```
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()
```

```
len(x_train)
```

```
60000
```

```
len(x_test)
```

```
10000
```

```
len(y_test)
```

```
10000
```

```
len(y_train)
```

```
60000
```

```
x_train.shape
```

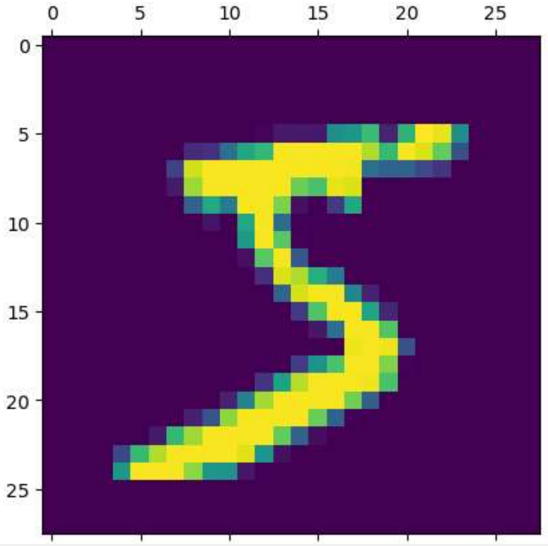
```
(60000, 28, 28)
```

```
x_train[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,   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)
```

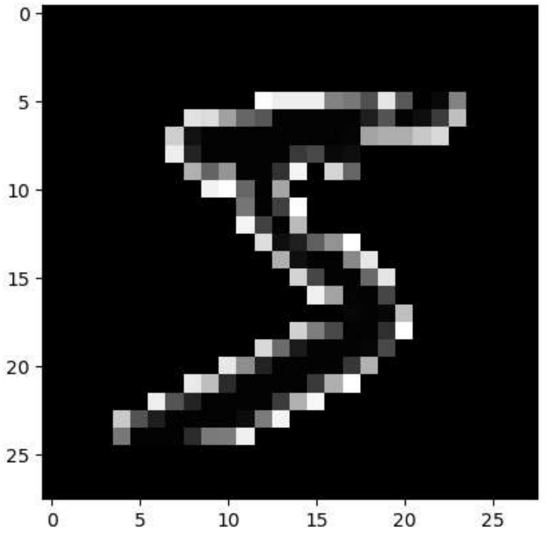
```
plt.matshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x2a465bc9b90>
```



```
plt.imshow(-x_train[0], cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x2a465d9c750>
```



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

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

```
x_train[0]
```

```
        3.89081123e-03, 3.09111880e-03, 1.19953864e-03, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        3.53710111e-04, 1.01499423e-03, 3.27566321e-03, 3.89081123e-03,
        3.89081123e-03, 3.89081123e-03, 3.89081123e-03, 3.04498270e-03,
        1.24567474e-03, 3.07574010e-05, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 2.76816609e-04, 2.62975779e-03,
        3.36793541e-03, 3.89081123e-03, 3.89081123e-03, 3.89081123e-03,
        3.89081123e-03, 2.99884660e-03, 1.23029604e-03, 1.38408304e-04,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        8.45828527e-04, 2.64513649e-03, 3.47558631e-03, 3.89081123e-03,
        3.89081123e-03, 3.89081123e-03, 3.89081123e-03, 3.75240292e-03,
        2.04536717e-03, 1.69165705e-04, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        2.09150327e-03, 3.89081123e-03, 3.89081123e-03, 3.89081123e-03,
        3.26028451e-03, 2.07612457e-03, 2.02998847e-03, 2.46059208e-04,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00]]])
```

```
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),    #Input layer
keras.layers.Dense(128, activation="relu"),    #hidden layer abs
keras.layers.Dense(10, activation="softmax")   #output layer
])
model.summary()
```

```
C:\Users\hp\anaconda3\Lib\site-packages\keras\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_c
  super().__init__(**kwargs)
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)
 Non-trainable params: 0 (0.00 B)
```

```
model.compile(optimizer="sgd",   # Stochastic Gradient Descent
loss="sparse_categorical_crossentropy",    # crossentropy reduces the loss
metrics=['accuracy'])
```

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

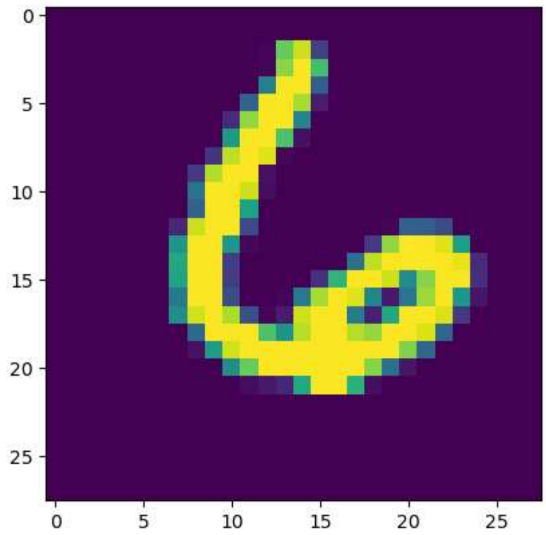
```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.1161 - loss: 2.3006 - val_accuracy: 0.1135 - val_loss: 2.2994
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1127 - loss: 2.2997 - val_accuracy: 0.1135 - val_loss: 2.2990
```

```
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.1122 - loss: 2.2993 - val_accuracy: 0.1135 - val_loss: 2.2988
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1119 - loss: 2.2994 - val_accuracy: 0.1135 - val_loss: 2.2982
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1106 - loss: 2.2990 - val_accuracy: 0.1135 - val_loss: 2.2980
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.1097 - loss: 2.2984 - val_accuracy: 0.1135 - val_loss: 2.2976
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1121 - loss: 2.2980 - val_accuracy: 0.1135 - val_loss: 2.2971
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1116 - loss: 2.2974 - val_accuracy: 0.1135 - val_loss: 2.2966
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1140 - loss: 2.2964 - val_accuracy: 0.1135 - val_loss: 2.2961
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.1129 - loss: 2.2963 - val_accuracy: 0.1135 - val_loss: 2.2956
```

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.1160 - loss: 2.2958
Loss=2.296
Accuracy=0.113
```

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



```
predicted_value=model.predict(x_test)
print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Handwritten number in the image is= 1
```

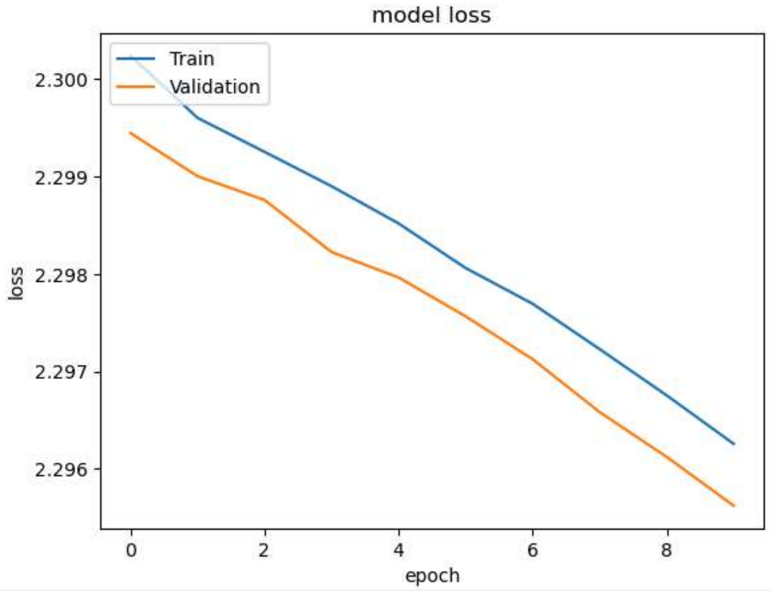
```
history.history??
```

```
Type:        dict
String form: {'accuracy': [0.11311666667461395, 0.11236666887998581, 0.11236666887998581, 0.11236666887998581, 0.11236666887998581,
0.11236666887998581, 0.11236666887998581, 0.11236666887998581, 0.11236666887998581, 0.11236666887998581], 'loss':
[2.3002381324768066, 2.2996013164520264, 2.299252510070801, 2.2988998889923096, 2.298517942428589, 2.298060894012451,
2.297693967819214, 2.2972288131713867, 2.2967543601989746, 2.2962567806243896], 'val_accuracy': [0.11349999904632568,
0.11349999904632568, 0.11349999904632568, 0.11349999904632568, 0.11349999904632568, 0.11349999904632568, 0.11349999904632568,
0.11349999904632568, 0.11349999904632568, 0.11349999904632568], 'val_loss': [2.299445152282715, 2.2990026473999023,
2.2987585067749023, 2.298224449157715, 2.297963857650757, 2.297565221786499, 2.2971255779266357, 2.2965824604034424,
2.2961227893829346, 2.2956223487854004]}
Length:      4
Docstring:
dict() -> new empty dictionary
dict(mapping) -> new dictionary initialized from a mapping object's
    (key, value) pairs
dict(iterable) -> new dictionary initialized as if via:
    d = {}
    for k, v in iterable:
        d[k] = v
```

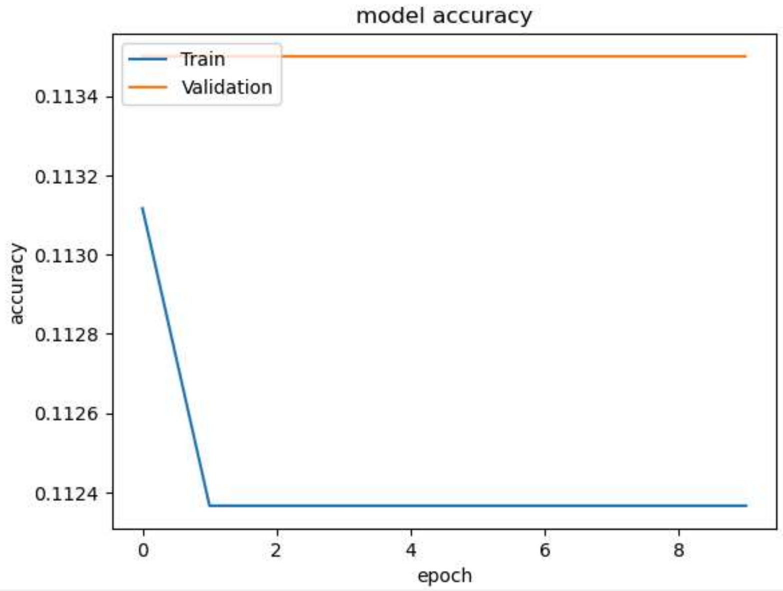
```
history.history.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training Loss and accuracy')
plt.ylabel('accuracy/Loss')
plt.xlabel('epoch')
plt.legend(['accuracy', 'val_accuracy','loss','val_loss'])
plt.show()
```