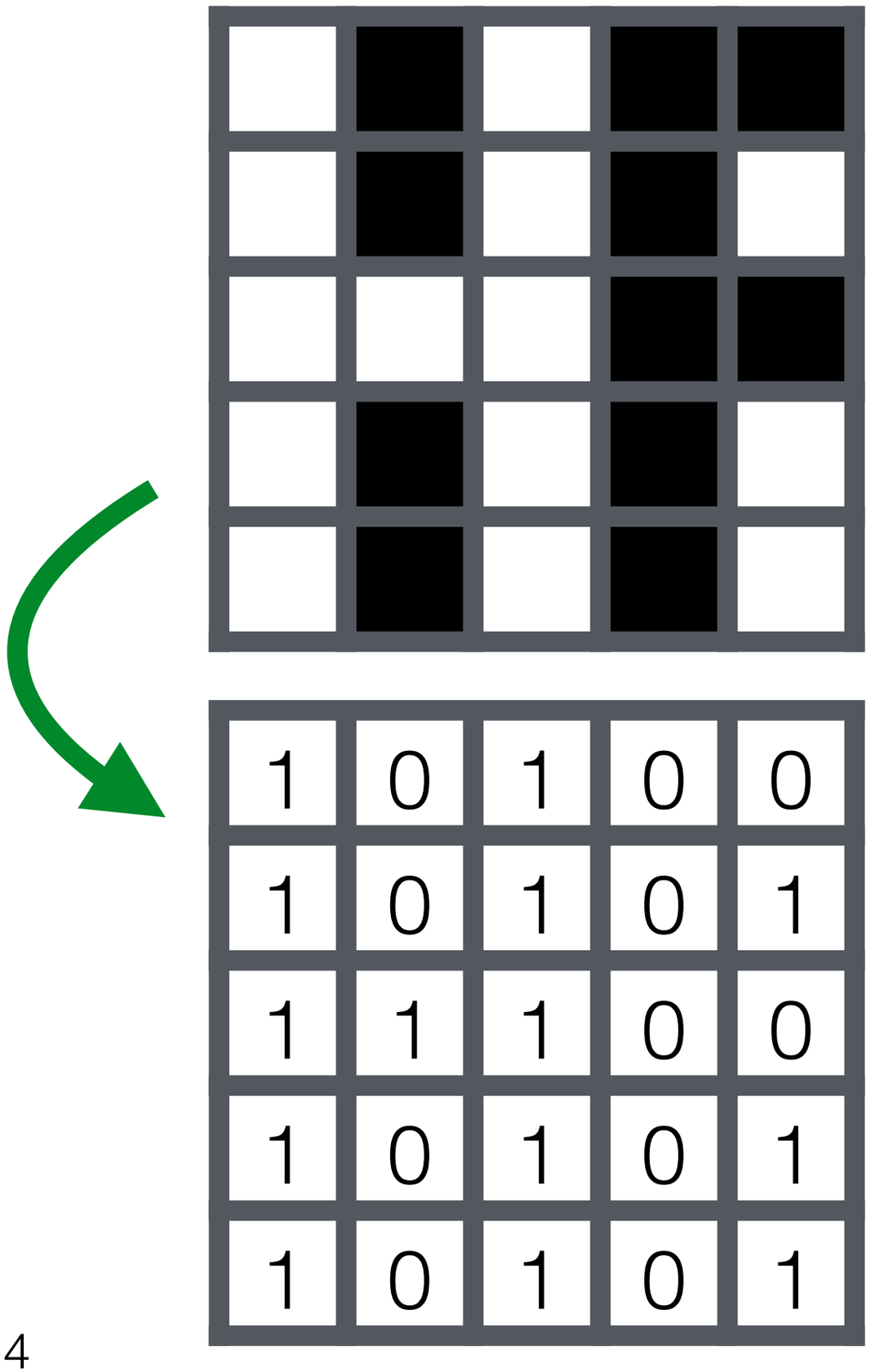
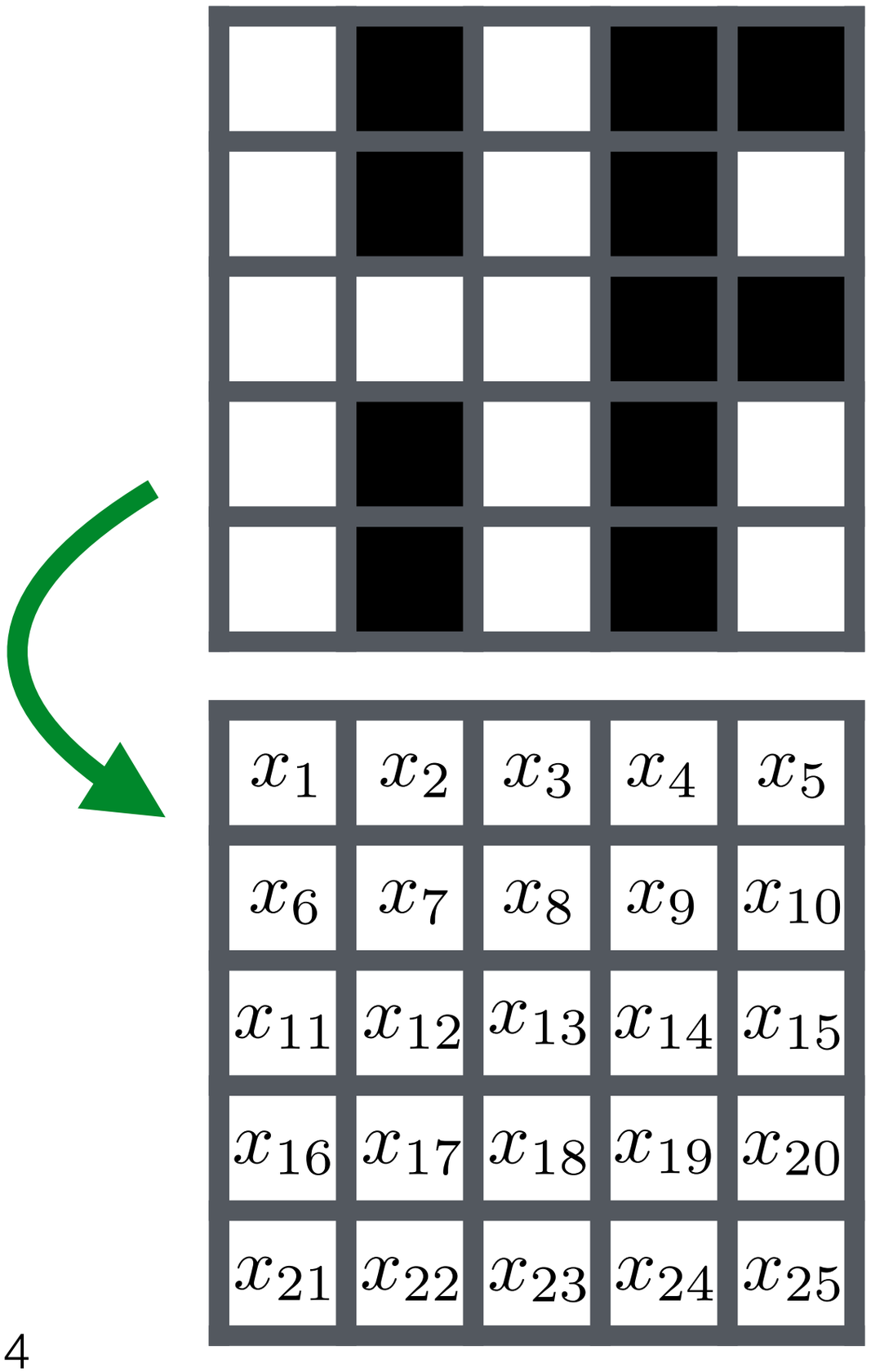
Images



- 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?

Images

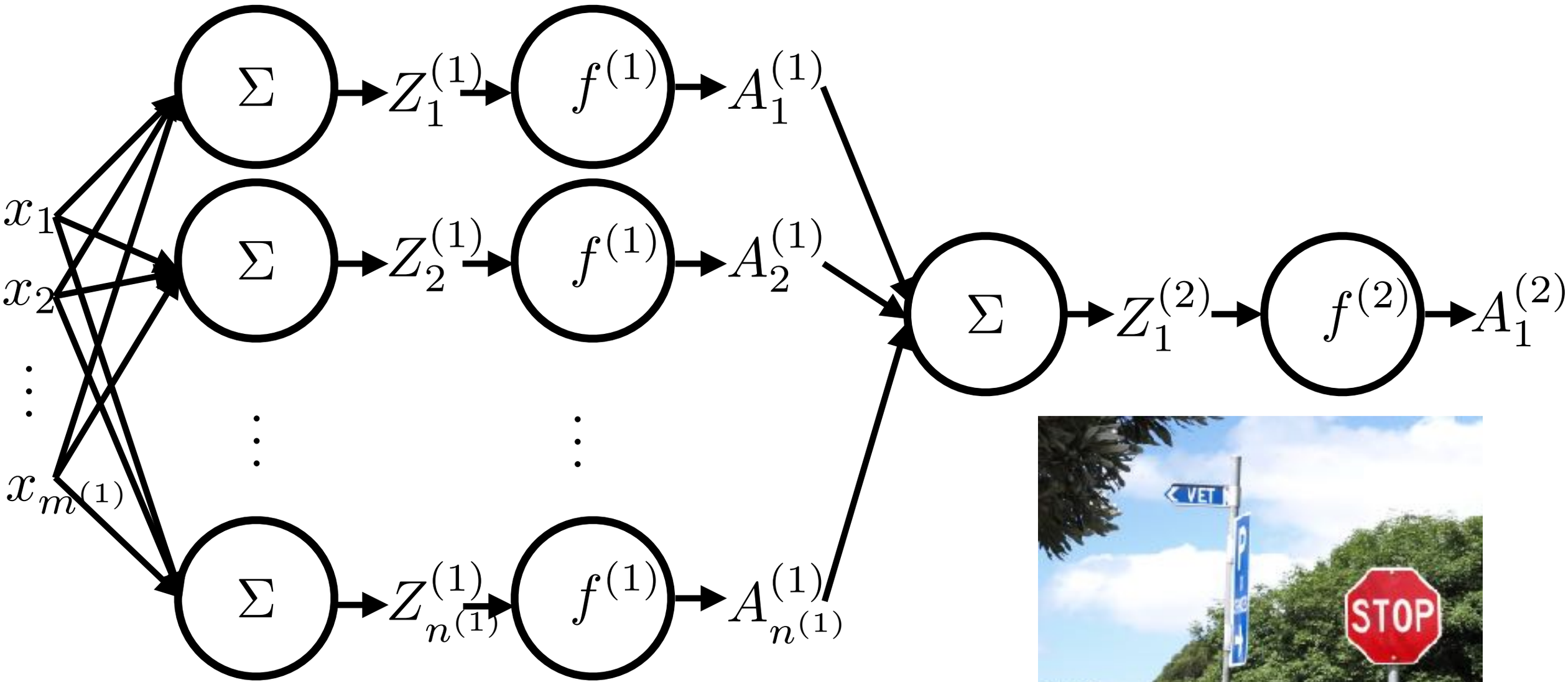


- We'll focus on grayscale images
  - Each pixel takes a value between 0 and P
  - Here, 0: black, 1: white
  - Larger P in Lab Week 08

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

Previous neural nets in this class

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

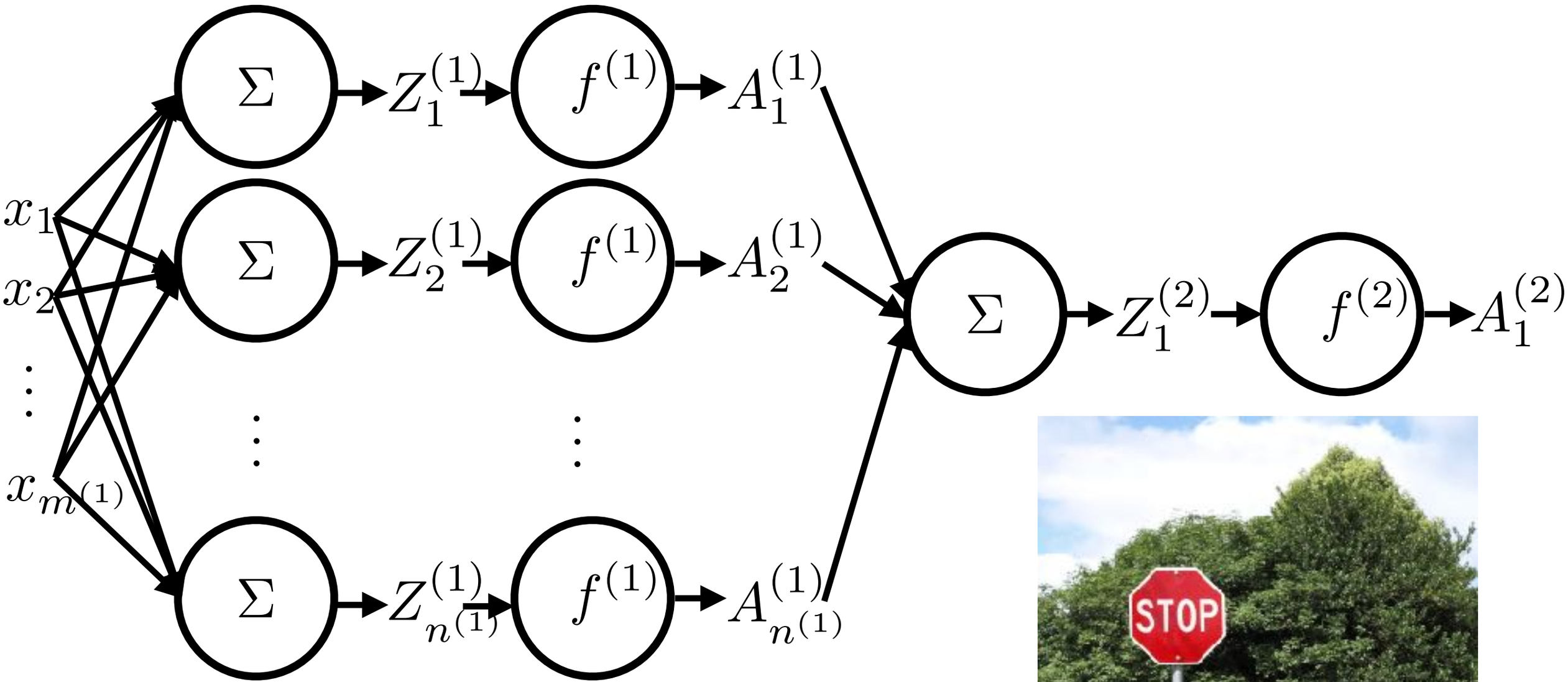


But we know more about images:

- Spatial locality
- Translation invariance

Previous neural nets in this class

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

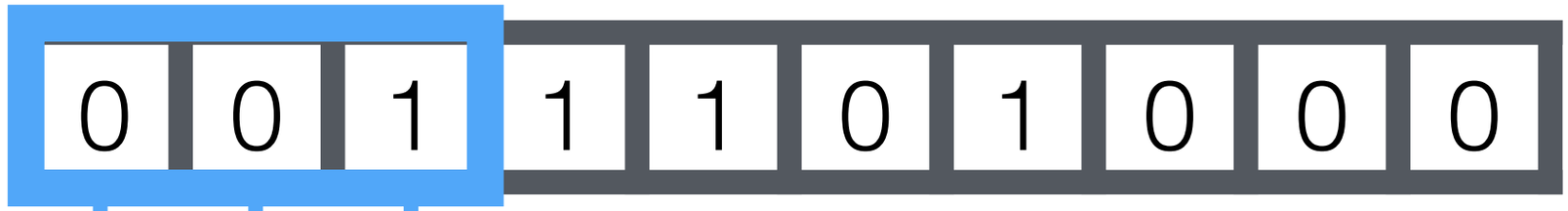


But we know more about images:

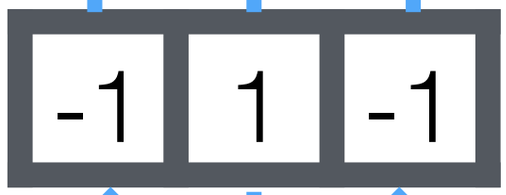
- Spatial locality
- Translation invariance

Convolutional Layer: 1D example

A 1D image:

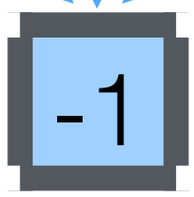


A filter:



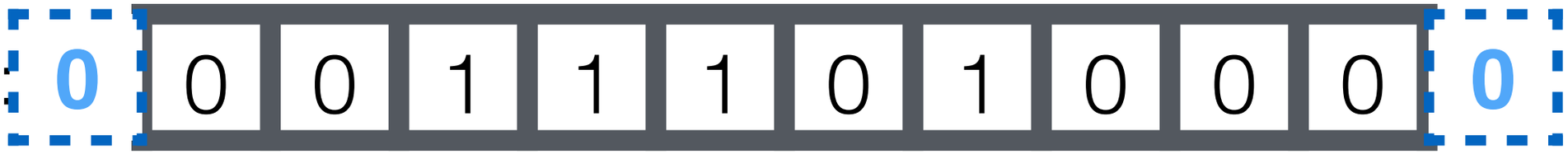
0 * -1 + 0 * 1 + 1 * -1 = -1

After convolution*:

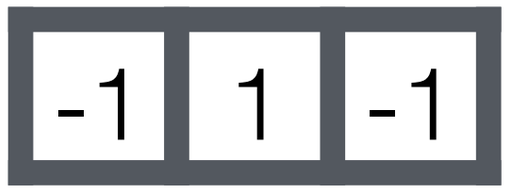


Convolutional Layer: 1D example

A 1D image:

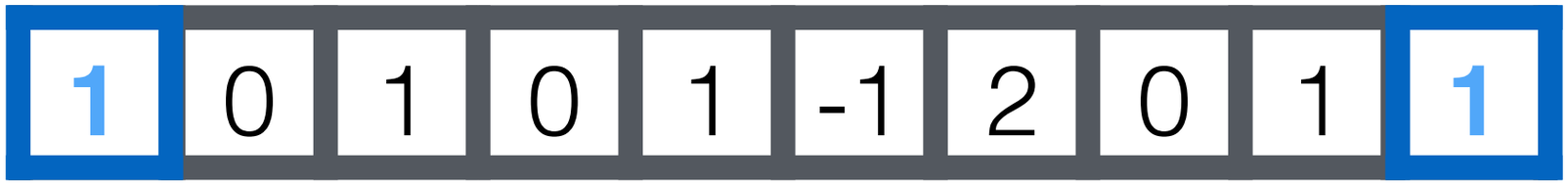


A filter:

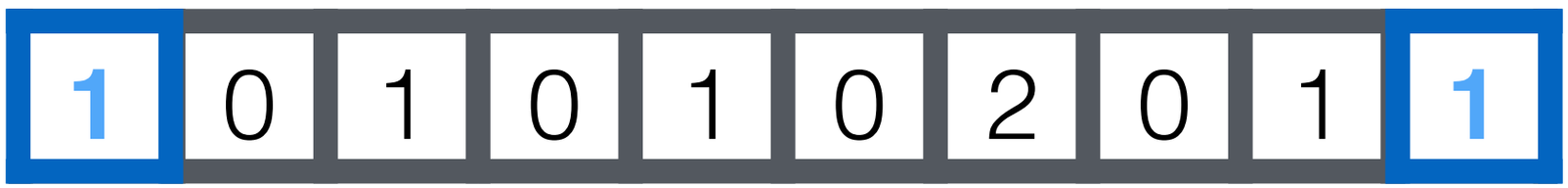


with bias +1

After convolution*:

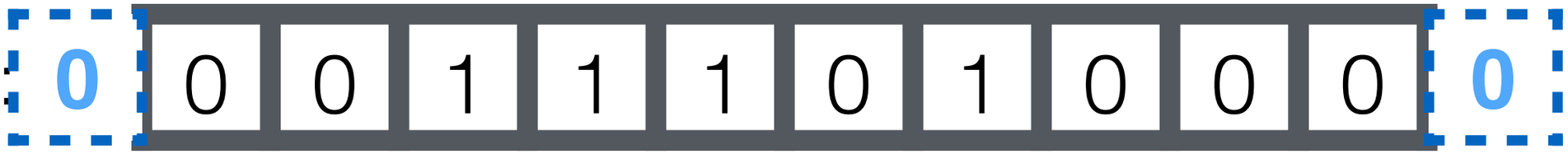


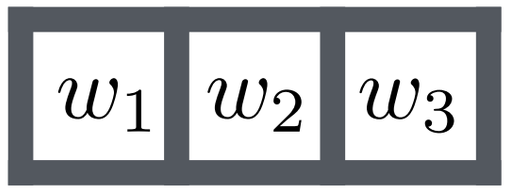
After ReLU:



*correlation

Convolutional Layer: 1D example

A 1D image: 

A filter: 

with bias *b*

After convolution*: 

After ReLU: 

- 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:

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:

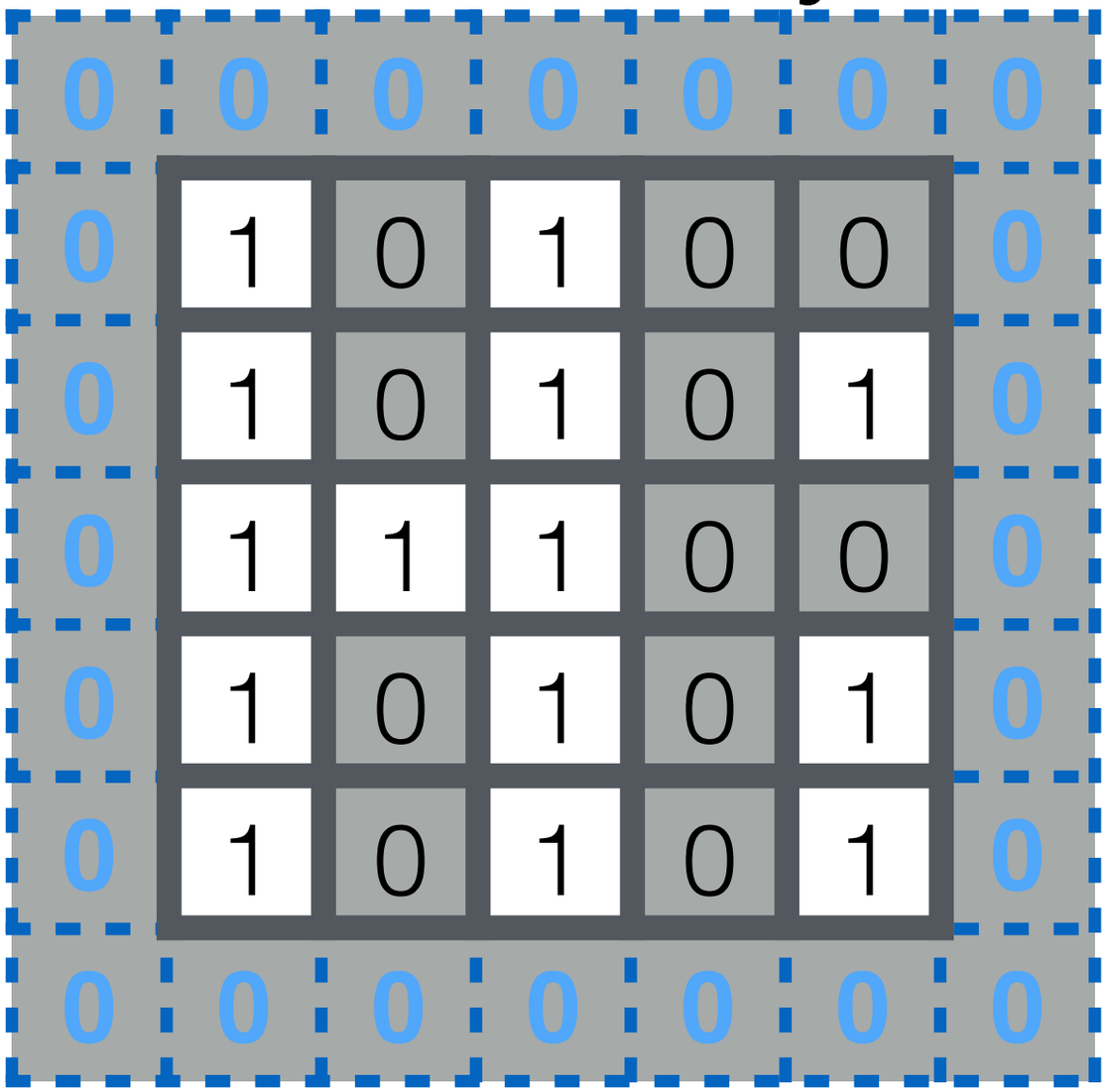
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

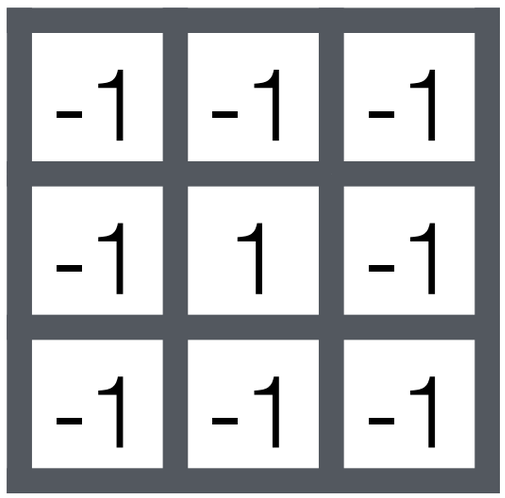
0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

Convolutional Layer: 2D example

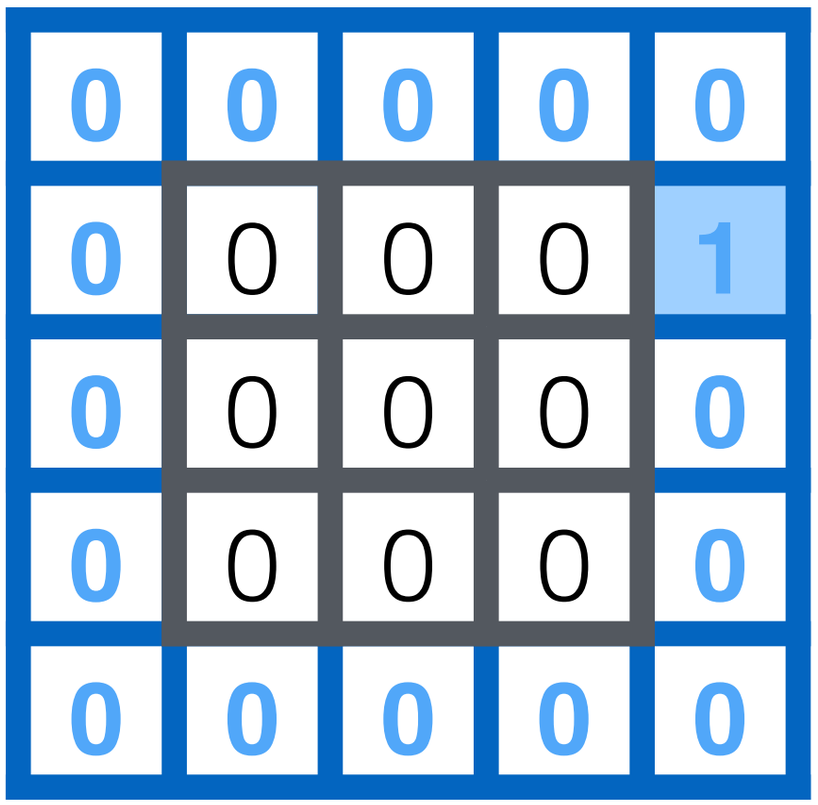
A 2D image:



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

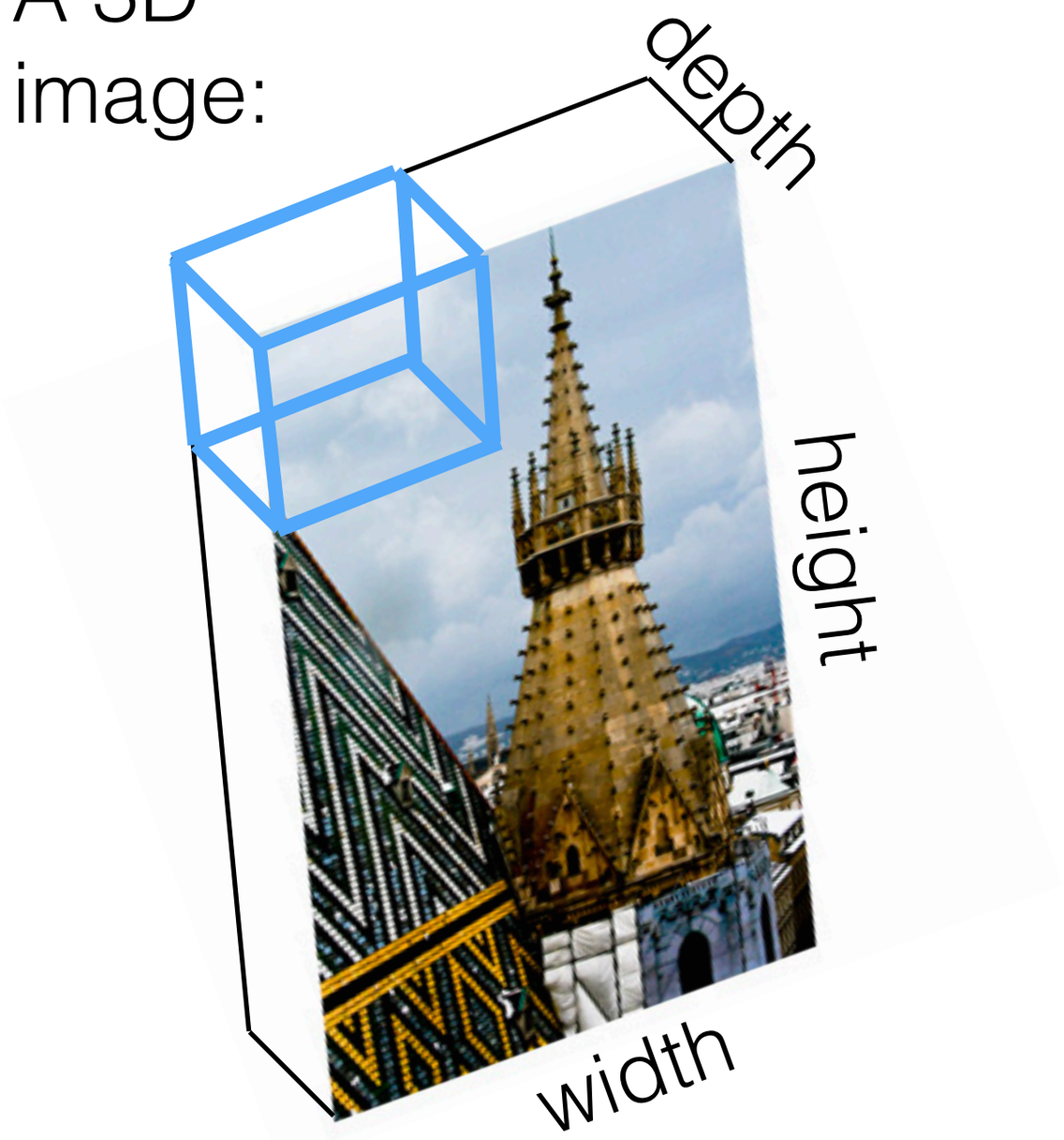
A filter:

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

with bias b

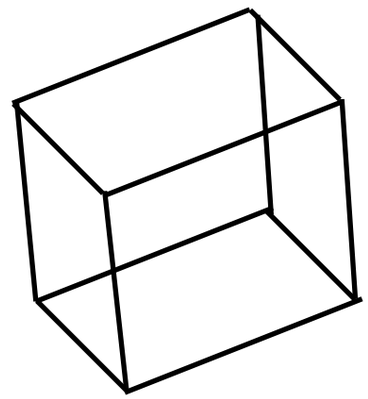
Convolutional Layer: 3D example

A 3D image:



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

A filter:

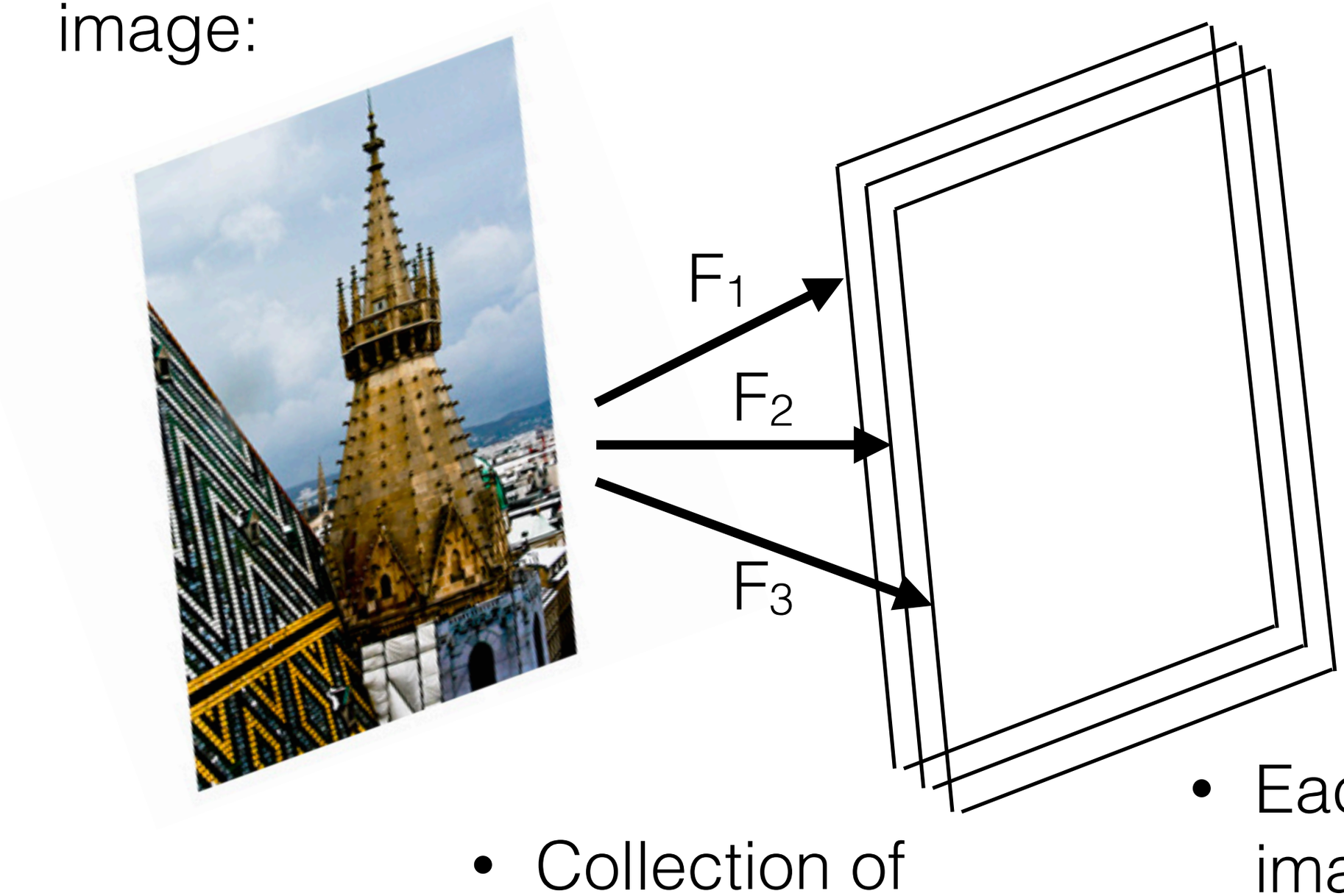


- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

[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”)

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

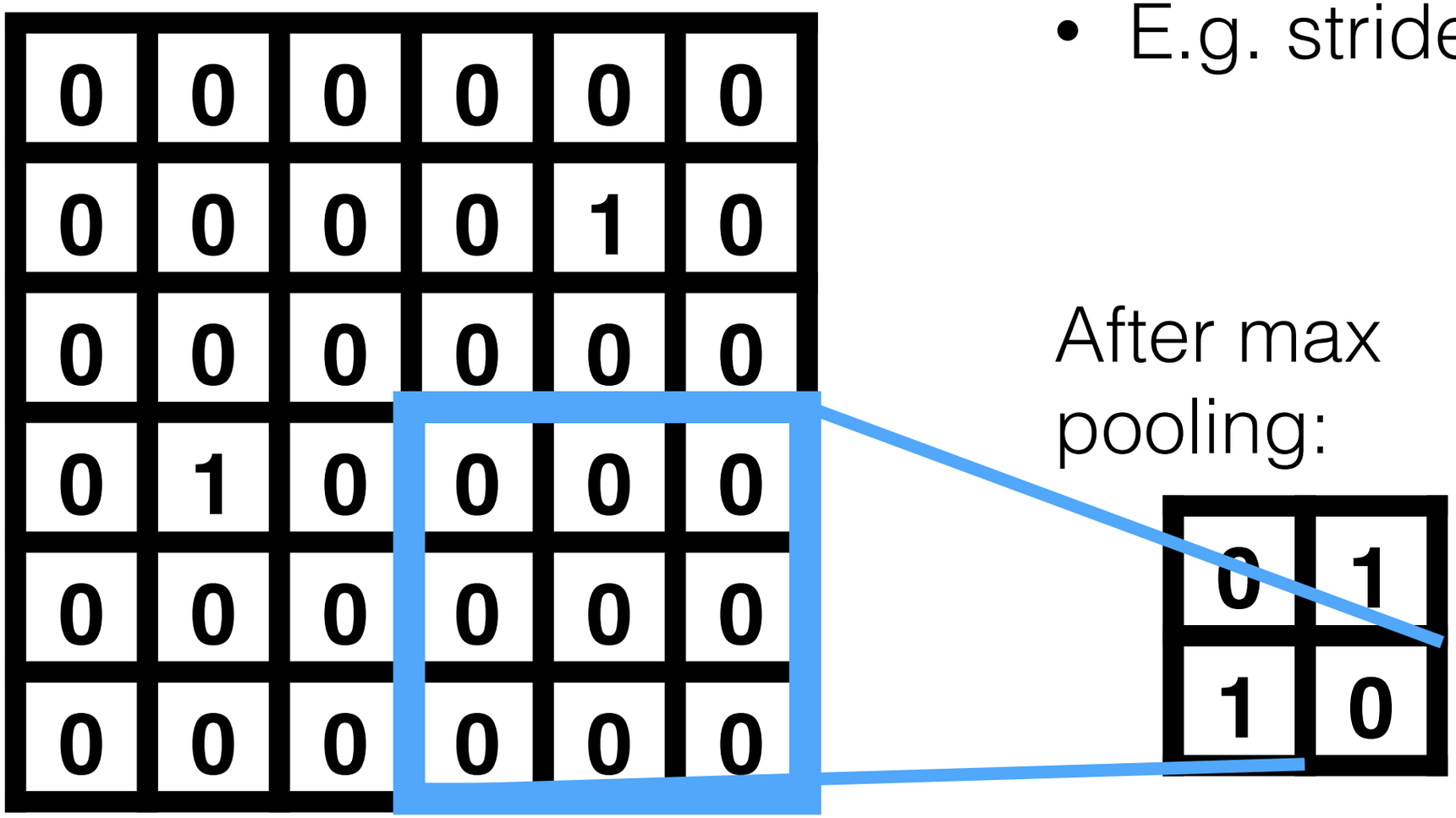
- E.g. size 3x3 (“size 3”)
- E.g. stride 1

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:



Max pooling: returns max of its arguments

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

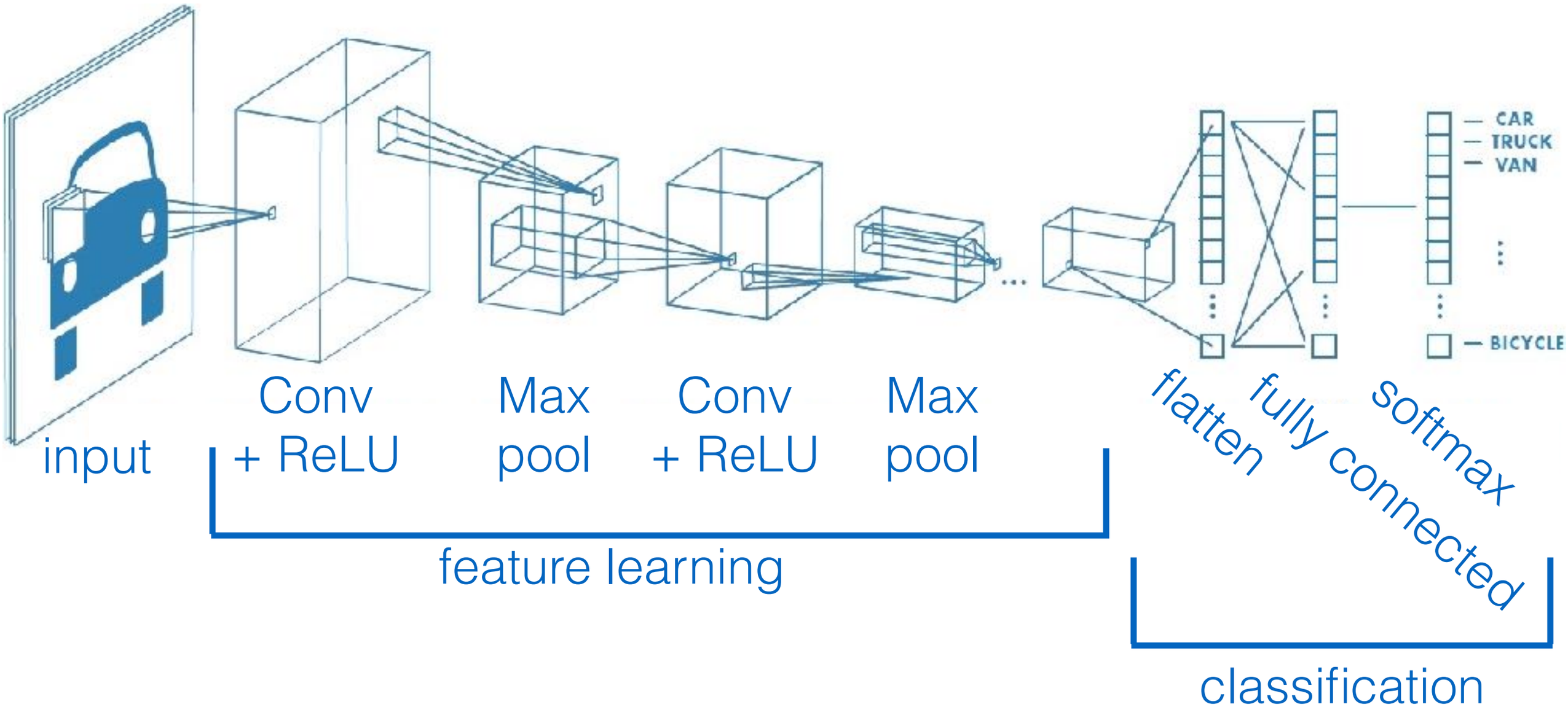
- E.g. size 3x3 (“size 3”)
- E.g. stride 3

After max pooling:

0	1
1	0

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

CNNs: typical 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

CNNs: a taste of backpropagation

Regression. 1 filter: size 3 & padding; $x^{(j)}$ dimension: 5x1

- Forward pass:

$$Z_i^1 = (W^1)^\top X_{[i-1,i,i+1]}$$ Z^1: 5x1

$$A_i^1 = \mathrm{ReLU}(Z_i^1)$$ A^1: 5x1

$$A^2 = (W^2)^\top A^1$$ A^2: 1x1

$$L(A^2, y) = (A^2 - y)^2$$ Loss: 1x1

- Part of the derivative for SGD:

$$\underset{3\text{x}1}{\frac{\partial \text{loss}}{\partial W^1}} = \underset{3\text{x}5}{\frac{\partial Z^1}{\partial W^1}} \cdot \underset{5\text{x}5}{\frac{\partial A^1}{\partial Z^1}} \cdot \underset{5\text{x}1}{\frac{\partial \text{loss}}{\partial A^1}}$$

$$Z_2^1 = W_1^1 X_1 + W_2^1 X_2 + W_3^1 X_3$$

$$\frac{\partial Z^1}{\partial W^1} = \begin{array}{c} \begin{matrix} Z_1^1 & Z_2^1 & Z_3^1 & Z_4^1 & Z_5^1 \end{matrix} \\ \begin{bmatrix} X_0 & X_1 & X_2 & X_3 & X_4 \\ X_1 & X_2 & X_3 & X_4 & X_5 \\ X_2 & X_3 & X_4 & X_5 & X_6 \end{bmatrix} \end{array} \begin{matrix} W_1^1 \\ W_2^1 \\ W_3^1 \end{matrix}$$