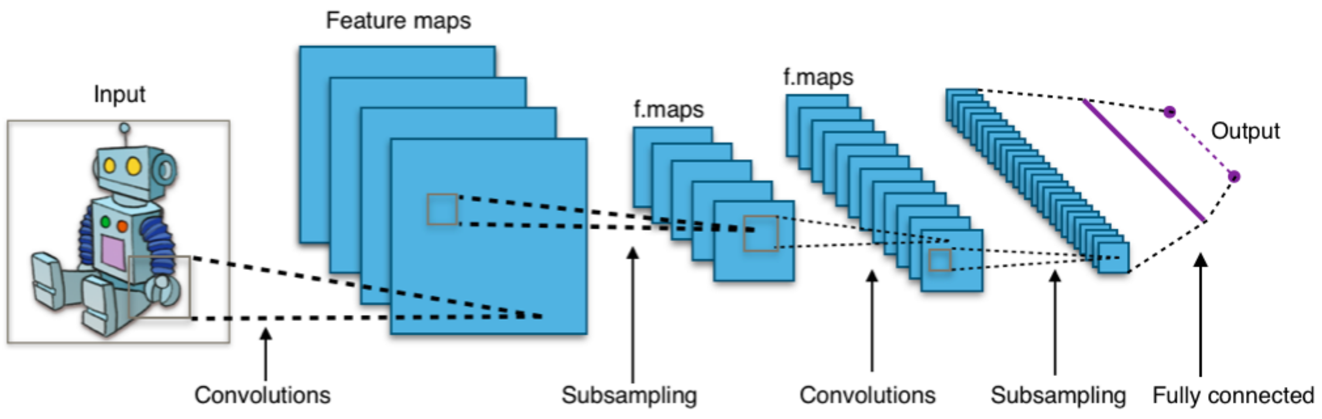
A Basic Introduction to Convolutional Neural Network

Himadri Sankar Chatterjee Jul 16, 2019



With the world of programmers and scientists going crazy over artificial intelligence, it's becoming a need of the hour to stay updated on the various research on deep learning that is helping in faster growth of AI. This post is about one of the deep learning model successfully applied in the image recognition industry, over the years. In 1995, **Yann LeCun** and **Yoshua Bengio** introduced the concept of Convolutional Neural Networks.

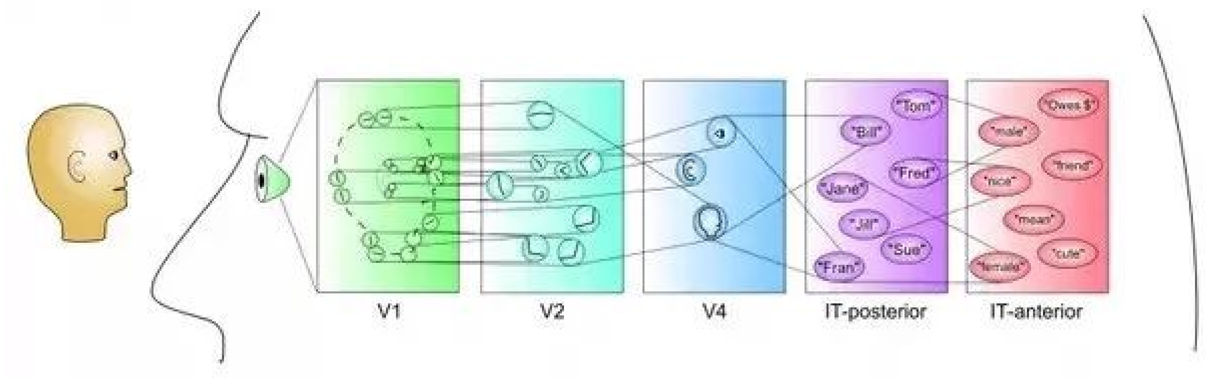
Yann LeCun and Yoshua Bengio

As a sort of formal definition, "**Convolutional Neural Networks** or **CNNs**, are a special kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called **convolution**. Convolution is a specialized kind of linear operation…………. *Blah BlahBlah!!!*" Was too much to take in at once, but, believe me, it will get easy as we go in the details.

CNNs are very **similar to ordinary Neural Networks** — they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. Now for those who have a general idea on how the neural network works, it will be easy to visualize. For the others who don't know anything about Simple Neural Network, it will be better if you could at least read about the basic working from some resource.

Background:

So, what exactly led to the idea for its development? **CNNs** are biologically-inspired models inspired by research of **D. H. Hubel** and **T. N. Wiesel**. They proposed an explanation for the way in which mammals visually perceive the world around them using a layered architecture of neurons in the brain, and this in turn inspired engineers to attempt to develop similar pattern recognition mechanisms in computer vision.
In their hypothesis, within the visual cortex, complex functional responses generated by "complex cells" are constructed from more simplistic responses from "simple cells'. For instances, simple cells would respond to oriented edges etc, while complex cells will also respond to oriented edges but with a degree of spatial invariances.



How humans see and recognizes objects.

Receptive fields exist for cells, where a cell responds to a summation of inputs from other local cells. The architecture of deep convolutional neural networks was inspired by the ideas mentioned above:

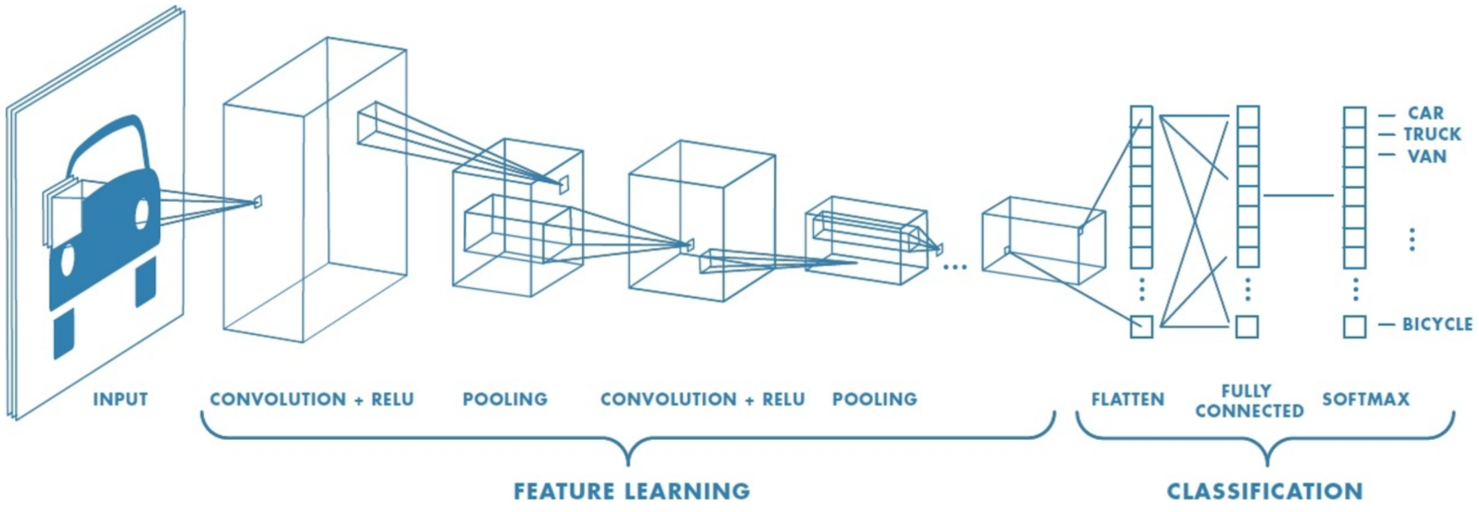
1. **local connections**

2. **layering**

3. **spatial invariance** (shifting the input signal) results in an equally shifted output signal, most of us can recognize specific faces under a variety of conditions because we learn abstraction. These abstractions are thus invariant to size, contrast, rotation, orientation

As an example, let's consider a car. How does a human recognize that it is a car? We basically search for the characteristics that are unique to a car. We look for wheels, head-lights, doors, rear trunk, glass windows, hood and other features that differ it from other modes of transport. Similarly, while recognizing a wheel, we look for circular-shaped objects, comparatively dark colored with a rough texture, positioned below the main structure of the car. We take into account all the little details that together constitute to form some basic information. These little informations together bunch up to form a particular characteristic that is unique to an object that we are recognizing.

Layout of the Components:



The basic layers of CNN.

"*A simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function*." What it actually means is that, each layer is associated with converting the information from the values, available in the previous layers, into some more complex information and pass on to the next layers for further generalization.
The CNN is a combination of two basic building blocks:

1. **The Convolution Block** — Consists of the Convolution Layer and the Pooling Layer. This layer forms the essential component of *Feature-Extraction*

2. **The Fully Connected Block** — Consists of a fully connected simple neural network architecture. This layer performs the task of *Classification* based on the input from the convolutional block.

We shall define each of these layers now. People who have a little knowledge of image processing might find it easy to grasp, however others can find it easy too.

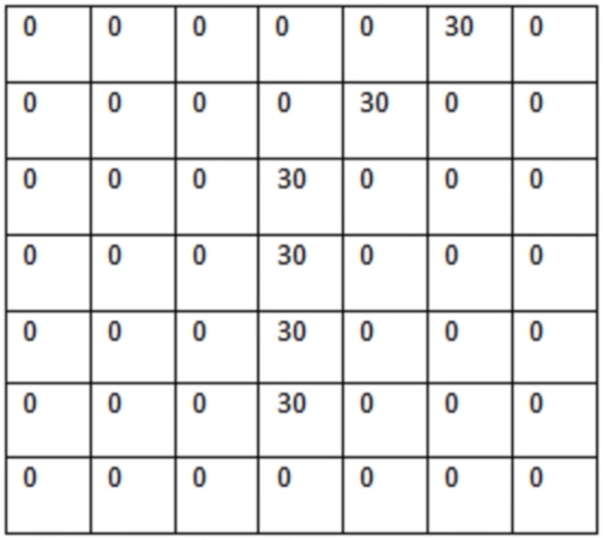
Defining the layers:

We shall describe the working theory of each of the layers.

CONVOLUTIONAL LAYER:

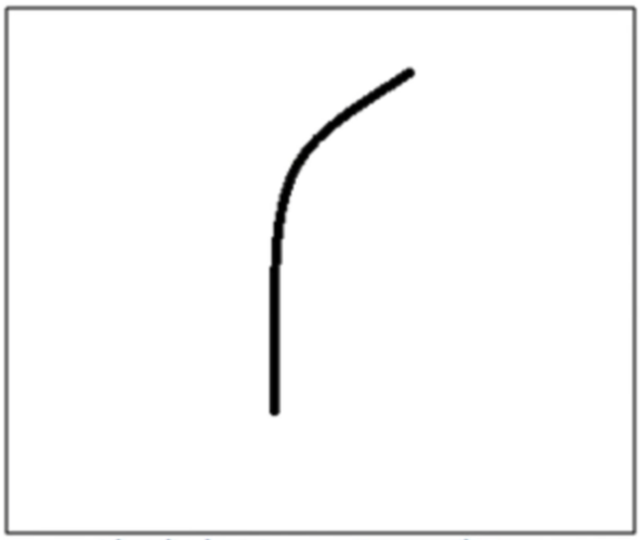
The **CONVOLUTIONAL LAYER is related to feature extraction**. First let us get clear of the idea of '*filters*' and '*convolution*', then, we shall move on to its implementation in the layer.

Filters: Filters or 'kernels' are also an image that depict a particular feature. For example, let us take the picture of this curve. We take this as a sample feature that we will try to recognize, i.e., determine whether it is present in an image.



0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



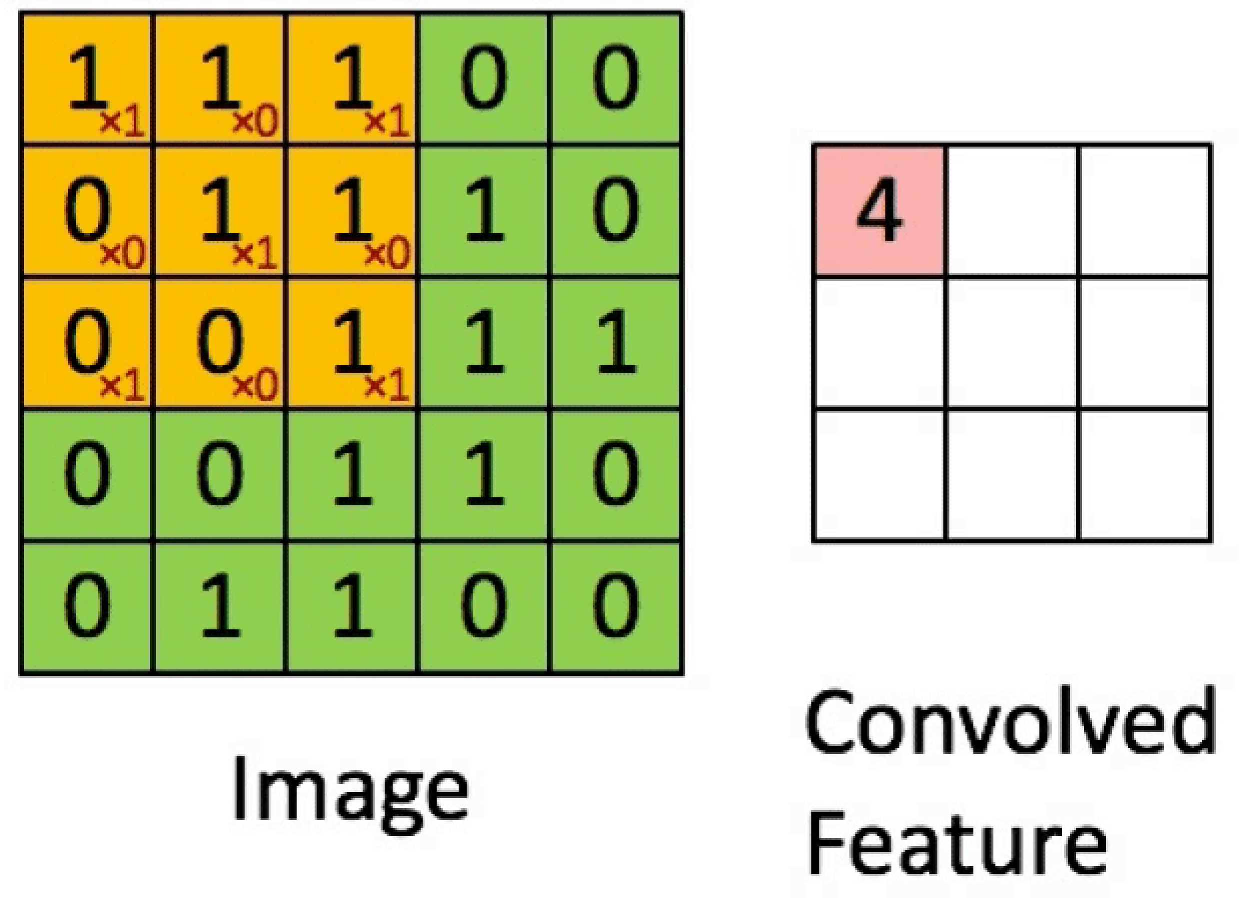
Visualization of a curve detector filter

An example of a simple filter depicting a curved line.

Convolution: It is a special operation applied on a particular matrix (, usually the image matrix) using another matrix (, usually the filter-matrix). The operation involves **multiplying the values of a cell corresponding to a particular row and column, of the image matrix, with the value of the corresponding cell in the filter matrix. We do this for the values of all the cells within the span of the filter matrix and add them together to form an output.**

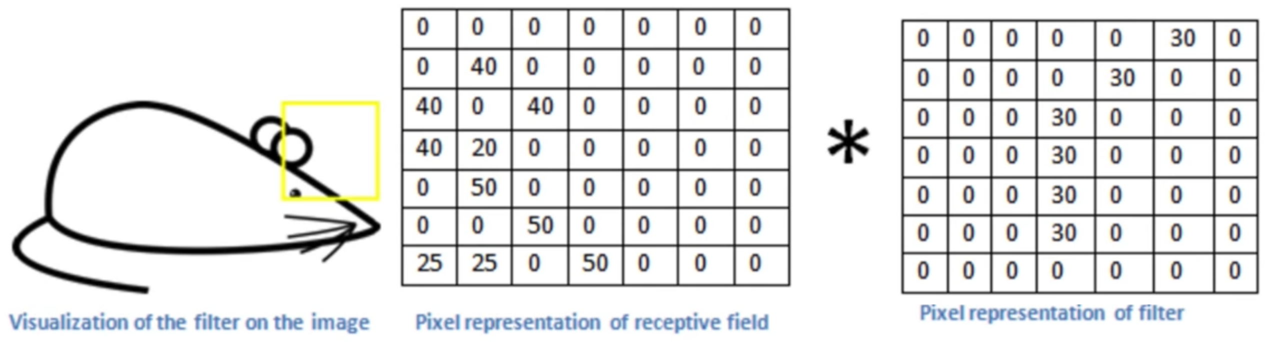
For example, A (part of the image matrix) = [2 5 17] and B (part of the filter matrix) = [1 0 1]. Then the answer of A *(convolve) B = [2*1 + 5*0 + 17*1] = [2 + 17] = [19]. This is just an example. In practice both of the matrix are 2-D, but the sense of the

operation remains the same.
This will explain the operation nicely. You can see, how the filter-matrix spans over the whole matrix, performing the operation at each step and generating the desired output-matrix:



Depicts how convolution works.

So, now, how do we find a particular feature? We simply convolve the ‘filter-matrix’ over the image matrix and constitute another matrix that contain some values. Now, what are these values? Let’s get back to our filter image, we considered before and take the image of a simple mouse.


*
Visualization of the filter on the image
Pixel representation of receptive field
Pixel representation of filter

Pixel representation of receptive field

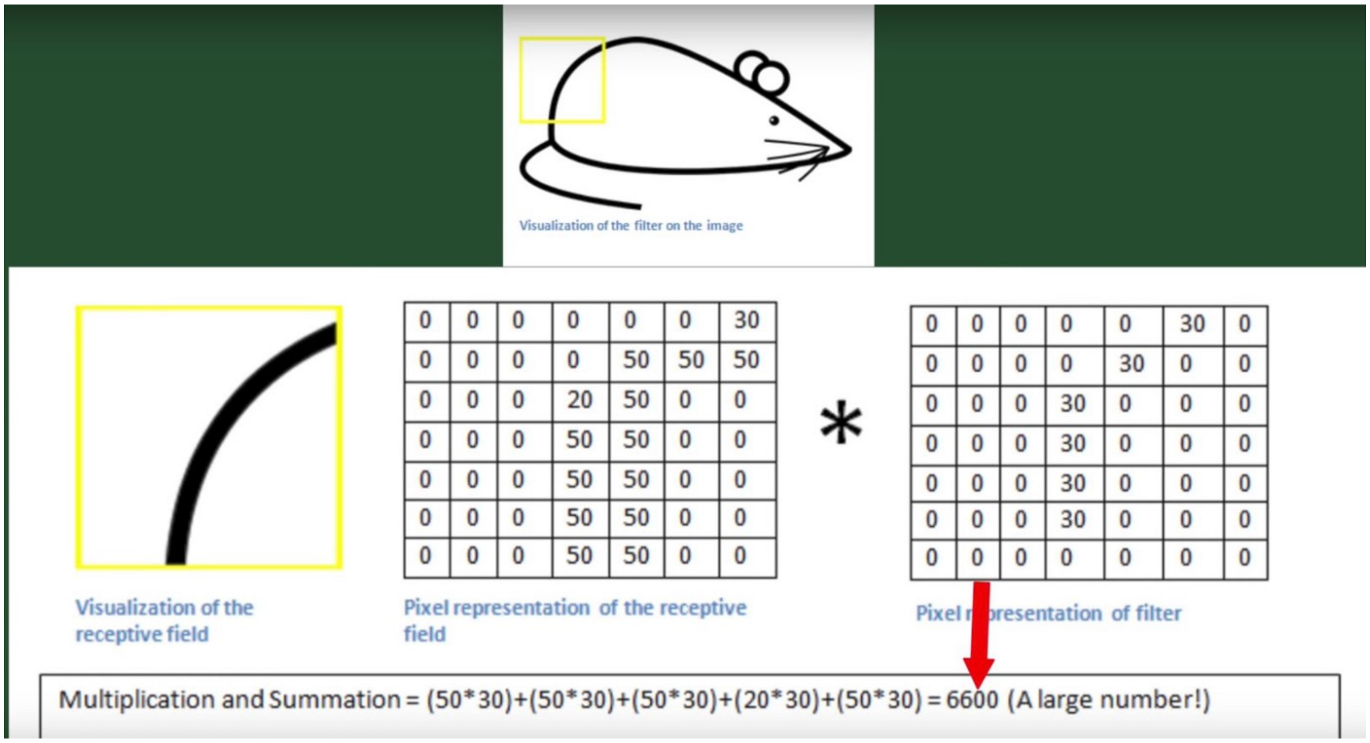
0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Multiplication and Summation = 0

Image 1: An example with result 0, thus confirming absence of the feature.


Visualization of the filter on the image
Visualization of the receptive field
Pixel representation of the receptive field
*
Pixel r presentation of filter

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Image 2: An example with a large value as result, thus confirming the presence of the filter

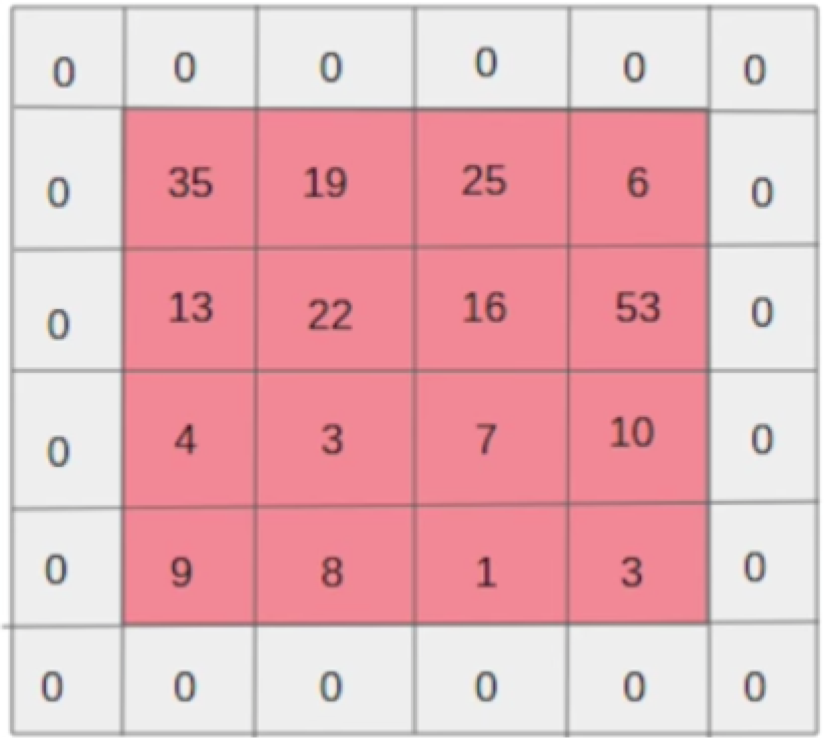
Notice that the rear of the mouse is of a similar shape. So, we convolve the filter within that region of the image-matrix. We also consider another region of the image that does not possess the same curve-like feature. We also perform the operation on that section of the image using the same filter-matrix. Now, take a look at the values of the corresponding operation. **(Do not get confused at the matrix representation of the filer image and that, depicting a part of the mouse image.)**

Image 1 depicts a very large value, while Image 2 results to 0. Thus, **the basic intuition to be taken from here, is that, if the feature is present in a particular part of the image, it returns a very large value on convolution, while for the other places, it would return a small value, depicting that the feature is not present.** Now, that we have a clear idea of both, we shall apply them in feature extraction. We have as our input say a colored image, i.e. a 2-D RGB image matrix. We also have no. of filter-matrix, which are 2-D matrix depicting some characteristics. So**, we take each filter-matrix, convolve it over the corresponding part of the image-matrix of all the red, green**

and blue channel-matrices and add the values form each channel together to form the value of cell of the output-matrix.

In doing so, we are essentially trying to find out whether a particular feature is present in the image we are trying to recognize. Now, there are a few other operations required to be performed on the image during convolution.

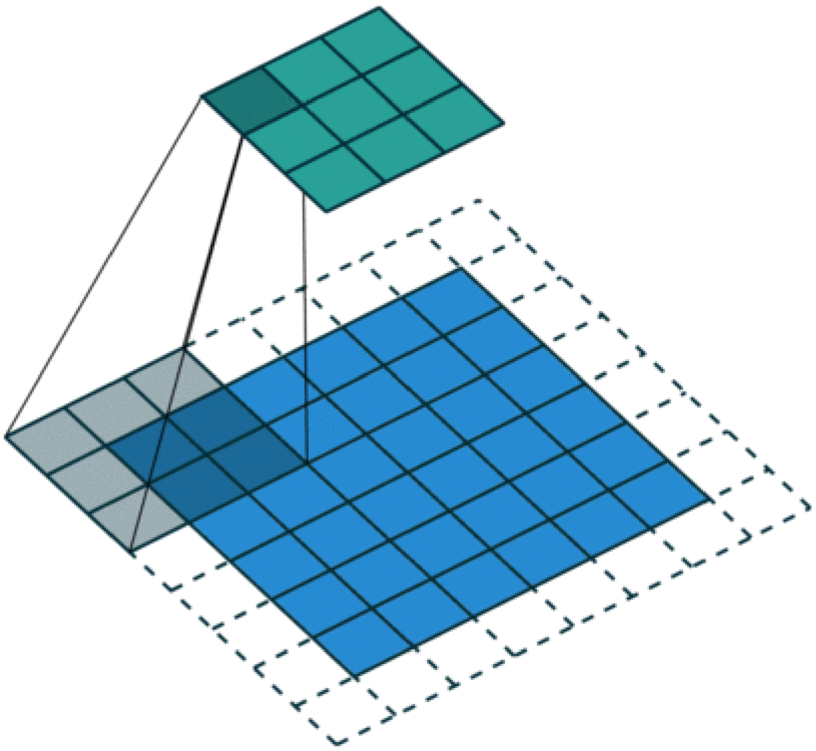
- **Padding:** If we try to visualize the operation of convolution, in our head, as the filter matrix moves over the whole image, we find that the no of times, the values of the cells lying within the matrix is considered for the operation is more than the no. of times, the values of the cells in the corners or at the borders, are accounted for. This implies that the values at the corners or around the borders are not being given equal weightage. To overcome this, we add another row and column, of only 0, at all the sides of the image matrix. This idea is known as padding. In actual sense, these values being '0' wouldn't supply any extra information, but will help into accounting the previously less-accounted for values to be given more weightage.



0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

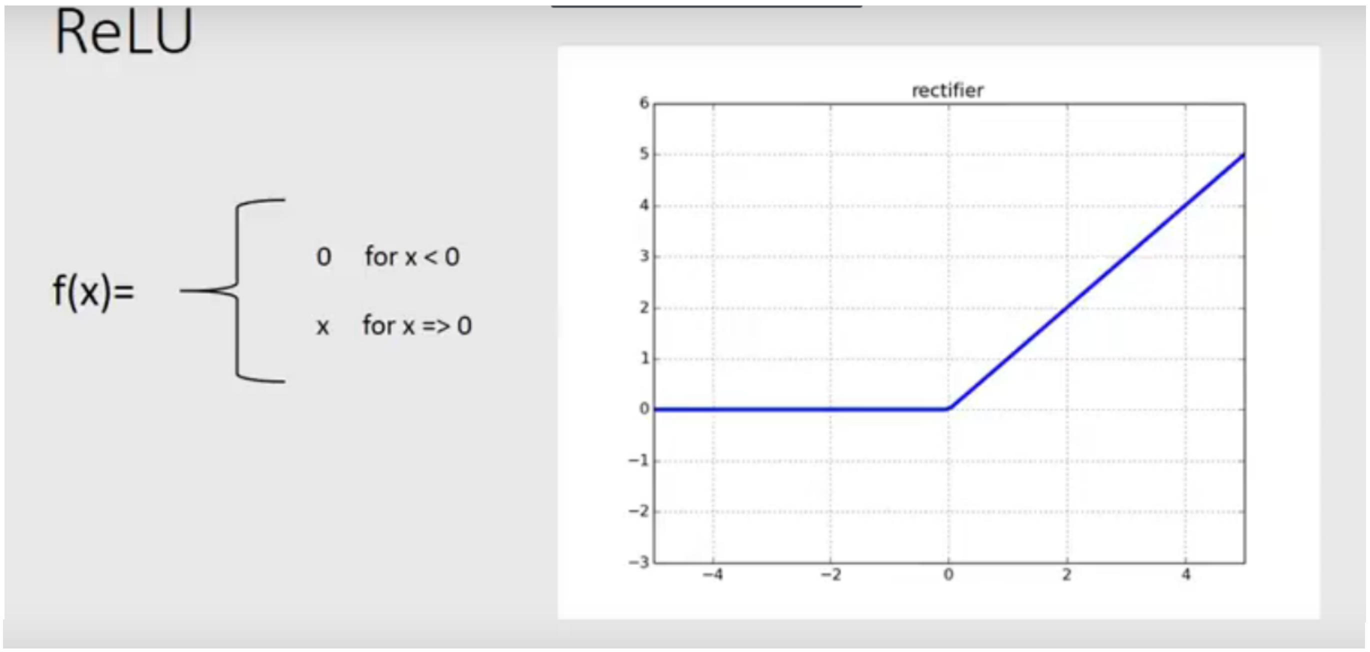
A padding of 0's around the actual matrix.

- **Striding:** In 'strided' convolution, instead of shifting the filter one-row or one-column at a time, we shift it, maybe, 2 or 3 rows or columns, each time. This is generally done to reduce the no of calculation and also reduce the size of the output matrix. For large image, this doesn't results in loss of data, but reduces computation cost on a large scale.



This implementation with stride 2 shows the particular cells that will be convolved.

- **RELU Activation:** *RELU or Rectified Linear Unit* is applied on all the cells of all the output-matrix. The function is defined as:

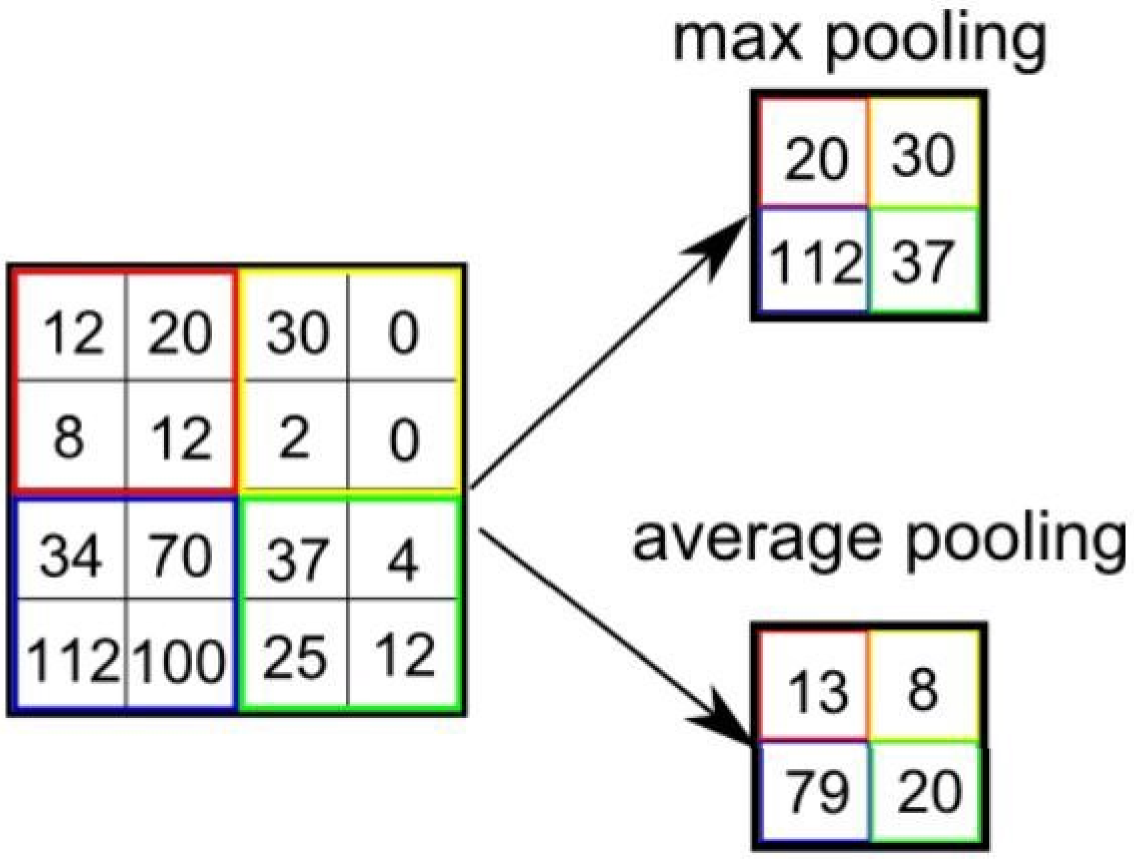


Graph of the RELU function

The basic intuition to derive from here is that, after convolution, if a particular convolution function results in ‘0’ or a negative value, it implies that the feature is not present there and we denote it by ‘0’, and for all the other cases we keep the value. Together with all the operations and the functions applied on the input image, we form the first part of the Convolutional Block.

POOLING LAYER:

The Pooling layer consist of performing the process of extracting a particular value from a set of values, usually the max value or the average value of all the values. This reduces the size of the output matrix. For example, for MAX-POOLING, **we take in the max value among all the values of say a 2 X 2 part of the matrix. Thus, we are actually taking in the values denoting the presence of a feature in that section of the image. In this way we are getting rid of unwanted information regarding the presence of a feature in a particular portion of the image and considering only what is required to know**. It is common to periodically insert a Pooling layer in-between successive convolutional blocks in a CNN architecture. Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network.



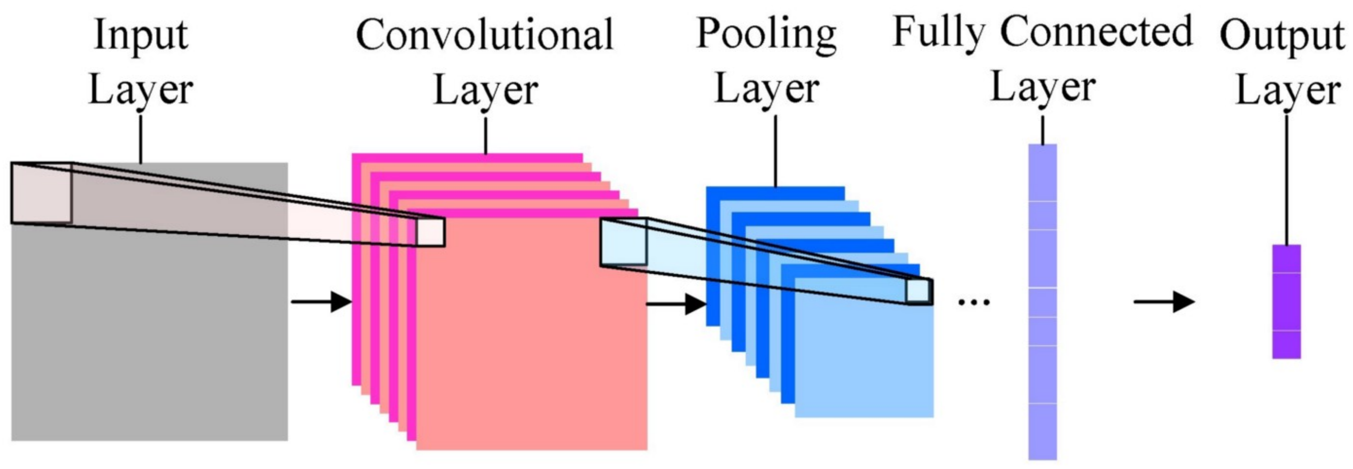
An example of both Max-Pooling and Average-Pooling

Together with the **CONVOLUTIONAL LAYER** and the **POOLING LAYER**, we form the **CONVOLUTIONAL BLOCK** of the CNN architecture. Generally, a simple CNN architecture constitutes of a minimum of three of these Convolutional Block, that performs feature extraction at various levels.

FULLY CONNECTED LAYER:

This layer forms the last block of the CNN architecture, related to the task of classification. This is essentially a Fully connected Simple Neural Network, consisting of two or three hidden layers and an output layer generally implemented using 'Softmax Regression', that performs the work of classification among a large no of categories. Hope you can understand the basic idea of the regression analysis from the link.

BRINGING EVERYTHING TOGETHER:



Stacking the concepts under one roof

So, what happens from the beginning to the end of the CNN.

