CNN

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and learn the image and can differentiate them from the others. They are very similar to the neural networks but a differential feature of the CNN is that they make the explicit assumption that the entries are images, which allows us to encode certain properties in the architecture to recognize specific elements in the images.

CNNs have proven to be successful in many different real-life case studies and applications like Image classification, object detection, face recognition, self-driving cars.

Convolutional neural networks are used to find patterns in an image. You do that by convoluting over an image and looking for patterns. Each CNN layer learns filters of increasing complexity. The first layers learn basic feature detection filters like edges, corners. The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses. The last layers have higher representations like they learn to recognize full objects, in different shapes and positions.

Architecture

A CNN model can be thought as a combination of two components: feature extraction part and the classification part.

The convolution + pooling layers perform feature extraction. Given an image the convolution layer detects features such as two eyes, long ears, four legs, a short tail and so on.

The fully connected layers then act as a classifier on top of these features and assign a probability for the input image being a label.

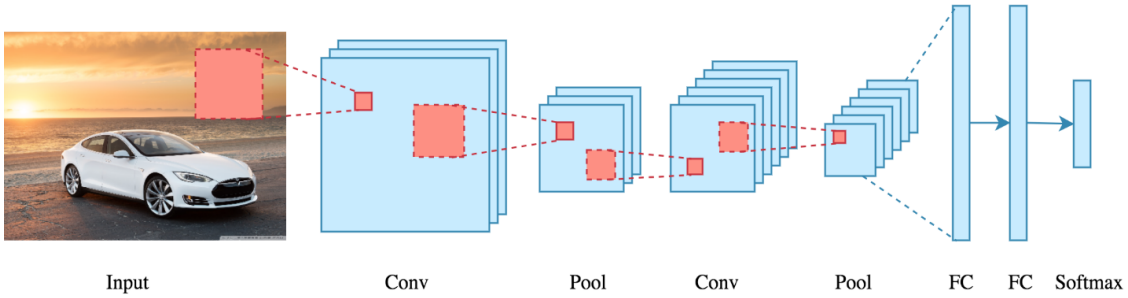
The process of building a Convolutional Neural Network always involves four major steps.

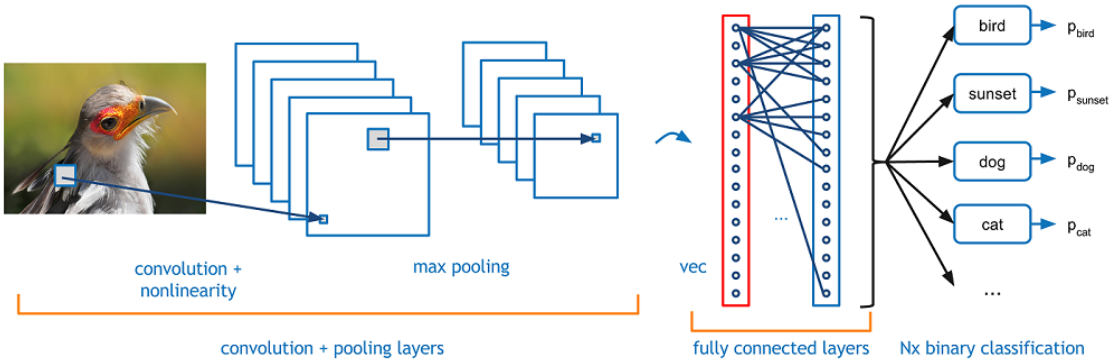
Step - 1: Convolution

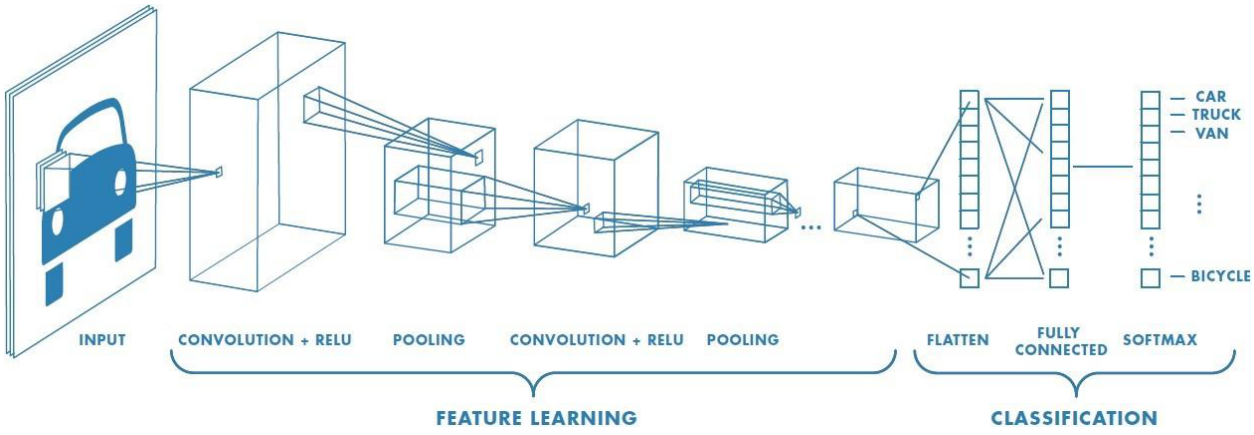
Step - 2: Pooling

Step - 3: Flattening

Step - 4: Full connection



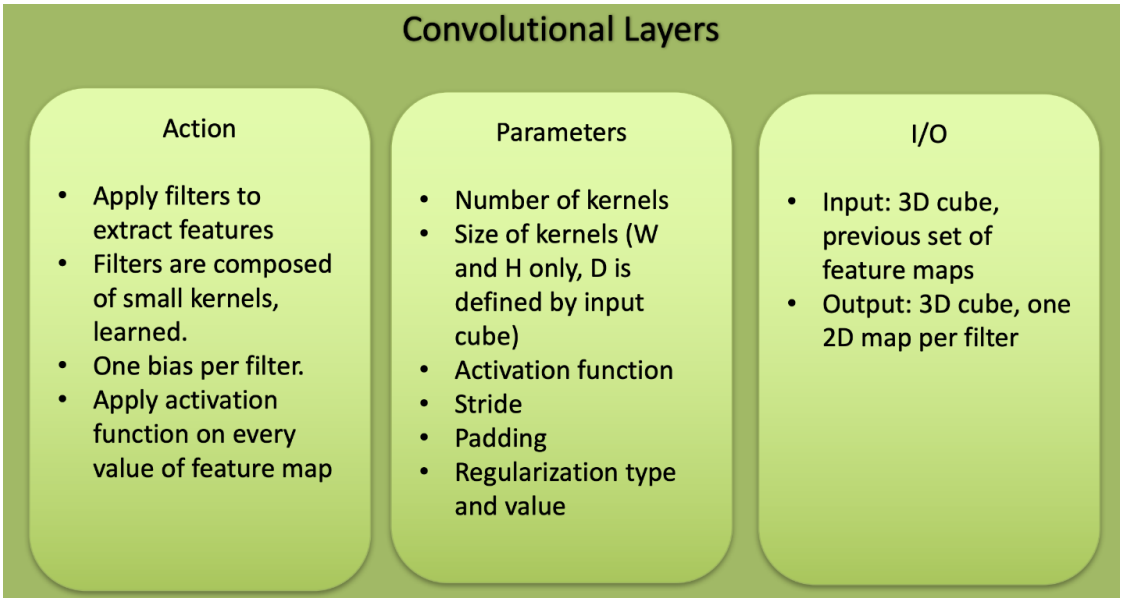




Convolution

The main building block of CNN is the convolutional layer.. The goal of a convolutional layer is extract features from an input image or filtering in other sense.

Convolution is a mathematical operation that takes two inputs such as image matrix and a filter/kernel(stacks of weights represented as a vector). It is a mathematical operation to merge two sets of information. The most important parameters in CNN are the number of filters/kernels and the size of the filters/kernels.



Convolutional layers are the layers where filters/kernels are applied to the original image or to other feature maps in a deep CNN. Convolving the image with a filter/kernel produces a feature map that highlights the presence of a given feature in the image. Convolution of an image with different filters/kernels can perform operations such as edge detection, blur and much more. As we move over an image we effective check for patterns in that section of the image. This works because of filters/kernels which are multiplied by the values outputted by the convolution. When training an image, these weights change, and so when it is time to evaluate an image, these weights return high values if it thinks it is seeing a pattern it has seen before. The combinations of high weights from various filters/kernels let the network predict the content of an image.

