Deep learning

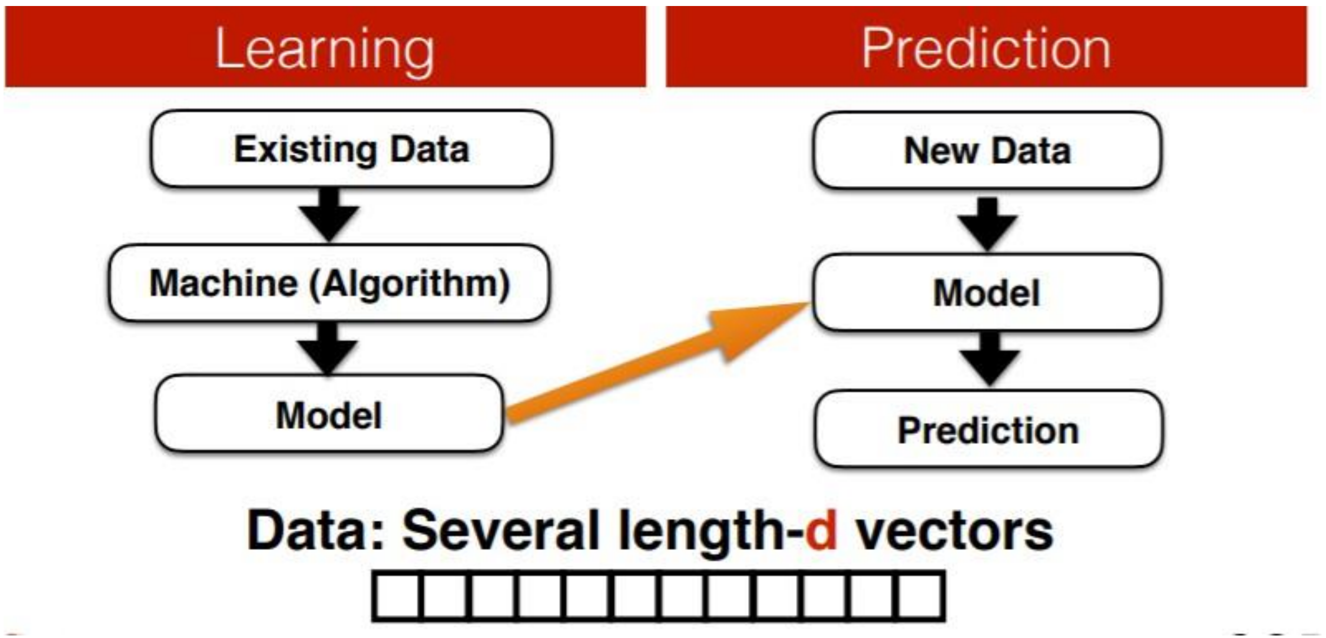
What is deep learning?

- Deep learning is a type of machine learning that uses artificial neural networks to learn from data.
- Deep learning algorithms are typically trained on large datasets of labeled data.

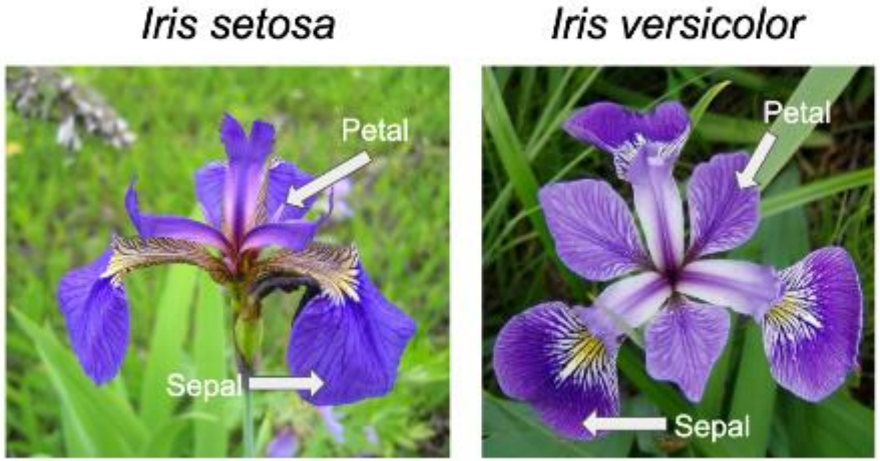
Why deep learning?

- Deep learning is a subset of machine learning involving neural networks with many layers
    - **Can learn complex relationships between features in data**: This makes them more powerful than traditional machine learning methods.
    - **Large dataset training**: This makes them very scalable, and able to learn from a wider range of experiences, making more accurate predictions.
    - **Data-driven learning**: DL models can learn in a data-driven way, requiring less human intervention to train them, increasing efficiency and scalability. These models learn from data that is constantly being generated, such as data from sensors or social media.

What machine/ Deep learning does?



Example

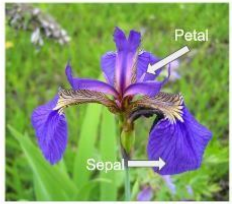


Labelled Dataset

Iris Flower species

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	**versicolor**
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	**versicolor**
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	**versicolor**
10	4.9	3.1	1.5	0.1	**versicolor**

“Traditional” machine learning:



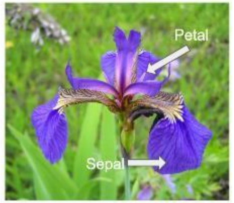
handcrafted features

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

learned classifier

Setosa

Deep, “end-to-end” learning:



learned low-level features

learned mid-level features

learned high-level features

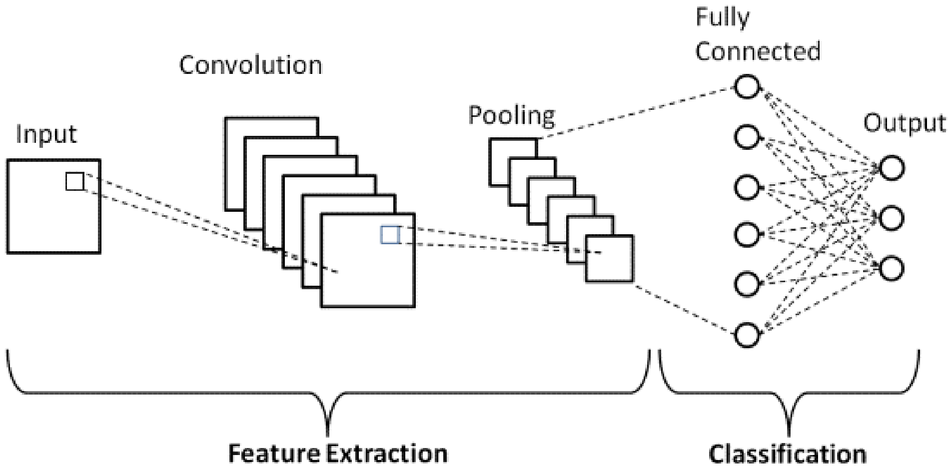
learned classifier

Setosa

Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN)

- Deep Learning algorithm specially designed for working with Images and videos.
- Can be used for sequential data handling/ handling other types of data
- The CNN model works in two steps:
  - **Feature Extraction (convolution and pooling)**
    - Filters are applied to the images to extract the information
    - layers
  - **Prediction (Classification / Regression**)
    - A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.



https://training.galaxyproject.org/training-material/topics/statistics/tutorials/CNN/slides.html#20

