eIQ i.MX Hands-on

v1.0, Apr 22, 2019

Table of Contents

Overview

This document targets the eIQ ML Hands-On training and it is divided in three sections: Arm NN demos, MNIST demos and OpenCV demo.

The Arm NN section shows how to run the prebuilt demo from eIQ to recognize three types of animals. The second part of this section shows a modified demo to extend this for any object detection.

The MNIST section focuses on a comparison of inference time between different models (TensorFlow and Caffe) for handwritten digits recognition.

Finally, the OpenCV section uses the DNN module for inference from a TensorFlow model for object detection from an image. The second part of this section shows a similar demonstration for face/object detection using the camera input.

The procedures described in this document target a Windows host PC.

For the detailed procedures on how to get the source code, prepare the models for inference and build new applications on top of eIQ using ArmNN, check the Running on Linux Appendix chapter. These procedures require a Linux host PC.

Hands-On Kit

Your hands-on kit contains:

- i.MX 8MM EVK board
- USB cable (micro-B to standard-A)
- USB Type-C to A Adapter
- USB Type-C 45W Power Delivery Supply
- IMX-MIPI-HDMI Daughter Card
- MINISASTOCSI Camera Daughter Card
- 2×Mini-SAS cable
- AI/ML BSP flashed into the SDCard
- Ethernet cable
- USB Mouse
- HDMI Cable
- Monitor

Software Requirement

On the Windows PC, install a terminal application like PuTTY for communicating with the board.

Preparing the Board

The following section describes the steps to boot the i.MX 8MM EVK.

