Deep learning project

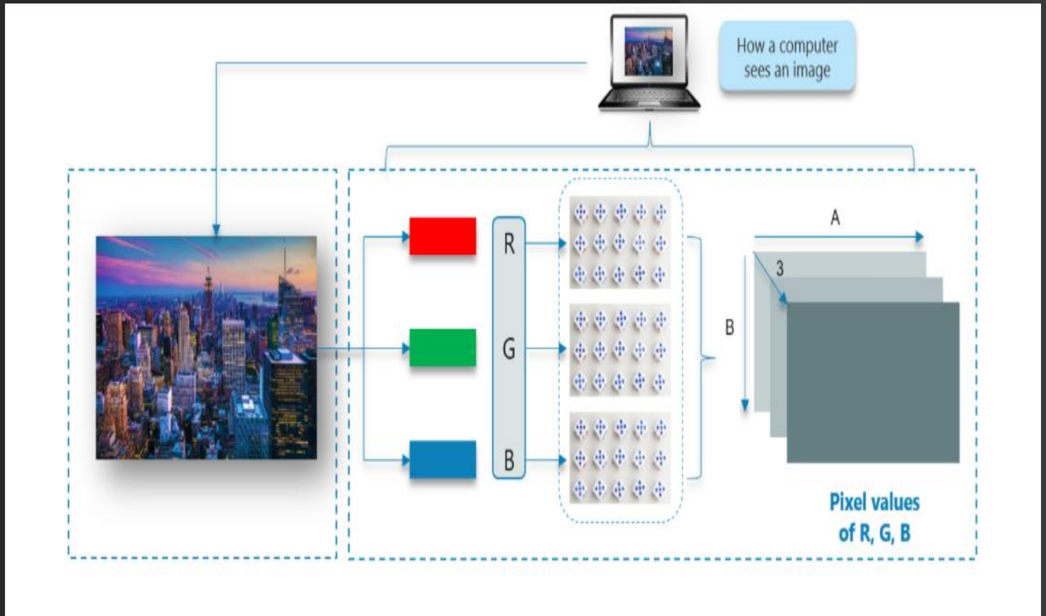
Fashion MNIST Classification

Agenda

1. **Concepts of Convolutional Neural Network?**
2. **Problem statement**
3. **Import Libraries**
4. **Load Data**
5. **Show Image from Numbers**
6. **Change Dimension / Feature Scaling**
7. **Build First Convolutional Neural Network**
8. **Train Model**
9. **Test & Evaluate Model**
10. **Confusion Matrix**
11. **Classification Report**
12. **Save Mode**
13. **Build 2 Complex CNN**

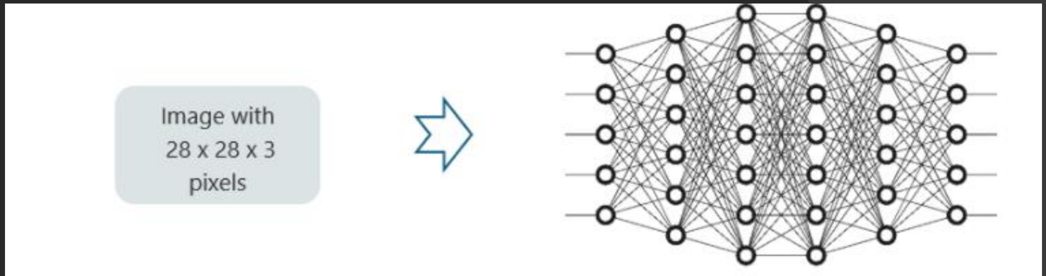
How does a computer read an image ?

- The image is **broken down** into 3 color-channels which is **Red, Green** and **Blue.** Each of these color channels are **mapped** to the **image's pixel.**
- Then, the **computer recognizes** the value associated with **each pixel** and **determine** the **size** of the image.
- For **black-white** images, there is only **one channel** and the **concept** is the **same.**



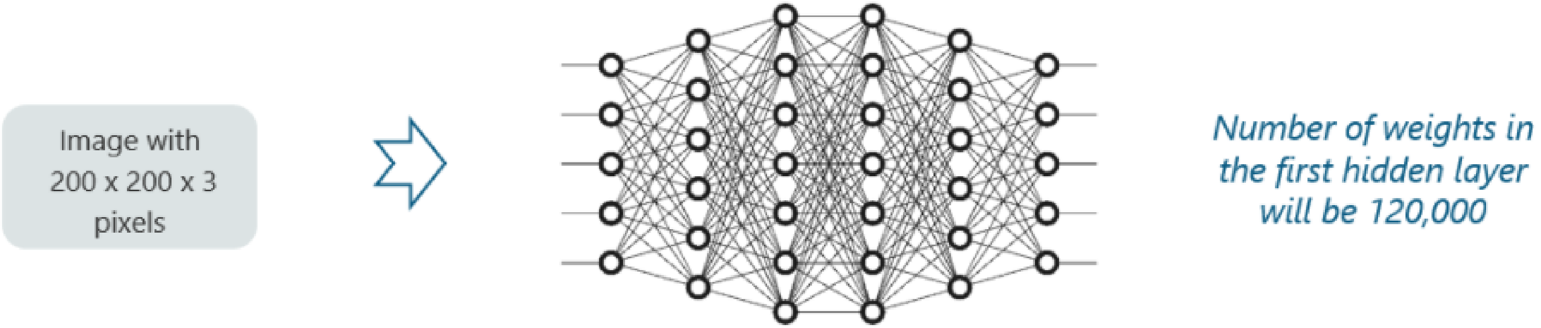
Why not fully connected Network

- We **cannot** make use of fully connected networks when it comes to **Convolutional Neural Networks,** here’s why!
- Consider the given image:
- Here, wehave **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.



Why not fully connected network ?

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 CNN ?

- 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.

CNN Example

- 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.

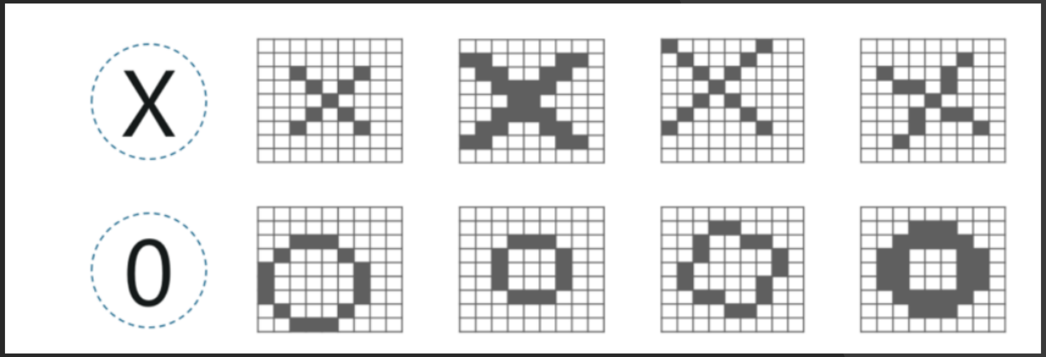
How CNN works ?

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

1. Convolution,
2. ReLu,
3. Pooling and
4. Full Connectedness (Fully Connected Layer).

Example of CNN:

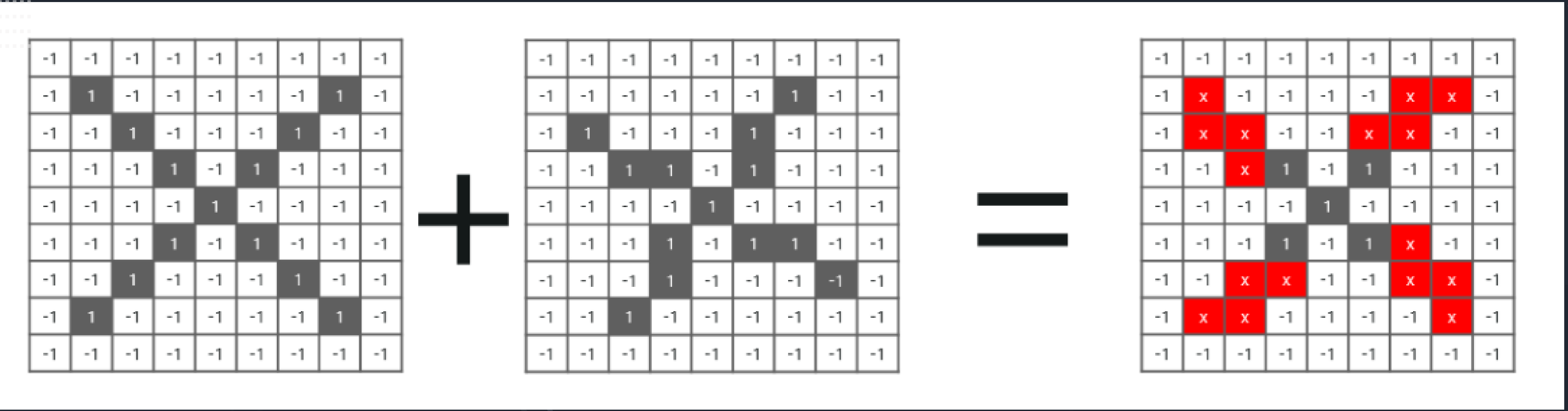
- There are multiple renditions of X and O’s. This makes it tricky for the computer to recognize. But the goal is that if the **input signal** looks like **previous** images it has seen before, the **“image” reference** signal will be mixed into, or **convolved** with, the **input** signal. The resulting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



Example of CNN :

- 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.

