**Deccan Education Society's**

**Kirti M. Doongursee College of Arts, Science and Commerce**

**[NAAC Accredited: "A Grade"]**

**M.Sc. [Computer Science]**

# Practical Journal

**PAPER:** PSCSP402

## Roll Number [_____]

## Department of Computer Science and Information Technology

**Department of Computer Science and Information Technology**

**Deccan Education Society's**

**Kirti M. Doongursee College of Arts, Science and Commerce**

**[NAAC Accredited: "A Grade"]**

# C E R T I F I C A T E

This is to certify that Mr./Mrs. _____

of M.Sc. (Computer Science) with Roll No._____ has completed **6** Practicals of Paper **PSCSP402** under my supervision in this College during the year 2022-2023.

**Lecturer-In-Charge**

Date:

**Examined by:**

Date:

**H.O.D.**

**Dept of CS & IT**

Date:

**Remarks:**

_____

# Index

## Practical No. 1

## Aim: Feed Forward Neural Network for binary classification on Pima Diabetes Dataset

```
import pandas as pd
```

```
df = pd.read_csv('diabetes.csv', names= list)
```

```
df.head(1)
```

| | Preg | BP1 | BP2 | GL | IL | BMI | Skin Tone | Age | Class |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |

```
list = ['Preg', 'BP1', 'BP2', 'GL', 'IL', 'BMI', 'Skin Tone', 'Age', 'Class']
```

```
x = df.iloc[:, :-1]
```

```
x
```

| | Preg | BP1 | BP2 | GL | IL | BMI | Skin Tone | Age |
|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **763** | 10 | 101 | 76 | 48 | 180 | 32.9 | 0.171 | 63 |
| **764** | 2 | 122 | 70 | 27 | 0 | 36.8 | 0.340 | 27 |
| **765** | 5 | 121 | 72 | 23 | 112 | 26.2 | 0.245 | 30 |
| **766** | 1 | 126 | 60 | 0 | 0 | 30.1 | 0.349 | 47 |
| **767** | 1 | 93 | 70 | 31 | 0 | 30.4 | 0.315 | 23 |

768 rows × 8 columns

```
y = df.iloc[:, -1]
```

```
y
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Class, Length: 768, dtype: int64
```

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_text, y_train, y_test = train_test_split(x, y, test_size= 0.25, random_state=1)
```

```
from keras.models import Sequential
```

```
from keras.layers import Dense
```

```
model = Sequential()
```

```
model.add(Dense(units = 10, activation = 'relu', input_dim = 8))
model.add(Dense(units = 8, activation = 'relu'))
model.add(Dense(units = 1, activation = 'sigmoid'))
```

```
model.get_weights()
```

```
[array([[-5.76351464e-01,  1.13994479e-01,  2.96345294e-01,
         -3.51937056e-01,  7.86898732e-02,  1.26476645e-01,
         -2.33206034e-01, -3.80760223e-01,  4.78078842e-01,
         -1.22534454e-01],
        [-2.39596605e-01,  2.28325725e-02,  4.67275977e-01,
         -2.05934048e-04,  6.91922307e-02, -2.58608013e-01,
          1.06054544e-02, -5.92546463e-02,  4.95849371e-01,
         -8.92637968e-02],
        [ 4.69256163e-01,  5.63312173e-01,  5.70974231e-01,
          1.33033991e-02,  2.21366286e-02,  2.64824450e-01,
          2.80220330e-01,  3.43943000e-01, -5.63554049e-01,
          3.48502517e-01],
        [-1.06273293e-02,  3.18214357e-01, -2.81240046e-01,
          1.09517634e-01, -2.69869924e-01, -2.60802031e-01,
         -4.96660173e-01, -2.62459487e-01, -4.62389499e-01,
         -7.79162645e-02],
        [ 6.35100007e-02, -5.11386931e-01, -8.12408030e-02,
         -3.19058299e-02,  2.63592899e-01, -1.10914528e-01,
          2.91414082e-01,  3.13997984e-01,  4.37785983e-01,
          4.47722077e-01],
        [-5.76488316e-01, -4.04310942e-01,  5.13805628e-01,
         -2.15868741e-01, -2.94898480e-01,  9.63056087e-02,
         -8.21345747e-02,  5.14592528e-01,  2.03541398e-01,
          4.46760178e-01],
        [ 5.75723410e-01,  1.60794854e-01, -2.73232877e-01,
```

```
            -2.73833275e-02, -2.02077895e-01,  5.02375007e-01,
            -1.85588539e-01,  4.20549929e-01,  1.83541000e-01,
             1.08909607e-02],
           [ 5.32470107e-01,  2.31829822e-01,  4.23220038e-01,
             4.57944989e-01,  5.54332972e-01, -4.60347533e-02,
             2.44126141e-01,  9.47476625e-02, -1.78356916e-01,
            -3.91281039e-01]], dtype=float32),
     array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
     array([[-0.23398581, -0.18254027,  0.05644751,  0.41369432, -0.35973996,
            -0.2605614 , -0.5044145 , -0.29831058],
           [-0.3942616 ,  0.03347492, -0.46178037, -0.5050927 ,  0.3634426 ,
             0.5378643 , -0.519408  , -0.17563212],
           [ 0.43930888,  0.24798632, -0.35115796,  0.14867783, -0.03777707,
            -0.5113057 ,  0.42622626, -0.21367294],
           [-0.42324483, -0.05086112,  0.31224537,  0.07328981,  0.29252523,
             0.35138524, -0.42211914,  0.07560468],
           [ 0.17862505, -0.5514264 ,  0.31691962,  0.33828235,  0.4863721 ,
            -0.10003829, -0.26642728,  0.47834897],
           [-0.2791574 ,  0.34731638,  0.15640628,  0.4404739 ,  0.28299415,
            -0.37726623,  0.28064746, -0.11323324],
           [-0.05220306,  0.07340902, -0.5003282 ,  0.22457355, -0.24711251,
             0.29543573,  0.5427711 ,  0.56940997],
           [ 0.25234944, -0.43505555,  0.06046969, -0.36972487,  0.4219011 ,
             0.18869162, -0.08157903, -0.07560879],
           [-0.3999349 ,  0.542735  , -0.5460352 , -0.49435562, -0.4125024 ,
            -0.35385028,  0.16188973, -0.21992561],
           [-0.07563359, -0.01150805, -0.47440422,  0.15752631, -0.08567634,
             0.48781383,  0.3423943 , -0.37177712]], dtype=float32),
     array([0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32),
     array([[ 0.43605053],
           [-0.77755696],
           [ 0.04577094],
           [ 0.11027336]
```

```
model.compile(loss = 'binary_crossentropy', optimizer= 'adam', metrics= 'accuracy')
```

```
model.fit(x_train, y_train, epochs= 500)
```

```
Epoch 1/500
18/18 [==============================] - 1s 5ms/step - loss: 10.2167 - accuracy: 0.5625
Epoch 2/500
18/18 [==============================] - 0s 3ms/step - loss: 6.0994 - accuracy: 0.4062
Epoch 3/500
18/18 [==============================] - 0s 3ms/step - loss: 4.5599 - accuracy: 0.4236
Epoch 4/500
18/18 [==============================] - 0s 3ms/step - loss: 3.4897 - accuracy: 0.5295
Epoch 5/500
18/18 [==============================] - 0s 4ms/step - loss: 2.7531 - accuracy: 0.5399
Epoch 6/500
18/18 [==============================] - 0s 3ms/step - loss: 2.1650 - accuracy: 0.5556
Epoch 7/500
18/18 [==============================] - 0s 3ms/step - loss: 1.7178 - accuracy: 0.5660
Epoch 8/500
18/18 [==============================] - 0s 3ms/step - loss: 1.3645 - accuracy: 0.5903
Epoch 9/500
18/18 [==============================] - 0s 3ms/step - loss: 1.1832 - accuracy: 0.5920
Epoch 10/500
18/18 [==============================] - 0s 3ms/step - loss: 1.0640 - accuracy: 0.6111
Epoch 11/500
18/18 [==============================] - 0s 3ms/step - loss: 0.9942 - accuracy: 0.6319
```

```
Epoch 12/500
18/18 [==============================] - 0s 3ms/step - loss: 0.9463 - accuracy: 0.6302
Epoch 13/500
18/18 [==============================] - 0s 4ms/step - loss: 0.8638 - accuracy: 0.6476
Epoch 14/500
18/18 [==============================] - 0s 3ms/step - loss: 0.8353 - accuracy: 0.6372
Epoch 15/500
18/18 [==============================] - 0s 3ms/step - loss: 0.8077 - accuracy: 0.6649
Epoch 16/500
18/18 [==============================] - 0s 2ms/step - loss: 0.7879 - accuracy: 0.6597
Epoch 17/500
18/18 [==============================] - 0s 3ms/step - loss: 0.7633 - accuracy: 0.6736
Epoch 18/500
18/18 [==============================] - 0s 3ms/step - loss: 0.7509 - accuracy: 0.6719
Epoch 19/500
18/18 [==============================] - 0s 3ms/step - loss: 0.7442 - accuracy: 0.6493
Epoch 20/500
18/18 [==============================] - 0s 2ms/step - loss: 0.7032 - accuracy: 0.6701
Epoch 21/500
18/18 [==============================] - 0s 3ms/step - loss: 0.6904 - accuracy: 0.6701
Epoch 22/500
18/18 [==============================] - 0s 3ms/step - loss: 0.6743 - accuracy: 0.6806
Epoch 23/500
18/18 [==============================] - 0s 3ms/step - loss: 0.6658 - accuracy: 0.6858
Epoch 24/500
18/18 [==============================] - 0s 2ms/step - loss: 0.6627 - accuracy: 0.6771
Epoch 25/500
18/18 [==============================] - 0s 3ms/step - loss: 0.6467 - accuracy: 0.6788
Epoch 26/500
18/18 [==============================] - 0s 2ms/step - loss: 0.6347 - accuracy: 0.6875
Epoch 27/500
18/18 [==============================] - 0s 3ms/step - loss: 0.6330 - accuracy: 0.6719
Epoch 28/500
18/18 [==============================] - 0s 3ms/step - loss: 0.6237 - accuracy: 0.6823
Epoch 29/500
```

```
model.get_weights()
```

```
[array([[-1.0193615 , -0.7998783 ,  0.21630175, -0.36132455,  0.4083842 ,
          0.1850338 , -0.56222606, -0.41923752,  0.68521357, -0.74010795],
        [-0.17547238, -0.04869542,  0.36112285,  0.03565129,  0.12301555,
         -0.07940839, -0.01799607,  0.03640669,  0.40965888, -0.05829513],
        [ 0.3746896 ,  0.50478345,  0.47368687, -0.0574043 , -0.00338464,
          0.32431084,  0.18289994,  0.37723348, -0.5601721 ,  0.35124502],
        [ 0.01713938,  0.30530906, -0.1917603 ,  0.06848543, -0.16867633,
         -0.03941232, -0.29646465, -0.32071832, -0.6689581 , -0.0295758 ],
        [ 0.211138  , -0.56394935, -0.11688343, -0.02406065,  0.3015624 ,
         -0.02302786,  0.19865103,  0.29994327,  0.34378204,  0.33716786],
        [-0.6225652 , -0.36775473,  0.32729942, -0.17173134, -0.0762496 ,
          0.10070616, -0.17341469,  0.5849934 ,  0.07909144,  0.1863475 ],
        [ 0.88478833,  0.34240985, -0.42319292, -0.42886555, -0.01347255,
          0.02750231, -0.42247266,  0.7268161 ,  0.43515396, -0.32450455],
        [ 0.4791351 ,  0.08500058,  0.38419428,  0.51942414,  0.69290906,
          0.04838531,  0.30030215,  0.08377503, -0.13941619, -0.42959628]],
       dtype=float32),
 array([-1.0929743 , -0.23392558, -1.1026931 ,  1.5583355 , -1.2501293 ,
         1.5201178 ,  1.3633183 , -1.2011082 ,  0.5617469 ,  1.4727436 ],
       dtype=float32),
 array([[-3.34666163e-01, -1.85093582e-01,  5.64475060e-02,
         -1.22436732e-01, -2.49337539e-01, -3.58221769e-01,
```

```
          -4.12938118e-01, -3.81430477e-01],
         [-3.97891164e-01, -1.12556769e-02, -4.61780369e-01,
          -6.80680454e-01,  3.39822203e-01,  4.82675672e-01,
          -4.92306560e-01, -4.26289558e-01],
         [ 4.67828542e-01,  1.29091144e-01, -3.51157963e-01,
           1.01081692e-01, -1.89247765e-02, -5.21179616e-01,
           3.99796784e-01, -1.56250834e-01],
         [-5.00844955e-01, -2.08707392e-01,  3.12245369e-01,
           1.65220454e-01,  4.22747970e-01,  3.42319578e-01,
          -3.22548896e-01,  2.84771919e-01],
         [ 2.04572245e-01, -6.73583388e-01,  3.16919625e-01,
           3.95481825e-01,  5.01760840e-01, -1.42136753e-01,
          -2.88896143e-01,  5.63941598e-01],
         [-1.90342978e-01,  2.96552122e-01,  1.56406283e-01,
           3.19689780e-01,  1.77559242e-01, -4.25031781e-01,
           1.97990224e-01, -1.83038905e-01],
         [-1.00422194e-02,  3.92984003e-02, -5.00328183e-01,
           5.37031181e-02, -2.87201881e-01,  2.60262877e-01,
           4.99231815e-01,  2.81946152e-01],
         [ 2.86377519e-01, -5.22136271e-01,  6.04696870e-02,
          -4.13695514e-01,  4.24179822e-01,  1.43579319e-01,
          -1.20422482e-01,  5.46673546e-04],
         [-2.63551563e-01,  4.09666300e-01, -5.46035171e-01,
          -5.26037991e-01, -5.32835305e-01, -3.84325147e-01,
           3.10432948e-02, -1.89262554e-01],
         [-3.42150368e-02, -3.85727398e-02, -4.74404216e-01,
           4.19578254e-02, -1.16382286e-01,  4.44164813e-01,
           2.99792111e-01, -3.68973076e-01]], dtype=float32),
 array([-1.5108297 , -0.11005322,  0.        ,  0.5513515 ,  1.6805108 ,
        -0.03905296,  1.4935019 ,  1.0239908 ], dtype=float32),
 array([[ 0.43011555],
        [-0.6968025 ],
        [ 0.04577094],
        [-0.29001507],
        [-0.1675244 ],
```

```
predictions = model.predict(x_text)
```

```
6/6 [==============================] - 0s 3ms/step
```

```
predictions
```

```
array([[0.70388275],
       [0.53173137],
       [0.48346993],
       [0.07104975],
       [0.36412728],
       [0.37310907],
       [0.52387214],
       [0.2600502 ],
       [0.14527108],
       [0.3671134 ],
       [0.47270727],
       [0.04896887],
       [0.89712435],
       [0.8102267 ],
       [0.23963562],
       [0.7815331 ],
       [0.48004118],
       [0.25691614],
```

```
            [0.37625885],
            [0.45481673],
            [0.6058107 ],
            [0.16842838],
            [0.9327141 ],
            [0.37526774],
            [0.03564999],
            [0.53809714],
            [0.41033557],
            [0.85888654],
            [0.1802825 ],
            [0.5358011 ],
            [0.29958186],
            [0.6224153 ],
            [0.06290238],
            [0.80559576],
            [0.21385732],
            [0.81833977],
            [0.17728046],
            [0.3946962 ],
            [0.17329827],
            [0.68751866],
            [0.33606356],
            [0.25351778],
            [0.03635573],
            [0.22435172],
            [0.13350281],
            [0.07900093],
            [0.9785833 ],
            [0.93245286],
            [0.34957546],
            [0.3010678 ],
            [0.01968112],
            [0.39631346],
            [0.87881994],
            [0.15051498],
            [0.78858644],
            [0.10984007],
            [0.80604297],
            [0.07782565],
```

```
class_labels = []


for i in range(predictions.size):
  if (predictions[i] > 0.51):
    class_labels.append(1)
  else:
    class_labels.append(0)


from sklearn.metrics import accuracy_score


print(accuracy_score(class_labels, y_test))
```

```
     0.78125
```

## Practical No. 2

## Aim: Feed Forward Neural Network for binary classification on Cancer Dataset using Adam and SGD

```
from sklearn.datasets import load_breast_cancer

import pandas as pd

from keras.models import Sequential
from keras.layers import Dense

from sklearn.metrics import accuracy_score

dataset = load_breast_cancer()

dataset
```

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
```

```
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset\n-------
-------------------------------------\n\n**Data Set Characteristics:**\n\n    :Number of
Instances: 569\n\n    :Number of Attributes: 30 numeric, predictive attributes and the
class\n\n    :Attribute Information:\n        - radius (mean of distances from center to
points on the perimeter)\n        - texture (standard deviation of gray-scale values)\n
- perimeter\n        - area\n        - smoothness (local variation in radius lengths)\n
- compactness (perimeter^2 / area - 1.0)\n        - concavity (severity of concave portions of
the contour)\n        - concave points (number of concave portions of the contour)\n        -
symmetry\n        - fractal dimension ("coastline approximation" - 1)\n\n        The mean,
standard error, and "worst" or largest (mean of the three\n        worst/largest values) of
these features were computed for each image,\n        resulting in 30 features.  For instance,
field 0 is Mean Radius, field\n        10 is Radius SE, field 20 is Worst Radius.\n\n        -
class:\n                - WDBC-Malignant\n                - WDBC-Benign\n\n    :Summary
Statistics:\n\n    ===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius (mean):
6.981  28.11\n    texture (mean):                       9.71   39.28\n    perimeter (mean):
```

```
df = pd.DataFrame(dataset.data, columns = dataset.feature_names)
```

```
df.head(1)
```

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points |
|---|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.8 | 1001.0 | 0.1184 | 0.2776 | 0.3001 | 0.1471 |

1 rows × 30 columns

```
df['class'] = dataset.target
```

```
df.head(1)
```

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | fr dime |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.8 | 1001.0 | 0.1184 | 0.2776 | 0.3001 | 0.1471 | 0.2419 | 0. |

1 rows × 31 columns

```
df['class'].unique
```

```
<bound method Series.unique of 0      0
1      0
2      0
3      0
4      0
```

```
..
564    0
565    0
566    0
567    0
568    1
Name: class, Length: 569, dtype: int64>
```

```
x = df.iloc[:, :-1]
```

```
x
```

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | di |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.30010 | 0.14710 | 0.2419 | |
| **1** | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.08690 | 0.07017 | 0.1812 | |
| **2** | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.19740 | 0.12790 | 0.2069 | |
| **3** | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.24140 | 0.10520 | 0.2597 | |
| **4** | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.19800 | 0.10430 | 0.1809 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **564** | 21.56 | 22.39 | 142.00 | 1479.0 | 0.11100 | 0.11590 | 0.24390 | 0.13890 | 0.1726 | |
| **565** | 20.13 | 28.25 | 131.20 | 1261.0 | 0.09780 | 0.10340 | 0.14400 | 0.09791 | 0.1752 | |
| **566** | 16.60 | 28.08 | 108.30 | 858.1 | 0.08455 | 0.10230 | 0.09251 | 0.05302 | 0.1590 | |
| **567** | 20.60 | 29.33 | 140.10 | 1265.0 | 0.11780 | 0.27700 | 0.35140 | 0.15200 | 0.2397 | |
| **568** | 7.76 | 24.54 | 47.92 | 181.0 | 0.05263 | 0.04362 | 0.00000 | 0.00000 | 0.1587 | |

569 rows × 30 columns

```
y = df.iloc[:, -1]
```

```
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: class, Length: 569, dtype: int64
```

```
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 = 1)

model = Sequential()

model.add(Dense(15, activation = 'relu', input_dim = 30))

model.add(Dense(10, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = 'accuracy')

model.fit(x_train, y_train, epochs = 100)
```

```
Epoch 1/100
15/15 [==============================] - 1s 3ms/step - loss: 6.0931 - accuracy: 0.5604
Epoch 2/100
15/15 [==============================] - 0s 3ms/step - loss: 0.7953 - accuracy: 0.3538
Epoch 3/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6846 - accuracy: 0.3736
Epoch 4/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6717 - accuracy: 0.4747
Epoch 5/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6077 - accuracy: 0.7736
Epoch 6/100
15/15 [==============================] - 0s 2ms/step - loss: 0.5253 - accuracy: 0.8769
Epoch 7/100
15/15 [==============================] - 0s 2ms/step - loss: 0.5114 - accuracy: 0.8989
Epoch 8/100
15/15 [==============================] - 0s 2ms/step - loss: 0.5047 - accuracy: 0.8945
Epoch 9/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5066 - accuracy: 0.8967
Epoch 10/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4965 - accuracy: 0.8967
Epoch 11/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5000 - accuracy: 0.8879
Epoch 12/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4910 - accuracy: 0.8989
Epoch 13/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4851 - accuracy: 0.8967
Epoch 14/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4815 - accuracy: 0.8989
Epoch 15/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4749 - accuracy: 0.9055
Epoch 16/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4765 - accuracy: 0.8901
Epoch 17/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4682 - accuracy: 0.8989
Epoch 18/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4688 - accuracy: 0.8901
Epoch 19/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4633 - accuracy: 0.9033
Epoch 20/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4665 - accuracy: 0.8945
```

```
Epoch 21/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4575 - accuracy: 0.8989
Epoch 22/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4577 - accuracy: 0.8791
Epoch 23/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4501 - accuracy: 0.8989
Epoch 24/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4430 - accuracy: 0.9033
Epoch 25/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4428 - accuracy: 0.9033
Epoch 26/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4404 - accuracy: 0.8945
Epoch 27/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4395 - accuracy: 0.8967
Epoch 28/100
15/15 [==============================] - 0s 2ms/step - loss: 0.4361 - accuracy: 0.9077
Epoch 29/100
```

```
predictions = model.predict(x_test)
```

```
4/4 [==============================] - 0s 3ms/step
```

```
predictions
```

```
array([[7.6093352e-01],
       [6.0712910e-01],
       [7.6093352e-01],
       [7.6255962e-02],
       [7.6093352e-01],
       [6.4233929e-02],
       [1.8969402e-02],
       [7.0078957e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [1.1428808e-02],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [5.9851474e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [6.9900067e-04],
       [1.6016141e-08],
       [2.4602570e-01],
       [9.1816168e-05],
       [7.6093352e-01],
       [6.0353044e-04],
       [3.6961806e-01],
       [7.6093352e-01],
       [7.6093352e-01],
```

```
            [3.5094842e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.1161455e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [3.6648637e-01],
            [1.6112577e-02],
            [9.6084050e-11],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
            [7.6093352e-01],
```

```
predictions.size
```

```
     114
```

```
class_labels = []


for i in range(predictions.size):
  if(predictions[i] > 0.5):
    class_labels.append(1)
  else:
    class_labels.append(0)
```

```
class_labels
```

```
     [1,
      1,
      1,
      0,
      1,
      0,
      0,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      0,
      1,
```

```
       1,
       1,
       1,
       1,
       1,
       1,
       0,
       0,
       0,
       0,
       1,
       0,
       0,
       1,
       1,
       0,
       1,
       1,
       1,
       1,
       1,
       1,
       1,
       1,
       1,
       1,
       1,
       1,
       0,
       0,
       0,
       1,
       1,
       1,
       1,
       1,
       1,
```

```
print(accuracy_score(y_test, class_labels))
```

```
0.8333333333333334
```

```
model1 = Sequential()
```

```
model1.add(Dense(15, activation = 'relu', input_dim = 30))
```

```
model1.add(Dense(10, activation = 'relu'))
model1.add(Dense(1, activation = 'sigmoid'))
```

```
model1.compile(loss = 'binary_crossentropy', optimizer = 'sgd', metrics = 'accuracy')
```

```
model1.fit(x_train, y_train, epochs = 100)
```

```
Epoch 1/100
15/15 [==============================] - 1s 2ms/step - loss: 109.9805 - accuracy: 0.6088
Epoch 2/100
```

```
15/15 [==============================] - 0s 2ms/step - loss: 0.6894 - accuracy: 0.6264
Epoch 3/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6872 - accuracy: 0.6264
Epoch 4/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6854 - accuracy: 0.6264
Epoch 5/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6836 - accuracy: 0.6264
Epoch 6/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6822 - accuracy: 0.6264
Epoch 7/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6807 - accuracy: 0.6264
Epoch 8/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6793 - accuracy: 0.6264
Epoch 9/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6780 - accuracy: 0.6264
Epoch 10/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6769 - accuracy: 0.6264
Epoch 11/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6758 - accuracy: 0.6264
Epoch 12/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6749 - accuracy: 0.6264
Epoch 13/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6738 - accuracy: 0.6264
Epoch 14/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6729 - accuracy: 0.6264
Epoch 15/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6722 - accuracy: 0.6264
Epoch 16/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6715 - accuracy: 0.6264
Epoch 17/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6709 - accuracy: 0.6264
Epoch 18/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6701 - accuracy: 0.6264
Epoch 19/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6693 - accuracy: 0.6264
Epoch 20/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6686 - accuracy: 0.6264
Epoch 21/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6680 - accuracy: 0.6264
Epoch 22/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6675 - accuracy: 0.6264
Epoch 23/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6671 - accuracy: 0.6264
Epoch 24/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6667 - accuracy: 0.6264
Epoch 25/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6662 - accuracy: 0.6264
Epoch 26/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6658 - accuracy: 0.6264
Epoch 27/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6655 - accuracy: 0.6264
Epoch 28/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6652 - accuracy: 0.6264
Epoch 29/100
15/15 [==============================] - 0s 2ms/step - loss: 0.6650 - accuracy: 0.6264
```

```
predictions1 = model.predict(x_test)
```

```
4/4 [==============================] - 0s 3ms/step
```

```
predictions1
```

```
array([[7.6093352e-01],
       [6.0712910e-01],
       [7.6093352e-01],
       [7.6255962e-02],
       [7.6093352e-01],
       [6.4233929e-02],
       [1.8969402e-02],
       [7.0078957e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [1.1428808e-02],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [5.9851474e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [6.9900067e-04],
       [1.6016141e-08],
       [2.4602570e-01],
       [9.1816168e-05],
       [7.6093352e-01],
       [6.0353044e-04],
       [3.6961806e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [3.5094842e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.1161455e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [3.6648637e-01],
       [1.6112577e-02],
       [9.6084050e-11],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
       [7.6093352e-01],
```

```
class_labels1 = []


for i in range(predictions1.size):
  if (predictions1[i] > 0.5):
    class_labels1.append(1)
  else:
    class_labels1.append(0)


class_labels1
```

```
[1,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 1,
 0,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 0,
 1,
 0,
 1,
 1,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 1,
 1,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 0,
 1,
 1,
 1,
```

```
       0,
       0,
       0,
       1,
       1,
       1,
       1,
       1,
       0,
```

```
print(accuracy_score(y_test, class_labels1))
```

```
     0.9473684210526315
```

**Practical No. 3**

**Aim: Stochastics Gradient Descent Optimizers and Comparing there Results**

```
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_moons
```

```
X, y = make_moons(n_samples = 100, noise = 0.2, random_state = 1)
```

```
X.shape
```

```
(100, 2)
```

```
y
```

```
array([1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0])
```

```
n_train = 30
```

```
x_train, x_test = X[:n_train], X[n_train:]
```

```
x_train
```

```
array([[ 1.36698238, -0.23541584],
       [ 1.76404402, -0.34563288],
       [-0.37868174,  0.41004375],
       [ 1.15113747, -0.13597622],
       [ 2.31168314,  0.32295125],
       [ 0.53866045,  0.73704603],
       [-0.93583639,  1.00686001],
       [ 1.32563024, -0.13540284],
       [ 0.75398022, -0.37261326],
       [ 0.42764536, -0.38163078],
       [ 1.86426147, -0.04447877],
       [-0.83255618,  0.71258899],
       [ 0.22507434,  0.27186939],
       [ 0.21098847,  0.2244366 ],
       [ 0.05359594,  0.20843943],
       [-0.14691002,  0.85157719],
       [-0.75060111,  0.86559773],
       [-0.91043983,  0.56466749],
       [ 1.21265683, -0.6289975 ],
       [ 0.64660318,  0.72011433],
       [ 0.68237381,  0.08692362],
       [ 0.61031853, -0.48706856],
       [ 1.99715631,  0.24389764],
       [ 0.2077874 , -0.42914145],
       [ 0.35027414,  0.74199317],
       [-0.68407286,  0.66774772],
       [ 1.76290471,  0.02997561],
       [-0.13029538,  0.8163553 ],
       [-1.34872432, -0.03686236],
       [-0.879935  ,  0.82462905]])
```

```
x_train.shape
```

```
(30, 2)
```

```
x_test
```

```
array([[ 0.69322932, -0.08679283],
       [ 2.30858186,  0.41045366],
       [-0.32283333, -0.11202163],
       [ 1.76628178,  0.16627015],
```

```
            [ 1.86869763,  0.11172855],
            [-0.81925203,  0.48890104],
            [-1.07708794, -0.07841617],
            [-0.10935722, -0.21307804],
            [-0.00275911,  0.96708982],
            [-0.74705081,  1.02954741],
            [ 1.72626831, -0.15849493],
            [-0.04902655,  0.33099072],
            [ 1.27447994, -0.34985574],
            [ 0.91687068,  0.33577803],
            [ 1.21815598, -0.34586299],
            [-0.886625  ,  0.65913593],
            [-0.89381479,  0.15139579],
            [ 1.72422728, -0.04288797],
            [ 0.13871945,  1.04830955],
            [ 1.69682347,  0.64749479],
            [-0.89185761,  0.7268536 ],
            [-0.09457701,  0.28788561],
            [-0.10713202,  1.35521438],
            [ 1.15027259,  0.41680378],
            [ 0.27096453, -0.02497953],
            [-1.04069441,  0.08443341],
            [ 0.49389774,  0.9326833 ],
            [ 1.98289223, -0.5145263 ],
            [ 0.96245599,  0.13005483],
            [ 1.01374105, -0.59939387],
            [-0.35030927,  0.87891408],
            [ 0.46258369, -0.30221636],
            [ 0.52237109, -0.4723288 ],
            [-0.69719401,  0.43550886],
            [-1.34861991,  0.18401244],
            [ 1.95593429, -0.154812  ],
            [ 0.85676993,  0.58746968],
            [ 0.38380987, -0.467539  ],
            [ 1.05339036, -0.02197593],
            [ 1.69685427, -0.20380166],
            [ 0.02257804,  0.17118028],
            [ 0.19671026,  1.25356483],
            [ 0.31211999, -0.4038874 ],
            [ 0.30297732, -0.35095037],
            [ 1.11049974,  0.47772861],
            [ 1.84316777,  0.4702879 ],
            [ 0.59461521,  0.46711159],
            [ 0.85910841, -0.61549596],
            [ 1.53155945, -0.23913659],
            [ 0.57626466,  0.44048843],
            [-0.54006984,  0.8207721 ],
            [-0.21688266,  0.89003157],
            [ 0.79663915,  0.43646711],
            [ 0.34766968,  0.67404085],
            [ 0.05729368,  1.14428225],
            [ 0.01880585,  0.80152232],
            [ 0.19365301,  1.02502389],
```

```
x_test.shape
```

```
(70, 2)
```

```
y_train, y_test = y[:n_train], y[n_train:]
```

```
y_train
```

```
array([1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0])
```

```
y_train.shape
```

```
(30,)
```

```
y_test
```

```
array([1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 1, 0, 0])
```

```
y_test.shape
```

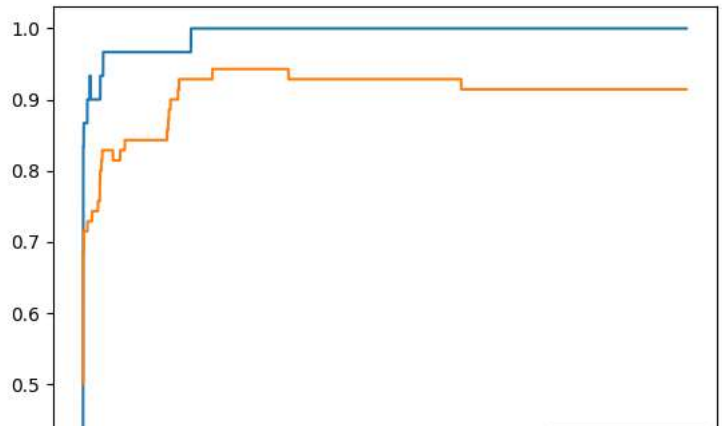
```
(70,)
```

```
model = Sequential()


model.add(Dense(500, input_dim = 2, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = 'accuracy')


history = model.fit(x_train, y_train, validation_data = (x_test, y_test), epochs = 4000)
```

```
Streaming output truncated to the last 5000 lines.
1/1 [==============================] - 0s 85ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3200 - val_accuracy: 0.9286
Epoch 1502/4000
1/1 [==============================] - 0s 74ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3201 - val_accuracy: 0.9286
Epoch 1503/4000
1/1 [==============================] - 0s 69ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3202 - val_accuracy: 0.9286
Epoch 1504/4000
1/1 [==============================] - 0s 73ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3203 - val_accuracy: 0.9286
Epoch 1505/4000
1/1 [==============================] - 0s 71ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3204 - val_accuracy: 0.9286
Epoch 1506/4000
1/1 [==============================] - 0s 80ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3205 - val_accuracy: 0.9286
Epoch 1507/4000
1/1 [==============================] - 0s 77ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3206 - val_accuracy: 0.9286
Epoch 1508/4000
1/1 [==============================] - 0s 71ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3208 - val_accuracy: 0.9286
Epoch 1509/4000
1/1 [==============================] - 0s 75ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3209 - val_accuracy: 0.9286
Epoch 1510/4000
1/1 [==============================] - 0s 74ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3210 - val_accuracy: 0.9286
Epoch 1511/4000
1/1 [==============================] - 0s 80ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3211 - val_accuracy: 0.9286
Epoch 1512/4000
1/1 [==============================] - 0s 73ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.3213 - val_accuracy: 0.9286
Epoch 1513/4000
1/1 [==============================] - 0s 79ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3214 - val_accuracy: 0.9286
Epoch 1514/4000
1/1 [==============================] - 0s 64ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3215 - val_accuracy: 0.9286
Epoch 1515/4000
1/1 [==============================] - 0s 85ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3216 - val_accuracy: 0.9286
Epoch 1516/4000
1/1 [==============================] - 0s 73ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3217 - val_accuracy: 0.9286
Epoch 1517/4000
1/1 [==============================] - 0s 58ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3218 - val_accuracy: 0.9286
Epoch 1518/4000
1/1 [==============================] - 0s 59ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3219 - val_accuracy: 0.9286
Epoch 1519/4000
1/1 [==============================] - 0s 67ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3220 - val_accuracy: 0.9286
Epoch 1520/4000
1/1 [==============================] - 0s 67ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3221 - val_accuracy: 0.9286
Epoch 1521/4000
1/1 [==============================] - 0s 72ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3223 - val_accuracy: 0.9286
Epoch 1522/4000
1/1 [==============================] - 0s 69ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3224 - val_accuracy: 0.9286
Epoch 1523/4000
1/1 [==============================] - 0s 79ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3225 - val_accuracy: 0.9286
Epoch 1524/4000
1/1 [==============================] - 0s 75ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3226 - val_accuracy: 0.9286
Epoch 1525/4000
1/1 [==============================] - 0s 84ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3227 - val_accuracy: 0.9286
Epoch 1526/4000
1/1 [==============================] - 0s 67ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.3228 - val_accuracy: 0.9286
Epoch 1527/4000
1/1 [==============================] - 0s 71ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.3229 - val_accuracy: 0.9286
Epoch 1528/4000
1/1 [==============================] - 0s 70ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.3231 - val_accuracy: 0.9286
Epoch 1529/4000
1/1 [==============================] - 0s 68ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.3232 - val_accuracy: 0.9286
```

```
plt.plot(history.history['accuracy'], label = 'train')
plt.plot(history.history['val_accuracy'], label = 'validation')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f699ca0d360>
```




L2 Regularisation

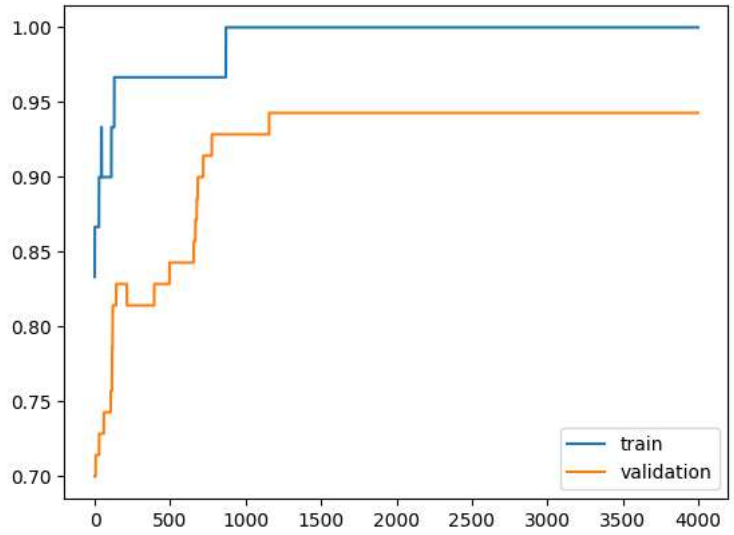
```
model_l2 = Sequential()


from keras.regularizers import l2


model_l2.add(Dense(500, input_dim = 2, activation = 'relu', kernel_regularizer = l2(0.001)))
model_l2.add(Dense(1, activation = 'sigmoid'))


model_l2.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = 'accuracy')


history_l2 = model_l2.fit(x_train, y_train, validation_data = (x_test, y_test), epochs = 4000)
```

```
Streaming output truncated to the last 5000 lines.
1/1 [==============================] - 0s 67ms/step - loss: 0.0360 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1502/4000
1/1 [==============================] - 0s 107ms/step - loss: 0.0360 - accuracy: 1.0000 - val_loss: 0.2511 - val_accuracy: 0.9429
Epoch 1503/4000
1/1 [==============================] - 0s 79ms/step - loss: 0.0360 - accuracy: 1.0000 - val_loss: 0.2511 - val_accuracy: 0.9429
Epoch 1504/4000
1/1 [==============================] - 0s 130ms/step - loss: 0.0360 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1505/4000
1/1 [==============================] - 0s 93ms/step - loss: 0.0359 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1506/4000
1/1 [==============================] - 0s 84ms/step - loss: 0.0359 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1507/4000
1/1 [==============================] - 0s 121ms/step - loss: 0.0359 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1508/4000
1/1 [==============================] - 0s 118ms/step - loss: 0.0359 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1509/4000
1/1 [==============================] - 0s 67ms/step - loss: 0.0359 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1510/4000
1/1 [==============================] - 0s 112ms/step - loss: 0.0359 - accuracy: 1.0000 - val_loss: 0.2510 - val_accuracy: 0.9429
Epoch 1511/4000
1/1 [==============================] - 0s 114ms/step - loss: 0.0358 - accuracy: 1.0000 - val_loss: 0.2511 - val_accuracy: 0.9429
Epoch 1512/4000
1/1 [==============================] - 0s 117ms/step - loss: 0.0358 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1513/4000
1/1 [==============================] - 0s 109ms/step - loss: 0.0358 - accuracy: 1.0000 - val_loss: 0.2511 - val_accuracy: 0.9429
Epoch 1514/4000
1/1 [==============================] - 0s 138ms/step - loss: 0.0358 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1515/4000
1/1 [==============================] - 0s 133ms/step - loss: 0.0358 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1516/4000
1/1 [==============================] - 0s 126ms/step - loss: 0.0357 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1517/4000
1/1 [==============================] - 0s 106ms/step - loss: 0.0357 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1518/4000
1/1 [==============================] - 0s 79ms/step - loss: 0.0357 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1519/4000
1/1 [==============================] - 0s 95ms/step - loss: 0.0357 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1520/4000
1/1 [==============================] - 0s 82ms/step - loss: 0.0357 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1521/4000
1/1 [==============================] - 0s 79ms/step - loss: 0.0356 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1522/4000
1/1 [==============================] - 0s 91ms/step - loss: 0.0356 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1523/4000
```

```
1/1 [==============================] - 0s 90ms/step - loss: 0.0356 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1524/4000
1/1 [==============================] - 0s 77ms/step - loss: 0.0356 - accuracy: 1.0000 - val_loss: 0.2513 - val_accuracy: 0.9429
Epoch 1525/4000
1/1 [==============================] - 0s 77ms/step - loss: 0.0356 - accuracy: 1.0000 - val_loss: 0.2513 - val_accuracy: 0.9429
Epoch 1526/4000
1/1 [==============================] - 0s 83ms/step - loss: 0.0355 - accuracy: 1.0000 - val_loss: 0.2513 - val_accuracy: 0.9429
Epoch 1527/4000
1/1 [==============================] - 0s 76ms/step - loss: 0.0355 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1528/4000
1/1 [==============================] - 0s 72ms/step - loss: 0.0355 - accuracy: 1.0000 - val_loss: 0.2512 - val_accuracy: 0.9429
Epoch 1529/4000
```

```
plt.plot(history_l2.history['accuracy'], label = 'train')
plt.plot(history_l2.history['val_accuracy'], label = 'validation')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f699d016140>
```




L1 regularisation

```
model_l1 = Sequential()
```

```
from keras.regularizers import l1
```

```
model_l1.add(Dense(500, input_dim = 2, activation = 'relu', kernel_regularizer = l1(0.001)))
model_l1.add(Dense(1, activation = 'sigmoid'))
```

```
model_l1.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = 'accuracy')
```

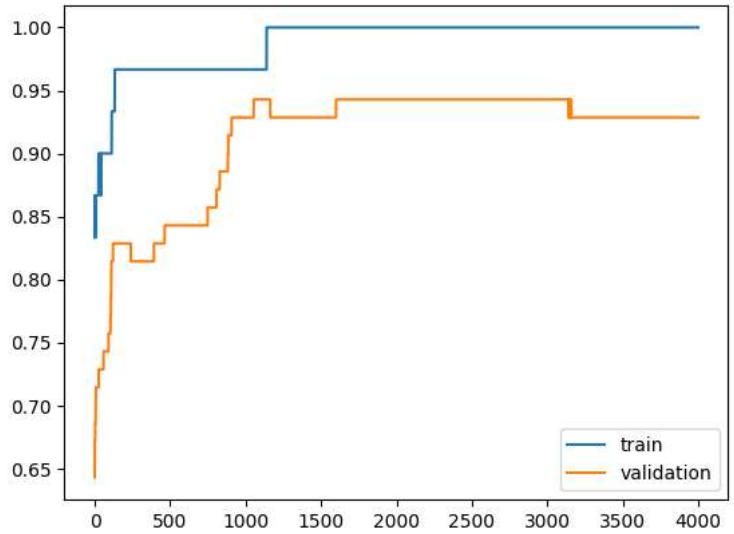
```
history_l1 = model_l1.fit(x_train, y_train, validation_data = (x_test, y_test), epochs = 4000)
```

```
Streaming output truncated to the last 5000 lines.
1/1 [==============================] - 0s 67ms/step - loss: 0.0614 - accuracy: 1.0000 - val_loss: 0.2166 - val_accuracy: 0.9286
Epoch 1502/4000
1/1 [==============================] - 0s 78ms/step - loss: 0.0613 - accuracy: 1.0000 - val_loss: 0.2166 - val_accuracy: 0.9286
Epoch 1503/4000
1/1 [==============================] - 0s 81ms/step - loss: 0.0613 - accuracy: 1.0000 - val_loss: 0.2167 - val_accuracy: 0.9286
Epoch 1504/4000
1/1 [==============================] - 0s 94ms/step - loss: 0.0612 - accuracy: 1.0000 - val_loss: 0.2167 - val_accuracy: 0.9286
Epoch 1505/4000
1/1 [==============================] - 0s 77ms/step - loss: 0.0612 - accuracy: 1.0000 - val_loss: 0.2166 - val_accuracy: 0.9286
Epoch 1506/4000
1/1 [==============================] - 0s 68ms/step - loss: 0.0611 - accuracy: 1.0000 - val_loss: 0.2166 - val_accuracy: 0.9286
Epoch 1507/4000
1/1 [==============================] - 0s 73ms/step - loss: 0.0611 - accuracy: 1.0000 - val_loss: 0.2165 - val_accuracy: 0.9286
Epoch 1508/4000
1/1 [==============================] - 0s 95ms/step - loss: 0.0610 - accuracy: 1.0000 - val_loss: 0.2165 - val_accuracy: 0.9286
Epoch 1509/4000
1/1 [==============================] - 0s 97ms/step - loss: 0.0610 - accuracy: 1.0000 - val_loss: 0.2164 - val_accuracy: 0.9286
Epoch 1510/4000
1/1 [==============================] - 0s 64ms/step - loss: 0.0609 - accuracy: 1.0000 - val_loss: 0.2163 - val_accuracy: 0.9286
Epoch 1511/4000
```

```
1/1 [==============================] - 0s 74ms/step - loss: 0.0609 - accuracy: 1.0000 - val_loss: 0.2163 - val_accuracy: 0.9286
Epoch 1512/4000
1/1 [==============================] - 0s 72ms/step - loss: 0.0608 - accuracy: 1.0000 - val_loss: 0.2163 - val_accuracy: 0.9286
Epoch 1513/4000
1/1 [==============================] - 0s 123ms/step - loss: 0.0608 - accuracy: 1.0000 - val_loss: 0.2162 - val_accuracy: 0.9286
Epoch 1514/4000
1/1 [==============================] - 0s 66ms/step - loss: 0.0607 - accuracy: 1.0000 - val_loss: 0.2162 - val_accuracy: 0.9286
Epoch 1515/4000
1/1 [==============================] - 0s 73ms/step - loss: 0.0607 - accuracy: 1.0000 - val_loss: 0.2162 - val_accuracy: 0.9286
Epoch 1516/4000
1/1 [==============================] - 0s 78ms/step - loss: 0.0607 - accuracy: 1.0000 - val_loss: 0.2161 - val_accuracy: 0.9286
Epoch 1517/4000
1/1 [==============================] - 0s 70ms/step - loss: 0.0606 - accuracy: 1.0000 - val_loss: 0.2161 - val_accuracy: 0.9286
Epoch 1518/4000
1/1 [==============================] - 0s 99ms/step - loss: 0.0606 - accuracy: 1.0000 - val_loss: 0.2160 - val_accuracy: 0.9286
Epoch 1519/4000
1/1 [==============================] - 0s 72ms/step - loss: 0.0605 - accuracy: 1.0000 - val_loss: 0.2160 - val_accuracy: 0.9286
Epoch 1520/4000
1/1 [==============================] - 0s 88ms/step - loss: 0.0605 - accuracy: 1.0000 - val_loss: 0.2160 - val_accuracy: 0.9286
Epoch 1521/4000
1/1 [==============================] - 0s 73ms/step - loss: 0.0604 - accuracy: 1.0000 - val_loss: 0.2160 - val_accuracy: 0.9286
Epoch 1522/4000
1/1 [==============================] - 0s 96ms/step - loss: 0.0604 - accuracy: 1.0000 - val_loss: 0.2160 - val_accuracy: 0.9286
Epoch 1523/4000
1/1 [==============================] - 0s 93ms/step - loss: 0.0603 - accuracy: 1.0000 - val_loss: 0.2160 - val_accuracy: 0.9286
Epoch 1524/4000
1/1 [==============================] - 0s 91ms/step - loss: 0.0603 - accuracy: 1.0000 - val_loss: 0.2159 - val_accuracy: 0.9286
Epoch 1525/4000
1/1 [==============================] - 0s 61ms/step - loss: 0.0602 - accuracy: 1.0000 - val_loss: 0.2159 - val_accuracy: 0.9286
Epoch 1526/4000
1/1 [==============================] - 0s 61ms/step - loss: 0.0602 - accuracy: 1.0000 - val_loss: 0.2159 - val_accuracy: 0.9286
Epoch 1527/4000
1/1 [==============================] - 0s 70ms/step - loss: 0.0601 - accuracy: 1.0000 - val_loss: 0.2159 - val_accuracy: 0.9286
Epoch 1528/4000
1/1 [==============================] - 0s 94ms/step - loss: 0.0601 - accuracy: 1.0000 - val_loss: 0.2160 - val_accuracy: 0.9286
Epoch 1529/4000
```

```
plt.plot(history_l1.history['accuracy'], label = 'train')
plt.plot(history_l1.history['val_accuracy'], label = 'validation')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f699cdc6560>
```



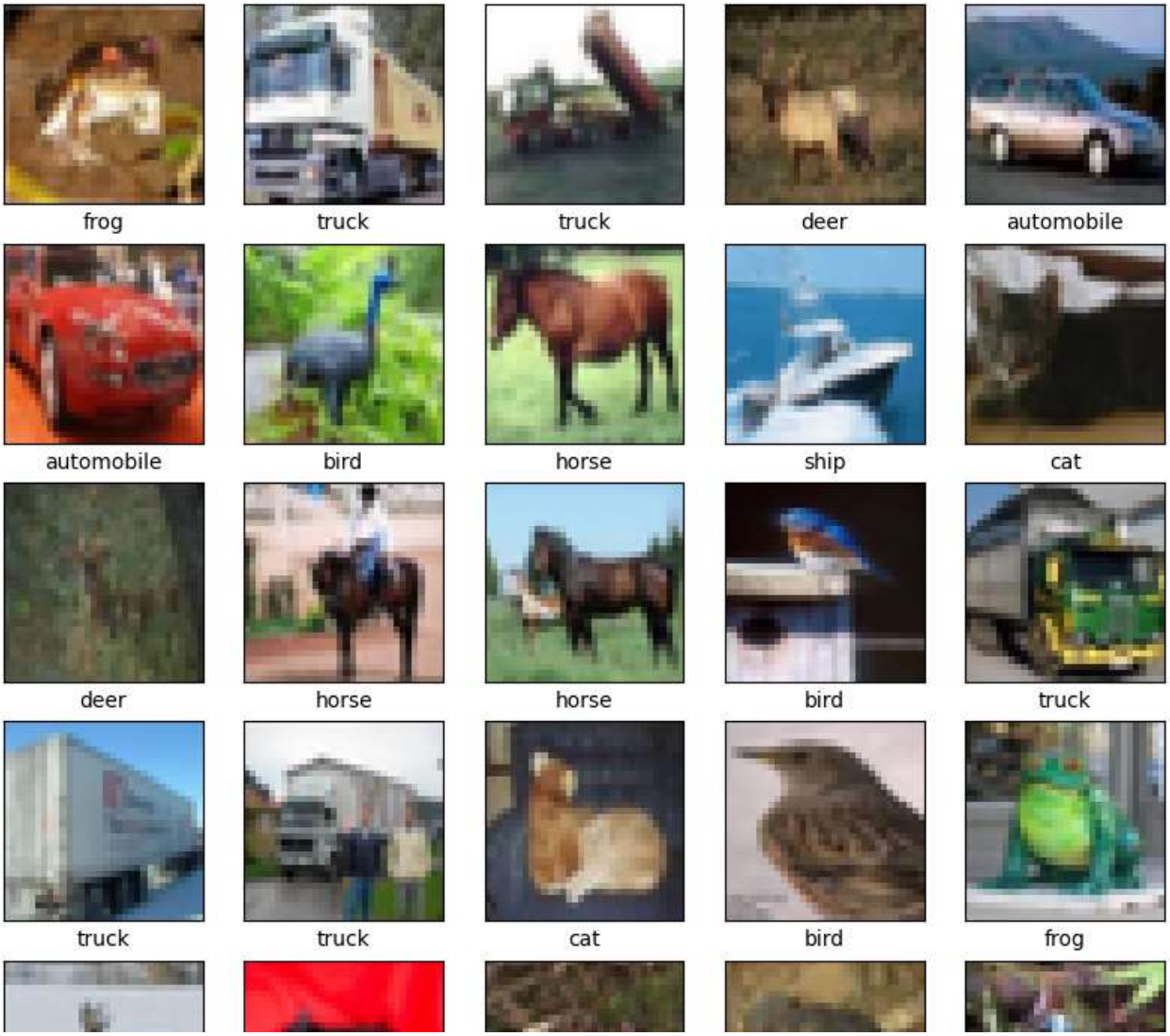
## Practical No. 4

## Aim: Convolutional Neural Network on CIFAR Dataset

```
from keras.datasets import cifar10

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D

import numpy as np

import matplotlib.pyplot as plt

( x_train, y_train ), (x_test, y_test) = cifar10.load_data()
```

```
    Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
    170498071/170498071 [==============================] - 11s 0us/step
```

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship
plt.figure(figsize=(10, 10))
for i in range(25):
  plt.subplot(5, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(x_train[i])
  plt.xlabel(class_names[y_train[i][0]])
plt.show()
```




```
x_train.shape
```

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

deer cat frog frog bird

```
x_test.shape
```

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

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

```
x_train /= 255
```

```
x_test /= 255
```

```
from keras.models import Sequential
model = Sequential()
```

```
model.add(Conv2D(32, (3, 3), padding="same", input_shape = x_train.shape[1:]))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(10))

model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer="adam", metrics="accuracy")

from keras.utils import to_categorical

y_train = to_categorical(y_train, 10)

y_train[0]
```

```
array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)
```

```
y_test = to_categorical(y_test, 10)

model.fit(x_train, y_train, batch_size=260, epochs=3)
```

```
Epoch 1/3
193/193 [==============================] - 82s 420ms/step - loss: 1.6306 - accuracy: 0.4
Epoch 2/3
193/193 [==============================] - 80s 414ms/step - loss: 1.3157 - accuracy: 0.5
Epoch 3/3
193/193 [==============================] - 80s 414ms/step - loss: 1.2048 - accuracy: 0.5
<keras.callbacks.History at 0x7f295cefbd30>
```

```
predictions = model.predict(x_test)
for i in range(5):
  print(y_test[i])
  print(predictions[i])
```

```
    313/313 [==============================] - 7s 21ms/step
    [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
    [1.2926904e-02 6.4506085e-04 3.7113272e-02 5.5824965e-01 1.3839691e-02
     3.4183404e-01 2.0514002e-02 3.6730543e-03 1.0814722e-02 3.8961717e-04]
    [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
    [1.1162371e-01 3.7739566e-01 2.8830691e-04 5.7039517e-05 9.5591655e-05
     3.4233992e-06 2.2089525e-05 2.0494715e-06 4.8432919e-01 2.6182899e-02]
    [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
    [2.4837700e-01 8.4696516e-02 1.3544215e-02 7.6984102e-03 4.2522913e-03
     1.1331713e-03 3.7455614e-04 1.7913333e-03 5.7991070e-01 5.8221817e-02]
    [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
    [6.0585666e-01 1.0444272e-02 4.2456090e-02 1.5233783e-03 8.9421533e-03
     2.1772734e-04 8.3546870e-04 4.1096259e-04 3.2291490e-01 6.3983430e-03]
    [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
    [1.8565614e-04 8.2820363e-05 6.1406154e-02 1.2850280e-01 3.7702543e-01
     3.5299309e-02 3.9233810e-01 4.8352317e-03 1.9762994e-04 1.2696354e-04]
```

## Practical No. 5

## Aim: Implementation Convolutional Neural Network on MNIST Dataset

```
from keras.datasets import mnist
```

```
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
from keras.models import Sequential
```

```
import matplotlib.pyplot as plt
```

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

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

```
train_images.shape
```

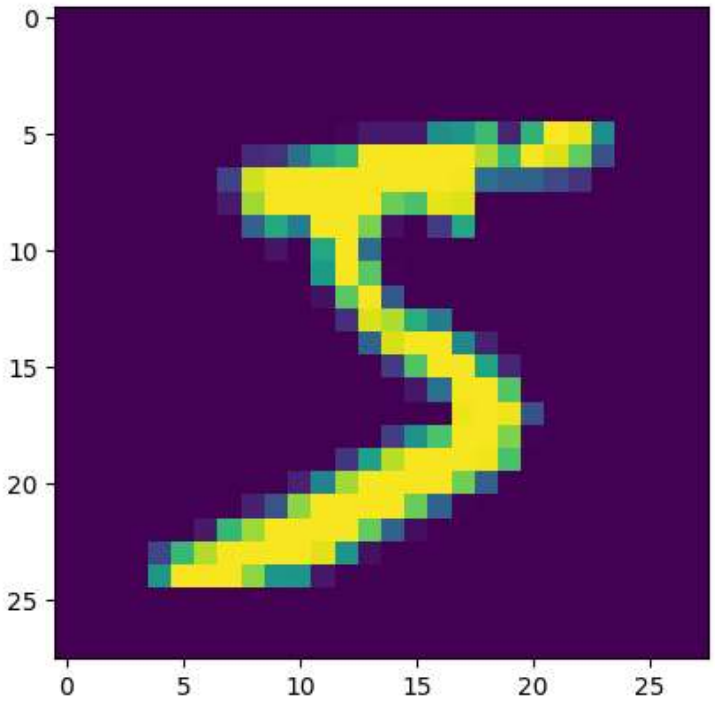
```
    (60000, 28, 28)
```

```
test_images.shape
```

```
    (10000, 28, 28)
```

```
plt.imshow(train_images[0])
```

```
    <matplotlib.image.AxesImage at 0x7f0bcb5aaec0>
```

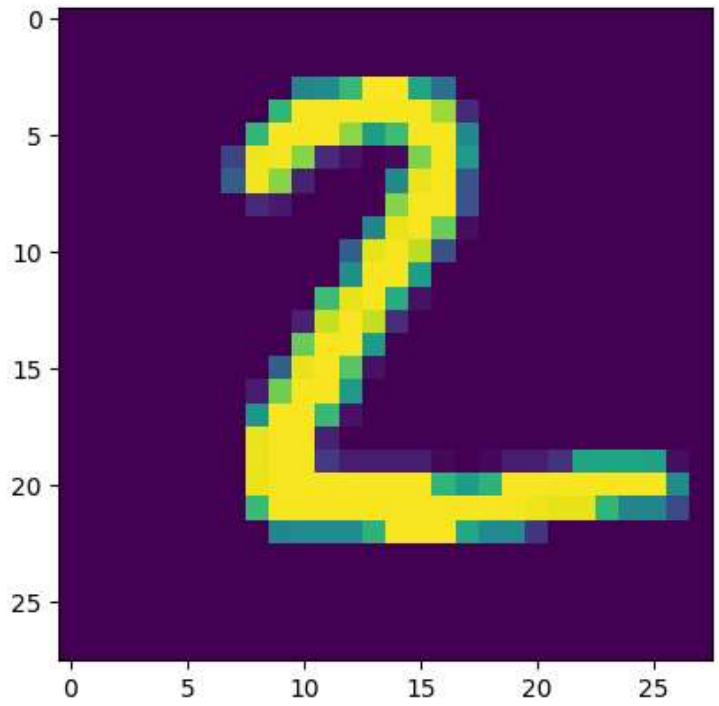


```
train_labels[0]
```

```
5
```

```
plt.imshow(test_images[1])
```

```
<matplotlib.image.AxesImage at 0x7f0bc87beaa0>
```




```
test_labels[1]
```

```
2
```

```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
import numpy as np
```

```
train_images = train_images.reshape(60000, 28, 28, 1)
test_images = test_images.reshape(10000, 28, 28, 1)
```

```
from keras.utils import to_categorical
```

```
train_labels = to_categorical(train_labels)
```

```
test_labels = to_categorical(test_labels)
```

```
test_labels[1]
```

```
array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
train_labels[1]
```

```
     array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)


cnn_model = Sequential()


cnn_model.add(Conv2D(32, (3, 3), activation = 'relu',
                     input_shape = (28, 28, 1)))
cnn_model.add(MaxPooling2D(2, 2))
cnn_model.add(Conv2D(64, (3, 3), activation = 'relu'))
cnn_model.add(Flatten())
cnn_model.add(Dense(10, activation = 'softmax'))


cnn_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
                  metrics = 'accuracy')


cnn_model.fit(train_images, train_labels, epochs = 3)

     Epoch 1/3
     1875/1875 [==============================] - 70s 36ms/step - loss: 0.1299 - accuracy: 0.9606
     Epoch 2/3
     1875/1875 [==============================] - 57s 30ms/step - loss: 0.0451 - accuracy: 0.9867
     Epoch 3/3
     1875/1875 [==============================] - 56s 30ms/step - loss: 0.0324 - accuracy: 0.9900
     <keras.callbacks.History at 0x7f0bcb411930>


predictions = cnn_model.predict(test_images)

     313/313 [==============================] - 4s 12ms/step


for i in range(3):
  print("Predictions", predictions[i])
  print("Test Labels", test_labels[i])

     Predictions [3.3264688e-10 2.1316327e-11 1.1563329e-06 1.5999116e-06 2.1771649e-11
      5.6543898e-11 5.5886281e-16 9.9999684e-01 5.7116383e-09 3.4036998e-07]
     Test Labels [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
     Predictions [9.2074833e-07 1.0564368e-06 9.9999219e-01 7.3657713e-10 2.9802199e-11
      1.1779913e-11 5.6247491e-06 6.9878122e-12 1.4509399e-07 2.5135127e-13]
     Test Labels [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
     Predictions [5.2429419e-07 9.9932617e-01 1.6355267e-05 1.1128916e-07 3.3275786e-04
      3.6343110e-06 2.3185432e-05 1.6421058e-04 1.3250085e-04 3.9722966e-07]
     Test Labels [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
```

## Practical No. 6

## Aim: Transfer Learning

```
import tensorflow as tf
```

```
url='https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
```

```
pathtozip=tf.keras.utils.get_file('cats_and_dogs.zip',origin=url,extract=True)
```

```
    Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
    68606236/68606236 [==============================] - 1s 0us/step
```

```
import os
```

```
path=os.path.join(os.path.dirname(pathtozip),'cats_and_dogs_filtered')
```

```
train_dir=os.path.join(path,'train')
```

```
val_dir=os.path.join(path,'validation')
```

```
train_dataset=tf.keras.utils.image_dataset_from_directory(train_dir,
                                                          shuffle=True,
                                                          batch_size=32,
                                                          image_size=(160,160))
```

```
    Found 2000 files belonging to 2 classes.
```

```
val_dataset=tf.keras.utils.image_dataset_from_directory(val_dir,shuffle=True,
                                                        batch_size=32,
                                                        image_size=(160,160))
```

```
    Found 1000 files belonging to 2 classes.
```

```
val_batches=tf.data.experimental.cardinality(val_dataset)
test_dataset=val_dataset.take(val_batches // 5)
print(val_batches)
```

```
    tf.Tensor(32, shape=(), dtype=int64)
```

```
tf.data.experimental.cardinality(test_dataset)
```

```
    <tf.Tensor: shape=(), dtype=int64, numpy=6>
```

```
val_dataset = val_dataset.skip(val_batches // 5)
```

```
tf.data.experimental.cardinality(val_dataset)
```

```
<tf.Tensor: shape=(), dtype=int64, numpy=26>
```

```
AUTOTUNE=tf.data.AUTOTUNE
train_dataset=train_dataset.prefetch(buffer_size=AUTOTUNE)
val_dataset=val_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset=test_dataset.prefetch(buffer_size=AUTOTUNE)
```

```
data_augmentation=tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),])
```

```
preprocess_input=tf.keras.applications.mobilenet_v2.preprocess_input
```

```
BATCH_SIZE = 32
IMG_SIZE = (160, 160)
IMG_SHAPE = IMG_SIZE + (3,)
```

```
base_model=tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                             include_top=False,
                                             weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobil
9406464/9406464 [==============================] - 0s 0us/step
```

```
base_model.trainable=False
```

```
inputs=tf.keras.Input(shape=(160,160,3))
```

```
X=data_augmentation(inputs)
```

```
X=preprocess_input(X)
```

```
X=base_model(X,training=False)
```

```
global_average_layer=tf.keras.layers.GlobalAveragePooling2D()
```

```
#feature_batch_average=global_average_layer(feature_batch)
```

```
X=global_average_layer(X)
```

```
X=tf.keras.layers.Dropout(0.2)(X)
```

```
prediction_layer=tf.keras.layers.Dense(1)
```

```
outputs=prediction_layer(X)
```

```
model=tf.keras.Model(inputs,outputs)


model.compile(optimizer='adam',loss='binary_crossentropy',metrics='accuracy')


model.fit(train_dataset,epochs=3,validation_data=val_dataset)
```

```
Epoch 1/3
63/63 [==============================] - 46s 645ms/step - loss: 2.3293 - accuracy: 0.7880 - val_loss
Epoch 2/3
63/63 [==============================] - 36s 572ms/step - loss: 1.1925 - accuracy: 0.8820 - val_loss
Epoch 3/3
63/63 [==============================] - 37s 576ms/step - loss: 1.2357 - accuracy: 0.8810 - val_loss
<keras.callbacks.History at 0x7f9a84157880>
```