Convolutional Neural Network (CNN) Mini Project: MNIST Digit Classification

Goal: Build, train, and evaluate a Convolutional Neural Network (CNN) to classify handwritten digits (0-9) using the MNIST dataset.

1. Project Setup and Imports

First, we import the necessary libraries. TensorFlow/Keras is the core deep learning framework, and Matplotlib is used for visualization.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
import os

# Set hyperparameters
NUM_CLASSES = 10
EPOCHS = 8  # Increased epochs for better accuracy (feel free to experiment)
BATCH_SIZE = 128
IMG_SHAPE = (28, 28, 1)

print(f"TensorFlow Version: {tf.__version__}")
# Disable TensorFlow warnings for cleaner output
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

2. Load and Preprocess Data

The MNIST dataset is built into Keras, making it easy to load. We must normalize the pixel values and reshape the images for the CNN.

```
# Load the dataset
print("Loading and preparing MNIST data...")
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# 1. Normalize Images (Scale pixel values from [0-255] to [0.0-1.0])
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# 2. Reshape for CNN Input (Add the channel dimension: 28x28 -> 28x28x1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# 3. One-Hot Encode Labels (Convert integer labels to vectors)
y_train = to_categorical(y_train, NUM_CLASSES)
y_test = to_categorical(y_test, NUM_CLASSES)

print(f"x_train shape after preprocessing: {x_train.shape}")
print(f"y_train shape after preprocessing: {y_train.shape}")
```

3. Visualize Sample Data

A good practice in any ML project is to visually inspect the data. Let's plot the first 9 images from the training set.

```
# Visualize the first 9 images
plt.figure(figsize=(8, 8))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    # The image is 28x28x1, so we drop the channel dimension for plotting
    plt.imshow(x_train[i].reshape(28, 28), cmap='gray')
```

```
    # Get the true label by reversing the one-hot encoding
    plt.title(f"Label: {np.argmax(y_train[i])}")
    plt.axis('off')
plt.show()
#
```

4. Define the CNN Architecture

We use a standard, yet effective, architecture for image classification: two sets of Convolutional and Pooling layers, followed by a Flatten layer and Dense layers for classification.

```
# Define the CNN model
def create_cnn_model(input_shape, num_classes):
    model = Sequential([
        # Layer 1: Convolution + ReLU Activation
        Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape),
        # Layer 2: Max Pooling (downsamples feature maps)
        MaxPooling2D(pool_size=(2, 2)),

        # Layer 3: Second Convolution + ReLU
        Conv2D(64, (3, 3), activation='relu'),
        # Layer 4: Second Max Pooling
        MaxPooling2D(pool_size=(2, 2)),

        # Layer 5: Flatten for Dense layers
        Flatten(),

        # Layer 6: Fully Connected (Dense) hidden layer
        Dense(128, activation='relu'),

        # Layer 7: Output layer (10 classes) with Softmax for probability scores
        Dense(num_classes, activation='softmax')
    ])
```

```
    return model

model = create_cnn_model(IMG_SHAPE, NUM_CLASSES)

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

model.summary()
```

5. Train the Model

We fit the model to the training data and use the test data for validation during training. The results are stored in the `history` object.

```
print("\nStarting model training...")
# Train the model
history = model.fit(
    x_train, y_train,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    verbose=1,
    validation_data=(x_test, y_test)
)
print("Model training complete.")
```

6. Evaluate and Visualize Training History

This step is critical to check for overfitting and monitor learning progress.

```
# --- A. Final Evaluation ---
print("\nFinal Evaluation on Test Set:")
score = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Loss: {score[0]:.4f}")
print(f"Test Accuracy: {score[1]*100:.2f}%")

# --- B. Plotting Loss and Accuracy ---
plt.figure(figsize=(12, 4))

# Plot Training & Validation Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Plot Training & Validation Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

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

```
#
```

7. Make Predictions (Inference)

Let's use the trained model to predict a few examples from the test set.

```
# Predict the first 5 images in the test set
predictions = model.predict(x_test[:5])

print("\nSample Predictions:")
for i in range(5):
    predicted_class = np.argmax(predictions[i])
    true_class = np.argmax(y_test[i])

    # Display the image and results
    plt.figure(figsize=(2, 2))
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {true_class}\nPred: {predicted_class}",
              color='green' if predicted_class == true_class else 'red')
    plt.axis('off')
    plt.show()

    print(f"Image {i+1}: True Label = {true_class}, Predicted Label = {predicted_class}")
    print("-" * 30)
```