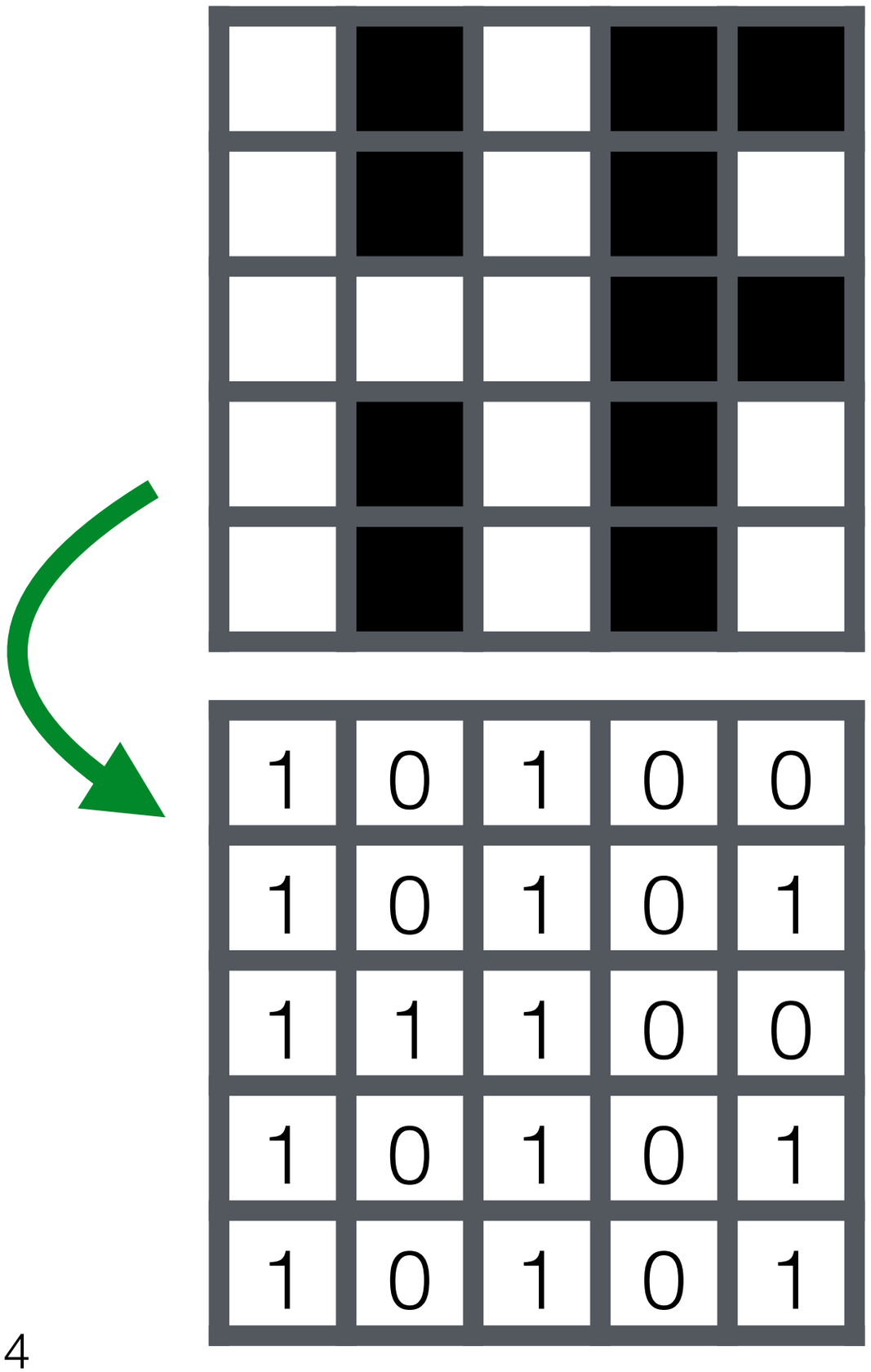
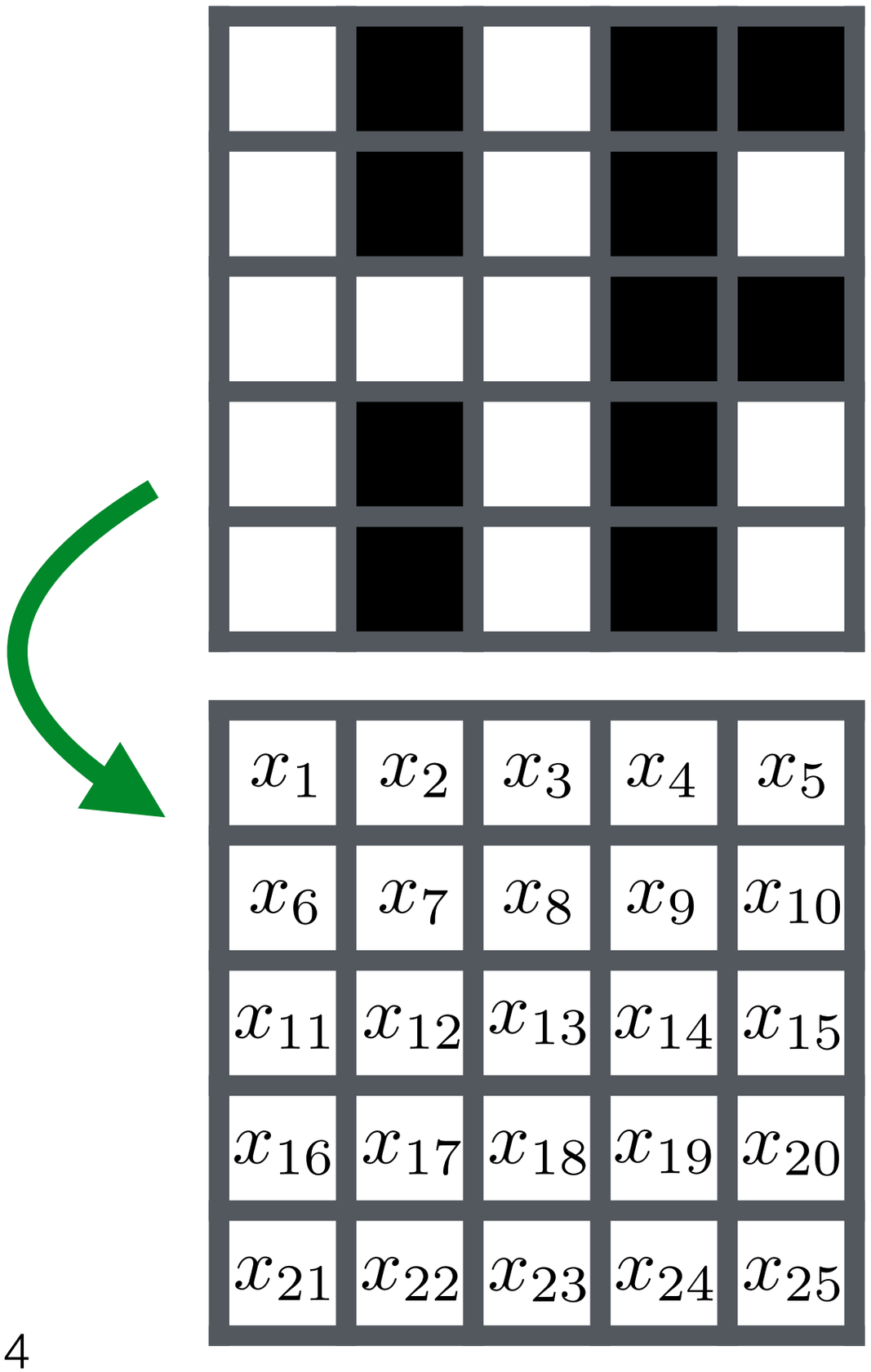
Images



- 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?

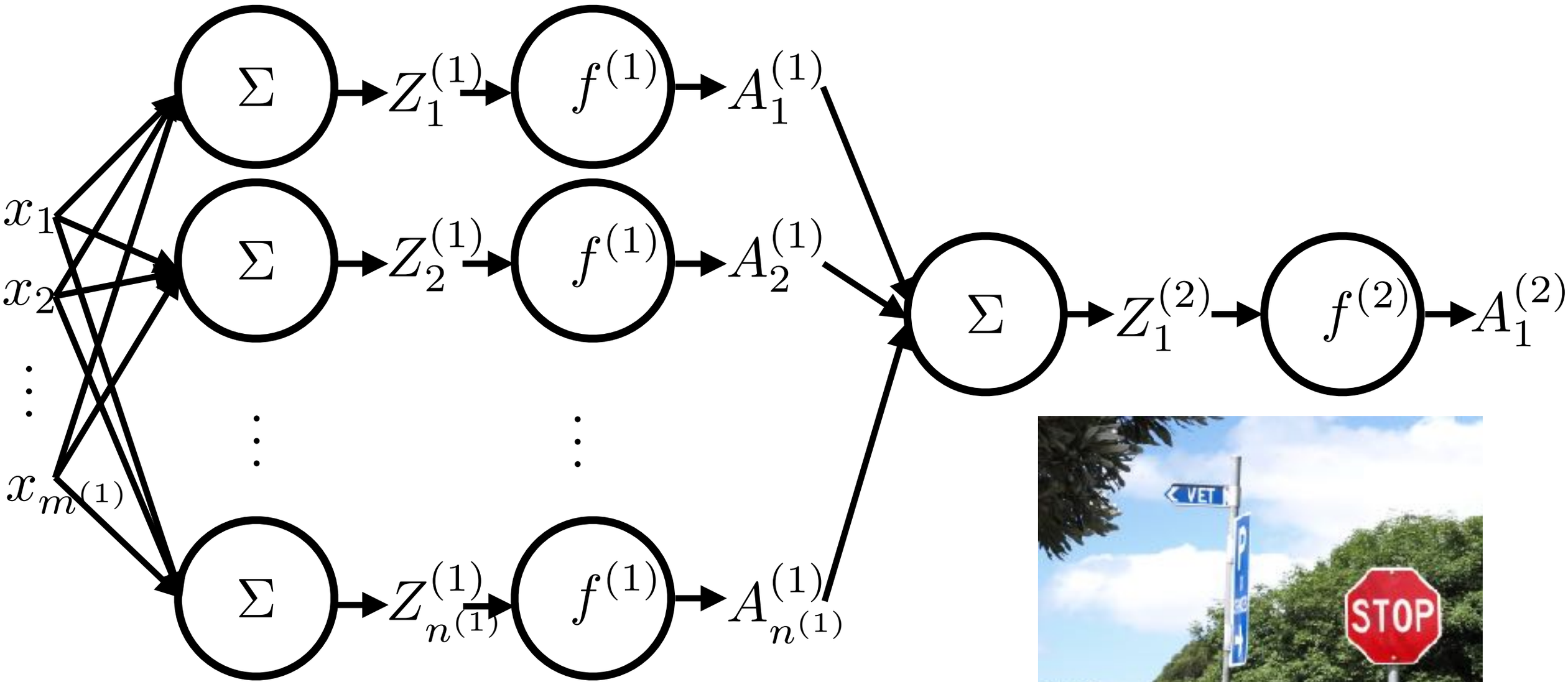
Lecture 8 : Convolutional Neural Networks

★ Encoding data from Images (Slide - 4 - 16,17)

- Since, here we are only dealing with black and white images (not grayscale) there can only be two values for a pixel: 0 & 1 for B&W.
- The simplest way to convert image to data is you convert all the pixels into features. (like given in the slide) & 1 single image essentially becomes 1 single data point.
- Note:- The ordering (or numbering) of the features in the image may or may not be important. It depends on what or how you're gonna process & learn the image.

