

```
In [ ]: #Lab assignment 2
```

```
In [2]: # Import the necessary packages
import tensorflow as tf
```

```
In [3]: from tensorflow import keras
```

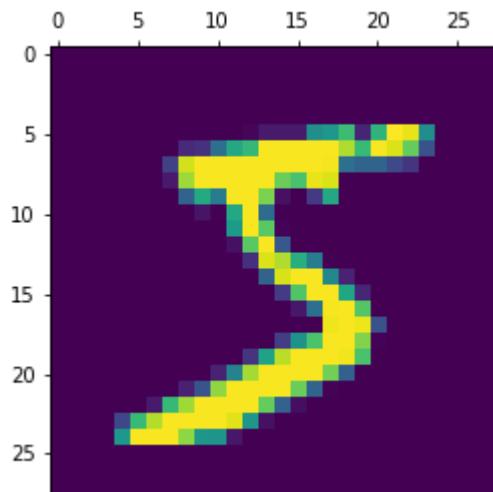
```
In [4]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
%matplotlib inline
```

```
In [5]: #Load the training and testing data (MNIST/CIFAR10)
mnist=tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [=====] - 9s 1us/step

```
In [6]: #Define the network architecture
plt.matshow(x_train[0])
```

```
Out[6]: <matplotlib.image.AxesImage at 0x2af5c89c1f0>
```



```
In [7]: x_train=x_train/255
x_test=x_test/255
```

```
In [8]: x_train[0]
```


O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , 0.54509804, 0.99215686, 0.74509804, 0.00784314,
O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , 0.04313725, 0.74509804, 0.99215686, 0.2745098 ,
O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , 0.31764706, 0.94117647,
0.99215686, 0.99215686, 0.46666667, 0.09803922, O. ,
O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , O. , 0.17647059,
0.72941176, 0.99215686, 0.99215686, 0.58823529, 0.10588235,
O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , 0.0627451 , 0.36470588, 0.98823529, 0.99215686, 0.73333333,
O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , 0.97647059, 0.99215686, 0.97647059,
0.25098039, O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , O. , 0.18039216,
0.50980392, 0.71764706, 0.99215686, 0.99215686, 0.81176471,
0.00784314, O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , 0.15294118, 0.58039216, 0.89803922,
0.99215686, 0.99215686, 0.99215686, 0.98039216, 0.71372549,
O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , 0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.78823529, 0.30588235, O. ,
O. , O. , O. , O. , O. ,
O. , O. , O.],
[O. , O. , O. , O. , O. ,
O. , O. , O. , O. , O. ,
O. , O. , O. , 0.09019608, 0.25882353,

0.83529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.77647059, 0.31764706, 0.00784314, 0. , 0. ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0.],
[0. , 0. , 0. , 0. , 0. , ,
0. , 0.07058824, 0.67058824, 0.85882353, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.76470588, 0.31372549,
0.03529412, 0. , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0.],
[0. , 0. , 0. , 0. , 0.21568627,
0.6745098 , 0.88627451, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.95686275, 0.52156863, 0.04313725, 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0.],
[0. , 0. , 0. , 0. , 0.53333333,
0.99215686, 0.99215686, 0.99215686, 0.83137255, 0.52941176,
0.51764706, 0.0627451 , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0.],
[0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0. , 0. , 0. , ,
0. , 0. , 0.],
[0. , 0. , 0. , 0. , 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 [9]: #Train the model using SGD  
model=keras.Sequential([  
    keras.layers.Flatten(input_shape=(28,28)),  
    keras.layers.Dense(128,activation='relu'),  
    keras.layers.Dense(10,activation='softmax')  
])
```

In [10]: model.summary()

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290
<hr/>		
Total params: 101770 (397.54 KB)		
Trainable params: 101770 (397.54 KB)		
Non-trainable params: 0 (0.00 Byte)		

```
In [11]: model.compile(optimizer='sgd',
                     loss='sparse_categorical_crossentropy',
                     metrics=['accuracy'])
```

```
In [12]: # Evaluate the network
history=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10)
```

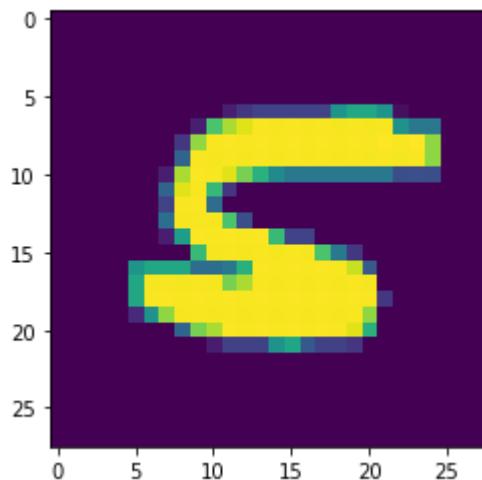
```
Epoch 1/10
1875/1875 [=====] - 5s 2ms/step - loss: 0.6451 - accuracy: 0.838
2 - val_loss: 0.3547 - val_accuracy: 0.9027
Epoch 2/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.3343 - accuracy: 0.906
7 - val_loss: 0.2920 - val_accuracy: 0.9185
Epoch 3/10
1875/1875 [=====] - 4s 2ms/step - loss: 0.2867 - accuracy: 0.918
6 - val_loss: 0.2610 - val_accuracy: 0.9267
Epoch 4/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.2571 - accuracy: 0.9272
- val_loss: 0.2383 - val_accuracy: 0.9322
Epoch 5/10
1875/1875 [=====] - 4s 2ms/step - loss: 0.2348 - accuracy: 0.934
0 - val_loss: 0.2195 - val_accuracy: 0.9376
Epoch 6/10
1875/1875 [=====] - 4s 2ms/step - loss: 0.2169 - accuracy: 0.938
6 - val_loss: 0.2049 - val_accuracy: 0.9420
Epoch 7/10
1875/1875 [=====] - 4s 2ms/step - loss: 0.2011 - accuracy: 0.9431
- val_loss: 0.1915 - val_accuracy: 0.9452
Epoch 8/10
1875/1875 [=====] - 4s 2ms/step - loss: 0.1877 - accuracy: 0.9467
- val_loss: 0.1819 - val_accuracy: 0.9458
Epoch 9/10
1875/1875 [=====] - 4s 2ms/step - loss: 0.1759 - accuracy: 0.9501
- val_loss: 0.1702 - val_accuracy: 0.9515
Epoch 10/10
1875/1875 [=====] - 4s 2ms/step - loss: 0.1653 - accuracy: 0.9532
- val_loss: 0.1611 - val_accuracy: 0.9525
```

```
In [13]: # Plot the training loss and accuracy
test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%3f" %test_loss)
print("Accuracy=%3f" %test_acc)
```

```
313/313 [=====] - 0s 1ms/step - loss: 0.1611 - accuracy: 0.9525
Loss=0.161
Accuracy=0.952
```

```
In [14]: n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show
```

```
Out[14]: <function matplotlib.pyplot.show(close=None, block=None)>
```



```
In [15]: test_predict=model.predict(x_test)
test_predict_labels=np.argmax(test_predict,axis=1)
confusion_matrix=tf.math.confusion_matrix(labels=y_test,predictions=test_predict_labels)
print('confusion matrix of the test set :\n',confusion_matrix)
```

313/313 [=====] - 0s 1ms/step

confusion matrix of the test set :

```
tf.Tensor(
[[ 963   0   1   1   0   4   7   2   2   0]
 [  0 1112   4   2   1   1   4   2   9   0]
 [  8   2 975   5   7   2   7  10  15   1]
 [  0   1   5 961   1  15   1  10  12   4]
 [  1   0   6   0 937   0  10   3   2  23]
 [  8   1   1  20   2 830  10   2  10   8]
 [ 10   3   2   0   8   8 921   1   5   0]
 [  1   6  17   6   3   1   0 980   3  11]
 [  4   3   3  15   8   9  11   9 909   3]
 [  9   6   1   9  25   4   1  12   5 937]], shape=(10, 10), dtype=int32)
```

```
In [ ]:
```