Machine Learning User's Guide

Version 0.1 | 2021-07-05

Table of Contents

Chapter 1. Introduction

This document walks through the Artificial Intelligence (AI) technology and explains how to start developing applications to run on the embedded systems. However, to get there we need to understand what is Machine Learning (ML) and the reason this field is gaining space in several industries, especially, in the embedded world.

1.1. Machine Learning Overview

If one starts thinking about ML its mind probably deliberates about robots and talking machines; some might even relate this technology to robots bent on destroying the universe. Not sure about the future, but nowadays AI/ML is more about building intelligent systems with decision-making abilities. ML is a subfield of AI that can analyze large amounts of data, and with that draw knowledge from previous experience. Keep in mind that ML exists solely as software, but for most cases, it requires the use of hardware components to build intelligent machines. ML combined with embedded systems can reach significant improvements and a certain level of smartness.

1.2. How Does Machine Learning Work?

This field boils down to spending time writing many lines of code that eventually solve a problem by applying some type of intelligent algorithm. For instance, some smart homes have lighting systems that automatically turn on and off based on whether anyone is present in the room. This idea does not amuse, but the system is actually making decisions on its own.

A decision is a choice made under a certain circumstance, taking some information into account. Think about the example above, the lighting system. How does the system decide to turn the lights on and off? A motion sensor could be used to detect if there is someone in the room, and it would be translated to a boolean variable in a computer program that controls the lights. The daylight could also be taken into consideration. A light sensor could be installed and if the room is already lightened by daylight, then there is no need to turn the lights on. This sensor would now become another variable in the program.

As more variables, you have to analyze, the more complex the program gets. What if there are pets in this house, but the lights should be turned on only if there is a person in the room? If the pet enters the room, the motion sensor would trigger anyway, so another solution needs to be found. There could be a camera and the program would analyze the image to detect if there is a person there. To accomplish it, a set of rules would need to be written in the program and it would start to become really complex compared to the previous one. That works fine for simple images, but as deeper it dives into the problem, the messier it gets and the written rules start to break down.

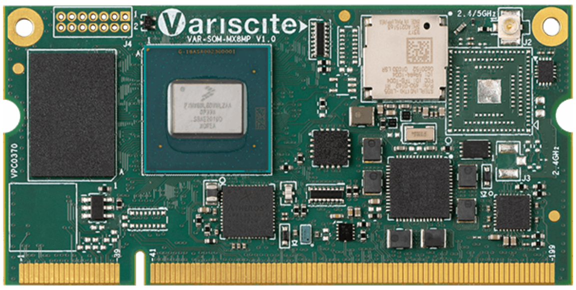
Imagine if the system now has to take a dog into account, but not a cat. A new set of rules needs to be written, and how simple rules can properly handle this task properly? In fact, for just about any rule it is possible to find a situation where it will not work for sure. After spending many hours it can almost get there, but that’s just to tell the difference between the two kinds of animals. The problem begins with any new problem, where it is necessary to start all over again and write more rules for the new issue. That’s when ML thrives. ML is more about developing algorithms that can learn from examples and experiences instead of relying on hard-coded rules.

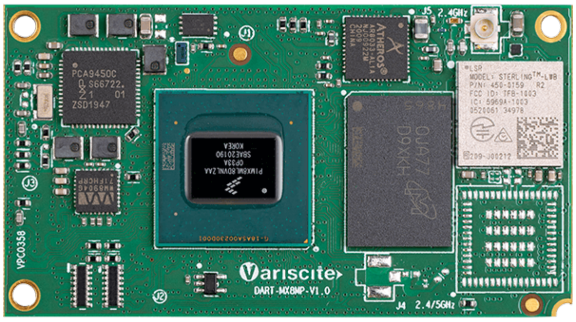
1.3. Where Machine Learning Can Be Used?

Known as The Fourth Industrial Revolution or 4.0 Industry, which in fact is happening right now, this concept consists of a smart factory where production processes are connected, such as machines, interfaces, and components communicating each other. Large amounts of data can be collected to optimize the manufacturing process, and AI expected to influence the growth of these companies by using the ML field. Fully automated production lines, quality control, security systems are only a little portion of where ML can be applied on the industries. As the amount of data increases, the system becomes better at optimizing itself and making more accurate predictions. Allied to IoT devices and 5G connection, ML certainly is the future, not only in the industries but also in home automation and in the everyday tasks of people around the world.

