Artificial Intelligence

Convolutional Neural Networks

Agenda

01 Disadvantages of fully connected network

02 What is CNN?

03 Different Layers in CNN

04 Build a model with CNN using Keras

Let us understand how our computer reads the images that we provide to it!

Image Input

What We See

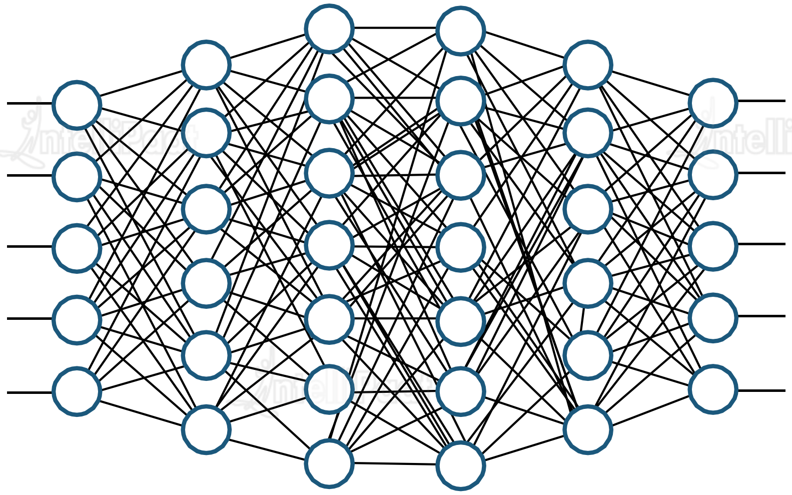
```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
4+ 49 99 40 17 81 18 57 60 87 17 40 98 43 69 18 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 76 36 54 20 35 17 12 50
32 96 41 20 64 23 67 10 26 35 40 67 59 54 70 66 10 30 64 70
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
4+ 49 99 40 17 81 18 57 60 87 17 40 98 43 69 18 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 76 36 54 20 35 17 12 50
32 96 41 20 64 23 67 10 26 35 40 67 59 54 70 66 10 30 64 70
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
4+ 49 99 40 17 81 18 57 60 87 17 40 98 43 69 18 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 76 36 54 20 35 17 12 50
```

What Computer See

- When a computer sees an image (takes an image as input), it will see an array of pixel values
- Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (*3 refers to RGB values*)
- Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point
- These numbers are the only inputs available to the computer

Problem with a Fully Connected Network

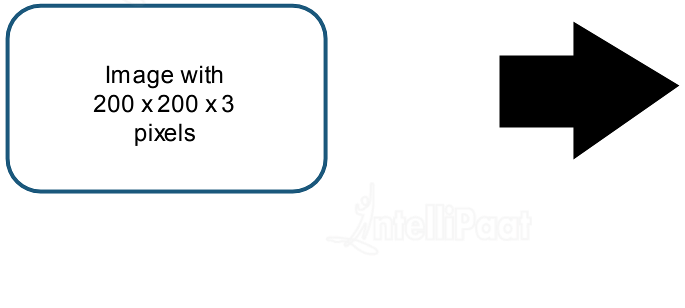
Image with
28 x 28 x 3
pixels



Number of weights in the first Hidden Layer will be 2352

A single fully connected neuron in the first Hidden Layer of a regular neural network would have ***28*28*3 = 2,352*** weights

Problem with a Fully Connected Network

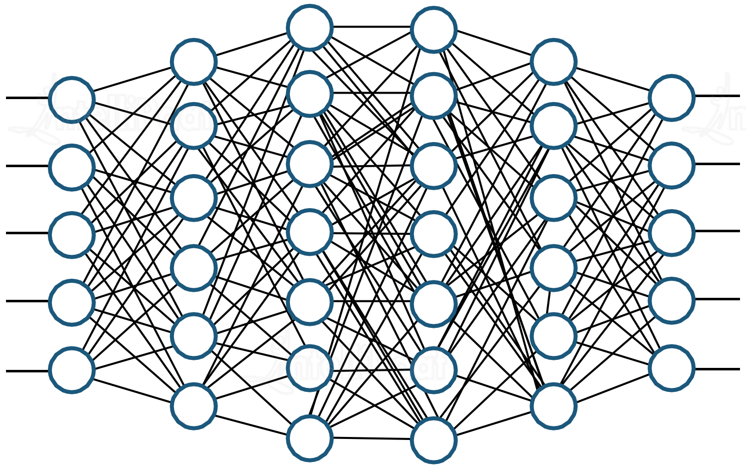


Number of weights in the first Hidden Layer will be 120 000

An image of size ***200*200*3 = 2,352*** would have ***120,000*** weights at the Hidden Layer

Problem with a Fully Connected Network

Image with
200 x 200 x 3
pixels



Number of weights in
the first Hidden Layer
will be 120 000

An image of size ***200*200*3 = 2,352*** would have ***120,000*** weights at the Hidden Layer

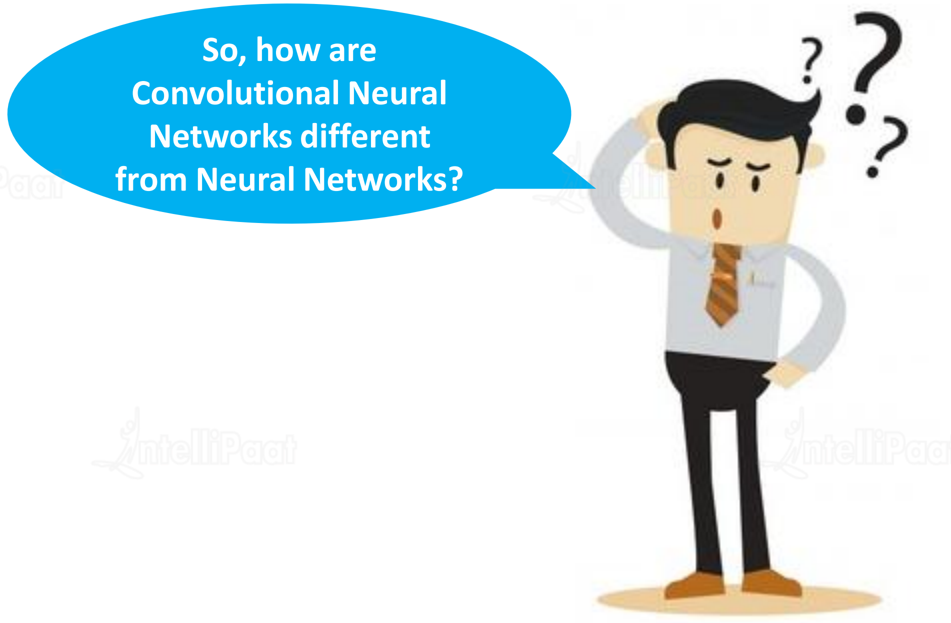
Also, we will have several such layers of neurons leading to several parameters. Thus, this connectivity would be a waste as the huge number of parameters would lead to overfitting!

Let us understand what CNN is and how it helps overcome this problem!

Convolutional Neural Networks (CNNs/ConvNets)

CNNs are like neural networks (that we have already studied before) and are made up of neurons with learnable weights and biases.
Each neuron receives several inputs, takes a weighted sum of them, passes it through an activation function, and responds with an output

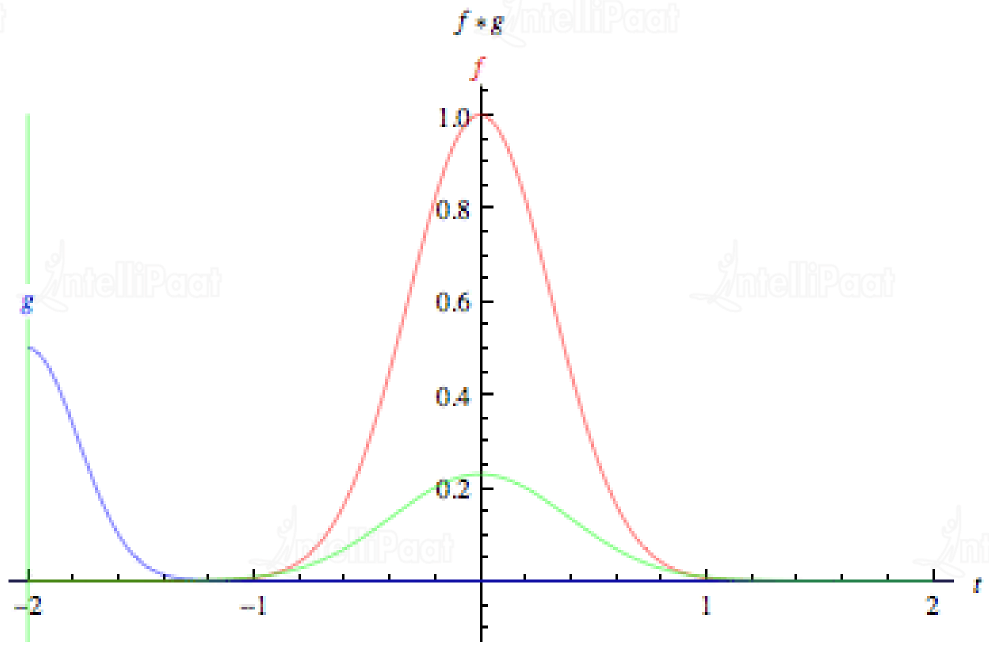
- The whole network has a loss function, and all tips and tricks that we developed for neural networks still apply on CNNs
- A ConvNet perceives *images as volumes*, i.e., 3-dimensional objects, rather than as flat canvases to be measured only by width and height


So, how are Convolutional Neural Networks different from Neural Networks?

Convolution: Definition

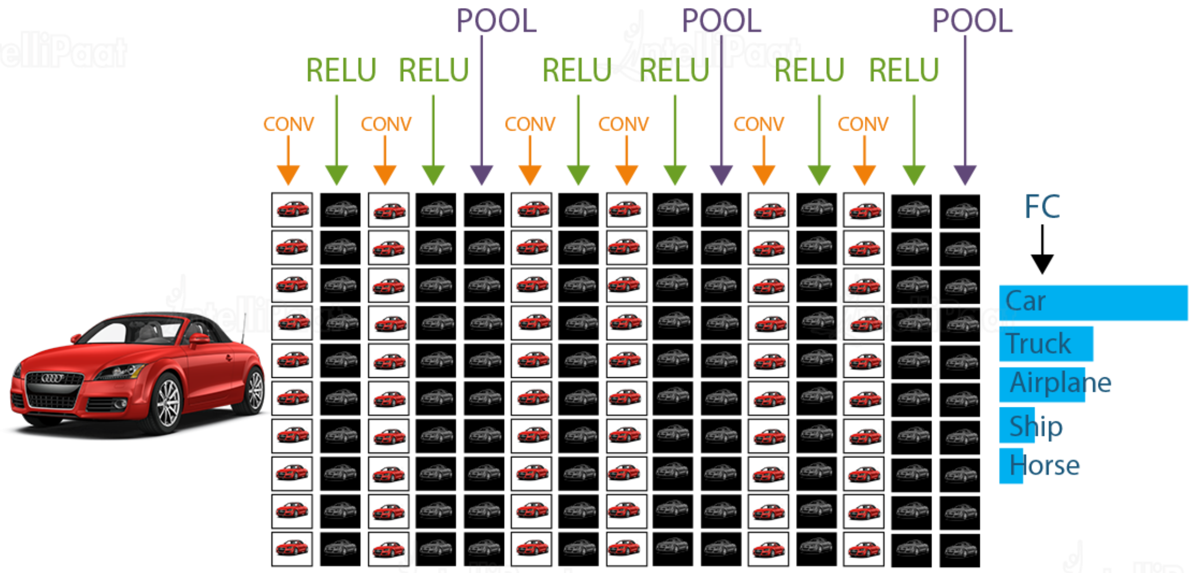
"*to convolve*" means to roll together

- From mathematical perspective, a convolution is the integral measuring of how much two functions overlap as one passes over the other
- Think of convolution as a way of mixing two functions by multiplying them



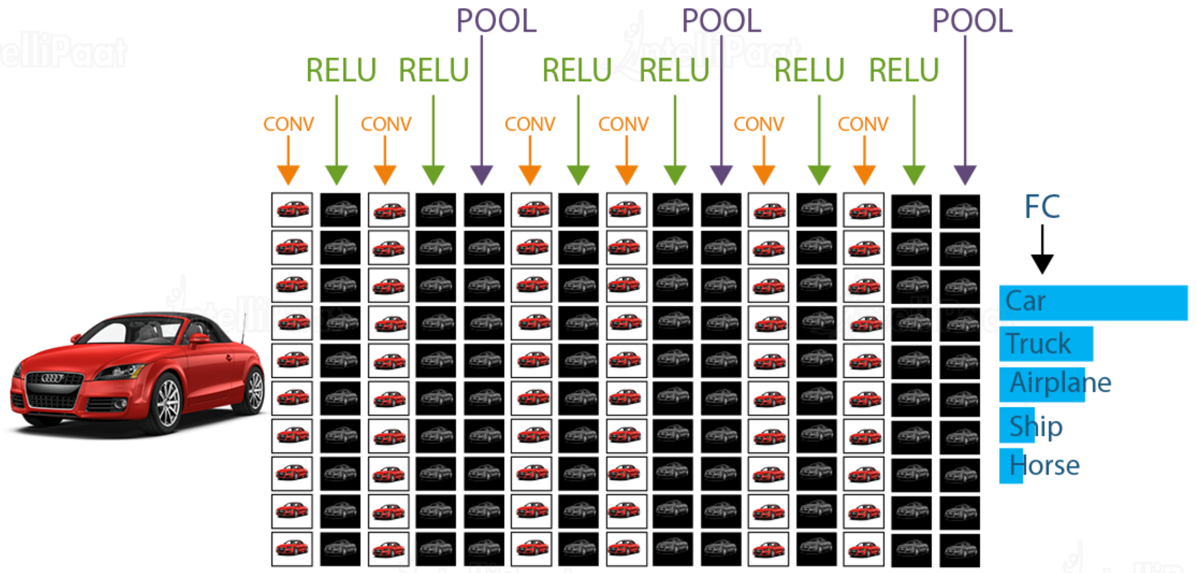
More About CNNs

- ConvNets pass many filters over a single image, each one picking up a different signal
- At a fairly early layer, you could imagine them as passing a horizontal line filter, a vertical line filter, and a diagonal line filter to create a map of the edges in the image
- CNNs take those filters, slice of the image's feature space, and map them one by one; i.e., they create a map of each place wherever feature occurs
- By learning different portions of a feature space, convolutional nets allow for easily scalable and robust feature engineering



