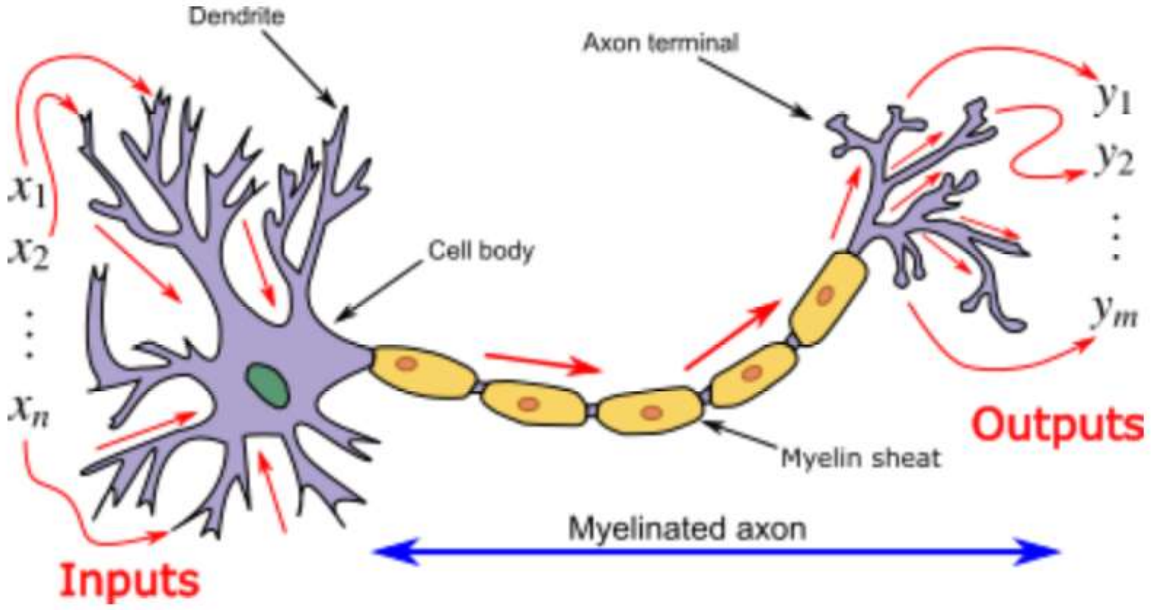
DEEP LEARNING

Deep learning is an aspect of artificial intelligence (AI) that is to simulate the activity of the human brain specifically, pattern recognition by passing input through various layers of the neural network.

Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs.



Biological neuron	Artificial neuron
dendrites	inputs
synapses	weight or inter connection
axon	output
cell body (Soma)	summation and threshold

```
[ ] import pandas as pd
    import numpy as np
    import tensorflow as tf
```

```
[ ] dataset = pd.read_csv('Churn_Modelling.csv')
```

```
[ ] X = dataset.iloc[:, 3:-1].values
    Y = dataset.iloc[:, -1].values
```

```
[ ] print(X)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

```
[ ] from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    X[:, 2] = le.fit_transform(X[:, 2])
```

```
    print(X)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

```
[ ] from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import OneHotEncoder
    ct = ColumnTransformer(transformers=[('encoder',OneHotEncoder(), [1])], remainder = 'passthrough')
    X = np.array (ct.fit_transform(X))
```

```
[ ] print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

```
[ ] from sklearn.model_selection import train_test_split
    X_train,X_test,Y_train,Y_test =train_test_split(X,Y,test_size=0.2,random_state=0)
```

