1. What are channels and Kernels

* An image can consist of a one/two/three-dimensional tensor of integers; each dimension corresponds to a color channel. An image can be represented in multiple formats - RGB, YUV, Grayscale images etc. But here for simplicity, if we consider an RGB image, the three dimensions - R, G and B are called channels. The numbers range for each color channel is between 0 (black) to 255(white)

A kernel is a matrix of weights which are multiplied with the input (image/channel) to extract relevant features. The dimensions of the kernel matrix is *how the convolution* gets it’s name. For example, in 2D convolutions, the kernel matrix is a 2D matrix.  
  
A filter however is a concatenation of multiple kernels, each kernel assigned to a particular channel of the input. Filters are always one dimension more than the kernels. For example, in 2D convolutions, filters are 3D matrices (which is essentially a concatenation of 2D matrices i.e. the kernels). So for a CNN layer with kernel dimensions h\*w and input channels k, the filter dimensions are k\*h\*w.

Each type of filter helps to extract different aspects or features from the input image, e.g. horizontal / vertical / diagonal edge. A “Kernel” refers to a 2D array of weights. The term “filter” is for 3D structures of multiple kernels stacked together. For a 2D filter, the filter is the same as the kernel. But for a 3D filter and most convolutions in deep learning, a filter is a collection of kernels. Each kernel is unique, emphasizing different aspects of the input channel.

1. Why should we (nearly) always use 3x3 kernels?

* There’s a couple advantages to using smaller filters. Say for instance if we used a 5x5 filter with stride 2. This would give you a receptive field of 5 at a computational cost of 25 multiply-adds for a single channel. If instead we used two layers of 3x3 filters with strides 1,2 or a with a dilation of 2, we will get the same receptive field and same stride but at a cost roughly proportional to 3\*3+3\*3=18 multiply-adds. So decoupling a 5x5 into 2 layers of 3x3 is an optimization. In short, a 3x3 convolution with a dilation of 2 OR two-3x3 convolutions will give us the same receptive field of a single 5x5 convolution operation but with less computational parameters.

1. How many times do we need to perform 3x3 convolutions operations to reach close to 1x1 from 199x199 (type each layer output like 199x199 > 197x197...)

* We need to perform 100 iterations. According to the formula :   
  OUTPUT SHAPE = [(N+2P-F/ S) + 1]

N = Input Image Size,

P = Padding

F = Filter Size

S = Stride

For our case, N = 199 (initially) , F = 3 and considering P=0 and S=1, we get output   
 Shape of 197. I.e. a decrease of 2

And if we continue doing the same operation continuously with decreasing size, we  
 Will start getting shapes like 197, 195, 193 and so on

Hence the 100 iterations are as below -

199, 197, 195, 193, 191,

189, 187, 185, 183, 181,

179, 177, 175, 173, 171,

169, 167, 165, 163, 161,

159, 157, 155, 153, 151,

149, 147, 145, 143, 141,

139, 137, 135, 133, 131,

129, 127, 125, 123, 121,

119, 117, 115, 113, 111,

109, 107, 105, 103, 101,

99, 97, 95, 93, 91,

89, 87, 85, 83, 81,

79, 77, 75, 73, 71,

69, 67, 65, 63, 61,

59, 57, 55, 53, 51,

49, 47, 45, 43, 41,

39, 37, 35, 33, 31,

29, 27, 25, 23, 21,

19, 17, 15, 13, 11,

9, 7, 5, 3, 1

1. How are kernels initialized?

* *“ We almost always initialize all the weights in the model to values drawn randomly from a Gaussian or uniform distribution. The choice of Gaussian or uniform distribution does not seem to matter very much, but has not been exhaustively studied. The scale of the initial distribution, however, does have a large effect on both the outcome of the optimization procedure and on the ability of the network to generalize ”*

Reference — Page 302, [Deep Learning](https://amzn.to/3qSk3C2), 2016.

But more sophisticated approaches like Xavier Initialisation, Kiaming He initialisation have been developed over the past few years and most of the frameworks like PyTorch, TF etc use Kiaming He Initialisation by default when activation function is ReLU, for TanH; Xavier Initialisation has been a standard practise

Talking about Random Initialisation, they help in preventing our optimisation algorithm from being stuck in a local optimum. Since initial values of kernel would be different every time, it acts like a Non-Deterministic algorithm and in order helps us to find a global optima (possibly best solution)

The algorithm uses randomness in order to find a good enough set of weights for the specific mapping function from inputs to outputs in the data that is being learned. It means that a specific network on our specific training data will fit a different network with a different model skill each time the training algorithm is run.

1. What happens during the training of a DNN?

* Training a neural network mathematically is just finding a best fit/curve - an optimal mathematical expression which has optimal weight parameters. We ideally want our optimisation algorithm to reach the Global Minima to minimise the error rate.

Let’s say we want to design a neural network that can recognize if a picture is a picture of a dog or a cat, then given an input image, the neural network will try to predict whether the picture is a dog or a cat. With the result of the prediction, the neural net will compute the error between the result and the expected result. This error will be back propagated through the network layers and will evolve the parameters of layers in order to reduce the error for the next predictions.

Example -

x = [1, 2, 3, 4, 5, 6] and

y = [1, 4, 9, 16, 25, 36] then based on this our mind can say that this is a function of   
***Y = X^2***

So the next time someone says that INPUT is 8, show me the output, we would calculate that Y = (8)^2 = 64 and that is the answer

But what would happen when our data is based on image and complex values, we cant do that manually so that is where Neural Nets come in. The same procedure as mentioned above happens and it finds a curve/equation let say hypothetically

y = x^1.2 + x^5 - x^2.5 + x^7 ..... etc

The values (1.2, 5, -2.5, 7....) etc are coefficients which our neural network learn and they are SAVED as WEIGHTS

And the way how we find this optimal coefficients/parameters is as mentioned above  
 Learning through backpropagation

So during inference/testing, whatever value we give, that will be replaced instead of "X" in the above equation and that's how we get the output