eIQ: Transfer Learning – Without Camera

Lab Hand Out - Revision 5

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 inference engine that is part of NXPs eIQ package, the model can be ran on an i.MX RT embedded device.

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.

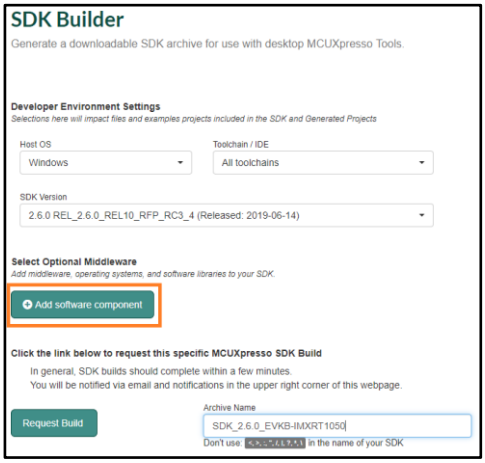
This lab is written for the RT1060-EVK. It can also be used with the RT1050-EVKB and RT1064-EVK with minor modifications. Also note that the RT1060-EVK and RT1064-EVK come with a camera sensor. The RT1050-EVKB does not come with a camera sensor. In all cases the LCD must be purchased separately.

2 Software and Hardware Installation

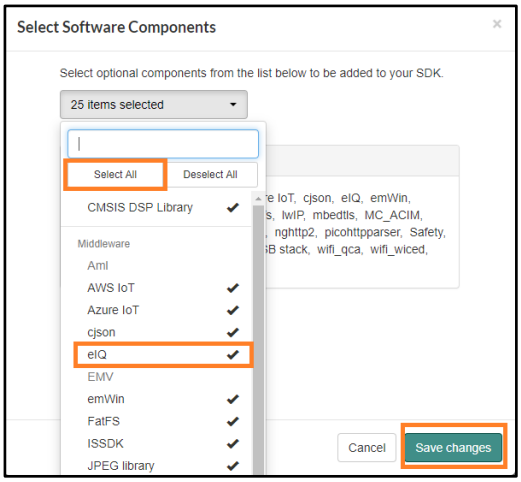
This section will cover the steps needed to install the eIQ software and TensorFlow on your computer.

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”



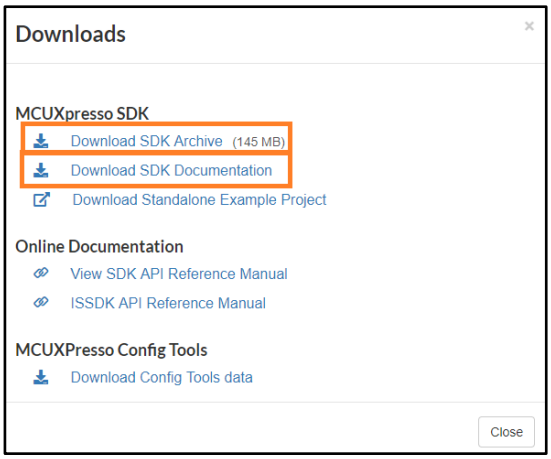
c) Select **eIQ** and **CMSIS DSP Library** to just get the eIQ software. Alternatively, to get all the software packages for the device, click on “Select All” and verify the eIQ software option is now checked. Then click on “Save Changes”



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)



2.2 TensorFlow Installation

4. Download and install Python 3.7. ****The 64-bit edition is required****: https://www.python.org/downloads/
5. Verify that the python command corresponds to Python 3.x. You may need to use “python3” for all the commands instead of “python”:

python -V

6. Update the python installer tools:
   python -m pip install -U pip
   python -m pip install -U setuptools

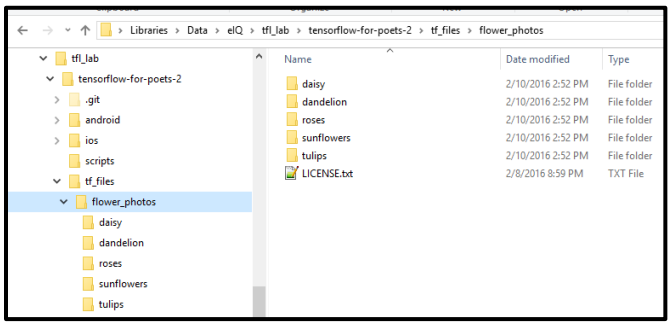
7. Install the Tensorflow libraries and support for python. Need at least version 1.13.1 if you already had TensorFlow installed previously.
   python -m pip install tensorflow
8. Install other useful python packages. Not all of these will be used for this lab but will be useful for other eIQ demos and scripts.
   python -m pip install 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

9. 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.
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 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.

12. Install Git: https://git-scm.com/downloads
13. 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
14. 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
15. 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**
16. 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:



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_traini
ng_steps=500 --model_dir=tf_files/models --summaries_dir=tf_files/training_summaries/mobilenet_0.25_128 --output_graph=t
f_files/retrained_graph.pb --output_labels=tf_files/retrained_labels.txt --architecture=mobilenet_0.25_128 --image_dir=t
f_files/flower_photos
```

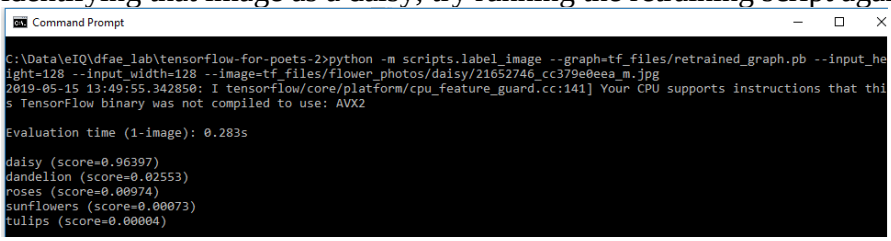
3. This will take several minutes to run. While waiting, here's an explanation for the arguments in the command you just ran:
   - ***--bottleneck_dir=tf_files/bottlenecks***
     Directory to store cached data information
   - ***--how_many_training_steps=500***

# 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.



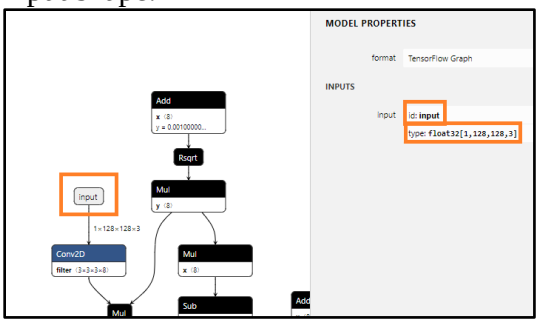
4 Convert Model and Data

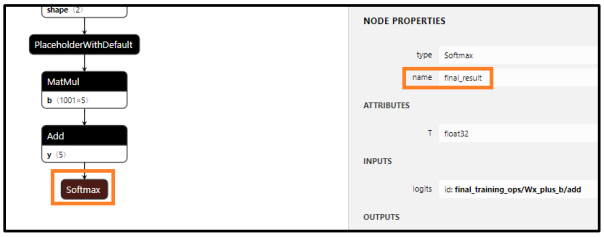
Now that the retrained model is running on your laptop, the next step is to use a TensorFlow utility named **tflite_convert** to convert that model into a file that can be loaded onto an embedded device like the i.MXRT1060.

4.1 Convert TensorFlow model

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

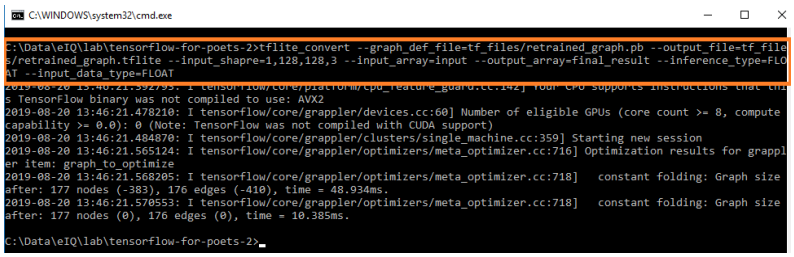
1. The first and last layer names need to be read from the TensorFlow .pb model file, which will be used later in the conversion process. This can be done with a neural network visualization tool called Netron.
    a. Use **Netron** by using the following command on a .pb file:
    netron tf_files\retrained_graph.pb
    b. 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

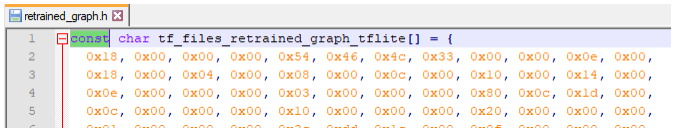
2. Use **tflite_convert** to transform the model file into a .tflite file. There are options available to quantize and optimize the model during this step, but they seem to significantly decrease the accuracy of the converted model. The following options keep the accuracy about the same as the full model. 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



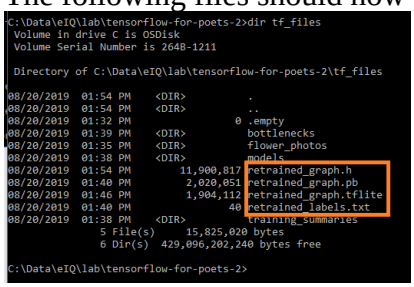
Here is what each of those arguments determine:

- ***--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.



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



4.2 Image Data Conversion – No Camera Only

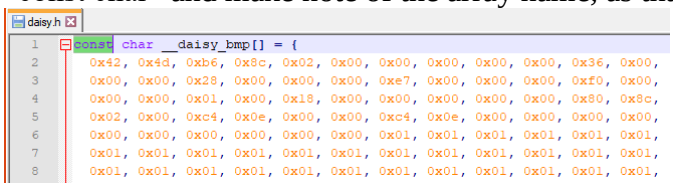
Now that the model has been converted, an image also needs to be converted into a C array. The Label Image example in eIQ takes in 24-bit BMP image files, so will convert one of the images in the dataset to a BMP file and then convert that into a C array.

8. Convert one of the JPEG flower image files into a 24-bit BMP file. Preferably pick an image that was labeled already using the label_image python script so that result can be compared to the result on the RT1060. In this case we'll pick the file **tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg.**

   Save the new .bmp file as daisy.bmp

   - In Linux use:
     convert tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg -type truecolor tf_files/daisy.bmp
   - In Windows use a paint program like MS Paint.

9. After converting the .jpg file to a .BMP file, use the xxd program to convert it to a C array. Do not run this command in Powershell as it will cause compilation issues later. Run it in a standard command prompt:
   xxd -i daisy.bmp > daisy.h
10. Open the generated header file and, if necessary, change the array type from "unsigned char" to "const char" and make note of the array name, as that name will be used in the next section:



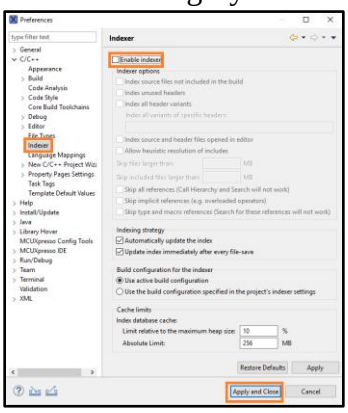
5 Run Demo

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

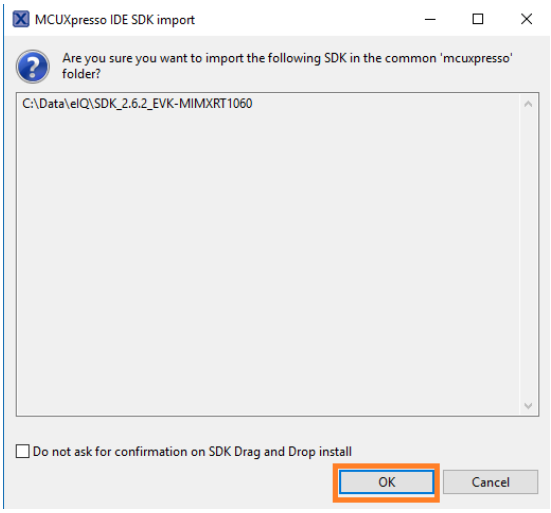
5.1 Copy and Create Files

1. Open MCUXpresso IDE and select a workspace location in an empty directory.

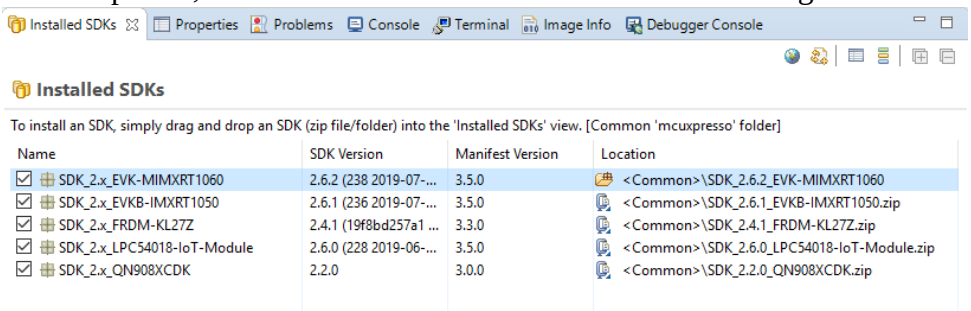
2. Because of the large size of the model file, the indexer settings need to be changed by going to **Window->Preferences** from the menu bar. In the dialog box that comes up, go to the **C/C++->Indexer** category and uncheck **Enable Indexer**. Then click on **Apply and Close**.



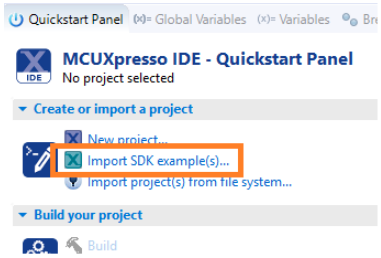
3. Drag-and-drop the unzipped SDK folder into the Installed SDKs window. It should have updated files as described in the first section. You will get the following pop-up, so hit OK.



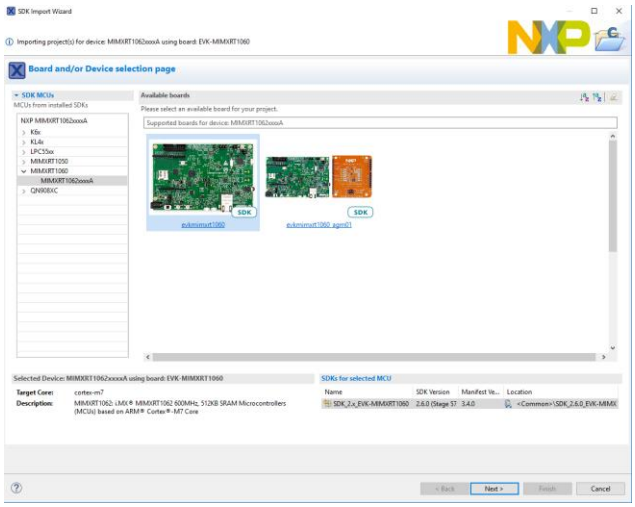
4. Once imported, the Installed SDK window will look something like this



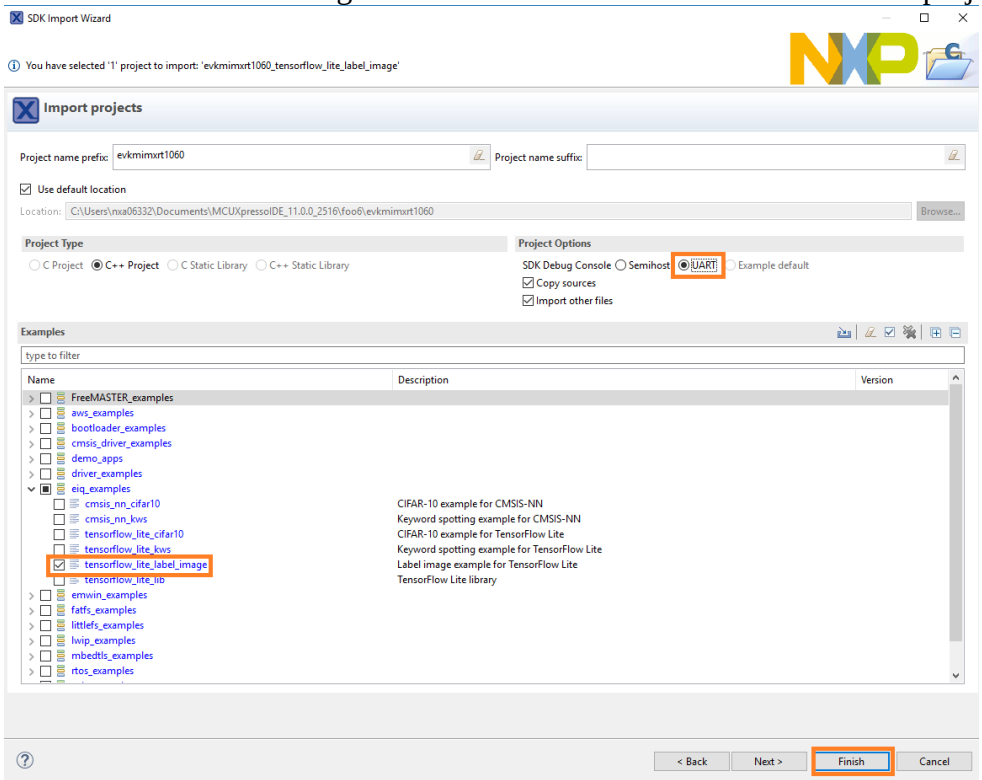
5. Next import the desired project. In the Quickstart Panel, select **Import SDK examples(s)…**



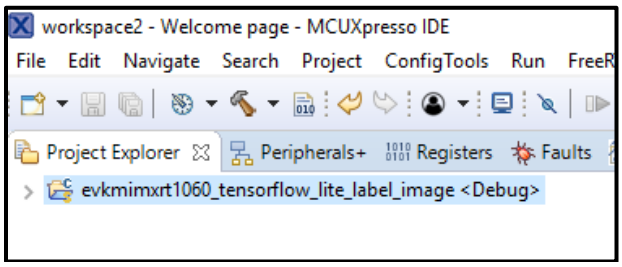
6. Select the evkmimxrt1060 board and click on Next



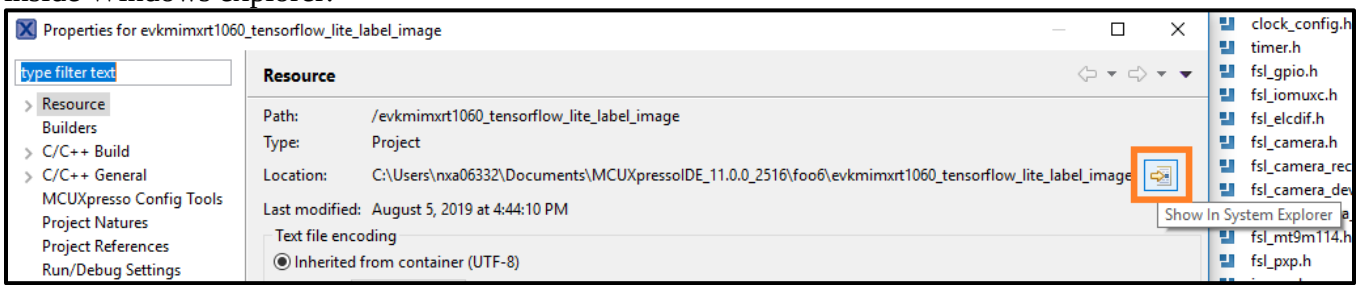
7. Then expand the **eiq_examples** folder and select **tensorflow_lite_label_image**. Also select **UART** for the SDK Debug Console. Then click on Finish to select that project.



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



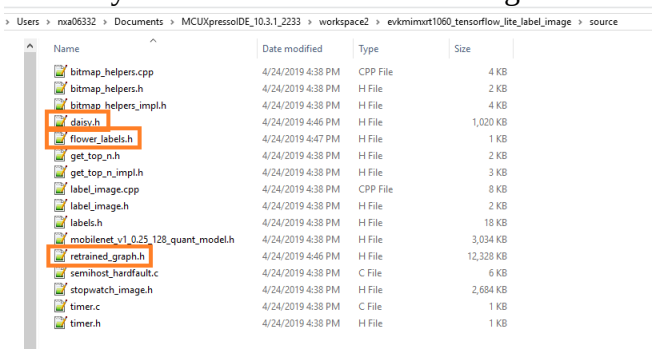
11. Now we need to import the retrained model file that was generated in the last section into this project.
12. Find the directory location that this example was copied to by right clicking on the project name, and select Properties. In the dialog box that comes up, click on the icon to open up that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



13. 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**
14. Inside that **source** directory, copy the **retrained_graph.h** file generated in the previous section.
15. If not using the camera, also copy the **daisy.h** file generated from Section 4.2
16. 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:

```
std::string labels_txt = R"(daisy
dandelion
roses
sunflowers
tulips
)";
```

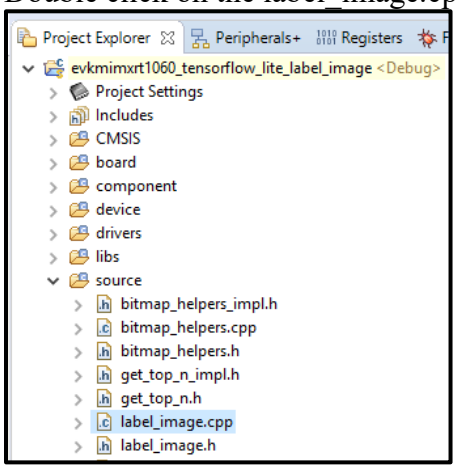
17. Directory should look like the following when finished:



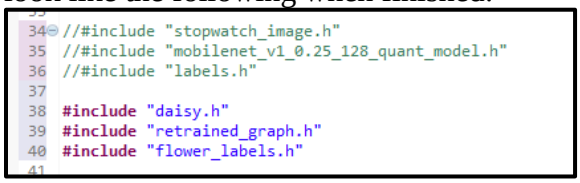
5.2 Modify Source Code

Now edit the source files to include these new files

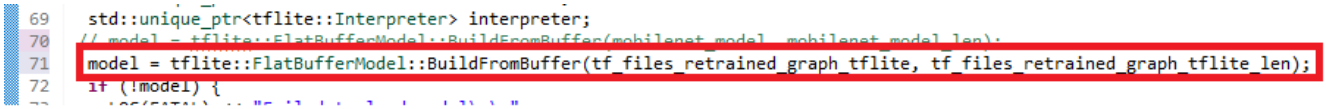
18. Double click on the label_image.cpp file under the "source" folder in the Project View to open it.



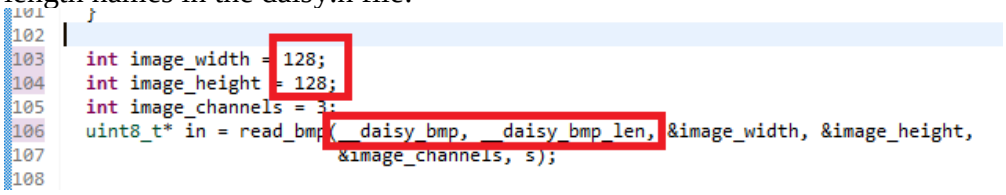
19. Starting on line 34, comment out original #includes for the image, model, and label files. Then add new #includes to bring in the new image, model, and label files that were copied in. It should look like the following when finished:



20. At around line 70, 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:



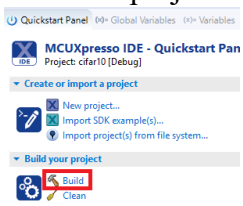
21. At around line 103, change the image height and width to 128 if they are not already, and then update the image buffer name and image length with the new image. The names may be a slightly different name than the one listed below and should match the array name and array length names in the daisy.h file:



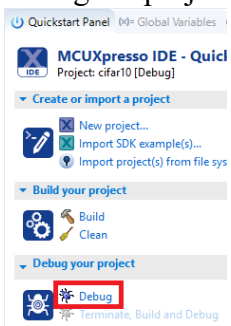
5.3 Run Example

22. 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.
23. On the i.MXRT1060 board, change SW7 to configure the board to boot from the flash. SW7 should be OFF-OFF-ON-OFF:

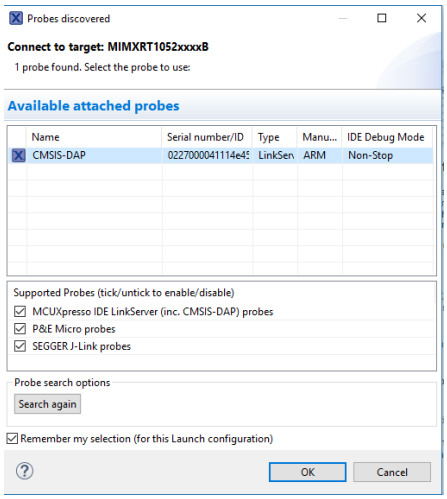
24. Plug the micro-B USB cable into the board at J41.
25. Open TeraTerm or other terminal program, and connect to the COM port that the board enumerated as. Use 115200 baud, 1 stop bit, no parity.
26. Build the project by clicking on “Build” in the Quickstart Panel.



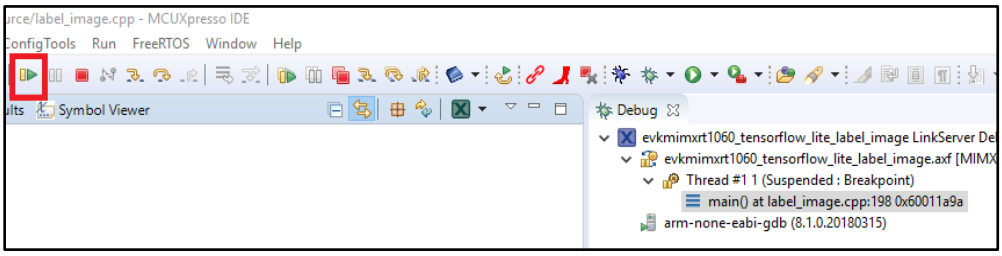
27. Debug the project by clicking on “Debug” in the Quickstart Panel.



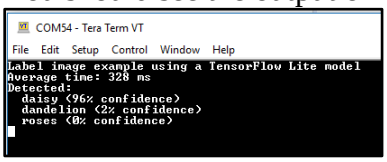
28. It will ask what interface to use. Select CMSIS-DAP.



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



30. You should see the output on the console:



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 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.