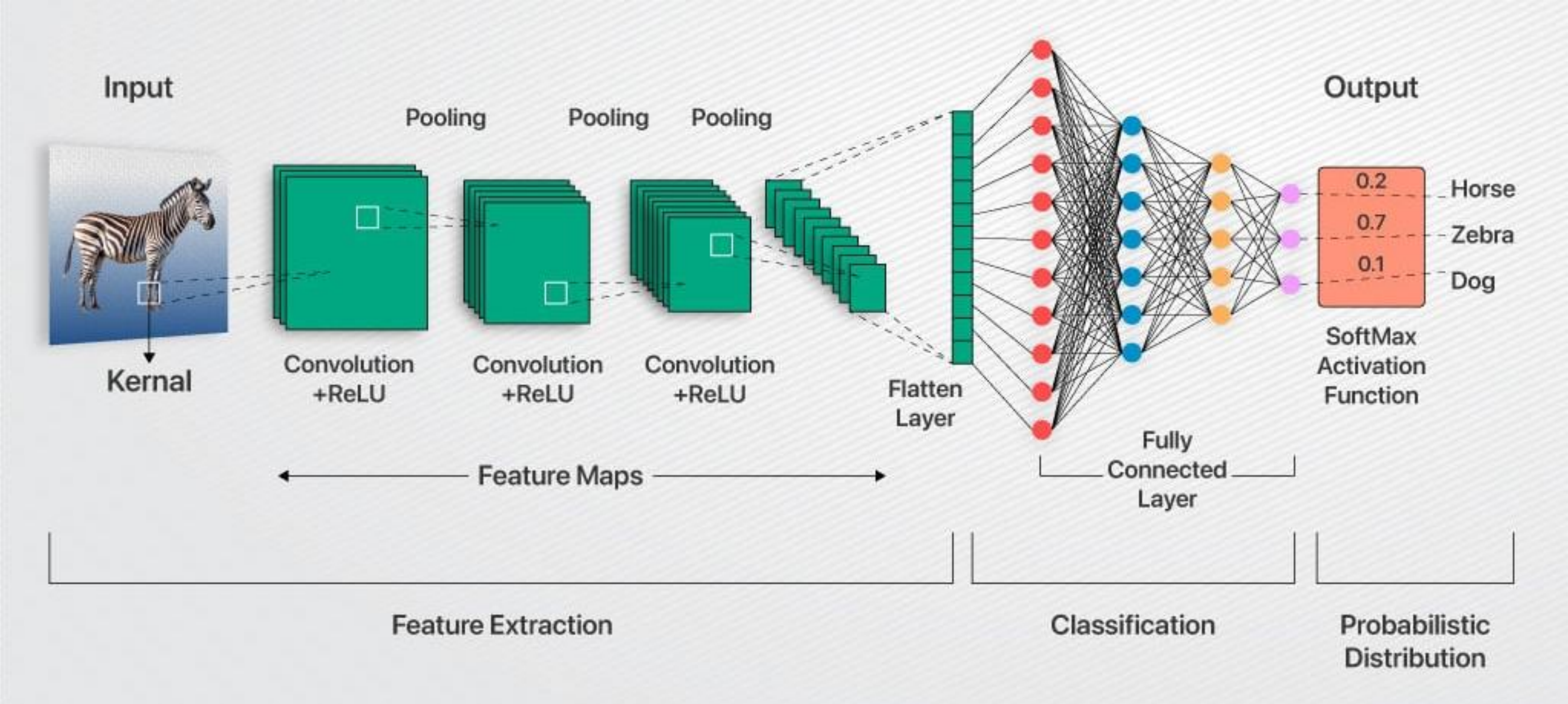


Image courtesy:-https://www.analytixlabs.co.in/blog/convolutional-neural-network/

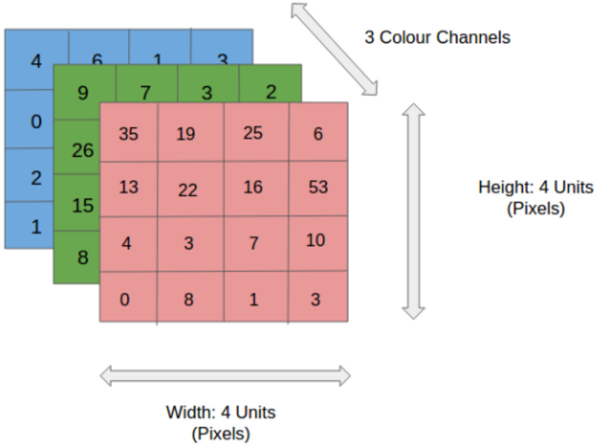
Building blocks of CNN

- **Input layer**
- **Convolution Layer**
- **Pooling layers**
- **Flattening Layer**
- **Fully connected layer**

1. Input layer

- The input layer in a Convolutional Neural Network (CNN) is the entry point for the raw data.
- It serves as a container for the input (ex. image).
- This input is typically a numerical representation of the image, often in the form of a three-dimensional matrix (height, width, and color channels)

- Sample input



Images?

```
>> I=im2bw(I)

I =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     1     1     0     0     0     0
     0     1     0     0     1     1     0     0     0     0
     0     0     1     1     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

```
>> I=imresize(I, [10 10])

I =

   47   49   50   54   53   53   51   49   47   43
   48   50   49   54   55   54   54   60   49   45
   48   55   71   83  137   94   87  104   50   53
   46   75  116   82  104  105   90   86  108   85
   49   75   67   78  153  166  116  122  111   82
   95  141  116   93  178  164   97   84   66   47
  110  122  155  130  108   99   88   70  131  109
  106   89  111  127   74   69   57   59  182  181
   56   47   65   48   62   69   68   63   64   64
   73   60   97   86   50   48   57   51   42   37
```

```
>> I=imresize(I, [10 10])

I(:,:,1) =

    66    68    68    72    72    71    69    68    65    62
    66    68    66    72    73    71    72    77    67    63
    66    71    85    97   157   111   104   123    67    71
    65    81   139   102   142   118    88   120   191   155
    71   103   128   164   215   210   181   223   236   180
   152   215   169   171   251   249   186   199   180   128
   174   188   240   197   182   131   162   180   197   149
   205   169   144   159    81    67    97   123   207   197
   133   109    84    67    81    90    88    82    83    83
   101    82   120   107    67    67    79    71    60    53
```

```
I(:,:,2) =

    34    36    37    40    40    40    38    36    34    31
    35    37    36    41    43    41    43    50    36    33
    35    45    69    77   131    95    87   108    41    44
    32    80   121    80    96   115   104    84    83    60
    35    70    43    42   143   170    97    81    64    44
    79   128   101    64   173   150    64    33    15     9
    95   108   124   108    84    95    62    21   102    89
    67    60   107   126    75    73    37    29   174   176
    22    19    52    36    47    53    53    49    51    51
    54    45    77    68    37    35    43    37    30    26
```

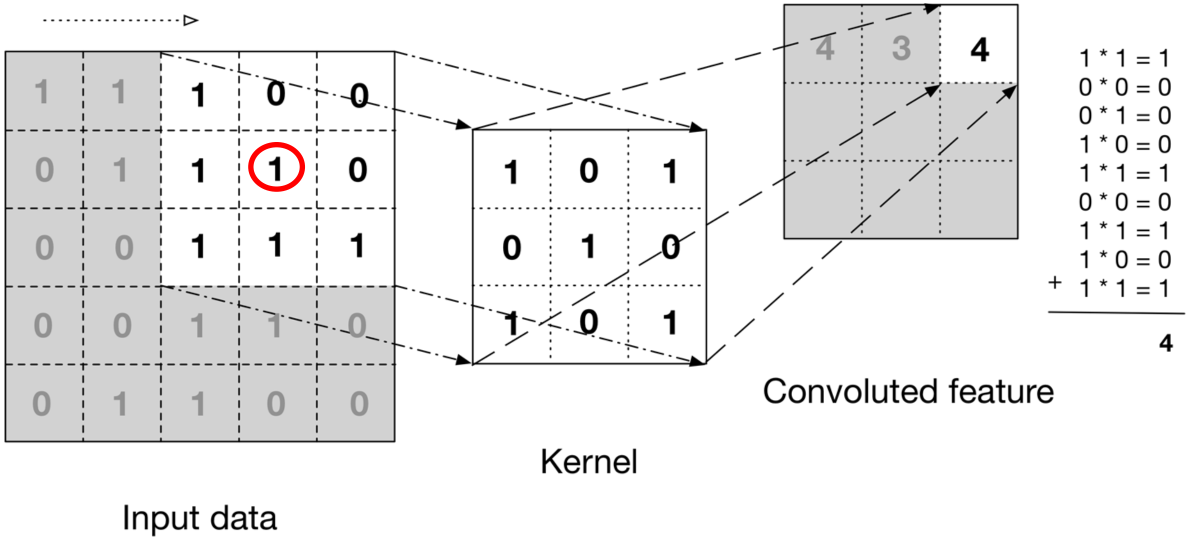
```
I(:,:,3) =

