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

Lab Hand Out - Revision 5
August 2023

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

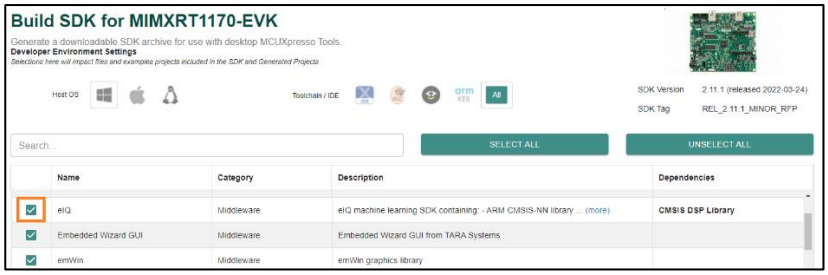
Also note that the i.MX RT1170, i.MX RT1060, and i.MX RT1064 evaluation kits come with a camera sensor. The i.MX RT1050-EVKB does not come with a camera sensor but uses the same camera as the i.MX RT1060. If using the camera, it is highly recommended to also purchase the LCD screen as well. An LCD screen 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 a terminal program like TeraTerm.
3. 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.

4. If on Windows, install Vim 9.0: https://www.vim.org/download.php#pc. There is a binary convertor program named xxd.exe located inside that package that will be needed.
5. Verify the PATH was set correctly by opening a Windows command prompt and typing "**xxd -v**" into the command prompt. You should not get any errors about an unrecognized command.

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. Use the **xxd** utility to convert the .tflite binary file into a C array that can be imported into an embedded project. Navigate to the directory that the .tflite file is located and then convert it using XXD:
   - If using Windows Command Prompt: **xxd -i flower_model.tflite > flower_model.h**
   - If using Windows Powershell: **xxd -i flower_model.tflite | out-file -encoding ASCII flower_model.h**
2. The generated 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. 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.

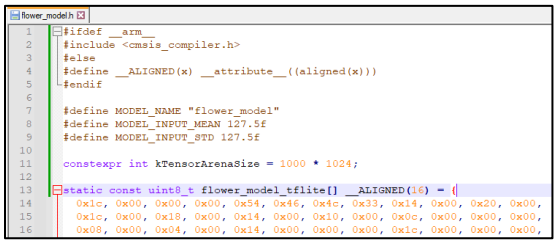
```
#ifdef __arm__
#include <cmsis_compiler.h>
#else
#define __ALIGNED(x) __attribute__((aligned(x)))
#endif

#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) = {
```

3. It should look like the following when changed:



4. 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:

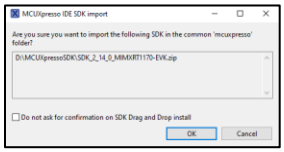
```
const char* labels[] = {
                        "daisy",
                        "dandelion",
                        "roses",
                        "sunflowers",
                        "tulips"
};
```

5 Run Demo with TensorFlow Lite for Microcontrollers

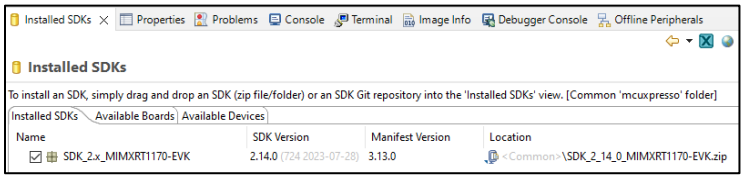
The final step is to take the TensorFlow Lite Micro Label Image example and modify it to use the newly retrained model.

5.1 Copy and Create Files

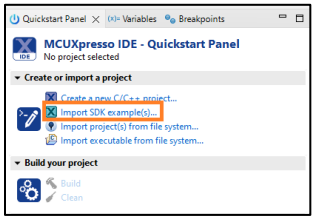
1. Open MCUXpresso IDE and select a workspace location in an empty directory.
2. Drag-and-drop the unzipped SDK folder into the Installed SDKs window, located on a tab at the bottom of the screen named “Installed SDKs”. You will get the following pop-up, so hit OK.



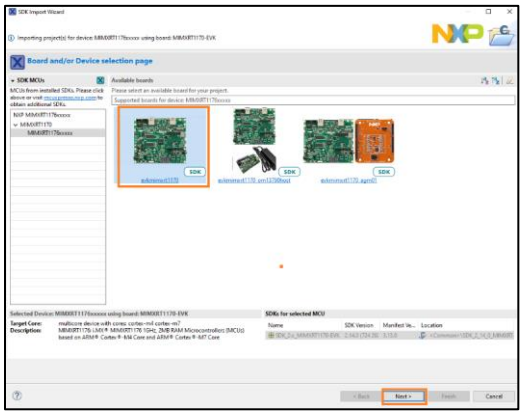
3. Once imported, the Installed SDK panel will look something like this



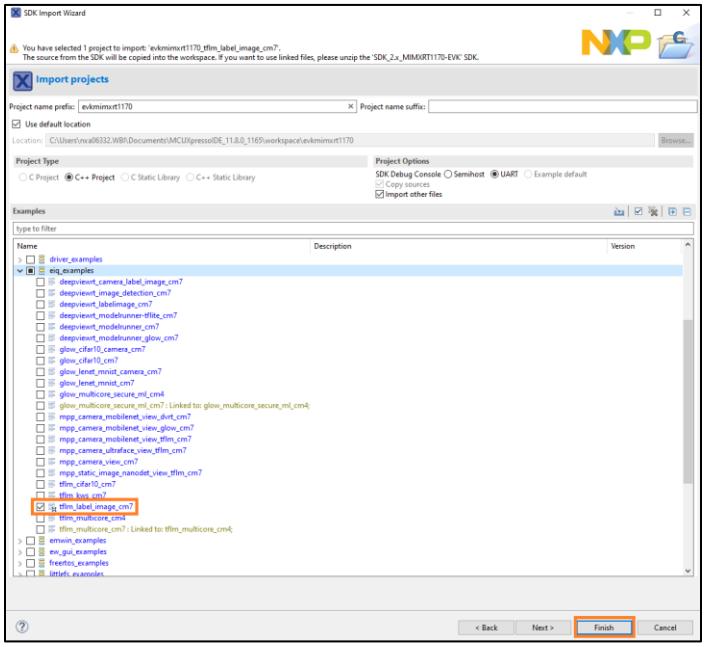
4. Next import the desired project. In the Quickstart Panel, select **Import SDK examples(s)…**



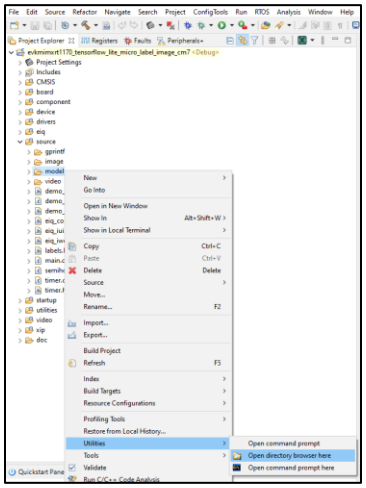
5. Select the evkmimxrt1170 board and click on Next



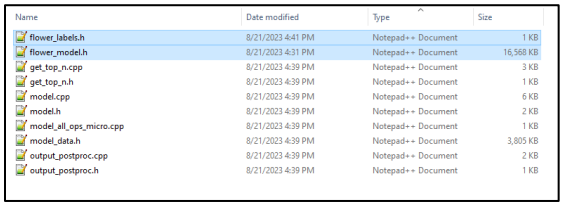
6. Import the **tflm_label_image_cm7** example. Then click on **Finish** to select that project.



5. Now we need to import both the new model file and labels file that was generated in the last section into this project.
6. 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**



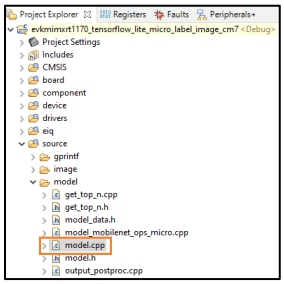
7. It should open a directory at something like something like: **C:\Users\<username>\Documents\MCUXpressoIDE_11.8.0_1165\workspace\evkmimxrt1170_tflm_label_image_cm7\source\model**
8. Inside that **model** directory, copy the **flower_model.h** file and the **flower_labels.h** file generated in the previous section.
9. Directory should look like the following when finished:



5.2 Modify Source Code

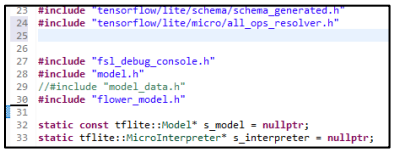
Now edit the source files to include these new files

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

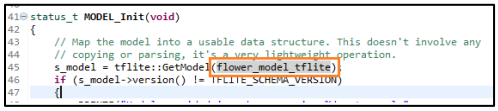


7. On line 24 add the following #include for the ops resolver that supports all the operands used by this retrained model:
   #include "tensorflow/lite/micro/all_ops_resolver.h"

8. On line 29, 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:



9. On line 45, change the model name to the array name in **flower_model.h**:



10. 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 can be determined by analyzing the model with an application called **netron** and then manually add the operands as described in Section 7.1 of the eIQ TensorFlow Lite Library User's Guide. Or alternatively all the TFLite operands can be supported in a project by

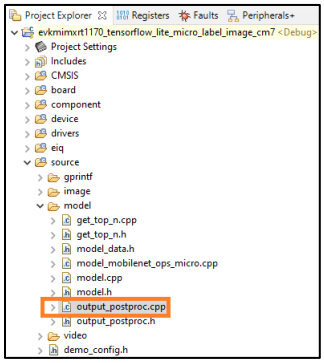
using the built in **tflite::AllOpsResolver** method. For this lab we'll use the latter method in order to provide the greatest compatibility with other models. On line 63 in model.cpp, comment out the original resolver line. Then add a new line

```
tflite::AllOpsResolver micro_op_resolver;
```

It will look like the following when done:

```
62   // tflite::MicroOpResolver &micro_op_resolver = MODEL_GetOpsResolver();
63   tflite::AllOpsResolver micro_op_resolver;
64
65   // Build an interpreter to run the model with.
```

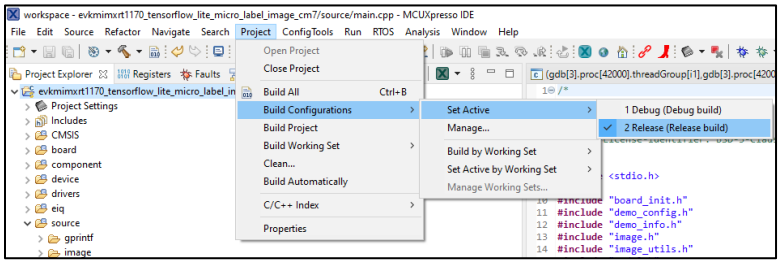
11. Next open the **output_postproc.cpp** file.



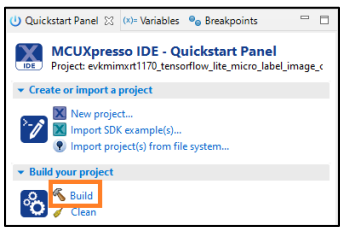
12. On line 12, comment out original #include for the original label file. Then add a new #include to bring in the new labels file. It should look like the following when finished:

```
12 //#include "labels.h"
13 #include "flower_labels.h"
```

13. Finally change the build configuration to "Release" by clicking on the project and going to the menu bar and going to **Project->Build Configurations->Set Active->Release**. This will enables high compile optimizations. This will significantly decrease the inference time of TFLM projects.



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



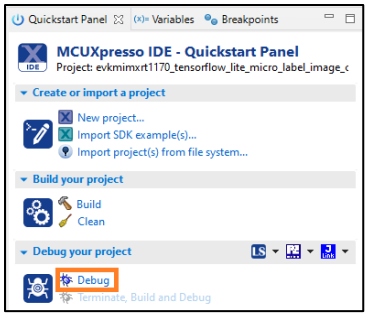
5.3 Attach LCD and Camera

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.

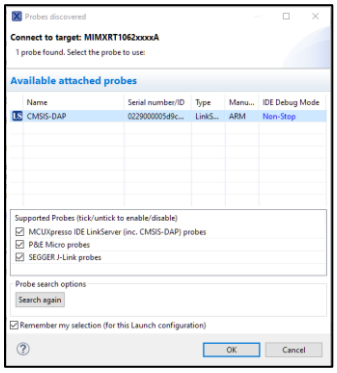
This lab can be completed without using the camera+LCD by running the inferencing on a static image instead, but it is recommended to use the camera+LCD.

5.4 Run Example

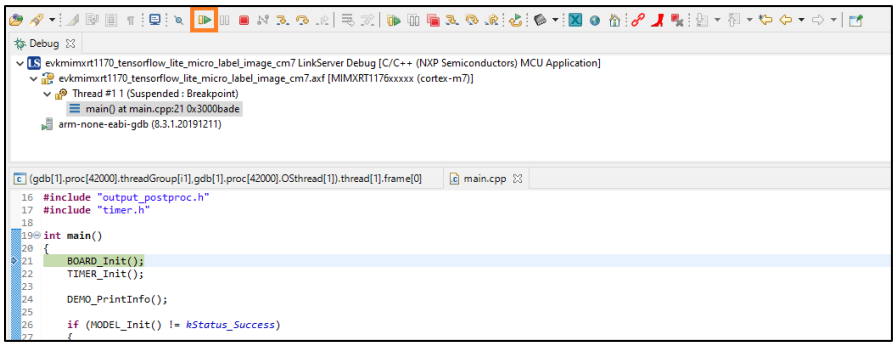
15. Plug the micro-B USB cable into the board at J11 on the i.MXRT1170 board.
16. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
17. Debug the project by clicking on “Debug” in the Quickstart Panel.



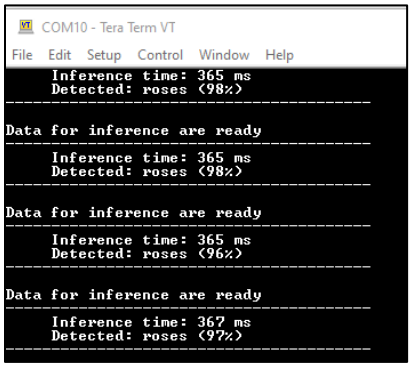
18. It will ask what interface to use. Select CMSIS-DAP.



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.