Task 8 Implement Simple Neural Network using Mnist digit Classification dataset

In [1]:
```
import tensorflow as tf
```

In [2]:
```
from tensorflow import keras
```

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [4]:
```
(X_train,y_train),(X_test,y_test)  = tf.keras.datasets.mnist.load_data("")
```

In [5]:
```
len(X_train)
```

Out[5]:
```
60000
```

In [6]:
```
len(X_test)
```

Out[6]:
```
10000
```

In [7]:
```
X_train[0].shape
```

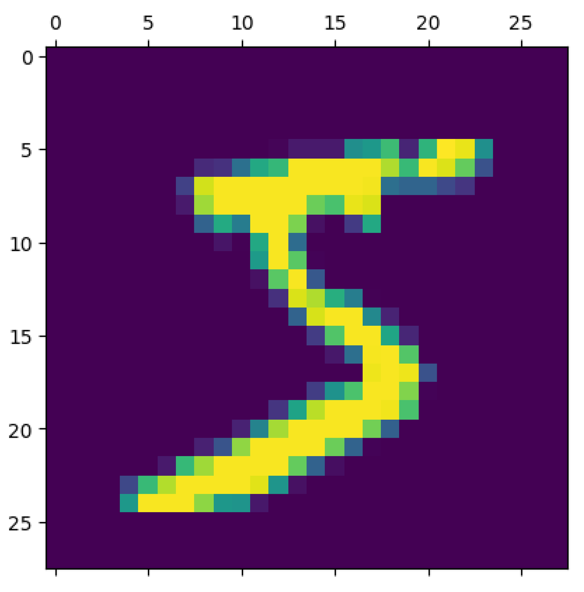
Out[7]:
```
(28, 28)
```

In [8]:
```
X_train[0]
```

Out[8]:
```
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,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   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, 139, 253,
        190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
        253,  70,   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,  35,
        241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  16,  93, 252, 253, 187,   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, 249, 253, 249,  64,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
        148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
        253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
        253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
        195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
         11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   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)
```

In [9]:
```
plt.matshow(X_train[0])
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x2624f6df090>
```



In [10]:
```
y_train[0]
```

Out[10]:
```
5
```

In [11]:
```
X_train=X_train/255
X_test = X_test/255
```

In [12]:
```
X_train_flattened = X_train.reshape(len(X_train),28*28)
X_test_flattened  = X_test.reshape(len(X_test),28*28)
```

In [13]:
```
X_train_flattened.shape
```

Out[13]:
```
(60000, 784)
```

In [14]:
```
model = keras.Sequential([
    keras.layers.Dense(100,input_shape=(784,),activation="sigmoid"),
    keras.layers.Dense(10,activation="sigmoid"),
])

model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

model.fit(X_train_flattened,y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 7s 3ms/step - loss: 0.4202 - accuracy: 0.8942
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1993 - accuracy: 0.9434
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1485 - accuracy: 0.9572
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1175 - accuracy: 0.9664
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0969 - accuracy: 0.9722
```

Out[14]:
```
<keras.src.callbacks.History at 0x2624ffa09d0>
```

In [15]:
```
model.evaluate(X_test_flattened,y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1034 - accuracy: 0.9695
```

Out[15]:
```
[0.10335471481084824, 0.9695000052452087]
```

In [16]:
```
y_predicted = model.predict(X_test_flattened)
y_predicted[0]
```

```
313/313 [==============================] - 1s 2ms/step
```

Out[16]:
```
array([6.5776512e-02, 1.4099288e-03, 2.9291666e-01, 7.8086680e-01,
       5.8065815e-04, 2.9643521e-02, 9.4134257e-06, 9.9954641e-01,
       1.1329570e-02, 6.9856659e-02], dtype=float32)
```

In [17]:
```
np.argmax(y_predicted[0])
```

Out[17]:
```
7
```

In [18]:
```
y_pred_label = [np.argmax(i) for i in y_predicted]
```

In [19]:
```
cm = tf.math.confusion_matrix(labels=y_test[:1000],predictions=y_pred_label[:1000])
```

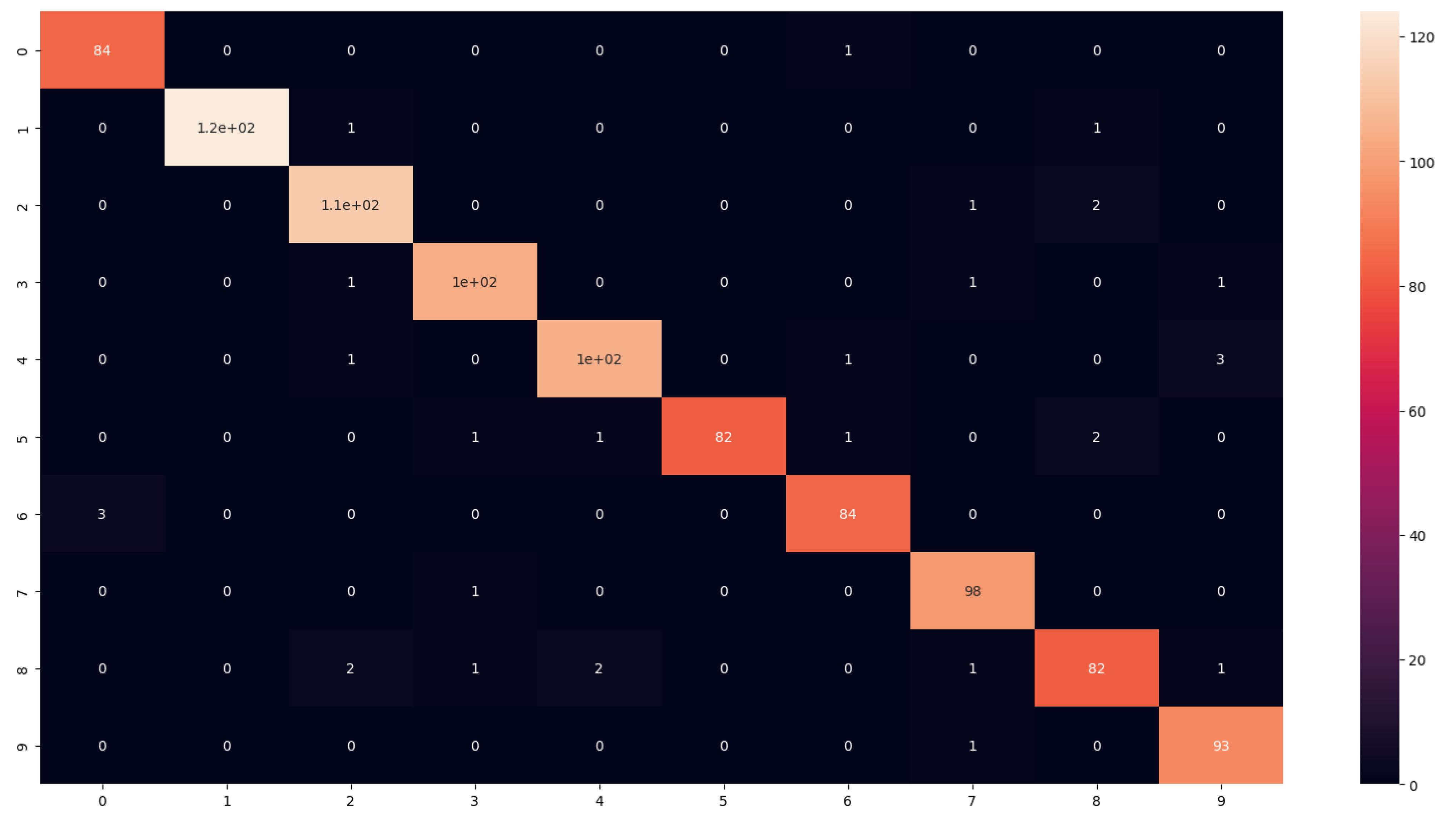
In [20]:
```
cm
```

Out[20]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 84,   0,   0,   0,   0,   0,   1,   0,   0,   0],
       [  0, 124,   1,   0,   0,   0,   0,   0,   1,   0],
       [  0,   0, 113,   0,   0,   0,   0,   1,   2,   0],
       [  0,   0,   1, 104,   0,   0,   0,   1,   0,   1],
       [  0,   0,   1,   0, 105,   0,   1,   0,   0,   3],
       [  0,   0,   0,   1,   1,  82,   1,   0,   2,   0],
       [  3,   0,   0,   0,   0,   0,  84,   0,   0,   0],
       [  0,   0,   0,   1,   0,   0,   0,  98,   0,   0],
       [  0,   0,   2,   1,   2,   0,   0,   1,  82,   1],
       [  0,   0,   0,   0,   0,   0,   0,   1,   0,  93]])>
```

In [21]:
```
import seaborn as sns
```

In [22]:
```
plt.figure(figsize= (20,10))
sns.heatmap(cm,annot=True)
```

Out[22]:
```
<Axes: >
```



In [23]:
```
y_pred_label[:5]
```

Out[23]:
```
[7, 2, 1, 0, 4]
```

In [24]:
```
import pickle
pickle.dump(model, open('./mnist.pkl', 'wb'))
```

Documentation

1. Introduction

1.1 Project Overview : This document provides documentation for the MNIST Digit Classification project. The objective is to build a machine learning model capable of accurately classifying handwritten digits (0-9) from the MNIST dataset.

1.2 Project Objectives : Develop a robust digit classification model using machine learning techniques. Evaluate the model's performance on the MNIST dataset. Provide insights into the model's predictions and potential use cases. 2. Dataset : 2.1 Data Source The MNIST dataset is a collection of 28x28 pixel grayscale images of handwritten digits. It consists of 60,000 training images and 10,000 testing images.

2.2 Data Exploration : Explore the structure of the dataset. Visualize sample images and their corresponding labels. Check for class imbalances and preprocess the data if necessary. 3. Machine Learning Model :
3.1 Model Selection : For this project, a Convolutional Neural Network (CNN) is chosen due to its effectiveness in image classification tasks.

3.2 Model Architecture : Design a CNN architecture with convolutional layers, pooling layers, and fully connected layers. Specify activation functions, dropout, and other hyperparameters.

3.3 Model Training : Split the dataset into training and testing sets. Train the CNN model using the training data. Fine-tune hyperparameters for optimal performance. Monitor training/validation accuracy and loss.

3.4 Model Evaluation : Evaluate the model's performance using metrics such as accuracy, precision, recall, and confusion matrix on the test set.

4. Deployment :

Deploy the trained model for inference, either as a standalone application or through a web service.

5. Conclusion :

Summarize the key findings of the project, including the model's accuracy, challenges faced, and potential improvements.

6. Future Work :

Discuss potential future enhancements, such as model optimization, exploring different architectures, or deploying on edge devices.

7. References

Include references to relevant papers, libraries, and resources used during the project.

In [ ]:

```

```