# Convolutional Neural Networks

### Computer Vision

Some of the computer vision problems which we will be solving in this article are:

1. Image classification
2. Object detection
3. Neural style transfer

One major problem with computer vision problems is that the input data can get really big. Suppose an image is of the size 68 X 68 X 3. The input feature dimension then becomes 12,288. This will be even bigger if we have larger images (say, of size 720 X 720 X 3). Now, if we pass such a big input to a neural network, the number of parameters will swell up to a HUGE number (depending on the number of hidden layers and hidden units). This will result in more computational and memory requirements – not something most of us can deal with.

## A brief history of convolutional neural networks

## Convolutional neural networks, also called ConvNets, were first introduced in the 1980s by Yann LeCun, a postdoctoral computer science researcher. LeCun had built on the work done by Kunihiko Fukushima, a Japanese scientist who, a few years earlier, had invented the neocognitron, a very basic image recognition neural network.

## The early version of CNNs, called LeNet (after LeCun), could recognize handwritten digits. CNNs found a niche market in banking and postal services and banking, where they read zip codes on envelopes and digits on checks.

## But despite their ingenuity, ConvNets remained on the sidelines of computer vision and artificial intelligence because they faced a serious problem: They could not scale. CNNs needed a lot of data and compute resources to work efficiently for large images. At the time, the technique was only applicable to images with low resolutions.

## In 2012, AlexNet showed that perhaps the time had come to revisit [deep learning](https://bdtechtalks.com/2019/02/15/what-is-deep-learning-neural-networks/), the branch of AI that uses multi-layered neural networks. The availability of large sets of data, namely the ImageNet dataset with millions of labeled pictures, and vast compute resources enabled researchers to create complex CNNs that could perform computer vision tasks that were previously impossible.

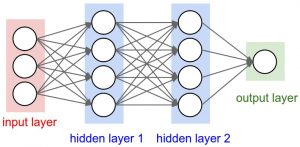
## How do CNNs work?

It is assumed that reader knows the concept of Neural Network.  
When it comes to Machine Learning, [Artificial Neural Networks](https://www.geeksforgeeks.org/implementing-ann-training-process-in-python/) perform really well. Artificial Neural Networks are used in various classification task like image, audio, words. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural Network. In this blog, we are going to build basic building block for CNN.  
Before diving into the Convolution Neural Network, let us first revisit some concepts of Neural Network. In a regular Neural Network there are three types of layers:

1. **Input Layers:** It’s the layer in which we give input to our model. The number of neurons in this layer is equal to total number of features in our data (number of pixels in case of an image).
2. **Hidden Layer:** The input from Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layers can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by addition of learnable biases followed by activation function which makes the network nonlinear.
3. **Output Layer:** The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into probability score of each class.

The data is then fed into the model and output from each layer is obtained this step is called feed forward, we then calculate the error using an error function, some common error functions are cross entropy, square loss error etc. After that, we back propagate into the model by calculating the derivatives. This step is called Back propagation which basically is used to minimize the loss.  
Here’s the basic python code for a neural network with random inputs and two hidden layers. 

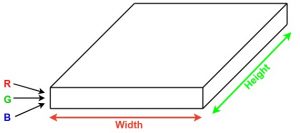
**W1,W2,W3,b1,b2,b3** are learnable parameter of the model. 



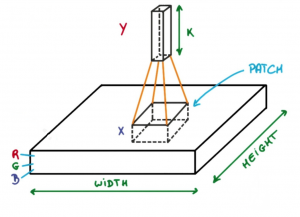
*Image source: cs231n.stanford.edu*

**Convolution Neural Network**

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image) and height (as image generally has red, green, and blue channels). 



Now imagine taking a small patch of this image and running a small neural network on it, with say, k outputs and represent them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different width, height, and depth. Instead of just R, G and B channels now we have more channels but lesser width and height. This operation is called Convolution. If patch size is same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights. 