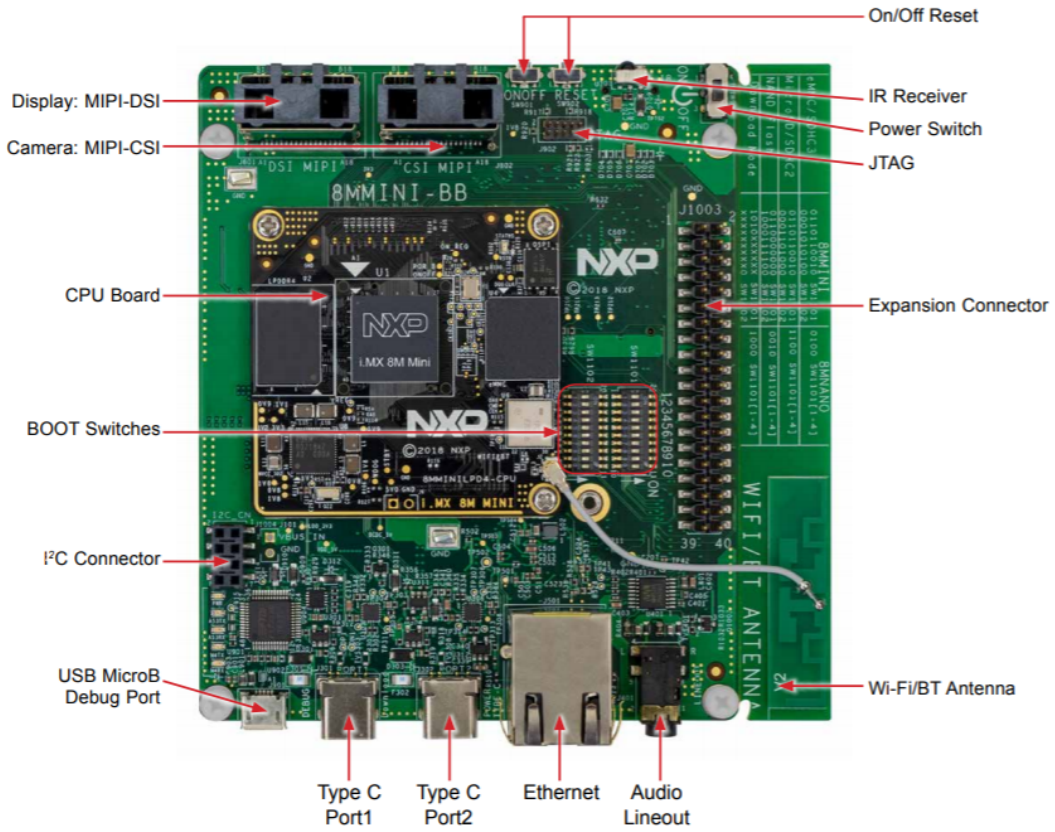

Figure 1. i.MX 8MM EVK board top view

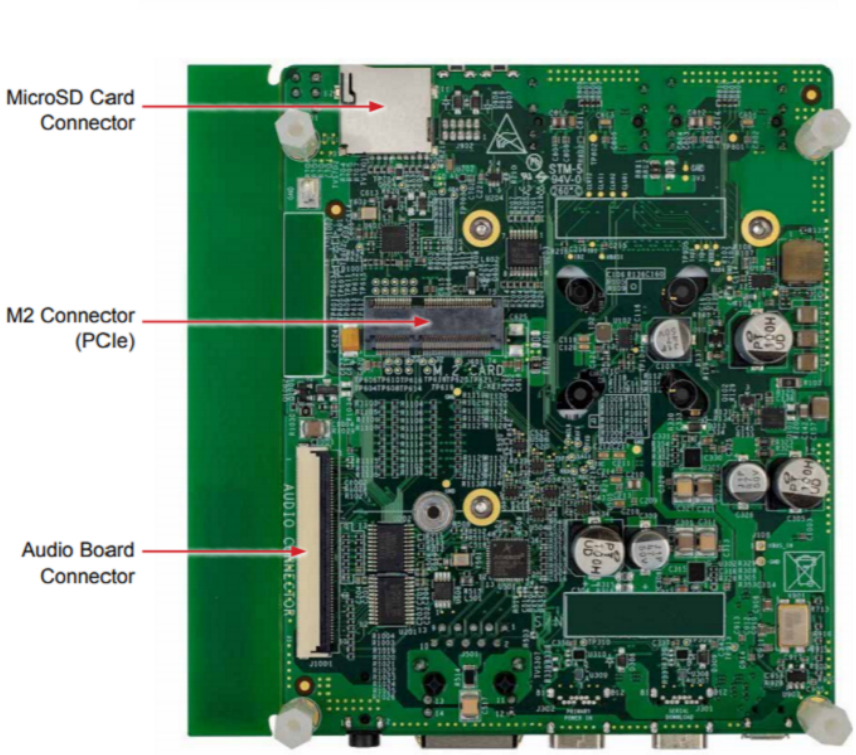

Figure 2. i.MX 8MM EVK board back view

Connect the IMX-MIPI-HDMI daughter card to the Mini-SAS cable and into connector labeled DSI MIPI (J801) and then connect the HDMI monitor cable into it.

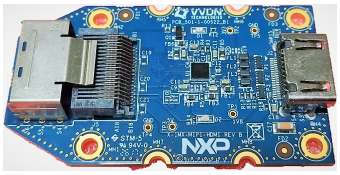

Figure 3. IMX-MIPI-HDMI daughter card

Connect the MINISASTOCSI camera daughter card to the Mini-SAS cable and into connector labeled CSI MIPI (J802).

Figure 4. MINISASTOCSI camera daughter card

Connect the micro-B end of the supplied USB cable into Debug UART port J901. Connect the other end of the cable to the host Windows PC. Configure PuTTY with the board COM port number and set the baud rate to 115200.

NOTE	You can find the board COM port on the Windows Device Manager. The board mounts two COM ports. Take the one with the highest number, which is used to communicate with Cortex-A.

Connect the MicroSD Card to the MicroSD Card Connector J701 in the board back side. In order to Boot the board from the MicroSD Card, change the Boot Switches SW1101 and SW1102 (Figure 5) according to the table below:

Table 1. Boot Device Setting

BOOT Device	SW1101	SW1102
MicroSD/uSDHC2	0110110010	0001101000

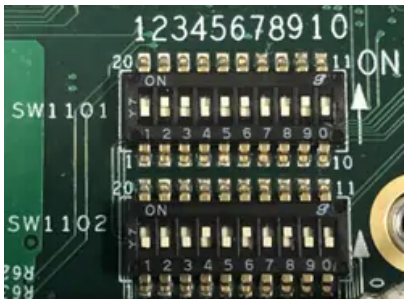

Figure 5. Boot Device Settings

NOTE	The boot device settings above apply to the revision C i.MX 8MM EVK board. Other revisions of the boards may have a different number of boot mode switches and slightly different settings. Please follow the SW1101 and SW1102 values printed on your specific board for booting from the MicroSD Card.

Connect the power supply cable to the power connector `J302` and power on the board by flipping the switch button `SW101`.

NOTE	For more details on the board peripherals, please consult the i.MX 8MM EVK Getting Started.

Preparing the Image

The SD card provided for this training is flashed with an image built following the eIQ User Guide and all the code and needed input files for running each demo presented here.

Some of the demos need write permission to run correctly. After booting, grant the needed permissions by running the following command lines:

```
root@imx8mmevk:# chmod 777 /opt/armnn
root@imx8mmevk:# chmod 777 /opt/mnist
root@imx8mmevk:# chmod 777 /opt/opencv
```

Arm NN Demos

eIQ TensorFlow InceptionV3 Demo

1. At user space, enter the `armnn` folder which holds the demo files:

```
root@imx8mmevk:~# cd /opt/armnn
root@imx8mmevk:/opt/armnn#
```

   This image already includes all needed files to run the eIQ demo. Here is what the `armnn` folders should look like:

```
| ...
├──── data
|    ├──── Cat.jpg
|    ├──── Dog.jpg
|    └──── shark.jpg
├──── model
|    └──── inception_v3_2016_08_28_frozen_transformed.pb
| ...
```

2. Run the demo:

```
root@imx8mmevk:/opt/armnn# TfInceptionV3-Armnn --data-dir=data --model-dir=models

= Prediction values for test #0
Top(1) prediction is 208 with confidence: 93.5791%
Top(2) prediction is 209 with confidence: 2.06653%
Top(3) prediction is 223 with confidence: 0.693557%
Top(4) prediction is 170 with confidence: 0.210818%
Top(5) prediction is 232 with confidence: 0.177887%
= Prediction values for test #1
Top(1) prediction is 283 with confidence: 72.4617%
Top(2) prediction is 282 with confidence: 22.5384%
Top(3) prediction is 286 with confidence: 0.838241%
Top(4) prediction is 288 with confidence: 0.0822042%
Top(5) prediction is 841 with confidence: 0.05987%
= Prediction values for test #2
Top(1) prediction is 3 with confidence: 62.0632%
Top(2) prediction is 4 with confidence: 12.8319%
Top(3) prediction is 5 with confidence: 1.25482%
Top(4) prediction is 154 with confidence: 0.177708%
Top(5) prediction is 149 with confidence: 0.116998%
Total time for 3 test cases: 2.369 seconds
Average time per test case: 789.765 ms
Overall accuracy: 1.000
```

The `TfInceptionV3-Armnn` demo runs the inference on the three expected input images: one containing a dog, one with a cat and one with a shark. The output shows the top 5 inference results and their confidence percentage. The higher the confidence, the better the input image fits the expected content.

There is a chance to get the following result by running the demo:

```
Prediction for test case 0 (x) is incorrect (should be y)
One or more test cases failed
```

NOTE *(x)* refers to the ID of the detected object, *(y)* refers to the ID expected object.

This is **not** an execution error. This occurs because the `TfInceptionV3-Armnn` test expects a specific type of dog, cat and shark to be found so if a different type/breed of these animals is passed to the test, it returns a case failed.

The expected inputs for this test are:

Table 2. Expected inference results

ID	Label	File name
208	Golden Retriever	Dog.jpg
283	Tiger Cat	Cat.jpg
3	White Shark	shark.jpg

The complete list of supported objects can be found here.

Try passing different `.jpg` images to the test, including the expected types as well as other types and see the confidence percentage increasing when you match the expected breeds. Remember to rename the images according to the expect input (`Dog.jpg`, `Cat.jpg`, `shark.jpg`, case sensitive).

To rename a file, use the `mv` command:

```
root@imx8mmevk:/opt/armnn/data# mv <name>.jpg <new_name>.jpg
```

The next section shows how to modify this demo to identify any object.

Modified Demo

This section shows how to use the `TfInceptionV3-Armnn` test from eIQ for general object detection. The list of all object detection supported by this model can be found here.

1. Enter the demo directory and run the demo:

   ```
   root@imx8mmevk:/opt/armnn# python3 2-example.py
   ```

   This runs the `TfInceptionV3-Armnn` test and parses the inference results to return any recognized object, not only the three expected types of animals.

2. Show the provided flash cards to the camera and wait for the detection message: `Image captured, wait`. The flash cards should not be twisted or curved on this step.
3. After a few seconds, the demo returns the detected object.

NOTE This can return `False` if the image was not correctly captured. In this case, try showing the flash card again.

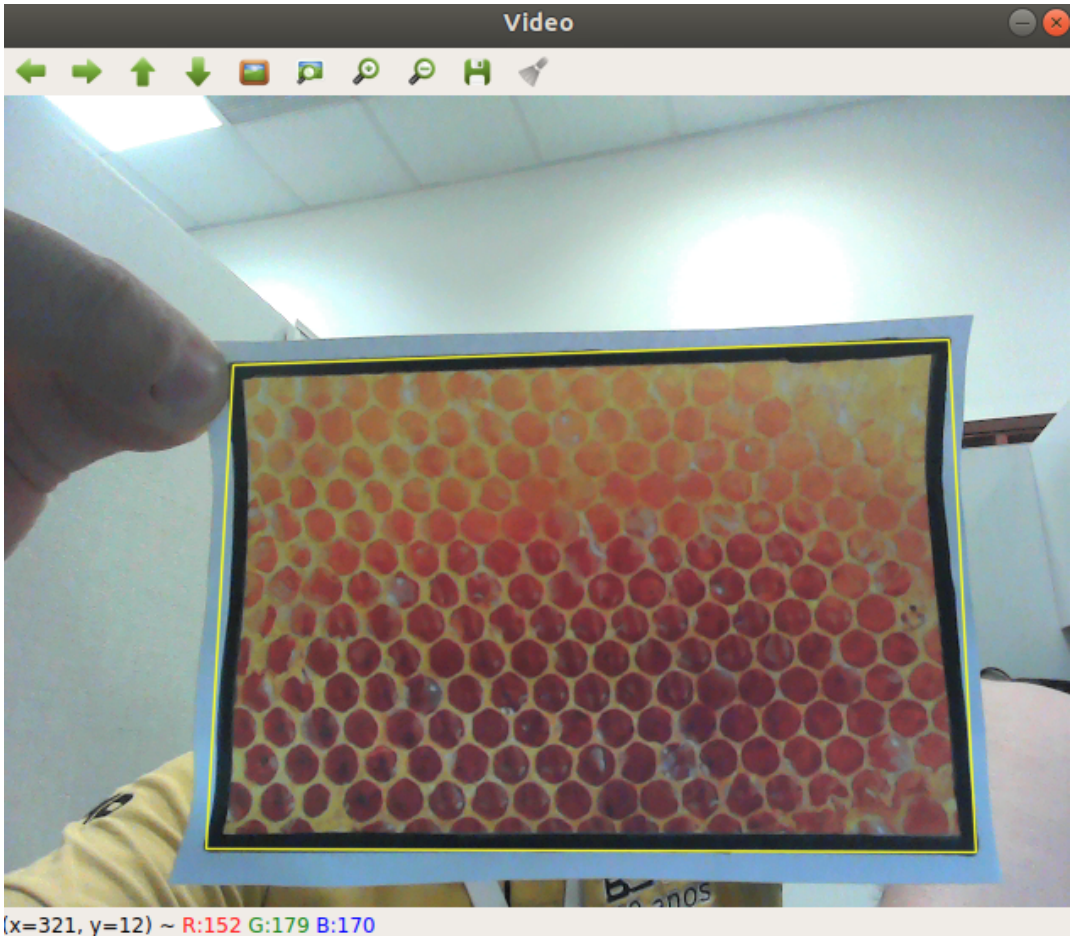

Figure 6. Captured Flash Card

MNIST Handwritten Digits

The `MNIST` is a large database of handwritten digits commonly used for training various image processing systems. This section provides a comparison of a Caffe and TensorFlow models for *Handwritten Digit Recognition*.

The data set used for these applications is from Yann Lecun. Follow a `MNIST` data set sample:

Figure 7. MNIST data set

MNIST Comparison Applications

1. At user space, enter the `mnist` folder which holds the demo files:

   ```
   root@imx8mmevk:/opt/mnist#
   ```

   This is how the `mnist` folder structure should look like:

   ```
   | ...
   ├──── 1-example
   ├──── 2-example
   ├──── 3-example
   ├──── data
   |     ├──── t10k-images-idx3-ubyte
   |     └──── t10k-labels-idx1-ubyte
   └──── model
       ├──── lenet_iter_9000.caffemodel
       ├──── optimized_mnist_tf.pb
       ├──── simple_mnist_tf.pb
       └──── simple_mnist_tf.prototxt
   ```

2. Run the applications:

   NOTE For running these applications, please provide the wanted number of predictions, which can vary from `0` to `9999` since the dataset has 10k images.

a. The `1-example` uses a `Caffe` model for inference:

```
root@imx8mmevk:/opt/mnist# ./1-example 10
[0]     Caffe >> Actual: 7  Predict: 7  Time: 0.0336484s
[1]     Caffe >> Actual: 2  Predict: 2  Time: 0.028399s
[2]     Caffe >> Actual: 1  Predict: 1  Time: 0.0283713s
[3]     Caffe >> Actual: 0  Predict: 0  Time: 0.0284133s
[4]     Caffe >> Actual: 4  Predict: 4  Time: 0.0280637s
[5]     Caffe >> Actual: 1  Predict: 1  Time: 0.0281574s
[6]     Caffe >> Actual: 4  Predict: 4  Time: 0.0285136s
[7]     Caffe >> Actual: 9  Predict: 9  Time: 0.0283779s
[8]     Caffe >> Actual: 5  Predict: 5  Time: 0.0283902s
[9]     Caffe >> Actual: 9  Predict: 9  Time: 0.0283282s
Total Time: 0.296081s   Sucessfull: 10  Failed: 0
```

b. The `2-example` uses a `TensorFlow` model for inference:

```
root@imx8mmevk:/opt/mnist# ./2-example 10
[0]     Tensor >> Actual: 7  Predict: 7  Time: 0.00670075s
[1]     Tensor >> Actual: 2  Predict: 2  Time: 0.00377025s
[2]     Tensor >> Actual: 1  Predict: 1  Time: 0.0036785s
[3]     Tensor >> Actual: 0  Predict: 0  Time: 0.0036815s
[4]     Tensor >> Actual: 4  Predict: 4  Time: 0.00372875s
[5]     Tensor >> Actual: 1  Predict: 1  Time: 0.003669s
[6]     Tensor >> Actual: 4  Predict: 4  Time: 0.00367825s
[7]     Tensor >> Actual: 9  Predict: 9  Time: 0.0036955s
[8]     Tensor >> Actual: 5  Predict: 6  Time: 0.00367488s      FAILED
[9]     Tensor >> Actual: 9  Predict: 9  Time: 0.0036025s
Total Time: 0.0414569s  Sucessfull: 10  Failed: 1
```

NOTE The argument `10` refers to the number of predictions for each test.

These tests run the inference on the input MNIST dataset images (`Actual`), showing the inference results (`Predict`) and how long it took to complete the prediction.

The input images for this test are in the binary form and can be found at the t10k-images-idx3-ubyte.gz package from Yann Lecun.

By the output results, it is possible to notice that the `Caffe` model is slower than `TensorFlow`, but in the other hand it is also more accurate than the latter. Change the argument to compare further results between the two models.

OpenCV Inference

Why OpenCV?

OpenCV offers a unitary solution for both neural network inference (DNN module) and classic machine learning algorithms (ML module). Moreover, it includes many computer vision functions, making it easier to build complex machine learning applications in a short amount of time and without having dependencies on other libraries.

The OpenCV DNN model is basically an inference engine. It does not aim to provide any model training capabilities. For training, one should use dedicated solutions, such as machine learning frameworks. OpenCV's inference engine supports a wide set of input model formats: TensorFlow, Caffe, Torch/PyTorch.

Comparison with Arm NN

Arm NN is a library deeply focused on neural networks. It offers acceleration for Arm Neon, while Vivante GPUs are not currently supported. Arm NN does not support classical non-neural machine learning algorithms.

OpenCV is a more complex library focused on computer vision. Besides image and vision specific algorithms, it offers support for neural network machine learning, but also for traditional non-neural machine learning algorithms. OpenCV is the best choice in case your application needs a neural network inference engine, but also other computer vision functionalities.

Setting Up the Board

In the target device, export the required variables:

```
root@imx8mmevk:~# export LD_LIBRARY_PATH=/usr/local/lib
root@imx8mmevk:~# export PYTHONPATH=/usr/local/lib/python3.5/site-packages/
```

OpenCV Applications

This application was based on SSD: Single Shot MultiBox Detector and Caffe SSD Implementation.

Example Using Caffe with Images

The folder structure must be equal to:

```
$ tree
 ├──── 1-example.py
 ├──── 2-example.py
 ├──── Makefile
 ├──── media
 │     └──── ...
 ├──── model
 │     ├──── MobileNetSSD_deploy.caffemodel
 │     └──── MobileNetSSD_deploy.prototxt
```

This example runs a single picture for example, but you pass as many pictures as you want and save them inside `media/` folder. The application tries to recognize all the objects in the picture.

For copying new images to the `media/` folder:

```
root@imx8mmevk:/opt/opencv/media# cp <path_to_image> .
```

Run the example image:

```
root@imx8mmevk:/opt/opencv# ./1-example.py
```

NOTE If `GPU` is available, the example shows: [INFO:0] Initialize OpenCL runtime

This demo runs the inference using a Caffe model to recognize a few type of objects for all the images inside the `media/` folder. It includes labels for each recognized object in the input images.

The processed images are available in the `media-labeled/` folder. See before and after labeling:

Display the labeled image with the following line:

```
root@imx8mmevk:/opt/opencv/media-labeled# gst-launch-1.0 filesrc location=<image> \
! jpegdec ! imagefreeze ! autovideosink
```

Example Using Caffe with MIPI Camera

This example is the same as above, except that it uses a camera input. It enables the MIPI camera and runs an inference on each captured frame, then displays it in a window interface in real time:

```
root@imx8mmevk:/opt/opencv# ./2-example.py
```

Appendix A: Running on Linux

While this document focus on how to run the demos, this Appendix shows detailed information and steps on where to get the source code and models, how to prepare models for inference, how to build new C++ applications and further information not suitable for Windows OS. This shows the procedure of getting an image built following the eIQ User Guide and enabling all the demonstrations of this training in a fresh new image.

Preparing the Image

1. Download a prebuilt image for i.MX 8MM EVK from here.

NOTE	Prebuilt image with demos for i.MX 8MM EVK from here.

1. Extract the image and flash it to the SD card:

```
$ bunzip2 -f eiq-handson-8mm.sdcard.bz2
$ dd if=eiq-handson-8mm.sdcard of=/dev/sd<x> status=progress && sync
```

NOTE	<*x*> refers to the SD card device. You may need root privileges (sudo) for running the `dd` command.

Setting Up the Board

1. Create the following folders and grant them permission as it follows:

```
root@imx8mmevk:# mkdir -p /opt/armnn/model
root@imx8mmevk:# mkdir -p /opt/armnn/data
root@imx8mmevk:# chmod 777 /opt/armnn
```

```
root@imx8mmevk:~# mkdir -p /opt/mnist
root@imx8mmevk:~# chmod 777 /opt/mnist
```

```
root@imx8mmevk:# mkdir -p /opt/opencv/model
root@imx8mmevk:# mkdir -p /opt/opencv/media
root@imx8mmevk:# chmod 777 /opt/opencv
```

2. To easily deploy the demos to the board, get the boards IP address using `ifconfig` command, then set the `IMX_INET_ADDR` environment variable as it follows:

```
$ export IMX_INET_ADDR=<imx_ip>
```

Arm NN Demos

1. Install TensorFlow on Host PC for preparing the model for inference:

```
$ apt-get install python-pip
$ pip install tensorflow
$ git clone https://github.com/tensorflow/tensorflow.git
```

NOTE You may need root privileges (sudo) for running the `apt-get` command.

2. Generate the graph used to prepare the TensorFlow InceptionV3 model for inference:

```
$ mkdir checkpoints
$ git clone https://github.com/tensorflow/models.git
$ cd models/research/slim/
$ python export_inference_graph.py --model_name=inception_v3 \
--output_file=../../../checkpoints/inception_v3_inf_graph.pb
```

3. Download the pretrained model and prepare it for inference with the generated graph:

```
$ cd ../../../checkpoints
$ wget http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz \
-qO- | tar -xvz # download pretrained model
$ python <path_to_tensorflow_repo>/tensorflow/python/tools/freeze_graph.py \
--input_graph=inception_v3_inf_graph.pb --input_checkpoint=inception_v3.ckpt \
--input_binary=true --output_graph=inception_v3_2016_08_28_frozen_transformed.pb \
--output_node_names=InceptionV3/Predictions/Reshape_1
```

NOTE *<path_to_tensorflow_repo>* refers to the cloned TensorFlow path from `Step 1`.