Previous neural nets in this class

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

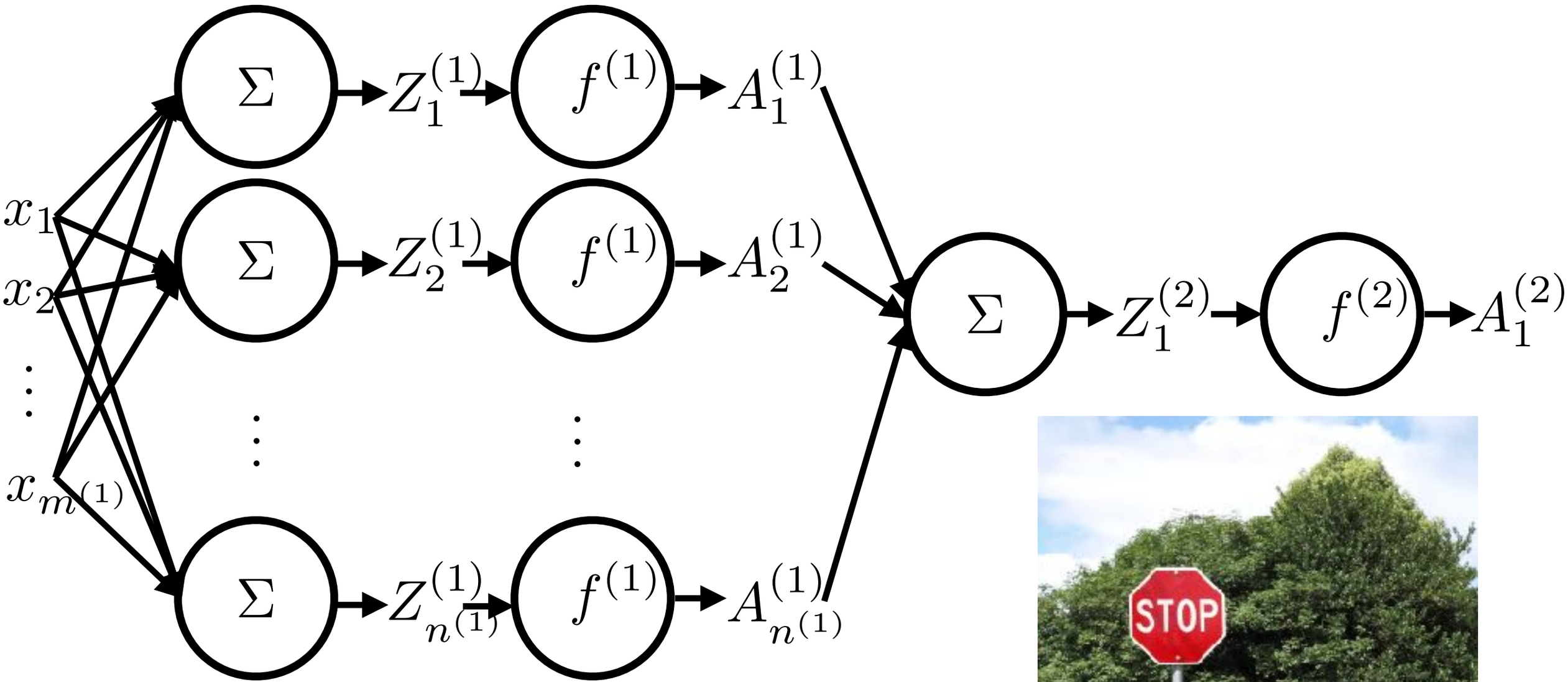


But we know more about images:

- 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

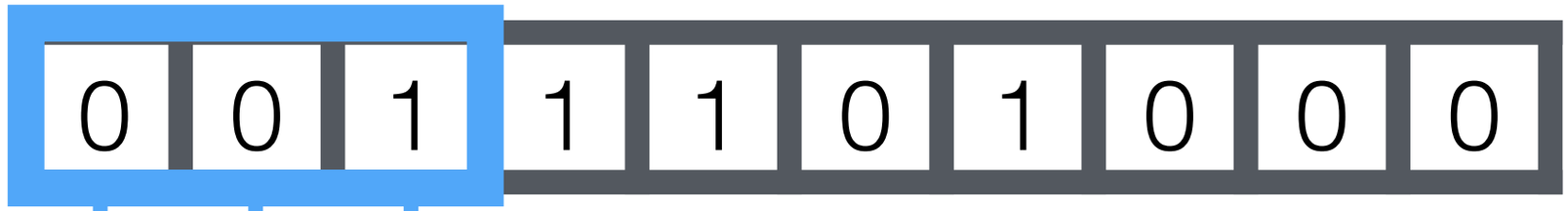
* Previous Neural Nets (& Should we use them on images?) (Slide- 5- 29, 31)

- In the previously seen neural nets, the last layer was the deterministic one & previous layers could be called 'hidden' layers.
- ~~The last layer must be fully connected in any case to get the results.~~

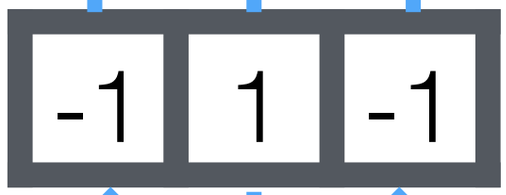
- The 1st layer was fully connected. i.e. all inputs or you could say all features were sent to all the nodes of the layer. We had to learn the parameters of each node on the same data point. (would that work or would that be redundant when dealing with images ?)
- Images are different than regular data. They have some special characteristics like spatial locality & translational invariance.
- Spatial locality :- Take an example, where you wanna detect a stop sign. Now you know that the stop sign pixels aren't gonna be scattered throughout the image. They are gonna be clustered together. This is spatial locality.
- Translational invariance :- The stop sign will be stop sign no ~~matter~~ matter where it is in the image. It's pixels may be clustered in the corners ~~or~~ ~~the~~ of image or centre of image or anywhere else in the image. It would still mean the same thing. (You could say than ~~on~~ the order of only some features may be important in the data.) You wanna be able to locate a stop sign wherever it is in the image.
- So, you could maybe see that fully connected features may cause a problem because they won't be able to encode ~~or~~ these characteristics or ~~detect~~ leverage these characteristics of these images.
- So, you wanna be able to ~~identify~~ pass & analyse & train paramaters on chunks of data. Or you wanna know whats happening in what chunks of data, you use convolutional layer!

