NXP Semiconductors
User's Guide

Document identifier: IMXMLUG
Rev. L5.4.70_2.3.0, 31 December 2020

i.MX Machine Learning User's Guide

Contents

Chapter 1
Software Stack Introduction

The NXP® eIQ™ for i.MX toolkit provides a set of libraries and development tools for machine learning applications targeting NXP microcontrollers and application processors. The NXP eIQ toolkit is contained in the *meta-imx/meta-ml* layer.

These five inference engines are currently supported in the NXP eIQ software stack: ArmNN, TensorFlow Lite, ONNX Runtime, PyTorch, and OpenCV. The following figure describes the supported eIQ inference engines accross the computing units.

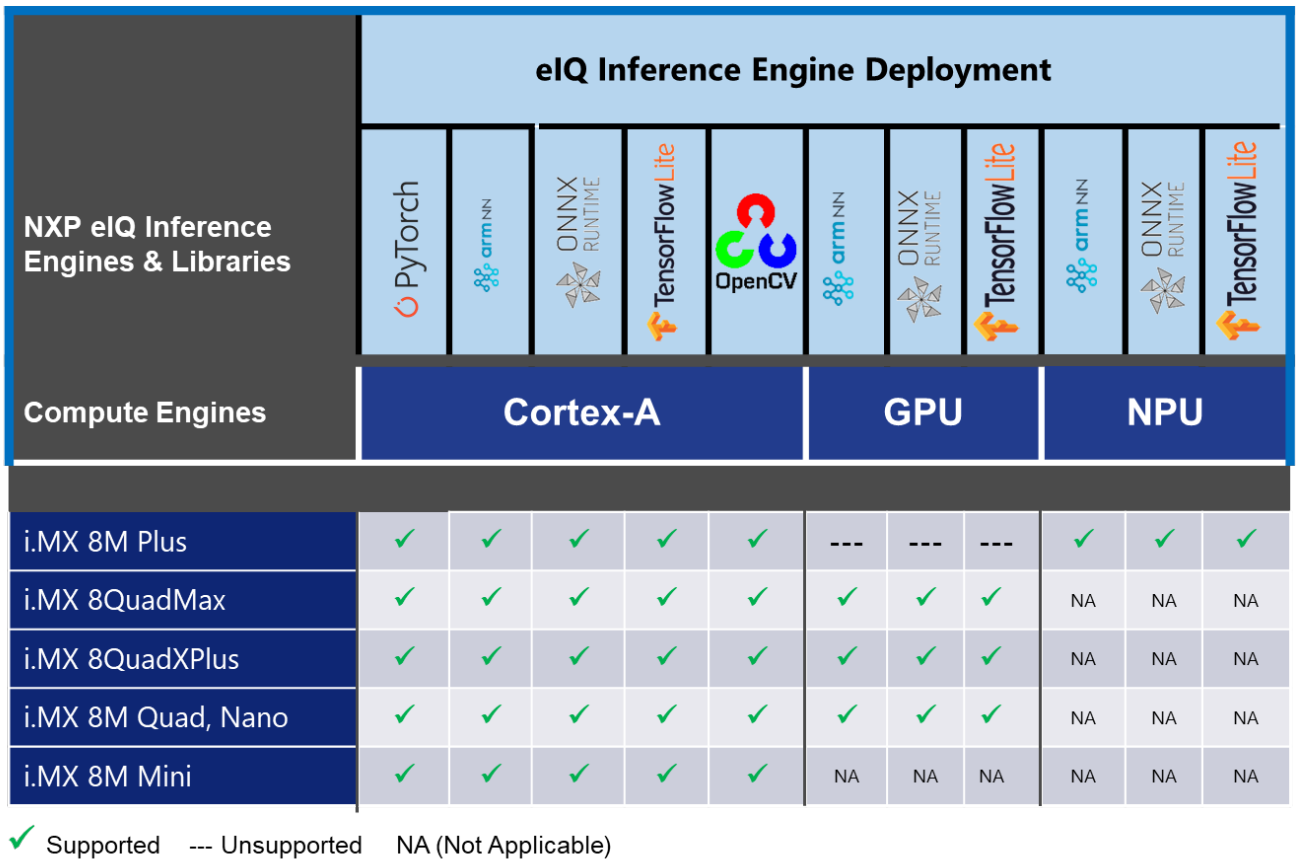

Figure 1. NXP eIQ supported compute vs. inference enginees

The NXP eIQ inference engines support multi-threaded execution on Cortex-A cores. Additionally, ArmNN, ONNX Runtime, and Tensorflow Lite also support acceleration on the GPU or NPU through Neural Network Runtime (NNRT). See also Neural Network Runtime Overview.

The NXP eIQ toolkit supports these key application domains:

- **Vision**
  - Multi camera observation
  - Active object recognition
  - Gesture control
- **Voice**
  - Voice processing

  - Home entertainment
- **Sound**
  - Smart sense and control
  - Visual inspection
  - Sound monitoring

Chapter 2
Neural Network Runtime Overview

The chapter describes an overview of the NXP eIQ software stack for use with the NXP Neural Network Accelerator IPs (GPU or NPU). The following figure shows the data flow between each element. The key part of this diagram is the Neural Network Runtime (NNRT), which is a middleware bridging various inference frameworks and the NN accelerator driver. The NNRT supplies different backends for Android NN HAL, Arm NN, ONNX, and TensorFlow Lite allowing quick application deployment. The NNRT also empowers an application-oriented framework for use with i.MX8 processors. Application frameworks such as Android NN, TensorFlow Lite, and Arm NN can be speed up by NNRT directly benefiting from its built-in backend plugins. Additional backend can be also implemented to expand support for other frameworks.

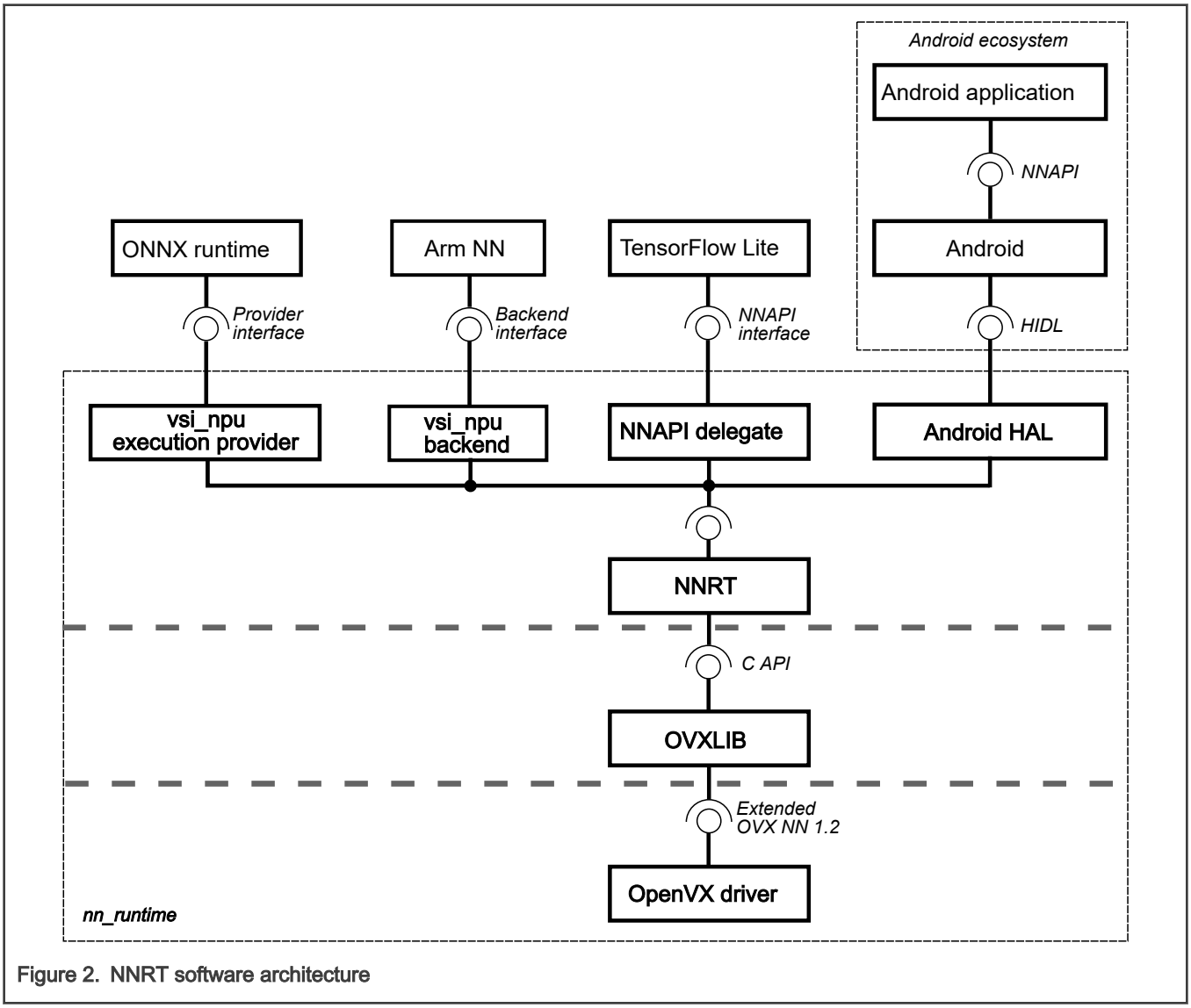

Figure 2. NNRT software architecture

NNRT supports different Machine Learning frameworks by registering itself as a compute backend. Because each framework defines a different backend API, a lightweight backend layer is designed for each:

- For Android NN, the NNRT follows the Android HIDL definition. It is compatible with v1.2 HAL interface
- For TensorFlow Lite, the NNRT supports NNAPI Delegate. It supports most operations in Android NNAPI v1.2
- For ArmNN, the NNRT registers itself as a compute backend

- For ONNX Runtime, the NNRT registers itself as an execution provider

In doing so, NNRT unifies application framework differences and provides an universal runtime interface into the driver stack. At the same time, NNRT also acts as the heterogeneous compute platform for further distributing workloads efficiently across i.MX8 compute devices, such as NPU, GPU and CPU.

NOTE

Both the OpenCV and PyTorch inference enginees are currently not supported for running on the NXP NN accelerators. Therefore, both frameworks are not included in the above NXP-NN architecture diagram.

Chapter 3
TensorFlow Lite

TensorFlow Lite is an open-source software library focused on running machine learning models on mobile and embedded devices (available at http://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 Networks API (NNAPI) on various i.MX 8 platforms.

Features:

- TensorFlow Lite v2.3.1
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units)
- C++ and Python API (supported Python version 3)
- Per-tensor and Per-channel quantized models support

3.1 TensorFlow Lite Software Stack

The TensorFlow Lite software stack is shown on the below picture. The TensorFlow Lite supports computation on the following HW units:

- CPU Arm Cortex-A cores
- GPU/NPU hardware accelerator using the Android NN API driver

See Software Stack Introduction for some details about supporting of computation on GPU/NPU hardware accelerator on different HW platforms.

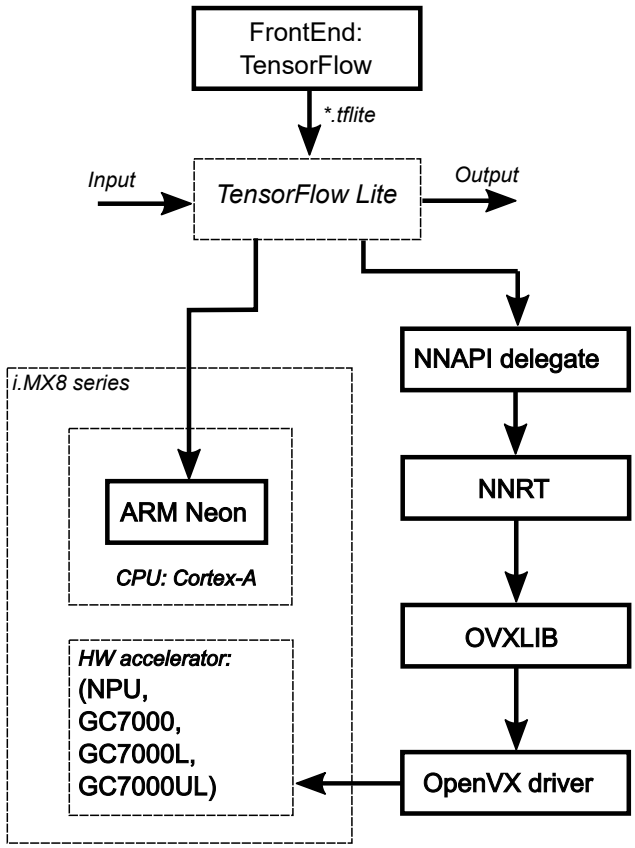

Figure 3. TensorFlow Lite SW stack

NOTE

The TensorFlow Lite library uses the Android NN API implementation from the GPU/NPU driver for running inference using the GPU/NPU hardware accelerator. The implemented NN API version is 1.2, which has limitations in supported tensor data types and operations, compared to the feature set of TensorFlow Lite. Therefore, some models may work without acceleration enabled, but may fail when using the NN API. For the full list of supported features, see the NN HAL versions section of the NN API documentation: https://source.android.com/devices/neural-networks#hal-versions.

The first execution of model inference using the NN API always takes many times longer, because of the time required for computational graph initialization by the GPU/NPU driver. The iterations following the graph initialization are performed many times faster.

The NN API implementation uses the OpenVX library for computational graph execution on the GPU/NPU hardware accelerator. Therefore, OpenVX library support must be available for the selected device to be able to use the acceleration. For more details on the OpenVX library availability, see the *i.MX Graphics User's Guide* (IMXGRAPHICUG).

The GPU/NPU hardware accelerator driver support both per-tensor and per-channel quantized models. In case of per-channel quantized models, performance degradation varies from slight differences, depending on the model used. This is caused by a hardware limitation, which is designed for per-tensor quantized models.

The TensorFlow Lite Converter V2 uses Quantize and Dequantize nodes to encapsulate some operation during quantization. Typically, the input and output tensor are followed/preceded by these nodes. As they are not fully supported by NN API their execution falls back to CPU. This causes the computational graph segmentation, leading to performance degradation.

3.2 Running image classification example

A Yocto Linux BSP image with machine learning layer included by default contains a simple pre-installed example called ‘label_image’ usable with image classification models. The example binary file location is at:

```
/usr/bin/tensorflow-lite-2.3.1/examples
```

Figure 4. TensorFlow image classification input

Demo instructions:

To run the example with mobilenet model on the CPU, use the following command line arguments:

```
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i grace_hopper.bmp -l labels.txt
```

The output of a successful classification for the 'grace_hopper.bmp' input image is as follows:

```
Loaded model mobilenet_v1_1.0_224_quant.tflite
resolved reporter
invoked
average time: 39.271 ms
0.780392: 653 military uniform
0.105882: 907 Windsor tie
0.0156863: 458 bow tie
0.0117647: 466 bulletproof vest
0.00784314: 835 suit
```

To run the example with the same model on the GPU/NPU hardware accelerator, add the -a 1 command line argument. To differentiate between the 3D GPU and the NPU, use the 'USE_GPU_INFERENCE' switch. For example, for the NPU hardware accelerator, use this command:

```
$ USE_GPU_INFERENCE=0 ./label_image -m mobilenet_v1_1.0_224_quant.tflite -i grace_hopper.bmp -l
labels.txt -a 1
```

The output with NPU acceleration enabled should be the following:

```
Loaded model mobilenet_v1_1.0_224_quant.tflite
resolved reporter
INFO: Created TensorFlow Lite delegate for NNAPI.
Applied NNAPI delegate.
invoked
average time: 2.967 ms
0.74902: 653 military uniform
```

```
0.121569: 907 Windsor tie
0.0196078: 458 bow tie
0.0117647: 466 bulletproof vest
0.00784314: 835 suit
```

Alternatively, the example using the TensorFlow Lite interpreter-only Python API can be run. The example file location is:

```
/usr/bin/tensorflow-lite-2.3.1/examples
```

To run the example using the predefined command line arguments, use the following command:

```
$ python3 label_image.py
```

The output should be the following:

```
INFO: Created TensorFlow Lite delegate for NNAPI.
Applied NNAPI delegate.
Warm-up time: 9862.1 ms

Inference time: 3.2 ms

0.678431: military uniform
0.129412: Windsor tie
0.039216: bow tie
0.027451: mortarboard
0.019608: bulletproof vest
```

NOTE

The TensorFlow Lite Python API does not contain functions for switching between execution on CPU and GPU/NPU hardware accelerator. By default, GPU/NPU hardware accelerator is used for hardware acceleration. The backend selection depends on the availability of the `libneuralnetworks.so` or `libneuralnetworks.so.1` in the `/usr/lib` directory. If the library is found by the shared library search mechanism, then the GPU/NPU backend is used.

3.3 Running benchmark applications

A Yocto Linux BSP image with machine learning layer included by default contains a pre-installed benchmarking application. It performs a simple TensorFlow Lite model inference and prints benchmarking information. The application binary file location is at:

```
/usr/bin/tensorflow-lite-2.3.1/examples
```

Benchmarking instructions:

To run the benchmark with computation on CPU, use the following command line arguments:

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite
```

You can optionally specify the number of threads with the `--num_threads=X` parameter to run the inference on multiple cores. For highest performance, set X to the number of cores available.

The output of the benchmarking application should be similar to:

```
STARTING!
Duplicate flags: num_threads
Min num runs: [50]
Min runs duration (seconds): [1]
Max runs duration (seconds): [150]
Inter-run delay (seconds): [-1]
```

```
Num threads: [1]
Use caching: [0]
Benchmark name: []
Output prefix: []
Min warmup runs: [1]
Min warmup runs duration (seconds): [0.5]
Graph: [mobilenet_v1_1.0_224_quant.tflite]
Input layers: []
Input shapes: []
Input value ranges: []
Input layer values files: []
Allow fp16 : [0]
Require full delegation : [0]
Enable op profiling: [0]
Max profiling buffer entries: [1024]
CSV File to export profiling data to: []
Enable platform-wide tracing: [0]
#threads used for CPU inference: [1]
Max number of delegated partitions : [0]
Min nodes per partition : [0]
Loaded model mobilenet_v1_1.0_224_quant.tflite
The input model file size (MB): 4.27635
Initialized session in 93.252ms.
Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding
150 seconds.
count=4 first=147477 curr=140410 min=140279 max=147477 avg=142382 std=2971

Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding
150 seconds.
count=50 first=140422 curr=140269 min=140269 max=140532 avg=140391 std=67

Inference timings in us: Init: 93252, First inference: 147477, Warmup (avg): 142382, Inference
(avg): 140391
Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the
actual memory footprint of the model at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=3.14062 overall=10.043
```

To run the inference using the GPU/NPU hardware accelerator, add the `--use_nnapi=true` command line argument:

```
$ ./benchmark_model --graph=mobilenet_v1_1.0_224_quant.tflite --use_nnapi=true
```

The output with GPU/NPU module acceleration enabled should be similar to:

```
STARTING!
Duplicate flags: num_threads
Min num runs: [50]
Min runs duration (seconds): [1]
Max runs duration (seconds): [150]
Inter-run delay (seconds): [-1]
Num threads: [1]
Use caching: [0]
Benchmark name: []
Output prefix: []
Min warmup runs: [1]
Min warmup runs duration (seconds): [0.5]
Graph: [mobilenet_v1_1.0_224_quant.tflite]
Input layers: []
Input shapes: []
Input value ranges: []
Input layer values files: []
```

```
Allow fp16 : [0]
Require full delegation : [0]
Enable op profiling: [0]
Max profiling buffer entries: [1024]
CSV File to export profiling data to: []
Enable platform-wide tracing: [0]
#threads used for CPU inference: [1]
Max number of delegated partitions : [0]
Min nodes per partition : [0]
Loaded model mobilenet_v1_1.0_224_quant.tflite
INFO: Created TensorFlow Lite delegate for NNAPI.
Applied NNAPI delegate, and the model graph will be completely executed w/ the delegate.
The input model file size (MB): 4.27635
Initialized session in 18.648ms.
Running benchmark for at least 1 iterations and at least 0.5 seconds but terminate if exceeding
150 seconds.
count=1 curr=5969598

Running benchmark for at least 50 iterations and at least 1 seconds but terminate if exceeding
150 seconds.
count=306 first=3321 curr=3171 min=3161 max=3321 avg=3188.46 std=18

Inference timings in us: Init: 18648, First inference: 5969598, Warmup (avg): 5.9696e+06, Inference
(avg): 3188.46
Note: as the benchmark tool itself affects memory footprint, the following is only APPROXIMATE to the
actual memory footprint of the model at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=7.60938 overall=33.7773
```

The benchmark application is also useful to check the optional segmentation of the models if accelerated on GPU/NPU hardware accelerator. For this purpose, the combination of the `--enable_op_profiling=true` and `--max_delegated_partitions=<big number>` (e.g., 1000) options can be used.

As previously mentioned in Section TensorFlow Lite Software Stack, when using the TensorFlow Lite Converter V2 (https://www.tensorflow.org/lite/convert) tool, there will be QUANTIZE nodes inserted at the beginning and end of the model. As they are not fully supported by NN API, their execution falls back to CPU. This causes the computational graph segmentation, leading to performance degradation.

As the mobilenet_v1_1.0_224_quant.tflite model available on the image is fully quantized and no segmentation happens there, download the following model:

```
$ wget https://storage.googleapis.com/tfhub-lite-models/tensorflow/lite-model/efficientnet/lite0/
int8/2.tflite -O efficientnet_lite0_int8_2.tflite
```

```
$ ./benchmark_model --graph=efficientnet_lite0_int8_2.tflite --use_nnapi=true --
enable_op_profiling=true --max_delegated_partitions=1000
```

In addition to the output presented above, detailed information is available why a particular layer was refused by the delegate:

```
Loaded model efficientnet_lite0_int8_2.tflite
INFO: Created TensorFlow Lite delegate for NNAPI.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Value should be Float32.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Output should be kTfLiteUInt8.
WARNING: Operator QUANTIZE (v1) refused by NNAPI delegate: Value should be Float32.
Applied NNAPI delegate, and the model graph will be partially executed w/ the delegate.
```

And detailed profiling information is available:

```
Profiling Info for Benchmark Initialization:
============================== Run Order ==============================
```

```
            [node type]     [start]   [first]   [avg ms]       [%]     [cdf%]    [mem KB]
ModifyGraphWithDelegate       0.000    10.938    10.938    98.505%    98.505%    1756.000
        AllocateTensors      10.904     0.162     0.083     1.495%   100.000%       0.000

============================== Top by Computation Time ==============================
            [node type]     [start]   [first]   [avg ms]       [%]     [cdf%]    [mem KB]
ModifyGraphWithDelegate       0.000    10.938    10.938    98.505%    98.505%    1756.000
        AllocateTensors      10.904     0.162     0.083     1.495%   100.000%       0.000

Number of nodes executed: 2
============================== Summary by node type ==============================
            [Node type]    [count]  [avg ms]    [avg %]     [cdf %]    [mem KB]   [times called]
ModifyGraphWithDelegate       1      10.938    98.505%     98.505%    1756.000         1
        AllocateTensors       1       0.166    1.495%     100.000%       0.000         2

Timings (microseconds): count=1 curr=11104
Memory (bytes): count=0
2 nodes observed



Operator-wise Profiling Info for Regular Benchmark Runs:
============================== Run Order ==============================
        [node type]      [start]   [first]    [avg ms]       [%]     [cdf%]   [mem KB]
           QUANTIZE       0.000      0.597       0.547    8.163%     8.163%      0.000
TfLiteNnapiDelegate       0.548      6.309       6.149   91.707%    99.870%      0.000
           QUANTIZE       6.698      0.010       0.009    0.130%   100.000%      0.000

============================== Top by Computation Time ==============================
        [node type]     [start]   [first]   [avg ms]        [%]      [cdf%]  [mem KB]
TfLiteNnapiDelegate      0.548     6.309      6.149    91.707%     91.707%      0.000
           QUANTIZE      0.000     0.597      0.547     8.163%     99.870%      0.000
           QUANTIZE      6.698     0.010      0.009     0.130%    100.000%      0.000

Number of nodes executed: 3
============================== Summary by node type ==============================
        [Node type]     [count]   [avg ms]  [avg %]     [cdf %]    [mem KB]   [times called]
TfLiteNnapiDelegate        1       6.148    91.720%     91.720%       0.000          1
           QUANTIZE        2       0.555     8.280%    100.000%       0.000          2
```

Based on section “Number of nodes executed” in the output, it can be determined which part of the computation graph was executed on GPU/NPU hardware accelerator. Every node except TfLiteNnapiDelegate falls back to CPU. In the example above, the QUANTIZE node falls back to CPU.

3.4 Application development

This section describes how to use TensorFlow Lite C++ API in the application development.

To start with TensorFlow Lite C++ application development, a Yocto SDK has to be generated first. See also the *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG) for more information.

After building the Yocto SDK, the TensorFlow Lite artefacts are located as follows:

- TensorFlow Lite static library (libtensorflow-lite.a) in */usr/lib*
- TensorFlow Lite header files in */usr/include*

To build the image classification demo from https://github.com/tensorflow/tensorflow/tree/v2.3.1/tensorflow/lite/examples/label_image for example, run the following compiler command under the generated Yocto SDK environment:

```
$CC label_image.cc bitmap_helpers.cc ../../tools/evaluation/utils.cc \
    -I=/usr/include/tensorflow/lite/tools/make/downloads/flatbuffers/include \
    -I=/usr/include/tensorflow/lite/tools/make/downloads/absl \ -O1 -DTFLITE_WITHOUT_XNNPACK
    -ltensorflow-lite -lstdc++ -lpthread -lm -ldl -lrt
```

NOTE

TensorFlow Lite static library is built without supporting the XNNPACK backend.

For more information about the C++ API, see the API reference at https://www.tensorflow.org/lite/api_docs/cc.

Alternatively, the Python API can be used for application development. For more information, see the Python version of the image classification example and the Python guide at https://www.tensorflow.org/lite/guide/python.

3.5 Post training quantization using TensorFlow Lite converter

TensorFlow offers several methods for model quantization:

- Post training quantization with TensorFlow Lite Converter
- Quantization aware training using Model Optimization Toolkits and TensorFlow Lite Converter
- Various other methods available in previous TensorFlow releases

Covering all of them is beyond the scope of this documentation. This section describes the recommended approach for the post training quantization using the TensorFlow Lite Converter.

The Converter is available as a part of standard TensorFlow desktop installation. It is used to convert and optionally quantize TensorFlow model into TensorFlow Lite model format. There are two options how to use the tool:

- The Python API (recommended)
- Command line script

The post training quantization using the Python API is described in this chapter. The documentation useful for model conversion and quantization is available here:

- Python API documentation: https://www.tensorflow.org/versions/r2.3/api_docs/python/tf/lite/TFLiteConverter
- Guide for model conversion: www.tensorflow.org/lite/convert
- Guide for model quantization:https://www.tensorflow.org/lite/performance/post_training_quantization

NOTE

The guides on TensorFlow page usually covers the most up to date version of TensorFlow, which might be different from the version available in the NXP eIQ. To see what features are available, check the corresponding API for the specific version of the TensorFlow or TensorFlow Lite.

The current version of the TensorFlow Lite available in the eIQ is 2.3.1. It is recommended to use the converter from TensorFlow version 2.3.1or older. The 2.3.1 version of the converter has the following properties:

- In the post training quantization regime, the per-channel quantization is the only option. The per-tensor quantization is available only in connection with quantization aware training.
- Input and output tensors quantization is supported by setting the required data type in `inference_input_type` and `inference_output_type`.
- TOCO or MLIR based conversions are available. This is controlled by the `experimental_new_converter` attribute. As TOCO is becoming obsolete, MLIR-based conversion is already set by default in the 2.3.1 version of the converter.

NOTE

MLIR is a new NN compiler used by TensorFlow, which supports quantization. Before MLIR, quantization was performed by TOCO (or TOCOConverter), which is now obsolete. See https://www.tensorflow.org/api_docs/python/tf/compat/v1/lite/TocoConverter. For details about MLIR, see https://www.tensorflow.org/mlir.

NOTE

Do not use the dynamic range method for models being run on NN accelerators (GPU or NPU). It converts only the weights to 8-bit integers, but retains the activations in fp32, which results in the inference running in fp32 with an additional overhead for data conversion. In fact, the inference is even slower compared to a fp32 model, because the conversion is done on the fly.

For the post training quantization, a representative dataset is needed. The proper choose of samples in representative dataset highly influences the accuracy of the final quantized model. The best practices for creating the representative dataset are:

- Use train samples for which the original floating points model has very good accuracy, based on metrics the model used (e.g., SoftMax score for classification models, IOU for object detection models, etc.).
- There shall be enough samples in representative dataset.
- The size of representative dataset and the specific samples available in it are considered as hyperparameters to tune, with respect of the required model accuracy.

For more information about quantization using TensorFlow ecosystem, see these links:

- www.tensorflow.org/lite/convert
- www.tensorflow.org/lite/performance/post_training_quantization
- www.tensorflow.org/model_optimization

Chapter 4
Arm Compute Library

Arm Compute Library (ACL) is a collection of low-level functions optimized for Arm CPU and GPU (not supported on i.MX 8 devices) architectures targeted at image processing, computer vision, and machine learning.

Arm Compute Library is designed as a compute engine for the Arm NN framework, so it is suggested to use Arm NN unless there is a need for a more optimized runtime.

Source codes using which ACL is being built are available at https://source.codeaurora.org/external/imx/arm-computelibrary-imx.

Features:

- Arm Compute Library 20.02.01
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores
- C++ API only
- Low-level control over computation

4.1 Running a DNN with random weights and inputs

Arm compute library comes with examples for most common DNN architectures like: AlexNet, MobileNet, ResNet, Inception v3, Inception v4, Squeezenet, etc.

All available examples can be found in this example build location:

```
/usr/share/arm-compute-library/build/examples
```

Each model architecture can be tested using graph_[dnn_model] application.

For example, to run the MobileNet v2 DNN model, use the following command:

```
$ ./graph_mobilenet_v2 --data=<path_cnn_data> --image=<input_image> --labels=<labels> --target=neon --
type=<data_type> --threads=<num_of_threads>
```

The parameters are not mandatory. When not provided, the application runs the model with random weights and inputs. If inference finishes successfully, the “Test passed" message is printed.

4.1.1 Running AlexNet using graph API

In 2012, AlexNet shot to fame 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 a ReLu activation function, except for the last fully connected layer, where the Softmax function is used.

Location of the C++ AlexNet example implementation using the graph API is in this folder:

```
/usr/share/arm-compute-library/build/examples
```

Demo instructions:

- Download the archive file to the example location folder from this link: compute_library_alexnet.zip.
- Create a new sub-folder and unzip the file.
- Run the example with command line arguments from the default location:

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

Set environment variables for execution:

```
$ export PATH_ASSETS=/usr/share/arm-compute-library/build/examples/assets_alexnet/
$ ./graph_alexnet --data=$PATH_ASSETS --image=$PATH_ASSETS/go_kart.ppm --labels=$PATH_ASSETS/
labels.txt --target=neon --type=f32 --threads=4
```

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

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

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

Test passed
```

Chapter 5
Arm NN

Arm NN is an open-source inference engine framework developed by Linaro Artificial Intelligence Initiative, which NXP is a part of. It does not perform computations on its own, but rather delegates the input from multiple model formats such as Caffe, TensorFlow, TensorFlow Lite, or ONNX, to specialized compute engines.

