```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

```
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
X_train
```

```
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],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       ...,

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [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)
```

```
X_train.shape
```

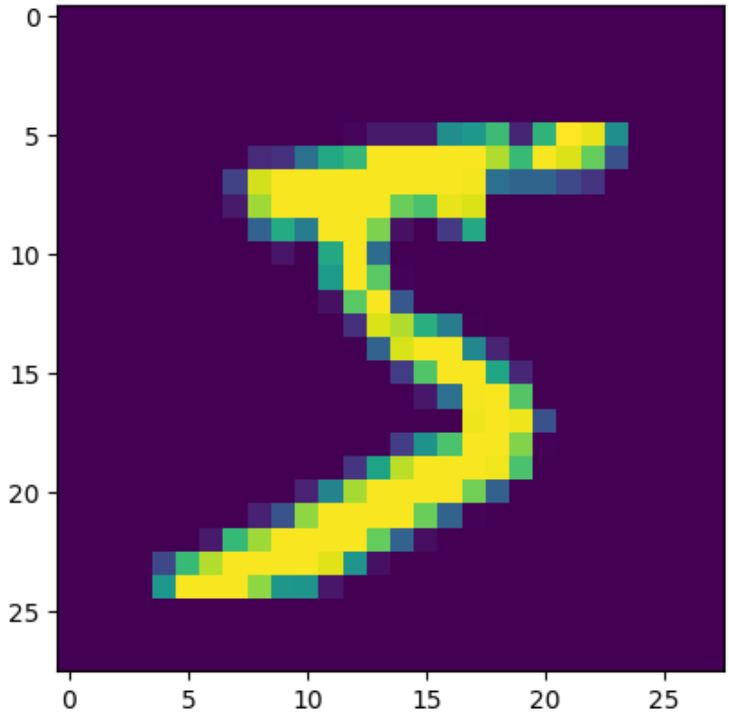
```
(60000, 28, 28)
```

```
y_train
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
import matplotlib.pyplot as plt
plt.imshow(X_train[0])
```

```
<matplotlib.image.AxesImage at 0x78b687310990>
```




```
#scaling
X_train = X_train/255
X_test = X_test/255
```

```
X_train[0]
```

```
 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.        ]])
```

```
#Input features are a total of  28*28 = 784, but they're stored in a 2D manner before sending them
#We need to flatten the dataset (28,28) -> (1,784)
model = Sequential()
model.add(Flatten(input_shape = (28,28))) # Added the input Layer (layer0)
model.add(Dense(128, activation = 'relu')) #Layer1 = Hidden Layer
model.add(Dense(32, activation = 'relu')) #Layer2 = Hidden Layer
model.add(Dense(10, activation = 'softmax')) #Layer3 = Output Layer = 10 labels(0 to 9)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning:
  super().__init__(**kwargs)
```

```
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
flatten_3 (Flatten)	(None, 784)	0
dense_6 (Dense)	(None, 128)	100,480
dense_7 (Dense)	(None, 32)	4,128
dense_8 (Dense)	(None, 10)	330

Total params: 104,938 (409.91 KB)
Trainable params: 104,938 (409.91 KB)
Non-trainable params: 0 (0.00 B)

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

```
history = model.fit(X_train, y_train, epochs = 15, validation_split = 0.2)
```

```
Epoch 1/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9959 - loss: 0.0111 - val_accuracy: 0.
Epoch 2/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9969 - loss: 0.0100 - val_accuracy: 0
Epoch 3/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9969 - loss: 0.0097 - val_accuracy: 0.
Epoch 4/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9971 - loss: 0.0090 - val_accuracy: 0
Epoch 5/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9974 - loss: 0.0074 - val_accuracy: 0.
Epoch 6/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9976 - loss: 0.0079 - val_accuracy: 0.
Epoch 7/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9975 - loss: 0.0073 - val_accuracy: 0.
Epoch 8/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9965 - loss: 0.0106 - val_accuracy: 0
Epoch 9/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.9975 - loss: 0.0070 - val_accuracy: 0
Epoch 10/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - accuracy: 0.9981 - loss: 0.0060 - val_accuracy: 0
Epoch 11/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - accuracy: 0.9970 - loss: 0.0080 - val_accuracy: 0
Epoch 12/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 21s 8ms/step - accuracy: 0.9967 - loss: 0.0102 - val_accuracy: 0
Epoch 13/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 22s 9ms/step - accuracy: 0.9980 - loss: 0.0065 - val_accuracy: 0
Epoch 14/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.9984 - loss: 0.0052 - val_accuracy: 0
Epoch 15/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 4ms/step - accuracy: 0.9972 - loss: 0.0090 - val_accuracy: 0
```

```
y_prob = model.predict(X_test) # returns a 2D Matrix, where each row coltains 10columns, each entit
Y_pred = y_prob.argmax(axis= 1) # return ns the index with the most prob for each row
#Coincidentally each index ==  value of output too
```

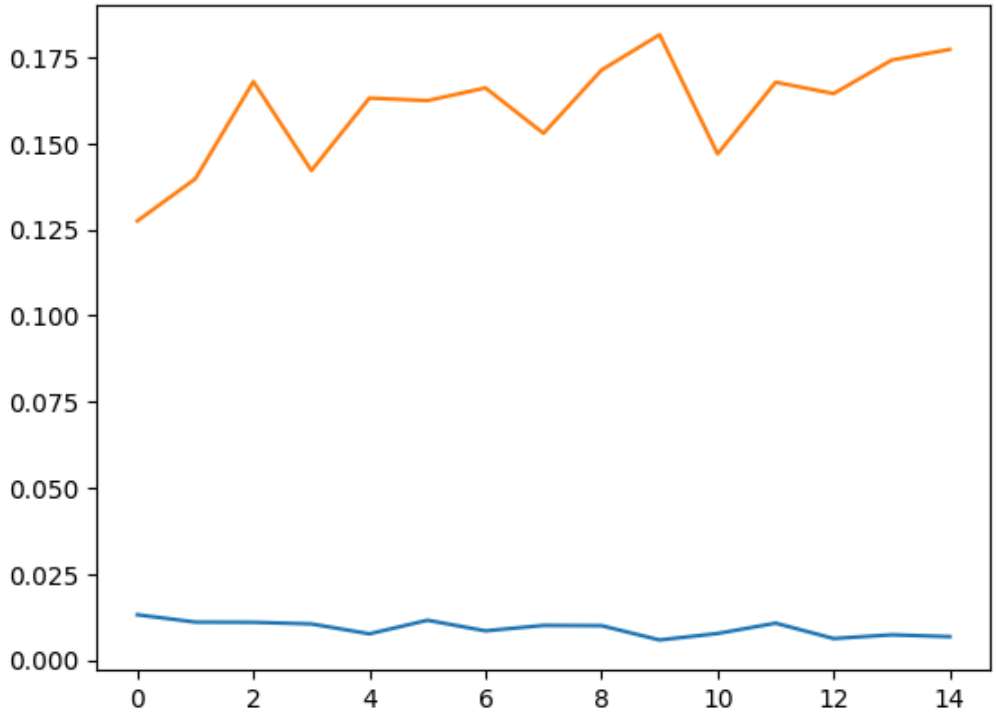
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, Y_pred)
```

```
0.9772
```

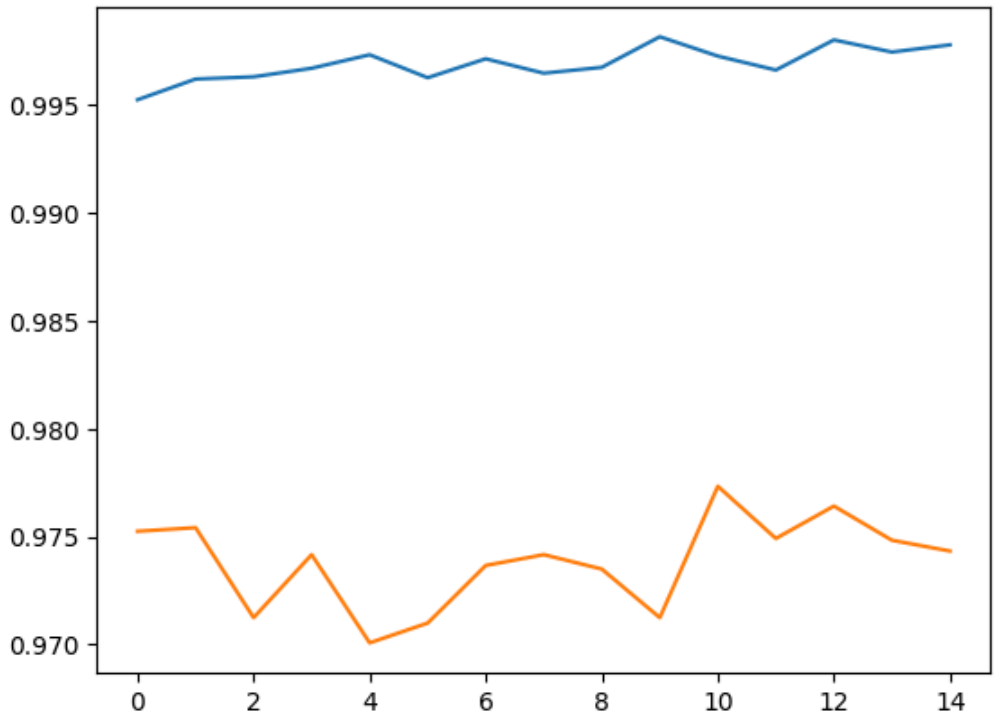
```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x78b658f17e10>]
```



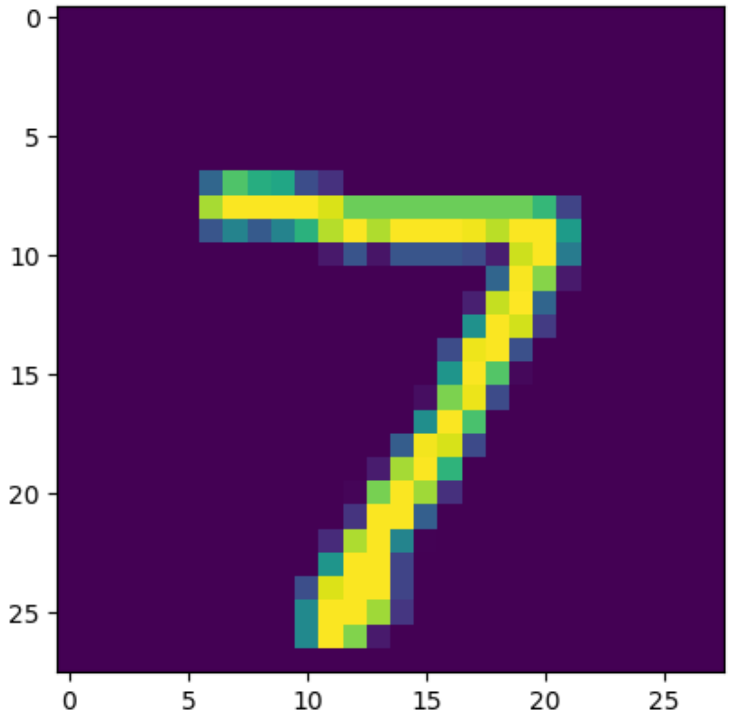

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x78b6fc5ed850>]
```




```
plt.imshow(X_test[0])
model.predict(X_test[0].reshape(1,28,28)).argmax(axis = 1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
array([7])
```



Start coding or generate with AI.