CNN: Convolution Neural Networks

**INTRODUCTION:**

A Convolutional Neural Network (CNN) is a type of Deep Learning architecture commonly used for image classification and recognition tasks. It consists of multiple layers, including Convolutional layers, Pooling layers, and fully connected layers. The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

For, an Algorithm to be classified as a CNN algorithm, it MUST HAVE Convolution Layers in them.

**FILTERS or KERNELS:**

When adding a convolutional layer to a model we also have to specify how many filters we want the layer to have.

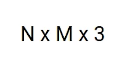
A filter can technically just be thought of as a relatively small matrix for which we decide the number of rows and number of columns that this matrix has. The values within the matrix are initialized with random numbers and the network learns the optimal filters through backpropagation and gradient descent.

**Working of a Filter/Kernel:**

NOTE : MUST READ ABOUT HOW IMAGES ARE MADE? WHAT ARE PIXELS AND PIXEL VALUES? AND WHAT ARE GREYSCALE AND RGB IMAGES?

<https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/>

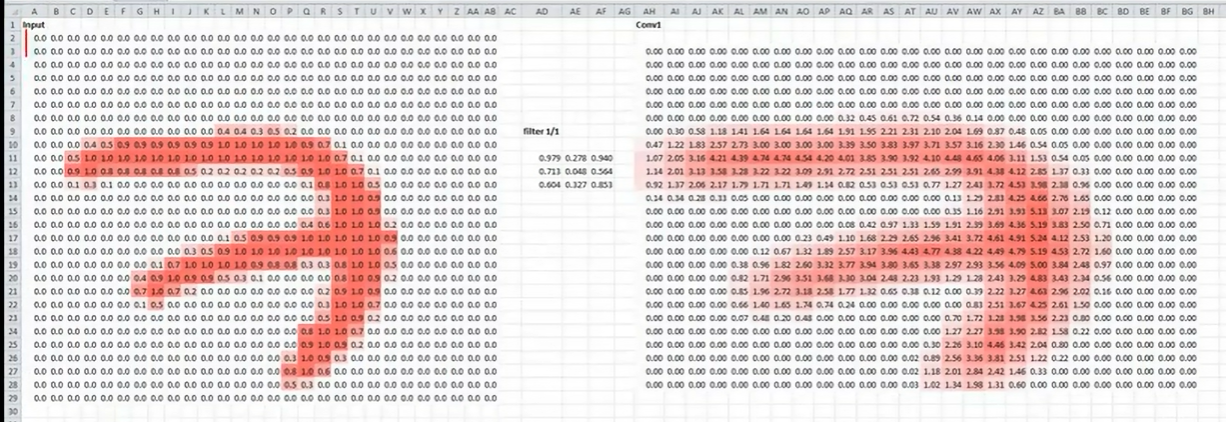
1. For a [grayscale images](https://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm), the pixel value is a single number that represents the brightness of the pixel. The most common *pixel format* is the *byte image*, where this number is stored as an integer which can take from a range of possible values of 0 to 255. Typically, zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray.
2. For RGB [color images](https://homepages.inf.ed.ac.uk/rbf/HIPR2/colimage.htm), separate red, green and blue components must be specified for each pixel, and so the pixel `value' is actually a vector of three numbers. Often the three different components are stored as three separates `grayscale' images known as *color planes* (one for each of red, green and blue), which have to be superimposed when displaying or processing. Each of these R,G & B metrices would have their own matrix (each with shape = NxM), having values ranging from 0 to 255, where each of these numbers represents the intensity of the pixels or the shades of red, green, and blue in their respective matrix. Finally, all of these channels or all of these matrices are superimposed so the shape of the image, when loaded in a computer, will be-



where N is the number of pixels across the height, M would be the number of pixels across the width, and 3 is representing the number of channels i.e R,G and B.

( …..continue)

Say, we have a filter of size 3 X 3 (size chosen by us). Now when this convolutional layer receives input(say, input can be a image of say 28\*28 pixels), the filter will slide over each 3x3 set of pixels from the input itself until it has slid over every 3x3 block of pixels from the entire image. This sliding is actually referred to as convolving. So really we should say that the filter is going to convolve across each 3x3 block of pixels from the input.



Left image: Input image of 28 X 28 pixels

Centre matrix/image : Filter with 3X3 size

Right Image : Output values of the convolution layer (NOTE: Ignore the 7 written, in output we only get a 26X 26 matrix filled with values) .

**NOTE for Calculating the size of OUTPUT:**

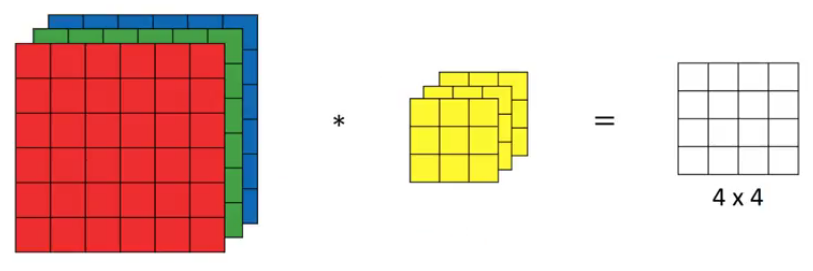
i)If the Input is a 2D Greyscale Image: We can generalize it and say that if the input is n X n and the filter size is f X f, then the output size will be (n-f+1) X (n-f+1):

* **Input:** n X n
* **Filter size:** f X f
* **Output after Convolution:** (n-f+1) X (n-f+1)