More About CNNs

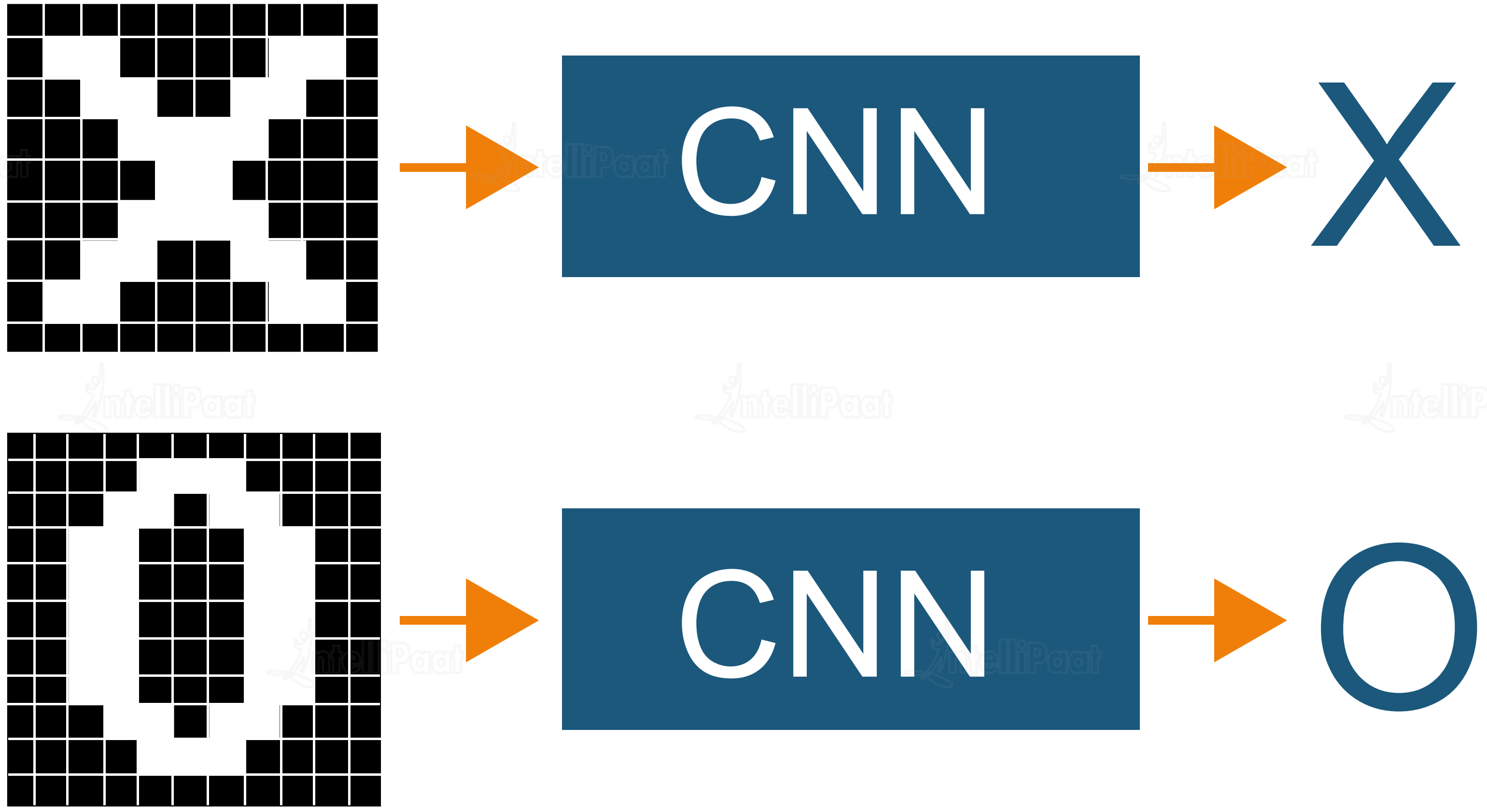
- A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function
- There are four layered concepts we should understand in Convolutional Neural Networks:
  - ***Convolution***
  - ***ReLU***
  - ***Pooling***
  - ***Full Connectedness (Fully Connected Layer)***



Before going deep into this concept,

let us see an example!

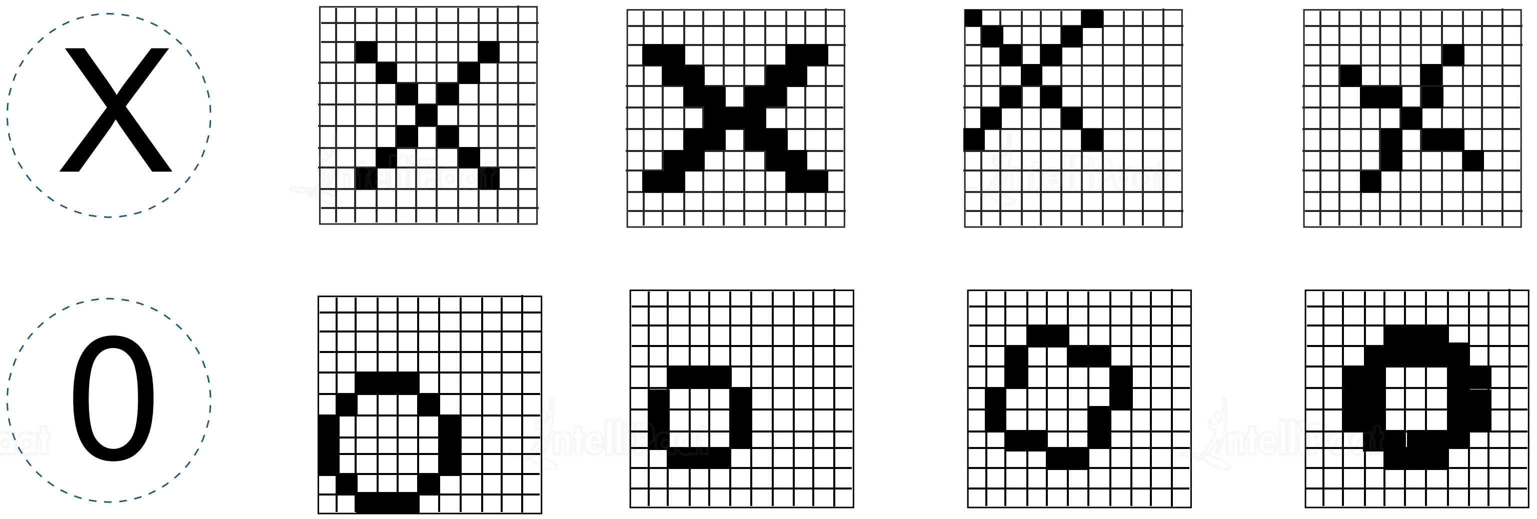
Use Case 1



In this simple example, we will determine whether the image is of an ***O*** or of an ***X***

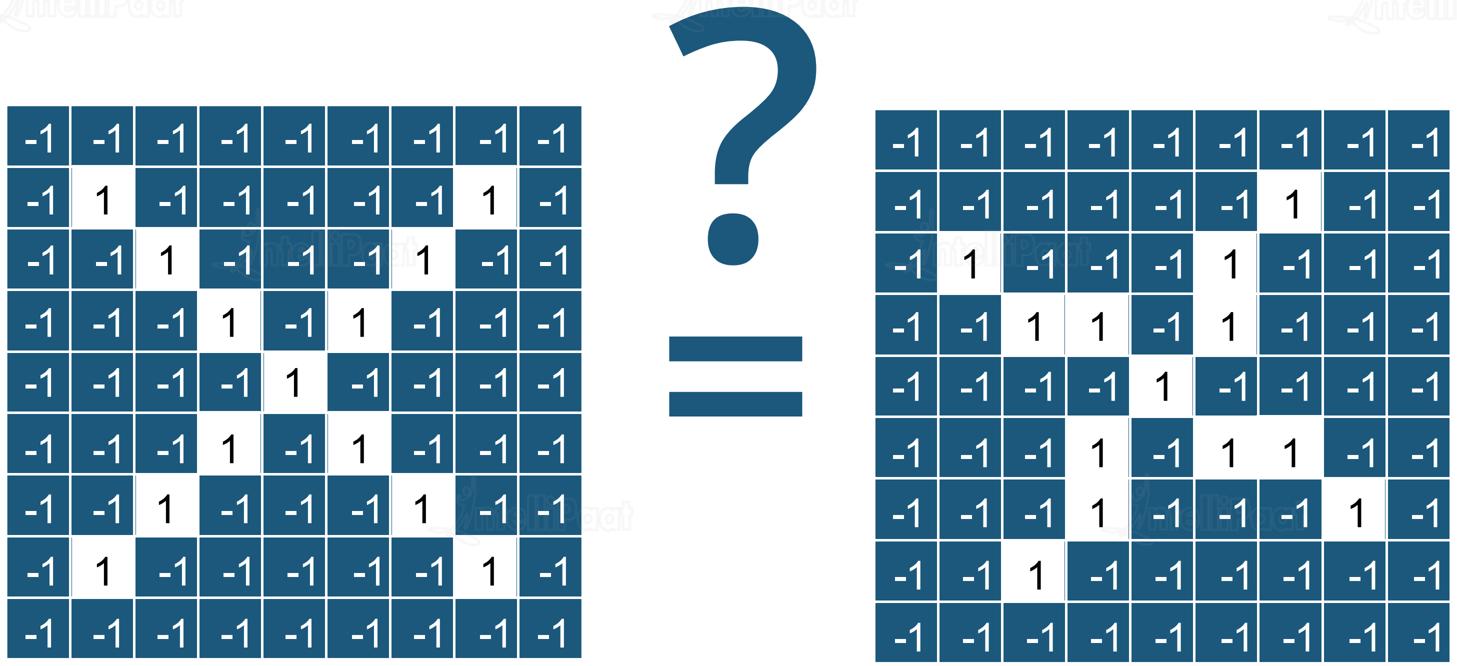
Use Case 1

- Here, there are multiple renditions of *X*s and *O*s
- This makes it tricky for a computer to recognize them
- But the goal is that if the input signal looks like images it has seen before, the 'image' reference signal will be mixed into, or convolved with, the input signal
- The resulting output signal is then passed on to the next layer



Use Case 1

- A naïve approach to solve this problem is to save images of *X* and *O* and compare every new image with our examples to check which is a better match
- To a computer, an image looks like a 2-dimensional array of pixels (think of giant checkerboard) with a number in each position
- When comparing two images, if any pixel value doesn't match, then these images don't match, at least according to the computer
- Ideally, we would like to be able to see *X*s and *O*s even if they're shifted, shrunken, rotated, or deformed. ***This is where CNNs come in***



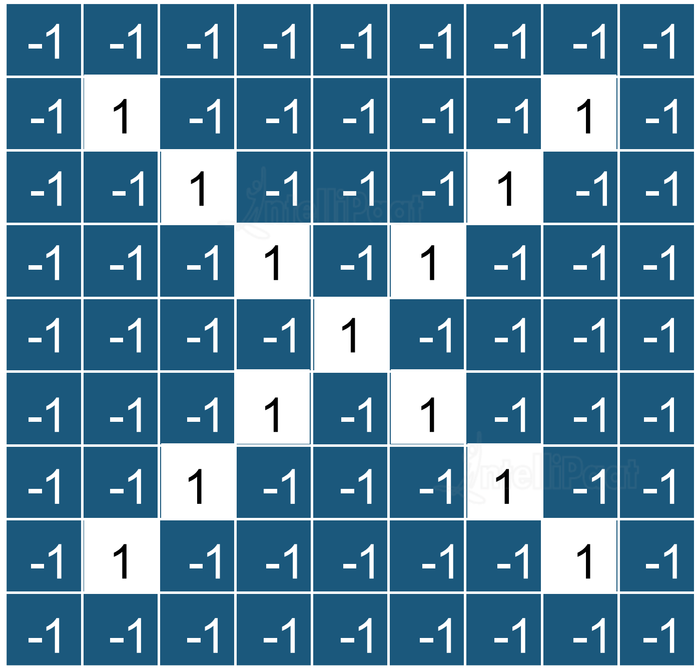
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-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

Probably, I think we need a *classifier* which can be used with images and correctly predict what it is! Agree or not?

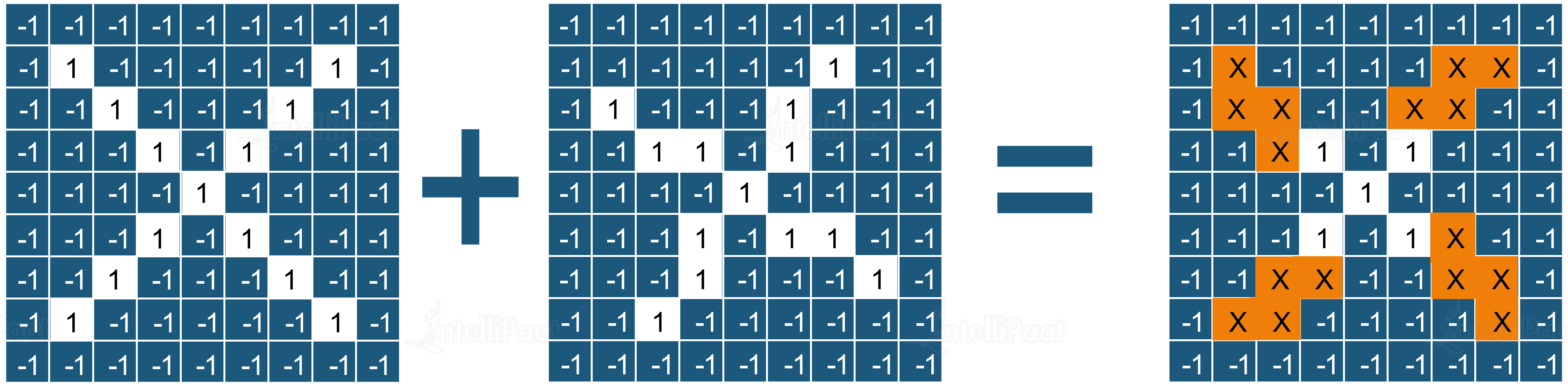
Use Case 1

- A computer understands an image using numbers at each pixel as shown in the figure
- Here, blue pixels have −1 value, while the white pixels have a value of 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	-1

Use Case 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	-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	-1

=

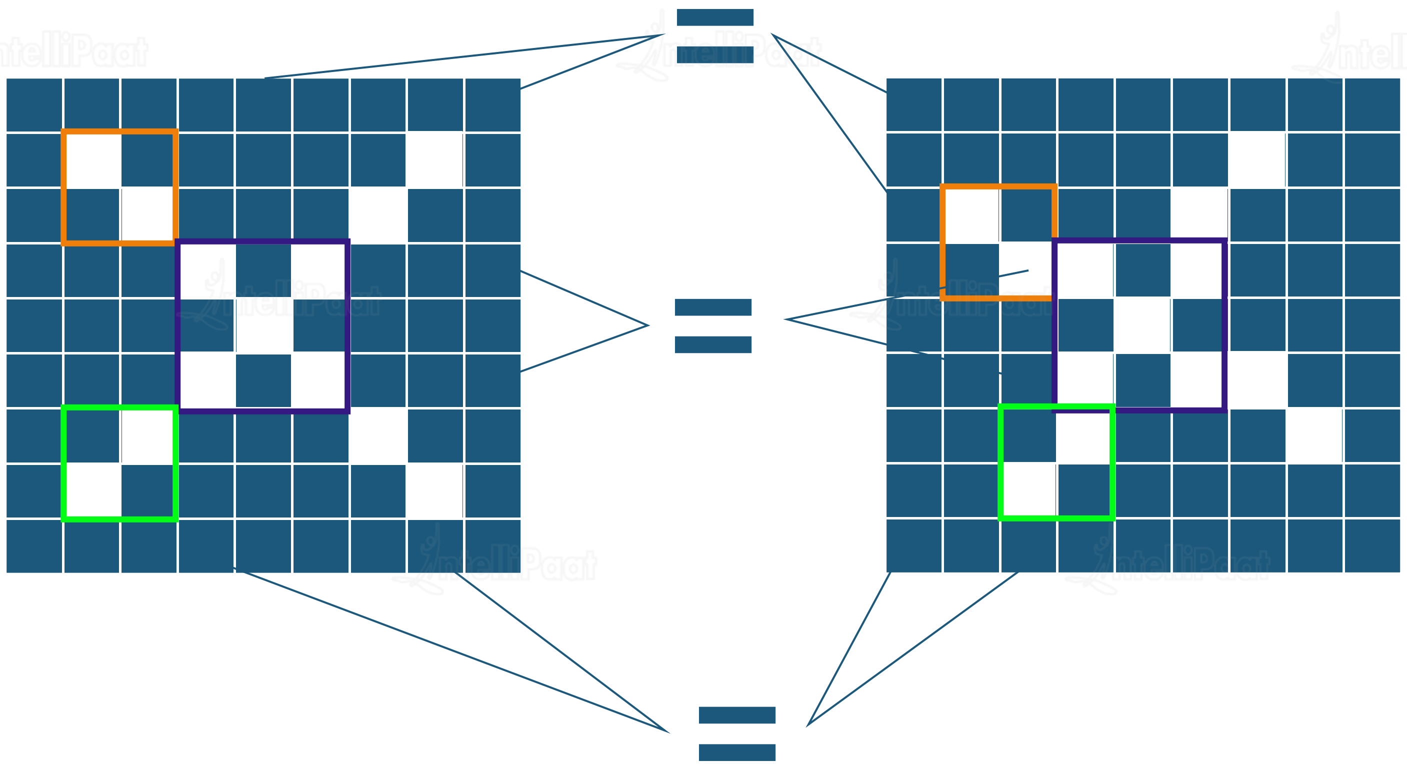
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

If we just normally search and compare the values between a normal image and another '*X*' rendition, we would get a lot of missing pixels, which means, this is not an optimal way of image classification since it requires exactly the same images to classify

Let’s see how CNN solves this problem

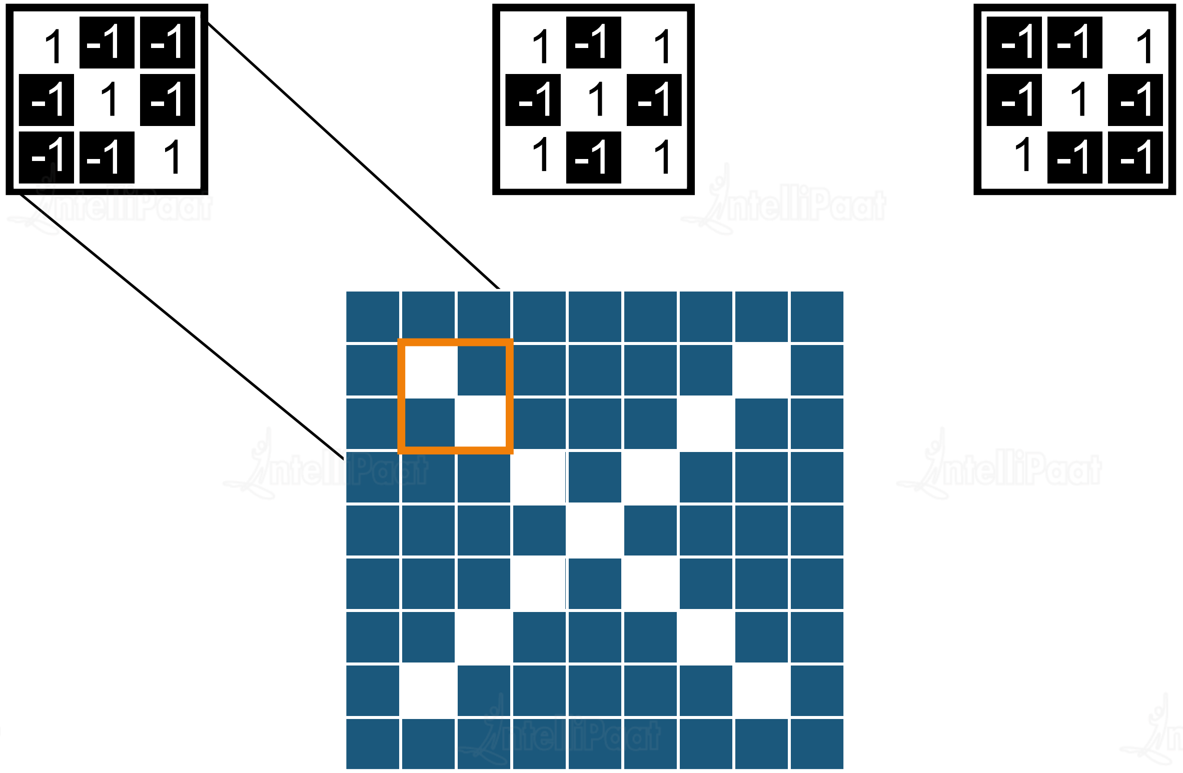
Feature Matching

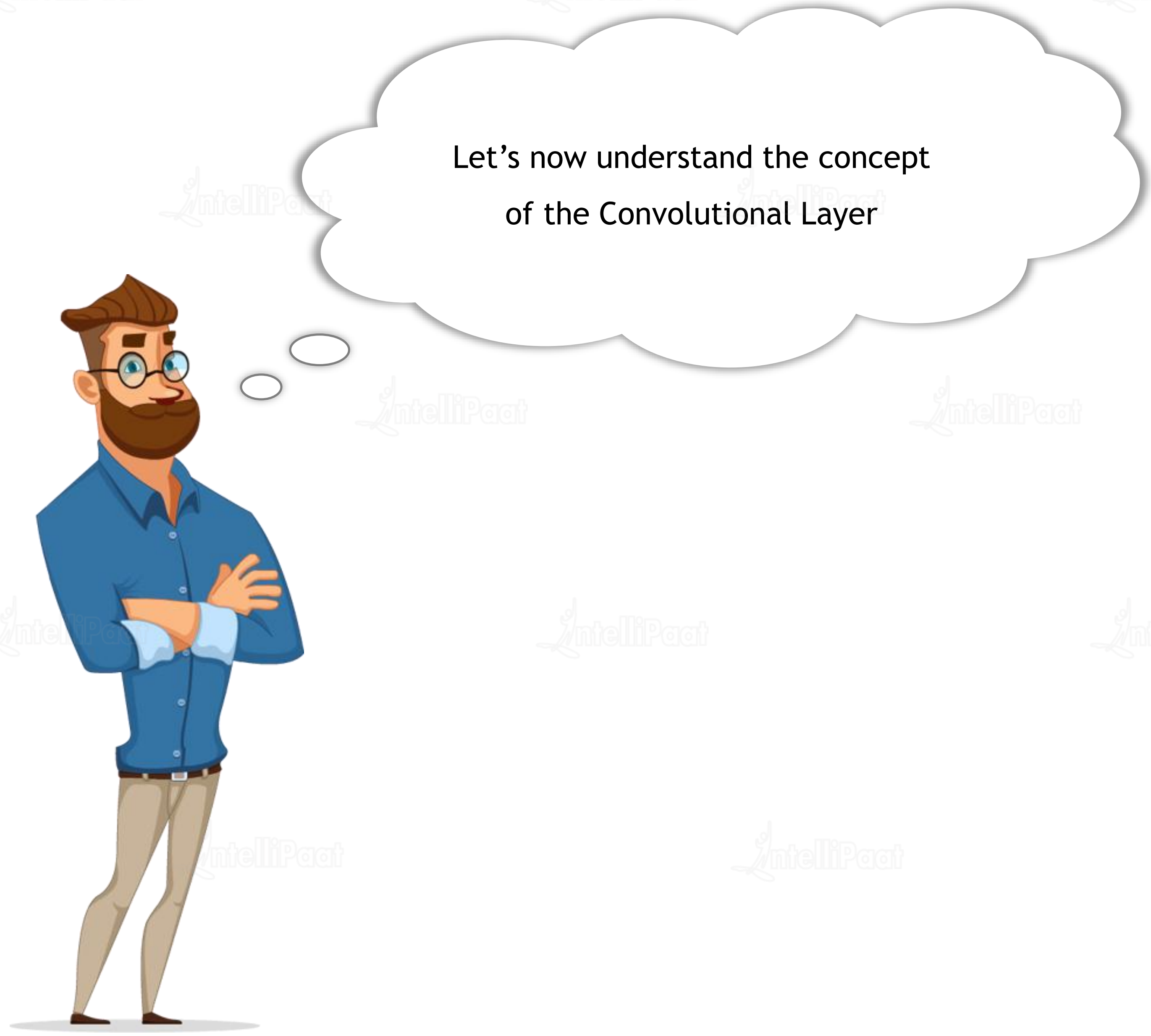
- CNNs compare images piece by piece
- The pieces that it looks for are called features
- By finding rough feature matches in roughly the same positions in the two images, CNNs get a lot better at seeing similarity, than the whole-image matching schemes



How Do CNNs Work?

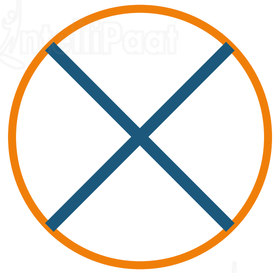
- Each feature is like a mini-image—a small 2-dimensional array of values
- Features match common aspects of the images
- In the case of *X* images, features consisting of diagonal lines and a crossing capture all the important characteristics of most Xs
- These features will probably match the arms and the center of any image of an *X*



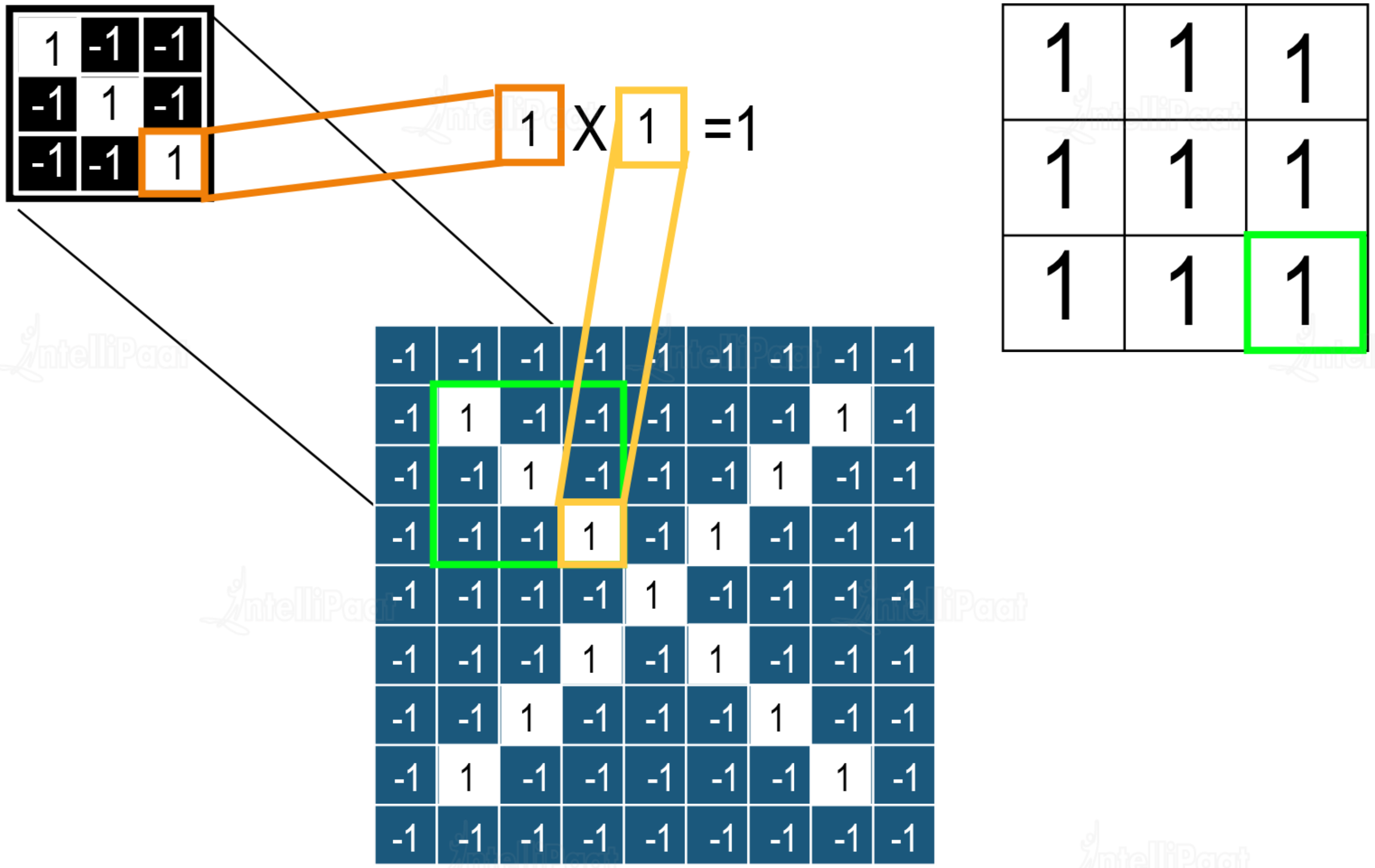

Let’s now understand the concept of the Convolutional Layer

Convolutional Layer

- It is the first layer of a CNN
- When presented with a new image, the CNN doesn’t know exactly where these features will match, so it tries them everywhere, in every possible position
- In calculating the match of a feature across the whole image, they (ConvNet) act as filters
- The math used to perform this is called convolution, from which Convolutional Neural Networks get their name
- We have four steps in convolution:
  - Line up the feature and the image
  - Multiply each image pixel by the corresponding feature pixel
  - Add the values and find the sum
  - Divide the sum by the total number of pixels in the feature


Symbol for Convolution

Convolutional Layer



- In our example, consider a feature image and one pixel from it
- Multiply this with the existing image, and the product will be stored in another buffer feature image

Convolutional Layer

- Then, add up the answers and divide by the total number of pixels in the feature
- If both pixels are white (with a value of 1) then 1 * 1 = 1
- If both are black (−1), then (−1) * (−1) = 1
- Either way, every matching pixel results in 1
- Similarly, any mismatch results in −1
- If all pixels in a feature match, then adding them up and dividing by the total number of pixels gives a value, 1
- Similarly, if none of the pixels in a feature match the image patch, then the answer is −1

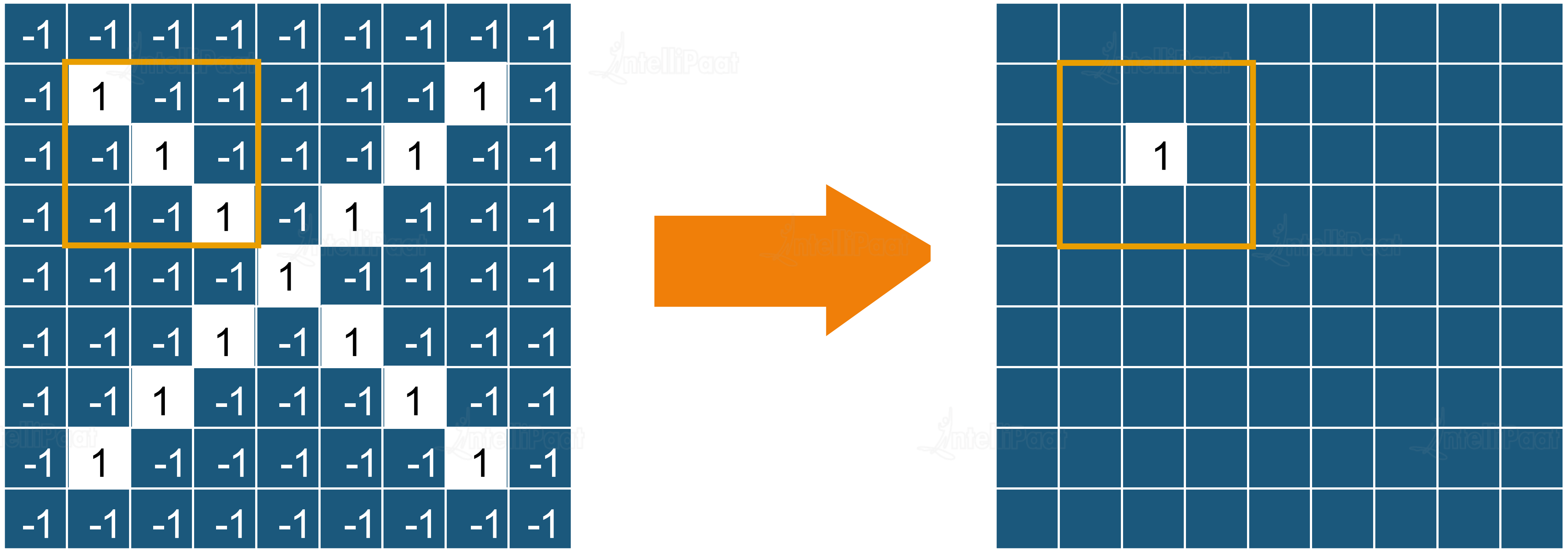
1	-1	-1
-1	1	-1
-1	-1	1

$$\frac{1+1+1+1+1+1+1+1+1}{9} = 1$$

-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	-1

1	1	1
1	1	1
1	1	1

Convolutional Layer



- The final value obtained from the math that is performed in the last step is placed at the center of the filtered image as shown above

Convolutional Layer

Now, move this filter around and do the same at any pixel in the image. Consider the below example:

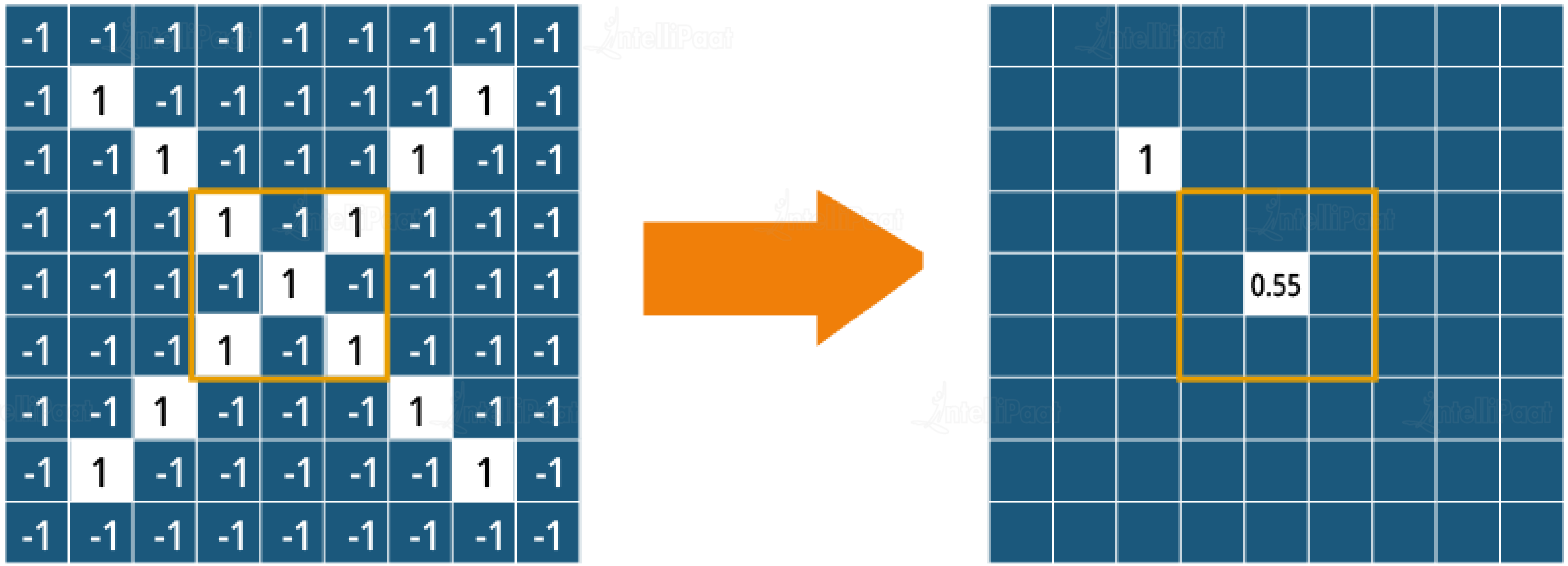
1	-1	-1
-1	1	-1
-1	-1	1

$$\frac{1+1-1+1+1+1-1+1+1}{9} = 0.55$$

-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

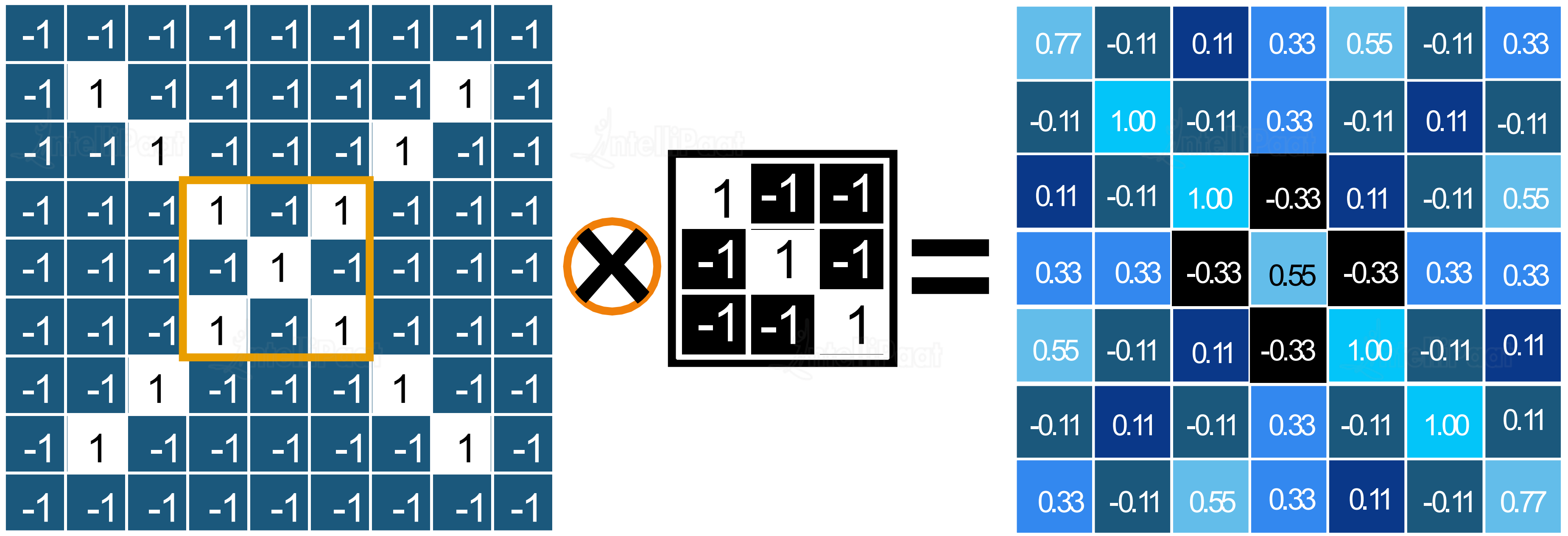
1	1	-1
1	1	1
-1	1	1

Convolutional Layer



- As you can see, here after performing all the steps, the value is 0.55!
- Take this value and place it in the image as explained before

Convolutional Layer



-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

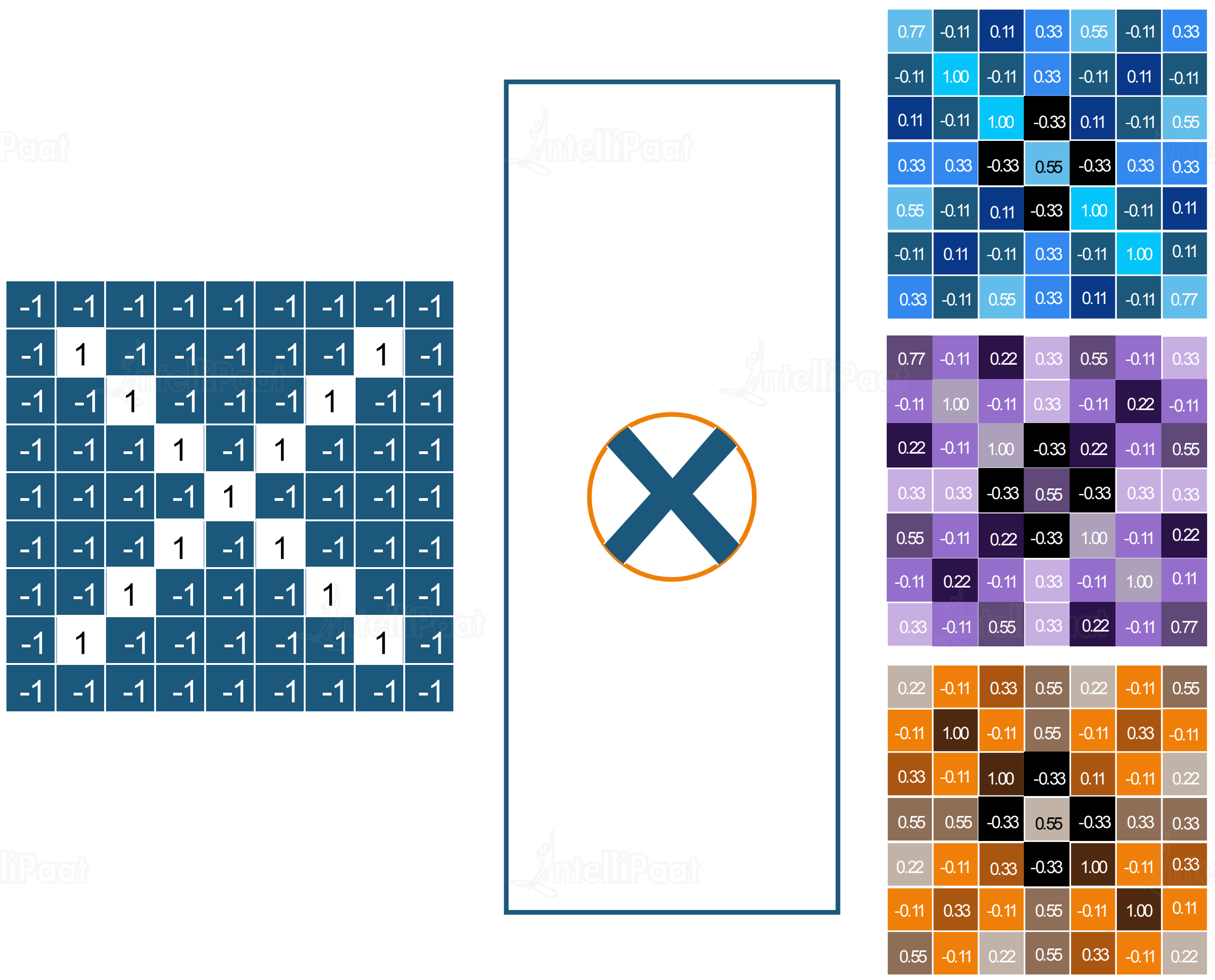
1	-1	-1
-1	1	-1
-1	-1	1

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

- To complete the convolution, repeat this process, lining up the feature with every possible image patch
- Take the answer from each convolution and make a new 2-dimensional array from it, based on where in the image each patch is located
- This map of matches is also a filtered version of the original image
- It’s a map showing where in the image the feature can be found
- Values close to 1 show strong matches, values close to −1 show strong matches for the photographic negative of our feature, and values near 0 show no match of any sort

Convolutional Layer

- The next step is to repeat the convolution process in its entirety for each of the other features. The result is a set of filtered images, one for each of the filters
- It's convenient to think of this whole collection of convolution operations as a single processing step



-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

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77	-0.11	0.22	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.22	-0.11
0.22	-0.11	1.00	-0.33	0.22	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.22	-0.33	1.00	-0.11	0.22
-0.11	0.22	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.22	-0.11	0.77

0.22	-0.11	0.33	0.55	0.22	-0.11	0.55
-0.11	1.00	-0.11	0.55	-0.11	0.33	-0.11
0.33	-0.11	1.00	-0.33	0.11	-0.11	0.22
0.55	0.55	-0.33	0.55	-0.33	0.33	0.33
0.22	-0.11	0.33	-0.33	1.00	-0.11	0.33
-0.11	0.33	-0.11	0.55	-0.11	1.00	0.11
0.55	-0.11	0.22	0.55	0.33	-0.11	0.22

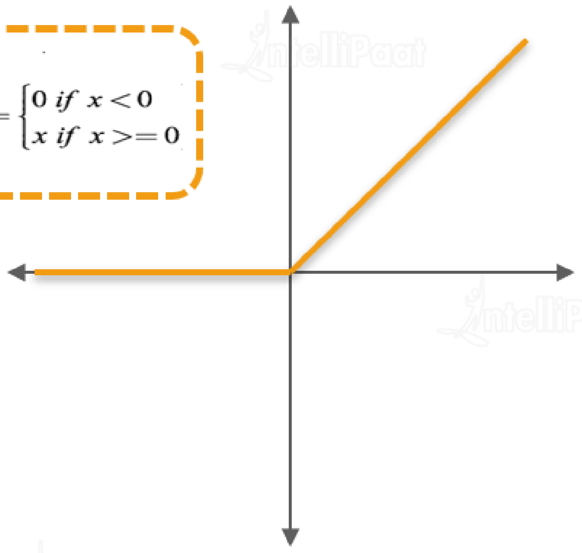
Let’s understand the concept of the ReLU Layer

Rectified Linear Units

- ReLU is an activation function (as discussed earlier)
- This function activates a node only if the input is above a certain quantity.
- When the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable

x	f(x) = x	F(x)
−2	f(−2) = 0	0
−6	f(−6) = 0	0
2	f(2) = 2	2
6	f(6) = 6	6

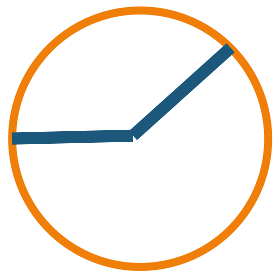
$$f(x) = \begin{cases} 0 \ if \ x < 0 \\ x \ if \ x >= 0 \end{cases}$$



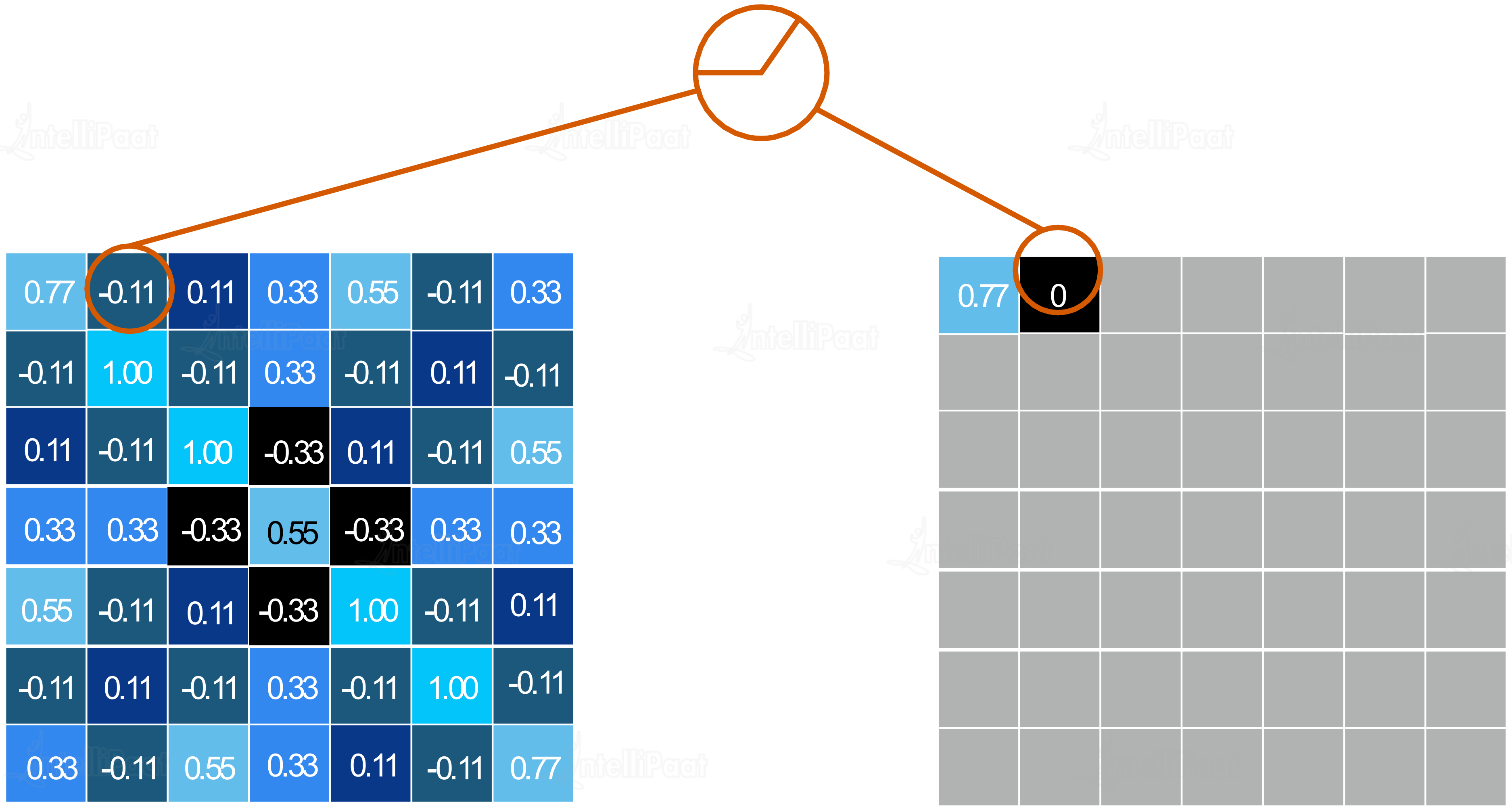
- We have considered a simple function with values as mentioned above. So, the function only performs an operation if that value is obtained by the dependent variable

ReLU Layer

- A small but important player in CNNs is the ReLU layer
- Its math is also very simple—wherever a negative number occurs, swap it out for a 0
- This helps the CNN stay mathematically healthy by keeping learned values from getting stuck near 0 or blowing them up toward infinity


Symbol for ReLU

ReLU Layer

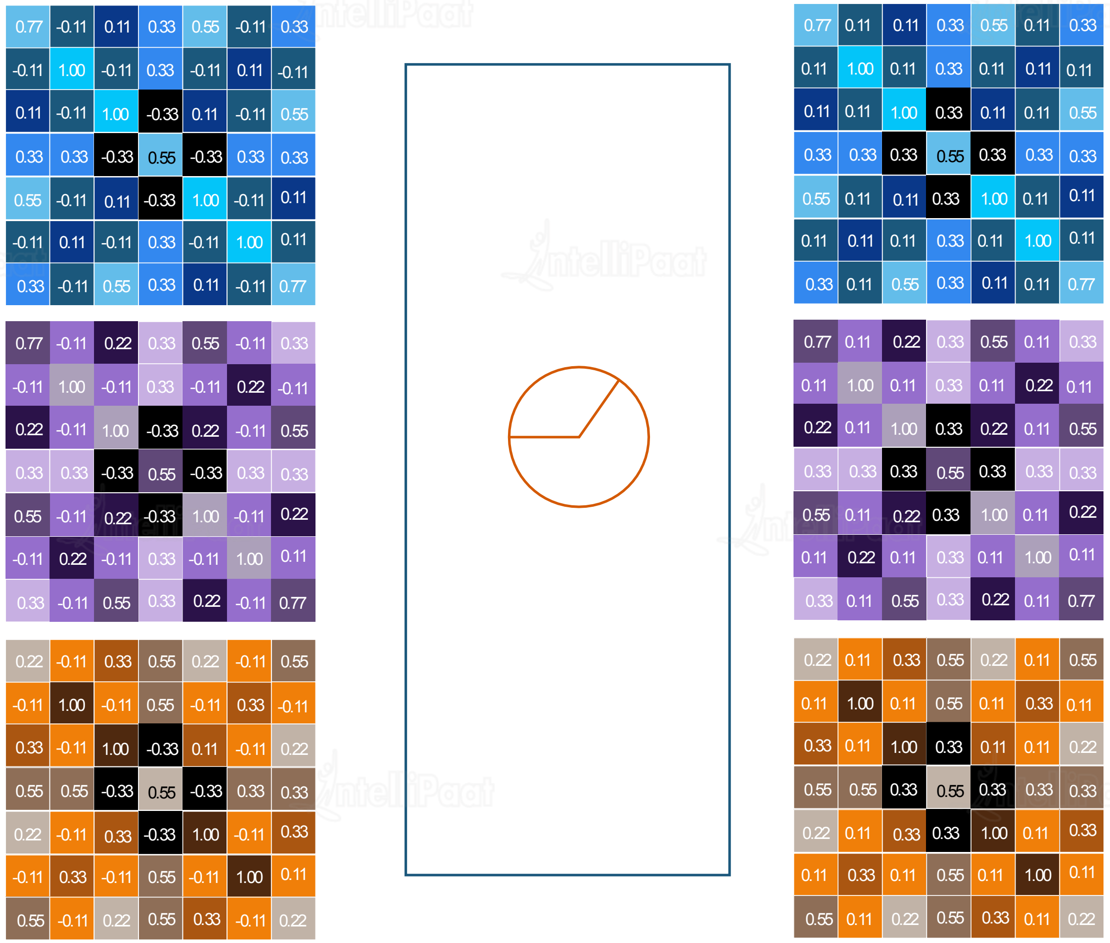


0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77	0					

ReLU Layer

Similarly, we do the same process to all other feature images. The output of a ReLU layer is of the same size as whatever is put into it, just with all negative values removed



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77	0.11	0.11	0.33	0.55	0.11	0.33
0.11	1.00	0.11	0.33	0.11	0.11	0.11
0.11	0.11	1.00	0.33	0.11	0.11	0.55
0.33	0.33	0.33	0.55	0.33	0.33	0.33
0.55	0.11	0.11	0.33	1.00	0.11	0.11
0.11	0.11	0.11	0.33	0.11	1.00	0.11
0.33	0.11	0.55	0.33	0.11	0.11	0.77

0.77	-0.11	0.22	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.22	-0.11
0.22	-0.11	1.00	-0.33	0.22	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.22	-0.33	1.00	-0.11	0.22
-0.11	0.22	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.22	-0.11	0.77

0.77	0.11	0.22	0.33	0.55	0.11	0.33
0.11	1.00	0.11	0.33	0.11	0.22	0.11
0.22	0.11	1.00	0.33	0.22	0.11	0.55
0.33	0.33	0.33	0.55	0.33	0.33	0.33
0.55	0.11	0.22	0.33	1.00	0.11	0.22
0.11	0.22	0.11	0.33	0.11	1.00	0.11
0.33	0.11	0.55	0.33	0.22	0.11	0.77

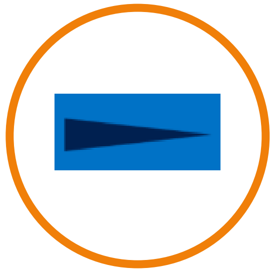
0.22	-0.11	0.33	0.55	0.22	-0.11	0.55
-0.11	1.00	-0.11	0.55	-0.11	0.33	-0.11
0.33	-0.11	1.00	-0.33	0.11	-0.11	0.22
0.55	0.55	-0.33	0.55	-0.33	0.33	0.33
0.22	-0.11	0.33	-0.33	1.00	-0.11	0.33
-0.11	0.33	-0.11	0.55	-0.11	1.00	0.11
0.55	-0.11	0.22	0.55	0.33	-0.11	0.22

0.22	0.11	0.33	0.55	0.22	0.11	0.55
0.11	1.00	0.11	0.55	0.11	0.33	0.11
0.33	0.11	1.00	0.33	0.11	0.11	0.22
0.55	0.55	0.33	0.55	0.33	0.33	0.33
0.22	0.11	0.33	0.33	1.00	0.11	0.33
0.11	0.33	0.11	0.55	0.11	1.00	0.11
0.55	0.11	0.22	0.55	0.33	0.11	0.22

Let’s understand the concept of the Pooling Layer

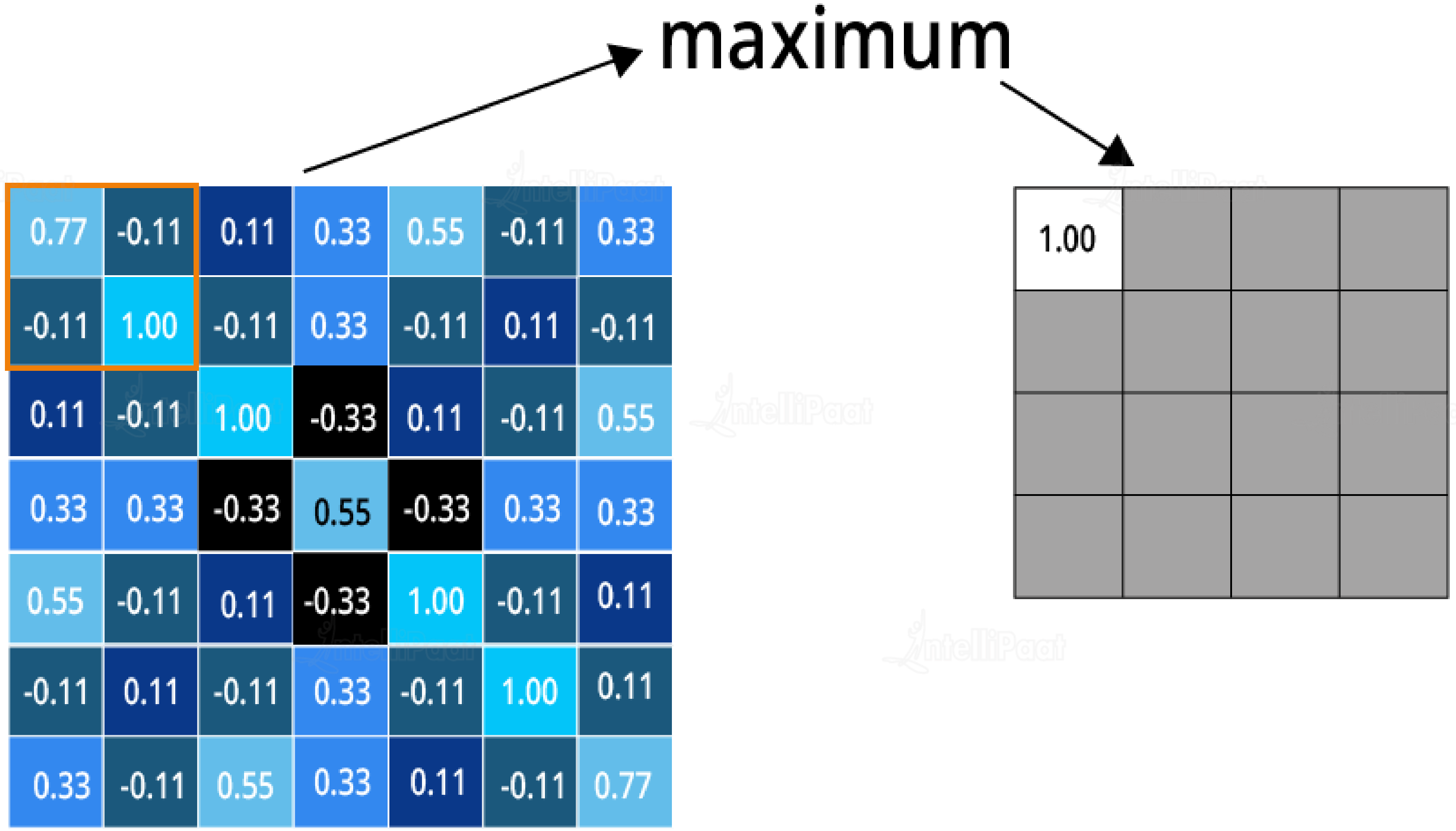
Pooling Layer

- Another powerful tool that CNNs use is called pooling
- Pooling is a way to take large images and shrink them down while preserving the most important information in them
- It consists of stepping a small window across an image and taking the maximum value from the window at each step
- In practice, a window (2x2 or 3x3) and a step of 2 works well



Symbol for Pooling

Pooling Layer



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

- In this case, we took the window size to be 2 and we got four values to choose from
- In those four values, the maximum value is 1, so we pick 1. Also, note that we started with a 7×7 matrix, but now the same matrix after pooling came down to be a 4×4 matrix

Pooling Layer

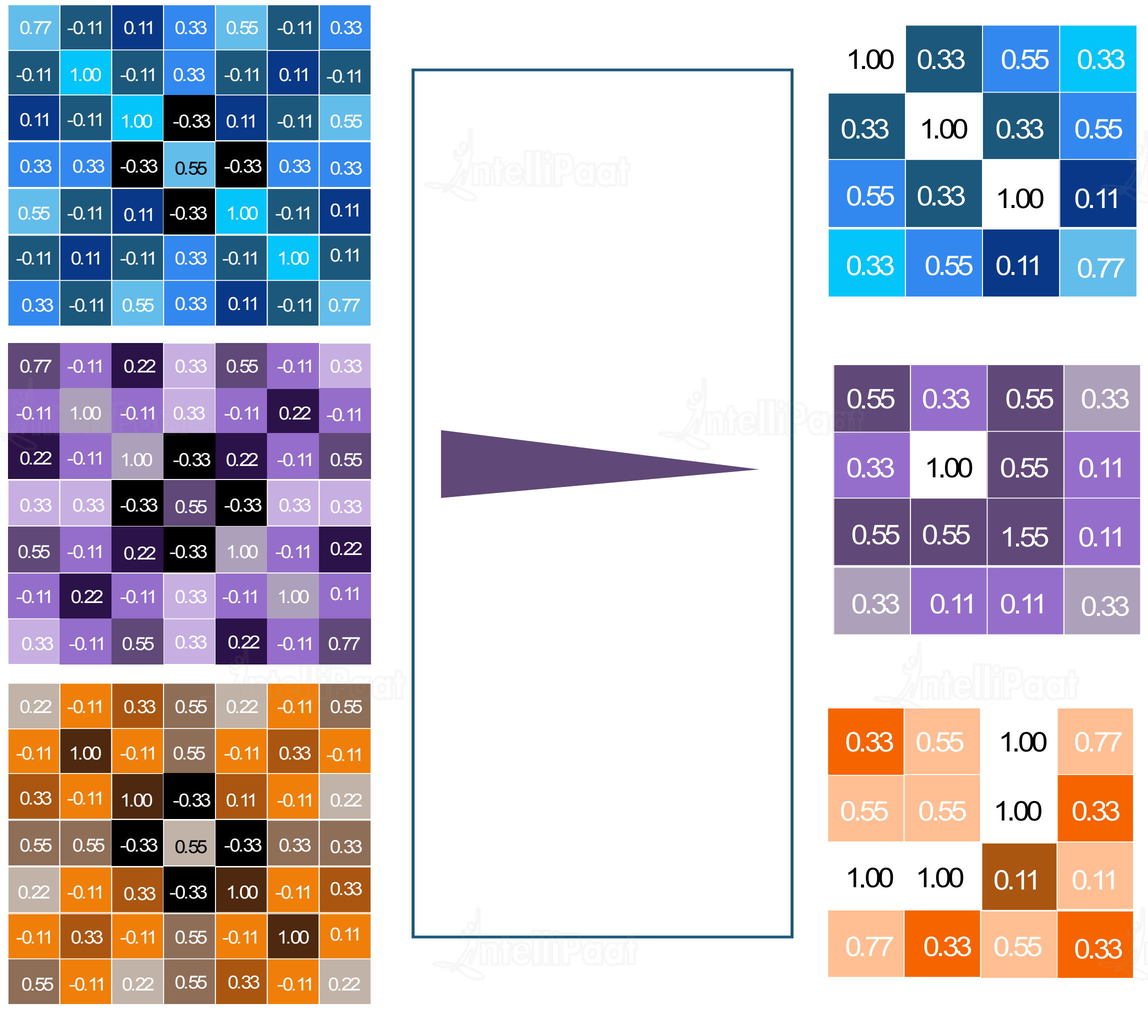
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

- We need to move the window across the entire image
- The procedure is exactly as same as the above, and we need to repeat it for the entire image

Pooling Layer

- We need to do it for two other filters as well. Once this is done, we get the adjacent result
- The output will have the same number of images, but they will have fewer pixels
- This is helpful in managing the computational load
- Making an 8 megapixel image down to a 2 megapixel image makes life a lot easier for everything downstream



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

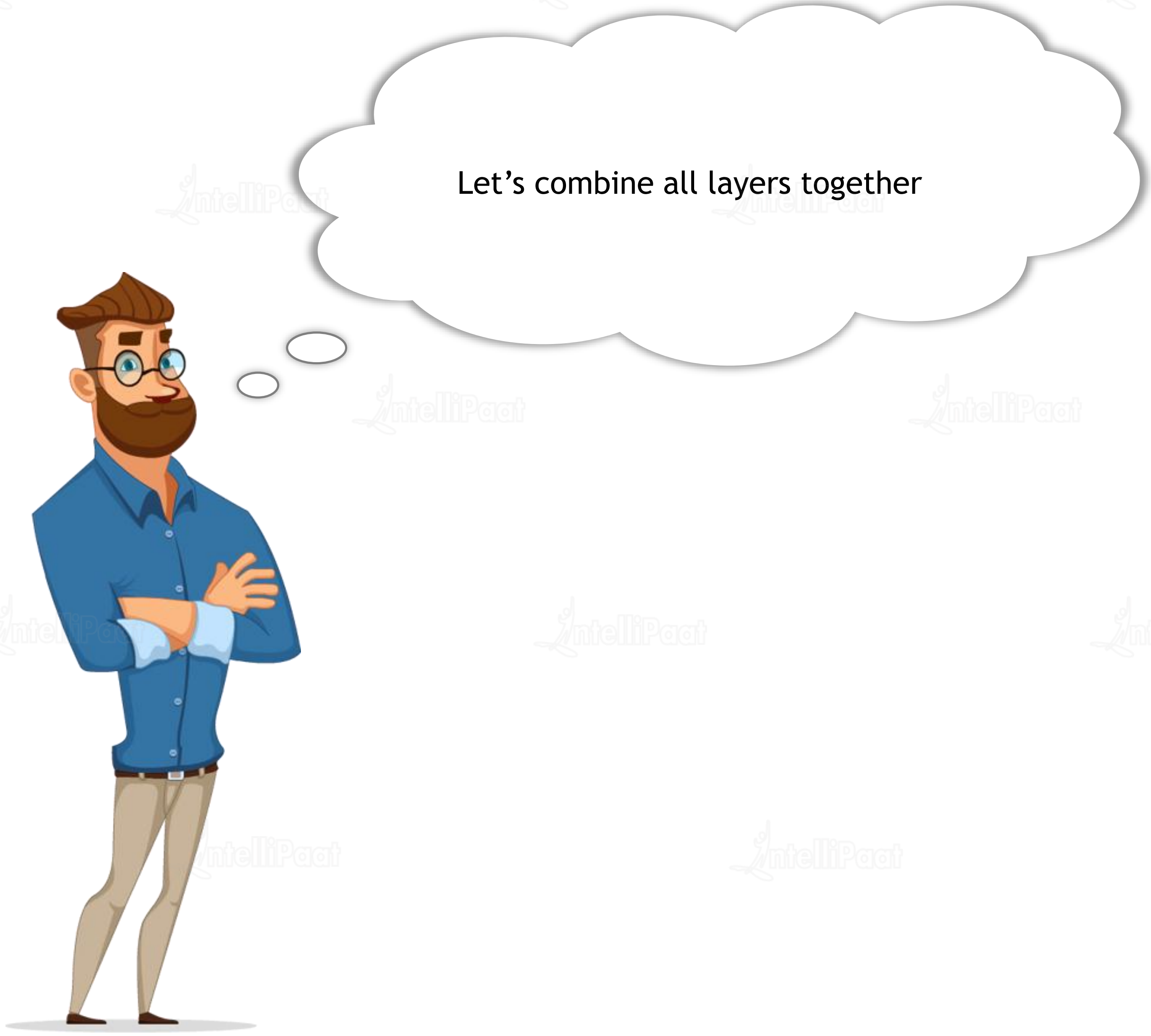
0.77	-0.11	0.22	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.22	-0.11
0.22	-0.11	1.00	-0.33	0.22	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.22	-0.33	1.00	-0.11	0.22
-0.11	0.22	-0.11	0.33	-0.11	1.00	0.11
0.33	-0.11	0.55	0.33	0.22	-0.11	0.77

0.22	-0.11	0.33	0.55	0.22	-0.11	0.55
-0.11	1.00	-0.11	0.55	-0.11	0.33	-0.11
0.33	-0.11	1.00	-0.33	0.11	-0.11	0.22
0.55	0.55	-0.33	0.55	-0.33	0.33	0.33
0.22	-0.11	0.33	-0.33	1.00	-0.11	0.33
-0.11	0.33	-0.11	0.55	-0.11	1.00	0.11
0.55	-0.11	0.22	0.55	0.33	-0.11	0.22

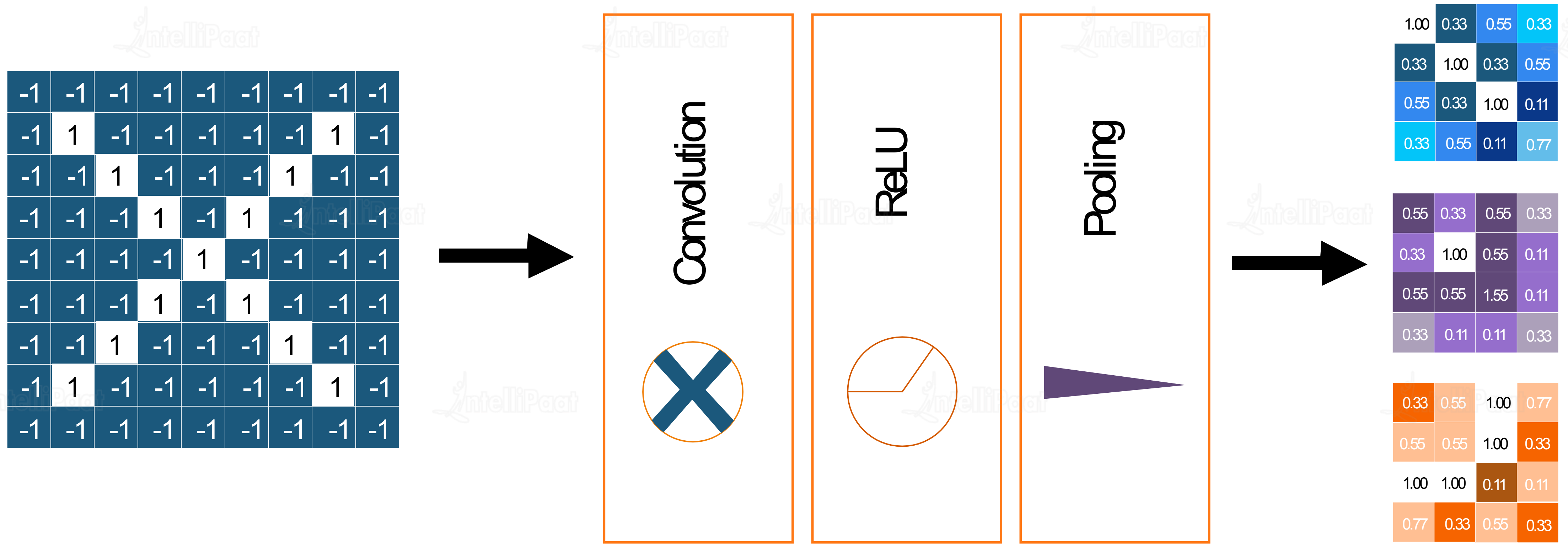
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	1.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.11
0.77	0.33	0.55	0.33


Let’s combine all layers together

All Layers Together!



-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

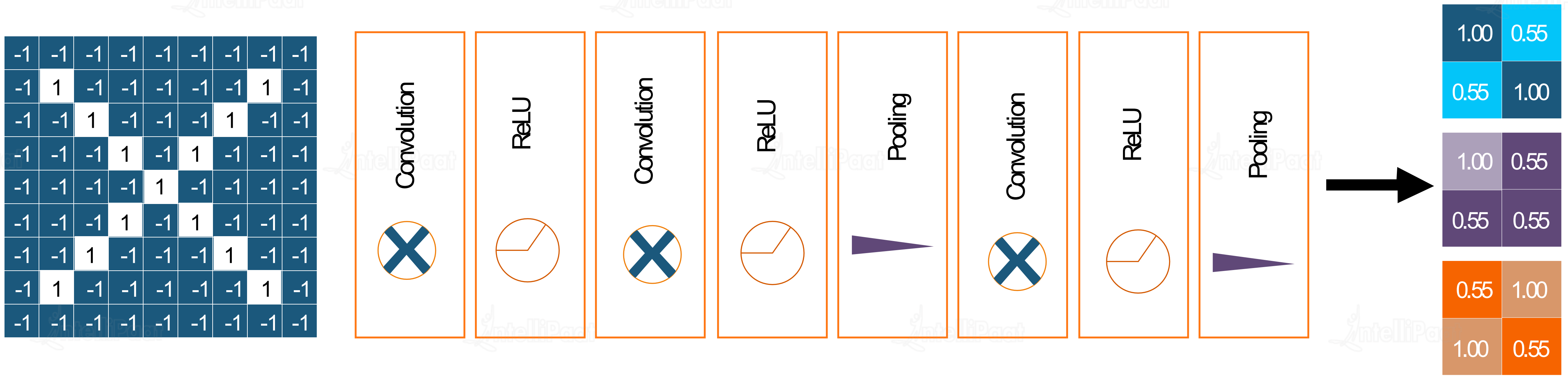
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	1.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.11
0.77	0.33	0.55	0.33

- You’ve probably noticed that the output of one layer is taken as the input to the other. Because of this, we can stack them like Lego bricks

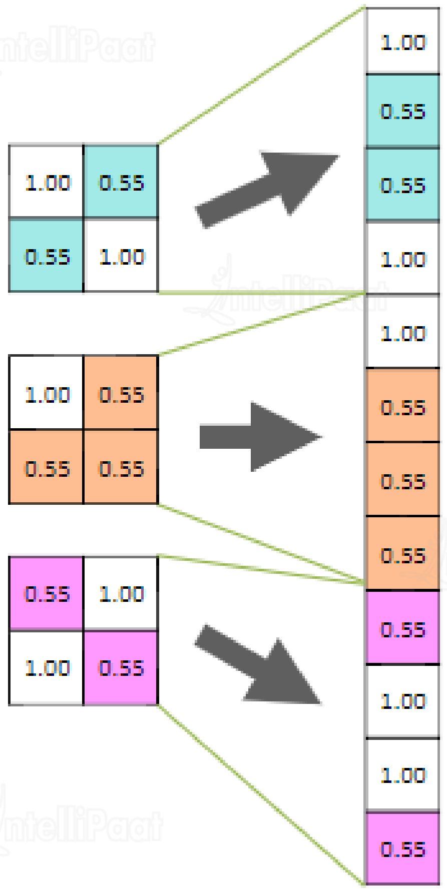
All Layers Together!



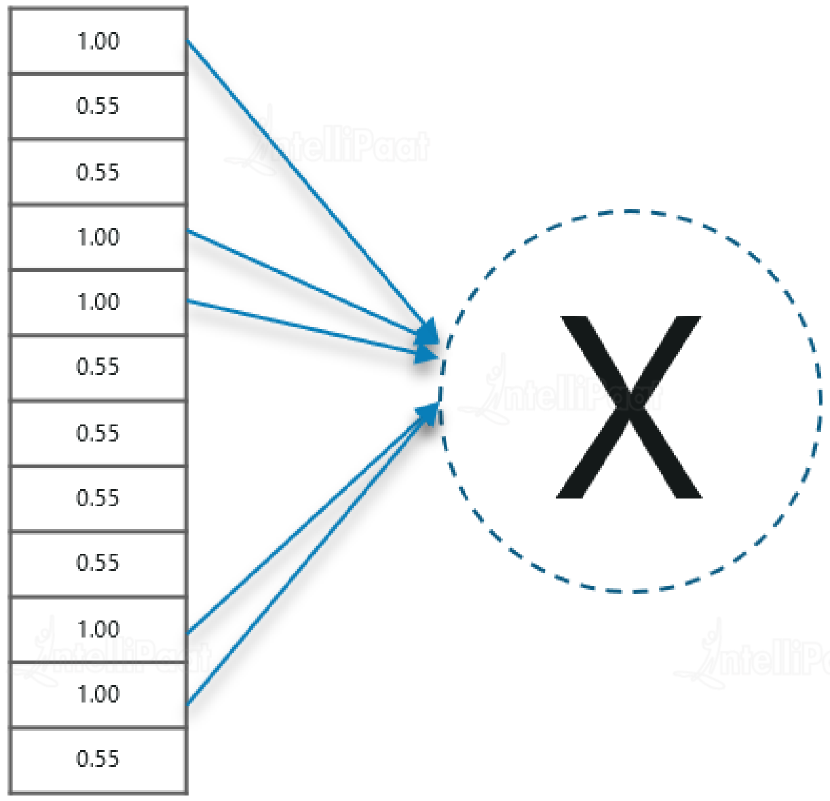
- Raw images get filtered, rectified, and pooled to create a set of shrunken, feature-filtered images. These can be filtered and shrunken again and again

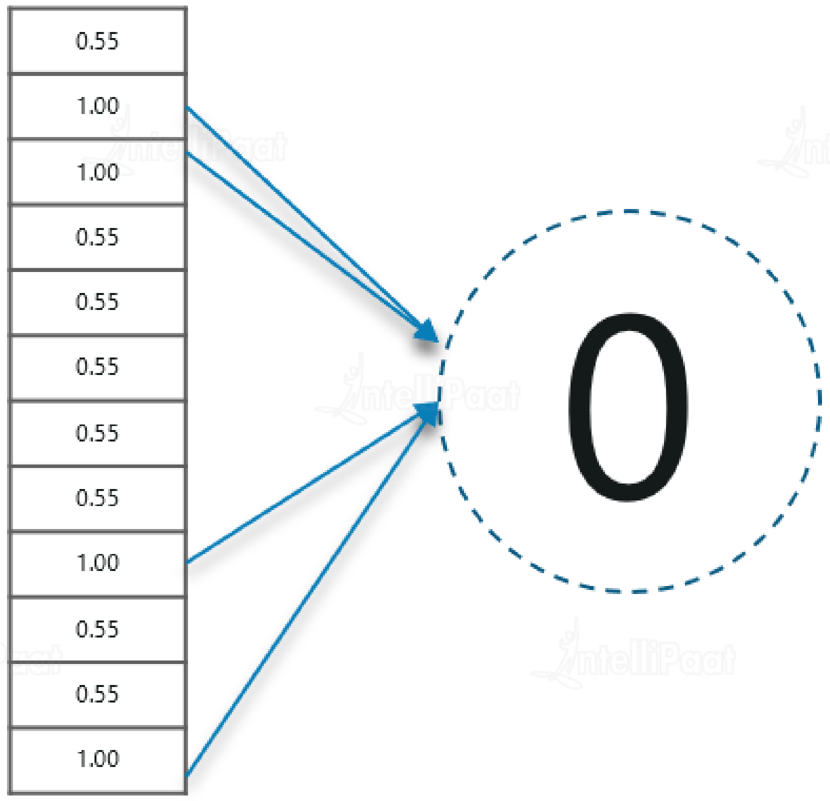
Fully Connected Layer

- It is the final layer where the classification actually happens
- Fully connected layers take the high-level filtered images and translate them into votes
- Here, we take our filtered and shrunk images and put them into one single list as shown in the figure
- Instead of treating inputs as a 2-dimensional array, they are treated as a single list, and all are treated identically
- Every value gets its own vote on whether the current image is an *X* or an *O*



