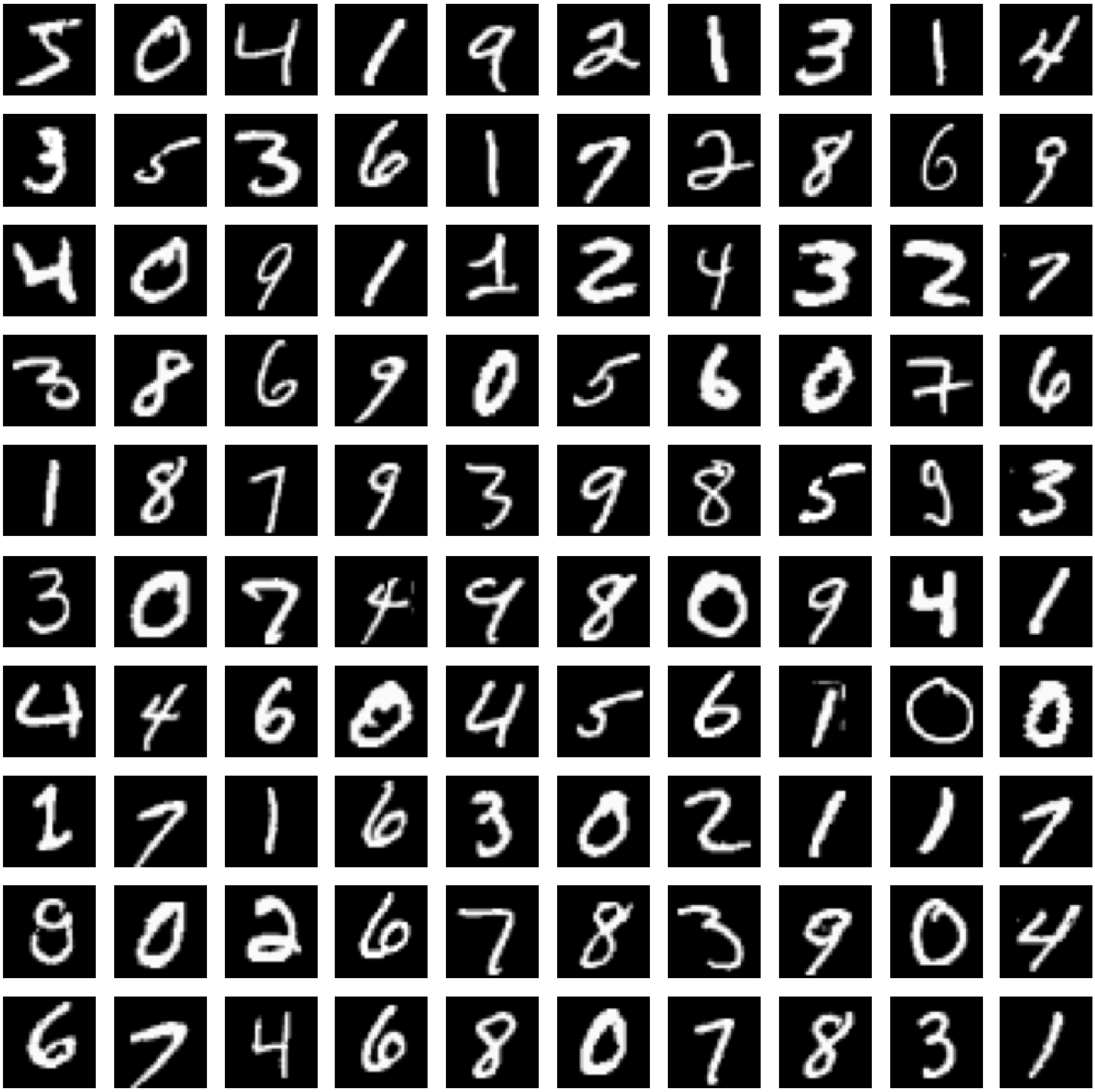
```
# Importing modules
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Activation
import matplotlib.pyplot as plt

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize image pixel values by dividing by 255
gray_scale = 255.0
x_train /= gray_scale
x_test /= gray_scale
print("Feature matrix:", x_train.shape)
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)
```

```
Feature matrix: (60000, 28, 28)
Target matrix: (10000, 28, 28)
Feature matrix: (60000,)
Target matrix: (10000,)
```

```
# Displaying a grid of 100 images from the MNIST dataset
fig, ax = plt.subplots(10, 10, figsize=(10, 10))  # Create 10x10 subplots

k = 0
for i in range(10):
    for j in range(10):
        ax[i][j].imshow(x_train[k].reshape(28, 28), cmap='gray', aspect='auto')
        ax[i][j].axis('off')  # Hide axes for clarity
        k += 1
plt.show()
```



```
# Defining the model
model = Sequential([
    # Reshape 28x28 data to a flat vector of 28*28 (784) elements
    Flatten(input_shape=(28, 28)),

    # Dense layer 1 with 256 neurons and 'sigmoid' activation
    Dense(256, activation='sigmoid'),

    # Dense layer 2 with 128 neurons and 'sigmoid' activation
    Dense(128, activation='sigmoid'),

    # Output layer with 10 neurons (for 10 classes) and 'softmax' activation
    Dense(10, activation='softmax')
])
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_3 (Dense)	(None, 256)	200,960
dense_4 (Dense)	(None, 128)	32,896
dense_5 (Dense)	(None, 10)	1,290

Total params: 235,146 (918.54 KB)
Trainable params: 235,146 (918.54 KB)

```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10,
batch_size=2000,
validation_split=0.2)
```

```
Epoch 1/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 58ms/step - accuracy: 0.2293 - loss: 2.2531 - val_accura
Epoch 2/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.6783 - loss: 1.5952 - val_accura
Epoch 3/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.7784 - loss: 1.0094 - val_accura
Epoch 4/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.8391 - loss: 0.6985 - val_accura
Epoch 5/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.8758 - loss: 0.5288 - val_accura
Epoch 6/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 81ms/step - accuracy: 0.8907 - loss: 0.4394 - val_accura
Epoch 7/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - accuracy: 0.8995 - loss: 0.3800 - val_accura
Epoch 8/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9070 - loss: 0.3467 - val_accura
Epoch 9/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9160 - loss: 0.3143 - val_accura
Epoch 10/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.9193 - loss: 0.2944 - val_accura
<keras.src.callbacks.history.History at 0x7c621dae7a60>
```

```
results = model.evaluate(x_test, y_test, verbose = 0)
print('test loss, test acc:', results)
```

```
test loss, test acc: [0.2753802537918091, 0.9239000082015991]
```