INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY

HYDERABAD

CSE578:
Computer Vision

Professor : Anoop Namboodiri
Notes By : Sai Manaswini Reddy I

2021

CNN

1D convolution :

Convolution is an important operation in signal and image processing. Convolution operates on two signals (in 1D) or two images (in 2D): you can think of one as the “input” signal (feature vector) (or image), and the other (called the kernel) as a “filter” on the input image, producing an output image (feature map) (so convolution takes two images as input and produces a third as output). Convolution is an incredibly important concept in many areas of math and engineering (including computer vision, as we’ll see later).

Definition :

Let’s start with 1D convolution (a 1D “image,” is also known as a signal, and can be represented by a regular 1D vector in Matlab). Let’s call our input vector f and our kernel g, and say that f has length n, and g has length m. The convolution f ∗ g of f and g is defined as:

$$(f * g)(i) = \sum_{j=1}^{\cdot} g(j) \cdot f(i - j + m/2)$$

One way to think of this operation is that we’re sliding the kernel over the input image. For each position of the kernel, we multiply the overlapping values of the kernel and image together, and add up the results. This sum of products will be the value of the output image at the point in the input image where the kernel is centered. Let’s look at a simple example.

Suppose our input 1D image is: f =

10	50	60	10	20	40	30

and our kernel is: g =

1/3	1/3	1/3

Let’s call the output image h. What is the value of h(3)? To compute this, we slide the kernel so that it is centered around f(3) :

10	50	60	10	20	40	30
	1/3	1/3	1/3			

For now, we’ll assume that the value of the input image and kernel is 0 everywhere outside the boundary, so we could rewrite this as:

10	50	60	10	20	40	30
0	1/3	1/3	1/3	0	0	0

We now multiply the corresponding (lined-up) values of f and g, then add up the products. Most of these products will be 0, except for at the three non-zero entries of the kernel. So the product is:

$$50*(⅓) + 60*(⅓) + 10*(⅓) = 40$$

Thus, h(3) = 40. From the above, it should be clear that what this kernel is doing is computing a windowed average of the image, i.e., replacing each entry with the average of that entry and its left and right neighbor.

Using this intuition, we can compute the other values of h as well : h =

20	40	40	30	20	30	23.33

We can shift the filter to the right by any units. Size of output Feature map is

(input size - filter size + 2*padding)/(stride) + 1

Size of output feature map depends on increment in j :

If j = 3 :

Feature vector : 12 × 1

Filter : 3 × 1

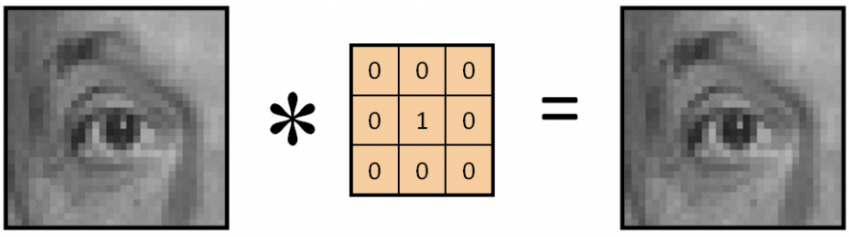
Feature Map : 4 × 1

(Remember that we're assuming all entries outside the image boundaries are 0—this is important because when we slide the kernel to the edges of the image, the kernel will spill out over the image boundaries).

2D Convolution :

$$C(j,k) = \sum_{p}\sum_{q} A(p,q)B(j-p+1,k-q+1)$$

We can also apply convolution in 2D—our images and kernels are now 2D functions (or matrices in Matlab; we'll stick with intensity images for now, and leave color for another time). For 2D convolution, just as before, we slide the kernel over each pixel of the image, multiply the corresponding entries of the input image and kernel, and add them up—the result is the new value of the image. Let's see the result of convolving an image with some example kernels. We'll use this image as our input: One very simple kernel is just a single pixel with a value of 1. This is the identity kernel, and leaves the image unchanged:



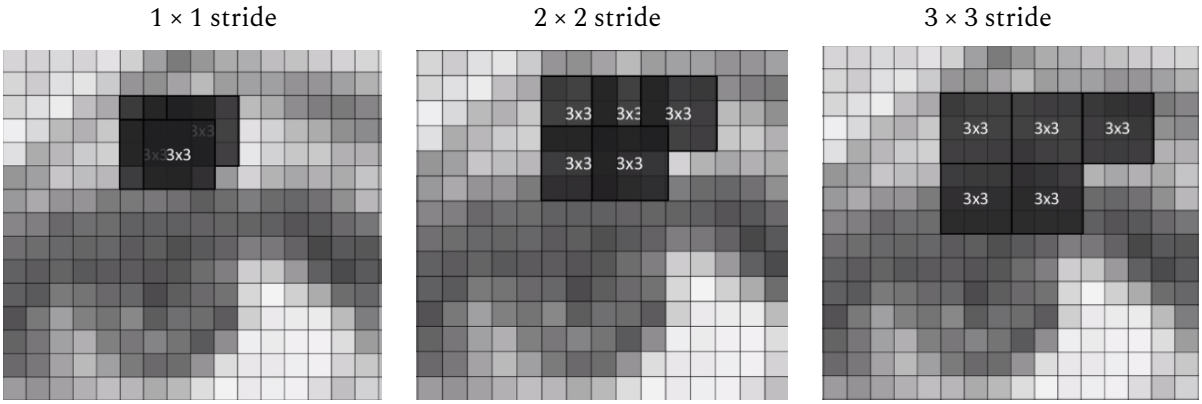
Original Identical image

This is, we are *building* **another image** by applying the filter to our input image. Note that depending on the filter we apply (its shape and values), we will get a different image.

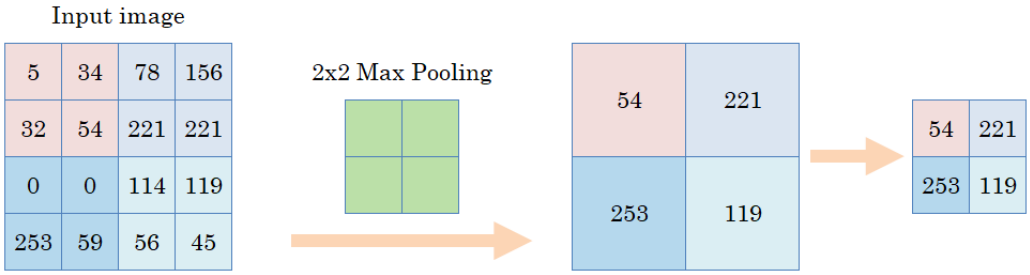
We usually use filters to modify the image or perform some tasks like :

- Edge detection
- Blurring
- Image sharpening
- Noise removal

Stride :

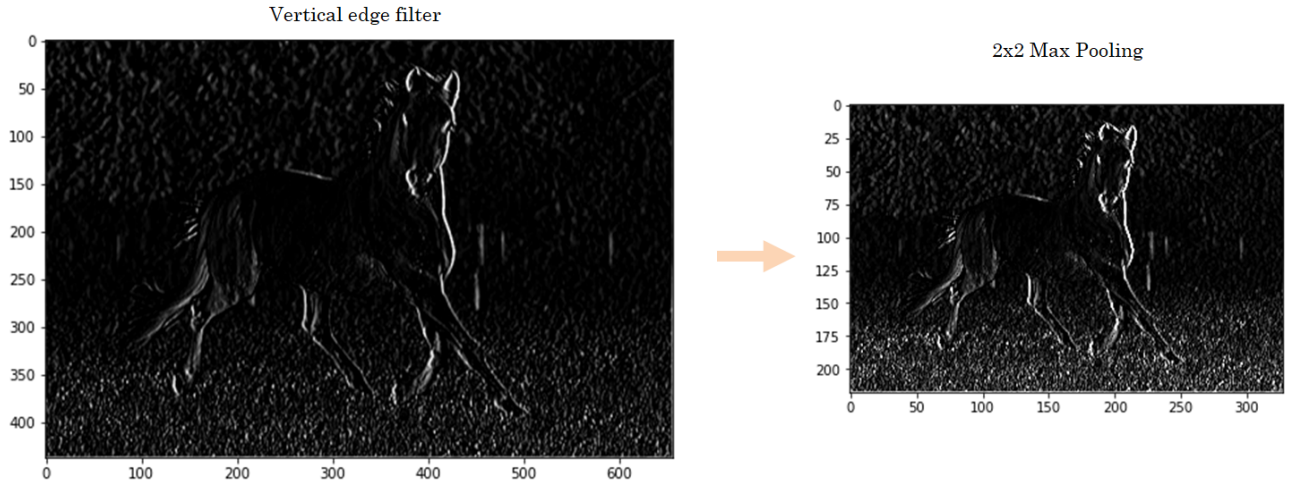


Pooling :



This is, we are going to take groups of pixels (for example, groups of 2x2 pixels) and perform an aggregation over them. One of the possible aggregations we can make is take the maximum value of the pixels in the group (this is known as **Max Pooling**). Another common aggregation is taking the average (**Average Pooling**).

You may often use max or average pooling between convolution dimensionality instead of varying the stride length. Pooling looks at a region, which lets you assume, is 2 x 2 and keeps only the largest or average value. The following image depicts a 2 x 2 matrix that depicts pooling:



A pooling region always has the same-sized stride as the pool size. This helps to avoid overlapping.

We can see two things:

- First of all, **the image size is reduced** to its half: by taking groups of 2x2 pixels and only retaining the maximum, now the image is half bigger.
- The edges we kept when applying the convolution with the vertical edge filter not only are maintained, but are also **intensified**.

This means we have been able to reduce the information the image contains (by keeping only half of the pixels), but still we are able to intensify the useful features the filters show when convolving the image (i.e., reducing the size by retaining the important information).

CNNs :

A basic convolutional neural network can be viewed as a series of convolutional layers, followed by an activation function, followed by a pooling (downscaling) layer, repeated many times.

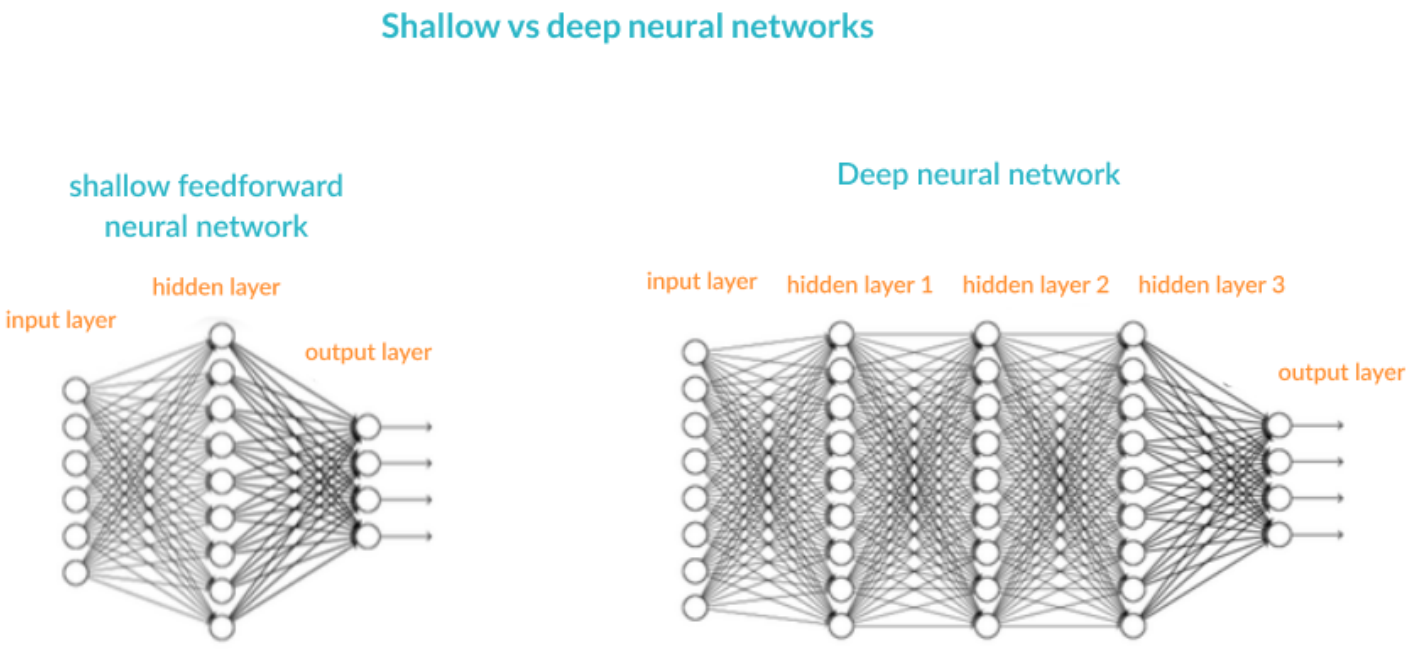
CNNs have two main parts:

