AN12781

Caffe Model Development on MNIST Dataset with CMSIS-NN Library

Rev. 0 — 17 April 2020 Application Note

Contents

1 Introduction

CMSIS-NN is a collection of optimized Neural Network (NN) functions for Arm Cortex-M core microcontrollers to enable neural networks and machine learning. The MCUXpresso SDK includes a software package with a pre-integrated eIQ CMSIS-NN library based on CMSIS-NN 1.0.0. This document describes the process to train a Caffe model on MNIST dataset for digit classification. The trained Caffe model is converted to a source file that can run on i.MX RT platforms.

2 Deep neural network model development

There are several frameworks available for developing and training the deep neural network model, such as TensorFlow, Caffe, and Keras. This deep neural network has been designed with Caffe framework so that trained model can be converted to a source file by a python script provided by Arm. This python script from Arm for trained Caffe model can be converted to CMSIS-NN function for execution on the edge.

This application note is an extension to the Handwritten Digit Recognition Using TensorFlow Lite on RT1060 application note. This document describes step-by-step process of deep neural network model training using the Caffe framework on MNIST dataset, which contains 60,000 handwritten grayscale images. The document also describes how to export the trained model on i.MX RT board to recognize the handwritten digits and how to replace the TF-Lite MNIST lock application with CMSIS-NN MNIST application.

2.1 Training Caffe model on MNIST dataset

Before heading towards model training, you need to set up the system for Caffe framework. There are different ways to set up the Caffe framework; this document uses a Docker image that already contains the Caffe framework. You need to install some additional libraries required for this application note. Make sure that Windows system is installed with MCUXpresso IDE (latest version) and a serial terminal emulator (TeraTerm). This application note assumes that the user has basic knowledge of Linux commands.

Following are the steps of the Docker setup on Windows 10:

1. Install Docker Desktop on Windows using the below link.

   https://docs.docker.com/docker-for-windows/install

   Novice users can follow the link below to get basics of Docker.

   https://stackify.com/docker-tutorial/

2. Log in to the Caffe container by running the command below from Windows command prompt.

```
docker run -ti bvlc/caffe:cpu bash
```

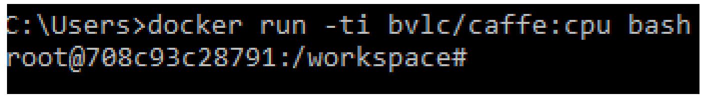


Figure 1. Caffe bash shell

Now you are in the Docker container (Caffe bash shell, and container ID is '708c93c28791').

> **NOTE**
> When you are in Docker container, it does not support command for execution on Windows command prompt. For executing commands on Windows command prompt, you must open a separate CMD terminal.

Below is the package required by the Python script in Docker container (Caffe bash shell).

```
apt-get update
pip install -U scikit-image
pip install opencv-python
pip install xlwt
pip install xlrd
apt-get install python-tk
apt-get install wget
apt-get install gzip
```

3. Install nano text editor (or any other preferred text editor) for text editing.

```
apt-get install nano
```

4. It is recommended to commit Docker container as a backup. To commit the current Docker container, open a separate Windows command prompt, and run the below command.

```
docker commit <container Id> imageNameforSave
```

5. Get container ID **'708c93c28791'** from below figure.

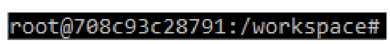


Figure 2. Docker container ID

The Docker basic commands are available with the release package in docker_readme.txt.

> **NOTE**
> Return to container from the state in which it was shut down using the command below:

```
docker start --interactive <container ID>
```

The steps below describe the Caffe model training and image classification.

1. Data preparation: Download MNIST dataset from the MNIST website using the following command in Docker container (Caffe bash shell).

```
cd $CAFFE_ROOT
./data/mnist/get_mnist.sh
./examples/mnist/create_mnist.sh
```

> **NOTE**
> All commands are executed from Caffe root directory [/opt/caffe/] in Caffe bash shell unless explicitly mentioned to execute on Windows command prompt.

2. Data normalization: Subtract the mean image from each input image to ensure every feature pixel has zero mean. It is required by Caffe model. For this, create the mean image file and format in the **mnist_mean.binaryproto** file using the command below.

```
build/tools/compute_image_mean -backend=lmdb examples/mnist/mnist_train_lmdb examples/mnist/
mnist_mean.binaryproto
```

```
root@708c93c28791:/opt/caffe# build/tools/compute_image_mean -backend=lmdb examples/mnist/mnist_train_
lmdb examples/mnist/mnist_mean.binaryproto
I1217 07:29:23.736666   989 db_lmdb.cpp:35] Opened lmdb examples/mnist/mnist_train_lmdb
I1217 07:29:23.736981   989 compute_image_mean.cpp:70] Starting iteration
I1217 07:29:23.745985   989 compute_image_mean.cpp:95] Processed 10000 files.
I1217 07:29:23.754249   989 compute_image_mean.cpp:95] Processed 20000 files.
I1217 07:29:23.763078   989 compute_image_mean.cpp:95] Processed 30000 files.
I1217 07:29:23.771242   989 compute_image_mean.cpp:95] Processed 40000 files.
I1217 07:29:23.779996   989 compute_image_mean.cpp:95] Processed 50000 files.
I1217 07:29:23.788414   989 compute_image_mean.cpp:95] Processed 60000 files.
I1217 07:29:23.788492   989 compute_image_mean.cpp:108] Write to examples/mnist/mnist_mean.binaryproto

I1217 07:29:23.788655   989 compute_image_mean.cpp:114] Number of channels: 1
I1217 07:29:23.788681   989 compute_image_mean.cpp:119] mean_value channel [0]: 33.3184
```

Figure 3. Command for creating mean image file

The mean image file is created in the folder with the name, **/opt/caffe/examples/mnist/mnist_mean.binaryproto.**

This completes data pre-processing required during training and testing phase.

3. Next, Caffe model definition file, '**lenet_train_test.prototxt**', is required, which specifies Convolutional Neural Network (CNN) architecture for MNIST handwritten digit classification.

   The Caffe model definition file is available in the **/opt/caffe/examples/mnist/** folder. Use the AlexNet CNN architecture for model development and conversion to source file. Use the command below to copy model definition file, **alexnet_train_test.prototxt**, available with the release package in the **MNIST_model** folder, from Windows desktop to Docker container.

```
docker cp d:\path\to\folder\alexnet_train_test.prototxt [containerID]:/opt/caffe/examples/mnist
```

   This step is independent of step 3. Check all Docker containers by executing the following command on Windows command prompt. This command displays all running Docker containers. When you make a commit to create a backup, a new row with a new container appears.

```
docker ps -a
```

```
C:\Users>docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
708c93c28791        caffeimage4_9_9     "bash"              About an hour ago   Up About an hour
```

Figure 4. Docker container ID

4. Use the command below to copy file from Windows to Docker container ID 708c93c28791.

```
docker cp d:\path\to\folder\alexnet_train_test.prototxt 708c93c28791:/opt/caffe/examples/mnist
```

   The Caffe model file for training is available in the **MNIST_model** folder with release package. It needs to be copied one by one using the following steps.

5. Apart from model definition file, the Solver definition file (lenet_solver.prototxt) is required. Use the command below to copy Solver definition file **alexnet_solver.prototxt** available with release package in the folder, **MNIST_model**, from Windows desktop to Docker container.

```
docker cp d:\path\to\folder\alexnet_solver.prototxt 708c93c28791:/opt/caffe/examples/mnist
```

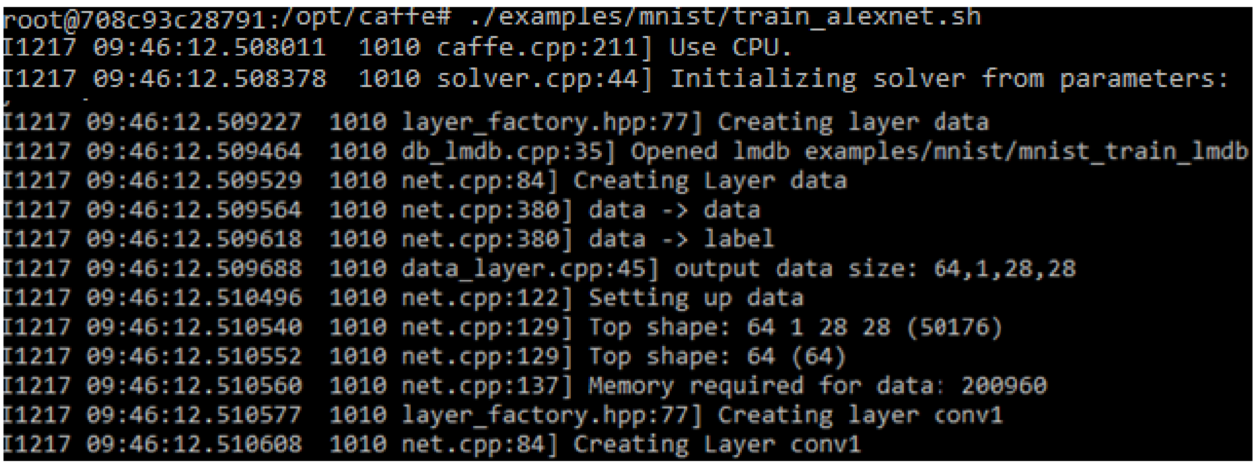
6. Training Model starts by running the **train_alexnet.sh** script available with the release package in the **MNIST_model** folder. Use the command below to copy this script from Windows desktop to Docker container.

```
docker cp d:\path\to\folder\train_alexnet.sh 708c93c28791:/opt/caffe/examples/mnist
```

7. Start training by using the command below from Caffe root directory /opt/caffe/.

```
./examples/mnist/train_alexnet.sh
```

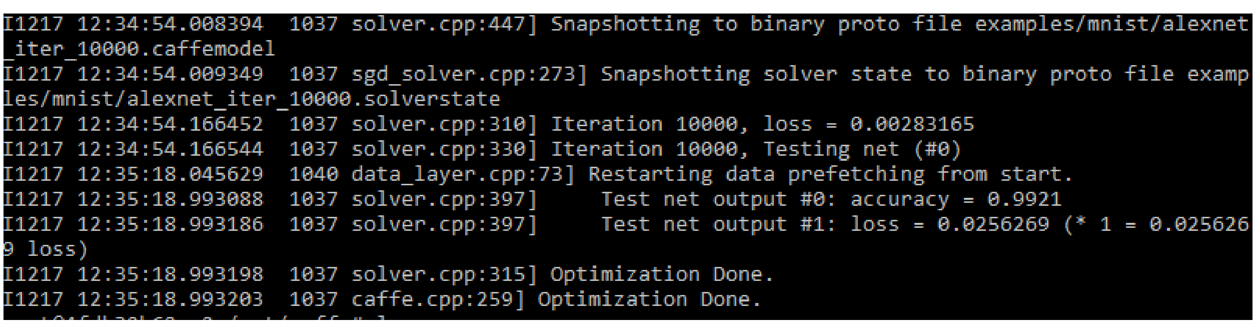
While training, the following message displays on the screen. It takes 45 to 60 minutes to complete training depending upon system configuration.



```
root@708c93c28791:/opt/caffe# ./examples/mnist/train_alexnet.sh
I1217 09:46:12.508011  1010 caffe.cpp:211] Use CPU.
I1217 09:46:12.508378  1010 solver.cpp:44] Initializing solver from parameters:
I1217 09:46:12.509227  1010 layer_factory.hpp:77] Creating layer data
I1217 09:46:12.509464  1010 db_lmdb.cpp:35] Opened lmdb examples/mnist/mnist_train_lmdb
I1217 09:46:12.509529  1010 net.cpp:84] Creating Layer data
I1217 09:46:12.509564  1010 net.cpp:380] data -> data
I1217 09:46:12.509618  1010 net.cpp:380] data -> label
I1217 09:46:12.509688  1010 data_layer.cpp:45] output data size: 64,1,28,28
I1217 09:46:12.510496  1010 net.cpp:122] Setting up data
I1217 09:46:12.510540  1010 net.cpp:129] Top shape: 64 1 28 28 (50176)
I1217 09:46:12.510552  1010 net.cpp:129] Top shape: 64 (64)
I1217 09:46:12.510560  1010 net.cpp:137] Memory required for data: 200960
I1217 09:46:12.510577  1010 layer_factory.hpp:77] Creating layer conv1
I1217 09:46:12.510608  1010 net.cpp:84] Creating Layer conv1
```

Figure 5. Caffe training message on console

After training completes, the trained Caffe model file, **alexnet_iter_10000.caffemodel**, generates in the **/opt/caffe/examples/mnist/** folder.



```
I1217 12:34:54.008394  1037 solver.cpp:447] Snapshotting to binary proto file examples/mnist/alexnet
_iter_10000.caffemodel
I1217 12:34:54.009349  1037 sgd_solver.cpp:273] Snapshotting solver state to binary proto file examp
les/mnist/alexnet_iter_10000.solverstate
I1217 12:34:54.166452  1037 solver.cpp:310] Iteration 10000, loss = 0.00283165
I1217 12:34:54.166544  1037 solver.cpp:330] Iteration 10000, Testing net (#0)
I1217 12:35:18.045629  1040 data_layer.cpp:73] Restarting data prefetching from start.
I1217 12:35:18.993088  1037 solver.cpp:397]     Test net output #0: accuracy = 0.9921
I1217 12:35:18.993186  1037 solver.cpp:397]     Test net output #1: loss = 0.0256269 (* 1 = 0.025626
9 loss)
I1217 12:35:18.993198  1037 solver.cpp:315] Optimization Done.
I1217 12:35:18.993203  1037 caffe.cpp:259] Optimization Done.
```

Figure 6. Caffe model training complete message

8. Image Classification: For image classification, the **classify_image.py** script needs the following files. To classify an unknown image, the trained model file is stored as **caffemodel**, which has trained model weights. So you need to load these files and preprocess the input images. The output digit image is then predicted by the **classify_image.py** script.

   - Mean file, **mnist_mean.binaryproto**, which is generated in the previous step.
   - Deploy file, **alexnet_deploy.prototxt**, in which input test image information and CNN architecture is mentioned.

- Trained Cafffe model file **alexnet_iter_10000.caffemodel.**
- Input test image, 28x28 grayscale image, from MNIST dataset, available in the **image** folder in the release package.

9. Copy the test image for digit two, **two(4).png**, from Windows using the command below:

```
docker cp d:\path\to\folder\two(4).png 708c93c28791:/opt/caffe/
```

If user wants to test the image for any other digit, it must be in MNIST dataset format. Download the MNIST dataset handwritten digit image using the command below:

```
git clone https://github.com/myleott/mnist_png.git
```

NOTE

The downloaded MNIST dataset is in zip format; unzip it for using it.

10. Perform the image classification by running the **classify_image.py** python script. The files, **classify_image.py** and **alexnet_deploy.prototxt**, are available in the release package. Copy them from Windows to Docker container using the commands below:

```
docker cp d:\path\to\folder\classify_image.py 708c93c28791:/opt/caffe/
docker cp d:\path\to\folder\alexnet_deploy.prototxt 708c93c28791:/opt/caffe/examples/mnist
```

11. Execute the **classify_image.py** script, and for image classification, run the command below from Caffe root directory, **/opt/caffe/.**

```
python classify_image.py
```

The following message displays on the screen after executing the **classify_image.py** script for test image, **two(4).png**

.

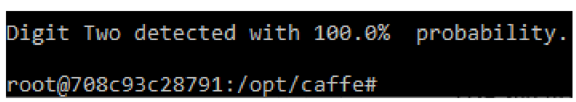


Figure 7. Image classification output

3 Caffe model conversion

3.1 Quantization

The Neural Network (NN) operation is trained using 32-bit floating-point data. Operating 32-bit floating point requires much memory and high-processing power, which is a constraint for an embedded device. Quantization converts the Caffe model weights and activations from 32-bit floating point to an 8-bit and fixed-point format, reducing the size of the model without sacrificing the performance.

The output of this script is a serialized Python pickle(.pkl) file, which includes the network's model, quantized weights and activations, and the quantization format of each layer. You can download the script from the following link: https://github.com/ARM-software/ML-examples.git.

It is recommended to use the **nn_quantizer.py** python script available with this document in the /Scripts folder. This is because quantization script available from Arm supports only conversion of trained Caffe model for cifar10 and requires some changes for supporting the model trained with the MNIST dataset.

Copy script from Windows to the docker container using the command below:

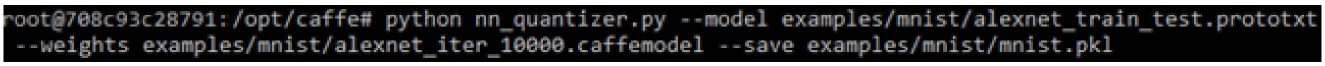
```
docker cp d:\path\to\folder\nn_quantizer.py 708c93c28791:/opt/caffe/
```

NOTE

Trained Caffe model files are available in the **trained_model** folder with this release package.

Start quantization using the command below from Caffe root directory:

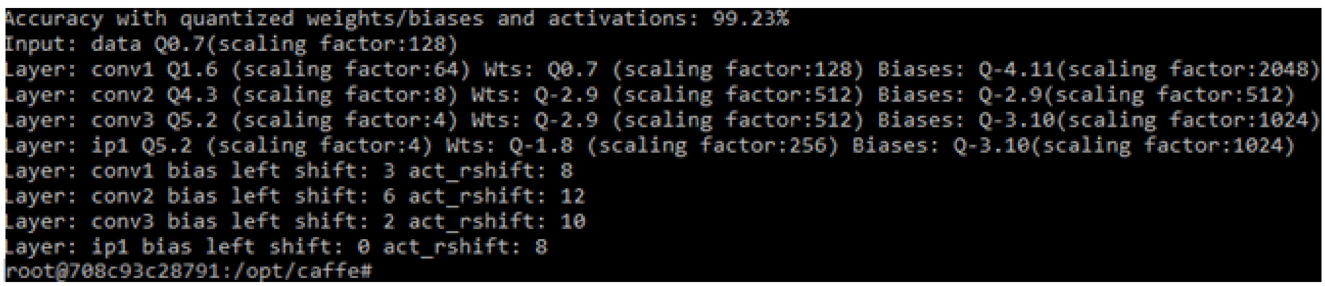
```
python nn_quantizer.py --model examples/mnist/alexnet_train_test.prototxt --weights examples/mnist/
alexnet_iter_10000.caffemodel --save examples/mnist/mnist.pkl
```



```
root@708c93c28791:/opt/caffe# python nn_quantizer.py --model examples/mnist/alexnet_train_test.prototxt
 --weights examples/mnist/alexnet_iter_10000.caffemodel --save examples/mnist/mnist.pkl
```

Figure 8. Quantization script running command

The following message displays after quantization completes. It takes 10 - 20 minutes to complete quantization depending upon system configuration. The **mnist.pkl** file generates in the **/opt/caffe/examples/mnist** folder.



```
Accuracy with quantized weights/biases and activations: 99.23%
Input: data Q0.7(scaling factor:128)
Layer: conv1 Q1.6 (scaling factor:64) Wts: Q0.7 (scaling factor:128) Biases: Q-4.11(scaling factor:2048)
Layer: conv2 Q4.3 (scaling factor:8) Wts: Q-2.9 (scaling factor:512) Biases: Q-2.9(scaling factor:512)
Layer: conv3 Q5.2 (scaling factor:4) Wts: Q-2.9 (scaling factor:512) Biases: Q-3.10(scaling factor:1024)
Layer: ip1 Q5.2 (scaling factor:4) Wts: Q-1.8 (scaling factor:256) Biases: Q-3.10(scaling factor:1024)
Layer: conv1 bias left shift: 3 act_rshift: 8
Layer: conv2 bias left shift: 6 act_rshift: 12
Layer: conv3 bias left shift: 2 act_rshift: 10
Layer: ip1 bias left shift: 0 act_rshift: 8
root@708c93c28791:/opt/caffe#
```

Figure 9. Message after quantization

3.2 Converting the quantized model into source file

The Quantized model file, **mnist.pkl**, is used to generate source file. The file generates **weights.h** and **parameter.h** consisting of quantization ranges. You must include the **nn.cpp** and **nn.h** files in the application to run the Neural Network (NN) on the EVK i.MX RT board. The **mnist.pkl** file is used as a parameter in the **code_gen.py** script which you can run to generate source file.

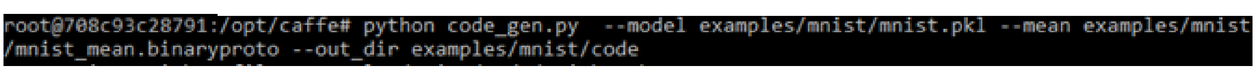
Follow the steps below to convert the quantized model into source file:

1. Copy the **code_gen.py** script from Windows to Docker container using the command below:

```
docker cp d:\path\to\folder\ code_gen.py 708c93c28791:/opt/caffe/
```

2. Execute the command below to run script for generating source file:

```
python code_gen.py --model examples/mnist/mnist.pkl --mean examples/mnist/mnist_mean.binaryproto
--out_dir examples/mnist/code
```



```
root@708c93c28791:/opt/caffe# python code_gen.py  --model examples/mnist/mnist.pkl --mean examples/mnist
/mnist_mean.binaryproto --out_dir examples/mnist/code
```

Figure 10. Command for converting Quantized model into source file

The following message displays after execution of **code_gen.py**.

```
Generating parameter file: examples/mnist/code/parameter.h
Generating file: examples/mnist/code/nn.h
Layer: conv1, required memory: 2000, im2col buffer size: 2000
Layer: conv2, required memory: 2000, im2col buffer size: 2000
Layer: conv3, required memory: 5000, im2col buffer size: 5000
Layer: ip1, required memory: 1600, im2col buffer size: 5000
Layer: conv1, required memory: 16464, buffer size: 16464
Layer: pool1, required memory: 27200, buffer size: 27200
Layer: conv2, required memory: 23040, buffer size: 27200
Layer: pool2, required memory: 14400, buffer size: 27200
Layer: conv3, required memory: 6080, buffer size: 27200
Layer: pool3, required memory: 4000, buffer size: 27200
Layer: ip1, required memory: 810, buffer size: 27200
root@708c93c28791:/opt/caffe#
```

Figure 11. Message after source file is generated

After execution of the **code_gen.py** script, the **weights.h**, **parameter.h**, and **nn.cpp** source files are available in the **examples/mnist/code** folder.

3. Copy the source file folder **code** to Windows from Docker container using the command below. It is required at a later stage for running model in the application.

```
docker cp 708c93c28791:/opt/caffe/examples/mnist/code d:\path\to\folder\
```

4 Adding source file in an application

It is assumed that the user is familiar with the eIQ demo application cmsis_nn_cifar10. Novice users can follow the document **Getting Started with the eIQ CMSIS-NN Library.pdf**. This user guide (with document number EIQCMSISNNGSUG) is available in SDK_2.x_EVK-MIMXRT1060 with the eIQ component.

4.1 Adding source code in eIQ cmsis-nn cifar10 application

Follow the steps below to replace the CIFAR10 dataset with the MNIST dataset (with the AlexNet CNN architecture) using existing eIQ cmsis-nn_cifar10 demo application:

1. Copy and replace **weights.h** and **parameter.h** header files in the application project with the header file available in the **code** folder. The **code** folder was copied (in previous section) from Docker container. The below figure shows the right file structure:

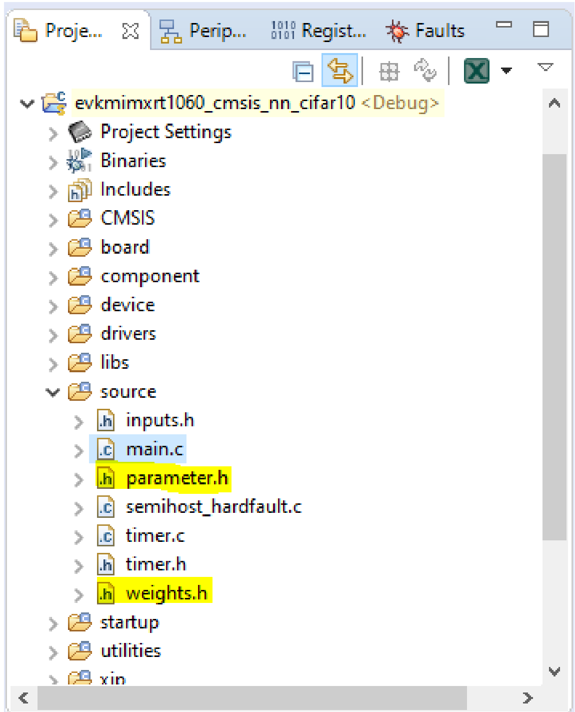


Figure 12. Replacing header file

2. Add the Neural Network function (nn_run(uint8_t*)) and its buffer in the main source file of the application project by copying the content of **nn.cpp** to the main source file, as shown in figures below:

Welcome | main.c | weights.h | parameter.h | inputs.h

```
#include "inputs.h"
#include "parameter.h"
#include "weights.h"
#include "arm_nnfunctions.h"

static const uint8_t mean[DATA_OUT_CH*DATA_OUT_DIM*DATA_OUT_DIM] = MEAN_DATA;

static const q7_t conv1_wt[CONV1_IN_CH*CONV1_KER_DIM*CONV1_KER_DIM*CONV1_OUT_CH] = CONV1_WT;
static const q7_t conv1_bias[CONV1_OUT_CH] = CONV1_BIAS;

static const q7_t conv2_wt[CONV2_IN_CH*CONV2_KER_DIM*CONV2_KER_DIM*CONV2_OUT_CH] = CONV2_WT;
static const q7_t conv2_bias[CONV2_OUT_CH] = CONV2_BIAS;

static const q7_t conv3_wt[CONV3_IN_CH*CONV3_KER_DIM*CONV3_KER_DIM*CONV3_OUT_CH] = CONV3_WT;
static const q7_t conv3_bias[CONV3_OUT_CH] = CONV3_BIAS;

static const q7_t ip1_wt[IP1_IN_DIM*IP1_OUT_DIM] = IP1_WT;
static const q7_t ip1_bias[IP1_OUT_DIM] = IP1_BIAS;

//Add input_data and output_data in top main.cpp file
const char* labels[] = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};
uint8_t image_data[DATA_OUT_CH*DATA_OUT_DIM*DATA_OUT_DIM]=DIGIT_IMG_DATA;
q7_t output_data[IP1_OUT_DIM];

q7_t col_buffer[5000];
q7_t scratch_buffer[27200];

void mean_subtract(uint8_t* image_data) {
  for(int i=0; i<DATA_OUT_CH*DATA_OUT_DIM*DATA_OUT_DIM; i++) {
    image_data[i] = (q7_t)__SSAT( ((int)(image_data[i] - mean[i]) >> DATA_RSHIFT), 8);
  }
}
```

Figure 13. Adding Neural Network function and its buffer in main source file

```
void run_nn(uint8_t* input_data) {
  q7_t* buffer1 = scratch_buffer;
  q7_t* buffer2 = buffer1 + 15680;
  mean_subtract(input_data);
  arm_convolve_HWC_q7_basic((q7_t*)input_data, CONV1_IN_DIM, CONV1_IN_CH, conv1_wt, CONV1_OUT_CH,
  arm_relu_q7(buffer1, RELU1_OUT_DIM*RELU1_OUT_DIM*RELU1_OUT_CH);
  arm_maxpool_q7_HWC(buffer1, POOL1_IN_DIM, POOL1_IN_CH, POOL1_KER_DIM, POOL1_PAD, POOL1_STRIDE, P
  arm_convolve_HWC_q7_fast(buffer2, CONV2_IN_DIM, CONV2_IN_CH, conv2_wt, CONV2_OUT_CH, CONV2_KER_D
  arm_relu_q7(buffer1, RELU2_OUT_DIM*RELU2_OUT_DIM*RELU2_OUT_CH);
  arm_maxpool_q7_HWC(buffer1, POOL2_IN_DIM, POOL2_IN_CH, POOL2_KER_DIM, POOL2_PAD, POOL2_STRIDE, P
  arm_convolve_HWC_q7_fast(buffer2, CONV3_IN_DIM, CONV3_IN_CH, conv3_wt, CONV3_OUT_CH, CONV3_KER_D
  arm_relu_q7(buffer1, RELU3_OUT_DIM*RELU3_OUT_DIM*RELU3_OUT_CH);
  arm_maxpool_q7_HWC(buffer1, POOL3_IN_DIM, POOL3_IN_CH, POOL3_KER_DIM, POOL3_PAD, POOL3_STRIDE, P
  arm_fully_connected_q7_opt(buffer2, ip1_wt, IP1_IN_DIM, IP1_OUT_DIM, IP1_BIAS_LSHIFT, IP1_OUT_RS
  arm_softmax_q7(output_data, 10, output_data);
}
```

Figure 14. Adding Neural Network function

3. Add print message statement and comment the statement under **int main(void)**, as shown below:

```
    uint32_t start_time, stop_time;
//    uint8_t image_data[32 * 32 * 3] = SHIP_IMG_DATA;
    q7_t max_value;
    uint32_t max_index;

//    PRINTF("CIFAR-10 object recognition example using convolutional neural network\r\n");
    PRINTF("MNIST dataset example using AlexNet CNN Architecture \r\n");
    start_time = get_time_in_us();
```

Figure 15. Comment statement and add print message

4. To execute image classification on the edge (i.MX RT board), generate image buffer array using the **mnist_png_to_array.py** script. It is available in the **scripts** folder in the release package. Copy it from Windows using the command below:

```
docker cp d:\path\to\folder\mnist_png_to_array.py 4fdb32b62ee2:/opt/caffe/
```

The execution of **mnist_png_to_array.py** generates image buffer array, **DIGIT_IMG_DATA**, in the **inputs.h** file. This file generates in the **/opt/caffe** folder.

5. Copy and replace image buffer array file, **inputs.h**, in your eIQ cmsis-nn_cifar10 demo application, as shown below:

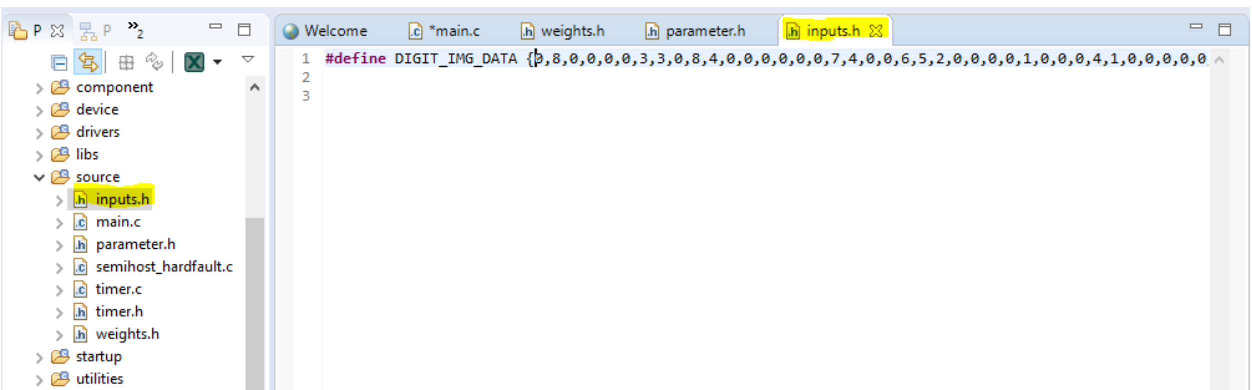


Figure 16. Image buffer array file

6. Save the changes in the file. Flash the binary in the i.MX RT board through debug mode.

The following message displays in TeraTerm serial terminal for MNIST handwritten digit prediction. The output displays Digit 2 also known as predicted class by the model for input image **two(4).png**.

```
Elapsed time: 303 ms
Predicted class: 2 (100% confidence)
```

Figure 17. Image classification output on TeraTerm serial terminal

4.2 Replacing TensorFlow-Lite with CMSIS-NN in Lock application

Following are the steps to replace MNIST TF-Lite lock source code with MNIST CMSIS-NN inference. You can find detailed information on MNIST TF-Lite lock application in AN12603 (TensorFlow Lite Model to Perform Handwritten Digit Recognition) application note.

1. Add the Neural Network library. Get the library from the cmsis_nn_cifar10 project available in the **CMSIS** folder. The library added to the tensorflow_lite_mnist_lock project should appear as displayed in figure below:

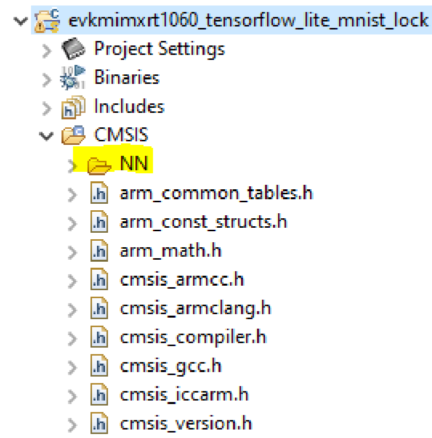


Figure 18. Neural Network library path

2. Get the **libarm_cortexM7lfdp_math.a** library from cmsis_nn_cifar10 project under **libs** folder as shown in figure below:

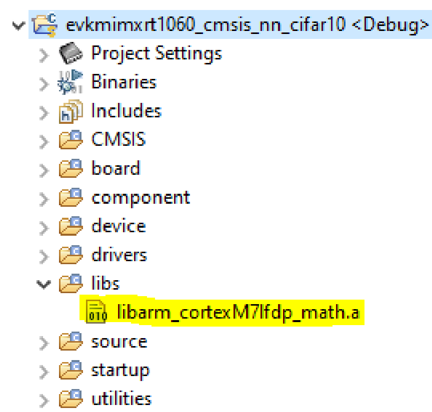


Figure 19. Static Library path

The library added to the MNIST CMSIS-NN application should appear as shown below:

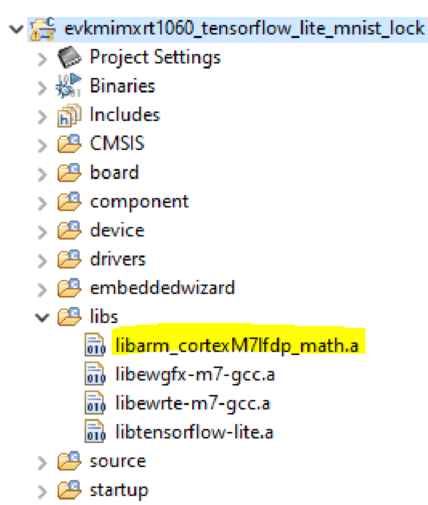


Figure 20. Static library added

3. Include NN library path in compiler/IDE. Go to Project Explorer windows and select your application project. Then right-click to navigate and select **properties**.
4. In the dialog box that appears, select **Settings** from the **C/C++ Build** drop-down menu on left side.
5. Select **Includes** from the **MCU C Compiler** drop-down menu on right side, and add the directory path for NN/Include folder, as shown below:

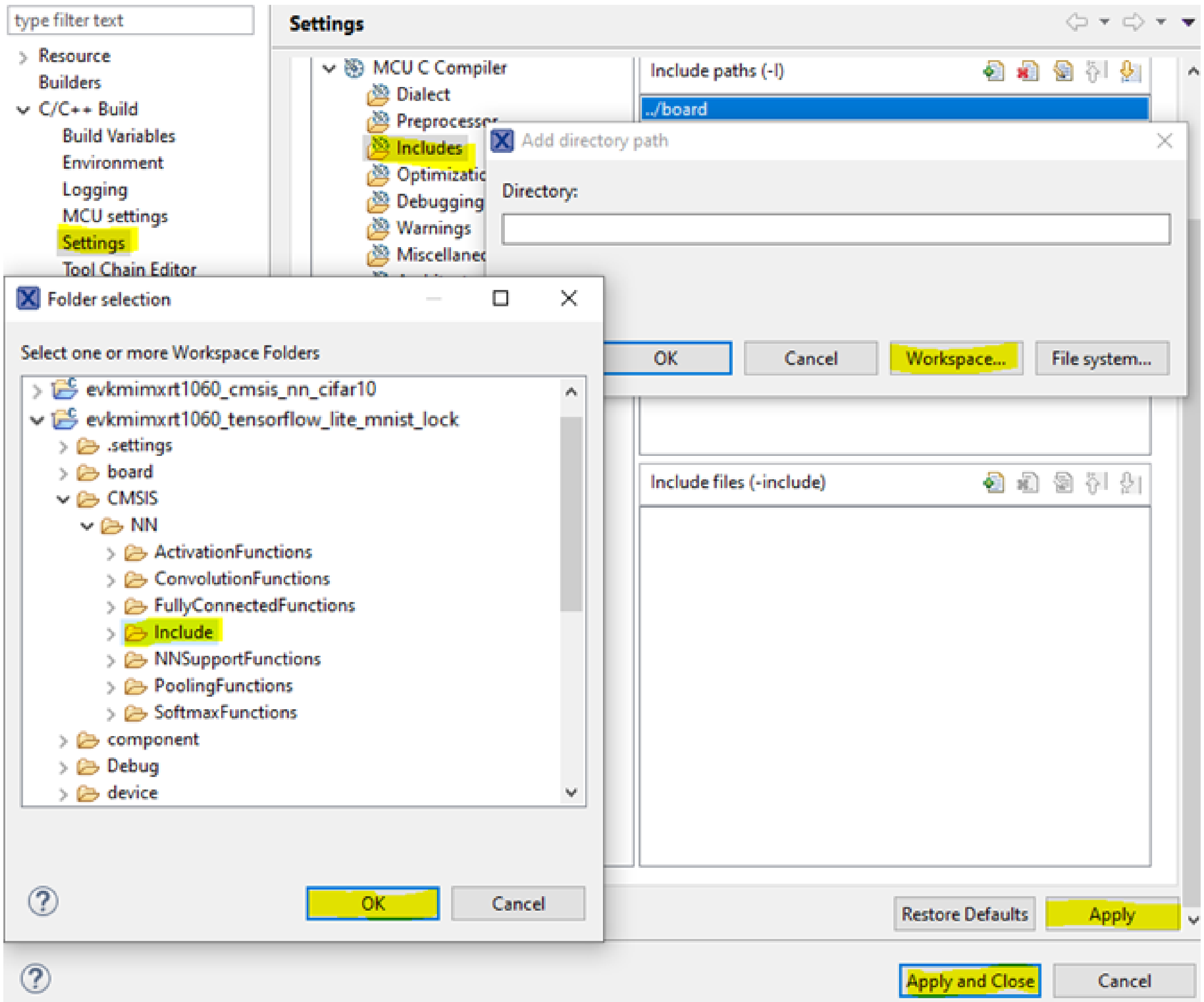


Figure 21. Neural network library

6. Select **Libraries** from the **MCU Linker** drop-down menu, and link the **arm_cortexM7lfdp_math** library as shown in figure below:

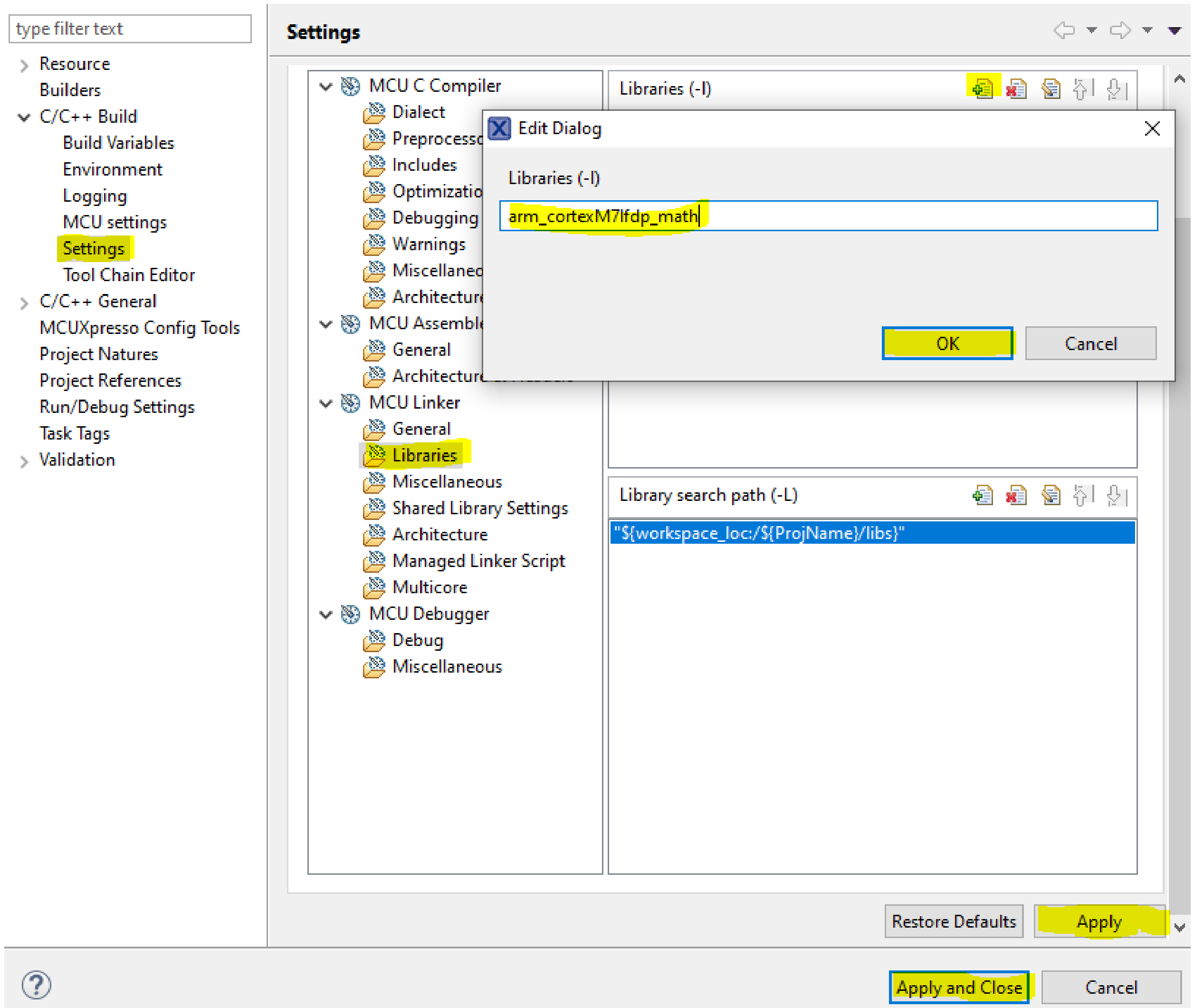


Figure 22. The arm_cortexM7lfdp_math library

7. Make sure **ARM_MATH_CM7=1** symbol is defined under **MCU C Compiler -> Preprocessor**, as shown in figure below.

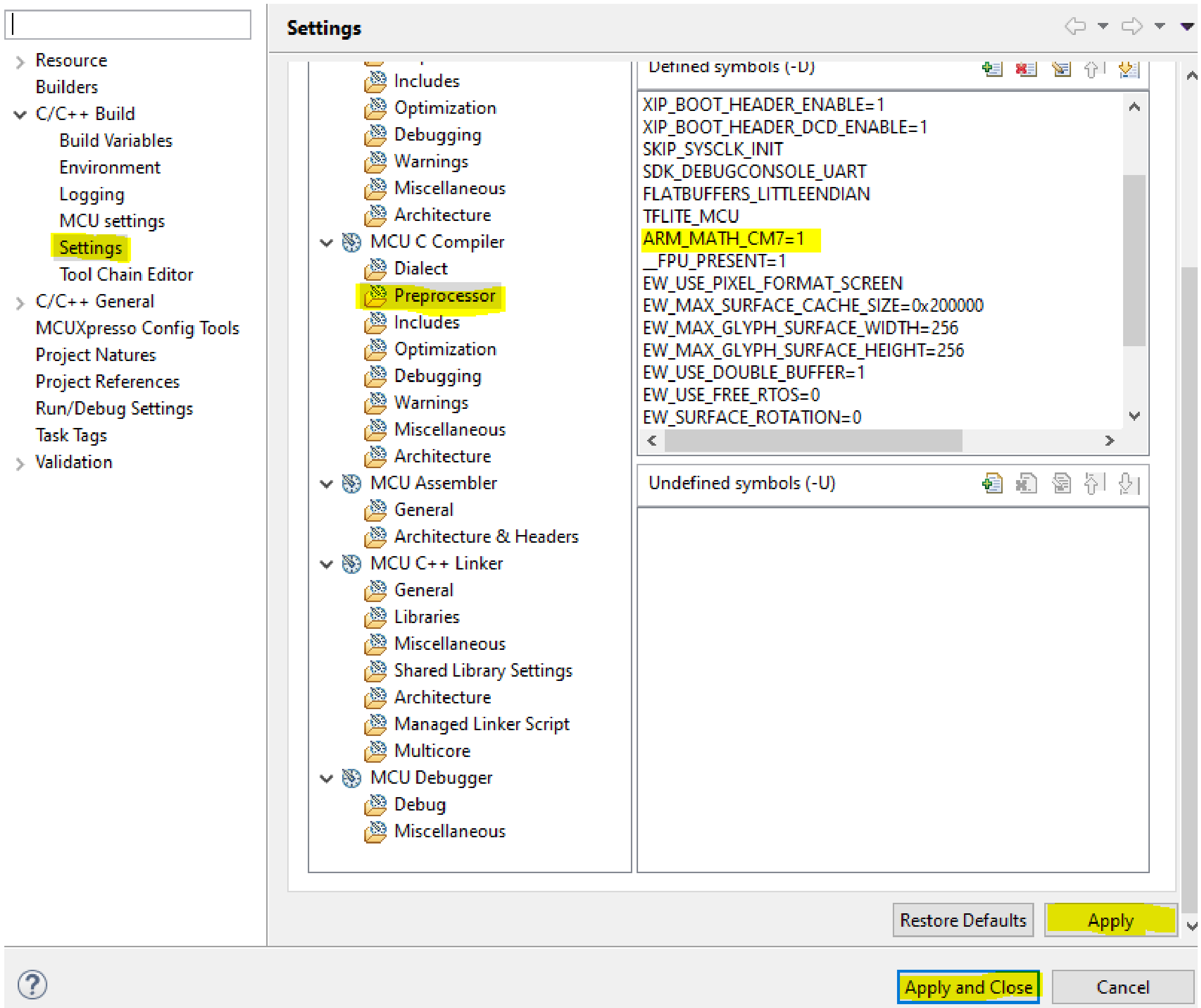


Figure 23. Enable ARM_MATH_CM7

8. Copy and replace **weights.h** and **parameter.h** header files in the application project with the header file available in the **code** folder (copied in previous section from Docker container). Your file should appear as shown below:

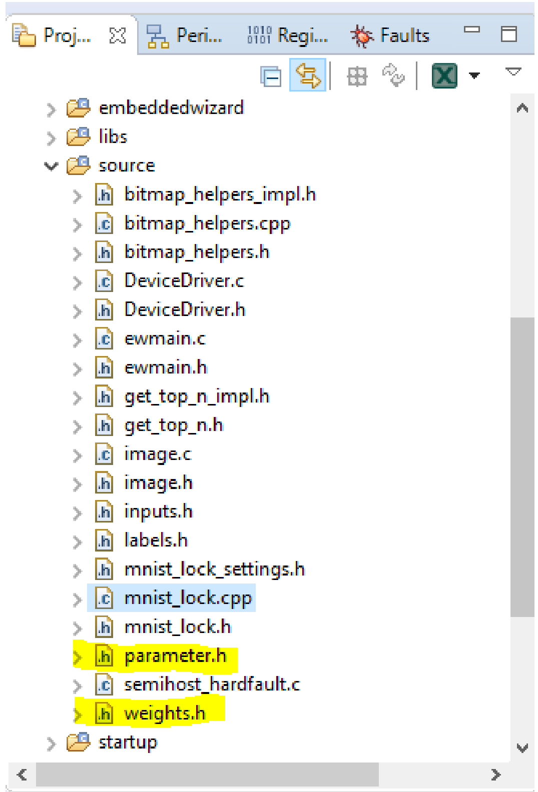


Figure 24. Adding header file

9. Add the Neural Network function (nn_run(uint8_t*)) and its buffer in the main source file of the application project. Do this by copying the contents of **nn.cpp** and **nn.h**, and replace the existing TF-Lite **InferenceInit()** and **RunInference()** functions with this content, in the main source file. Also, comment the statements as highlighted in figure below:

```
#define LOG(x) std::cout
#define PRINT_INPUT false
#define PRINT_CONFIDENCE false
#include "arm_math.h"
#include "parameter.h"
#include "weights.h"
#include "CMSIS/NN/Include/arm_nnfunctions.h"
static const uint8_t mean[DATA_OUT_CH*DATA_OUT_DIM*DATA_OUT_DIM] = MEAN_DATA;

static const q7_t conv1_wt[CONV1_IN_CH*CONV1_KER_DIM*CONV1_KER_DIM*CONV1_OUT_CH] = CONV1_WT;
static const q7_t conv1_bias[CONV1_OUT_CH] = CONV1_BIAS;

static const q7_t conv2_wt[CONV2_IN_CH*CONV2_KER_DIM*CONV2_KER_DIM*CONV2_OUT_CH] = CONV2_WT;
static const q7_t conv2_bias[CONV2_OUT_CH] = CONV2_BIAS;

static const q7_t conv3_wt[CONV3_IN_CH*CONV3_KER_DIM*CONV3_KER_DIM*CONV3_OUT_CH] = CONV3_WT;
static const q7_t conv3_bias[CONV3_OUT_CH] = CONV3_BIAS;

static const q7_t ip1_wt[IP1_IN_DIM*IP1_OUT_DIM] = IP1_WT;
static const q7_t ip1_bias[IP1_OUT_DIM] = IP1_BIAS;

//Add input_data and output_data in top main.cpp file
const char* labels[] = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};
//uint8_t image_data[DATA_OUT_CH*DATA_OUT_DIM*DATA_OUT_DIM]=DIGIT_IMG_DATA;
q7_t output_data[IP1_OUT_DIM];

q7_t col_buffer[5000];
q7_t scratch_buffer[27200];

void mean_subtract(uint8_t* image_data) {
  for(int i=0; i<DATA_OUT_CH*DATA_OUT_DIM*DATA_OUT_DIM; i++) {
    image_data[i] = (q7_t)__SSAT( ((int)(image_data[i] - mean[i]) >> DATA_RSHIFT), 8);
  }
}
```

Figure 25. Adding Neural Network function and its buffer in main source file

10. In the Neural Network function, (**run_nn()**), add marked statement, and make sure that return type for **rnn_nn()** function should be **int pointer** as shown below:

```
int *run_nn(uint8_t* input_data) {
  q7_t* buffer1 = scratch_buffer;
  q7_t* buffer2 = buffer1 + 15680;
  mean_subtract(input_data);
  arm_convolve_HWC_q7_basic((q7_t*)input_data, CONV1_IN_DIM, CONV1_IN_CH, conv1_wt, CONV1_OUT_CH, CONV1_KER_DIM,
  arm_relu_q7(buffer1, RELU1_OUT_DIM*RELU1_OUT_DIM*RELU1_OUT_CH);
  arm_maxpool_q7_HWC(buffer1, POOL1_IN_DIM, POOL1_IN_CH, POOL1_KER_DIM, POOL1_PAD, POOL1_STRIDE, POOL1_OUT_DIM,
  arm_convolve_HWC_q7_fast(buffer2, CONV2_IN_DIM, CONV2_IN_CH, conv2_wt, CONV2_OUT_CH, CONV2_KER_DIM, CONV2_PAD,
  arm_relu_q7(buffer1, RELU2_OUT_DIM*RELU2_OUT_DIM*RELU2_OUT_CH);
  arm_maxpool_q7_HWC(buffer1, POOL2_IN_DIM, POOL2_IN_CH, POOL2_KER_DIM, POOL2_PAD, POOL2_STRIDE, POOL2_OUT_DIM,
  arm_convolve_HWC_q7_fast(buffer2, CONV3_IN_DIM, CONV3_IN_CH, conv3_wt, CONV3_OUT_CH, CONV3_KER_DIM, CONV3_PAD,
  arm_relu_q7(buffer1, RELU3_OUT_DIM*RELU3_OUT_DIM*RELU3_OUT_CH);
  arm_maxpool_q7_HWC(buffer1, POOL3_IN_DIM, POOL3_IN_CH, POOL3_KER_DIM, POOL3_PAD, POOL3_STRIDE, POOL3_OUT_DIM,
  arm_fully_connected_q7_opt(buffer2, ip1_wt, IP1_IN_DIM, IP1_OUT_DIM, IP1_BIAS_LSHIFT, IP1_OUT_RSHIFT, ip1_bias
  arm_softmax_q7(output_data, 10, output_data);

  /* Get the object class with the highest confidence value */
  q7_t max_value;
  uint32_t max_index;
  arm_max_q7(output_data, 10, &max_value, &max_index);

  int *result_array = (int*) malloc(sizeof(int) * 2);
  result_array[0] = -1;
  result_array[1] = -1;

  if (result_array[0] == -1)
  {
    result_array[0] = (int)((((int)max_value + 128) * 100) / 255);
    result_array[1] = max_index;
  }
  if (PRINT_CONFIDENCE)
      LOG(INFO) << "  " << labels[max_index] << " (" << (int)result_array[0] << "% confidence)\r\n";
  return result_array;
}
```

Figure 26. Adding neural network function

11. Under **main() function** in the **mnist_lock.cpp** file, comment the TF-Lite **InferenceInit()** function, as shown in below figure.

```

    /* Tensorflow-lite initialization. */
//  tflite::mnist::InferenceInit(false);

```

Figure 27. Comment TF-Lite InferenceInit() function

12. Under **processImage()** function in the **mnist_lock.cpp** file, replace TF-Lite tensors with the **inputImage[]** array. Comment the TF-Lite tensors and add the marked statement as shown in below figure.

```
/*  int input = interpreter->inputs()[0];
    float* input_tensor = interpreter->typed_tensor<float>(input);

    int k = 0;
    for (int h = 0; h < 28; h++)
    {
        for (int w = 0; w < 28; w++)
        {
            input_tensor[k] = img->imageData[k] / 255.0;
            k++;
        }
    }

    if (PRINT_INPUT)
    {
        LOG(INFO) << "Input Tensor:\n\r";
        int l = 0;
        for (int h = 0; h < 28; h++)
        {
          for (int w = 0; w < 28; w++)
          {
              if (interpreter->typed_tensor<float>(input)[l] == 0)
                LOG(INFO) << "0";
              else
                LOG(INFO) << "1";

              l++;
          }
          LOG(INFO) << "\n\r";
        }
        LOG(INFO) << "\n\n\n\r";
        std::flush(std::cout);
    }*/
    uint8_t inputImage[785];
    int k = 0;
    for (int h = 0; h < 28; h++)
    {
        for (int w = 0; w < 28; w++)
        {
            inputImage[k] = img->imageData[k];
            // add extra 3 pixels to improve the quality of image
            if(img->imageData[k] == 255){       //        x
                if((h-1) < 28 )                 //      x 1 x
                    inputImage[w + (h - 1) * 28] =255;
                if((w-1)>0)
                    inputImage[(w - 1) + h * 28] =255;
                if((w+1) < 28)
                    inputImage[(w + 1) + h * 28] =255;
            }
            k++;
        }
    }
```

Figure 28. Replacing TF-lite tensors with inputImage[] array

13. Under **processImage()** function in the **mnist_lock.cpp** file, replace the TF-Lite inference function, **RunInference()**, with the CMSIS-NN function, **run_nn()**, in return statement.

```
//  tflite::mnist::Settings s; // use the default settings
//  return tflite::mnist::RunInference(&s);
    return run_nn(inputImage);
}
```

Figure 29. Replacing TF-lite inference function with CMSIS-NN function

14. Remove TF-Lite related files from the project. Below is a list of files to exclude from the project.

    - mnsit_lock_settings.h
    - labels.h
    - bitmap_helpers.h / bitmap_helpers.cpp
    - bitmap_helpers_impl.h
    - converted_model.h
    - get_top_n_impl.h
    - get_top_n.h
    - tensorflow_lite (folder)

    After removing all the TF-Lite project file, the file structure should appear as shown below.

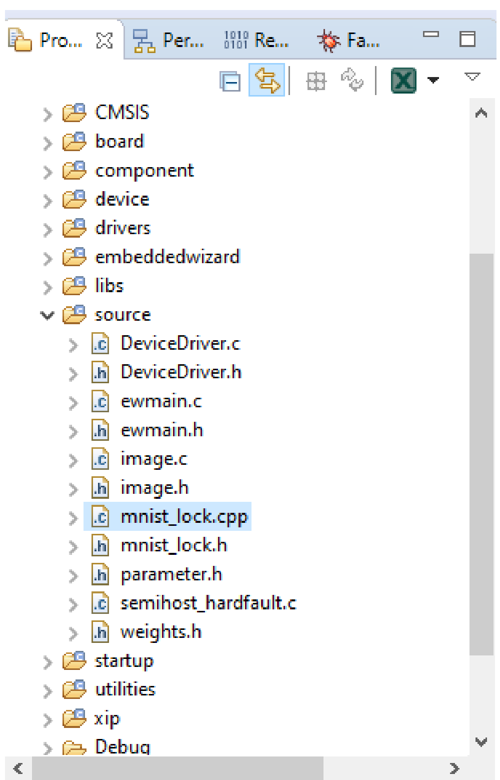


Figure 30. Removing TF-lite project files

4.3 Application details

In application code, when user drags finger over user input slot area on TFT LCD, single pixel line is drawn under the finger with pixel value “1” and all other pixels are assumed “0”. After pressing unlock/lock UI button on LCD, input digit image gets captured. The size of the captured image is 112x112 as compared to MNIST dataset 28x28. So, it needs to be resized to 28x28 to match the inference input.

To preserve the captured image, extra pixel of 3x3 matrix size is added across each pixel value “1”. Then image is cropped and resized to 28x28. To improve the quality of image further, three extra pixels are added.

The application passes the image as pointer argument to CNN by calling the **run_nn(uint8_t*)** function. The **arm_softmax_q7( )** function returns the prediction of highest class for image recognition.

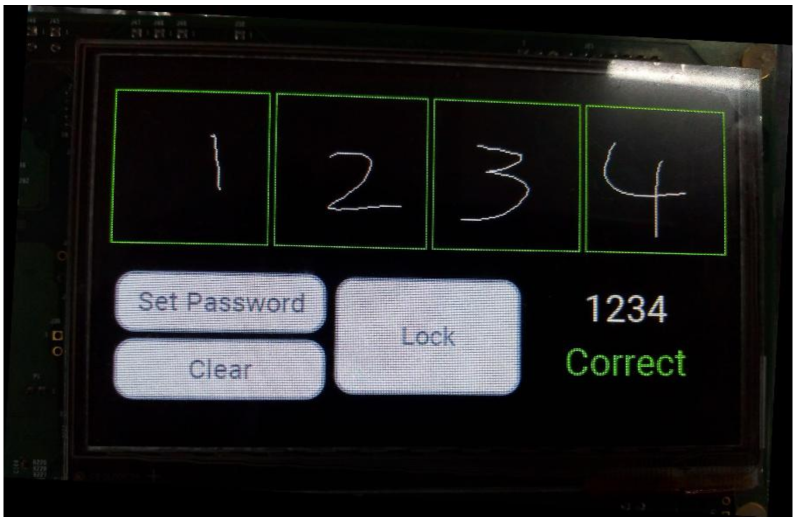


Figure 31. Correct input

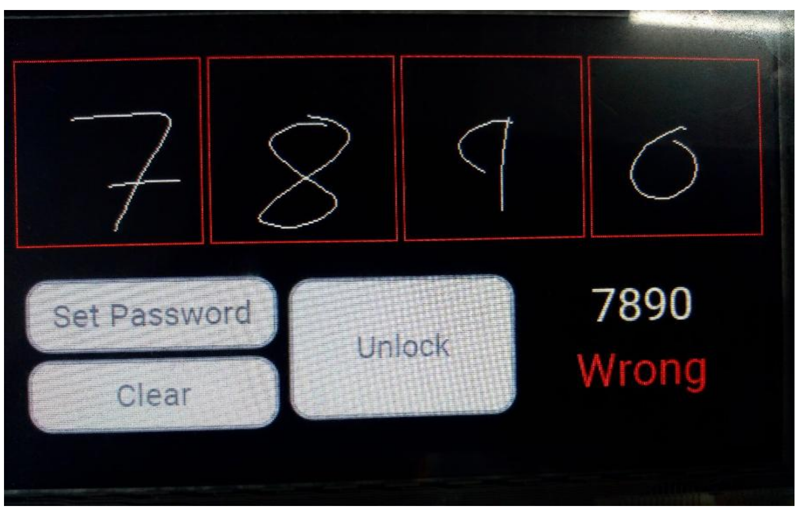


Figure 32. Incorrect input

For more details on the MNIST Lock application, see application note AN12603.

5 Test report on MNIST dataset

The section illustrates results that demonstrate the behavior of the application when using different parameters for the CNN definition and using user-defined training data.

5.1 Redesigning Caffe model and testing

You can redesign Caffe model by changing CNN parameter in the following designs.

- **Design 1**: Default design in Caffe model definition file, **alexnet_train_test.prototxt**
- **Design 2**:
  - Filters in CONV1 layer = 20 (default)
  - Filters in CONV2 layer vary from 20 to 32
  - Filters in CONV2 layer vary from 50 to 64 in CNN Architecture.
- **Design 3**: Iteration for training Caffe model varies from 10k to 20k.

The Test report table below shows result of all the designs for each test case after redesigning Caffe model.

Table 1. Test report after redesigning Caffe model

Test type	Total no. of digits	Accuracy(%)
Design 1	100	98
Design 2	100	92
Design 3	100	97

Note: The test was performed with MNIST dataset on CMSIS-NN hello world application for total 100 images (10 images for each digit from 0 to 9). Same image was used for all the test cases.

5.2 Retrain model using captured image from LCD screen

The test is performed by training the Caffe model with the captured images from LCD screen or MNIST dataset.

- **Design 1**: Retrained Caffe model = 8 k images (LCD).
- **Design 2**: Trained Caffe model = 8 k images (LCD), 54 K images (MNIST dataset).
- **Design 3**: Trained Caffe model = 8 k images (LCD), 72 K images (MNIST dataset).

The table below shows the test report of each test case after training Caffe model with captured LCD image.

Table 2. Test report after training Caffe model with captured LCD image

Test type	Total no. of digits	Accuracy(%)
Design 1	100	98
Design 2	100	97
Design 3	100	99

NOTE

Accuracy for image testing may vary depending upon dataset on which model is trained and on the images through which testing is performed.

6 Conclusion

This document explains Caffe model training with MNIST dataset for image classification of trained model using CMSIS-NN. The document also describes how the trained model is converted to C source files and how to implement it on different existing projects running on i.MX RT platforms.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 17 April 2020

Document identifier: AN12781