Source codes using which Arm NN is being built are available at https://source.codeaurora.org/external/imx/armnn-imx.

Features:

- Arm NN 20.02.01
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores provided by the ACL Neon backend
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units) provided by the VSI NPU backend
- C++ and Python API (supported Python version 3)
- Supports multiple input formats (TensorFlow, TensorFlow Lite, Caffe, ONNX)
- Off-line tools for serialization, deserialization, and quantization (must be built from source)

5.1 Arm NN Software Stack

The Arm NN software stack is shown in the picture below. Arm NN supports computation on the following HW units:

- CPU Arm Cortex-A cores
- GPU/NPU hardware accelerator using the VSI NPU backend, which runs on both the GPU and the NPU depending on which is available

See Software Stack Introduction for details about the support of GPU/NPU accelerators for each hardware platform.

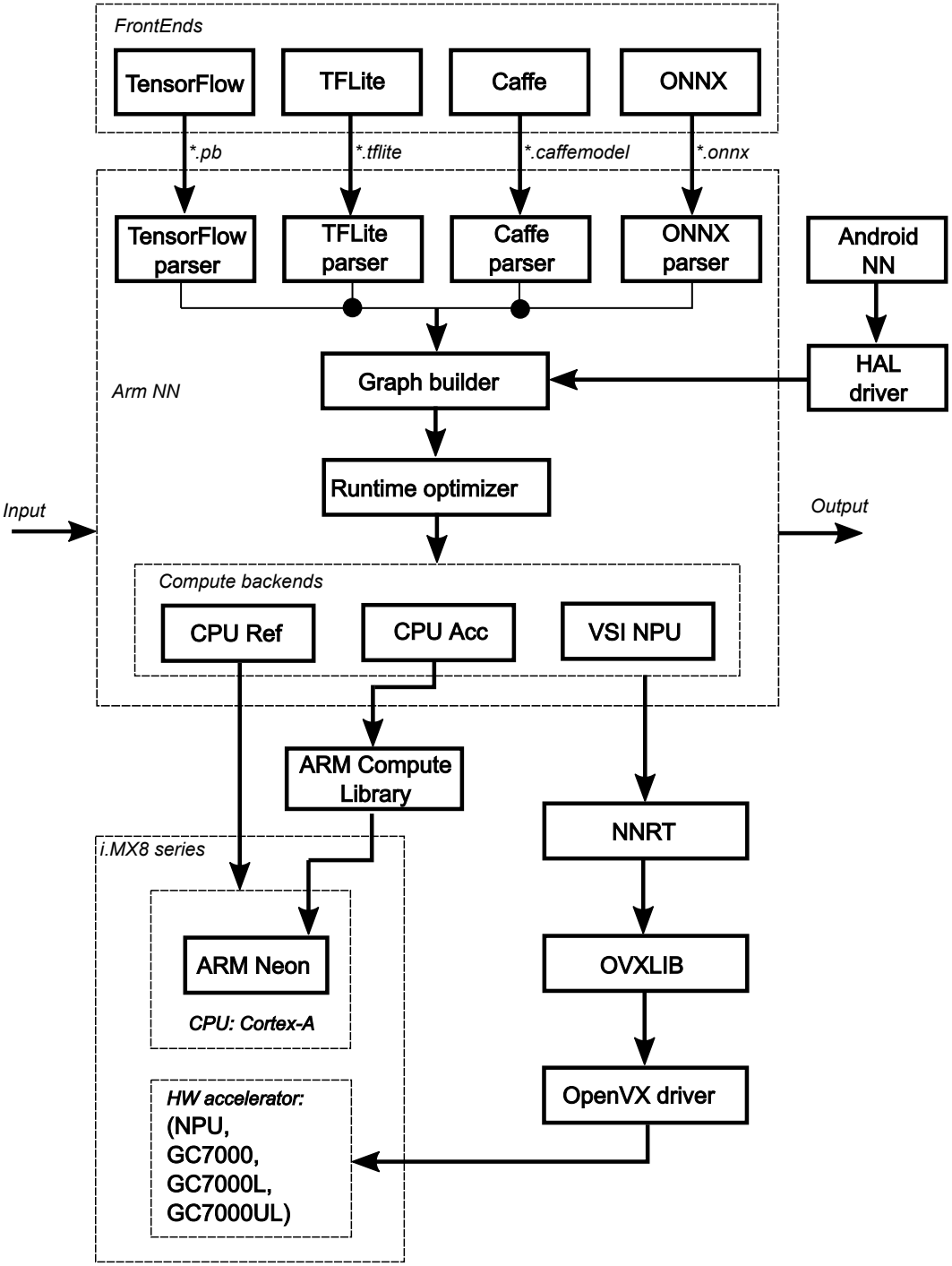

Figure 5. Arm NN SW stack

NOTE

The OpenCL backend (GPU Acc) is not supported on i.MX 8 devices.

5.2 Compute backends

Arm NN on its own does not specialize in implementing compute operations. There is only the C++ reference backend running on the CPU which is not optimized for performance and should be used for testing, checking results, prototyping, or as the final

fallback if none of the other backends support a specific layer. The other backends delegate compute operations to other more specialized libraries such as Arm Compute Library (ACL).

- **For the CPU** there is the NEON backend, which uses Arm Compute Library with the Arm NEON SIMD extension.
- **For the GPUs and NPUs** NXP provides the VSI NPU backend, which leverages the full capabilities of i.MX 8's GPUs/NPUs using OpenVX and provides a great performance boost. ACL OpenCL backend, which you might notice in the source codes, is not supported due to Arm NN OpenCL requirements not being fulfilled by the i.MX8 GPUs.

To activate the chosen backend while running the examples described in the following sections, add the following argument. The user can give multiple backends for the example applications. A layer in the model will be executed by the first backend which supports the layer:

```
<example_binary> --compute=CpuAcc --compute=VsiNpu --compute=CpuRef
```

- **CpuRef** = ArmNN C++ backend (no SIMD instructions); a set of reference implementations with NO acceleration on the CPU, which is used for testing, prototyping, or as the final fallback. It is very slow.
- **CpuAcc** = ACL NEON backend (runs on CPU with NEON instructions = SIMD)
- **VsiNpu** = For the GPUs and NPUs, NXP provides the VSI NPU backend, which leverages the full capabilities of i.MX8's GPUs.

If you are developing your own application you must make sure that you pass the chosen backend (CpuAcc, VsiNpu, or CpuRef) to the Optimize function for inference.

5.3 Running Arm NN tests

Arm NN SDK provides a set of tests which can also be considered as demos showing what Arm NN does and how to use it. They load neural network models of various formats (Caffe, TensorFlow, TensorFlow Lite, ONNX), run the inference on a specified input data, and output the inference result. Arm NN tests are built by default when building the Yocto image and are installed in */usr/bin*. Note that input data, model configurations, and model weights are not distributed with Arm NN. The user must download them separately and make sure they are available on the device before running the tests. However, Arm NN tests don't come with a documentation. Input file names are hardcoded, so investigate the code to find out what input file names are expected.

To help in getting started with Arm NN, the following sections provide details about how to prepare the input data and how to run Arm NN tests. All of them use well-known neural network models, therefore, with only a few exceptions, such pre-trained networks are available freely on the Internet. Input images, models, formats, and their content was deduced using code analysis. However, this was not possible for all the tests, because either the models are not publicly available or it is not possible to deduce clearly what input files are required by the application. General work-flow is first to prepare data on a host machine and then to deploy it on the board, where the actual Arm NN tests will be run.

The following sections assume that neural network model files are stored in a folder called models and 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
```

5.3.1 Caffe tests

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

```
/usr/bin/CaffeAlexNet-Armnn
/usr/bin/CaffeCifar10AcrossChannels-Armnn
/usr/bin/CaffeInception_BN-Armnn
/usr/bin/CaffeMnist-Armnn
/usr/bin/CaffeResNet-Armnn
```

```
/usr/bin/CaffeVGG-Armnn
/usr/bin/CaffeYolo-Armnn
```

Two important limitations might require preprocessing of the Caffe model file prior to running an Arm NN Caffe test. First, Arm NN tests require the batch size to be set to 1. Second, Arm NN does not support all Caffe syntaxes, therefore some older neural network model files will require updates to the latest Caffe syntax.

Details about how to perform these preprocessing steps are described on the Arm NN GitHub page. Install Caffe on the host. Also check Arm NN documentation for Caffe support.

For example, if a Caffe model has a batch size different from one or uses an older Caffe version 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 and change the batch size to 1 and finally run the following python script:

```
import caffe

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

NOTE

For the full list of the supported operators, see caffe support.

The below table shows the list of all dependencies for each Arm NN Caffe binary example.

Table 1. Arm NN Caffe example dependencies

Arm NN binary	Model file name	Model definition	Input data	Renamed model file name
CaffeAlexNet-Armnn	bvlc_alexnet.caffemodel	deploy.prototxt	shark.jpg	bvlc_alexnet_1.caffemodel
CaffeInception_BN-Armnn	Inception21k.caffemodel	deploy.prototxt	shark.jpg	Inception-BN-batchsize1.caffemodel
CaffeMnist-Armnn	lenet_iter_9000.caffemodel	lenet.prototxt	t10k-images.idx3-ubyte, t10k-labels.idx1-ubyte	lenet_iter_9000.caffemodel
CaffeResNet-Armnn	Model not available	N/A	N/A	N/A
CaffeVGG-Armnn	VGG_ILSVRC_19_layers.caffemodel	VGG_ILSVRC_19_layers_deploy.prototxt	shark.jpg	VGG_CNN_S.caffemodel
CaffeCifar10AcrossChannels-Armnn	model not available	N/A	N/A	N/A
CaffeYolo-Armnn	model not available	N/A	N/A	N/A

Follow the below steps to run each of the above examples:

1. Download the model and model definition files (see columns 2 and 3 of the table)
2. Transform the network as explained in this section
3. Rename the original model name to the new model name (see column 5 of the table)

4. Copy renamed model to the *models* folder on the device
5. Download the input data (column 4) and copy it to the *data* folder on the device.
6. Run the test:

```
$ cd ArmnnTests
$ <armnn_binary> --data-dir=data --model-dir=models
```

5.3.2 TensorFlow tests

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

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

NOTE

For the full list of the supported operators, see TensorFlow support.

The below table shows the list of all dependencies for each Arm NN TensorFlow binary example.

Table 2. Arm NN TF example dependencies

Arm NN binary	Model file name	Input data
TfInceptionV3-Armnn	Inception_v3_2016_08_28_frozen.pb	shark.jpg, Dog.jpg, Cat.jpg
TfMnist-Armnn	simple_mnist_tf.prototxt	t10k-images.idx3-ubyte, t10k-labels.idx1-ubyte
TfMobileNet-Armnn	mobilenet_v1_1.0_224_frozen.pb	shark.jpg, Dog.jpg, Cat.jpg
TfResNext-Armnn	model not available	N/A
TfCifar10-Armnn	model not available	N/A

Follow the below steps to run each of the above examples:

1. Download the model (column 2 of the table) and copy it to the *models* folder on the device
2. Download the input data (column 3 of the table) and copy it to the *data* folder on the device
3. Run the test:

```
$ cd ArmnnTests
$ <armnn_binary> --data-dir=data --model-dir=models
```

5.3.3 TensorFlow Lite tests

Arm NN SDK provides the following test for TensorFlow Lite models:

```
/usr/bin/TfLiteInceptionV3Quantized-Armnn
/usr/bin/TfLiteInceptionV4Quantized-Armnn
/usr/bin/TfLiteMnasNet-Armnn
/usr/bin/TfLiteMobileNetSsd-Armnn
/usr/bin/TfLiteMobilenetQuantized-Armnn
/usr/bin/TfLiteMobilenetV2Quantized-Armnn
/usr/bin/TfLiteResNetV2-Armnn
/usr/bin/TfLiteVGG16Quantized-Armnn
```

```
/usr/bin/TfLiteResNetV2-50-Quantized-Armnn
/usr/bin/TfLiteMobileNetQuantizedSoftmax-Armnn
```

NOTE

For the full list of the supported operators, see TensorFlow Lite support.

The below table shows the list of all dependencies for each Arm NN TensorFlow Lite binary example.

Table 3. Arm NN TensorFlow Lite example dependencies

Arm NN binary	Model file name	Input data
TfLiteInceptionV3Quantized-Armnn	inception_v3_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMnasNet-Armnn	mnasnet_1.3_224.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMobilenetQuantized-Armnn	mobilenet_v1_1.0_224_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteMobilenetV2Quantized-Armnn	mobilenet_v2_1.0_224_quant.tflite	shark.jpg, Dog.jpg, Cat.jpg
TfLiteResNetV2-50-Quantized-Armnn	model not available	N/A
TfLiteInceptionV4Quantized-Armnn	model not available	N/A
TfLiteMobileNetSsd-Armnn	model not available	N/A
TfLiteResNetV2-Armnn	model not available	N/A
TfLiteVGG16Quantized-Armnn	model not available	N/A
TfLiteMobileNetQuantizedSoftmax-Armnn	model not available	N/A

Follow the below steps to run each of the above examples:

1. Download the model (column 2 of the table) and copy it to the *models* folder on the device
2. Download the input data (column 3 of the table) and copy it to the *data* folder on the device
3. Run the test:

```
$ cd ArmnnTests
$ <armnn_binary> --data-dir=data --model-dir=models
```

5.3.4 ONNX tests

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

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

NOTE

For the full list of the supported operators, see ONNX support.

The below table shows the list of all dependencies for each Arm NN ONNX binary example.

Table 4. Arm NN ONNX example dependencies

Arm NN binary	Model file name	Input data	Renamed model file name
OnnxMnist-Armnn	model.onnx	t10k-images.idx3-ubyte, t10k-labels.idx1-ubyte	mnist_onnx.onnx
OnnxMobileNet-Armnn	mobilenetv2-1.0.onnx	shark.jpg, Dog.jpg, Cat.jpg	mobilenetv2-1.0.onnx

Follow the below steps to run each of the above examples:

1. Download the model (column 2 of the table)
2. Rename the original model name to the new model name (column 4 of the table) and copy it to the *models* folder on the device
3. Download the input data (column 3 of the table) and copy it to the *data* folder on the device
4. Run the test:

```
$ cd ArmnnTests
$ <armnn_binary> --data-dir=data --model-dir=models
```

5.4 Using Arm NN in a custom C/C++ application

You can create your own C/C++ applications for the i.MX 8 family of devices using Arm NN capabilities. This requires writing the code using the Arm NN API, setting up the build dependencies, cross-compiling the code for an aarch64 architecture, and deploying your application. Below is a detailed description for each of these steps:

1. Write the code.

   A good starting point to understand how to use Arm NN API in your own application is to go through "How-to guides" provided by Arm. These include two applications; one shows how to load and run inference for an MNIST TensorFlow model, and the second one shows how to load and run inference for an MNIST Caffe model.

2. Prepare and install the SDK.

   From a software developer's perspective, Arm NN is a library. Therefore, to create and build an application, which uses Arm NN, you need header files and matching libraries. For how to build the Yocto SDK, see the *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG). By default, header files and libraries are not added. To make sure that the SDK contains both the header files and the libraries, add the following to your `local.conf`.

   ```
   TOOLCHAIN_TARGET_TASK_append += " armnn-dev"
   ```

3. Build the code.

   To build the "armnn-mnist" example provided by Arm, you need to make a few modifications to make it work with a Yocto cross-compile environment:

   - Remove the definition of *ARMNN_INC* and all its uses from Makefile. The Arm NN headers are already available in the default include directories.
   - Remove the definition of *ARMNN_LIB* and all its uses from Makefile. The Arm NN libraries are already available in the default linker search path.
   - Replace "g++" with "${CXX}" in Makefile.

   Build the example:

- Setup the SDK environment:

```
$ source <Yocto_SDK_install_folder>/environment-setup-aarch64-poky-linux
```

- Run make:

```
$ make
```

4. Copy the built application to the board.

Input data are described in the "How-to guides". If the image you are using on your board is the same as the one for which you built the SDK, all the runtime dynamic libraries needed to run the application should be available on the board.

5.5 Python interface to Arm NN (PyArmNN)

PyArmNN is a Python extension for Arm NN SDK. PyArmNN provides interface similar to Arm NN C++ API. It is supported only for Python 3.x and not Python 2.x.

For full API documentation please refer to NXPmicro GitHub: https://github.com/NXPmicro/pyarmnn-release

5.5.1 Getting started

The easiest way to begin using PyArmNN is by using the Parsers. We will demonstrate how to use them below:

Create a parser object and load your model file.

```
import pyarmnn as ann
import imageio

# ONNX, Caffe and TF parsers also exist.
parser = ann.ITfLiteParser()
network = parser.CreateNetworkFromBinaryFile('./model.tflite')
```

Get the input binding information by using the name of the input layer.

```
input_binding_info = parser.GetNetworkInputBindingInfo(0, 'input_layer_name')

# Create a runtime object that will perform inference.
options = ann.CreationOptions()
runtime = ann.IRuntime(options)
```

Choose preferred backends for execution and optimize the network.

```
# Backend choices earlier in the list have higher preference.
preferredBackends = [ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]
opt_network, messages = ann.Optimize(network, preferredBackends, runtime.GetDeviceSpec(),
ann.OptimizerOptions())

# Load the optimized network into the runtime.
net_id, _ = runtime.LoadNetwork(opt_network)
```

Make workload tensors using input and output binding information.

```
# Load an image and create an inputTensor for inference.
# img must have the same size as the input layer; PIL or skimage might be used for resizing if img
has a different size
```

```
img = imageio.imread('./image.png')
input_tensors = ann.make_input_tensors([input_binding_info], [img])

# Get output binding information for an output layer by using the layer name.
output_binding_info = parser.GetNetworkOutputBindingInfo(0, 'output_layer_name')
output_tensors = ann.make_output_tensors([outputs_binding_info])
```

Perform inference and get the results back into a numpy array.

```


runtime.EnqueueWorkload(0, input_tensors, output_tensors)

results = ann.workload_tensors_to_ndarray(output_tensors)
print(results)
```

5.5.2 Running examples

For a more complete Arm NN experience, there are examples located in */usr/bin/armnn-20.02/examples/*, which require requests, PIL and maybe some other Python3 modules depending on your image. You may install the missing modules using pip3.

To run these examples you may simply execute them using the Python3 interpreter. There are no arguments and the resources are downloaded by the scripts:

```
$ python3 /usr/bin/armnn-20.02/examples/tflite_mobilenetv1_quantized.py
```

The output should look similar to the following:

```
Downloading 'mobilenet_v1_1.0_224_quant_and_labels.zip' from 'https://storage.googleapis.com/
download.tensorflow.org/models/tflite/mobilenet_v1_1.0_224_quant_and_labels.zip' ...
Finished.
Downloading 'kitten.jpg' from 'https://s3.amazonaws.com/model-server/inputs/kitten.jpg' ...
Finished.
Running inference on 'kitten.jpg' ...
class=tabby ; value=99
class=Egyptian cat ; value=84
class=tiger cat ; value=71
class=cricket ; value=0
class=zebra ; value=0
```

NOTE

example_utils.py is a file containg common functions for the rest of the scripts and it does not execute anything on its own.

Chapter 6
ONNX Runtime

ONNX Runtime is an open-source inferencing framework, which enables the acceleration of machine learning models across all of your deployment targets using a single set of API. Source codes are available at https://source.codeaurora.org/external/imx/onnxruntime-imx.

NOTE

For the full list of the CPU supported operators, see the 'operator kernels' documentation section: OperatorKernels.

For the know limitations, see the NXPReleaseNotes.

Features:

- ONNX Runtime 1.1.2
- Multithreaded computation with acceleration using Arm Neon SIMD instructions on Cortex-A cores provided by the ACL and Arm NN execution providers
- Parallel computation using GPU/NPU hardware acceleration (on shader or convolution units) provided by the VSI NPU execution provider
- C++ and Python API (supported Python version 3)

6.1 ONNX Runtime software stack

The ONNX Runtime software stack is shown in the following figure. The ONNX Runtime supports computation on the following HW units:

- CPU Arm Cortex-A cores
- GPU/NPU hardware accelerator using the execution providers (EP)

See Software Stack Introduction for some details about supporting of computation on GPU/NPU hardware accelerator on different HW platforms.

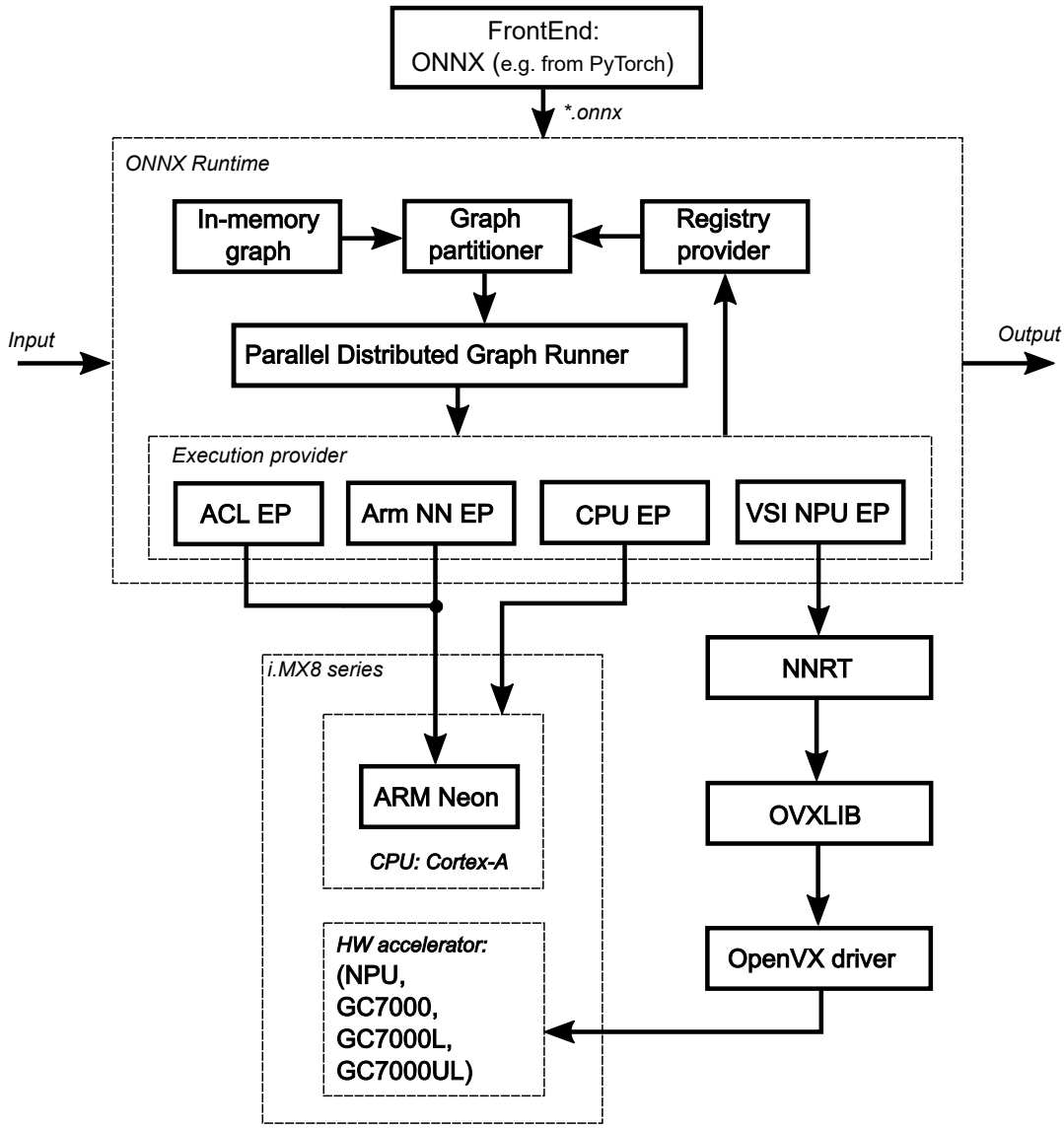

Figure 6. ONNX Runtime software stack

6.2 Execution Providers

Execution providers (EP) are a mechanism to delegate inference execution to an underlying framework or hardware. By default, the ONNX Runtime uses the CPU EP, which executes inference on the CPU without any specialized optimizations. For optimized inference, ACL, Arm NN, and VSI NPU EPs are supported. ACL EP uses the Arm NEON accelerated backend on the CPU directly. Arm NN EP uses the Arm NEON accelerated backend via ACL. For GPU or NPU acceleration, you may use the VSI NPU EP, which delegates the execution to OpenVX. The inference will be executed depending on what hardware is available on your i.MX8 device, thus it will be the NPU for i.MX8MP. For other i.MX8 devices it will be typically the GPU.

6.2.1 ONNX model test

ONNX Runtime provides a tool that can run the collection of standard tests provided in the ONNX model Zoo. The tool named onnx_test_runner is installed in /usr/bin.

ONNX models are available at https://github.com/onnx/models and consist of models and sample test data. Because some models require a lot of disk space, it is advised to store the ONNX test files on a larger partition, as described in the SD card image flashing section.

The following models from ONNX Zoo where tested with this release: MobileNet v2, ResNet50 v2, ResNet50 v1, SSD Mobilenet v1, Yolo v3.

Here is an example with the steps required to run the mobilenet version 2 test:

- Download and unpack the mobilenet version 2 test archive located at https://github.com/onnx/models/blob/master/vision/classification/mobilenet/model/mobilenetv2-7.tar.gz.
- Copy the mobilenetv2-1.0 folder containing the model and test data on the device in case the previous step was not executed directly on the board, for example, to the /home/root folder.
- Run the onnx_test_runner tool providing mobilenetv2-1.0 folder path and setting the execution provider to Arm NN:

```
$ ls ./mobilenetv2-1.0
mobilenet_v2_1.0_224.onnx  test_data_set_0  test_data_set_1  test_data_set_2
$ onnx_test_runner -j 1 -c 1 -r 1 -e [armnn/acl/vsi_npu] ./mobilenetv2-1.0/
result:
        Models: 1
        Total test cases: 3
                Succeeded: 3
                Not implemented: 0
                Failed: 0
        Stats by Operator type:
                Not implemented(0):
                Failed:
Failed Test Cases:
$
```

For the full list of supported options type:

```
$ onnx_test_runner -h
```

6.2.2 C API

ONNX Runtime also provides a C API sample code described here: https://github.com/microsoft/onnxruntime/blob/v1.1.2/docs/C_API.md.

To build the sample from https://raw.githubusercontent.com/microsoft/onnxruntime/v1.1.2/csharp/test/Microsoft.ML.OnnxRuntime.EndToEndTests.Capi/C_Api_Sample.cpp, run the following build command under the generated Yocto SDK environment:

```
$CXX -std=c++0x C_Api_Sample.cpp -o onnxruntime_sample -I=/usr/include/onnxruntime/core/session -
lonnxruntime
```

ONNX Runtime libraries and header files are not included in the SDK by default. To make sure that they will be installed, add the following to your `local.conf`:

```
TOOLCHAIN_TARGET_TASK_append += " onnxruntime-dev"
```

6.2.2.1 Enabling execution provider

To enable a specific execution provider, you need to do the following in your code:

- Set the execution provider in code (see the previous C API sample how that is done for the CUDA EP). You have options to set the following EPs in your code. If not set, the default CPU EP would be used:
  - `OrtSessionOptionsAppendExecutionProvider_ArmNN(session_options, 0);` for the Arm NN EP.
  - `OrtSessionOptionsAppendExecutionProvider_ACL(session_options, 0);` for the ACL EP.
  - `OrtSessionOptionsAppendExecutionProvider_VsiNpu(session_options, 0);` for the VSI NPU EP.

- Include headers based on the EP used in the code: `#include "armnn_provider_factory.h"`, `#include "acl_provider_factory.h"` or `#include "vsi_npu_provider_factory.h"`.
- Add includes to the build command: `-I=/usr/include/onnxruntime/core/providers/armnn/`, `-I=/usr/include/onnxruntime/core/providers/acl/`, or `-I=/usr/include/onnxruntime/core/providers/vsi_npu/`

Chapter 7
PyTorch

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

Features:

- PyTorch 1.6.0
- Tensor computation (like NumPy) with strong GPU acceleration
- Deep neural networks built on a tape-based autograd sytem

NOTE

This release of PyTorch V1.6.0 does not yet support the tensor computation on the NXP GPU/NPU. Only the CPU is supported. By default, the PyTorch runtime is running with floating point model. To enable quantized model, the quantized engine should be specified explicitly like below:

```
torch.backends.quantized.engine = 'qnnpack'
```

7.1 Running image classification example

There is an example located in the examples folder, which requires urllib, PIL and maybe some other Python3 modules depending on your image. You may install the missing modules using pip3.

```
$ cd /usr/bin/pytorch/examples
```

To run the example with inference computation on the CPU, use the following command. There are no arguments and the resources will be downloaded automatically by the script. Make sure, the internet access is available.

```
$ python3 pytorch_mobilenetv2.py
```

The output should look similar as the following:

```
File does not exist, download it from
https://download.pytorch.org/models/mobilenet_v2-b0353104.pth
... 100.00%, downloaded size: 13.55 MB
File does not exist, download it from
https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.txt
... 100.00%, downloaded size: 0.02 MB
File does not exist, download it from
https://s3.amazonaws.com/model-server/inputs/kitten.jpg
... 100.00%, downloaded size: 0.11 MB
('tabby, tabby cat', 46.34805679321289)
('Egyptian cat', 15.802854537963867)
('lynx, catamount', 1.1611212491989136)
('lynx, catamount', 1.1611212491989136)
('tiger, Panthera tigris', 0.20774540305137634)
```

7.2 Build and install wheel packages

This release includes building script for PyTorch and TorchVision on aarch64 platform. Currently, it supports the native building on the NXP aarch64 platform with BSP SDK.

NOTE

Generally, in the yocto rootfs of the BSP SDK, the PyTorch and TorchVision wheel packages are already integrated. There is no need to build and install from scratch. Anyway, if you would like to build them by your own, please follow below steps.

7.2.1 How to build

Follow the below steps:

1. Get the latest i.MX BSP from https://source.codeaurora.org/external/imx/imx-manifest.
2. Set-up the build environment for one of the NXP aarch64 platforms and edit the *local.conf* to add the following dependency for PyTorch native build:

```
IMAGE_INSTALL_append = " python3-dev python3-pip python3-wheel python3-pillow python3-setuptools
python3-numpy python3-pyyaml
python3-cffi python3-future cmake ninja packagegroup-core-buildessential git git-perltools
libxcrypt libxcrypt-dev
```

3. Build the BSP images like the following command:

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

4. Get into the pytorch folder and execute the build script on NXP aarch64 platform to generate wheel packages. You can get the source from https://github.com/NXPmicro/pytorch-release as well:

```
$ cd /path/to/pytorch/src
$ ./build.sh
```

7.2.2 How to install

If the building is successful, the wheel packages should be found under */path/to/pytorch/src/dist* folder:

```
$ pip3 install /path/to/torch-1.6.0-cp37-cp37m-linux_aarch64.whl
$ pip3 install /path/to/torchvision-0.7.0-cp37-cp37m-linux_aarch64.whl
```

Chapter 8
OpenCV machine learning demos

OpenCV is an open source computer vision library and one of its modules, called ML, provides traditional machine learning algorithms. OpenCV offers a unified solution for both neural network inference (DNN module) and classic machine learning algorithms (ML module).

Features:

- OpenCV 4.4.0
- C++ and Python API (supported Python version 3)
- Only CPU computation is supported
- Input image or live camera (webcam) is supported

8.1 Downloading OpenCV demos

OpenCV DNN demos are located at:

```
/usr/share/OpenCV/samples/bin
```

Input data, and model configurations are located at:

```
/usr/share/opencv4/testdata/dnn
```

NOTE

To have the above "testdata/dnn" directory on the image, put the following in `local.conf` before the image building. See also Section "NXP® eIQ™ machine learning" in the *i.MX Yocto Project User's Guide* (IMXLXYOCTOUG).

```
PACKAGECONFIG_append_pn-opencv_mx8 += " test"
```

Model weights are not located in the image, because of the size. Before running the DNN demos, these files should be downloaded to the device from the above directory:

```
$ python3 download_models_basic.py
```

NOTE

Use original *download_models.py* script if all possible models and configuration files are needed (10GB SD card size is needed). Use *download_models_basic.py* script if only basic models for below DNN examples are needed (1GB SD card size is needed).

Copy all above dependencies (models, inputs, and weights) to:

```
/usr/share/OpenCV/samples/bin
```

Download the configuration model.yml. This file contains preprocessing parameters for some DNN examples, which accepts the `--zoo` parameter. Copy the model file to:

```
/usr/share/OpenCV/samples/bin
```

8.2 OpenCV DNN demos

The OpenCV DNN module implements an inference engine and does not provide any functionalities for neural network training.

8.2.1 Image classification demo

This demo performs image classification using a pretrained SqueezeNet network. Demo dependencies are from opencv_extra-4.4.0.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- squeezenet_v1.1.caffemodel
- squeezenet_v1.1.prototxt

Other demo dependencies:

- classification_classes_ILSVRC2012.txt from

  ```
  /usr/share/OpenCV/samples/data/dnn
  ```

- models.yml from github

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

```
$ ./example_dnn_classification --input=dog416.png --zoo=models.yml squeezenet
```

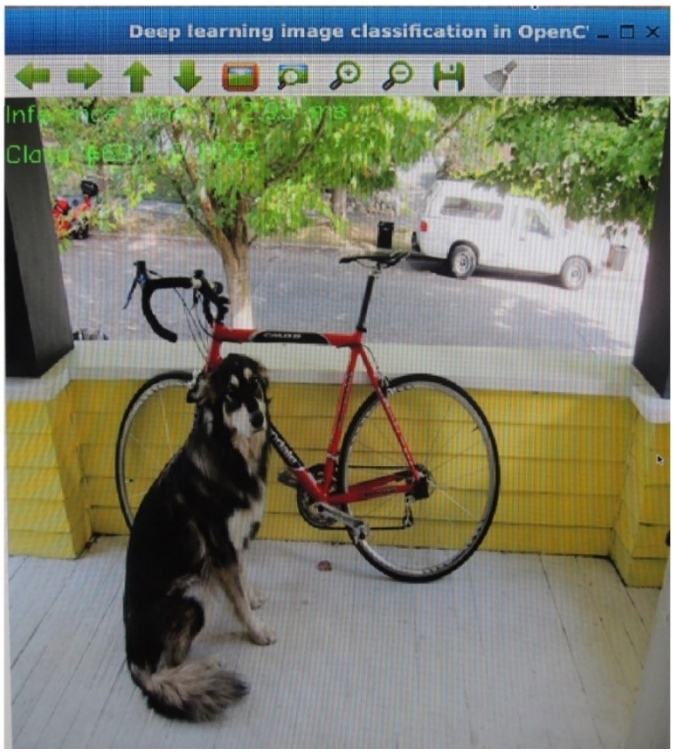

Figure 7. Image classification graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_classification --device=3 --zoo=models.yml squeezenet
```

NOTE

Choose the right port where the camera is currently connected. Use the `v4l2-ctl --list-devices` command to check it.

8.2.2 YOLO object detection example

The YOLO object detection demo performs object detection using You Only Look Once (YOLO) detector. It detects objects on camera, video, or image. Find out more information about this demo at OpenCV Yolo DNNs page. Demo dependencies are from opencv_extra-4.4.0.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- yolov3.weights
- yolov3.cfg

Other demo dependencies:

- models.yml from github
- object_detection_classes_yolov3.txt from

```
/usr/share/OpenCV/samples/data/dnn
```

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

```
$ ./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 -input=dog416.png -rgb -
zoo=models.yml yolo
```

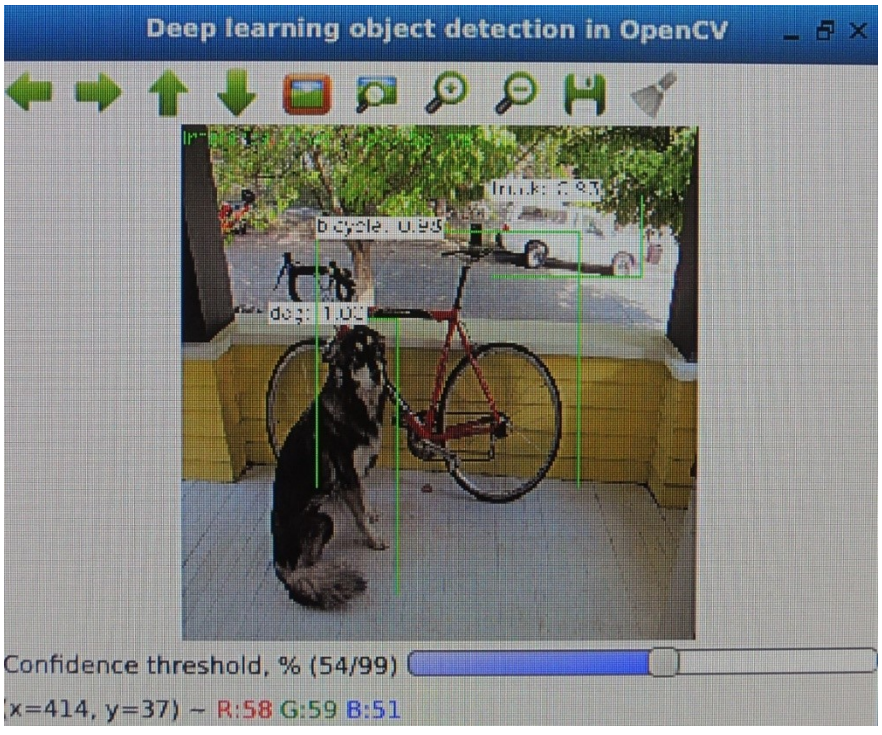

Figure 8. YOLO object detection graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_object_detection -width=1024 -height=1024 -scale=0.00392 --device=3 -rgb -
zoo=models.yml yolo
```

NOTE

Choose the right port where the camera is currently connected. Use the `v4l2-ctl --list-devices` command to check it.

NOTE

Running this example with live camera input is quite slow, because of running the example on the CPU only.

8.2.3 Image segmentation demo

The image segmentation means dividing the image into groups of pixels based on some criteria grouping based on color, texture, or some other criteria. Demo dependencies are from opencv_extra-4.4.0.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- dog416.png
- fcn8s-heavy-pascal.caffemodel
- fcn8s-heavy-pascal.prototxt

Other demo dependencies are models.yml from github. Run the C++ example with image input from the default location:

```
$ ./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --input=dog416.png --
zoo=models.yml fcn8s
```

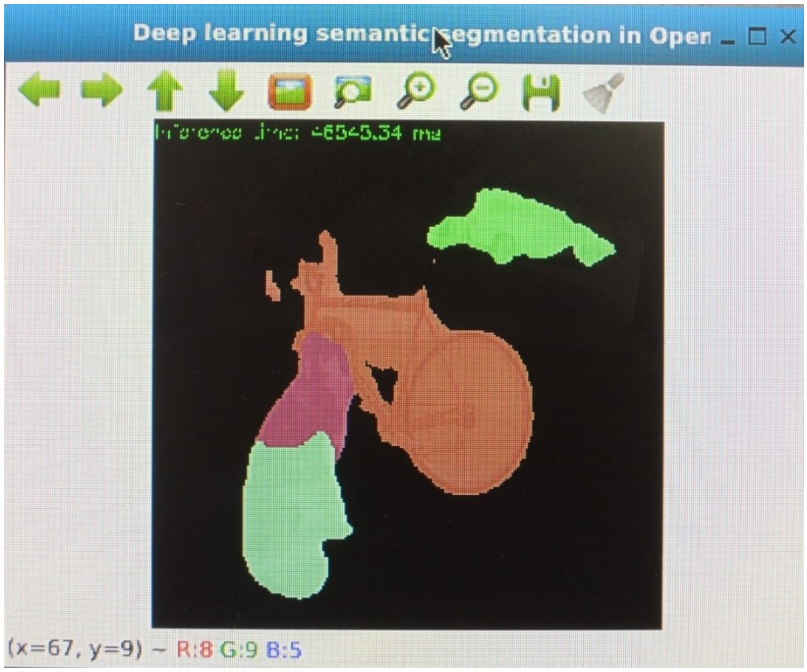

Figure 9. Image segmentation graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --device=3 --zoo=models.yml fcn8s
```

NOTE

Choose the right port where the camera is currently connected. Use the `v4l2-ctl --list-devices` command to check it.

NOTE

Running this example with live camera input is quite slow, because of running the example on the CPU only.

8.2.4 Image colorization demo

This sample demonstrates recoloring grayscale images with DNN. The demo supports input images only, not the live camera input. Demo dependencies are from opencv_extra-4.4.0.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- colorization_release_v2.caffemodel
- colorization_deploy_v2.prototxt

Other demo dependencies are basketball1.png from

```
/usr/share/OpenCV/examples/data
```

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

```
$ ./example_dnn_colorization --model=colorization_release_v2.caffemodel --
proto=colorization_deploy_v2.prototxt --image=../data/basketball1.png
```

Figure 10. Image colorization graphics output

8.2.5 Human pose detection demo

This application demonstrates human or hand pose detection with a pretrained OpenPose DNN. The demo supports input images only and no live camera input. Demo dependencies are from opencv_extra-4.4.0.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- grace_hopper_227.png
- openpose_pose_coco.caffemodel
- openpose_pose_coco.prototxt

Running the C++ example with 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 --dataset=COCO
```

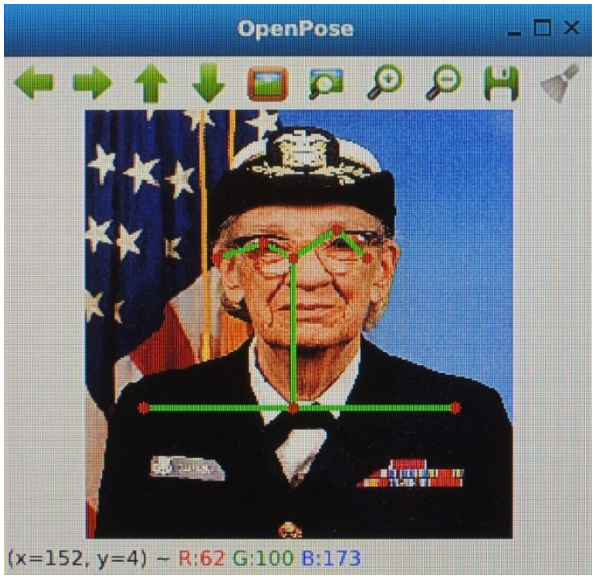

Figure 11. Human pose estimation graphics output

8.2.6 Object Detection Example

This demo performs object detection using a pretrained SqueezeDet network. The demo supports input images only, not the live camera input. Demo dependencies are the following:

- SqueezeDet.caffemodel model weight file
- SqueezeDet_deploy.prototxt model definition file
- Input image aeroplane.jpg

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

```
$ ./example_dnn_objdetect_obj_detect SqueezeDet_deploy.prototxt SqueezeDet.caffemodel aeroplane.jpg
```

Running the model on the aeroplane.jpg image produces the following text results in the console:

```
------
Class: aeroplane
```

```
Probability: 0.845181
Co-ordinates:
```

Figure 12. Object detection graphics output

8.2.7 CNN image classification example

This demo performs image classification using a pretrained SqueezeNet network. The demo supports input images only, not the live camera input. Demo dependencies are the following:

- SqueezeNet.caffemodel model weight file
- SqueezeNet_deploy.prototxt model definition file
- Input image space_shuttle.jpg from

```
/usr/share/opencv4/testdata/dnn
```

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

```
$ ./example_dnn_objdetect_image_classification SqueezeNet_deploy.prototxt SqueezeNet.caffemodel
space_shuttle.jpg
```

Running the model on the space_shuttle.jpg image produces the following text results in the console:

```
Best class Index: 812
Time taken: 0.649153
Probability: 15.8467
```

8.2.8 Text detection

This demo is used for text detection in the image using EAST algorithm. Demo dependencies are from opencv_extra-4.4.0.zip or from:

```
/usr/share/opencv4/testdata/dnn
```

- frozen_east_text_detection.pb

Other demo dependencies are imageTextN.png from

```
/usr/share/OpenCV/samples/data
```

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

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

NOTE

This example accepts the PNG image format only.

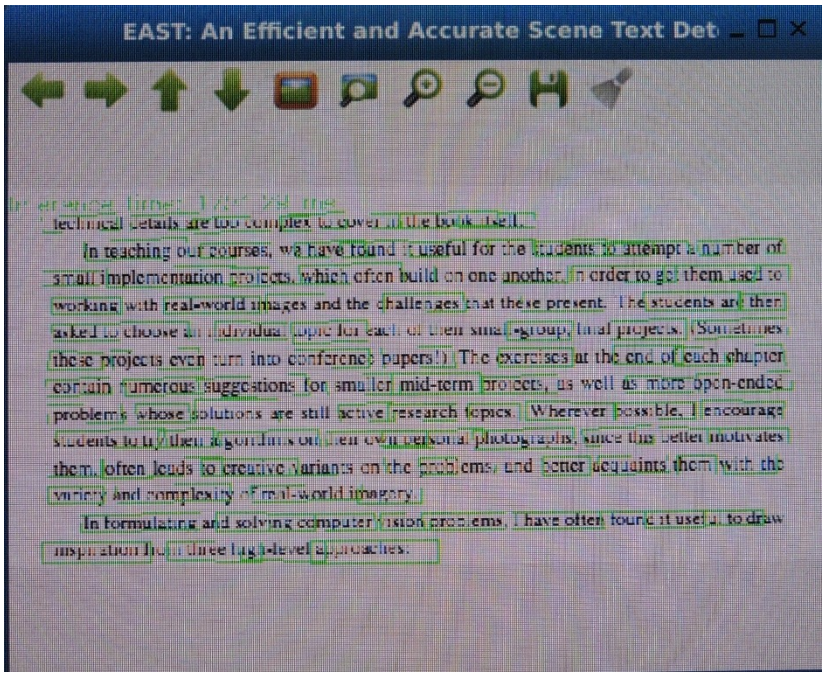

Figure 13. Text detection graphics output

Running the C++ example with the live camera connected to the port 3:

```
$ ./example_dnn_text_detection --model=frozen_east_text_detection.pb --device=3
```

NOTE

Choose the right port where the camera is currently connected. Use the `v4l2-ctl --list-devices` command to check it.

8.3 OpenCV classical machine learning demos

After deploying OpenCV on the target device, Non-Neural Networks demos are installed in the rootfs in

```
/usr/share/OpenCV/samples/bin/
```

8.3.1 SVM Introduction

This example demonstrates how to create and train an SVM model using training data. Once the model is trained, labels for test data are predicted. The full description of the example can be found in (tutorial_introduction_to_svm). For displaying the result, an image with Qt5 enabled is required.

After running the demo, the graphics result is shown on the screen:

```
$ ./example_tutorial_introduction_to_svm
```

Result:

- The code opens an image and shows the training examples of both classes. The points of one class are represented with white circles, and other class uses black points.
- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into a blue region and a green region. The boundary between both regions is the optimal separating hyperplane.
- Finally, the support vectors are shown using gray rings around the training examples.

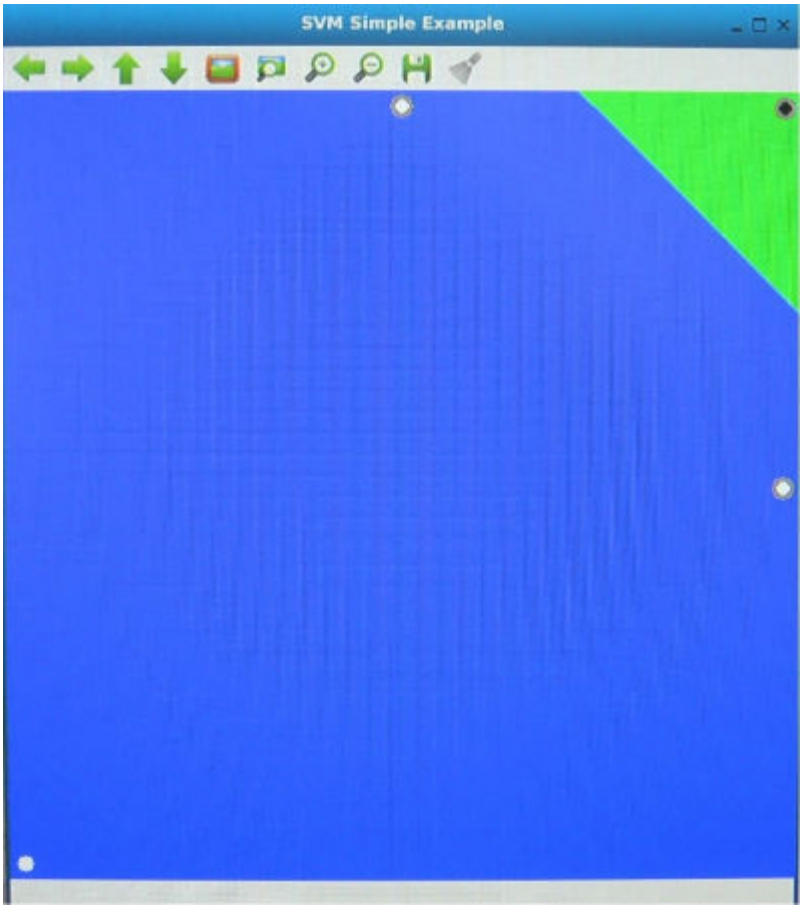

Figure 14. SVM introduction graphics output

8.3.2 SVM for non-linearly separable data

This example deals with non-linearly separable data and shows how to set parameters of SVM with linear kernel for this data. For more details, go to SVM_non_linearly_separable_data.

After running the demo, the graphics result is shown on the screen (it requires Qt5 support):

```
$ ./example_tutorial_non_linear_svms
```

Result:

- The code opens an image and shows the training data of both classes. The points of one class are represented with light green, the other class uses light blue points.
- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into blue green regions. The boundary between both regions is the separating hyperplane. Since the training data is non-linearly separable, some of the examples of both classes are misclassified; some green points lay on the blue region and some blue points lay on the green one.
- Finally, the support vectors are shown using gray rings around the training examples.

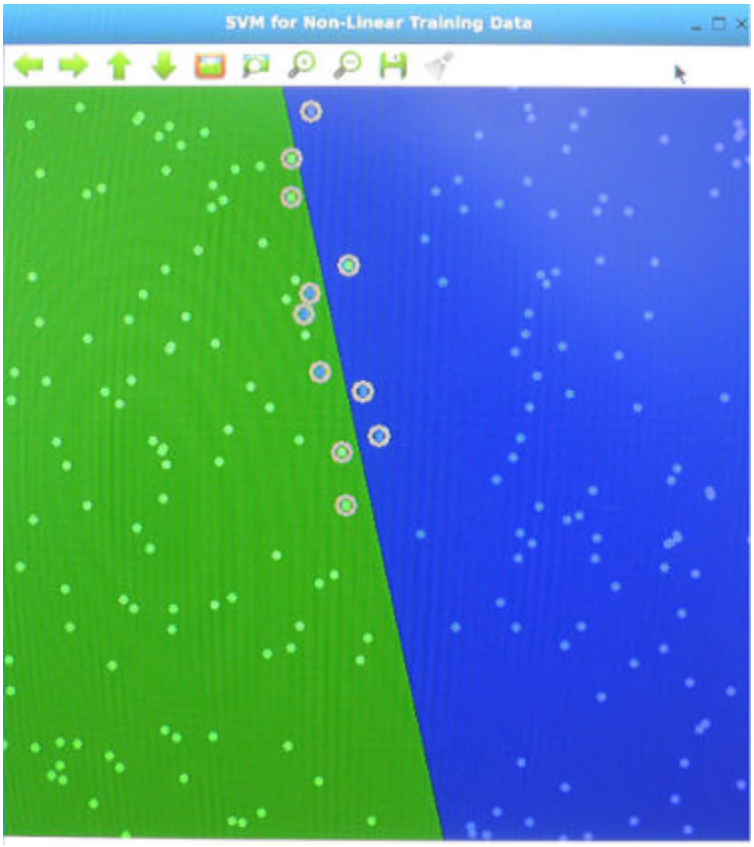

Figure 15. SVM for Non-linear training data

8.3.3 Prinicipal Component Analysis (PCA) introduction

Principal Component Analysis (PCA) is a statistical method that extracts the most important features of a dataset. In this tutorial you will learn how to use PCA to calculate the orientation of an object. For more details, check the OpenCV tutorial Introduction_to_PCA.

After running the demo, the graphics result is shown on the screen (it requires Qt 5 support):

```
$ ./example_tutorial_introduction_to_pca ../data/pca_test1.jpg
```

Results:

- Open an image (loaded from ../data/pca_test1.jpg)
- Find the orientation of the detected objects of interest
- Visualizes the result by drawing the contours of the detected objects of interest, the center point, and the x-axis, y-axis regarding the extracted orientation.

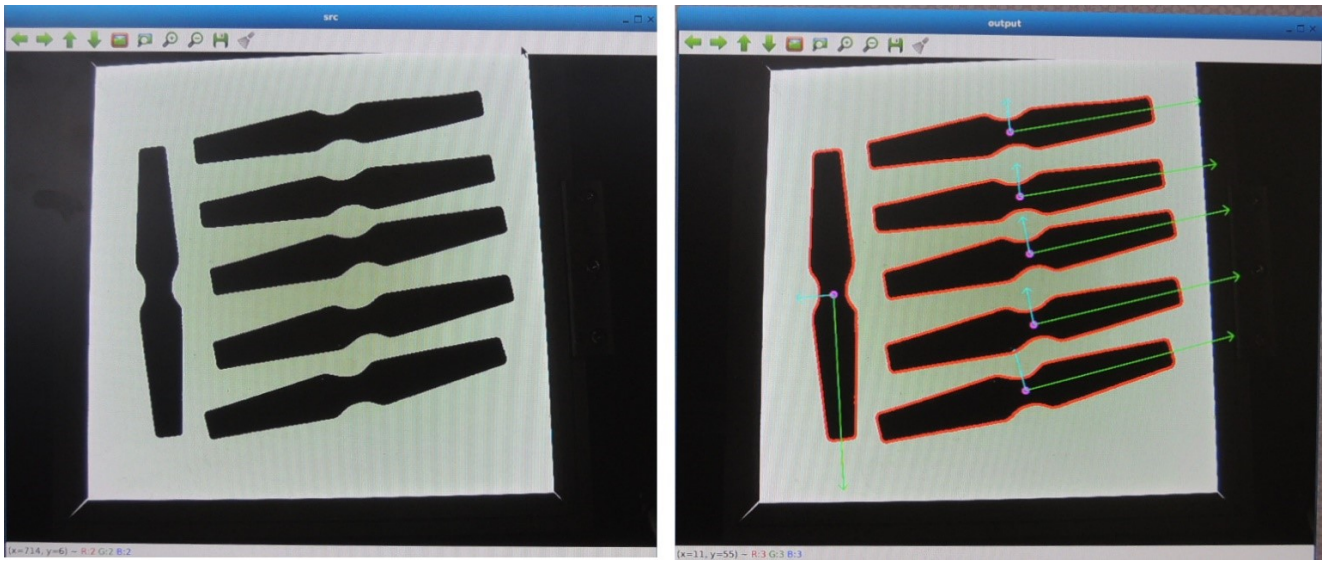

Figure 16. PCA graphics output

8.3.4 Logistic regression

In this sample, logistic regression is used for prediction of two characters (0 or 1) from an image. First, every image matrix is reshaped from its original size of 28x28 to 1x784. A logistic regression model is created and trained on 20 images. After training, the model can predict labels of test images. The source code is located on the logistic_regression link, and can be run by typing the following command.

Demo dependencies (preparing the train data files):

```
$ wget https://raw.githubusercontent.com/opencv/opencv/4.4.0/samples/data/data01.xml
```

After running the demo, the graphics result is shown on the screen (it requires Qt 5 support):

```
$ ./example_cpp_logistic_regression
```

Results:

- Training and test data are shown
- Comparison between original and predicted labels is displayed.

The console text output is as follows (the trained model reaches 95% accuracy):

```
original vs predicted:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
```

```
accuracy: 95%
saving the classifier to NewLR_Trained.xml
loading a new classifier from NewLR_Trained.xml
predicting the dataset using the loaded classifier...done!
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1]
accuracy: 95%
```

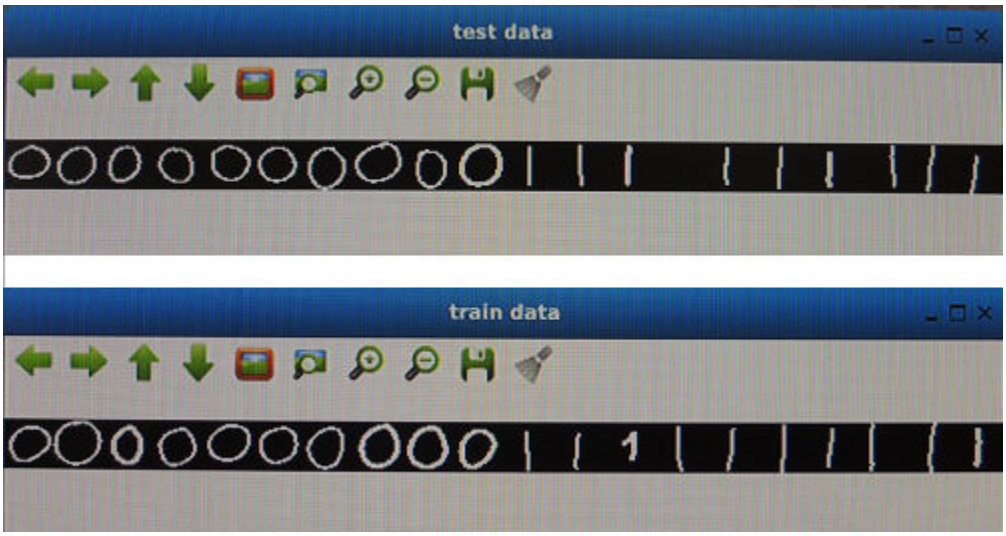

Figure 17. Logistic regression graphics output

Chapter 9
NN Execution on Hardware Accelerators

9.1 Profiling on hardware accelerators

This section describes the steps to enable profiler on the GPU/NPU, and how to capture logs.

1. Stop the EVK board in U-Boot by pressing Enter.
2. Update mmcargs by adding `galcore.showArgs=1` and `galcore.gpuProfiler=1`.

```
u-boot=> editenv mmcargs
        edit: setenv bootargs ${jh_clk} console=${console} root=${mmcroot}
galcore.showArgs=1 galcore.gpuProfiler=1
        u-boot=> boot
```

3. Boot the board and wait for the Linux OS prompt.
4. The following environment flags should be enabled before executing the application. `VIV_VX_DEBUG_LEVEL` and `VIV_VX_PROFILE` flags should always be **1** during the process of profiling. The `CNN_PERF` flag enables the driver's ability to generate per layer profile log. `NN_EXT_SHOW_PERF` shows the details of how compiler estimates performance and determines tiling based on it.

```
export CNN_PERF=1 NN_EXT_SHOW_PERF=1 VIV_VX_DEBUG_LEVEL=1 VIV_VX_PROFILE=1
```

5. Capture the profiler log. We use the sample ML example part of standard NXP Linux release to explain the following section.
   - TensorFlow Lite profiling

     Run the TFLite application with GPU/NPU backend as follows:

```
$ cd /usr/bin/tensorflow-lite-2.3.1/examples
$ ./label_image -m mobilenet_v1_1.0_224_quant.tflite -t 1 -i grace_hopper.bmp -l labels.txt
-a 1 -v 0 > viv_test_app_profile.log 2>&1
```

   - Armnn profiling

     Run the ArmNN application (here TfMobilNet is taken as example) with GPU/NPU backend as follows:

```
$ /usr/bin/TfMobileNet-Armnn --data-dir=data --model-dir=models --compute=VsiNpu >
viv_test_app_profile.log 2>&1
```

   The log captures detailed information of the execution clock cycles and DDR data transmission in each layer.

NOTE

The average time for inference might be longer than usual, as the profiler overhead is added.

9.2 Hardware accelerators warmup time

For both Arm NN and TensorFlow Lite, the initial execution of model inference takes longer time, because of the model graph initialization needed by the GPU/NPU hardware accelerator. The initialization phase is known as warmup. This time duration can be decreased for subsequent application that runs by storing on disk the information resulted from the initial OpenVX graph processing. The following environment variables should be used for this purpose:

`VIV_VX_ENABLE_CACHE_GRAPH_BINARY`: flag to enable/disable OpenVX graph caching

`VIV_VX_CACHE_BINARY_GRAPH_DIR`: set location of the cached information on disk

For example, set these variables on the console in this way:

```
export VIV_VX_ENABLE_CACHE_GRAPH_BINARY="1"
export VIV_VX_CACHE_BINARY_GRAPH_DIR =`pwd`
```

By setting up these variables, the result of the OpenVX graph compilation is stored on disk as network binary graph files (`*.nb`). The runtime performs a quick hash check on the network and if it matches the `*.nb` file hash, it loads it into the NPU memory directly. These environment variables need to be set persistently, for example, available after reboot. Otherwise, the caching mechanism is bypassed even if the `*.nb` files are available.

The iterations following the graph initialization are performed many times faster. When evaluating the performance of an application running on GPU/NPU, the time should be measured separately for warmup and inference. Warmup time usually affects only the first inference run. However, depending on the machine learning model type, it might be noticeable for the first few inference runs. Some preliminary tests must be done to make a decision on what to consider warmup time. When this phase is well delimited, the subsequent inference runs can be considered as pure inference and used to compute an average for the inference phase.

Chapter 10
Revision History

This table provides the revision history.

Table 5. Revision history

Revision number	Date	Substantive changes
L5.4.47_2.2.0	09/2020	Initial release
L5.4.70_2.3.0	12/2020	i.MX 5.4 consolidated GA for release i.MX boards including i.MX 8M Plus and i.MX 8DXL.

Appendix A
Neural network API reference

The neural-network operations and corresponding supported API functions are listed in the following table.

Table 6. Neural-network operations and supported API functions

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
Activation					
elu	vx_kernel (ELU)	-	ELU	-	Elu
floor	vxTensorRounding Node	ANEURALNETWORKS_FLOOR	Floor	Floor	Floor
leakyrelu	vxLeakyReluLayer	-	LEAKY_RELU	Activation/ LeakyReLu	LeakyReLu
prelu	vx_kernel (PRELU)	ANEURALNETWORKS_PRELU	PRELU	PreLu	PreLu
relu	vxActivationLayer	ANEURALNETWORKS_RELU	RELU	Activation/ReLu	ReLu
relu1	vxActivationLayer	ANEURALNETWORKS_RELU1	RELU1	-	-
relu6	vxActivationLayer	ANEURALNETWORKS_RELU6	RELU6	-	-
relun	vxActivationLayer	-	RELU_N1_TO_1	-	-
swish	-	-	-	-	-
Hard_swish	-	ANEURALNETWORKS_HARD_SWISH	HARD_SWISH	-	-
rsqrt	vxActivationLayer	ANEURALNETWORKS_RSQRT	RSQRT	-	-
sigmoid	vxActivationLayer	ANEURALNETWORKS_LOGISTIC	LOGISTIC	Activation/Sigmoid	Sigmoid
softmax	vxSoftmaxLayer	ANEURALNETWORKS_SOFTMAX	SOFTMAX	Softmax	Softmax
softrelu	vxActivationLayer	-	-	Activation/ SoftReLu	-
sqrt	vxActivationLayer	ANEURALNETWORKS_SQRT	SQRT	Activation/Sqrt	Sqrt
tanh	vxActivationLayer	ANEURALNETWORKS_TANH	TANH	Activation/TanH	TanH

Table continues on the next page...

Table 6. Neural-network operations and supported API functions (continued)

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
bounded	-	-	-	Activation/ BoundedReLu	-
linear	-	-	-	Activation/Linear	-
Dense Layers					
convolution_relu	vxConvolutionReluLayer	-	-	-	-
convolution_relu_pool	vxConvolutionReluPoolingLayer	-	-	-	-
fullyconnected_relu	vxFullyConnectedReluLayer	-	-	-	-
Element Wise					
abs	vxLeakyReluLayer	ANEURALNETWORKS_ABS	ABS	Activation/Abs	Abs
add	vxTensorAddNode	ANEURALNETWORKS_ADD	ADD	Addition	Add
add_n	vx_kernel (ADDN)	-	ADD_N	-	-
clip_by_value	vx_kernel (CLIP)	-	-	-	Clip
div	vxTensorDivideNode	ANEURALNETWORKS_DIV	DIV	Division	Div
equal	vx_kernel (EQUAL)	ANEURALNETWORKS_EQUAL	EQUAL	-	Equal
exp	vx_kernel (EXP)	ANEURALNETWORKS_EXP	EXP	-	Exp
log	vx_kernel (LOG)	ANEURALNETWORKS_LOG	LOG	-	Log
floor_div	vx_kernel (FLOOR_DIV)	-	FLOOR_DIV	-	-
greater	vx_kernel (GREATER)	ANEURALNETWORKS_GREATER	GREATER	-	Greater
greater_equal	vx_kernel (GREATER_EQUAL)	ANEURALNETWORKS_GREATER_EQUAL	GREATER_EQUAL	-	-
less	vx_kernel (LESS)	ANEURALNETWORKS_LESS	LESS	-	Less

Table continues on the next page...

Table 6. Neural-network operations and supported API functions (continued)

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
less_equal	vx_kernel (LESS_EQUAL)	ANEURALNETWORKS_LESS_EQUAL	LESS_EQUAL	-	-
logical_and	vx_kernel (LOGICAL_AND)	ANEURALNETWORKS_LOGICAL_AND	LOGICAL_AND	-	And
logical_or	vx_kernel (LOGICAL_OR)	ANEURALNETWORKS_LOGICAL_OR	LOGICAL_OR	-	Or
minimum	vx_kernel (MINIMUM)	ANEURALNETWORKS_MINIMUM	MINIMUM	Minimum	Min
maximum	vx_kernel (MAXIMUM)	ANEURALNETWORKS_MAXIMUM	MAXIMUM	Maximum	Max
multiply	vxTensorMultiplyNode	ANEURALNETWORKS_MUL	MUL	Multiplication	Mul
negative	vx_kernel (NEG)	ANEURALNETWORKS_NEG	NEG	-	Neg
not_equal	vx_kernel (NOT_EQUAL)	ANEURALNETWORKS_NOT_EQUAL	NOT_EQUAL	-	-
pow	vx_kernel (POW)	ANEURALNETWORKS_POW	POW	-	POW
real_div	vxTensorDivideNode	-	-	-	-
select	vx_kernel (SELECT)	ANEURALNETWORKS_SELECT	SELECT	-	-
square	vxActivationLayer	-	SQUARE	Activation/Square	-
sub	vxTensorSubtractNode	ANEURALNETWORKS_SUB	SUB	Substraction	Sub
where	vx_kernel (SELECT)	-	WHERE	-	Where
Image Processing					
image_resize	vxTensorScaleNode	ANEURALNETWORKS_RESIZE_BILINEAR	RESIZE_BILINEAR	-	Unsample
image_resize	vxTensorScaleNode	ANEURALNETWORKS_RESIZE_NEAREST_NEIGHBOR	RESIZE_NEAREST_NEIGHBOR	-	Resize

Table continues on the next page...

Table 6. Neural-network operations and supported API functions (continued)

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
yuv_rgb_scale	vxYUV2RGBScaleNode	-	-	-	-
Matrix Multiplication					
fullconnect	vxFullyConnectedLayer	ANEURALNETWORKS_FULLY_CONNECTED	FULLY_CONNECTED	FullyConnected	-
matrix_mul	vx_kernel (MATRIXMUL)	-	-	-	-
Normalization					-
batch_normalize	vxBatchNormalizationLayer	-	-	BatchNormalization	BatchNormalization
instance_normalize	vx_kernel (INSTANCE_NORM)	-	-	Normalization	InstanceNormalization
l2normalize	vxL2NormalizeLayer	ANEURALNETWORKS_L2_NORMALIZATION	L2_NORMALIZATION	L2Normalization	-
layer_normalize	vx_kernel (LAYER_NORM)	-	-	-	-
localresponsenormalization	vxNormalizationLayer	ANEURALNETWORKS_LOCAL_RESPONSE_NORMALIZATION	LOCAL_RESPONSE_NORMALIZATION	-	LRN
Reshape					
batch2space	vxReorgLayer2	ANEURALNETWORKS_BATCH_TO_SPACE_ND	BATH_TO_SPACE_ND	BatchToSpaceNd	-
concat	vxCreateTensorView	ANEURALNETWORKS_CONCATENATION	CONCATENATION	Concat	Concat
crop	vx_kernel (CROP)	-	-	-	-
depth_to_space	vxReorgLayer2	ANEURALNETWORKS_DEPTH_TO_SPACE	DEPTH_TO_SPACE	-	DepthToSpace
expanddims	vxReshapeTensor	ANEURALNETWORKS_EXPAND_DIMS	EXPAND_DIMS	-	-
flatten	vxReshapeTensor	ANEURALNETWORKS_RESHAPE	-	-	-

Table continues on the next page...

Table 6. Neural-network operations and supported API functions (continued)

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
gather	vx_kernel (GATHER)	ANEURALNETWORKS_GATHER	GATHER	-	Gather
pad	vxTensorPadNode	ANEURALNETWORKS_PAD	PAD/ MIRROR_PAD	Pad	Pad
permute	vxTensorPermuteNode	ANEURALNETWORKS_TRANSPOSE	TRANSPOSE	Permute	Transpose
reducemean	vxTensorMeanNode	ANEURALNETWORKS_MEAN	MEAN	Mean	ReduceMean
reducesum	vxTensorReduceSumNode	ANEURALNETWORKS_SUM	SUM	-	ReduseSum
gathernd	-	-	GATHER_ND	-	GatherND
reducemax	-	ANEURALNETWORKS_REDUCE_MAX	REDUCE_MAX	-	ReduceMax
reducemin	-	ANEURALNETWORKS_REDUCE_MIN	REDUCE_MIN	-	ReduceMin
reorg	vxReorgLayer	-	-	-	-
reshape	vxReshapeTensor	ANEURALNETWORKS_RESHAPE	RESHAPE	Reshape	Reshape
reverse	vxTensorReverse	-	-	-	ReverseSequence
reverse_squeeze	vxTensorReverse	-	-	-	-
slice	vxCreateTensorView	ANEURALNETWORKS_SLICE	SLICE	-	Slice
space2batch	vxReorgLayer2	ANEURALNETWORKS_SPACE_TO_BATCH_ND	SPACE_TO_BATCH_ND	SpaceToBatchNd	-
space2depth	vxReorgLayer2	ANEURALNETWORKS_SPACE_TO_DEPTH	SPACE_TO_DEPTH	SpaceToDepth	SpaceToDepth
split	vxCreateTensorView	ANEURALNETWORKS_SPLIT	SPLIT	Split	Split
squeeze	vxReshapeTensor	ANEURALNETWORKS_SQUEEZE	SQUEEZE	Squeeze	Squeeze
stack	vx_kernel (STACK)	-	-	-	-

Table continues on the next page...