Chapter 2. Machine Learning Dedicated Modules

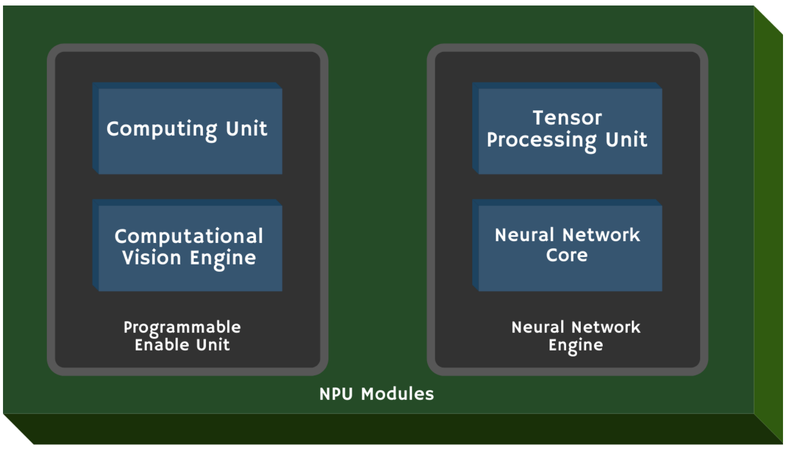
Powered by the i.MX 8M Plus, the VAR-SOM-MX8M-PLUS, and DART-MX8M-PLUS present the first generation of System on Modules with dedicated AI/ML capabilities due to NPU (Neural Processing Unit). See more details:

VAR-SOM-MX8M-PLUS	Main Specifications
	**CPU**: NXP i.MX 8M Plus **Memory**: Up to 4GB LPDDR4-4000 **GPU**: Vivante™ GC7000UL 3D and GC520L 2D **NPU**: Verisilicon VIP 8000 (up to 2.3 TOPS). For more information, visit Variscite's VAR-SOM-MX8M-PLUS Software Wiki and Product Page.

DART-MX8M-PLUS	Main Specifications
	**CPU**: NXP i.MX 8M Plus **Memory**: Up to 4GB LPDDR4-3000 **GPU**: Vivante™ GC7000UL 3D and GC520L 2D **NPU**: Verisilicon VIP 8000 (up to 2.3 TOPS). For more information, visit Variscite's DART-MX8M-PLUS Software Wiki and Product Page.

2.1. Neural Processing Unit (NPU)

The NPU is a dedicated processing unit to optimize the inference performance on AI/ML applications, leveraging the other cores such as CPU and GPU. See more details:



2.1.1. Programmable Engine Unit

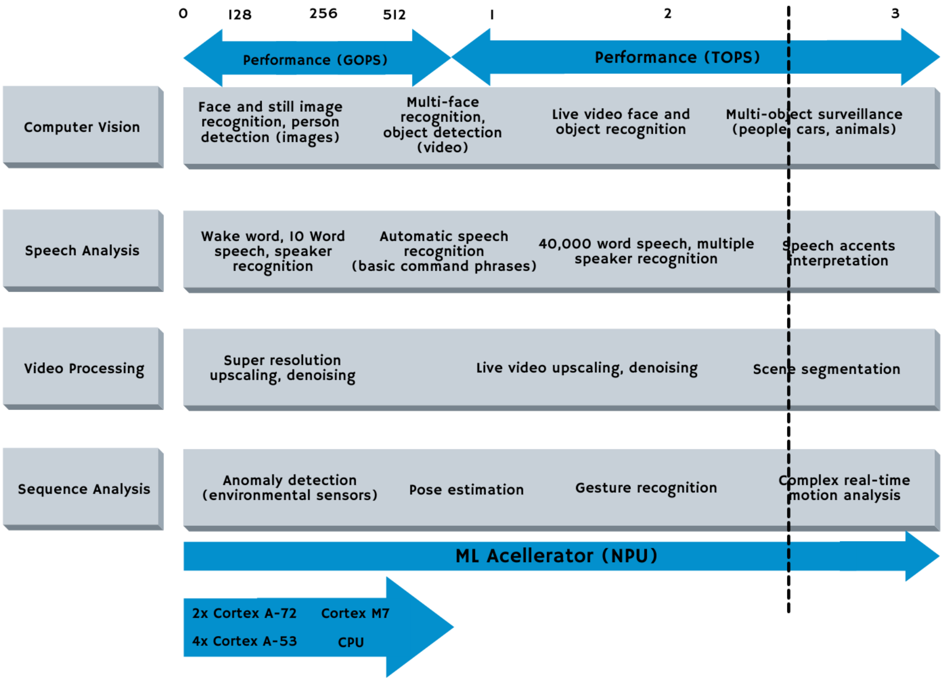
- **Computing Unit**: Flexible programming unit to handle 8/16/32b integer and 16/32b floating-point operations;
- **Computational Vision Unit**: Provides advanced image processing functions.

2.1.2. Neural Network Engine

- **Tensor Processing Core**: 8/16b integer and 16b floating-point tensor operations such as non-convolution layers, multi-lane processing for data shuffling, normalization, pooling, and more;
- **Neural Network Core**: 8b integer tensor operations such as convolution layer, maximum pooling, and more.

2.2. Machine Learning Applications Use Cases

Machine Learning can be used to solve a series of cases from simple problems, such as image/video recognition and detection, pose and gesture estimation to more complex ones. For instance problems as scene segmentation or multi-object surveillance. See more details:



The simpler the problem, the less processing is needed but as the problem gets more complex more processing is required, and the NPU is designed to offer a high-performance solution for these problems.

Real-world problems requires not just processing power it also needs inputs from other peripherals. The VAR-SOM-MX8M-PLUS and DART-MX8M-PLUS are designed to support audio and cameras that combined with the i.MX 8M Plus can achieve great results on ML applications.

Chapter 3. eIQ™ ML Software from NXP

The Variscite Yocto Linux BSP includes the NXP eIQ™ ML Software Development layer, which provides a set of libraries and development tools for machine learning applications targeting the **VAR-SOM-MX8M-PLUS** and the **DART-MX8M-PLUS** modules. The inference engines currently supported in the software stack are the *ArmNN*, *TensorFlow Lite*, *ONNX Runtime*, *PyTorch*, and *OpenCV*.

3.1. eIQ™ Inference Engines and Libraries Overview

3.1.1. TensorFlow Lite

TensorFlow Lite is an open-source software library focused on running machine learning models on embedded devices providing *low latency inference* and *small binary size*. See the main features:

Currently Version	CPU Inference	GPU Inference	NPU Inference	API Support
2.4.0	✔	✔	✔	C++/Python3

- Examples are available in the TensorFlow Lite Examples section.

3.1.2. Arm Compute Library

Arm Compute Library is a compute engine for the *Arm NN* framework that provides a collection of low-level optimized functions for Arm CPU and GPU architectures. See the main features:

Currently Version	CPU Inference	GPU Inference	NPU Inference	API Support
20.08	✔	✔	-	C++

- Examples are available in the Arm Compute Library Examples section.

3.1.3. Arm NN

Arm NN is an open-source inference engine framework that instead of performing computations, delegates the input from multiple model formats such as *Caffe*, *TensorFlow Lite*, or *ONNX* to the according compute engines. See the main features:

Currently Version	CPU Inference	GPU Inference	NPU Inference	API Support
20.08	✔	✔	✔	C++/Python3

- Examples are available in the Arm NN Examples section.

3.1.4. ONNX Runtime

ONNX Runtime is an open-source inference engine framework that enables the acceleration of machine learning models across all the targets using a single set of API. See the main features:

Currently Version	CPU Inference	GPU Inference	NPU Inference	API Support
1.5.3	✔	✔	✔	C++/Python3

- Examples are available in the ONNX Runtime Examples section.

3.1.5. PyTorch

PyTorch is a scientific computing package based on Python that facilitates building deep learning projects using power of graphics processing units. See the main features:

Currently Version	CPU Inference	GPU Inference	NPU Inference	API Support
1.7.1	✔	-	-	Python3

- Examples are available in the PyTorch Examples section.

3.1.6. OpenCV

OpenCV is an open-source computer vision library, and it provides machine learning algorithms that offer solutions for neural network inferece process. See the main features:

Currently Version	CPU Inference	GPU Inference	NPU Inference	API Support
4.4.0	✔	-	-	C++/Python3

- Examples are available in the OpenCV Examples section.

3.2. Build Yocto with the eIQ™ Enablements

1. Retrieve the latest version (Zeus):

```
$ repo init -u https://github.com/varigit/variscite-bsp-platform.git \
            -b fsl-zeus \
            -m imx-5.4.70-2.3.2-var01.xml
$ repo sync -j$(nproc) ①
```

① The *repo sync* step may take a while to finish ☕

   a. Prepare the environment to build the image for the DART-MX8M-PLUS:

   ```
   $ MACHINE=imx8mp-var-dart DISTRO=fsl-imx-xwayland . var-setup-release.sh -b build_xwayland
   ```

   b. Build the XWayland full image:

   ```
   $ bitbake imx-image-full ②
   ```

② This step may take several hours to complete depending on your computer's specifications ☕

2. Flash the Full Image to the SD Card:

```
$ cd ~/var-fslc-yocto/build/tmp/deploy/images/imx8mm-var-dart/
$ zcat imx-image-full-imx8mp-var-dart.wic.gz \
  | sudo dd of=/dev/sd<x> bs=1M status=progress conv=fsync ①
```

> ① Make sure to use the correct path to the SD card instead of */dev/sd<x>* (**e.g., /dev/sdb**). To check it, use the *lsblk* command.
>
> **BE CAREFUL** not to use the path to the primary disk - it can overwrite the host OS.

3. After these steps, insert the SD card into the custom board and power on to boot the custom BSP.

Chapter 4. eIQ™ ML Default Applications

This section explains how to run the default examples built from the machine learning layer on Yocto Linux BSP.

4.1. TensorFlow Lite Examples

Table 1. TensorFlow Lite

Example Name	Language	Location Folder	Default Model	Default Data
label_image	*C++/Python*	*/usr/bin/tensorflow-lite-2.4.0/examples*	*mobilenet_v1_1.0_224 _quant.tflite*	*grace_hopper.bmp*

4.1.1. Image Classification Example in C++

1. Enter in the following folder;

   /usr/bin/tensorflow-lite-2.4.0/examples

   a. Execute the label image example running the inference on **CPU**:

   ```
   $ ./label_image -a 0
   ```

   i. The output of a successful classification should be similar as the one below:

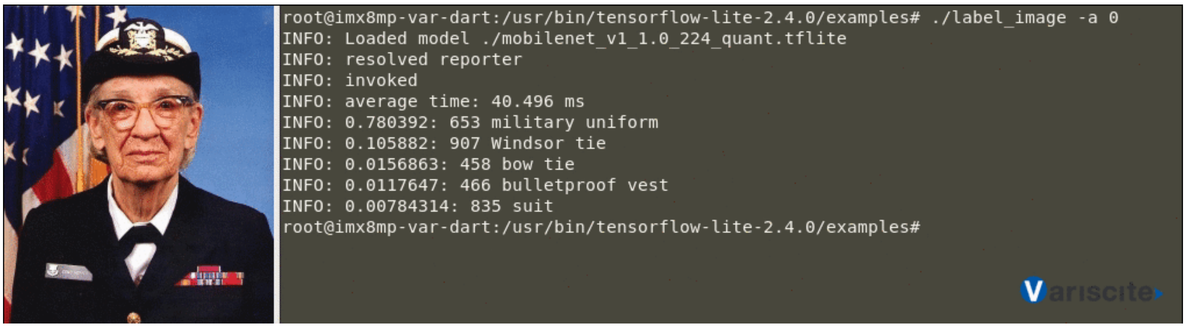


Figure 1. TensorFlow Lite Image Classification Example Input (CPU Inference)

Inference Time on CPU: **40.496 milliseconds**.

   b. Execute the label image example running the inference on **NPU**:

   ```
   $ ./label_image -a 1
   ```

   i. The output of a successful classification should be similar as the one below:

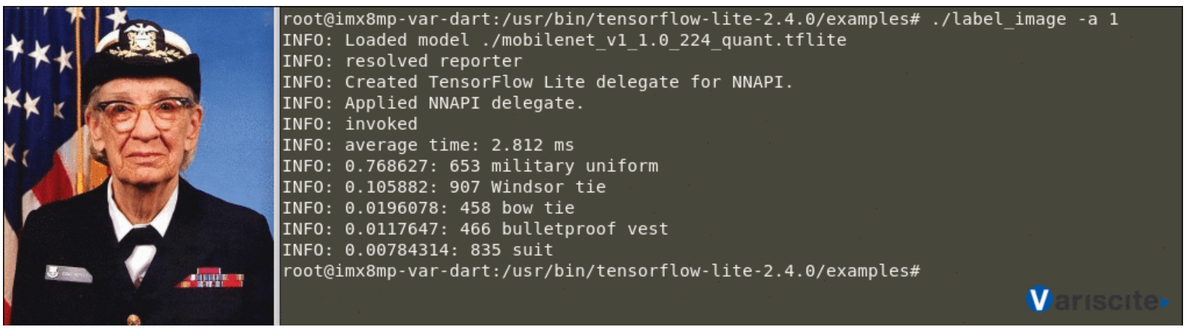


Figure 2. TensorFlow Lite Image Classification Example Input (NPU Inference)

Inference Time on NPU: **2.812 milliseconds**.

c. To run the same example using a different input image, use the **--image** argument:

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i <file_name.extension> -l labels.txt
```

4.1.2. Image Classification Example in Python

1. Enter in the following folder;

   /usr/bin/tensorflow-lite-2.4.0/examples

   a. Execute the label image example running the inference on **NPU**:

   ```
   $ python3 label_image.py
   ```

   i. The output of a successful classification should be similar as the one below:

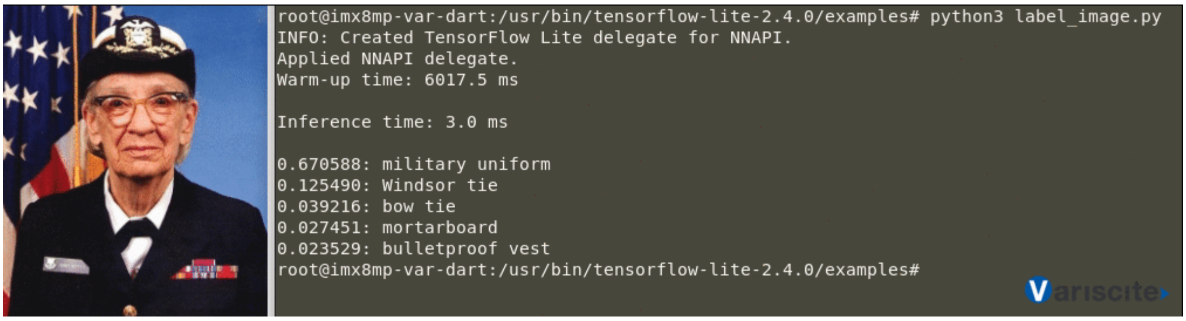


Figure 3. TensorFlow Lite Image Classification Example Input (NPU Inference)

Inference Time on NPU: **3.0 milliseconds**.

   b. To run the same example using a different input image, use the **--image** argument:

   ```
   $ python3 label_image.py -m mobilenet_v1_1.0_224_quant.tflite -i <file_name.extension> -l labels.txt
   ```

> Tensorflow Lite does not provide parameters to choose the inference core. By default, the Python examples run inference only in the NPU core.

4.2. Arm Compute Library Examples

Table 2. Arm Compute Library

Example Name	Language	Location Folder	Default Model	Default Data
graph_alexnet	*C++*	*/usr/bin/arm-compute-library-20.08/examples*	*alexnet_model*	*go_kart.ppm*

1. Enter in the following folder;

   /usr/bin/arm-compute-library-20.08/examples

   a. Download the archive file *alexnet_model* from the table *Arm Compute Library* in the *examples* folder;

      i. Create a new sub-folder and unzip the archive file:

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

2. Set environment variables for execution:

```
$ export PATH_ASSETS=/usr/bin/arm-compute-library-20.08/examples/assets_alexnet
```

   a. Run the example with following command line arguments:

