**EVA 5 – Assignment-1 :**

1. What are Channels and Kernels (according to EVA)?

A channel is a container holding a specific set of information with respect to the image. Each channel is a collection of kernels which has a unique feature information. A colour image will have 3 channels (R,G,B) where each will have that colour specific information, similar way we can have multiple channels for based on multiple features in the image.



A Kernel or Feature Extractor or Filter is a N x N matrix(usually 3x3) which will be convolved over the input image. The output from each of these kernels will form a new pixel in next layer(Channel) of the image.

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

* **Why always odd (N x N) size kernel is preferred:**

For an odd-sized filter, all the previous layer pixels would be symmetrically around the output pixel. Without this symmetry, we will have to account for distortions across the layers which happens when using an even-sized kernel.



Therefore, even-sized kernel filters are mostly skipped to promote implementation simplicity. If you think of convolution as an interpolation from the given pixels to a center pixel, we cannot interpolate to a center pixel using an even-sized filter.



* **Why 3 x 3:**
* A 1 x 1 kernel would interact only with one pixel on the input image and not with the neighbouring pixels, the output pixel generated by this operation will not hold any information about neighbouring pixels which is not useful.
* A 5 x 5 kernel at once will cover more pixels from the input image, so specific/tiny features may be ignored. Also, 5x5 will use more kernel weights(5\*5 = 25) than a kernel size of 3x3 ([3\*3] +[3\*3] = 18). The 5x5 convolution results in1.4x heavier kernel than the two stacked 3x3 convolutions which may cause overfitting.



* Hence a 3x3 kernel would give lower number of weights and helps capture smaller, complex features in the image.

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

100 times.

199x199 > 197x197 > 195x195 > 193x193 > 191x191 >

189x189 > 187x187 > 185x185 > 183x183 > 181x181 >

179x179 > 177x177 > 175x175 > 173x173 > 171x171 >

169x169 > 167x167 > 165x165 > 163x163 > 161x161 >

159x159 > 157x157 > 155x155 > 153x153 > 151x151 >

149x149 > 147x147 > 145x145 > 143x143 > 141x141 >

139x139 > 137x137 > 135x135 > 133x133 > 131x131 >

129x129 > 127x127 > 125x125 > 123x123 > 121x121 >

119x119 > 117x117 > 115x115 > 113x113 > 111x111 >

109x109 > 107x107 > 105x105 > 103x103 > 101x101 >

99x99 > 97x97 > 95x95 > 93x93 > 91x91 >

89x89 > 87x87 > 85x85 > 83x83 > 81x81 >

79x79 > 77x77 > 75x75 > 73x73 > 71x71 >

69x69 > 67x67 > 65x65 > 63x63 > 61x61 >

59x59 > 57x57 > 55x55 > 53x53 > 51x51 >

49x49 > 47x47 > 45x45 > 43x43 > 41x41 >

39x39 > 37x37 > 35x35 > 33x33 > 31x31 >

29x29 > 27x27 > 25x25 > 23x23 > 21x21 >

19x19 > 17x17 > 15x15 > 13x13 > 11x11 >

9x9 > 7x7 > 5x5 > 3x3 > 1x1

1. How are kernels initialized?

Kernel or filter elements of the layers are usually initialised by random values as weights. The weights are chosen such that the values are normally distributed along the mean(gaussian distribution). This normal distribution will prevent layer activation outputs from exploding or vanishing during the course of a forward pass in a NN.

* ***xavier\_uniform or kaiming\_uniform*** initialization techniques are commonly used in torch convolution for kernel weights initialization.

1. What happens during the training of a DNN?

During training of Deep Neural Networks, the goal is to predict weights for input features such that the loss calculated between actual and predicted output is minimum. A DNN consists of an input layer, n x hidden layer, final output layer. On a forward propagation, the each input values will be multiplied with corresponding weights on the multiple layers and produce an output. The difference between actual and predicted value is loss.

The value of this loss is then passed backwards through these layers and used to adjust the values of the weights to effectively minimize the loss. This way the value of the weights are adjusted during training and system is said to have converged when the loss is minimized.



In the above image in order to achieve lower loss function(J(theta0,1)), an optimized weights(theta0,1) will be chosen by Neural Network via SGD and Back propagation. In a DNN architecture, multiple hidden layers are used