Handwritten digits classification using neural network

In this notebook we will classify handwritten digits using a simple neural network which has only input and output layers. We will than add a hidden layer and see how the performance of the model improves

In [2]:
```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [3]:
```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11490434/11490434 [==============================] - 1s 0us/step
```

In [4]:
```
len(X_train)
```

Out[4]:
```
60000
```

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

Out[5]:
```
10000
```

In [6]:
```
X_train[0].shape
```

Out[6]:
```
(28, 28)
```

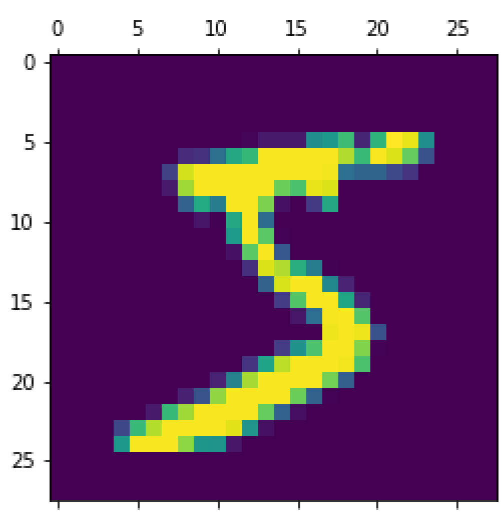
In [7]:
```
X_train[0]
```

Out[7]:
```
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 [8]:
```
plt.matshow(X_train[0])
```

Out[8]: <matplotlib.image.AxesImage at 0x20591f11700>



In [9]:
```
y_train[0]
```

Out[9]: 5

In [10]:
```
X_train = X_train / 255
X_test = X_test / 255
```

In [11]:
```
X_train[0]
```

Out[11]:
```
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.        ,
```

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]:
```
X_train_flattened[0]
```

```
       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.        ,
```

Very simple neural network with no hidden layers

In [15]:
```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), 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 [==============================] - 2s 680us/step - loss: 0.4709 - acc
uracy: 0.8758
Epoch 2/5
1875/1875 [==============================] - 1s 652us/step - loss: 0.3037 - acc
uracy: 0.9149
Epoch 3/5
1875/1875 [==============================] - 1s 649us/step - loss: 0.2827 - acc
uracy: 0.9216
Epoch 4/5
1875/1875 [==============================] - 1s 647us/step - loss: 0.2730 - acc
uracy: 0.9240
Epoch 5/5
1875/1875 [==============================] - 1s 637us/step - loss: 0.2670 - acc
uracy: 0.9255
```

Out[15]:
```
<keras.callbacks.History at 0x2059194ce80>
```

In [16]:
```
model.evaluate(X_test_flattened, y_test)
```

```
313/313 [==============================] - 0s 563us/step - loss: 0.2715 - accur
acy: 0.9245
```

Out[16]:
```
[0.2714834213256836, 0.9244999885559082]
```

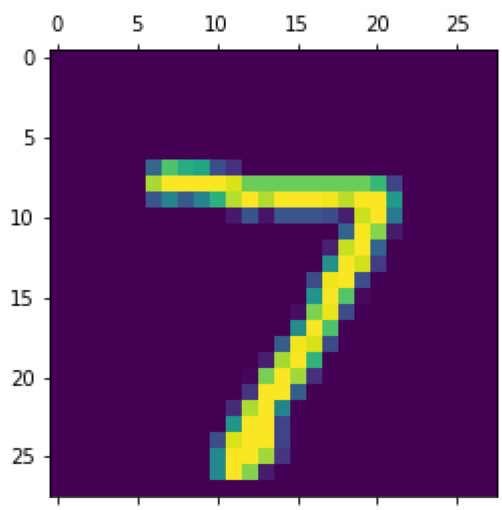
In [17]:
```
y_predicted = model.predict(X_test_flattened)
y_predicted[0]
```

```
313/313 [==============================] - 0s 470us/step
```

Out[17]:
```
array([3.68300751e-02, 3.70372277e-07, 9.22595412e-02, 9.49832320e-01,
       1.59104867e-03, 1.35646909e-01, 1.48944628e-06, 9.99841928e-01,
       1.00577824e-01, 6.78470373e-01], dtype=float32)
```

In [18]:
```
plt.matshow(X_test[0])
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x205c3a24ac0>
```



np.argmax finds a maximum element from an array and returns the index of it

In [19]:
```
np.argmax(y_predicted[0])
```

Out[19]:
```
7
```

In [20]:
```
y_predicted_labels = [np.argmax(i) for i in y_predicted]
```

In [21]:
```
y_predicted_labels[:5]
```

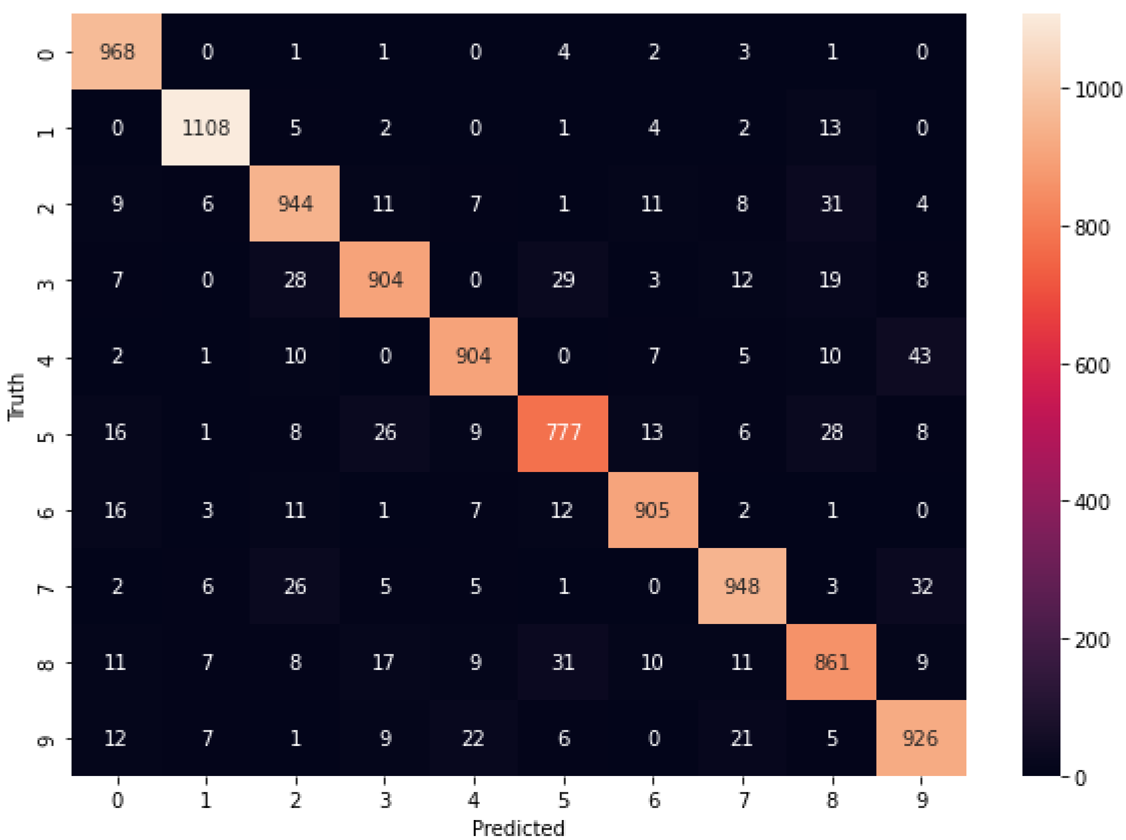
Out[21]:
```
[7, 2, 1, 0, 4]
```

In [22]:
```
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
cm
```

Out[22]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 968,    0,    1,    1,    0,    4,    2,    3,    1,    0],
       [   0, 1108,    5,    2,    0,    1,    4,    2,   13,    0],
       [   9,    6,  944,   11,    7,    1,   11,    8,   31,    4],
       [   7,    0,   28,  904,    0,   29,    3,   12,   19,    8],
       [   2,    1,   10,    0,  904,    0,    7,    5,   10,   43],
       [  16,    1,    8,   26,    9,  777,   13,    6,   28,    8],
       [  16,    3,   11,    1,    7,   12,  905,    2,    1,    0],
       [   2,    6,   26,    5,    5,    1,    0,  948,    3,   32],
       [  11,    7,    8,   17,    9,   31,   10,   11,  861,    9],
       [  12,    7,    1,    9,   22,    6,    0,   21,    5,  926]])>
```

In [23]:
```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[23]:
```
Text(69.0, 0.5, 'Truth')
```



Using hidden layer

In [24]:
```
model = keras.Sequential([
    keras.layers.Dense(100, input_shape=(784,), activation='relu'),
    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 [==============================] - 2s 815us/step - loss: 0.2671 - acc
uracy: 0.9237
Epoch 2/5
1875/1875 [==============================] - 2s 810us/step - loss: 0.1229 - acc
uracy: 0.9637
Epoch 3/5
1875/1875 [==============================] - 2s 827us/step - loss: 0.0850 - acc
uracy: 0.9745
Epoch 4/5
1875/1875 [==============================] - 2s 805us/step - loss: 0.0642 - acc
uracy: 0.9810
Epoch 5/5
1875/1875 [==============================] - 2s 809us/step - loss: 0.0517 - acc
uracy: 0.9840
```

Out[24]:
```
<keras.callbacks.History at 0x205c3b51ee0>
```

In [25]:
```
model.evaluate(X_test_flattened,y_test)
```

```
313/313 [==============================] - 0s 681us/step - loss: 0.0745 - accur
acy: 0.9762
```

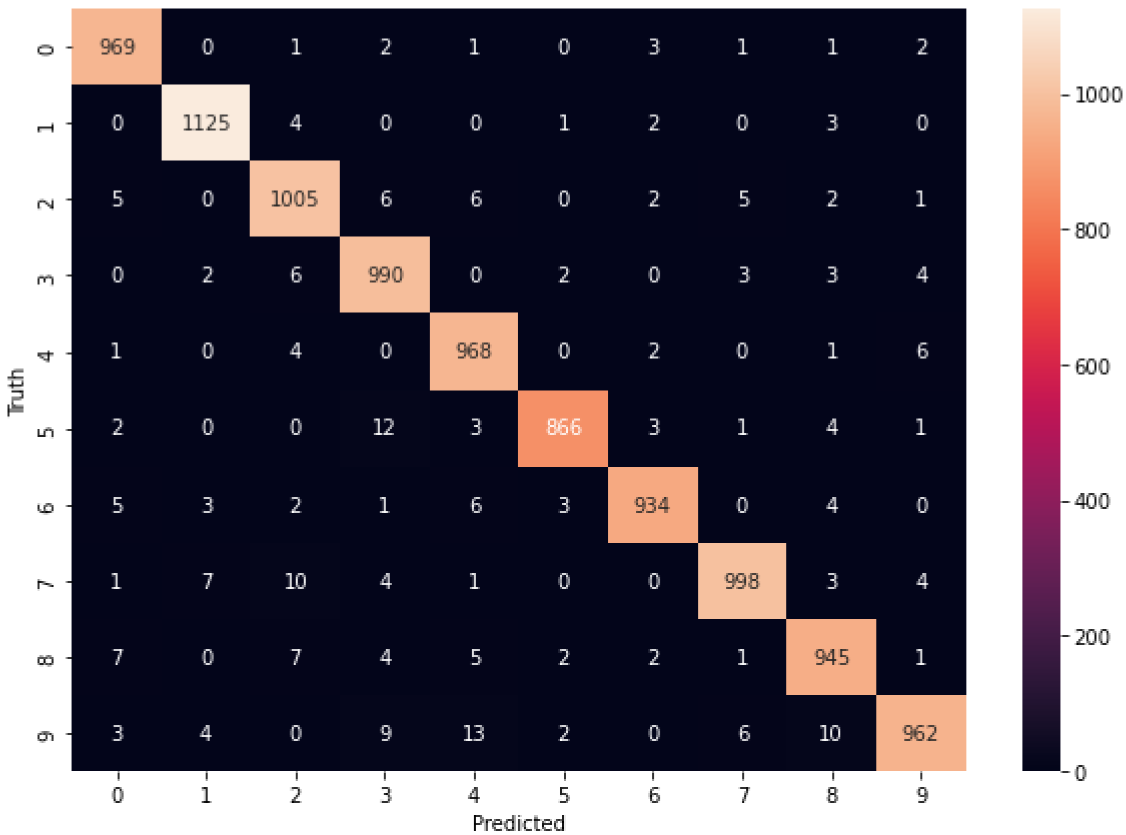
Out[25]:
```
[0.07447880506515503, 0.9761999845504761]
```

In [26]:
```
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 [==============================] - 0s 591us/step
```

Out[26]: `Text(69.0, 0.5, 'Truth')`



Using Flatten layer so that we don't have to call .reshape on input dataset

In [27]:
```python
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

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

model.fit(X_train, y_train, epochs=10)
```

```
1875/1875 [==============================] - 2s 814us/step - loss: 0.1238 - a
ccuracy: 0.9632
Epoch 3/10
1875/1875 [==============================] - 2s 812us/step - loss: 0.0870 - a
ccuracy: 0.9743
Epoch 4/10
1875/1875 [==============================] - 2s 845us/step - loss: 0.0664 - a
ccuracy: 0.9798
Epoch 5/10
1875/1875 [==============================] - 2s 811us/step - loss: 0.0528 - a
ccuracy: 0.9836
Epoch 6/10
1875/1875 [==============================] - 2s 814us/step - loss: 0.0418 - a
ccuracy: 0.9872
Epoch 7/10
1875/1875 [==============================] - 2s 818us/step - loss: 0.0338 - a
ccuracy: 0.9895
Epoch 8/10
1875/1875 [==============================] - 2s 810us/step - loss: 0.0276 - a
ccuracy: 0.9914
```

In [28]:
```python
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 655us/step - loss: 0.0868 - accur
acy: 0.9766
```

Out[28]:
```
[0.0867568701505661, 0.9765999913215637]
```