



# EIQTFLITEUG

elQ TensorFlow Lite Library User's Guide

Rev. 6 — 1 June 2022

User guide

## Document information

Information	Content
Keywords	elQ, TensorFlow, Lite, TensorFlow Lite
Abstract	This document describes the steps required to download/use the library and create an application for running pre-trained models.



## 1 Overview

---

TensorFlow Lite is an open source software library for running machine learning models on mobile and embedded devices. For more information, see [www.tensorflow.org/lite](http://www.tensorflow.org/lite).

For memory constrained devices, the library contains TensorFlow Lite for Microcontrollers. For more information, see [www.tensorflow.org/lite/microcontrollers](http://www.tensorflow.org/lite/microcontrollers).

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated TensorFlow Lite for Microcontrollers based on TensorFlow Lite version 2.2.0 (from the 16th of February 2022). This document describes the steps required to download and start using the library. Additionally, the document describes the steps required to create an application for running pre-trained models.

**Note:** *The document also assumes knowledge of machine learning frameworks for model training.*

## 2 Deployment

---

The eIQ TensorFlow Lite for Microcontrollers library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on [mcuxpresso.nxp.com](http://mcuxpresso.nxp.com). To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See [Figure 1](#).

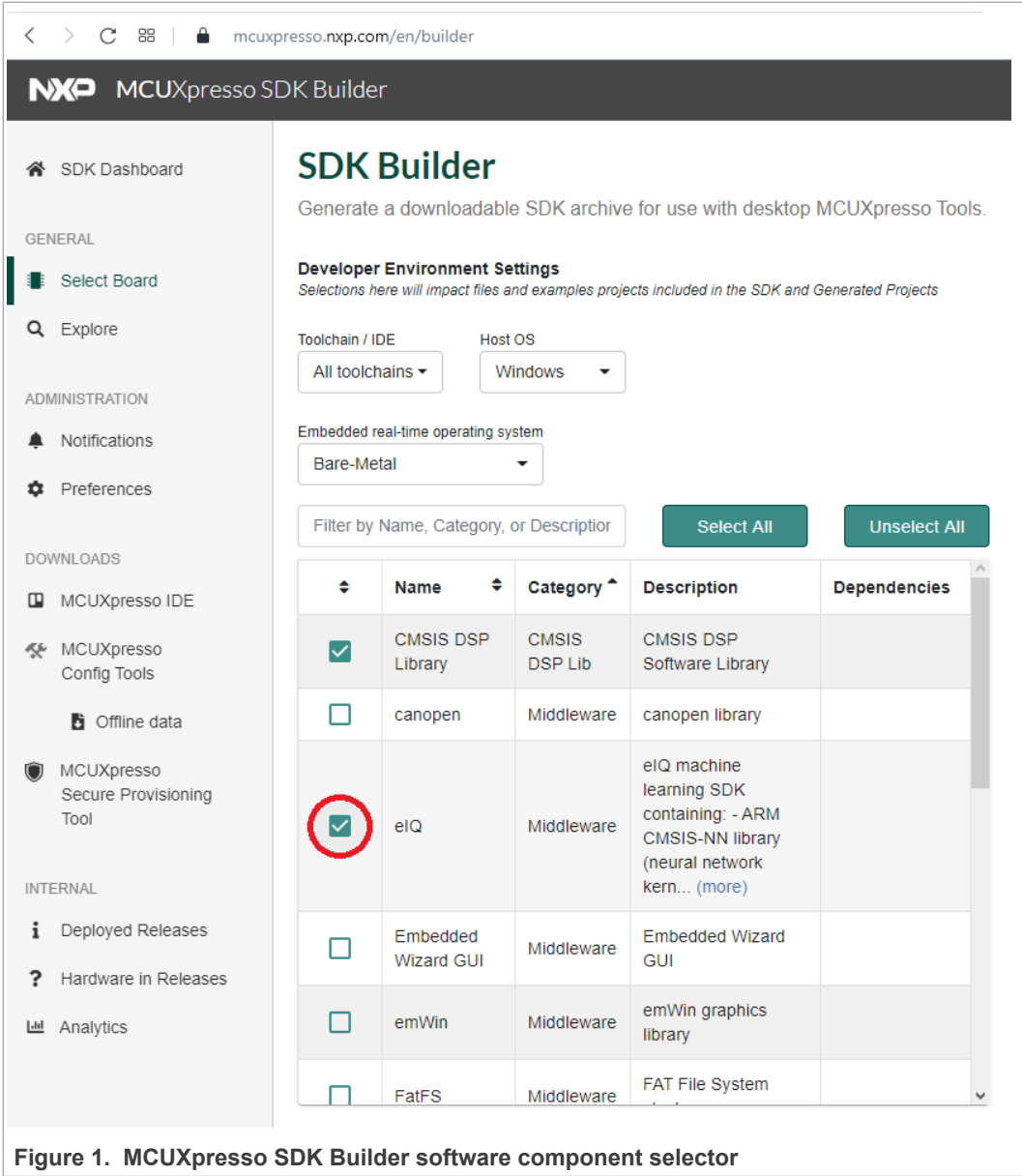


Figure 1. MCUXpresso SDK Builder software component selector

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to [Figure 2](#). The eIQ TensorFlow Lite library directories are highlighted in red.

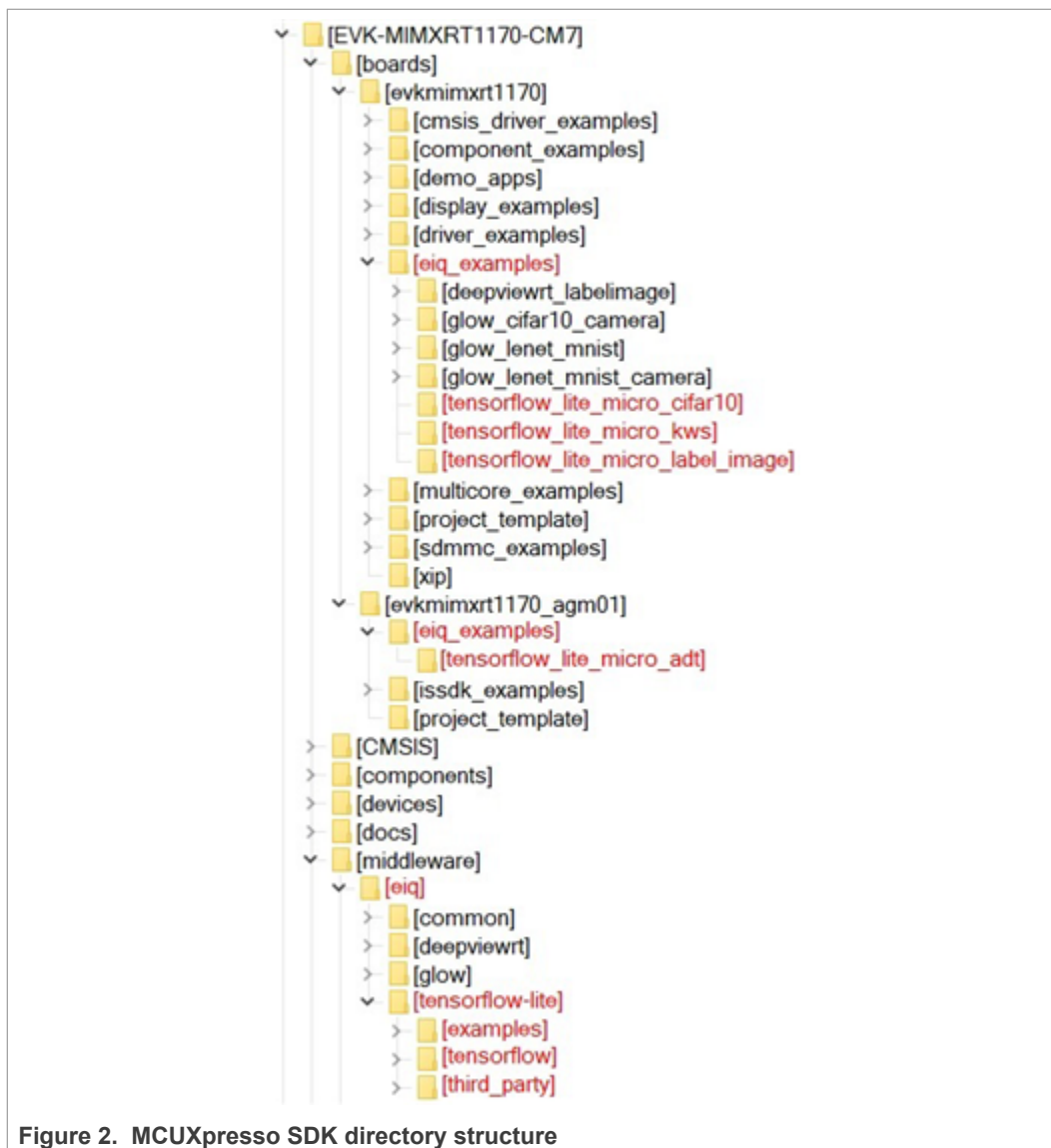


Figure 2. MCUXpresso SDK directory structure

The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

### 3 Example applications

The eIQ TensorFlow Lite library is provided with a set of example applications. For details, see [Table 1](#). The applications demonstrate the usage of the library in several use cases.

Table 1. List of example applications

Name	Description	Availability
tensorflow_lite_micro_adt	Anomaly detection application using an autoencoder model. The example requires external development kit FRDMSTBC-AGM01 with sensors.	EVKB-IMXRT1050-AGM01 EVK-MIMXRT1060-AGM01 EVK-MIMXRT1064-AGM01 EVK-MIMXRT1160-AGM01 EVK-MIMXRT1170-AGM01 MIMXRT1060-EVKB-AGM01
tensorflow_lite_micro_cifar10	CIFAR-10 classification of 32 × 32 RGB pixel images into 10 categories using a small Convolutional Neural Network (CNN).	EVKB-IMXRT1050 EVK-MIMXRT1060 EVK-MIMXRT1064 EVK-MIMXRT1160 EVK-MIMXRT1170 EVK-MIMXRT595 (no camera and display support) EVK-MIMXRT685 (no camera and display support) MIMXRT1060-EVKB MIMXRT685-AUD-EVK (no camera and display support)
tensorflow_lite_micro_kws	Keyword spotting application using a neural network for word detection in pre-processed audio input.	EVKB-IMXRT1050 EVK-MIMXRT1060 EVK-MIMXRT1064 EVK-MIMXRT1160 EVK-MIMXRT1170 EVK-MIMXRT595 EVK-MIMXRT685 MIMXRT1060-EVKB MIMXRT685-AUD-EVK
tensorflow_lite_micro_label_image	Image recognition application using a MobileNet model architecture to classify 128 × 128 RGB pixel images into 1000 categories.	EVKB-IMXRT1050 EVK-MIMXRT1060 EVK-MIMXRT1064 EVK-MIMXRT1160 EVK-MIMXRT1170 EVK-MIMXRT595 (no camera and display support) EVK-MIMXRT685 (no camera and display support) MIMXRT1060-EVKB MIMXRT685-AUD-EVK (no camera and display support)
tensorflow_lite_micro_multicore	Image recognition application running on Cortex-M7 core with wake up from keyword spotting application running on Cortex-M4 core.	EVK-MIMXRT1160 EVK-MIMXRT1170
tensorflow_lite_micro_xaf	Keyword spotting application using the Xtensa Audio Framework for configurable audio pipeline components management.	EVK-MIMXRT595 EVK-MIMXRT685 MIMXRT685-AUD-EVK

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG). When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in [Figure 3](#).

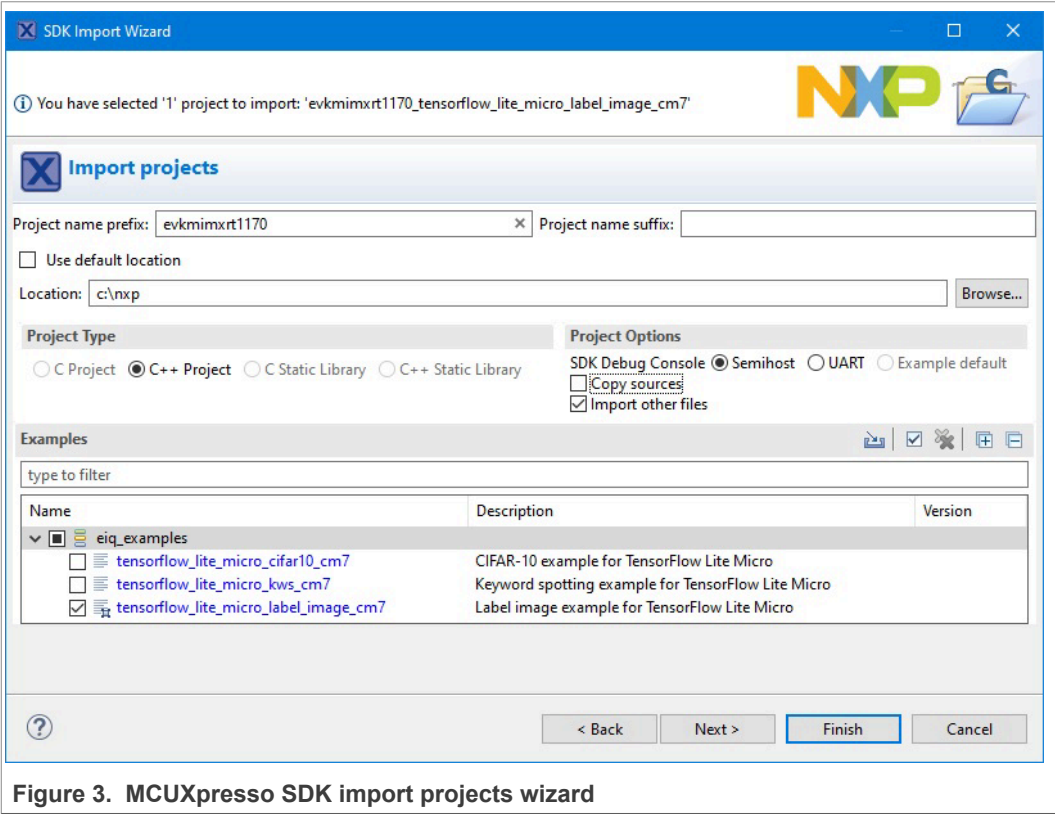


Figure 3. MCUXpresso SDK import projects wizard

After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, [Figure 4](#) shows the output of the `tensorflow_lite_micro_label_image_cm7` example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.

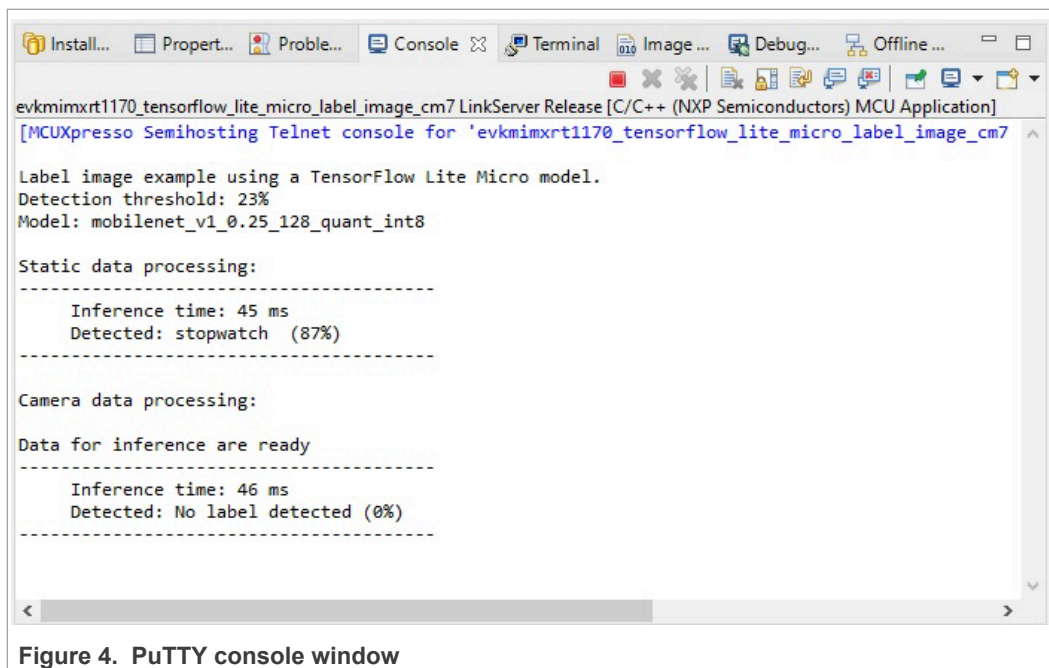


Figure 4. PuTTY console window

## 4 Comparison

The TensorFlow Lite library since version 2.3 provides an alternative implementation optimized for microcontrollers with low memory capacity called TensorFlow Lite for Microcontrollers (or TensorFlow Lite Micro). In comparison to TensorFlow Lite, the Micro version uses static memory allocation, has no dependencies on standard C or C++ libraries and contains implementations of operation kernels optimized for Arm Cortex-M architecture using Arm's CMSIS-NN library. The following table contains a comparison of supported operations by both libraries.

Table 2. Supported operations

TensorFlow Lite operations	Supported by TensorFlow Lite for Microcontrollers
ABS	Yes
ADD	Yes
ADD_N	Yes
ARG_MAX	Yes
ARG_MIN	Yes
AUDIO_SPECTROGRAM	No
AVERAGE_POOL_2D	Yes
BATCH_MATMUL	No
BATCH_TO_SPACE_ND	Yes
BIDIRECTIONAL_SEQUENCE_LSTM	No
BIDIRECTIONAL_SEQUENCE_RNN	No
CAST	Yes
CEIL	Yes
CONCATENATION	Yes
CONV_2D	Yes
COS	Yes

Table 2. Supported operations...continued

TensorFlow Lite operations	Supported by TensorFlow Lite for Microcontrollers
DENSIFY	No
DEPTH_TO_SPACE	Yes
DEPTHWISE_CONV_2D	Yes
DEQUANTIZE	Yes
DETECTION_POSTPROCESS	Yes
DIV	Yes
ELU	Yes
EMBEDDING_LOOKUP	No
EMBEDDING_LOOKUP_SPARSE	No
EQUAL	Yes
EXP	Yes
EXPAND_DIMS	Yes
FAKE_QUANT	No
FILL	Yes
FLOOR	Yes
FLOOR_DIV	Yes
FLOOR_MOD	Yes
FULLY_CONNECTED	Yes
GATHER	Yes
GATHER_ND	Yes
GREATER	Yes
GREATER_EQUAL	Yes
HARD_SWISH	Yes
HASHTABLE_LOOKUP	No
IF	Yes
L2_NORMALIZATION	Yes
L2_POOL_2D	Yes
LEAKY_RELU	Yes
LESS	Yes
LESS_EQUAL	Yes
LOCAL_RESPONSE_NORMALIZATION	No
LOG	Yes
LOG_SOFTMAX	No
LOGICAL_AND	Yes
LOGICAL_NOT	Yes
LOGICAL_OR	Yes
LOGISTIC	Yes
LSH_PROJECTION	No
LSTM	No
MATRIX_DIAG	No
MATRIX_SET_DIAG	No
MAX_POOL_2D	Yes
MAXIMUM	Yes
MEAN	Yes
MFCC	No



Table 2. Supported operations...continued

TensorFlow Lite operations	Supported by TensorFlow Lite for Microcontrollers
MINIMUM	Yes
MIRROR_PAD	Yes
MUL	Yes
NEG	Yes
NON_MAX_SUPPRESSION_V4	No
NON_MAX_SUPPRESSION_V5	No
NOT_EQUAL	Yes
NUMERIC_VERIFY	No
ONE_HOT	No
PACK	Yes
PAD	Yes
PADV2	Yes
POW	No
PRELU	Yes
QUANTIZE	Yes
RANGE	No
RANK	No
REDUCE_ANY	No
REDUCE_MAX	Yes
REDUCE_MIN	No
REDUCE_PROD	No
RELU	Yes
RELU_N1_TO_1	No
RELU6	Yes
RESHAPE	Yes
RESIZE_BILINEAR	Yes
RESIZE_NEAREST_NEIGHBOR	Yes
REVERSE_SEQUENCE	No
REVERSE_V2	No
RNN	No
ROUND	Yes
RSQRT	Yes
SCATTER_ND	No
SEGMENT_SUM	No
SELECT	No
SELECT_V2	No
SHAPE	Yes
SIN	Yes
SKIP_GRAM	No
SLICE	Yes
SOFTMAX	Yes
SPACE_TO_BATCH_ND	Yes
SPACE_TO_DEPTH	Yes
SPARSE_TO_DENSE	No
SPLIT	Yes

Table 2. Supported operations...continued

TensorFlow Lite operations	Supported by TensorFlow Lite for Microcontrollers
SPLIT_V	Yes
SQRT	Yes
SQUARE	Yes
SQUARED_DIFFERENCE	No
SQUEEZE	Yes
STRIDED_SLICE	Yes
SUB	Yes
SUM	No
SVDF	Yes
TANH	Yes
TILE	No
TOPK_V2	No
TRANSPOSE	Yes
TRANSPOSE_CONV	Yes
UNIDIRECTIONAL_SEQUENCE_LSTM	Yes (on Xtensa cores)
UNIDIRECTIONAL_SEQUENCE_RNN	No
UNIQUE	No
UNPACK	Yes
WHERE	No
WHILE	No
ZEROS_LIKE	Yes

[Table 3](#) contains an overview of hardware optimized TensorFlow Lite Micro operators on supported devices. Operators not listed in the table are reference C++ implementations only. Optimized operators for ARM Cortex-M cores leverage the ARM CMSIS-NN library. For details, see the `middleware/eiq/tensorflow-lite/third_party/cmsis/CMSIS/NN/README.md` file. Optimized operators for Cadence Xtensa cores (HiFi4 and FusionF1) leverage the Xtensa HiFi4 NN library.

Table 3. Hardware optimized TFLM operators

Operator	Operator input type	i.MX RT685 (HiFi4 core)	i.MX RT595 (FusionF1 core)	i.MX RT1050 i.MX RT1060 i.MX RT1064 i.MX RT1160 i.MX RT1170 i.MX RT595 (Cortex-M33 core) i.MX RT685 (Cortex-M33 core)
ADD	float	No	No	No
	uint8 (PTQ)	No	No	No
	int8 (PCQ)	No	No	Yes
AVERAGE_POOL_2D	float	No	No	No
	uint8 (PTQ)	No	No	No
	int8 (PCQ)	No	No	Yes

Table 3. Hardware optimized TFLM operators...continued

Operator	Operator input type	i.MX RT685 (HiFi4 core)	i.MX RT595 (FusionF1 core)	i.MX RT1050 i.MX RT1060 i.MX RT1064 i.MX RT1160 i.MX RT1170 i.MX RT595 (Cortex-M33 core) i.MX RT685 (Cortex-M33 core)
CONV_2D	float	No	No	No
	uint8 (PTQ)	No	No	No
	int8 (PCQ)	Yes	Yes	Yes
DEPTHWISE_CONV_2D	float	No	No	No
	uint8 (PTQ)	No	No	Yes
	int8 (PCQ)	Yes	Yes	Yes
FULLY_CONNECTED	float	No	No	No
	uint8 (PTQ)	No	No	No
	int8 (PCQ)	Yes	Yes	Yes
MAX_POOL_2D	float	No	No	No
	uint8 (PTQ)	No	No	No
	int8 (PCQ)	No	No	Yes
MUL	float	No	No	No
	uint8 (PTQ)	No	No	No
	int8 (PCQ)	No	No	Yes
SOFTMAX	float	No	No	No
	uint8 (PTQ)	No	No	No
	int8 (PCQ)	Yes	Yes	Yes
SVDF	float	No	No	No
	uint8 (PTQ)	N/A	N/A	N/A
	int8 (PCQ)	Yes	Yes	Yes

**Note:**

- PTQ — Per-tensor quantized (asymmetric 8-bit quantization).
- PCQ — Per-channel quantized (symmetric 8-bit quantization).

## 5 Converting a model

TensorFlow contains a conversion tool and a Python API for converting TensorFlow models stored in the Protocol Buffers format (.pb) or similar alternatives to TensorFlow Lite models stored in the FlatBuffers format (.tflite). Therefore, to convert a model to the TensorFlow Lite format, the user must first install TensorFlow. For more information, see [www.tensorflow.org/lite/convert](http://www.tensorflow.org/lite/convert).

Use the following steps:

1. On Linux OS (Ubuntu 16.04 or later (64-bit)):
  - a. Install Python using the command:

- ```
sudo apt-get install python3
```
- b. Install TensorFlow with Python *pip* package manager: `pip3 install --user tensorflow==2.8.0`
  2. On macOS (10.12.6 (Sierra) or later (64-bit)):
    - a. Install Python (version 3.9.x) using an installer downloaded from [www.python.org/downloads/mac-osx](http://www.python.org/downloads/mac-osx).
    - b. Install TensorFlow with Python *pip* package manager from the command line: `pip3 install --user tensorflow==2.8.0`
    - c. Update the system path to include the user's Library/Python/3.9/bin directory: `export PATH="$HOME/Library/Python/3.9/bin:$PATH"`
  3. On Windows OS (Windows OS 7 or later (64-bit)):
    - a. Install Python (version 3.9.x) using an installer downloaded from <https://www.python.org/downloads/windows>.
    - b. Update the Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019 to the latest version available at <https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads>.
    - c. Install TensorFlow with Python *pip* package manager: `pip install --user tensorflow==2.8.0`

**Note:** Detailed instructions on how to install TensorFlow can be found at [www.tensorflow.org/install](http://www.tensorflow.org/install). The page also contains a list of all supported platforms and alternative installation methods.

**Note:** It is out of the document scope to describe the process of creating and training a model. To learn how to obtain a trained model, see the tutorials and reference manuals of machine learning frameworks. For more information, see <https://www.tensorflow.org/tutorials/images/cnn>.

The following command in [Section 5.1](#) converts a trained model stored in the `mobilenet_v1_0.25_128_frozen.pb` file (from the package available at [download.tensorflow.org/models/mobilenet\\_v1\\_2018\\_08\\_02/mobilenet\\_v1\\_0.25\\_128.tgz](http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_0.25_128.tgz)) to a TensorFlow Lite model file `mobilenet_v1_0.25_128.tflite` performing floating-point inference. The `input` and `MobilenetV1/Predictions/Reshape_1` parameter values are the input and output node names.

## 5.1 Converting a model performing floating-point inference

```
tflite_convert \
--output_file=mobilenet_v1_0.25_128.tflite \
--graph_def_file=mobilenet_v1_0.25_128_frozen.pb \
--input_arrays=input \
--output_arrays=MobilenetV1/Predictions/Reshape_1 \
--enable_v1_converter
```

The description of the parameters can be displayed by running `tflite_convert -help`.

The names of the input and output nodes can be retrieved from a model visualizer like <https://lutzroeder.github.io/netron/>, for example. By clicking the first node in the displayed graph, its name is shown in the properties panel. This is typically the input node. Similarly, by clicking the last node, the name of the output node can be retrieved.

**Note:** TensorFlow Lite supports only a subset of TensorFlow operations. During the conversion process, some of those operations might be deleted or fused. Since the set of TensorFlow Lite operations is smaller than TensorFlow's, not every model is convertible. If the original TensorFlow model uses an operation not supported by TensorFlow Lite,

the converter reports it as an error. For more information, see the guide at [https://www.tensorflow.org/lite/guide/ops\\_compatibility](https://www.tensorflow.org/lite/guide/ops_compatibility).

## 5.2 Quantization

Quantization is a technique to reduce the model size while maintaining only small degradation of the model precision. Typically, quantization reduces the number of bits associated with weights and activations from 32-bit floating-point values to 8-bits. This yields a 4x reduction in model size. Furthermore, processors with fixed-point SIMD instructions can leverage these for faster computations.

Probably the best way to perform model quantization is [quantization-aware training](#). The model can be afterward converted using the converter, which is compatible with models quantized by TensorFlow. Alternatively, a model can be quantized [post-training](#) using per-channel quantization. The Python script shown in [Section 5.2.1](#) quantizes and converts the floating point model from `mobilenet_v1_0.25_128_frozen.pb` (from the package available at [download.tensorflow.org/models/mobilenet\\_v1\\_2018\\_08\\_02/mobilenet\\_v1\\_0.25\\_128.tgz](https://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_0.25_128.tgz)) to a TensorFlow Lite model `mobilenet_v1_0.25_128_quant_int8.tflite`. The input and output node names can be retrieved using a model visualizer like [Netron](#). The quantization is performed using images from the training dataset, which the script expects to be stored in the `dataset` directory.

### 5.2.1 Quantizing and converting a model

```
import os
import numpy as np
from PIL import Image
import tensorflow as tf
input_mean = 127.5
input_std = 127.5
converter = tf.compat.v1.lite.TFLiteConverter.from_frozen_graph(
    'mobilenet_v1_0.25_128_frozen.pb',          # TensorFlow model file
    ['input'],                                  # input node names
    ['MobilenetV1/Predictions/Reshape_1'],      # output node names
    input_shapes={'input': (1, 128, 128, 3)})  # input tensor dimensions
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.float32
def representative_dataset_gen():
    images = []
    for image in os.listdir('dataset'):
        image = os.path.join('dataset', image)
        if os.path.isfile(image):
            image = np.array(Image.open(image).resize((128, 128)))
            image = (image - input_mean) / input_std
            image = image.astype(np.float32)
            image = image.reshape((1, 128, 128, 3))
            images.append(image)
    for image in images:
        yield [image]
converter.representative_dataset = representative_dataset_gen
tflite_quant_model = converter.convert()
with open('mobilenet_v1_0.25_128_quant_int8.tflite', 'wb') as f:
    f.write(tflite_quant_model)
```

## 6 Running an inference

After converting the model to the TensorFlow Lite format, it is converted into a C language array to include it in the application source code. The `xxd` utility can be used for

this purpose (distributed with the *Vim* editor for many platforms on <https://www.vim.org/>) as shown in [Section 6.1](#). The utility converts a TensorFlow Lite model into a C header file with an array definition containing the binary image of the model and a variable containing the data size.

## 6.1 Converting a model to a C language header file

```
xxd -i mobilenet_v1_0.25_128_quant.tflite > mobilenet_v1_0.25_128_quant_model.h
```

After the header file is generated, the type of the array is changed from `unsigned char` to `const char` to match the library API input parameters and the default array name can be changed to a more convenient one. The user must align the buffer to at least 64-bit boundary (the size of a double precision floating-point number) to avoid misaligned memory access. The alignment can be achieved by using the `__ALIGNED(16)` macro from the `cmsis_compiler.h` header file (available in the MCUXpresso SDK) in the array declaration before the data assignment.

The easiest way to create an application with the proper configuration is to copy and modify an existing example application. To learn where to find the example applications and how to build them, see the [Section 3](#).

Running an inference using TensorFlow Lite for Microcontrollers involves several steps (shown for quantized model with signed 8-bit values as input and 32-floating point values as output):

1. Include the necessary eIQ TensorFlow Lite Micro library header files and the converted model.

### Including header files

```
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/version.h"
#include "mobilenet_v1_0.25_128_quant_model.h"
```

2. Allocate a static memory buffer for input and output tensors and intermediate arrays. Load the FlatBuffer model image (assuming the `mobilenet_v1_0.25_128_quant_model.h` file generated in [Section 6.1](#) defines an array named `mobilenet_model` and a size variable named `mobilenet_model_len`), build the interpreter object and allocate memory for tensors.

### Loading the FlatBuffer model

```
constexpr int kTensorArenaSize = 1024 * 1024;
static uint8_t tensorArena[kTensorArenaSize];
const tflite::Model* model = tflite::GetModel(mobilenet_model);
// TODO: Report an error if model->version() != TFLITE_SCHEMA_VERSION
static tflite::AllOpsResolver microOpResolver;
static tflite::MicroErrorReporter microErrorReporter;
static tflite::MicroInterpreter interpreter(model,
    microOpResolver, tensorArena, kTensorArenaSize,
    microErrorReporter);
interpreter->AllocateTensors();
// TODO: Check return value for kTfLiteOk
```

3. Fill-in the input data into the input tensor. For example, if a speech recognition model, image data from a camera or audio data from a microphone. The dimensions of the input data must be the same as the dimensions of the input tensor. These dimensions were specified when the model was created.

### Filling-in input data

```
// Get access to the input tensor data
TfLiteTensor* inputTensor = interpreter->input(0);
```

```
// Copy the input tensor data from an application buffer
for (int i = 0; i < inputTensor->bytes; i++)
    inputTensor->data.int8[i] = input_data[i];
```

4. Run the inference and read the output data from the output tensor. The dimensions of the output data must be the same as the dimensions of the output tensor. These dimensions were specified when the model was created.

#### Running inference and reading output data

```
// Run the inference
interpreter->Invoke();
// TODO: Check the return value for TfLiteOk
// Get access to the output tensor data
TfLiteTensor* outputTensor = interpreter->output(0);
// Copy the output tensor data to an application buffer
for (int i = 0; i < outputTensor->bytes / sizeof(float32); i++)
    output_data[i] = outputTensor->data.f[i];
```

## 7 Code size optimization

Typically, models do not use all the operators that are available in TensorFlow Lite. However, because of the default operator registration mechanism used in the library, the toolchain linker is not able to remove the code of unused operators. In order to reduce code size, it is possible to only register the specific operators used by a model. To determine which operators are used by a particular model, a model visualizer tool like Netron can be used. Then a mutable operator resolver object can be created that only registers the operators that are used by the model being inferred.

### 7.1 Register all operators

Use the `tflite::AllOpsResolver` object class, which is later passed to the `tflite::MicroInterpreter` object. Make sure to include the `tensorflow/lite/micro/all_ops_resolver.h` header file.

#### Register all available operators in TensorFlow Lite Micro

```
#include "tensorflow/lite/micro/all_ops_resolver.h"
tflite::AllOpsResolver microOpResolver;
static tflite::MicroInterpreter interpreter(
    model, microOpResolver, tensorArena, kTensorArenaSize,
    microErrorReporter);
```

### 7.2 Register only used operators

Use the `tflite::MicroMutableOpResolver` object template, which is later passed to the `tflite::MicroInterpreter` object. Depending on the list of used operators, the result should be similar to [Section 7.2.1](#). Make sure to update the `MicroMutableOpResolver` template parameter to reflect the number of operators that need to be registered.

#### 7.2.1 Register only used operators in TensorFlow Lite Micro

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
tflite::MicroMutableOpResolver<6> microOpResolver;
microOpResolver.AddAveragePool2D();
```

```
microOpResolver.AddConv2D();  
microOpResolver.AddDepthwiseConv2D();  
microOpResolver.AddDequantize();  
microOpResolver.AddReshape();  
microOpResolver.AddSoftmax();  
static tflite::MicroInterpreter interpreter(  
    model, microOpResolver, tensorArena, kTensorArenaSize,  
    microErrorReporter);
```

## 8 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2019 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 9 Revision history

[Table 4](#) summarizes the changes done to this document since the initial release.

Table 4. Revision history

| Revision number | Date             | Substantive changes                                                                                                                         |
|-----------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 0               | 12/2019          | Initial release                                                                                                                             |
| 1               | 04/2020          | Updated to TensorFlow Lite 2.1                                                                                                              |
| 2               | 17 December 2020 | Updated TensorFlow Lite to version 2.3.0 with TensorFlow Lite for Microcontrollers                                                          |
| 3               | 10 July 2021     | Removed TensorFlow Lite in favor of TensorFlow Lite for Microcontrollers.<br>Updated TensorFlow Lite for Microcontrollers to version 2.4.1. |



Table 4. Revision history...continued

| Revision number | Date             | Substantive changes                                                                              |
|-----------------|------------------|--------------------------------------------------------------------------------------------------|
| 4               | 22 October 2021  | Updated TensorFlow Lite for Microcontrollers to version 2.5.0 targeted for i.MX RT1170 CM7 core. |
| 5               | 06 December 2021 | Updated TensorFlow Lite for Microcontrollers to version 2.6.0.                                   |
| 6               | 01 June 2022     | Updated TensorFlow Lite for Microcontrollers to version 22-02-16.                                |

## 10 Legal information

### 10.1 Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

### 10.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <http://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at [PSIRT@nxp.com](mailto:PSIRT@nxp.com)) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

### 10.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

## Contents

---

|           |                                                              |           |
|-----------|--------------------------------------------------------------|-----------|
| <b>1</b>  | <b>Overview .....</b>                                        | <b>2</b>  |
| <b>2</b>  | <b>Deployment .....</b>                                      | <b>2</b>  |
| <b>3</b>  | <b>Example applications .....</b>                            | <b>4</b>  |
| <b>4</b>  | <b>Comparison .....</b>                                      | <b>7</b>  |
| <b>5</b>  | <b>Converting a model .....</b>                              | <b>11</b> |
| 5.1       | Converting a model performing floating-point inference ..... | 12        |
| 5.2       | Quantization .....                                           | 13        |
| 5.2.1     | Quantizing and converting a model .....                      | 13        |
| <b>6</b>  | <b>Running an inference .....</b>                            | <b>13</b> |
| 6.1       | Converting a model to a C language header file .....         | 14        |
| <b>7</b>  | <b>Code size optimization .....</b>                          | <b>15</b> |
| 7.1       | Register all operators .....                                 | 15        |
| 7.2       | Register only used operators .....                           | 15        |
| 7.2.1     | Register only used operators in TensorFlow Lite Micro .....  | 15        |
| <b>8</b>  | <b>Note about the source code in the document .....</b>      | <b>16</b> |
| <b>9</b>  | <b>Revision history .....</b>                                | <b>16</b> |
| <b>10</b> | <b>Legal information .....</b>                               | <b>18</b> |

---

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

---

© NXP B.V. 2022.

All rights reserved.

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

Date of release: 1 June 2022

Document identifier: EIQTFLITEUG