EIQGLOWAOTUG

eIQ Glow Ahead of Time User Guide

Rev. 1 — 14 April 2020

User's Guide

Contents

1 Overview

Glow is a machine learning compiler for neural network graphs. It is designed to optimize the neural network graphs and generate code for various hardware devices available at https://github.com/pytorch/glow. Glow comes in two flavors: Just in Time (JIT) and Ahead of Time (AOT) compilations. JIT compilation is performed at runtime just before the model is executed. AOT compilation is performed offline and generates an object file (bundle) which is later linked with the application code.

MCUXpresso Software Development Kit (MCUXpresso SDK) provides a set of helper functions that allows integration of Glow AOT bundles.

The official Glow documentation for building AOT applications can be found here. However, this document provides extra information regarding the NXP deliverable of the Glow compiler which includes some extra features and optimizations.

This document describes steps to download, start using Glow AOT, and create an application that integrates bundles generated using the Glow AOT compiler. The document assumes that the Glow AOT tools are installed on user workstation using the Glow AOT installer and the location is included in the system path. For more information, see *eIQ Glow AOT Windows Installer Guide*.

2 Deployment

The eIQ Glow AOT 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. To include eIQ machine learning into an MCUXpresso SDK package, the eIQ middleware component has to be selected in the software component selector on the SDK Builder when building a new package. For details, see Figure 1.

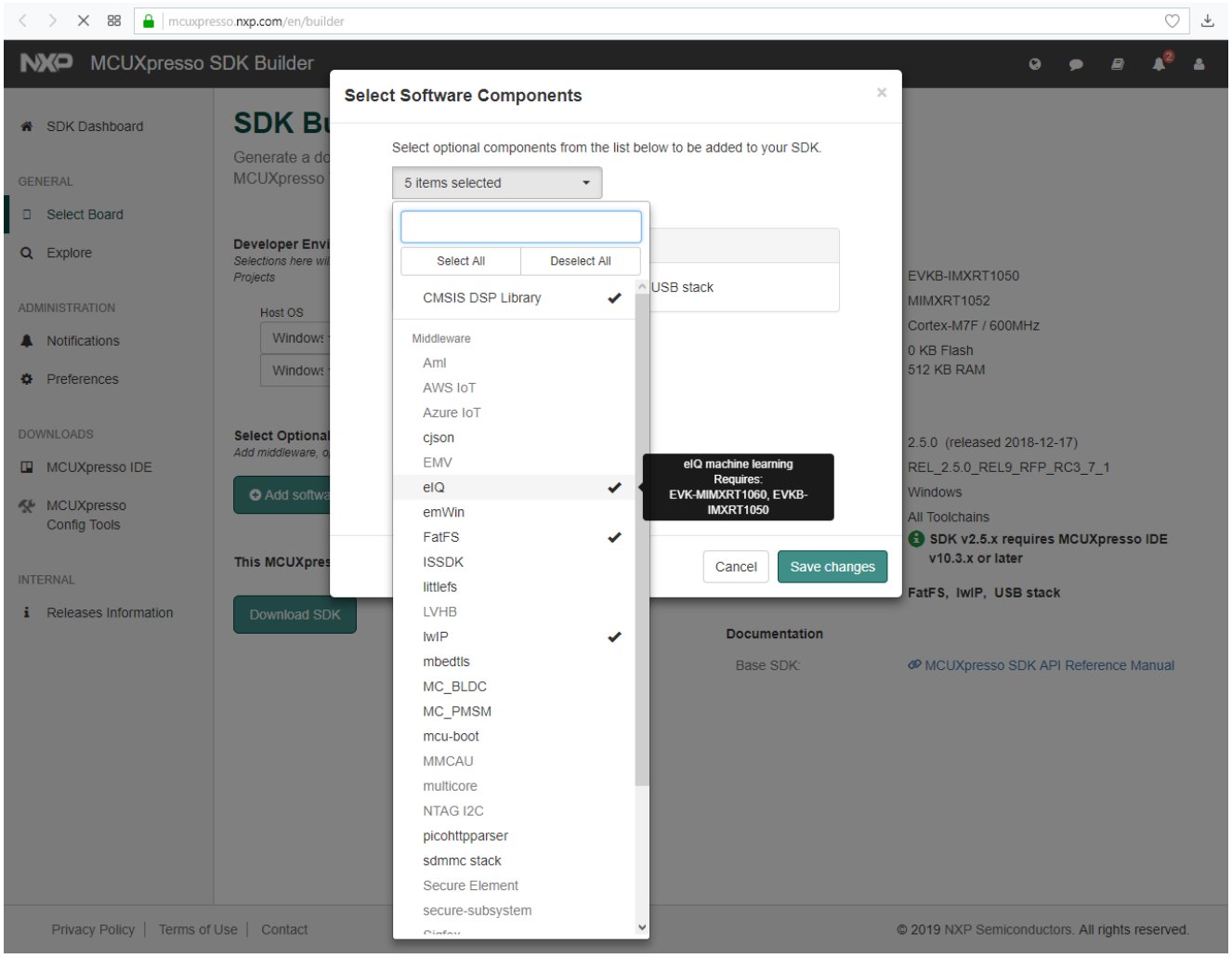

Figure 1. MCUXpresso SDK Builder software component selector

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. To get familiar with the MCUXpresso SDK folder structure, see the *Getting Started with MCUXpresso SDK* document.

The package directory structure might look like Figure 2. The eIQ Glow library directories are highlighted in red.

Figure 2. MCUXpresso EVKB-IMRT1050 SDK structure

The package directory structure might look like Figure 3. The eIQ Glow library directories are highlighted in red.

Figure 3. MCUXpresso EVKB-IMRT1060 SDK structure

The package directory structure might look like Figure 4. The eIQ Glow library directories are highlighted in red.

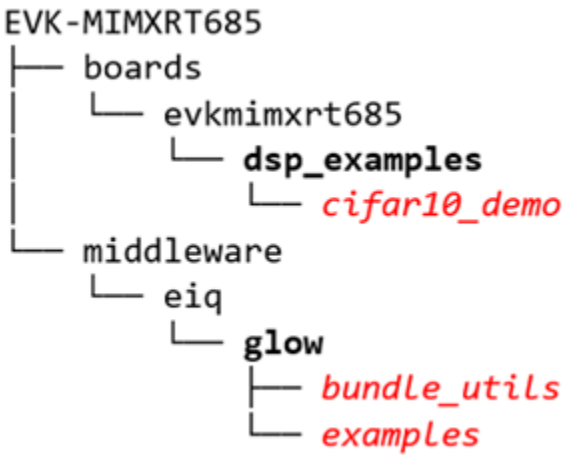

Figure 4. MCUXpresso EVKB-MIMXRT685 SDK structure

The *boards* directory contains example application projects for supported toolchains (for the list of supported toolchains, see the *MCUXpresso SDK Release Notes* document). The *middleware* directory contains the eIQ library source codes, pre-compiled library binaries and example application source codes and data.

NOTE

Installing the Glow AOT tool on the target machine is out of the scope of this document. It is assumed that the tool is already installed on the workstation and its location is included in the system path (see document *eIQ Glow AOT Windows Installer Guide* for more details).

3 Example applications

The eIQ Glow AOT is provided with a set of example applications. For details, see Table 1. The applications demonstrate the usage of the Glow AOT in several use cases.

Table 1. List of example applications

Name	Description
glow_cifar10	CIFAR-10 classification of 32x32 RGB pixel images into 10 categories using a small convolutional neural network (CNN).
glow_lenet_mnist	Performs handwritten digit classification using the LeNet neural network trained on MNIST database.
cifar10_demo	CIFAR-10 classification of 32x32 RGB pixel images into 10 categories using a small convolutional neural network (CNN). This project is using the HiFi-NN firmware.

For details on how to build and run the example applications with supported toolchains, see the *Getting Started with MCUXpresso SDK* document. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in Figure 5.

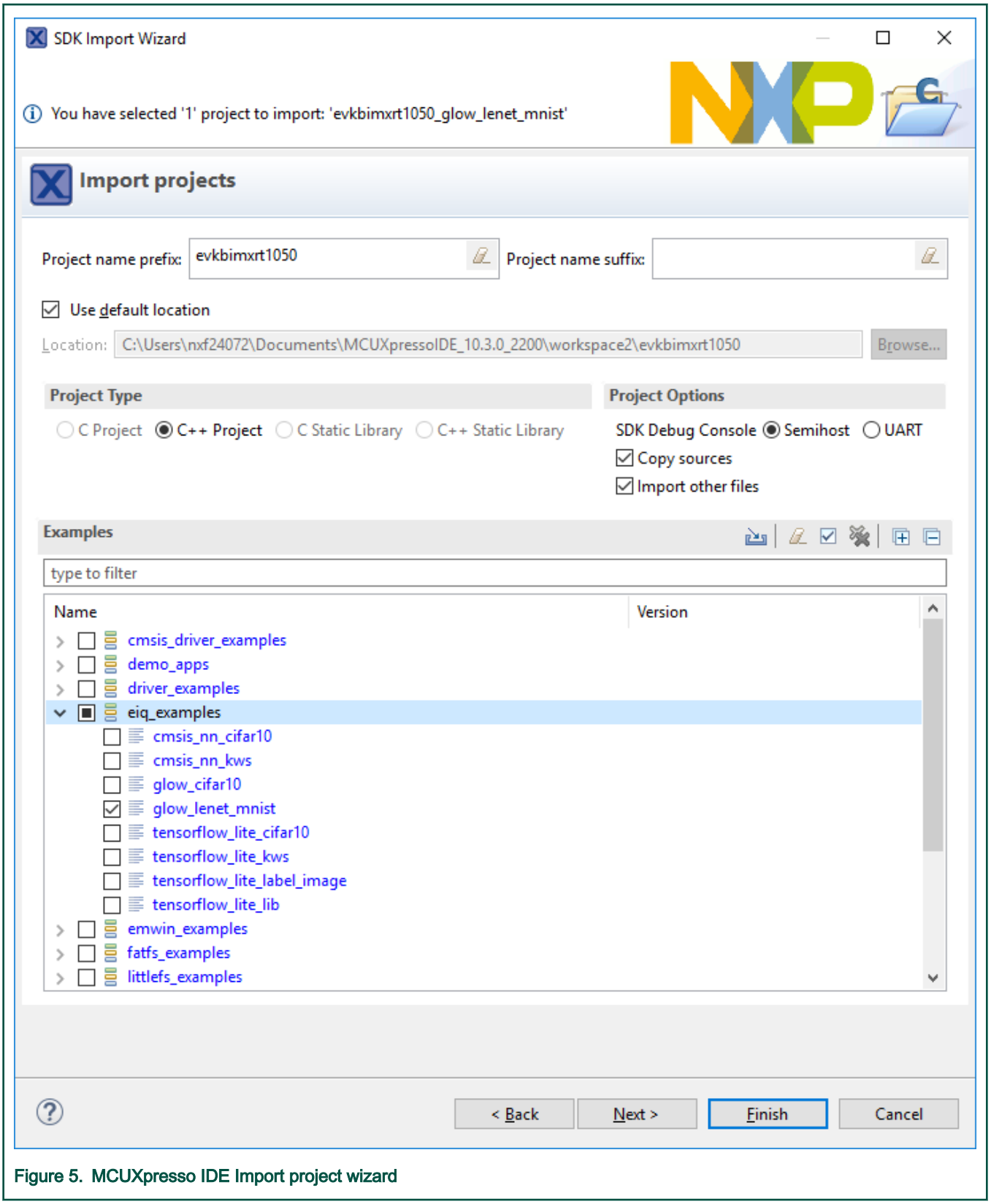

Figure 5. MCUXpresso IDE Import project wizard

Each example application contains a *readme.txt* file in the path *<project_folder>\doc* which describes the required hardware and the steps required to build and run the application. After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution is resumed, an output message should be displayed on the connected

terminal. For example, Figure 6 shows the output of the *glow_lenet_mnist* example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.

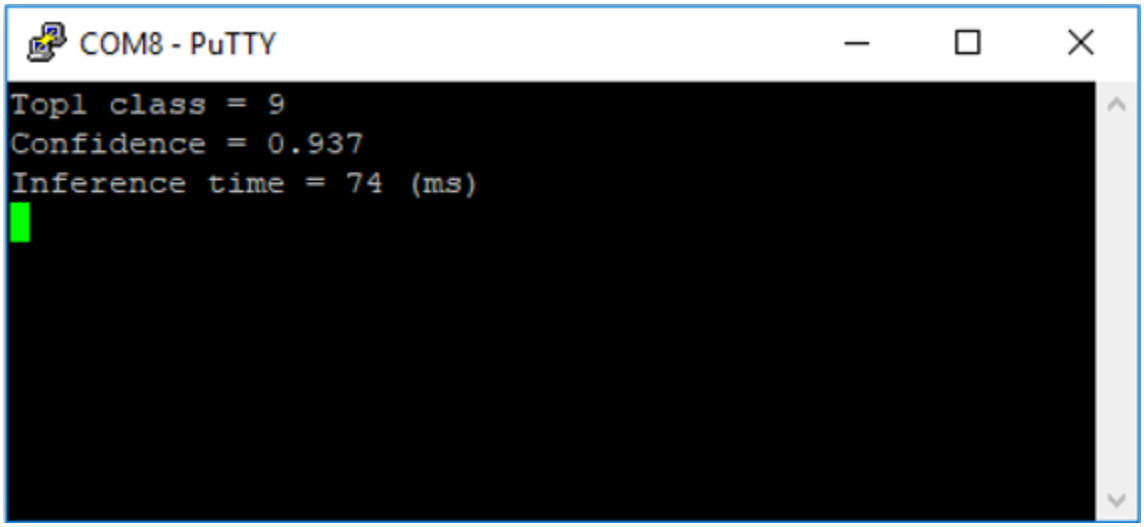

Figure 6. Serial connection window

4 Model compilation

- Bundle generation
- Model profiling
- Model tuning

4.1 Bundle generation

Bundle generation represents the model compilation to a binary object file (bundle). Bundle generation is performed using the *model-compiler* tool as shown in Figure 7.

Figure 7. Generating a bundle with the model-compiler tool

It is possible to generate both floating point and quantized bundles using the model-compiler tool. Generating quantized bundles is recommended as it can significantly reduce both the inference time and the memory footprint of the application.

The Glow compiler has an LLVM backend and is capable to cross-compile bundles for different target architectures. The following compile options apply when targeting some of the basic architectures of interest. Later this document explains how to significantly increase the performance by utilizing Arm CMSIS-NN and Cadence NN Library.

- For cross-compiling a bundle for the Arm Cortex M7 core from the i.MX RT 1050/1060 board: `-target=arm -mcpu=cortex-m7 -float-abi=hard`

- For cross-compiling a bundle for the Arm Cortex M33 core from the i.MX RT 685 board: `-target=arm -mcpu=cortex-m33 -float-abi=hard`

In Example: Generating floating point bundle for a network there is a sample command line that generates a **floating-point** bundle for the LeNet Caffe2 model. The model (files *init_net.pb* and *predict_net.pb*) is stored in the folder *models\lenet_mnist* and the bundle is saved in the folder *bundle* specified with the *emit-bundle* option. You can download the model files for LeNet from the following links:

- http://fb-glow-assets.s3.amazonaws.com/models/lenet_mnist/predict_net.pb
- http://fb-glow-assets.s3.amazonaws.com/models/lenet_mnist/init_net.pb

Parameter *-model-input-name* is used to specify the name of the input tensor (*data*), the type (*float*) and the shape (*[1,1,28,28]*). Example: Generating floating point bundle for a network assumes that the target architecture is the Arm Cortex M7 core.

Example: Generating floating point bundle for a network

```
model-compiler.exe ^
        -model=models\lenet_mnist -model-input=data,float,[1,1,28,28] -emit-bundle=bundle ^
        -backend=CPU -target=arm -mcpu=cortex-m7 -float-abi=hard
```

To generate a **quantized** bundle, the profiling information is required for the model. To generate the model profile (*profile.yml*) see the instructions in Model Profiling.

Example: Generating quantized bundle for a network shows an example of generating a quantized bundle by loading the profile *profile.yml* assumed available in the current directory (note the use of *–load-profile=profile.yml*). By default, quantization is performed according to an asymmetric 8-bit schema (QASYMM8).

Example: Generating quantized bundle for a network

```
model-compiler.exe ^
         -model=models\lenet_mnist -model-input=data,float,[1,1,28,28] -emit-bundle=bundle ^
         -backend=CPU -target=arm -mcpu=cortex-m7 -float-abi=hard ^
         -load-profile=profile.yml
```

When choosing the quantization schema, there is always a tradeoff between preserving the network accuracy and reducing the amount of computation. QASYMM8 schema is designed to preserve the accuracy at the cost of introducing extra computation steps.

If targeting best performance at the cost of losing some accuracy, you should use the quantization schema *symmetric_with_power2_scale* as shown in Example: Generating quantized bundle with CMSIS-NN. Parameter *-quantization-schema* lets you choose the quantization schema, while parameter *-use-cmsis* instructs the compiler to generate a bundle that uses the kernel implementations from the CMSIS-NN library. Additionally, the quantization procedure must quantize the bias of the Convolution and Fully Connected operators using int8 precision in order for the generic graph implementations to be replaced with the optimized implementations of the CMSIS-NN. This option is specified using the *quantization-precision-bias* option.

Example: Generating quantized bundle with CMSIS-NN

```
model-compiler.exe ^
     -model=models\lenet_mnist -model-input=data,float,[1,1,28,28] -emit-bundle=bundle ^
     -backend=CPU -target=arm -mcpu=cortex-m7 -float-abi=hard ^
     -load-profile=profile.yml ^
     -quantization-schema=symmetric_with_power2_scale ^
     -quantization-precision-bias=Int8 -use-cmsis
```

NOTE

CMSIS-NN is a library developed for Arm Cortex M0, M3, M4, M7 and M33 series implementing the following NN operations: Convolution, Fully Connected, Pooling and Activation layers. All the library implementations are for quantized models (no floating-point implementations are available) and require the quantization schema to be *symmetric_with_power2_scale* and int8 precision for the bias of Convolution and Fully Connected operators. Therefore, use the compilation flag *-use-cmsis* when building quantized bundles with this quantization schema for the i.MX RT devices (Arm Cortex M7 or M33 cores).symmetric_with_power2_scale symmetric_with_power2_scale symmetric_with_power2_scale.

For the i.MX RT 685 board which has an additional HiFi DSP core, we can instruct the compiler to use the HiFi-NN firmware for DSP acceleration with the flag *-use-hifi.* The generated bundle will run natively on the Arm Cortex M33 core but will dispatch NN operations to the HiFi DSP. *In* Example: Generating floating-point bundle with HiFi-NN *we have the command for building the floating-point bundle from* Example 1 but using the HiFi-NN firmware support for acceleration.

Example: Generating floating-point bundle with HiFi-NN

```
model-compiler.exe ^
    -model=models\lenet_mnist -model-input=data,float,[1,1,28,28] -emit-bundle=bundle ^
    -backend=CPU -target=arm -mcpu=cortex-m33 -float-abi=hard ^
    -use-hifi
```

In Example: Generating quantized bundle with HiFi-NN we have the command for building the quantized bundle from Example: Generating quantized bundle with CMSIS-NN but using the HiFi-NN firmware support for acceleration.

Example: Generating quantized bundle with HiFi-NN

```
 model-compiler.exe ^
        -model=models\lenet_mnist -model-input=data,float,[1,1,28,28] -emit-bundle=bundle ^
        -backend=CPU -target=arm -mcpu=cortex-m33 -float-abi=hard ^
        -load-profile=profile.yml ^
       -quantization-schema=symmetric_with_power2_scale ^
        -quantization-precision-bias=Int8 -use-hifi
```

NOTE

The HiFi-NN firmware is a software interface used by the Arm core to dispatch NN operations to the HiFi4 DSP. The DSP has a library with kernel implementations for NN operations. The library features both floating-point and quantized implementations using the symmetric_with_power2_scale quantization schema and int8 precision for the bias of Convolution and Fully Connected operators. Therefore, use the compilation flag -use-hifi only when building floating-point bundles or quantized bundles with the mentioned schema for the i.MX RT 685.

4.2 Model profiling

Glow uses profile guided quantization, running inference to extract statistics regarding possible numeric values of each tensor within the neural network. Later, during model compilation, Glow uses these statistics to quantize the model. The profiling procedure is independent of the target architecture and therefore no architecture specifications are needed. The profiling information required to quantize the model can be obtained using the *image-classifier* tool as shown in Figure 8.

Figure 8. Generating the profile with the *image-classifier* tool

To obtain the model profile for the default QASSYM8 quantization schema, you can use the *image-classifier* tool as shown in Example: Profiling a model using image-classifier tool. The profiling procedure requires a set of image examples to run the inference and derive the dynamic ranges of all the values involved in inference computations in order to choose the optimal quantization parameters. Download the images provided in the following link (MNIST dataset examples from the official Glow repository) into a new folder named *images*:

- https://github.com/pytorch/glow/tree/master/tests/images/mnist

The following command performs profiling for the LeNet model stored in folder *models\lenet_mnist* (files init_net.pb and predict_net.pb) and stores the profile in file *profile.yml*. Parameter *-model-input-name* is used to specify the name of the input layer name (*data*).

Example: Profiling a model using image-classifier tool

```
image-classifier.exe ^
        images\*.png ^
        -image-mode=0to1 ^
        -image-layout=NCHW ^
        -image-channel-order=BGR ^
        -model=models\lenet_mnist ^
        -model-input-name=data ^
        -dump-profile=profile.yml
```

If you need to generate a profile to use with CMSIS-NN, you should also pass the *quantization-schema* and the *-quantization-precision-bias* command line arguments to the *image-classifier* tool as shown in Example: Profiling a model for CMSIS-NN or HIFI-NN.

Example: Profiling a model for CMSIS-NN or HIFI-NN

```
image-classifier.exe ^
        images\*.png ^
        -image-mode=0to1 ^
        -image-layout=NCHW ^
        -image-channel-order=BGR ^
        -model=models\lenet_mnist ^
        -model-input-name=data ^
        -dump-profile=profile.yml ^
       -quantization-schema=symmetric_with_power2_scale ^
        -quantization-precision-bias=Int8
```

When generating the model profile, it is important to preprocess the input images the same way they were processed when training the model. See below some parameters of the image-classifier tool that control the image preprocessing:

- *-image-mode*: specifies the values range for the input tensor:
  - *neg1to1* for values in [-1, 1]

  - *0to1* for values in [0, 1]
  - *neg128to127* for values in [-128, 127]
  - *0to255* for values in [0, 255]
- *-image-layout*: specifies the layout to use:
  - NHWC (channel is inner-most dimension, channels are stored in memory with stride 1)
  - NCHW (width is inner-most dimension, widths are stored in memory with stride 1)
- *-image-channel-order*: specifies the order of the channels:
  - RGB
  - BGR

4.3 Model tuning

When generating the quantization profile for the first time using the *image-classifier* tool, the quantization parameters might not be the optimal ones in terms of model accuracy. When computing the quantization parameters, the *image-classifier* tool chooses the maximum dynamic ranges for the quantized tensors such that no saturation occurs. This means that the quantization step is the largest possible. The reality is that there are tensors for which most of the values are concentrated within a narrow range while also having a couple of outlier values. For these kinds of tensors, it would be better to narrow down the dynamic range to have a finer representation (with smaller quantization step) for most values while saturating the outlier values.

