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

Lab Hand Out - Revision 1

Contents

1 Lab Overview

This lab will cover how to take an existing TensorFlow image classification model and re-train it to categorize images of flowers. This is known as transfer learning. This updated model will then be converted into a TensorFlow Lite file. By using that file with the TensorFlow Lite for Microcontrollers (TFLM) inference engine that is part of NXPs eIQ software package, the model can be run on an i.MX RT embedded device. 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 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 RT1170 board can be found here.

2 Software and Hardware Installation

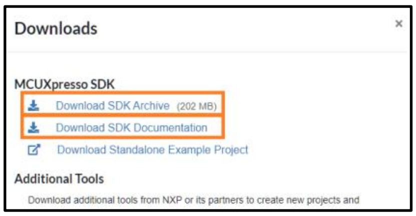
This section will cover the steps needed to install the eIQ software and TensorFlow on your computer.

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. Then click on the **Download SDK** button.

b) Accept the license agreement

c) In the pop-up, download both the SDK Archive (contains the source code and project files) package and the SDK Documentation package (contains the eIQ documentation)



2.2 TensorFlow Installation

4. Download and install Python 3.8. ****The 64-bit edition is required and TFLite 2.4 does not support Python 3.9, so for this lab it is highly recommended to use Python 64-bit 3.8.x****: https://www.python.org/downloads/

5. Open a Windows command prompt and verify that the python command corresponds to Python 3.8.x. You may need to use "python3" for all the commands instead:
   python -V
6. Update the python installer tools:
   python -m pip install -U pip
   python -m pip install -U setuptools
7. Install the Tensorflow libraries and support for python. To ensure software compatibility, it must be these version numbers:
   python -m pip install tensorflow==2.4
   python -m pip install tflite-model-maker==0.2.4

8. Install other useful python packages. Not all of these will be used for this lab but will be useful for other eIQ demos and scripts.
   python -m pip install onnxmltools mmdnn tensorflow-datasets opencv-python PILLOW
   python -m pip install numpy scipy matplotlib ipython jupyter pandas sympy nose imageio
   python -m pip install netron

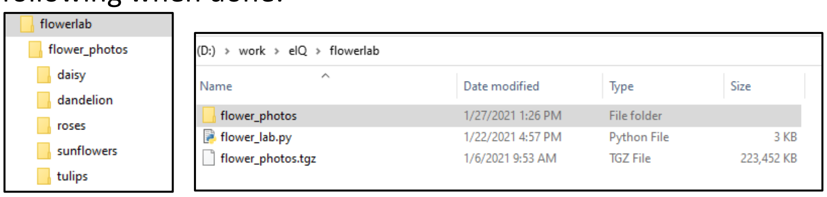
9. If on Windows, install Vim 8.1: https://www.vim.org/download.php#pc. There is a binary convertor program named xxd.exe located inside that package that will be needed.
10. If on Windows, add the following directories to your executable PATH if they are not already:
    1. <python_install_directory>/scripts
    2. <vim_install_directory>

11. 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.

2.3 Lab Setup

We'll be retraining the model to recognize photos of flowers and categorize them into different types. The new flower data that the model will be retrained on will also be download.

12. Create a new directory on your PC and download the **flower_lab.py** Python script attached to this Community post.
13. Download a set of Creative Commons licensed flowers images that have already been categorized into 5 different classes: http://download.tensorflow.org/example_images/flower_photos.tgz
14. Unzip that file which will create a "flower_photos" directory:
    a. If on Windows, you may need to install 7-zip or Winzip to unzip the .tgz file.
    b. If on Linux, use: **tar -xvzf flower_photos.tgz**
15. Place the "flower_photos" directory inside directory you created. It should look like the following when done:



3 Retrain Existing Model

For this lab we will retrain an already existing model with new data. This is called transfer learning. The structure of the model has already been setup for image classification, so the goal is to retrain one layer to classify new images with new custom labels. This greatly shortens the amount of time it will take to train the model. Once retrained, the model can be exported in TensorFlow Lite format and ran on the i.MX RT device. The following steps are based on this Google CodeLabs tutorial, however for this lab we will use Mobilenet v1 as the base model:
https://www.tensorflow.org/lite/tutorials/model_maker_image_classification

1. The model that will be used is called MobileNet v1. The script that was downloaded earlier will be used to retrain the model. It won't need to be modified, but open it in a text editor to see the key lines of code:

```
6  #Specify image directory
7  image_path = os.path.join(os.getcwd(), 'flower_photos')
8
9  #Split up images into different training categories for training, validation, and testing.
10 data = ImageClassifierDataLoader.from_folder(image_path)
11 train_data, rest_data = data.split(0.8)
12 validation_data, test_data = rest_data.split(0.5)
13
14 #Retrain model on new images
15 mobilenetv1_spec = model_spec.ImageModelSpec(uri='https://tfhub.dev/google/imagenet/mobilenet_v1_025_128/feature_vector/4')
16 mobilenetv1_spec.input_image_shape = [128, 128]
17 model = image_classifier.create(train_data, model_spec=mobilenetv1_spec, validation_data=validation_data)
18 model.summary()
19
20 #Evaluate final model
21 print('Done training\n')
22 loss, accuracy = model.evaluate(test_data)
23
24 #Write out .tflite file
25 print('Write out model\n')
26 model.export(export_dir='.',tflite_filename='flower_model.tflite',label_filename='flower_labels.txt',with_metadata=False)
```

- **Line #7** specifies the directory to find the images to retrain the model on
- **Lines #10-12** splits up that data into different categories for training, validation, and testing.
- **Line #15** specifies the baseline model to use from the TFHub model zoo. In this case it is Mobilenet v1 model that takes in a 128x128 image.
- **Line #17** does the retraining
- **Line #26** exports the retrained model as a .tflite file

2. Open a command prompt and go to the directory that was created in the last section. It should be something like this:
   cd C:\eiq\flower_lab

3. Then run the script. This may take several minutes to complete.
   python flower_lab.py

4. After running the script, you should see a new file called **flower_model.tflite**. This is the retrained model in TFLite format. A label file named **flower_labels.txt** should also have been created:

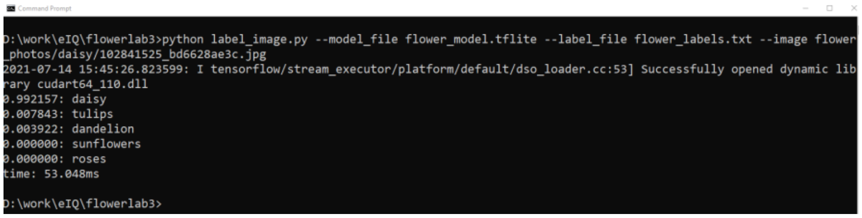
A (D:) › work › eIQ › flowerlab

Name	Date modified	Type	Size
flower_photos	4/1/2021 2:23 PM	File folder	
flower_lab.py	4/1/2021 2:36 PM	Python File	2 KB
flower_labels.txt	4/1/2021 2:40 PM	Notepad++ Docu...	1 KB
flower_model.tflite	4/1/2021 2:40 PM	TFLITE File	884 KB

5. You can test this newly generated model against the flower images using a python script that can be downloaded from here. Place it in the directory and have it analyze a daisy image with the following command as **one long continuous line:**

   python label_image.py --model_file flower_model.tflite --label_file flower_labels.txt --image flower_photos/daisy/102841525_bd6628ae3c.jpg

6. It should respond back that that photo is of a daisy. The confidence level may vary slightly as due to randomness in training, the model will be slightly different each time. If you see a low confidence level (<.80) for identifying that image as a daisy, try running the retraining script again. You may also see a warning about not being able to open cudart64_110.dll but that warning can be ignored (it is an optional ML acceleration library by NVIDIA but is not needed for this lab).



```
D:\work\eIQ\flowerlab3>python label_image.py --model_file flower_model.tflite --label_file flower_labels.txt --image flower
_photos/daisy/102841525_bd6628ae3c.jpg
2021-07-14 15:45:26.823599: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic lib
rary cudart64_110.dll
0.992157: daisy
0.007843: tulips
0.003922: dandelion
0.000000: sunflowers
0.000000: roses
time: 53.048ms

D:\work\eIQ\flowerlab3>
```

4 Convert Model and Labels

Now that the retrained model is running on your laptop, the next step is to convert the TFLite file and labels file into a C header files that can be imported into an MCUXpresso SDK example.

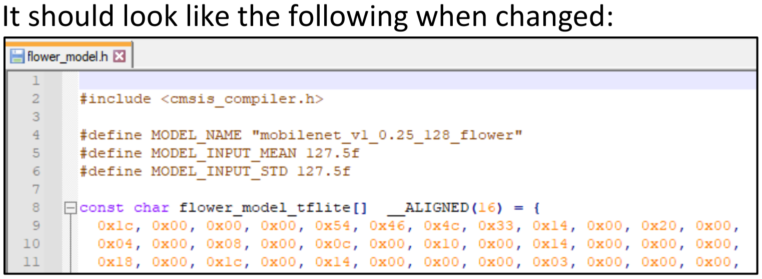
4.1 Convert TensorFlow model

1. Use the **xxd** utility to convert the .tflite binary file into a C array that can be imported into an embedded project.
   - 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 should be about 5.5MB in size and 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.

```
#include <cmsis_compiler.h>

#define MODEL_NAME "mobilenet_v1_0.25_128_flower"
#define MODEL_INPUT_MEAN 127.5f
#define MODEL_INPUT_STD 127.5f

const char flower_model_tflite[]  __ALIGNED(16) = {
```

3. It should look like the following when changed:

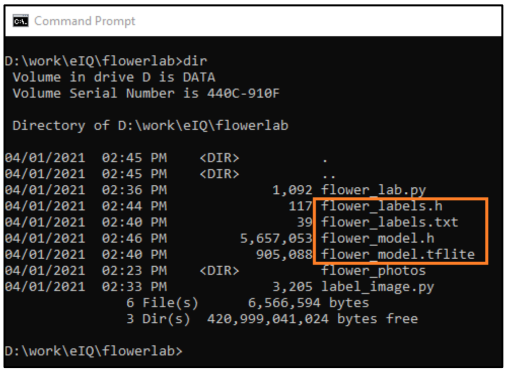


```
flower_model.h

#include <cmsis_compiler.h>

#define MODEL_NAME "mobilenet_v1_0.25_128_flower"
#define MODEL_INPUT_MEAN 127.5f
#define MODEL_INPUT_STD 127.5f

const char flower_model_tflite[]  __ALIGNED(16) = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
  0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00, 0x14, 0x00, 0x00, 0x00,
  0x18, 0x00, 0x1c, 0x00, 0x14, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
```

4. Next, use a text editor to create a new file named flower_labels.h to create an array of the label names. It should look like the following:

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

5. The following files should now be in the directory:

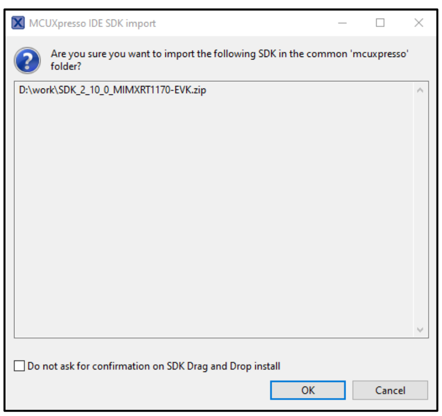


5 Run Demo with TensorFlow Lite for Microcontrollers

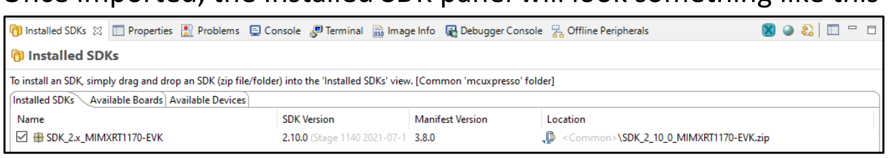
The final step is to take the TensorFlow Lite Micro Label Image example and modify it to use the newly retrained model.

5.1 Copy and Create Files

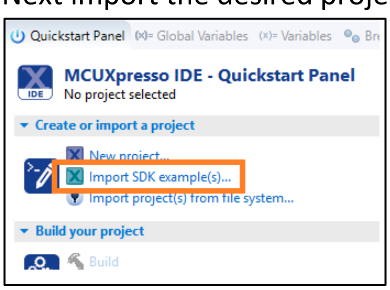
1. Open MCUXpresso IDE and select a workspace location in an empty directory.
2. Drag-and-drop the unzipped SDK folder into the Installed SDKs window, located on a tab at the bottom of the screen named “Installed SDKs”. You will get the following pop-up, so hit OK.



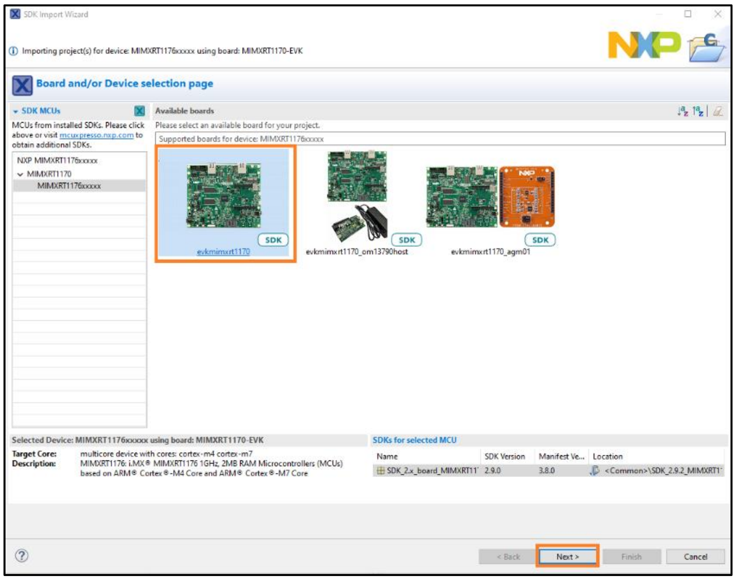
3. Once imported, the Installed SDK panel will look something like this



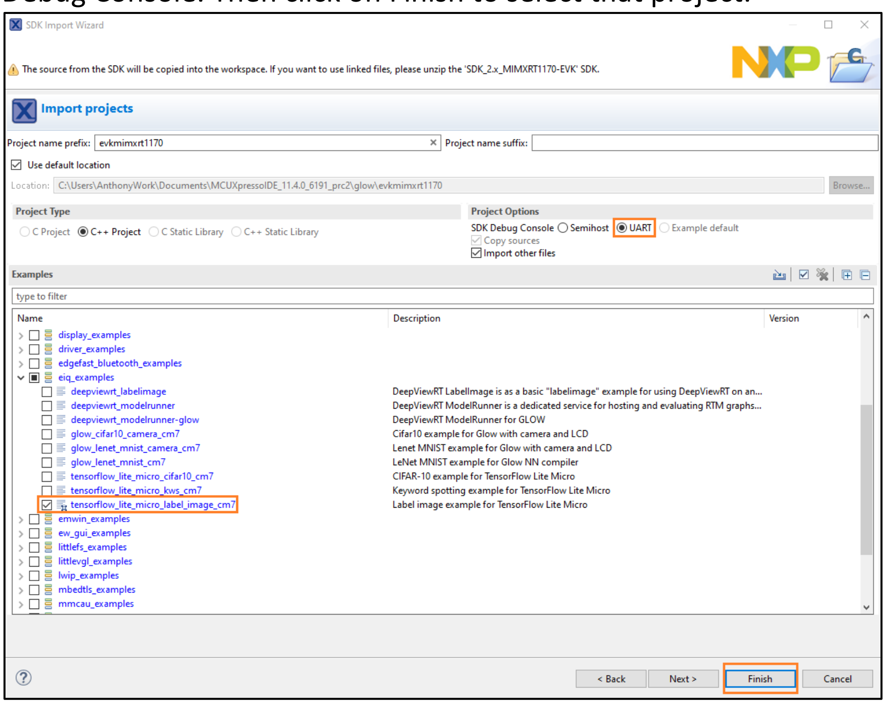
4. Next import the desired project. In the Quickstart Panel, select **Import SDK examples(s)…**



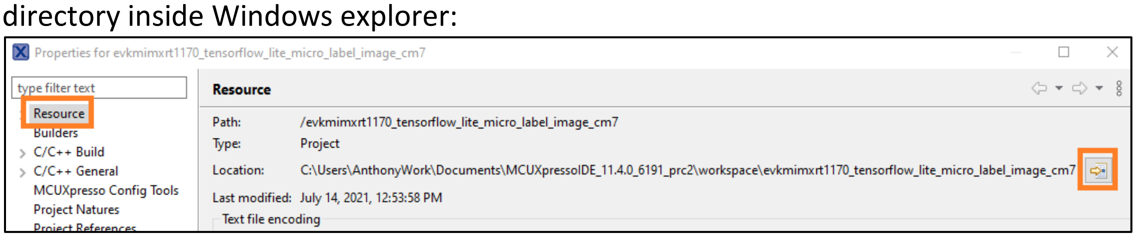
5. Select the evkmimxrt1170 board and click on Next



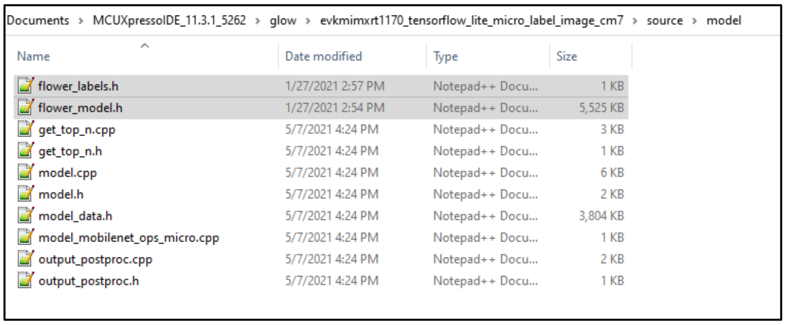
6. Import the **tensorflow_lite_micro_label_image_cm7** example. Also select **UART** for the SDK Debug Console. Then click on Finish to select that project.



6. Now we need to import both the retrained model file and labels file that was generated in the last section into this project.
7. Find the directory location that this example was copied to by right clicking on the project name, and select Properties. In the dialog box that comes up, click on the icon to open up that directory inside Windows explorer:



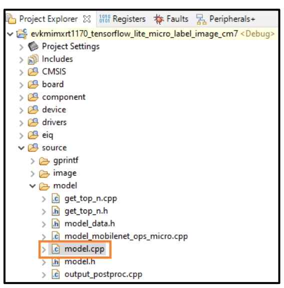
8. Go to the "**source**" directory inside the **evkmimxrt1170_tensorflow_lite_label_image** folder that you just opened. It should be something like:
   C:\Users\user_name\Documents\MCUXpressoIDE_11.4.0_5262\workspace\evkmimxrt1170_tensorflow_lite_micro_label_image_cm7\source\model
9. Inside that **model** directory, copy the **flower_model.h** file and the **flower_labels.h** file generated in the previous section.
10. Directory should look like the following when finished:



5.2 Modify Source Code

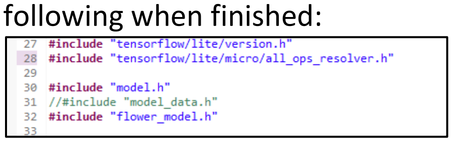
Now edit the source files to include these new files

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

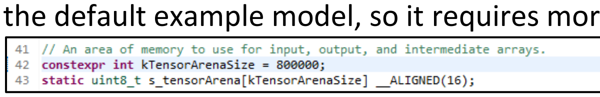


7. On line 28 add the following #include for the ops resolver that supports all the operands used by this retrained model:
   #include "tensorflow/lite/micro/all_ops_resolver.h"

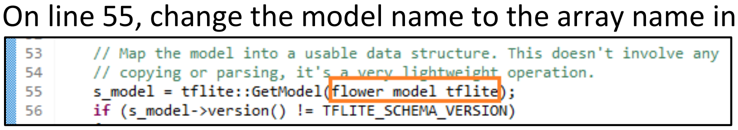
8. On line 31, comment out original #include for the original model defined in **model_data.h**. Then add a new #include to bring in the new model with **flower_model.h**. It should look like the following when finished:



9. On line 42, change the kTensorArenaSize variable to 800000. This flower model is larger than the default example model, so it requires more memory space.



10. On line 55, change the model name to the array name in **flower_model.h**:



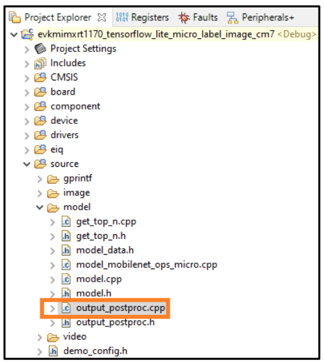
11. To reduce the size of the project, the Label Image example only supports the specific operands required by the default Mobilenet model. Our retrained model uses a few new operands. These specific operands can be determined by analyzing the model with an application called **netron** and then manually 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 73 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:

```
72    // NOLINTNEXTLINE(runtime-global-variables)
73    //tflite::MicroOpResolver &micro_op_resolver = MODEL_GetOpsResolver();
74    tflite::AllOpsResolver micro_op_resolver;
75
76    // Build an interpreter to run the model with.
```

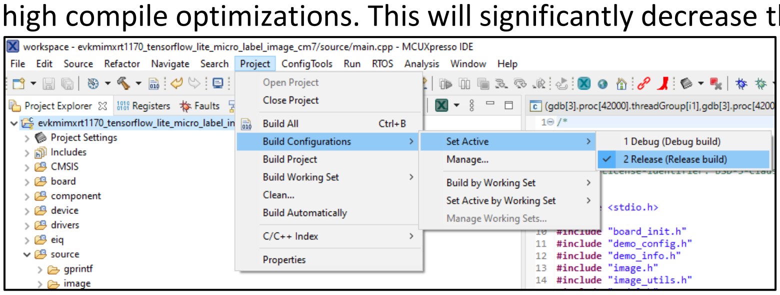
12. Next open the **output_postproc.cpp** file.



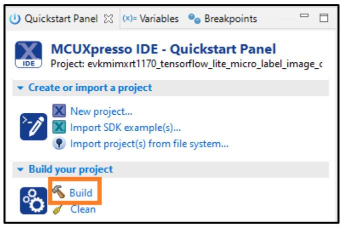
13. On line 13, comment out original #include for the original label file. Then add a new #include to bring in the new labels file. It should look like the following when finished:

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

14. 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.



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



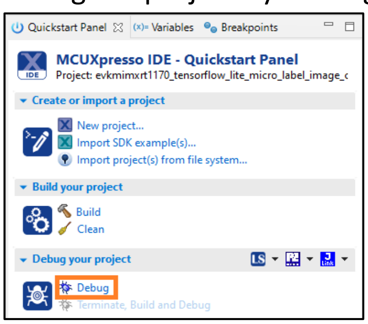
5.3 Attach LCD and Camera

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.

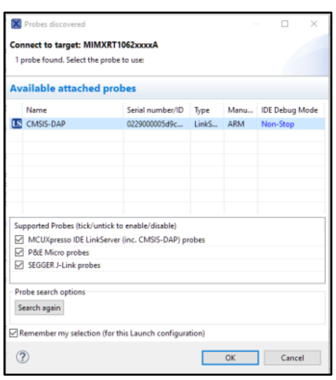
This lab can be completed without using the camera+LCD by running the inferencing on a static image instead, but it is recommended to use the camera+LCD.

5.4 Run Example

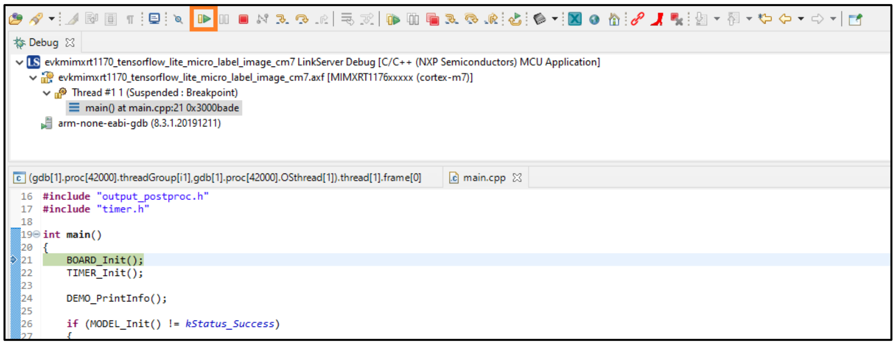
16. Plug the micro-B USB cable into the board at J11 on the i.MXRT1170 board.
17. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
18. Debug the project by clicking on "Debug" in the Quickstart Panel.



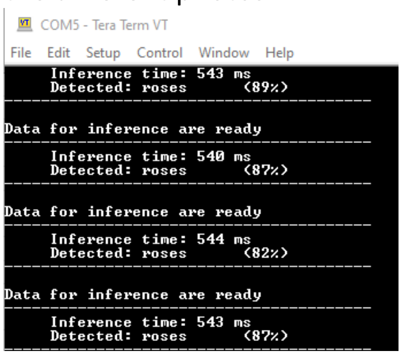
19. It will ask what interface to use. Select CMSIS-DAP.



20. 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.



21. On the LCD screen, you should see what the camera is pointing at.
22. 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.



23. Try opening the daisy photo that was tested earlier on the command line. You may notice it is not as accurate when using the camera. This is because the model was trained on the specific images, and the camera output won't quite match those images, thus leading to decreased accuracy. This is why in a production system, it is best to train on the actual data generated by the camera. There is an example available to show how to capture camera data for machine learning training data with the i.MX RT1060 EVK.
24. You also 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 use TensorFlow to generate a retrained TensorFlow model in TFLite format that can be imported and ran on an embedded system using the eIQ software platform.

This particular model was used to classify flower images. However, the model can also be trained on other types of images by re-running the script. Just add a new directory name and example images of that classification to the flower_photos directory, and new images can be recognized by this model.

Other types of TensorFlow models can be converted to TFLite format as well by using the tools included with TensorFlow or by using the eIQ Model Tool that is part of the eIQ Toolkit. The eIQ Toolkit can also be used to generate new image classification models that can be inferenced with TFLM.

By enabling machine learning in embedded systems, there's a wide world of opportunity for new smarter applications.