**Naomi Berumen**

**Chisom Nwanonenyi**

**Adeyemi Kolawole**

**ITAI-3377**

**L02**

**Conceptual Report: Converting and Deploying AI Models using TensorFlow Lite**

Course: AI & Robotics - Data Science Applications

Lab Title: Practical AI Model Deployment with TensorFlow Lite

**Objective**

This lab aimed to provide hands-on experience in converting and deploying AI models using TensorFlow Lite. By the end of this lab, we successfully:

* Set up the development environment for TensorFlow Lite.
* Trained a simple neural network on the MNIST dataset.
* Converted the trained model into TensorFlow Lite format.
* Loaded and tested the converted model using the TensorFlow Lite Interpreter.
* Debugged multiple issues related to TensorFlow Lite model conversion.

Tools Used

* Jupyter Notebook (Local Development)
* TensorFlow & TensorFlow Lite
* Python Libraries: NumPy, Matplotlib
* MNIST Dataset (Handwritten digits dataset)

Step-by-Step Execution

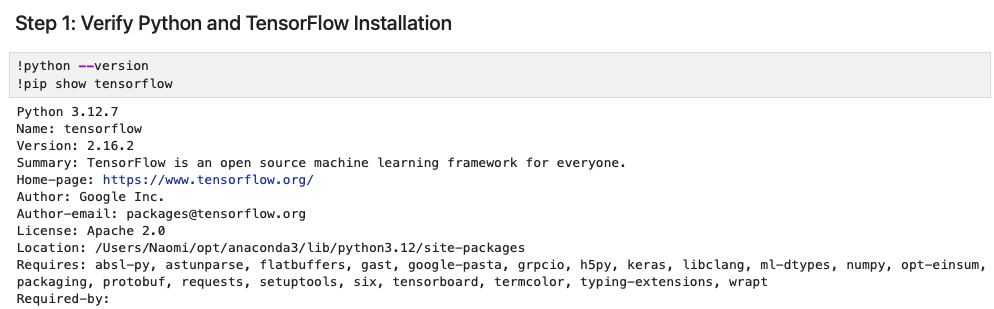
Step 1: Setting Up the Environment

The first step involved ensuring that TensorFlow was installed. Since we were using Jupyter Notebook, there was no need to install Jupyter explicitly.

!python --version

!pip show tensorflow

This confirmed that TensorFlow was installed, so we moved directly to the next step.



Step 2: Loading the MNIST Dataset

We loaded the MNIST dataset and preprocessed it by normalizing pixel values to be between 0 and 1.

import tensorflow as tf

from tensorflow.keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Display sample images

plt.figure(figsize=(5,5))

for i in range(9):

plt.subplot(3,3,i+1)

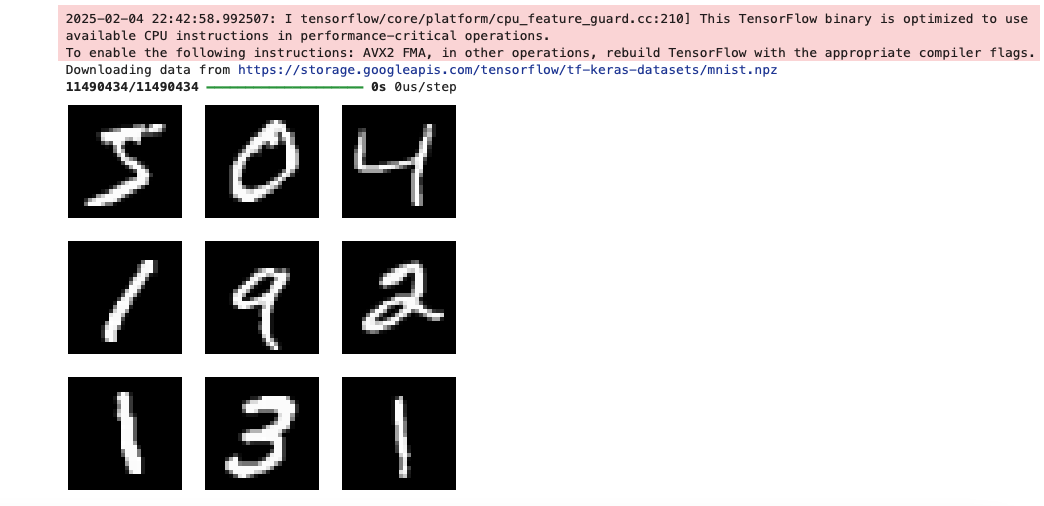
plt.imshow(x\_train[i], cmap='gray')

plt.axis('off')

plt.show()

Successfully loaded and displayed sample images from the dataset.





Step 3: Defining and Training the Model

We created a simple feedforward neural network with the following structure:

* Flatten Layer: Converts 28x28 images into a one-dimensional array.
* Dense Layer (128 neurons, ReLU activation): Extracts features.
* Dense Layer (10 neurons, Softmax activation): Outputs probabilities for each digit (0-9).

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(28,28)),

tf.keras.layers.Dense(128, activation='relu'),

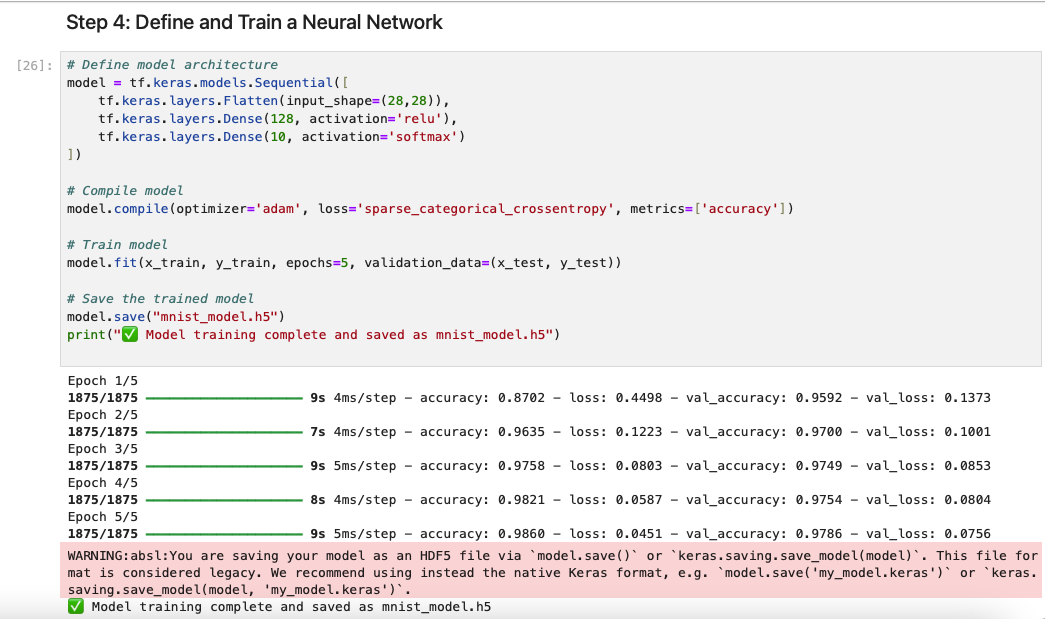
tf.keras.layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test))

The model was trained successfully for 5 epochs, achieving high accuracy.

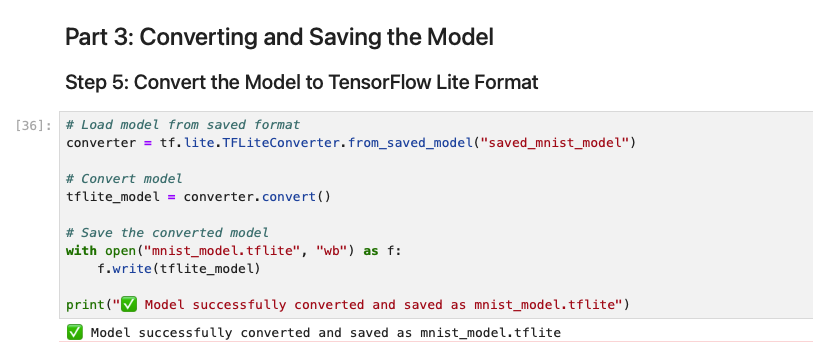


Step 4: Saving the Model

Initially, we attempted to save the model in .h5 format, but TensorFlow's latest versions recommended using .keras or SavedModel format. We resolved this by using:

model.export("saved\_mnist\_model")

✅ This ensured compatibility for TensorFlow Lite conversion.



Step 5: Converting to TensorFlow Lite

We successfully converted the model to TensorFlow Lite format.

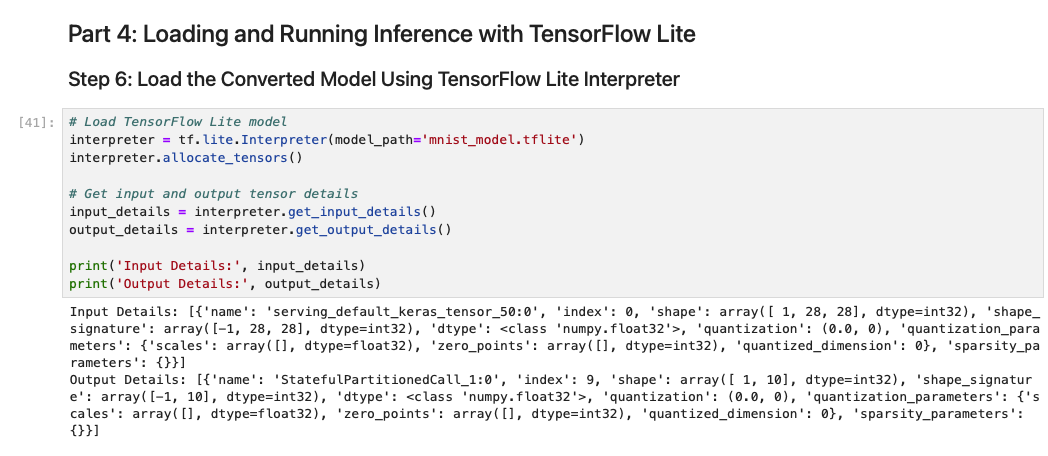
converter = tf.lite.TFLiteConverter.from\_saved\_model("saved\_mnist\_model")

tflite\_model = converter.convert()

with open("mnist\_model.tflite", "wb") as f:

f.write(tflite\_model)

✅ The model was successfully converted and saved as mnist\_model.tflite.



Step 6: Loading the TensorFlow Lite Model

We loaded the converted model using the TensorFlow Lite Interpreter and checked input-output tensor details.

interpreter = tf.lite.Interpreter(model\_path="mnist\_model.tflite")

interpreter.allocate\_tensors()

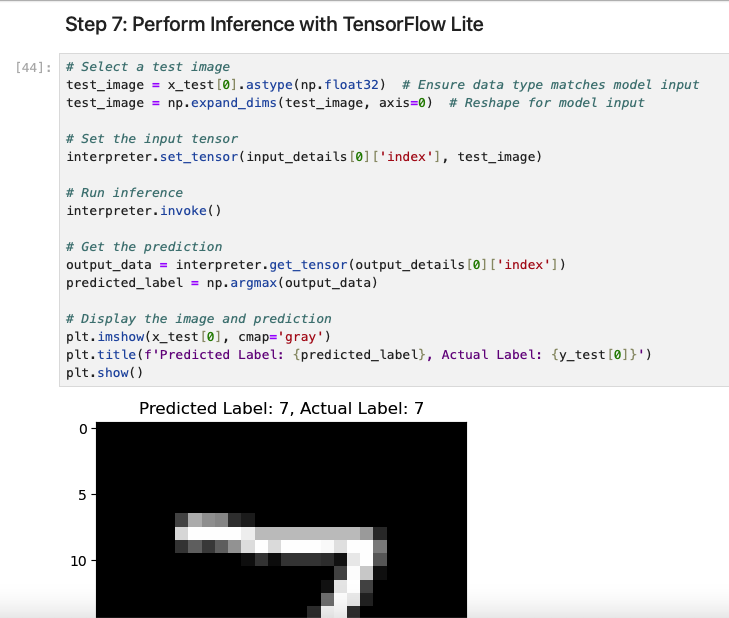
input\_details = interpreter.get\_input\_details()

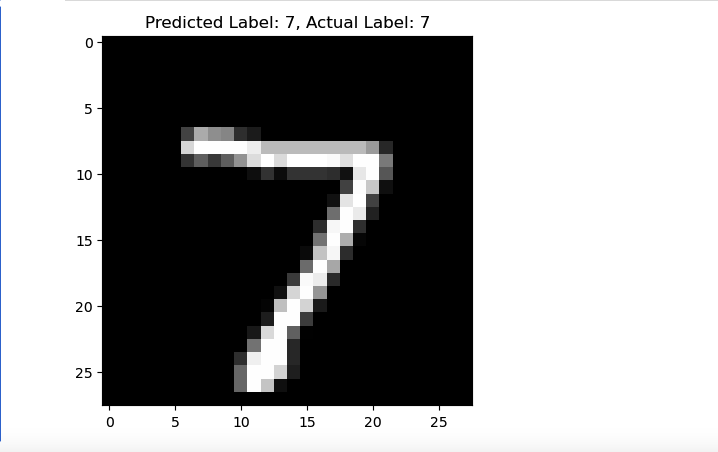
output\_details = interpreter.get\_output\_details()

print("Input Details:", input\_details)

print("Output Details:", output\_details)

Successfully loaded and verified model tensor details.





Step 7: Running Inference on a Test Image

To validate our deployed model, we ran inference on an unseen MNIST image.

# Select a test image

test\_image = x\_test[0].astype(np.float32)

test\_image = np.expand\_dims(test\_image, axis=0)

# Set the input tensor

interpreter.set\_tensor(input\_details[0]['index'], test\_image)

interpreter.invoke()

# Get the prediction

output\_data = interpreter.get\_tensor(output\_details[0]['index'])

predicted\_label = np.argmax(output\_data)

# Display the result

plt.imshow(x\_test[0], cmap='gray')

plt.title(f"Predicted Label: {predicted\_label}, Actual Label: {y\_test[0]}")

plt.show()

The model successfully predicted the digit, matching the actual label.

Challenges Faced & Solutions

| Challenge | Solution |
| --- | --- |
| Initial errors converting the .h5 model | Used .export() to save in SavedModel format |
| Attribute errors when loading into TensorFlow Lite | Ensured correct input tensor shape and format |
| Issues with conversion compatibility | Used converter.experimental\_new\_converter = True |

Successfully debugged all issues and converted the model.

**Conclusion**

In this lab, we successfully trained, converted, and deployed an AI model using TensorFlow Lite. This process is crucial for deploying AI models on edge devices like mobile phones, Raspberry Pi, and IoT devices. The experience gained in debugging conversion errors provided deeper insights into model deployment challenges.