For this purpose, we have the *model-tuner* tool which takes a model, an input quantization profile (obtained initially with the *image-classifier* tool), and a labeled data set (a set of pairs of images and classification labels) and optimizes (tunes) the quantization profile for maximum accuracy. The optimized quantization profile (named *profile_tuned.yml*) can be further used by the *model-compiler* tool to compile the model according to the best quantization strategy. The *model-tuner* flow is shown in Figure 9.

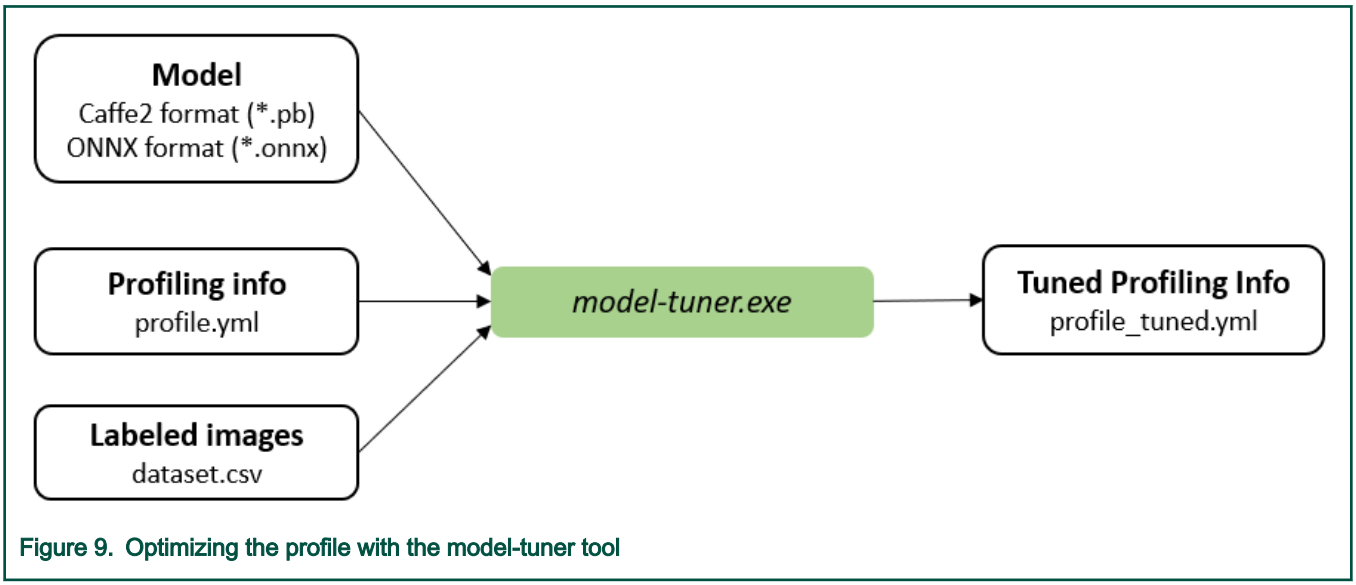

Figure 9. Optimizing the profile with the model-tuner tool

For example, in order to optimize the quantization for the model used in previous sections, we can use the following command:

Example: Tuning the quantization for a model

```
model-tuner.exe ^
   -dataset-file=dataset.csv ^
    -dataset-path=<path> ^
    -image-mode=0to1 ^
    -image-layout=NCHW ^
    -image-channel-order=BGR ^
```

```
    -model=models\lenet_mnist ^
    -model-input=data,float,[1,1,28,28] ^
    -load-profile=profile.yml ^
    -dump-tuned-profile=profile_tuned.yml ^
    -backend=CPU ^
    -quantization-precision-bias=Int8 ^
    -quantization-schema=symmetric_with_power2_scale
```

The labeled dataset description file is a text file which has on each line an entry with an image name and an image classification label separated by space (“ “) or comma (“,”). The classification labels are assumed to start with 0 (0, 1, …) just as the C index variables. Below we have an example of such a dataset file (the *dataset.csv* file in the example above) where the fields are comma separated (CSV file).

Example: Format of the labeled dataset file

```
image0.png,0,
image1.png,5,
image2.png,9,
……
```

As seen, the dataset file only contains the image names. In order for the tool to localize the images, the second parameter *dataset-path* is given which is the base directory where all the images can be found.

NOTE

The tuning procedure takes a long time to complete. Depending on the model size (number of nodes), the procedure might complete in hours or even days. In order to reduce the procedure time, one might choose to stop the tuning when a given accuracy has been reached, otherwise the tuning runs until completion. For example, you can choose to stop the tuning when the accuracy has reached the value 0.95 (95 %) by setting the *target-accuracy* flag:

Example: Tuning the quantization for a model

```
model-tuner.exe ^
    -target-accuracy=0.95 ^
     -dataset-file=dataset.csv ^
     -dataset-path=<path> ^
     -image-mode=0to1 ^
     -image-layout=NCHW ^
     -image-channel-order=BGR ^
     -model=models\lenet_mnist ^
     -model-input=data,float,[1,1,28,28] ^
     -load-profile=profile.yml ^
     -dump-tuned-profile=profile_tuned.yml ^
     -backend=CPU ^
     -quantization-precision-bias=Int8 ^
     -quantization-schema=symmetric_with_power2_scale
```

5 Creating the application project

The easiest way to create an application with the proper configuration is to copy and modify an existing example application. For details on where to find the example applications and how to build them, see Example applications.

5.1 Bundle API

The Bundle API defines the interface between the user application and the model compiled with Glow AOT. Each bundle has its own API which is generated when the model is compiled.

The most important component of the API is the entry point, the function that performs the model inference. The entry point has the same name as the model. The user may override the model name using the *-network-name* parameter of the *model-compiler* tool.

Besides the entry-point, the bundle API contains a series of macros that help in allocating and preparing the memory buffers for inference. In Example: Bundle API example you can see the actual bundle API header generated for the LeNet model compiled in the previous steps.

- *<MODEL_NAME>_CONSTANT_MEM_SIZE* – defines the size of the constant memory (contains the weights of the model)
- *<MODEL_NAME>_MUTABLE_MEM_SIZE* – defines the size of the mutable memory (contains the inputs and outputs of the model)
- *<MODEL_NAME>_ACTIVATIONS_MEM_SIZE* – defines the size of the memory buffers that is internally used for storing intermediate values (activations) during inference; buffer is used as a scratch buffer and hence can be reutilized by the user code
- *<MODEL_NAME>_MEM_ALIGN* – defines the memory alignment requirement for all the allocated buffers
- *<MODEL_NAME>_<placeholder_name>* – defines the offset of the model placeholder *<placeholder_name>* within the mutable memory area. In simple terms the placeholders of a graph/model are the overall inputs and outputs.

Example: Bundle API example

```
#ifndef _GLOW_BUNDLE_LENET_MNIST_H
#define _GLOW_BUNDLE_LENET_MNIST_H

#include <stdint.h>

// ----------------------------------------------------------------
//                       Common definitions
// ----------------------------------------------------------------
#ifndef _GLOW_BUNDLE_COMMON_DEFS
#define _GLOW_BUNDLE_COMMON_DEFS

// Memory alignment definition with given alignment size
// for static allocation of memory.
#define GLOW_MEM_ALIGN(size)  __attribute__((aligned(size)))

// Macro function to get the absolute address of a
// placeholder using the base address of the mutable
// weight buffer and placeholder offset definition.
#define GLOW_GET_ADDR(mutableBaseAddr, placeholderOff)     \
   (((uint8_t*)(mutableBaseAddr)) + placeholderOff)

#endif

// ----------------------------------------------------------------
//                         Bundle API
// ----------------------------------------------------------------
// Model name: "lenet_mnist"
// Total data size: 1785472 (bytes)
// Placeholders:
//
//   Name: "data"
//   Type: float
//   Shape: [1, 1, 28, 28]
//   Size: 784 (elements)
//   Size: 3136 (bytes)
//   Offset: 0 (bytes)
//
```

```
//   Name: "softmax"
//   Type: float
//   Shape: [1, 10]
//   Size: 10 (elements)
//   Size: 40 (bytes)
//   Offset: 3136 (bytes)
//
// NOTE: Placeholders are allocated within the "mutableWeight"
// buffer and are identified using an offset relative to base.
// ----------------------------------------------------------------
#ifdef __cplusplus
extern "C" {
#endif

// Placeholder address offsets within mutable buffer (bytes)
#define LENET_MNIST_data     0
#define LENET_MNIST_softmax  3136

// Memory sizes (bytes)
#define LENET_MNIST_CONSTANT_MEM_SIZE     1724672
#define LENET_MNIST_MUTABLE_MEM_SIZE      3200
#define LENET_MNIST_ACTIVATIONS_MEM_SIZE  57600

// Memory alignment (bytes)
#define LENET_MNIST_MEM_ALIGN  64

// Bundle entry point (inference function)
void lenet_mnist(uint8_t *constantWeight, uint8_t *mutableWeight,
                 uint8_t *activations);

#ifdef __cplusplus
}
#endif
#endif

#endif // _GLOW_BUNDLE_LENET_MNIST_H
```

5.2 Integrating the bundle

The model compilation produces the following set of files.

- *<model_name>.o* – the bundle object file
- *<model_name>.h* – the bundle header file (API)
- *<model_name>.weights.txt* – the weights of the model as C array data
- *<model_name>.weights.bin* – the weights of the model stored in binary format

The model parameters (weights) are provided in two formats:

- ***.txt*** (text format) suitable to initialize the weights statically (at compile time) in the application using the *"#include"* pre-processor directive
- ***.bin*** (binary format) suitable to initialize the weights dynamically (at runtime) by the application using standard library functions like *"fread"*

To integrate the bundle, include the first three generated files in the project. After copying these files in the *source* folder of the project,perform the following steps.

1. Change the project properties and modify the linker options such that it includes the bundle object file when linking the application. To modify the linker options:

- **For MCUXpresso IDE**
  a. Right-click the project and select "Properties".
  b. Select *"C/C++ Build"* > "Settings".
  c. In the "Tool Setting" tab, select *"MCU C++ Linker"* > *"Miscellaneous"*.
  d. Add the bundle to *"Other objects"*.
  e. Click *"Add..."* and specify the relative path to the bundle in the project. That is, "*../source/<model_name>.o"*.
- **For IAR Embedded Workbench**
  a. Right click on the project and select "Add" > "Add Files …".
  b. Choose the file category "Library/Object Files (*.r;*.a;*.lib;*.o)".
  c. Select the bundle *<model_name>.o*.
  d. Click "Open".
- **For Keil uVision MDK**
  a. Right click on the target and select "Options for Target …".
  b. Select the "Linker" tab.
  c. In the "Misc controls" section add to the existing string with a separating space character the relative path to the bundle (i.e. "*../source/<model_name>.o")*.

2. Include the *glow_bundle_utils.h* header file and the bundle API header file in the application main source file. For example, *main.cpp*. Assuming that the model is compiled as in Example: Including bundle, the header file is named *lenet_mnist.h*.

   Example: Including bundle API

```
#include "lenet_mnist.h"
#include "glow_bundle_utils.h"
```

3. Declare the buffers for *constant weights*, *mutable weights,* and *activations* in the application main source file. For example, *main.cpp*. Initialize the constant weights with values included from the *.txt* file.

   Example: Declaring and initializing the memory buffers file

```
// Statically allocate memory for constant weights (model weights) and initialize.
GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
uint8_t constantWeight[LENET_MNIST_CONSTANT_MEM_SIZE] = {
#include "lenet_mnist.weights.txt"
};

// Statically allocate memory for mutable weights (model input/output data).
GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
uint8_t mutableWeight[LENET_MNIST_MUTABLE_MEM_SIZE];

// Statically allocate memory for activations (model intermediate results).
GLOW_MEM_ALIGN(LENET_MNIST_MEM_ALIGN)
uint8_t activations[LENET_MNIST_ACTIVATIONS_MEM_SIZE];
```

NOTE

In this sample, we have opted for static allocation of the buffers. If dynamic allocation is necessary, you must pay attention to memory alignment. For this purpose, we provide the *alignedAlloc* function declared in header *glow_bundle_utils.h*. The function takes as parameters the required *alignment* (in bytes) and the requested allocation *size*. The function returns a pointer to the allocated memory buffer on success and NULL on error. Use function *alignedFree* to deallocate memory previously allocated with *alignedAlloc*.

4. Use *GLOW_GET_ADDR* macro to obtain the absolute address for each of the mutable weights of the model (model inputs & outputs).

   Example: Obtaining the addresses for the model input and output

   ```
   // Bundle input data absolute address.
   uint8_t *inputAddr = GLOW_GET_ADDR(mutableWeight, LENET_MNIST_data);

   // Bundle output data absolute address.
   uint8_t *outputAddr = GLOW_GET_ADDR(mutableWeight, LENET_MNIST_save_softmax);
   ```

5. Initialize the model input. In a real scenario, the input data for the inference is obtained by the user code through an acquisition procedure from sensors. For example, cameras or transmitted from other devices through wired or wireless communication. For simplicity, in this example we provide a buffer *imageData* which is statically initialized with a sample image. The buffer is initialized using the text file *input_image.inc* produced by a Python script by reading, preprocessing, and serializing a sample *png* image. In this example, copy the data from the initialized buffer *imageData* to the *inputAddr* buffer where the inference function expects to find the input. In a real application the user can avoid this copy by filling directly the buffer pointed to by *inputAddr*.

   Example: Initializing the model input

   ```
   memcpy(inputAddr, imageData, sizeof(imageData));
   ```

   NOTE

   When generating the input data, it is important to preprocess it before running inference. In the sample application projects delivered with the MCUXpresso SDK. There is a Python script (*glow_process_image.py*) in the project *scripts* subfolder that generates the C array representation of the preprocessed input image.

6. Now everything is prepared to run inference by passing the three memory buffers initialized above to the bundle entry-point function.

   Example: Calling the inference function

   ```
   lenet_mnist(constantWeights, mutableWeightVarsAddr, activationsAddr);
   ```

7. After running the inference, the results are available in the *outputAddr* buffer. Since LeNet is a hand-written digit classification model, the result is an array of 10 floating-point numbers. Each number representing the confidence score for each of the 10 digits (0 to 9). For convenience, provide the code which prints the class and the confidence score for the most certain class (with maximum confidence, also known as *top1*).

   Example: Processing the inference result

   ```
   // Get classification top1 result and confidence
   float *out_data = (float*)(outputAddr);
   float max_val = 0.0;
   uint32_t max_idx = 0;
   for(int i = 0; i < LENET_MNIST_OUTPUT_CLASS; i++) {
   if (out_data[i] > max_val) {
   max_val = out_data[i];
   max_idx = i;
   }
   }
   // Print classification results
   PRINTF("Top1 class = %lu\r\n", max_idx);
   PRINTF("Confidence = %f\r\n", max_val);
   ```

NOTE

When importing a bundle generated with *-use-hifi* parameter, you have to configure the project to use the CMSIS-NN middleware SDK component.

6 Utilities

This section describes utilities which can be used to convert, visualize and debug models.

6.1 Model conversion

The Glow compiler currently has support only for Caffe2 and ONNX model formats. Since a lot of well-known models are available in other formats, for example TensorFlow, it might be of interest to have some tools to convert models between different formats. The most used tools for format conversion are MMDNN and tf2onnx:

- MMDNN: https://github.com/Microsoft/MMdnn
- tf2onnx: https://github.com/onnx/tensorflow-onnx

We will exemplify how to convert a TensorFlow model to ONNX using the MMDNN tool. We will convert a MobileNet V1 image classification model which operates on 128 x 128 RGB images and 1001 classes. Download the MobileNet V1 model archive from here:

- http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_0.25_128.tgz

After you install MMDNN run the following command to convert the TensorFlow frozen file mobilenet_v1_0.25_128_frozen.pb to the ONNX model file mobilenet_v1_0.25_128_frozen_2018.onnx. mobilenet_v1_0.25_128_frozen_2018.onnx mobilenet_v1_0.25_128_frozen.pb mobilenet_v1_0.25_128_frozen.pb

Example: Convert model from TensorFlow format to ONNX using MMDNN

```
mmconvert ^
    -sf tensorflow ^
    -iw mobilenet_v1_0.25_128_frozen.pb ^
    --inNodeName input ^
    --inputShape 128,128,3 ^
    --dstNodeName MobilenetV1/Predictions/Softmax ^
    -df onnx ^
    -om mobilenet_v1_0.25_128_frozen_2018.onnx
```

You can find additional models in the links below, either directly in ONNX format or other formats which can be converted to ONNX using the conversion tools previously mentioned.