Fully Connected Layer

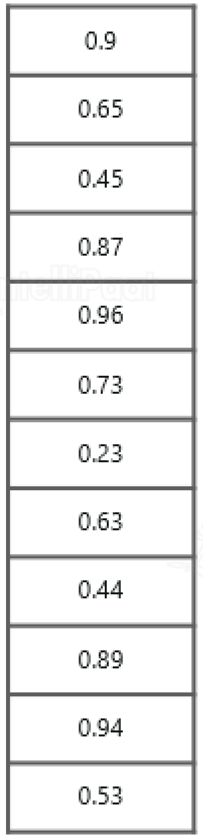




- Similarly, we will feed an image of O where we will have certain values which are high than others
- Some values are much better than the others at knowing when the image is an X, and some are particularly good at knowing when the image is an O
- These get larger votes than the others. These votes are expressed as weights, or connection strengths, between each value and each category
- We’re done with training the network, and now we can begin to predict and check the working of the classifier

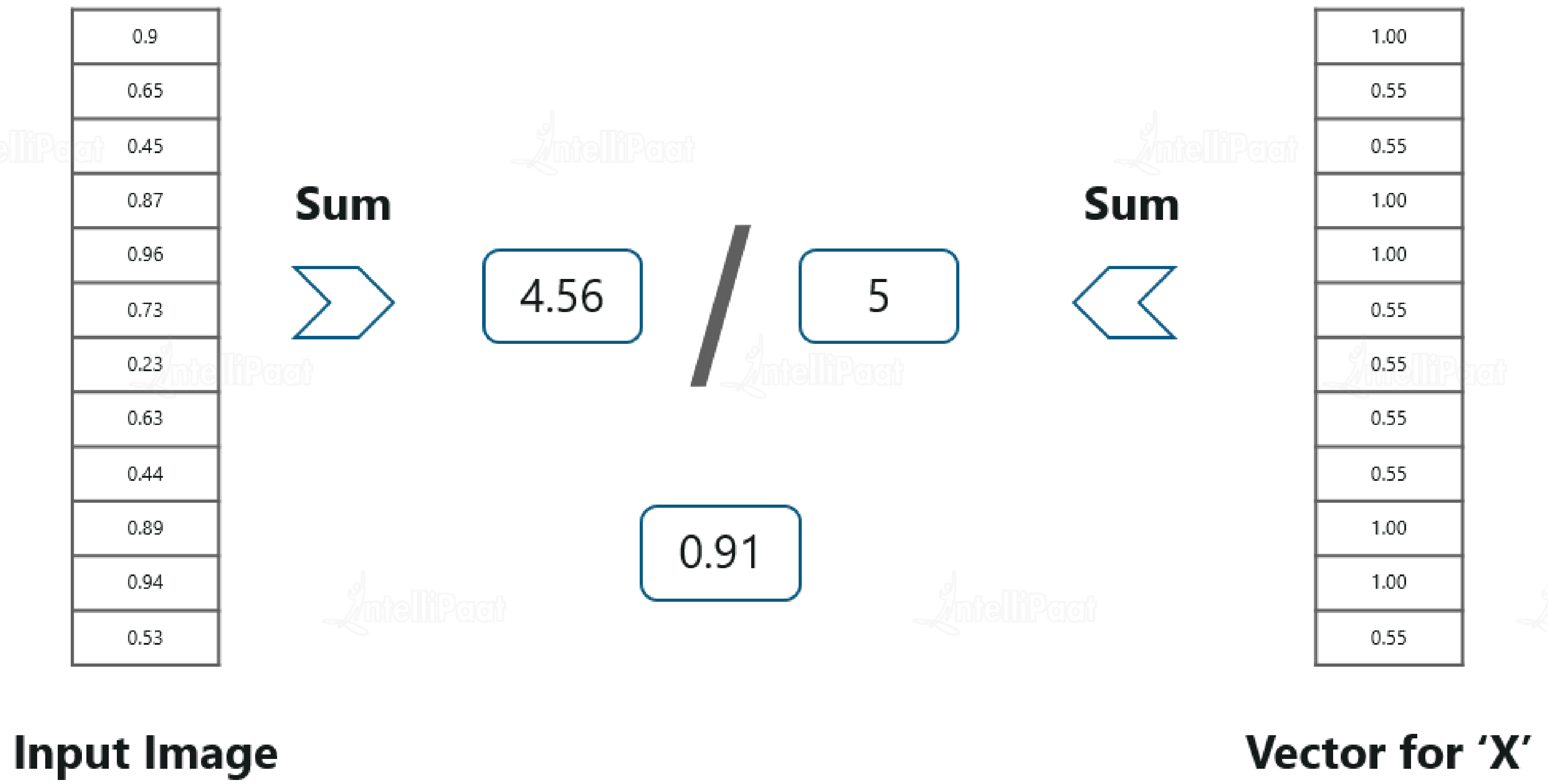
Fully Connected Layer

- Consider getting a new input where we have a 12-element vector obtained after passing the input through all layers of our network
- We make predictions based on the output data by comparing the obtained values with the list of ‘Xs’ and ‘Os’ to check what we’ve obtained is right or wrong



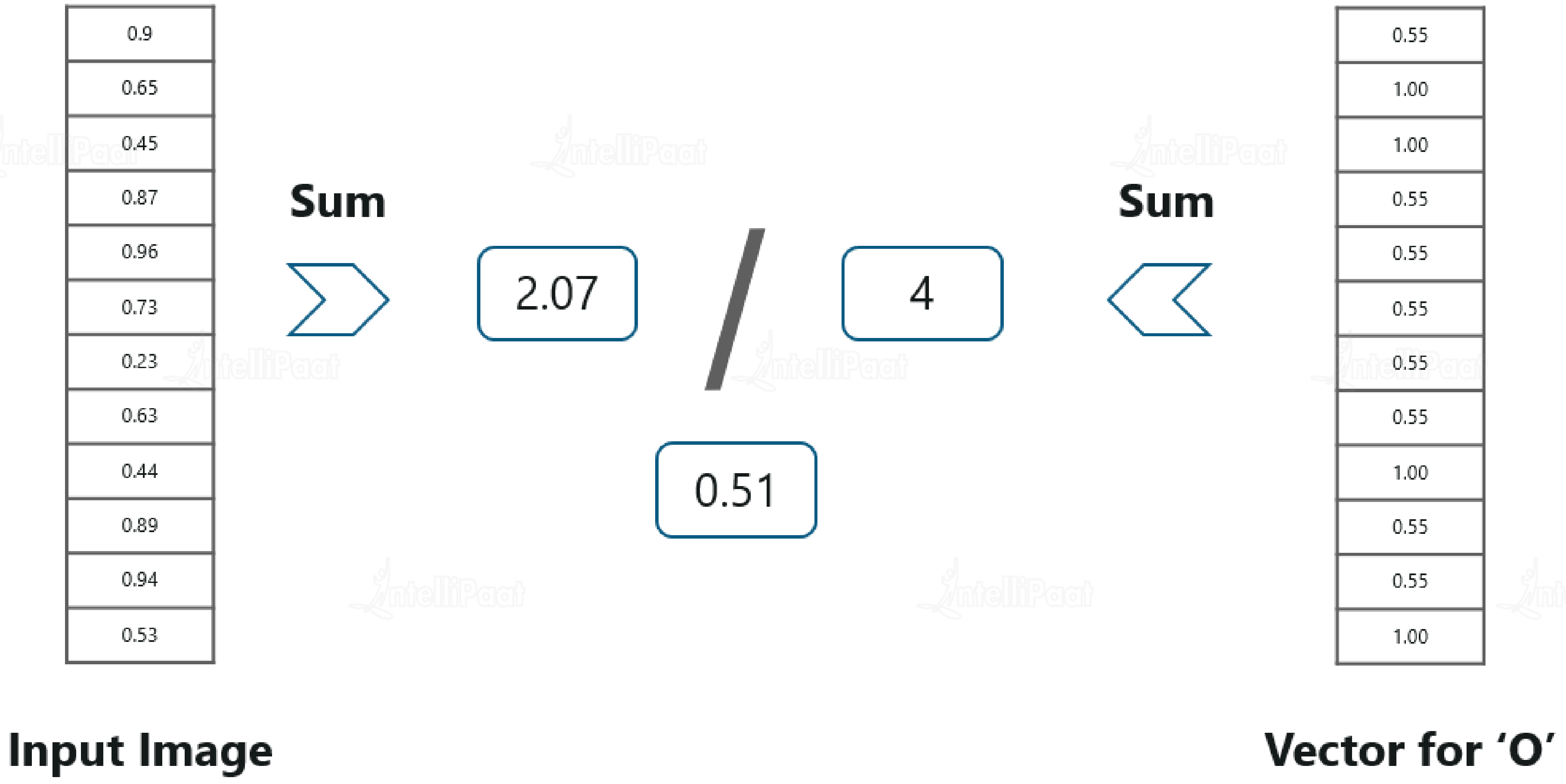
0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53

Fully Connected Layer



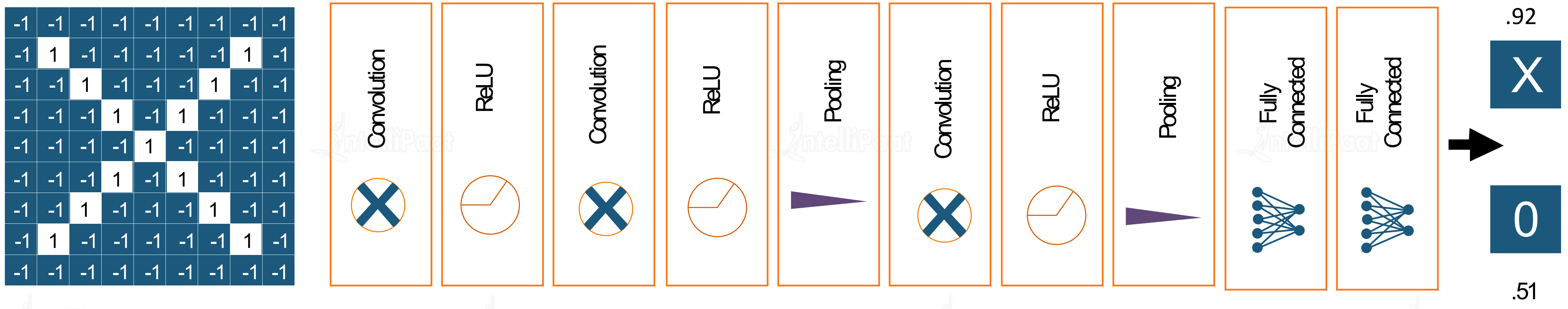
- We just added the values which we found out as high (1st, 4th, 5th, 10th, and 11th) from the vector table of *X* and we got the sum as 5
- We did exactly the same thing with the input image and got a value of 4.56
- When we divide the values, we have a probability match of 0.91!

Fully Connected Layer



- Doing the same with the vector table of *O*, we have an output of 0.51

Final Output



- "Well, since 0.51 is less than 0.91, the probability for the input image to be of an O is less, isn't it?"
- So, we can conclude that the resulting input image is of an 'X'!
- In practice, several fully connected layers are often stacked together, with each intermediate layer voting on phantom 'hidden' categories
- In effect, each additional layer lets the network learn ever more sophisticated combinations of features that help it make better decisions

Object Recognition with Convolutional Neural Networks in the Keras Deep Learning Library

Use Case 1

CIFAR-10 is an established computer-vision dataset used for object recognition. The CIFAR-10 data consists of 60,000 (32×32) color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images in the official data. The label classes in the dataset are:

airplane

automobile

bird

cat

deer

dog

frog

horse

ship

Truck

Let's look into full python implementation of object recognition task on CIFAR-10 dataset

Quiz

Quiz 1

Which of following activation function can’t be used at output layer to classify an image ?

A	sigmoid
B	Tanh
C	ReLU
D	None of the above

Answer 1

Which of following activation function can’t be used at output layer to classify an image ?

A	sigmoid
B	Tanh
[✓ C]	[✓ ReLU]
D	None of the above

Quiz 2

Which of the following statements is true when you use 1×1 convolutions in a CNN?

A	It can help in dimensionality reduction
B	It can be used for feature pooling
C	It suffers less overfitting due to small kernel size
D	All of the above

Answer 2

Which of the following statements is true when you use 1×1 convolutions in a CNN?

A It can help in dimensionality reduction

B It can be used for feature pooling

C It suffers less overfitting due to small kernel size

[✓ D All of the above]

Quiz 3

In a simple MLP model with 8 neurons in the input layer, 5 neurons in the hidden layer and 1 neuron in the output layer. What is the size of the weight matrices between hidden output layer and input hidden layer?

A	[1 X 5] , [5 X 8]
B	[8 X 5] , [1 X 5]
C	[8 X 5] , [5 X 1]
D	[5 x 1] , [8 X 5]

Answer 3

In a simple MLP model with 8 neurons in the input layer, 5 neurons in the hidden layer and 1 neuron in the output layer. What is the size of the weight matrices between hidden output layer and input hidden layer?

A	[1 X 5] , [5 X 8]
B	[8 X 5] , [1 X 5]
C	[8 X 5] , [5 X 1]
D	[5 x 1] , [8 X 5]

Thank you!

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor