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)

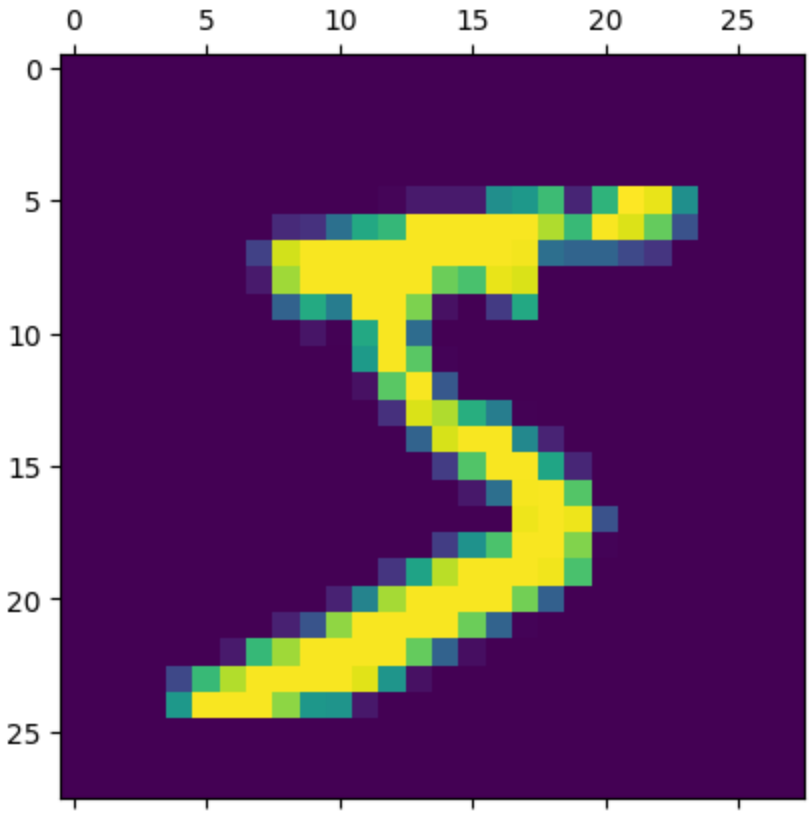
In [8]:
```
X_train[0]
```

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 - acc
uracy: 0.8942
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1993 - acc
uracy: 0.9434
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1485 - acc
uracy: 0.9572
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1175 - acc
uracy: 0.9664
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0969 - acc
uracy: 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 - accur
acy: 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[:1
```

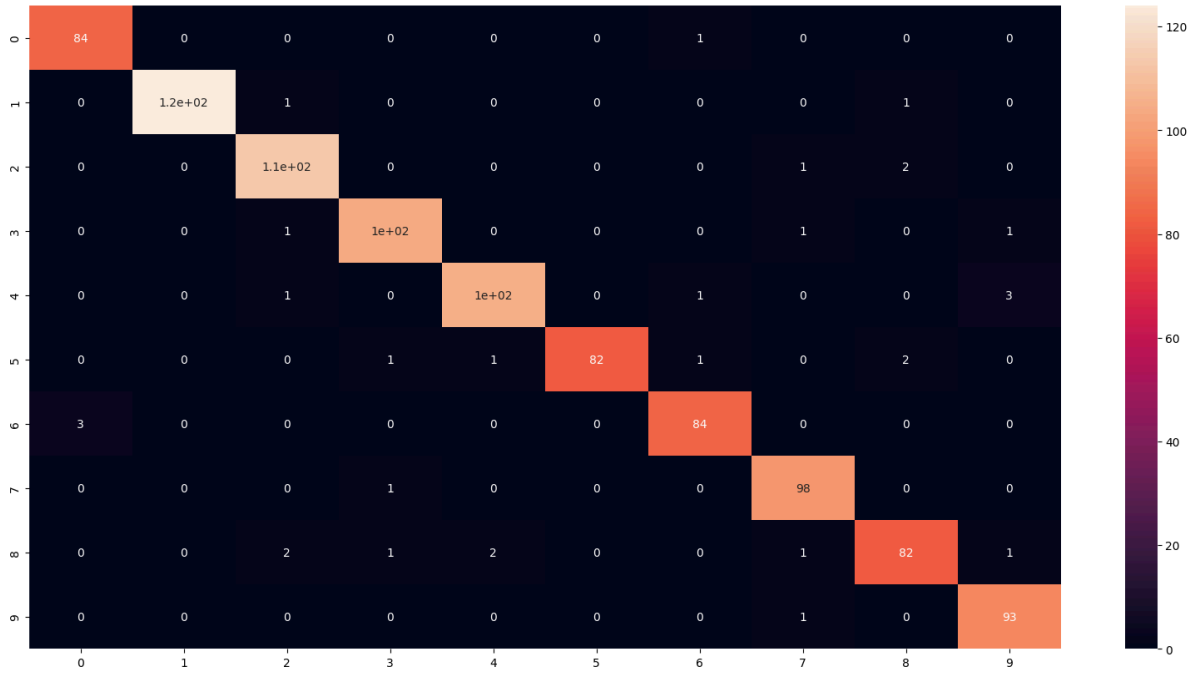
In [20]:
```
cm
```

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'))
```

Documenting the purpose of the task of MNIST handwritten digit classification

The MNIST dataset used, the neural network architecture, and the evaluation results follows a similar structure to the general guidelines mentioned earlier. Here's a specific guide for documenting these aspects for an MNIST classification project:

```
Title and Introduction:

Title: "MNIST Handwritten Digit Classification"
Introduction: Provide an overview of the task, such as recognizing
handwritten digits from 0 to 9. Explain the importance of this task, which is
often used as a benchmark for machine learning models.

Purpose of the Task:


Explain the purpose of the task, which is to develop a model that can
accurately classify handwritten digits. Mention that this task is a
fundamental building block in computer vision and machine learning.
```

```
MNIST Dataset:

Describe the MNIST dataset:
Source: The MNIST dataset is a well-known dataset of handwritten digits,
readily available through libraries like TensorFlow or PyTorch.
Size: Mention that it consists of 60,000 training images and 10,000 test
images, each with 28x28 pixels.
Preprocessing: State if any preprocessing was applied, such as normalization
of pixel values (0-255) to (0-1).
Neural Network Architecture:

Describe the neural network architecture used for MNIST classification. For
simplicity, you can use a basic feedforward neural network:
Type of model: Feedforward neural network.
Layers: Specify the number of layers, units/neurons in each layer, and
activation functions (e.g., ReLU, softmax for the output layer).
Hyperparameters: Include learning rate, batch size, and the number of epochs.
Any specific features or techniques unique to your model.
Model Training:

Explain how the model was trained:
Training process: Describe how the model was trained, including the loss
function (cross-entropy is common) and the optimizer (e.g., Adam, SGD).
Data augmentation: Mention if any data augmentation techniques were used
(typically not required for MNIST).
Overfitting prevention: Explain how overfitting was addressed, e.g., by using
dropout layers.
Metrics: State which evaluation metrics you used during training, such as
accuracy.
Evaluation Results:

Present the evaluation results for your MNIST classification model:
Performance: Report accuracy, possibly along with confusion matrices or
class-wise metrics.
Validation and test results: Separate the model's performance on the
validation and test datasets.
Visualization: Optionally, include example predictions, visualization of
filters or feature maps, and other relevant visualizations.
Discussion:

Interpret the results:
Discuss the model's performance and how it relates to the complexity of the
task.
Mention any challenges or limitations specific to MNIST classification.
Provide insights into potential model improvements or alternative
architectures.
Conclusion:
```

In [ ]: