Introduction to CNN
Convolutional Neural Network is a Deep Learning algorithm specially designed for working with Images and videos. It takes images as inputs, extracts and learns the features of the image, and classifies them based on the learned features.

This algorithm **is inspired by the working of a part of the human brain which is the Visual Cortex.** The visual Cortex is a part of the human brain which is responsible for processing visual information from the outside world. It has various layers and each layer has its own functioning i.e each layer extracts some information from the image or any visual and at last all the information received from each layer is combined and the image/visual is interpreted or classified.

Similarly, CNN has various filters, and each filter extracts some information from the image such as edges, different kinds of shapes (vertical, horizontal, round), and then all of these are combined to identify the image.

Why can't we use Artificial Neural Networks for the same purpose? This is because there are some disadvantages with ANN:

•It is too much computation for an ANN model to train large-size images and different types of image channels.

•The next disadvantage is that it is unable to capture all the information from an image whereas a CNN model can capture the spatial dependencies of the image.

•Another reason is that ANN is sensitive to the location of the object in the image i.e if the location or place of the same object changes, it will not be able to classify properly.

Components of CNN

The CNN model works in two steps: **feature extraction and Classification**
Feature Extraction is a phase where various filters and layers are applied to the images to extract the information and features out of it and once it's done it is passed on to the next phase i.e **Classification** where they are classified based on the target variable of the problem.

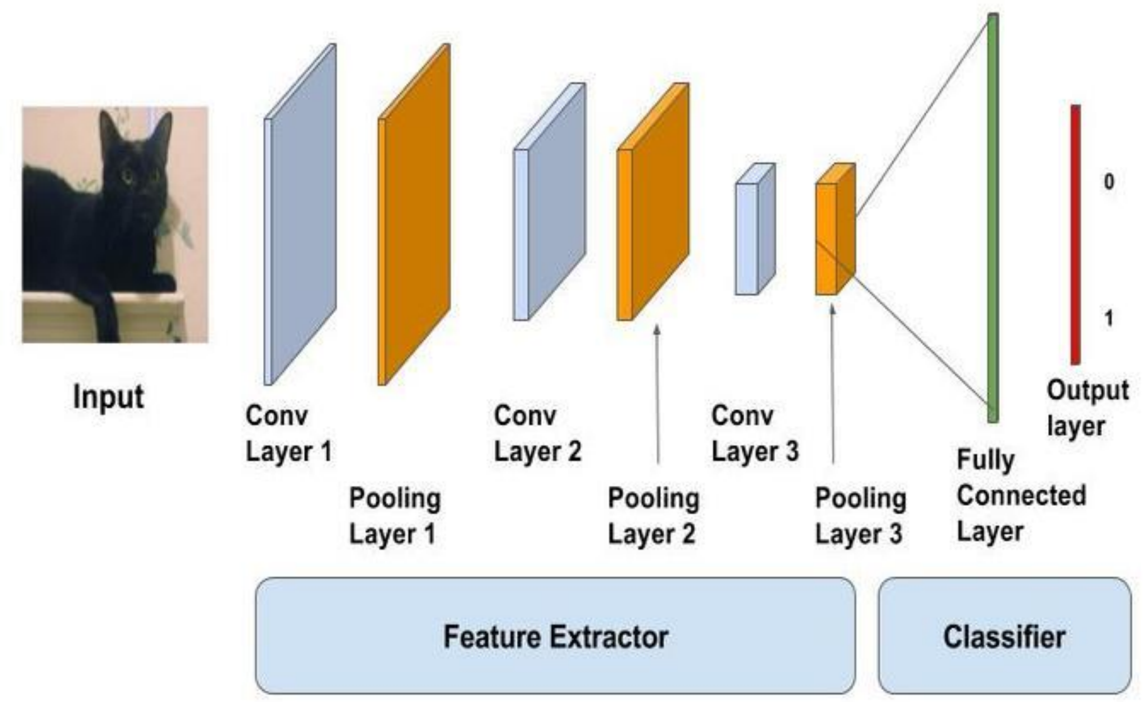
A typical CNN model looks like this:

•Input layer

•Convolution layer + Activation function

•Pooling layer

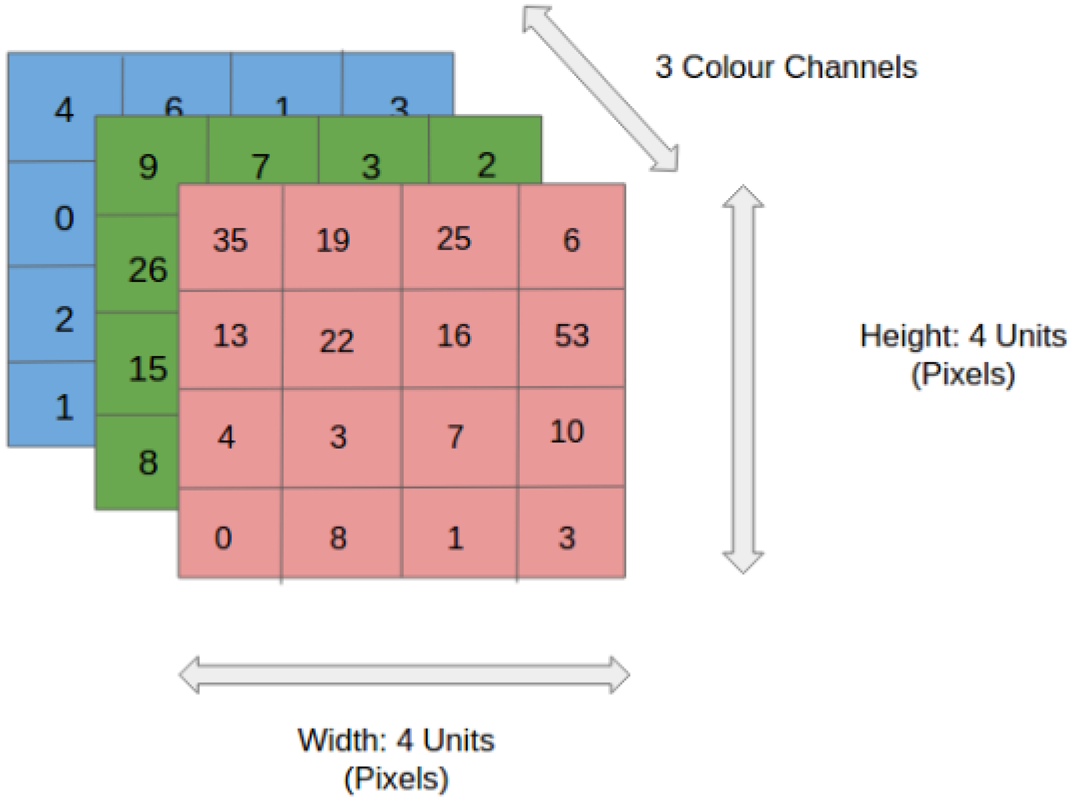
•Fully Connected Layer



Input layer

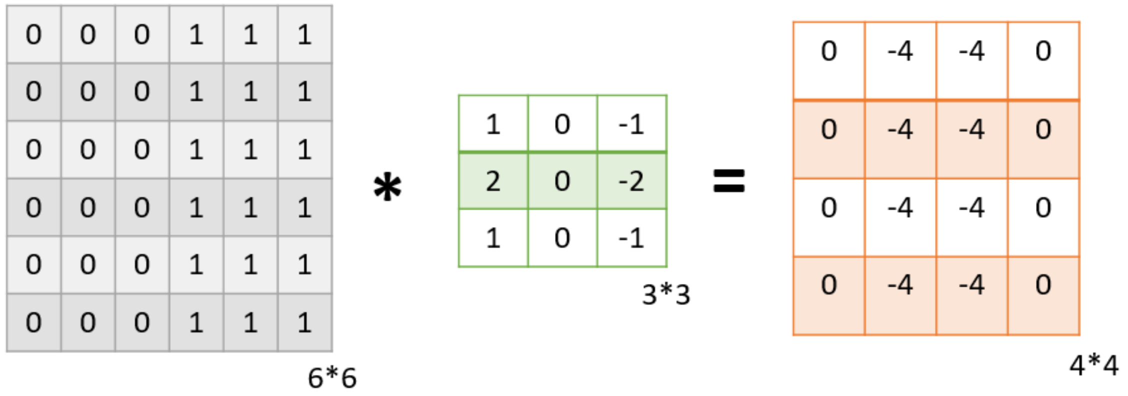
As the name says, it's our input image and can be Grayscale or RGB. Every image is made up of pixels that range from 0 to 255. We need to normalize them i.e convert the range between 0 to 1 before passing it to the model.

Below is the example of an input image of size 4*4 and has 3 channels i.e RGB and pixel values.



Convolution Layer

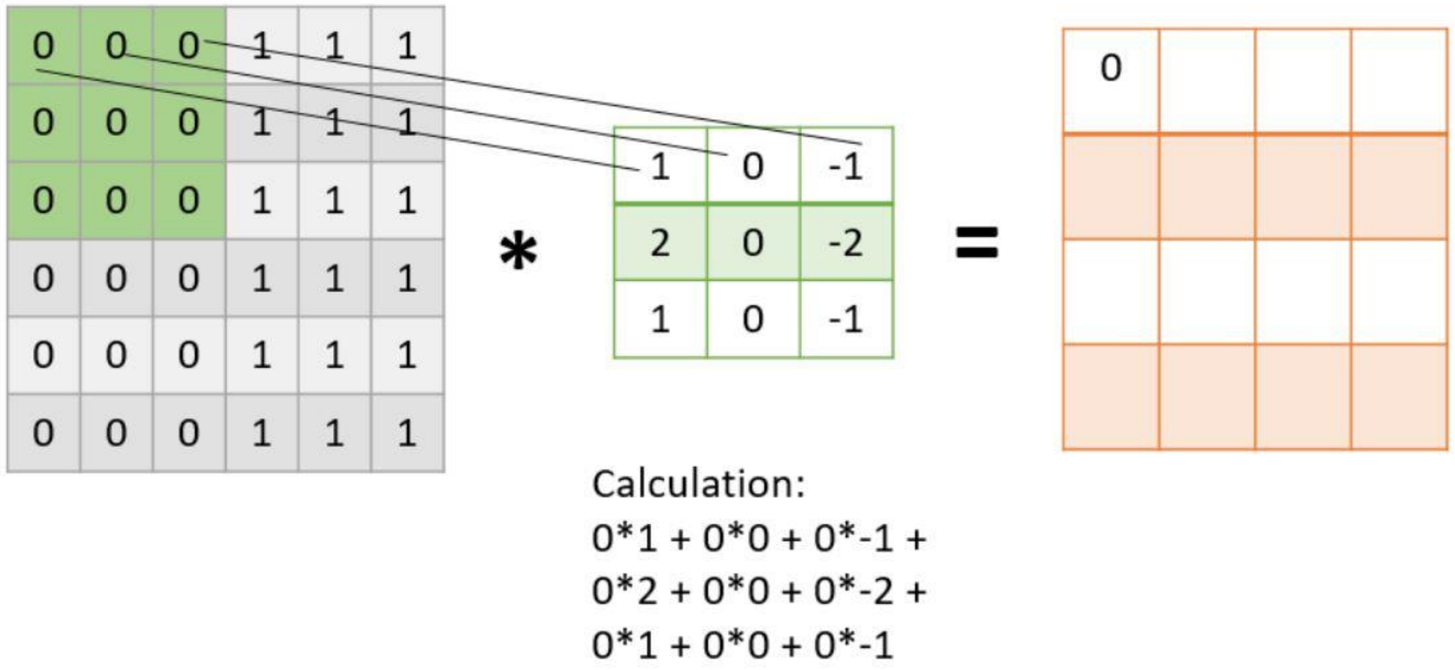
The convolution layer is the layer where the **filter is applied to our input image** to extract or detect its features. A filter is applied to the image multiple times and creates a feature map which helps in classifying the input image. Let's understand this with the help of an example. For simplicity, we will take a 2D input image with normalized pixels.



$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}}_{6*6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}}_{3*3} = \underbrace{\begin{bmatrix} 0 & -4 & -4 & 0 \\ 0 & -4 & -4 & 0 \\ 0 & -4 & -4 & 0 \\ 0 & -4 & -4 & 0 \end{bmatrix}}_{4*4}$$

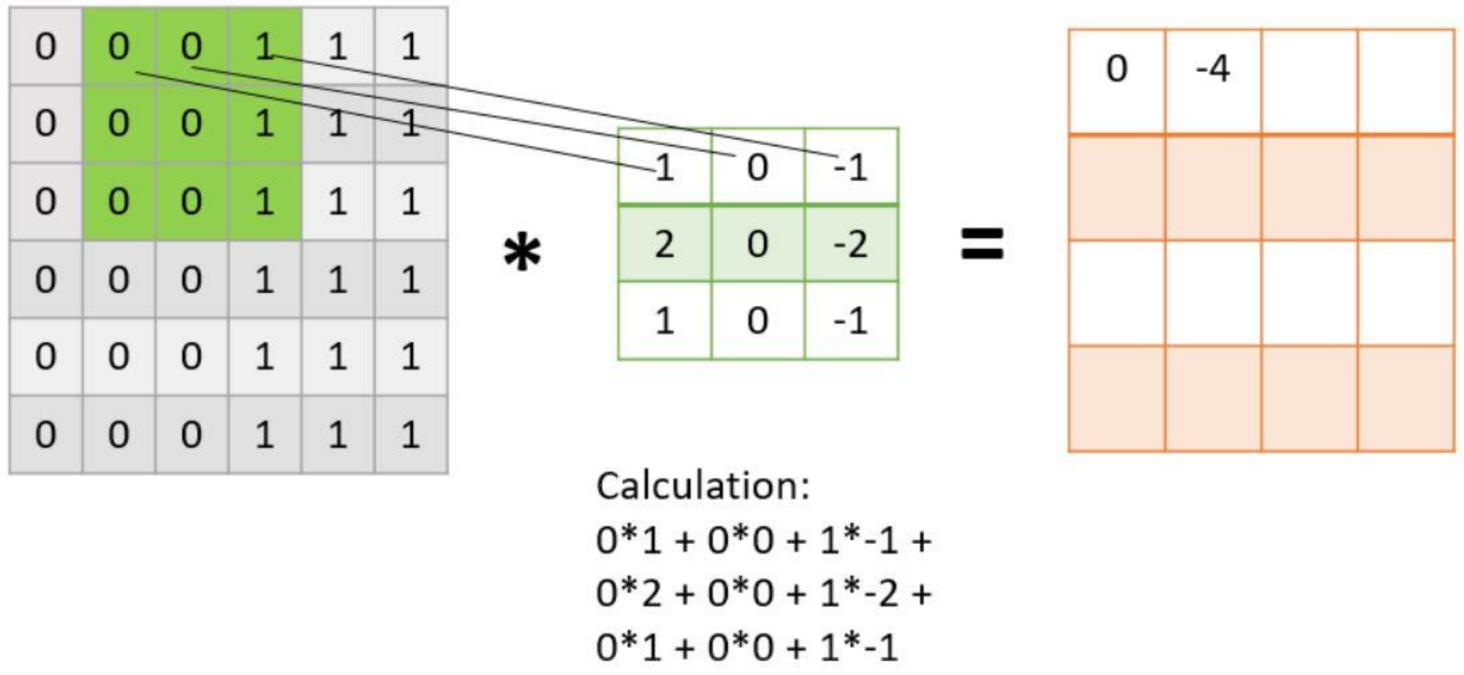
In the above figure, we have an input image of size 6*6 and applied a filter of 3*3 on it to detect some features. In this example, we have applied only one filter but in practice, many such filters are applied to extract information from the image. **The result of applying the filter to the image is that we get a Feature Map of 4*4** which has some information about the input image. Many such feature maps are generated in practical applications.

Let’s get into some maths behind getting the feature map in the above image.

