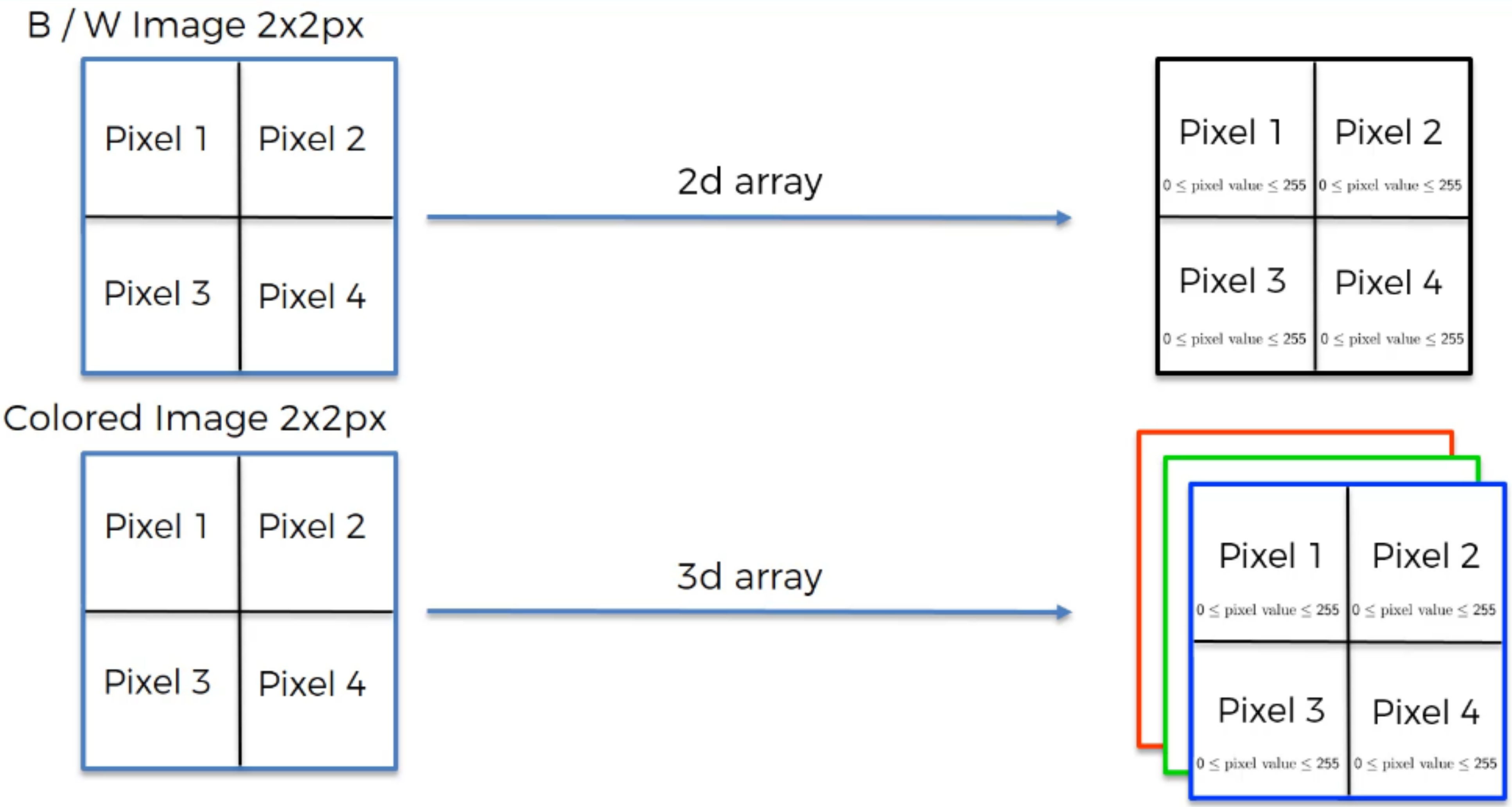
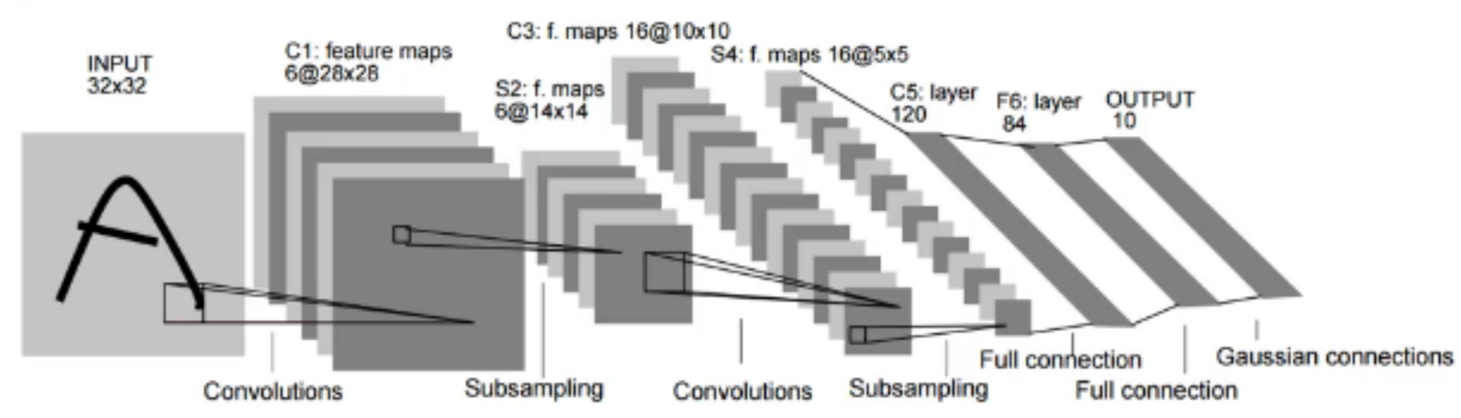
Convolutional Neural Networks



