Convolutional Neural Network

Image Classification using CNN

Group Members:

Mehar Abdul Wahab (201980053)

Mehar Hamid Ishfaq (201980038)

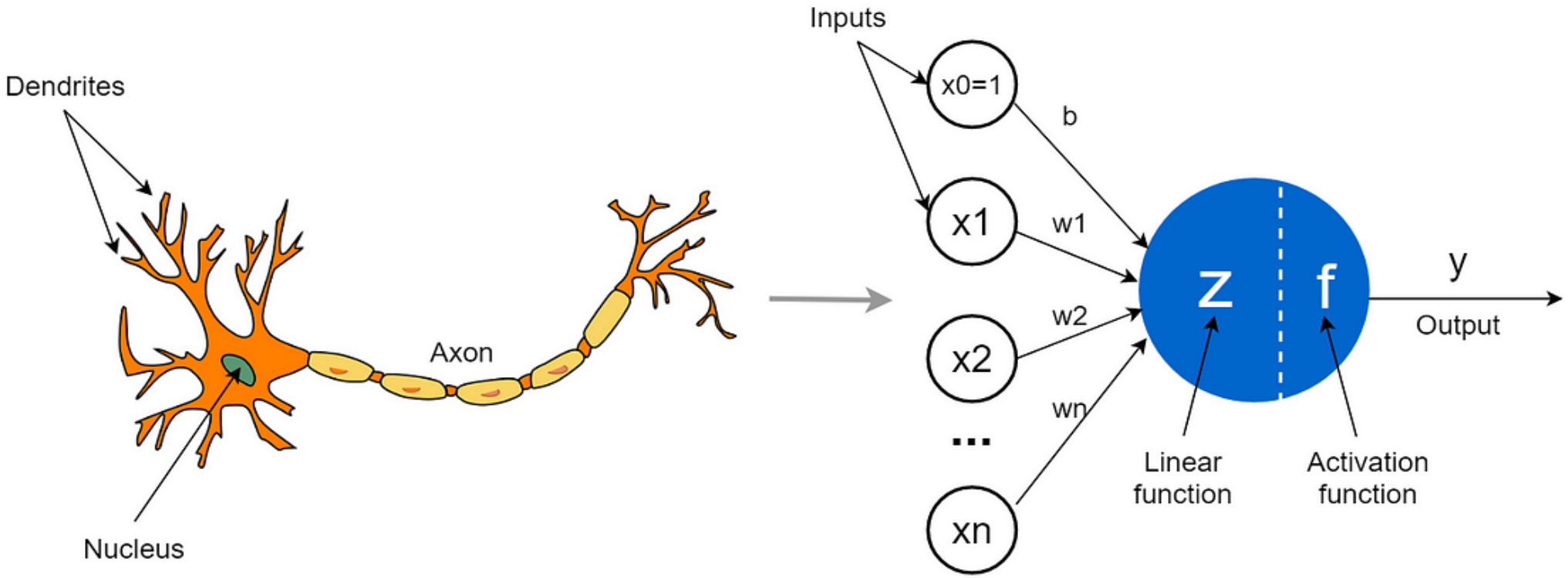
Rajpoot Sufyan Ahmad (201980013)

Mughal Nadeem (201980050)

Applications

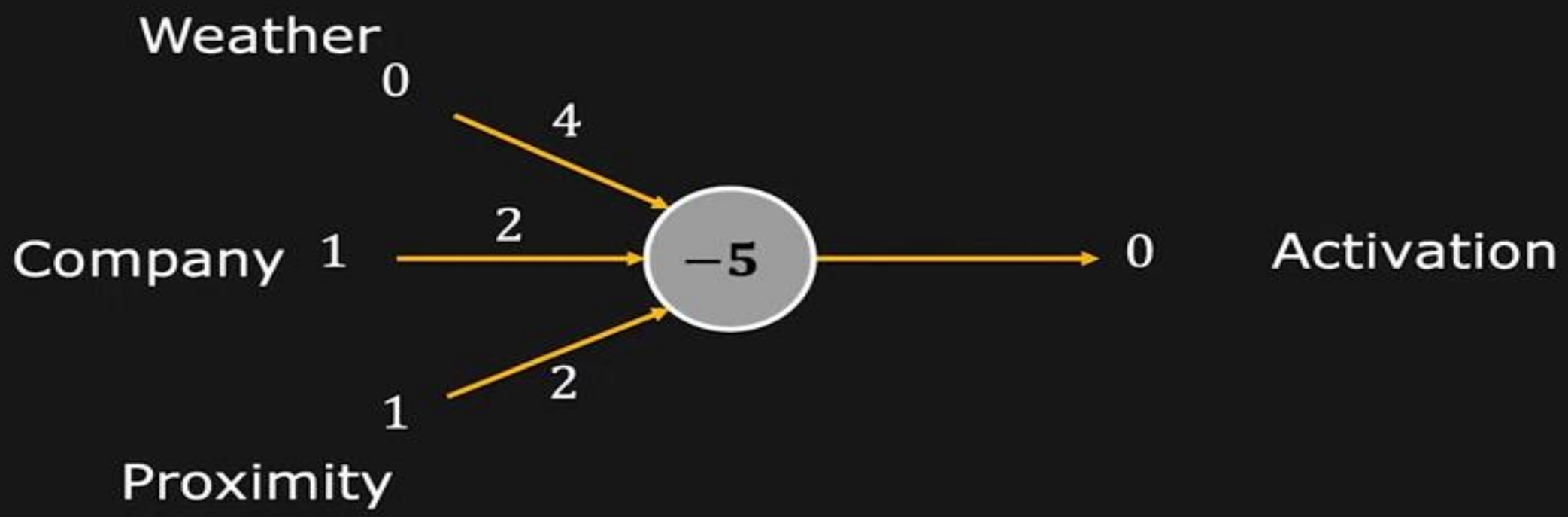
- Object Recognition and Classification
- Quality Control in Manufacturing
- Agriculture
- Food Recognition
- Social Media
- Gesture Recognition
- Self-Driving Cars

Biological vs Artificial Neuron

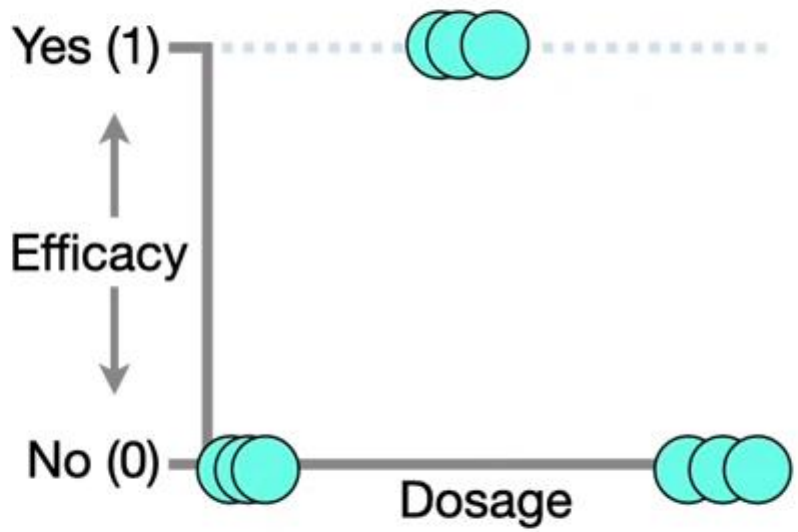


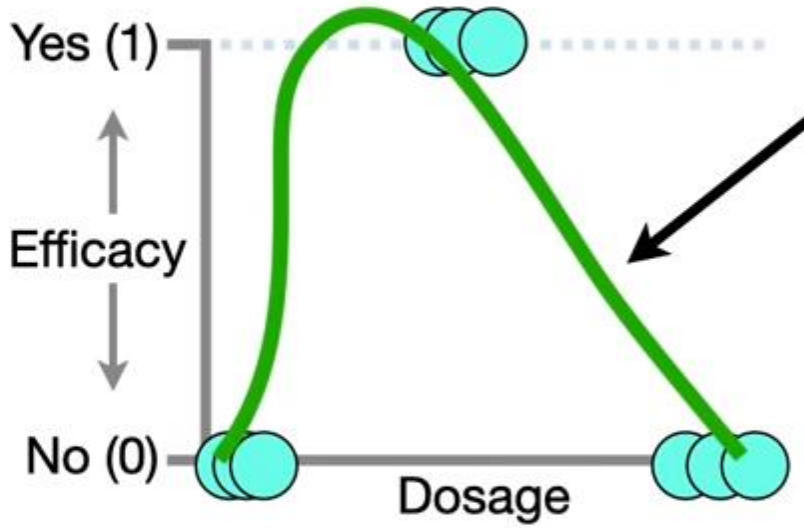
Perceptron: Example

Will you go to the movies? No

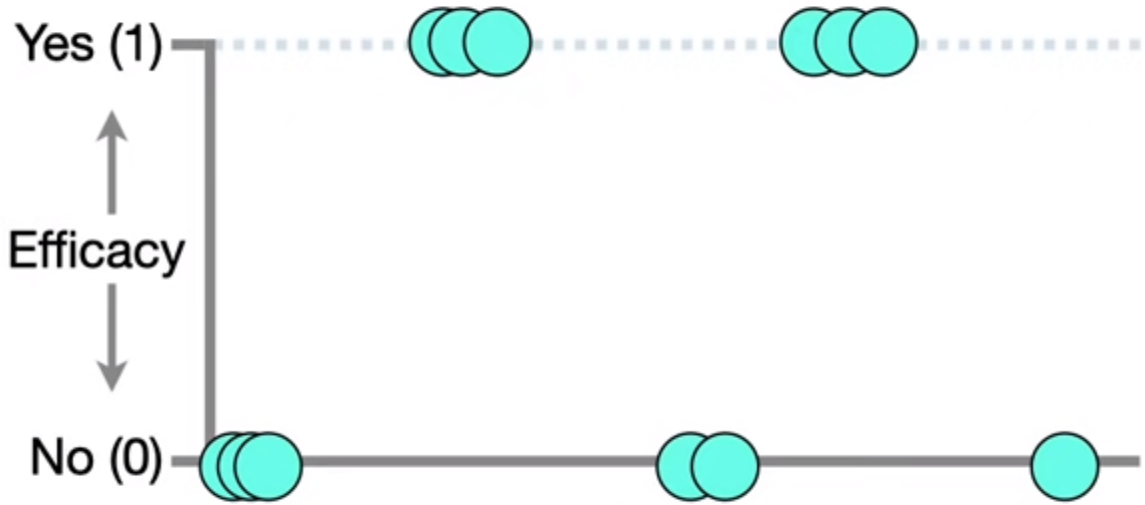


Now that we have this data, we would like to use it to predict whether or not a future **Dosage** will be Effective.



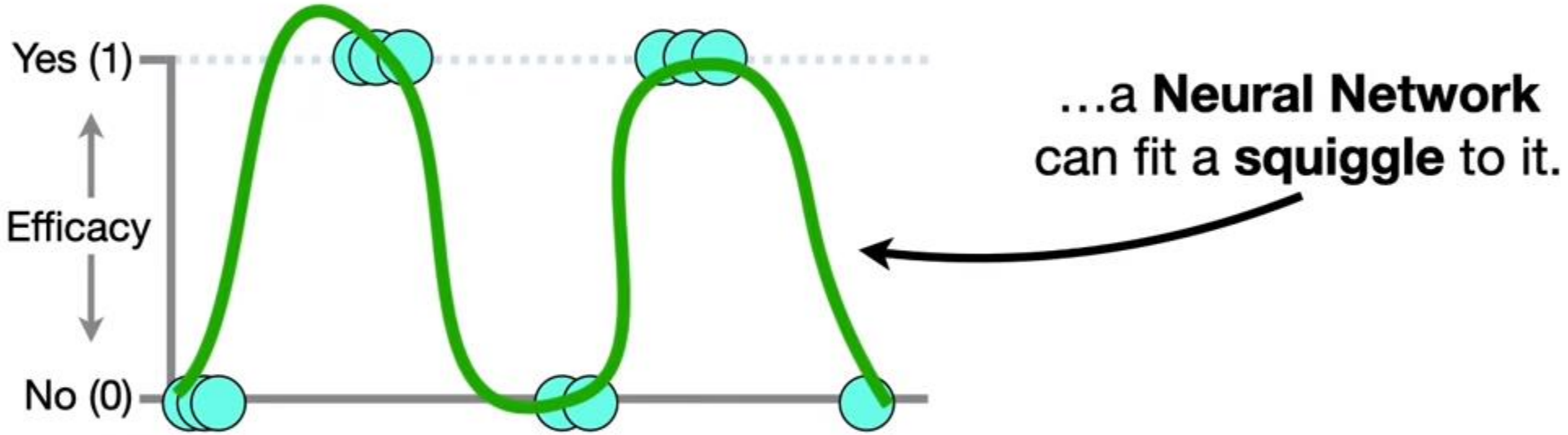


The good news is that a **Neural Network** can fit a **squiggle** to the data.

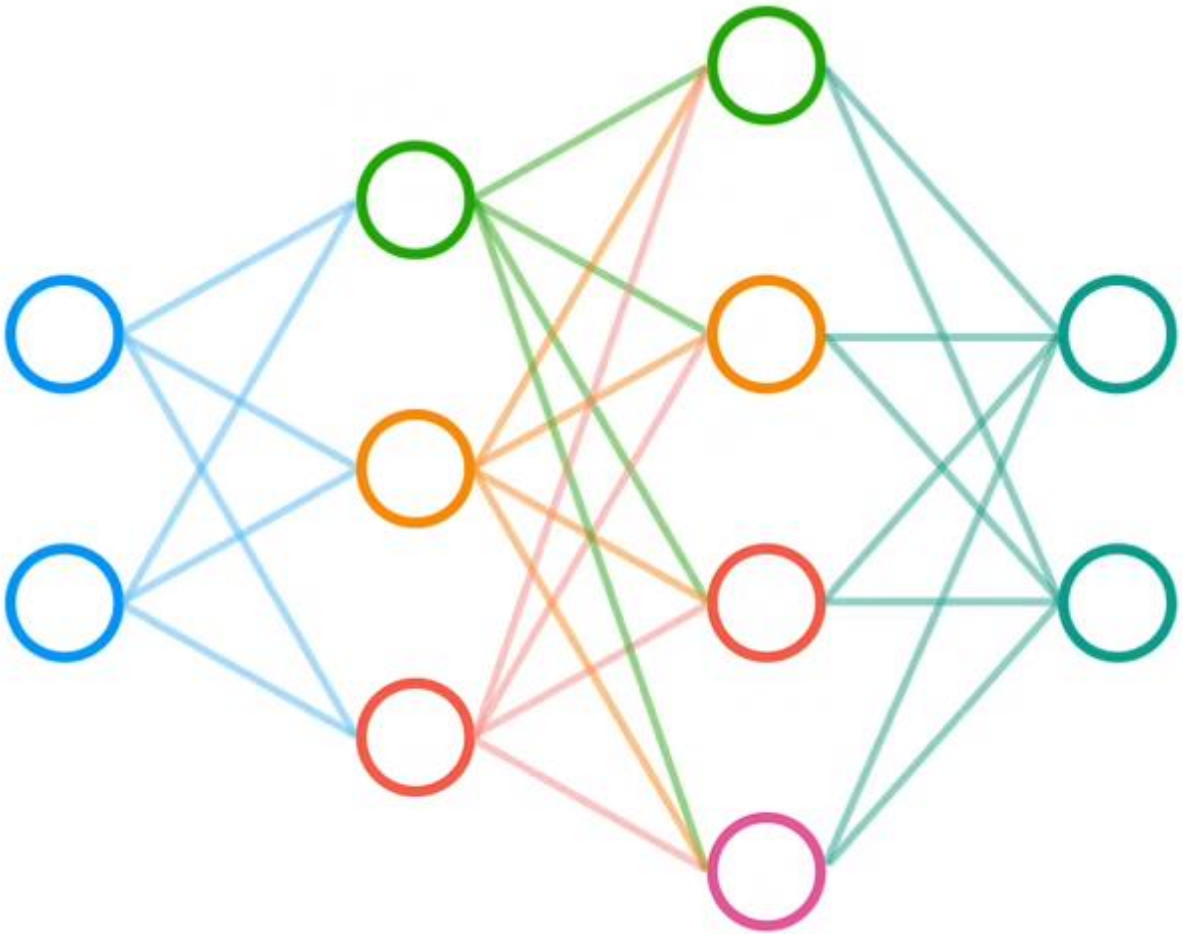


And even if we a have really complicated dataset like this…

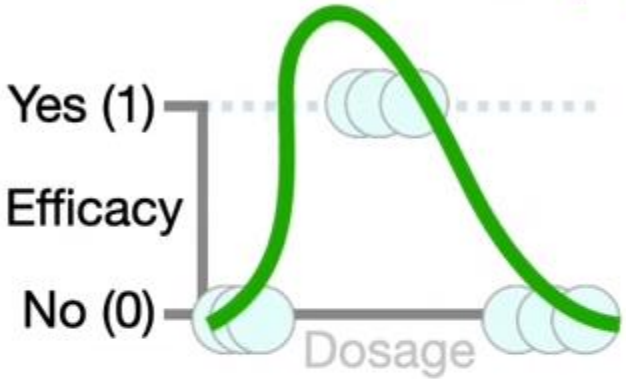

DOUBLY
BAM!!!
SQ!

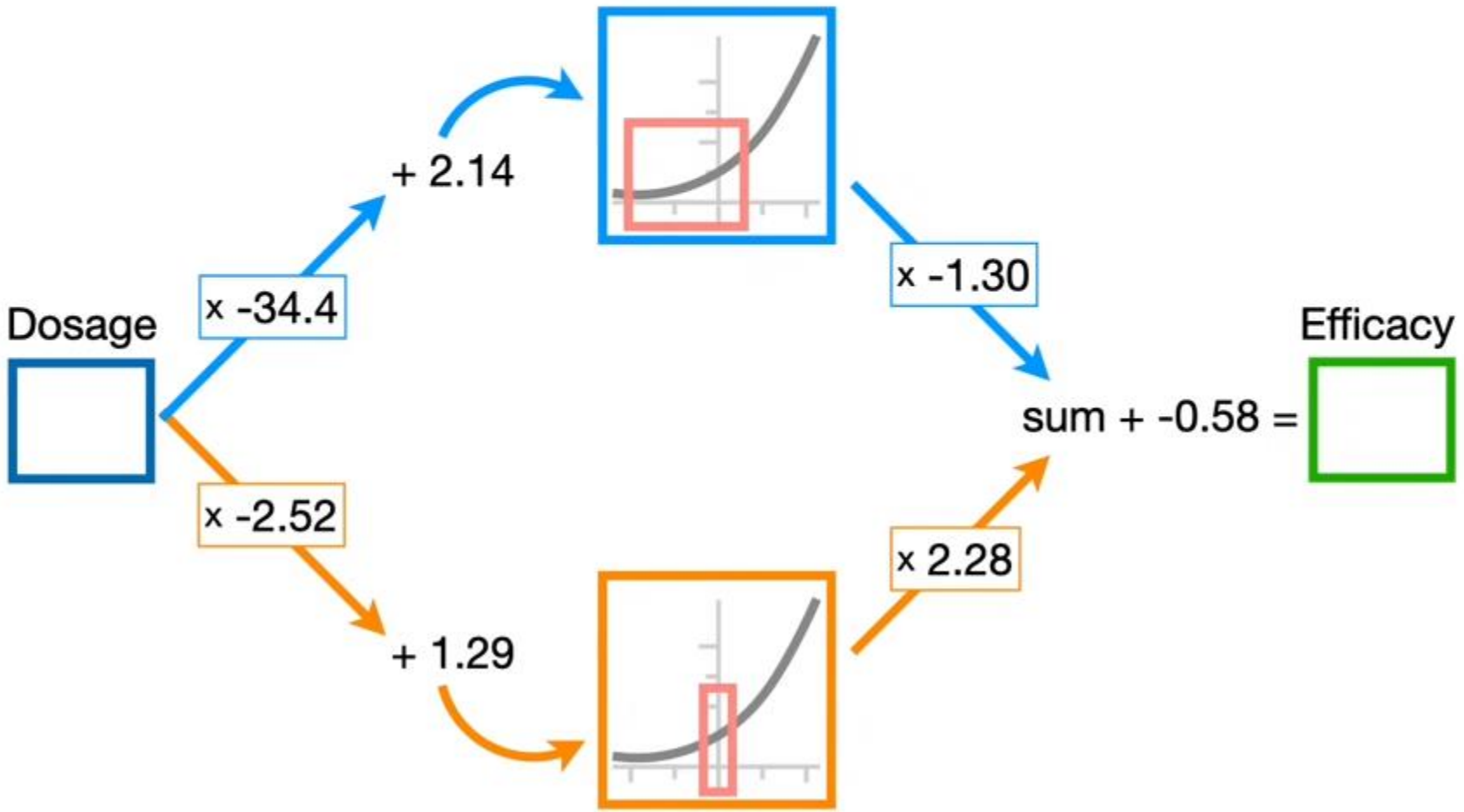

Yes (1)
Efficacy
No (0)
…a **Neural Network**
can fit a **squiggle** to it.

Now, even though this **Neural Network** looks fancy, it is still made from the same parts…

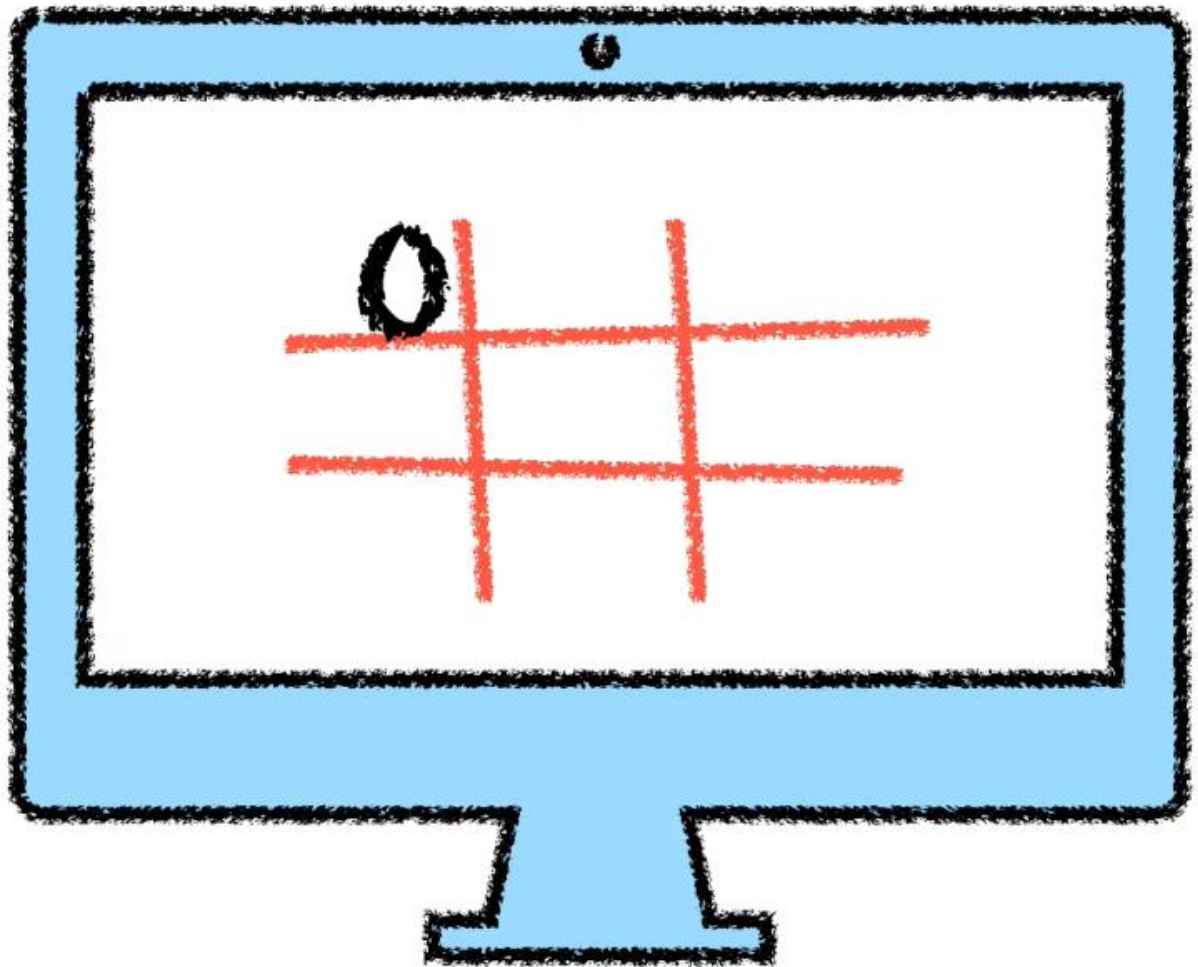


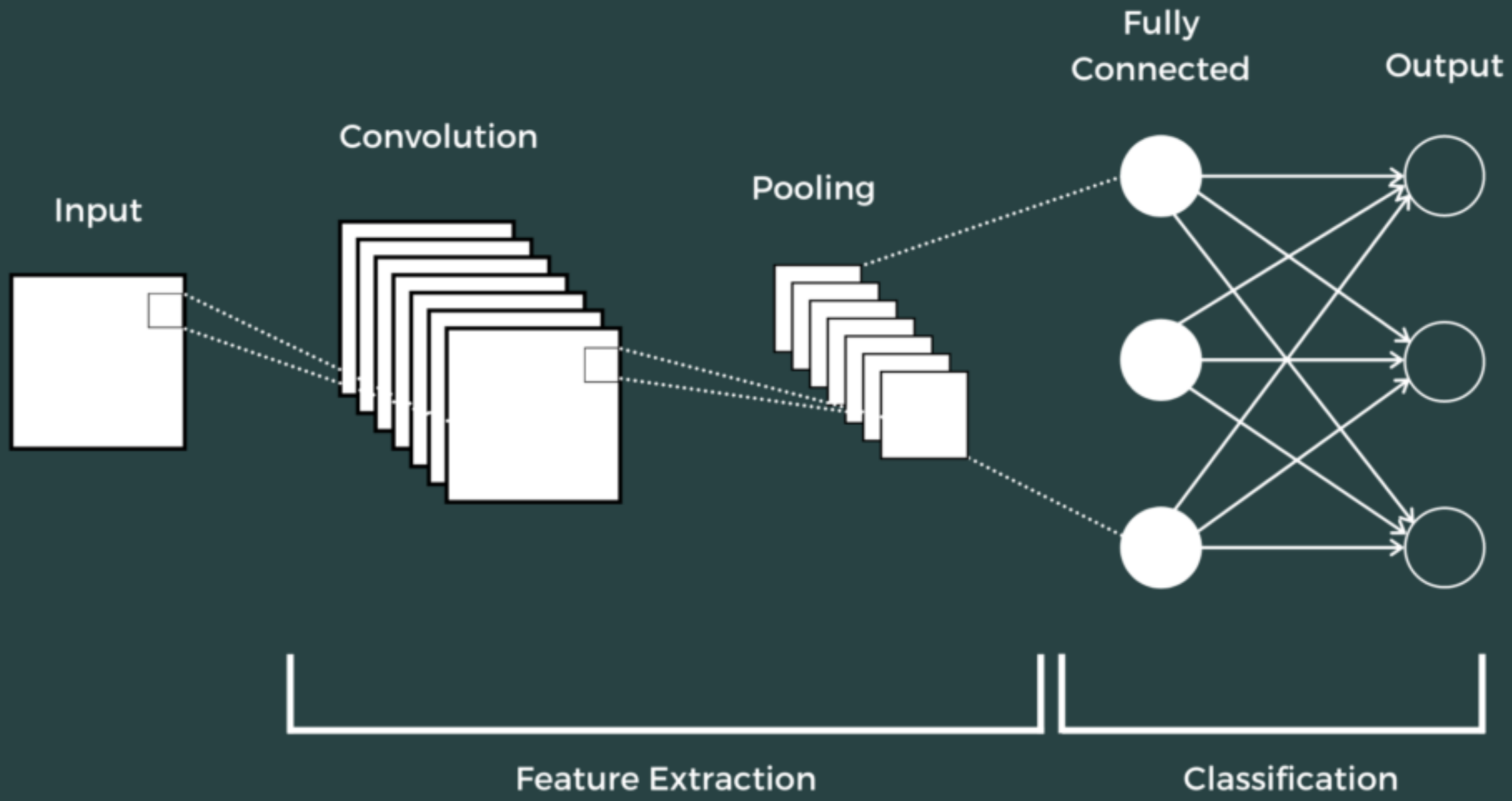
…and we have a **green squiggle** that fits the data.

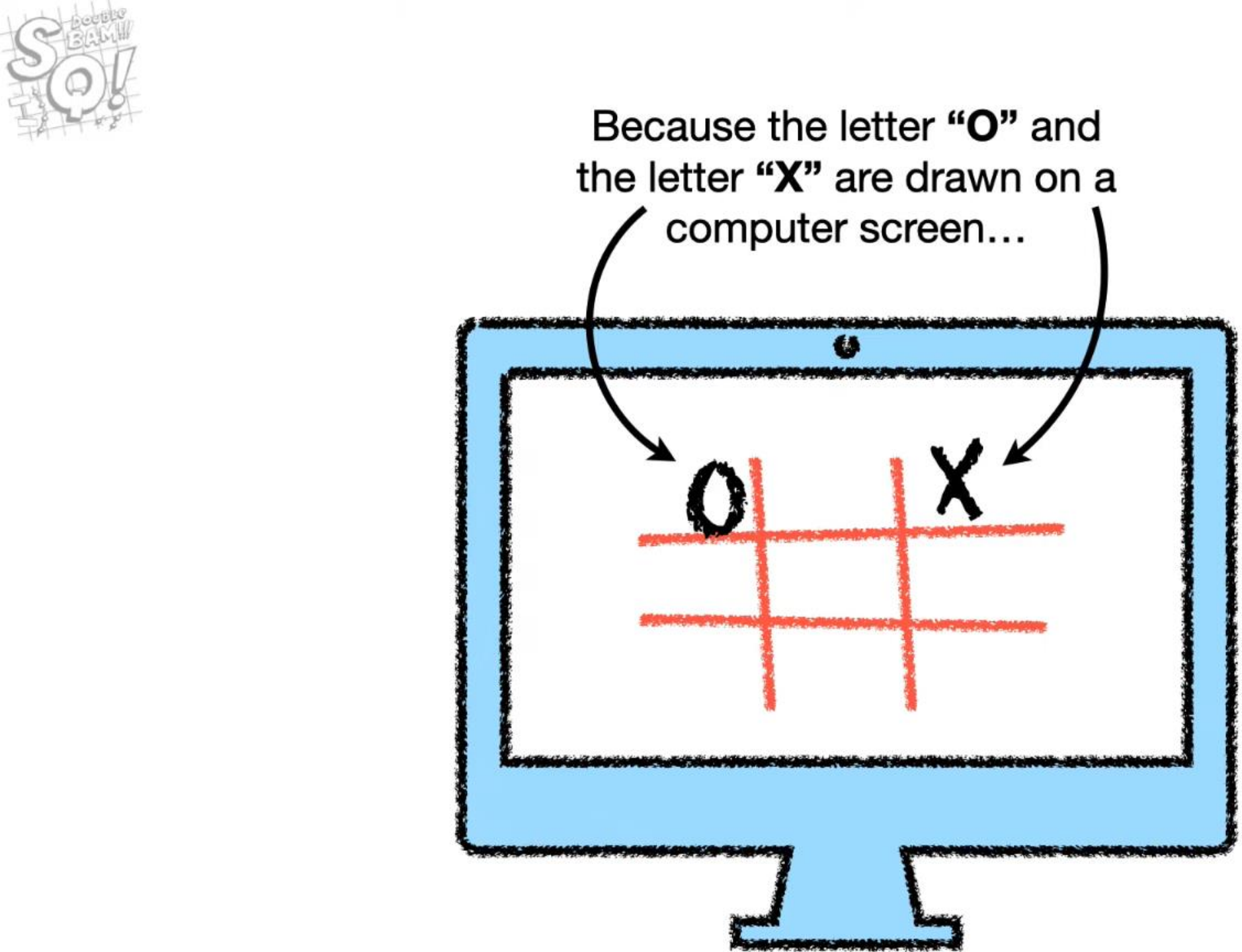




Convolutional Neural Network!!!

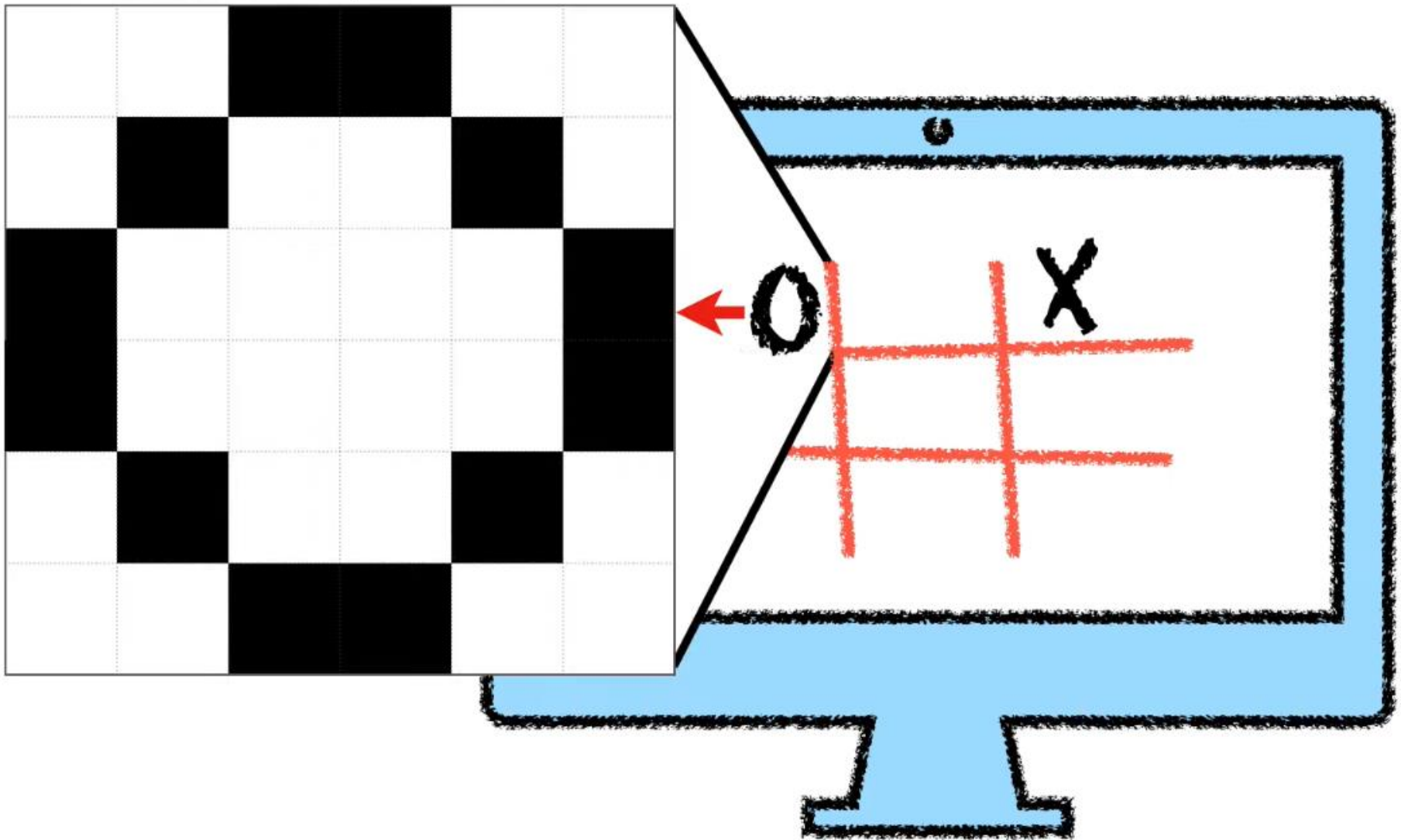



Input
Convolution
Pooling
Fully
Connected
Output
Feature Extraction
Classification


Because the letter “O” and the letter “X” are drawn on a computer screen…
O
X

…we can zoom in
on the letter **“O”**…

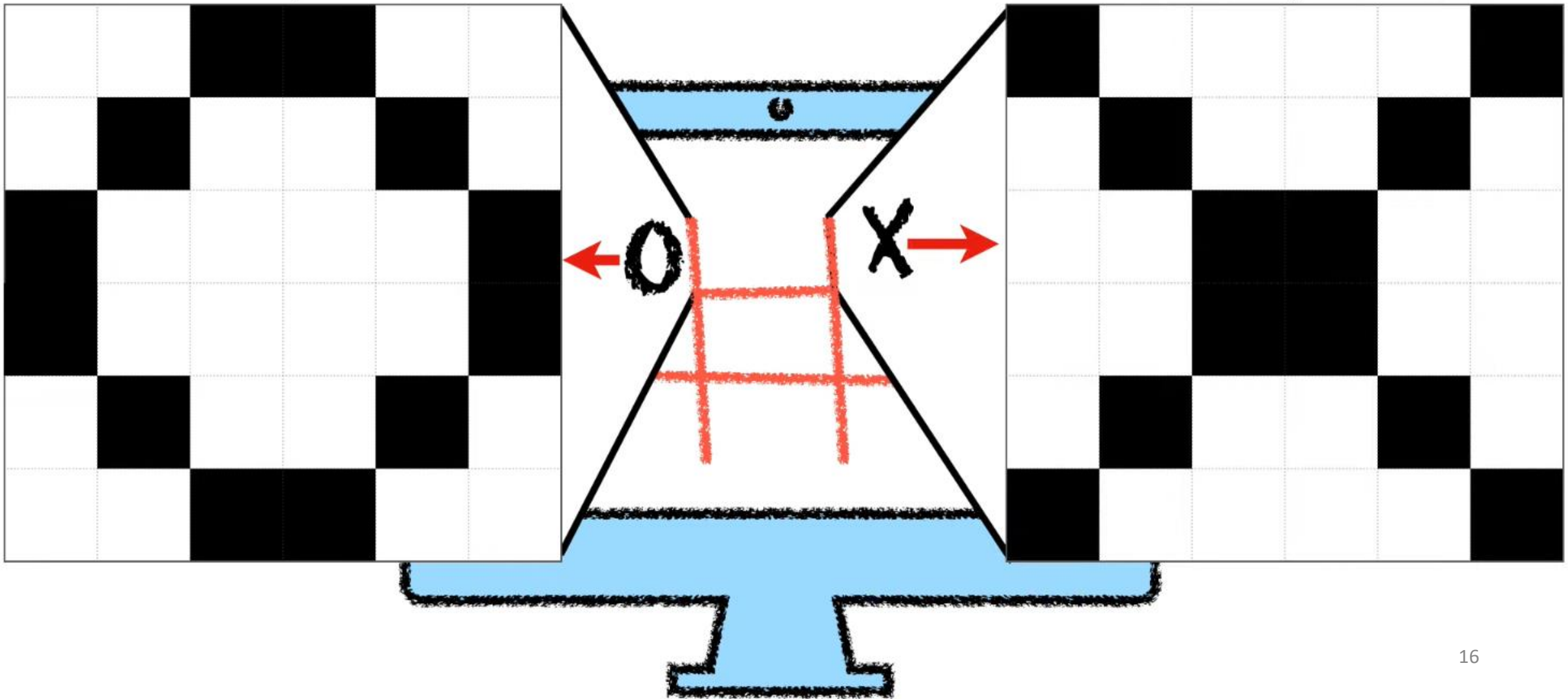
The letter “O”, zoomed in.



…and the letter “X”…

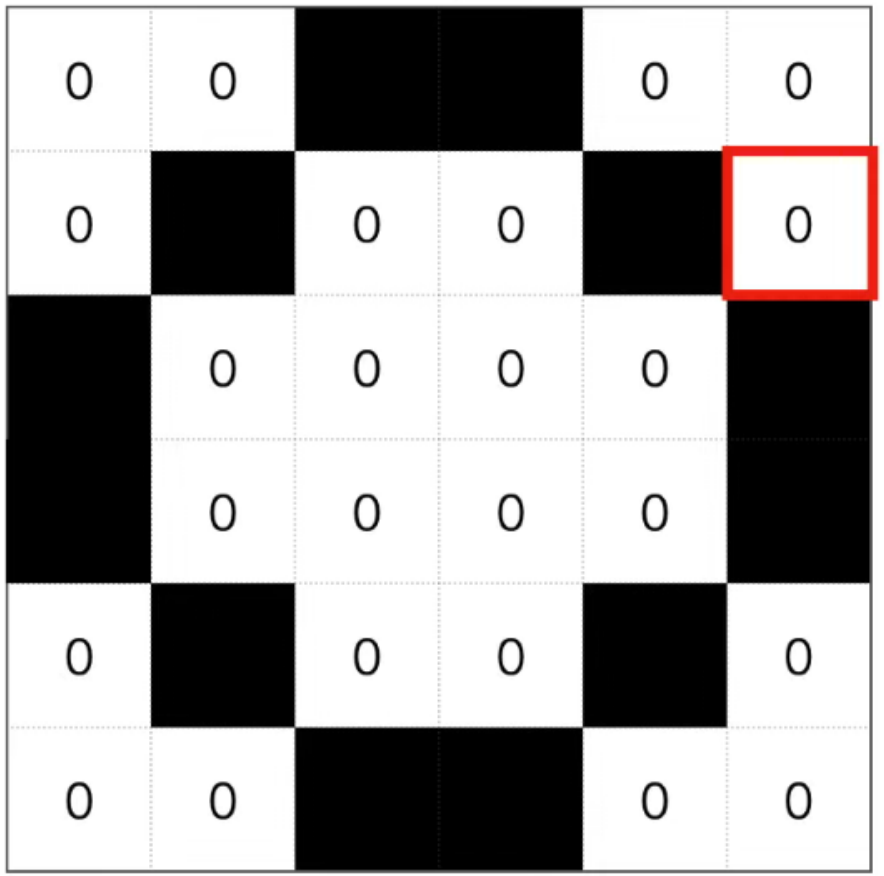
The letter “O”, zoomed in.

The letter “X”, zoomed in.

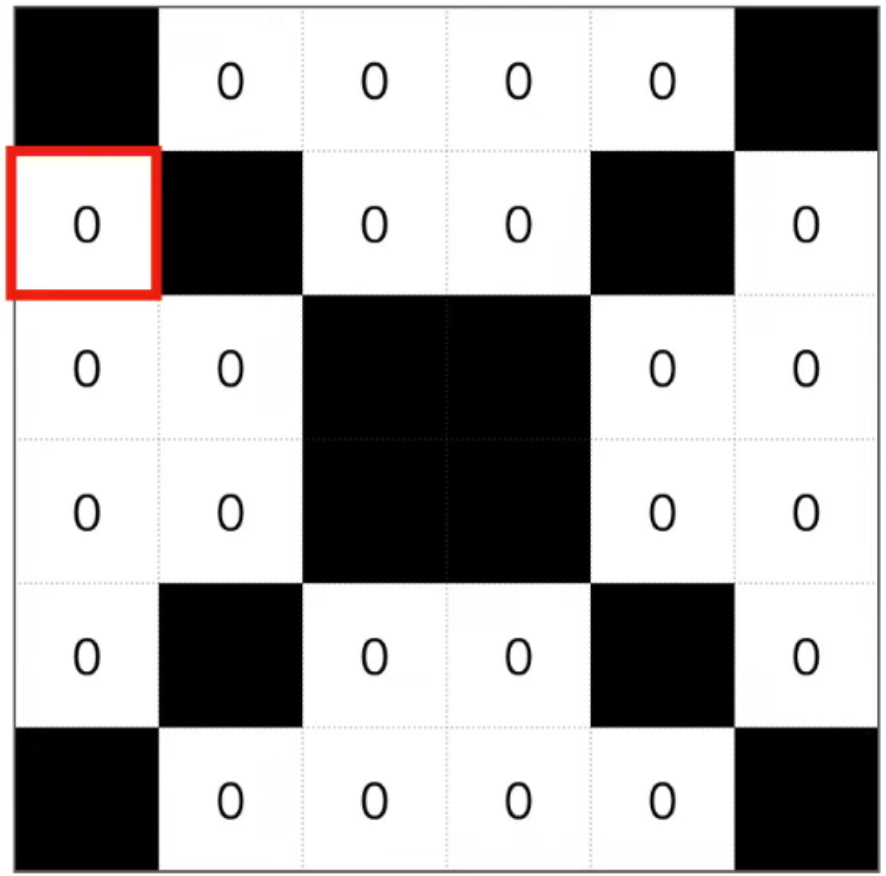


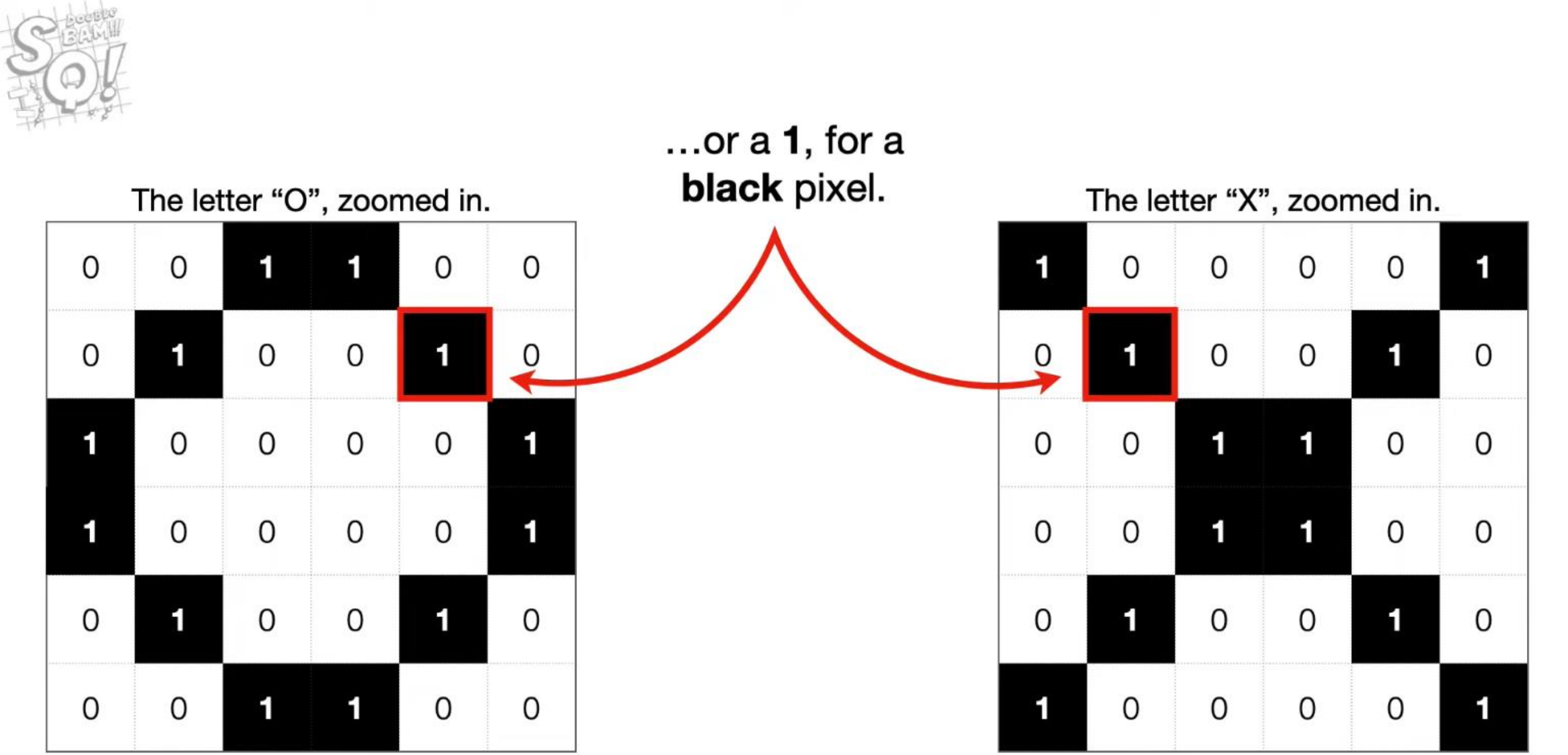
…and, in this case, each pixel is represented by either a **0**, for a white pixel…

The letter “O”, zoomed in.



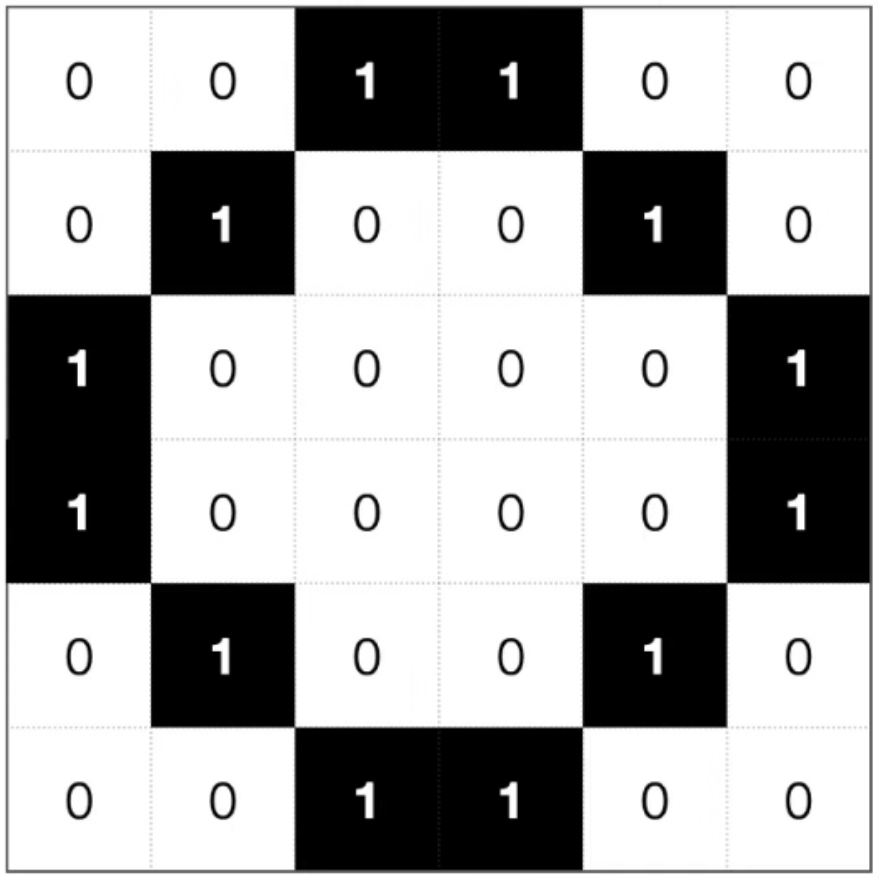
The letter “X”, zoomed in.

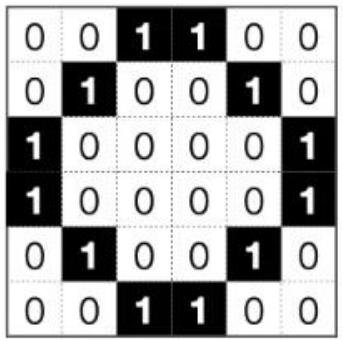


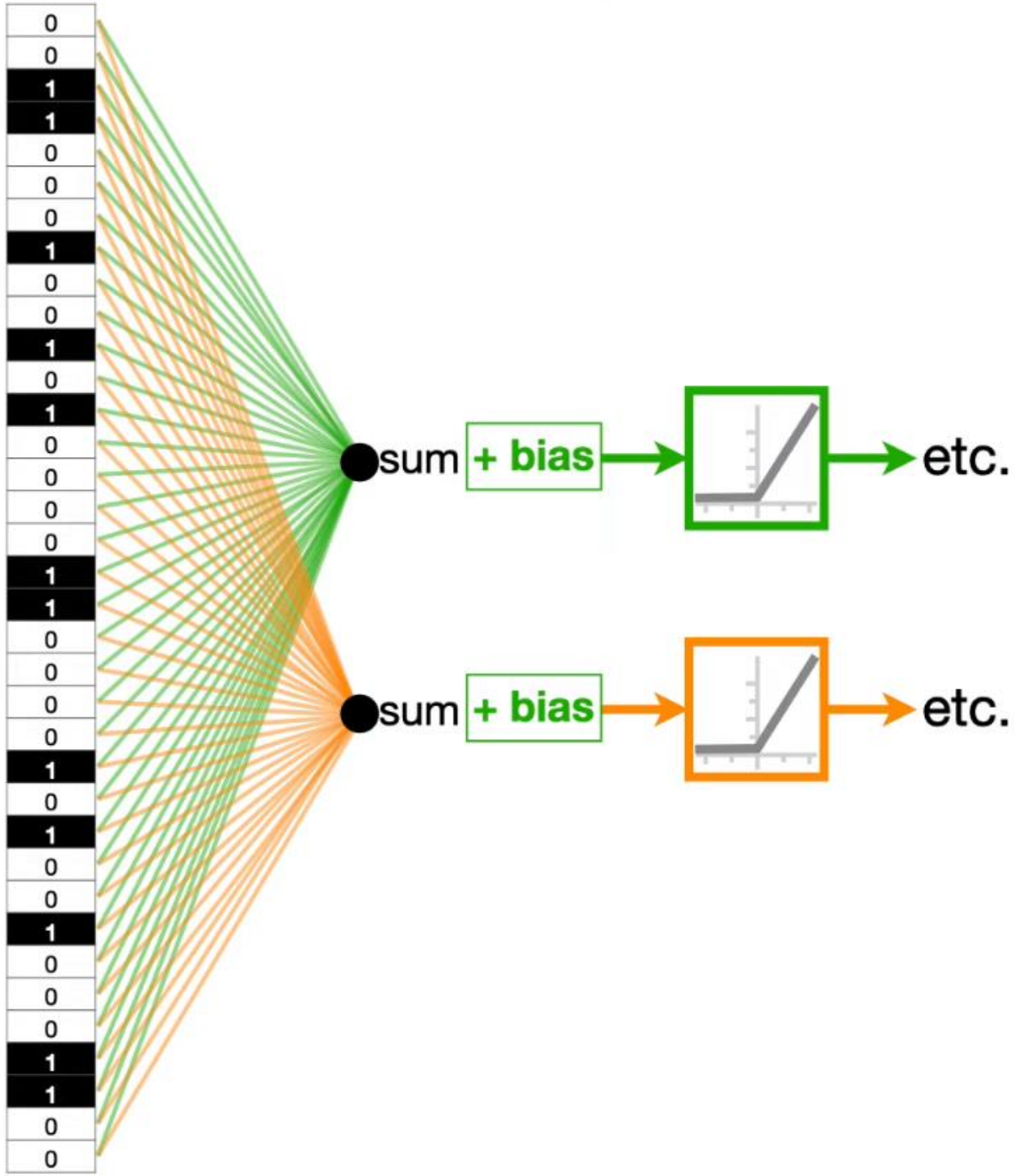

…or a **1**, for a **black** pixel.
The letter “O”, zoomed in.
0 0 1 1 0 0
0 1 0 0 1 0
1 0 0 0 0 1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
The letter “X”, zoomed in.
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
0 0 1 1 0 0
0 1 0 0 1 0
1 0 0 0 0 1

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

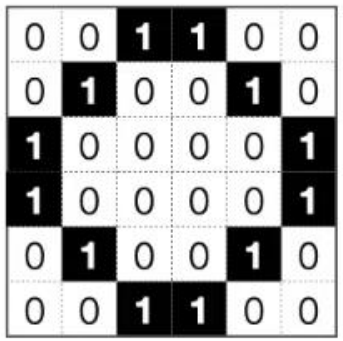
The letter “O”, zoomed in.

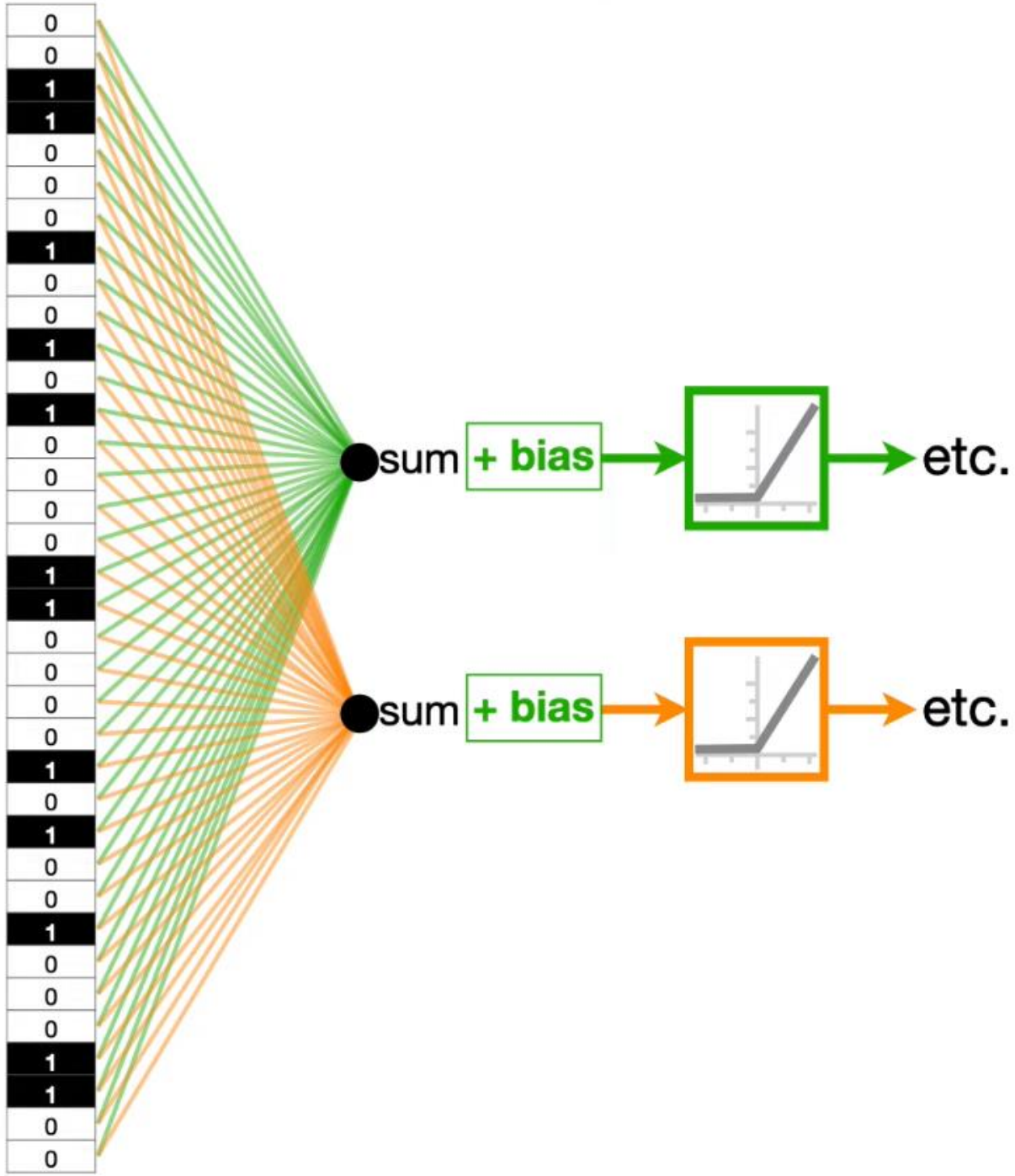




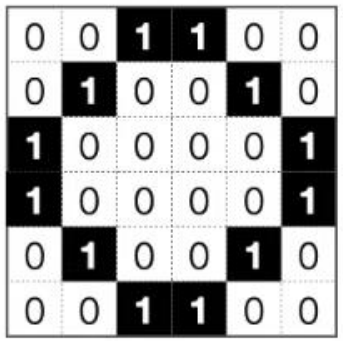


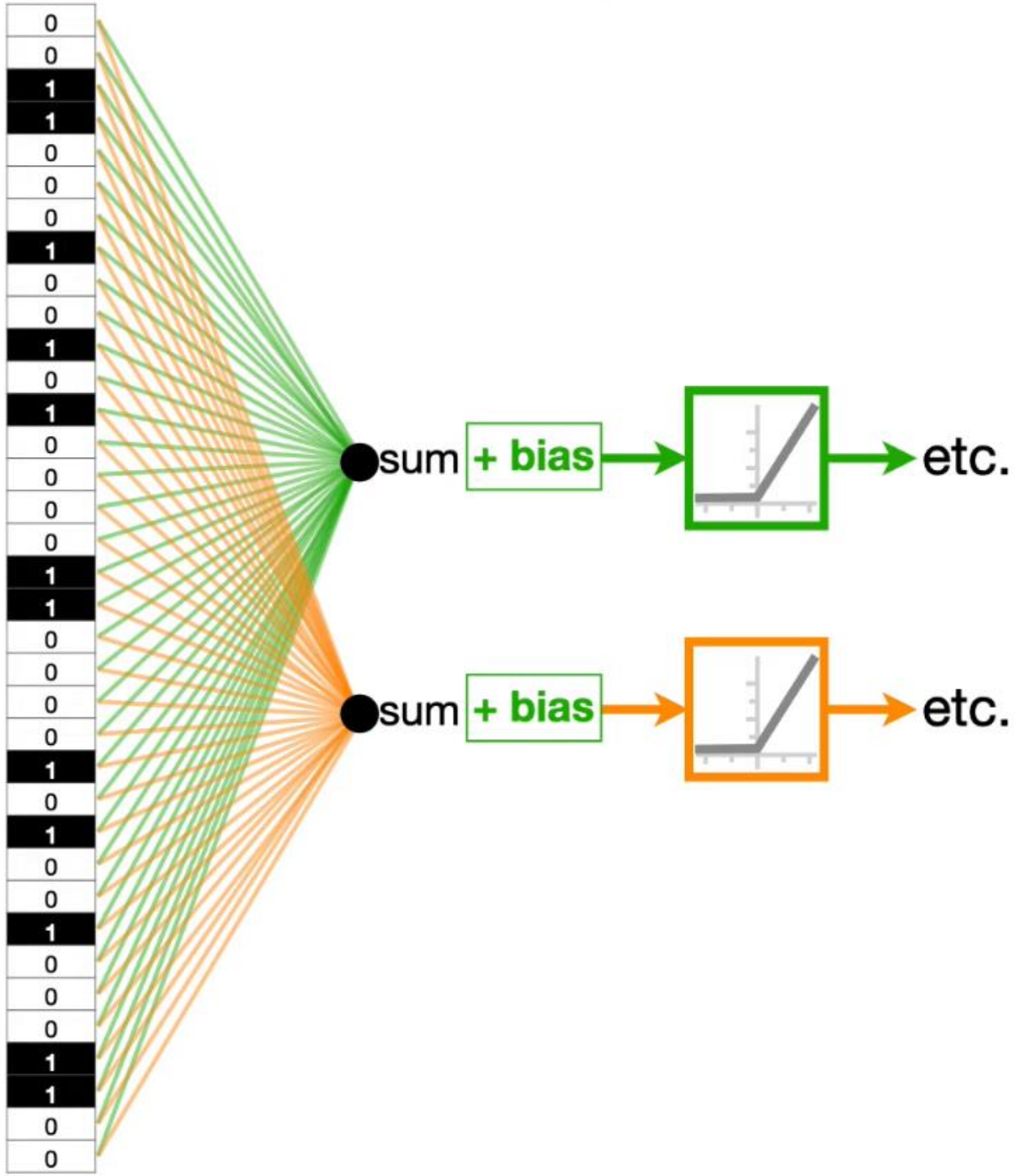
However, if we had a larger image, like **100** pixels by **100** pixels, which is still pretty small compared to real world pictures…





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

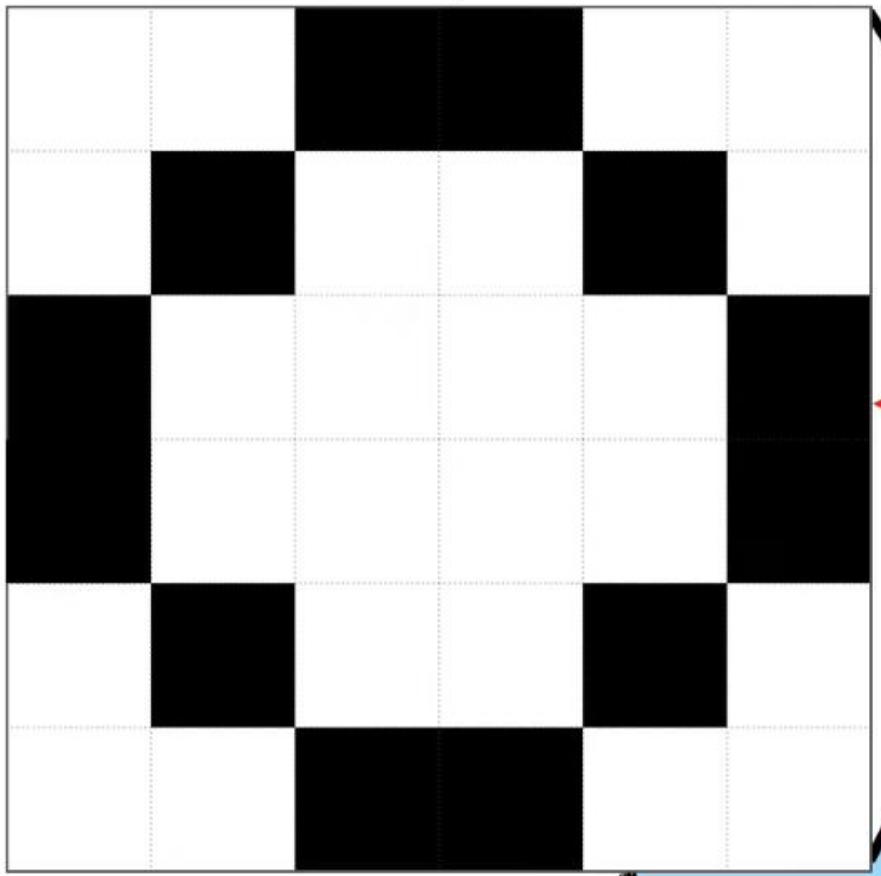


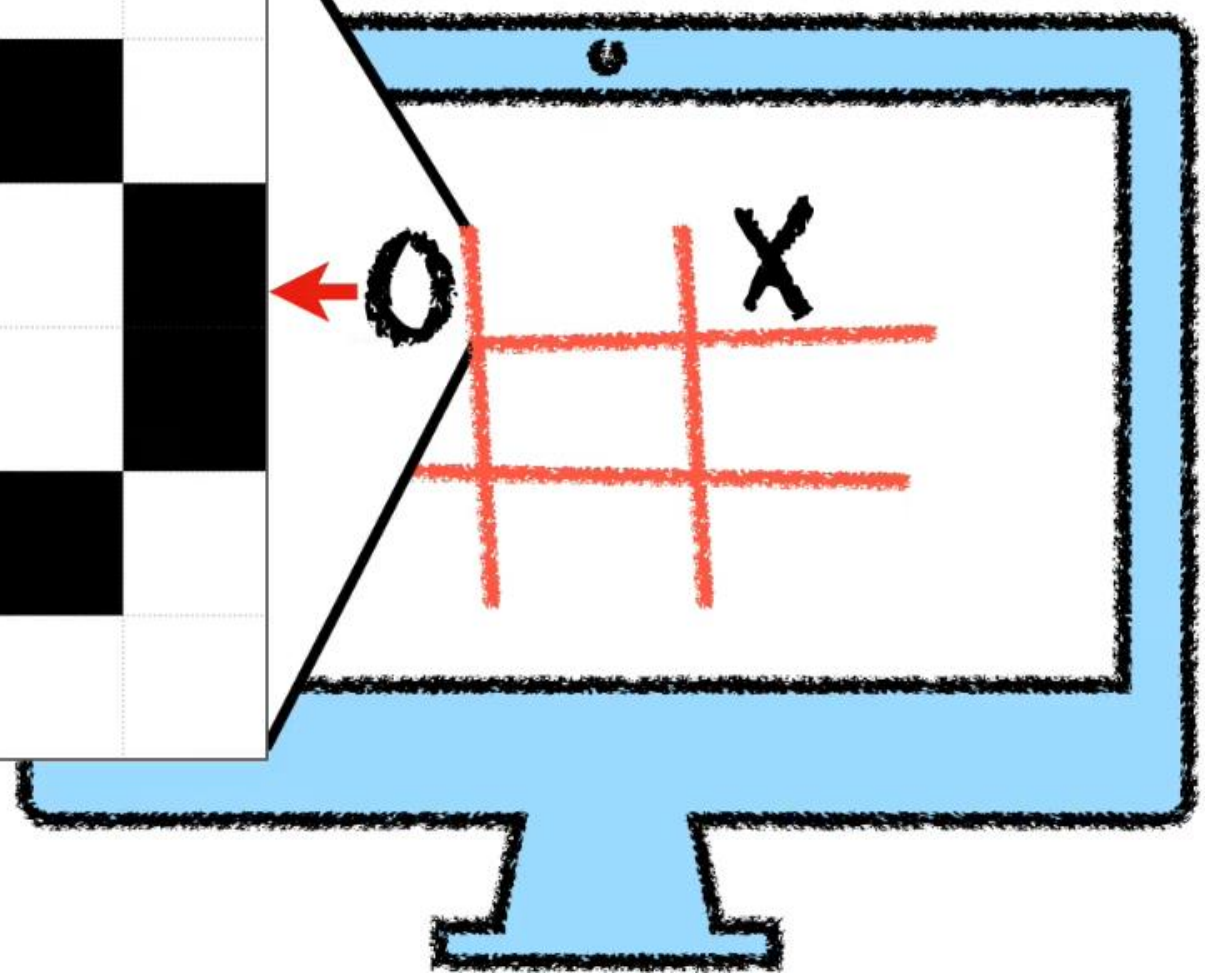


So, this method doesn't scale very well.

…and see how a **Convolutional Neural Network** can recognize this letter **“O”**!!!

The letter “O”, zoomed in.





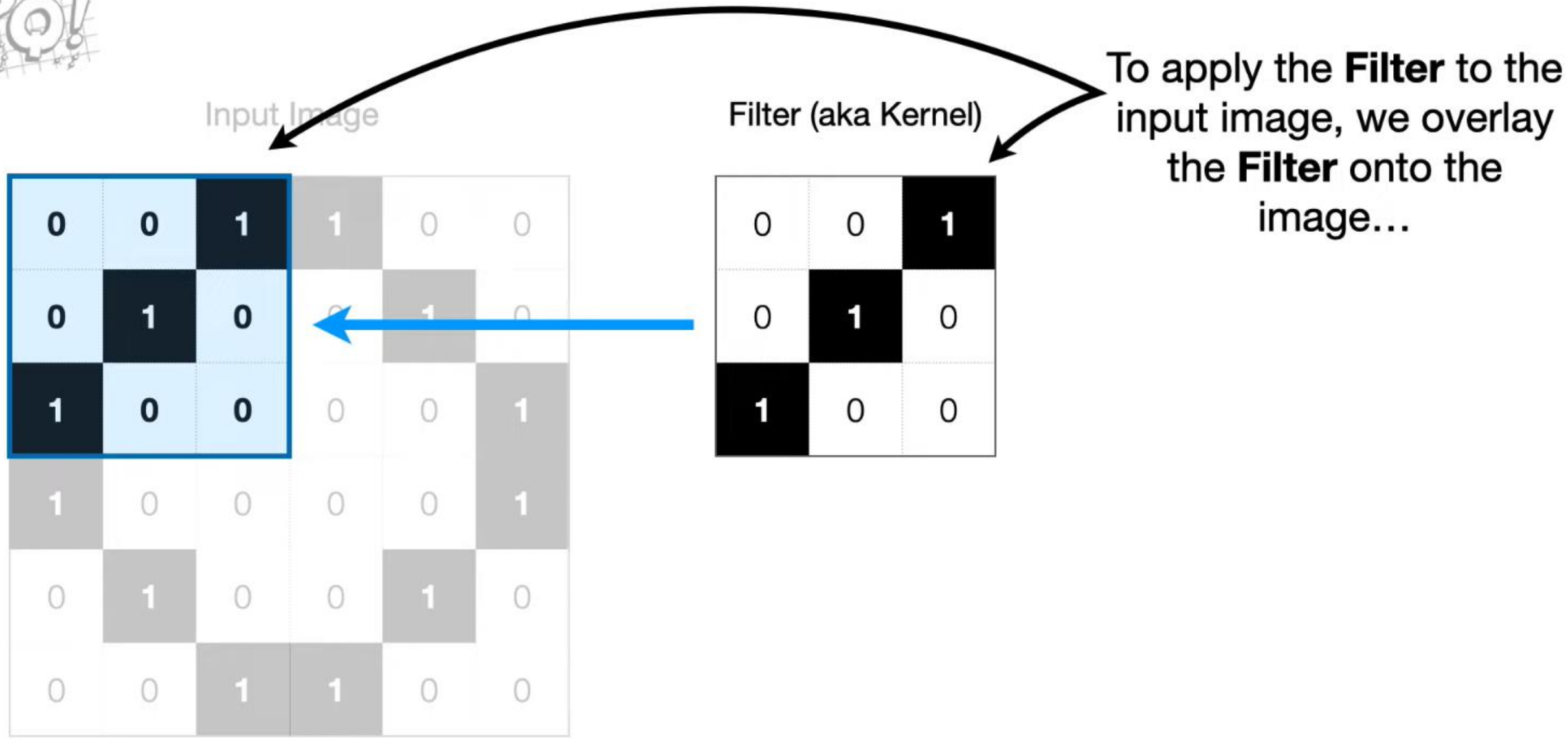
Input Image

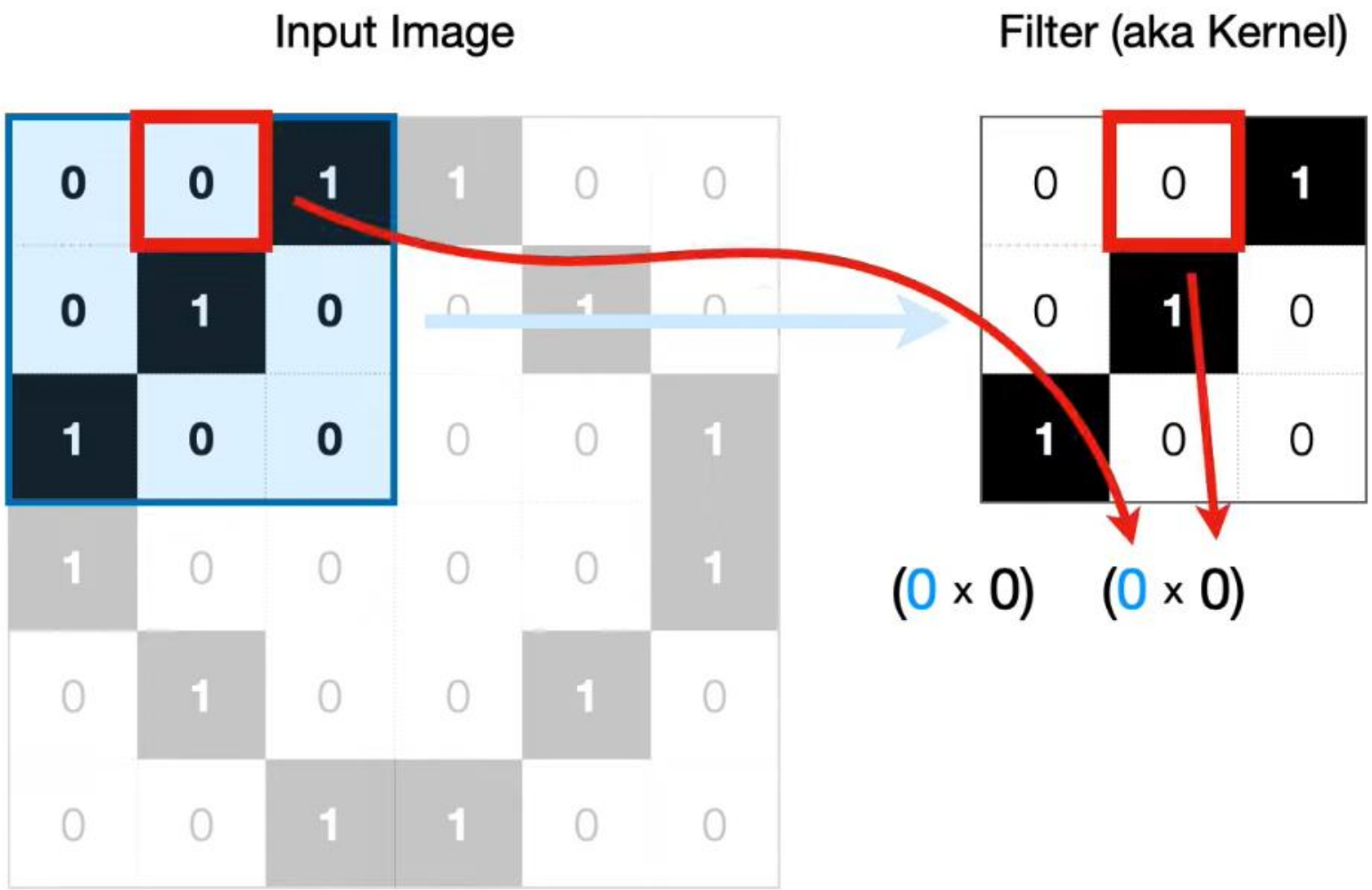
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

Filter (aka Kernel)

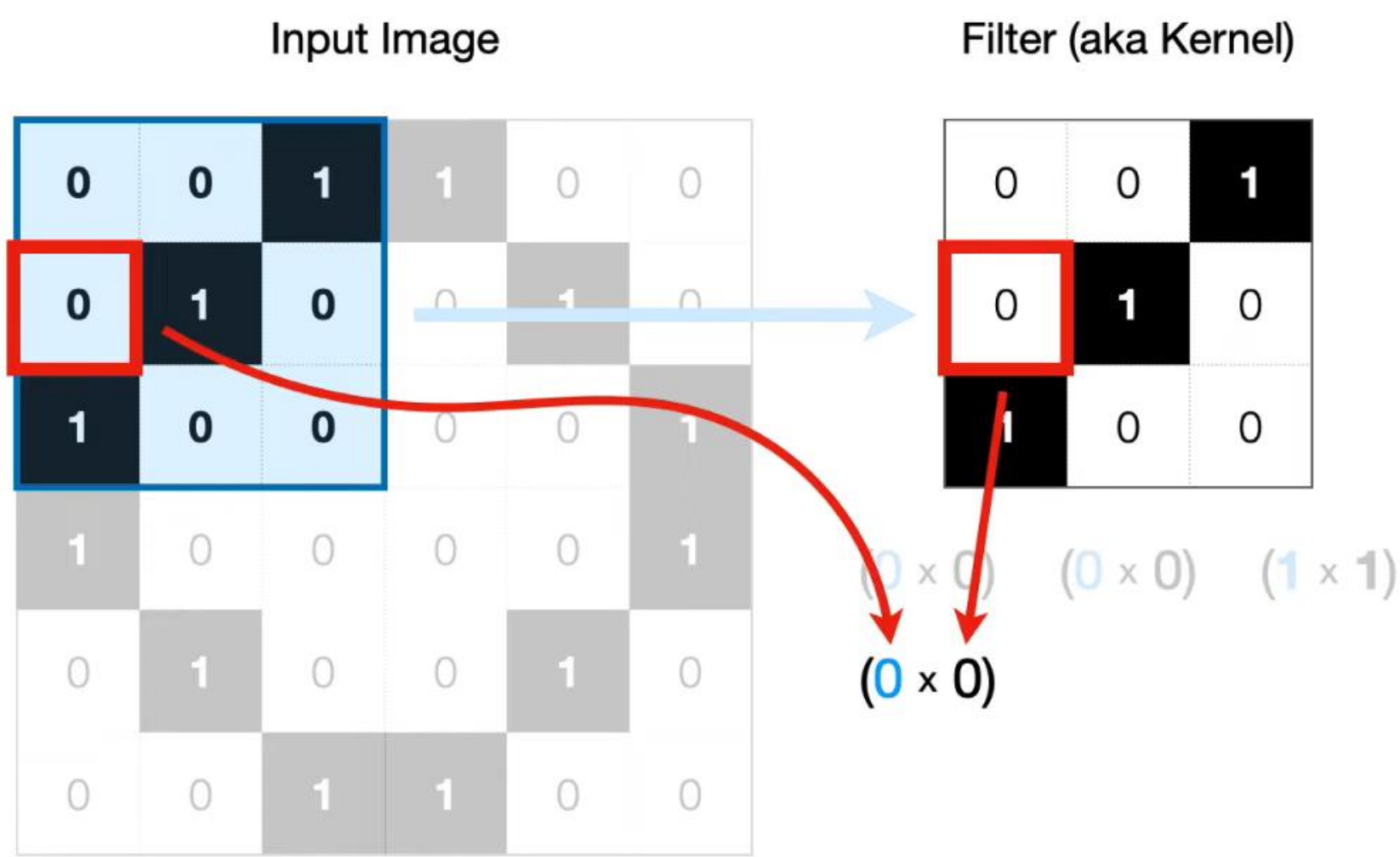
0	0	1
0	1	0
1	0	0

The first thing a **Convolutional Neural Network** does is apply a **Filter** to the **Input Image**.

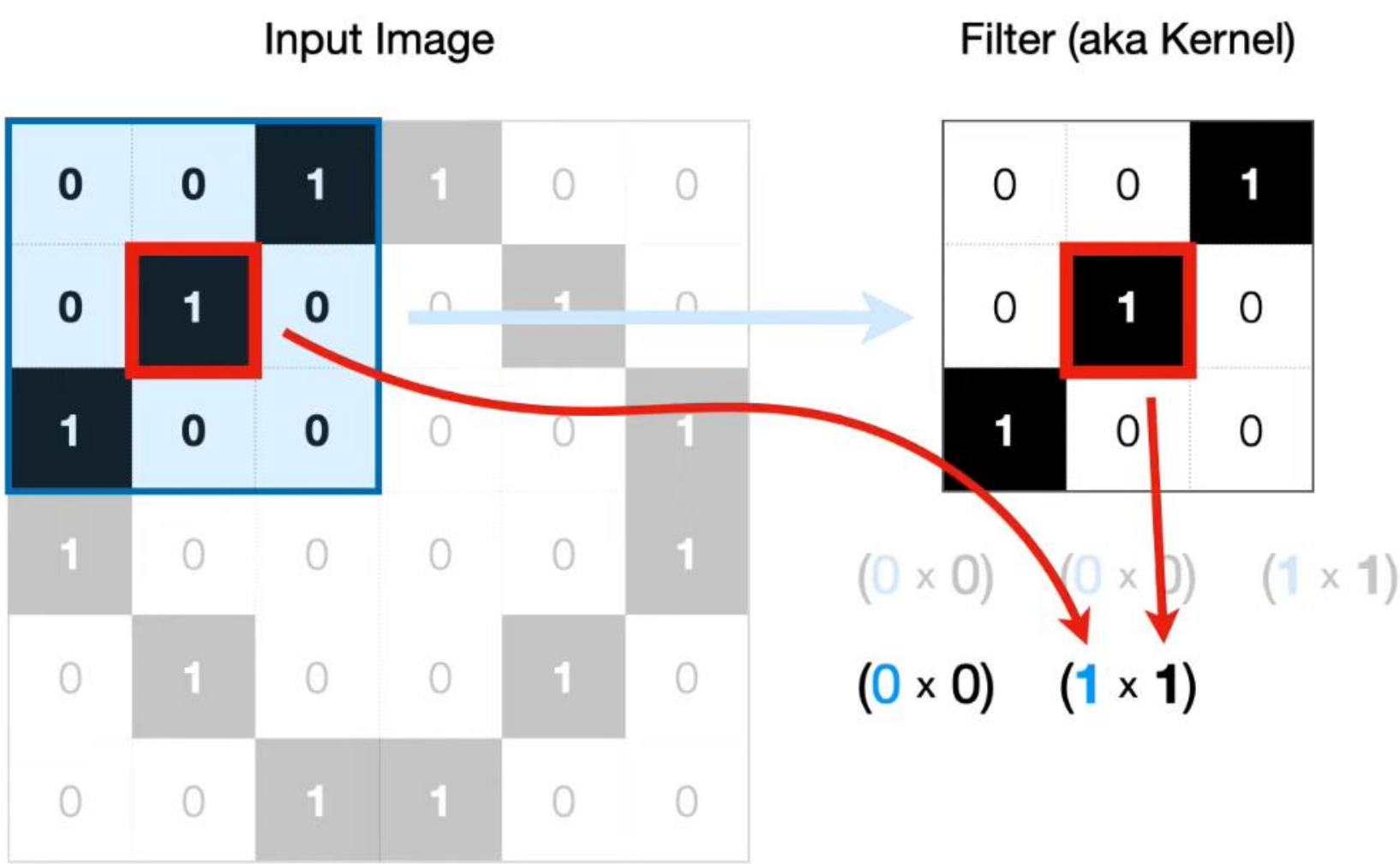

Input Image
Filter (aka Kernel)
To apply the **Filter** to the input image, we overlay the **Filter** onto the image…
0 0 1 1 0 0
0 1 0 0 1 0
1 0 0 0 0 1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
0 0 1
0 1 0
1 0 0



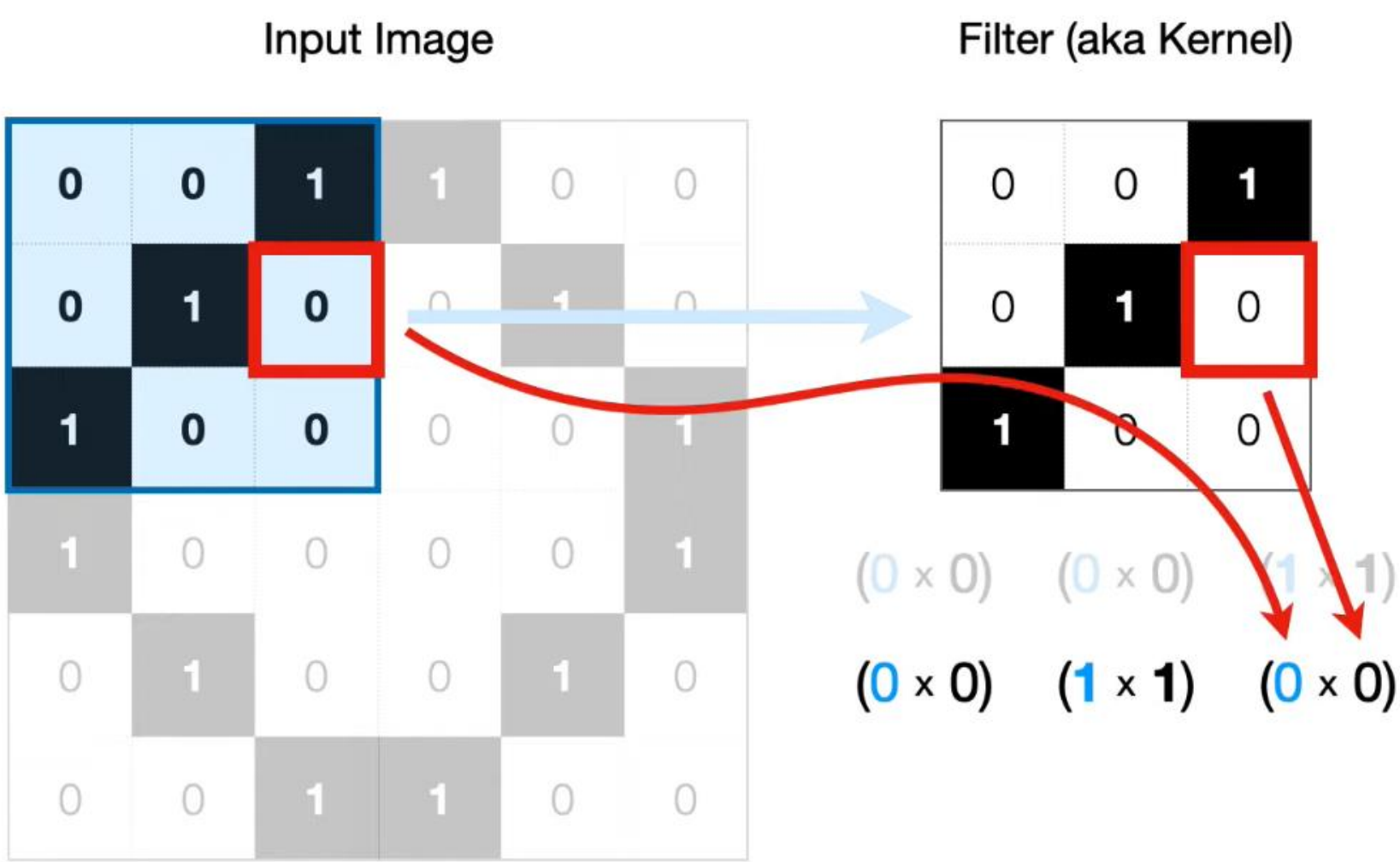
…and then we multiply together each overlapping pixel…



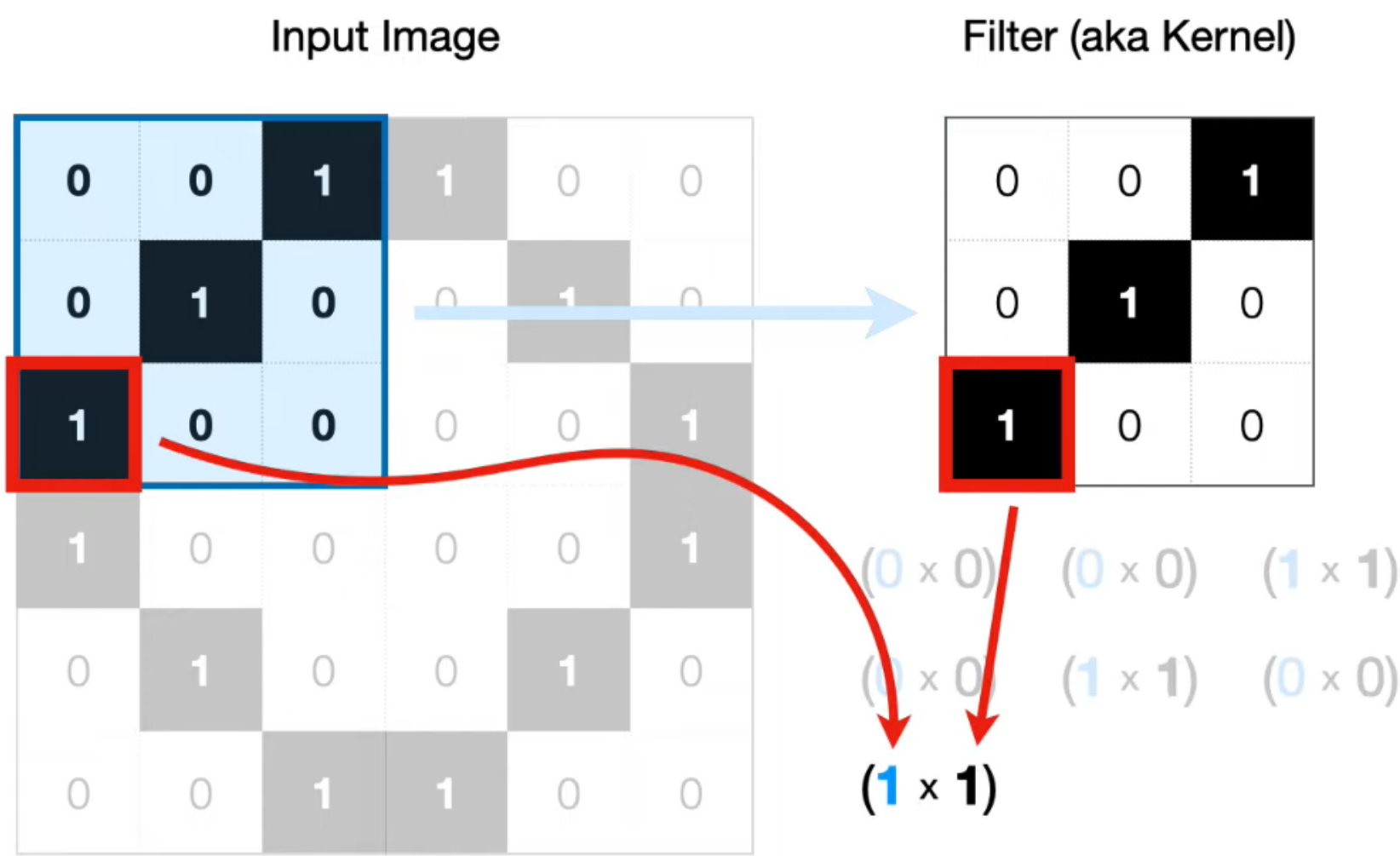
…and then we multiply together each overlapping pixel…



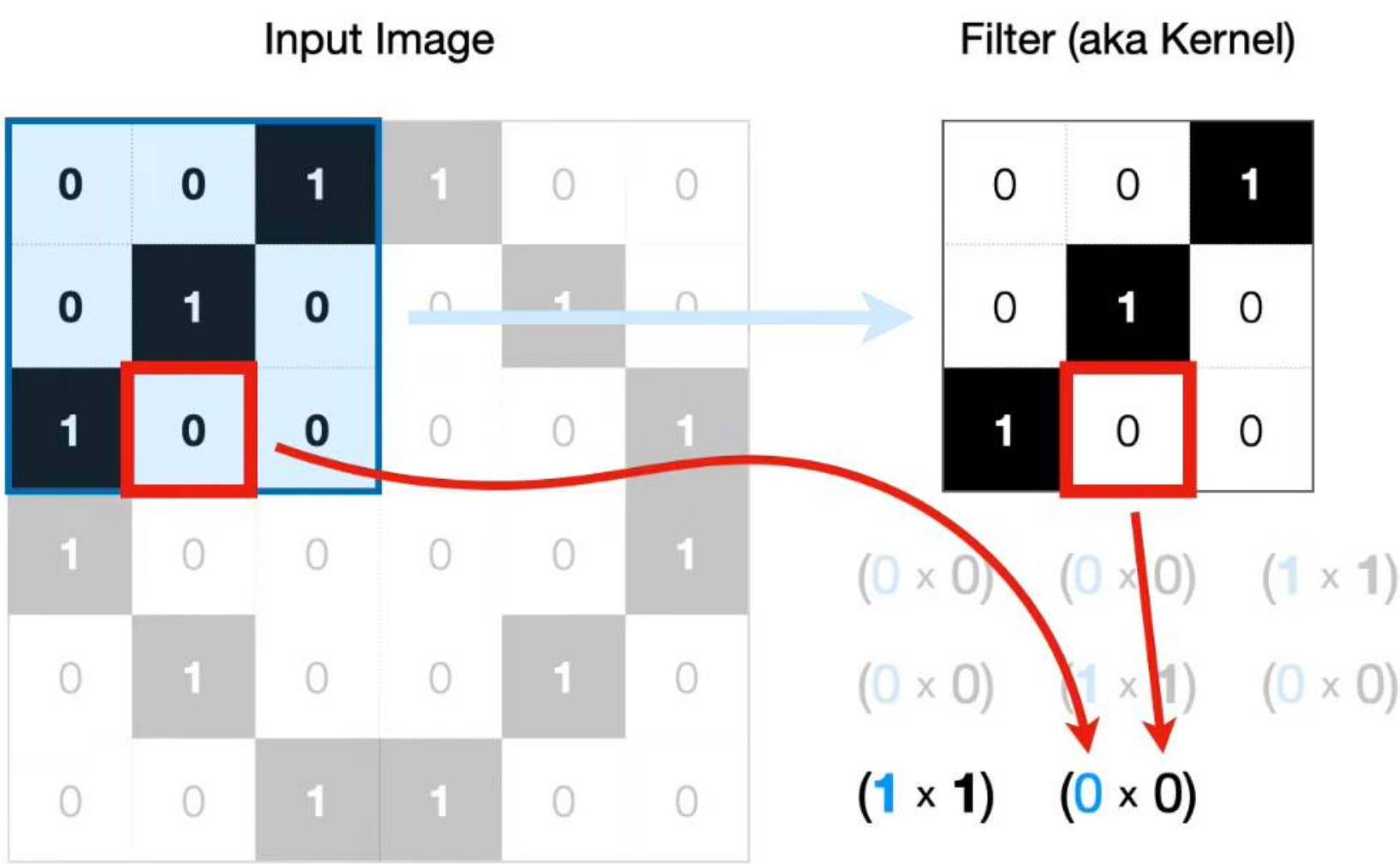
…and then we multiply together each overlapping pixel…



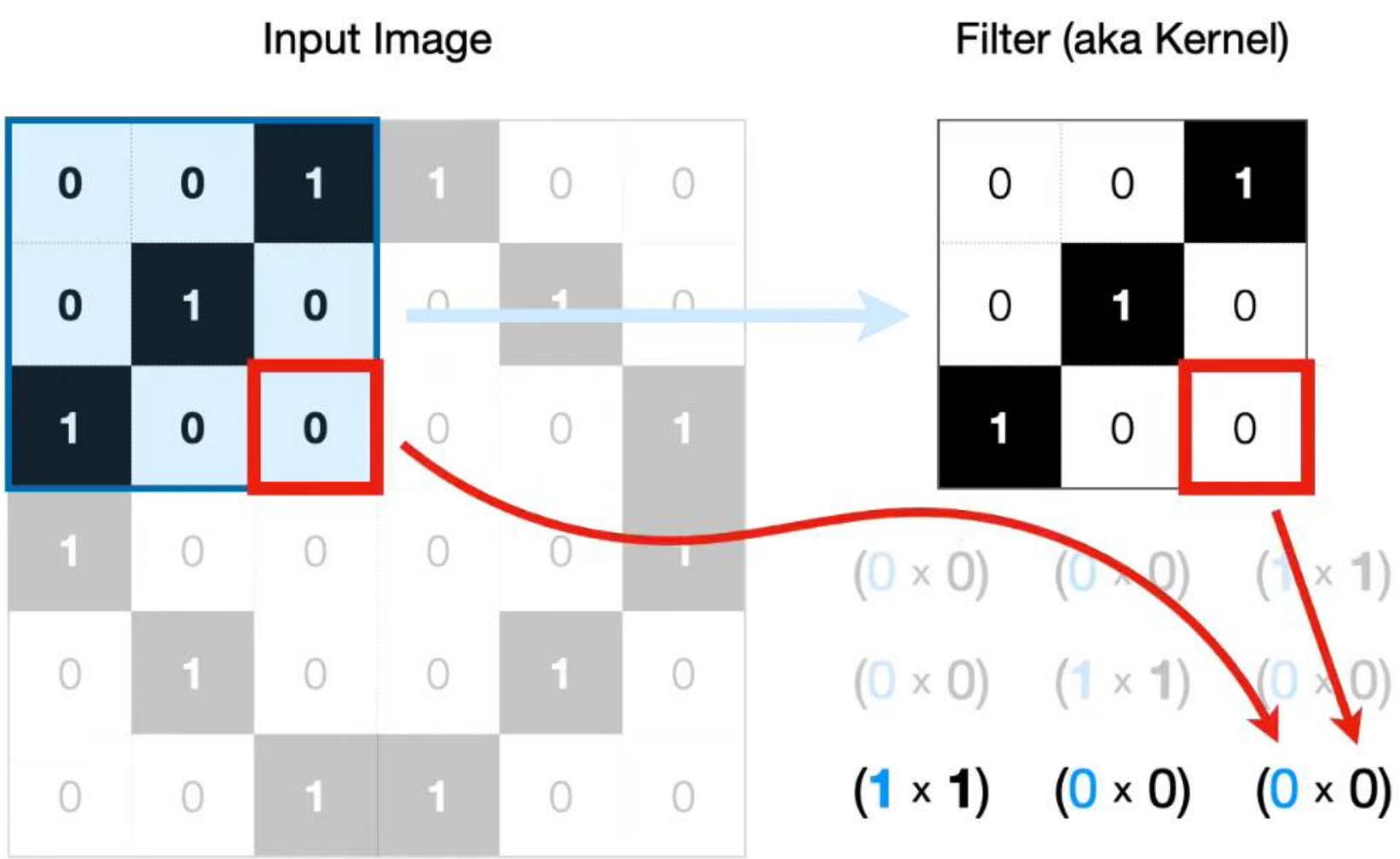
…and then we multiply together each overlapping pixel…



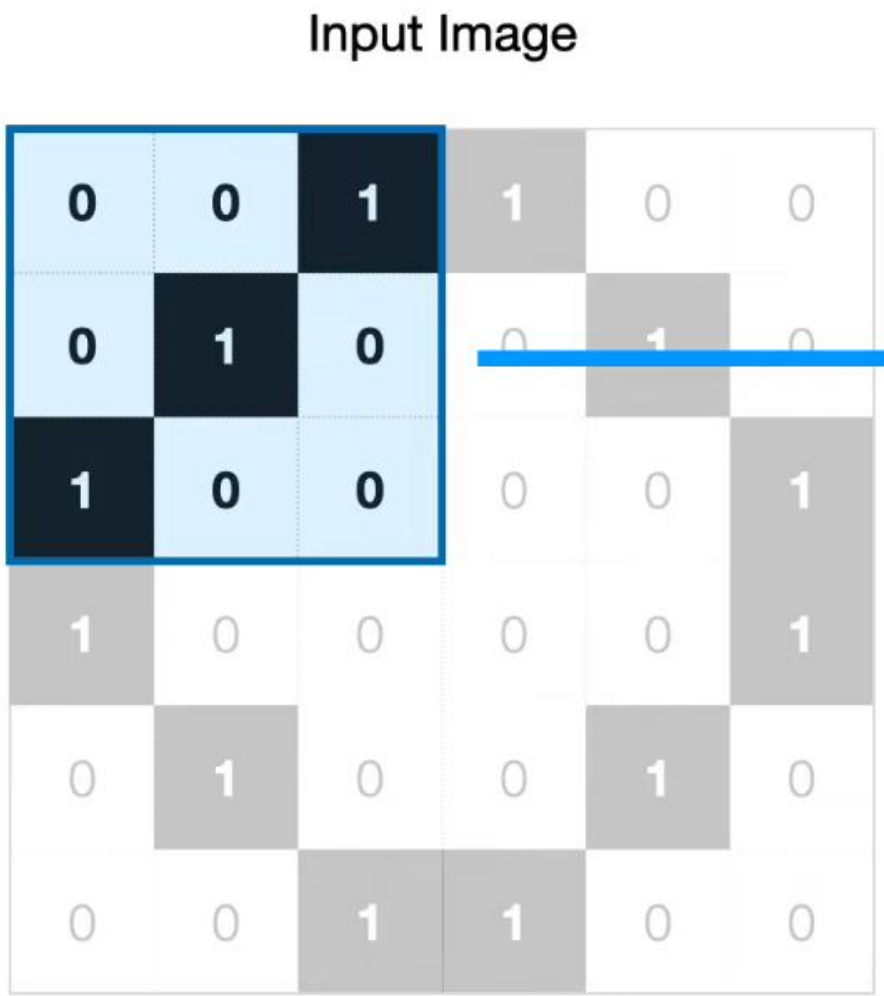
…and then we multiply together each overlapping pixel…

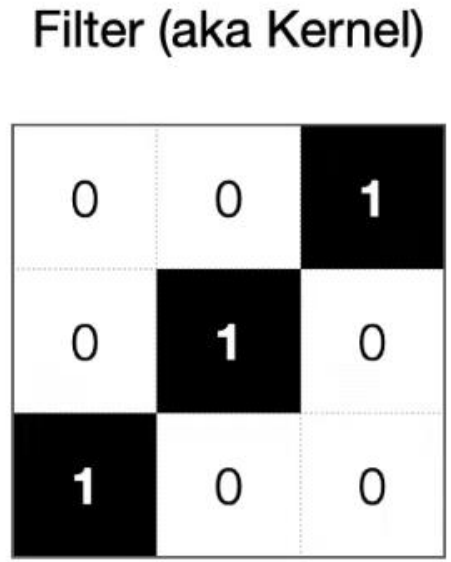


…and then we multiply together each overlapping pixel…



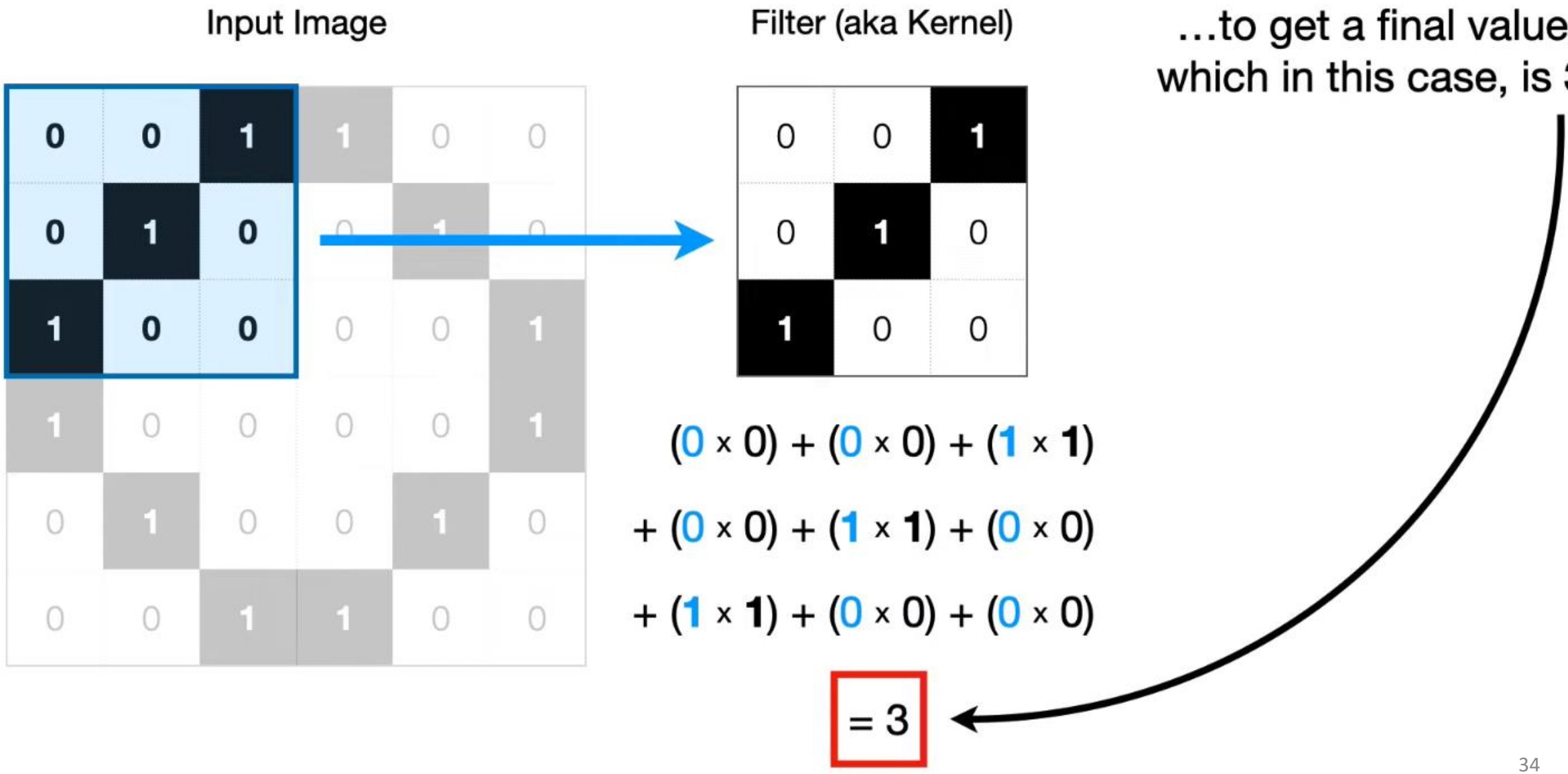
…and then we multiply together each overlapping pixel…

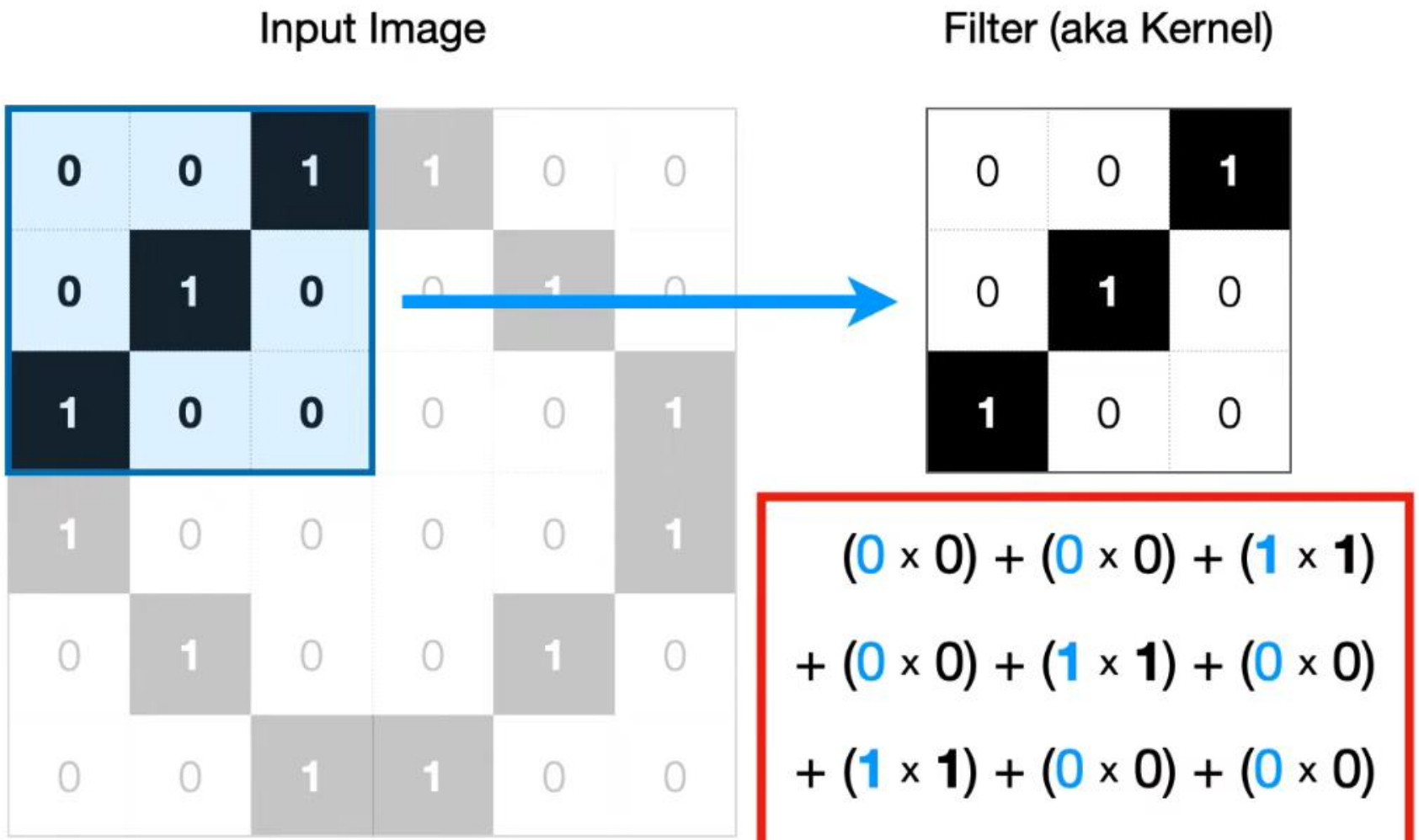




…and then we add each product together…

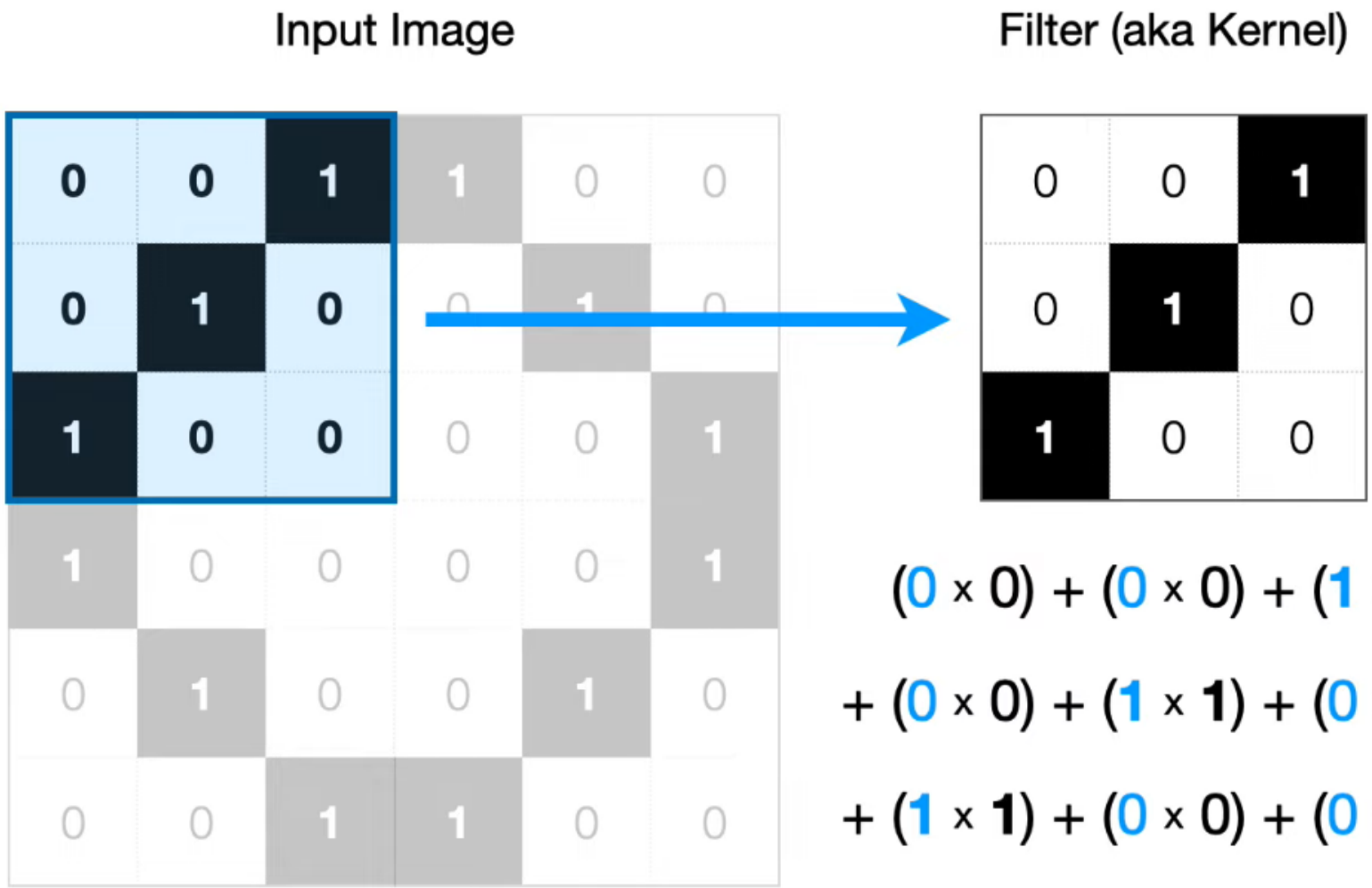
$$(0 \times 0) + (0 \times 0) + (1 \times 1)$$
$$+ (0 \times 0) + (1 \times 1) + (0 \times 0)$$
$$+ (1 \times 1) + (0 \times 0) + (0 \times 0)$$


Input Image
Filter (aka Kernel)
…to get a final value, which in this case, is 3.
(0 × 0) + (0 × 0) + (1 × 1)
+ (0 × 0) + (1 × 1) + (0 × 0)
+ (1 × 1) + (0 × 0) + (0 × 0)
= 3

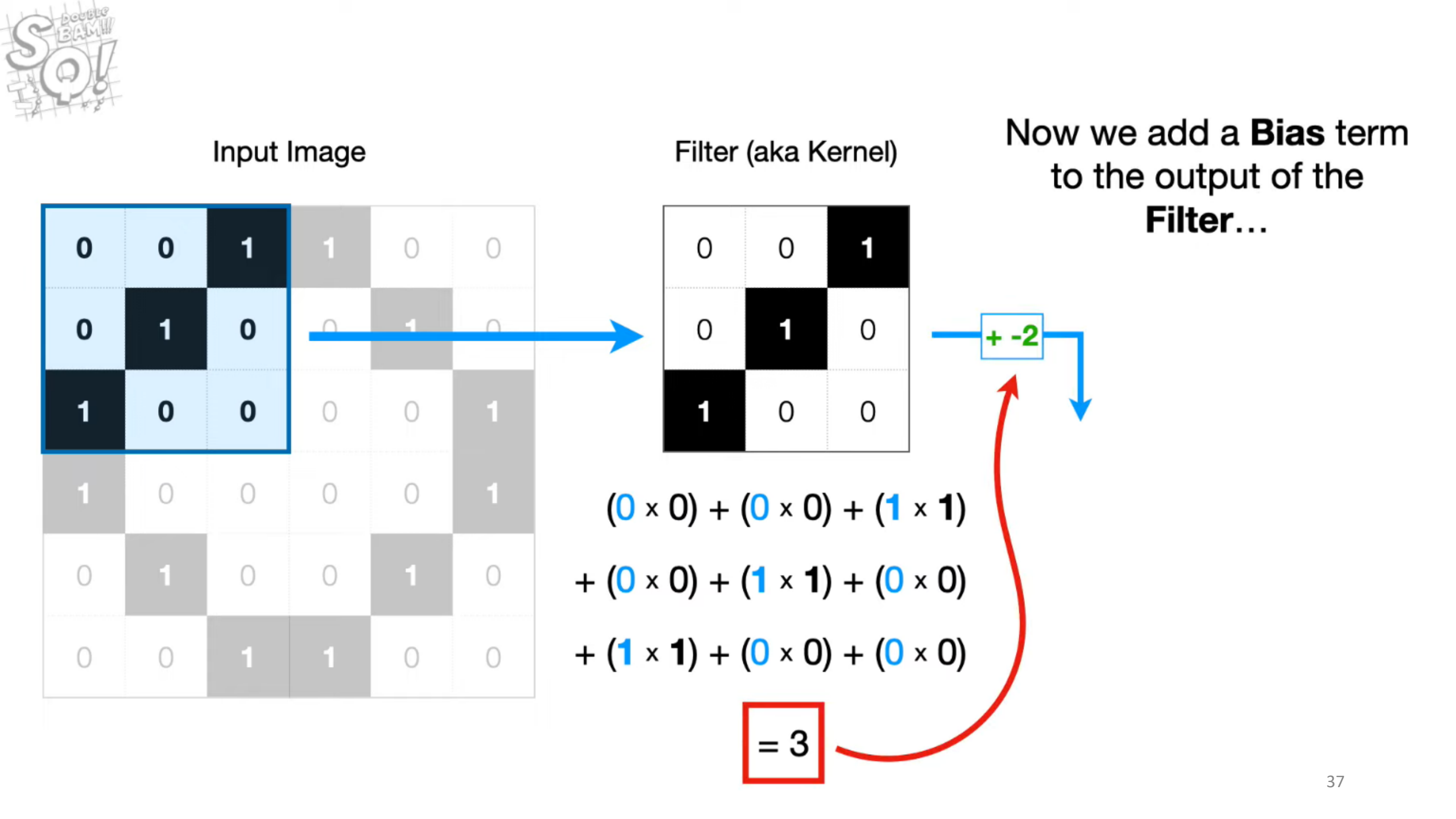


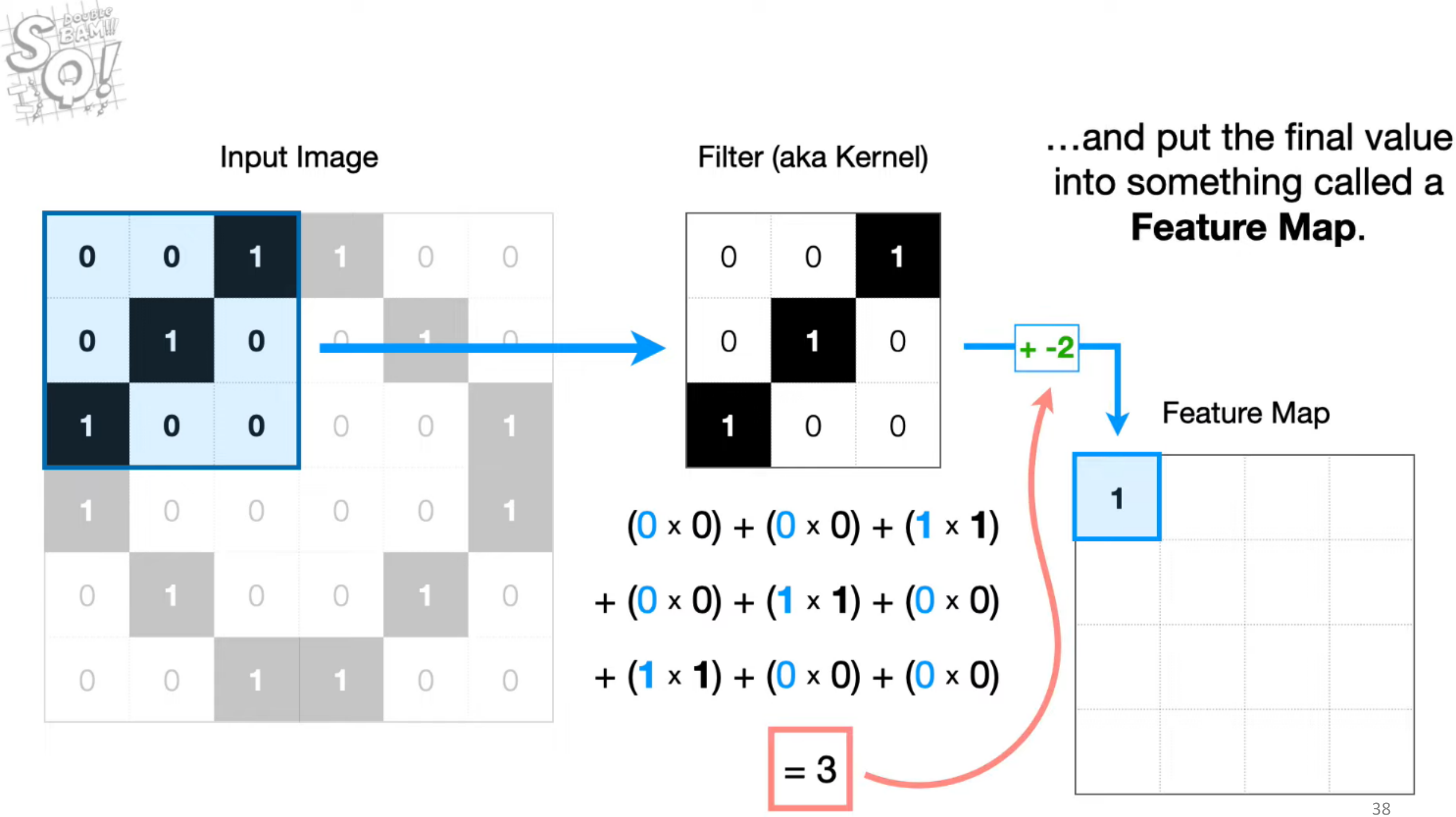
Terminology Alert!!!