    65    68    68    72    71    70    69    68    64    61
    67    70    71    74    74    74    70    65    66    63
    67    61    48    75   117    41    41    30    54    53
    67    37    31    37    47    20    22     9    19    23
    60    26    31    40    37    26    46    68    30    18
    31    15    52    43    16     8    34    44    30    26
    20    24    88    65    36    33    32    30   105   101
    48    32    50    52    56    54    47    46   160   168
    30    29    86    62    85    94    94    85    76    76
   101    84   139   122    73    65    75    69    61    54
```

2. Convolution Layer

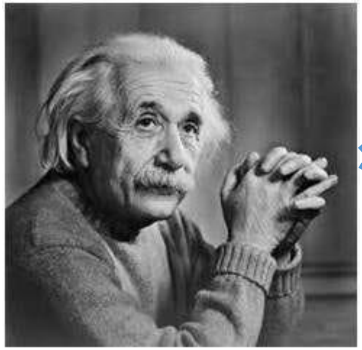
- Applies filters (kernels or masks) to the input image to extract features.
    - Involves sliding a kernel over the input image to extract relevant response
- Detects patterns like **edges, corners, and textures**.
- Uses activation functions (ReLU, sigmoid, etc.) to introduce non-linearity.

Convolution

Applying filters to extract features



Filters : What they do?



1	0	-1
2	0	-2
1	0	-1

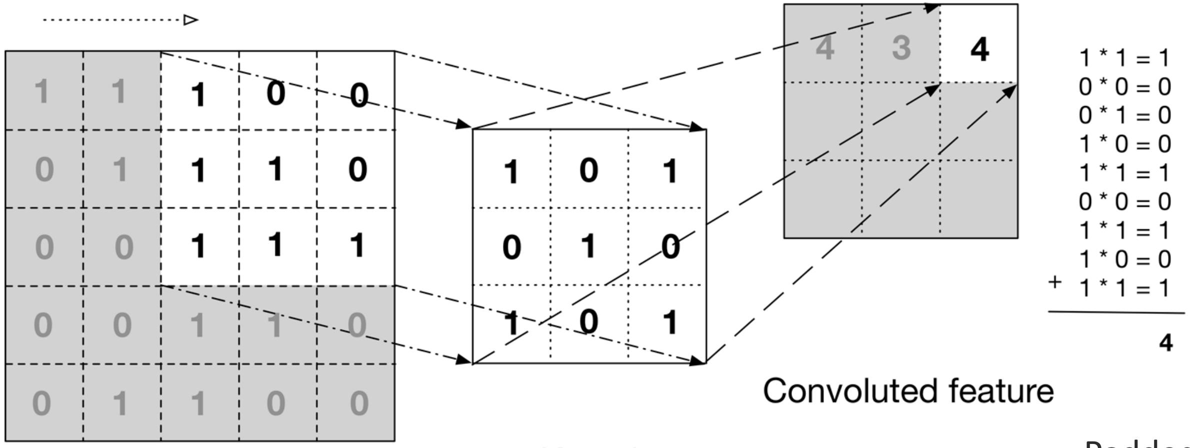
Sobel Vertical

Vertical edges highlighted

1	2	1
0	0	0
-1	-2	-1

Sobel Horizontal

Horizontal edges highlighted



Convoluted feature

Kernel

Input data

Convolution parameters

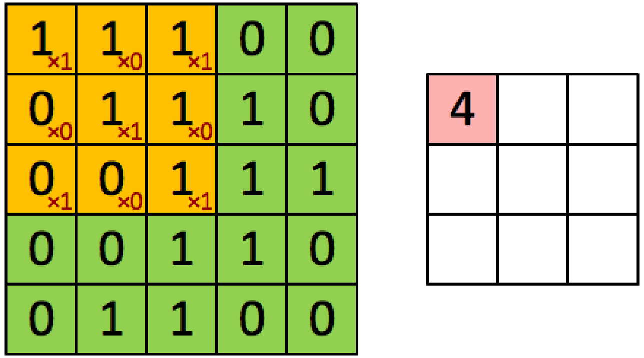
- Filter/ Kernel size
- Padding
- Stride
- Activation function

Strides

How quickly window/ filter/kernel slides.
Stride 2 means, window moves by 2 pixels at a time.

Padded image

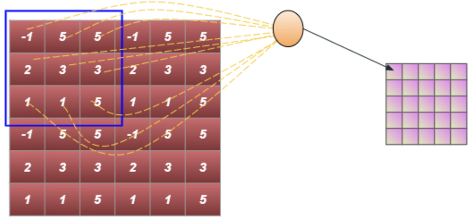
0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0



Image

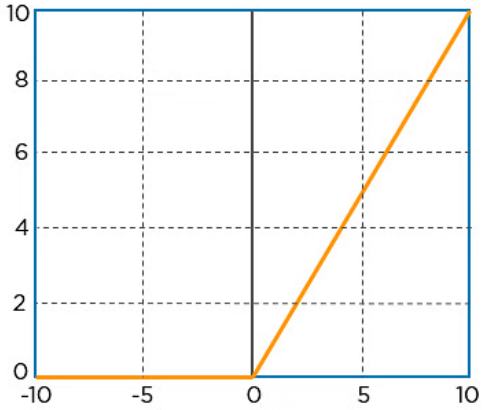
Convolved Feature

Local Receptive Field



ReLU

- Rectified linear unit.
- Performs an element-wise operation and sets all the negative values to 0.
- It introduces non-linearity to the network, and the generated output is a rectified feature map.



R(z) = max(0, z)

Input

-249	-91	-37
250	-134	101
27	61	-153

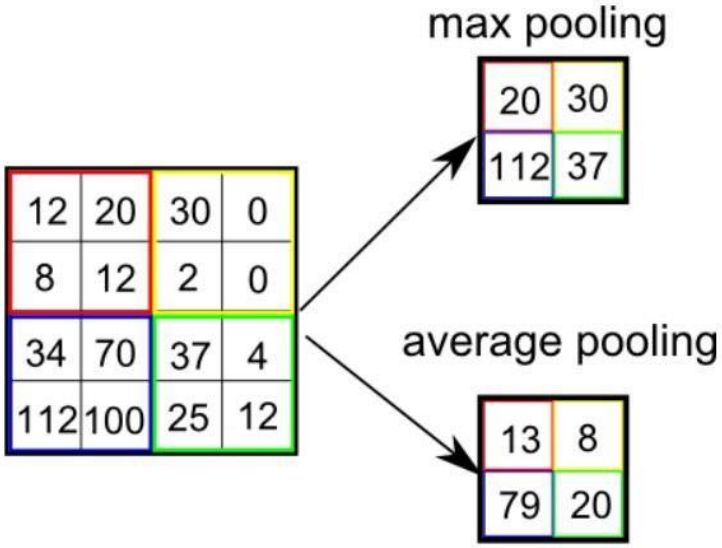
ReLU

0	0	0
250	0	101
27	61	0

Example

Pooling

- **Pooling layers**
  - Down-samples the feature maps to reduce dimensionality. Ie, it reduces the spatial size of the Convolved Feature.
  - Helps in reducing computational cost and overfitting.
  - Common types: Max pooling, average pooling.



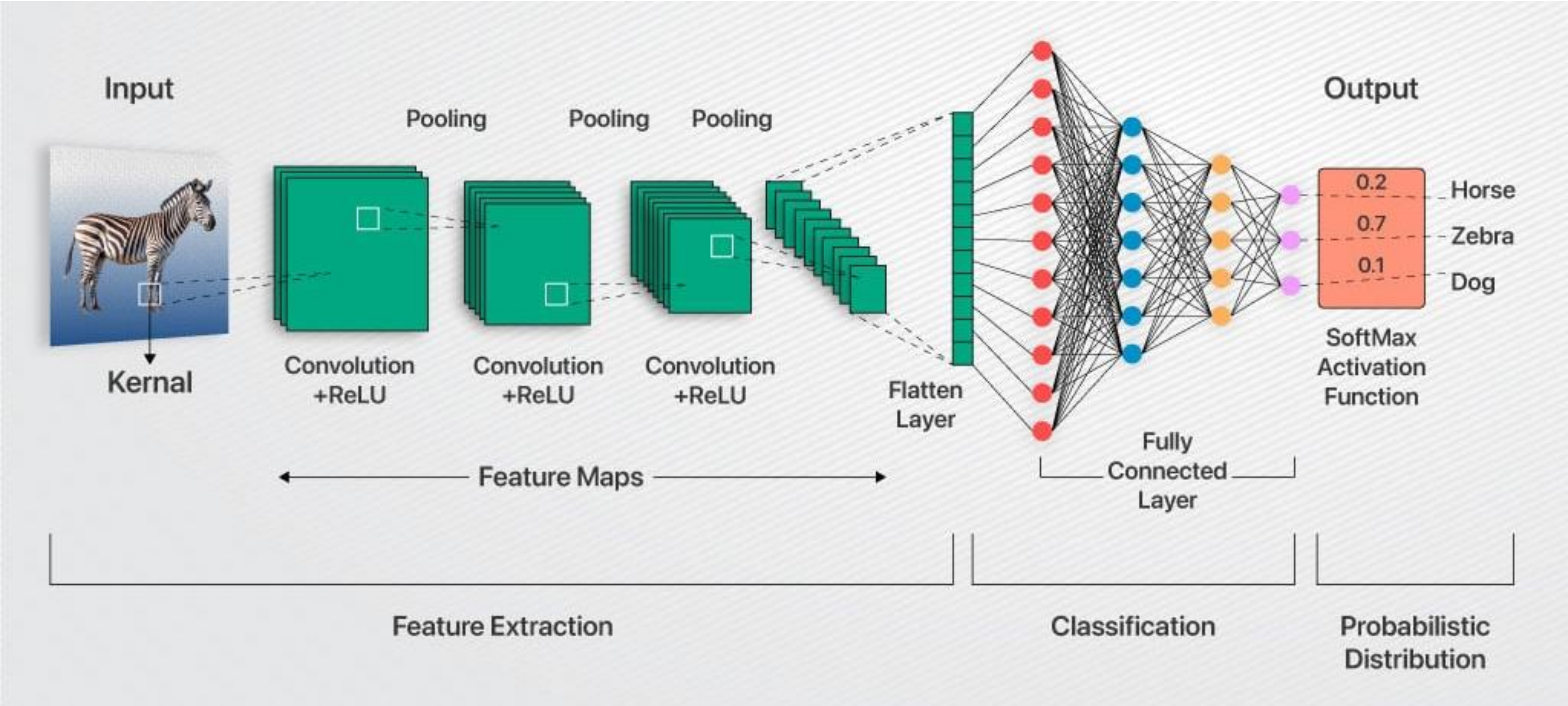
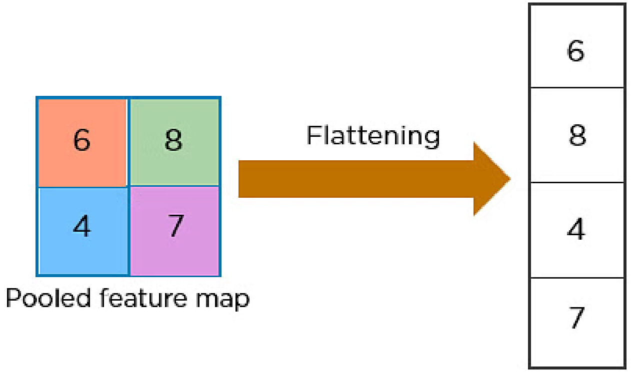
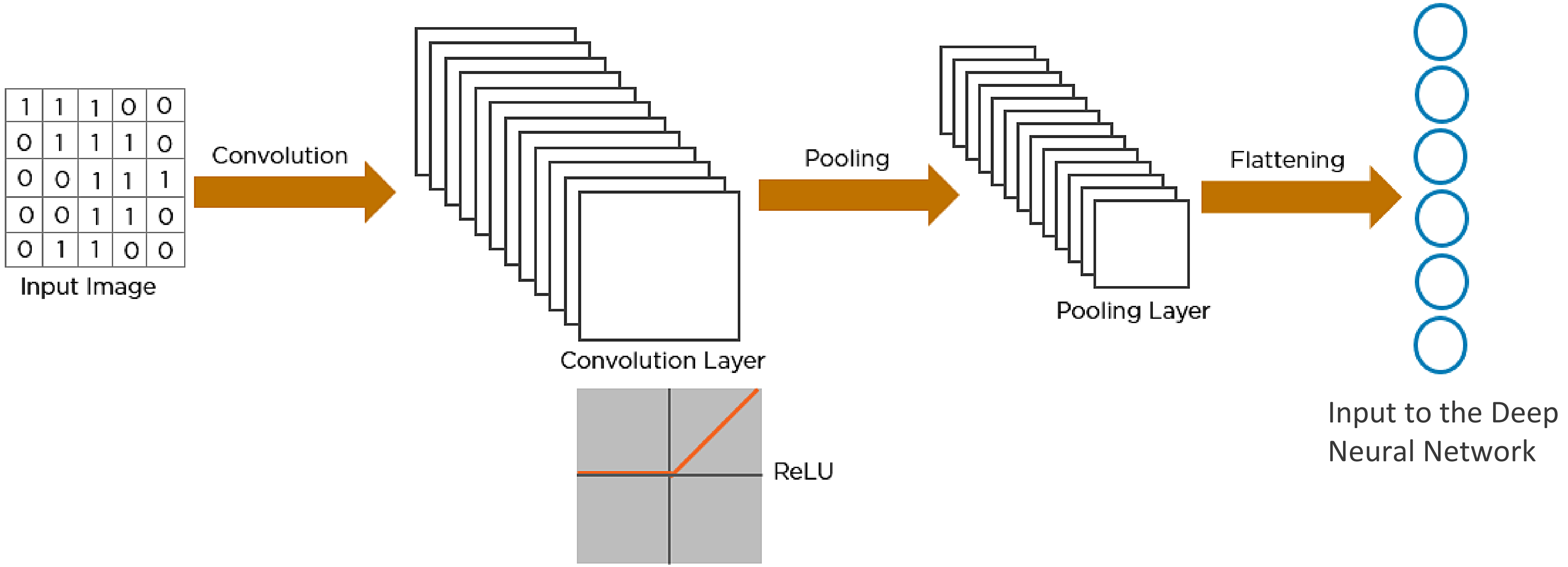


Image courtesy:-https://www.analytixlabs.co.in/blog/convolutional-neural-network/

Flattening

- **Flattening Layer**
  - Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector. Ie, Converts the 2D feature maps into a 1D vector.
  - Prepares the data for the fully connected layer





1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
Input Image
Convolution
Convolution Layer
ReLU
Pooling
Pooling Layer
Flattening
Input to the Deep
Neural Network

Fully Connected layer

- The Fully connected layer (as we have in ANN) is used for classifying the input image/ data into a label.
- This layer connects the information extracted from the previous steps (i.e Convolution layer and Pooling layers) to the output layer and eventually classifies the input into the desired label.

- Now that we have converted our input image into a suitable form for Multi-Level Perceptron, we shall flatten the image into a column vector.
- The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training.
- Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using appropriate **Classification** technique.
- While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

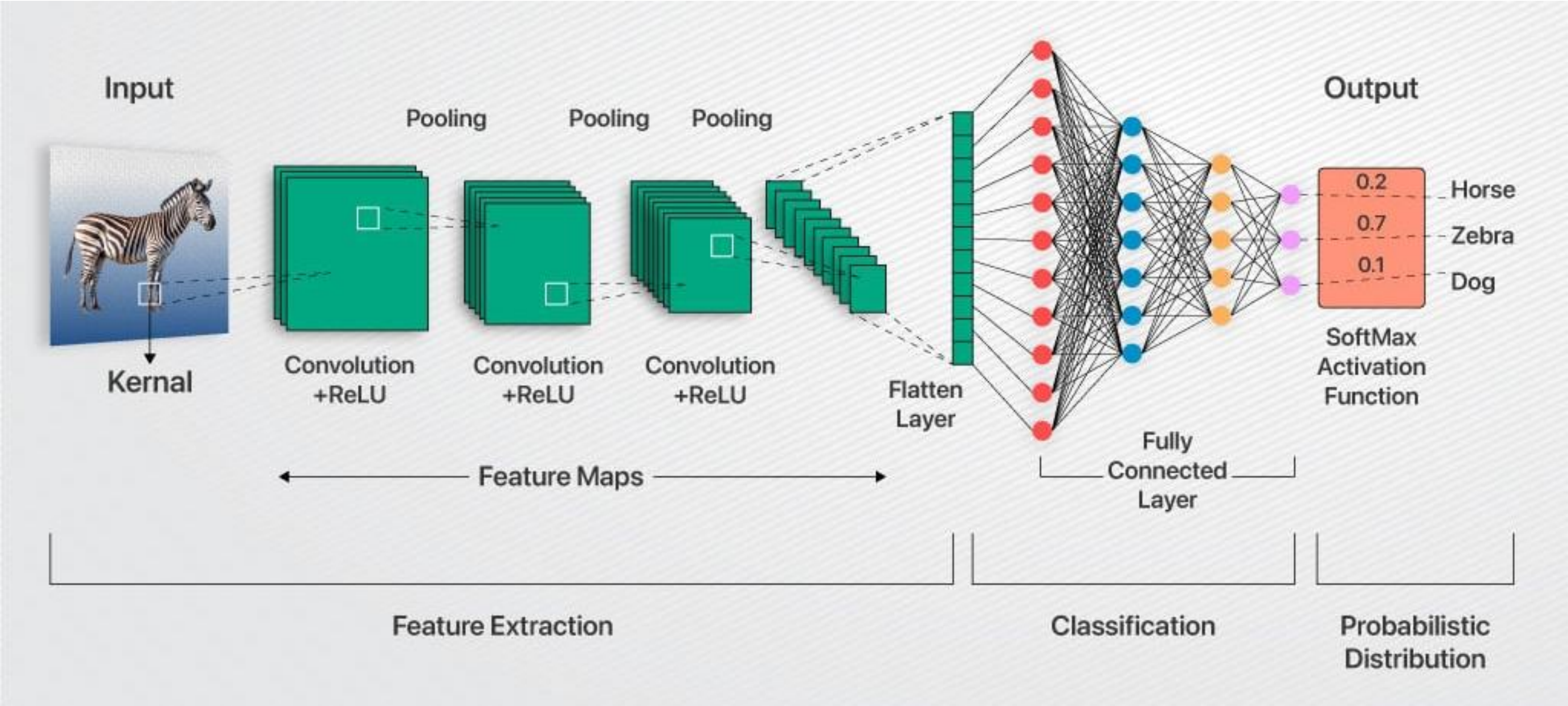
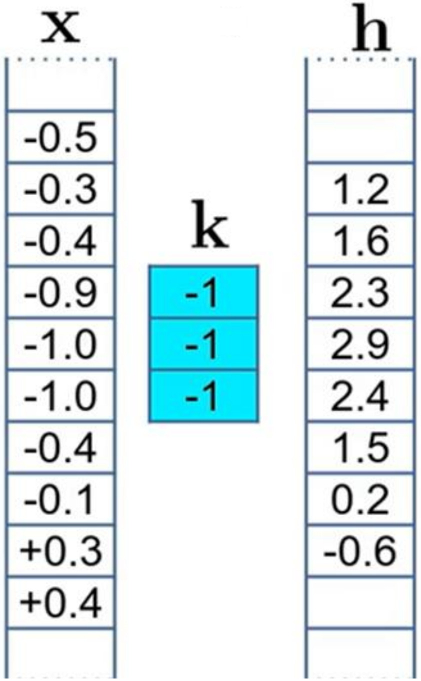


Image courtesy:-https://www.analytixlabs.co.in/blog/convolutional-neural-network/

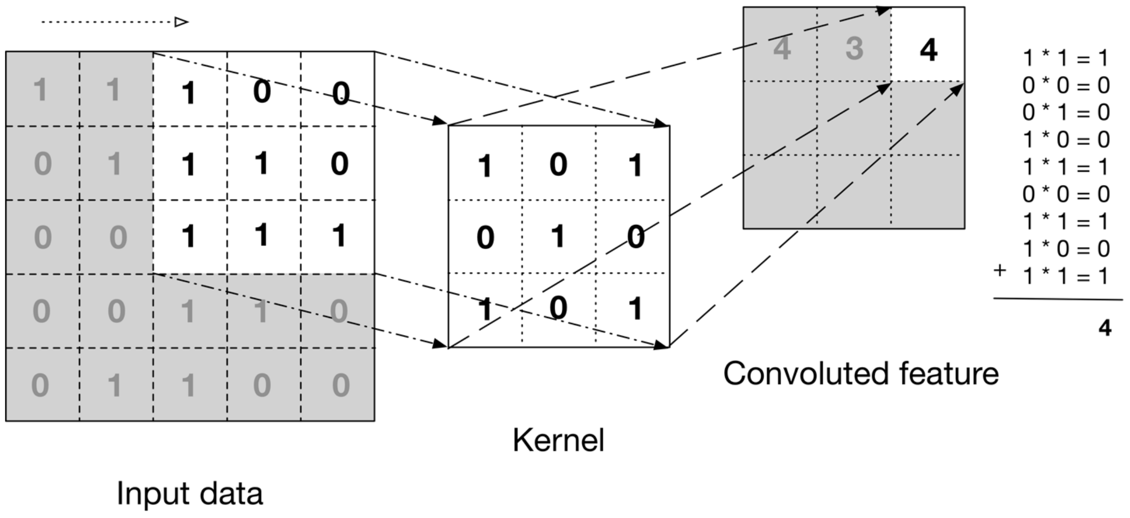