ONNX Model Zoo: https://github.com/onnx/models

MobileNetV1: https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

MobileNetV2: https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet

6.2 Model visualizer

A very popular tool for visualizing the **original model** before compiling with Glow is **Netron** which has an online browser version here: https://lutzroeder.github.io/netron/. In order to use Netron drag and drop the model file into the browser window.

The Glow compiler integrates the *graphviz* utility for exporting the graph visual representation of the **compiled model** in *dot* format. The graph will depict all the optimizations and conversions performed on the original model by Glow including the node specializations when using CMSIS-NN or HIFI-NN. Note that the compile command from Example: Generating floating point bundle for a network but with the addition of the *-dump-graph-DAG=graph.dot* option which exports the graph visual representation in the file *graph.dot* as presented below.

Example: Dump model graph visual representation to DOT file

```
model-compiler.exe ^
    -model=models\lenet_mnist -model-input=data,float,[1,1,28,28] -emit-bundle=bundle ^
    -backend=CPU -target=arm -mcpu=cortex-m7 -float-abi=hard ^
    -dump-graph-DAG=graph.dot
```

The DOT format is a text description file which can be used to generate visual representations of the graph. We can use the “dot.exe” utility (which is installed together with the Glow tools for Windows) to convert the DOT file to PDF or PNG file formats as depicted below.

Example: Convert graph DOT format to PDF/PNG format

```
dot -Tpdf graph.dot -o graph.pdf -Nfontname="Times New Roman,"
dot -Tpng graph.dot -o graph.png -Nfontname="Times New Roman,"
```

The model graph representation for LeNet generated as PDF file might be as shown in Figure 10.

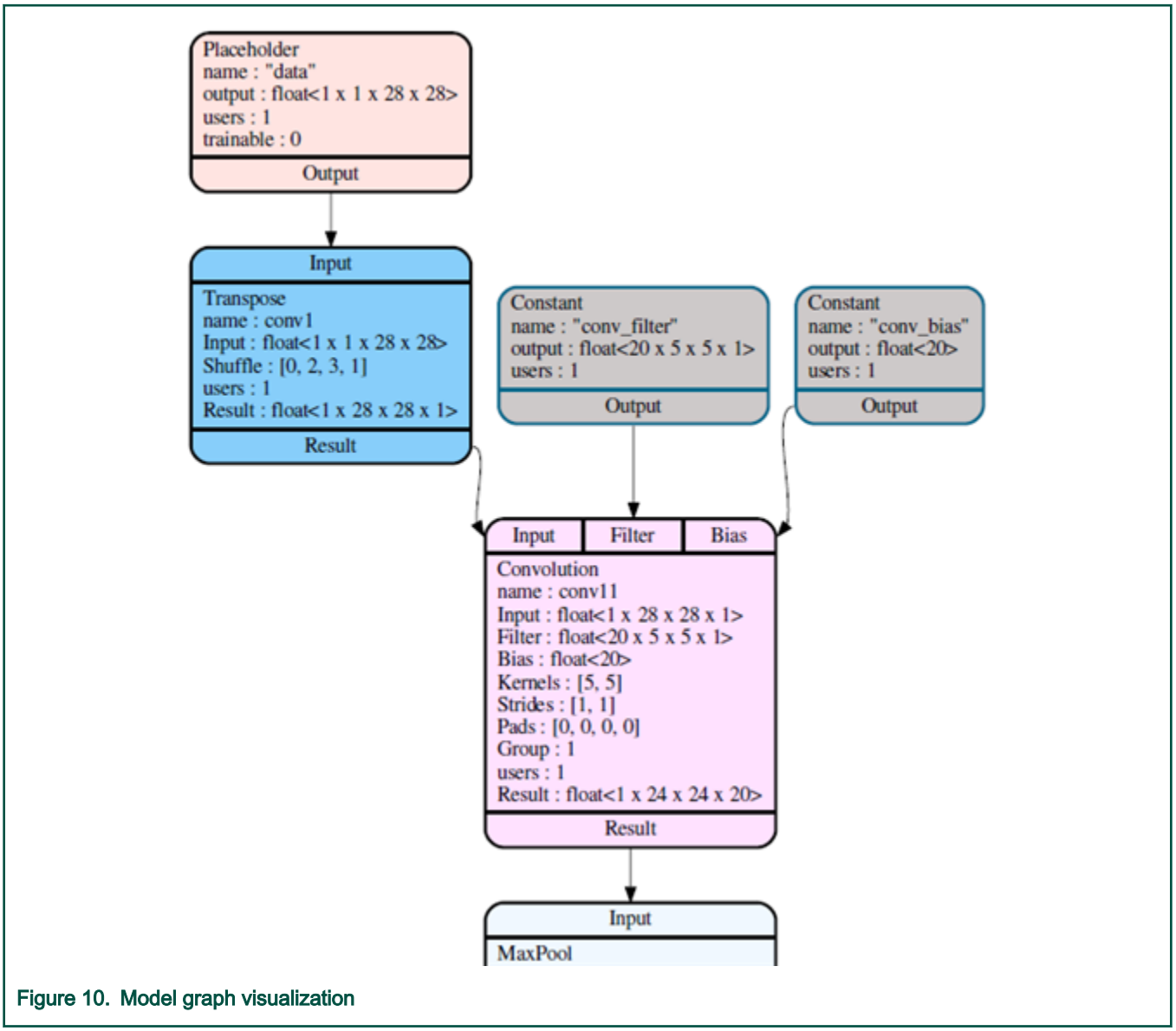

Figure 10. Model graph visualization

7 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.

8 Revision history

Table 2 summarizes the changes done to the document since the initial release.

Table 2. Revision history

Revision number	Date	Substantive changes
0	09/2019	Initial release with CMSIS-NN support
1	04/2020	Updated for HiFi-NN support

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 14 April 2020

Document identifier: EIQGLOWAOTUG

arm