*Practical - 8 *

B: Image Data Neural Network implementation on mnist dataset (Image as an input)

Task 1: Creating First Artificial Neural Network (ANN) using Keras and Tensor flow.

Dataset: MNIST

Task 2: Improve the performance of Artificial Neural Network.

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

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

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

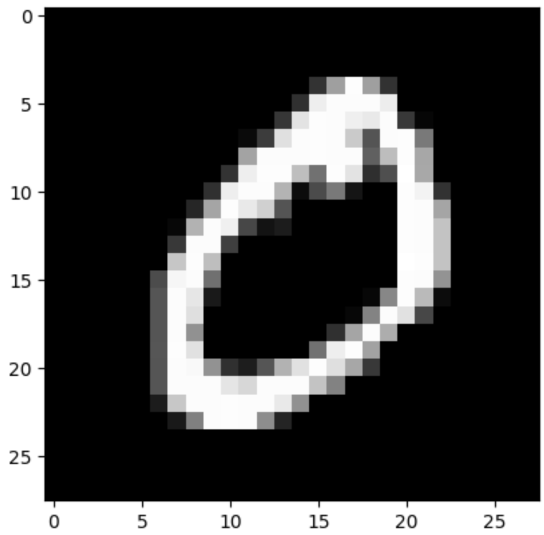
```
x_train
```

```
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, 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]]], dtype=uint8)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

```
import matplotlib.pyplot as plt
plt.imshow(x_train[1,:,:].reshape(28,28),cmap='gray')
plt.show()
```



```
y_train[1]
```

```
     0
```

```
#converting labels in hot vector
from keras.utils import to_categorical
y_train_encoded =to_categorical(y_train)
y_test_encoded= to_categorical(y_test)
```

```
y_train_encoded[1]
```

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

```
#Normalize the images.
x_train_norm =(x_train/255)-0.5
x_test_norm =(x_test/255)-0.5
```

```
print(x_train_norm.shape)
print(x_train[1,1,1],x_train_norm[1,1,1])
```

```
     (60000, 28, 28)
     0 -0.5
```

```
#Flatten the images.
x_train_images=x_train_norm.reshape((-1,784))
x_test_images = x_test_norm.reshape((-1,784))
```

```
print(x_train_images.shape)
print(y_train_encoded.shape)
print(x_test_images.shape)
print(y_test_encoded.shape)
```

```
     (60000, 784)
     (60000, 10)
     (10000, 784)
     (10000, 10)
```

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

model=Sequential()
model.add(Dense(64, input_dim=784,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax'))
```

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

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

```
  dense_1 (Dense)             (None, 32)                2080      
                                                                 
  dense_2 (Dense)             (None, 10)                330       
                                                                 
=================================================================
Total params: 52650 (205.66 KB)
Trainable params: 52650 (205.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Training a model on Train data and at the end it will update the weights.
model.fit(x_train_images,y_train_encoded,epochs=20,batch_size=16)
```

```
Epoch 1/20
3750/3750 [==============================] - 8s 2ms/step - loss: 0.3498 - accuracy: 0.8938
Epoch 2/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.1855 - accuracy: 0.9429
Epoch 3/20
3750/3750 [==============================] - 8s 2ms/step - loss: 0.1451 - accuracy: 0.9557
Epoch 4/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.1236 - accuracy: 0.9617
Epoch 5/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.1090 - accuracy: 0.9660
Epoch 6/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.1004 - accuracy: 0.9693
Epoch 7/20
3750/3750 [==============================] - 8s 2ms/step - loss: 0.0892 - accuracy: 0.9719
Epoch 8/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.0842 - accuracy: 0.9735
Epoch 9/20
3750/3750 [==============================] - 6s 2ms/step - loss: 0.0773 - accuracy: 0.9750
Epoch 10/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.0729 - accuracy: 0.9768
Epoch 11/20
3750/3750 [==============================] - 6s 2ms/step - loss: 0.0704 - accuracy: 0.9772
Epoch 12/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.0653 - accuracy: 0.9787
Epoch 13/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.0629 - accuracy: 0.9793
Epoch 14/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.0587 - accuracy: 0.9809
Epoch 15/20
3750/3750 [==============================] - 8s 2ms/step - loss: 0.0568 - accuracy: 0.9814
Epoch 16/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.0561 - accuracy: 0.9820
Epoch 17/20
3750/3750 [==============================] - 8s 2ms/step - loss: 0.0532 - accuracy: 0.9827
Epoch 18/20
3750/3750 [==============================] - 6s 2ms/step - loss: 0.0493 - accuracy: 0.9833
Epoch 19/20
3750/3750 [==============================] - 8s 2ms/step - loss: 0.0501 - accuracy: 0.9833
Epoch 20/20
3750/3750 [==============================] - 7s 2ms/step - loss: 0.0473 - accuracy: 0.9844
```

```
#Evaluate the trained neural network model's performance
#evaluate the model
scores = model.evaluate(x_test_images,y_test_encoded)
print("\nAccuracy: %.2f%%" % (scores[1]*100))
```

```
# save the model's saved weights
model.save_weights("mnistmodel.h5")

#build the model again and load the weights.

# load the model's saved weights.
model.load_weights("mnistmodel.h5")
```

Complete