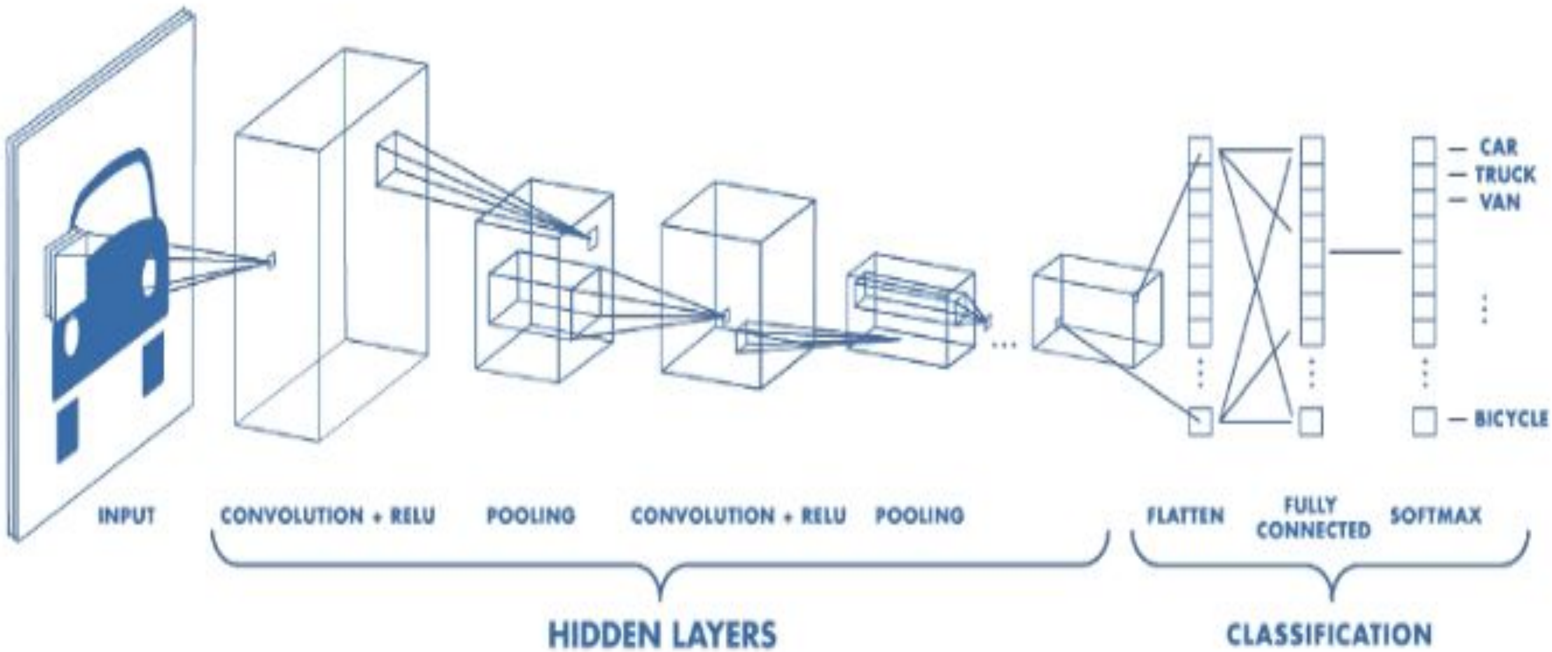
Convolution Neural Network (CNN)



INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
CAR
TRUCK
VAN
BICYCLE
HIDDEN LAYERS
CLASSIFICATION

Convolutional Neural Network (CNN)

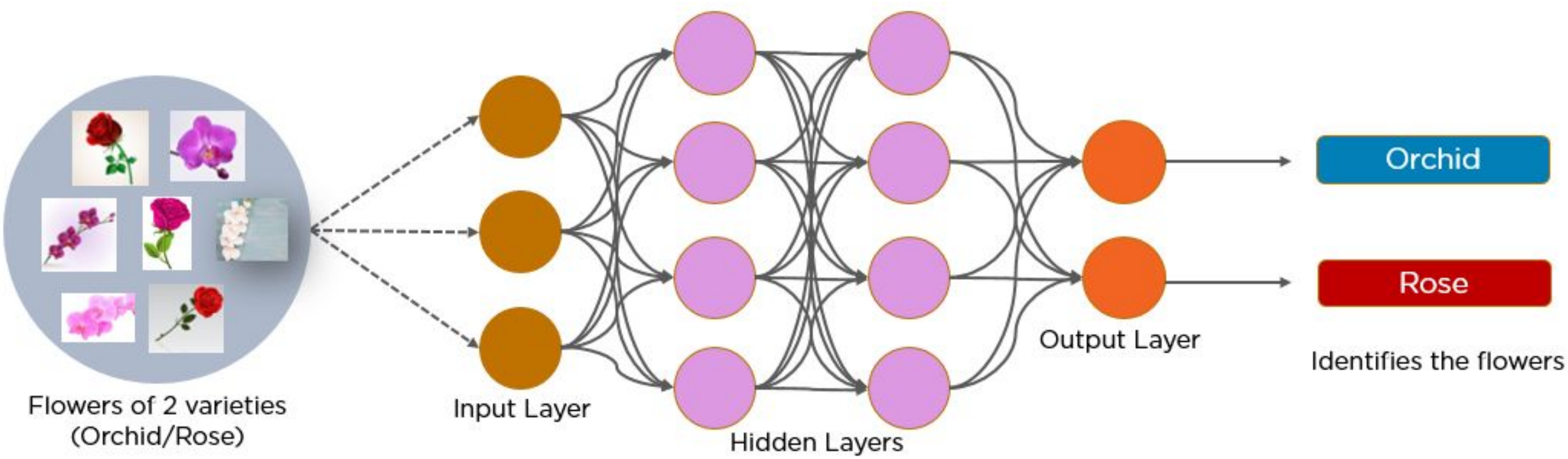
- The **architecture** behind Convolutional Neural Networks which are designed to **address image recognition** systems and **classification** problems.

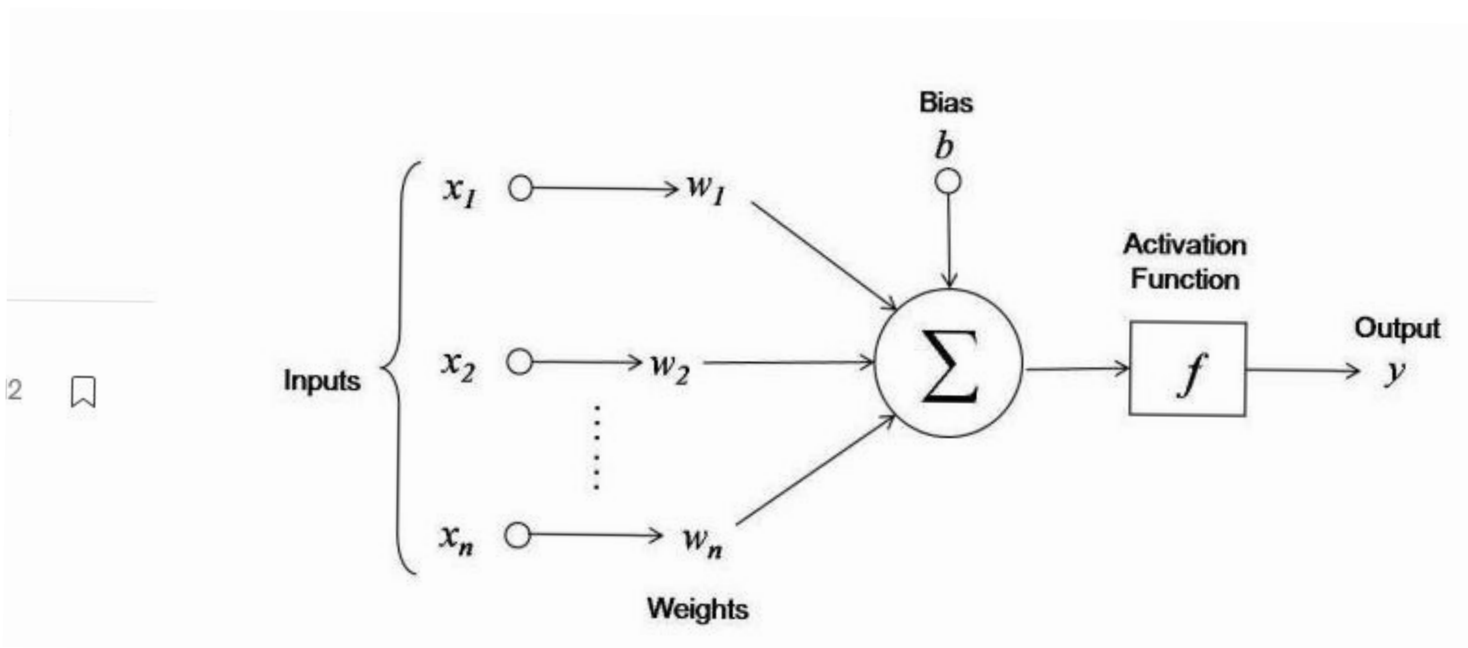
- Convolutional Neural Networks have **wide applications** in image and video recognition, recommendation systems and **natural language processing**.

- A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It's also known as a ConvNet.

- A convolutional neural network is used to detect and classify objects in an image.

- Below is a neural network that identifies two types of flowers: Orchid and Rose.

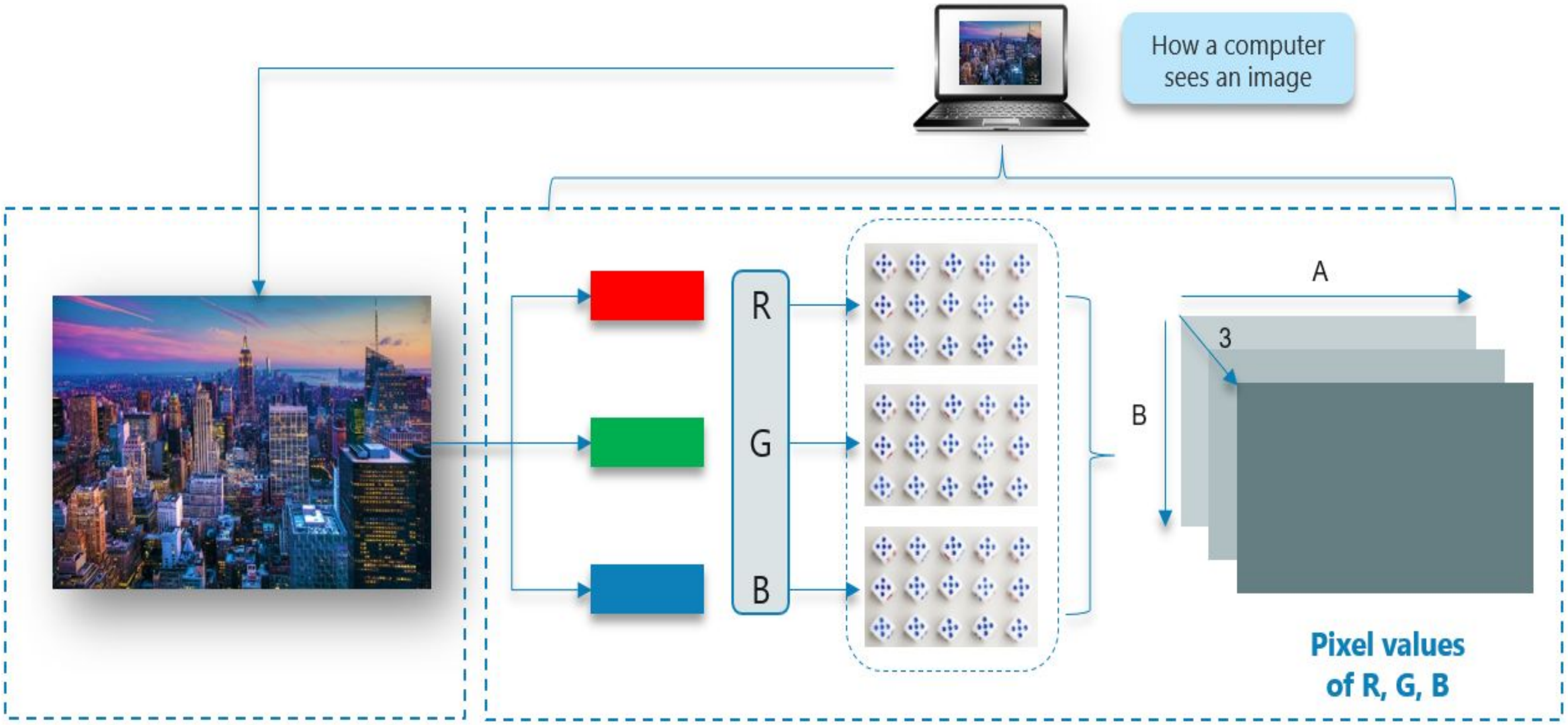



Bias
b
x1
x2
xn
w1
w2
wn
Inputs
Weights
Activation
Function
f
Output
y
2

How Does A Computer Read an Image?

- Consider this image of the **New York skyline**, upon first glance you will see a lot of **buildings** and **colors.** So how does the computer **process** this image?

- The image is **broken down** into 3 color-channels which is **Red, Green** and **Blue.** Each of these color channels are **mapped** to the the **image's pixel.**



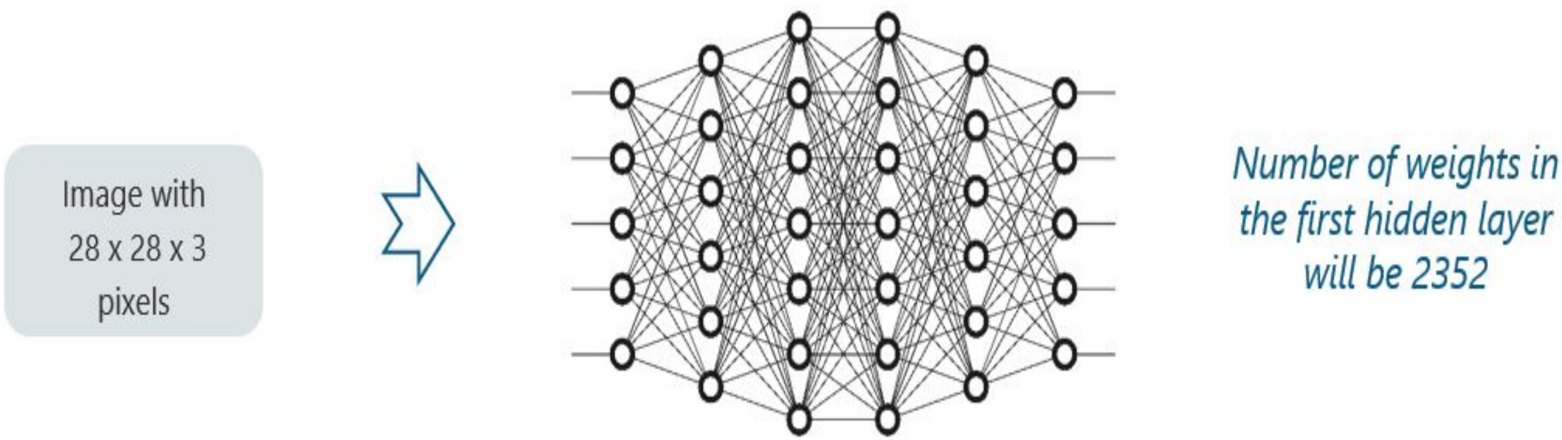
- Then, the **computer recognizes** the value associated with **each pixel** and **determine** the **size** of the image.

- However, for **black-white** images, there is only **one channel** and the **concept** is the **same.**

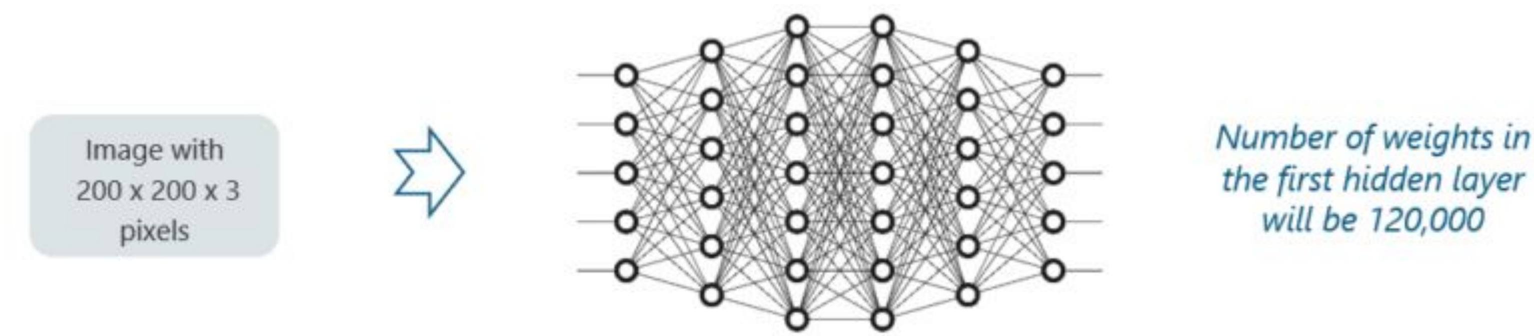
Why Not Fully Connected Networks?

- We cannot make use of fully connected networks when it comes to Convolutional Neural Networks, here's why!

- Consider the following image:



- Here, we have **considered** an **input** of images with the size **28x28x3** pixels. If we **input** this to our Convolutional Neural Network, we will have about **2352 weights** in the **first** hidden layer itself.
- But this case **isn’t practical**. Now, take a look at this:



- Any generic input image will atleast have 200x200x3 pixels in size.
- The size of the first hidden layer becomes a whooping 120,000.
- If this is just the first hidden layer, imagine the number of neurons needed to process an entire complex image-set.
- This leads to over-fitting and isn’t practical. Hence, we cannot make use of fully connected networks.

What Are Convolutional Neural Networks?

- Convolutional Neural Networks, like neural networks, are made up of **neurons** with **learnable weights** and **biases**.

- Each **neuron** receives several **inputs**, takes a weighted **sum** over them, **pass** it through an **activation function** and responds with an **output**.

- The whole network has a **loss function** and all the tips and tricks that we developed for neural networks still apply on **Convolutional Neural Networks.**

- **Neural networks**, as its name suggests, is a **machine learning technique** which is modeled after the **brain** structure. It comprises of a network of **learning units** called neurons.

- These **neurons** learn how to convert **input signals** (e.g. picture of a cat) into corresponding **output signals** (e.g. the label "cat"), forming the basis of automated recognition.

- Let's take the example of **automatic image recognition.** The process of **determining** whether a **picture** contains a **cat** involves an **activation function**. If the picture resembles prior cat images the neurons have **seen before,** the label **"cat"** would be **activated.**

- **Hence,** the **more** labeled images the neurons are **exposed** to, the **better** it learns how to recognize other unlabelled images. We call this the process of **training** neurons.

Origin Of Convolutional Neural Networks

- The intelligence of neural networks is **uncanny.**
- While artificial neural networks were **researched** as early in **1960s** by **Rosenblatt**
- It was only in late **2000s** when deep learning using neural networks **took off.**
- The key **enabler** was the scale of **computation power** and **datasets** with **Google** pioneering research into deep learning.
- In July 2012, researchers at **Google** exposed an **advanced neural network** to a series of **unlabelled,** static **images** sliced from **YouTube** videos.
- To their **surprise,** they discovered that the neural network **learned** a **cat-detecting** neuron on its **own,** supporting the popular assertion that **“the internet is made of cats”.**

Layers in a Convolutional Neural Network

- A convolution neural network has multiple hidden layers that help in extracting information from an image. The four important layers in CNN are:
  - Convolution layer
  - ReLU layer
  - Pooling layer
  - Fully connected layer

Convolution Layer

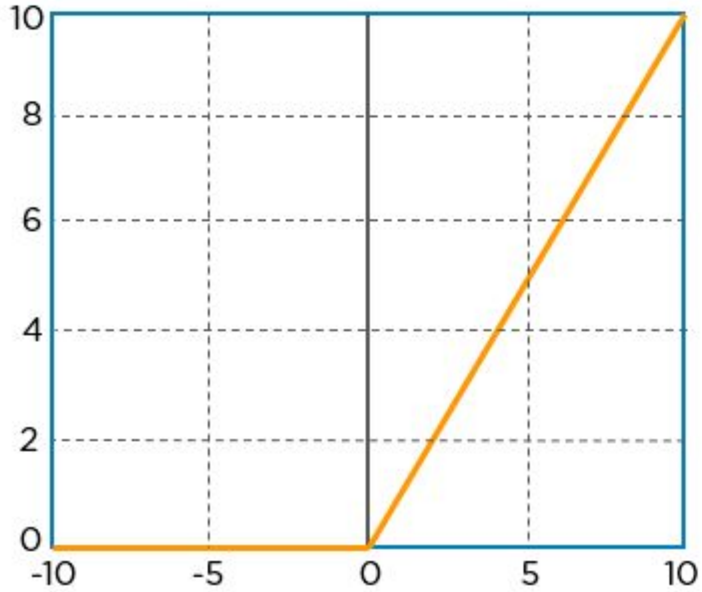
- This is the first step in the process of extracting valuable features from an image.
- A convolution layer has several filters that perform the convolution operation.
- Every image is considered as a matrix of pixel values.
- Consider the following 5x5 image whose pixel values are either 0 or 1.
- There’s also a filter matrix with a dimension of 3x3.
- Slide the filter matrix over the image and compute the dot product to get the convolved feature matrix.

ReLU layer

- ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer.

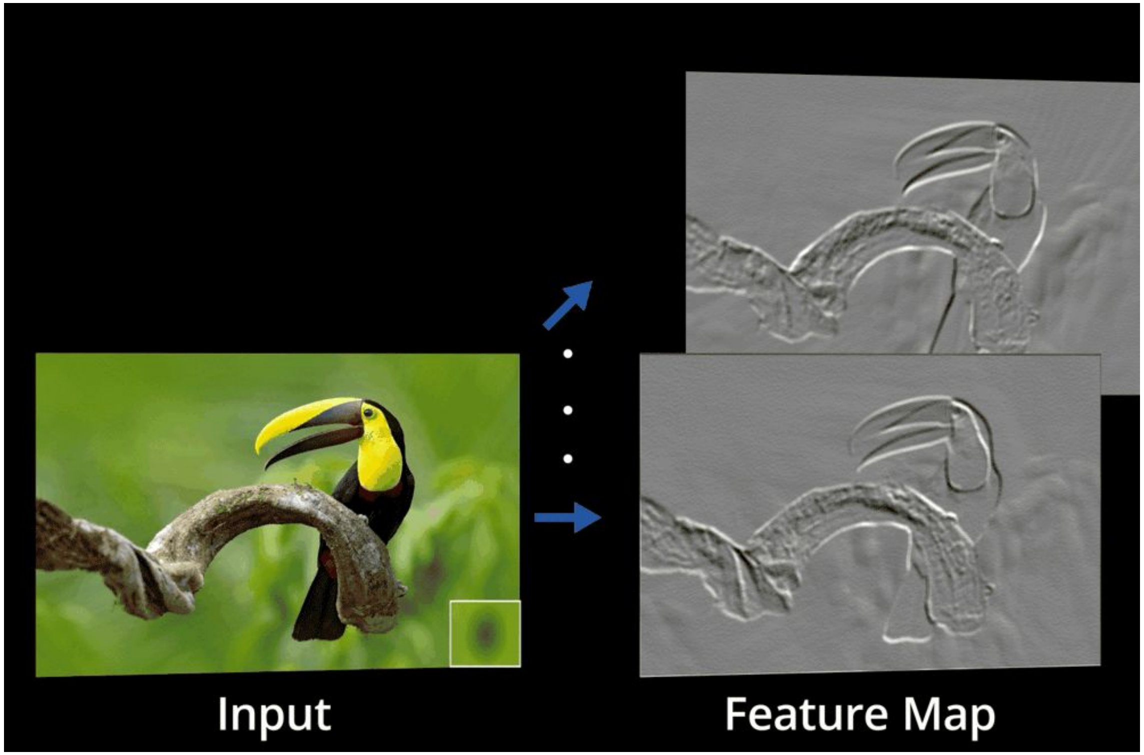
- ReLU performs an element-wise operation and sets all the negative pixels to 0. It introduces non-linearity to the network, and the generated output is a rectified feature map. Below is the graph of a ReLU function:

- The original image is scanned with multiple convolutions and ReLU layers for locating the features.

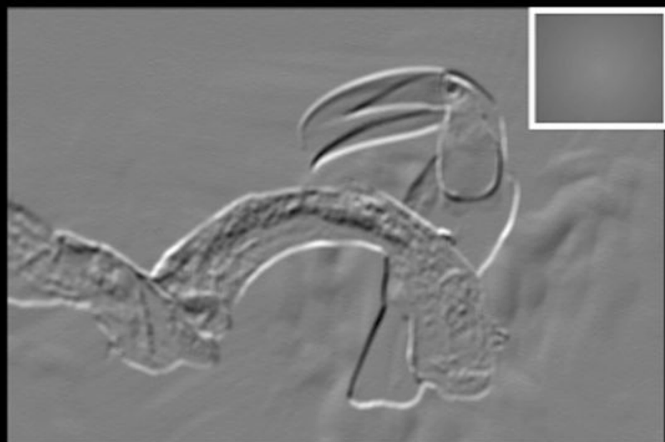


R(z) = max(0, z)

- The original image is scanned with multiple convolutions and ReLU layers for locating the features.



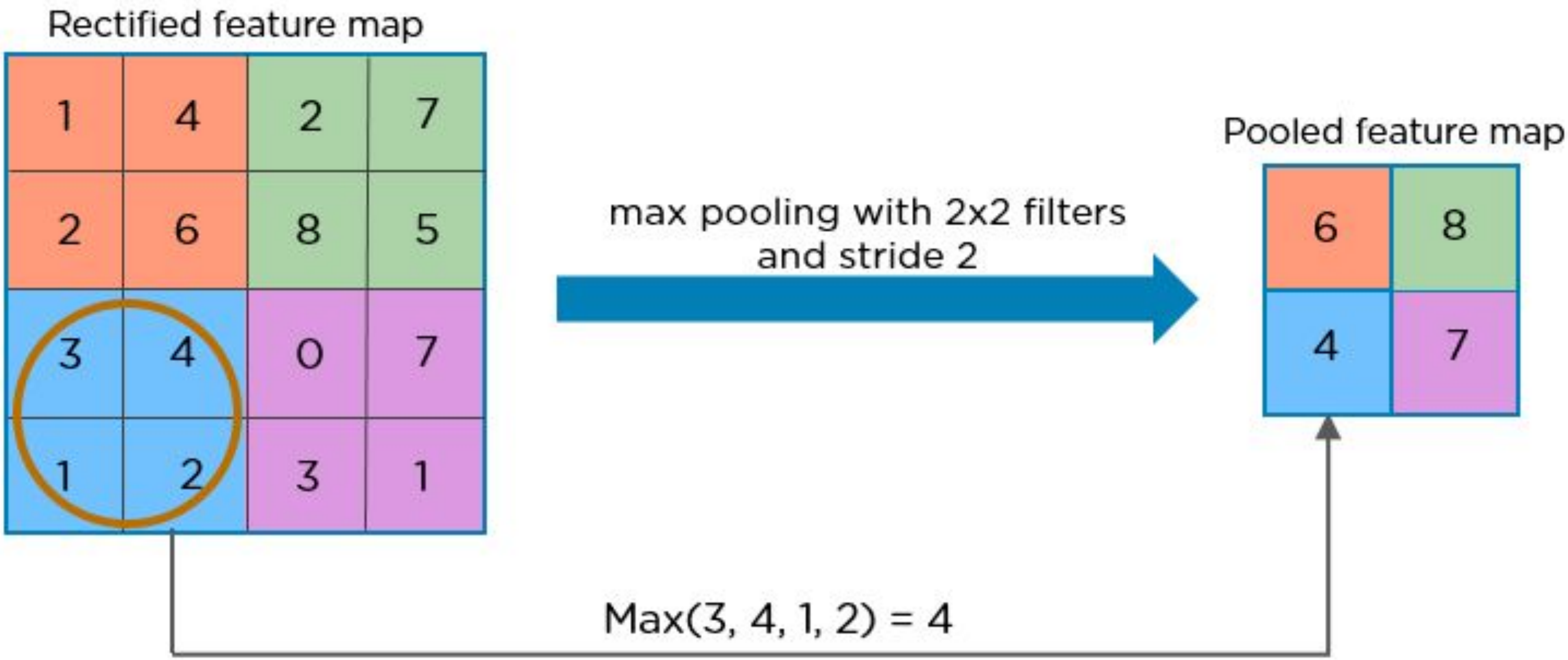
Input Feature Map



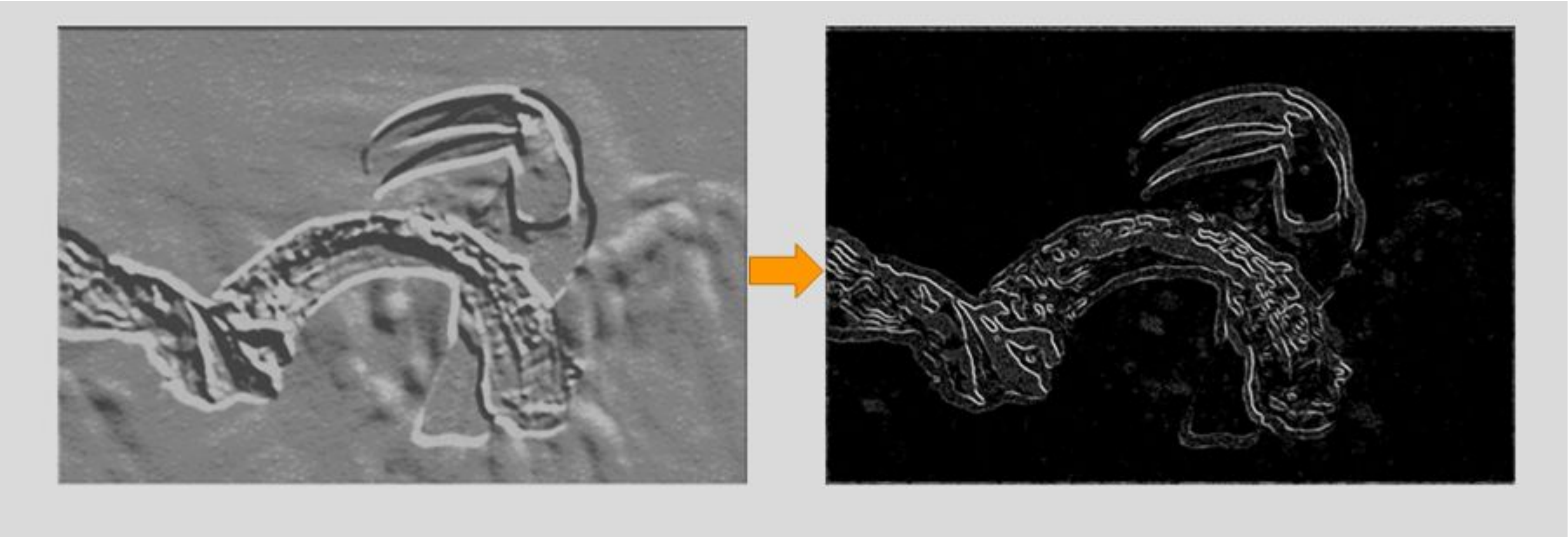
Pooling Layer

- Pooling is a down-sampling operation that reduces the dimensionality of the feature map.

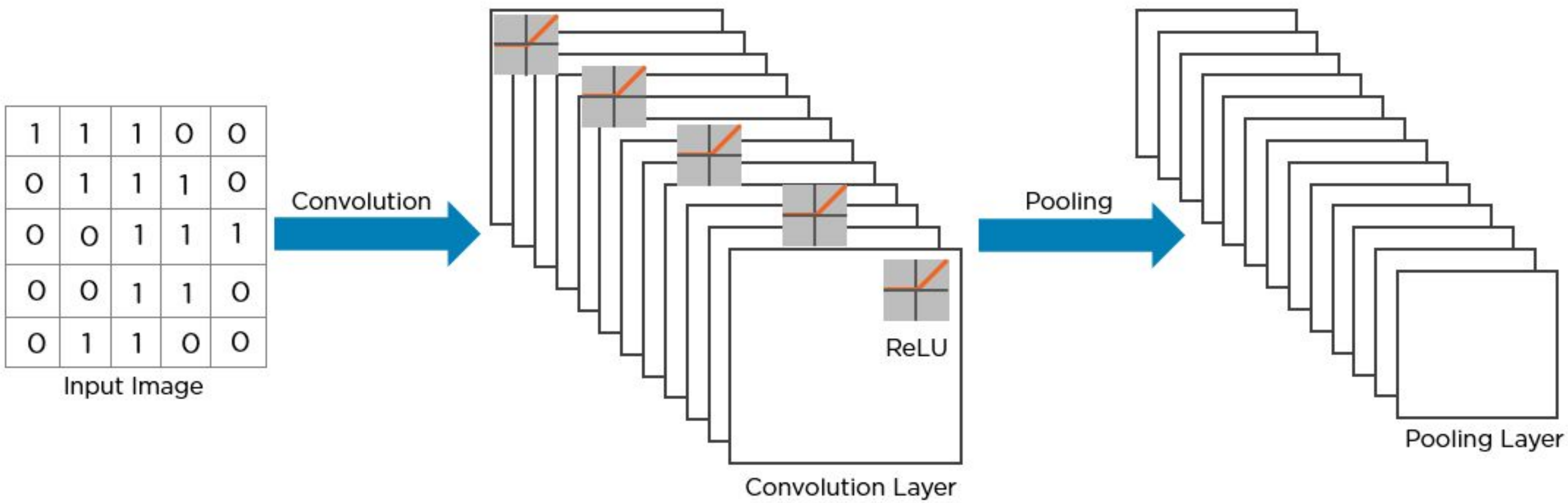
- The rectified feature map now goes through a pooling layer to generate a pooled feature map.


Rectified feature map
1
4
2
7
2
6
8
5
3
4
0
7
1
2
3
1
max pooling with 2x2 filters
and stride 2
Pooled feature map
6
8
4
7
Max(3, 4, 1, 2) = 4

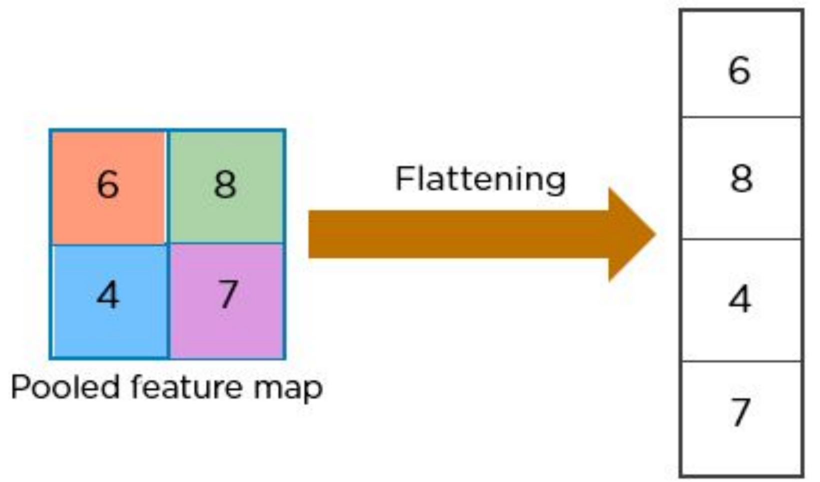
- The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes, and beak.



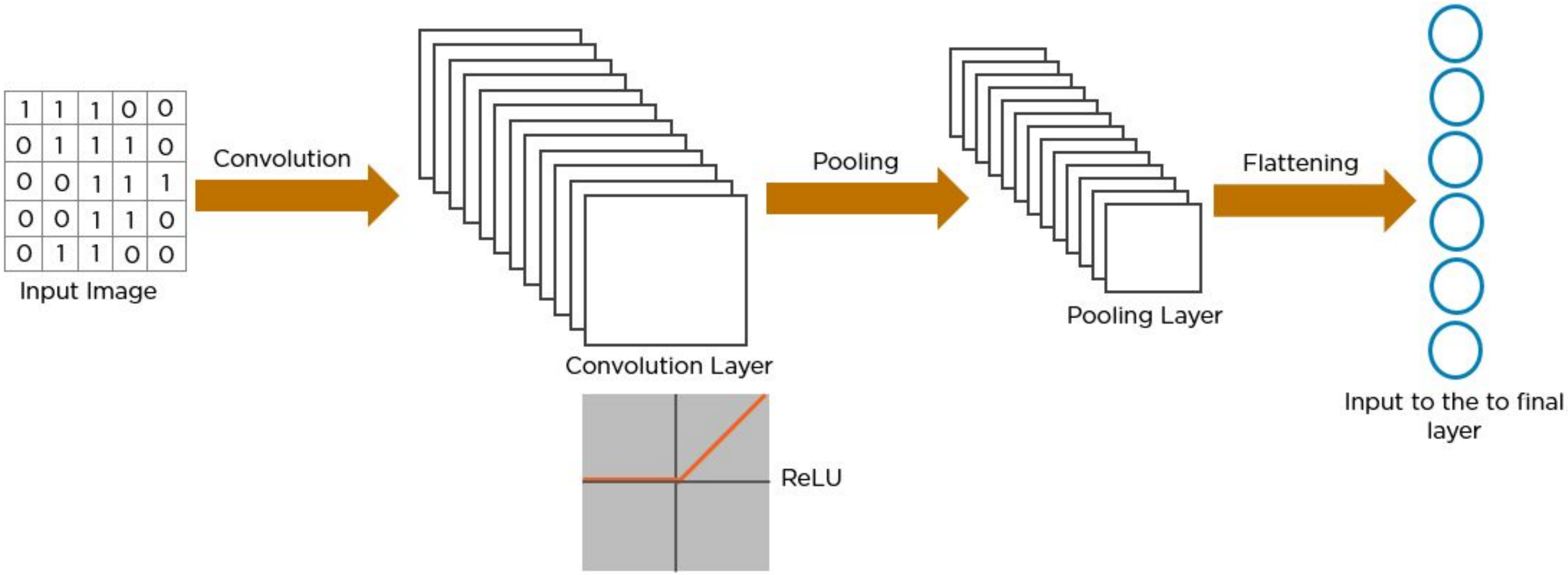
- Here’s how the structure of the convolution neural network looks so far:

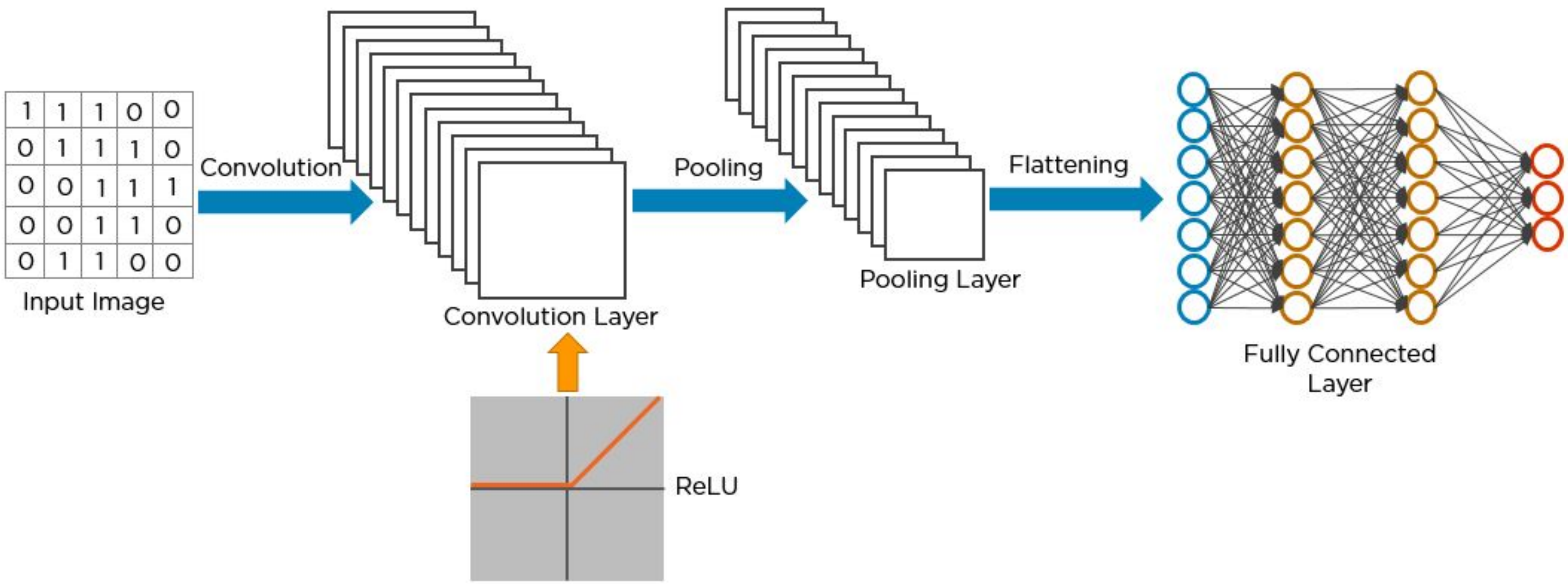


- The next step in the process is called flattening. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.

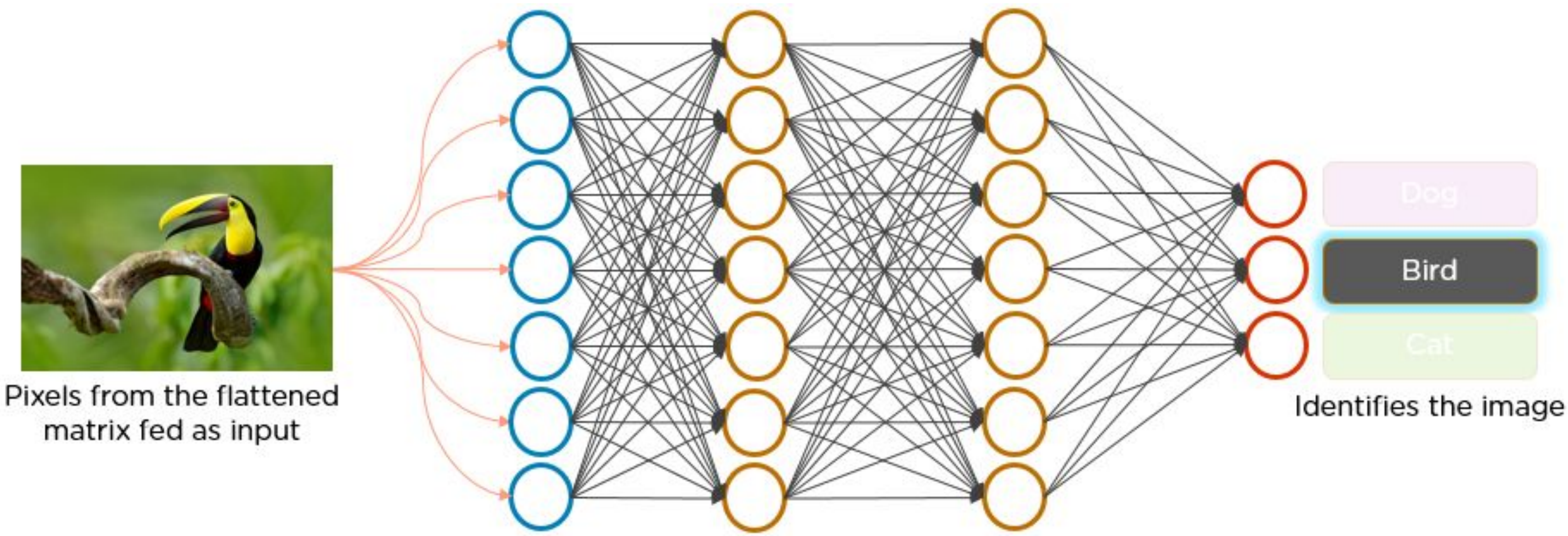


- The flattened matrix is fed as input to the fully connected layer to classify the image.





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
Input Image
Convolution
Convolution Layer
ReLU
Pooling
Pooling Layer
Flattening
Fully Connected Layer

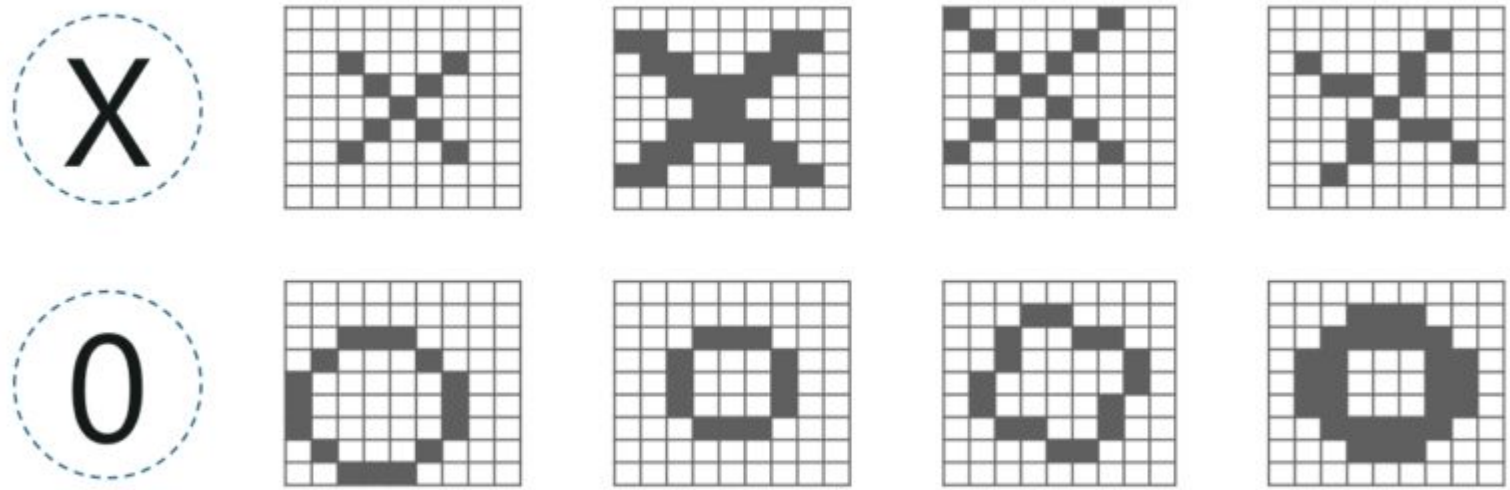

Pixels from the flattened matrix fed as input
Dog
Bird
Cat
Identifies the image

How Do Convolutional Neural Networks Work with Example?

- There are **four** layered **concepts** we should understand in Convolutional Neural Networks:
  - Convolution,
  - ReLu,
  - Pooling and
  - Full Connectedness (Fully Connected Layer).

Example of CNN:

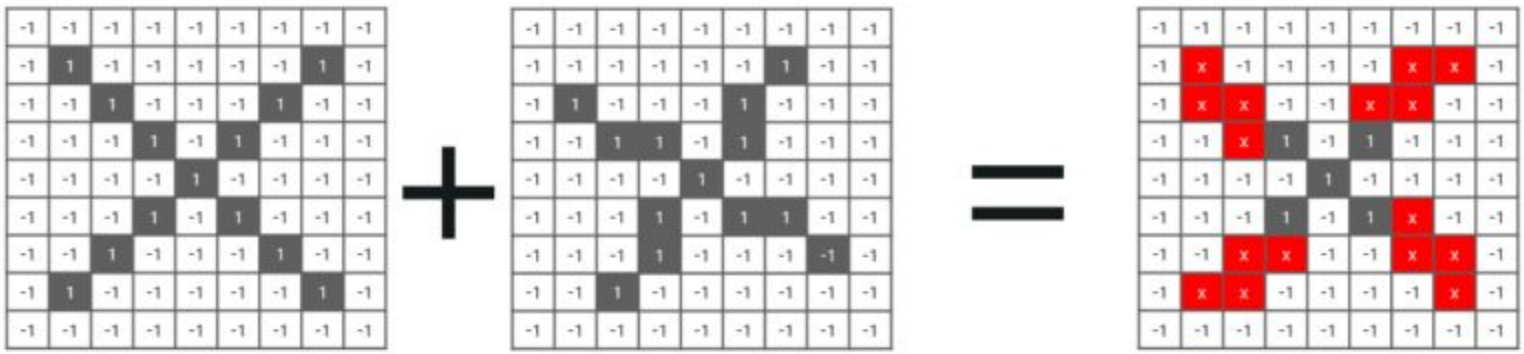
- Consider the image below:



- Here, there are multiple renditions of X and O's. This makes it tricky for the computer to recognize. But the goal is that if the **input signal** looks like **previous** 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.**

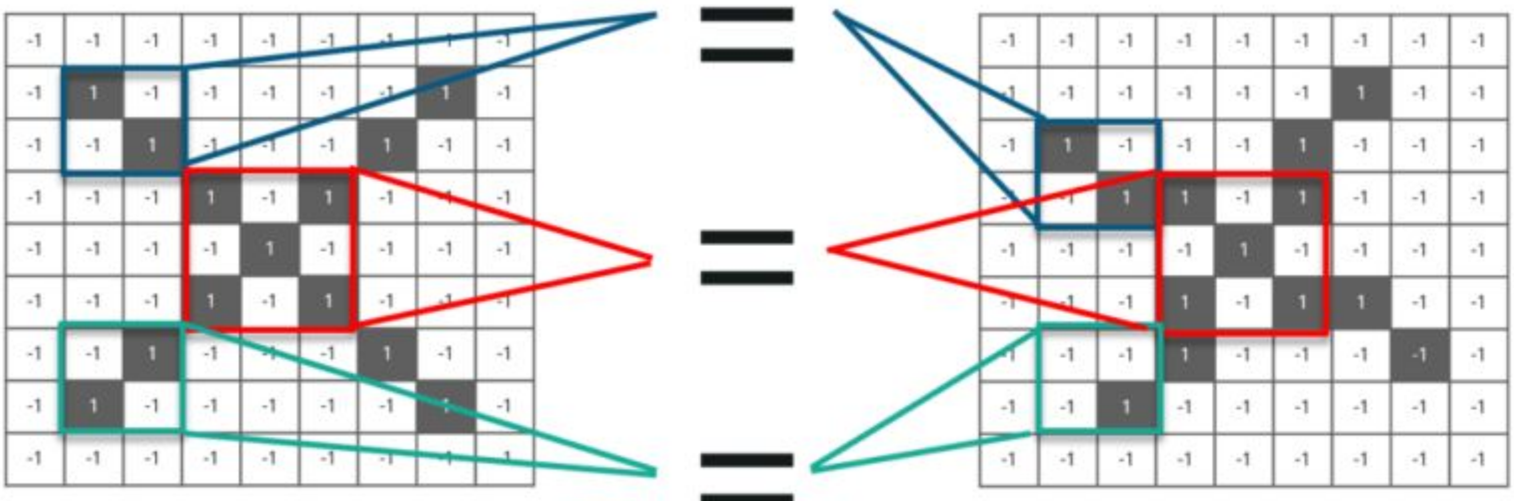
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

- So, the **computer understands** every pixel. In this case, the **white** pixels are said to be **-1** while the **black** ones are **1.** This is just the way we've implemented to **differentiate the pixels** in a basic binary classification.



- Now if we would just **normally search** and **compare** the **values** between a normal image and another **‘x’ rendition,** we would get a **lot** of **missing pixels.**

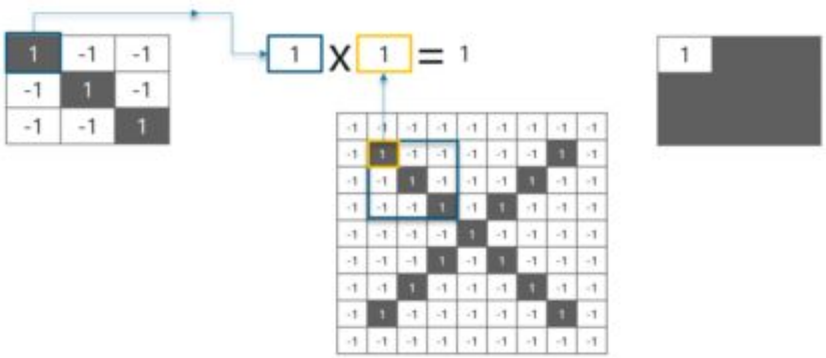
So, how do we fix this?

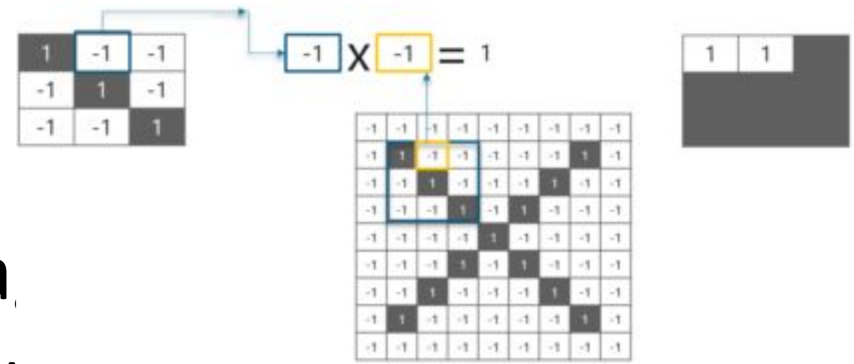


- We take **small patches** of the pixels called **filters** and try to **match** them in the corresponding **nearby** locations to see if we get a **match.** By doing this, the Convolutional Neural Network **gets a lot better** at seeing **similarity** than directly trying to match the **entire image.**

Convolution Of An Image

- Convolution has the nice property of being **translational invariant**. Intuitively, this means that **each** convolution filter represents a **feature** of interest (e.g **pixels in letters)** and the Convolutional Neural Network **algorithm** learns which **features** comprise the **resulting reference** (i.e. alphabet).
- We have **4 steps** for convolution:
  - **Line up** the feature and the image
  - **Multiply** each **image** pixel by corresponding **feature** pixel
  - **Add** the values and find the **sum**
  - **Divide** the sum by the **total** number of pixels in the **feature**





- Consider the above image – we are **done** with the first **2 steps**. We considered a **feature image** and **one pixel** from it. We **multiplied** this with the **existing image** and the product is stored in another **buffer feature image**.

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

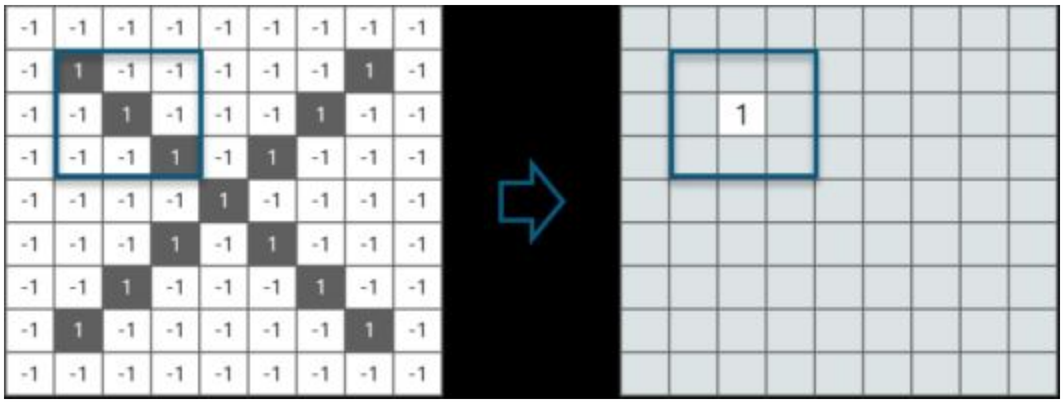
$$\frac{1+1+1+1+1+1+1+1+1}{9} = 1$$

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

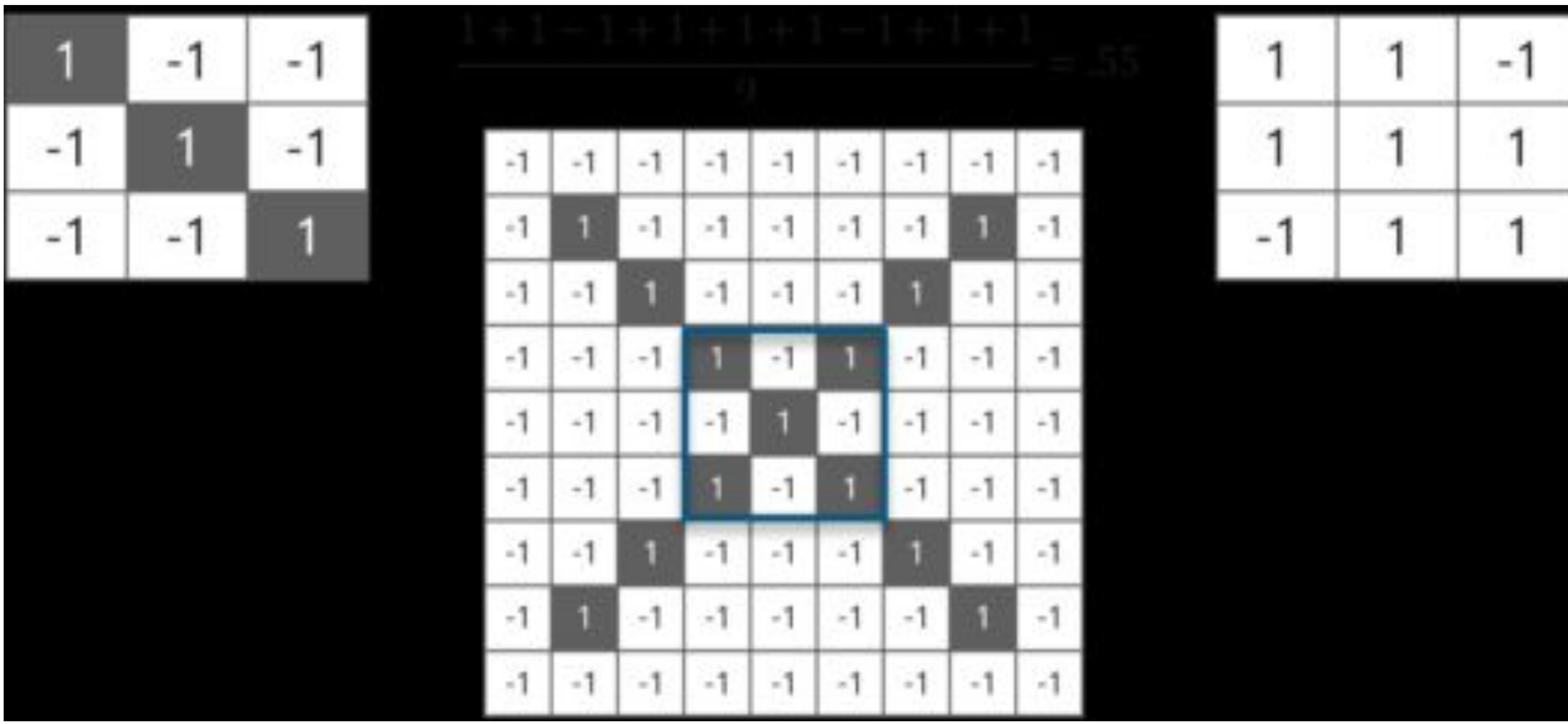
- With this **image,** we completed the l**ast 2 steps.**
- We added the **values** which led to the **sum.**
- We then, **divide** this **number** by the **total** number of pixels in the **feature image.**

- When that is done, the **final value** obtained is placed at the **center** of the **filtered image** as shown below:

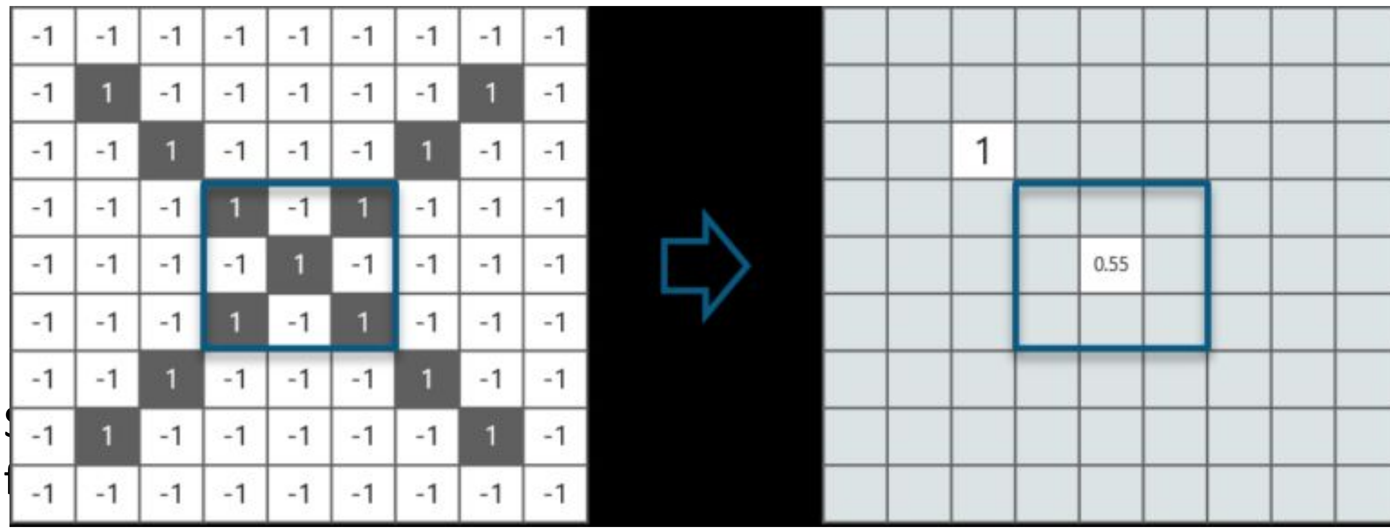


- Now, we can **move** this **filter** around and do the **same** at **any pixel** in the image. For **better clarity,** let’s consider **another example:**

- $$\frac{1+1-1+1+1+1-1+1+1}{9}=.55$$



- As you can see, here after performing the first 4 steps we have the value at 0.55! We take this value and place it in the image as explained before. This is done in the following image:



- S ... ee how the f

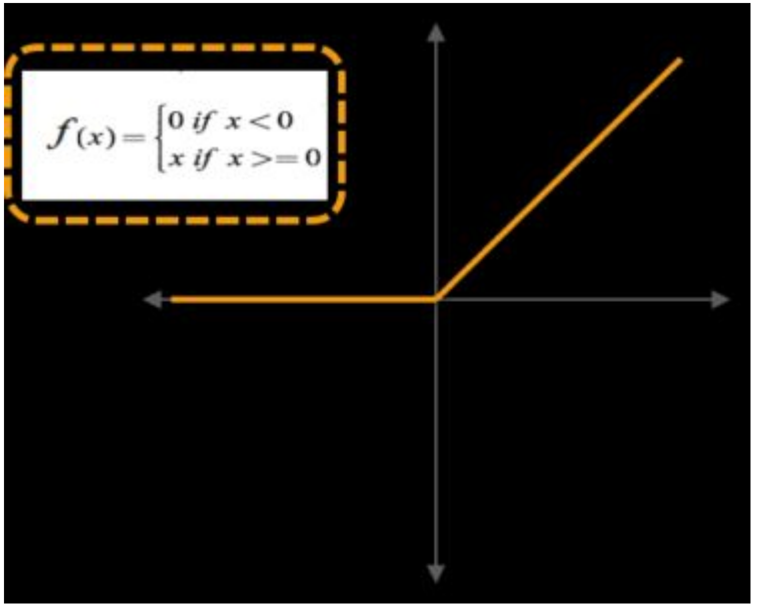
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

- Here we considered just one filter. Similarly, we will perform the same convolution with every other filter to get the convolution of that filter.
- The **output** signal **strength** is not dependent on where the **features** are located, but simply whether the **features** are **present.** Hence, an alphabet could be sitting in **different positions** and the **Convolutional Neural Network** algorithm would still be able to **recognize it.**

ReLU Layer

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

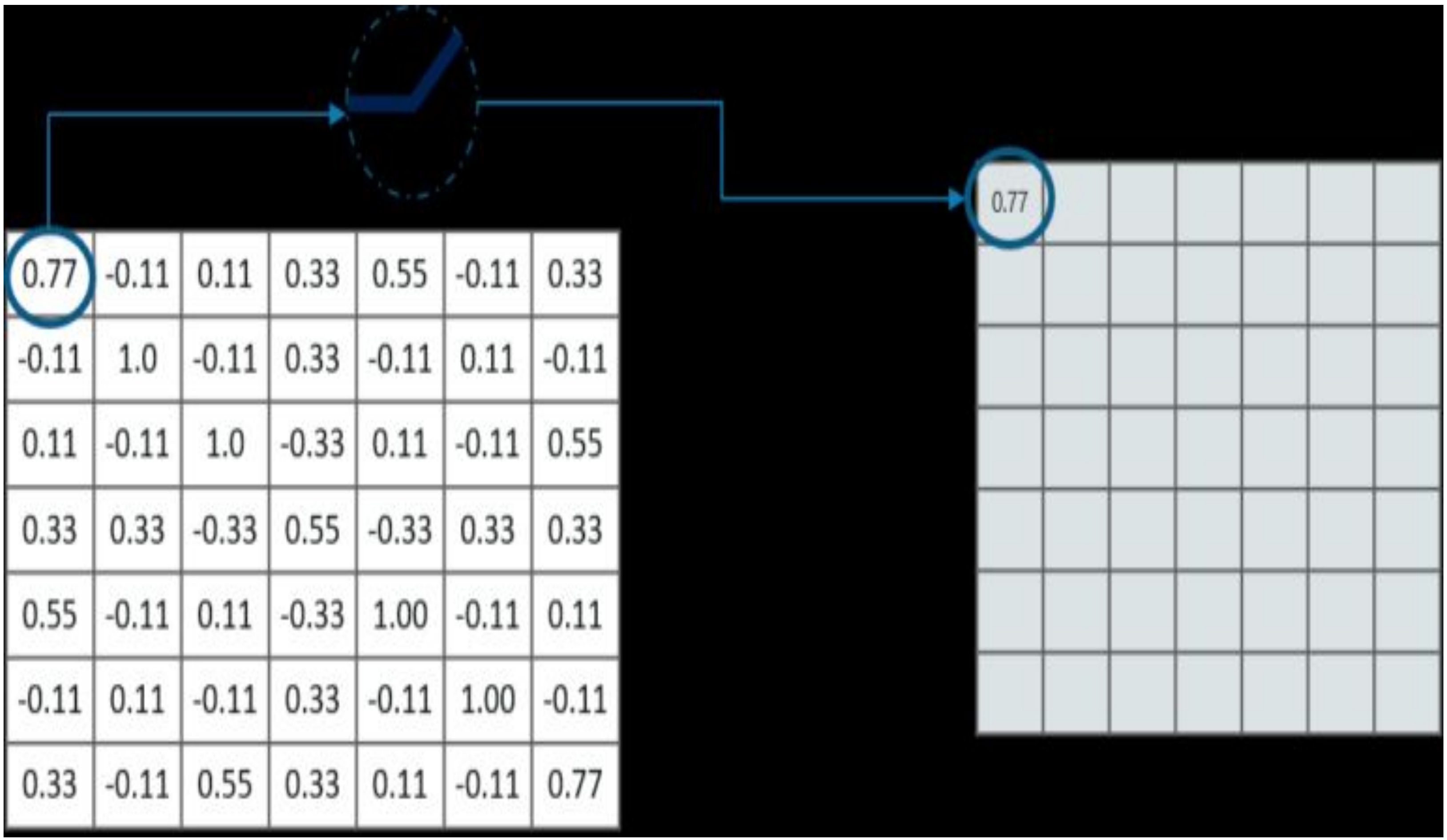
- Consider the below example



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

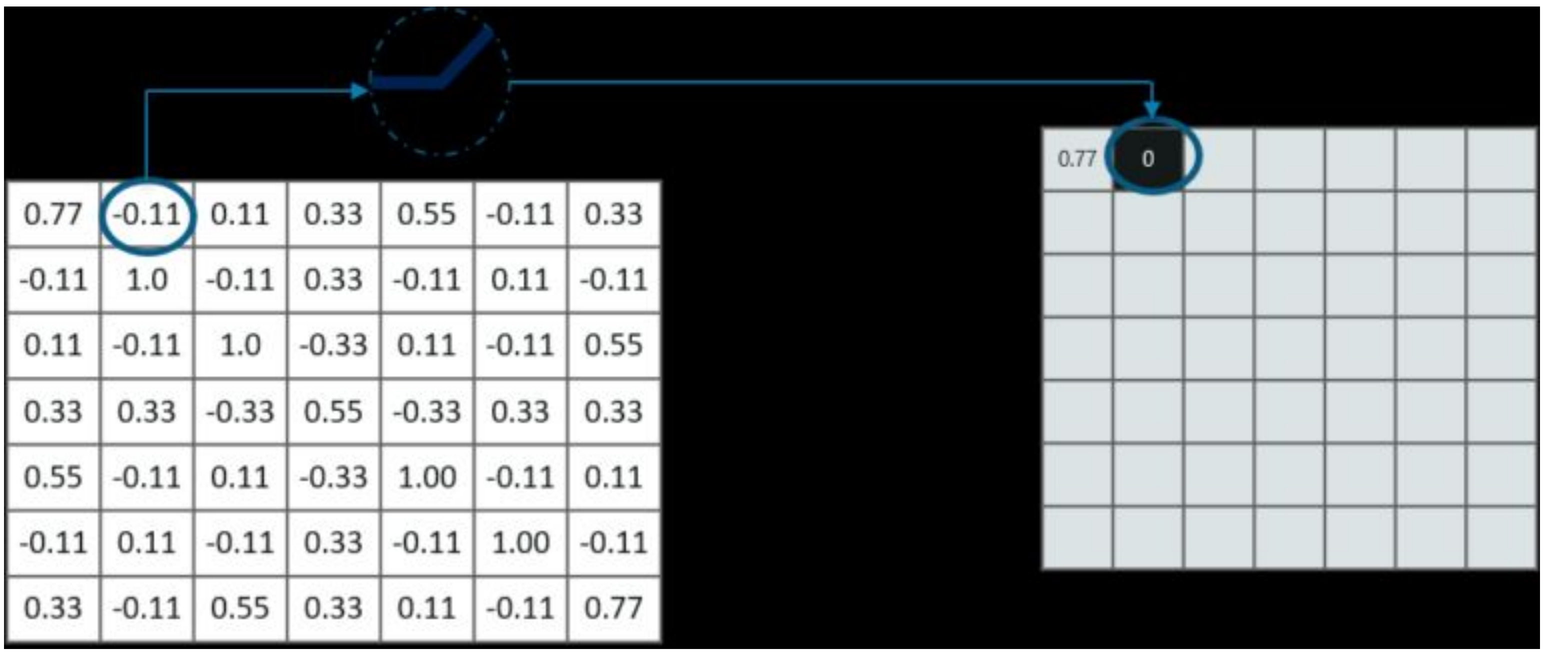
x	$f(x)=x$	F(x)
-3	$f(-3) = 0$	0
-5	$f(-5) = 0$	0
3	$f(3) = 3$	3
5	$f(5) = 5$	5

Why do we require ReLU here



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-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

- The main aim is to remove all the negative values from the convolution. All the positive values remain the same but all the negative values get changed to zero as shown below:



- So after we process this particular feature we get the following output:

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-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	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	1.77

- Now, similarly we do the same process to all the other feature images as well:

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-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

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

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

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	1.77

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

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

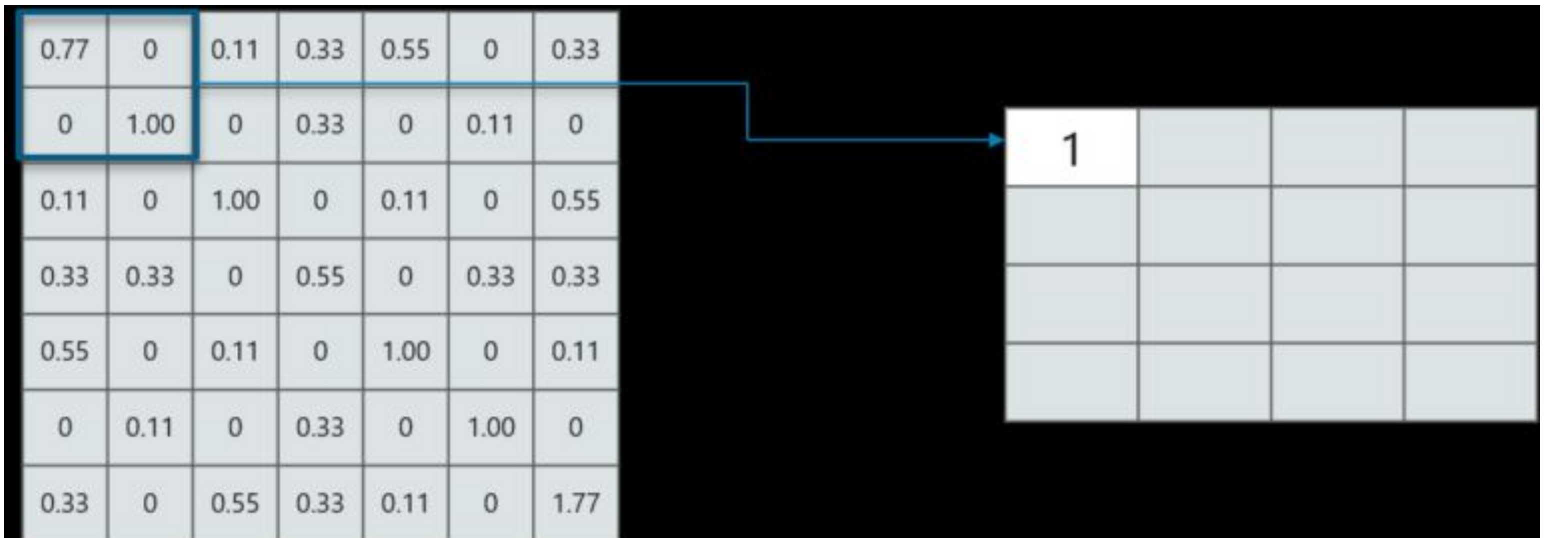
- **Inputs** from the convolution layer can be **"smoothened"** to **reduce** the **sensitivity** of the **filters** to **noise** and **variations.** This smoothing process is called **subsampling** and can be **achieved** by taking **averages** or taking the **maximum** over a **sample** of the signal.

Pooling Layer

- In this layer we **shrink** the **image** stack into a **smaller size.** Pooling is done **after passing** through the **activation** layer.

- We do this by implementing the following 4 steps:
  - Pick a **window size** (usually 2 or 3)
  - Pick a **stride** (usually 2)
  - **Walk** your window **across** your **filtered** images
  - From each **window,** take the **maximum** value

- Let us understand this with an example. Consider performing pooling with a window size of 2 and stride being 2 as well.



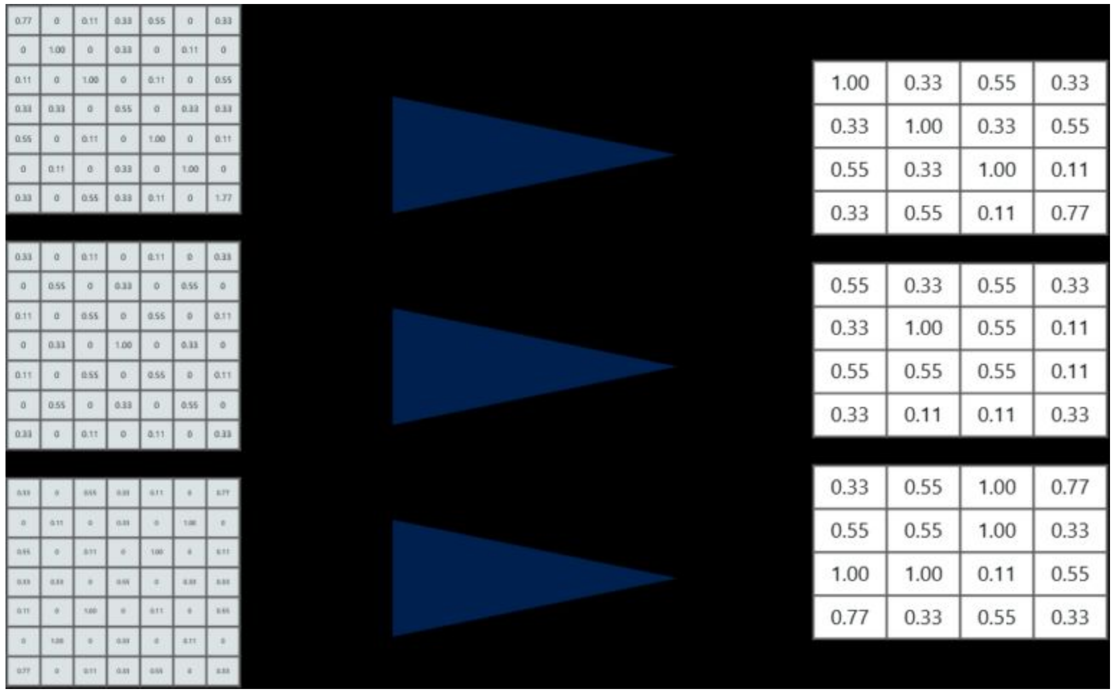
- So in this case, we took **window size** to be **2** and we got **4 values** to choose from. From those 4 values, the **maximum value** there is 1 so we pick 1. Also, note that we **started out** with a **7×7** matrix but now the same matrix after **pooling** came down to **4×4.**

- But we need to **move** the **window across** the **entire** image. The procedure is exactly as same as above and we need to repeat that for the entire image.

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	1.77

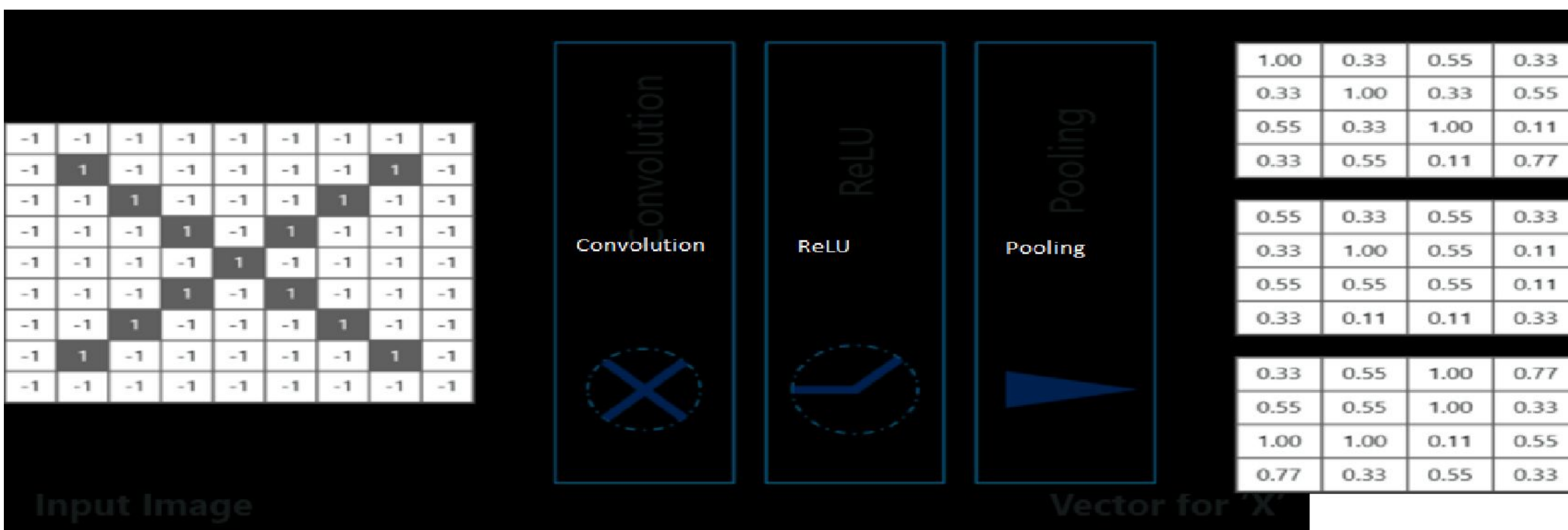
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

- Do note that this is for **one filter.** We need to do it for 2 other filters as well. This is done and we arrive at the following result:



- Well the **easy part** of this **process** is **over.** Next up, we need to **stack up all these layers!**

- **Stacking Up The Layers**
- So to get the **time-frame** in one picture we're here with a **4×4** matrix from a **7×7** matrix after passing the input through 3 layers – **Convolution, ReLU** and **Pooling** as shown below:



