CO4

PROGRAM : 1

AIM:

Programs on feedforward network to classify any standard dataset available in the public domain.

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from matplotlib import pyplot as plt
```

Load Data

```
(x_train,y_train),(x_valid,y_valid)=mnist.load_data()
```

```
type(x_train)
```

```
    numpy.ndarray
```

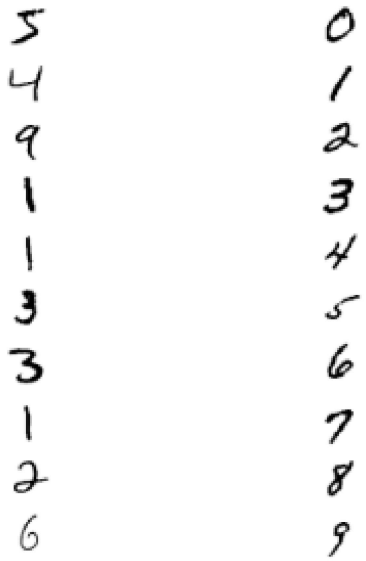
```
x_train.shape
```

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

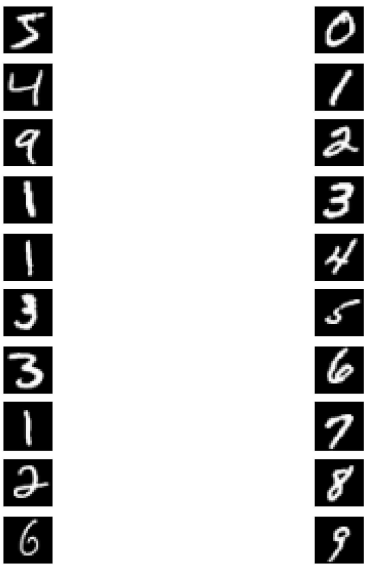
```
y_train[0:12]
```

```
    array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5], dtype=uint8)
```

```
plt.figure(figsize=(5,5))
for k in range(20):
  plt.subplot(10,2,k+1)
  plt.imshow(x_train[k],cmap='Greys')
  plt.axis('off')
plt.show()
```

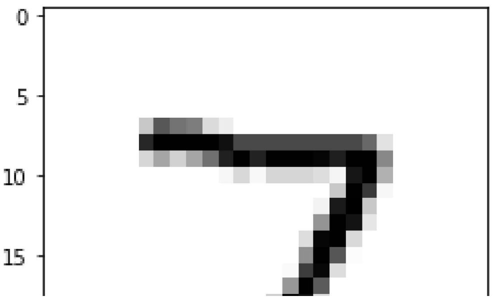


```
plt.figure(figsize=(5,5))
for k in range(20):
  plt.subplot(10,2,k+1)
  plt.imshow(x_train[k],cmap='Greys_r')
  plt.axis('off')
plt.show()
```



```
plt.imshow(x_valid[0],cmap='Greys')
```

```
<matplotlib.image.AxesImage at 0x7f7e9673cdd0>
```



```
x_valid[0]
```

```
          0,   0],
       [  0,   0,   0,   0,   0,   0, 222, 254, 254, 254, 254, 241, 198,
        198, 198, 198, 198, 198, 198, 198, 170,  52,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  67, 114,  72, 114, 163, 227, 254,
        225, 254, 254, 254, 250, 229, 254, 254, 140,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  17,  66,
         14,  67,  67,  67,  59,  21, 236, 254, 106,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,  83, 253, 209,  18,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,  22, 233, 255,  83,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 129, 254, 238,  44,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  59, 249, 254,  62,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0, 133, 254, 187,   5,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   9, 205, 248,  58,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, 126, 254, 182,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  75, 251, 240,  57,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         19, 221, 254, 166,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
        203, 254, 219,  35,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  38,
        254, 254,  77,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
  0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  31, 224,
 254, 115,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 133, 254,
 254,  52,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  61, 242, 254,
 254,  52,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 121, 254, 254,
 219,  40,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 121, 254, 207,
  18,   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,
```

Preprocess Data

```
x_train=x_train.reshape(60000,784).astype('float32')
x_valid=x_valid.reshape(10000,784).astype('float32')
```

```
x_train/=225
x_valid/=225
```

```
x_valid[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.        , 0.04      , 0.9111111 ,
       1.1022222 , 0.25777778, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.56      , 1.1288888 , 0.8088889 , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.33333334, 1.1155555 ,
       1.0666667 , 0.25333333, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.08444444, 0.9822222 , 1.1288888 , 0.73777777, 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.        , 0.01333333, 0.9022222 , 1.1288888 ,
0.97333336, 0.15555556, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.16888888, 1.1288888 , 1.1288888 , 0.3422222 , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.13777778, 0.9955556 , 1.1288888 ,
0.51111114, 0.00444444, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,

0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.5911111 , 1.1288888 , 1.1288888 , 0.23111111, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.2711111 , 1.0755556 , 1.1288888 ,
1.1288888 , 0.23111111, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.5377778 , 1.1288888 , 1.1288888 , 0.97333336, 0.17777778,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
n_classes=10
y_train=keras.utils.np_utils.to_categorical(y_train,n_classes)
y_valid=keras.utils.np_utils.to_categorical(y_valid,n_classes)
```

```
y_valid[0]
```

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

```
model=Sequential()
```

```
model.add(Dense(64,activation='sigmoid',input_shape=(784,)))
```

```
model.add(Dense(10,activation='softmax'))
```

(64*784)

```
50176
```

(64*784)+64

```
50240
```

(1064)+10+(64784)+64

```
65922
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_2 (Dense)              (None, 64)                50240

 dense_3 (Dense)              (None, 10)                650

=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```

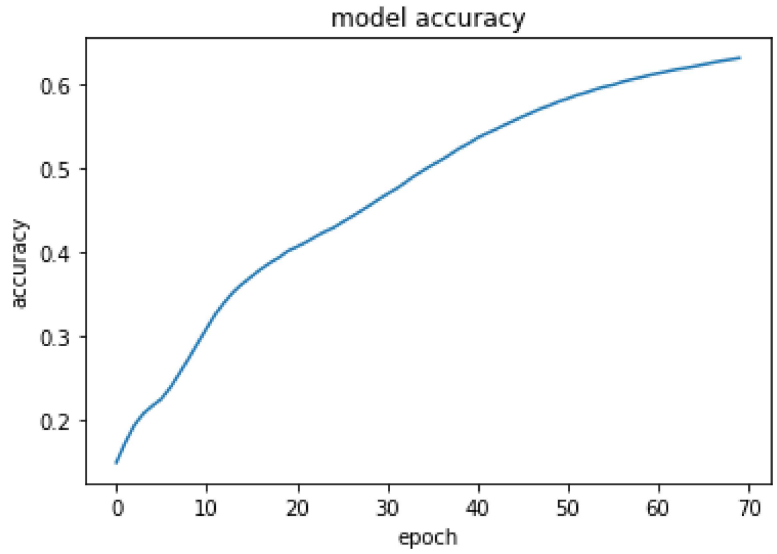
```
model.compile(loss='mean_squared_error',optimizer=SGD(learning_rate=0.01),metrics=['a
```

```
history=model.fit(x_train, y_train, batch_size=128, epochs=70, verbose=1)
```

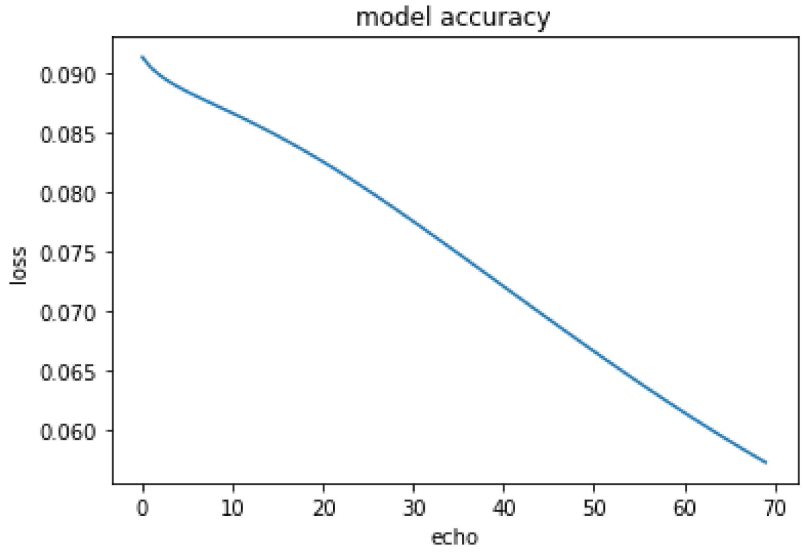
```
Epoch 1/70
469/469 [==============================] - 2s 3ms/step - loss: 0.0914 - accura
Epoch 2/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0904 - accura
Epoch 3/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0898 - accura
Epoch 4/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0893 - accura
Epoch 5/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0888 - accura
Epoch 6/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0884 - accura
Epoch 7/70
```

```
469/469 [==============================] - 1s 3ms/step - loss: 0.0881 - accura
Epoch 8/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0877 - accura
Epoch 9/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0873 - accura
Epoch 10/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0870 - accura
Epoch 11/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0866 - accura
Epoch 12/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0863 - accura
Epoch 13/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0859 - accura
Epoch 14/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0855 - accura
Epoch 15/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0851 - accura
Epoch 16/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0847 - accura
Epoch 17/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0843 - accura
Epoch 18/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0839 - accura
Epoch 19/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0834 - accura
Epoch 20/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0830 - accura
Epoch 21/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0825 - accura
Epoch 22/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0821 - accura
Epoch 23/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0816 - accura
Epoch 24/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0811 - accura
Epoch 25/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0806 - accura
Epoch 26/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0801 - accura
Epoch 27/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0796 - accura
Epoch 28/70
469/469 [==============================] - 1s 3ms/step - loss: 0.0791 - accura
Epoch 29/70
```

```
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.title('model accuracy')
plt.ylabel('loss')
plt.xlabel('echo')
plt.show()
```



RESULT:

Program is executed successfully and output is obtained.

PROGRAM : 2

AIM:

Programs on convolutional neural network to classify images from any standard dataset in the public domain.

DATASET: cifar10

```
import tensorflow as tf
from tensorflow.keras import datasets,layers,models
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report
```

```
(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
170500096/170498071 [==============================] - 4s 0us/step
170508288/170498071 [==============================] - 4s 0us/step
```

```
x_train.shape
```

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

```
x_test.shape
```

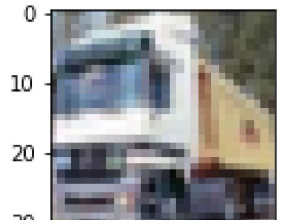
```
(10000, 32, 32, 3)
```

```
classes=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','tru
```

```
def plot_sample(x,y,index):
  plt.figure(figsize=(14,2))
  plt.imshow(x[index])
  plt.xlabel(y[index])
```

```
plot_sample(x_train,y_train,1)
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/text.py:1165: FutureWarning: e
  if s != self._text:
```




```
#normalize
x_train=x_train/255
x_test=x_test/255


#model
cnn=models.Sequential([
                      #feature extraction
                      layers.Conv2D(filters=32,activation='relu',kernel_size=(3,3),i
                      layers.MaxPool2D((2,2)),

                      layers.Conv2D(filters=32,activation='relu',kernel_size=(3,3),i
                      layers.MaxPool2D((2,2)),

                      #classification
                      layers.Flatten(),
                      layers.Dense(64,activation='relu'),
                      layers.Dense(10,activation='softmax')


])

cnn.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accurac



cnn.fit(x_train,y_train,epochs=20)
```

```
Epoch 1/20
1563/1563 [==============================] - 82s 52ms/step - loss: 1.4751 - accu
Epoch 2/20
1563/1563 [==============================] - 47s 30ms/step - loss: 1.1515 - accu
Epoch 3/20
1563/1563 [==============================] - 48s 31ms/step - loss: 1.0330 - accu
Epoch 4/20
1563/1563 [==============================] - 47s 30ms/step - loss: 0.9625 - accu
Epoch 5/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.9020 - accu
Epoch 6/20
1563/1563 [==============================] - 47s 30ms/step - loss: 0.8502 - accu
```

```
Epoch 7/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.8085 - accu
Epoch 8/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.7746 - accu
Epoch 9/20
1563/1563 [==============================] - 47s 30ms/step - loss: 0.7406 - accu
Epoch 10/20
1563/1563 [==============================] - 47s 30ms/step - loss: 0.7160 - accu
Epoch 11/20
1563/1563 [==============================] - 46s 29ms/step - loss: 0.6902 - accu
Epoch 12/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.6613 - accu
Epoch 13/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.6382 - accu
Epoch 14/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.6143 - accu
Epoch 15/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.5939 - accu
Epoch 16/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.5758 - accu
Epoch 17/20
1563/1563 [==============================] - 53s 34ms/step - loss: 0.5533 - accu
Epoch 18/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.5361 - accu
Epoch 19/20
1563/1563 [==============================] - 47s 30ms/step - loss: 0.5208 - accu
Epoch 20/20
1563/1563 [==============================] - 46s 30ms/step - loss: 0.5003 - accu
<keras.callbacks.History at 0x7faff9dcc590>
```

```
y_pred=cnn.predict(x_test)
```

```
cnn.evaluate(x_test,y_test)
```

```
313/313 [==============================] - 3s 9ms/step - loss: 1.1198 - accuracy
[1.119814157485962, 0.6704000234603882]
```

```
y_test=y_test.reshape(-1)
y_pred=cnn.predict(x_test)
```

```
y_classes=[np.argmax(element) for element in y_pred]
print('Classification report: \n',classification_report(y_test,y_classes))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.72      0.69      0.70      1000
```

```
              1       0.77      0.80      0.79      1000
              2       0.53      0.59      0.56      1000
              3       0.52      0.39      0.45      1000
              4       0.59      0.64      0.61      1000
              5       0.64      0.54      0.58      1000
              6       0.66      0.83      0.74      1000
              7       0.75      0.70      0.72      1000
              8       0.72      0.82      0.77      1000
              9       0.79      0.71      0.75      1000

       accuracy                           0.67     10000
      macro avg       0.67      0.67      0.67     10000
   weighted avg       0.67      0.67      0.67     10000
```

RESULT:

Program is executed successfully and output is obtained.