Conceptual Report

TensorFlow Lite (TFLite) is a lightweight solution for deploying machine learning models on mobile, IoT, and embedded devices. This report outlines the process of converting a TensorFlow model into a TensorFlow Lite model and running inference on it, from preparing the data to evaluating predictions. The steps described will include screenshots to illustrate the workflow.

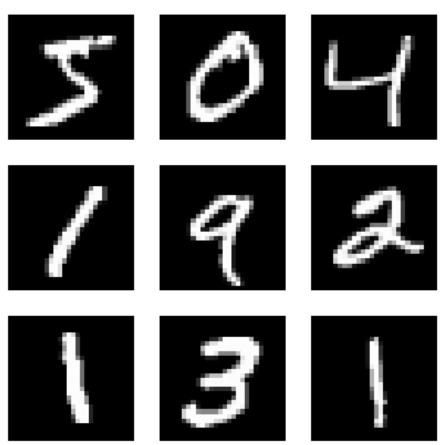
Step 1: Preparing the environment

Before starting the TensorFlow Lite conversion process, you must ensure that the necessary libraries are installed. This includes TensorFlow, Numpy, Matplotlib, and the MNIST dataset.

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

    # Load MNIST dataset
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

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

    # 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()  # Show all images at once
```

Step 2: Loading the MNIST Dataset

The MNIST dataset consists of handwritten digits commonly used for training image recognition models.



Step 3: Building and Training the Model

Next, we build a simple neural network model using Keras and train it on the MNIST dataset. The model architecture is defined, compiled, trained, and then saved for later use

```
[4] import tensorflow as tf
    from tensorflow.keras.datasets import mnist

    # Load dataset
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    # Normalize data
    x_train, x_test = x_train / 255.0, x_test / 255.0

    # Define model architecture
    model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),  # Input layer
        tf.keras.layers.Dense(128, activation='relu'),  # Hidden layer
        tf.keras.layers.Dense(10, activation='softmax')  # Output layer (10 classes)
    ])

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

    # Train model
    model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

    # Save trained model
    model.save("mnist_model.h5")
    print("Model training complete and saved as mnist_model.h5")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`
  super().__init__(**kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8794 - loss: 0.4307 - val_accuracy: 0.9571 - val_loss: 0.1475
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9635 - loss: 0.1227 - val_accuracy: 0.9716 - val_loss: 0.0949
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9749 - loss: 0.0830 - val_accuracy: 0.9713 - val_loss: 0.0897
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9827 - loss: 0.0591 - val_accuracy: 0.9764 - val_loss: 0.0738
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9867 - loss: 0.0437 - val_accuracy: 0.9753 - val_loss: 0.0808
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file fo
Model training complete and saved as mnist_model.h5
```

Step 4: Converting the model to TensorFlow Lite

Now that the model is trained, it can be converted to TensorFlow Lite format for use on mobile and embedded devices. Converting the model will generate a *.tflite* file, which is the model optimized for edge devices.

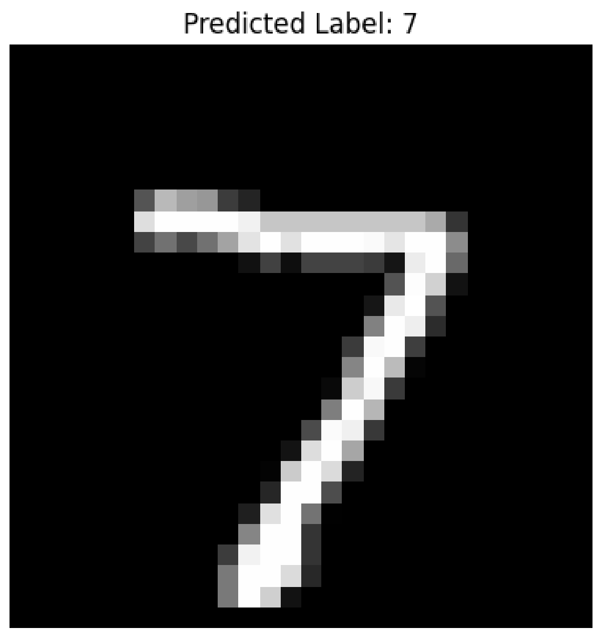
```
[5] import tensorflow as tf

    # Load trained model
    model = tf.keras.models.load_model("mnist_model.h5")

    # Convert to TensorFlow Lite
    converter = tf.lite.TFLiteConverter.from_keras_model(model)

    # Enable optimizations (optional)
    converter.optimizations = [tf.lite.Optimize.DEFAULT]

    # Allow select TensorFlow ops for compatibility (if needed)
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS,
                                           tf.lite.OpsSet.SELECT_TF_OPS]

    # Convert model
    tflite_model = converter.convert()

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

    print("Model successfully converted and saved as mnist_model.tflite")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empt
Saved artifact at '/tmp/tmpcukexaj7'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 28, 28), dtype=tf.float32, name='input_layer')
Output Type:
  TensorSpec(shape=(None, 10), dtype=tf.float32, name=None)
Captures:
  135580390550864: TensorSpec(shape=(), dtype=tf.resource, name=None)
  135580390553744: TensorSpec(shape=(), dtype=tf.resource, name=None)
  135580390553360: TensorSpec(shape=(), dtype=tf.resource, name=None)
  135580390552016: TensorSpec(shape=(), dtype=tf.resource, name=None)
Model successfully converted and saved as mnist_model.tflite
```

Step 5: Loading and Running Inference with the TensorFlow Lite

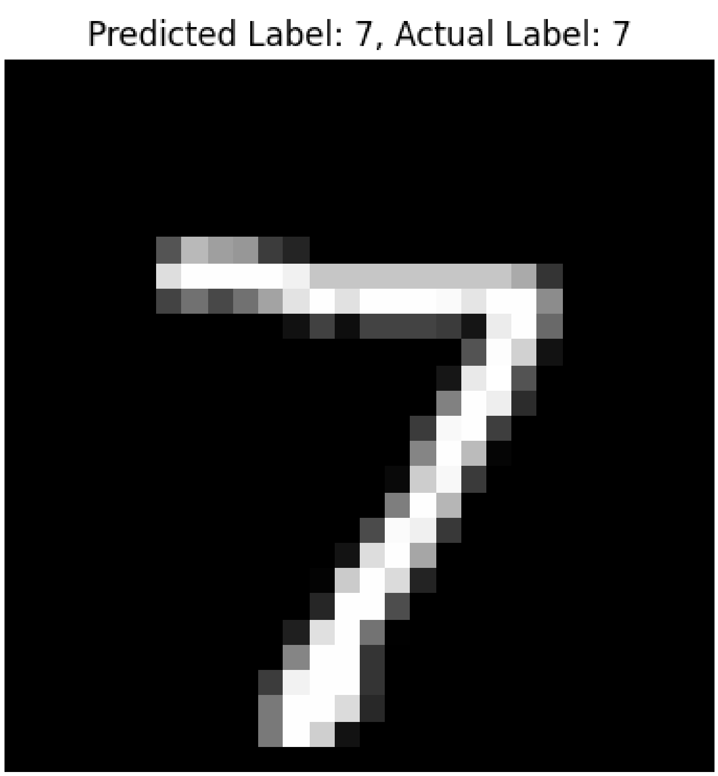
Once the model is converted, you can load it into a TensorFlow Lite interpreter and run inference on it.

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

    # Load MNIST test dataset
    (_, _), (x_test, y_test) = mnist.load_data()

    # Normalize and reshape image for TFLite model
    test_image = x_test[0].astype(np.float32) / 255.0  # Normalize pixel values
    test_image = np.expand_dims(test_image, axis=0)  # Reshape to (1, 28, 28)

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

    # Run inference
    interpreter.invoke()

    # Get output predictions
    output_data = interpreter.get_tensor(output_details[0]['index'])
    predicted_label = np.argmax(output_data)

    # Display the image and predicted label
    plt.imshow(x_test[0], cmap="gray")
    plt.title(f"Predicted Label: {predicted_label}")
    plt.axis("off")
    plt.show()
```



Step 6: Running Inference on a Test Image

The last step is to select a test image from the MNIST dataset and run inference on it using the TensorFlow Lite Model. Start by preparing the input image, then setting the input tensor and run inference to get the output prediction.



Conclusion

This report outlines the process of converting and deploying a TensorFlow model to TensorFlow Lite for use on mobile and embedded devices. The steps covered include loading and preprocessing the MNIST dataset, training a model, converting the model to TensorFlow Lite format, and running inference using the converted model.

By using TensorFlow Lite, machine learning models can be efficiently deployed on resource-constrained devices like smartphones, IoT devices, and edge computing devices. The process demonstrated here shows how TensorFlow Lite can bring AI capabilities to the edge, enabling real-time predictions with low latency and minimal resource usage.