```
Start coding or generate with AI.
```

Task

Implement a simple neural network for image classification using the MNIST dataset.

Load and prepare data

Subtask:

Load the MNIST dataset and preprocess the image data (normalize pixel values, reshape images).

Reasoning: Load the MNIST dataset, normalize the pixel values, reshape the images, and one-hot encode the labels using Keras.

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

# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the pixel values
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

# Reshape the images
train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))

# One-hot encode the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

print("MNIST dataset loaded and preprocessed.")
print("Training images shape:", train_images.shape)
print("Testing images shape:", test_images.shape)
print("Training labels shape:", train_labels.shape)
print("Testing labels shape:", test_labels.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
MNIST dataset loaded and preprocessed.
Training images shape: (60000, 784)
Testing images shape: (10000, 784)
Training labels shape: (60000, 10)
Testing labels shape: (10000, 10)
```

Build the neural network model

Subtask:

Define a simple neural network architecture using a library like TensorFlow or PyTorch.

Reasoning: Define a simple neural network architecture using TensorFlow Keras as instructed.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the neural network architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do n
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	401,920
dense_1 (Dense)	(None, 10)	5,130

Total params: 407,050 (1.55 MB)
Trainable params: 407,050 (1.55 MB)
Non-trainable params: 0 (0.00 B)

Compile the model

Subtask:

Configure the model for training by specifying the optimizer, loss function, and metrics.

Reasoning: Configure the model for training by compiling it with the specified optimizer, loss function, and metrics.

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

print("Model compiled successfully.")
```

```
Model compiled successfully.
```

Train the model

Subtask:

Train the neural network on the training data.

Reasoning: Train the compiled neural network model on the preprocessed training data with specified epochs and batch size, and store the training history.

```
# Train the model
history = model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.8730 - loss: 0.4386
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.9665 - loss: 0.1153
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.9787 - loss: 0.0713
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9847 - loss: 0.0506
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9895 - loss: 0.0377
```

Evaluate the model

Subtask:

Evaluate the trained model's performance on the test data.

Reasoning: Evaluate the trained model on the test data and print the results.

```
# Evaluate the model on the test data
loss, accuracy = model.evaluate(test_images, test_labels)

# Print the evaluation results
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9749 - loss: 0.0859
Test Loss: 0.0717
Test Accuracy: 0.9788
```

Make predictions

Subtask:

Use the trained model to make predictions on new images.

Reasoning: Select a few images from the test set and use the trained model to predict their labels, then get the predicted digit by finding the index of the highest probability.

```
import numpy as np

# Select a few images from the test set (e.g., the first 5 images)
num_images_to_predict = 5
sample_images = test_images[:num_images_to_predict]
sample_labels = test_labels[:num_images_to_predict]

# Use the trained model to make predictions
predictions = model.predict(sample_images)

# Get the predicted digit for each image
predicted_labels = np.argmax(predictions, axis=1)
actual_labels = np.argmax(sample_labels, axis=1)

print("Predictions for the first", num_images_to_predict, "test images:")
for i in range(num_images_to_predict):
    print(f"Image {i+1}: Predicted Digit = {predicted_labels[i]}, Actual Digit = {actual_la
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
Predictions for the first 5 test images:
Image 1: Predicted Digit = 7, Actual Digit = 7
Image 2: Predicted Digit = 2, Actual Digit = 2
Image 3: Predicted Digit = 1, Actual Digit = 1
Image 4: Predicted Digit = 0, Actual Digit = 0
Image 5: Predicted Digit = 4, Actual Digit = 4
```

Visualize results

Subtask:

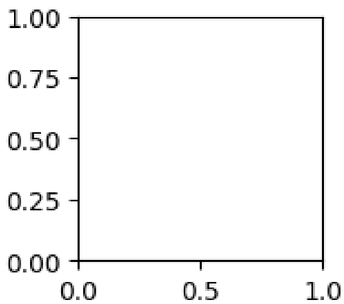
Display some sample images and their predicted labels.

Reasoning: Display the sample images and their predicted and actual labels using matplotlib.

```
import matplotlib.pyplot as plt

# Reshape the original test_images for display
original_test_images = mnist.load_data()[0][1] # Get test images before flattening

plt.figure(figsize=(10, 4))
for i in range(num_images_to_predict):
    plt.subplot(1, num_images_to_predict, i + 1)
    # Reshape the flattened image back to 28x28
    image = original_test_images[i]
    plt.imshow(image, cmap='gray')
    plt.title(f"Pred: {predicted_labels[i]}\nActual: {actual_labels[i]}")
    plt.axis('off')

plt.tight_layout()
```

```
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-1319238308.py in <cell line: 0>()
      9     # Reshape the flattened image back to 28x28
     10     image = original_test_images[i]
---> 11     plt.imshow(image, cmap='gray')
     12     plt.title(f"Pred: {predicted_labels[i]}\nActual: {actual_labels[i]}")
     13     plt.axis('off')

4 frames
/usr/local/lib/python3.12/dist-packages/matplotlib/image.py in _normalize_image_array(A)
    641             A = A.squeeze(-1)  # If just (M, N, 1), assume scalar and apply
colormap.
    642         if not (A.ndim == 2 or A.ndim == 3 and A.shape[-1] in [3, 4]):
--> 643             raise TypeError(f"Invalid shape {A.shape} for image data")
    644         if A.ndim == 3:
    645             # If the input data has values outside the valid range (after

TypeError: Invalid shape () for image data
```

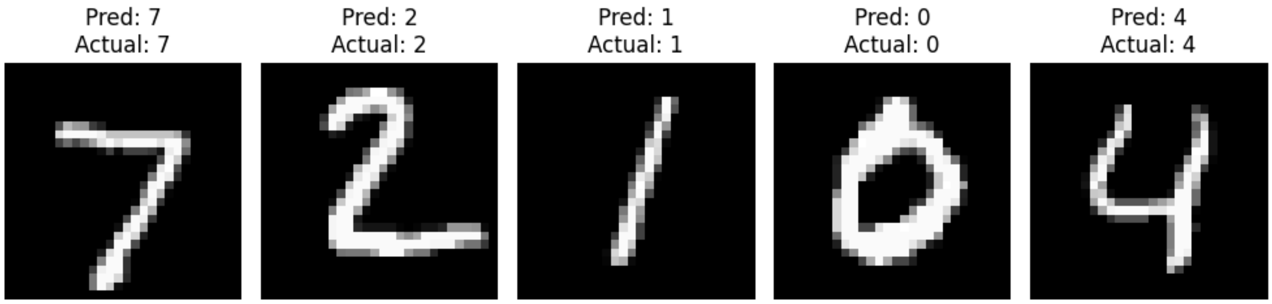


Next steps: Explain error

Reasoning: The error "Invalid shape () for image data" indicates that the image data being passed to `plt.imshow()` is empty or has an incorrect shape. This is likely because accessing `mnist.load_data()[0][1]` does not return the test images in the expected format. The original `test_images` before flattening were stored in a variable during the data loading step. I will use that variable instead.

```
import matplotlib.pyplot as plt

# Use the original test_images variable from the data loading step
(x_train_orig, y_train_orig), (x_test_orig, y_test_orig) = mnist.load_data()


plt.figure(figsize=(10, 4))
for i in range(num_images_to_predict):
    plt.subplot(1, num_images_to_predict, i + 1)
    # Use the original test image before flattening
    image = x_test_orig[i]
    plt.imshow(image, cmap='gray')
```

```
        plt.title(f"Pred: {predicted_labels[i]}\nActual: {actual_labels[i]}")
        plt.axis('off')

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



Summary:

Data Analysis Key Findings

- The MNIST dataset was successfully loaded, and the image data was preprocessed by normalizing pixel values to the range [0, 1] and reshaping the 28x28 images into flattened vectors of size 784. The labels were one-hot encoded.
- A simple sequential neural network model was built with one dense hidden layer of 512 units using the ReLU activation and an output layer of 10 units using the softmax activation.
- The model was compiled using the 'rmsprop' optimizer, 'categorical_crossentropy' loss function, and 'accuracy' as a metric.
- The model was trained on the training data for 5 epochs with a batch size of 128.
- The trained model achieved a Test Loss of 0.0717 and a Test Accuracy of 0.9788 on the test dataset.
- The model successfully made predictions on sample test images, correctly identifying the digits for the first 5 examples shown.

Insights or Next Steps

- The simple neural network architecture and training process resulted in a high accuracy (>97%) on the MNIST test set, indicating good performance on this dataset.
- Further steps could involve exploring more complex architectures (e.g., Convolutional Neural Networks), tuning hyperparameters, or evaluating the model on a larger set of test images to confirm performance consistency.