Ex No: 4 **HANDWRITTEN DIGITS RECOGNITION WITH MNIST**

AIM:

To build a handwritten digit's recognition with MNIST dataset.

PROCEDURE:

1. Download and load the MNIST dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

PROGRAM:

```
import numpy as np
from tensorflow.keras.models import load_model
from tkinter import *
import tkinter as tk
#import win32gui
from PIL import ImageGrab, Image


from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import backend as K


(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape, y_train.shape)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

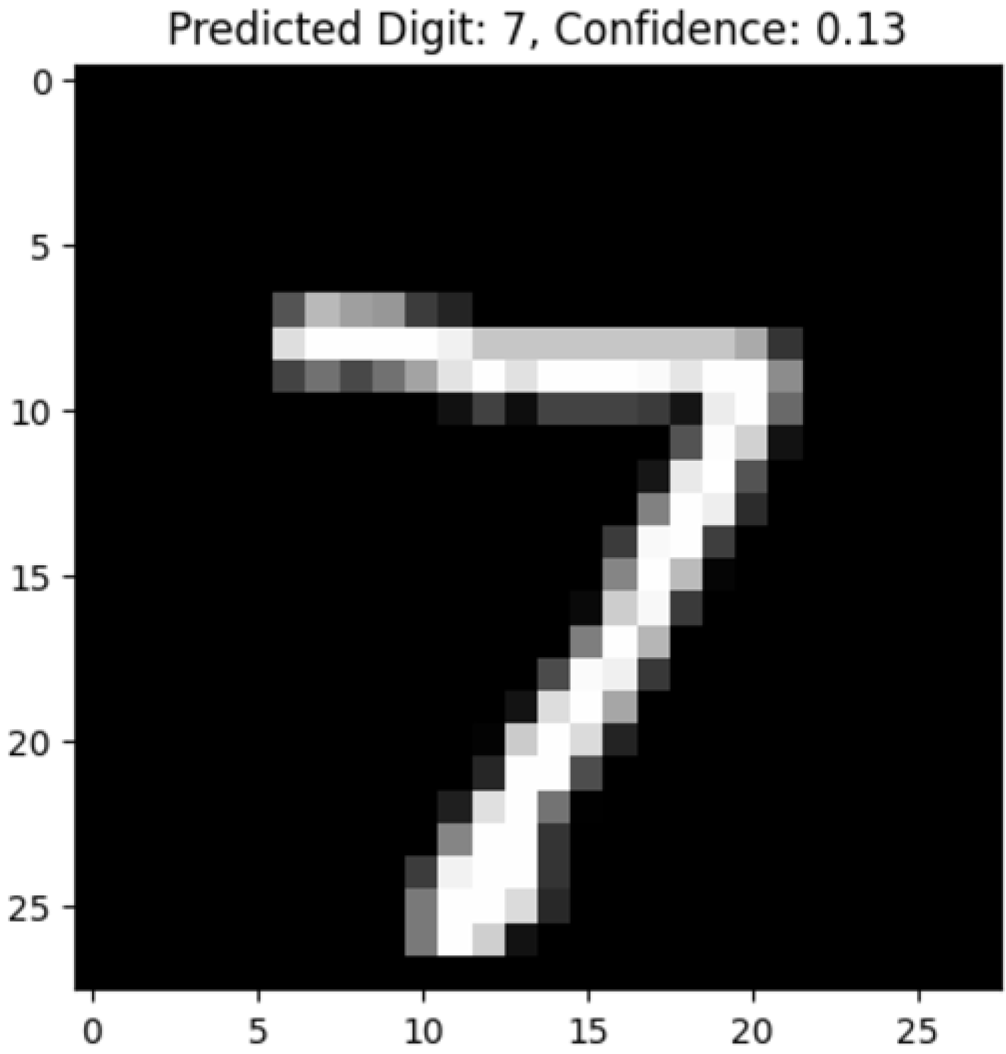
```
input_shape = (28, 28, 1)
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
batch_size = 128
num_classes = 10
epochs = 15
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))AA
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])
hist = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
print("The model has successfully trained")
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
model.save('mnist.h5')
```

```
print("Saving the model as mnist.h5")
model = load_model('mnist.h5')
def predict_digit(img):
    #resize image to 28x28 pixels
    img = img.resize((28,28))
    #convert rgb to grayscale
    img = img.convert('L')
    img = np.array(img)
    img = img.reshape(1,28,28,1)
    img = img/255.0
    img = 1 - img
    #predicting
    res = model.predict([img])[0]
    return np.argmax(res), max(res)
import matplotlib.pyplot as plt

# Use an image from the MNIST test dataset
test_image_array = x_test[0]  # Change the index to use different images from the test set
test_image_pil = Image.fromarray((test_image_array.squeeze() * 255).astype(np.uint8))

# Predict the digit in the image
predicted_digit, confidence = predict_digit(test_image_pil)

# Print the results
print(f"Predicted Digit: {predicted_digit}")
print(f"Confidence: {confidence:.2f}")

# Show the test image
plt.imshow(test_image_array.squeeze(), cmap='gray')
plt.title(f"Predicted Digit: {predicted_digit}, Confidence: {confidence:.2f}")
plt.show()
```

OUTPUT:

```
Epoch 1/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 60s 124ms/step - accuracy: 0.0946 - loss: 2.3087 - val_accuracy: 0.1088 - val_loss: 2.2966
Epoch 2/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 76s 111ms/step - accuracy: 0.1050 - loss: 2.2967 - val_accuracy: 0.1152 - val_loss: 2.2854
Epoch 3/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 83s 112ms/step - accuracy: 0.1209 - loss: 2.2863 - val_accuracy: 0.1149 - val_loss: 2.2749
Epoch 4/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 83s 114ms/step - accuracy: 0.1315 - loss: 2.2772 - val_accuracy: 0.1229 - val_loss: 2.2648
Epoch 5/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 53s 113ms/step - accuracy: 0.1469 - loss: 2.2682 - val_accuracy: 0.1488 - val_loss: 2.2542
Epoch 6/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 82s 115ms/step - accuracy: 0.1582 - loss: 2.2590 - val_accuracy: 0.1873 - val_loss: 2.2430
Epoch 7/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 80s 111ms/step - accuracy: 0.1770 - loss: 2.2493 - val_accuracy: 0.2393 - val_loss: 2.2305
Epoch 8/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 80s 108ms/step - accuracy: 0.1970 - loss: 2.2385 - val_accuracy: 0.3082 - val_loss: 2.2167
Epoch 9/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 83s 111ms/step - accuracy: 0.2099 - loss: 2.2258 - val_accuracy: 0.3684 - val_loss: 2.2012
Epoch 10/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 82s 111ms/step - accuracy: 0.2241 - loss: 2.2128 - val_accuracy: 0.4172 - val_loss: 2.1840
Epoch 11/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 83s 113ms/step - accuracy: 0.2417 - loss: 2.1981 - val_accuracy: 0.4587 - val_loss: 2.1646
Epoch 12/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 80s 108ms/step - accuracy: 0.2584 - loss: 2.1834 - val_accuracy: 0.5016 - val_loss: 2.1431
Epoch 13/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 85s 116ms/step - accuracy: 0.2745 - loss: 2.1648 - val_accuracy: 0.5442 - val_loss: 2.1193
Epoch 14/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 80s 112ms/step - accuracy: 0.2872 - loss: 2.1457 - val_accuracy: 0.5815 - val_loss: 2.0932
Epoch 15/15
469/469 ━━━━━━━━━━━━━━━━━━━━ 81s 110ms/step - accuracy: 0.3053 - loss: 2.1227 - val_accuracy: 0.6110 - val_loss: 2.0644
The model has successfully trained
```

```
Predicted Digit: 7
Confidence: 0.13
```



RESULT:

Thus a handwritten digit’s recognition with MNIST dataset is built.