



eIQ: Transfer Learning – With Camera

Lab Hand Out - Revision 5

Contents

1	Lab Overview	3
2	Software and Hardware Installation	3
2.1	NXP SDK Installation	3
2.2	TensorFlow Installation.....	5
2.3	Lab scripts Installation.....	6
2.4	Attach LCD and Camera.....	6
3	Retrain Existing Model	8
4	Convert Model and Data.....	10
4.1	Convert TensorFlow model.....	10
5	Run Demo	12
5.1	Copy and Create Files.....	12
5.2	Modify Source Code	15
5.3	Run Example.....	16
6	Conclusion.....	18

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 inference engine that is part of NXP's eIQ 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 Transfer Learning Lab – Without Camera.pdf" document.

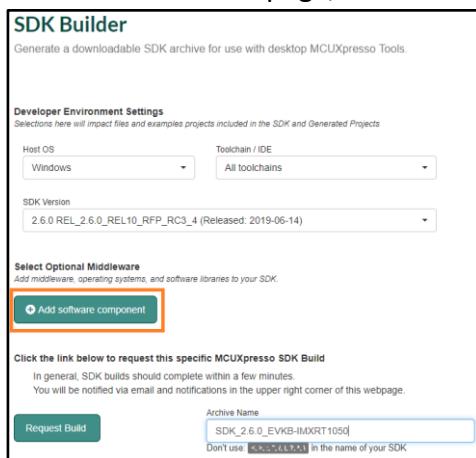
This lab is written for the RT1060-EVK. It can also be used with the RT1050-EVKB and RT1064-EVK with minor modifications. Also note that the RT1060-EVK and RT1064-EVK come with a camera sensor. The RT1050-EVKB does not come with a camera sensor. In all cases the LCD must be purchased separately.

2 Software and Hardware Installation

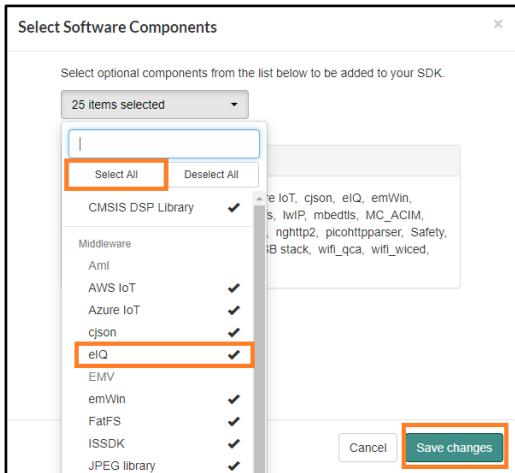
This section will cover the steps needed to install the eIQ software and TensorFlow on your computer. It will also cover how to attach the LCD and camera to the RT1060 board.

2.1 NXP SDK Installation

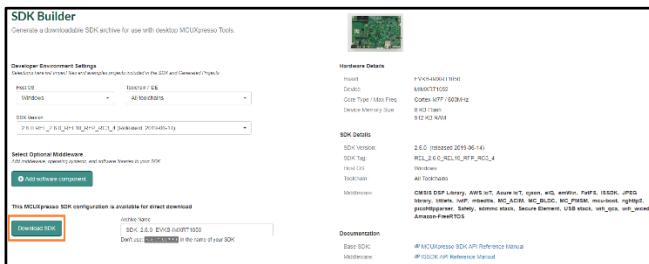
1. Install the latest version of [MCUXpresso IDE](#)
2. Install a terminal program like [TeraTerm](#).
3. Download the latest MCUXpresso SDK for i.MXRT1060. It includes the eIQ software platform and demos:
 - a) Go to the SDK builder:
 - a. If using RT1060-EVK or RT1064-EVK use:
<https://mcuxpresso.nxp.com/en/select?device=EVK-MIMXRT1060>
 - b. If using RT1050-EVKB use:
<https://mcuxpresso.nxp.com/en/select?device=EVKB-IMXRT1050>
 - b) On the SDK builder page, click on "Add software component"



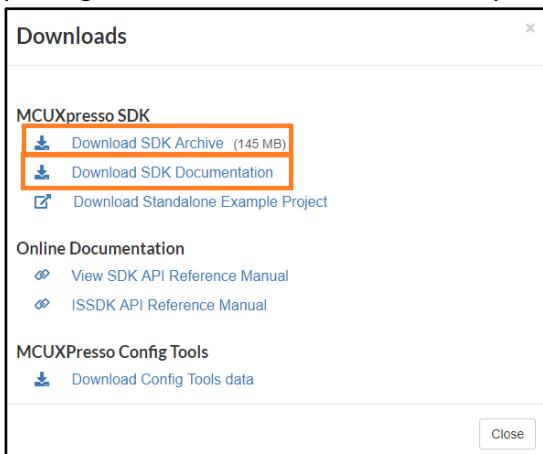
- c) Select **eIQ** and **CMSIS DSP Library** to just get the eIQ software. Alternatively, to get all the software packages for the device, click on “Select All” and verify the eIQ software option is now checked. Then click on “Save Changes”



- d) Click on Download SDK.



- e) Accept the license agreement
f) In the pop-up, download both the SDK Archive (containing the source code and project files) package and the SDK Documentation package (containing the eIQ documentation)



4. Download the elQupdate.exe file which will be used to add camera and LCD support:
<https://community.nxp.com/docs/DOC-344227>

5. Update the SDK with the files from the eIQ update installer:
 - a) Unzip the SDK zip file
 - b) Run the eIQ update installer and make a note of the extraction location.
 - c) Delete the **eiq_examples** folder in the SDK at:
 \SDK_2.6.2_EVK-MIMXRT1060\boards\evkmimxrt1060
 - d) Copy the **eiq_examples** folder from the eIQ update folder for your board and put it at:
 \SDK_2.6.2_EVK-MIMXRT1060\boards\evkmimxrt1060
 - e) Delete the **eiq** folder in the SDK at:
 \SDK_2.6.2_EVK-MIMXRT1060\middleware
 - f) Copy the **eiq** folder from the eIQ update folder for your board and put it at:
 \SDK_2.6.2_EVK-MIMXRT1060\middleware
 - g) Delete the **drivers** folder in the SDK at:
 \SDK_2.6.2_EVK-MIMXRT1060\devices\MIMXRT1062
 - h) Copy the **drivers** folder from the eIQ update folder for your board and put it at:
 \SDK_2.6.2_EVK-MIMXRT1060\devices\MIMXRT1062

2.2 TensorFlow Installation

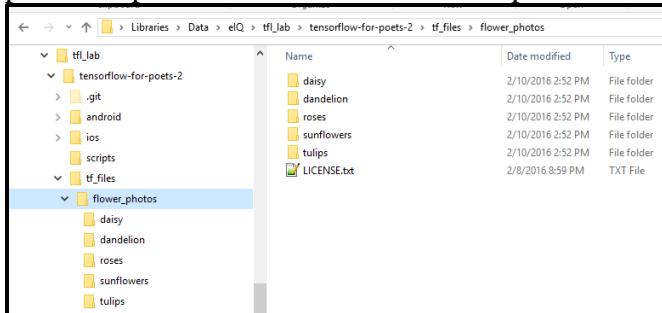
6. Download and install Python 3.7. ****The 64-bit edition is required**:**
<https://www.python.org/downloads/>
7. Verify that the python command corresponds to Python 3.x. You may need to use “python3” for all the commands instead:
python -V
8. Update the python installer tools:
python -m pip install -U pip
python -m pip install -U setuptools
9. Install the Tensorflow libraries and support for python. Need at least version 1.13.1 if you already had TensorFlow installed previously.
python -m pip install tensorflow
10. 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 tensorflow-datasets
python -m pip install numpy scipy matplotlib ipython jupyter pandas sympy nose
python -m pip install opencv-python
python -m pip install PILLOW
python -m pip install netron
11. If on Windows, install Vim 8.1: <https://www.vim.org/download.php#pc>. There is a binary convertor programmed named xxd.exe located inside that package that will be needed.
12. 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>

13. Verify the PATH was set correctly by typing “`tflite_convert`” and “`xxd -v`” in a command prompt. You should not get any errors about an unrecognized command.

2.3 Lab scripts Installation

The python scripts that will be used to retrain an already existing model will be downloaded via Git. 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.

14. Install Git: <https://git-scm.com/downloads>
 15. Open a command prompt, and in a directory of your choosing, download the tutorial repository with git:
`git clone https://github.com/googlecodelabs/tensorflow-for-poets-2`
 16. 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
 17. 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`
 18. Place the “flower_photos” directory inside the “tf_files” directory that is in the tensorflow-for-poets-2 repo downloaded in the first step. It should look like the following when done:

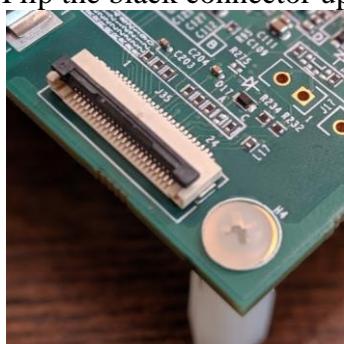


2.4 Attach LCD and Camera

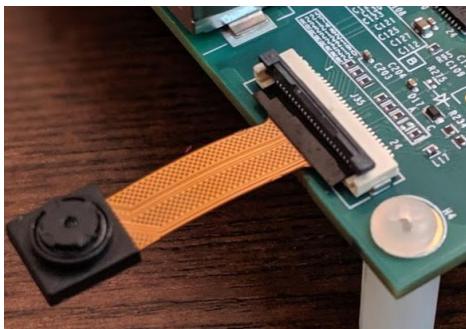
Now attach the camera and LCD. The camera is only available as part of the RT1060-EVK or RT1064-EVK. The [LCD](#) can be bought separately. This lab can be completed without using the camera+LCD, but it is recommended to use them.

Note: When the camera is connected, the on-board microphone cannot be used since they share SAI pins.

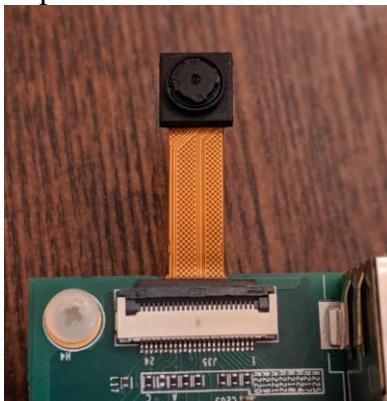
19. The camera connector is on J35
a. Flip the black connector up.



- b. Then slide in the flat ribbon connector of the camera



- c. Flip the black connector back down. It should keep the ribbon cable snug.

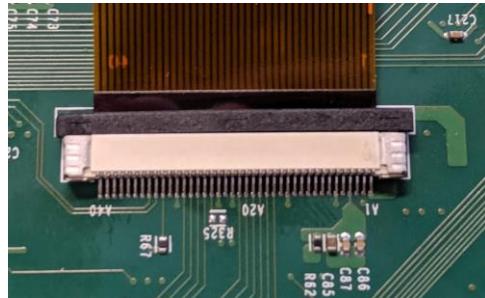


20. Attach the LCD on the back of the board.

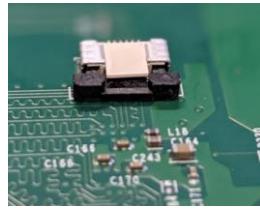
- a. On the back of the board, slide the black connector forward.



- b. Then slide in the flat LCD ribbon cable underneath the black connector.
c. Slide the black connector back to its original position. The cable should be snug.



- d. If desired, do the same for the touch controller connector. It is not used for this lab but is used with other demos.
e. Slide the black connector forward



- f. Then insert the cable between the black connector and the white top so that it is in the middle. It might take a few tries. You could also use needle nose pliers to help guide in the cable but be careful about damaging the cable.
- g. Then slide the black connector back to the original position. The cable should be snug.
- h. It should look like the following when complete:



- i. After testing out the LCD with this lab to ensure it was connected correctly, you can use thick foam tape to keep the LCD attached to the board.



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 converted to TensorFlow Lite format and ran on the i.MXRT device. The following steps are based on this Google CodeLabs tutorial:

TensorFlow for Poets <https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/index.html>

1. Open a command prompt and go to the directory that was created by cloning the git repository in the last section. It should be something like this:
cd C:\eiq\tensorflow-for-poets-2
2. The model being used is called MobileNet. We will retrain this model for 128x128 pixel images using a python script found inside the tutorial folder. Navigate to the main root directory and run the following command.

This should be one long continuous line like in the image below:

```
python -m scripts.retrain
--bottleneck_dir=tf_files/bottlenecks
--how_many_training_steps=500
--model_dir=tf_files/models/
--summaries_dir=tf_files/training_summaries/mobilenet_0.25_128
--output_graph=tf_files/retrained_graph.pb
--output_labels=tf_files/retrained_labels.txt
--architecture=mobilenet_0.25_128
--image_dir=tf_files/flower_photos
```

```
C:\Data\eiQ\lab\tensorflow-for-poets-2>python -m scripts.retrain --bottleneck_dir=tf_files/bottlenecks --how_many_training_steps=500 --model_dir=tf_files/models --summaries_dir=tf_files/training_summaries/mobilenet_0.25_128 --output_graph=tf_files/retrained_graph.pb --output_labels=tf_files/retrained_labels.txt --architecture=mobilenet_0.25_128 --image_dir=tf_files/flower_photos
```

3. This will take several minutes to run. While waiting, here's an explanation for the arguments in the command you just ran:
 - **--bottleneck_dir=tf_files/bottlenecks**
Directory to store cached data information
 - **--how_many_training_steps=500**
of iterations. The more iterations the longer the training takes, but the more accurate the model will likely be
 - **--model_dir=tf_files/models/**
Directory location to download the pre-existing model
 - **--summaries_dir=tf_files/training_summaries/mobilenet_0.25_128**
Output directory for data files used by an optional analysis program called TensorBoard
 - **--output_graph=tf_files/retrained_graph.pb**
Name of the retrained model in Protocol Buffer (pb) format.
 - **--output_labels=tf_files/retrained_labels.txt**
Text file with the labels for the model (determined by the names of the sub-directories of the training data)
 - **--architecture=mobilenet_0.25_128**
The particular type of MobileNet model to use as a starting point
 - **--image_dir=tf_files/flower_photos**
Location of the data to train the model on. Each sub-directory is a label classifying those images
4. Once finished, look inside the **tf_files** directory. You should have a model file named **retrained_graph.pb**. This is the newly trained model.

5. You can test this model file against flower images using the `label_image` python script. From the main directory, call a script to use the new graph and have it analyze a daisy image with the following command as one long continuous line:

```
python -m scripts.label_image
--graph=tf_files/retrained_graph.pb
--input_height=128
--input_width=128
--image=tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
```

6. It should respond back that that photo is of a daisy. The confidence level may vary slightly as the model training 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.

```
C:\Data\elQ\dfae_lab\tensorflow-for-poets-2>python -m scripts.label_image --graph=tf_files/retrained_graph.pb --input_height=128 --input_width=128 --image=tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
2019-05-15 13:49:55.342850: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
Evaluation time (1-image): 0.283s
daisy (score=0.96397)
dandelion (score=0.02553)
roses (score=0.00974)
sunflowers (score=0.00073)
tulips (score=0.00004)
```

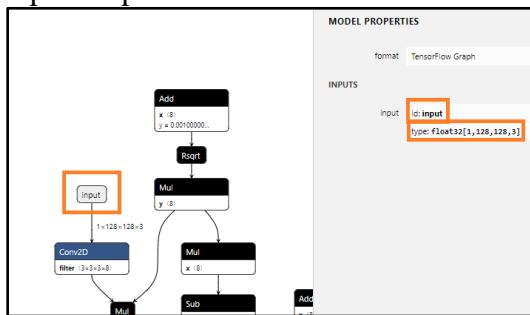
4 Convert Model and Data

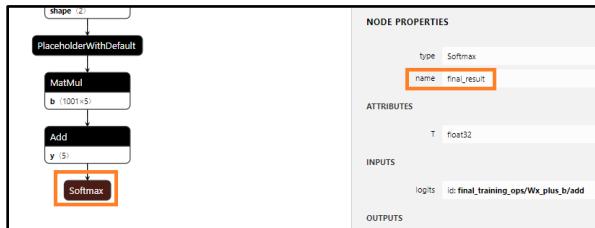
Now that the retrained model is running on your laptop, the next step is to use a TensorFlow utility named `tflite_convert` to convert that model into a file that can be loaded onto an embedded device like the i.MXRT1060.

4.1 Convert TensorFlow model

Convert the .pb model file into a format that can be imported into the RT1060 project with `tflite_convert`.

- The first and last layer names need to be read from the TensorFlow .pb model file, which will be used later in the conversion process. This can be done with a neural network visualization tool called Netron.
 - Use **Netron** by using the following command on a .pb file:
netron tf_files/retrained_graph.pb
 - While the command is running, open up a web browser and navigate to <http://localhost:8080> and click on the first and last nodes to get the layer names and the input shape.





- c. After reading the labels, go back to the terminal window and hit Ctrl+c to stop the server and return to the command prompt
2. Use **tflite_convert** to transform the model file into a .tflite file. There are options available to quantize and optimize the model during this step, but they seem to significantly decrease the accuracy of the converted model. The following options keep the accuracy about the same as the full model. Type in the command as one long continuous line like below:

```
tflite_convert
--graph_def_file=tf_files/retrained_graph.pb
--output_file=tf_files/retrained_graph.tflite
--input_shape=1,128,128,3
--input_array=input
--output_array=final_result
--inference_type=FLOAT
--input_data_type=FLOAT
```

```
C:\Data\elQ\lab\tensorflow-for-poets-2>tflite_convert --graph_def_file=tf_files/retrained_graph.pb --output_file=tf_files/retrained_graph.tflite --input_shape=1,128,128,3 --input_array=input --output_array=final_result --inference_type=FLOAT --input_data_type=FLOAT
2019-08-20 13:46:21.592793: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2019-08-20 13:46:21.478210: I tensorflow/core/grappler/devices.cc:60] Number of eligible GPUs (core count > 8, compute capability > 0.0): 0 (Note: TensorFlow was not compiled with CUDA support)
2019-08-20 13:46:21.494870: I tensorflow/core/grappler/clusters/single_machine.cc:359] Starting new session
2019-08-20 13:46:21.565124: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:716] Optimization results for graph: retrained_graph.tflite
2019-08-20 13:46:21.568205: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:718] constant folding: Graph size after: 177 nodes (-383), 176 edges (-410), time = 48.93ms
2019-08-20 13:46:21.570553: I tensorflow/core/grappler/optimizers/meta_optimizer.cc:718] constant folding: Graph size after: 177 nodes (0), 176 edges (0), time = 10.385ms.
C:\Data\elQ\lab\tensorflow-for-poets-2>
```

Here is what each of those arguments determine:

- **--graph_def_file=tf_files/retrained_graph.pb**
Name of the TensorFlow model to convert
- **--output_file=tf_files/retrained_graph.tflite**
Name of the converted tflite file
- **--input_shape=1,128,128,3**
This model takes in a 128 x 128 image that has 3 color channels
- **--input_array=input**
Name of the first layer of the model
- **--output_array=final_result**
Name of the last layer of the model
- **--inference_type=FLOAT**
Model uses floating (instead of 8-bit quantized) inference
- **--input_data_type=FLOAT**
Model uses floating (instead of 8-bit quantized) input

3. You may get a warning about AXV2 instructions not being supported. You can ignore this warning.

4. Inside the tf_files directory you should see a new file named **retrained_graph.tflite**
5. Use the **xxd** utility to convert the .tflite binary file into a C array that can be imported into an embedded project. Do not run this command in Powershell as it will cause compilation issues later. Run it in a standard command prompt:
xxd -i tf_files/retrained_graph.tflite > tf_files/retrained_graph.h
6. The generated header file may need to be modified slightly to define it as a const. Open up the file and, if necessary, change “unsigned char” to “const char”. Also make note of the array name as it will be used in the next section.

```
retrained_graph.h
1 const char tf_files_retrained_graph_tflite[] = {
2     0x18, 0x00, 0x00, 0x00, 0x54, 0x46, 0xc, 0x33, 0x00, 0x00, 0xe, 0x0,
3     0x18, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00, 0x14, 0x00,
4     0x0e, 0x00, 0x00, 0x03, 0x00, 0x00, 0x80, 0x0c, 0x1d, 0x00,
5     0x0c, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00,
```

7. The following files should now be in the tf_files directory:

```
C:\Data\el0\lab\tensorflow-for-poets-2>dir tf_files
Volume in drive C is OSDisk
Volume Serial Number is 264B-1211

Directory of C:\Data\el0\lab\tensorflow-for-poets-2\tf_files

08/20/2019 01:54 PM <DIR> .
08/20/2019 01:54 PM <DIR> ..
08/20/2019 01:32 PM 0 .empty
08/20/2019 01:32 PM <DIR> bottlenecks
08/20/2019 01:35 PM <DIR> flower_photos
08/20/2019 01:38 PM <DIR> models
08/20/2019 01:54 PM 11,900,817 retrained_graph.h
08/20/2019 01:48 PM 2,020,051 retrained_graph.pb
08/20/2019 01:46 PM 1,904,112 retrained_graph.tflite
08/20/2019 01:48 PM 40 retrained_labels.txt
08/20/2019 01:38 PM <DIR> training_summaries
5 File(s) 15,825,028 bytes
6 Dir(s) 429,096,202,248 bytes free

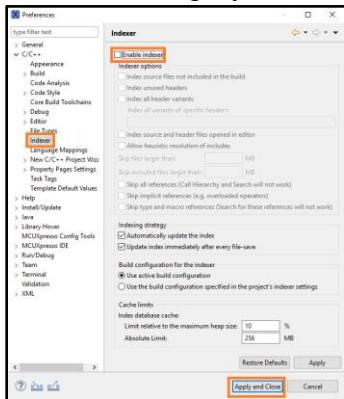
C:\Data\el0\lab\tensorflow-for-poets-2>
```

5 Run Demo

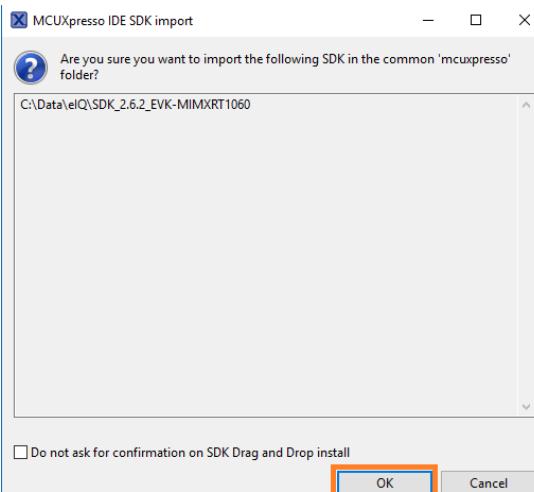
The final step is to take the Label Image example and modify it to use the retrained model with the new image.

5.1 Copy and Create Files

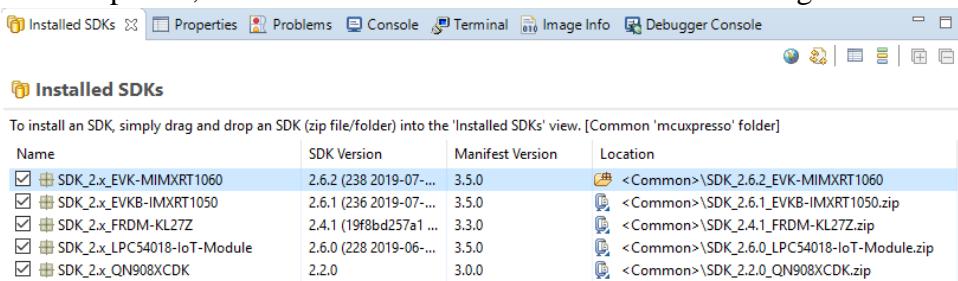
1. Open MCUXpresso IDE and select a workspace location in an empty directory.
2. Because of the large size of the model file, the indexer settings need to be changed by going to **Window->Preferences** from the menu bar. In the dialog box that comes up, go to the **C/C++->Indexer** category and uncheck **Enable Indexer**. Then click on **Apply and Close**.



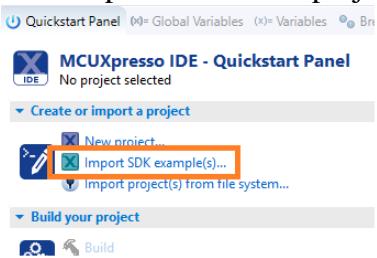
3. Drag-and-drop the unzipped SDK folder into the Installed SDKs window. It should have updated files as described in the first section. You will get the following pop-up, so hit OK.



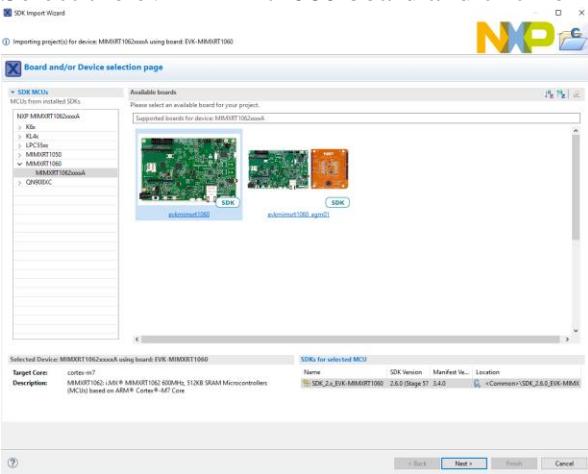
4. Once imported, the Installed SDK window will look something like this



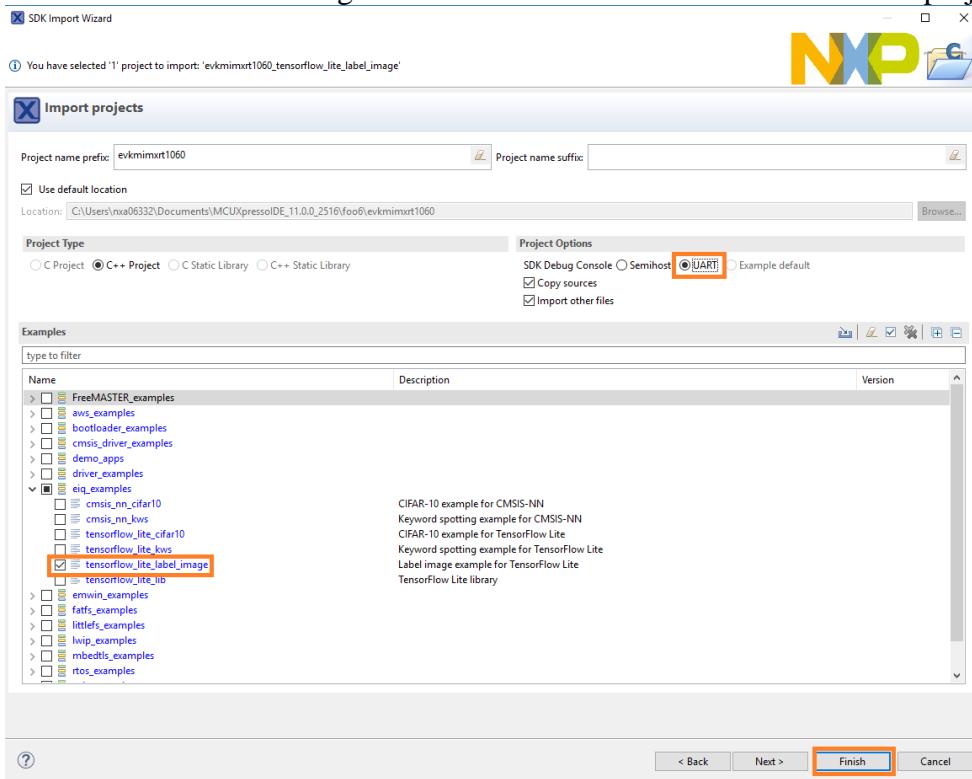
5. Next import the desired project. In the Quickstart Panel, select **Import SDK examples(s)...**



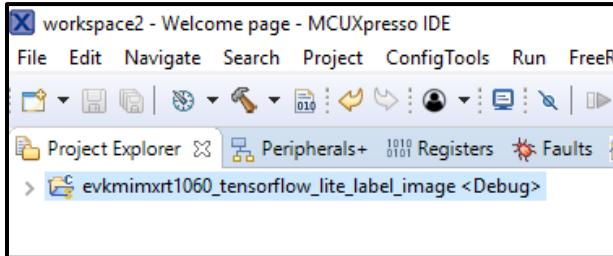
6. Select the evkmimxrt1060 board and click on Next



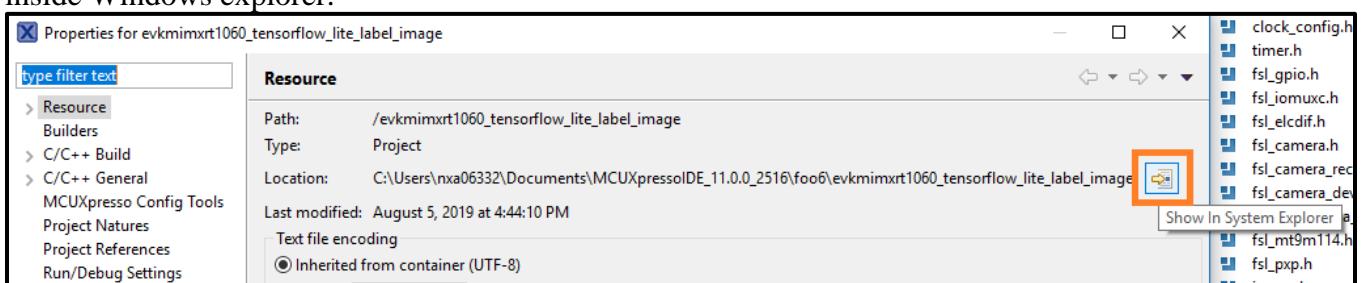
7. Then expand the **eiq_examples** folder and select **tensorflow_lite_label_image**. Also select **UART** for the SDK Debug Console. Then click on Finish to select that project.



8. It will look like the following when imported into the Project Explorer window:



8. Now we need to import the retrained model file that was generated in the last section into this project.
 9. 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:



10. Go to the “source” directory inside the **evkmimxrt1060_tensorflow_lite_label_image** folder that you just opened. It should be something like:
C:\Users\nxp_training\Documents\MCUXpressoIDE_11.0.0_2516\workspace\evkmimxrt1060_tensorflow_lite_label_image\source

11. Inside that **source** directory, copy the **retrained_graph.h** file generated in the previous section.
12. In that same directory, create a new header file named **flower_labels.h** and put in the following text, which will define the labels used to classify the flower images. This new file will be used to provide the classification labels instead of the labels.h file that was used by the default example. The file should look exactly like the following:

```
#include <string>

const std::string labels[] = {
    "daisy",
    "dandelion",
    "roses",
    "sunflowers",
    "tulips",
};
```

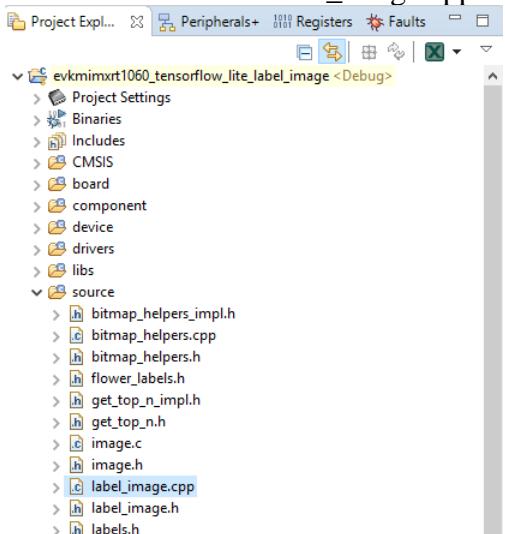
13. Directory should look like the following when finished:

Name	Date modified	Type	Size
bitmap_helpers.cpp	8/5/2019 4:43 PM	CPP File	4 KB
bitmap_helpers.h	8/5/2019 4:43 PM	H File	2 KB
bitmap_helpers_impl.h	8/5/2019 4:43 PM	H File	4 KB
flower_labels.h	7/29/2019 3:06 PM	H File	1 KB
get_top_n.h	8/5/2019 4:43 PM	H File	2 KB
get_top_n_impl.h	8/5/2019 4:43 PM	H File	3 KB
image.c	8/5/2019 4:43 PM	C File	8 KB
image.h	8/5/2019 4:43 PM	H File	4 KB
label_image.cpp	8/5/2019 4:43 PM	CPP File	22 KB
label_image.h	8/5/2019 4:43 PM	H File	2 KB
labels.h	8/5/2019 4:43 PM	H File	18 KB
mobilenet_v1_0.25_128_quant_model.h	8/5/2019 4:43 PM	H File	3,034 KB
retrained_graph.h	7/29/2019 10:59 AM	H File	11,622 KB
semihost_fault.c	8/5/2019 4:43 PM	C File	6 KB
stopwatch_image.h	8/5/2019 4:43 PM	H File	2,684 KB
timer.c	8/5/2019 4:43 PM	C File	3 KB
timer.h	8/5/2019 4:43 PM	H File	1 KB

5.2 Modify Source Code

Now edit the source files to include these new files

14. Double click on the **label_image.cpp** file under the “source” folder in the Project View to open it.



15. On line 43, comment out original #include for the original model, and then on line 47 comment out the #include for the original label file. Then on line 50, add new #includes to bring in the new model and label files that were copied in. It should look like the following when finished:

```

43 //#include "mobilenet_v1_0.25_128_quant_model.h"
44
45 #include "get_top_n.h"
46 #include "bitmap_helpers.h"
47 //#include "labels.h"
48 #include "label_image.h"
49
50 #include "retrained_graph.h"
51 #include "flower_labels.h"
52

```

16. At around line 458, comment out the API call to load the default model, and replace it with the new model name and model length from the header file. It may be a slightly different name than the one listed below. It should match the array name and length in retrained_graph.h:

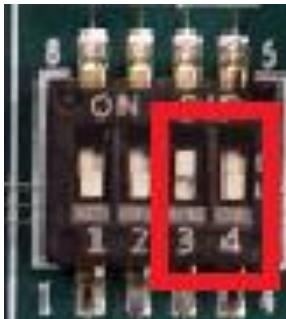
```

456
457 void InferenceInit(std::unique_ptr<tflite::FlatBufferModel> &model, std::unique_ptr<tflite::Interpreter> &interpreter, TfLiteTensor
458     model = tflite::FlatBufferModel::BuildFromBuffer(tf_files_retrained_graph_tflite, tf_files_retrained_graph_tflite_len);
459

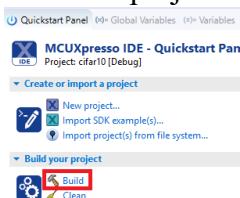
```

5.3 Run Example

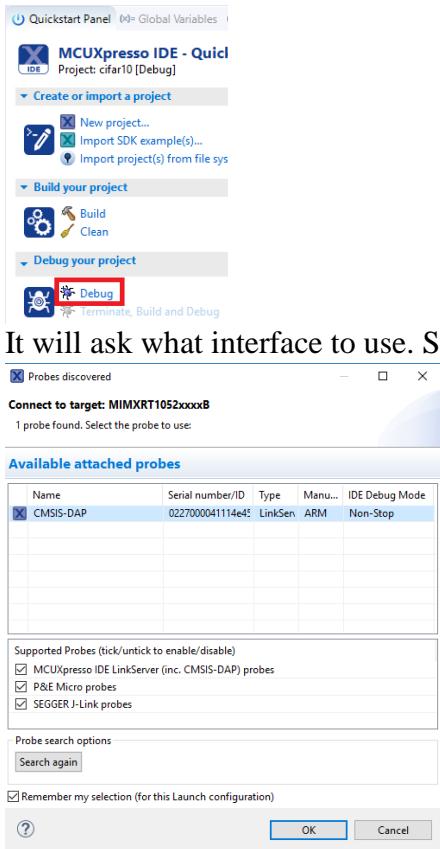
17. If using the i.MXRT1064 board, make the changes outlined in the following document:
<https://community.nxp.com/docs/DOC-344225>. If using the i.MXRT1050 or i.MXRT1060 boards, this step is not needed.
18. On the i.MXRT1060 board, change SW7 to configure the board to boot from the flash. SW7 should be OFF-OFF-ON-OFF:



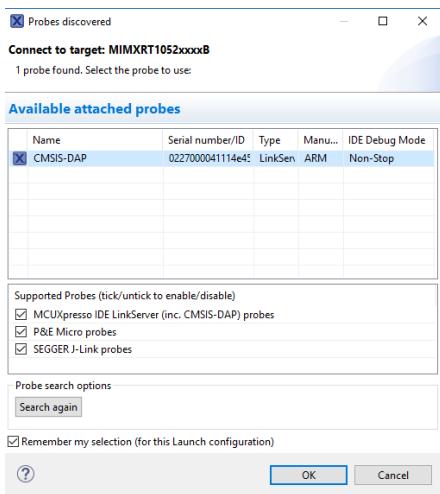
19. Plug the micro-B USB cable into the board at J41.
20. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
21. Build the project by clicking on “Build” in the Quickstart Panel.



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



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



24. The debugger will download the firmware and open up the debug view. Click on the Resume button to start running.

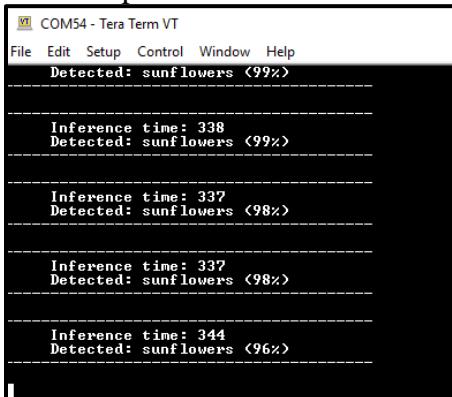


25. On the LCD screen, you should see what the camera is pointing at.

26. You will likely need to adjust the camera focus. This can be done by using some needle nose pliers or your fingers to twist the front of the camera lens clockwise until it is in focus. It should take about two revolutions. It should turn freely, and should not take much strength to turn the lens.



27. 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.



The screenshot shows a terminal window titled "COM54 - Tera Term VT". The menu bar includes "File", "Edit", "Setup", "Control", "Window", and "Help". The main text area displays five lines of inference results, each consisting of an inference time and a detection confidence for "sunflowers".

```
File Edit Setup Control Window Help
Detected: sunflowers <99%>
-----
Inference time: 338
Detected: sunflowers <99%>
-----
Inference time: 337
Detected: sunflowers <98%>
-----
Inference time: 337
Detected: sunflowers <98%>
-----
Inference time: 344
Detected: sunflowers <96%>
```

28. 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.
29. 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 the **tflite_convert** utility convert a TensorFlow model into a format that can be imported and ran on an embedded system using the eIQ software platform.

The particular model was used to classify flower images. However, the model can also be trained on new types of images by retraining it. 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 with this same process as well. By enabling machine learning in embedded systems, there's a wide world of opportunity for new applications.