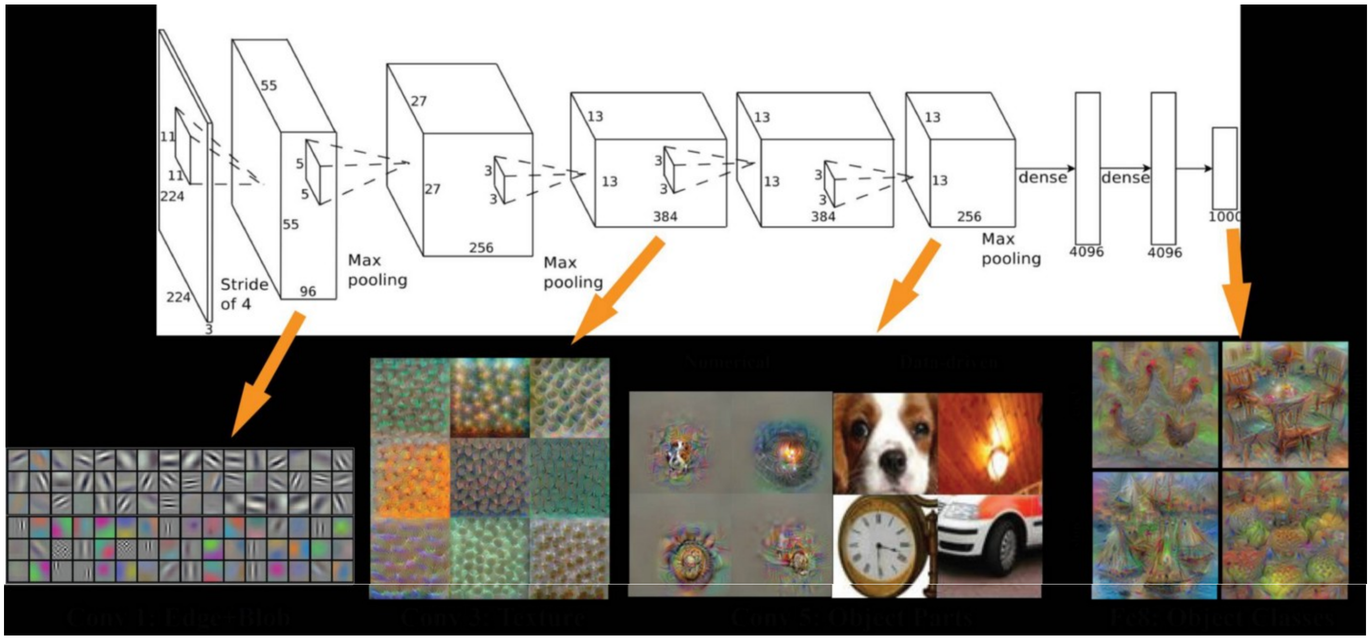
1. We give input an RGB image. It is generally a 2-D matrix defined for the 3 color channels. Let each channel be of size n X n. Thus, the input is a n X n X 3 dimension matrix.
2. We have a 3-D matrix, consisting of (say) 'k' no. of filters of a size (say f X f).

3. We perform PADDING on the image of say ‘p’ rows & columns. Thus, the input-matrix becomes (n+2*p) X (n+2*p) X 3 dimensions.
4. Next, we perform the strided convolution operation of the filter-matrices on the input image-matrices, as described before, using a stride of say ‘s’. Thus, the output matrix becomes
5. We perform POOLING over the output-matrix of each layer. The dimension of the output-matrices depends of the size of the pooling-filter and the stride length we have defined.
6. We perform the same operation from step 3–5, nearly three time.
7. On receiving the output-matrices of some dimension say a X b X *l*, we flatten the output into a 1-D array, i.e., we arrange all the values from the matrices sequentially in an array which forms the input matrix for the Fully — Connected Neural Network.
8. This neural network performs the desired calculation and gives the result.

BACKPROPAGATION:

So, a general question remains regarding defining the matrix for each filter. This is not predefined. The CNN is trained over a larger no of image during the training phase and for each time, the error generated is fed back into to CNN to adjust the values of the matrices in each layer. The basic working is similar to that of the training of a Simple Neural Network. The concept is **BACKPROPAGATION.** Since, this article deals with only the basics, I shall not delve into the mathematical background of backpropagation and the adjustment of values. May be, this topic can be covered in some other post.

WHAT WE SAW HAPPENING:



The visualization obtained at each layers in all the levels.

This image is of the various features at each Convolutional Block of a fully trained CNN. The basic thing to understand from this image is that, the filters at various layers are adjusted to determine various features of the image. The filters at the beginning captures some basic edges or blobs. The filters in the next layer considers various textures. The filters in the deeper layers are able to recognize various object parts that constitute an object. In the end, the Fully Connected Layer is trained to classify the image, based on the information gathered, into various object classes.

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. For example, 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 dog.

The convolution layers are the main powerhouse of a CNN model. Automatically detecting meaningful features given only an image and a label is not an easy task. The convolution layers learn such complex features by building on top of each other. The first layers detect edges, the next layers combine them to detect shapes, to following layers merge this information to infer that this is a nose. To be clear, the CNN doesn't know what a nose is. By seeing a lot of them in images, it learns to detect that as a feature. The fully connected layers learn how to use these features produced by convolutions in order to correctly classify the images.

WHY USE IT AT ALL:

If you think carefully, for an color image of size 256 X 256 X 3(for each color channel), the size of the input layer of a simple neural network would be 196,608 neurons, each connected to a large no. of neurons in the hidden layers. This increases the size of the stored data and the overall computation cost. You can very well image the size of the matrix required to store the weights for the connections of each neuron of one layer to the neurons of the next layer. This implementation is infeasible compared to the capability of the computers used for day-to-day use. Moreover, CNN works in close proximity to the idea of how humans **"might"** visualize and recognize objects. There is re-usability of the filters that helps us confirm the presence of features throughout the image. This is in contrast to the working of a Simple NN, that does nothing of these sort. The various operations like striding, pooling reduces the total no of parameters to be accounted for. This decrease is nearly a thousand times more effective in reducing the data space and computation cost of that of a Simple NN.