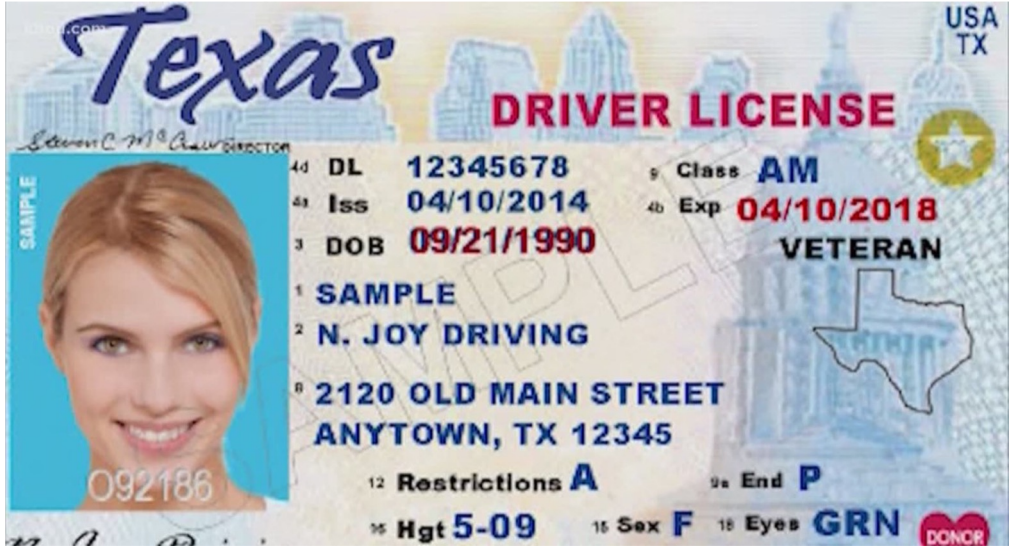
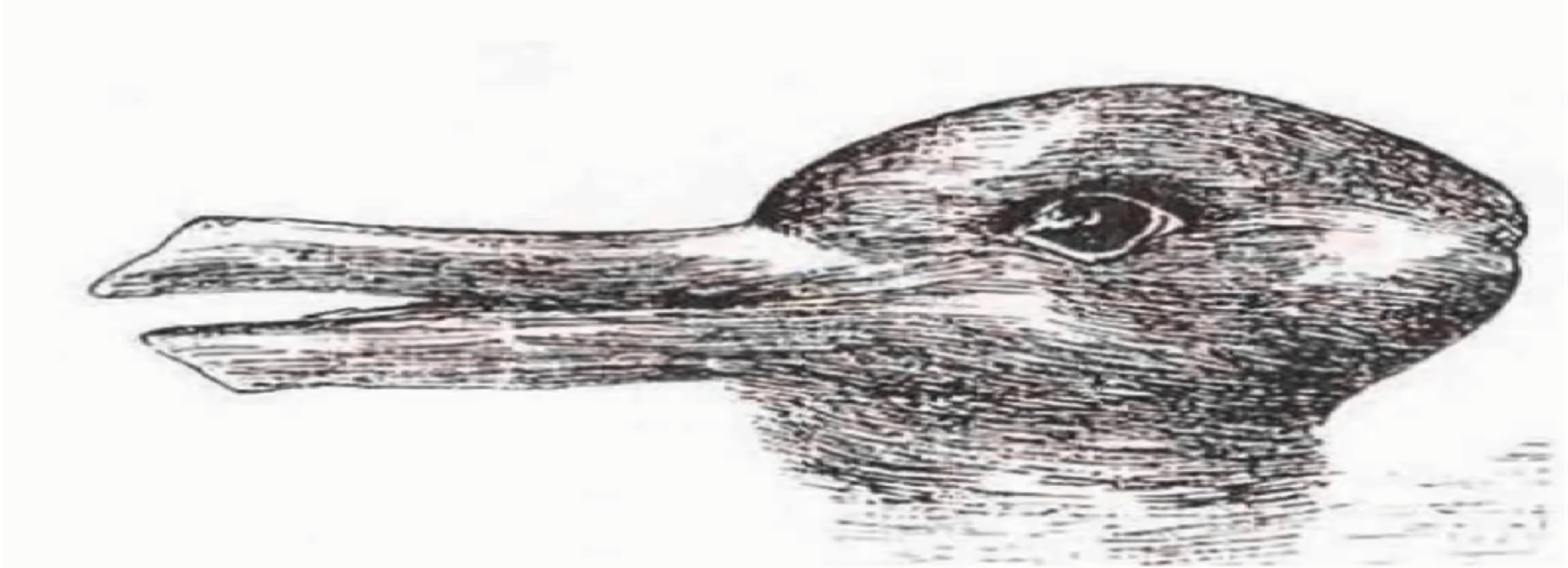
Business Use Case



- **How do our brains work?**

- Our brains depend on detecting features and they categorize the objects we see accordingly.
- What happens there is that your brain detects the object for the first time, but because the look was brief, your brain does not get to process enough of the object's features so as to categorize it correctly.



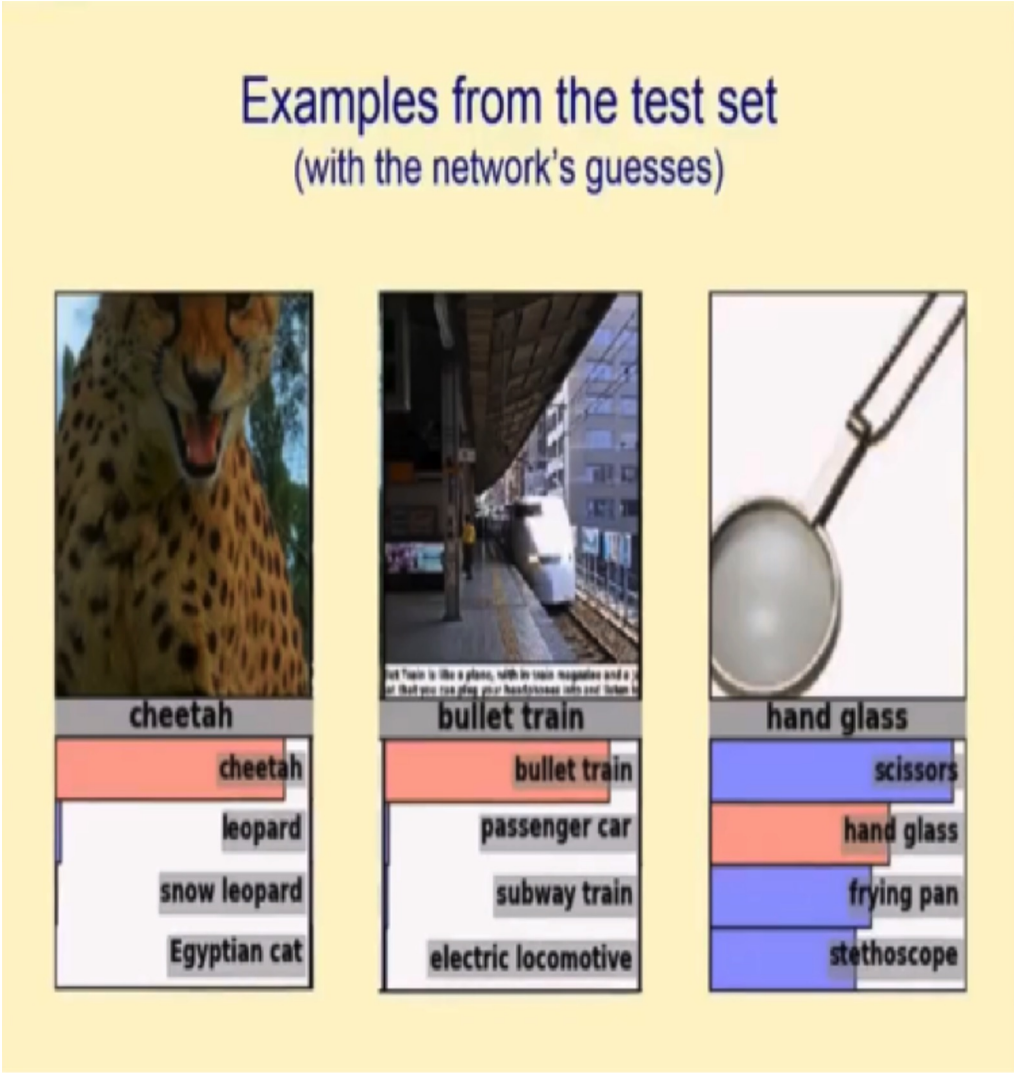
Do you see a rabbit or a duck?

That's enough with the games. The point to be made here is that your brain classifies objects in an image based on the features that it detects first.

As we proceed in our section on convolutional neural networks, you will realize the staggering degree of similarity between how these networks operate and how your brain does. They, too, categorize objects or images based on the set of features that are passed through them and that they manage to detect.

It's worth noting that the four categories that show up on this guess list are far from being the only categories that the network gets to choose from.

It usually has to sift through hundreds, or more likely thousands, of categories until it shortlists the ones you see in the image above, which it then ranks according to their probability of being the right category for the presented image.



- **Why Convolutional Neural Networks Are So Important**

To understand this, you can ask yourself a couple of simple questions:

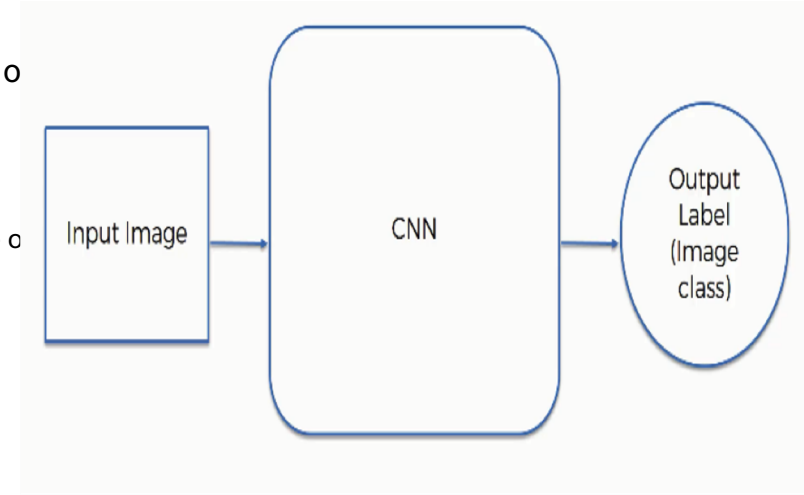
- How do self-driving cars recognize other cars as well as pedestrians and street objects?
- How did Facebook go from making you tag people in images yourself, to being able to identify your friends and automatically tag them as it does now?

And the answer to both questions would be: through the magic o

So, how do convolutional neural networks actually operate?

The first thing you need to know here is the elements that are included in the o

- Input image
- Convolutional Neural Network
- Output label (image class)



Types of image

Both types of images are similar in the following respects:

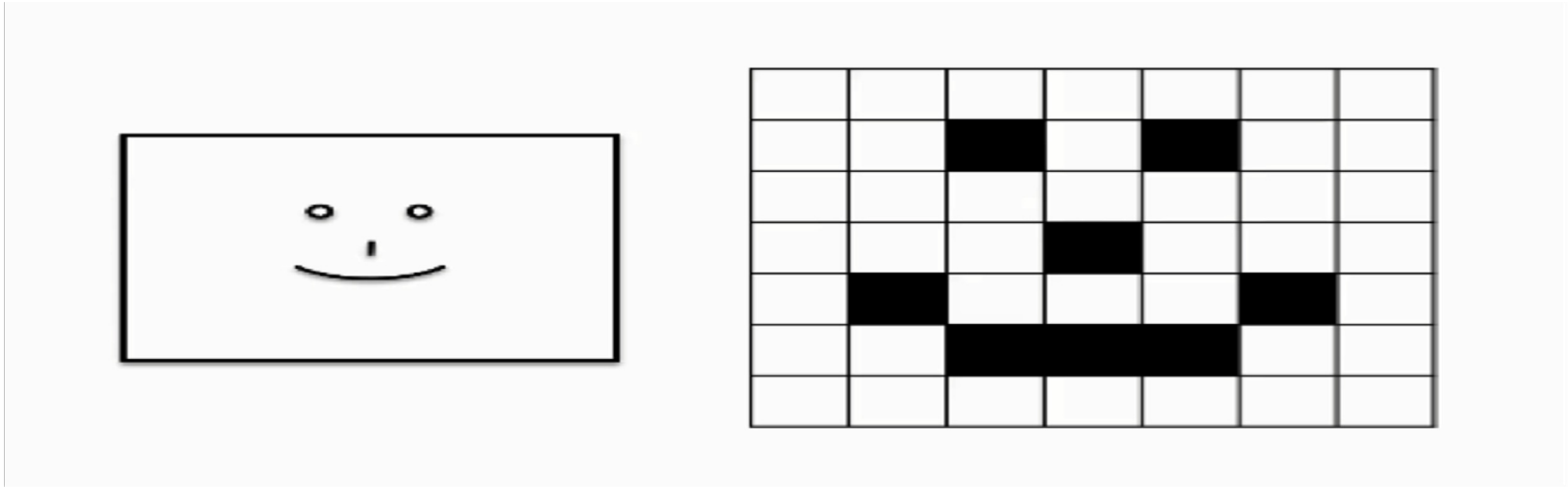
- Each pixel contains 8 bits (1 byte) of information.
- Colors are represented on a scale from 0 to 255. The reason for this is that bits are binary units, and since we have 8 of these per byte, a byte can have any of 256 (2^8) possible values. Since we count 0 as the first possible value, we can go up to 255.
- In this model, 0 is pitch black and 255 is pure white, and in between are the various (definitely more than 50!) shades of gray.
- The network does not actually learn colors. Since computers understand nothing but 1's and 0's, the colors' numerical values are represented to the network in binary terms.

Each pixel inside a colored image is represented on three levels. Since any color is a combination of red, green, and blue at different levels of concentration, a single pixel in a colored image is assigned a separate value for each of these layers.

That means that the red layer is represented with a number between 0 and 255, and so are the blue and the green layers. They are then presented in an RGB format. For example, a "hot pink" pixel would be presented to the neural network as (255, 105, 180)

Detecting Facial Expressions

Let's look at a case in which a convolutional neural network is asked to read a human smile. For the purpose of simplification, the following figure shows an extremely basic depiction of a human (loosely speaking, of course) smile.



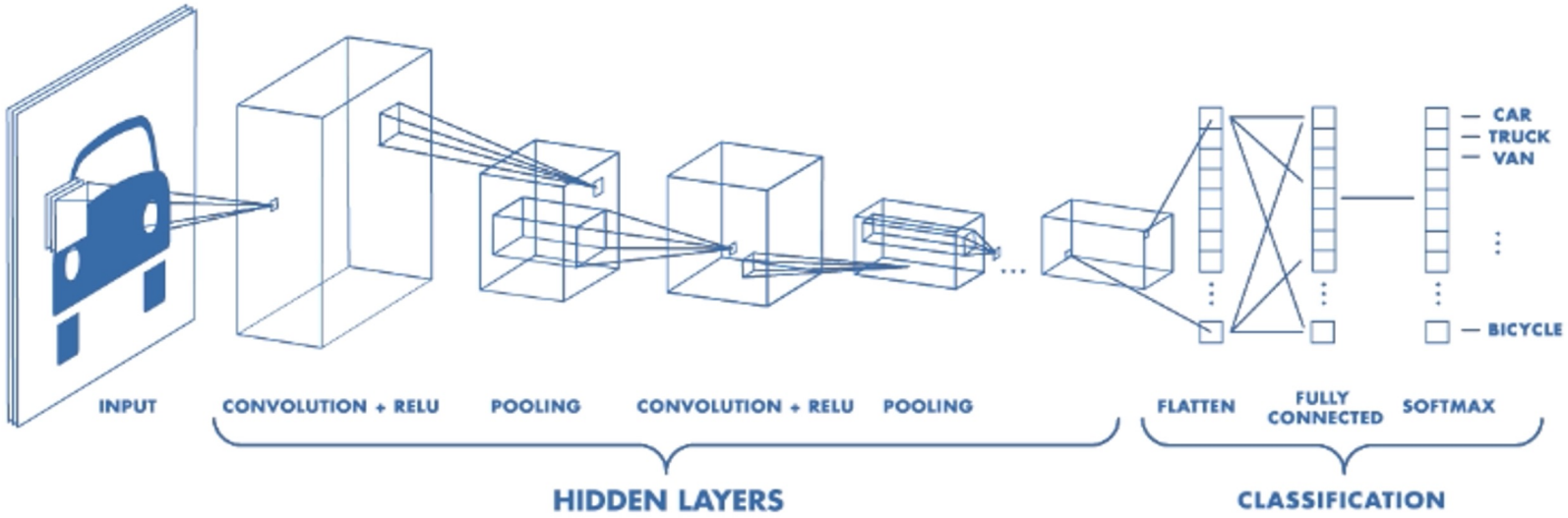
The steps that go into this process are broken down as follows:

- Step 1: Convolution
- Step 1: Activation Functions
- Step 2: Pooling
- Step 3: Flattening
- Step 4: Full Connection

CNNs have two components:

- The Hidden layers/Feature extraction part
- The Classification part

Architecture of a CNN



Feature extraction

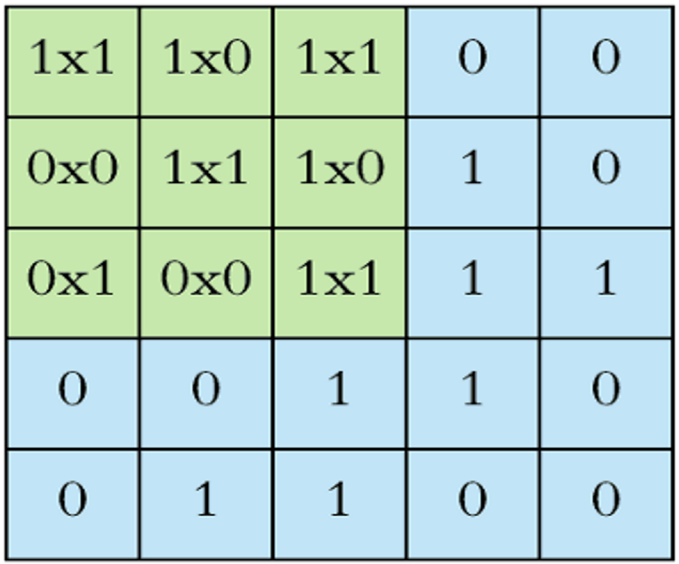
Convolution is one of the main building blocks of a CNN. The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information.

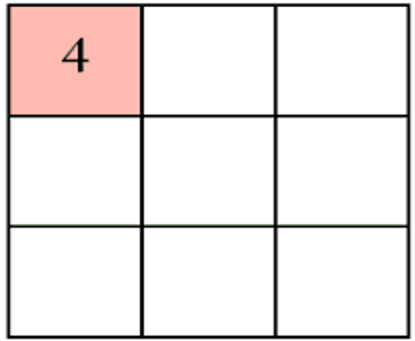
In the case of a CNN, the convolution is performed on the input data with the use of a **filter** or **kernel** (these terms are used interchangeably) to then produce a **feature map**.

We execute a convolution by sliding the filter over the input. At every location, a matrix multiplication is performed and sums the result onto the feature map.

In the animation below, you can see the convolution operation. You can see the **filter** (the green square) is sliding over our **input** (the blue square) and the sum of the convolution goes into the **feature map** (the red square).

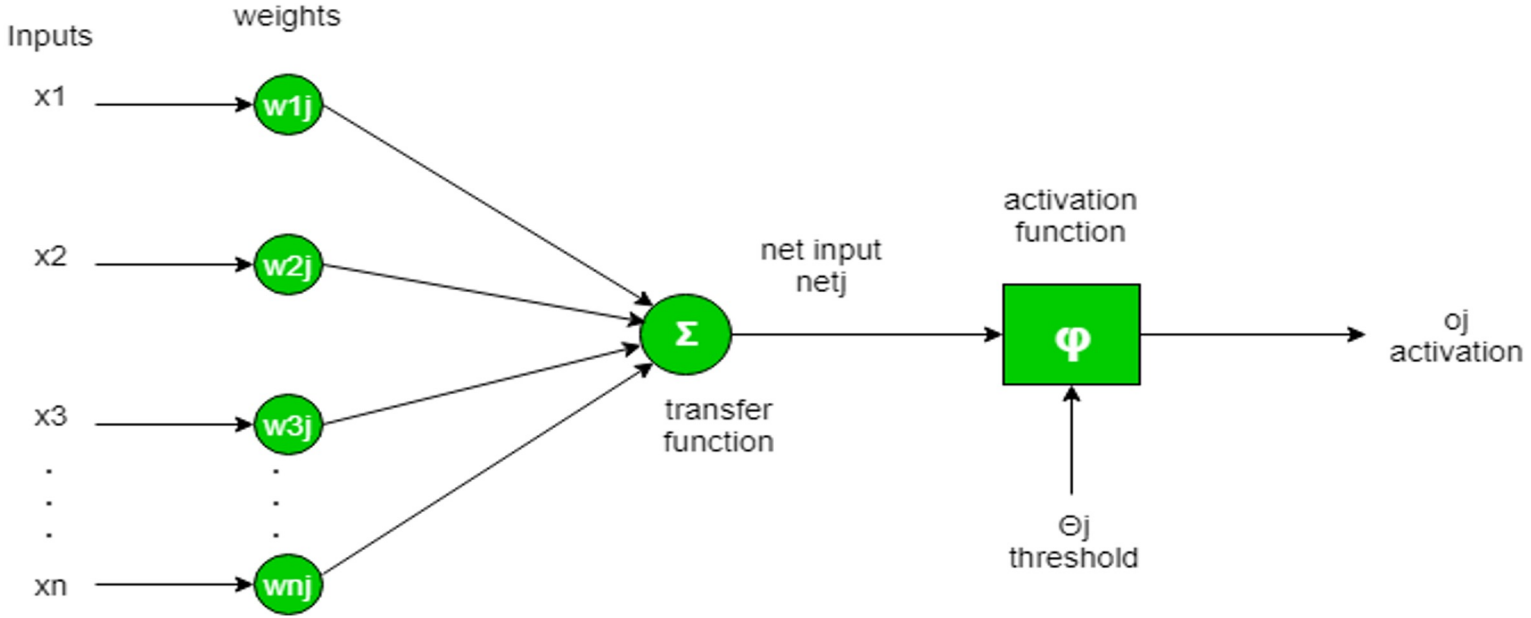
The area of our filter is also called the receptive field, named after the neuron cells! The size of this filter is 3x3.


1x1 1x0 1x1 0 0
0x0 1x1 1x0 1 0
0x1 0x0 1x1 1 1
0 0 1 1 0
0 1 1 0 0


4

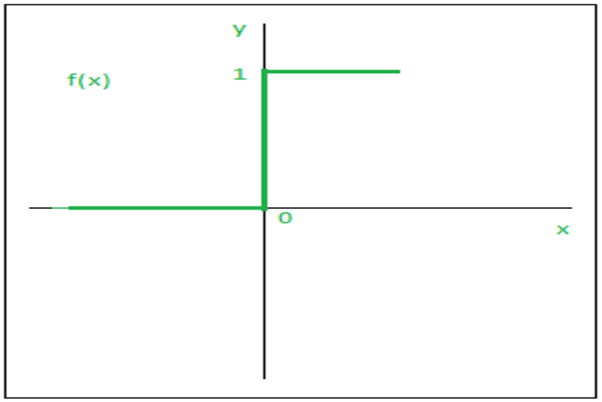
Activation Function

To put in simple terms, an artificial neuron calculates the ‘weighted sum’ of its inputs and adds a bias, as shown in the figure below by the net input.

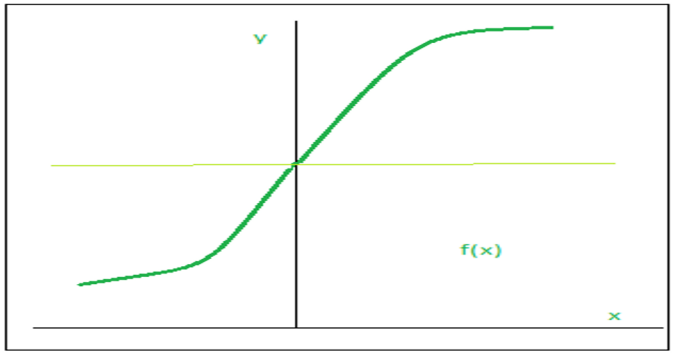


Types of Activation Functions –

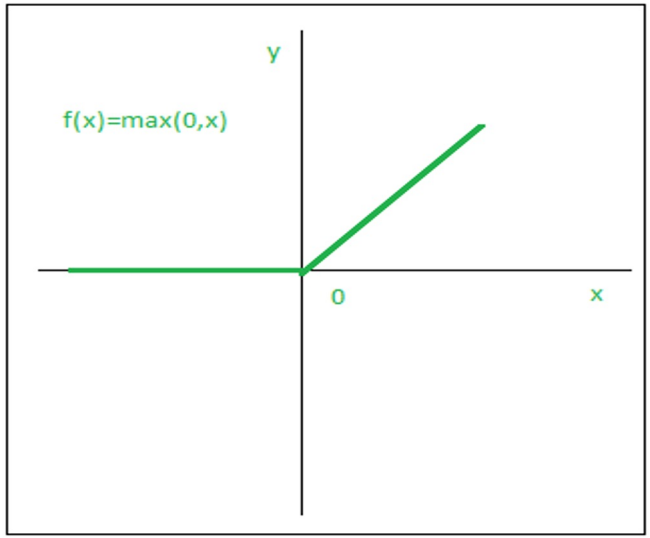
1) **Step Function:**



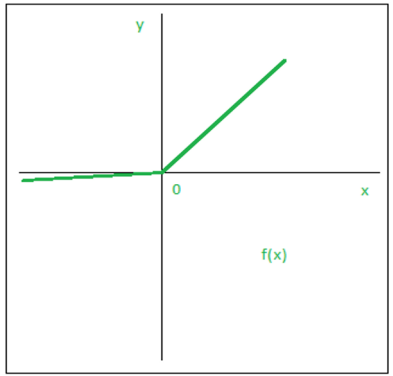
1) **Sigmoid Function :**



3) Relu:



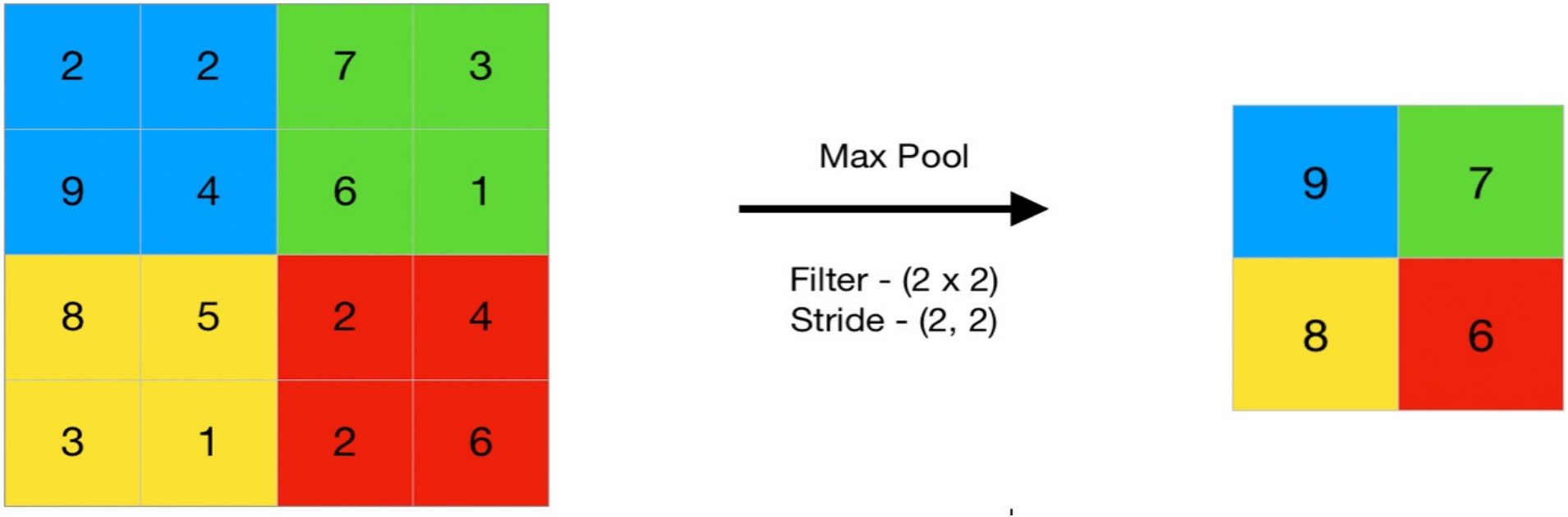
4) **Leaky ReLU:**



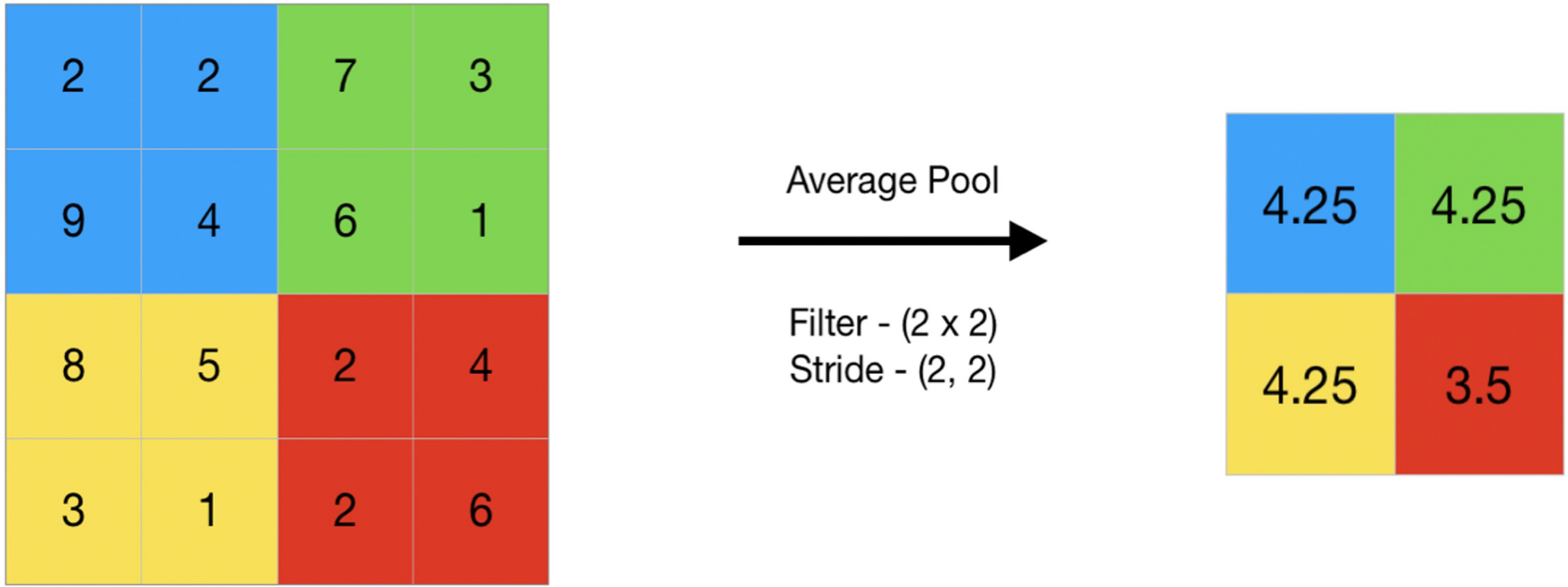
Pooling Layer

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarizing the features lying within the region covered by the filter.

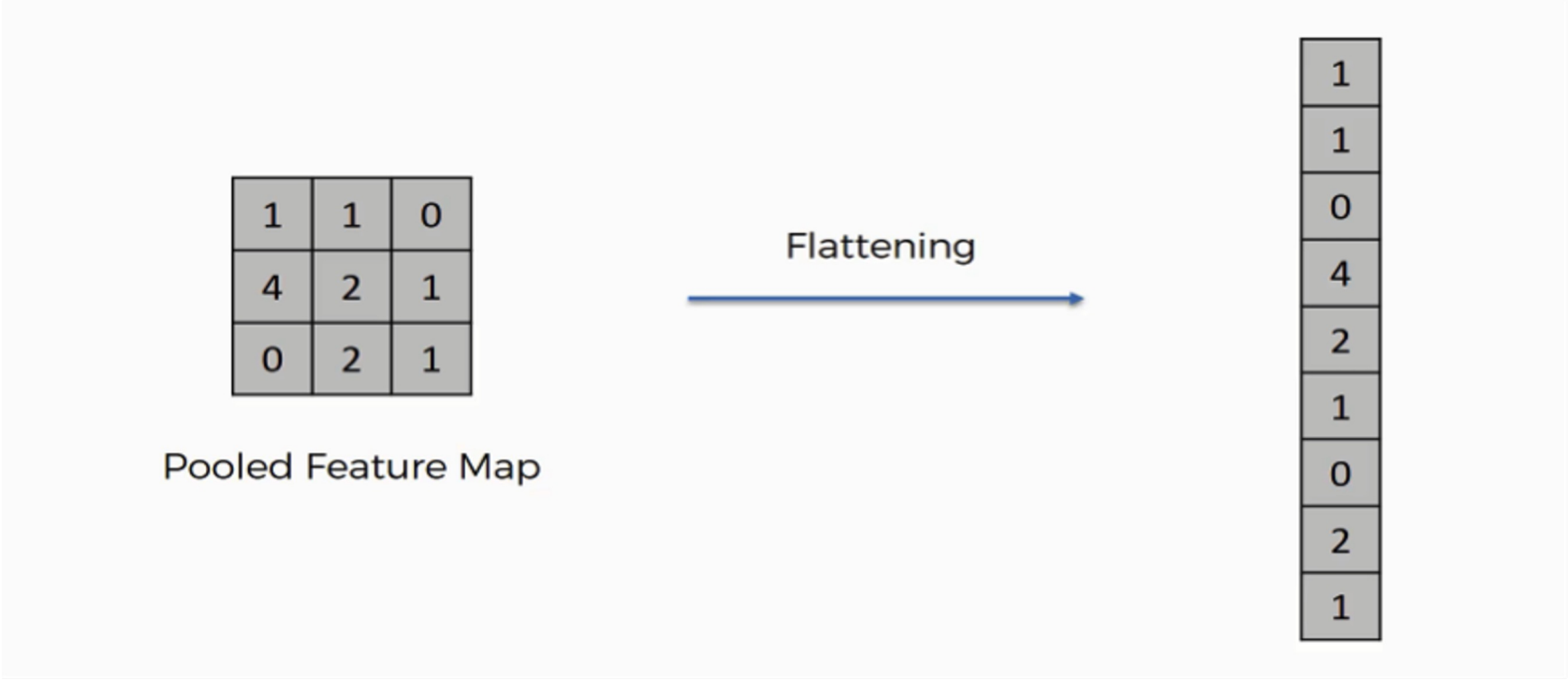
1) Max Pooling

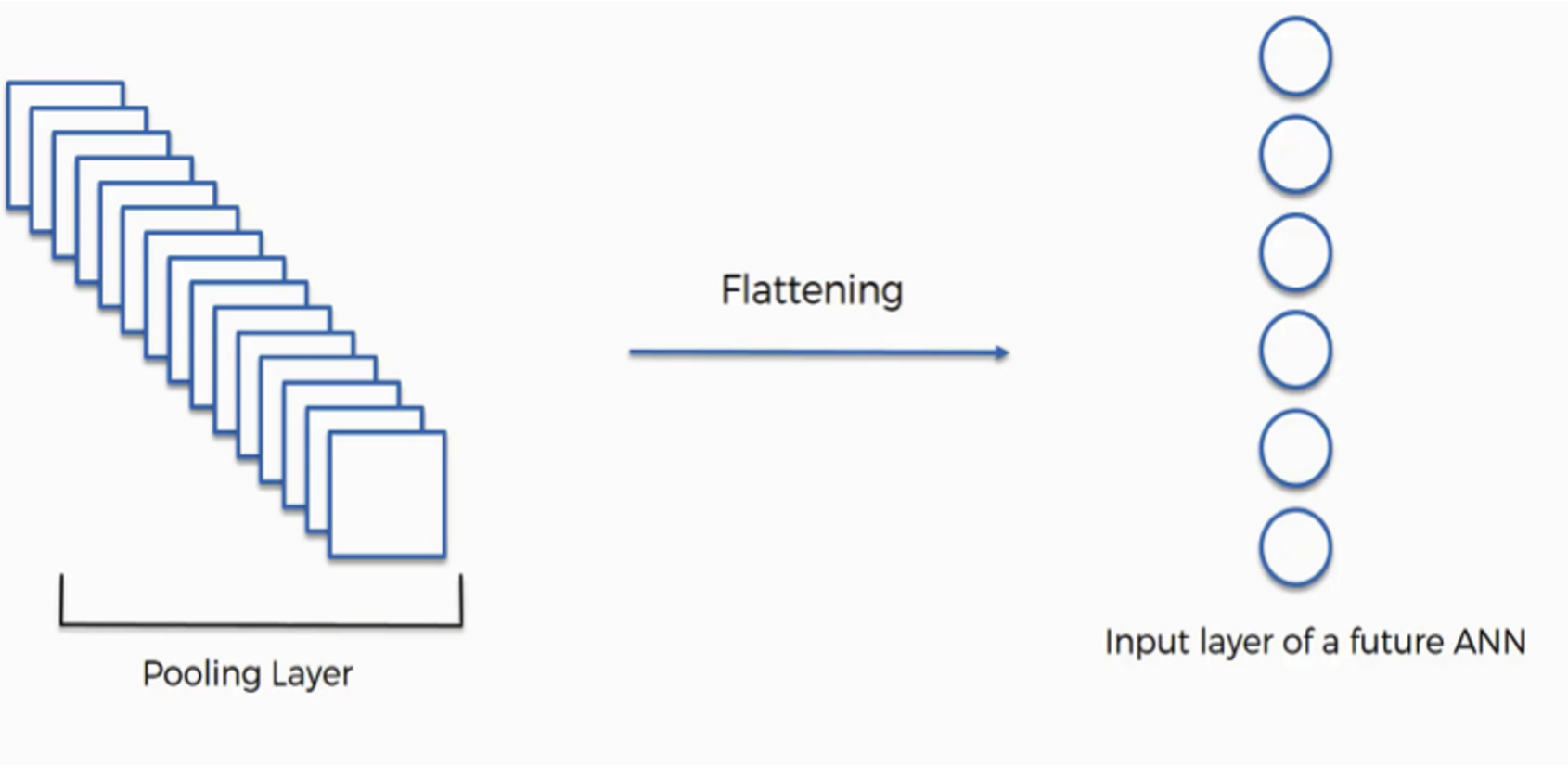


2) Average Pooling

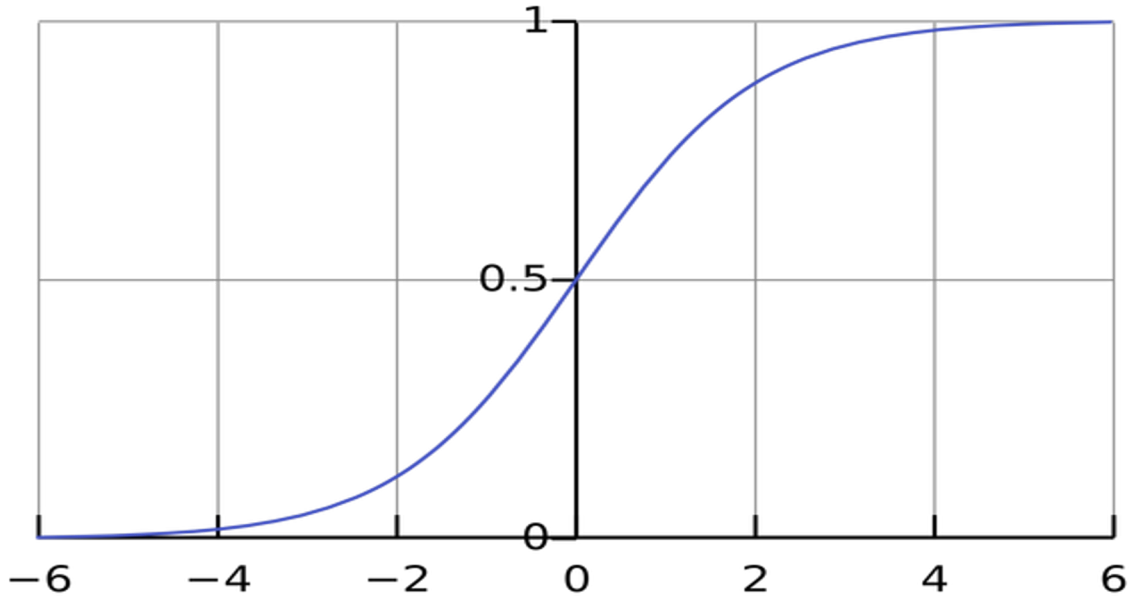


Flatenning



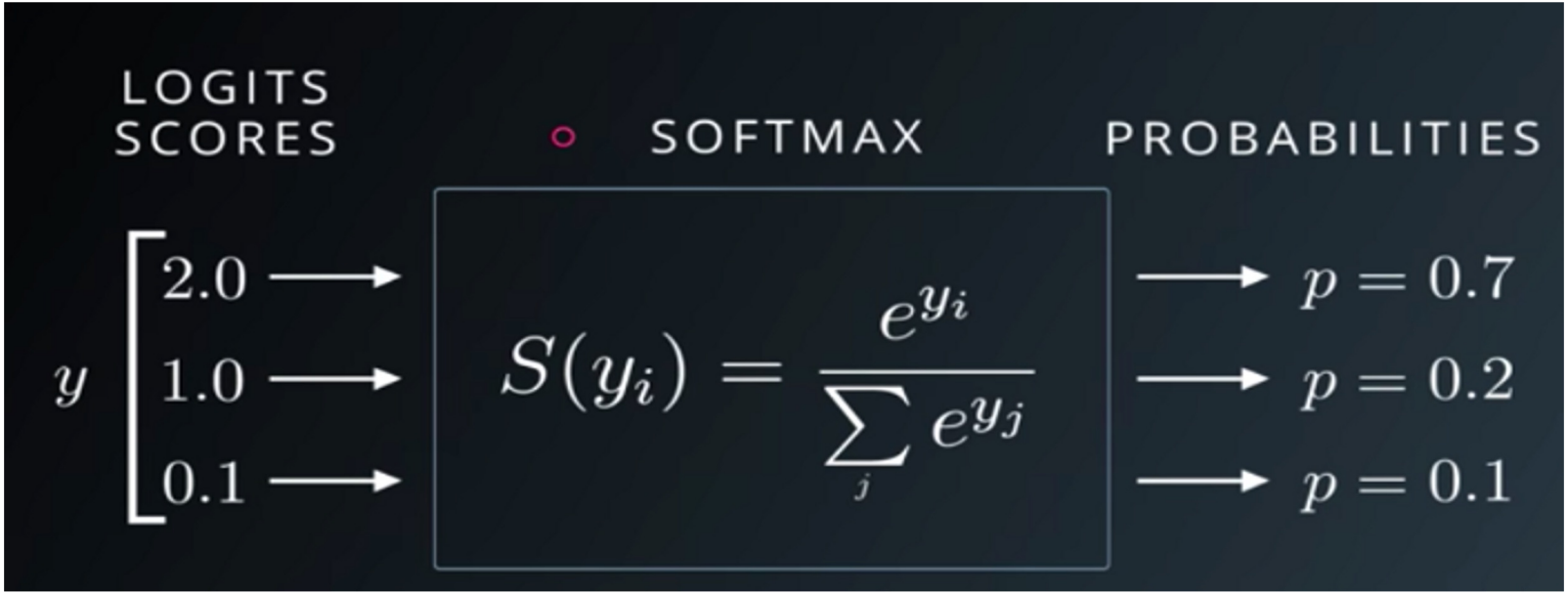

Flattening
Pooling Layer
Input layer of a future ANN

Sigmoid



$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Softmax


LOGITS
SCORES
SOFTMAX
PROBABILITIES
y
2.0
1.0
0.1
$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$
$p = 0.7$
$p = 0.2$
$p = 0.1$

Sigmoid vs Softmax

Sigmoid input values: -0.5, 1.2, -0.1, 2.4

Sigmoid output values: 0.37, 0.77, 0.48, 0.91

SoftMax input values: -0.5, 1.2, -0.1, 2.4

SoftMaxoutput values: 0.04, 0.21, 0.05, 0.70

The key takeaway from this example is:

- **Sigmoid**: probabilities produced by a Sigmoid are independent. Furthermore, they are *not* constrained to sum to one: 0.37 + 0.77 + 0.48 + 0.91 = 2.53. The reason for this is because the Sigmoid looks at each raw output value separately.
- **Softmax**: the outputs are interrelated. The Softmax probabilities will always sum to one by design: 0.04 + 0.21 + 0.05 + 0.70 = 1.00. In this case, if we want to increase the likelihood of one class, the other has to decrease by an equal amount.

Bus|Car|Truck|Train--Sigmoid -> 0.37|0.77|0.48|0.91---> Multilabel classification

Bus|Car|Truck|Train--Softmax-> 0.04|0.21|0.05|0.70 ---> Multi class classification

Error and Loss Function:

In most learning networks, error is calculated as the difference between the actual output and the predicted output.

$$J(w) = p - \hat{p}$$

Binary Cross Entropy Loss

Entropy is the measure of randomness in the information being processed, and cross entropy is a measure of the difference of the randomness between two random variables.

If the divergence of the predicted probability from the actual label increases, the cross-entropy loss increases. Going by this, predicting a probability of .011 when the actual observation label is 1 would result in a high loss value. In an ideal situation, a “perfect” model would have a log loss of 0. Looking at the loss function would make things even clearer

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

Other major Classification losses

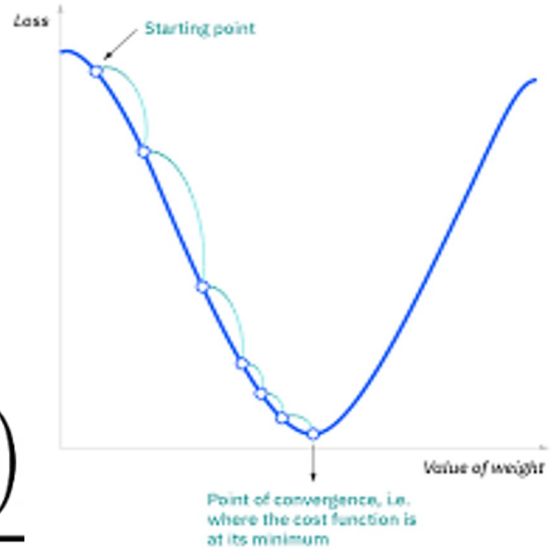
1) Categorical Cross Entropy Loss
2) Hinge Loss
3) Kullback Leibler Divergence Loss (KL Loss)

Optimizers

Optimizers are algorithms or methods used to minimize an error function(*loss function*)or to maximize the efficiency of production. Optimizers are mathematical functions which are dependent on model's learnable parameters i.e Weights & Biases. Optimizers help to know how to change weights and learning rate of neural network to reduce the losses.

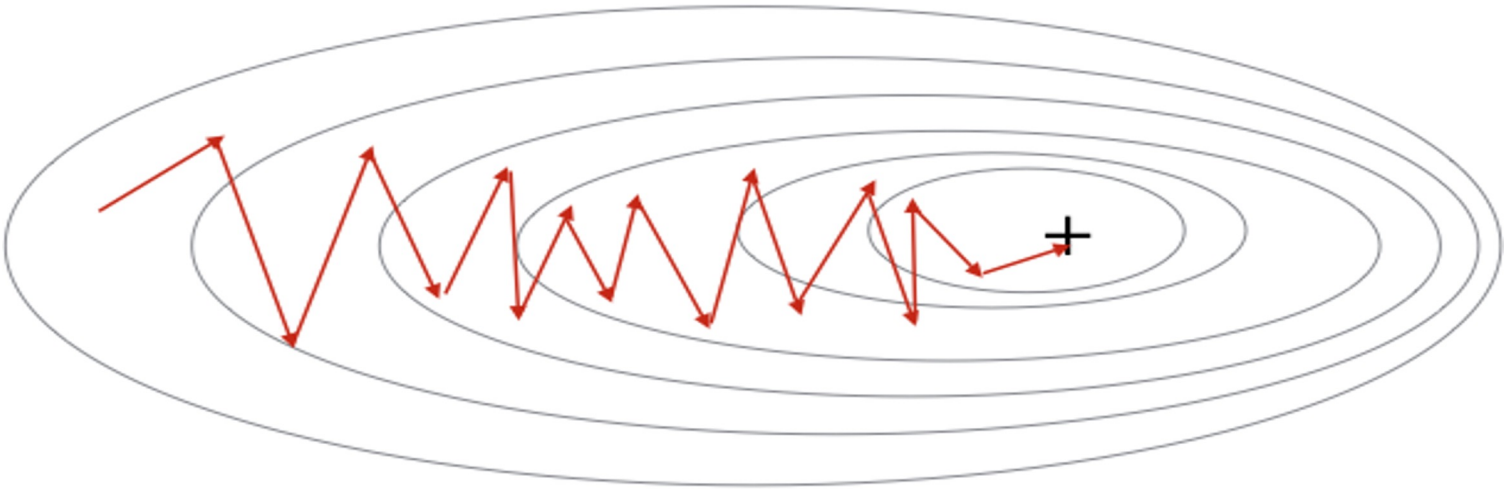
Gradient Descent

$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

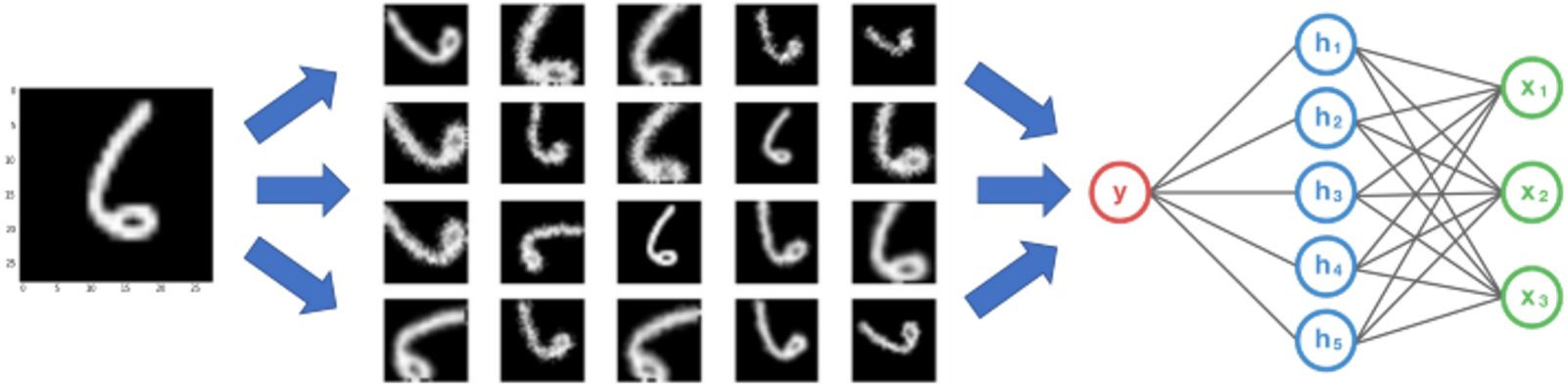


Learning Rate

Stochastic Gradient Descent



Augmentation



Calculating No of parameters

CONV layer:

(shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters).

POOL layer: 0

Fully Connected Layer (FC): ((current layer neurons c * previous layer neurons p)+1*c).