```python
In [4]: import tensorflow as tf
        from tensorflow import keras
        import matplotlib.pyplot as plt
        %matplotlib inline
        import numpy as np
        import pandas as pd
```

```python
In [5]: (X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
```

```python
In [6]: len(X_train)
```

Out[6]: 60000

```python
In [7]: len(X_test)
```

Out[7]: 10000

```python
In [8]: X_train[0].shape
```

Out[8]: (28, 28)

each element in the array is an 28x28 pixel image

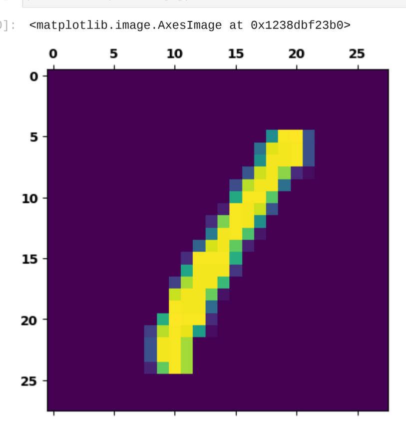
```python
In [9]: X_train[0]
```

```
Out[9]: 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)
```

let's see it more clearly with matplot

```python
In [10]: plt.matshow(X_train[3])
```

Out[10]: <matplotlib.image.AxesImage at 0x1238dbf23b0>



```python
In [11]: y_train[3]
```

Out[11]: 1

```python
In [12]: y_train[:5]
```

Out[12]: array([5, 0, 4, 1, 9], dtype=uint8)

so the X array contains matricial representation of the hand-written number and the Y array contains the tags of what number it is. I need to reshape the tridimensional array into an bidimensional one

```python
In [13]: X_train.shape
```

Out[13]: (60000, 28, 28)

I need to normalize the data [0 - 1]

```python
In [14]: X_train = X_train /255
         X_test = X_test /255
```

the first dimension is the number of examples it has, and then each individual image is 28x28 pixels long

```python
In [15]: X_train_flattened = X_train.reshape(len(X_train), 28*28)
         X_test_flattened = X_test.reshape(len(X_test), 28*28)
```

```python
In [16]: X_train_flattened
```

```
Out[16]: 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.]])
```

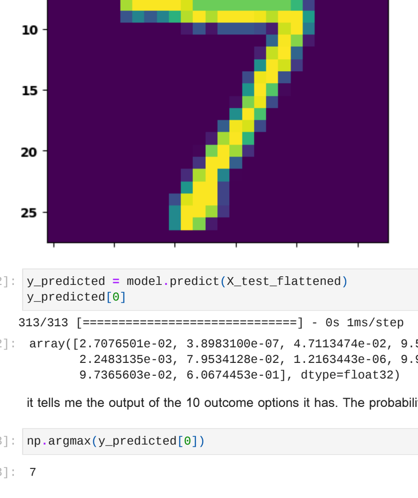
```python
In [17]: print(X_train_flattened.shape)
         print(X_test_flattened.shape)
```

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

```python
In [18]: X_train_flattened[0]
```

```
Out[18]: 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.        , 0.01176471, 0.07058824, 0.07058824,
       0.07058824, 0.49411765, 0.53333333, 0.68627451, 0.10196078,
       0.65098039, 1.        , 0.96862745, 0.49803922, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.11764706, 0.14117647, 0.36862745, 0.60392157,
       0.66666667, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.88235294, 0.6745098 , 0.99215686, 0.94901961,
       0.76470588, 0.25098039, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.19215686, 0.93333333,
       0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.98431373, 0.36470588,
       0.32156863, 0.32156863, 0.21960784, 0.15294118, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.07058824, 0.85882353, 0.99215686, 0.99215686,
       0.99215686, 0.99215686, 0.99215686, 0.77647059, 0.71372549,
       0.96862745, 0.94509804, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.31372549, 0.61176471, 0.41960784, 0.99215686, 0.99215686,
       0.80392157, 0.04313725, 0.        , 0.16862745, 0.60392157,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.05490196,
       0.00392157, 0.60392157, 0.99215686, 0.35294118, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.54509804,
       0.99215686, 0.74509804, 0.00784314, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.04313725, 0.74509804, 0.99215686,
       0.2745098 , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.1372549 , 0.94509804, 0.88235294, 0.62745098,
       0.42352941, 0.00392157, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.31764706, 0.94117647, 0.99215686, 0.99215686, 0.46666667,
       0.09803922, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.17647059,
       0.72941176, 0.99215686, 0.99215686, 0.58823529, 0.10588235,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.0627451 , 0.36470588,
       0.98823529, 0.99215686, 0.73333333, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.97647059, 0.99215686,
       0.97647059, 0.25098039, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.18039216, 0.50980392,
       0.71764706, 0.99215686, 0.99215686, 0.81176471, 0.00784314,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.15294118,
       0.58039216, 0.89803922, 0.99215686, 0.99215686, 0.99215686,
       0.98039216, 0.71372549, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       [illegible]
```

[illegible]



```python
In [22]: y_predicted = model.predict(X_test_flattened)
         y_predicted[0]
```

```
313/313 [==============================] - 0s 1ms/step
Out[22]: array([2.7076501e-02, 3.8983100e-07, 4.7113474e-02, 9.5962936e-01,
       2.2483135e-03, 7.9534128e-02, 1.2163443e-06, 9.9977410e-01,
       9.7365603e-02, 6.0674453e-01], dtype=float32)
```

it tells me the output of the 10 outcome options it has. The probabilities for every output

```python
In [23]: np.argmax(y_predicted[0])
```

Out[23]: 7

The prediction looks to work very well, we are going to use a confusion matrix to check the overall performance

```python
In [24]: y_predicted_labels = [np.argmax(i) for i in y_predicted]
         y_predicted_labels[:5]
```

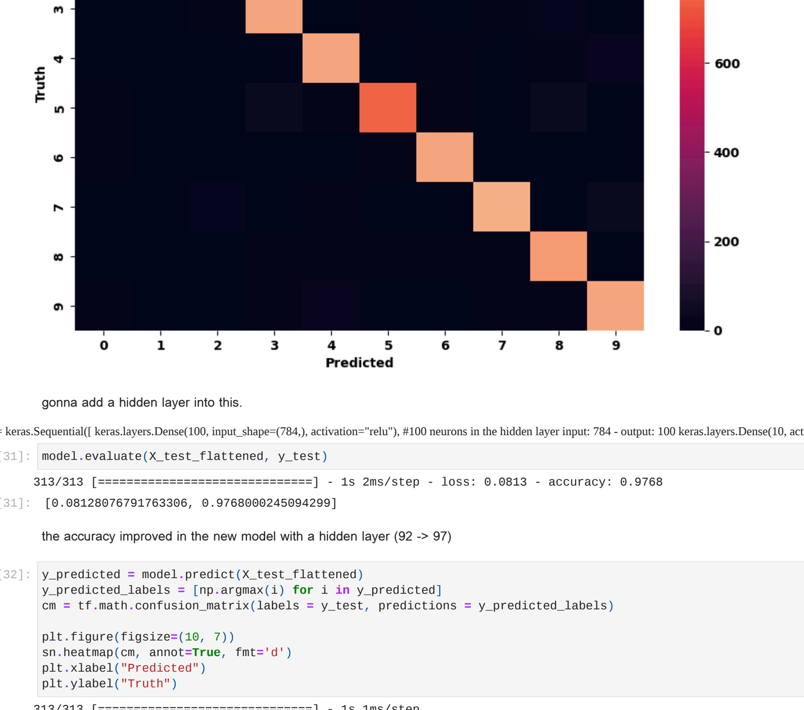
Out[24]: [7, 2, 1, 0, 4]

```python
In [25]: cm = tf.math.confusion_matrix(labels = y_test, predictions = y_predicted_labels)

         cm
```

```
Out[25]: <tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 964,    0,    1,    2,    0,    5,    5,    2,    1,    0],
       [   0, 1110,    3,    2,    0,    1,    4,    2,   13,    0],
       [   8,    9,  926,   16,    8,    2,   12,    9,   39,    3],
       [   4,    0,   20,  923,    1,   20,    2,    9,   24,    7],
       [   1,    1,    4,    1,  920,    0,   10,    3,   10,   32],
       [  11,    3,    5,   38,   11,  755,   14,    6,   42,    7],
       [  14,    3,    6,    1,    7,   10,  912,    2,    3,    0],
       [   1,    5,   26,    6,   10,    0,    0,  942,    3,   35],
       [   7,    7,    6,   16,    9,   18,    9,   10,  886,    6],
       [  11,    7,    1,   11,   32,    5,    0,   12,   10,  920]])>
```

```python
In [28]: import seaborn as sn

         plt.figure(figsize=(10, 7))
         sn.heatmap(cm, annot=True, fmt='d')
         plt.xlabel("Predicted")
         plt.ylabel("Truth")
```

Out[28]: Text(95.72222222222221, 0.5, 'Truth')



gonna add a hidden layer into this.

model = keras.Sequential([keras.layers.Dense(100, input_shape=(784,), activation="relu"), #100 neurons in the hidden layer input: 784 - output: 100 keras.layers.Dense(10, activation="sigmoid") #input: 100 - output: 10]) model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"]) model.fit(X_train_flattened, y_train, epochs=5)

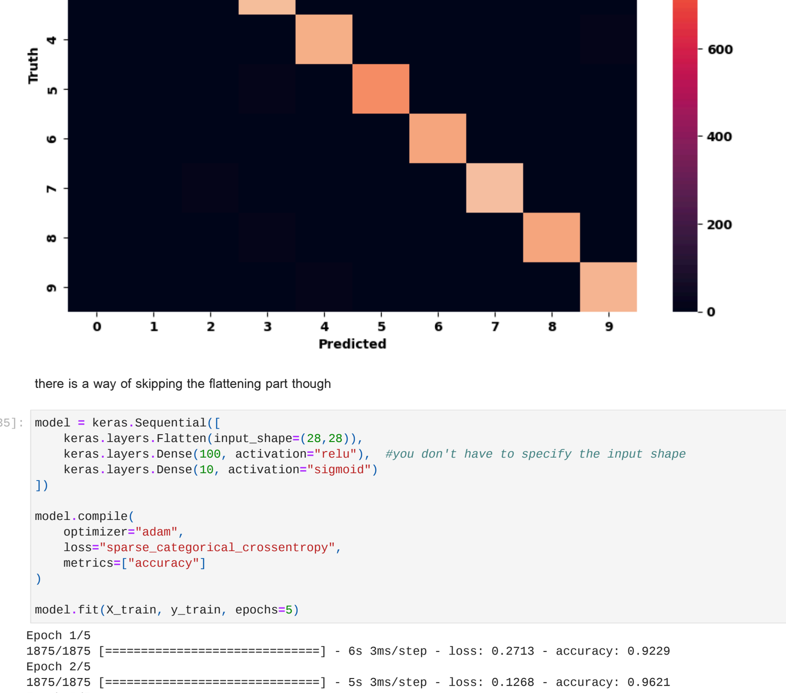
```python
In [31]: model.evaluate(X_test_flattened, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0813 - accuracy: 0.9768
Out[31]: [0.08128076791763306, 0.9768000245094299]
```

the accuracy improved in the new model with a hidden layer (92 -> 97)

```python
In [32]: y_predicted = model.predict(X_test_flattened)
         y_predicted_labels = [np.argmax(i) for i in y_predicted]
         cm = tf.math.confusion_matrix(labels = y_test, predictions = y_predicted_labels)

         plt.figure(figsize=(10, 7))
         sn.heatmap(cm, annot=True, fmt='d')
         plt.xlabel("Predicted")
         plt.ylabel("Truth")
```

```
313/313 [==============================] - 1s 1ms/step
Out[32]: Text(95.72222222222221, 0.5, 'Truth')
```



there is a way of skipping the flattening part though

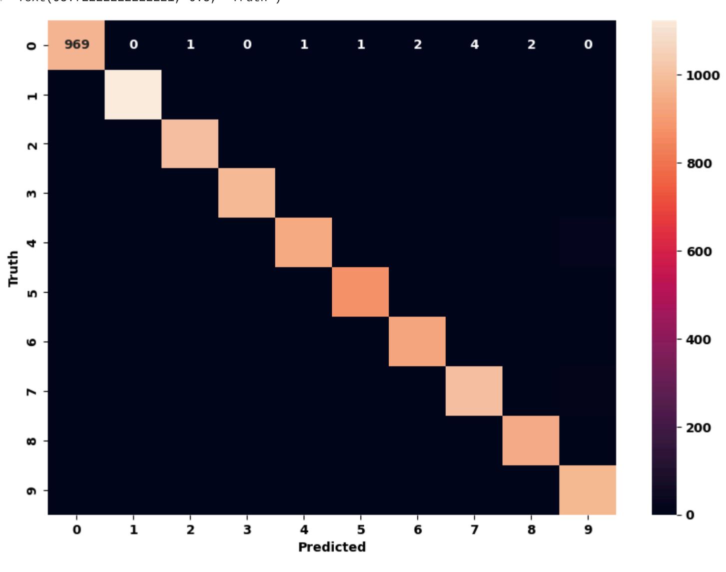
```python
In [35]: model = keras.Sequential([
             keras.layers.Flatten(input_shape=(28,28)),
             keras.layers.Dense(100, activation="relu"),  #you don't have to specify the input shape
             keras.layers.Dense(10, activation="sigmoid")
         ])

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

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

```
Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2713 - accuracy: 0.9229
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1268 - accuracy: 0.9621
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0888 - accuracy: 0.9737
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0663 - accuracy: 0.9805
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0531 - accuracy: 0.9839
Out[35]: <keras.callbacks.History at 0x123bd4bebc0>
```

```python
In [36]: y_predicted = model.predict(X_test)
         y_predicted_labels = [np.argmax(i) for i in y_predicted]
         cm = tf.math.confusion_matrix(labels = y_test, predictions = y_predicted_labels)

         plt.figure(figsize=(10, 7))
         sn.heatmap(cm, annot=True, fmt='d')
         plt.xlabel("Predicted")
         plt.ylabel("Truth")
```

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

Out[36]: Text(95.72222222222221, 0.5, 'Truth')



In [ ]: