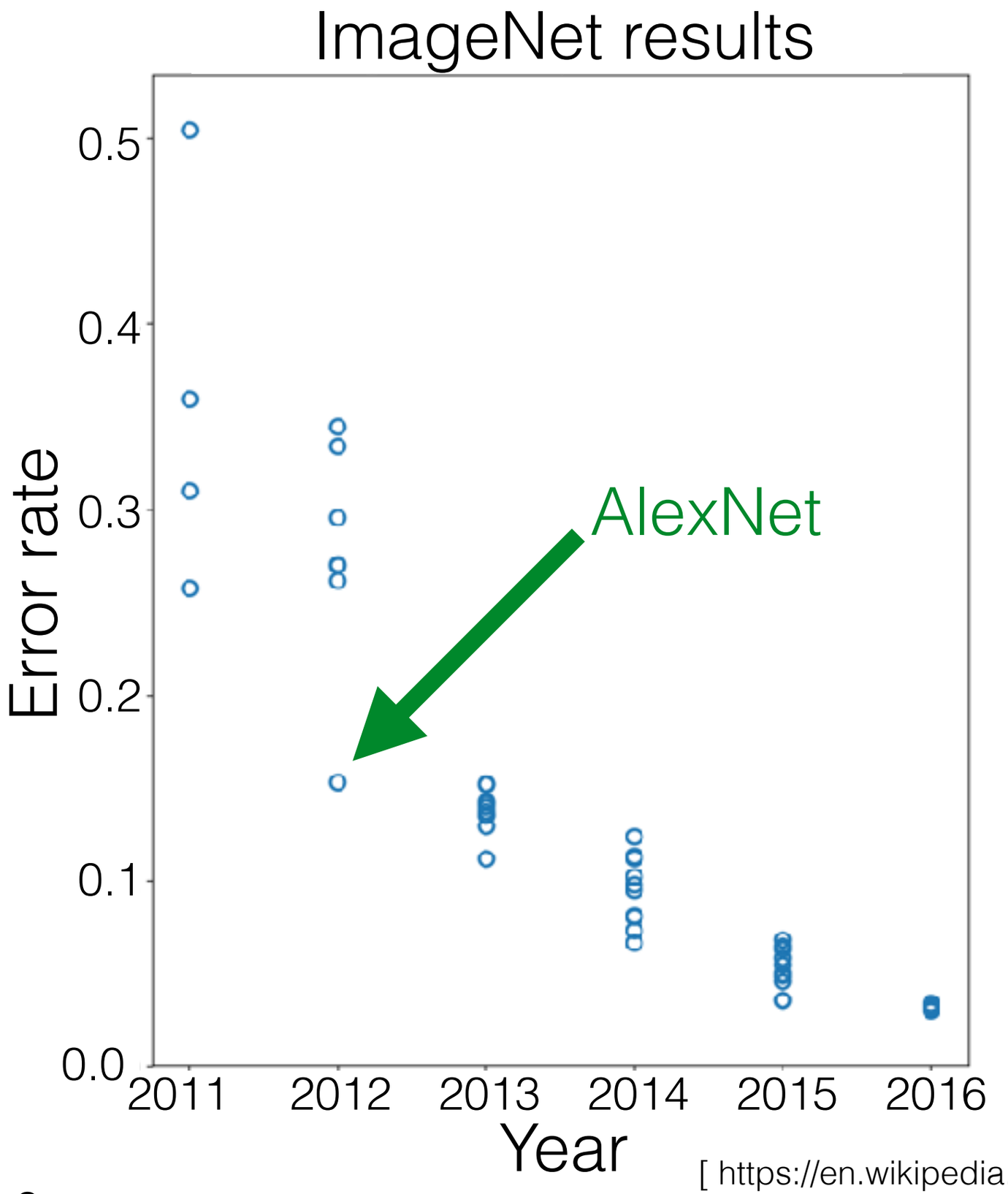
Convolutional Neural Networks

Prof. Tamara Broderick

Edited From 6.036 Fall21 Offering

Impact of CNNs

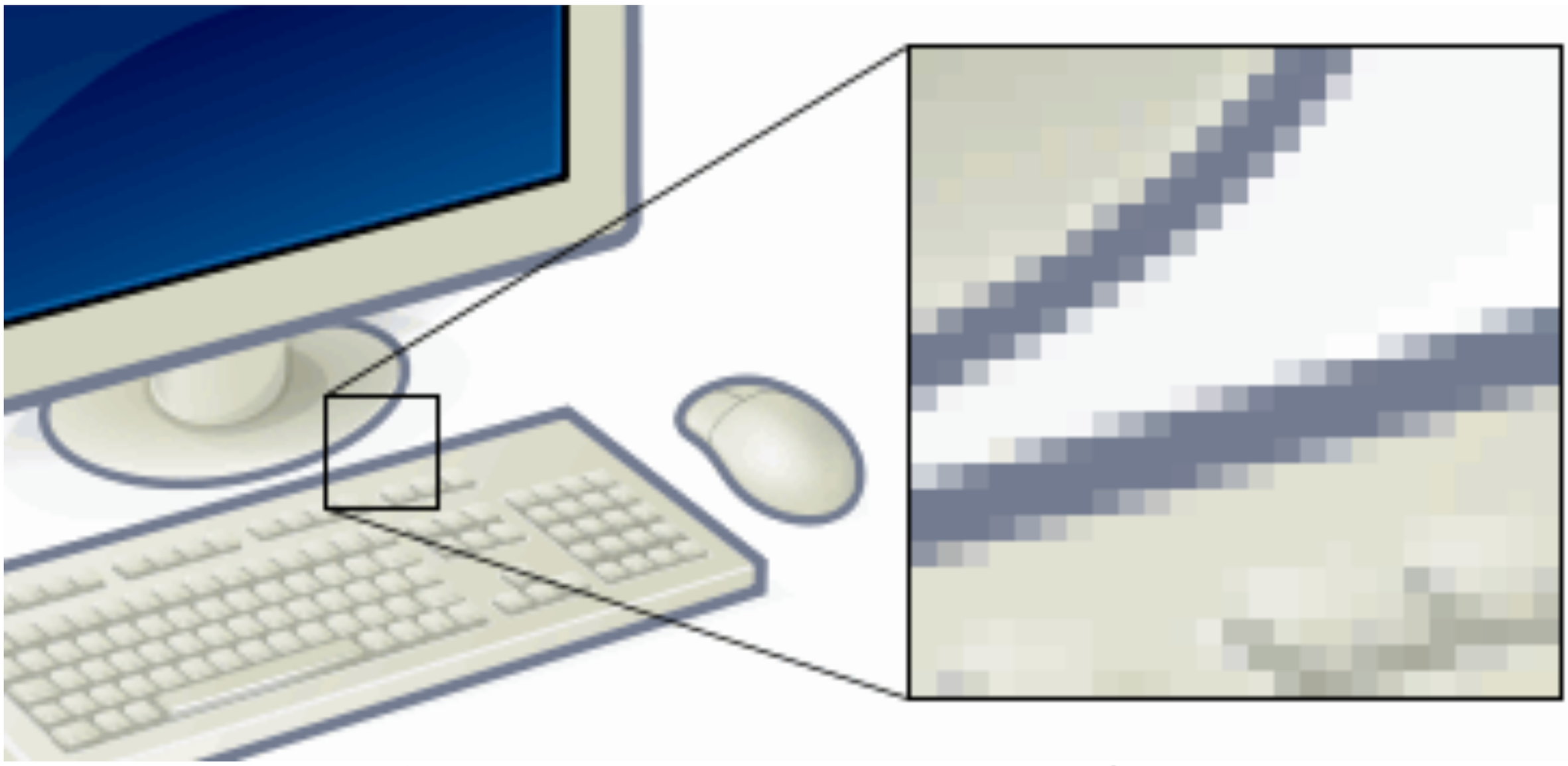


- 2010–2017: large-scale image classification challenge
- Recent AI boom
- 1960s, 1980s, today: neural networks
- Since 1980s: CNNs
- Around and before and after 2012: other CNNs on GPUs

[https://en.wikipedia.org/wiki/ImageNet#History_of_the_ImageNet_Challenge]
[Russakovsky et al, "ImageNet Large Scale Visual Recognition Challenge", IJCV, 2015]

Images

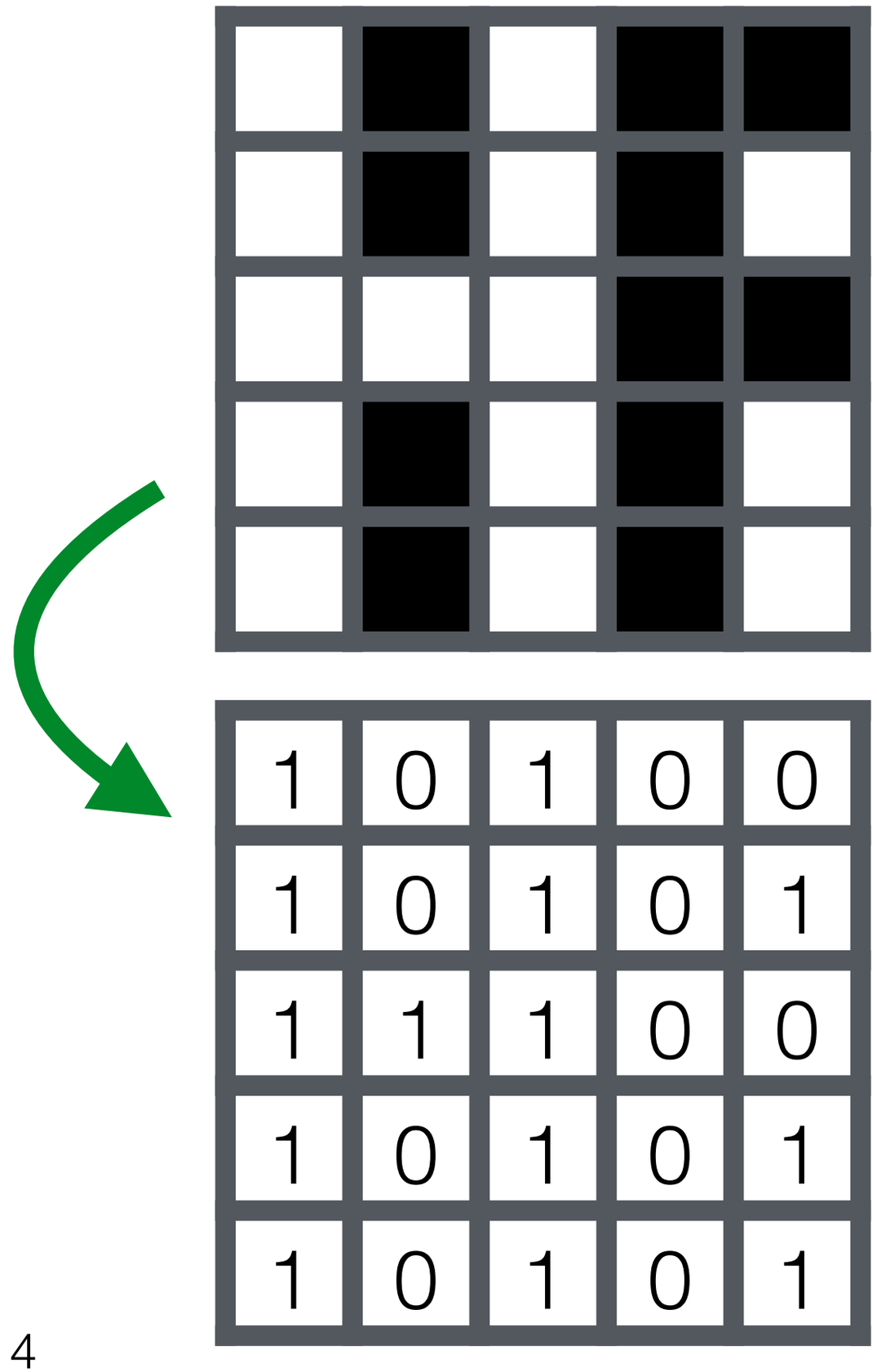
- Potential uses of image classification: Detect tumor (type) from medical scans, image search online, autonomous driving



- Recall: images are made of pixels

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

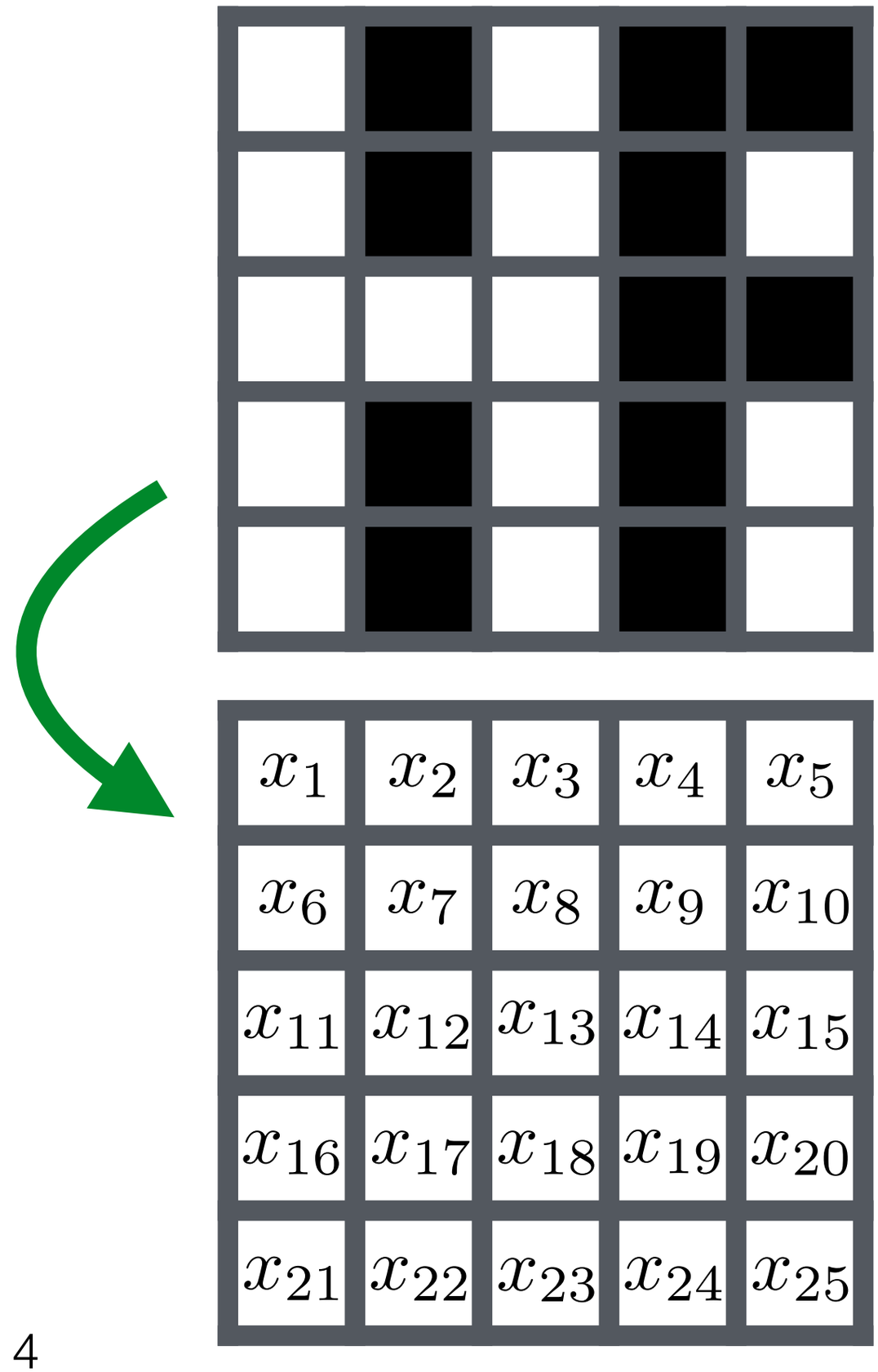
Images



- We'll focus on grayscale images
  - Each pixel takes a value between 0 and *P*
  - Here, 0: black, 1: white
  - Common to use larger *P*

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

Images

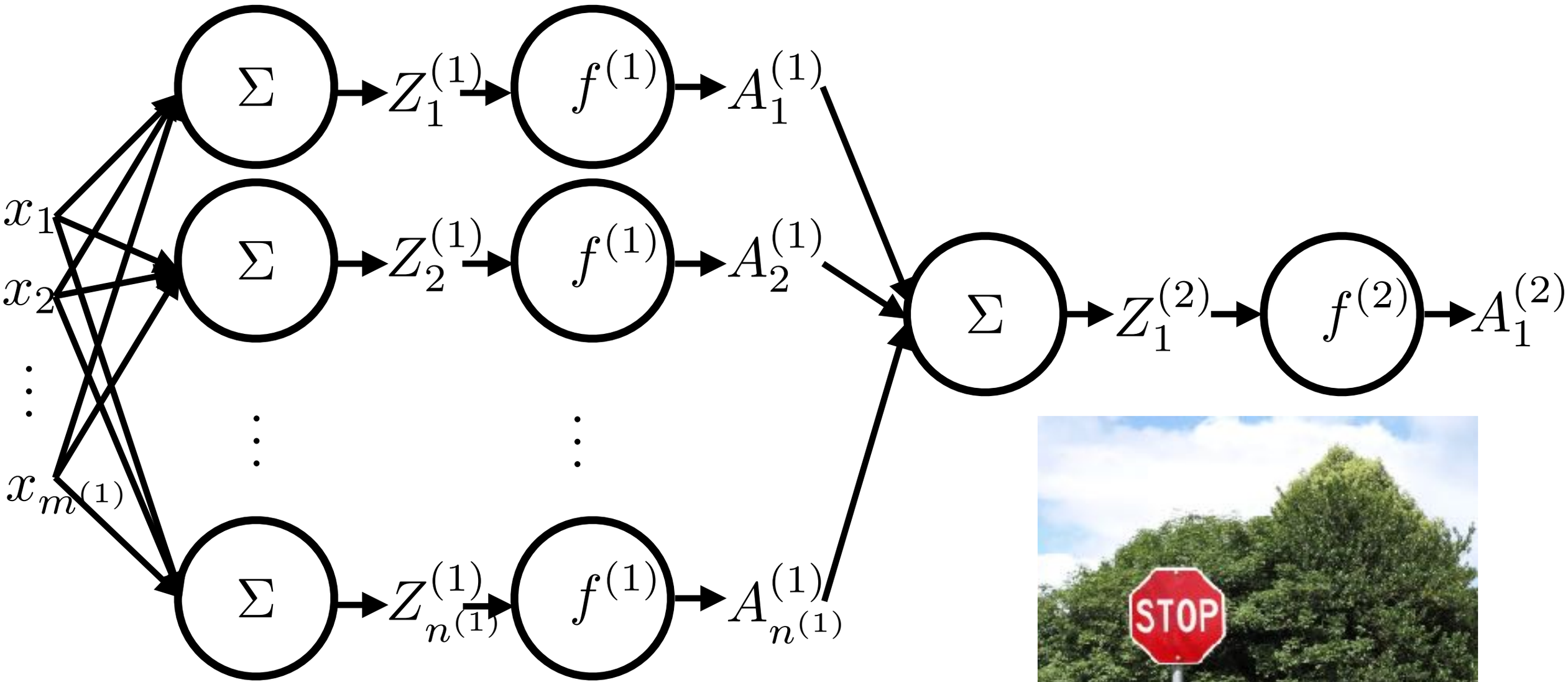


- We'll focus on grayscale images
  - Each pixel takes a value between 0 and P
  - Here, 0: black, 1: white
  - Common to use larger P

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

Previous neural nets in this class

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

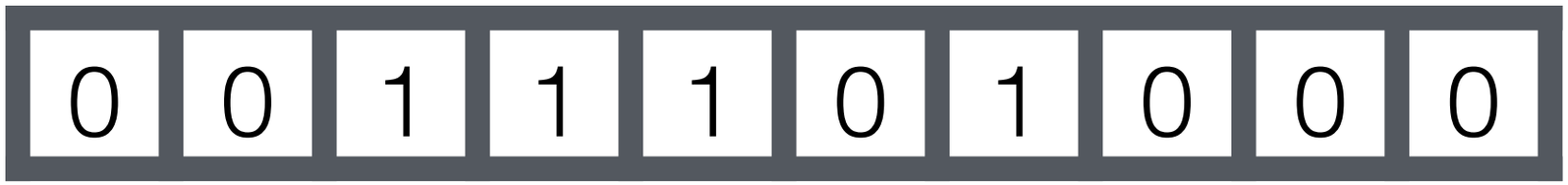


But we know more about images:

- Spatial locality
- Translation invariance

Convolutional Layer: 1D example

A 1D image:



Letter | Published: 07 January 2019

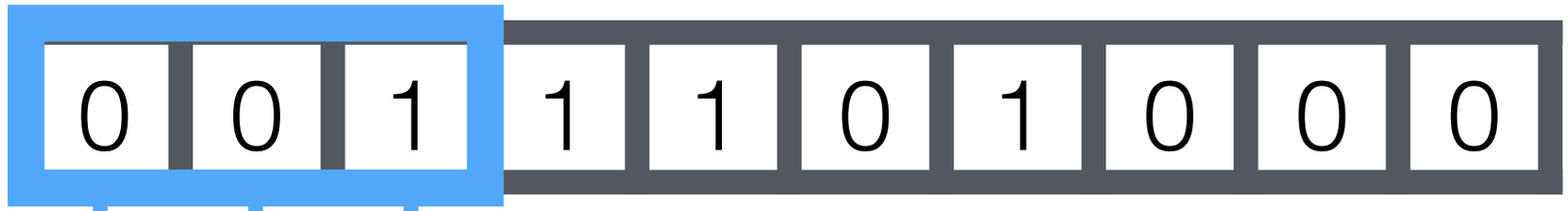
Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network

Awni Y. Hannun ✉, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H. Tison, Codie Bourn, Mintu P. Turakhia & Andrew Y. Ng

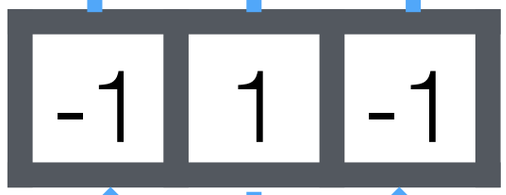
[https://www.nature.com/articles/s41591-018-0268-3]

Convolutional Layer: 1D example

A 1D image:

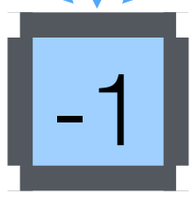


A filter:



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

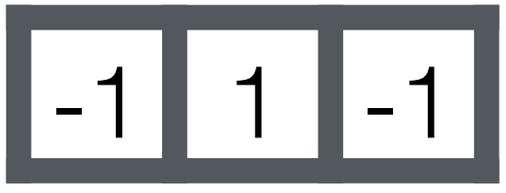
After convolution*:



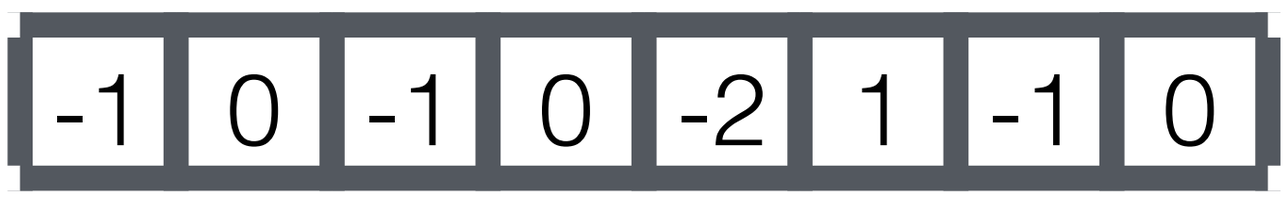
Convolutional Layer: 1D example

A 1D image:

A filter:



After convolution*:

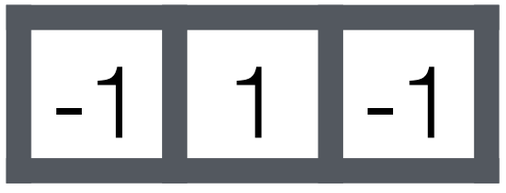


*correlation

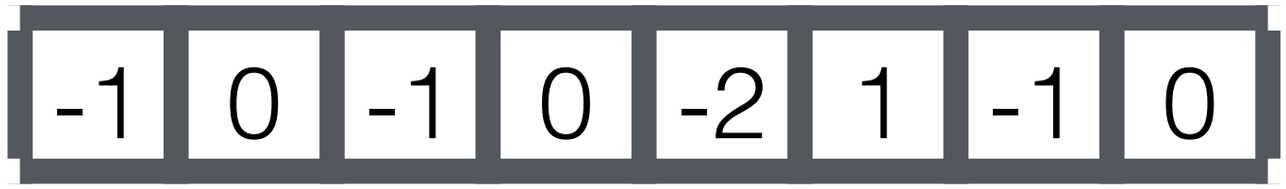
Convolutional Layer: 1D example

A 1D image:

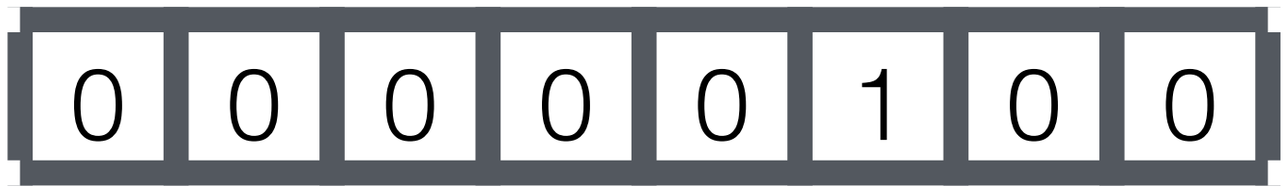
A filter:



After convolution*:



After ReLU:

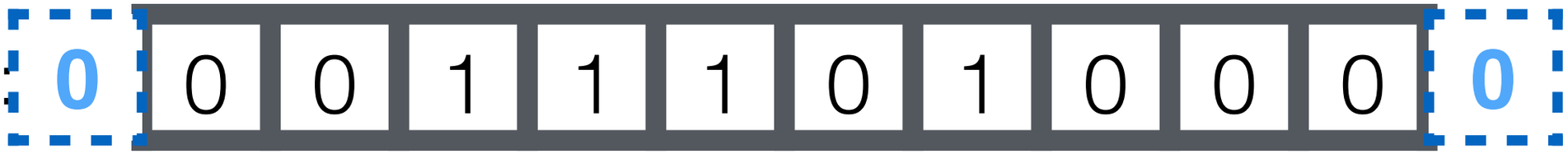


What does the filter do?

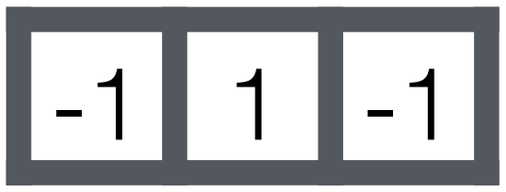
*correlation

Convolutional Layer: 1D example

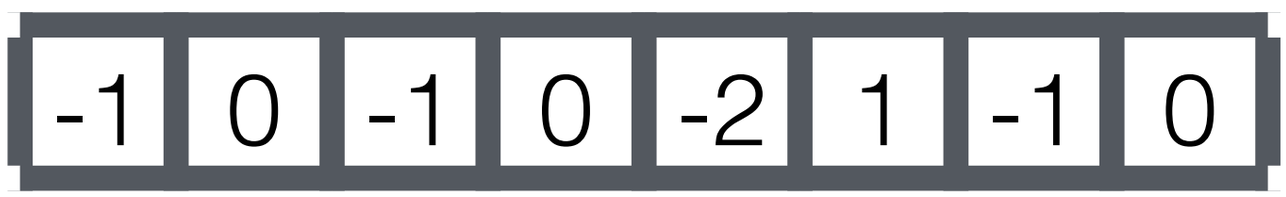
A 1D image:



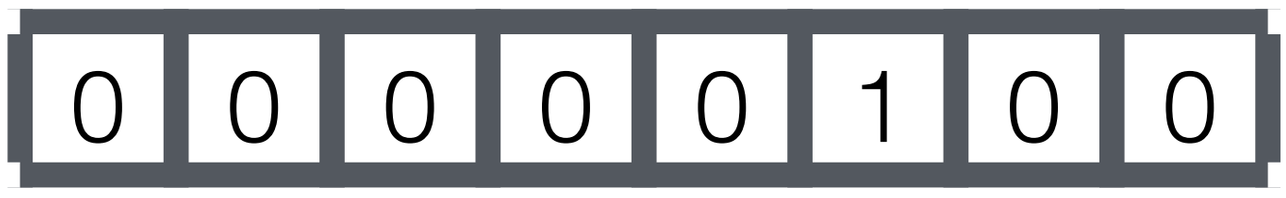
A filter:



After convolution*:

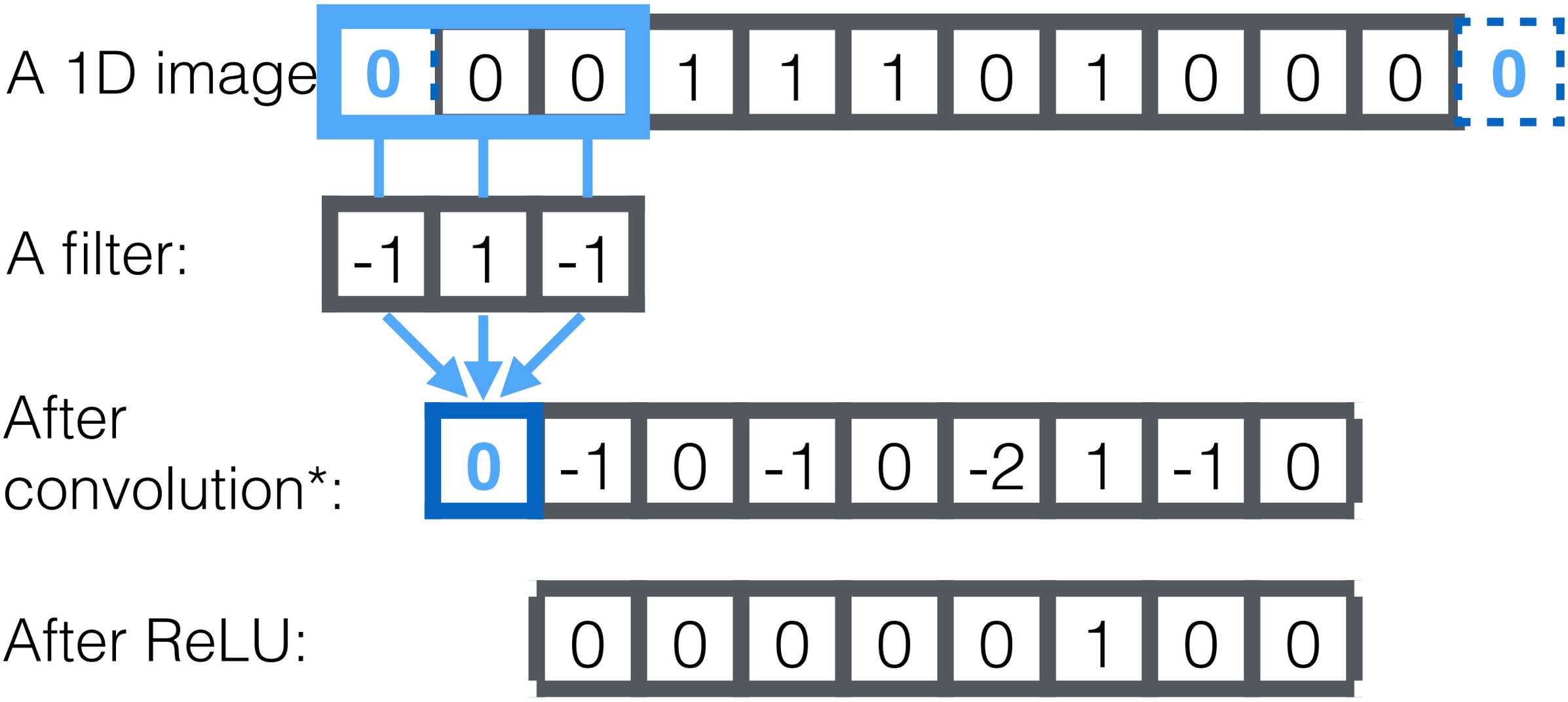


After ReLU:



*correlation

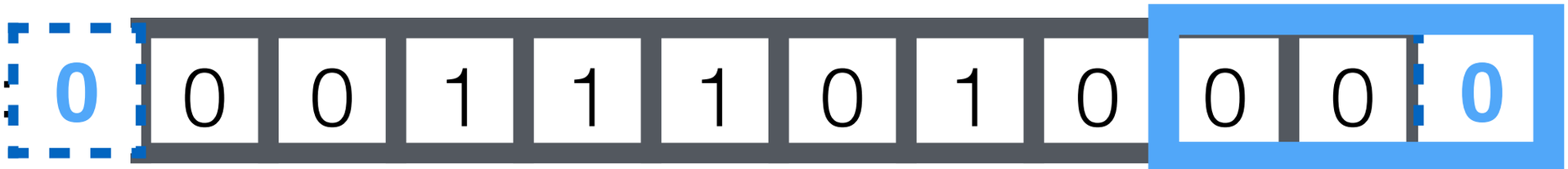
Convolutional Layer: 1D example



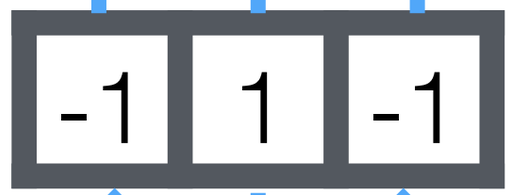
*correlation

Convolutional Layer: 1D example

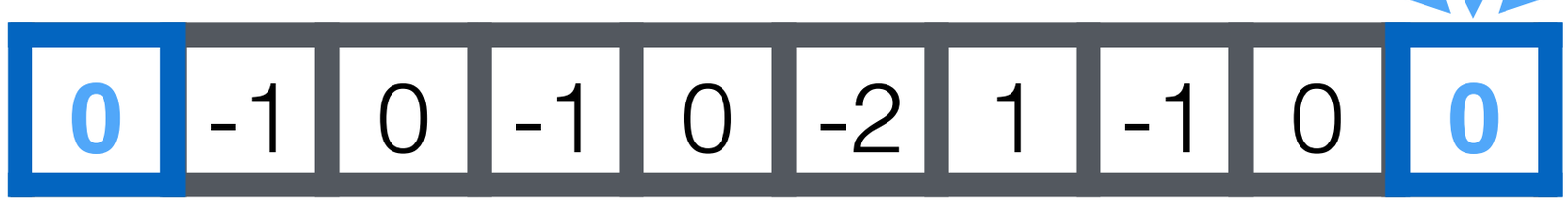
A 1D image:



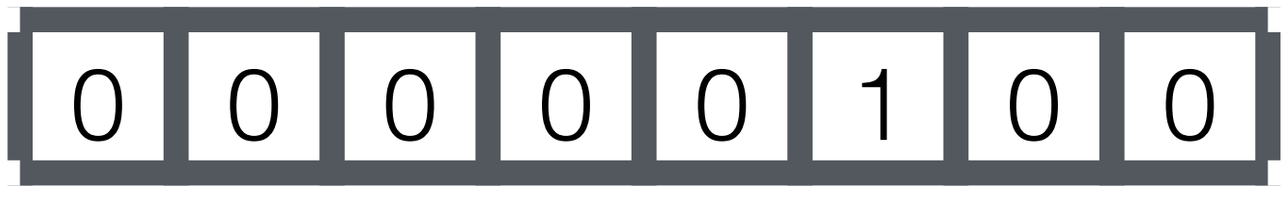
A filter:



After convolution*:



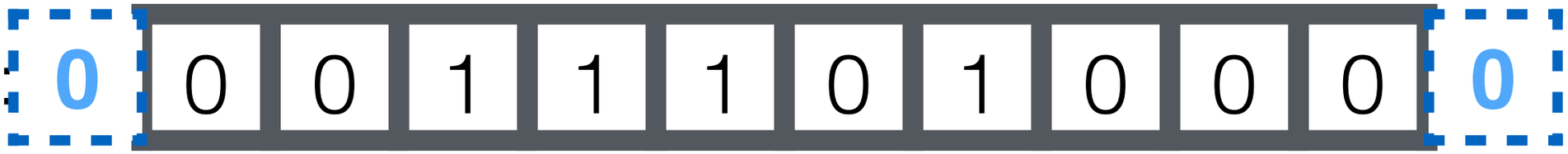
After ReLU:



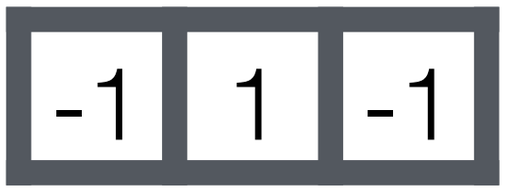
*correlation

Convolutional Layer: 1D example

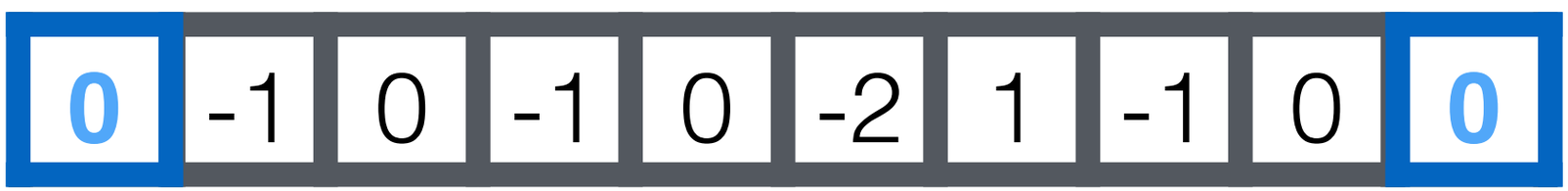
A 1D image:



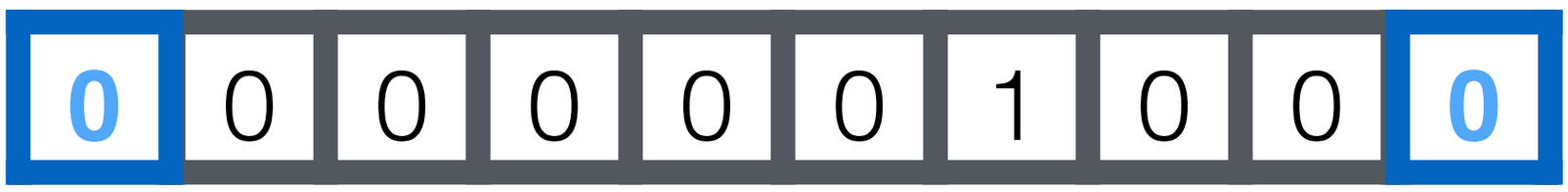
A filter:



After convolution*:

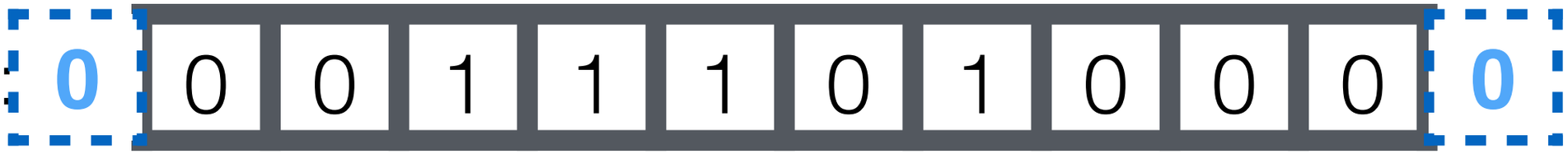


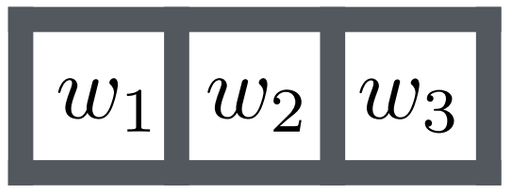
After ReLU:



*correlation

Convolutional Layer: 1D example

A 1D image: 

A filter: 

with bias *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

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

-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:

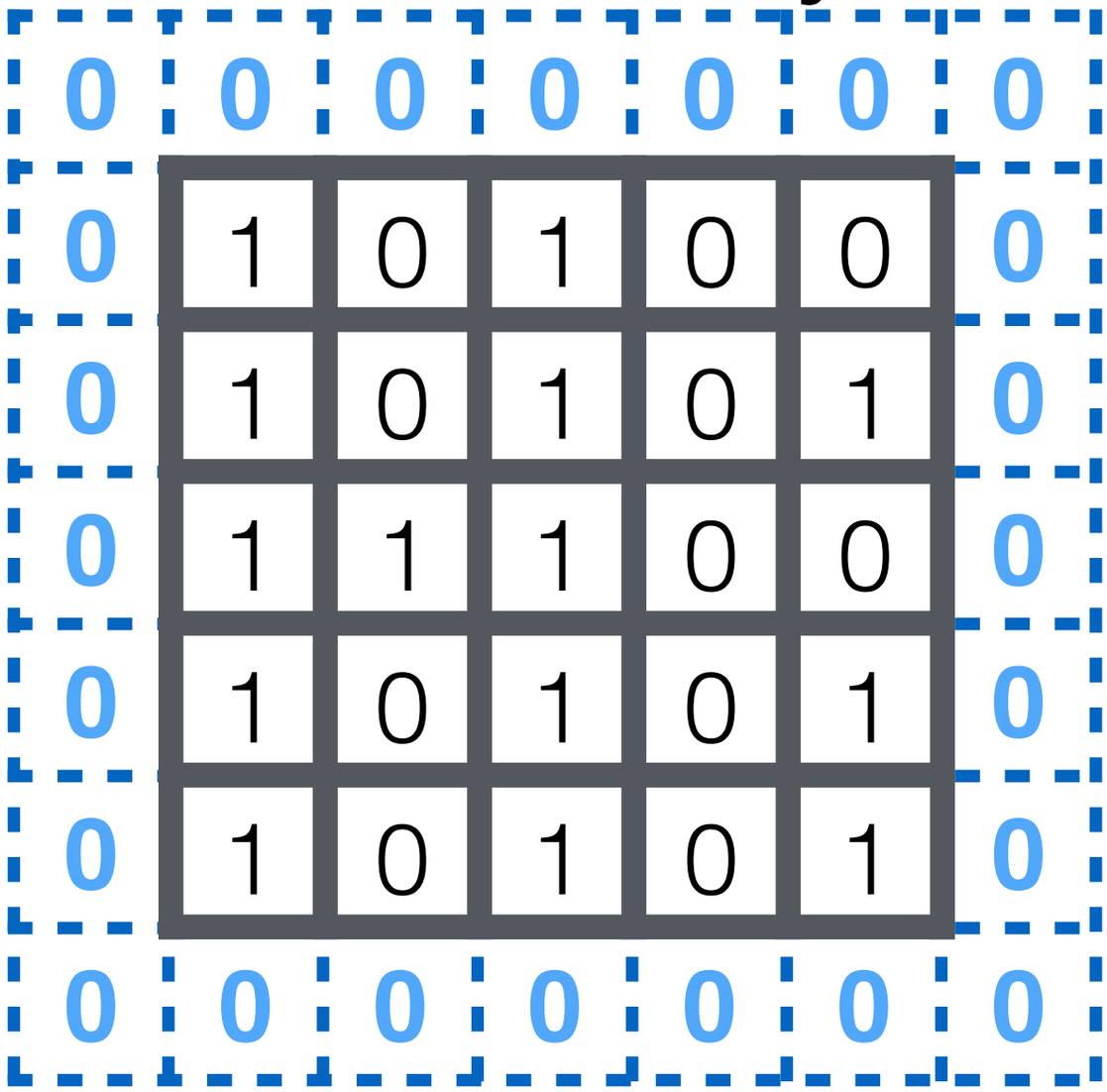
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

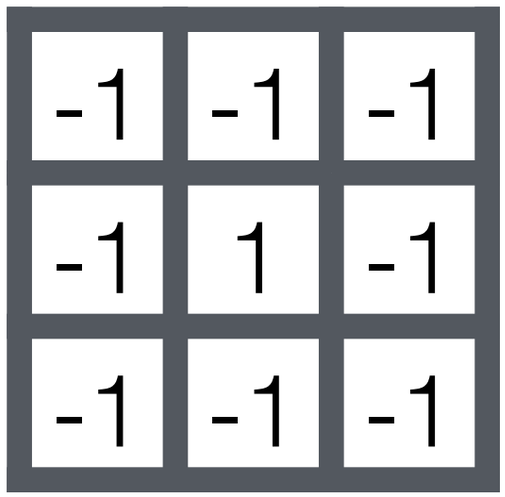
-7	-2	-4
-5	-2	-5
-7	-2	-5

Convolutional Layer: 2D example

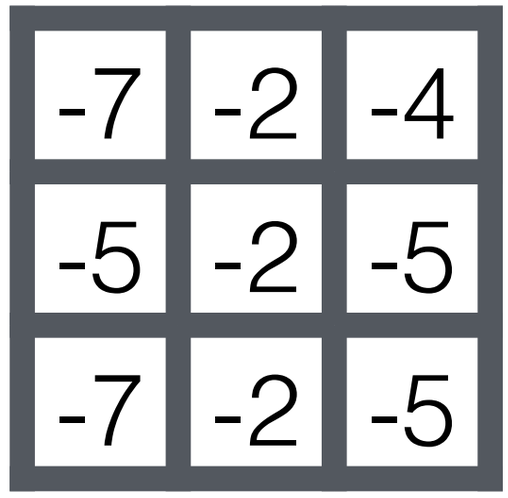
A 2D image:



A filter:



After convolution:



Convolutional Layer: 2D example

A 2D image:

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

A filter:

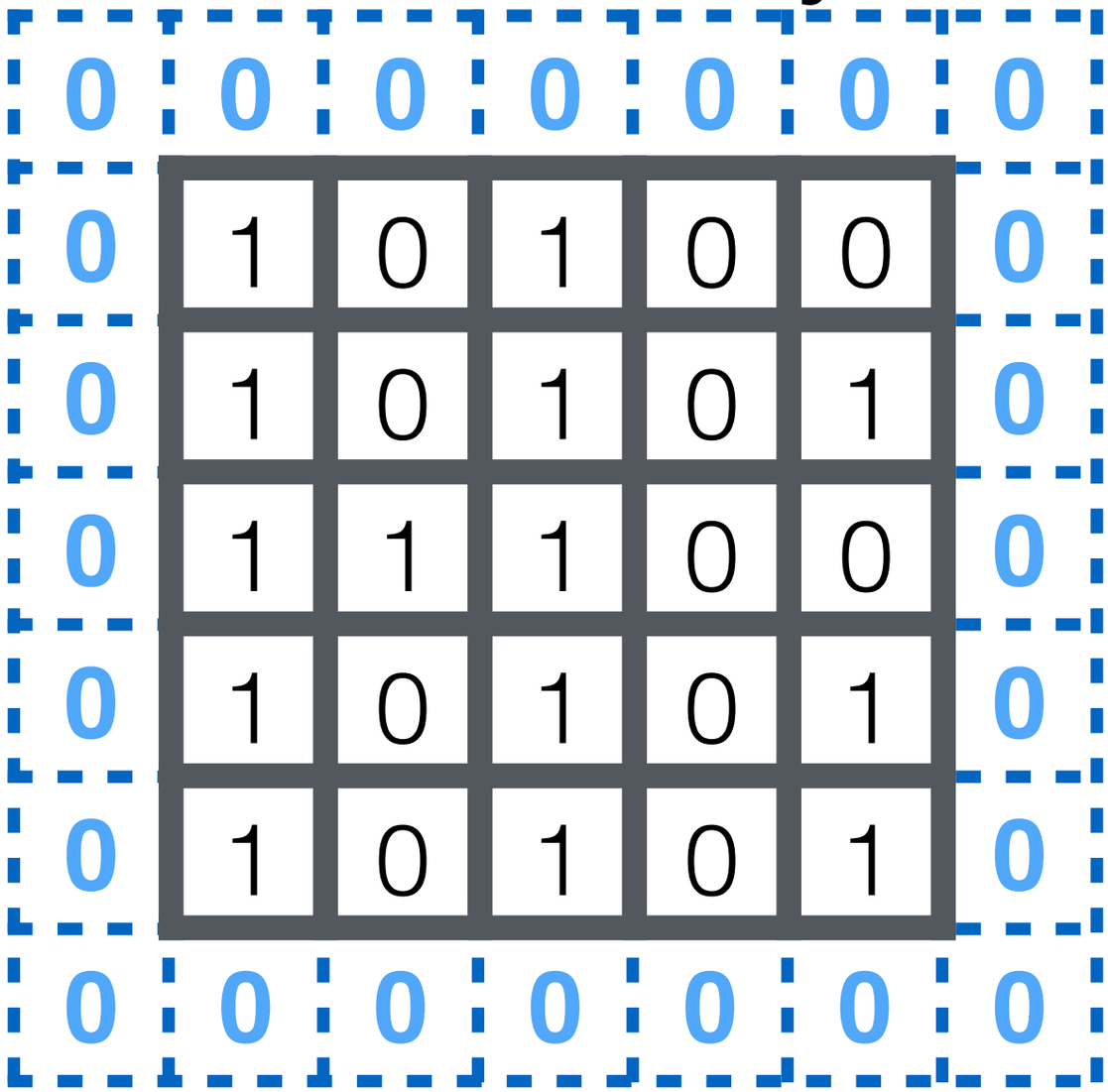
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

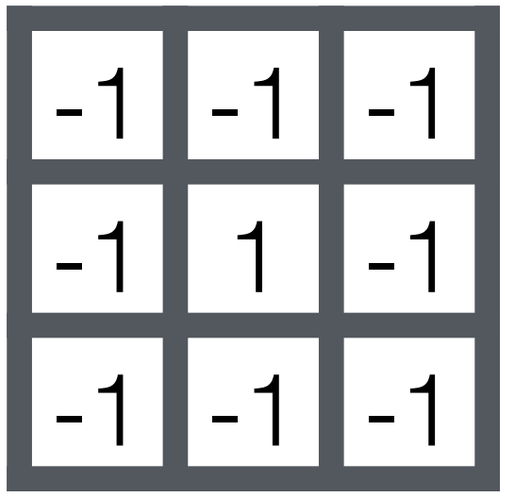
0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

Convolutional Layer: 2D example

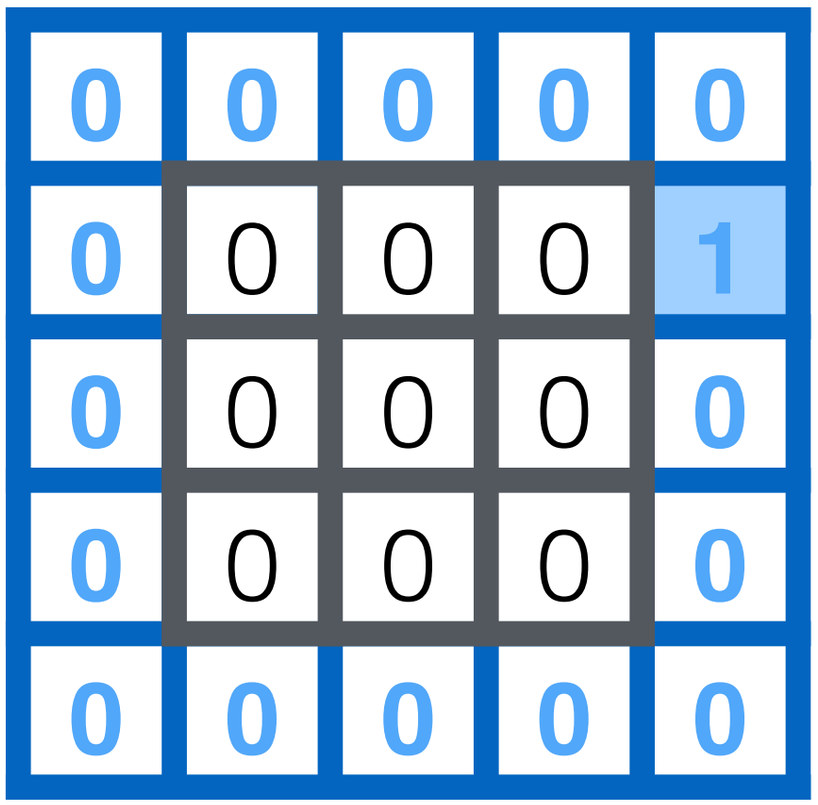
A 2D image:



A filter:



After convolution & ReLU:



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

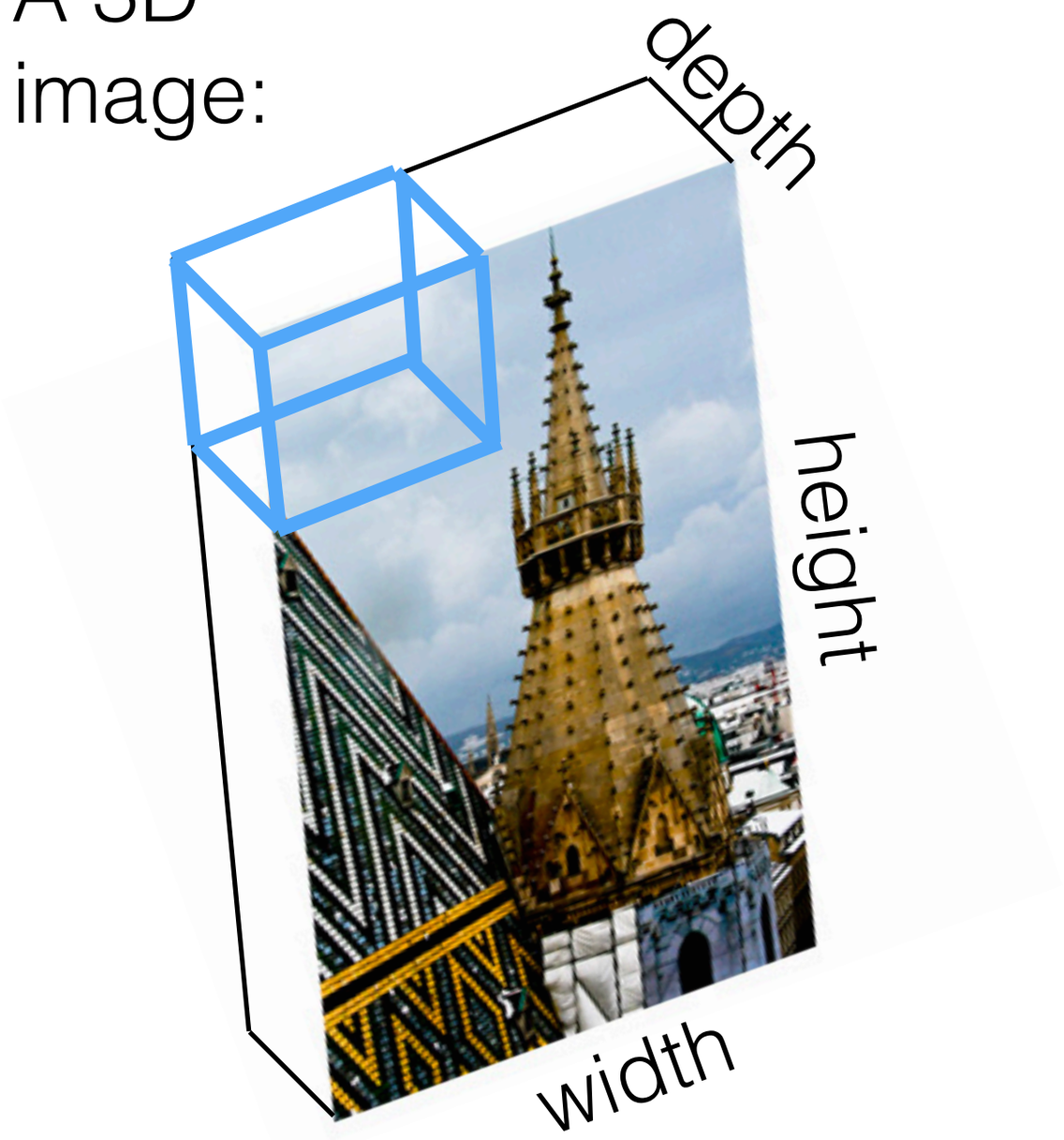
A filter:

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

with bias b

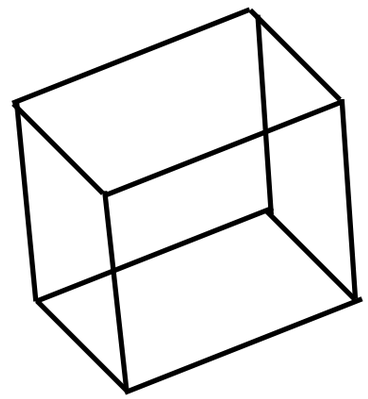
Convolutional Layer: 3D example

A 3D image:



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

A filter:

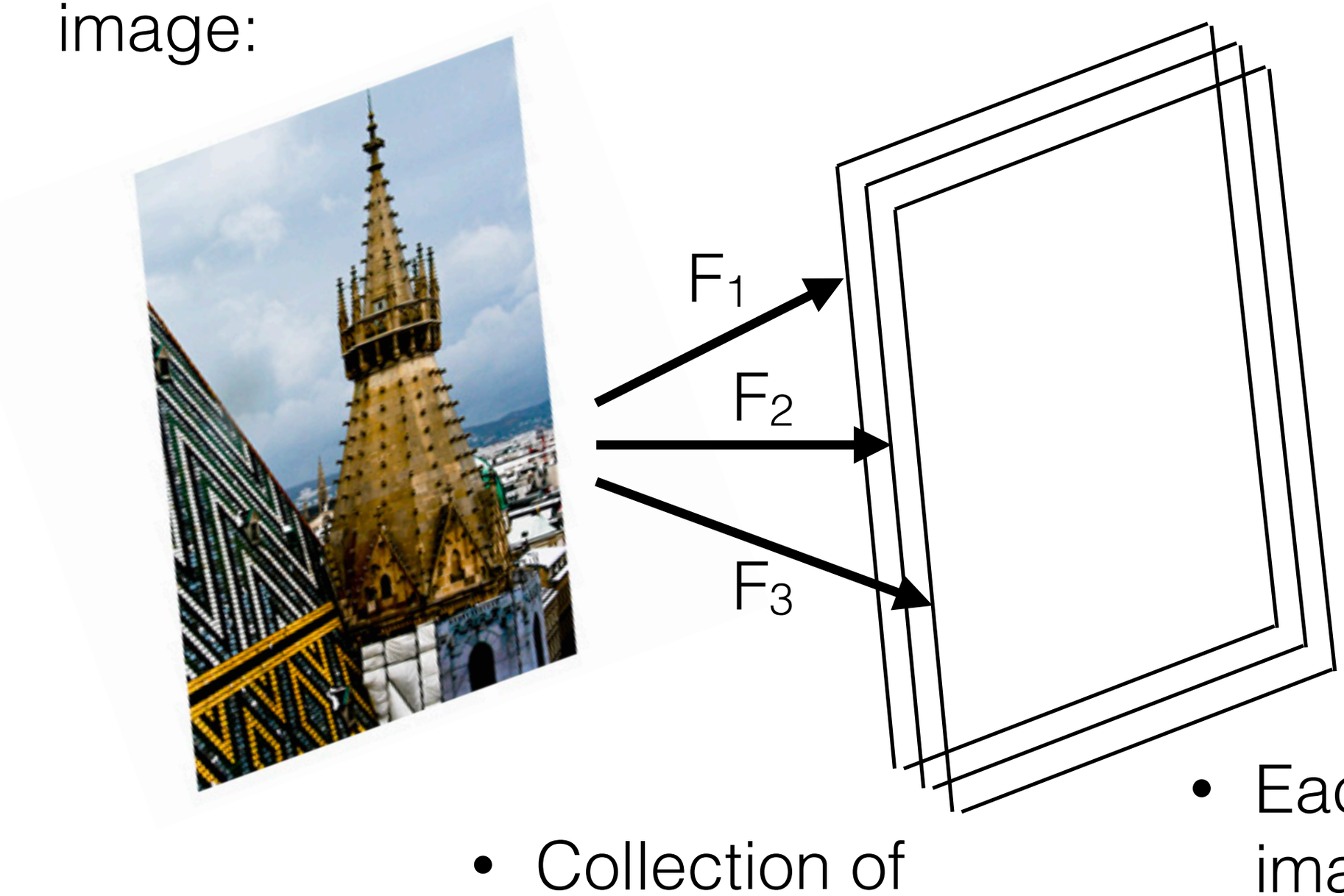


- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

[https://en.wikipedia.org/wiki/TensorFlow]

Convolutional Layer: multiple filters

An image:



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

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

- E.g. size 3x3 (“size 3”)
- E.g. stride 1

After max pooling:

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

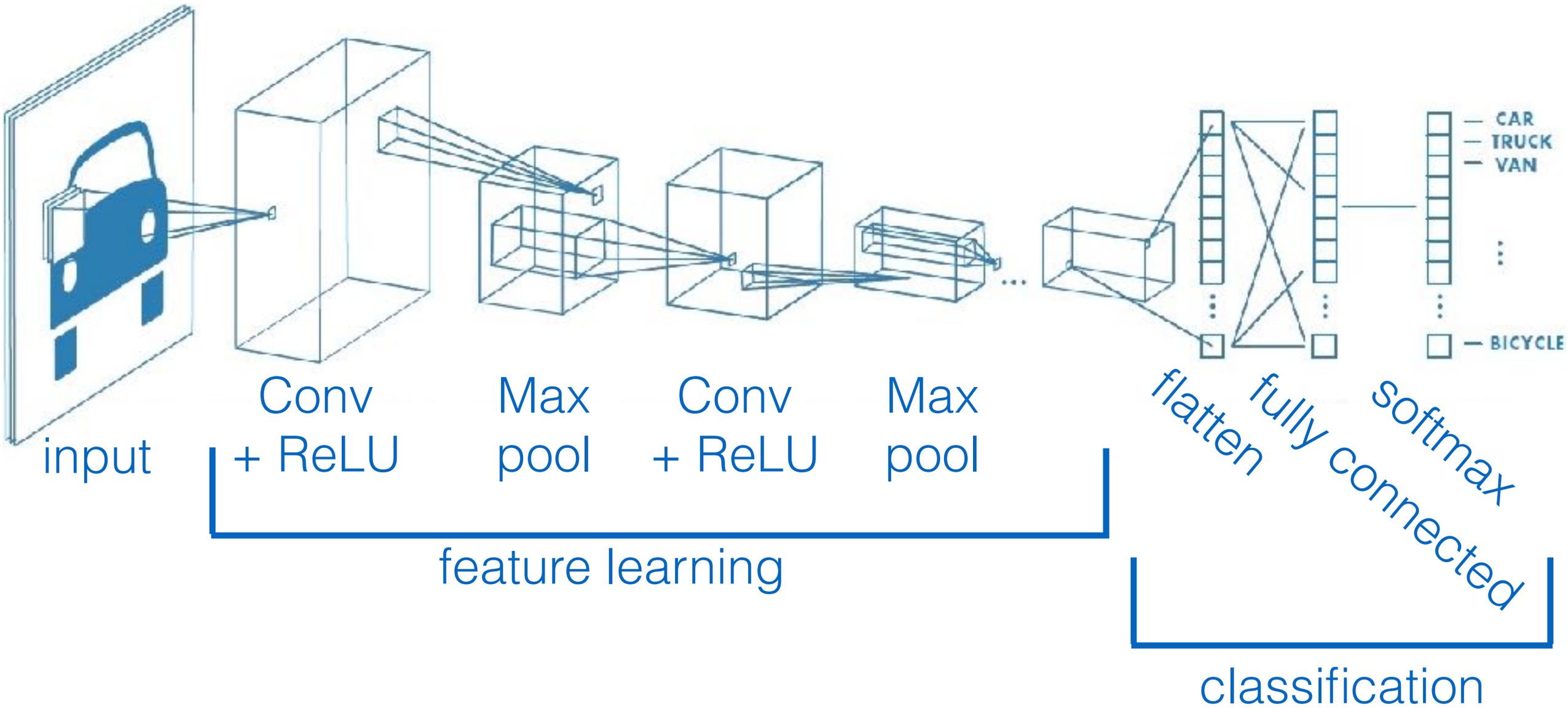
- E.g. size 3x3 (“size 3”)
- E.g. stride 3

After max pooling:

0	1
1	0

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

CNNs: example architecture



x → $\mathrm{NN}(x; W, W_0)$

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

A familiar pattern

1. Choose how to predict label (given features & parameters)
2. Choose a loss (between guessed label & actual label)
3. Choose parameters by trying to minimize the training loss

	*i*th data point	prediction for *i*th point	training loss over points 1 to *n*
Logistic regression:	$x^{(i)}$	$\mathrm{LogiReg}(x^{(i)}; \theta, \theta_0)$	$J_{\mathrm{Logi}}(\theta, \theta_0)$
Linear regression:	$x^{(i)}$	$\mathrm{LinReg}(x^{(i)}; \theta, \theta_0)$	$J_{\mathrm{Lin}}(\theta, \theta_0)$
Neural networks (e.g. CNNs):	$x^{(i)}$	$\mathrm{NN}(x^{(i)}; W, W_0)$	$J_{\mathrm{NN}}(W, W_0)$

* a key difference: NN can mean a lot of different things