TensorFlow Lite for Microcontrollers

This example demonstrates how to deploy a simple TensorFlow Lite model for addition on a microcontroller using TensorFlow Lite Micro library.

Steps Overview

Train and Convert a Model to TensorFlow Lite.

Generate a C Array from the TFLite Model.

Deploy the Model on Microcontroller (with Arduino as an example).

1. Train and Convert the Model

The model will add two numbers. Train the model in Python and export it as a .tflite file.

```
import tensorflow as tf

import numpy as np


# Define a simple addition model

def create_model():

   model = tf.keras.Sequential([

      tf.keras.layers.Input(shape=(2,)),  # Input two numbers

      tf.keras.layers.Dense(1)        # Output their sum

   ])

   model.compile(optimizer='adam', loss='mse')

   return model


# Create and train the model

model = create_model()

x_train = np.random.rand(1000, 2)  # 1000 pairs of random numbers
```

```
y_train = np.sum(x_train, axis=1)  # Their sums
model.fit(x_train, y_train, epochs=10, verbose=0)

# Convert the model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TFLite model to a file
with open("addition_model.tflite", "wb") as f:
    f.write(tflite_model)
```

2. Generate a C Array

Use the xxd tool to convert the .tflite model into a C array.

```
xxd -i addition_model.tflite > addition_model.h
```

The generated addition_model.h will contain the model data as a C array.

3. Deploy on Microcontroller

Use the TensorFlow Lite for Microcontrollers library and integrate the model in an Arduino project.

Install TensorFlow Lite for Microcontrollers

Install Arduino IDE.

Install the TensorFlow Lite library via Tools > Manage Libraries (search for TensorFlow).

Example Code for Arduino

```
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
#include "addition_model.h"  // Include the model C array

// Globals
constexpr int tensor_arena_size = 2 * 1024;
uint8_t tensor_arena[tensor_arena_size];

void setup() {
  // Initialize serial communication
  Serial.begin(9600);

  // Set up logging
  tflite::MicroErrorReporter error_reporter;

  // Map the model
  const tflite::Model* model = tflite::GetModel(addition_model);
```

```
 if (model->version() != TFLITE_SCHEMA_VERSION) {
  Serial.println("Model schema version mismatch!");
  return;
 }

 // Define the resolver and interpreter
 tflite::MicroMutableOpResolver<10> resolver;  // Resolver for operator support
 resolver.AddBuiltin(tflite::BuiltinOperator_ADD, tflite::ops::micro::Register_ADD());

 tflite::MicroInterpreter interpreter(model, resolver, tensor_arena, tensor_arena_size,
&error_reporter);
 interpreter.AllocateTensors();

 // Get input and output tensors
 TfLiteTensor* input = interpreter.input(0);
 TfLiteTensor* output = interpreter.output(0);

 // Input two numbers
 input->data.f[0] = 2.5f;  // First number
 input->data.f[1] = 3.5f;  // Second number

 // Run inference
 if (interpreter.Invoke() != kTfLiteOk) {
  Serial.println("Invoke failed!");
  return;
 }

 // Output result
```

```
  float result = output->data.f[0];
  Serial.print("Result: ");
  Serial.println(result);
}

void loop() {
  // Do nothing in loop
}
```

Key Notes

Microcontroller Compatibility: Ensure your microcontroller supports TensorFlow Lite for Microcontrollers, such as Arduino Nano 33 BLE Sense or ESP32.

Memory Constraints: Keep the model small (<2 KB for tiny models).

TensorFlow Lite Micro Library: Use the appropriate library for your embedded platform.

This simple addition model demonstrates the workflow to deploy a TensorFlow Lite model on microcontrollers.

What is tiny micro in tensor flow lite ?

TensorFlow Lite for Microcontrollers (TFLM), often referred to as "Tiny Micro", is a version of TensorFlow Lite specifically optimized to run machine learning models on microcontrollers and other resource-constrained devices. It is designed for ultra-low-power environments where devices typically have limited memory (e.g., kilobytes) and low computational power.

Key Features of TensorFlow Lite for Microcontrollers

Optimized for Low Resources:

Runs on devices with as little as tens of kilobytes of RAM and low-power CPUs.

Models and operations are optimized for size and efficiency.

No Operating System (RTOS/FreeRTOS Compatible):

Designed to work without requiring an operating system.

Compatible with bare-metal applications and lightweight real-time operating systems (RTOS).

Static Memory Allocation:

Uses a fixed amount of memory allocated at compile-time to ensure stability on constrained devices.

Reduced Code Size:

Only includes the operations necessary for the deployed model, minimizing code size.

Typical binary size can be less than 20 KB.

Broad Hardware Support:

Supports popular microcontrollers like ARM Cortex-M (STM32, Nordic, NXP), ESP32, Arduino Nano 33 BLE Sense, and others.

Works with specialized accelerators like Coral Edge TPU, Ethos-U, and NPU units.

Precompiled Kernels for Efficiency:

Optimized kernels for specific operations (e.g., convolution, matrix multiplication) ensure fast inference times.

Supports Quantized Models:

Focuses on running 8-bit integer quantized models to further reduce memory and compute requirements.

Workflow with TensorFlow Lite for Microcontrollers

Model Design and Training:

Train a TensorFlow model using Python (e.g., for tasks like gesture recognition, speech processing, or anomaly detection).

Use tools like post-training quantization to reduce the model size.

Convert Model to TFLite:

Convert the trained TensorFlow model into a .tflite format using the TensorFlow Lite Converter.

Optionally, use the TensorFlow Lite Optimizer for int8 quantization.

Integrate with Microcontroller:

Use the TensorFlow Lite Micro C++ library to run the model.

Deploy the model as a static C array in your embedded application.

Deploy and Test:

Deploy the firmware on the target device.

Test the model's inference capabilities in the constrained environment.

Use Cases of TensorFlow Lite for Microcontrollers

Speech Recognition:

Wake-word detection ("Hey Google", "Alexa").

Keyword spotting on embedded devices.

Gesture Recognition:

Using accelerometer/gyroscope data for gesture classification.

Image Processing:

Object detection and classification using small models like MobileNet.

Predictive Maintenance:

Anomaly detection in IoT sensors (e.g., vibration or temperature sensors).

Health Monitoring:

ECG and heart rate analysis using wearables.

Environmental Monitoring:

Monitoring air quality, noise levels, or other metrics in smart devices.

Why Use TensorFlow Lite for Microcontrollers?

Efficiency: Tailored for devices with extremely limited resources.

Portability: Works across a wide variety of microcontroller architectures.

Open Source: Fully open-source and actively maintained by TensorFlow.

Ecosystem Support: Provides pre-built examples, extensive documentation, and integration with popular development tools.

Limitations

Limited Operations: Not all TensorFlow operations are supported due to resource constraints.

Manual Optimization: Often requires model optimization, including quantization and pruning.

Resource Constraints: May not support larger or more complex models.

In short, TensorFlow Lite for Microcontrollers (or Tiny Micro) brings the power of machine learning to even the smallest devices, enabling smart, on-device ML applications in environments with extreme resource limitations.