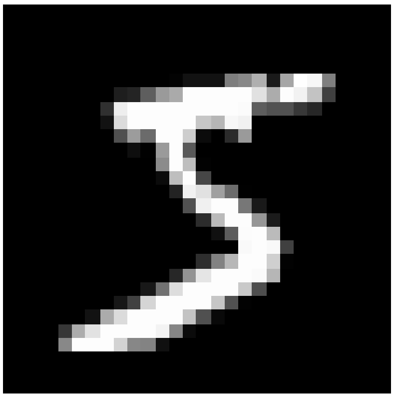
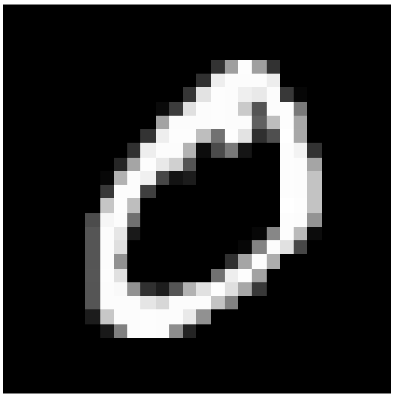
In [14]:
```python
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib as plt
```

In [15]:
```python
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

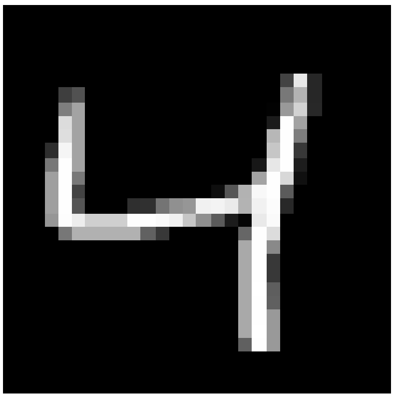
In [20]:
```python
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# Define a function to plot an image
def plot_image(image, figsize=(3, 3)):  # Adjust figsize for desired image size
    plt.figure(figsize=figsize)
    plt.imshow(image.reshape(28, 28), cmap='gray')
    plt.axis('off')
    plt.show()

# Visualize some training images (modify range for more/less images)
for i in range(9):
    plot_image(x_train[i])
    print(f"Label: {y_train[i]}")
```



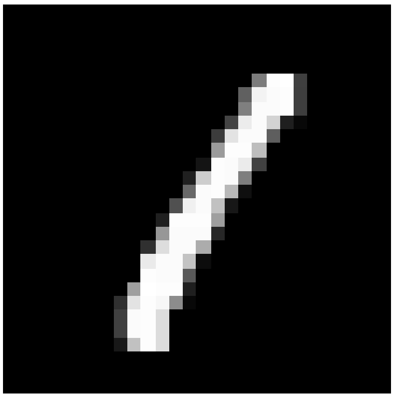
```
Label: 5
```



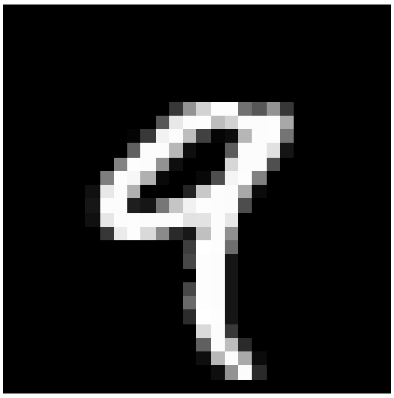
```
Label: 0
```



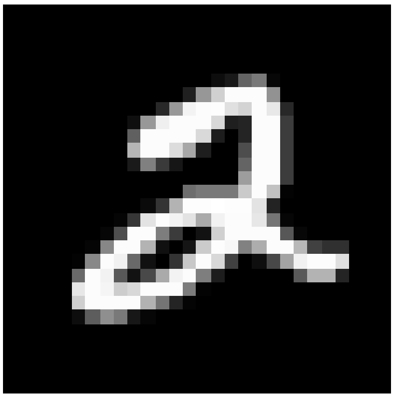
```
Label: 4
```



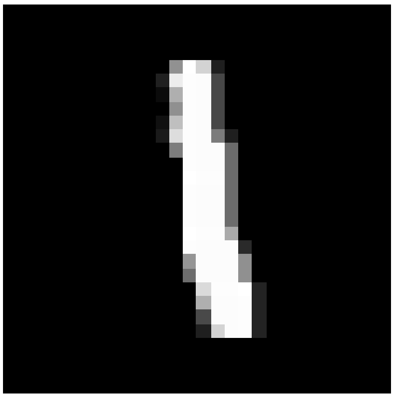
Label: 1



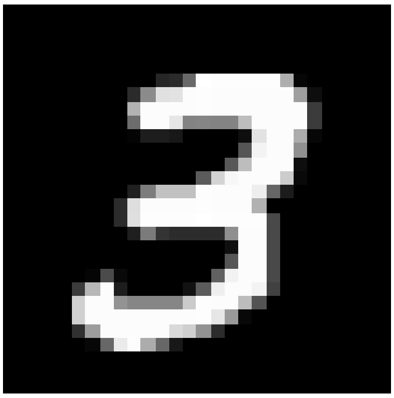
Label: 9



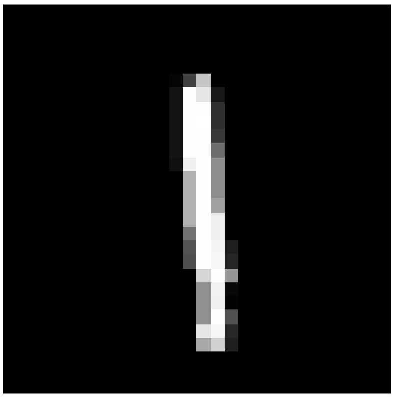
Label: 2



Label: 1



Label: 3



Label: 1

In [3]:
```
# Reshape for CNN input (28x28 pixels and 1 color channel)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# Normalize pixel values to [0, 1] range
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

In [4]:
```
# Convert labels to one-hot vectors for multiclass classification
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [5]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')  # 10 output units for 10 digits
])
```

In [6]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 75s 37ms/step - loss: 0.1320 - accuracy: 0.9594 - val_loss: 0.0425 -
val_accuracy: 0.9867
Epoch 2/10
1875/1875 [==============================] - 63s 34ms/step - loss: 0.0433 - accuracy: 0.9866 - val_loss: 0.0350 -
val_accuracy: 0.9884
Epoch 3/10
1875/1875 [==============================] - 62s 33ms/step - loss: 0.0295 - accuracy: 0.9905 - val_loss: 0.0347 -
val_accuracy: 0.9881
Epoch 4/10
1875/1875 [==============================] - 64s 34ms/step - loss: 0.0220 - accuracy: 0.9930 - val_loss: 0.0330 -
val_accuracy: 0.9898
Epoch 5/10
1875/1875 [==============================] - 63s 33ms/step - loss: 0.0156 - accuracy: 0.9949 - val_loss: 0.0361 -
val_accuracy: 0.9882
Epoch 6/10
1875/1875 [==============================] - 74s 40ms/step - loss: 0.0128 - accuracy: 0.9956 - val_loss: 0.0416 -
val_accuracy: 0.9882
Epoch 7/10
1875/1875 [==============================] - 66s 35ms/step - loss: 0.0098 - accuracy: 0.9966 - val_loss: 0.0308 -
val_accuracy: 0.9911
Epoch 8/10
1875/1875 [==============================] - 57s 31ms/step - loss: 0.0084 - accuracy: 0.9970 - val_loss: 0.0324 -
val_accuracy: 0.9908
Epoch 9/10
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0068 - accuracy: 0.9977 - val_loss: 0.0327 -
val_accuracy: 0.9920
Epoch 10/10
1875/1875 [==============================] - 57s 31ms/step - loss: 0.0055 - accuracy: 0.9981 - val_loss: 0.0446 -
val_accuracy: 0.9904
```

Out[6]:

```
<keras.callbacks.History at 0x254b748f0a0>
```

In [7]:

```python
y_pred = model.predict(x_test)
```

```
313/313 [==============================] - 3s 8ms/step
```

In [8]:

```python
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[ 978    0    0    0    1    0    0    1    0    0]
 [   0 1132    0    0    0    1    0    0    1    1]
 [   1    0 1019    0    2    0    0    8    2    0]
 [   0    1    0  980    0   24    0    2    3    0]
 [   0    0    0    0  976    0    1    0    2    3]
 [   0    0    0    1    0  889    1    0    1    0]
 [   2    2    0    0    2    4  946    0    2    0]
 [   0    5    0    0    0    0    0 1021    1    1]
 [   2    0    1    0    0    0    0    0  969    2]
 [   0    2    0    0    3    5    0    2    3  994]]
```

In [9]:

```python
import numpy as np

accuracy = np.mean(y_test.argmax(axis=1) == y_pred.argmax(axis=1))
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9904
```

In [10]:

```python
# Make predictions on a small subset of test data
predictions = model.predict(x_test[:10])

# Get the predicted class for each sample (argmax gives the index of maximum value)
predicted_classes = predictions.argmax(axis=1)

# Get the actual labels from the one-hot encoded test data
actual_labels = y_test[:10].argmax(axis=1)

# Print the predicted and actual labels for the first 10 samples
print("Sample Predictions and Actual Labels:")
for i in range(10):
    print(f"Sample {i+1}: Predicted: {predicted_classes[i]}, Actual: {actual_labels[i]}")
```

```
1/1 [==============================] - 0s 86ms/step
Sample Predictions and Actual Labels:
Sample 1: Predicted: 7, Actual: 7
Sample 2: Predicted: 2, Actual: 2
Sample 3: Predicted: 1, Actual: 1
Sample 4: Predicted: 0, Actual: 0
Sample 5: Predicted: 4, Actual: 4
Sample 6: Predicted: 1, Actual: 1
Sample 7: Predicted: 4, Actual: 4
Sample 8: Predicted: 9, Actual: 9
Sample 9: Predicted: 5, Actual: 5
Sample 10: Predicted: 9, Actual: 9
```