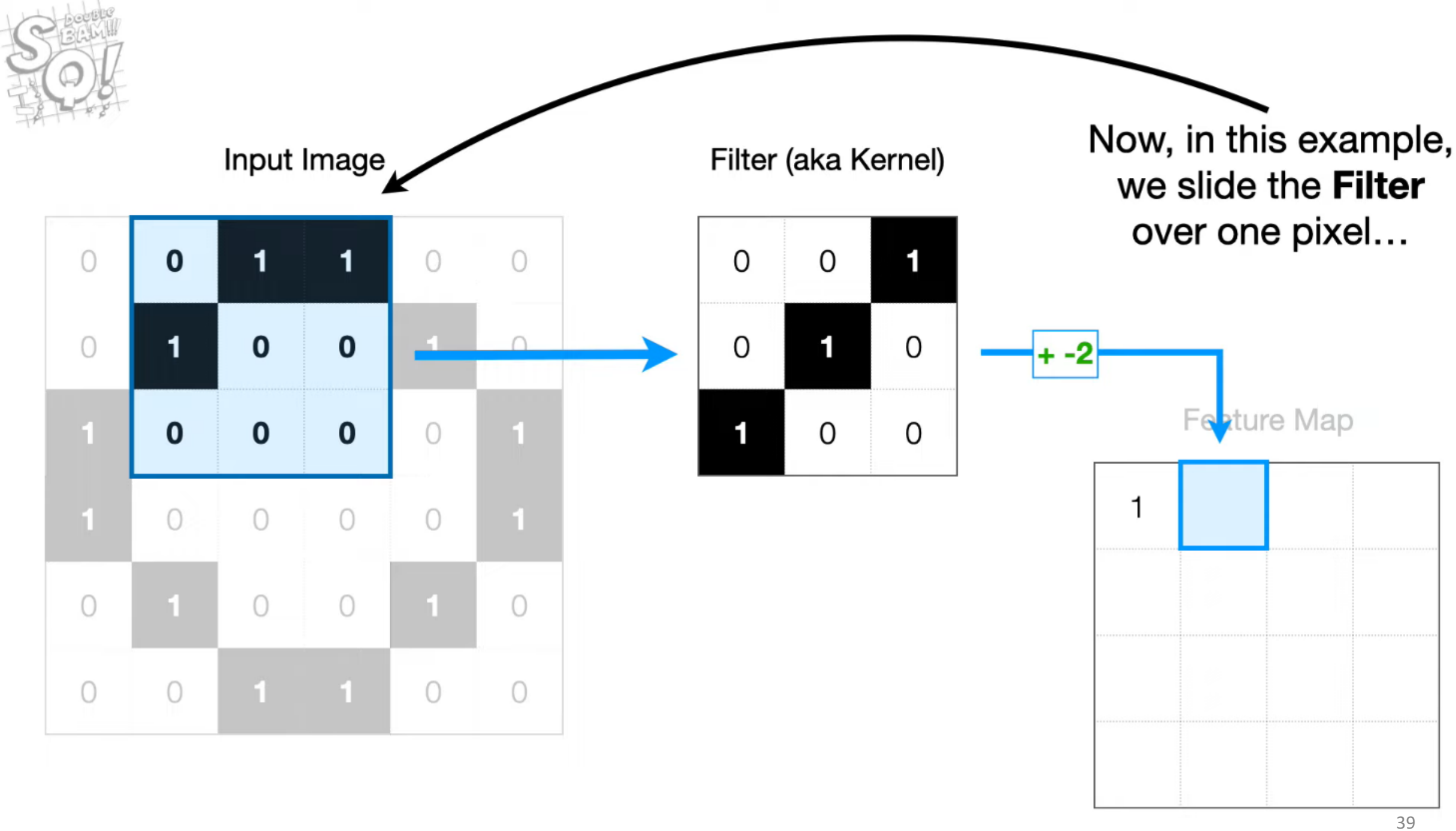
In fancy math lingo, we call this sum of products a **Dot Product**.

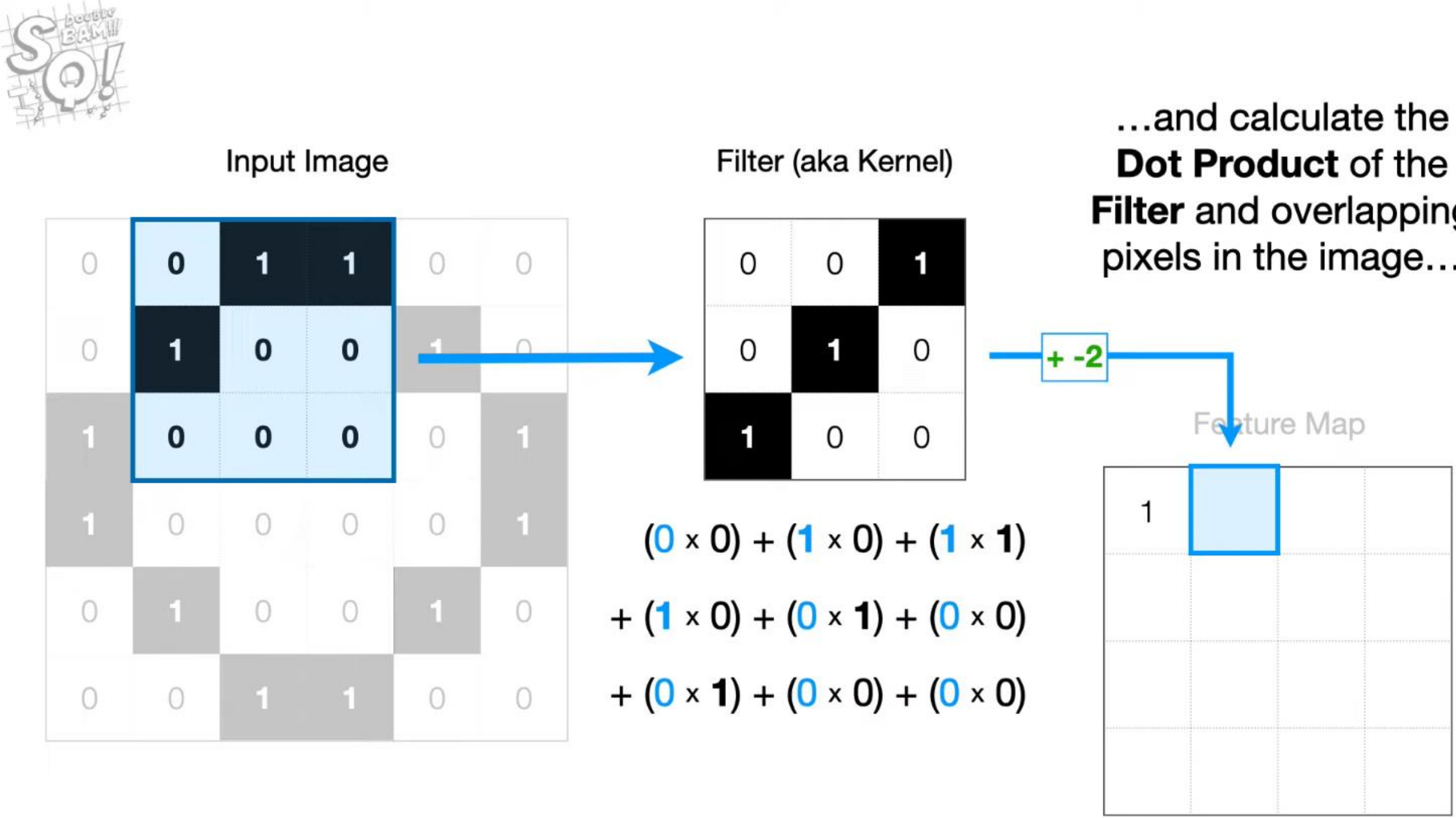


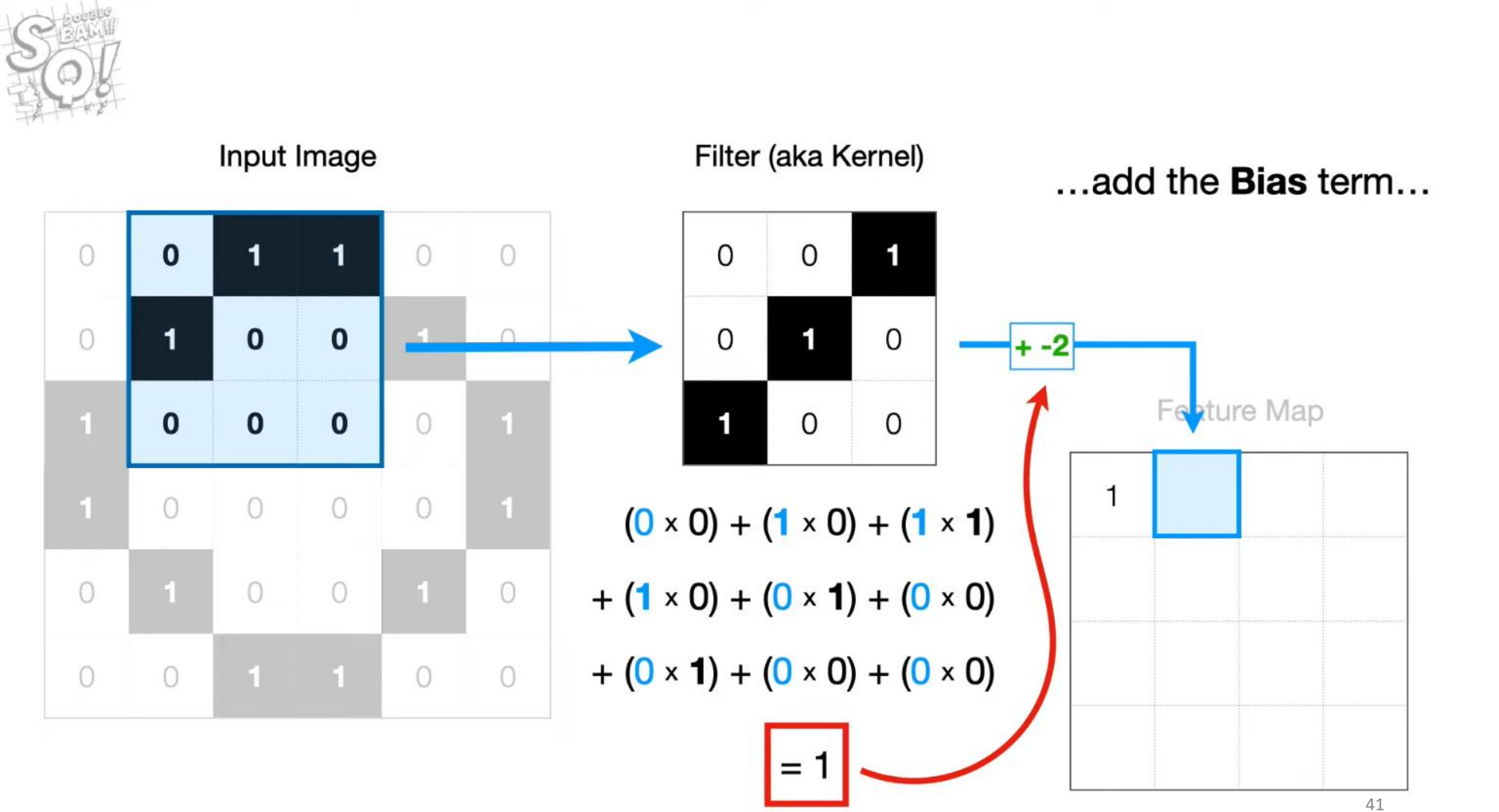
By computing the **Dot Product** between the input and the **Filter**, we can say that the **Filter** is ***Convolved*** with the input, and that's what gives **Convolutional Neural Networks** their name.

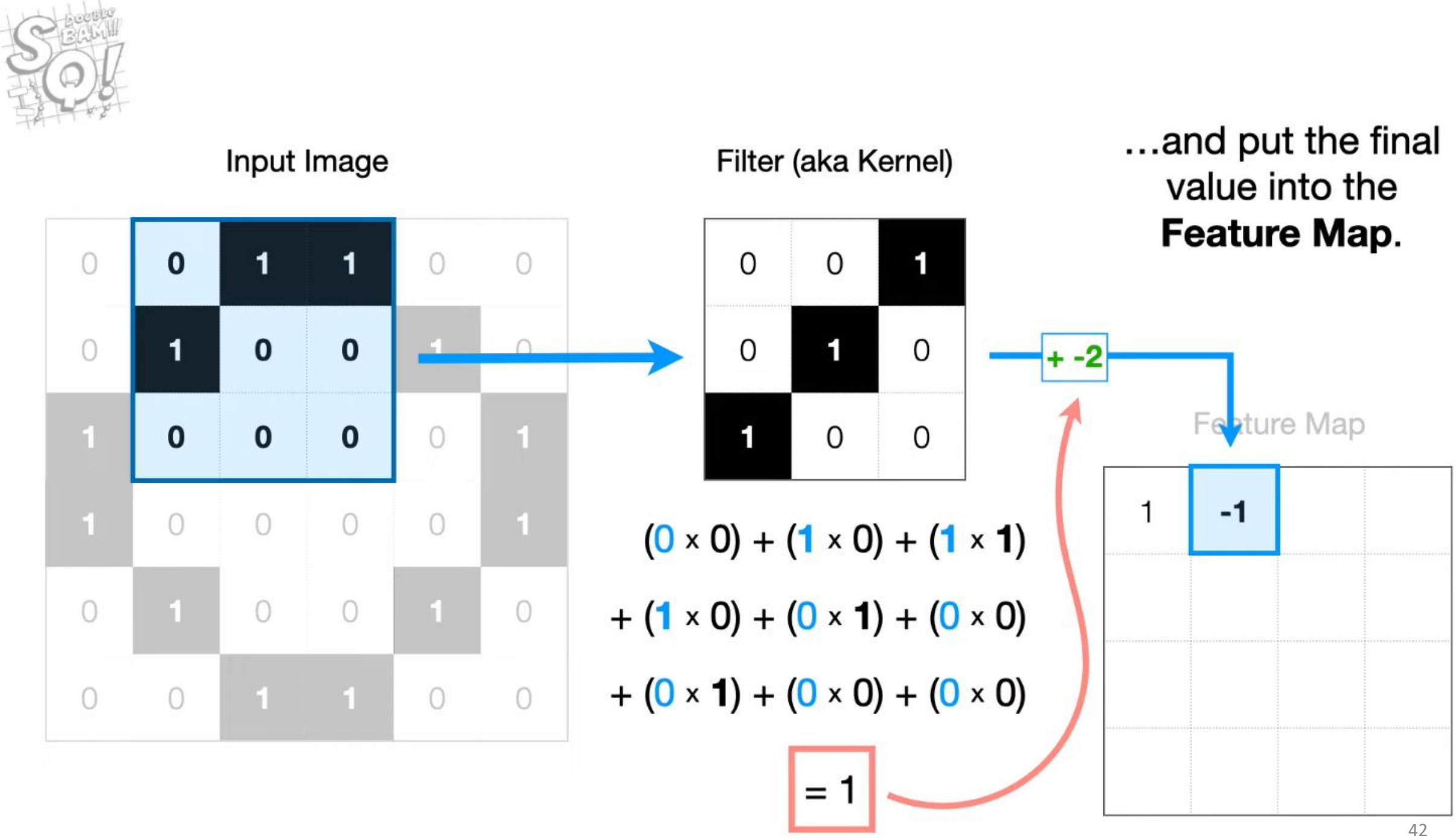

Input Image
Filter (aka Kernel)
Now we add a **Bias** term to the output of the **Filter**…
0 0 1 1 0 0
0 1 0 0 1 0
1 0 0 0 0 1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
0 0 1
0 1 0
1 0 0
+ -2
(0 x 0) + (0 x 0) + (1 x 1)
+ (0 x 0) + (1 x 1) + (0 x 0)
+ (1 x 1) + (0 x 0) + (0 x 0)
= 3


Input Image
Filter (aka Kernel)
…and put the final value into something called a **Feature Map**.
+ -2
Feature Map
(0 × 0) + (0 × 0) + (1 × 1)
+ (0 × 0) + (1 × 1) + (0 × 0)
+ (1 × 1) + (0 × 0) + (0 × 0)
= 3


Input Image
Filter (aka Kernel)
Now, in this example, we slide the **Filter** over one pixel…
+ -2
Feature Map
1


Input Image
Filter (aka Kernel)
…and calculate the **Dot Product** of the **Filter** and overlapping pixels in the image…
+ -2
Feature Map
(0 × 0) + (1 × 0) + (1 × 1)
+ (1 × 0) + (0 × 1) + (0 × 0)
+ (0 × 1) + (0 × 0) + (0 × 0)


Input Image
Filter (aka Kernel)
…add the **Bias** term…
+ -2
Feature Map
(0 × 0) + (1 × 0) + (1 × 1)
+ (1 × 0) + (0 × 1) + (0 × 0)
+ (0 × 1) + (0 × 0) + (0 × 0)
= 1


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

Input Image

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

Filter (aka Kernel)

0	0	1
0	1	0
1	0	0

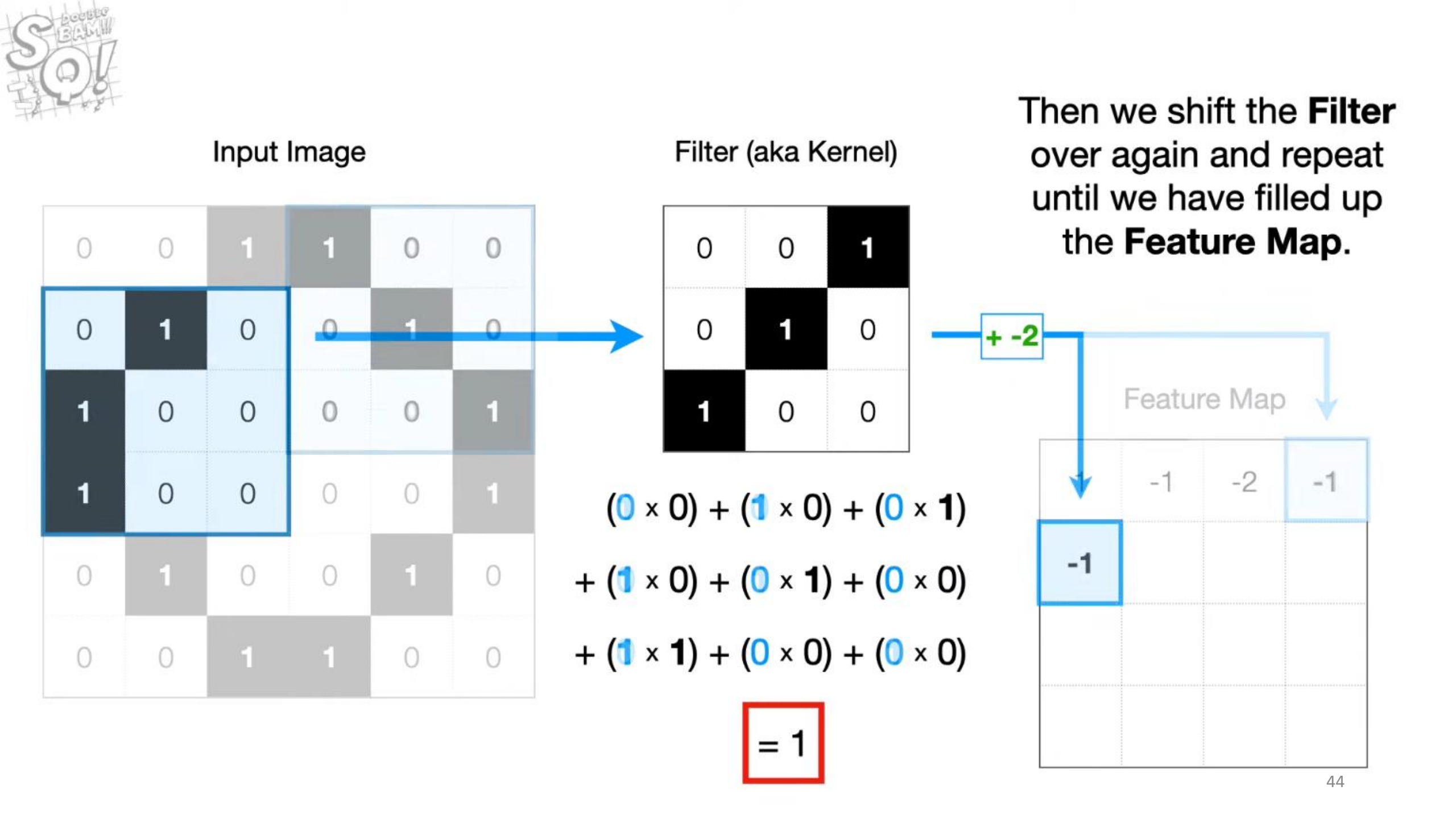
Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map.**

$$(1 \times 0) + (1 \times 0) + (0 \times 1)$$
$$+ (0 \times 0) + (0 \times 1) + (1 \times 0)$$
$$+ (0 \times 1) + (0 \times 0) + (0 \times 0)$$
$$= 0$$

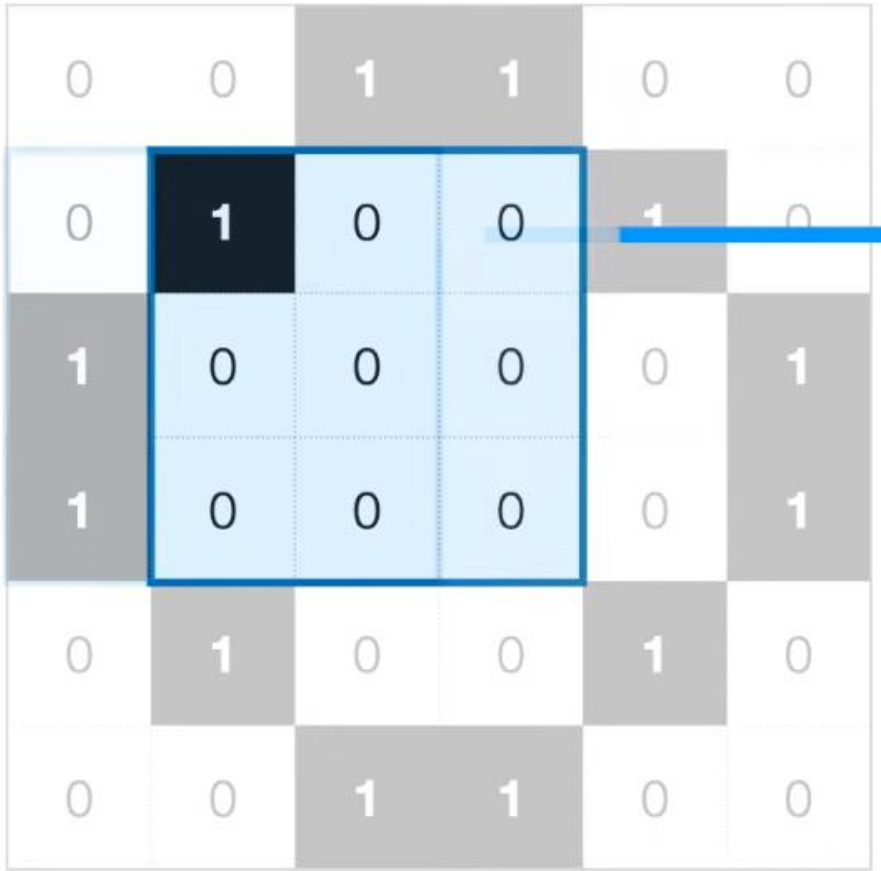
+ -2

Feature Map

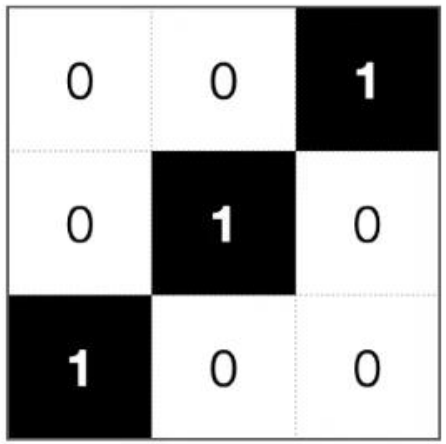
1	-1	


Input Image
Filter (aka Kernel)
Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.
(0 × 0) + (1 × 0) + (0 × 1)
+ (1 × 0) + (0 × 1) + (0 × 0)
+ (1 × 1) + (0 × 0) + (0 × 0)
= 1
+ -2
Feature Map
-1
-2
-1
-1

Input Image



Filter (aka Kernel)



$(1 \times 0) + (0 \times 0) + (0 \times 1)$

$+ (0 \times 0) + (0 \times 1) + (0 \times 0)$

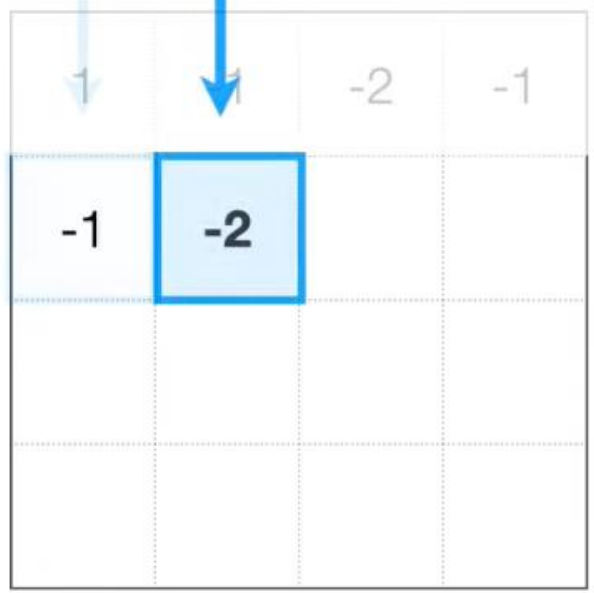
$+ (0 \times 1) + (0 \times 0) + (0 \times 0)$

$= 0$

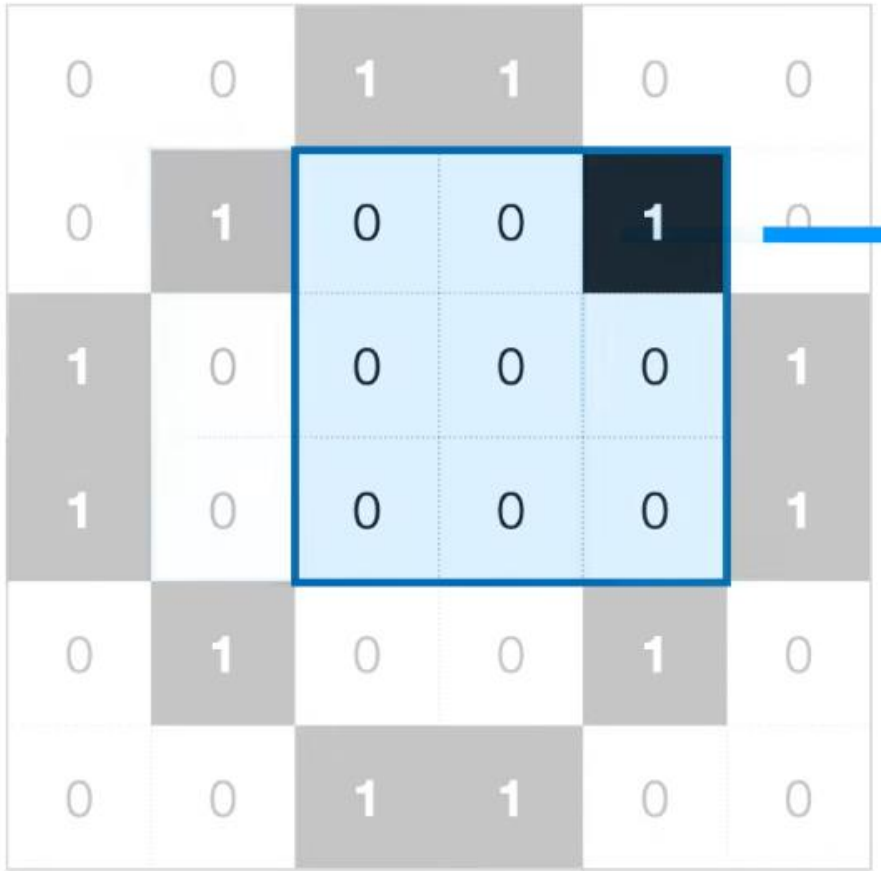
Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.

+ -2

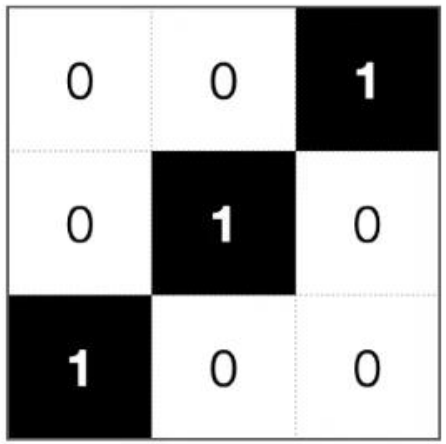
Feature Map



Input Image



Filter (aka Kernel)



$(0 \times 0) + (0 \times 0) + (1 \times 1)$

$+ (0 \times 0) + (0 \times 1) + (0 \times 0)$

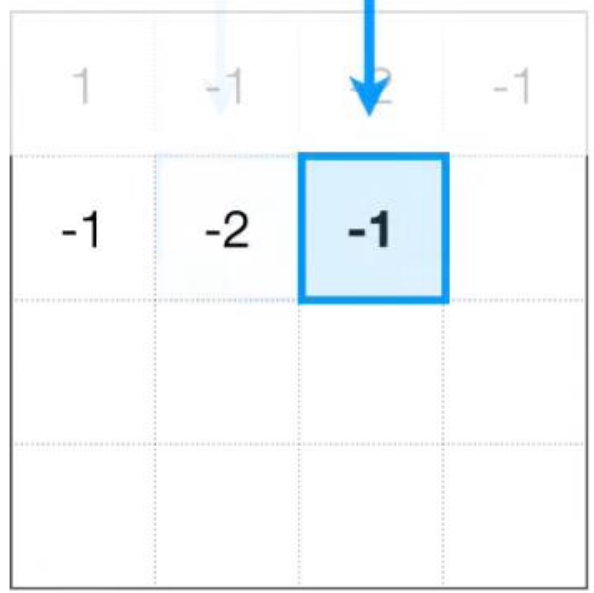
$+ (0 \times 1) + (0 \times 0) + (0 \times 0)$

$= 1$

Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map.**

+ -2

Feature Map



Input Image

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

Filter (aka Kernel)

0	0	1
0	1	0
1	0	0

$(0 \times 0) + (1 \times 0) + (0 \times 1)$

$+ (0 \times 0) + (0 \times 1) + (1 \times 0)$

$+ (0 \times 1) + (0 \times 0) + (1 \times 0)$

$= 0$

Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.

+ -2

Feature Map

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

Input Image

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

Filter (aka Kernel)

0	0	1
0	1	0
1	0	0

$(1 \times 0) + (0 \times 0) + (0 \times 1)$

$+ (1 \times 0) + (0 \times 1) + (0 \times 0)$

$+ (0 \times 1) + (1 \times 0) + (0 \times 0)$

$= 0$

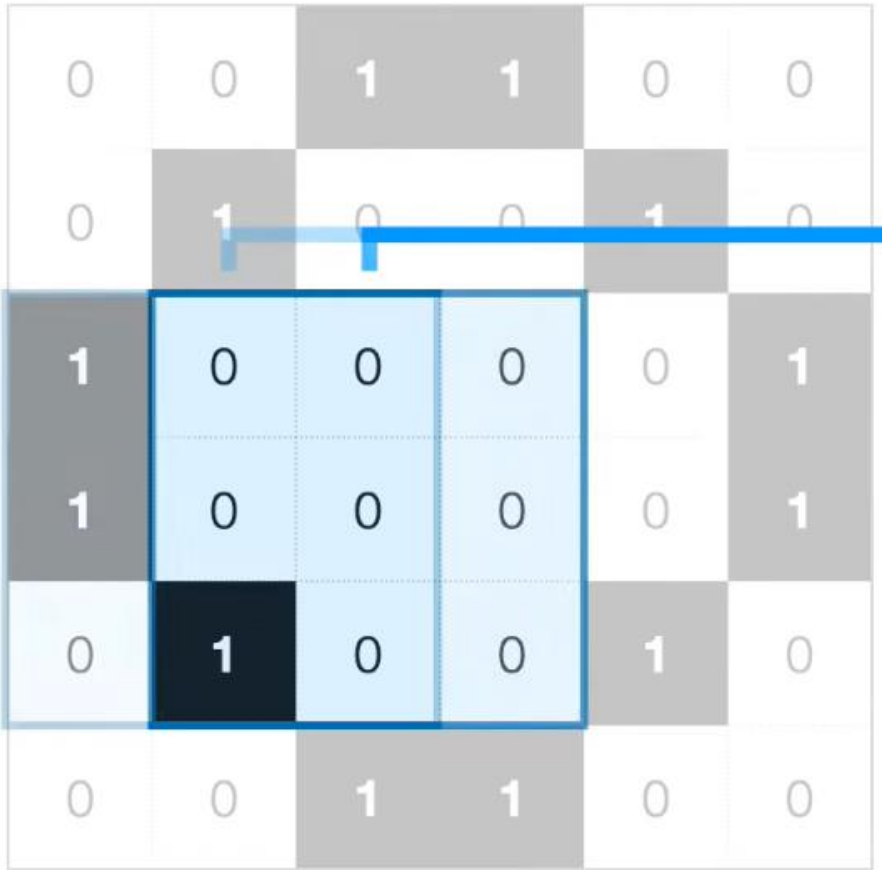
+ -2

Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.

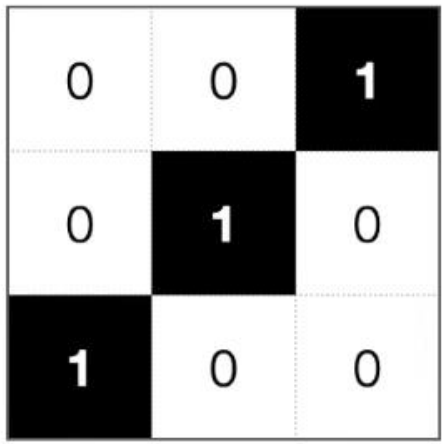
Feature Map

-2	-1	-2	-1
-1	-2	-1	-2

Input Image



Filter (aka Kernel)



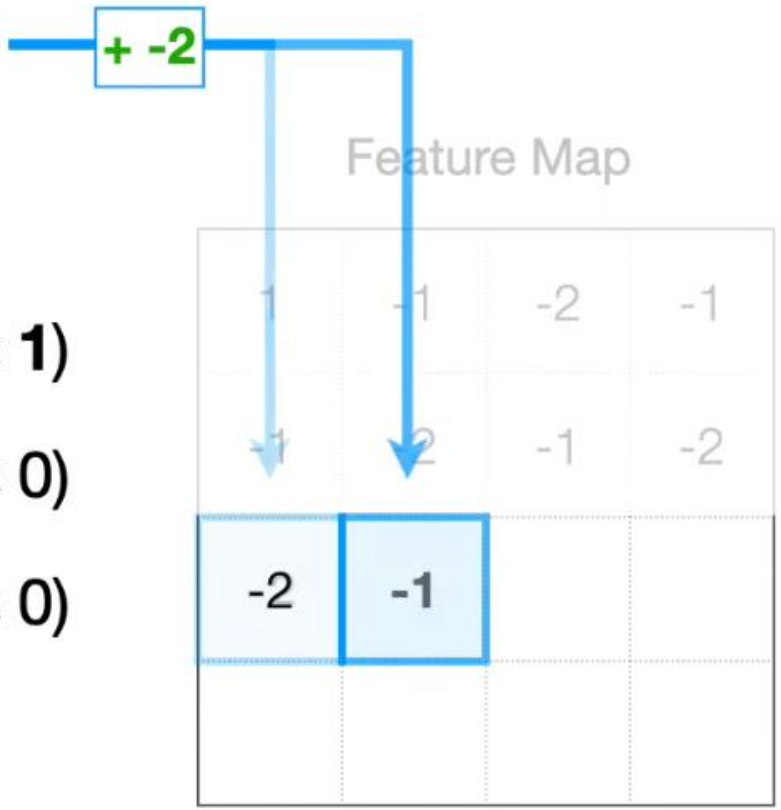
$(0 \times 0) + (0 \times 0) + (0 \times 1)$

$+ (0 \times 0) + (0 \times 1) + (0 \times 0)$

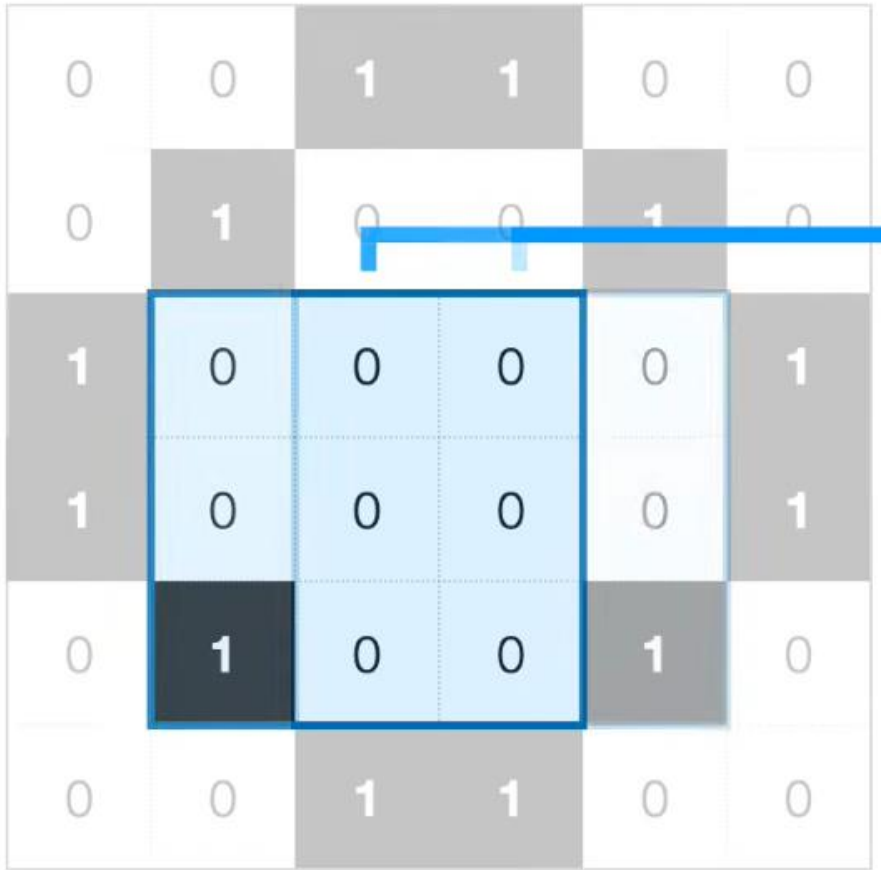
$+ (1 \times 1) + (0 \times 0) + (0 \times 0)$

$= 1$

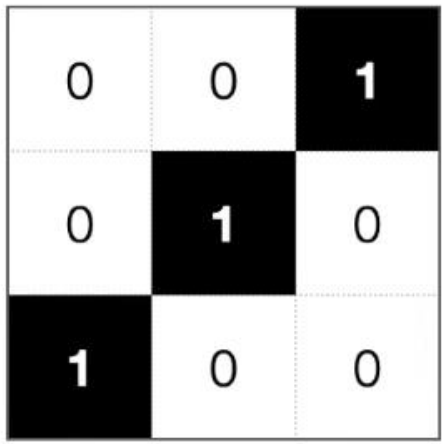
Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.



Input Image



Filter (aka Kernel)



$(0 \times 0) + (0 \times 0) + (0 \times 1)$

$+ (0 \times 0) + (0 \times 1) + (0 \times 0)$

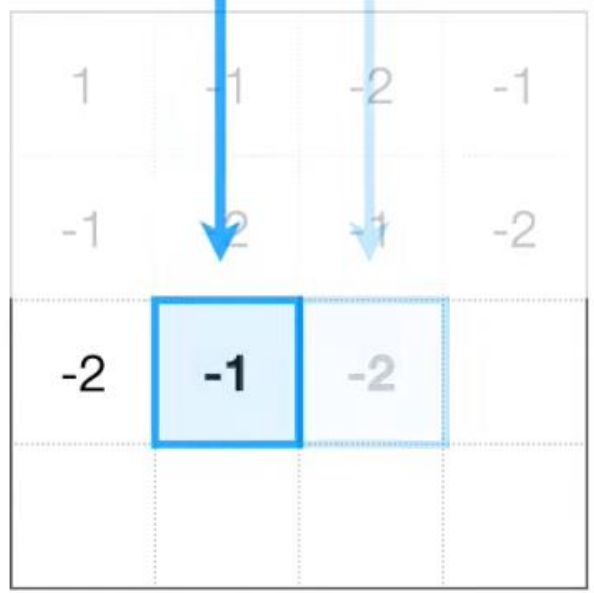
$+ (1 \times 1) + (0 \times 0) + (0 \times 0)$

$= 1$

Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.

+ -2

Feature Map



Input Image

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

Filter (aka Kernel)

0	0	1
0	1	0
1	0	0

$$(0 \times 0) + (0 \times 0) + (1 \times 1)$$
$$+ (0 \times 0) + (0 \times 1) + (1 \times 0)$$
$$+ (0 \times 1) + (1 \times 0) + (0 \times 0)$$
$$= 1$$

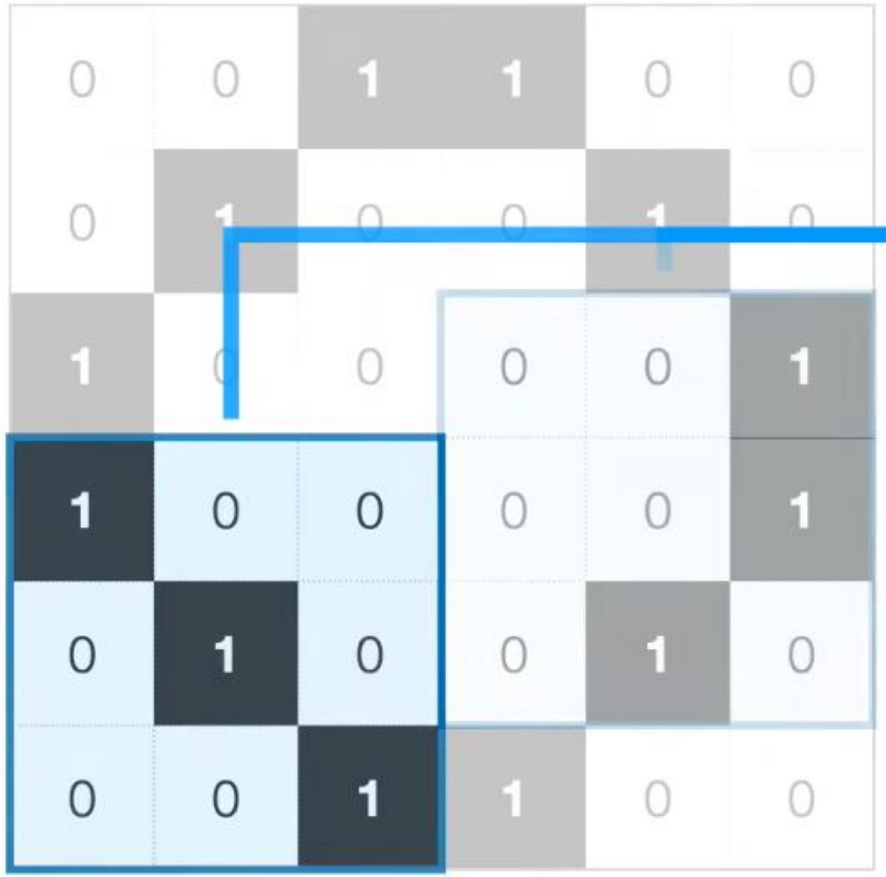
Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.

+ -2

Feature Map

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1

Input Image



Filter (aka Kernel)

0	0	1
0	1	0
1	0	0

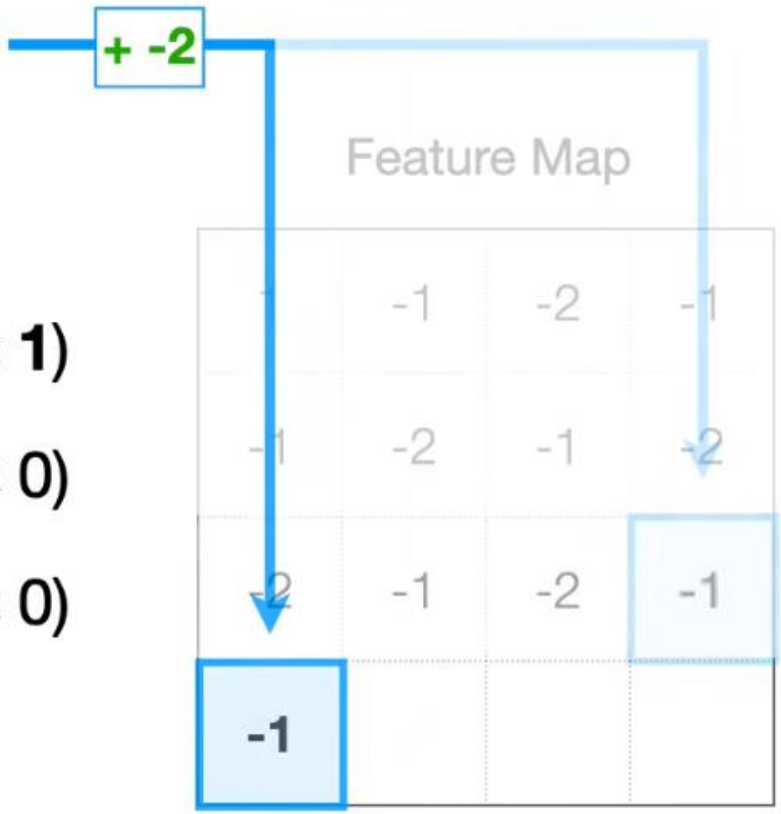
$(1 \times 0) + (0 \times 0) + (0 \times 1)$

$+ (0 \times 0) + (1 \times 1) + (0 \times 0)$

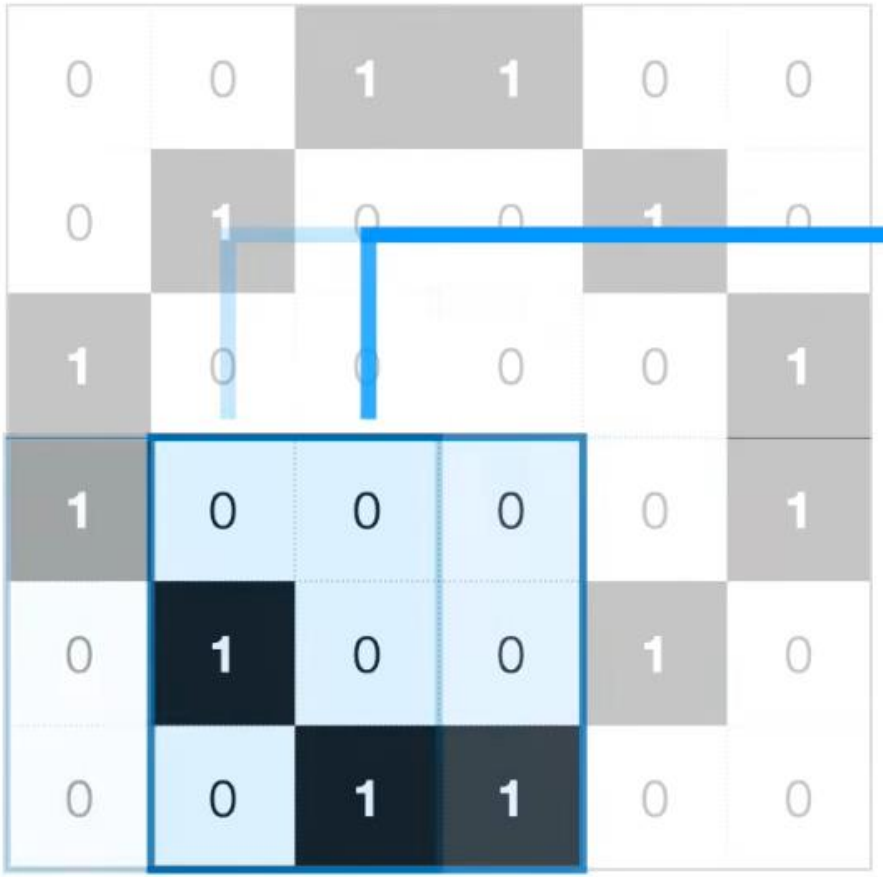
$+ (0 \times 1) + (0 \times 0) + (1 \times 0)$

$= 1$

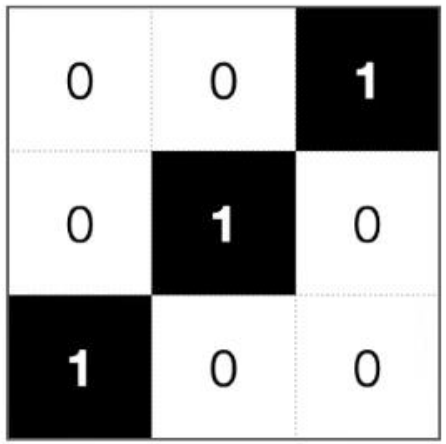
Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map.**



Input Image



Filter (aka Kernel)



$(0 \times 0) + (0 \times 0) + (0 \times 1)$

$+ (1 \times 0) + (0 \times 1) + (0 \times 0)$

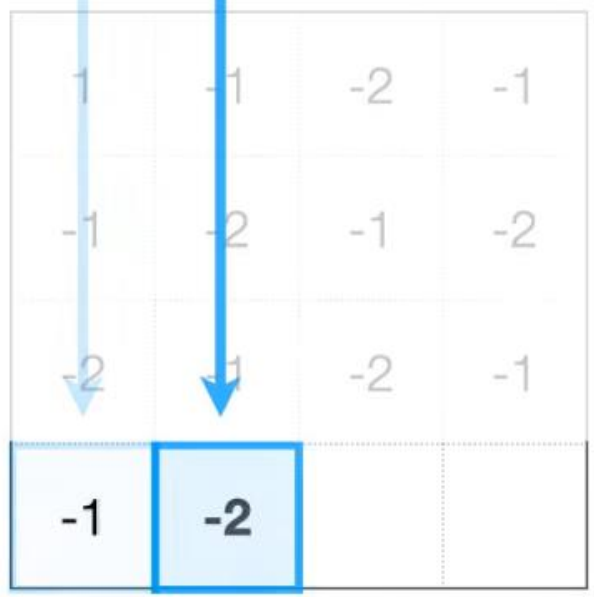
$+ (0 \times 1) + (1 \times 0) + (1 \times 0)$

$= 0$

Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.

+ -2

Feature Map



Input Image

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

Filter (aka Kernel)

0	0	1
0	1	0
1	0	0

$(0 \times 0) + (0 \times 0) + (0 \times 1)$

$+ (0 \times 0) + (0 \times 1) + (1 \times 0)$

$+ (1 \times 1) + (1 \times 0) + (0 \times 0)$

$= 1$

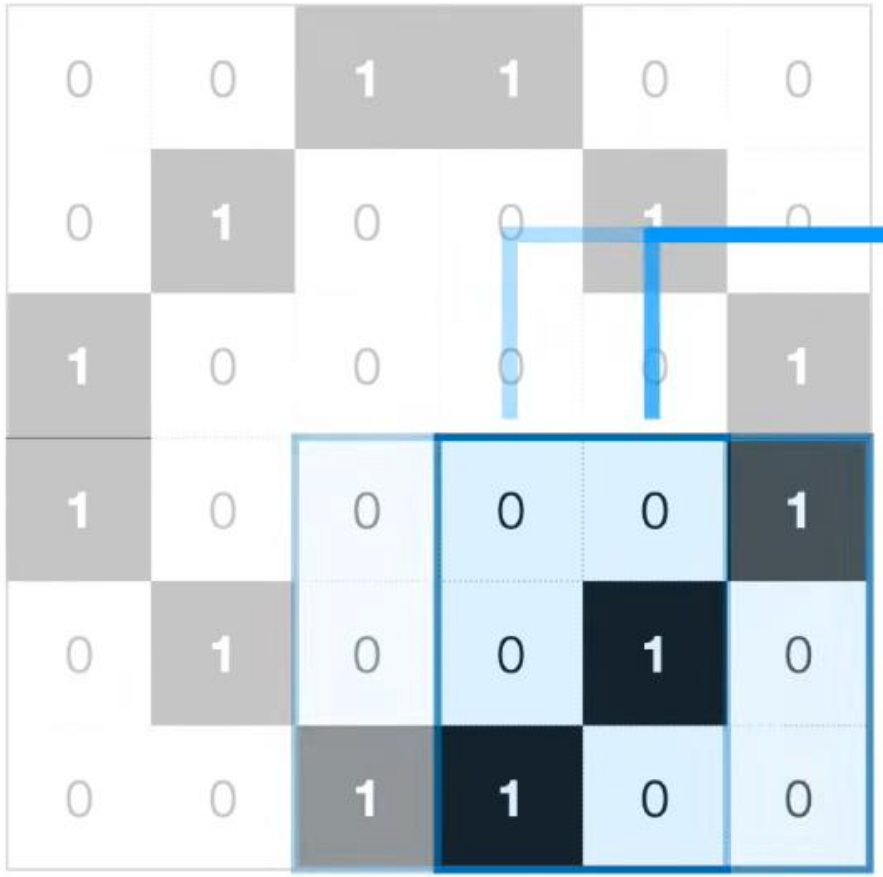
Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map**.

+ -2

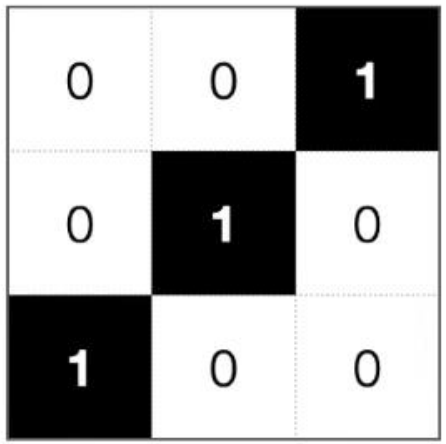
Feature Map

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	

Input Image



Filter (aka Kernel)



$(0 \times 0) + (0 \times 0) + (1 \times 1)$

$+ (0 \times 0) + (1 \times 1) + (0 \times 0)$

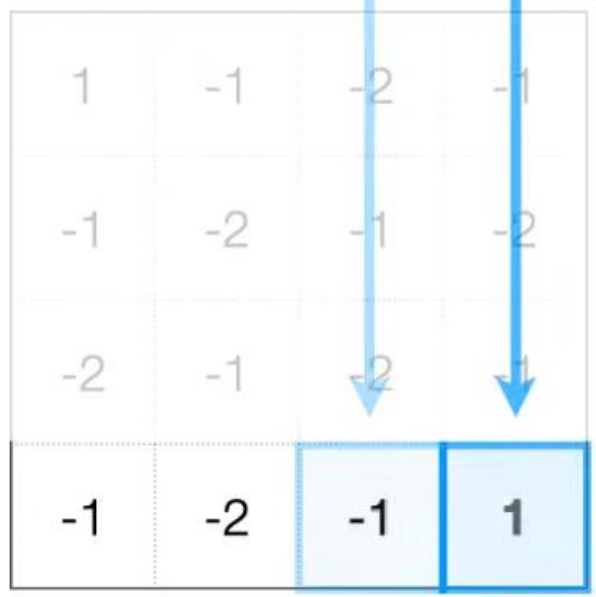
$+ (1 \times 1) + (0 \times 0) + (0 \times 0)$

$= 3$

Then we shift the **Filter** over again and repeat until we have filled up the **Feature Map.**

+ -2

Feature Map



Input Image

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

Filter (aka Kernel)

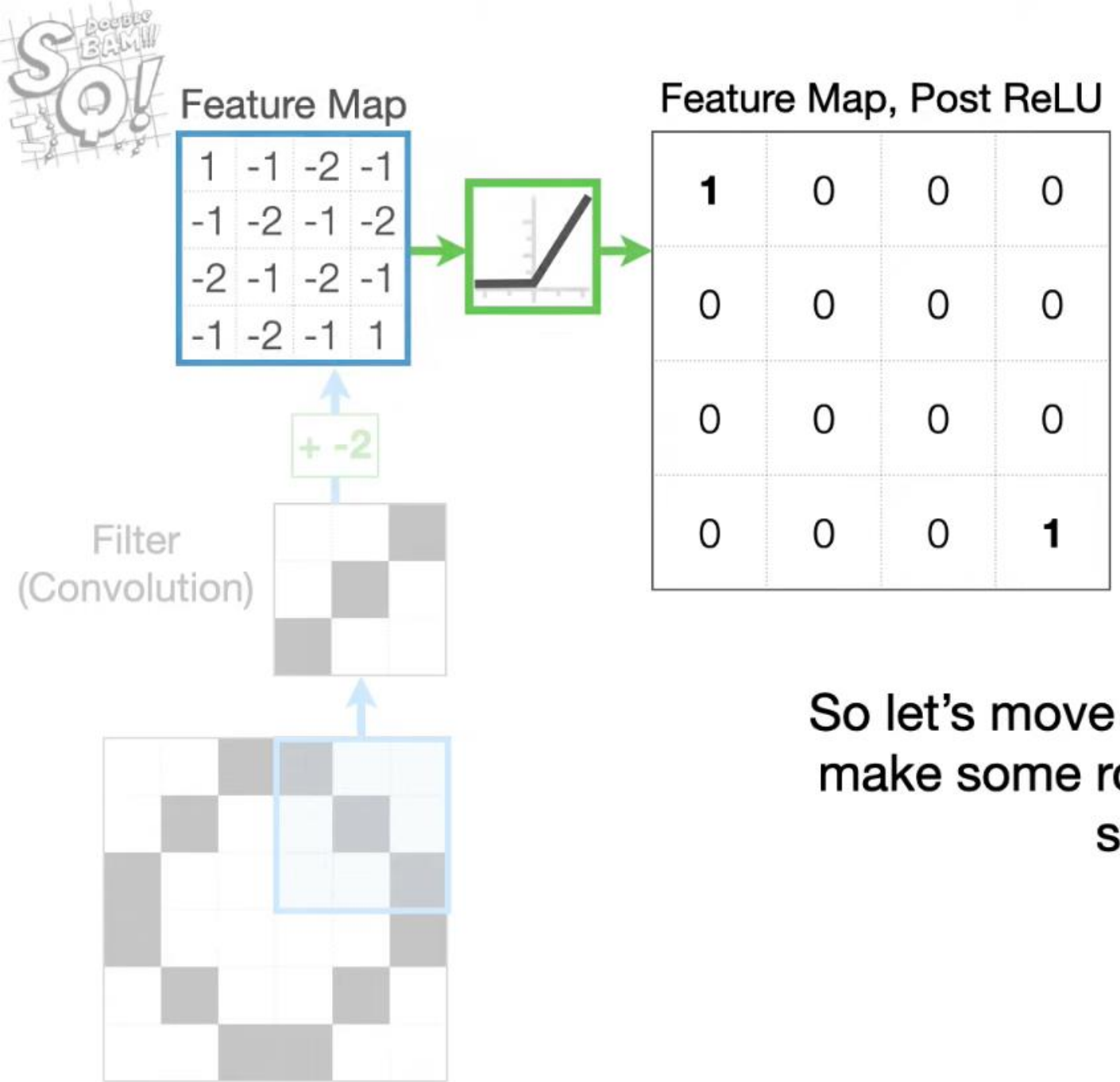
0	0	1
0	1	0
1	0	0

+ -2

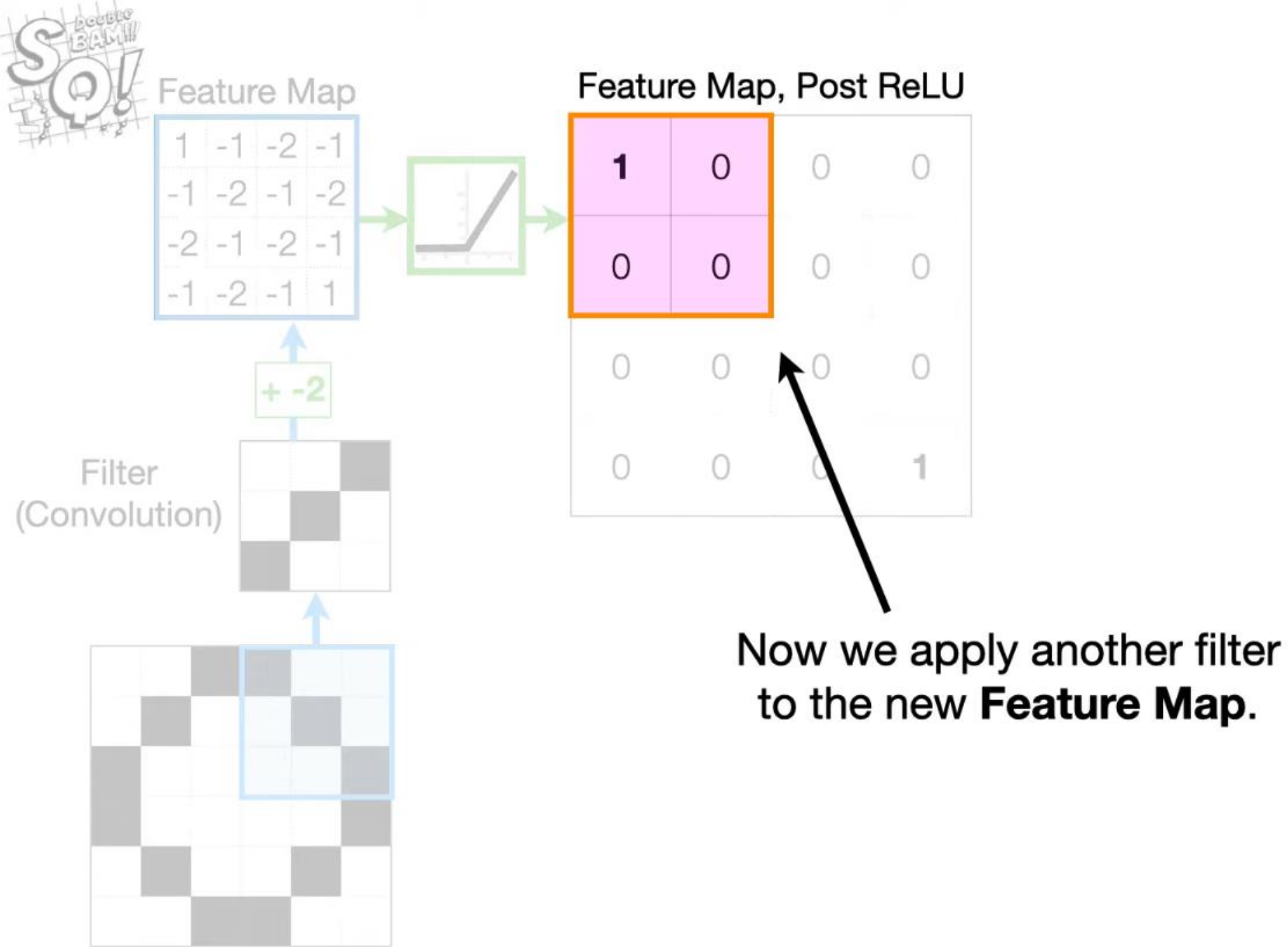
BAM!!! We've filled in the whole **Feature Map.**

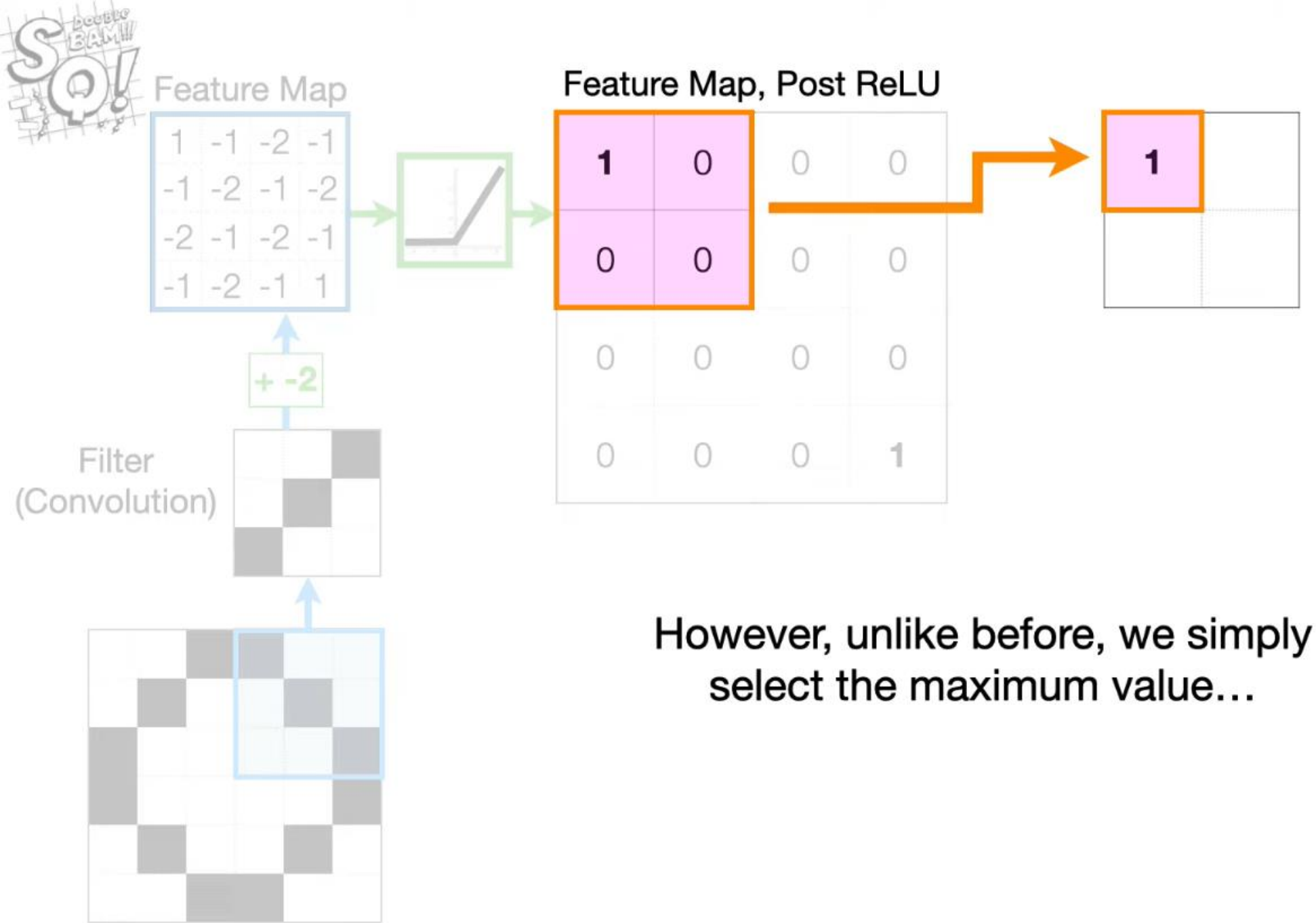
Feature Map

1	-1	-2	-1
-1	-2	-1	-2
-2	-1	-2	-1
-1	-2	-1	1

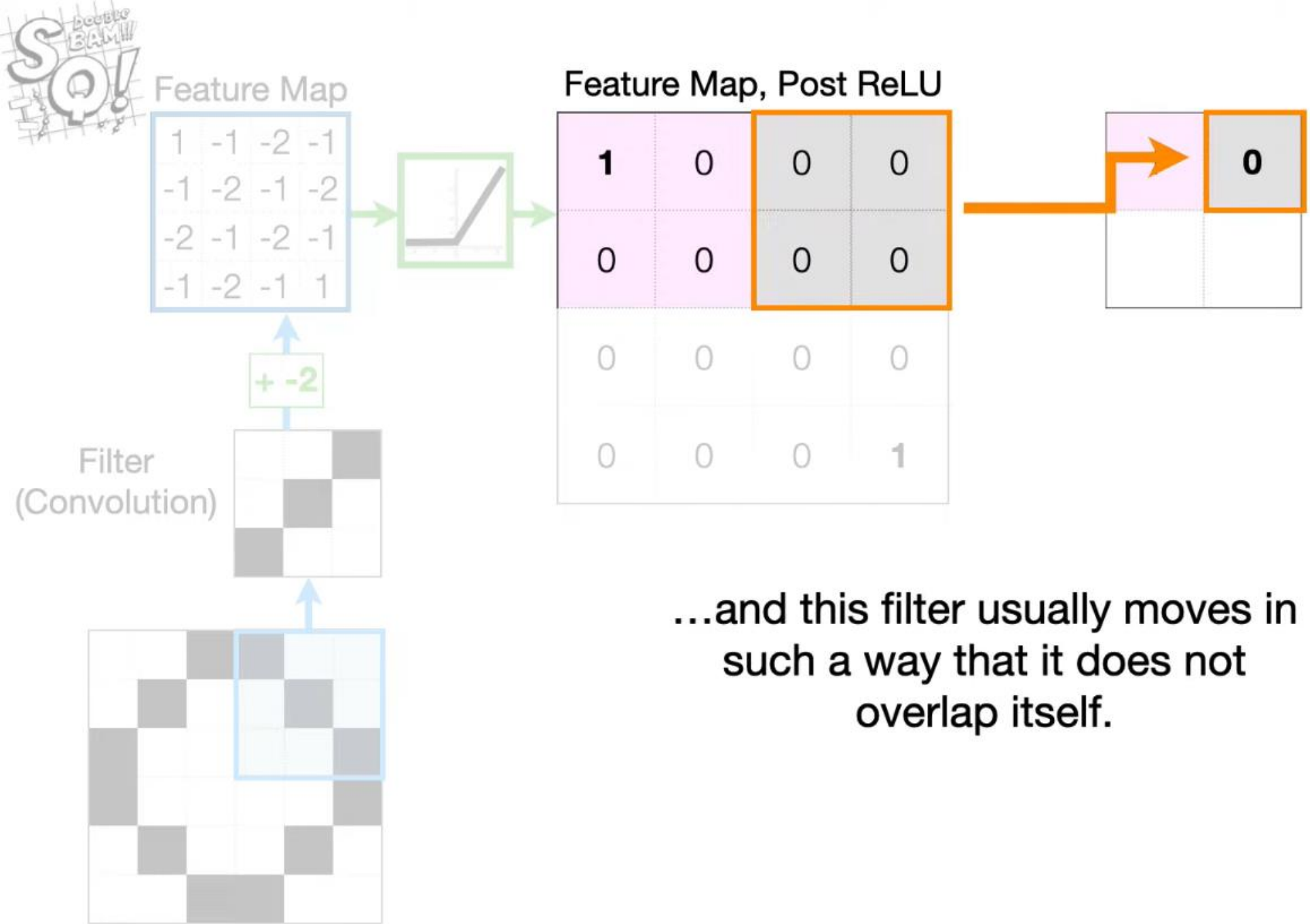


So let's move this stuff over to make some room for the next step.

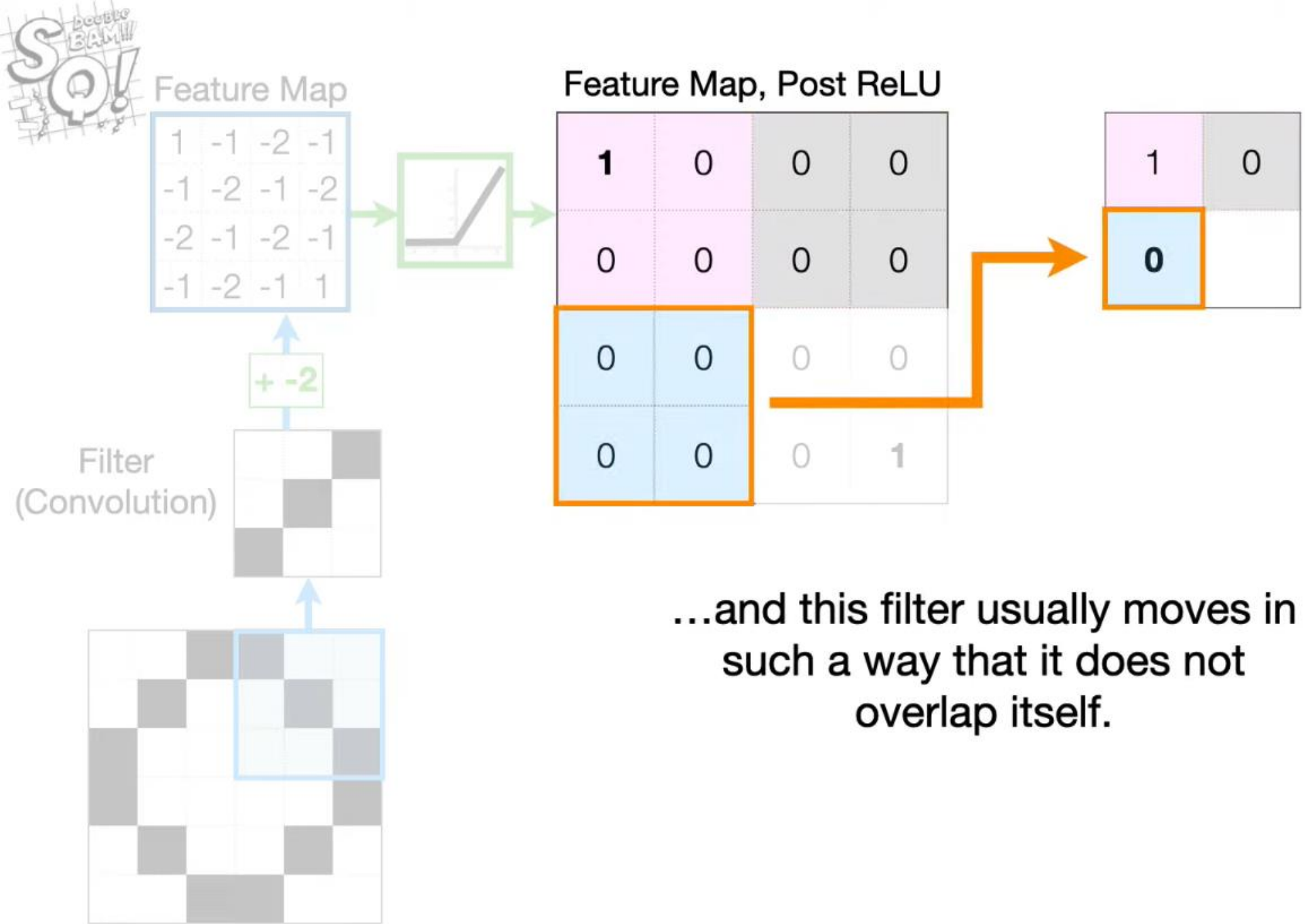

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
Feature Map, Post ReLU
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
Now we apply another filter
to the new **Feature Map**.



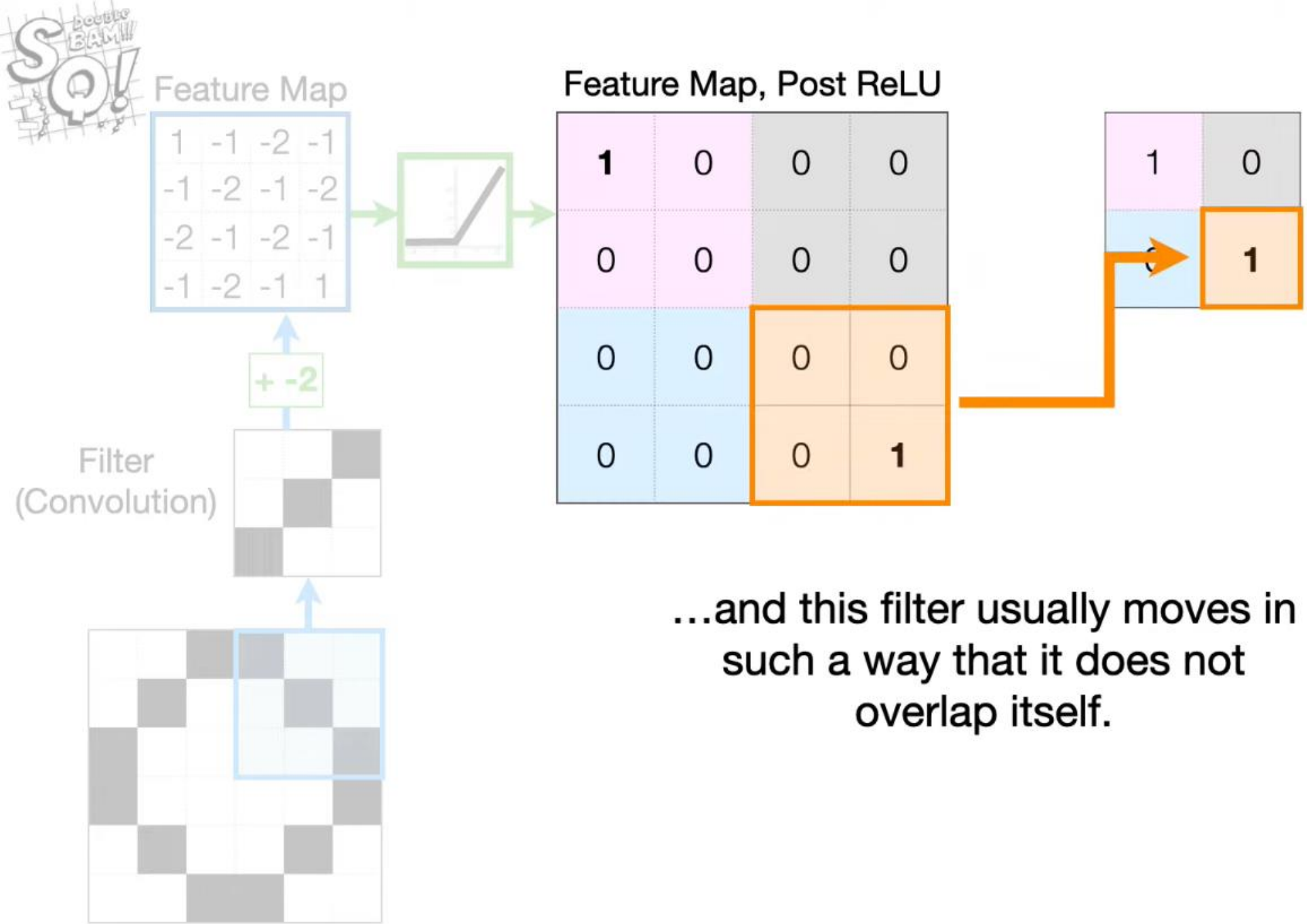
However, unlike before, we simply select the maximum value…



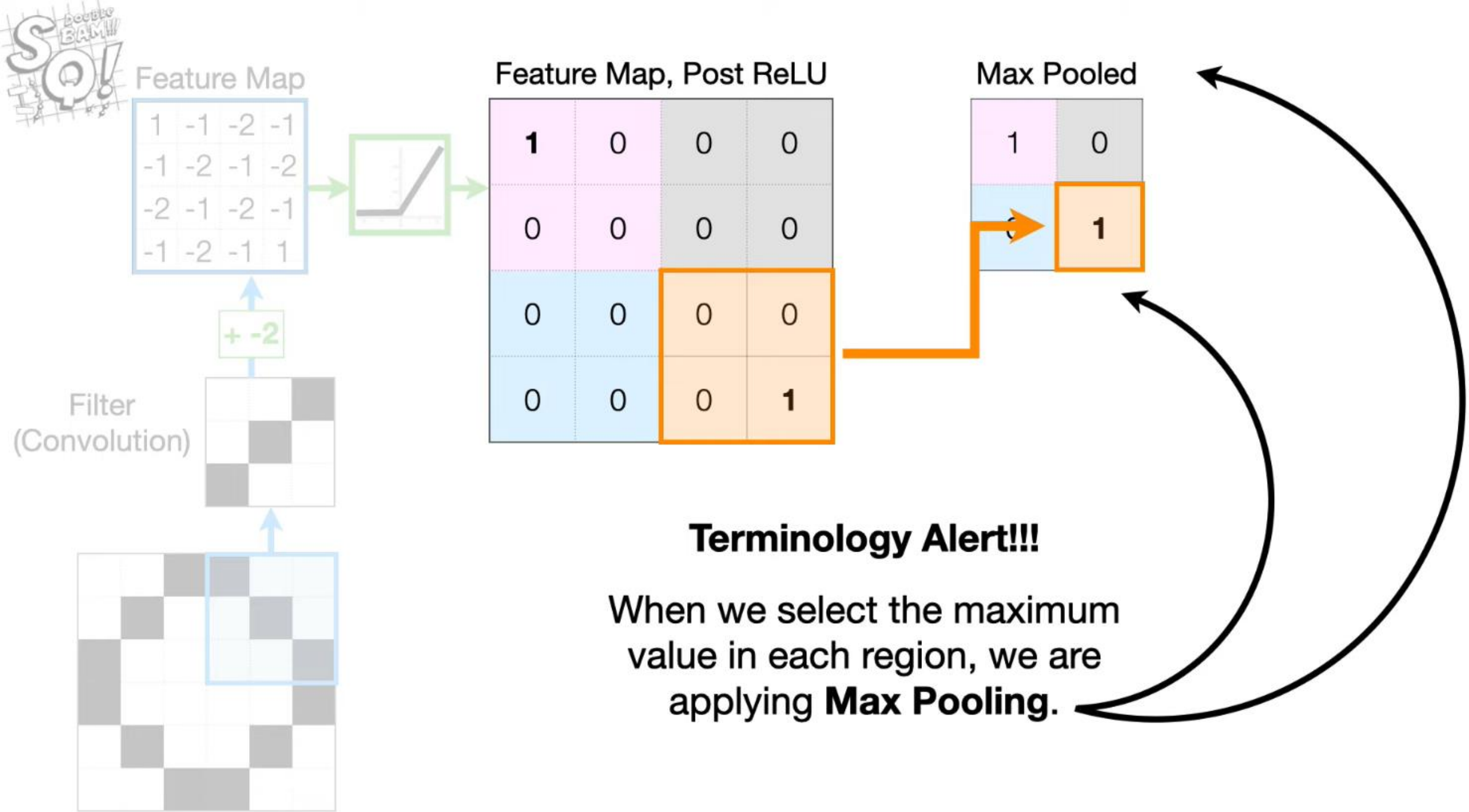
…and this filter usually moves in such a way that it does not overlap itself.

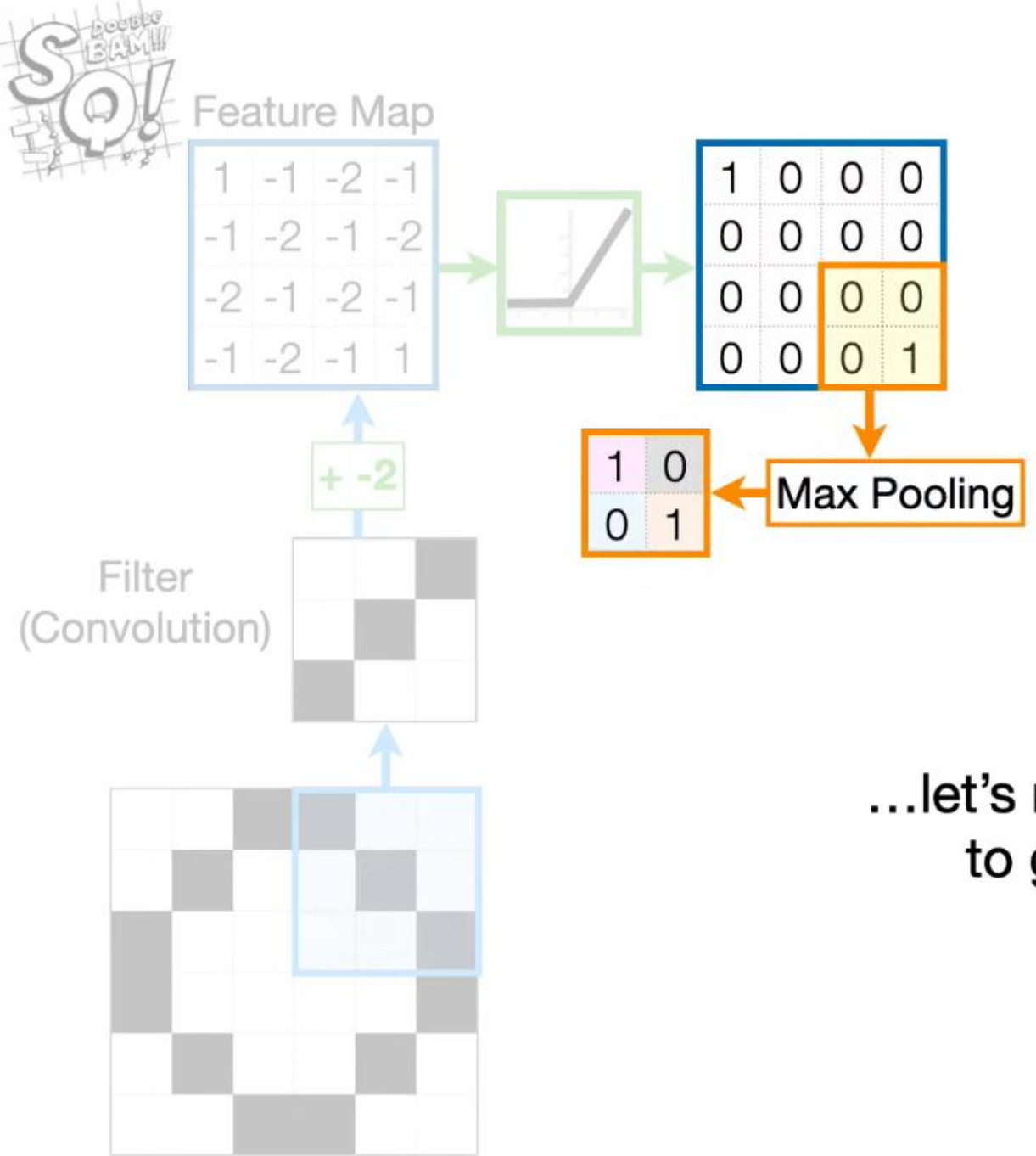


…and this filter usually moves in such a way that it does not overlap itself.

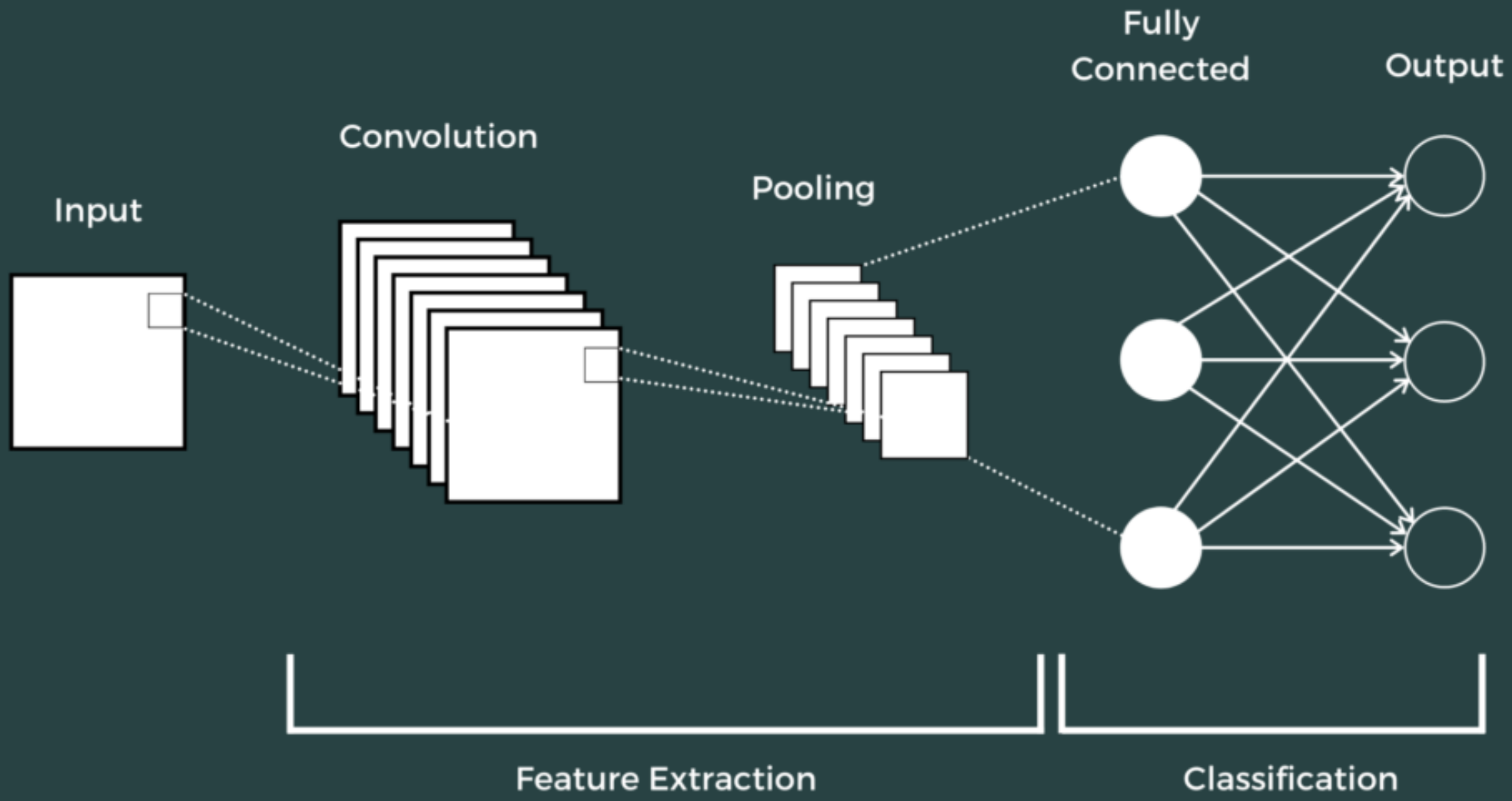


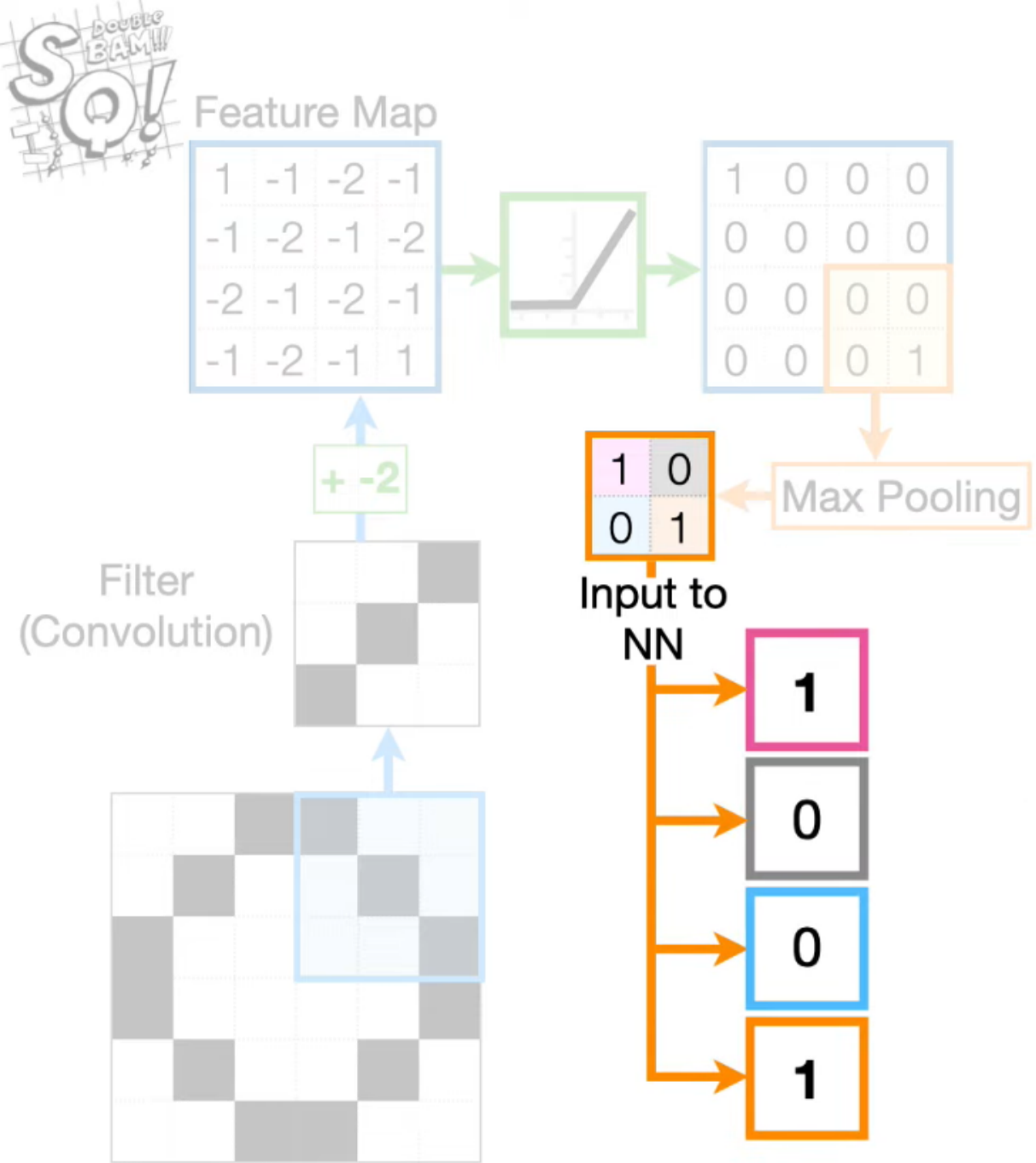
…and this filter usually moves in such a way that it does not overlap itself.


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
1
Terminology Alert!!!
When we select the maximum value in each region, we are applying Max Pooling.

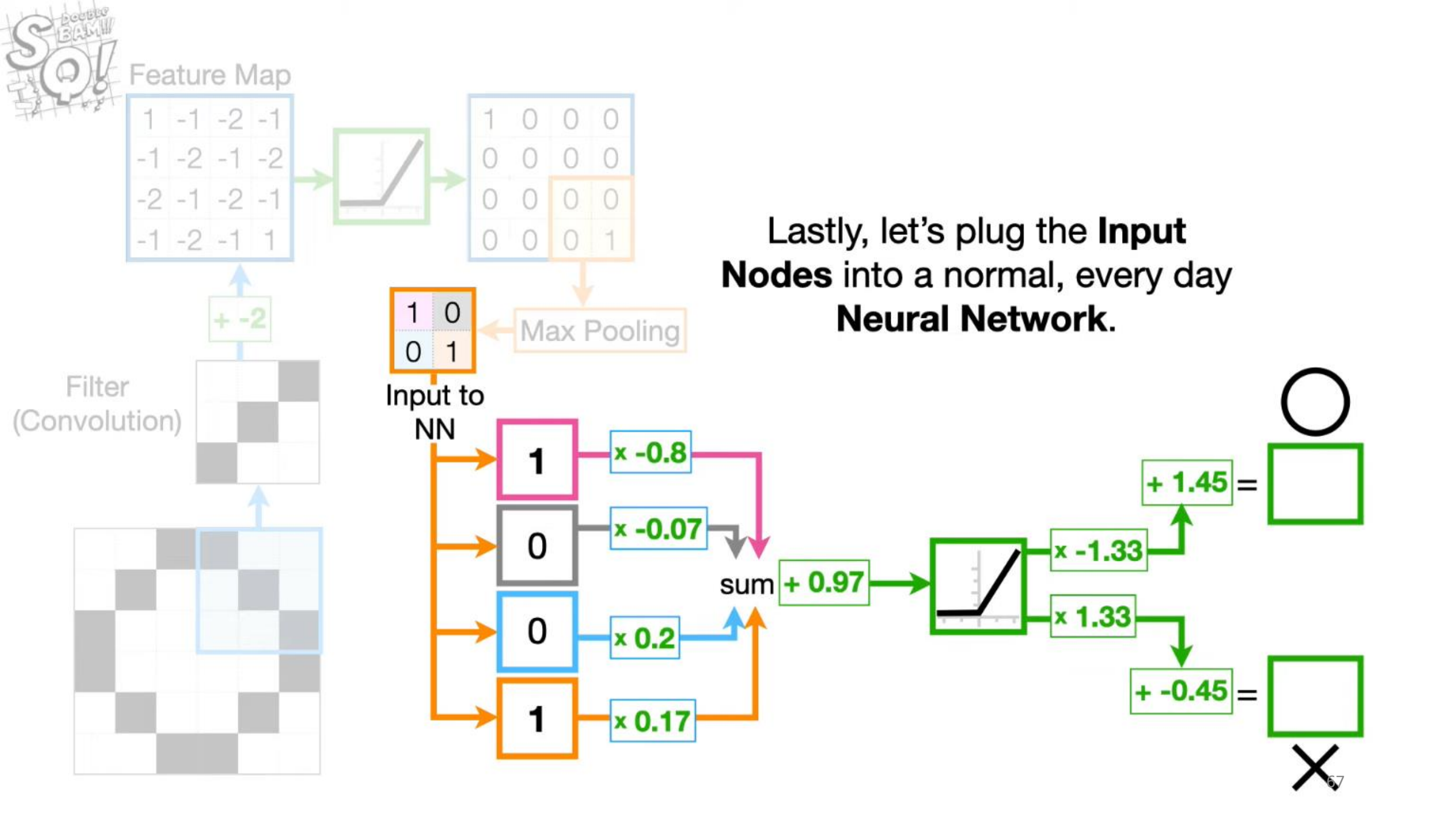


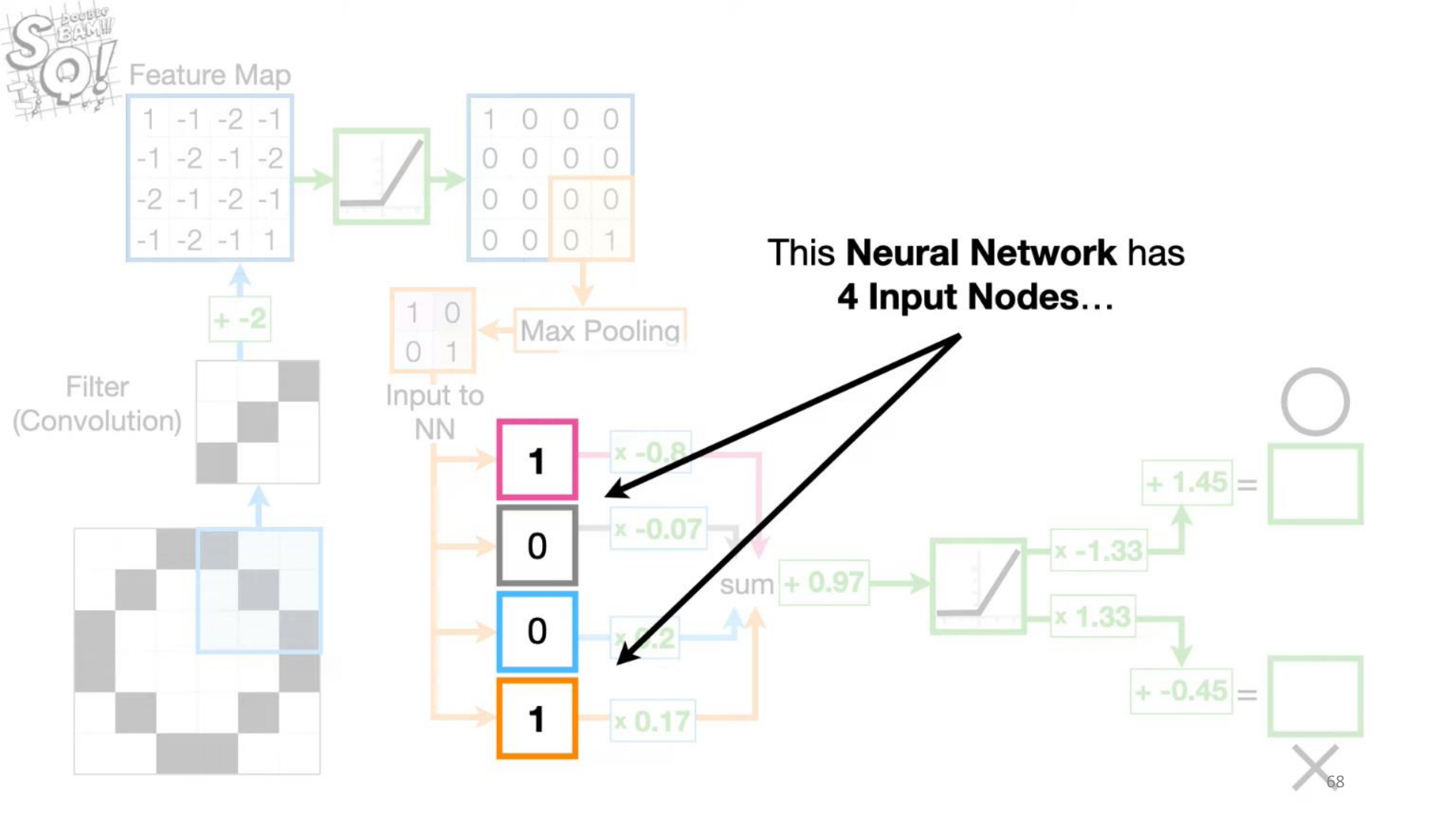
…let's move and shrink things to give us more room.

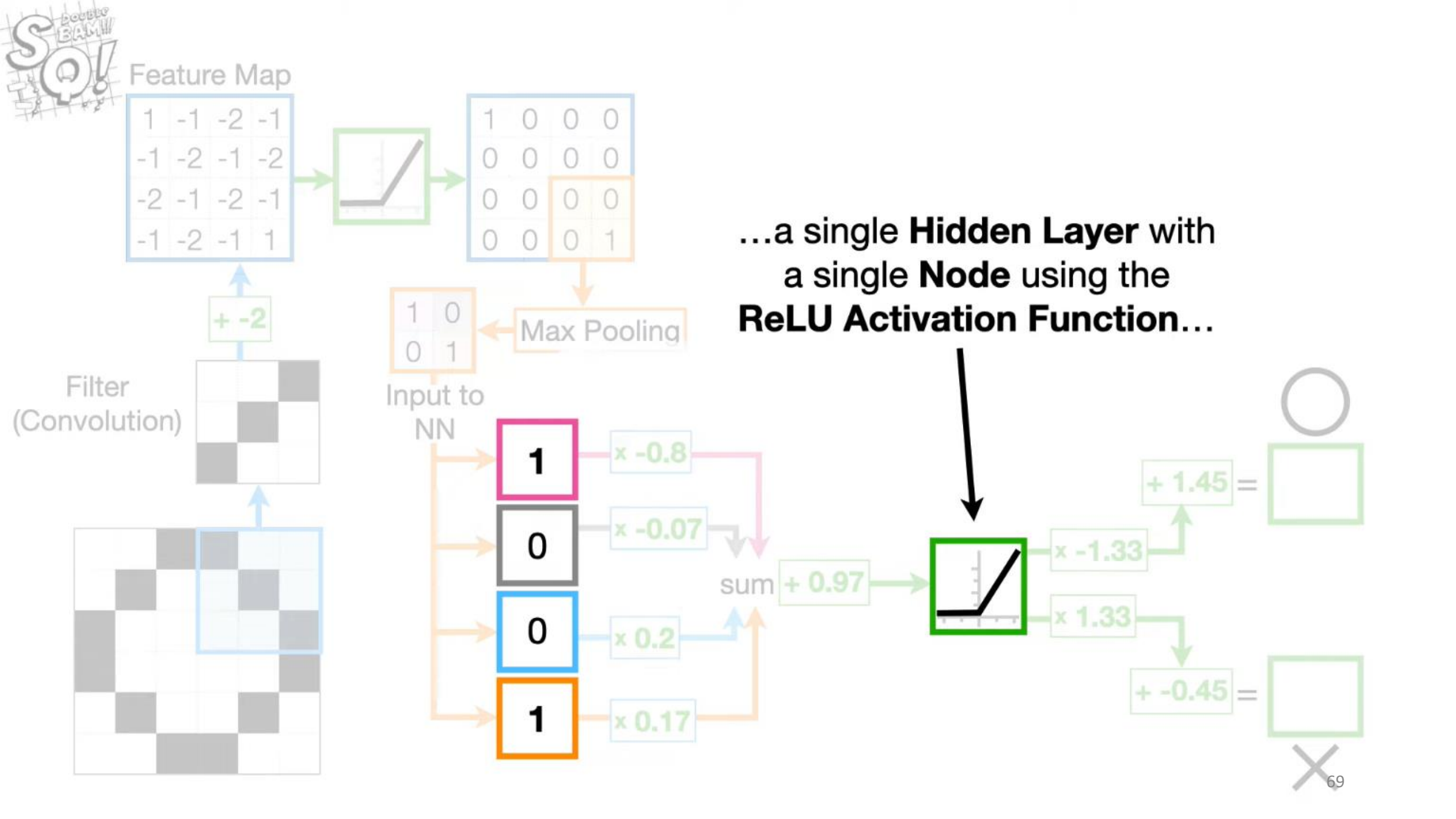

Input
Convolution
Pooling
Fully
Connected
Output
Feature Extraction
Classification

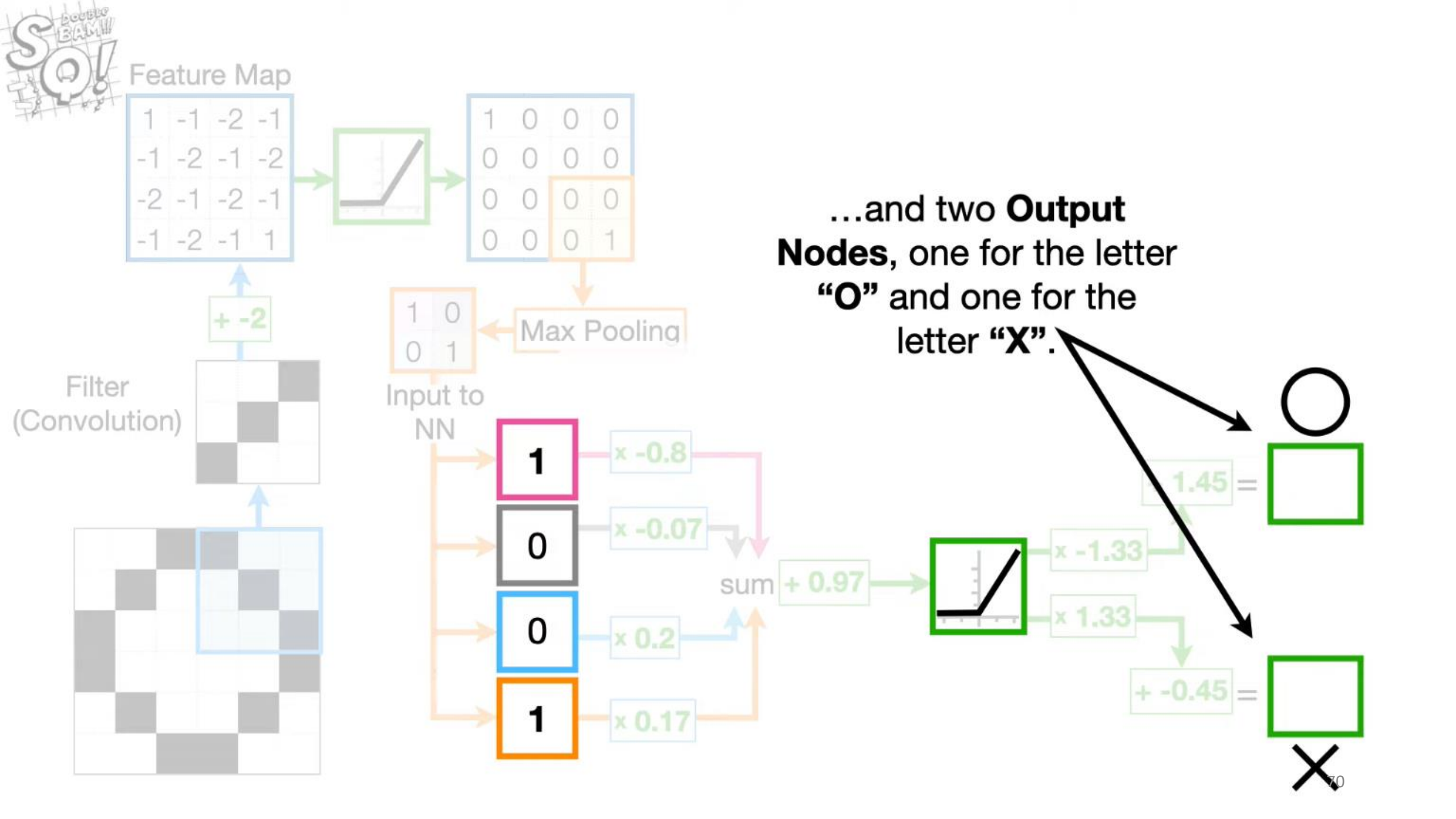


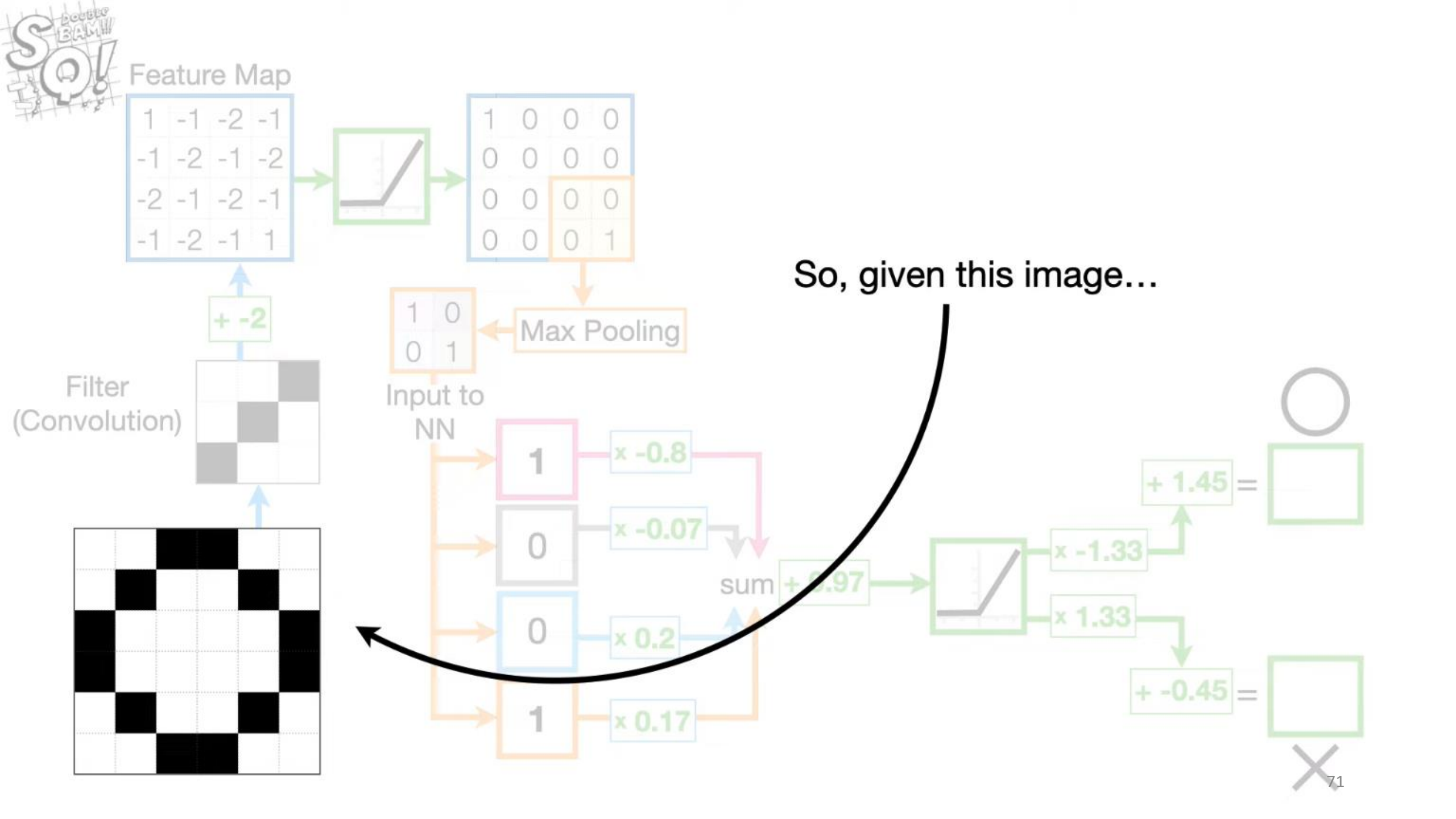
Now let's convert the **Pooled Layer** into a column of **Input Nodes**.

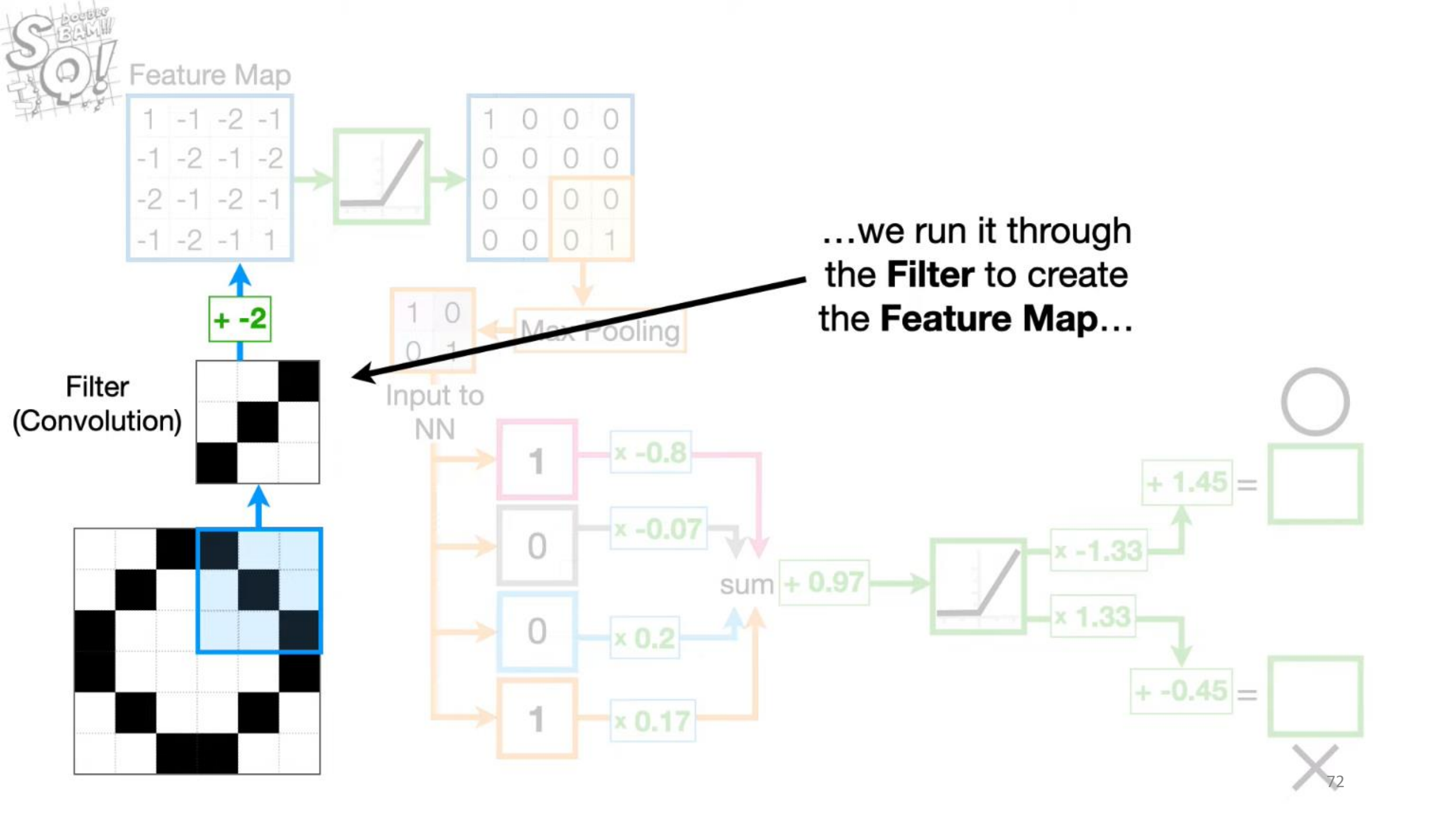

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
Max Pooling
1 0
0 1
+ -2
Filter
(Convolution)
Input to
NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =
Lastly, let's plug the **Input Nodes** into a normal, every day **Neural Network.**

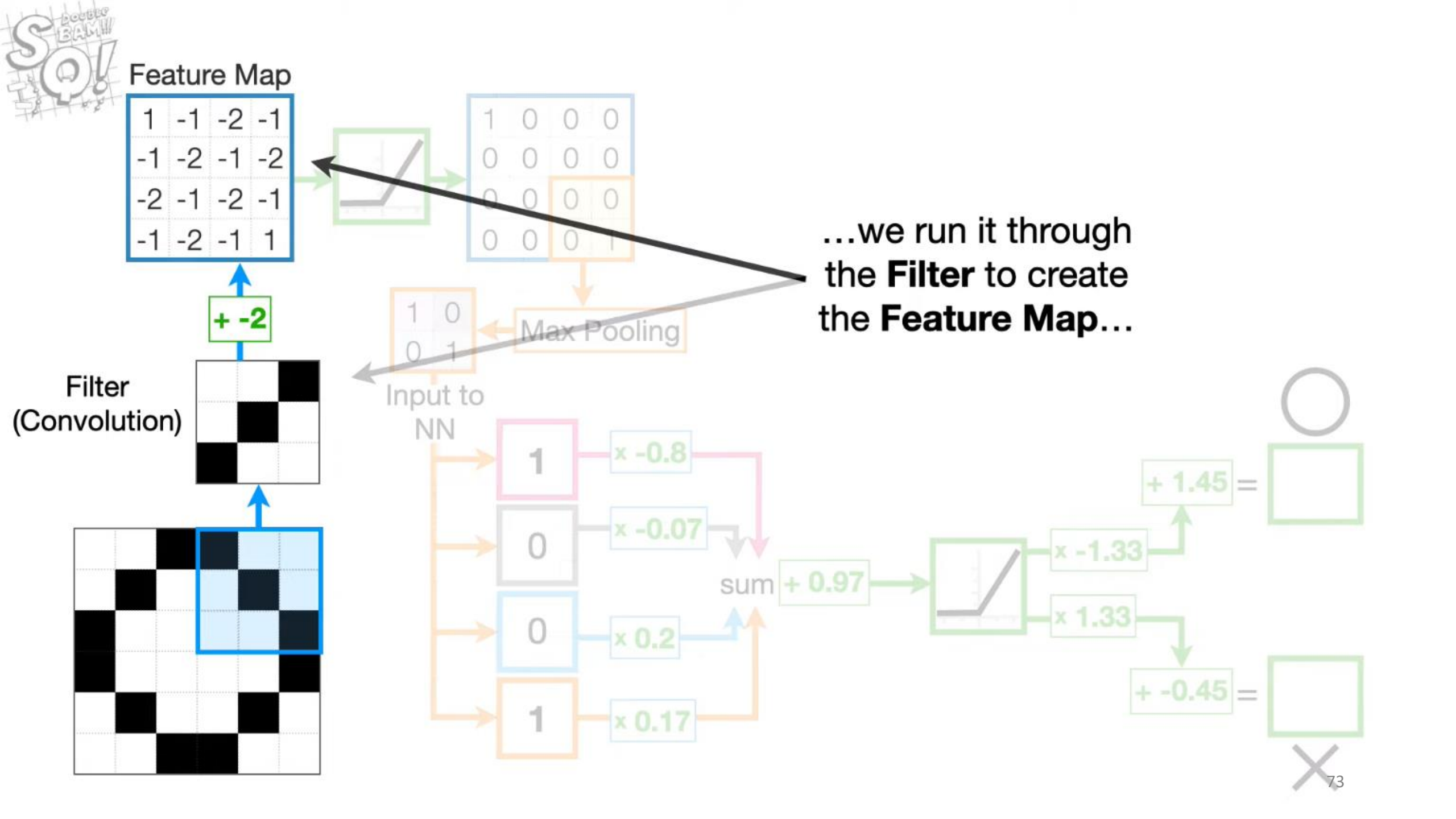

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
+ -2
Filter
(Convolution)
1 0
0 1
Max Pooling
Input to
NN
This Neural Network has
4 Input Nodes…
1
0
0
1
x -0.8
x -0.07
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =

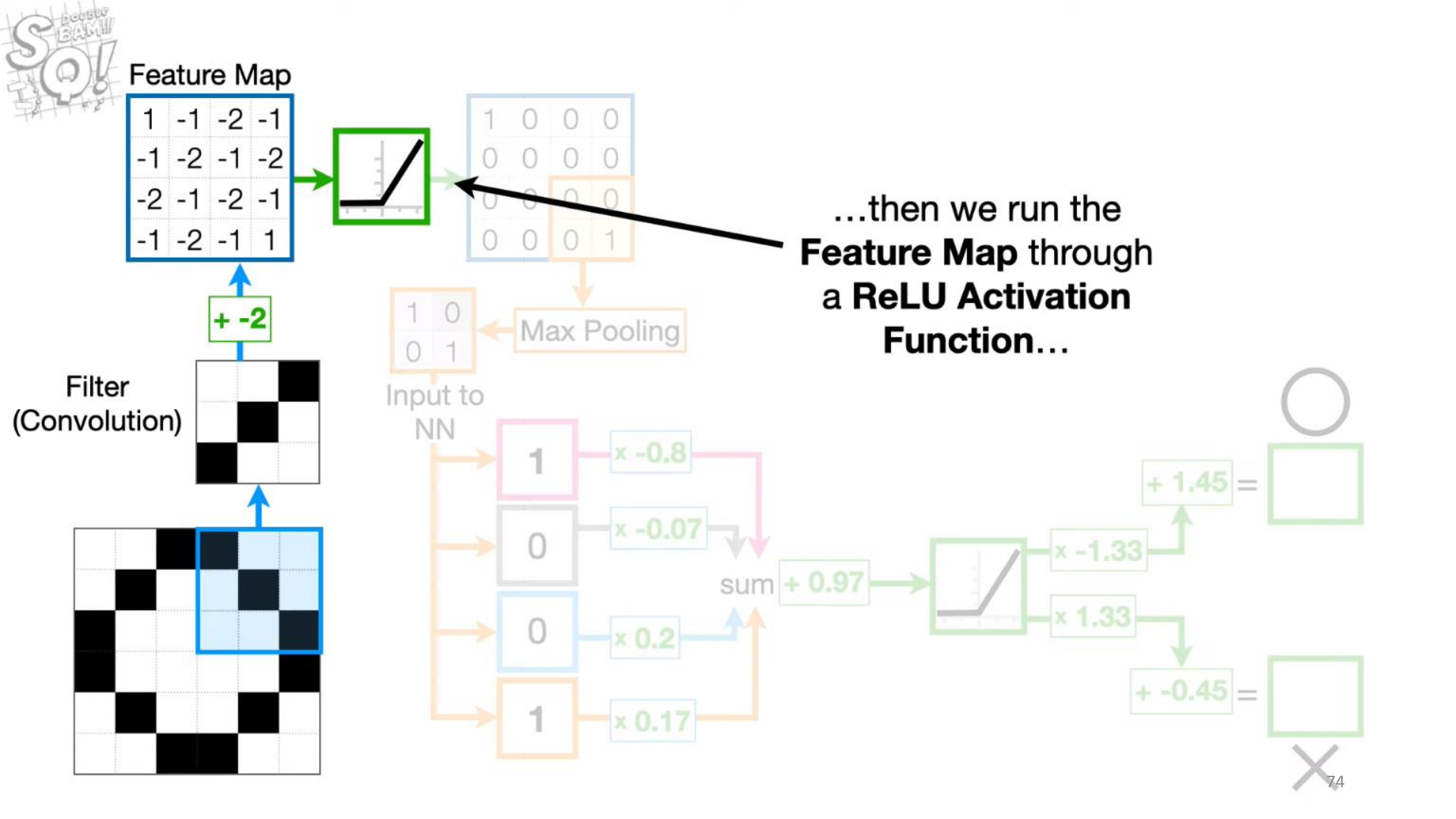

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
Max Pooling
Input to
NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
…a single Hidden Layer with
a single Node using the
ReLU Activation Function…
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =

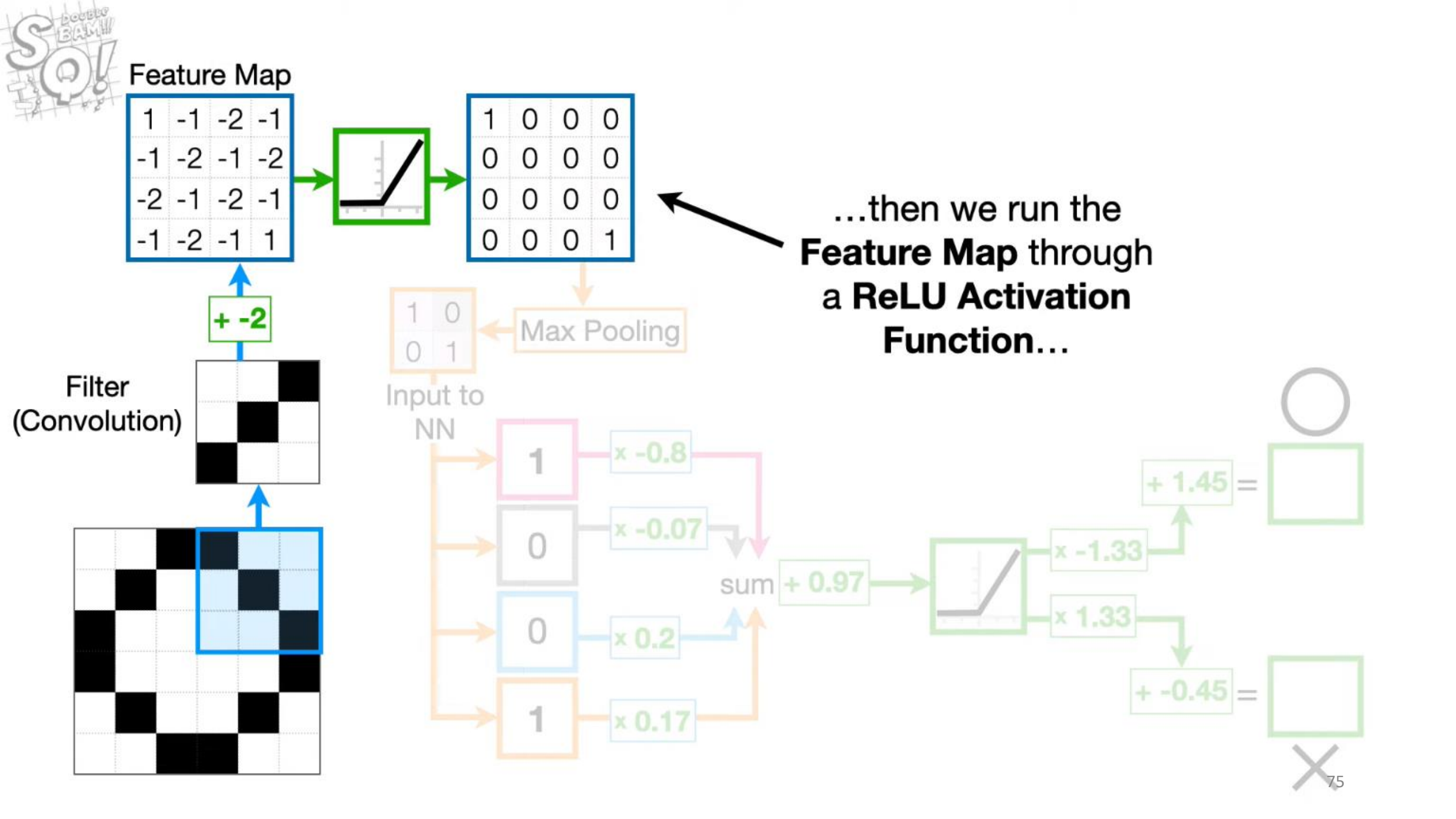

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
Max Pooling
1 0
0 1
Input to NN
+ -2
Filter (Convolution)
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
…and two Output Nodes, one for the letter “O” and one for the letter “X”.
x -1.33
x 1.33
1.45 =
+ -0.45 =

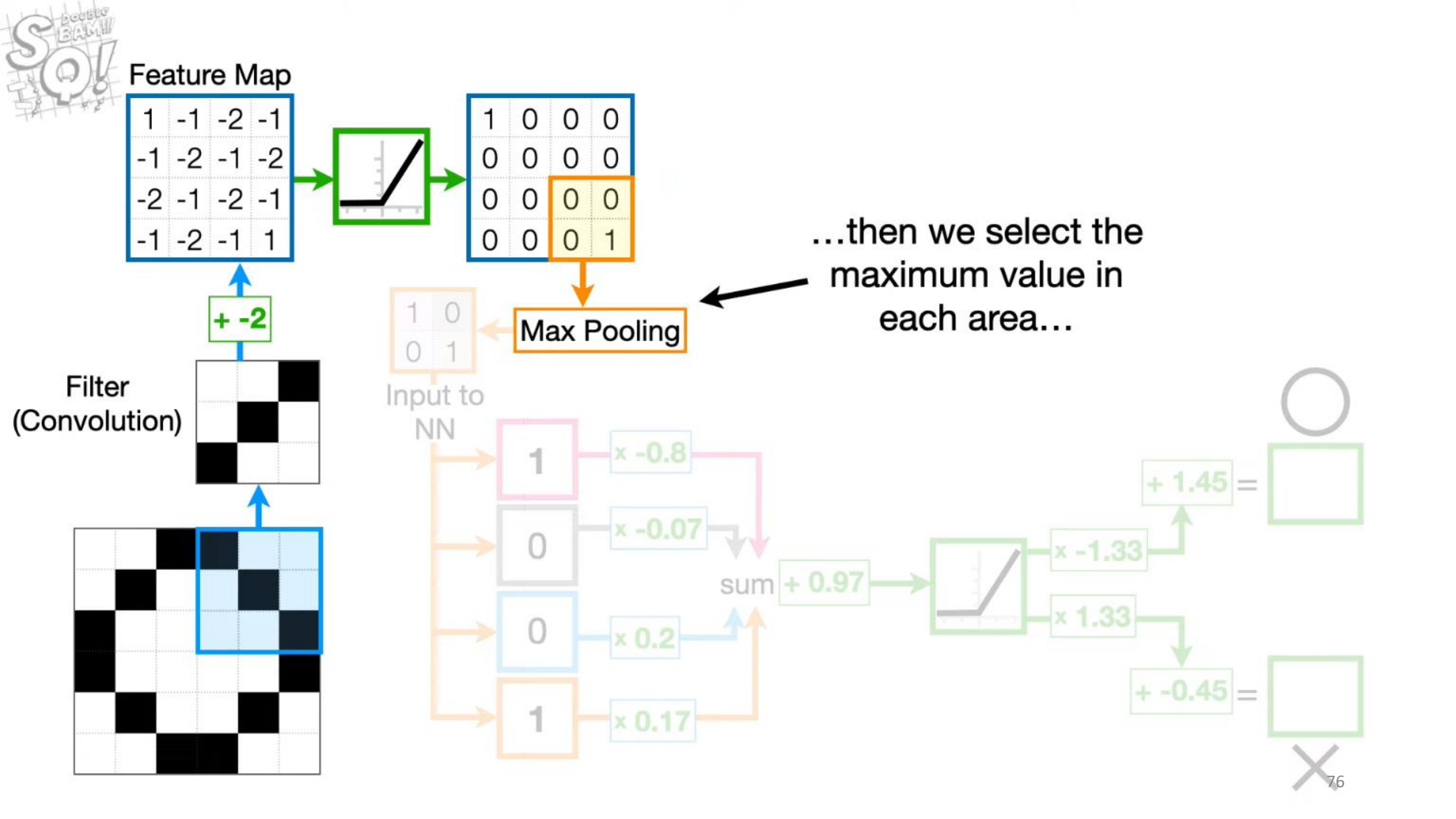

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
Max Pooling
Input to
NN
So, given this image…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 1.45 =
x -1.33
x 1.33
+ -0.45 =

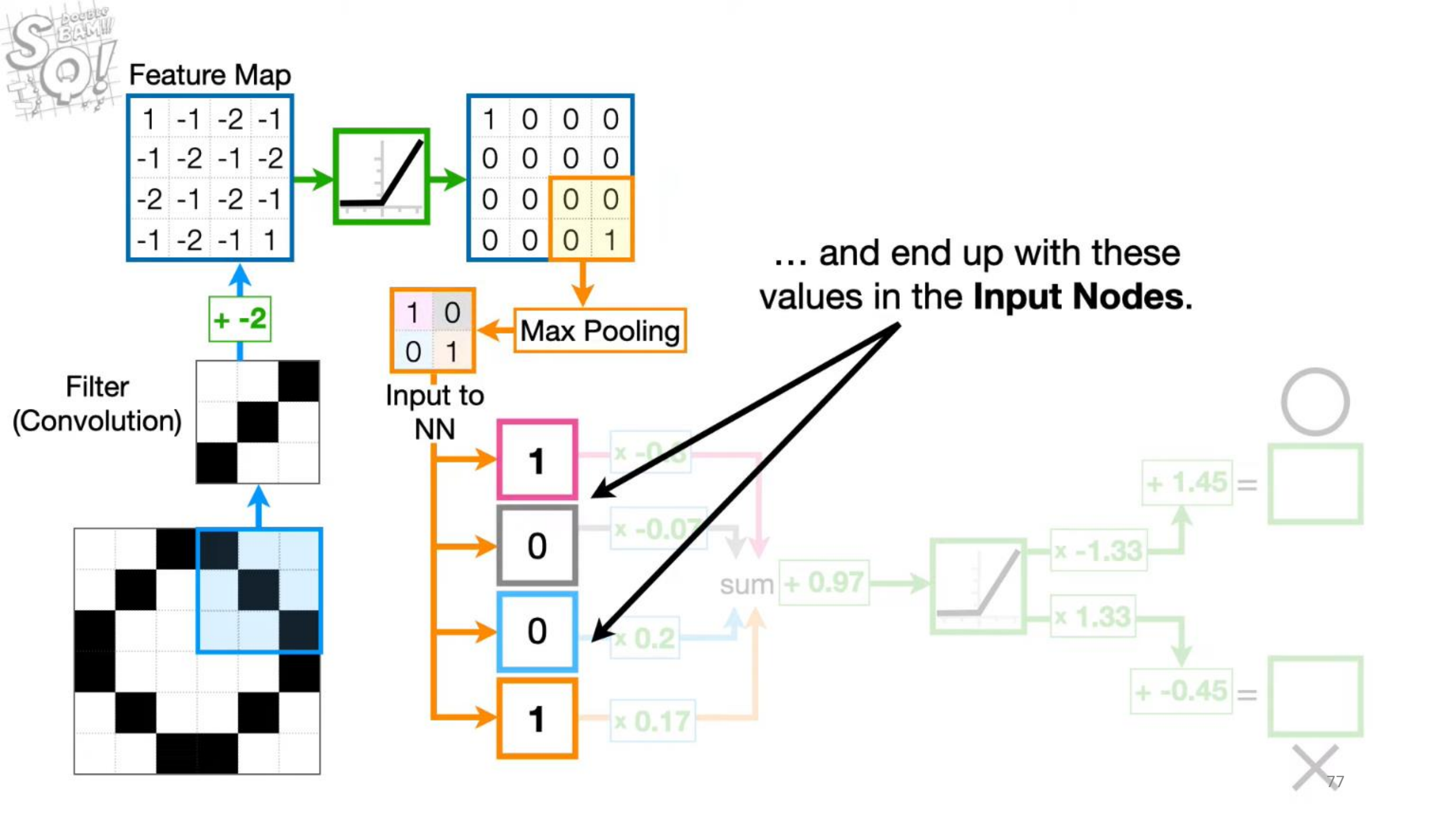

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
+ -2
Filter
(Convolution)
1 0
0 1
Max Pooling
Input to
NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =
…we run it through
the Filter to create
the Feature Map…

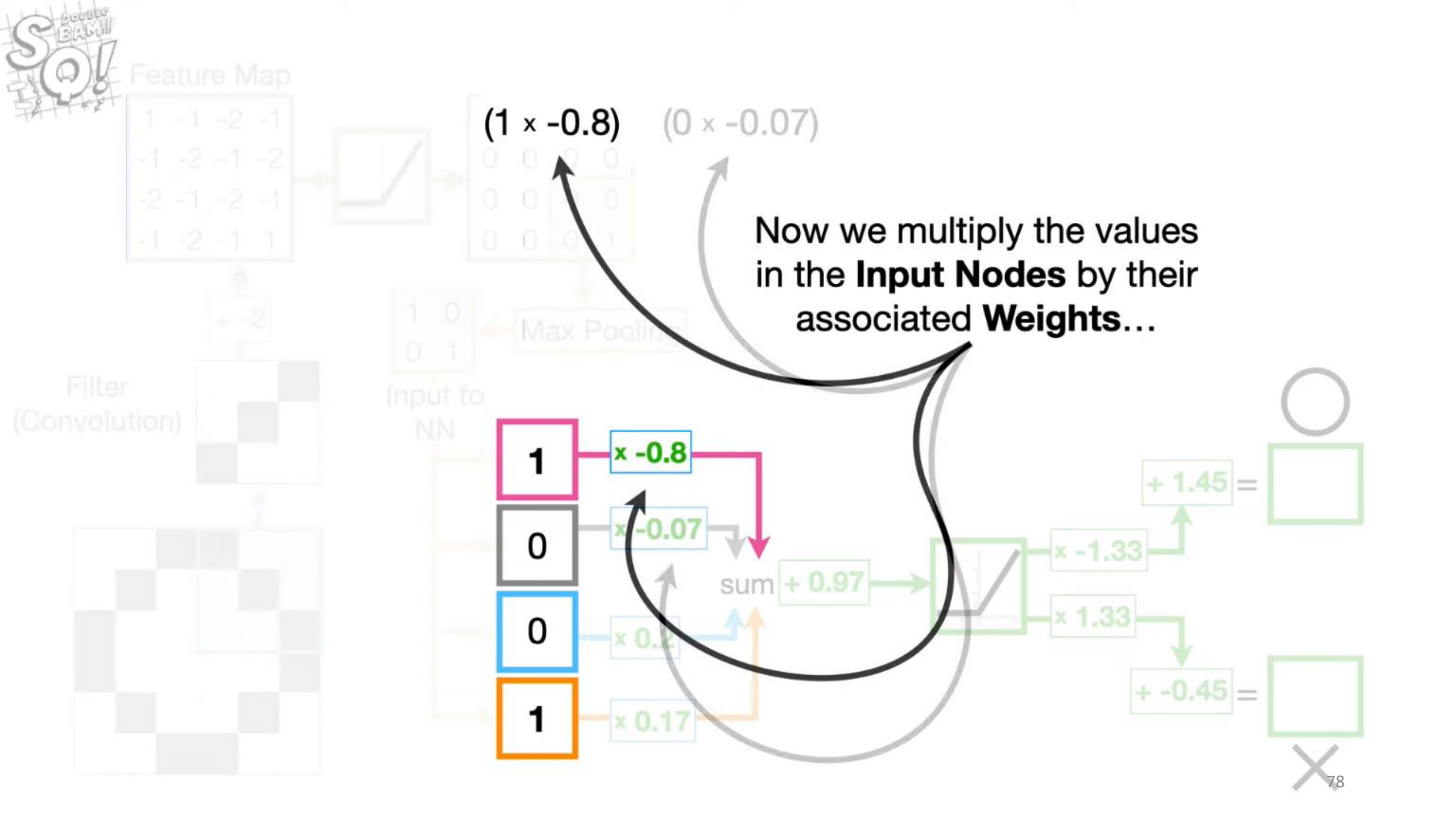

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
Max Pooling
Input to
NN
…we run it through
the Filter to create
the Feature Map…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =

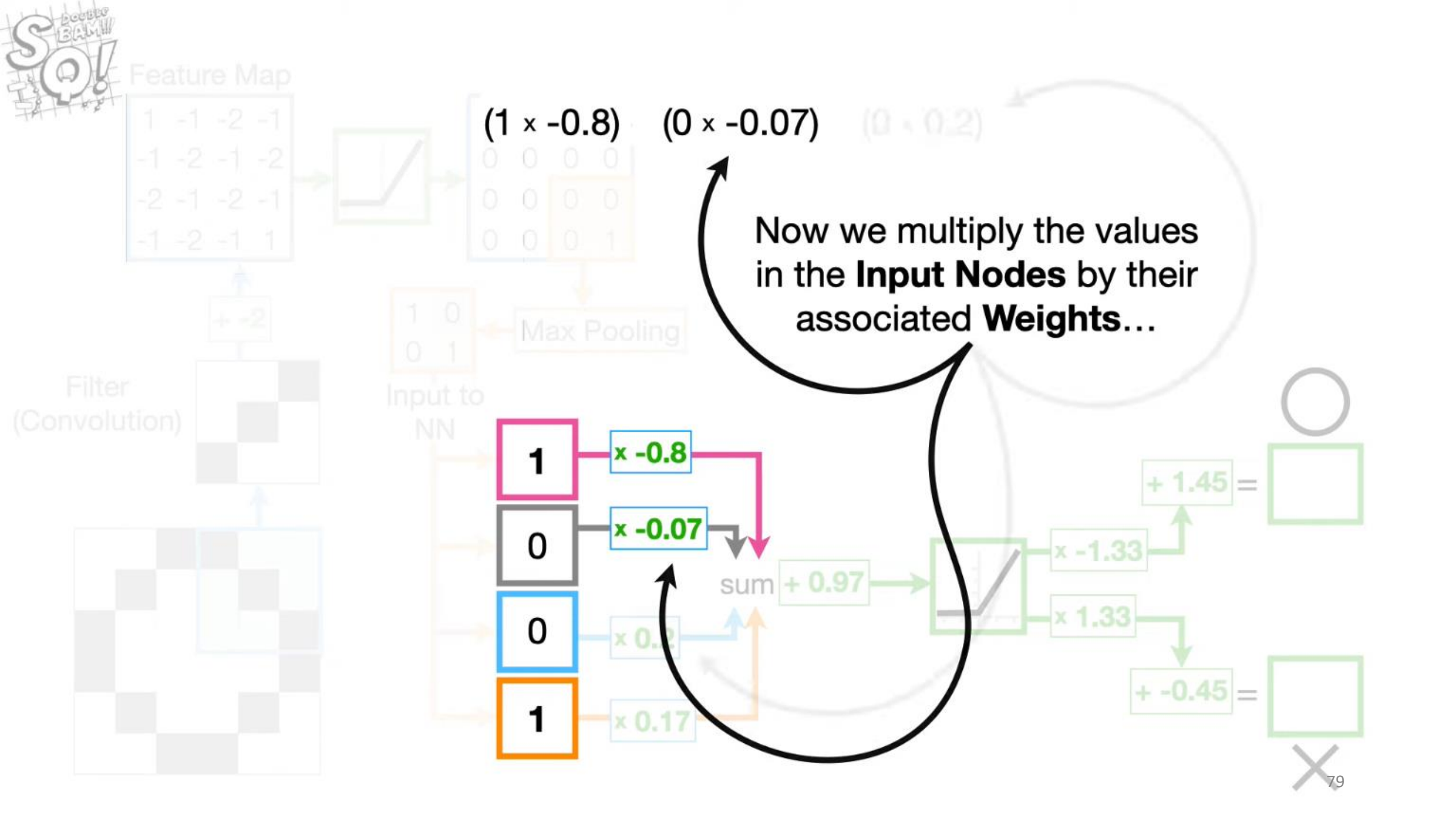

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
Max Pooling
Input to
NN
…then we run the
Feature Map through
a ReLU Activation
Function…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =


Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
…then we run the Feature Map through a ReLU Activation Function…
+ -2
Filter (Convolution)
1 0
0 1
Max Pooling
Input to NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =


Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
+ -2
Filter
(Convolution)
Max Pooling
1 0
0 1
Input to
NN
…then we select the
maximum value in
each area…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =

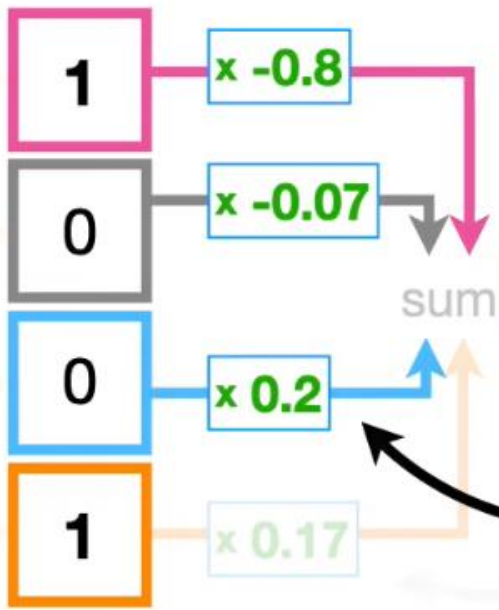

Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter
(Convolution)
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
Max Pooling
Input to
NN
… and end up with these
values in the Input Nodes.
1
0
0
1
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =

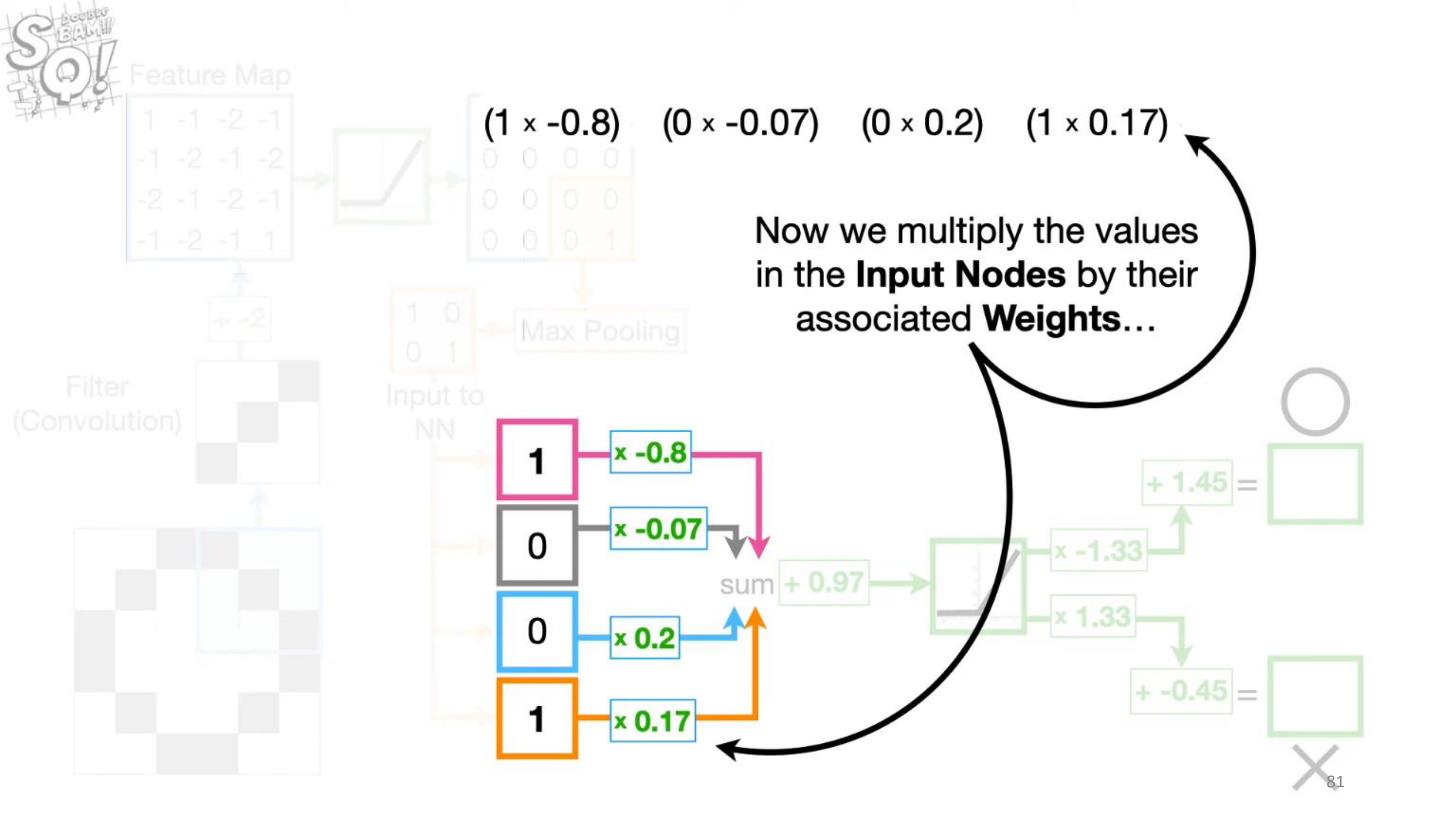

(1 x -0.8)
(0 x -0.07)
Now we multiply the values in the **Input Nodes** by their associated **Weights**…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =

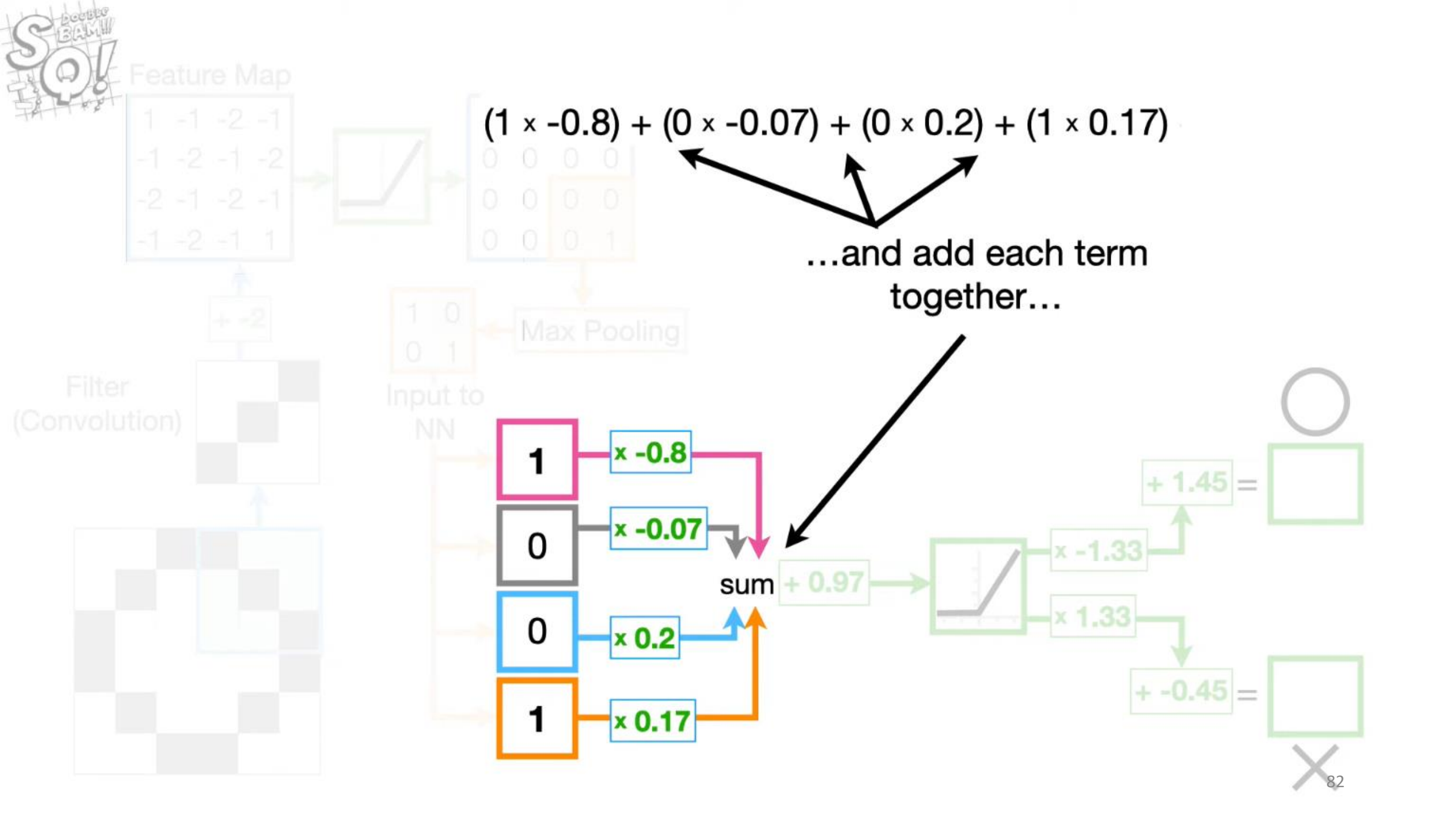

(1 x -0.8)
(0 x -0.07)
Now we multiply the values in the **Input Nodes** by their associated **Weights**…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =
Feature Map
Max Pooling
Input to NN
Filter (Convolution)

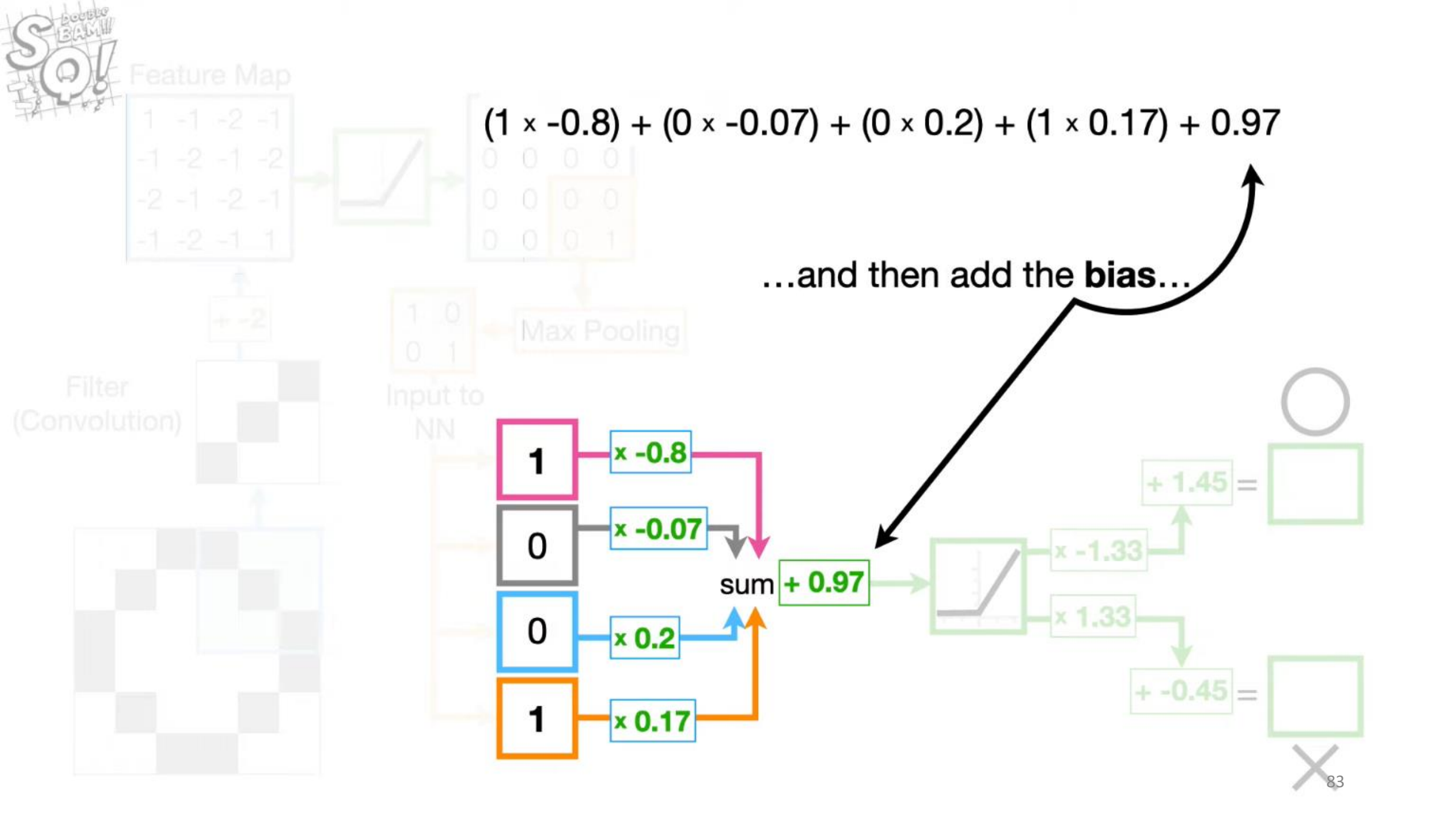
(1 x -0.8) (0 x -0.07) (0 x 0.2)

Now we multiply the values in the **Input Nodes** by their associated **Weights**…



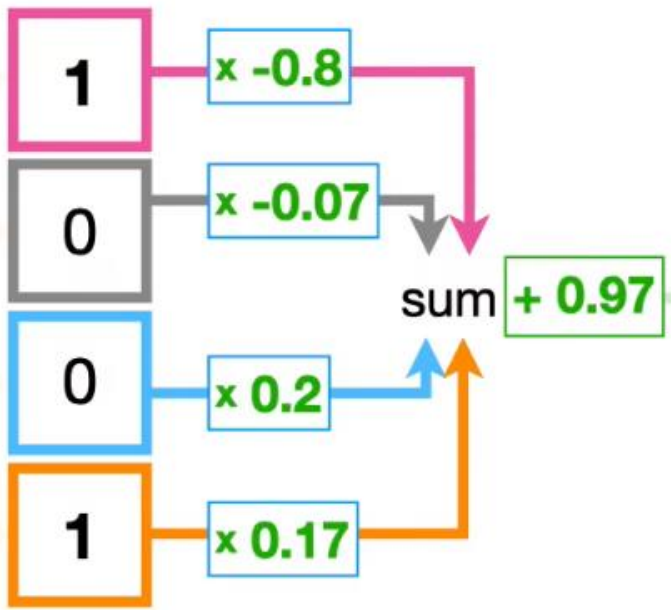

(1 x -0.8)
(0 x -0.07)
(0 x 0.2)
(1 x 0.17)
Now we multiply the values in the **Input Nodes** by their associated **Weights**…
1
x -0.8
0
x -0.07
0
x 0.2
1
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 =
+ -0.45 =
Feature Map
Max Pooling
Input to NN
Filter (Convolution)


(1 x -0.8) + (0 x -0.07) + (0 x 0.2) + (1 x 0.17)
…and add each term together…
1
x -0.8
0
x -0.07
0
x 0.2
1
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45
+ -0.45
Feature Map
Max Pooling
Filter (Convolution)
Input to NN


(1 x -0.8) + (0 x -0.07) + (0 x 0.2) + (1 x 0.17) + 0.97
…and then add the **bias**…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
+ 1.45 =
x -1.33
x 1.33
+ -0.45 =

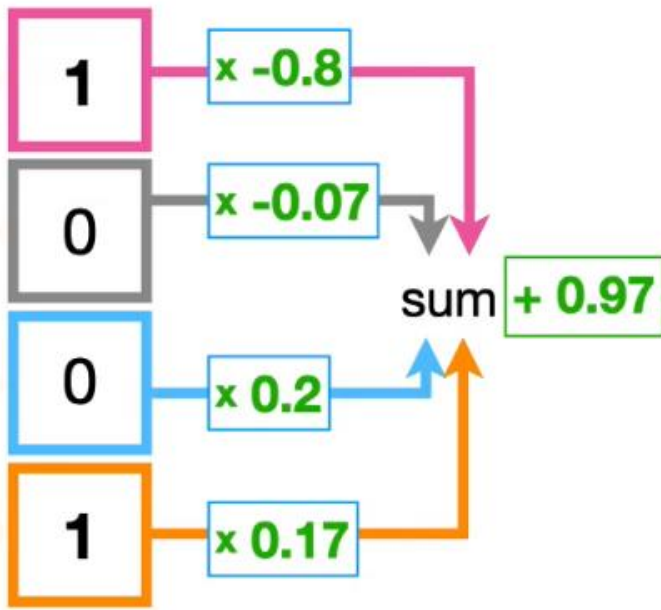
$(1 \times -0.8) + (0 \times -0.07) + (0 \times 0.2) + (1 \times 0.17) + 0.97 = 0.34$

…and we end up with **0.34**.



$$(1 \times -0.8) + (0 \times -0.07) + (0 \times 0.2) + (1 \times 0.17) + 0.97 = 0.34$$

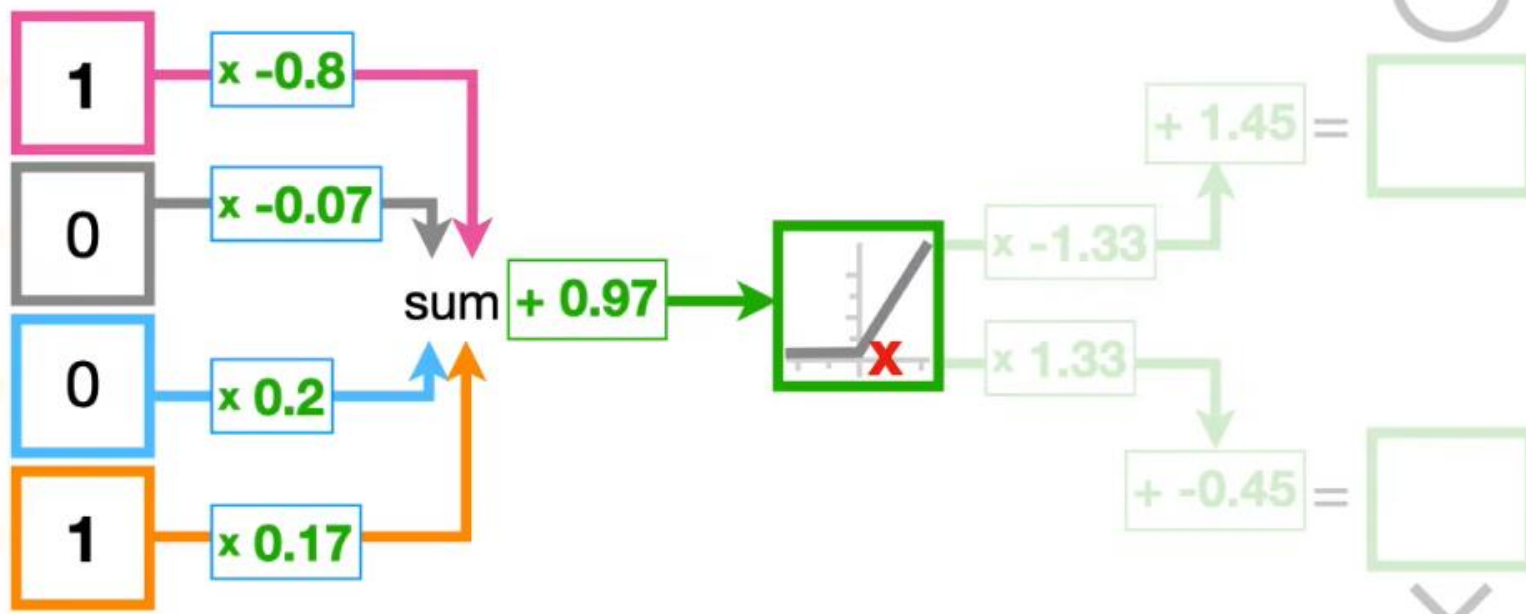
And thus, the x-axis coordinate for the **Activation Function** is **0.34**.



$$(1 \times -0.8) + (0 \times -0.07) + (0 \times 0.2) + (1 \times 0.17) + 0.97 = 0.34$$

$f(x) = \max(0, x) = \text{y-axis coordinate}$

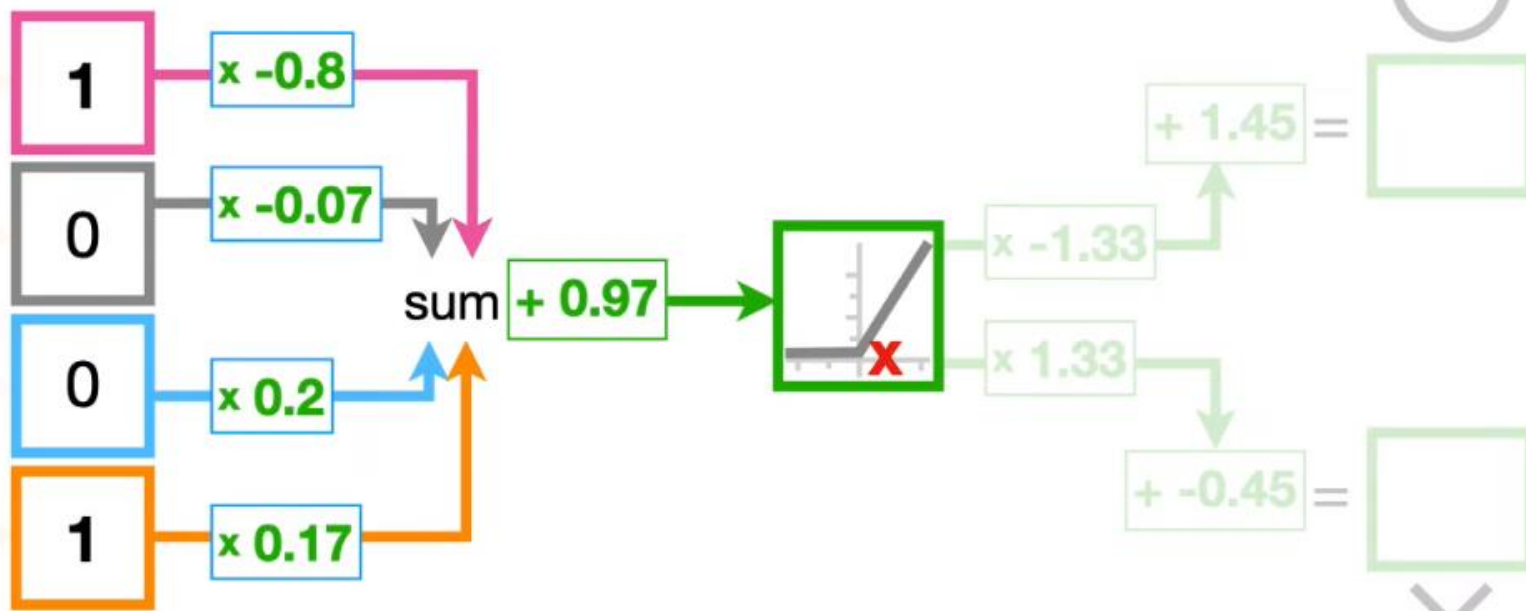
Now we plug **0.34** into the **ReLU Activation Function**…

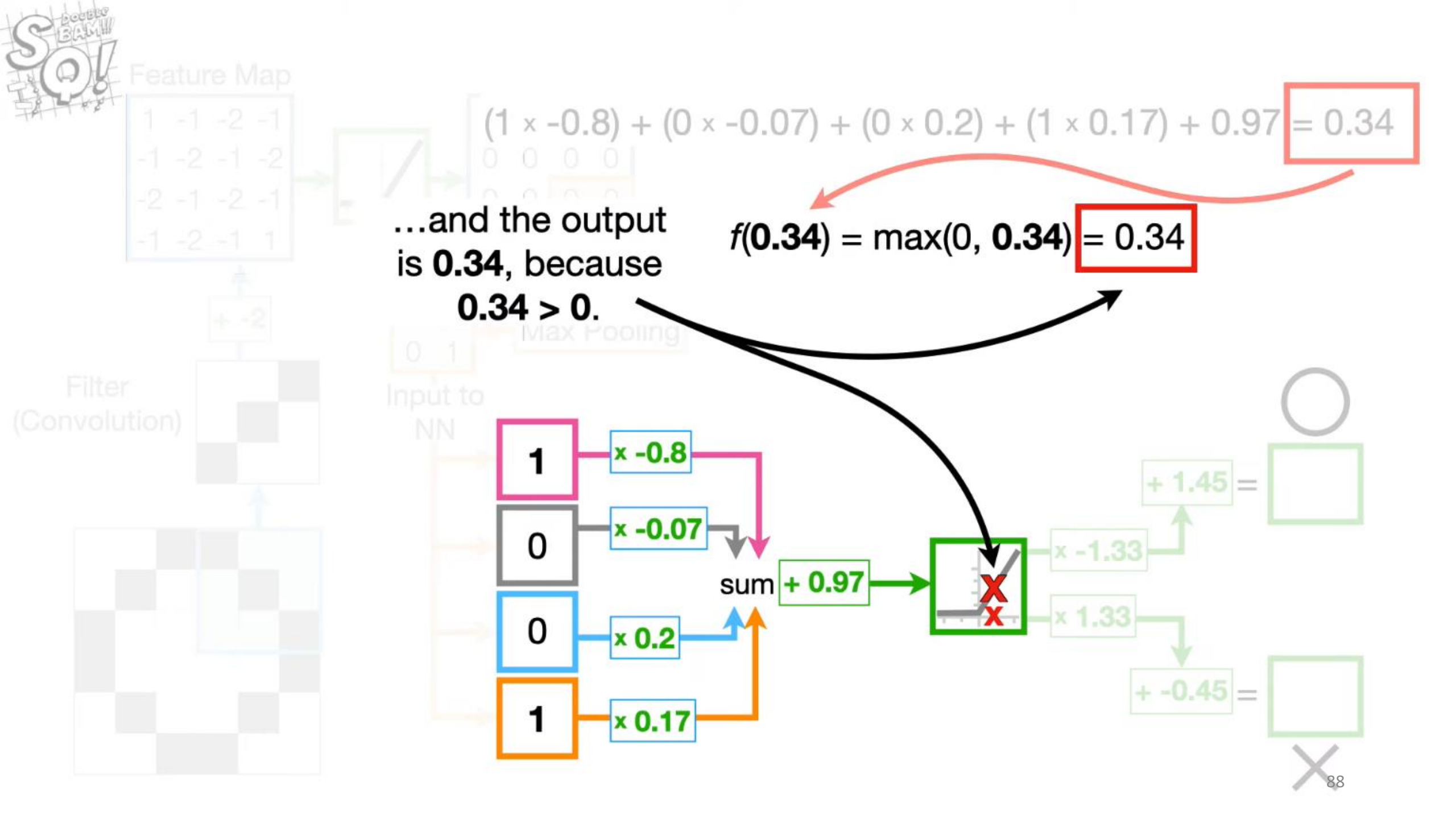


$(1 \times -0.8) + (0 \times -0.07) + (0 \times 0.2) + (1 \times 0.17) + 0.97 = 0.34$

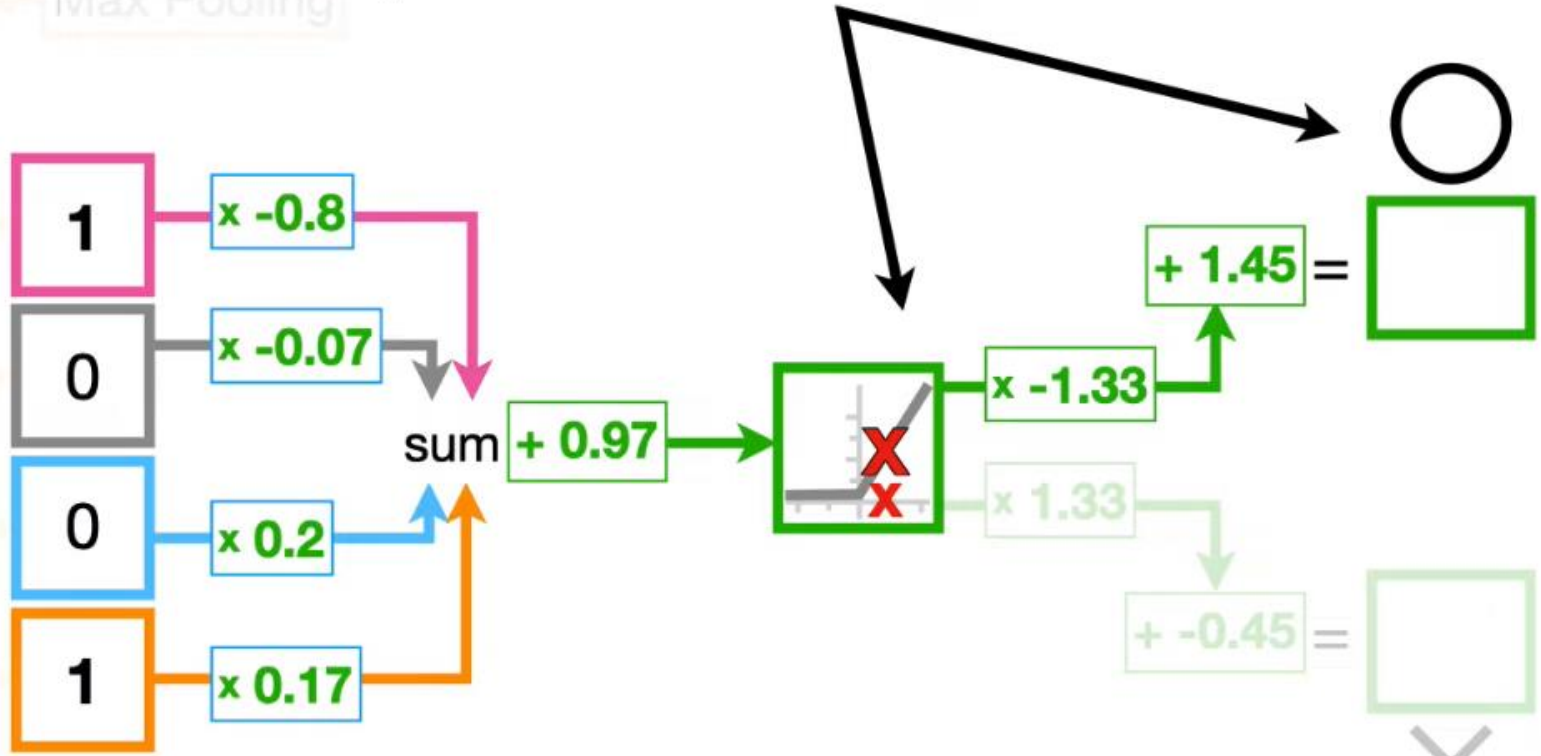
Now we plug **0.34** into the **ReLU Activation Function**…

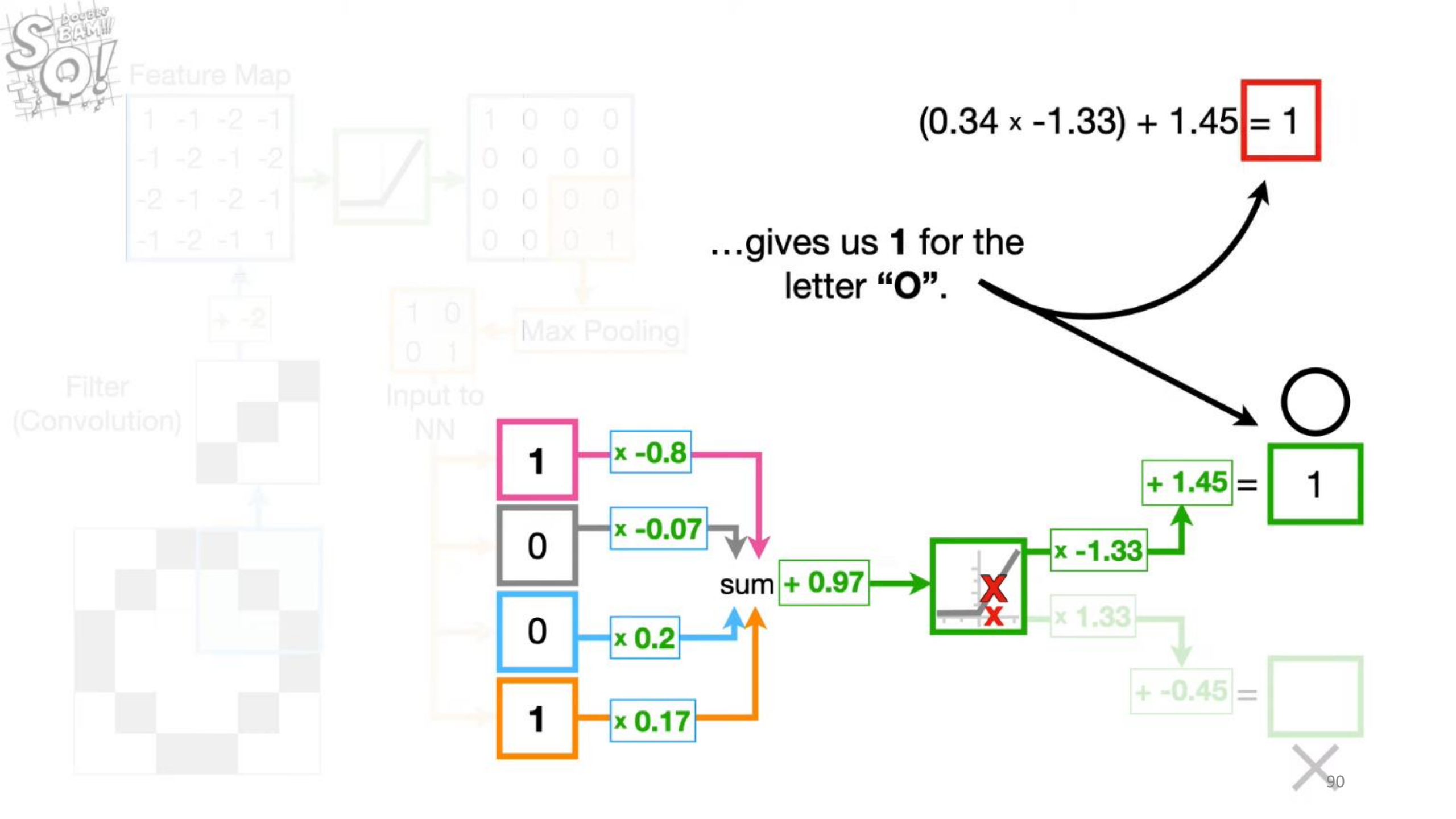
$f(\mathbf{0.34}) = \max(0, \mathbf{0.34}) =$ y-axis coordinate

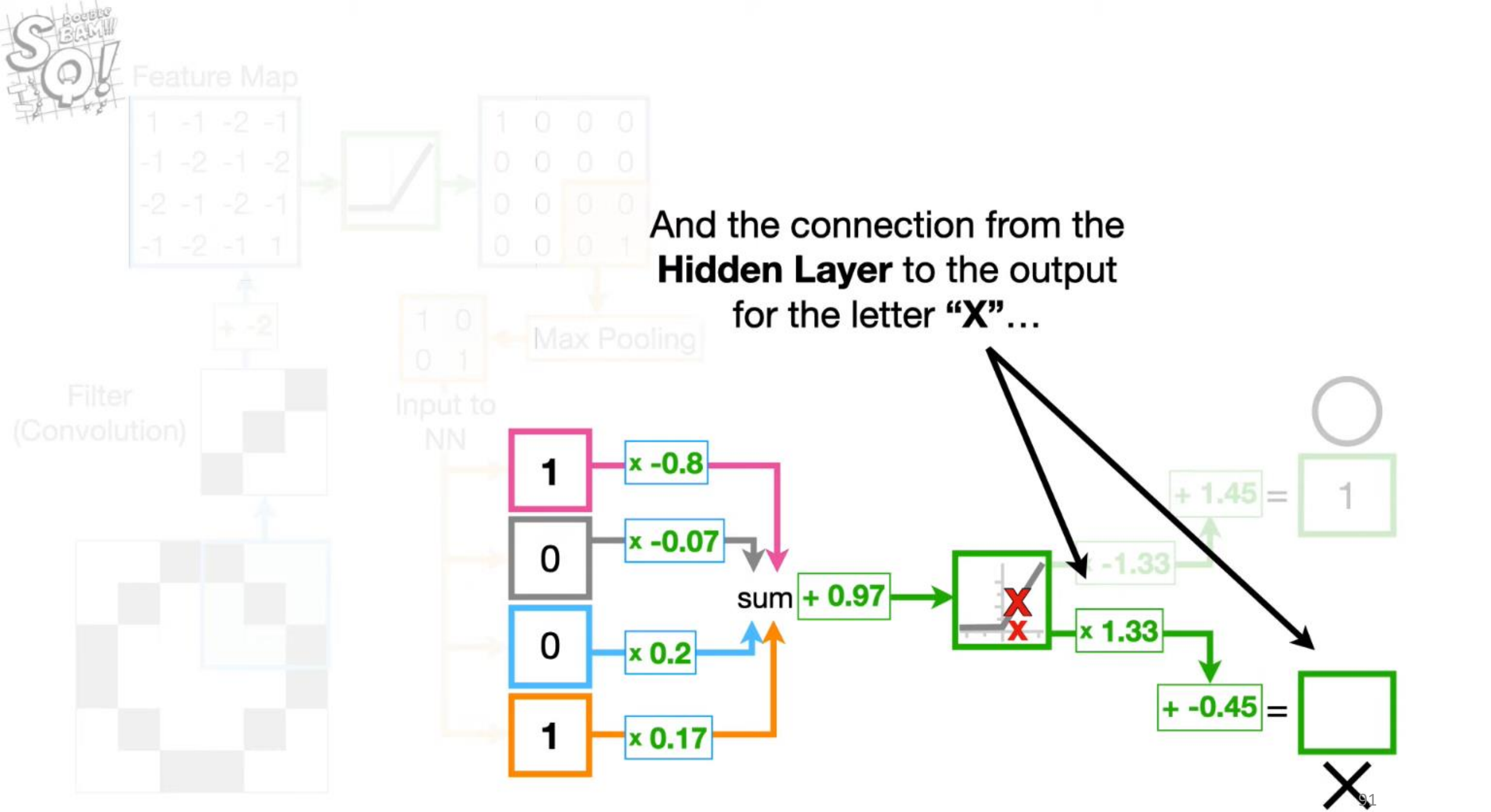


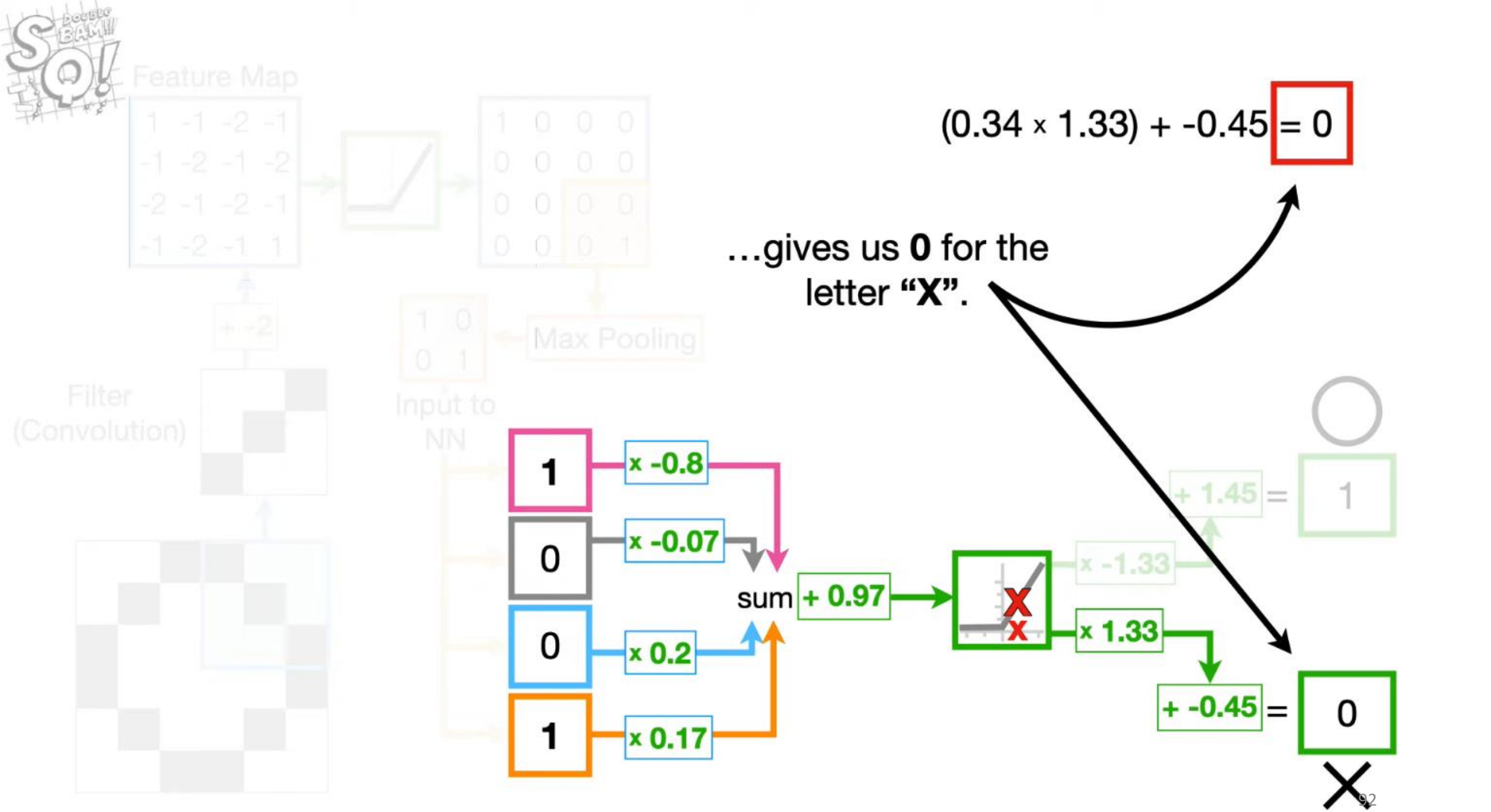

(1 x -0.8) + (0 x -0.07) + (0 x 0.2) + (1 x 0.17) + 0.97 = 0.34
…and the output is **0.34**, because **0.34 > 0**.
f(**0.34**) = max(0, **0.34**) = 0.34
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
+ 1.45 =
x -1.33
x 1.33
+ -0.45 =

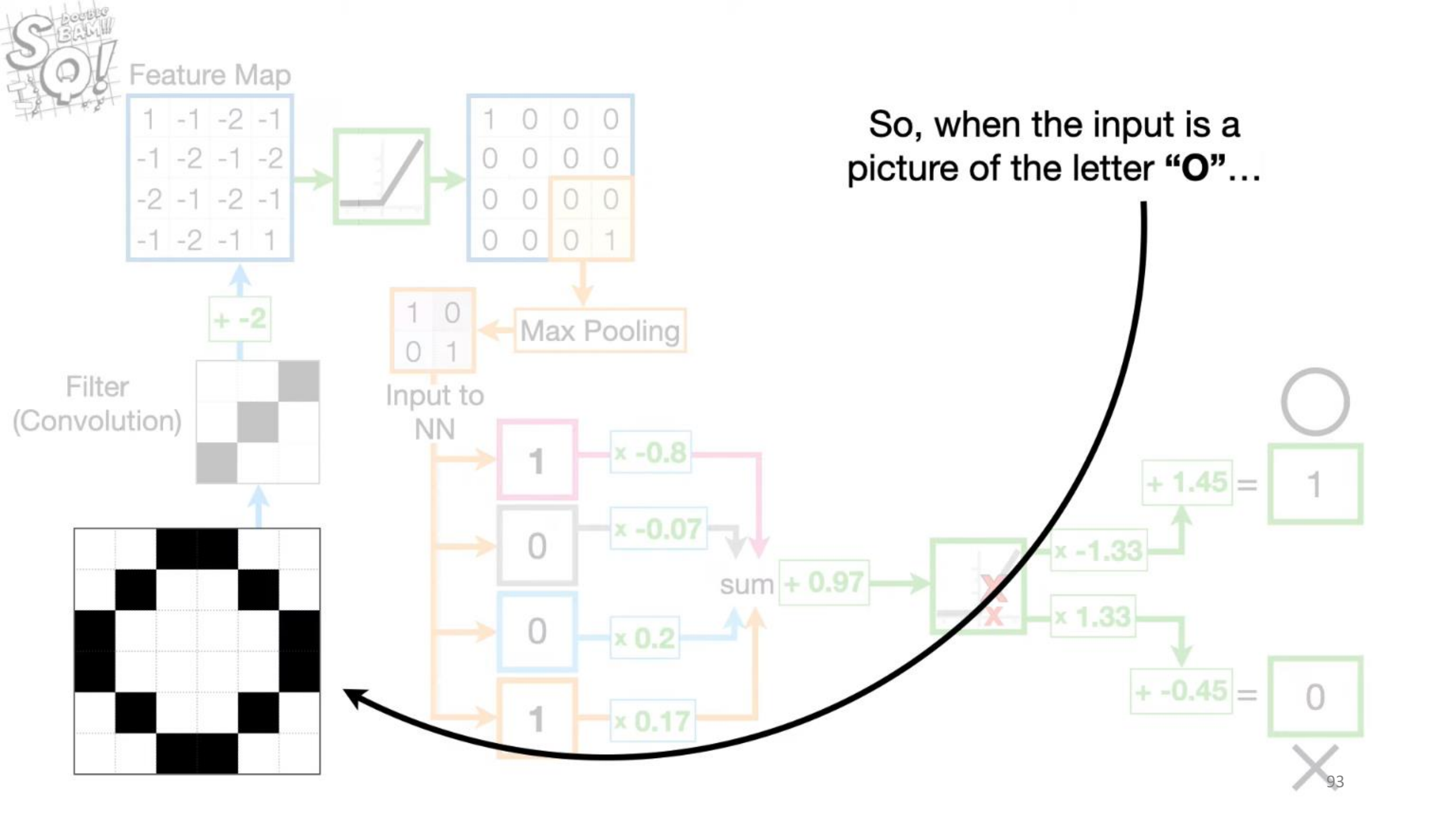
Now, this connection, from the **Hidden Layer** to the output for the letter **"O"**…

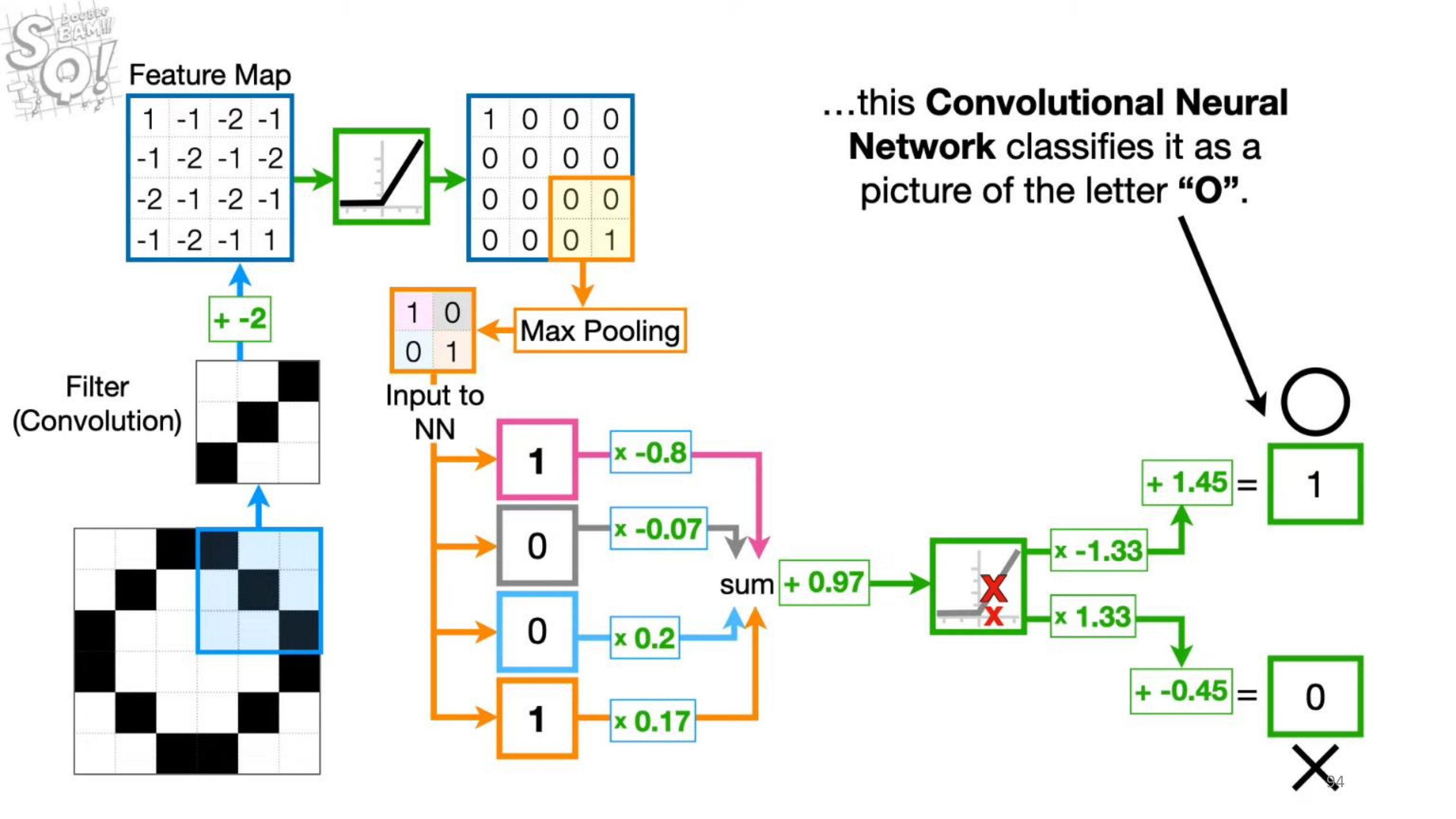



(0.34 x -1.33) + 1.45 = 1
…gives us 1 for the letter “O”.
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
+ 1.45 = 1
x 1.33
+ -0.45 =


Feature Map
Max Pooling
Filter (Convolution)
Input to NN
And the connection from the **Hidden Layer** to the output for the letter "X"…
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
-1.33
x 1.33
+ -0.45 =
+ 1.45 =
1


(0.34 × 1.33) + -0.45 = 0
…gives us 0 for the letter "X".
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ -0.45 = 0
+ 1.45 = 1


So, when the input is a picture of the letter "O"…
Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
Max Pooling
+ -2
Filter (Convolution)
Input to NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 = 1
+ -0.45 = 0


Feature Map
1 -1 -2 -1
-1 -2 -1 -2
-2 -1 -2 -1
-1 -2 -1 1
+ -2
Filter (Convolution)
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
1 0
0 1
Max Pooling
Input to NN
1
0
0
1
x -0.8
x -0.07
x 0.2
x 0.17
sum
+ 0.97
x -1.33
x 1.33
+ 1.45 = 1
+ -0.45 = 0
…this Convolutional Neural Network classifies it as a picture of the letter "O".

Summary:

- Convolution Layer → Feature Map.
- Max-Pooling Layer → Max Value from group of Neighboring values.
- Fully Connected layer(NN) → Understand high level features extracted by Previous layers.

About Time Complexity…

Time complexity is not a crucial aspect of machine learning and neural networks and it's not always the primary focus of discussion for a few reasons

- **Empirical Emphasis**
- **Algorithmic Complexity**
- **Complexity Trade-offs**
- **Parallelization and Distributed Computing**

Time Complexity:

$$\mathbf{O(N^2 * K^2 * C\_in * C\_out + N^2 * C\_out * P^2 + L * M^2)}$$

- Input feature map size: N x N
- Kernel size: K x K
- Number of input channels: C_in
- Number of output channels (filters): C_out

  O(N2 * K^2 * C_in * C_out)

- Input feature map size: N x N
- Number of input channels: C_out (from the previous layer)
- Pooling window size: P x P

  O(N^2 * C_out * P^2)

- Assuming you have L fully connected layers with the following parameters:
- Number of neurons in each fully connected layer: M

  O(L * M^2)

THANK YOU !