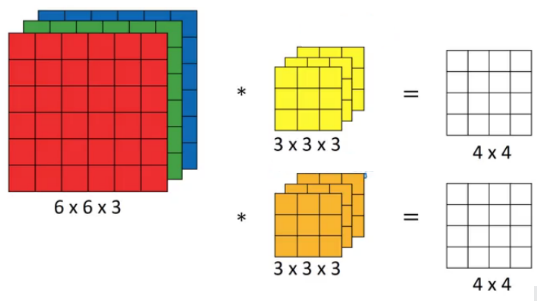
But,

ii) If the image is a 3D color image i.e. along its depth, it is composed of three (RGB) channels, then the output kernel height and width will be spatially small, but the depth extends up to all three channels. ***Keep in mind that the number of channels in the input and filter should be same.***

* **Input:** n X n X d (here, d= 3 for R,G,B)
* **Filter size:** f X f X d (d=3 for R,G,B)
* **Output after Convolution:** (n-f+1) X (n-f+1) X 1 (NOTE: …..here 1 is output depth or we can say that we get only 1 feature map at output b/c It is important to understand, that we don't simply have a 3x3 filter, but actually a 3x3x3 filter, as our input has 3 dimensions. And we learn 1 feature map from, convolving 3x3x3 filter over the input of nXnX3.)



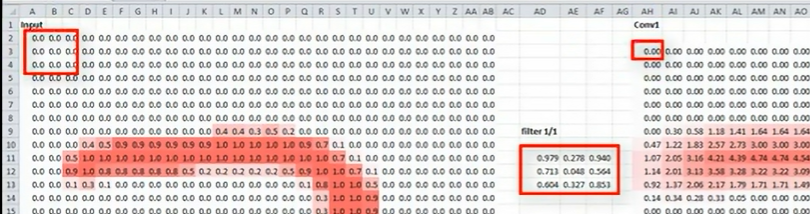
* **Note: Instead of using just a single filter (of 3X3X3), we can use multiple filters(ex. here we have two X (3X3X3) filters) as well**. For example, let’s say the first filter will detect vertical edges and the second filter will detect horizontal edges from the image. If we use multiple filters, the output dimension will change. So, instead of having a 4 X 4 output as in the above example, we would have a 4 X 4 X 2 output (if we have used 2 filters):



iii) If we have a 3d input image and also, we have used “Padding” operation( with amount of Padding = P) and “Strides” (with Strides = S) in our Convolution operation, then

* **Input:** n X n X d …………(here, d= 3 for R,G,B)
* **Filter size:** f X f X d ……….(d=3 for R,G,B)
* **Output after Convolution:**  X X d

So here our 3x3 filter is of random numbers. Here when the filter first lands on the first 3x3 block of input pixels (see below in red boxes on the left side), the dot product of the filter itself with the 3x3 block of input pixels from the input will be computed and stored (as a 1X1 matrix, see right portion of image in red box, value here = 0).

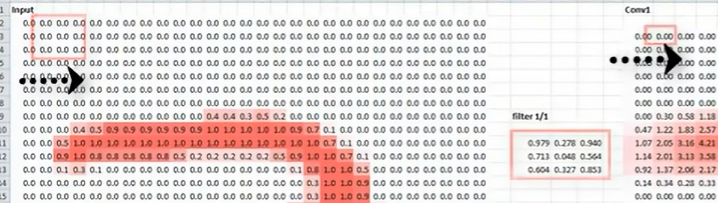


Dot product explained:



(……. continue)

This will occur for each 3x3 set of pixels that the filter convulses. (see below) i.e. we now we slide to the next 3x3 block take the dot product with the filter and then store the value in the output matrix (as shown below)…..and so on.



Types of Filters:

One type of quote pattern that a filter could detect could be edges in images. So this filter would be called an **edge detector**. For example some filters may detect corners some may detect circles other squares. Now these simple and kind of **geometric filters** are what we'd see at the start of our network.

The deeper our network goes the more sophisticated these filters become, so in later layers rather than edges in simple shapes, our filters may be able to detect specific objects like eyes ears hair or fur feathers scales and beaks even, and,

In even deeper layers the filters are able to take even more sophisticated objects like full dogs cats lizards and birds.

Example of Edge Detector Filters:

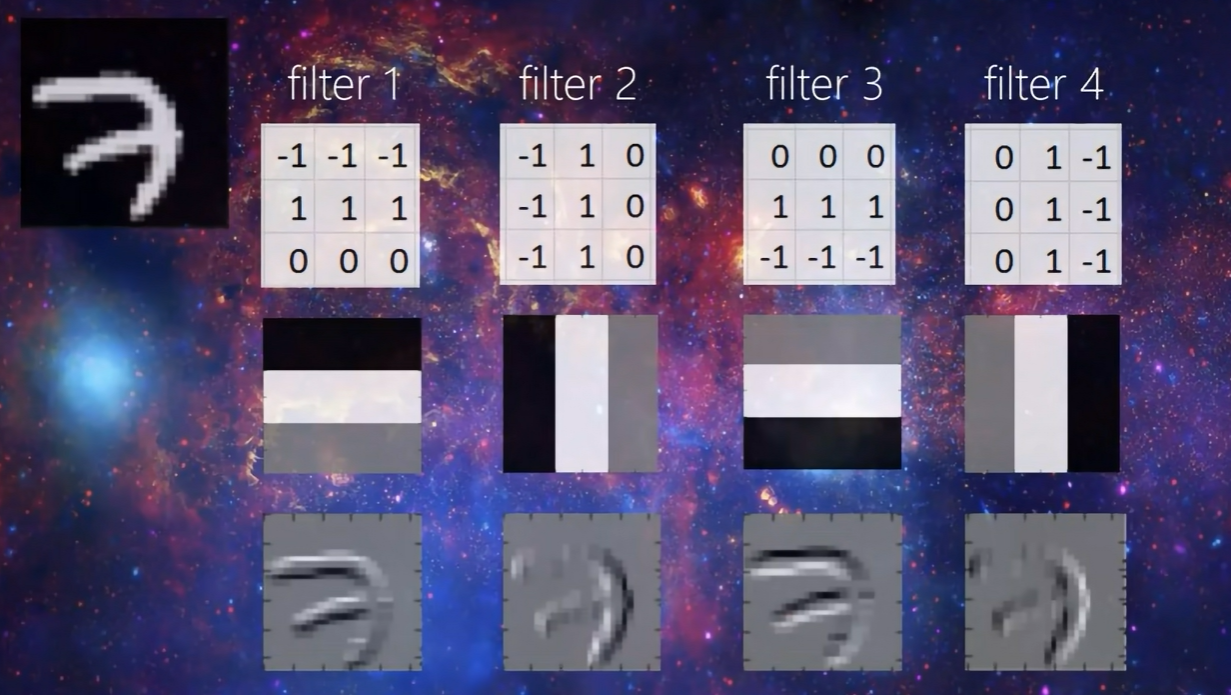


Fig: Upper corner 7 is the “Real Input Image with say nXn pixels”

Top Layer: Different Filters

Middle Layers: Interpretation of these Filters i.e.

-1 = black in output as

+1 = White

0 = Grey

Bottom Layers: Output after applying filter

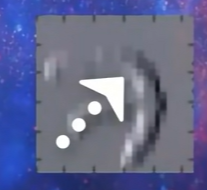
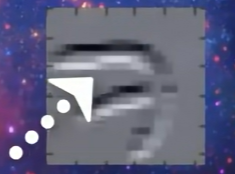
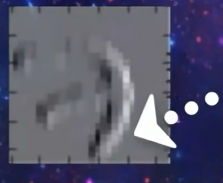
NOTE: All 4 of the filters are detecting “Edges”

1st filter detects : top horizontal edges of the input image 7. (see below)

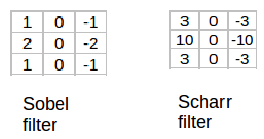
2nd filter detects : inner vertical edges of the input image 7. (see below)

3rd filter detects : bottom horizontal edges of the input image 7. (see below)

4th filter detects : outer vertical edges of the input image 7. (see below)

Some of the commonly used filters are:

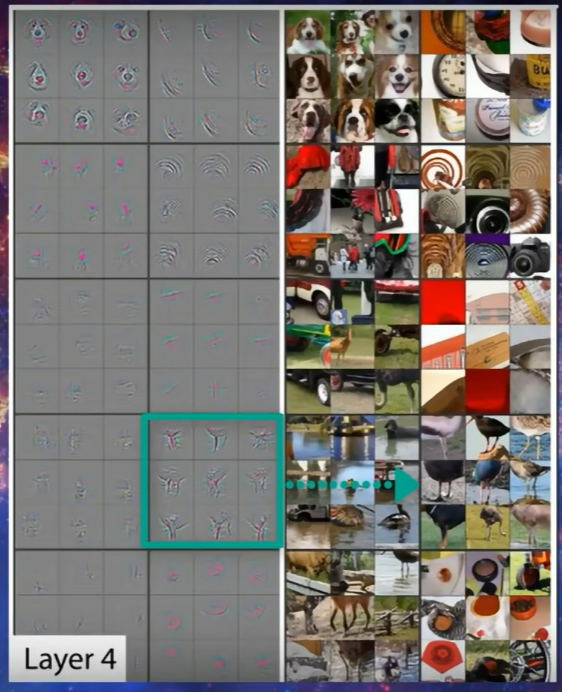


The Sobel filter puts a little bit more weight on the central pixels.

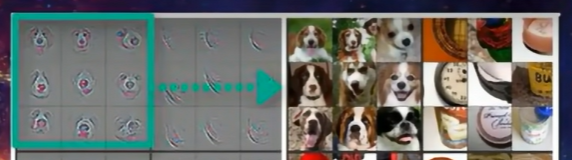
NOTE: Instead of using these filters, we can create our own as well and treat them as a parameter which the model will learn using backpropagation.

Other Examples:

1. Bird legs being detected by CNN



1. Dog faces detected by CNN

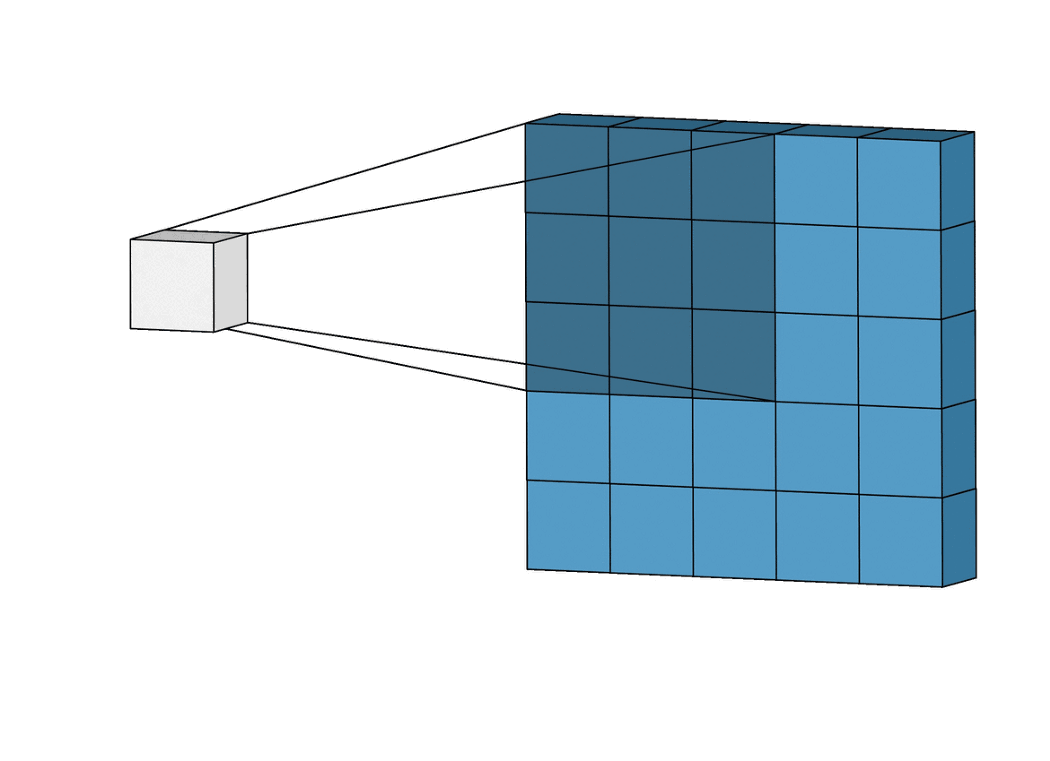


In  a convolutional network (ConvNet), there are basically three types of layers:

1. ***Convolution layer***
2. ***Pooling layer***
3. ***Fully connected layer, or dense layer, or ANN***

Let us understand them one by one.

1. **CONVOLUTION OPERATION AND LAYER:**



Challenges with Artificial Neural Network (ANN) and why we need CNN ?

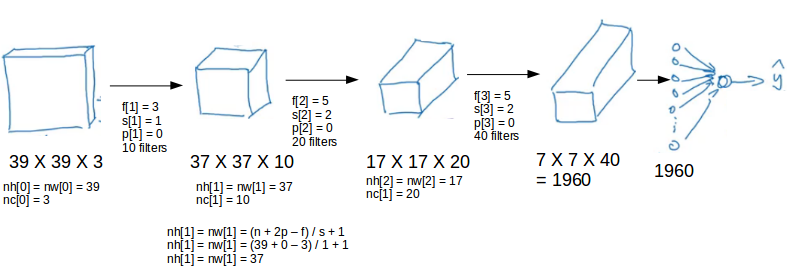
* While solving an image classification problem using ANN, the first step is to convert a 2-dimensional image into a 1-dimensional vector prior to training the model. This has two drawbacks:
  + The number of trainable parameters increases drastically with an increase in the size of the image. **This means huge quantities of calculations for derivatives and backpropogation of w and b**. On the other hand**, CNN uses filters and pooling layers which reduce the computations drastically**.
  + ANN loses the spatial features of an image. Spatial features refer to the arrangement of the pixels in an image. So**, if a test image is shifted to left or Right by even 2 or 3 pixels, the ANN (after the training has been done), will not be able to identify the same Image correctly**, Hence give wrong answer. On the other hand, **CNN has no such issues with it, because the filters/Kernels are passed over the entire image, hence shifting of the image has no impact on Convolution operation.**
  + ANN cannot capture sequential information in the input data which is required for dealing with sequence data

There are primarily three major advantages of using convolutional layers over using just fully connected layers(i.e. over an ANN or dense Neural Networks, all are same):

1. **Parameter sharing**: In convolutions, we share the parameters while convolving through the input. The intuition behind this is that a feature detector, which is helpful in one part of the image, is probably also useful in another part of the image. So a single filter is convolved over the entire input and hence the parameters are shared.
2. **Sparsity of connections**: The second advantage of convolution is the sparsity of connections i.e. NOT every node is connected to every other node (like it happens in ANN). For each layer, each output value depends on a small number of inputs, instead of taking into account all the inputs. b/c here we have a filter that at any instance takes into account only the small region over which it is placed.
3. **Location Invariant Feature detection**: Pooling + Convolution together helps us identify features wherever they are, irrespective of Location. For example: a Dog can be lying down or standing up, we would still be able to detect any feature, say its nose, in both the locations.

**Simple Convolutional Network Example:**

This is how a typical convolutional network looks like:



We take an input image (size = 39 X 39 X 3 in our case), convolve it with 10 filters of size 3 X 3 X 3, and take the stride as 1 and no padding. This will give us an output of 37 X 37 X 10. We convolve this output further and get an output of 7 X 7 X 40 as shown above. Once we get an output after convolving over the entire image using a filter, we add a bias term to those outputs save this output, called as a **Feature Map**. We can then chose to apply an RELU activation function (to introduce Non-Linearity) and then carry out Pooling operation(to reduce the size of input image so that there is less computation while training the Neurons during Classification stage using ANN, which comes after Feature Extraction………see below image)

Finally, we take all these numbers (7 X 7 X 40 = 1960, assuming No Pooling or RELU was used), **unroll them into a large single column vector (This process of converting a 2D array to 1D array is called Flatten operation) so that the image becomes suitable to be given as a input for Classification**, i.e. we pass them to the Basic Neural Network Classifier (also called as A.N.N.) for “Classification” that will make predictions. This is a microcosm of how a convolutional network works.

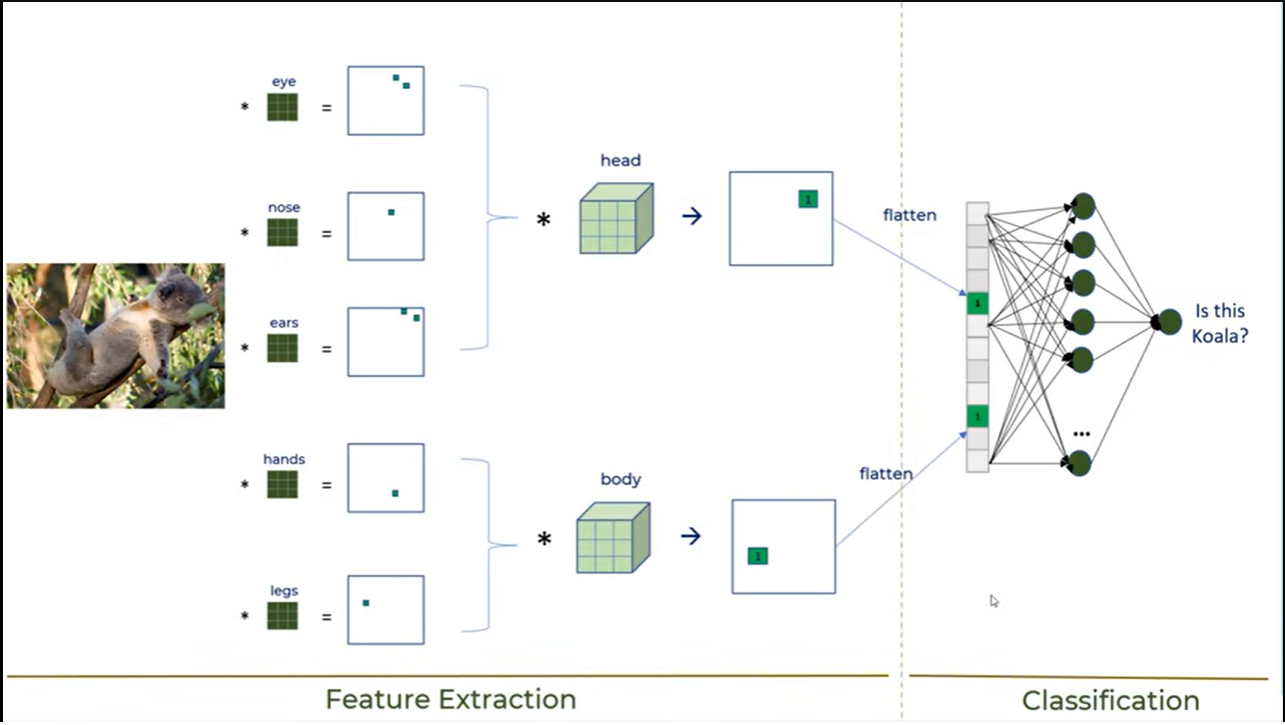


Fig: CNN without Pooling or RELU

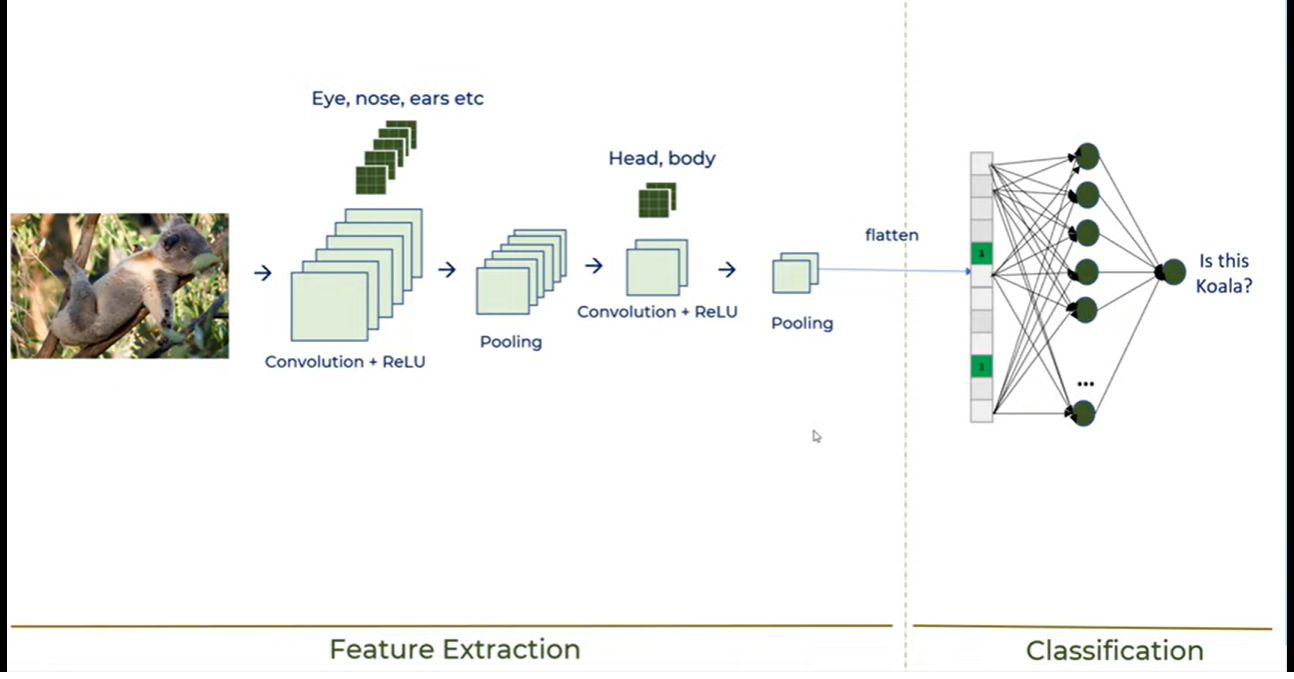
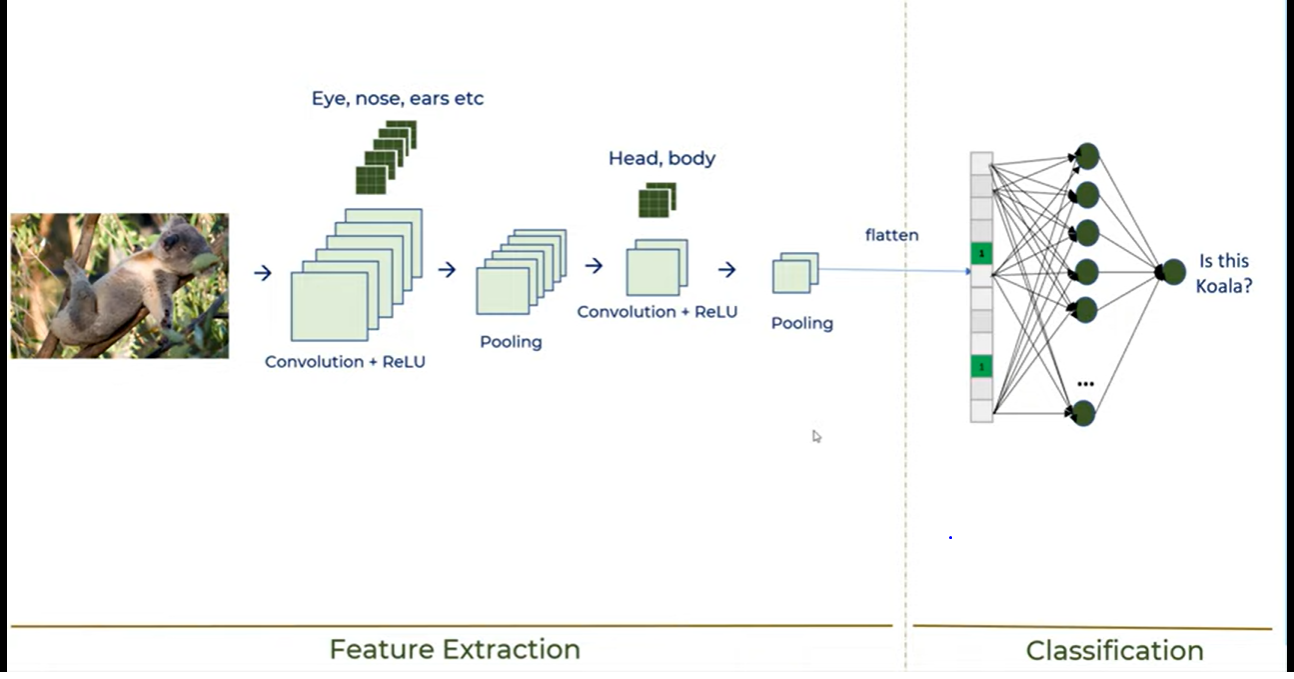
 

Fig: CNN with Pooling and RELU

**Padding (with Convolution)**

We have seen that convolving an input of 6 X 6 dimension with a 3 X 3 filter results in 4 X 4 output. We can generalize it and say that if the input is n X n and the filter size is f X f, then the output size will be (n-f+1) X (n-f+1):

There is a Primary Disadvantage of this i.e*. Pixels present in the corner of the image are used only a few number of times during convolution as compared to the central pixels. Hence, we do not focus too much on the corners since that can lead to information loss*

To overcome these issues, we can pad the image with an additional border, i.e., we add one pixel all around the edges. This means that the input will be an 8 X 8 matrix (instead of a 6 X 6 matrix). Applying convolution of 3 X 3 on it will result in an output of 6 X 6 matrix, which is what the original shape of the image was. This is where padding comes to the rescue.

There are two common choices for padding while coding:

1. **Valid:** It means No padding. If we are using valid padding, the output will be (n-f+1) X (n-f+1)
2. **Same:** Here, we apply padding so that the output size is the same as the input size, i.e.,  
   n+2p-f+1 = n  
   So, p = (f-1)/2

**Strids (in Convolution)**

Suppose we choose a stride of 2. So, while convoluting through the image, we will take two steps – both in the horizontal and vertical directions separately.

1. POOLING:

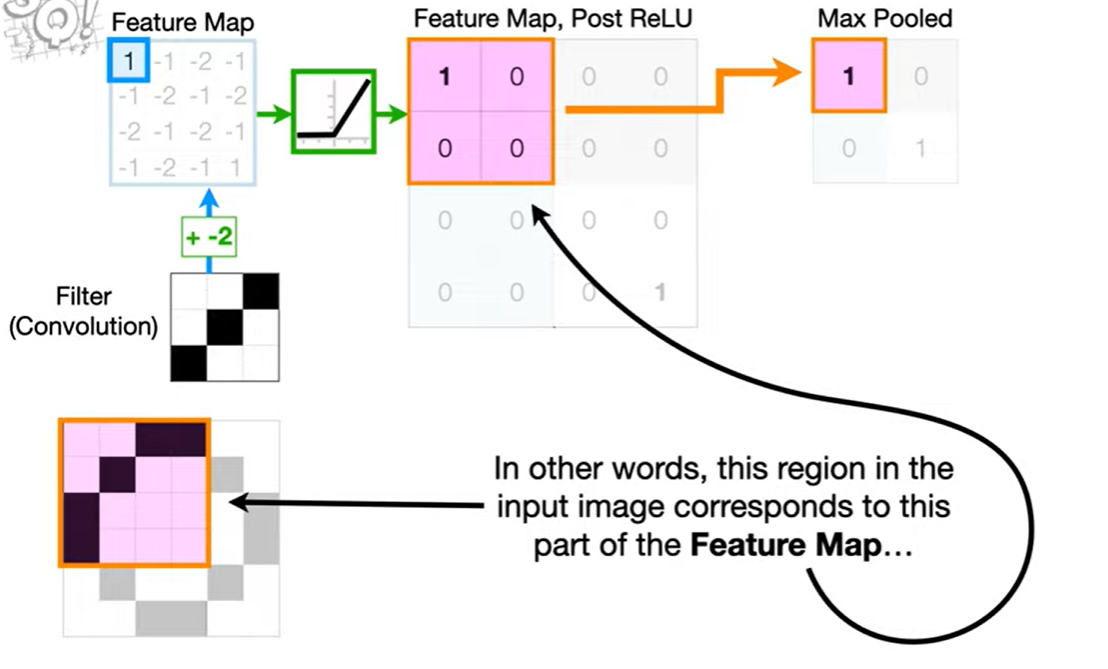
Pooling layers are generally used to reduce the size of the inputs and hence speed up the computation and also prevents Over fitting.

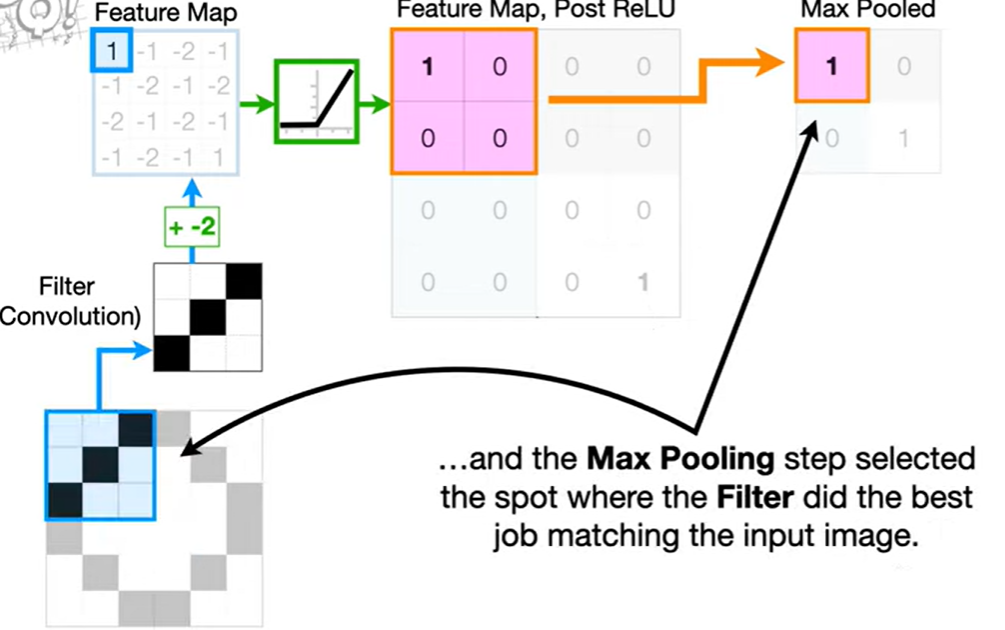
**Working and Interpretation of Pooling Layers:**

**Consider the Image Below.** The input image is 6X6 pixel of letter “O”. The used Filter detects 2 features of letter O, both at +45 degree to horizontal. (i.e features would typically lie in 2nd and 4th quadrant od letter “O”. This is reflected in the Output after the Max Pooling with value = 1.

This value of 1 in output after Max pooling, denotes that filter was able to pick up the desired feature (which that filter was designed to pick), in the 2nd and 4th quadrant of the input image.

In 2nd quadrant of Image, the feature was detected and picked up by the filter, which is what is reflected in the output matrix after Pooling as 1.



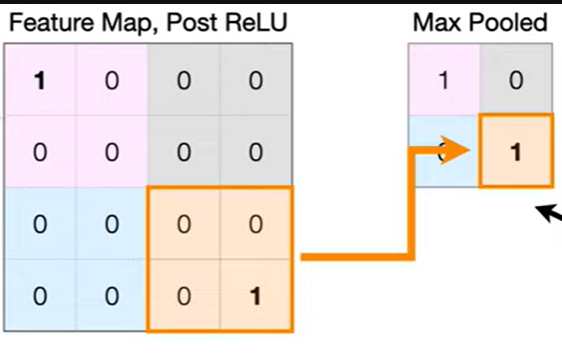
And 

Slly, at the other location (bottom portion in 4th quadrant of image) in letter “O”, again a perfect match of filter with input image was found, which is again denoted as 1 in the output matrix after pooling (see below).

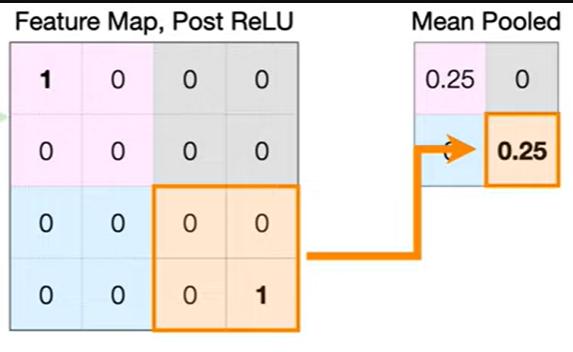


Two common types of pooling layers are **max pooling** and **average pooling.**

* 1. **Max Pooling (used 99% of the time):** Here, we simply pick the highest value from the “pooling filter” for every consecutive block. Ex: Below, we have a pooling filter of 2X2 and slide this filter with a stride of 2 and simply pick the highest value per block .



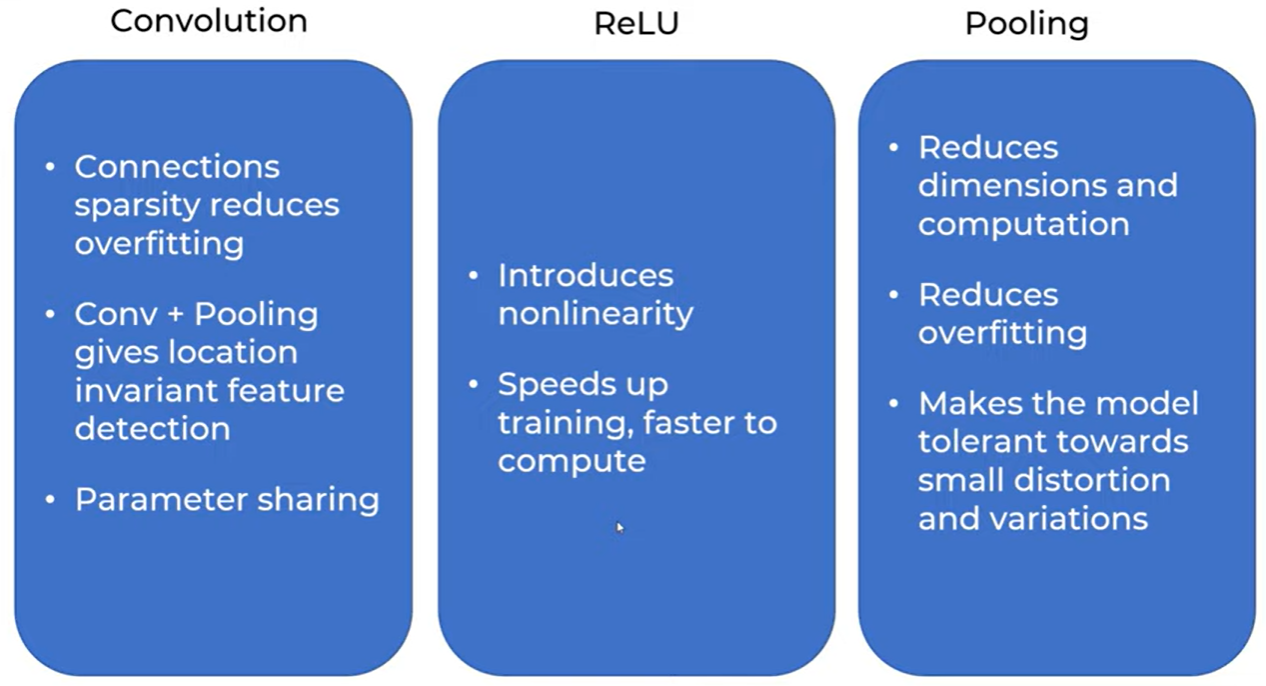
2. **Average Pooling:** Here instead of taking highest value in each block, we compute the average value and store that.



Benefits of Pooling:

* 1. Reduces Dimensions & Computations.
  2. Reduces Overfitting as there are less parameters
  3. Makes model more tolerant towards variations and distortions

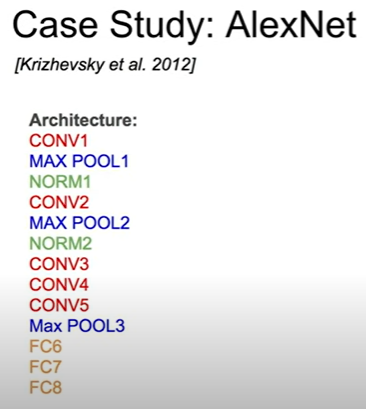
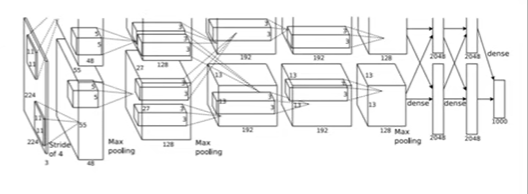
COMPARISON OF VARIOUS LAYERS USED IN CNN:

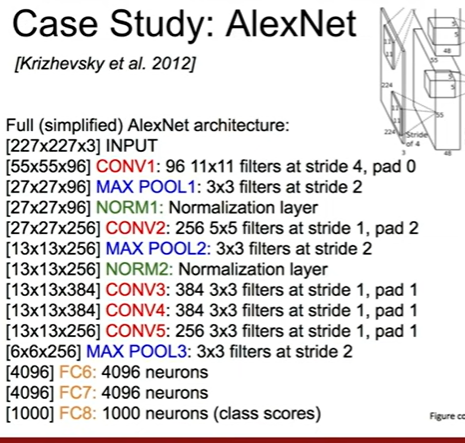
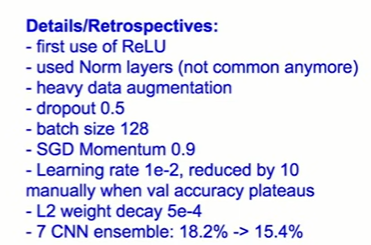




VARIOUS USE CASES : these are examples of CNN which use various Architectures

1.AlexNet: Successfully did Image Classification and outperformed all other image classification models prior to 2012.

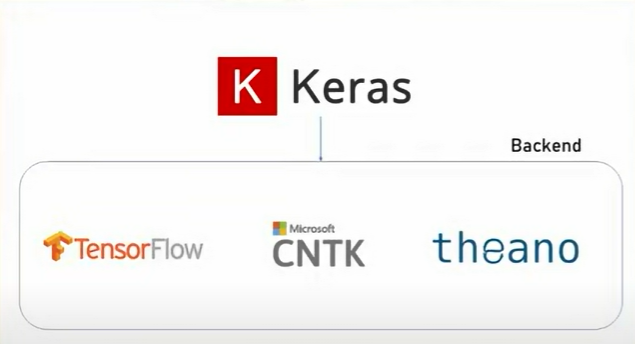
 

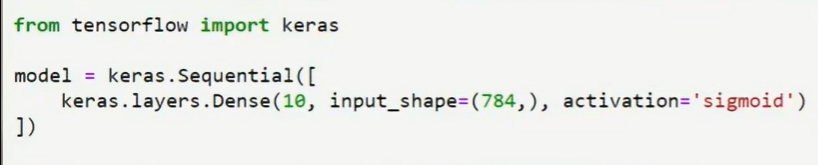
DL FRAMEWORKS:

1. Keras : It is a wrapper around TF, CNTK and Theano.ie. it uses all of these to create a framework and also make coding easy.

You can import Keras independently, or, from TensorFLow







2.