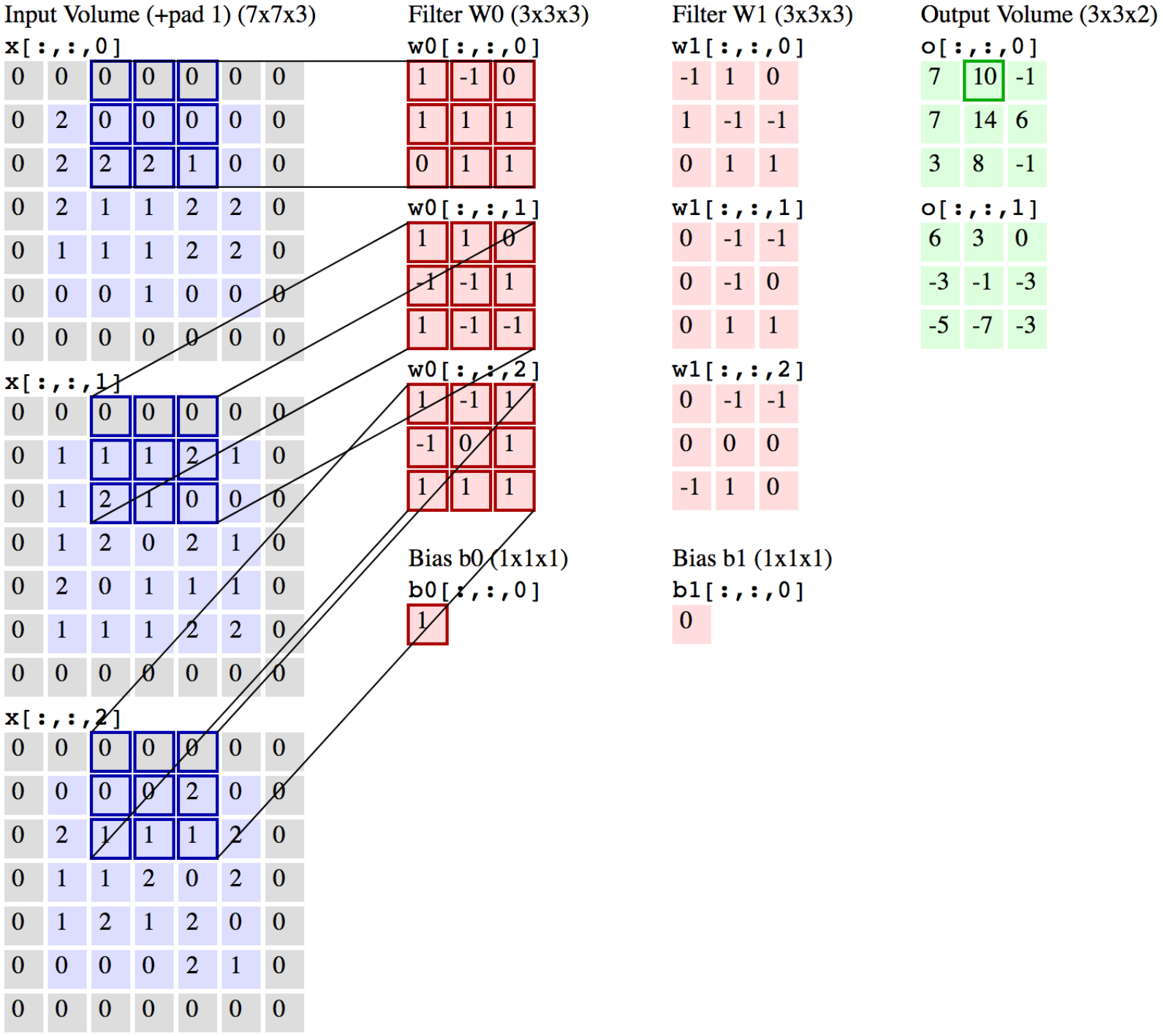

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 0 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 0 0 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 0 0 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
I(1, 1, 1)
I(1, 1, 2)
I(1, 1, 3)
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
K(1)
K(2)
K(3)
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3
S(i, j)
activation matrix
from first filter
volume and filter depth d
Convolution example
Parameters: F=3, P=1, N=5, N'=7, S=2, d=3
$$S(i,j) = (I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(m,n) + b$$

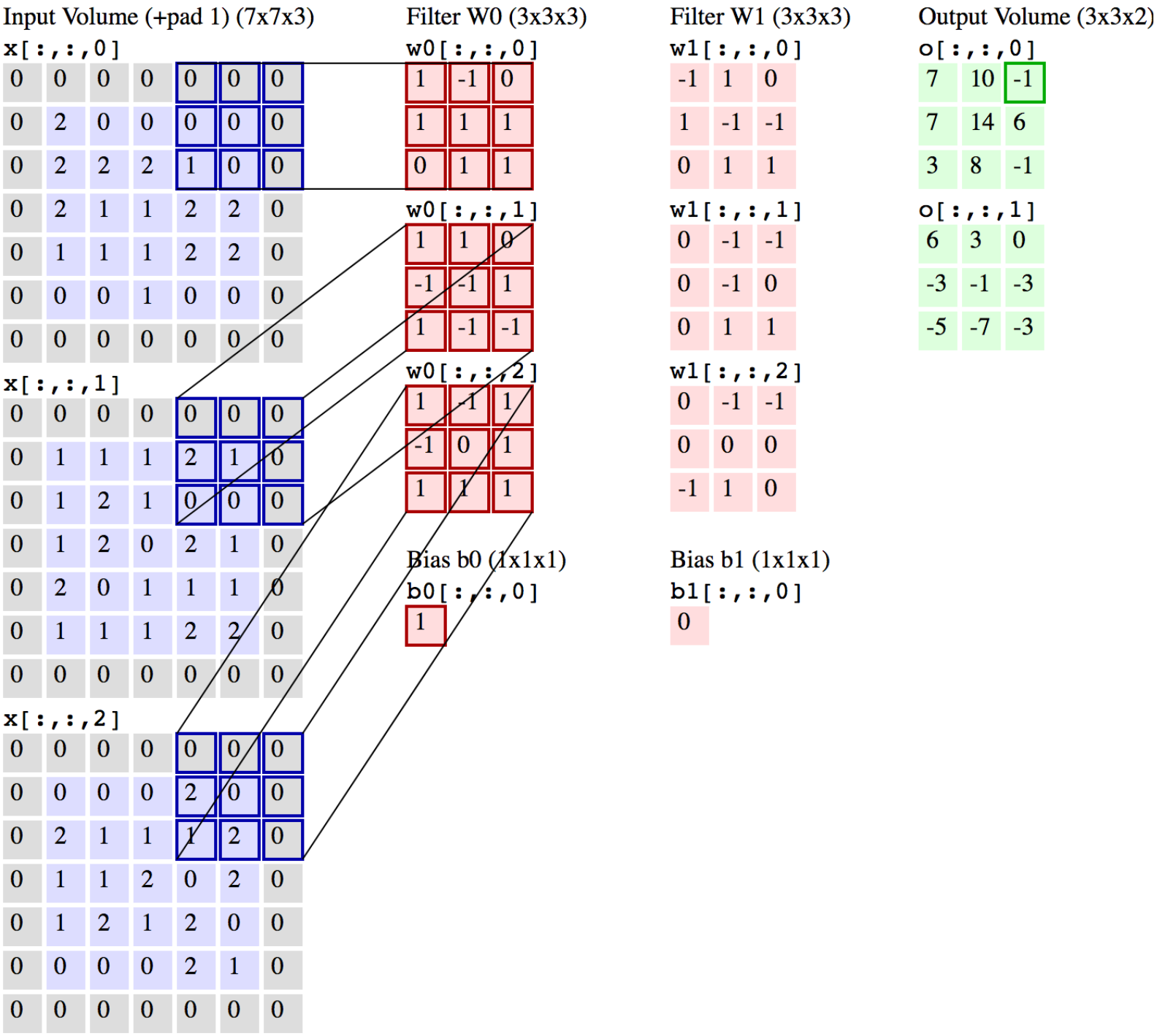


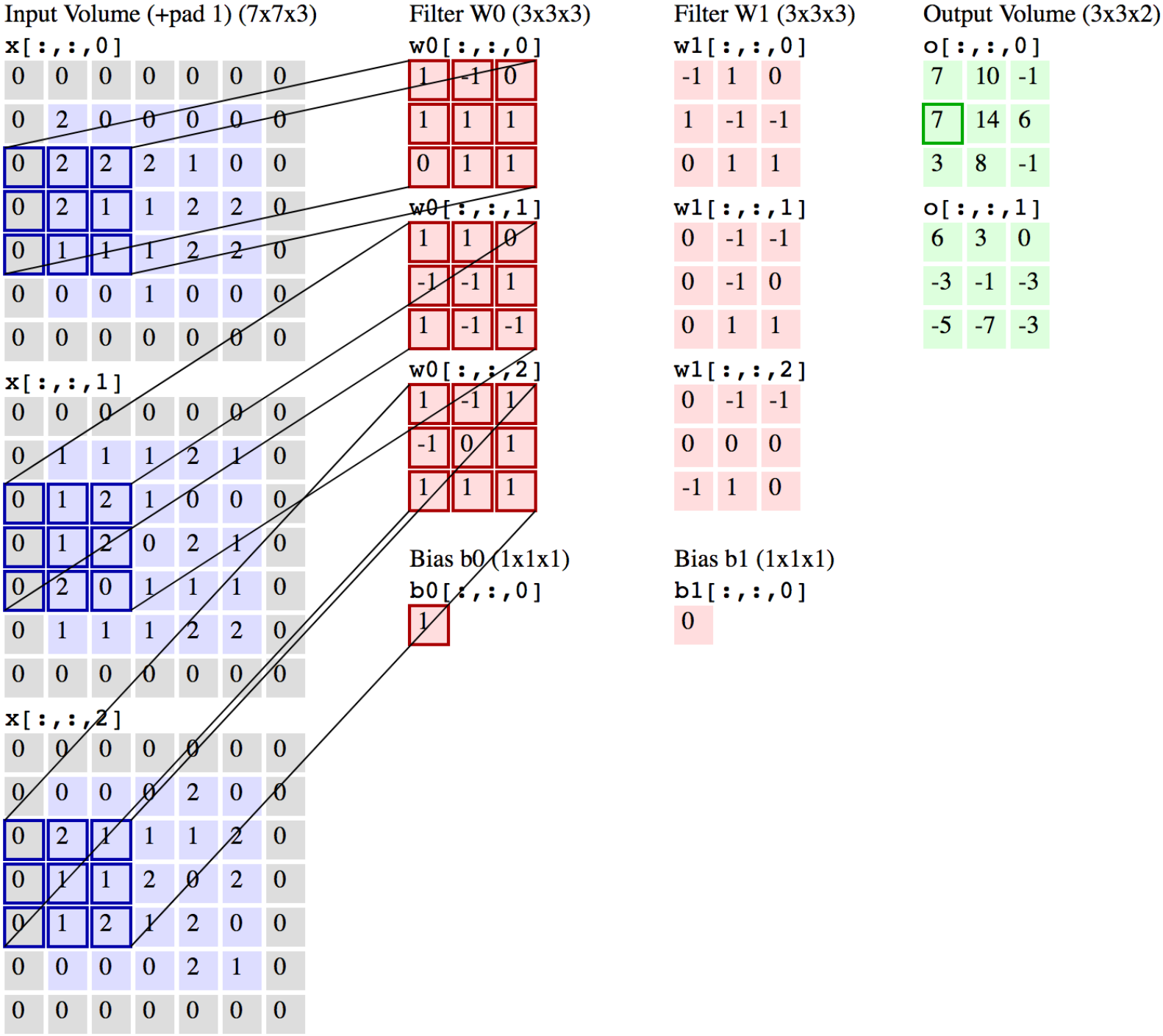
Convolution example (cross-correlation)

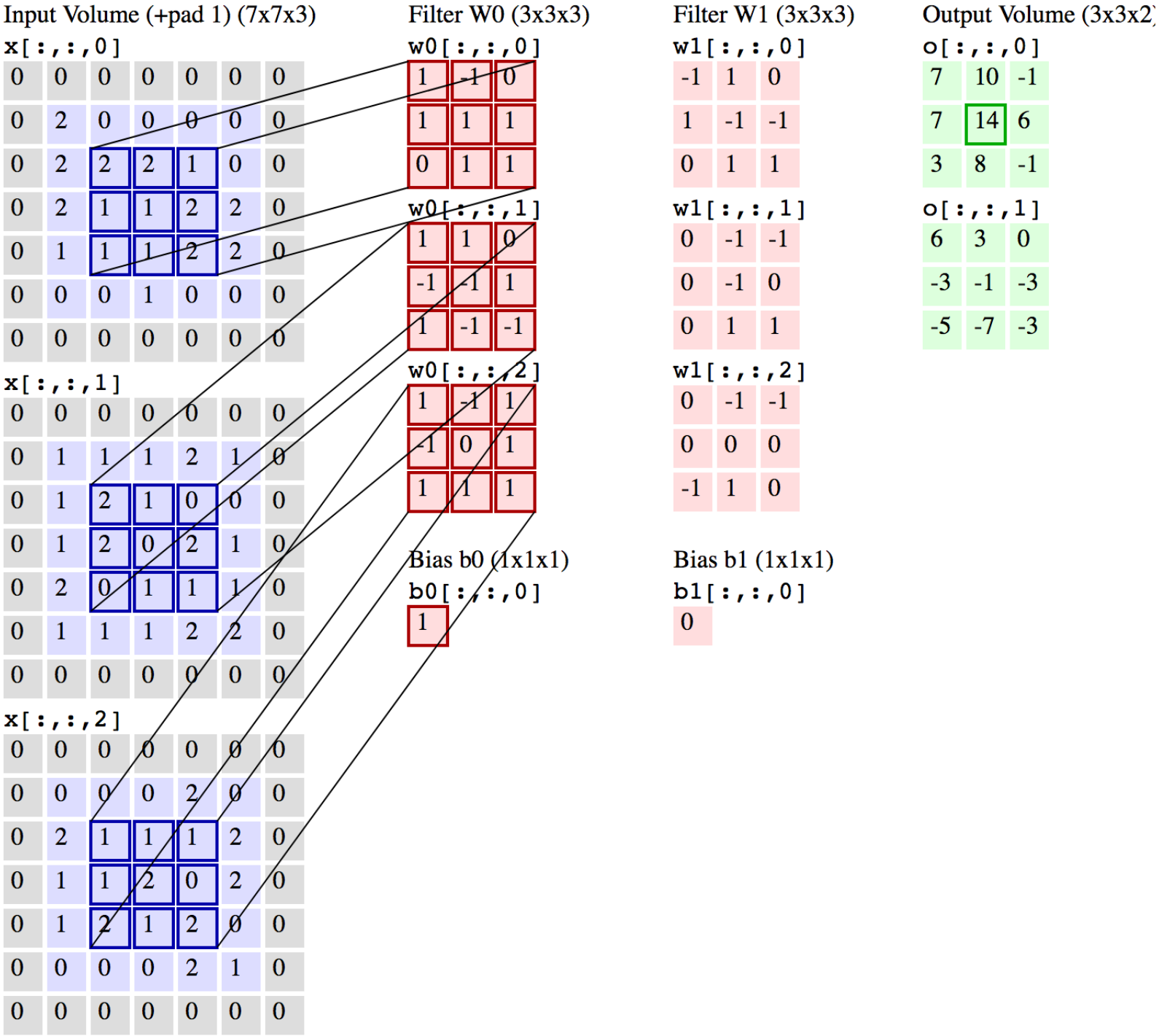
Parameters: F=3, P=1, N=5, N'=7, S=2

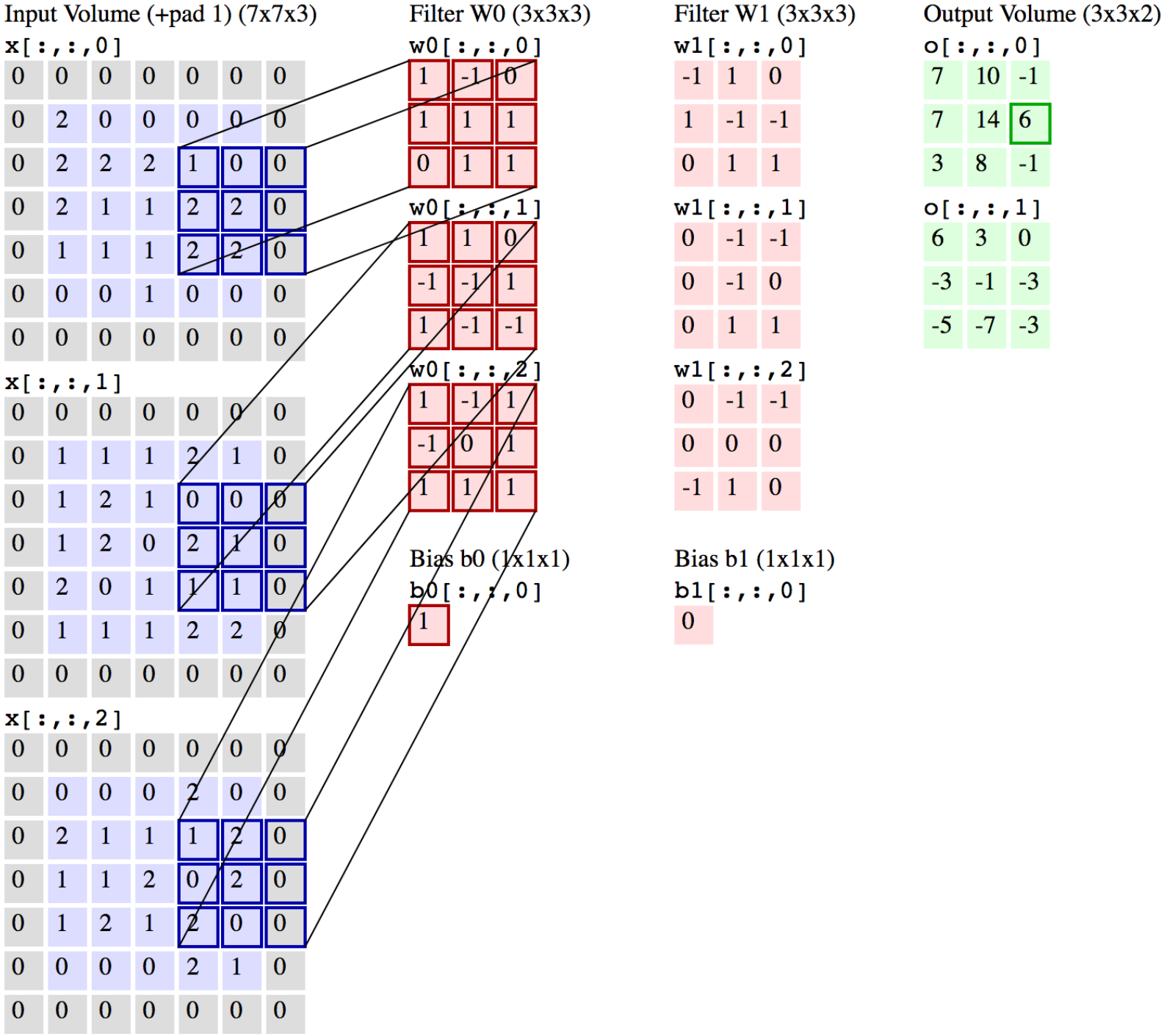
$$S(i,j) = (I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(m,n) + b$$

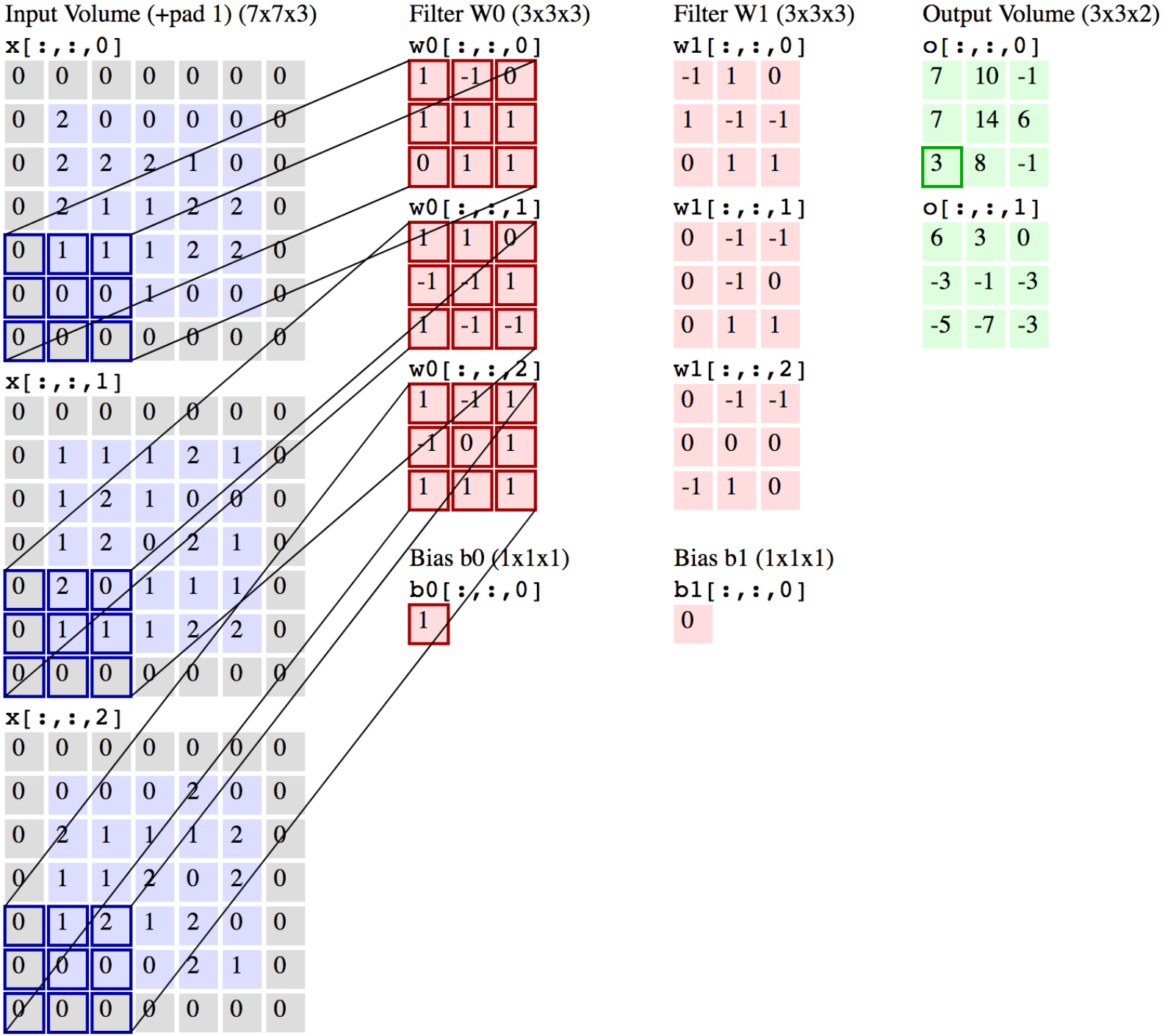

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3

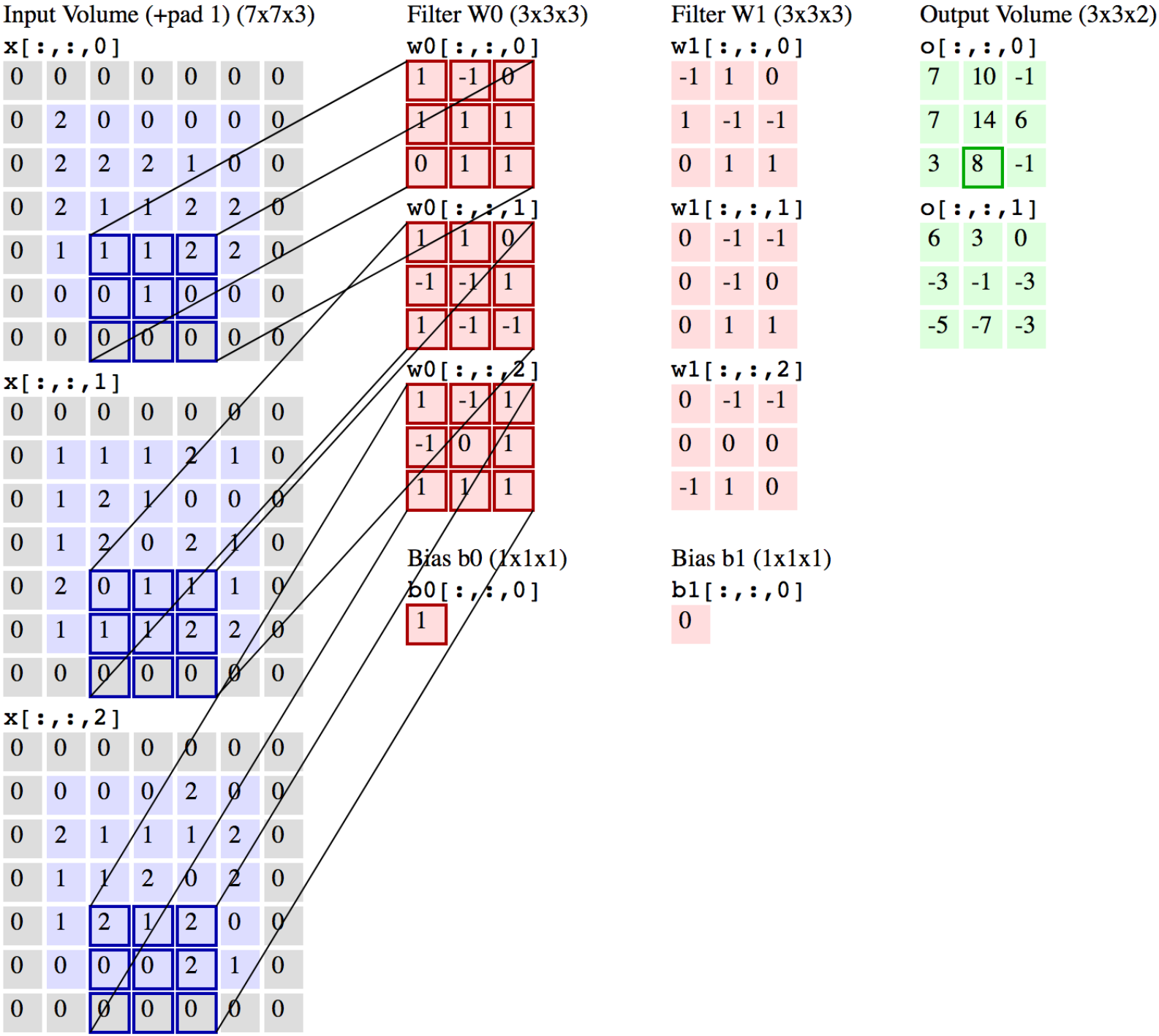

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3

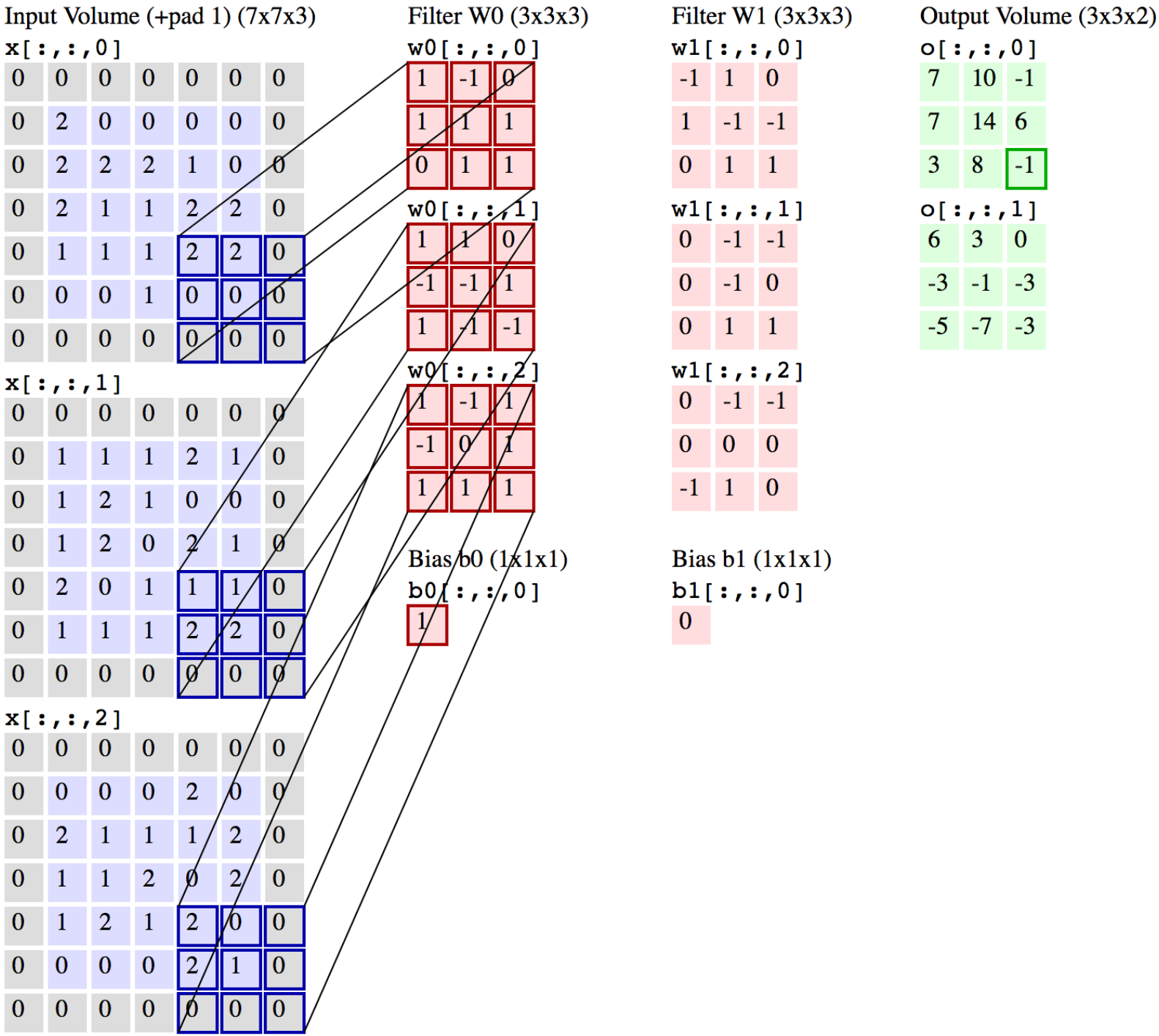

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3

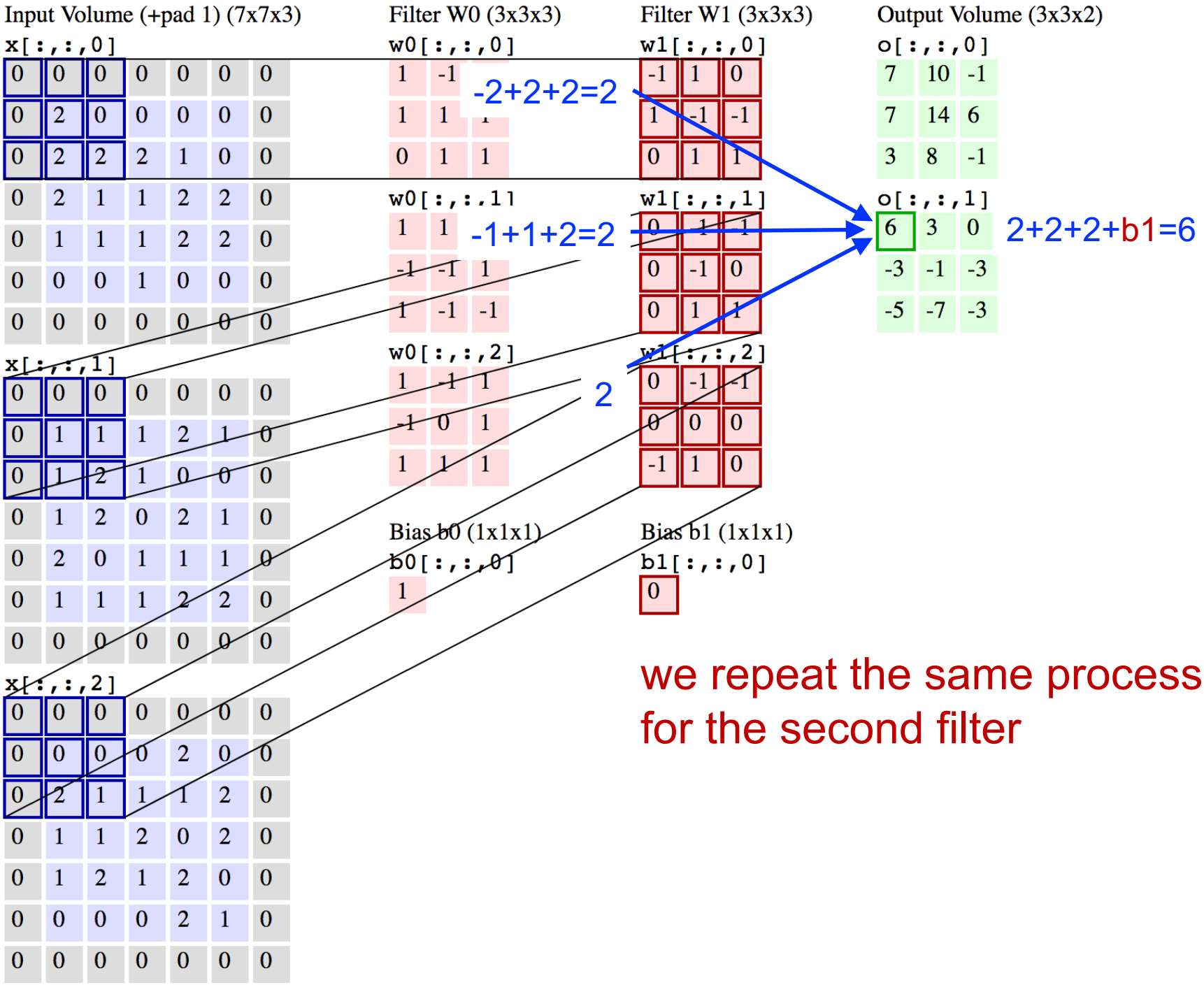

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3

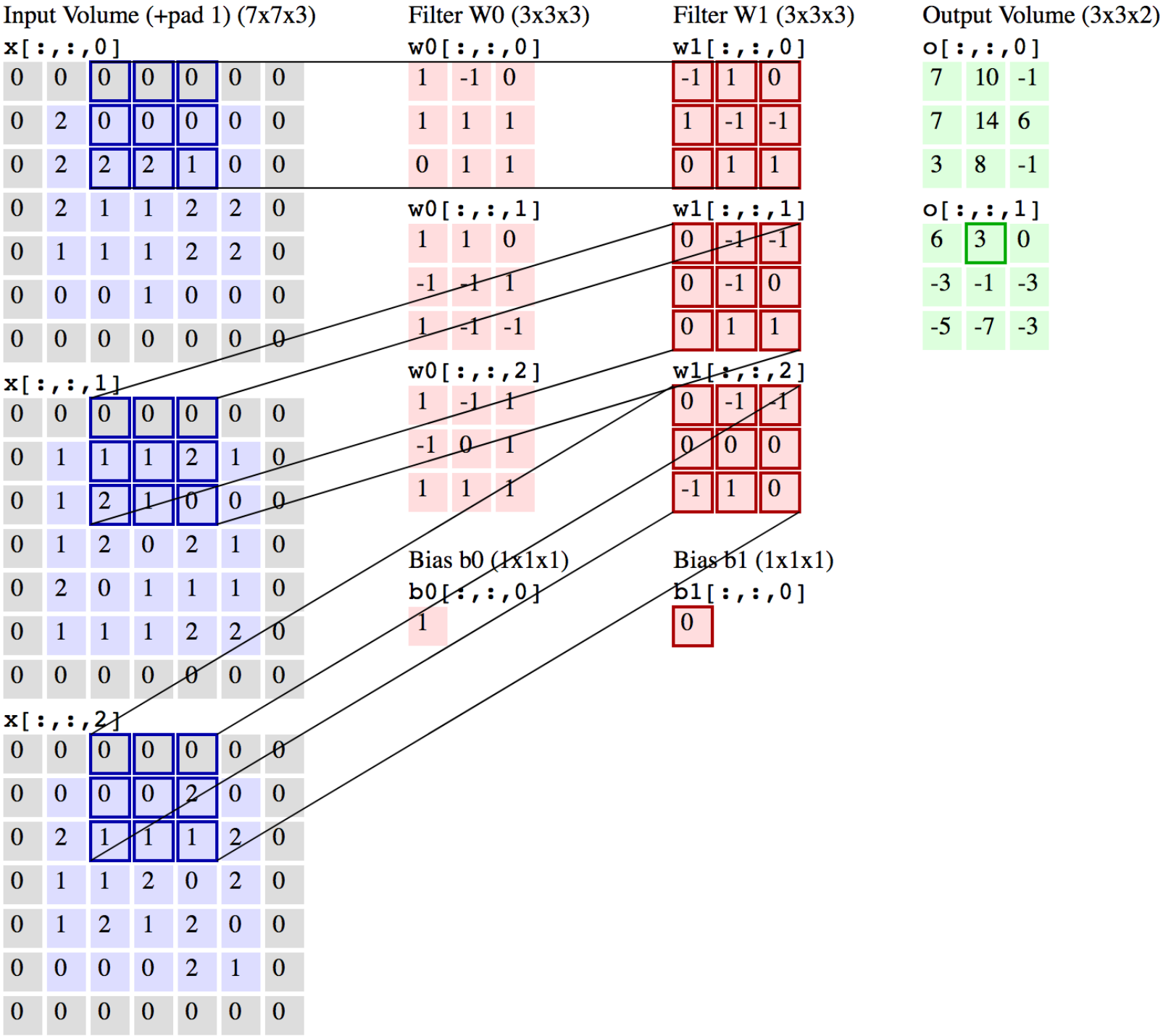

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3

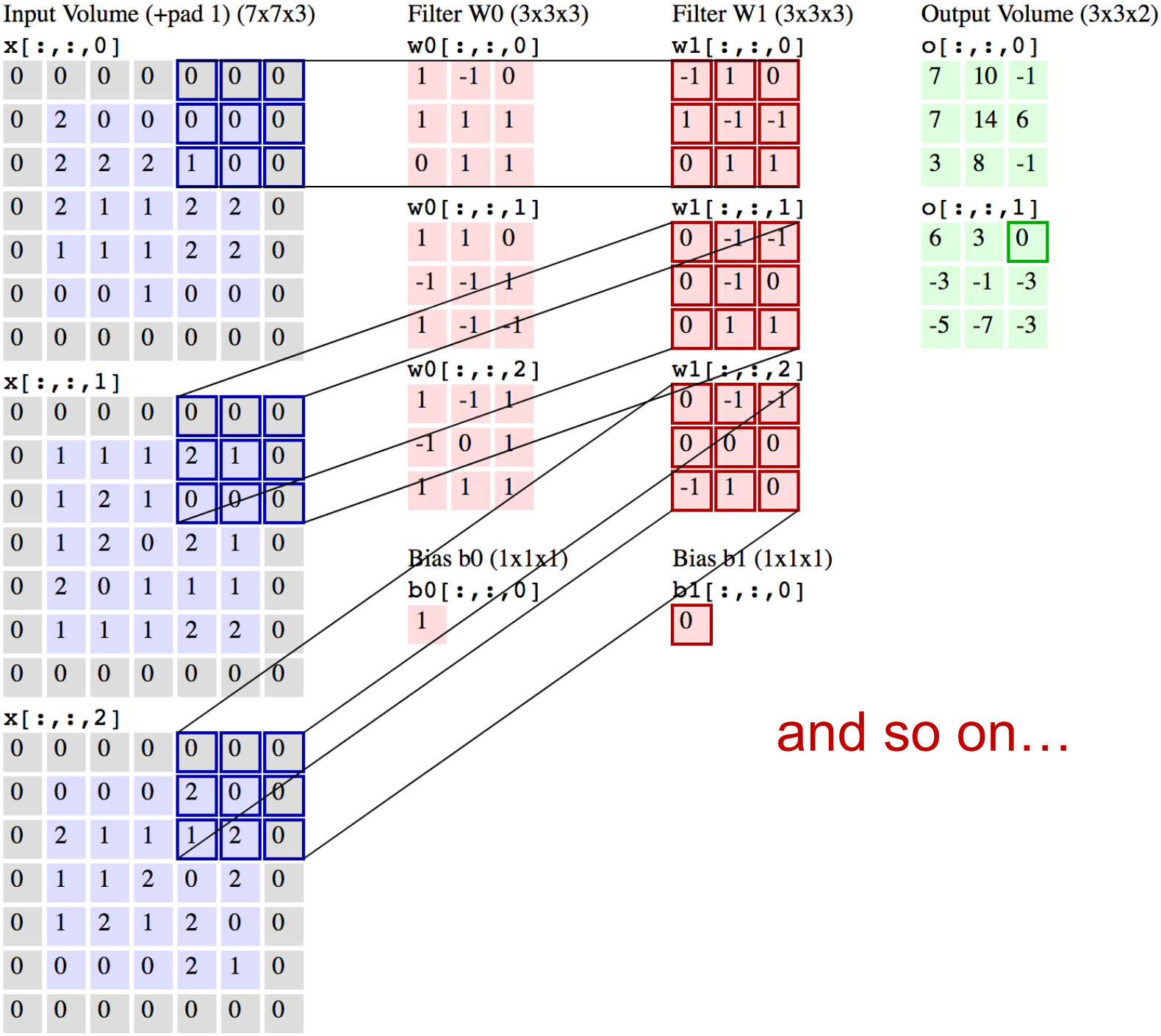

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1
1 1
0 1 1
w0[:,:,1]
1 1
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3
Bias b0 (1x1x1)
b0[:,:,0]
1
Bias b1 (1x1x1)
b1[:,:,0]
0
-2+2+2=2
-1+1+2=2
2
2+2+2+b1=6
we repeat the same process
for the second filter


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 0 0 0 0 0
0 2 2 2 1 0 0
0 2 1 1 2 2 0
0 1 1 1 2 2 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 1 0
0 1 2 1 0 0 0
0 1 2 0 2 1 0
0 2 0 1 1 1 0
0 1 1 1 2 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 0 2 0 0
0 2 1 1 1 2 0
0 1 1 2 0 2 0
0 1 2 1 2 0 0
0 0 0 0 2 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
1 1 1
0 1 1
w0[:,:,1]
1 1 0
-1 -1 1
1 -1 -1
w0[:,:,2]
1 -1 1
-1 0 1
1 1 1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 -1 -1
0 1 1
w1[:,:,1]
0 -1 -1
0 -1 0
0 1 1
w1[:,:,2]
0 -1 -1
0 0 0
-1 1 0
Output Volume (3x3x2)
o[:,:,0]
7 10 -1
7 14 6
3 8 -1
o[:,:,1]
6 3 0
-3 -1 -3
-5 -7 -3
Bias b0 (1x1x1)
b0[:,:,0]
1
Bias b1 (1x1x1)
b1[:,:,0]
0
and so on…

CNN hyper-parameters

- Depth
  - It corresponds to the number of filters we would like to use
  - Each filter looks for something different in the input
- Stride
  - Controls the way in which the filter convolves around the input volume
  - Normally larger strides if
    - We want receptive fields (see shortly) to overlap less
    - We want to decrease the spatial dimension
- Zero-padding
  - Main reason for using it is to preserve volume size from input to output
  - Without, the volume size would reduce too fast → not good to build deep networks (with many layers)

Characteristics of CNNs

- CNNs rest upon 3 distinguishing features

1) Sparse interactions
2) Parameter sharing
3) Equivalent representations

Sparse interactions

- Traditional FFNNs
  - Connectivity structure is dense: from one layer to the next, there is a link (weight) connecting each neuron in the previous layer to each neuron in the next one, this means that the structure is fully connected
  - This means that each input neuron interacts with each output one
- CNNs
  - Connectivity structure is sparse: this is accomplished by making the size of the Kernel smaller than that of the input
  - Example: an image may have thousands or millions of pixels, but we may detect small meaningful features, such as edges, using Kernels that occupy only tens or hundreds of pixels
  - Small Kernels mean: storing fewer parameters (weights), which also means *higher energy* and *computation efficiencies*

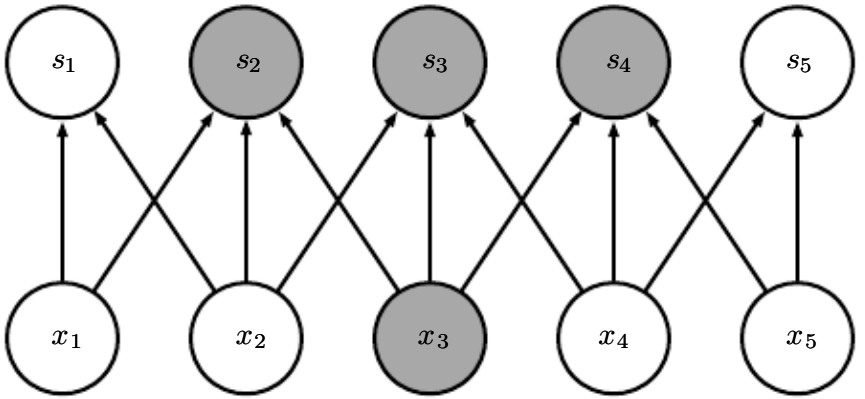
Sparse interactions

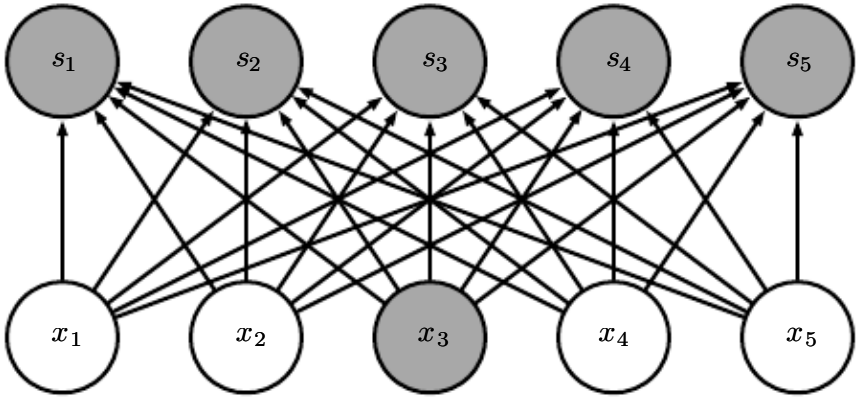
- Traditional FFNNs
  - Connectivity structure is dense: from one layer to the next, there is a link (weight) connecting each neuron in the previous layer to each neuron in the next one, this means that the structure is fully connected
  - This means that each input neuron interacts with each output one
- CNNs
  - Connectivity structure is *sparse*: this is accomplished by making the size of the Kernel smaller than that of the input
  - Example: an image may have thousands or millions of pixels, but we may detect small meaningful features, such as edges, using Kernels that occupy only tens or hundreds of pixels
  - Small Kernels mean: storing fewer parameters (weights), which also means *higher energy* and *computation efficiency*

Sparse connectivity (1/2)

- 1D convolution example
  - Sparse connectivity viewed *from below*: we highlight one input unit x_3 and also highlight the output units s_i that are affected by this input
  - Here: S=1, F=3

CNN

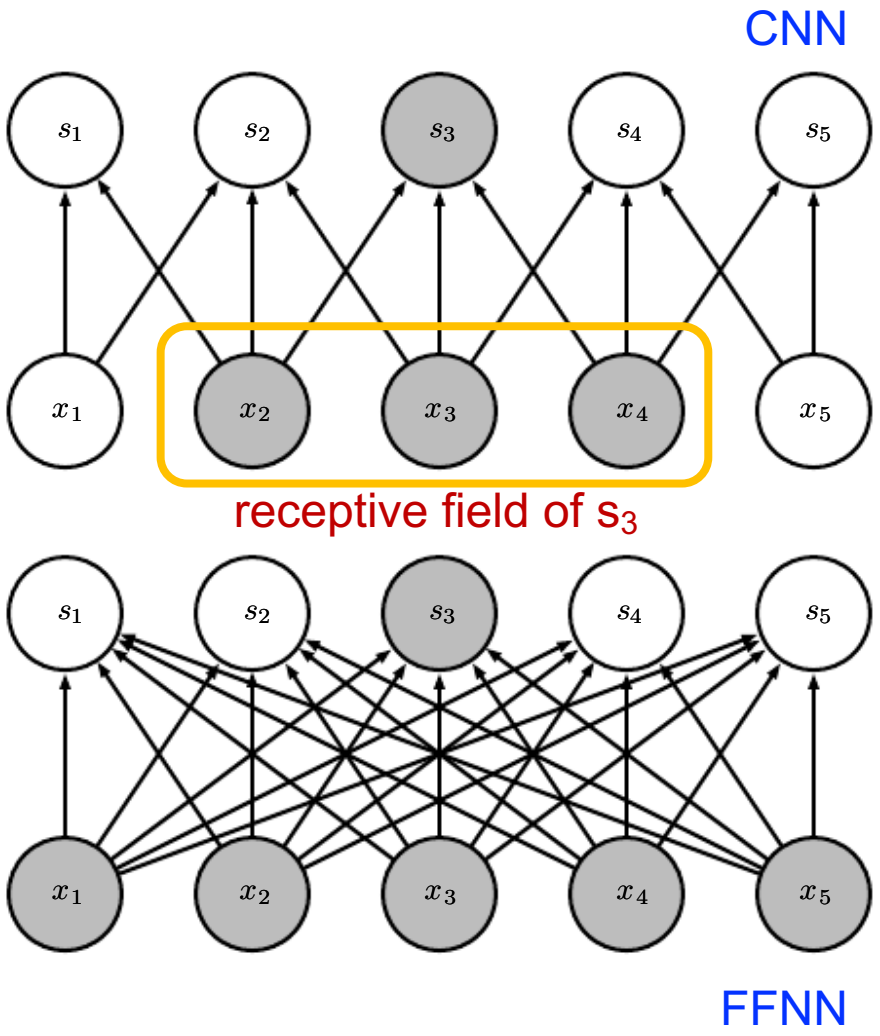




FFNN

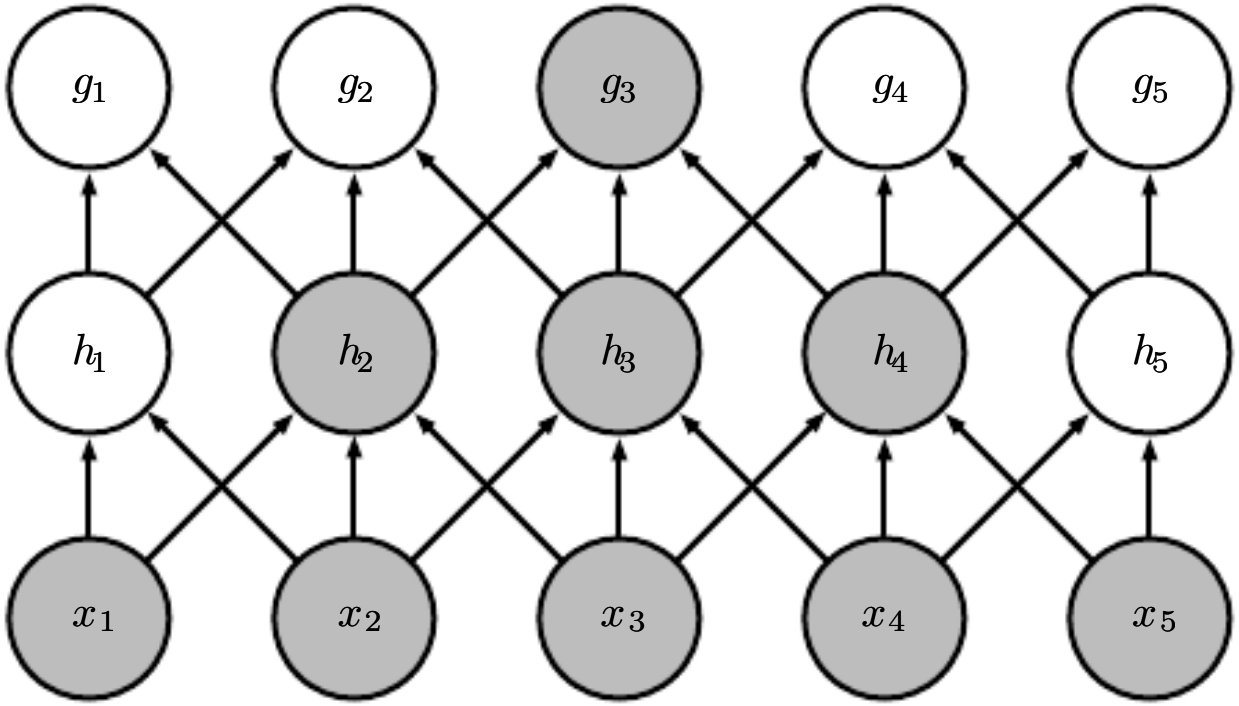
Sparse connectivity (2/2)

- 1D convolution example
  - Sparse connectivity viewed *from above*: we highlight one output unit s_3 and also highlight the input units x_i that affect it
  - These input units are also called the receptive field of s_3



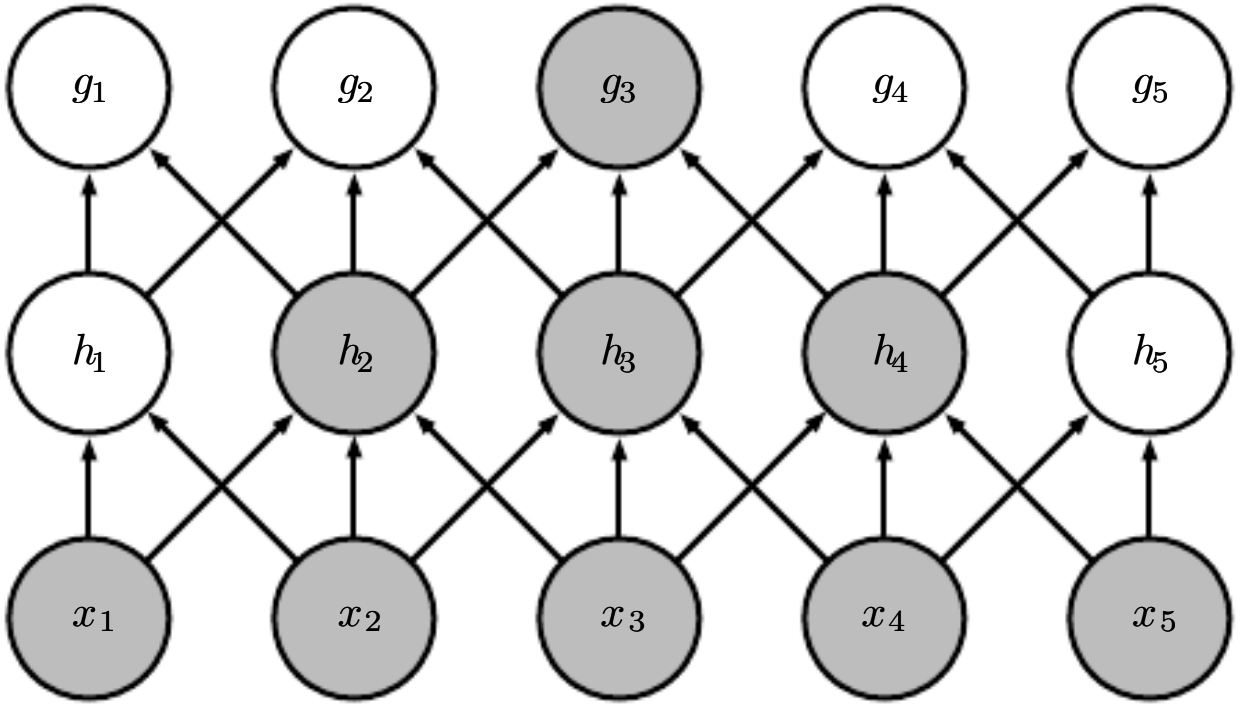
Receptive fields in deep networks (1/2)

- The receptive field of the units in the deeper layers (top of the diagram) of a CNN is larger than the receptive field of the units in the shallow layers (bottom)



Receptive fields in deep networks (2/2)

- This means that, although connectivity in a CNN is sparse, units in the deepest layer can still be connected to most (or all) the points in the input volume



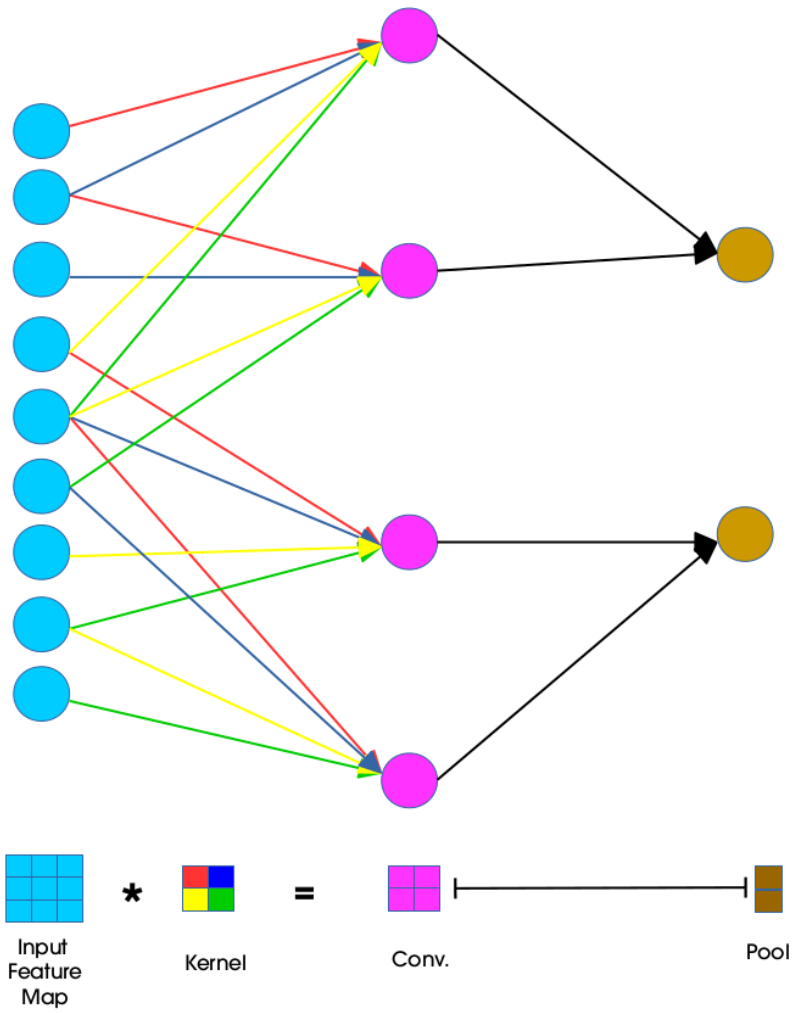
Parameters sharing

- Refers to using the same parameter for more than one function in a model
- In a standard FFNN
  - Each parameter (weight) is used only once to compute the corresponding output, and then it is never re-used for the current pass
- In a CNN
  - Each elements of the Kernel is applied (used) with each position in the input (except for some boundary points, depending on the design decisions for the boundaries)
  - Parameter sharing: means that instead of learning a set of parameters for every input location, we learn a single set of parameters (Kernel) and apply them to all input locations, by “moving the Kernel around the input”

Parameters sharing

- Refers to using the same parameter for more than one function in a model
- In a standard FFNN
  - Each parameter (weight) is used only once to compute the corresponding output, and then it is never re-used for the current pass
- In a CNN
  - Each elements of the Kernel is applied (used) with each position in the input (except for some boundary points, depending on the design decisions for the boundaries)
  - Parameter sharing: means that instead of learning a set of parameters for every input location, we learn a single set of parameters (Kernel) and apply them to all input locations, by “moving the Kernel around the input”

Parameters sharing



Hyper-parameters

Input: 3x3 (N=3)
Kernel: 2x2 (F=2)
Stride = 1 (S=1)
Output: 2x2 (O=2)

Input and Output unfolded for clarity

Same Kernel parameters:
- red, blue, yellow and green lines used to compute all the output values
- violet filled circles

Equivalent representations

- Equivariance
  - It is a mathematical property assessing that, given two functions g() and f(), the following relation holds:

$$f(g(x)) = g(f(x))$$

- Convolution is equivariant to translations
  - If g() is any function that translates the input signal (i.e., shifts it)
  - The convolution function f() is equivariant to g()
  - Example:
    - let I() be a function giving image brightness at integer coordinates
    - let g() be another function mapping I() into I'(), as I'(x,y) = I(x-1,y) (shifts every pixel of I() one bit to the right). If we apply this transformation to I() and then apply convolution to I'(), the result will be the same as if we applied convolution to I() and then applied g() to this output

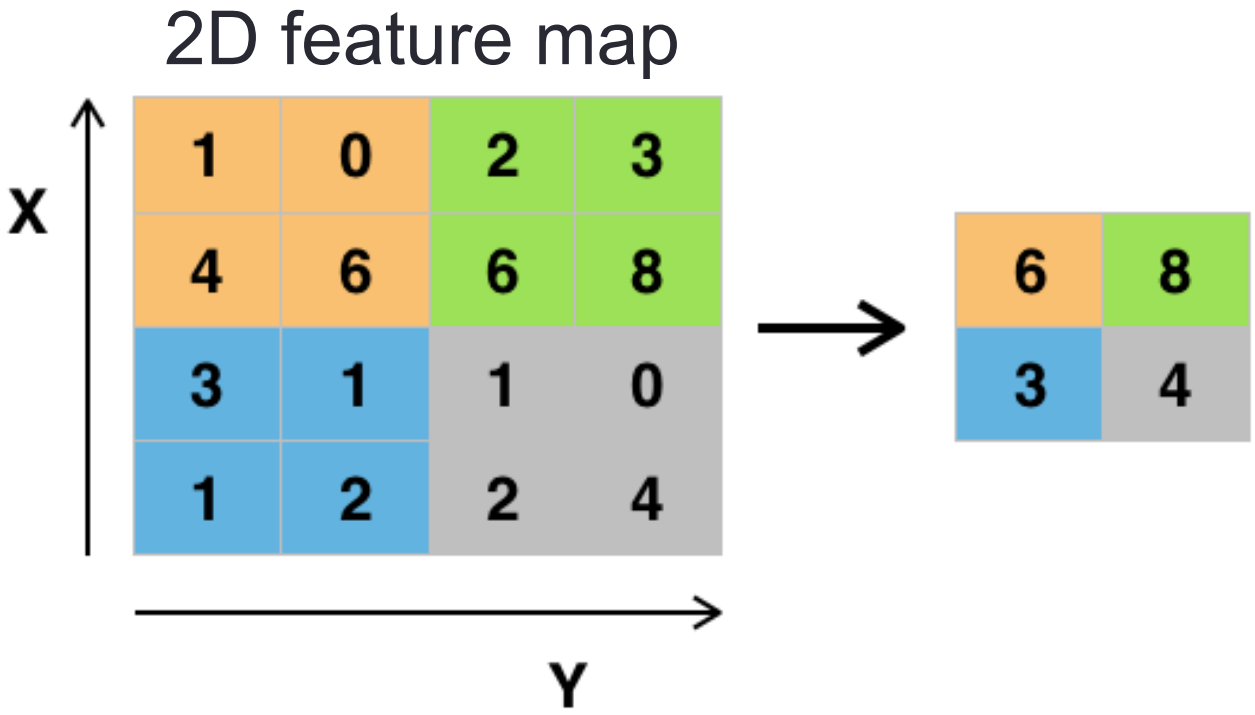
Equivalent representations

- Implications for time series (1D signals)
  - Equivariance means that convolution produces a sort of time line that shows when different features appear in the input
  - If we move an event later in time in the input, the exact same representation of it will appear in the output, just later in time
- Implications for images (2D signals)
  - Convolution creates a 2D map of where certain features appear in the input
  - If we move a certain object in the input, its representation will move by the same amount in the output
  - This is useful for, e.g., when we know that some function of a small number of neighboring pixels has to be applied to multiple input locations. For example, when processing images, it is useful to detect edges in the first layer of a CNN
  - NOTE: convolution is not naturally equivariant to rotations and scaling

Equivalent representations

- Implications for time series (1D signals)
  - Equivariance means that convolution produces a sort of time line that shows when different features appear in the input
  - If we move an event *later in time* in the input, the exact same representation of it will appear in the output, just later in time
- Implications for images (2D signals)
  - Convolution creates a 2D map of where certain features appear in the input
  - If we move a certain object in the input, its representation will move by the same amount in the output
  - This is useful for, e.g., when we know that some function of a small number of neighboring pixels has to be applied to multiple input locations. For example, when processing images, it is useful to detect edges in the first layer of a CNN
  - NOTE: convolution is not naturally equivariant to rotations and scaling

Pooling example: max pooling



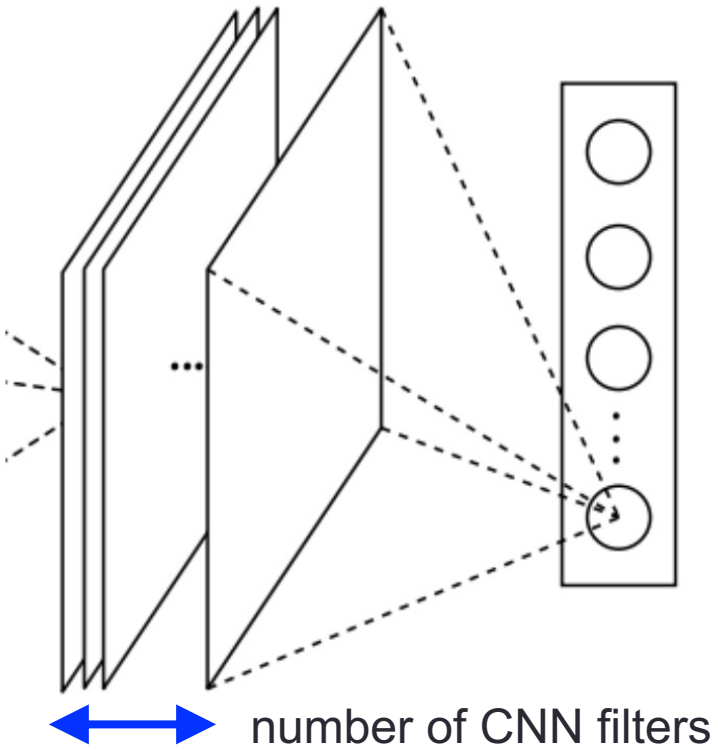
Example of maxpool with a 2x2 filter and a stride of 2

Pooling example: global average pooling

- Given a feature vector (1D) or a feature map (2D)
- Averages the values in it and returns a single value
- 2D CNN: returns a single value for each 2D feature map
  - In a 2D CNN → 1 map per filter

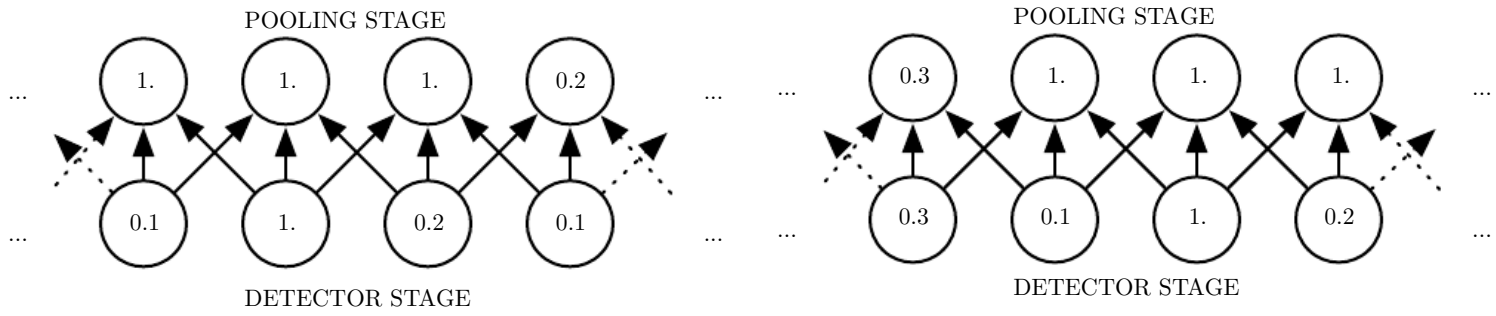
Main advantages

- No weights are to be trained
- Does not depend on ordering
- Output: order independent feature vector



Pooling advantages

- Pooling helps in all cases
  - To become invariant to small translations in the input
  - This means that, if we translate the input of a small amount, the value of the pooled output does not change much
  - Variance to translation is very important: if we care about whether a feature (e.g., human eyes in an image) is present rather than where it is



DETECTOR STAGE: shows the output of the ReLU non-linearity. Max pooling region width is of three pixels. Stride (movement) is one pixel to the left for the values in the detector layer: *every value in the bottom row has changed*, but *only half of those in the top row have changed* due to max pooling, which is only sensitive to max value in the neighborhood and not to its location

Three stages of a typical CONV layer

- A typical layer in a CNN consists of three stages
  - **First stage (convolution):** a layer performs several convolutions in parallel to produce a set of (linear) activations (i.e., the output of the convolution operation that we have seen previously)
  - **Second stage (detection):** each linear activation is run through a nonlinear activation function, such as the rectified linear activation function (ReLU)
  - **Third stage (pooling):** to modify the output of the second stage further. A pooling function replaces the output of a net at a certain location with a summary statistics of the nearby outputs. For example the max pooling function reports the maximum output within a rectangular neighborhood
    - other popular pooling functions: average over a rectangular neighborhood, the L_2 norm of a rectangular neighborhood or a weighted average based on the distance from the central point (pixel)

Three stages of a typical CONV layer

- A typical layer in a CNN consists of three stages
  - **First stage (convolution):** a layer performs several convolutions in parallel to produce a set of (linear) activations (i.e., the output of the convolution operation that we have seen previously)
  - **Second stage (detection):** each linear activation is run through a nonlinear activation function, such as the rectified linear activation function (ReLU)
  - **Third stage (pooling):** to modify the output of the second stage further. A pooling function replaces the output of a net at a certain location with a summary statistics of the nearby outputs. For example the max pooling function reports the maximum output within a rectangular neighborhood
    - other popular pooling functions: average over a rectangular neighborhood, the L_2 norm of a rectangular neighborhood or a weighted average based on the distance from the central point (pixel)

Three stages of a typical CONV layer

- A typical layer in a CNN consists of three stages
  - **First stage (convolution):** a layer performs several convolutions in parallel to produce a set of (linear) activations (i.e., the output of the convolution operation that we have seen previously)
  - **Second stage (detection):** each linear activation is run through a nonlinear activation function, such as the rectified linear activation function (ReLU)
  - **Third stage (pooling):** to modify the output of the second stage further. A pooling function replaces the output of a net at a certain location with a summary statistics of the nearby outputs. For example the max pooling function reports the maximum output within a rectangular neighborhood
    - other popular pooling functions: average over a rectangular neighborhood, the L_2 norm of a rectangular neighborhood or a weighted average based on the distance from the central point (pixel)

CONV layer

Left

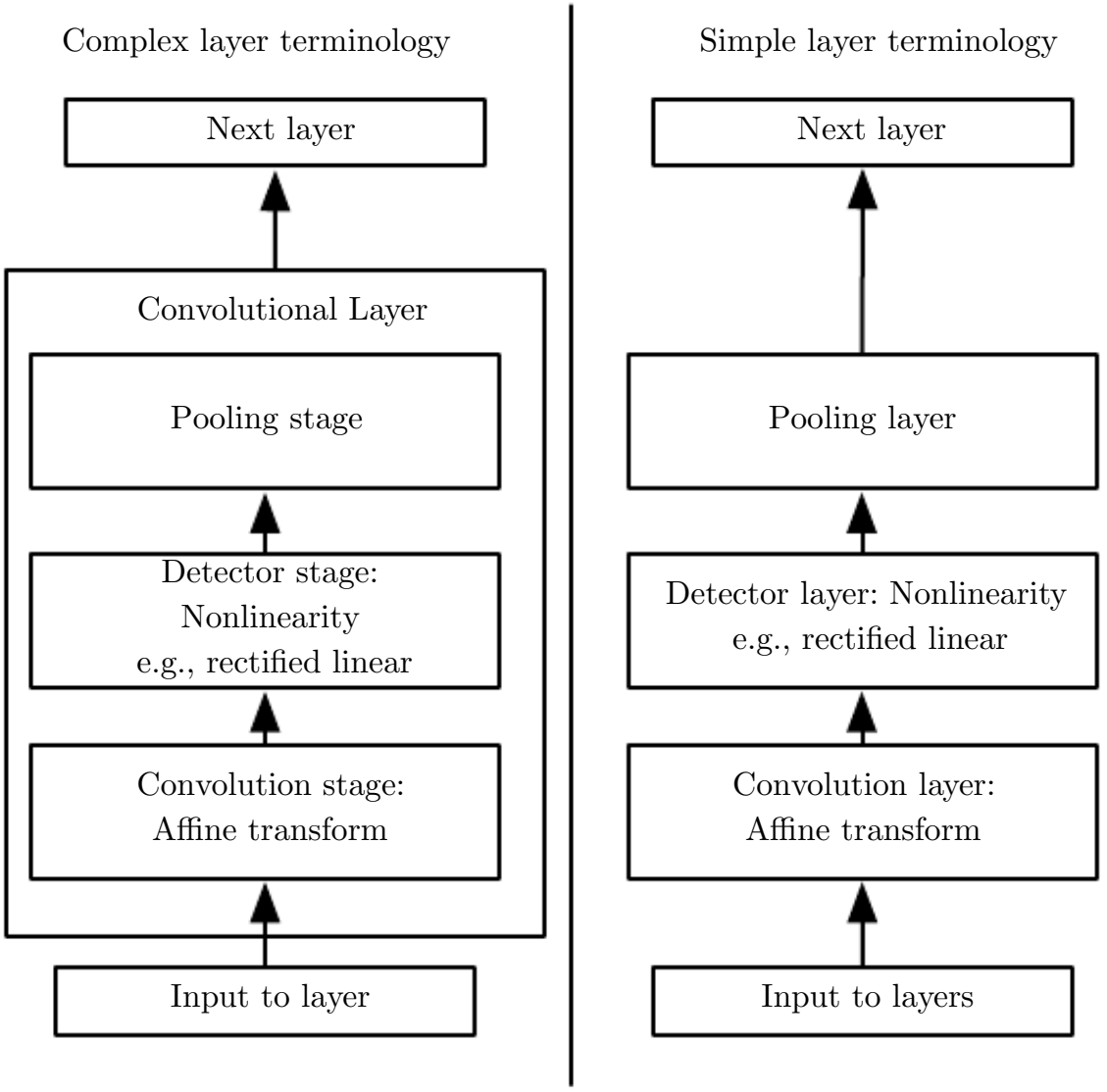
- common
- small no. of complex layers

Right

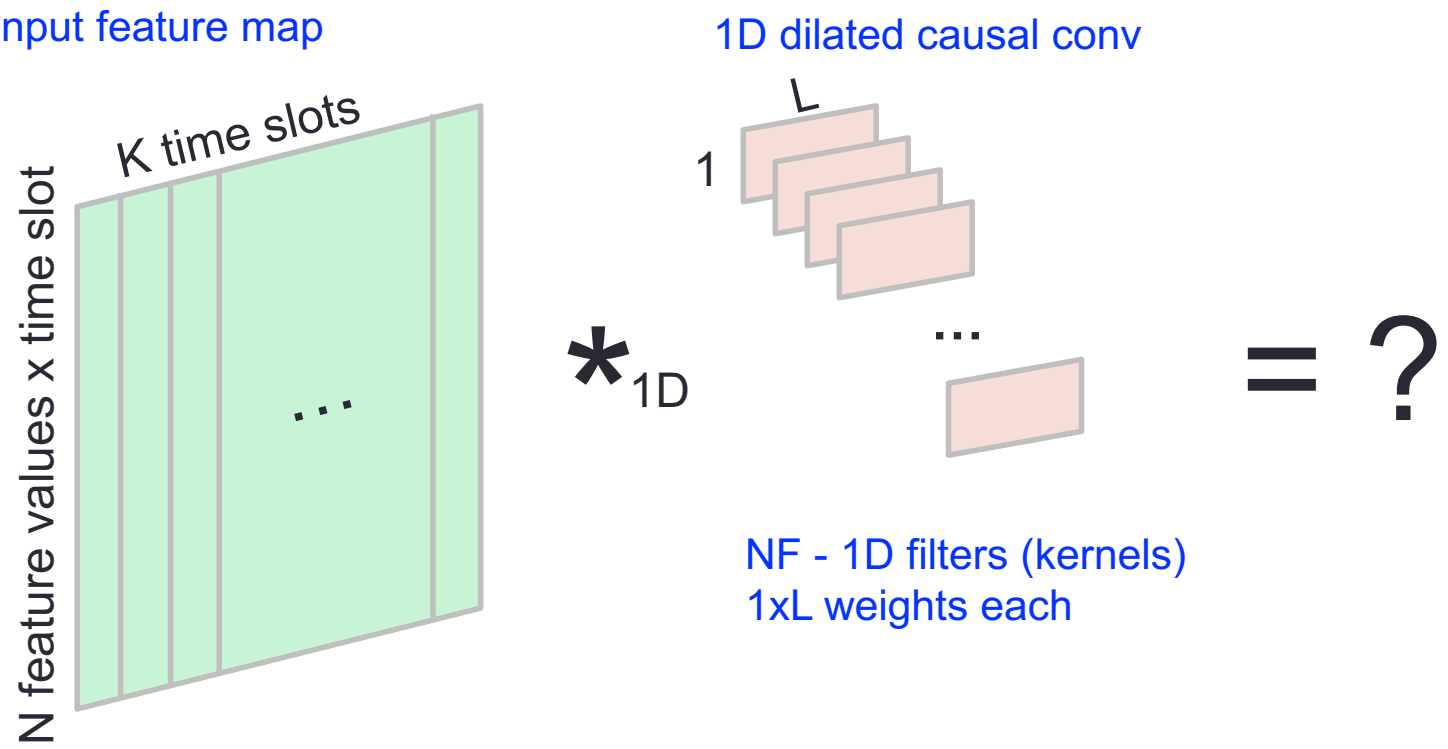
- less common
- Large no. of simple layers

Pooling

- May or may not be included after the nonlinearity



Temporal causal dilated 1D convolution



Input: N feature vectors

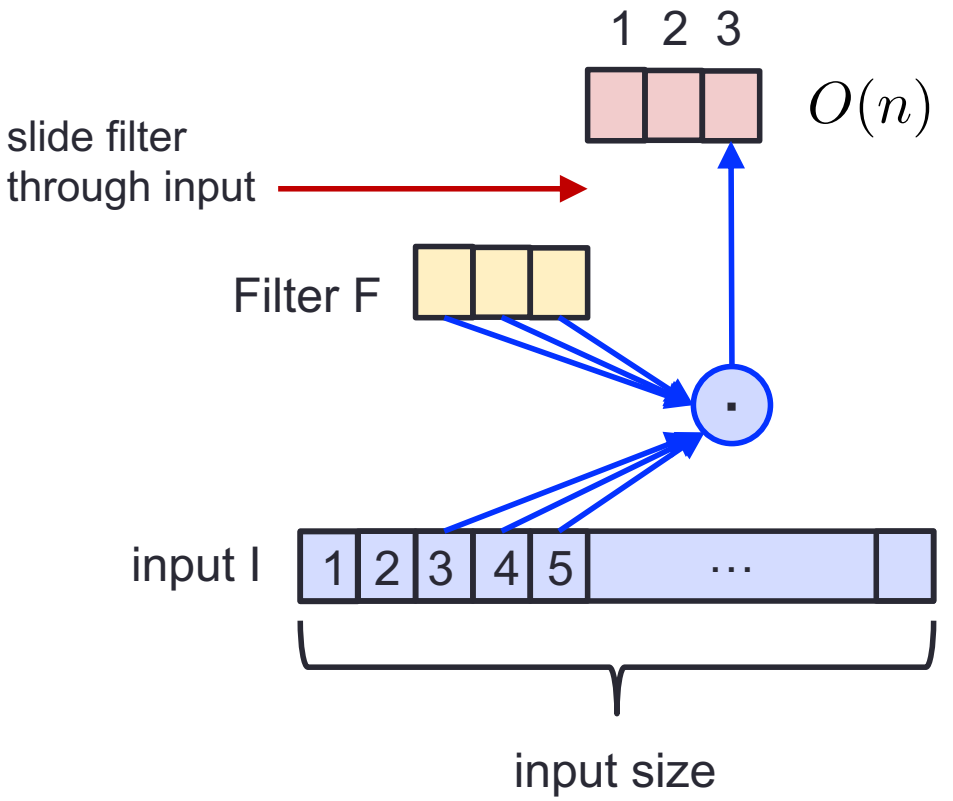
- One vector per time slot for K time slots

1D convolution

Example: kernel of 3 elements (kernel size, KS = 3)

1) slide filter (**F**, or kernel) through the input (**I**)

2) dot product of kernel weights and elements in the input



$$O(n) = \boldsymbol{F} \cdot \boldsymbol{I}_n = \sum_{j=0}^{KS-1} F(j) I(j+n)$$

Note: output at time 3 here depends on input entries at time 3, 4, 5

Length of outpus seq = length of input seq - 2

Causality

- Desirable property of a forecasting model
  - the value of a specific entry in the output depends on all previous entries in the input, i.e. all entries that have an index smaller or equal to itself
  - should not depend on the values of the input that follow

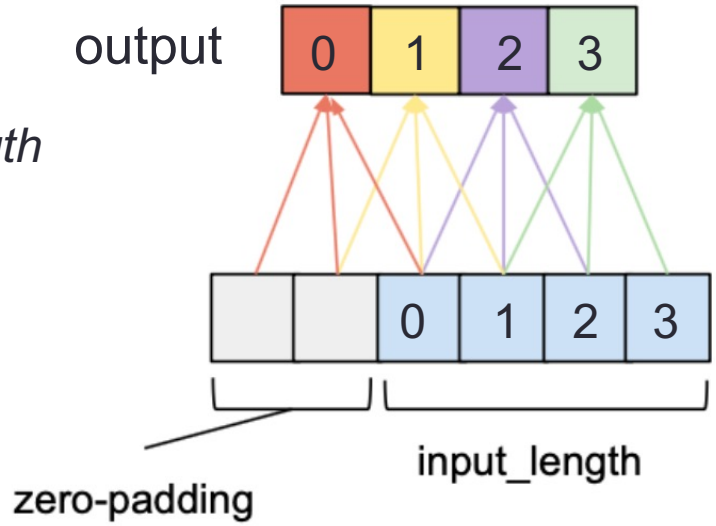
1D causal convolution

For every n in [0, …, input_length-1] the n-th element of the output sequence may only depend on the elements of the input sequence with indices [0, …, n]

Or, an element in the output sequence can only depend on elements that come before in the input sequence

Solution: Zero padding on the left-side of the input seq.

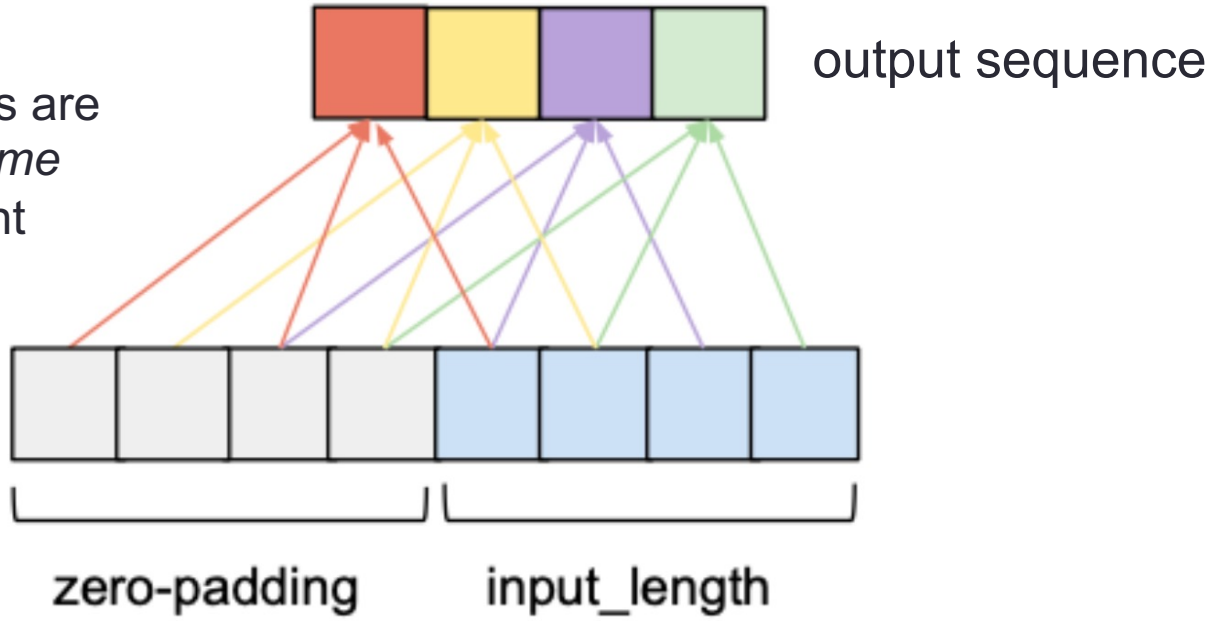
Zero padding leads to *same length* for input and output sequences



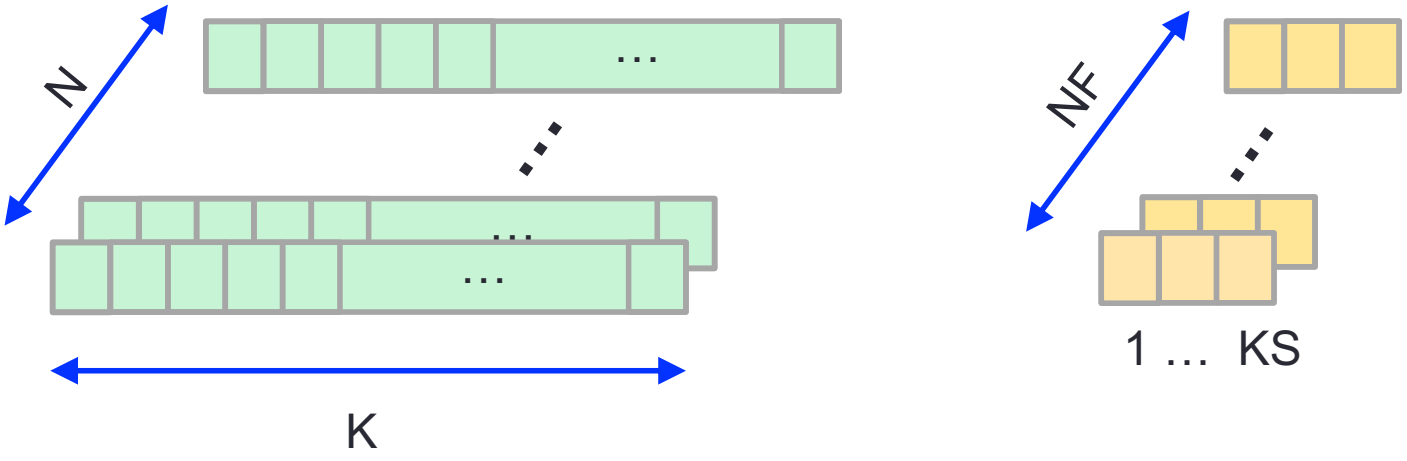
Dilation with causality

Dilation in the context of a convolutional layer refers to the **distance** between the elements of the input sequence that are used to compute one entry of the output sequence. The following figure shows an example of a 2-dilated layer (dilation rate = 2) with input_length=4, kernel_size=3

(shared) filter weights are indicated with the *same color* here on the right



Temporal causal dilated convolution



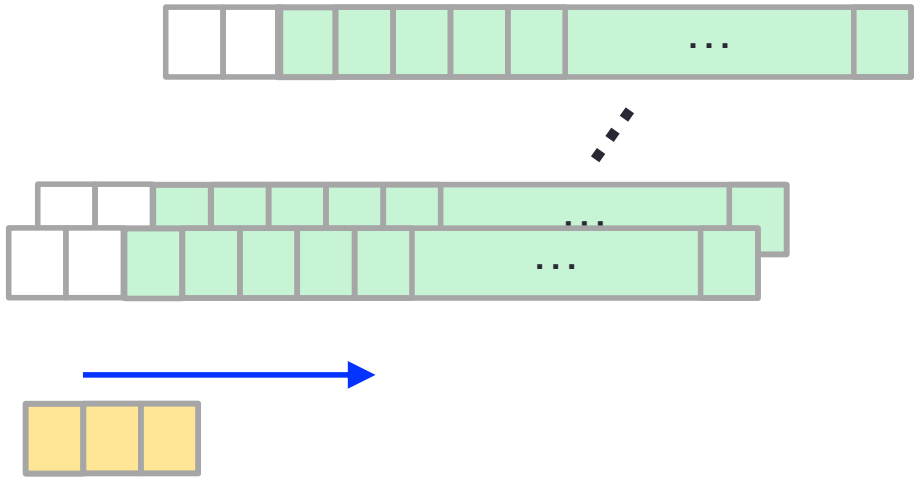
Input feature vectors (K time frames)

Conv filters

Temporal causal dilated convolution

add padding on left side of input sequence:

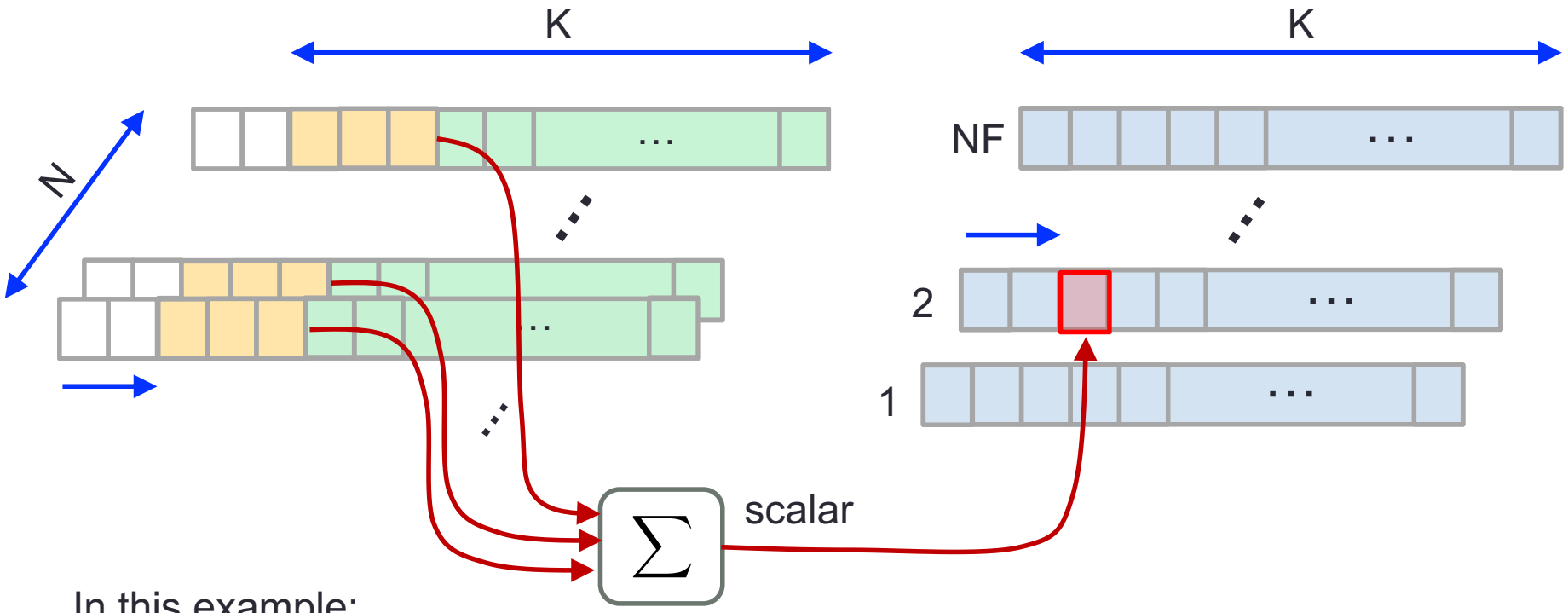
- to achieve causality and
- to preserve same size at output



Pick a filter and convolve it with the input

- Independently convolve the same filter with each input feature vector
- Shared weights
  - same filter weights applied to multiple input vectors

Temporal causal dilated convolution

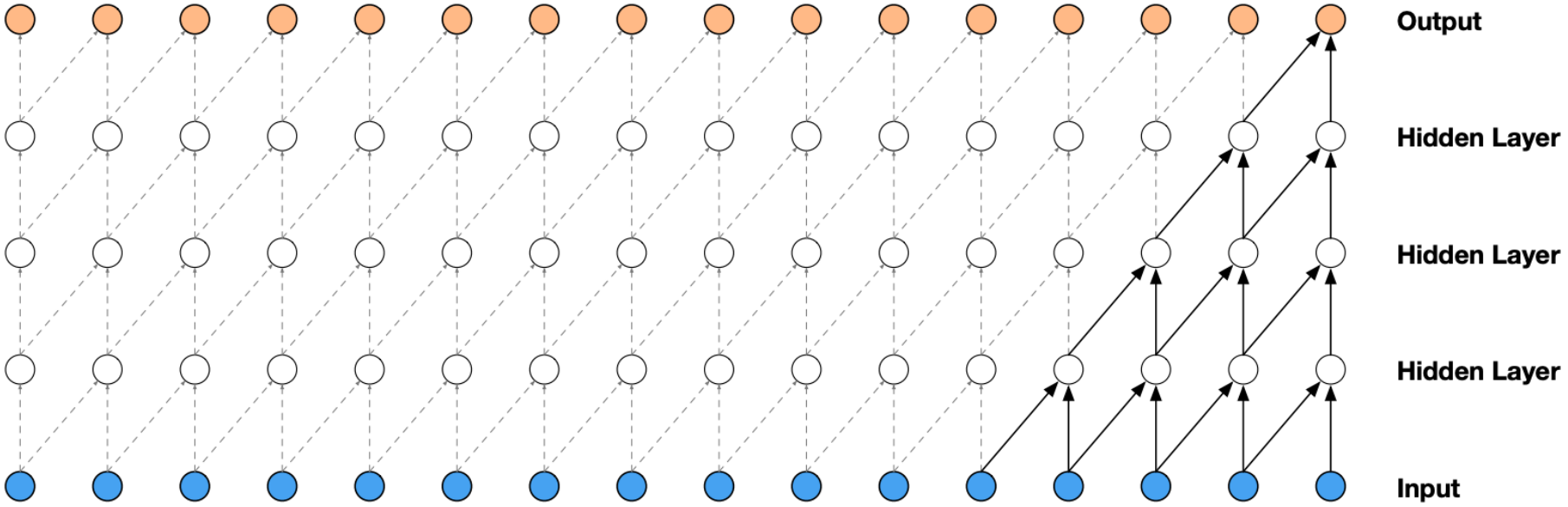


In this example:

- Convolve kernel no. 2
- Same kernel convolved in position 3 with all feature vectors
- This will lead to a single output feature
  - for vector no. 2 (same as kernel no.)
  - In position 3 (same as temporal sliding index)

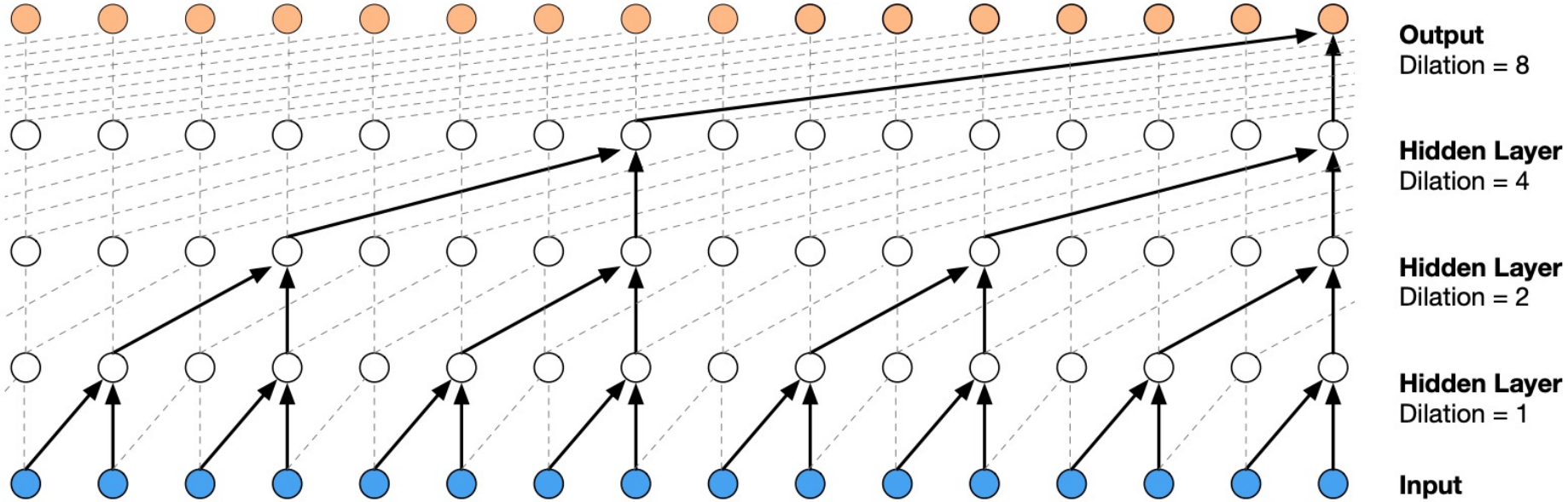
Dilation and receptive field (1/2)

- 3 convolutional layers
  - Causal 1D convolution
  - Each output feature depends on 5 subsequent input features



Dilation and receptive field (2/2)

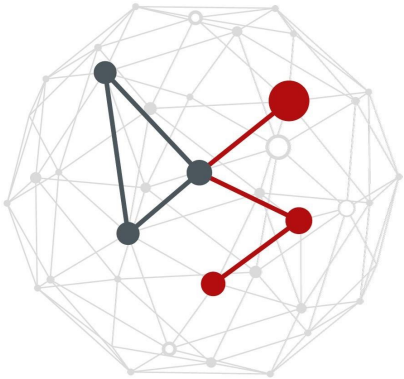
- 3 convolutional layers
  - Causal dilated 1D convolution (*large receptive fields* with a few layers)
  - Each output feature depends on all input features



  - WaveNet (Google-Deepmind 2016)
    - https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

MIME
Master's degree ICT Internet Multimedia Engineering

BACKPROPAGATION FOR CNN



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO
MATEMATICA

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Notation

1. $\ell = 1, 2, \ldots, L$ is the ℓ^{th} layer
2. Input feature map of dimension $H \times W$ with iterators i and j
3. Kernel $\boldsymbol{w}$ is of dimension $k_1 \times k_2$ and has m and n as iterators
4. $w^{\ell}_{m,n}$ is the weight connecting neurons from layer $\ell - 1$ to layer ℓ
5. b^{ℓ} is the bias unit at layer ℓ
6. $a^{\ell}_{i,j}$ is the activation at layer ℓ : input map is the output from previous layer

$$a^{\ell}_{i,j} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} w^{\ell}_{m,n} y^{\ell-1}_{i+m,j+n} + b^{\ell}$$

cross-correlation

7. $y^{\ell}_{i,j}$ is the output at layer ℓ given by :

$$y^{\ell}_{i,j} = f(a^{\ell}_{i,j})$$

8. $f(\cdot)$ is the **activation function**

weights $w_{m,n}$ and b only depend on layer no. They are shared across all input values

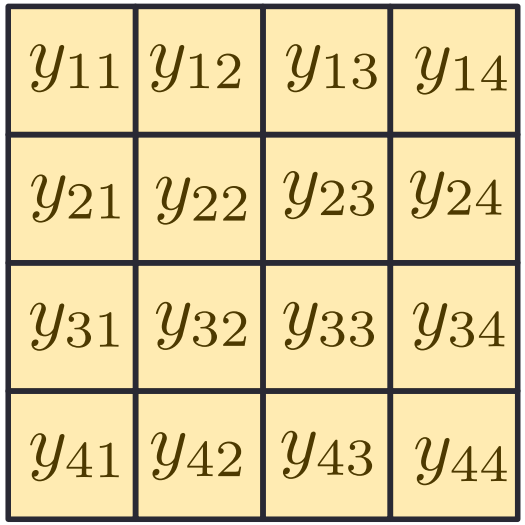
Backpropagation for CNNs

- Works exactly as for Feed Forward Neural Networks
  - Backpropagating the error deltas
  - Recursive computation of partial derivatives on graphs
- Output layer: the deltas are computed as seen for FFNNs
  - Last layer of a CNN is dense
  - Deltas in the last layer depend on the last layer type, i.e.,
  - Regression: linear (dense layer)
  - Classification: softmax (dense layer)
- The backprop steps are
  - Output layer: compute deltas as done for FFNNs
  - For each hidden layer
    - If layer is dense (FFNN): use backprop equations derived for FFNNs
    - If layer is CONV: replace backprop equations with new ones (see next)

CONV layer diagram (P=0, S=1, d=1)

element-wise application of activation function

$H \times W$

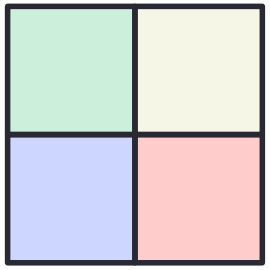


$y_{i,j}^{\ell-1}$

input feature map = output from previous layer

$\otimes$

$k_1 \times k_2$

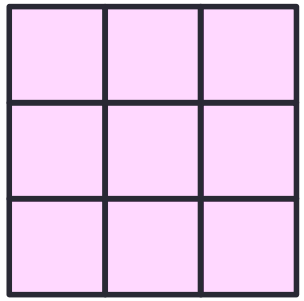


$w_{m,n}^{\ell}$

Kernel (or filter)

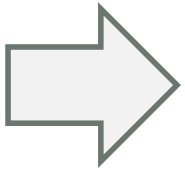
$=$

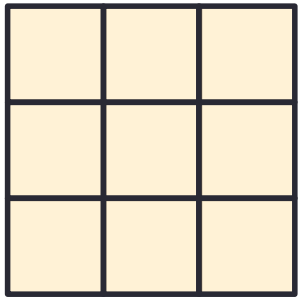
$(H - k_1 + 1) \times (W - k_2 + 1)$



$a_{i,j}^{\ell}$

linear activation





$y_{i,j}^{\ell} = f(a_{i,j}^{\ell})$

output of current layer

$$a_{i,j}^{\ell} = \sum_m \sum_n w_{m,n}^{\ell} y_{i+m,j+n}^{\ell-1} + b^{\ell}$$

Point-wise error gradient

- For each input pattern we need to compute:

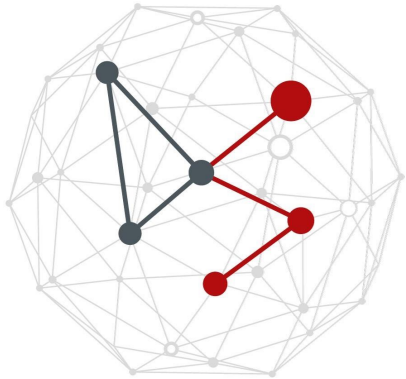
$$\frac{\partial E(\boldsymbol{w})}{\partial w_{m,n}^{\ell}} = ?$$

- This quantifies
  - the variation of the output error $E(\boldsymbol{w})$
  - with respect to a corresponding change in the weight (m,n) in layer ℓ

- We neglect the time index (input sample no.) for simplicity
  - the idea is that we use Stochastic Gradient Descent (SGD)
  - and thus we update the gradient in a pointwise manner
  - as opposed to computing the real gradient (aka, “batch mode”)

MIME
Master's degree ICT Internet Multimedia Engineering

UPDATING THE WEIGHTS



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

UNIVERSITÀ DEGLI STUDI DI PADOVA

Gradient wrt weights

- For each input pattern (time index neglected), we have:

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{m,n}^{\ell}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E(\boldsymbol{w})}{\partial a_{i,j}^{\ell}} \frac{\partial a_{i,j}^{\ell}}{\partial w_{m,n}^{\ell}}$$

$$= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^{\ell} \frac{\partial a_{i,j}^{\ell}}{\partial w_{m,n}^{\ell}} \quad (1)$$

- with:

$$a_{i,j}^{\ell} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} w_{m,n}^{\ell} y_{i+m,j+n}^{\ell-1} + b^{\ell}$$

Derivative of activations wrt weights

- We now compute: $\frac{\partial a^{\ell}_{i,j}}{\partial w^{\ell}_{m,n}} = ?$

Being: $a^{\ell}_{i,j} = \sum_{m'} \sum_{n'} w^{\ell}_{m',n'} y^{\ell-1}_{i+m',j+n'} + b^{\ell}$

We get: $\frac{\partial a^{\ell}_{i,j}}{\partial w^{\ell}_{m,n}} = y^{\ell-1}_{i+m,j+n}$

That substituting into (1) leads to:

$$\frac{\partial E(\boldsymbol{w})}{\partial w^{\ell}_{m,n}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^{\ell}_{i,j} \frac{\partial a^{\ell}_{i,j}}{\partial w^{\ell}_{m,n}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^{\ell}_{i,j} y^{\ell-1}_{i+m,j+n}$$

Derivative of activations wrt weights

- So far:

$$\frac{\partial E(\boldsymbol{w})}{\partial w^{\ell}_{m,n}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^{\ell}_{i,j} y^{\ell-1}_{i+m,j+n}$$

- This is precisely the cross-correlation

$$\frac{\partial E(\boldsymbol{w})}{\partial w^{\ell}_{m,n}} = \boldsymbol{\delta}^{\ell} \otimes \boldsymbol{y}^{\ell-1}_{m,n}$$

Matrix of deltas
in current layer
size: (H-k_1+1)x(W-k_2+1)

Output from previous layer
patch of size (H-k_1+1)x(W-k_2+1)
with *upper-left corner* in (m,n)

Updating the kernel weights (1/5)

$H \times W$

y_{00}	y_{01}	y_{02}	y_{03}
y_{10}	y_{11}	y_{12}	y_{13}
y_{20}	y_{21}	y_{22}	y_{23}
y_{30}	y_{31}	y_{32}	y_{33}

$y_{i,j}^{\ell-1}$

$(H - k_1 + 1) \times (W - k_2 + 1)$

same size as matrix $a_{i,j}^{\ell}$

δ_{00}	δ_{01}	δ_{02}
δ_{10}	δ_{11}	δ_{12}
δ_{20}	δ_{21}	δ_{22}

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{m,n}^{\ell}} = \boldsymbol{\delta}^{\ell} \otimes \boldsymbol{y}_{m,n}^{\ell-1}$$

$i = 0, \ldots, H - 1$

$j = 0, \ldots, W - 1$

$m = 0, \ldots, k_1 - 1$

$n = 0, \ldots, k_2 - 1$

Updating the kernel weights (2/5)

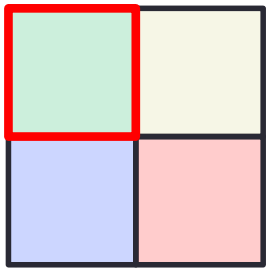
$H \times W$

δ_{00}	δ_{01}	δ_{02}	y_{03}
δ_{10}	δ_{11}	δ_{12}	y_{13}
δ_{20}	δ_{21}	δ_{22}	y_{23}
y_{30}	y_{31}	y_{32}	y_{33}

$(H - k_1 + 1) \times (W - k_2 + 1)$

δ_{00}	δ_{01}	δ_{02}
δ_{10}	δ_{11}	δ_{12}
δ_{20}	δ_{21}	δ_{22}

$y_{i,j}^{\ell-1}$



Weight matrix (Kernel)

Point-wise gradient descent:

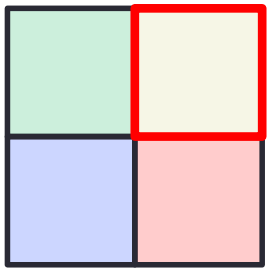
$$\frac{\partial E(\boldsymbol{w})}{\partial w_{0,0}^{\ell}} \Rightarrow w_{0,0}^{\ell} = w_{0,0}^{\ell} - \eta \frac{\partial E(\boldsymbol{w})}{\partial w_{0,0}^{\ell}}$$

Updating the kernel weights (3/5)

$H \times W$

y_{00}	δ_{00}	δ_{01}	δ_{02}
y_{10}	δ_{10}	δ_{11}	δ_{12}
y_{20}	δ_{20}	δ_{21}	δ_{22}
y_{30}	y_{31}	y_{32}	y_{33}

$y_{i,j}^{\ell-1}$



Weight matrix (Kernel)

$(H - k_1 + 1) \times (W - k_2 + 1)$

δ_{00}	δ_{01}	δ_{02}
δ_{10}	δ_{11}	δ_{12}
δ_{20}	δ_{21}	δ_{22}

Point-wise gradient descent:

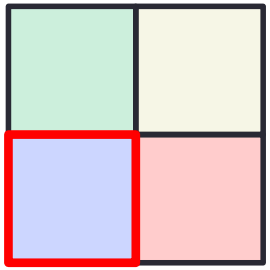
$$\frac{\partial E(\boldsymbol{w})}{\partial w_{0,1}^{\ell}} \Rightarrow w_{0,1}^{\ell} = w_{0,1}^{\ell} - \eta \frac{\partial E(\boldsymbol{w})}{\partial w_{0,1}^{\ell}}$$

Updating the kernel weights (4/5)

$H \times W$

y_{00}	y_{01}	y_{02}	y_{03}
δ_{00}	δ_{01}	δ_{02}	y_{13}
δ_{10}	δ_{11}	δ_{12}	y_{23}
δ_{20}	δ_{21}	δ_{22}	y_{33}

$y_{i,j}^{\ell-1}$



Weight matrix (Kernel)

$(H - k_1 + 1) \times (W - k_2 + 1)$

δ_{00}	δ_{01}	δ_{02}
δ_{10}	δ_{11}	δ_{12}
δ_{20}	δ_{21}	δ_{22}

Point-wise gradient descent:

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{1,0}^{\ell}} \Rightarrow w_{1,0}^{\ell} = w_{1,0}^{\ell} - \eta \frac{\partial E(\boldsymbol{w})}{\partial w_{1,0}^{\ell}}$$

Updating the kernel weights (5/5)

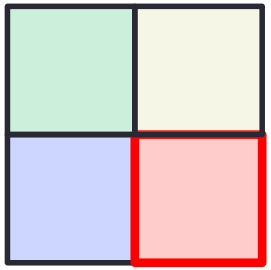
$H \times W$

y_{00}	y_{01}	y_{02}	y_{03}
y_{10}	δ_{00}	δ_{01}	δ_{02}
y_{20}	δ_{10}	δ_{11}	δ_{12}
y_{30}	δ_{20}	δ_{21}	δ_{22}

$y_{i,j}^{\ell-1}$

$(H - k_1 + 1) \times (W - k_2 + 1)$

δ_{00}	δ_{01}	δ_{02}
δ_{10}	δ_{11}	δ_{12}
δ_{20}	δ_{21}	δ_{22}



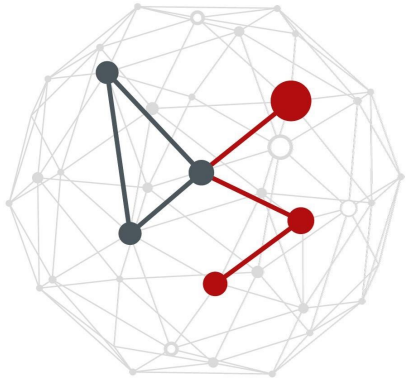
Weight matrix (Kernel)

Point-wise gradient descent:

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{1,1}^{\ell}} \Rightarrow w_{1,1}^{\ell} = w_{1,1}^{\ell} - \eta \frac{\partial E(\boldsymbol{w})}{\partial w_{1,1}^{\ell}}$$

MIME
Master's degree ICT Internet Multimedia Engineering

UPDATING THE DELTAS



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

UNIVERSITÀ DEGLI STUDI DI PADOVA

Gradient of E(**w**) wrt activations

- We now look at: $\delta_{i,j}^{\ell} \triangleq \frac{\partial E(\boldsymbol{w})}{\partial a_{i,j}^{\ell}} = ?$

$$\delta_{i,j}^{\ell} \triangleq \frac{\partial E(\boldsymbol{w})}{\partial a_{i,j}^{\ell}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \frac{\partial E(\boldsymbol{w})}{\partial a_{i-m,j-n}^{\ell+1}} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^{\ell}} =$$

$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i-m,j-n}^{\ell+1} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^{\ell}} \qquad (2)$$

- indices (m,n) span over the receptive field of the kernel that is applied to the output of layer l to compute the activations for the next layer l+1
- the indices (i-m) and (j-n) must be within the valid indices for the δ matrix in the next layer l+1 (e.g., if negative the result is zero)

Gradient of E(**w**) wrt activations

- We now look at: $\delta_{i,j}^{\ell} \triangleq \dfrac{\partial E(\boldsymbol{w})}{\partial a_{i,j}^{\ell}} = ?$

$$\delta_{i,j}^{\ell} \triangleq \frac{\partial E(\boldsymbol{w})}{\partial a_{i,j}^{\ell}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \frac{\partial E(\boldsymbol{w})}{\partial a_{i-m,j-n}^{\ell+1}} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^{\ell}} =$$

$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i-m,j-n}^{\ell+1} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^{\ell}} \qquad (2)$$

Next step: $\dfrac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^{\ell}} = ?$

Gradient of E(**w**) wrt activations

- We now look at: $\dfrac{\partial a^{\ell+1}_{i-m,j-n}}{\partial a^{\ell}_{i,j}} = ?$

$$a^{\ell+1}_{i-m,j-n} = \sum_{m'=0}^{k_1-1}\sum_{n'=0}^{k_2-1} w^{\ell+1}_{m',n'} y^{\ell}_{i-m+m',j-n+n'} + b^{\ell+1}$$

$$\frac{\partial a^{\ell+1}_{i-m,j-n}}{\partial a^{\ell}_{i,j}} = \frac{\partial}{\partial a^{\ell}_{i,j}}\left(\sum_{m'}\sum_{n'} w^{\ell+1}_{m',n'} y^{\ell}_{i-m+m',j-n+n'} + b^{\ell+1}\right)$$

$$= \frac{\partial}{\partial a^{\ell}_{i,j}}\left(\sum_{m'}\sum_{n'} w^{\ell+1}_{m',n'} f(a^{\ell}_{i-m+m',j-n+n'})\right)$$

Gradient of E(**w**) wrt activations

$$\frac{\partial a^{\ell+1}_{i-m,j-n}}{\partial a^{\ell}_{i,j}} = \frac{\partial}{\partial a^{\ell}_{i,j}} \left(\sum_{m'} \sum_{n'} w^{\ell+1}_{m',n'} f(a^{\ell}_{i-m+m',j-n+n'}) \right)$$

$$= w^{\ell+1}_{m,n} \frac{\partial f(a^{\ell}_{i,j})}{\partial a^{\ell}_{i,j}} \triangleq w^{\ell+1}_{m,n} f'(a^{\ell}_{i,j})$$

- The derivative is non-zero only for: $m = m', n = n'$
  - so that i-m+m'=i and j-n+n'=j
- We now use this result into Eq. (2)

Gradient of E(**w**) wrt activations

Eq. (2):

$$\delta_{i,j}^{\ell} = \frac{\partial E(\boldsymbol{w})}{\partial a_{i,j}^{\ell}} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i-m,j-n}^{\ell+1} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^{\ell}} =$$

$$= \sum_{m} \sum_{n} \delta_{i-m,j-n}^{\ell+1} w_{m,n}^{\ell+1} f'(a_{i,j}^{\ell}) =$$

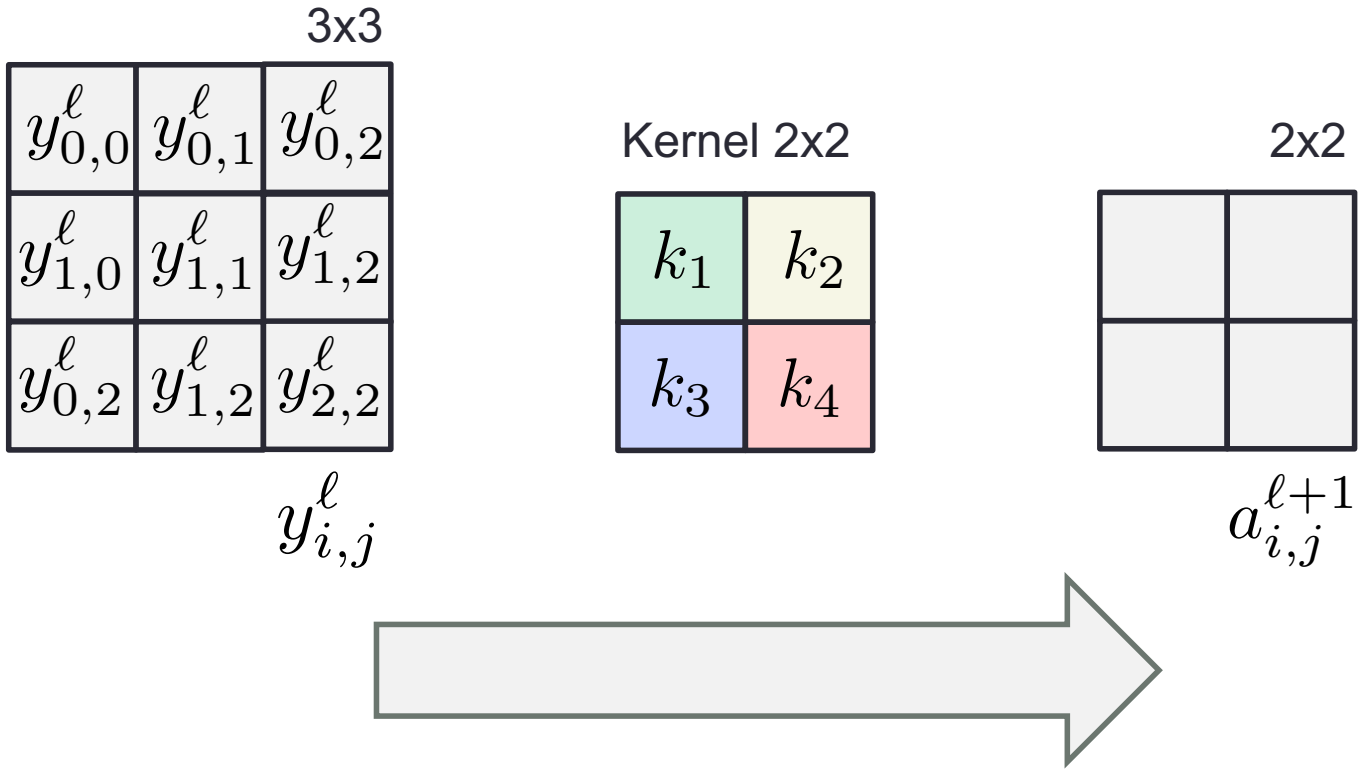
$$= f'(a_{i,j}^{\ell}) \boldsymbol{\delta}_{i,j}^{\ell+1} \otimes \mathrm{rot}_{180^\circ}(\boldsymbol{w}^{\ell+1})$$

it is a cross-correlation with a flipped Kernel

matrix δ is shifted with respect to rot_{180°(**w**) according to (i,j)

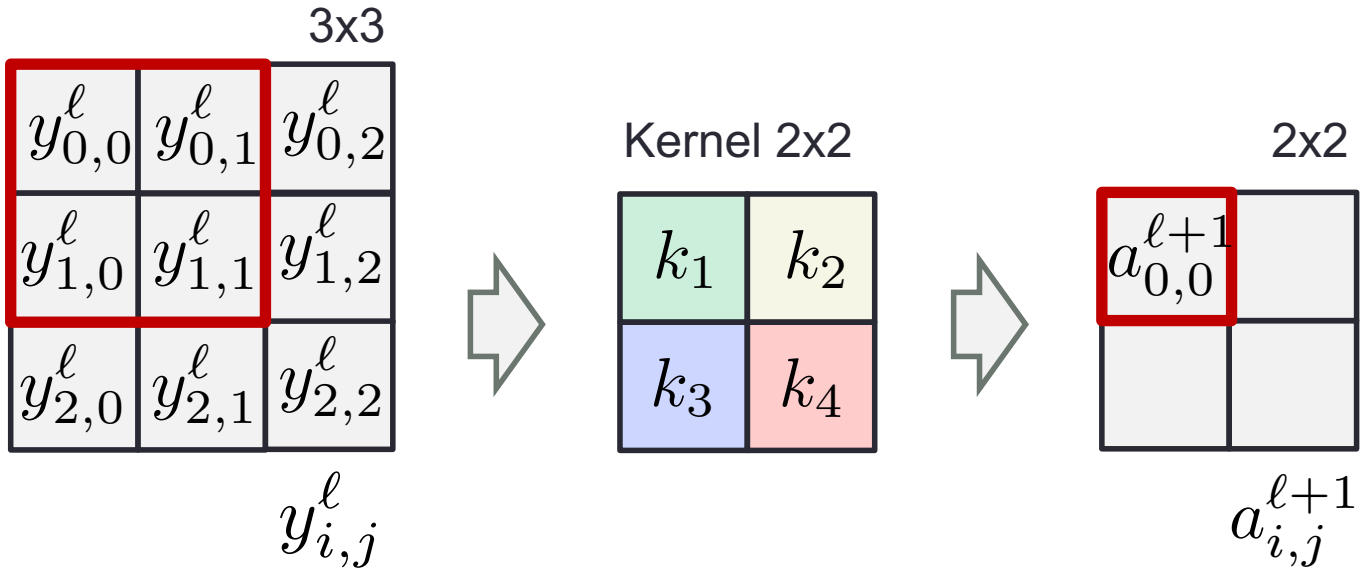
What are we doing? We are backtracking the error δ coming from all the output elements in the next layer $\ell + 1$ that, in the forward propagation phase, receive the signal from (are connected to) the activation $a_{i,j}^{\ell}$

Forward pass example



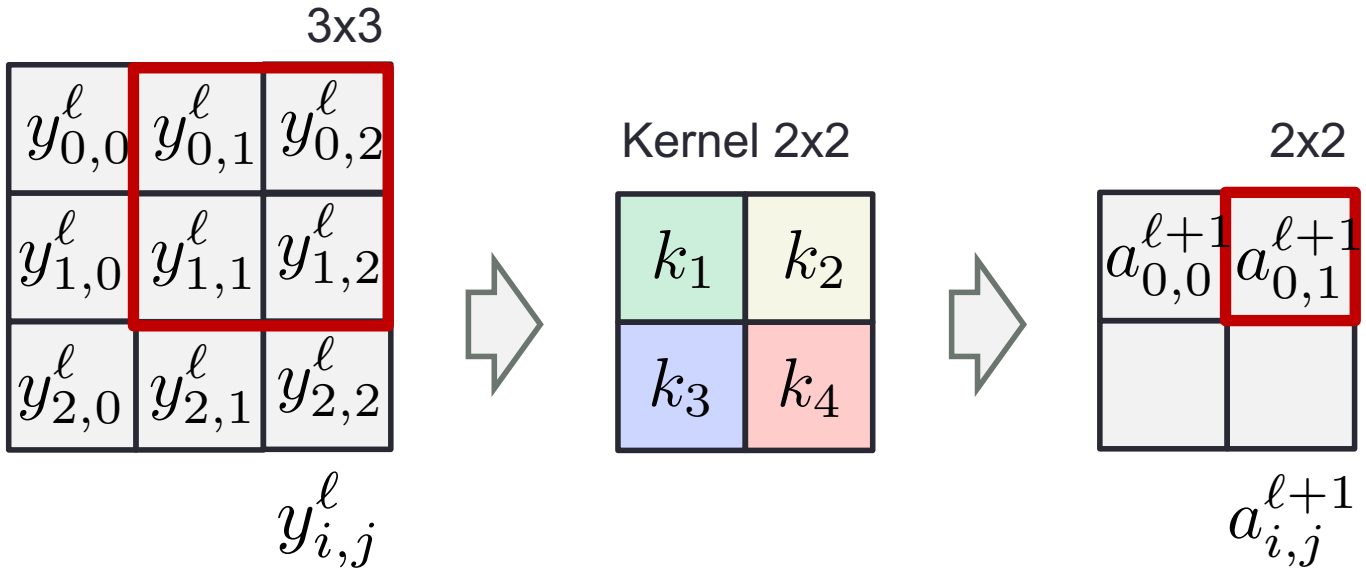
forward pass = cross-correlation (stride=1)

Forward pass example



$$a^{\ell+1}_{0,0} = y^{\ell}_{0,0}k_1 + y^{\ell}_{0,1}k_2 + y^{\ell}_{1,0}k_3 + y^{\ell}_{1,1}k_4$$

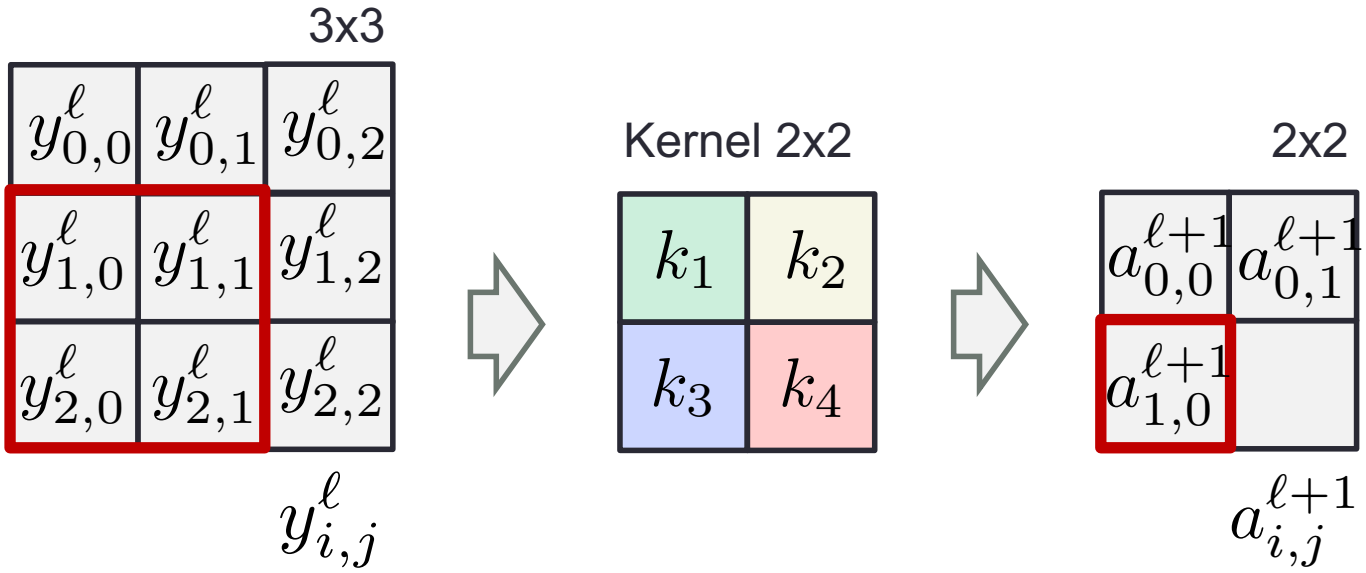
Forward pass example



$$a^{\ell+1}_{0,0} = y^{\ell}_{0,0}k_1 + y^{\ell}_{0,1}k_2 + y^{\ell}_{1,0}k_3 + y^{\ell}_{1,1}k_4$$

$$a^{\ell+1}_{0,1} = y^{\ell}_{0,1}k_1 + y^{\ell}_{0,2}k_2 + y^{\ell}_{1,1}k_3 + y^{\ell}_{1,2}k_4$$

Forward pass example

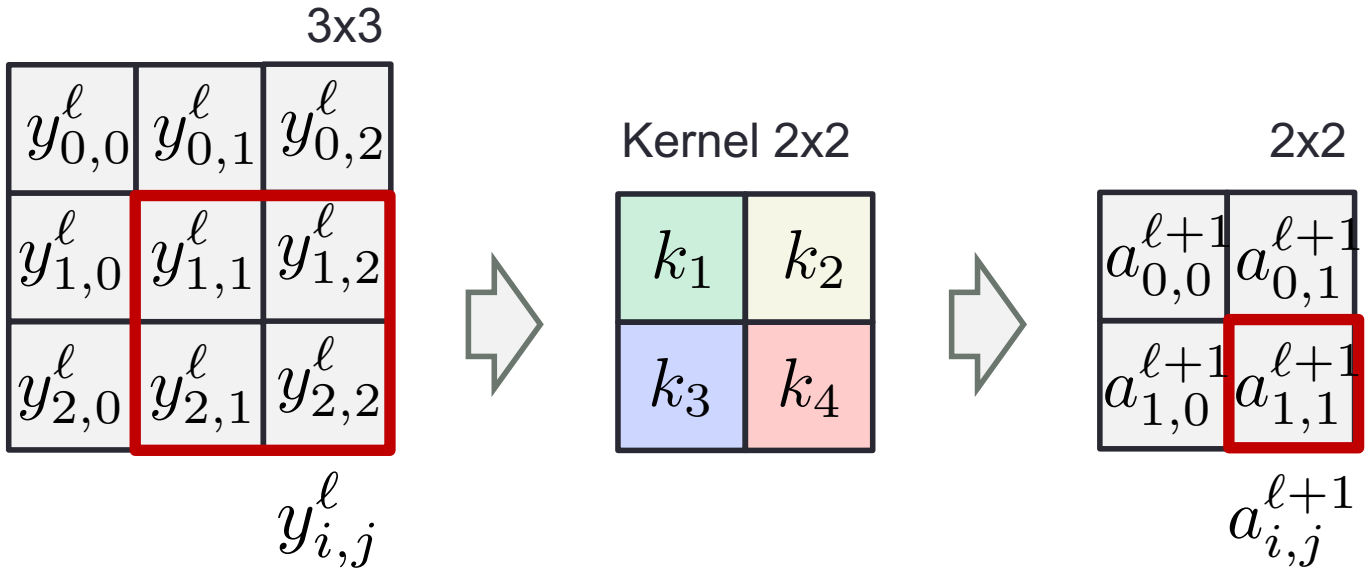


$$a^{\ell+1}_{0,0} = y^{\ell}_{0,0}k_1 + y^{\ell}_{0,1}k_2 + y^{\ell}_{1,0}k_3 + y^{\ell}_{1,1}k_4$$

$$a^{\ell+1}_{0,1} = y^{\ell}_{0,1}k_1 + y^{\ell}_{0,2}k_2 + y^{\ell}_{1,1}k_3 + y^{\ell}_{1,2}k_4$$

$$a^{\ell+1}_{1,0} = y^{\ell}_{1,0}k_1 + y^{\ell}_{1,1}k_2 + y^{\ell}_{2,0}k_3 + y^{\ell}_{2,1}k_4$$

Forward pass example



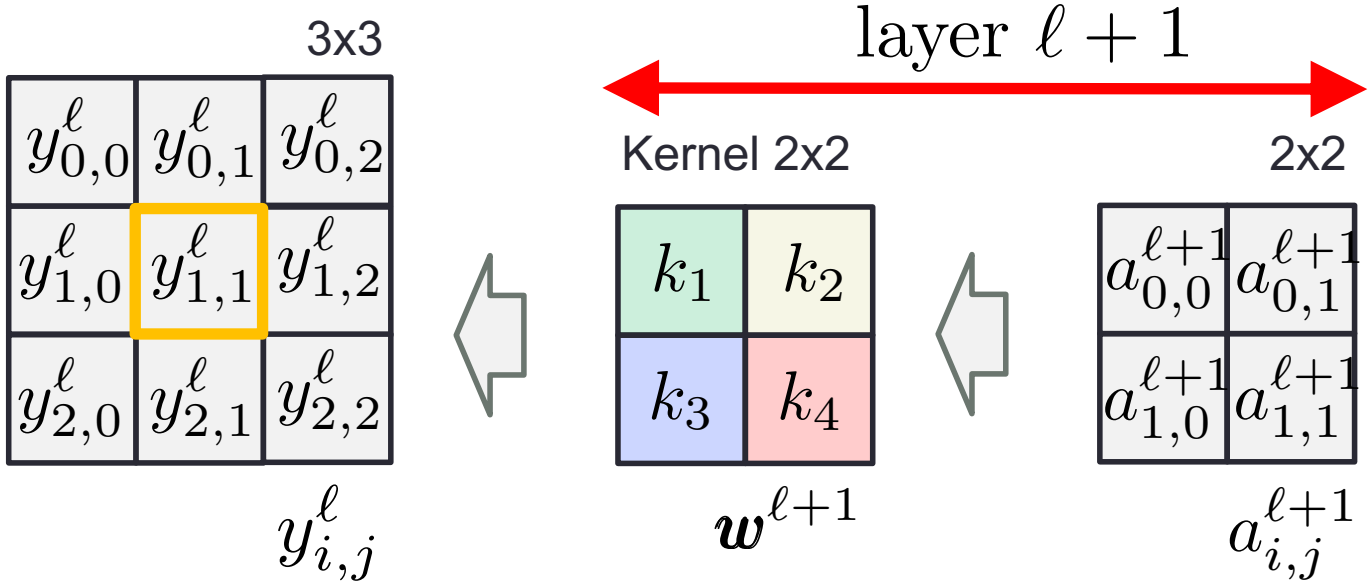
$$a^{\ell+1}_{0,0} = y^{\ell}_{0,0}k_1 + y^{\ell}_{0,1}k_2 + y^{\ell}_{1,0}k_3 + y^{\ell}_{1,1}k_4$$

$$a^{\ell+1}_{0,1} = y^{\ell}_{0,1}k_1 + y^{\ell}_{0,2}k_2 + y^{\ell}_{1,1}k_3 + y^{\ell}_{1,2}k_4$$

$$a^{\ell+1}_{1,0} = y^{\ell}_{1,0}k_1 + y^{\ell}_{1,1}k_2 + y^{\ell}_{2,0}k_3 + y^{\ell}_{2,1}k_4$$

$$a^{\ell+1}_{1,1} = y^{\ell}_{1,1}k_1 + y^{\ell}_{1,2}k_2 + y^{\ell}_{2,1}k_3 + y^{\ell}_{2,2}k_4$$

Backtracking dependencies from forward pass



$$a^{\ell+1}_{0,0} = y^{\ell}_{0,0}k_1 + y^{\ell}_{0,1}k_2 + y^{\ell}_{1,0}k_3 + y^{\ell}_{1,1}k_4$$

$$a^{\ell+1}_{0,1} = y^{\ell}_{0,1}k_1 + y^{\ell}_{0,2}k_2 + y^{\ell}_{1,1}k_3 + y^{\ell}_{1,2}k_4$$

$$a^{\ell+1}_{1,0} = y^{\ell}_{1,0}k_1 + y^{\ell}_{1,1}k_2 + y^{\ell}_{2,0}k_3 + y^{\ell}_{2,1}k_4$$

$$a^{\ell+1}_{1,1} = y^{\ell}_{1,1}k_1 + y^{\ell}_{1,2}k_2 + y^{\ell}_{2,1}k_3 + y^{\ell}_{2,2}k_4$$

where the rotation comes from

$$\frac{\partial \boldsymbol{a}^{\ell+1}_{i,j}}{\partial y^{\ell}_{1,1}} = \mathrm{rot}_{180^\circ}(\boldsymbol{w}^{\ell+1})$$

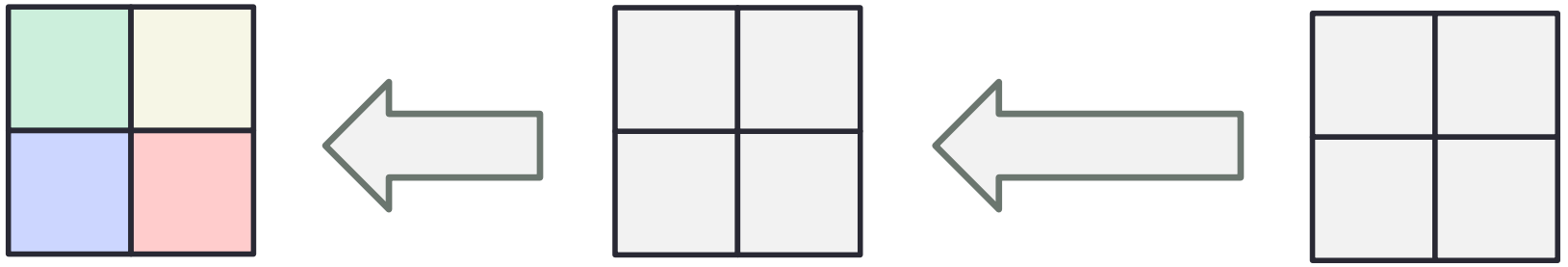
$$\text{for } i,j = 0,1$$

Putting it all together

layer ℓ — layer $\ell + 1$

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{m,n}^{\ell}} = \boldsymbol{\delta}^{\ell} \otimes \boldsymbol{y}_{m,n}^{\ell-1}$$

$\delta_{i,j}^{\ell+1}$



$$\delta_{i,j}^{\ell} = f'(a_{i,j}^{\ell})\boldsymbol{\delta}_{i,j}^{\ell+1} \otimes \mathrm{rot}_{180^\circ}(\boldsymbol{w}^{\ell+1})$$

With ReLU:

$$f'(a_{i,j}^{\ell}) = \begin{cases} 0 & a_{i,j}^{\ell} \leq 0 \\ 1 & a_{i,j}^{\ell} > 0 \end{cases}$$

Update of biases (1/2)

- Following the same steps we took for the weights, we have:

$$\frac{\partial E(\boldsymbol{w})}{\partial b^{\ell}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E(\boldsymbol{w})}{\partial a_{i,j}^{\ell}} \frac{\partial a_{i,j}^{\ell}}{\partial b^{\ell}}$$

$$= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^{\ell} \frac{\partial a_{i,j}^{\ell}}{\partial b^{\ell}}$$

$$\frac{\partial a_{i,j}^{\ell}}{\partial b^{\ell}} = ?$$

Update of biases (2/3)

- Following the same approach as before, we get:

$$\frac{\partial a_{i,j}^{\ell}}{\partial b^{\ell}} = ?$$

$$\frac{\partial a_{i,j}^{\ell}}{\partial b^{\ell}} = \frac{\partial}{\partial b^{\ell}} \left(\sum_{m} \sum_{n} w_{m,n}^{\ell} y_{i+m,j+n}^{\ell-1} + b^{\ell} \right) = 1$$

Update of biases (3/3)

- Following the same approach as before, we get:

$$\frac{\partial E(\boldsymbol{w})}{\partial b^{\ell}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^{\ell} \frac{\partial a_{i,j}^{\ell}}{\partial b^{\ell}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^{\ell}$$

- And the bias is updated via stochastic gradient descent, as:

$$b^{\ell} = b^{\ell} - \eta \frac{\partial E(\boldsymbol{w})}{\partial b^{\ell}}$$

Accounting for the pooling layer (1/2)

- At the poling layer, forward propagation results in
  - we take the output image of the current CONV layer
  - we subdivide it into blocks of NxN values
- Each block is reduced to a single value
- This is the value of the "winning" unit (max. pooling)
- To keep track of this winning unit
  - its value is saved during the forward pass
  - and used to route the gradient during backpropagation

Accounting for the pooling layer (2/2)

- Gradient routing during backpropagation

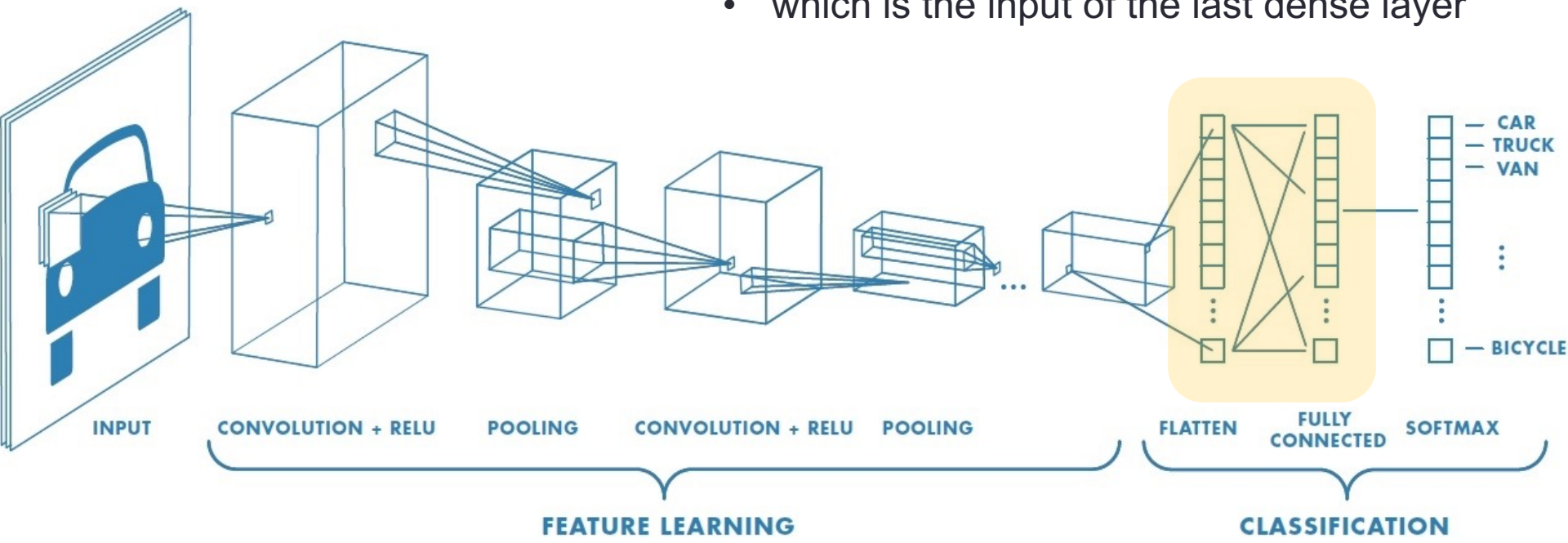
- Max-pooling - the error is just assigned to where it comes from - the “winning unit”, because other units in the previous layer’s pooling blocks did not contribute to it. Thus, all the other units gets zero

- Average pooling - the error is multiplied by 1/(NxN) and assigned to the whole corresponding pooling block, all units get this same (average) value

CNN - the whole picture

Flattening

- from 3D volume to vector
- which is the input of the last dense layer



Bibliography

[1] I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning,” Mit Press, 2016.

[2] V. Dumoulin, F. Visin. “A guide to convolution arithmetic for deep learning,” 2016.

- https://arxiv.org/pdf/1603.07285.pdf
- https://github.com/vdumoulin/conv_arithmetic

[3] A. v.d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A Senior, K. Kavukcuoglu, “WaveNet: a Generative Model for Raw Audio,” published as **arXiv:1609.03499v2**, September 2016.

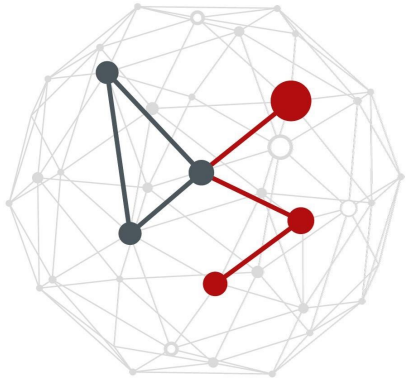
- https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

MiME
Master's degree ICT Internet Multimedia Engineering

CONVOLUTIONAL NEURAL NETWORKS (CNN)

Michele Rossi
michele.rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

UNIVERSITÀ DEGLI STUDI DI PADOVA