Fundamentals

Machine Learning Fundamentals

MVP Accelerator

Fundamentals

Machine Learning Fundamentals

TensorFlow Lite for Microcontrollers is a framework that provides a set of tools for running neural network inference on microcontrollers. It contains a wide selection of kernel operators with good support for 8-bit integer quantized networks. The framework is limited to model inference and does not support training. For information about how to train a neural network, see the Silicon Labs Machine Learning Toolkit (MLTK).

Silicon Labs provides an integration of TensorFlow Lite for Microcontrollers with the Simplicity SDK. See the Getting Started Guides for step-by-step instructions on how to make use of Machine Learning in your project.

SDK Component Overview

The software components required to use TensorFlow Lite for Microcontrollers can be found under Machine Learning > TensorFlow in the software component browser UI in the Simplicity Studio project configurator.

TensorFlow Lite Micro

This component contains the full TensorFlow Lite for Microcontrollers framework, and automatically pulls in the most optimal implementation of kernels for the device selected for the project by default. To use TensorFlow Lite Micro, this component is the only one that needs to be explicitly installed. It is however possible to manually install different kernel implementations if so desired, for instance to compare inference performance or code size, and to manually install a different debug logging implementation. By default, the TensorFlow Lite Micro component makes use of the Flatbuffer Converter Tool to convert a `.tflite` file into a C array and to initialize this neural network model automatically. See the section on automatic initialization for more details.

Kernel Implementations

Reference Kernels

This component provides unoptimized software implementations of all kernels. This is a default implementation that is designed to be easy to read and can run on any platform. As a result, these kernels may run more slowly than an optimal implementation.

CMSIS-NN Optimized Kernels

Some kernels have implementations that have been optimized for certain CPU architectures using the CMSIS-NN library. Using these kernels when available can improve inference performance significantly. By enabling this component, the available optimized kernel implementations are added to the project, replacing the corresponding reference kernel implementations. The remaining kernels fall back to using the reference implementations by depending on the reference kernel component.

MVP Accelerated Kernels

Some kernels have implementations optimized for the MVP accelerator available on select Silicon Labs parts. Using these kernels will improve inference performance. By enabling this component, the available accelerated kernel implementations are added to the project, replacing the corresponding optimized or reference kernel implementations. The remaining kernels fall back to use the optimized or reference implementations by depending on the corresponding components. See more details about the accelerator to learn what kernels are supported, and what constraints apply.

Debug Logging using I/O Stream / Disabled

Debug logging is used in TensorFlow to display debug and error information. Additionally, it can be used to display inference results. Debug logging is enabled by default, with an implementation that uses I/O Stream to print over UART to the virtual COM port on development kits (VCOM). Logging can be disabled by ensuring that the component "Debug Logging Disabled" is included in the project.

TensorFlow Third Party Dependencies

A specific version of the CMSIS-NN library is used as with TensorFlow Lite for Microcontrollers to optimize certain kernels. This library is included in the project together with TensorFlow Lite for Microcontrollers. TensorFlow depends on a bleeding-edge version of CMSIS-NN, while the rest of the Simplicity SDK uses a stable CMSIS release. It is strongly recommended to avoid using functions from the Simplicity SDK version of CMSIS-DSP and CMSIS-NN elsewhere in the project and instead use the version bundled with TensorFlow Lite for Microcontrollers to avoid versioning conflicts between the two.

Audio Feature Generator

The audio feature generator can be used to extract time-frequency features from an audio signal for use with machine learning (ML) audio classification applications. The generated feature array is a mel-scaled spectrogram, representing the frequency information of the signal of a given sample length of audio.

When used together with the Flatbuffer Converter Tool, the audio feature generator by default consumes its configuration settings from the model parameters of the `.tflite` flatbuffer. Such metadata can be added to the flatbuffer by using the Silicon Labs Machine Learning Toolkit. This ensures that the settings used during inference on the embedded device match the settings used during training. If models without such metadata are used, the configuration option "Enable Manual Frontend Configurations" can be enabled, and configuration values set in the configuration header `sl_ml_audio_feature_generation_config.h`.

Automatic Initialization of Default Model

When the TensorFlow Lite Micro component is added to the project, it will by default attempt to automatically initialize a default model using the TFLite Micro Init API. It performs initialization of TensorFlow Lite Micro by creating an opcode resolver and interpreter for the given flatbuffer. In addition, it creates the tensor arena buffer.

The model used by the automatic initialization code comes from the Flatbuffer Converter Tool. If the flatbuffer was produced using the MLTK, it may contain metadata about the necessary tensor arena size. If such information is present, it will be automatically initialized to the correct size. If a non-MLTK flatbuffer is used, the tensor arena size must be configured manually using the configuration file for the TensorFlow Lite Micro component.

If automatic initialization at startup is not desired, this can be turned off using the Automatically initialize model (`SL_TFLITE_MICRO_INTERPRETER_INIT_ENABLE`) configuration option.

Version

The Silicon Labs AI/ML extension incorporates TensorFlow Lite for Microcontrollers version #02414075e7f718a2d0412775fcadbf28fb4cc5aa in `third_party/tflite-micro/`. The core TensorFlow Lite for Microcontrollers offering is unpatched, all additional content for Silicon Labs devices is delivered in the extension's root directory.

Third-party Tools and Partners

Tools

- Netron is a visualization tool for neural networks, compatible with `.tflite` model files. This is useful for viewing the operations used in a model, the sizes of tensors and kernels, etc.

AI/ML Partners

Silicon Labs AI/ML partners provide expertise and platforms for data collection, model development, and training. See the technology partner pages to learn more.

MVP Accelerator

MVP Accelerator

The MVP accelerator is a co-processor designed to perform matrix and vector operations. Using hardware accelerated kernel implementations will reduce neural network inference time, as well as off-load the main processor to allow it to perform other tasks or go to sleep.

Silicon Labs has implemented common neural network operators as programs to be executed on the MVP and integrated these with TensorFlow Lite for Microcontrollers. The MVP has 5 array controllers, each of which can support iterating in 3 independent dimensions. Each dimension is limited to 1024 elements, with a stride between each element of 2047. The limiting factor for most neural network operations is therefore the product of the width and depth dimensions, since this becomes the stride in the height dimension.

All MVP-accelerated operations take signed 8-bit integers as input and output. If the inner dimension of the tensor has even size, each element can contain two int8 values, interpreted as a single complex int8 value by the accelerator. The accelerator can then effectively support 2048 int8 values. If the inner dimension is odd, the accelerator must perform one computation at a time, which reduces performance and limits the dimension size to 1024 int8 values.

The operators listed below will be accelerated using the MVP if tensor sizes allow. If a specific tensor cannot be accelerated, the implementation will automatically fall back to using optimized (CMSIS-NN) or reference kernel implementations at runtime. To maximize the likelihood that an operator is supported by the accelerator, use even-valued numbers of channels when designing the model.

Internally, the MVP accelerator uses 16-bit floating point math, even when taking 8-bit integers as input. This means that there is a slight reduction in accuracy of computations, which may be especially noticeable when performing operations that accumulate many elements.

For more information about the MVP hardware accelerator, see the reference manual for EFR32xG24.

Accelerated TensorFlow operators

Add

TensorFlow operator name: `ADD`

Any tensor size is supported.

FullyConnected (Dense)

TensorFlow operator name: `FULLY_CONNECTED` , `FULLY_CONNECTED_INT8`

Supports tensors where all dimensions are within the 1024 element limit. Also supports larger tensors where the size of the last dimension is decomposable into two factors that are both within the 1024 element limit.

AveragePool2D

TensorFlow operator name: `AVERAGE_POOL_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit.

MaxPool2D

TensorFlow operator name: `MAX_POOL_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit.

Conv2D

TensorFlow operator name: `CONV_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit.

DepthwiseConv2D

TensorFlow operator name: `DEPTHWISE_CONV_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit. Dilation is not supported.

TransposeConv2D

TensorFlow operator name: `TRANSPOSE_CONV_2D`

Supports tensors where `width*channels` is within the 2047 element stride limit and all dimensions are within the 1024 element limit. Dilation is not supported.

Suspending Execution While Waiting for Accelerator

The software API for the MVP accelerator is blocking, meaning that any call to the MVP driver will wait for completion before returning from the function call. To save energy, the driver can optionally suspend execution of the main processor while waiting for the accelerator to complete an operation. By default, the main processor busy-waits for the accelerator.

No sleep (0)

When the "No sleep" option is used, the MCU core will busy-wait for the MVP to finish. This is the option which provides the fastest MVP execution time. The "No sleep" option can be used in a bare metal application or an application using a real-time operating system (RTOS).

Enter EM1 (1)

When the "Enter EM1" option is used, the MCU will be put into Energy Mode 1 whenever the driver waits for an MVP program to complete. The "Enter EM1" option is not safe to use in an application using RTOS because it will prevent proper RTOS scheduling.

Yield RTOS thread (2)

When the "Yield RTOS thread" option is used, the task waiting for the MVP program to complete will yield, allowing other tasks in the system to run or potentially let the scheduler put the system into a sleep mode. The "Yield RTOS thread" requires that the application is using an RTOS.

The power mode of the MVP driver can be configured by setting the `SL_MVP_POWER_MODE` configuration option in the `sl_mvp_config.h` configuration header.