Table 6. Neural-network operations and supported API functions (continued)

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
strided_slice	vxTensorStrideSliceNode	ANEURALNETWORKS_STRIDED_SLICE	STRIDED_SLICE	StridedSlice	-
tensor_stack_concat	vx_kernel (TENSORSTACKCONCAT)	-	-	-	-
unstack	vx_kernel (UNSTACK)	-	UNPACK	Unpack	-
RNN					
gru	vx_kernel (GRU_OVXLIB)	-	-	-	GRU
gru_cell	vx_kernel (GRUCELL_OVXLIB)	-	-	-	-
lstm	vxLSTMLayer	-	UNIDIRECTIONAL_SEQUEENCE_LSTM	-	-
lstmunit	vxLSTMUnitLayer	ANEURALNETWORKS_LSTM	LSTM	LstmUnit	LSTM
rnn	vxRNNLayer	ANEURALNETWORKS_RNN	RNN	-	-
Sliding Window					
avg_pool	vxPoolingLayer	ANEURALNETWORKS_AVERAGE_POOL	AVERAGE_POOL_2D	Pooling2D/avg	AveragePool
convolution	vxConvolutionLayer	ANEURALNETWORKS_CONV_2D	CONV_2D	Convolution2D	Conv
deconvolution	vxDeconvolutionLayer	ANEURALNETWORKS_TRANSPOSE_CONV_2D	TRANSPOSE_CONV	-	ConvTranspose
depthhwise_convolution	vxConvolutionLayer	ANEURALNETWORKS_DEPTHWISE_CONV_2D	DEPTHWISE_CONV_2D	Depthwise Convolution	-
depthwise_conv1d	vx_kernel (DEPTHWISE_CONV1D)	-	-	-	-
group_conv1d	vx_kernel (CONV1D)	-	-	-	-

Table continues on the next page...

Table 6. Neural-network operations and supported API functions (continued)

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
Log_softmax	vx_kernel (LOG_SOFTMAX)	ANEURALNETWORKS_LOG_SOFTMAX	LOG_SOFTMAX	-	Logsoftmax
dilated_convolution	vxConvolutionLayer	-	-	-	-
l2pooling	vxPoolingLayer	ANEURALNETWORKS_L2_POOL	L2_POOL_2D	Pooling2D/L2	-
max_pool	vxPoolingLayer	ANEURALNETWORKS_MAX_POOL	MAX_POOL_2D	Pooling2D/max	MaxPool
max_pool_with_argmax	vx_kernel (POOLWITHARGMAX)	-	-	-	-
max_unpool	vx_kernel (UPSAMPLE)	-	-	-	-
Others					
argmax	vx_kernel (ARGMAX)	ANEURALNETWORKS_ARGMAX	ARGMAX	-	ArgMax
argmin	vx_kernel (ARGMIN)	ANEURALNETWORKS_ARGMIN	ARGMIN	-	ArgMin
dequantize	vxTensorCopyNode	ANEURALNETWORKS_DEQUANTIZE	DEQUANTIZE	Dequantize	DequantizeLinear
quantize	-	ANEURALNETWORKS_QUANTIZE	QUANTIZE	Quantize	QuantizeLinear
image_process	vx_kernel (IMAGE_PROCESS)	-	-	-	-
region_proposal	vxRPNLayer	-	-	-	-
roi_pool	vxROIPoolingLayer	ANEURALNETWORKS_ROI_ALIGN	-	-	-
shuffle_channel	vx_kernel (SHUFFLE_CHANNEL)	ANEURALNETWORKS_CHANNEL_SHUFFLE	-	-	-
tile	-	ANEURALNETWORKS_TILE	TILE	-	Tile
svdf	-	ANEURALNETWORKS_SVDF	SVDF	-	-

Table continues on the next page...

Table 6. Neural-network operations and supported API functions (continued)

Op Category/Name	OpenVX API 1.2	Android NNAPI 1.2	TensorFlow Lite 2.3.1	Arm NN 20.02.01	ONNX 1.1.2
embedding_lookup	-	ANEURALNETWORKS_EMBEDDING_LOOKUP	EMBEDDING_LOOKUP	-	-
cast	-	ANEURALNETWORKS_CAST	CAST	-	Cast

Appendix B
OVXLIB Operation Support with GPU

This section provides a summary of the neural network OVXLIB operations supported by the NXP Graphics Processing Unit (GPU) IP with hardware support for OpenVX and OpenCL and a compatible Software stacks. OVXLIB operations are listed in the following table.

The following abbreviations are used for format types:

- **asym-u8**: asymmetric_affine-uint8
- **asym-i8**: asymmetric_affine-int8
- **fp32**: float32
- **pc-sym-i8**: perchannel_symmetric_int8
- **h**: half
- **bool8**: bool8
- **int16**: int16
- **int32**: int32

Table 7. OVXLIB operation support with GPU

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
Basic Operations					
VSI_NN_OP_CONV2D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
VSI_NN_OP_CONV1D	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
VSI_NN_OP_DEPTHWISE_CONV1D	asym-u8	asym-u8	asym-u8	✓	
	asym-i8	asym-i8	asym-i8	✓	
VSI_NN_OP_DECONVOLUTION	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
VSI_NN_OP_FCL	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
Activation Operations					
VSI_NN_OP_ELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_HARD_SIGMOID	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SWISH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_LEAKY_RELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_PRELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_RELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_RELUN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_RSQRT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SIGMOID	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SOFTRELU	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SQRT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_TANH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_ABS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_CLIP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_EXP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	h		h	✓	✓
VSI_NN_OP_LOG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_NEG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_MISH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SOFTMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_LOG_SOFTMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SQUARE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SIN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
Elementwise Operations					
VSI_NN_OP_ADD	asym-u8		asym-u8	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SUBTRACT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_MULTIPLY	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_DIVIDE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_MAXIMUN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_MINIMUM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_POW	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_FLOORDIV	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_MATRIXMUL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_RELATIONAL_OPS	asym-u8		bool8	✓	✓
	asym-i8		bool8	✓	✓
	fp32		bool8	✓	✓
	h		bool8	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_LOGICAL_OPS	bool8		bool8	✓	✓
VSI_NN_OP_SELECT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
	bool8		bool8	✓	✓
VSI_NN_OP_ADDN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
Normalization Operations					
VSI_NN_OP_BATCH_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_LRN	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_LRN2	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_L2_NORMALIZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_L2NORMALZESCALE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_LAYER_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_BATCHNORM_SINGLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_MOMENTS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
Reshape Operations					
VSI_NN_OP_SLICE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_SPLIT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_CONCAT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_STACK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_UNSTACK	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_RESHAPE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SQUEEZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_PERMUTE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_REORG	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	h		h	✓	✓
VSI_NN_OP_SPACE2DEPTH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_DEPTH2SPACE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_BATCH2SPACE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SPACE2BATCH	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_PAD	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_REVERSE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_STRIDED_SLICE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_CROP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_REDUCE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_ARGMX	asym-u8		asym-u8/int16/int32	✓	✓
	asym-i8		asym-u8/int16/int32	✓	✓
	fp32		int32	✓	✓
	h		asym-u8/int16/int32	✓	✓
VSI_NN_OP_ARGMIN	asym-u8		asym-u8/int16/int32	✓	✓
	asym-i8		asym-u8/int16/int32	✓	✓
	fp32		int32	✓	✓
	h		asym-u8/int16/int32	✓	✓
VSI_NN_OP_SHUFFLECHANNEL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
RNN Operations					
VSI_NN_OP_LSTMUNIT_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
VSI_NN_OP_LSTM	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	pc-sym-i8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_GRUCELL_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
VSI_NN_OP_GRU_OVXLIB	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
VSI_NN_OP_SVDF	asym-u8	asym-u8	asym-u8	✓	✓
	asym-i8	p8	asym-i8	✓	✓
	fp32	fp32	fp32	✓	✓
	h	h	h	✓	✓
Pooling Operations					
VSI_NN_OP_ROI_POOL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_POOLWITHARGMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_UPSAMPLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
Miscellaneous Operations					
VSI_NN_OP_PROPOSAL	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	
	fp32		fp32	✓	
	h		h	✓	

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
VSI_NN_OP_VARIABLE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_DROPOUT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_RESIZE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_DATACONVERT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_A_TIMES_B_PLUS_C	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_FLOOR	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_EMBEDDING_LOOKUP	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_GATHER	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	h		h	✓	✓
VSI_NN_OP_GATHER_ND	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_TILE	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_RELU_KERAS	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_ELTWISEMAX	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_FCL2	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_POOL	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓
VSI_NN_OP_SIGNAL_FRAME	asym-u8		asym-u8	✓	
	asym-i8		asym-i8	✓	

Table continues on the next page...

Table 7. OVXLIB operation support with GPU (continued)

OVXLIB Operations	Tensors			Execution Engine	
	Input	Kernel	Output	OpenVX	OpenCL
	fp32		fp32	✓	
	h		h	✓	
VSI_NN_OP_CONCATSHIFT	asym-u8		asym-u8	✓	✓
	asym-i8		asym-i8	✓	✓
	fp32		fp32	✓	✓
	h		h	✓	✓

Appendix C
OVXLIB Operation Support with NPU

This section provides a summary of the neural network OVXLIB operations supported by the NXP Neural Processor Unit (NPU) IP and a compatible Software stacks. OVXLIB operations are listed in the following table.

The following abbreviations are used for format types:

- **asym-u8**: asymmetric_affine-uint8
- **asym-i8**: asymmetric_affine-int8
- **fp32**: float32
- **pc-sym-i8**: perchannel_symmetric-int8
- **h**: half
- **bool8**: bool8
- **int16**: int16
- **int32**: int32

The following abbreviations are used to reference key Execution Engines (NPU) in the hardware:

- **NN**: **N**eural-**N**etwork Engine
- **PPU**: **P**arallel **P**rocessing **U**nit
- **TP**: **T**ensor **P**rocessor

Table 8. OVXLIB operation support with NPU

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
Basic Operations						
VSI_NN_OP_CONV2D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	h	h	h			✓
VSI_NN_OP_CONV1D	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓
	fp32	fp32	fp32			✓
	h	h	h			✓
VSI_NN_OP_DEPTHWISE_CONV1D	asym-u8	asym-u8	asym-u8			✓
	asym-i8	asym-i8	asym-i8			✓
VSI_NN_OP_DECONVOLUTION	asym-u8	asym-u8	asym-u8	✓		
	asym-i8	pc-sym-i8	asym-i8	✓		✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp32	fp32	fp32			✓
	h	h	h			✓
VSI_NN_OP_FCL	asym-u8	asym-u8	asym-u8		✓	
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	h	h	h		✓	
Activation Operations						
VSI_NN_OP_ELU	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_HARD_SIGMOID	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_SWISH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_LEAKY_RELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_PRELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_RELU	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_RELUN	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_RSQRT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_SIGMOID	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_SOFTRELU	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_SQRT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_TANH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_ABS	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_CLIP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	h		h			✓
VSI_NN_OP_EXP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_LOG	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_NEG	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_MISH	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_SOFTMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_LOG_SOFTMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_SQUARE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_SIN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp32		fp32			✓
	h		h			✓
Elementwise Operations						
VSI_NN_OP_ADD	asym-u8		asym-u8	✓		
	asym-i8		asym-i8	✓		
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_SUBTRACT	asym-u8		asym-u8	✓		
	asym-i8		asym-i8	✓		
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_MULTIPLY	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_DIVIDE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_MAXIMUN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_MINIMUM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_POW	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_FLOORDIV	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_MATRIXMUL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_RELATIONAL_OPS	asym-u8		bool8			✓
	asym-i8		bool8			✓
	fp32		bool8			✓
	h		bool8			✓
	bool8		bool8			✓
VSI_NN_OP_LOGICAL_OPS	bool8		bool8			✓
VSI_NN_OP_SELECT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
	bool8		bool8			✓
VSI_NN_OP_ADDN	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
Normalization Operations						
VSI_NN_OP_BATCH_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_LRN	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_LRN2	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_L2_NORMALIZE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_L2NORMALZESCALE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_LAYER_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_BATCHNORM_SINGLE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_MOMENTS	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
Reshape Operations						

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_SLICE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_SPLIT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_CONCAT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_STACK	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_UNSTACK	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_RESHAPE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_SQUEEZE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_PERMUTE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	h		h		✓	
VSI_NN_OP_REORG	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_SPACE2DEPTH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_DEPTH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
	bool8		bool8			
VSI_NN_OP_BATCH2SPACE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_SPACE2BATCH	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_PAD	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_REVERSE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_STRIDED_SLICE	asym-u8		asym-u8		✓	

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_CROP	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_REDUCE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_ARGMAX	asym-u8		asym-u8/int16/int32			✓
	asym-i8		asym-u8/int16/int32			✓
	fp32		int32			✓
	h		asym-u8/int16/int32			✓
VSI_NN_OP_ARGMIN	asym-u8		asym-u8/int16/int32			✓
	asym-i8		asym-u8/int16/int32			✓
	fp32		int32			✓
	h		asym-u8/int16/int32			✓
VSI_NN_OP_SHUFFLECHANNEL	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
RNN Operations						
VSI_NN_OP_LSTMUNIT_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	h	h	h		✓	✓
VSI_NN_OP_LSTM	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	h	h	h		✓	✓
VSI_NN_OP_GRUCELL_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	h	h	h		✓	✓
VSI_NN_OP_GRU_OVXLIB	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	h	h	h		✓	✓
VSI_NN_OP_SVDF	asym-u8	asym-u8	asym-u8		✓	✓
	asym-i8	pc-sym-i8	asym-i8		✓	✓
	fp32	fp32	fp32			✓
	h	h	h		✓	✓
Pooling Operations						
VSI_NN_OP_ROI_POOL	asym-u8		asym-u8		✓	✓
	asym-i8		asym-i8		✓	✓
	fp32		fp32			✓
	h		h		✓	✓
VSI_NN_OP_POOLWITHARGMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_UPSAMPLE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
Miscellaneous Operations						
VSI_NN_OP_PROPOSAL	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_VARIABLE	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_DROPOUT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_RESIZE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_DATACONVERT	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_A_TIMES_B_PLUS_C	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_FLOOR	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
VSI_NN_OP_EMBEDDING_LOOKUP	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_GATHER	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_GATHER_ND	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_TILE	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_RELU_KERAS	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_ELTWISEMAX	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_INSTANCE_NORM	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_FCL2	asym-u8		asym-u8		✓	
	asym-i8		asym-i8		✓	
	fp32		fp32			✓

Table continues on the next page...

Table 8. OVXLIB operation support with NPU (continued)

OVXLIB Operations	Tensors			Execution Engine (NPU)		
	Input	Kernel	Output	NN	TP	PPU
	h		h		✓	
VSI_NN_OP_POOL	asym-u8		asym-u8	✓	✓	
	asym-i8		asym-i8	✓	✓	
	fp32		fp32			✓
	h		h		✓	
VSI_NN_OP_SIGNAL_FRAME	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓
VSI_NN_OP_CONCATSHIFT	asym-u8		asym-u8			✓
	asym-i8		asym-i8			✓
	fp32		fp32			✓
	h		h			✓

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: 31 December 2020

Document identifier: IMXMLUG

arm