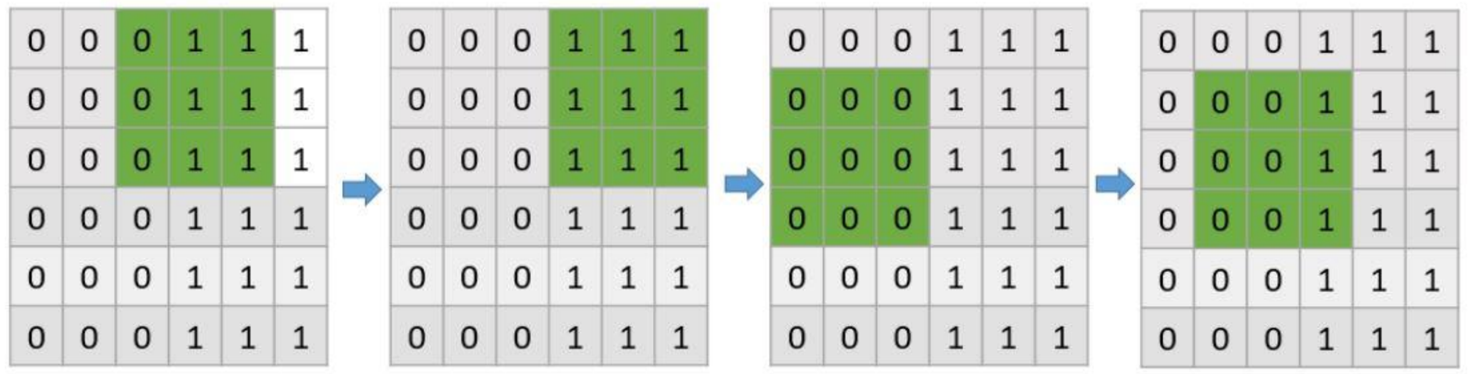


As presented in the above figure, **in the first step** the filter is applied to the green highlighted part of the image, and the pixel values of the image are multiplied with the values of the filter (as shown in the figure using lines) and then summed up to get the final value.

In the next step, the filter is shifted by one column as shown in the below figure. This jump to the next column or row is known **as stride** and in this example, we are taking a stride of 1 which means we are shifting by one column.



Similarly, the filter passes over the entire image and we get our final **Feature Map**. Once we get the feature map, an activation function is applied to it for introducing nonlinearity.
A point to note here is that the Feature map we get is smaller than the size of our image. As we increase the value of stride the size of the feature map decreases.



This is how a filter passes through the entire image with the stride of 1

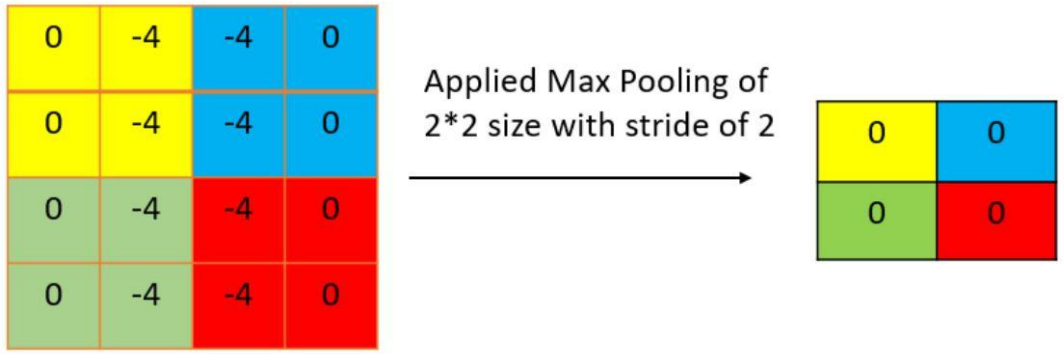
Pooling Layer

The pooling layer is applied after the Convolutional layer and is used to reduce the dimensions of the feature map which helps in preserving the important information or features of the input image and reduces the computation time.

Using pooling, a lower resolution version of input is created that still contains the large or important elements of the input image.

The most common types of Pooling are Max Pooling and Average Pooling. The below figure shows how Max Pooling works. Using the Feature map which we got from the above example to apply Pooling. Here we are using a Pooling layer of size 2*2 with a stride of 2.

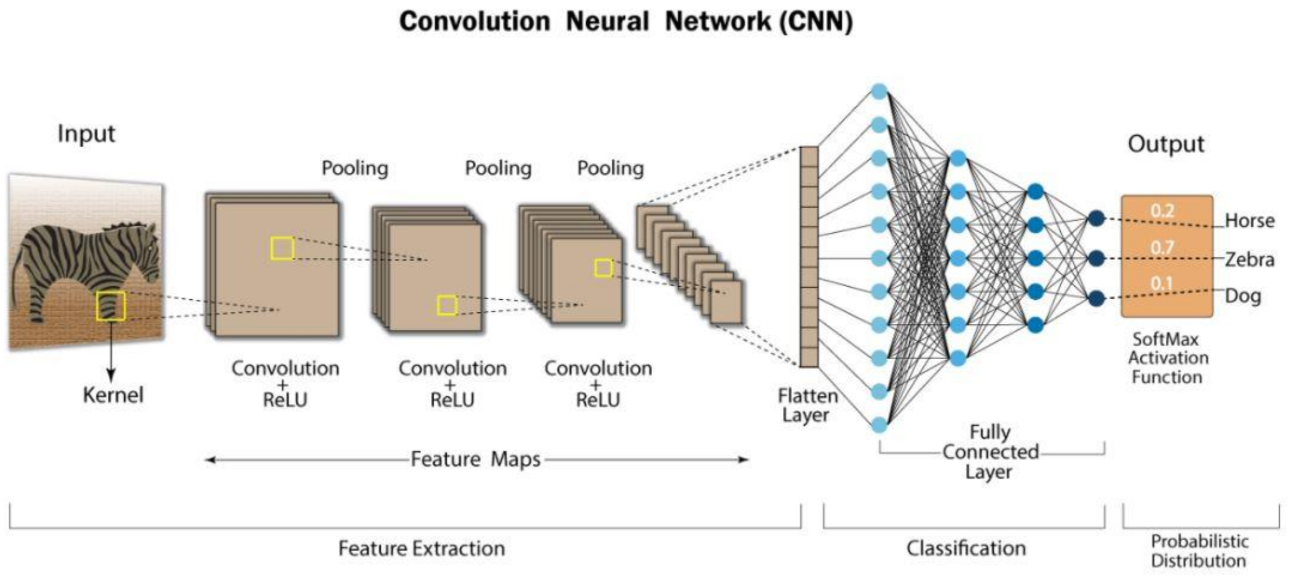
The maximum value from each highlighted area is taken and a **new version of the input image is obtained which is of size 2*2 so after applying Pooling the dimension of the feature map has reduced.**



Fully Connected Layer

Till now we have performed the Feature Extraction steps, now comes the Classification part. The Fully connected layer (as we have in ANN) is used for classifying the input image 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.

The complete process of a CNN model can be seen in the below image.



Foundations of Convolutional Neural Networks

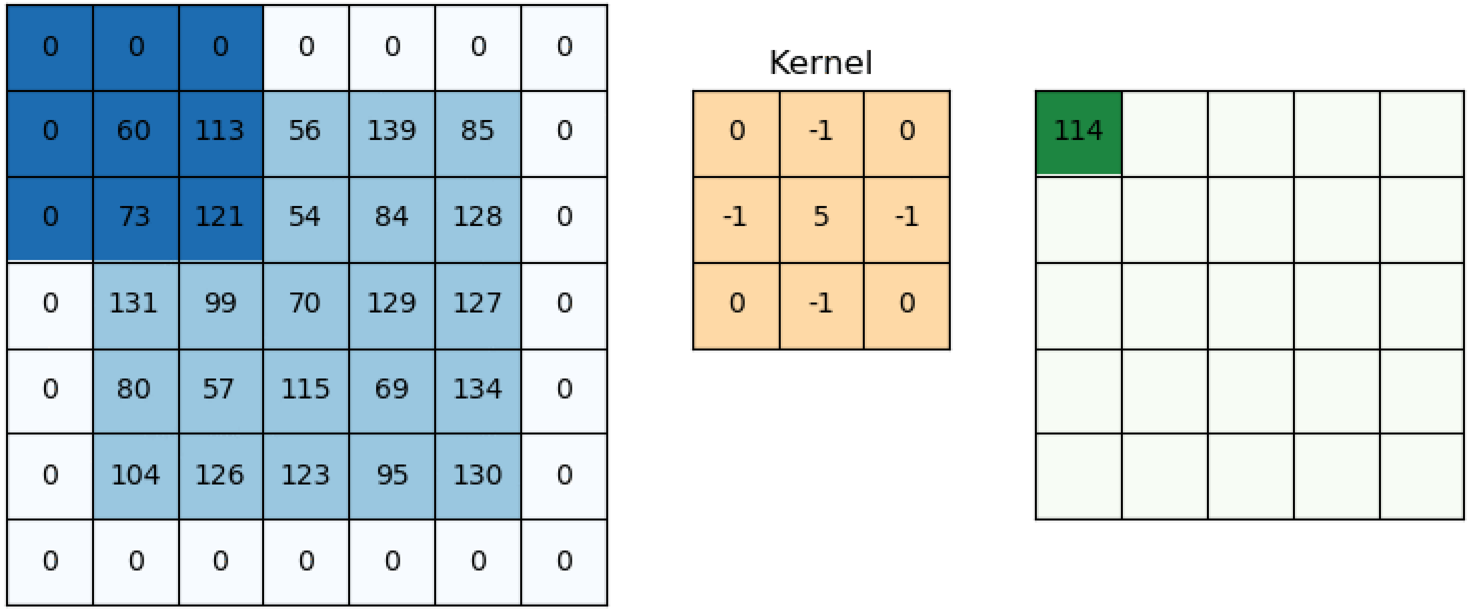
- To understand the convolution operation
- To understand the pooling operation
- Remembering the vocabulary used in convolutional neural networks (padding, stride, filter, etc.)
- Building a convolutional neural network for multi-class classification in images

Computer Vision

Some of the computer vision problems which we will be solving in this article are:

1. Image classification
2. Object detection
3. Neural style transfer

One major problem with computer vision problems is that the input data can get really big. Suppose an image is of the size 68 X 68 X 3. The input feature dimension then becomes 12,288. This will be even bigger if we have larger images (say, of size 720 X 720 X 3). Now, if we pass such a big input to a neural network, the number of parameters will swell up to a HUGE number (depending on the number of hidden layers and hidden units). This will result in more computational and memory requirements – not something most of us can deal with.



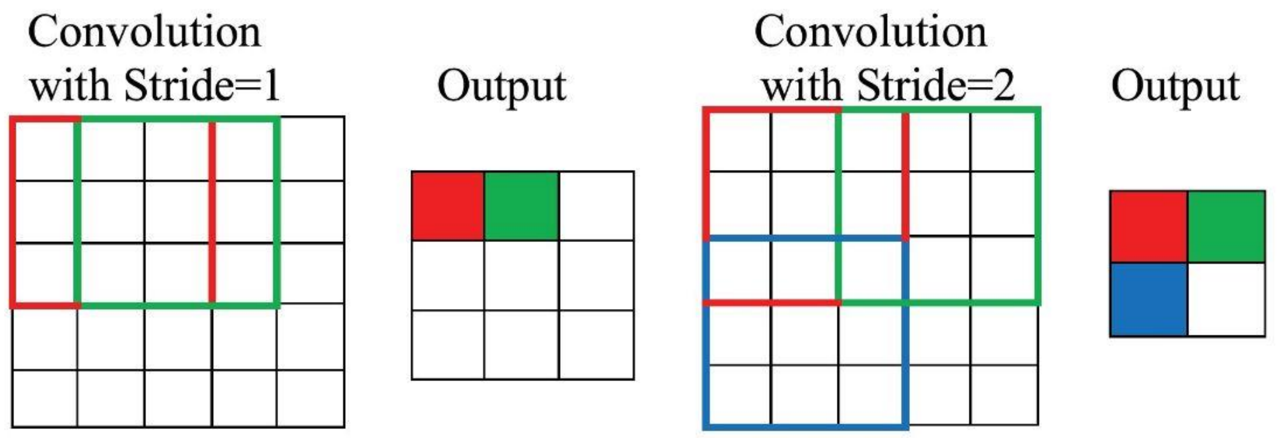
0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

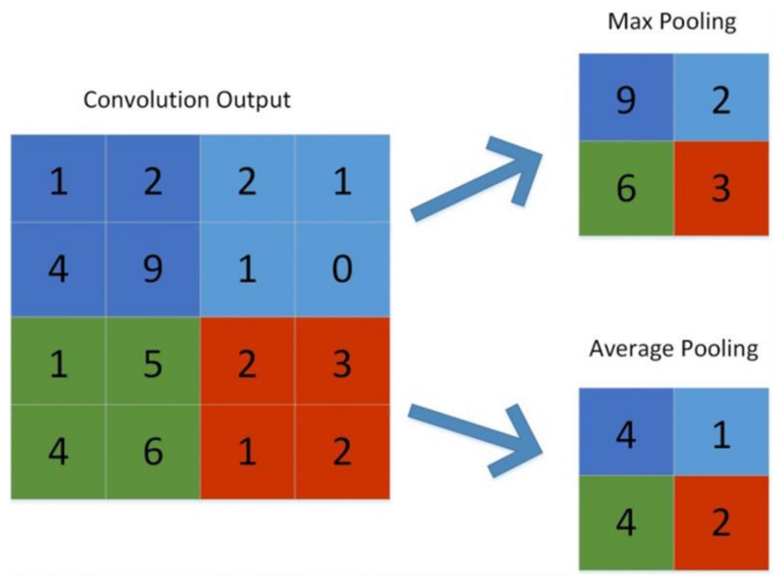
Padding works by increasing the processing region of a convolutional neural network. The kernel is a neural network filter that moves through a picture, scanning each pixel and turning the data into a smaller or bigger format. Padding is added to the image frame to aid the kernel in processing the image by providing more room for the kernel to cover the image. Adding padding to a CNN-processed image provides for more accurate image analysis.



Stride determines how the filter convolves over the input matrix, i.e. how many pixels shift. When stride is set to 1, the filter moves across one pixel at a time, and when the stride is set to 2, the filter moves across two pixels at a time. The smaller the stride value, the smaller the output, and vice versa.

Pooling

Its purpose is to gradually shrink the representation's spatial size to reduce the number of parameters and computations in the network. The pooling layer treats each feature map separately.

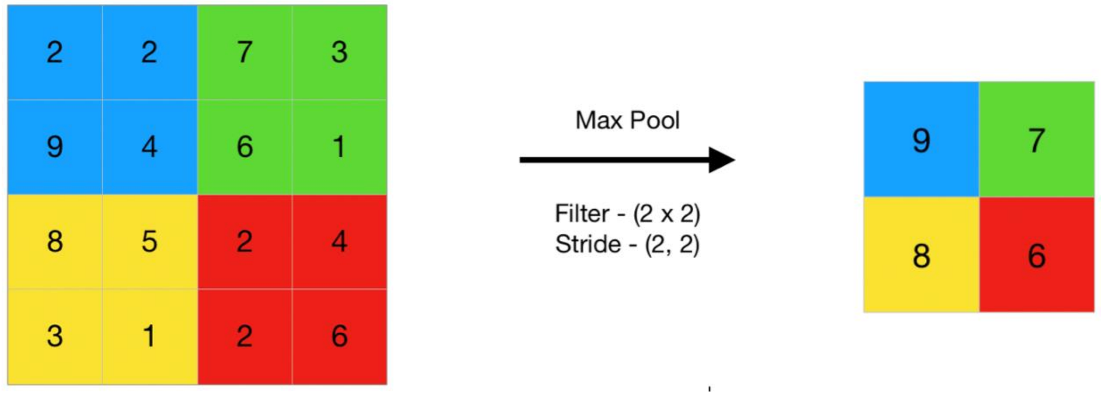


The following are some methods for pooling:
•***Max-pooling***: It chooses the most significant element from the feature map. The feature map's significant features are stored in the resulting max-pooled layer. It is the most popular method since it produces the best outcomes.
•***Average pooling***: It entails calculating the average for each region of the feature map.

Pooling gradually reduces the spatial dimension of the representation to reduce the number of parameters and computations in the network, as well as to prevent overfitting. If there is no pooling, the output has the same resolution as the input.

Max Pooling

1.Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



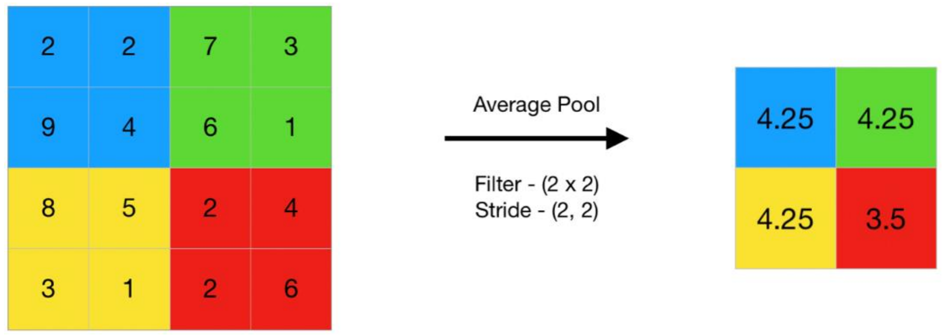
1.This can be achieved using MaxPooling2D layer in keras as follows:

Code #1 : Performing Max Pooling using keras

```
import numpy as np
from keras.models import Sequential
from keras.layers import MaxPooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single max pooling layer
model = Sequential(
   [MaxPooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

Average Pooling

1. Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.



1. Code #2 : Performing Average Pooling using keras

```
import numpy as np
from keras.models import Sequential
from keras.layers import AveragePooling2D

# define input image
image = np.array([[2, 2, 7, 3],
[9, 4, 6, 1],
[8, 5, 2, 4],
[3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single average pooling
layer
model = Sequential(
                [AveragePooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

Advantages of Pooling Layer:

Dimensionality reduction: The main advantage of pooling layers is that they help in reducing the spatial dimensions of the feature maps. This reduces the computational cost and also helps in avoiding overfitting by reducing the number of parameters in the model.

Translation invariance: Pooling layers are also useful in achieving translation invariance in the feature maps. This means that the position of an object in the image does not affect the classification result, as the same features are detected regardless of the position of the object.

Feature selection: Pooling layers can also help in selecting the most important features from the input, as max pooling selects the most salient features and average pooling preserves more information.

Disadvantages of Pooling Layer:

Information loss: One of the main disadvantages of pooling layers is that they discard some information from the input feature maps, which can be important for the final classification or regression task.

Over-smoothing: Pooling layers can also cause over-smoothing of the feature maps, which can result in the loss of some fine-grained details that are important for the final classification or regression task.

Hyperparameter tuning: Pooling layers also introduce hyperparameters such as the size of the pooling regions and the stride, which need to be tuned in order to achieve optimal performance. This can be time-consuming and requires some expertise in model building.

What Is Padding

padding is a technique used to preserve the spatial dimensions of the input image after convolution operations on a feature map. Padding involves adding extra pixels around the border of the input feature map before convolution.

This can be done in two ways:

•**Valid Padding**: In the valid padding, no padding is added to the input feature map, and the output feature map is smaller than the input feature map. This is useful when we want to reduce the spatial dimensions of the feature maps.

•**Same Padding**: In the same padding, padding is added to the input feature map such that the size of the output feature map is the same as the input feature map. This is useful when we want to preserve the spatial dimensions of the feature maps.

The number of pixels to be added for padding can be calculated based on the size of the kernel and the desired output of the feature map size. The most common padding value is zero-padding, which involves adding zeros to the borders of the input feature map.

Padding can help in reducing the loss of information at the borders of the input feature map and can improve the performance of the model. However, it also increases the computational cost of the convolution operation. Overall, padding is an important technique in CNNs that helps in preserving the spatial dimensions of the feature maps and can improve the performance of the model.

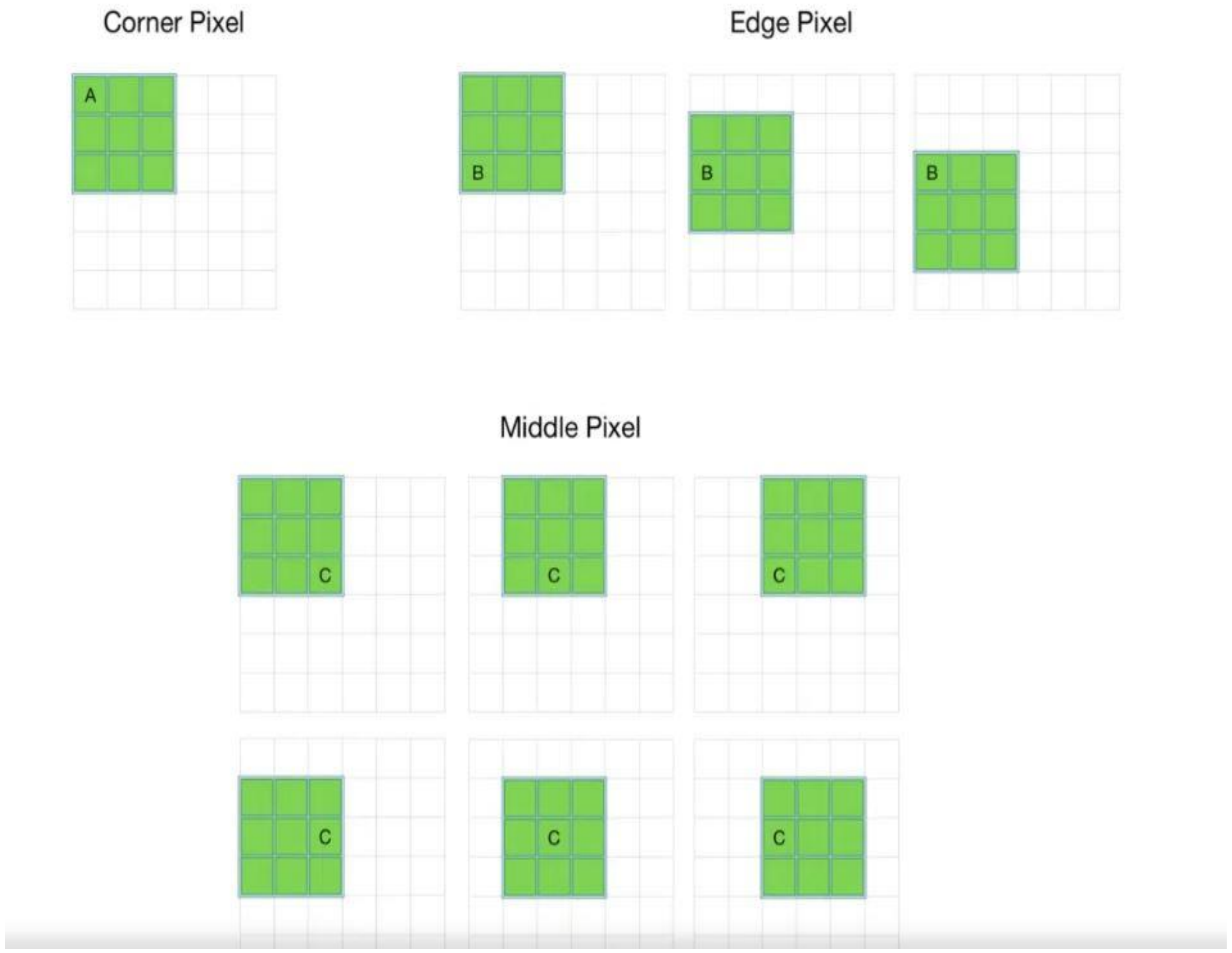
Problem With Convolution Layers Without Padding

- For a grayscale (n x n) image and (f x f) filter/kernel, the dimensions of the image resulting from a convolution operation is **(n – f + 1) x (n – f + 1)**.
-

For example, for an (8 x 8) image and (3 x 3) filter, the output resulting after the convolution operation would be of size (6 x 6). Thus, the image shrinks every time a convolution operation is performed. This places an upper limit to the number of times such an operation could be performed before the image reduces to nothing thereby precluding us from building deeper networks.

Also, the pixels on the corners and the edges are used much less than those in the middle.

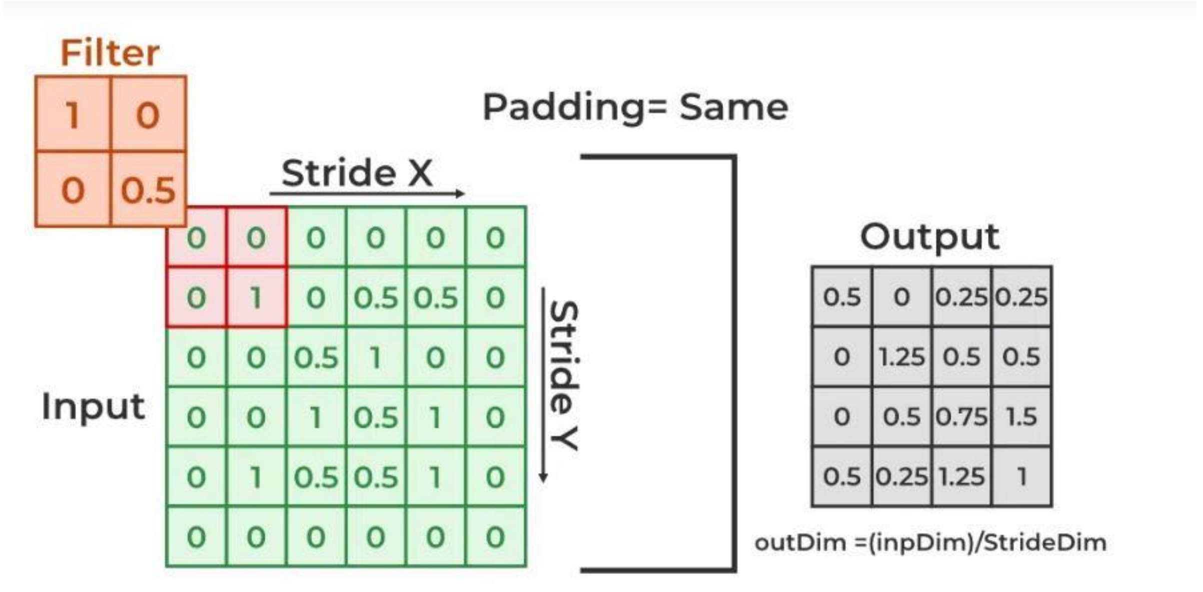
For example,



•Clearly, pixel A is touched in just one convolution operation and pixel B is touched in 3 convolution operations, while pixel C is touched in 9 convolution operations. In general, pixels in the middle are used more often than pixels on corners and edges. Consequently, the information on the borders of images is not preserved as well as the information in the middle.

Effect Of Padding On Input Images

Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems mentioned above through the following changes to the input image



padding in convolutional network

Padding prevents the shrinking of the input image.

p = number of layers of zeros added to the border of the image,

then (n x n) image —> (n + 2p) x (n + 2p) image after padding.

(n + 2p) x (n + 2p) * (f x f) ——> outputs (n + 2p – f + 1) x (n + 2p – f + 1) images

For example, by adding one layer of padding to an (8 x 8) image and using a (3 x 3) filter we would get an (8 x 8) output after performing a convolution operation.
This increases the contribution of the pixels at the border of the original image by bringing them into the middle of the padded image. Thus, information on the borders is preserved as well as the information in the middle of the image.