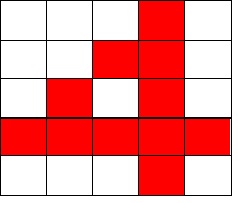
**1.What are Channels and Kernels (according to EVA)?**

Channels: Contextually specific information that are grouped and held together in a container is called channel. The channel contains collection of specific features, colors, frequency spectrum, Audio track etc.

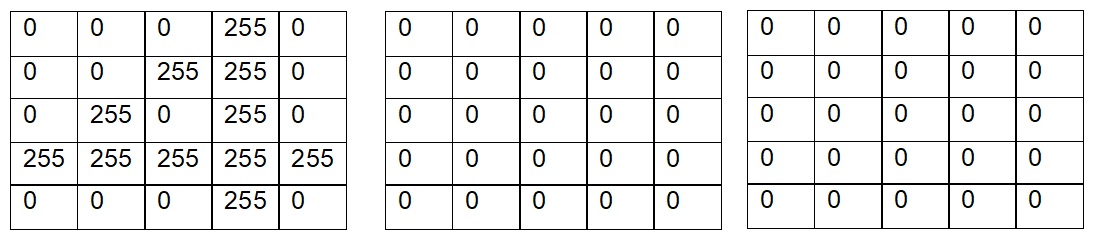
For example:

1. RadioCity is a radio station whose frequency modulation is 91.1 Hz. This frequency is allocated to RadioCity Channel.
2. When recording a music, the sounds of different musical instruments are collected in a separate container called track and later mixed to combine them to form a music. The track which contains specific sounds of musical instrument are called Channel.

Coming to an Image, the image is a combination of light intensity and color information. So, the colors are separated out and grouped with their intensity in a container called channel.



 5X5 Image of number 4



            Red Channel                                Green Channel                             Blue Channel

Channels are also called as feature maps (feature map name itself suggests that it is a collection of features). Convolution operation over input by a kernel/filter gives a single channel as output. So if there are n filters convolve over input will get n channels as output. (We apply some non-linearity function like ReLU on top of it)

So Channel is a representation of an image which has features (like edges/ gradients/ patterns/ textures / objects).

Kernel: A filter that is used to extract specific information from given input is called Kernel.

  The tea filter shown in above figure extracts the tea essence while separating out the tea residues. Same way to extracts features from image we use filters called Kernels.

Image will have features for example, like thin eyebrow, blue pupil, silky hair etc. made up of edges, gradient, texture etc. Kernels are used to extract these features from given image. Like how tea filter has holes to filter tea essence from tea residue,

In conventional image processing techniques Kernel or filters tailored manually but whereas in CNN, network learns automatically through optimization algorithms (gradient descent) in the back propagation step.

Example of conventional image processing filter is - Sobel filter which helps to find out the edges of the input image by calculating x-gradient and y-gradient & find the resulting gradient’s magnitude from them and it’s direction (angle - theta).

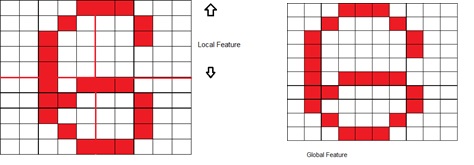
Kernel/Filter convolve over input (image or representation of image [hidden layers activations]) and gives the output channels/feature\_maps.

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



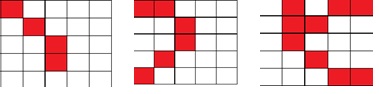
Images are made up of edges, gradients, texture, patterns and object. The edges are the basic building blocks of an image. The small portion of these edges put together is called **local feature** in a image.

Example



In the first image, the 10 x 10 image of number 6 is divided into 4 quadrants.  Here each quadrant represent a "Local feature". In second image, the entire edges of number 6 is called Global feature.

With local feature extraction we can identify granular feature which makes easier to extract similar patterns of features. For example, with local feature shown below



we could extract features of 9,8,0 etc. With global feature “6” we can only extract “6” present in any other image. Hence extracting local feature makes the architecture more robust.

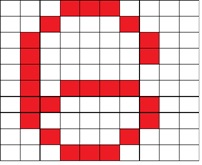
To extract local features, we look at small group pf pixels which is called “receptive field”. With smaller receptive field the amount of information of feature extracted is more. These features are useful in later layers when forming an object. Hence, we try to keep kernel size as low as possible to extract local features.



Let us say there is an image of size 100 x 100 which has many small objects inside the image, if you apply a large size kernel say 20x20, it misses the information of these small objects. But if you try with smaller kernel size it will capture the information of small objects as well as it convolves over a small region of the image again and again.

And, when we chose large kernel size, each convolution operation has lot of common region (common receptive field) where it is convolving when stride is also less. So many duplicate operations.

Weight initialization:



For the input image size of 10 x 10 of number 6, we would like to have a filter of size 3x3 and 5x5 for comparison. The function of this filter to identify horizontal edges.