*Image source: Deep Learning Udacity*

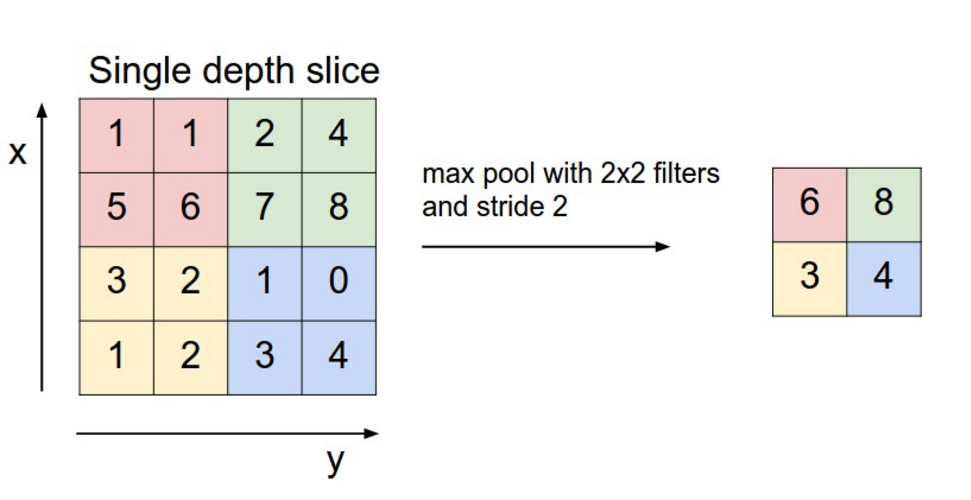
Now let’s talk about a bit of mathematics which is involved in the whole convolution process. 

* Convolution layers consist of a set of learnable filters (patch in the above image). Every filter has small width and height and the same depth as that of input volume (3 if the input layer is image input).
* For example, if we have to run convolution on an image with dimension 34x34x3. Possible size of filters can be axax3, where ‘a’ can be 3, 5, 7, etc but small as compared to image dimension.
* During forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have value 2 or 3 or even 4 for high dimensional images) and compute the dot product between the weights of filters and patch from input volume.
* As we slide our filters we’ll get a 2-D output for each filter and we’ll stack them together and as a result, we’ll get output volume having a depth equal to the number of filters. The network will learn all the filters.

**Layers used to build ConvNets**

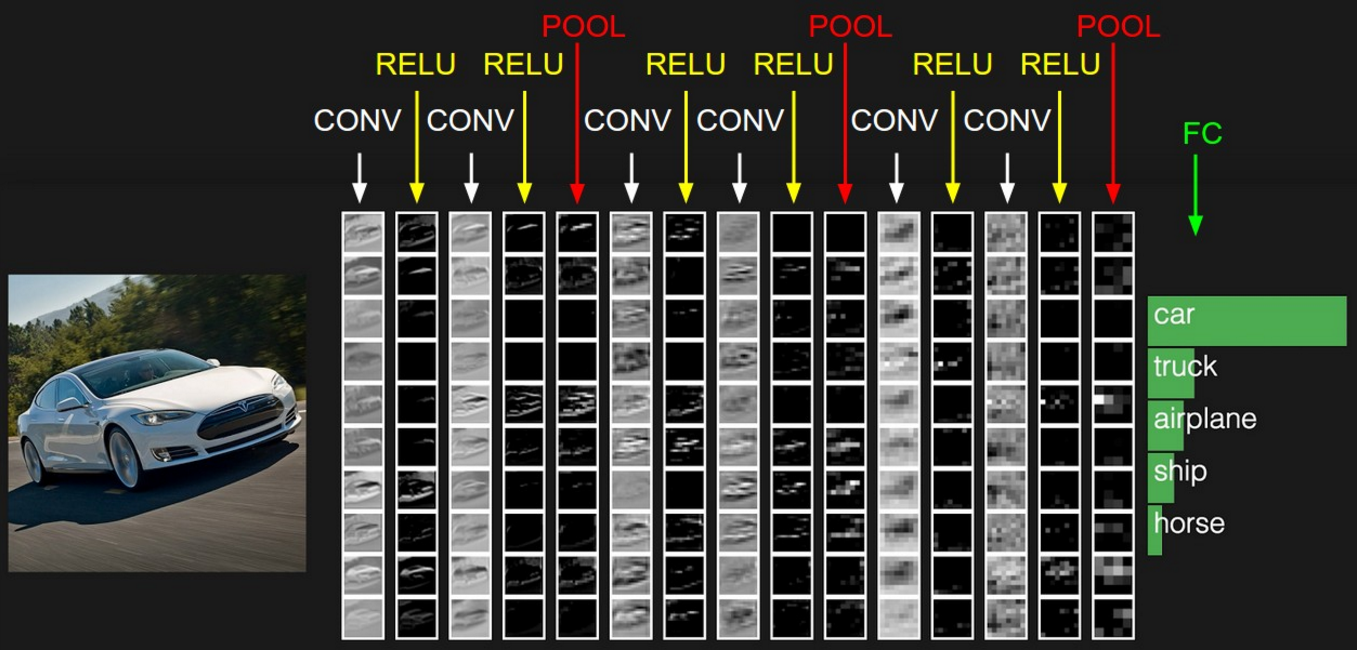
A covnets is a sequence of layers, and every layer transforms one volume to another through differentiable function.   
**Types of layers:**   
Let’s take an example by running a covnets on of image of dimension 32 x 32 x 3. 

1. **Input Layer:** This layer holds the raw input of image with width 32, height 32 and depth 3.
2. **Convolution Layer:** This layer computes the output volume by computing dot product between all filters and image patch. Suppose we use total 12 filters for this layer we’ll get output volume of dimension 32 x 32 x 12.
3. **Activation Function Layer:** This layer will apply element wise activation function to the output of convolution layer. Some common activation functions are RELU: max(0, x), Sigmoid: 1/(1+e^-x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimension 32 x 32 x 12.
4. **Pool Layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents from overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.



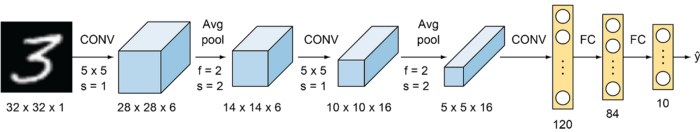
*Image source: cs231n.stanford.edu*

1. **Fully-Connected Layer:** This layer is regular neural network layer which takes input from the previous layer and computes the class scores and outputs the 1-D array of size equal to the number of classes.



# LeNet-5 (1998)

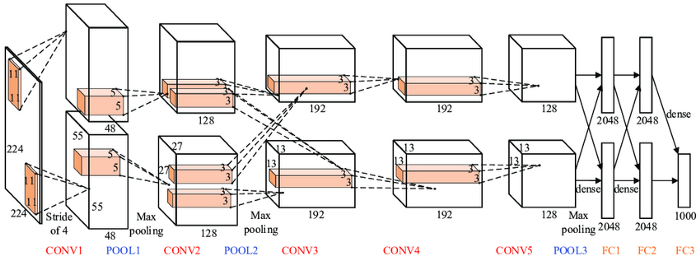
This is also known as the Classic Neural Network that was designed by Yann LeCun, Leon Bottou, Yosuha Bengio and Patrick Haffner for handwritten and machine-printed character recognition in 1990’s which they called LeNet-5. The architecture was designed to identify handwritten digits in the MNIST data-set. The architecture is pretty straightforward and simple to understand. The input images were gray scale with dimension of 32\*32\*1 followed by two pairs of Convolution layer with stride 2 and Average pooling layer with stride 1. Finally, fully connected layers with Softmax activation in the output layer. Traditionally, this network had 60,000 parameters in total.



Lenet-5 Architecture

2. AlexNet

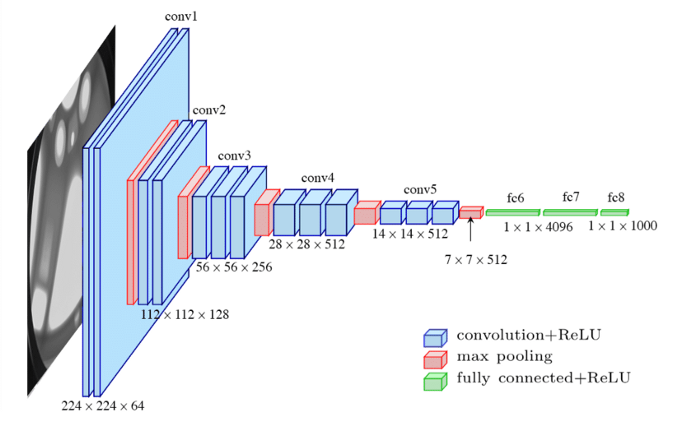
This network was very similar to LeNet-5 but was deeper with 8 layers, with more filters, stacked convolutional layers, max pooling, dropout, data augmentation, ReLU and SGD. AlexNet was the winner of the ImageNet ILSVRC-2012 competition, designed by Alex Krizhevsky, Ilya Sutskever and Geoffery E. Hinton. It was trained on two Nvidia Geforce GTX 580 GPUs, therefore, the network was split into two pipelines. AlexNet has 5 Convolution layers and 3 fully connected layers. AlexNet consists of approximately 60 M parameters. A major drawback of this network was that it comprises of too many hyper-parameters. A new concept of Local Response Normalization was also introduced in the paper.



AlexNet Architecture

3. VGG-16 Net

The major shortcoming of too many hyper-parameters of AlexNet was solved by VGG Net by replacing large kernel-sized filters (11 and 5 in the first and second convolution layer, respectively) with multiple 3×3 kernel-sized filters one after another. The architecture developed by Simonyan and Zisserman was the 1st runner up of the Visual Recognition Challenge of 2014. The architecture consist of 3\*3 Convolutional filters, 2\*2 Max Pooling layer with a stride of 1, keeping the padding same to preserve the dimension. In total, there are 16 layers in the network where the input image is RGB format with dimension of 224\*224\*3, followed by 5 pairs of Convolution(filters: 64, 128, 256,512,512) and Max Pooling. The output of these layers is fed into three fully connected layers and a softmax function in the output layer. In total there are 138 Million parameters in VGG Net.

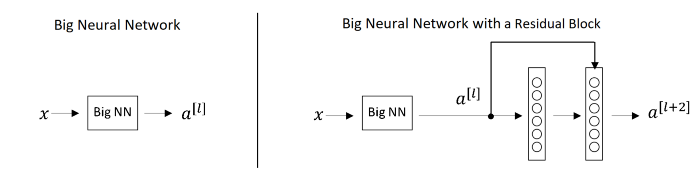


VGG-16 Architecture

**Drawbacks of VGG Net:**  
1. Long training time  
2. Heavy model  
3. Computationally expensive  
4. Vanishing/exploding gradient problem

**4. ResNet**

ResNet, the winner of ILSVRC-2015 competition are deep networks of over 100 layers. Residual networks are similar to VGG nets however with a sequential approach they also use “Skip connections” and “batch normalization” that helps to train deep layers without hampering the performance. After VGG Nets, as CNNs were going deep, it was becoming hard to train them because of vanishing gradients problem that makes the derivate infinitely small. Therefore, the overall performance saturates or even degrades. The idea of skips connection came from highway network where gated shortcut connections were used.



Normal Deep Networks vs Networks with skip connections

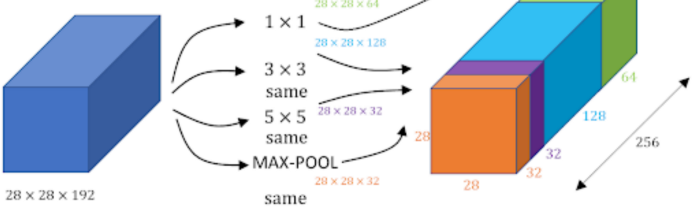
For the above figure for network with skip connection, a[l+2]=g(w[l+2]a[l+1]+ a[l])

Lets say for some reason, due to weight decay w[l+2] becomes 0, therefore, a[l+2]=g(a[l])

Hence, the layer that is introduced doesnot hurt the performance of the neural network. This the reason, increasing layers doesn’t decrease the training accuracy as some layers may make the result worse. The concept of skip connections can also be seen in LSTMs.

# 5. GoogLeNet/Inception(2014)

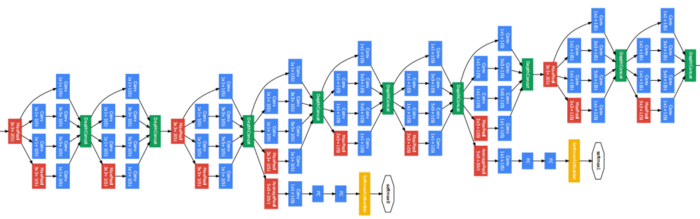
Inception network also known as GoogleLe Net was proposed by developers at google in “Going Deeper with Convolutions” in 2014. The motivation of InceptionNet comes from the presence of sparse features Salient parts in the image that can have a large variation in size. Due to this, the selection of right kernel size becomes extremely difficult as big kernels are selected for global features and small kernels when the features are locally located. The InceptionNets resolves this by stacking multiple kernels at the same level. Typically it uses 5\*5, 3\*3 and 1\*1 filters in one go. For better understanding refer to the image below:



Inception Module of GoogleLe Net

Note: Same padding is used to preserve the dimension of the image.

As we can see in the image, three different filters are applied in the same level and the output is combined and fed to the next layer. The combination increases the overall number of channels in the output. The problem with this structure was the number of parameter (120M approx.) that increases the computational cost. Therefore, 1\*1 filters were used before feeding the image directly to these filters that act as a bottleneck and reduces the number of channels. Using 1\*1 filters, the parameter were reduced to 1/10 of the actual. GoogLeNet has 9 such inception modules stacked linearly. It is 22 layers deep (27, including the pooling layers). It uses global average pooling at the end of the last inception module. Inception v2 and v3 were also mentioned in the same paper that further increased the accuracy and decreasing computational cost.



Several Inception modules are linked to form a dense network

Side branches can be seen in the network which predicts output in order to check the shallow network performance at lower levels.