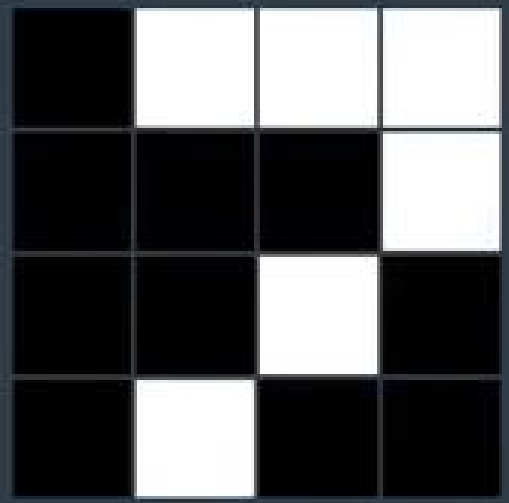
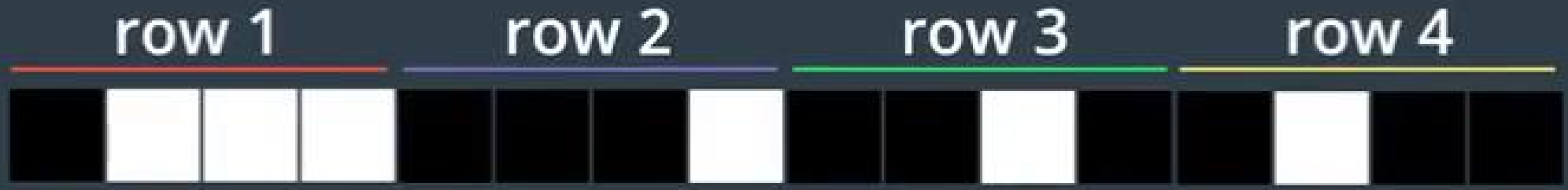
MNIST Database:

Centered, heavily pre-processed images

VS

Flattening



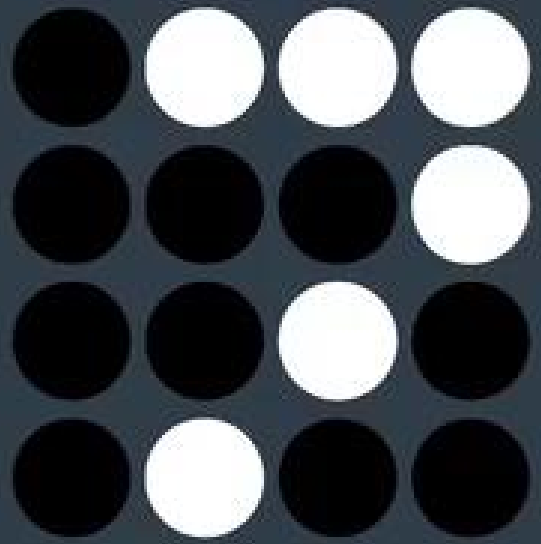


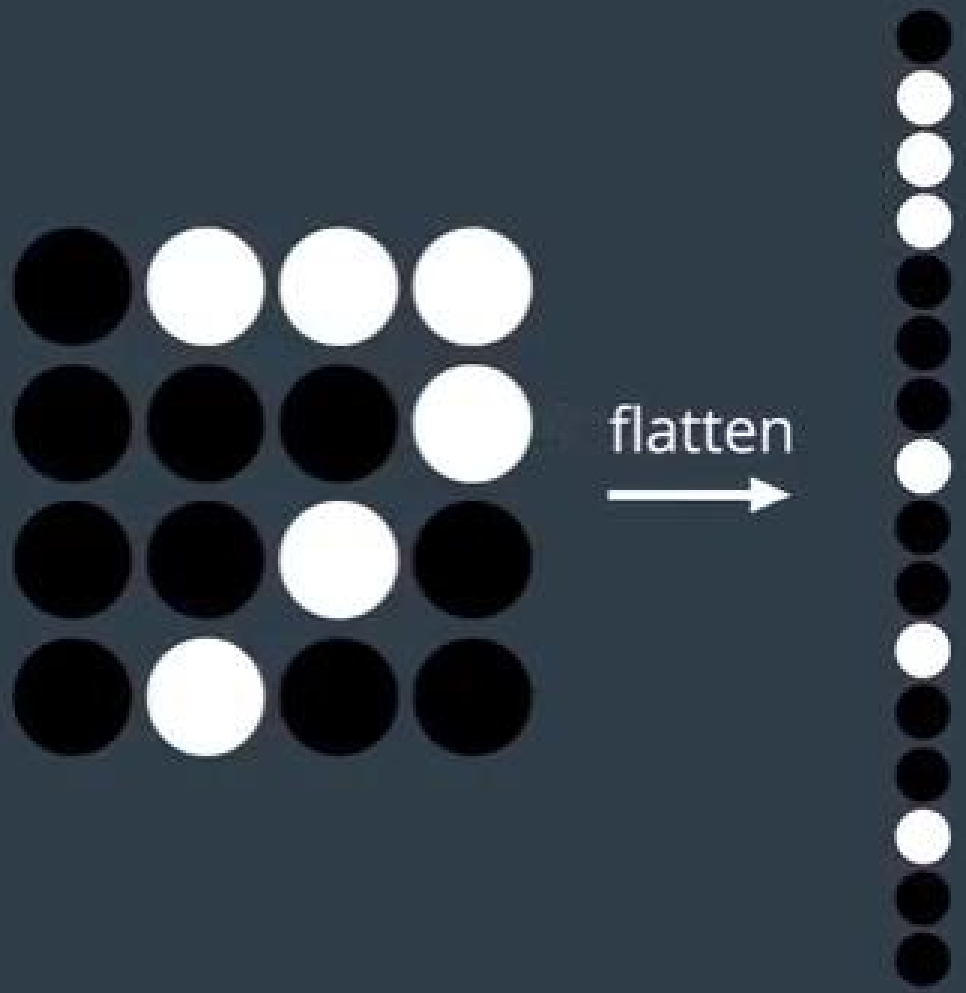
MLPs

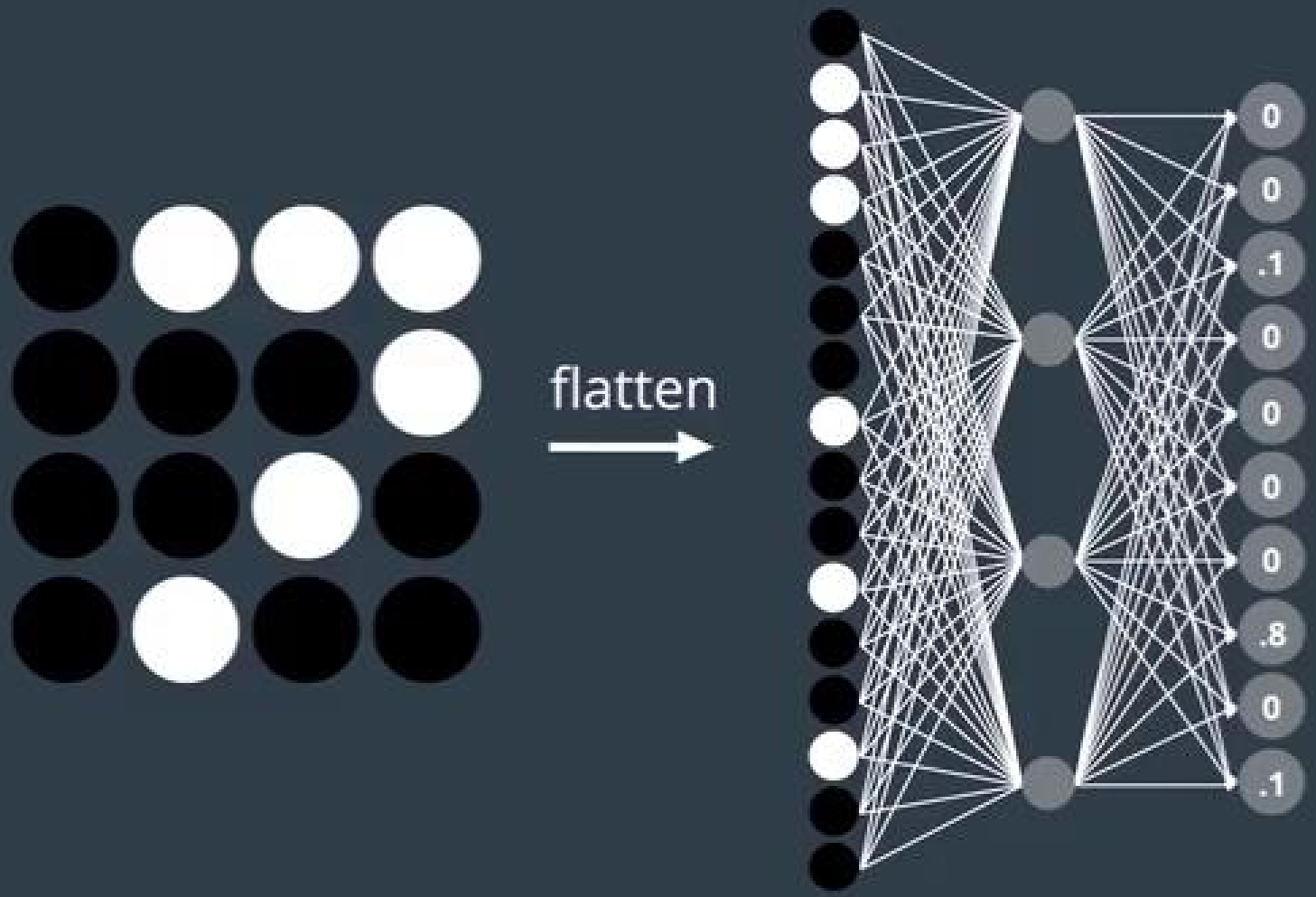
- Only use **fully** connected layers
- Only accept **vectors** as input

CNNs

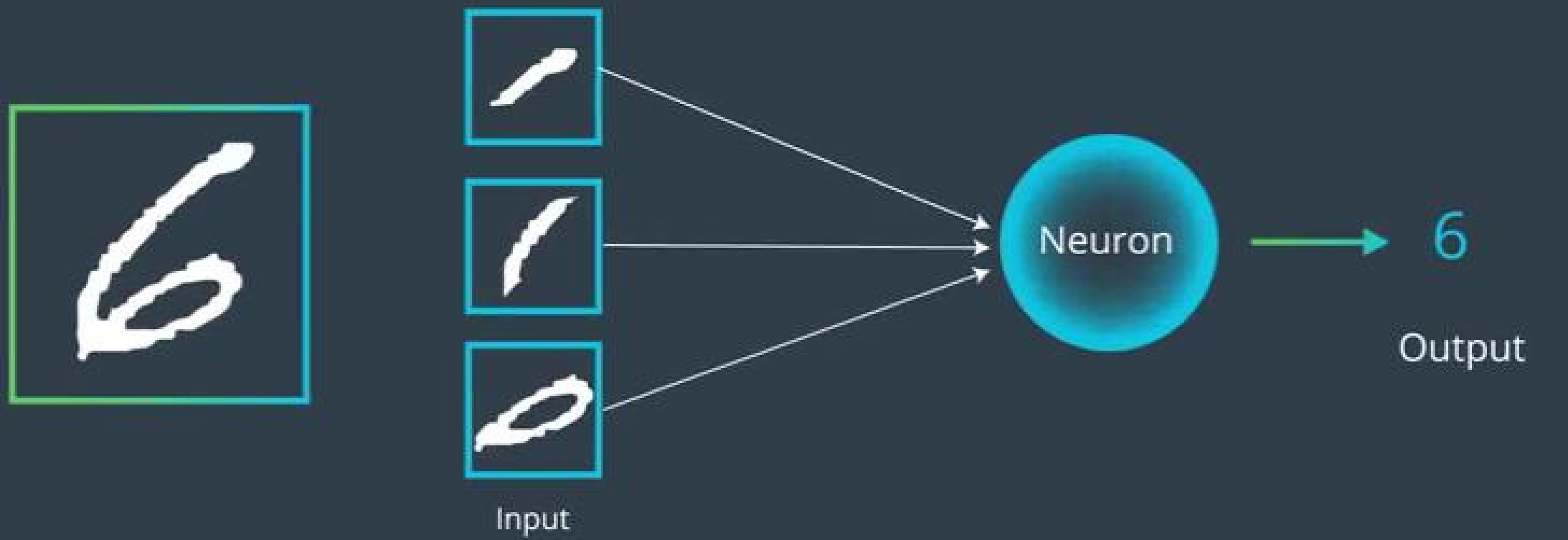
- Also use **sparsely** connected layers
- Also accept **matrices** as input

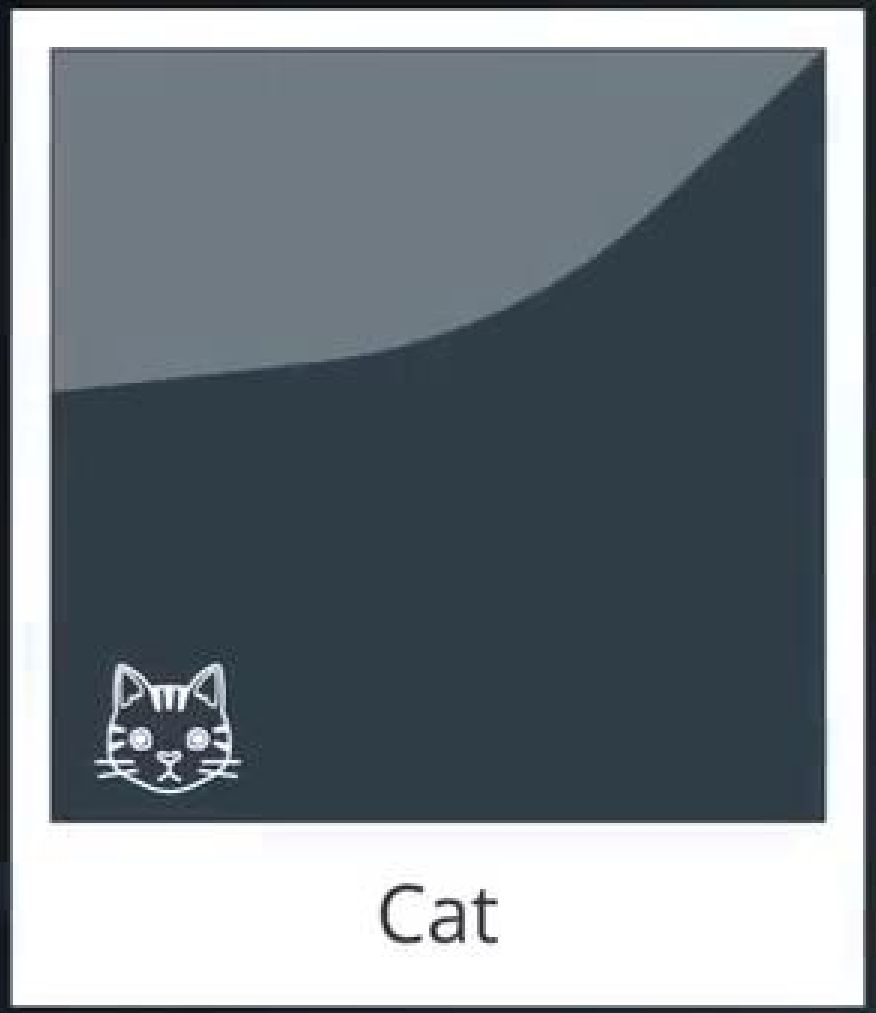


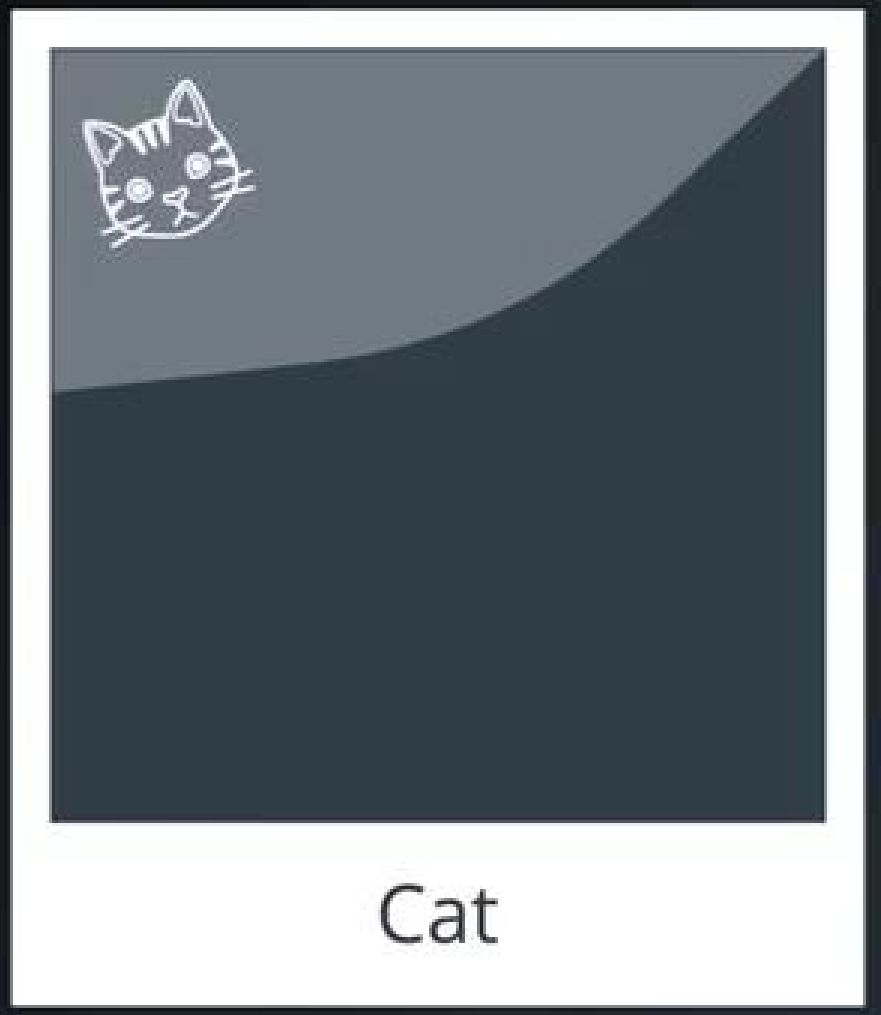

flatten


flatten
0
0
.1
0
0
0
0
.8
0
.1

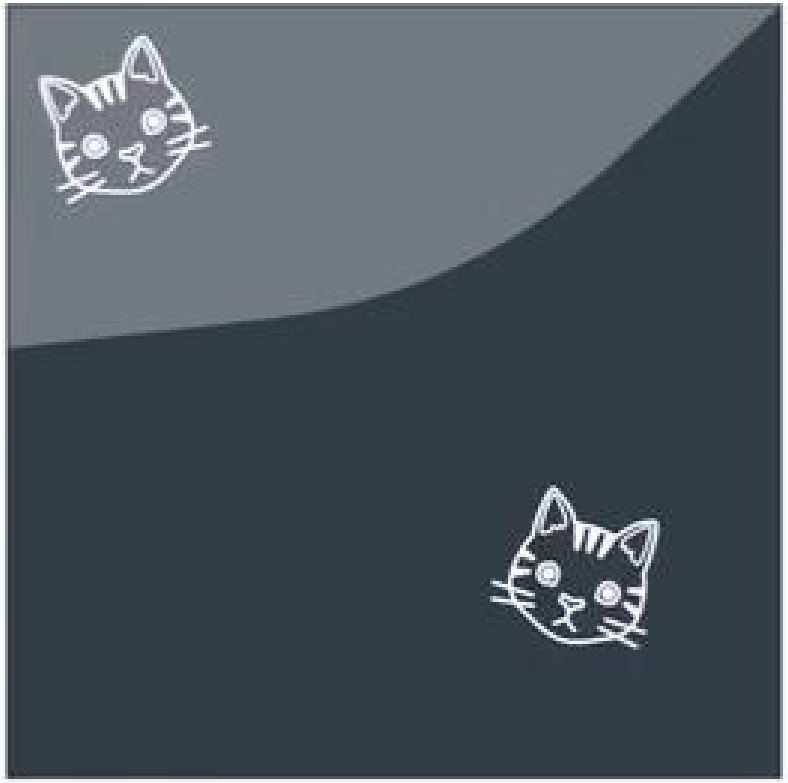
Input


Neuron
6
Output
Input


Cat



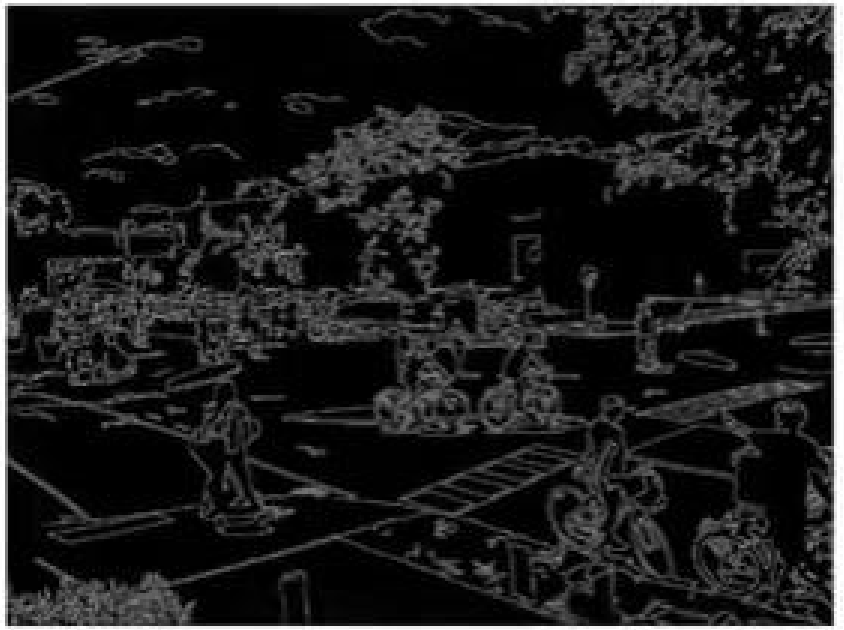
Cat



Cat

HIGH-PASS FILTERS

FILTERS



1. Filter out unwanted information
2. Amplify features of interest

HIGH-PASS FILTERS

HIGH-PASS FILTERS

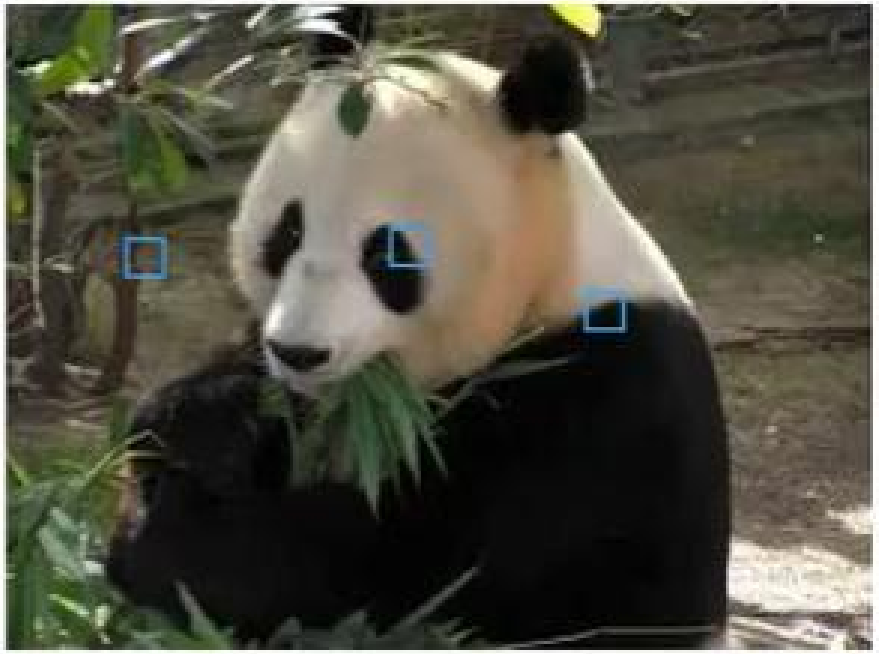
- Sharpen an image
- Enhance ***high-frequency*** parts of an image

HIGH-PASS FILTERS

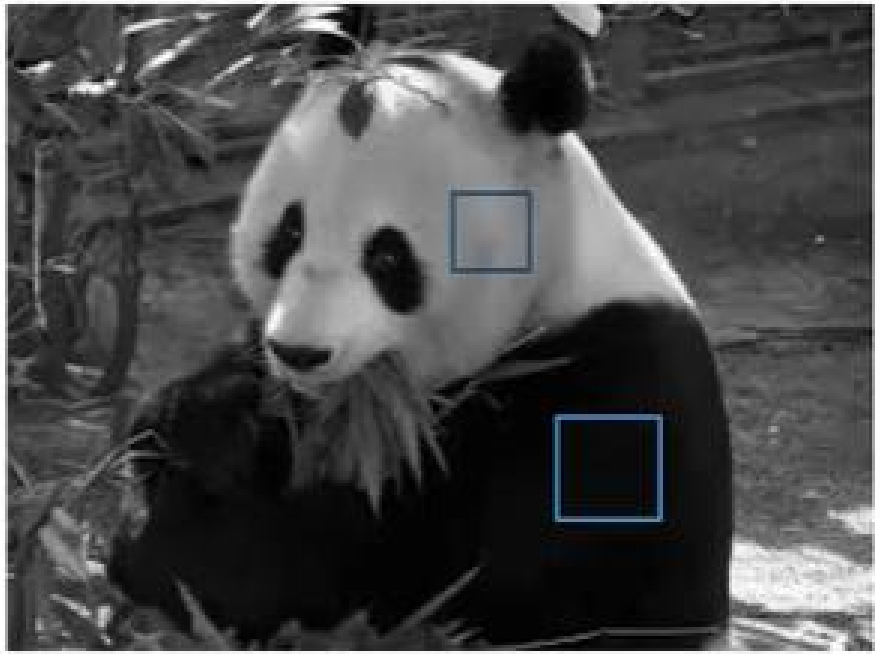
- Sharpen an image
- Enhance ***high-frequency*** parts of an image

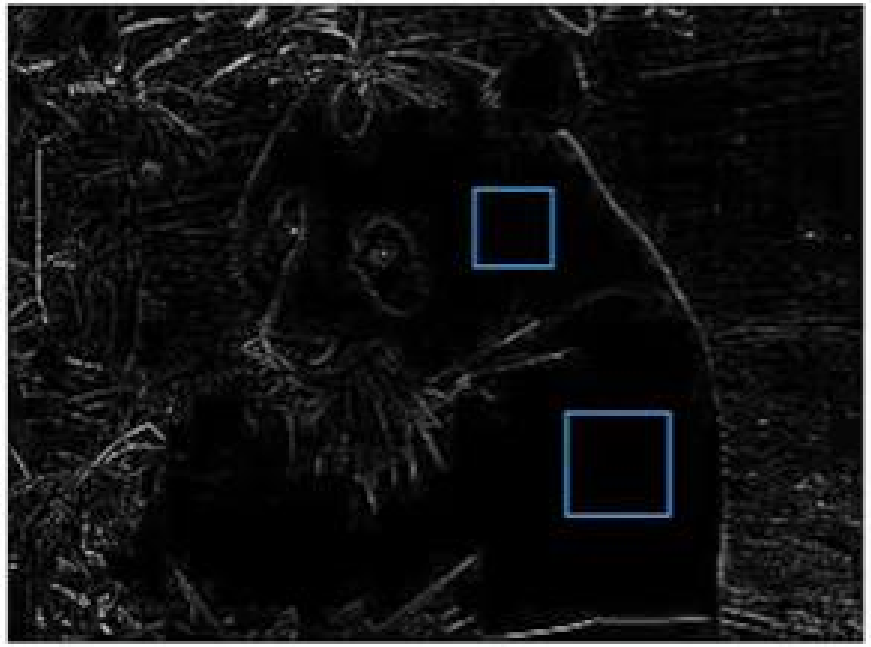
HIGH-PASS FILTERS

- Sharpen an image
- Enhance ***high-frequency*** parts of an image

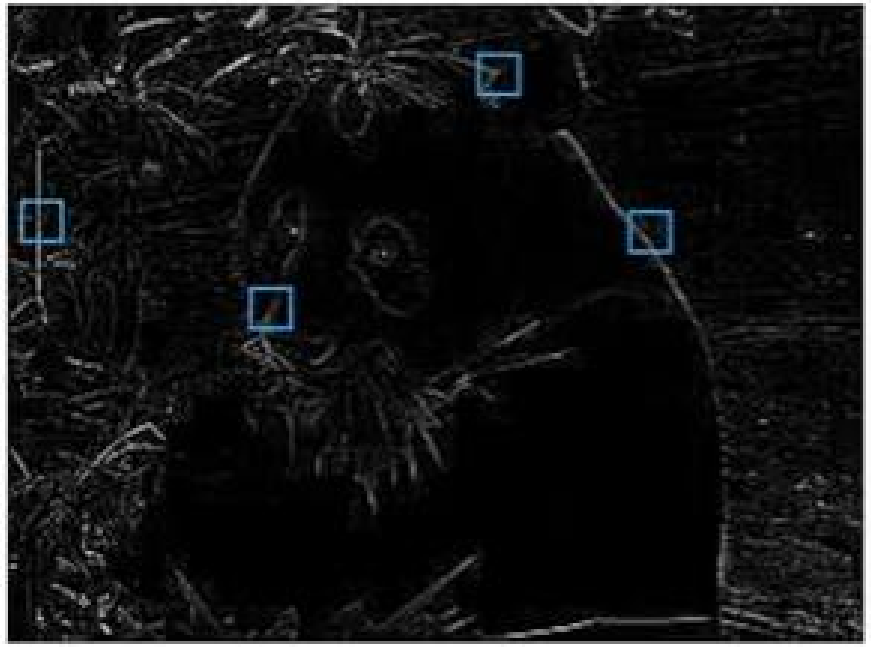


HIGH-PASS FILTERS



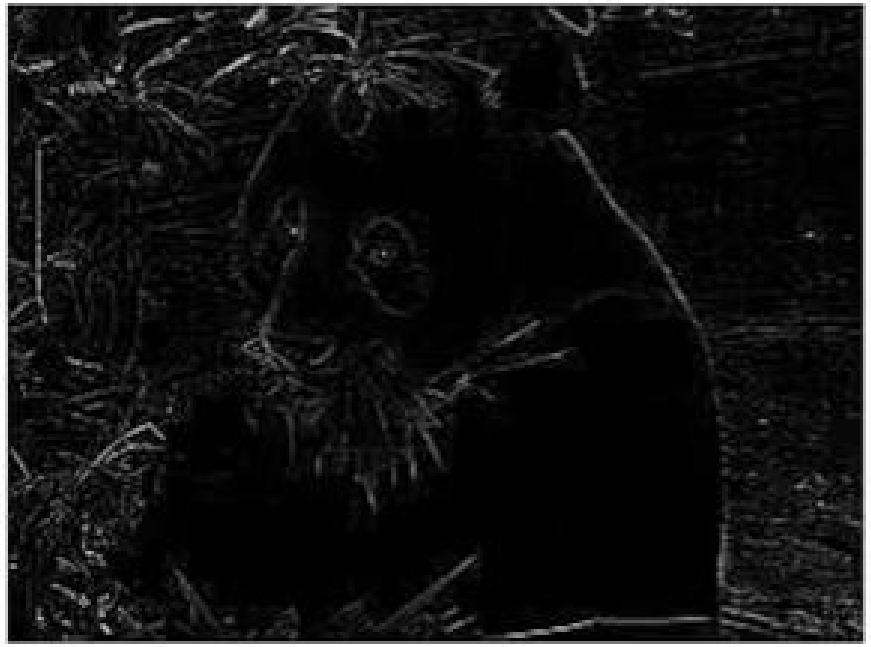


HIGH-PASS FILTERS



EDGE DETECTION

Emphasize Edges



Edges are areas in an image where the intensity changes very quickly, and they often indicate object boundaries

CONVOLUTION KERNELS

A kernel is a matrix of numbers that modifies an image

0	-1	0
-1	4	-1
0	-1	0

edge detection filter

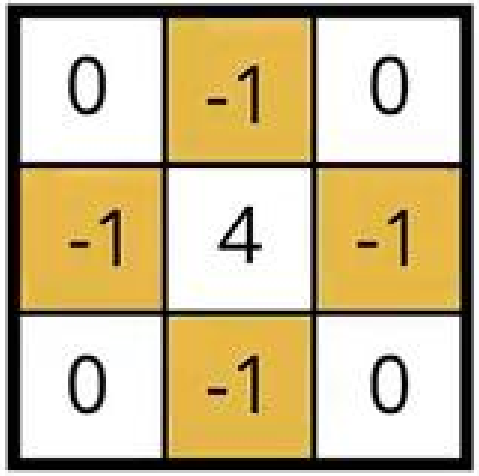
CONVOLUTION KERNELS

0	-1	0
-1	4	-1
0	-1	0

edge detection filter

$$0 + -1 + 0 + -1 + 4 + -1 + 0 + -1 + 0 = \mathbf{0}$$

CONVOLUTION KERNELS



edge detection filter

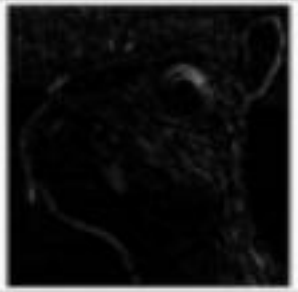
$$0 + -1 + 0 + -1 + 4 + -1 + 0 + -1 + 0 = \mathbf{0}$$

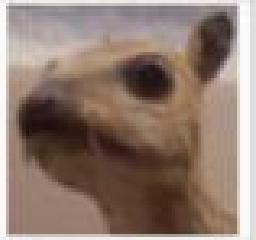
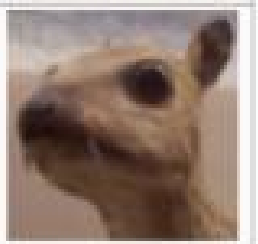
CONVOLUTION KERNELS

0	-1	0
-1	4	-1
0	-1	0

edge detection filter

$$0 + -1 + 0 + -1 + 4 + -1 + 0 + -1 + 0 = \mathbf{0}$$

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

CONVOLUTION KERNELS

0	-1	0
-1	**8**	-1
0	-1	0

0 + -1 + 0 + -1 + **8** + -1 + 0 + -1 + 0 = **4**

CONVOLUTION KERNELS

0	-1	0
-1	4	**-2**
0	-1	0

$$0 + -1 + 0 + -1 + 4 + \mathbf{-2} + 0 + -1 + 0 = \mathbf{-1}$$

CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

K

*

F(x,y)

K * F(x,y) = output image

CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

K * F(x,y) = output image

CONVOLUTION

K * F(x,y) = output image

CONVOLUTION

K * F(x,y) = output image

CONVOLUTION

K * F(x,y) = output image

CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

CONVOLUTION

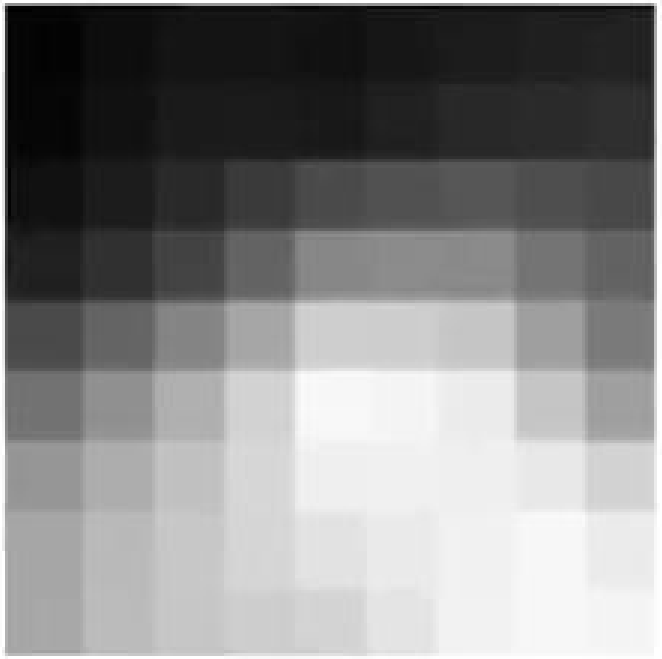
0	-1	0
-1	4	-1
0	-1	0

CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

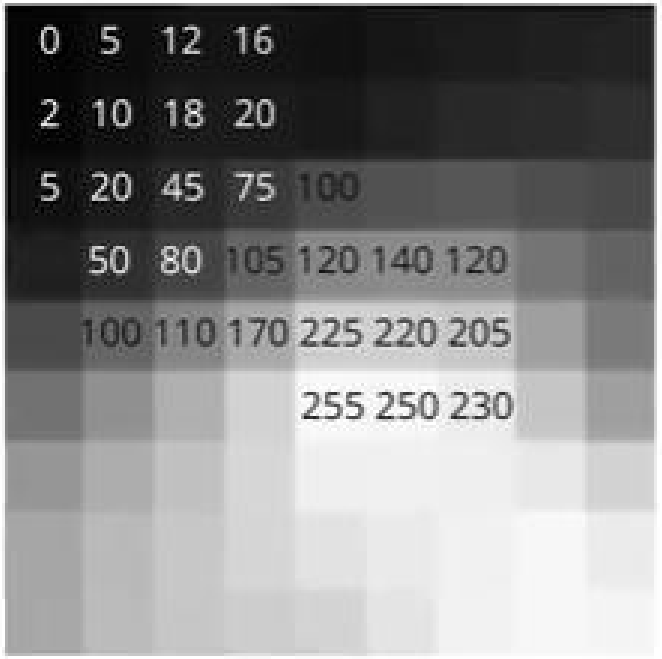
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



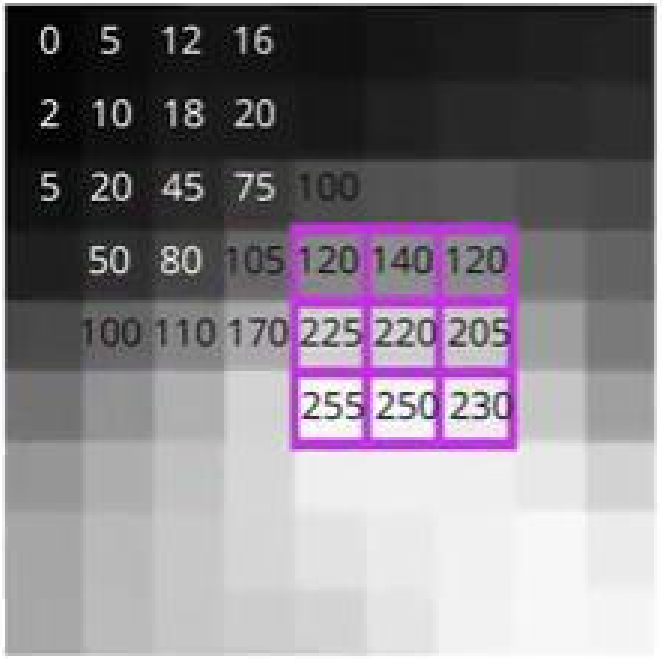
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



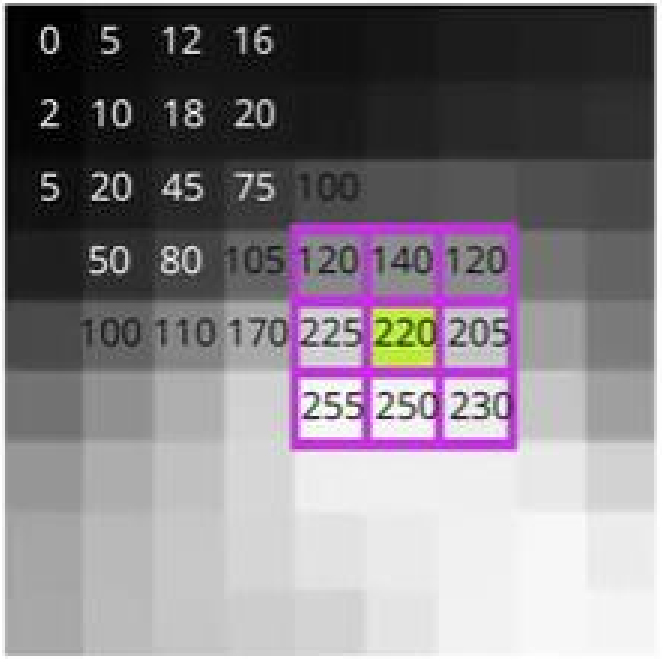
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



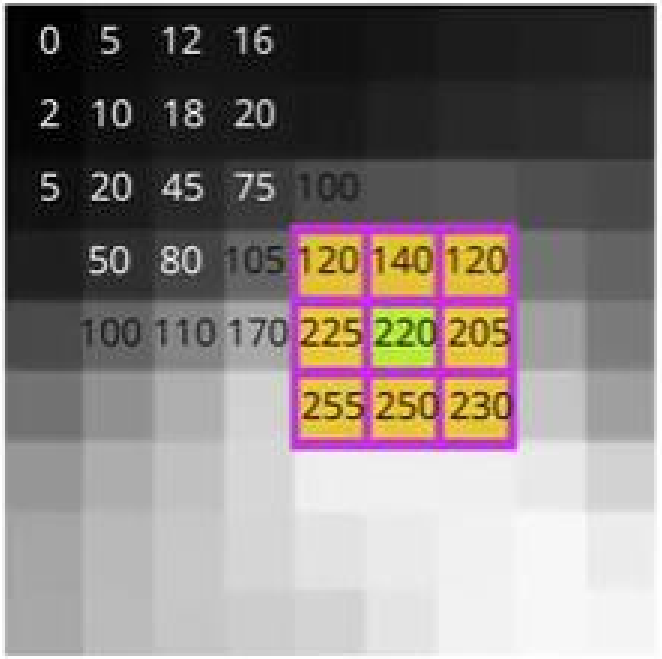
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



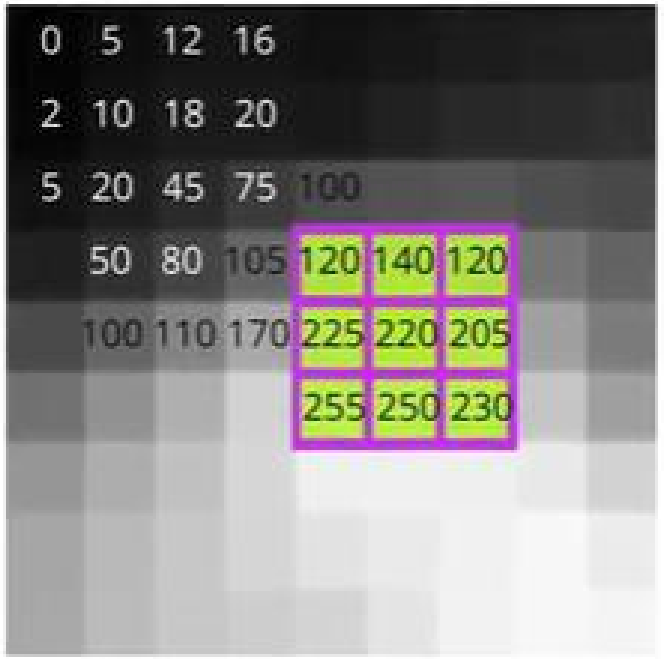
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



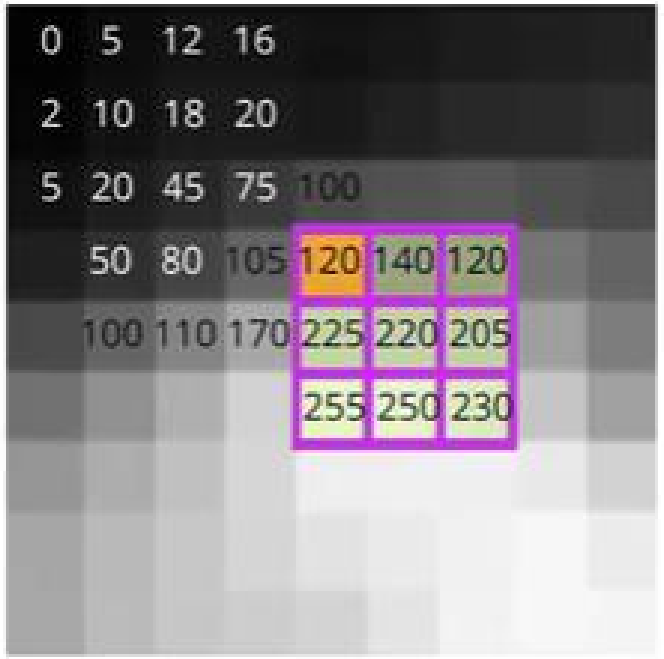
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



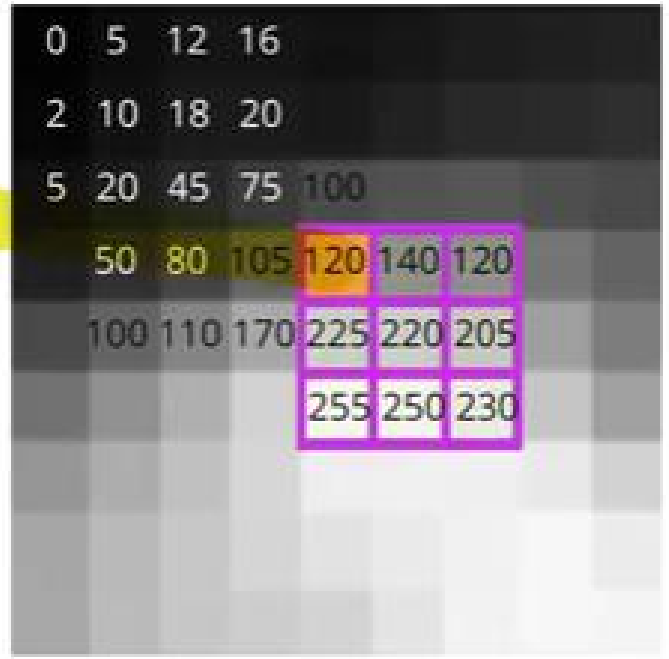
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

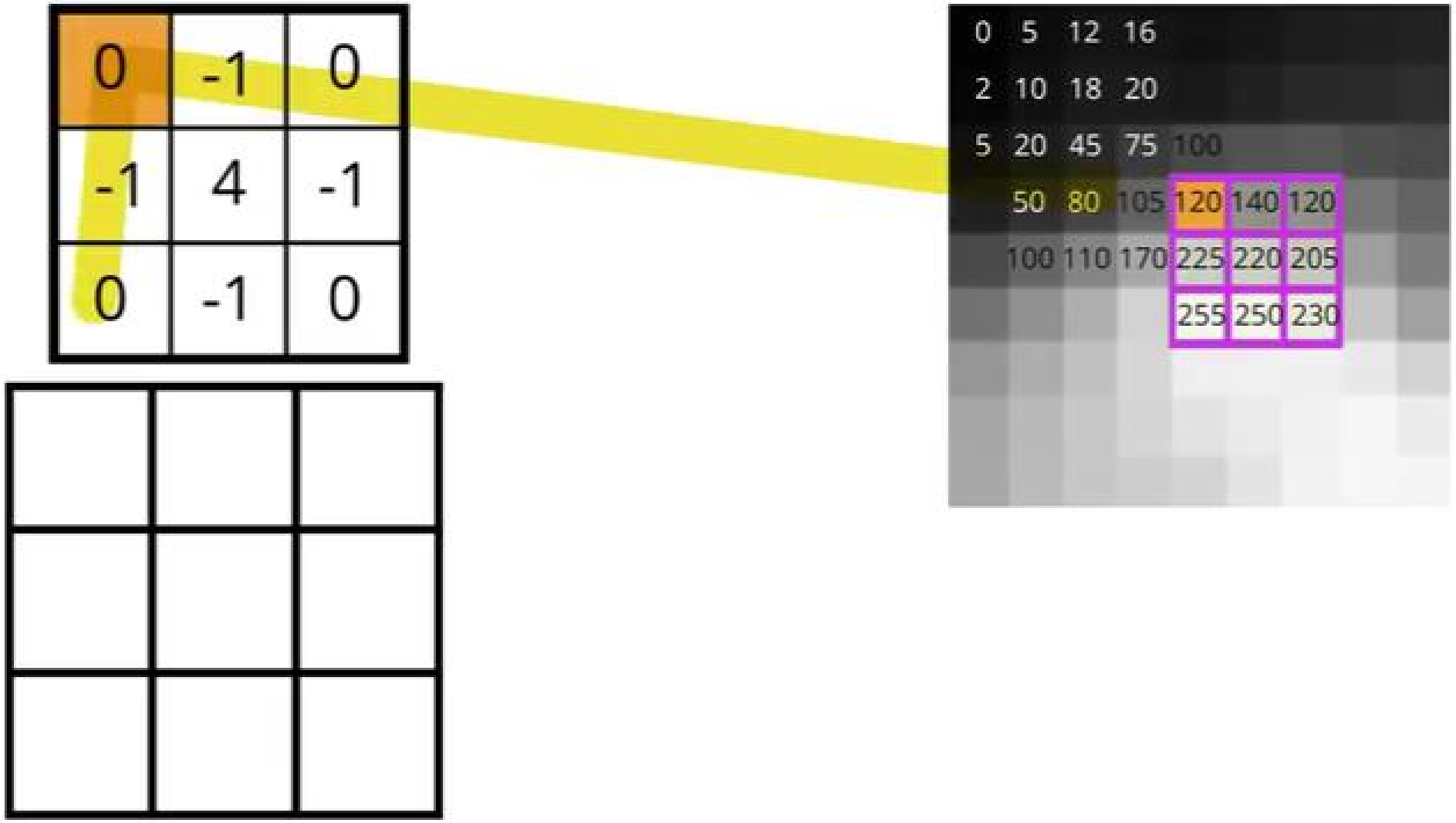


CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

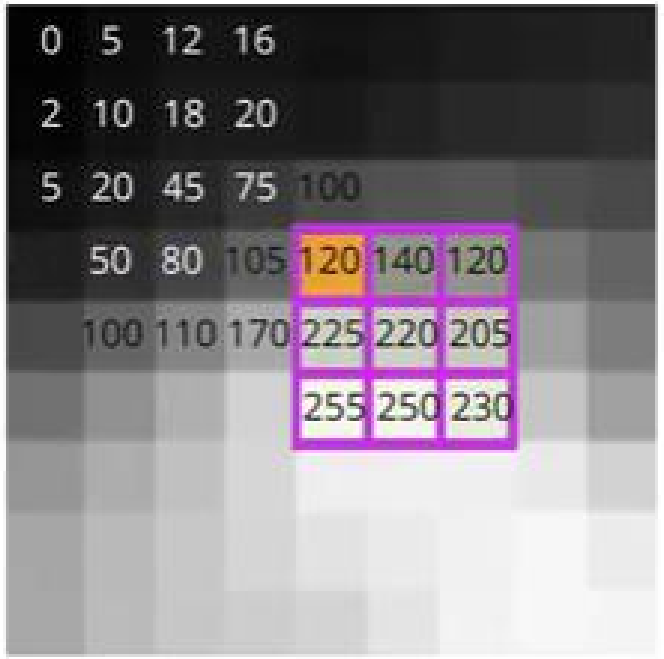


CONVOLUTION



CONVOLUTION

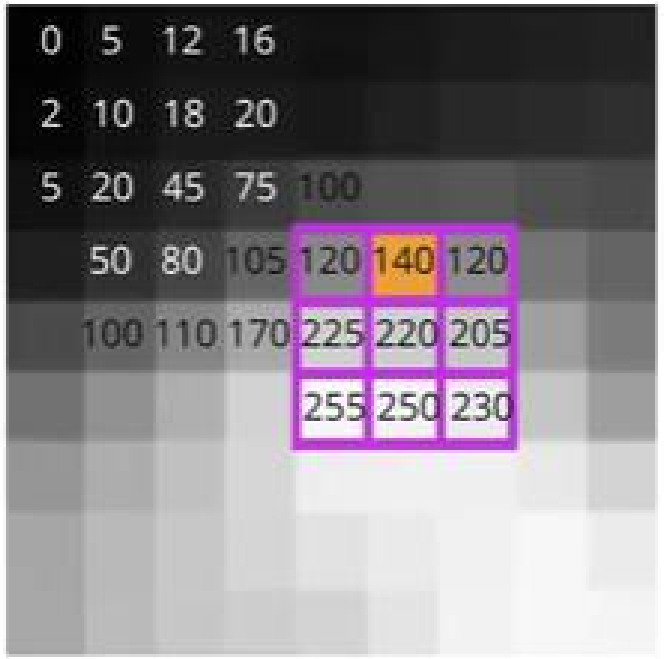
0	-1	0
-1	4	-1
0	-1	0



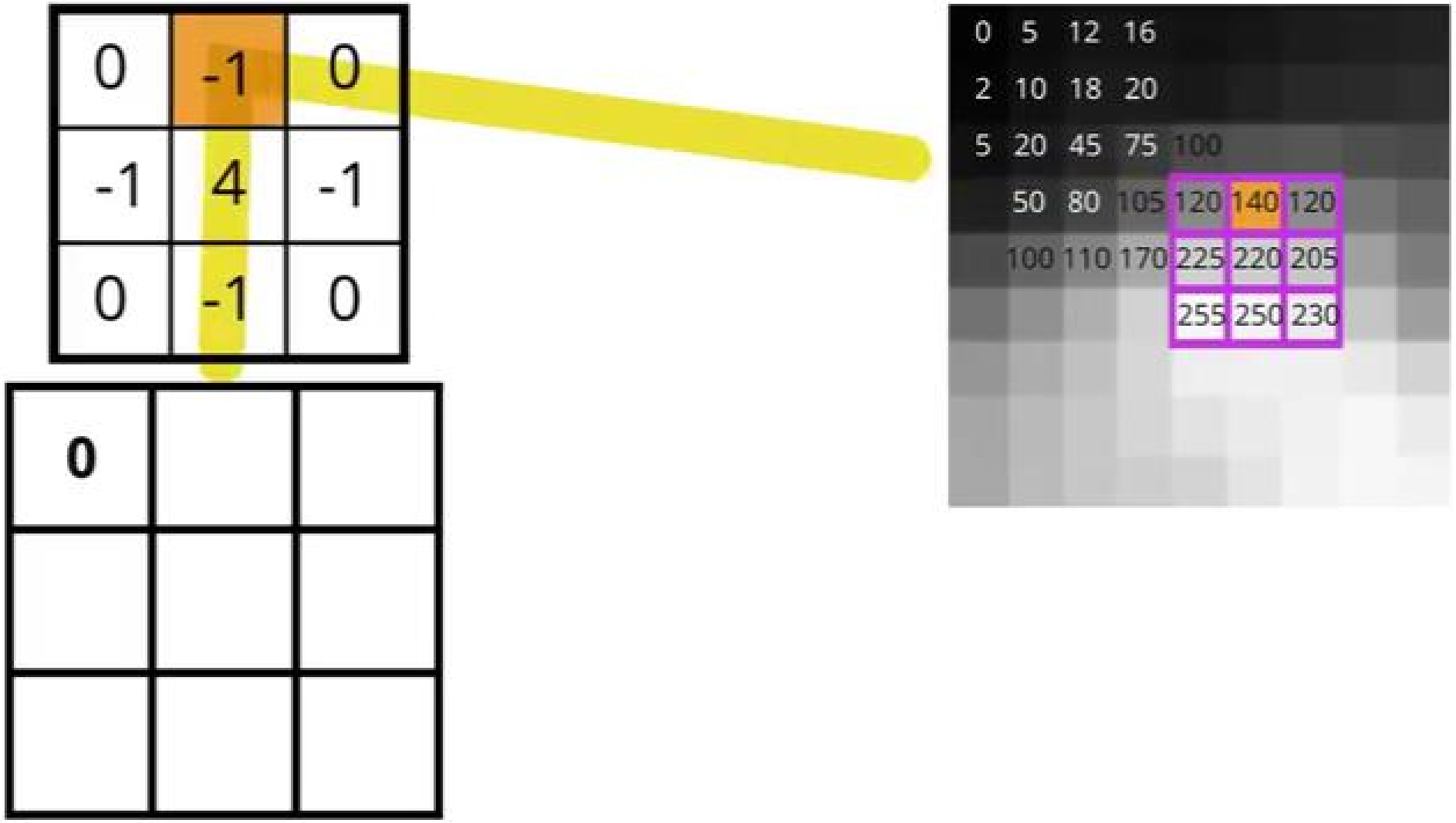
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

0		



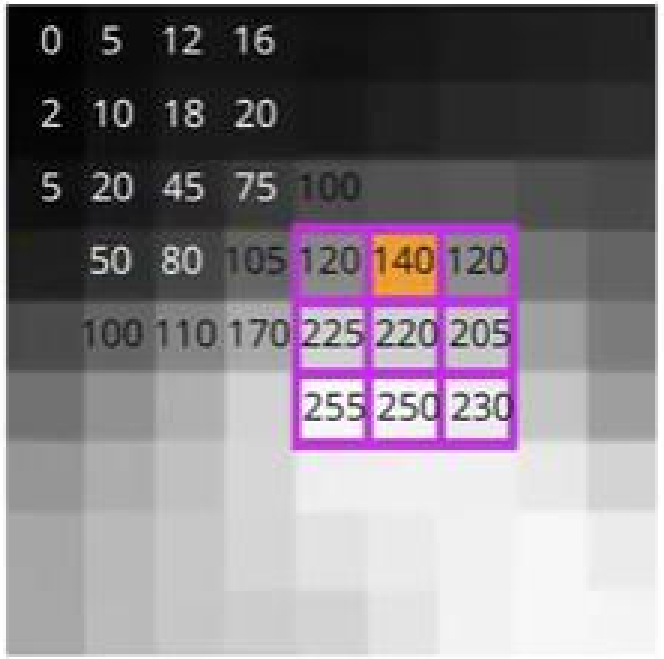
CONVOLUTION



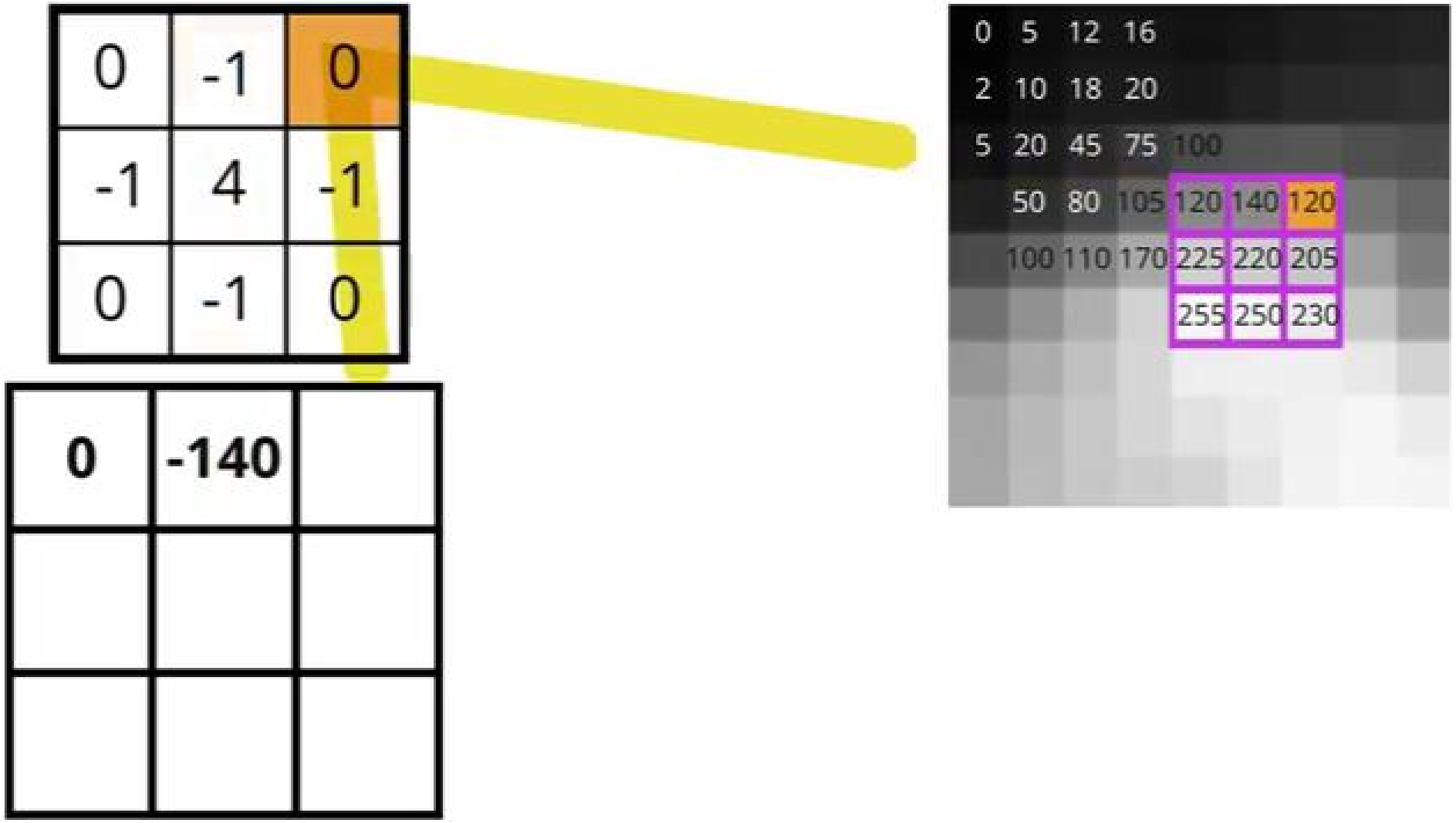
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

0	-140	



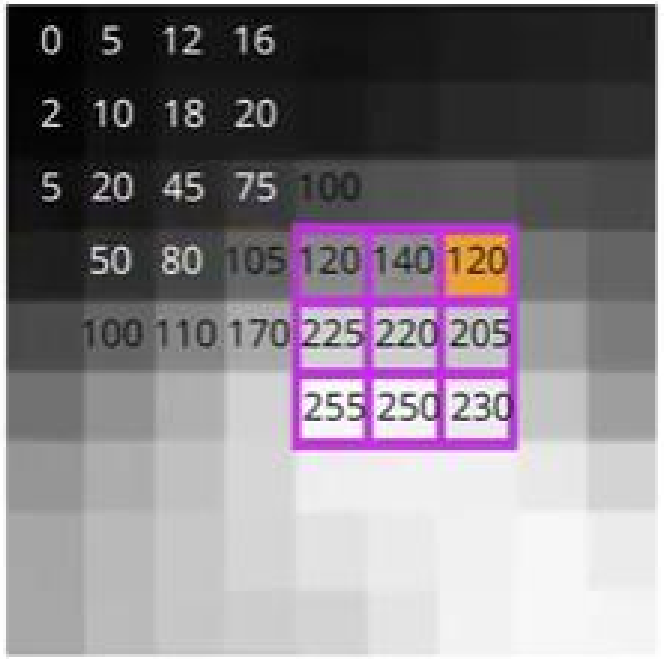
CONVOLUTION



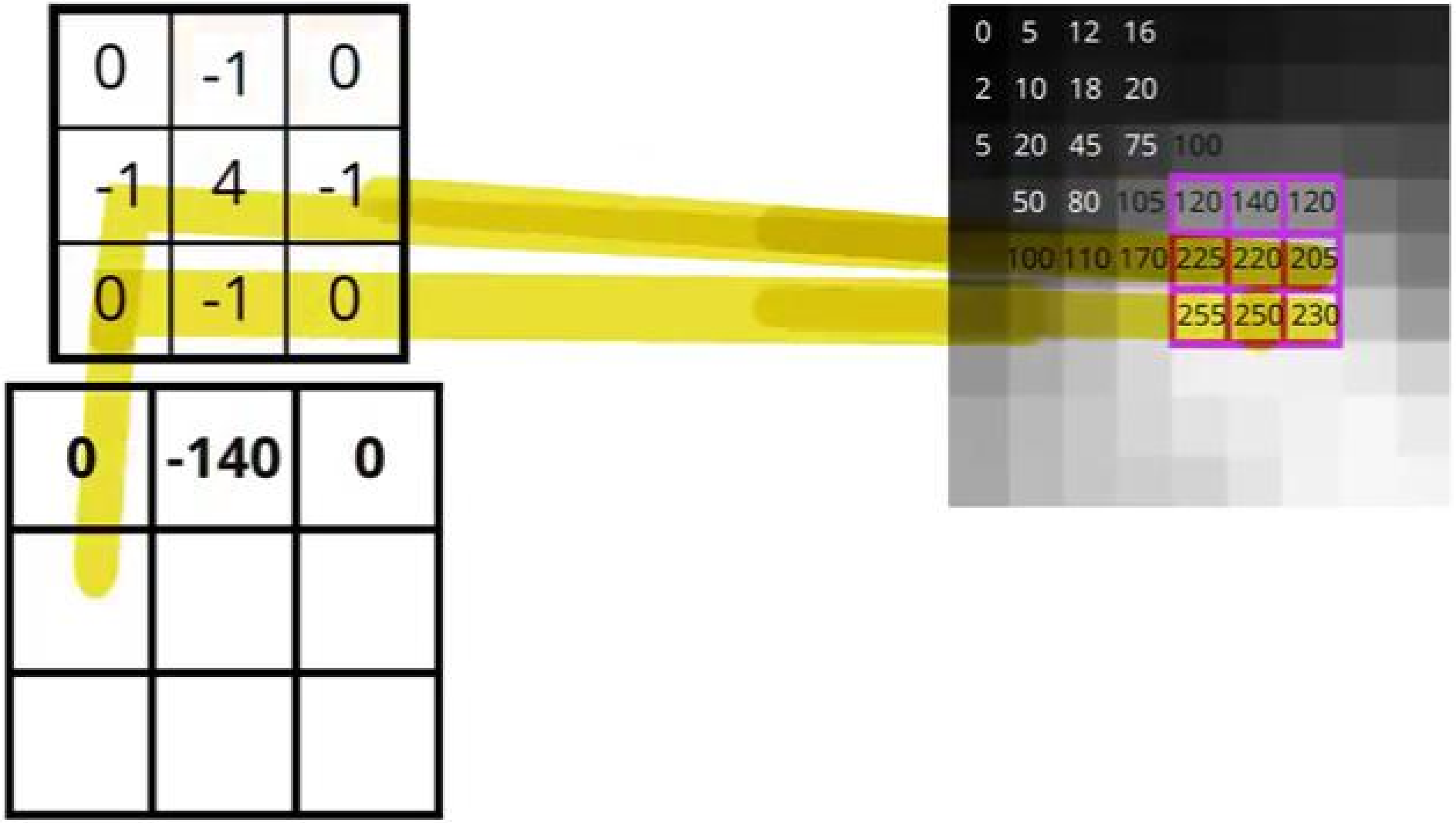
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

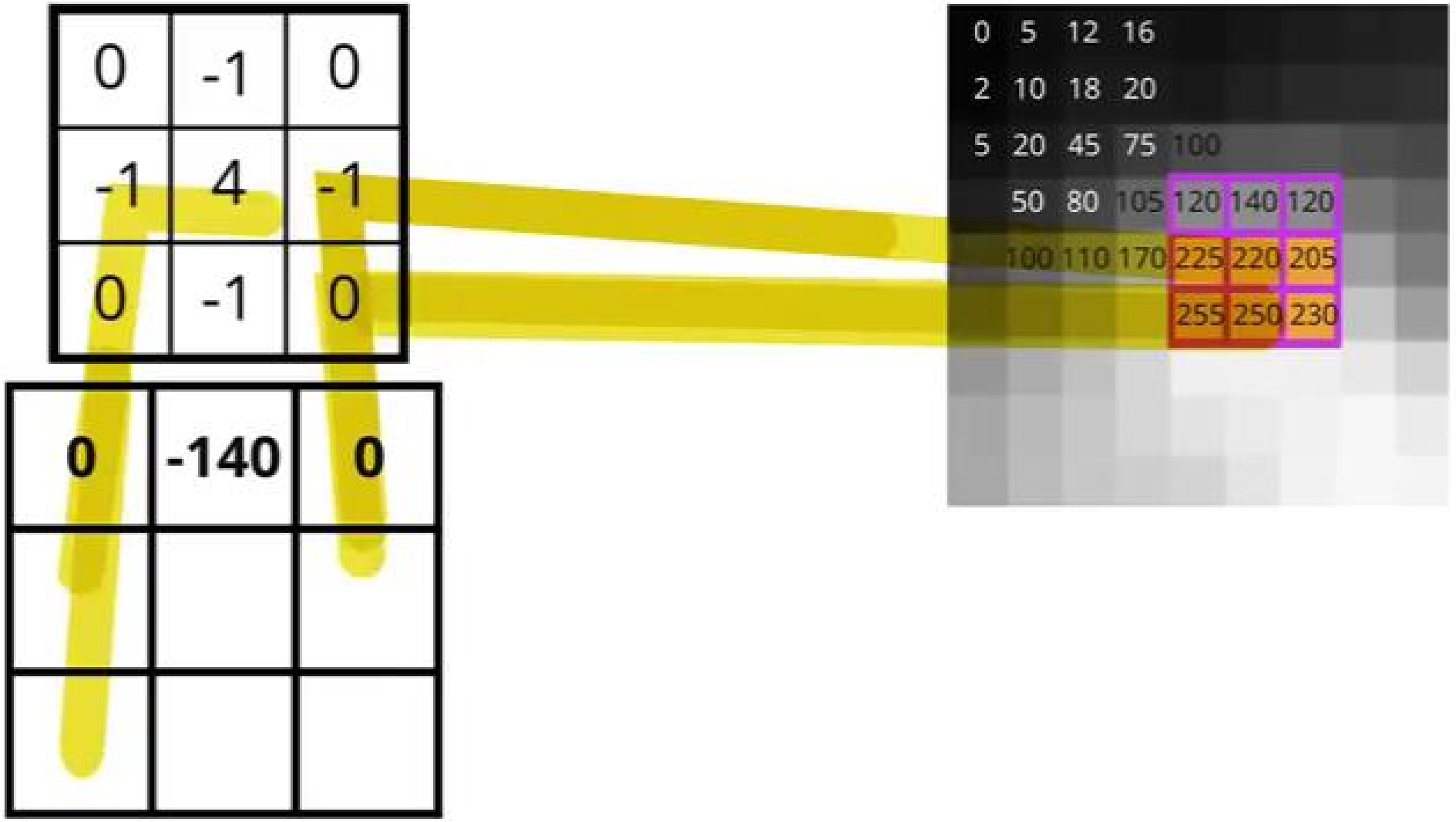
0	**-140**	



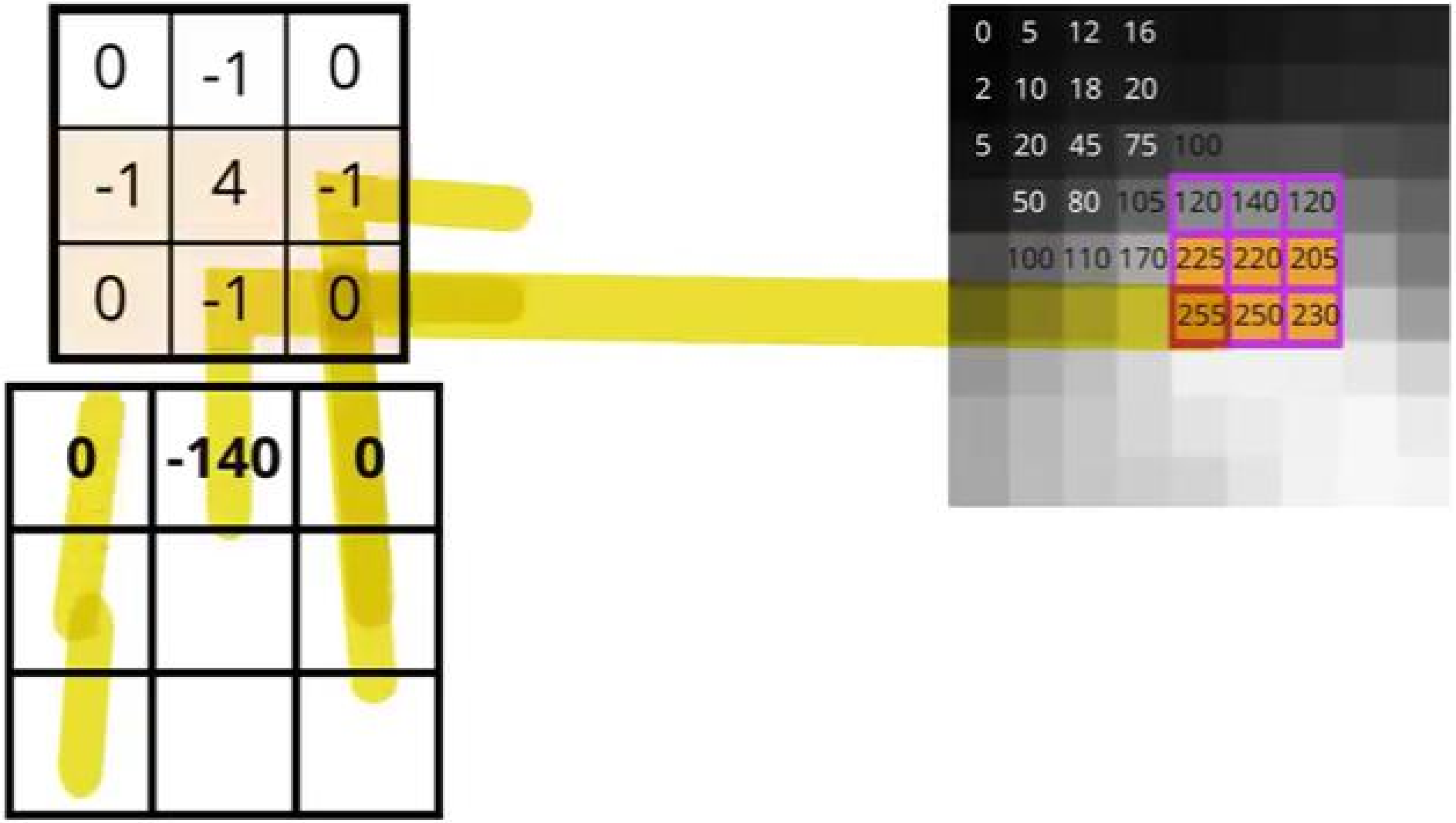
CONVOLUTION



CONVOLUTION



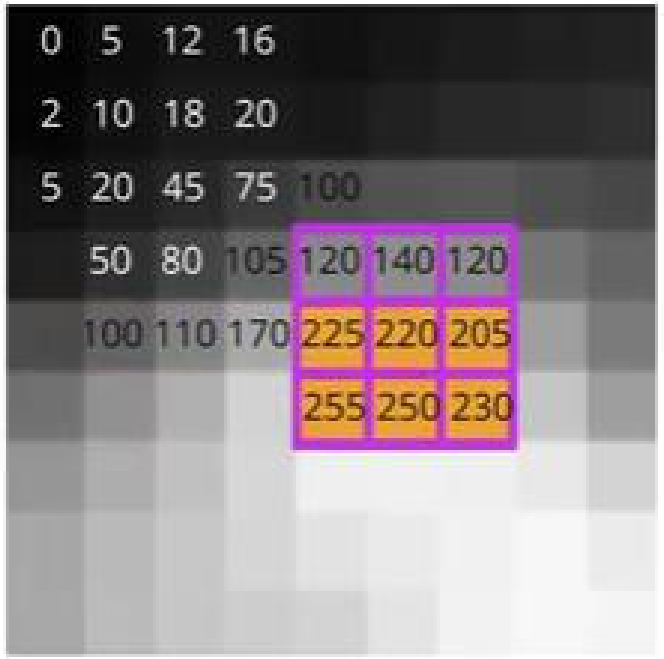
CONVOLUTION



CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

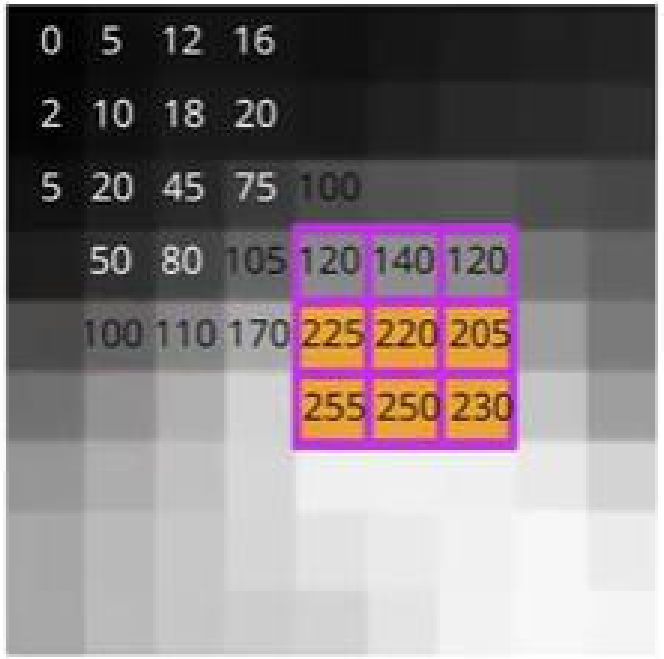
0	**-140**	**0**



CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

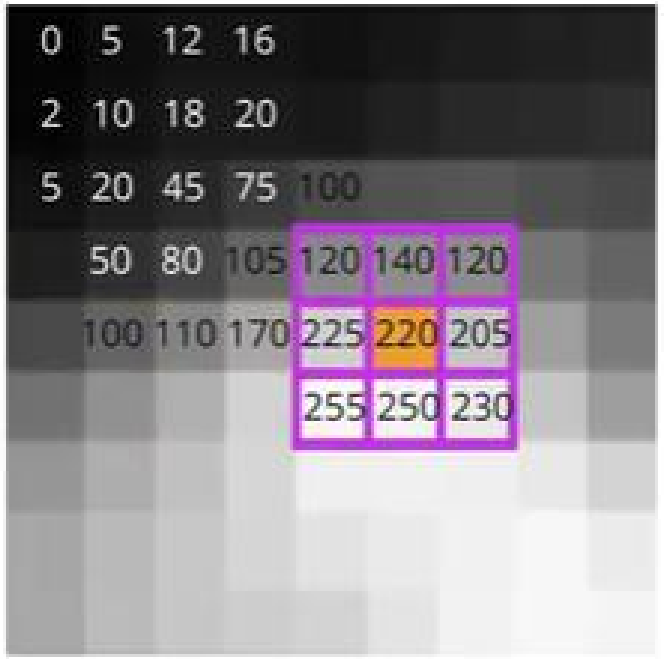
0	**-140**	**0**
-225		-205
0		0



CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

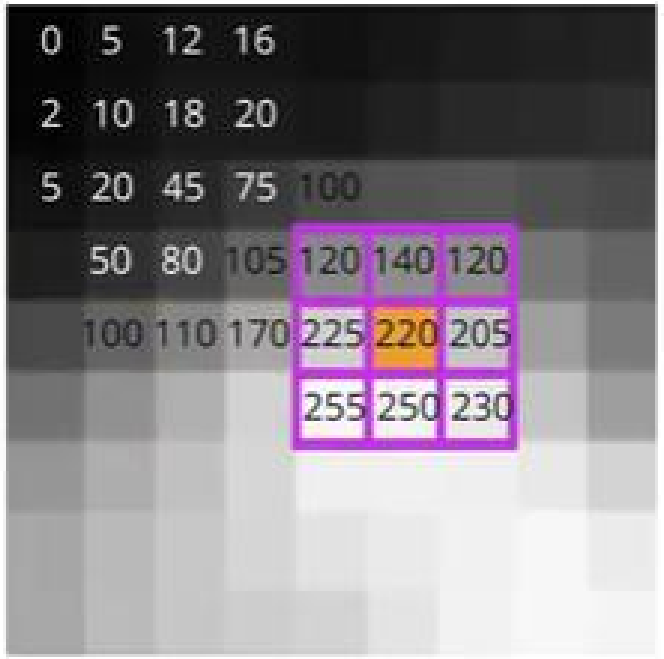
0	**-140**	**0**
-225		**-205**
0	**-250**	**0**



CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225		**-205**
0	**-250**	**0**

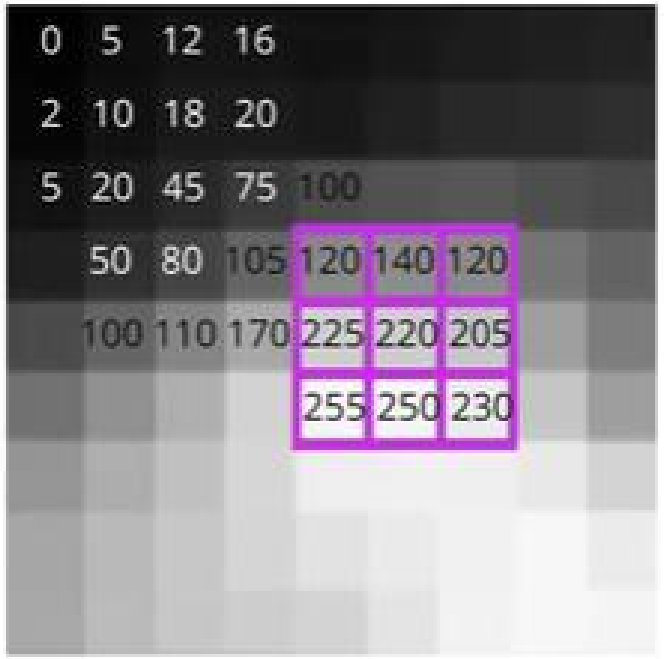


CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**

+ =

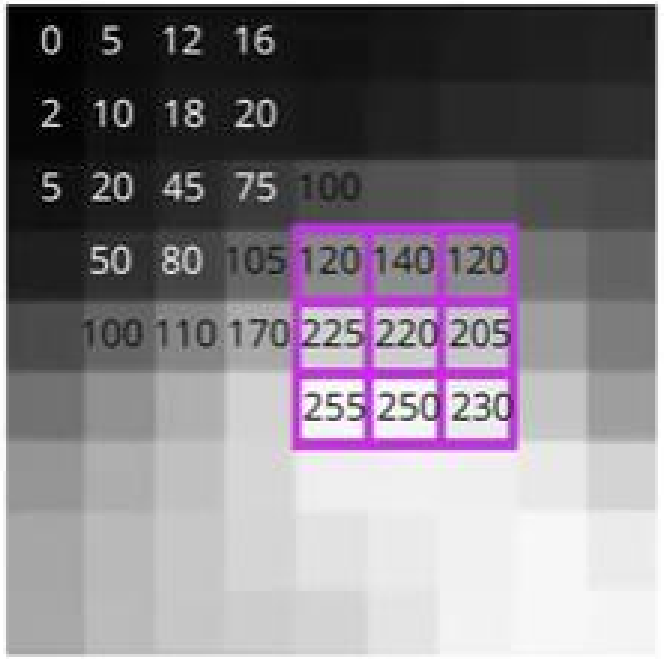


CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**

= *60*

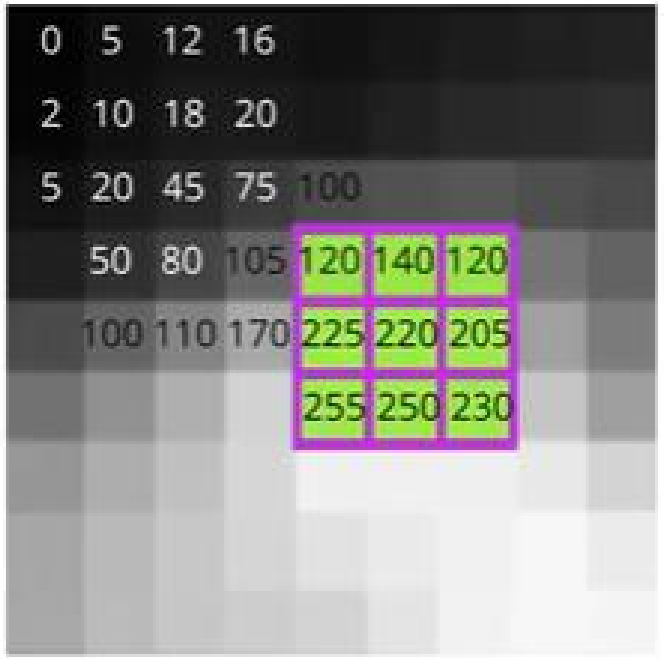


CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**

= *60*

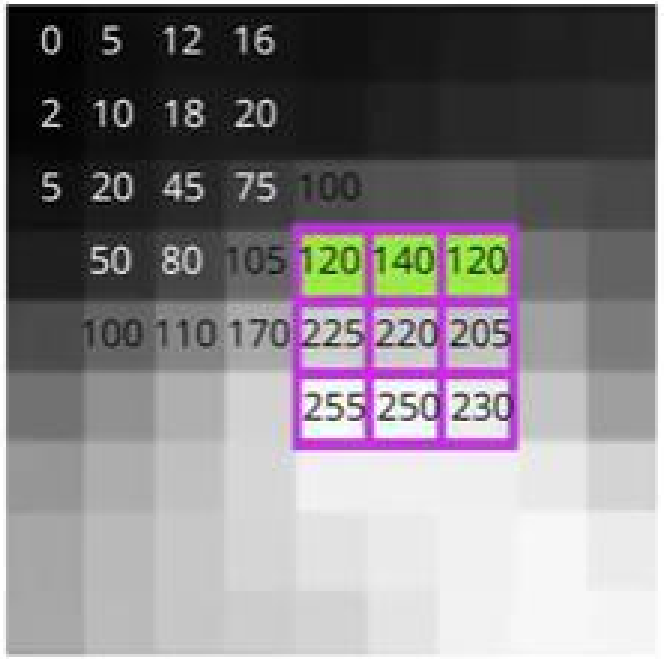


CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**

= *60*



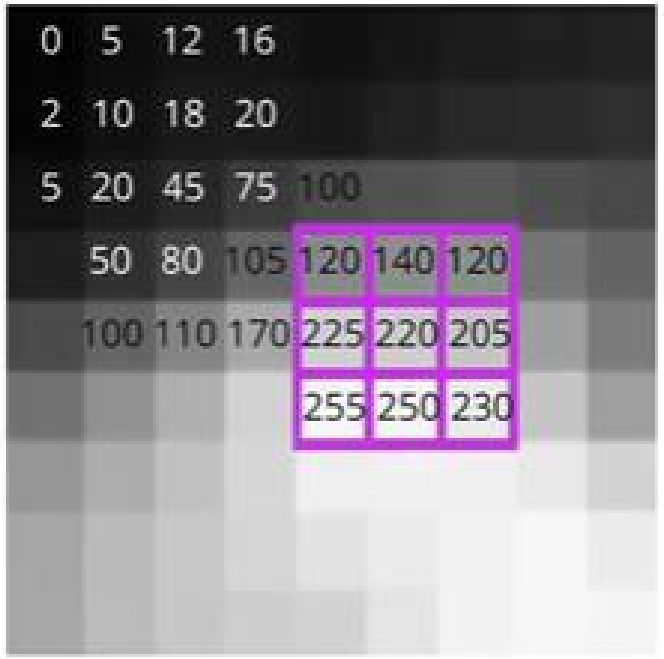
CONVOLUTION

Weights

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**

= *60*



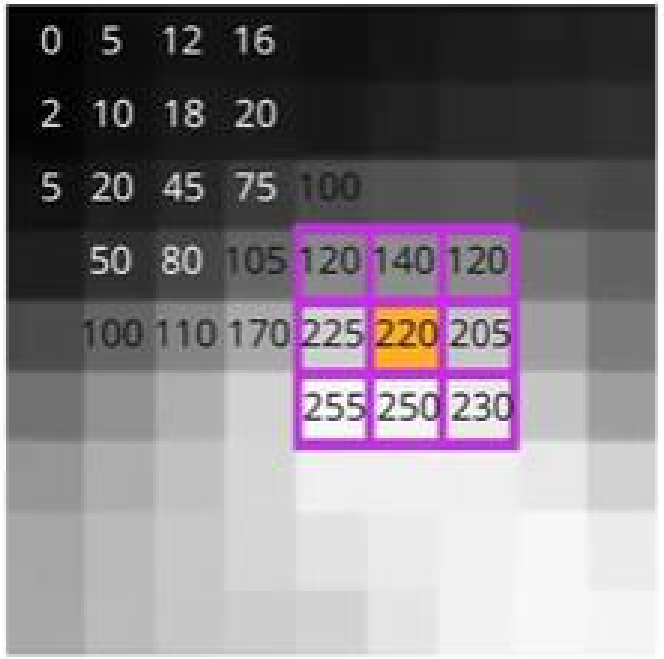
CONVOLUTION

Weights

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**

= *60*



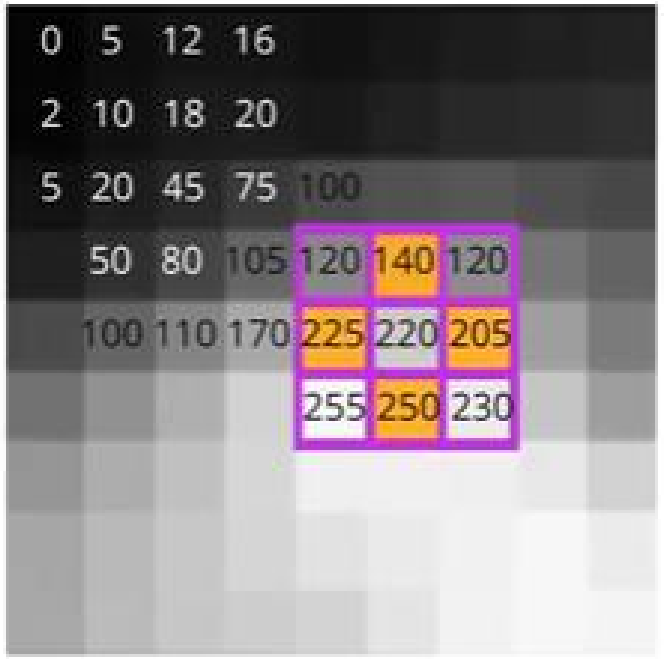
CONVOLUTION

Weights

0	-1	0
-1	4	-1
0	-1	0

0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**

= *60*

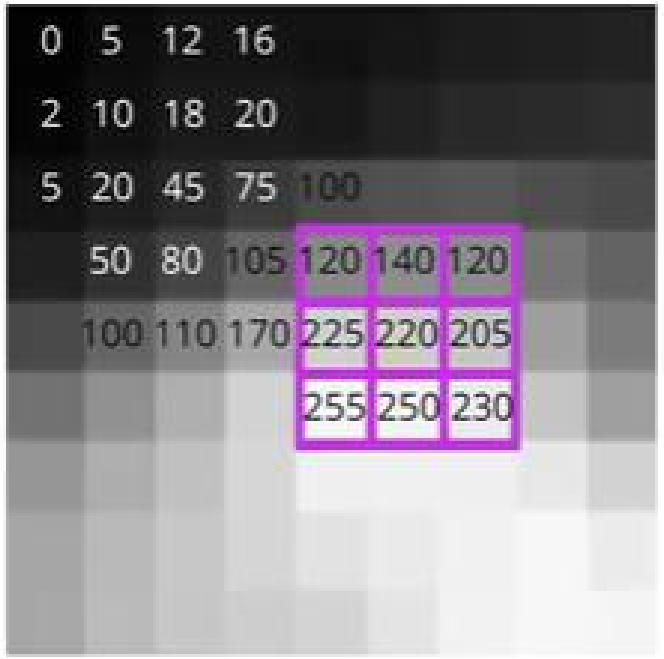


CONVOLUTION

Weights

0	-1	0
-1	4	-1
0	-1	0

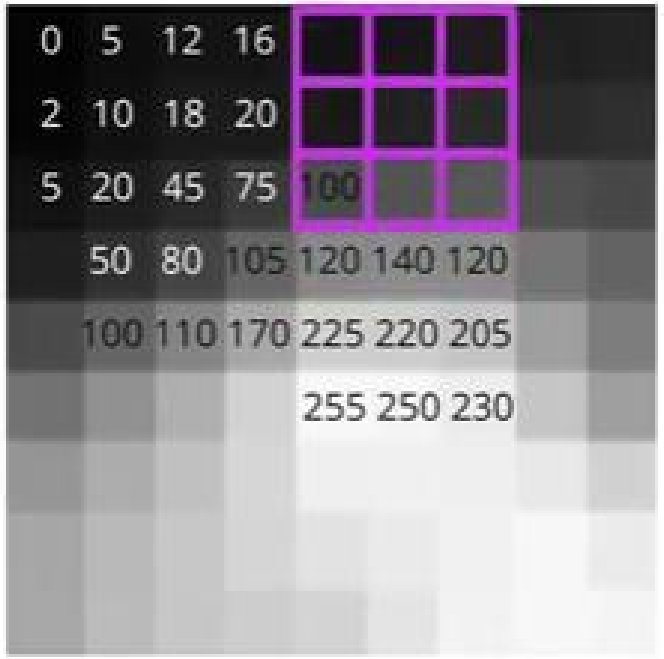
0	**-140**	**0**
-225	**880**	**-205**
0	**-250**	**0**



= ***60*** *Pixel value in the output image*

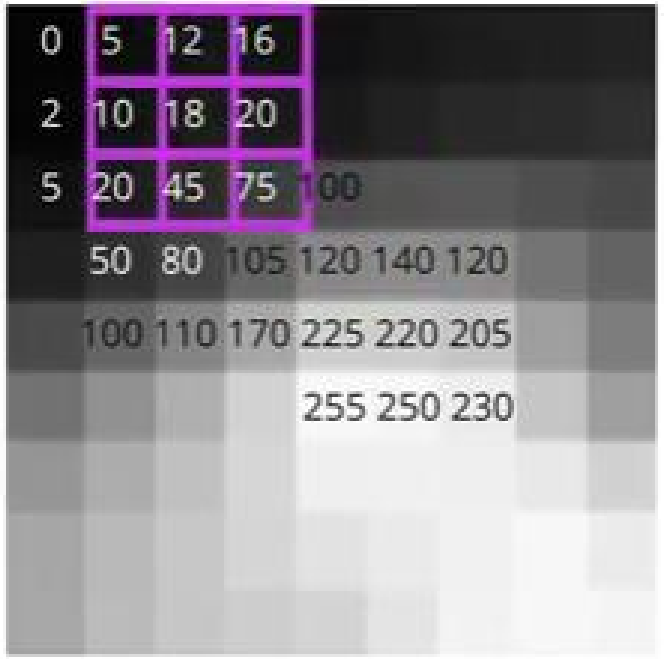
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



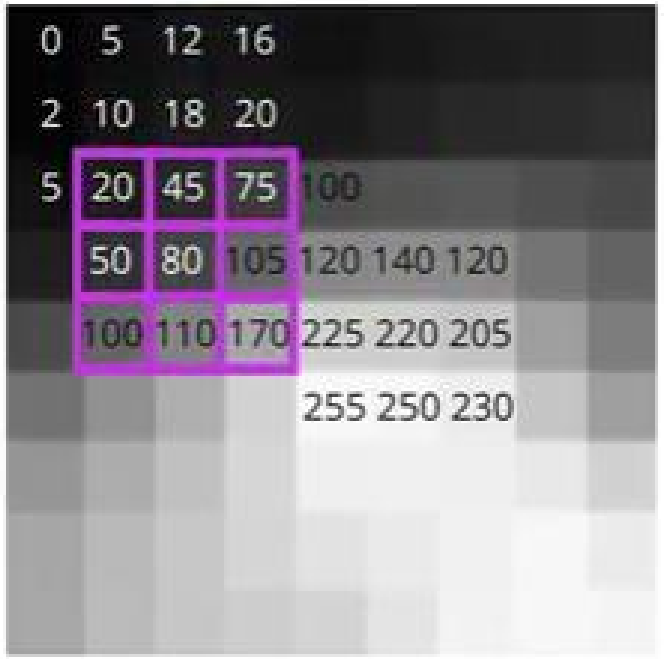
CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0



CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

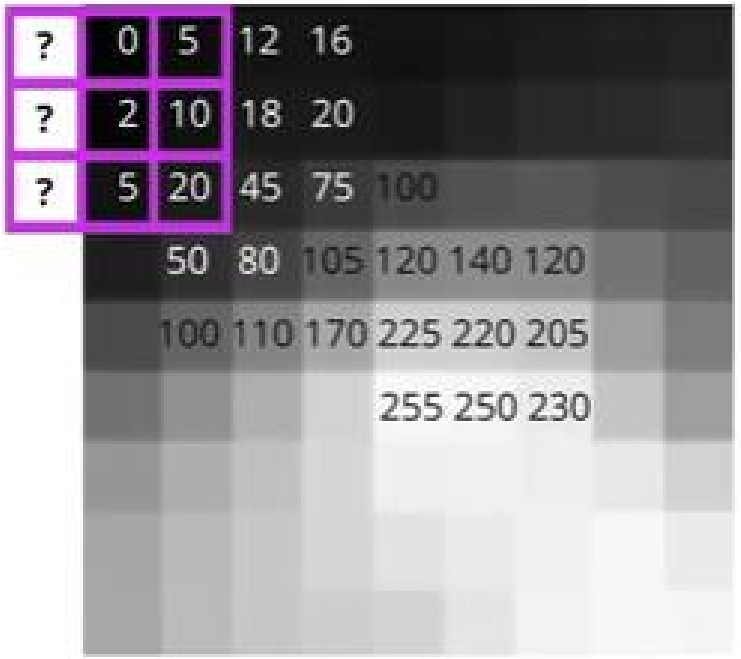


CONVOLUTION

0	-1	0
-1	4	-1
0	-1	0

CONVOLUTION

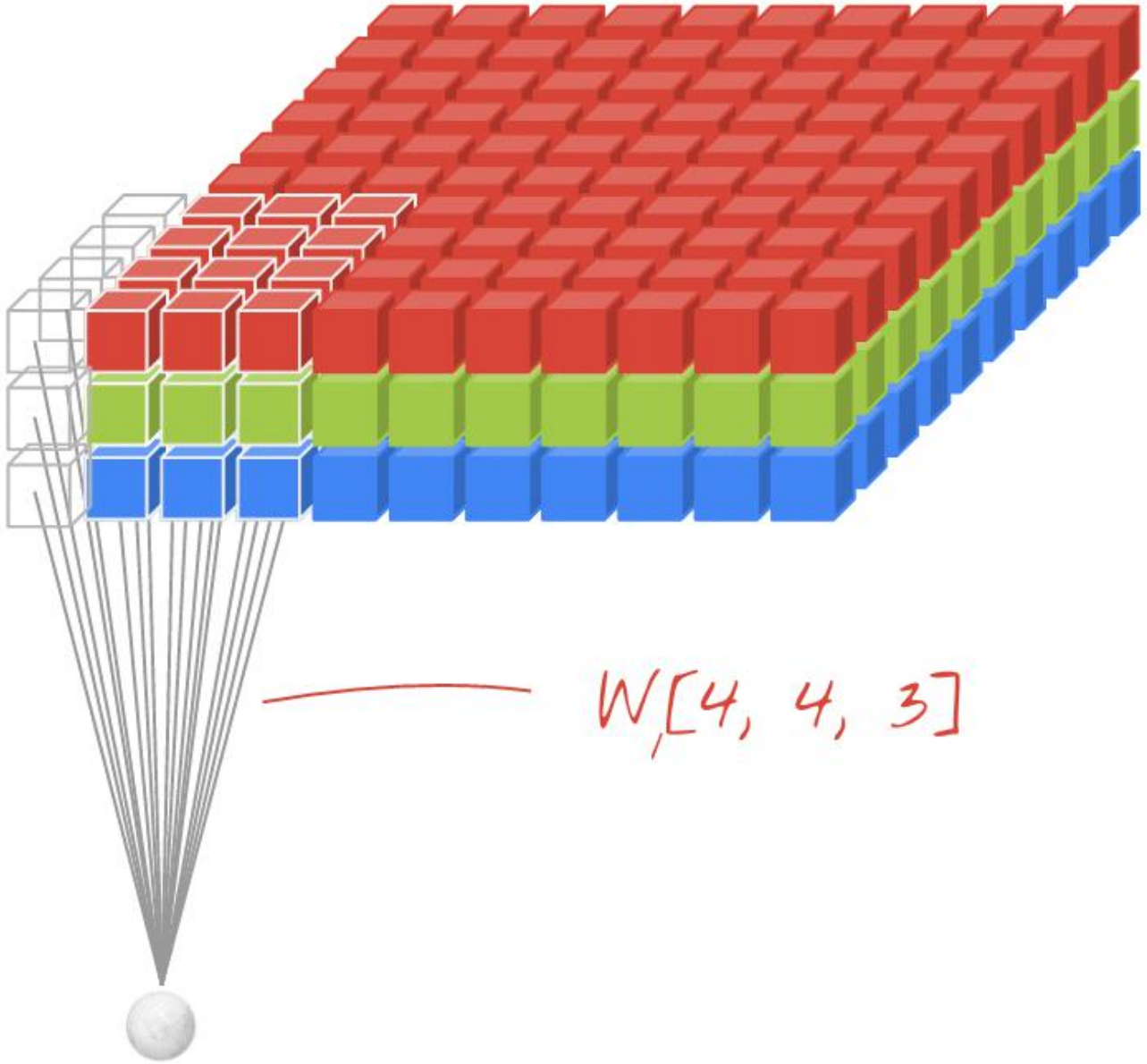
0	-1	0
-1	4	-1
0	-1	0

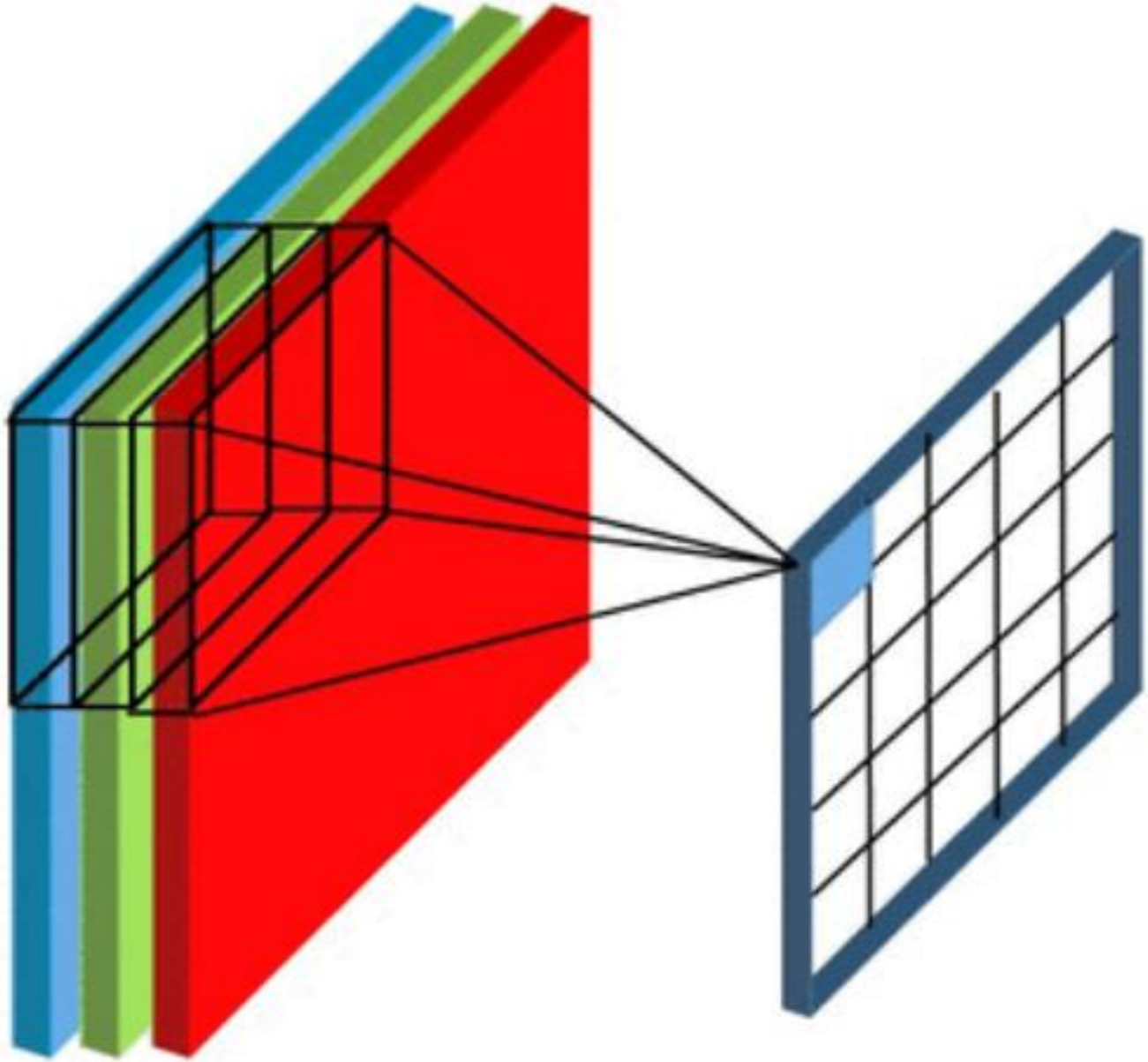


Padding

Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.


W1[4, 4, 3]



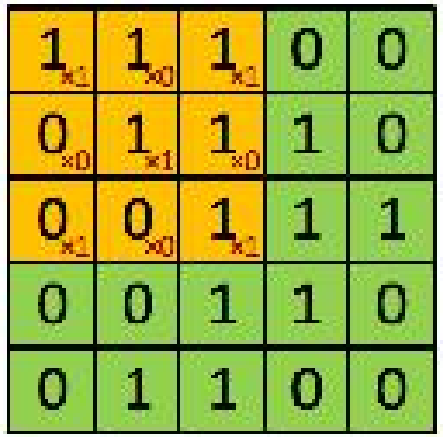
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 x 5 – Image Matrix

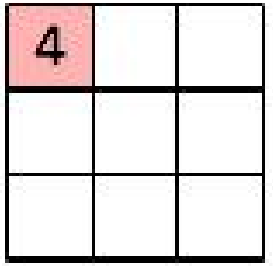
*

1	0	1
0	1	0
1	0	1

3 x 3 – Filter Matrix



Image



Convolved Feature

Strides

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

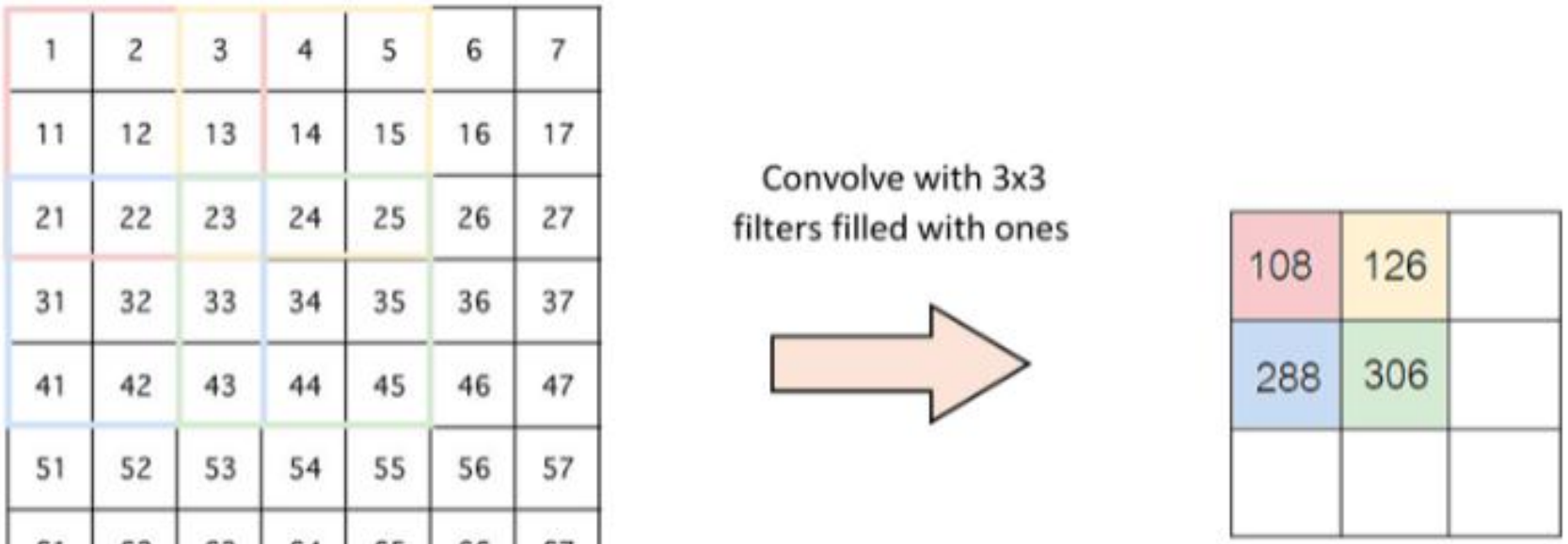
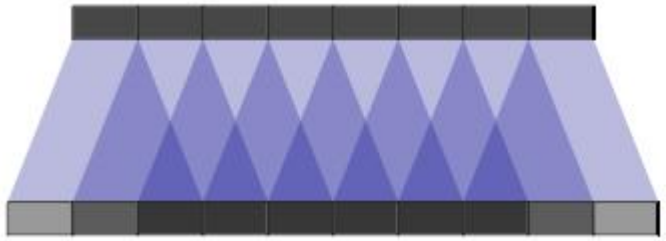
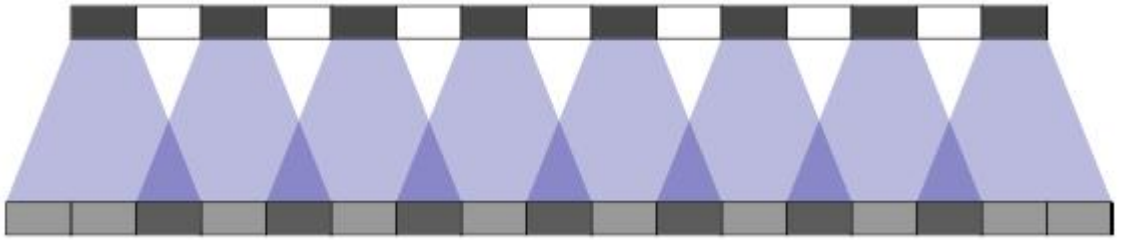


Figure 6 : Stride of 2 pixels



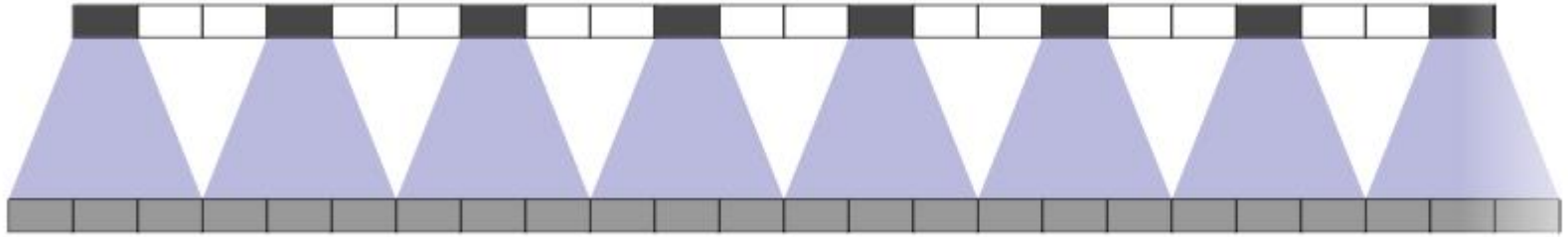
stride = 1

size = 3



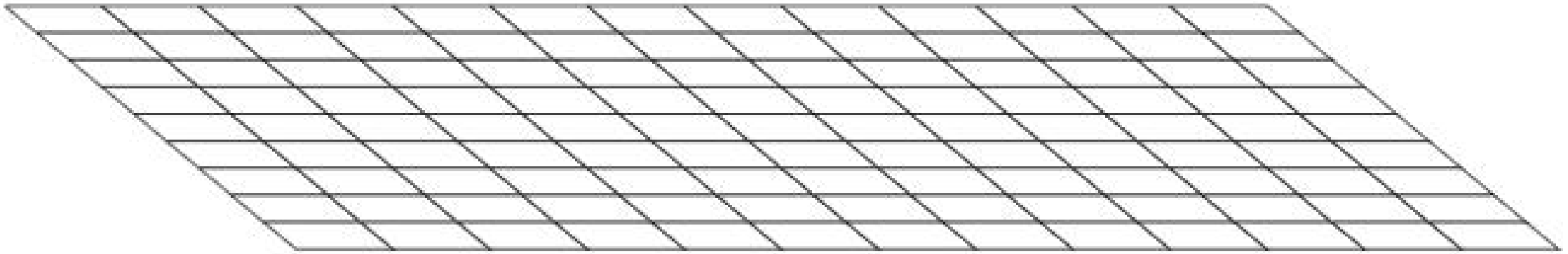
stride = 2

size = 3



stride = 3

size = 3



How to compute the output volume[W2xH2xD2]?

Answer:

- $W2=[(W1-F+2P)/S]+1$
- $H2=[(H1-F+2P)/S]+1$
- $D2=K$

where:

- [W1xH1xD1] : input volume size
- F: receptive field size
- S: stride
- P: amount of zero padding used on the border.
- K: depth

Kernel Size

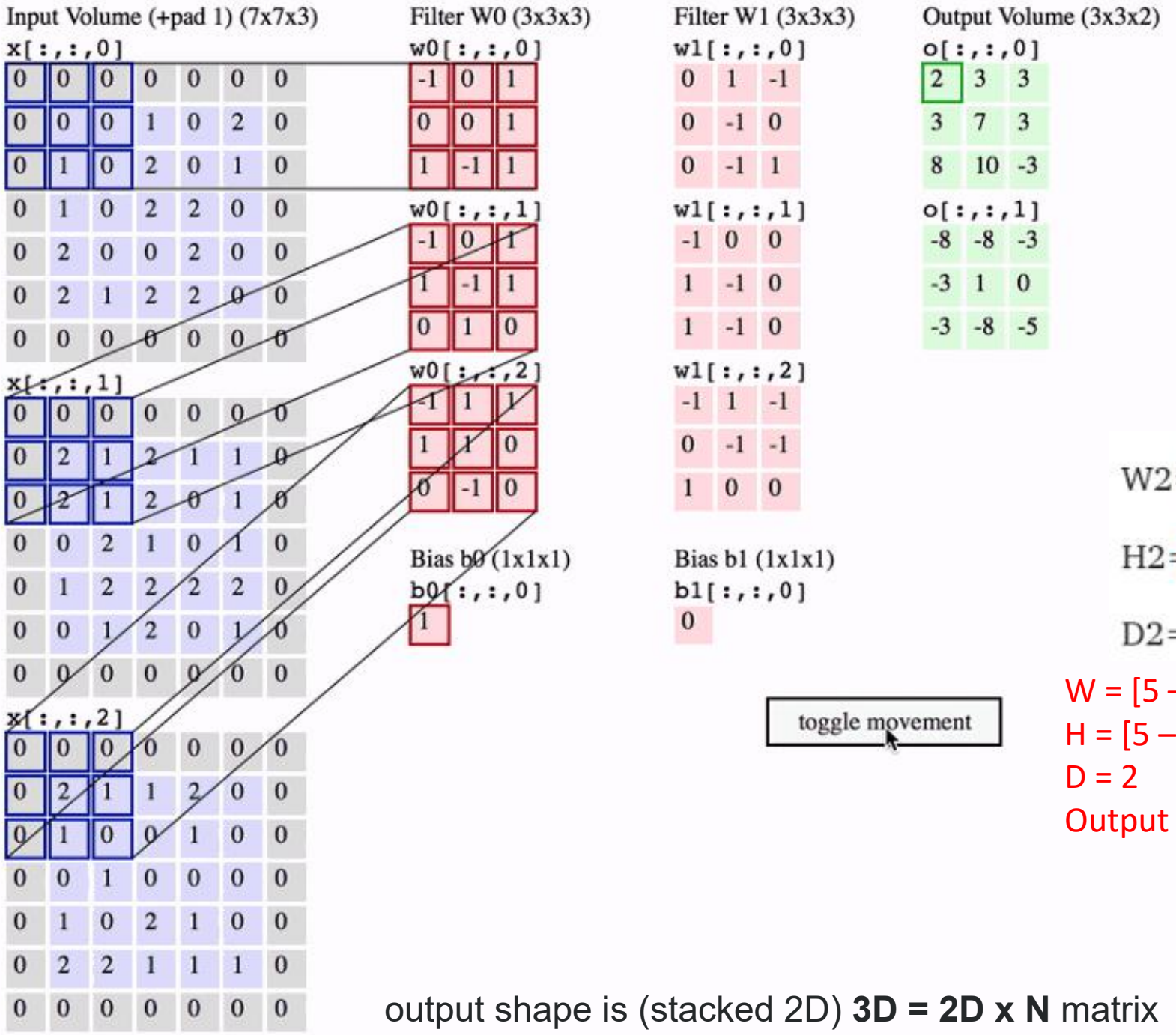
Example:

What is the output volume of the first Convolutional Layer of Krizhevsky et al. architecture that won the ImageNet challenge in 2012?

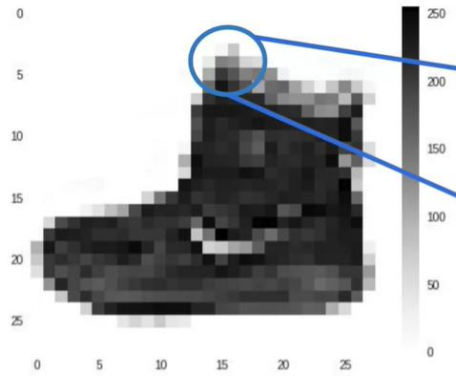
Input size: [227x227x3], W=227, F=11, S=4, P=0, and K=96.

Answer:

- (227 - 11) / 4 + 1 = 55
- The size of the Conv layer output volume is [55x55x96].



output shape is (stacked 2D) **3D = 2D x N** matrix



0	64	128
48	192	144
142	226	168

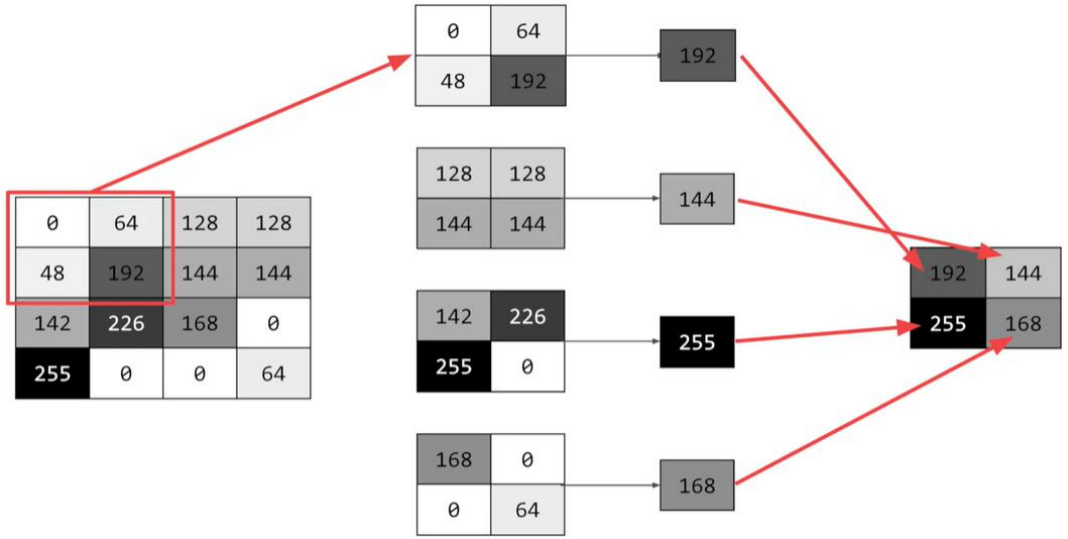
Current Pixel Value is 192

Consider neighbor Values

-1	0	-2
.5	4.5	-1.5
1.5	2	-3

Filter Definition

```
CURRENT_PIXEL_VALUE = 192
NEW_PIXEL_VALUE = (-1 * 0) + (0 * 64) + (-2 * 128) +
                  (.5 * 48) + (4.5 * 192) + (-1.5 * 144) +
                  (1.5 * 42) + (2 * 226) + (-3 * 168)
```



Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = max(0,x)$.

Why ReLU is important : ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

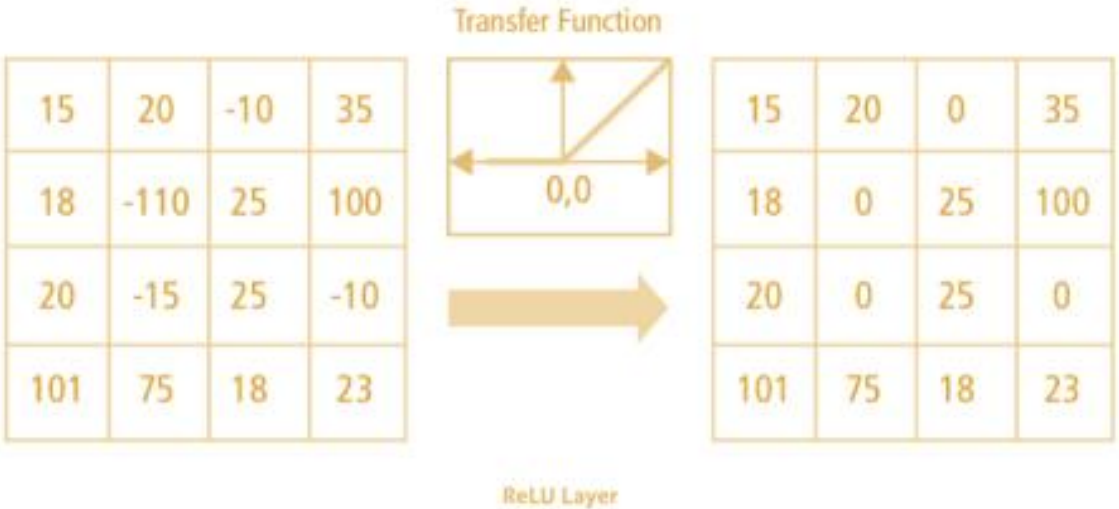
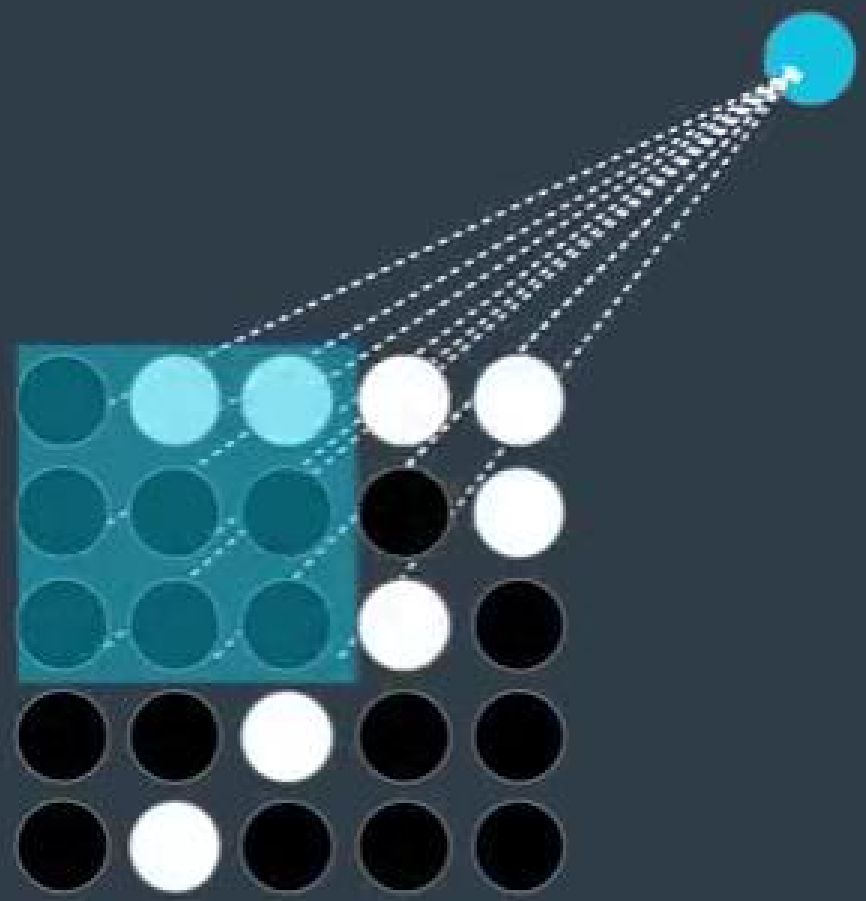


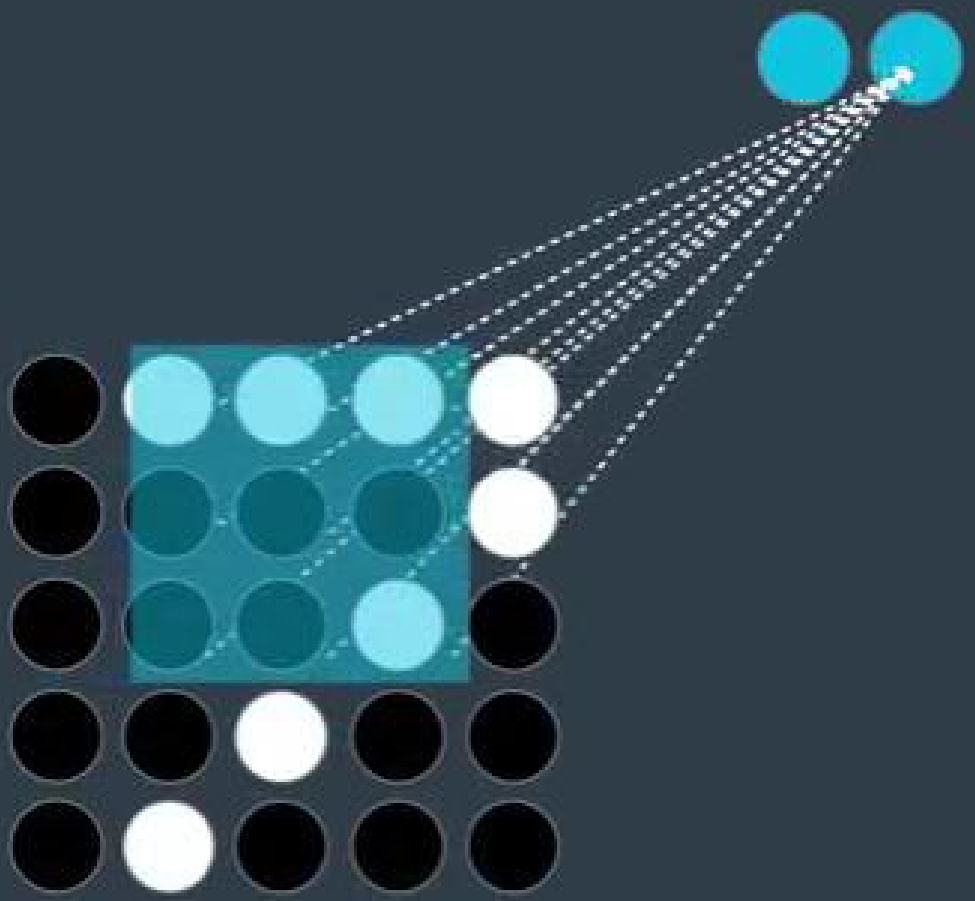
Figure 7 : ReLU operation

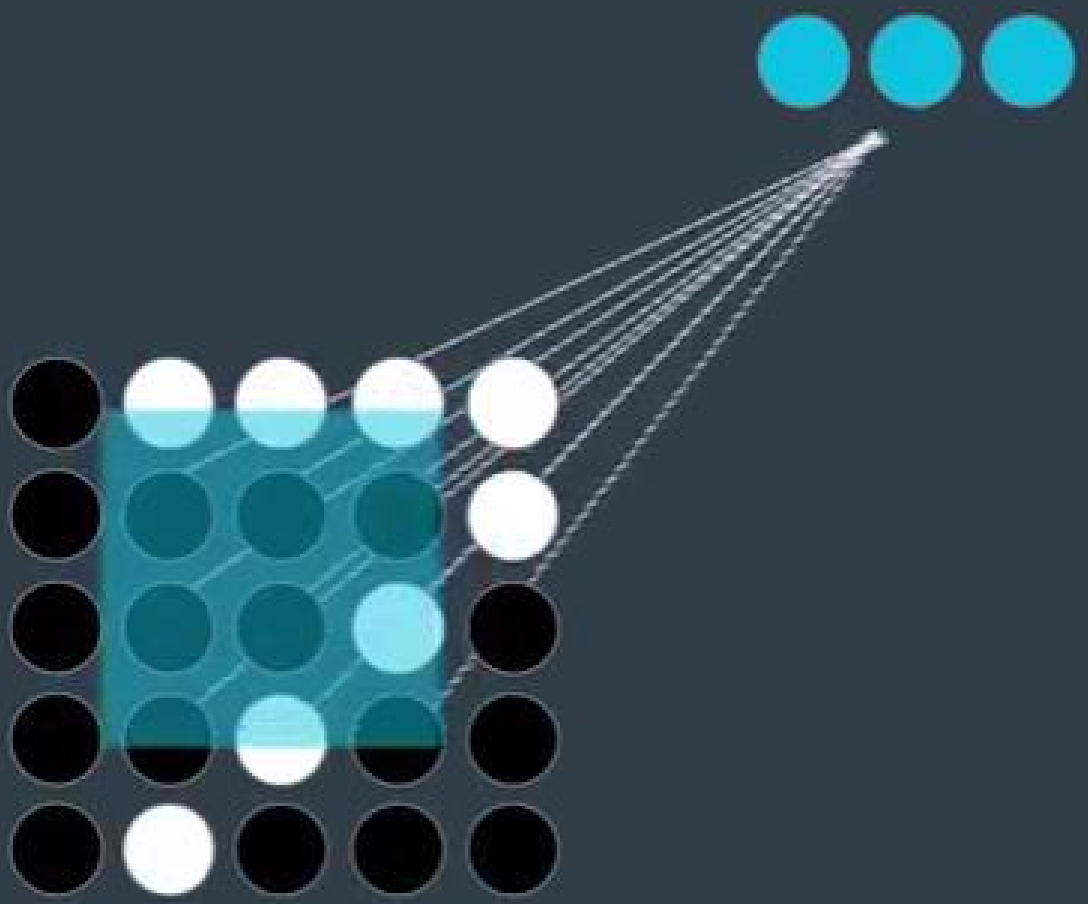
ReLU Effect

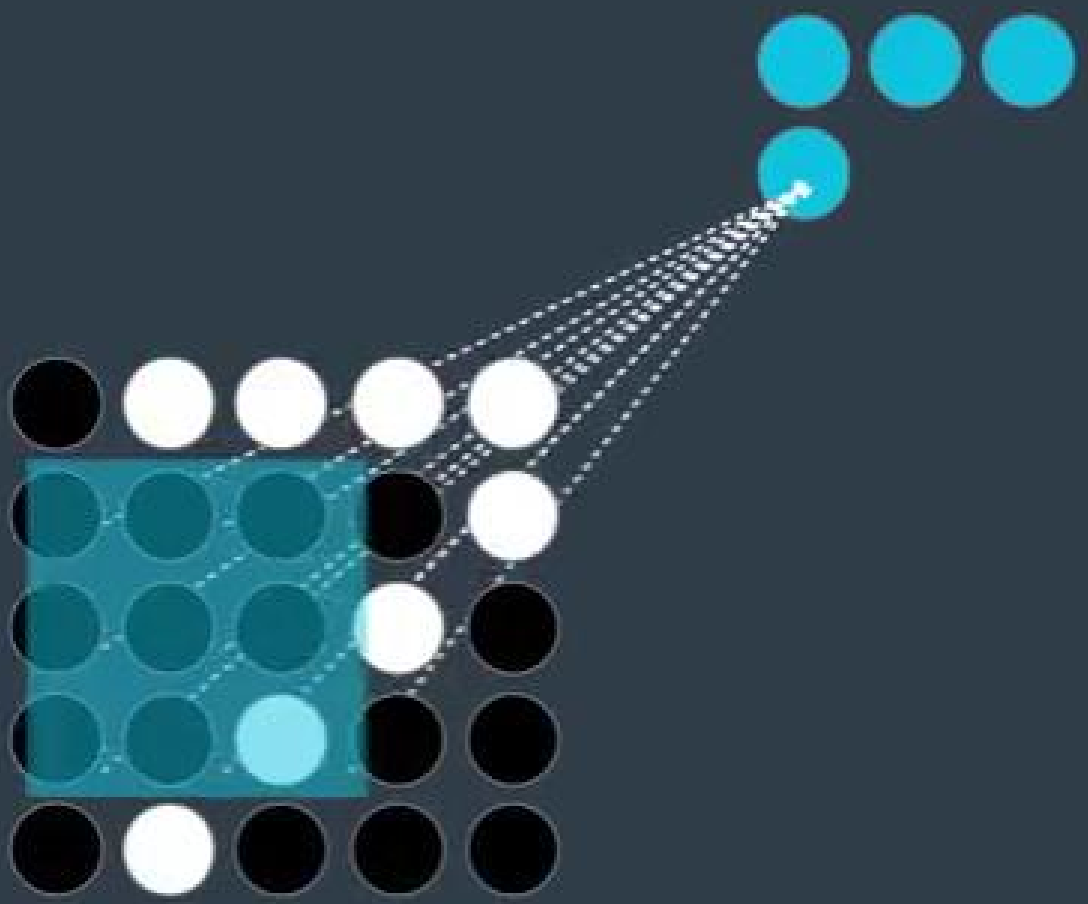
Kernel Effect

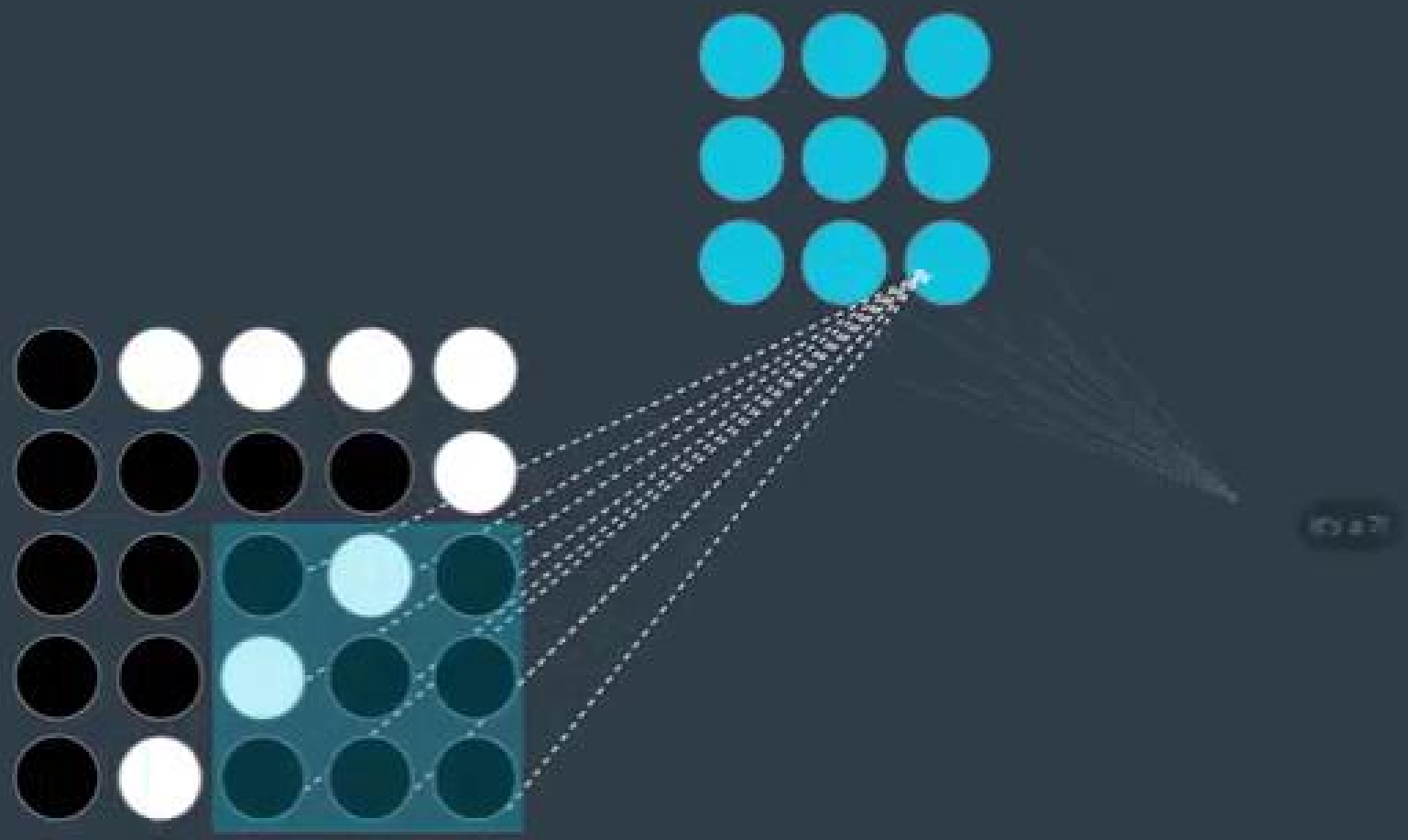
ReLU Effect

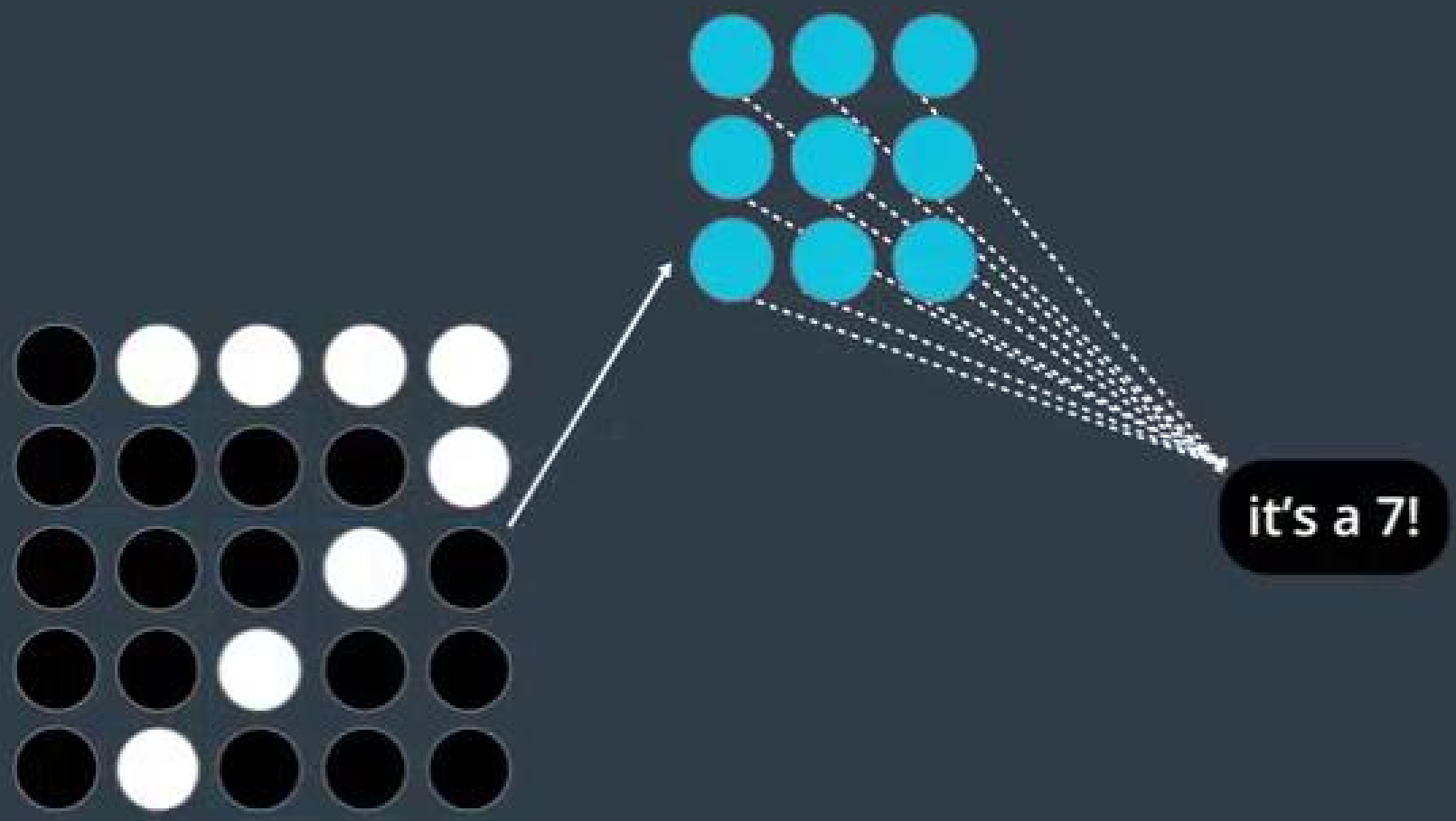


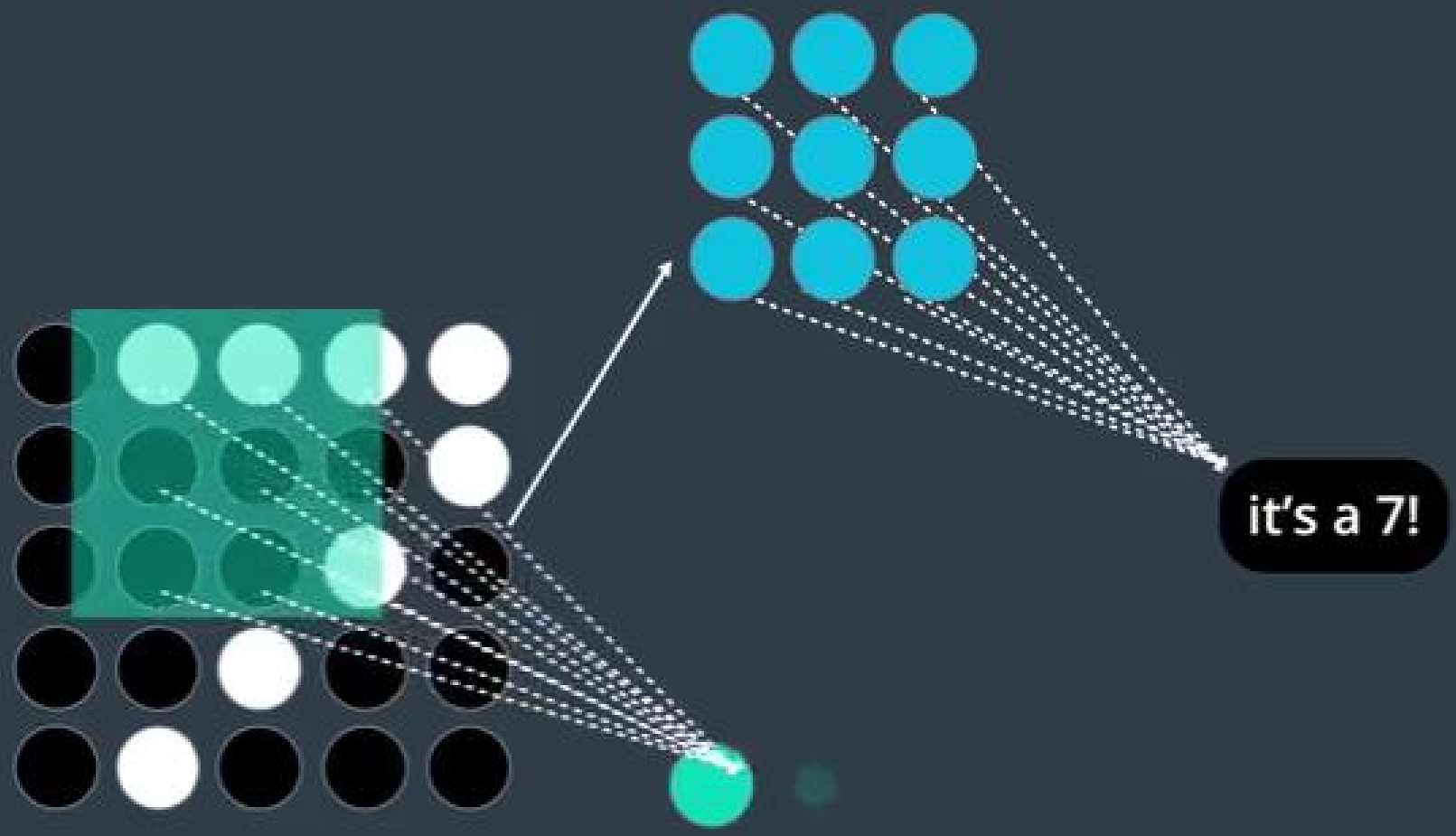


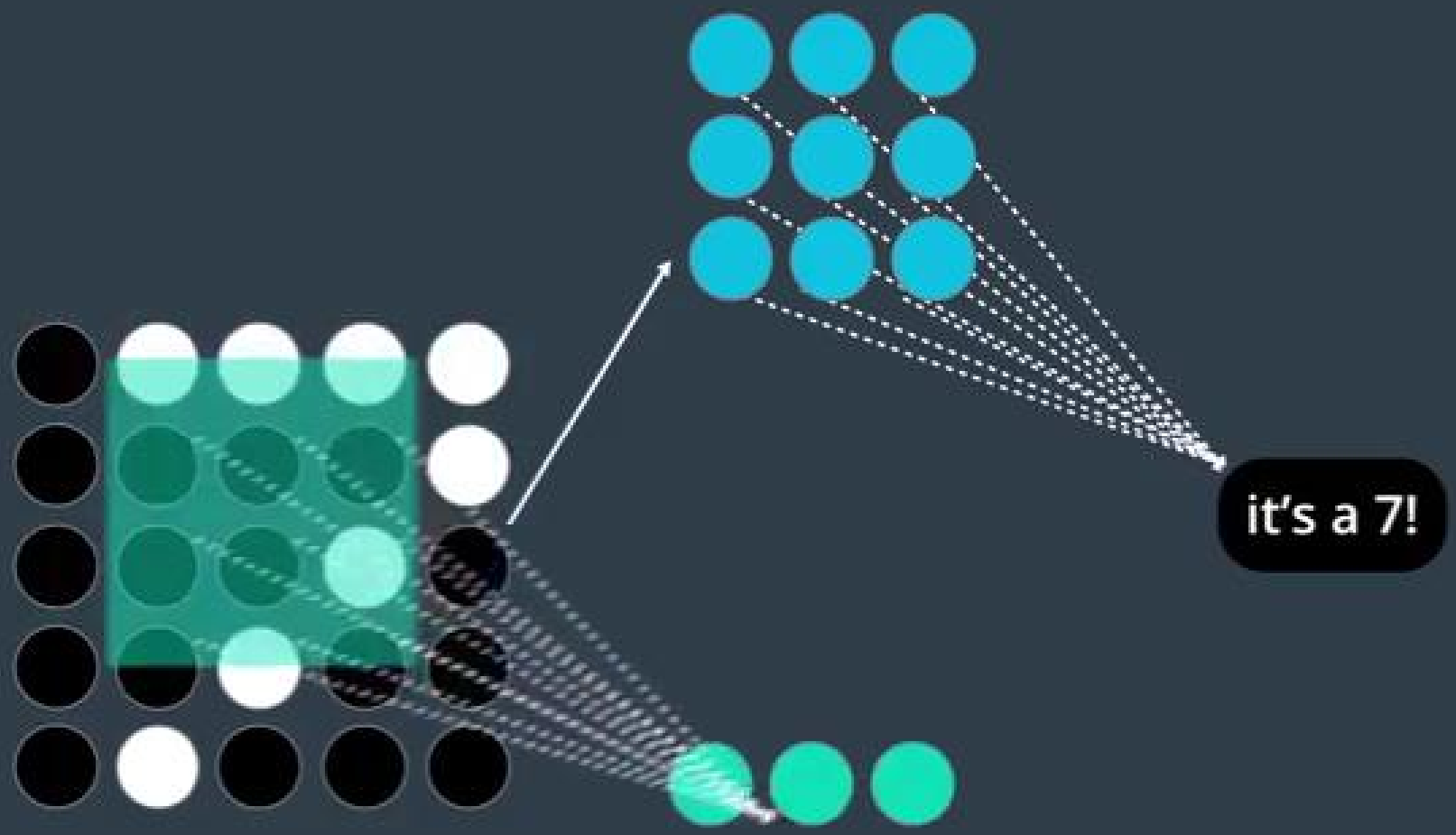


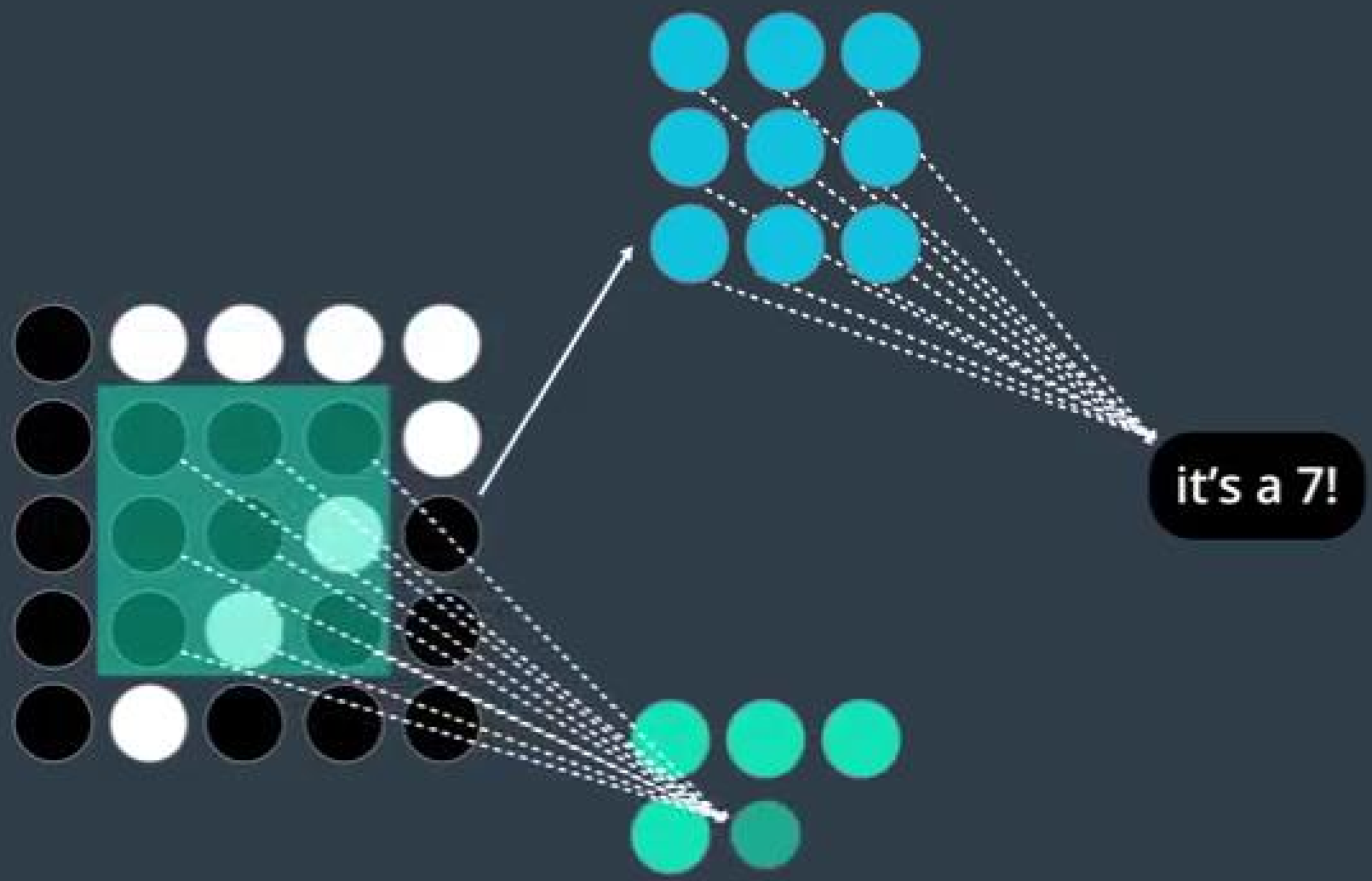


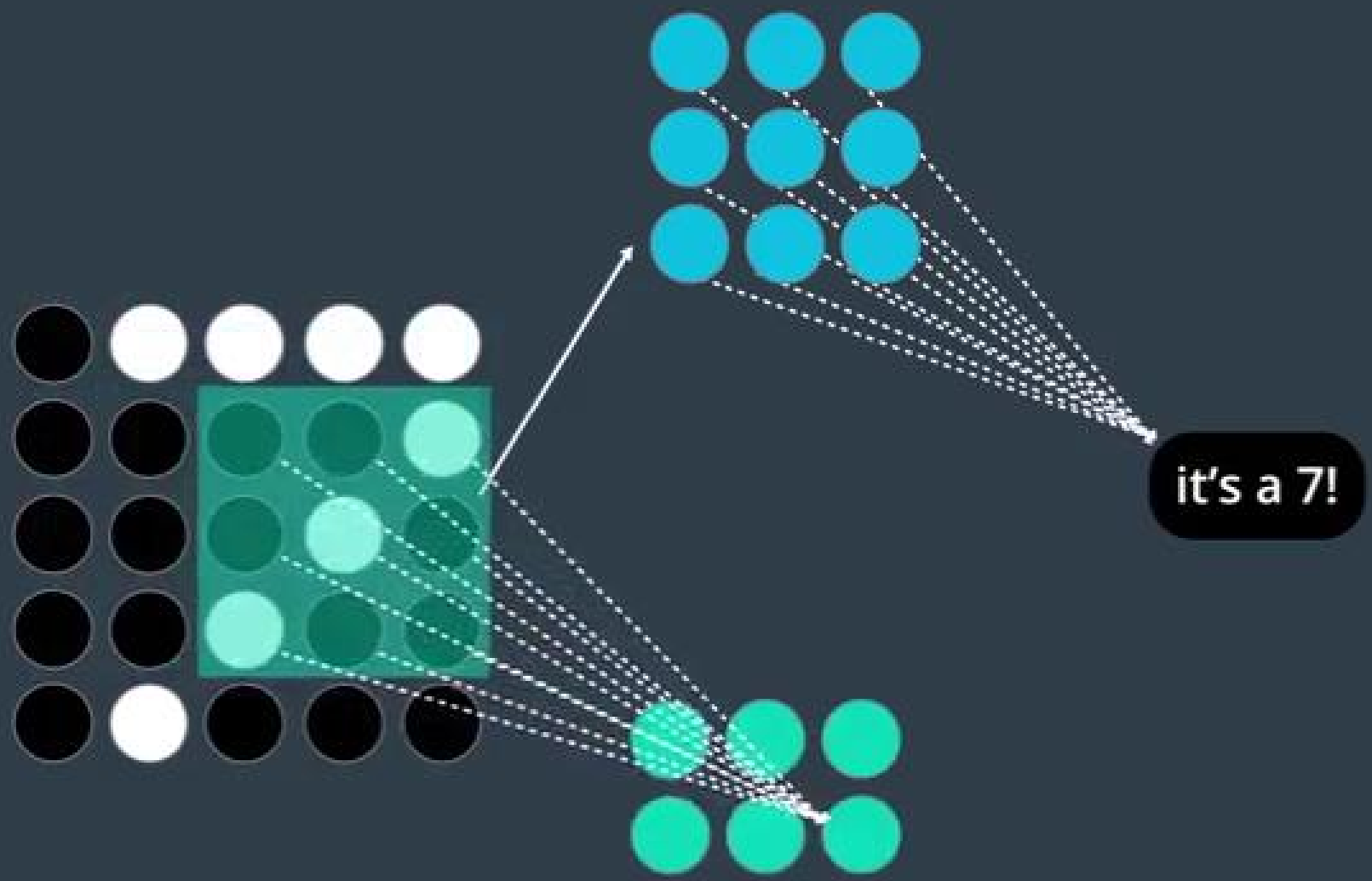


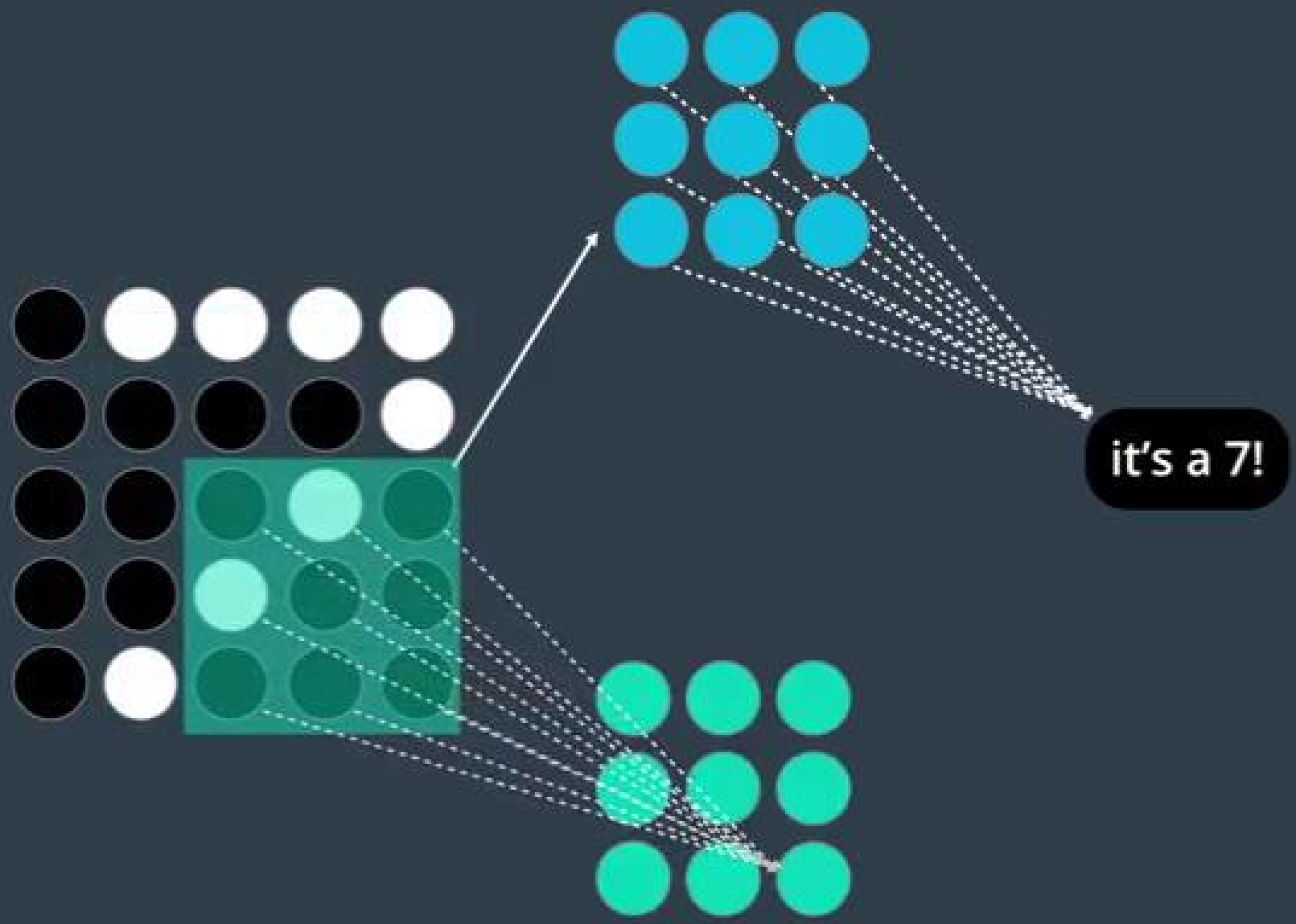

it's a 7!

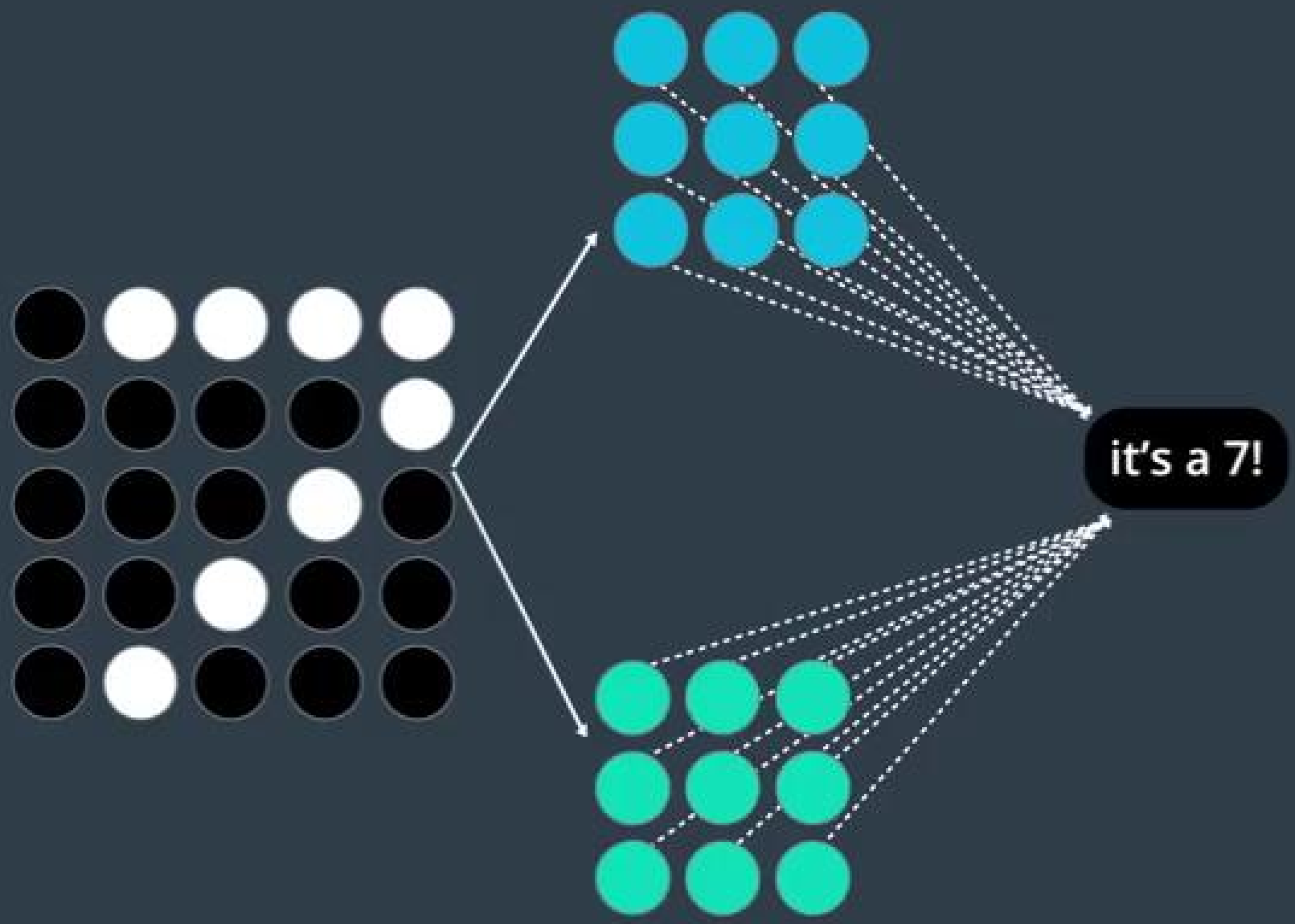

it's a 7!


it's a 7!

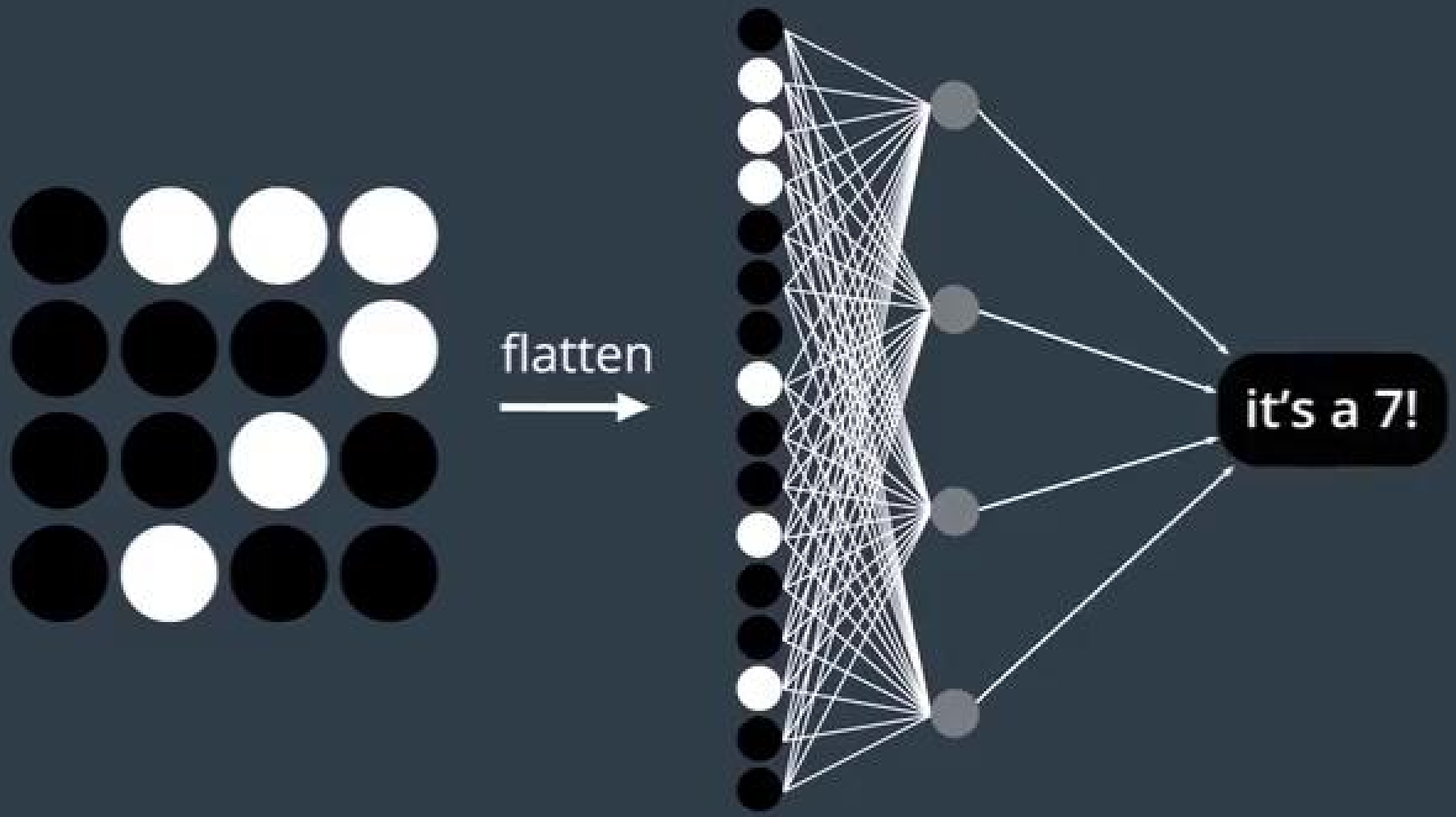

it's a 7!


it's a 7!

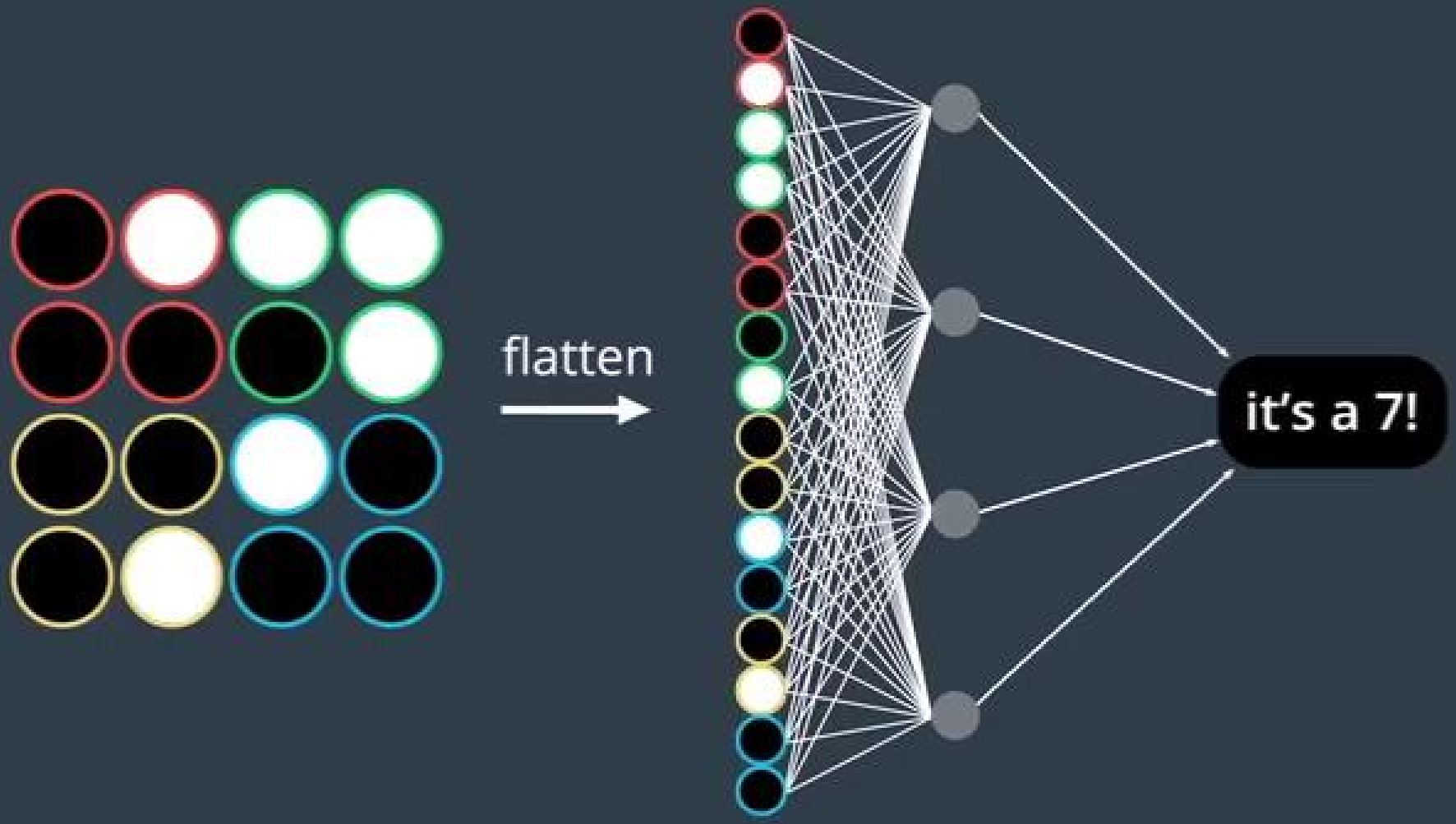

it's a 7!


it's a 7!

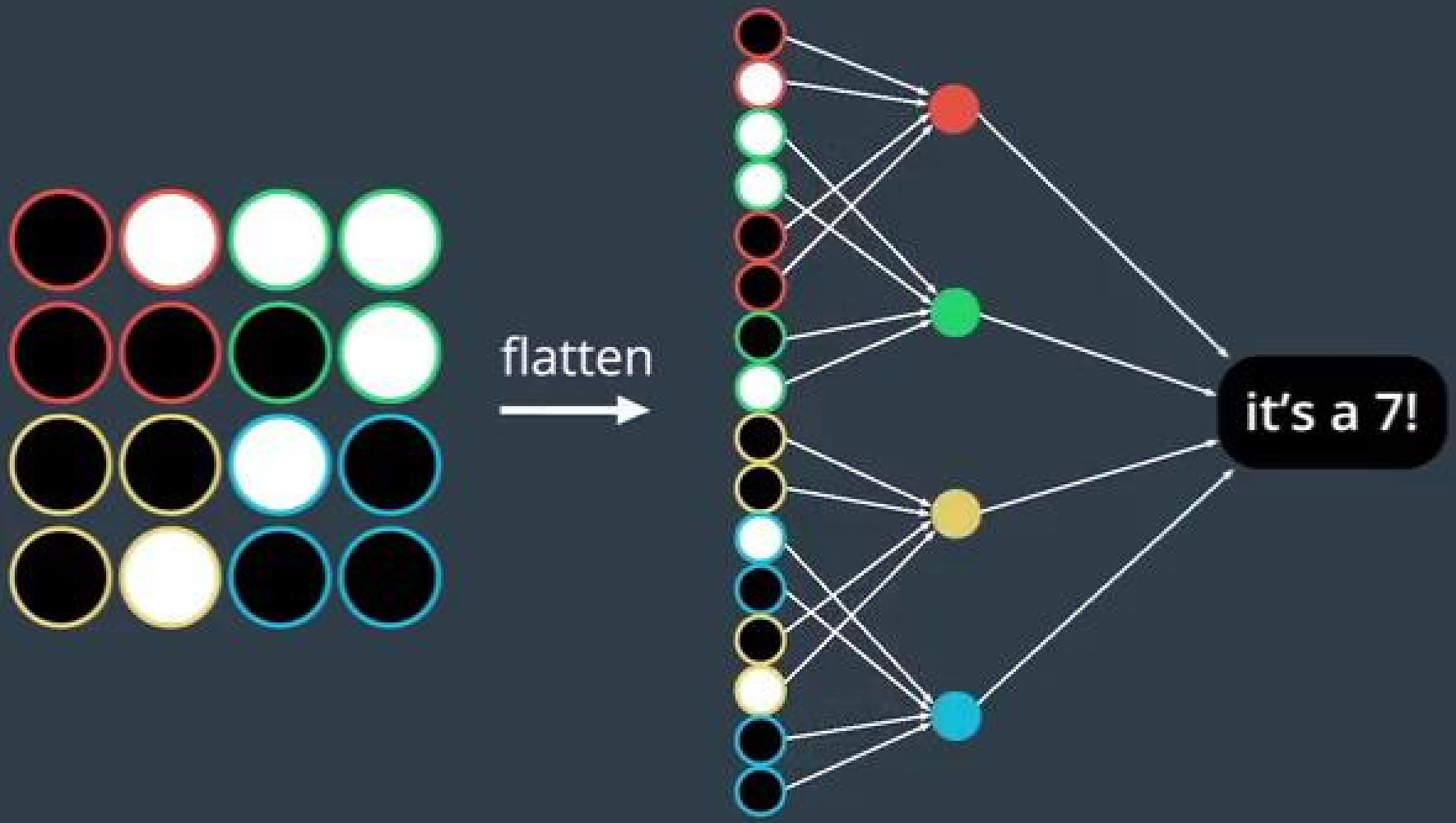
Densely Connected Layers



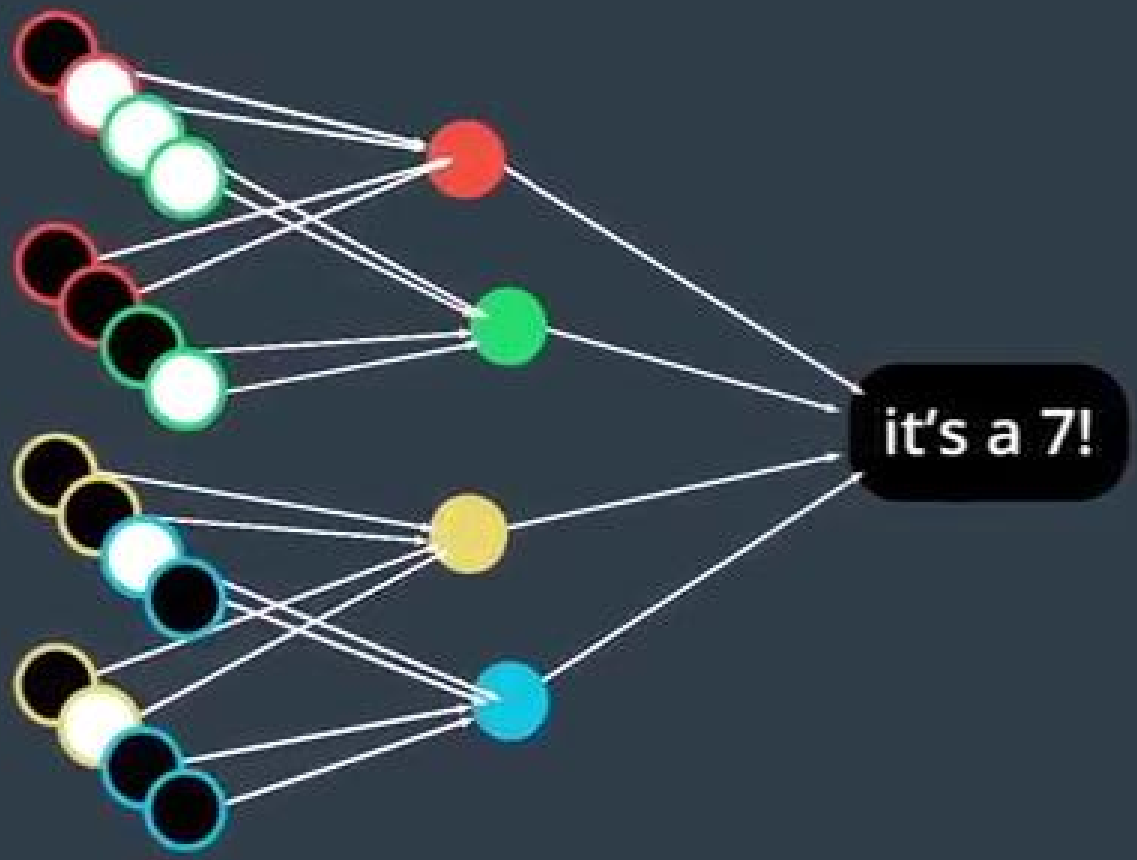
Densely Connected Layers



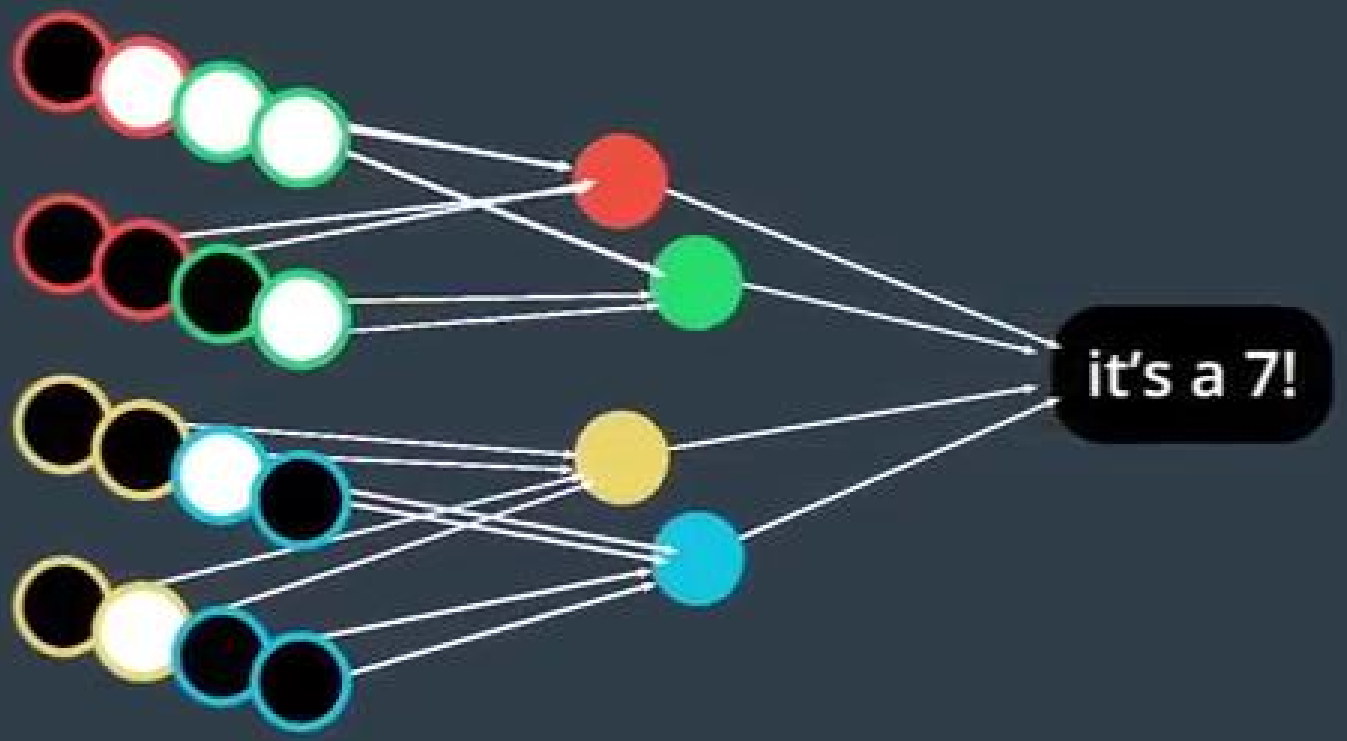
Locally Connected Layers



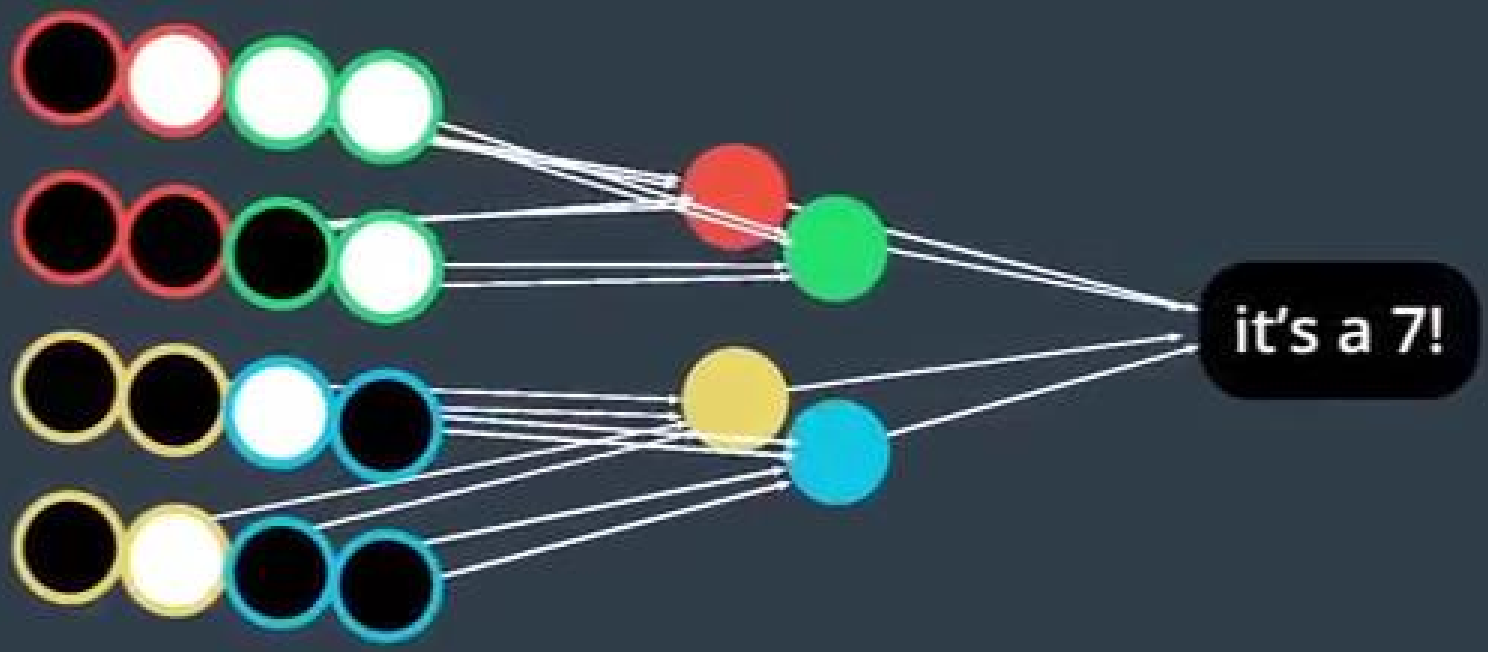
Locally Connected Layers



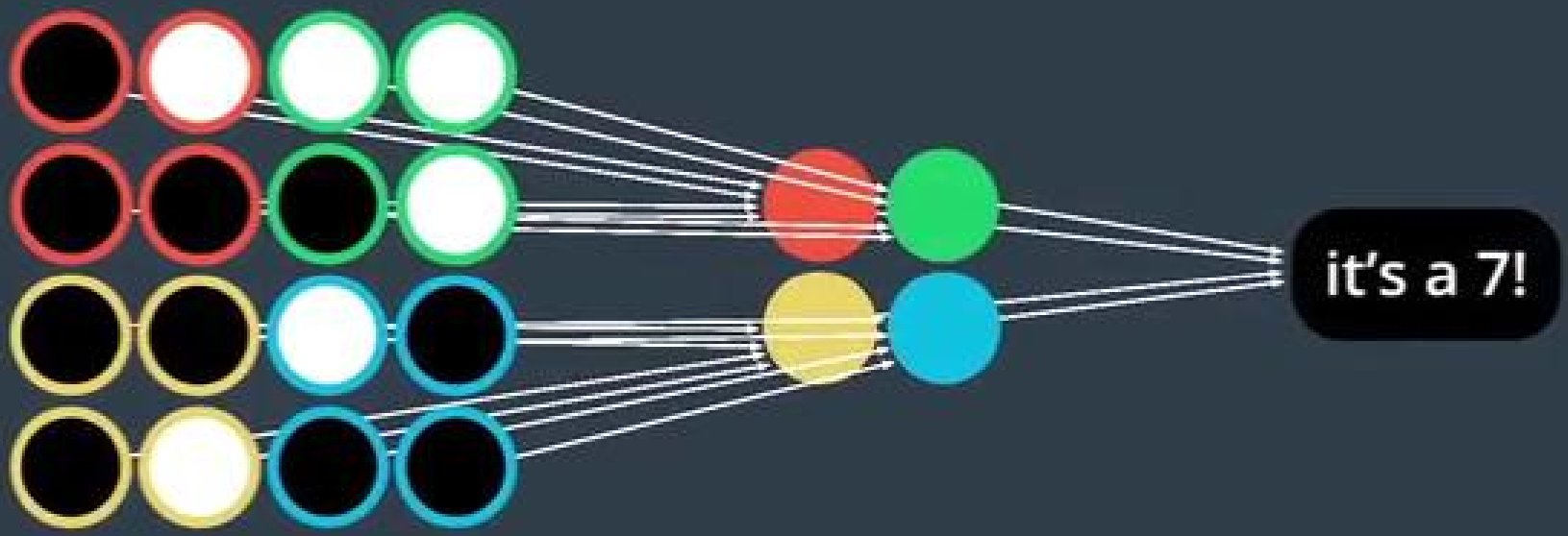
Locally Connected Layers



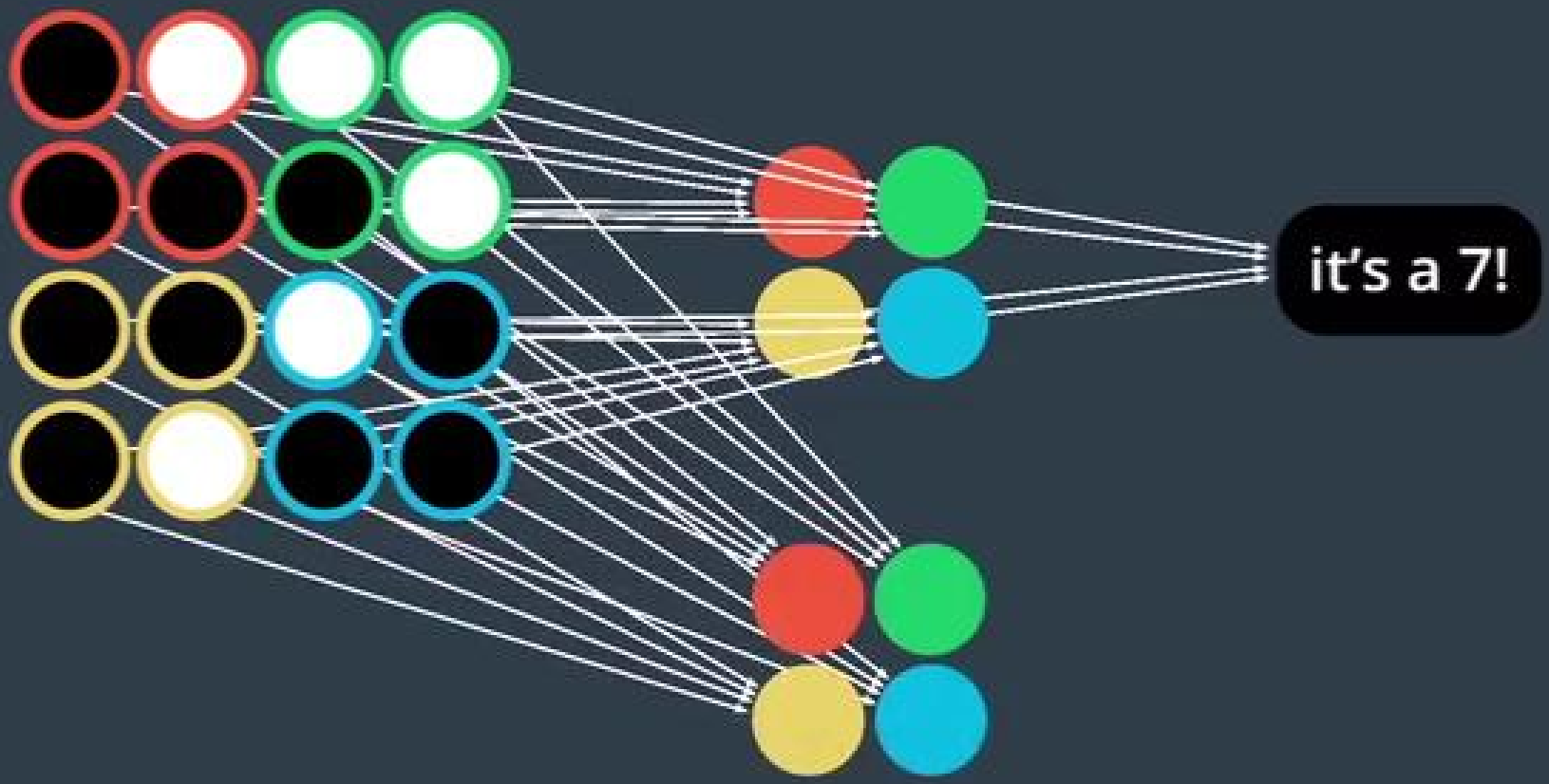
Locally Connected Layers



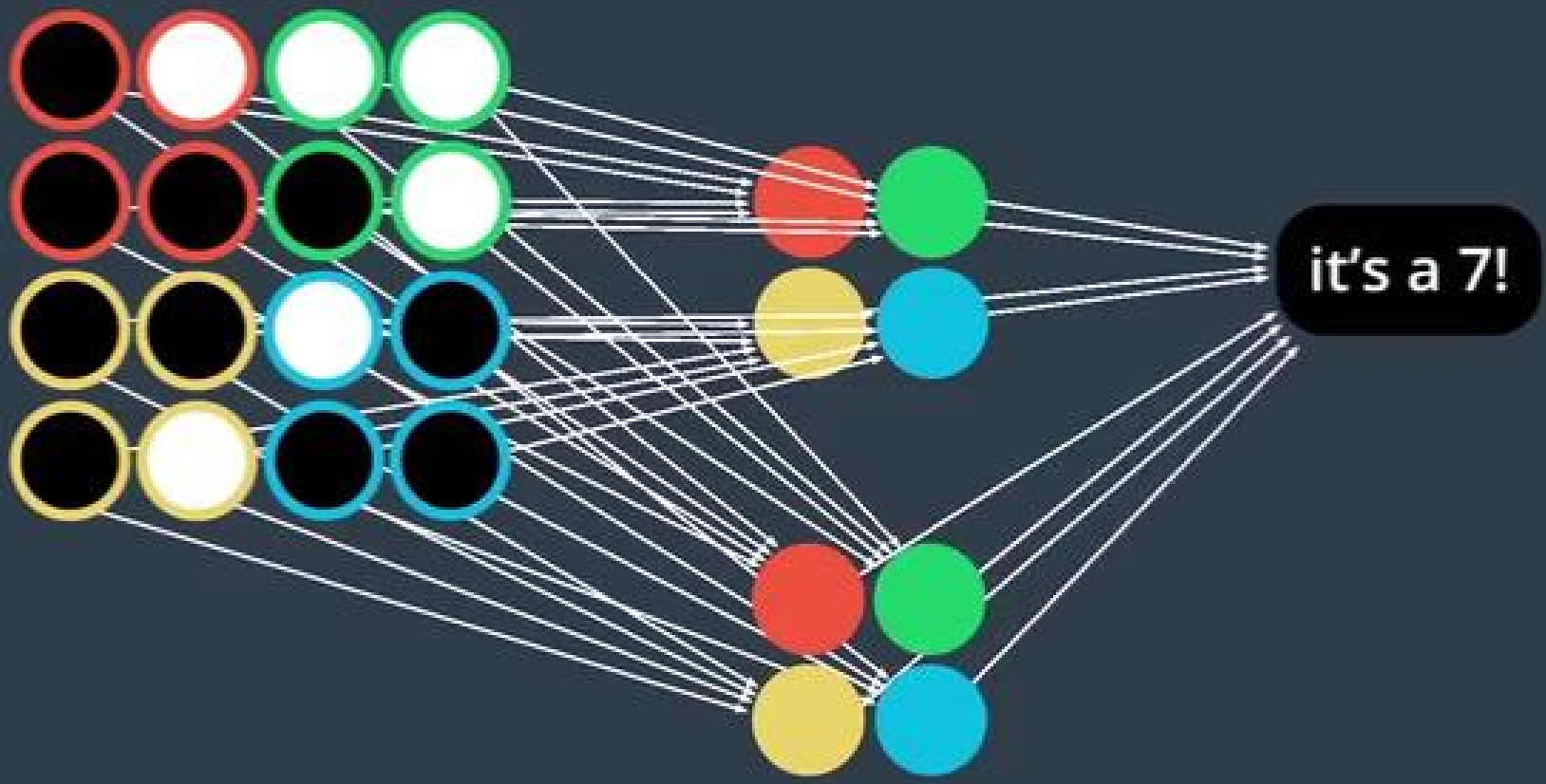
Locally Connected Layers

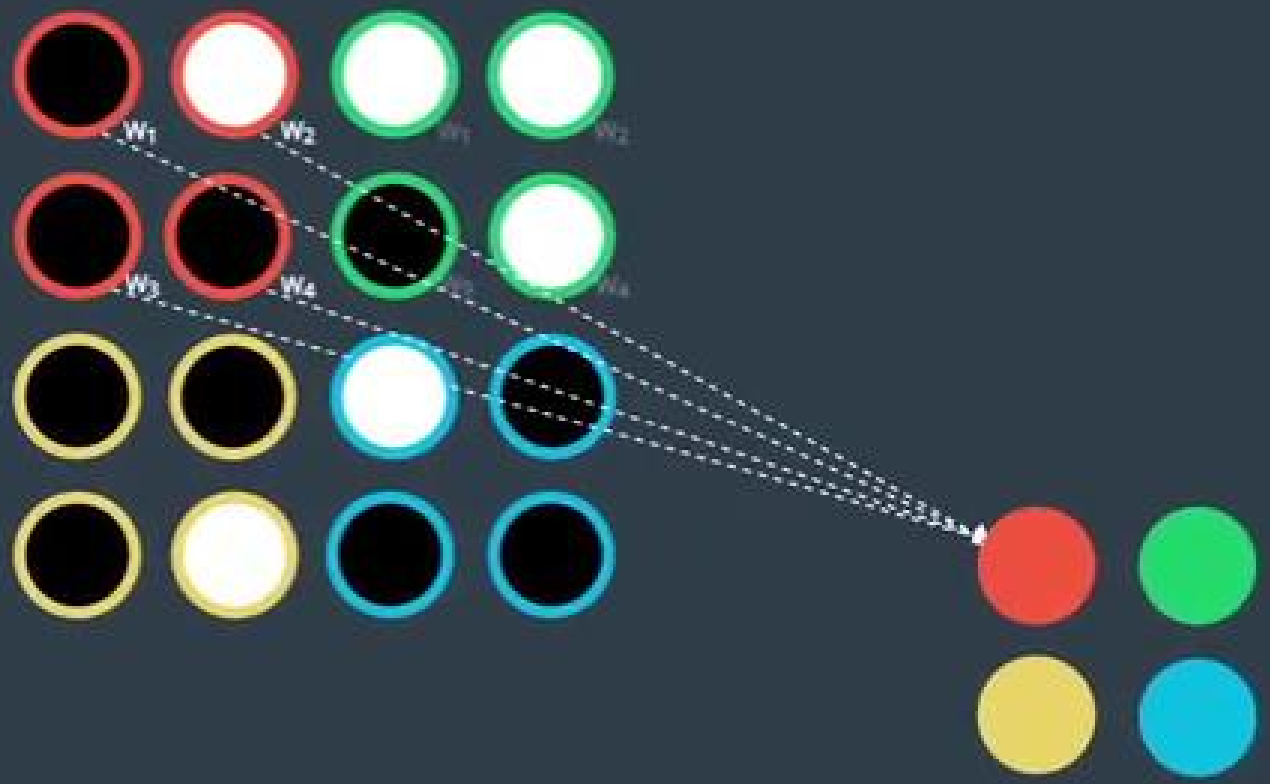


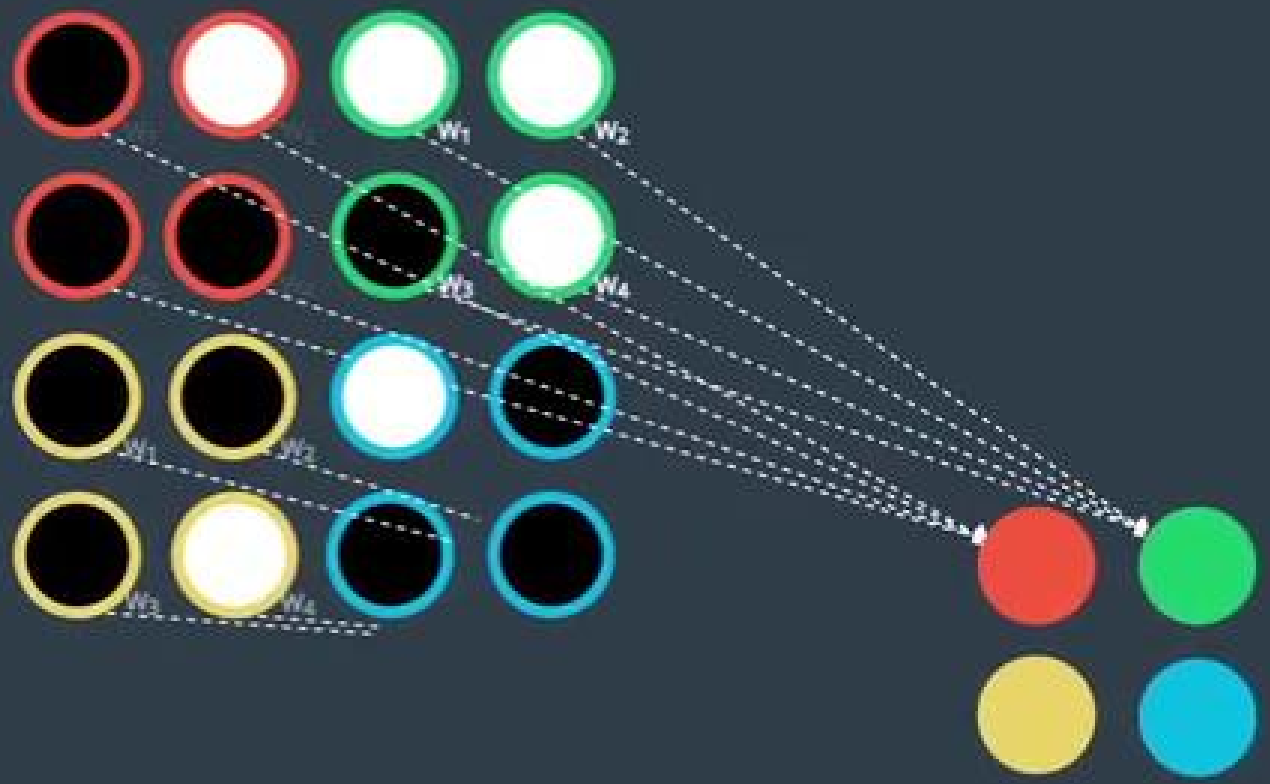
Locally Connected Layers

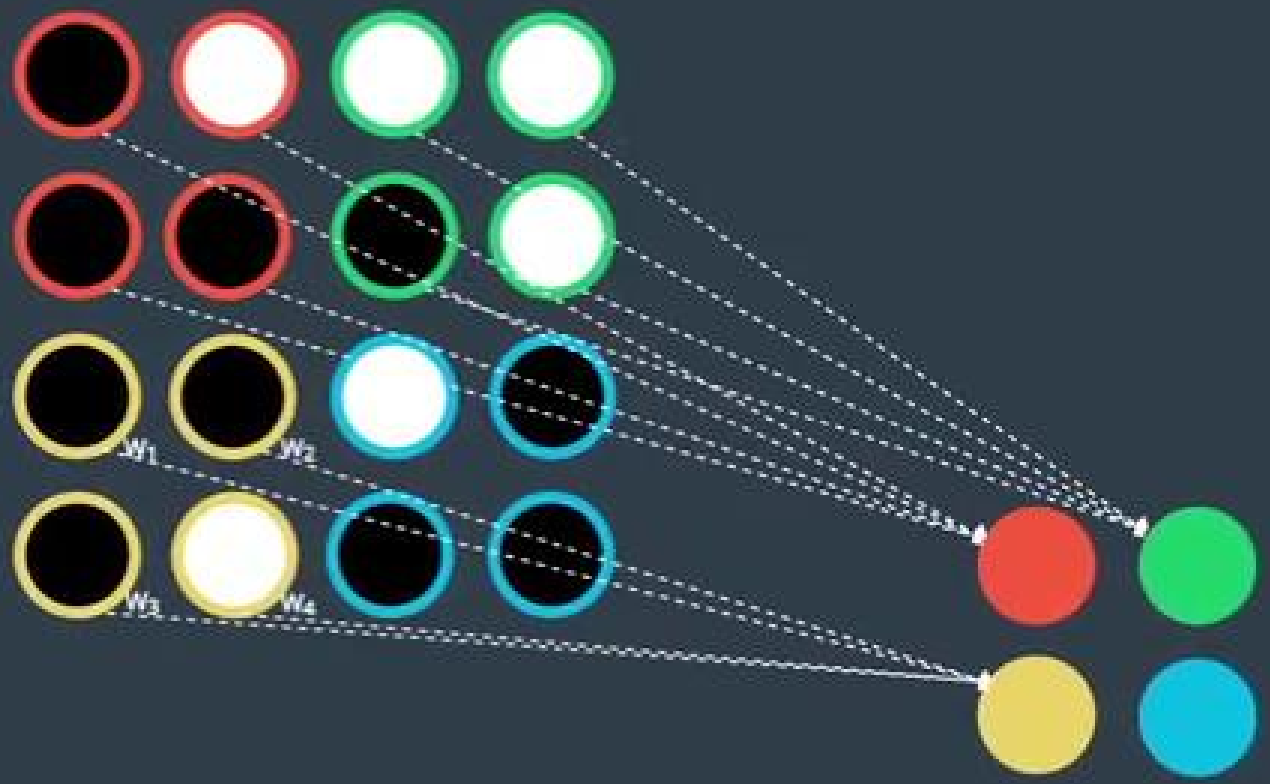


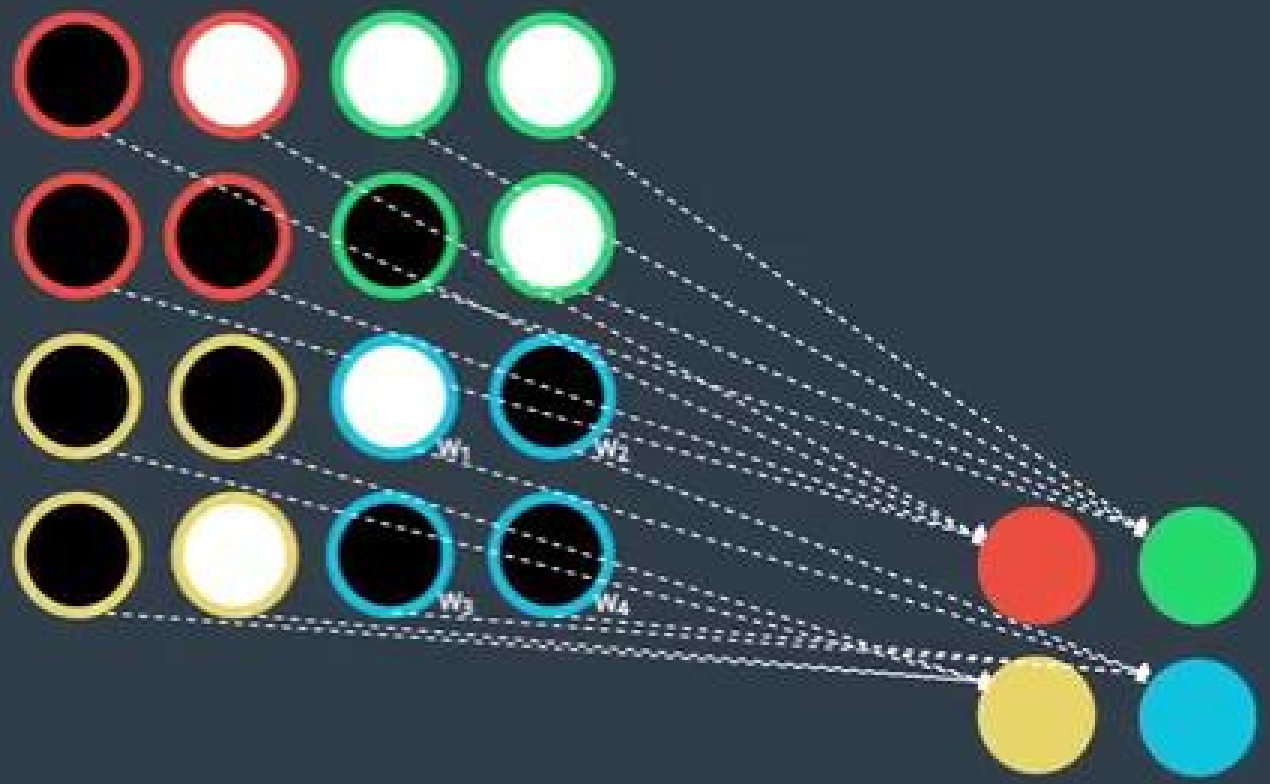
Locally Connected Layers



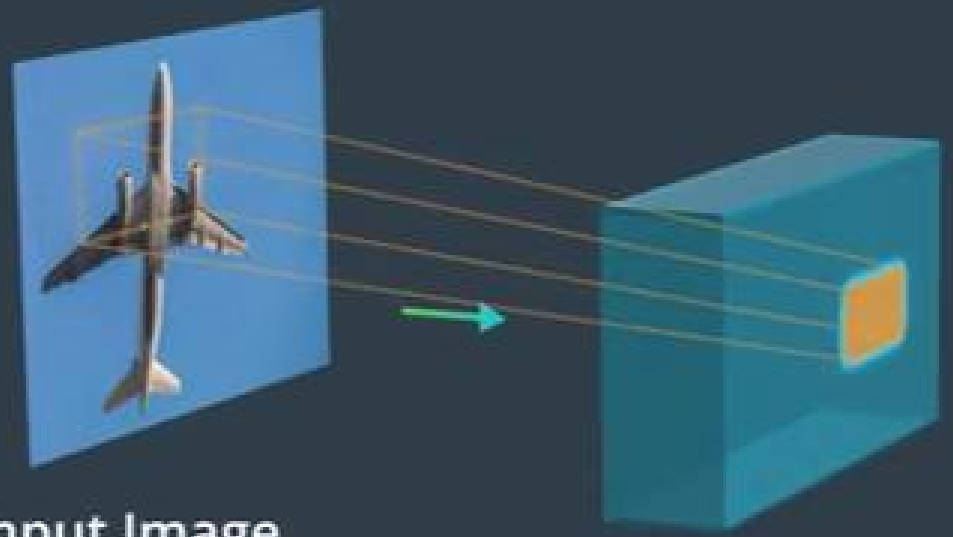

W1
W2
W3
W4


W1
W2
W3
W4


W1
W2
W3
W4

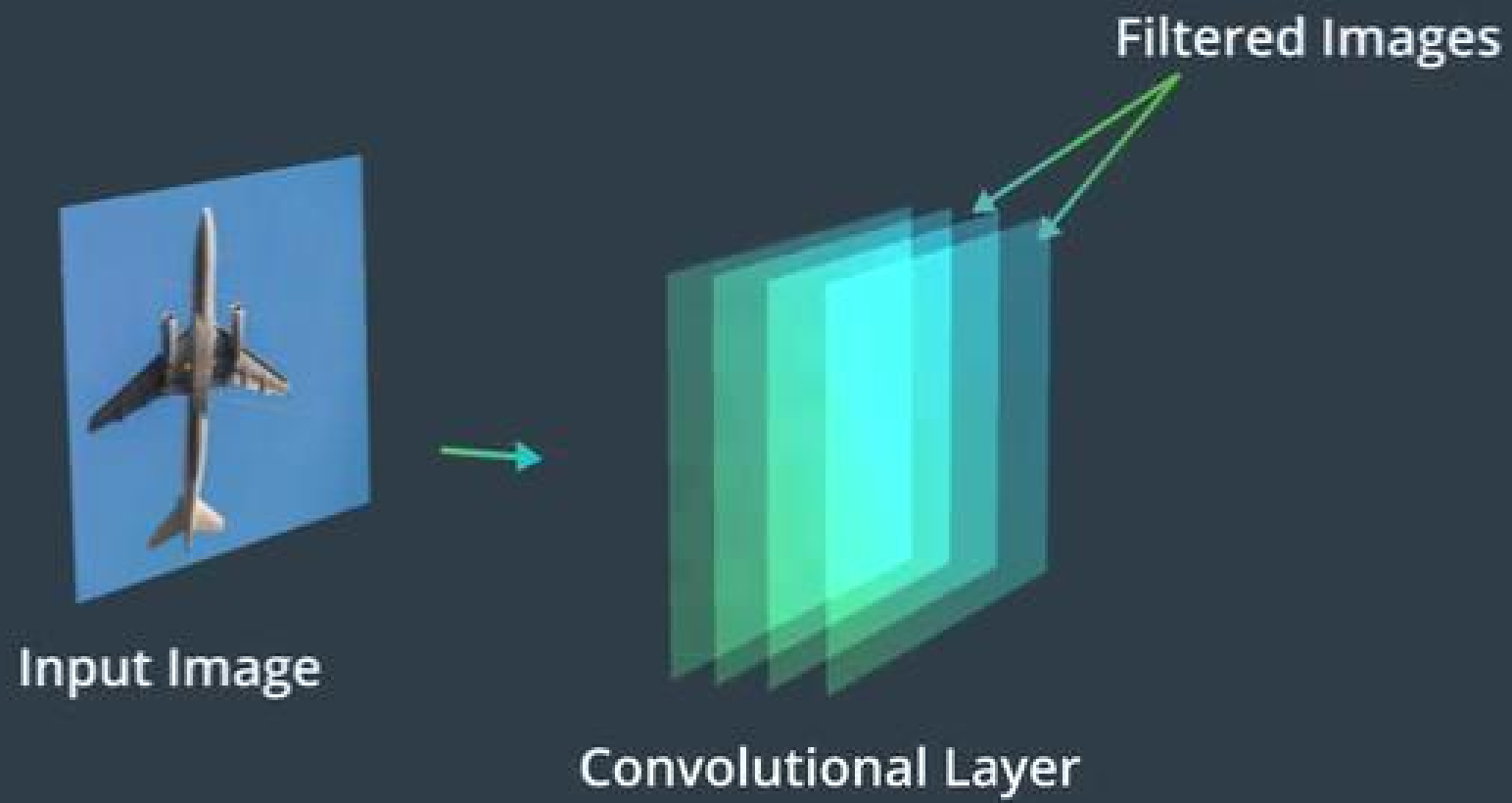

W1
W2
W3
W4

Convolutional Kernels



Input Image

Convolutional Layer


Filtered Images
Input Image
Convolutional Layer

Input Image

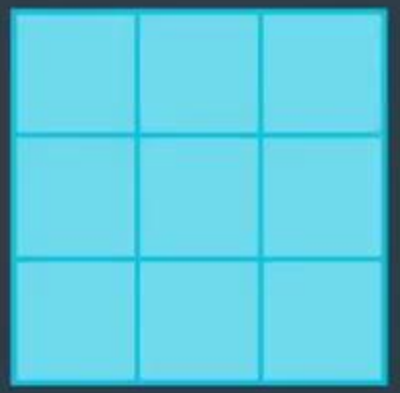
Filtered Images

Convolutional Layer

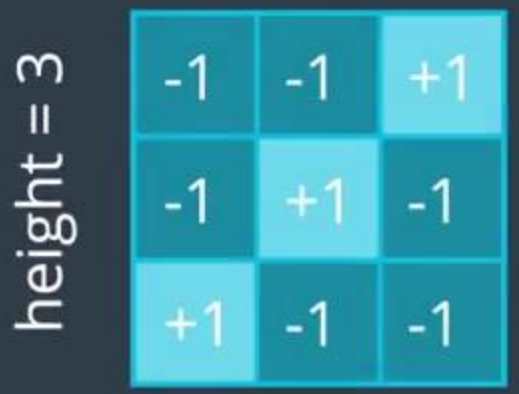
Window

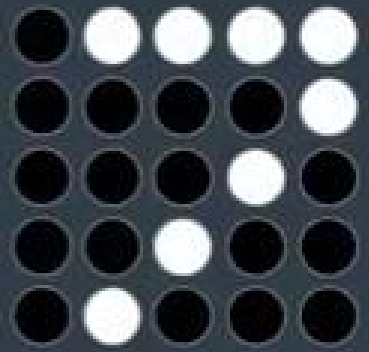
width = 3

height = 3



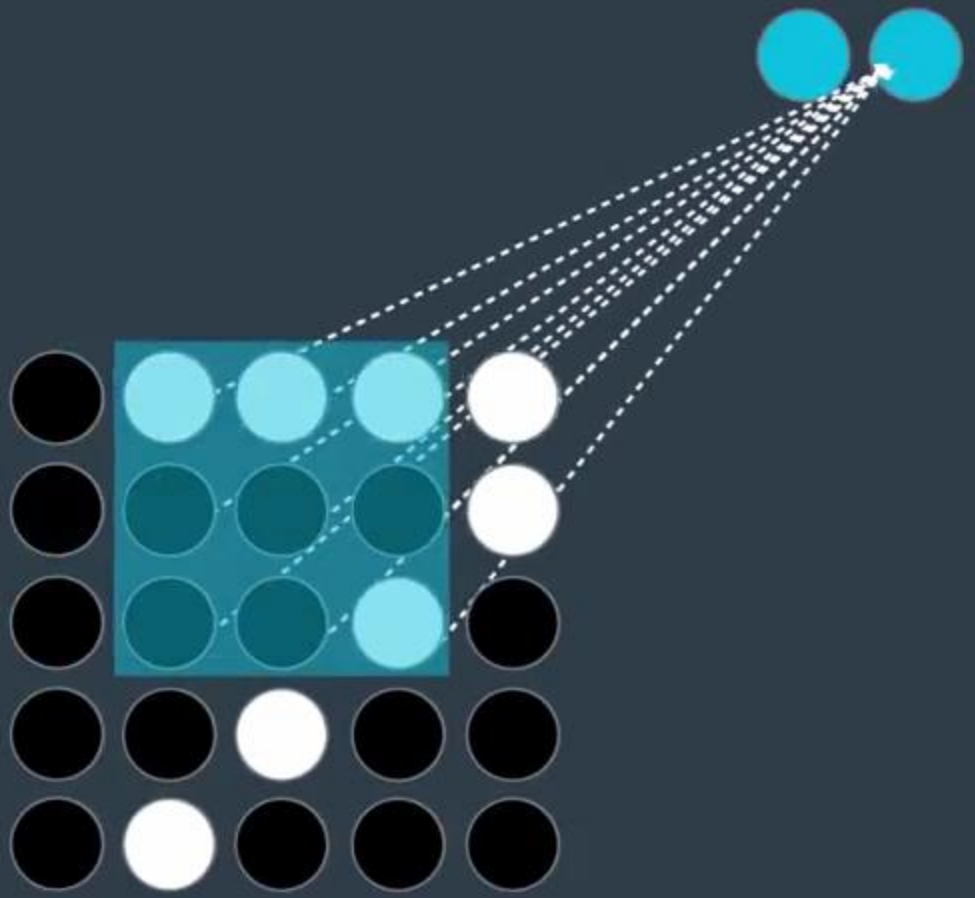
Filter

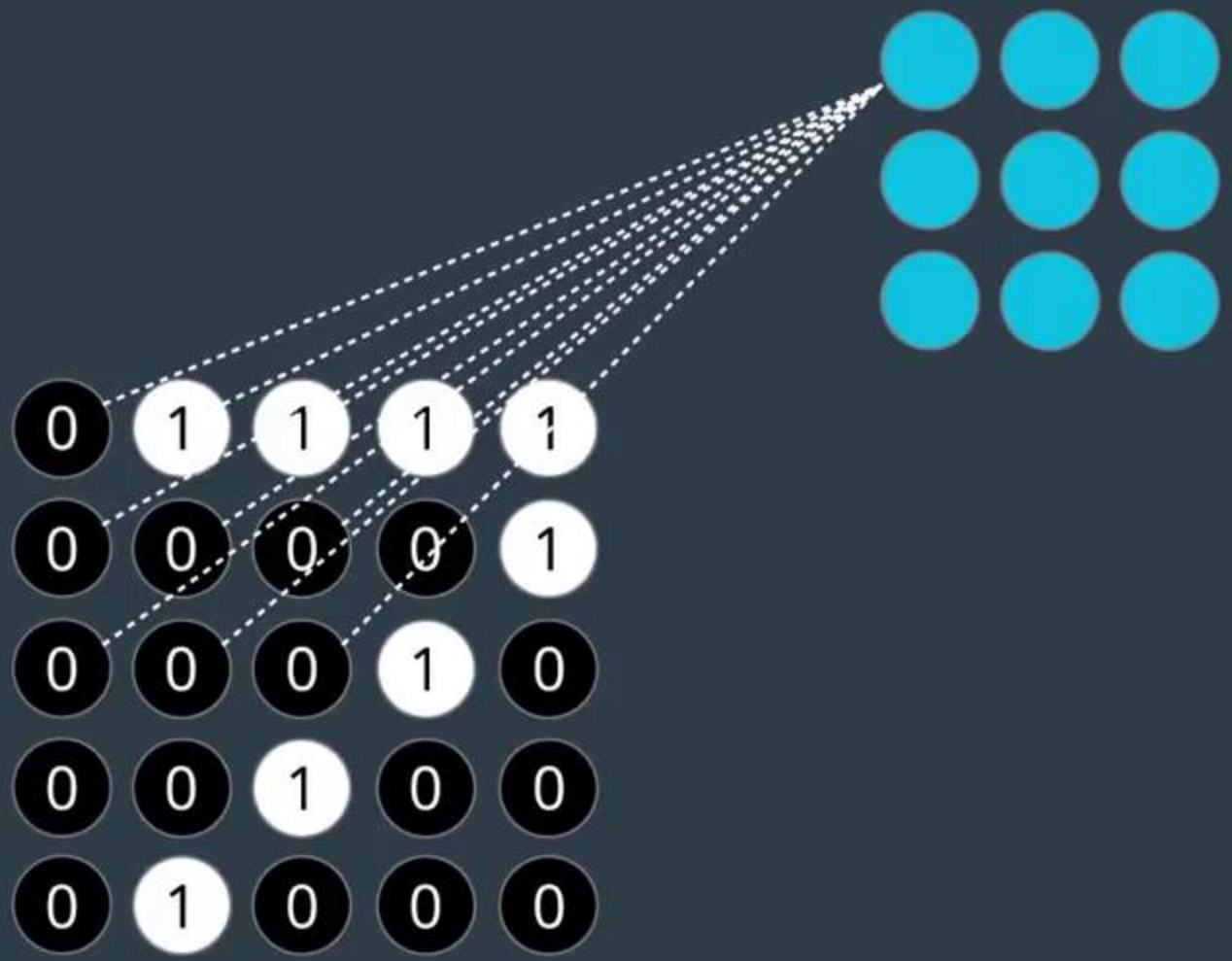


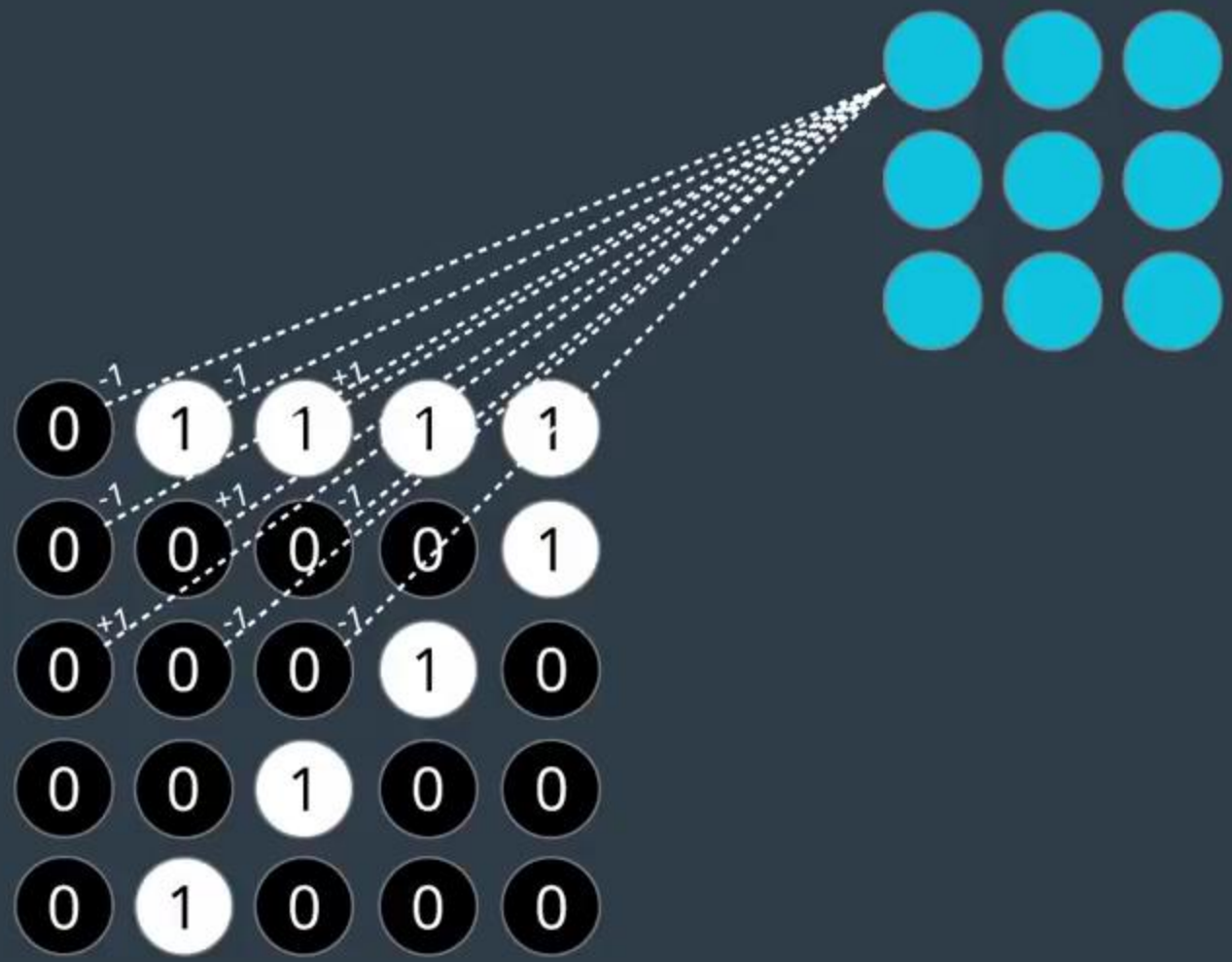


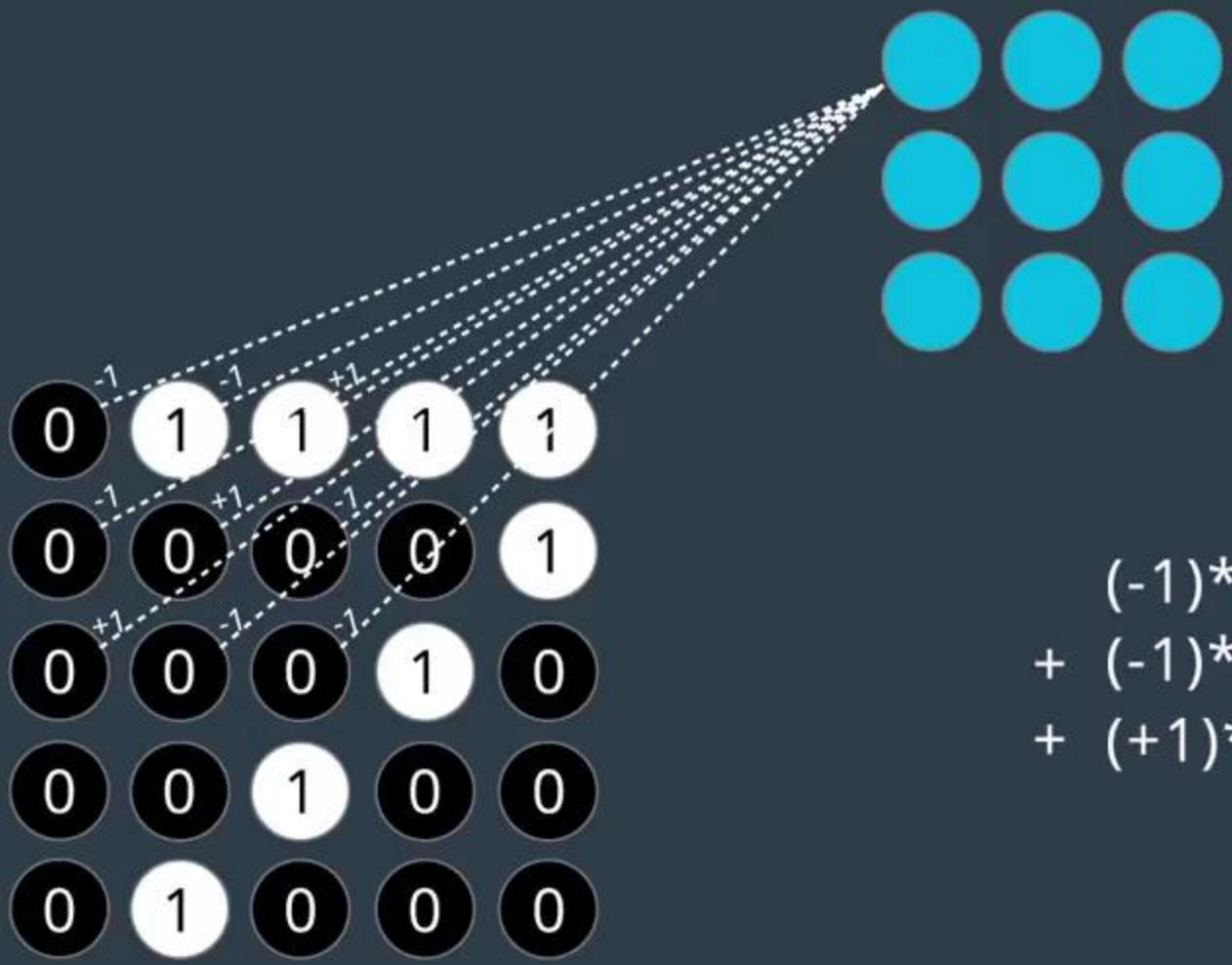
-1	-1	+1
-1	+1	-1
+1	-1	-1

Input Layer ⟶ Hidden Layer ⟶ Output Layer

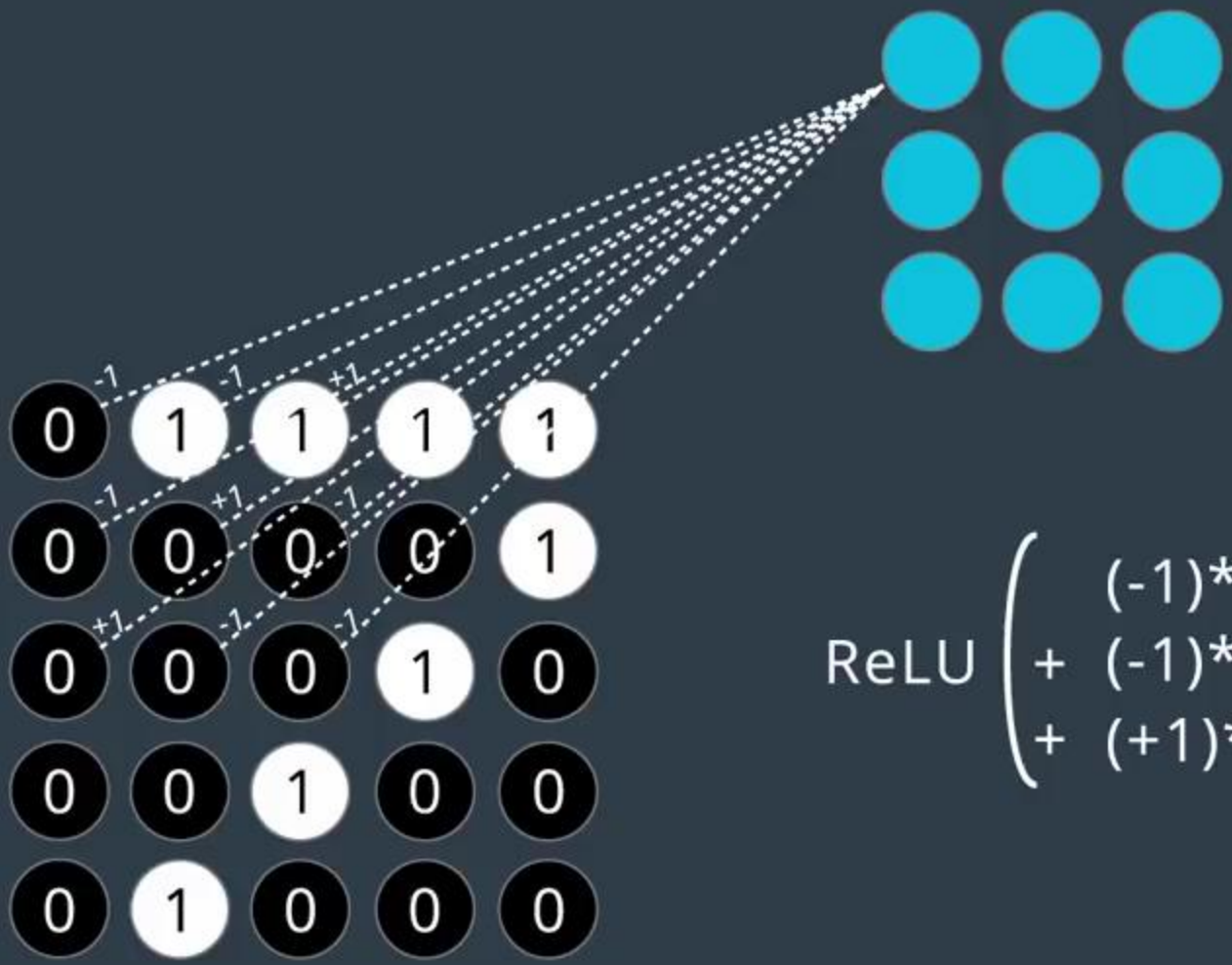



0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0

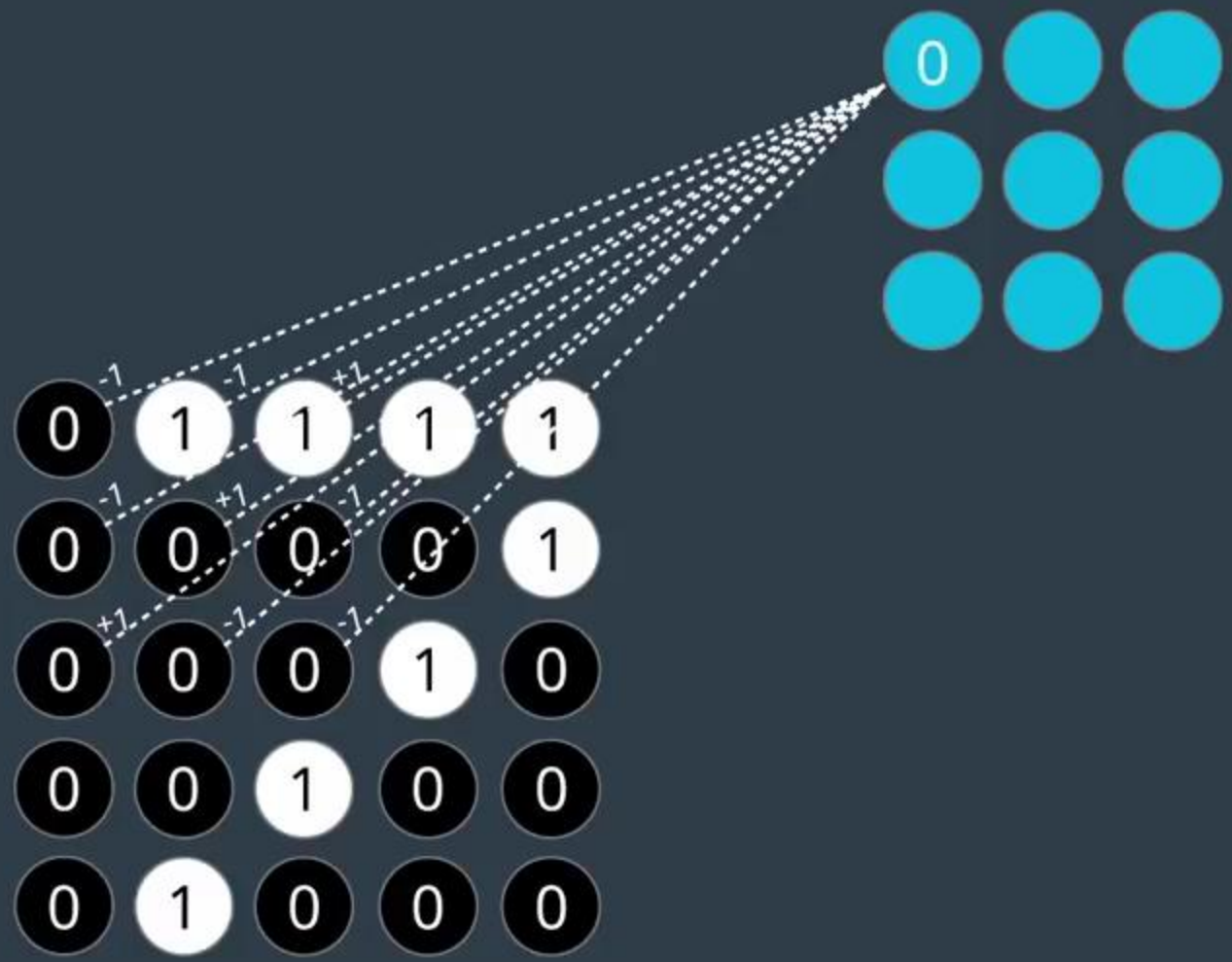

-1
-1
+1
0 1 1 1 1
-1
+1
-1
0 0 0 0 1
+1
-1
-1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0

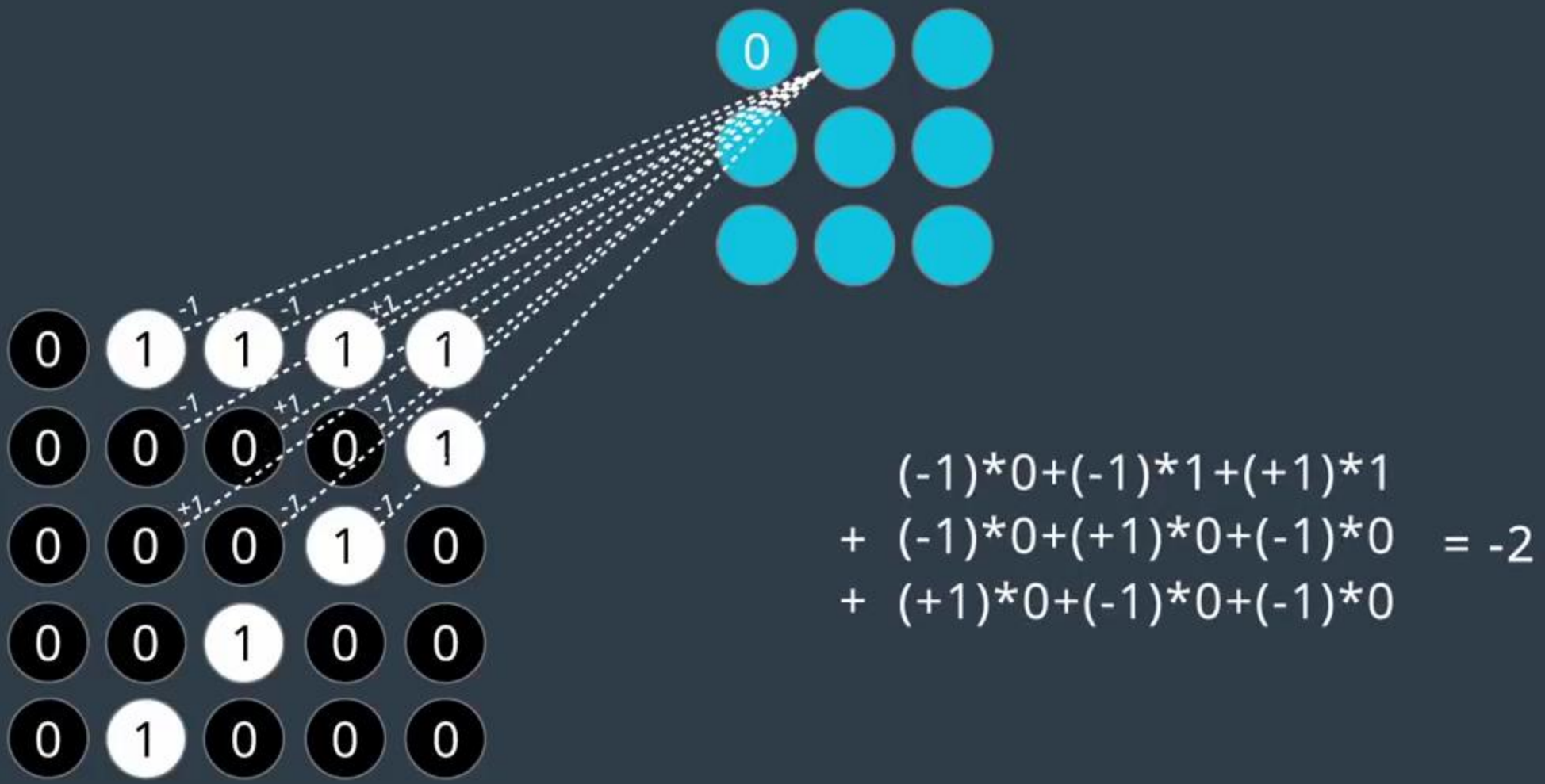


$$
\begin{aligned}
&(-1)*0+(-1)*1+(+1)*1 \\
+\;&(-1)*0+(+1)*0+(-1)*0 \quad = 0 \\
+\;&(+1)*0+(-1)*0+(-1)*0
\end{aligned}
$$

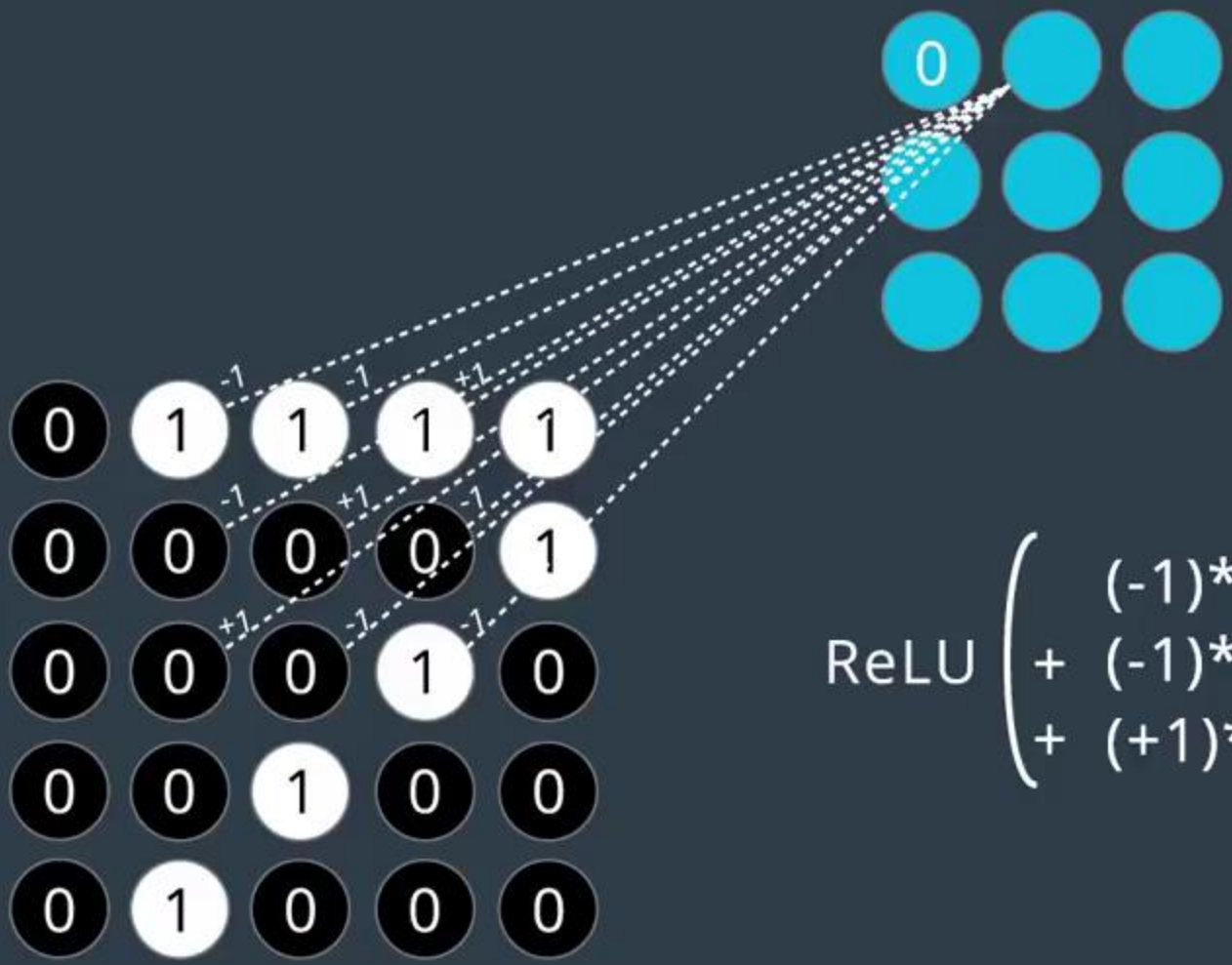


$$\text{ReLU}\left(\begin{matrix}(-1)*0+(-1)*1+(+1)*1\\ +\ (-1)*0+(+1)*0+(-1)*0\\ +\ (+1)*0+(-1)*0+(-1)*0\end{matrix}\right) = 0$$

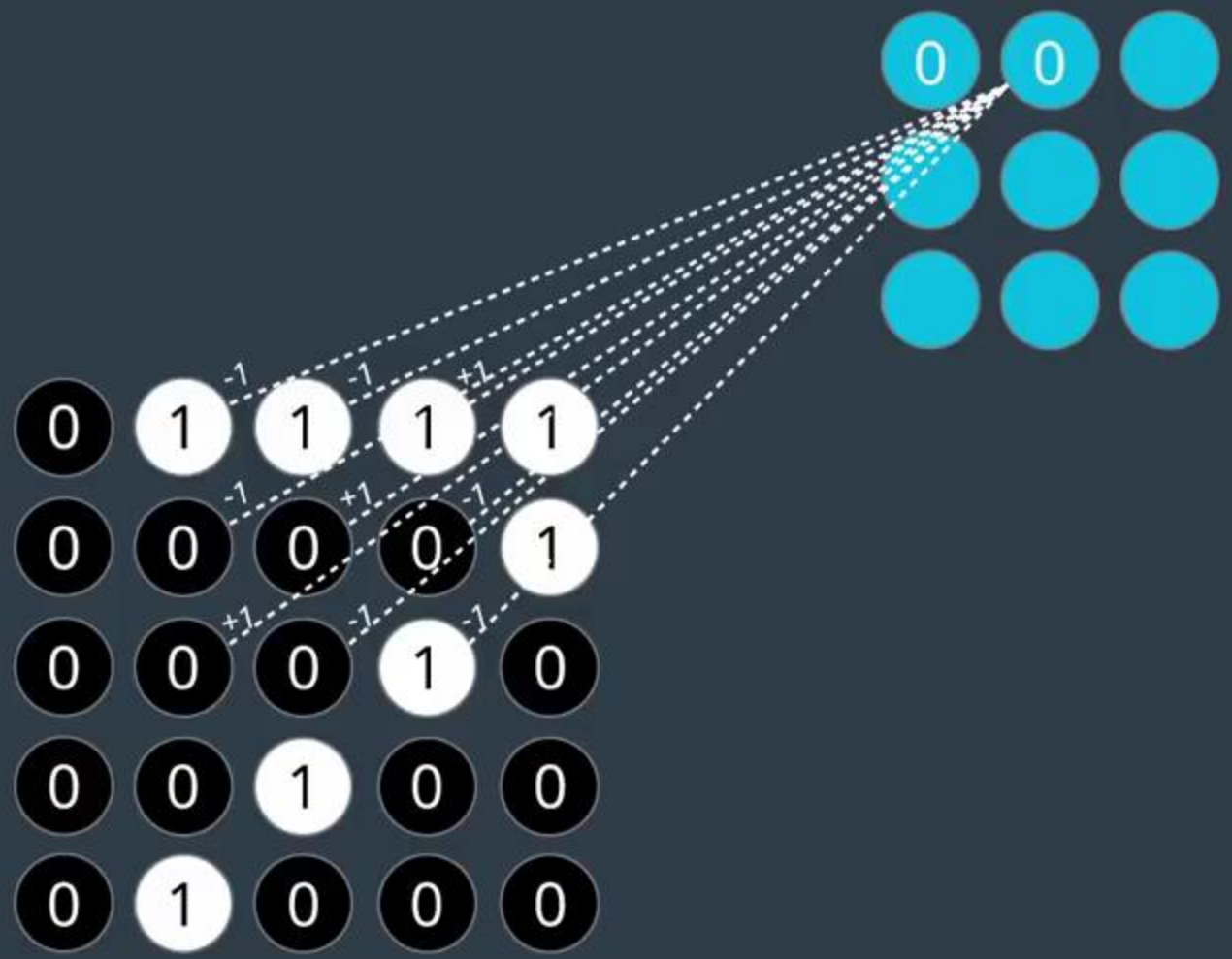

0
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
-1
-1
+1
-1
+1
-1
+1
-1
-1

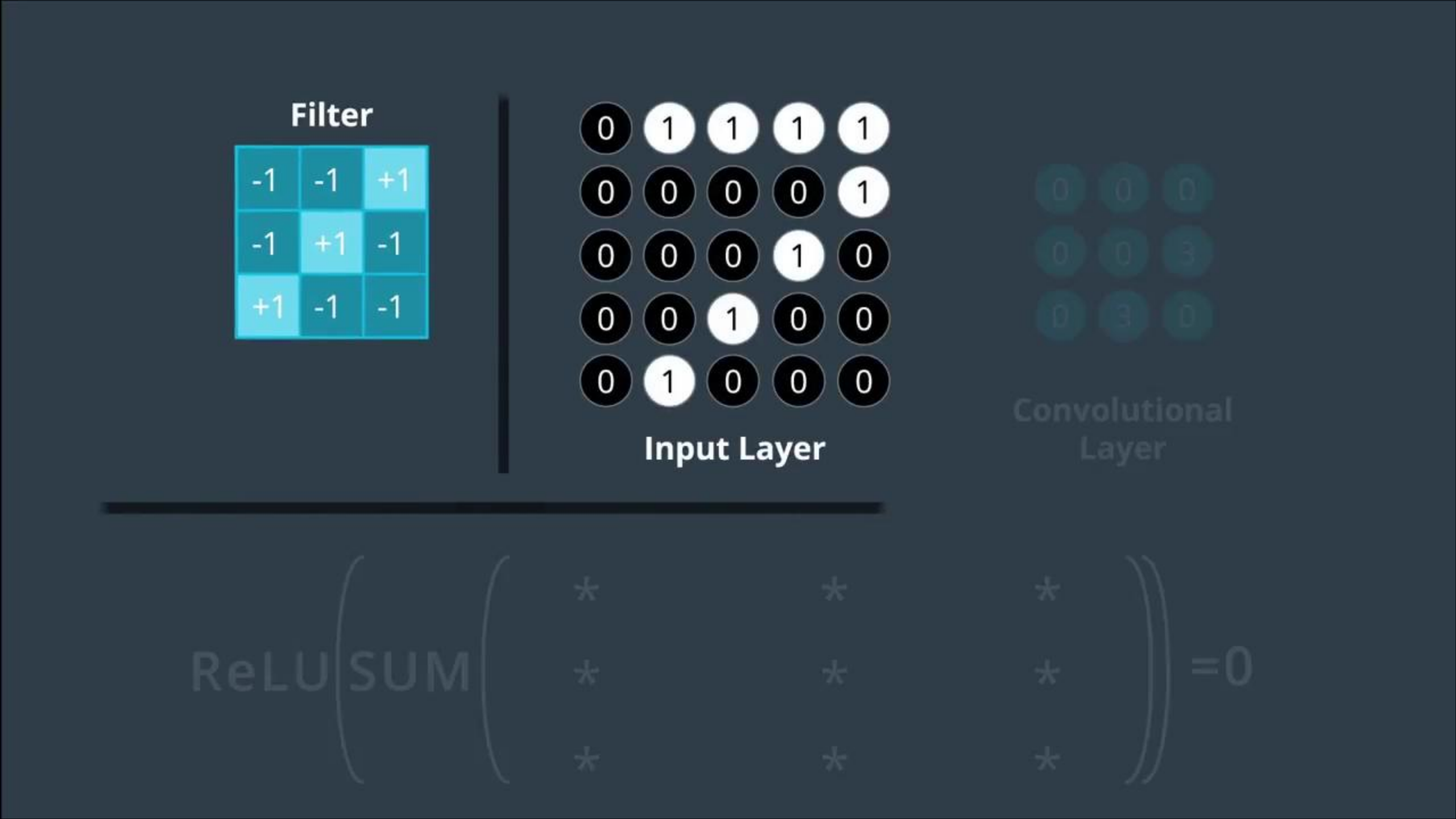


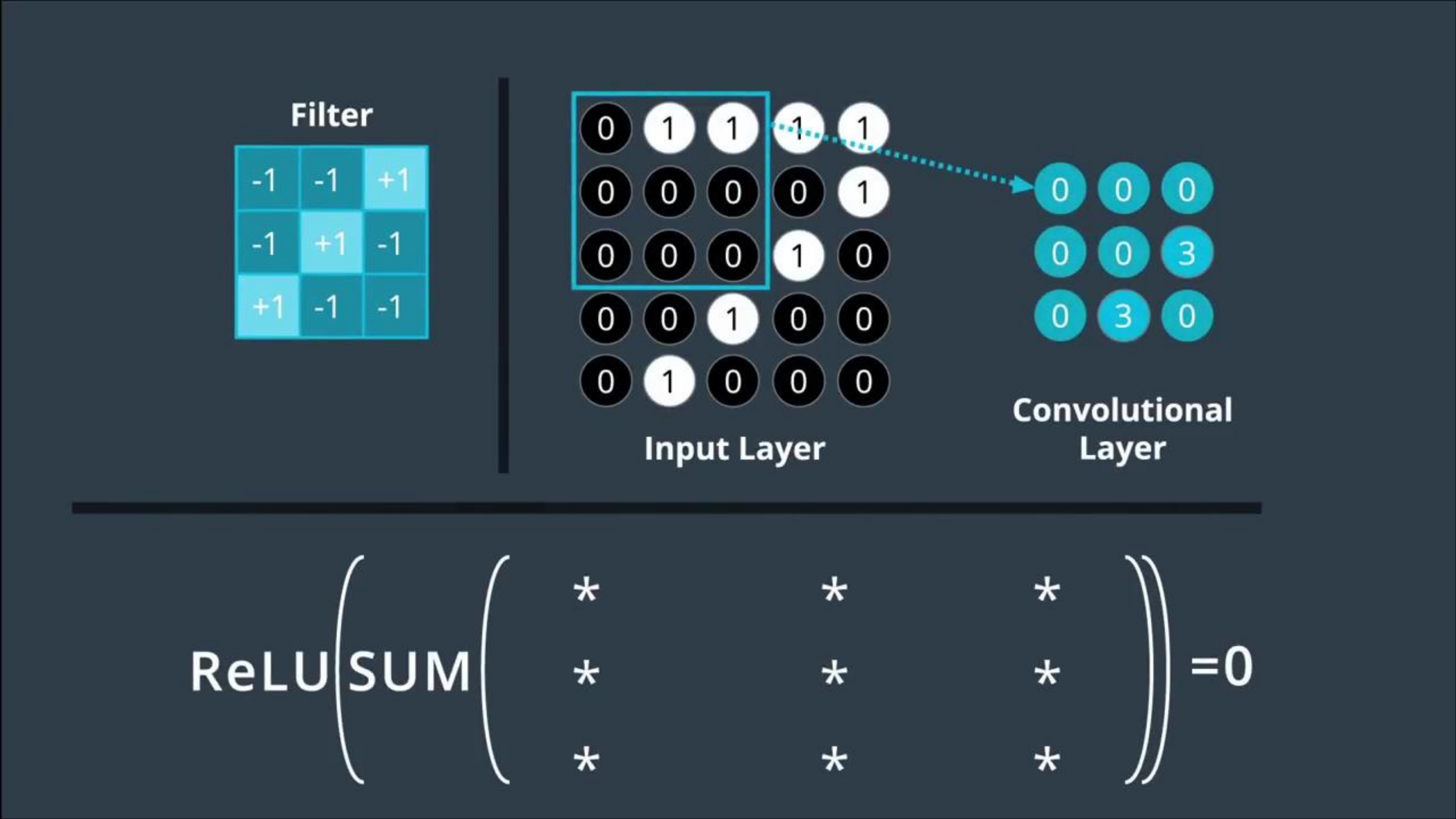
$$
\begin{aligned}
&(-1)*0+(-1)*1+(+1)*1 \\
+\ &(-1)*0+(+1)*0+(-1)*0 \quad = -2 \\
+\ &(+1)*0+(-1)*0+(-1)*0
\end{aligned}
$$

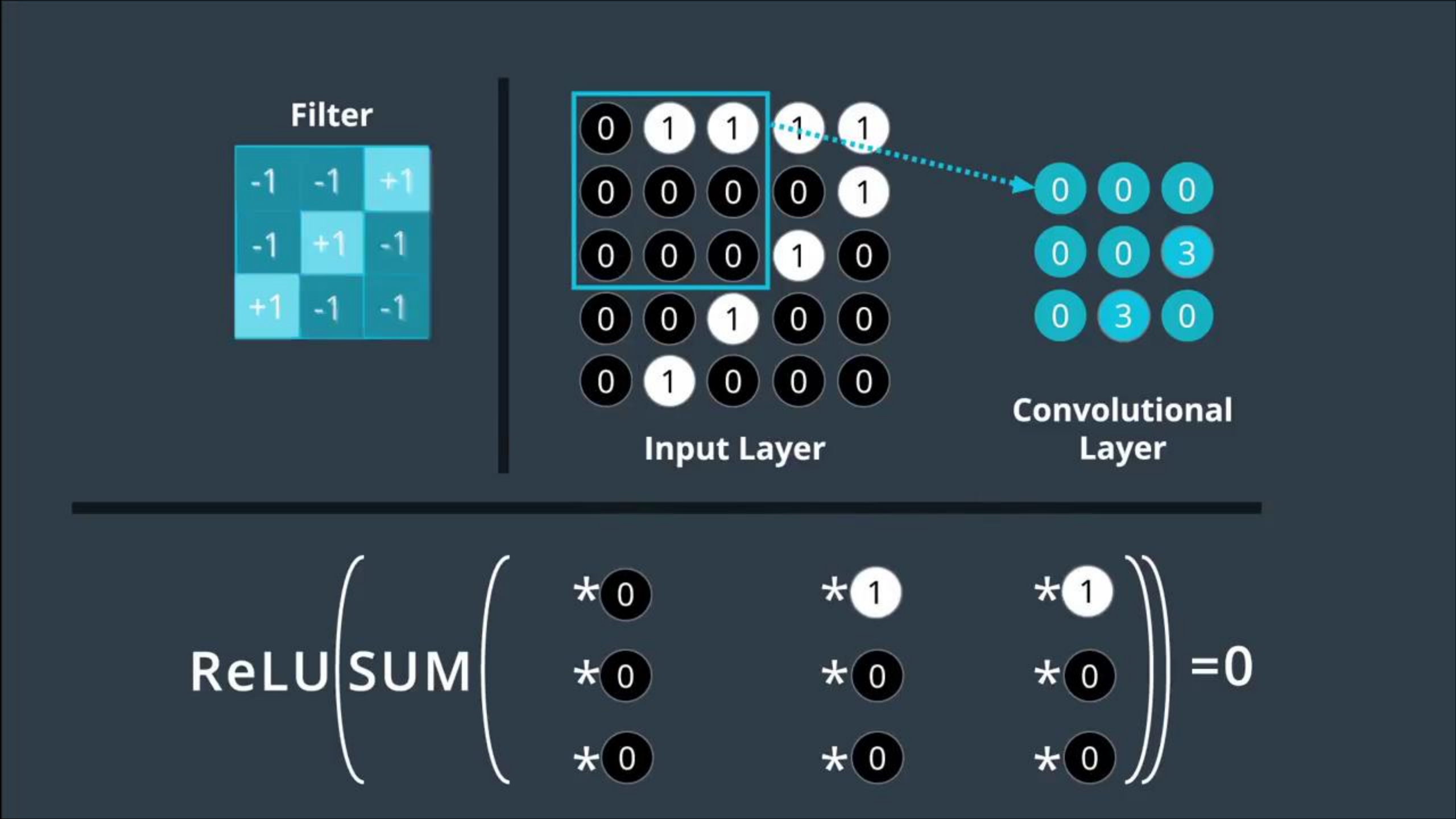


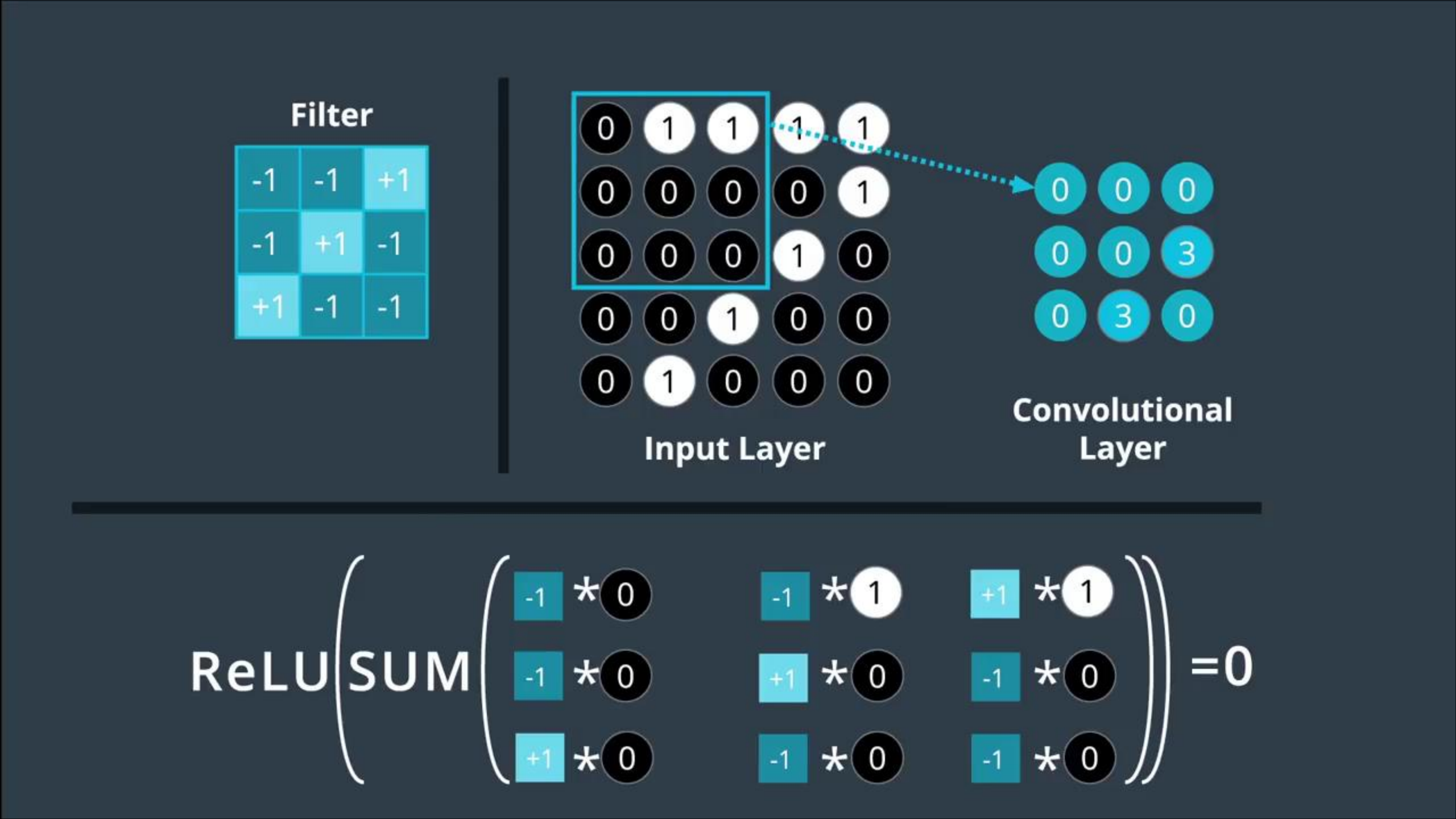
$$\text{ReLU}\left(\begin{matrix}(-1)*0+(-1)*1+(+1)*1\\ +\ (-1)*0+(+1)*0+(-1)*0\\ +\ (+1)*0+(-1)*0+(-1)*0\end{matrix}\right) = 0$$

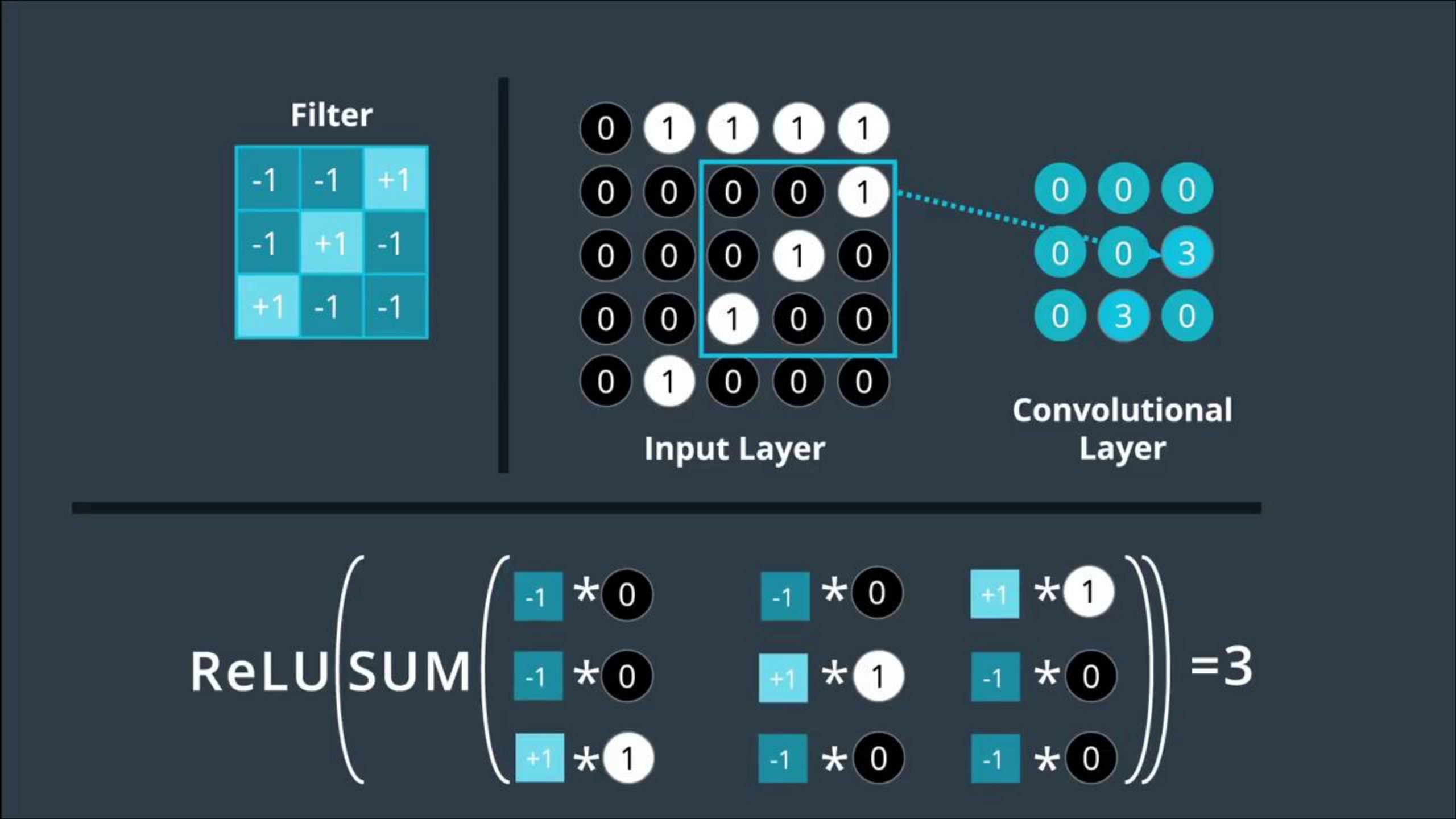

0 0
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
-1 -1 +1
-1 +1 -1
+1 -1 -1

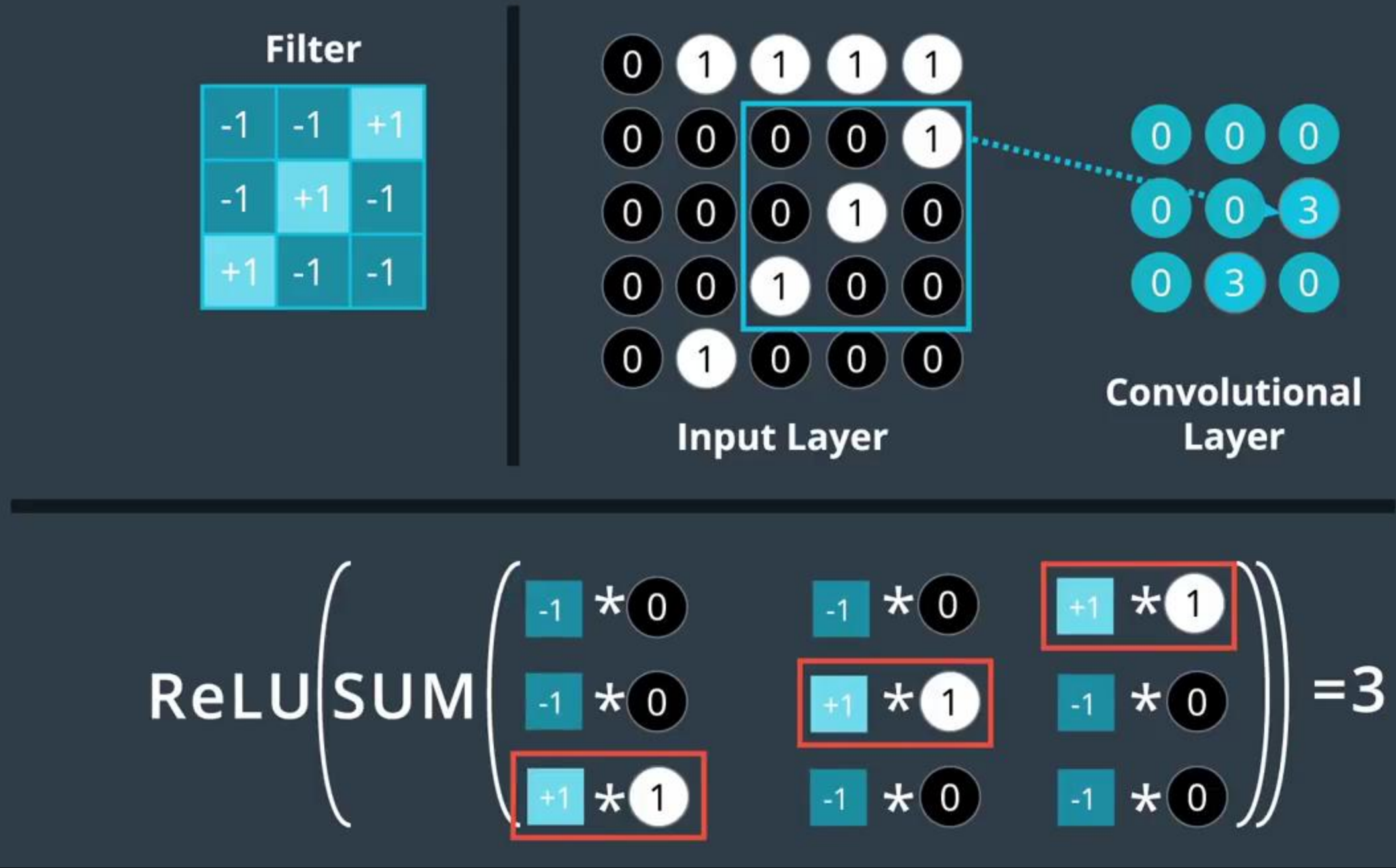

Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
Convolutional Layer
ReLU SUM =0

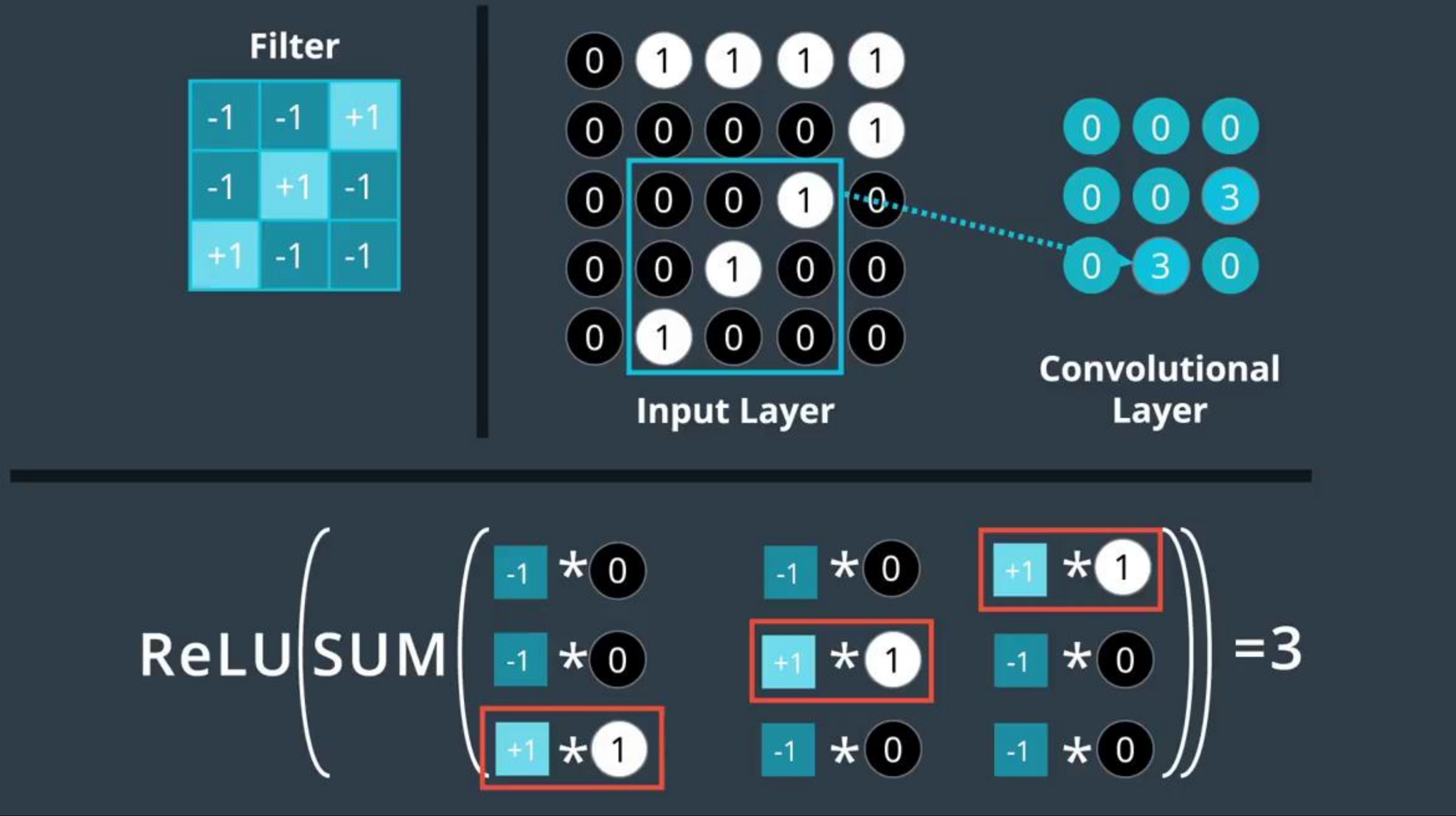

Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 -1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU(SUM(* * * * * * * * *))=0

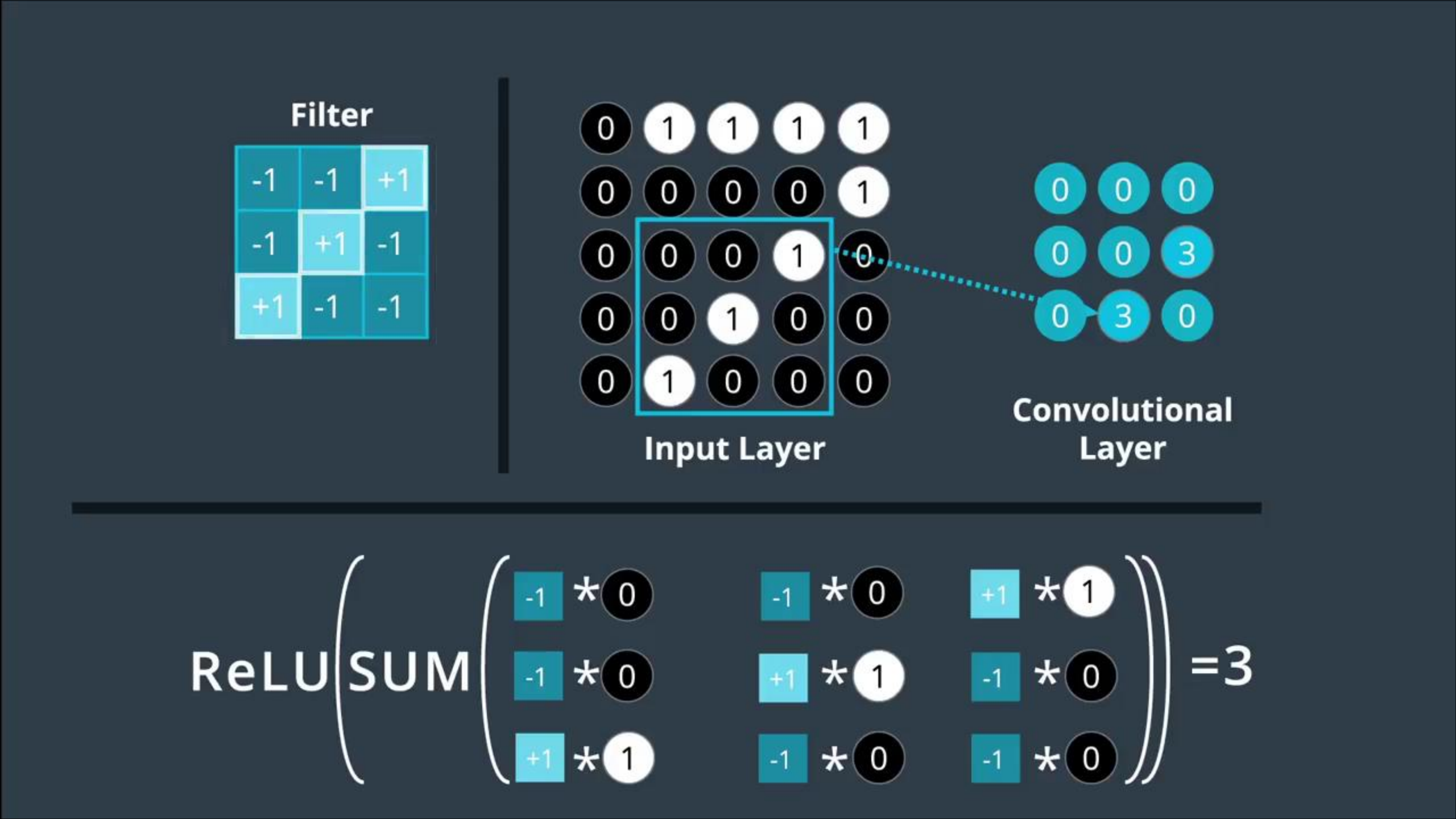

Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU(SUM(*0 *1 *1
*0 *0 *0
*0 *0 *0)) =0


Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 -1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM
-1 * 0
-1 * 1
+1 * 1
-1 * 0
+1 * 0
-1 * 0
+1 * 0
-1 * 0
-1 * 0
=0

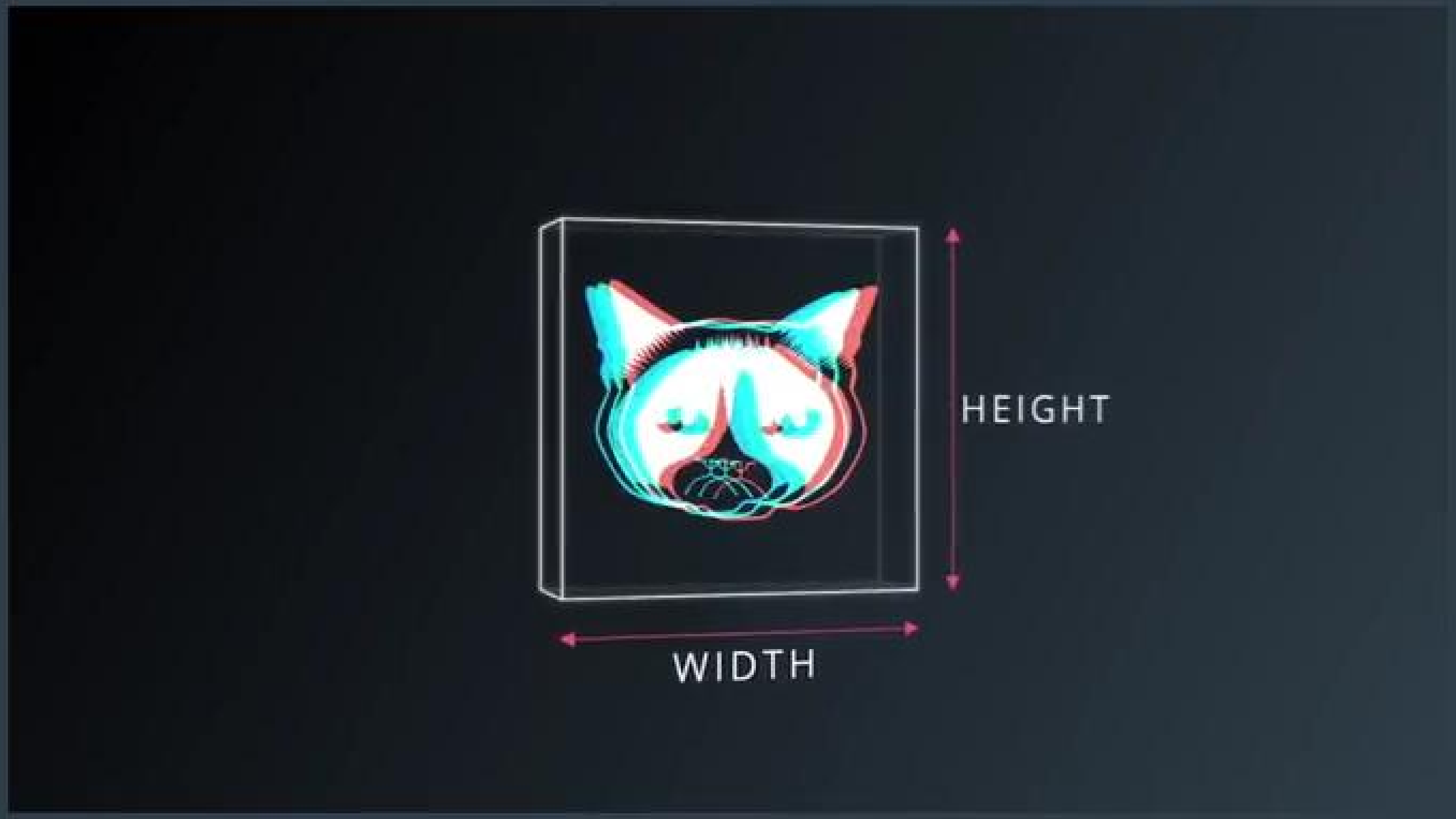

Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM
-1 * 0
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
-1 * 0
=3


Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM
-1 * 0
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
-1 * 0
=3


Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM
-1 * 0
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
-1 * 0
+1 * 1
-1 * 0
-1 * 0
=3


Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM
-1 * 0
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
-1 * 0
=3


HEIGHT
WIDTH


HEIGHT
WIDTH

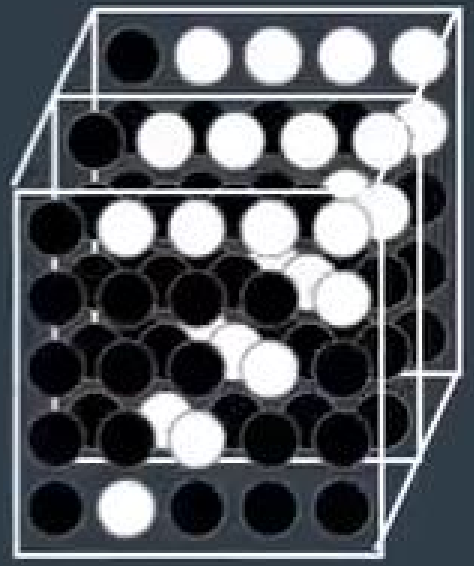

HEIGHT
WIDTH

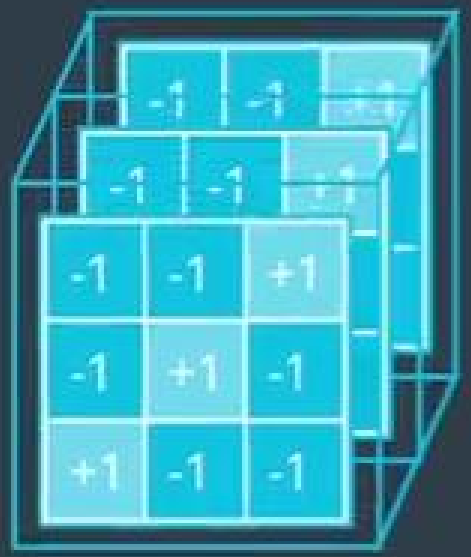

HEIGHT
WIDTH

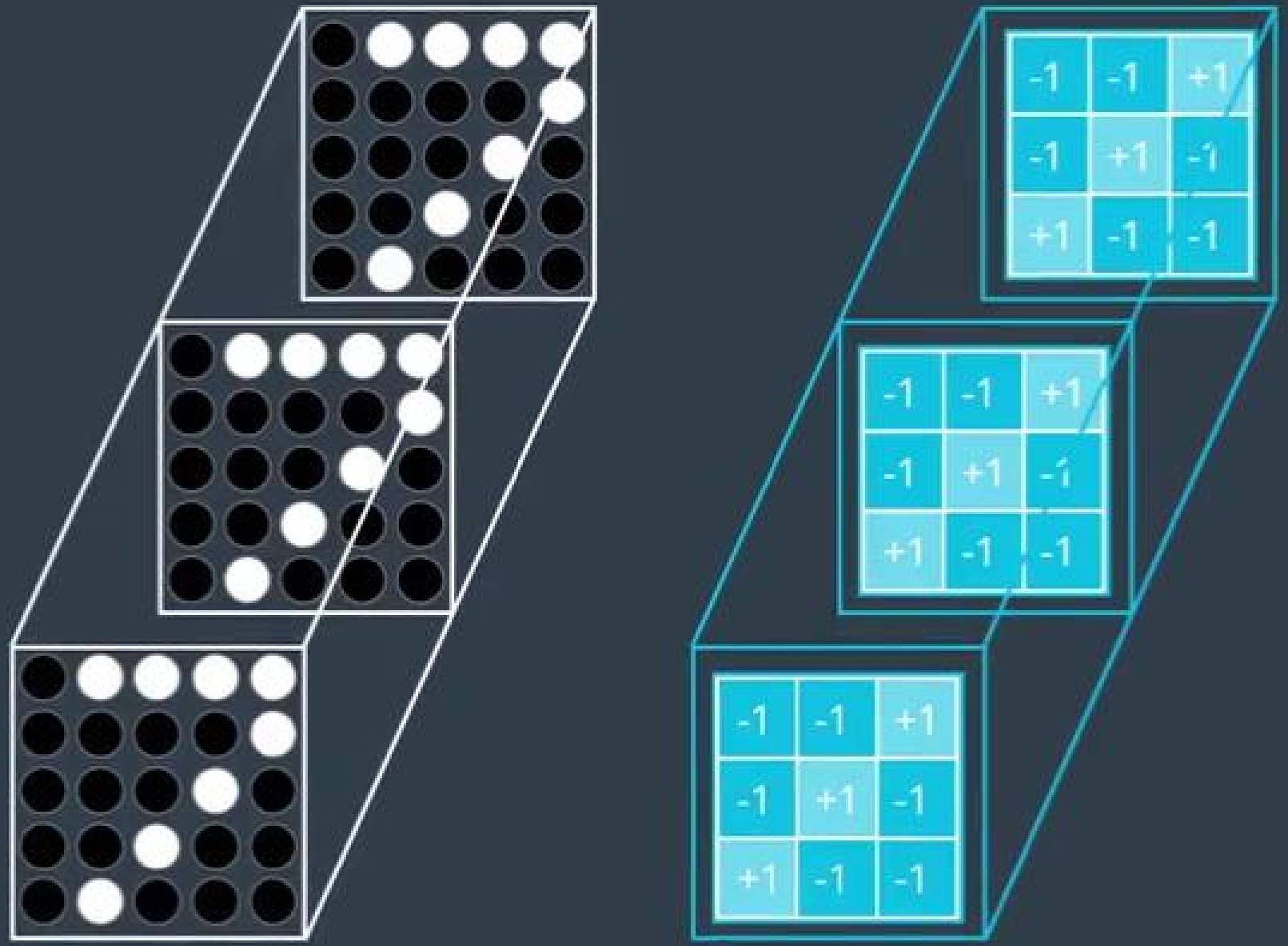

HEIGHT
WIDTH

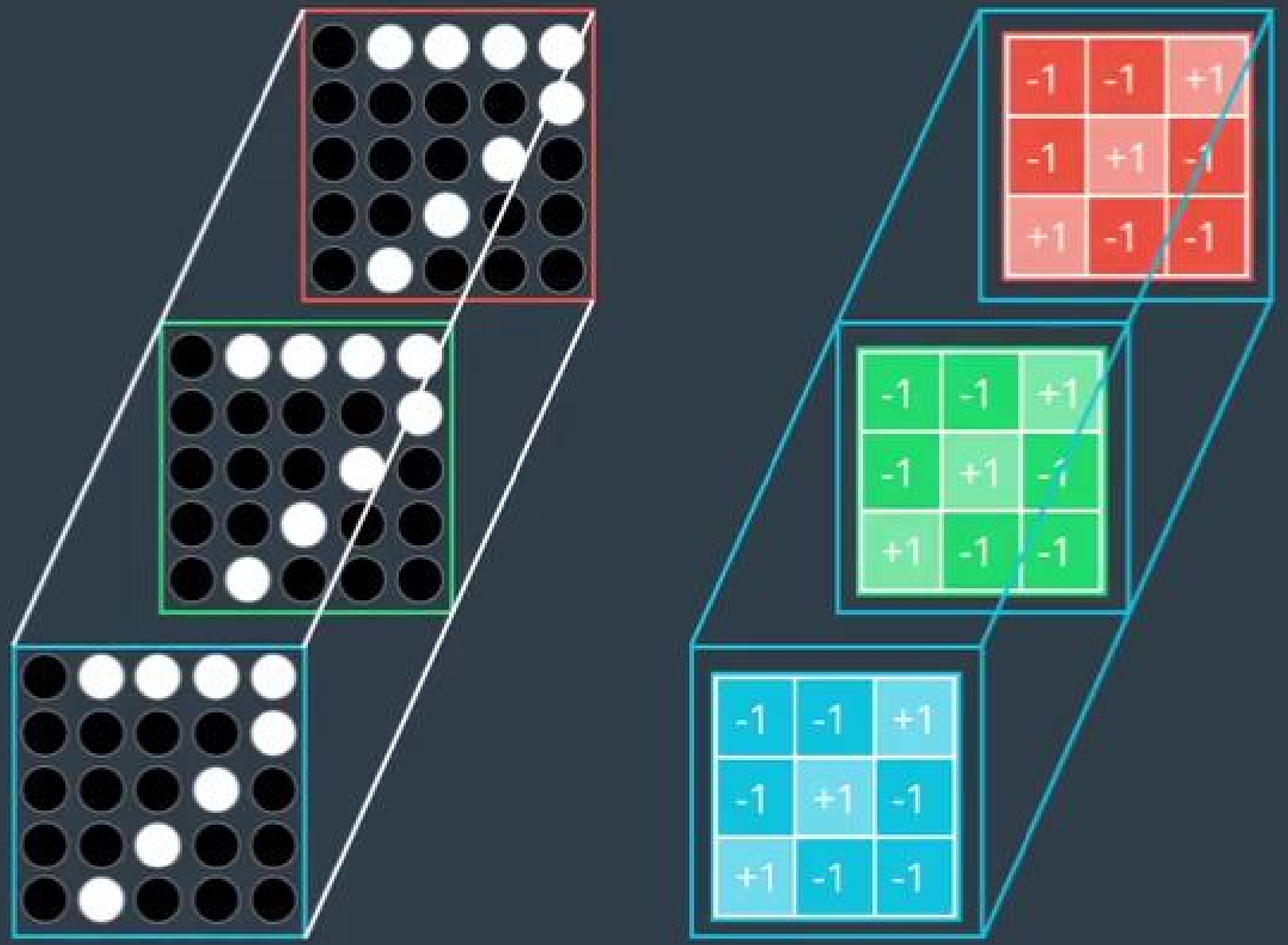

HEIGHT
DEPTH
WIDTH

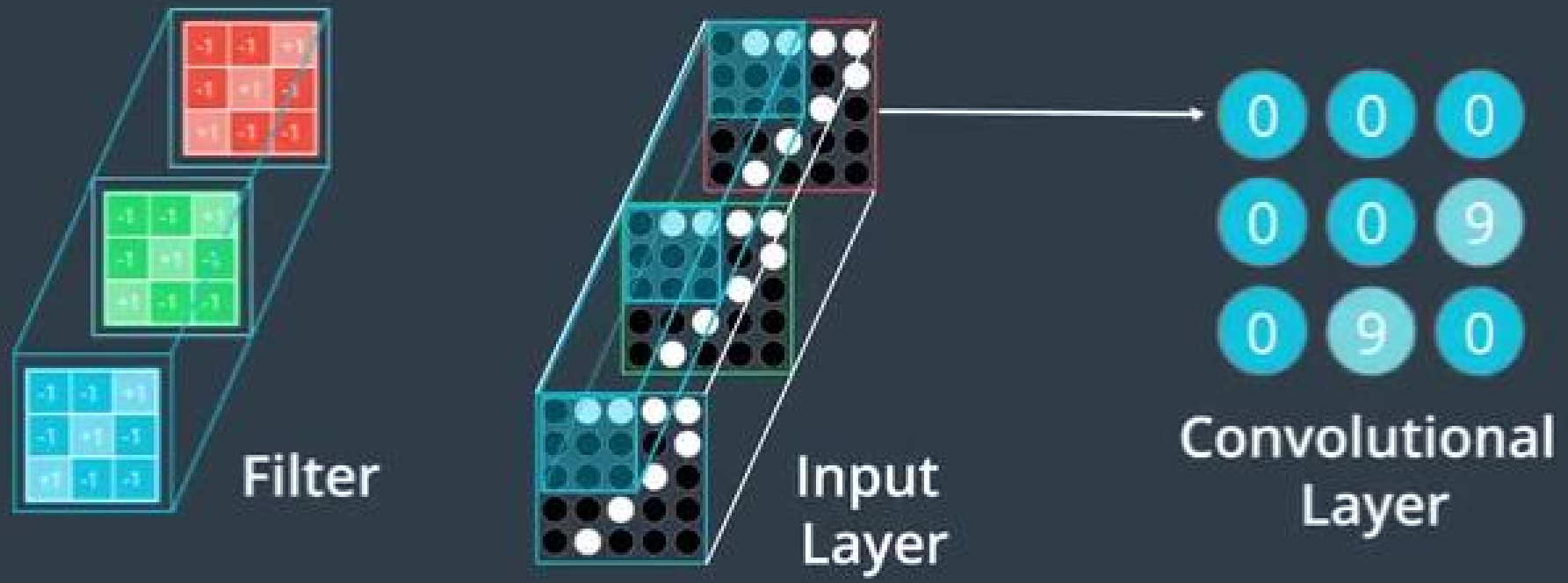
DEPTH

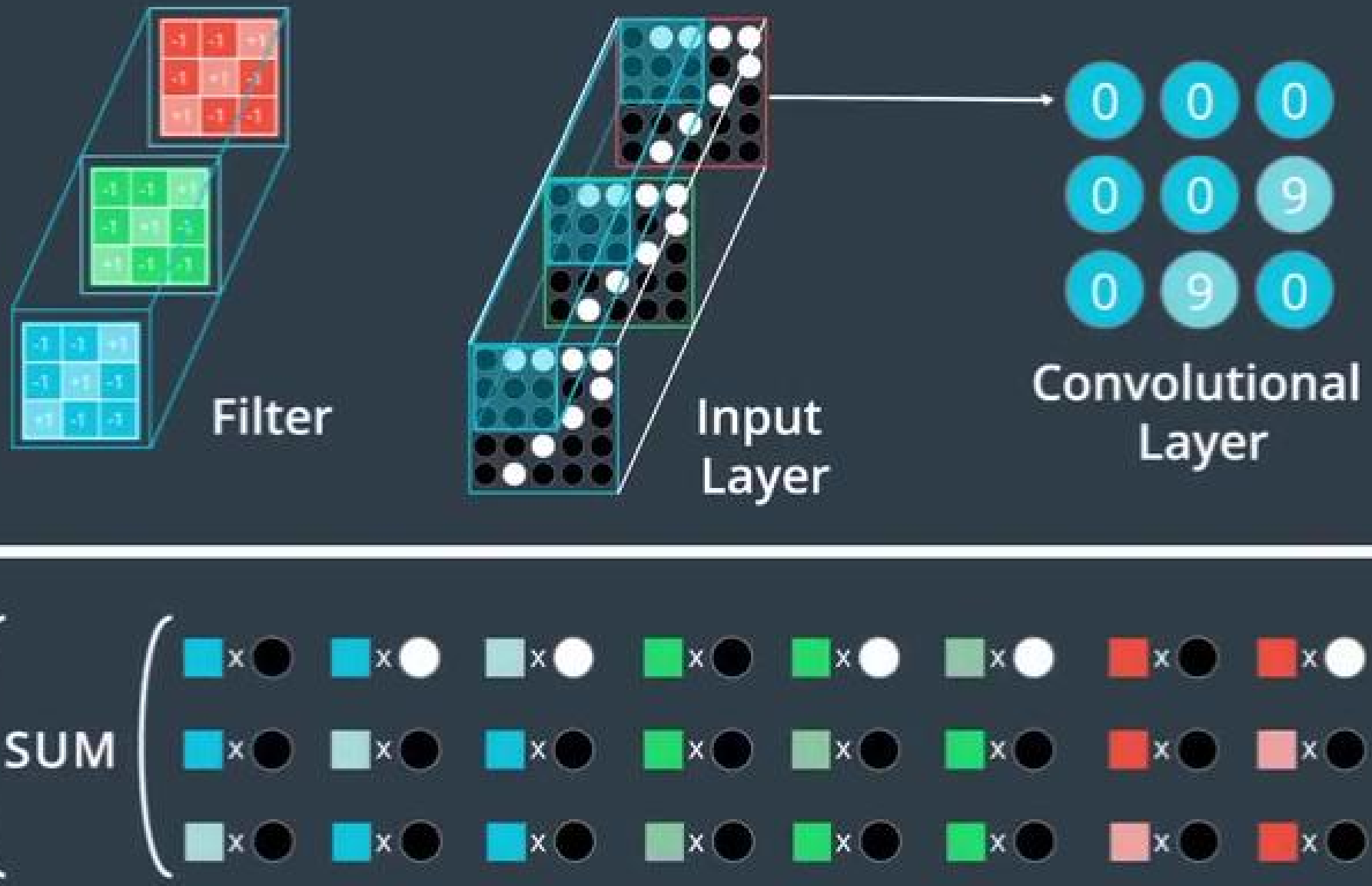


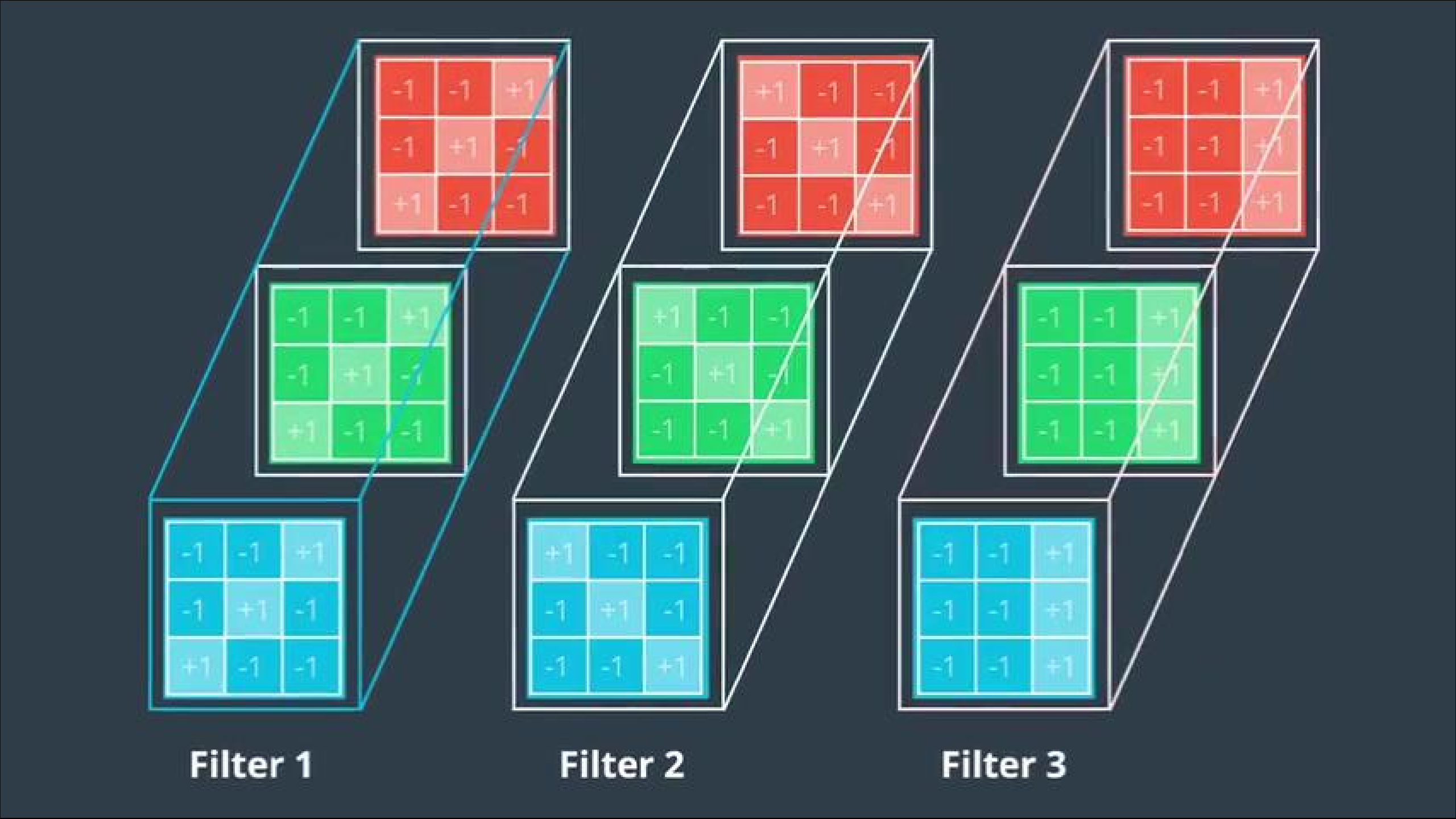

-1 -1
-1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1


-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1

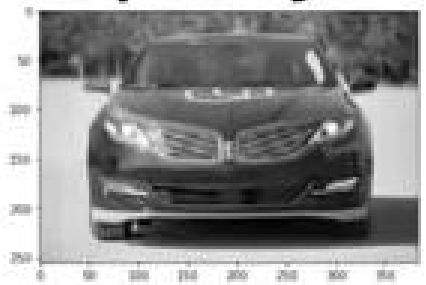

-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1


Filter
Input Layer
0 0 0
0 0 9
0 9 0
Convolutional Layer

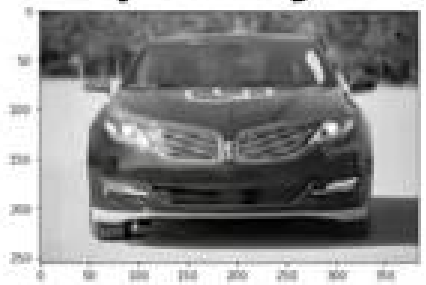

Filter
Input Layer
0 0 0
0 0 9
0 9 0
Convolutional Layer
ReLU
SUM


-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1
+1 -1 -1
-1 +1 -1
-1 -1 +1
+1 -1 -1
-1 +1 -1
-1 -1 +1
+1 -1 -1
-1 +1 -1
-1 -1 +1
-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
+1 -1 -1
-1 -1 +1
-1 +1 -1
-1 -1 +1
Filter 1
Filter 2
Filter 3

Input Layer



Input Layer



Filter 1

-1	-1	+1	+1
-1	-1	+1	+1
-1	-1	+1	+1
-1	-1	+1	+1

Filter 2

+1	+1	-1	-1
+1	+1	-1	-1
+1	+1	-1	-1
+1	+1	-1	-1

Filter 3

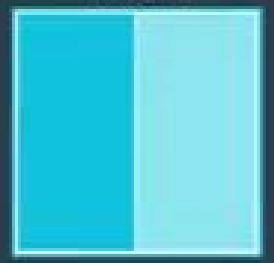
-1	-1	-1	-1
-1	-1	-1	-1
+1	+1	+1	+1
+1	+1	+1	+1

Filter 4

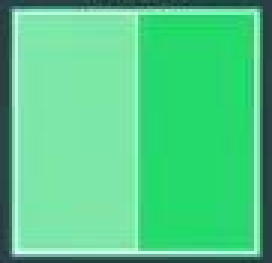
+1	+1	+1	+1
+1	+1	+1	+1
-1	-1	-1	-1
-1	-1	-1	-1

Input Layer

Filter 1

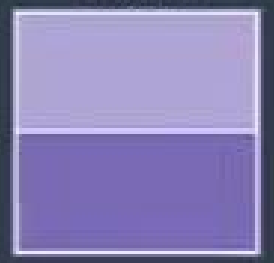


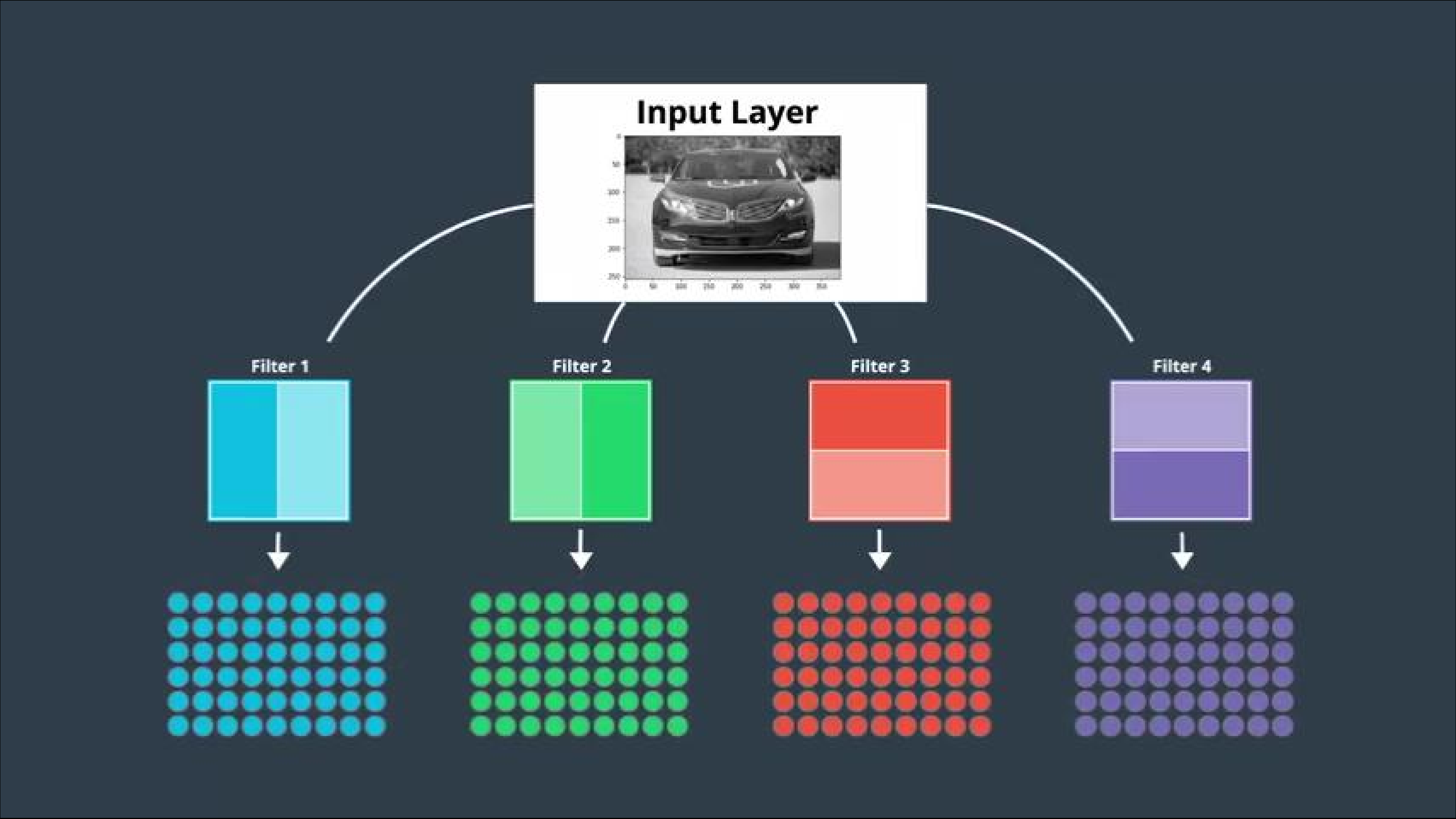
Filter 2

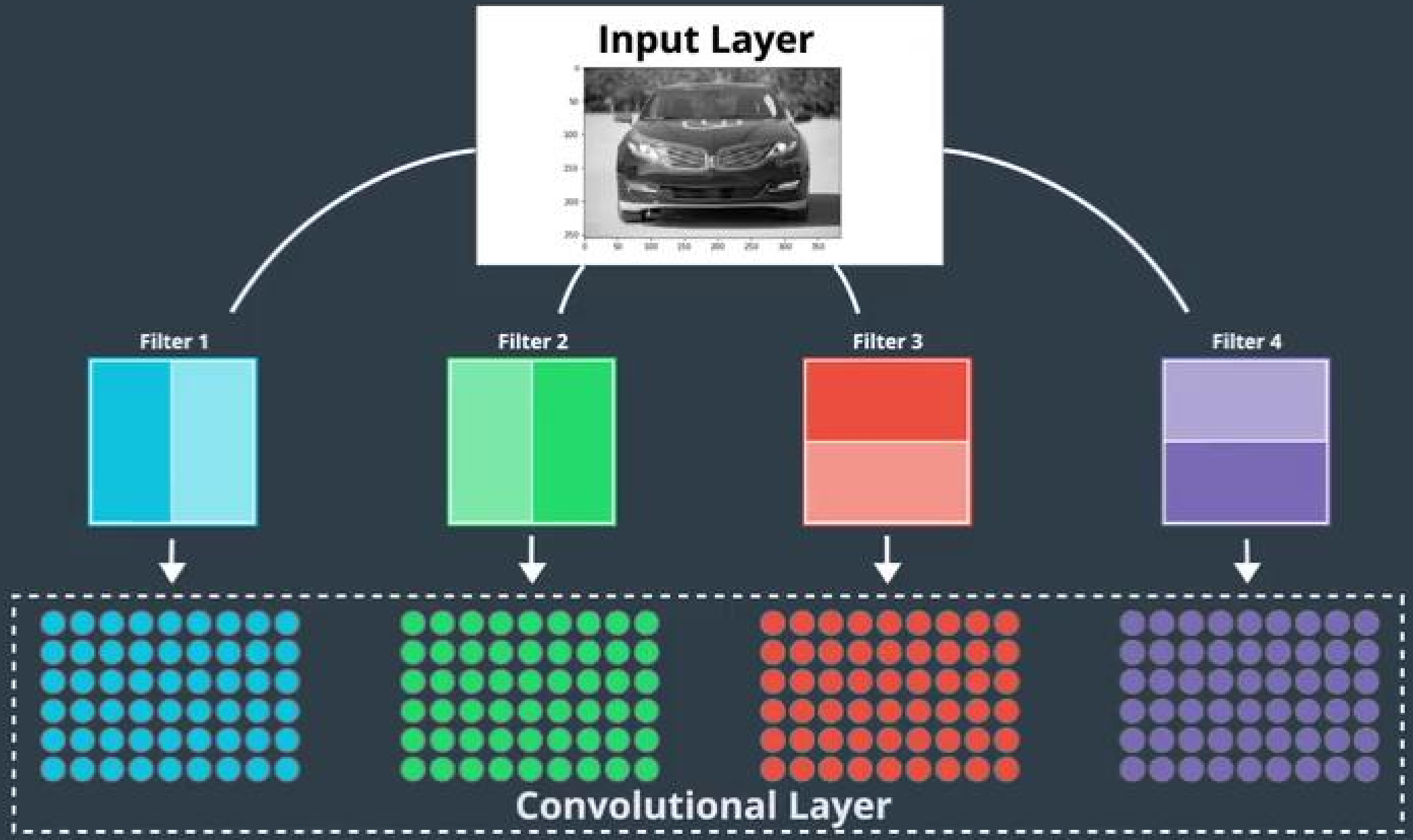


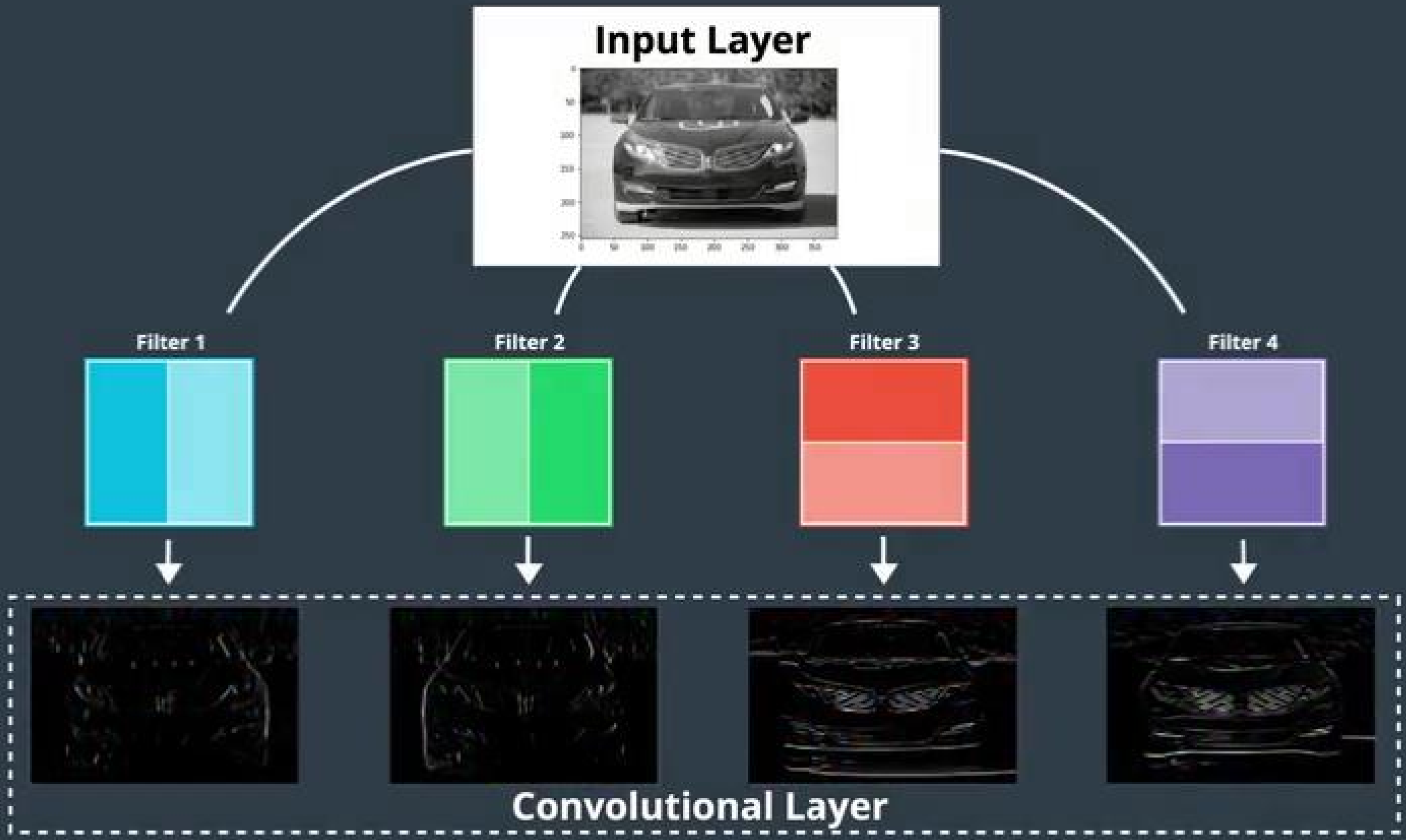
Filter 3

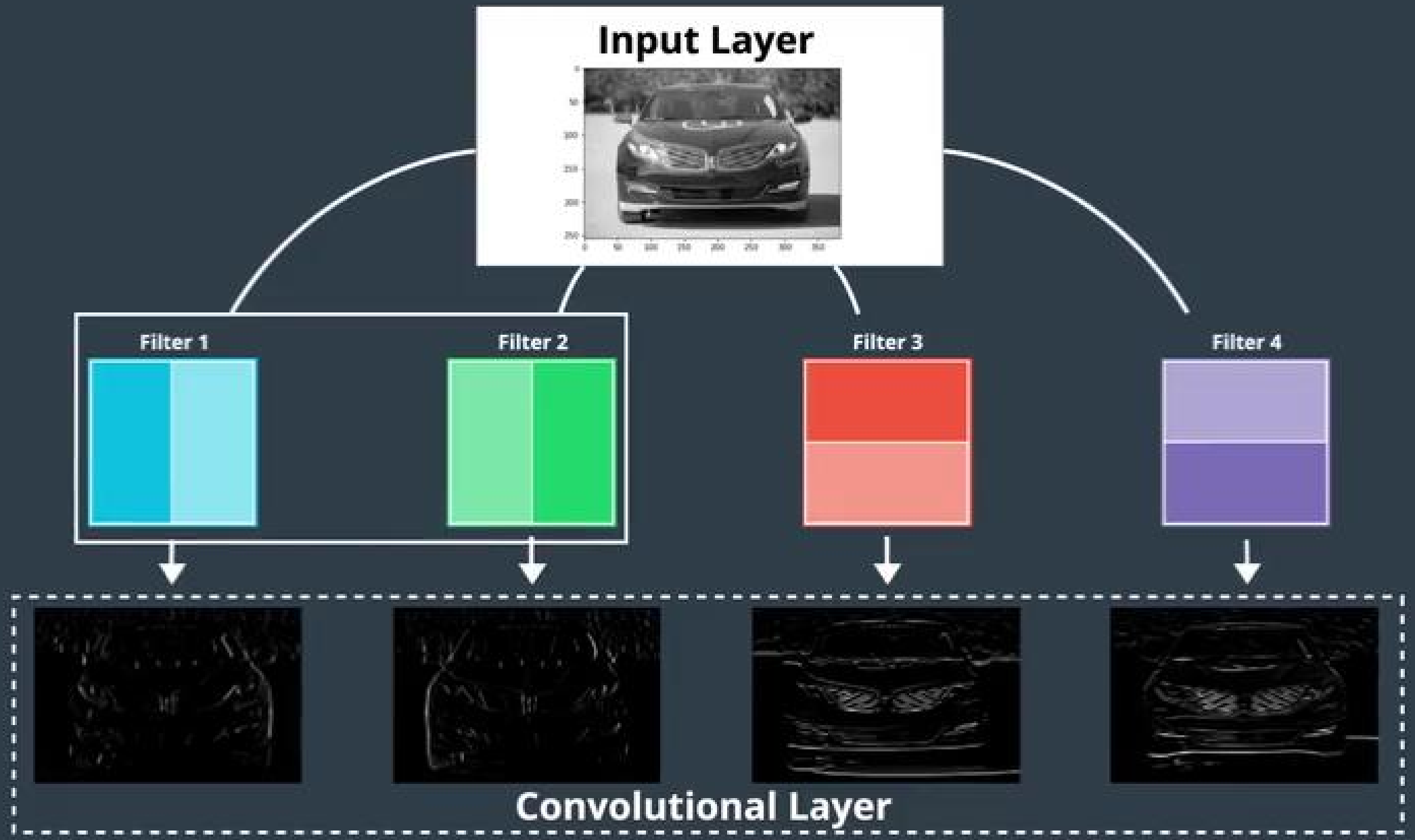
Filter 4

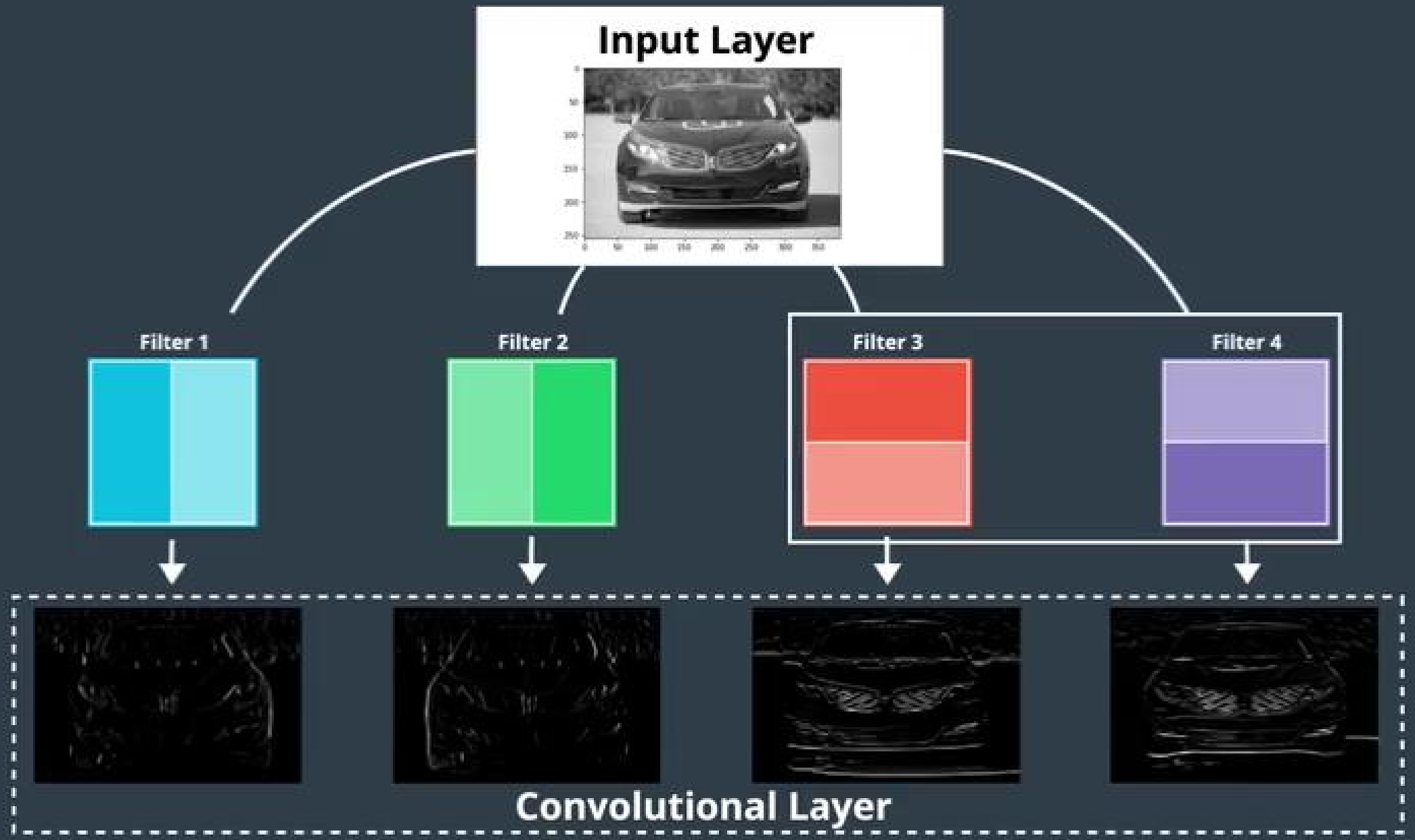


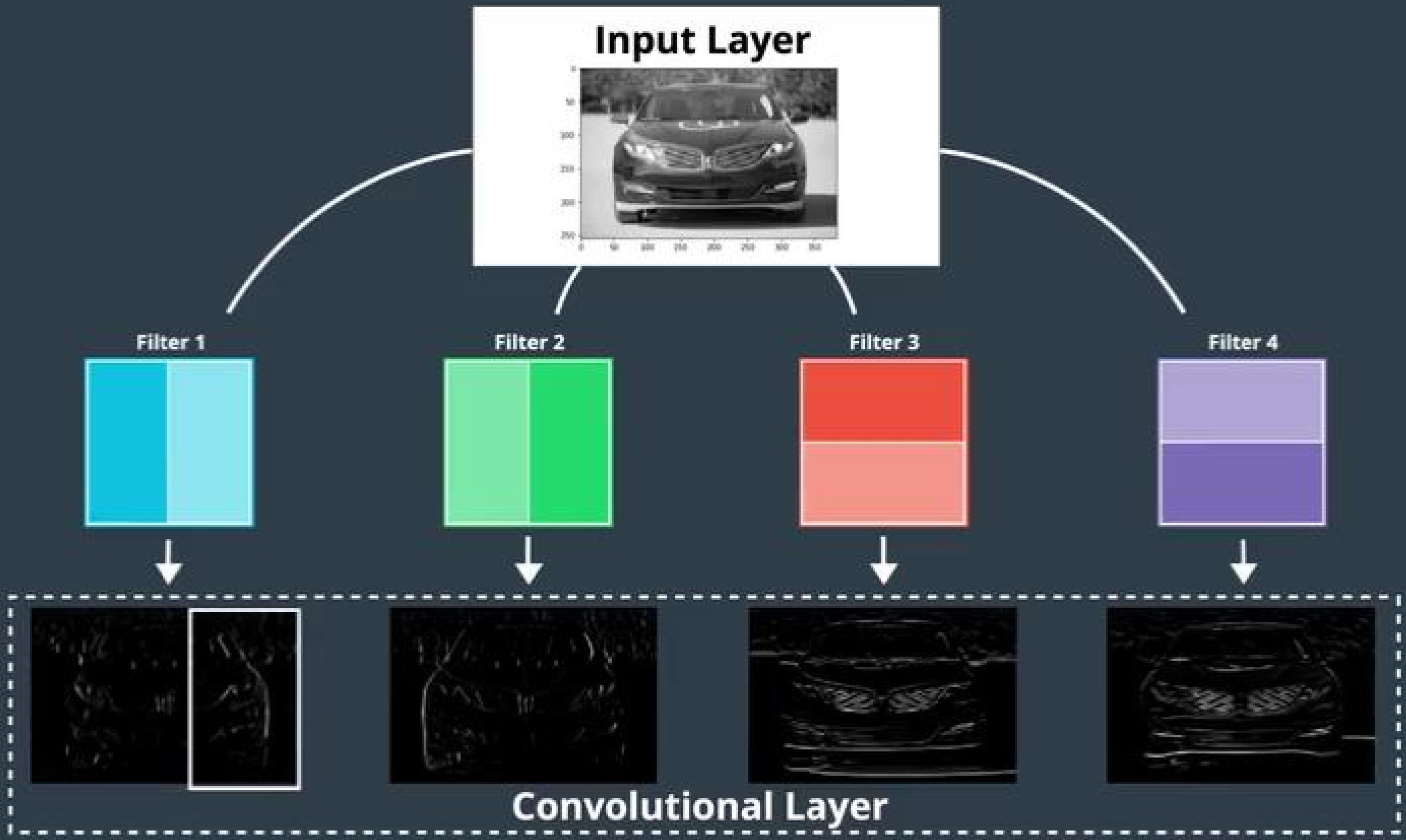

Input Layer
Filter 1
Filter 2
Filter 3
Filter 4

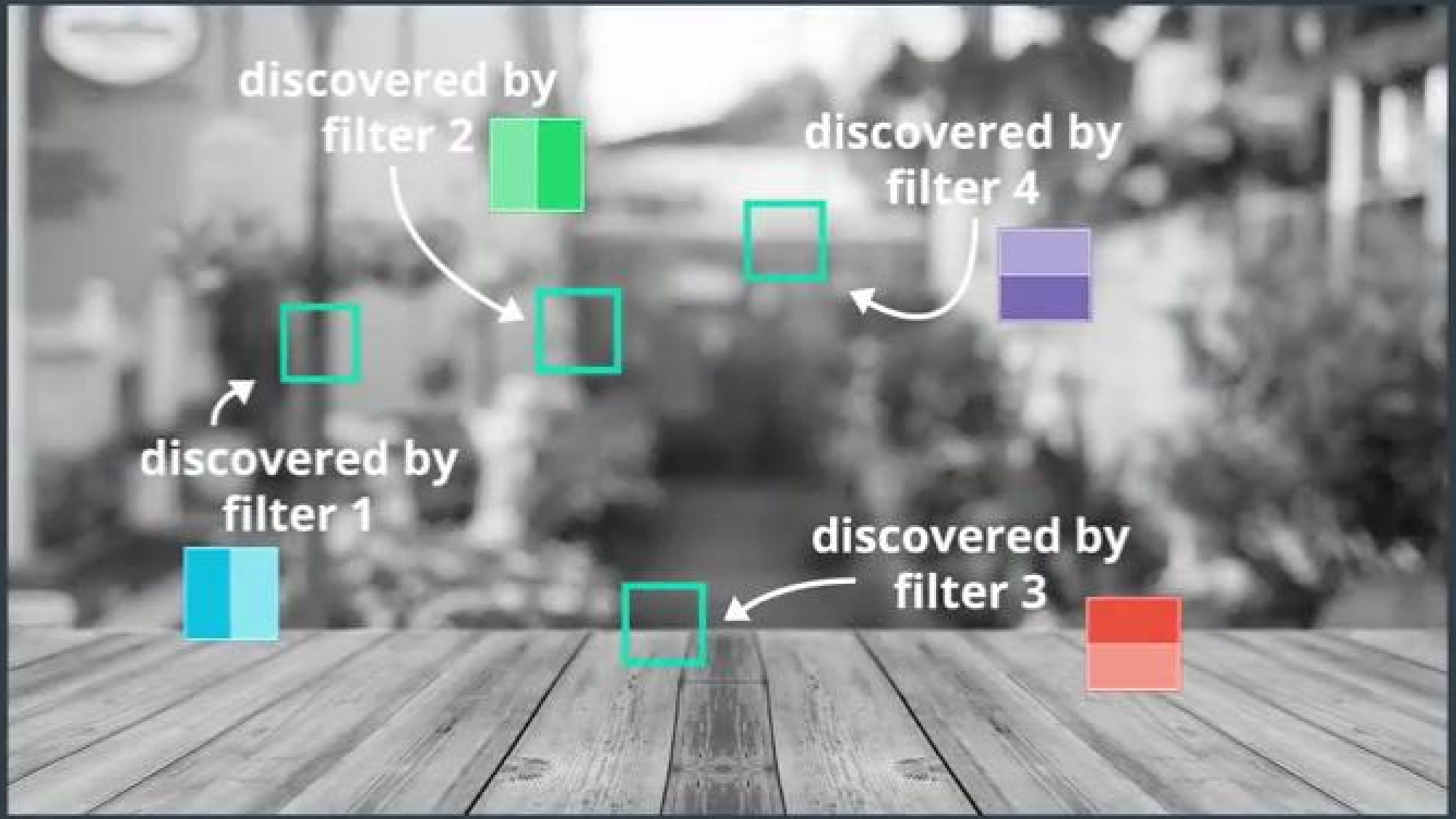

Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer

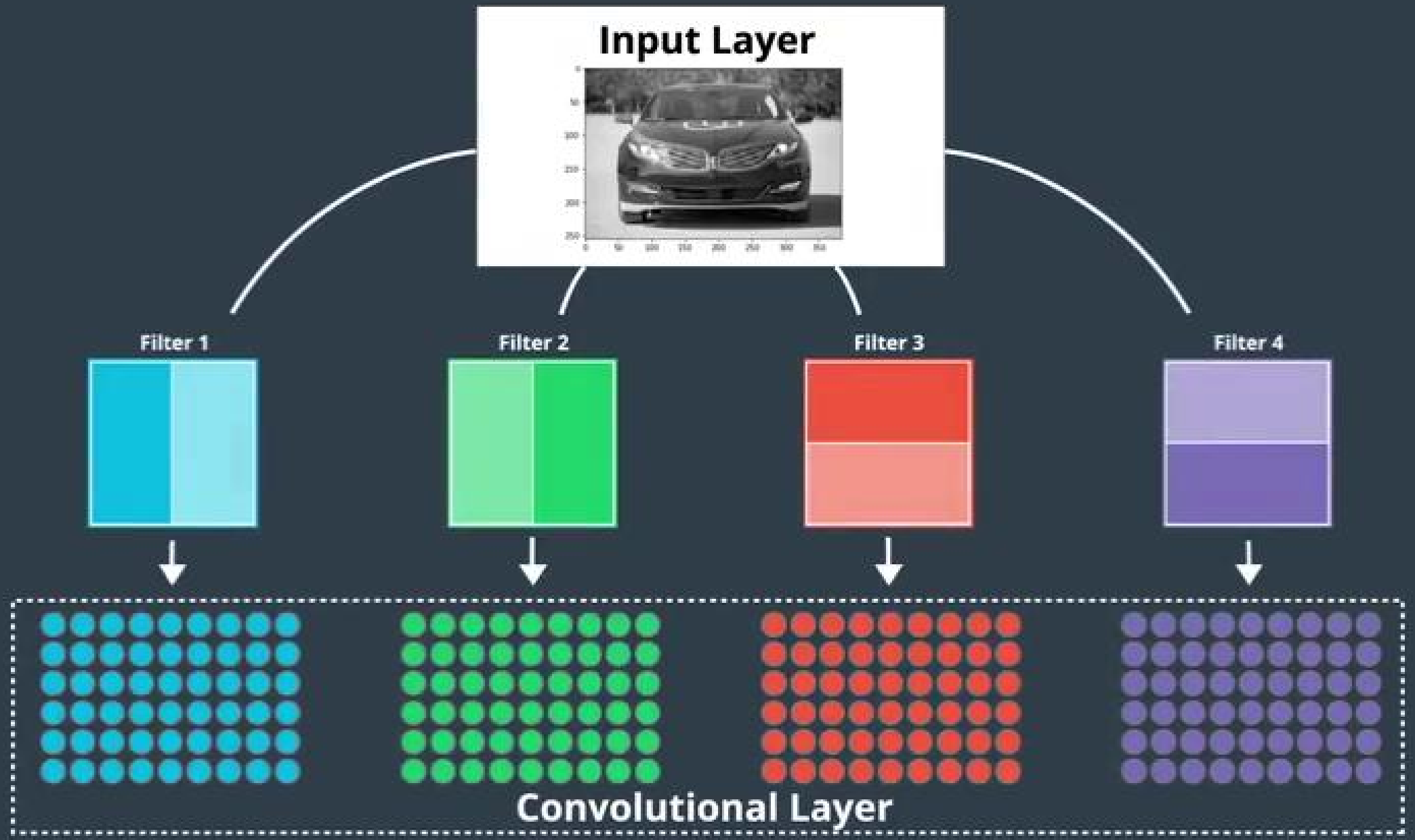

Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer

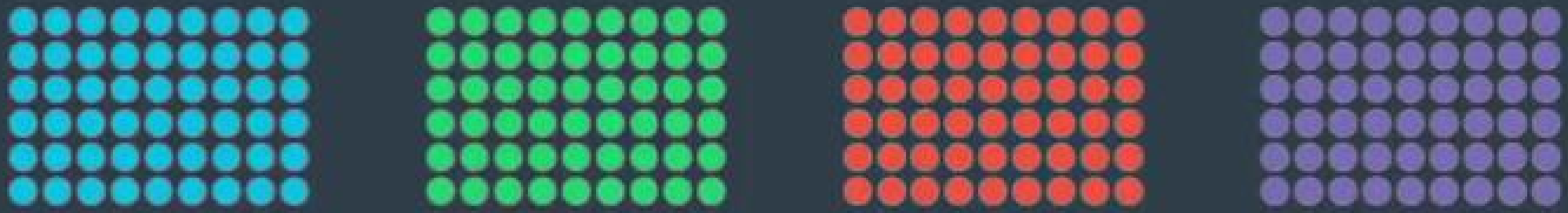

Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer

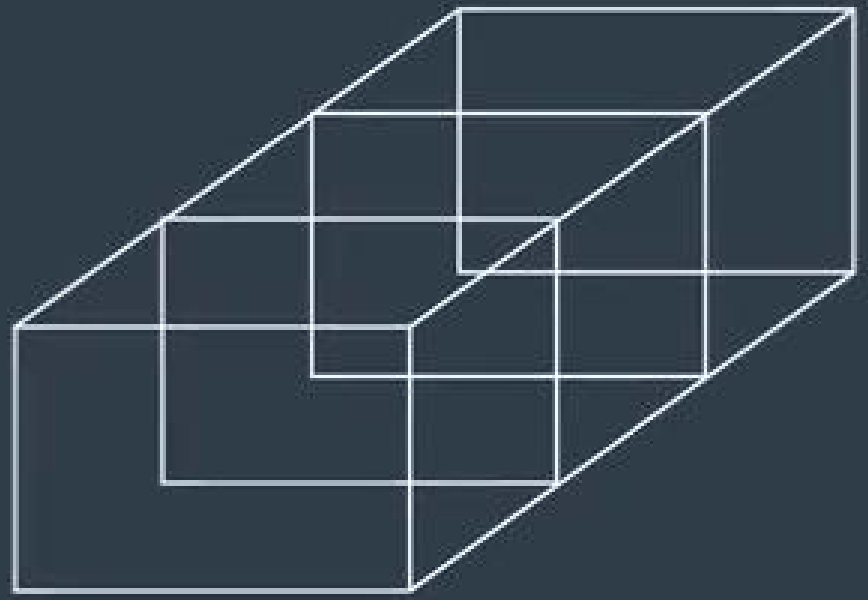

Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer

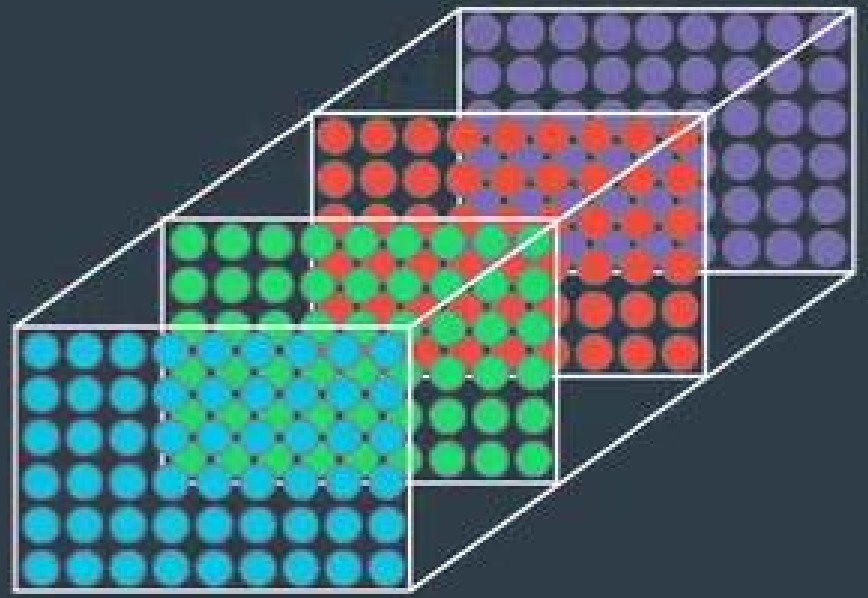

Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer

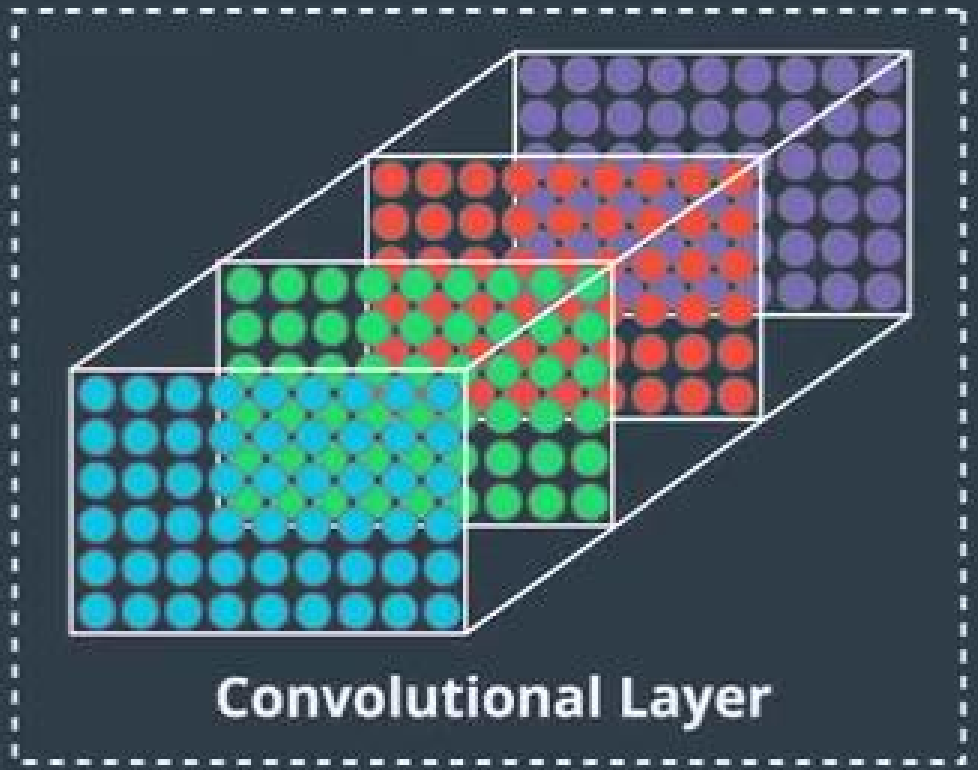

discovered by
filter 2
discovered by
filter 4
discovered by
filter 1
discovered by
filter 3

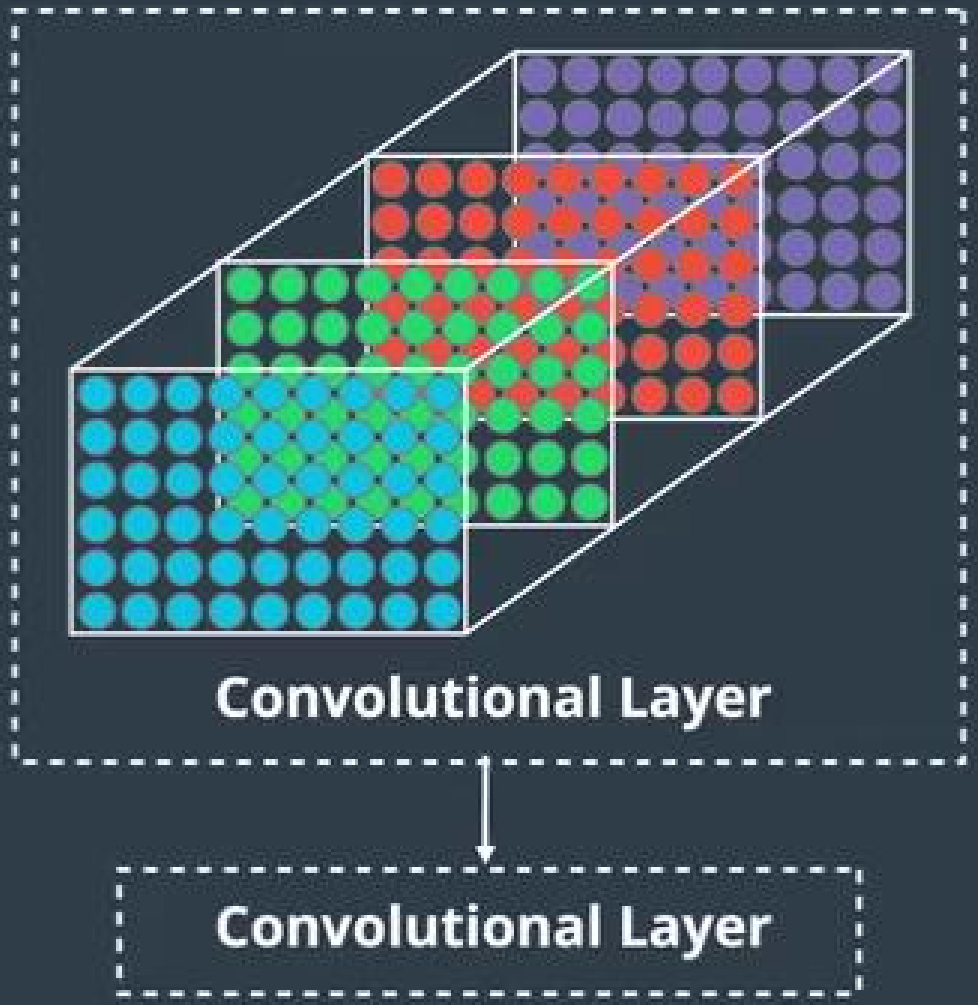

Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer

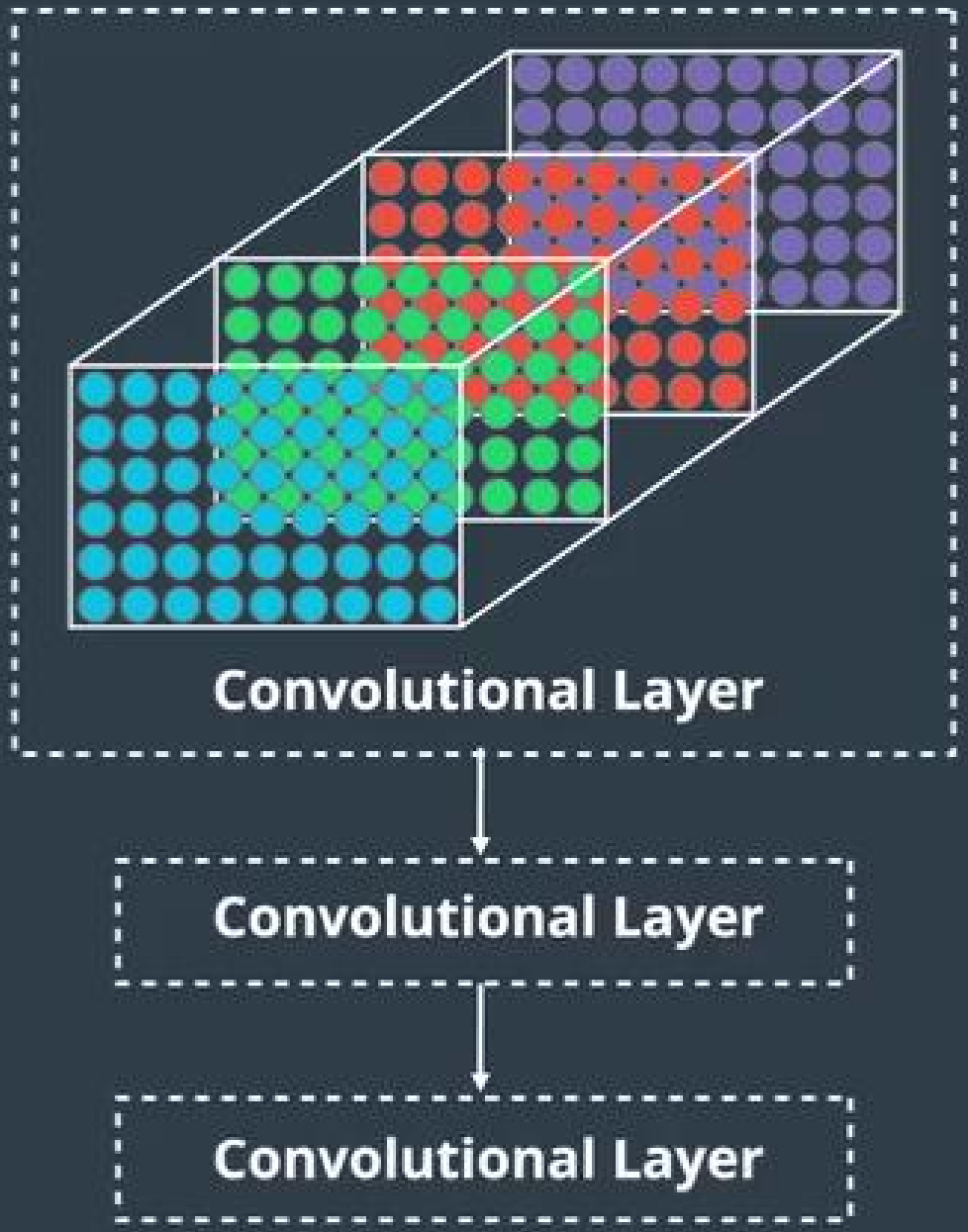




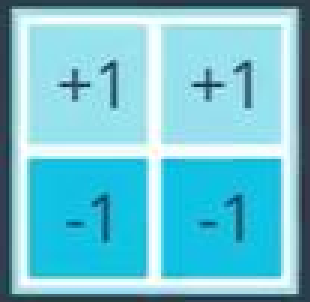



Convolutional Layer


Convolutional Layer
Convolutional Layer

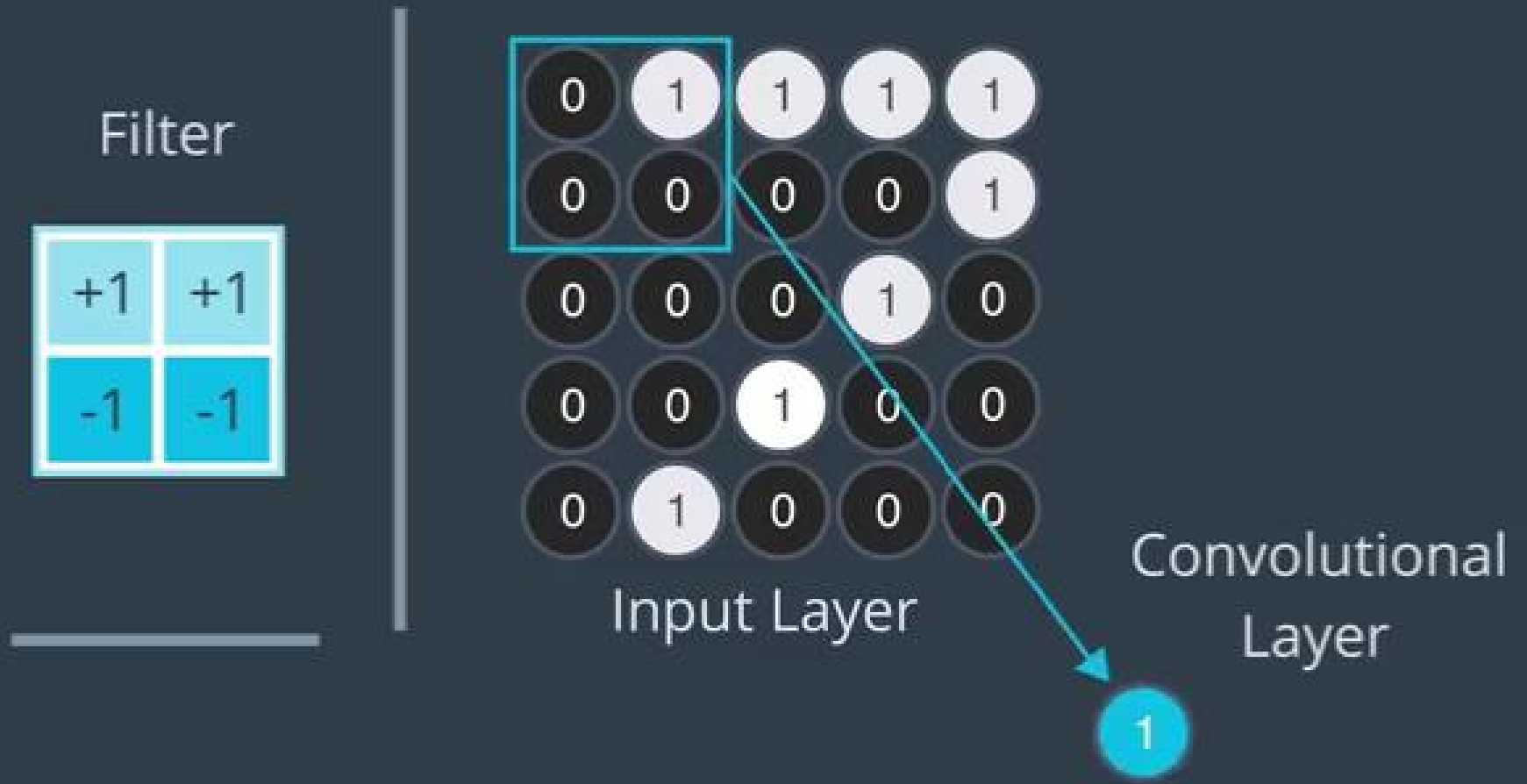

Convolutional Layer
Convolutional Layer
Convolutional Layer

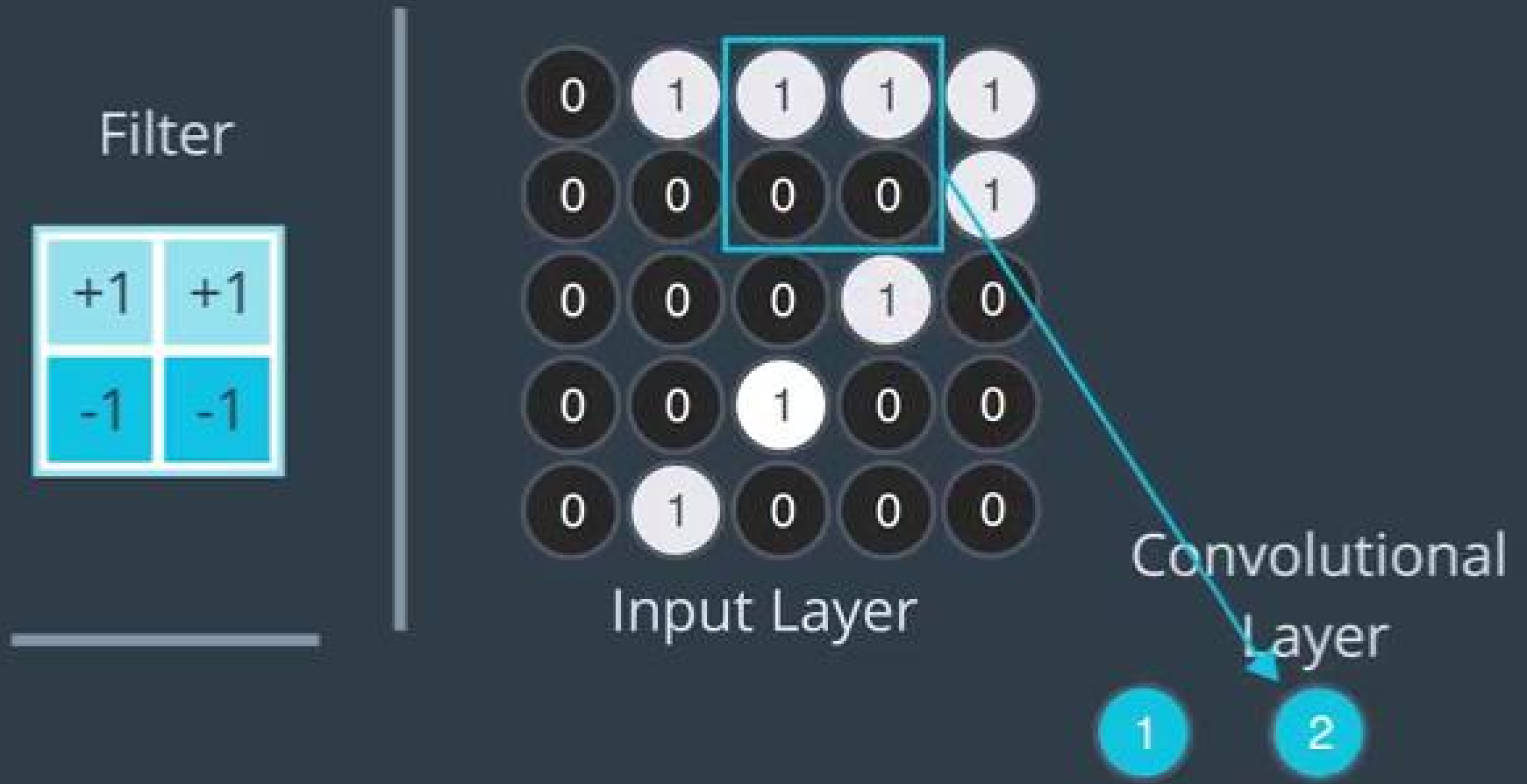
Filter

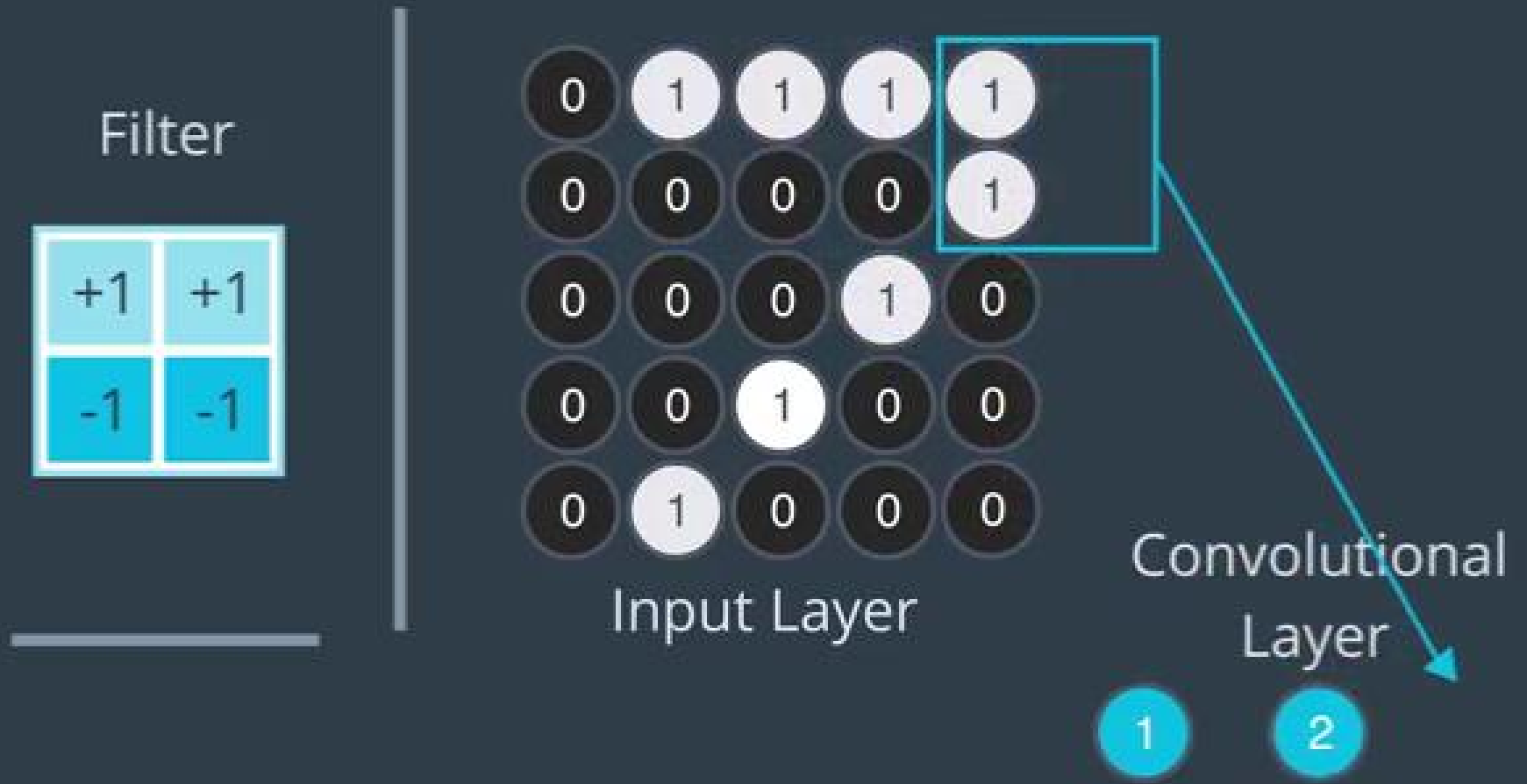


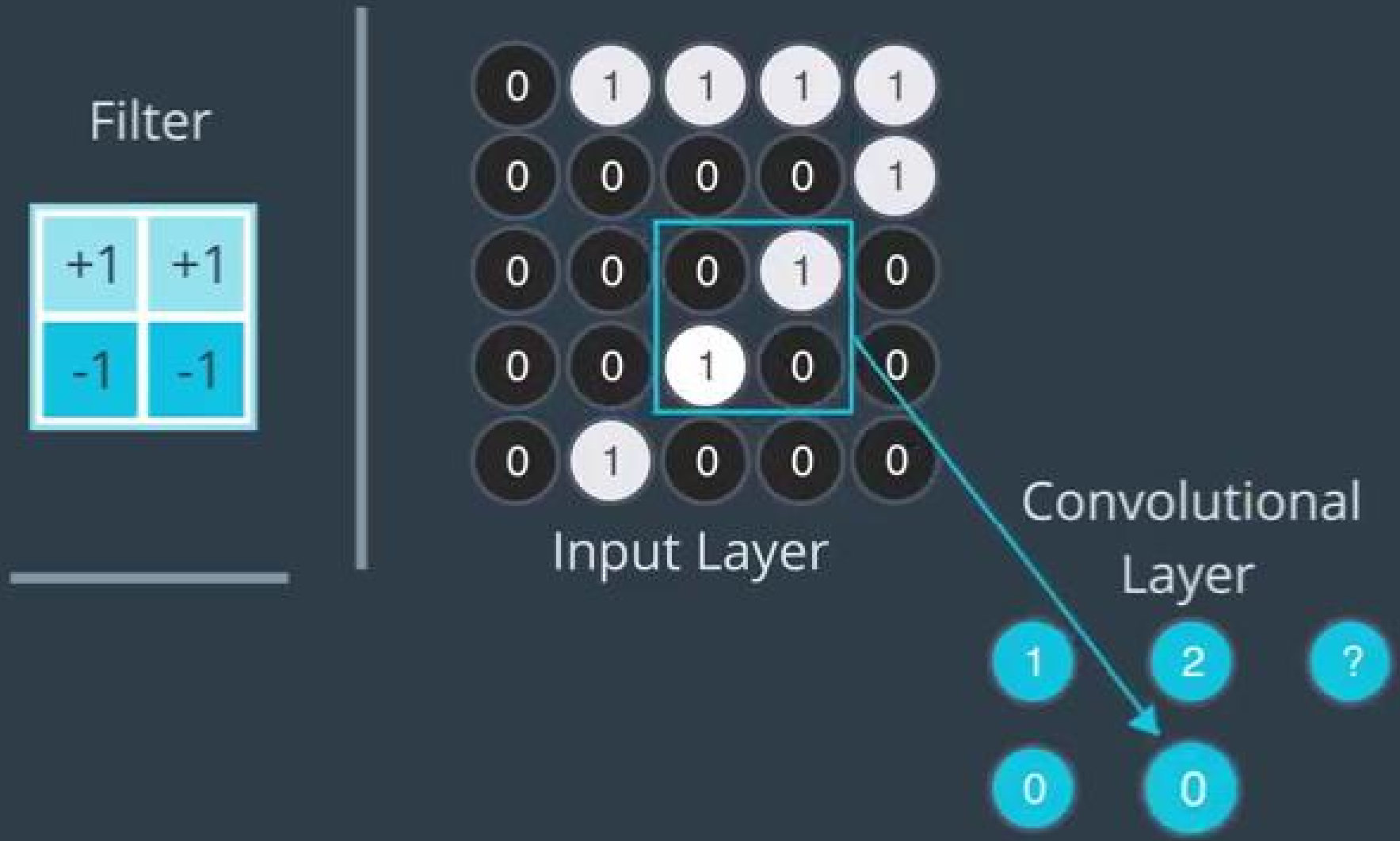
Input Layer

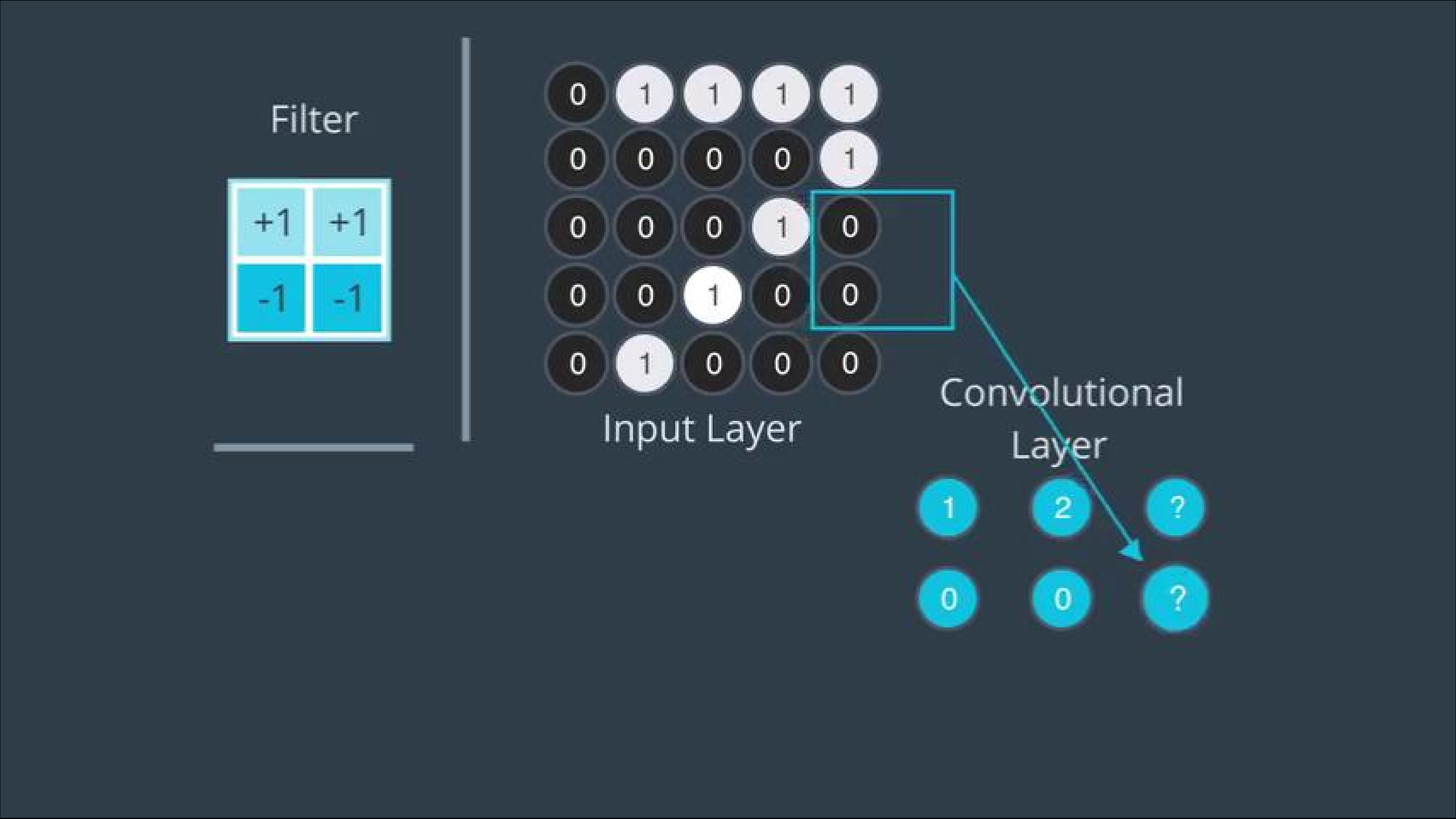
Convolutional Layer

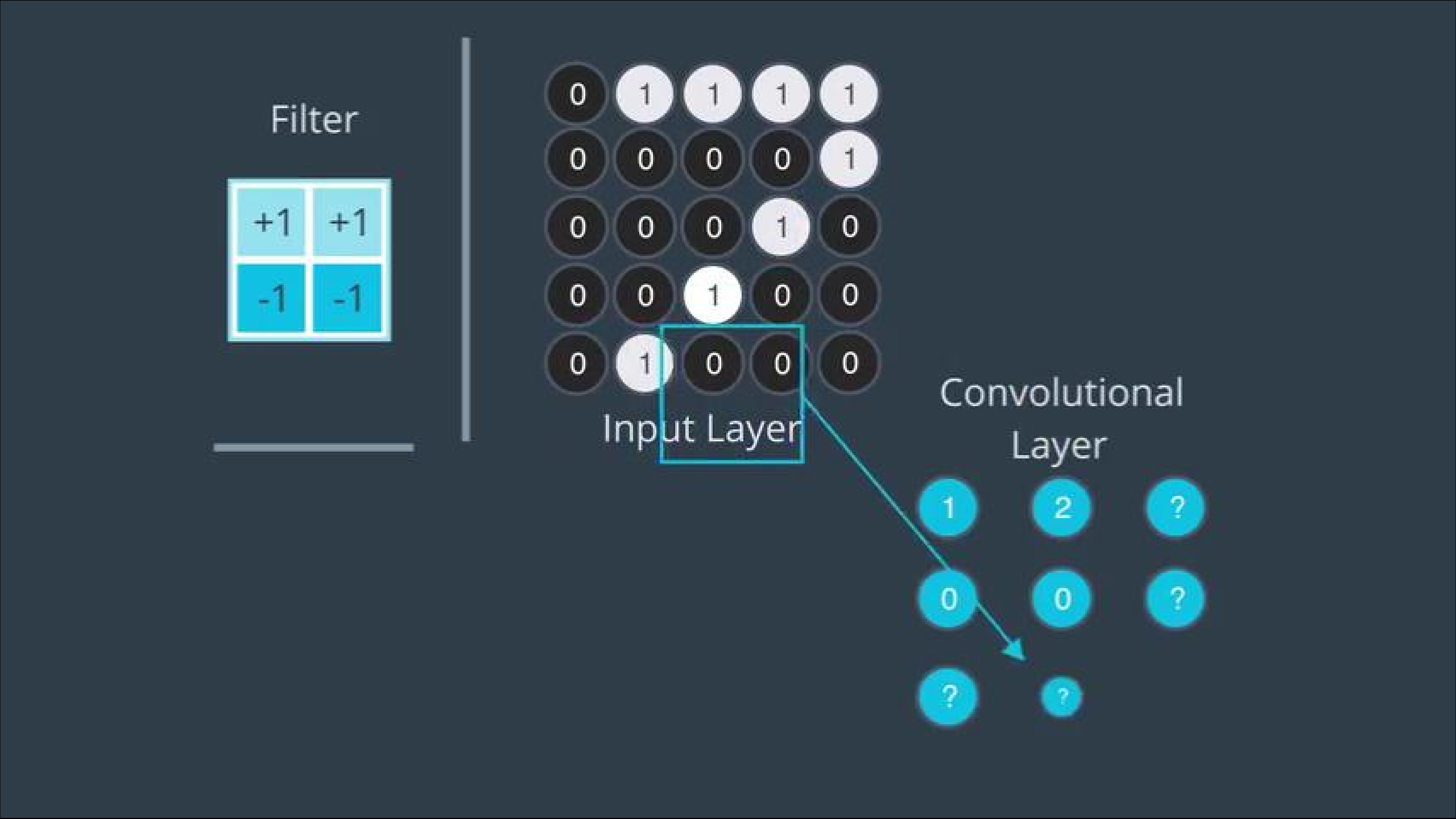

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
1
0
0
Input Layer
Convolutional Layer
1

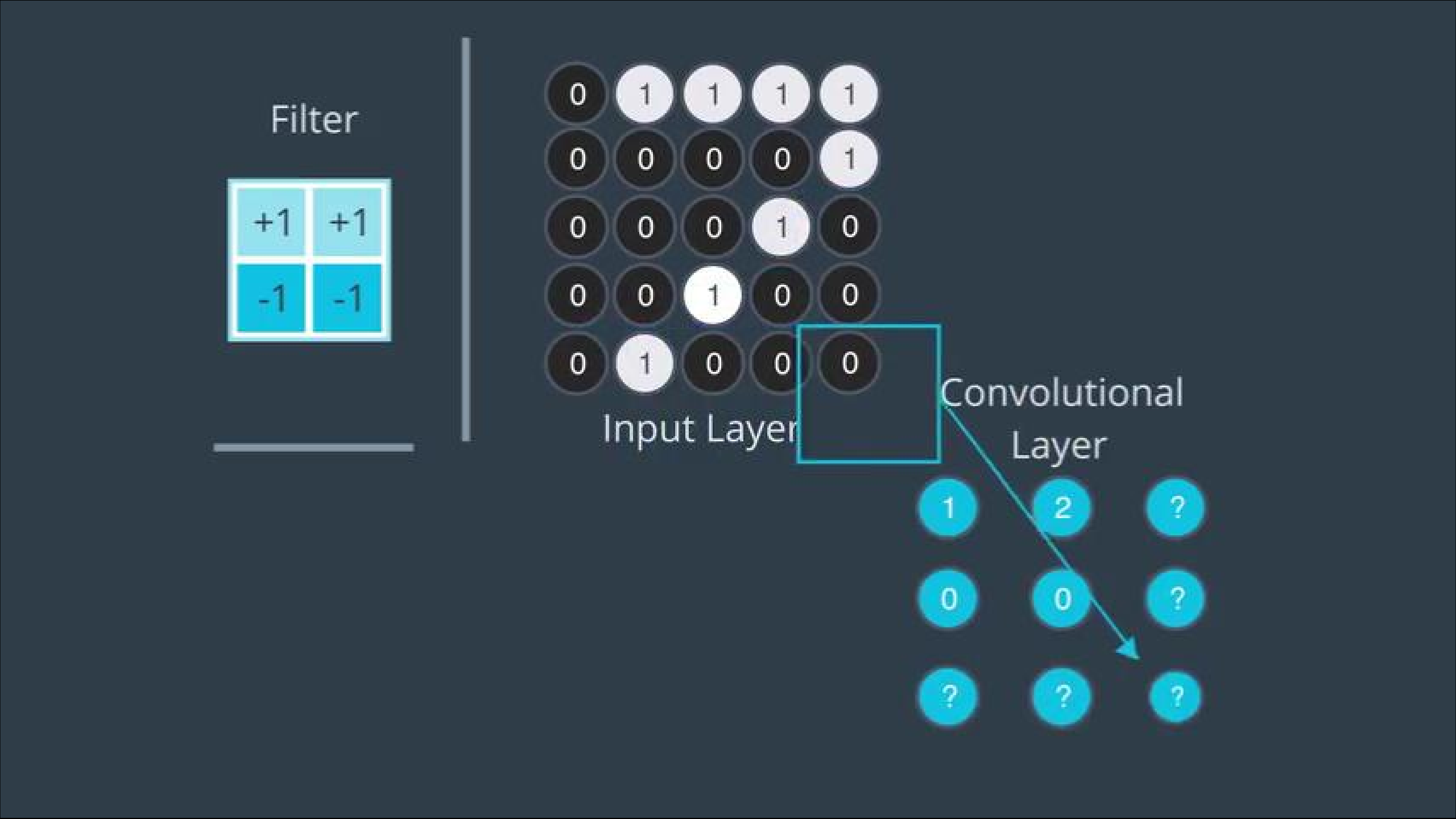

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
1
0
0
Input Layer
Convolutional Layer
1
2

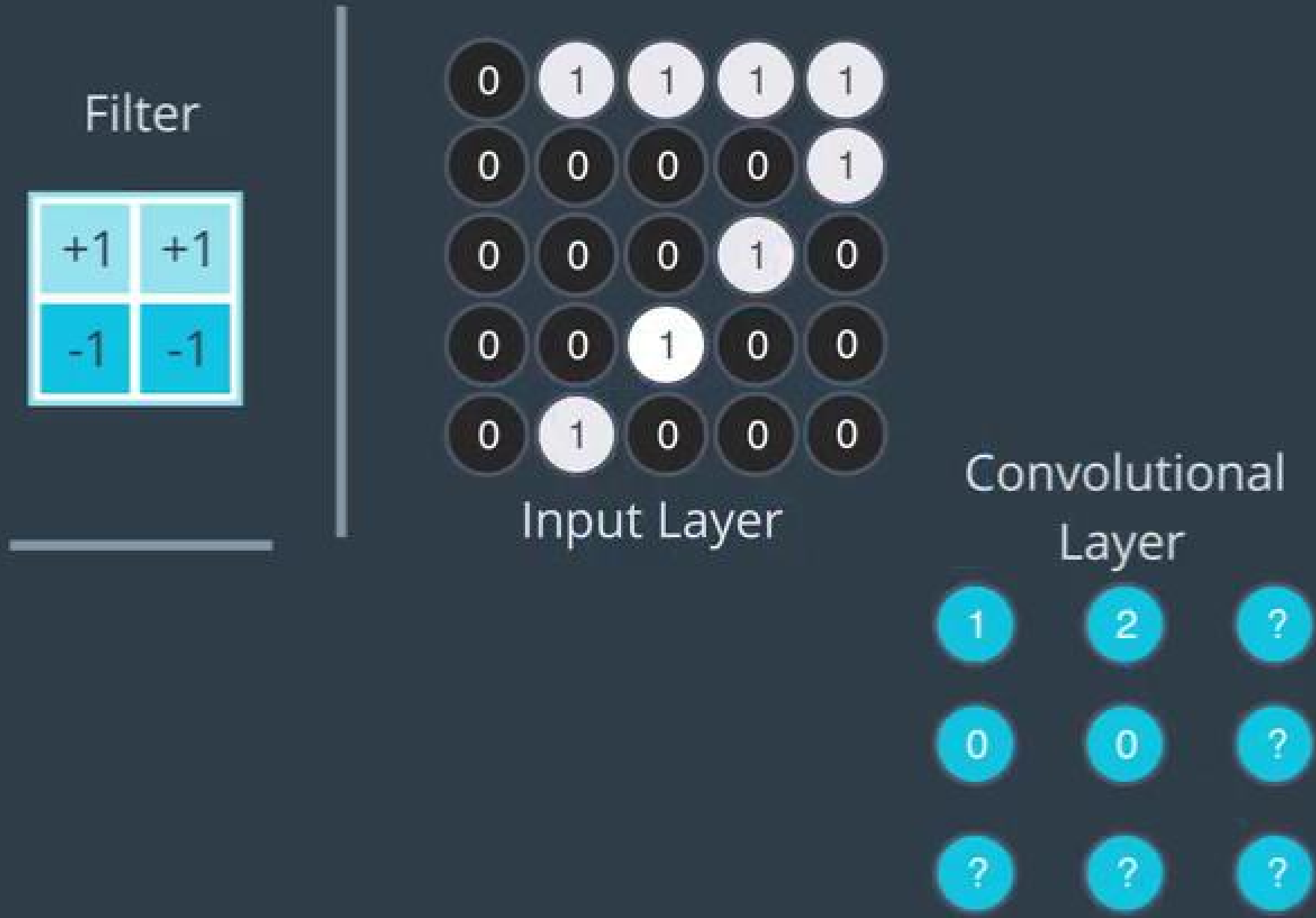

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
0
1
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2

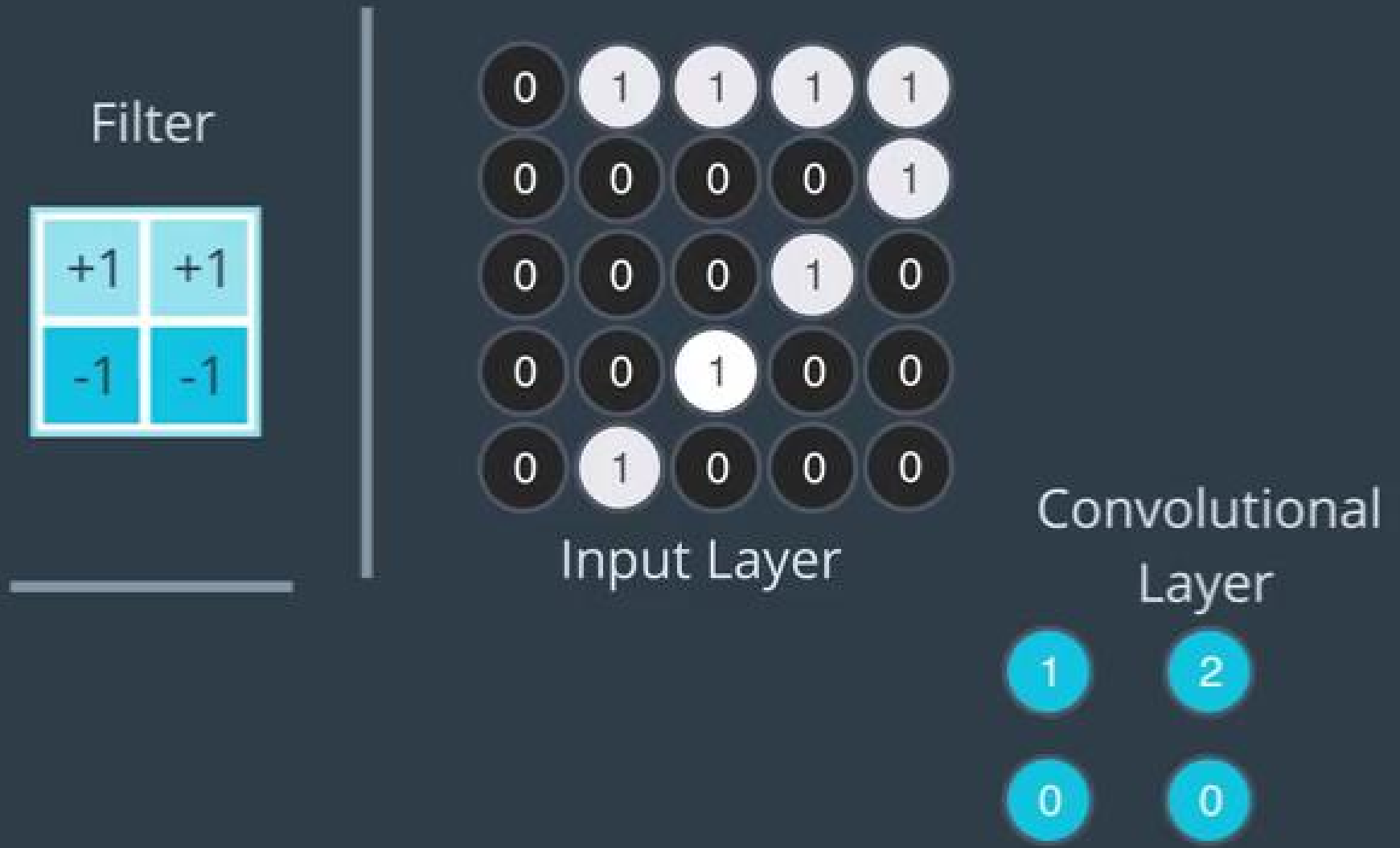

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
1
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2
?
0
0

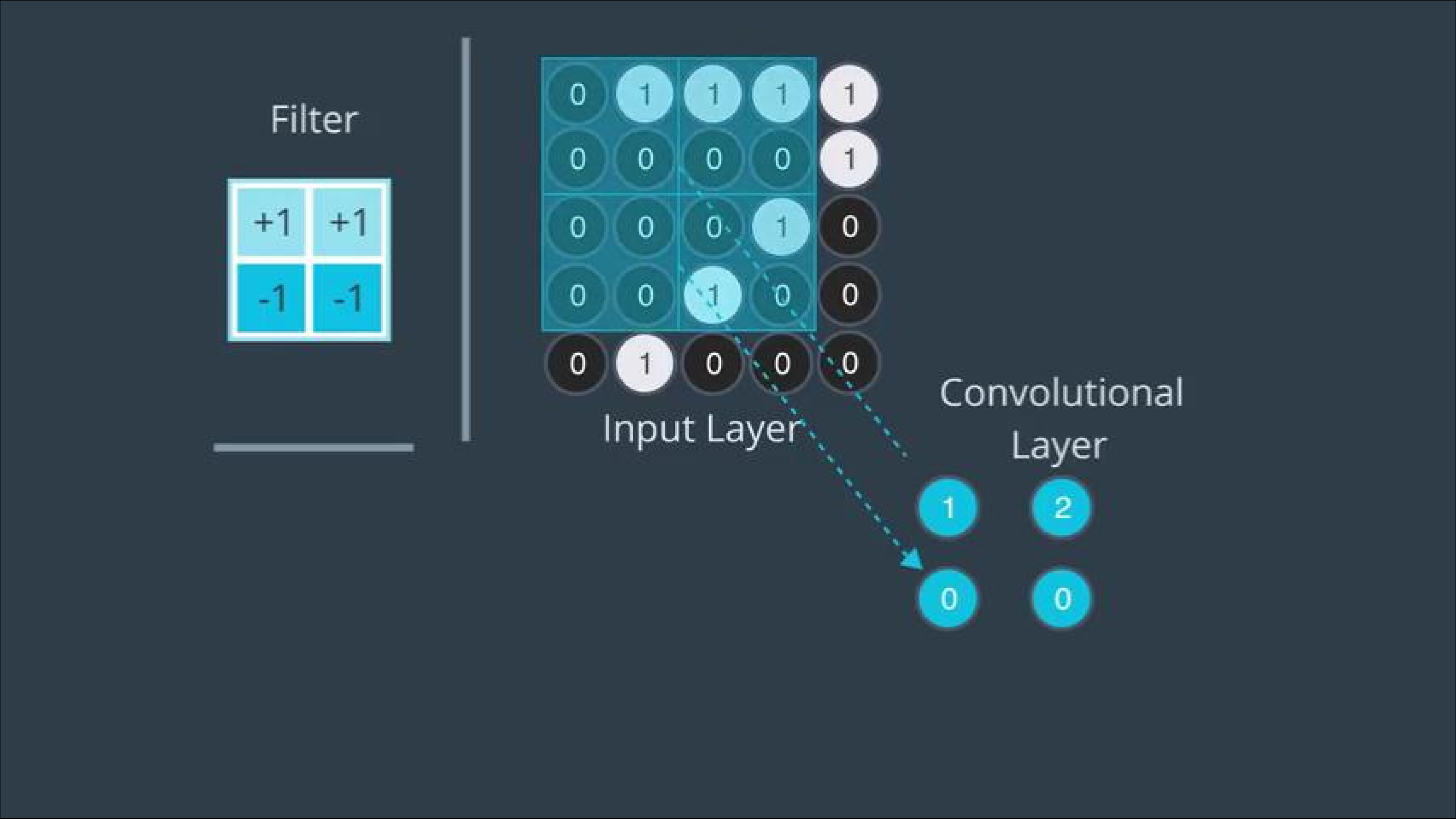

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2
?
0
0
?

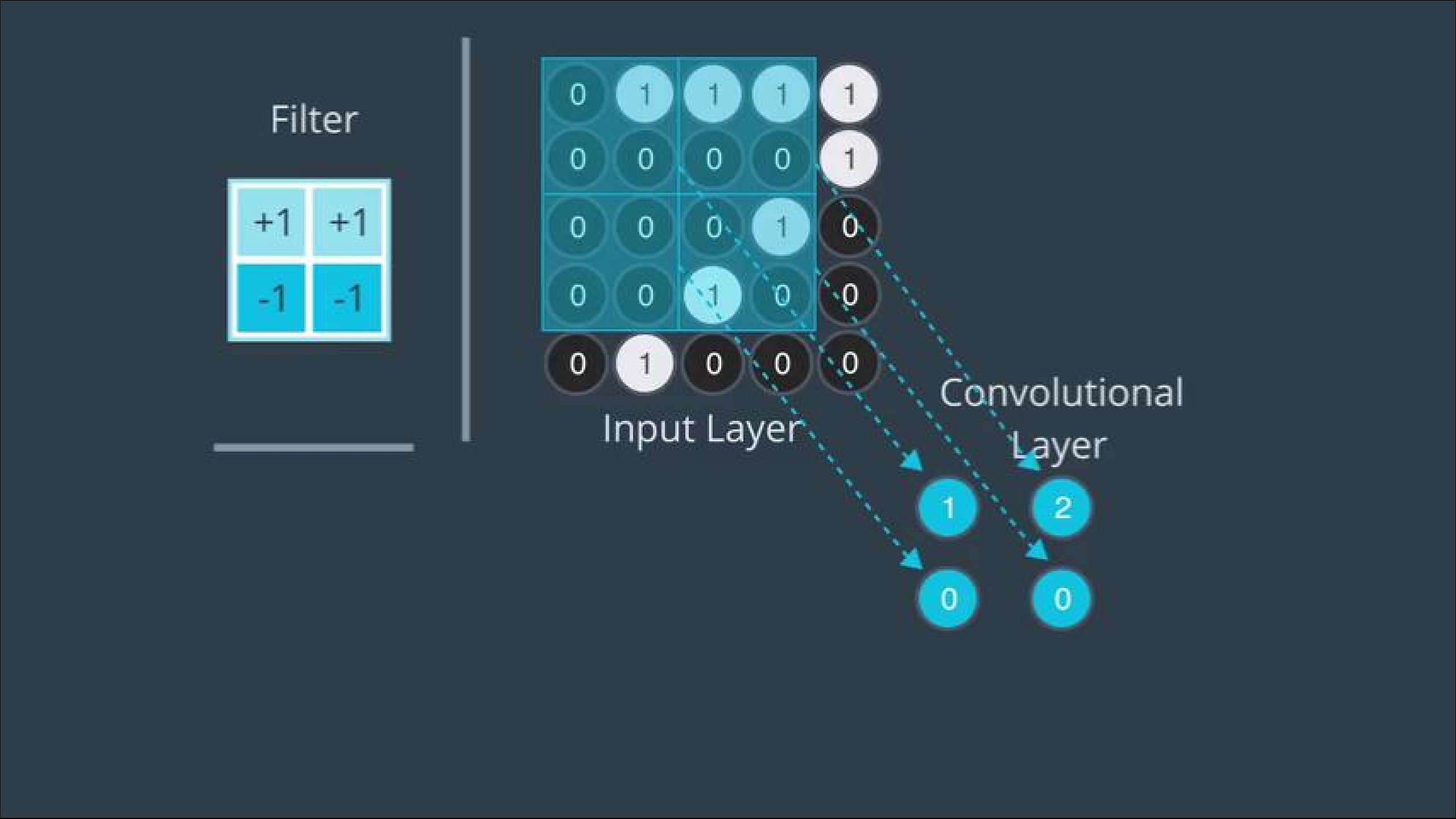

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
1
0
0
0
1
0
0
Input Layer
Convolutional Layer
1
2
?
0
0
?
?
?

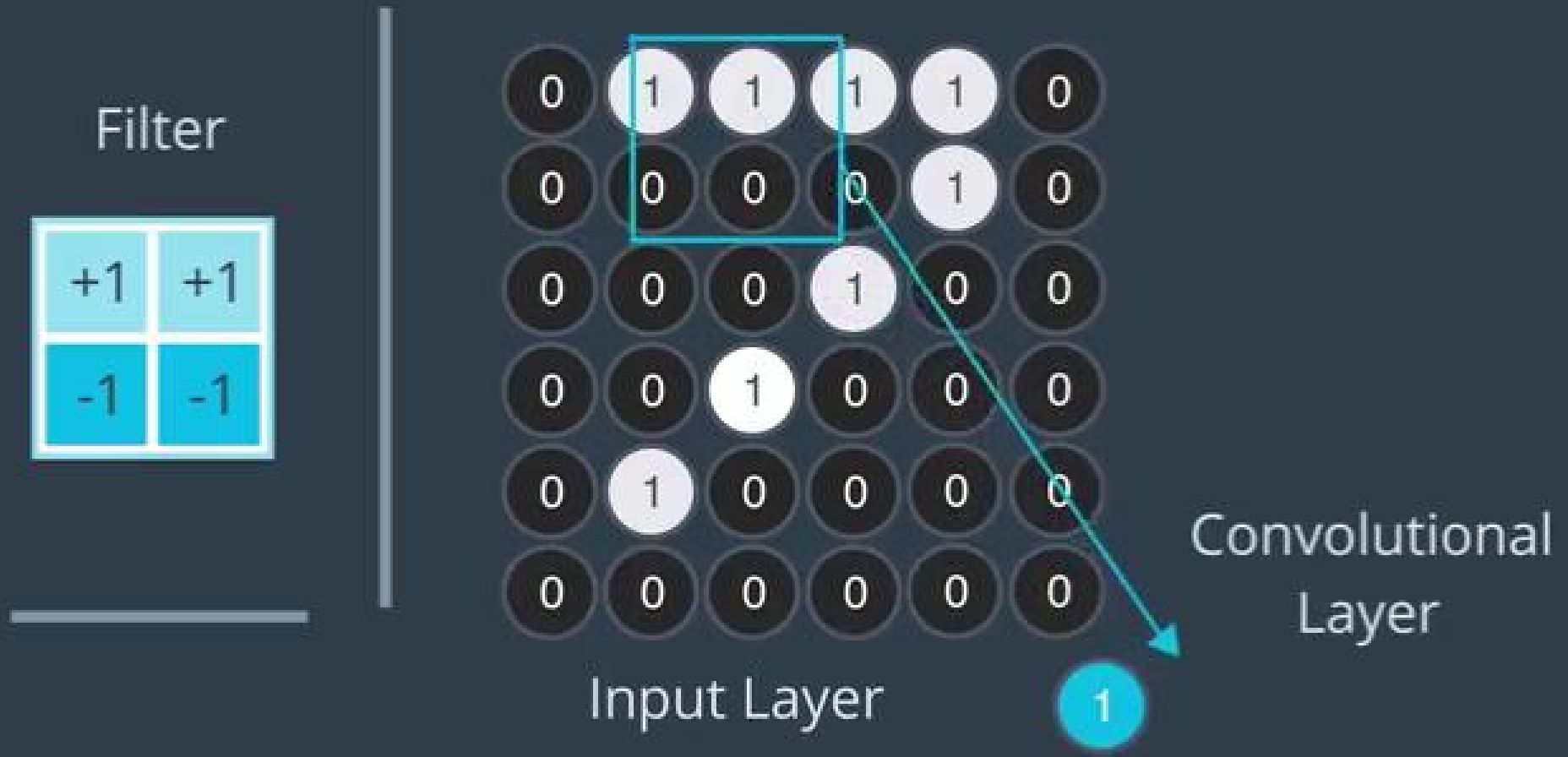

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
1
0
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2
?
0
0
?
?
?
?

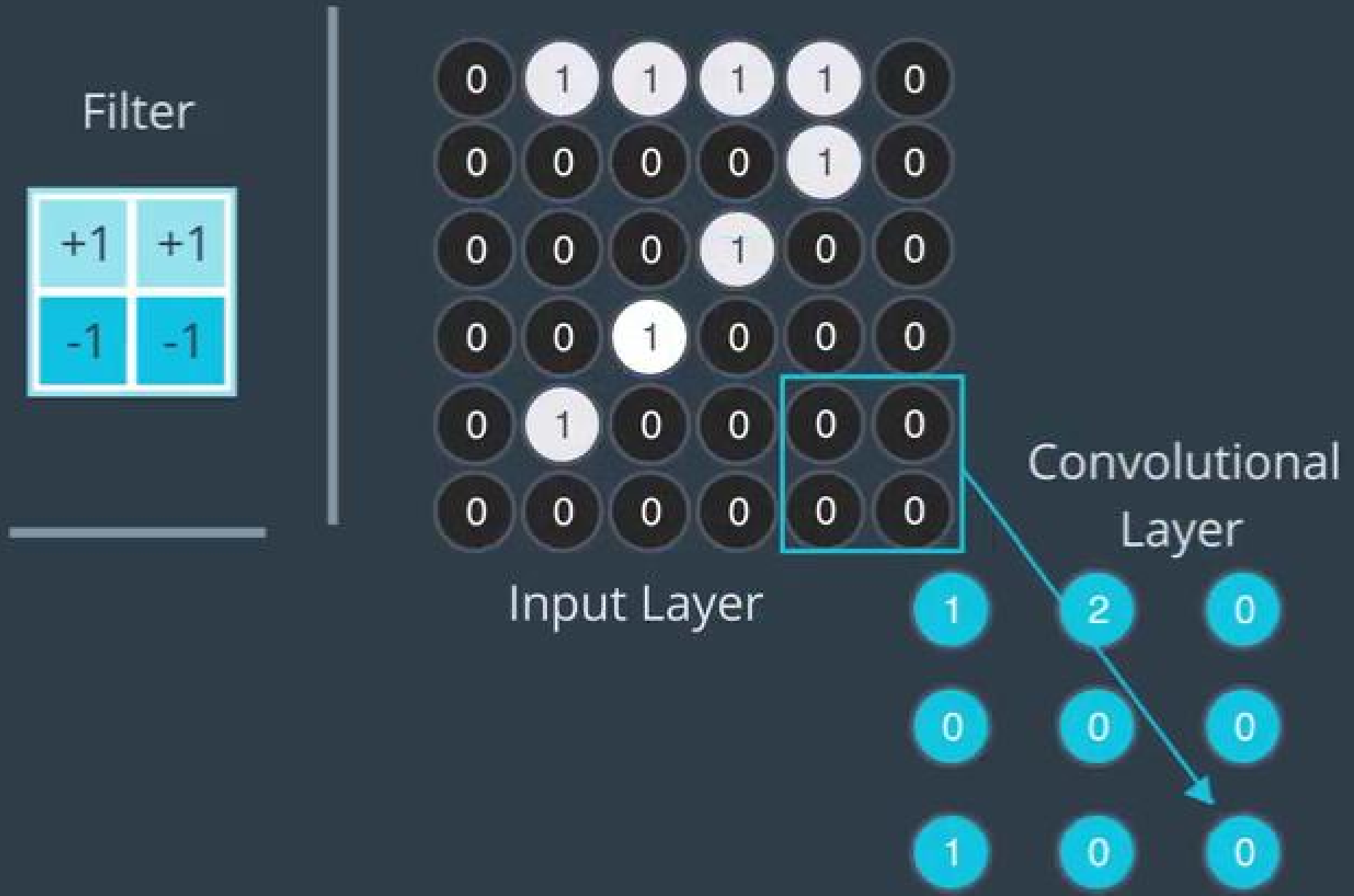

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
1
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2
?
0
0
?
?
?
?

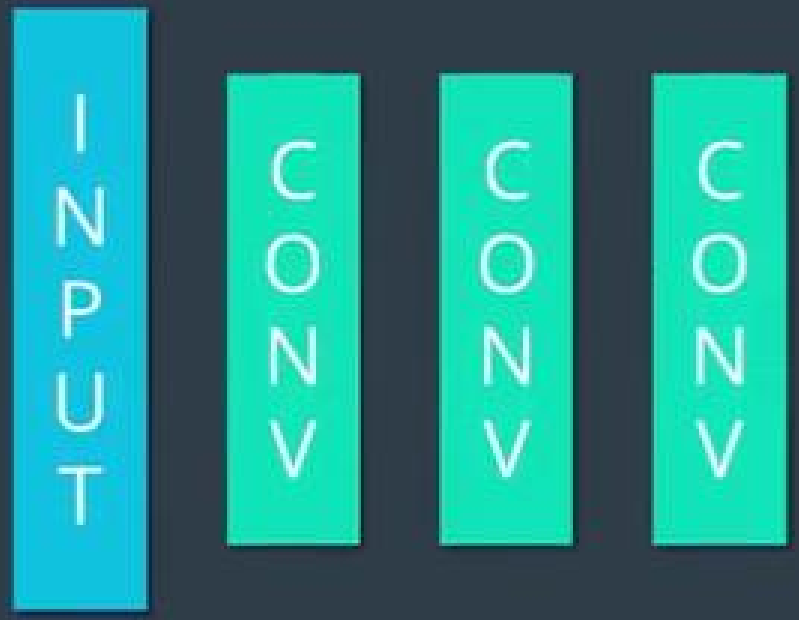

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
1
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2
0
0


Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2
0
0

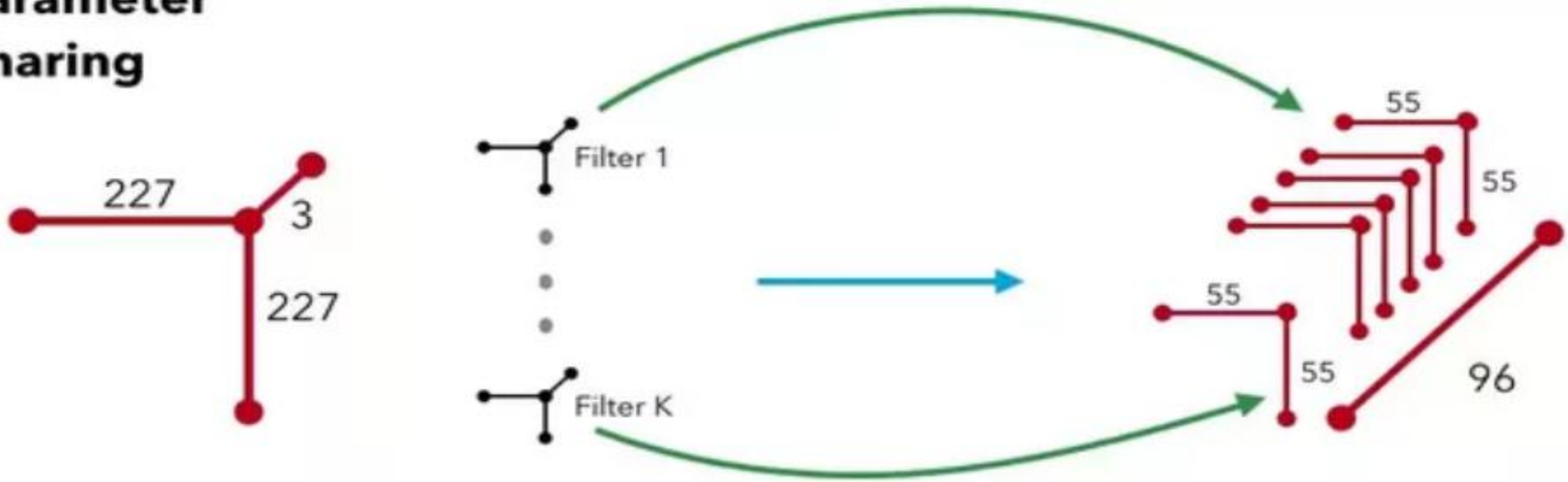

Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
1
0
0
0
Input Layer
Convolutional Layer
1
2
0
0


Filter
+1
+1
-1
-1
0
1
1
1
1
0
0
0
0
0
1
0
0
0
0
1
0
0
0
0
1
0
0
0
0
1
0
0
0
0
0
0
0
0
0
0
Input Layer
1
Convolutional
Layer


Filter
+1
+1
-1
-1
0 1 1 1 1 0
0 0 0 0 1 0
0 0 0 1 0 0
0 0 1 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0
Input Layer
Convolutional Layer
1 2 0
0 0 0
1 0 0


INPUT
CONV
CONV
CONV

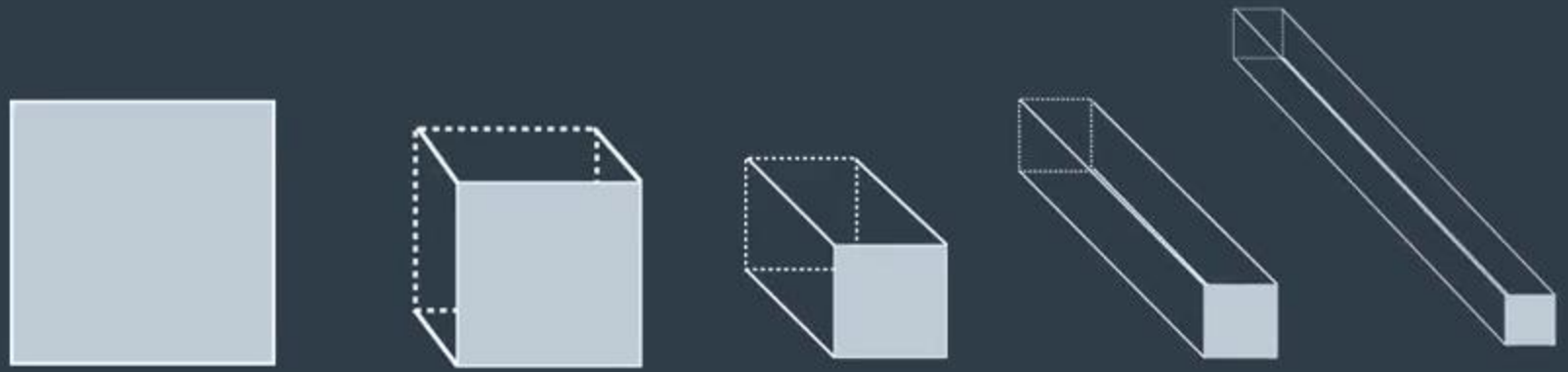
Parameter Sharing

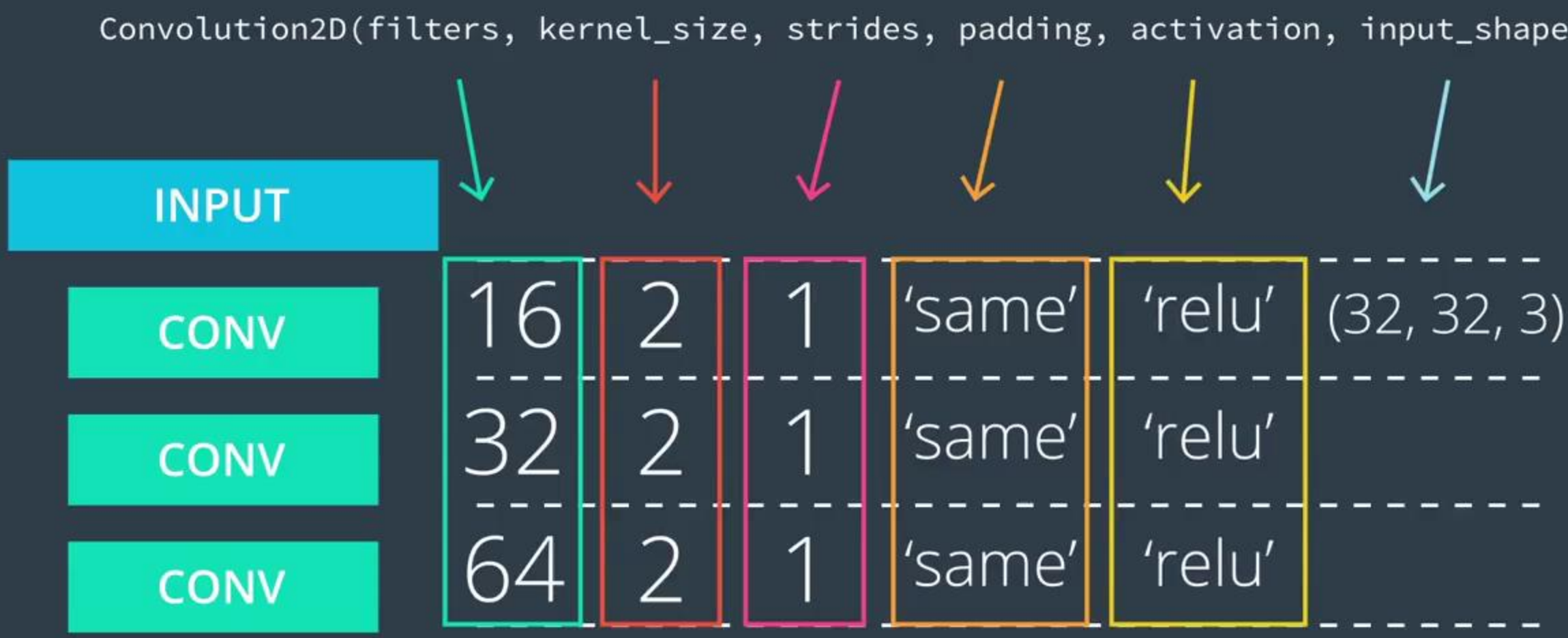


Each filter convolves on the input image and creates a 55 X 55 2D matrix. The K filters together create the output of the first conv layer which is 55 X 55 X K. Each of the 55 X 55 neurons for each depth slice of the output from the first conv layer use the same weights (11 X 11 X 3). The intuition here is that each filter represents a feature and it makes sense to apply the same filter for the entire image using the same weights.

A CNN has multiple layers. Weight sharing happens across the receptive field of the neurons(filters) in a particular layer.Weights are the numbers within each filter. So essentially we are trying to learn a filter. These filters act on a certain receptive field/ small section of the image. When the filter moves through the image, the filter does not change. The idea being, if an edge is important to learn in a particular part of an image, it is important in other parts of the image too.

Convolution Layers increase the image depth and decrease image Width and Height it also converts Spatial Values to Feature Values




Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)
INPUT
CONV
CONV
CONV
16
2
1
'same'
'relu'
(32, 32, 3)
32
2
1
'same'
'relu'
64
2
1
'same'
'relu'

```
from keras.models import Sequential
from keras.layers import Convolution2D

model = Sequential()
model.add(Convolution2D(filters=16, kernel_size=2, padding='same', activation='relu',
                        input_shape=(32, 32, 3)))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(Convolution2D(filters=64, kernel_size=2, padding='same', activation='relu'))

model.summary()
```

```
Using TensorFlow backend.
```

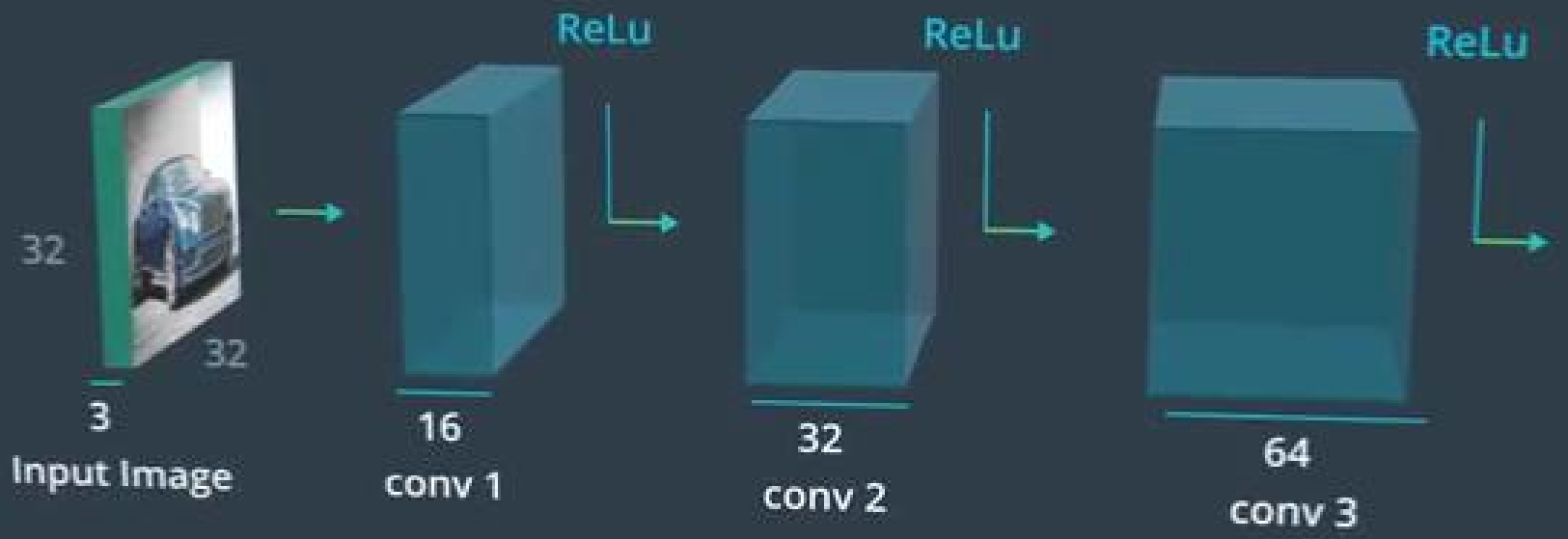
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 16)        208
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        2080
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        8256
=================================================================
Total params: 10,544.0
Trainable params: 10,544
Non-trainable params: 0.0
_________________________________________________________________
```

Define a convolutional layer in PyTorch

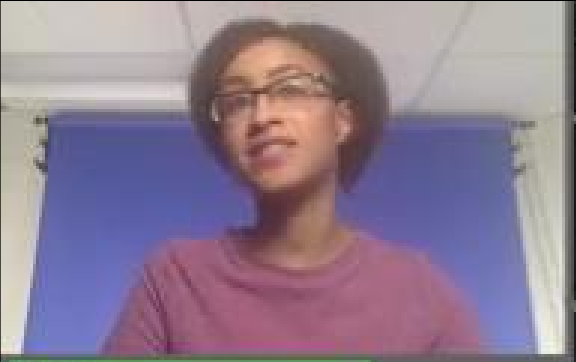
```
self.conv1 = nn.Conv2d(3,16,
    kernel_size, stride = 1, padding = 0)
```

Input Image

convolutional layer




```
self.conv1 = nn.Conv2d(3, 16, 3, padding = 1)
self.conv2 = nn.Conv2d(16, 32, 3, padding = 1)
self.conv3= nn.Conv2d(32, 64, 3, padding = 1)
```



Webcam

Load picture

View Feature maps

View Filters

CONV_0

LOCAL_RESPONSE_NORM_0

MAX_POOL_0

CONV_1

LOCAL_RESPONSE_NORM_1

MAX_POOL_1

CONV_2

CONV_3

CONV_4

MAX_POOL_2

FC_0

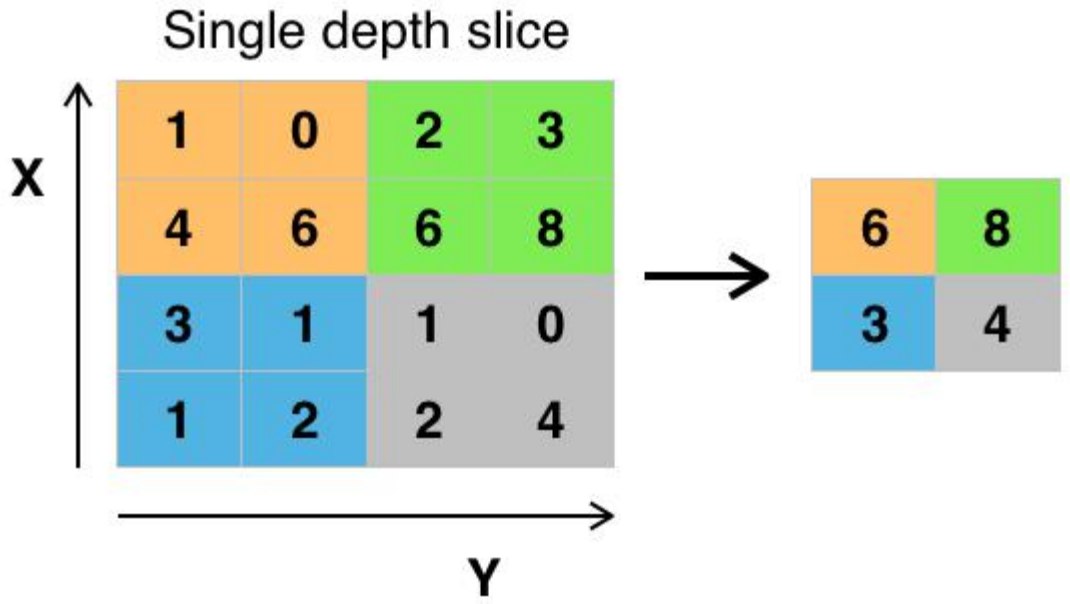
FC_1


INPUT
CONV
POOL
CONV
POOL
CONV
POOL

POOL layer:

Pool Layer performs a function to reduce the spatial dimensions of the input, and the computational complexity of our model. And it also controls overfitting. It operates independently on every depth slice of the input. There are different functions such as Max pooling, average pooling, or L2-norm pooling. However, Max pooling is the most used type of pooling which only takes the most important part (the value of the brightest pixel) of the input volume.

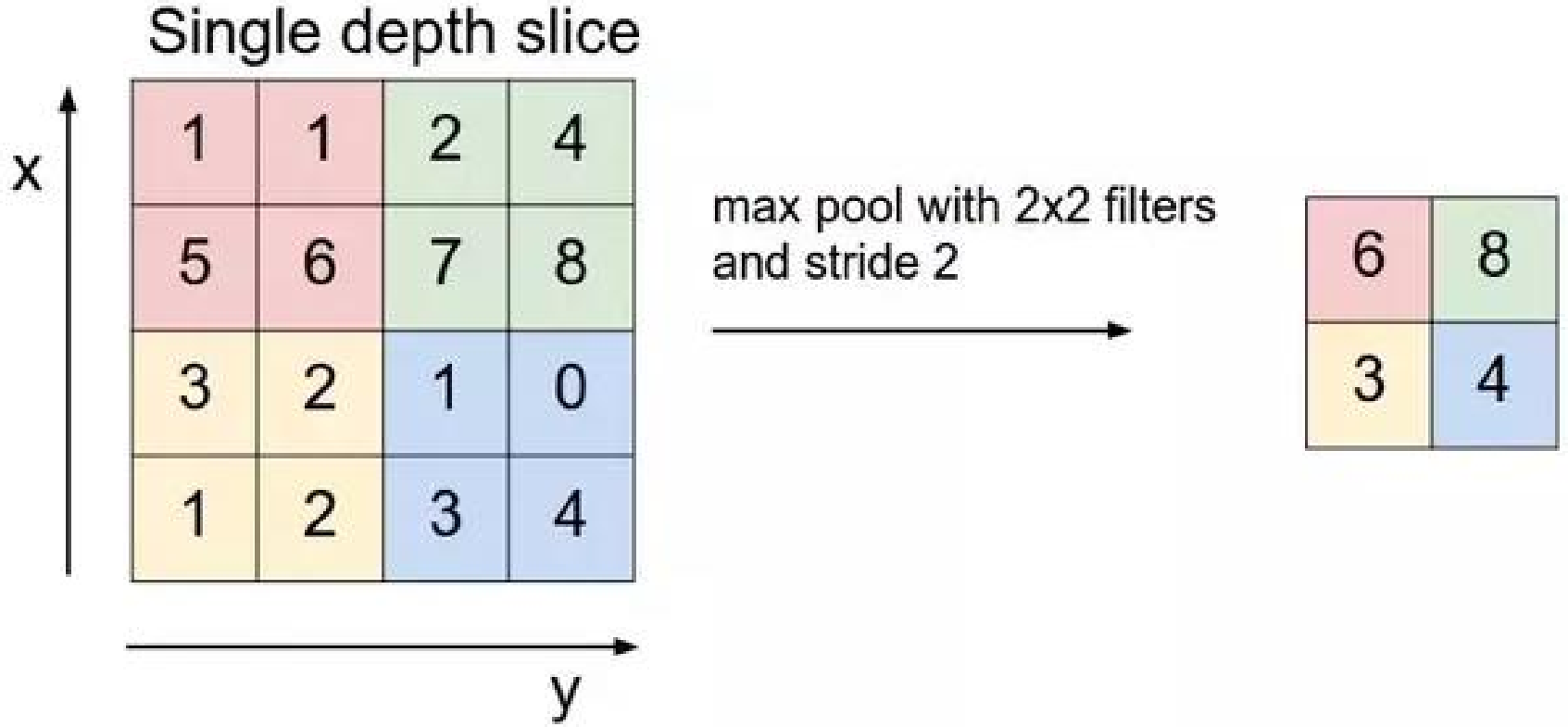
Example of a Max pooling with 2x2 filter and stride = 2. So, for each of the windows, max pooling takes the max value of the 4 pixels.



Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling



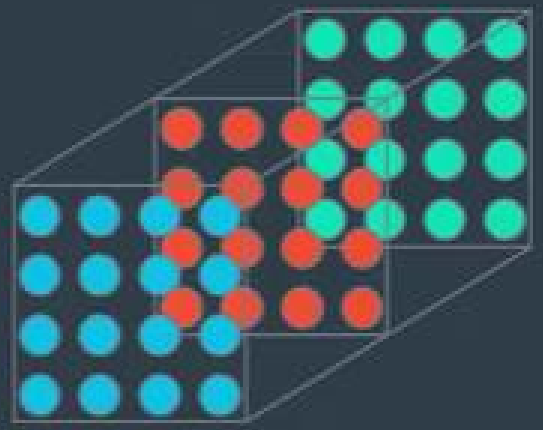
- Pool layer doesn't have parameters (the weights and biases of the neurons), and no zero padding, but it has two hyperparameters: Filter (F) and Stride (S). More generally, having the input $W1 \times H1 \times D1$, the pooling layer produces a volume of size $W2 \times H2 \times D2$ where:

- $W2 = (W1 - F)/S + 1$
- $H2 = (H1 - F)/S + 1$
- $D2 = D1$

A common form of a Max pooling is filters of size 2x2 applied with a stride of 2. The Pooling sizes with larger filters are too destructive and they usually lead to worse performance.

Many people don't like using a pooling layer because it throws away information and they replace it by a Conv layer with increased stride once in a while.

Getting rid of pooling. Many people dislike the pooling operation and think that we can get away without it. For example, Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while. Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs). It seems likely that future architectures will feature very few to no pooling layers.



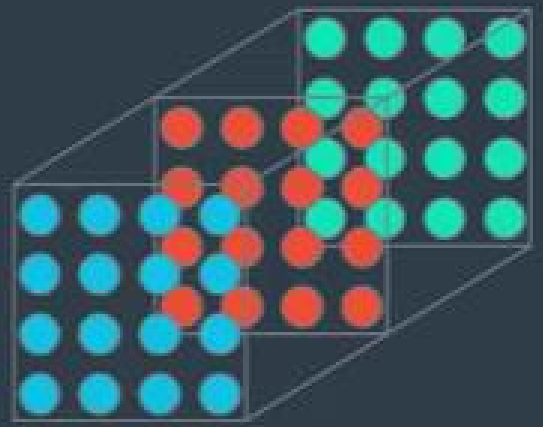
Convolutional Layer

1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0
9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0
7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

Max Pooling Layer

WINDOW SIZE: 2X2

STRIDE: 2



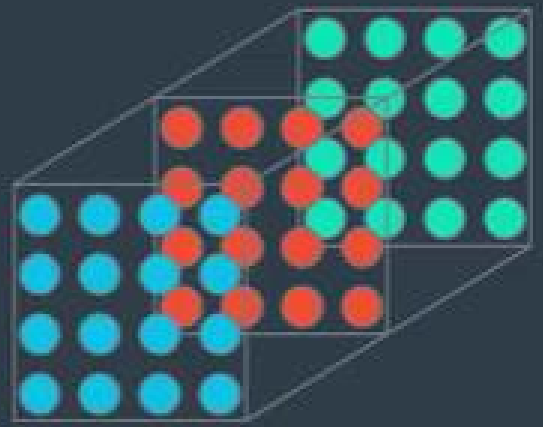
Convolutional Layer

1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0
9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0
7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

Max Pooling Layer

WINDOW SIZE: 2X2

STRIDE: 2



Convolutional Layer

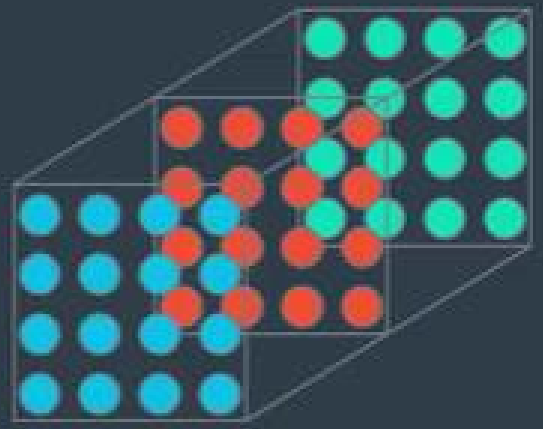
1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0
9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0
7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

9

Max Pooling Layer

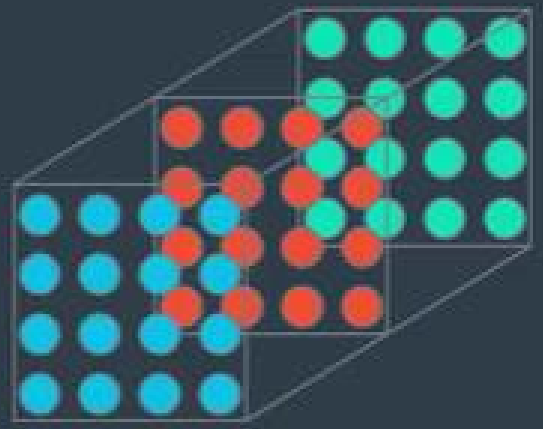
WINDOW SIZE: 2X2

STRIDE: 2



Convolutional Layer

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2



Convolutional Layer

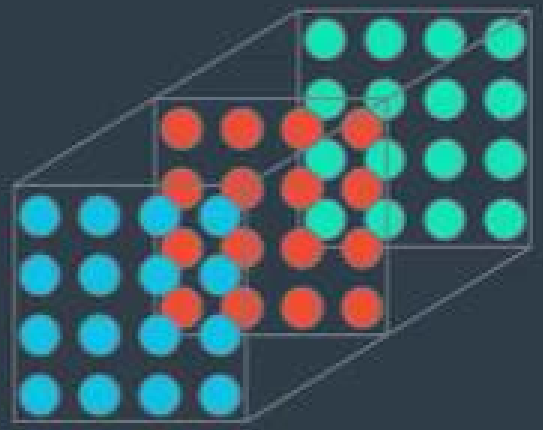
1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0
9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0
7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

9	8
7	

Max Pooling Layer

WINDOW SIZE: 2X2

STRIDE: 2



Convolutional Layer

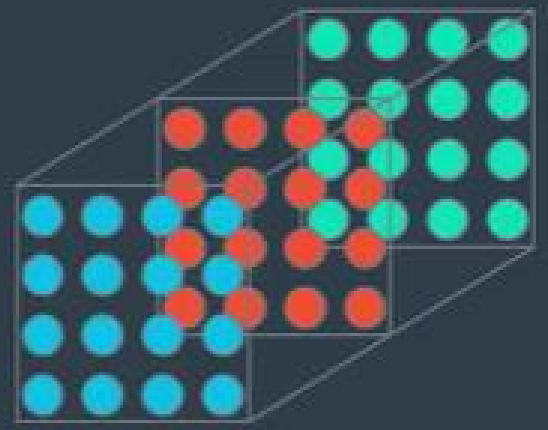
1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0
9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0
7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

9	8
7	9

Max Pooling Layer

WINDOW SIZE: 2X2

STRIDE: 2



Convolutional Layer

1	9	6	4
5	4	7	8
5	1	2	9
6	7	6	0
9	1	7	4
5	6	3	0
1	2	5	4
0	8	9	0
7	6	9	1
5	2	0	4
5	8	3	9
0	2	2	1

9	8
7	9
9	7
8	9
7	9
8	9

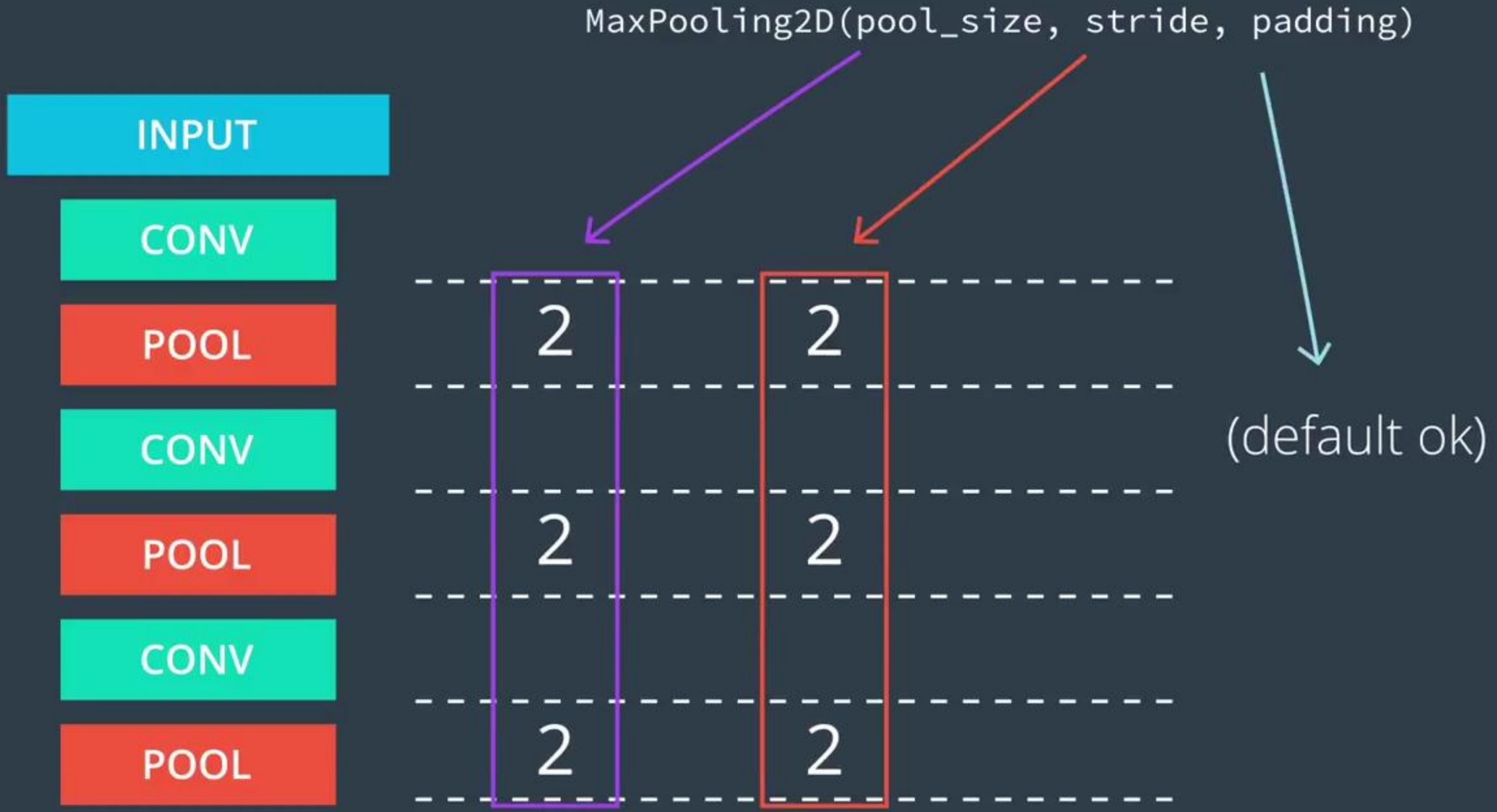
Max Pooling Layer

WINDOW SIZE: 2X2

STRIDE: 2

Convolutional Layer

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2


MaxPooling2D(pool_size, stride, padding)
INPUT
CONV
POOL
CONV
POOL
CONV
POOL
2
2
2
2
2
2
(default ok)

```python
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D

model = Sequential()
model.add(Convolution2D(filters=16, kernel_size=2, padding='same', activation='relu',
                        input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 32, 32, 16)        208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 64)          8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
=================================================================
Total params: 10,544.0
Trainable params: 10,544.0
Non-trainable params: 0.0
_________________________________________________________________
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(64, (3,3), activation='relu',
                         input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

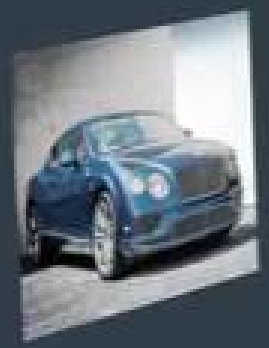
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               204928
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290
=================================================================
```


Input
Image
conv 1
maxpool
conv 2
maxpool
conv 3
maxpool

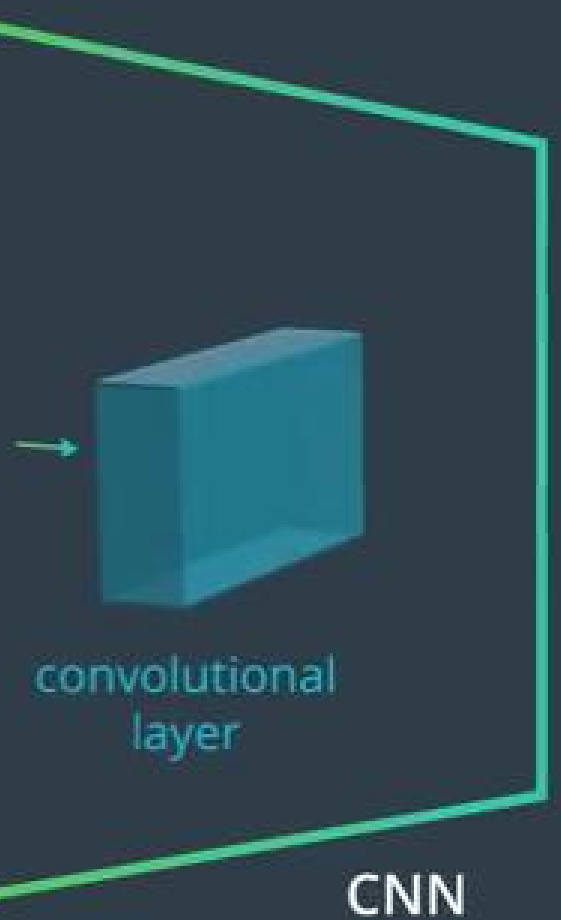
Input Image

Input Image

CNN

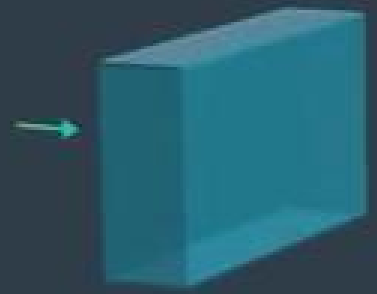


Input Image



CNN

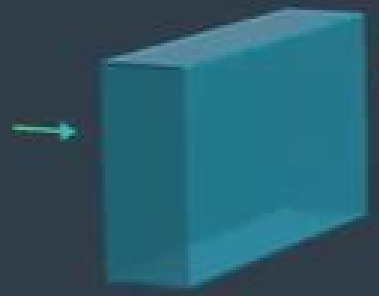
Input Image



convolutional layer

CNN

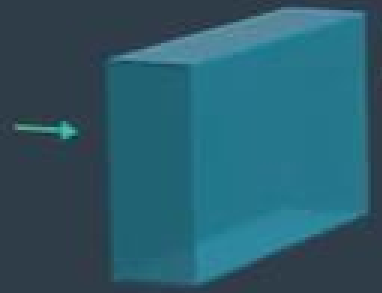
Input Image



convolutional layer

CNN

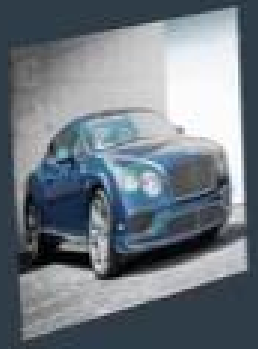
Input Image



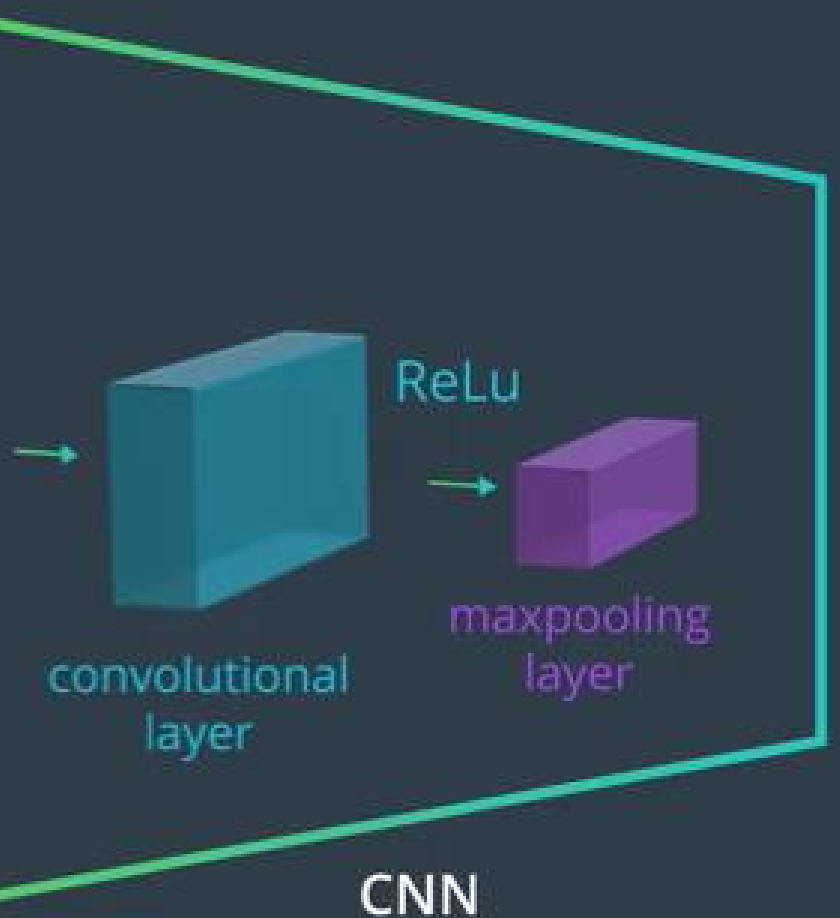
convolutional layer

ReLu

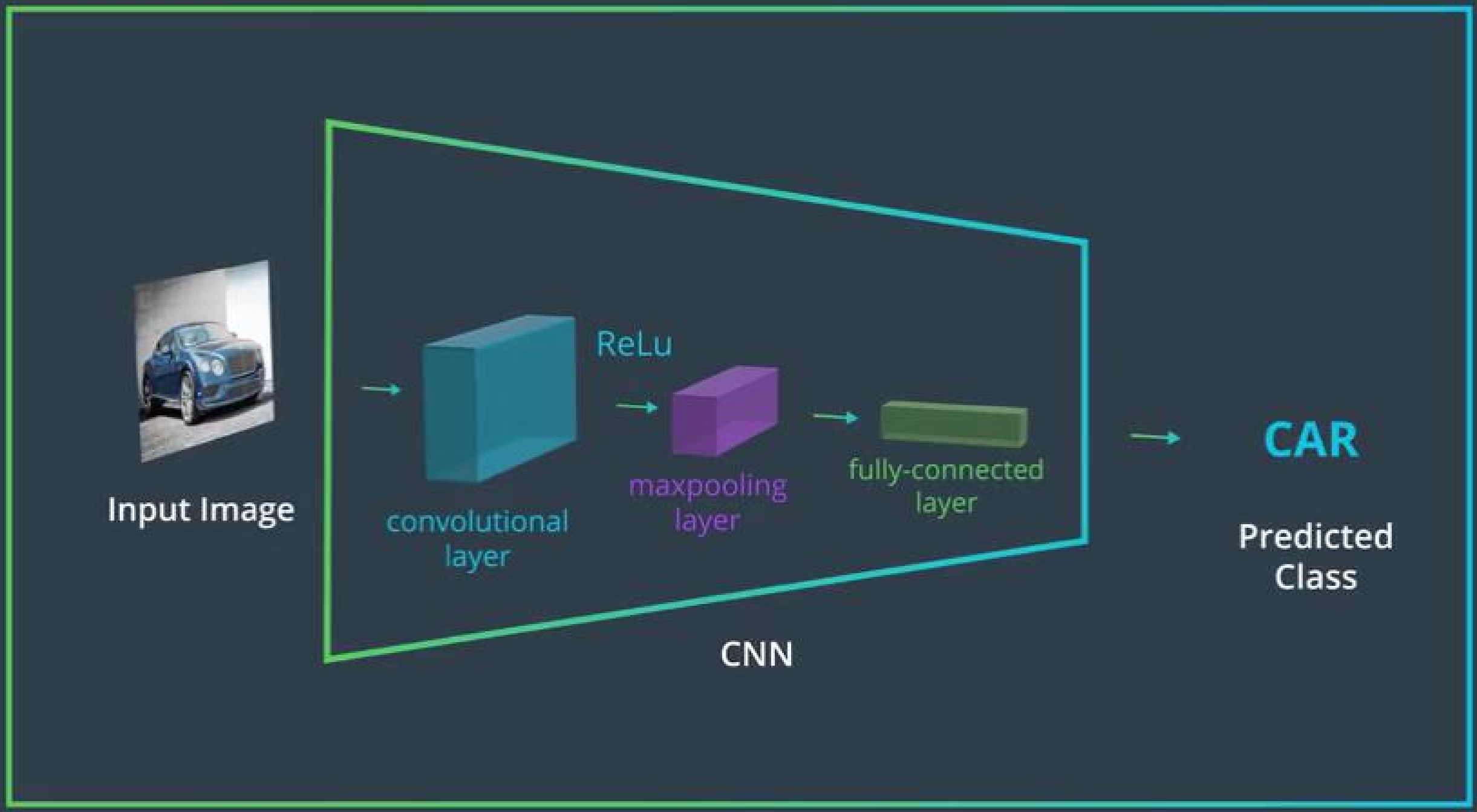
CNN



Input Image

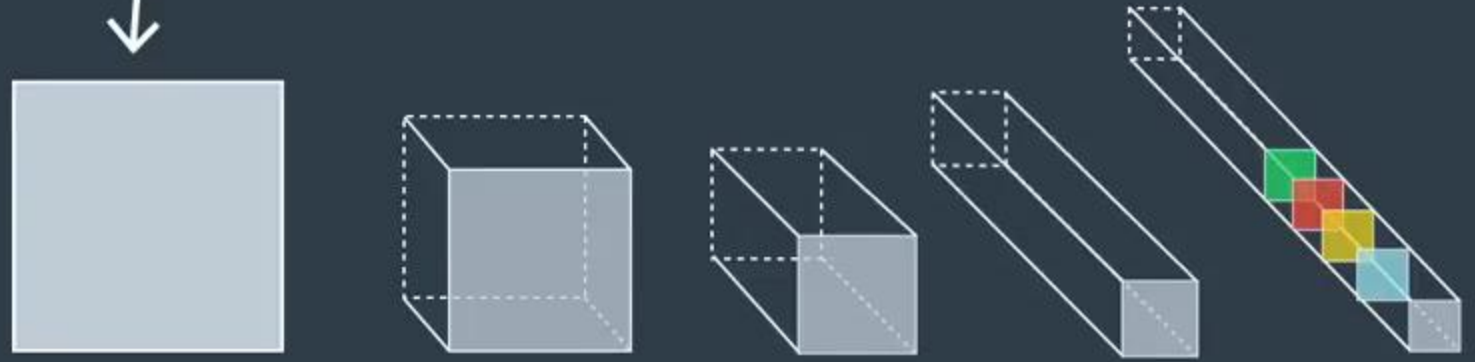


CNN


Input Image
ReLu
convolutional layer
maxpooling layer
fully-connected layer
CNN
CAR
Predicted Class

Contains a ton of spatial information

Contains no spatial information



Wheels? No

Eyes? Yes

Legs? Yes

Tails? Yes

Fully Connected Layer(s)

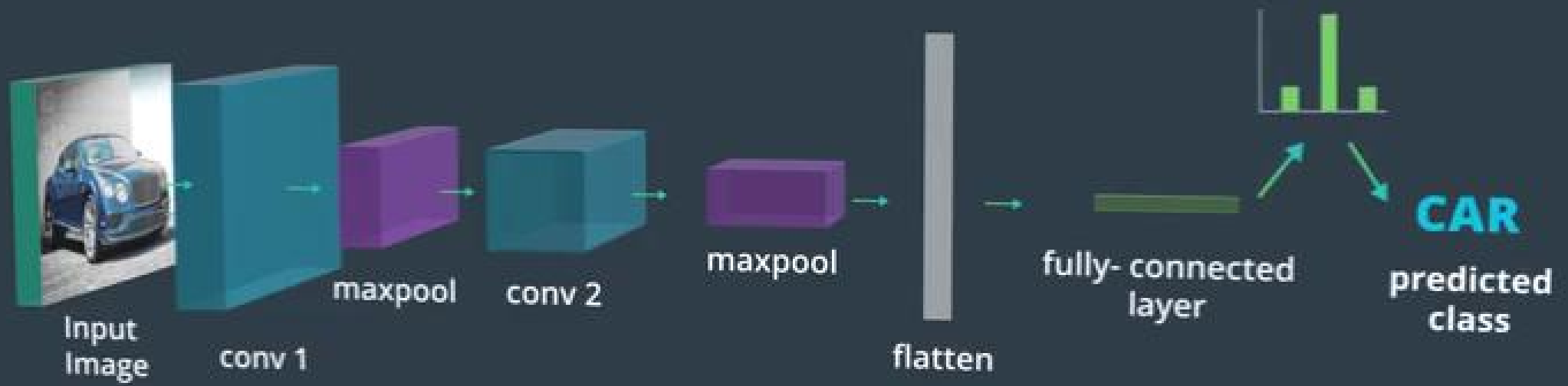
Wheels? Yes
Eyes? No
Legs? No
Tails? No

prob(car) = .99
prob(dog) = .01

Fully Connected Layer(s)

Wheels? No
Eyes? Yes
Legs? Yes
Tails? Yes

prob(car) = .01
prob(dog) = .99

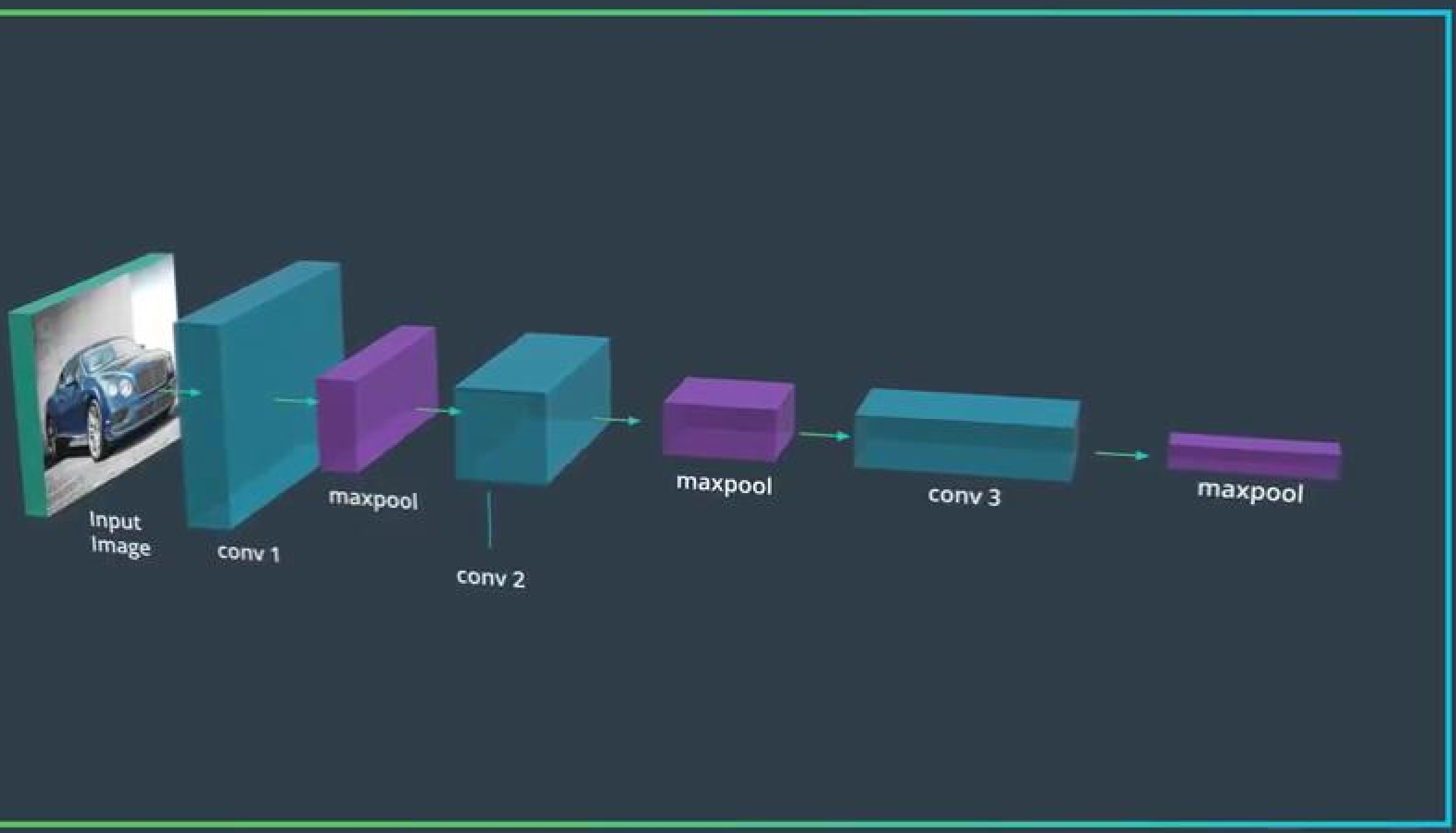


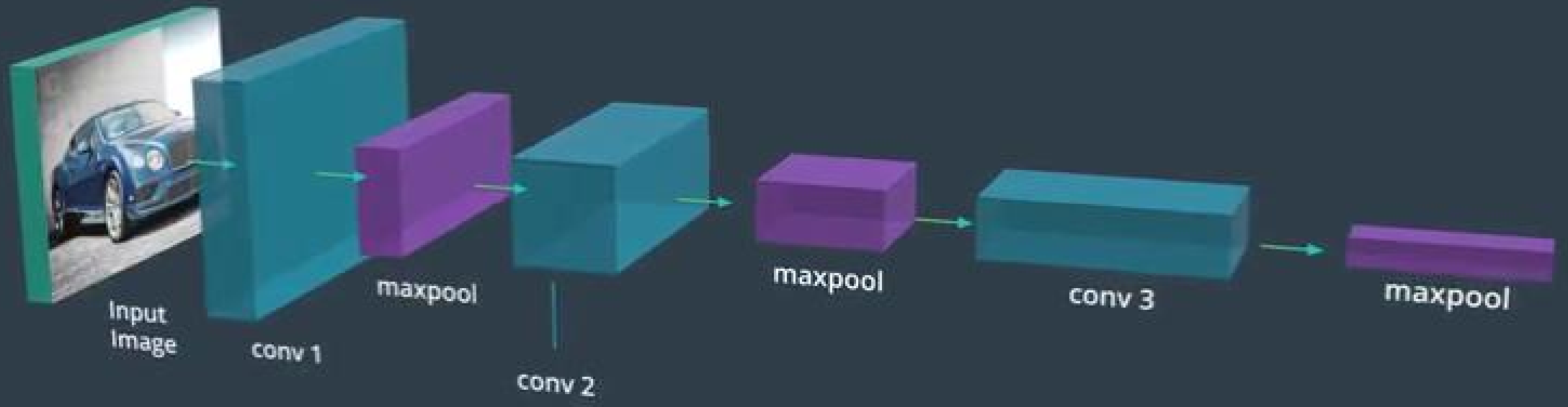
extracting more and more complex features

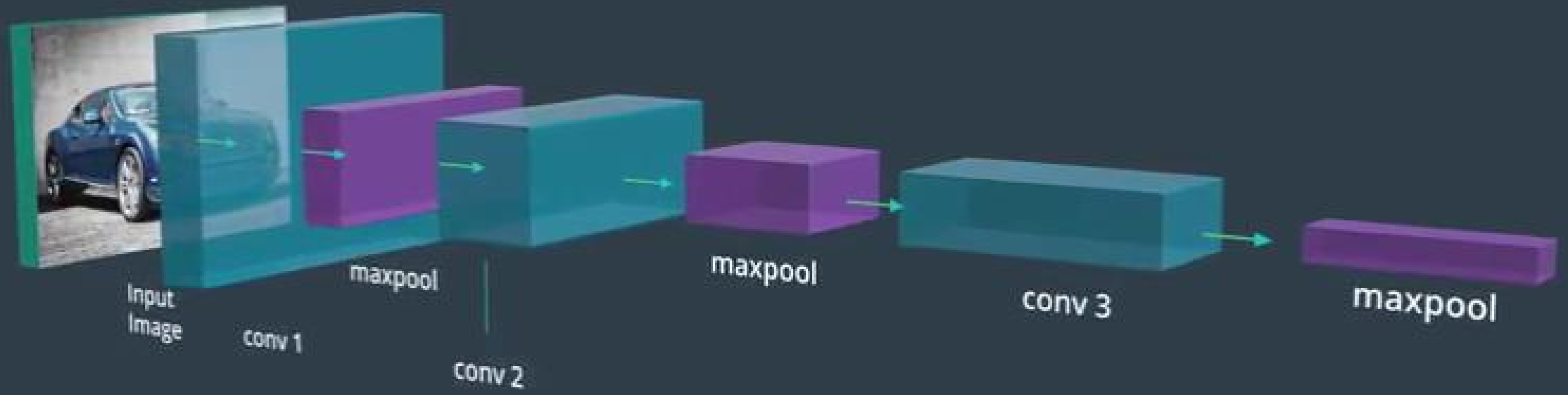
learns to distill the **content** of the image

```python
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense

model = Sequential()
model.add(Convolution2D(filters=16, kernel_size=2, padding='same', activation='relu',
                        input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 32, 32, 16)        208
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 32)        2080
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 64)          8256
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               512500
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5010
=================================================================
Total params: 528,054.0
Trainable params: 528,054.0
Non-trainable params: 0.0
_________________________________________________________________
```

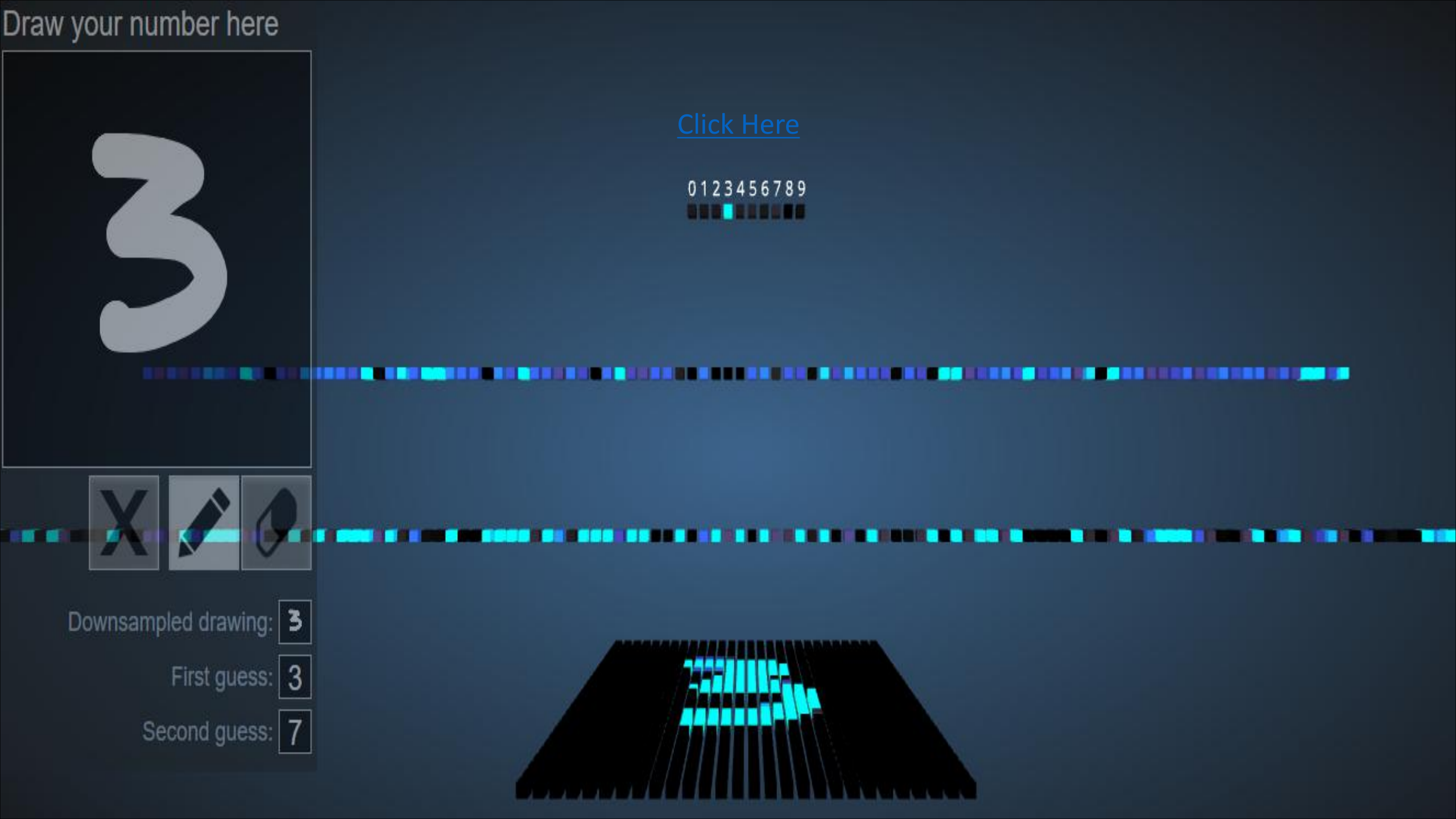

Input
Image
conv 1
maxpool
conv 2
maxpool
conv 3
maxpool


Input
Image
conv 1
maxpool
conv 2
maxpool
conv 3
maxpool


Input
Image
conv 1
maxpool
conv 2
maxpool
conv 3
maxpool

POOL RELU RELU CONV CONV POOL RELU RELU CONV CONV POOL RELU RELU CONV CONV POOL FC

car
truck
airplane
ship
horse


Draw your number here
Click Here
0 1 2 3 4 5 6 7 8 9
Downsampled drawing:
First guess: 3
Second guess: 7

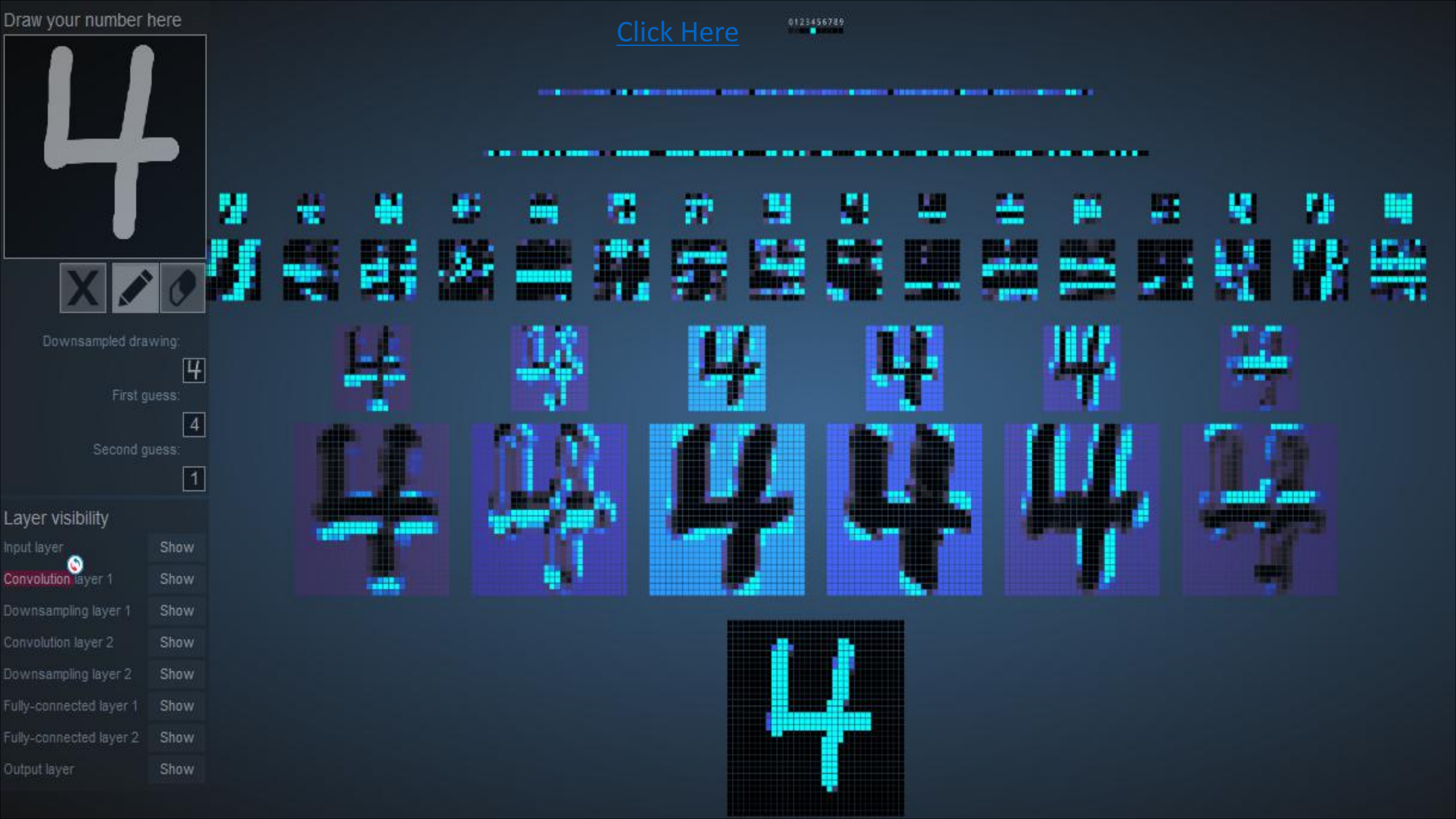

Draw your number here
Click Here
0123456789
Downsampled drawing:
First guess:
4
Second guess:
1
Layer visibility
Input layer
Show
Convolution layer 1
Show
Downsampling layer 1
Show
Convolution layer 2
Show
Downsampling layer 2
Show
Fully-connected layer 1
Show
Fully-connected layer 2
Show
Output layer
Show

Image Preprocessing and image augmentation

Image Scaling

32x32

32x32

32x32

Invariant representation

Scale Invariance

Rotation Invariance

Translation Invariance

Data Augmentation

Rotation invariance

Translation invariance

```
from tensorflow.keras.preprocessing.image
import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300, 300),
        batch_size=128,
        class_mode='binary')
```

5. Create and configure augmented image generator

```
from keras.preprocessing.image import ImageDataGenerator

# create and configure augmented image generator
datagen = ImageDataGenerator(
    width_shift_range=0.1, # randomly shift images horizontally (10% of total width)
    height_shift_range=0.1, # randomly shift images vertically (10% of total height)
    horizontal_flip=True) # randomly flip images horizontally

# fit augmented image generator on data
datagen.fit(x_train)
```

6. Visualize original and augmented images

```
import matplotlib.pyplot as plt

# take subset of training data
x_train_subset = x_train[:12]

# visualize subset of training data
fig = plt.figure(figsize=(20,2))
for i in range(0, len(x_train_subset)):
    ax = fig.add_subplot(1, 12, i+1)
    ax.imshow(x_train_subset[i])
fig.suptitle('Subset of Original Training Images', fontsize=20)

# visualize augmented images
fig = plt.figure(figsize=(20,2))
for x_batch in datagen.flow(x_train_subset, batch_size=12):
    for i in range(0, 12):
        ax = fig.add_subplot(1, 12, i+1)
        ax.imshow(x_batch[i])
    fig.suptitle('Augmented Images', fontsize=20)
    plt.show()
    break;
```

7. Define the model architecture

```
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential()
model.add(Convolution2D(filters=16, kernel_size=2, padding='same', activation='relu',
                        input_shape=(32, 32, 3))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

model.summary()
```

8. Compile the model

```
# compile the model
model.compile(loss='categorical_crossentropy' , optimizer='rmsprop',
                    metrics=['accuracy'])
```

9. Train the model

```
from keras.callbacks imports ModelCheckpoint

batch_size = 32
epochs = 100

# train the model
checkpointer = ModelCheckpoint(filepath='aug_model.weights.best.hdf5', verbose=1,
                               save_best_only=True)
model.fit_generator(datagen.flow(x_train, y_train, batch_size=batch_size),
                    steps_per_epoch=x_train.shape[0] // batch_size,
                    epochs=epochs, verbose=2, callbacks=[checkpointer],
                    validation_data=(x_valid, y_valid))
```

10. Load the model with the best validation accuracy

```
# load the weights that yielded the best validation accuracy
model.load_weights('aug_model.weights.best.hdf5')
```

11. Calculate classification accuracy on test set

```
# evaluate and print test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

```
Test accuracy 0.6825
```

Train your Own Data

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 100,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 100,
                                            class_mode = 'binary')

classifier.fit_generator(training_set,
                         samples_per_epoch = 8000,
                         nb_epoch = 25,
                         validation_data = test_set,
                         nb_val_samples = 2000)
```

Train your Own Data

samples_per_epoch is usually setted as:

```
samples_per_epoch=train_generator.nb_samples
```

This way you are ensuring that every epoch you are seeing a number of samples equal to the size of your training set. This means that you are seeing all of your training samples at every epoch.

---

nb_epoch is pretty much up to you. It determines how many times you iterate over a number defined by `samples_per_epoch`.

To give you an example, in your code right now your model is 'seeing' `(nb_epoch * samples_per_epoch)` images, wich in this case are 65000 images.

---

nb_val_samples determines over how many validation samples your model is evaluated after finishing every epoch. It is up to you aswell. The usual thing is to set:

```
nb_val_samples=validation_generator.nb_samples
```

In order to evaluate your model on the full validation set.

---

batch_size determines how many images are feeded **at the same time** to your gpu (or cpu). Rule of dumb is to set the largest `batch_size` that your gpu's memmory allows. Ideal batch_size is active area of research nowdays, but usually a bigger batch_size will work better.

```
classifier.fit_generator(training_set,
                         samples_per_epoch = 8000,
                         nb_epoch = 25,
                         validation_data = test_set,
                         nb_val_samples = 2000)
```

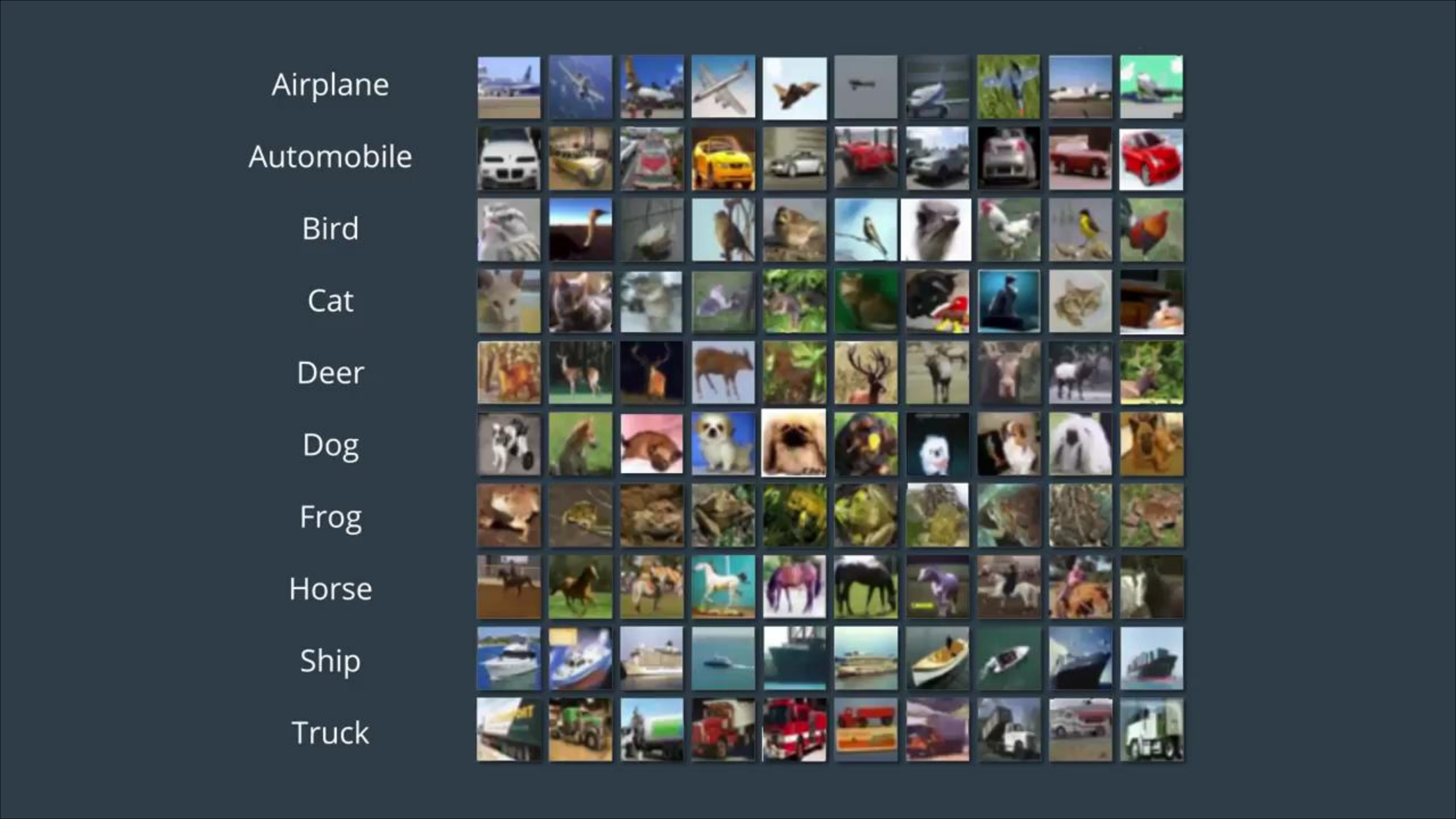
MNIST using CNN

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

MNIST using CNN

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

CIFAR10-CNN


Airplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck

1. Load CIFAR -10 Database

```
import keras
from keras.datasets import cifar10

# load the pre-shuffled train and test data
(x_train, y_train), (x_test, y_test) = cifar10.load_data ()

Using TensorFlow backend.
```

2. Visualize the first 24 training images

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

figt = plt.figure(figsize=(20,5))
for i in range (36):
    ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]))
```

3. Rescale the images by dividing every pixel in every image by 255

```
# rescale [0,255] --> [0,1]
x_train = x_train.astype('float32')/255
x_tests = x_test.astype('float32')/255
```

4. Break dataset into Training, Testing, and Validation sets

```
from keras.utils import np_utils

# one-hot encode the labels
num_classes = len(np.uniquely(y_train))
y_train = keras.utils.to_categorically(y_train, num_classes)
y_test = keras.utils.to_categorically(y_test, num_classes)

# break training set into training and validation sets
(x_train, x_valid) = x_train[5000:], x_train[:5000]
(y_train, y_valid) = y_train[5000:], y_train[:5000]

# print shape of training sets
print('x_train shape:', x_train.shape)

# print number of training, validation, and test images
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_valid.shape[0], 'validation samples')
```

```
x_train shape: (45000, 32, 32, 3)
45000 train samples
10000 test samples
5000 validation samples
```

5. Define the model architecture

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten

# define the model
model = Sequential()
model.add(Flatten(input_shape = x_train.shape[1:]))
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 3072)	0
dense_1 (Dense)	(None, 1000)	3073000
dropout_1 (Dropout)	(None, 1000)	0
dense_2 (Dense)	(None, 512)	512512
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

Total params: 3,590,642.0
Trainable params: 3,590,642.0
Non-trainable params: 0.0

6. Compile the model

```
# compile the model
model.compile(loss='categorical_crossentropy', optimizer= 'rmsprop',
                    metrics=['accuracy'])
```

7. Train the model

```
from keras.callbacks import ModelCheckpoint

# train the model
checkpointer = ModelCheckpoint(filepath='MLP.weights.best.hdf5', verbose=1,
                               save_best_only=True)
hist = model.fit(x_train,  y_train, batch_size=32, epochs=20,
          validation_data=(x_valid, y_valid), callbacks=[checkpointer],
          verbose=2, shuffle=True)
```

8. Load the model with the best classification accuracy on the validation set

```
# load the weights that yielded the best validation accuracy
model.load_weights('MLP.weights.best.hdf5')
```

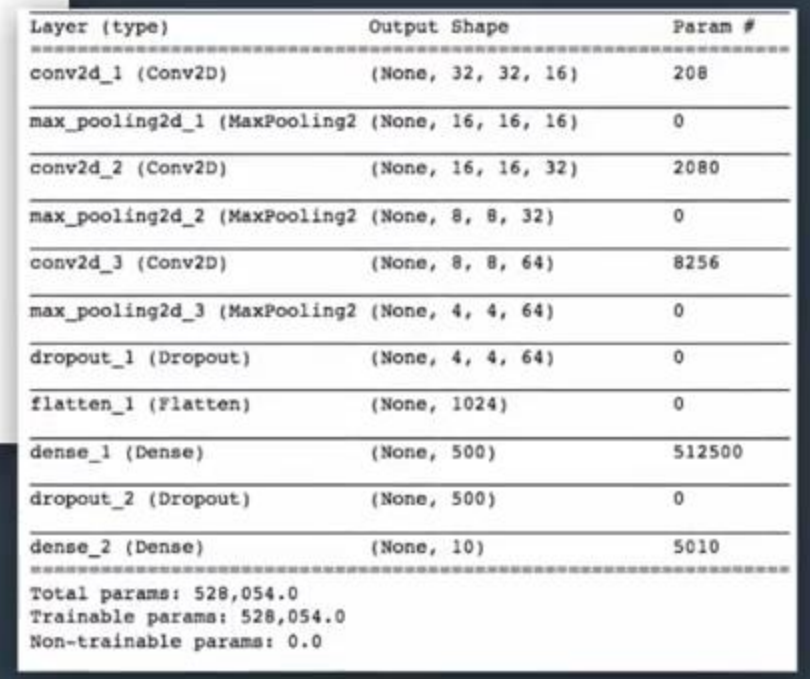
9. Calculate classification accuracy on test set

```
# evaluate and print test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:' , score[1])
```

```
Test accuracy 0.4
```

5. Define the model architecture

```
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential()
model.add(Convolution2D(filters=16, kernel_size=2, padding='same', activation='relu',
                        input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))


model.summary()
```



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 16)	208
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	2080
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 32)	0
conv2d_3 (Conv2D)	(None, 8, 8, 64)	8256
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 64)	0
dropout_1 (Dropout)	(None, 4, 4, 64)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_1 (Dense)	(None, 500)	512500
dropout_2 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 10)	5010

Total params: 528,054.0
Trainable params: 528,054.0
Non-trainable params: 0.0

6. Compile the model

```
# compile the model
model.compile(loss='categorical_crossentropy', optimizer= 'rmsprop',
                    metrics=['accuracy'])
```

7. Train the model

```
from keras.callbacks import ModelCheckpoint

# train the model
checkpointer = ModelCheckpoint(filepath='MLP.weights.best.hdf5', verbose=1,
                               save_best_only=True)
hist = model.fit(x_train,  y_train, batch_size=32, epochs=100,
          validation_data=(x_valid, y_valid), callbacks=[checkpointer],
          verbose=2, shuffle=True)
```

8. Load the model with the best classification accuracy on the validation set

```
# load the weights that yielded the best validation accuracy
model.load_weights('model.weights.best.hdf5')
```

9. Calculate classification accuracy on test set

```
# evaluate and print test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:' , score[1])
```

```
Test accuracy 0.6598
```