B / W Image 2x2px
Pixel 1
Pixel 2
Pixel 3
Pixel 4
2d array
Pixel 1
$0 \leq$ pixel value ≤ 255
Pixel 2
$0 \leq$ pixel value ≤ 255
Pixel 3
$0 \leq$ pixel value ≤ 255
Pixel 4
$0 \leq$ pixel value ≤ 255
Colored Image 2x2px
Pixel 1
Pixel 2
Pixel 3
Pixel 4
3d array
Pixel 1
$0 \leq$ pixel value ≤ 255
Pixel 2
$0 \leq$ pixel value ≤ 255
Pixel 3
$0 \leq$ pixel value ≤ 255
Pixel 4
$0 \leq$ pixel value ≤ 255

CNN Steps

- Convolution (with ReLU)
- Max Pooling
- Flattening
- Full Connection



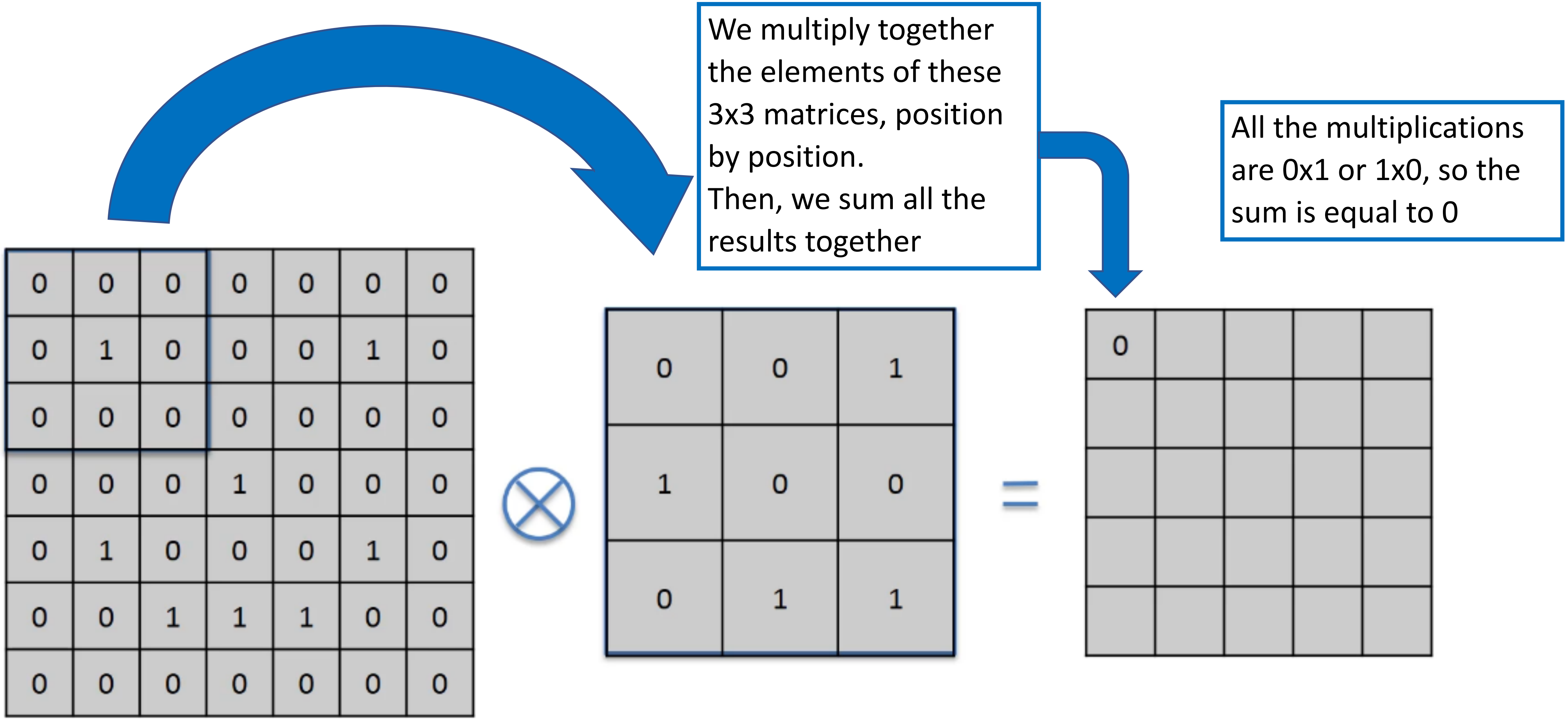
Step 1 - Convolution

- This is the transformation in mathematical terms

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

- The verb to use is “to convolve”

  That translated from Math is ‘to combine’



<u>Input Image</u>

Simplified as just 1s and 0s per pixel

<u>Feature Detector</u>

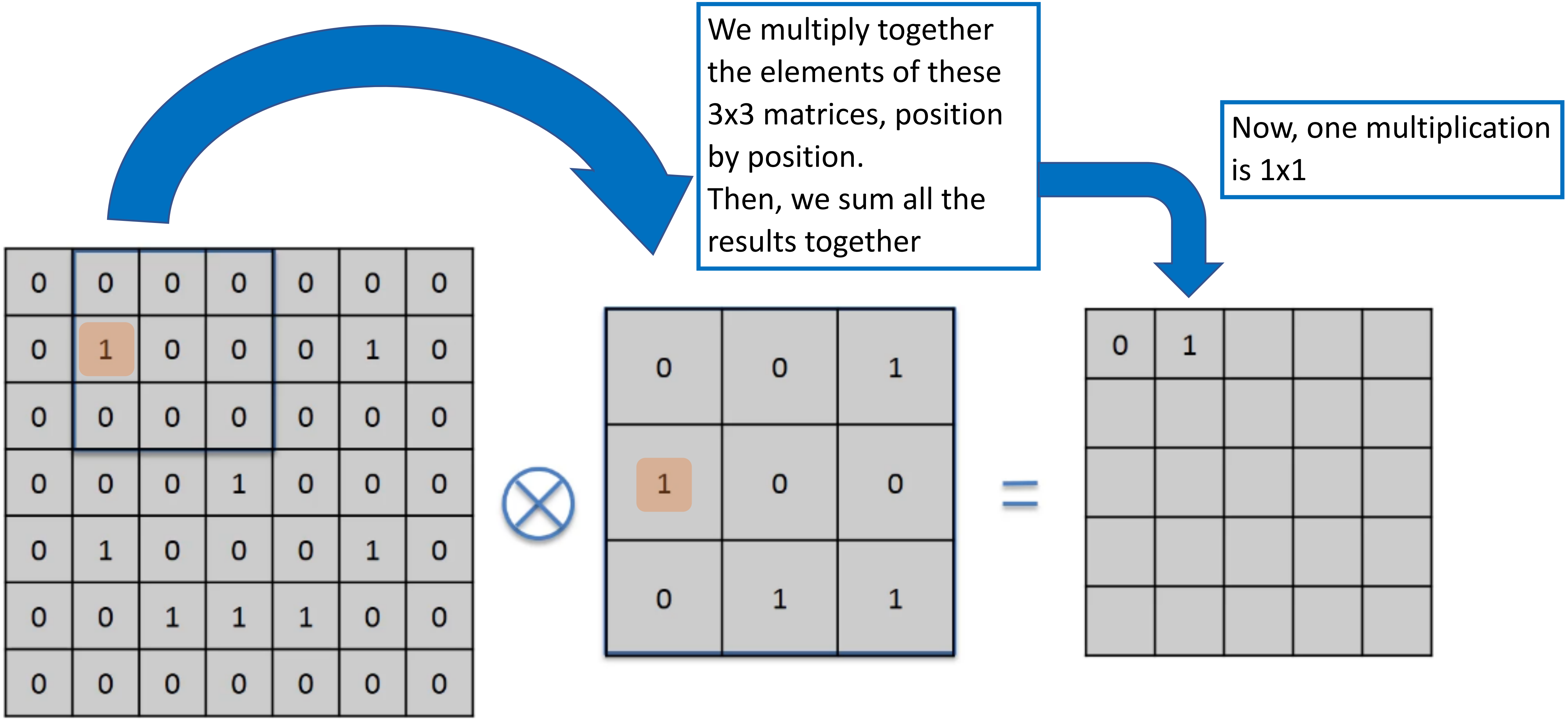
AKA Filter

Usually, 3x3 but it could be larger

It can contain negative values too

<u>Feature Map</u>

Simplified as just 1s and 0s per pixel



Input Image

Simplified as just 1s and 0s per pixel

Feature Detector

AKA Filter

Usually, 3x3 but it could be larger

It can contain negative values too

Feature Map

Simplified as just 1s and 0s per pixel

The movement of this matrix is called **stride**

- Here we have a stride of 1 pixel

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

$\otimes$

0	0	1
1	0	0
0	1	1

$=$

0	1			

Input Image

Simplified as just 1s and 0s per pixel

Feature Detector

AKA Filter

Usually, 3x3 but it could be larger

It can contain negative values too

Feature Map

Simplified as just 1s and 0s per pixel

After 10 strides

- Best values for the stride is by 1 or 2 pixels

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

$\otimes$

0	0	1
1	0	0
0	1	1

=

0	1	0	0	0
0	1	1	1	

<u>Input Image</u>

Simplified as just 1s and 0s per pixel

<u>Feature Detector</u>

AKA Filter

Usually, 3x3 but it could be larger

It can contain negative values too

<u>Feature Map</u>

Simplified as just 1s and 0s per pixel

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

$\otimes$

0	0	1
1	0	0
0	1	1

$=$

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

<u>Input Image</u>

Simplified as just 1s and 0s per pixel

<u>Feature Detector</u>

AKA Filter

Usually, 3x3 but it could be larger

It can contain negative values too

<u>Feature Map</u>

Simplified as just 1s and 0s per pixel

Step 1 - Convolution

- What is the **Feature Map**? What did we obtain?

1. We reduced the size of the original matrix

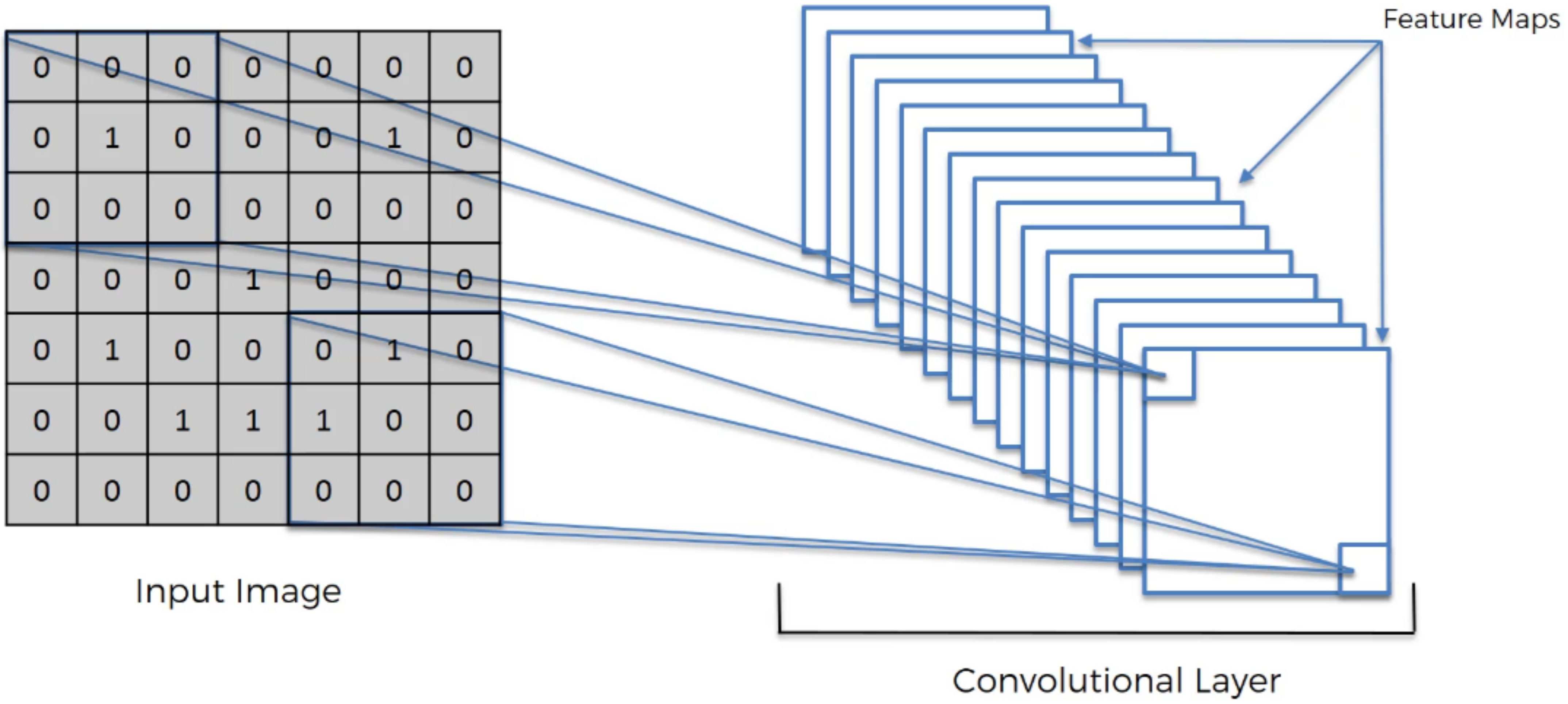
   The more, the larger is the **stride**

