Introduction to Computer Vision CNNs

Convolutional Neural Network

A special mathematical operation

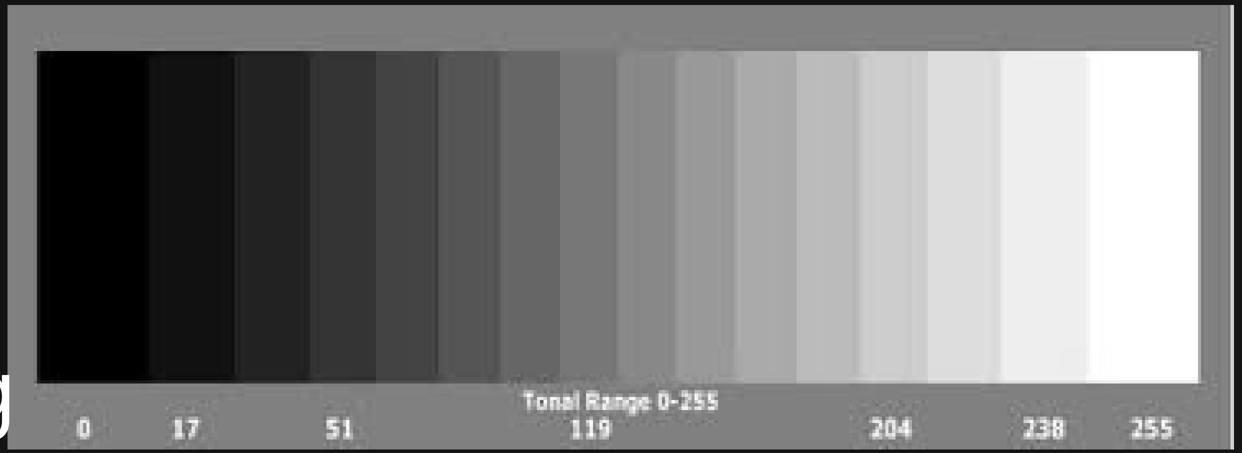
It's another class of neural networks used to get important information in images

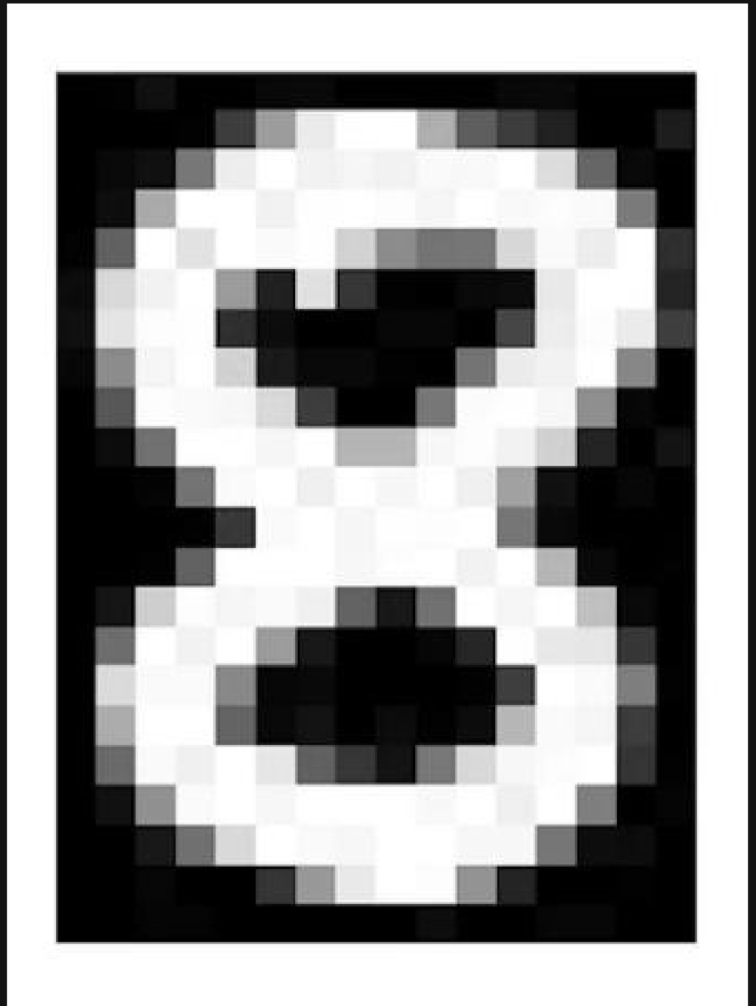
CONTENT

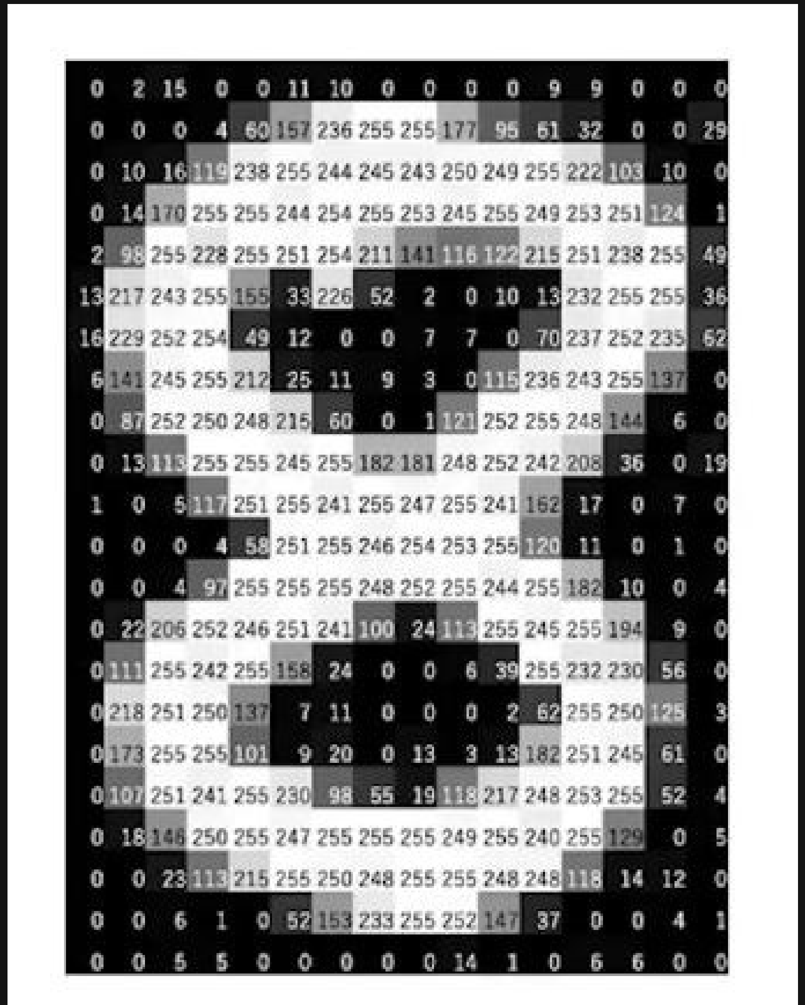
HOW DO COMPUTERS PROCESS IMAGES?

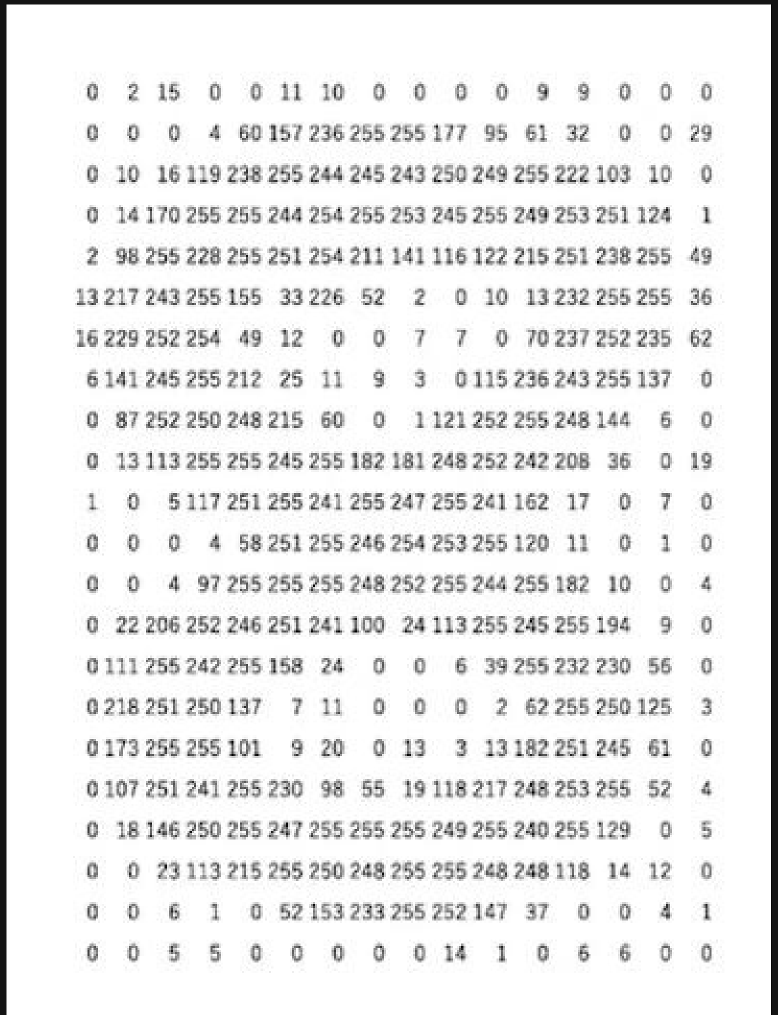
We convert it into a grayscale/RGB image

Each pixel of the image has a different intensity of the colour white. Namely, this intensity ranges from 0 to 255. Thus we convert each pixel into its corresponding numerical value.





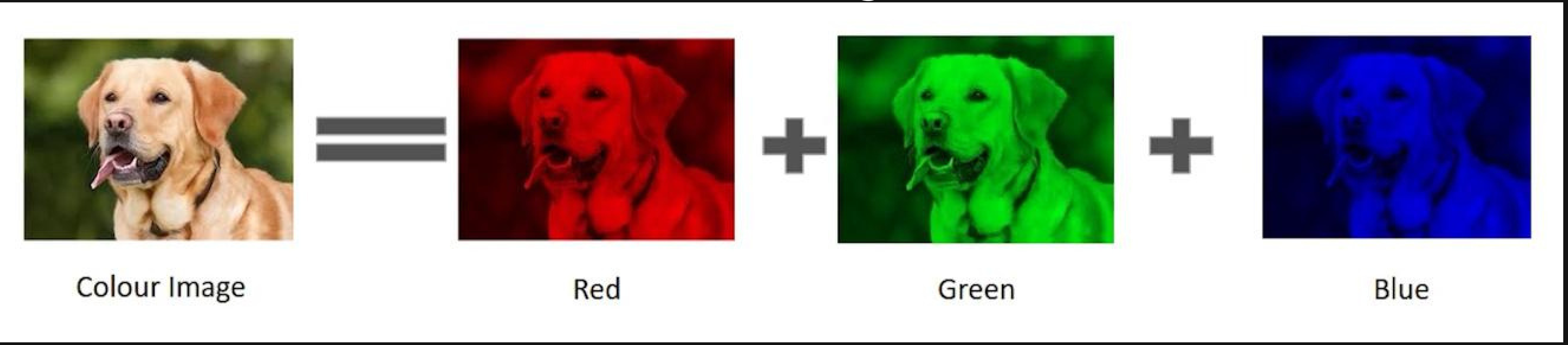


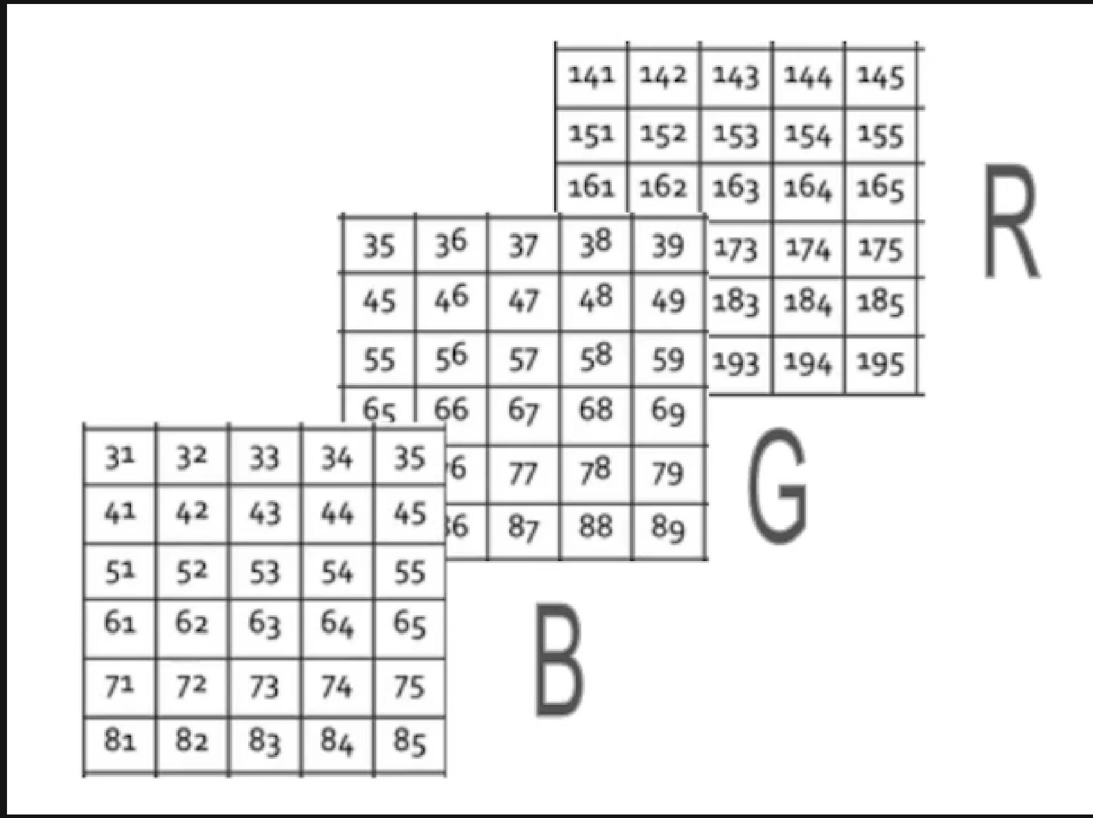


0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

RED GREEN BLUE IMAGES(RGB)

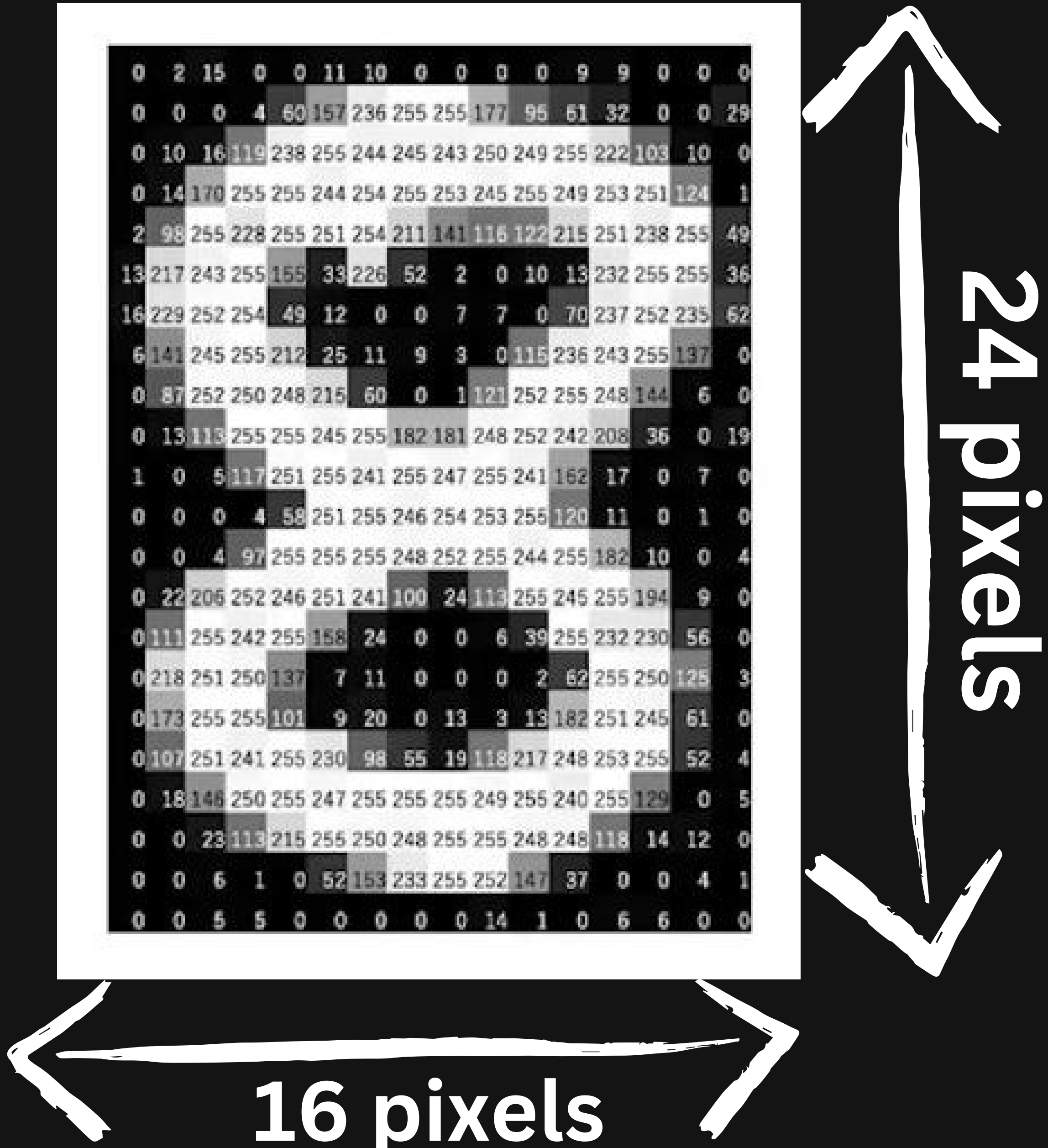
In most cases we deal with colour images which can be represented simply with their red, blue and green parts. Just like in Grayscale images, each of the pixels in the Red, Green and Blue images are converted into their corresponding numerical values depending on the intensity of the colour in the image





WHY?

Neural networks will take the numerical value from each pixel as an input. That's 24x16=384 nodes in the input layer of our neural network!


24 pixels
16 pixels

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Each of the input node will be connected to each node of the hidden layer. If we assume a hidden layer of 36 nodes, we need 384*36 = 13,824 weights and 13,824 biases!. This takes a huge amount of time to compute. Thus we need to decrease the amount of inputs without losing out the important features of the image.

Now for a different image of 8, the previous network will not be as accurate because instead of capturing the features of the image, its been trained to assign weights and biases for each pixel.

Hence traditional neural networks are ineffective for image detection.

The solution? $\longrightarrow$ **CNNs**

THE CONVOLUTION OPERATION

Kernel

A kernel is a small matrix which extracts the required features from the given image. The kernel is much smaller than the input image and have different kernels for different tasks like blurring, sharpening or edge detection.

0	1	2
2	2	0
0	1	2

- Typically kernels of size 3x3
- The values of the kernels will be updated by back propagation

a small part from the image matrix

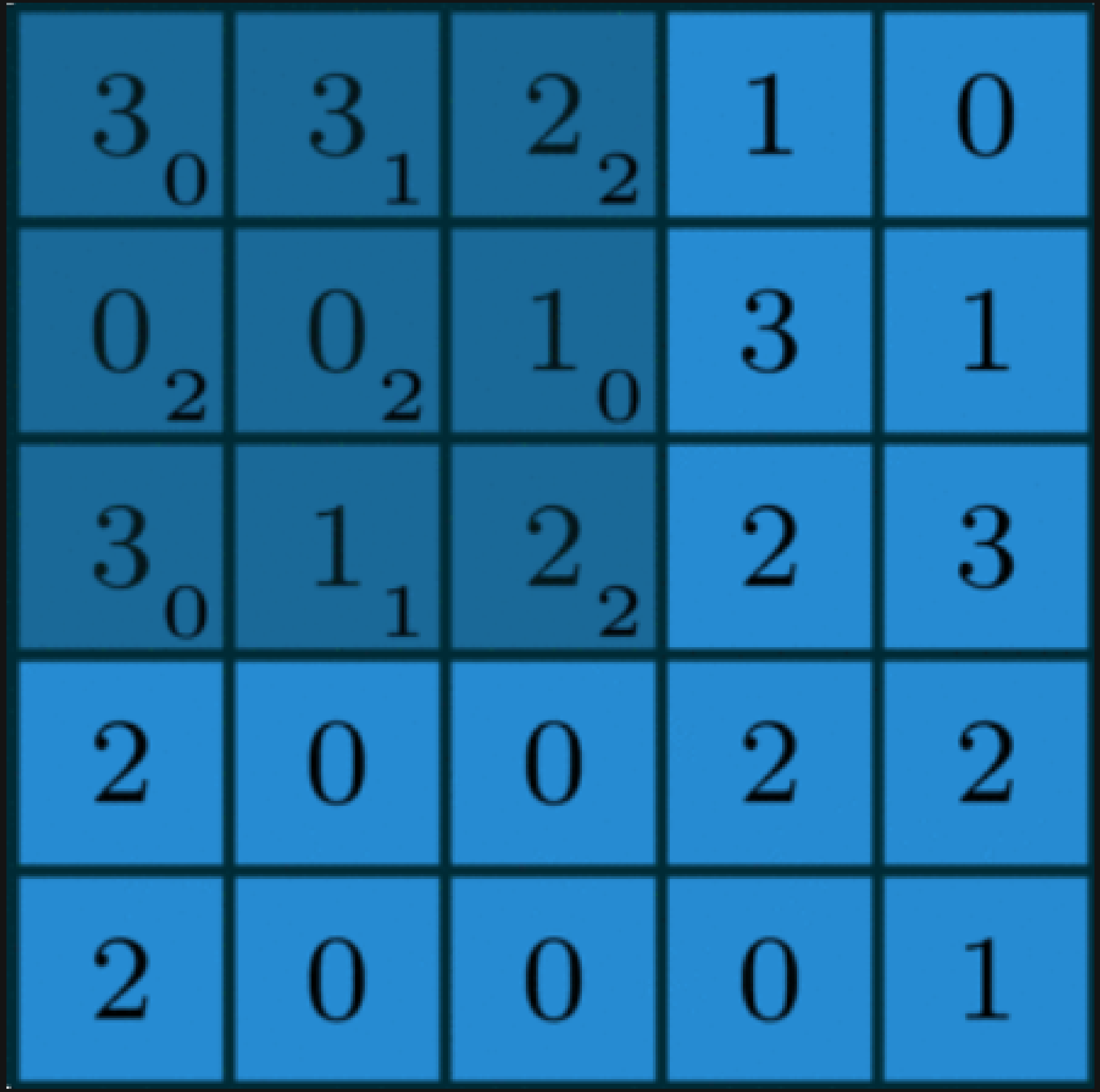
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

*

kernel

0	1	2
2	2	0
0	1	2

=



3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

$$\begin{aligned}&3(0)+3(1)+2(2)+\\&0(2)+0(2)+1(0)\qquad = \mathbf{12}\\&3(0)+1(1)+2(2)\end{aligned}$$

a small part from the image matrix

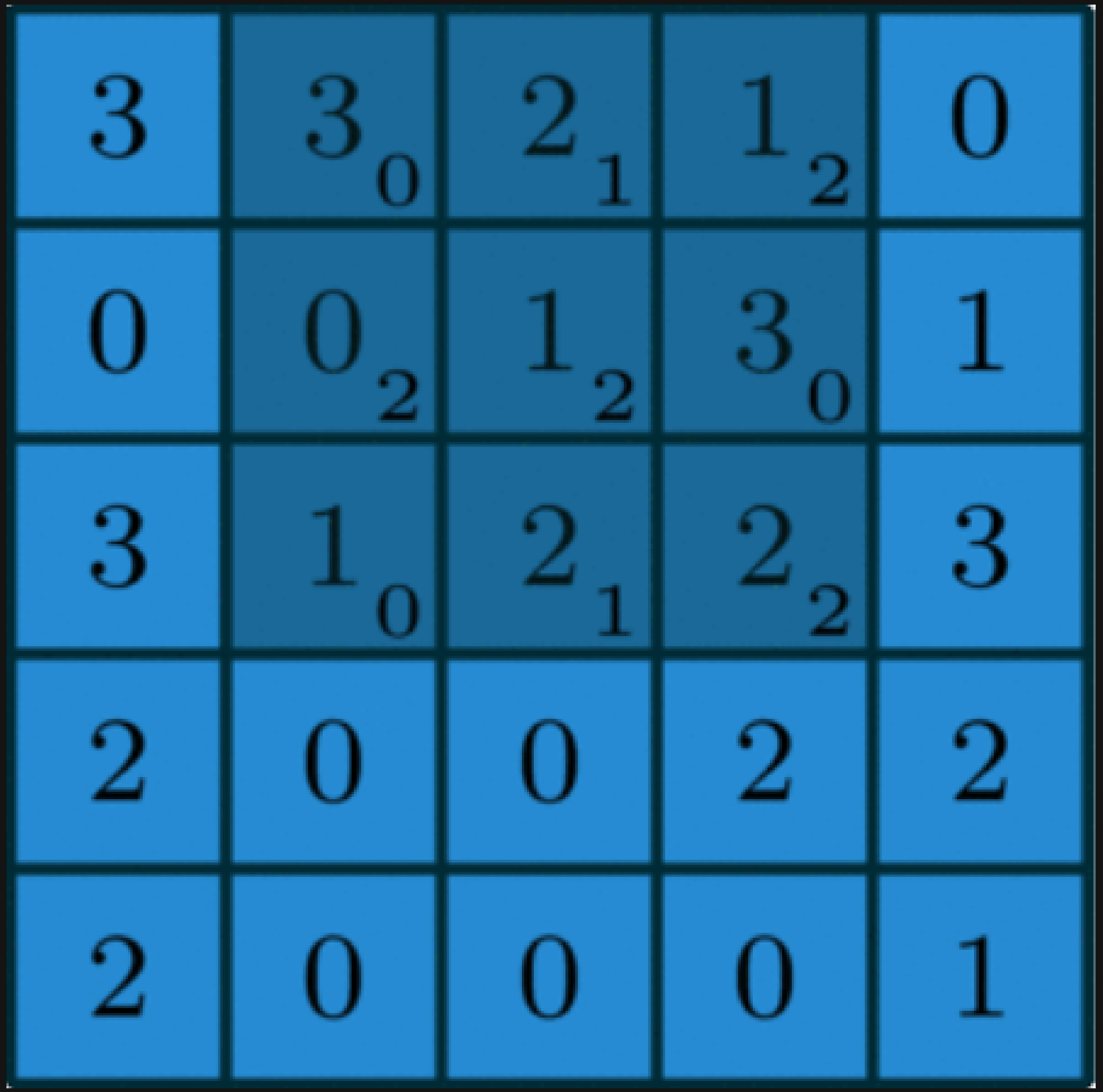
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

*

kernel

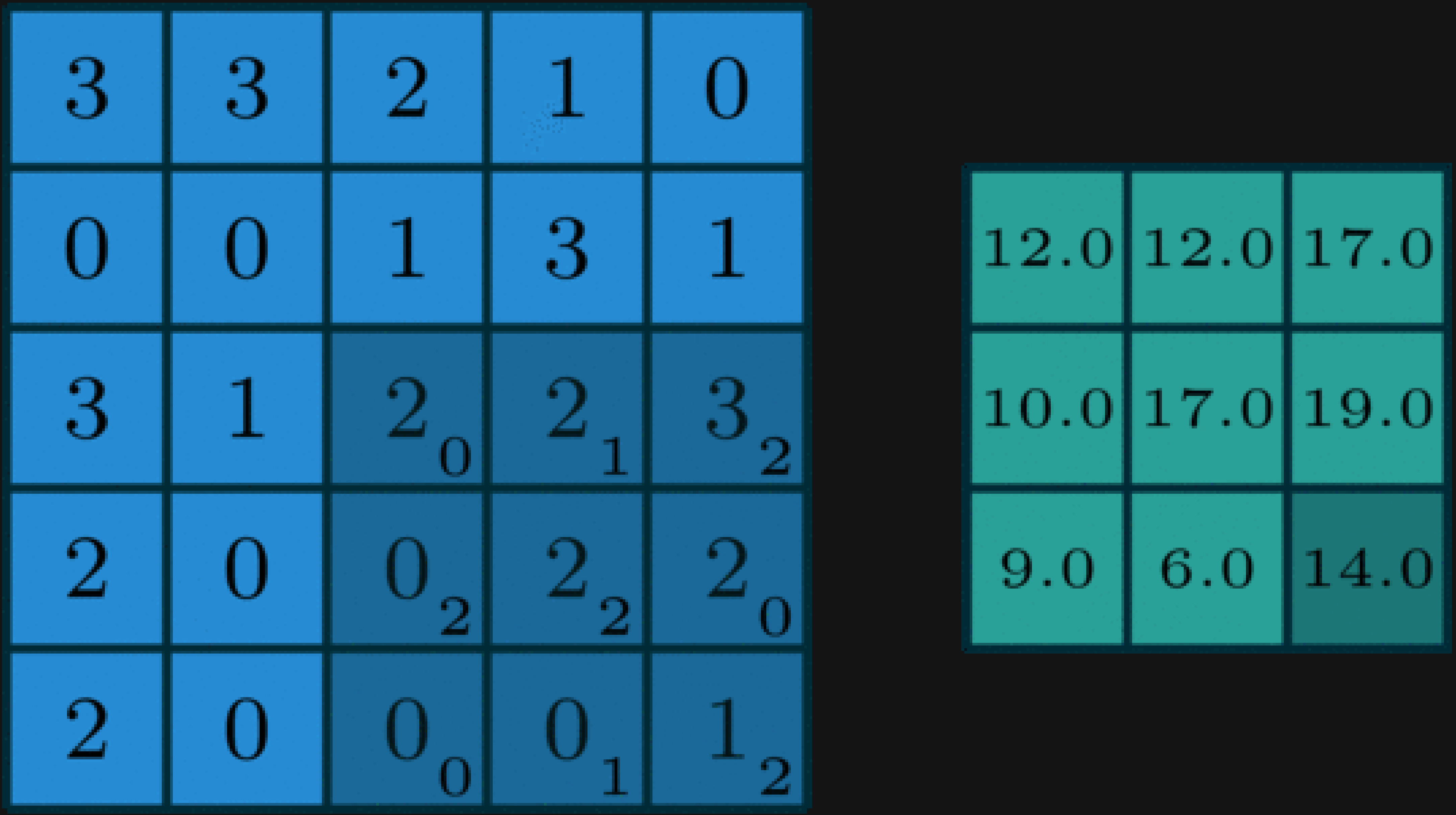
0	1	2
2	2	0
0	1	2

=



3	3_0	2_1	1_2	0
0	0_2	1_2	3_0	1
3	1_0	2_1	2_2	3
2	0	0	2	2
2	0	0	0	1

3(0)+2(1)+1(2)+
0(2)+1(2)+3(0)   **= 12**
1(0)+2(1)+2(2)



$$\begin{array}{|c|c|c|c|c|}\hline 3 & 3 & 2 & 1 & 0 \\ \hline 0 & 0 & 1 & 3 & 1 \\ \hline 3 & 1 & 2_0 & 2_1 & 3_2 \\ \hline 2 & 0 & 0_2 & 2_2 & 2_0 \\ \hline 2 & 0 & 0_0 & 0_1 & 1_2 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|}\hline 12.0 & 12.0 & 17.0 \\ \hline 10.0 & 17.0 & 19.0 \\ \hline 9.0 & 6.0 & 14.0 \\ \hline \end{array}$$

THE CONVOLUTION OPERATOR

$$\begin{bmatrix} 4 & 9 & 0 \\ 8 & 6 & 1 \\ 7 & 2 & 34 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{matrix} 4(1) + 9(0) + 0(1) \\ + 8(0) + 6(2) + 1(0) \\ + 7(1) + 2(0) + 34(1) \end{matrix}$$

$$= 57$$

CONVOLUTION OF AN IMAGE

50(1) + 165(0) + 67(-1) + ...

$$\begin{bmatrix} 50 & 165 & 67 & 0 \\ 94 & 23 & 88 & 12 \\ 178 & 56 & 90 & 64 \\ 234 & 204 & 78 & 123 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 83 & 179 \\ 338 & 80 \end{bmatrix}$$

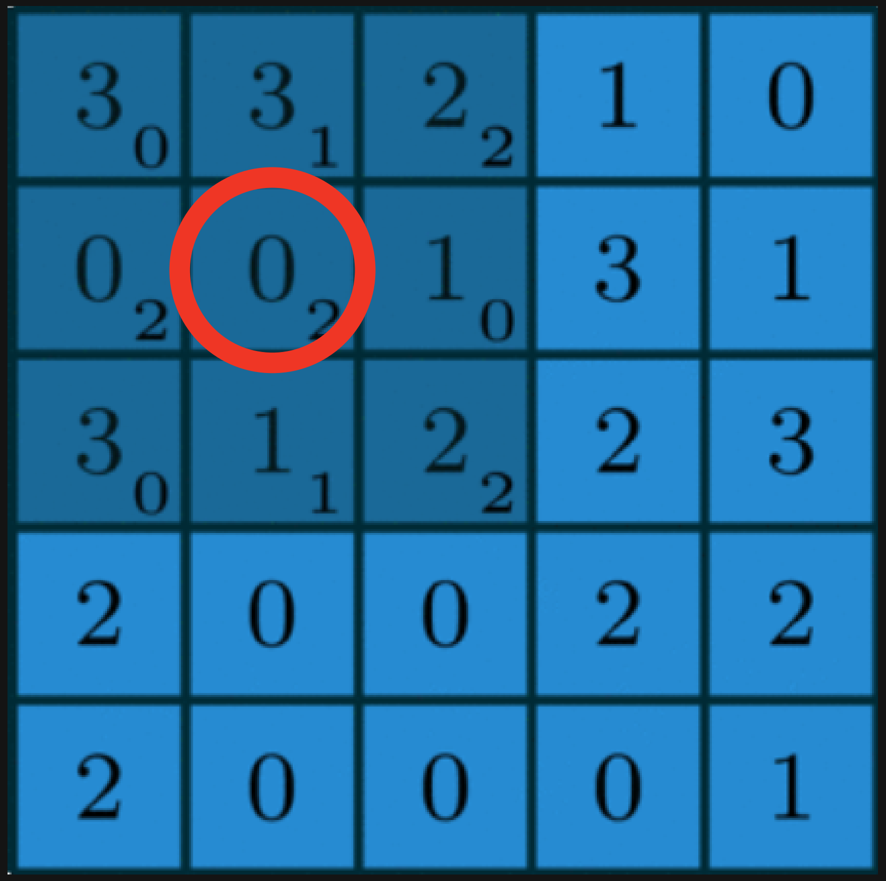
Image

Kernel

The new image

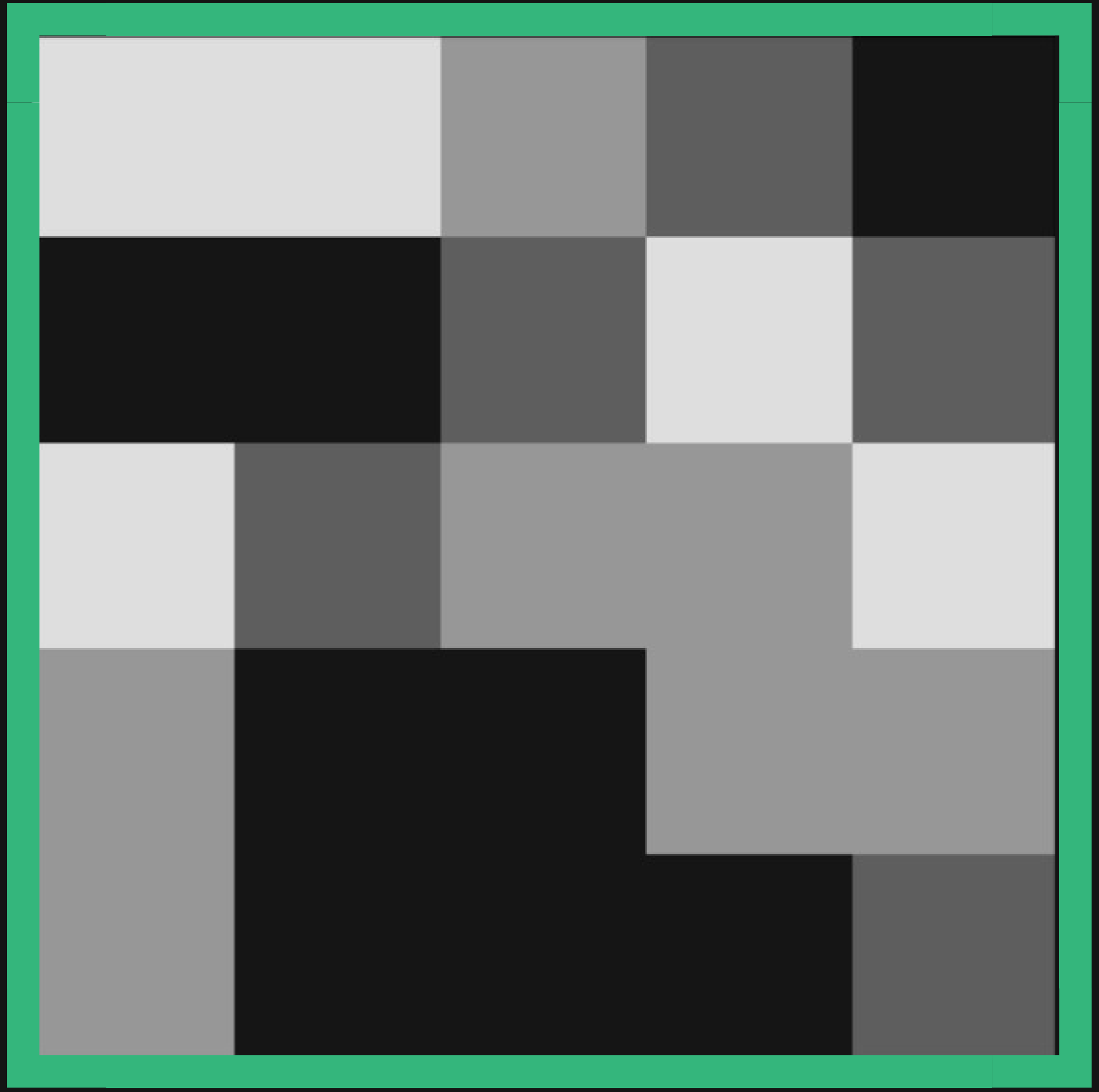
WHY IS CONVOLUTION NEEDED FOR IMAGE DETECTION?

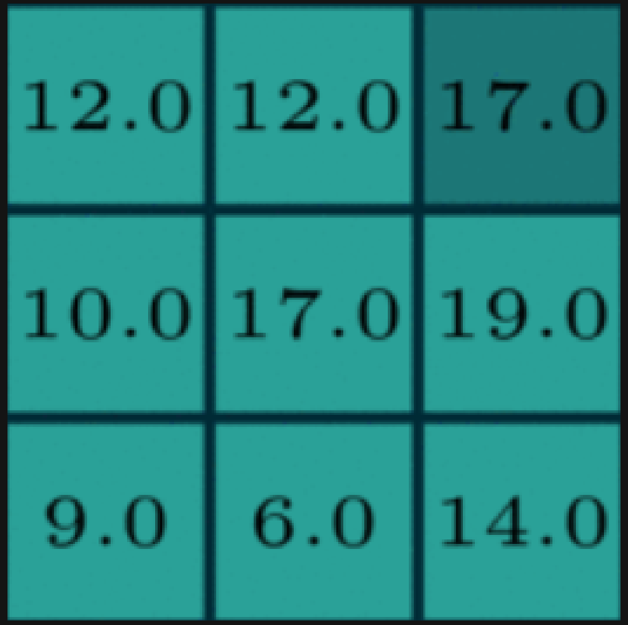
By Convolving the Image matrix with the given kernel we extract the information from each pixel along with the influence its neighboring pixels have. Doing this allows us to extract the features of the image.



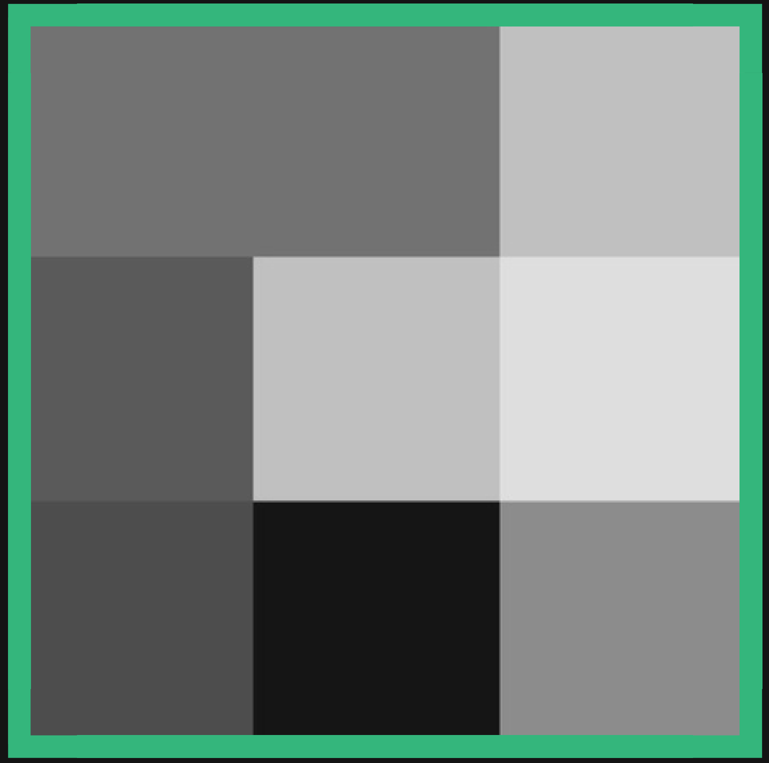
by applying the convolution we are focusing on the central pixel and extract its info along with it neighbours

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1



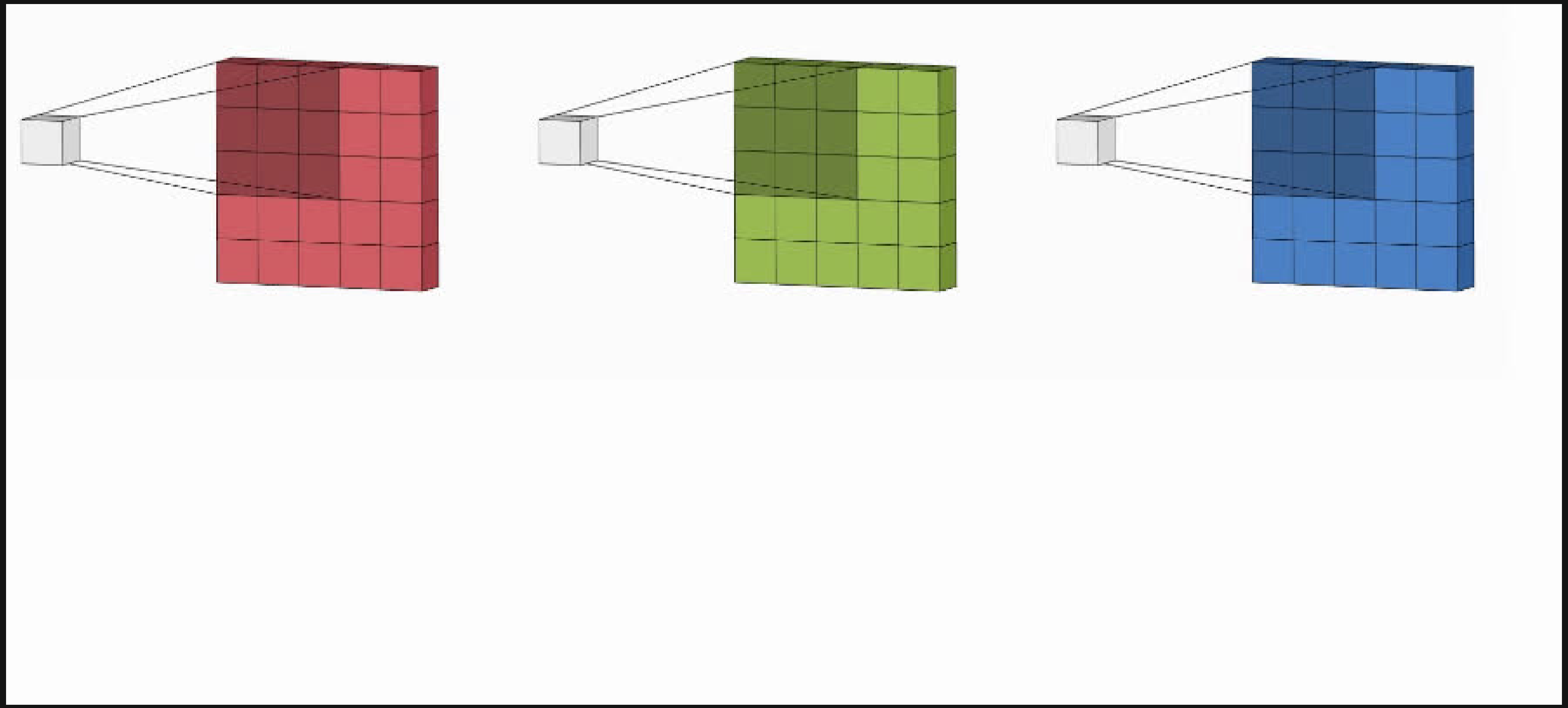


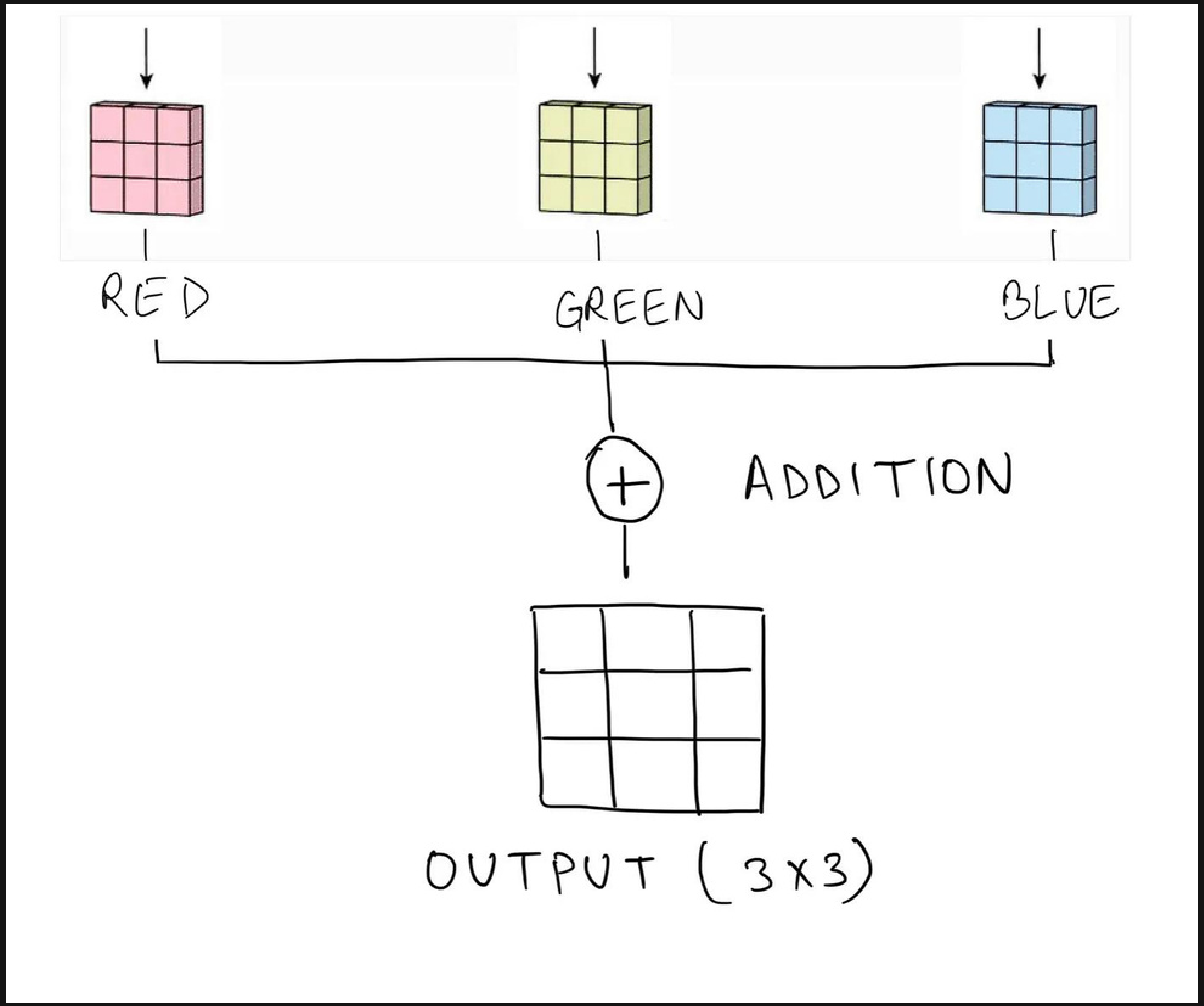
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0



FILTERS VS KERNELS

Often in computer visiion , the terms kernel and filter are used interchangeably but there is a slight difference between them. Filters are groups of kernels and a reused in convoling RGB images, which have three channels (red, green, blue) instead of just one. IN each filter the kernels might be the same or sifferennt according to the features we need to extract.





The resultant three matrices are then combined using matrix addition to get a single output.

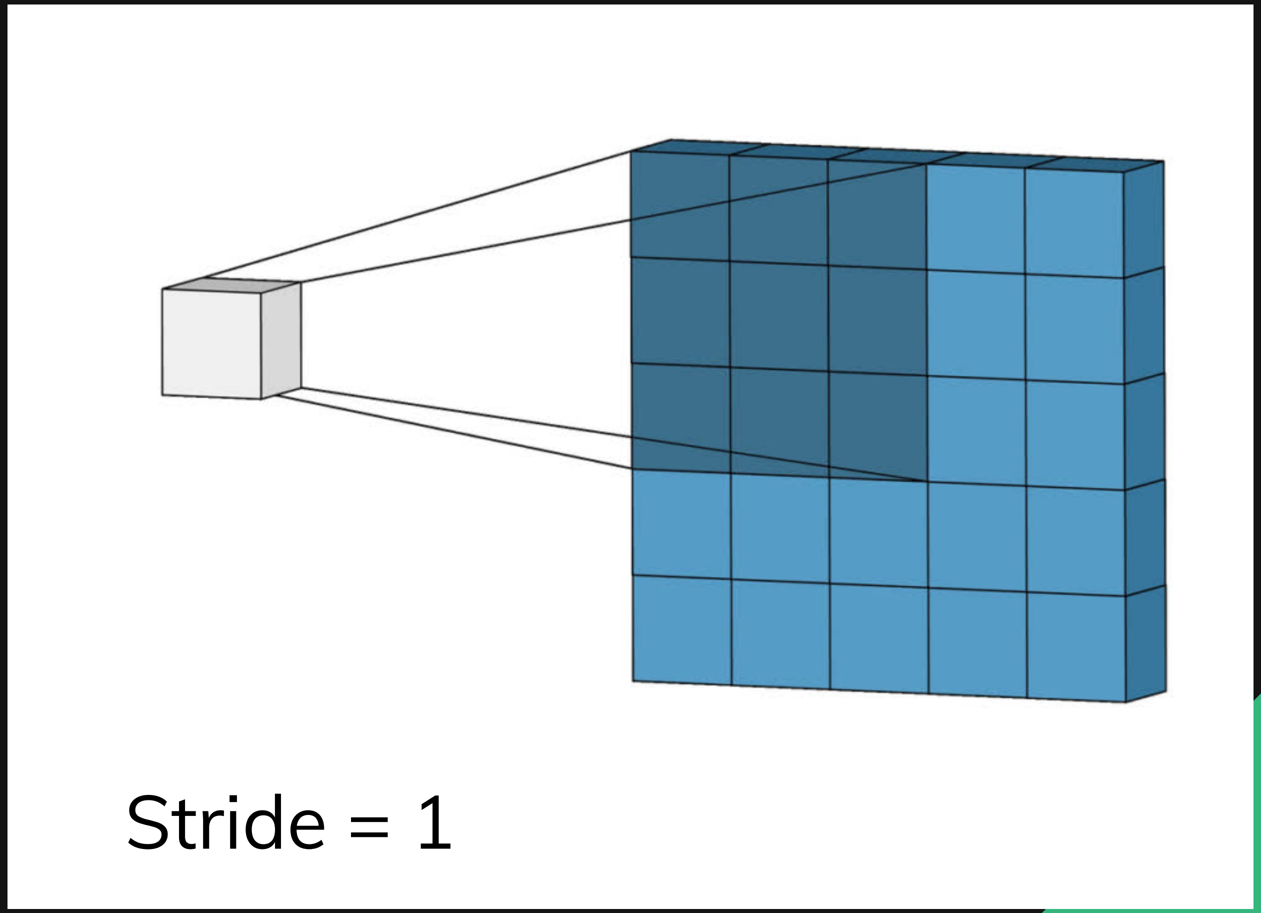
Remember that this is only for RGB images. in a gray scale a image filer is equal to the kernel because we only have one channel of input

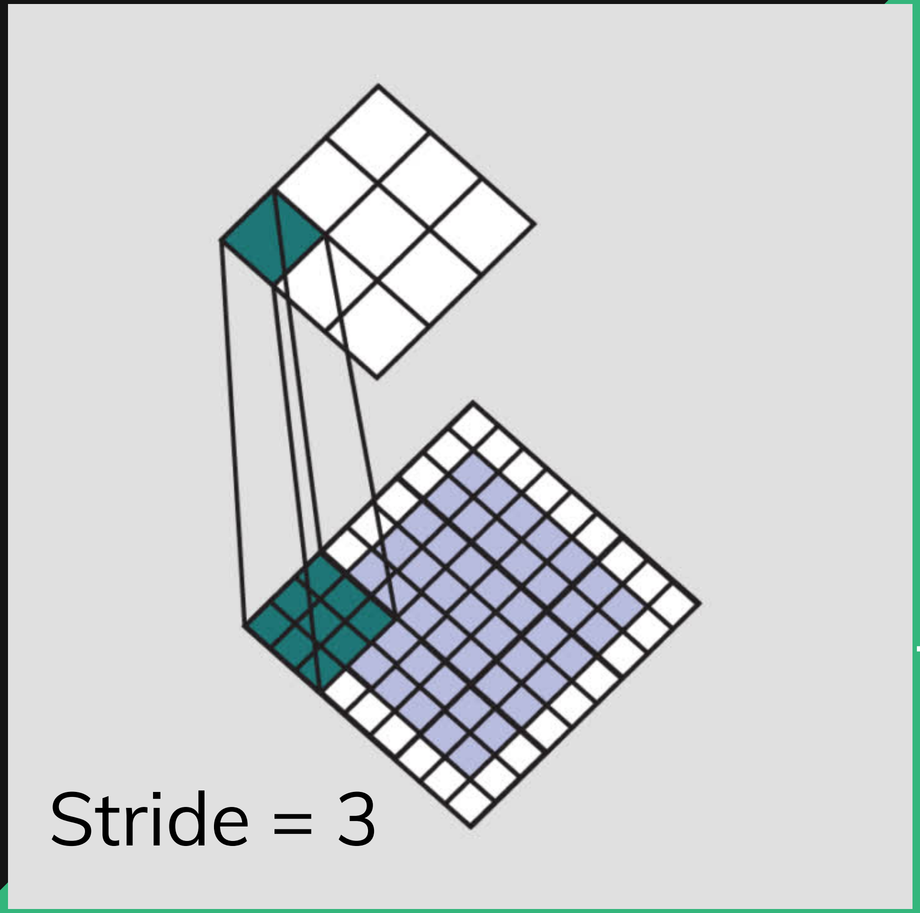
Few more terminologies in CNNs

STRIDE

Stride is the no. of steps we move the kernel each time we convolve.

Normally we use a stride of 1, so every possible section of the image is taken.


Stride = 1

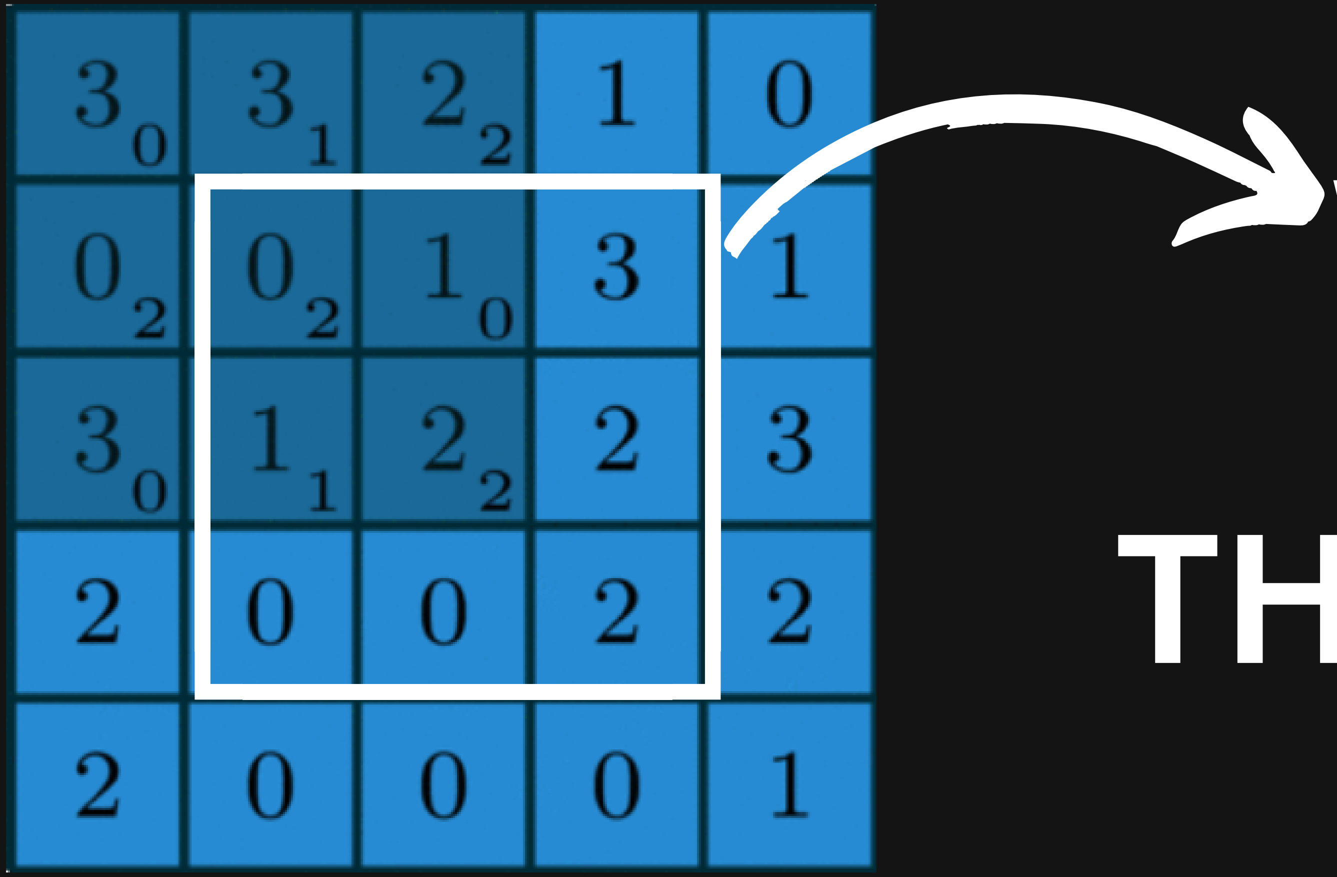

Stride = 3

a different stride value might be
taken according to the need of the CNN model

PADDING

During the convolution operation we extract the information of each pixel by applying a kernel over it such the required pixel is at the middle of the kernel.

However, we end up losing the data on the edges of the image because we are unable to apply the kernel over those pixels.



we retain the information of these pixels but lose out of the edge pixels

THE SOLUTION?

Padding the image

Padding is adding a layer of zeros around the image. It will help in convolving the entire image without distorting any information as the zeros will have no effect on the convolution

kernel

$$\begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 3 & 3 & 2 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 3 & 1 & 0 \\ \hline 0 & 3 & 1 & 2 & 2 & 3 & 0 \\ \hline 0 & 2 & 0 & 0 & 2 & 2 & 0 \\ \hline 0 & 2 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 2 & 2 & 0 \\ \hline 0 & 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|}\hline 6 & 14 & 17 & 11 & 3 \\ \hline 14 & 12 & 12 & 17 & 11 \\ \hline 8 & 10 & 17 & 19 & 13 \\ \hline 11 & 9 & 6 & 14 & 12 \\ \hline 6 & 4 & 4 & 6 & 4 \\ \hline \end{array}$$

thus by padding the image we have retained its size and the information of the edges

PADDING

0	0	0	0	0	0
0	50	165	67	0	0
0	94	23	88	12	0
0	178	56	90	64	0
0	234	204	78	123	0
0	0	0	0	0	0

1	0	-1
2	0	-2
1	0	-1

-353	28	341	222
-267	83	179	333
-343	338	80	346
-472	400	162	246

We add zeros around the image

The kernel is the same

Now the size is the same and entire information of the image is retained

Congratulations! You just learnt the basics of a Convolutional Layer!

Before we move forward, let's look at syntax of convolutions in PyTorch:

```
nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, stride=1 padding=1)
```

No of input channels (depends on the colours we have

No of output channels (depends on the number of filters we use)

kernel size

stride

padding

Also here's a simple formula

It's just for reference

dont worry about memorizing this

$$o = \left\lfloor \frac{i - k + 2 \cdot p}{s} \right\rfloor + 1$$

where

o is the output size

i is the input size

k is kernel size

p is padding

s is stride

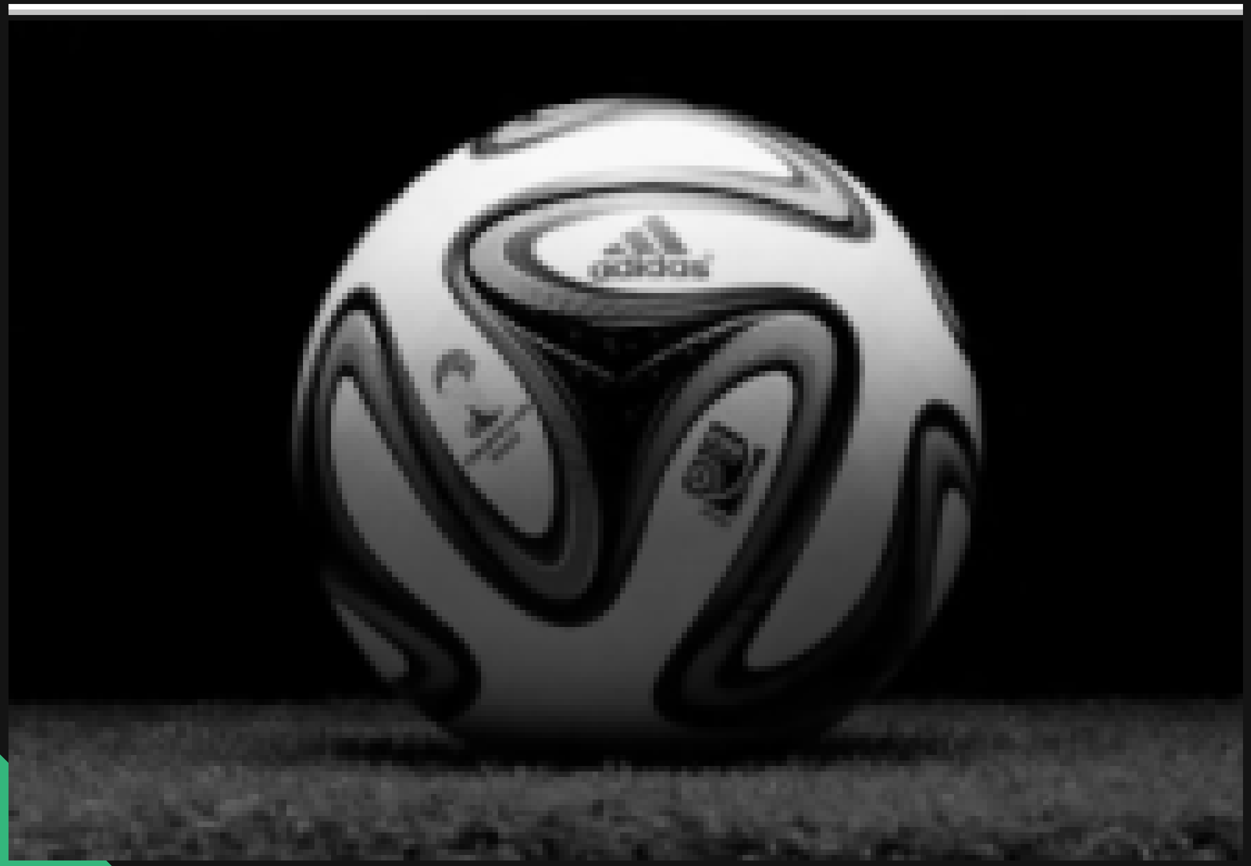
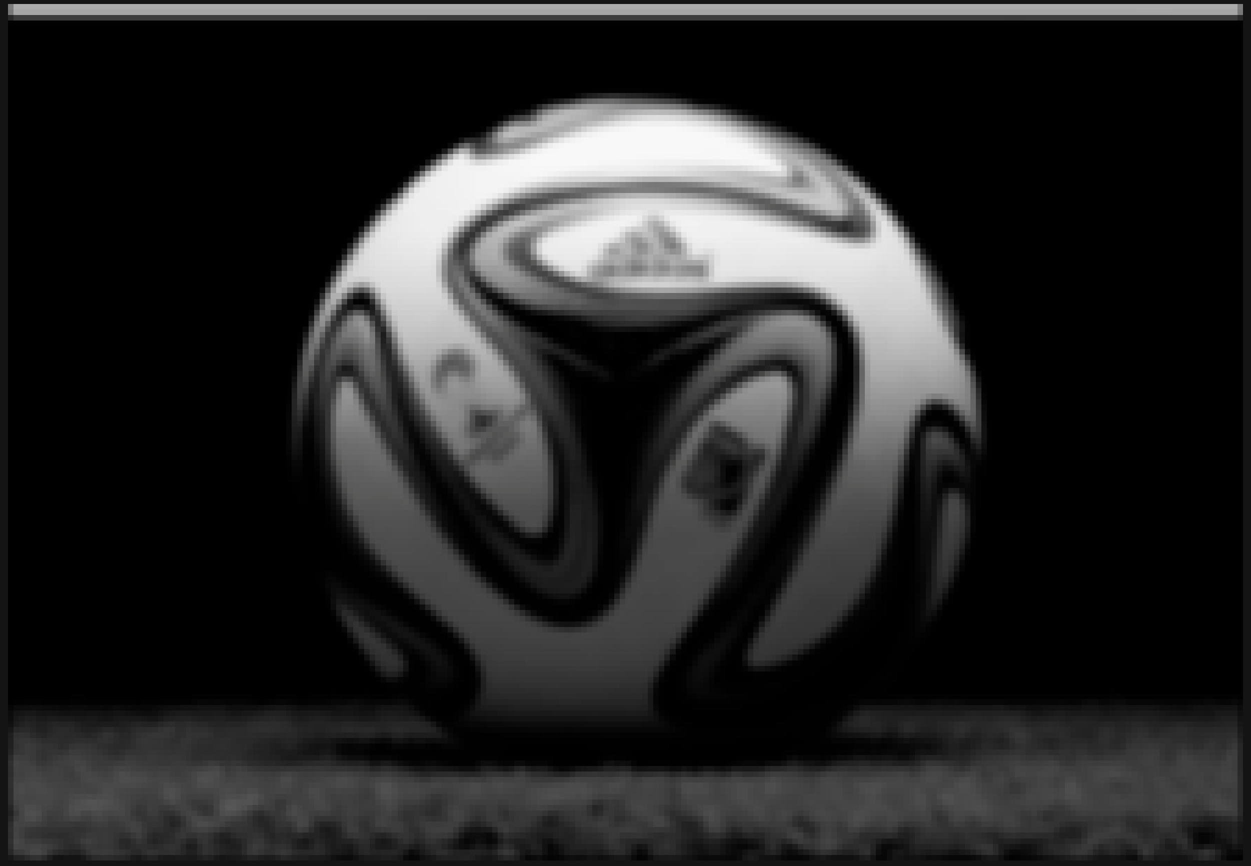
Some Examples of Kernels

GAUSSIAN BLUR

$$\begin{bmatrix} .075 & .124 & .075 \\ .124 & .204 & .124 \\ .075 & .124 & .075 \end{bmatrix}$$

lets apply gaussian blur on this image!

Gaussian blurring is a technique that blurs a pixel giving more weight to middle pixel. Its similar to the gaussian curve. Blurring the image reduces computation time by filering out noise(unwanted data) in the image while reataing the context

Blurring the image retained important information
but also got rid of unnecessary noise

THE SOBEL OPERATOR

Vertical Edges

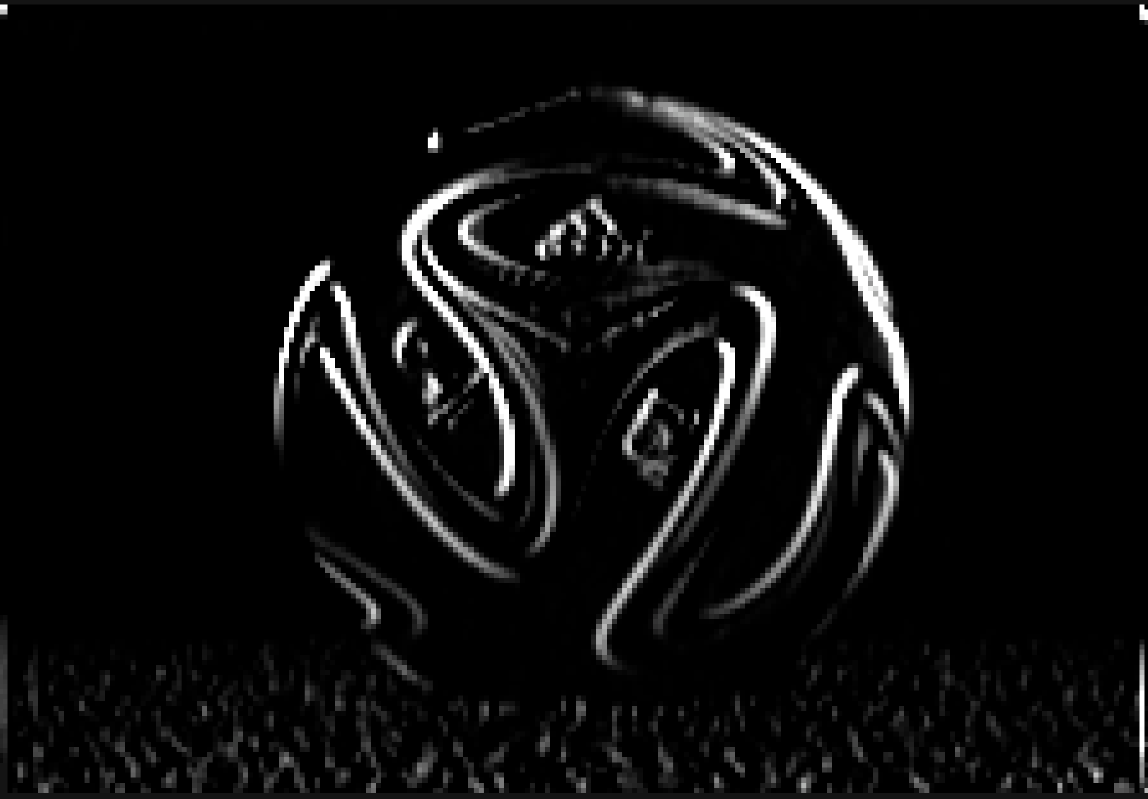
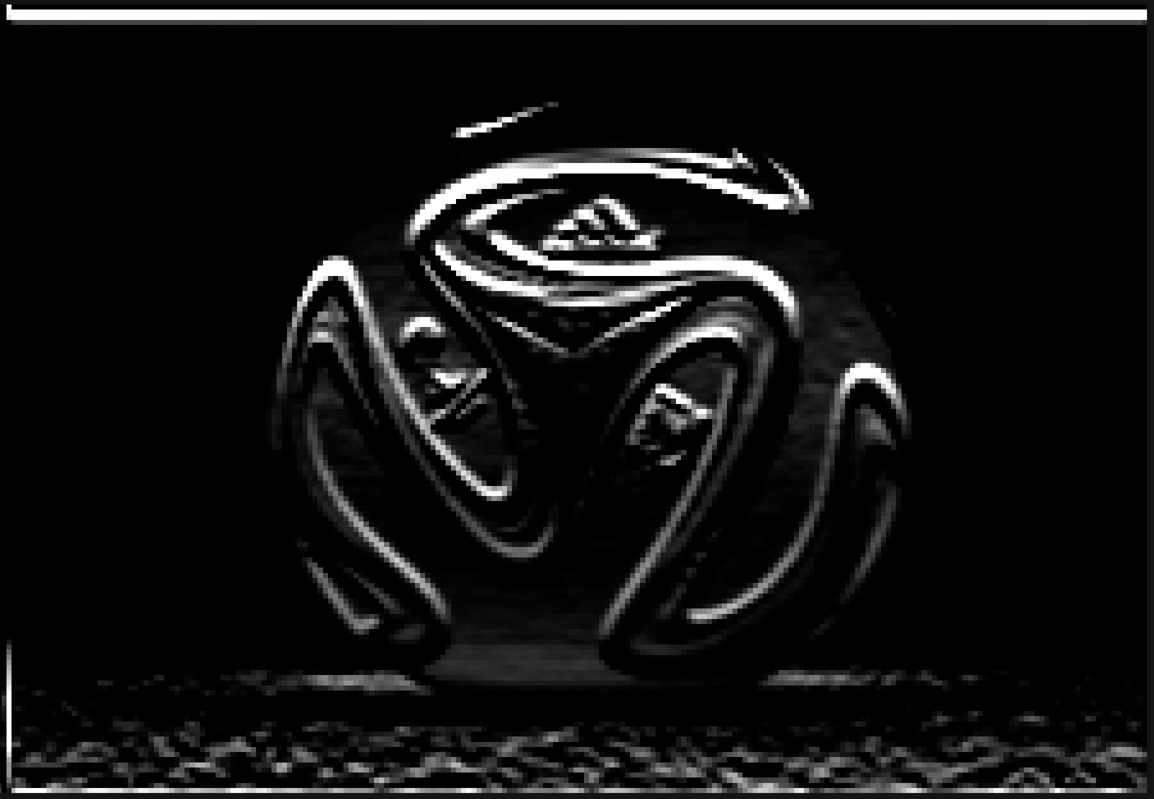
1	0	-1
2	0	-2
1	0	-1

Horizontal Edges

1	2	1
0	0	0
-1	-2	-1

The kernel shown before is actually the Sobel operator - it's used to find the edges in an image.

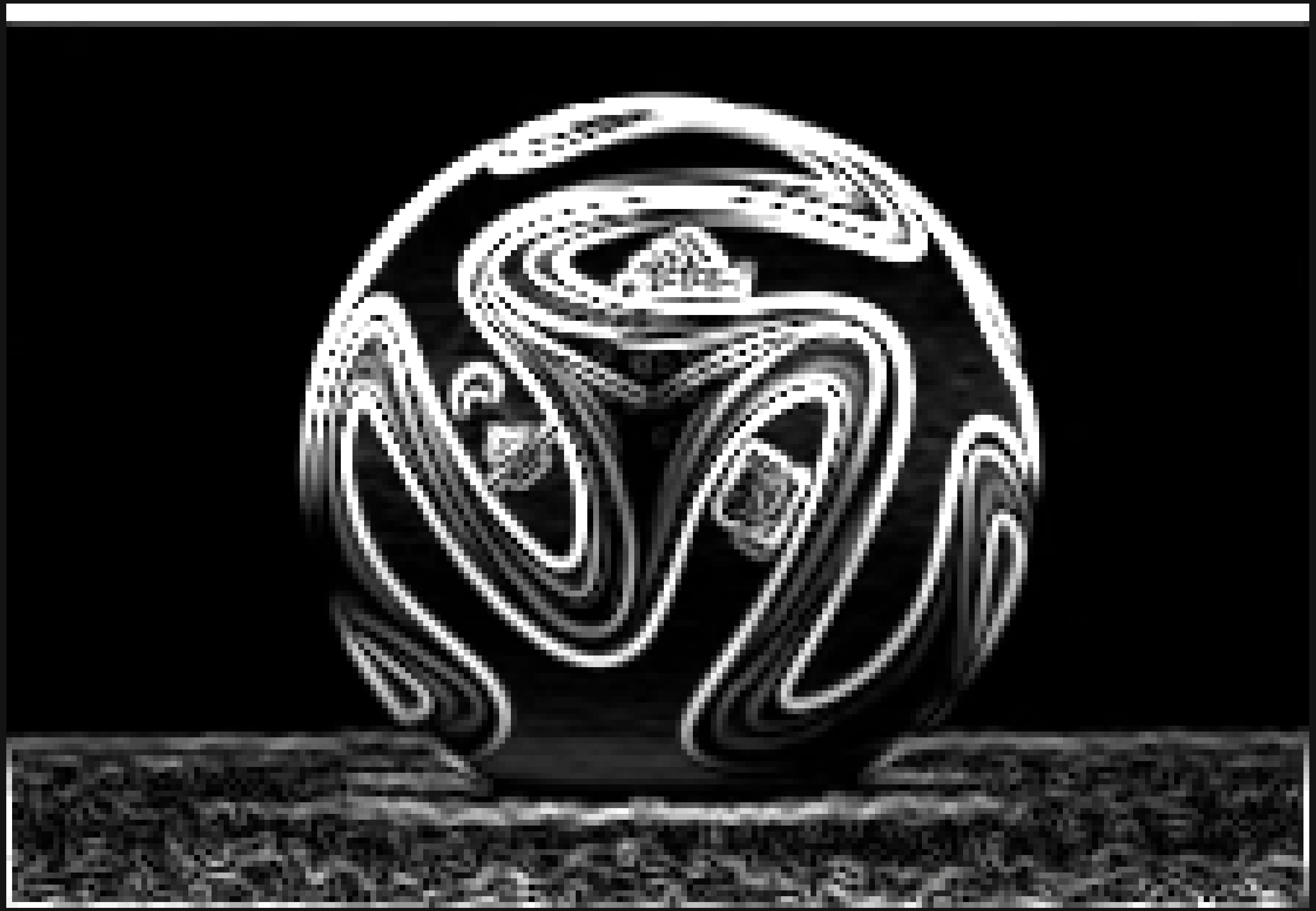
SOBEL OPERATOR

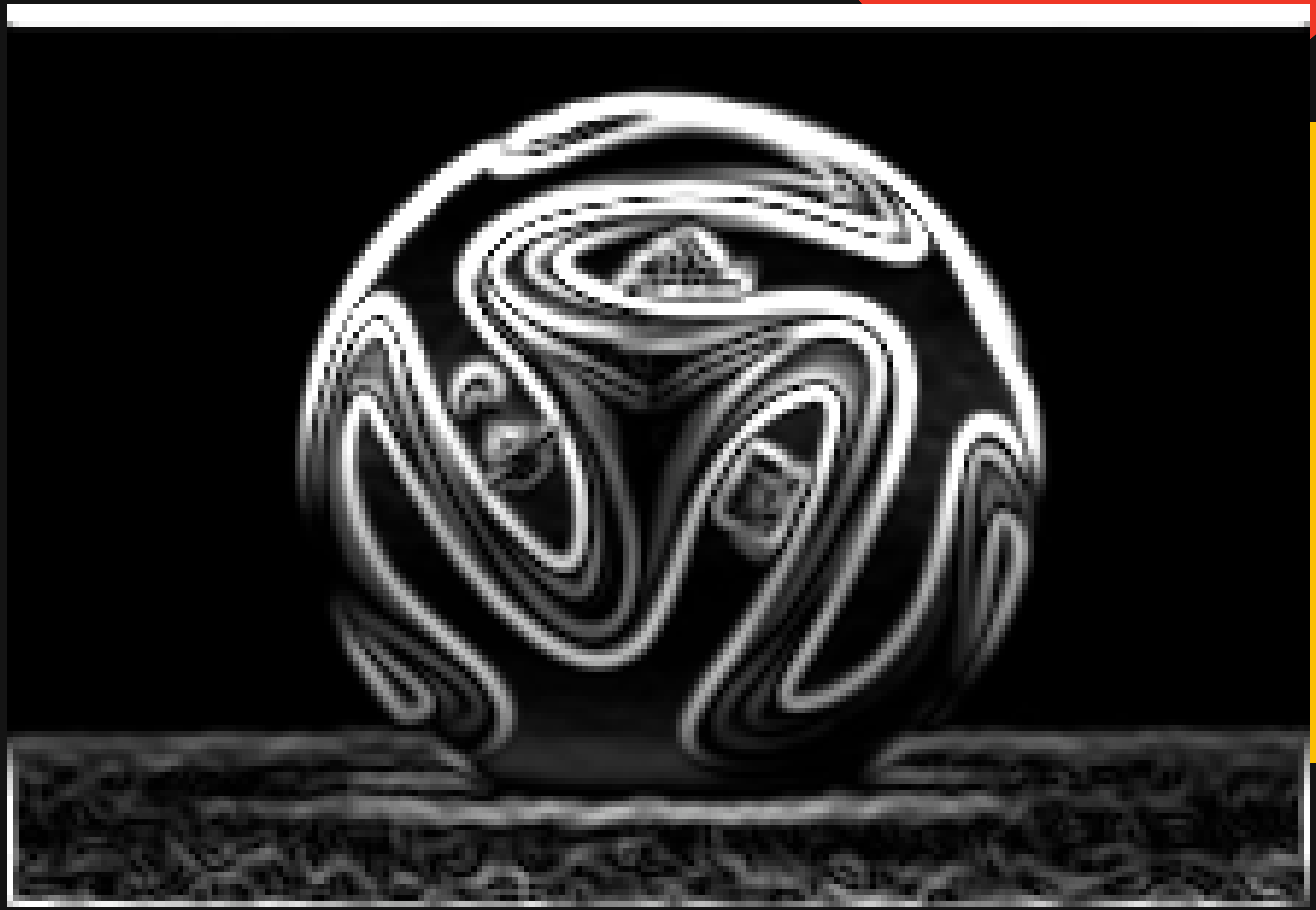
🟢 Original image 🟨 vertical sobel ◣ horizontal sobel

We then combine both outputs to get our final result

Gaussian blur also reduces noise, as you can see when you compare the edges detected in both images



Sobel
(Horizontal and Vertical Combined)



Gaussian Blur + Sobel

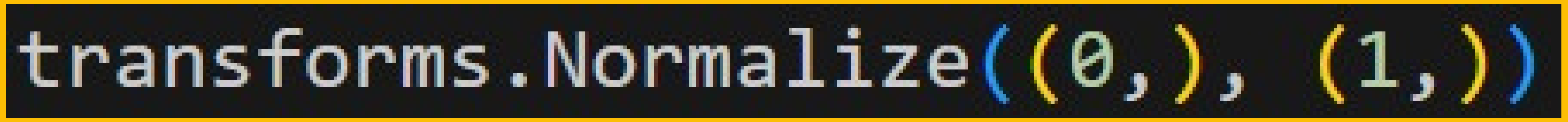
Other Layers of the CNN

Images by default have intensities ranging from 0 to 255, but neural networks usually work best if inputs are normalized, i.e, between 0 and 1. This helps in reducing the learning time and helps in preventing overfitting of the model

So, we divide our numpy array before applying convolutions.

NORMALIZATION

PyTorch code



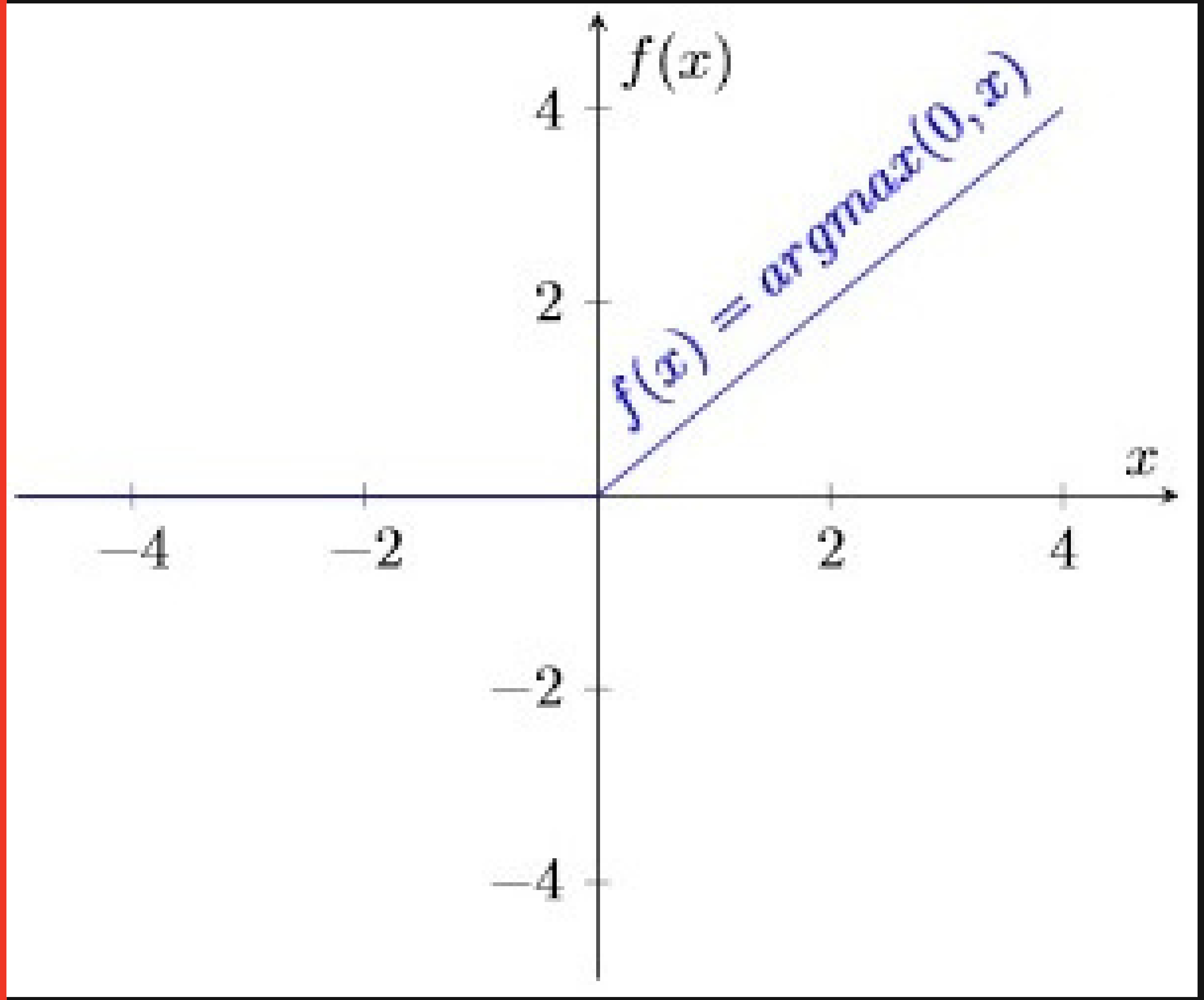
```
transforms.Normalize((0,), (1,))
```

ACTIVATION FUNCTIONS

Just like in artificial neural networks, we can make use of activation functions like ReLU here.

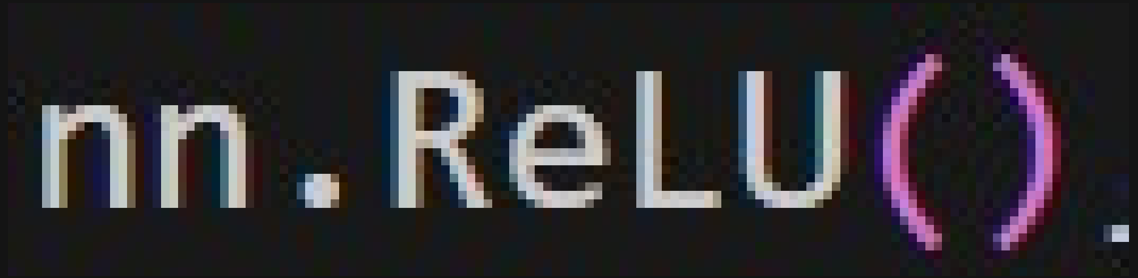
After we apply convolution and batch normalization to an image, we can apply an activation function to each pixel.

Relu



f(x) = argmax(0,x) where

x is the value of each pixel in the image array



```
nn.ReLU().
```

POOLING

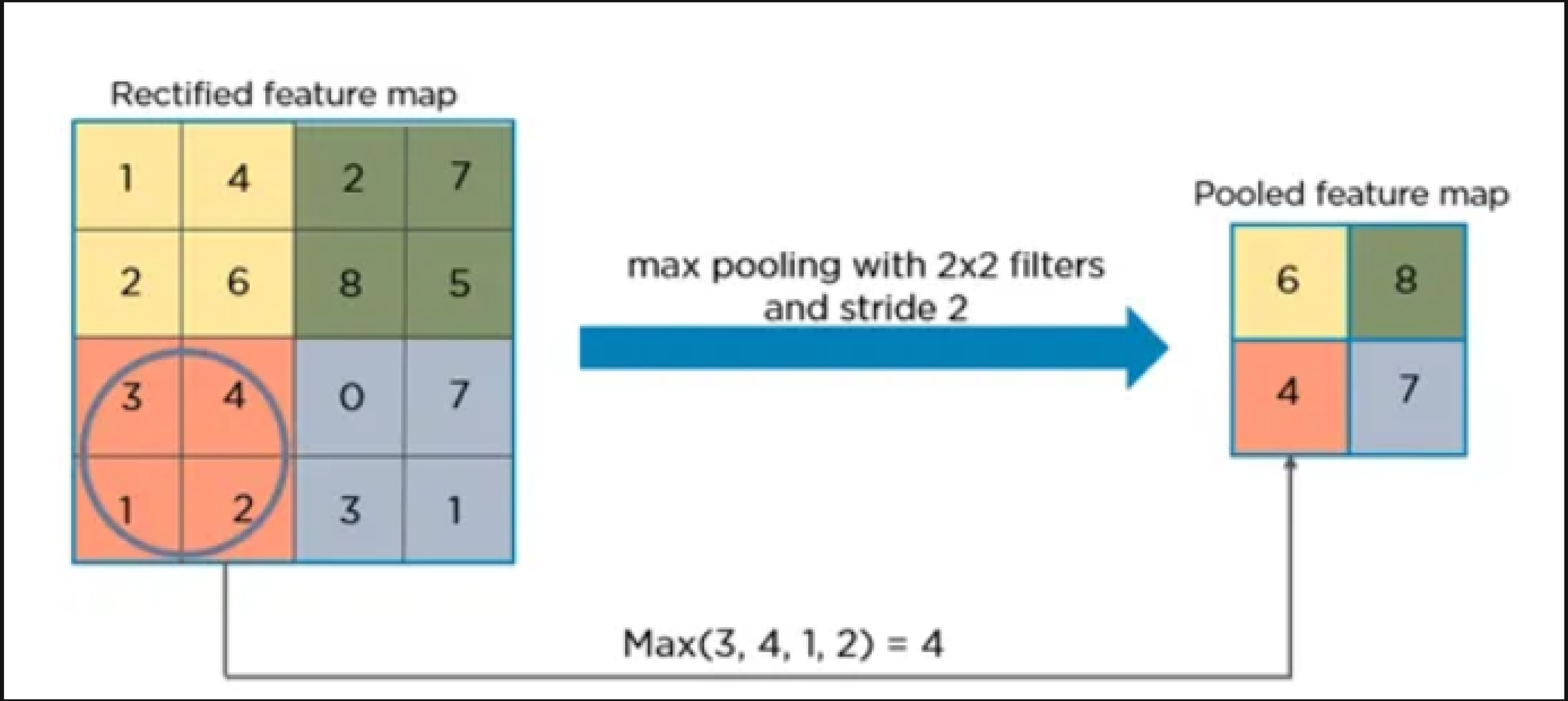
We reduce the dimensionality of the convolved image by pooling it. Normally, a stride of the size of the block is used (so there's no overlap)

This greatly reduces the size of the image, and also removes unnecessary detail (preventing overfitting)

Max Pooling

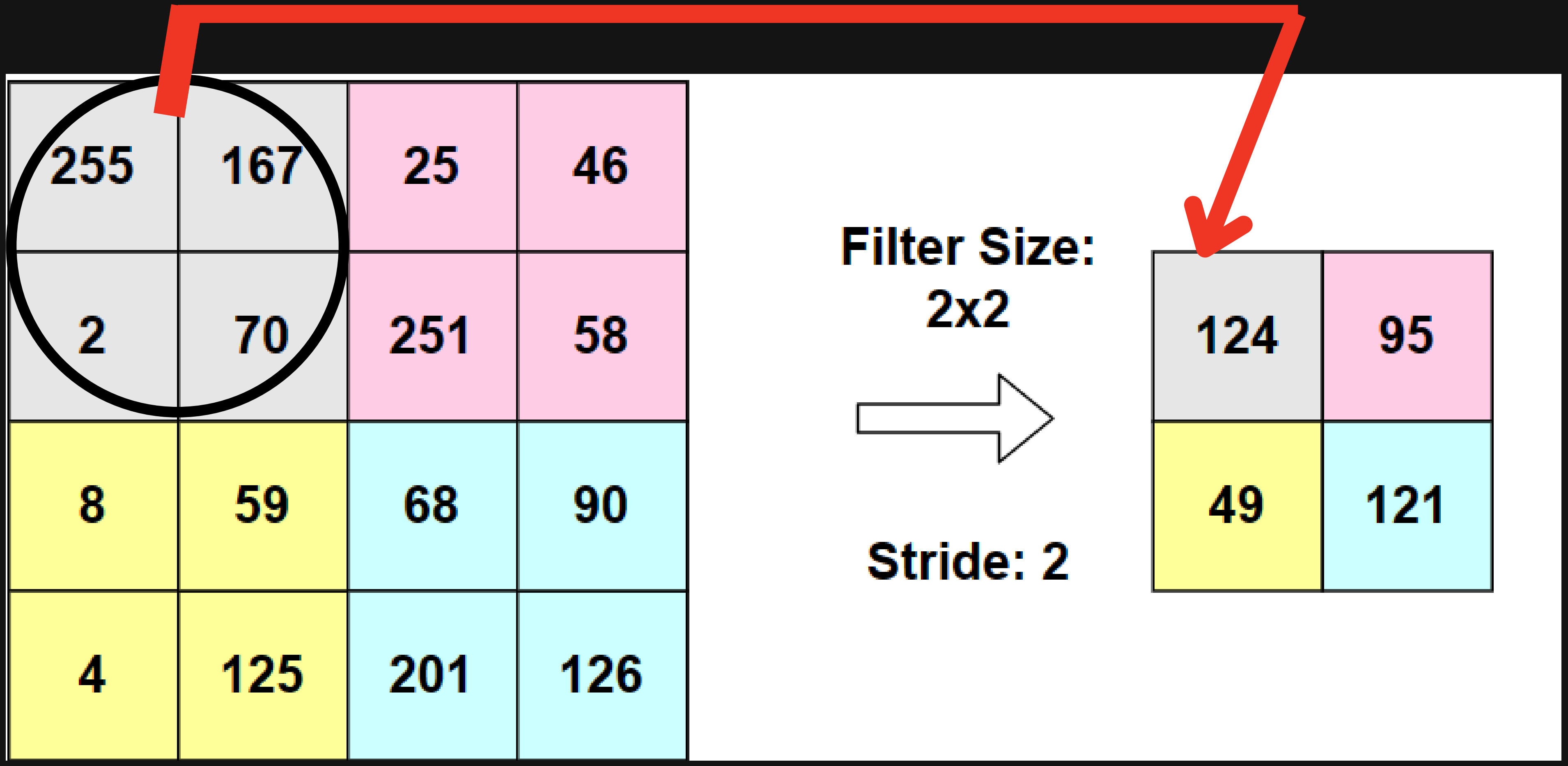
Average Pooling

MAX POOLING



Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map

AVERAGE POOLING

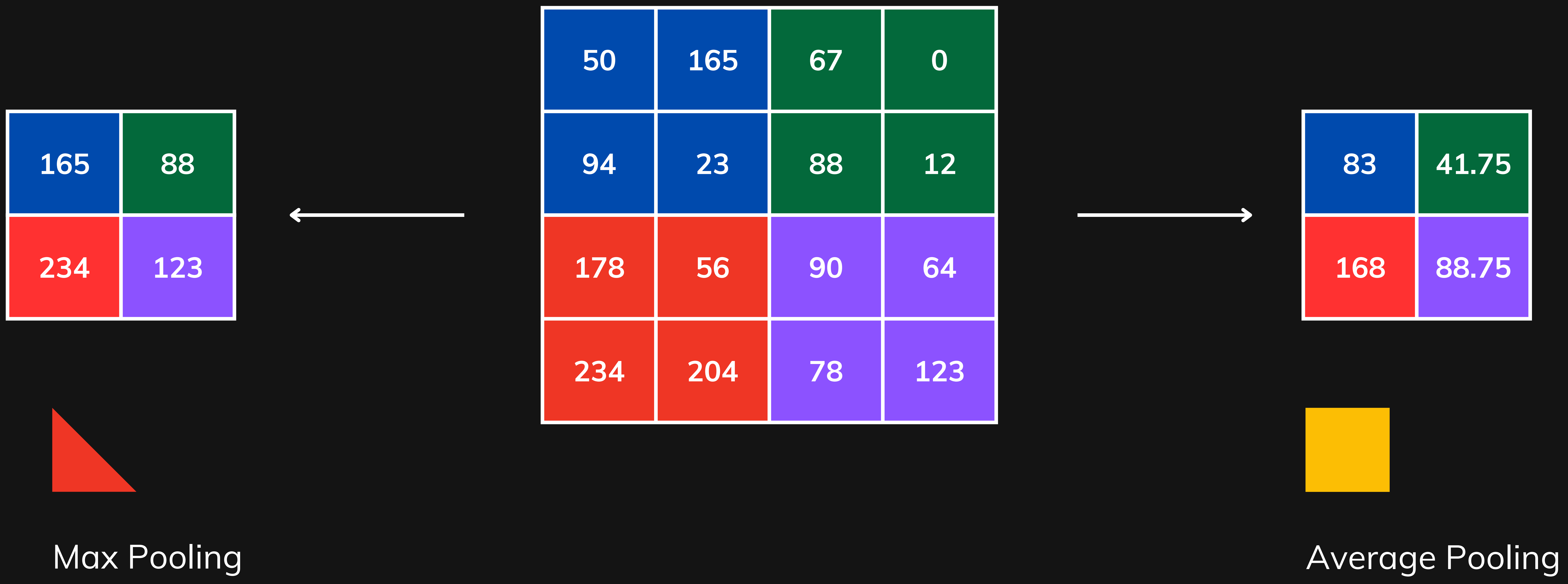


255	167	25	46
2	70	251	58
8	59	68	90
4	125	201	126

124	95
49	121

Average pooling gives us the average of the features in a filter size

TYPES OF POOLING



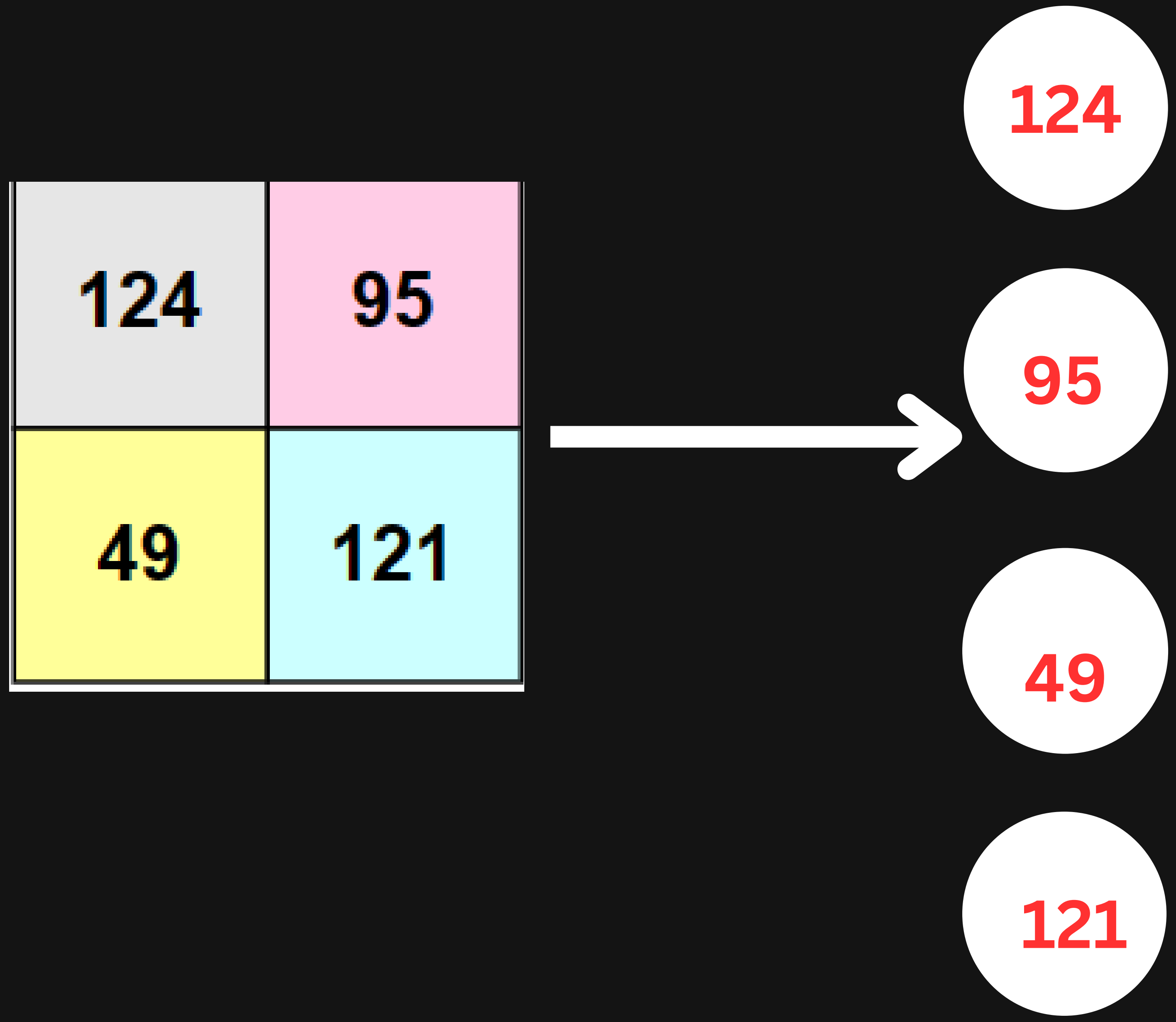
50	165	67	0
94	23	88	12
178	56	90	64
234	204	78	123

165	88
234	123

83	41.75
168	88.75

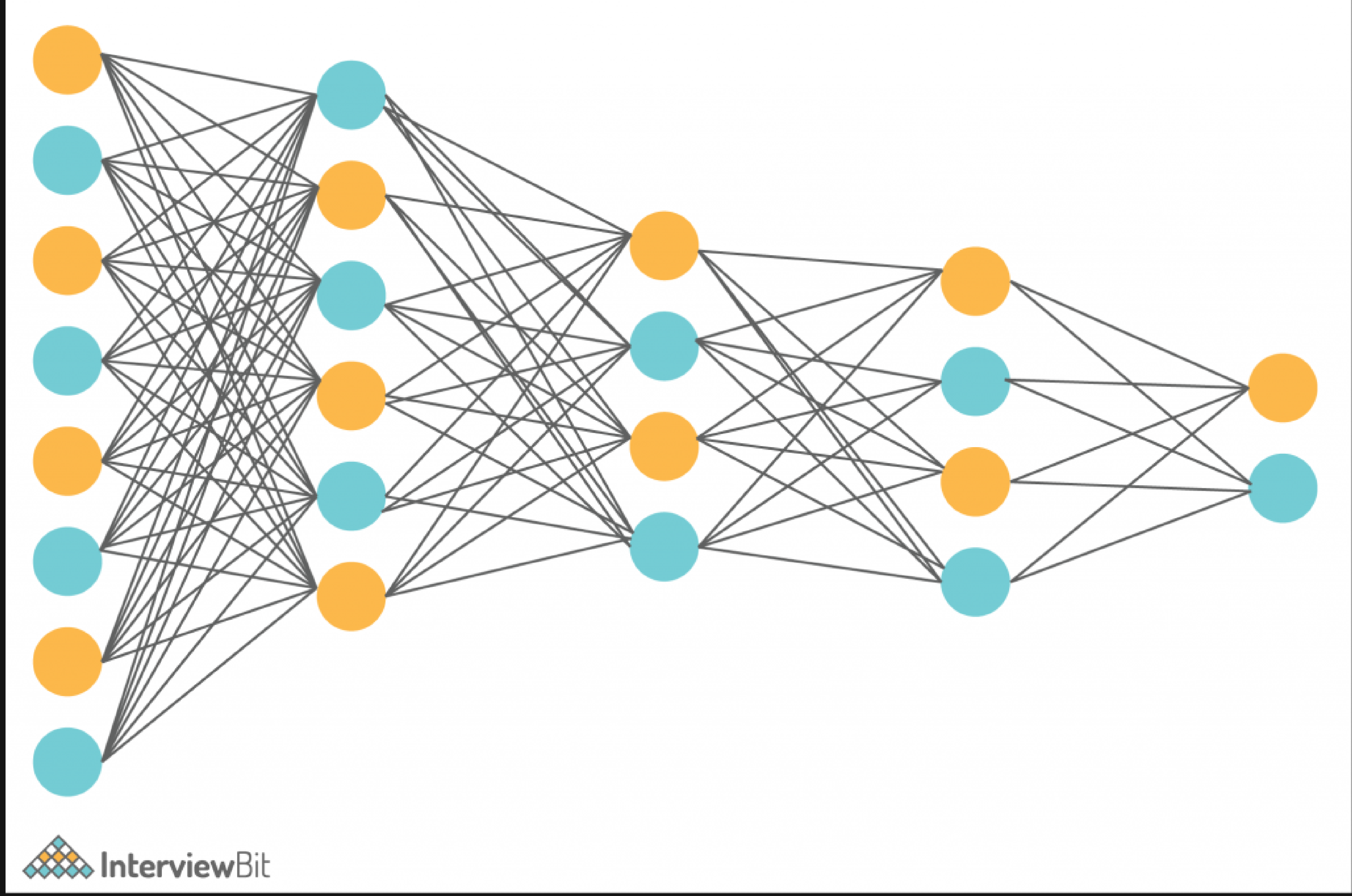
FLATTENING

The output of the final pooling layer is fed into a fully connected neural network. To do this we "flatten" the outputs



FULLY CONNECTED LAYER

Finally we input our results into a neural network to run the classification

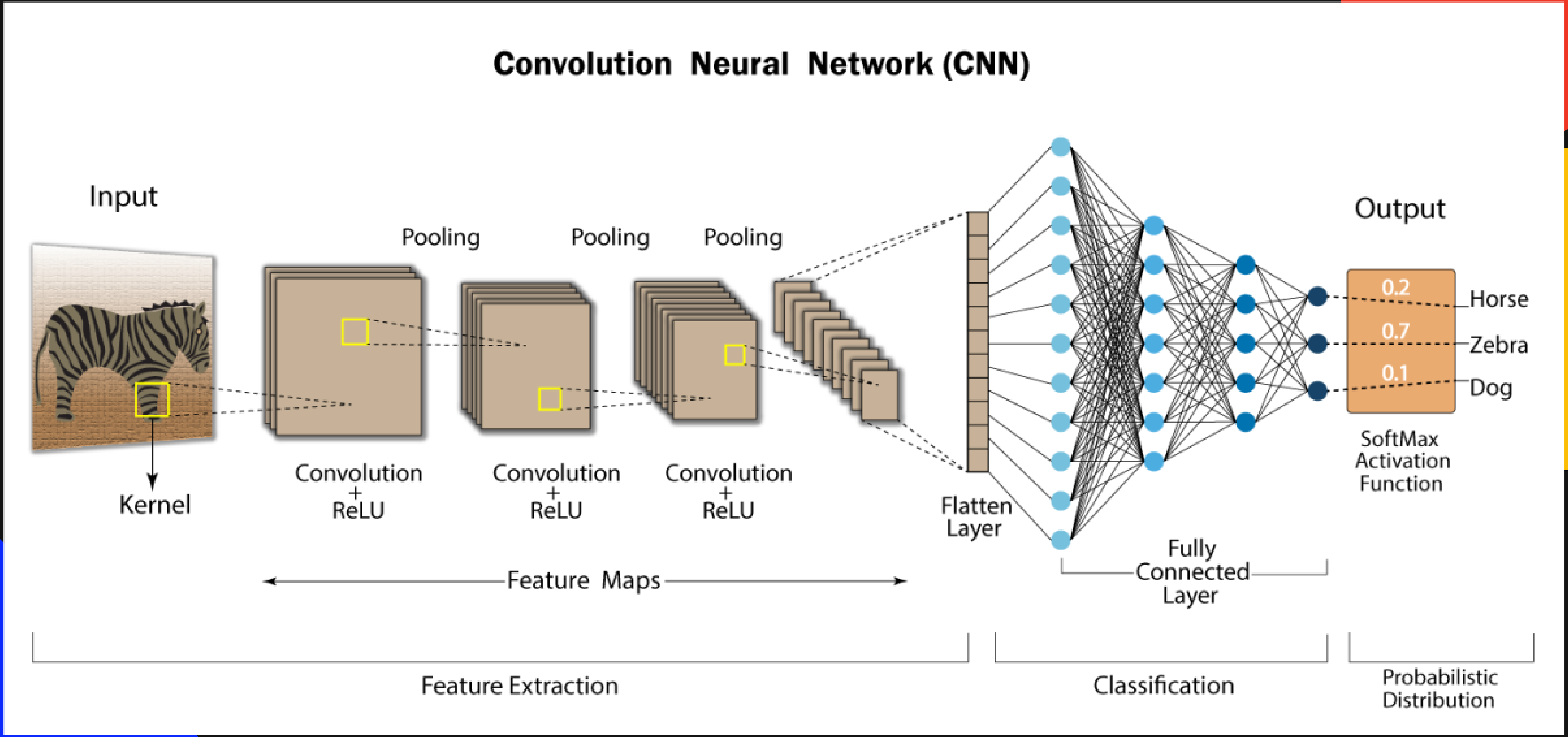


Convolutions just let the network learn features of the image. We then use these in a normal network to get our results

First, we take our image, which is a matrix, and flatten it. This acts as the input layer for our artificial neural network

Next, we add hidden layers and output layers as in a normal neural network.

THE ENTIRE NETWORK

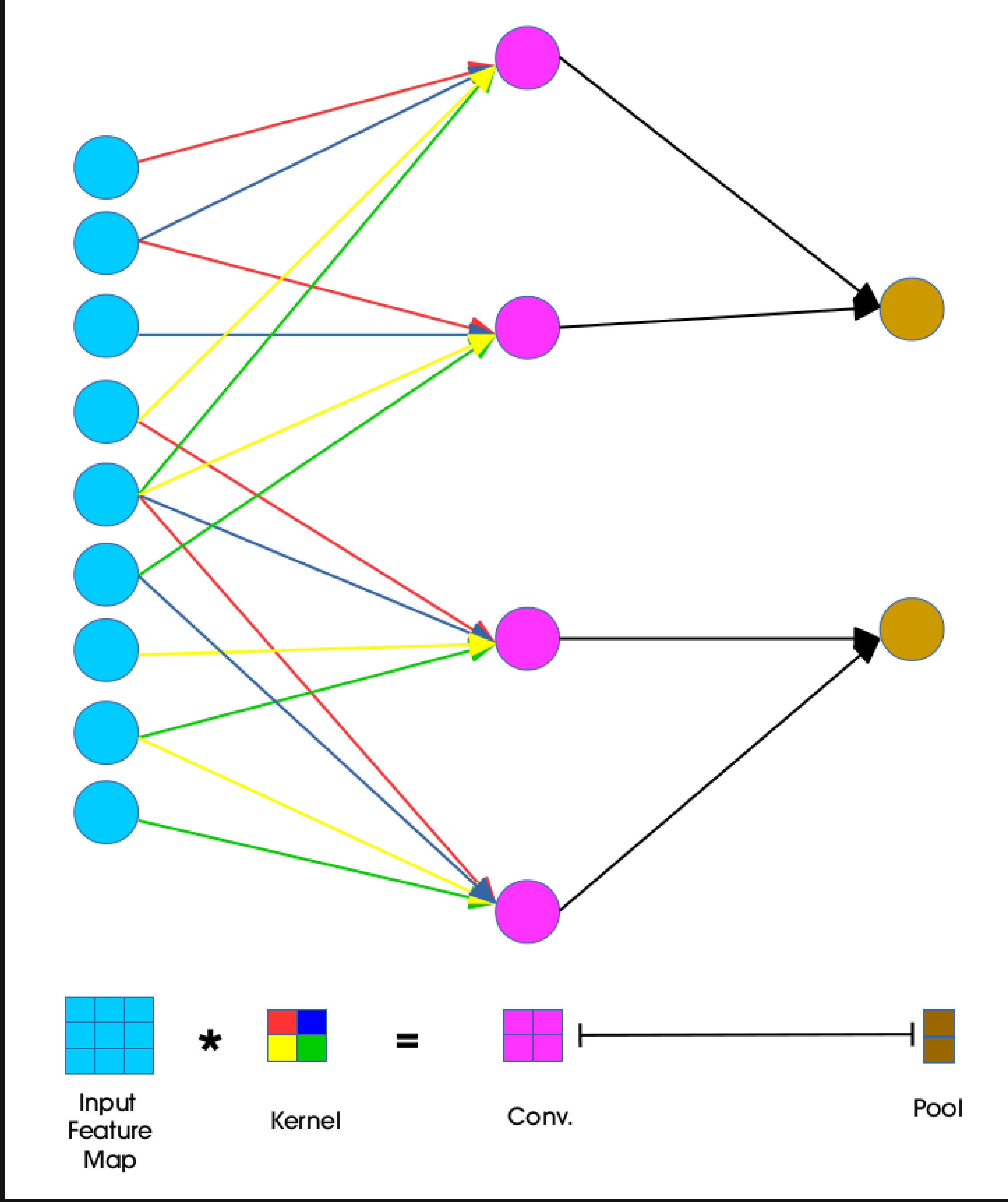


BACKPROPAGATION

Backpropagation is applied on the kernels in CNNs.

CNNs can be considered a partially connected neural network

~~We~~ Pytorch can do the standard backpropagation by only considering connected neurons



$$\begin{bmatrix} 50 & 165 & 67 & 0 \\ 94 & 23 & 88 & 12 \\ 178 & 56 & 90 & 64 \\ 234 & 204 & 78 & 123 \end{bmatrix} * \overset{W_{11}}{\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}} = \begin{bmatrix} 83 & 179 \\ 338 & 80 \end{bmatrix}$$

Our target is to update W11. to do that lets see which pixels it affects.

$$\begin{bmatrix}50&165&67&0\\94&23&88&12\\178&56&90&64\\234&204&78&123\end{bmatrix} * \overset{W_{11}}{\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}} = \begin{bmatrix}83&179\\338&80\end{bmatrix}$$

$$\frac{\partial L}{\partial W_{11}} = (y_{11} - t_{11})x_{11} + (y_{12} - t_{12})x_{12} + (y_{21} - t_{21})x_{21} + (y_{22} - t_{22})x_{22}$$

where x11, x12 so on are from the image like 50, 165
t11,t12 are the targets for each output pixel
y11 ,y12 etc are the present values, that is 83, 179