In [86]:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
#Keras in an neural netowrk API written in Python and integrated with Tensorflow
```

In [56]:

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
# We are using hand written data set from keras library
# https://keras.io/api/datasets/
```

In [87]:

```
len(X_train)
```

Out[87]:

```
60000
```

In [88]:

```
len(X_test)
```

Out[88]:

```
10000
```

In [93]:

```
X_train[0].shape
```

Out[93]:

```
(28, 28)
```

In [96]:

```
X_train[0]
```

Out[96]:

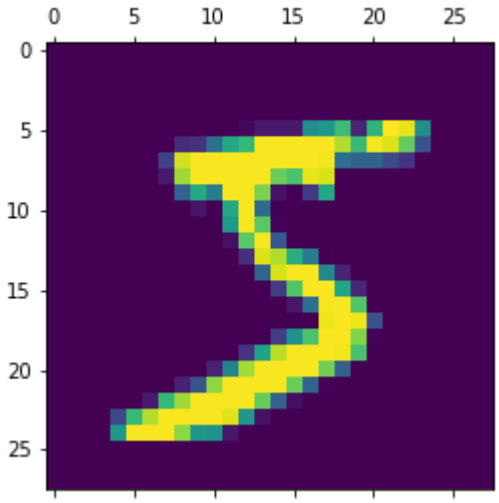
```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 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.        ],
```

In [106]:

```
plt.matshow(X_train[0])
```

Out[106]:

```
<matplotlib.image.AxesImage at 0x20f7f9cba60>
```



In [107]:

```
y_train[0]
```

Out[107]:

```
5
```

In [111]:

```
y_train[:5]
```

Out[111]:

```
array([8, 3, 5, 6], dtype=uint8)
```

In [64]:

```
X_train = X_train / 255
X_test = X_test / 255
```

In [112]:

```
X_train[0]
```

Out[112]:

```
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.        ].
```

In [66]:

```
X_train_flattened = X_train.reshape(len(X_train), 28*28)
X_test_flattened = X_test.reshape(len(X_test), 28*28)
```

In [67]:

```
X_train_flattened.shape
```

Out[67]:

```
(60000, 784)
```

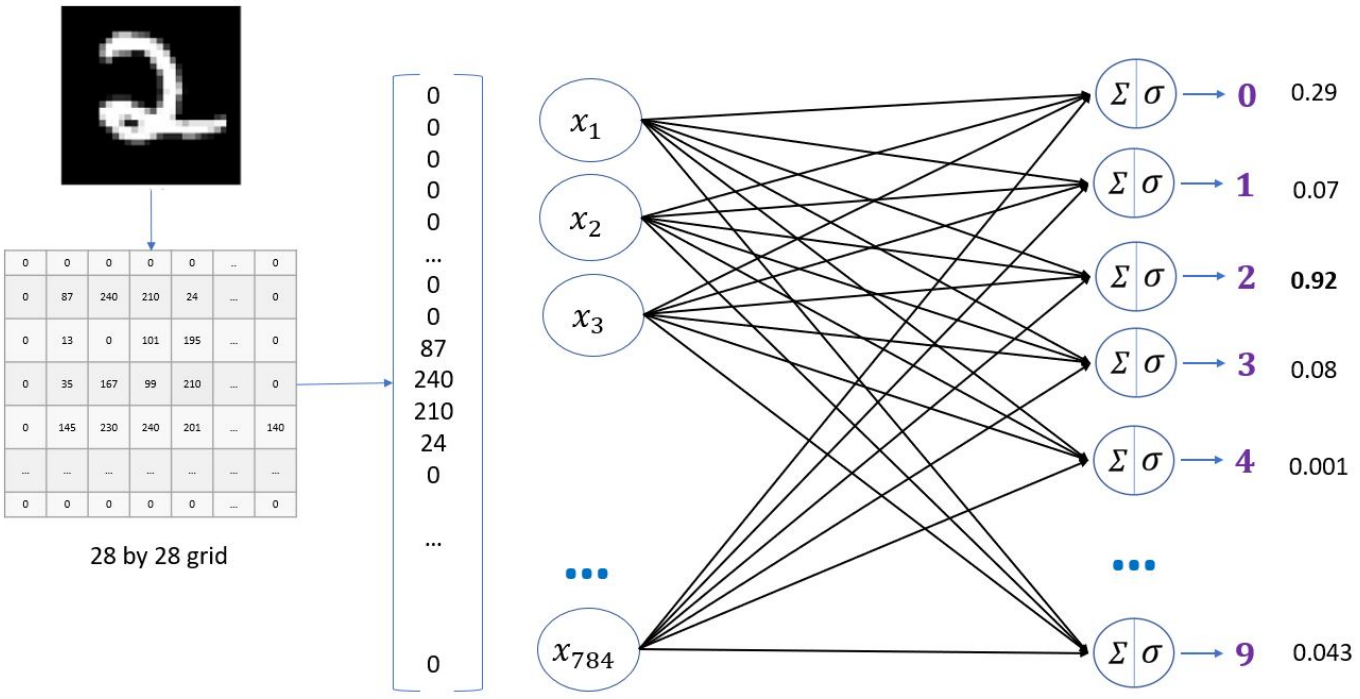
In [68]:

```
X_train_flattened[0]
```

Out[68]:

```
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.        ,
```

Very simple neural network with no hidden layers



In [113]:

```
#sequential- Stack of layers. Accepts every layer as an element
#Dense-all neurons connected
#In sparse_categorical_crossentropy, categorical means output class is categorical
#In sparse_categorical_crossentropy, sparse means output variable is actually an integer
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

model.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
              #loss='mean_squared_error',
              #loss='poisson',
              #loss='kl_divergence',
              #loss='mean_absolute_error',
              metrics=['accuracy'])

model.fit(X_train_flattened, y_train, epochs=10)
#Training happens using model.fit
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4899 - ac
curacy: 0.8788
Epoch 2/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3063 - ac
curacy: 0.9156
Epoch 3/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2855 - ac
curacy: 0.9216
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2750 - ac
curacy: 0.9245
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2676 - ac
curacy: 0.9262
Epoch 6/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2627 - ac
curacy: 0.9276
Epoch 7/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2594 - ac
curacy: 0.9292
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2556 - ac
curacy: 0.9305
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2527 - ac
curacy: 0.9307
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2506 - ac
curacy: 0.9307
```

Out[113]:

```
<tensorflow.python.keras.callbacks.History at 0x20f9889ea60>
```

In [70]:

```
model.evaluate(X_test_flattened, y_test)
```

```
313/313 [==============================] - 0s 421us/step - loss: 0.2618 - ac
curacy: 0.9279
```

Out[70]:

```
[0.2617749869823456, 0.9279000163078308]
```

In [120]:

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

Out[120]:

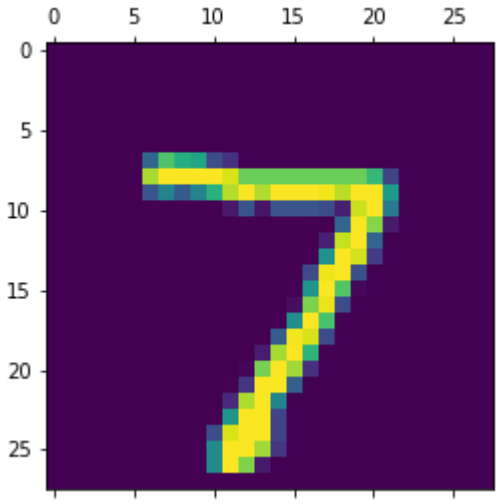
```
array([1.1853375e-06, 1.2054593e-12, 3.8440626e-06, 6.7408085e-03,
       2.6812589e-07, 5.1215618e-05, 1.6888341e-11, 6.6895723e-01,
       2.8096782e-05, 3.2195449e-04], dtype=float32)
```

In [121]:

```
plt.matshow(X_test[0])
```

Out[121]:

```
<matplotlib.image.AxesImage at 0x20f7ab8a0a0>
```



In [73]:

```
np.argmax(y_predicted[0])
```

Out[73]:

```
7
```

In [74]:

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

In [126]:

```
y_predicted_labels[:5]
```

Out[126]:

```
[7, 2, 1, 0, 4]
```

In [76]:

```
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
cm
```

Out[76]:

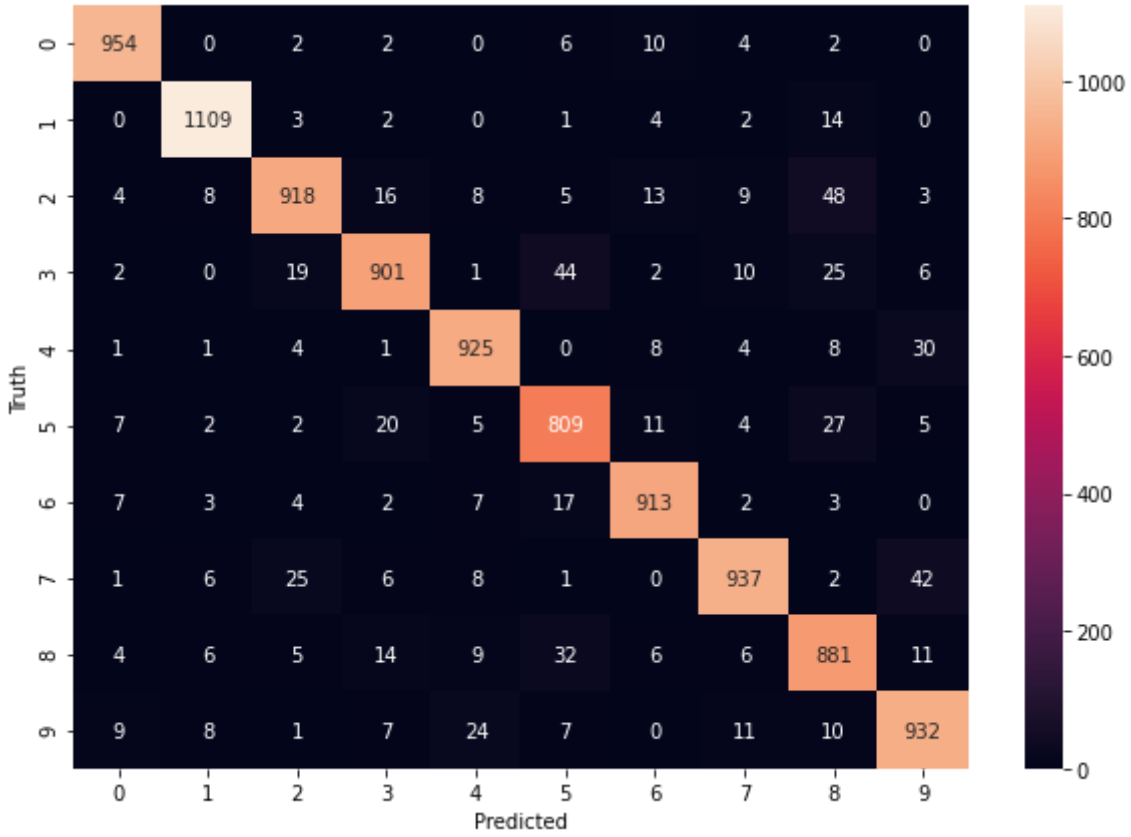
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 954,    0,    2,    2,    0,    6,   10,    4,    2,    0],
       [   0, 1109,    3,    2,    0,    1,    4,    2,   14,    0],
       [   4,    8,  918,   16,    8,    5,   13,    9,   48,    3],
       [   2,    0,   19,  901,    1,   44,    2,   10,   25,    6],
       [   1,    1,    4,    1,  925,    0,    8,    4,    8,   30],
       [   7,    2,    2,   20,    5,  809,   11,    4,   27,    5],
       [   7,    3,    4,    2,    7,   17,  913,    2,    3,    0],
       [   1,    6,   25,    6,    8,    1,    0,  937,    2,   42],
       [   4,    6,    5,   14,    9,   32,    6,    6,  881,   11],
       [   9,    8,    1,    7,   24,    7,    0,   11,   10,  932]])>
```

In [77]:

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

Out[77]:

```
Text(69.0, 0.5, 'Truth')
```



Using hidden layer

In [127]:

```
model = keras.Sequential([
    keras.layers.Dense(100, input_shape=(784,), activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

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

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

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2876 - ac
curacy: 0.9195
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1372 - ac
curacy: 0.9602
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1005 - ac
curacy: 0.9700
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0788 - ac
curacy: 0.9758
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0636 - ac
curacy: 0.9809
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0539 - ac
curacy: 0.9834
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0450 - ac
curacy: 0.9859
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0381 - ac
curacy: 0.9884
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0327 - ac
curacy: 0.9903
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0272 - ac
curacy: 0.9918
```

Out[127]:

```
<tensorflow.python.keras.callbacks.History at 0x20f78cef7f0>
```

In [128]:

```
model = keras.Sequential([
    keras.layers.Dense(100, input_shape=(784,), activation='relu'),
    keras.layers.Dense(50, input_shape=(100,), activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])

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

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

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2838 - ac
curacy: 0.9179: 0s - loss: 0.2993 - ac
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1272 - ac
curacy: 0.9620
Epoch 3/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0909 - ac
curacy: 0.9730
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0721 - ac
curacy: 0.9780
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0588 - ac
curacy: 0.9812
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0475 - ac
curacy: 0.9847
Epoch 7/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0395 - ac
curacy: 0.9871
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0341 - ac
curacy: 0.9888
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0296 - ac
curacy: 0.9903
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0251 - ac
curacy: 0.9921
```

Out[128]:

```
<tensorflow.python.keras.callbacks.History at 0x20f780c4c70>
```

In [129]:

```
model.evaluate(X_test_flattened,y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0968 - accu
racy: 0.9752
```

Out[129]:

```
[0.09683504700660706, 0.9751999974250793]
```

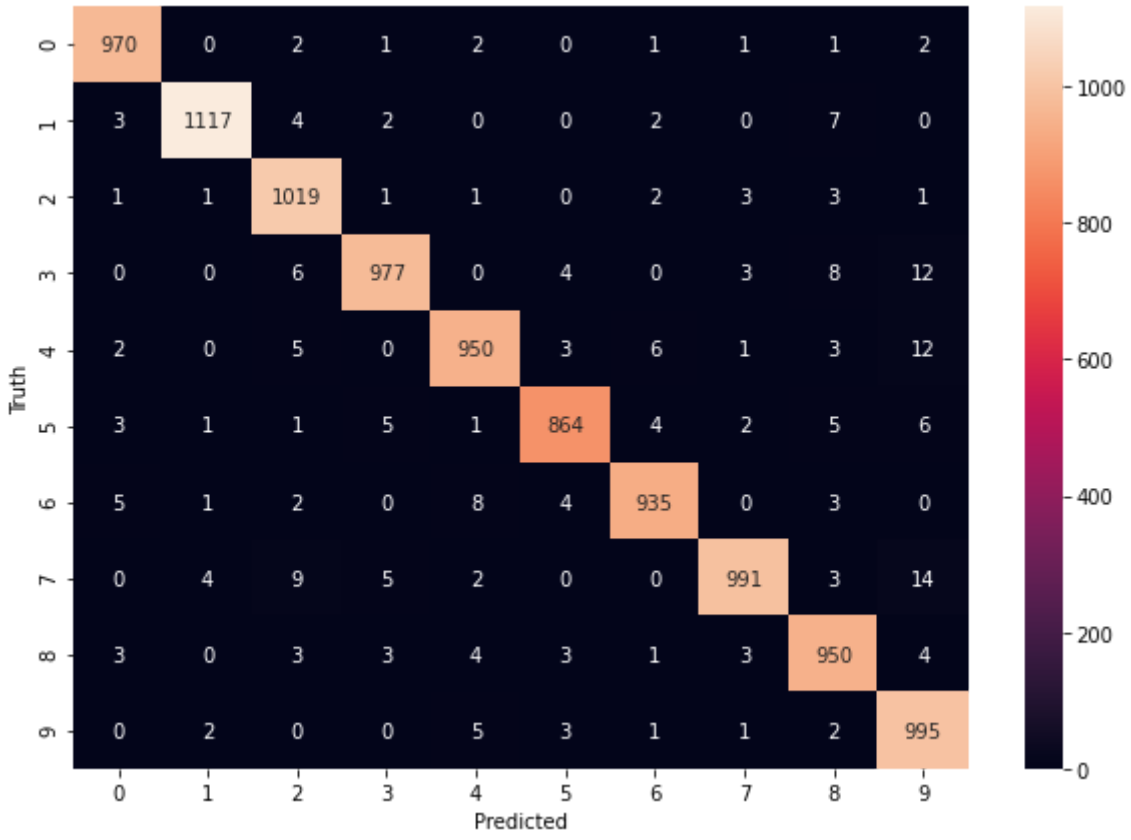
In [81]:

```
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')
```

Out[81]:

```
Text(69.0, 0.5, 'Truth')
```



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

In [130]:

```
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',
              #loss= 'poisson',
              metrics=['accuracy'])

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

```
Epoch 1/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2929 - ac
curacy: 0.9185
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1398 - ac
curacy: 0.9588
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0994 - ac
curacy: 0.9702
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0769 - ac
curacy: 0.9767
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0600 - ac
curacy: 0.9818
```

Out[130]:

```
<tensorflow.python.keras.callbacks.History at 0x20f78474910>
```

In [83]:

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 0s 621us/step - loss: 0.0825 - ac
curacy: 0.9748
```

Out[83]:

```
[0.08245112001895905, 0.9747999906539917]
```

In [84]:

```
#Loss is 0.101 & Accuracy is 97.22 percentage
```

In [85]:

```

```