In a convolutional layer, we are basically applying multiple filters/kernels at over the image to extract different features but most importantly, we are learning those filters/kernels and output of this operation is a feature map. To be clear, each filter/kernel is convolved with the entirety of the 3D input cube but generates a 2D feature map. Because we have multiple filters/kernels, we end up with a 3D output - one 2D feature map per filter/kernel. The feature map dimension can change drastically from one convolutional layer to the next. We can enter a layer with a 32x32x16 input and exit with a 32x32x128 output if that layer has 128 filters/kernels.

How convolution operation works?

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

1	0	1
0	1	0
1	0	1

Filter / Kernel

On the left side is the input to the convolution layer. On the right is the convolution filter, also called the filter/kernel.

We perform the convolution operation by sliding this filter over the input. At every location, we do element-wise matrix multiplication and sum the result. This sum goes into the feature map. The green area where the convolution operation takes place is called the receptive field. This is called a 3x3 convolution due to the shape of the filter/kernel. Due to the size of the filter/kernel the receptive field is also 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

Input x Filter

4		

Feature Map

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

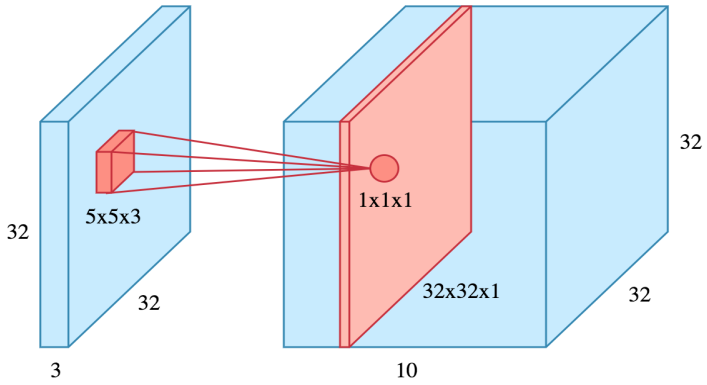
Input x Filter

4	3	

Feature Map

This was an example convolution operation shown in 2D using a 3x3 filter/kernel. But in reality these convolutions are performed in 3D. In reality an image is represented as a 3D matrix with dimensions of height, width and depth, where depth corresponds to color channels (RGB). A convolution filter/kernel has a specific height and width like 3x3 or 5x5 and by design it covers the entire depth of its input so it needs to be 3D as well.

We perform multiple convolutions on an input, each using a different filter/kernel resulting in a distinct feature map. We then stack all these feature maps together and that becomes the final output of the convolution layer. But first let's start simple and visualize a convolution using a single filter.



Let's say we have a 32x32x3 image and we use a filter/kernel of size 5x5x3 (note that the depth of the convolution filter matches the depth of the image with both being 3). When the filter/kernel is at a particular location it covers a small volume of the input and we perform the convolution operation described above. The only difference is this time we do the sum of matrix multiply in 3D instead of 2D but the result is still a scalar. We slide the filter/kernel over the input like above and perform the convolution at every location aggregating the result in a feature map. This feature map is of size 32x32x1 (shown as the red slice on the right)

If we used 10 different filters we would have 10 feature maps of size 32x32x1 and stacking them along the depth dimension would give us the final output of the convolution layer i.e. a volume of size 32x32x10 (shown as the large blue box on the right). Note that the height and width of the feature map are unchanged and still 32 and it's due to padding and we will elaborate on that shortly.

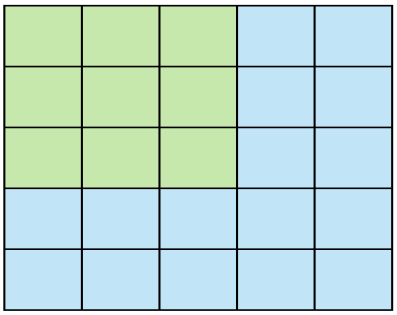
Activation Function

We apply activation function to output of convulation. Activation functions work exactly as in other neural networks i.e. a value is passed through a function that squashes the value into a range.

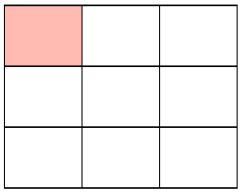
The most commonly used activation function is ReLu activation function. It takes an input 'x' and returns 'x' if it is positive else it returns 0. The reason ReLu function is used because it is really cheap to perform.

Stride

Stride is the number of pixels shifted over the input matrix. Stride specifies how much we move the convolution filter at each step. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. By default, the value is 1, as you can see in the figure below.



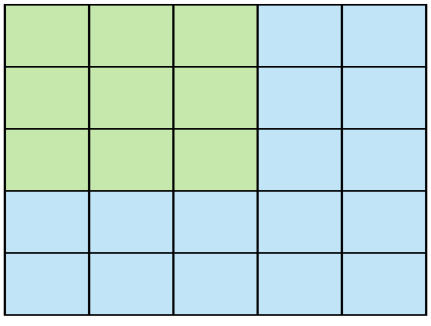
Stride 1



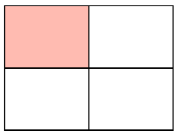
Feature Map

In the image above, the green part of input is receptive field. The red part of feature map is output generated with help of only that receptive field. We can have bigger strides if we want less overlap between the receptive fields. This also makes the resulting feature map smaller since we are skipping over potential locations.

The following figure demonstrates a stride of 2. We see that the size of the feature map is smaller than the input. If we want to maintain the same dimensionality we can use padding to surround the input with zeros.



Stride 2



Feature Map

Padding

Padding is done to make the size of feature map match with the size of the input. For this we resize the input shape. This is done by surrounding the input with zeros or the values on the edge. Now the dimensionality of the feature map matches the input.

Padding is commonly used in CNN to preserve the size of the feature maps otherwise they would shrink at each layer which is not desirable. The 3D convolution figures we saw above used padding, that's why the height and width of the feature map was the same as the input (both 32x32) and only the depth changed.

Hyperparameters

We have 4 important hyper parameters to decide on

Filter size - We typically use 3x3 filters, but 5x5 or 7x7 are also used depending on the application. There are also 1x1 filters but they have interesting applications. Remember that these filters are 3D and have a depth dimension as well but since the depth of a filter at a given layer is equal to the depth of its input we omit that.

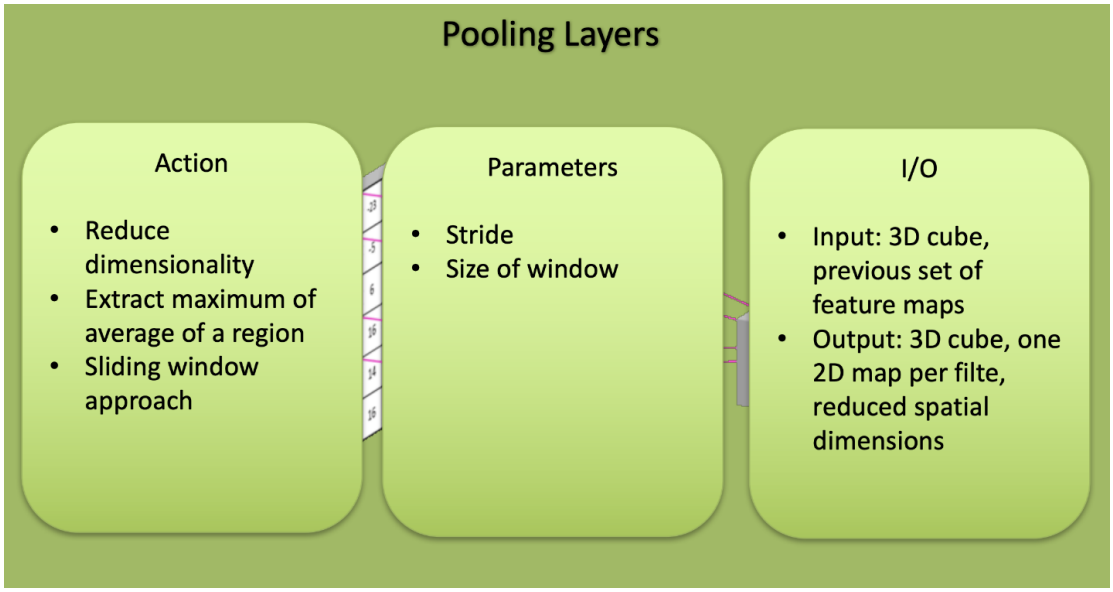
Filter count: This is the most variable parameter. It's a power of two anywhere between 32 and 1024. Using more filters results in a more powerful model but we risk overfitting due to increased parameter count. Usually we start with a small number of filters at the initial layers and progressively increase the count as we go deeper into the network.

Stride: we keep it at the default value 1.

Padding: we usually use padding.

In CNN architectures convolution is typically performed with 3x3 windows, stride 1 and with padding.

Pooling



After a convolution operation we usually perform pooling to reduce the dimensionality. Pooling layers work on each feature map independently i.e. it down sample each feature map independently reducing the height and width keeping the depth intact. This enables us to reduce the number of parameters which both shortens the training time and combats overfitting.

This layer partitions the input image into a set of non-overlapping rectangles and for each such sub-region it outputs a value. The intuition is that the exact location of a feature is less important than its rough location relative to other features. Contrary to the convolution operation pooling has no parameters. It slides a window over its input, and simply takes the max or average value in the window. Similar to a convolution, we specify the window size and stride.

Spatial pooling

Spatial pooling also called subsampling or down sampling reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types

-Max pooling takes the largest element from the rectified feature map.

-Average pooling takes average of the elements.

-Sum of all elements in the feature map call as sum pooling.

Overlapping Max Pooling

Max Pooling layers are usually used to down sample the width and height of the tensors keeping the depth same. Overlapping Max Pool layers are similar to the Max Pool layers, except

the adjacent windows over which the max is computed overlap each other. The authors used pooling windows of size 3×3 with a stride of 2 between the adjacent windows. This overlapping nature of pooling helps to reduce the top-1 error rate.

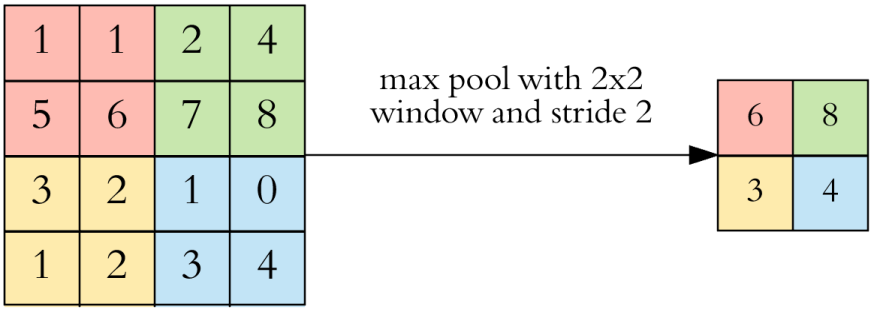
The pooling layer is used to reduce spatial dimensions but not the depth. The main advantages of reducing spatial dimensions are

-By having less spatial information you gain computation performance.

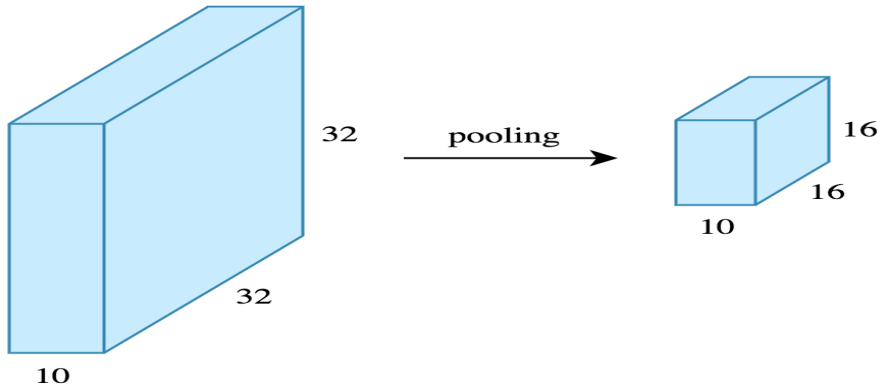
-By having less spatial information means you have less parameters to train the model on thus reduce chances of over-fitting.

How pooling works?

Here is the result of max pooling using a 2x2 window and stride 2. Each color denotes a different window. Since both the window size and stride are 2, the windows are not overlapping.



If the input to the pooling layer has the dimensionality 32x32x10, using the same pooling parameters described above, the result will be a 16x16x10 feature map. Both the height and width of the feature map are halved, but the depth doesn't change because pooling works independently on each depth slice the input.



By halving the height and the width, we reduced the number of weights to 1/4 of the input. Considering that we typically deal with millions of weights in CNN architectures, this reduction is a pretty big deal.

In CNN architectures, pooling is typically performed with 2*2 windows, stride 2 and with no padding.

Flattening

Remember that the output of both convolution and pooling layers are 3D volumes but a fully connected layer expects a 1D vector of numbers. So we flatten the output of the final pooling layer to a vector and that becomes the input to the fully connected layer. Flattening is simply arranging the 3D volume of numbers into a 1D vector.

Fully Connected

After the flattening layer we add a couple of fully connected layers to wrap up the CNN architecture. 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. The Flattened vectors from final pooling or convolutional layer is then connected to a few fully connected layers which are same as ANN layers and perform the same mathematical operations.

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

$$g(Wx + b)$$

where

x is the input vector with dimension [p, 1]

W is the weight matrix with dimensions [p, n] where, p is the number of neurons in the previous layer and n is the number of neurons in the current layer

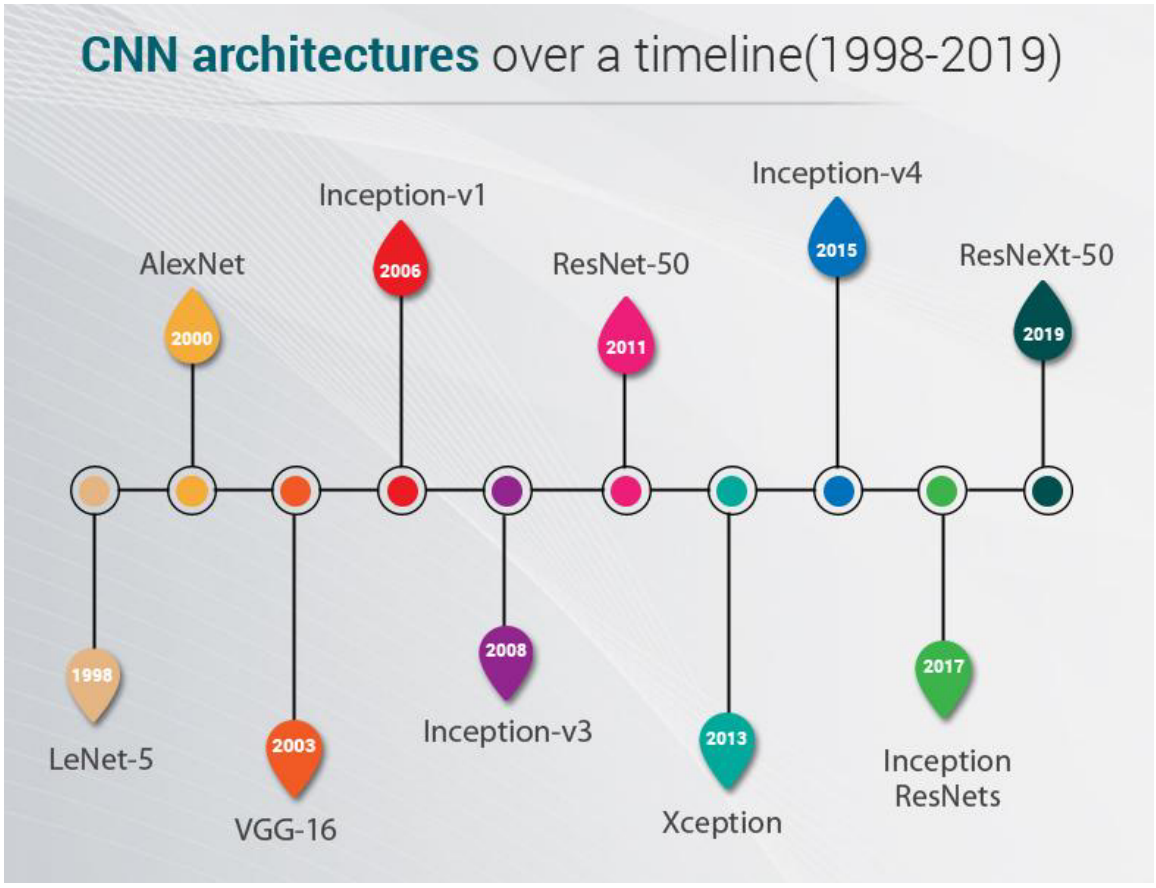
b is the bias vector with dimension [p 1]

g is the activation function which is usually ReLU.

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.

CNN Architectures



Some of the famous CNN architectures are

LeNet-5

AlexNet

VGG

Inception-v1

Inception-v3

ResNet-50

Xception

Inception-v4

Inception ResNets

ResNext-50