- A convolution/pooling mechanism that breaks up the image into features and analyzes them
- A fully connected layer that takes the output of convolution/pooling and predicts the best label to describe the image

Convolutional Neural Networks vs Fully Connected Neural Networks :

the classic neural network architecture, in which all neurons connect to all neurons in the next layer. A convolutional neural network is a special kind of feedforward neural network with fewer weights than a fully-connected network.In a fully-connected feedforward neural network, every node in the input is tied to every node in the first layer, and so on. There is no convolution kernel.

Convolutional neural networks enable deep learning for computer vision.



The classic neural network architecture was found to be inefficient for computer vision tasks. Images represent a large input for a neural network (they can have hundreds or thousands of pixels and up to 3 color channels). In a classic fully connected network, this requires a huge number of connections and network parameters.

A convolutional neural network leverages the fact that an image is composed of smaller details, or features, and creates a mechanism for analyzing each feature in isolation, which informs a decision about the image as a whole.

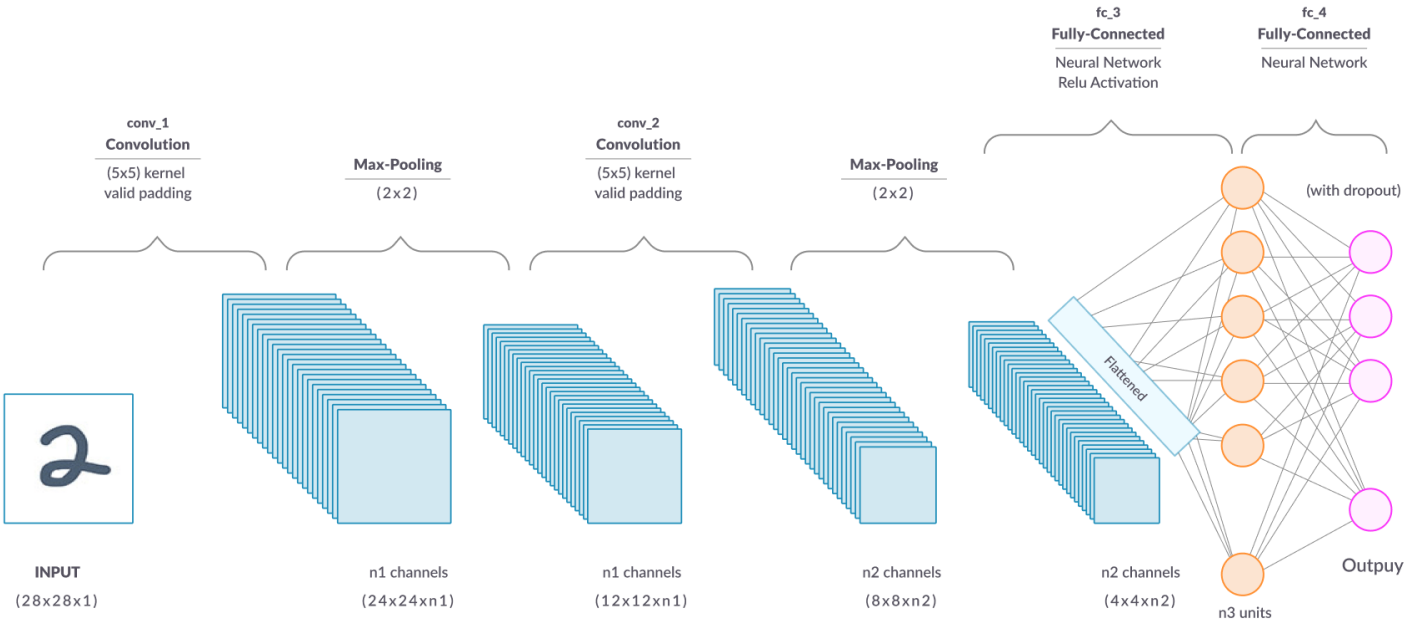
As part of the convolutional network, there is also a **fully connected layer** that takes the end result of the convolution/pooling process and reaches a classification decision.

CNN Architecture: Types of Layers

Convolutional Neural Networks have several types of layers:

- **Convolutional layer**—a "filter" passes over the image, scanning a few pixels at a time and creating a feature map that predicts the class to which each feature belongs.
- **Pooling layer** (**downsampling**)—reduces the amount of information in each feature obtained in the convolutional layer while maintaining the most important information (there are usually several rounds of convolution and pooling).
- **Fully connected input layer** (**flatten**)—takes the output of the previous layers, "flattens" them and turns them into a single vector that can be an input for the next stage.
- **The first fully connected layer**—takes the inputs from the feature analysis and applies weights to predict the correct label.
- **Fully connected output layer**—gives the final probabilities for each label.

Below is an example showing the layers needed to process an image of a written digit, with the number of pixels processed in every stage. This is a very simple image—larger and more complex images would require more convolutional/pooling layers.



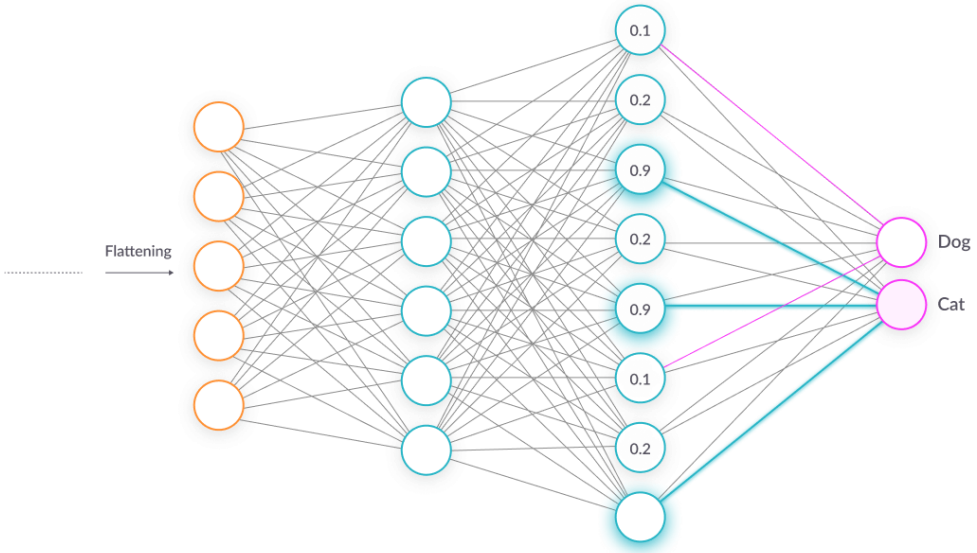
The role of a fully connected layer in a CNN architecture

The objective of a fully connected layer is to take the results of the convolution/pooling process and use them to classify the image into a label (in a simple classification example).

The output of convolution/pooling is flattened into a single vector of values, each representing a probability that a certain feature belongs to a label. For example, if the image is of a cat, features representing things like whiskers or fur should have high probabilities for the label "cat".

The image below illustrates how the input values flow into the first layer of neurons. They are multiplied by weights and pass through an activation function (typically ReLu), just like in a classic artificial neural network. They then pass forward to the output layer, in which every neuron represents a classification label.

The fully connected part of the CNN network goes through its own backpropagation process to determine the most accurate weights. Each neuron receives weights that prioritize the most appropriate label. Finally, the neurons "vote" on each of the labels, and the winner of that vote is the classification decision.



How does convolution and pooling fit in a convolutional neural network to perform vision related tasks?

To wrap up, I will provide an intuitive explanation on how these two ideas are built inside the convolutional neural networks.

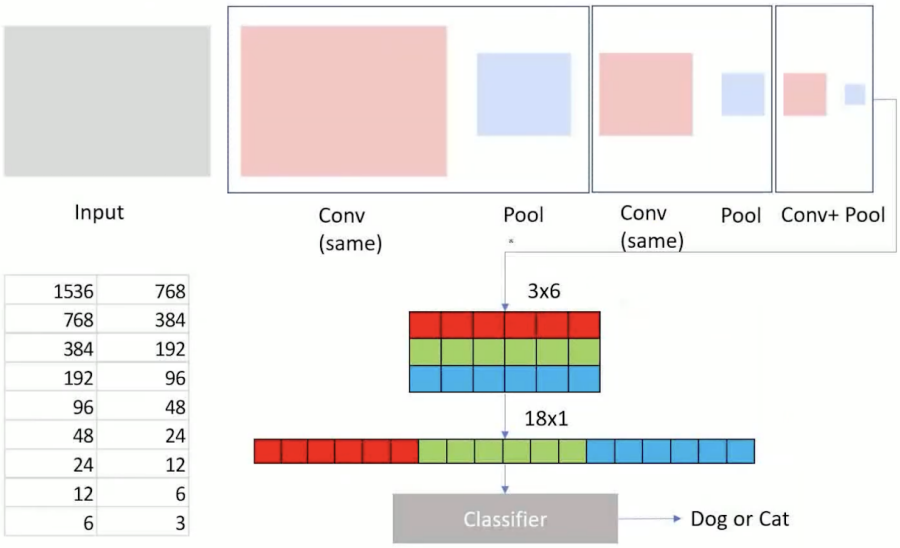
A basic convolutional neural network can be seen as a sequence of convolution layers and pooling layers. When the image goes through them, **the important features are kept** in the convolution layers, and thanks to the pooling layers, **these features are intensified** and kept over the network, while **discarding all the information** that doesn't make a difference for the task.

These important features can go **from single lines or edges** (as we saw in the examples) to **more complex things** (such as, for example, the ears of a dog) while we are travelling through the neural network.

And there is a slight, but crucial detail: when applying convolutions in the examples we have seen, we chose a filter and then observed the output. In practice, if, as an example, we are performing an image classification task, the network **will learn which filters allow us to extract the most insightful features so as to distinguish between the classes** in our training dataset.

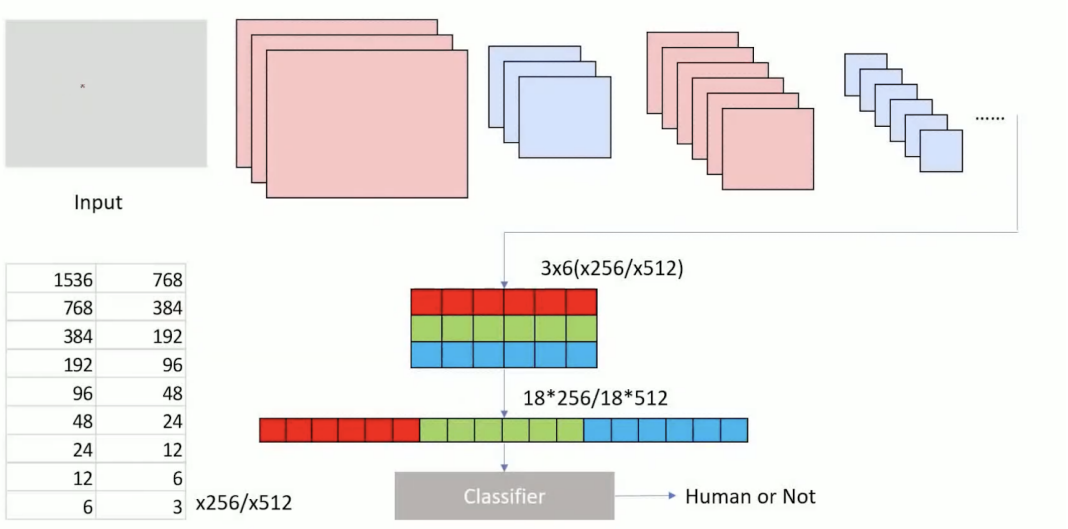
Convolution + Pool Cascade in CNN:

We usually do multiple layers of convolution and pooling until we get the feature map of required size at multiple layers. And feed it to the classifier. We usually use multiple filters based on our requirements.



N-Convolution + Pool Cascade :

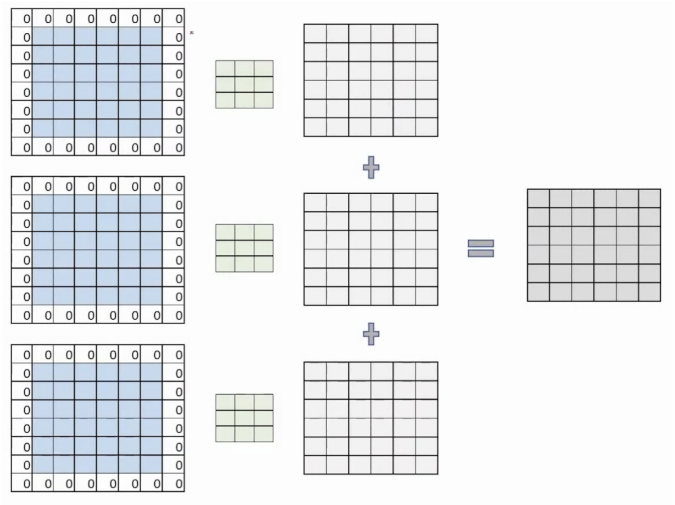
We take many filters. In the first conv+pool layer we used three different filters and convolve. Now, in layer 2 use 2 different filters for each previous feature map (current feature vector). In each round, the depth of the feature map will keep increasing. We usually get 256/512 feature maps, we send this to the classifier after flattening. So, CNNs are memory intensive.



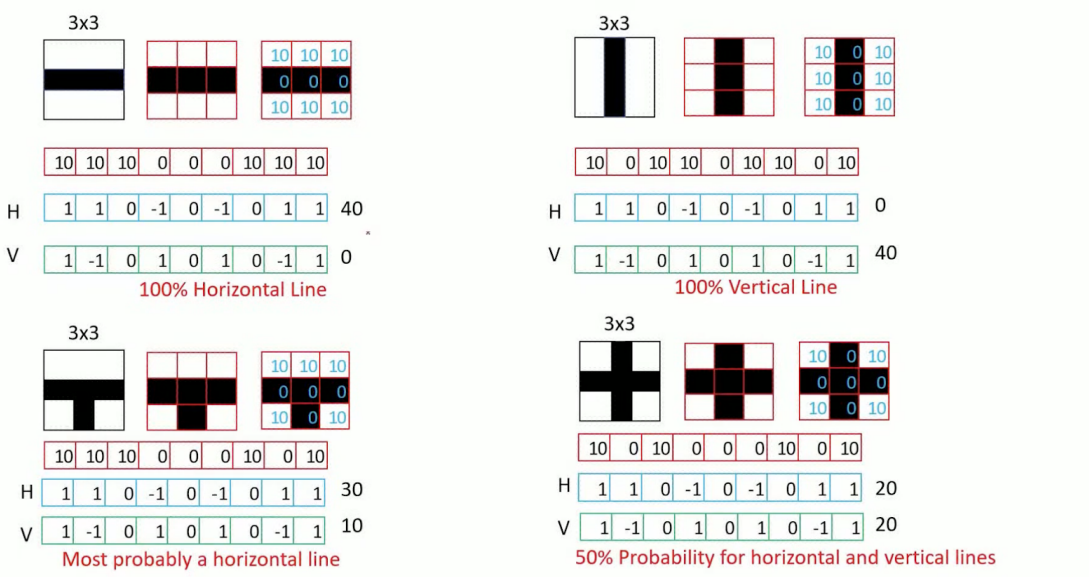
N layer convolution and M feature maps :

We used 3 different filters and added the result to obtain the feature map. If we use M sets of 3-filters we get M feature maps. We used m*3 filters.

So, CNNs are memory intensive.

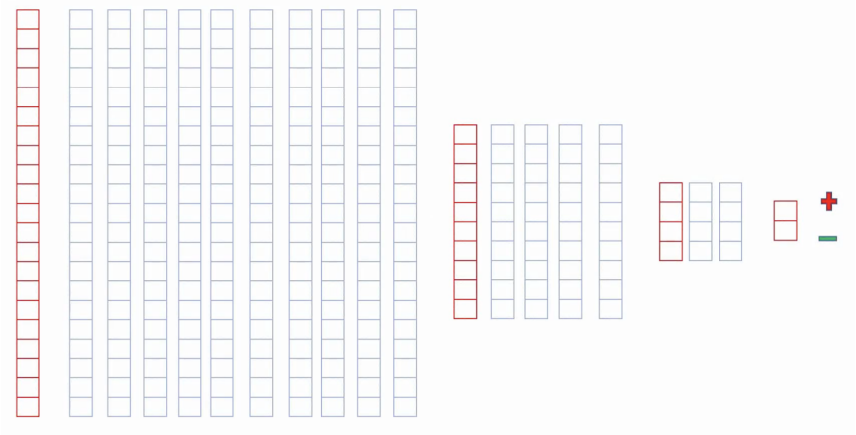


Classification using Fully Connected layers :



FC layer cascade :

Fully connected layer cascading is just similar to what we did in conv+pool. For binary classification we only want two outputs so we sample down the FC layer. These layers in between before sampling down are hidden layers.



Softmax :

The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Computation :

Softmax (1,4) :

S = e^1 + e^4

Softmax (0,4) = (e^1/s , e^4/s)

One use of the softmax function would be at the end of a neural network. Let us consider a convolutional neural network which recognizes if an image is a cat or a dog. Note that an image must be either a cat or a dog, and cannot be both, therefore the two classes are mutually exclusive. Typically, the final fully connected layer of this network would produce values like [-7.98, 2.39] which are not normalized and cannot be interpreted as probabilities. If we add a softmax layer to the network, it is possible to translate the numbers into a probability distribution. This means that the output can be displayed to a user, for example the app is 95% sure that this is a cat. It also means that the output can be fed into other machine learning algorithms without needing to be normalized, since it is guaranteed to lie between 0 and 1.

Note that if the network is classifying images into dogs and cats, and is configured to have only two output classes, then it is forced to categorize every image as either dog or cat, even if it is neither. If we need to allow for this possibility, then we must reconfigure the neural network to have a third output for miscellaneous.

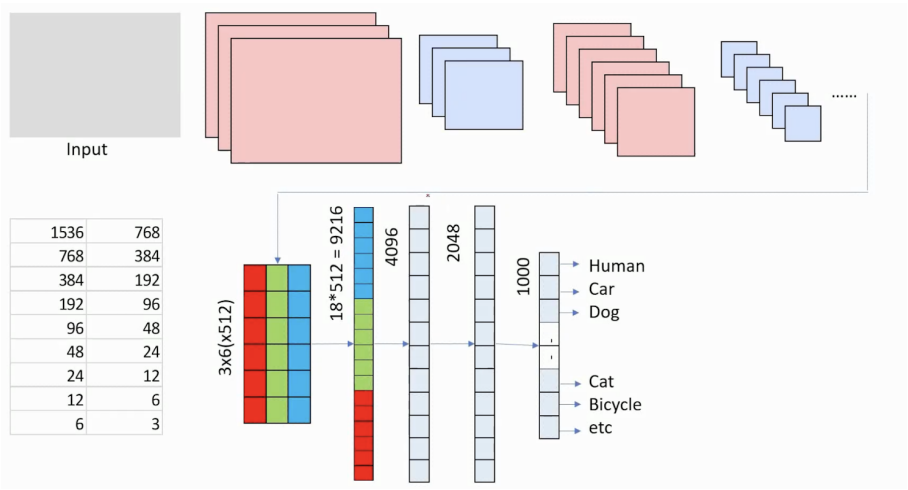
Example :

Cat	5	148.4131591	**0.502010161**
Dog	4.9	134.2897797	**0.454237578**
Hippo	1	2.718281828	0.009194637
Rhino	0.9	2.459603111	0.008319651
Elephant	-1	0.367879441	0.001244359
Mouse	2	7.389056099	0.024993614
		295.6377593	

5 and 4.9 are pretty close but the softmax scores help us in removing all these negative values.

Complete Convolution Network :

Output of the fully connected layer after cascading is sent into the softmax layer. Here we only have



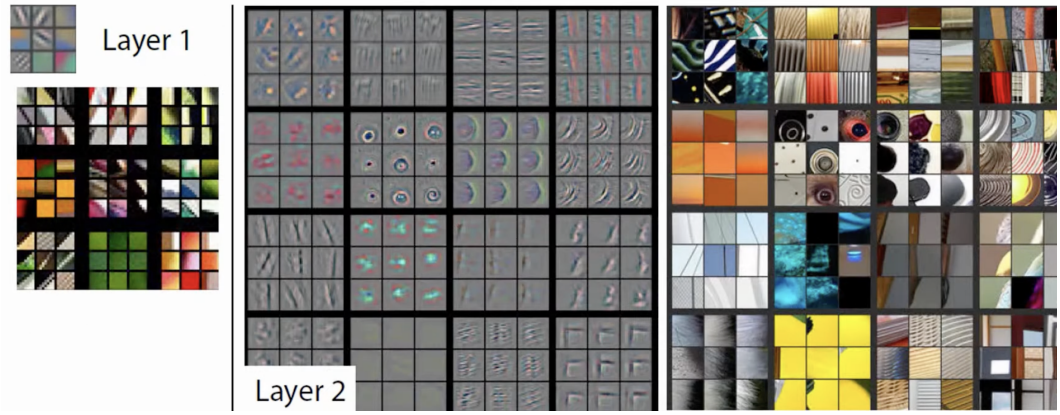
Reason for using multiple layers :

L1 :

In this layer model usually tries to learn some basic curves or edges.

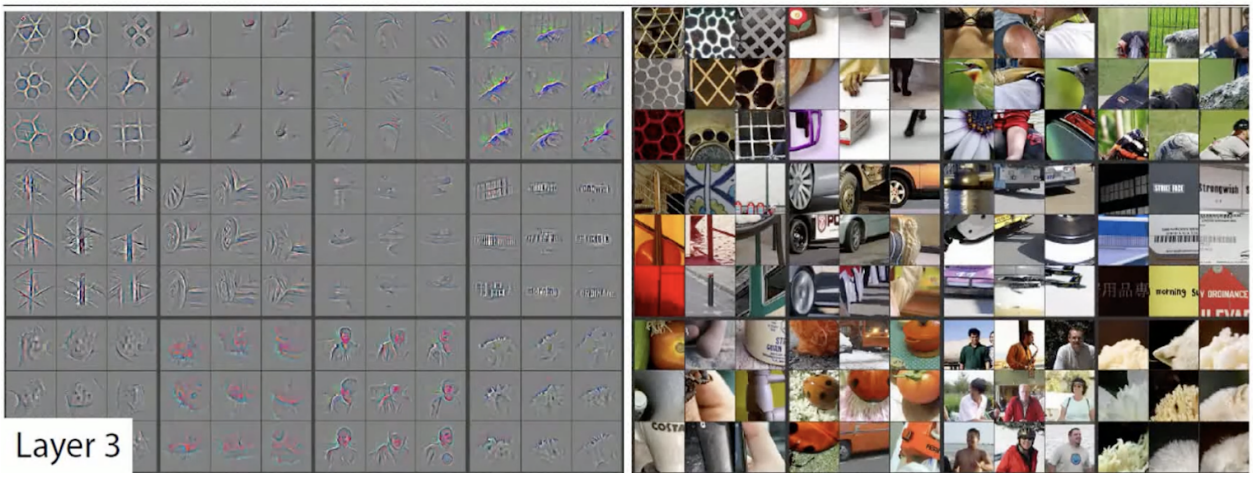
L2 :

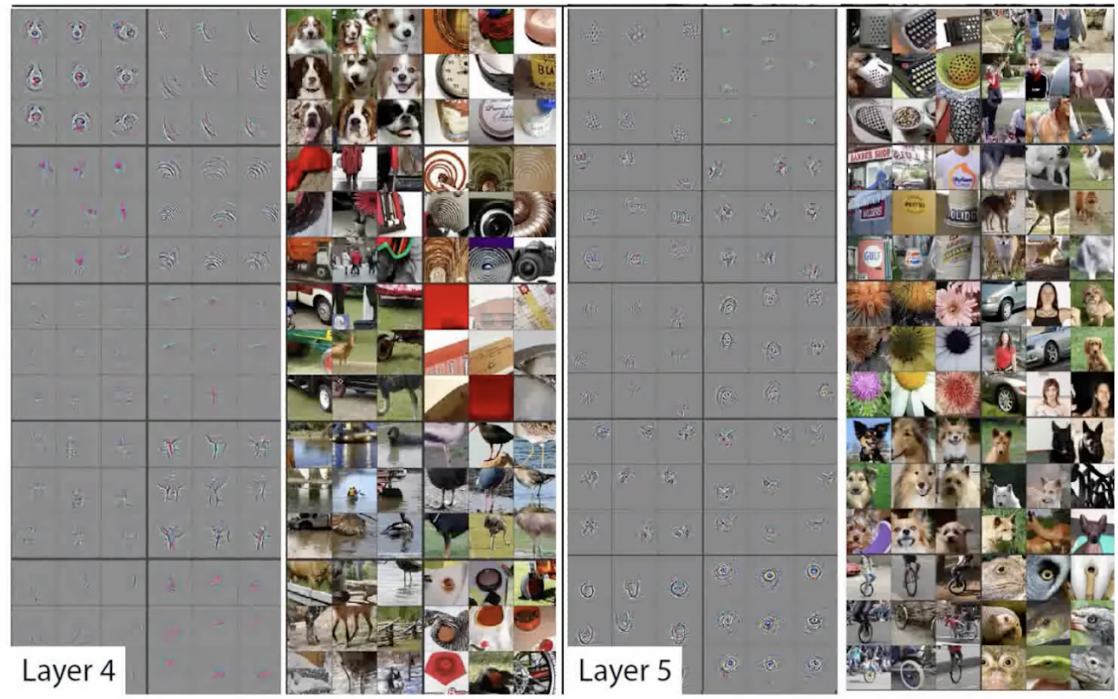
In this layer it tries to learn some of the complex curves by combining some of them from Layer 1.



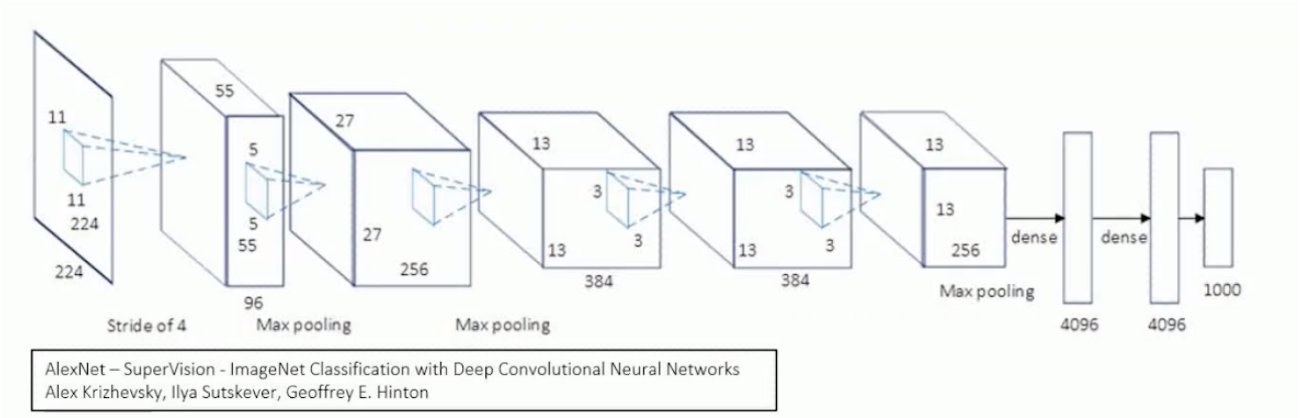
L3 :

In this layer model tries to learn much more complex patterns by putting together the one obtained from L2.

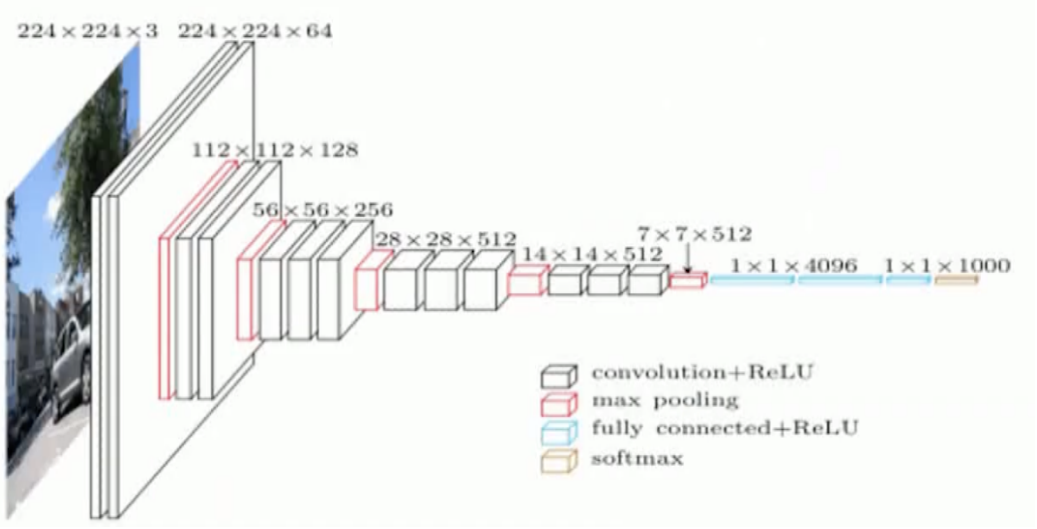




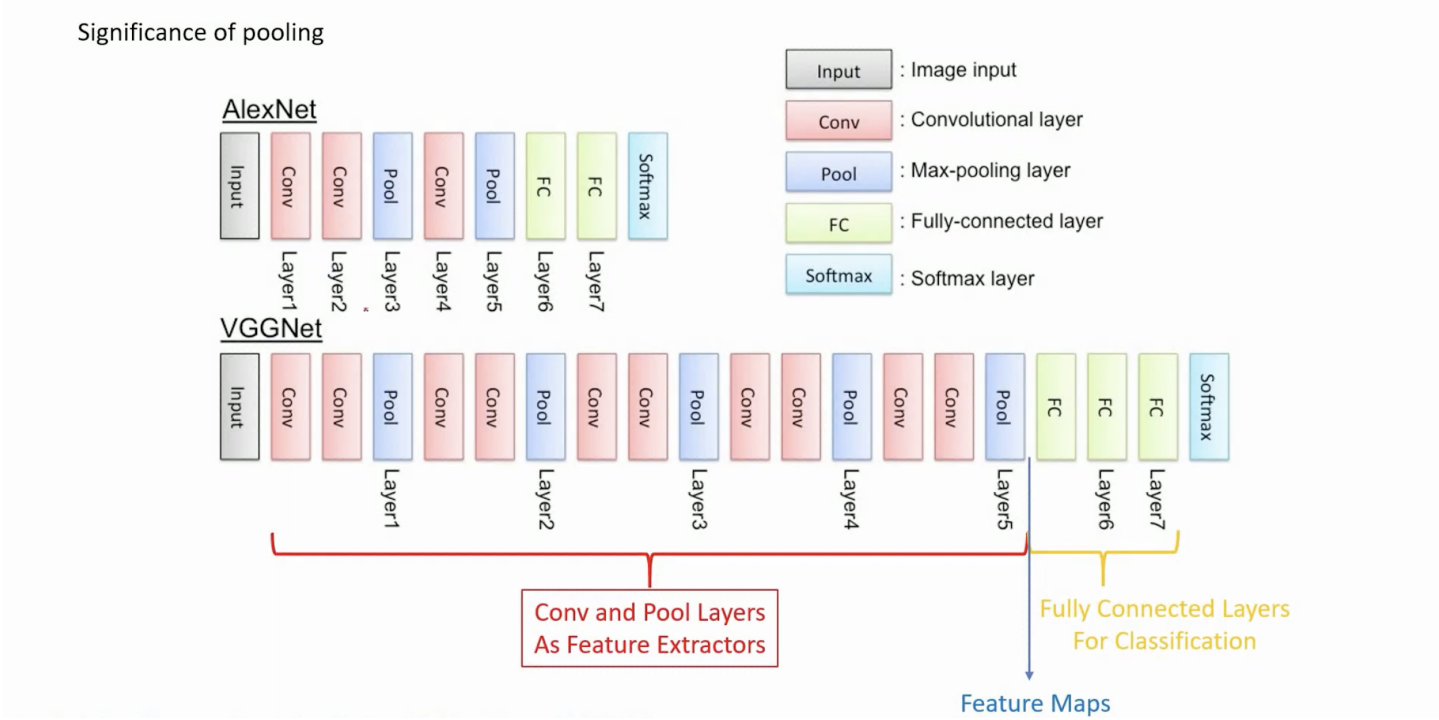
AlexNet :



VGG Net :



AlexNet and VGG Net :



Instead of HOG/SIFT we can use conv and pool layers to extract features.

Note :

Resnet 50 has 50 layers and resnet 152 has 152 layers. Way too much to compute or store the information.

Note :

Humans learn a task using Ready-Fire-Aim (reference) . similarly our system keeps adjusting the weights of the filter until it achieves the target (loss reduces). This is backpropagation. To make the model robust, we use test-train split. This makes the system immune to some changes and variations.