WEEK 09 - DEEP LEARNING II

CONVOLUTIONAL NEURAL NETWORKS

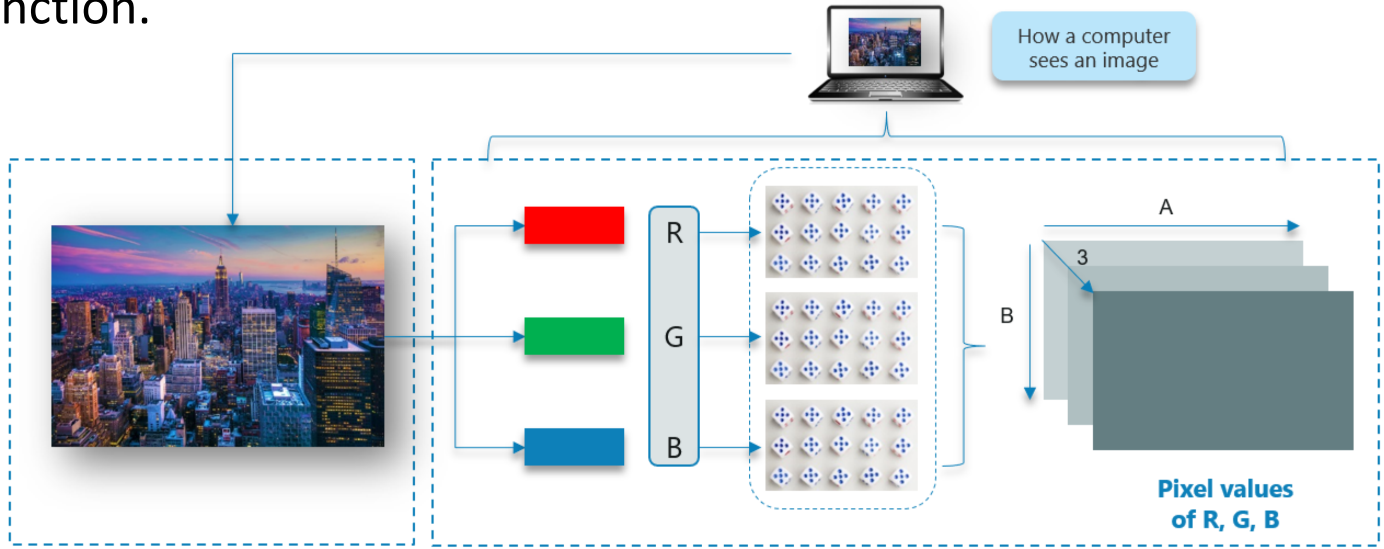
Thakshila Dasun
BSc. Hons in Mechanical Engineering
(Mechatronics Specialization)
CIMA, UK
Academy of Innovative Education

CONVOLUTIONAL NEURAL NETWORKS (CNN)

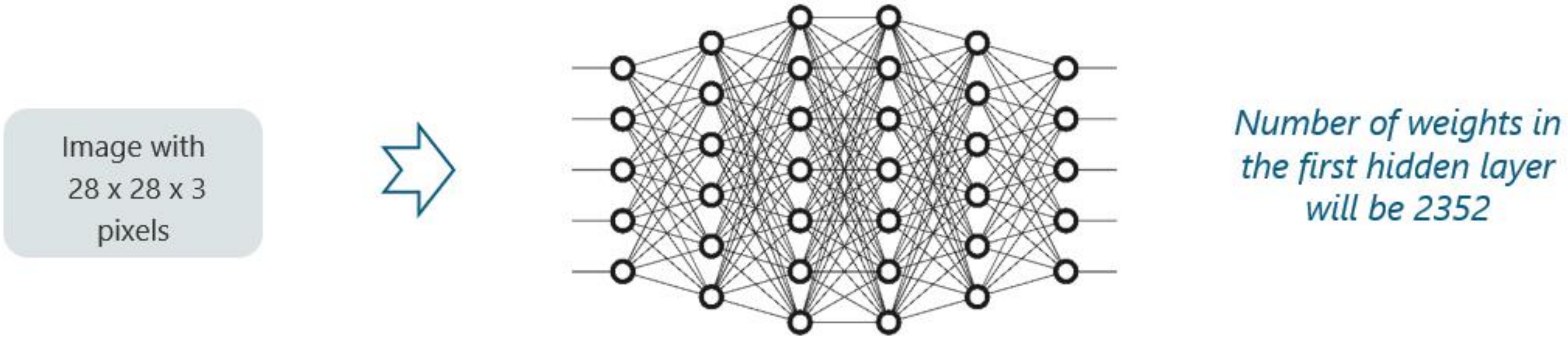
"Convolutional Neural Networks are designed to address image recognition systems and classification problems. Convolutional Neural Networks have wide applications in image and video recognition, recommendation systems and natural language processing"

Images in Computers

- The image is broken down into 3 color-channels which is Red, Green and Blue. Each of these color channels are mapped to the image's pixel.
- Idea of neural networks began unsurprisingly as a model of how neurons in the brain function.

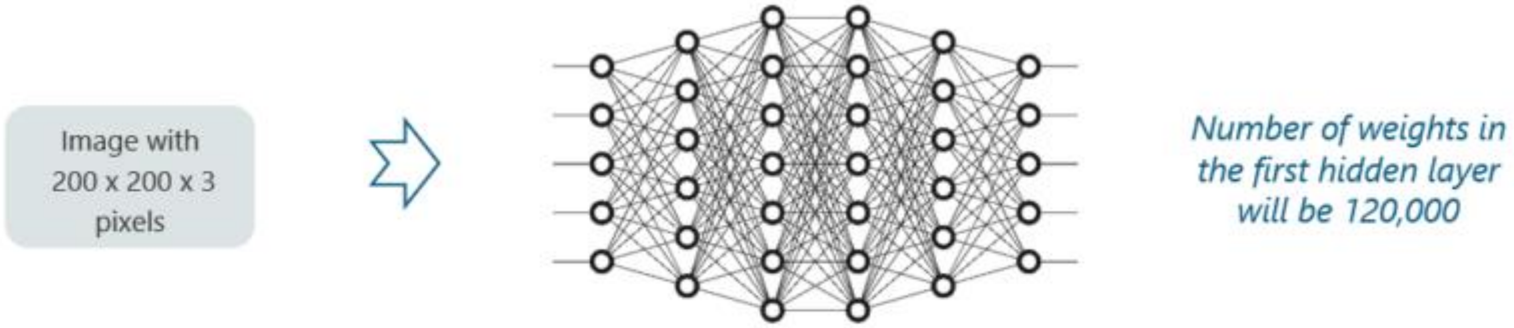


Why Not Fully Connected FFNNs?



- **Consider 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.

Why Not Fully Connected FFNNs?



- Any **generic** input **image** will **at least** 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.**

Convolutional Neural Networks (1)

- Convolutional Neural Networks, like FFNNs, 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.**

Convolutional Neural Networks (2)

- 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.

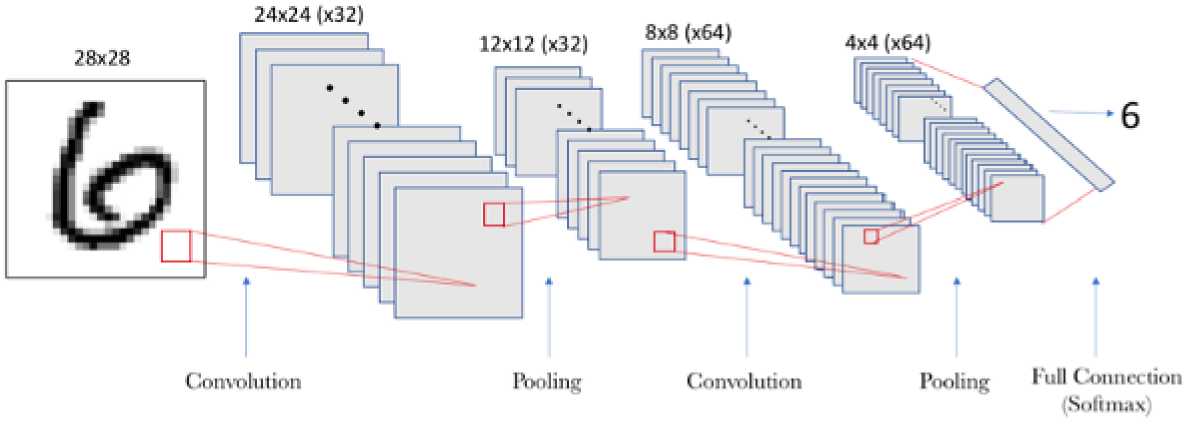
Origin Of Convolutional Neural Networks

- 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 **unlabeled,** 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".**

How Do Convolutional Neural Networks Work?

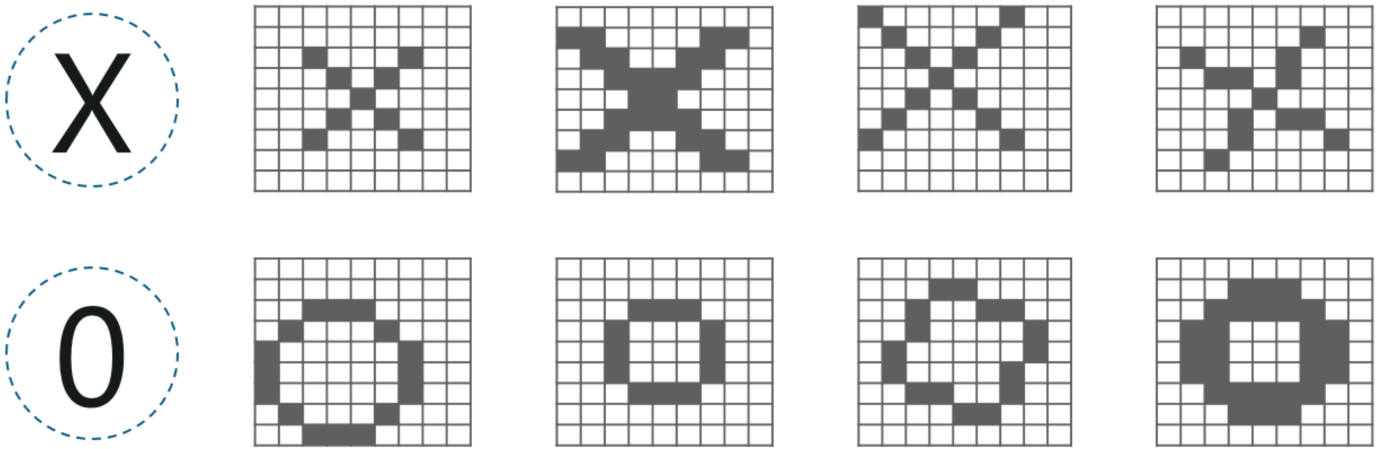
There are **four** layered **concepts** we should understand in Convolutional Neural Networks:

1. Convolution
2. ReLu
3. Pooling
4. Dense



How Do Convolutional Neural Networks Work?

- 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.**



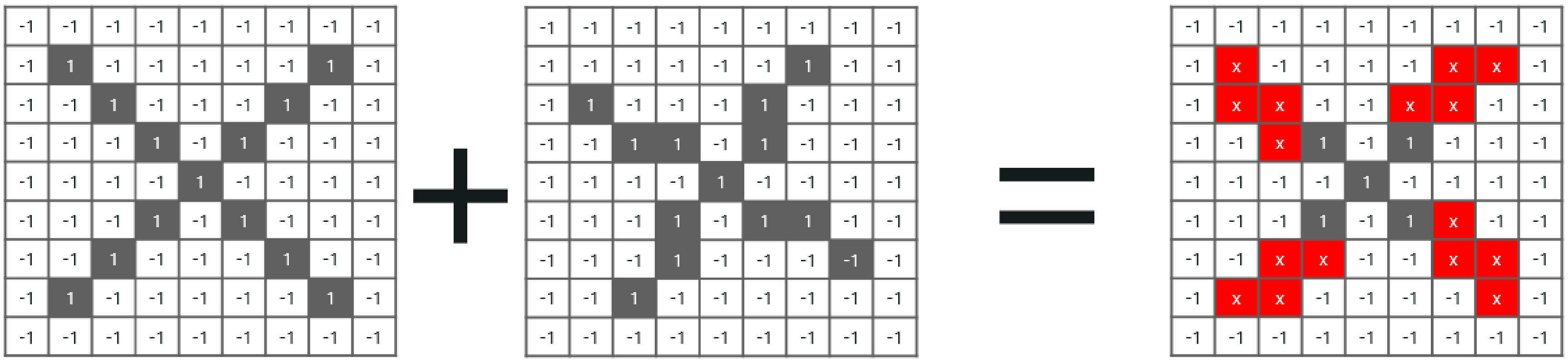
How Do Convolutional Neural Networks Work?

- 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.

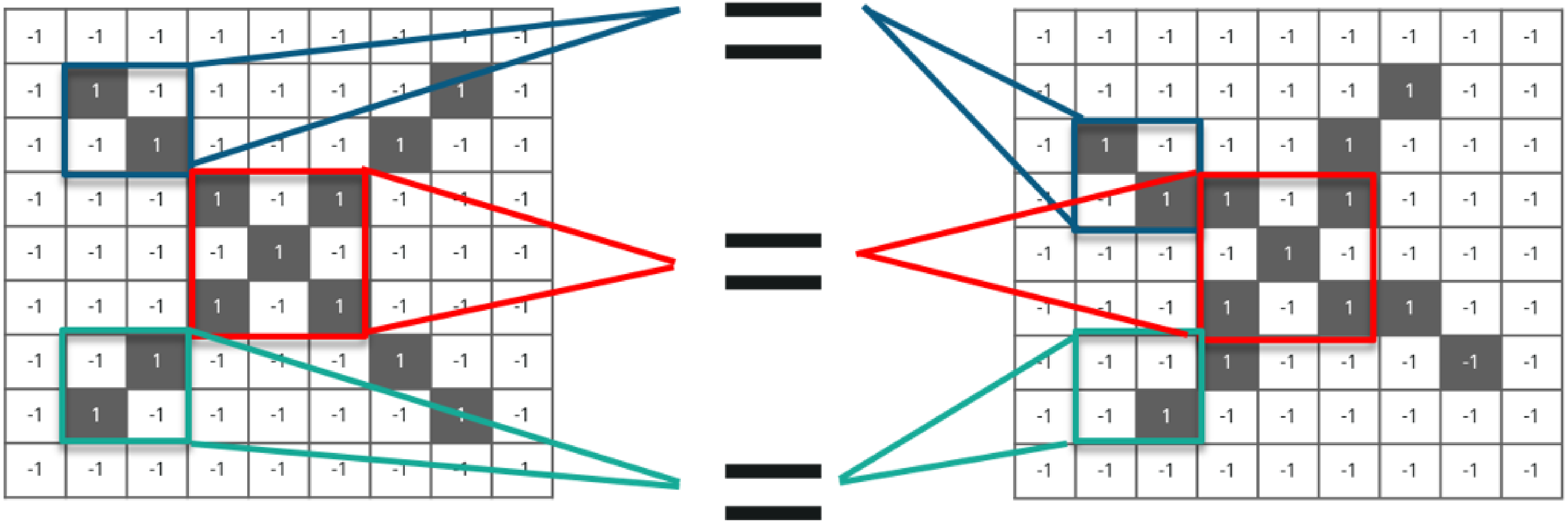
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

How Do Convolutional Neural Networks Work?

- 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.**

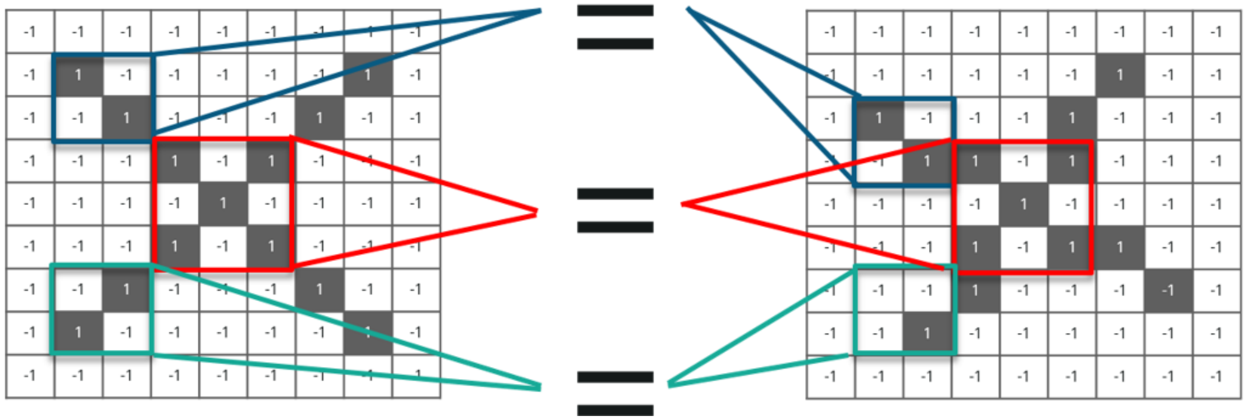


So, how do we fix this?



Attention! This is Tricky.

- 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.**

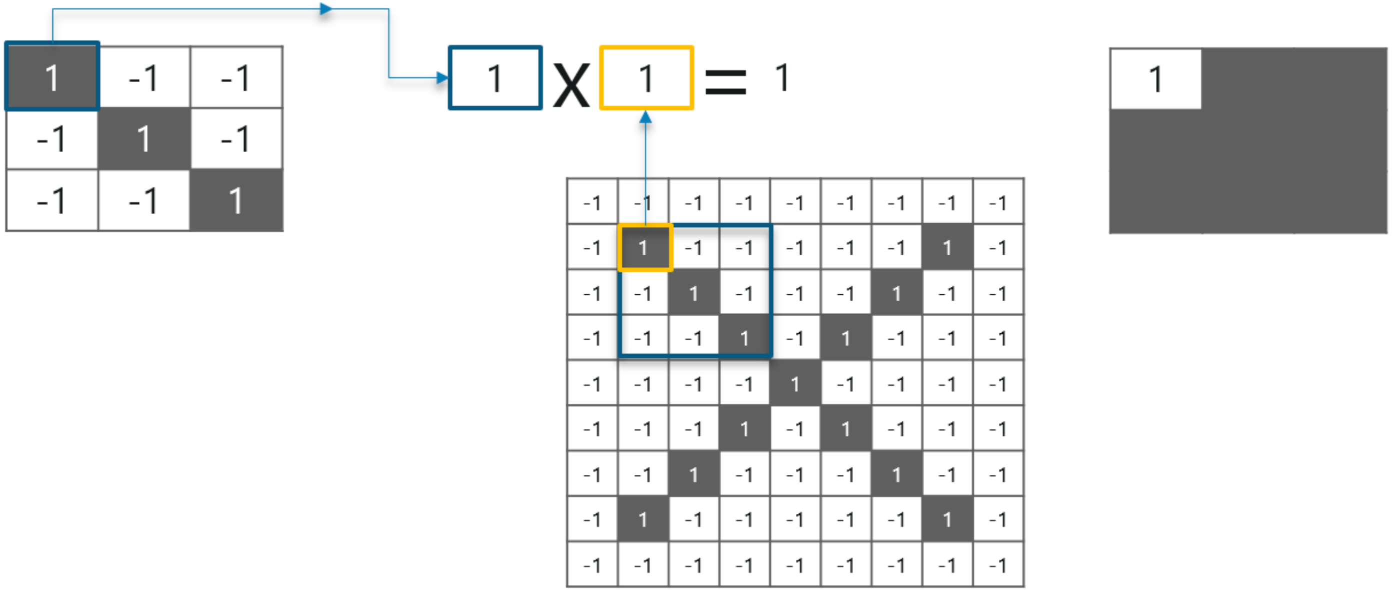


Convolution Of An Image

- 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:
  1. **Line up** the feature and the image
  2. **Multiply** each **image** pixel by corresponding **feature** pixel
  3. **Add** the values and find the **sum**
  4. **Divide** the sum by the **total** number of pixels in the **feature**

Convolution Of An Image



- Consider the above image – As you can see, 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**.

Convolution Of An 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 **last 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:

Convolution Of An Image

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

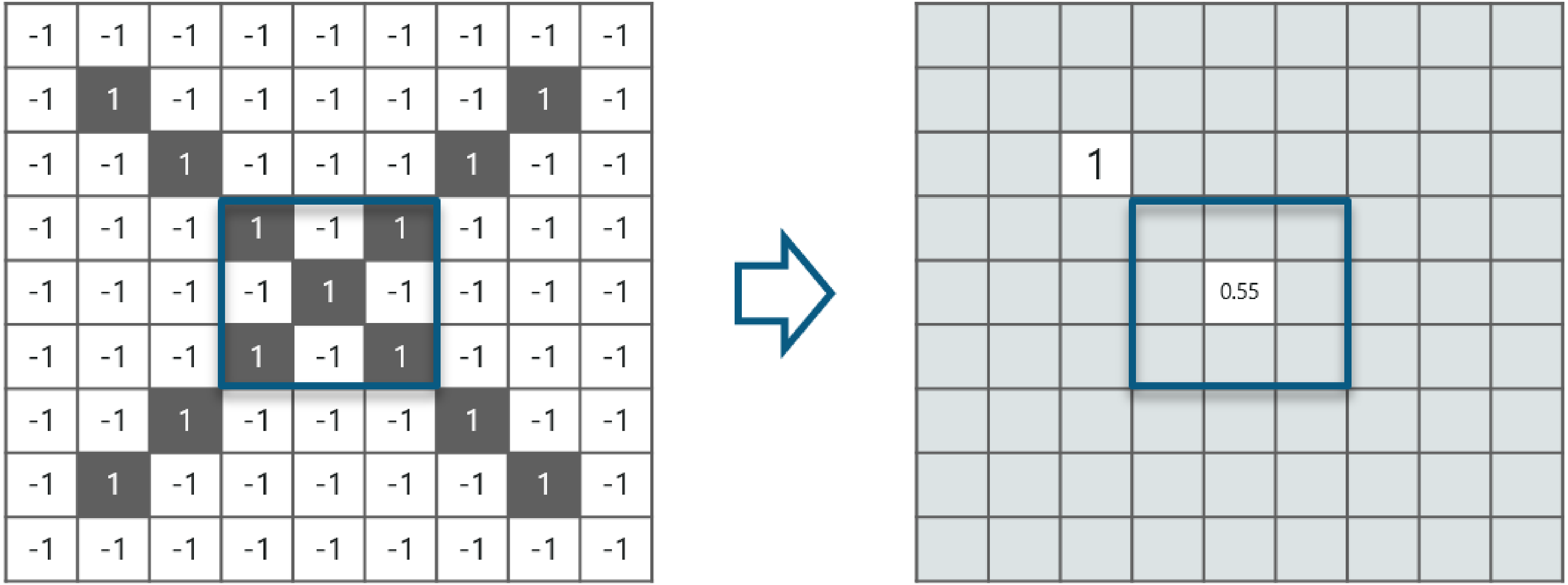
$$\frac{1+1-1+1+1+1-1+1+1}{9} = .55$$

-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

- 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:**

Convolution Of An Image



- Similarly, we move the feature to every other position in the image and see how the feature matches that area. So after doing this, we will get the output as:

Convolution Of An Image

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

Convolution Of An 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 **last 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:

Convolution Of An Image

- 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.**

Let’s consider an Example: Sobel Gradients

- The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.
- In theory at least, the operator consists of a pair of 3×3 convolution kernels as shown in Figure 1. One kernel is simply the other rotated by 90°.

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

Gx

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

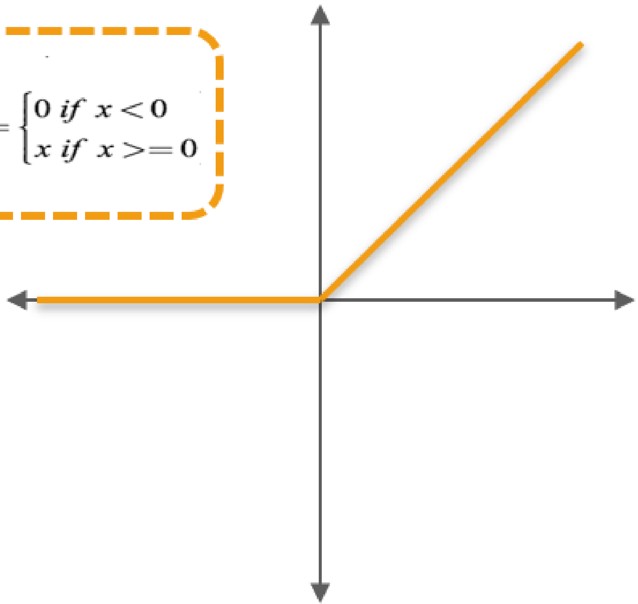
Gy

Convolution Of An Image

- 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

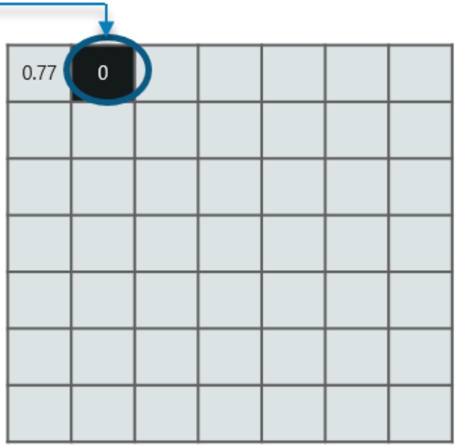
- **Rectified Linear Unit** (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.



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

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:

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

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

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

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

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	-0.55	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	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	-0.11	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

0.33	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

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:

1. Pick a **window size** (usually 2 or 3)
2. Pick a **stride** (usually 2)
3. **Walk** your window **across** your **filtered** images
4. From each **window,** take the **maximum** value

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			

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	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

- 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.

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	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

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

0.55	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

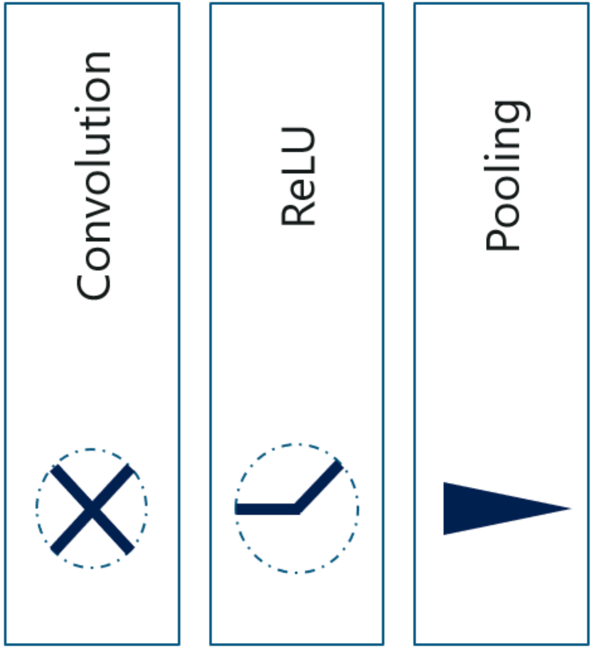
0.33	0	0.55	0.33	0.11	0	0.77
0	0.11	0	0.33	0	1.00	0
0.55	0	0.11	0	1.00	0	0.11
0.33	0.33	0	0.55	0	0.33	0.33
0.11	0	1.00	0	0.11	0	0.55
0	1.00	0	0.33	0	0.11	0
0.77	0	0.11	0.33	0.55	0	0.33

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

- 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 above result:

Stacking Up The Layers

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

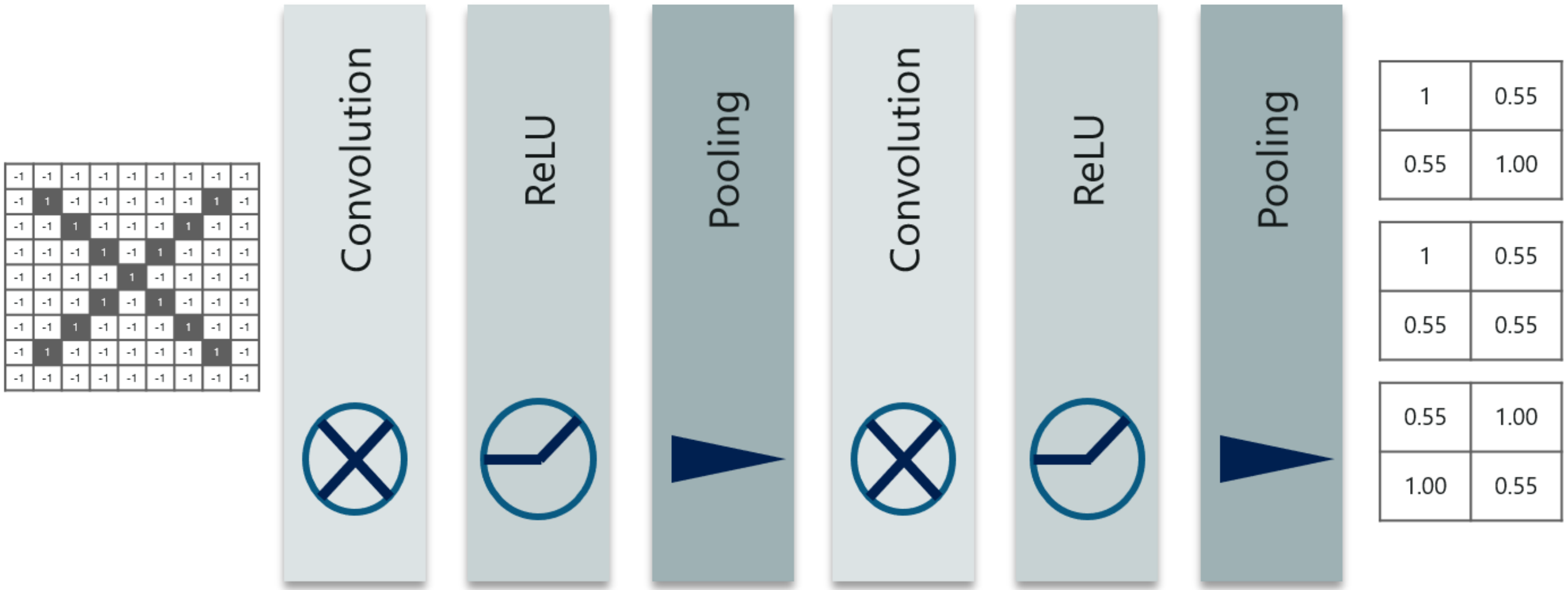
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

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

- 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

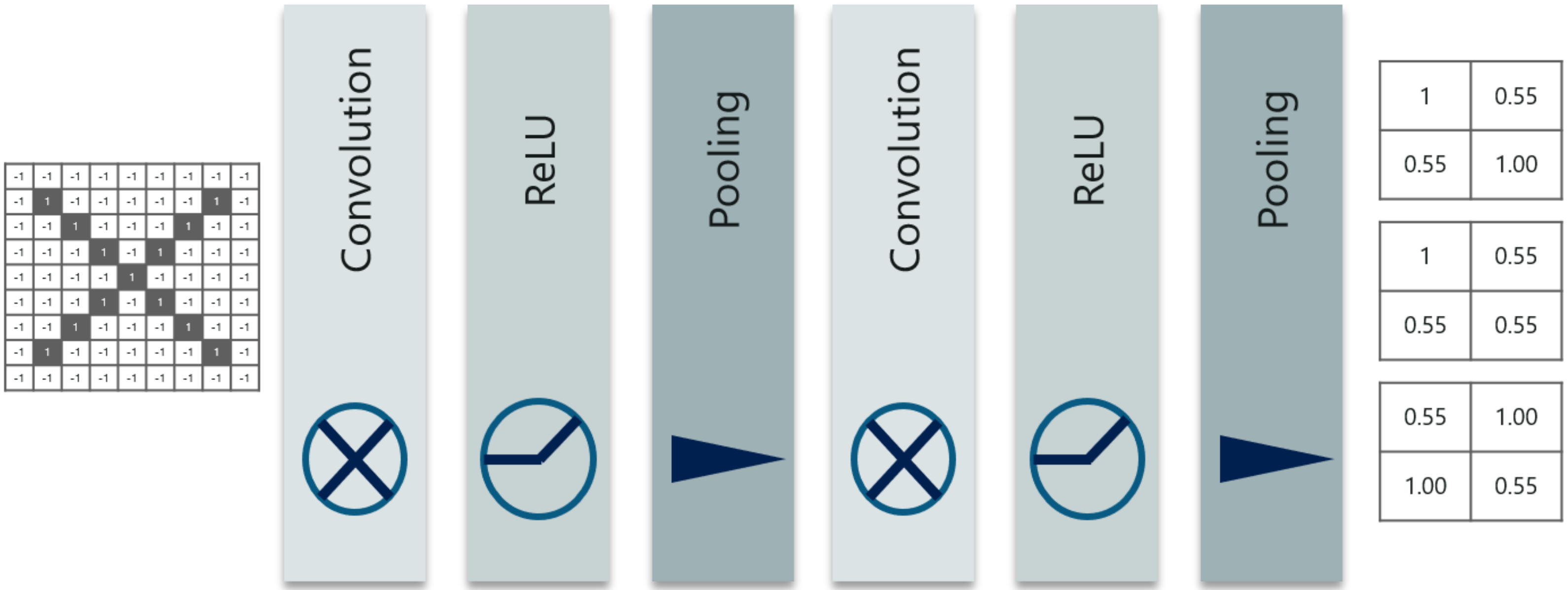
– **Convolution, ReLU** and **Pooling** as shown above:

Stacking Up The Layers



- 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:

Stacking Up The Layers



- 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:

Dense Layer- Last Layer

- 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

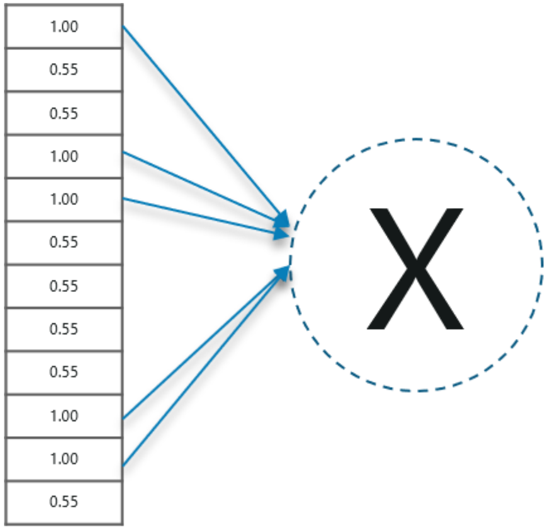
0.55	1.00
1.00	0.55

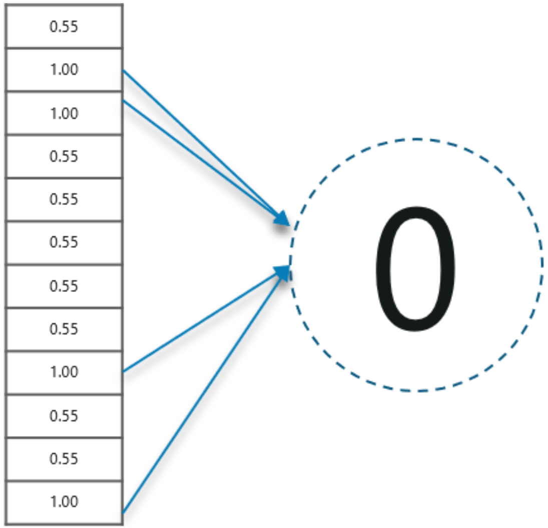
1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55

Dense Layer- Last Layer

- 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:**

Dense Layer- Last Layer



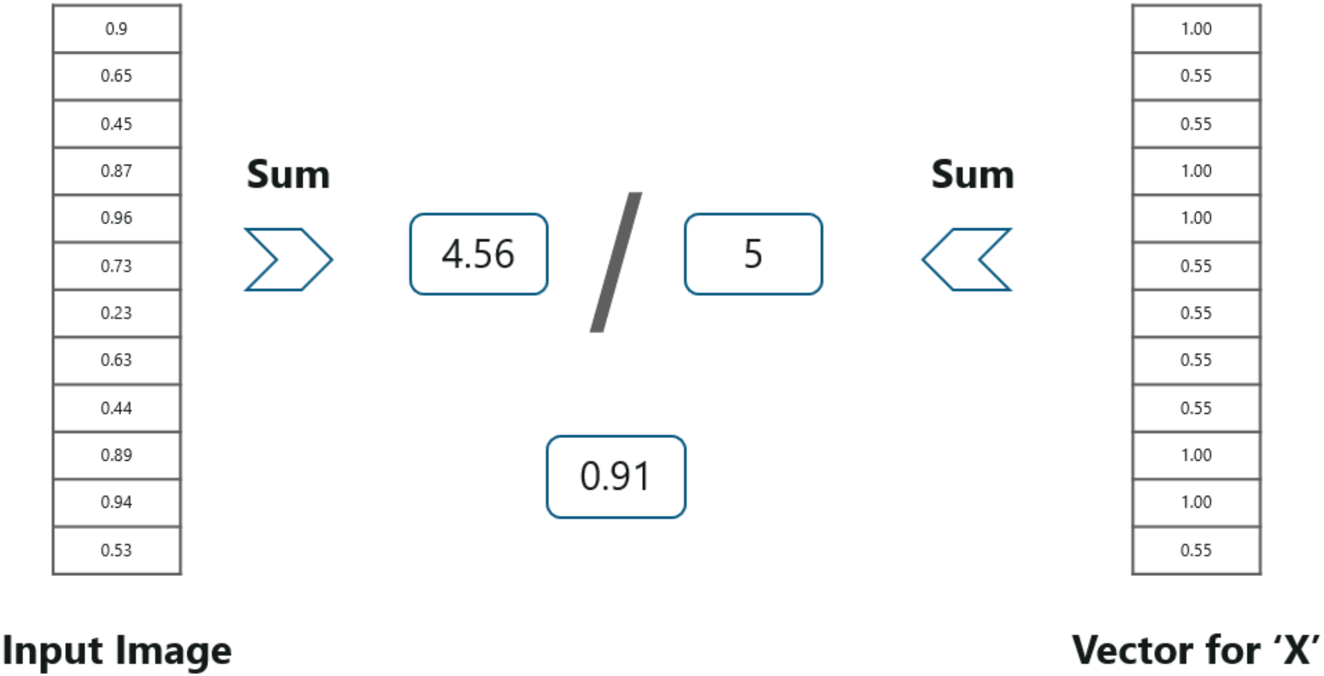


- When the **1st, 4th, 5th, 10th** and **11th** values are **high,** we can classify the image as **'x'.**
- The concept is similar for the other **alphabets** as well – when certain **values** are arranged the way they are, they can be **mapped** to an **actual** letter or a **number** which we **require.**

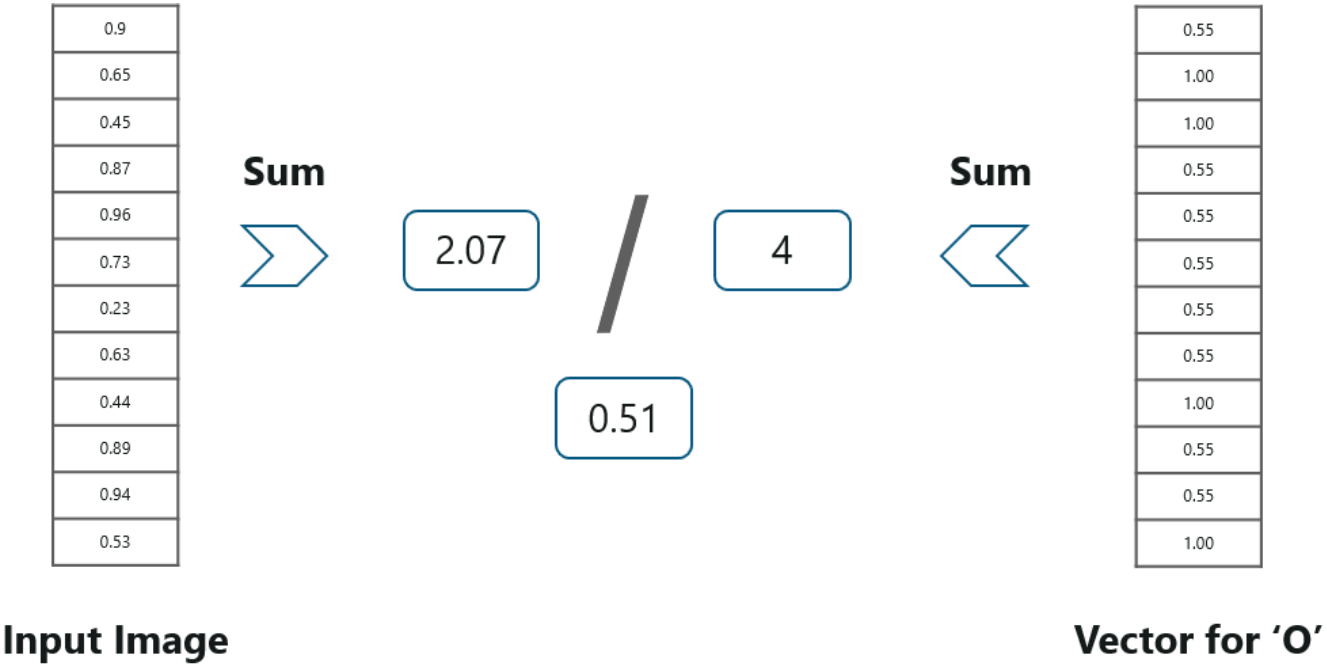
Prediction Of Image Using Convolutional Neural Networks – Fully Connected Layer

- 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.**
- 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



- 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:



- 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’!**