eIQ™ Inference with Tensorflow Lite for Microcontrollers on i.MX RT1170 - With Camera

Revision 7
November 2024

Contents

1 Lab Overview

This lab will cover how to take an existing TensorFlow image classification model and run it on an i.MX RT embedded device. A Flowers model that can be created in eIQ Toolkit will be used an example. A camera attached to the board can then be used to look at photos of flowers and the model will determine what type of flower the camera is looking at.

This lab can also be used without a camera+LCD, but the flower image will need to be converted to a C array and loaded at compile time. Instructions for that version of the lab can be found in the "**eIQ TensorFlow Lite for Microcontrollers Lab for RT1170 – Without Camera.pdf**" document.

This lab is written for the i.MX RT1170 evaluation board. It can also be used with the following boards that support a camera interface by downloading their respective SDK packages:

- i.MX RT1050
- i.MX RT1060
- i.MX RT1064
- i.MX RT1160
- i.MX RT1170
- i.MX RT1180

Also note that the i.MX RT1170, i.MX RT1060, and i.MX RT1064 evaluation kits come with a camera sensor. The i.MX RT1050-EVKB does not come with a camera sensor but uses the same camera as the i.MX RT1060. If using the camera, it is highly recommended to also purchase the LCD screen as well. An LCD screen compatible with the i.MX RT1060, i.MX RT1050, and i.MX RT1064 boards can be found here. An LCD screen compatible with the i.MX RT1160 and i.MX RT1170 boards can be found here. Also be aware that there is an updated version for each of those LCD panels so if you have an older version of the LCD panel there may be a small software change required for any SDK examples that use the LCD.

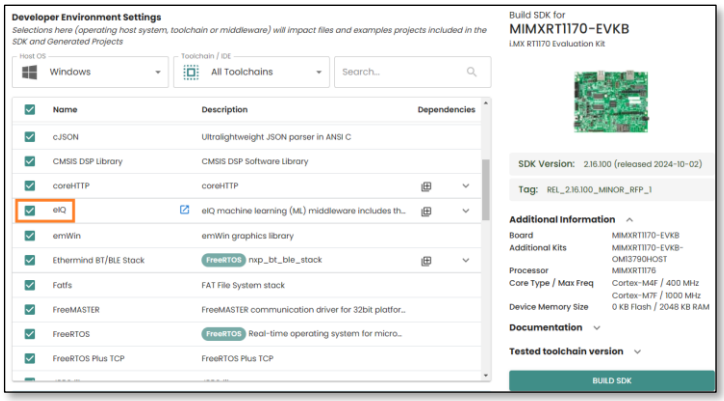
2 Software and Hardware Installation

This section will cover the steps needed to install the eIQ software and TensorFlow on your computer.

2.1 NXP MCUXpresso SDK Installation

1. Install the latest version of MCUXpresso IDE
2. Install the latest eIQ Toolkit
3. Install a terminal program like TeraTerm.
4. Download the latest MCUXpresso SDK for i.MXRT1170. It includes the eIQ software platform and demos.

a) On the SDK builder page, make sure to select the “eIQ” middleware.



b) Then click on the **Download SDK** button and accept the license agreement to download the zip file.

3 Create a TensorFlow Model

This lab will use the Flowers model generated by eIQ Toolkit as part of the eIQ Toolkit Data Import lab.

There are many other ways that a TensorFlow Lite model could be created including through cloud based ML tools, TensorFlow scripts and labs, and other methods. This lab will focus on how to get an already existing TFLite model to run on NXP silicon using eIQ enablement software.

4 Convert Model and Labels

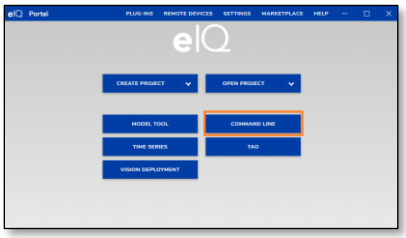
Once you have a Tensorflow Lite file, the next step is to convert the TFLite file into a C header file that can be imported into an MCUXpresso SDK example.

For this lab, we’ll use the flower identification model created using the eIQ Toolkit. It will be assumed that the name of that model is **flower_model.tflite**

4.1 Convert TensorFlow model

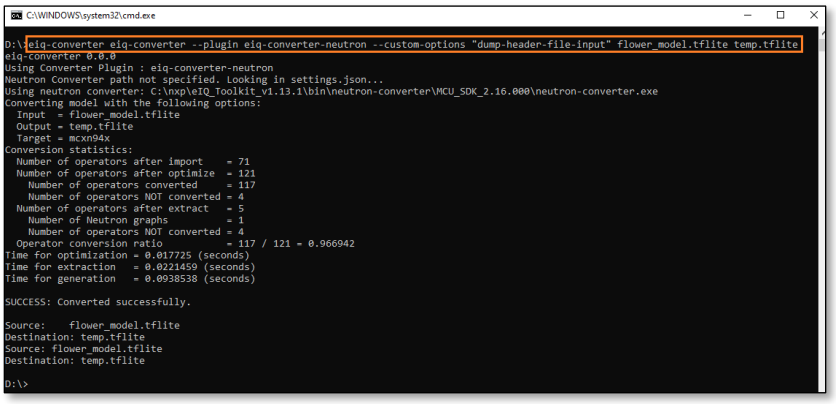
1. Open eIQ Toolkit

2. Open the Command Line utility



Note: *For this lab you must open the Command Line utility via eIQ Toolkit as it sets some global variables used by the converter. Do not use the generic Windows command line.*

3. Navigate to the directory location that the **flower_model.tflite** file is located. We'll use the Neutron Converter Tool to convert the .tflite binary file into a C array that can be imported into an embedded project. The Neutron converted temp.tflite file will not be used, we're just focused on the flower_model.h file that will be generated by this command (all on one line): **eiq-converter eiq-converter --plugin eiq-converter-neutron --custom-options "dump-header-file-input" flower_model.tflite temp.tflite**



4. The generated flower_model.h header file will need to be modified slightly to integrated it into the MCUXpresso SDK. Open up the **flower_model.h** file and make the following changes to the top of the file, but do not erase the commented out data at the top as that will be used later. Also make note of the array name as it will be used in the next section. Also the kTensorArenaSize may need to be larger depending on the model (or smaller to save RAM size). The value below works well for the Flower model being used for this lab.

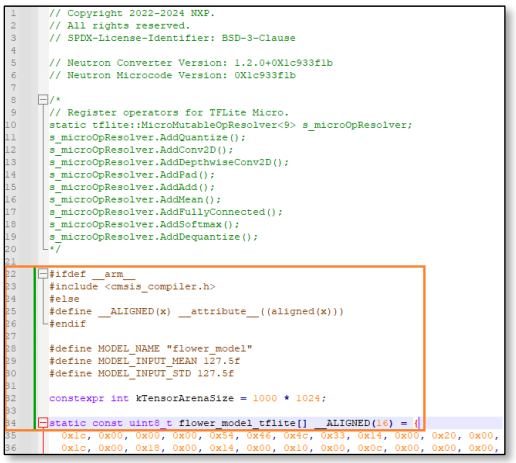
```
#ifdef __arm__
#include <cmsis_compiler.h>
#else
#define __ALIGNED(x) __attribute__((aligned(x)))
#endif

#define MODEL_NAME "flower_model"
#define MODEL_INPUT_MEAN 127.5f
#define MODEL_INPUT_STD 127.5f

constexpr int kTensorArenaSize = 1000 * 1024;

static const uint8_t flower_model_tflite[] __ALIGNED(16) = {
```

5. It should look like the following when changed:



6. Next, use a text editor to create a new file named **flower_labels.h** to create an array of the label names. Copy and paste the following code into that newly created **flower_labels.h** file:

```
const char* labels[] = {
                        "daisy",
                        "dandelion",
                        "roses",
                        "sunflowers",
                        "tulips"
};
```

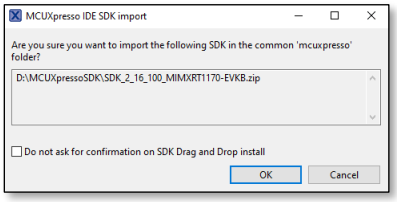
5 Run Demo with TensorFlow Lite for Microcontrollers

The final step is to take the TensorFlow Lite Micro Label Image example and modify it to use the newly retrained model.

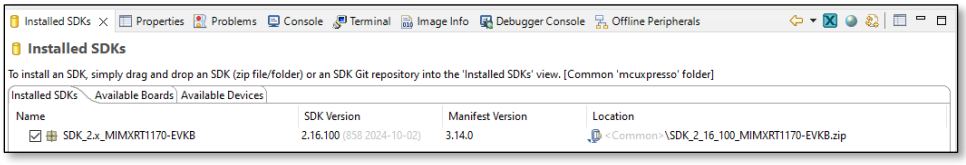
5.1 Copy and Create Files

1. Open MCUXpresso IDE and select a workspace.
2. Close the Welcome Screen tab by clicking on the X in that tab

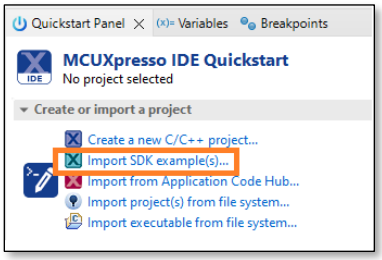
3. Drag-and-drop the i.MX RT1170 SDK zip file into the Installed SDKs window, located on a tab at the bottom of the screen named “Installed SDKs”. You will get the following pop-up, so hit OK.



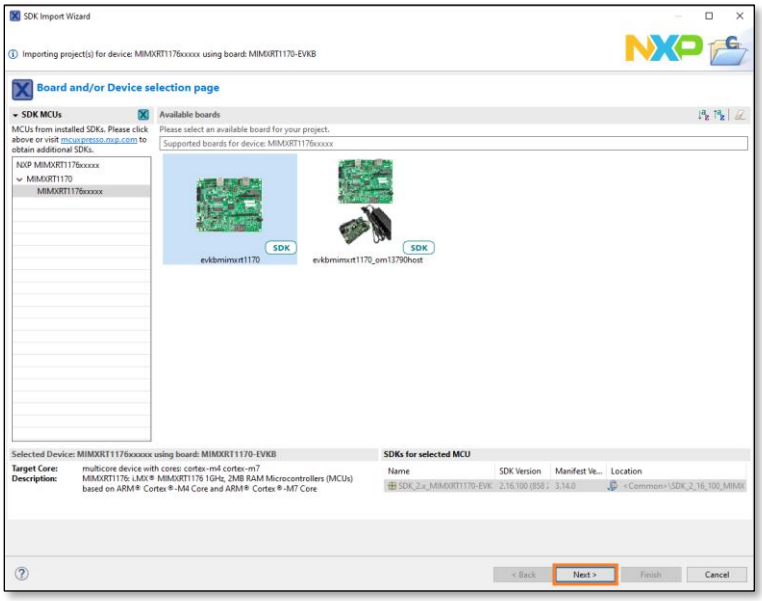
4. Once imported, the Installed SDK panel will look something like this:



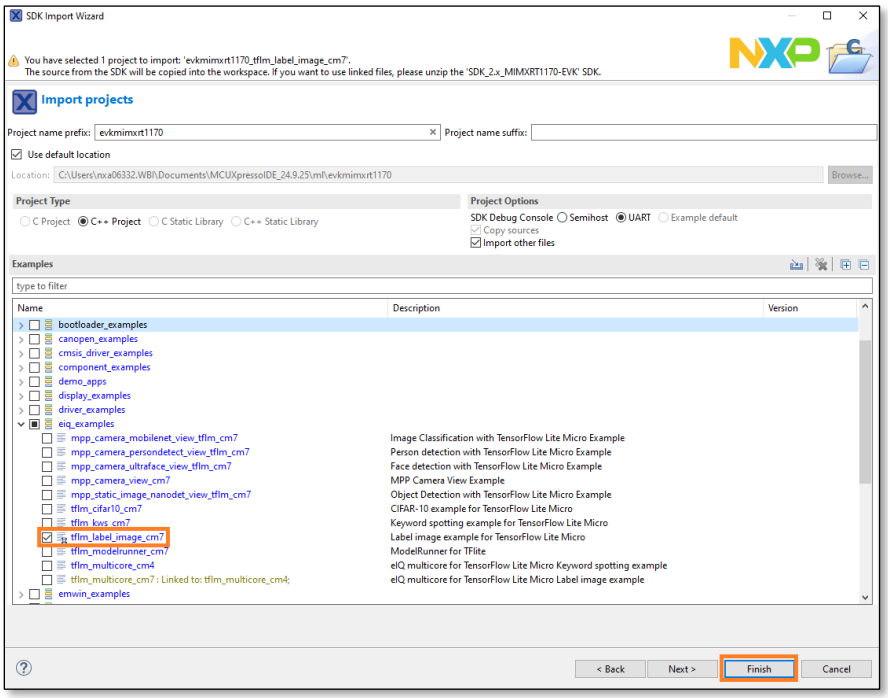
5. In the QuickStart menu, select **Import SDK example(s)...**



6. Select the RT1170-EVK (**evkmimxrt1170**) and click on **Next**

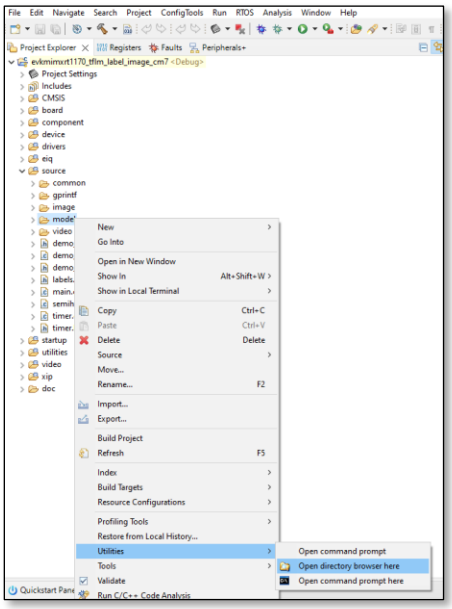


7. Import the **tflm_label_image_cm7** example. Then click on **Finish** to select that project.

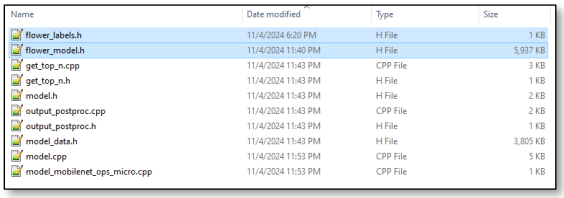


7. Now we need to import both the new model file and labels file that was generated in the last section into this project.

8. Open the directory location to place the model by right clicking on the model folder in the Project Explorer and selecting **Utilities->Open directory browser here**



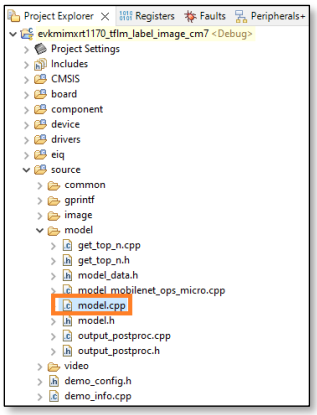
9. It should open a directory at something like something like:
C:\Users\<username>\Documents\MCUXpressoIDE_24.9.25\ml\evkmimxrt1170_tflm_label_image_cm7\source\model
10. Inside that **model** directory, copy the **flower_model.h** file and the **flower_labels.h** file generated in the previous section.
11. Directory should look like the following when finished:



5.2 Modify Source Code

Now edit the source files to include these new files

1. Double click on the **model.cpp** file under the "source\model" folder in the Project View to open it.



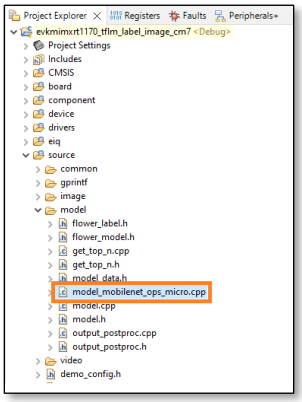
8. On line 27, comment out original #include for the original model defined in **model_data.h**. Then add a new #include to bring in the new model with **flower_model.h**. It should look like the following when finished:

```
24
25 #include "fsl_debug_console.h"
26 #include "model.h"
27 //#include "model_data.h"
28 #include "flower_model.h"
```

9. On line 43, change the model name to the array name in **flower_model.h**:

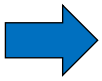
```
39 status_t MODEL_Init(void)
40 {
41     // Map the model into a usable data structure. This doesn't involve any
42     // copying or parsing, it's a very lightweight operation.
43     s_model = tflite::GetModel(flower_model_tflite);
44     if (s_model->version() != TFLITE_SCHEMA_VERSION)
45     {
```

10. Next open up **model_mobilenet_ops_micro.cpp**



11. To reduce the size of the project, the Label Image example only supports the specific operands required by the default Mobilenet model. Our retrained model uses a few new operands. These specific operands are conveniently listed as part of the conversion process in the comment block at the top of **flower_model.h**. To update the operand list, copy the list from that comment block in **flower_model.h** and replace the default operators in **MODEL_GetOpsResolver** in **model_mobilenet_ops_micro.cpp.** Note that the specific list of operators will vary depending on the particular model.

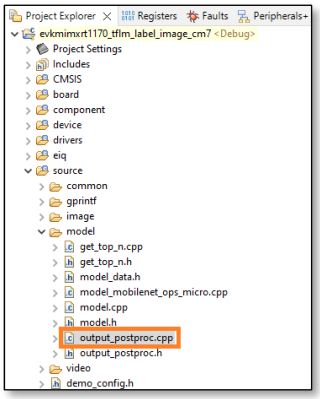
```
1  // Copyright 2022-2024 NXP.
2  // All rights reserved.
3  // SPDX-License-Identifier: BSD-3-Clause
4
5  // Neutron Converter Version: 1.2.0+0X1c933f1b
6  // Neutron Microcode Version: 0X1c933f1b
7
8  /*
9  // Register operators for TFLite Micro.
10 static tflite::MicroMutableOpResolver<9> s_microOpResolver;
11 s_microOpResolver.AddQuantize();
12 s_microOpResolver.AddConv2D();
13 s_microOpResolver.AddDepthwiseConv2D();
14 s_microOpResolver.AddPad();
15 s_microOpResolver.AddAdd();
16 s_microOpResolver.AddMean();
17 s_microOpResolver.AddFullyConnected();
18 s_microOpResolver.AddSoftmax();
19 s_microOpResolver.AddDequantize();
20 */
21
```



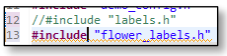
```
1  /*
2   * Copyright 2021-2023 NXP
3   * All rights reserved.
4   *
5   * SPDX-License-Identifier: BSD-3-Clause
6   */
7
8  #include "tensorflow/lite/micro/kernels/micro_ops.h"
9  #include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
10
11 tflite::MicroOpResolver &MODEL_GetOpsResolver()
12 {
13     static tflite::MicroMutableOpResolver<9> s_microOpResolver;
14     s_microOpResolver.AddQuantize();
15     s_microOpResolver.AddConv2D();
16     s_microOpResolver.AddDepthwiseConv2D();
17     s_microOpResolver.AddPad();
18     s_microOpResolver.AddAdd();
19     s_microOpResolver.AddMean();
20     s_microOpResolver.AddFullyConnected();
21     s_microOpResolver.AddSoftmax();
22     s_microOpResolver.AddDequantize();
23
24
25     return s_microOpResolver;
26 }
27
```

Make sure to not accidently remote the **return s_microOpResolver;** at the bottom of the function.

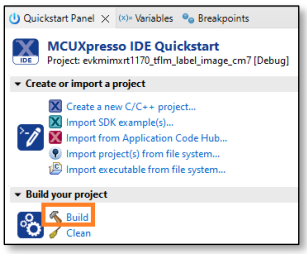
12. Next open the **output_postproc.cpp** file.



13. On line 12, comment out original #include for the original label file. Then add a new #include to bring in the new labels file. It should look like the following when finished:



14. Build the project by clicking on "Build" in the Quickstart Panel and make sure there are no errors.



5.3 Attach LCD and Camera

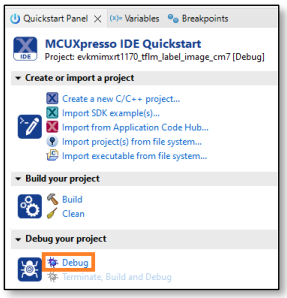
Now with all the software modifications completed, it's time to attach the camera and LCD. These steps can be found in this NXP Community post. The camera is only available as part of the i.MX RT1050, i.MX RT1060, i.MX RT1064, or i.MX RT1170 EVKs. An LCD screen compatible with the i.MX RT1060, i.MX RT1050, and i.MX RT1064 boards can be found here. An LCD screen compatible with the i.MX RT1170 board can be found here. Also be aware that there is an updated version for each of those LCD panels so if you have an older version of the LCD panel there may be a small software change required for any SDK examples that use the LCD.

This lab can be completed without using the camera+LCD by running the inferencing on a static image instead, but it is recommended to use the camera+LCD.

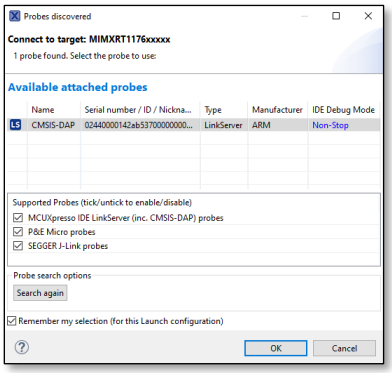
5.4 Run Example

15. Plug the micro-B USB cable into the board at J11 on the i.MXRT1170 board.
16. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.

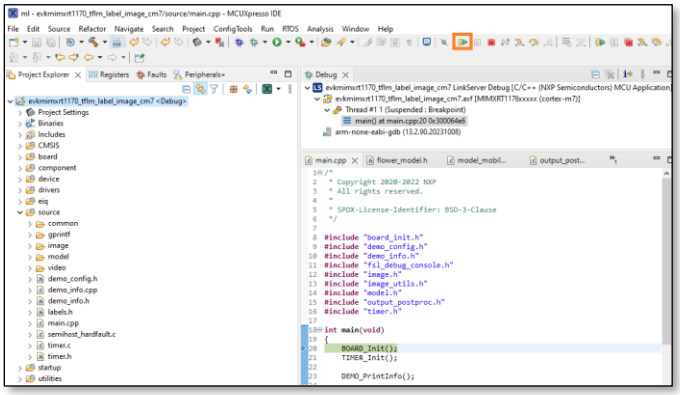
17. Debug the project by clicking on “Debug” in the Quickstart Panel.



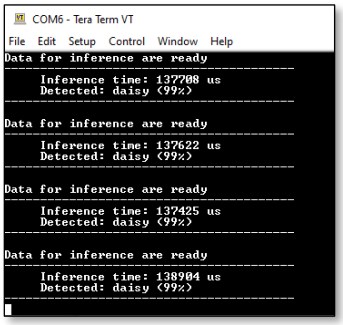
18. It will ask what interface to use. Select CMSIS-DAP.



19. The debugger will download the firmware and open up the debug view. It may take some time to download the firmware. Click on the **Resume** button to start running.



20. On the LCD screen, you should see what the camera is pointing at.
21. Open up a terminal window, and you the result of the inference from the camera input. Display the Flowers.pdf document on your computer and point the camera at your monitor to identify the different photos.



22. You may notice that even when the camera is pointed at random objects, it still attempt to categorize them as a flower type. This is because when the model was retrained, it was only retrained on flower images. The concept of any other type of object is unknown to the model, so it attempts to classify everything as one of the 5 types that it does know.

6 Conclusion

This lab demonstrated how to import a Tensorflow Lite model into the Tensorflow Lite for Microcontrollers inference engine provided as part of eIQ so that the model can be ran on embedded system. These same steps could also be used for other TFLite models to enable a wide world of opportunity for new smarter applications.