Convolutional Layer: 1D example

A 1D image:

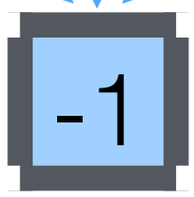


A filter:



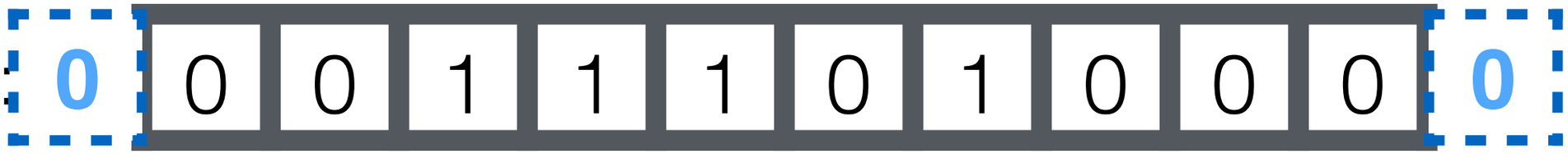
0 * -1 + 0 * 1 + 1 * -1 = -1

After convolution*:

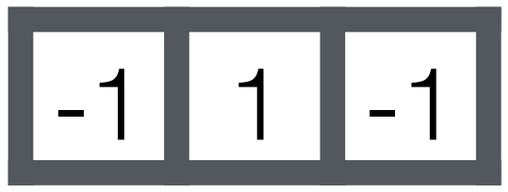


Convolutional Layer: 1D example

A 1D image:

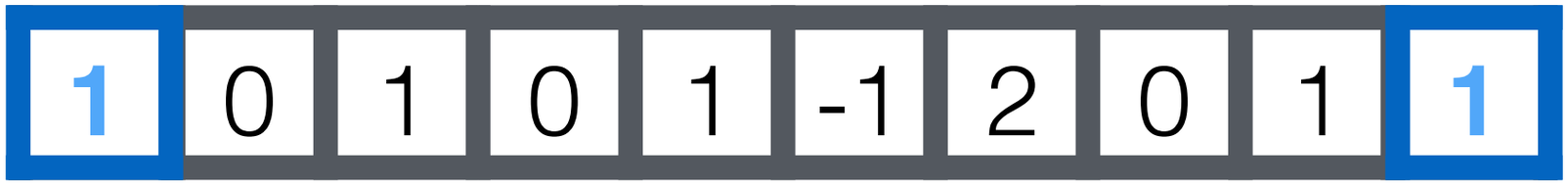


A filter:

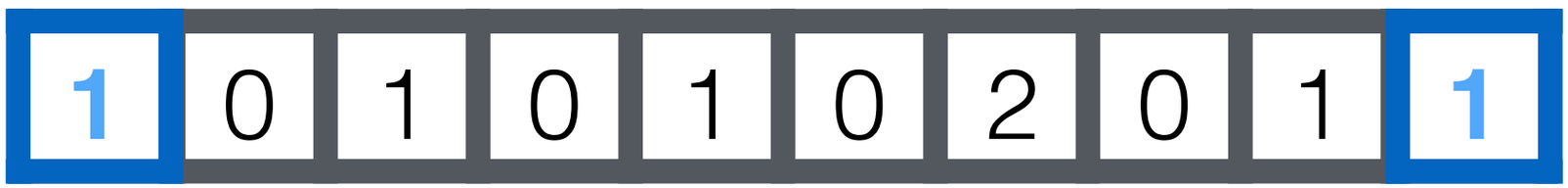


with bias +1

After convolution*:

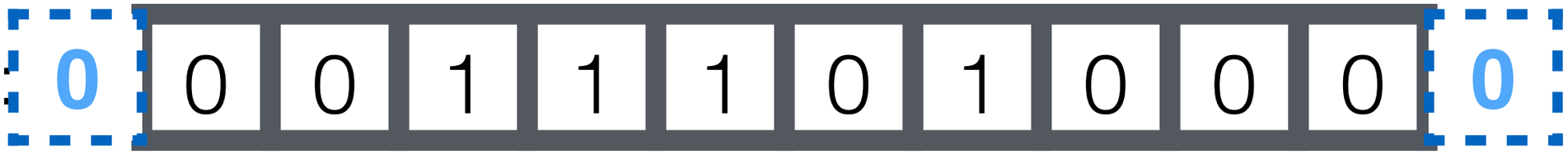


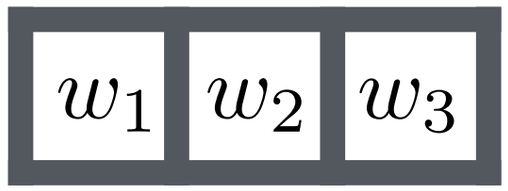
After ReLU:



*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 (1-D example) (Slide - 6 - 48, 90, 100)

- 1-D image may also be called a signal.
  For example:- You have pointed the telescope at a point in the sky & you wanna may be detect a light that indicates / implies a satellite, star, etc. Then the 1-pixel image may be expressed as a variable w.r.t. time & you arrange these pixels in a horizontal 1-D image / signal for a certain range of time.

- You first apply a filter and the result of the filter is the ~~convolutional layer~~ new features layer. As you see, the filter analyses chunks of data or sequences of data & turns them into new features.
  For example, if you apply a (-1, 1, -1) filter on a binary sequence, you will get the degree of isolation of 1s & 0s as new features. For ex, a sequence of 0, 1, 0 will give 1 which indicates the 1 'is lonely' & sequence 1, 0, 1 will give -2 which indicates 0 is 'lonely' in the sequence.

- But if you notice, in a filter like this, we are focusing on the middle number of a sequence, ~~of 3~~ 3 pixels. So, this will tell about the degree of isolation of the second pixel in every sequence. But, when you apply this on a signal, you won't get anything on the 1st & last pixel of the signal because they won't ever be the 2nd pixel of a sequence.

- And, if you have a signal of length m & you apply a filter of length n on it, the resulting new features vector will be only of size $m-n+1$.

- To solve the problems given above, (i.e. reduced size of feature vector & loss of data on 1st & last pixels) we introduce padding.

- You can also add bias to the dot product by literally adding a value to ~~it in the~~ the dot product. The contents of the filter are supposed to be chosen by you depending on how you wanna analyze the raw data. The contents of the filter ~~are~~ and bias are analogous to θ & θ_0 and are called 'weights'

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:

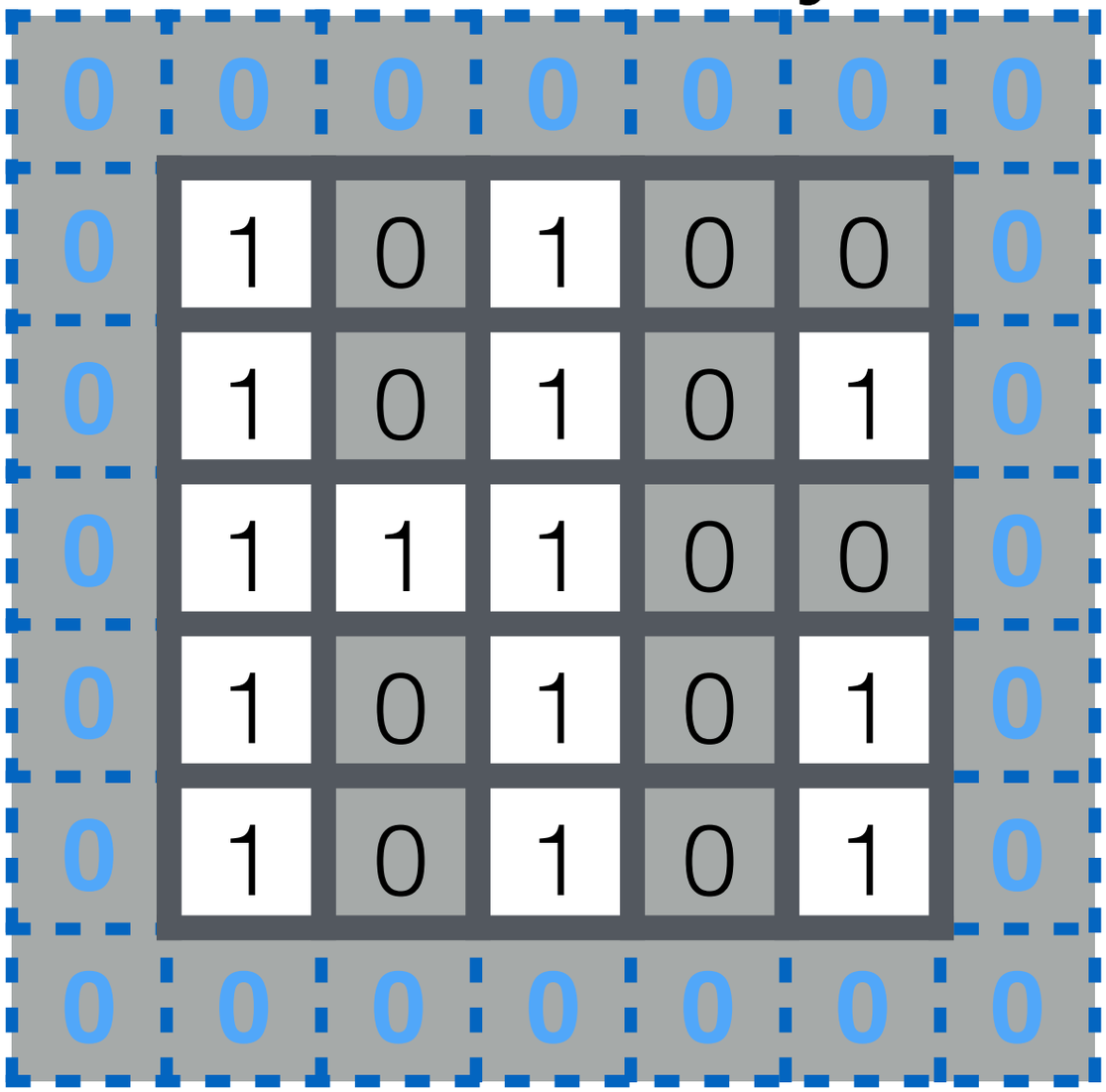
-1	-1	-1
-1	1	-1
-1	-1	-1

After convolution:

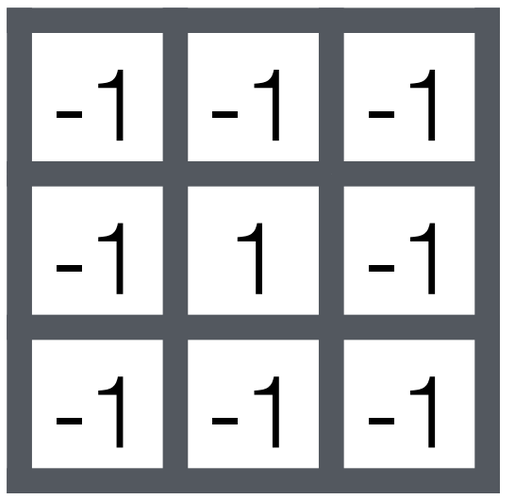
0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

Convolutional Layer: 2D example

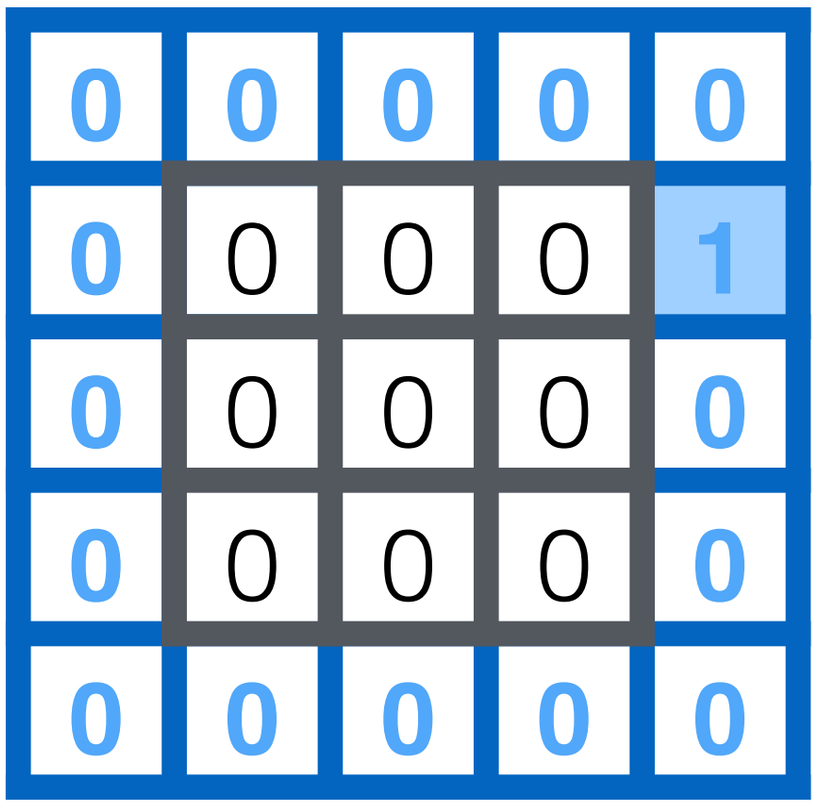
A 2D image:



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

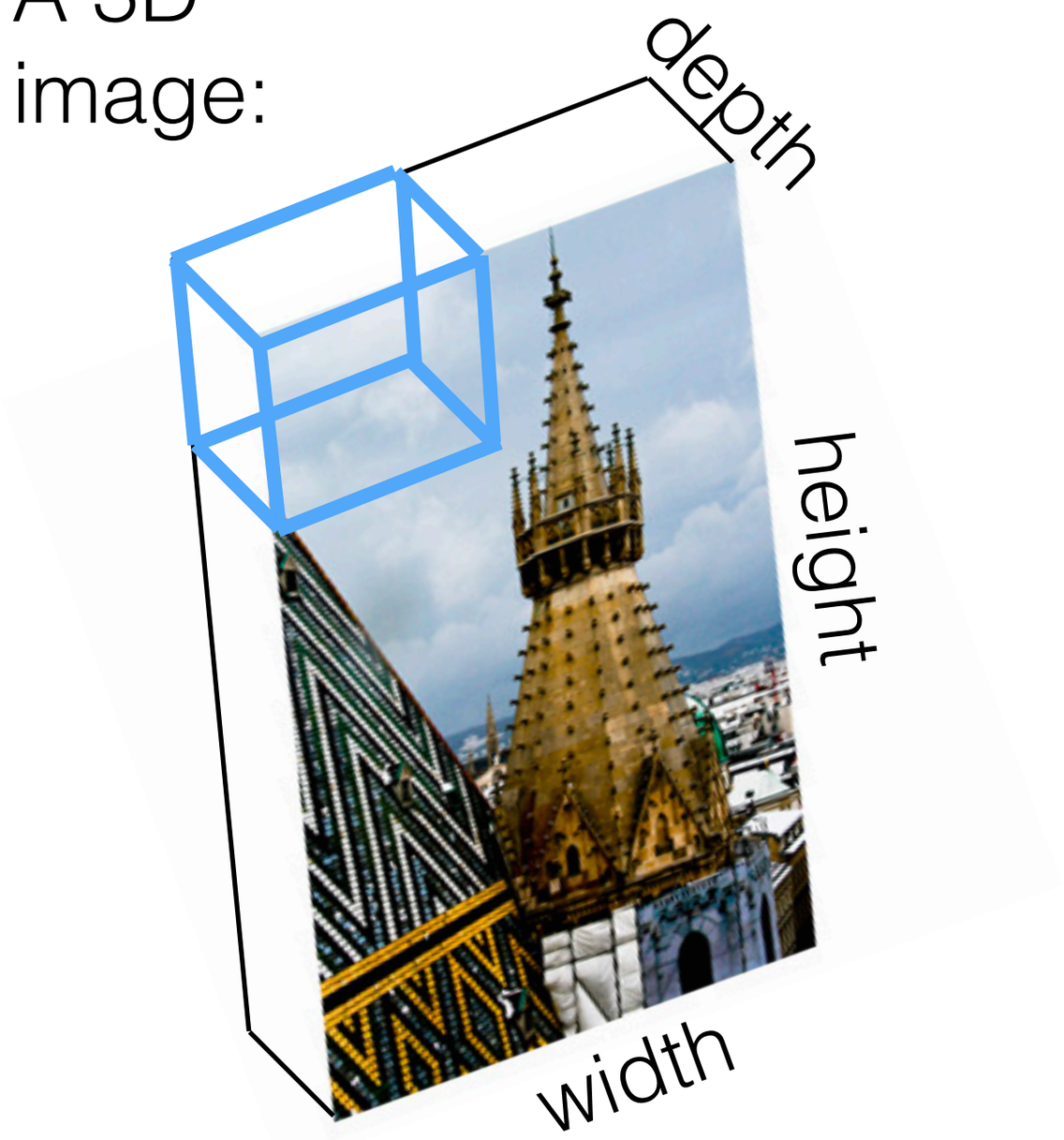
with bias b

★ Convolutional Layer (2-D example) (Slide - 138, 143, 154)

- In 2-D & 3-D and ~~in~~ any dimensions, you do element wise multiplication of filter & sequence matrices & not matrix multiplication. You take sum of resulting elements (i.e. after multiplication)
- In this 2-D example too, for a filter of [[-1,-1,-1], [-1, 1,-1], [-1,-1,-1]] it gives a 'lonely 1' from the original image.

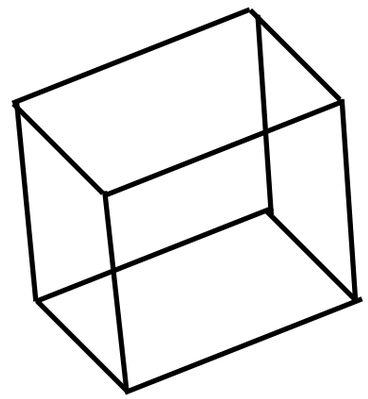
Convolutional Layer: 3D example

A 3D image:



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

A filter:

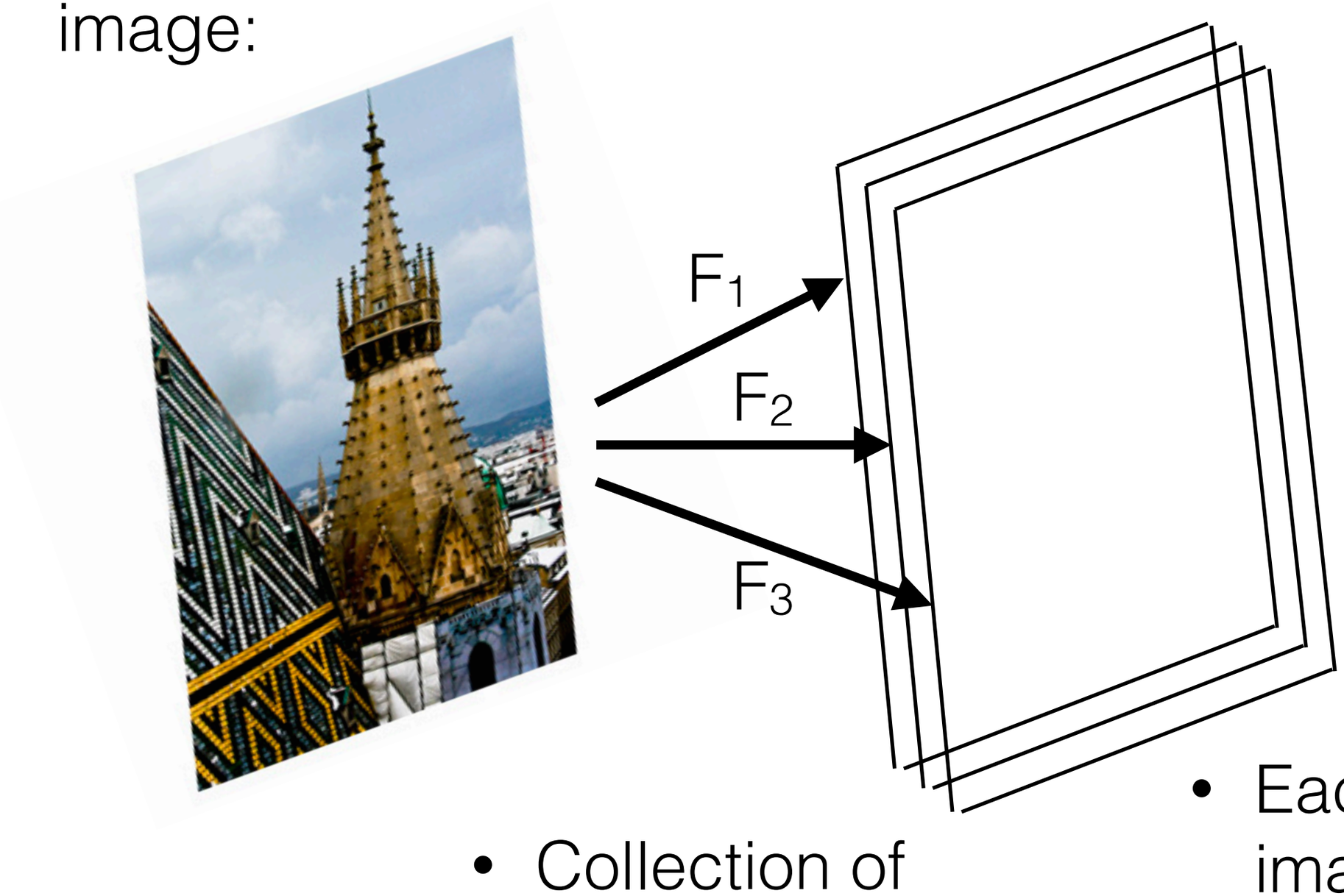


- Tensor: generalization of a matrix
  - E.g. 1D: vector, 2D: matrix

[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*

★ Convolutional layer: (3-D example)

(Slide - 162, 172,)

- Till now, we only looked at 2-D images with only black and white colours (~~not gray~~ not to be mistaken as grayscale)
- 3-D images are just 2-D images, but when representing them as data, we represent their colour as 3rd dimension.
- For example, a single pixel may have 3 values: Red, Green & Blue values. Image may be represented as a 3 x height x width image as data.

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

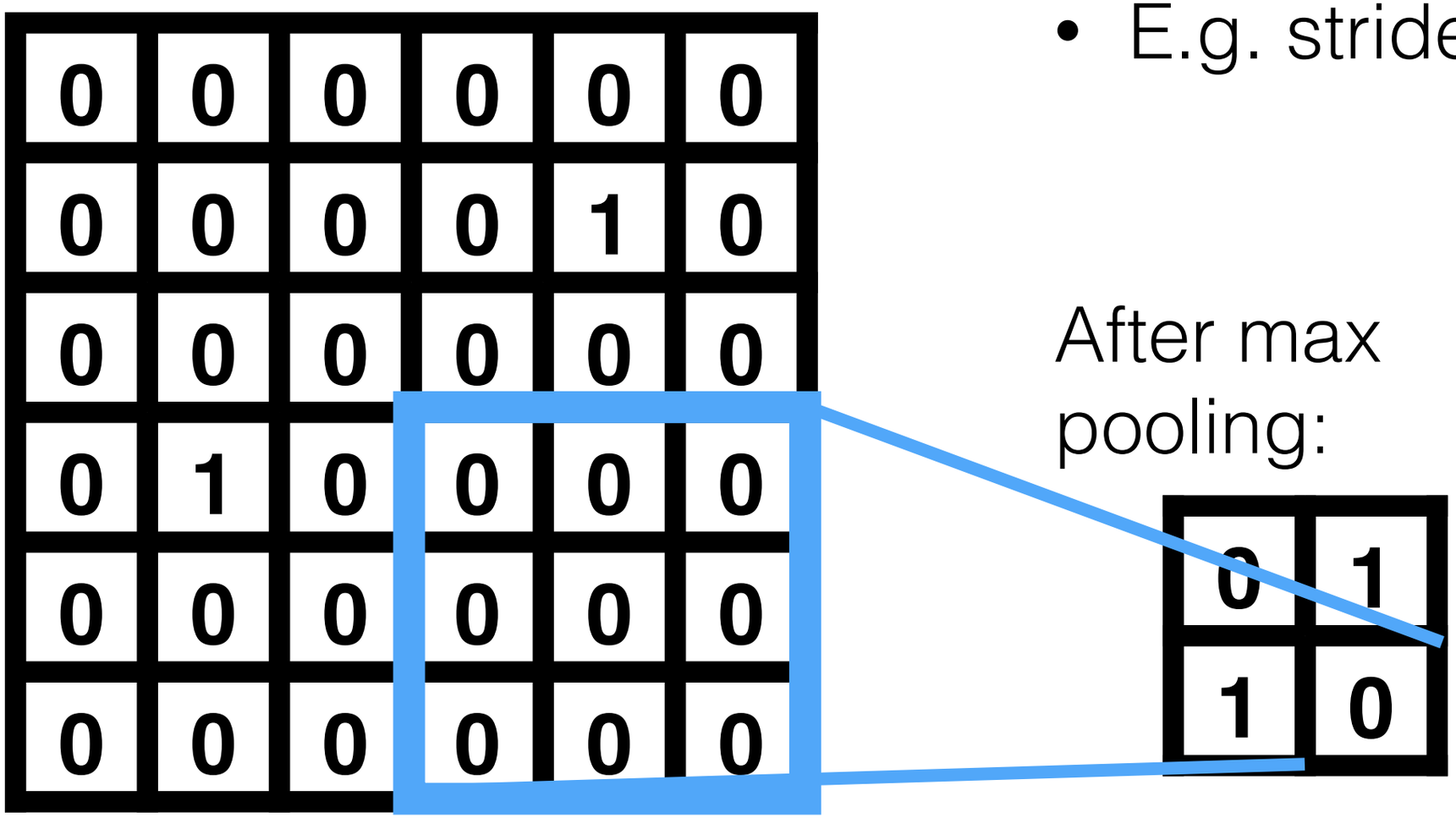
- E.g. size 3x3 (“size 3”)
- E.g. stride 1