```
$ ./graph_alexnet --data=$PATH_ASSETS --image=$PATH_ASSETS/go_kart.ppm \
                  --labels=$PATH_ASSETS/labels.txt --target=neon --type=f32 --threads=4
```

      i. The output of a successful classification should be similar as the one below:

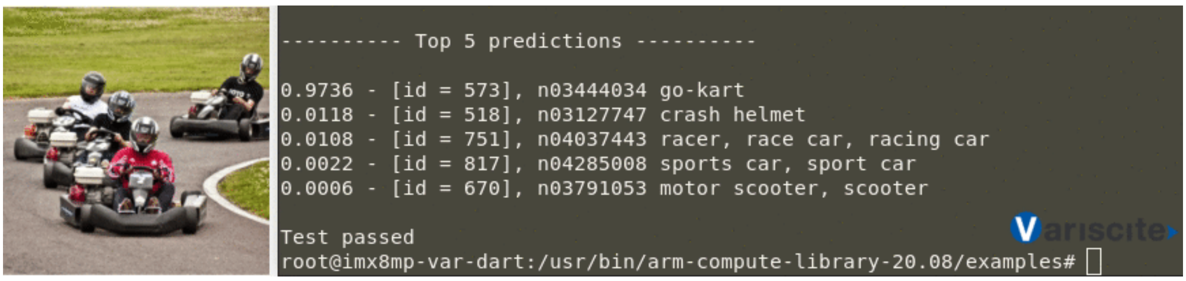


Figure 4. Arm NN Graph AlexNet Example Input (CPU Inference)

4.3. Arm NN Examples

The examples from this section assume that the neural network model files are stored in a folder called models, and the input image files are store in a folder called data.

1. Create theses folders using the following command lines:

```
$ cd /usr/bin/armnn-20.08
$ mkdir data
$ mkdir models
```

4.3.1. Caffe Tests

Table 3. Caffe Tests

Example Name	Language	Location Folder	Default Model	Default Data
CaffeAlexNet-Armnn	*C++*	*/usr/bin/armnn-20.08*	deploy.prototxt	*shark.jpg*
CaffeMnist-Armnn	*C++*	*/usr/bin/armnn-20.08*	lenet_iter_9000.caffemodel	*t10k-images-idx3-ubyte.gz*; *t10k-labels-idx1-ubyte.gz*.

1. Under construction; need to change the batch size.

4.3.2. TensorFlow Tests

Table 4. TensorFlow Tests

Example Name	Language	Location Folder	Default Model	Default Data
TfMnist-Armnn	*C++*	*/usr/bin/armnn-20.08*	*simple_mnist_tf.prototxt*	*t10k-images-idx3-ubyte.gz*; *t10k-labels-idx1-ubyte.gz*.
TfMobileNet-Armnn	*C++*	*/usr/bin/armnn-20.08*	*mobilenet_v1_1.0_224.tgz*	*shark.jpg*; Dog.jpg; Cat.jpg.

TfMnist-Armnn

1. Download the *simple_mnist_tf.prototxt* from the table *TensorFlow Tests* in the *models* folder:

   /usr/bin/armnn-20.08/models

2. Download the *t10k-images-idx3-ubyte.gz* and *t10k-labels-idx1-ubyte.gz* from the table *TensorFlow Tests* in the *data* folder:

   /usr/bin/armnn-20.08/data

3. Run the example with following command line arguments:

```
$ ./TfMnist-Armnn --data-dir=data --model-dir=models
```

TfMobileNet-Armnn

1. Download the *mobilenet_v1_1.0_224.tgz* from the table *TensorFlow Tests* in the *models* folder: (**need to unpack first**)

   /usr/bin/armnn-20.08/models

2. Download the three images from the table *TensorFlow Tests*;
   a. Rename them as *shark.jpg*, Dog.jpg, and Cat.jpg (case sensitive);
   b. Then, copy them to the *data* folder:

      /usr/bin/armnn-20.08/data

3. Run the example with following command line arguments:

```
$ ./TfMobileNet-Armnn  --data-dir=data --model-dir=models
```

4.3.3. TensorFlow Lite Tests

Table 5. TensorFlow Lite Tests

Example Name	Language	Location Folder	Default Model	Default Data
TfLiteMnasNet-Armnn	*C++*	*/usr/bin/armnn-20.08*	*mnasnet_1.3_224_09_07_2018.tgz*	*shark.jpg*; Dog.jpg; Cat.jpg.
TfLiteMobilenetQuantized-Armnn	*C++*	*/usr/bin/armnn-20.08*	*mobilenet_v1_1.0_224_quant.tgz*	*shark.jpg*; Dog.jpg; Cat.jpg.

TfLiteMnasNet-Armnn

1. Download the *mnasnet_1.3_224_09_07_2018.tgz* from the table *TensorFlow Lite Tests* in the *models* folder: (**need to unpack first**)

   /usr/bin/armnn-20.08/models

2. Download the three images from the table *TensorFlow Lite Tests*;
   a. Rename them as *shark.jpg*, Dog.jpg, and Cat.jpg (case sensitive);
   b. Then, copy them to the *data* folder:

      /usr/bin/armnn-20.08/data

3. Run the example with following command line arguments:

```
$ ./TfLiteMnasNet-Armnn --data-dir=data --model-dir=models
```

TfLiteMobilenetQuantized-Armnn

1. Download the *mobilenet_v1_1.0_224_quant.tgz* from the table *TensorFlow Lite Tests* in the *models* folder: (**need to unpack first**)

   /usr/bin/armnn-20.08/models

2. Download the three images from the table *TensorFlow Lite Tests*;
   a. Rename them as *shark.jpg*, Dog.jpg, and Cat.jpg (case sensitive);
   b. Then, copy them to the *data* folder:

      /usr/bin/armnn-20.08/data

3. Run the example with following command line arguments:

```
$ ./TfLiteMobilenetQuantized-Armnn --data-dir=data --model-dir=models
```

4.3.4. ONNX Tests

Table 6. ONNX Tests

Example Name	Language	Location Folder	Default Model	Default Data
OnnxMnist-Armnn	*C++*	*/usr/bin/armnn-20.08*	*mnist.tar.gz*	*t10k-images-idx3-ubyte.gz*; *t10k-labels-idx1-ubyte.gz*.
OnnxMobileNet-Armnn	*C++*	*/usr/bin/armnn-20.08*	*mobilenetv2-1.0.tar.gz*	*shark.jpg*; Dog.jpg; Cat.jpg.

OnnxMnist-Armnn

1. Download the *mnist.tar.gz* from the table *ONNX Tests* in the *models* folder:

   /usr/bin/armnn-20.08/models

2. Download the *t10k-images-idx3-ubyte.gz* and *t10k-labels-idx1-ubyte.gz* from the table *TensorFlow Tests* in the *data* folder:

   /usr/bin/armnn-20.08/data

3. Run the example with following command line arguments:

```
$ ./OnnxMnist-Armnn --data-dir=data --model-dir=models
```

OnnxMobileNet-Armnn

1. Download the *mobilenetv2-1.0.tar.gz* from the table *ONNX Tests* in the *models* folder: (**need to unpack first**)

   /usr/bin/armnn-20.08/models

2. Download the three images from the table *TensorFlow Lite Tests*;
   a. Rename them as *shark.jpg*, Dog.jpg, and Cat.jpg (case sensitive);
   b. Then, copy them to the *data* folder:

      /usr/bin/armnn-20.08/data

3. Run the example with following command line arguments:

```
$ ./OnnxMobileNet-Armnn --data-dir=data --model-dir=models
```

4.4. ONNX Runtime Examples

The following models from ONNX Zoo that were tested in this release: *MobileNet v2*, *ResNet50 v2*, *ResNet50 v1*, *SSD Mobilenet v1*, and *Yolo v3*.

Table 7. ONNX Runtime

Example Name	Language	Location Folder	Default Model	Default Data
onnx_test_runner	*C++*	*/usr/bin*	mobilenetv2-7.tar.gz	-

4.4.1. MobileNet V2

1. Download and unpack the *mobilenetv2-7.tar.gz* from the table *ONNX Runtime* in *root* folder:

   /home/root

   a. Run the example with following command line arguments:

```
$ onnx_test_runner -j 1 -c 1 -r 1 -e [armnn/acl/vsi_npu] /home/root/mobilenetv2-1.0/
```

4.5. PyTorch Examples

4.5.1. Image Classification Example

4.6. OpenCV Examples

Chapter 5. NPU Warm Up Time

Chapter 6. Machine Learning Developing

Teach how to develop a new demo using the enablements from NXP (tensorflow, pyarmnn, pytorch, onnx, etc).

Python: SSD demo using coco data set. Gets input from file, camera and video. It will be saved on var-demos.