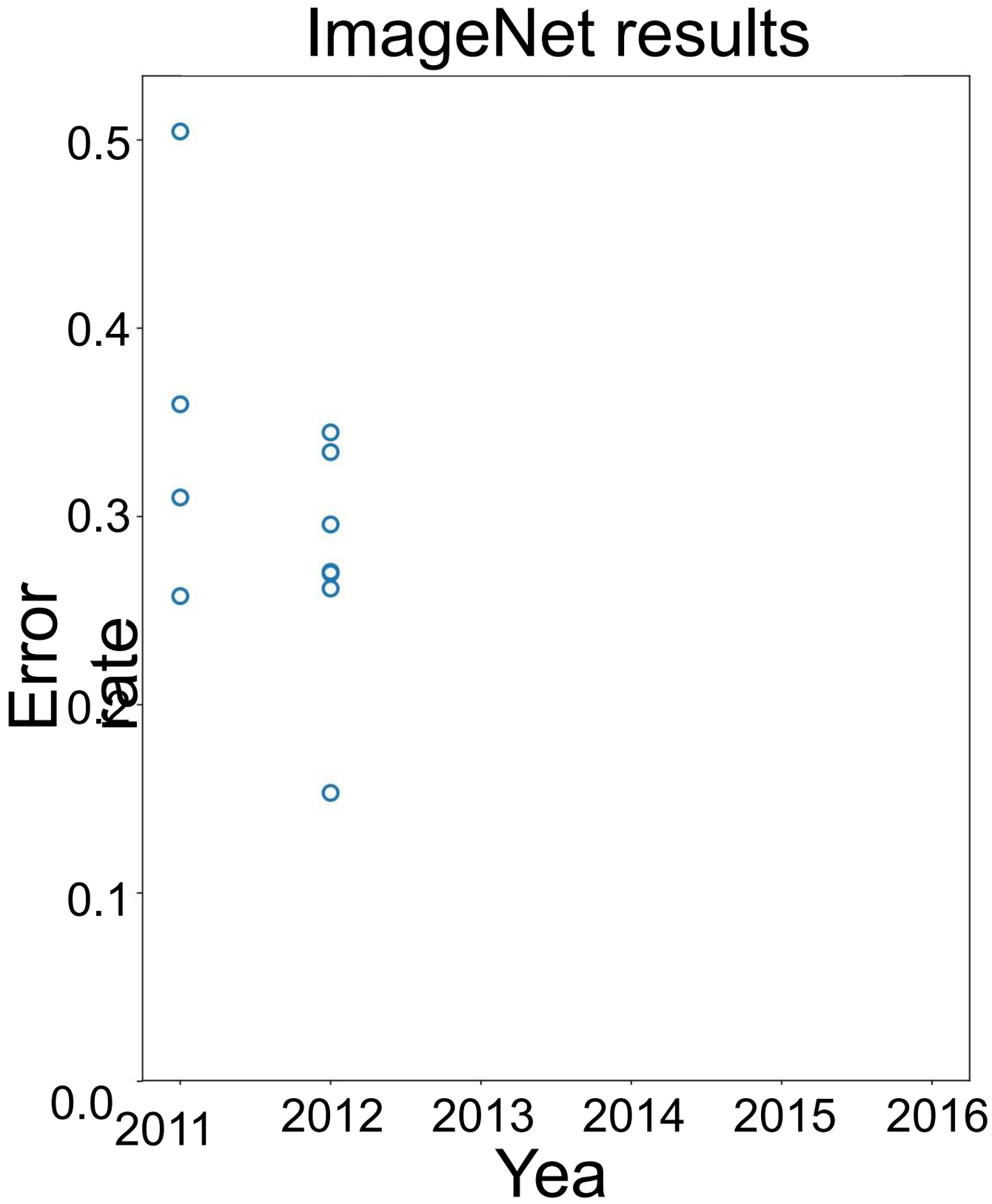
Convolutional Neural Networks (CNNs, ConvNets)

Somak Aditya
IIT KHARAGPUR

TAKEN FROM MIT EECS SLIDES.

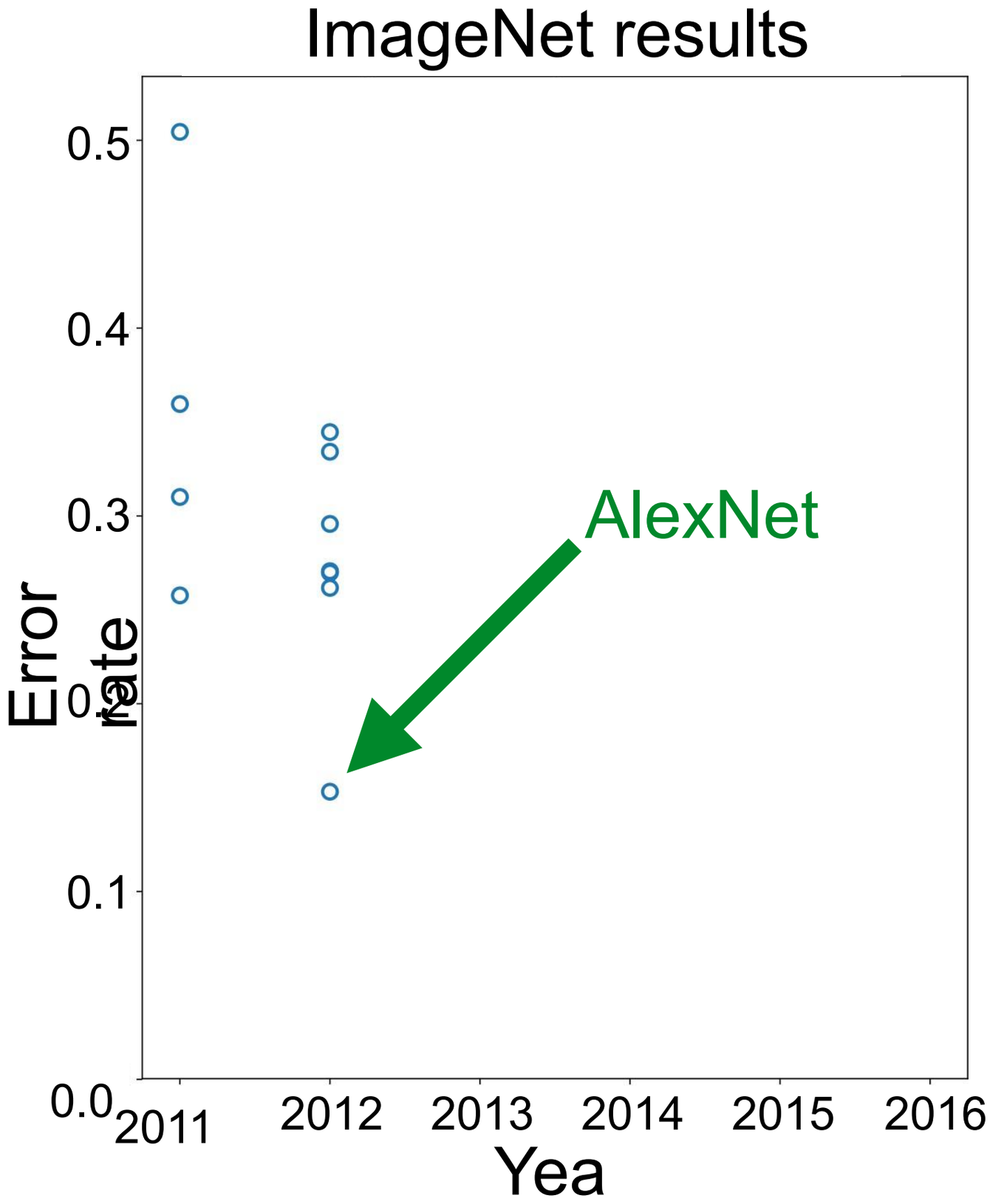
Impact of CNNs



- Since 2010: large-scale image classification challenge

[https://en.wikipedia.org/wiki/ImageNet#History_of_the_ImageNet_Challenge]

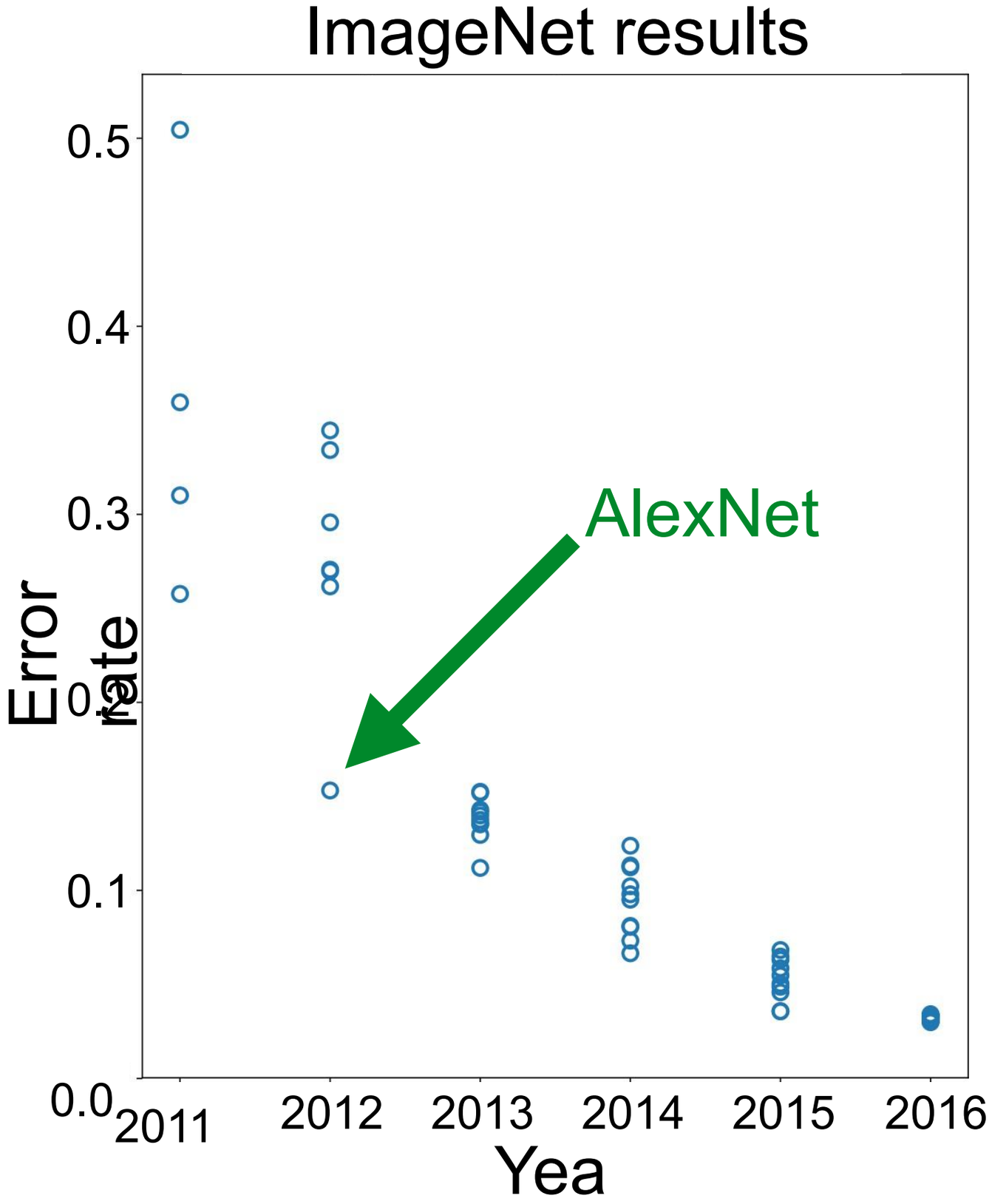
Impact of CNNs



- Since 2010: large-scale image classification challenge

[https://en.wikipedia.org/wiki/ImageNet#History_of_the_ImageNet_Challenge]

Impact of CNNs



- Since 2010: large-scale image classification challenge
- Recent AI boom
- 1960s, 1980s, today: neural networks
- Since 1980s: CNNs

[https://en.wikipedia.org/wiki/ImageNet#History_of_the_ImageNet_Challenge]

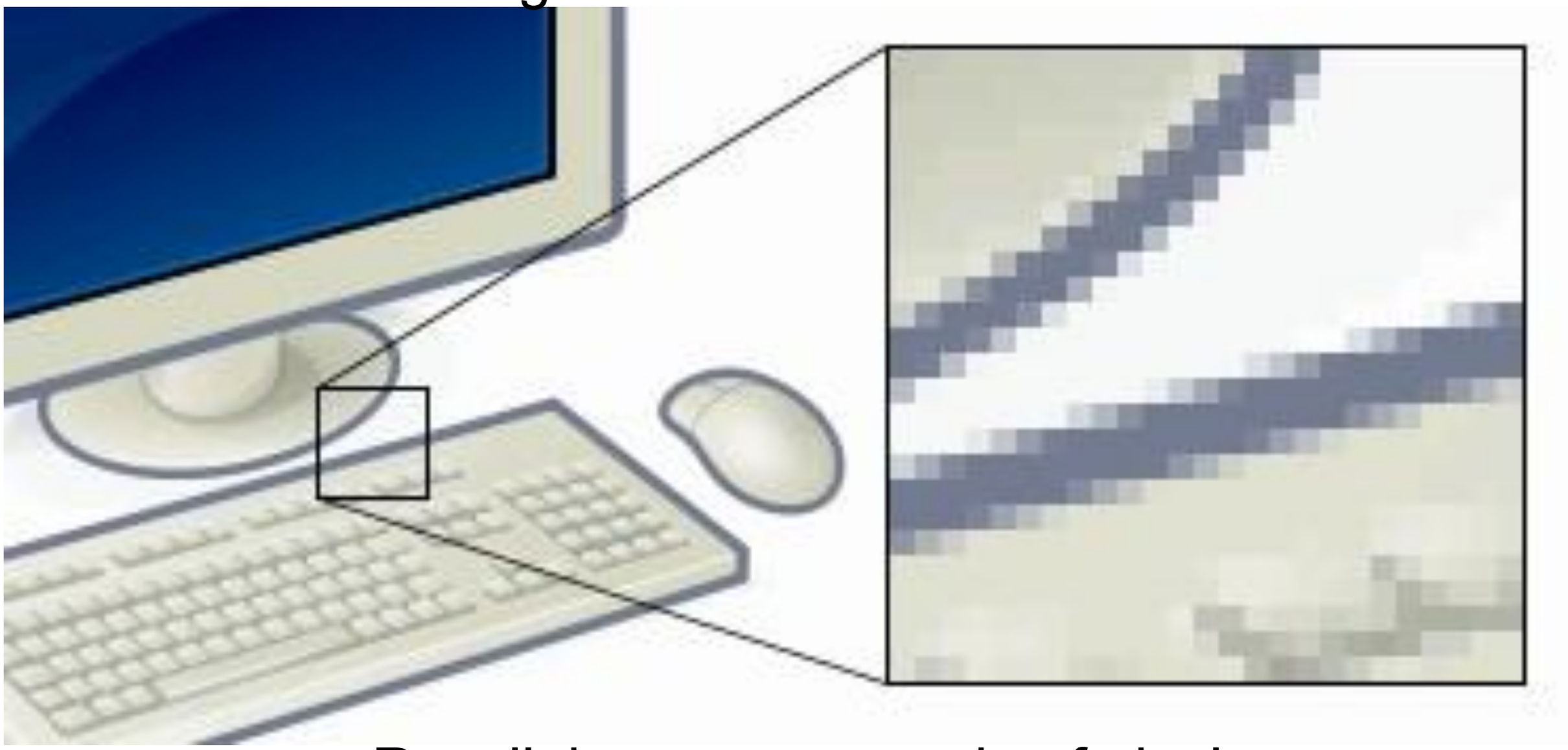
Images

- Potential uses of image classification: Detect tumor (type) from medical scans, image search online, autonomous driving

[https://en.wikipedia.org/wiki/Pixel#/media/File:Pixel-example.png]

Images

- Potential uses of image classification: Detect tumor (type) from medical scans, image search online, autonomous driving



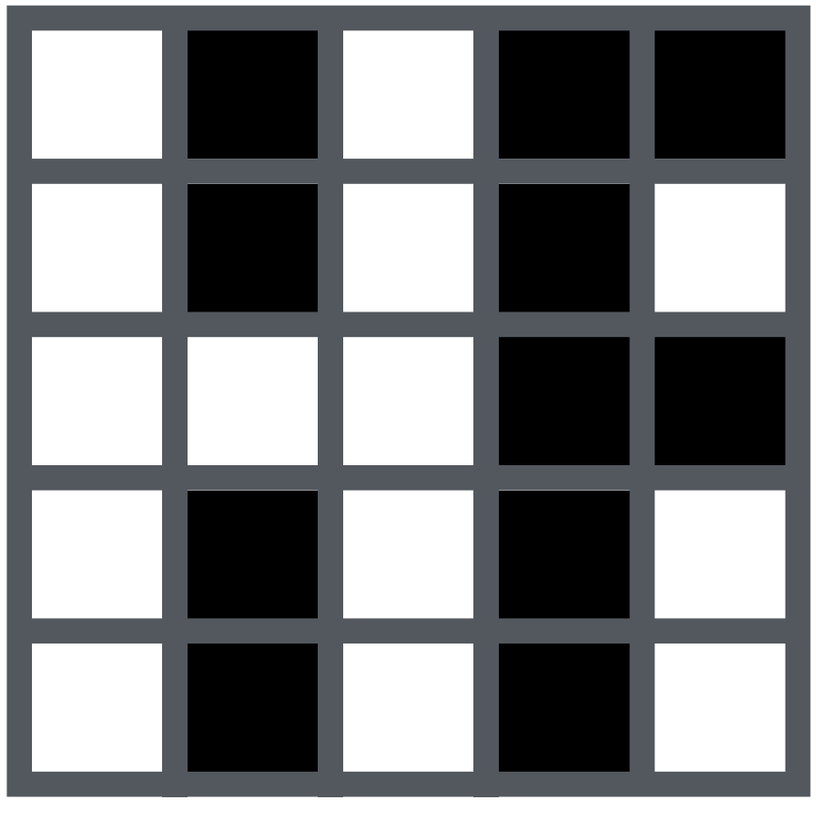
- Recall: images are made of pixels

[https://en.wikipedia.org/wiki/Pixel#/media/File:Pixel-example.png]

Images

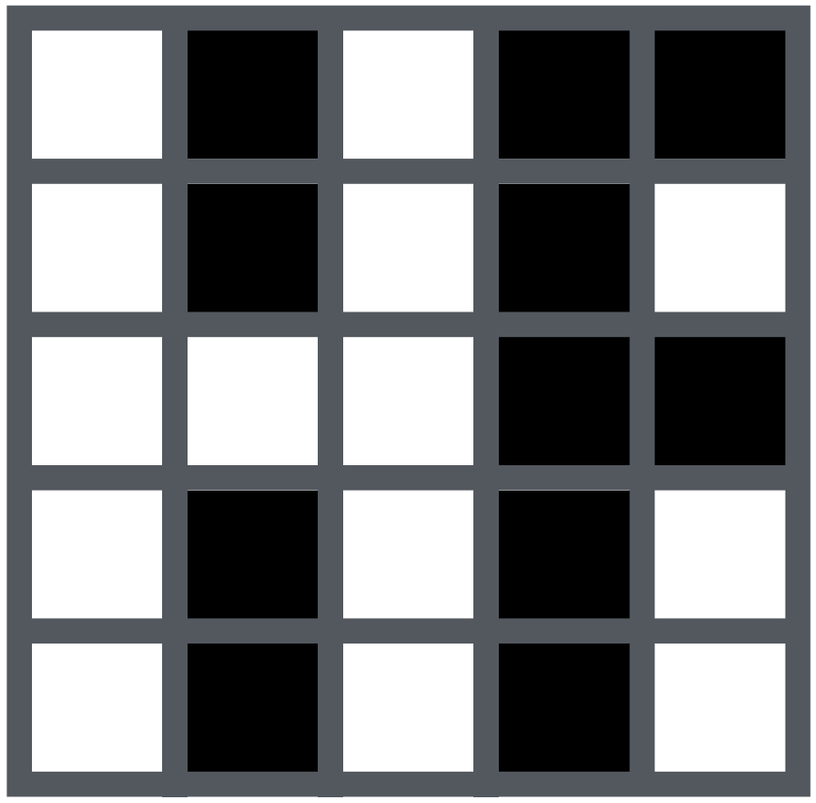
- We'll focus on grayscale images

Images



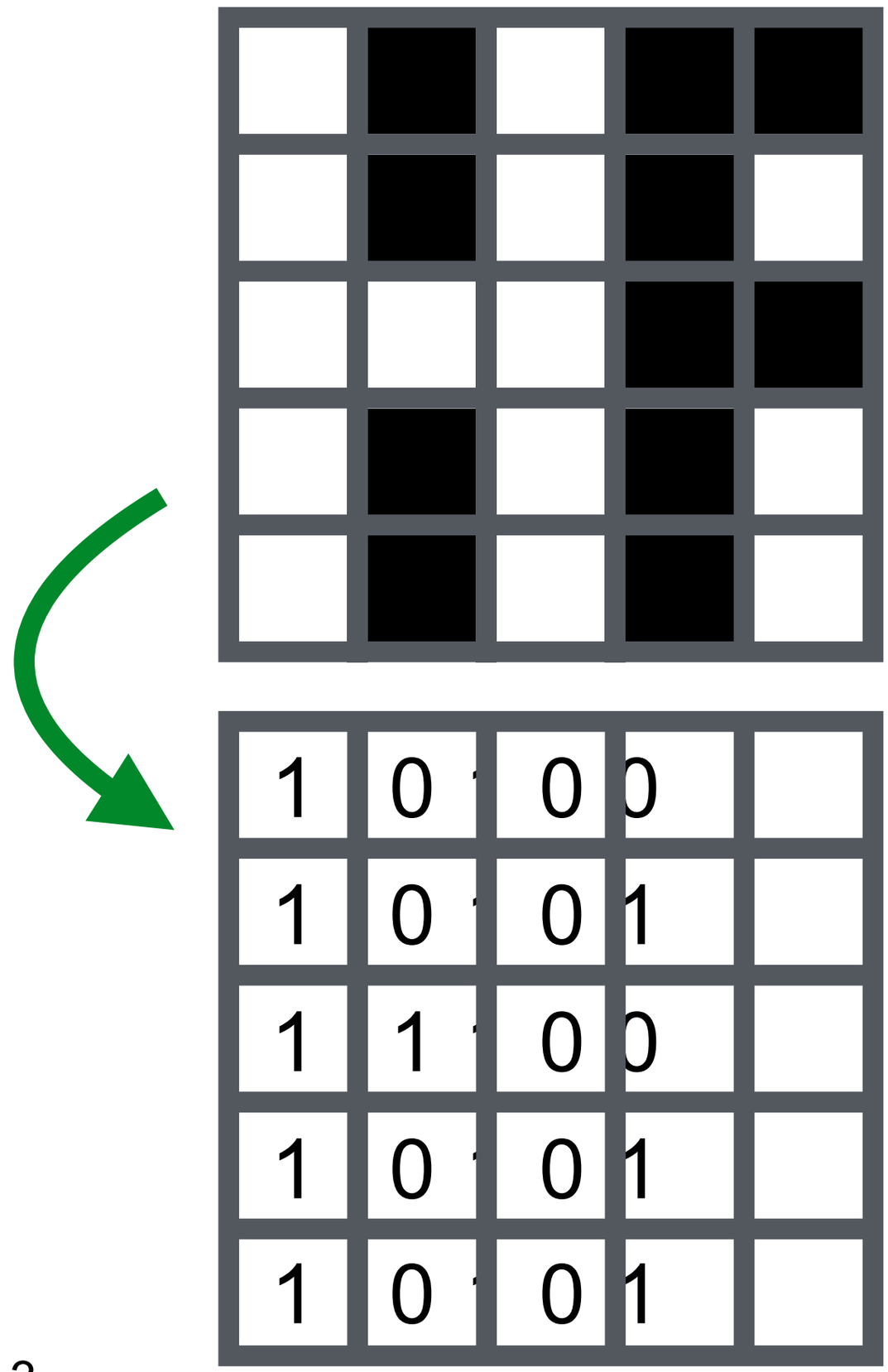
- We'll focus on grayscale images

Images



- We’ll focus on grayscale images
  - Each pixel takes a value between 0 and *P*
  - Here, 0: black, 1: white
  - Larger *P* in Lab Week 08

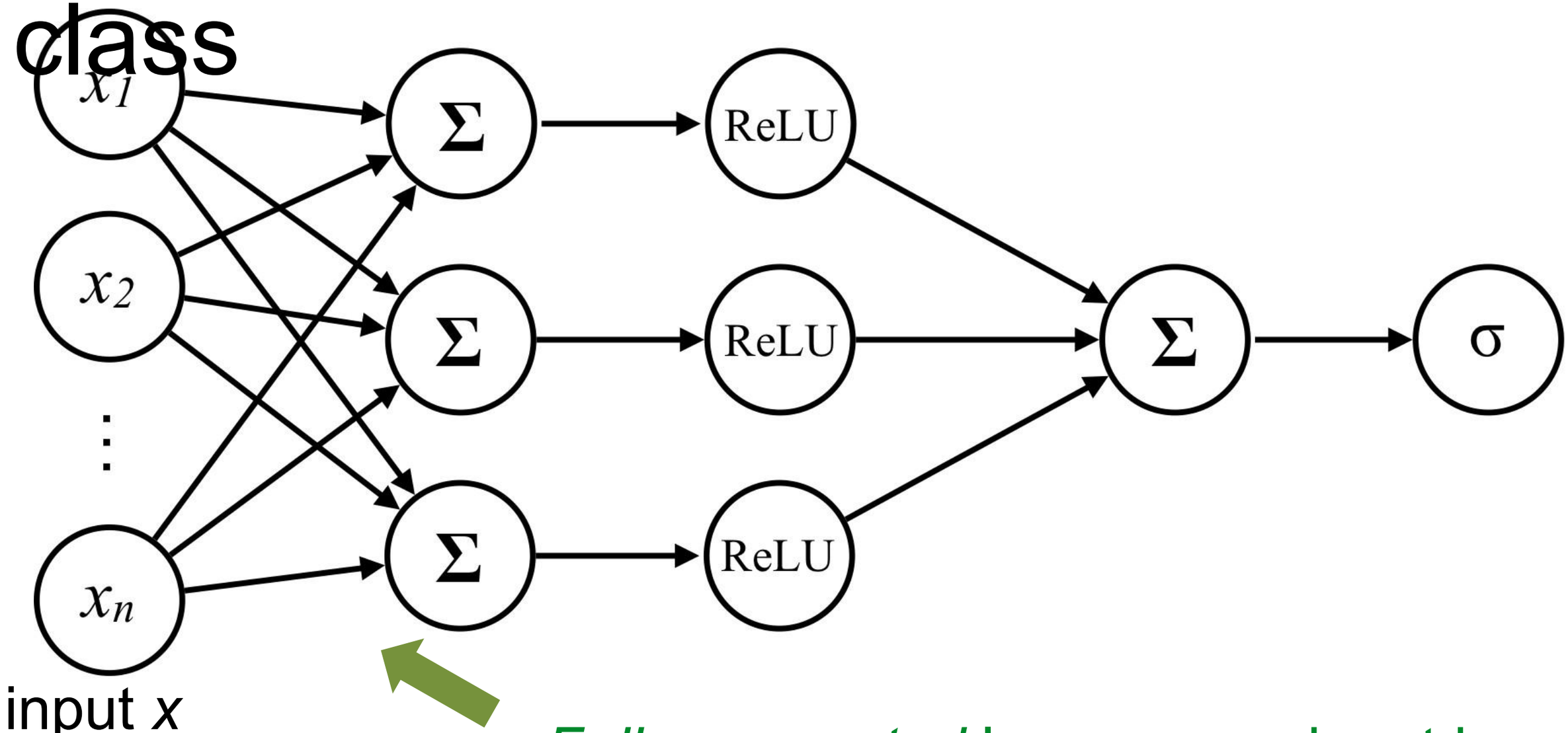
Images



- We'll focus on grayscale images
  - Each pixel takes a value between 0 and *P*
  - Here, 0: black, 1: white
  - Larger *P* in Lab Week 08

- How do we use an image as an input for a neural net?

Previous neural nets in this class



input *x*
(*n* x 1)

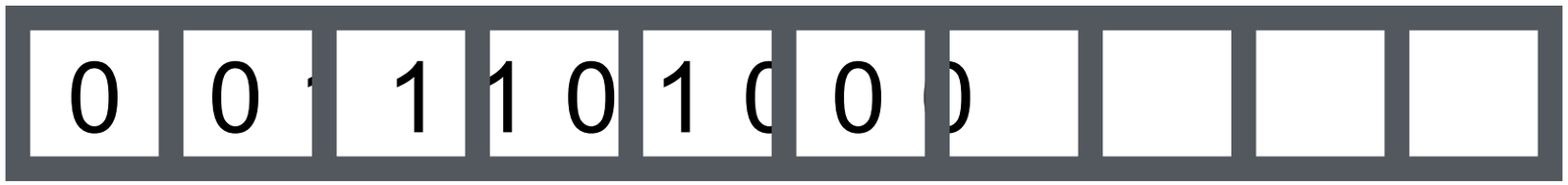
Fully connected layer: every input is connected to every output by a weight

But we know more about images:

- Spatial locality
  Translation
  invariance

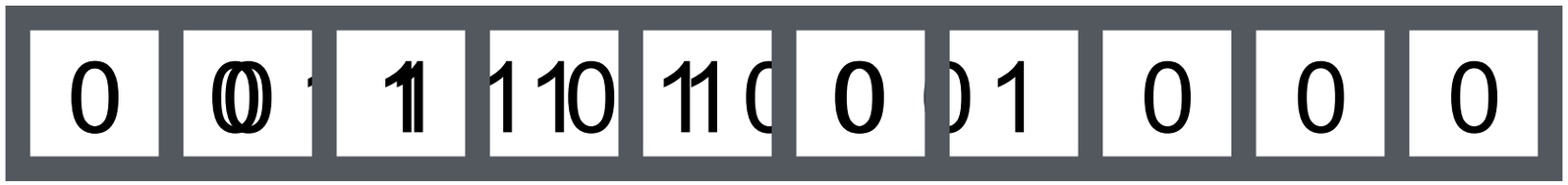
Convolutional Layer: 1D example

A 1D image:

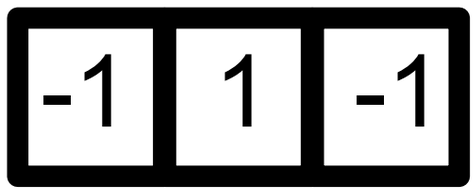


Convolutional Layer: 1D example

A 1D image:

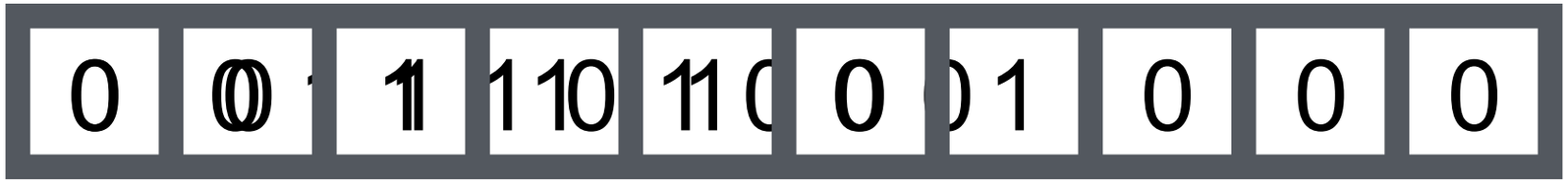


A filter:

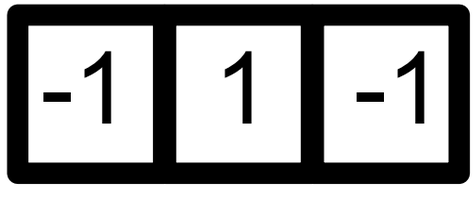


Convolutional Layer: 1D example

A 1D image:



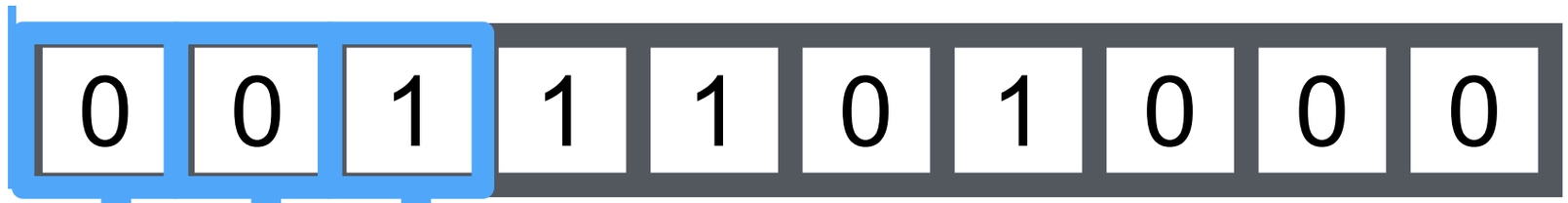
A filter:



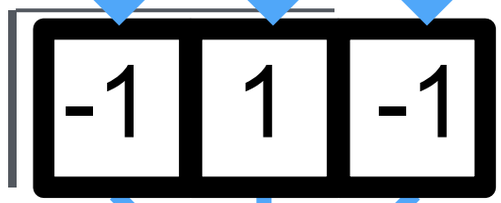
After convolution*:

Convolutional Layer: 1D example

A 1D image:



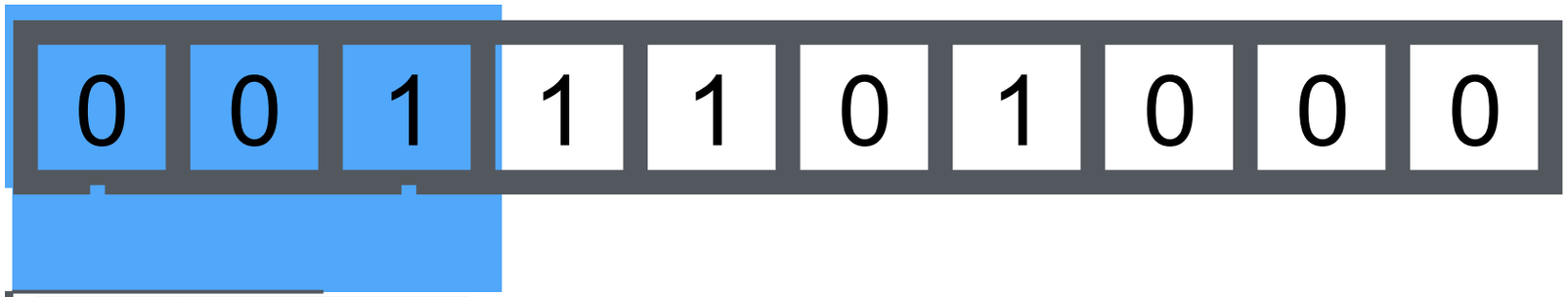
A filter:



After convolution*:

Convolutional Layer: 1D example

A 1D image:

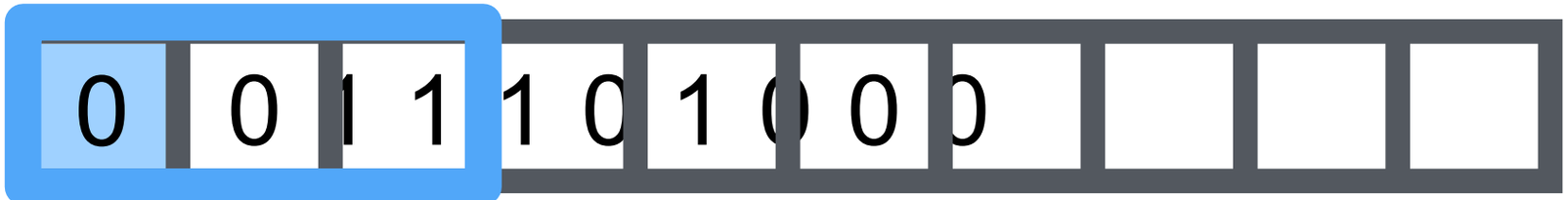


A filter:

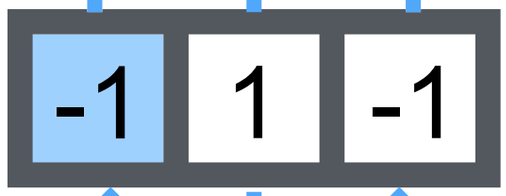
After convolution*:

Convolutional Layer: 1D example

A 1D image:

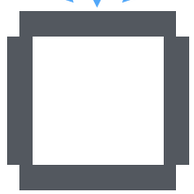


A filter:



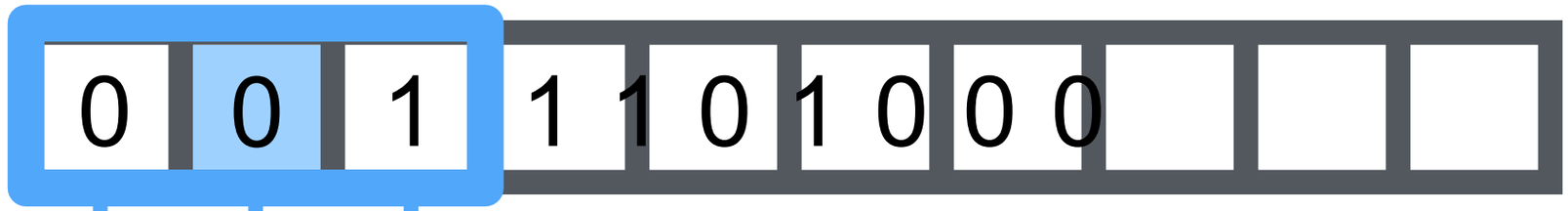
0 * -1

After convolution*:

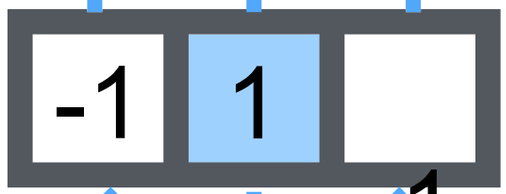


Convolutional Layer: 1D example

A 1D image:



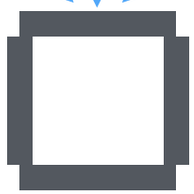
A filter:



0 * -1

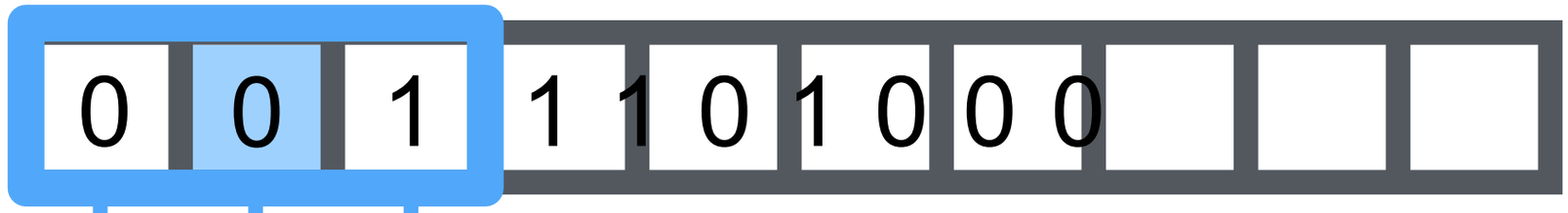
-1

After convolution*:

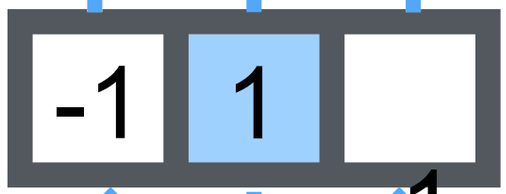


Convolutional Layer: 1D example

A 1D image:

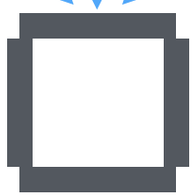


A filter:



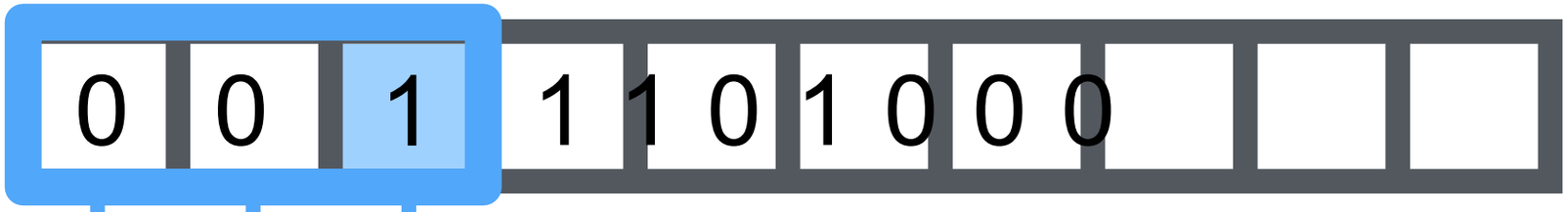
0 * -1 + 0 * 1

After convolution*:

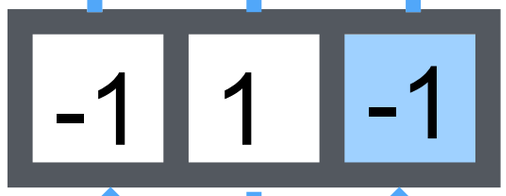


Convolutional Layer: 1D example

A 1D image:

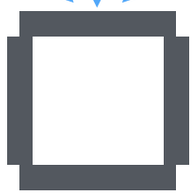


A filter:



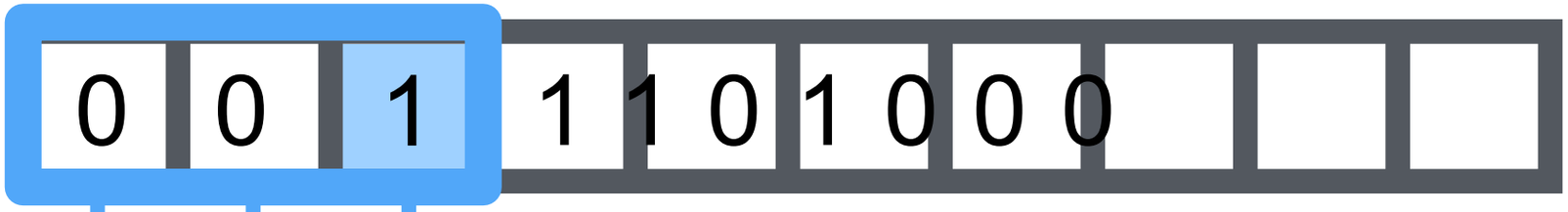
0 * -1 + 0 * 1

After convolution*:

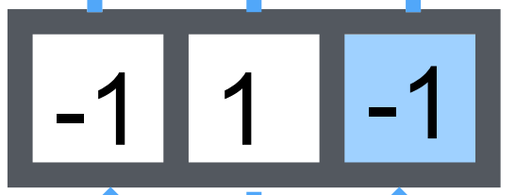


Convolutional Layer: 1D example

A 1D image:

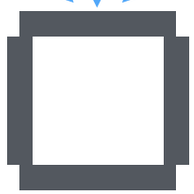


A filter:



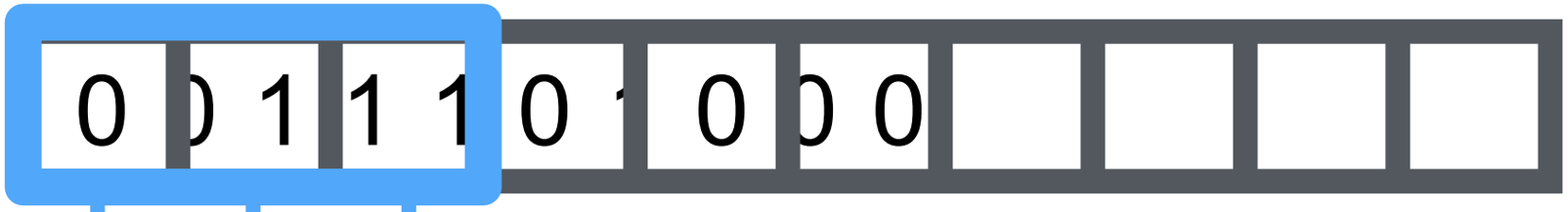
0 * -1 + 0 * 1 + 1 * -1

After convolution*:

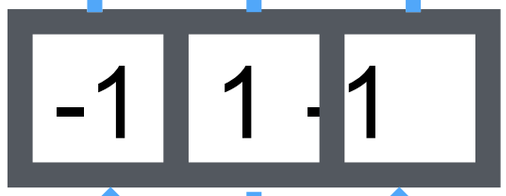


Convolutional Layer: 1D example

A 1D image:

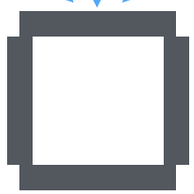


A filter:



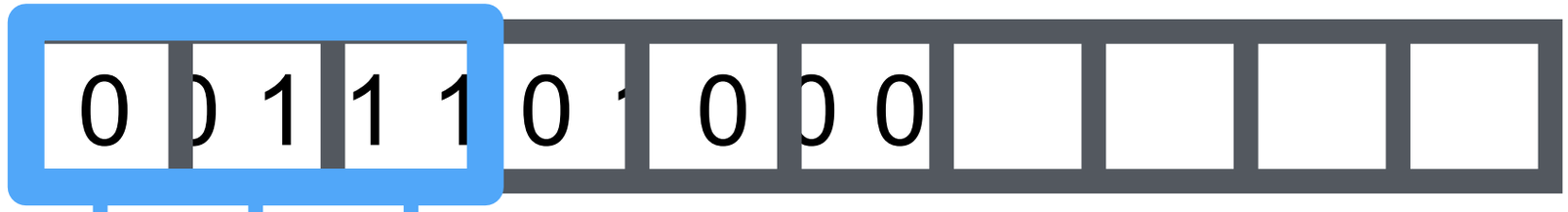
0 * -1 + 0 * 1 + 1 * -1

After convolution*:

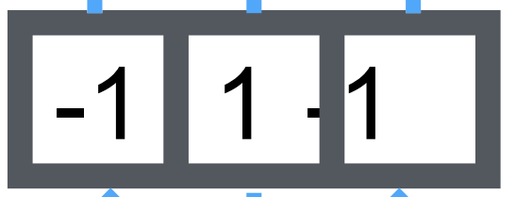


Convolutional Layer: 1D example

A 1D image:

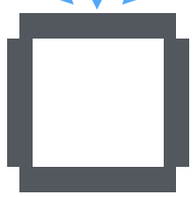


A filter:



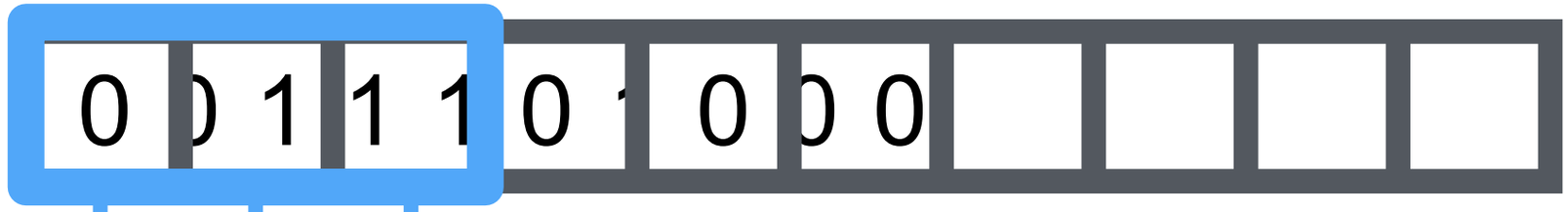
0 * -1 + 0 * 1 + 1 * -1 = -1

After convolution*:

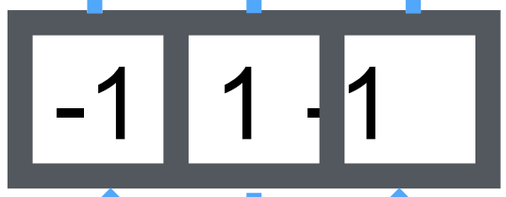


Convolutional Layer: 1D example

A 1D image:



A filter:

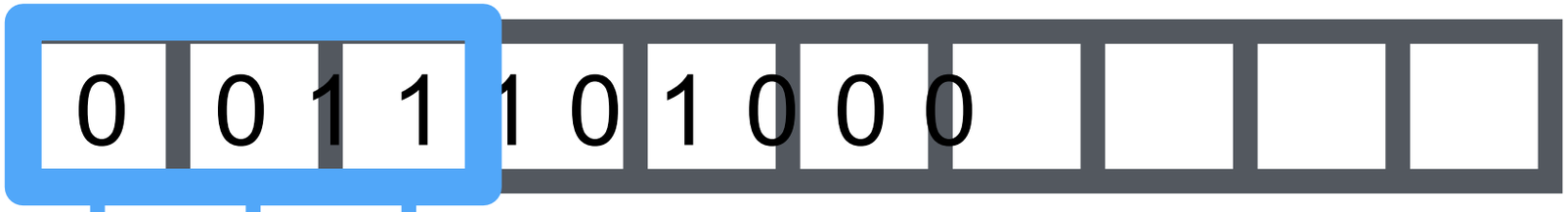


0 * -1 + 0 * 1 + 1 * -1 = -1

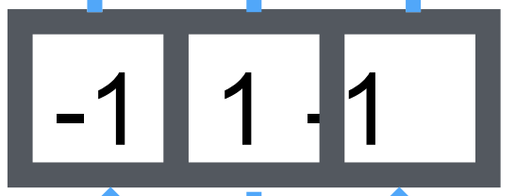
After convolution*:

Convolutional Layer: 1D example

A 1D image:



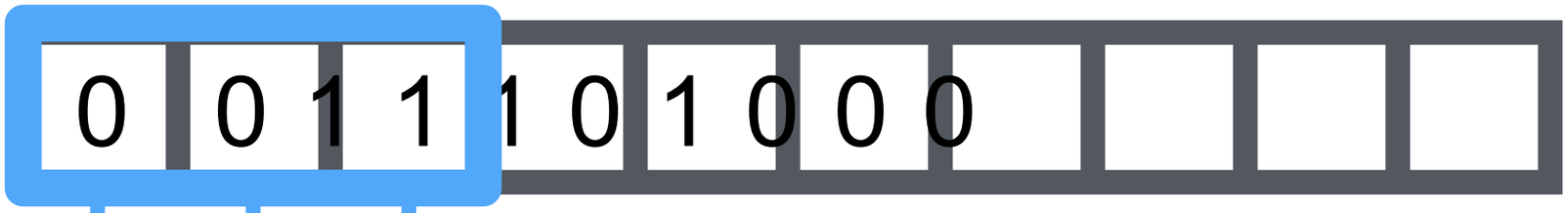
A filter:



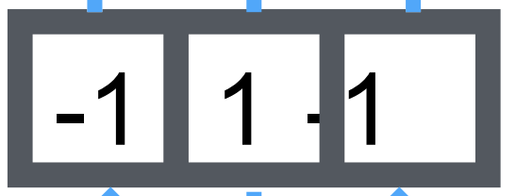
After convolution*:

Convolutional Layer: 1D example

A 1D image:



A filter:



After convolution*:

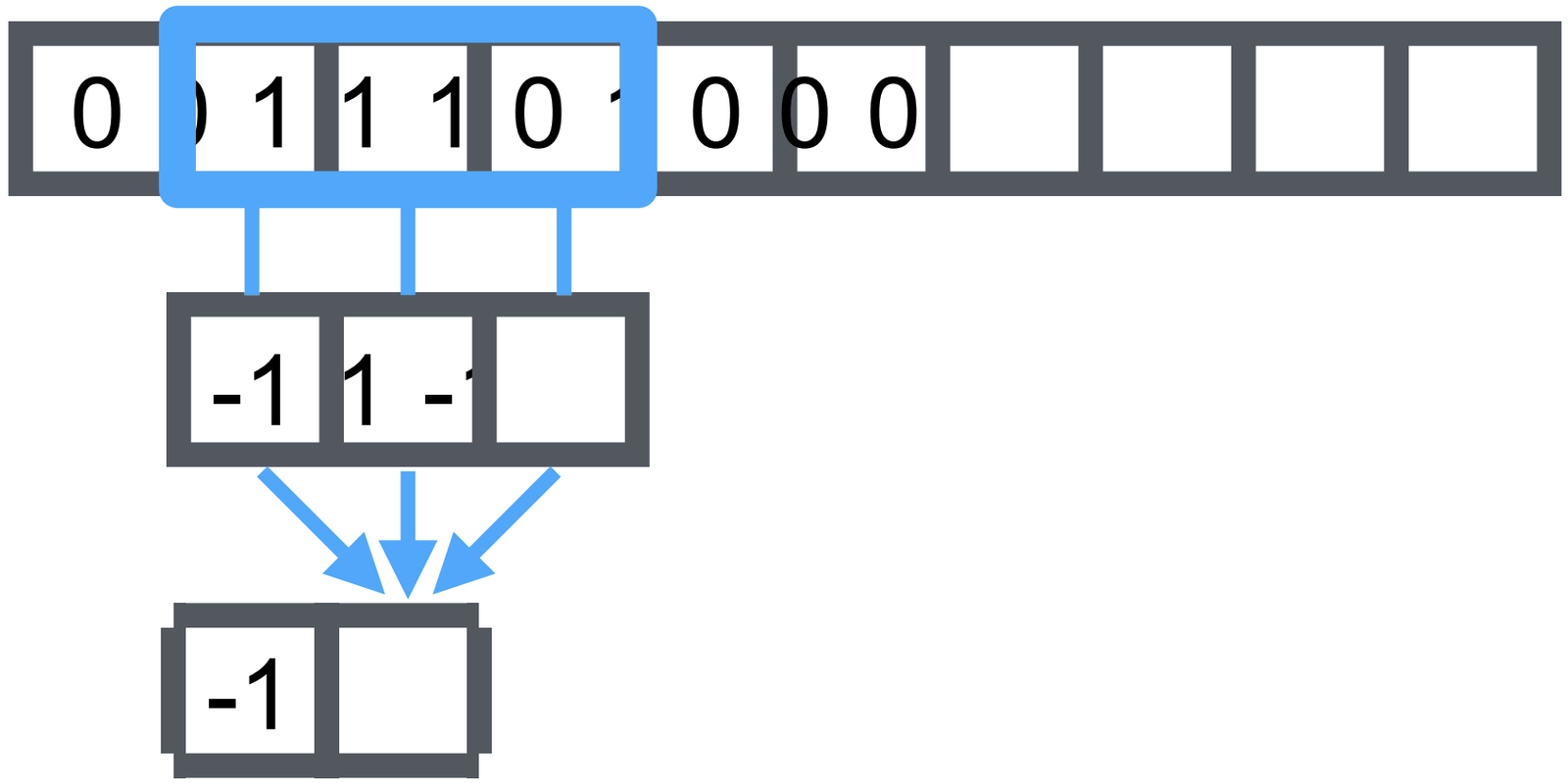
*correlation

Convolutional Layer: 1D example

A 1D image:

A filter:

After convolution*:



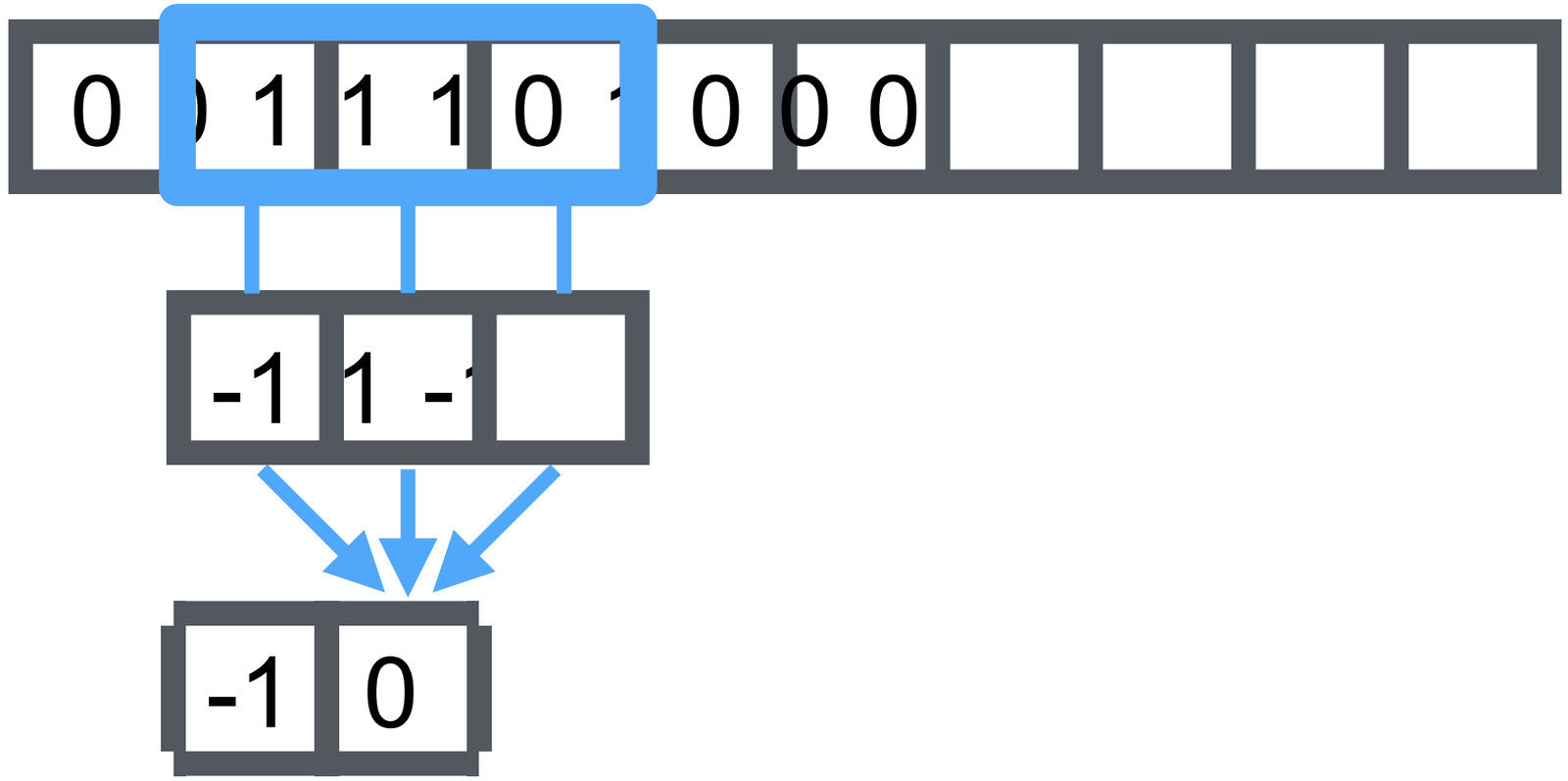
*correlation

Convolutional Layer: 1D example

A 1D image:

A filter:

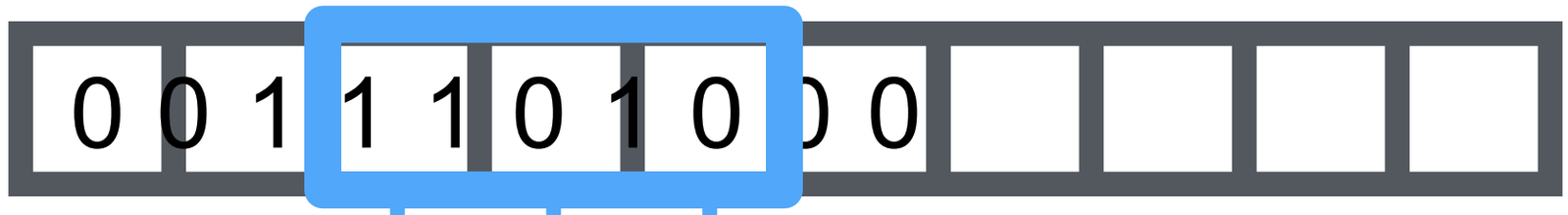
After convolution*:



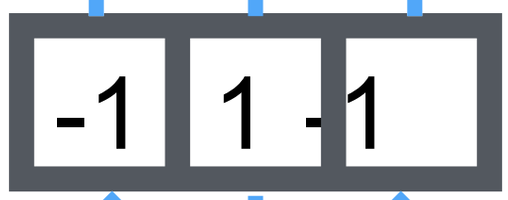
*correlation

Convolutional Layer: 1D example

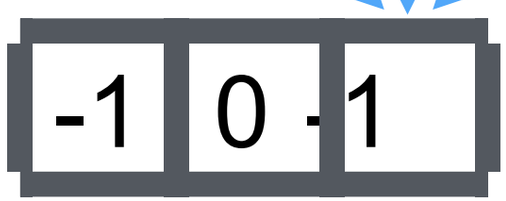
A 1D image:



A filter:



After convolution*:

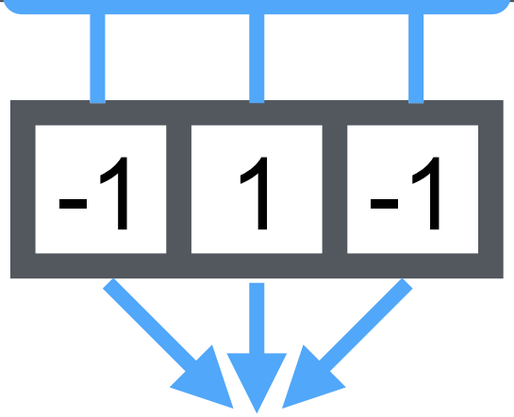


*correlation

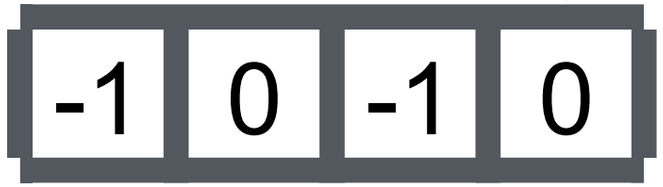
Convolutional Layer: 1D example

A 1D image:

A filter:



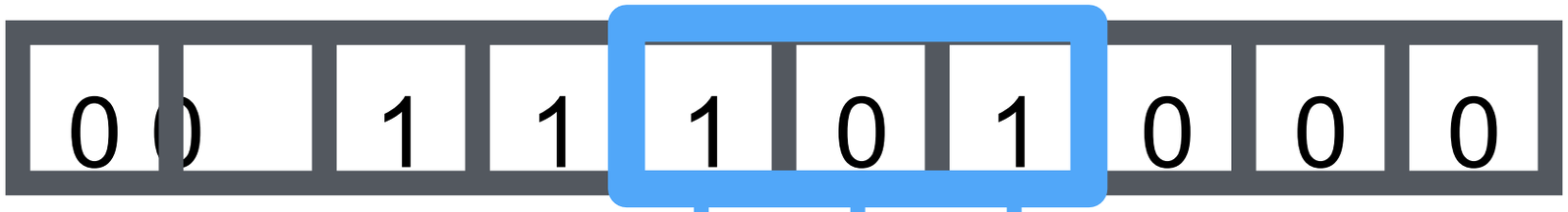
After convolution*:



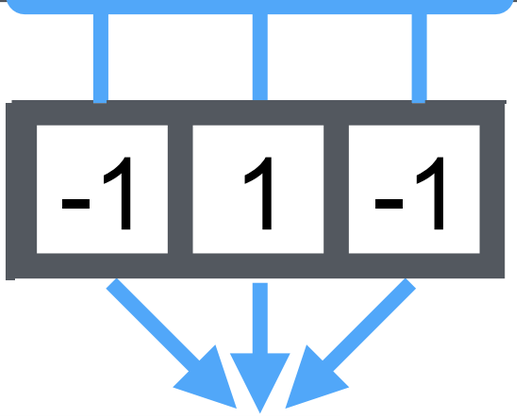
*correlation

Convolutional Layer: 1D example

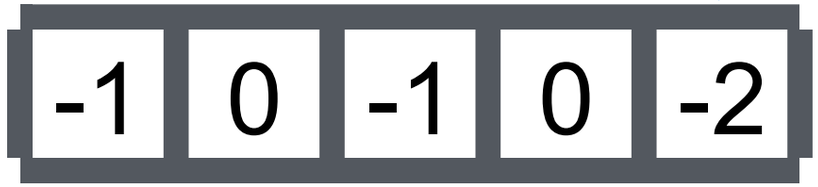
A 1D image:



A filter:



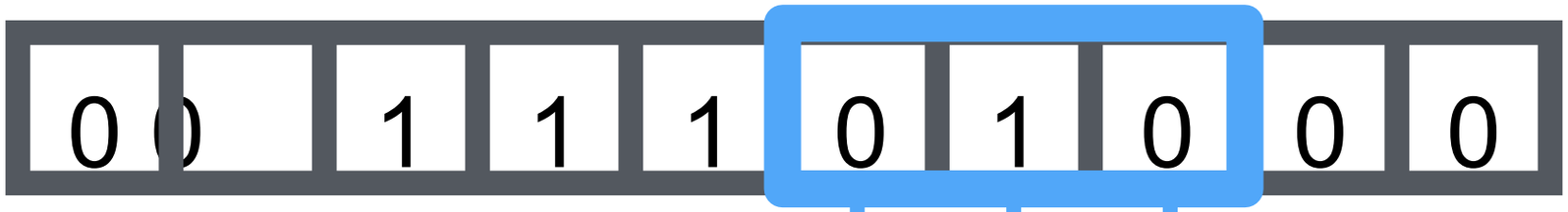
After convolution*:



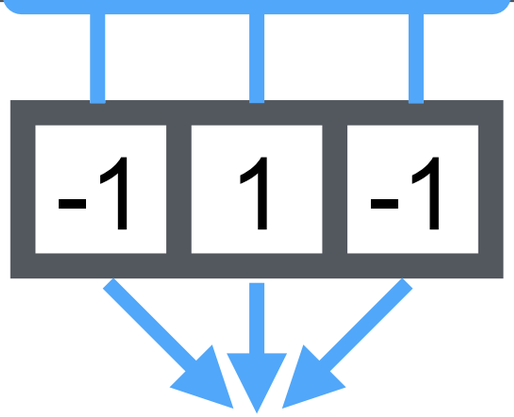
*correlation

Convolutional Layer: 1D example

A 1D image:



A filter:

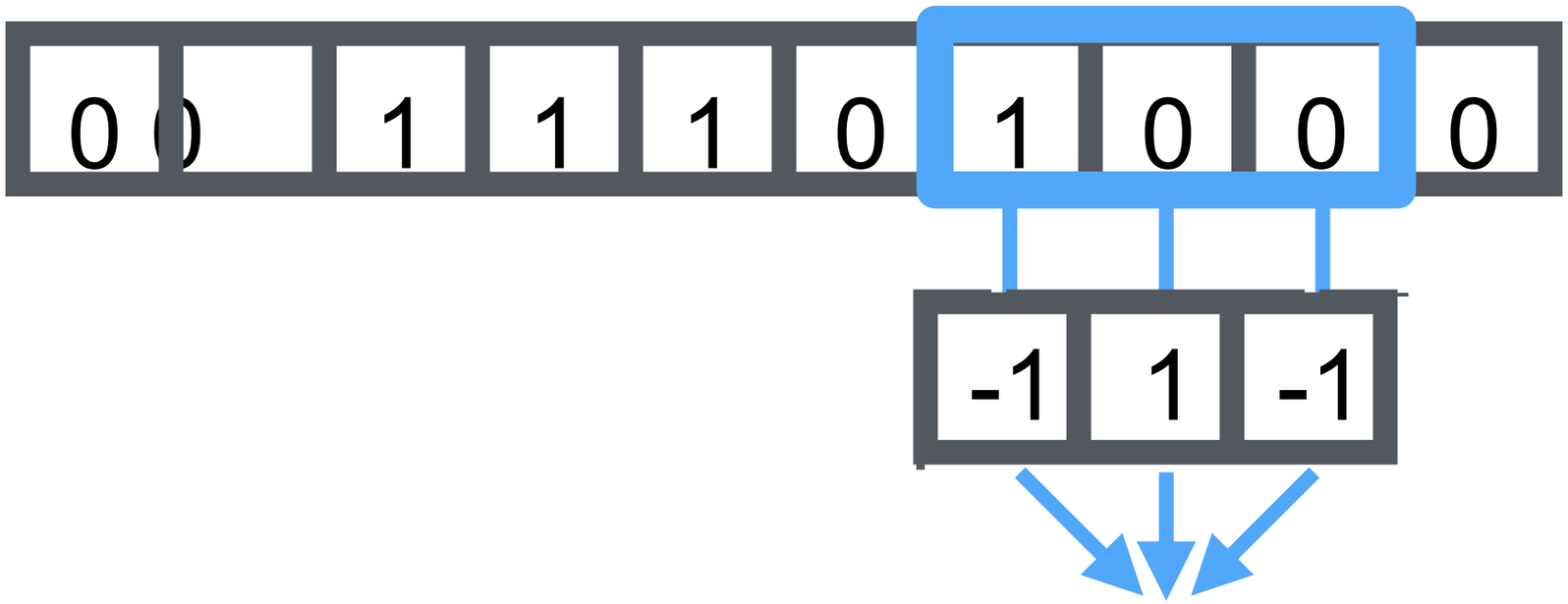


After convolution*:

*correlation

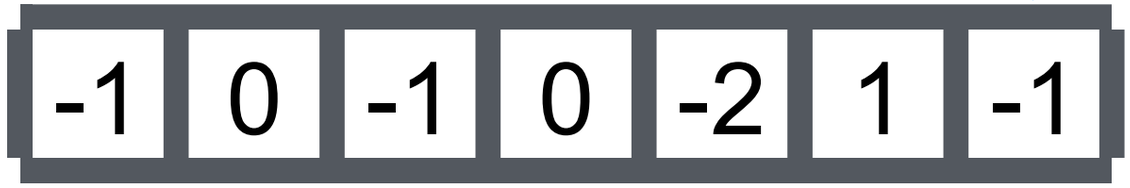
Convolutional Layer: 1D example

A 1D image:



A filter:

After convolution*:

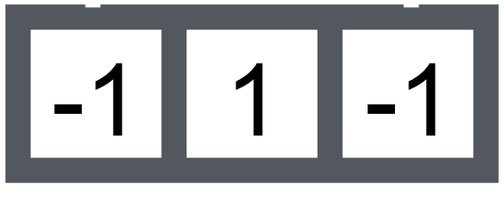


*correlation

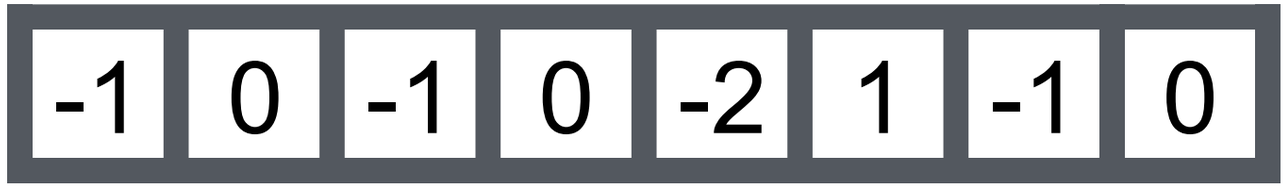
Convolutional Layer: 1D example

A 1D image:

A filter:



After convolution*:

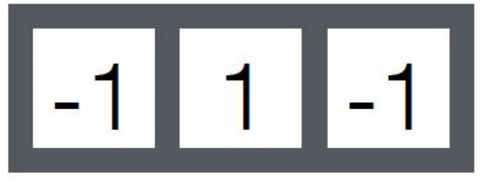


*correlation

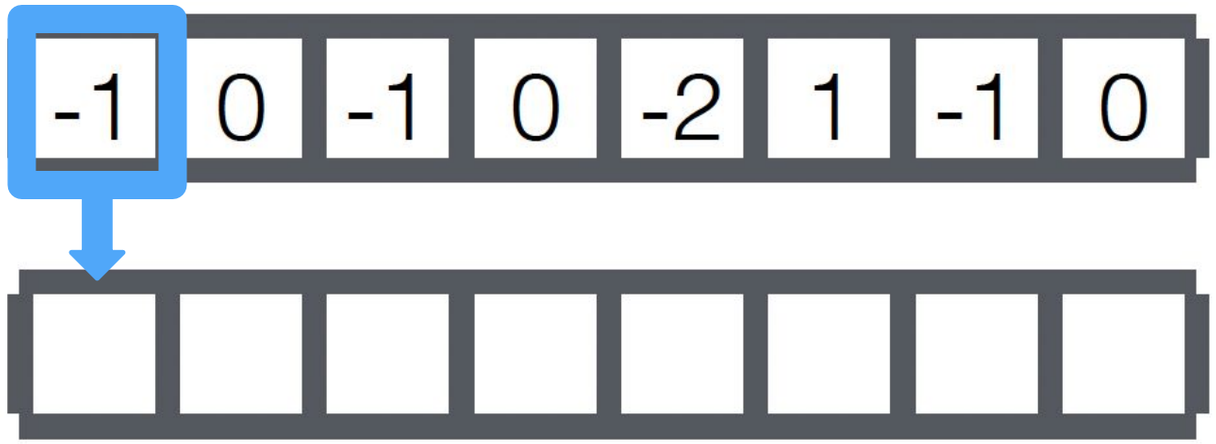
Convolutional Layer: 1D example

A 1D image:

A filter:



After convolution*:

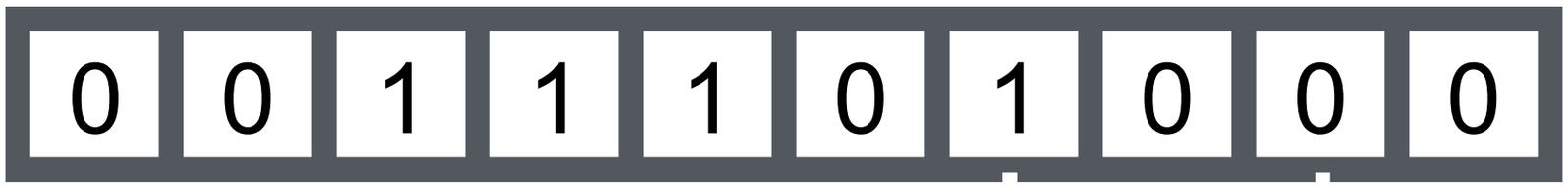


After ReLU:

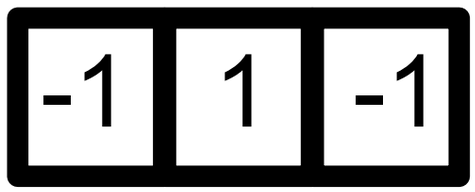
*correlation

Convolutional Layer: 1D example

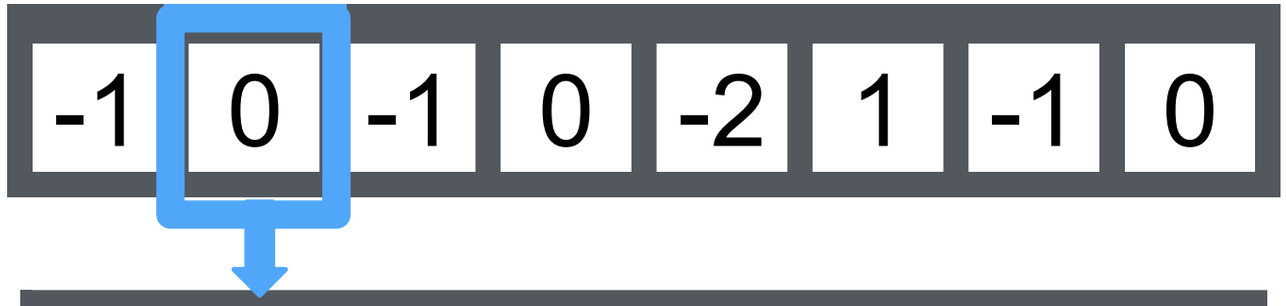
A 1D image:



A filter:



After convolution*:



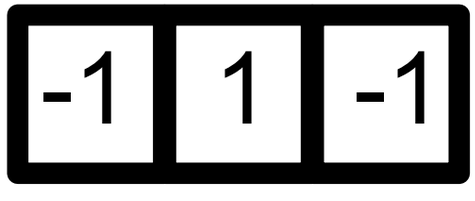
After ReLU:

*correlation

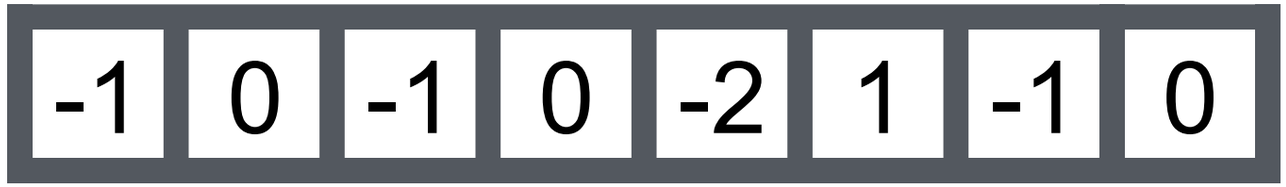
Convolutional Layer: 1D example

A 1D image:

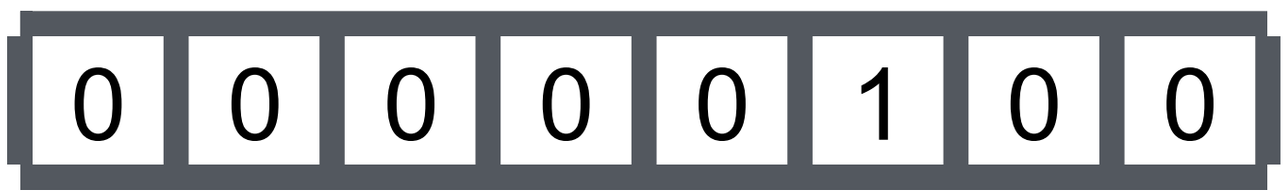
A filter:



After convolution*:



After ReLU:



What does the filter do?

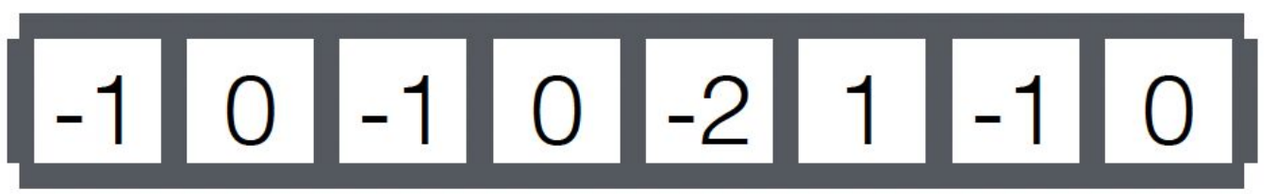
*correlation

Convolutional Layer: 1D example

A 1D image:

A filter:

After convolution*: *:

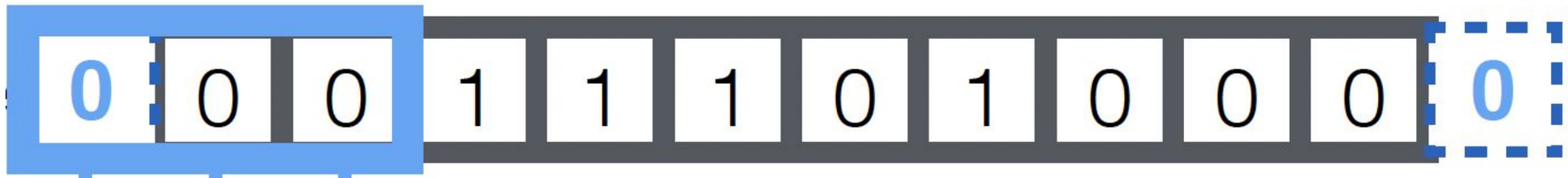


After ReLU:

*correlation

Convolutional Layer: 1D example

A 1D image:



A filter:

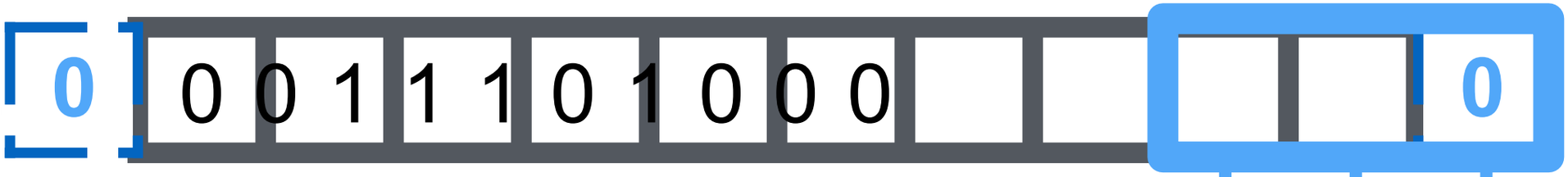
After convolution:

After ReLU:

*correlation

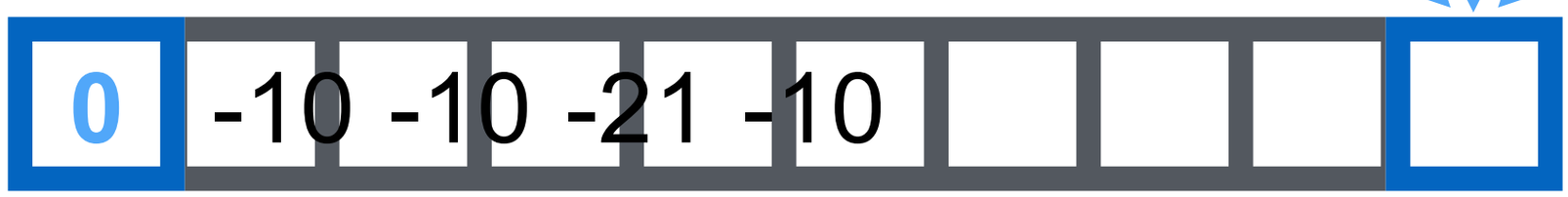
Convolutional Layer: 1D example

A 1D image:

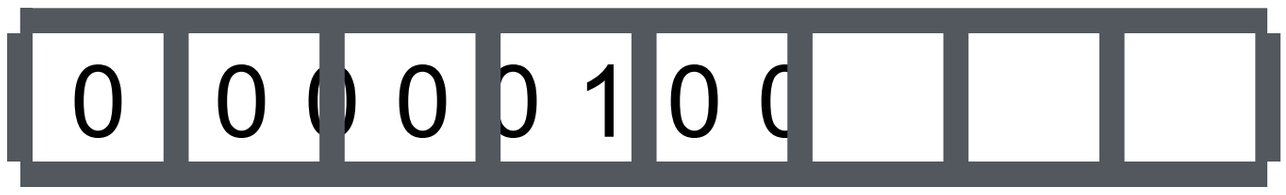


A filter:

After convolution*:



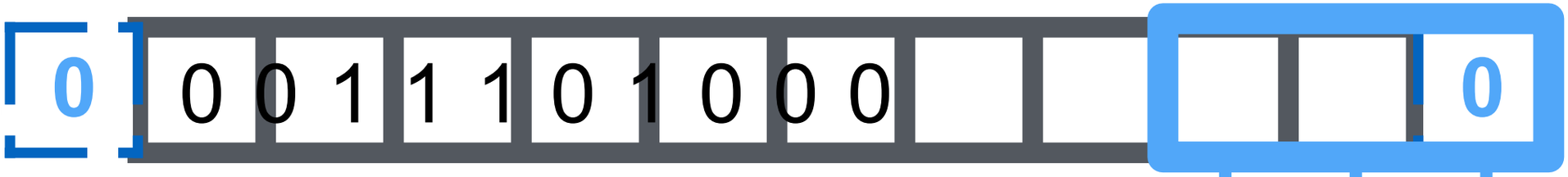
After ReLU:



*correlation

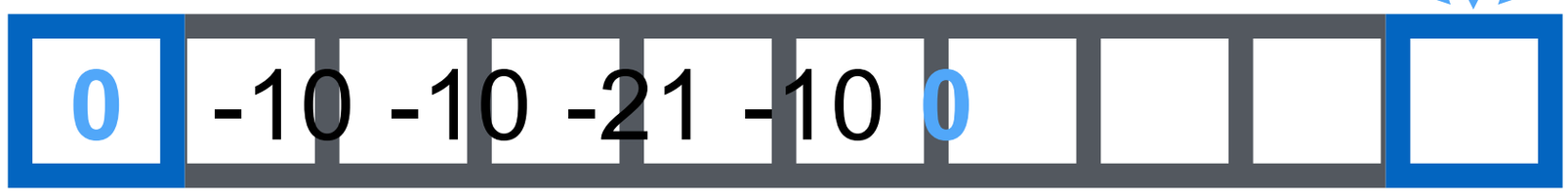
Convolutional Layer: 1D example

A 1D image:

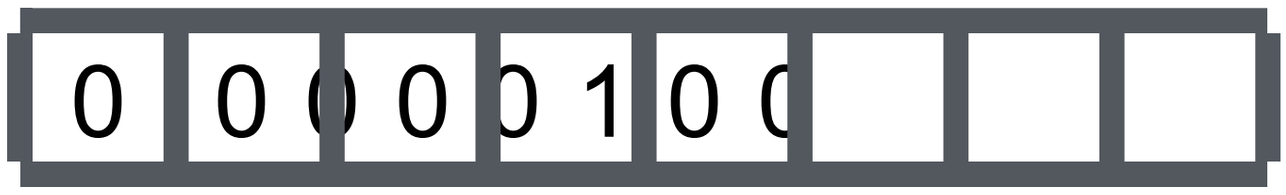


A filter:

After convolution*:



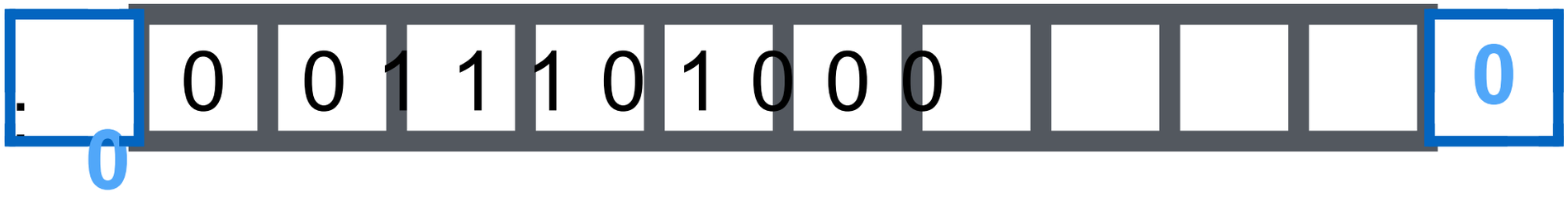
After ReLU:



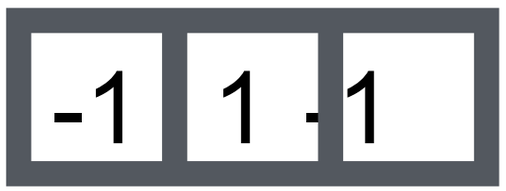
*correlation

Convolutional Layer: 1D example

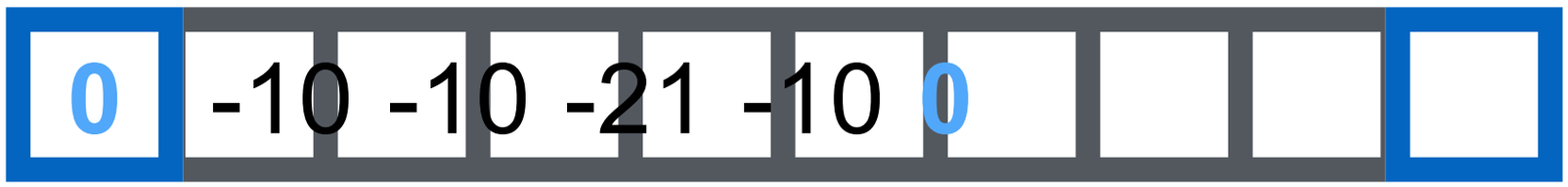
A 1D image



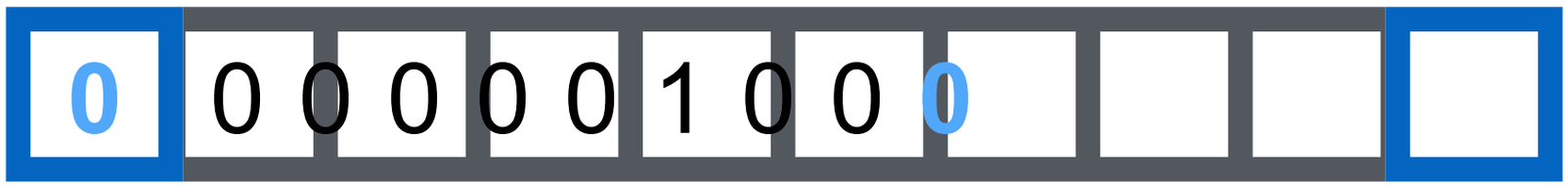
A filter:



After convolution*:



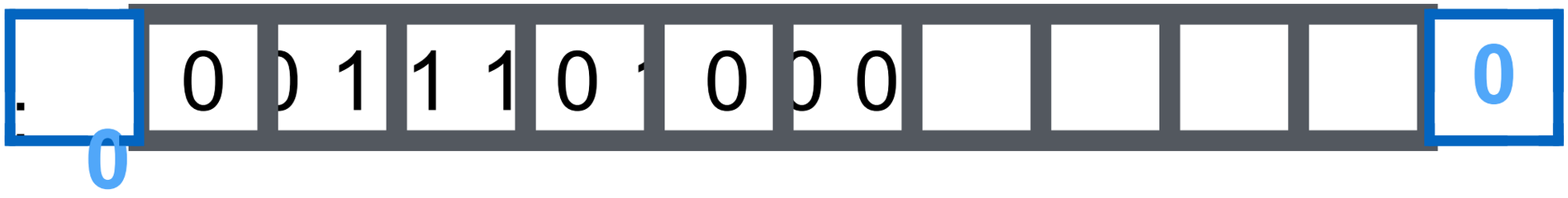
After ReLU:



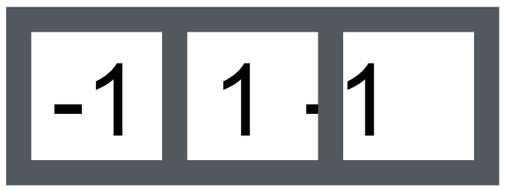
*correlation

Convolutional Layer: 1D example

A 1D image:



A filter:



with bias +1

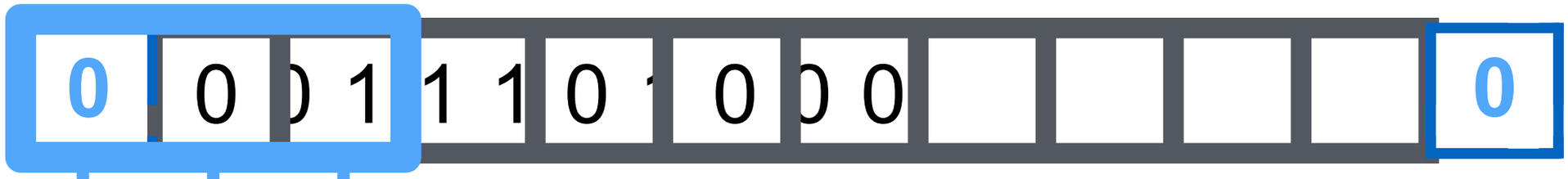
After convolution*:

After ReLU:

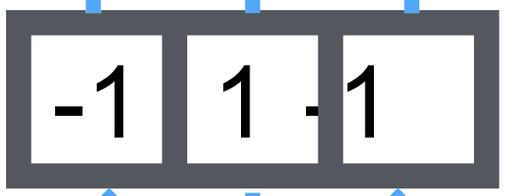
*correlation

Convolutional Layer: 1D example

A 1D image



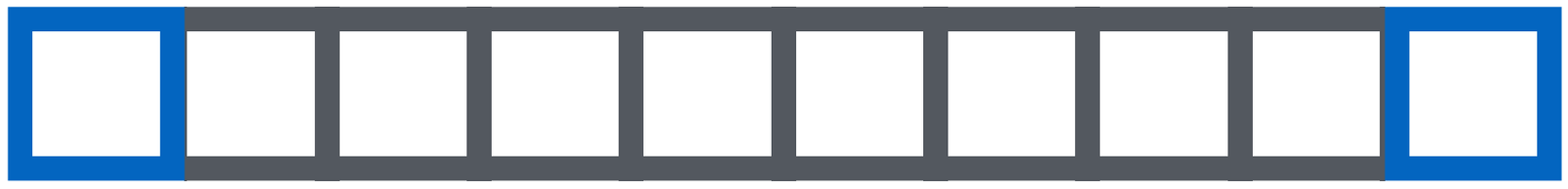
A filter:



with bias +1

After convolution*:

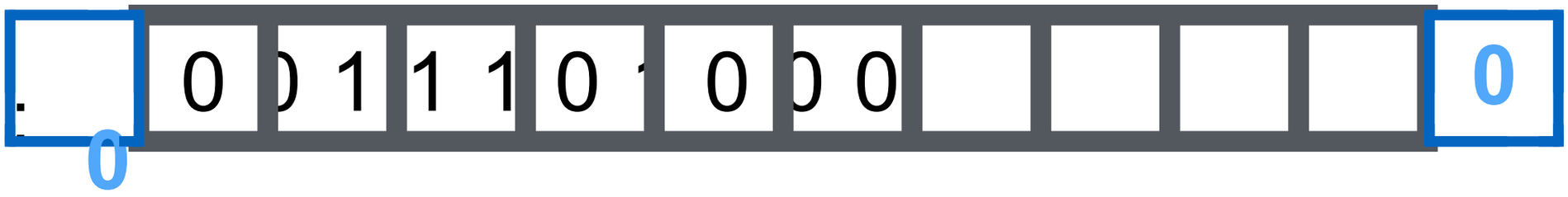
After ReLU:



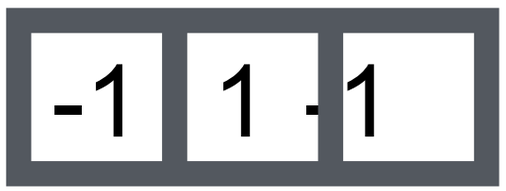
*correlation

Convolutional Layer: 1D example

A 1D image



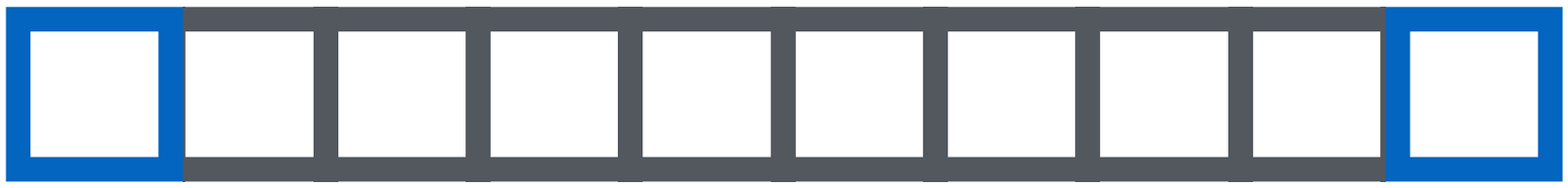
A filter:



with bias +1

After convolution*:

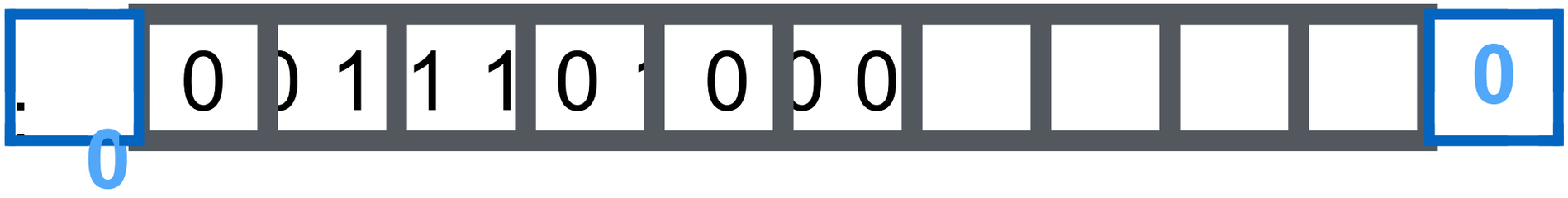
After ReLU:



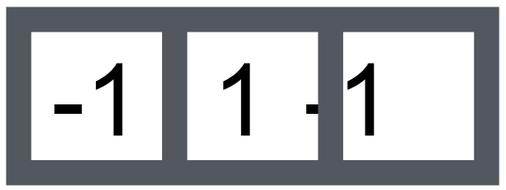
*correlation

Convolutional Layer: 1D example

A 1D image:



A filter:



with bias +1

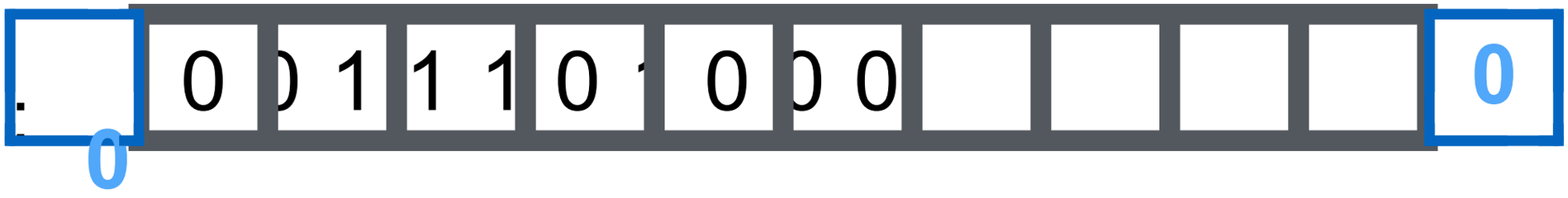
After convolution*:

After ReLU:

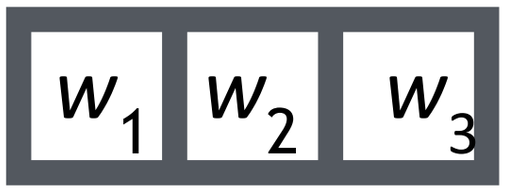
*correlation

Convolutional Layer: 1D example

A 1D image



A filter:



with bias b

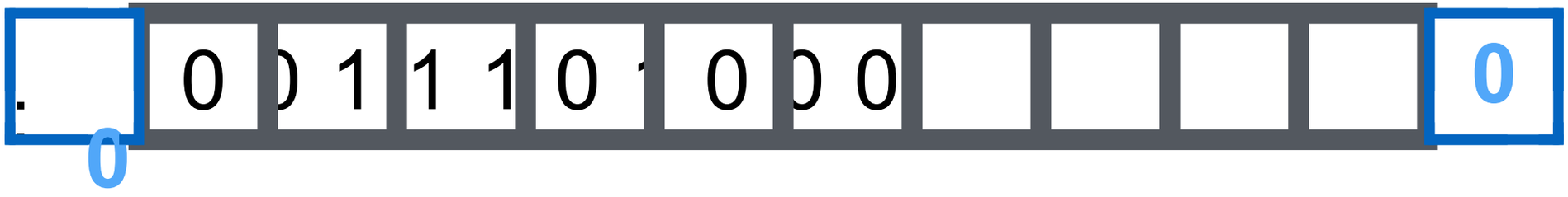
After convolution*:

After ReLU:

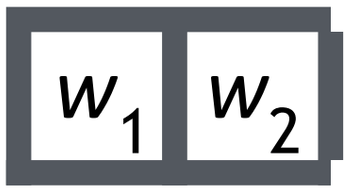
*correlation

Convolutional Layer: 1D example

A 1D image



A filter:



with bias b

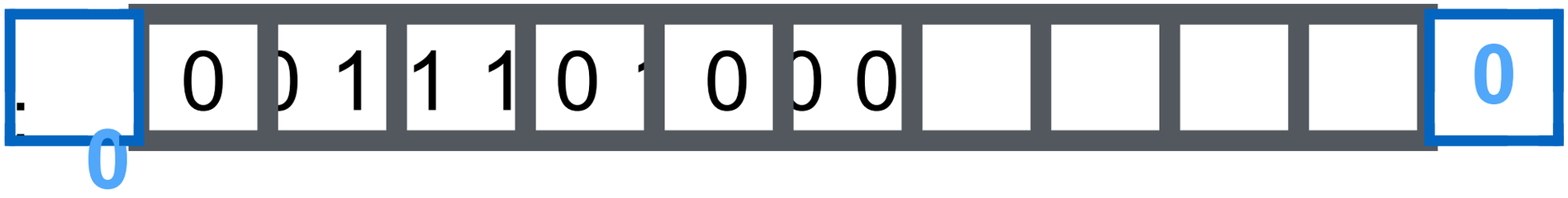
After convolution*:

After ReLU:

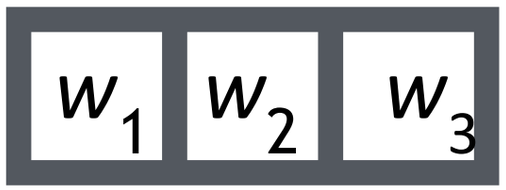
*correlation

Convolutional Layer: 1D example

A 1D image



A filter:



with bias b

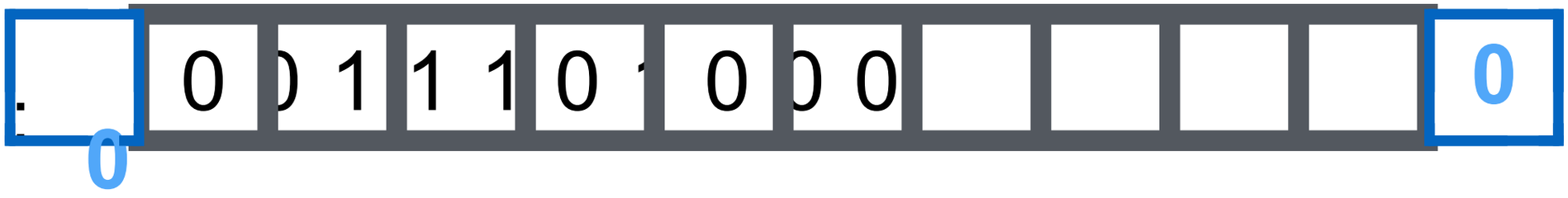
After convolution*:

After ReLU:

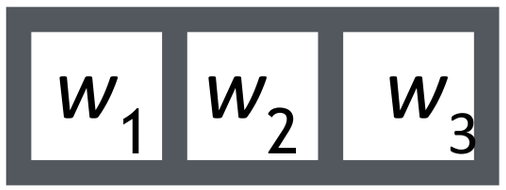
*correlation

Convolutional Layer: 1D example

A 1D image



A filter:



with bias b

After convolution*:

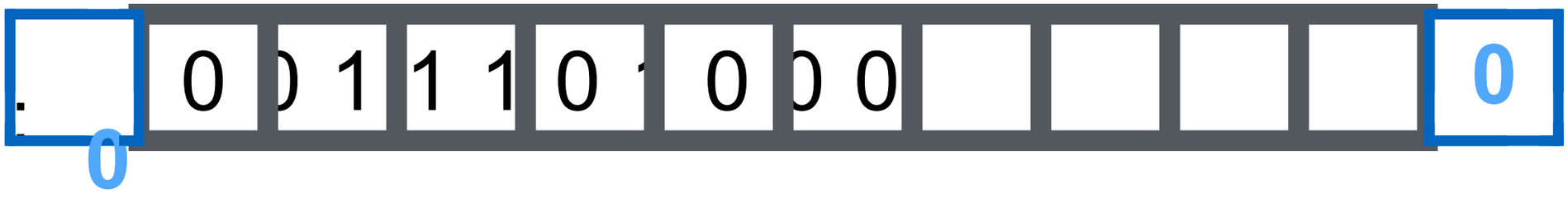
After ReLU:

- How many weights (including bias)? 4

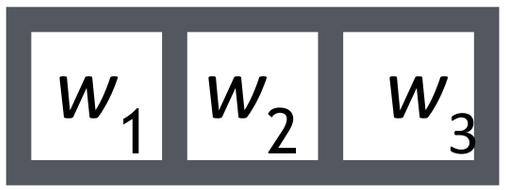
*correlation

Convolutional Layer: 1D example

A 1D image



A filter:



with bias b

After convolution*:

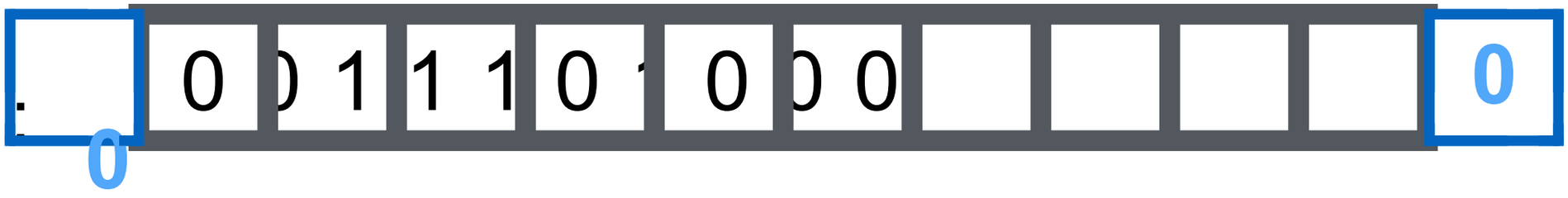
After ReLU:

- How many weights (including bias)? 4
- How many weights (including biases) for fully connected layer with 10 inputs & 10 outputs?

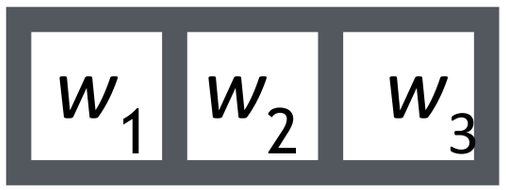
*correlation

Convolutional Layer: 1D example

A 1D image



A filter:



with bias b

After convolution*:

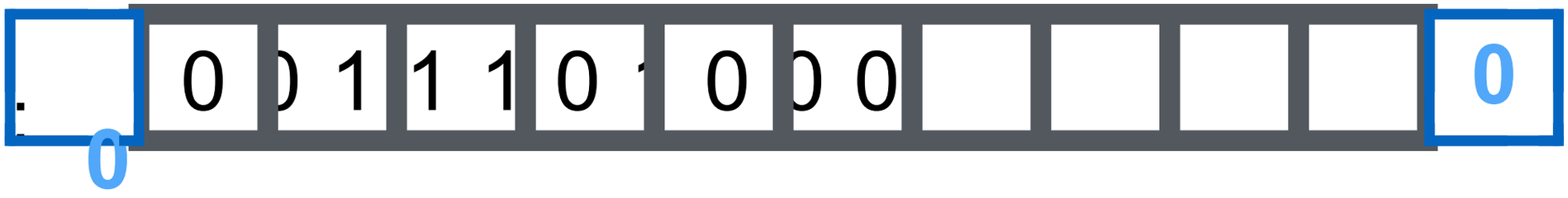
After ReLU:

- How many weights (including bias)? 4
- How many weights (including biases) for fully connected layer with 10 inputs & 10 outputs? 10 x 11 =

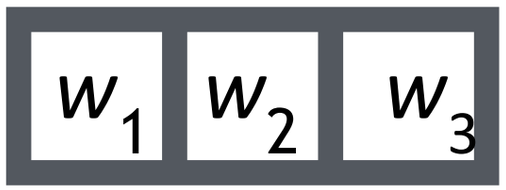
*correlation

Convolutional Layer: 1D example

A 1D image



A filter:



with bias b

After convolution*:

After ReLU:

- How many weights (including bias)? 4
- How many weights (including biases) for fully connected layer with 10 inputs & 10 outputs? 10 x 11 = 110

*correlation

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	1

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	1

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

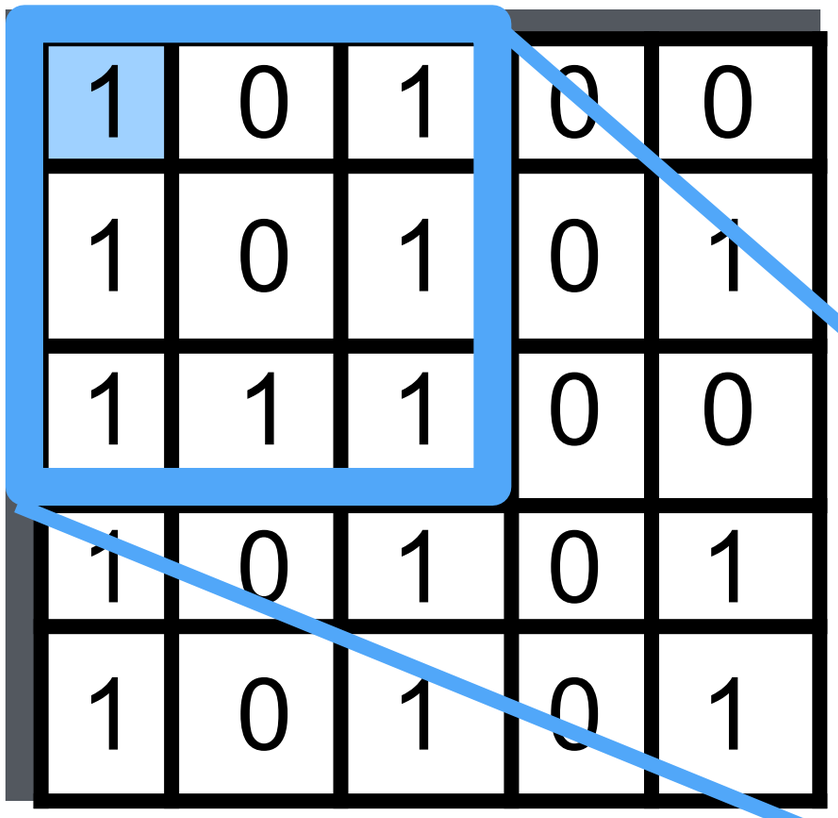
A filter:

-1	-1	1
-1	1	-1
-1	-1	1

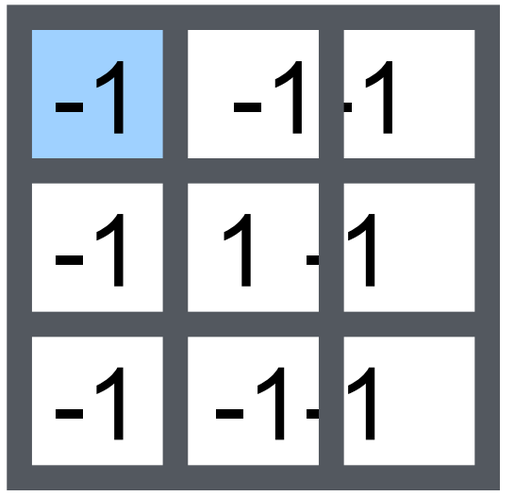
After convolution:

Convolutional Layer: 2D example

A 2D image:



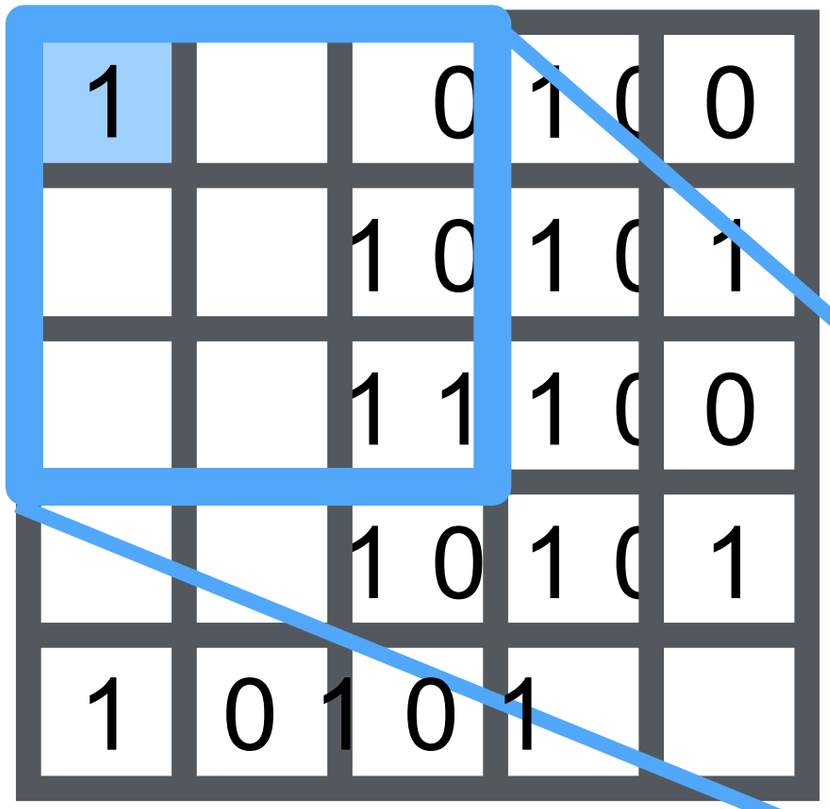
A filter:



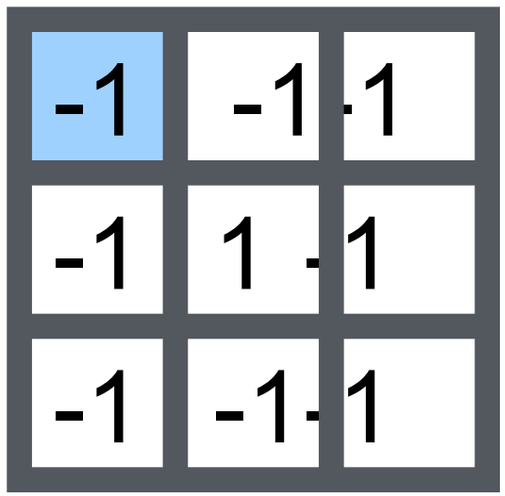
After convolution:

Convolutional Layer: 2D example

A 2D image:



A filter:

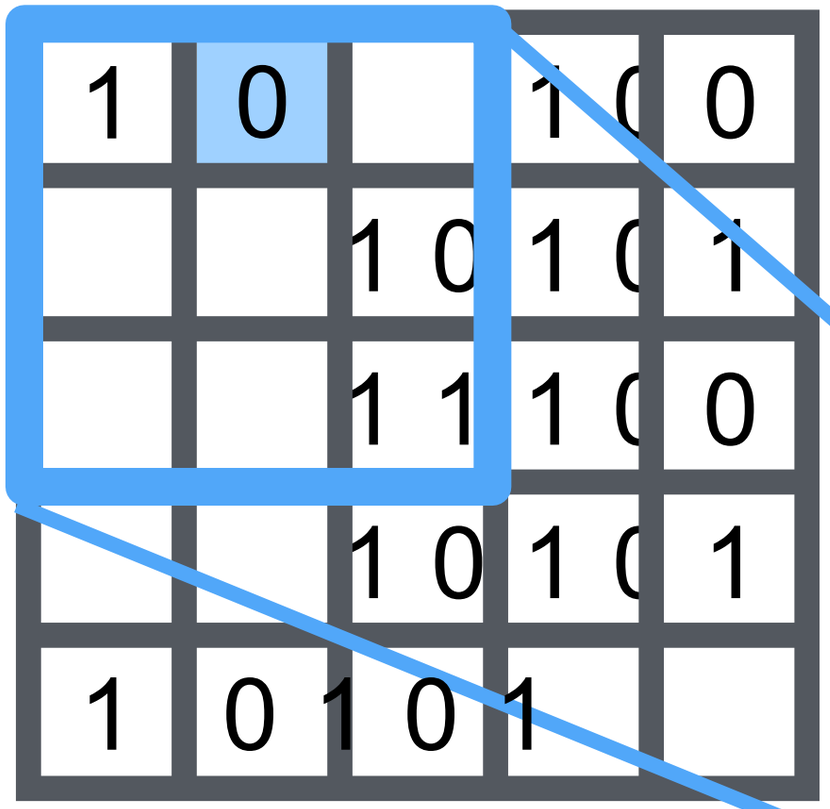


-1

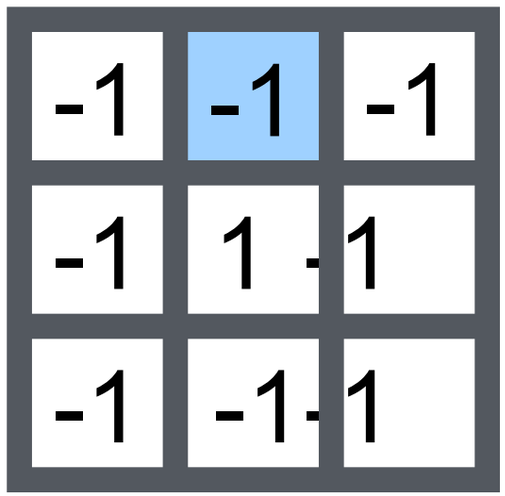
After convolution:

Convolutional Layer: 2D example

A 2D image:



A filter:



-1

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

-1 + 0

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

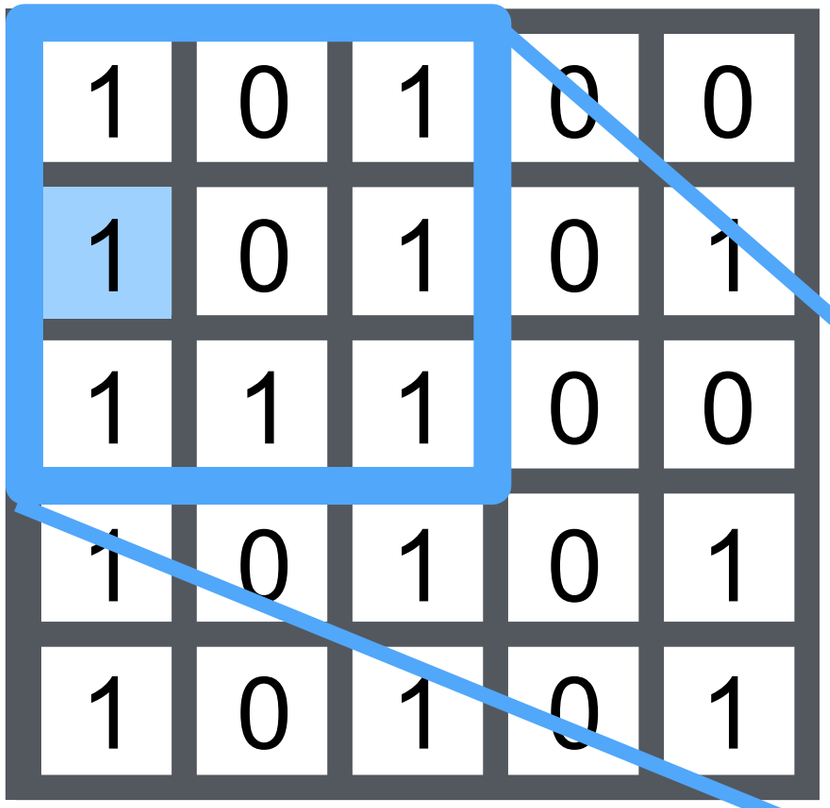
-1	-1	-1
-1	1	-1
-1	-1	-1

-1 + 0 + -1

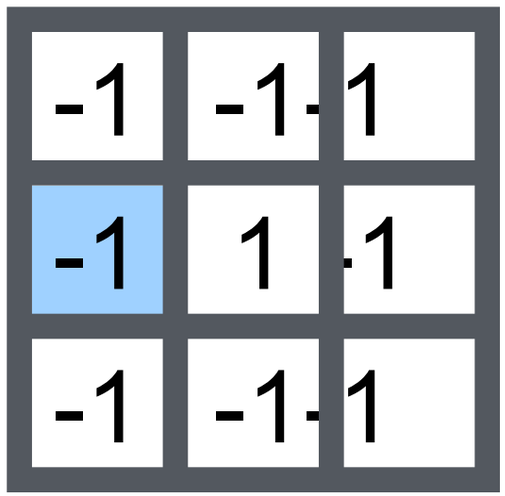
After convolution:

Convolutional Layer: 2D example

A 2D image:



A filter:

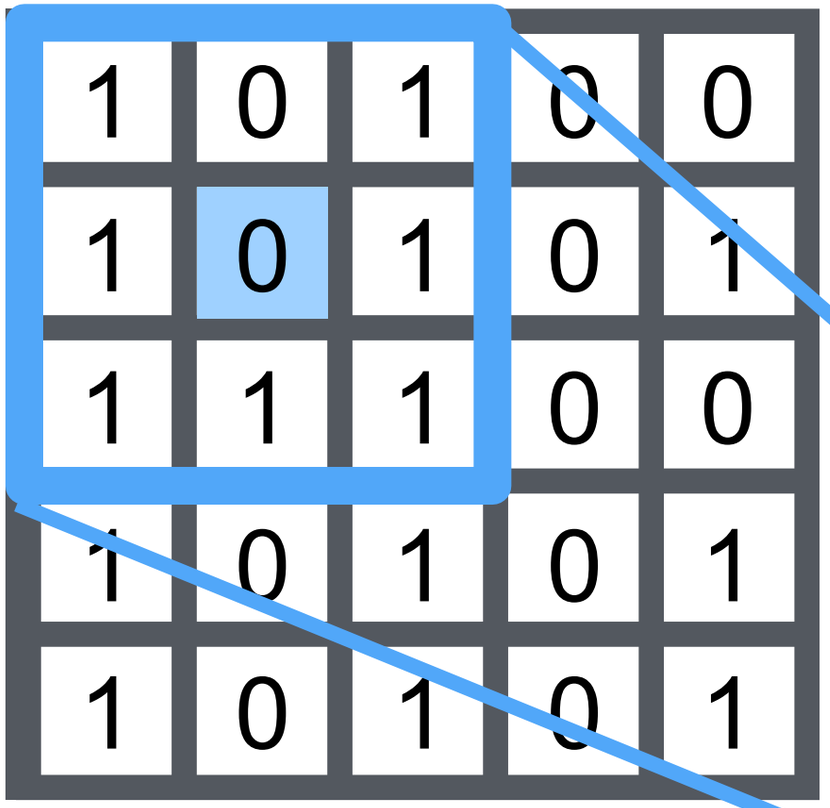


-1 + 0 + -1
+ -1

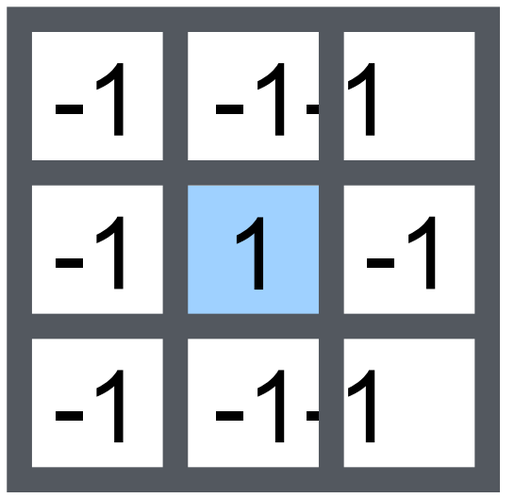
After convolution:

Convolutional Layer: 2D example

A 2D image:



A filter:



-1 + 0 + -1
+ -1 + 0

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	1

-1 + 0 + -1
+ -1 + 0 + -1

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

-1 + 0 + -1
+ -1 + 0 + -1
+ -1

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	-1

-1 + 0 + -1
+ -1 + 0 + -1
+ -1 + - 1

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

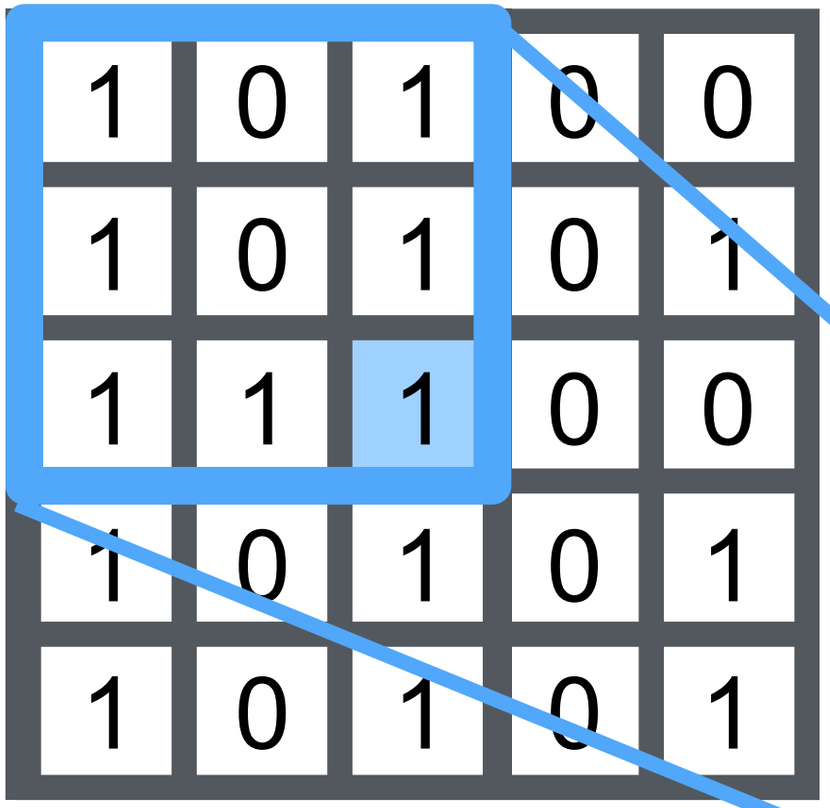
-1	-1	1
-1	1	-1
-1	-1	-1

-1 + 0 + -1
+ -1 + 0 + -1
+ -1 + - 1 + -1

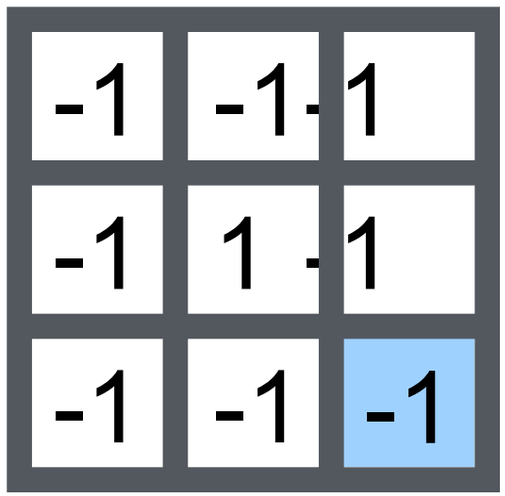
After convolution:

Convolutional Layer: 2D example

A 2D image:



A filter:



-1 + 0 + -1
+ -1 + 0 + -1
+ -1 + - 1 + -1
= -7

After convolution:

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	1

-1 + 0 + -1
+ -1 + 0 + -1
+ -1 + - 1 + -1
= -7

After convolution:

-7

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	1

After convolution:

-7

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	1

After convolution:

-7	-2

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	1
-1	1	-1
-1	-1	1

After convolution:

-7	-2	4

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

-7	-2	-4
-5		

Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

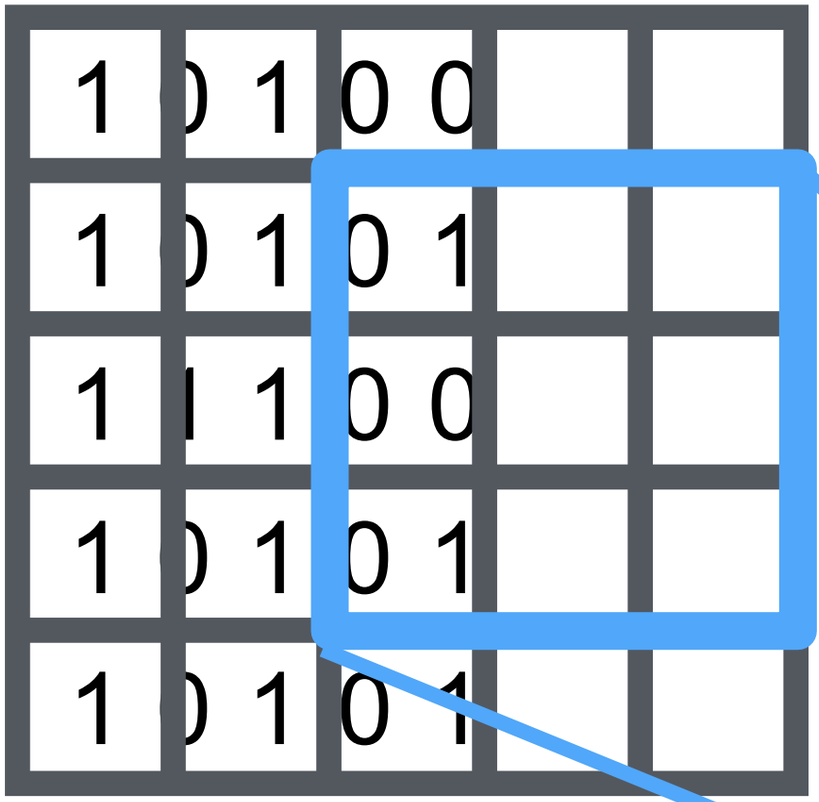
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

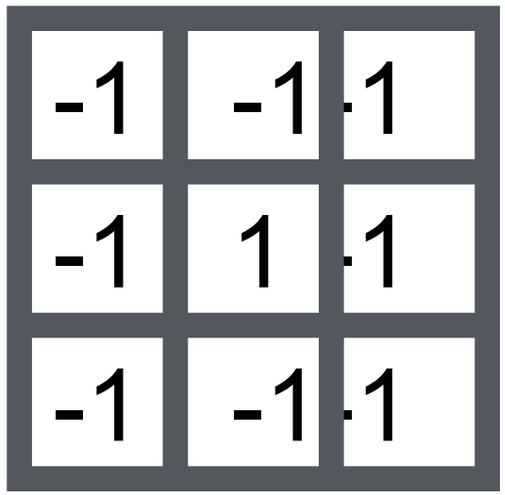
-7	-2	-4
-5		
-2		

Convolutional Layer: 2D example

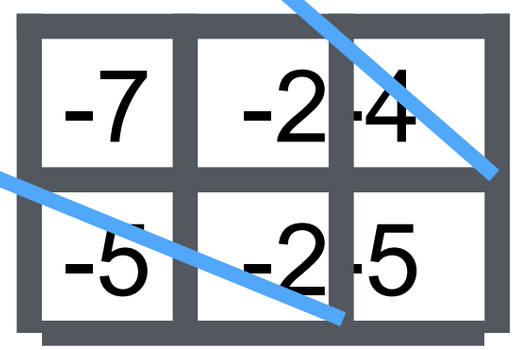
A 2D image:



A filter:



After convolution:



Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

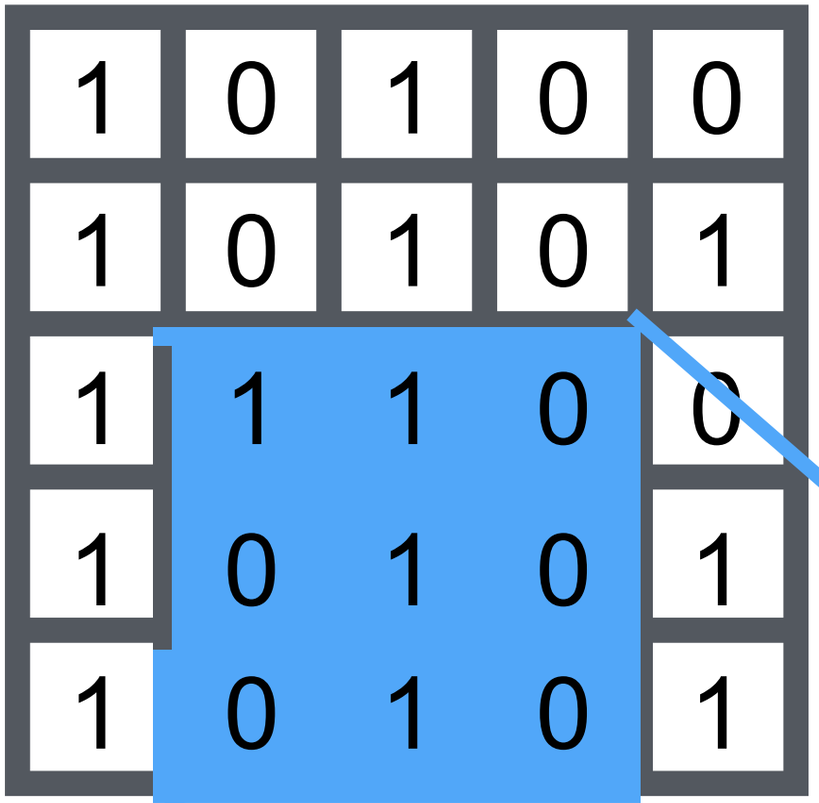
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

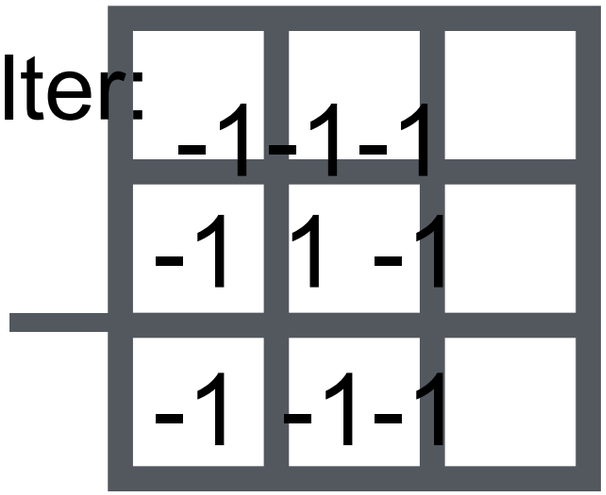
-7	-2	-4
-5	-2	-5
-7		

Convolutional Layer: 2D example

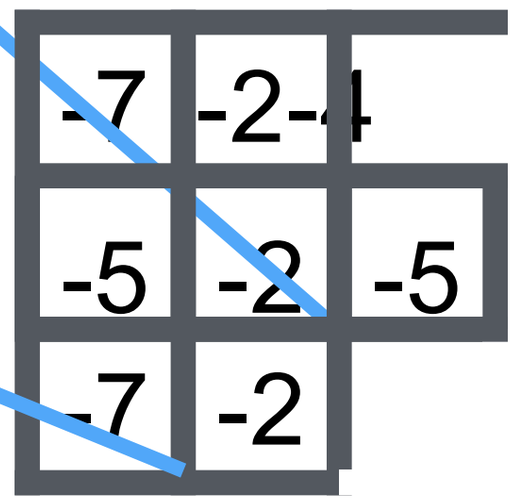
A 2D image:



A filter:



After convolution:



Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

-7	-2	-4
-5	-2	-5
-7	-2	-5

Convolutional Layer: 2D example

A 2D image:

1	0	0	0	
1	0	0	1	
1	1	0	0	
1	0	0	1	
1	0	0	1	

A filter:

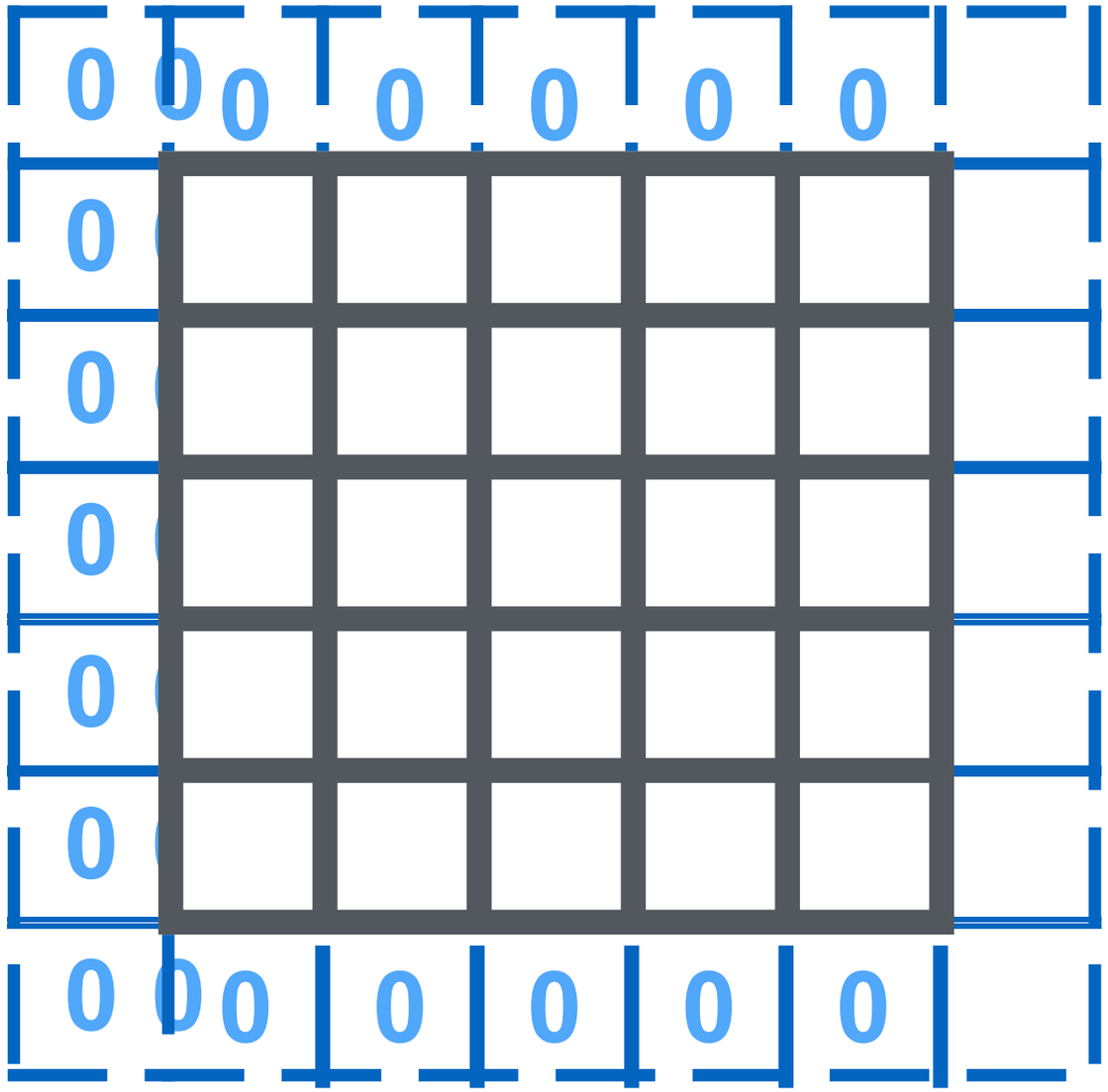
-1	-1	1
-1	1	-1
-1	-1	1

After convolution:

-7	-2	4
-5	-2	5
-7	-2	5

Convolutional Layer: 2D example

A 2D image:



A filter:

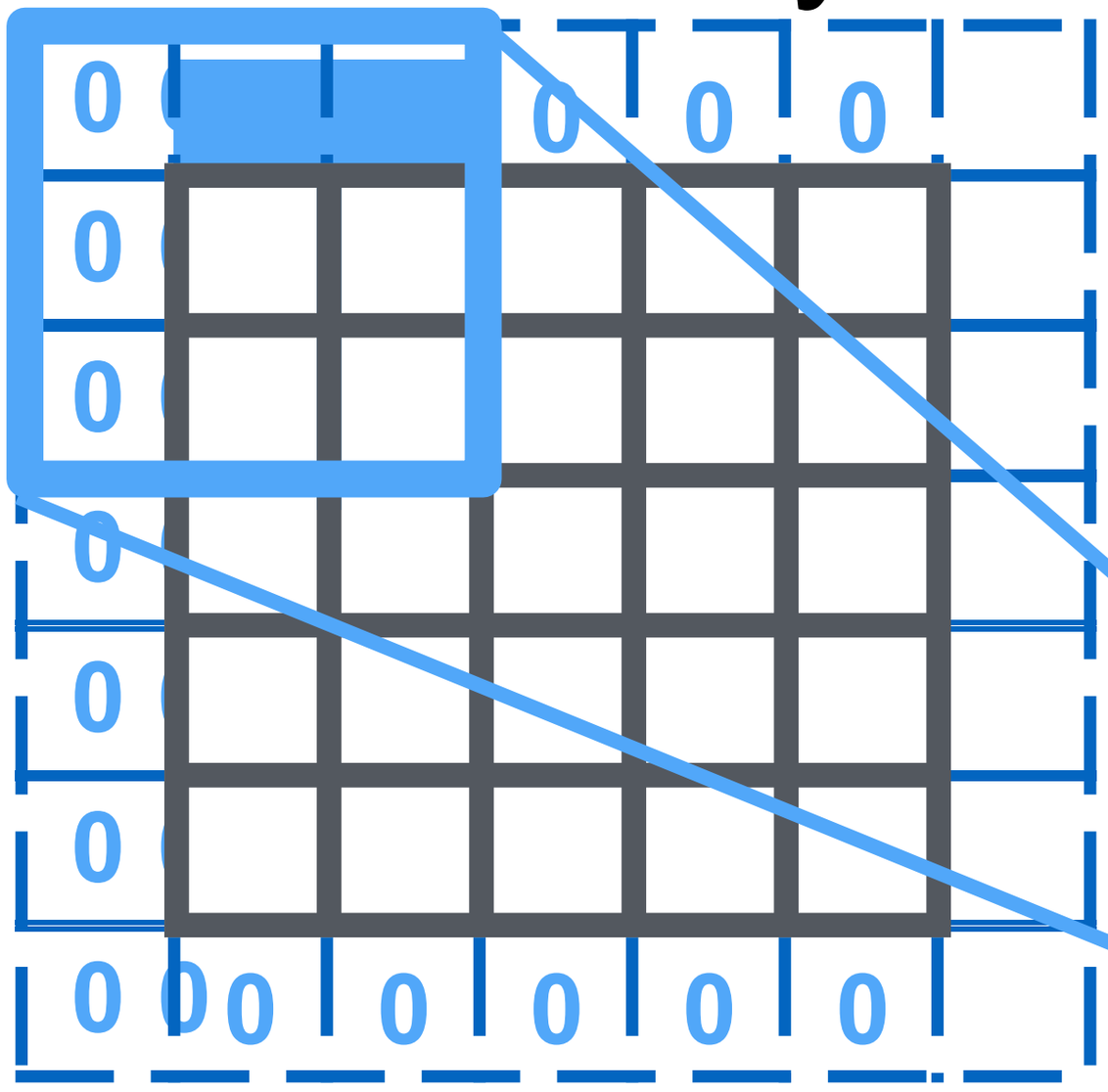
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

	-7	-2 -4
-5	-2	-5
-7	-2	-5

Convolutional Layer: 2D example

A 2D image:



A filter:

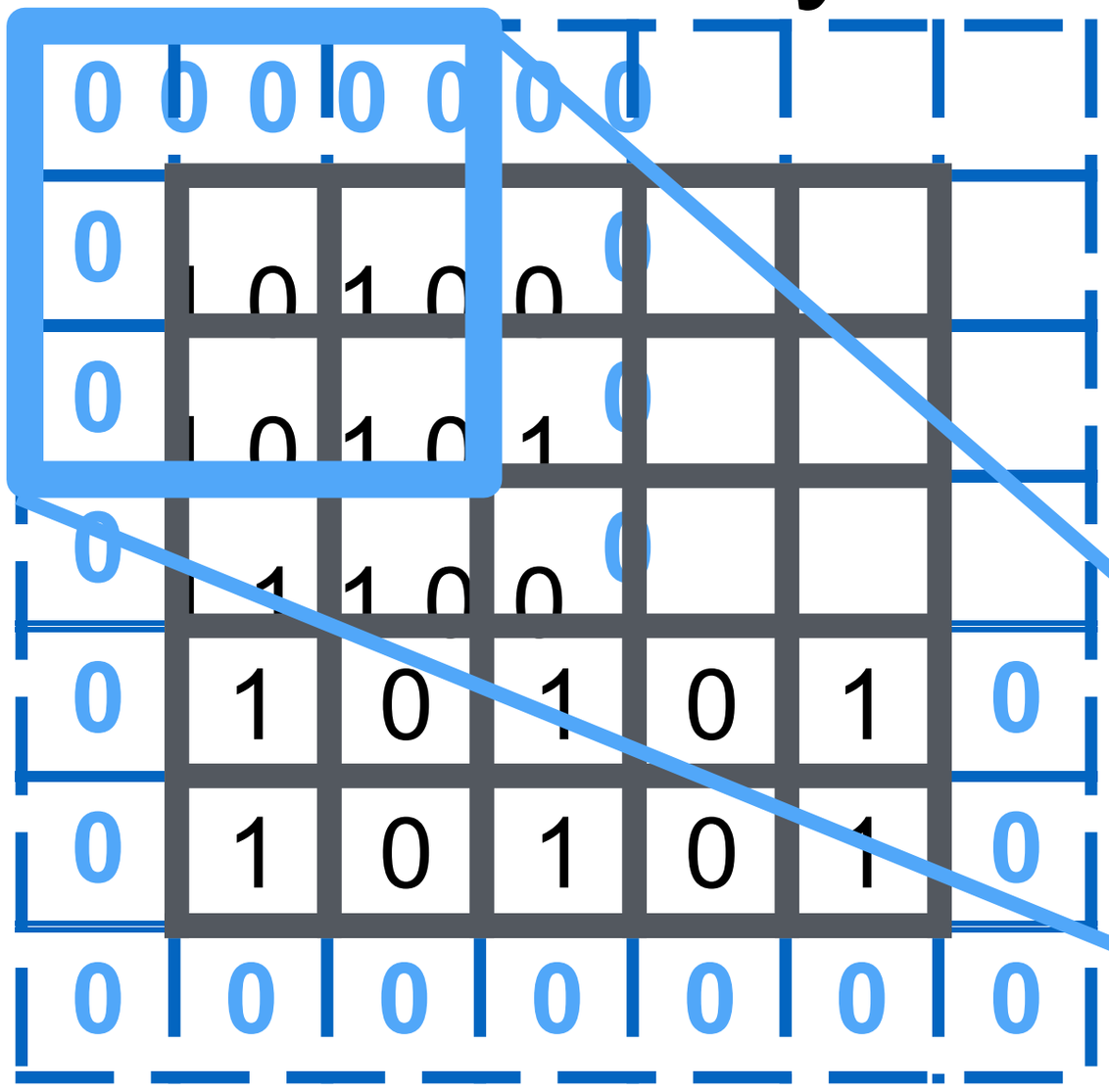
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

	-7	-2 -4
-5	-2	-5
-7	-2	-5

Convolutional Layer: 2D example

A 2D image:



A filter:

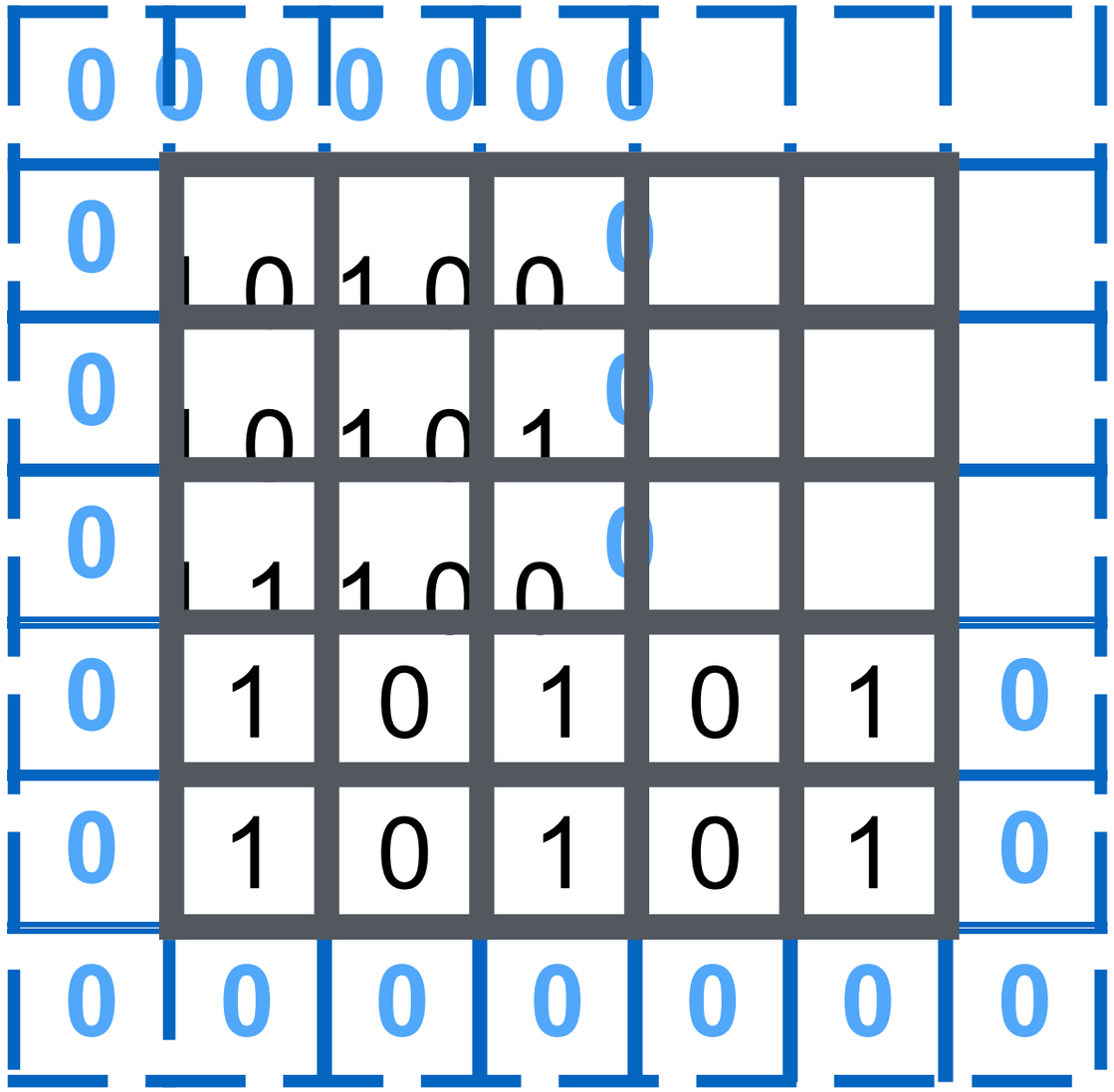
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

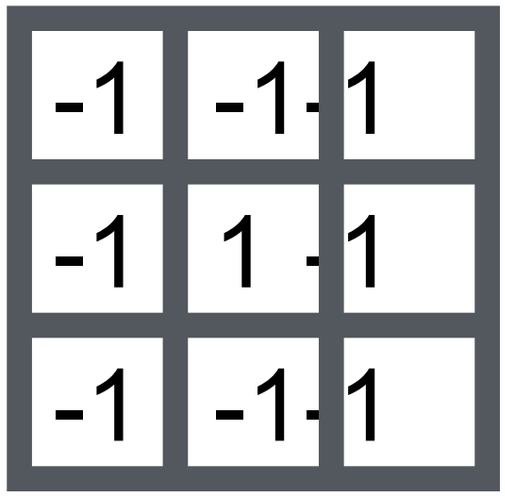
-7	-2	-4
-5	-2	-5
-7	-2	-5

Convolutional Layer: 2D example

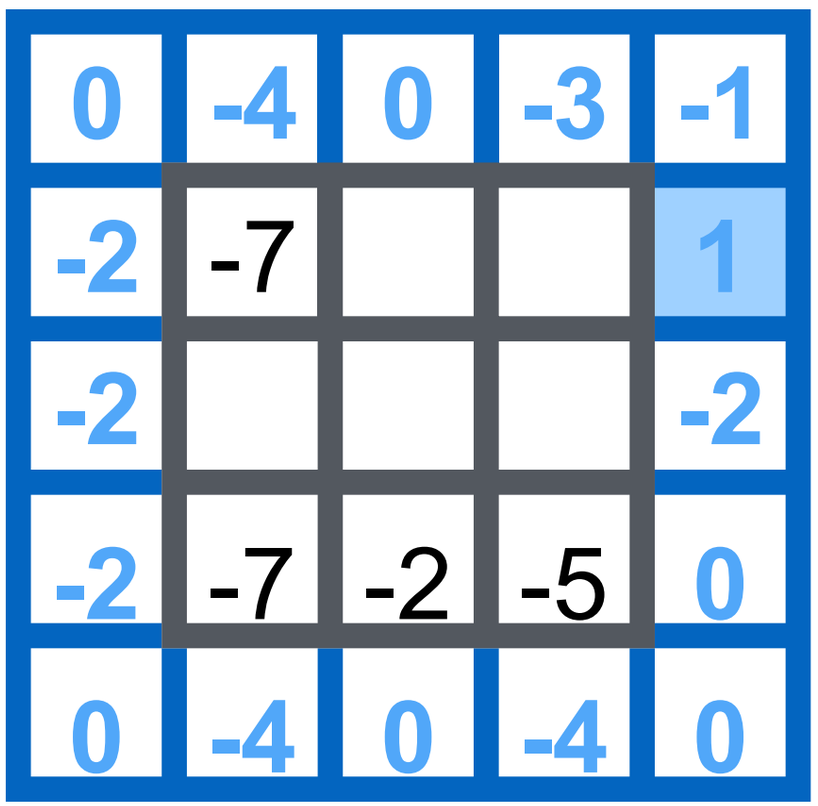
A 2D image:



A filter:

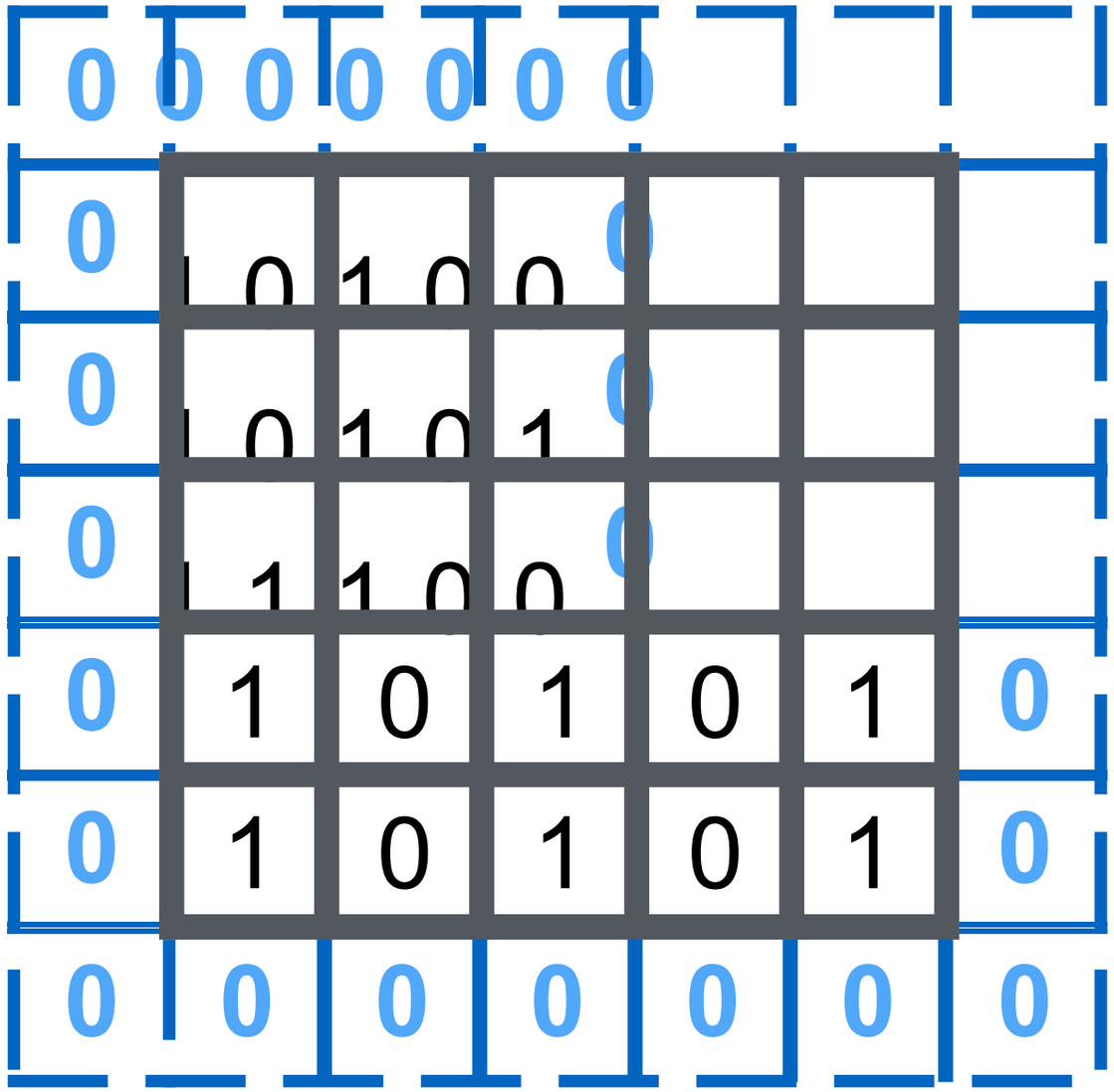


After convolution & ReLU:

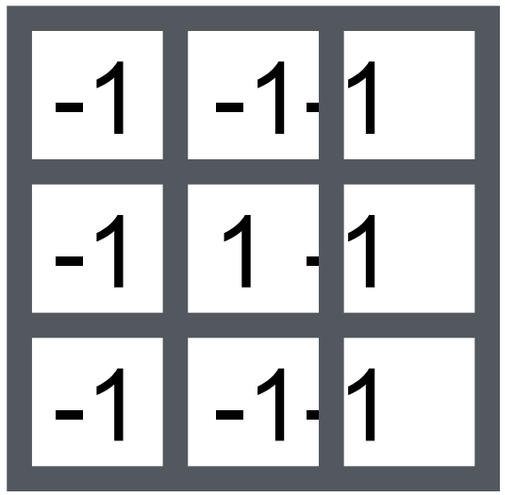


Convolutional Layer: 2D example

A 2D image:



A filter:



Detecting isolated 1's.

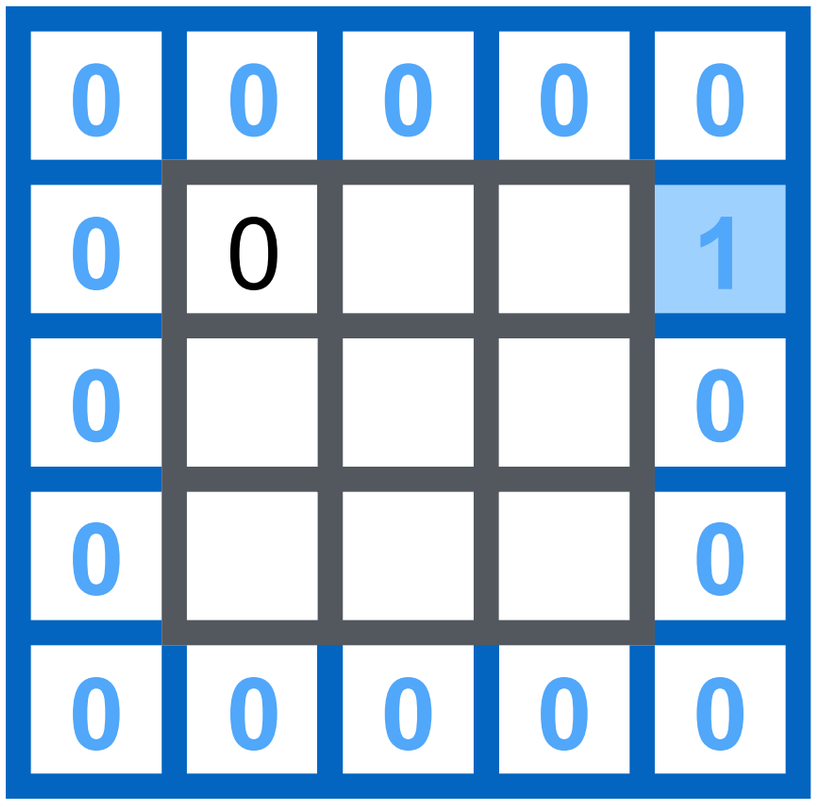
How do we detect all the 1's?

-- Filter - 1 surrounded by 0's.

How do we detect edges?

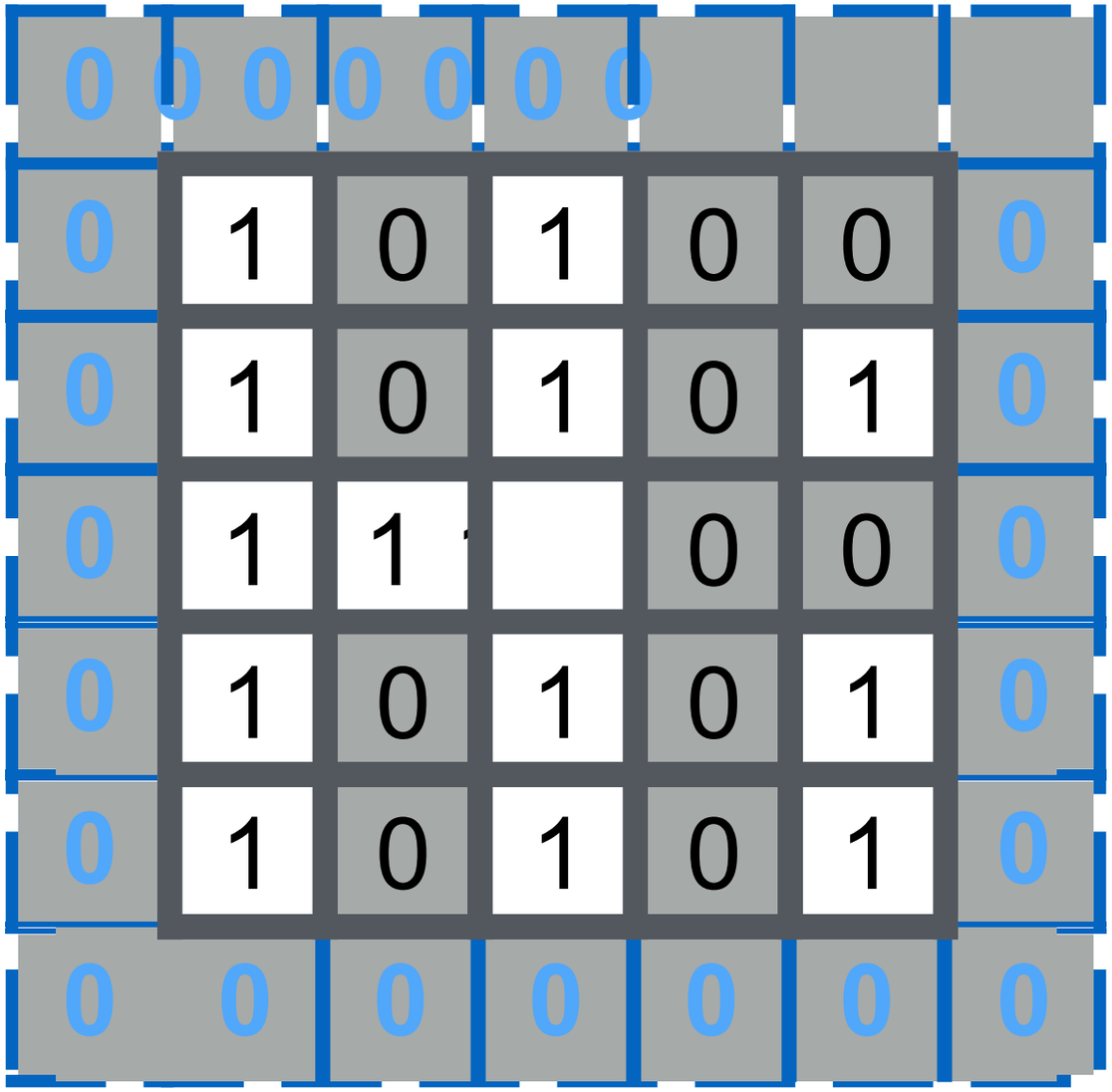
-- Exercise?

After convolution & ReLU:



Convolutional Layer: 2D example

A 2D image:



A filter:

-1	-1	1
-1	1	-1
-1	-1	1

Detecting isolated 1's.

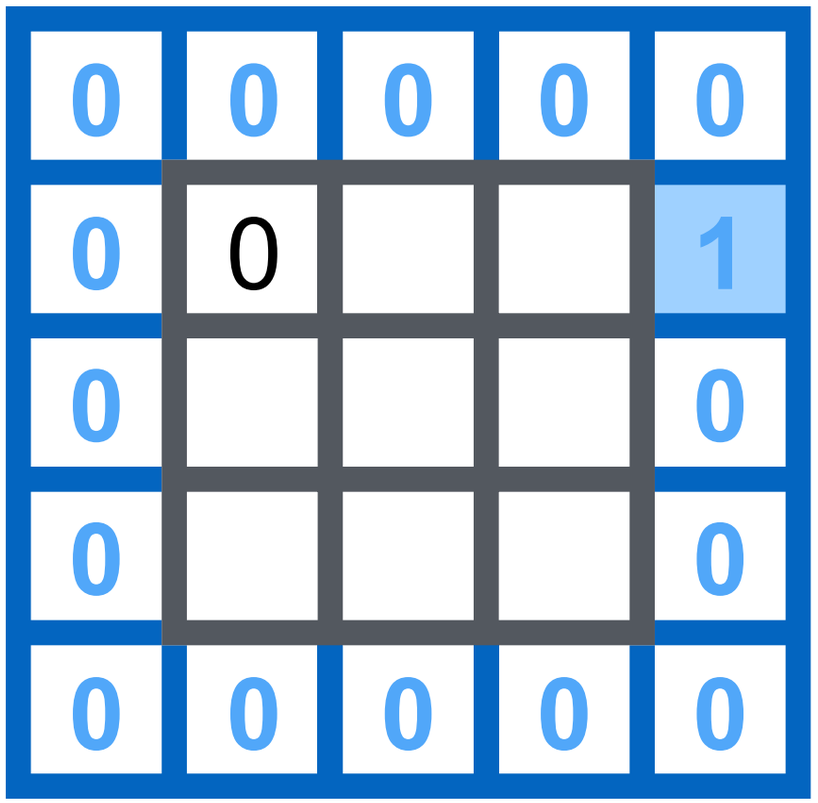
How do we detect all the 1's?

-- Filter - 1 surrounded by 0's.

How do we detect edges?

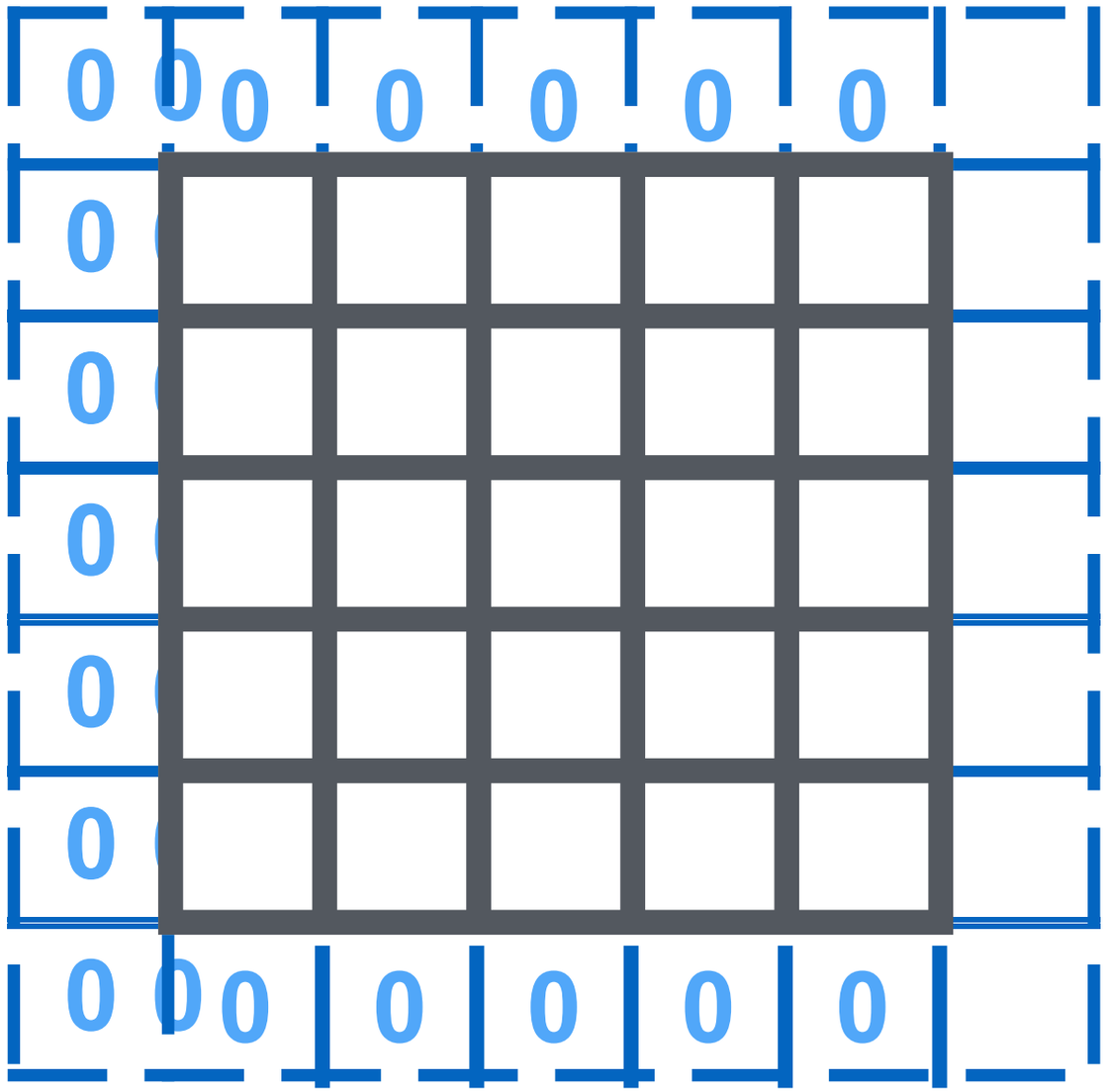
-- Exercise?

After convolution & ReLU:



Convolutional Layer: 2D example

A 2D image:

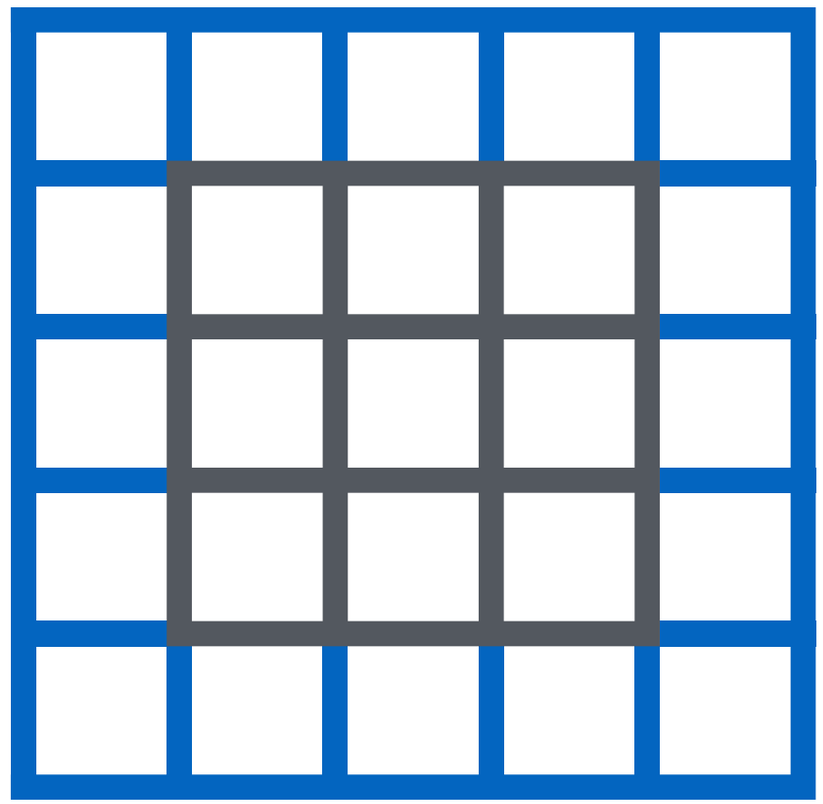


A filter:

-1	-1	1
-1	1	-1
-1	-1	1

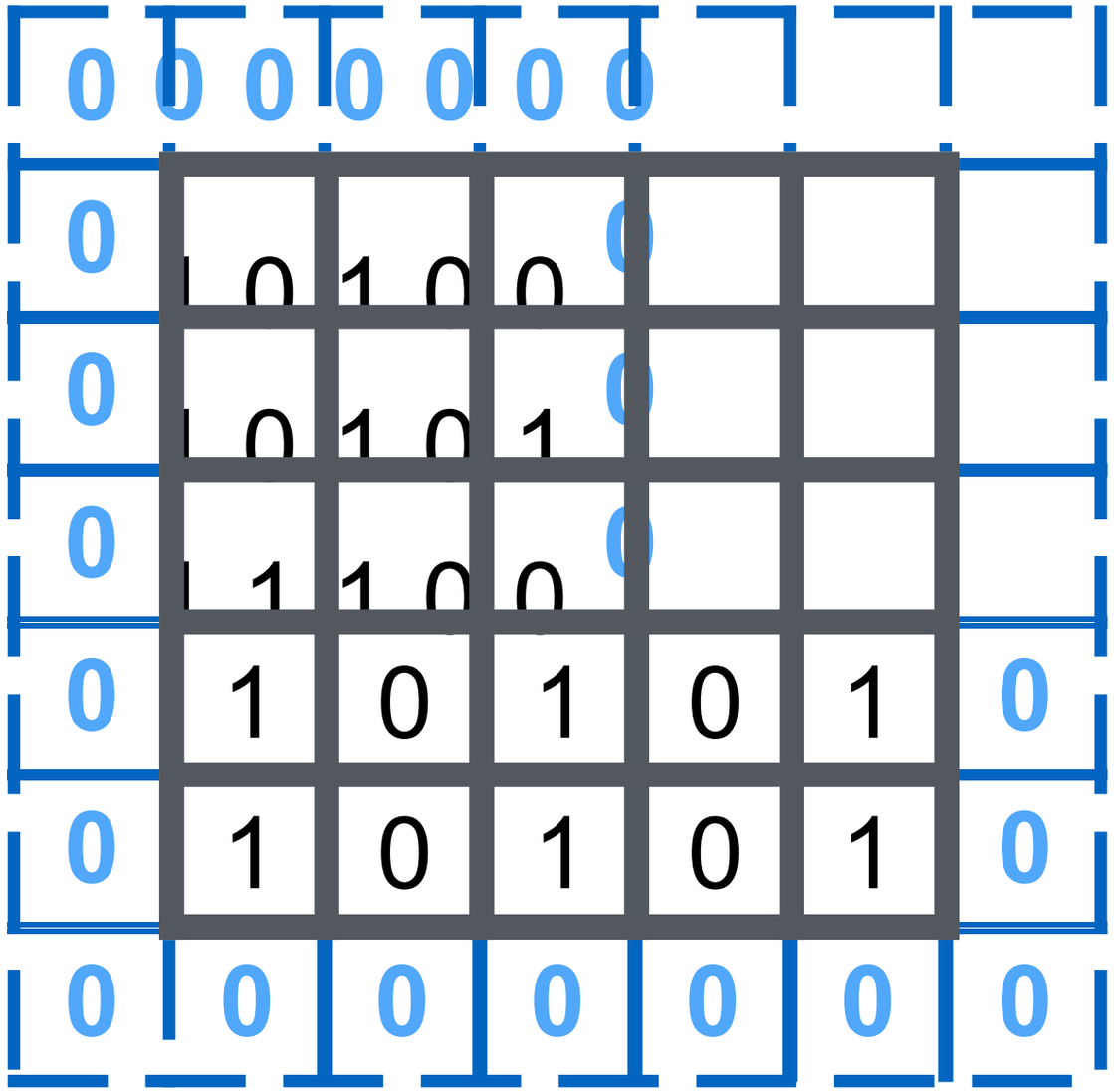
with bias 2

After convolution:

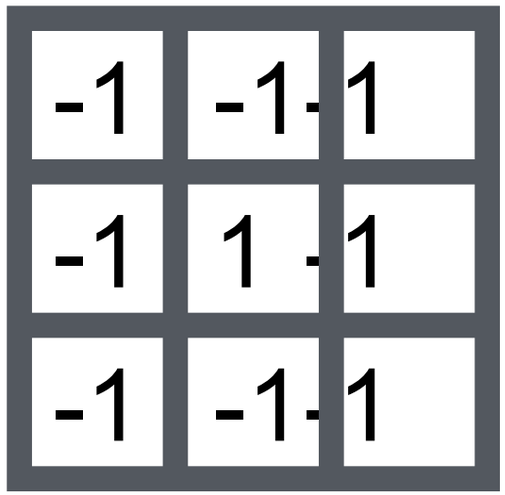


Convolutional Layer: 2D example

A 2D image:



A filter:

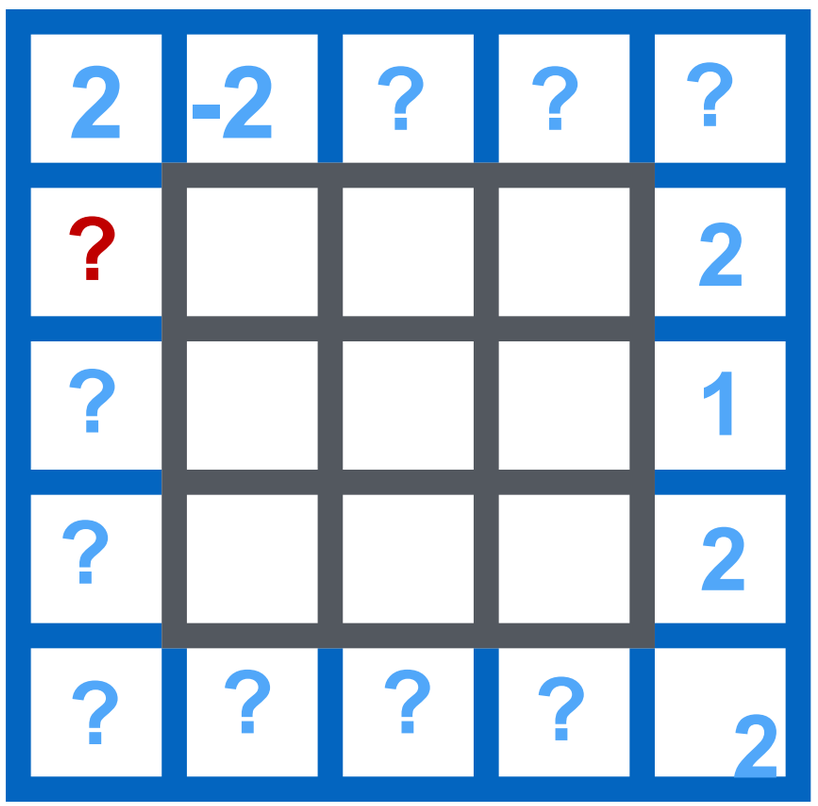


with bias 2

What does it detect?

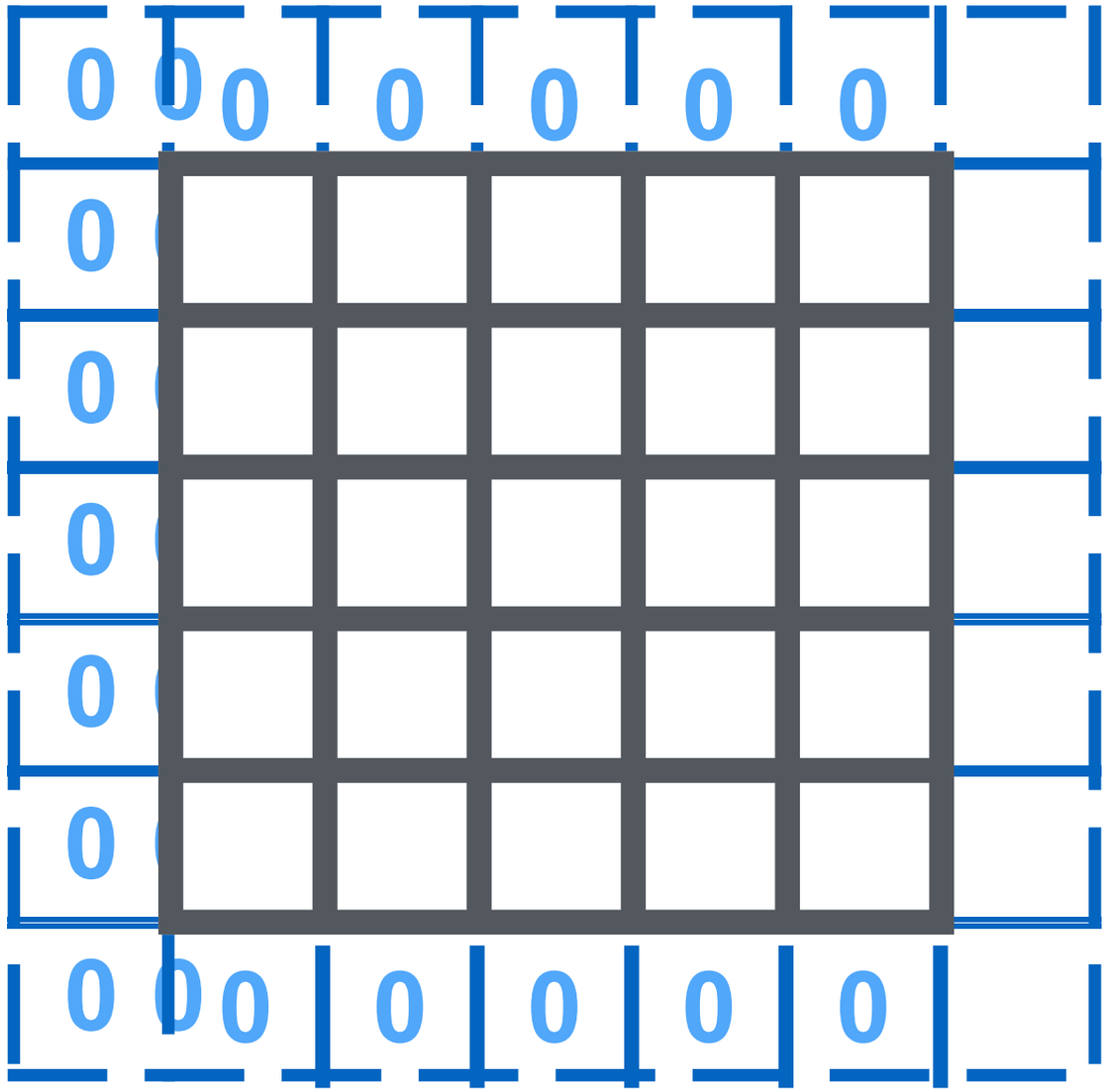
- **Corner points and**
- **isolated ones**

After convolution:



Convolutional Layer: 2D example

A 2D image:

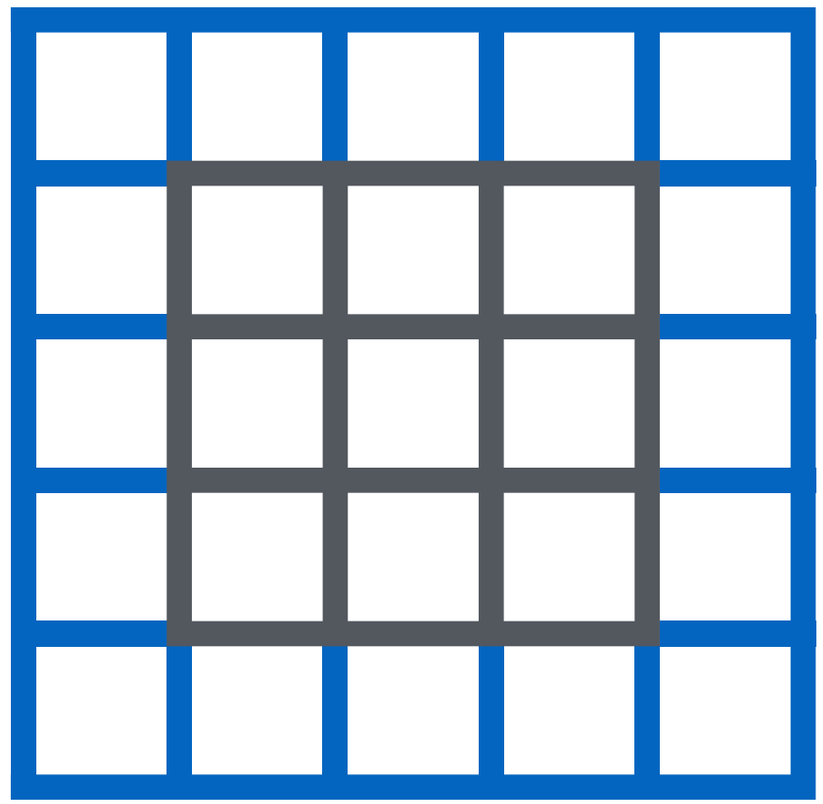


A filter:

-1	-1	1
-1	1	-1
-1	-1	1

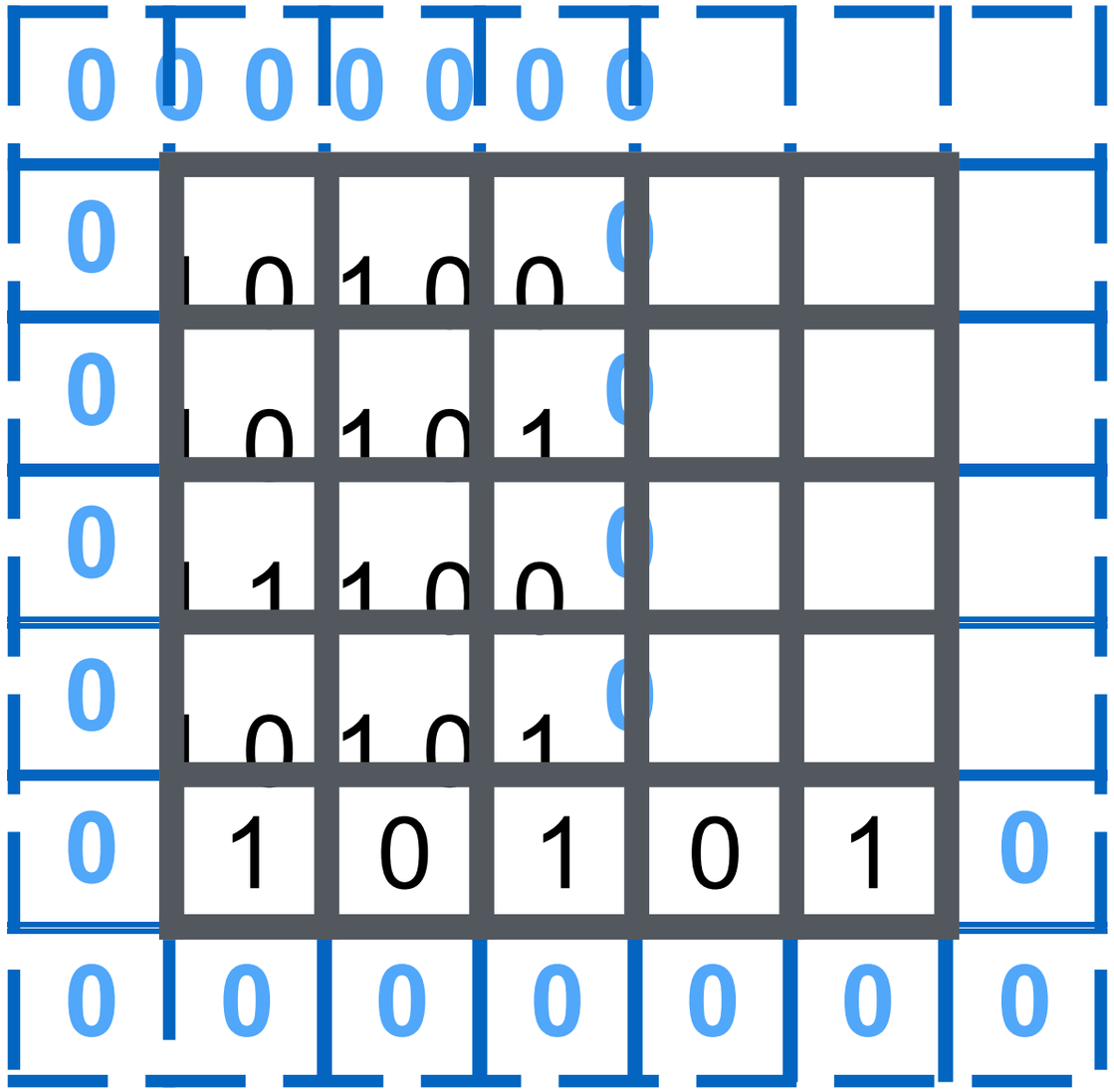
with bias 2

After convolution:

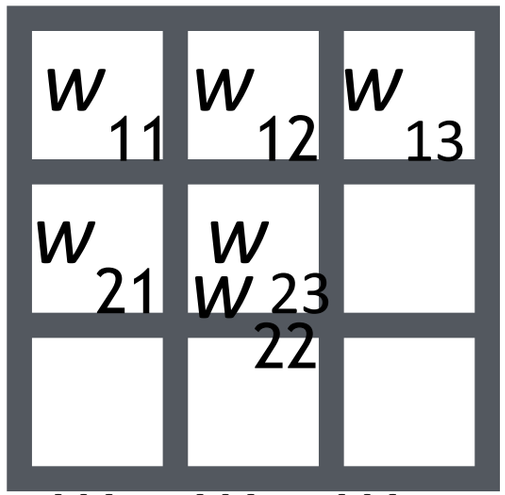


Convolutional Layer: 2D example

A 2D image:

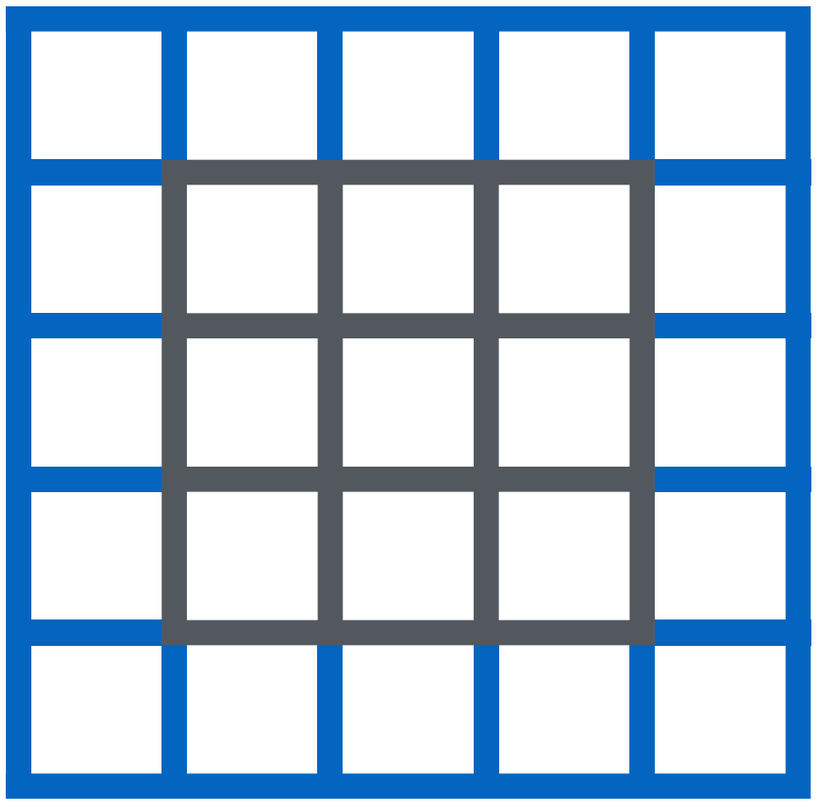


A filter:



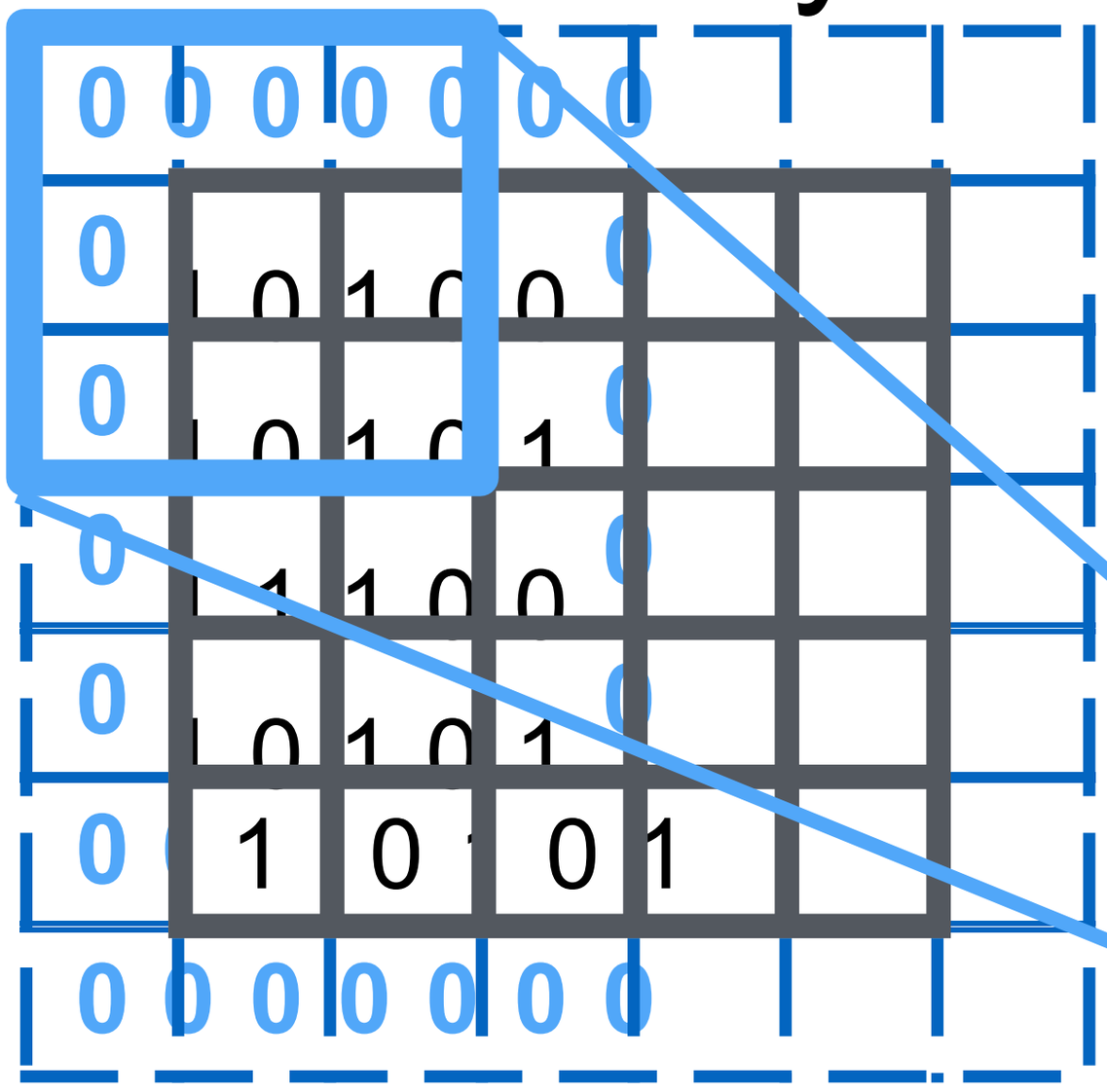
with bias b

After convolution:

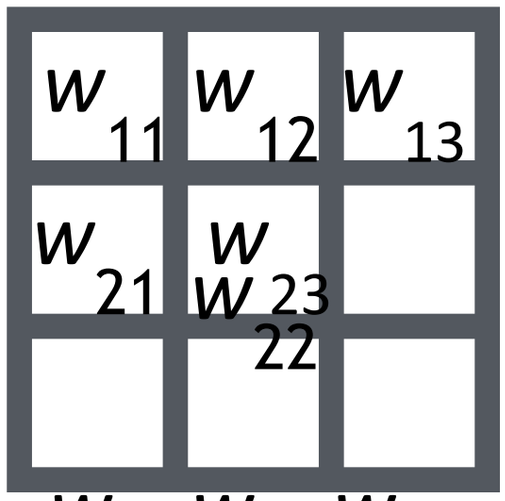


Convolutional Layer: 2D example

A 2D image:

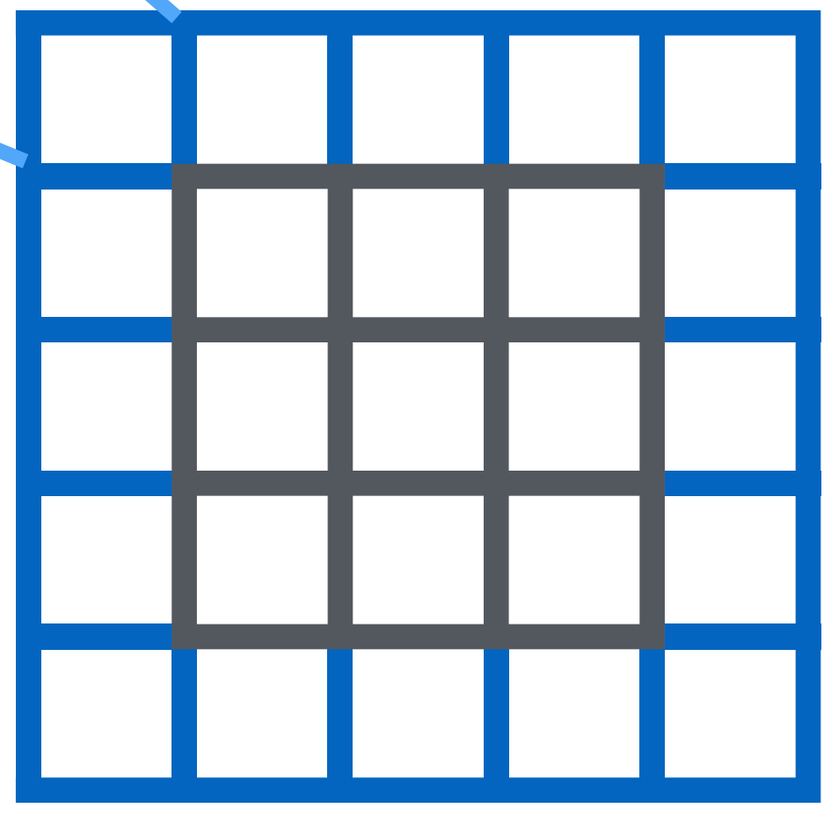


A filter:



with bias b

After convolution:



Convolutional Layer: 2D example

A 2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

A filter:

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

with bias b

What is the output size?

Convolutional Layer: 3D example

A 3D image:

[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

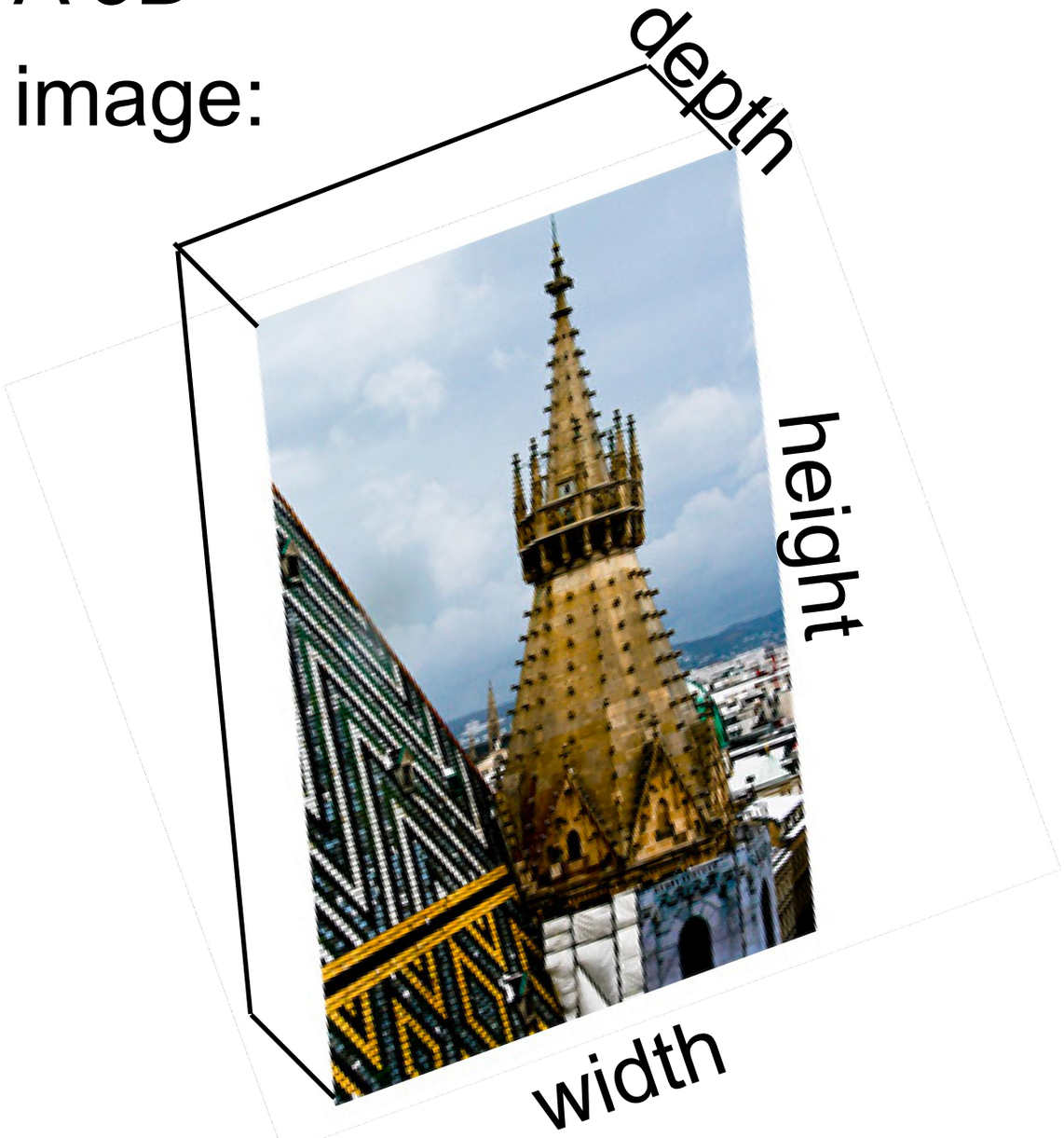
Convolutional Layer: 3D example

A 3D image:

[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

Convolutional Layer: 3D example

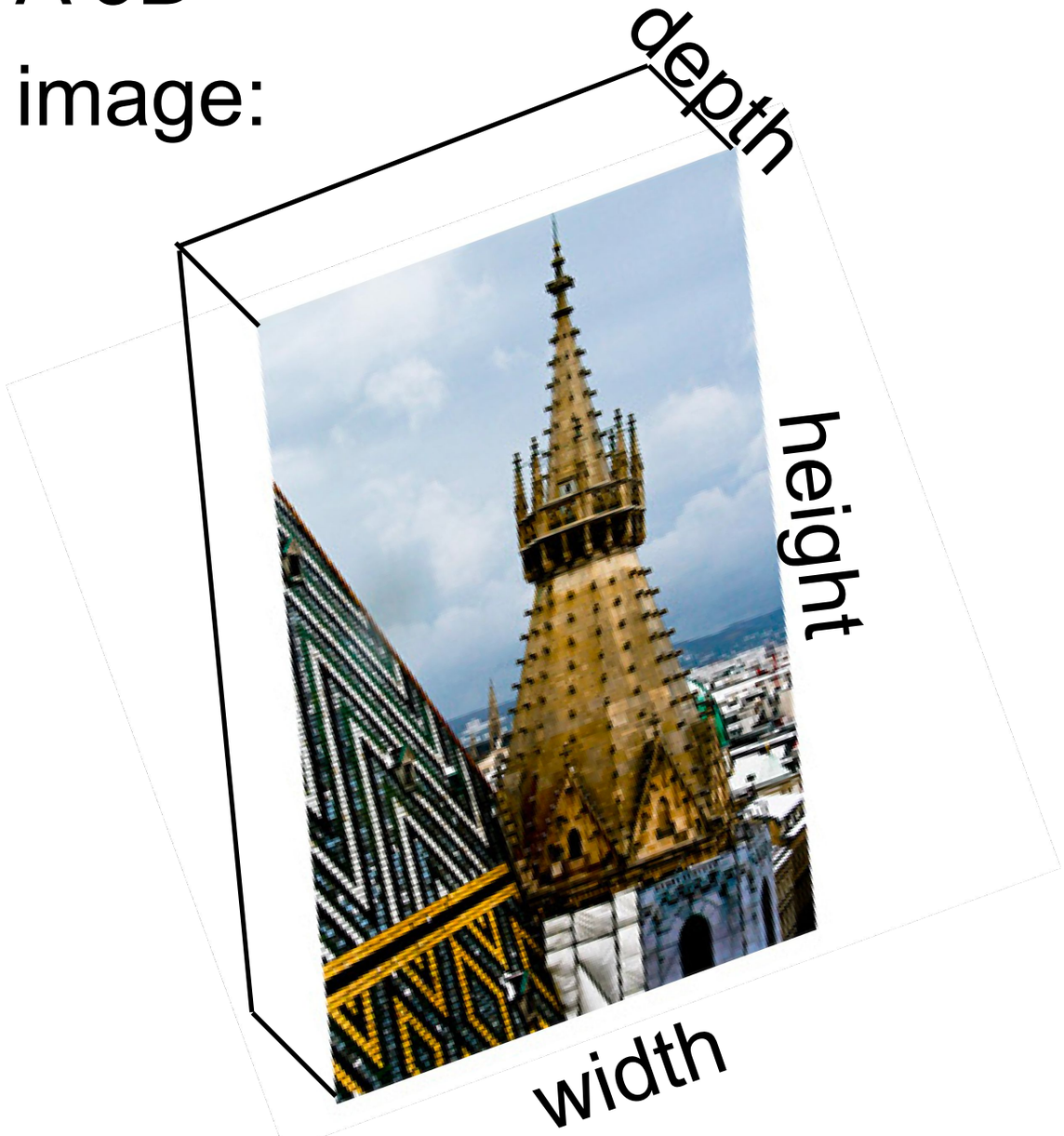
A 3D image:



[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

Convolutional Layer: 3D example

A 3D image:

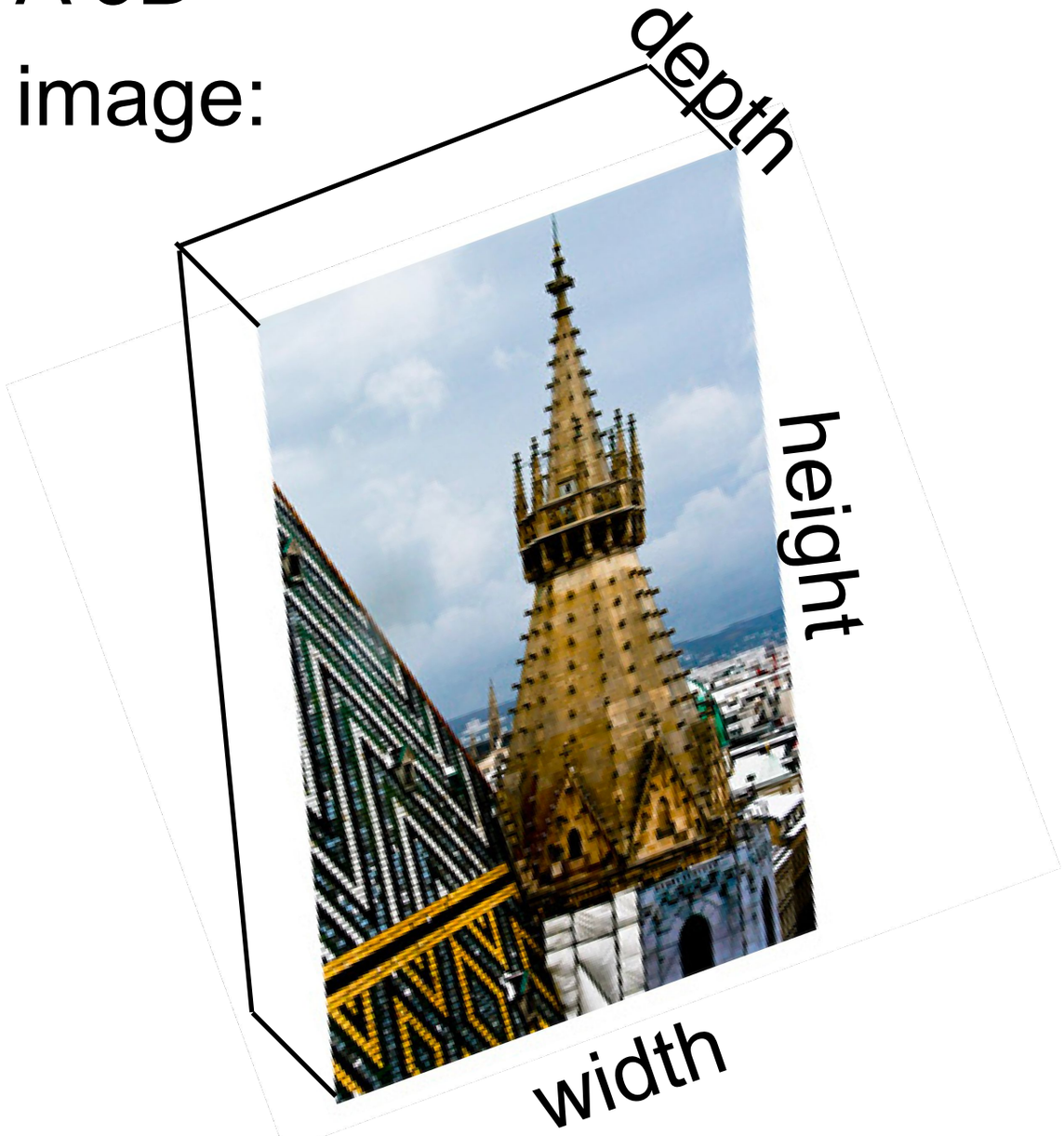


[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

Convolutional Layer: 3D example

A 3D image:

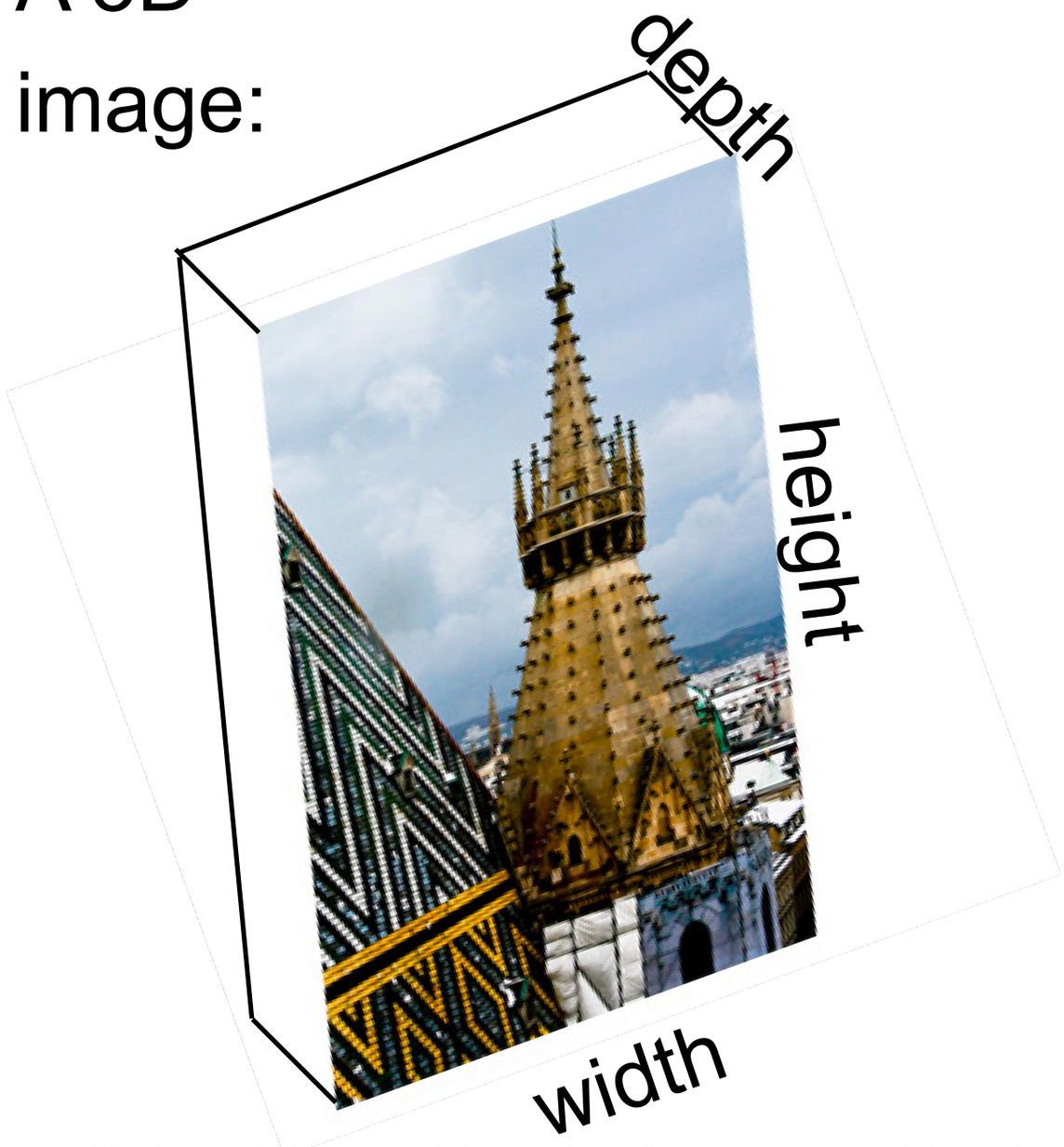


[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

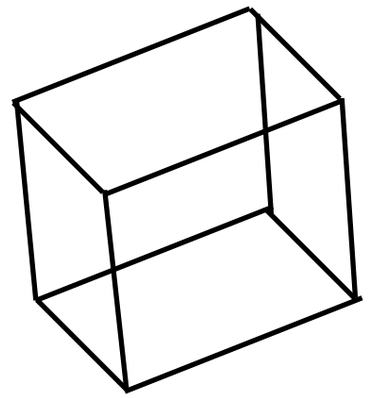
Convolutional Layer: 3D example

A 3D image:



[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

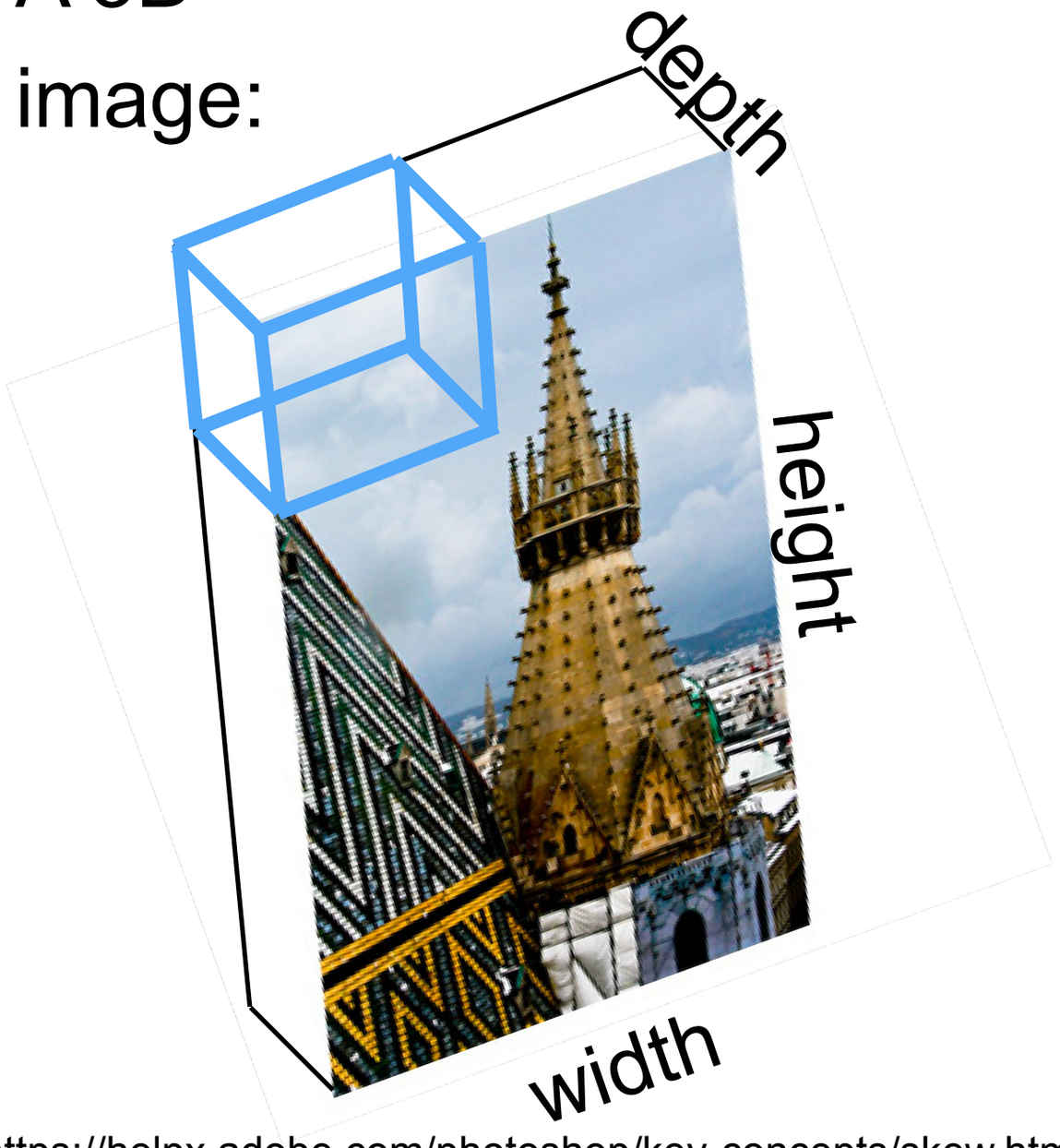
A filter:



- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

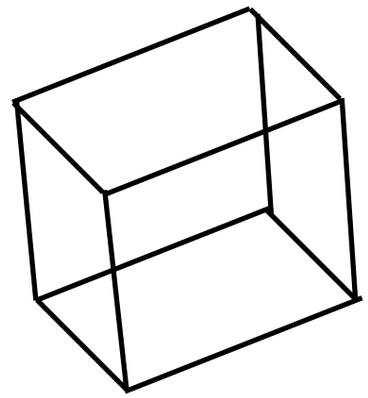
Convolutional Layer: 3D example

A 3D image:



[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

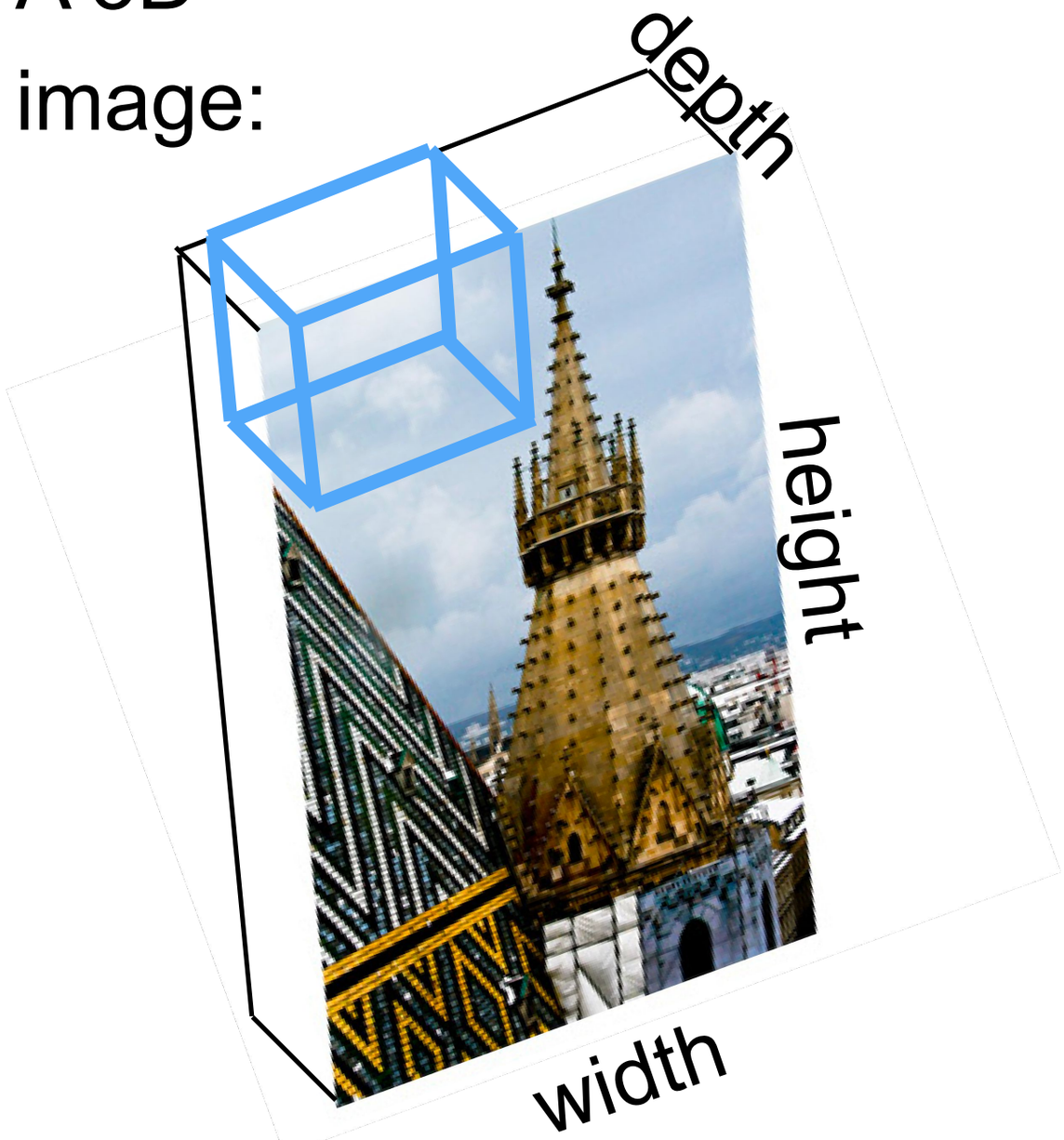
A filter:



- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

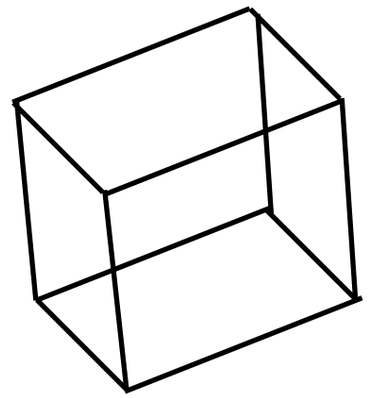
Convolutional Layer: 3D example

A 3D image:



[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

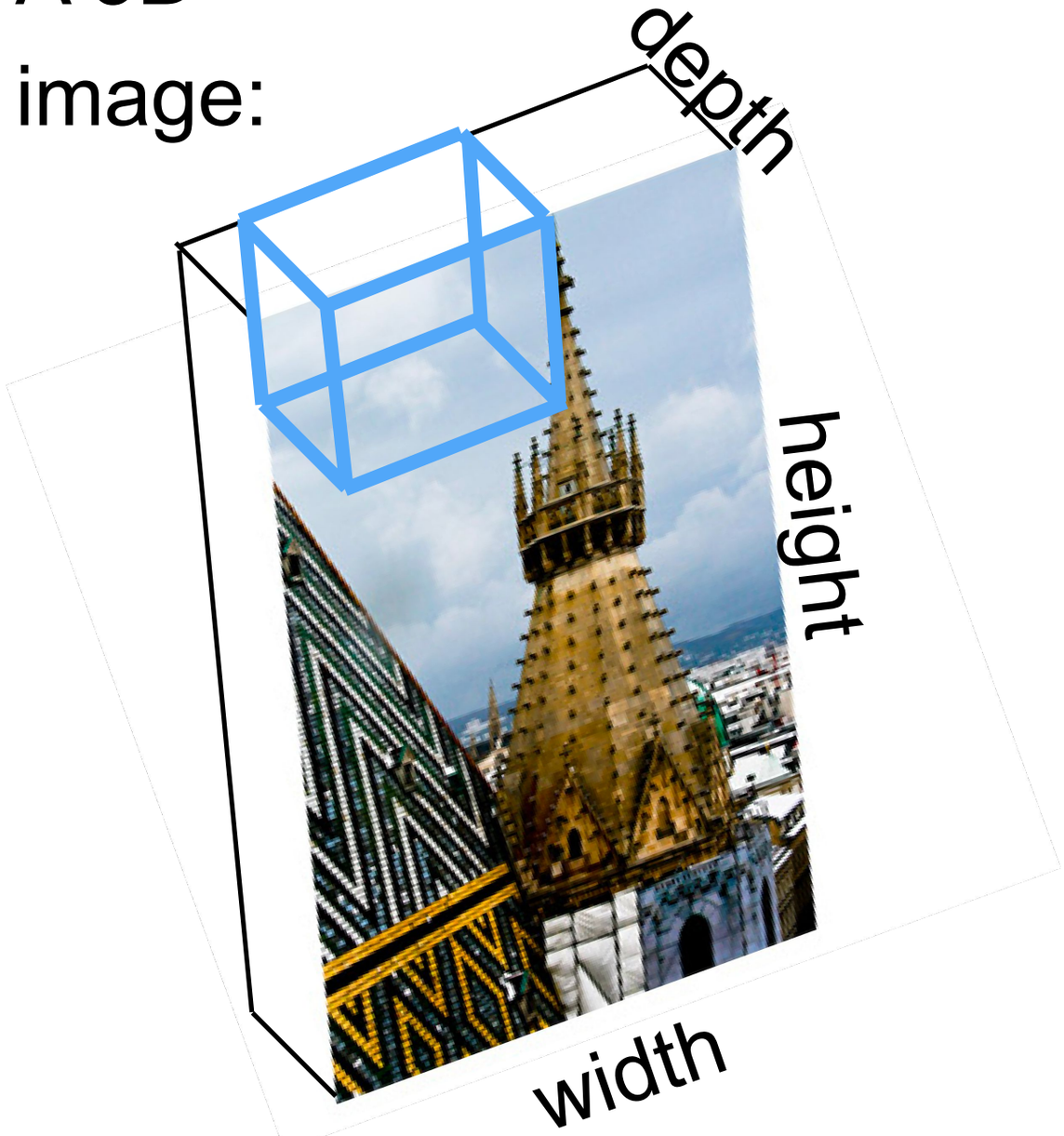
A filter:



- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

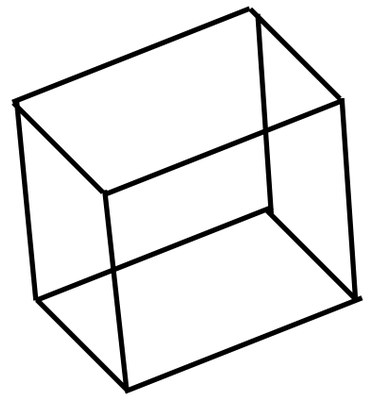
Convolutional Layer: 3D example

A 3D image:



[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

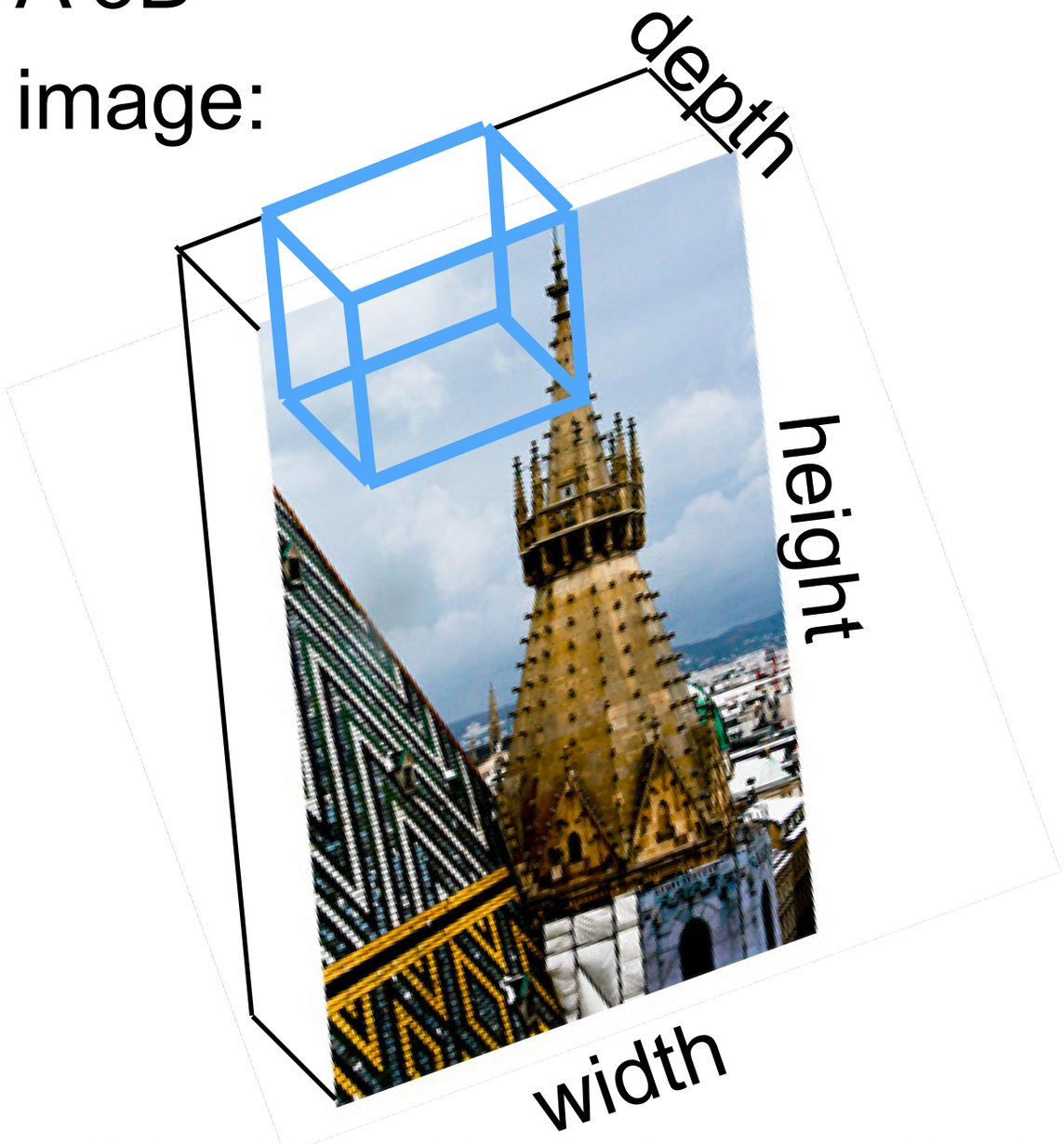
A filter:



- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

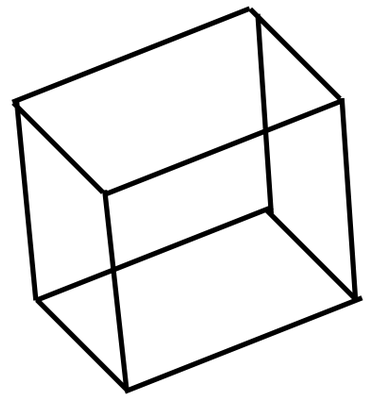
Convolutional Layer: 3D example

A 3D image:



[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

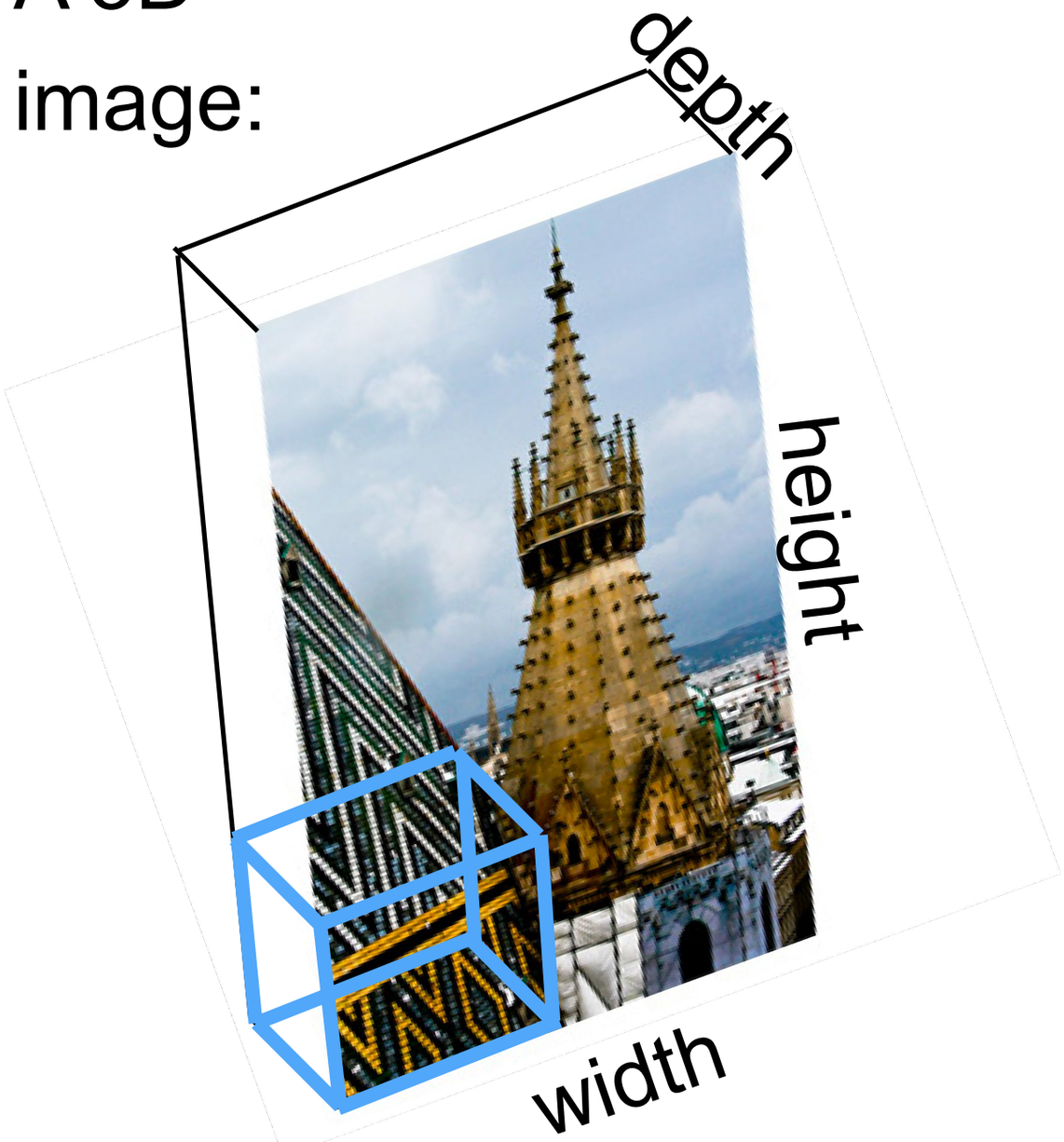
A filter:



- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

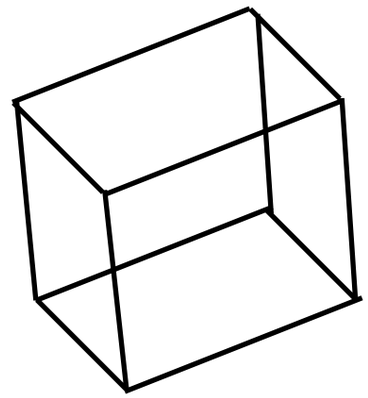
Convolutional Layer: 3D example

A 3D image:



[https://helpx.adobe.com/photoshop/key-concepts/skew.html]

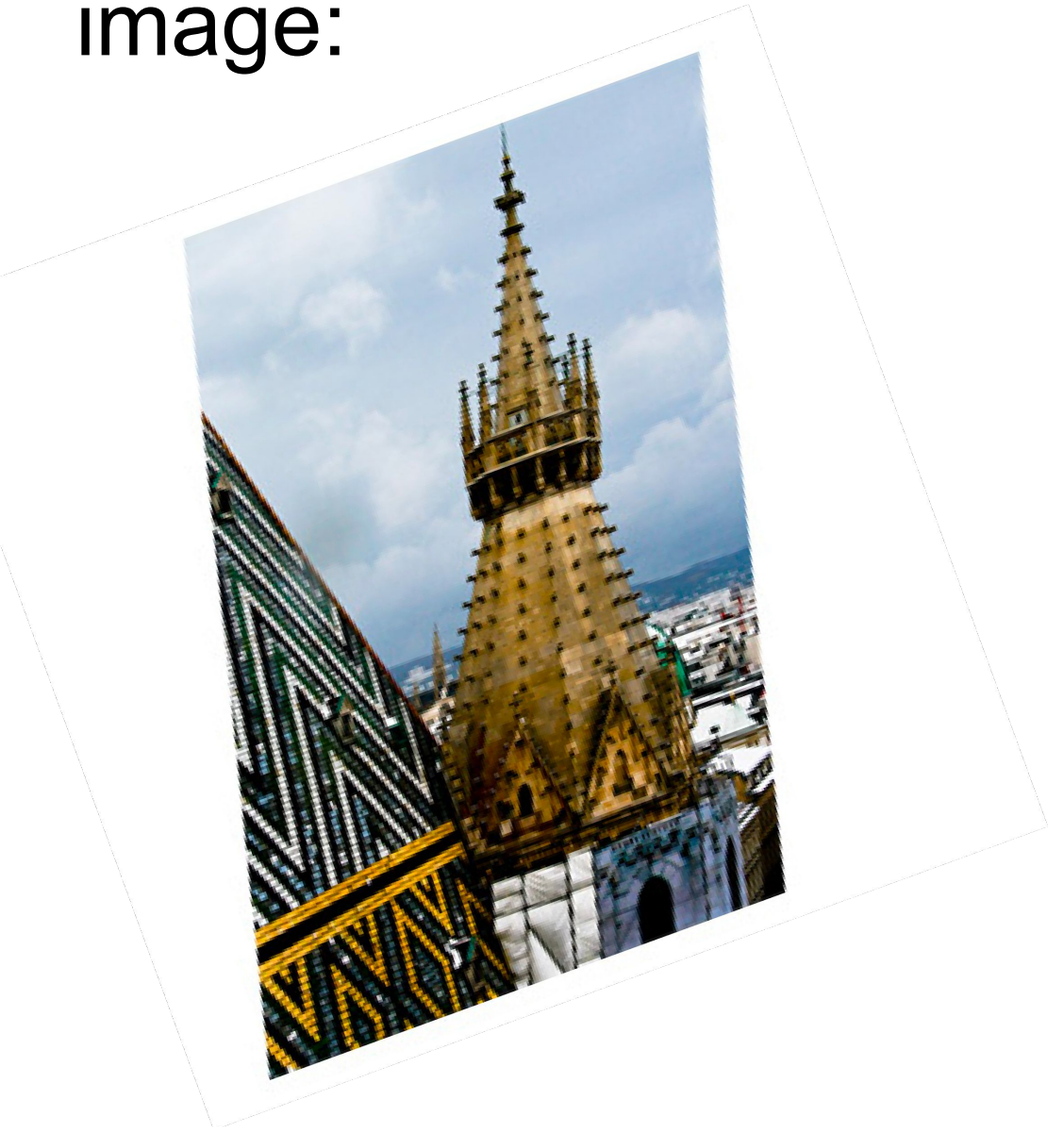
A filter:



- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

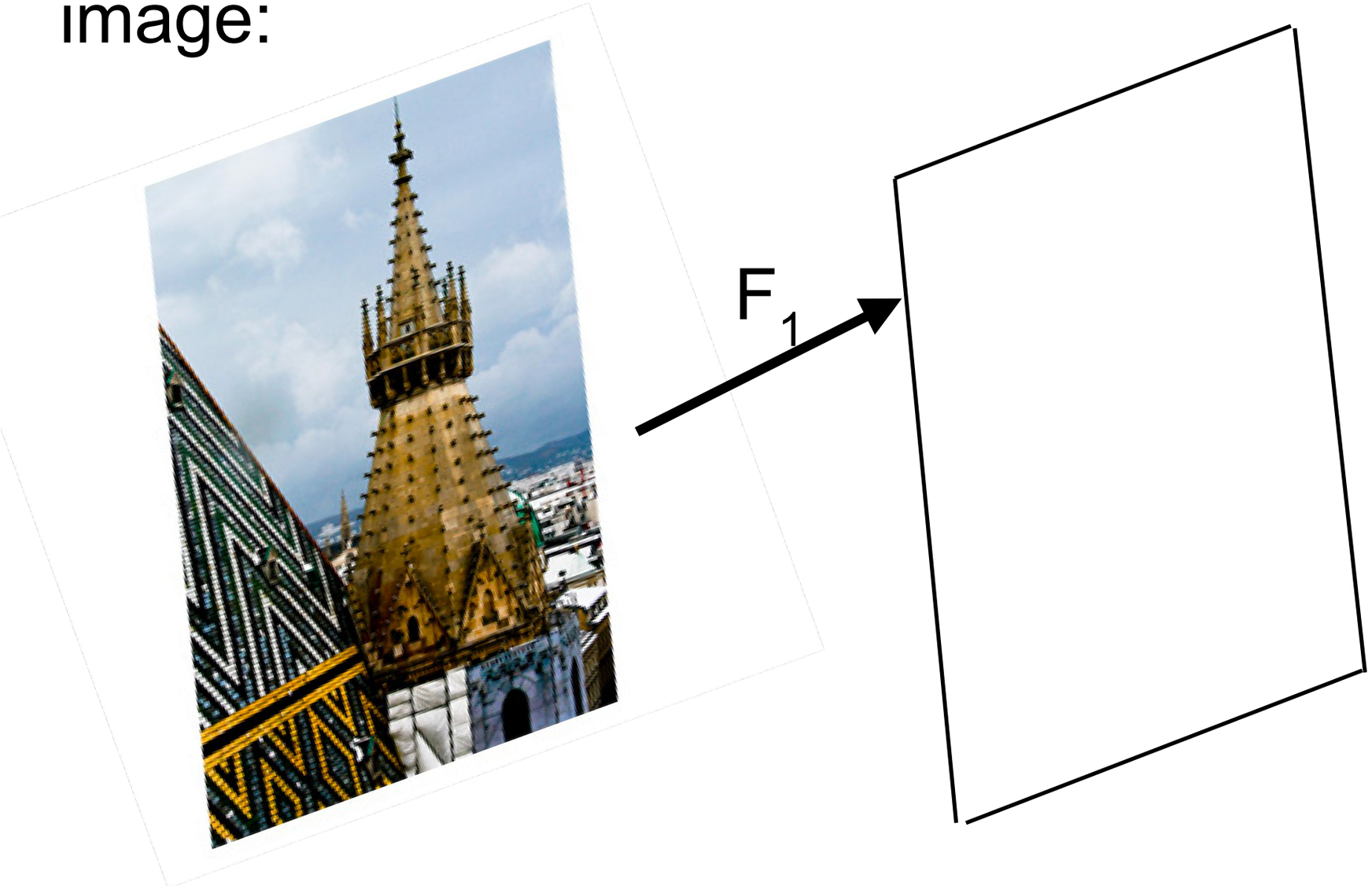
Convolutional Layer: multiple filters

An image:



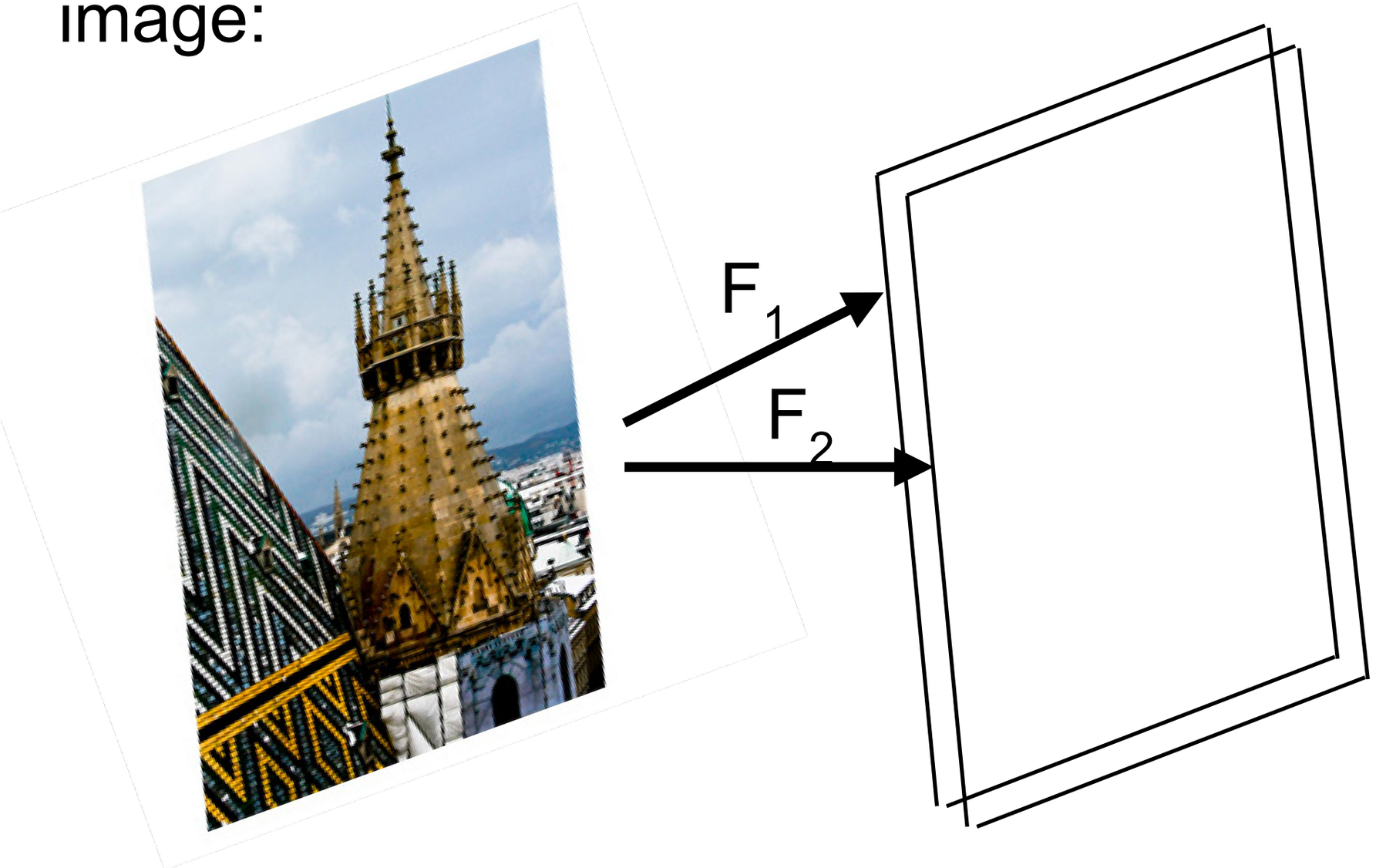
Convolutional Layer: multiple filters

An image:



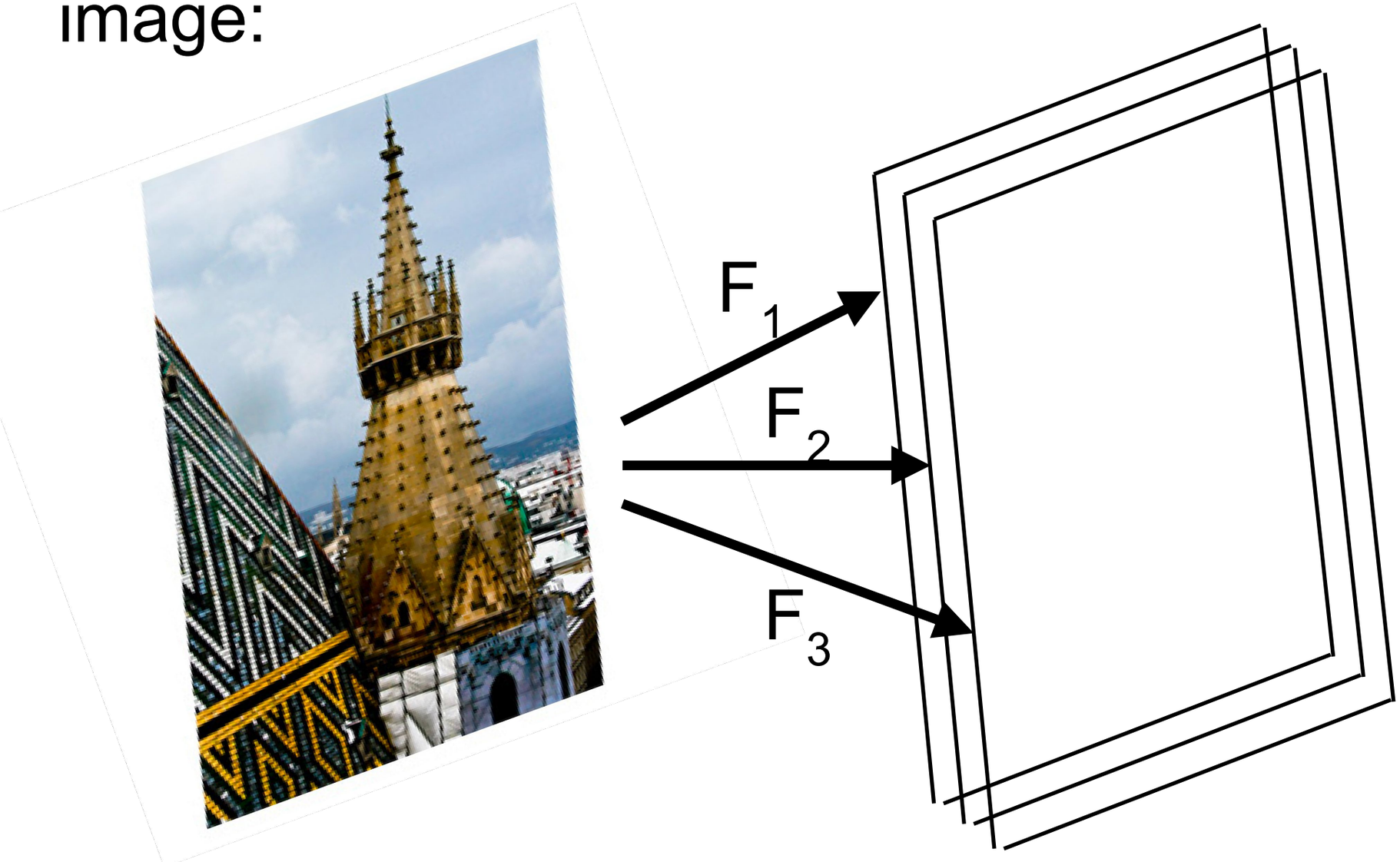
Convolutional Layer: multiple filters

An image:



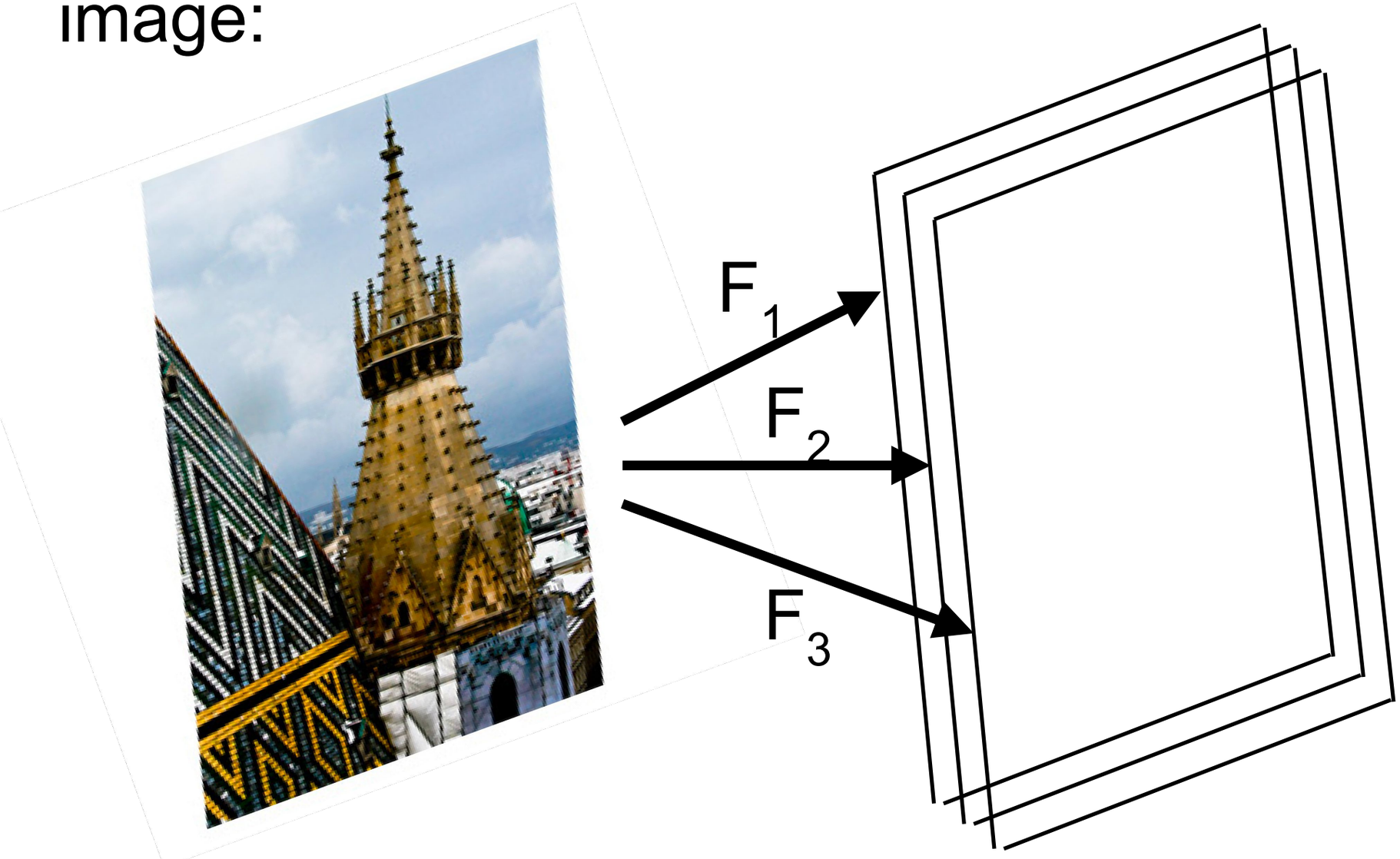
Convolutional Layer: multiple filters

An image:



Convolutional Layer: multiple filters

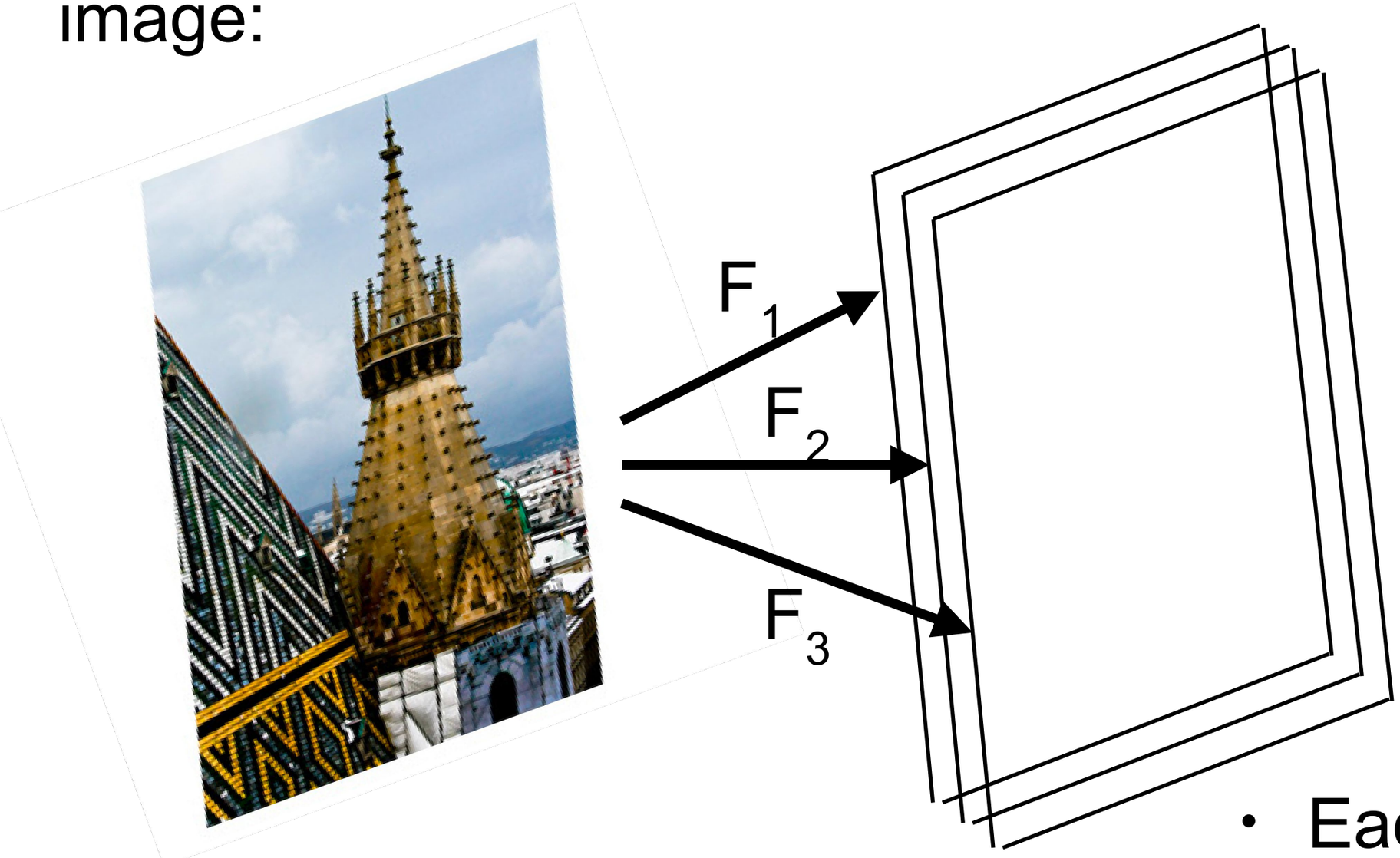
An image:



- Collection of filters in the layer: *filter bank*

Convolutional Layer: multiple filters

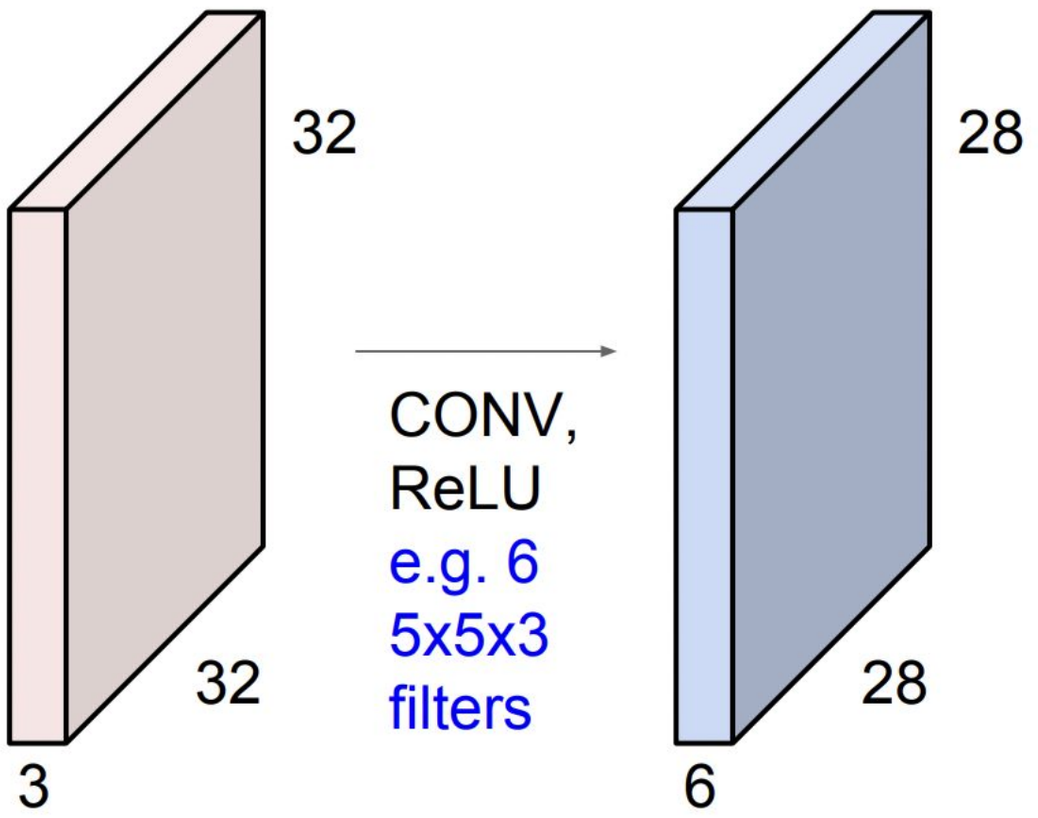
An image:



- Collection of filters in the layer: *filter bank*
- Each resulting image is a *channel*

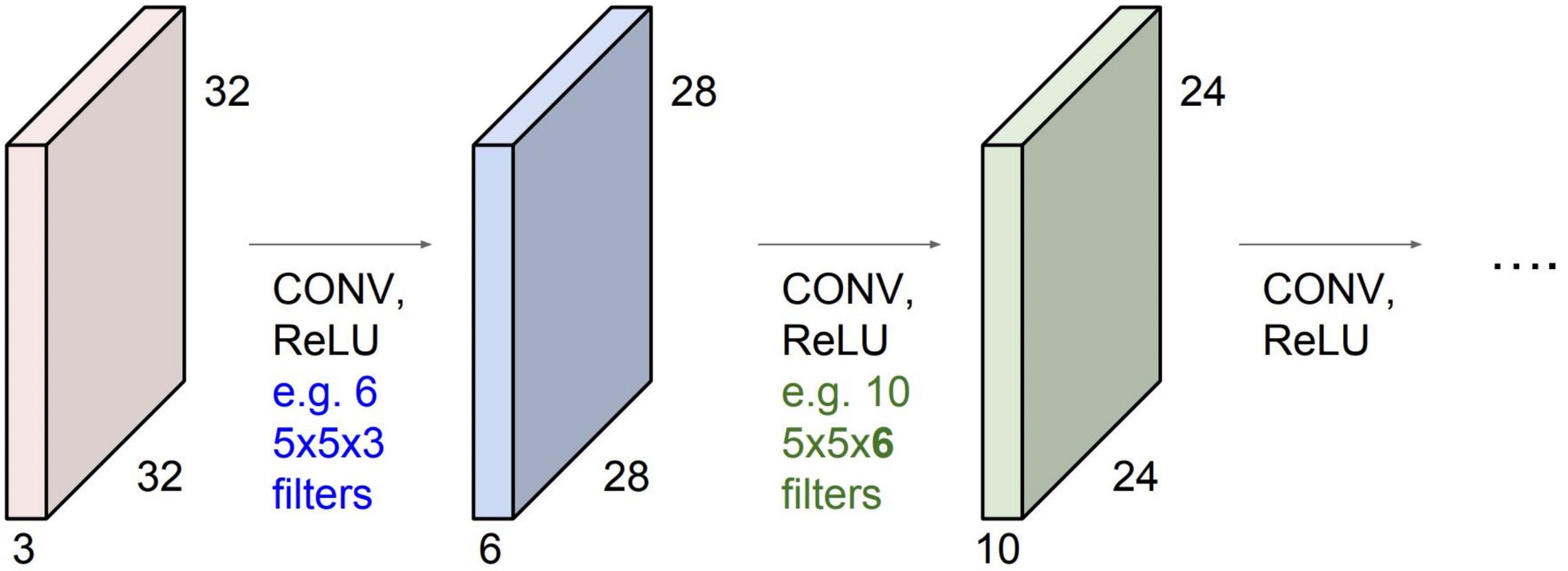
Conv Nets

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Conv Nets

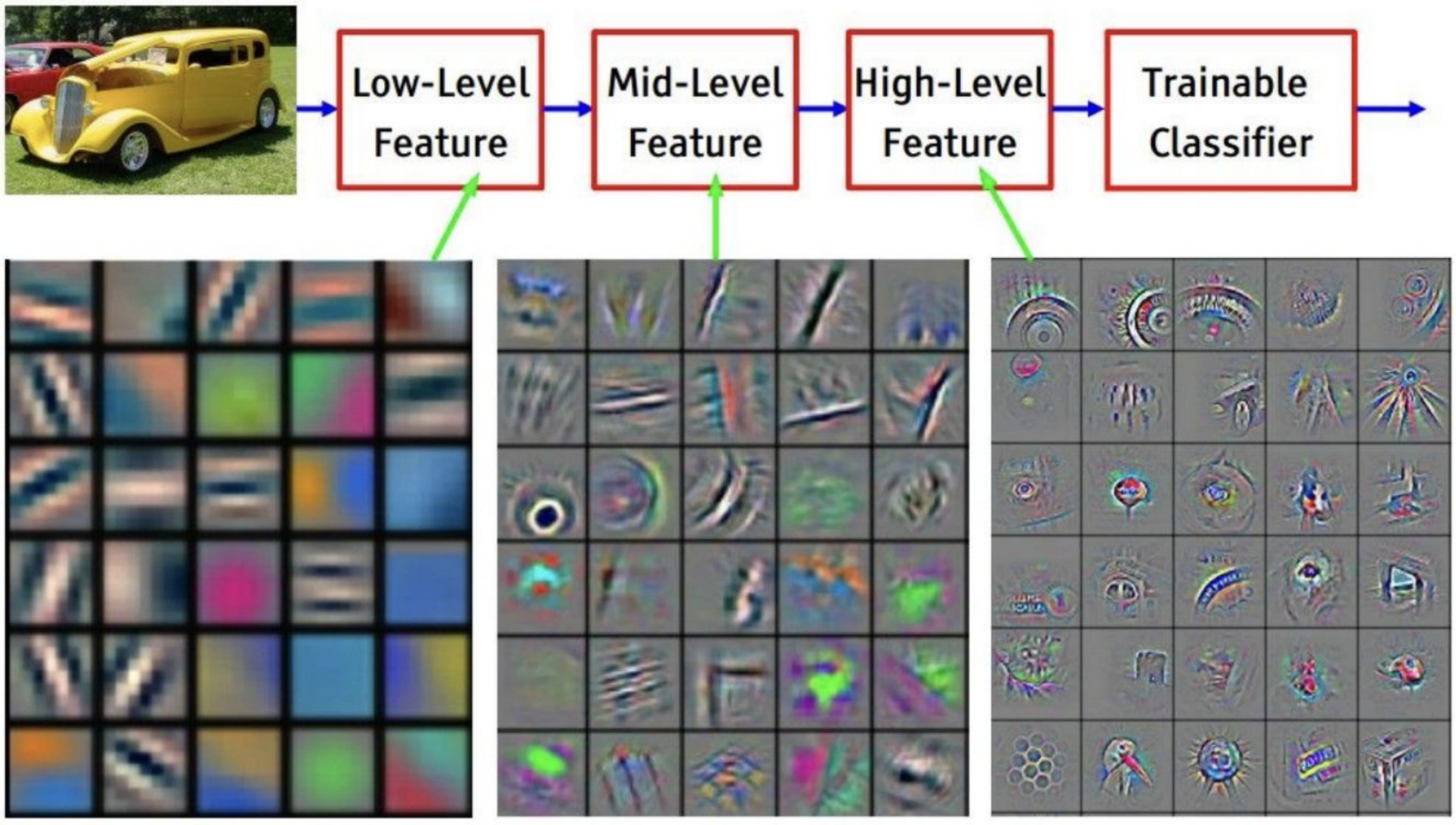
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Conv Nets Learn Hierarchy

Preview

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

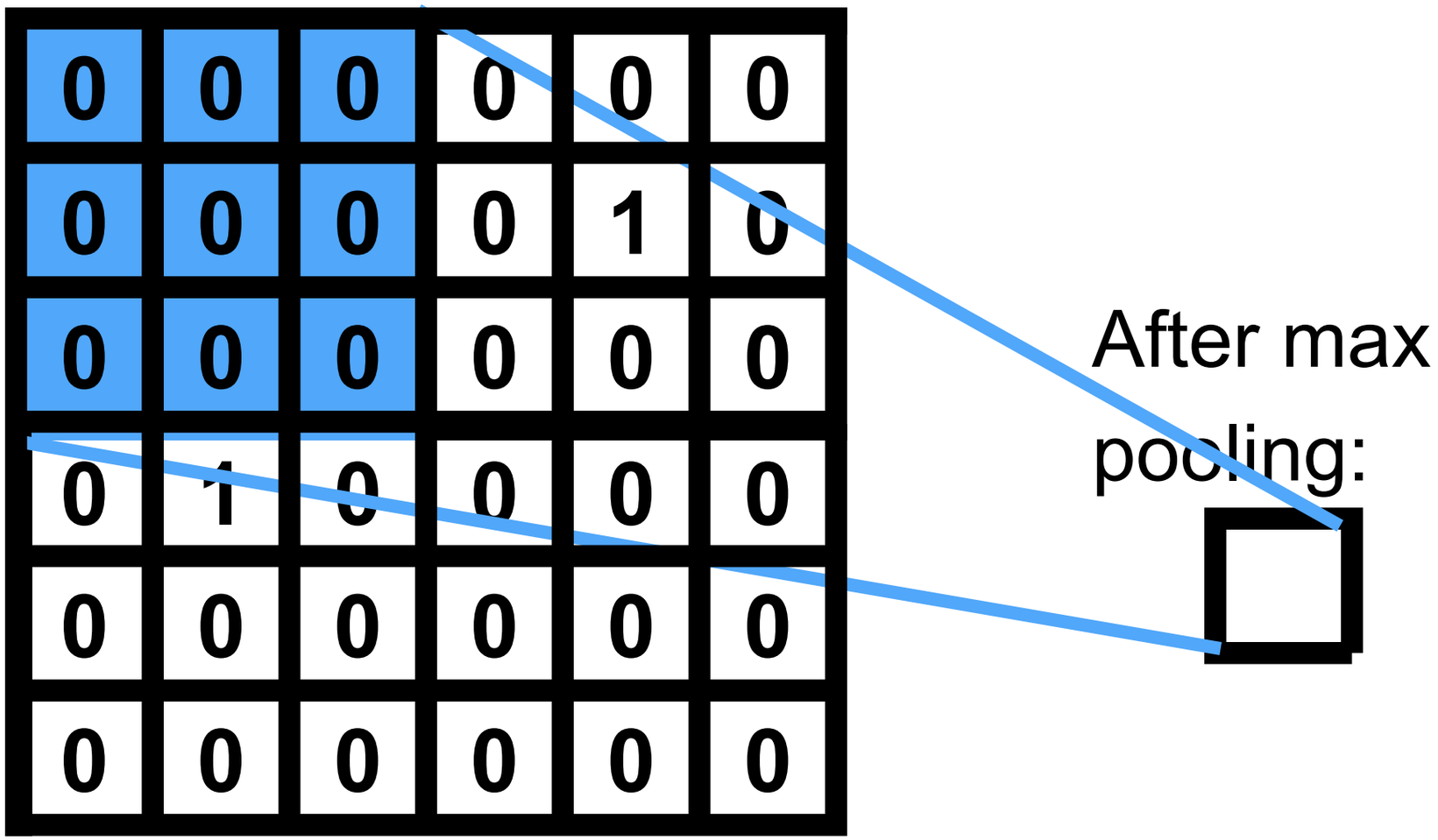
Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

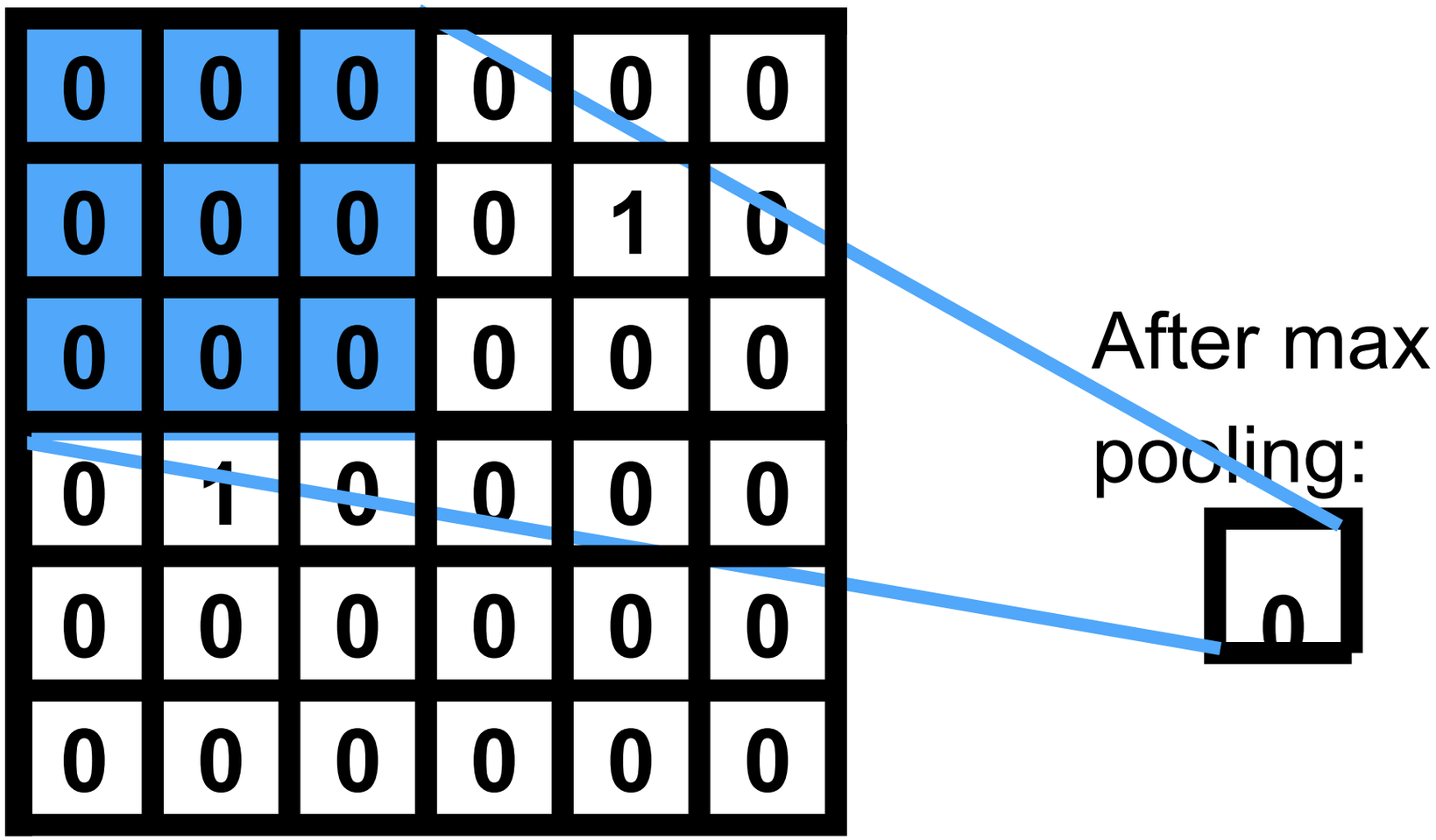


Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:



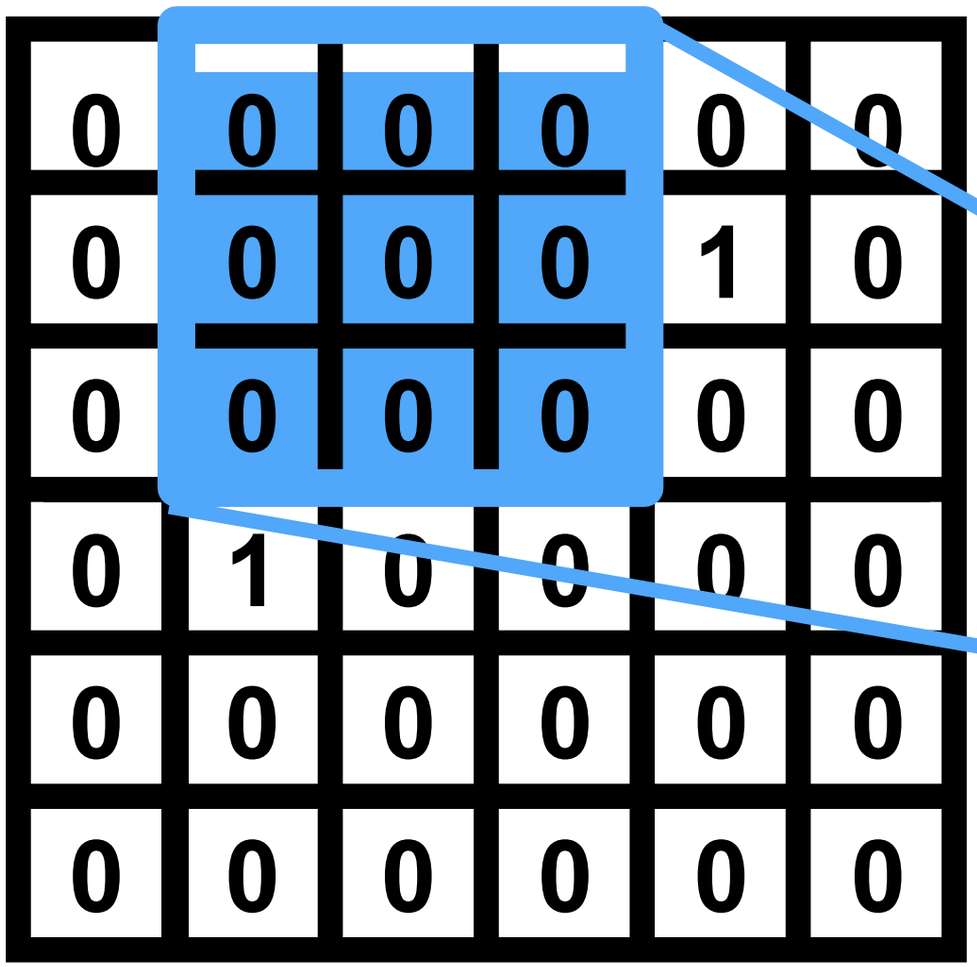
Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:



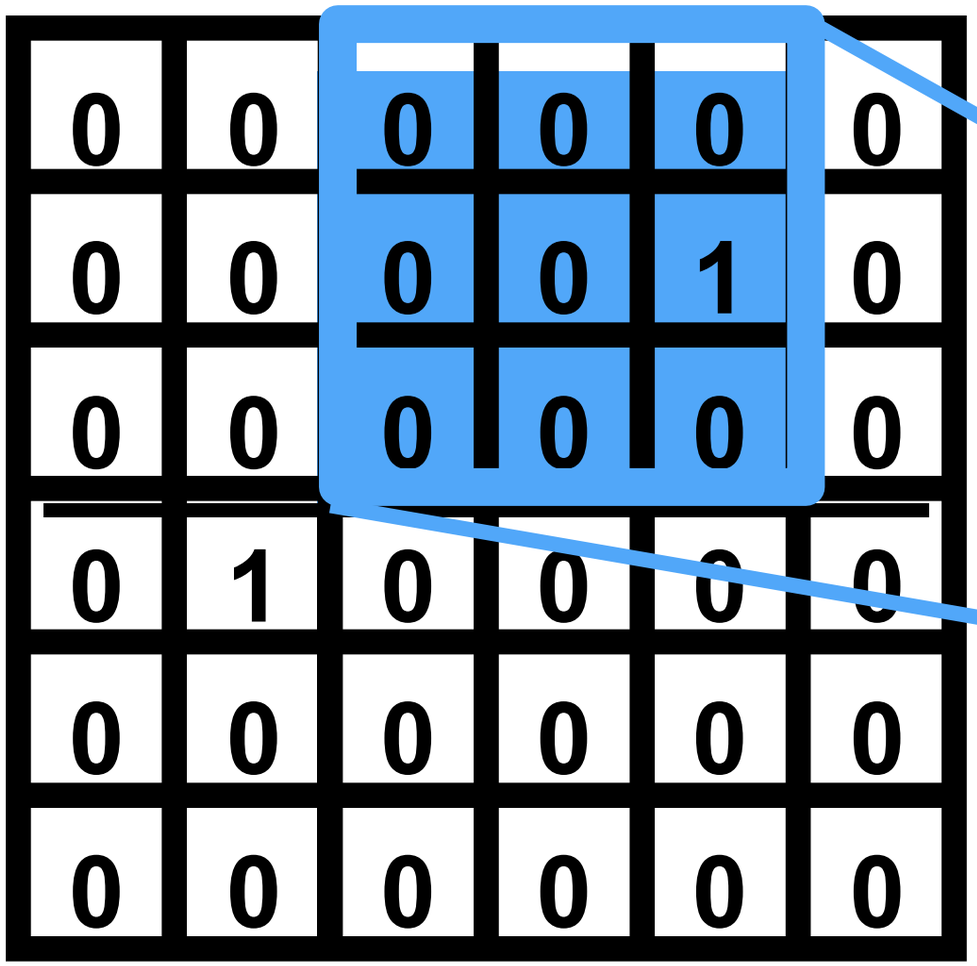
Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

Max pooling layer: 2D example

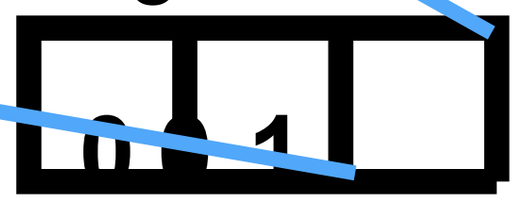
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

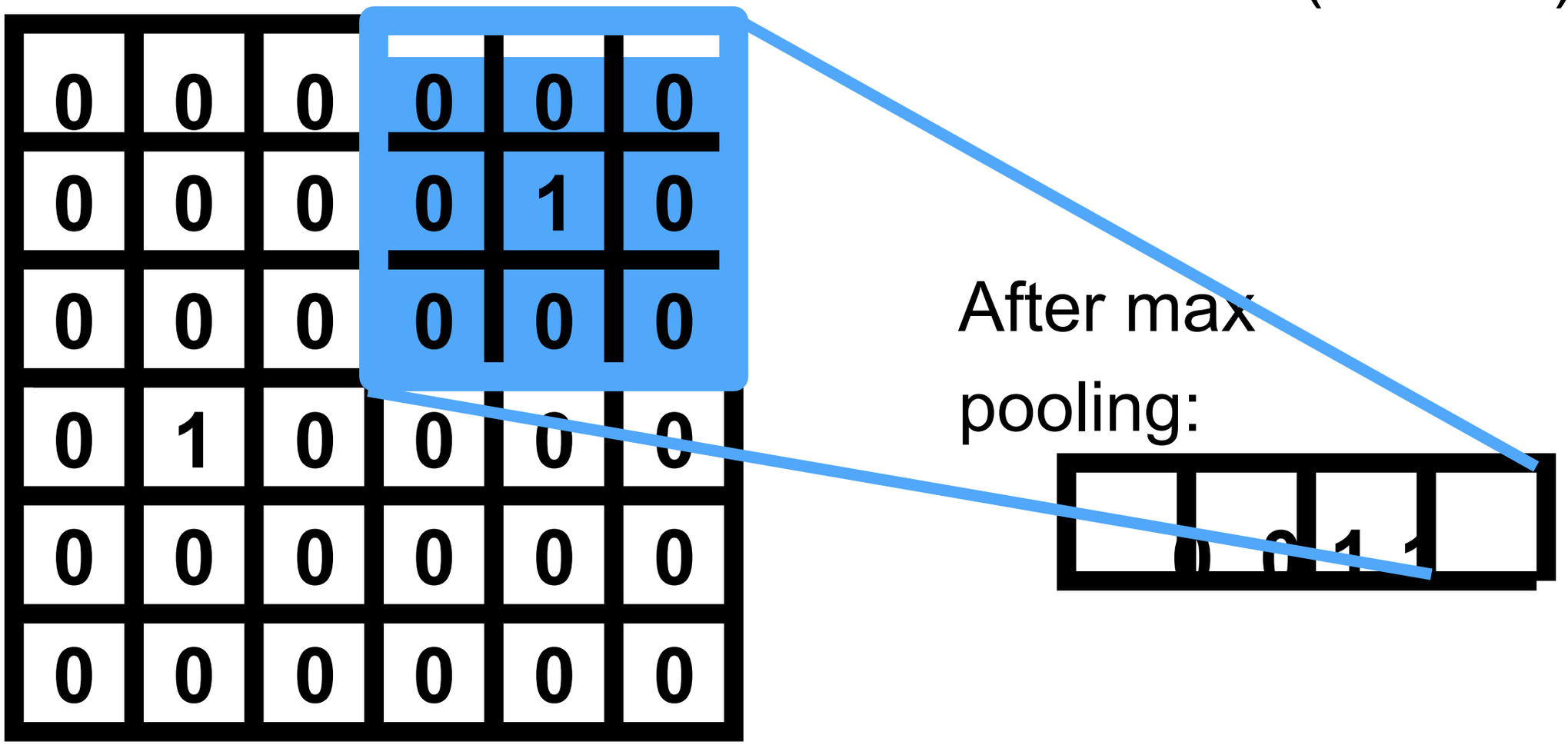
- size 3x3 (“size 3”)

After max pooling:



Max pooling layer: 2D example

Output from the convolutional layer & ReLU:



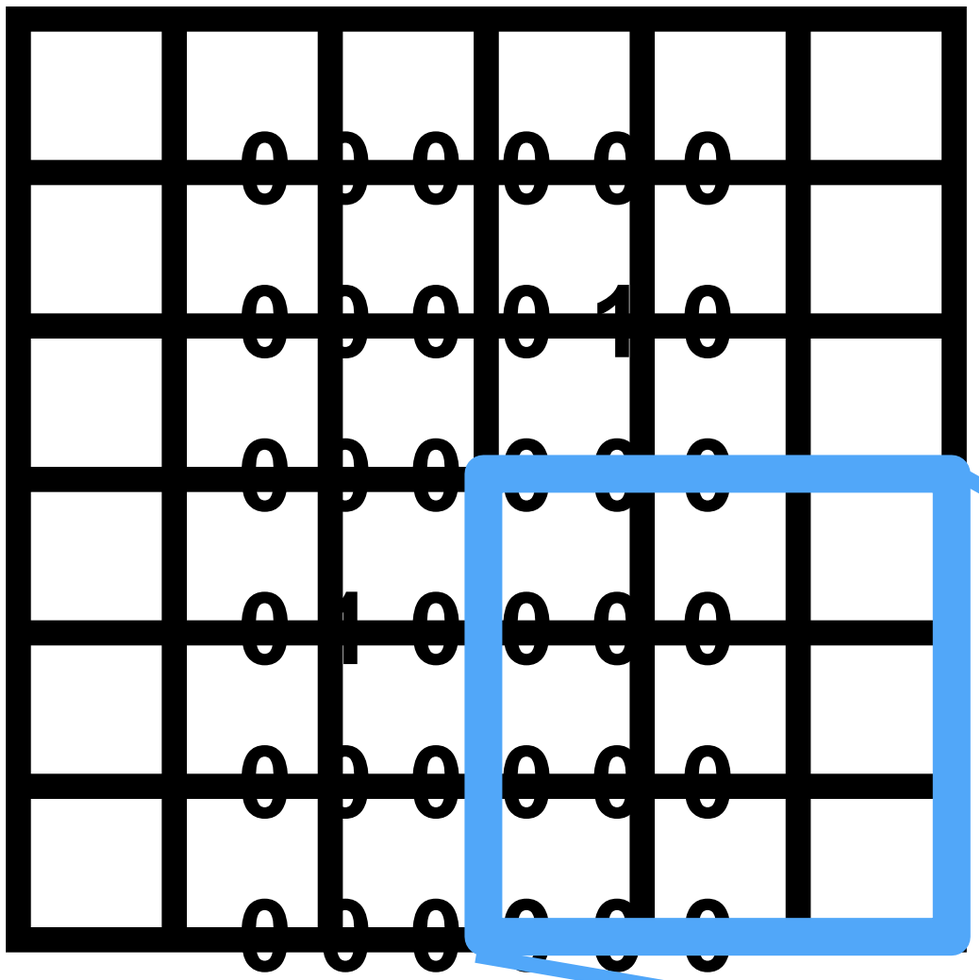
Max pooling: returns max of its arguments

- size 3x3 (“size 3”)

After max pooling:

Max pooling layer: 2D example

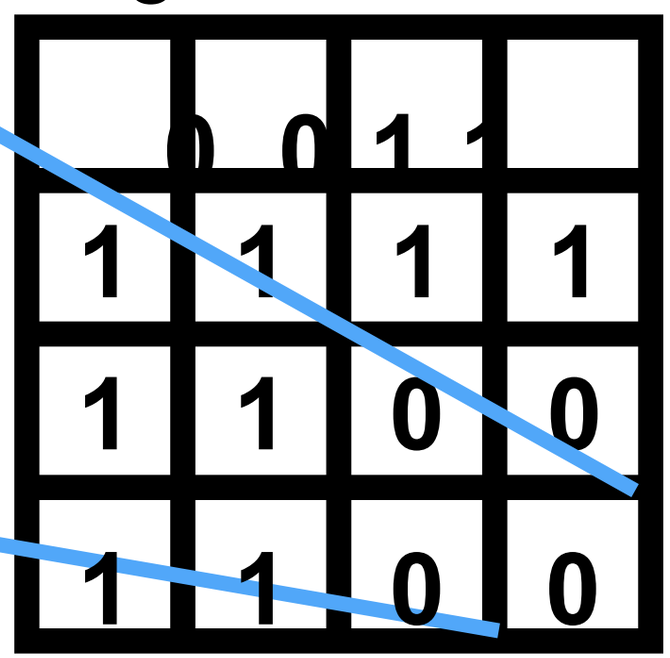
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

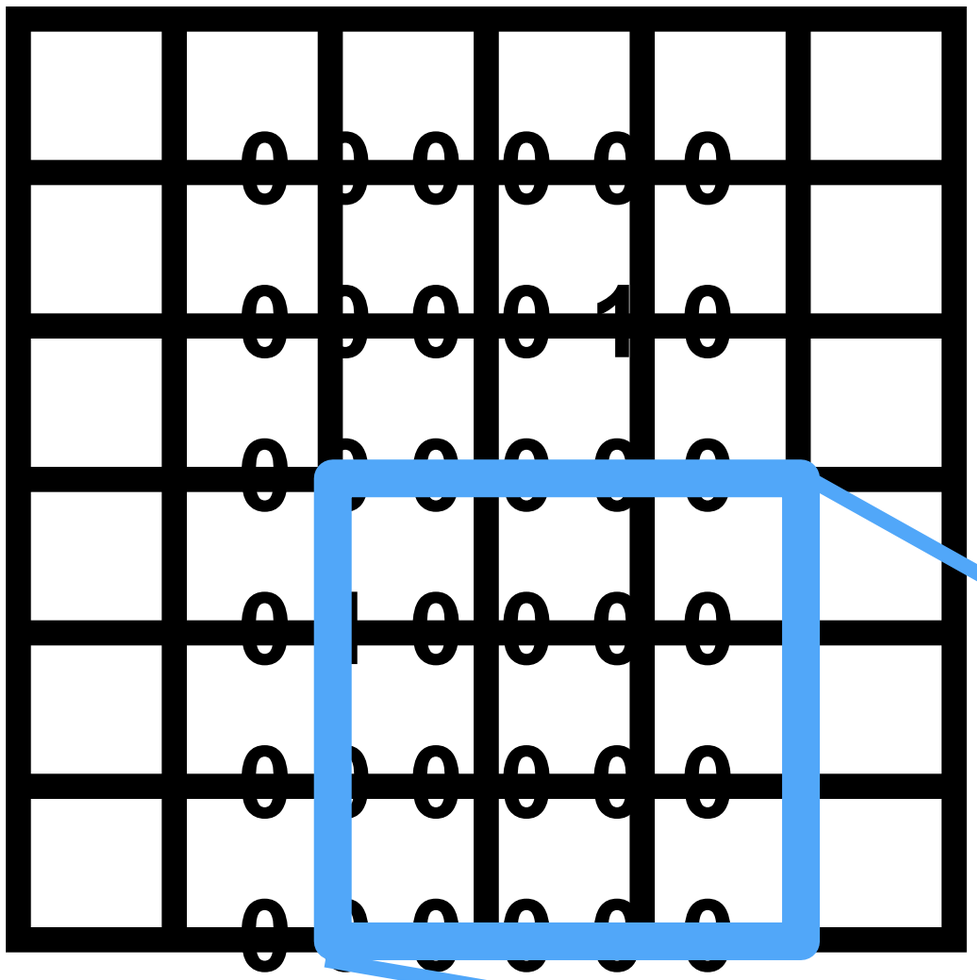
- size 3x3 (“size 3”)

After max pooling:



Max pooling layer: 2D example

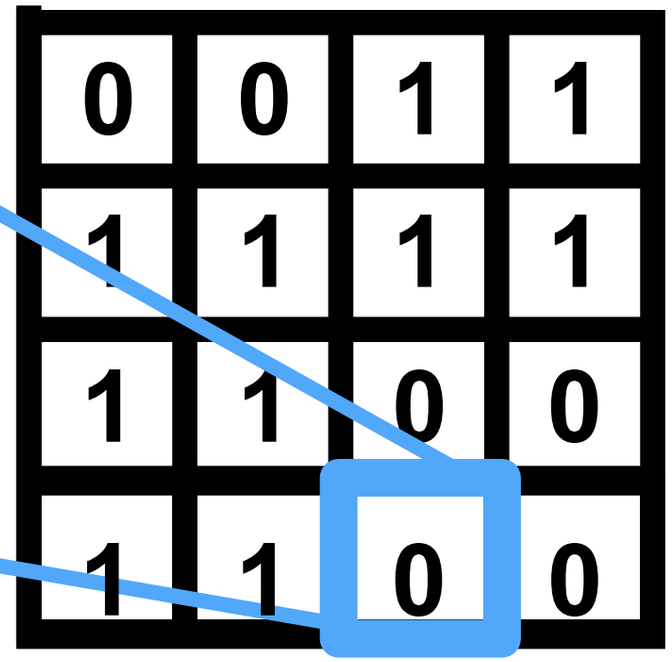
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

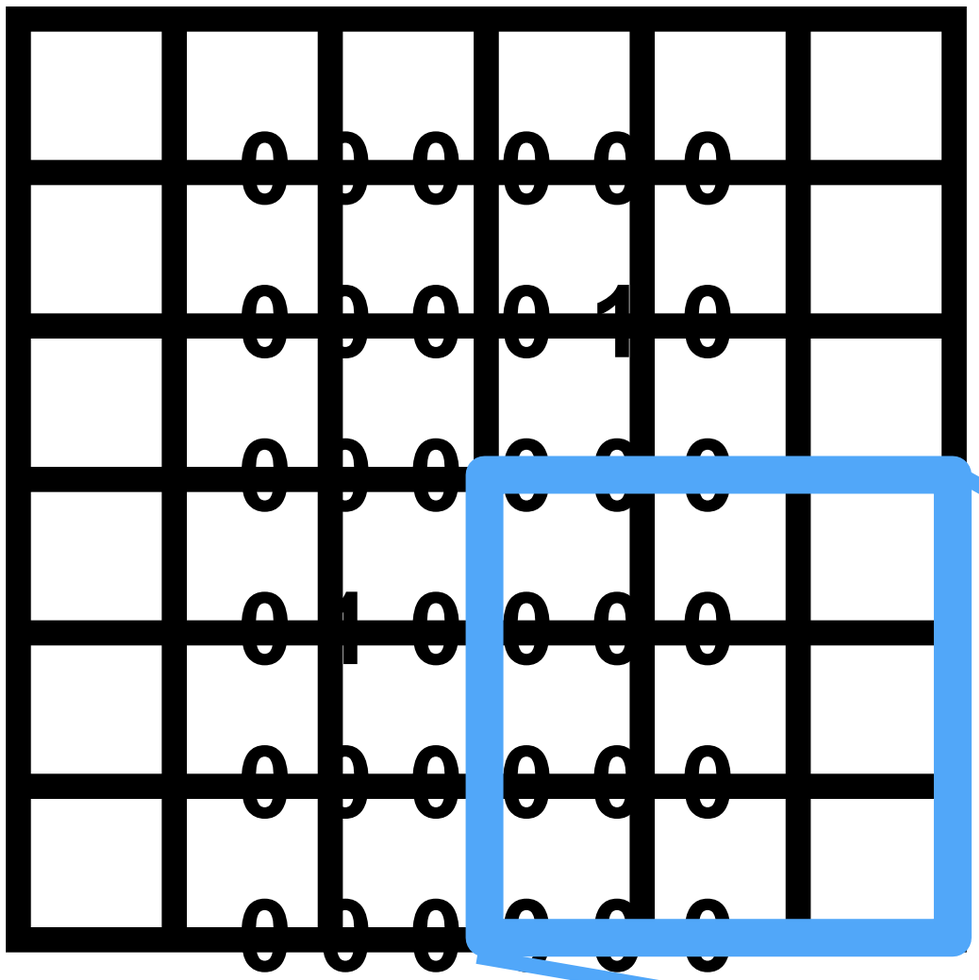
- size 3x3 (“size 3”)

After max pooling:



Max pooling layer: 2D example

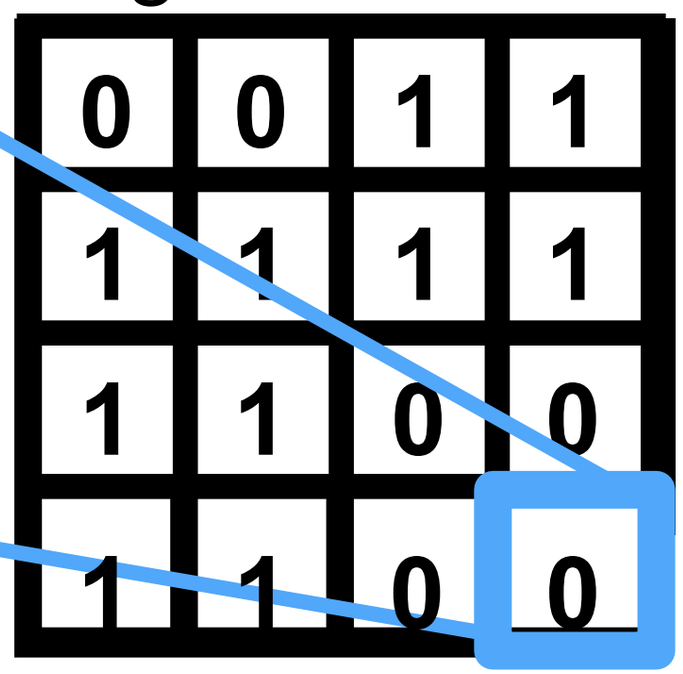
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

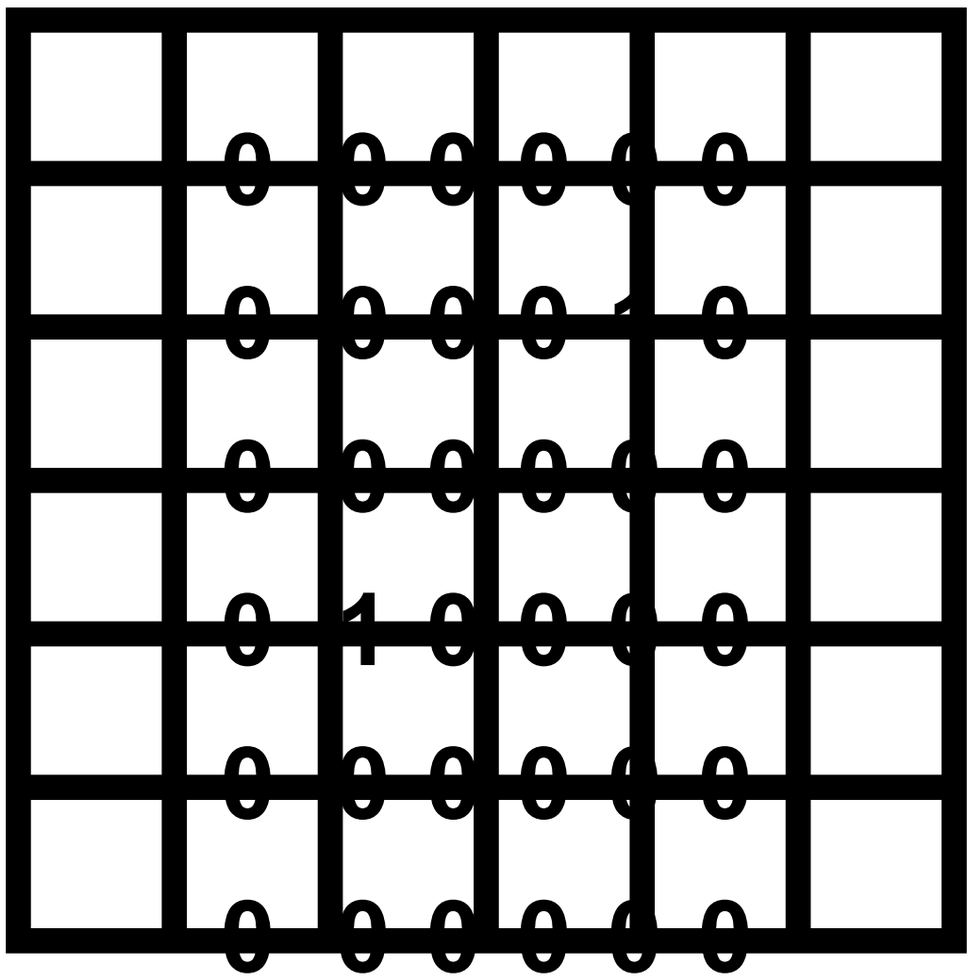
- size 3x3 (“size 3”)

After max pooling:



Max pooling layer: 2D example

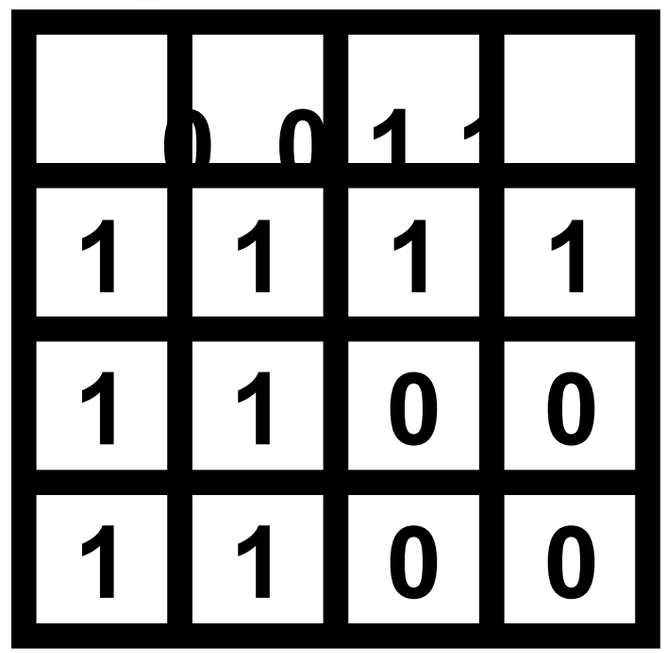
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

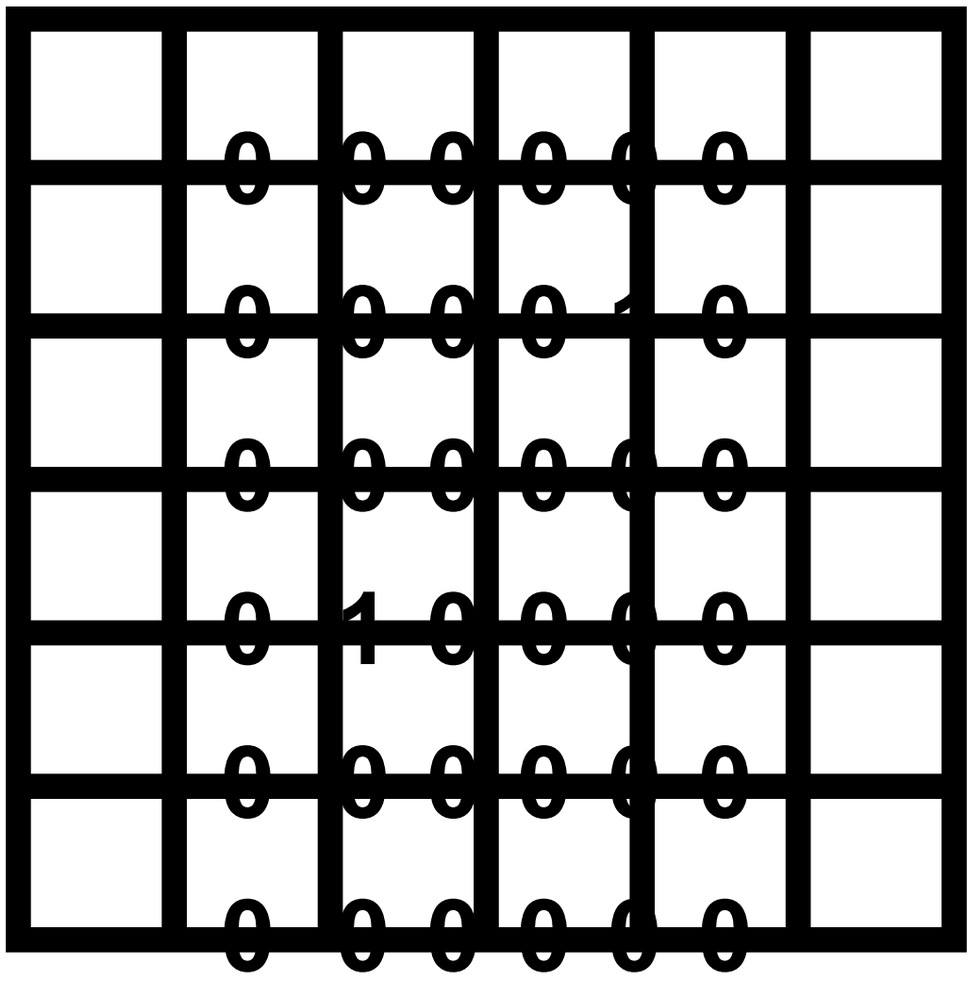
- size 3x3 (“size 3”)

After max pooling:



Max pooling layer: 2D example

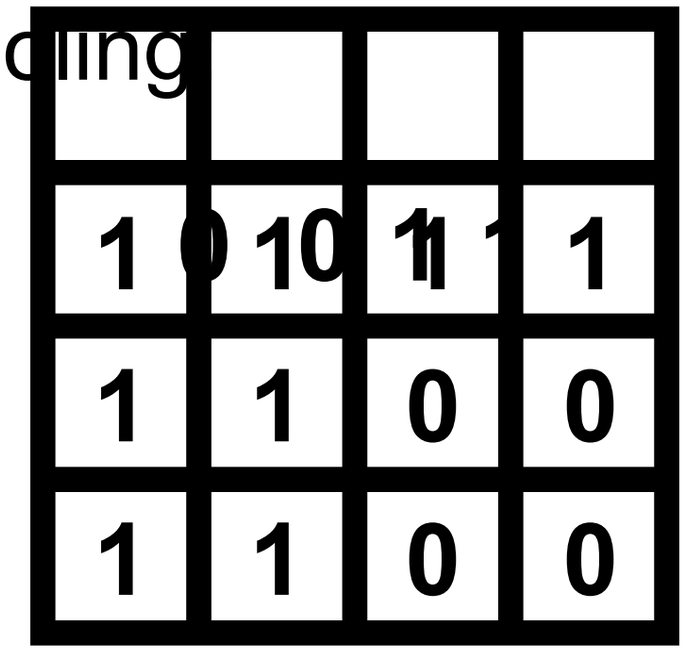
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 1

After max pooling



Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

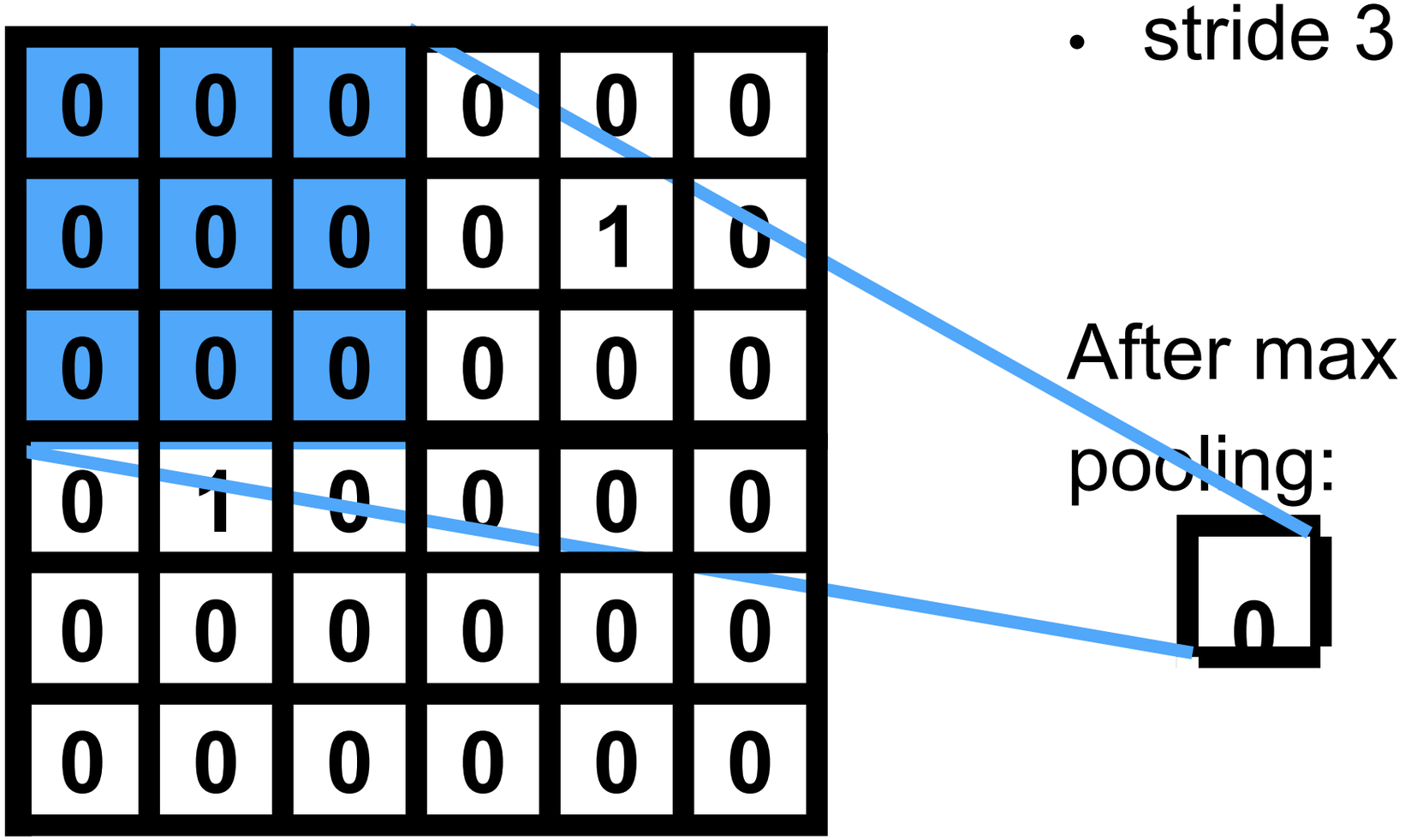
After max pooling

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:



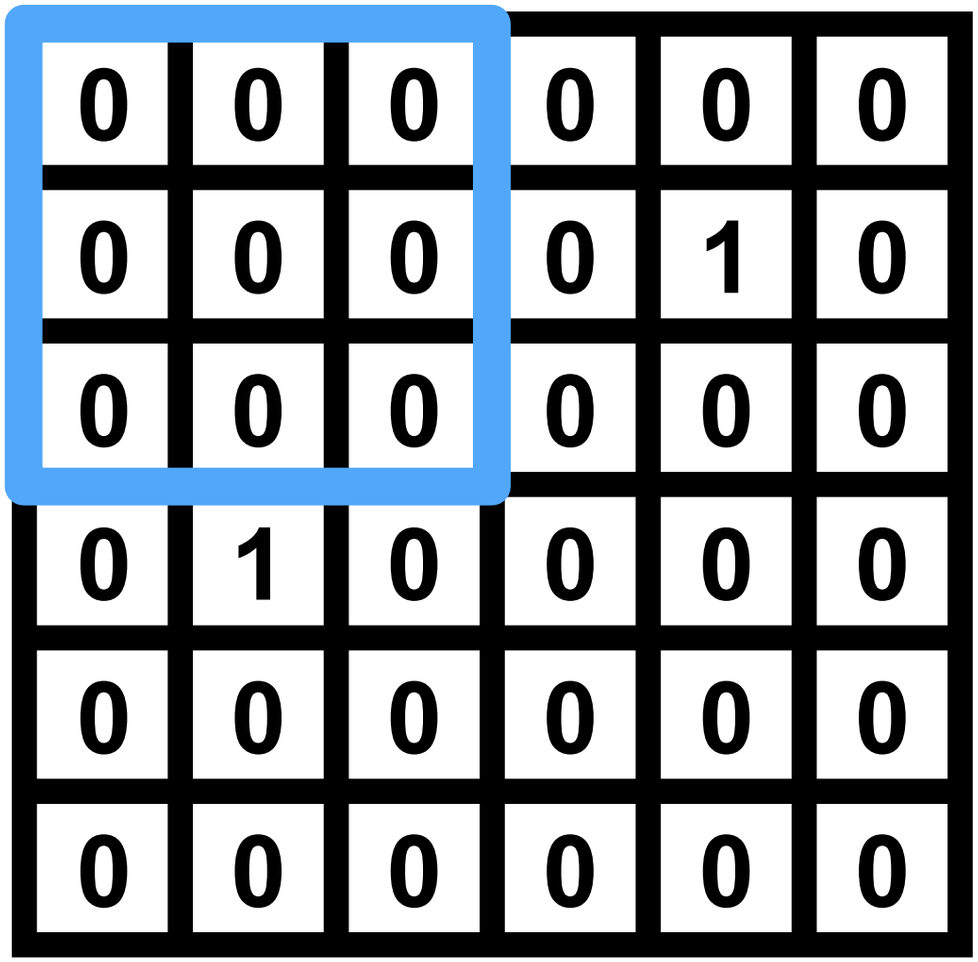
Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

Max pooling layer: 2D example

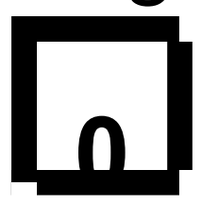
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

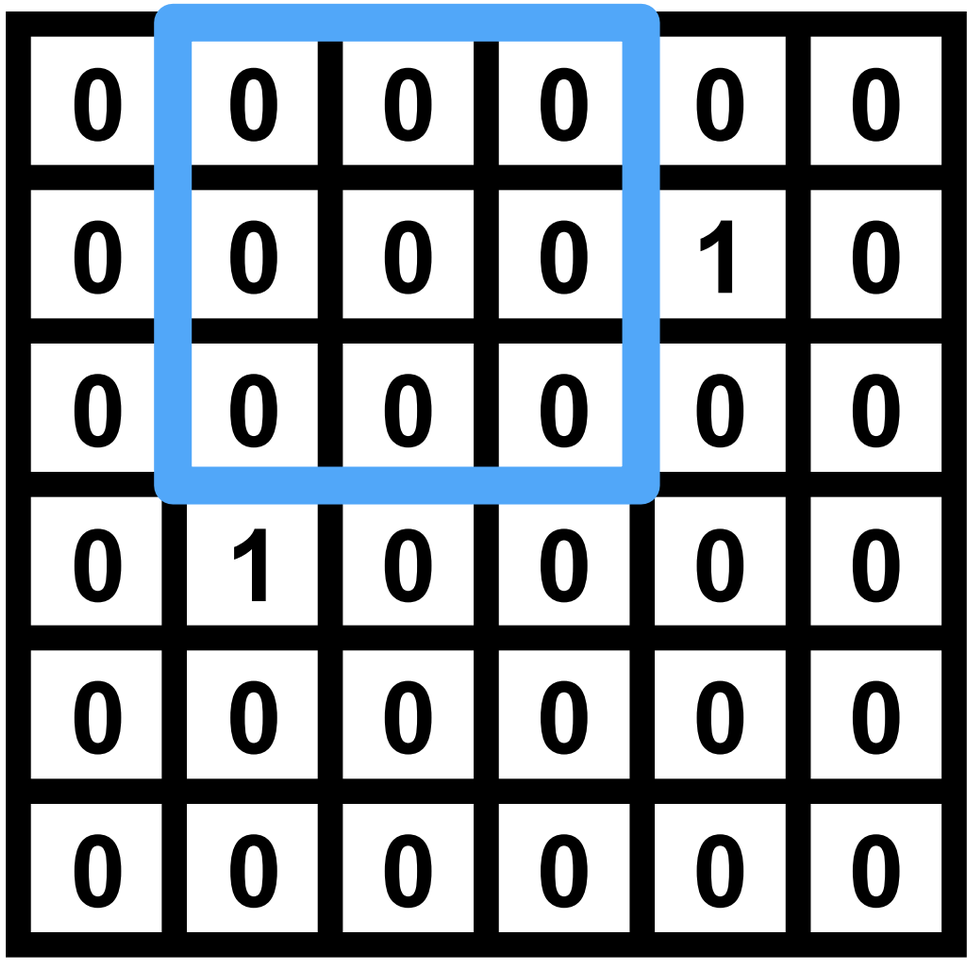
- size 3x3 (“size 3”)
- stride 3

After max pooling:



Max pooling layer: 2D example

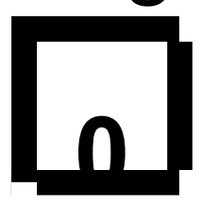
Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:



Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1

Max pooling layer: 2D example

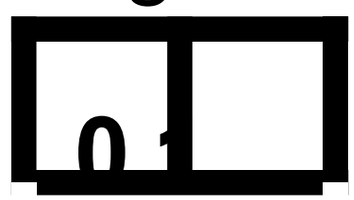
Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:



Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

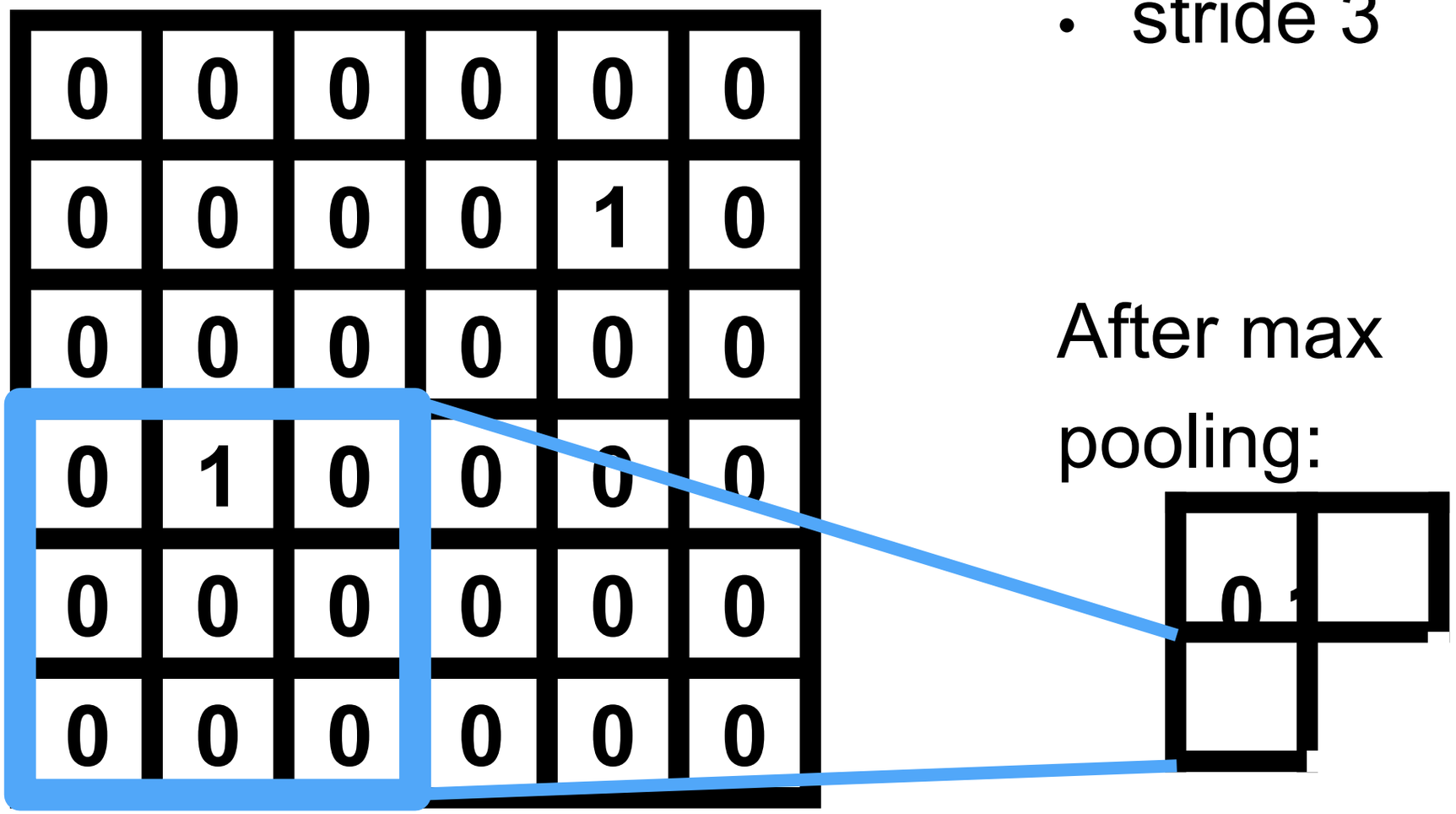
- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:



Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	0

Max pooling layer: 2D example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

After max pooling:

0	1
1	0

- Can use stride with filters too
- No weights in max pooling

Convolution (General Formulae)

Summary. To summarize, the Conv Layer:

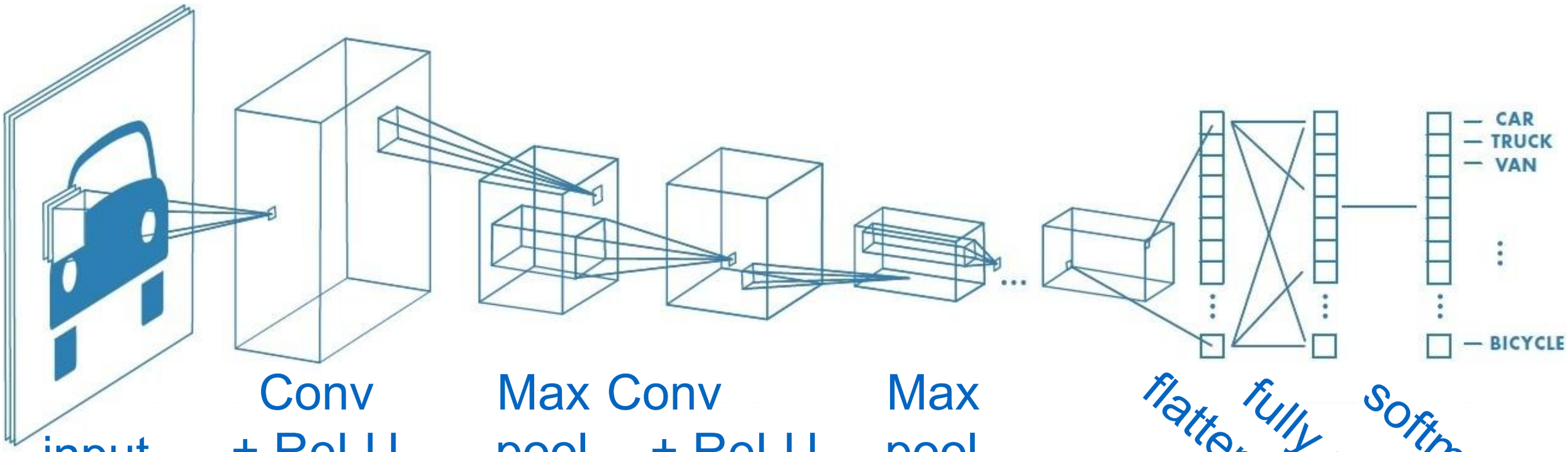
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

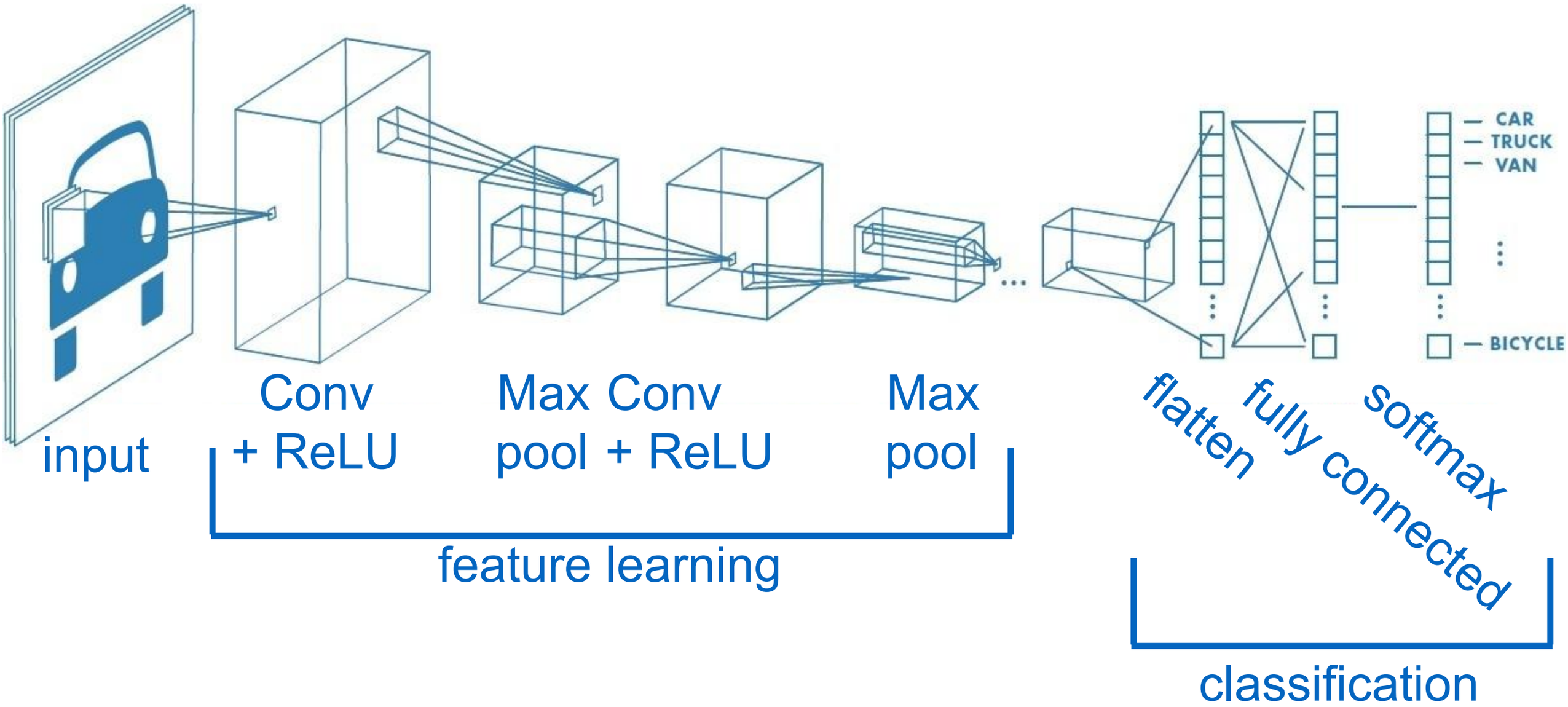
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

CNNs: typical architecture



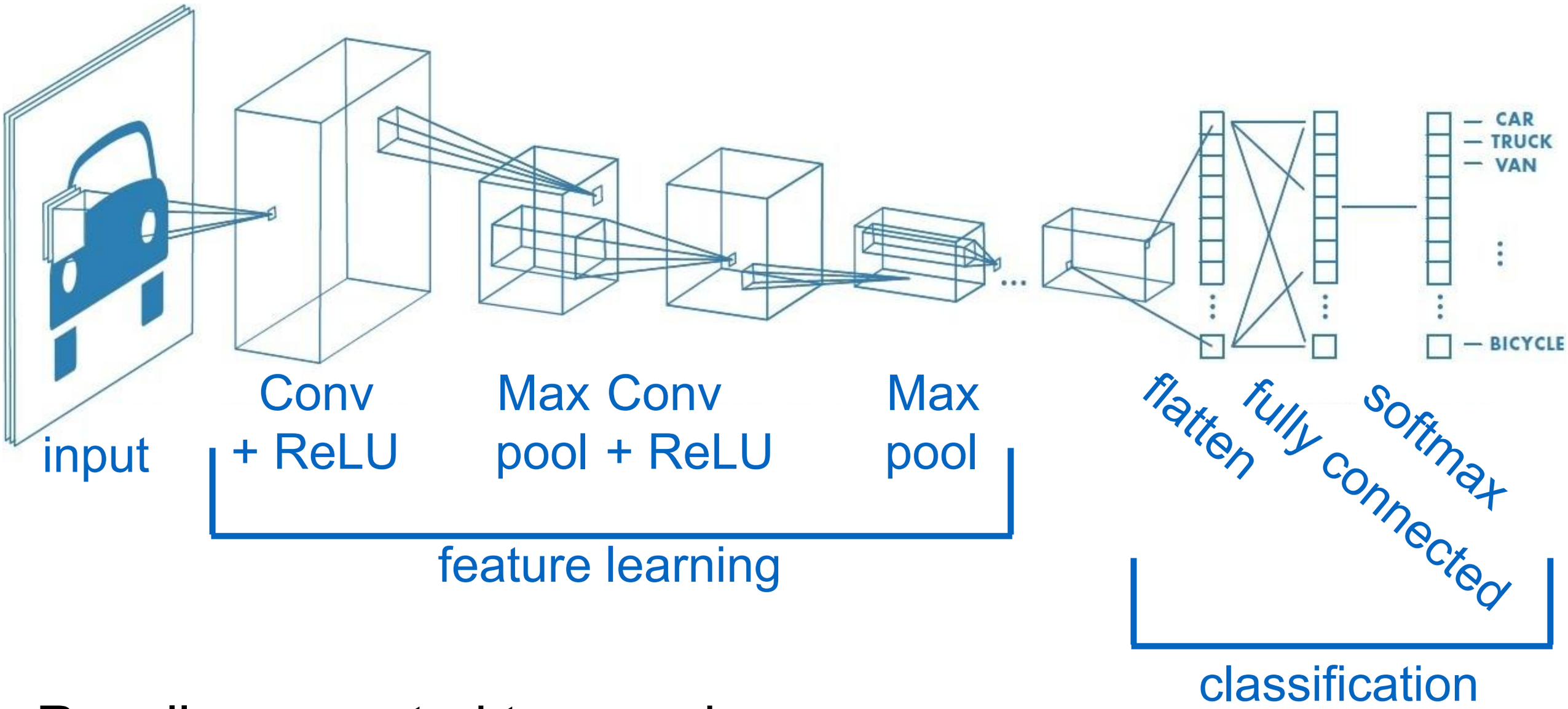
[https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html]

CNNs: typical architecture



[https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html]

CNNs: typical architecture

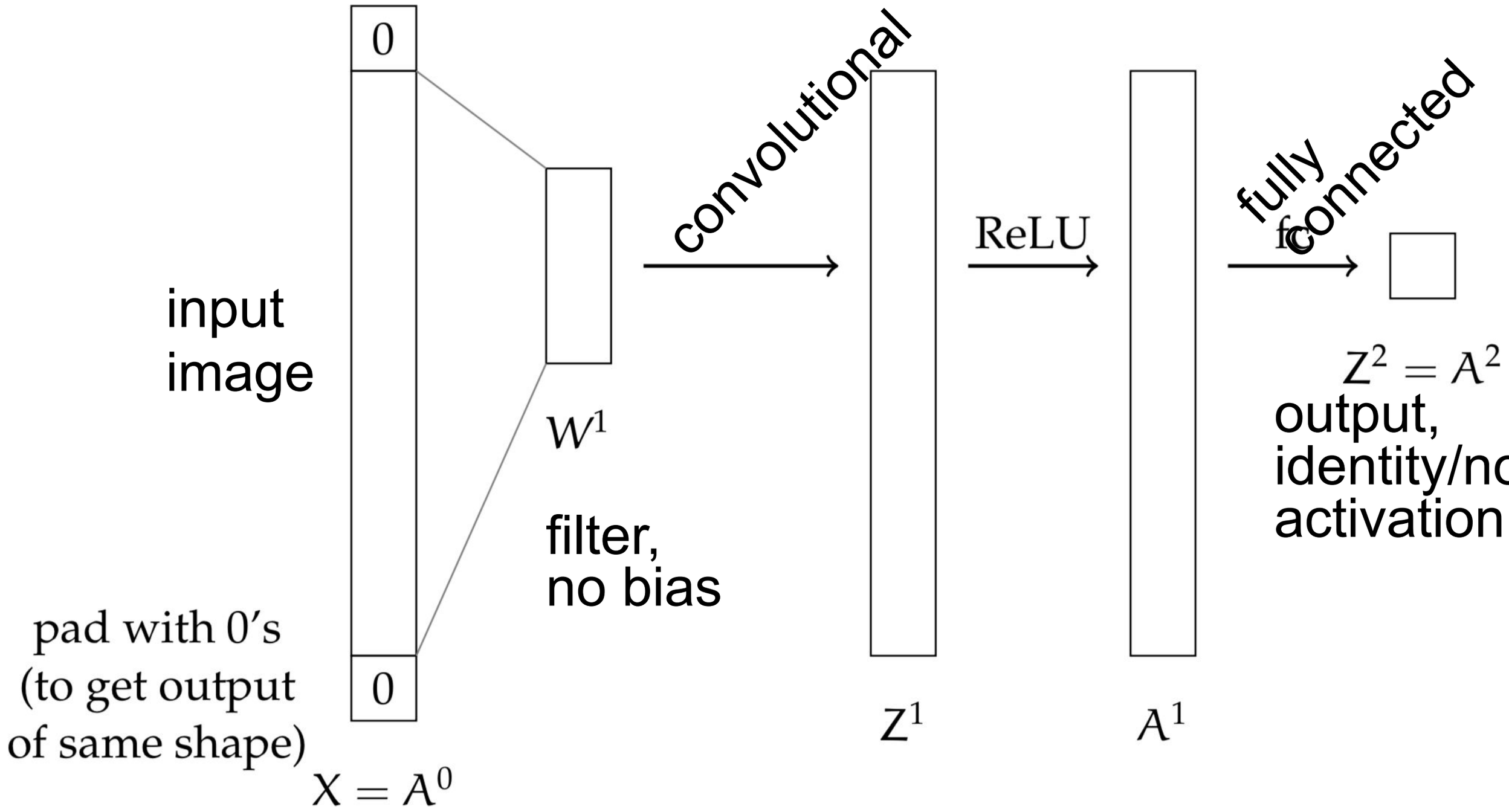


Recall: we wanted to encode

- Spatial locality
- Translation invariance

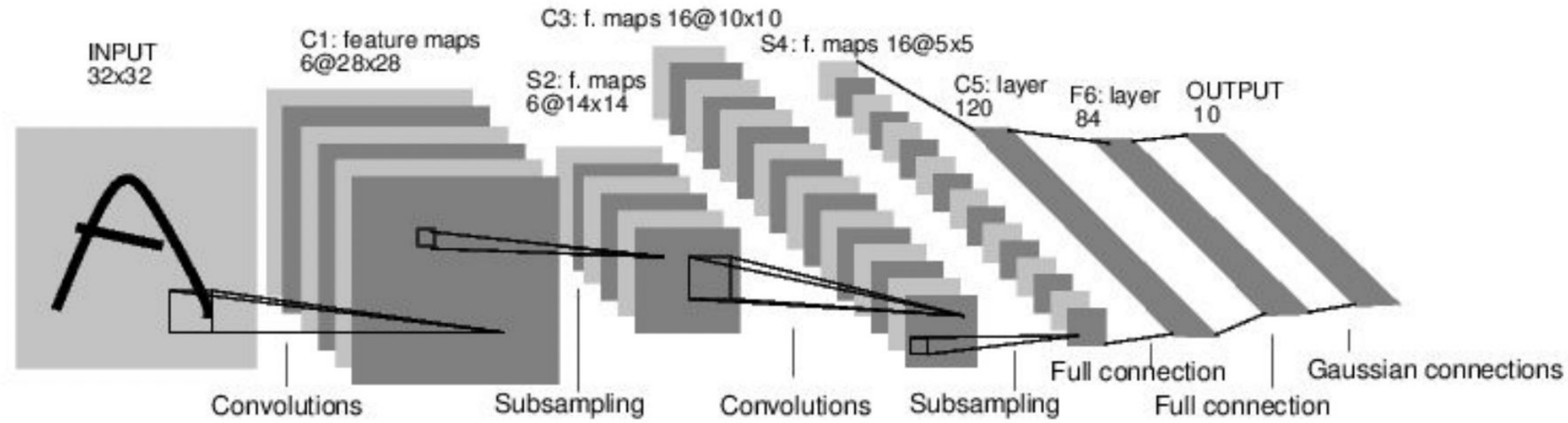
[https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html]

CNNs: a taste of backpropagation



Case Study: LeNet 5

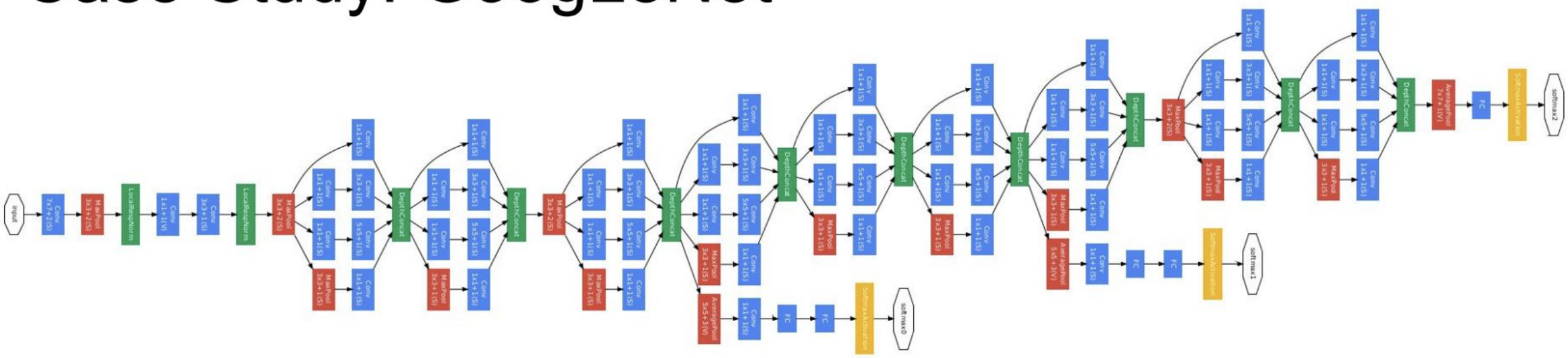
[LeCun et al., 1998]

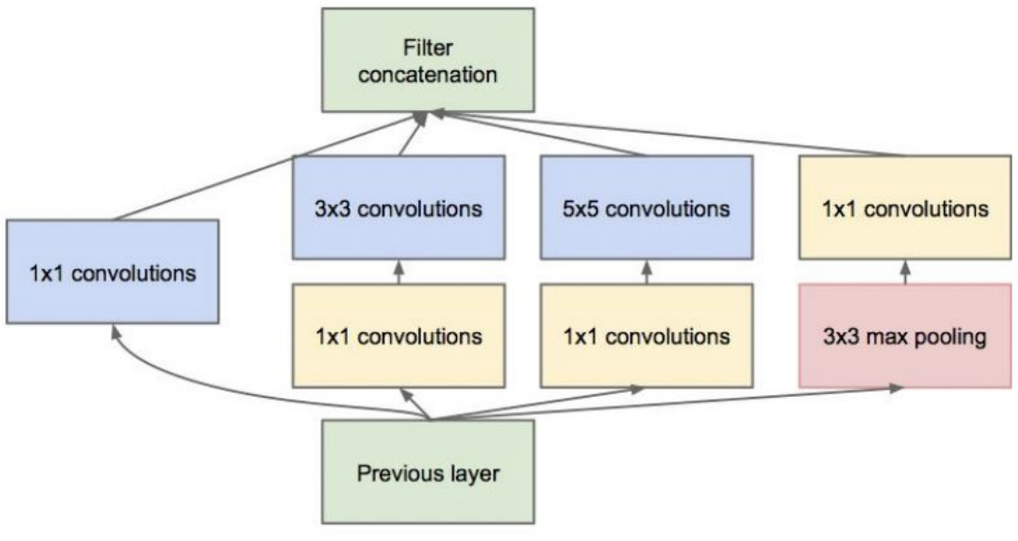


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: GoogleNet

Case Study: GoogLeNet *[Szegedy et al., 2014]*





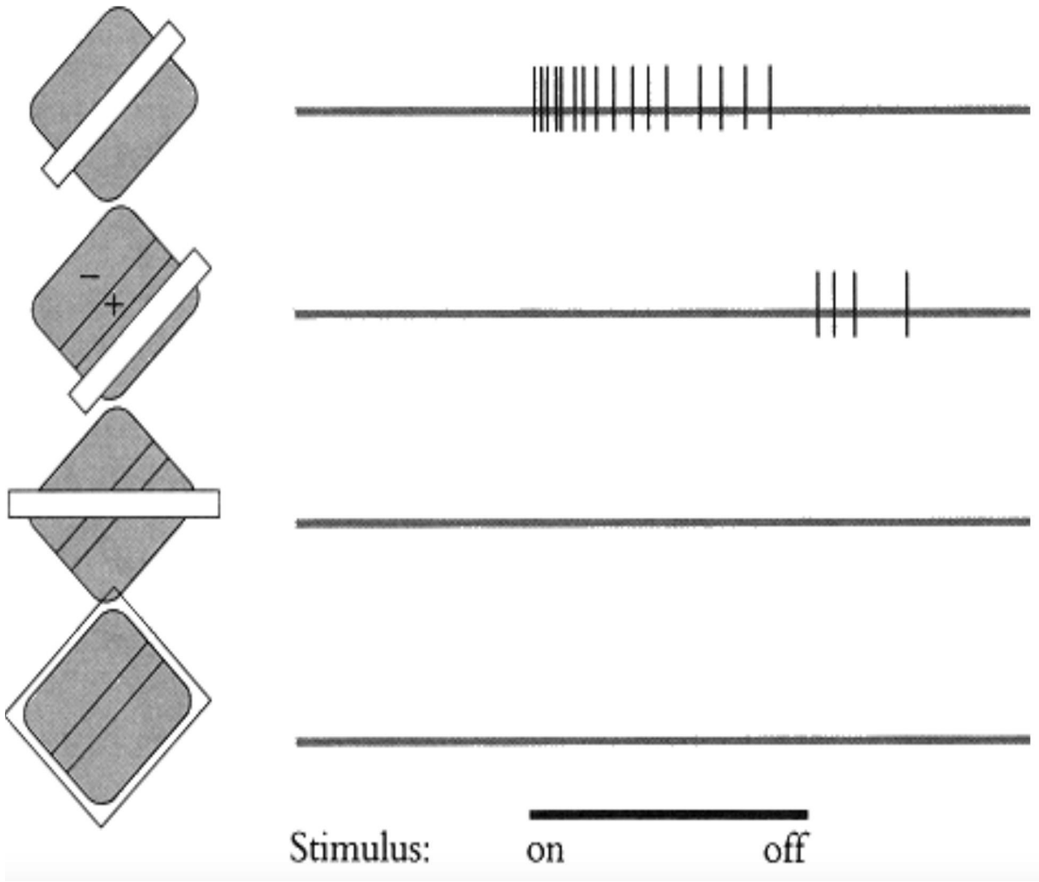
Inception module

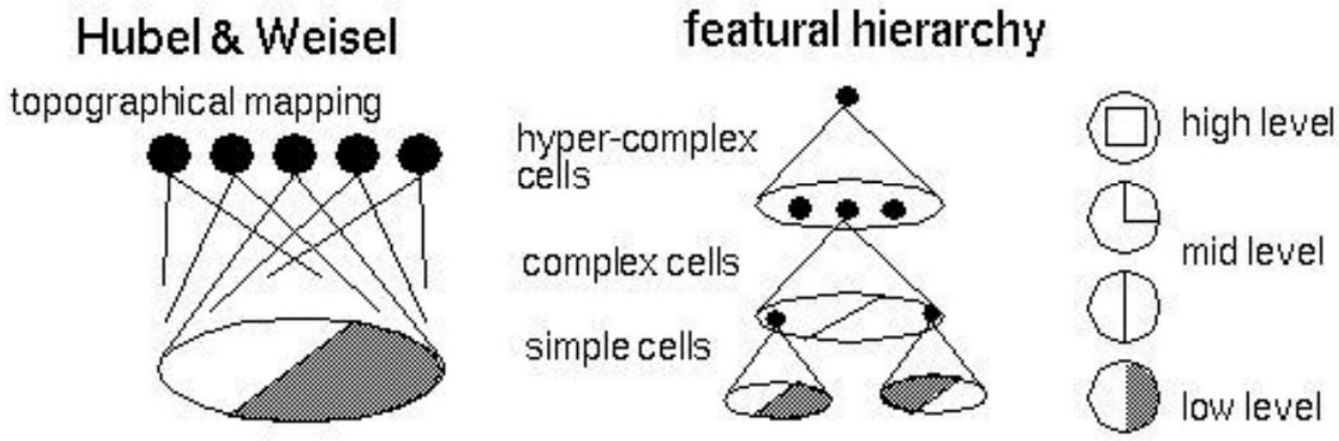
ILSVRC 2014 winner (6.7% top 5 error)

Cat neurons [Hubel, Weisel 1959, 1962]

(Be careful with biology analogies)

receptive field





- *simple cells*
- *complex cells*

[http://fourier.eng.hmc.edu/e180/lectures/v1/node7.html]

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like **[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX**
  - where N is usually up to ~5, M is large, 0 <= K <= 2.
  - but recent advances such as ResNet/GoogLeNet challenge this paradigm