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

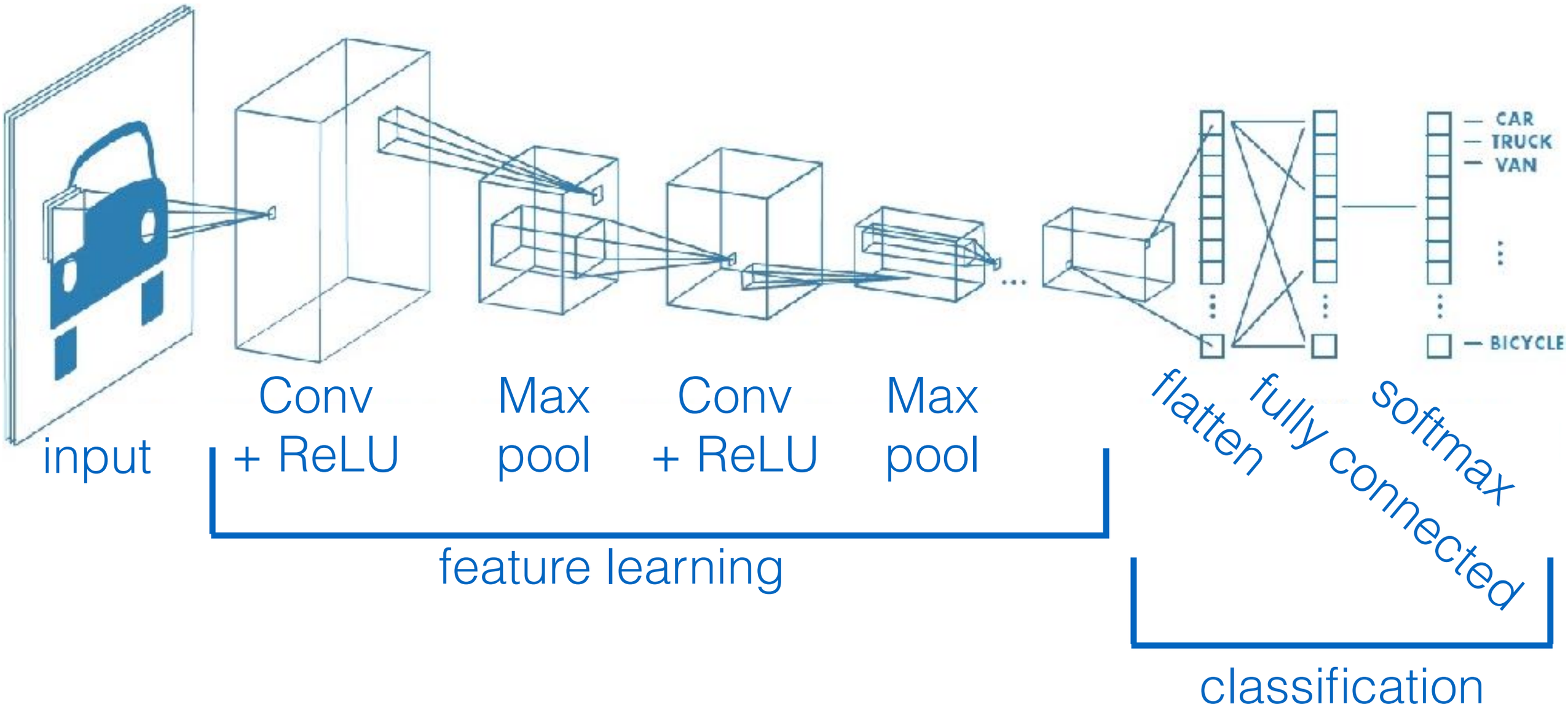
✱ Max pooling layer (Slide - 187, 189, 214, 217)

- Max pooling layer is also like a filter without the weights & dot product & bias.
- It basically tells, "do we detect something & what?" after convolutional layer / filter.
- It returns max of its arguments.

- If you're using a max pool of size 3 (i.e. 3×3) with stride 1, you basically skip 1 column of pixel while moving horizontally & 1 row of pixels when moving vertically. For stride 3, you ~~get~~ skip 3 columns of pixels horizontally moving & 3 ~~3~~ rows of pixels moving vertically.

- Note:- You can use different strides (strides other than 1) for filters too.

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:

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

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

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

★CNNs: a taste of Backpropagation (Slide-261)

- Consider an example where you want to do regression for a 5×1 image/signal & 3×1 ~~paddi~~ filter & appropriate padding.

- The ~~filter~~ pre-activation vector $Z^{(1)}$ given obtained as a result of filter $W^{(1)}$ applied on input X. $Z^{(1)}$ will be of size 5×1

  & $Z_i^{(1)} = (W^{(1)})^T \cdot X_{[i-1,\, i,\, i+1]}$

- We use the activation function ReLU on $Z^{(1)}$ to get the activation $A^{(1)}$
- Now, for the output of the final regression layer, $A^{(2)} = (W^{(2)})^T \cdot A^{(1)}$
- We use squared error loss, where the loss is the function of $A^{(2)}$
- So, to use SGD & calculating gradient, we back propagate in the chain rule to get the gradient/derivative.

Note:- The matrix derivatives is tricky part & learn linear algebra to understand it.