Convolution in Sequential data (1D data)

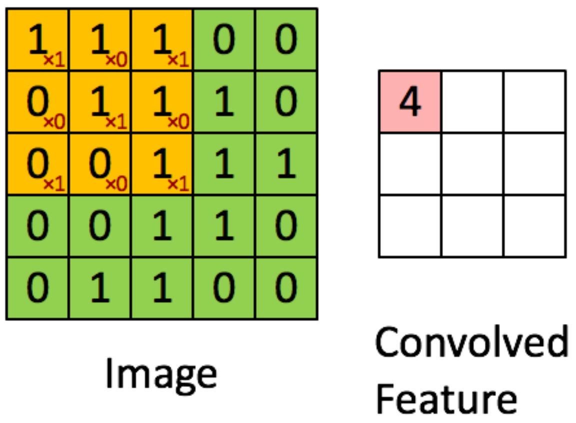
$$\mathbf{h}(t) = (\mathbf{x} * \mathbf{k})(t) = \sum_{\tau=-m}^{m} \mathbf{x}(t-\tau) \cdot \mathbf{k}(\tau)$$



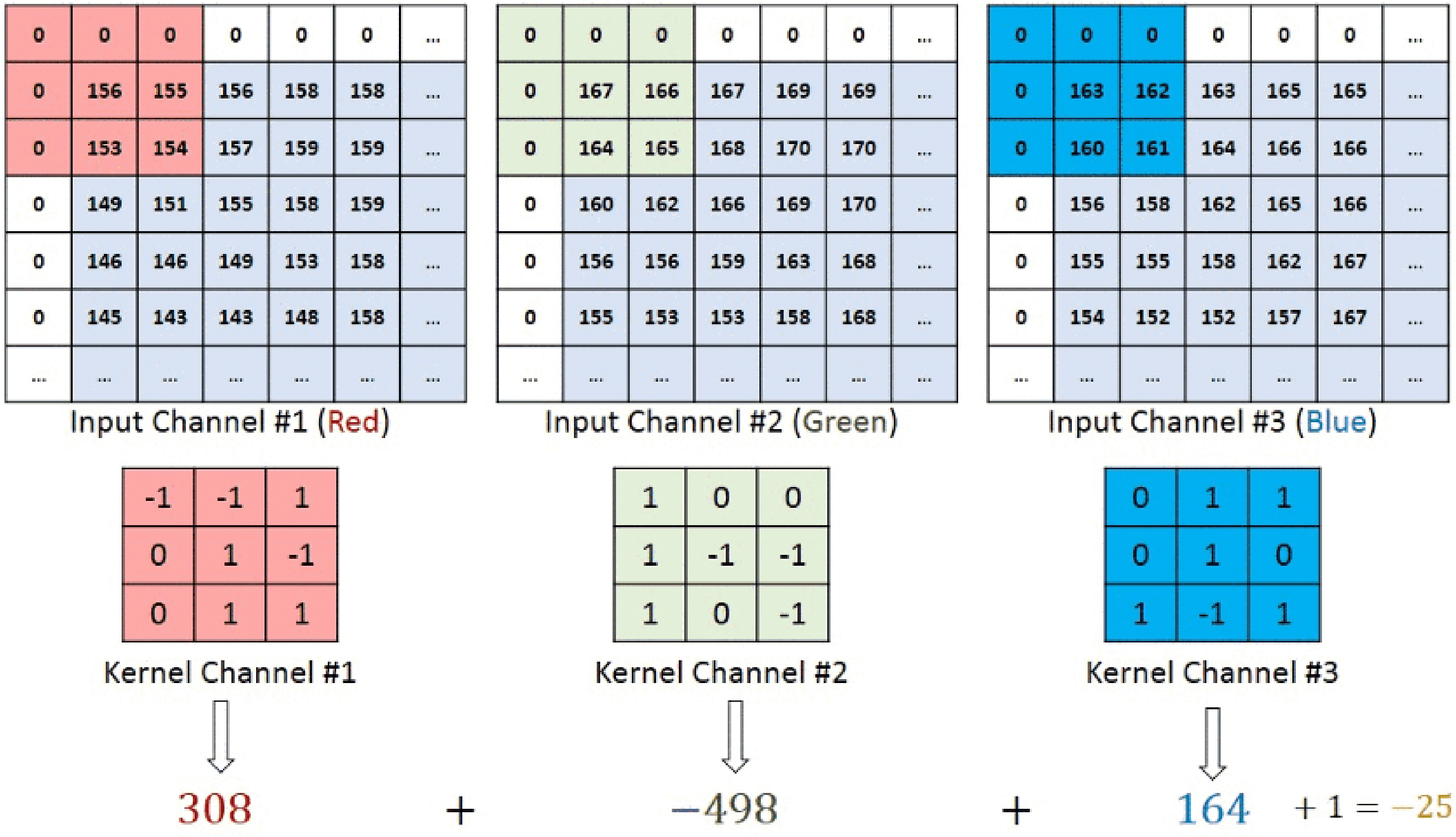
Some important Concepts

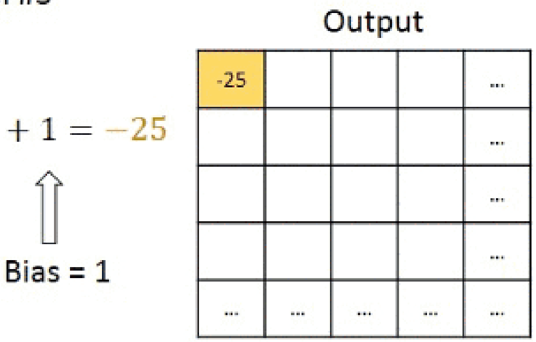
Recap: Convolution on images with 1 channel





Convolution on images with 3 channels

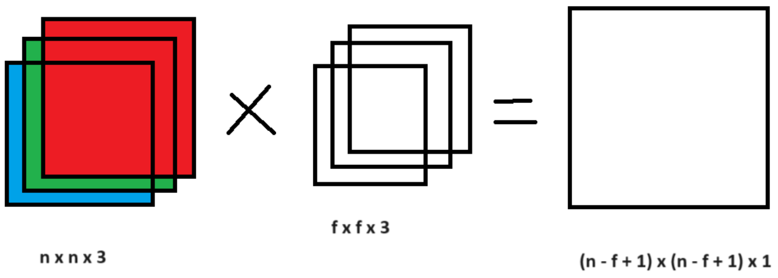




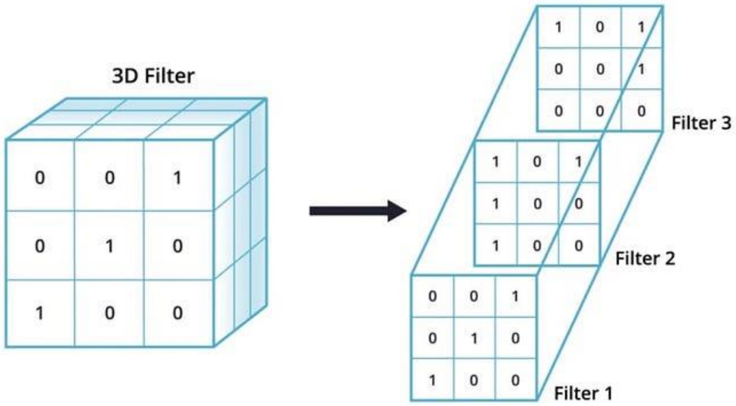
[Detailed understanding about convolution operation in a coloured image with the help of 3D kernels | by Abhishek Jain | Medium]()

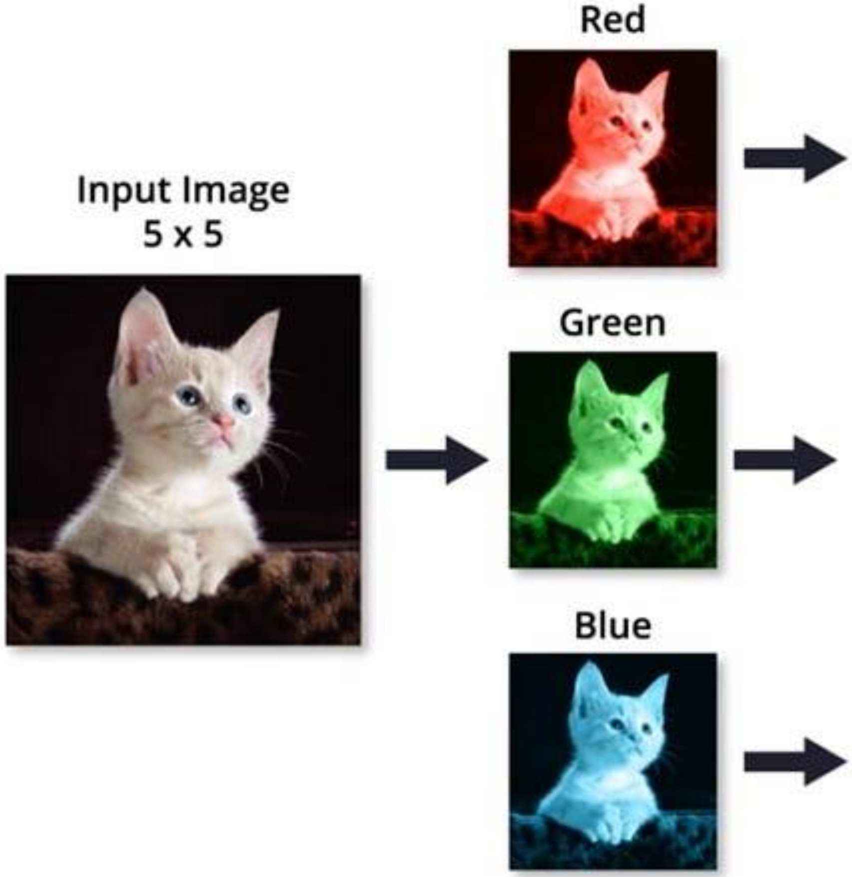
Convolution operation in a coloured image with the help of 3D kernels

- For convolution in multi channel images, the depth of the filter will be chosen to match the number of color channels in the image.
- **REMEMBER : Convolution operation only happens if the input image depth and kernels depth is same**



- This is because we’re going to convolve each color channel with its own two-dimensional filter. Therefore, if we’re working with RGB images, our 3D filter will have a depth of three.



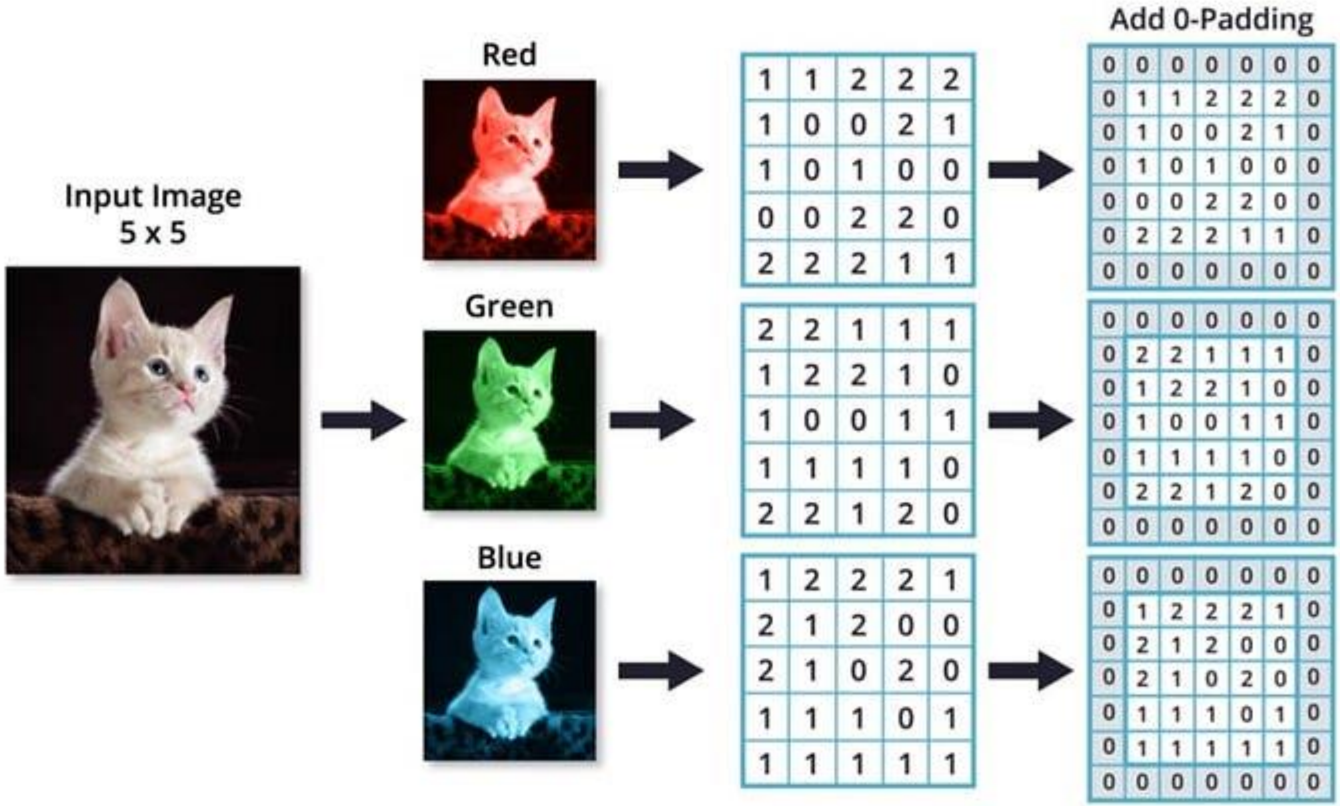


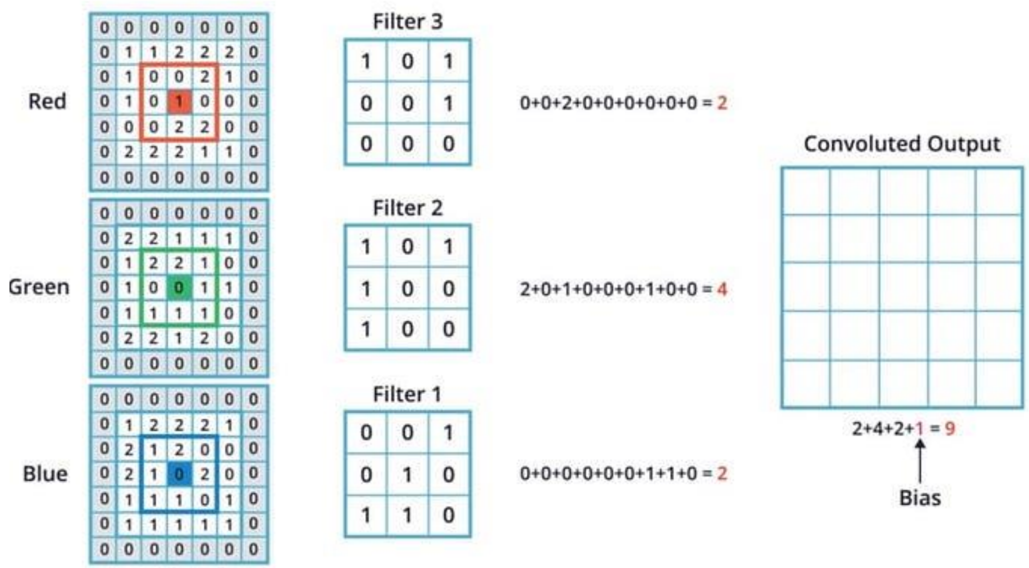
1	1	2	2	2
1	0	0	2	1
1	0	1	0	0
0	0	2	2	0
2	2	2	1	1

2	2	1	1	1
1	2	2	1	0
1	0	0	1	1
1	1	1	1	0
2	2	1	2	0

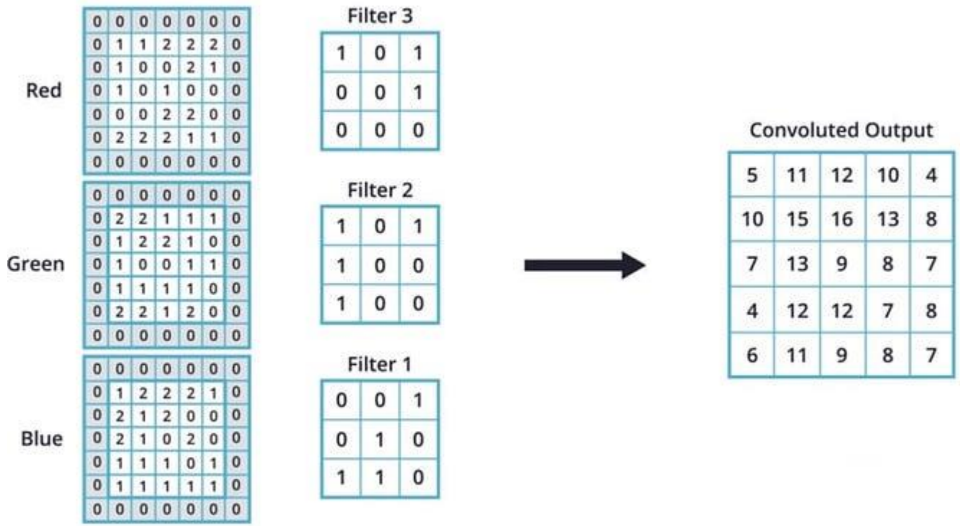
1	2	2	2	1
2	1	2	0	0
2	1	0	2	0
1	1	1	0	1
1	1	1	1	1

Just as we did with the grayscale images, we’ll add zero padding to each of these arrays in order to avoid losing information when performing the convolution.

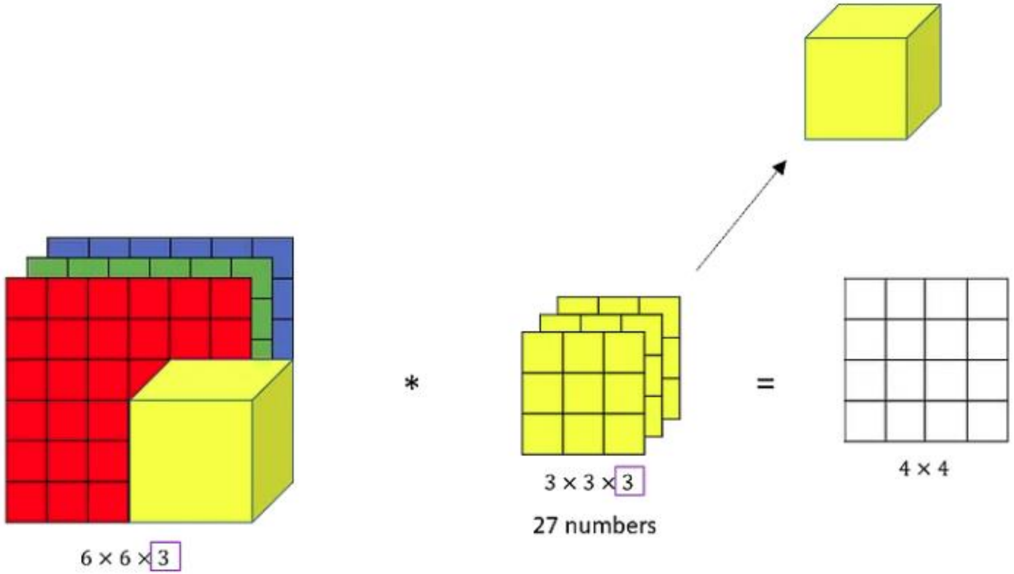




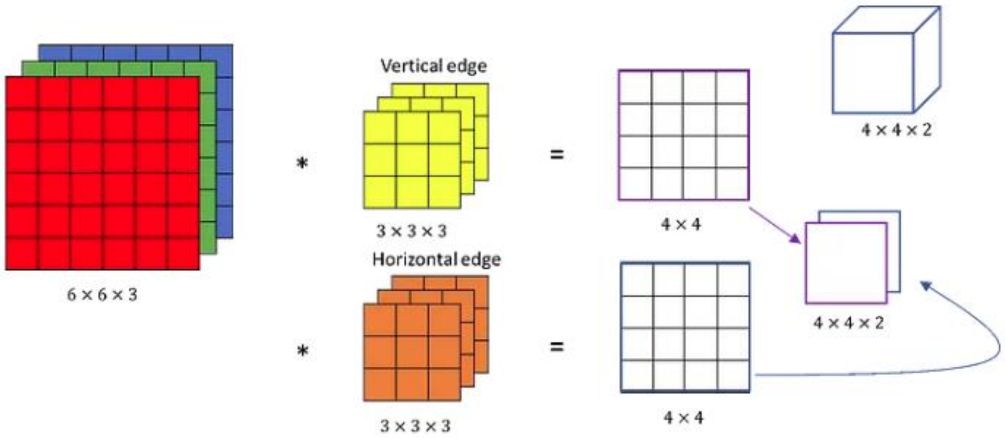
Red
Green
Blue
Filter 3
Filter 2
Filter 1
0+0+2+0+0+0+0+0+0 = 2
2+0+1+0+0+0+1+0+0 = 4
0+0+0+0+0+0+1+1+0 = 2
Convoluted Output
2+4+2+1 = 9
Bias



Red
Green
Blue
Filter 3
Filter 2
Filter 1
Convoluted Output

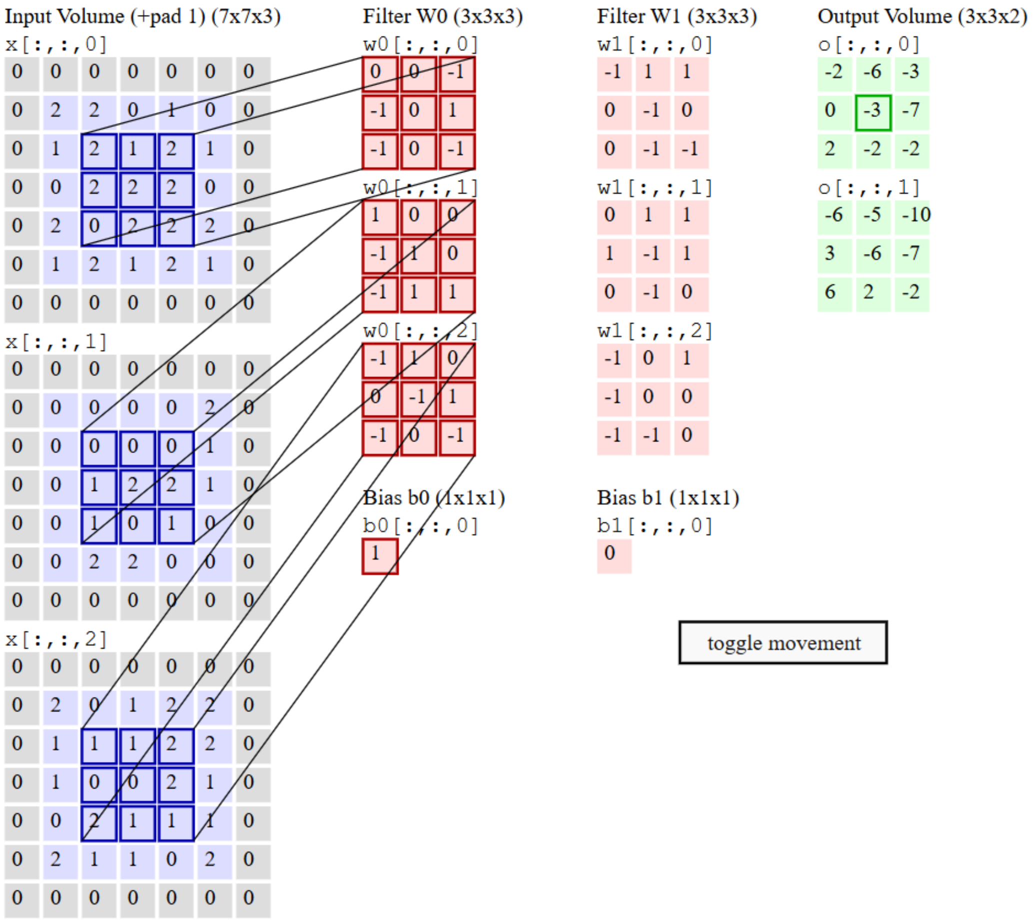


Convolution operation happening on an RGB image with a 3D kernel



Convolution operation happening on an RGB image with 2 3D kernel so the output size will be 4 × 4 × 2

Convolution demo (on image with 3 channels)

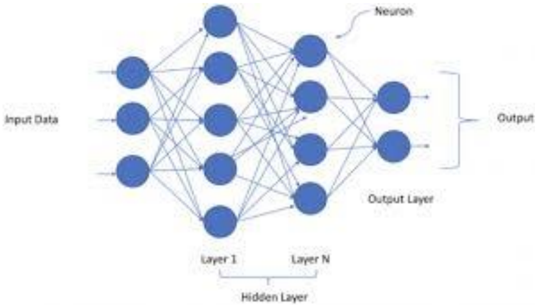


[CS231n Convolutional Neural Networks for Visual Recognition]

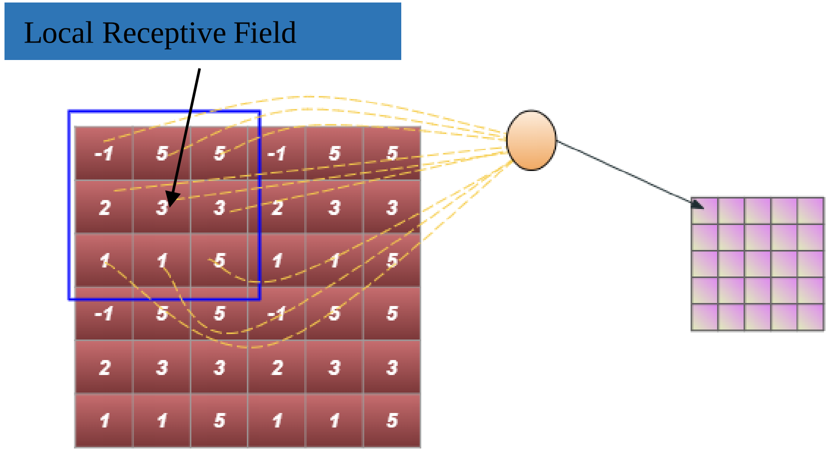
Sparse connectivity in CNN

- Let's say you have a 10x10 image. In a dense neural network:
  - *We will connect every 100 neurons to the 100 in the next layer.(Dense)*
  - *Over that, each all will have a distinct weight (No sharing)*

  So, total *parm = (100x100)+100 =10100*



In a Convolution Neural Network, the approach is as shown in this image:



Sparsity - The pixel at the next layer is not connected to all the 100 from the first layer i.e. only a local group is connected to one pixel of next layer. It is not trying to get information from the full image every time. We are harnessing the properties of an image that a group of near-by pixels has better info than grouping distant pixels

So, total parm(definitely size, number, and stride of the kernel will control it) With a 3x3 kernel,

(3 * 3) + 1 per kernel = 10 per kernel

Even with 200 kernels, it will be only 2K as compared to 10K

Some computations

- **Formula for convolution layer (size of convolution map):**
- Let the size of an image be W*W, the size of the filter be F*F, the padding be P and let stride be S. Then the formula to find the size of the convolution image is:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Considering the no. of channels,
Convolution Output dimension = [(W- F +2 *P) / S] +1 x D
Where, D is the depth (no. of channels)

- **Formula for pooling layer (o/p of pooling layer):**

$$W_{out} = \frac{W - F}{S} + 1$$

Example

CONV 1
Input Size ($W_1 \times H_1 \times D_1$) = 28 x 28 x 1
• Requires four hyperparameter: ○ Number of kernels, k = 16 ○ Spatial extend of each one, F = 5 ○ Stride Size, S = 1 ○ Amount of zero padding, P = 2
• Outputting volume of $W_2 \times H_2 \times D_2$ ○ $W_2 = (28 - 5 + 2 (2)) / 1 + 1 = 28$ ○ $H_2 = (28 - 5 + 2 (2)) / 1 + 1 = 28$ ○ $D_2 = k$
Output of Conv 1 ($W_2 \times H_2 \times D_2$) = 28 x 28 x 16

POOL 1
Input Size ($W_2 \times H_2 \times D_2$) = 28 x 28 x 16
• Requires two hyperparameter: ○ Spatial extend of each one, F = 2 ○ Stride Size, S = 2
• Outputting volume of $W_3 \times H_3 \times D_2$ ○ $W_3 = (28 - 2) / 2 + 1 = 14$ ○ $H_3 = (28 - 2) / 2 + 1 = 14$
Output of Pool 1 ($W_3 \times H_3 \times D_2$) = 14 x 14 x 16

CONV 2
Input Size (W_3 x H_3 x D_2) = 14 x 14 x 16
• Requires four hyperparameter: o Number of kernels, k = 32 o Spatial extend of each one, F = 5 o Stride Size, S = 1 o Amount of zero padding, P = 2
• Outputting volume of W_4 x H_4 x D_3 o $W_4 = (14 – 5 + 2 (2)) / 1 + 1 = 14$ o $H_4 = (14 – 5 + 2 (2)) / 1 + 1 = 14$ o $D_3 = k$
Output of Conv 2 (W_4 x H_4 x D_3) = 14 x 14 x 32

POOL 2
Input Size (W_4 x H_4 x D_3) = 14 x 14 x 32
• Requires two hyperparameter: o Spatial extend of each one, F = 2 o Stride Size, S = 2
• Outputting volume of W_5 x H_5 x D_3 o $W_5 = (14 – 2) / 2 + 1 = 7$ o $H_5 = (14 – 2) / 2 + 1 = 7$
Output of Pool 2 (W_5 x H_5 x D_3) = 7 x 7 x 32

FC Layer
Input Size (W_5 x H_5 x D_3) = 7 x 7 x 32
Output Size (Number of Classes) = 10

Computation of number of parameters

1. **Number of parameters in the convolution layers:**

- **Parameters = (Fw*Fh * number of channels + bias_term) * K**

  Fw: width of filter

  Fh: height of filter

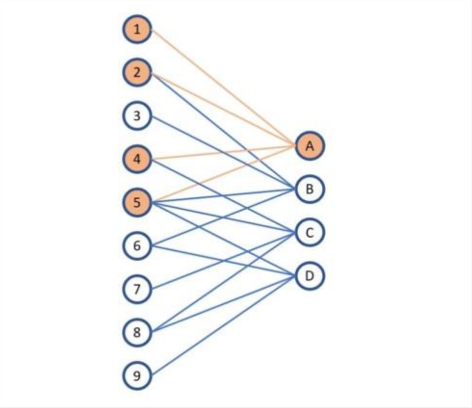
  K: no. of filters

  Example: kernel Size is (3x3) with 3 channels (RGB in the input), one bias term, and 5 filters:

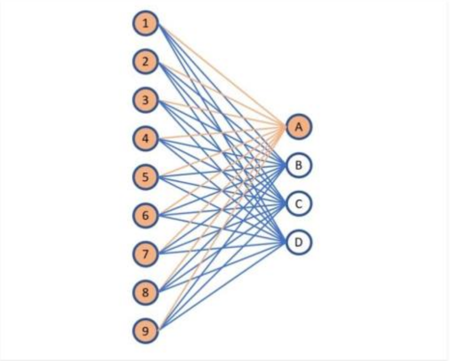
  Parameters = (3 * 3 * 3 + 1) * 5 = 140

- **2. Number of parameters in pooling layer:**

  Since pooling operation is a fixed function it introduces no additional parameters.



Neuron connection in Convolution layer



Neuron connection in fully connected(dense) layer

- **3. Number of parameters in Fully connected layer:**
  - For a fully connected layer, the number of parameters is given by the number of input units times the number of output units, plus one bias term for each output unit.
  - **#Parameters=(input units × output units)+output units**
  - Ex: fully connected layer with 128 input units and 64 output units
  - $Parameters=(128\times64)+64=8192+64=8256$

Example:

Compute the output shape and # parameters of each layer

Layers		Kernel size	Number of Kernel	stride
Sl. No	Image (32x32x1)			
1	Convolution	5x5	6	1
2	Avg. pooling	2x2	6	2
3	Convolution	5x5	16	1
4	Avg. pooling	2x2	16	2
5	Convolution	5x5	120	1
6	FC (# Neurons=84)			
7	FC (# Neurons=10)			

Example:

Layers		Kernel size	Number of Kernel	stride	Output feature map	# of parameters
Sl. No	Image (32x32x1)					
1	Convolution	5x5	6	1	28x28x6	(5*5*1+1)*6
2	Avg. pooling	2x2	6	2	14x14x6	0
3	Convolution	5x5	16	1	10x10x16	(5*5*6+1)*16
4	Avg. pooling	2x2	16	2	5x5x16	0
5	Convolution	5x5	120	1	1x1x120	(5*5*16+1)*120
6	FC (# Neurons=84)					(120*84)+84
7	FC (# Neurons=10)					(84*10)+10

Ex: 2

Layers		Kernel size	Number of Kernel	stride	Padding	Output feature map	# of parameters
Sl. No	Image (32x32x3)						
1	Convolution	11x11	96	4	Same (p=5)	8x8x96	(11x11x3+1)*96
2	Max. pooling	2x2	?	2		4x4x96	0
3	Convolution	5x5	256	1	Same (p=2)	4x4x256	(5x5x96+1)*256
4	Max. pooling	2x2	?	2		2x2x256	0
5	Convolution	3x3	384	1	Same (p=1)	2x2x384	(3x3x256+1)*384
	FC (# Neurons=84)						2x2x384x84+84
	FC (# Neurons=10)						84x10+10

For an input of size W×W and (kernel) of size FxF, the amount of padding P that needs to be added to each side of the input (height and width) can be computed using the following formula: P=(F−1)/2