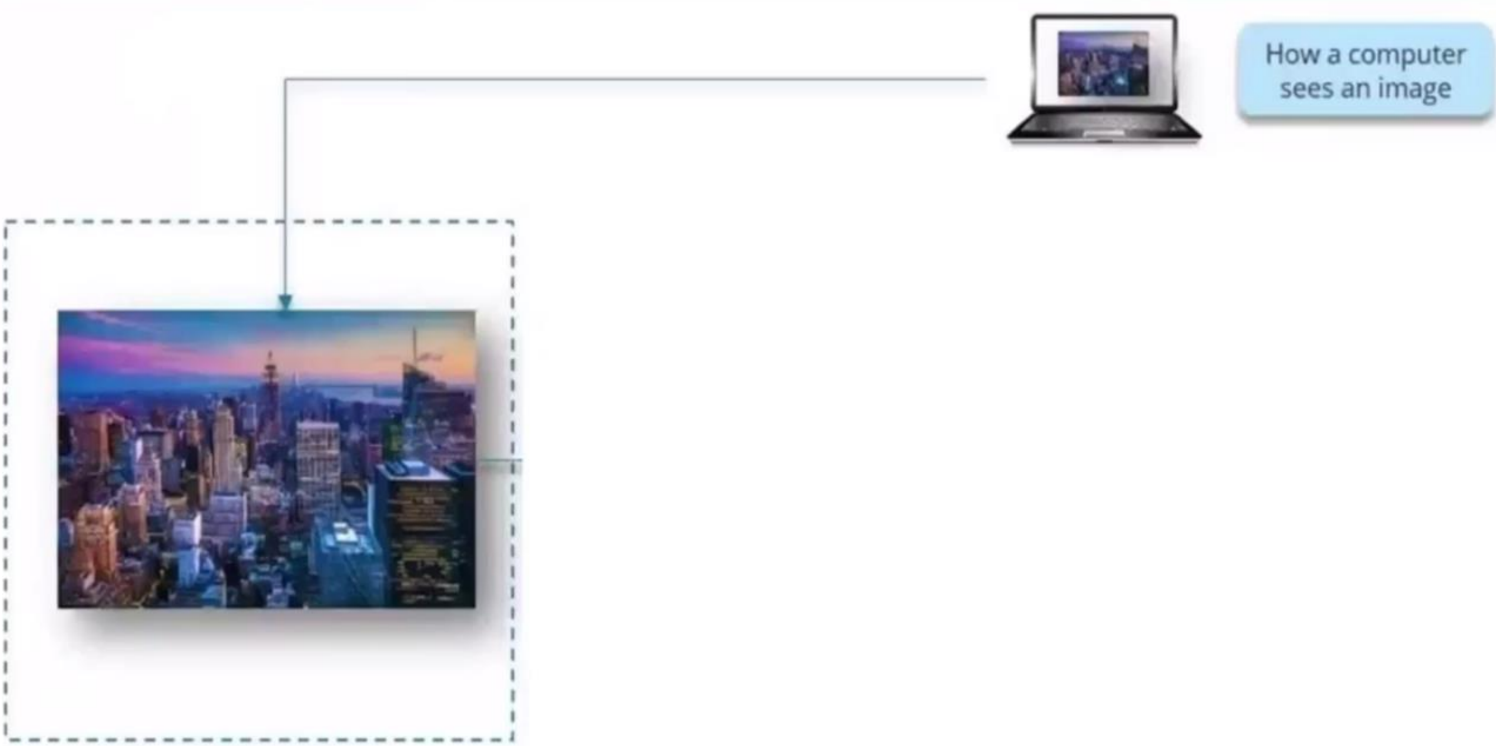
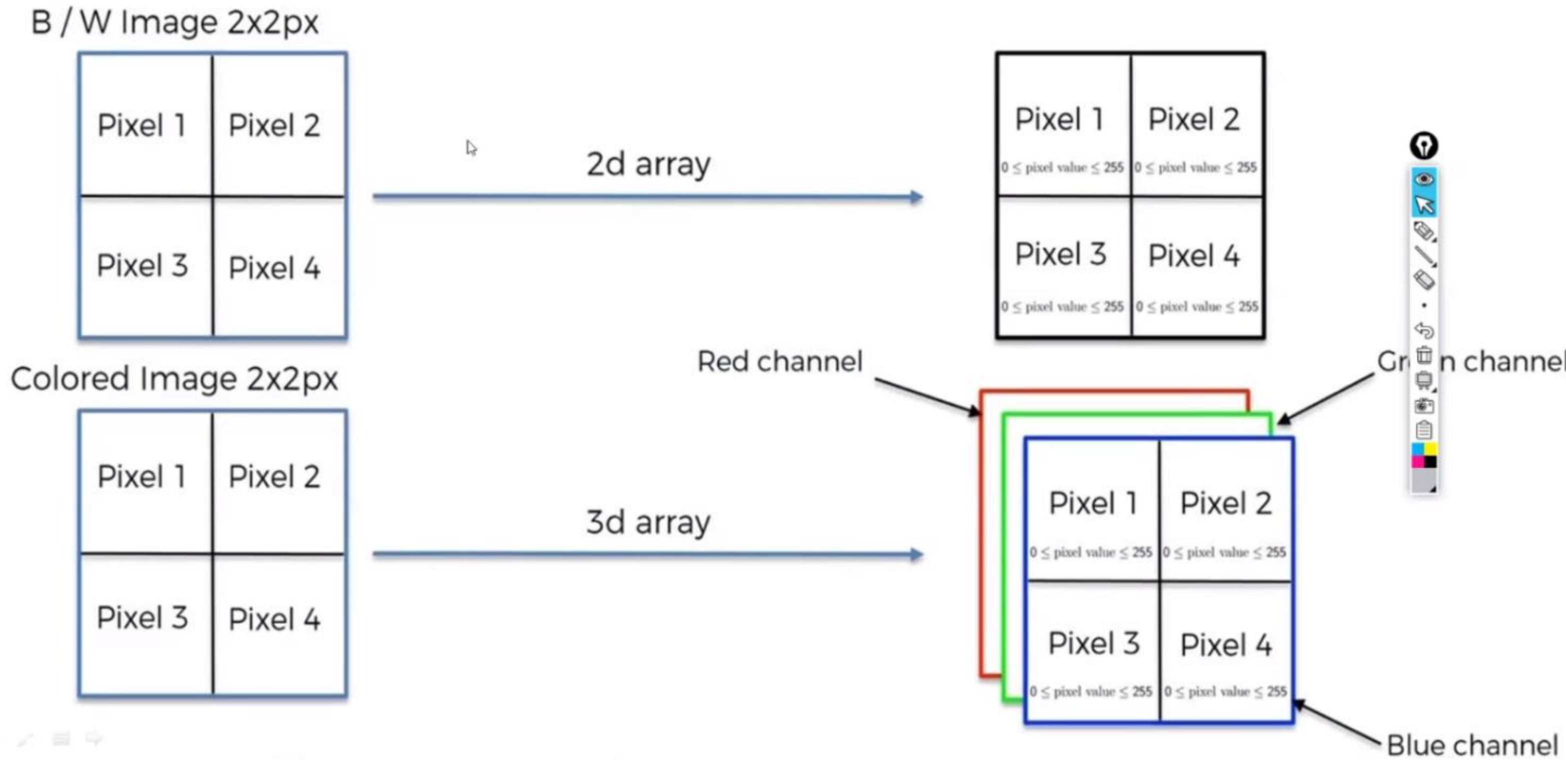
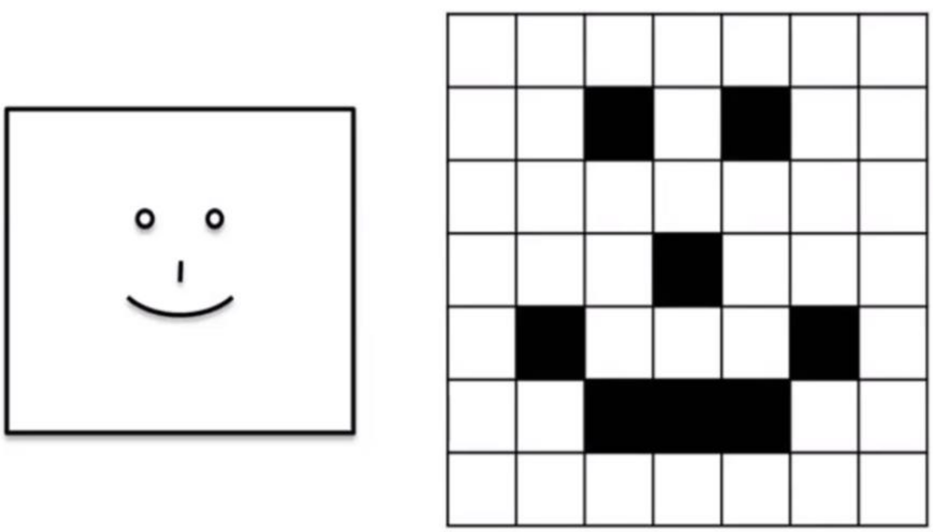
Convolutional Neural Networks

How a Computer Reads an Image







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

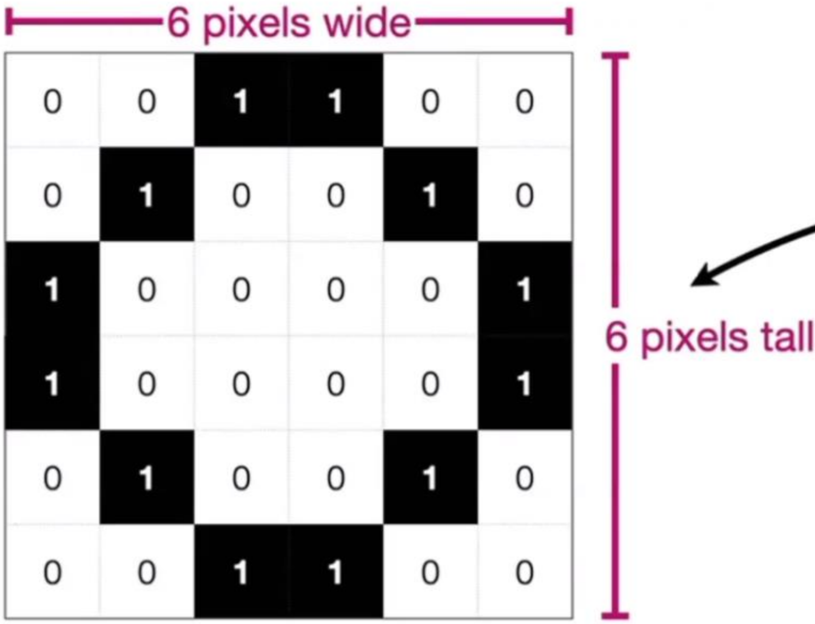
Why Not Fully Connected Networks

Let’s see why we cannot use fully connected networks for image classifications

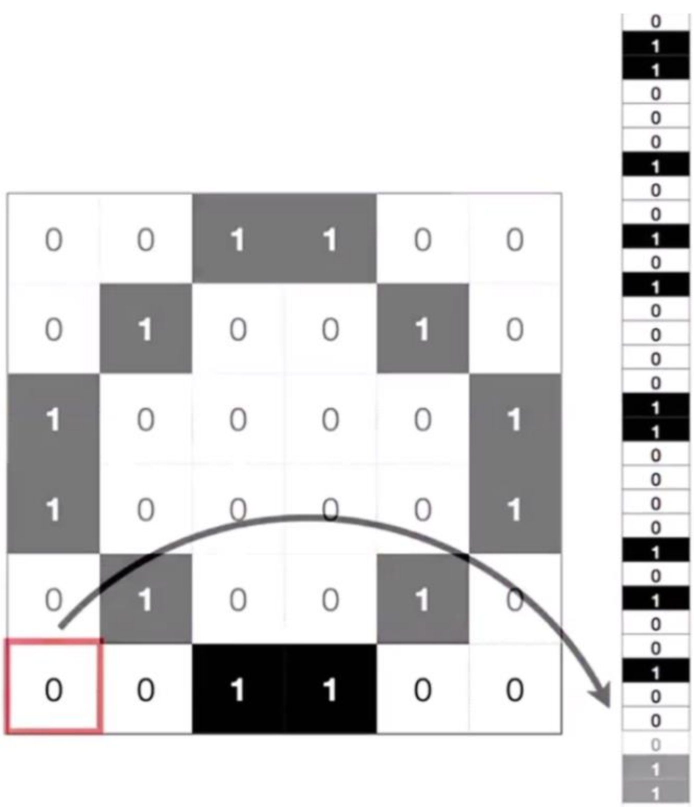
We will start with the image of the letter **“O”**.

The letter “O”, zoomed in.

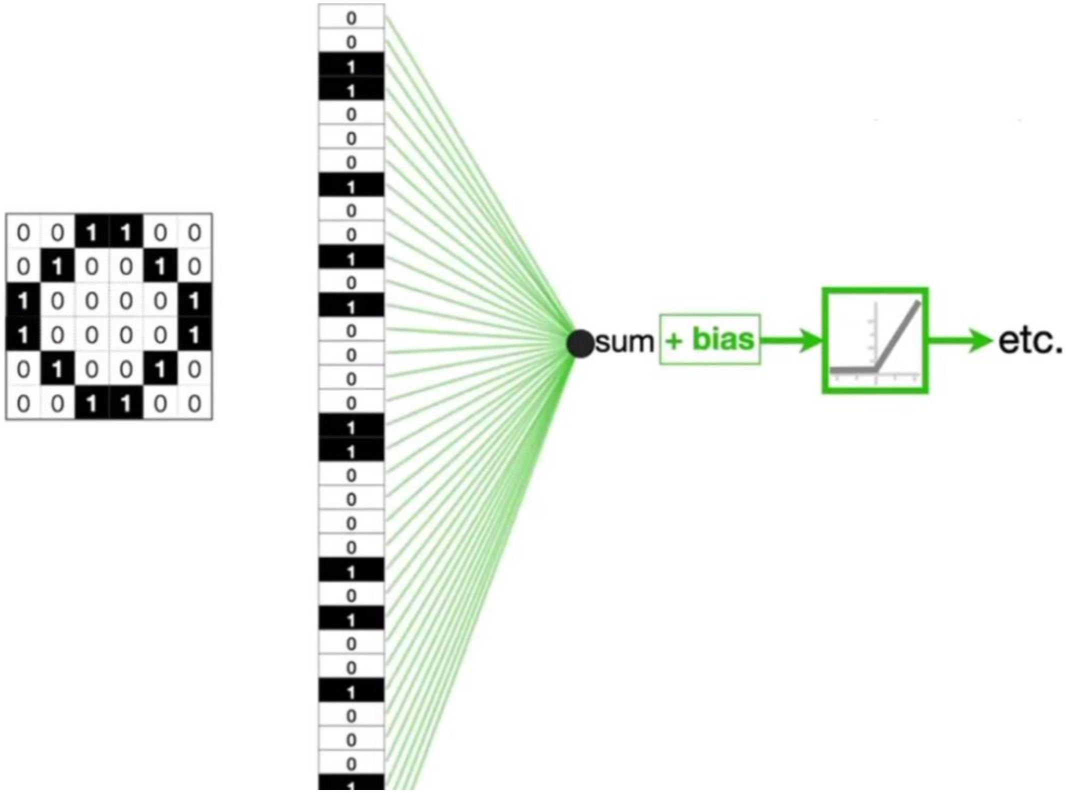
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0



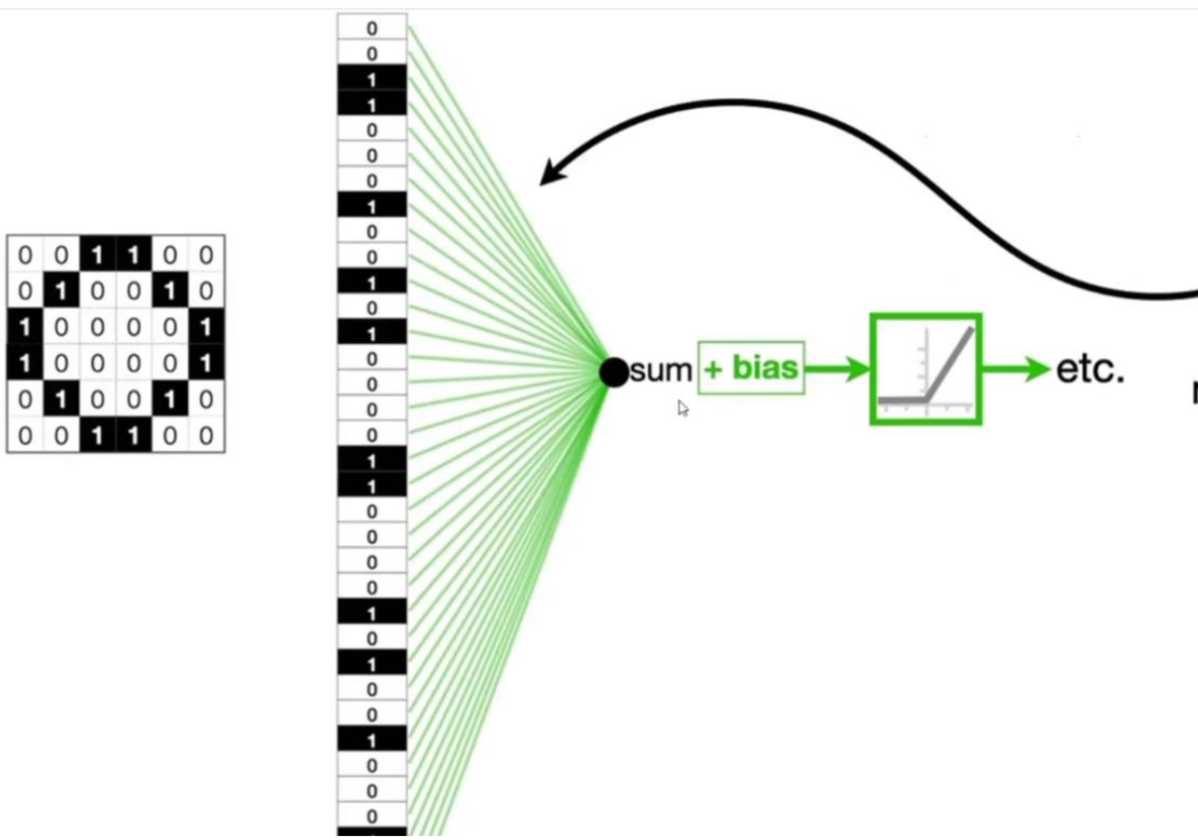
Now, because this image is so small, just **6** pixels by **6** pixels…



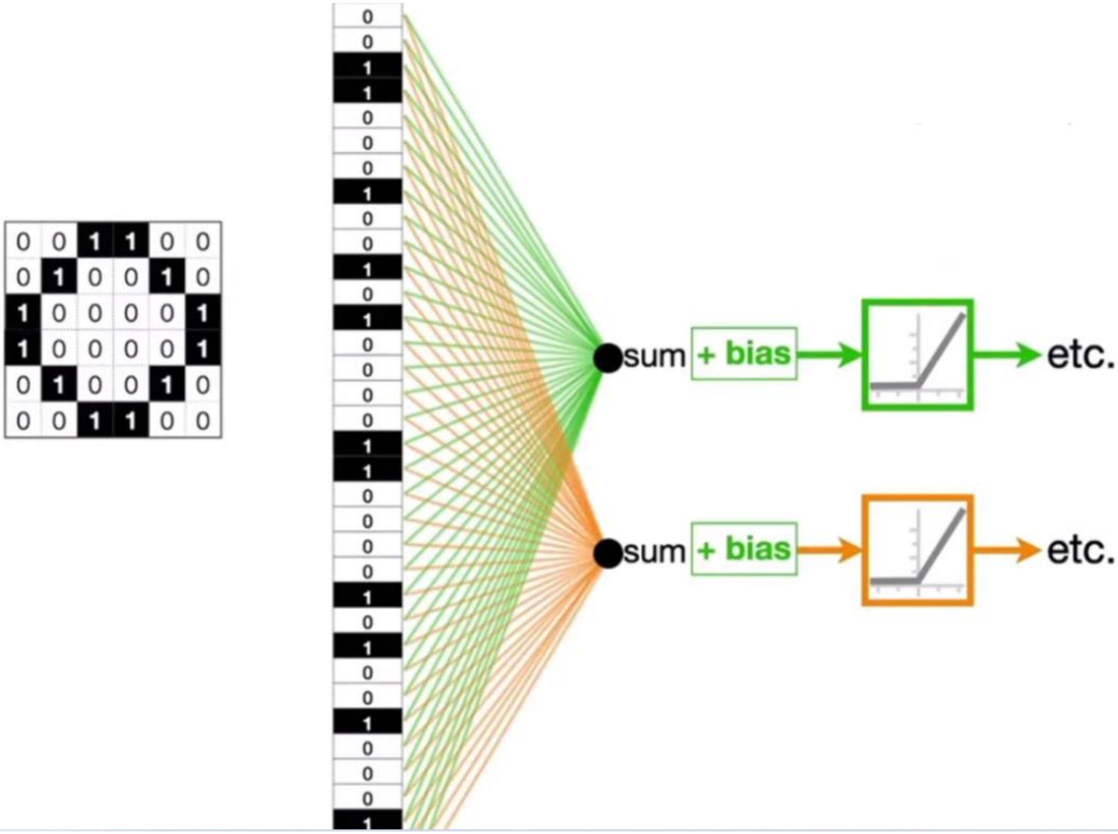
…into a single column of **36** input nodes…



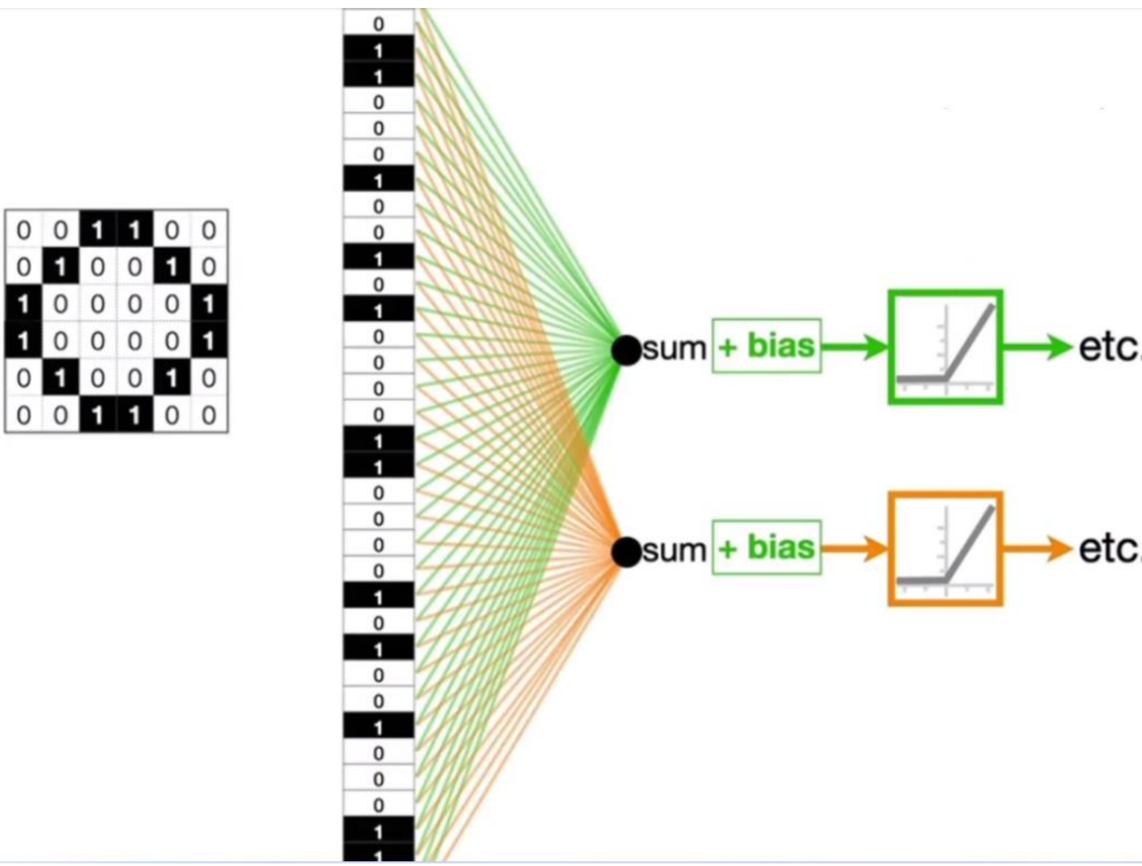
...and connect the input nodes to a **Hidden Layer**.



So here we have **36** connections from the **36** input nodes to this node in the **Hidden Layer**.



...and each additional node adds an additional **36 Weights** that we need to estimate.



Like I said, because the original image is small (**6x6**) and black and white, something like this could work.

However, if we had a larger image, like **100** pixels by **100** pixels, which is still pretty small compared to real world pictures...

...then we would end up with having to estimate **10,000 Weights *per* node** in the **Hidden Layer**!!!

So, this method doesn't scale very well.

How CNN Works?

Let's understand how Convolutional Neural Networks Work

Convolutional Neural Networks

STEP 1: Convolution

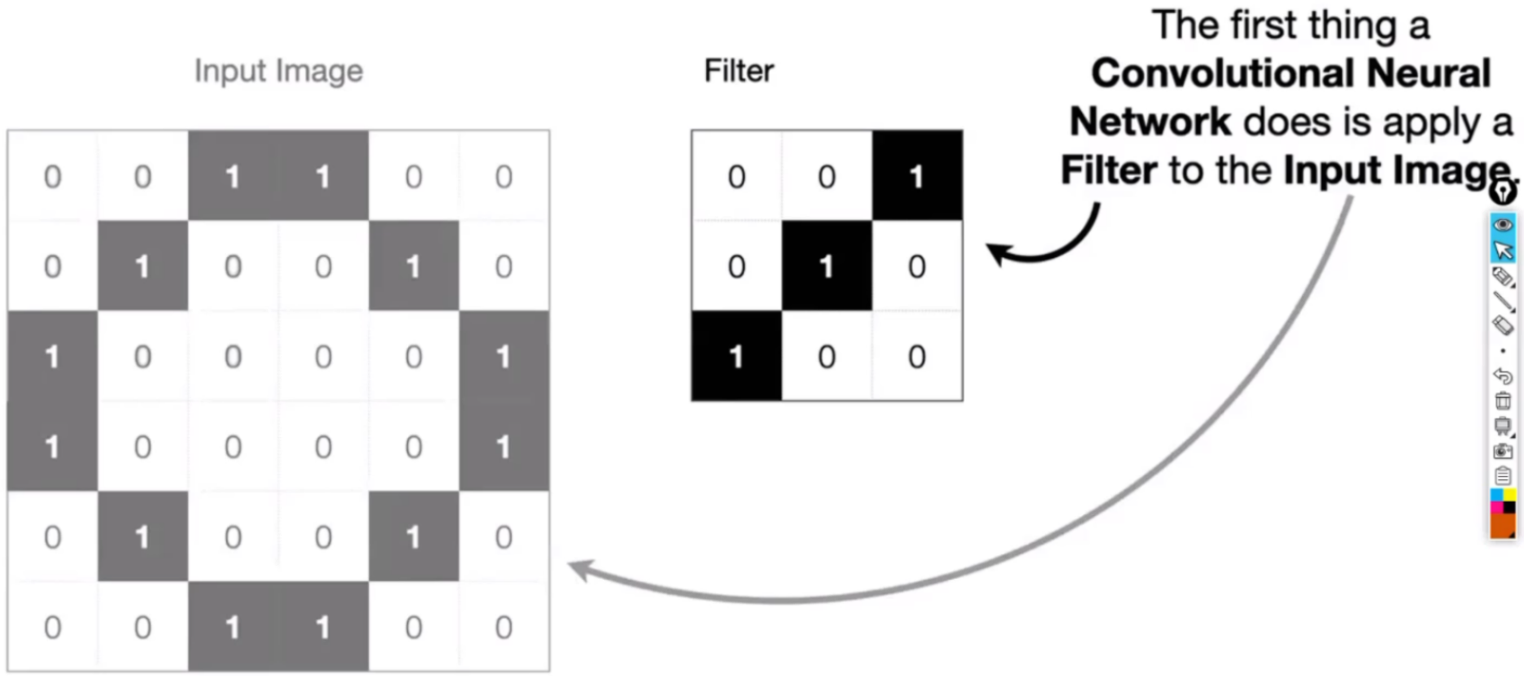
Step 1 - Convolution Operation
Step 1(b) - ReLU Layer

STEP 2: Max Pooling

STEP 3: Flattening

STEP 4: Full Connection

Step 1 – Convolution



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

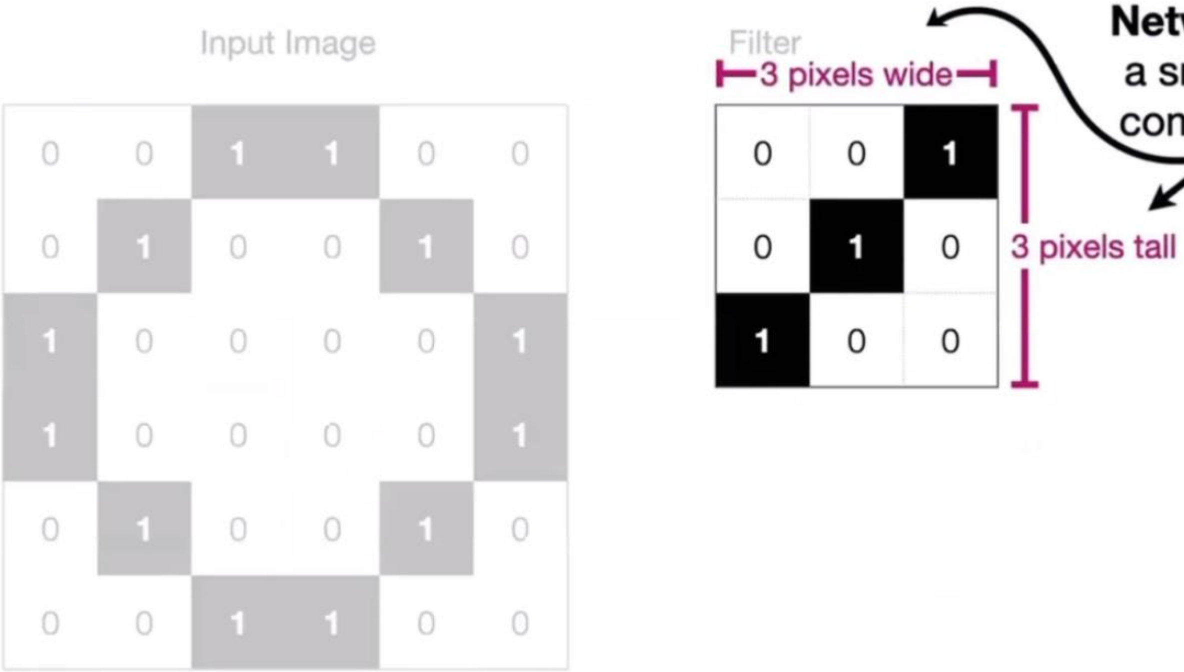
0	0	1
0	1	0
1	0	0

Within the filters the values are considered as Weights

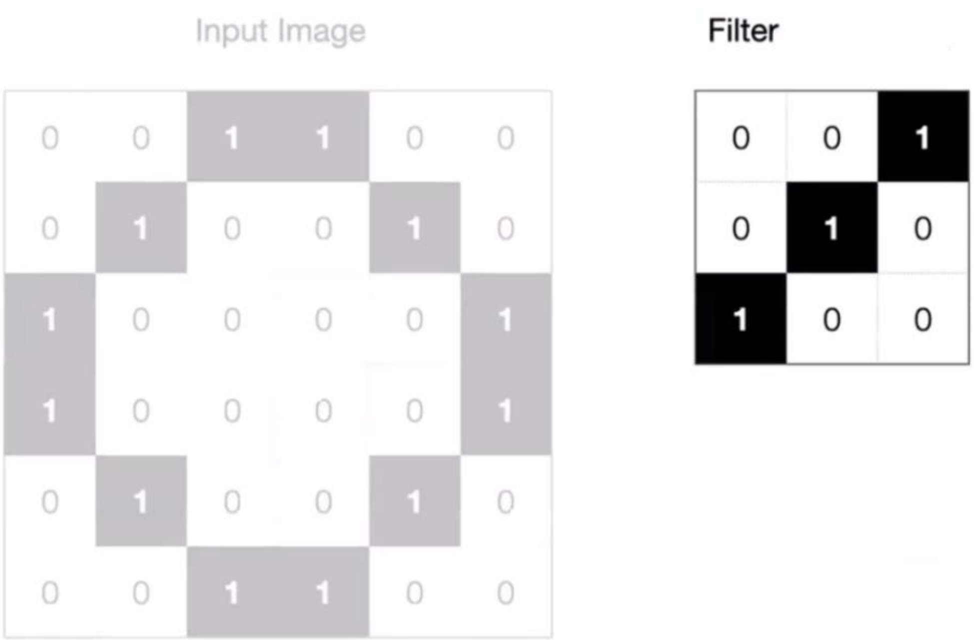
Step 1 - Convolution

$$* \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array}$$

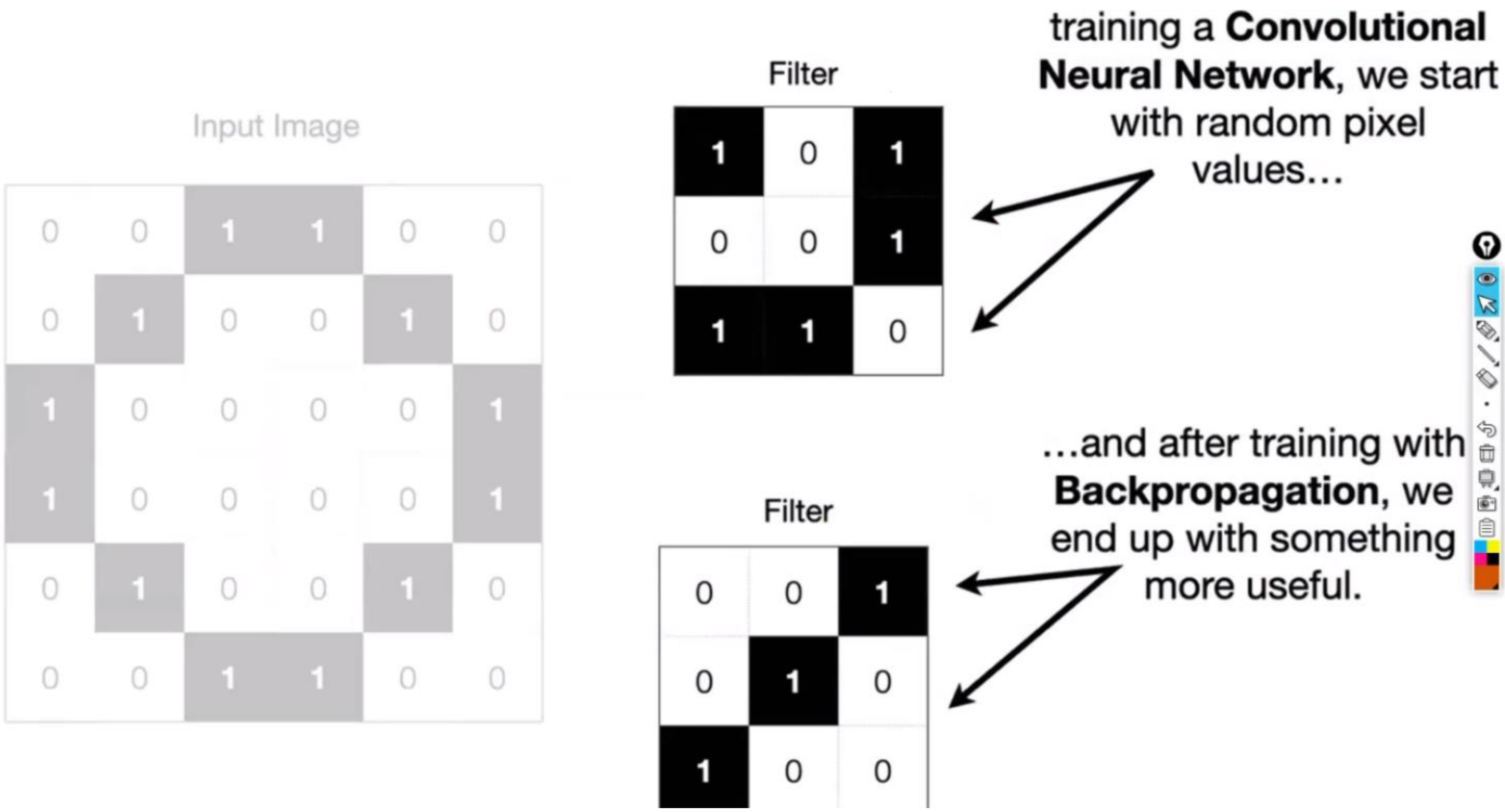
Here also Back propogation will be done to the filter

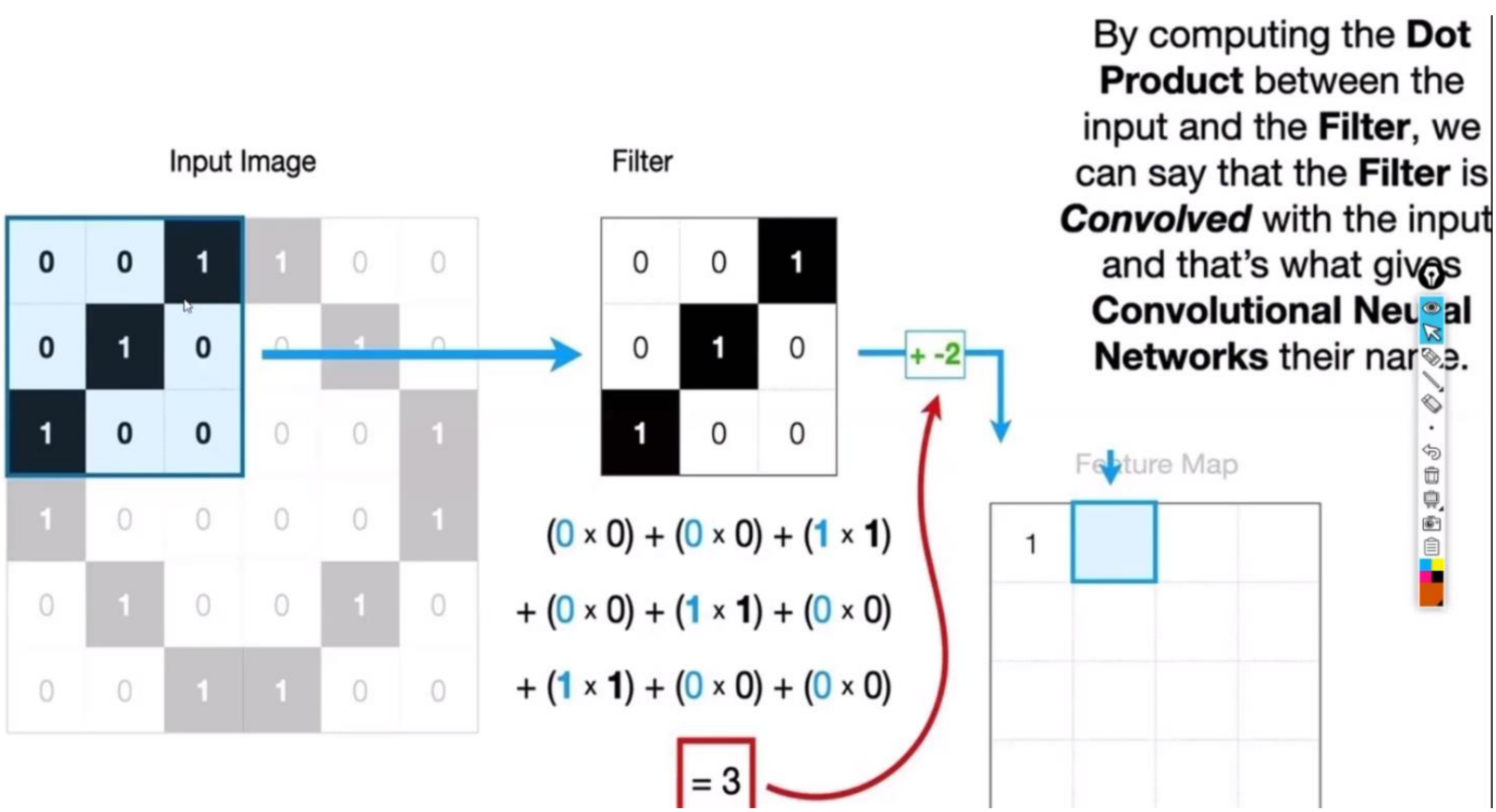


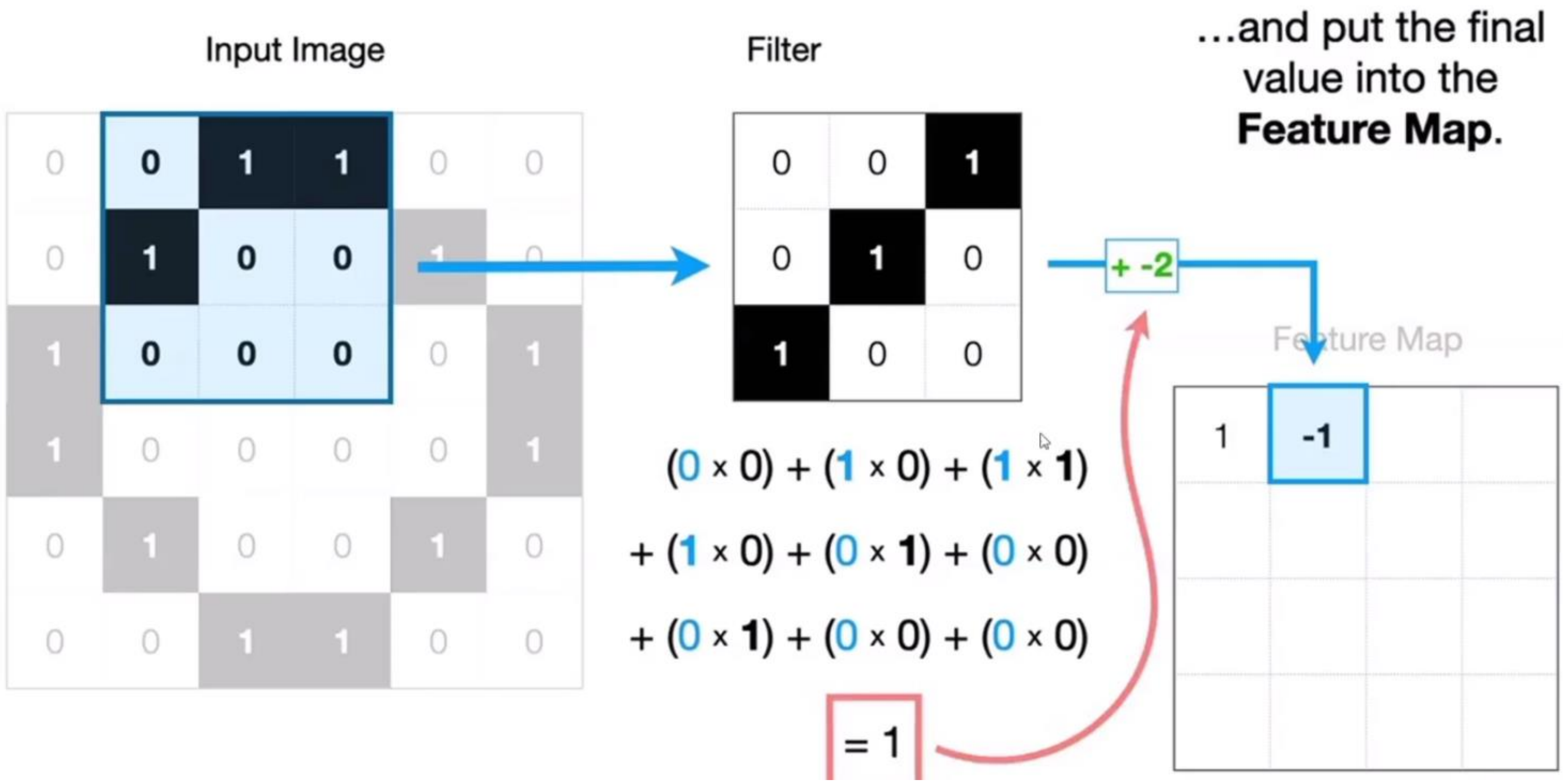
In **Convolutional Neural Networks**, a **filter** is just a smaller square that is commonly **3** pixels by **3** pixels...

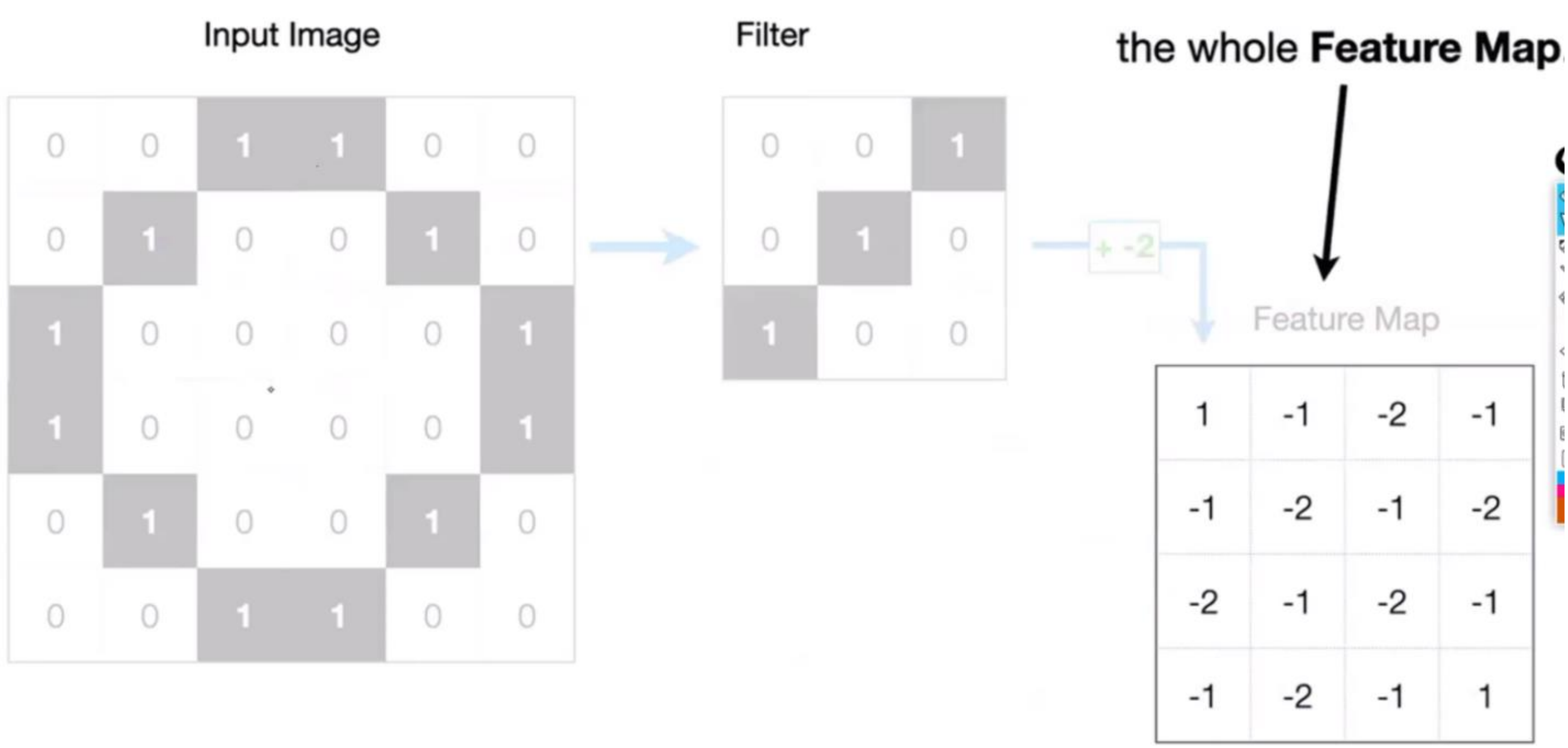


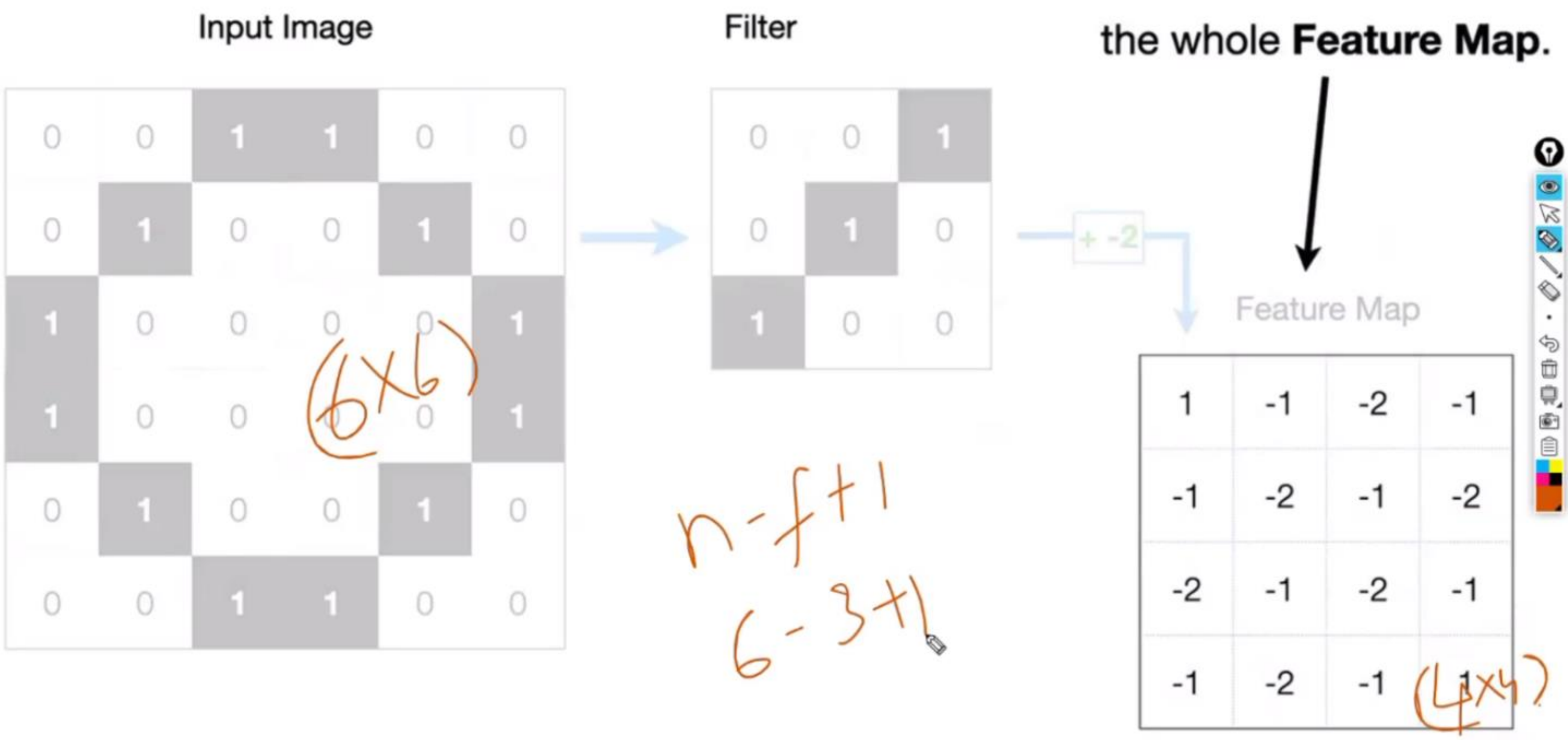
...and the intensity of each pixel in the filter is determined by **Backpropagation.**


training a **Convolutional Neural Network**, we start with random pixel values...
...and after training with **Backpropagation**, we end up with something more useful.
Input Image
Filter
Filter


By computing the **Dot Product** between the input and the **Filter**, we can say that the **Filter** is ***Convolved*** with the input and that's what gives **Convolutional Neural Networks** their name.
Input Image
Filter
+ -2
Feature Map
(0 x 0) + (0 x 0) + (1 x 1)
+ (0 x 0) + (1 x 1) + (0 x 0)
+ (1 x 1) + (0 x 0) + (0 x 0)
= 3


Input Image
Filter
…and put the final value into the **Feature Map**.
+ -2
Feature Map
(0 × 0) + (1 × 0) + (1 × 1)
+ (1 × 0) + (0 × 1) + (0 × 0)
+ (0 × 1) + (0 × 0) + (0 × 0)
= 1
1
-1


Input Image
Filter
the whole **Feature Map**.
+ -2
Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1



- **you should resize all images to the same dimensions** before feeding them into a CNN.
  - You can **resize directly** or **use padding** to maintain the aspect ratio.

Size of Feature Map = N – F + 1 , where N = input image pixels, F = filter pixels

Step 1 (B) – ReLU Layer

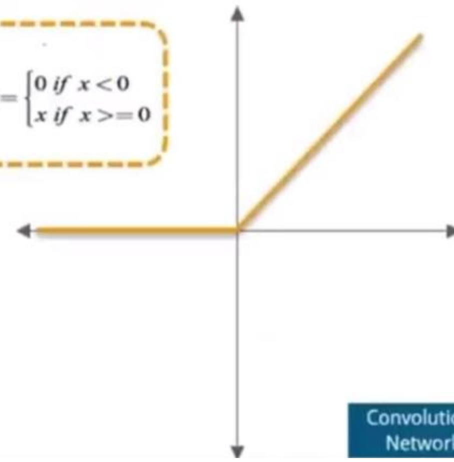
ReLU Layer

- ✓ In this layer we remove every negative values from the filtered images and replaces it with zero's
- ✓ This is done to avoid the values from summing up to zero

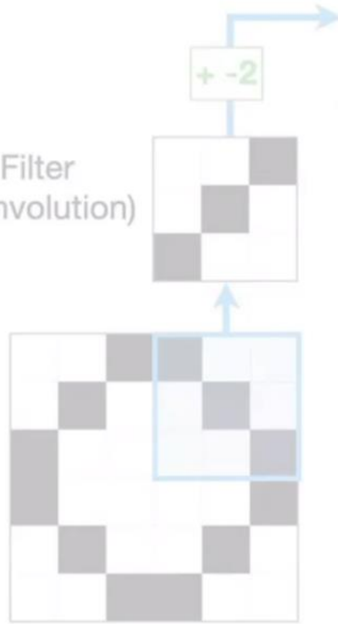
Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable

x	f(x)=x	F(x)
-3	f(-3) = 0	0
-5	f(-5) = 0	0
3	f(3) = 3	3
5	f(5) = 5	5

$$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$



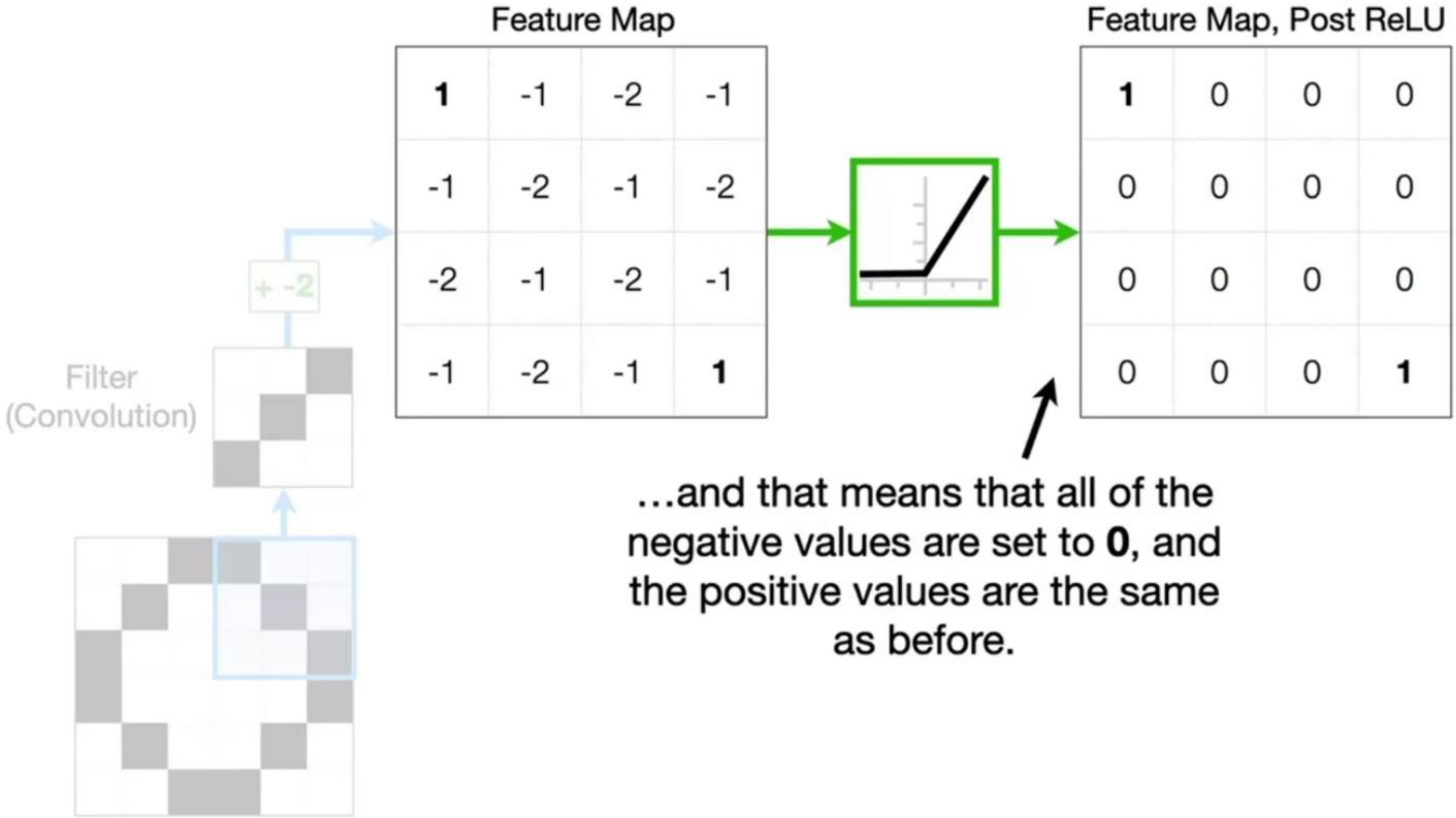
Filter (Convolution)



Feature Map

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	**1**

Now, typically, we run the **Feature Map** through a **ReLU Activation Function**...



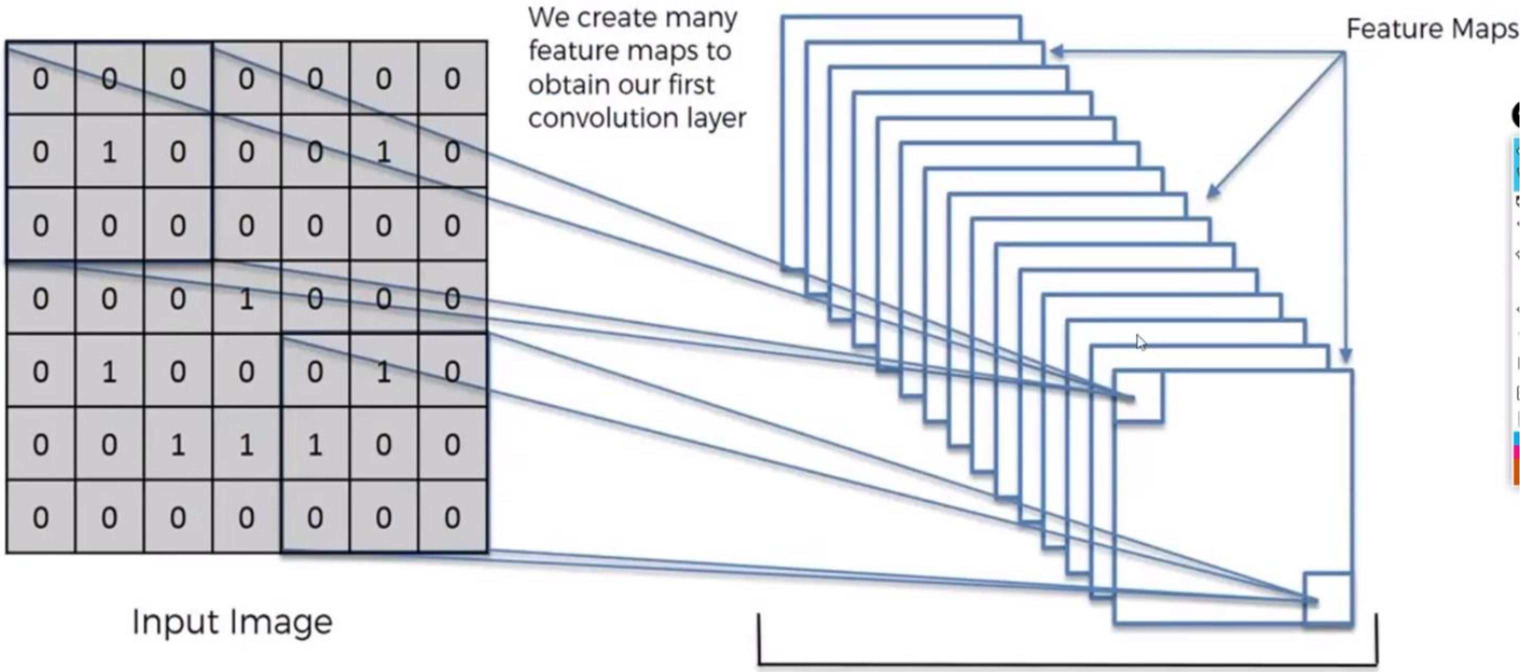
Step 1(B) – ReLU Layer

Step 1(B) – ReLU Layer

Here ReLU will remove the negitiveness.

Step 1(B) – ReLU Layer

Step 1 - Convolution



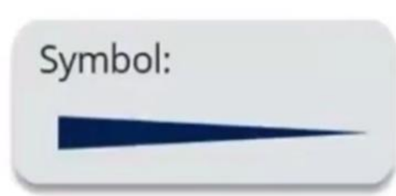
Step 2 – Pooling

Pooling Layer

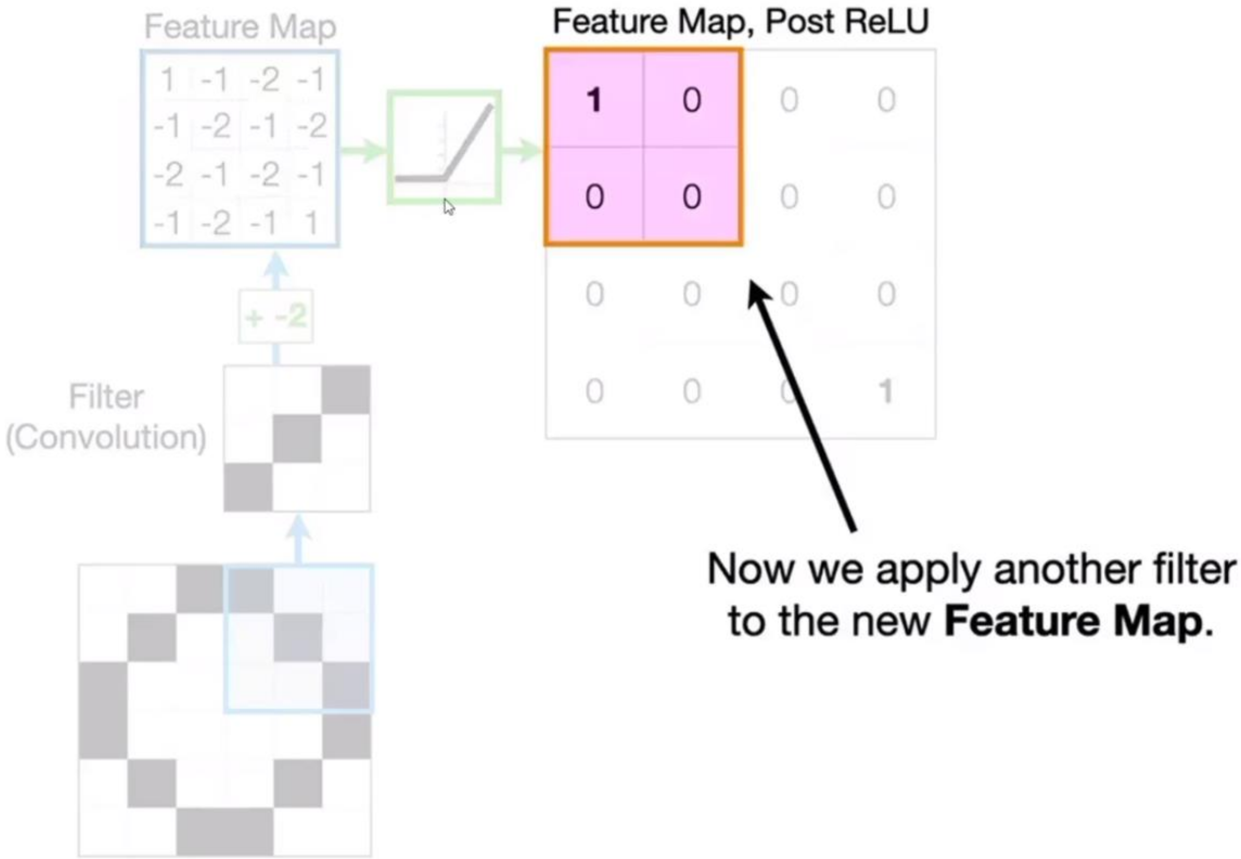
In this layer we shrink the image stack into a smaller size

Steps:

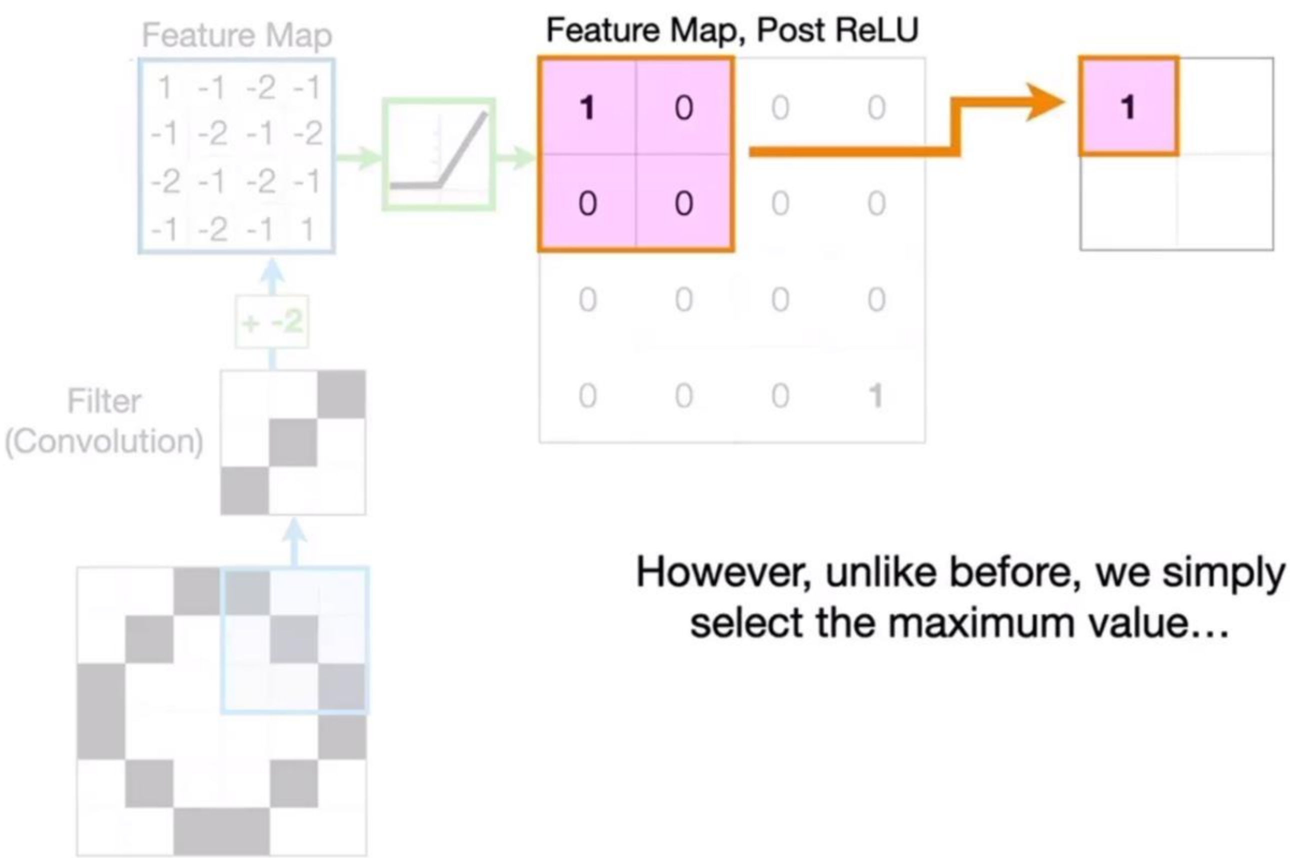
1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually 2).
3. Walk your window across your filtered images.
4. From each window, take the maximum value.

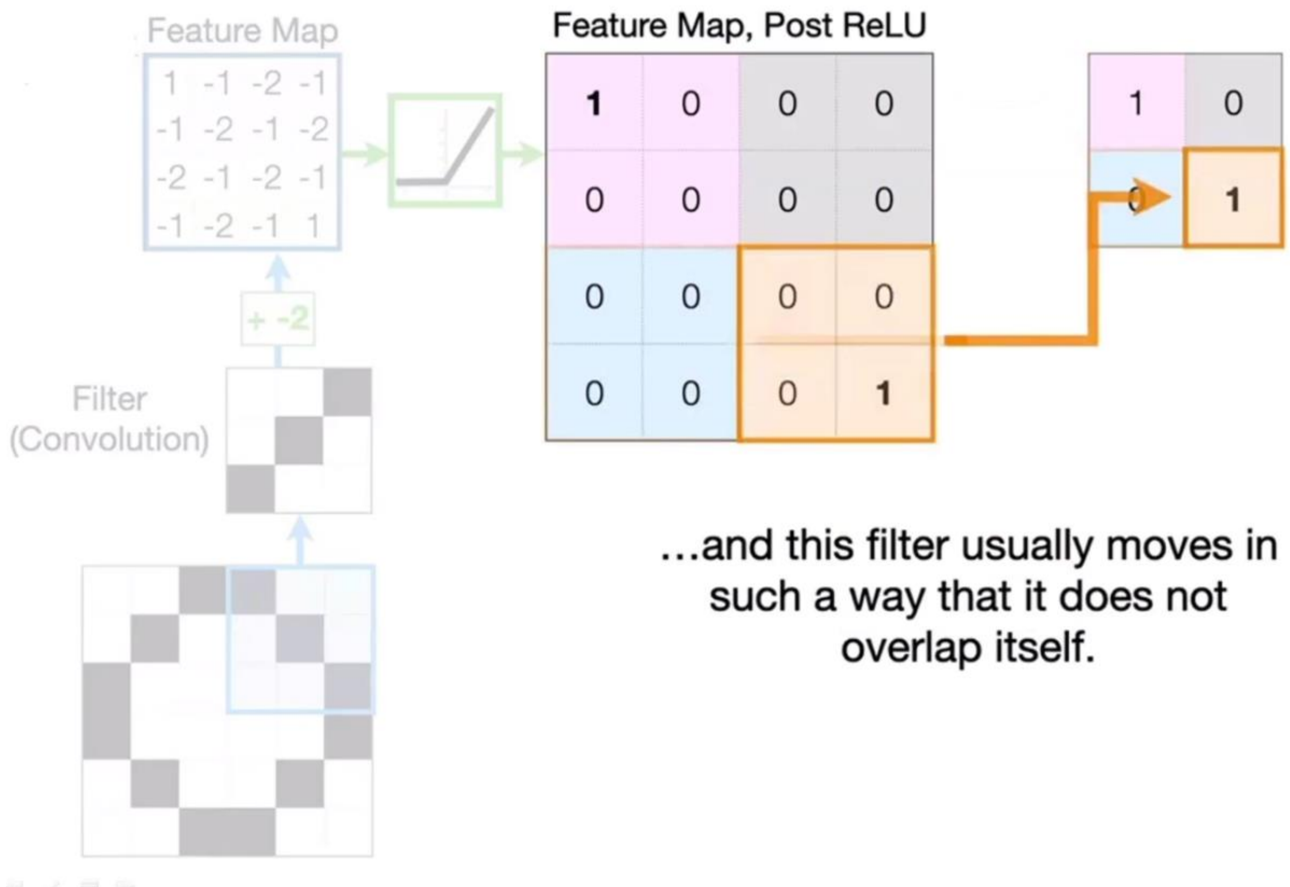


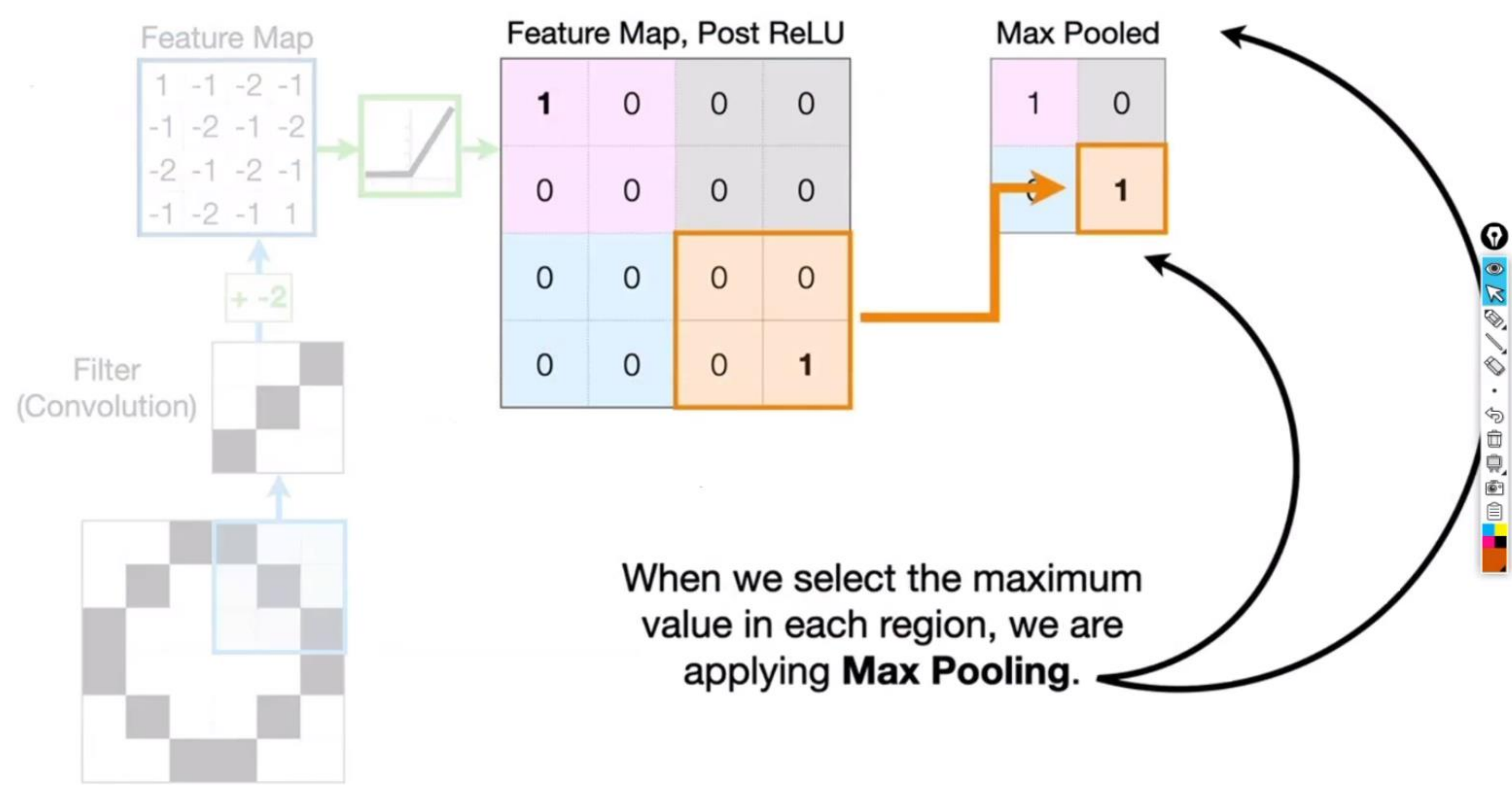
Let's perform pooling with a window size 2 and a stride 2



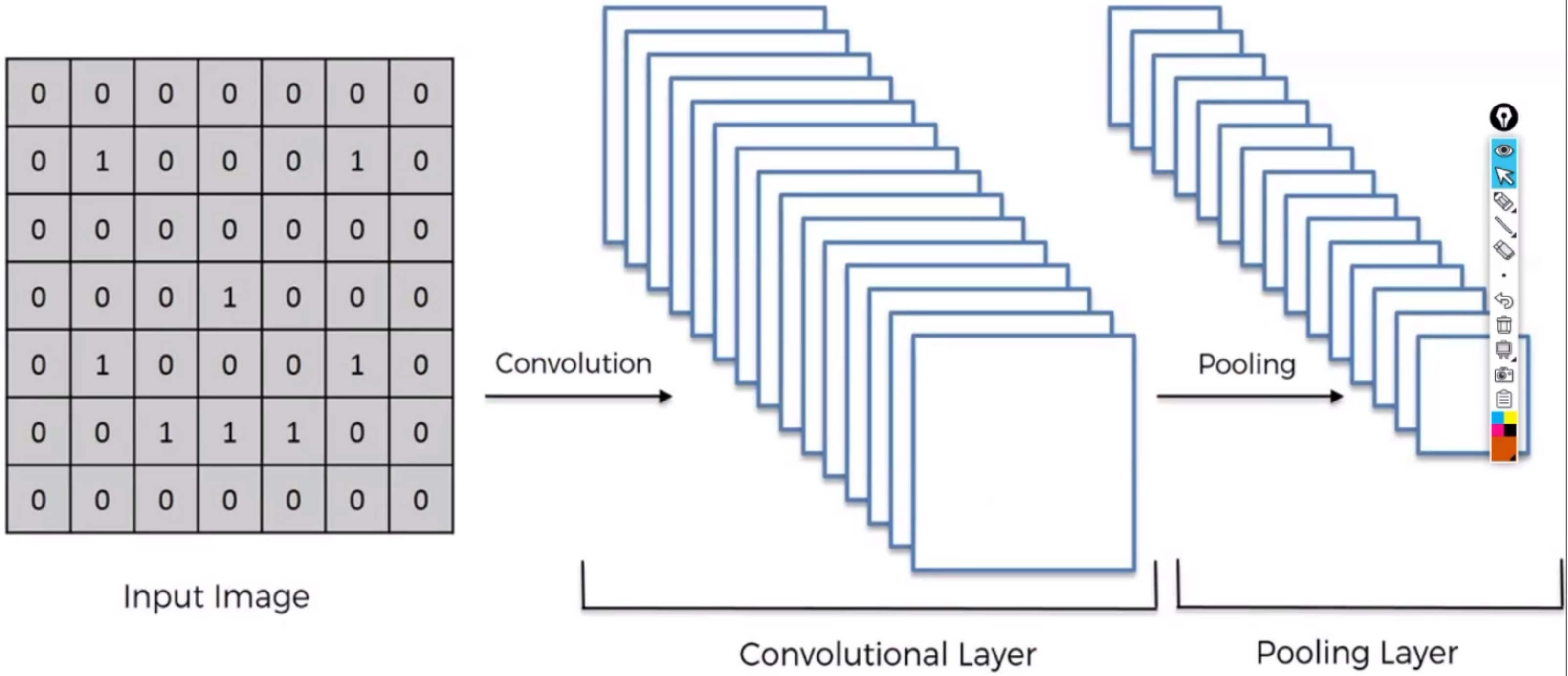
Now we take pool size of 2x2 for which we get max size of 1. This concept is called max pooling. but here it is 2x2 pool size operation is a little different than convolution as it does not overlap.



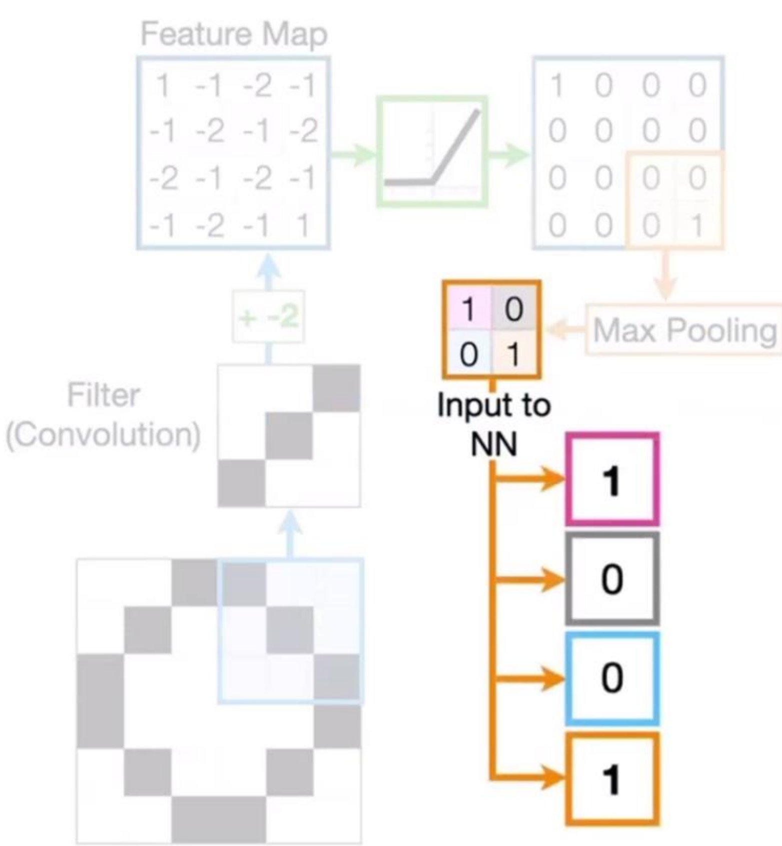

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
Feature Map, Post ReLU
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
...and this filter usually moves in such a way that it does not overlap itself.


Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
Feature Map, Post ReLU
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
Max Pooled
1 0
0 1
When we select the maximum value in each region, we are applying **Max Pooling**.

Step 2 - Max Pooling

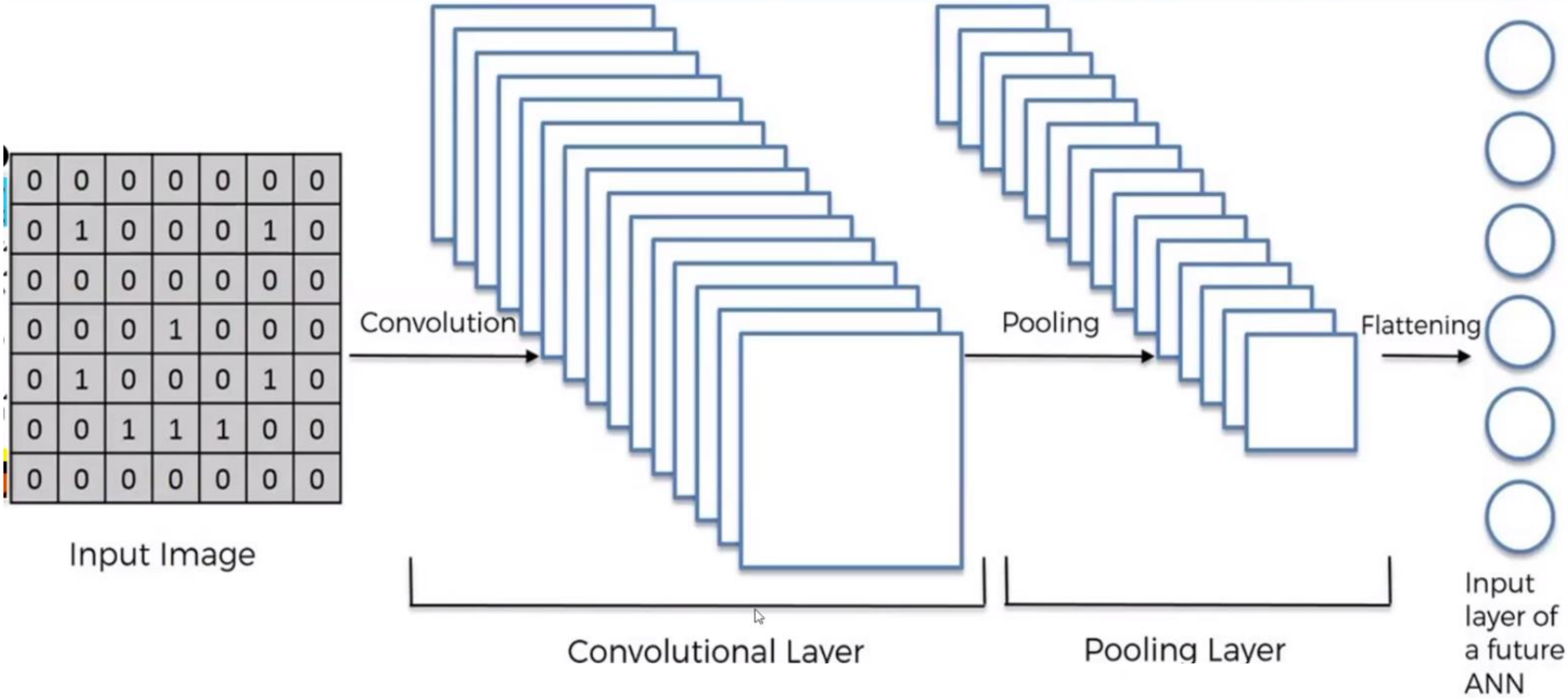


Step 3 – Flattening



Now let's convert the **Pooled Layer** into a column of **Input Nodes**.

Step 3 - Flattening



Step 4 – Full Connection