```
import tensorflow as tf
from tensorflow import keras
```

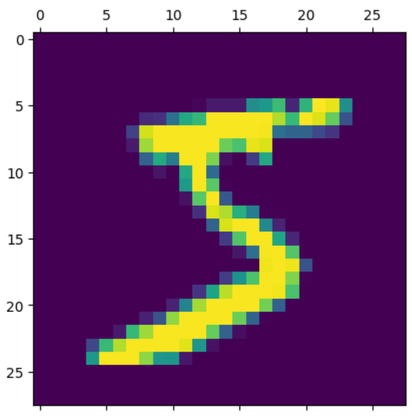
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
%matplotlib inline
```

```
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 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

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

```
<matplotlib.image.AxesImage at 0x7e68b3232930>
```



```
x_train = x_train/255
x_test = x_test/255
```

```
x_train[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.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
         0.99215686, 0.99215686, 0.78823529, 0.30588235, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 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.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ]])
```

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28,28)),
    keras.layers.Dense(128,activation='relu'),
    keras.layers.Dense(10,activation='softmax')
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Seque
  super().__init__(**kwargs)
```

```
model.summary()
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ flatten (Flatten)                │ (None, 784)            │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                    │ (None, 128)            │       100,480 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 10)             │         1,290 │
└──────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 101,770 (397.54 KB)
 Trainable params: 101,770 (397.54 KB)
```

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

```
history = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.7368 - loss: 1.0267 - val_accuracy: 0.9051 - val_loss: 0.3579
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9023 - loss: 0.3540 - val_accuracy: 0.9171 - val_loss: 0.2979
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9186 - loss: 0.2959 - val_accuracy: 0.9276 - val_loss: 0.2625
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9262 - loss: 0.2615 - val_accuracy: 0.9326 - val_loss: 0.2390
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9316 - loss: 0.2472 - val_accuracy: 0.9373 - val_loss: 0.2240
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9384 - loss: 0.2238 - val_accuracy: 0.9406 - val_loss: 0.2071
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9421 - loss: 0.2094 - val_accuracy: 0.9446 - val_loss: 0.1933
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9479 - loss: 0.1898 - val_accuracy: 0.9488 - val_loss: 0.1821
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9523 - loss: 0.1753 - val_accuracy: 0.9510 - val_loss: 0.1720
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9531 - loss: 0.1677 - val_accuracy: 0.9535 - val_loss: 0.1629
```

```
test_loss,test_acc = model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
```

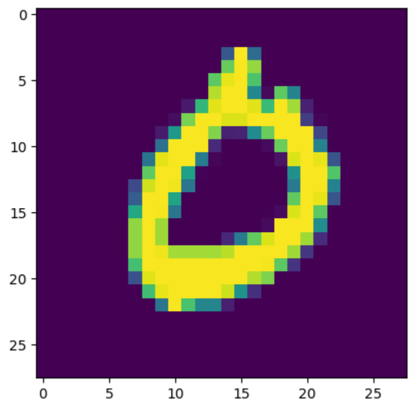
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9441 - loss: 0.1901
Loss=0.163
Accuracy=0.953
```

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

```
matplotlib.pyplot.show
def show(*args, **kwargs) -> None

/usr/local/lib/python3.12/dist-packages/matplotlib/pyplot.py
Display all open figures.

Parameters
----------
block : bool, optional
```

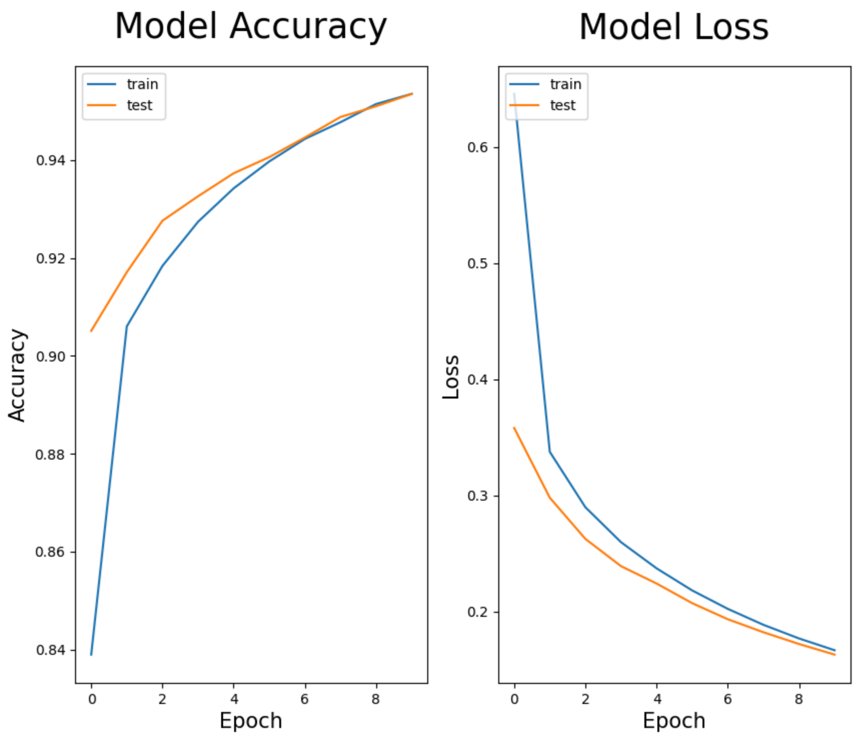


```
plt.figure(figsize=[10,8])

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')



plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



```
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 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Confusion matrix of the test set:
 tf.Tensor(
[[ 967    0    1    1    0    4    5    1    1    0]
 [    0 1118    2    2    0    1    4    2    6    0]
 [    7    2  980    8    8    1    9    8    8    1]
 [    0    0   10  965    0    7    2   10   12    4]
 [    1    1    5    1  946    0    8    2    2   16]
 [    9    2    1   20    2  832   10    1    9    6]
 [   10    3    3    1   10    9  917    0    5    0]
 [    1    7   21    6    4    1    0  969    3   16]
 [    5    3    5   14    8    7   11    9  908    4]
 [    8    7    2   12   24    6    1   10    6  933]], shape=(10, 10), dtype=int32)
```

Start coding or generate with AI.