When we initialize 3X3 and 5X5 matrix to a random value at the begin of training say,

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 4 |
| 8 | 8 | 8 |
| 6 | 7 | 2 |

3 x3 matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 5 | 4 | 3 | 2 | 1 |
| 6 | 6 | 6 | 6 | 6 |
| 9 | 8 | 3 | 2 | 4 |
| 3 | 2 | 9 | 7 | 7 |

5 x 5 matrix

After back propagation and the weight adjustments, we can see that smaller matrix will have better weight shared among them compared to a bigger kernel. Hence lower kernel size is computationally efficient. Because of low kernel size, multiple kernels must be used to learn much more features, thus more layers are created. With more number of layers we can learn more complex features.

So why not   1 X 1  ?

Kernel size of 1X1 will basically extract each pixel feature, which is fine grained with no information from neighboring pixel. So 1X1 is not used extensively.

So why not 2x2 ?

we could use 2x2 kernel. But 2 is a even number and when a kernel of size with even number is convoluted over a image, the resulting feature map will not be symmetric around the destination pixel. This will lead to distortion and will have to do padding to overcome distortion.  For example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

       5x5 image of line

Convolute with 2x2 filter

|  |  |
| --- | --- |
| -1 | 0 |
| -1 | 0 |

The resultant feature map is 4x4

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | -2 | 0 |
| 0 | 0 | -2 | 0 |
| 0 | 0 | -2 | 0 |
| 0 | 0 | -2 | 0 |

The resultant feature map is not symmetric.

In case of odd number kernel size,

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |

the resultant feature map

|  |  |  |
| --- | --- | --- |
| 3 | 0 | -3 |
| 3 | 0 | -3 |
| 3 | 0 | -3 |

We can see that resultant feature map is symmetric across center and their will not be distortion in the feature map.

When we do convolution operation by a kernel over an image, the image size(height and width) reduces gradually unless we do padding on the input. Here reducing means we are throwing out some information, so with padding we can maintain same size as input with 3x3 but can’t with 2x2.

Example:

Output of convolution formulated as (n-k+2p)/s +1n - input dimension k - kernel size p - padding s – stride

Let us say we have 10x10 input, 2x2 filter, padding=1, stride=1

Output dimension - (10-2+2)/1 + 1 = 11. Here we got 11 but we need 10 which is size of our input

Let us say we have 10x10 input, 3x3 filter, padding=1, stride=1

Output dimension - (10-3+2)/1 + 1 = 10. Here we got same size as input

Hence 3X3 is always chosen to be working perfect in most cases. 

**3. 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...)**

Output of the convolution is given by the formulae  (n-k+2p)/s +1 where n is the size of input image, k is the size of kernel, p is number of padding and s is number of strides.

Out input image is of size 199, kernel is of size 3, padding is 0 and stride is 1. we get the output size to be 197.

Lets do this till we get 1x1

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

Total we have 99 convolution operations to reach 1x1

**4. How are kernels initialized?**

The basic functionality of kernels are to extract features out of given image. kernels at the beginning do know what features to extract. It does not know how to extract edges, gradients, texture, patterns and objects. The kernels learn to extract these features over the period of training. The kernels in the beginning extracts some feature which does not even come close to the actual feature it has to extract. So the difference between the actual and the predicted values which is called as "loss" is calculated. The loss is back propagated and weights of the kernels are adjusted so that in the next iteration, the kernel learns to extracts correct features and the loss becomes minimum.

The effective learning of kernel highly depends upon the values with which its weights are initialized and the activation function that is been used in the network to excite the neuron. Any wrong set of weight initialization could cause the activation function to explode the gradient or vanish the gradient of the loss. If the loss function explodes or vanishes, the network will stop learning. Hence **kernel initialization depend on activation function that we use.**

When using Relu activation, the kernels are initialized using "Kaiming He" initialization and when using tanh activation, Kernels are initialized using "Xavier" initialization.

**You Could use yours and remove mine**

**5. What happens during the training of a DNN?**

When training a DNN, A function that represent f(input\_parameters) = output is derived. This function have weights and bias parameter to be learnt. During the training weights and bias are found using the loss functions, optimization methods under the process called Back propagation.

There are two path that is been taken during training.

Feed forward Path:

1. The depth and width of the neural network is defined.
2. Each neuron are associated with weights and are initialized.
3. The input is passed through the root node and at the output node it produces probability of that output class.
4. Based on the predicted probability, loss is calculated.

Backward propagation Path:

1. The gradients of loss with respect to change in weight is calculated.
2. Weights are adjusted with the calculated change in weights with respect to loss.

Single iteration of feed forward and backward propagation path happen till completion of all the inputs and it is called as epoch.

Several epochs are run till the loss get minimized and the model is learnt efficiently.

Once the network outputs with minimum loss, the weights and bias of the network forms the function that produces output for given input parameters.