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

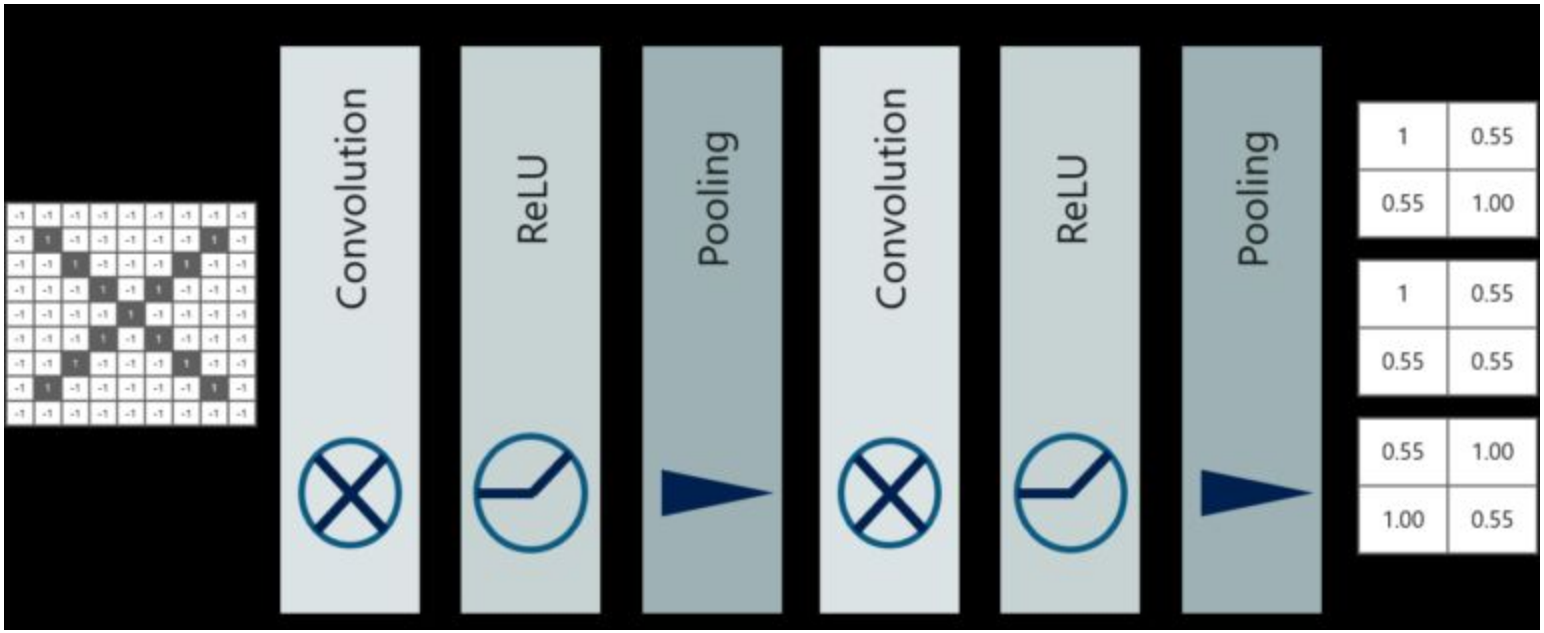
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

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

- But can we **further reduce** the image from **4×4** to **something lesser?**

- **Yes, we can!** We need to perform the 3 operations in an iteration after the first pass. So after the second pass we arrive at a 2×2 matrix as shown below:



- The last layers in the network are **fully connected,** meaning that neurons of preceding layers are **connected** to **every neuron** in **subsequent** layers.

- This **mimics high level reasoning** where all possible **pathways** from the **input** to **output** are considered.

- Also, fully connected layer is the final layer where the classification actually happens. Here we take our filtered and shrinked images and put them into one single list as shown below:

1	0.55
0.55	1.00

1	0.55
0.55	0.55

0.55	1.00
1.00	0.55

1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55

- So **next,** when we feed in, **‘X’** and **‘O’** there will be **some element** in the vector that will be **high.** Consider the image below, as you can see for ‘X’ there are **different elements** that are **high** and **similarly,** for **‘O’** we have **different elements** that are **high:**

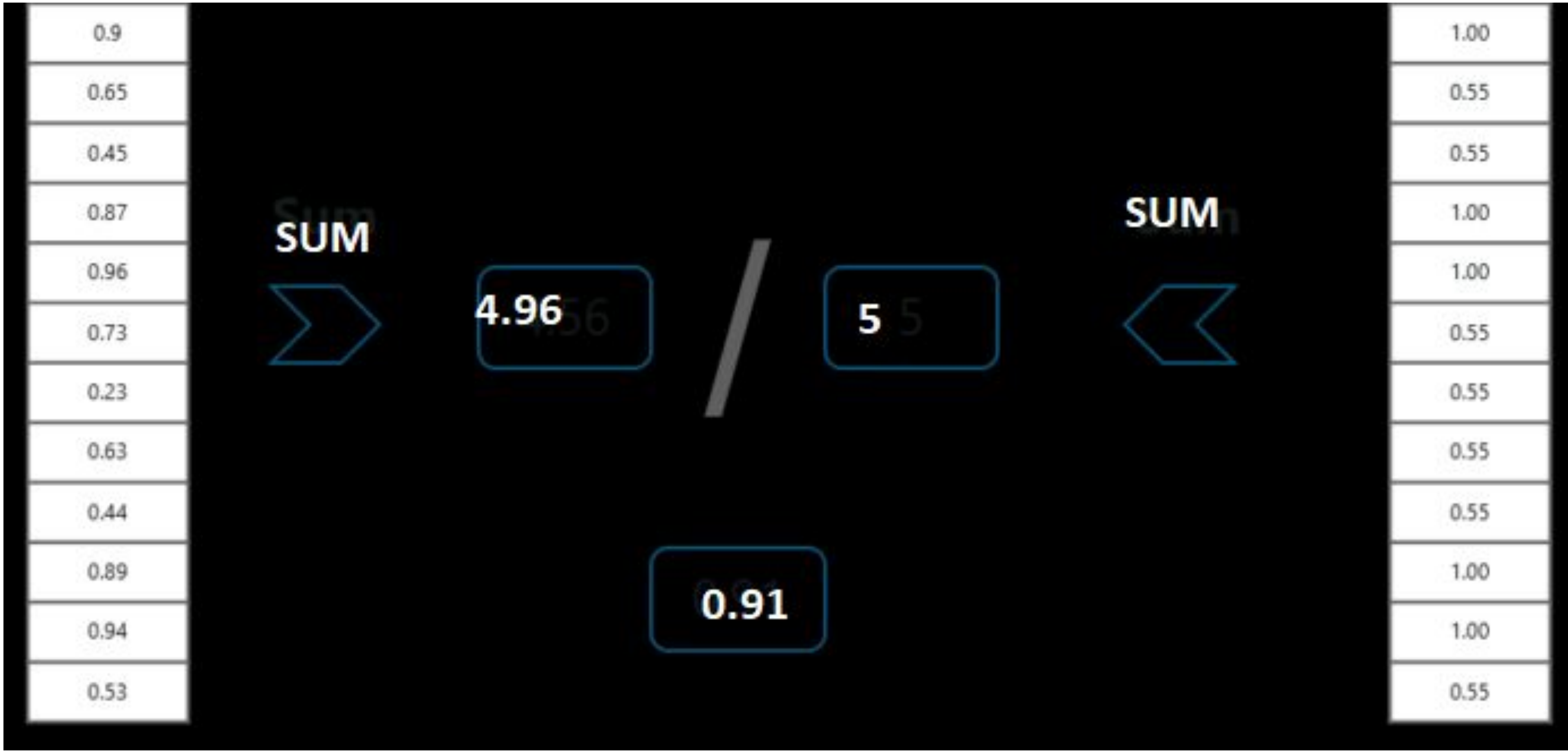
- When the **1st, 4th, 5th, 10th** and **11th** values are **high,** we can classify the image as **‘x’.** The concept is similar for the other **alphabets** as well – when certain **values** are arranged the way they are, they can be **mapped** to an **actual** letter or a **number** which we **require,** simple right?

Prediction Of Image Using Convolutional Neural Networks – Fully Connected Layer

- At this point in time, **we're done training** the network and we can begin to predict and **check** the **working** of the **classifier**. Let's check out a simple example:

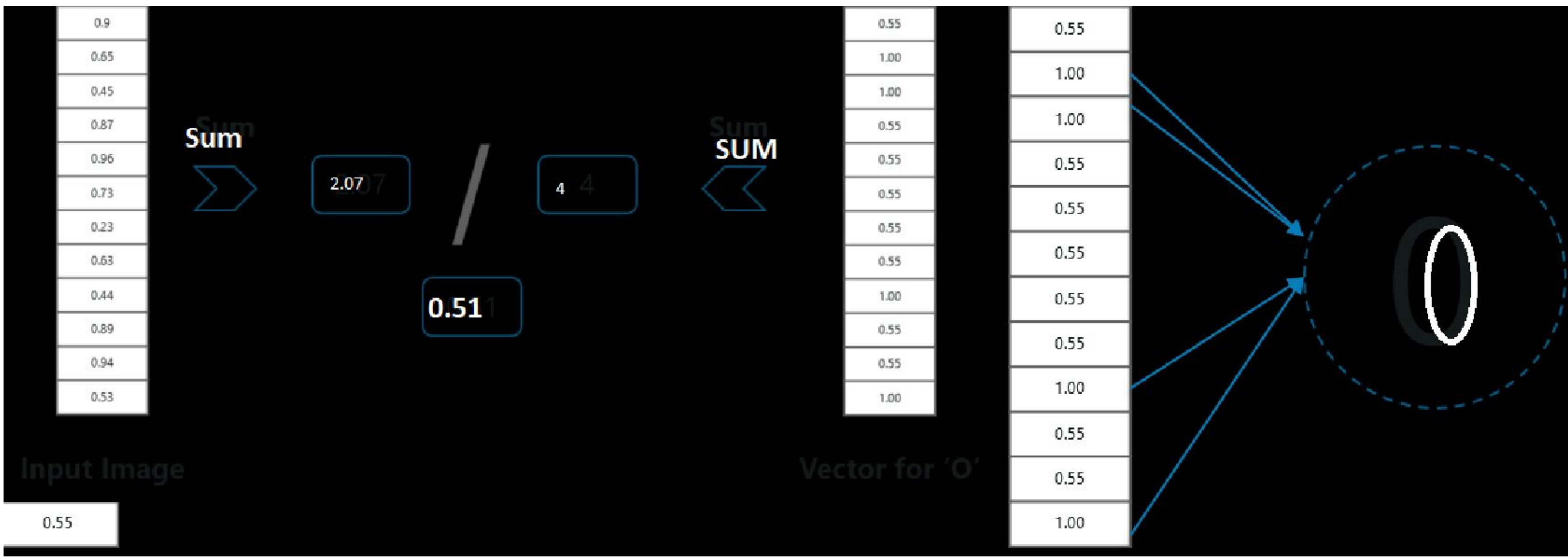
0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53

- In the above image, we have a **12 element** vector obtained after **passing** the **input** of a **random letter** through all the **layers** of our **network.**
- But, **how** do we check to know what **we've obtained** is right or wrong?
- We **make predictions** based on the **output** data by comparing the **obtained values** with list of 'x'and 'o'!


0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53
SUM
4.96
/
5
SUM
1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55
0.91

- Well, it is **really easy.** We just **added** the values we which found out as high (1st, 4th, 5th, 10th and 11th) from the **vector table** of **X** and we got the sum to be **5.** We did the **exact same thing** with the **input image** and got a value of **4.56**.

- When we **divide** the **value** we have a **probability match** to be **0.91!** Let’s do the **same** with the **vector table** of **‘o’** now:



- We have the output as 0.51 with this table. Well, probability being 0.51 is less than 0.91, isn’t it?

- So we can conclude that the resulting input image is an ‘x’!