Running this:

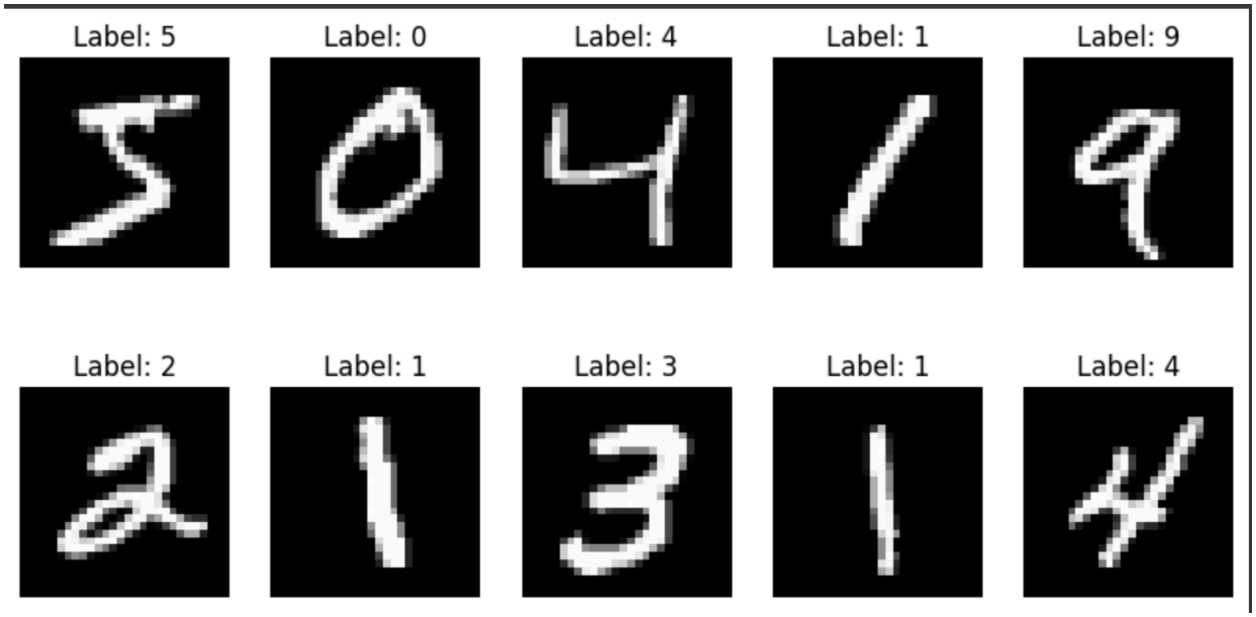
```
import matplotlib.pyplot as plt

IMAGE_WIDTH = 5
IMAGE_HEIGHT = 2

# Function to plot a grid of images
def plot_mnist_images(images, labels, num_images=10):
    plt.figure(figsize=(10, 5))
    for i in range(num_images):
        plt.subplot(2, 5, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(f"Label: {labels[i]}")
        plt.axis('off')
    plt.show()

# Plot 10 images and their labels
plot_mnist_images(x_train, y_train)
```

Results in:



Running:

```
from tensorflow.keras import layers, models

# Build a simple fully connected neural network
model = models.Sequential([
   layers.Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images into
784-dimensional vectors
   layers.Dense(128, activation='relu'),  # First hidden layer with 128
units
   layers.Dense(64, activation='relu'),   # Second hidden layer with 64
units
   layers.Dense(10, activation='softmax') # Output layer with 10 units
(for 10 classes)
])

model.summary()
```

Results in:

Model: "sequential_4"

Layer (type)	Output Shape	Param #
flatten_4 (Flatten)	(None, 784)	0
dense_12 (Dense)	(None, 128)	100,480
dense_13 (Dense)	(None, 64)	8,256
dense_14 (Dense)	(None, 10)	650

Total params: 109,386 (427.29 KB)
Trainable params: 109,386 (427.29 KB)
Non-trainable params: 0 (0.00 B)

Running:

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',  # or
'categorical_crossentropy' if labels are one-hot encoded
              metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10, batch_size=32,
validation_data=(x_test, y_test))

# Example: Access the accuracy and loss from the training process
print(history.history.keys())
```

Results:

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 14s 6ms/step - accuracy: 0.8788 - loss: 0.4155 - val_accuracy: 0.9605 - val_loss: 0.1243
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 8ms/step - accuracy: 0.9655 - loss: 0.1099 - val_accuracy: 0.9701 - val_loss: 0.0938
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 6ms/step - accuracy: 0.9792 - loss: 0.0676 - val_accuracy: 0.9770 - val_loss: 0.0780
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - accuracy: 0.9846 - loss: 0.0477 - val_accuracy: 0.9758 - val_loss: 0.0828
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.9875 - loss: 0.0391 - val_accuracy: 0.9766 - val_loss: 0.0797
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 7ms/step - accuracy: 0.9897 - loss: 0.0318 - val_accuracy: 0.9746 - val_loss: 0.0881
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9923 - loss: 0.0246 - val_accuracy: 0.9769 - val_loss: 0.0820
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9934 - loss: 0.0200 - val_accuracy: 0.9761 - val_loss: 0.0895
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9937 - loss: 0.0187 - val_accuracy: 0.9781 - val_loss: 0.0936
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9953 - loss: 0.0147 - val_accuracy: 0.9753 - val_loss: 0.1022
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Running:

```
import numpy as np

test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy}')

predictions = model.predict(x_test)

# Example: Get predicted class labels
predicted_labels = np.argmax(predictions, axis=1)
```

Results:

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9715 - loss: 0.1209
Test accuracy: 0.9753000140190125
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```