Convolutional Neural Network

An Introduction to Convolutional Neural Networks

Arunava Follow
Dec 25, 2018 · 6 min read

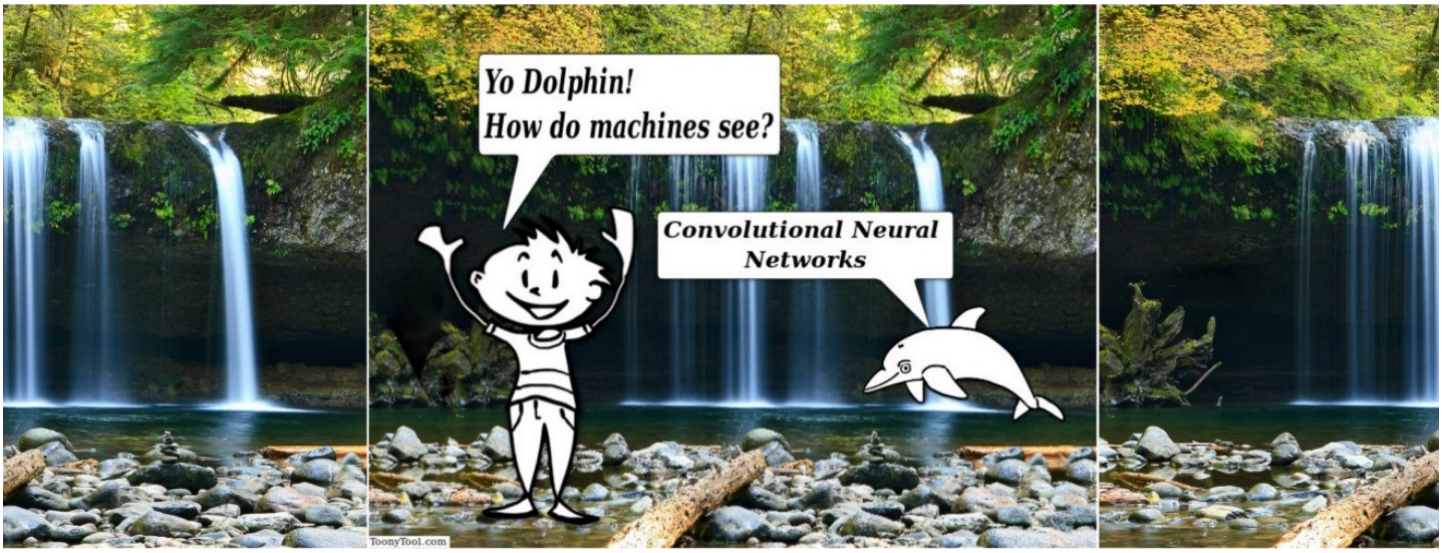
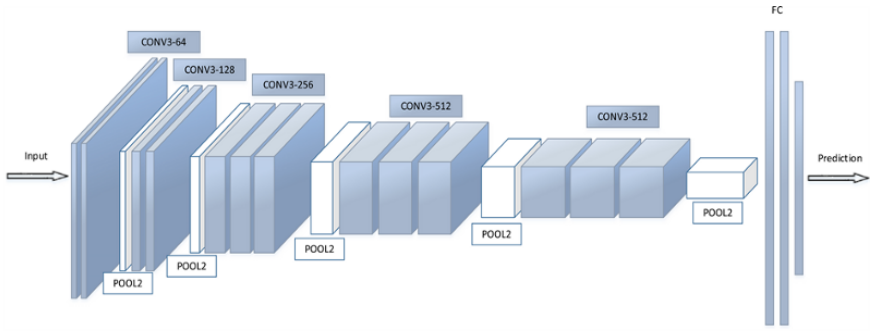


Fig 1. Toon Discovers how machines see

In this article, we will see what are Convolutional Neural Networks, ConvNets in short. ConvNets are the superheroes that took working with images in deep learning to the next level. With ConvNets, the input is a image, or more specifically, a 3D Matrix.

. . .

Let's start by looking at how a ConvNet looks!



[Fig 2.] Convolutional Neural Network

In a nutshell,
A ConvNet usually has 3 types of layers:
1) **Convolutional Layer** (*CONV*)
2) **Pooling Layer** (*POOL*)
3) **Fully Connected Layer** (*FC*)

Let's look at each of these layers in more detail:

Convolutional Layer

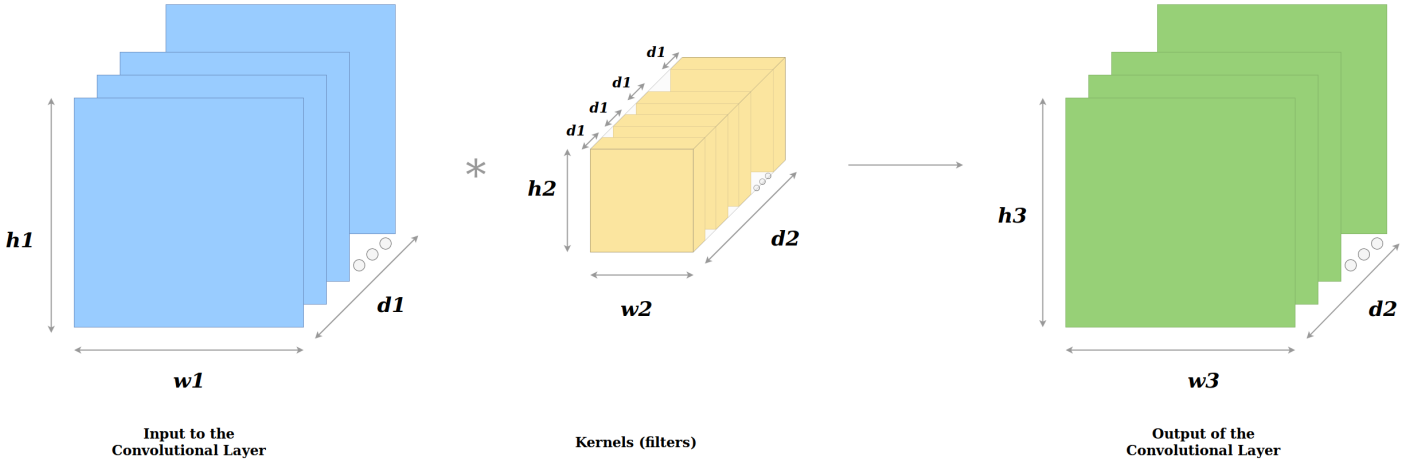


Fig 3. Convolutional Layer

Convolutional Layer is the first layer in a CNN.
It gets as ***input*** a matrix of the dimensions *[h1 * w1 * d1]*, which is the blue matrix in the above image.

Next, we have ***kernels*** (filters).
Kernels?
A kernel is a matrix with the dimensions *[h2 * w2 * d1],* which is *one* yellow cuboid of the multiple cuboid (kernels) stacked on top of each other (in the kernels layer) in the above image.
For each convolutional layer, there are multiple kernels stacked on top of each other, this is what forms the yellow 3-dimensional matrix in *Fig 2*, which is of dimensions *[h2 * w2 * d2]*, where *d2* is the number of kernels.

For each kernel, we have its respective **bias**, which is a scalar quantity.

And then, we have a ***output*** for this layer, the green matrix in Fig 2, which has dimensions *[h3 * w3 * d2]*.

Let's throw light on some obvious things from above.
1) The depth (*d1*) (or the number of channels) of the input and of *one* kernel **is the same**.
2) The depth (*d2*) of the output **is equal** to the number of kernels (i.e. the depth of the orange 3-dimensional matrix).

Alright, so we have inputs, kernels and outputs. Now let's look at what happens with a 2D input and a 2D kernel, i.e. $d1 = 1$.

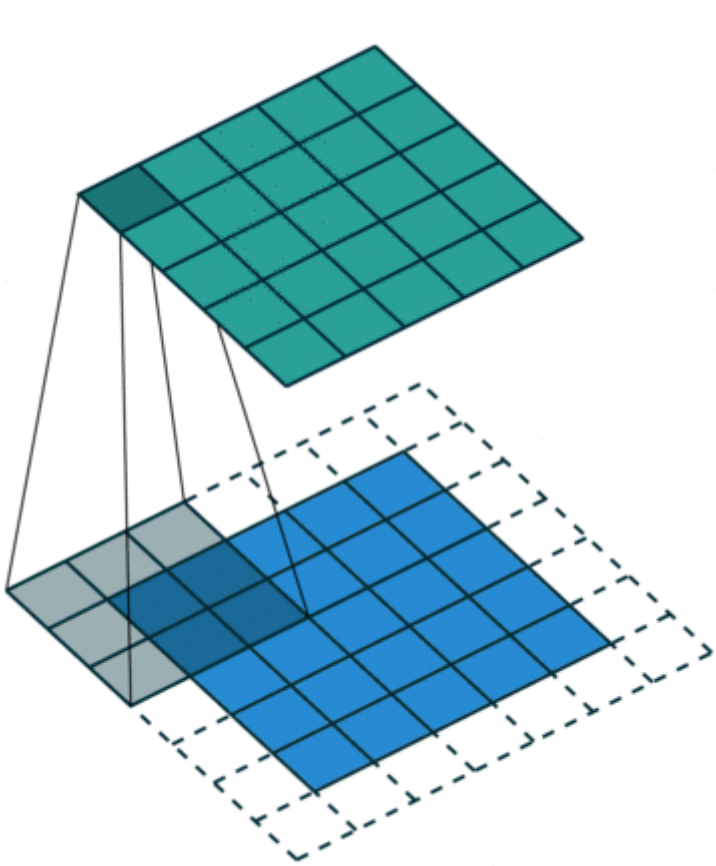


Fig 4. Calculation of output using 2D Convolution

For each position of the kernel on the image, each number on the kernel gets multiplied with the corresponding number on the input matrix (*blue matrix*) and then they all are summed up for the value in the corresponding position in the output matrix (*green matrix*).

With $d1 > 1$, the same thing occurs for each of the channels and then they are added up together and then summed up with the bias of the respective filter and this forms the value in corresponding position of the output matrix. Let's visualize!

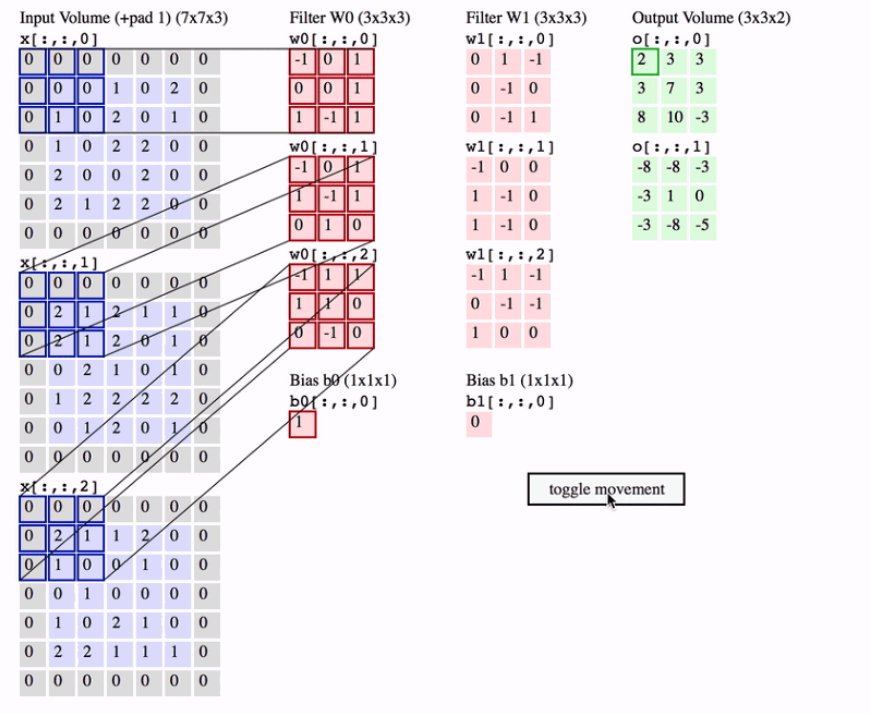


Fig 5. An example of how Inputs are mapped to Outputs

And this is forms one (of *d2*) matrix of the output layer.