```
[ ] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train =sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

```
    X_train
```

```
array([[-1.01460667, -0.5698444 ,  1.74309049, ...,  0.64259497,
        -1.03227043,  1.10643166],
       [-1.01460667,  1.75486502, -0.57369368, ...,  0.64259497,
         0.9687384 , -0.74866447],
       [ 0.98560362, -0.5698444 , -0.57369368, ...,  0.64259497,
        -1.03227043,  1.48533467],
       ...,
       [ 0.98560362, -0.5698444 , -0.57369368, ...,  0.64259497,
        -1.03227043,  1.41231994],
       [-1.01460667, -0.5698444 ,  1.74309049, ...,  0.64259497,
         0.9687384 ,  0.84432121],
       [-1.01460667,  1.75486502, -0.57369368, ...,  0.64259497,
        -1.03227043,  0.32472465]])
```

```
[ ] ann = tf.keras.models.Sequential()
```

```
[ ] ann.add(tf.keras.layers.Dense(units=6, activation= 'relu'))
```

```
[ ] ann.add(tf.keras.layers.Dense(units=6, activation= 'relu'))
```

```
[ ] ann.add(tf.keras.layers.Dense(units=1, activation= 'sigmoid'))
```

```
[ ] ann.compile(optimizer ='adam',loss = 'binary_crossentropy',metrics=['accuracy'] )
```

```
ann.fit(X_train, Y_train, batch_size =32, epochs = 100)
```

```
Epoch 1/100
250/250 [==============================] - 2s 2ms/step - loss: 0.7134 - accuracy: 0.5455
Epoch 2/100
250/250 [==============================] - 0s 2ms/step - loss: 0.5020 - accuracy: 0.7960
Epoch 3/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4504 - accuracy: 0.7964
Epoch 4/100
250/250 [==============================] - 0s 2ms/step - loss: 0.4312 - accuracy: 0.8033
Epoch 5/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4194 - accuracy: 0.8138
```

```
[ ] print (ann.predict(sc.transform([[1,0,0,600,1,40,3,60000,2,1,1,50000]]))> 0.5)
```

```
1/1 [==============================] - 0s 102ms/step
[[False]]
```

```
Y_pred = ann.predict(X_test)
Y_pred = (Y_pred >0.5)
print(np.concatenate((Y_pred.reshape(len(Y_pred),1),Y_test.reshape(len(Y_test),1)),1))
```

```
63/63 [==============================] - 0s 1ms/step
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

```
from sklearn.metrics import confusion_matrix , accuracy_score
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

```
[[1525   70]
 [ 198  207]]
```

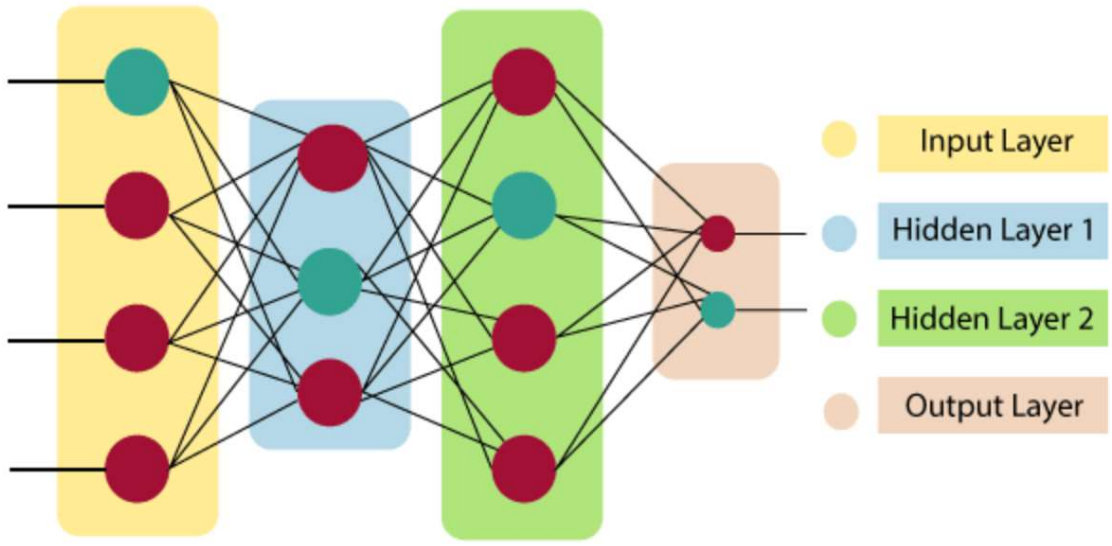
```
[ ] ac = accuracy_score(Y_test, Y_pred)
print(ac)
```

```
0.866
```

