Commented TensorFlow MNIST Code

```
import tensorflow as tf  # Import TensorFlow library
from tensorflow.keras.datasets import mnist  # Import MNIST dataset
from tensorflow.keras.models import Sequential  # Import Sequential model class
from tensorflow.keras.layers import Dense, Flatten  # Import layers: Dense and Flatten
from tensorflow.keras.utils import to_categorical  # Import utility for one-hot encoding
import matplotlib.pyplot as plt  # Import matplotlib for plotting
import numpy as np  # Import NumPy for numerical operations

# 1. Load and preprocess the data
(x_train, y_train), (x_test, y_test) = mnist.load_data()  # Load MNIST dataset (60k train, 10k test)

# Normalize pixel values to range [0,1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# One-hot encode labels (e.g., '3' -> [0,0,0,1,0,0,0,0,0,0])
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

# 2. Build the model
model = Sequential([
    Flatten(input_shape=(28,28)),        # Flatten 28x28 image into 784 vector
    Dense(128, activation='relu'),       # Fully connected hidden layer with ReLU
    Dense(10, activation='softmax')      # Output layer with 10 neurons for digits 0–9
])

# 3. Compile the model
model.compile(optimizer='adam',  # Adam optimizer
              loss='categorical_crossentropy',  # Loss for multi-class classification
              metrics=['accuracy'])  # Track accuracy during training

# 4. Train the model
model.fit(x_train, y_train_cat, epochs=5, batch_size=32, validation_split=0.1)
# Train for 5 epochs with batch size 32, using 10% of training data for validation

# 5. Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test_cat, verbose=0)  # Evaluate on test set
print(f"Test Accuracy: {test_acc:.4f}")  # Print test accuracy

# 6. Predict a single test sample
sample_index = 0  # Choose the first test image
sample_image = x_test[sample_index]  # Extract image
sample_label = y_test[sample_index]  # Extract true label

sample_input = np.expand_dims(sample_image, axis=0)  # Add batch dimension (1,28,28)
pred_probs = model.predict(sample_input)  # Predict softmax probabilities
pred_class = np.argmax(pred_probs)  # Get class with highest probability

print("True label:", sample_label)  # Print true label
print("Predicted label:", pred_class)  # Print predicted label
print("Softmax probabilities:", pred_probs[0])  # Print probability distribution

# 7. Optional: plot the image and probabilities
plt.figure(figsize=(8,4))

# Show the test image
plt.subplot(1,2,1)
plt.imshow(sample_image, cmap='gray')
plt.title(f"True: {sample_label}, Pred: {pred_class}")
plt.axis('off')
# Show softmax probabilities as a bar chart
```

```
plt.subplot(1,2,2)
plt.bar(range(10), pred_probs[0])
plt.xticks(range(10))
plt.xlabel("Digit")
plt.ylabel("Probability")
plt.title("Softmax Probabilities")

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