This entire process is repeated with all the *d2* kernels which forms the *d2* channels in the output layer.

Pooling Layer

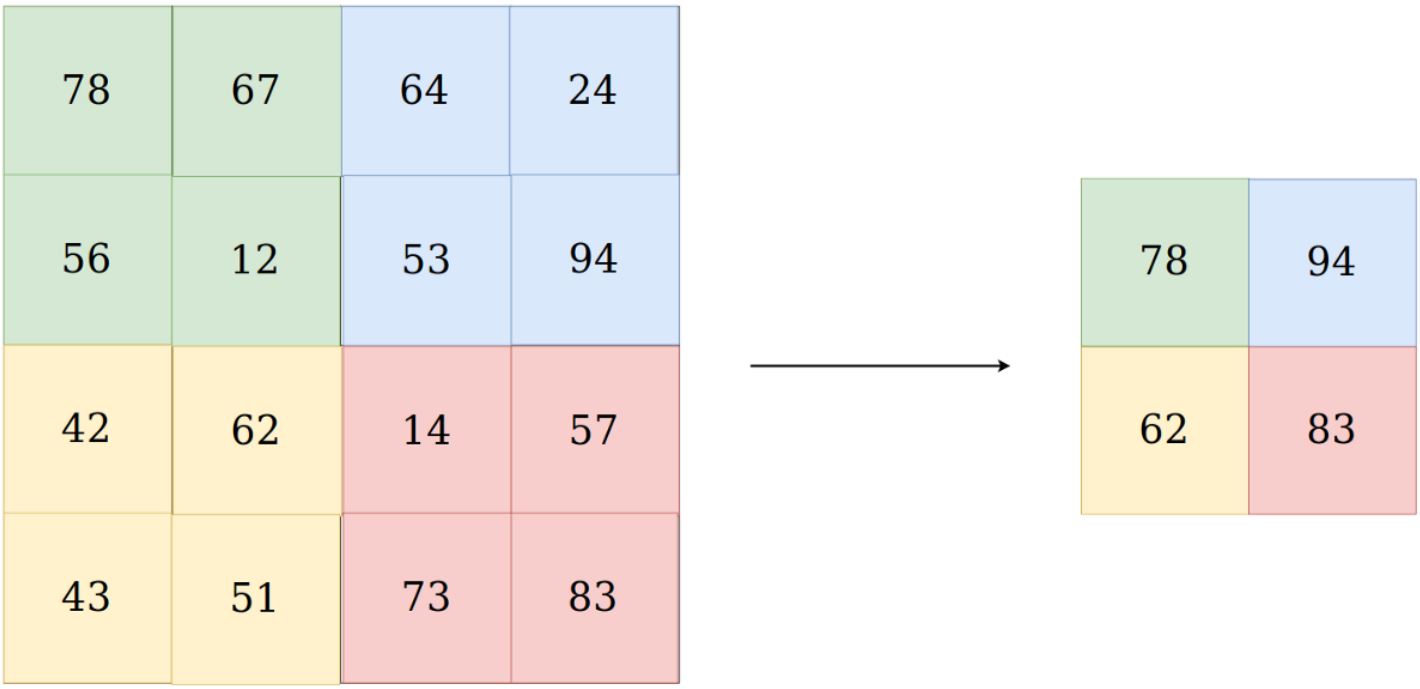


Fig 6. Max Pooling Layer

There are two types of pooling:
1) Max Pooling
2) Average Pooling

The main purpose of a pooling layer is to reduce the number of parameters of the input tensor and thus
- Helps reduce overfitting
- Extract representative features from the input tensor
- Reduces computation and thus aids efficiency

The input to the Pooling layer is tensor.

In case of Max Pooling, an example of which is shown in the *Fig 6,* a kernel of size `n*n` *(2x2 in the above example)* is moved across the matrix and for each position the **max value is taken** and put in the corresponding position of the output matrix.

In case of Average Pooling, a kernel of size `n*n` is moved across the matrix and for each position the **average is taken of all the values** and put in the corresponding position of the output matrix.

This is repeated for each channel in the input tensor. And so we get the output tensor.
So, a thing to note is, **Pooling downsamples the image in its height and width but the number of channels(depth) stays the same.**

Fully Connected Layer

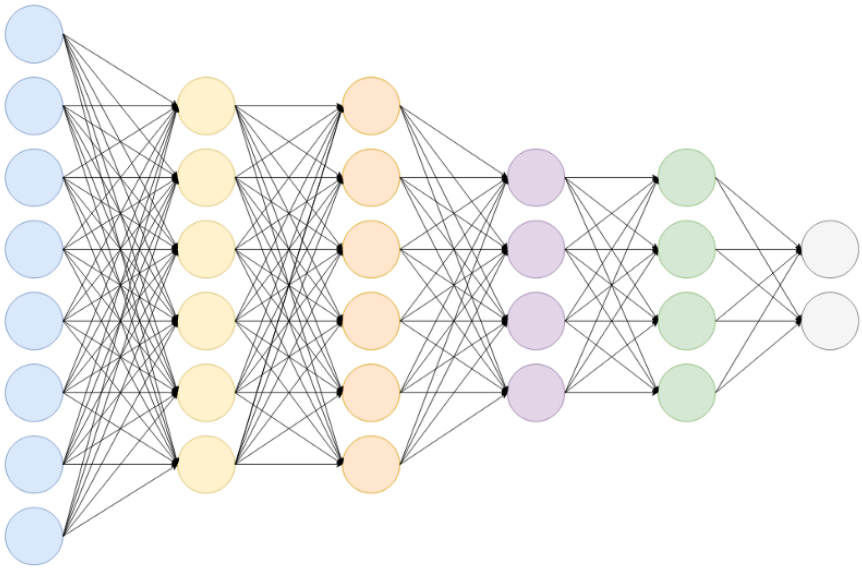


Fig 4. Fully Connected Network

Fully Connected Layer is simply, *feed forward neural networks*. Fully Connected Layers form the last few layers in the network.

The **input** to the fully connected layer is the output from the *final* Pooling or Convolutional Layer, which is ***flattened*** and then fed into the fully connected layer.

> *Flattened?*
> *The output from the final (and any) Pooling and Convolutional Layer is a 3-dimensional matrix, to flatten that is to unroll all its values into a vector.*
> *Let's visualize!*

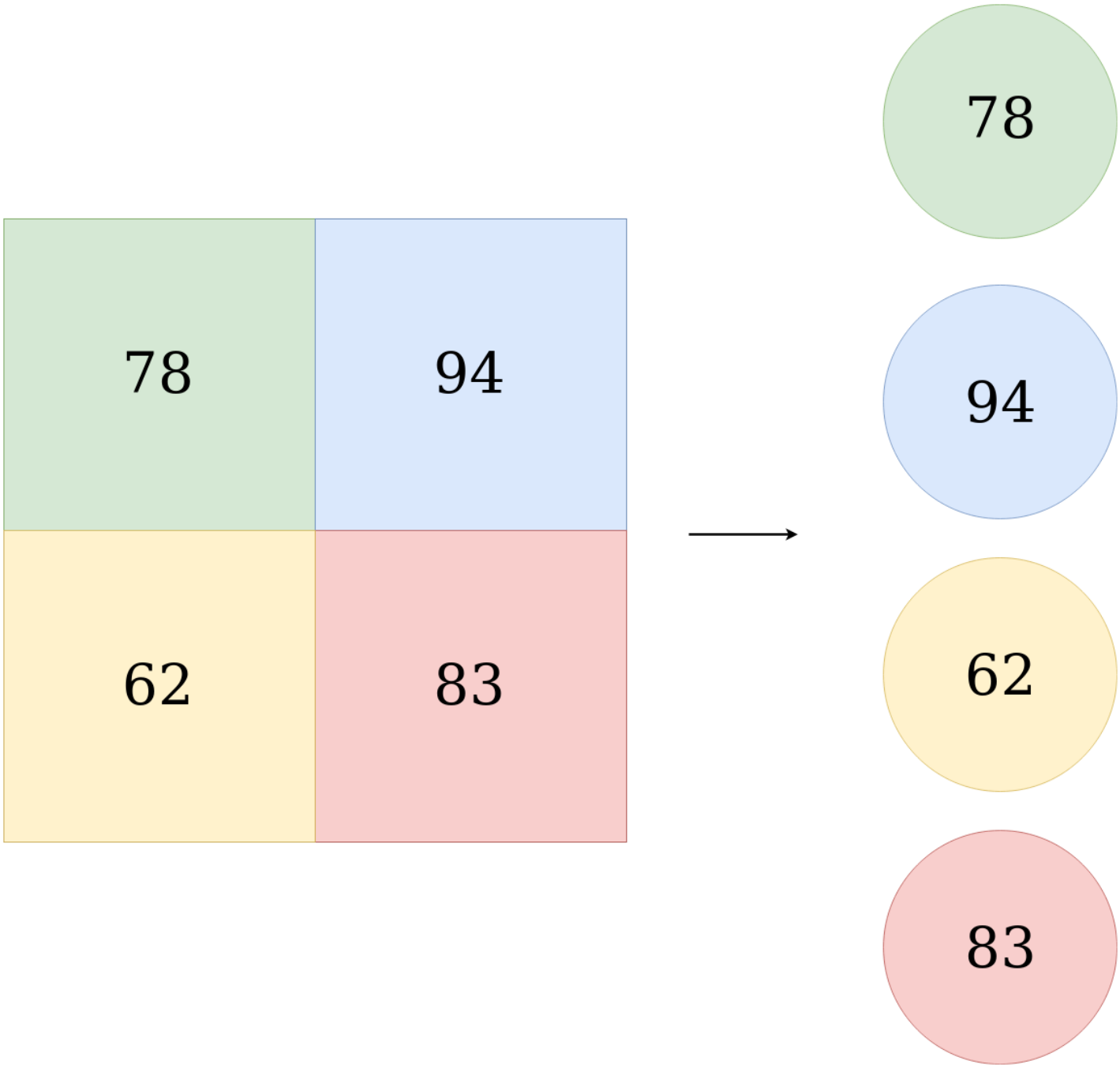


Fig 5. Flattening

This Flattened vector is then connected to a few fully connected layers which are same as Artificial Neural Networks and perform the same mathematical operations!

For each layer of the Artificial Neural Network, the following calculation takes place

$$g(Wx + b)$$

Fig 6. ANN Calculation for each layer

where,
x—*is the input vector with dimension* ***[p_l, 1]***
W—*Is the weight matrix with dimensions* ***[p_l, n_l]*** *where,* ***p_l*** *is the number of neurons in the previous layer and* ***n_l*** *is the number of neurons in the current layer.*
b—*Is the bias vector with dimension* ***[p_l, 1]***
g—*Is the activation function, which is usually* ***ReLU****.*

This calculation is repeated for each layer.

After passing through the fully connected layers, the final layer uses the ***softmax activation function*** *(instead of ReLU)* which is used to get probabilities of the input being in a particular class (*classification*).

And so finally, we have the probabilities of the object in the image belonging to the different classes!!

. . .

And that is how the Convolutional Neural Network works!!
And input images get classified as labels!!

. . .

Now, let's visualize how to calculate the dimensions of the output tensor from the input tensor.

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

Fig 7. Output Dimension Calculations from Input Dimensions

where,
W1*—is the width / height of the input tensor*
F*—is the width / height of the kernel*
P*—is the padding*
S*—is the stride*
W2*—is the output width / height*

> ***Padding?***
> *Usually, the input matrices are padded around with zero so that the kernel can move over the matrix uniformly and the resultant matrix have the desired dimension. So, P=1 means that all sides of the input matrix is padded with 1 layer of zeros as in Fig 4 with the dotted extras around the input (blue) matrix.*

> ***Stride?***
> *The kernel moves over the matrix 1 pixel at a time (Fig 4). So, this is said to have a stride of 1. We can increase the stride to 2 so that the kernel moves over the matrix 2 pixels at a time. This will, in turn, affect the dimensions of the output tensor and helps reduce overfitting.*

And what happens to the number of channels of the output tensor?
Well, in case of a Convolutional Layer, it is equal to the number of kernels.
And in case of a Pooling Layer, the channels in the input tensor and the output tensor remains the same!