2. The higher are the numbers, the more precisely the detected feature appears in the original image

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Note that it still preserves the spatial relationships between pixels

- We don’t lose the pattern



- The network will be trained to recognize a series of Features that are then stored in a collection of Feature Maps

    This is the **Convolutional Layer**

Step 1 - Convolution

- Another example of Feature Detector (Horizontal edge)

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Step 1 - Convolution

- After creating the set of Feature Maps, we need to add some non-linearity
- We apply the **Rectifier** activation function (**ReLU**)

➢All the negative values become positive values

- This helps reducing the graduality of the color palette.

Example: In a picture, colors go from bright to dark gradually.

If we remove the negatives, it will change more abruptly. Less linearly!

Step 2 - Pooling

- We need to ensure that our neural network has a property called **spatial invariance**

  Basically, that it doesn't care where the features are and doesn't care if the features are a bit tilted, different, relatively closer or farer apart, and so on.

Step 2 - Pooling

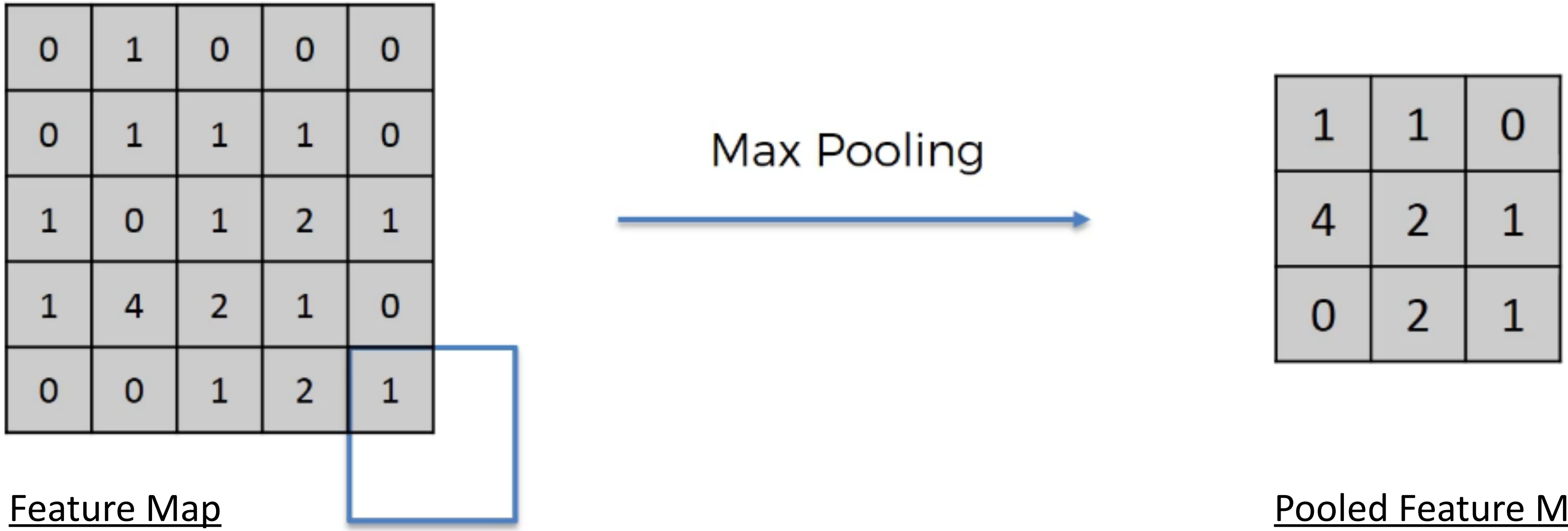
<u>Feature Map</u>

- We pick **the maximum value** in a 2x2 sub-square (the size can vary)
- Stride is 2 here, but it can vary
- If at the end of the row, we just continue

<u>Pooled Feature Map</u>

- Each position contains **the maximum value** found in the sub-squares

Step 2 - Pooling



Feature Map

- We pick **the maximum value** in a 2x2 sub-square (the size can vary)
- Stride is 2 here, but it can vary
- If at the end of the row, we just continue

Pooled Feature Map

- Each position contains **the maximum value** found in the sub-squares

Step 2 - Pooling

Feature Map

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

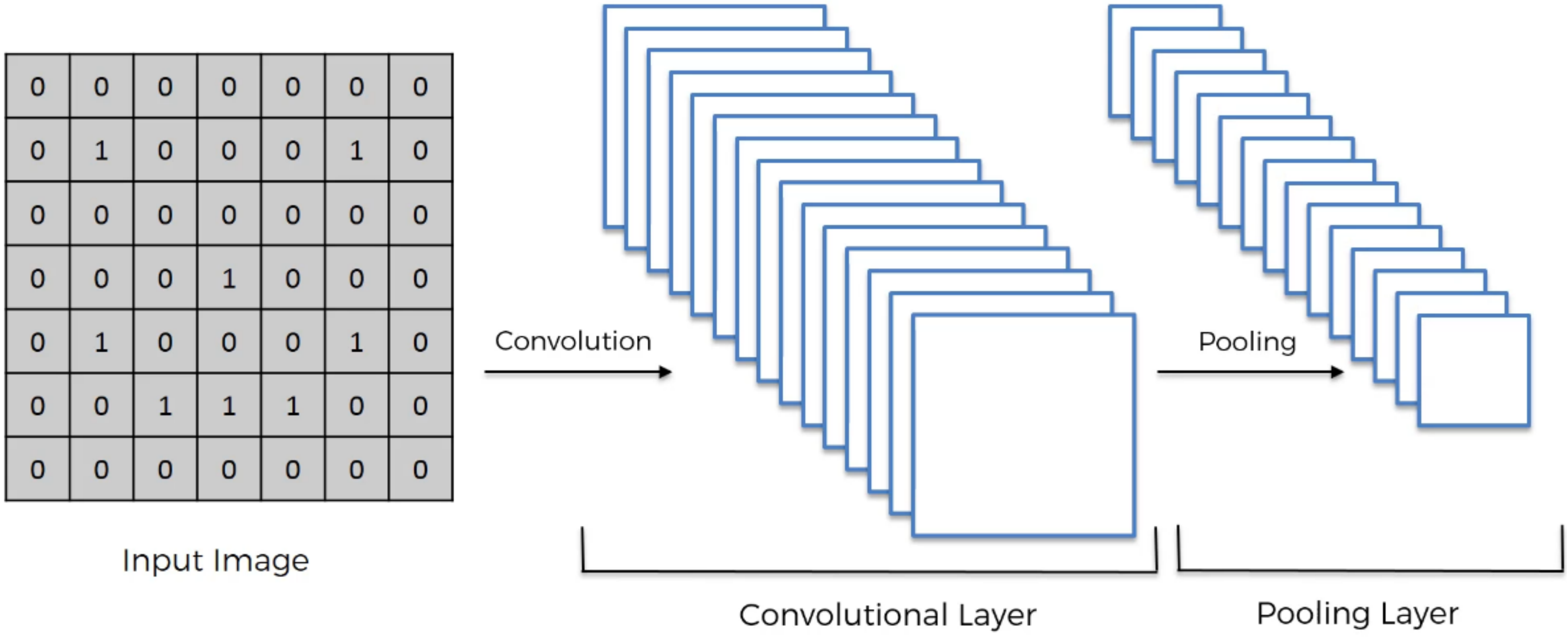
Max Pooling $\longrightarrow$

Pooled Feature Map

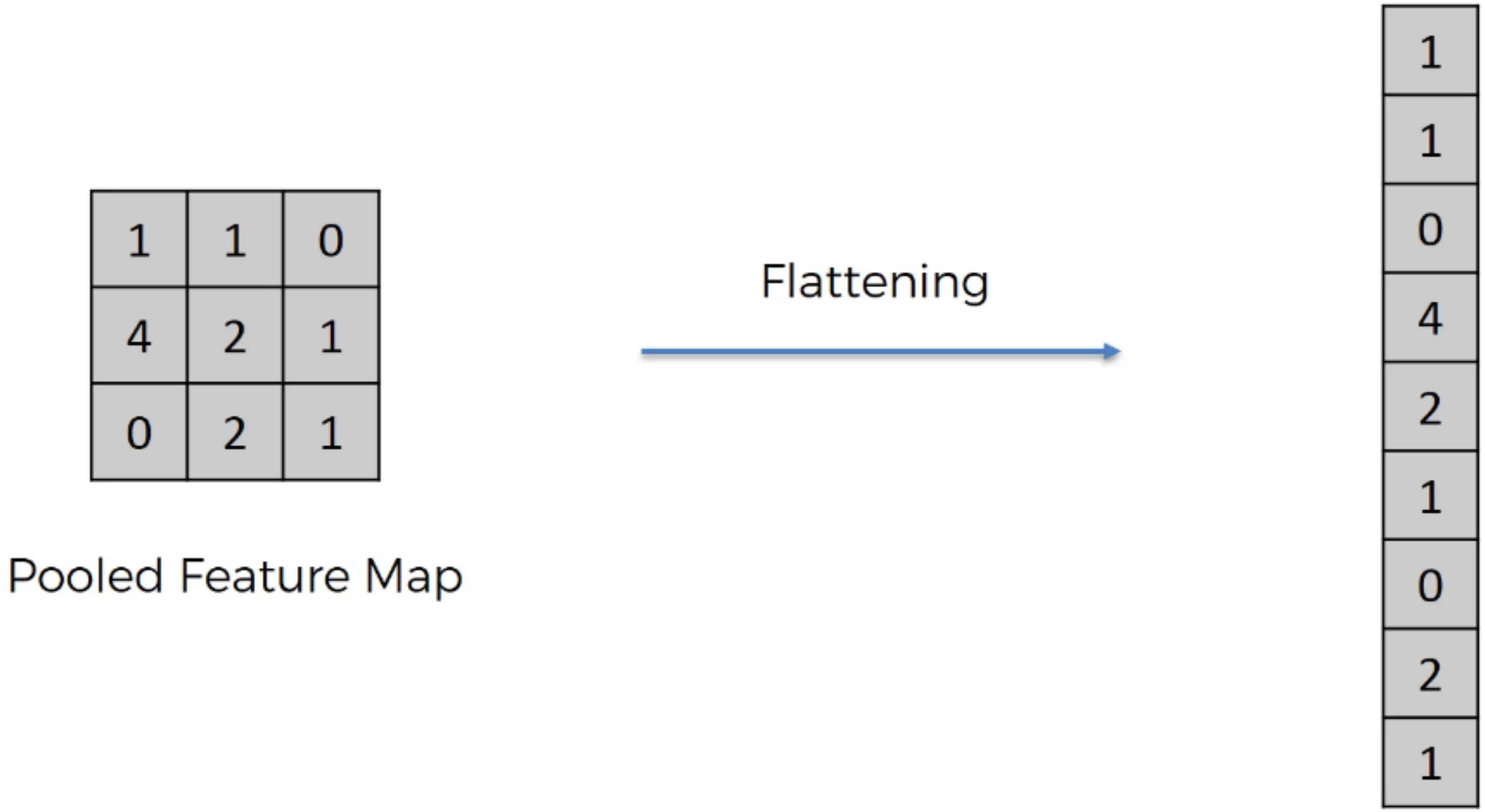
1	1	0
4	2	1
0	2	1

- In the **Pooled Feature Map** we pick the maximum value in a small area (the sub-square)
  - If in the Feature Map, the maximum value is moved in a different position, it will still be recognized and passed to the Pooled Feature Map
- We are also reducing the size of the Map, helping the work of the final layer
- Also, it reduces overfitting, because we are not training on the actual training data anymore

Convolution + Pooling

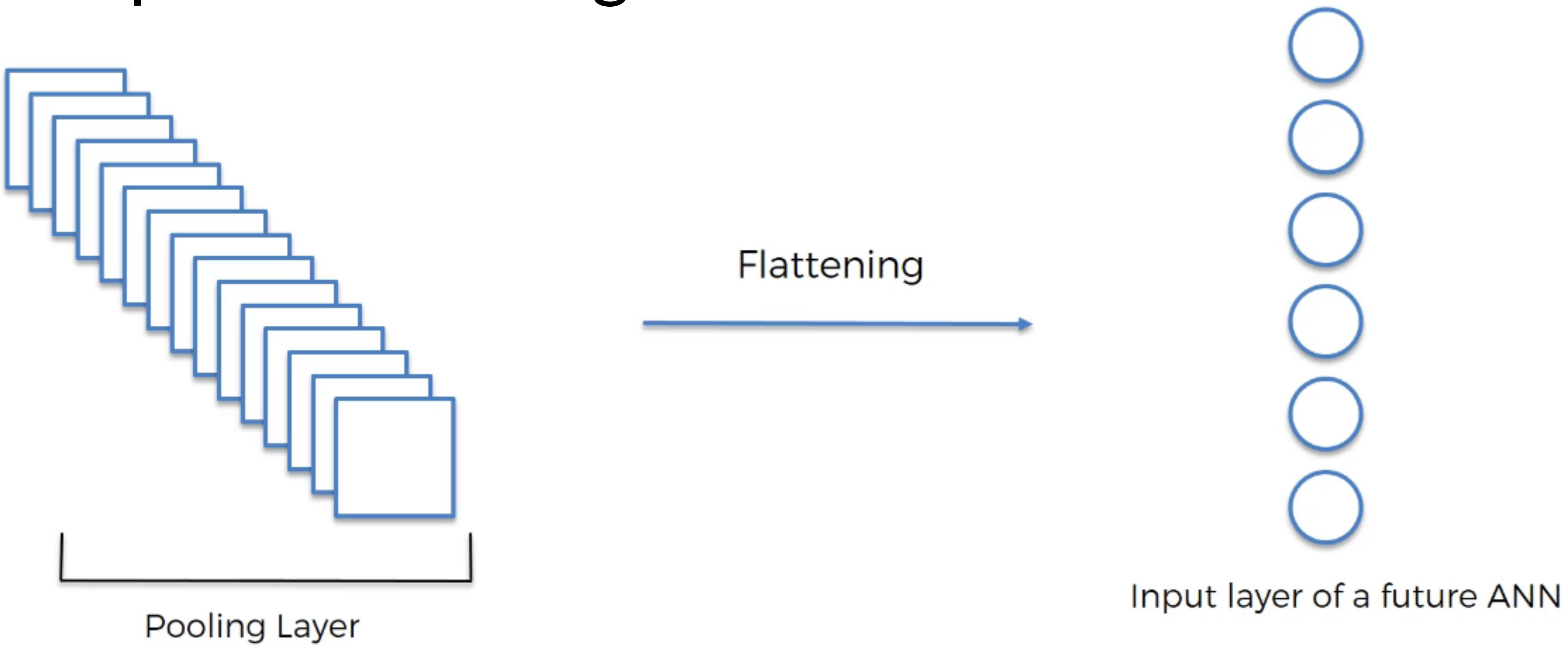


Step 3 - Flattening



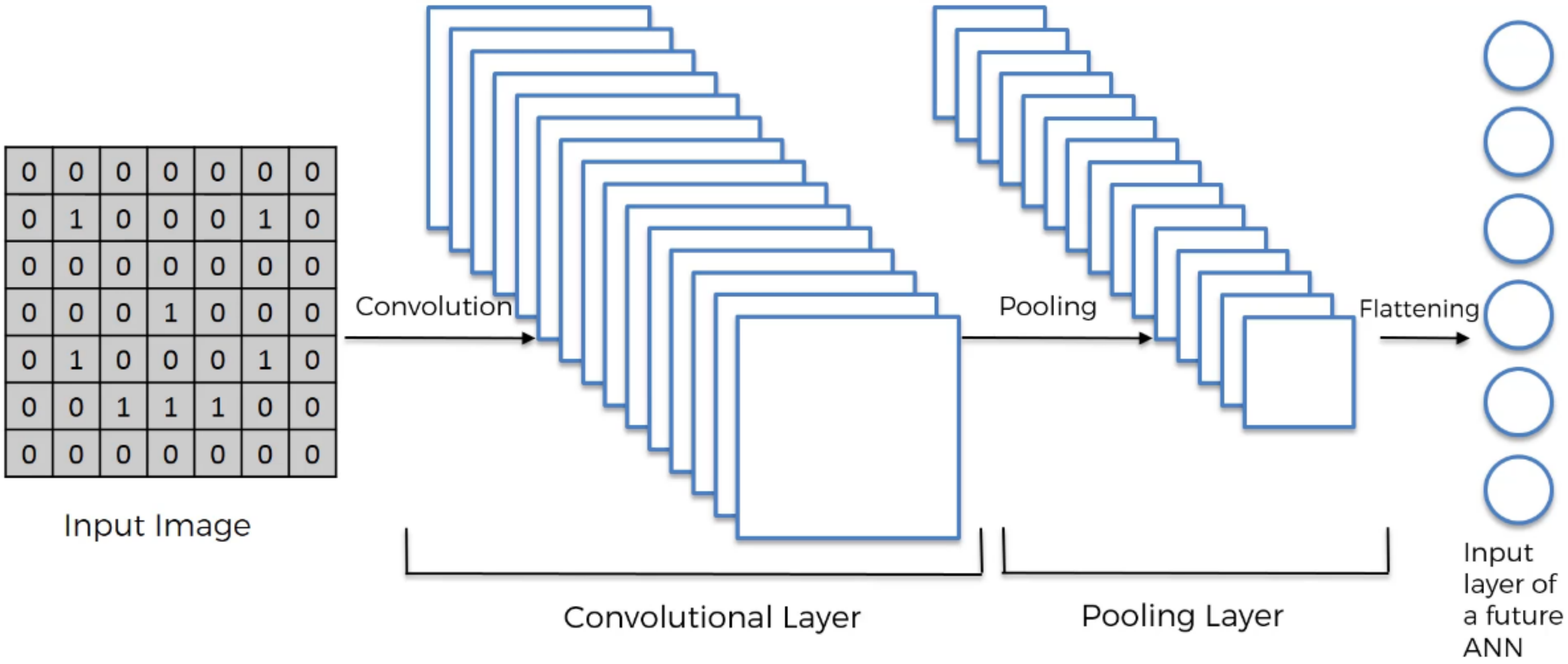
- Simply, we convert the matrices in a vector (by rows)

Step 3 - Flattening



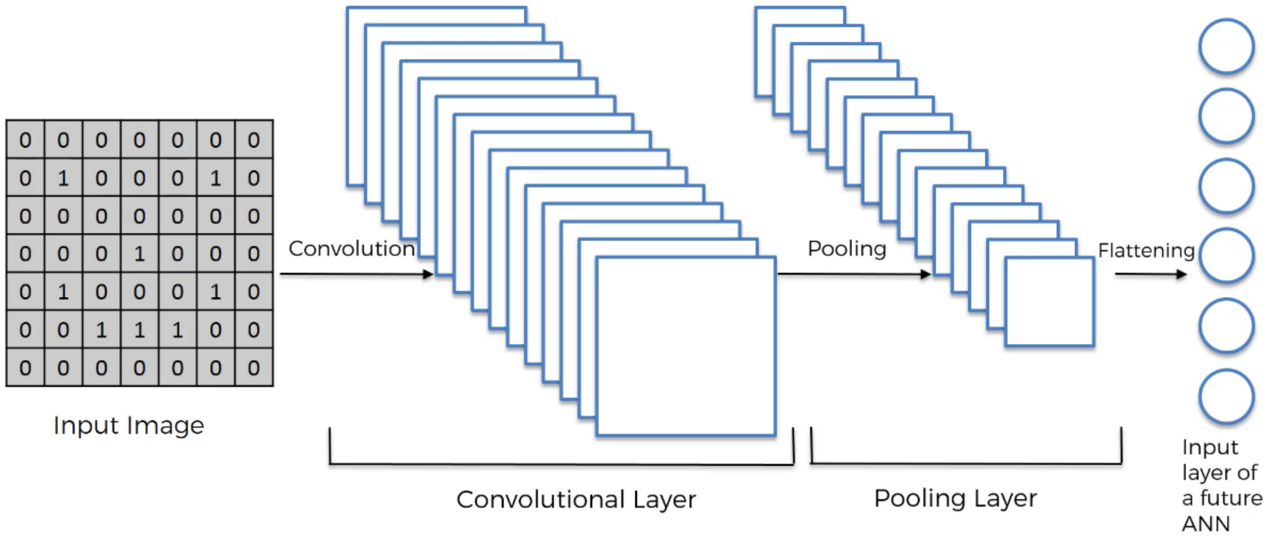
- Simply, we convert the matrices in a vector (by rows)

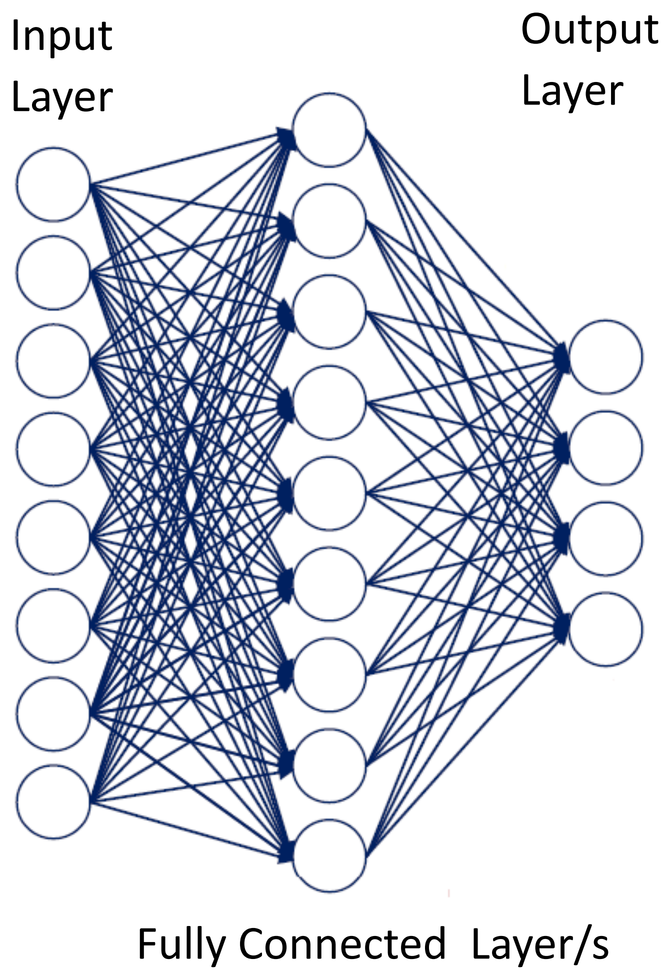
Convolution + Pooling + Flattening



Step 4 – Full Connection

- We finally attach the ANN



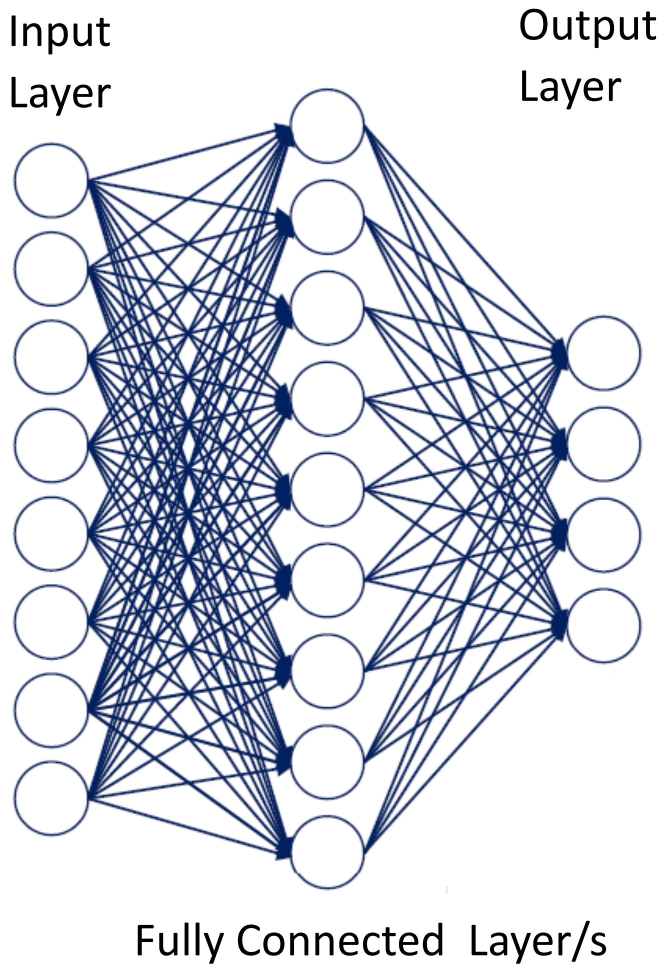


Note that in CNNs, the hidden layers are called "Fully Connected Layers"

Step 4 – Full Connection

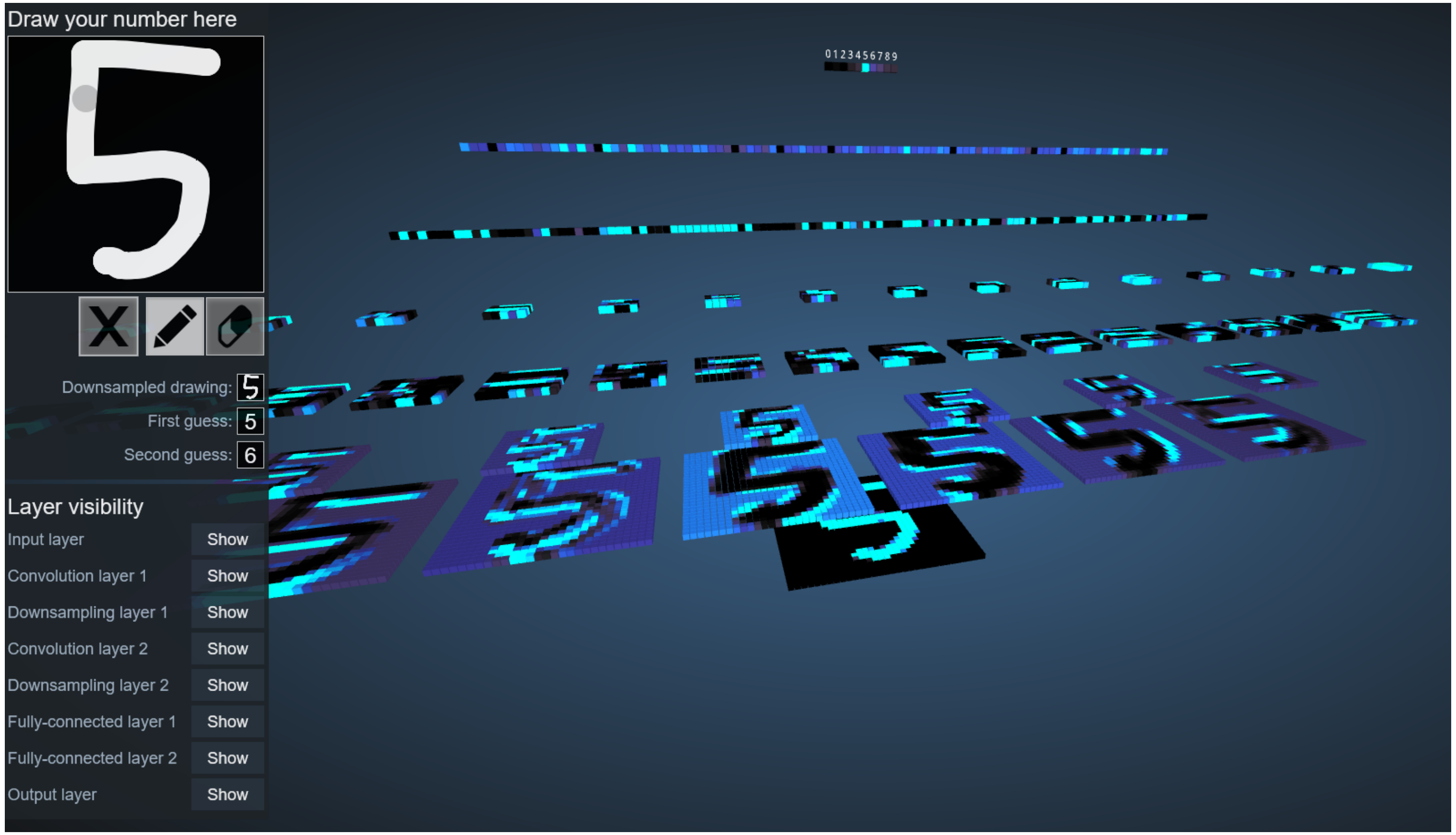
- ➢ In this network, though, the backpropagation will not simply identify the weights to update, but also the "features"

- In fact, the Feature Detectors are also modified based on the error (via gradient descent)



- Check this out:

  https://www.cs.ryerson.ca/~aharley/vis/conv/



CNNs in TensorFlow

- https://www.tensorflow.org/tutorials/images/cnn

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Dropout

- To reduce overfitting, we can apply **Dropout** to the network

  Basically, a form of regularization

- The idea is that we let a layer to randomly drop out a few output units from the layer itself during the training process

  By setting the activation to zero

  We can pick the percentage of the output units that are dropped out randomly from the applied layer. Example:

```
layers.MaxPooling2D(),
layers.Dropout(0.2),
layers.Flatten(),
```