4. Copy the prepared model `inception_v3_2016_08_28_frozen_transformed.pb` to `/opt/armnn/models`:

```
$ scp inception_v3_2016_08_28_frozen_transformed.pb root@<imx_ip>:/opt/armnn/model
```

5. Find three `.jpg` images on `Google`, one containing a dog, one with a cat and one with a shark. Rename them to `Dog.jpg`, `Cat.jpg` and `shark.jpg` accordingly (case sensitive) and copy them to the `/opt/armnn/data` folder on the device.

```
$ scp Dog.jpg Cat.jpg shark.jpg root@<imx_ip>:/opt/armnn/data
```

6. Run the demo following section eIQ TensorFlow InceptionV3 Demo.

NOTE For the modified demo, download this application and put it in `/opt/armnn`.

MNIST Demos

Setting Up the Host

Download this toolchain for *cross-compiling* the applications:

```
$ chmod +x fsl-imx-xwayland-glibc-x86_64-fsl-image-gui-aarch64-toolchain-4.14-sumo.sh
$ ./fsl-imx-xwayland-glibc-x86_64-fsl-image-gui-aarch64-toolchain-4.14-sumo.sh
```

This provides all needed setup for building ARM64 applications on a X86 machine.

1. Download this application.
2. Get the models and dataset. The following command-line creates the needed folder structure for the demos and retrieves the `mnist` dataset and the `Caffe` and `TensorFlow` models:

```
$ mkdir -p bin data model
$ wget -qN https://github.com/ARM-software/ML-examples/raw/master/armnn-
mnist/data/t10k-images-idx3-ubyte -P data/
$ wget -qN https://github.com/ARM-software/ML-examples/raw/master/armnn-
mnist/data/t10k-labels-idx1-ubyte -P data/
$ wget -qN https://github.com/ARM-software/ML-examples/raw/master/armnn-
mnist/model/lenet_iter_9000.caffemodel -P model/
$ wget -qN https://github.com/ARM-software/ML-examples/raw/master/armnn-
mnist/model/simple_mnist_tf.pb -P model/
$ wget -qN https://github.com/ARM-software/ML-examples/raw/master/armnn-
mnist/model/simple_mnist_tf.prototxt -P model/
$ wget -qN https://github.com/ARM-software/Tool-Solutions/raw/master/ml-tool-
examples/mnist-draw/model/optimized_mnist_tf.pb -P model/
```

3. Compile the source code using the downloaded toolchain:

```
$ source /opt/fsl-imx-xwayland/4.14-sumo/environment-setup-aarch64-poky-linux
$ ${CXX} -Wall -Wextra -O3 -std=c++14 1-example.cpp -o 1-example -larmnn
-larmnnCaffeParser
$ ${CXX} -Wall -Wextra -O3 -std=c++14 2-example.cpp -o 2-example -larmnn
-larmnnTfParser
$ ${CXX} -Wall -Wextra -O3 -std=c++14 3-example.cpp -o 3-example -larmnn
-larmnnTfParser
```

4. Deploy the built files to the board:

```
$ scp -r 1-example 2-example 3-example data/ model/
root@${IMX_INET_ADDR}:/opt/mnist
```

5. Run the demo following section MNIST Comparison Applications.

OpenCV Demos

Setting Up the Host

1. Download this application.
2. Get the models and dataset. The following command-line creates the needed folder structure for the demos and retrieves all needed data and model files for the demo:

```
$ mkdir -p model
$ wget -qN
https://github.com/diegohdorta/models/raw/master/caffe/MobileNetSSD_deploy.caffemod
el -P model/
$ wget -qN
https://github.com/diegohdorta/models/raw/master/caffe/MobileNetSSD_deploy.prototxt
-P model/
```

3. Deploy the built files to the board:

```
$ scp -r 1-example.py 2-example.py model/ root@${IMX_INET_ADDR}:/opt/opencv
```

4. Run the demo following section Example Using Caffe with Images.