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

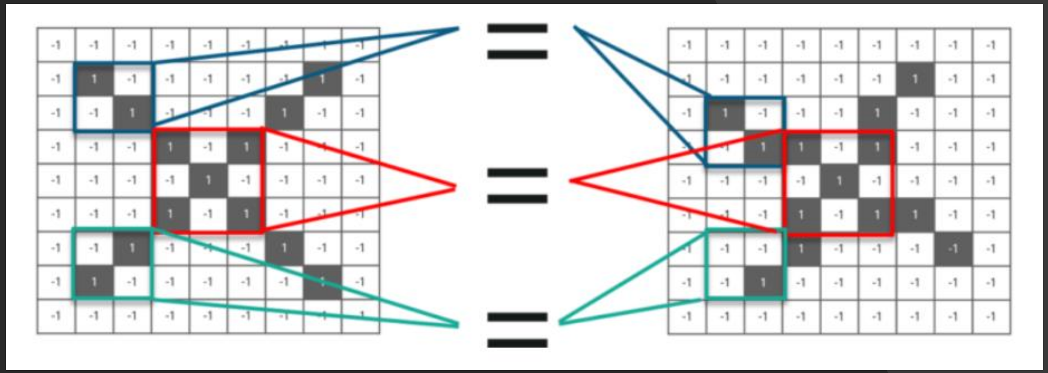


Example of CNN

- 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.**

How do we fix the previous issue ?

- 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).

4 steps for convolution

1

Line up the feature and the image

2

Multiply each **image** pixel by corresponding **feature** pixel

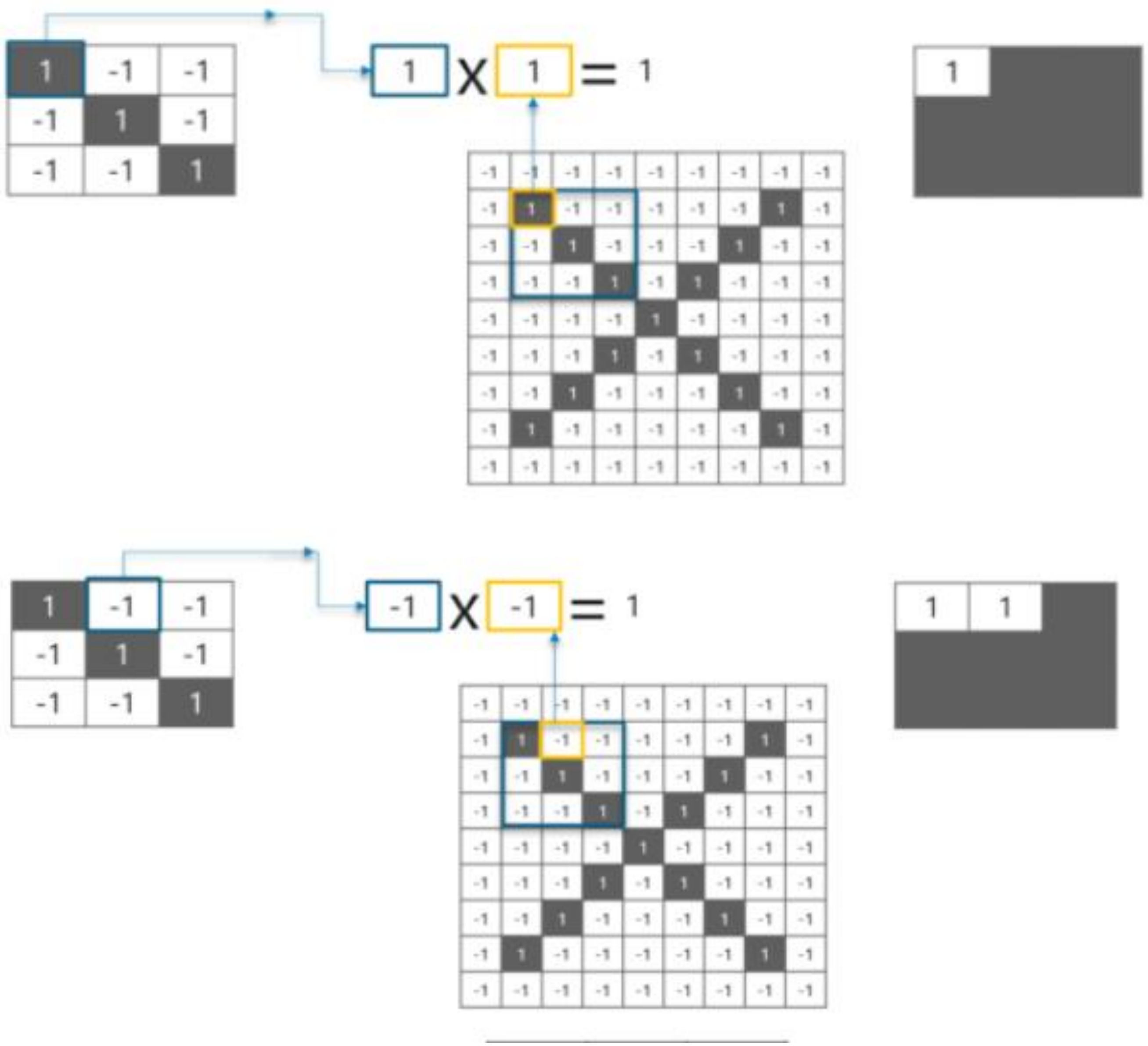
3

Add the values and find the **sum**

4

Divide the sum by the **total** number of pixels in the **feature**

Step 1 and 2



Step 3 and 4

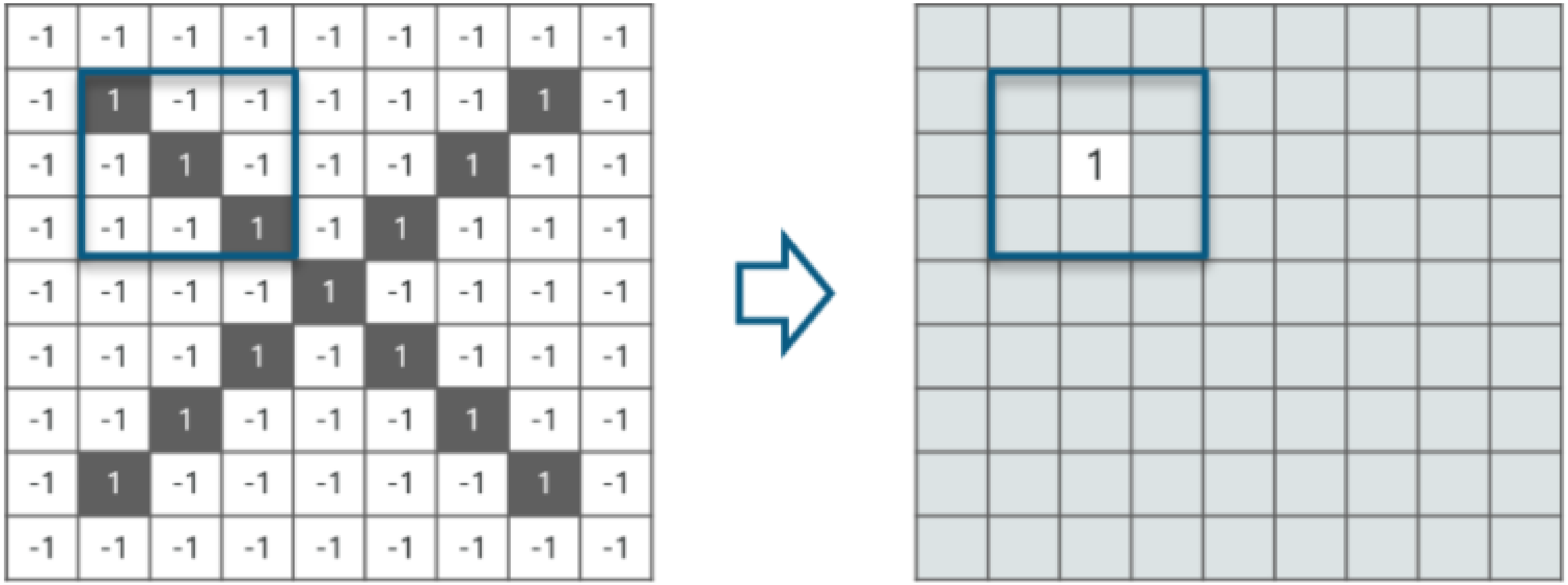
1	-1	-1
-1	1	-1
-1	-1	1

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

-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

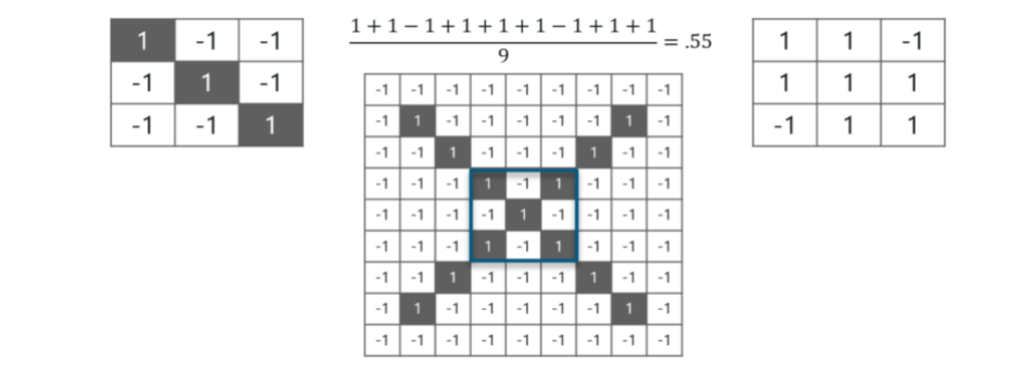
1	1	1
1	1	1
1	1	1

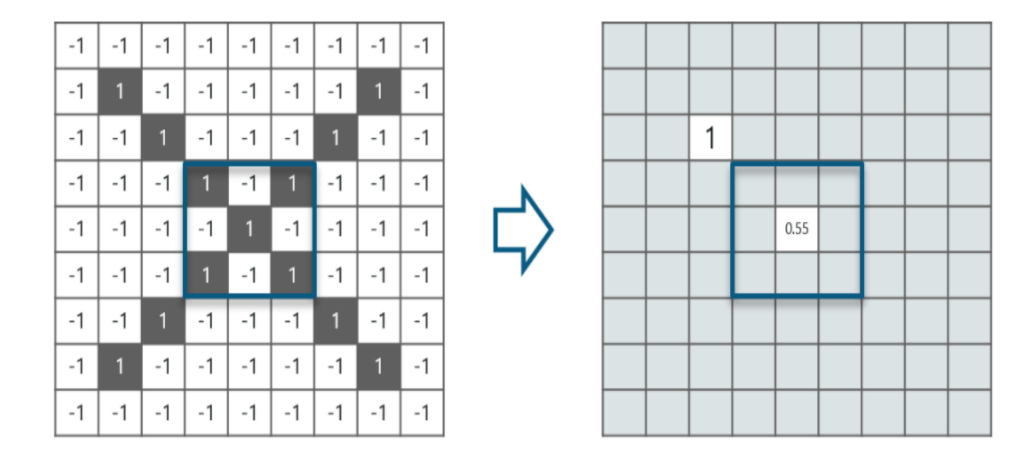
Put the value in center



One more example

- 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:As you can see ,here after performing first four steps we have value of 0.55**





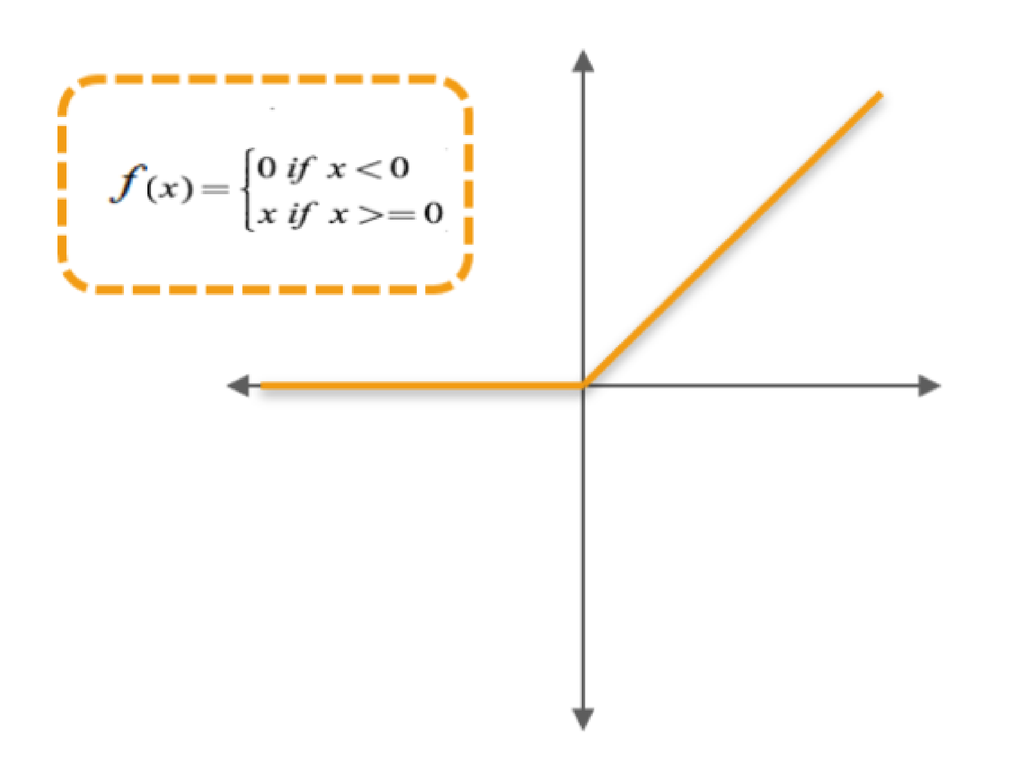
Next step

- 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:
- 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.**

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

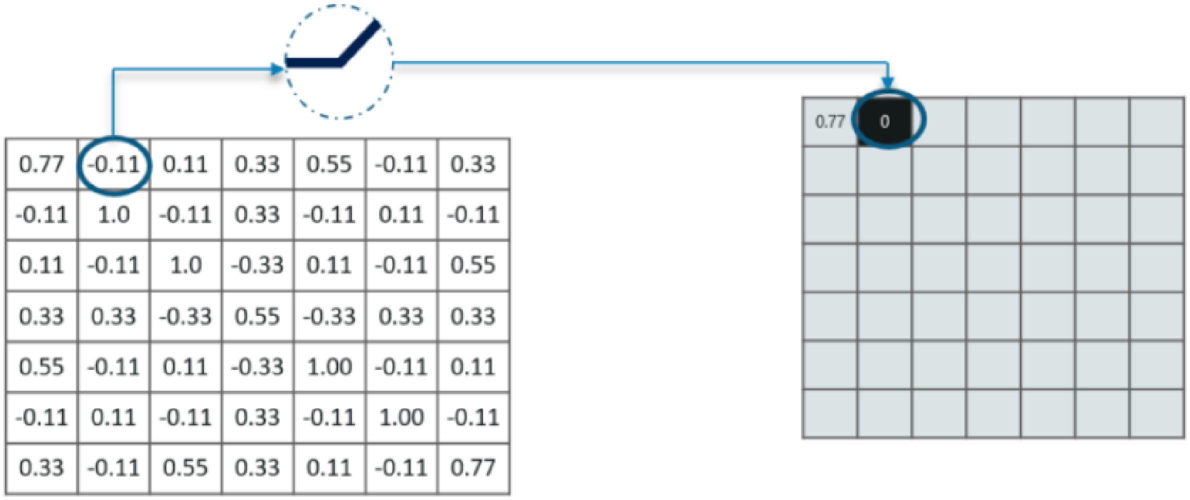
Relu Layer

- 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.



Why do we require Relu here ?

- 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 in the image :



Relu layer

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

Relu layer

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

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

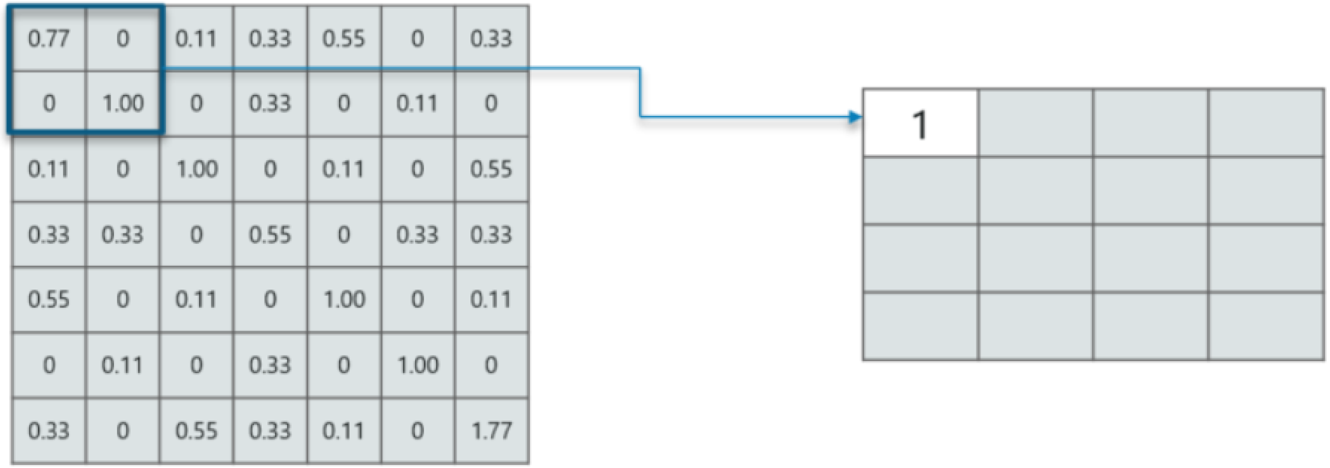
Pooling layer

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

Pooling layer 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.**



Pooling layer example

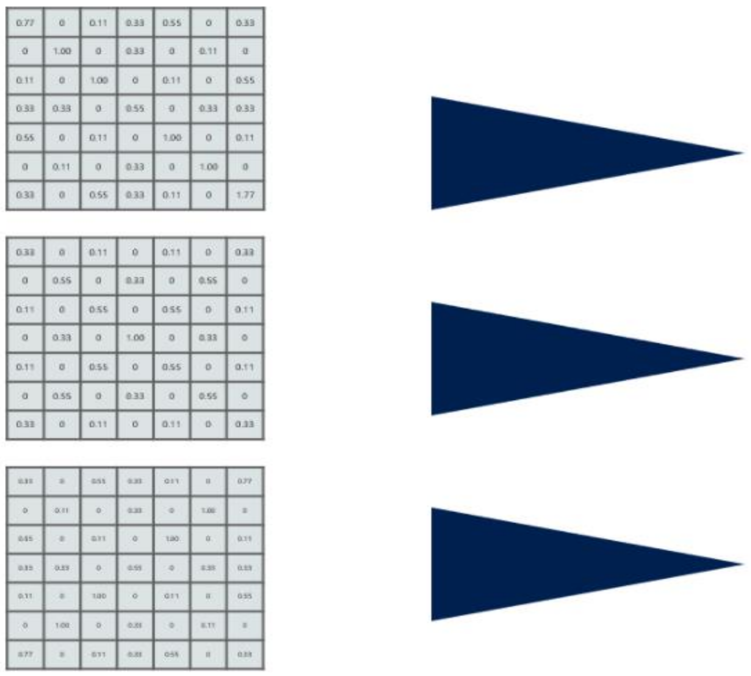
- But we need to **move** the **window across** the **entire** image. The procedure is exactly as same as above and we need to repeat that for the entire image.

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

Pooling layer example

- 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:



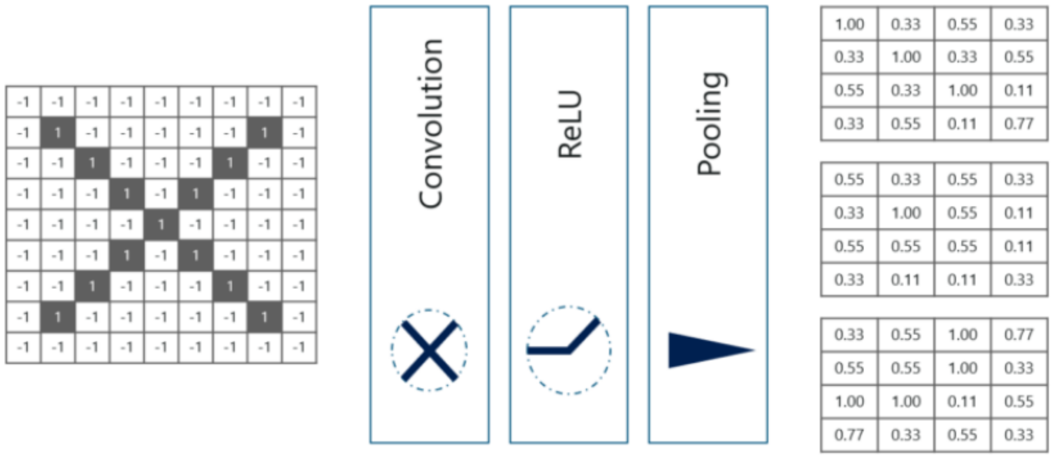
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

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

Stacking up all 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 in the image:



Iteration of the same operation

- So after the second pass we arrive at a 2×2 matrix as shown:

Next step

- 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 :

1	0.55
0.55	1.00

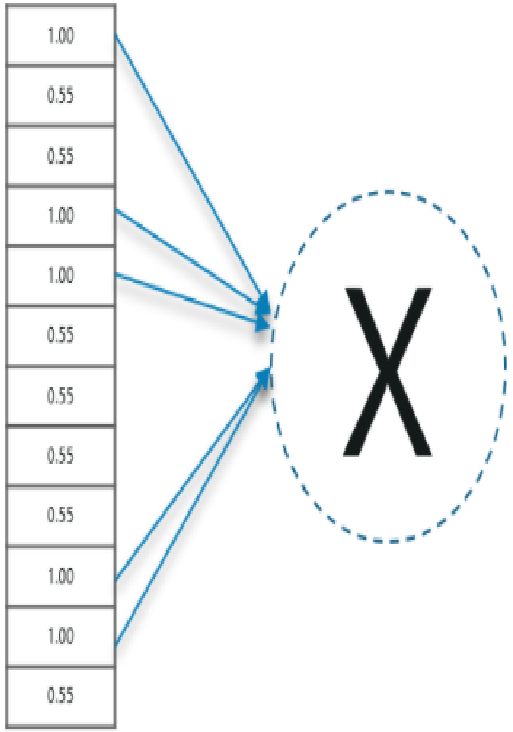
1	0.55
0.55	0.55

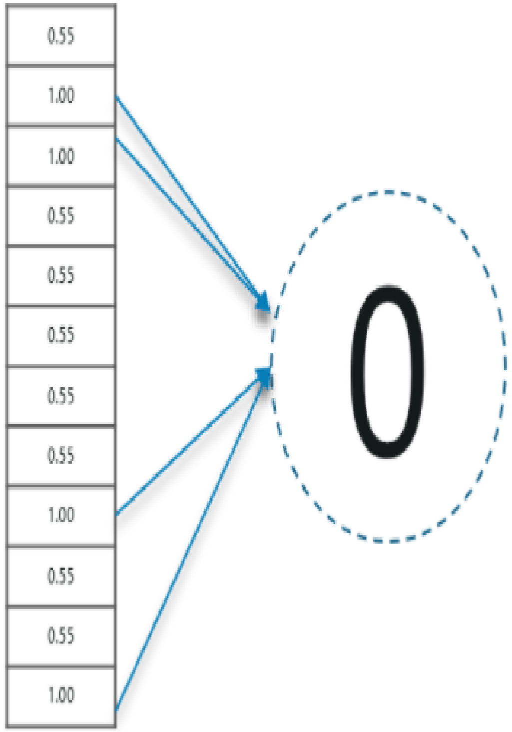
0.55	1.00
1.00	0.55

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

Next step

- 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**





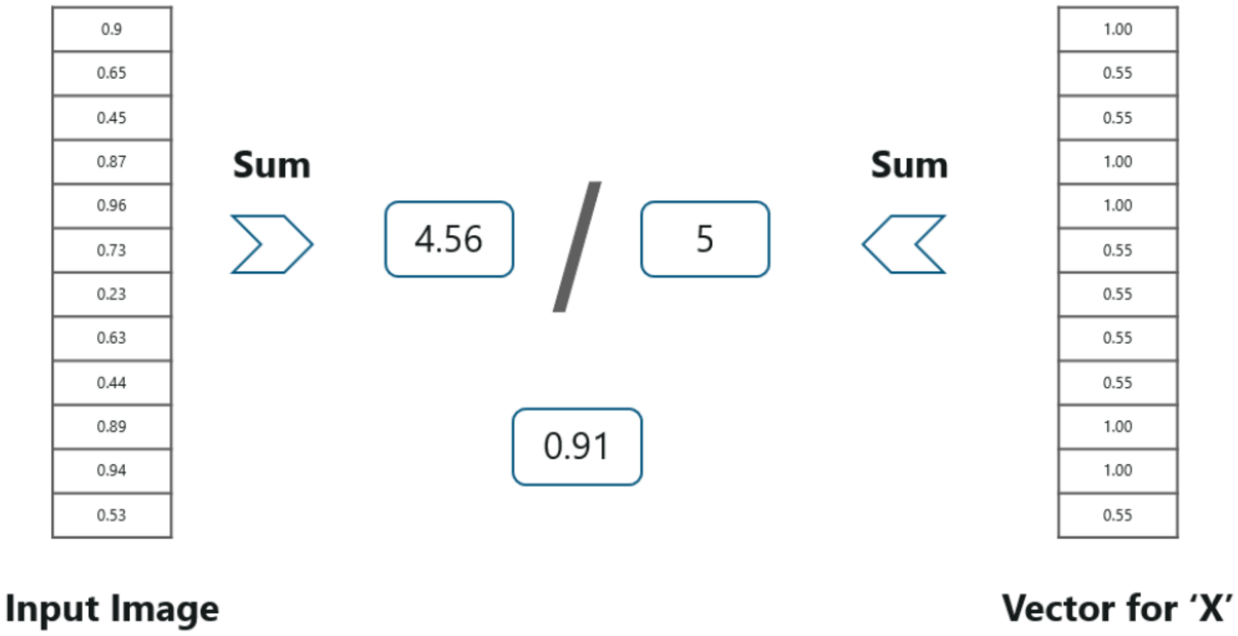
Testing of CNN

- 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

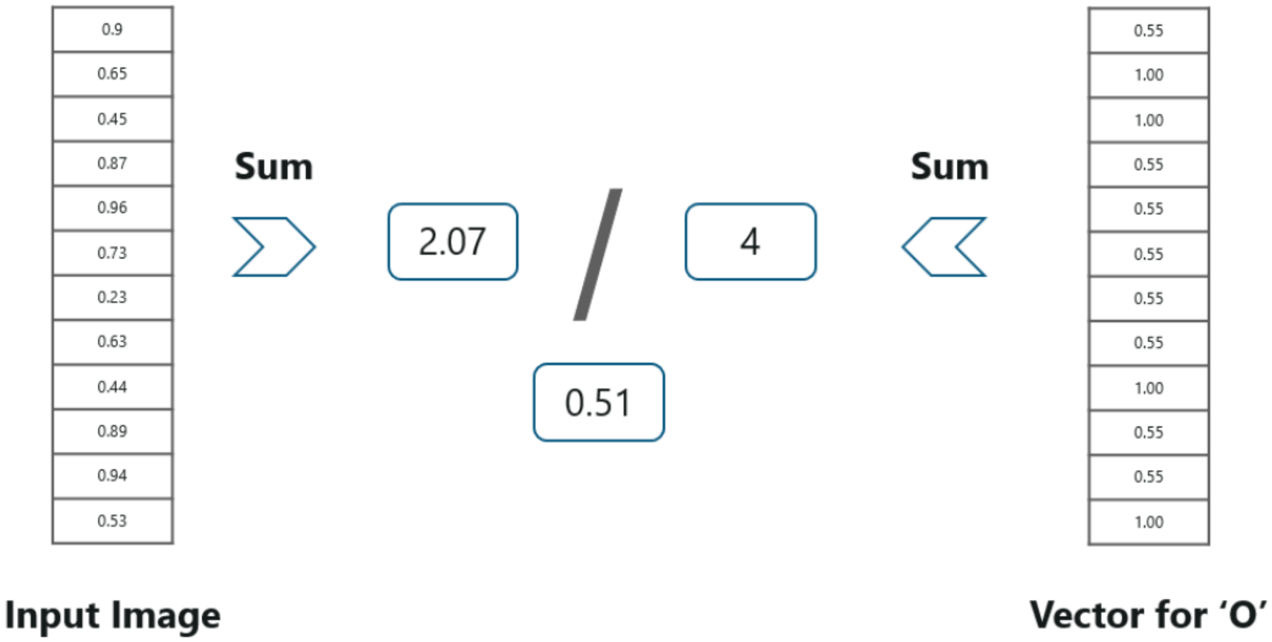
Testing of CNN

- 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!**



Testing of CNN

- 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'!**



Problem statement

The objective is to identify (predict) different fashion products from the given images using Convolutional neural network.

The 'target' dataset has 10 class labels, as we can see from above (0 – T-shirt/top, 1 – Trouser,,....9 – Ankle Boot).

Given the images of the articles, we need to classify them into one of these classes, hence, it is essentially a **'Multi-class Classification'** problem.

Understanding the Dataset

- Fashion MNIST Training dataset consists of 60,000 images and each image has 784 features (i.e. 28×28 pixels). Each pixel is a value from 0 to 255, describing the pixel intensity. 0 for white and 255 for black.

The class labels for Fashion MNIST are:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Data Preprocessing

1
Import Libraries

2
Load Data

3
Show Image from Numbers

4
Change Dimension / Feature Scaling

Build CNN

Model building by
convolutional neural network.

Test and evaluate model

Test & Evaluate Model

Confusion Matrix

Classification Report

Build 2 complex CNN

Including more hidden layers and test it.

The End

Thank you