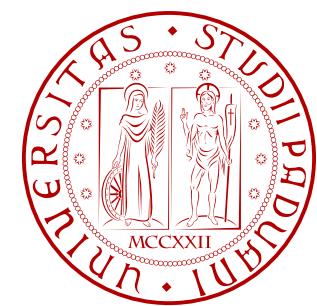


CONVOLUTIONAL NEURAL NETWORKS (CNN)

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



Outline

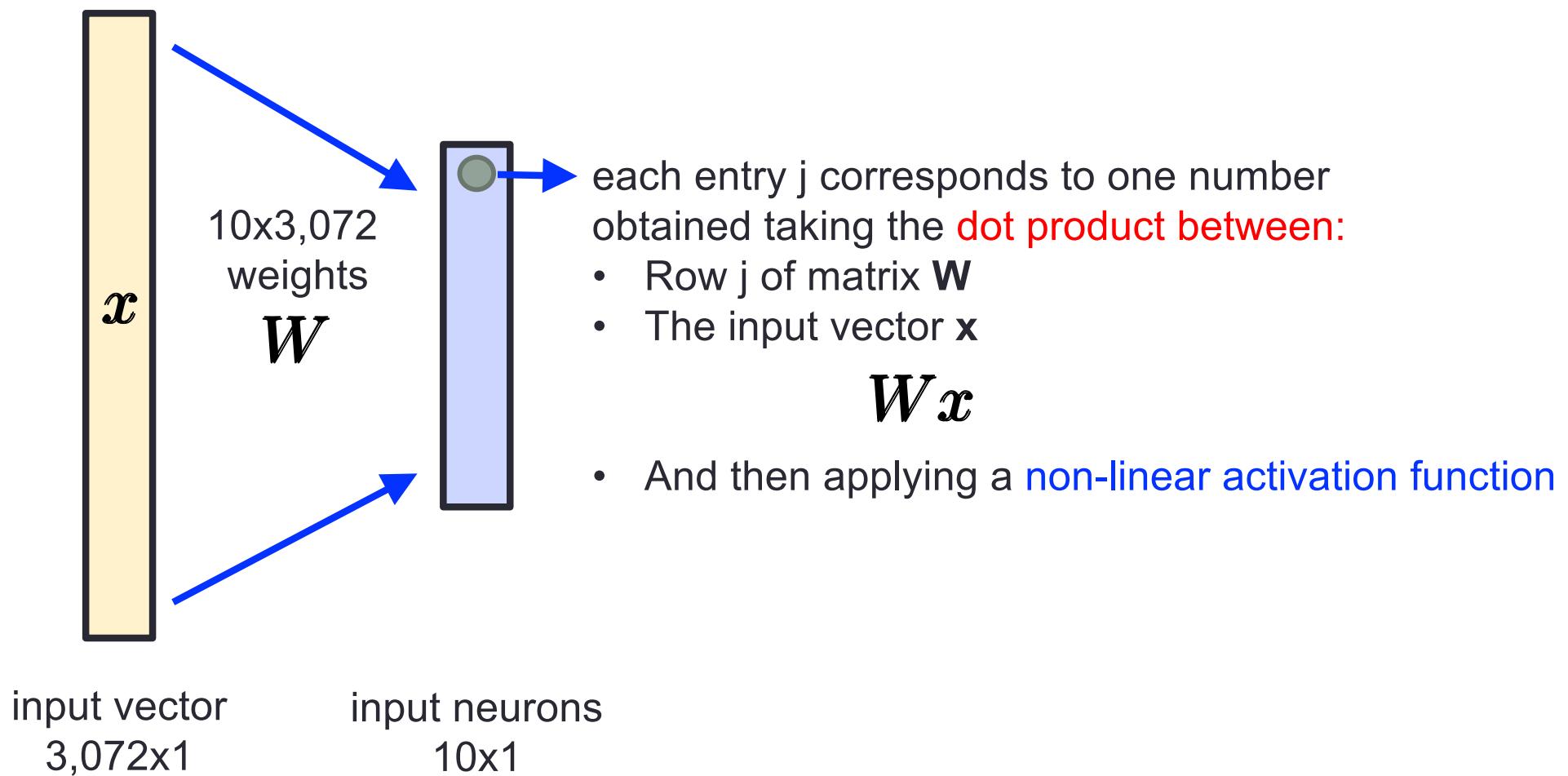
- Introduction to CNNs
- The convolution operation
 - Visual insight
 - Convolution operator
 - Input and output volumes
 - Kernels, padding
- Characteristics of CNNs
 - Sparse interactions
 - Parameter sharing
 - Equivariance (pooling)
- Backprop for CNN layers
- Bibliography

CNN - introduction

- 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

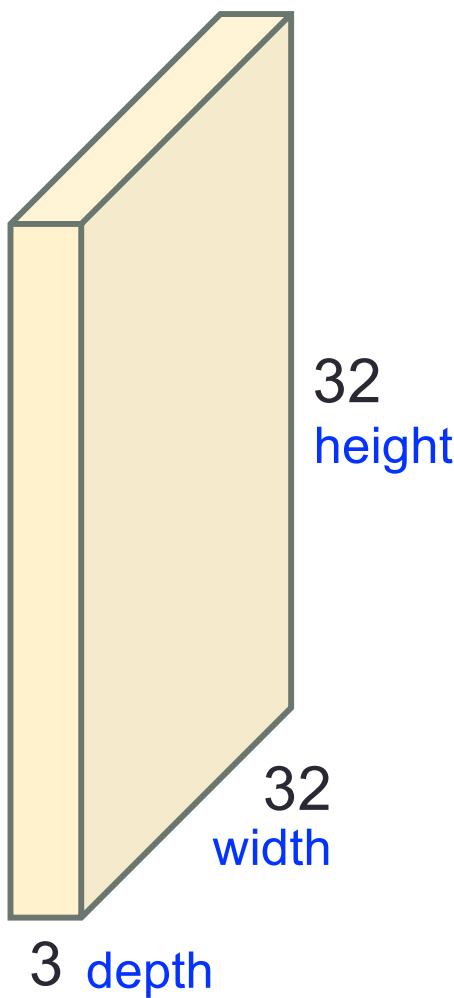
Fully connected layer

- Example 32x32x3 image (3 color ch) \rightarrow flatten to 3,072x1



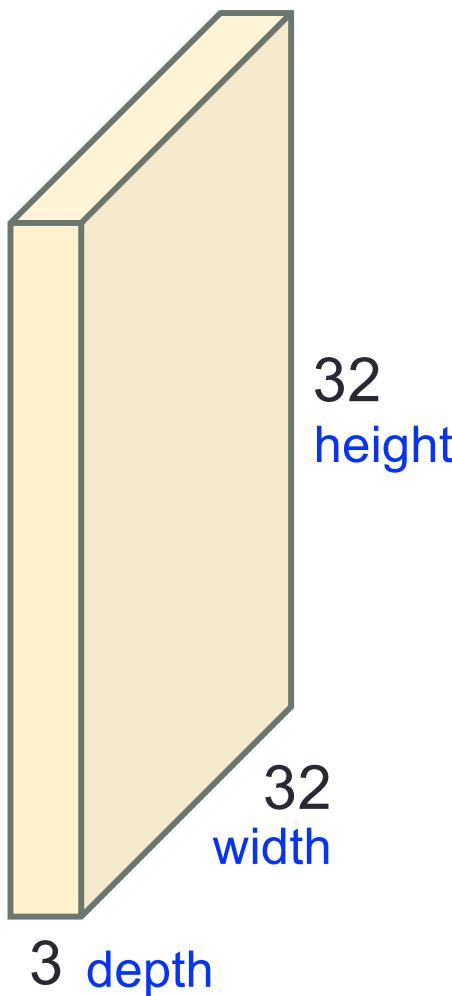
Convolution layer

- Example 32x32x3 image → preserve spatial structure

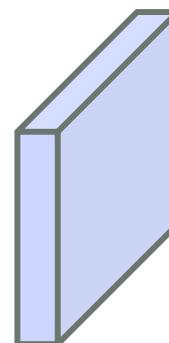


Convolution layer

32x32x3 image



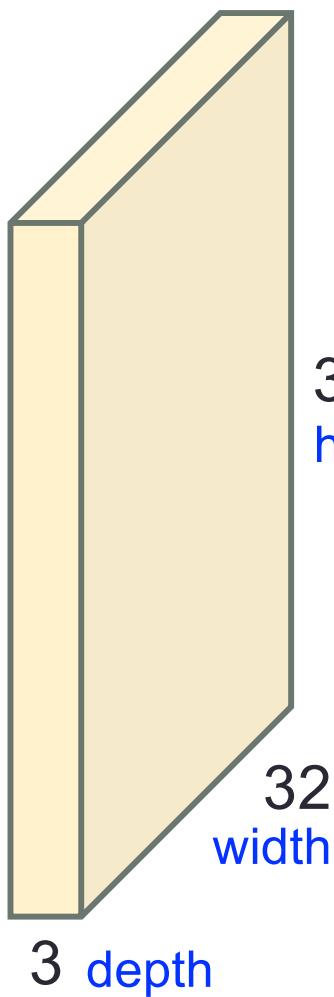
5x5x3 filter (or kernel)



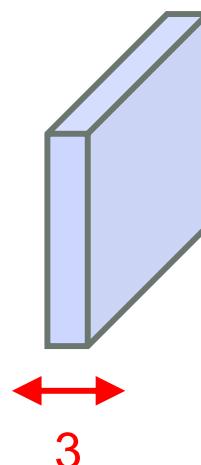
Convolve the filter with the input image, i.e., “slide over the image spatially convolving the kernel with each image patch”

Convolution 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

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

1D convolution (2/2)

$$s(t) = \int_{\text{feature map}}^{\text{input}} x(a)w(t-a)da \triangleq (x * w)(t)$$

- 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 will look into the future
 - These facts hold 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 **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

A 3x3 grid representing an input signal. The columns are labeled 0, 1, 2 and the rows are labeled 0, 1, 2. The elements are numbered 1 through 9. A diagonal line from the top-left corner to the bottom-right corner is labeled i and j , indicating the indices of the elements.

	j		
	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

$$I(i, j)$$

Input 2D signal

A 3x3 grid representing a kernel. The columns are labeled -1, 0, 1 and the rows are labeled -1, 0, 1. The elements are numbered -1 through 1. A diagonal line from the top-left corner to the bottom-right corner is labeled m and n , indicating the indices of the elements.

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

$$K(m, n)$$

Kernel

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

	j	0	1	2
i	0	1	2	3
1	4	5	6	
2	7	8	9	

$I(i, j)$

	n	-1	0	1
m	-1	-1	-2	-1
0	0	0	0	
1	1	2	1	

$K(m, n)$

Result (applying the convolution Equation):

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

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

A 3x3 grid representing matrix $I(i, j)$. The columns are labeled 0, 1, 2 at the top, and the rows are labeled 0, 1, 2 on the left. The letters a through i are placed in the grid: a(0,0), b(0,1), c(0,2), d(1,0), e(1,1), f(1,2), g(2,0), h(2,1), i(2,2). The row index i and column index j are shown as vectors originating from the top-left corner (0,0) pointing towards their respective row and column.

	j		
i	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

$$I(i, j)$$

A 3x3 grid representing matrix $K(m, n)$. The columns are labeled -1, 0, 1 at the top, and the rows are labeled -1, 0, 1 on the left. The letters i through a are placed in the grid: i(-1,-1), h(-1,0), g(-1,1), f(0,-1), e(0,0), d(0,1), c(1,-1), b(1,0), a(1,1). The row index m and column index n are shown as vectors originating from the top-left corner (-1,-1) pointing towards their respective row and column.

	n		
m	-1	0	1
-1	i	h	g
0	f	e	d
1	c	b	a

$$K(m, n)$$

Cells with the same letter are multiplied together!

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

Trick to speed up computation

A 3x3 grid representing an input matrix $I(i, j)$. The rows are indexed by i (0, 1, 2) and the columns by j (0, 1, 2). The elements are labeled as follows:

	j		
i	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

 $I(i, j)$

A 3x3 grid representing a kernel matrix $K(m, n)$. The rows are indexed by m (-1, 0, 1) and the columns by n (-1, 0, 1). The elements are labeled as follows:

	n		
m	-1	0	1
-1	i	h	g
0	f	e	d
1	c	b	a

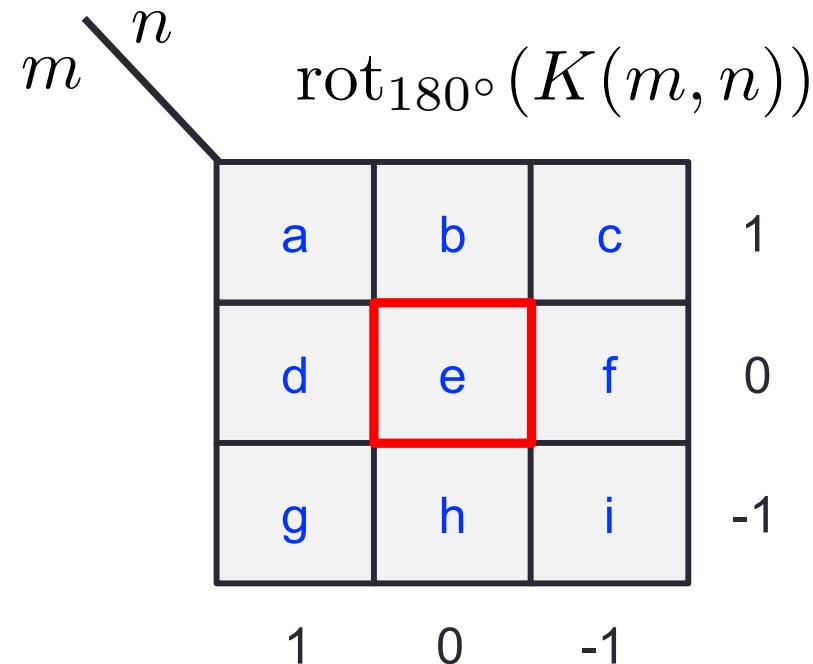
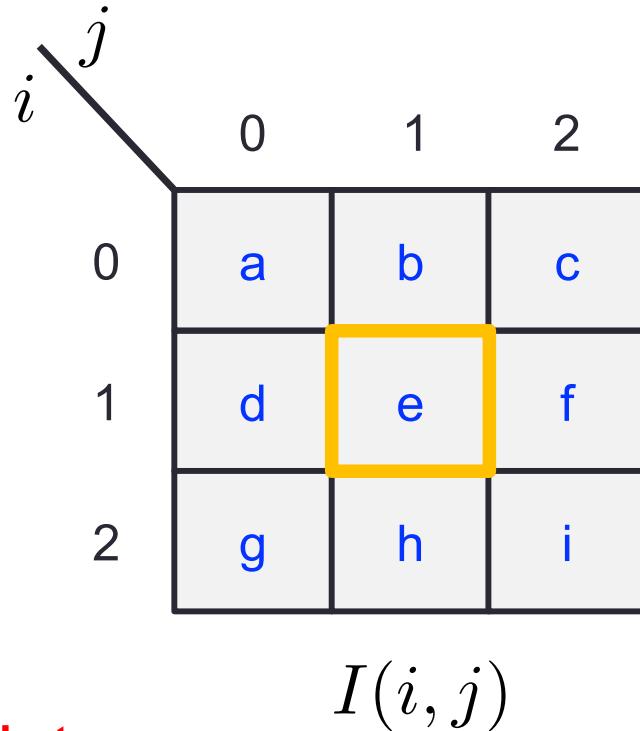
rotate 180 degrees

 $K(m, n)$

STEP 1) Rotate the Kernel 180 degrees

- It is equivalent to time-reversal (in 2D)

Trick to speed up computation



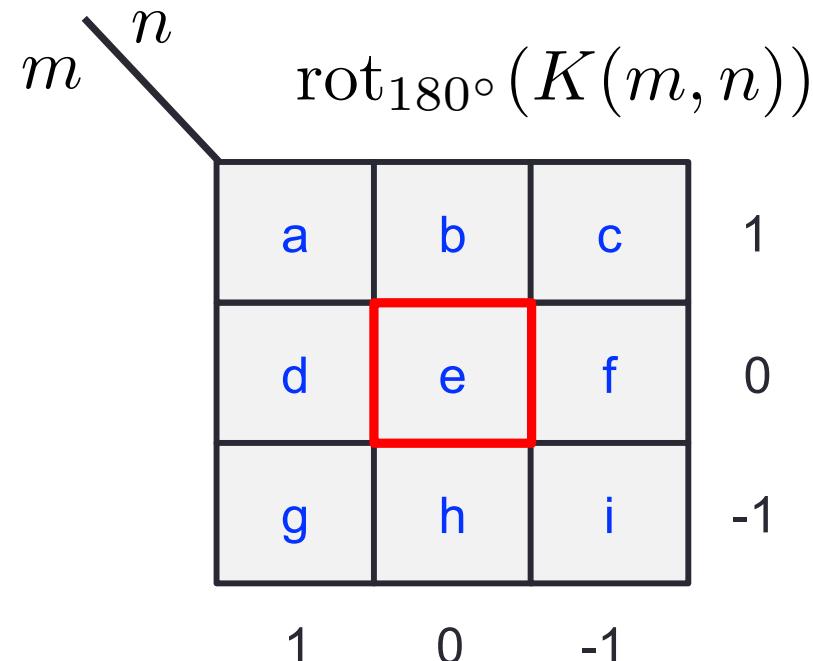
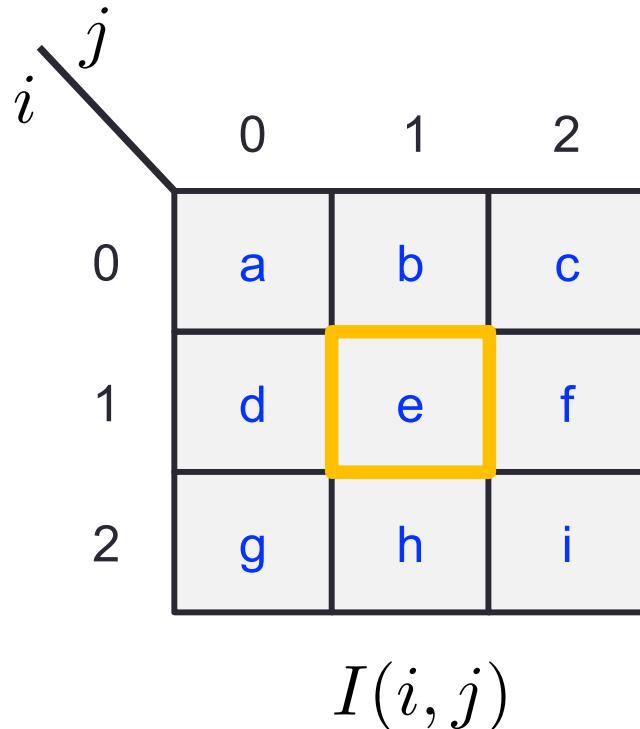
Note

- 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, $S(1, 1) = a^2 + b^2 + c^2 + \dots + i^2$

Computing $S(0,0)$

	i	j
0	0	1
1	2	3
2	4	5
	7	8
	9	

$$I(i, j)$$

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

Computing $S(0,0)$

	j		
i	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

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

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

Convolution

- Flip the Kernel

Computing S(0,0)

			rot _{180°} (K(m, n))			
1	2	1	I(i, j)			
0	0	1	0	2	3	
-1	-2	4	-1	5	6	
	7		8		9	

Convolution

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

Computing $S(0,0)$

			rot _{180°} ($K(m, n)$)			
1	2	1	I(i, j)			
0	0	1	0	2	3	
-1	-2	4	-1	5	6	
7		8		9		

Convolution

- Take element-wise products and sum

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

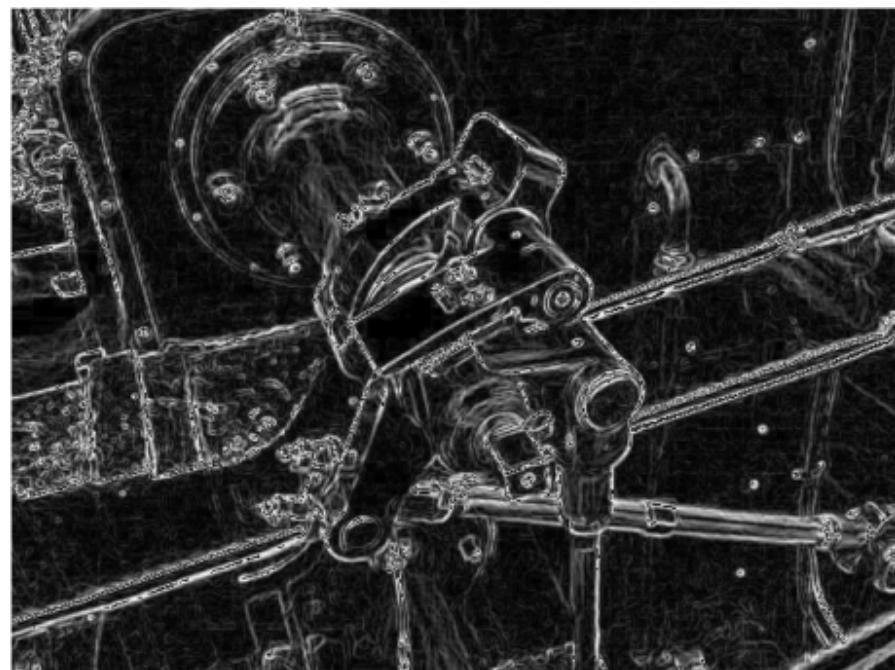
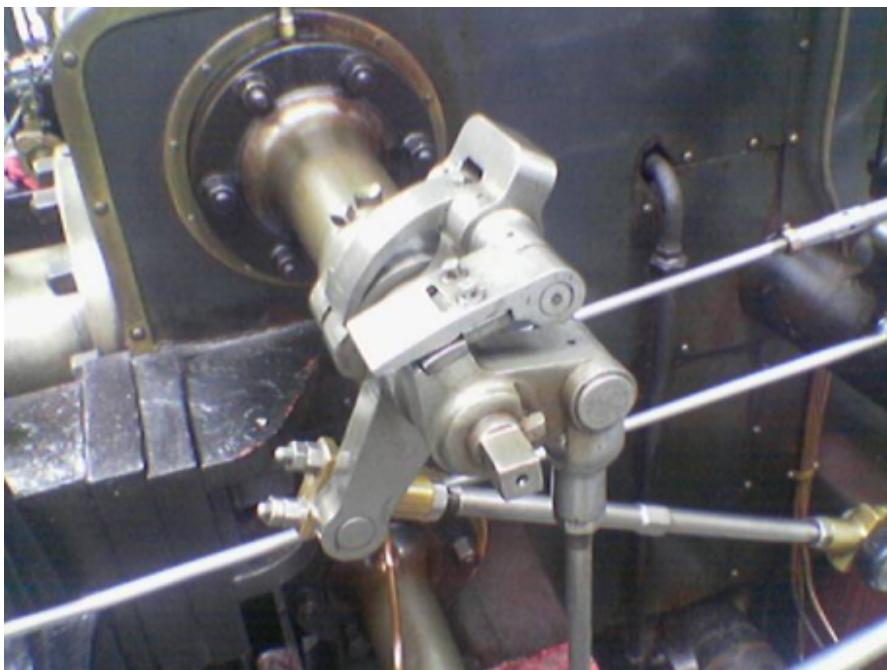
The output matrix for this example

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

$S(i, j)$

Repeating the calculation for all entries

Applying the 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:

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

We have that:

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

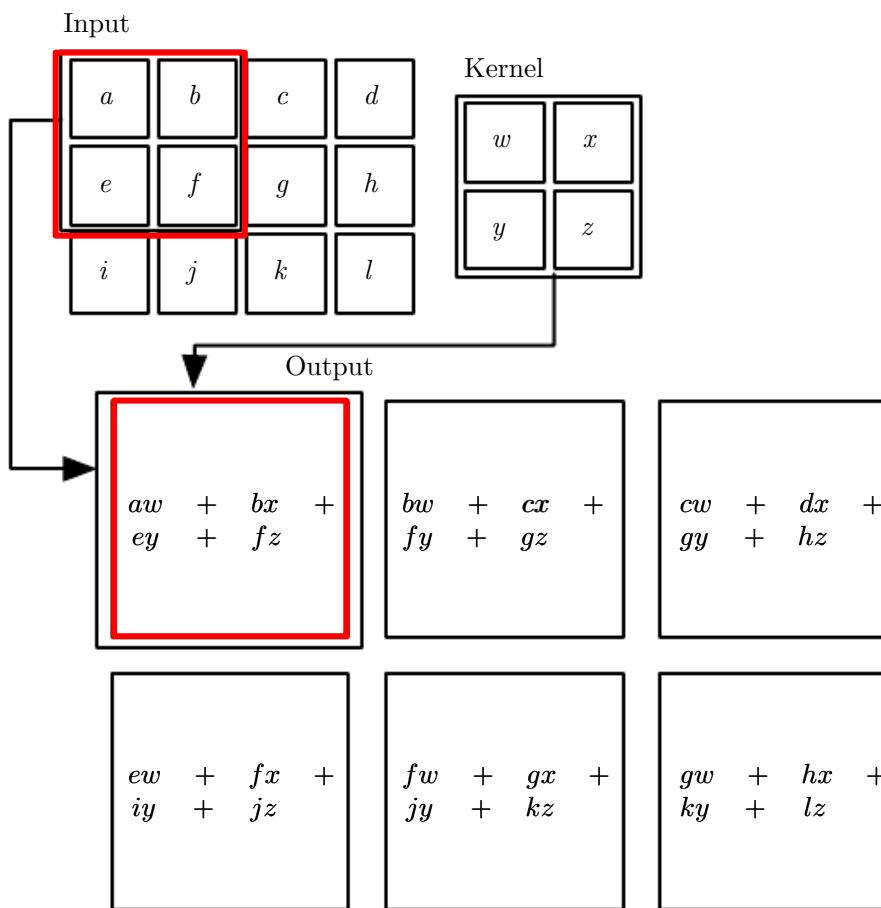
Convolution: equivalent to cross-correlation with a flipped Kernel

Convolution

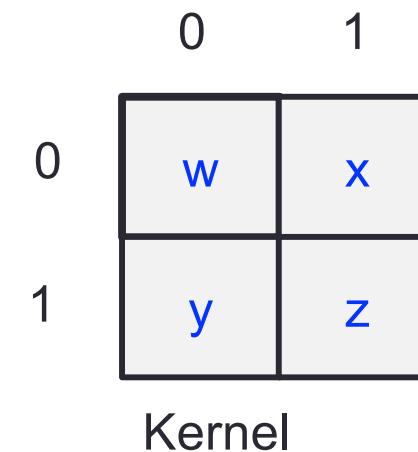
- In machine learning
 - Whether to flip the Kernel or not **is irrelevant**
 - The goal is **to learn a convolution Kernel**
- Simply
 - If flipping is considered → the learned Kernel will be flipped
 - If flipping is not used → the learned Kernel will not be flipped
- The following blocks will not be affected by this

Cross-correlation (1/2)

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

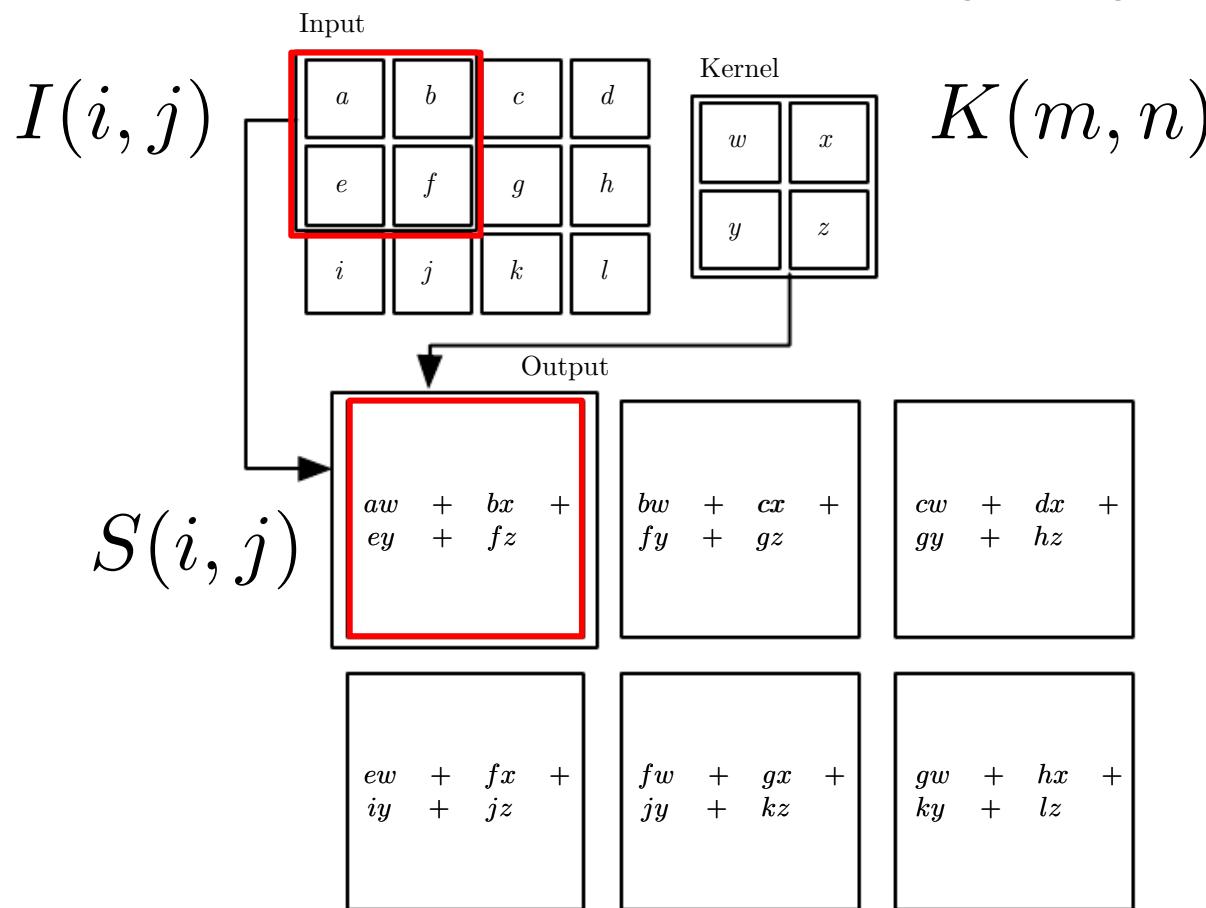


Kernel indexing

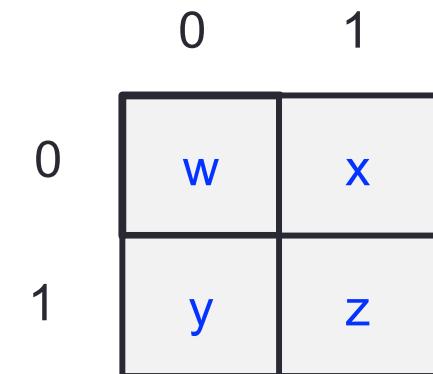


Cross-correlation (2/2)

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

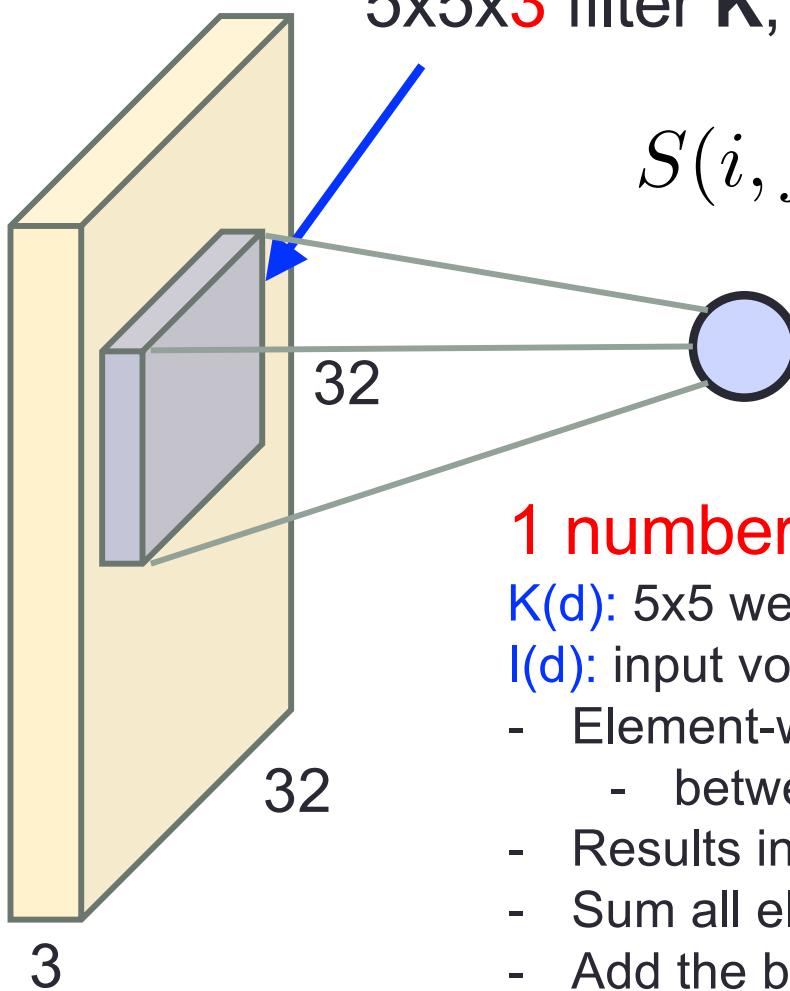


Kernel indexing



Convolution - input volume depth d=3

32x32x3 image \mathbf{I}



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

1 number - also called linear activation

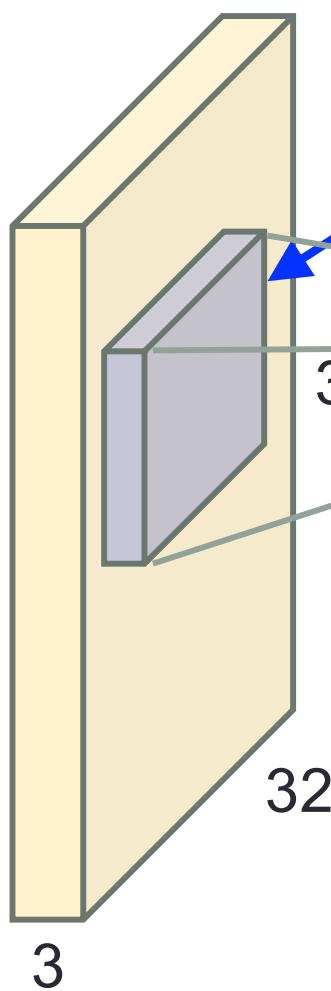
$K(d)$: 5x5 weight matrix along direction d

$I(d)$: input volume along direction d

- Element-wise product (cross-correlation)
 - between image $I(d)$ and $K(d)$
- Results into a 5x5 matrix, for each dimension d
- Sum all elements of all matrices
- Add the bias b

Convolution

32x32x3 image \mathbf{I}



5x5x3 filter \mathbf{K}

32

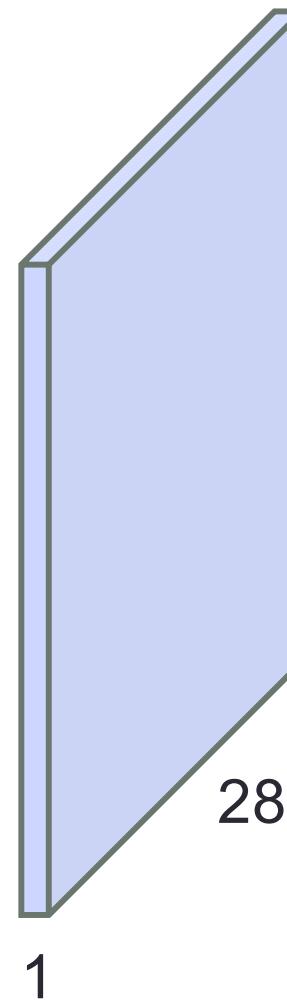
32

3

convolve (slide)
over all spatial
locations

output is 28x28x1

activation map



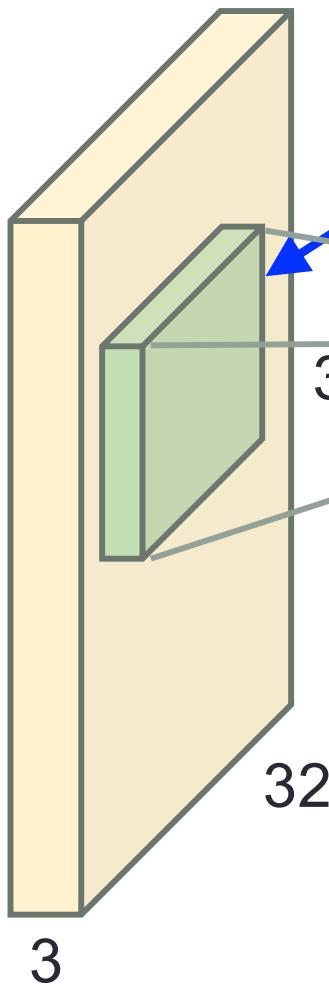
28

28

1

Convolution

32x32x3 image \mathbf{I}



5x5x3 filter \mathbf{K}

32

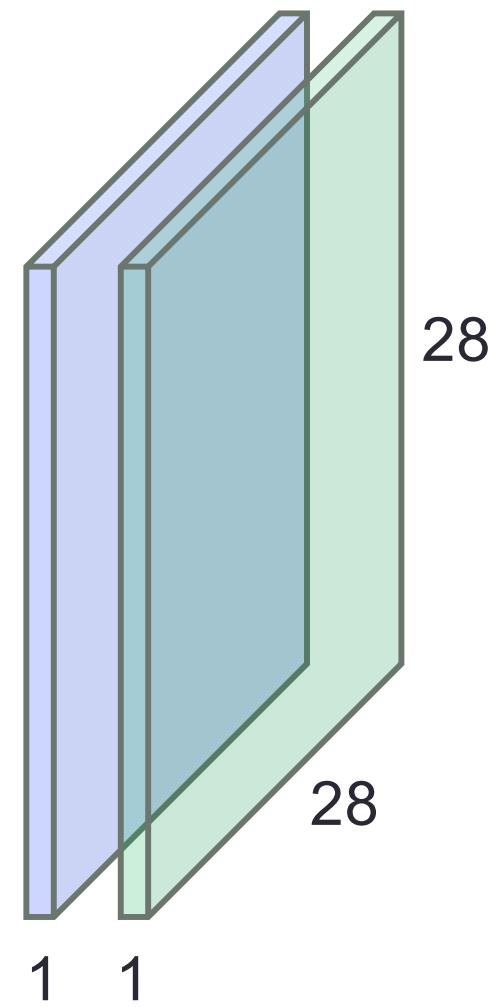
32

3

convolve (slide)
over all spatial
locations

output is 28x28x1

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



28

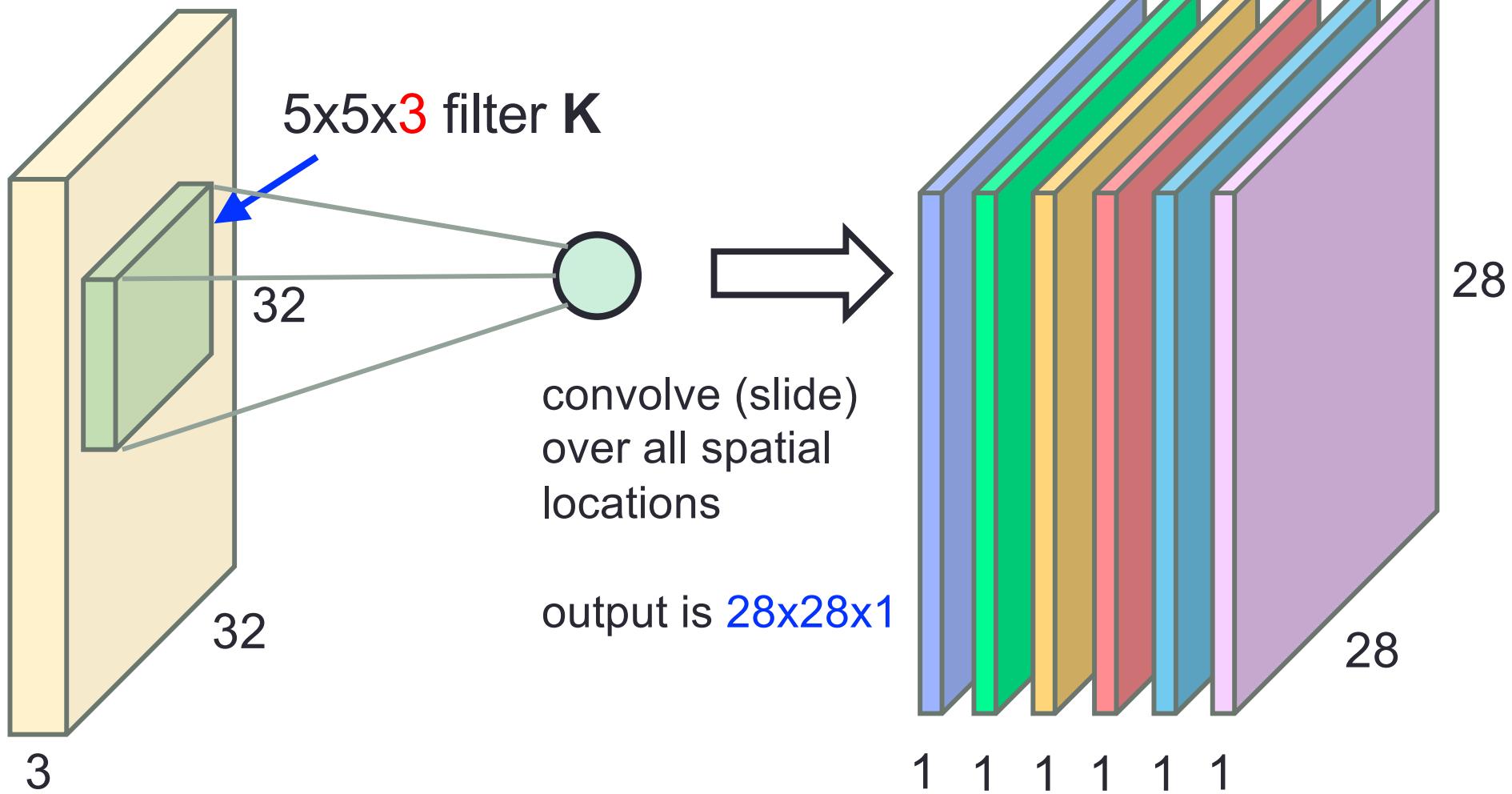
28

1

1

Convolution

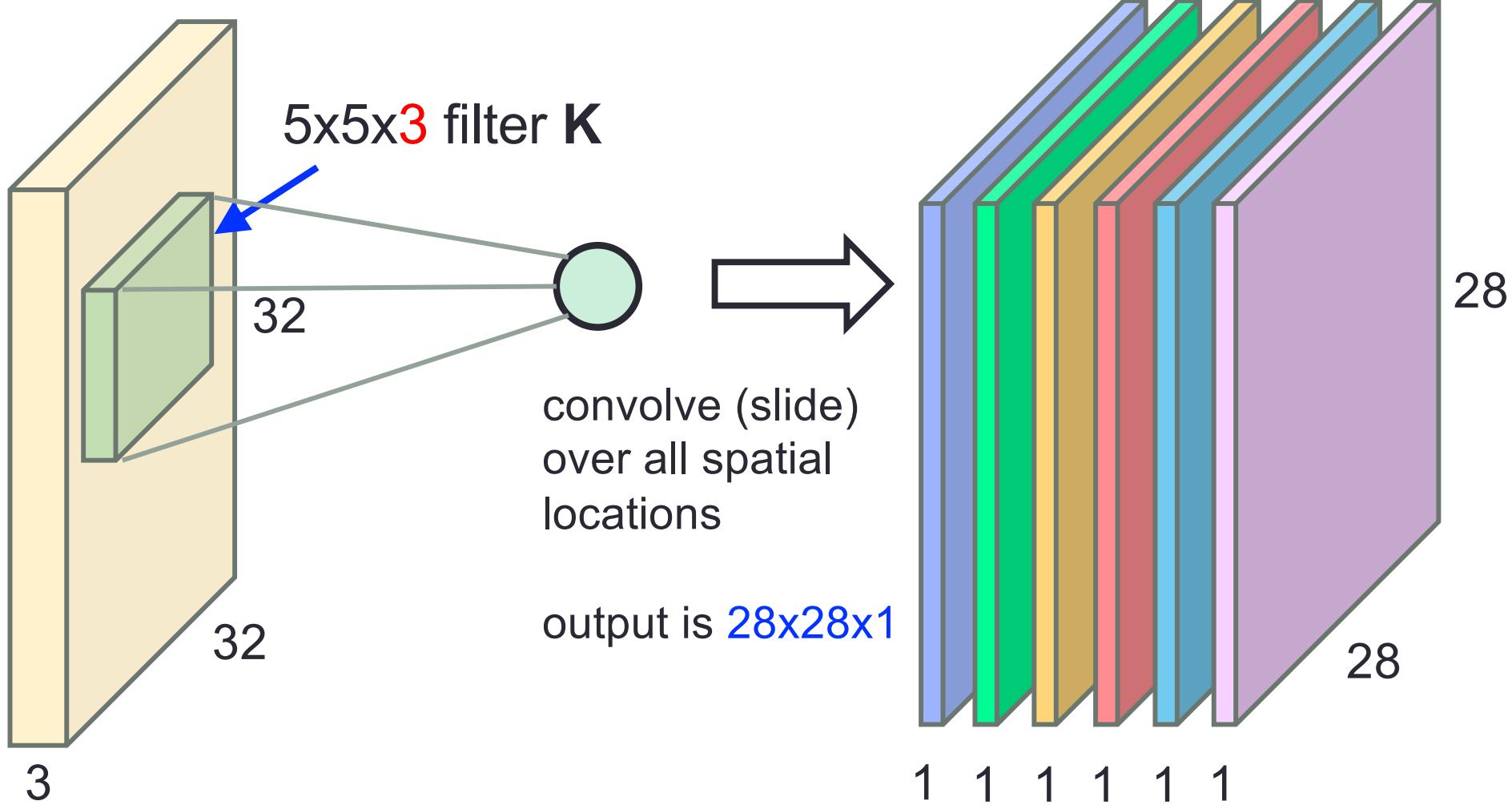
$32 \times 32 \times 3$ image \mathbf{I}



Convolution

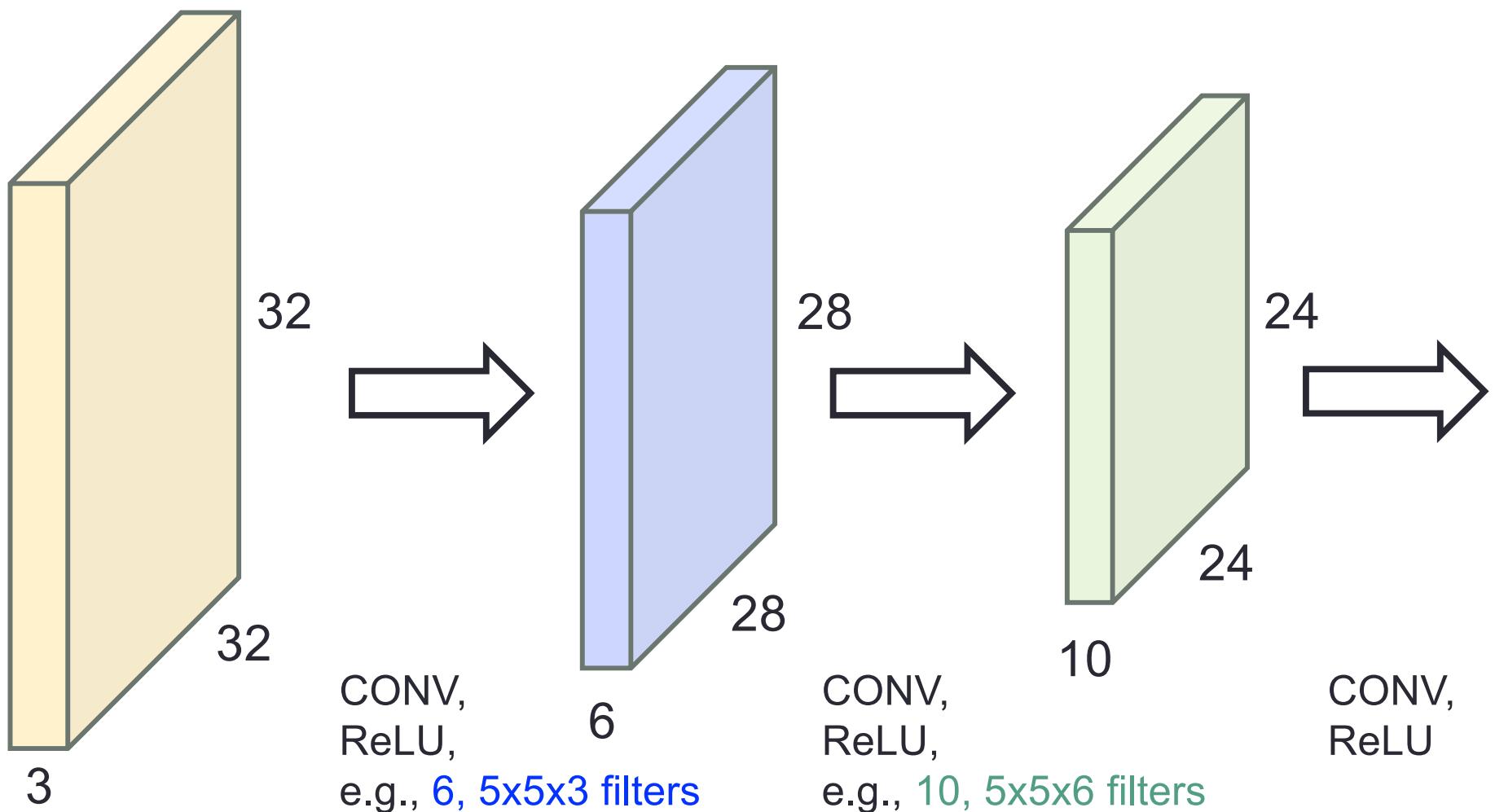
depth of output volume is 6 (6 filters)

$32 \times 32 \times 3$ image \mathbf{I}



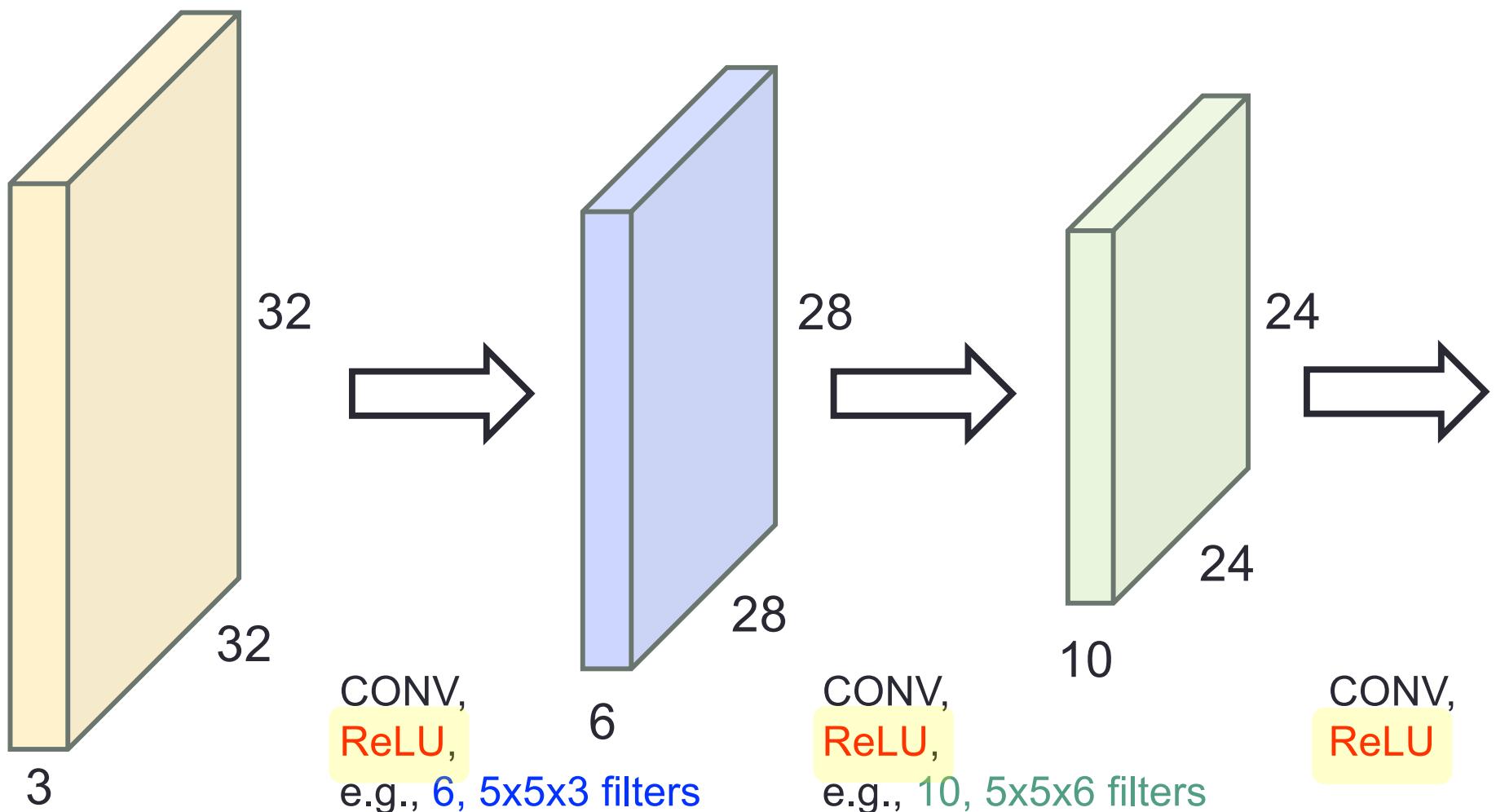
Convolutional network

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

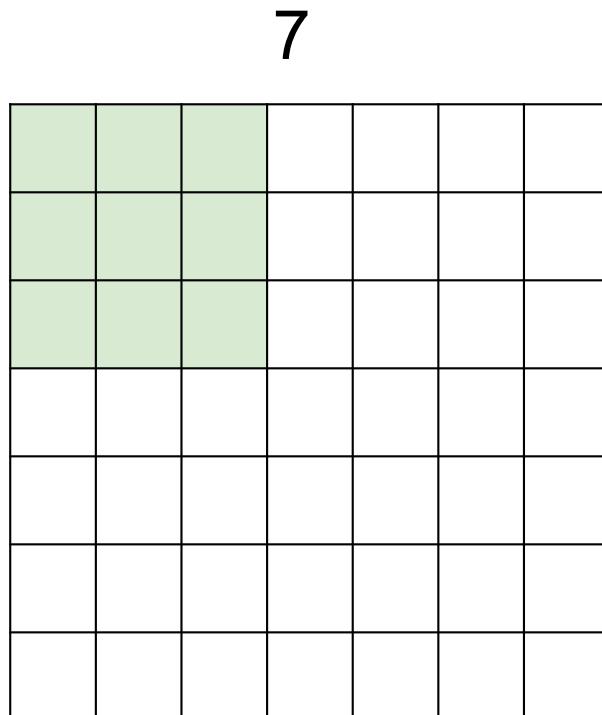


Convolutional network

It is a sequence of convolutions interspersed with activation functions



A closer look at spatial dimensions



STEP 1

7x7(x1) input

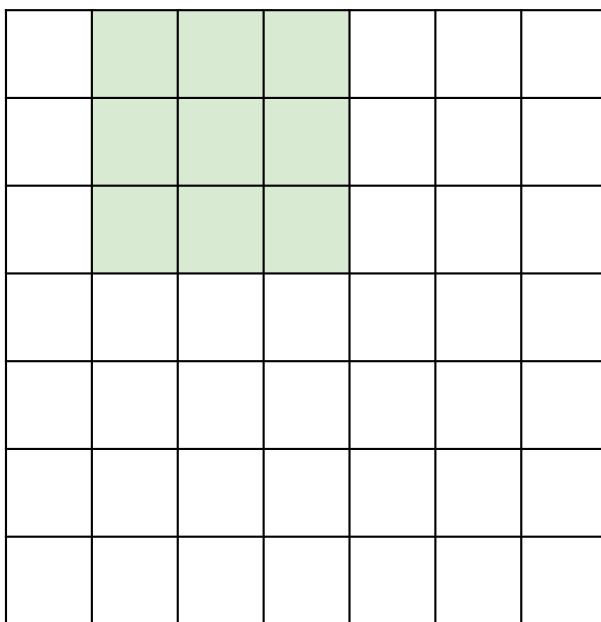
assume a 3x3(x1) green filter

sliding step = 1

sliding step is referred to as **stride**

A closer look at spatial dimensions

7



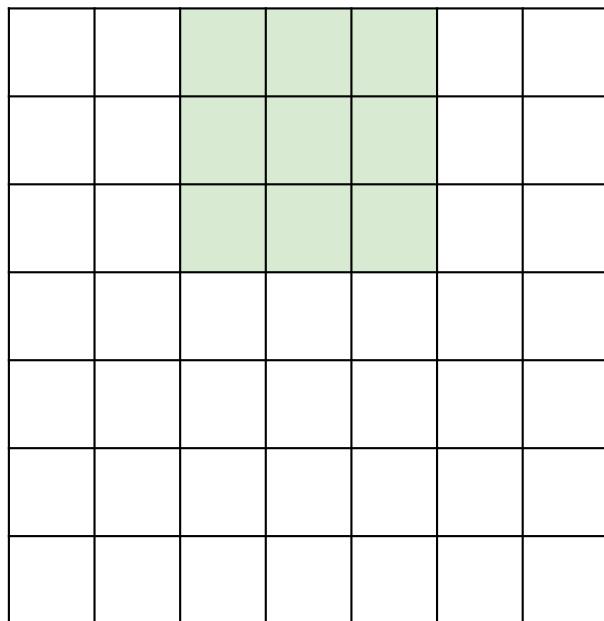
STEP 2

move filter one step to the right

7

A closer look at spatial dimensions

7



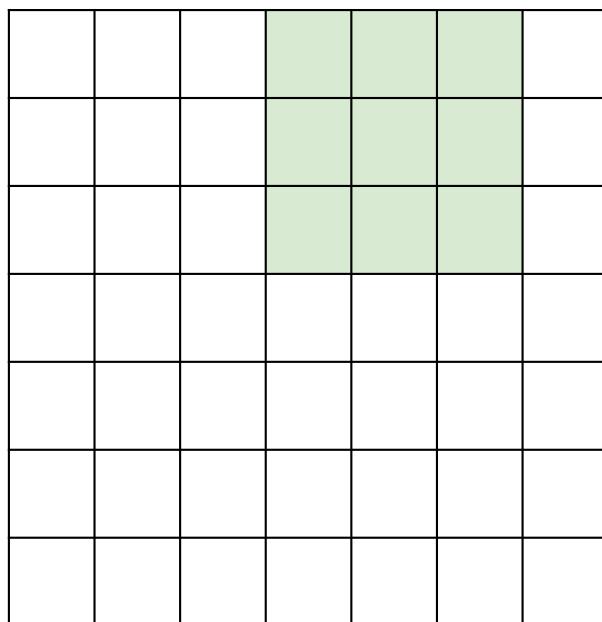
STEP 3

move one more step to the right

7

A closer look at spatial dimensions

7

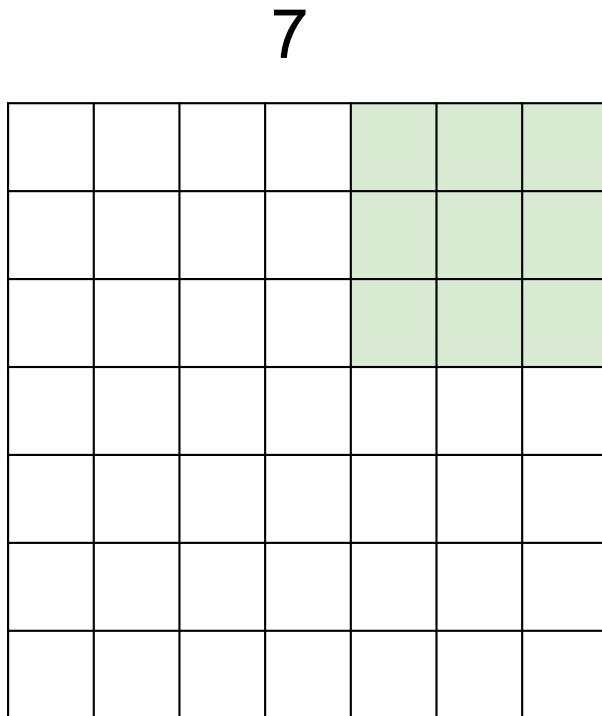


7

STEP 4

move one more step to the right

A closer look at spatial dimensions



STEP 5

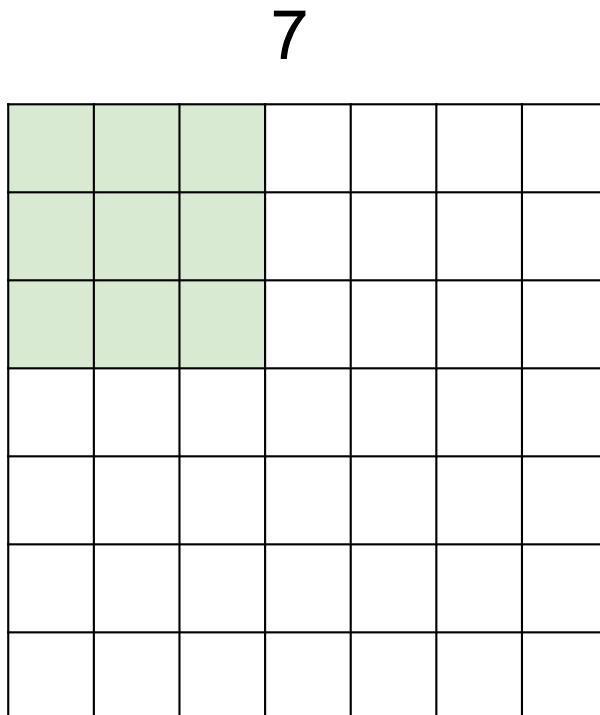
move one more step to the right

7



5x5 output

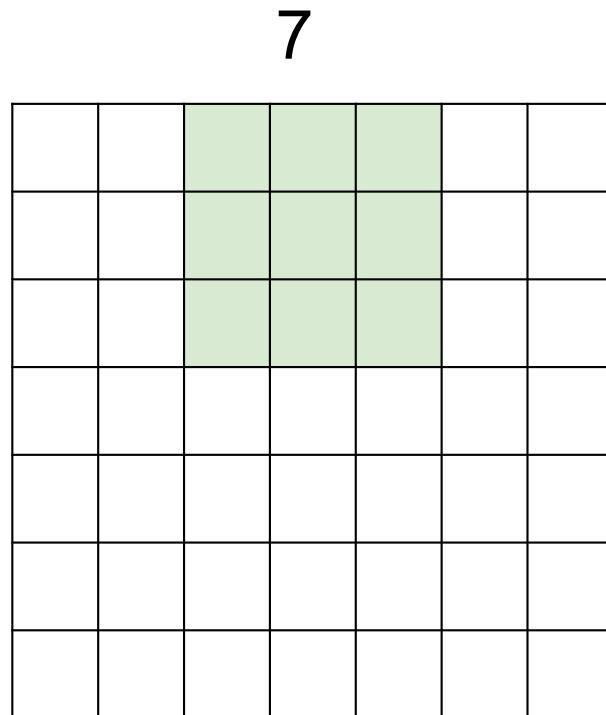
A closer look at spatial dimensions



STEP 1

7x7 spatial input
assume a 3x3 green filter
apply a **stride of 2**

A closer look at spatial dimensions

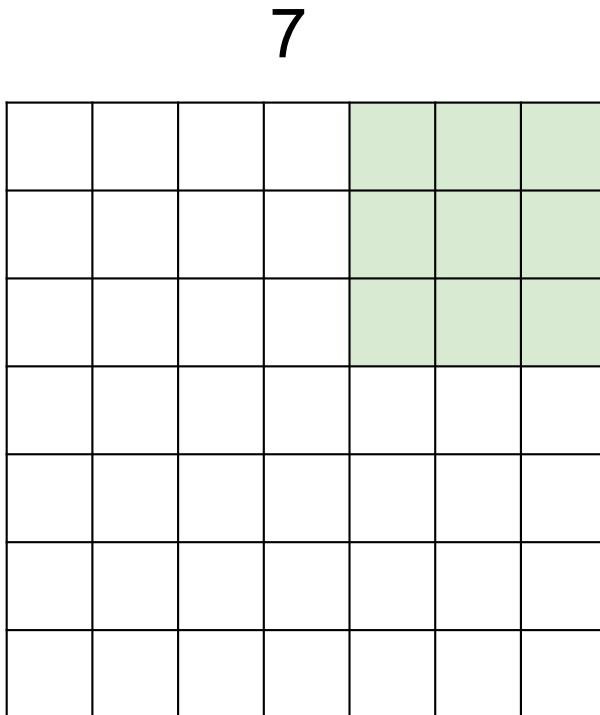


7

STEP 2

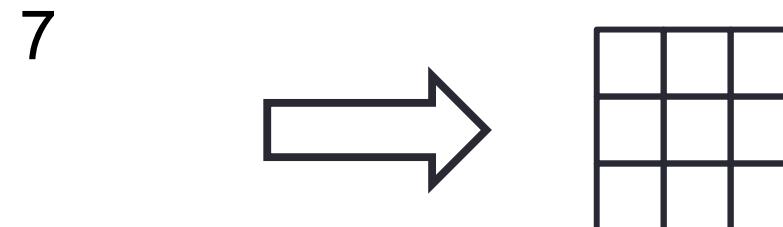
move 2 steps to the right

A closer look at spatial dimensions



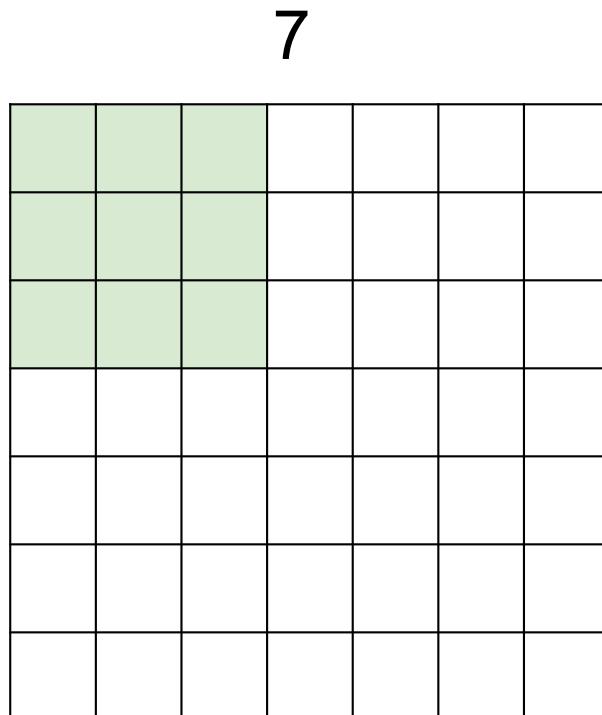
STEP 3

move 2 more steps to the right



3x3 output

Spatial dimensions: stride = 3

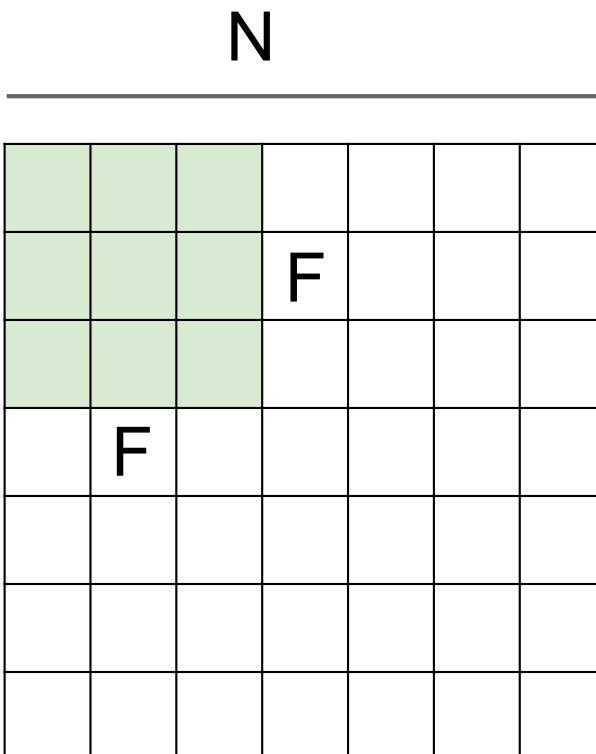


We now try with a stride of 3

Does not fit!

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

Output size vs filter size & stride



Output volume (OxO):

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

e.g., $N=7, F=3$

$$\text{stride}=1, O = (7-3)/1 + 1 = 5$$

$$\text{stride}=2, O = (7-3)/2 + 1 = 3$$

$$\text{stride}=3, O = (7-3)/3 + 1 = 2.33 :-($$

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

Solution

0	0	0	0	0	0			
0								
0								
0								
0								

It is common to zero-pad the border

Solution

If $\text{stride} = 1$, $F \times F$ 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

0	0	0	0	0	0			
0								
0								
0								
0								

P

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	2	0
0	2	2

 $I(1, 1, 1)$

Filter W0 (3x3x3)

1	-1	0
1	1	1
0	1	1

1	1	0
-1	-1	1
1	-1	-1

1	-1	1
-1	0	1
1	1	1

 $x[:, :, 1]$

0	0	0
0	1	1
0	1	2

 $I(1, 1, 2)$

0	1	2
0	2	0
0	1	1

 $x[:, :, 2]$

0	0	0
0	0	0
0	2	1

 $I(1, 1, 3)$

Filter W1 (3x3x3)

-1	1	0
1	-1	-1
0	1	1

0	-1	-1
0	-1	0
0	1	1

0	-1	-1
0	0	0
-1	1	0

1
0

Output Volume (3x3x2)

7	10	-1
7	14	6
3	8	-1

6	3	0
-3	-1	-3
-5	-7	-3

 $S(i, j)$ activation matrix
from first filter

Convolution example

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

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

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

Filter W0 (3x3x3)

 $w0[:, :, 0]$

1	-1	0
1	1	1
0	1	1

$2+2+2=6$

 $w0[:, :, 1]$

1	1	0
-1	-1	1
1	-1	-1

$2+1=3$

 $w0[:, :, 2]$

1	-1	1
-1	0	1
1	1	1

Bias $b0$ (1x1x1) $b0[:, :, 0]$

1

Filter W1 (3x3x3)

 $w1[:, :, 0]$

1	0
1	-1
0	1

 $w1[:, :, 1]$

-1	
0	-1
0	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

$6-3+3+b0=7$

 $o[:, :, 1]$

6	3	0
-3	-1	-3
-5	-7	-3

Bias $b1$ (1x1x1) $b1[:, :, 0]$

0

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)

$x[:, :, 0]$	Filter W_0 (3x3x3)
0 0 0 0 0 0 0	1 -1 0
0 2 0 0 0 0 0	1 1 1
0 2 2 2 1 0 0	0 1 1
0 2 1 1 2 2 0	$w_0[:, :, 1]$
0 1 1 1 2 2 0	1 1 0
0 0 0 1 0 0 0	-1 -1 1
0 0 0 0 0 0 0	1 -1 -1

$x[:, :, 1]$	Filter W_0 (3x3x3)
0 0 0 0 0 0 0	1 -1 1
0 1 1 1 2 1 0	-1 0 1
0 1 2 1 0 0 0	1 1 1
0 1 2 0 2 1 0	$w_0[:, :, 2]$
0 2 0 1 1 1 0	1 -1 1
0 1 1 1 2 2 0	-1 0 1
0 0 0 0 0 0 0	1 1 1

$x[:, :, 2]$	Bias b_0 (1x1x1)
0 0 0 0 0 0 0	$b_0[:, :, 0]$
0 0 0 0 2 0 0	1
0 2 1 1 2 0 0	$w_1[:, :, 0]$
0 1 1 2 1 2 0	-1 1 0
0 1 2 1 2 0 0	$w_1[:, :, 1]$
0 0 0 0 2 1 0	0 -1 -1
0 0 0 0 0 0 0	0 -1 0

Filter W_1 (3x3x3)

$w_1[:, :, 0]$
-1 1 0
1 -1 -1
0 1 1

$w_1[:, :, 1]$
0 -1 -1
0 -1 0
0 1 1

$w_1[:, :, 2]$
0 -1 -1
0 0 0
-1 1 0

Bias b_1 (1x1x1)

$b_1[:, :, 0]$
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

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

 $x[:, :, 1]$

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 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 b1 (1x1x1)

 $b1[:, :, 0]$

0

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	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)

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

$$0[:, :, :, 1] \\ \begin{bmatrix} & 1 & 0 \\ 1 & -1 & 1 \\ & -1 & -1 \end{bmatrix}$$

ias b0 (1x1x1)

$0[::, :, 0]$

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]
```

Output Volume (3x3x2)

$\circ[:, :, 0]$	7	10	-1
7	14	6	
3	8	-1	
$\circ[:, :, 1]$	6	3	0
-3	-1	-3	
-5	-7	-3	

Bias b1 (1x1x1)

```
b1[:, :, 0]
```

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]

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]

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
1 1 0
-1 -1 1
1 -1 -1

w0[::, :, 1]
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
0 -1 -1
0 -1 0
0 1 1

w1[::, :, 1]
0 -1 -1
0 -1 0
0 1 1

Bias b1 (1x1x1)

b1[::, :, 0]
0

Output Volume (3x3x2)

o[::, :, 0]
7 10 -1
7 14 6
3 8 -1
6 3 0
-3 -1 -3
-5 -7 -3

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

Filter W0 (3x3x3)

w0[::,:,0]
1 -1 0
1 1 1
0 1 1
1 1 0
-1 -1 1
1 -1 -1

w0[::,:,1]
1 1 0
-1 0 1
1 1 1
1 -1 1
-1 0 1
1 1 1

Bias b0 (1x1x1)

b0[::,:,0]
1

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

Bias b1 (1x1x1)

b1[::,:,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 W1 (3x3x3)

w1[::,:,0]
-1 1 0
1 -1 -1
0 1 1
1 1 0
-1 -1 1
0 1 1

w1[::,:,1]
0 -1 -1
0 -1 0
0 1 1
0 -1 -1
0 -1 0
0 0 0

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
6 3 0
-3 -1 -3
-5 -7 -3

o[::,:,1]

o[::,:,2]

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0
0	2	2	2	1	0	0	0
0	2	1	1	2	2	0	0
0	1	1	1	2	2	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$	1	-1	0
1	1	1	1
0	1	1	1

$w0[:, :, 1]$	1	1	0
-1	1	1	1
1	-1	-1	-1

$w0[:, :, 2]$	1	-1	1
-1	0	1	1
1	1	1	1

Bias b0 (1x1x1)
 $b0[:, :, 0]$

1

$x[:, :, 1]$	0	0	0	0	0	0	0
0	1	1	1	2	1	0	0
0	1	2	1	0	0	0	0
0	1	2	0	2	1	0	0
0	2	0	1	1	1	0	0
0	1	1	1	2	2	0	0
0	0	1	1	2	2	0	0
0	0	0	0	0	0	0	0

$w0[:, :, 0]$	1	-1	1
1	0	1	1
1	1	1	1

Bias b1 (1x1x1)
 $b1[:, :, 0]$

Filter W1 (3x3x3)

$w1[:, :, 0]$	-1	1	0
1	-1	-1	-1
0	1	1	1

$w1[:, :, 1]$	0	-1	-1
0	-1	0	0
0	1	1	1

$w1[:, :, 2]$	0	-1	-1
0	0	0	0
-1	1	0	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	

Bias b0 (1x1x1)
 $b0[:, :, 0]$

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 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 2 1 0 0
0 0 0 0 0 0 0

Filter W0 (3x3x3)

w0[::, :, 0]
1 -1 0
1 1 1
0 1 1
1 1 0

w0[::, :, 1]
1 1 0
-1 -1 1
1 -1 -1
1 -1 1

w0[::, :, 2]
1 -1 1
-1 0 1
1 1 1
1 -1 1

Bias b0 (1x1x1)
b0[::, :, 0]

1

Filter W1 (3x3x3)

w1[::, :, 0]
-1 1 0
1 -1 -1
0 1 1
0 -1 -1

w1[::, :, 1]
0 -1 -1
0 -1 0
0 1 1
0 -1 -1

w1[::, :, 2]
0 -1 -1
0 0 0
-1 1 0
0 -1 -1

Bias b1 (1x1x1)
b1[::, :, 0]

0

Output Volume (3x3x2)

o[::, :, 0]
7 10 -1
7 14 6
3 8 -1
6 3 0

o[::, :, 1]
6 3 0
-3 -1 -3
-5 -7 -3
-3 -1 -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

Filter W0 (3x3x3)

w0[:, :, 0]
1 -1 0
1 1 1
0 1 1
1 1 0
-1 -1 1
1 -1 -1

w0[:, :, 1]
1 -1 1
-1 0 1
1 1 1
1 -1 1
-1 0 1
1 1 -1

Bias b0 (1x1x1)

b0[:, :, 0]
1

x[:, :, 1]

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

w0[:, :, 2]
1 -1 1
-1 0 1
1 1 1
1 -1 1
-1 0 1
1 1 -1

w1[:, :, 0]
-1 1 0
1 -1 -1
0 1 1
0 -1 -1
0 -1 0
0 1 1

Bias b1 (1x1x1)

b1[:, :, 0]
0

Filter W1 (3x3x3)

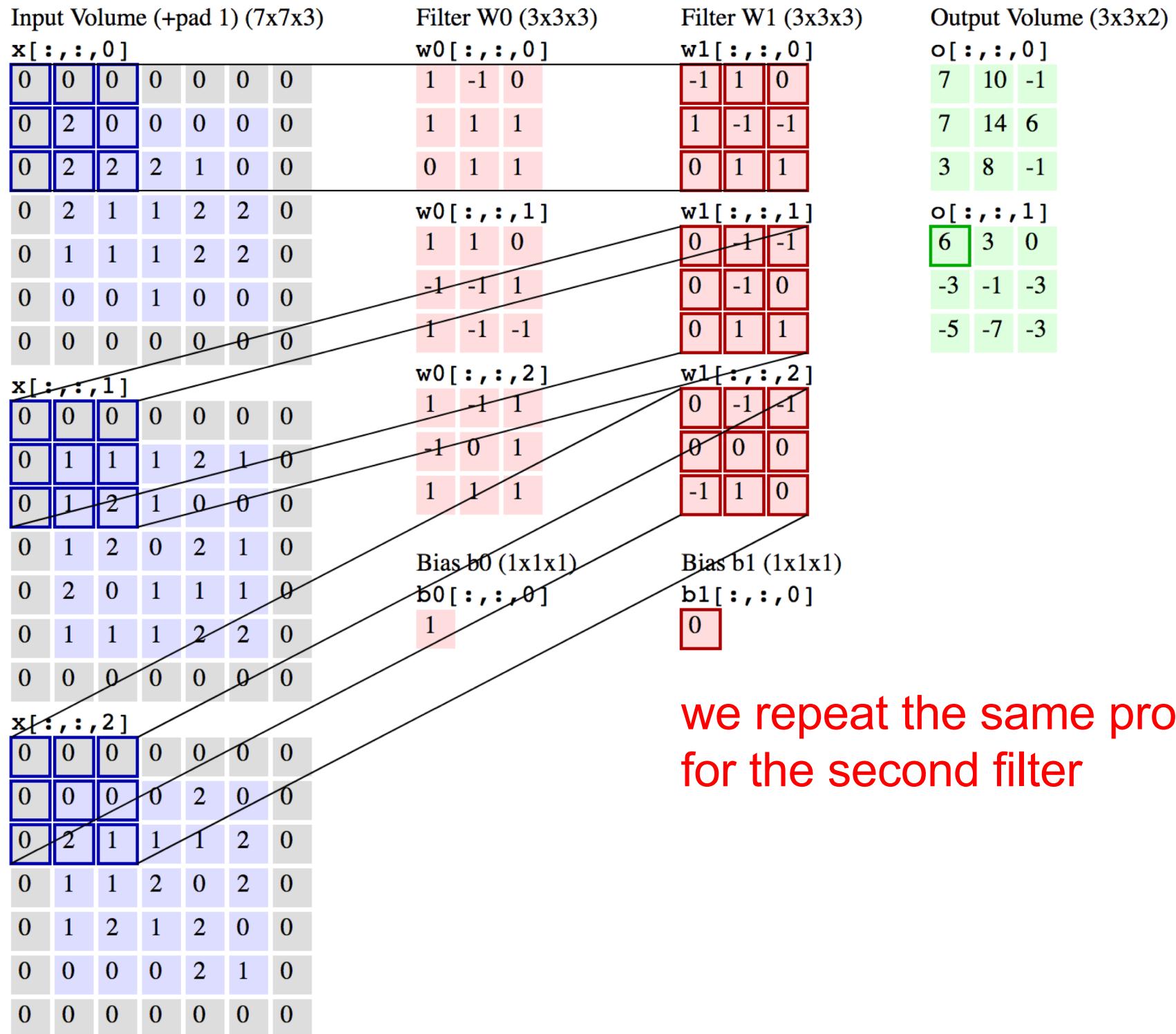
w1[:, :, 1]
0 -1 -1
0 -1 0
0 1 1
0 -1 -1
0 -1 0
0 1 1

w1[:, :, 2]
0 -1 -1
0 0 0
-1 1 0
0 -1 -1
0 0 0
-1 1 0

Output Volume (3x3x2)

o[:, :, 0]
7 10 -1
7 14 6
3 8 -1
6 3 0
-3 -1 -3
-5 -7 -3

o[:, :, 1]
0 -1 -1
0 -1 0
0 1 1
0 -1 -1
0 0 0
-1 1 0



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

Filter W0 (3x3x3)

 $w0[:, :, 0]$

1	-1	0
1	1	1
0	1	1

Filter W1 (3x3x3)

 $w1[:, :, 0]$

-1	1	0
1	-1	-1
0	1	1

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

 $x[:, :, 1]$

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

 $w0[:, :, 1]$

1	1	0
-1	-1	1
1	-1	-1

 $w1[:, :, 1]$

0	-1	-1
0	0	0
-1	1	0

Bias b0 (1x1x1)
 $b0[:, :, 0]$

1

Bias b1 (1x1x1)
 $b1[:, :, 0]$

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

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0
0	2	2	2	2	1	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$	1	-1	0
1	1	1	1
0	1	1	1

Filter W1 (3x3x3)

$w1[:, :, 0]$	-1	1	0
1	-1	-1	-1
0	1	1	1

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	

 $x[:, :, 1]$

$x[:, :, 1]$	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	

 $w0[:, :, 1]$

$w0[:, :, 1]$	1	1	0
-1	-1	1	
1	-1	-1	
-1	0	1	

 $w1[:, :, 1]$

$w1[:, :, 1]$	0	-1	-1
0	-1	0	
0	1	1	
-1	1	0	

Bias b0 (1x1x1)

 $b0[:, :, 0]$

1

Bias b1 (1x1x1)

 $b1[:, :, 0]$

0

 $x[:, :, 2]$

$x[:, :, 2]$	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	

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** 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 characteristics
 - 1) Sparse interactions
 - 2) Parameter sharing
 - 3) Equivalent representations

Sparse interactions

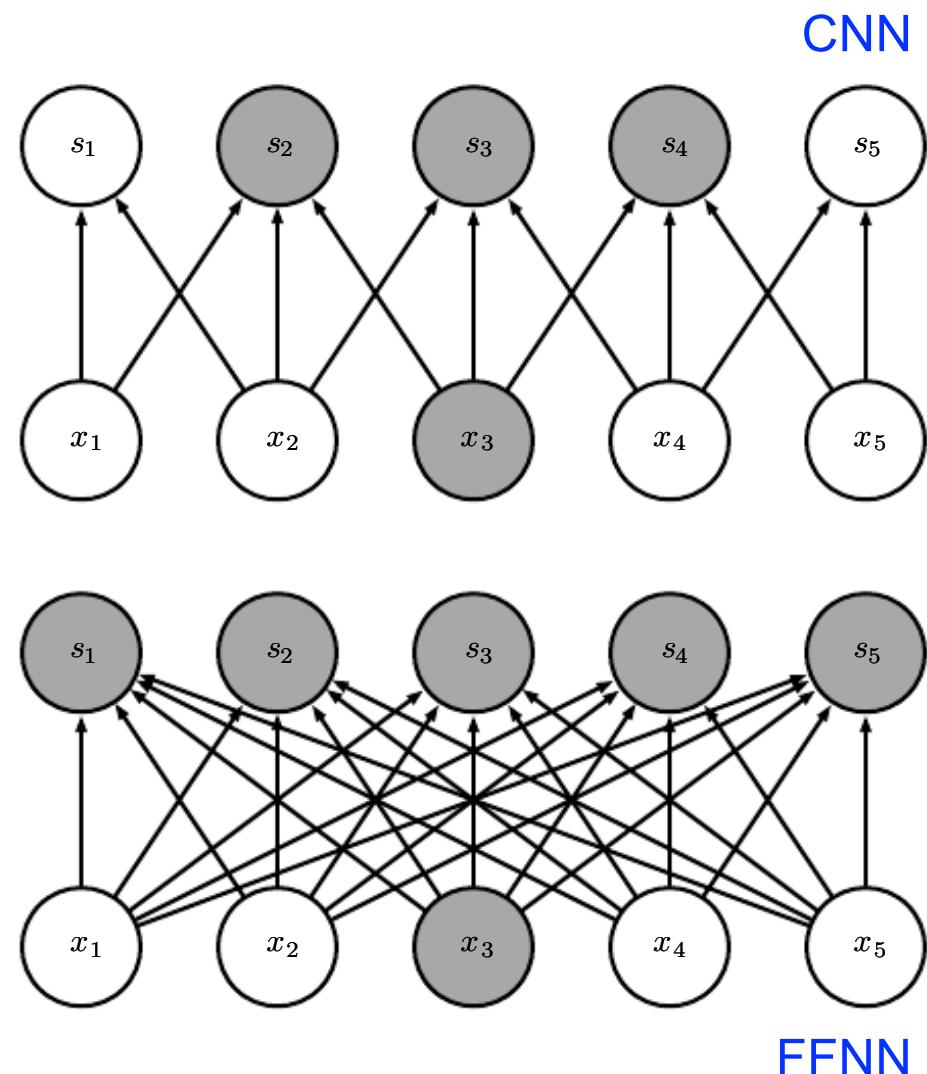
- Traditional FFNNs
 - Connectivity structure is dense: from one layer to the next, there is a link (weight) connecting each neuron in the previous layer to each neuron in the next one, this means that the structure is **fully connected**
 - This means that each input neuron interacts with each output 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*

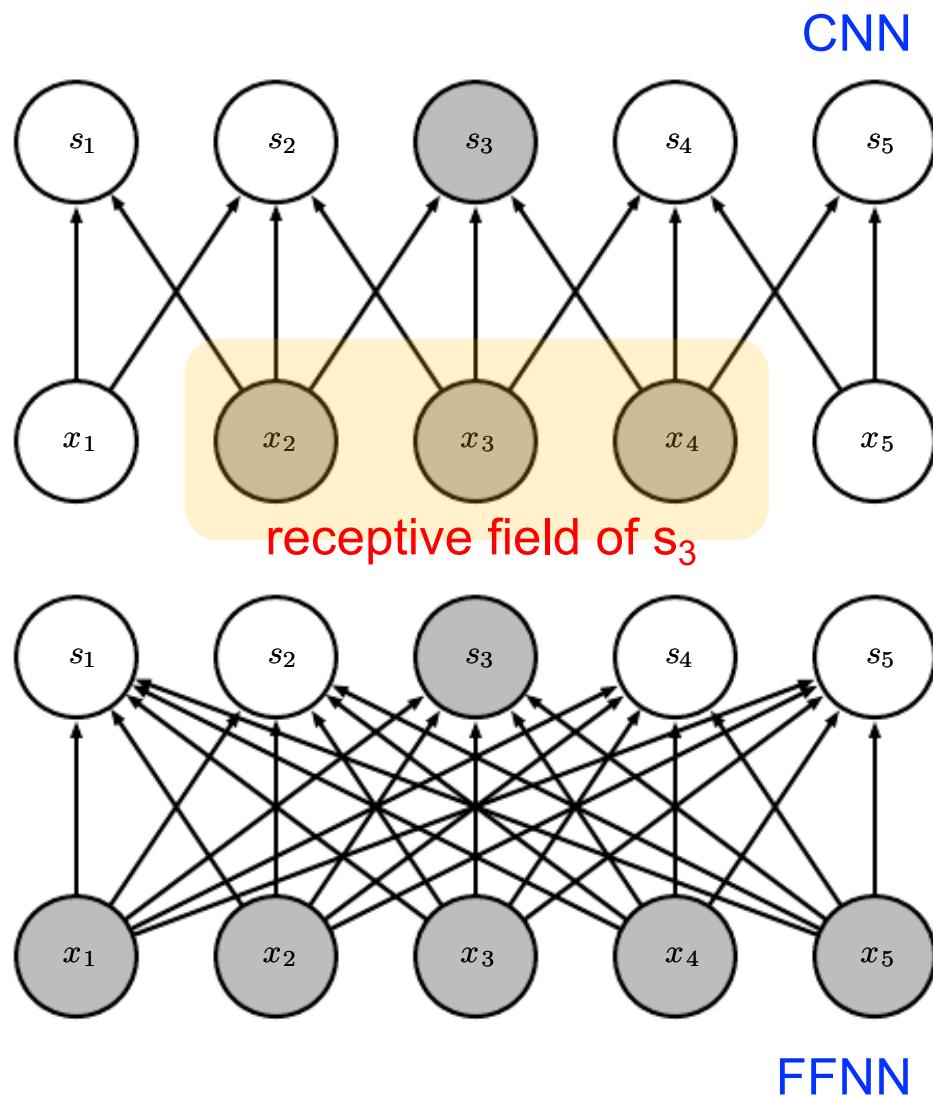
Sparse connectivity (1/2)

- **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: $S=1$, $F=3$



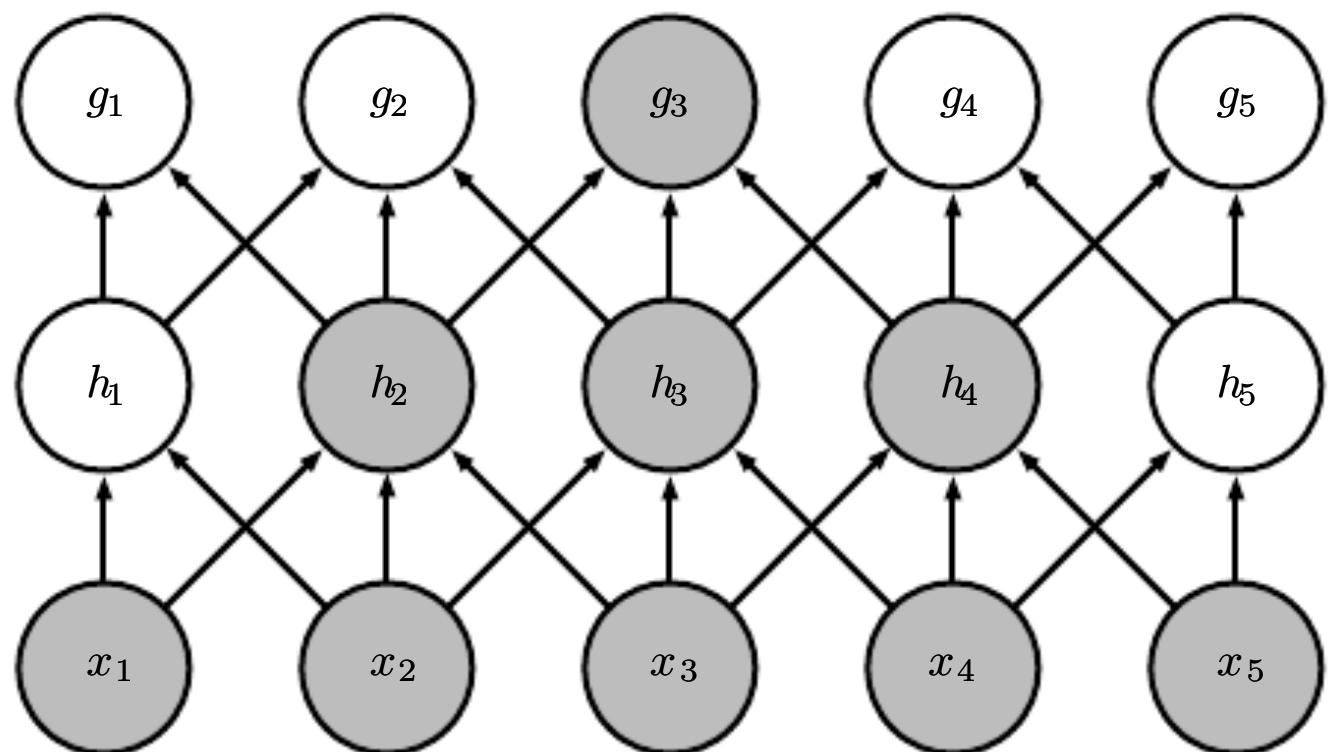
Sparse connectivity (2/2)

- **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**



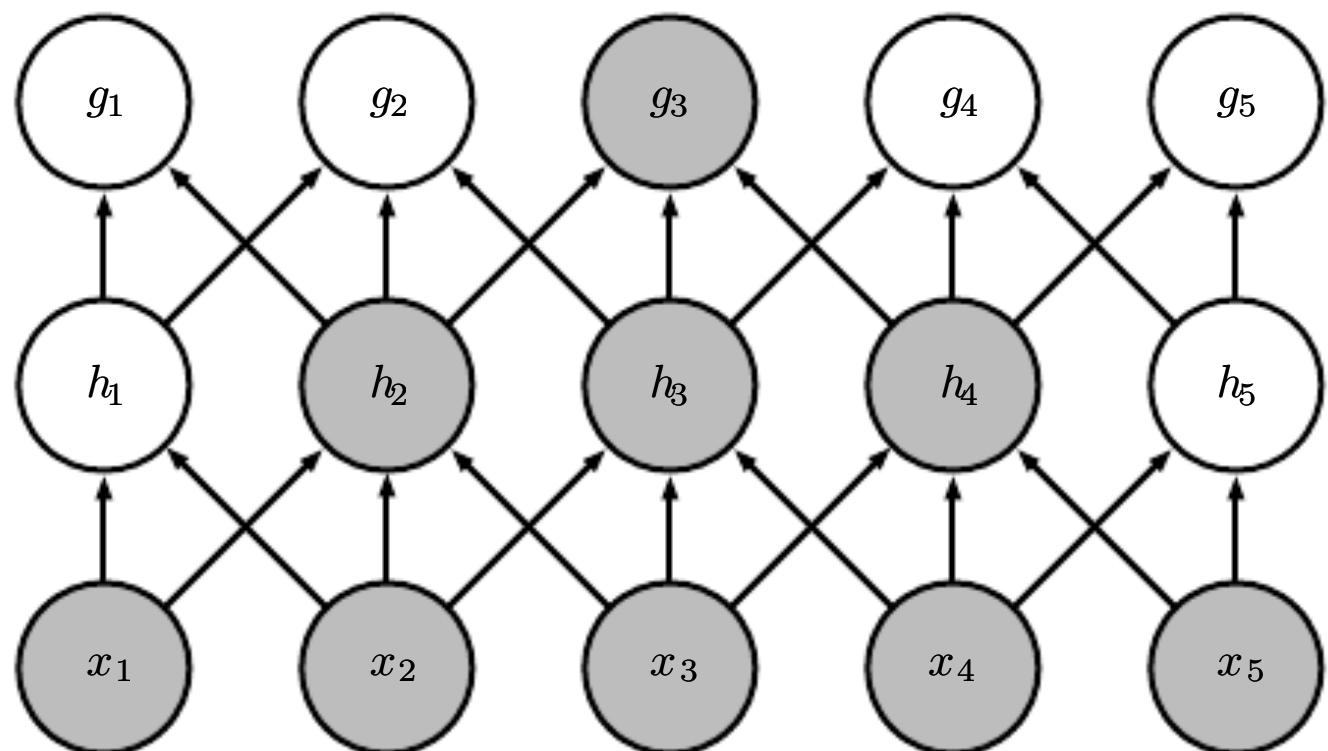
Receptive fields in deep networks (1/2)

- 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



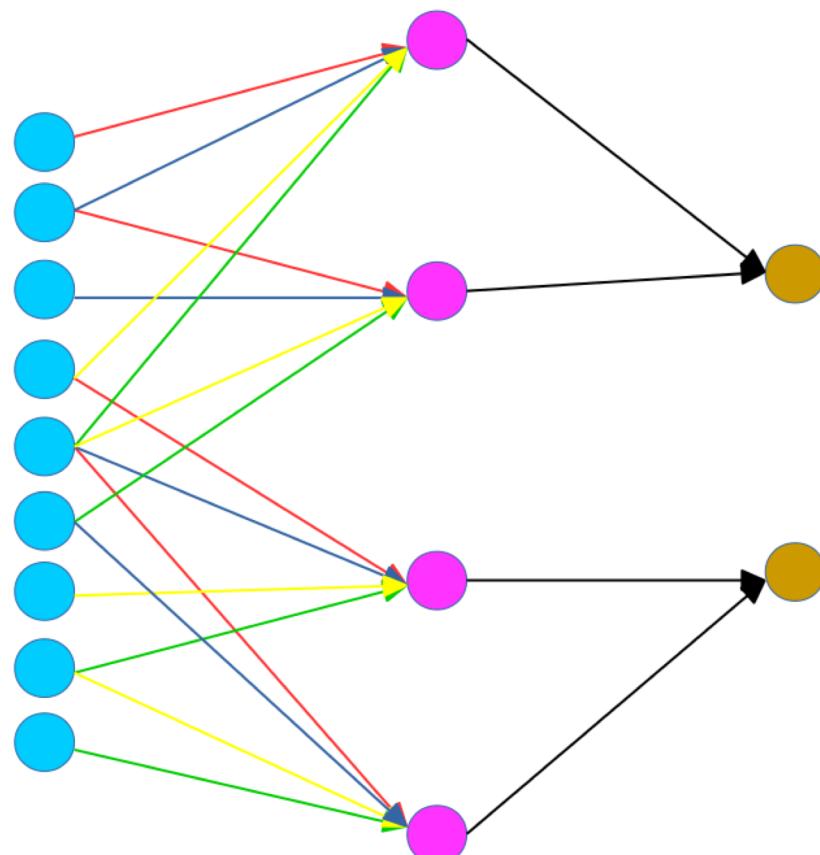
Parameters sharing

- 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



$$\begin{array}{c} \text{Input Feature Map} \\ \times \quad \text{Kernel} \\ = \quad \text{Conv.} \end{array} \qquad \qquad \text{Pool}$$

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

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

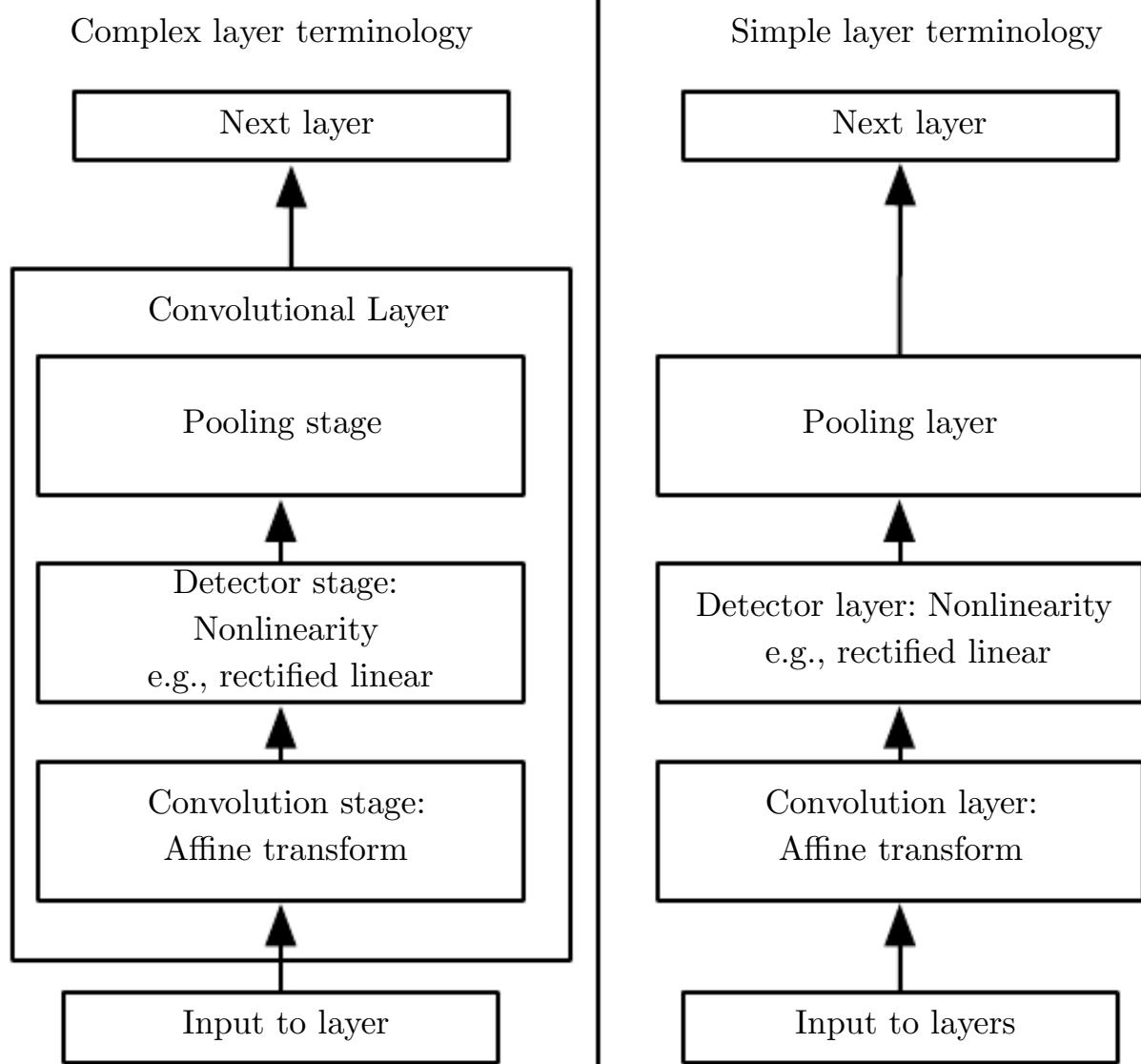
- common
- small no. of complex layers

Right

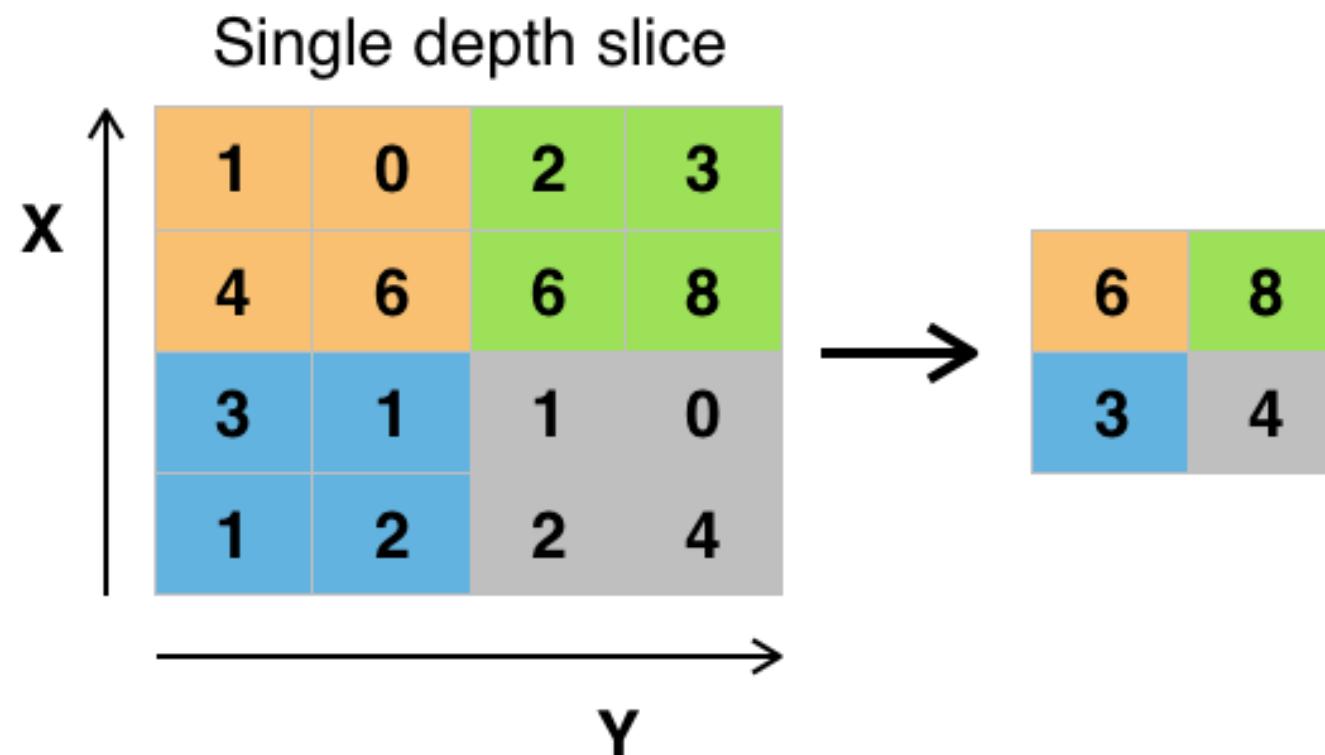
- less common
- Large no. of simple layers

Pooling

- May or may not be included after the nonlinearity



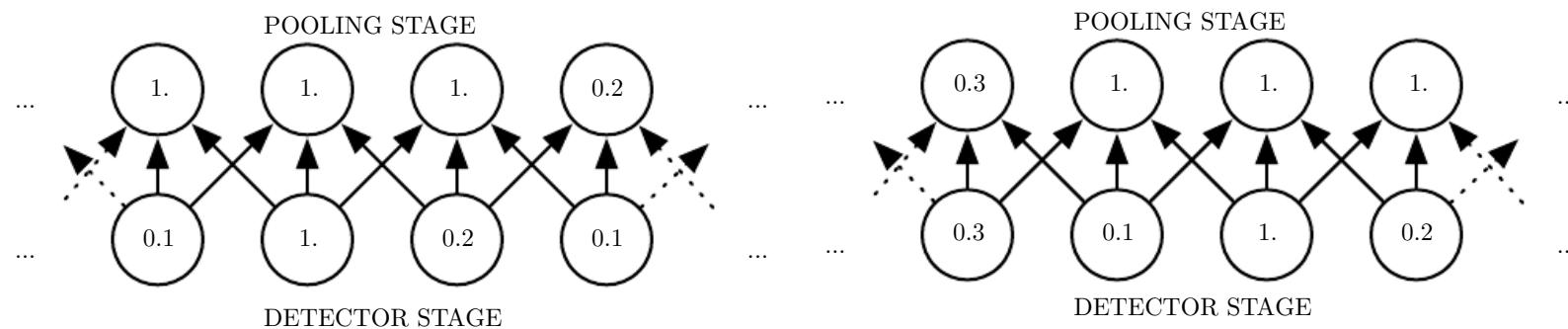
Max pooling example



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

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

BACKPROPAGATION FOR CNN

Notation

1. $\ell = 1, 2, \dots, L$ is the ℓ^{th} layer
2. Input feature map of dimension $H \times W$ with iterators i and j
3. Kernel \mathbf{w} is of dimension $k_1 \times k_2$ and has m and n as iterators
4. $w_{m,n}^\ell$ is the weight connecting neurons from layer $\ell - 1$ to layer ℓ
5. b^ℓ is the bias unit at layer ℓ
6. $a_{i,j}^\ell$ is the activation at layer ℓ : input map is the output from previous layer
$$a_{i,j}^\ell = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} w_{m,n}^\ell y_{i+m,j+n}^{\ell-1} + b^\ell \quad \text{cross-correlation}$$
7. $y_{i,j}^\ell$ is the output at layer ℓ given by : weights $w_{m,n}$ and b only depend on layer no. They are shared across all input values
$$y_{i,j}^\ell = f(a_{i,j}^\ell)$$
8. $f(\cdot)$ is the **activation function**

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

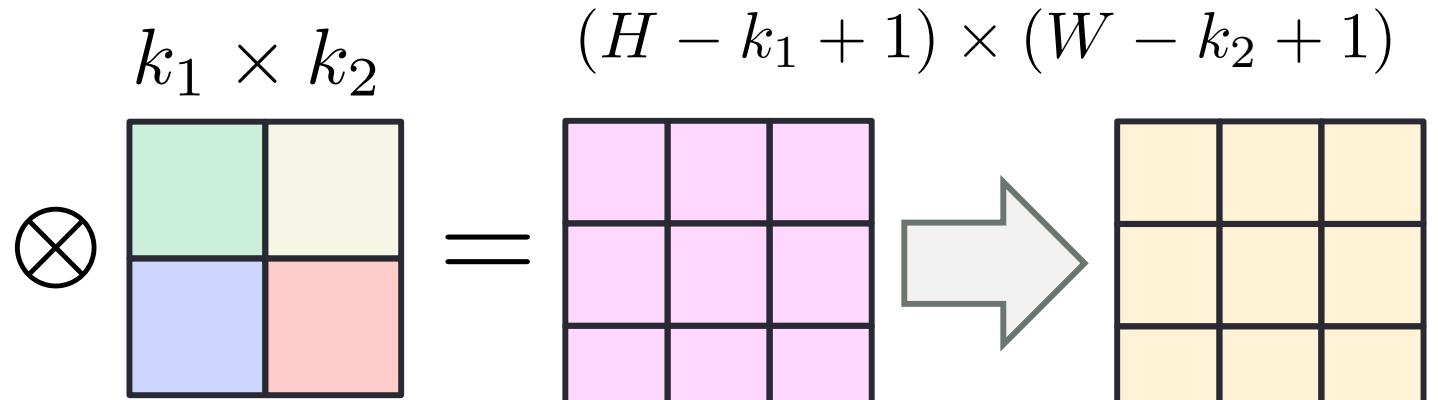
element-wise application of
activation function

$H \times W$

y_{11}	y_{12}	y_{13}	y_{14}
y_{21}	y_{22}	y_{23}	y_{24}
y_{31}	y_{32}	y_{33}	y_{34}
y_{41}	y_{42}	y_{43}	y_{44}

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

input feature map =
output of previous layer



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

Kernel (or filter)

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

activation

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

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

Point-wise error gradient

- For each input pattern we need to compute:

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

- This quantifies
 - The variation of the output error
 - 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”)

UPDATING THE WEIGHTS

Gradient wrt weights

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

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

- 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: $\frac{\partial a_{i,j}^\ell}{\partial w_{m,n}^\ell} = ?$

Being: $a_{i,j}^\ell = \sum_{m'} \sum_{n'} w_{m',n'}^\ell y_{i+m',j+n'}^{\ell-1} + b^\ell$

We get:
$$\frac{\partial a_{i,j}^\ell}{\partial w_{m,n}^\ell} = y_{i+m,j+n}^{\ell-1}$$

That substituting into (1) leads to:

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

Derivative of activations wrt weights

- So far:

$$\frac{\partial E(\mathbf{w})}{\partial w_{m,n}^\ell} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^\ell y_{i+m,j+n}^{\ell-1}$$

- This is precisely the **cross-correlation**

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

Matrix of deltas
size: $(H-k_1+1) \times (W-k_2+1)$

Patch of size $(H-k_1+1) \times (W-k_2+1)$
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}

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

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

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

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

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



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

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

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

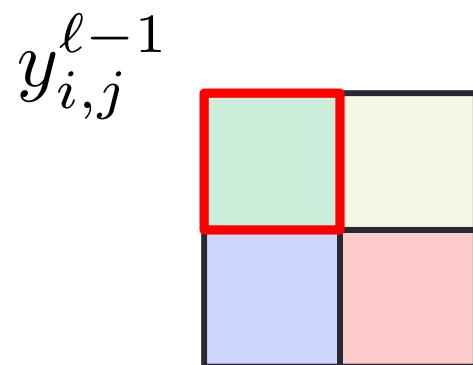
Updating the kernel weights (2/5)

$H \times W$

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}



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



Point-wise gradient descent:

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

Weight matrix (Kernel)

Updating the kernel weights (3/5)

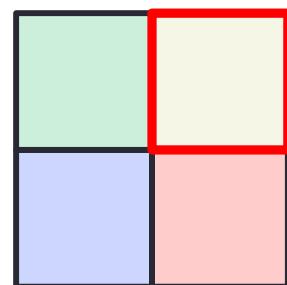
$H \times W$

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}



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

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



Point-wise gradient descent:

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

Weight matrix (Kernel)

Updating the kernel weights (4/5)

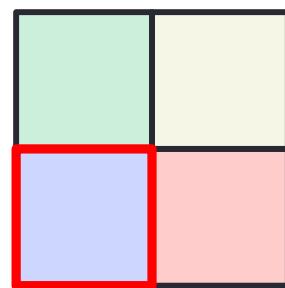
$H \times W$

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}



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

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



Point-wise gradient descent:

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

Weight matrix (Kernel)

Updating the kernel weights (5/5)

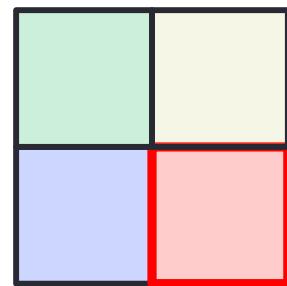
$H \times W$

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}



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

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



Point-wise gradient descent:

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

Weight matrix (Kernel)

UPDATING THE DELTAS

Gradient of $E(\mathbf{w})$ wrt activations

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

$$\begin{aligned}\delta_{i,j}^\ell &\triangleq \frac{\partial E(\mathbf{w})}{\partial a_{i,j}^\ell} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \frac{\partial E(\mathbf{w})}{\partial a_{i-m,j-n}^{\ell+1}} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^\ell} = \\ &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i-m,j-n}^{\ell+1} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^\ell} \quad (2)\end{aligned}$$

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

Gradient of E(w) wrt activations

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

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

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

Gradient of $E(\mathbf{w})$ wrt activations

$$\begin{aligned}\frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^\ell} &= \frac{\partial}{\partial a_{i,j}^\ell} \left(\sum_{m'} \sum_{n'} w_{m',n'}^{\ell+1} f(a_{i-m+m',j-n+n'}^\ell) \right) \\ &= w_{m,n}^{\ell+1} \frac{\partial f(a_{i,j}^\ell)}{\partial a_{i,j}^\ell} \triangleq w_{m,n}^{\ell+1} f'(a_{i,j}^\ell)\end{aligned}$$

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

Gradient of $E(\mathbf{w})$ wrt activations

Eq. (2):

$$\begin{aligned}\delta_{i,j}^\ell &= \frac{\partial E(\mathbf{w})}{\partial a_{i,j}^\ell} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i-m,j-n}^{\ell+1} \frac{\partial a_{i-m,j-n}^{\ell+1}}{\partial a_{i,j}^\ell} = \\ &= \sum_m \sum_n \delta_{i-m,j-n}^{\ell+1} w_{m,n}^{\ell+1} f'(a_{i,j}^\ell) = \\ &= f'(a_{i,j}^\ell) \sum_m \sum_n \delta_{m,n}^{\ell+1} w_{i-m,j-n}^{\ell+1} = \\ &= f'(a_{i,j}^\ell) [\delta_{i,j}^{\ell+1} \otimes \text{rot}_{180^\circ}(\mathbf{w}^{\ell+1})]\end{aligned}$$

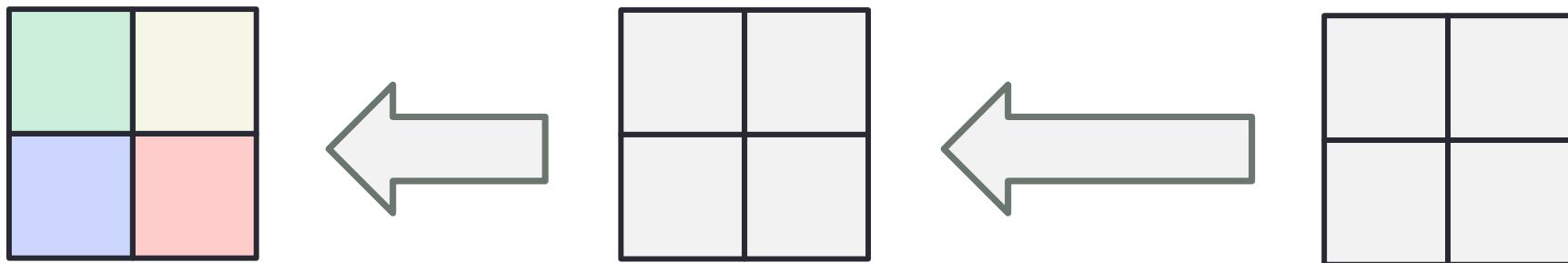


commutative property of convolution

Putting it all together

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

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



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

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:

$$\begin{aligned}\frac{\partial E(\mathbf{w})}{\partial b^\ell} &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial E(\mathbf{w})}{\partial a_{i,j}^\ell} \frac{\partial a_{i,j}^\ell}{\partial b^\ell} \\ &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^\ell \frac{\partial a_{i,j}^\ell}{\partial b^\ell} \\ \frac{\partial a_{i,j}^\ell}{\partial b^\ell} &=?\end{aligned}$$

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:

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

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

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

Accounting for the pooling layer (1/2)

- At the pooling 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
- To keep track of this winning unit
 - Its value is **saved** during the forward pass
 - and used for **gradient routing** during backpropagation

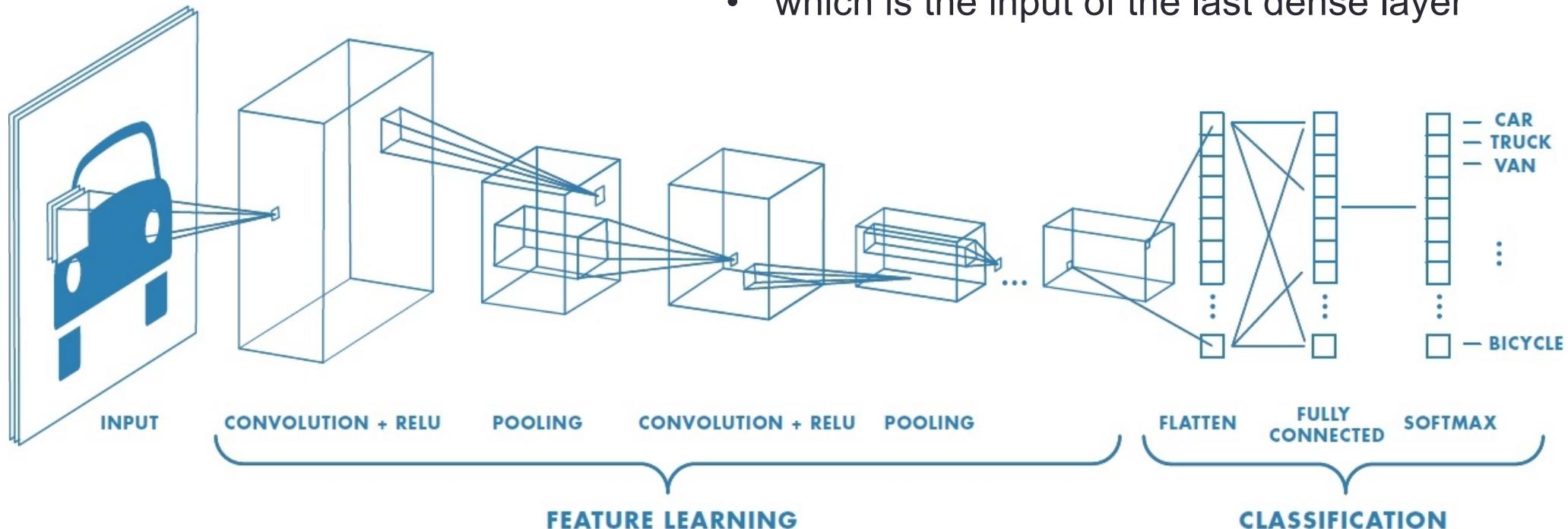
Accounting for the pooling layer (2/2)

- 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 $1/(N \times N)$ 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

The “Deep Learning Book”

[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>

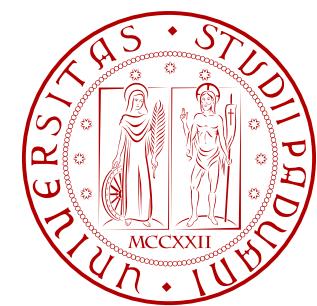
https://github.com/vdumoulin/conv_arithmetic

And plenty of blogs...

CONVOLUTIONAL NEURAL NETWORKS (CNN)

Michele Rossi
rossi@dei.unipd.it

Dept. of Information Engineering
University of Padova, IT



Derivative of activations wrt weights

- So far:

$$\frac{\partial E(\mathbf{w})}{\partial w_{m,n}^{\ell}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^{\ell} y_{i+m,j+n}^{\ell-1}$$

- Note:

- Instead of using the cross-correlation
- This Eq. can also be implemented using convolution

- In this case we need to:

- 1) flip the delta matrix 180 degrees
- 2) take the convolution of flipped matrix and y

- Cross-correlation often preferred to minimize no. operations