Exercise: Handwritten Digit Classification using ANN (MNIST Dataset)

1. Objective

To build, train, and evaluate a feedforward artificial neural network (ANN) that classifies handwritten digits from the MNIST dataset using both manual training loop (GradientTape) and Keras's high-level API.

2. Tools Required

- Python 3.x
- TensorFlow
- NumPy
- Matplotlib

3. Dataset Description

The MNIST dataset contains 70,000 grayscale images of handwritten digits (0 to 9), each of size 28x28 pixels. It is divided into:

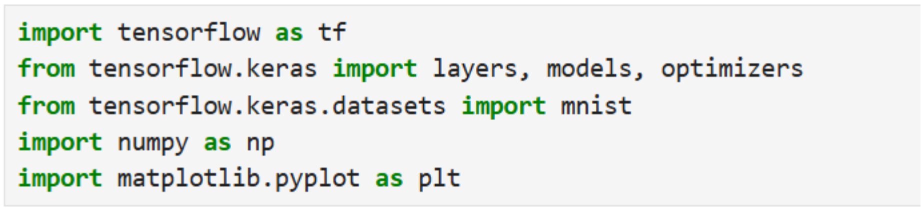
- Training Set: 60,000 images
- Test Set: 10,000 images

4. Summary of Key Concepts

Concept	Description	Role in Project
MNIST Dataset	Handwritten digit images and labels	Provides input images and expected outputs
ANN	Fully connected neural network	Learns patterns in image data to classify digits
Flatten Layer	Reshapes 28x28 to 784 vector	Prepares image data for dense layers
Dense Layer	Fully connected neural layer	Learns features through weighted connections
ReLU Activation	Applies ReLU non-linearity	Allows network to learn complex functions
Loss Function (CrossEntropy)	Measures difference between predicted and actual labels	Guides learning by minimizing classification error
Optimizer (Adam)	Optimizes weights using gradients	Adjusts model weights during training
GradientTape	Manual training method	Records operations for backpropagation
Epoch	One full pass over training data	Repeated passes help refine learning
Accuracy	Performance metric in classification	Measures correct predictions on test set
Softmax Layer	Converts logits to probabilities	Used during prediction for interpretation

5. Model Building Steps

Step 1: Import Libraries



```python
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
```

Explanation: These libraries are needed for building the ANN, processing data, and visualization.

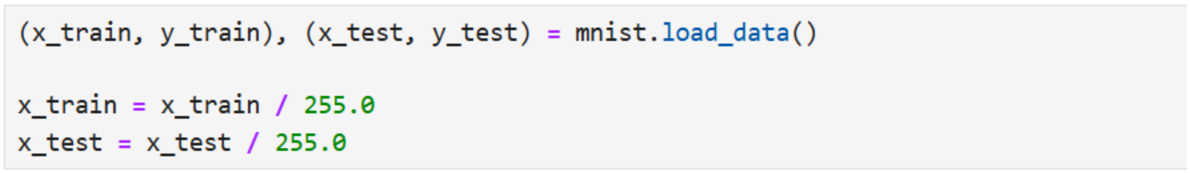
Question: Why do we import models from keras?
A: To use the Sequential model for stacking layers.

Question. What is the purpose of tensorflow.keras in this code?

Answer:

tensorflow.keras is a high-level API that allows us to build, train, and evaluate deep learning models easily. It includes layers, optimizers, and tools for loading datasets like MNIST.

Step 2: Load and Normalize the Data



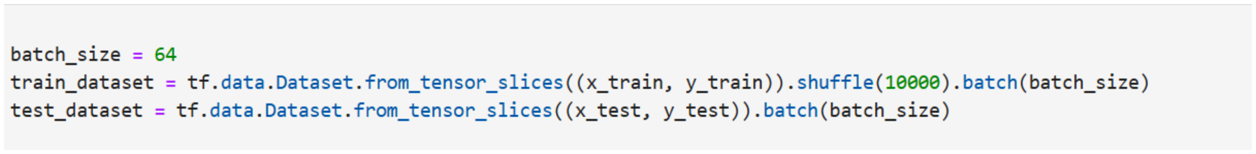
```python
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train / 255.0
x_test = x_test / 255.0
```

Explanation: Pixel values are scaled from [0, 255] to [0, 1] for faster learning.

Question: What is normalization?
Answer: Scaling features to a standard range, here [0, 1].
Question. Why do we divide the pixel values by 255?
Answer:

Pixel values range from 0 to 255. Dividing by 255 normalizes the values to a range of 0 to 1, which speeds up training and helps the model learn better.

Step 3: Create Training Batches

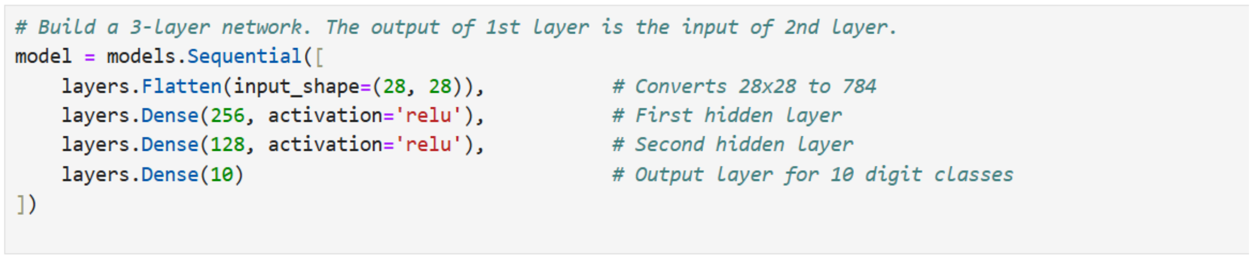


```
batch_size = 64
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(10000).batch(batch_size)
test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(batch_size)
```

Explanation: Batches help in efficient training. Shuffling ensures varied input order per epoch.

Question: Why use batching?
Answer: For computational efficiency and stable gradient estimates.

Step 4: Build the Neural Network Model



```
# Build a 3-layer network. The output of 1st layer is the input of 2nd layer.
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)),          # Converts 28x28 to 784
    layers.Dense(256, activation='relu'),          # First hidden layer
    layers.Dense(128, activation='relu'),          # Second hidden layer
    layers.Dense(10)                               # Output layer for 10 digit classes
])
```

Explanation: Sequential layers stack transformations on the input to produce logits.

Question. What is the purpose of the Flatten layer?
Answer:

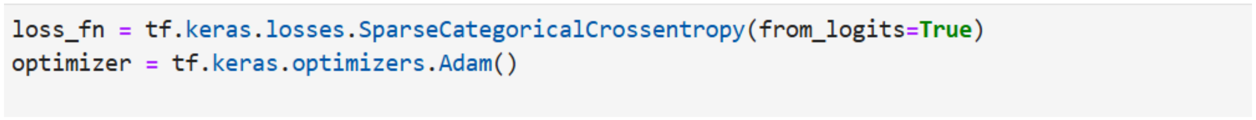
The Flatten layer converts the 2D image (28x28) into a 1D vector (784) so that it can be passed to the Dense layers.

Question. Why is the last Dense layer's output 10?
Answer:

Because we have 10 digit classes (0 to 9), we need 10 output neurons to represent the probability for each class.

Question: Why no softmax in the last layer?
Answer: We'll use logits with from_logits=True in loss function.

Step 5: Define Loss and Optimizer



```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam()
```

Explanation: Cross-entropy is suitable for classification; Adam adapts learning rates.

Question: Why SparseCategoricalCrossentropy?
Answer: Because labels are integers (not one-hot encoded).

Question. What does an optimizer do during training?
Answer:

The optimizer updates the model's weights using the gradients to reduce the loss and improve accuracy.

Section A: Manual Training Using GradientTape

Step 6: Manual Training Loop

```
epochs = 5
for epoch in range(epochs):
    total_loss = 0
    for step, (x_batch, y_batch) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            logits = model(x_batch, training=True)
            loss = loss_fn(y_batch, logits)
        grads = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        total_loss += loss.numpy()
    print(f"Epoch {epoch+1}, Loss: {total_loss/len(train_dataset):.4f}")
```

Explanation: Custom loop for educational clarity. Shows step-by-step learning and loss updates.

Question: What does GradientTape() do?
Answer: Records operations to compute gradients.

OUTPUT:

```
Epoch 1, Loss: 0.2277
Epoch 2, Loss: 0.0874
Epoch 3, Loss: 0.0580
Epoch 4, Loss: 0.0424
Epoch 5, Loss: 0.0332
```

What Do These Values Indicate?
The loss is decreasing steadily with each epoch:

- **Epoch 1 (0.2277) → relatively high, as the model starts with random weights.**
- **Epoch 5 (0.0332) → much lower, indicating the model has learned meaningful patterns from the training data.**

This suggests:

- ➢ **Your model is training correctly**
- ➢ **The optimizer is working**
- ➢ **Gradient descent is minimizing the loss**
- ➢ **The ANN is learning useful representations of the data**

NOTE: After every epoch, the model:

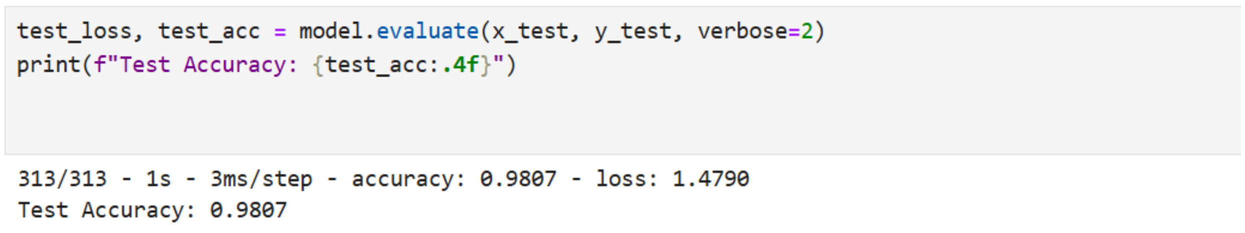
1. Makes predictions.
2. Compares predictions with actual labels.
3. Computes loss (error).
4. Adjusts weights using gradients to minimize the error.

This process, called backpropagation, helps the model improve its accuracy over time.

Summary Table of Questions

Code Line	Question	Answer
epochs = 5	What is an epoch?	One complete pass over the entire training dataset.
total_loss = 0	Why reset total_loss each epoch?	To calculate fresh average loss for the new epoch.
(x_batch, y_batch)	What is a batch?	A subset of training data processed in one step.
GradientTape()	What does it do?	Tracks operations to compute gradients.
training=True	Why set this flag?	To enable training behaviors like dropout.
loss_fn()	What does the loss function do?	Measures prediction error.
tape.gradient()	What are gradients?	Slopes that guide weight updates.
apply_gradients()	Why apply gradients?	To improve model accuracy by updating weights.
loss.numpy()	Why convert loss to NumPy?	To use it in Python arithmetic.
len(train_dataset)	Why divide by this?	To calculate average loss across all batches.

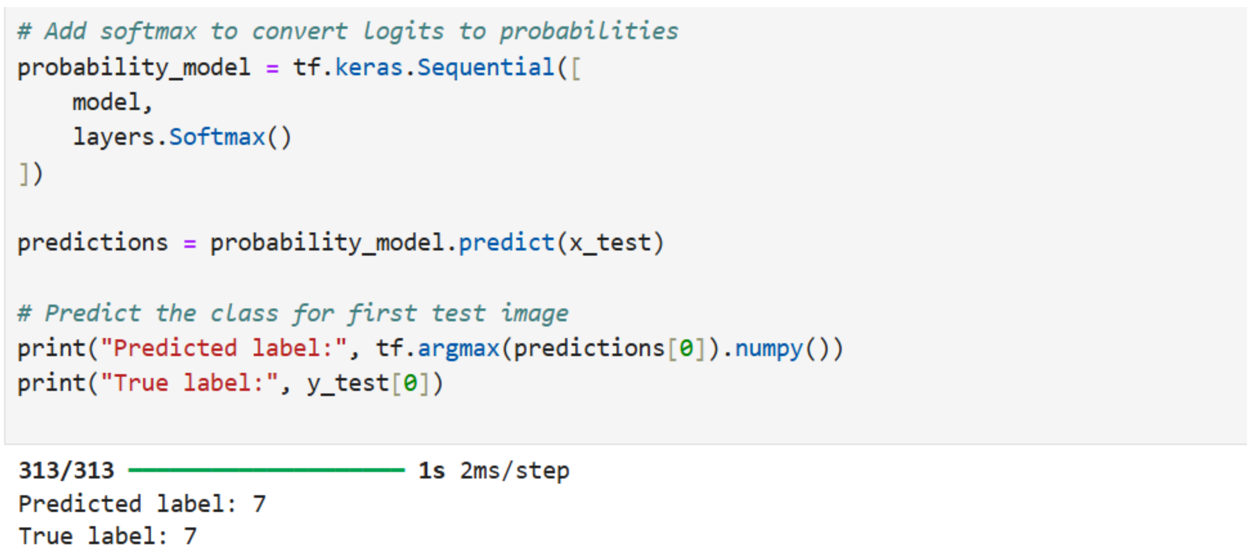
Step 7: Evaluate the Model Accuracy



```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
313/313 - 1s - 3ms/step - accuracy: 0.9807 - loss: 1.4790
Test Accuracy: 0.9807
```

Step 8: Make Predictions:



```python
# Add softmax to convert logits to probabilities
probability_model = tf.keras.Sequential([
    model,
    layers.Softmax()
])

predictions = probability_model.predict(x_test)

# Predict the class for first test image
print("Predicted label:", tf.argmax(predictions[0]).numpy())
print("True label:", y_test[0])
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Predicted label: 7
True label: 7
```

We compute accuracy by comparing predicted and true labels. argmax picks the class with the highest logit value.

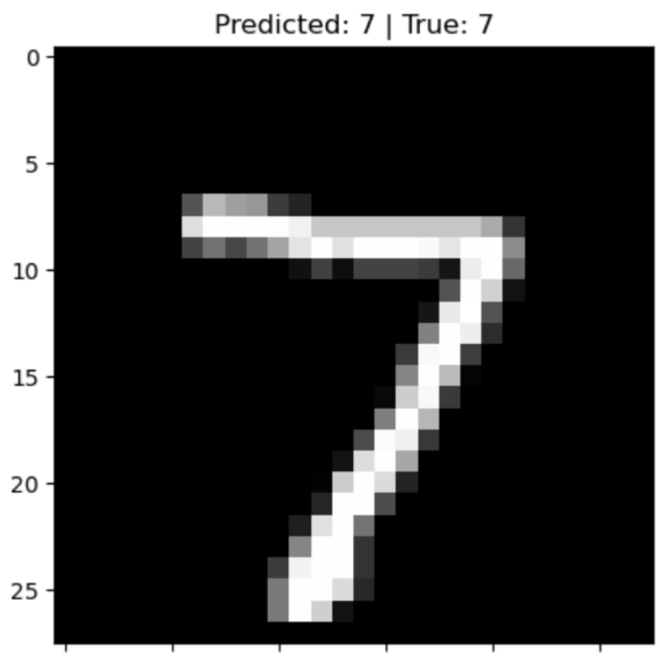
Question: Why set training=False during evaluation?
Answer: To disable layers like dropout or batch normalization.

Question: How is accuracy calculated?
Answer: Number of correct predictions divided by total test samples.

Question: Why use argmax?
Answer: To select the predicted class with highest score.

Step 9: Evaluate Individual Predictions:

```
plt.imshow(x_test[0], cmap='gray')
plt.title(f"Predicted: {tf.argmax(predictions[0]).numpy()} | True: {y_test[0]}")
plt.show()
```



Question: What does argmax(logits, axis=1) do?
Answer: Picks the index of the highest score (predicted class).

Question: Why use plt.imshow(images[i], cmap='gray')?
Answer: To display grayscale MNIST images.

Section B: Training Using High-Level Keras API

Complie the Model

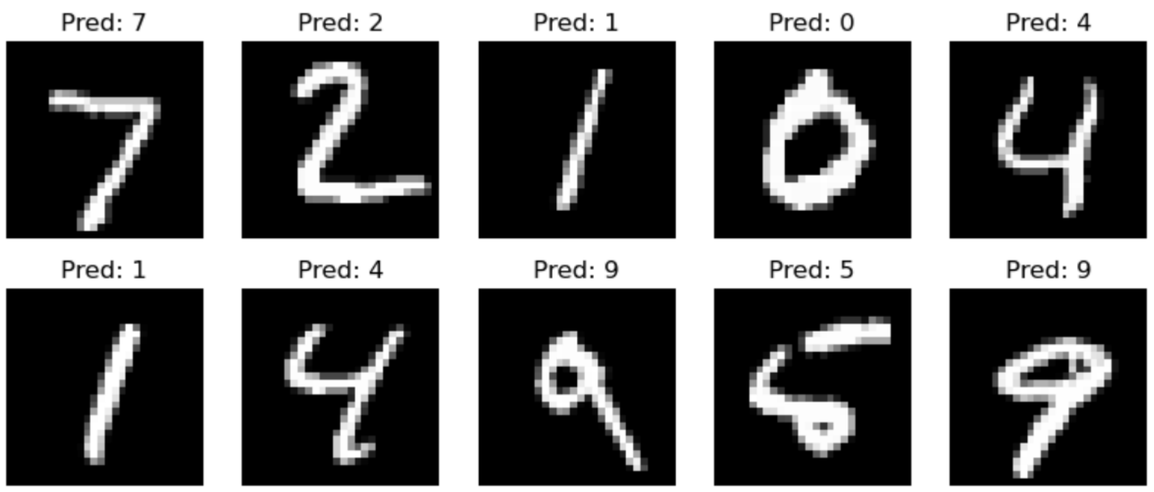
```
model.compile(
    optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

Explanation: Short and readable method using Keras's high-level API.

Question: When is model.fit() preferred?
Answer: When you don't need custom training steps.

6. Visualizing Predictions

```python
plt.figure(figsize=(10, 4))
for images, labels in test_dataset.take(1):
    logits = model(images)
    preds = tf.argmax(logits, axis=1)
    for i in range(10):
        plt.subplot(2, 5, i+1)
        plt.imshow(images[i], cmap='gray')
        plt.title(f"Pred: {preds[i].numpy()}")
        plt.axis('off')
    plt.show()
```



Explanation: Useful for visual verification of model predictions.

7. Real-world Applications

- Digit recognition on postal codes (OCR)
- Bank cheque processing
- Touchscreen handwriting input

8. Conclusion

This exercise demonstrated building and training a simple ANN for digit classification using both a manual and high-level API.