In [1]:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
C:\Users\ashish\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood a
s (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning: P
```

```
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\ashish\Anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [28]:

```
(x_train,y_train), (x_test,y_test) = keras.datasets.mnist.load_data()
```

In [29]:

```
len(x_train)
```

Out[29]:

```
60000
```

In [30]:

```
len(y_train)
```

Out[30]:

```
60000
```

In [31]:

```
len(x_test)
```

Out[31]:

```
10000
```

In [32]:

```
len(y_test)
```

Out[32]:

```
10000
```

In [33]:

```
x_train[0].shape
```

Out[33]:

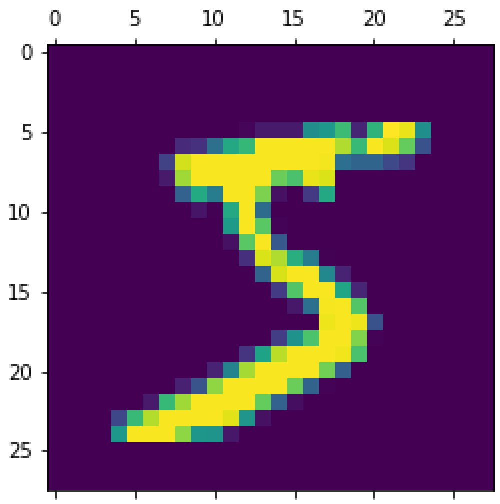
```
(28, 28)
```

In [34]:
```
x_train[0]
```

```
        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)
```

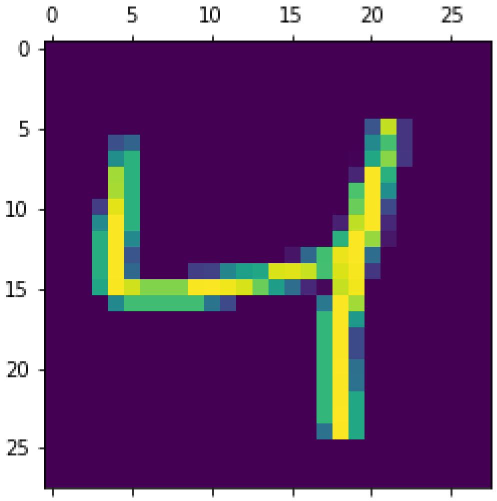
In [35]:
```
plt.matshow(x_train[0])
```

Out[35]:
```
<matplotlib.image.AxesImage at 0x18d4f0f5390>
```



In [36]:
```
plt.matshow(x_train[2])
```

Out[36]:
```
<matplotlib.image.AxesImage at 0x18d4e106b38>
```



In [37]:
```
y_train[2]
```

Out[37]:
```
4
```

In [38]:
```
y_train[0:5]
```

Out[38]:
```
array([5, 0, 4, 1, 9], dtype=uint8)
```

In [39]:
```
x_train.shape
```

Out[39]:
```
(60000, 28, 28)
```

```
In [40]: # flattening the data -> converting 2d array into 1d array

         x_train_fattened = x_train.reshape(len(x_train),28*28)
         x_test_fattened = x_test.reshape(len(x_test),28*28)
         print(x_test_fattened.shape)
         x_train_fattened.shape

         (10000, 784)

Out[40]: (60000, 784)
```

```
In [41]: x_train_fattened[0]
```

```
       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 [42]: # create simple neural network
```

In [43]:

```
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_fattened,y_train,epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 4s 62us/sample - loss: 2.0314 - acc: 0.3611
Epoch 2/5
60000/60000 [==============================] - 3s 55us/sample - loss: 1.5310 - acc: 0.4308
Epoch 3/5
60000/60000 [==============================] - 3s 54us/sample - loss: 1.4403 - acc: 0.4670
Epoch 4/5
60000/60000 [==============================] - 3s 56us/sample - loss: 1.3946 - acc: 0.4826
Epoch 5/5
60000/60000 [==============================] - 3s 55us/sample - loss: 1.3422 - acc: 0.5019
```

Out[43]:

```
<tensorflow.python.keras.callbacks.History at 0x18d6c8c7278>
```

In [44]:

```
# scaling -> cranging data between 0 to 1

x0_train = x_train / 255
x0_test = x_test / 255

x0_train_fattened = x0_train.reshape(len(x0_train),28*28)
x0_test_fattened = x0_test.reshape(len(x0_test),28*28)
print('x0_test_fattened_shape - ',x0_test_fattened.shape)
print('x0_train_fattened_shape - ',x0_train_fattened.shape)

x0_train_fattened[0]
```

Out[44]:

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

In [45]:

```
# create simple neural network

model = keras.Sequential([
    keras.layers.Dense(10,input_shape=(784,),activation='sigmoid')
])
model.compile(
    optimizer='adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)

model.fit(x0_train_fattened,y_train,epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 5s 76us/sample - loss: 0.4880 - acc: 0.8781
Epoch 2/5
60000/60000 [==============================] - 3s 58us/sample - loss: 0.3063 - acc: 0.9149
Epoch 3/5
60000/60000 [==============================] - 3s 58us/sample - loss: 0.2855 - acc: 0.9211
Epoch 4/5
60000/60000 [==============================] - 4s 69us/sample - loss: 0.2749 - acc: 0.9244
Epoch 5/5
60000/60000 [==============================] - 4s 62us/sample - loss: 0.2674 - acc: 0.9268
```

Out[45]:

```
<tensorflow.python.keras.callbacks.History at 0x18d4f361908>
```

***above you can see hoe scaling is more important for better accuracy***

In [46]:

```
# evalutationg accuracy in test dataset without scaling
model.evaluate(x_test_fattened,y_test)
```

```
10000/10000 [==============================] - 0s 34us/sample - loss: 2.0055 - acc: 0.2027
```

Out[46]:

```
[2.005459748458862, 0.2027]
```

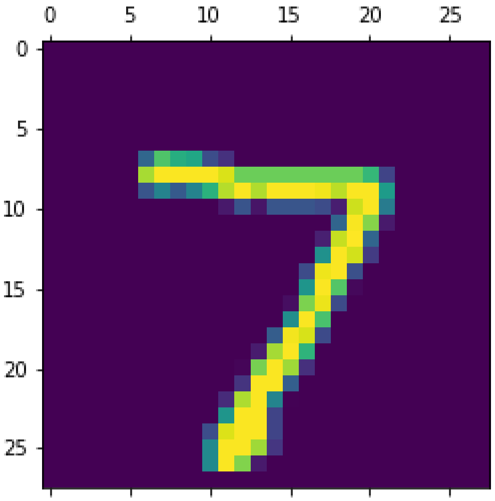
In [49]:
```
# evalutationg accuracy in test dataset with scaling

model.evaluate(x0_test_fattened,y_test)
```

```
10000/10000 [==============================] - 0s 31us/sample - loss: 0.2679 - acc: 0.9244
```

Out[49]:
```
[0.2679107921272516, 0.9244]
```

In [51]:
```
plt.matshow(x0_test[0])
```

Out[51]:
```
<matplotlib.image.AxesImage at 0x18d4d94a6a0>
```



In [52]:
```
y0_predicted = model.predict(x0_test_fattened)
y0_predicted[0]
```

Out[52]:
```
array([1.2993813e-05, 0.0000000e+00, 3.4838915e-05, 1.4789969e-02,
       1.2218952e-06, 7.4625015e-05, 0.0000000e+00, 7.5696909e-01,
       8.2343817e-05, 1.0018945e-03], dtype=float32)
```

In [53]:
```
np.argmax(y0_predicted[0])
```

Out[53]:
```
7
```

```
In [54]: np.argmax(y0_predicted[1])
```

```
Out[54]: 2
```

```
In [72]: y0_predicted_labels = [np.argmax(i) for i in y0_predicted]
         y0_predicted_labels[:5]
```

```
Out[72]: [7, 2, 1, 0, 4]
```

```
In [71]: y_test[:5]
```

```
Out[71]: array([7, 2, 1, 0, 4], dtype=uint8)
```

```
In [79]: sess = tf.Session()
```

```
In [81]: # confusion matrix

         cm = tf.math.confusion_matrix(y_test,y0_predicted_labels)
         cm = cm.eval(session=sess)
         cm
```

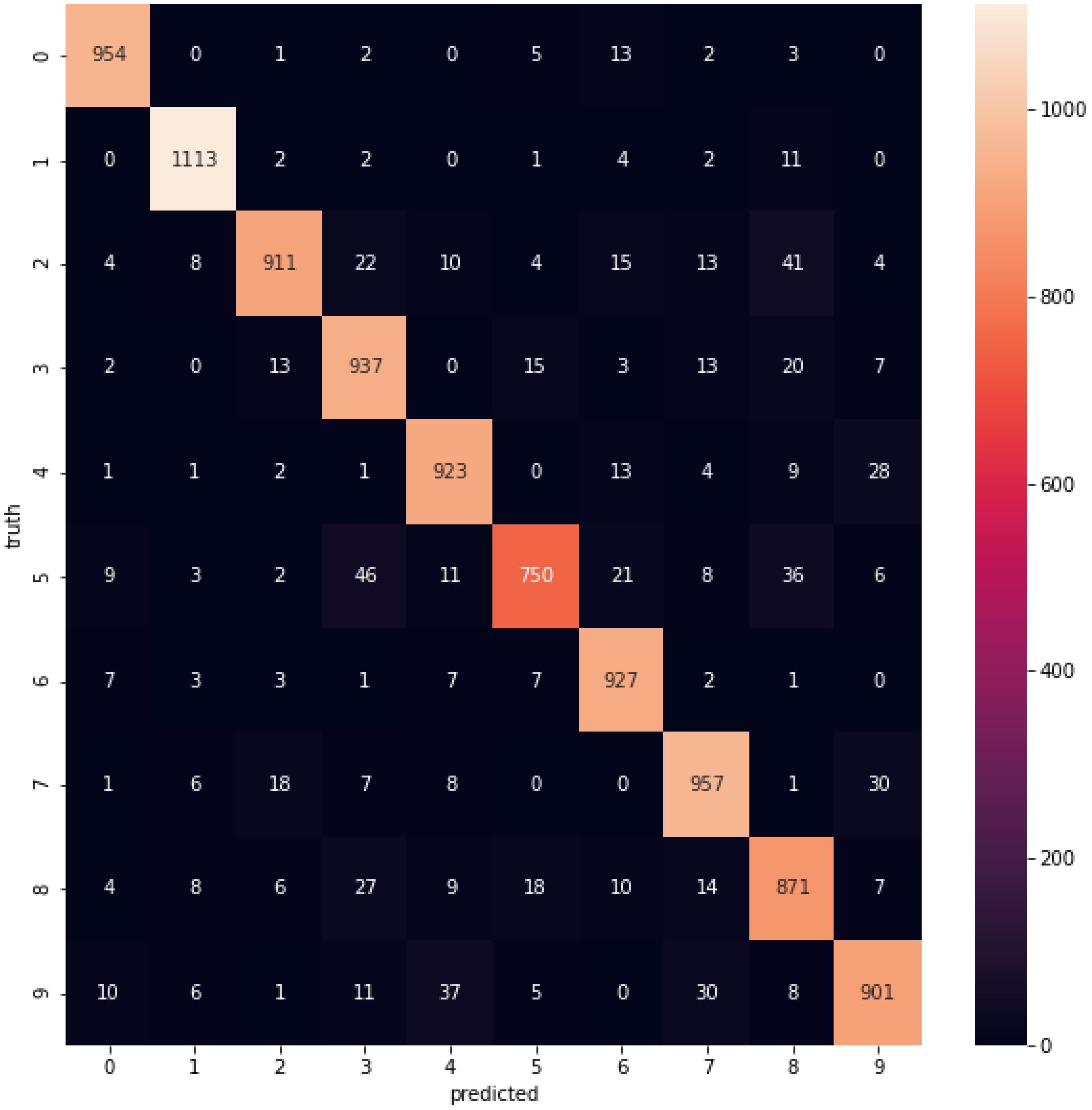
```
Out[81]: array([[ 954,    0,    1,    2,    0,    5,   13,    2,    3,    0],
                [   0, 1113,    2,    2,    0,    1,    4,    2,   11,    0],
                [   4,    8,  911,   22,   10,    4,   15,   13,   41,    4],
                [   2,    0,   13,  937,    0,   15,    3,   13,   20,    7],
                [   1,    1,    2,    1,  923,    0,   13,    4,    9,   28],
                [   9,    3,    2,   46,   11,  750,   21,    8,   36,    6],
                [   7,    3,    3,    1,    7,    7,  927,    2,    1,    0],
                [   1,    6,   18,    7,    8,    0,    0,  957,    1,   30],
                [   4,    8,    6,   27,    9,   18,   10,   14,  871,    7],
                [  10,    6,    1,   11,   37,    5,    0,   30,    8,  901]])
```

In [83]:

```
import seaborn as sns

plt.figure(figsize=(10,10))
sns.heatmap(cm,annot=True,fmt='d')
plt.xlabel('predicted')
plt.ylabel('truth')
```

Out[83]:

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



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 954 | 0 | 1 | 2 | 0 | 5 | 13 | 2 | 3 | 0 |
| 1 | 0 | 1113 | 2 | 2 | 0 | 1 | 4 | 2 | 11 | 0 |
| 2 | 4 | 8 | 911 | 22 | 10 | 4 | 15 | 13 | 41 | 4 |
| 3 | 2 | 0 | 13 | 937 | 0 | 15 | 3 | 13 | 20 | 7 |
| 4 | 1 | 1 | 2 | 1 | 923 | 0 | 13 | 4 | 9 | 28 |
| 5 | 9 | 3 | 2 | 46 | 11 | 750 | 21 | 8 | 36 | 6 |
| 6 | 7 | 3 | 3 | 1 | 7 | 7 | 927 | 2 | 1 | 0 |
| 7 | 1 | 6 | 18 | 7 | 8 | 0 | 0 | 957 | 1 | 30 |
| 8 | 4 | 8 | 6 | 27 | 9 | 18 | 10 | 14 | 871 | 7 |
| 9 | 10 | 6 | 1 | 11 | 37 | 5 | 0 | 30 | 8 | 901 |

In [86]:

```
# create simple neural network with hidden layer

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(x0_train_fattened,y_train,epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 6s 108us/sample - loss: 0.2933 - acc: 0.9214
Epoch 2/5
60000/60000 [==============================] - 5s 86us/sample - loss: 0.1371 - acc: 0.9596
Epoch 3/5
60000/60000 [==============================] - 5s 89us/sample - loss: 0.0985 - acc: 0.9704
Epoch 4/5
60000/60000 [==============================] - 5s 87us/sample - loss: 0.0736 - acc: 0.9782
Epoch 5/5
60000/60000 [==============================] - 6s 106us/sample - loss: 0.0602 - acc: 0.9820
```

Out[86]:

```
<tensorflow.python.keras.callbacks.History at 0x18d025fba90>
```

In [87]:

```
model.evaluate(x0_test_fattened,y_test)
```

```
10000/10000 [==============================] - 1s 79us/sample - loss: 0.0789 - acc: 0.9754
```

Out[87]:

```
[0.0788839316977188, 0.9754]
```

In [90]:

```
# create simple neural network with hidden layer

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(x0_train,y_train,epochs=5)
```

```
Epoch 1/5
60000/60000 [==============================] - 7s 108us/sample - loss: 0.2972 - acc: 0.9187
Epoch 2/5
60000/60000 [==============================] - 5s 90us/sample - loss: 0.1367 - acc: 0.9605
Epoch 3/5
60000/60000 [==============================] - 6s 93us/sample - loss: 0.0973 - acc: 0.9712
Epoch 4/5
60000/60000 [==============================] - 5s 91us/sample - loss: 0.0748 - acc: 0.9782
Epoch 5/5
60000/60000 [==============================] - 6s 93us/sample - loss: 0.0616 - acc: 0.9819
```

Out[90]:

```
<tensorflow.python.keras.callbacks.History at 0x18d02beaeb8>
```

In [ ]: