Adebanke Damola-Fashola

ITAI 3377- AI at the Edge and IIOT Environments

Professor Patricia McManus

February 04, 2025.

Converting and Deploying AI Models using TensorFlow Lite - Explanation of Steps

Part 1: Setting Up the Development Environment

Step 1 - Verify Python and TensorFlow Installation:

- This step confirms version Python 3.12.7 and TensorFlow 2.18.0 are installed and functional.

```
[3]: !python --version

Python 3.12.7
```

```
[4]: !pip show tensorflow

Name: tensorflow
Version: 2.18.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: C:\Users\banke\anaconda3\Lib\site-packages
Requires: tensorflow-intel
Required-by:
```

```
[5]: !pip install tensorflow

Requirement already satisfied: tensorflow in c:\users\banke\anaconda3\lib\site
Requirement already satisfied: tensorflow-intel==2.18.0 in c:\users\banke\anac
Requirement already satisfied: absl-py>=1.0.0 in c:\users\banke\anaconda3\lib\
Requirement already satisfied: astunparse>=1.6.0 in c:\users\banke\anaconda3\l
Requirement already satisfied: flatbuffers>=24.3.25 in c:\users\banke\anaconda
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\users
ow) (0.6.0)
```

Step 2 - Install Jupyter Notebook (if working locally):

- Jupyter Notebook is already installed on my device.

Part 2: Creating and Training an AI Model

Step 3 - Load the MNIST Dataset:

- The libraries required for the project were imported.

```
[7]: import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.datasets import mnist
     import numpy as np
     import matplotlib.pyplot as plt
```

- The MNIST dataset was loaded and split into training and testing sets.

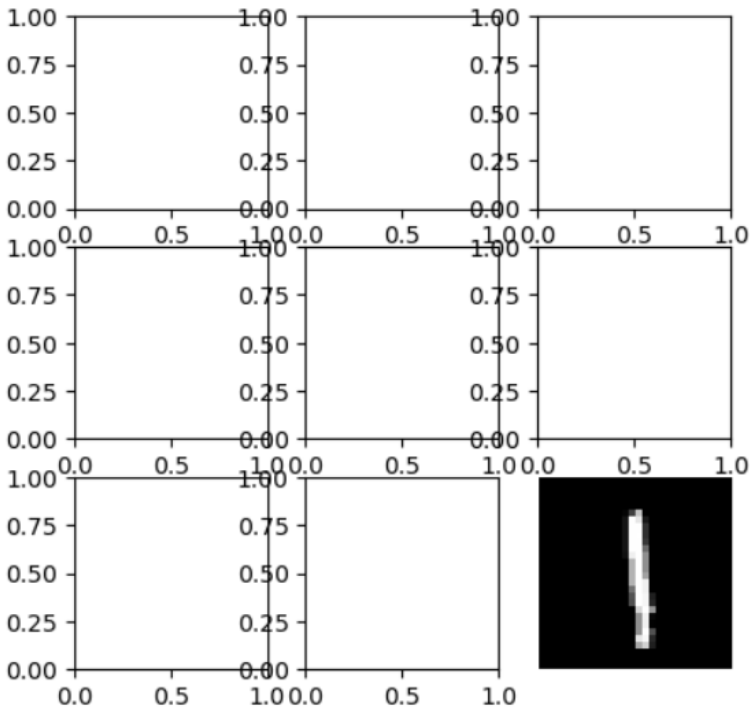
```
[8]: # Load MNIST dataset
     (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- The pixel values of the MNIST images are normalized to a range of 0 to 1 to prepare the data for models.

```
# Normalize data (scale pixel values between 0 and 1)
x_train, x_test = x_train / 255.0, x_test / 255.0
```

- The 9 images in the MNIST training set are displayed as a 3x3 grid of grayscale images, effectively visualizing a sample of the handwritten digits in the dataset.

[9]:
```
# Show sample images
plt.figure(figsize=(5,5))
for i in range(9): plt.subplot(3,3,i+1)
plt.imshow(x_train[i], cmap="gray")
plt.axis('off')
plt.show()
```



Step 4 - Define and Train a Neural Network:

- A defined feedforward neural network is used to classify handwritten digits (MNIST dataset). It takes a 28x28 image, flattens it into a 784-element vector, and passes it through a hidden layer with 128 neurons and ReLU activation. It finally passes through an output layer with 10 neurons and softmax activation to produce a probability distribution over 10 digits.

[12]:
```
# Define model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),  # Input layer
    tf.keras.layers.Dense(128, activation='relu'),  # Hidden layer
    tf.keras.layers.Dense(10, activation='softmax') # Output layer (10 classes)
])
```

- The model is prepared for training using the Adam optimizer to update the model's weights and using sparse categorical cross-entropy as the loss function to measure the difference between predicted and actual labels. It also tracks accuracy as the performance metric during training and evaluation.

```
[13]: # Compile model
      model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

- The neural network is trained for 5 epochs, using the training data to adjust the model's weights and the validation dataset (test set) to monitor its performance on unseen data. The output provides a log of the training process, showing the metrics at the end of each epoch.

```
[14]: # Train model
      model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8785 - loss: 0.4335 - val_accuracy: 0.9606 - val_loss: 0.1349
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.9636 - loss: 0.1216 - val_accuracy: 0.9688 - val_loss: 0.1085
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.9775 - loss: 0.0765 - val_accuracy: 0.9721 - val_loss: 0.0884
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.9823 - loss: 0.0570 - val_accuracy: 0.9766 - val_loss: 0.0804
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.9876 - loss: 0.0429 - val_accuracy: 0.9767 - val_loss: 0.0798
```

- The trained Keras model (architecture and weights) is saved to a file 'mnist_model.keras' without the optimizer state. Saving the model is important to avoid retraining the model every time it is required.

```
[15]: # Save trained model
      model.save("mnist_model.keras", include_optimizer=False)
```

Part 3: Converting and Saving the Model

Step 5 - Convert the Model to TensorFlow Lite Format:

- The pre-trained Keras model is loaded and converted to the TensorFlow Lite format.

```
[18]: # Load trained model
model = tf.keras.models.load_model("mnist_model.keras")
```

```
[19]: # Convert to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

```
INFO:tensorflow:Assets written to: C:\Users\banke\AppData\Local\Temp\tmp4ju887nu\assets
INFO:tensorflow:Assets written to: C:\Users\banke\AppData\Local\Temp\tmp4ju887nu\assets
Saved artifact at 'C:\Users\banke\AppData\Local\Temp\tmp4ju887nu'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(32, 28, 28), dtype=tf.float32, name='input_layer')
Output Type:
  TensorSpec(shape=(32, 10), dtype=tf.float32, name=None)
Captures:
  1840127155600: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1840127155024: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1840127154640: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1840127149648: TensorSpec(shape=(), dtype=tf.resource, name=None)
```

- The TensorFlow Lite model created is saved to a file ‘mnist_model.tflite’.

```
[20]: # Save the converted model
with open("mnist_model.tflite", "wb") as f:
    f.write(tflite_model)
```

Part 4: Loading and Running Inference with TensorFlow Lite

Step 6 - Load the Converted Model Using TensorFlow Lite Interpreter:

- The TensorFlow Lite model is loaded from the "mnist_model.tflite" file and prepared for inference by allocating the necessary memory for the tensors.

```
[23]: # Load TensorFlow Lite model
interpreter = tf.lite.Interpreter(model_path="mnist_model.tflite")
interpreter.allocate_tensors()
```

- This code retrieves metadata about the input and output tensors of the TensorFlow Lite model.

```
[24]: # Get input and output tensor details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

- Displays input details and output details.

```
[25]: print("Input Details:", input_details)
```

```
Input Details: [{'name': 'serving_default_input_layer:0', 'index': 0, 'shape': array([32, 28, 28]), 'shape_signature': array([32, 28, 28]), 'dtype': <cla
ss 'numpy.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32), 'zero_points': array([], dtype=int32), 'qu
antized_dimension': 0}, 'sparsity_parameters': {}}]
```

```
[26]: print("Output Details:", output_details)
```

```
Output Details: [{'name': 'StatefulPartitionedCall_1:0', 'index': 9, 'shape': array([32, 10]), 'shape_signature': array([32, 10]), 'dtype': <class 'nump
y.float32'>, 'quantization': (0.0, 0), 'quantization_parameters': {'scales': array([], dtype=float32), 'zero_points': array([], dtype=int32), 'quantized_
dimension': 0}, 'sparsity_parameters': {}}]
```

Step 7 - Perform Inference with TensorFlow Lite:

- This code selects the first image from the test dataset (x_test) and converts its data type to np.float32.

```
[28]: # Select a test image
test_image = x_test[0].astype(np.float32)
```

- It takes a single test image, creates a batch of 32 identical copies, and ensures the data type is float32, preparing it to be fed into the TensorFlow Lite model for classification.

```
[29]: # Ensure data type matches model input
test_image = np.expand_dims(test_image, axis=0)  # Add batch dimension (1, 28, 28)
test_image = np.tile(test_image, (32, 1, 1))  # (32, 28, 28)
# Convert to float32 (required for TensorFlow Lite)
test_image = test_image.astype(np.float32)
```

- It takes the prepared test image (test_image) and assigns it as the input to the TensorFlow Lite model using the correct tensor index.

```
[31]: # Set the input tensor
      interpreter.set_tensor(input_details[0]['index'], test_image)
```

- The code runs the TensorFlow Lite model and generates a prediction based on the provided input. It is the core step in using a TensorFlow Lite model for inference.

```
[32]: # Run inference
      interpreter.invoke()
```

- The code retrieves the model's output (the probability distribution over the digits), finds the digit with the highest probability using np.argmax(), and stores the predicted digit label in the predicted_label variable. This is the final step in classifying an image using the TensorFlow Lite model.

```
[33]: # Get the prediction
      output_data = interpreter.get_tensor(output_details[0]['index'])
      predicted_label = np.argmax(output_data)
```

- The code takes the first image from the test set, displays it in grayscale, and adds a title showing both the model's prediction (predicted_label) and the actual correct label (y_test[0]). The model's prediction can be compared with the ground truth. The output shows the image of the digit '7' and the title 'Predicted Label: 7, Actual Label: 7', indicating that the model correctly classified the image.

[34]:
```
# Display the image and prediction
plt.imshow(x_test[0], cmap="gray")
plt.title(f"Predicted Label: {predicted_label}, Actual Label: {y_test[0]}")
plt.show()
```