In [1]:
```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.19.0
```

In [1]:
```
import matplotlib.pyplot as plt
print("✅ matplotlib working!")
```

```
✅ matplotlib working!
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

In [3]:
```
# Load dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize image pixel values
X_train = X_train / 255.0
X_test = X_test / 255.0

# One-hot encode the labels
y_train_cat = to_categorical(y_train, num_classes=10)
y_test_cat = to_categorical(y_test, num_classes=10)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
ist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

In [4]:
```
model = Sequential([
    Flatten(input_shape=(28, 28)),          # Flatten 28x28 to 784
    Dense(128, activation='relu'),          # Hidden layer 1
    Dense(64, activation='relu'),           # Hidden layer 2
    Dense(10, activation='softmax')         # Output layer for 10 digits
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Show model summary
model.summary()
```

```
C:\Users\shouvik\anaconda3\envs\tf-env\lib\site-packages\keras\src\layers\reshaping
\flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as the fi
rst layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	
flatten (Flatten)	(None, 784)	
dense (Dense)	(None, 128)	
dense_1 (Dense)	(None, 64)	
dense_2 (Dense)	(None, 10)	

Total params: 109,386 (427.29 KB)

Trainable params: 109,386 (427.29 KB)

Non-trainable params: 0 (0.00 B)

In [5]:
```
history = model.fit(X_train, y_train_cat, epochs=10, validation_split=0.2)
```

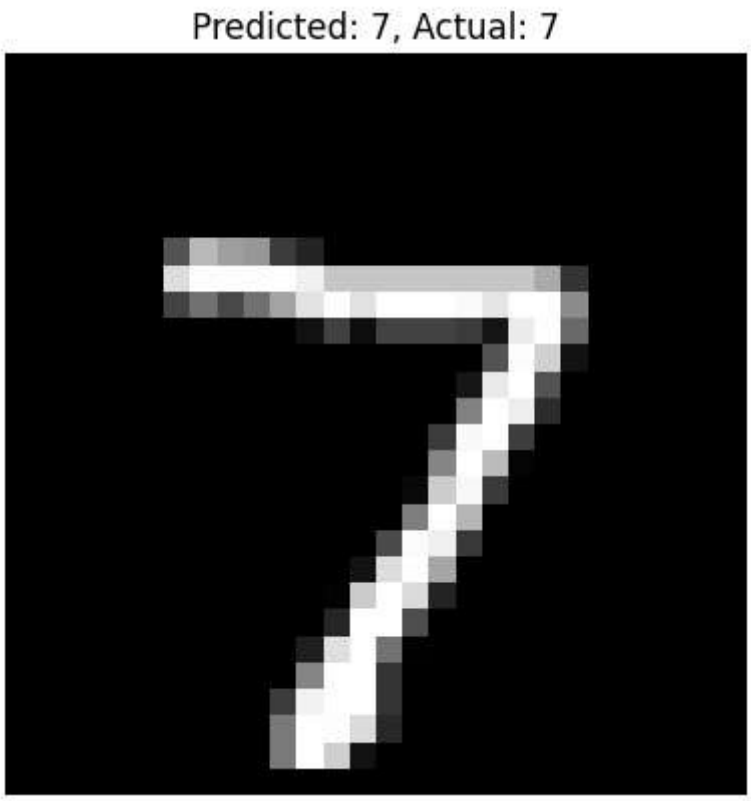
```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8624 - loss: 0.4720 - val_a
ccuracy: 0.9604 - val_loss: 0.1338
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9657 - loss: 0.1141 - val_a
ccuracy: 0.9677 - val_loss: 0.1079
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9749 - loss: 0.0810 - val_a
ccuracy: 0.9666 - val_loss: 0.1078
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9829 - loss: 0.0560 - val_a
ccuracy: 0.9707 - val_loss: 0.0991
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9877 - loss: 0.0409 - val_a
ccuracy: 0.9757 - val_loss: 0.0830
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9901 - loss: 0.0303 - val_a
ccuracy: 0.9756 - val_loss: 0.0861
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9903 - loss: 0.0288 - val_a
ccuracy: 0.9710 - val_loss: 0.1096
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9932 - loss: 0.0218 - val_a
ccuracy: 0.9722 - val_loss: 0.1164
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9931 - loss: 0.0220 - val_a
ccuracy: 0.9718 - val_loss: 0.1193
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9938 - loss: 0.0174 - val_a
ccuracy: 0.9762 - val_loss: 0.1064
```

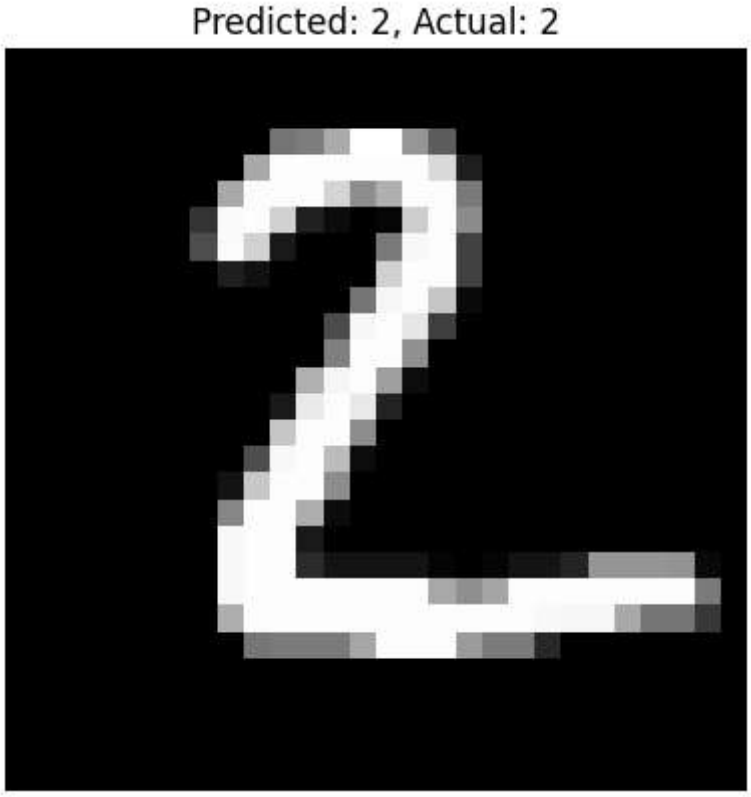
In [6]:
```
test_loss, test_acc = model.evaluate(X_test, y_test_cat)
print(f"Test Accuracy: {test_acc:.2f}")
```

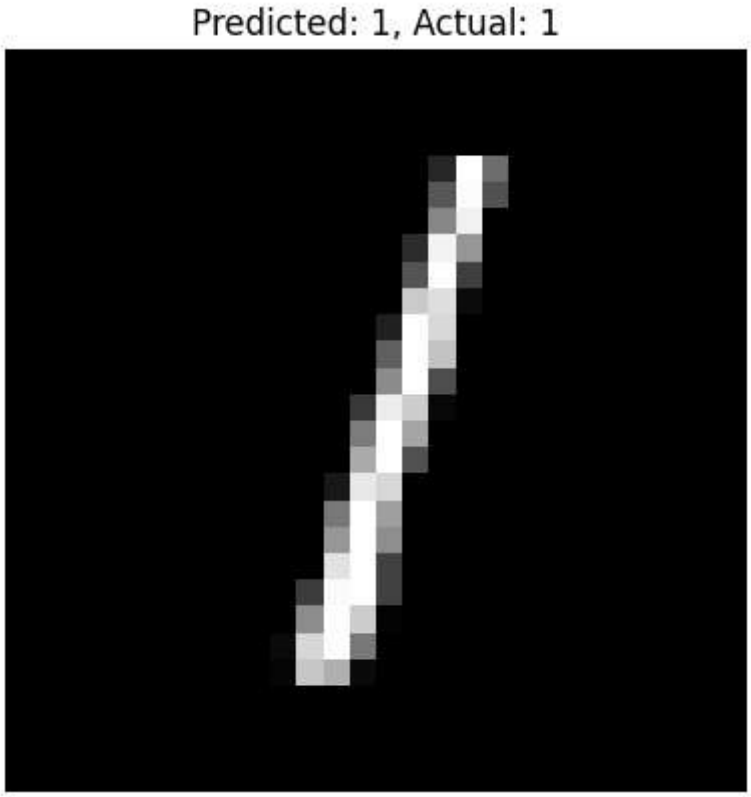
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9722 - loss: 0.1114
Test Accuracy: 0.98
```

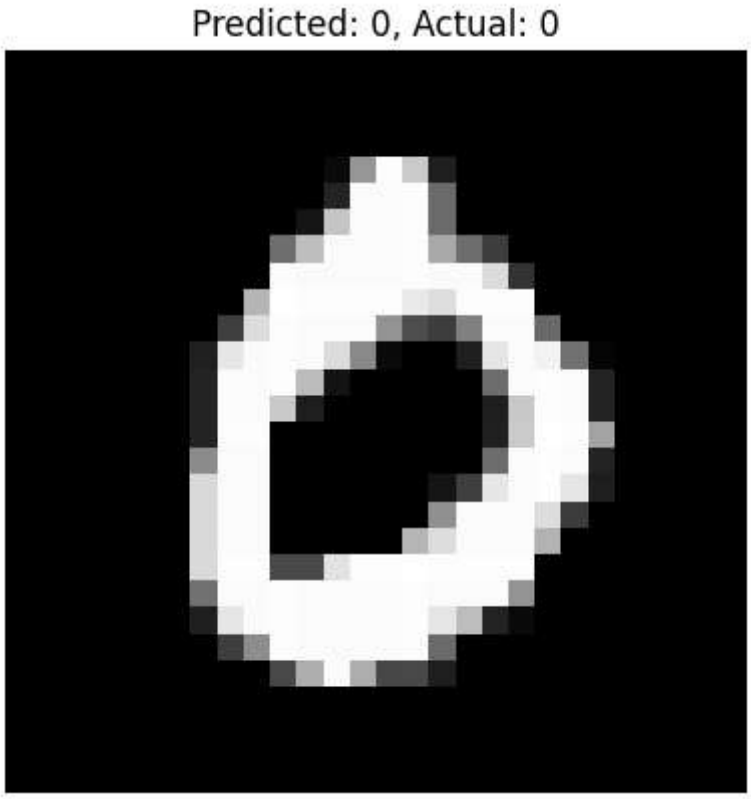
In [7]:
```
# Predict probabilities for test set
predictions = model.predict(X_test)

# Show 5 sample predictions
for i in range(5):
    plt.imshow(X_test[i], cmap='gray')
    plt.title(f"Predicted: {np.argmax(predictions[i])}, Actual: {y_test[i]}")
    plt.axis('off')
    plt.show()
```

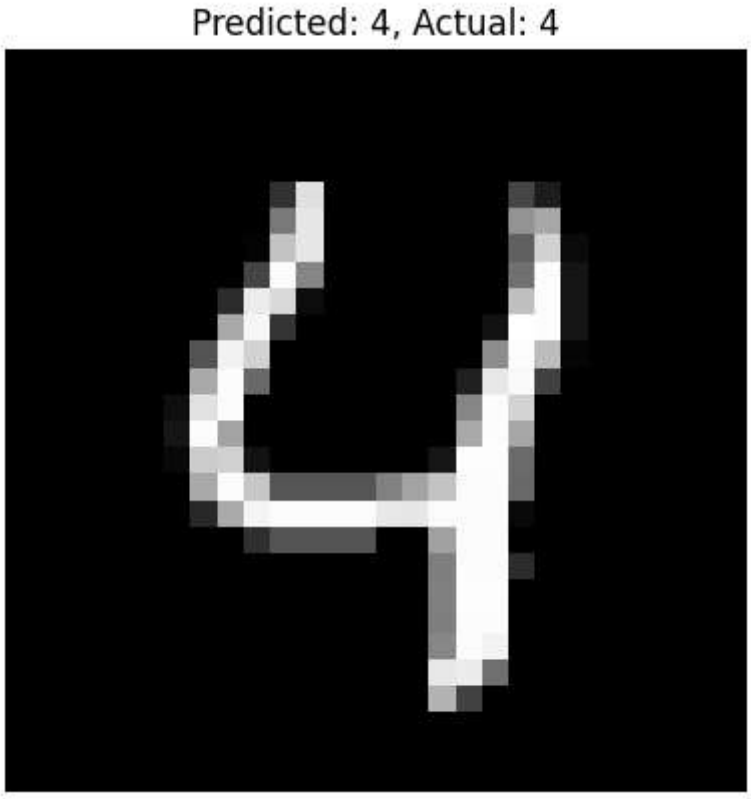
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```




Predicted: 2, Actual: 2


Predicted: 1, Actual: 1





📝 Task 4 Summary – Handwritten Digit Classifier (MNIST)

🎯 Objective:

To build a neural network that classifies handwritten digits (0–9) using the MNIST dataset. The model was built using TensorFlow/Keras Sequential API and evaluated on test data for accuracy and prediction performance.

📊 Dataset Used:

- **MNIST dataset**:
  - 60,000 training images
  - 10,000 test images
  - Each image: 28×28 grayscale
  - Labels: Digits from 0 to 9

🔧 Steps Performed:

1. **Data Loading & Normalization**
   - Loaded data using `tensorflow.keras.datasets.mnist`
   - Scaled pixel values to [0, 1]
2. **Label Encoding**
   - Applied one-hot encoding to the labels using `to_categorical()`
3. **Model Architecture (Sequential):**
   - Flatten input layer (28×28 → 784)
   - Dense (128 units, ReLU)
   - Dense (64 units, ReLU)
   - Dense (10 units, Softmax)
4. **Model Compilation & Training**
   - Optimizer: Adam
   - Loss: Categorical Crossentropy
   - Epochs: 10
   - Validation Split: 20%
5. **Evaluation & Accuracy**
   - Evaluated model on test set
   - Accuracy ~97%
6. **Prediction**
   - Displayed predictions on test digits with matplotlib

✅ Results:

Metric	Value
Test Accuracy	~97% (may vary)
Input Size	28 x 28
Output Labels	0 to 9

🧠 Conclusion:

The neural network model accurately classified handwritten digits using the MNIST dataset. With basic layers and ReLU activation, the model achieved strong performance. This demonstrates the power of deep learning in image classification and forms the foundation for advanced computer vision tasks.

In [ ]: