AN12580

NXP eIQ™ Machine Learning Software Development Environment for QorIQ Layerscape Applications Processors

Rev. 1 — 09/2019 Application Note

Contents

1 Introduction

This document provides guidance for the machine learning software for the QorIQ Layerscape processors. The document is divided into separate sections, starting with the NXP eIQ introduction, the eIQ installation guide, and the step-by step guide for running few examples.

Machine Learning (ML) is a computer science domain that has its roots in the 1960s. ML provides algorithms capable of finding patterns and rules in data. It is a category of algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of ML is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.

In 2010, the so-called deep learning started. It is a fast-growing subdomain of ML, based on Neural Networks (NN). Inspired by the human brain, deep learning achieved state-of-the-art results in various tasks; for example, Computer Vision (CV) and Natural Language Processing (NLP). Neural networks are capable of learning complex patterns from millions of examples. A huge adaptation is expected in the embedded world, where NXP is the leader. NXP created eIQ™ machine learning software for QorIQ Layerscape applications processors, a set of ML tools which allows developing and deploying ML applications on the QorIQ Layerscape family of devices.

2 NXP eIQ software

The NXP eIQ[1] machine learning software development environment provides a shell script to install machine learning applications targeted at NXP QorIQ Layerscape processors. The NXP eIQ software is concerned only with neural networks inference and standard machine-learning algorithms; leaving neural network training to other specialized software tools and dedicated hardware. The NXP eIQ is continuously expanding to include data-acquisition and curation tools and model conversion for a wide range of NN frameworks and inference engines, such as TensorFlow, TensorFlow Lite, Arm® NN, and Arm Compute Library.

The NXP eIQ software delivers machine learning enablement by providing ML support in LSDK for the two QorIQ Layerscape processors LS1046A and LX2160A. The main features of NXP eIQ software are:

- OpenCV 4.0.1
- Arm Compute Library 19.02
- Arm NN 19.02
- TensorFlow 1.12
- TensorFlow Lite 1.12

For up-to-date information about NXP machine learning solutions, see the official NXP webpage[2] for machine learning and artificial intelligence.

[1] NXP eIQ Software
[2] NXP eIQ Software Support Community

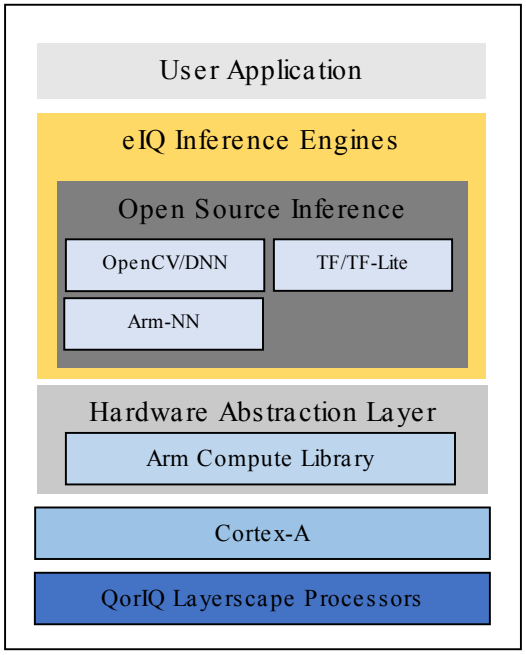

Figure 1. NXP eIQ machine learning software architecture

3 eIQ installation guide

3.1 Prerequisites

Hardware requirements

- 1 x LS1046ARDB or LX2160RDB board with a minimum 64 GB of space available
- Make sure that the Ethernet port of the device is up and can access the internet

Software requirements

- Host Operating System: Ubuntu (tested with Ubuntu 18.04), for more information about the Ubuntu setup, see the Layerscape Software Development Kit User Guide.[3]
- Host packages:
  - Ubuntu OS host basic packages setup `$: sudo apt-get install wget unzip sed git curl`
- Before running eIQ installation script, make sure the Ethernet port of the device is up and is able to access internet, date is set correctly for `apt-get`, and `git` is configured properly

3.2 Installing eIQ software

When Ubuntu bootup is successful on the device, run the eIQ software installation script. The installation script is located in this folder:*/opt/eiq*.

[3] Layerscape SDK User Guide

Run this command to execute the installation script:

```
$  ./eiq_builder.sh [OPTION]
```

```
OPTIONs:
   --with-tflite       build with tflite
   --with-tensorflow   build with tensorflow
   --with-opencv       build with opencv
   --with-armnn        build with armnn
   --skip-dependency   do not install dependency before building
-c --clean             cleanup build env before building
-j --jobs              the number of jobs for building, default value is 4
-h --help              display this help and exit
```

For example, to install OpenCV framework run this command:

```
$ ./eiq_builder.sh --with-opencv
```

NOTE

If the Linux host machine is in a subnet that needs HTTP proxy to access external Internet, set environment variable `http_proxy` and `https_proxy` as follows:

```
export http_proxy=http://<account>:<password>@<domain>:<port>
export https_proxy=http://<account>:<password>@<domain>:<port>
```

4 OpenCV (Open source computer vision)

OpenCV is an open-source computer vision library. OpenCV offers a unitary solution for both the neural network inference (DNN module) and the standard machine learning algorithms (ML module). It includes many computer vision functions, making it easier to build complex machine learning applications in a short amount of time and without being dependent on other libraries.

OpenCV has wide adoption in the computer vision field and is supported by a strong and active community. The key algorithms are specifically optimized for various devices and instructions sets. For QorIQ Layerscape processor, OpenCV uses the Arm NEON acceleration. The Arm NEON technology is an advanced SIMD (Single Instruction Multiple Data) architecture extension for the Arm Cortex-A series. The Arm NEON technology is intended to improve multimedia user experience by accelerating the audio and video encoding/decoding, user interface, 2D/3D graphics, or gaming. The Arm NEON can also accelerate the signal-processing algorithms and functions to speed up applications such as the audio and video processing, voice and facial recognition, computer vision, and deep learning.

At its core, the OpenCV DNN module implements an inference engine and does not provide any functionalities for neural network training. For more details about the supported models and layers, see the official OpenCV DNN wiki page[4].

On the other hand, the OpenCV ML module contains classes and functions for solving machine learning problems such as classification, regression, or clustering. It involves algorithms such as Support Vector Machine (SVM), decision trees, random trees, expectation maximization, k-nearest neighbors, classic Bayes classifier, logistic regression, and boosted trees. For more information, see the official reference manual and machine learning overview. For more details about OpenCV 4.0.1, see the official OpenCV change log webpage[5].

4.1 OpenCV DNN demos

OpenCV DNN demos are installed in this folder: */usr/share/OpenCV/samples/bin/*

However, the input data, model configurations, and model weights are not located in this folder, because of their size. Download these files to the device before running the demos:

[4] Deep Learning in OpenCV
[5] OpenCV Change Logs

- Download the *opencv_extra.zip* package using this link: github.com/opencv/opencv_extra/tree/4.0.1 and unpack the file to the home directory <home_dir>

```
$ unzip opencv_extra-4.0.1.zip
```

- The input images, model configurations for some OpenCV examples are in this folder:*<home_dir>/opencv_extra-4.0.1/testdata/dnn*
- The models files comes with two options:
  - Option 1: Run `opencv_extra` script to download all the `opencv_extra` dependencies

    ```
    $ python download_models.py
    ```

    The script downloads the NN models, configuration files, and input images for some OpenCV examples. This operation may take time to complete, depending upon the network speed.
  - Option 2: Direct download using `wget` command. For example, to download caffe model of SqueezeNet, run this command:

    ```
    $ wget https://raw.githubusercontent.com/DeepScale/SqueezeNet/
    b5c3f1a23713c8b3fd7b801d229f6b04c64374a5/SqueezeNet_v1.1/squeezenet_v1.1.caffemodel
    ```

- Copy these downloaded dependencies to this folder: */usr/share/OpenCV/samples/bin*
- Configuration model file *models.yml* is already in folder: */usr/share/OpenCV/samples/bin/*. The *models.yml* file contains the pre-processing parameters for some DNN examples, which accepts the `--zoo` parameter.
- When taking a remote login to the device by ssh command, add `-X` parameter to enable X11 forwarding, to display example's image result

```
$ ssh -X <device IP address>
```

- Set the environment variables for execution:

```
$ export LD_LIBRARY_PATH=/usr/share/OpenCV/samples/lib:$ LD_LIBRARY_PATH
```

4.1.1 Image classification example

This demo performs image classification using a pre-trained SqueezeNet network.

Demo dependencies (taken from *<home_dir>/opencv_extra-4.0.1/testdata/dnn* directory):

- *dog416.png*
- *squeezenet_v1.1.prototxt*

Demo dependencies for model file (using `wget` as an option):

- *squeezenet_v1.1.caffemodel*

```
$ wget https://raw.githubusercontent.com/DeepScale/SqueezeNet/
b5c3f1a23713c8b3fd7b801d229f6b04c64374a5/SqueezeNet_v1.1/squeezenet_v1.1.caffemodel
```

Other demo dependencies:

- *classification_classes_ILSVRC2012.txt* from */usr/share/OpenCV/samples/data/dnn*
- *models.yml* from */usr/share/OpenCV/samples/bin/*

Running the C++ example with the image input from the default location:

```
$ ./example_dnn_classification --input=dog416.png --zoo=models.yml --classes=../data/dnn/
classification_classes_ILSVRC2012.txt squeezenet
```

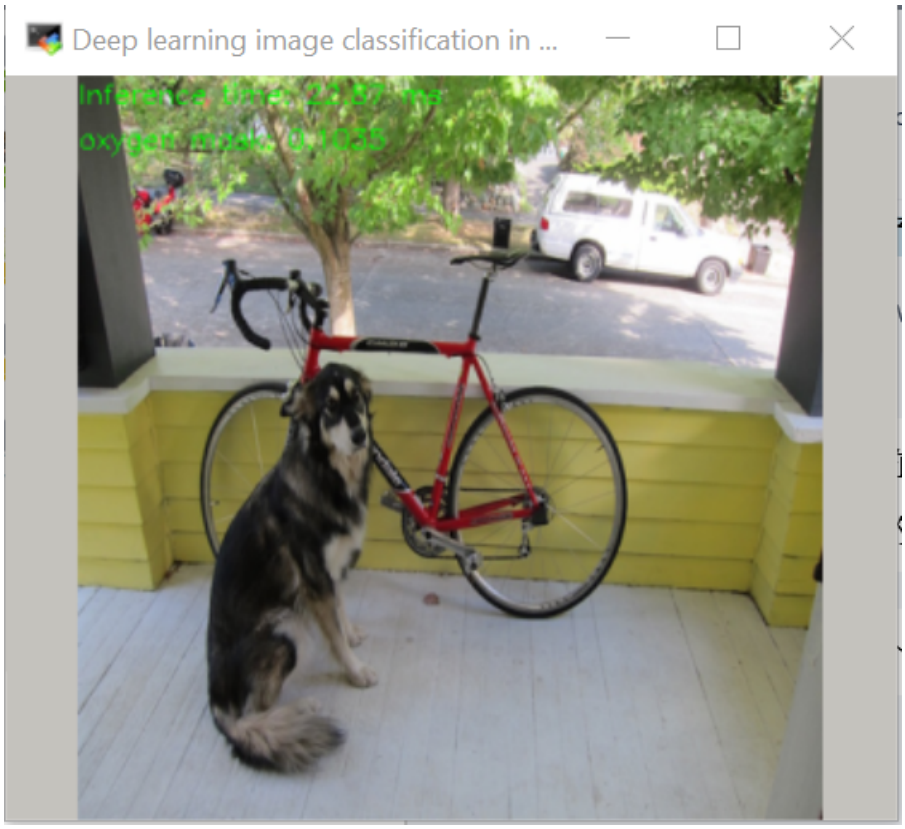

Figure 2. Image classification graphics output

4.1.2 Human pose estimation example

This application demonstrates the human or hand pose detection with a pretrained OpenPose DNN. The demo supports only input images, not the live camera input.

Demo dependencies (taken from *<home_dir>/opencv_extra-4.0.1/testdata/dnn* directory):

- *grace_hopper_227.png*
- *openpose_pose_coco.prototxt*

Demo dependencies for model file (using `wget` as an option):

- *openpose_pose_coco.caffemodel*

```
$ wget http://posefs1.perception.cs.cmu.edu/OpenPose/models/pose/coco/pose_iter_440000.caffemodel -
O openpose_pose_coco.caffemodel
```

Running the C++ example with the image input from the default location:

```
$ ./example_dnn_openpose --model=openpose_pose_coco.caffemodel --proto=openpose_pose_coco.prototxt --
image=grace_hopper_227.png --width=227 --height=227
```

Figure 3. Human pose estimation graphics output

4.1.3 Text detection example

This demo is used for text detection in the image using the EAST algorithm.

Demo dependencies (using `wget` as an option):

- *frozen_east_text_detection.pb*

  Download and unpack the model file:

```
$ wget https://www.dropbox.com/s/r2ingd0l3zt8hxs/frozen_east_text_detection.tar.gz?dl=1 -O
frozen_east_text_detection.tar.gz
$ tar xvf frozen_east_text_detection.tar.gz
```

Other demo dependencies (taken from */usr/share/OpenCV/samples/data/* directory):

- *imageTextN.png*

Running the C++ example with the image input from the default location:

```
$ ./example_dnn_text_detection --model=frozen_east_text_detection.pb --input=../data/imageTextN.png
```

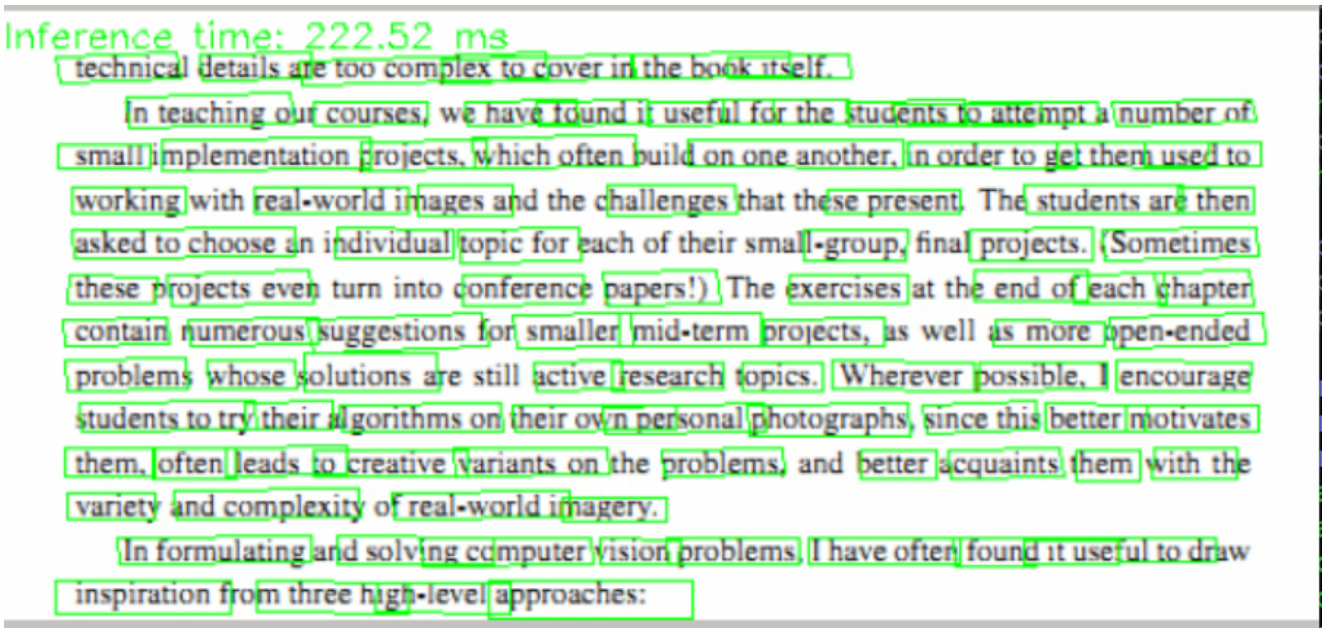

Figure 4. Text detection graphics output

5 Arm Compute Library

The Arm Compute Library[6] is a collection of low-level software functions optimized for Arm Cortex CPU and Arm Mali GPU architectures, targeted at a variety of use-cases including: image processing, computer vision and machine learning. It is a convenient repository of low-level, optimized software functions that developers can source individually or use as part of complex pipelines in order to accelerate their algorithms and applications.

The example codes of Arm Compute Library are located at folder *ComputeLibrary/examples* and the following section is an example based on AlexNet using the graph API.

5.1 Running AlexNet using graph API

In 2012, AlexNet became famous when it won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual challenge that aims to evaluate algorithms for object detection and image classification. AlexNet is made up of eight trainable layers (five convolution layers and three fully-connected layers). All the trainable layers are followed by the ReLu activation function, except for the last fully-connected layer, where the Softmax function is used.

Follow this procedure to run the demo:

1. Download the archive file to the example location folder from:

   ```
   https://armkeil.blob.core.windows.net/developer/developer/technologies/Machine%20learning%20on
   %20Arm/Tutorials/Running%20AlexNet%20on%20Pi%20with%20Compute%20Library/
   compute_library_alexnet.zip
   ```

   This file contains:

   - The trainable parameters from AlexNet model.
   - A text file, containing the ImageNet labels that are required to map the predicted objects to the name of the classes.
   - Several images in the ppm file format that are ready to be used with the network.

[6] Arm Compute Llibrary

2. Create new sub-folder and unzip the file

```
$ mkdir assets_alexnet
$ unzip compute_library_alexnet.zip -d assets_alexnet
```

3. Run alexnet example:

```
$ graph_alexnet --data=./assets_alexnet/ --image=./assets_alexnet/go_kart.ppm --labels=./
assets_alexnet/labels.txt
```

The output of the successful classification is as follows:

```
graph_alexnet

Threads : 1
Target : NEON
Data type : F32
Data layout : NHWC
Tuner enabled? : false
Tuner file :
Fast math enabled? : false
Data path : ./assets_alexnet/
Image file : ./assets_alexnet/go_kart.ppm
Labels file : ./assets_alexnet/labels.txt

---------- Top 5 predictions ----------

0.9736 - [id = 573], n03444034 go-kart
0.0118 - [id = 518], n03127747 crash helmet
0.0108 - [id = 751], n04037443 racer, race car, racing car
0.0022 - [id = 817], n04285008 sports car, sport car
0.0006 - [id = 670], n03791053 motor scooter, scooter

Test passed
```

6 TensorFlow

TensorFlow[7] is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that enable the researchers to push the state-of-the-art in ML and give the developers the ability to easily build and deploy ML-powered applications.

TensorFlow provides a collection of workflows[8] with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages. TensorFlow provides a variety of different toolkits that enable you to construct models at your preferred level of abstraction. Use the lower-level APIs to build models by defining a series of mathematical operations. Alternatively, you can use higher-level APIs to specify pre-defined architectures, such as linear regressors or neural networks.

6.1 Running image classification example

This simple example is pre-installed by the build script. Its name is *label_image.py*. It classifies images such as hats and shirts. A proper model file for this example is not included in the target image by default due to its size. The location of the example binary file is: */usr/share/tflite*.

Demo dependencies:

[7] TensorFlow
[8] TensorFlow sources

- Download the TensorFlow model file to the example folder.

```
$ wget https://storage.googleapis.com/download.tensorflow.org/models/
inception_v3_2016_08_28_frozen.pb.tar.gz
```

- Unpack the model file:

```
$ tar xvf inception_v3_2016_08_28_frozen.pb.tar.gz
```

- Run the example with the command-line arguments from the default location:

```
$ python label_image.py --graph inception_v3_2016_08_28_frozen.pb --labels
imagenet_slim_labels.txt --image grace_hopper.jpg
```

Figure 5. Image classification input picture

The output of a successful classification for the above picture as an input is as follows:

```
military uniform 0.834305
mortarboard 0.0218695
academic gown 0.0103581
pickelhaube 0.00800817
bulletproof vest 0.00535086
```

7 TensorFlow Lite

TensorFlow Lite is a light-weight version of and a next step from TensorFlow. TensorFlow Lite is an open-source software library focused on running machine learning models on mobile and embedded devices (available at www.tensorflow.org/lite). It enables on-device machine learning inference with low latency and small binary size. TensorFlow Lite also supports hardware acceleration using Android™ OS neural network APIs.

TensorFlow Lite supports a set of core operators (both quantized and float) tuned for mobile platforms. They incorporate pre-fused activations and biases to further enhance the performance and quantized accuracy. Additionally, TensorFlow Lite also supports the use of custom operations in models.

TensorFlow Lite defines a new model file format, based on FlatBuffers[9]. FlatBuffers is an open-source, efficient, cross-platform serialization library. It is similar to protocol buffers, but the primary difference is that FlatBuffers does not need a parsing/unpacking step for a secondary representation before you can access the data, often coupled with per-object memory allocation. Also, the code footprint of FlatBuffers is an order of magnitude smaller than protocol buffers.

TensorFlow Lite has a new mobile-optimized interpreter, which has the key goal to keep apps lean and fast. The interpreter uses static graph ordering and a custom (less-dynamic) memory allocator to ensure minimal load, initialization, and execution latency.

[9] FlatBuffers

7.1 Running benchmark application

This example is pre-installed by the build script and its name is *benchmark_model*. It performs simple TensorFlow Lite benchmarking using the pre-defined models. The model file is not included in the target image, because of its size. The location of the example binary file is: */usr/share/tflite*.

Demo dependencies:

- Download the model file[10] used in this example:

```
$ wget download.tensorflow.org/models/mobilenet_v1_224_android_quant_2017_11_08.zip
```

- Unpack the model file:

```
$ unzip mobilenet_v1_224_android_quant_2017_11_08.zip
```

- Run the example with the command-line arguments from the default location:

```
$ ./benchmark_model --graph=mobilenet_quant_v1_224.tflite
```

The output of a successful TensorFlow Lite benchmarking is as follows:

```
STARTING!
Num runs: [50]
Inter-run delay (seconds): [-1]
Num threads: [1]
Benchmark name: []
Output prefix: []
Warmup runs: [1]
Graph: [mobilenet_quant_v1_224.tflite]
Input layers: []
Input shapes: []
Use nnapi : [0]
Loaded model mobilenet_quant_v1_224.tflite
resolved reporter
Initialized session in 10.902ms
Running benchmark for 1 iterations
count=1 curr=79853

Running benchmark for 50 iterations
count=50 first=72898 curr=72600 min=72568 max=73216 avg=72712.1 std=131

Average inference timings in us: Warmup: 79853, Init: 10902, no stats: 72712.1
```

8 Arm NN

Arm NN is an open-source inference engine framework developed by Arm and supporting a wide range of neural-network model formats, such as Caffe, TensorFlow, TensorFlow Lite, and ONNX. For Layerscape CPUs, Arm NN runs on the CPU with NEON, and has multi-core support.

8.1 Running Arm NN tests

The Arm NN SDK provides a set of tests, which can also be considered as demos, showing what the Arm NN does and how to use it. They load neural network models of various formats (Caffe, TensorFlow, TensorFlow Lite, and ONNX), run the inference on a specified input data, and provides inference result.

[10] TensorFlow Hosted Models

NOTE

- The input data, model configurations, and model weights are not distributed with Arm NN. Download them separately and make sure they are available on the device before running the tests.
- The input file names are hardcoded. So, investigate the code to find out what input file names are expected.

To get started with Arm NN, the following sections explain how to prepare the input data and how to run the Arm NN tests. All of them use well-known neural network models. With only few exceptions, such pre-trained networks are available to download from the internet. The input image files and their name, format, and content are deduced by analyzing the code. However, this was not possible for all the tests. It is recommended to prepare the data on the host and then deploy them on the Layerscape board, where the current Arm NN tests are run.

The following sections assume that the neural network model files are stored in a folder called models, and the input image files are stored in a folder called data. Both of them are created inside a folder called *ArmnnTests*. Create this folder structure on the larger partition using the following commands:

```
$ mkdir ArmnnTests
$ cd ArmnnTests
$ mkdir data
$ mkdir models
```

8.2 Caffe tests

The Arm NN 19.02 SDK provides the following set of tests for the Caffe models:

```
/usr/local/bin/CaffeAlexNet-Armnn
/usr/local/bin/CaffeCifar10AcrossChannels-Armnn
/usr/local/bin/CaffeInception_BN-Armnn
/usr/local/bin/CaffeMnist-Armnn
/usr/local/bin/CaffeResNet-Armnn
/usr/local/bin/CaffeVGG-Armnn
/usr/local/bin/CaffeYolo-Armnn
```

Two important limitations might require a pre-processing of the Caffe model file before running the Arm NN Caffe test. Firstly, the Arm NN tests require the batch size to be set to 1. Secondly, the Arm NN does not support all Caffe syntaxes, so some previous neural-network model files require updates to the latest Caffe syntax. How to perform these pre-processing steps is described at the Arm NN GitHub page. Also, install Caffe[11] on the host.

For example, you have a Caffe model that either has the batch size different than 1 or uses another Caffe defined by files *model_name.prototxt* and *model_name.caffemodel*, create a copy of the **.prototxt* file (*new_model_name.prototxt*), modify this file to use the new Caffe syntax, change the batch size to 1, and execute this Python script:

```
import caffe
new_net = caffe.Net('new_model_name.prototxt', 'model_name.caffemodel', caffe.TEST)
new_net.save('new_model_name.caffemodel')
```

Following is an example about how to run Caffe ArmNN.

8.2.1 CaffeAlexNet-Armnn

To run this test use the folder structure described in the introductory part and perform these steps:

- Download the caffe model files:

```
$ wget raw.githubusercontent.com/BVLC/caffe/master/models/bvlc_alexnet/deploy.prototxt
$ wget dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel
```

[11] Arm NN documentation for caffe support

- Transform the network as explained in the introductory part of Caffe tests.
- Rename *bvlc_alexnet.caffemodel* to *bvlc_alexnet_1.caffemodel*.
- Copy the *bvlc_alexnet_1.caffemodel* file to the models folder on the device.
- Find a **.jpg* file that contains a shark. Rename it to *shark.jpg* and copy it to the data folder on the device.
- Run the test:

```
$ cd ArmnnTests
$ CaffeAlexNet-Armnn --data-dir=data --model-dir=models
```

The output of a successfully classification is as following:

```
ArmNN v20190200

= Prediction values for test #0
Top(1) prediction is 2 with confidence: 98.7989%
Top(2) prediction is 0 with confidence: 2.81055e-06%
Total time for 1 test cases: 0.123 seconds
Average time per test case: 123.500 ms
Overall accuracy: 1.000
```

8.3 Tensorflow test

The Arm NN 19.02 SDK provides the following set of tests for the TensorFlow models:

```
/usr/local/bin/TfCifar10-Armnn
/usr/local/bin/TfInceptionV3-Armnn
/usr/local/bin/TfMnist-Armnn
/usr/local/bin/TfMobileNet-Armnn
/usr/bin/TfResNext-Armnn
```

Following section is a TensorFlow example based on MobileNet. Before running the tests, the TensorFlow models must be prepared for inference.

8.3.1 TfMobileNet-Armnn

To run this test use the folder structure described in the introductory part and perform these steps:

- From your host machine, download, and unpack the model file:

```
$ wget download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_1.0_224.tgz
```

- Copy the *mobilenet_v1_1.0_224_frozen.pb* file to the models folder on the device.
- Find a **.jpg* file containing a shark. Rename it to *shark.jpg* and copy it to the data folder on the device.
- Find a **.jpg* file containing a Labrador dog. Rename it to *Dog.jpg* and copy it to the data folder on the device.
- Find a **.jpg* file containing a tiger cat. Rename it to *Cat.jpg* and copy it to the data folder on the device.
- Run the test:

```
$ cd ArmnnTests
$ TfMobileNet-Armnn --data-dir=data --model-dir=models
```

The output of a successfully classification is as following:

```
ArmNN v20190200
```

```
= Prediction values for test #0
Top(1) prediction is 209 with confidence: 91.5388%
Top(2) prediction is 208 with confidence: 7.11163%
Top(3) prediction is 169 with confidence: 0.204635%
Top(4) prediction is 163 with confidence: 0.0989012%
Top(5) prediction is 160 with confidence: 0.0457351%
= Prediction values for test #1
Top(1) prediction is 283 with confidence: 67.1147%
Top(2) prediction is 282 with confidence: 12.0025%
Top(3) prediction is 67 with confidence: 0.000347048%
Top(4) prediction is 25 with confidence: 6.53171e-05%
Top(5) prediction is 6 with confidence: 4.69391e-05%
= Prediction values for test #2
Top(1) prediction is 3 with confidence: 62.8025%
Top(2) prediction is 1 with confidence: 0.00161201%
Top(3) prediction is 0 with confidence: 1.48381e-08%
Total time for 3 test cases: 0.281 seconds
Average time per test case: 93.611 ms
Overall accuracy: 1.000
```

8.4 Tensorflow Lite test

The Arm NN 19.02 SDK provides the following set of tests for the TensorFlow models:

```
/usr/local/bin/TfCifar10-Armnn
/usr/local/bin/TfInceptionV3-Armnn
/usr/local/bin/TfMnist-Armnn
/usr/local/bin/TfMobileNet-Armnn
/usr/bin/TfResNext-Armnn
```

Following section is a TensorFlow Lite example based on MobileNet. Before running the tests, the TensorFlow Lite models must be prepared for inference.

8.4.1 TfLiteMobilenetQuantized-Armnn

To run this test use the folder structure described in the introductory part and perform these steps:

- From your host machine, download, and unpack the model file:

  ```
  http://download.tensorflow.org/models/mobilenet_v1_2018_08_02/mobilenet_v1_1.0_224_quant.tgz
  ```

- Copy the *mobilenet_v1_1.0_224_quant.tflite* file to the models folder on the device.
- Find a **.jpg* file containing a shark. Rename it to *shark.jpg* and copy it to the *data* folder on the device.
- Find a **.jpg* file containing a Labrador dog. Rename it to *Dog.jpg* and copy it to the *data* folder on the device.
- Find a **.jpg* file containing a tiger cat. Rename it to *Cat.jpg* and copy it to the *data* folder on the device.
- Run the test:

  ```
  $ cd ArmnnTests
  $ TfLiteMobilenetQuantized-Armnn --data-dir=data --model-dir=models
  ```

The output of a successfully classification is as following:

```
ArmNN v20190200

= Prediction values for test #0
Top(1) prediction is 209 with confidence: 93.75%
Top(2) prediction is 208 with confidence: 5.46875%
```

```
Top(3) prediction is 0 with confidence: 0%
= Prediction values for test #1
Top(1) prediction is 283 with confidence: 54.6875%
Top(2) prediction is 282 with confidence: 19.9219%
Top(3) prediction is 0 with confidence: 0%
= Prediction values for test #2
Top(1) prediction is 3 with confidence: 73.8281%
Top(2) prediction is 0 with confidence: 0%
Total time for 3 test cases: 0.280 seconds
Average time per test case: 93.475 ms
Overall accuracy: 1.000
```

8.5 ONNX test

The Arm NN provides the following set of tests for ONNX models:

```
/usr/bin/OnnxMnist-Armnn
/usr/bin/OnnxMobileNet-Armnn
```

The following section is an example about how to run ONNX test.

8.5.1 OnnxMnist-Armnn

To run this test use the folder structure described in the introductory part and perform these steps:

- From your host machine, download, and unpack the model file:

  ```
  s3.amazonaws.com/onnx-model-zoo/mobilenet/mobilenetv2-1.0/mobilenetv2-1.0.tar.gz
  ```

- Copy the unpacked *mobilenetv2-1.0.onnx* file to the models folder on the device.
- Find a **.jpg* file containing a shark. Rename it to *shark.jpg* and copy it to the data folder on the device.
- Find a **.jpg* file containing a Labrador dog. Rename it to *Dog.jpg* and copy it to the data folder on the device.
- Find a **.jpg* file containing a tiger cat. Rename it to *Cat.jpg* and copy it to the data folder on the device.
- Run the test:

  ```
  $ cd ArmnnTests
  $ OnnxMobileNet-Armnn --data-dir=data --model-dir=models -i 3
  ```

The output of a successfully classification is as following:

```
ArmNN v20190200

= Prediction values for test #0
Top(1) prediction is 208 with confidence: 1635.36%
Top(2) prediction is 207 with confidence: 1401.98%
Top(3) prediction is 162 with confidence: 1125.92%
Top(4) prediction is 154 with confidence: 822.216%
Top(5) prediction is 153 with confidence: 708.845%
= Prediction values for test #1
Top(1) prediction is 282 with confidence: 1712.58%
Top(2) prediction is 281 with confidence: 1680.99%
Top(3) prediction is 280 with confidence: 626.642%
Top(4) prediction is 106 with confidence: 600.787%
Top(5) prediction is 66 with confidence: 536.63%
= Prediction values for test #2
Top(1) prediction is 2 with confidence: 2028.62%
Top(2) prediction is 0 with confidence: 590.439%
```

```
Total time for 3 test cases: 0.553 seconds
Average time per test case: 184.185 ms
Overall accuracy: 0.667
```

9 Revision history

The table below summarizes the revisions to this document.

Table 1. Revision history

Revision	Date	Topic cross-reference	Change description
Rev. 1	09/2019	Installing eIQ software	Updated the file path where eIQ software installation script is located.
Rev. 0	09/2019	-	Initial public release.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 09/2019
Document identifier: AN12580

arm