



eIQ™ Inference with Tensorflow Lite for Microcontrollers on i.MX RT1170 - Without Camera

Lab Hand Out - Revision 4

Contents

1 Lab Overview	3
2 Software and Hardware Installation	3
2.1 NXP MCUXpresso SDK Installation.....	3
2.2 TensorFlow Installation.....	4
2.3 Lab Setup.....	4
3 Retrain Existing Model	5
4 Convert Model and Labels	7
4.1 Convert TensorFlow model.....	7
4.2 Image Data Conversion – No Camera Only	8
5 Run Demo with TensorFlow Lite for Microcontrollers	8
5.1 Copy and Create Files.....	8
5.2 Modify Source Code	10
5.3 Run Example.....	12
6 Conclusion.....	13

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 NXP's eIQ software package, the model can be run on an i.MX RT embedded device. A camera attached to the board could then be used to look at photos of flowers and the model will determine what type of flower the camera is looking at.

This version of the lab does not require use of a camera or LCD connected to the EVB. Instead of using a camera to look at images of flowers to run the inference on, a static image will be converted to a C array and loaded at compile time.

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

- i.MX RT595
- i.MX RT685
- i.MX RT1050
- i.MX RT1060
- i.MX RT1064
- i.MX RT1160
- i.MX RT1170

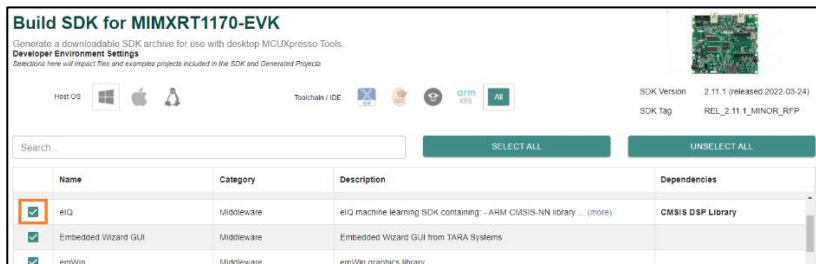
The i.MX RT1060, i.MX RT1064, i.MX RT1170 EVKs come with a camera sensor and a version of this lab that makes use of that camera can be found in the “**eIQ TensorFlow Lite for Microcontrollers Lab for RT1170 – With Camera.pdf**” document.

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.

2.2 TensorFlow Installation

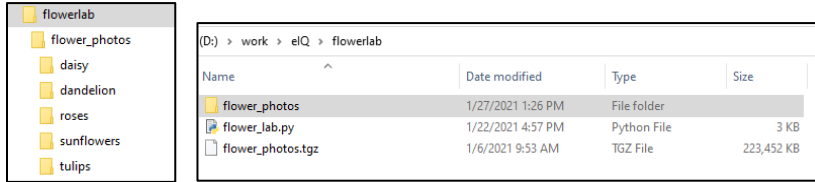
4. Download and install Python 3.9. ****The 64-bit edition is required and for compatibility it is highly recommended to use Python 64-bit 3.9.x****: <https://www.python.org/downloads/>
5. Open a Windows command prompt and verify that the python command corresponds to Python 3.9.x.
python -V
6. Update 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==2.6
python -m pip install keras==2.6
python -m pip install tensorflow-estimator==2.6.0
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 tf-lite-model-maker
python -m pip install numpy scipy matplotlib ipython jupyter pandas sympy nose imageio
python -m pip install netron seaborn west pyserial sklearn opencv-python pillow
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:

```

1 import os
2 import tensorflow as tf
3 from tf_lite_model_maker import image_classifier
4 from tf_lite_model_maker import ImageClassifierDataLoader
5 from tf_lite_model_maker.config import QuantizationConfig
6
7 #Specify image directory
8 image_path = os.path.join(os.getcwd(), 'flower_photos')
9
10 #Split up images into different training categories for training, validation, and testing.
11 data = ImageClassifierDataLoader.from_folder(image_path)
12 train_data, rest_data = data.split(0.5)
13 validation_data, test_data = rest_data.split(0.5)
14
15 #Retrain model on new images
16 mobilenetv1_spec = image_classifier.ModelSpec(uri='https://tfhub.dev/google/imagenet/mobilenet_v1_025_128/feature_vector/5/')
17 print("MEAN", mobilenetv1_spec.mean_rgb)
18 print("STD", mobilenetv1_spec.stddev_rgb)
19 mobilenetv1_spec.input_image_shape = [128, 128]
20 model = image_classifier.create(train_data, model_spec=mobilenetv1_spec, validation_data=validation_data)
21 model.summary()
22
23 #Evaluate final model
24 print('Done training\n')
25 loss, accuracy = model.evaluate(test_data)
26
27 #Write out .tflite file
28 print('Write out model\n')
29 config = QuantizationConfig(inference_input_type=tf.int8,inference_output_type=tf.int8,supported_ops=[tf.lite.OpsSet.TFLITE_BUILTINS_INT8],representative_data = test_data)
30 model.export(export_dir='.',tflite_filename='flower_model.tflite',label_filename='flower_labels.txt',with_metadata=False,quantization_config=config)
31
32 #Evaluate quantized TFLite model
33 print(model.evaluate_tflite('flower_model.tflite', test_data))
34

```

- **Line #8** specifies the directory to find the images to retrain the model on
- **Lines #11-12** splits up that data into different categories for training, validation, and testing.
- **Line #16** 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 #20** does the retraining
- **Line #30** exports the retrained model as a quantized .tflite file
- **Line #33** verifies the export .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
```

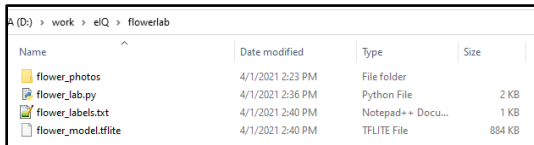
3. Then run the script.

```
python flower_lab.py
```

This may take several minutes to complete depending on the speed of your PC. At the end of the script it will evaluate the generated TFLite model. The last verification step does not have any progress output and will take some time to run. The confidence level may vary slightly each time the script is ran due to randomness in training, so the model will be slightly different each time. If you see a low confidence level (<.80) 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).

```
2022-04-12 15:50:03.268102: I tensorflow/compiler/mlir/lite/python/tf_flatbuffer_helpers.cc:354] Ignored drop_control_dependency.
2022-04-12 15:50:03.317014: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:210] disabling MLIR crash reproducer, set env var 'MLIR_CRASH_REPRODUCER_DIRECTORY' to enable.
fully_quantize: 0, Inference_type: 6, input_inference_type: 9, output_inference_type: 9
('accuracy': 0.8392370572207084)
D:\flowerlab>
```

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:



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 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 2.1MB in size and will need to be modified slightly to integrate 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.

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

#define MODEL_NAME "mobilenet_v1_0.25_128_flower"
#define MODEL_INPUT_MEAN 0.0f
#define MODEL_INPUT_STD 255.0f

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. The order and labels will be based on the labels generated in flower_labels.txt but modified to be a C array. Copy and paste the following code into that newly created **flower_labels.h** file:

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

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

4.2 Image Data Conversion – No Camera Only

Now that the model has been converted, an image also needs to be converted into a C array. Preferably pick an image in the photo flower repository so that result on the PC can be compared to the result on the RT1170. For this lab, we'll pick the file **flower_photos/daisy/102841525_bd6628ae3c.jpg**

- Use the following Python script program to convert the selected JPG file to a C array. Make sure the path to the image is correct relative to the location the script is executed from and also make sure to include the indentions of the last 4 lines:

```
import cv2
import numpy as np
img = cv2.imread('flower_photos/daisy/102841525_bd6628ae3c.jpg')
img = cv2.resize(img, (128, 128))
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
with open('daisy.h', 'w') as fout:
    print('#define STATIC_IMAGE_NAME "daisy"', file=fout)
    print('static const uint8_t daisy [] = {', file=fout)
    img.tofile(fout, ', ', '0x%02X')
    print('};\n', file=fout)
```

- It should look like the following when you open the newly generated daisy.h file:

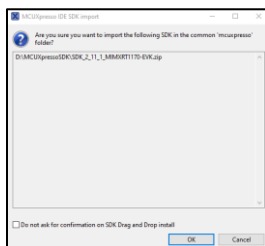
```
#define STATIC_IMAGE_NAME "daisy"
static const uint8_t daisy [] = {
0x59, 0x59, 0x71, 0x8D, 0x8C, 0x82, 0xc6, 0xc3, 0x8a, 0xc3, 0xc5, 0x8b,
0xc5, 0xc5, 0xa5, 0x8a, 0xd6, 0x86, 0x81, 0x80, 0xc4, 0x8b, 0xd9, 0xc8,
0xc6, 0xc8, 0x8b, 0xab, 0xab, 0xa2, 0x70, 0x8a, 0x63, 0x47, 0x43, 0x3d,
0x50, 0x4d, 0x46, 0x52, 0x53, 0x4d, 0x43, 0x41, 0x3d, 0x2c, 0x2b, 0x26,
0x2f, 0x25, 0x1d, 0x21, 0x23, 0x19, 0x18, 0x18, 0x1d, 0x1b, 0x23, 0x18,
```

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

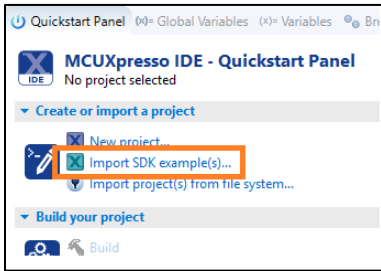
- Open MCUXpresso IDE and select a workspace location in an empty directory.
- Drag-and-drop the unzipped SDK folder into the Installed SDKs window, located on a tab at the bottom of the screen named "Installed SDKs". You will get the following pop-up, so hit OK.



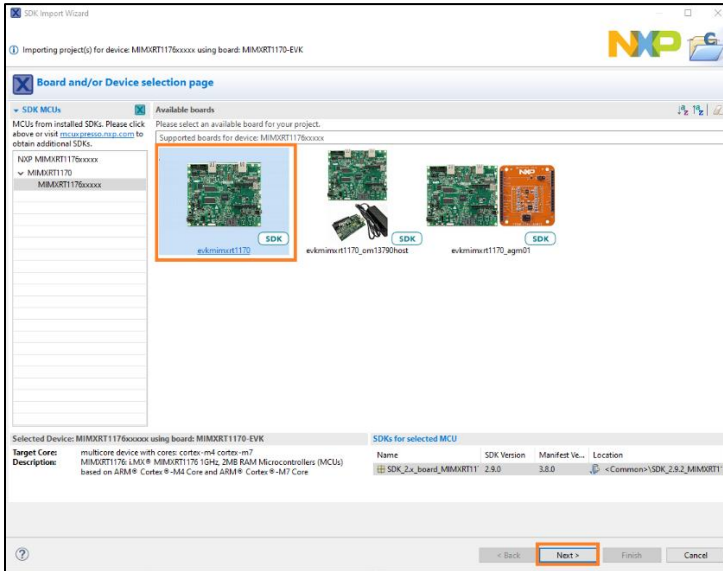
- Once imported, the Installed SDK panel will look something like this



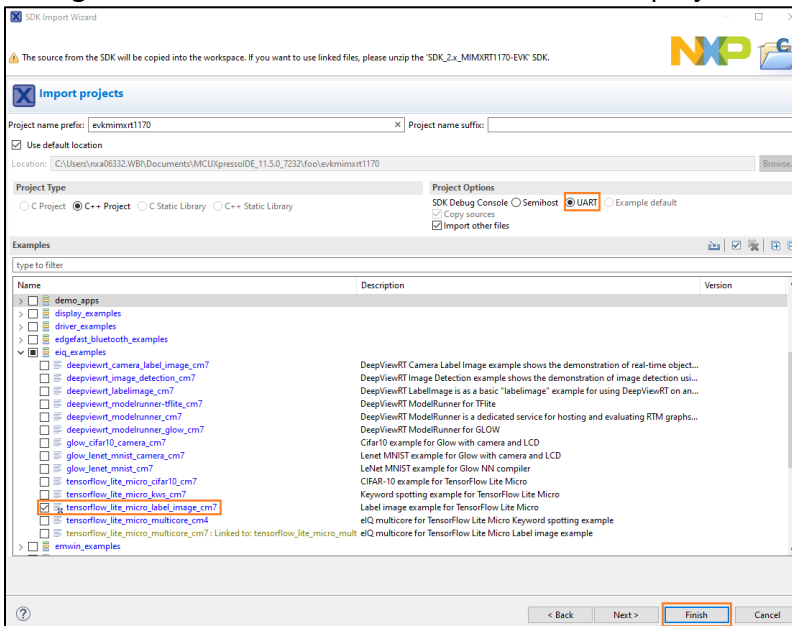
- Next import the desired project. In the Quickstart Panel, select **Import SDK example(s)...**



- Select the **evkmimxrt1170** board and click on Next

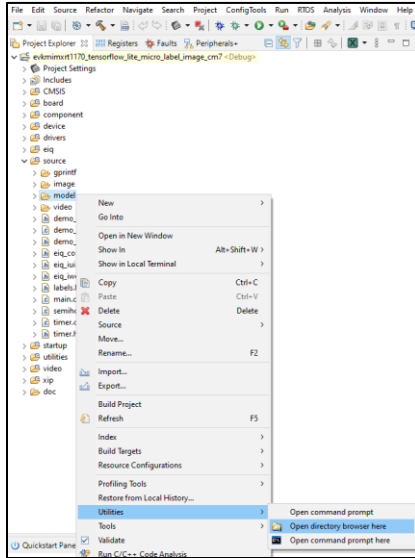


- Import the **tensorflow_lite_micro_label_image_cm7** example. Also select **UART** for the SDK Debug Console. Then click on Finish to select that project.



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

- 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**



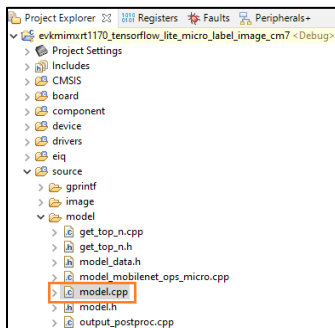
- It should open a directory at something like something like:
C:\Users\user_name\Documents\MCUXpressoIDE_11.5.0_5262\workspace\evkmimxrt170_tensorflow_lite_micro_label_image_cm7\source\model
- Inside that **model** directory, copy the **flower_model.h**, **flower_labels.h**, and **daisy.h** files generated in the previous section.
- Directory should look like the following when finished:

Name	Date modified	Type	Size
daisy.h	4/12/2022 4:00 PM	Notepad++ Document	289 KB
flower_labels.h	4/12/2022 4:10 PM	Notepad++ Document	1 KB
flower_model.h	4/12/2022 4:05 PM	Notepad++ Document	2,117 KB
get_top_n.cpp	4/12/2022 4:04 PM	Notepad++ Document	3 KB
get_top_n.h	4/12/2022 4:04 PM	Notepad++ Document	1 KB
model.cpp	4/12/2022 4:04 PM	Notepad++ Document	6 KB
model.h	4/12/2022 4:04 PM	Notepad++ Document	2 KB
model_all_ops_micro.cpp	4/12/2022 4:04 PM	Notepad++ Document	1 KB
model_data.h	4/12/2022 4:05 PM	Notepad++ Document	3,805 KB
output_postproc.cpp	4/12/2022 4:04 PM	Notepad++ Document	2 KB
output_postproc.h	4/12/2022 4:04 PM	Notepad++ Document	1 KB

5.2 Modify Source Code

Now edit the source files to include these new files

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



- On line 27 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"`
- 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:

```
26 #include "tensorflow/lite/schema/schema_generated.h"
27 #include "tensorflow/lite/micro/all_ops_resolver.h"
28
29 #include "model.h"
30 //#include "model_data.h"
31 #include "flower_model.h"
```

- 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.

```
41 // An area of memory to use for input, output, and intermediate arrays.
42 constexpr int kTensorArenaSize = 800000;
43 static uint8_t s_tensorArena[kTensorArenaSize] __ALIGNED(16);
```

- On line 55, change the model name to the array name in `flower_model.h`:

```
53 // Map the model into a usable data structure. This doesn't involve any
54 // copying or parsing, it's a very lightweight operation.
55 s_model = tflite::GetModel(flower_model_tflite);
56 if (s_model->version() != TFLITE_SCHEMA_VERSION)
```

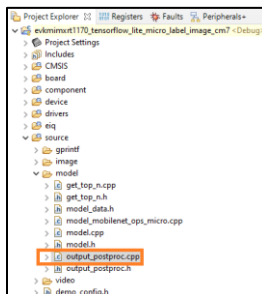
- To reduce the size of the project, the Label Image example only supports the specific operands required by the default Mobilenet model. Our retrained model uses a few new operands. These specific operands can be determined by analyzing the model with an application called **netron** and then manually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

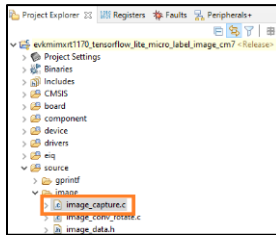
- Next open the `output_postproc.cpp` file.



- On line 13 in `output_postproc.cpp`, 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 "demo_config.h"
13 //#include "labels.h"
14 #include "flower_labels.h"
```

13. Next open the **image_capture.c** file



14. Make the following changes to bring in the daisy image for inferencing in image_capture.c:

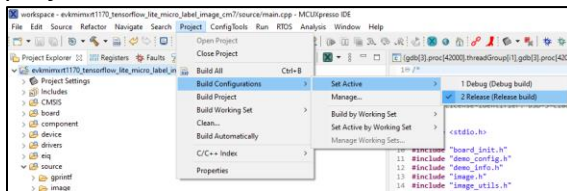
- a. Comment out the #include on line 13 for image_data.h and add **#include "daisy.h"**.

```
13 //include "image_data.h"
14 #include "daisy.h"
```

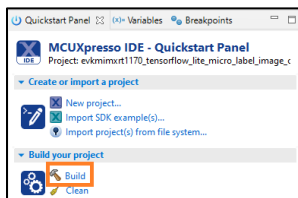
- b. Change line 53 for the IMAGE_Decode argument to the name of the daisy array, **daisy**

```
47 status_t IMAGE_GetImage(uint8_t* dstData, int32_t dstWidth, int32_t dstHeight, int32_t dstChannels)
48 {
49     /* Switch to camera image capture after one static image. */
50     if (s_static)
51     {
52         s_static = false;
53         printf(EOL "Static data processing:" EOL);
54         return IMAGE_Decode(daisy, dstData, dstWidth, dstHeight, dstChannels);
55     }
56     else
57     {
```

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 enable high compile optimizations. This will significantly decrease the inference time of TFMicro projects.

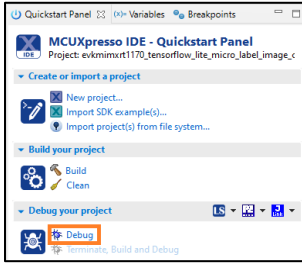


15. Build the project by clicking on “Build” in the Quickstart Panel and make sure there are no errors.

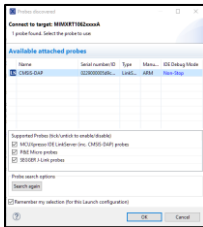


5.3 Run Example

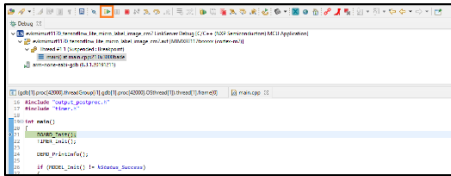
16. Plug the micro-B USB cable into the board at J11 on the i.MXRT1170 board.
17. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
18. Debug the project by clicking on “Debug” in the Quickstart Panel.



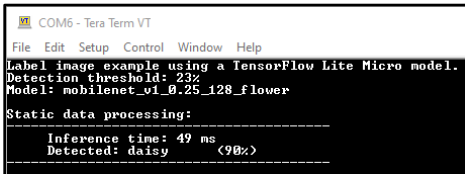
19. It will ask what interface to use. Select CMSIS-DAP.



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. Open up a terminal window, and you the result of the inference from the static image:



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](#) in .tflite format that can be inferred with TFLM. The model used in the Python script can also be changed to improve accuracy at the cost of inference time due to the increased complexity.

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