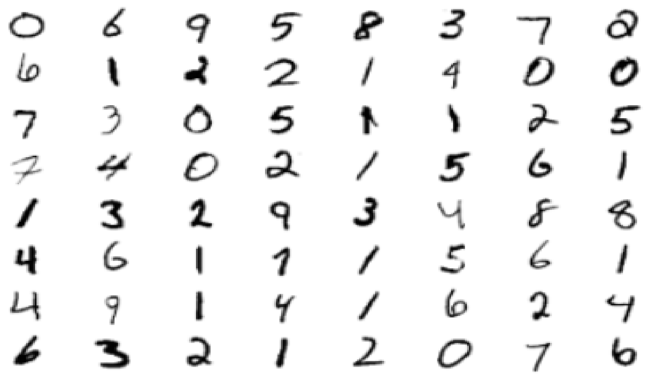
EXPERIMENT- 3

AIM: Write a program to implement Artificial Neural Network for MNIST dataset.

CODE and OUTPUT:

```
import matp
import seab
import nump
from sklear

import keras
from keras.da
from keras.la
from keras.mo
from matplotl
from random i

# Preparing t
# Setup train
(x_train, y_t

# Making a co
x_train_draw
print("X_Tra
print("y_tra

print("X_Tra
print("y_tra

image_size
x_train = x
x_test = x_

print("Afte
print("X_Tr
print("x_te

# Convert c
num_classes
y_train = k
y_test = ke

After resha
X_Train Sh
x_test Sh

print(y_t
print(y_t

(60000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
for i in range(64):
    ax = plt.subplot(8, 8, i+1)
    ax.axis('off')
    plt.imshow(x_train_drawing[randint(0, x_train.shape[0])], cmap='Greys')
```

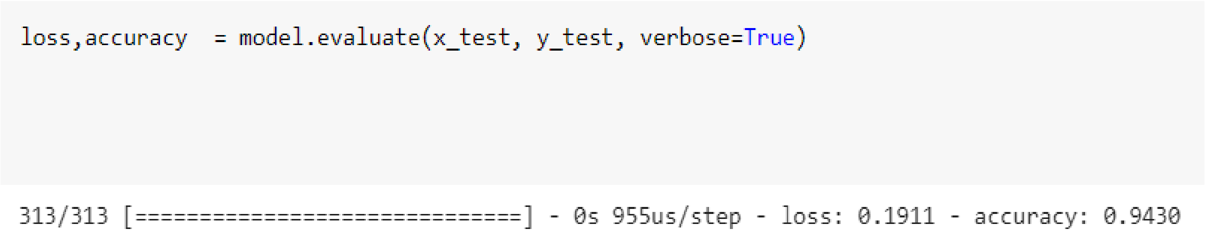


```
model = Sequential()

# The input layer requires the special input_shape parameter which should match
# the shape of our training data.
model.add(Dense(units=32, activation='sigmoid', input_shape=(image_size,)))
model.add(Dense(units=num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
dense (Dense)               (None, 32)                25120

_________________________________________________________________
dense_1 (Dense)             (None, 10)                330
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

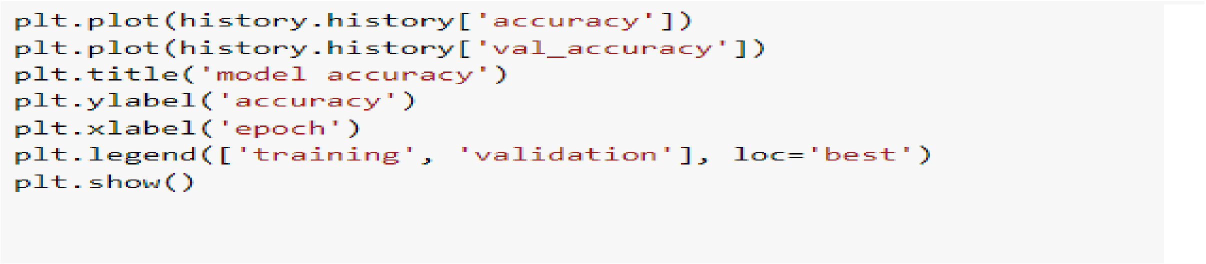
```
model.compile(optimizer="sgd", loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=128, epochs=100, verbose=True, validation_split=.1)
```

```
Epoch 1/100
422/422 [==============================] - 1s 3ms/step - loss: 0.1584 - accuracy: 0.9538 - val_loss: 0.1714 - val_accuracy: 0.9517
Epoch 2/100
422/422 [==============================] - 1s 2ms/step - loss: 0.1527 - accuracy: 0.9551 - val_loss: 0.1730 - val_accuracy: 0.9508
Epoch 3/100
422/422 [==============================] - 1s 2ms/step - loss: 0.1607 - accuracy: 0.9525 - val_loss: 0.1670 - val_accuracy: 0.9517

Epoch 97/100
422/422 [==============================] - 1s 2ms/step - loss: 0.1289 - accuracy: 0.9612 - val_loss: 0.1630 - val_accuracy: 0.9550
Epoch 98/100
422/422 [==============================] - 1s 2ms/step - loss: 0.1299 - accuracy: 0.9604 - val_loss: 0.1583 - val_accuracy: 0.9532
Epoch 99/100
422/422 [==============================] - 1s 2ms/step - loss: 0.1292 - accuracy: 0.9606 - val_loss: 0.1627 - val_accuracy: 0.9495
Epoch 100/100
422/422 [==============================] - 1s 2ms/step - loss: 0.1284 - accuracy: 0.9609 - val_loss: 0.1578 - val_accuracy: 0.9545
```

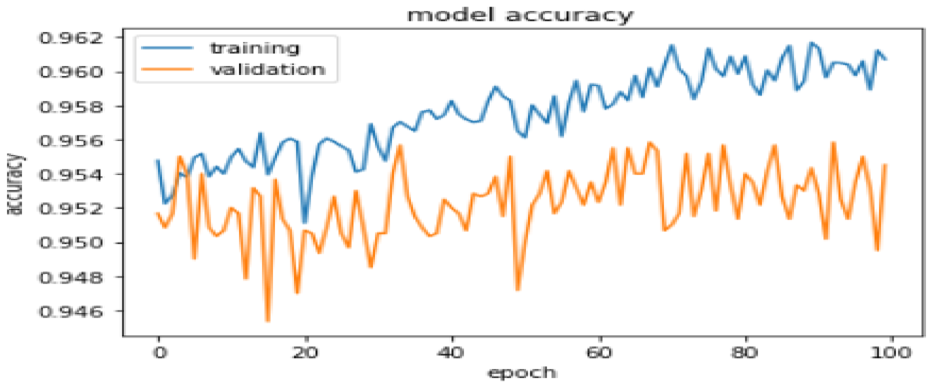


```
loss,accuracy  = model.evaluate(x_test, y_test, verbose=True)
```

```
313/313 [==============================] - 0s 955us/step - loss: 0.1911 - accuracy: 0.9430
```



```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```


model accuracy
training
validation
accuracy
epoch