Artificial Neural Network:

*An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.*

ANN Architecture:



Input Layer:
As the name suggests, it accepts inputs in several different formats provided by the programmer.

Hidden Layer:
The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

Output Layer:
The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
[ ]
```

```
[ ] (X_train, Y_train),(X_test, Y_test ) = keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
[ ] len(X_train)
```

```
60000
```

```
[ ] len(X_test)
```

```
10000
```

```
[ ] X_train[0].shape
```

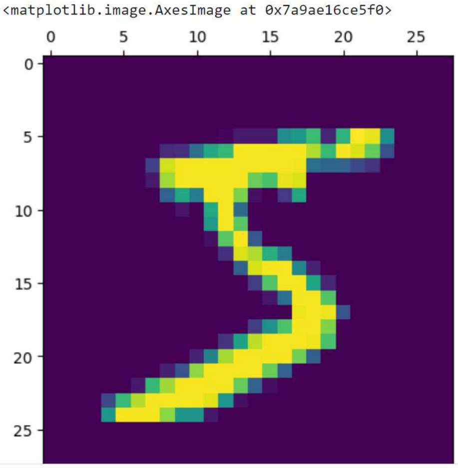
```
(28, 28)
```

```
X_train[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
```

```
plt.matshow(X_train[0])
```

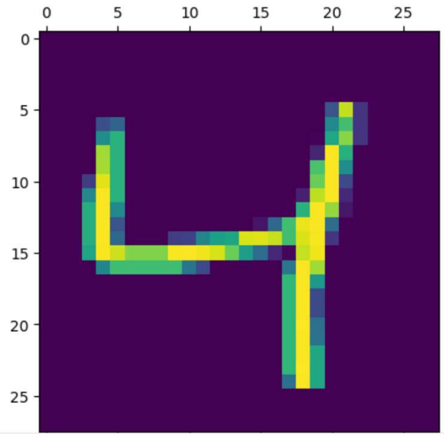
```
<matplotlib.image.AxesImage at 0x7a9ae16ce5f0>
```



```
[ ] X_test=X_test / 255
```

```
[ ] plt.matshow(X_train[2])
```

```
<matplotlib.image.AxesImage at 0x7a9ae147fc70>
```




```
Y_train[2]
```

```
4
```

```
[ ] X_train[0]
```

```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
[ ] X_train_flattened = X_train.reshape(len(X_train),28*28)
    X_test_flattened = X_test.reshape(len(X_test),28*28)
```

```
[ ] X_train_flattened.shape
```

```
(60000, 784)
```

```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])
```

```
[ ] model.compile(optimizer ='adam',loss = 'sparse_categorical_crossentropy',metrics=['accuracy'] )
```

```
[ ] model.fit(X_train_flattened, Y_train, epochs = 5)
```

```
Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.4688 - accuracy: 0.8781
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3036 - accuracy: 0.9155
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2833 - accuracy: 0.9206
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2727 - accuracy: 0.9238
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2663 - accuracy: 0.9259
<keras.callbacks.History at 0x7a9ae3ea5780>
```

```
[ ] model.fit(X_train_flattened, Y_train, epochs = 10)
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2618 - accuracy: 0.9271
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2585 - accuracy: 0.9285
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2553 - accuracy: 0.9291
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2529 - accuracy: 0.9296
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2511 - accuracy: 0.9309
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2494 - accuracy: 0.9314
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2476 - accuracy: 0.9318
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2466 - accuracy: 0.9319
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2450 - accuracy: 0.9319
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2436 - accuracy: 0.9326
<keras.callbacks.History at 0x7a9ae3c86a10>
```

```
[ ] model.evaluate(X_test_flattened, Y_test)
```

```
313/313 [==============================] - 0s 2ms/step - loss: 0.2697 - accuracy: 0.9265
[0.26971399784088135, 0.9265000224113464]
```

```
model.fit(X_train_flattened, Y_train, epochs = 20)
```

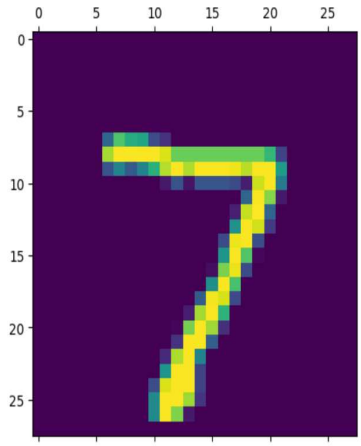
```
Epoch 1/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2427 - accuracy: 0.9328
Epoch 2/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2422 - accuracy: 0.9334
Epoch 3/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2412 - accuracy: 0.9330
Epoch 4/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2402 - accuracy: 0.9338
Epoch 5/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2393 - accuracy: 0.9337
Epoch 6/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2393 - accuracy: 0.9336
```

```
y_predicted = model.predict(X_test_flattened)
y_predicted[0]
```

```
313/313 [==============================] - 0s 1ms/step
array([1.1244274e-03, 3.6816223e-09, 4.3390929e-03, 9.7977084e-01,
       1.3053014e-03, 1.3632046e-01, 3.0628149e-09, 9.9991661e-01,
       6.7287862e-02, 7.0347941e-01], dtype=float32)
```

```
[ ] plt.matshow(X_test[0])
```

```
<matplotlib.image.AxesImage at 0x7a9ae3dbe200>
```



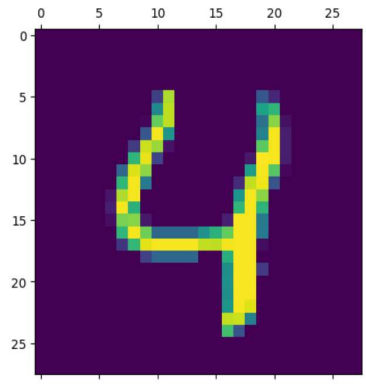
```
np.argmax(y_predicted[0])
```

```
7
```

+ Code + Text

```
plt.matshow(X_test[4])
```

```
<matplotlib.image.AxesImage at 0x7a9ae3c25450>
```



```
np.argmax(y_predicted[4])
```

```
4
```

```
y_predicted_labels =[np.argmax(i) for i in y_predicted]
```

```
y_predicted_labels[:5]
```

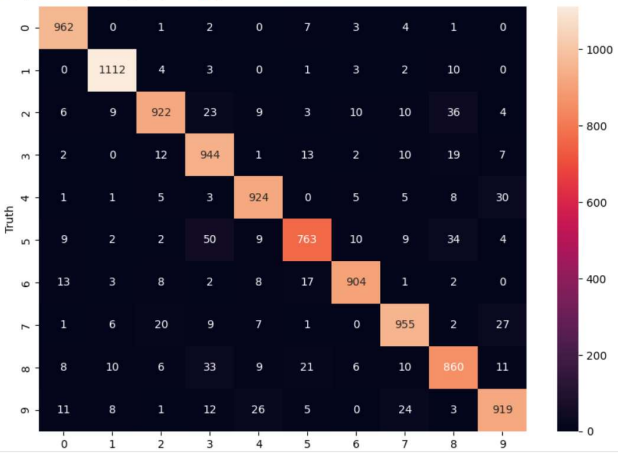
```
[7, 2, 1, 0, 4]
```

```
from sklearn.metrics import confusion_matrix , accuracy_score
cm = tf.math.confusion_matrix(labels=Y_test,predictions=y_predicted_labels)
cm
```

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 962,    0,    1,    2,    0,    7,    3,    4,    1,    0],
       [   0, 1112,    4,    3,    0,    1,    3,    2,   10,    0],
       [   6,    9,  922,   23,    9,    3,   10,   10,   36,    4],
       [   2,    0,   12,  944,    1,   13,    2,   10,   19,    7],
       [   1,    1,    5,    3,  924,    0,    5,    5,    8,   30],
       [   9,    2,    2,   50,    9,  763,   10,    9,   34,    4],
       [  13,    3,    8,    2,    8,   17,  904,    1,    2,    0],
       [   1,    6,   20,    9,    7,    1,    0,  955,    2,   27],
       [   8,   10,    6,   33,    9,   21,    6,   10,  860,   11],
       [  11,    8,    1,   12,   26,    5,    0,   24,    3,  919]],
      dtype=int32)>
```

```
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

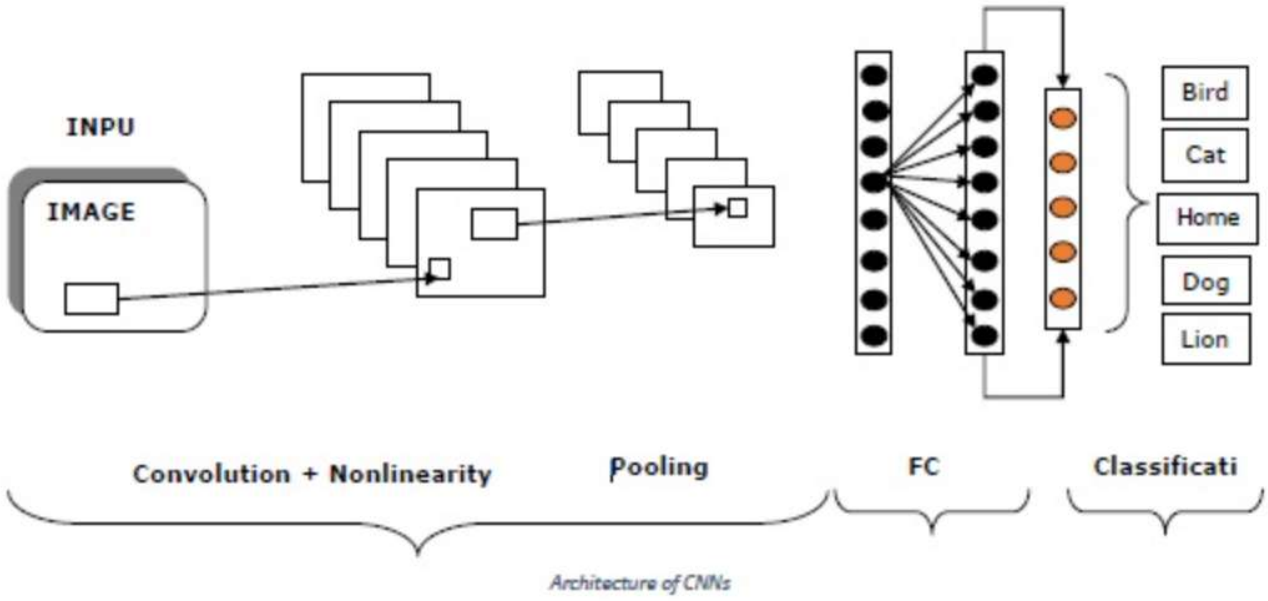
```
Text(95.72222222222221, 0.5, 'Truth')
```



Convolutional Neural Networks (CNN)

*A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.*

CNN Architecture :



Architecture of CNNs

*The architecture of CNN is basically a list of layers that transforms the 3-dimensional, i.e. width, height and depth of image volume into a 3-dimensional output volume. One important point to note here is that, every neuron in the current layer is connected to a small patch of the output from the previous layer, which is like overlaying a N*N filter on the input image.*

It uses M filters, which are basically feature extractors that extract features like edges, corner and so on. Following are the layers [INPUT-CONV-RELU-POOL-FC] that are used to construct Convolutional neural networks (CNNs)−

***INPUT−** As the name implies, this layer holds the raw pixel values. Raw pixel values mean the data of the image as it is. Example, INPUT [64×64×3] is a 3-channeled RGB image of width-64, height-64 and depth-3.*

***CONV−** This layer is one of the building blocks of CNNs as most of the computation is done in this layer. Example - if we use 6 filters on the above mentioned INPUT [64×64×3], this may result in the volume [64×64×6].*

RELU–*Also called rectified linear unit layer, that applies an activation function to the output of previous layer. In other manner, a non-linearity would be added to the network by RELU.*

POOL– *This layer, i.e. Pooling layer is one other building block of CNNs. The main task of this layer is down-sampling, which means it operates independently on every slice of the input and resizes it spatially.*

FC– *It is called Fully Connected layer or more specifically the output layer. It is used to compute output class score and the resulting output is volume of the size 1*1*L where L is the number corresponding to class score.*

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets,layers,models
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
(X_train, Y_train),(X_test, Y_test ) = datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

```
[ ] X_train.shape
```

```
(50000, 32, 32, 3)
```

```
[ ] X_test.shape
```

```
(10000, 32, 32, 3)
```

```
[ ] Y_train.shape
```

```
(50000, 1)
```

```
[ ] Y_train[:5]
```

```
array([[6],
       [9],
       [9],
       [4],
       [1]], dtype=uint8)
```

```
[ ] Y_train=Y_train.reshape(-1,)
```

```
[ ] Y_train[:5]
```

```
array([6, 9, 9, 4, 1], dtype=uint8)
```

```
[ ] Y_test = Y_test.reshape(-1,)
```

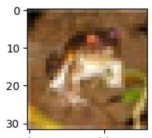
```
[ ] classes = ["aeropalane","automobile","bird","cat","deer","horse","frog","ship","truck"]
```

```
[ ] def plot_sample(X, Y, index):
      plt.figure(figsize=(15,2))
      plt.imshow(X[index])
      plt.xlabel(classes[Y[index]])
```

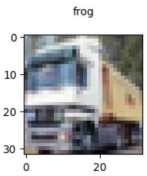
```
[ ] plot_sample(X_train, Y_train, 0)
    plot_sample(X_train, Y_train, 1)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-40-1cc89e947941> in <cell line: 2>()
      1 plot_sample(X_train, Y_train, 0)
----> 2 plot_sample(X_train, Y_train, 1)

<ipython-input-23-f0a9ec3fbe83> in plot_sample(X, Y, index)
      2   plt.figure(figsize=(15,2))
      3   plt.imshow(X[index])
----> 4   plt.xlabel(classes[Y[index]])

IndexError: list index out of range
```

SEARCH STACK OVERFLOW



[ ]



[ ]
```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

[ ]
```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
        layers.Dense(3000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

[ ]
```
ann.compile(optimizer ='SGD',loss = 'sparse_categorical_crossentropy',metrics=['accuracy'] )
```

[ ]
```
ann.fit(X_train, Y_train, epochs = 5)
```

```
Epoch 1/5
1563/1563 [==============================] - 247s 158ms/step - loss: 1.7874 - accuracy: 0.3672
Epoch 2/5
1563/1563 [==============================] - 248s 159ms/step - loss: 1.6170 - accuracy: 0.4313
Epoch 3/5
1563/1563 [==============================] - 297s 190ms/step - loss: 1.5352 - accuracy: 0.4597
Epoch 4/5
1563/1563 [==============================] - 246s 158ms/step - loss: 1.4747 - accuracy: 0.4808
Epoch 5/5
1563/1563 [==============================] - 294s 188ms/step - loss: 1.4247 - accuracy: 0.5016
<keras.callbacks.History at 0x7c1bcb57c820>
```

[ ]
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes=[np.argmax(element) for element in y_pred]
print("Classification Report : \n",classification_report(Y_test, y_pred_classes))
```

```
313/313 [==============================] - 15s 47ms/step
Classification Report :
              precision    recall  f1-score   support

           0       0.39      0.69      0.50      1000
           1       0.52      0.65      0.58      1000
           2       0.34      0.43      0.38      1000
           3       0.37      0.34      0.36      1000
           4       0.43      0.42      0.42      1000
           5       0.52      0.24      0.33      1000
           6       0.51      0.57      0.54      1000
           7       0.68      0.42      0.52      1000
           8       0.59      0.58      0.59      1000
           9       0.61      0.39      0.48      1000

    accuracy                           0.47     10000
   macro avg       0.50      0.47      0.47     10000
weighted avg       0.50      0.47      0.47     10000
```

[ ]
```
cnn = models.Sequential([
        layers.Conv2D(filters=32,kernel_size=(3, 3),activation='relu',input_shape=(32,32,3)),
        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(filters=64,kernel_size=(3, 3),activation='relu'),
        layers.MaxPooling2D((2, 2)),


        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
```

[ ]
```
cnn.compile(optimizer ='adam',loss = 'sparse_categorical_crossentropy',metrics=['accuracy'] )
```

```
cnn.fit(X_train, Y_train, epochs=5)
```

```
Epoch 1/5
1563/1563 [==============================] - 88s 56ms/step - loss: 1.4754 - accuracy: 0.4724
Epoch 2/5
1563/1563 [==============================] - 71s 46ms/step - loss: 1.1282 - accuracy: 0.6067
Epoch 3/5
1563/1563 [==============================] - 73s 47ms/step - loss: 0.9926 - accuracy: 0.6555
Epoch 4/5
1563/1563 [==============================] - 70s 45ms/step - loss: 0.9113 - accuracy: 0.6872
Epoch 5/5
1563/1563 [==============================] - 74s 48ms/step - loss: 0.8467 - accuracy: 0.7092
<keras.callbacks.History at 0x7c1b55b60ee0>
```

[ ]
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
313/313 [==============================] - 9s 29ms/step
array([[1.71831821e-03, 6.59377765e-05, 1.18115963e-02, 7.44306862e-01,
        5.45485457e-03, 1.09645784e-01, 1.40003711e-02, 2.81721266e-04,
        1.12682894e-01, 3.17499398e-05],
       [5.94656914e-03, 6.61861658e-01, 3.10682881e-05, 4.58769318e-06,
        4.99457565e-06, 1.06754828e-06, 2.33595188e-07, 2.00855339e-07,
        3.30306381e-01, 1.84317876e-03],
       [1.39298752e-01, 2.14389756e-01, 3.34494840e-03, 2.00385065e-03,
        2.16697995e-03, 6.11400523e-04, 4.32278175e-04, 6.27348374e-04,
        6.15417719e-01, 2.17070114e-02],
       [8.00836205e-01, 6.21537073e-03, 4.34178188e-02, 4.89246380e-03,
        2.57877614e-02, 2.39708097e-04, 2.16571428e-03, 7.79164024e-04,
        1.15329601e-01, 3.36188852e-04],
       [1.94945715e-05, 3.60517413e-04, 6.92958245e-03, 5.51242046e-02,
        4.67642486e-01, 3.67716178e-02, 4.33071464e-01, 1.42619083e-05,
        5.52918245e-05, 1.10924602e-05]], dtype=float32)
```

[ ]
```
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
```

```
[3, 1, 8, 0, 4]
```

```
[ ] Y_test[:5]
```

```
array([3, 8, 8, 0, 6], dtype=uint8)
```