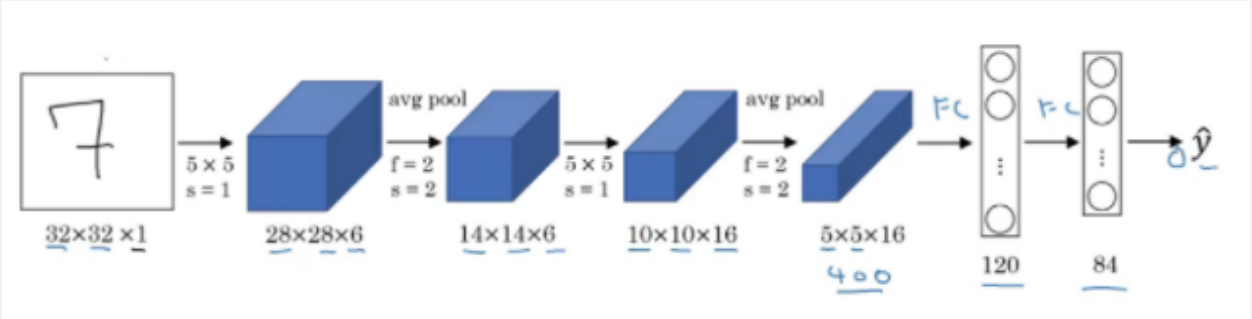
# CNN Architectures

—

In this section, we will look at the following popular networks:

1. **LeNet-5**
2. **AlexNet**
3. **VGG**
4. **ResNet**

## LeNet-5




It takes a grayscale image as input. Once we pass it through a combination of convolution and pooling layers, the output will be passed through fully connected layers and classified into corresponding classes. The total number of parameters in LeNet-5 are:

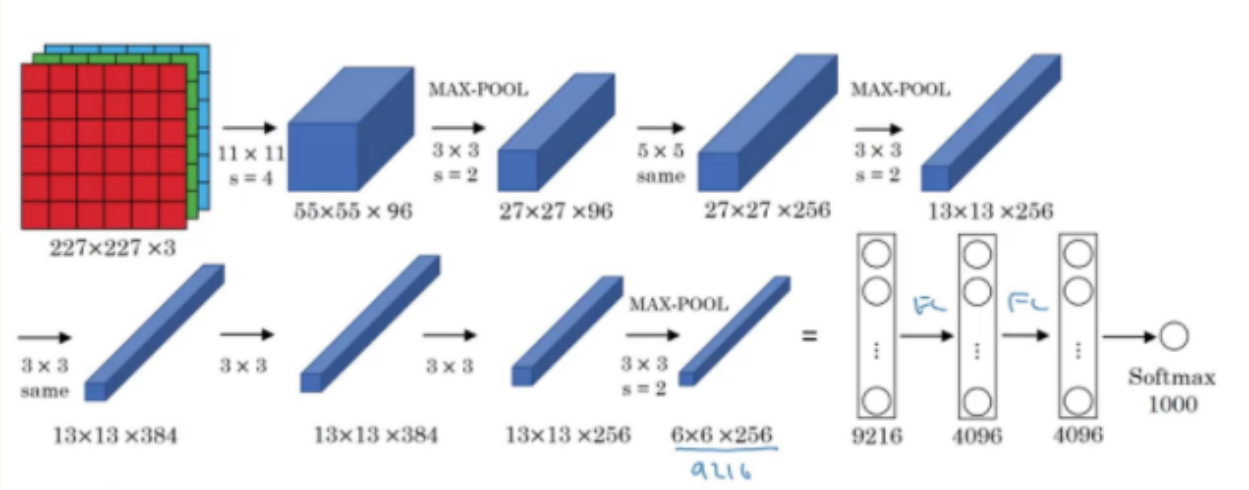
Parameters: 60k

Layers flow: Conv -> Pool -> Conv -> Pool -> FC -> FC -> Output

Activation functions: Sigmoid/tanh and ReLu

## AlexNet

An illustrated summary of AlexNet is given below:




This network is similar to LeNet-5 with just more convolution and pooling layers:

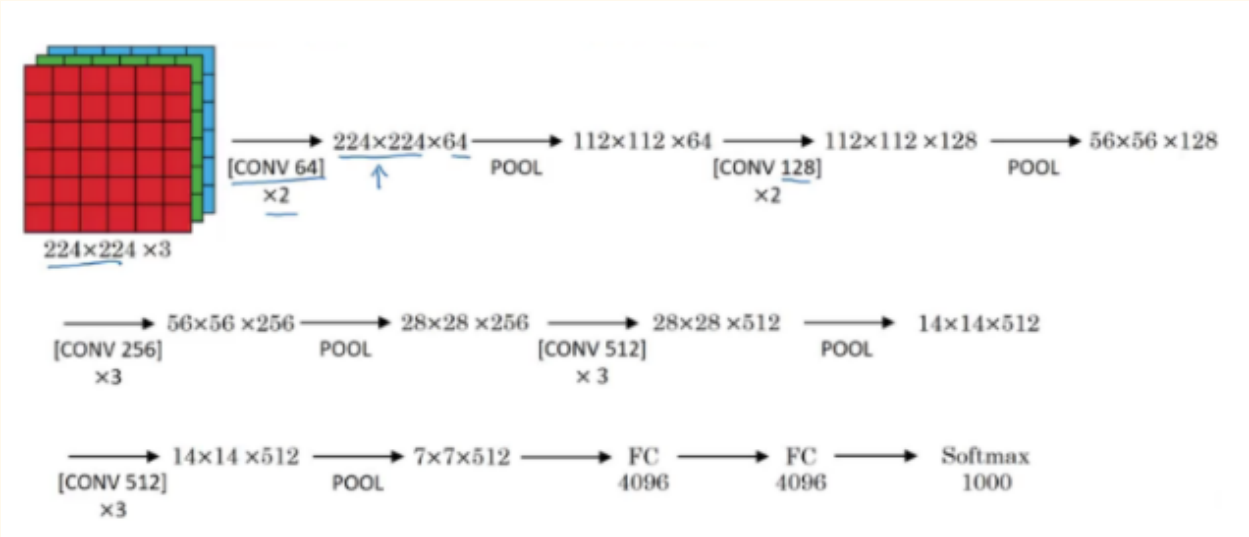
Parameters: 60 million

Activation function: ReLu

## VGG-16

The underlying idea behind VGG-16 was to use a much simpler network where the focus is on having convolution layers that have 3 X 3 filters with a stride of 1 (and always using the same padding). The max pool layer is used after each convolution layer with a filter size of 2 and a stride of 2.

Let's look at the architecture of VGG-16:




As it is a bigger network, the number of parameters are also more.

Parameters: 138 million

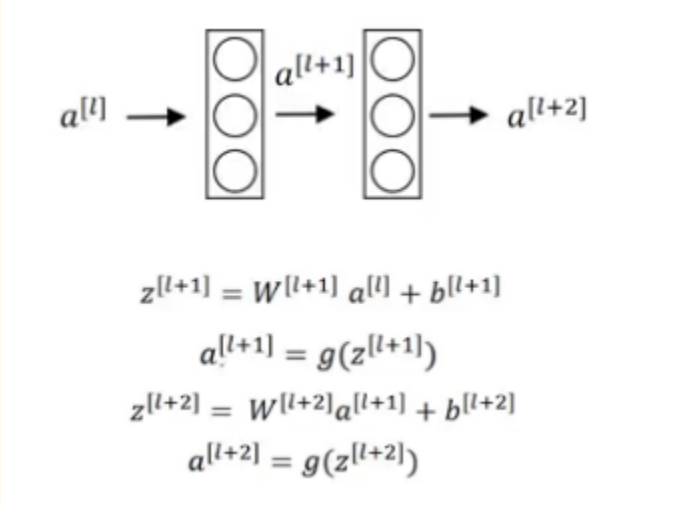
These are three classic architectures. Next, we'll look at more advanced architecture starting with ResNet.

## ResNet

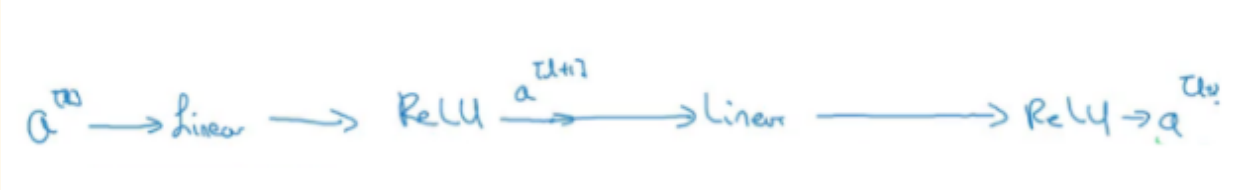
Training very deep networks can lead to problems like vanishing and exploding gradients. How do we deal with these issues? We can skip connections where we take activations from one layer and feed it to another layer that is even deeper in the network. There are residual blocks in ResNet which help in training deeper networks.

Residual Blocks

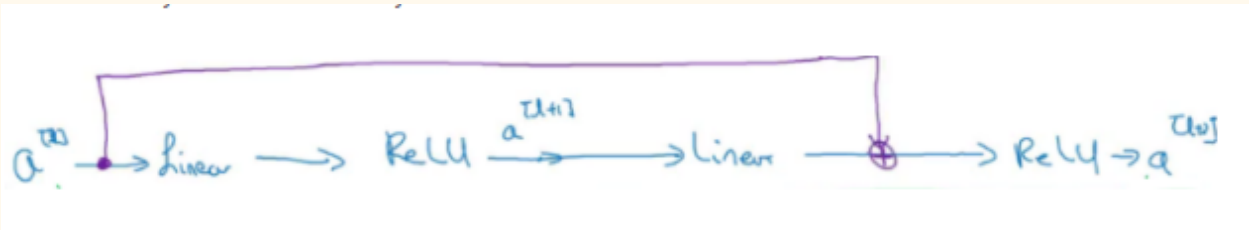
The general flow to calculate activations from different layers can be given as:




This is how we calculate the activations a[l+2] using the activations a[l] and then a[l+1]. a[l] needs to go through all these steps to generate a[l+2]:



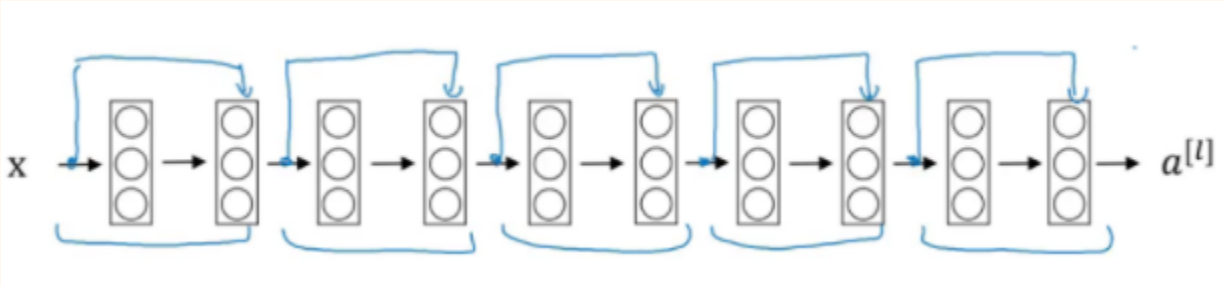

In a residual network, we make a change in this path. We take the activations a[l] and pass them directly to the second layer:



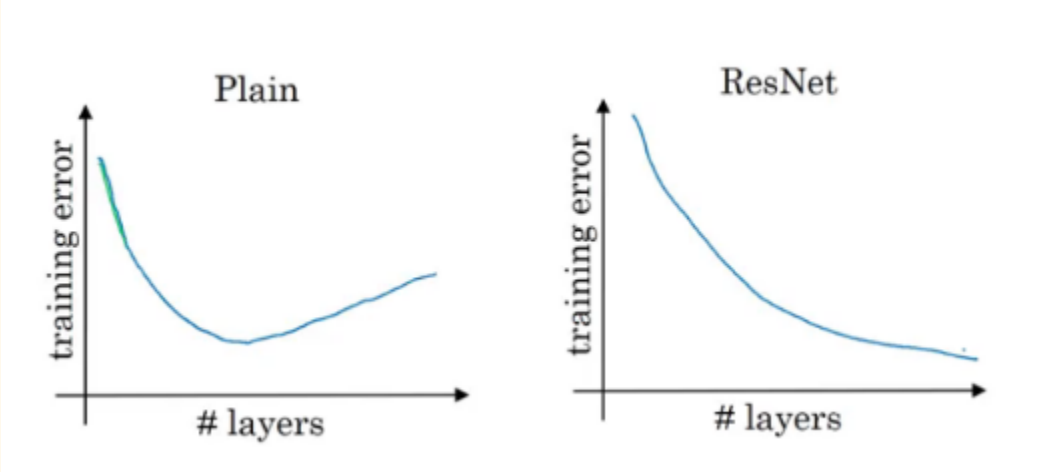

So, the activations a[l+2] will be:

a[l+2] = g(z[l+2] + a[l])

The residual network can be shown as:




The benefit of training a residual network is that even if we train deeper networks, the training error does not increase. Whereas in case of a plain network, the training error first decreases as we train a deeper network and then starts to rapidly increase:




We now have an overview of how ResNet works. But why does it perform so well? Let's find out!

Why ResNets Work?

In order to make a good model, we first have to make sure that it's performance on the training data is good. That's the first test and there really is no point in moving forward if our model fails here. We have seen earlier that training deeper networks using a plain network increases the training error after a point of time. But while training a residual

network, this isn't the case. Even when we build a deeper residual network, the training error generally does not increase.

The equation to calculate activation using a residual block is given by:

$$a[l+2] = g(z[l+2] + a[l])$$

$$a[l+2] = g(w[l+2] * a[l+1] + b[l+2] + a[l])$$

Now, say $w[l+2] = 0$ and the bias $b[l+2]$ is also 0, then:

$$a[l+2] = g(a[l])$$

It is fairly easy to calculate a[l+2] knowing just the value of a[l]. As per the research paper, ResNet is given by: