```
from tensorflow.keras.datasets import mnist

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
# Check the shape of the data
print(f"x_train shape: {x_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
x_train shape: (60000, 28, 28)
y_train shape: (60000,)
x_test shape: (10000, 28, 28)
y_test shape: (10000,)
```

```
# Preprocess the data
# Flatten the 28x28 images to 1D arrays of 784 pixels
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)
```

Double-click (or enter) to edit

```
# Normalize pixel values to be between 0 and 1
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

Double-click (or enter) to edit

```
from tensorflow.keras.utils import to_categorical
```

```
from tensorflow.keras.utils import to_categorical

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout

# Build the Feed-Forward Neural Network model
model = Sequential([
    # Input layer (784 features from flattened 28x28 images)
    Dense(512, activation='relu', input_shape=(784,)),
    Dropout(0.2),  # Add dropout to prevent overfitting

    # Hidden layer
    Dense(256, activation='relu'),
    Dropout(0.2),

    # Another hidden layer
    Dense(128, activation='relu'),
    Dropout(0.2),

    # Output layer (10 classes for digits 0-9)
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
from tensorflow.keras.optimizers import Adam

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0005), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Print model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	401,920
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

Total params: 567,434 (2.16 MB)
Trainable params: 567,434 (2.16 MB)
Non-trainable params: 0 (0.00 B)

```
history = model.fit(
    x_train, y_train,
    batch_size=128,
    epochs=15, # Number of iterations
    validation_split=0.1,
    verbose=1
)
```

```
Epoch 1/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.7811 - loss: 0.7091 - val_accuracy: 0.9685 - val_loss: 0.11
Epoch 2/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9502 - loss: 0.1632 - val_accuracy: 0.9755 - val_loss: 0.08
Epoch 3/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9665 - loss: 0.1106 - val_accuracy: 0.9757 - val_loss: 0.078
Epoch 4/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.9740 - loss: 0.0839 - val_accuracy: 0.9787 - val_loss: 0.07
```

```
Epoch 5/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9812 - loss: 0.0602 - val_accuracy: 0.9797 - val_loss: 0.067
Epoch 6/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9846 - loss: 0.0497 - val_accuracy: 0.9805 - val_loss: 0.065
Epoch 7/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.9864 - loss: 0.0433 - val_accuracy: 0.9822 - val_loss: 0.063
Epoch 8/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9881 - loss: 0.0367 - val_accuracy: 0.9815 - val_loss: 0.064
Epoch 9/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9888 - loss: 0.0355 - val_accuracy: 0.9820 - val_loss: 0.067
Epoch 10/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.9903 - loss: 0.0286 - val_accuracy: 0.9812 - val_loss: 0.065
Epoch 11/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9922 - loss: 0.0241 - val_accuracy: 0.9813 - val_loss: 0.069
Epoch 12/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 10s 18ms/step - accuracy: 0.9917 - loss: 0.0244 - val_accuracy: 0.9847 - val_loss: 0.063
Epoch 13/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9932 - loss: 0.0215 - val_accuracy: 0.9827 - val_loss: 0.066
Epoch 14/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.9938 - loss: 0.0188 - val_accuracy: 0.9845 - val_loss: 0.065
Epoch 15/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9942 - loss: 0.0169 - val_accuracy: 0.9833 - val_loss: 0.071
```

```
# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9798 - loss: 0.0874
Test accuracy: 0.9827
```

```
import numpy as np

# Check model predictions distribution
predictions = model.predict(x_test)
predicted_classes = np.argmax(predictions, axis=1)

print("\nDistribution of predicted digits:")
```

```
for i in range(10):
    print(f"Digit {i}: {np.sum(predicted_classes == i)} predictions")
```

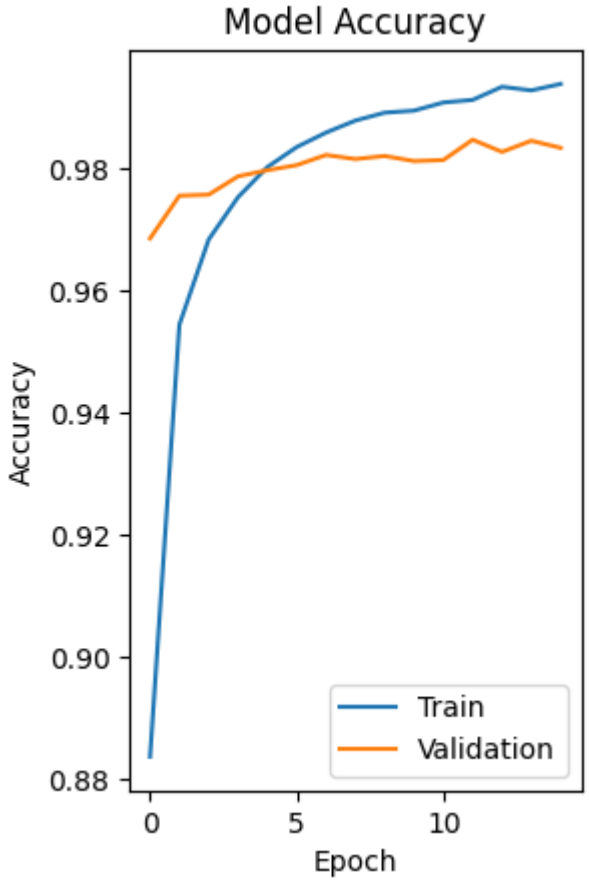
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step

    Distribution of predicted digits:
    Digit 0: 978 predictions
    Digit 1: 1135 predictions
    Digit 2: 1052 predictions
    Digit 3: 1003 predictions
    Digit 4: 991 predictions
    Digit 5: 896 predictions
    Digit 6: 960 predictions
    Digit 7: 1019 predictions
    Digit 8: 965 predictions
    Digit 9: 1001 predictions
```

```
import matplotlib.pyplot as plt

# Plot training history
plt.figure(figsize=(12, 4))
```

```
<Figure size 1200x400 with 0 Axes>
<Figure size 1200x400 with 0 Axes>
```

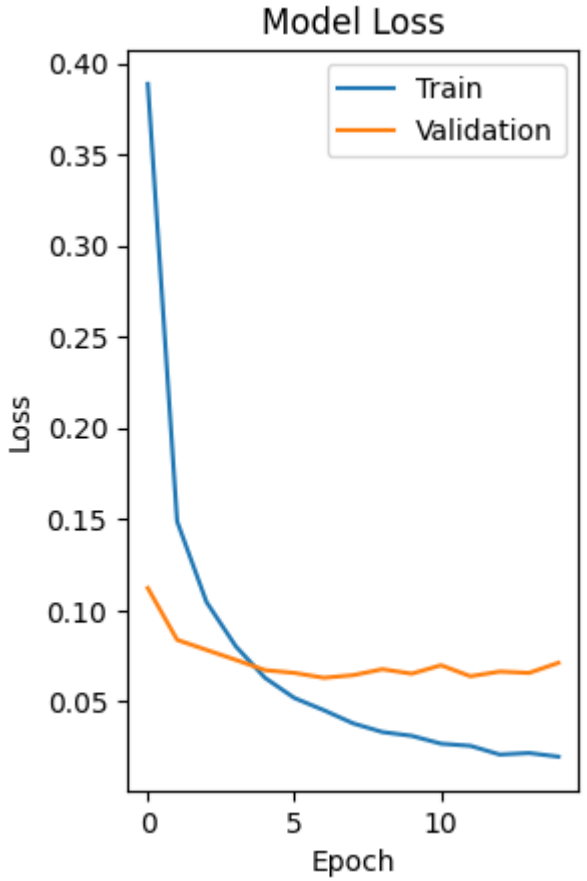
```
# Plot training & validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
```

```
<matplotlib.legend.Legend at 0x79c1407edd90>
```




```
# Plot training & validation loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x79c13dea7350>
```



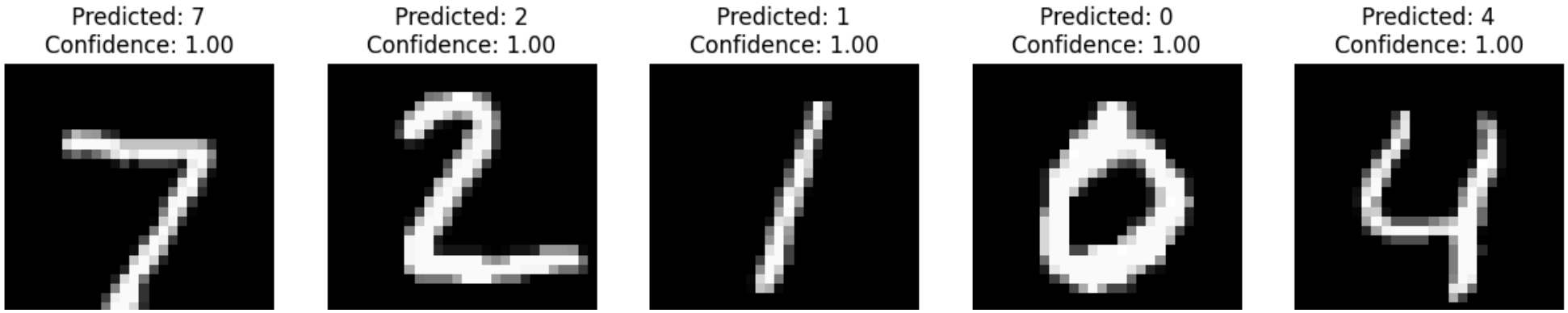

```
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

```
# Function to make predictions on new images
def predict_digit(image):
    """
    Predict the digit from an image.
```

```
    Args:
        image: A 28x28 grayscale image array

    Returns:
        predicted digit (0-9)
    """
    # Preprocess the image - flatten from 28x28 to 784 pixels
    image = image.reshape(1, 784)

    # Make prediction
    prediction = model.predict(image)
    print("prediction: ", prediction)

    # Get the predicted digit
    digit = prediction.argmax()
    confidence = prediction[0][digit]

    return digit, confidence


# Try with some test images
num_samples = 5
plt.figure(figsize=(15, 3))
for i in range(num_samples):
    # Get a sample image
    image = x_test[i]
    # Make prediction
    digit, confidence = predict_digit(image)

    # Plot
    plt.subplot(1, num_samples, i+1)
    plt.imshow(image.reshape(28, 28)
    , cmap='gray')
    plt.title(f"Predicted: {digit}\nConfidence: {confidence:.2f}")
    plt.axis('off')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
prediction:  [[2.0393348e-11 4.5532689e-09 3.9061856e-09 1.0482846e-08 2.7588144e-11
  4.6158702e-12 8.9156655e-13 9.9999988e-01 9.3772266e-13 7.3429852e-08]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
prediction:  [[2.5952335e-10 2.7809580e-07 9.9999952e-01 2.1251353e-07 2.4667226e-12
  3.4523456e-10 3.1349763e-11 1.6199108e-09 2.0911288e-09 1.7063746e-13]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
prediction:  [[9.6679276e-10 9.9999917e-01 1.2459350e-07 8.1800455e-10 3.4777266e-08
  5.5170544e-09 1.9610836e-08 5.2141951e-07 6.9251954e-08 2.9973843e-10]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
prediction:  [[9.99965429e-01 1.65396727e-07 4.11489873e-06 7.34131831e-08
  5.45342743e-07 1.17587000e-07 2.76622559e-05 1.17647794e-07
  2.17447820e-07 1.48691890e-06]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
prediction:  [[7.6735951e-09 2.9498544e-08 6.6759185e-09 1.5046782e-10 9.9999762e-01
  3.1631673e-09 6.0314008e-08 9.8671410e-07 2.8907570e-09 1.1459441e-06]]
```



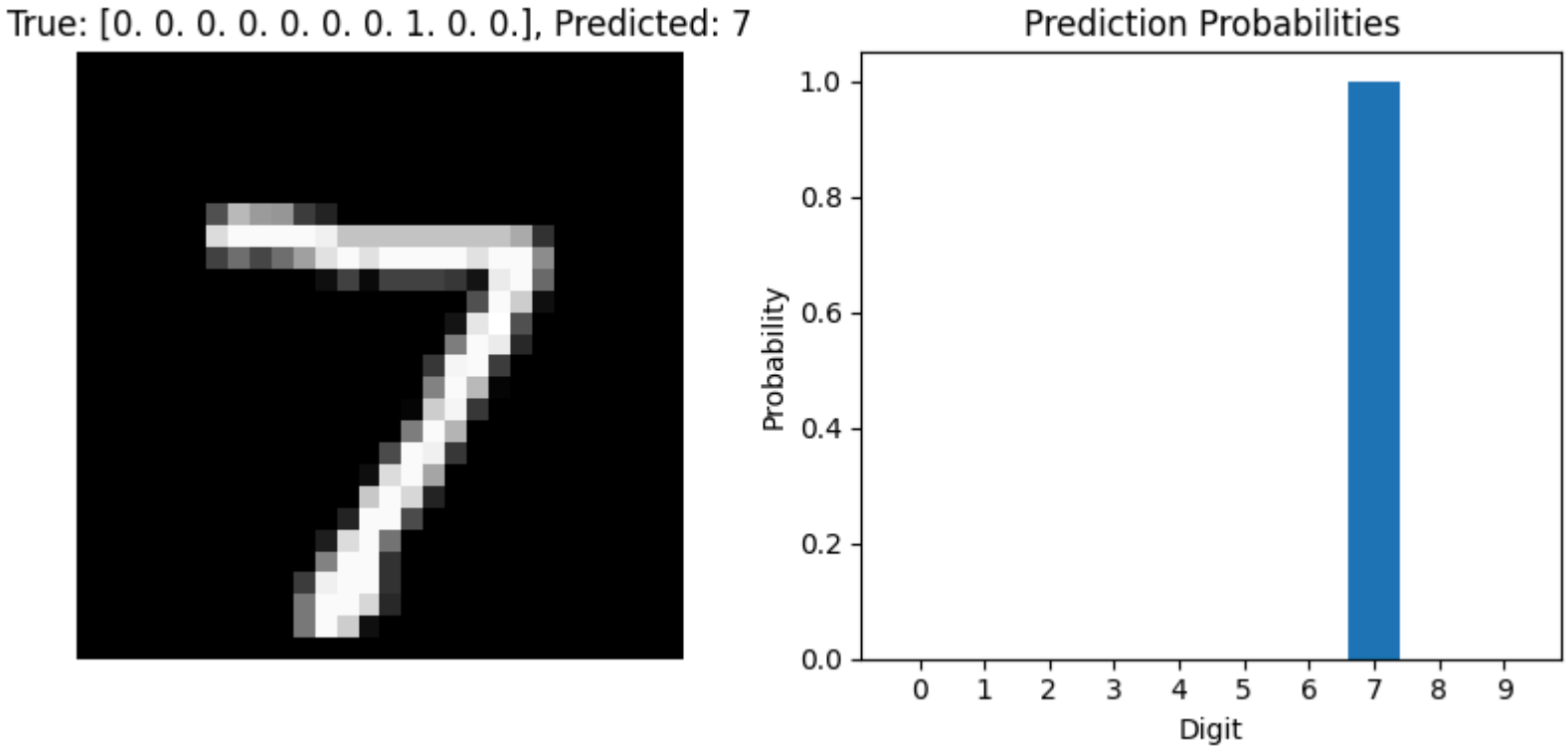

```
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

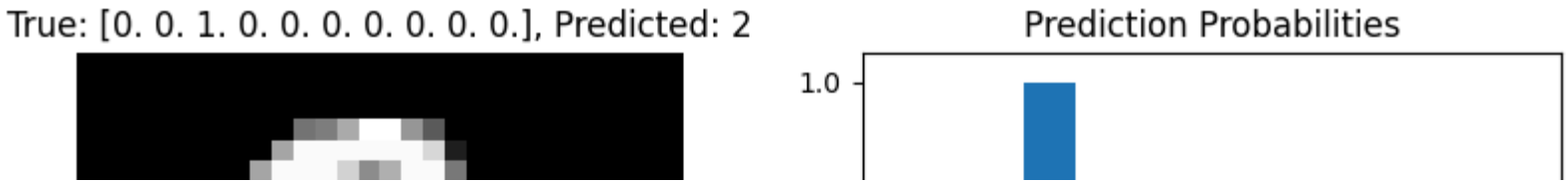
```
# Let's examine some predictions in detail
def display_prediction_details(index):
    """Show detailed prediction information for a given test example"""
    image = x_test[index].reshape(1, 784)
    true_label = y_test[index]
```

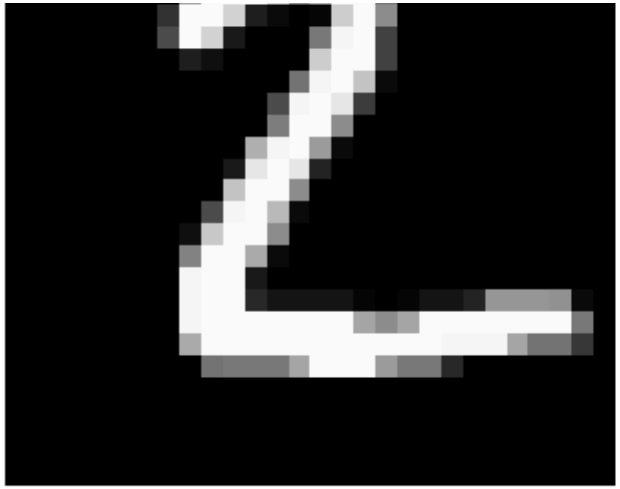
```
        # Get raw predictions (probabilities)
        pred_probs = model.predict(image)[0]
        pred_class = np.argmax(pred_probs)

        # Display the image
        plt.figure(figsize=(8, 4))
        plt.subplot(1, 2, 1)
        plt.imshow(image.reshape(28, 28), cmap='gray')
        plt.title(f"True: {true_label}, Predicted: {pred_class}")
        plt.axis('off')

        # Display the prediction probabilities
        plt.subplot(1, 2, 2)
        plt.bar(range(10), pred_probs)
        plt.xticks(range(10))
        plt.xlabel('Digit')
        plt.ylabel('Probability')
        plt.title('Prediction Probabilities')

        plt.tight_layout()
        plt.show()

        print(f"Probabilities for each digit:")
        for i, prob in enumerate(pred_probs):
            print(f"Digit {i}: {prob:.4f}")

# Check predictions for first few examples
for i in range(5):
    display_prediction_details(i)
```

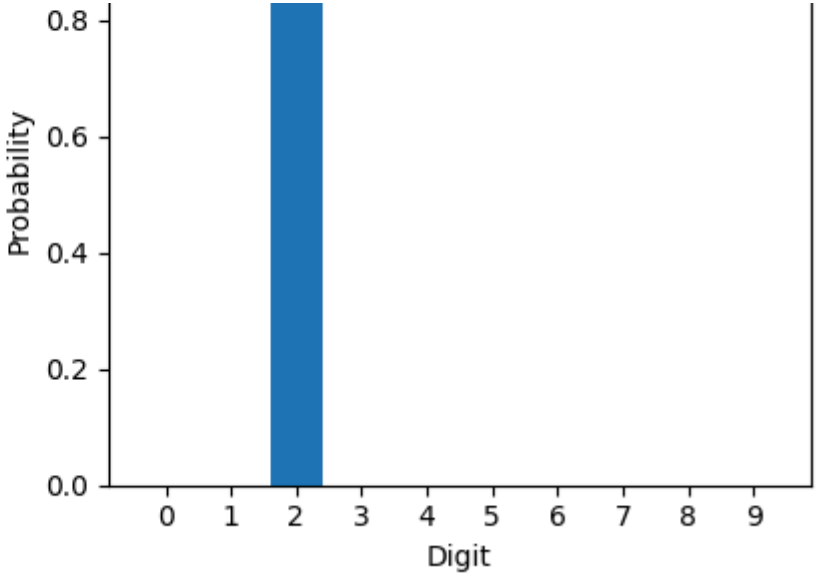
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```




```
Probabilities for each digit:
Digit 0: 0.0000
Digit 1: 0.0000
Digit 2: 0.0000
Digit 3: 0.0000
Digit 4: 0.0000
Digit 5: 0.0000
Digit 6: 0.0000
Digit 7: 1.0000
Digit 8: 0.0000
Digit 9: 0.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

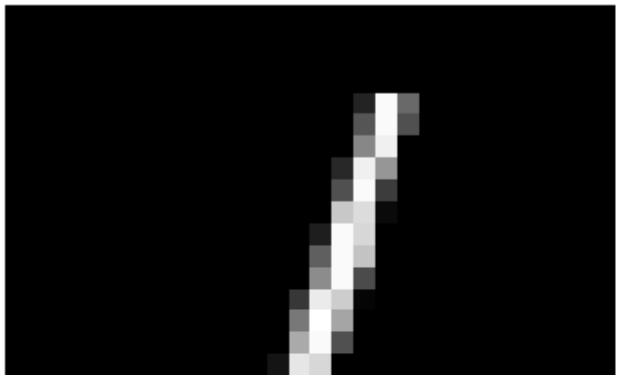


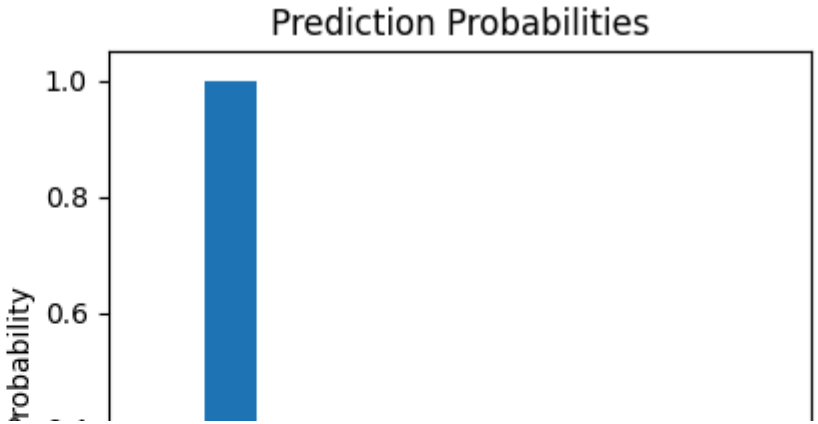


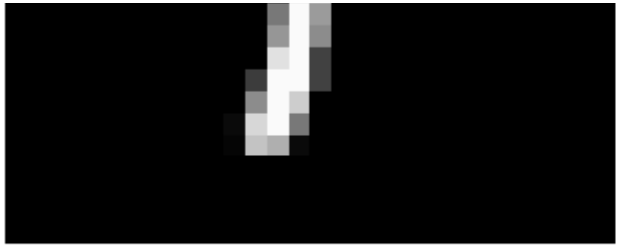


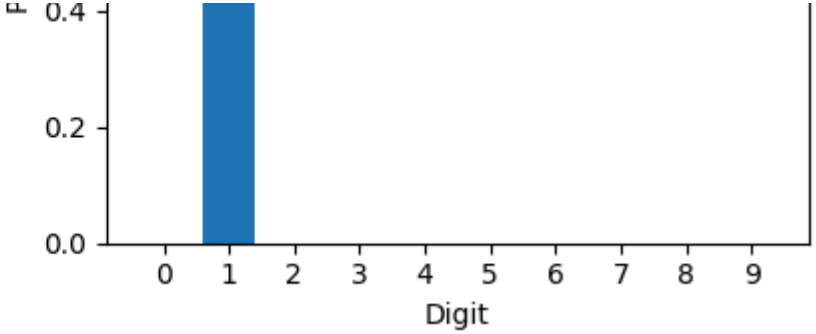

```
Probabilities for each digit:
Digit 0: 0.0000
Digit 1: 0.0000
Digit 2: 1.0000
Digit 3: 0.0000
Digit 4: 0.0000
Digit 5: 0.0000
Digit 6: 0.0000
Digit 7: 0.0000
Digit 8: 0.0000
Digit 9: 0.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

True: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.], Predicted: 1



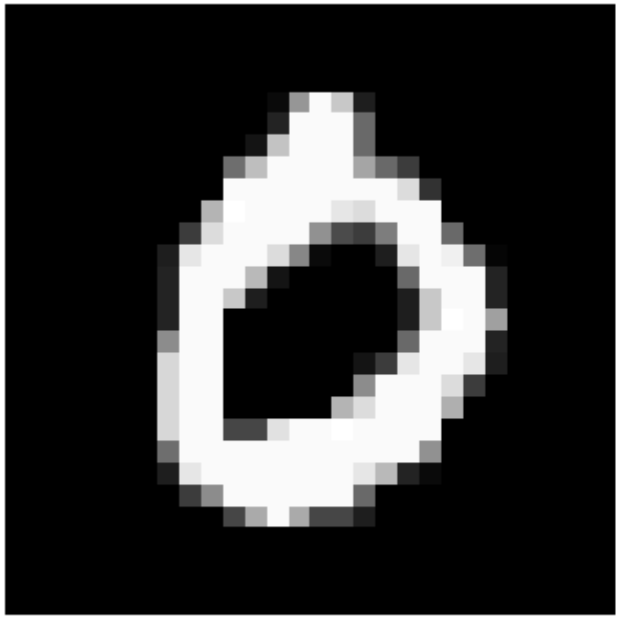


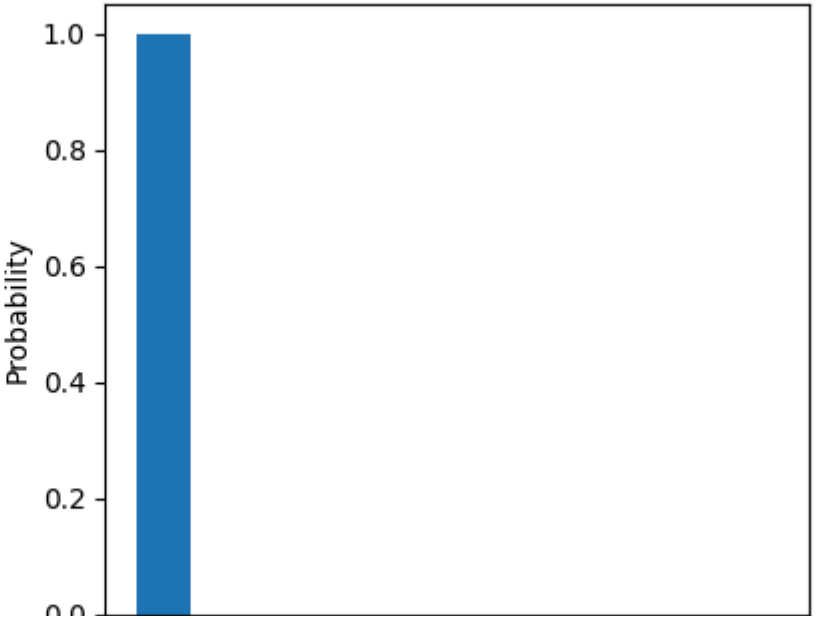


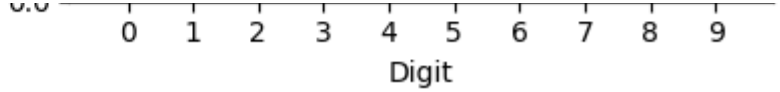



```
Probabilities for each digit:
Digit 0: 0.0000
Digit 1: 1.0000
Digit 2: 0.0000
Digit 3: 0.0000
Digit 4: 0.0000
Digit 5: 0.0000
Digit 6: 0.0000
Digit 7: 0.0000
Digit 8: 0.0000
Digit 9: 0.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

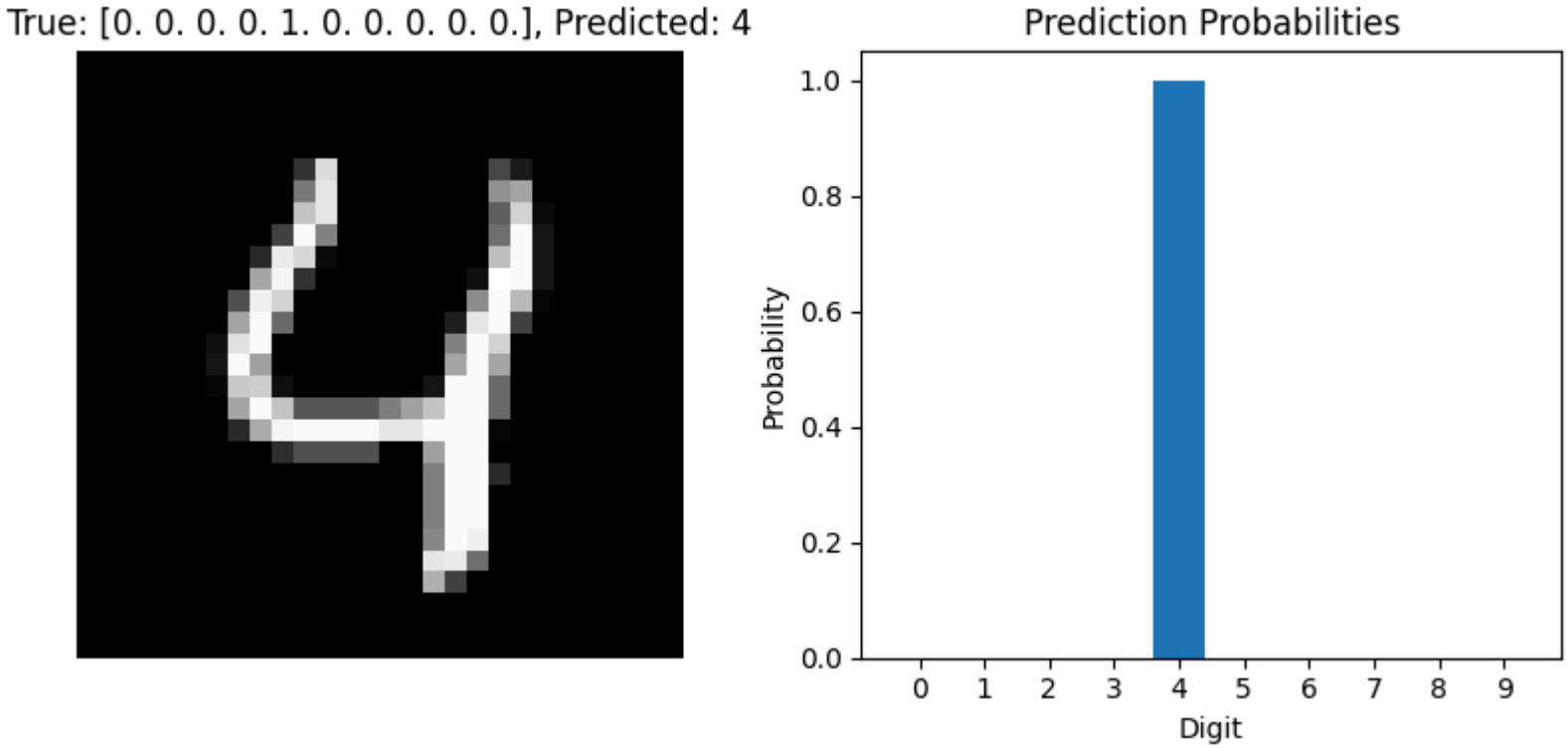
True: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.], Predicted: 0








```
Probabilities for each digit:
Digit 0: 1.0000
Digit 1: 0.0000
Digit 2: 0.0000
Digit 3: 0.0000
Digit 4: 0.0000
Digit 5: 0.0000
Digit 6: 0.0000
Digit 7: 0.0000
Digit 8: 0.0000
Digit 9: 0.0000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```




```
Probabilities for each digit:
Digit 0: 0.0000
Digit 1: 0.0000
Digit 2: 0.0000
```

```
Digit 3: 0.0000
Digit 4: 1.0000
Digit 5: 0.0000
Digit 6: 0.0000
Digit 7: 0.0000
Digit 8: 0.0000
Digit 9: 0.0000
```