Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Qualcomm ThunderSoft®

SNPE Training Part 2

Aaron Yue

2021

Copyright 2008-2020 Thunder Software Co., Ltd.
Company Confidential

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Contents

- Review
- An Image Classifiers Demo
- SNPE Introduction
- SNPE Workflow
- Supported Chipsets / Supported Network Layers
- User-defined Operations (UDO) Workflow
- Limitations
- CPU vs GPU vs DSP
- Run SNPE on Linux Machine
- Building and Running the C++ Application on ARM Android
- Thermal Measurement
- Solve Problem with SNPE
- Q & A

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT
Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Review

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Question 1

What are the two phases of Deep Learning?

Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Answer

- Training phase – Process for machine to learn and optimize a model from data
  - Collecting and preparing data
  - Creating a model
  - Training the model
- Inference phase – Use trained model to predict outcomes from new observations in an efficient way
  - Evaluate model
  - Improving the performance

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Question 2

What Are the Results of Testing Object Detection Inference Speed with Different Runtimes ?

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Answer : Let's test it together.

Runtimes	Time Consuming Per Time (ms)	FPS(Frame Per Second)
CPU		
GPU		
DSP		

Accuracy changes?

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Test inference speed with different Runtimes

Runtimes	Time Consuming Per Time (ms)	FPS(Frame Per Second)
CPU	220	4.55
GPU	46.7	21.4
DSP	13.0	76.9

Accuracy changes?

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT
Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

An Image Classifiers Demo

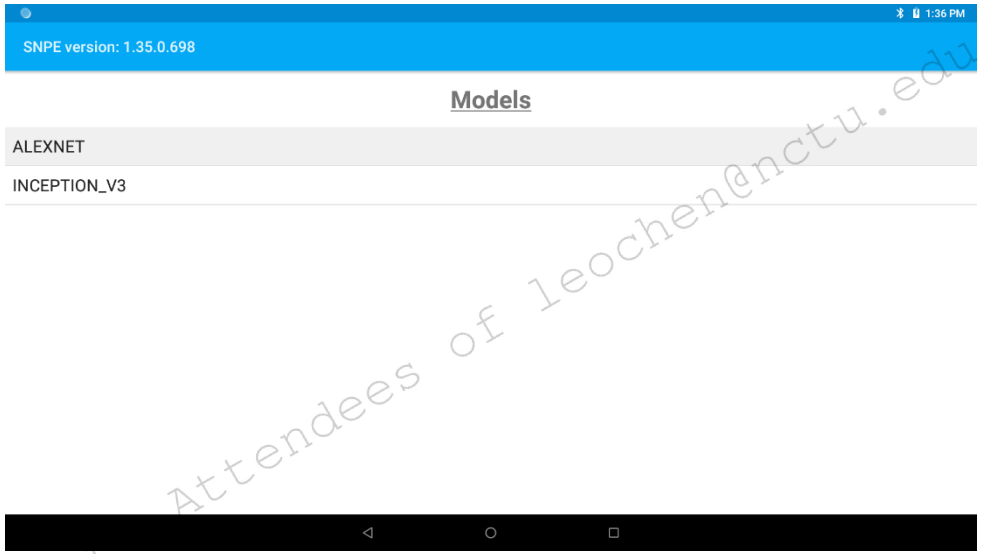
Please keep this document for personal use only
DO NOT DISTRIBUTE IT

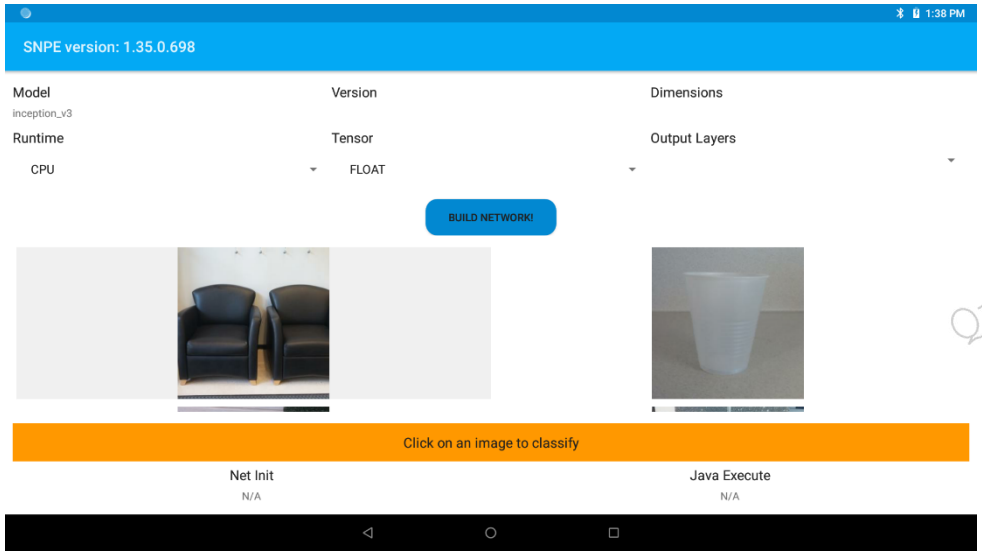
Image Classifiers Sample

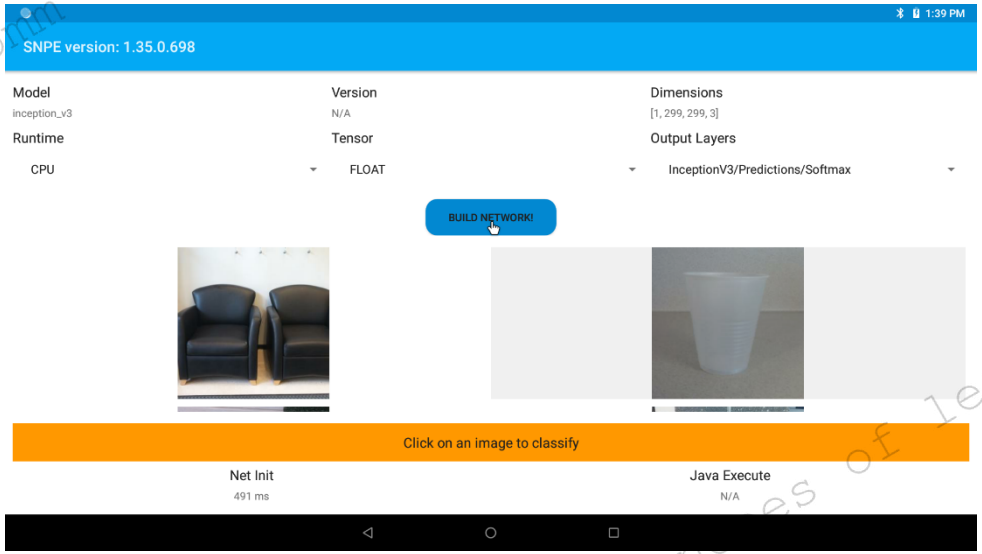
- The SNPE Android SDK includes a sample application that showcases the SDK features. The application source code is in:
  - $SNPE_ROOT/examples/android/image-classifiers
- Note that SNPE provides the following AAR file which include necessary binaries:
  - snpe-release.aar: Native binaries compiled with clang using libc++ STL
- Please set environment variable SNPE_AAR to this AAR file.
- The following slides provide screenshot of the samples:

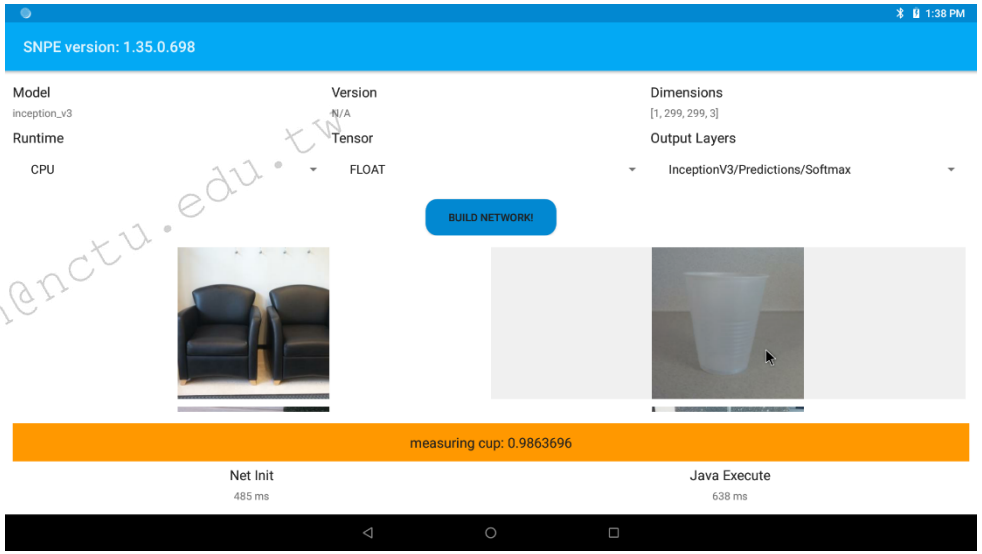
Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Image Classifiers Sample (cont.)





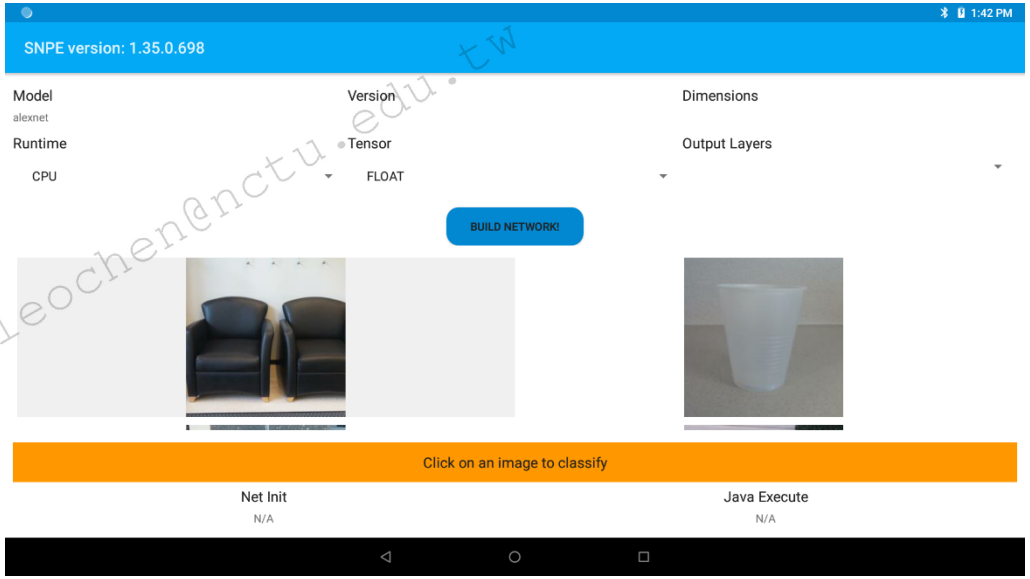


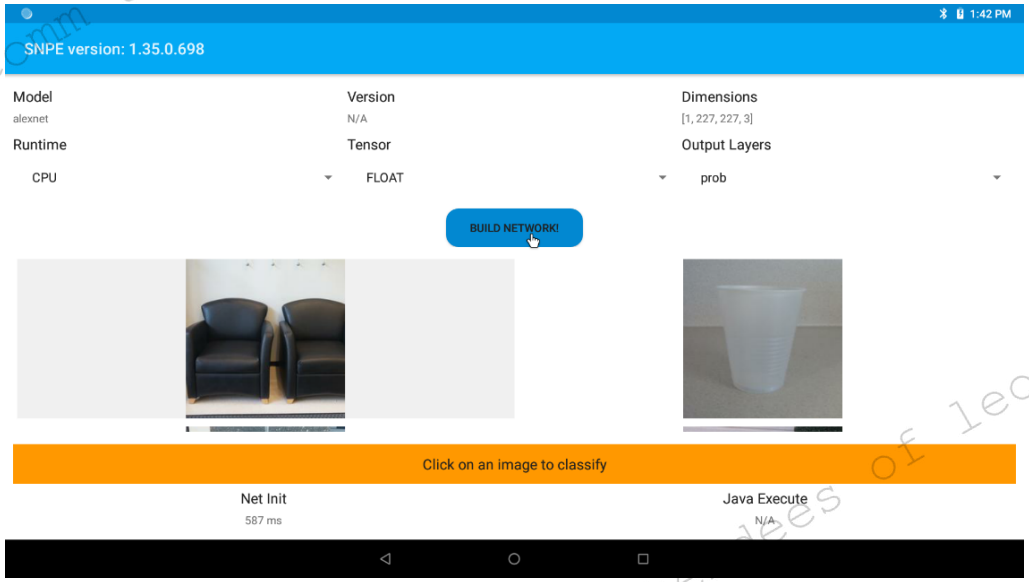


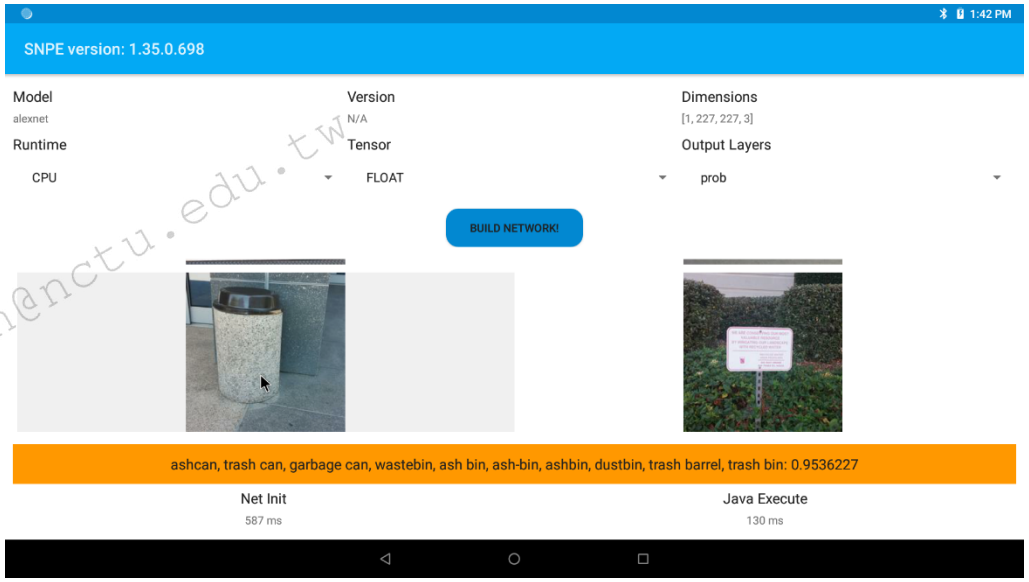
Qualcomm t Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Image Classifiers Sample (cont.)







Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Image Classifiers Sample

- To build this sample, include the SNPE SDK AAR as described above and build with the following commands.
  - export SNPE_AAR=snpe-release.aar
  - cd $SNPE_ROOT/examples/android/image-classifiers
  - bash ./setup_alexnet.sh
  - bash ./setup_inceptionv3.sh
  - cp ../../../android/$SNPE_AAR app/libs/snpe-release.aar
  - ./gradlew assembleDebug

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Sample Build

- **Note:**
  - To build the sample, import the network model and sample images by invoking the **setup_models.sh** script as described above.
  - If building produces the error **gradle build failure due to "SDK location not found"**, set the environment variable ANDROID_HOME to point to your sdk location.
  - Building the sample code with gradle requires java 8.
- After the build successfully completes, the output APK can be found in the application build folder:
- $SNPE_ROOT/examples/android/image-classifiers/app/build/outputs/apk

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Hands-on

Compile and Run Image Classifiers

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

SNPE Introduction

Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

SNPE Brief Introduction

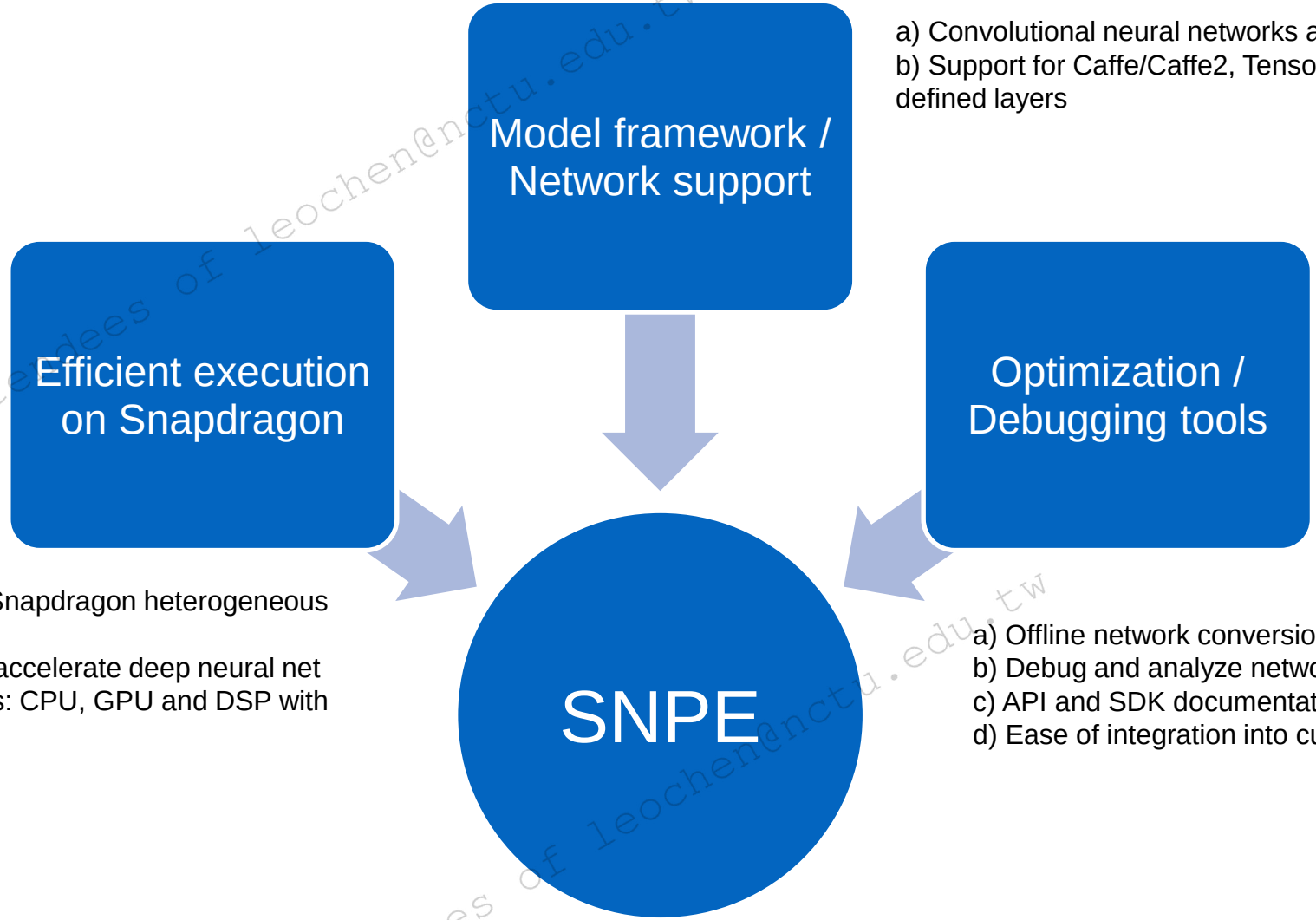
- SNPE is a Qualcomm Snapdragon software accelerated runtime for the execution of deep neural networks (for inference)

- With SNPE, users can:
  - Convert Caffe, Caffe2 and TensorFlow models to a SNPE deep learning container (DLC) file
  - Quantize DLC files to 8 bit fixed point for execution on the Qualcomm® Hexagon™ DSP/HVX HTA
  - Integrate a network into applications and other code via C++ or Java
  - Execute the network on the Snapdragon CPU, the Qualcomm® AdrenoTM GPU, or the Hexagon DSP with HVX* support
  - Execute an arbitrarily deep neural network
  - Debug the network model execution on x86 Ubuntu Linux
  - Debug and analyze the performance of the network model with SNPE tools
  - Benchmark a network model for different targets

Note*: ADSP with HVX support or CDSP where dedicated for HVX alone. For more information refer to the supported Snapdragon devices section.

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Software Accelerated Runtime for Execution of Deep Neural Networks on Device



a) Convolutional neural networks and LSTMs
b) Support for Caffe/Caffe2, TensorFlow, and user/developer defined layers

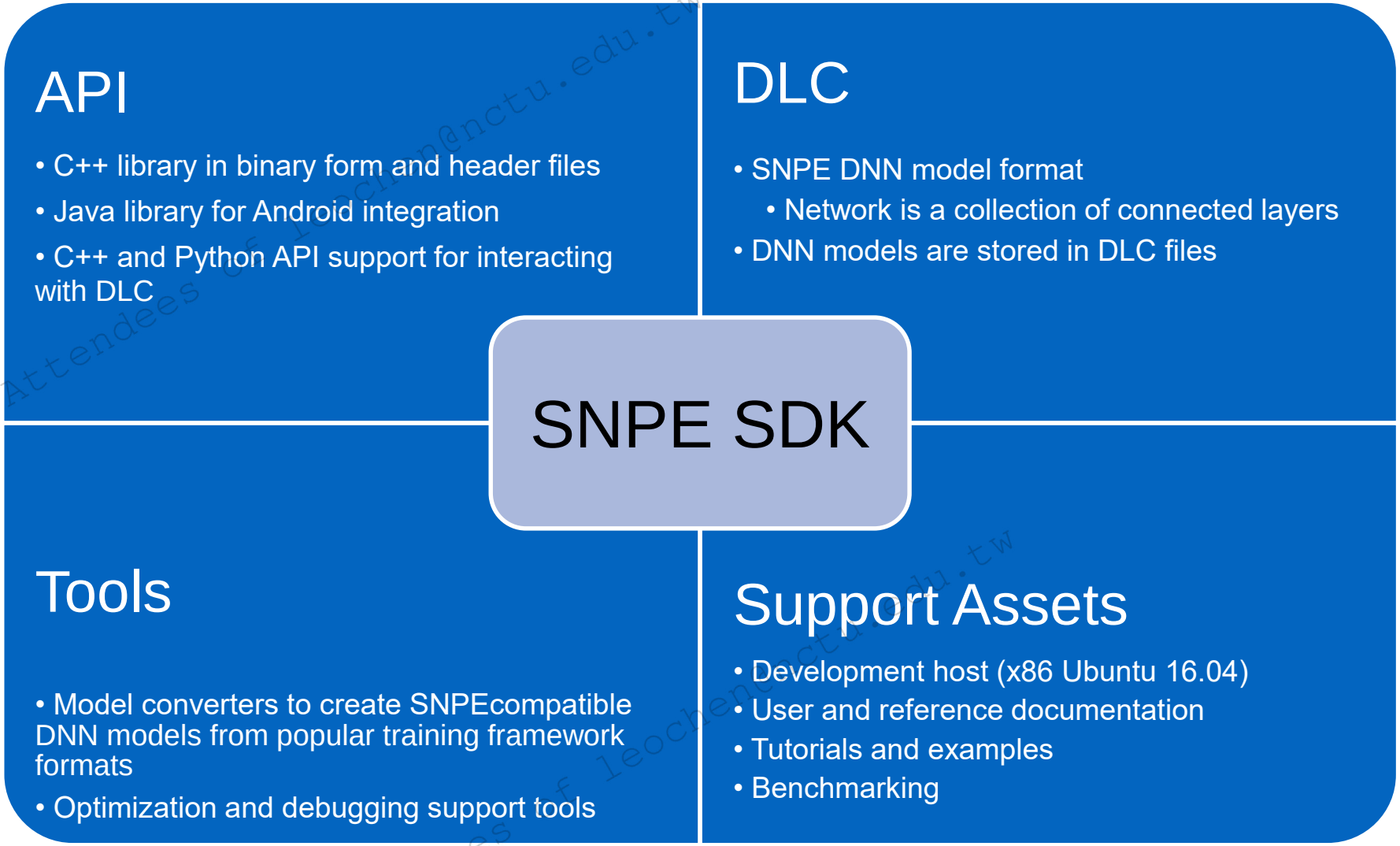
a) Takes advantages of Snapdragon heterogeneous computing capabilities
b) Runtime and libraries accelerate deep neural net processing on all engines: CPU, GPU and DSP with vector extensions

a) Offline network conversion tools
b) Debug and analyze network performance
c) API and SDK documentation with sample code
d) Ease of integration into customer applications

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

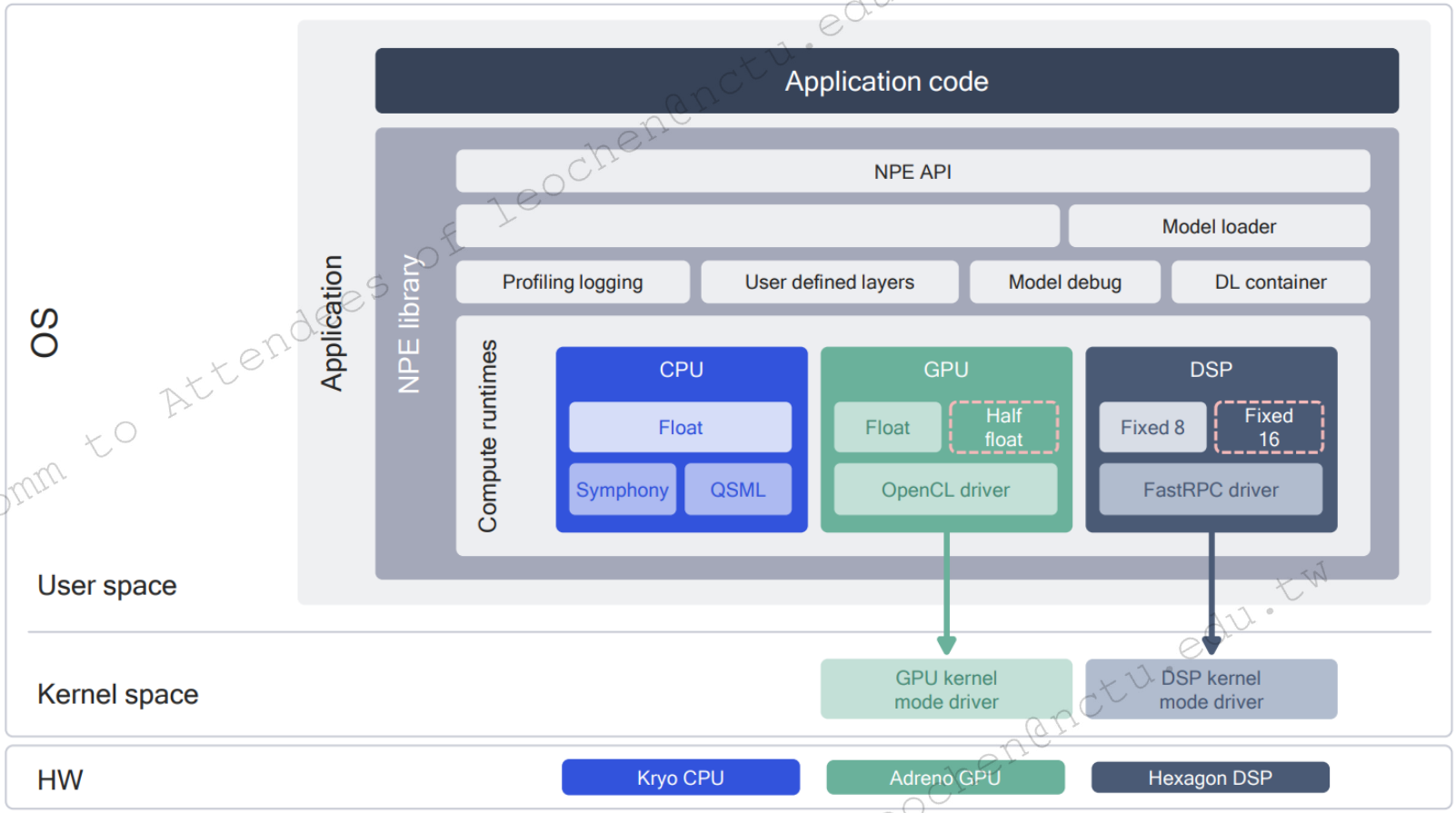
Elements of SNPE SDK



Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

SNPE Architecture



User developed

Future

OS support:

- ARM Android
- ARM Linux
- x86 Linux

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

SNPE Runtime Characteristics

Runtime	Math	Characteristics	DLC requirements
CPU	32 bit floating point math and data	• Not highly optimized • Supports all SNPE layers	Supports any DLC contents (for example, can dequantize data if stored in DLC as fixed point data)
GPU	Half-hybrid Math is 32-bit floating point Data load/store as 16-bit float values	• Highly optimized • Near parity with CPU in layer support	Supports any DLC contents (for example, can dequantize data if stored in DLC as fixed point data)
DSP	8-bit fixed point math and data	• Highly optimized (using HVX based NN-graph from DSP team) • Some GPU/CPU layers not supported on DSP • Some layers are supported on DSP and not GPU For more information on layer support, see the Supported Network Layers section.	Requires fixed point quantization metadata for both weights/biases and activations (however, can quantize DLC data if stored as floats)
AIP	8-bit fixed point math and data	• Highly optimized (using HTA based aix-graph from HTA team) • Some GPU/CPU layers not supported on HTA	Requires fixed point quantization metadata for both weights/biases and activations Requires HTA specific metadata

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Supported Snapdragon Devices

Snapdragon Device	CPU	GPU	DSP	AIP
Qualcomm Snapdragon 855	Yes	Yes	Yes (CDSP*)	Yes
Qualcomm Snapdragon 845	Yes	Yes	Yes (CDSP*)	No
Qualcomm Snapdragon 835	Yes	Yes	Yes (ADSP)	No
Qualcomm Snapdragon 821	Yes	Yes	Yes (ADSP)	No
Qualcomm Snapdragon 820	Yes	Yes	Yes (ADSP)	No
Qualcomm Snapdragon 670	Yes	Yes	Yes (CDSP*)	No
Qualcomm Snapdragon 660	Yes	Yes	Yes (CDSP*)	No
Qualcomm Snapdragon 652	Yes	Yes	No	No
Qualcomm Snapdragon 630	Yes	Yes	No	No
Qualcomm Snapdragon 636	Yes	Yes	No	No
Qualcomm Snapdragon 625	Yes	Yes	No	No
Qualcomm Snapdragon 450	Yes	Yes	No	No

Note*: CDSP is a SNPE-supported DSP platform. For complete list of supported snapdragon devices, see the SDK Documentation overview page in <workspace>/snpe-<version>/doc/html/overview.html.

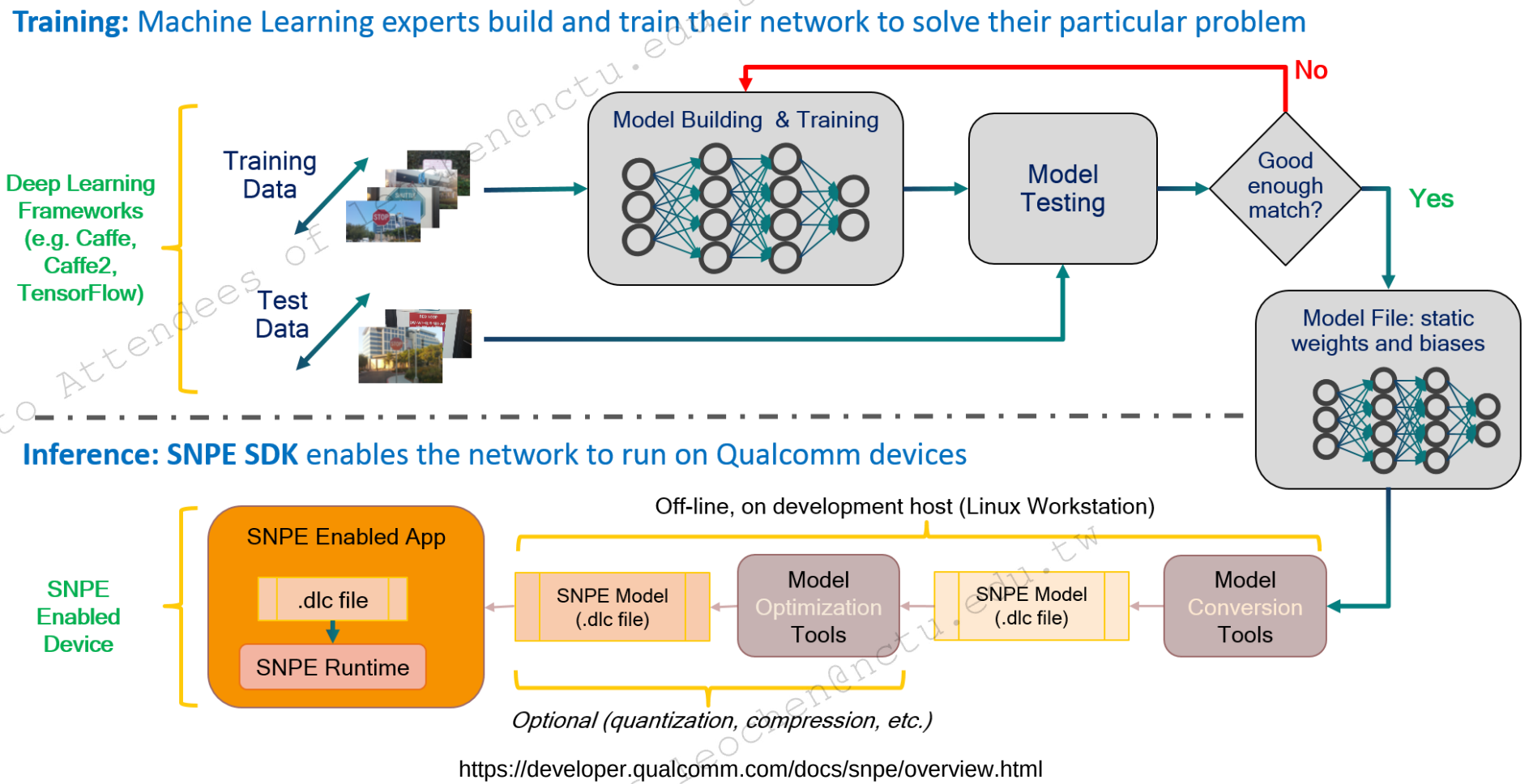
Please keep this document for personal use only
DO NOT DISTRIBUTE IT

SNPE Workflow

Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Model to Runtime Workflow



https://developer.qualcomm.com/docs/snpe/overview.html

Note: SNPE primarily enables the already trained network models to run on the CPU, GPU, and DSP and outputs inferences with optimal execution time. Training a network model is the responsibility of OEMs.

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Basic SNPE Workflow

Convert network

- Convert the network model to a DLC file that can be loaded by SNPE
- Optionally quantize for running on the Hexagon DSP

Offline

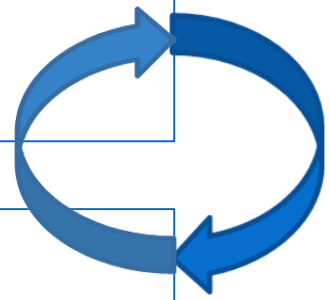
Load network

- Instantiate SNPE interface and load DLC file
- Select desired runtime (for example, CPU, GPU, DSP, HTA), and other runtime options

Online

Prepare input data

- Prepare input data for the model (for example, preprocessing to match model input data layer dimensions, quantizing input data for DSP)



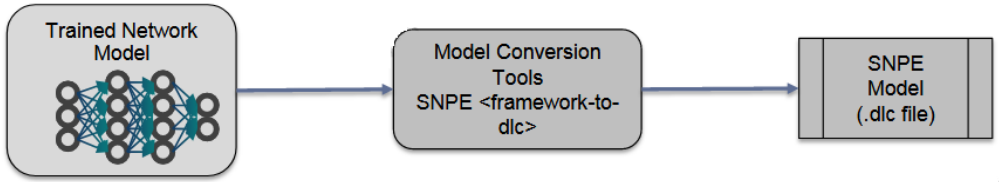
Run network

- Execute the model using SNPE runtime, one input at the time

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Converting a Network Model

- Use snpe-<framework>-to -dlc to convert the network model to DLC
  - Use snpe-Caffe-to-dlc to convert Caffe-based models
  - Use snpe-Caffe2-to-dlc to convert Caffe2-based models
  - Use snpe-tensorflow-to-dlc to convert TensorFlow-based models
- It is as simple as running a Python script
- Input to the script is the model in the originating training framework format
- Output is a DLC file with the converted model that can be passed directly into the SNPE



80-NL315-18 Qualcomm® Snapdragon™ Neural Processing Engine (SNPE) Overview

- For additional details on conversion tools and syntax, refer to the SDK documentation in <workspace>/snpe-<version>/doc/html/tools.html

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Quantizing a Model

- Default output of snpe-<framework>-to-dlc is a non-quantized model
  - All the network parameters are left in an 32 floating point representation as present in the original model
- Quantization is a process of converting a network model into an 8-bit fixed point format which is optimal to run on the DSP
- snpe-dlc-quantize is the offline tool that converts non-quantized DLC models into 8-bit quantized
- DLC models
- Quantization method
  - Static quantization of weights, biases, and activations with support for asymmetric dynamic range and arbitrary step size (similar to TensorFlowstyle quantization)
- Quantized model is necessary for SNPE runtimes that support only quantized fixed point arithmetic (such as DSP HTA)

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

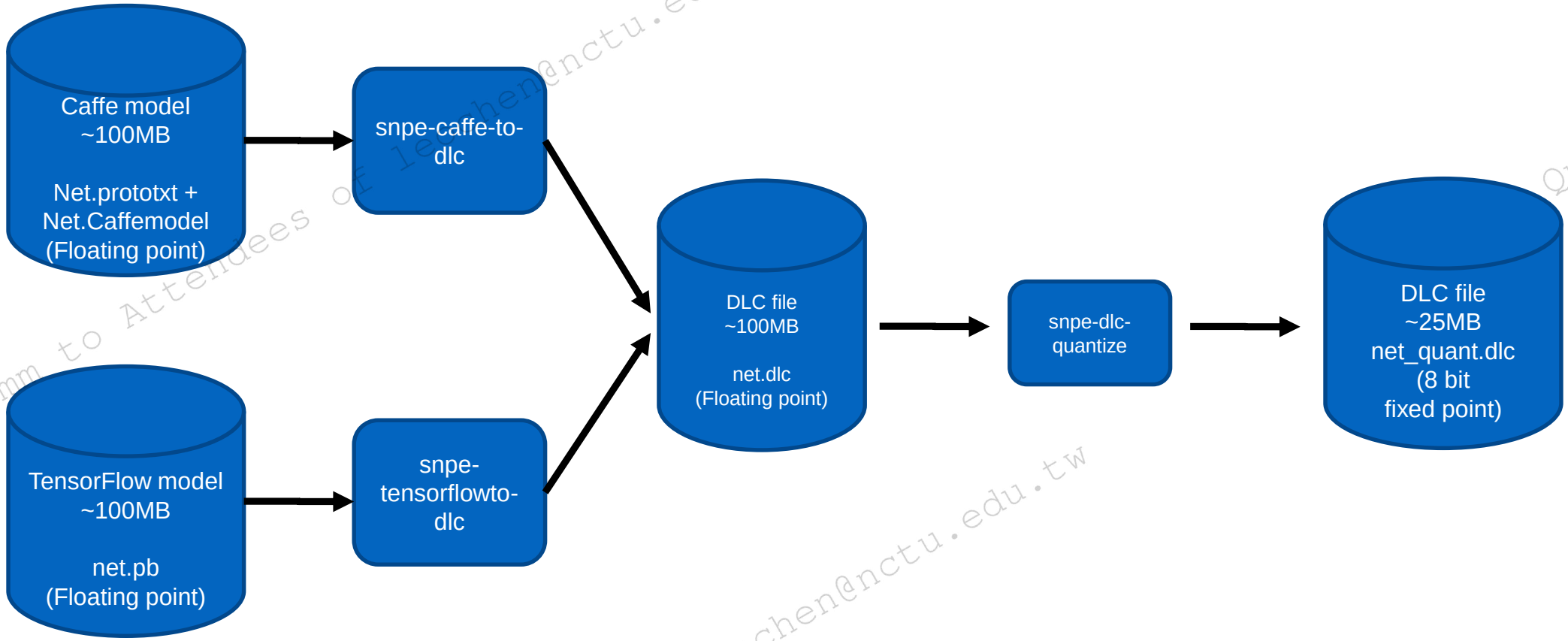
Choosing Between a Quantized or Non-Quantized Model

- CPU and GPU
  - The GPU and CPU always use floating point (non-quantized) network parameters
  - If network initialization time is a concern, it is recommended to use non-quantized DLC files (default) for both GPU and CPU
  - Quantization of the DLC file does introduce noise, as quantization is lossy
- DSP
  - The DSP always uses quantized network parameters
  - Using a non-quantized DLC file on the DSP is supported. Network initialization time will dramatically increase as the SNPE will automatically quantize the network parameters in order to run on the DSP.
- HTA
  - The HTA always uses quantized network parameters
- For additional details on quantization, refer to the SDK documentation in <workspace>/snpe<version>/doc/html/quantized_models.html

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

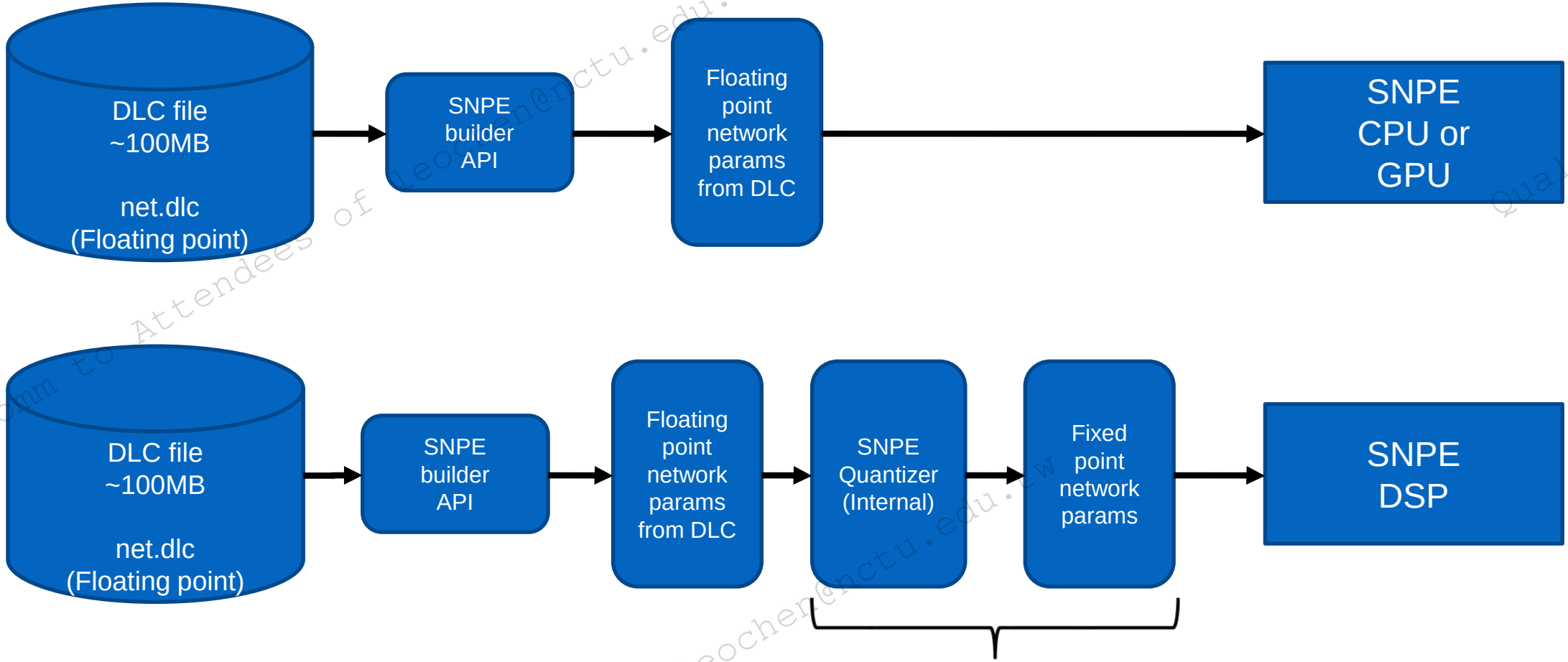
Quantizing a Model - Offline - Tensorflow OR Caffe models



Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

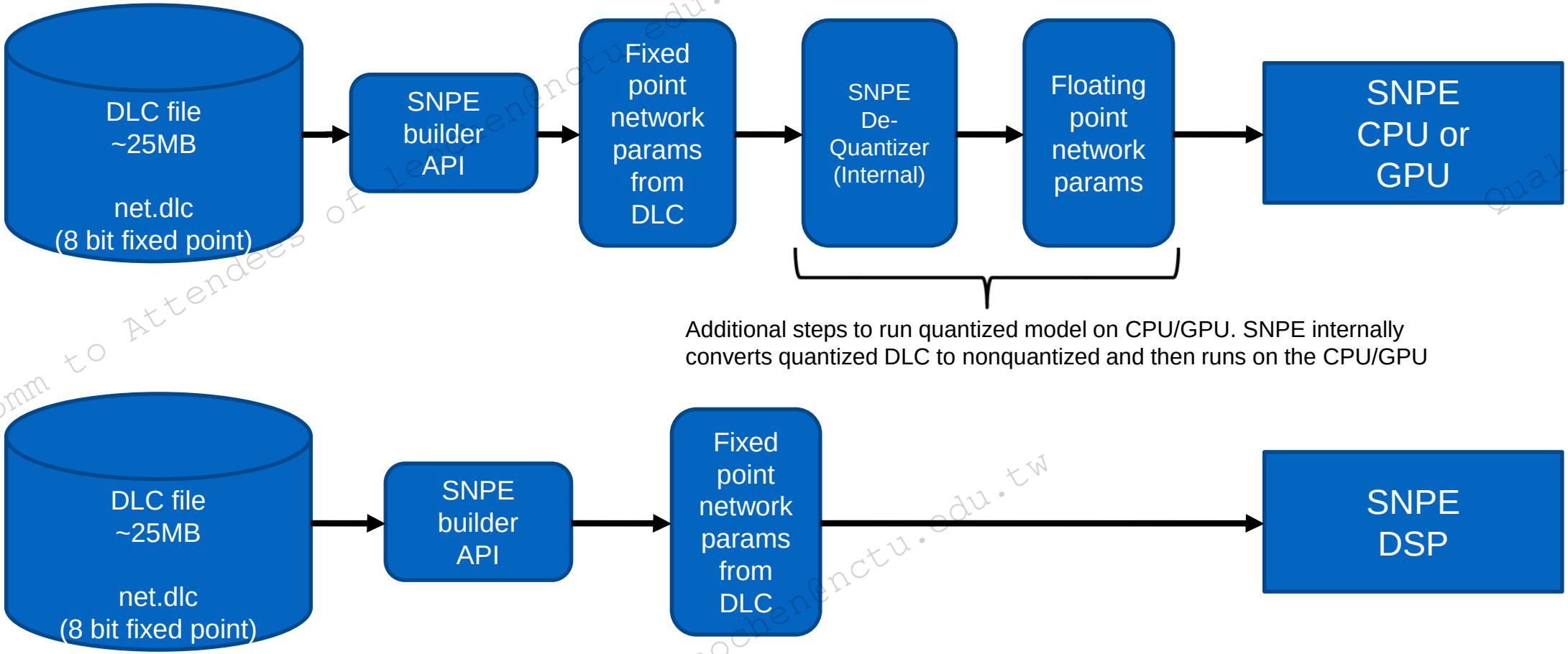
Initializing SNPE with non-Quantized DLC



Additional steps to run non-quantized model on DSP. SNPE internally converts nonquantized DLC to quantized and then runs on the DSP

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Initializing SNPE with Quantized DLC



Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

After Network Creation - Running an Image through the Network

SNPE Execute() API

Input image in floating point

SNPE CPU or GPU

Floating point input image

Quantization of the input image occurs regardless of whether the original DLC file was quantized or nonquantized. This is currently done on the CPU before sending it to the DSP.

SNPE Execute() API

Input image in floating point

SNPE Quantizer (Internal)

Input image in fixed point

SNPE DSP

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Load Network

- SNPE SDK has APIs to:
  - Instantiate SNPE runtime
  - Load DLC file
  - Select runtime (CPU, GPU, DSP or AIP)
- The runtime must be selected where a network model has to be executed
- SNPE provides APIs to query whether a particular runtime is available or not and to set the selected runtime if its available
- SNPE SDK has C++ and android example API usage tutorials :
  - <workspace>/snpe-<version>/doc/html/cplus_plus_tutorial.html
  - <workspace>/snpe-<version>/doc/html/android_tutorial.html

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Prepare Input Data

- SNPE requires the input image to be in a specific format that is different from Caffe or TensorFlow networks
- In SNPE, the image must be presented as a tensor of shape (height x width x channel), where channel is the fastest-changing dimension
- The channel order used during inference must be the same as that used during training. For example, Imagenet models trained in Caffe require a channel order of BGR.
- For SNPE, for network models which are trained with images where a mean is subtracted, the mean image must be manually subtracted before the image data is written to the input layer
- For additional details on input data processing, refer the SDK documentation in <workspace>/snpe<version>/doc/html/image_input.html

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Prepare Input Data (cont.)

- The following image shows the two different input image memory layouts required by Caffe and SNPE for the bvlc_alexnet model. The input image size is 227x227.

https://developer.qualcomm.com/docs/snpe/image_input.html

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Prepare Input Data (cont.)

- SNPE provides support for common image preprocessing operations such as color space conversion (for example, NV21 to BGR format), scaling, cropping and mean subtraction on all supported runtimes
- These operations are added as layers to the network and are performed as part of the forward propagate pipeline
- These image preprocessing operations are currently only supported for DLC networks converted from a Caffe model
- For additional details on supported operations and its syntax, refer to the SDK Documentation in

  <workspace>/snpe-<version>/doc/html/input_preprocessing.html

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT
Qualcomm to Attendees of leochen@nctu.edu.tw

Supported Chipsets / Supported Network Layers

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Supported Chipsets

- Android:

  Snapdragon 865, 855, 845, 835, 821, 820, 765, 720G, 712, 710, 675, 670, 660, 653, 652, 650, 636, 632, 630, 626, 625, 450, 439, 429, QCS605, QCS403, SM7150, SM6125, SXR1130

- Linux:

  SDM820A, APQ626, APQ625, QCS405, QCS403

More details, please refer to the document:

kba-200308032408_1_snpe_1.36.0_partner_release_notes.pdf

Qualcomm Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Supported Network Layers

Layer type	Description	CPU	GPU	DSP/ AIP
Batch normalization (+ scaling)	Batch normalization followed by scaling operation. Batch norm operation can be performed by itself or in combination with scaling.	✓	✓	✓
Color space conversion	Converts input image color format (encoding type) into SNPE native color space. Color space conversion parameters are provided as an option to the model converter tool.	✓	✓	✓
Concatenation	This layer concatenates multiple inputs into a single output	✓	✓	✓
Convolution	Computes dot products between the entries of the filter and the input at any position	✓	✓	✓
Crop	Crops one layer to the dimensions of a reference layer	✓	✓	✓
CrossMap response normalization	This is an option within LRN layer	✓	✓	✓
Deconvolution	Performs deconvolution operation	✓	✓	✓
Depthwise convolution	Performs a 2D depthwise convolution	✓	✓	✓
Dropout	Layer is used for training only. Converters remove this layer from DLC creation.	NA	NA	NA
Elementwise	Supports SUM, PROD, and MAX mode with coefficients	✓	✓	✓

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Supported Network Layers(cont.)

Layer type	Description	CPU	GPU	DSP/ AIP
Flatten	Flatten an input to a layer	✓	✓	✓
Fully connected	Similar to convolution, but with connections to full input region (that is, the filter size is exactly the size of the input volume)	✓	✓	✓
Input	This is an input layer to the network	✓	✓	✓
Local response normalization (LRN)	Performs a lateral inhibition by normalizing over local input regions	✓	✓	✓
LSTM	LSTM recurrent network cell	✓	✓	✗
Mean subtraction	Performs image mean subtraction on the input	✓	✓	✓
Output	There is no explicit output layer as the results from any layer in the network can be specified as an output when loading a network	NA	NA	NA
Permute	Permute is used to rearrange the dimensions of a tensor	✓	✓	✓
Pooling	Pooling operation down samples the input volume spatially. Both average and max pooling are supported.	✓	✓	✓
Prelu	Activation function – prelu [i.e., y = max(0, x) + a*min(0,x)]	✓	✓	✓

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Supported Network Layers(cont.)

Layer type	Description	CPU	GPU	DSP/ AIP
Proposal	Outputs region proposals, usually for consumption of an ROIPooling layer. Typically used in Faster RCNN.	✓	✗	✓
Relu	Activation function – relu [i.e., y = max(0,x)]	✓	✓	✓
Reshape	Change dimensions of the input to a layer	✓	✓	✓
ROI Pooling	Region of interest pooling. Typically used in Faster RCNN.	✓	✗	✓
Sigmoid	Activation function – sigmoid [i.e., y = 1/(1 + exp(-x)]	✓	✓	✓
Tanh	Activation function – tanh [i.e., y = tanh(x)]	✓	✓	✓
Scale	Input image scaling, maintains aspect ratio. This function is primarily intended for images, but technically any 2D input data can be processed if it makes sense. Scaling parameters are provided as an option to the model converter tool.	✓	✓	✓
Silence	Silence is handled and removed from the model during conversion, similar to Dropout	✓	✓	✓
Slice	Slices an input layer into multiple output layers.	NA	NA	NA
Softmax	Supports 1D and 2D modes	✓	✓	✓

For additional details on network layers supported refer to the SDK documentation <workspace>/snpe<version>/doc/html/network_layers.html

Please keep this document for personal use only
DO NOT DISTRIBUTE IT
Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

User-defined Operations (UDO) Workflow

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Introduction

- SNPE 1.35.0 introduces User Defined Operations (UDOs).

- SNPE provides the ability for users to plug in custom neural network operations that may not be inherently supported by the runtime engine in the form of User-defined Operations (hereafter referred to as UDO). These could be operations defined in popular training frameworks such as Tensorflow or custom operations that are built based as framework extensions but not available in the SNPE SDK. They can be natively executed on any of the supported hardware accelerators for which they are implemented. SNPE provides the infrastructure to execute these operations in a seamless fashion with little to no overhead compared to executing internally supported operations.

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Introduction (cont.)

Using this feature, users can define additional operations and extend SNPE's capabilities to support additional networks. In other words, it allows user to define network operations that are not supported by the running engine.

- Users can write operations for the various SNPE accelerated runtimes (CPU, GPU and DSP)
- The operations run directly on the accelerator, and don't perform a context switch back to the CPU (like UDL does).
- Operations share attribute information with SNPE for better integration with built-in operations.
- Operations and Operation Packages can be defined with multiple operations for multiple runtimes and platforms.
- A generator tool is provided to create the skeleton for an Operation and Operation Package to make it easier to get started creating UDOs

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Comparing UDO with UDL

- SNPE has existing support for users to include custom operations to be executed on CPU at runtime with User-defined layers, abbreviated as UDL (see User-Defined Layers (UDL) Tutorial). User-defined operations can be considered an enhancement over UDLs for the following reasons:

- 1. Additional Training Frameworks supported

- UDL is supported only with Caffe-based networks. UDOs are supported on Tensorflow and ONNX models with support for Caffe to be added soon.

  - Notice: in snpe 1.36.0, UDO supports Caffe models, UDL support is deprecated, and will be retired in a future release.

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Comparing UDO with UDL (cont.)

- 2. Integration with improved visibility

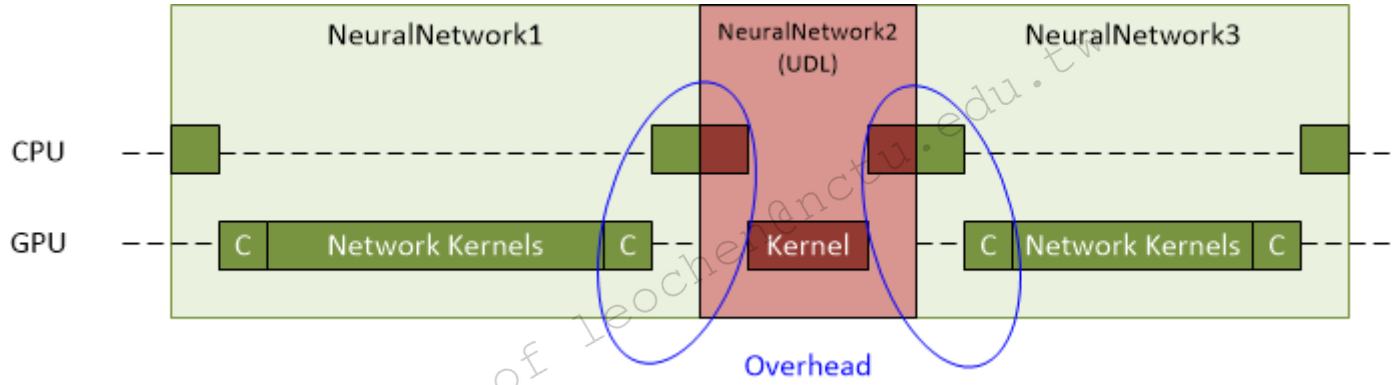
- UDLs are implemented in ways that make attributes of the layers completely opaque to SNPE. This requires them to be handled in total isolation from the rest of the layers in the network and in their own subnets. On the other hand, UDOs are designed to be able to express their attributes with SNPE components while preserving opacity with the actual implementation kernels. This allows SNPE to understand their properties such as input and output tensor dimensions and connect them with their neighboring operations in the network, resulting in optimal network partitions.

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Comparing UDO with UDL (cont.)

- 3. Extended targets for execution
- The UDL mechanism allows users to register a callback to switch the execution context during runtime from SNPE into their proprietary method that implements the operation of their custom layer on the CPU. SNPE partitions the network model into subnets that separate native layers from user-defined ones. In cases when other subnets are scheduled to execute on other hardware accelerators such as the DSP or the GPU, this mechanism induces significant overhead in switching the execution context over to the ARM CPU, which is the only target on which a UDL can run. This mechanism is illustrated in the figure below with the example of a network scheduled to run on the GPU:

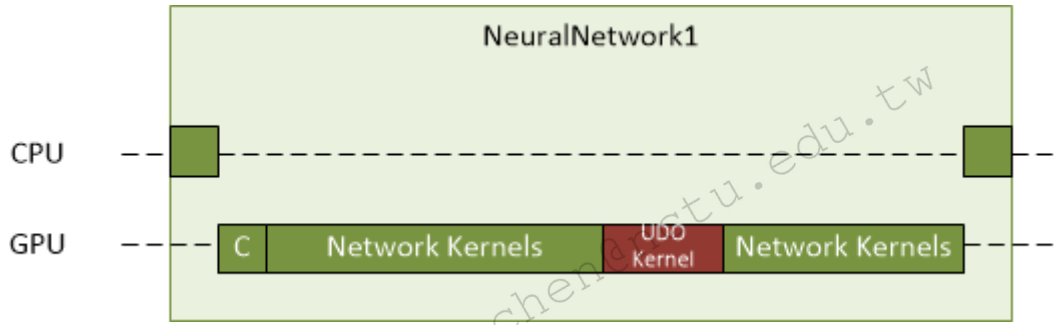


UDL Mechanism

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Comparing UDO with UDL (cont.)

- 3. Extended targets for execution
- The UDO mechanism improves on this approach by allowing users to integrate their custom operations on any supported hardware accelerator by compiling for specific targets such as the GPU or DSP in addition to ARM CPU. The UDO mechanism allows SNPE to construct subnets that may encompass UDOs without having to switch to the CPU by default. It reduces context switching overheads as well as allows users to take advantage of executing their operations on desired accelerators and get superior performance. This mechanism is illustrated in the figure below with the example of a network scheduled to run on the GPU and the UDO compiled for GPU as well:



UDO Mechanism

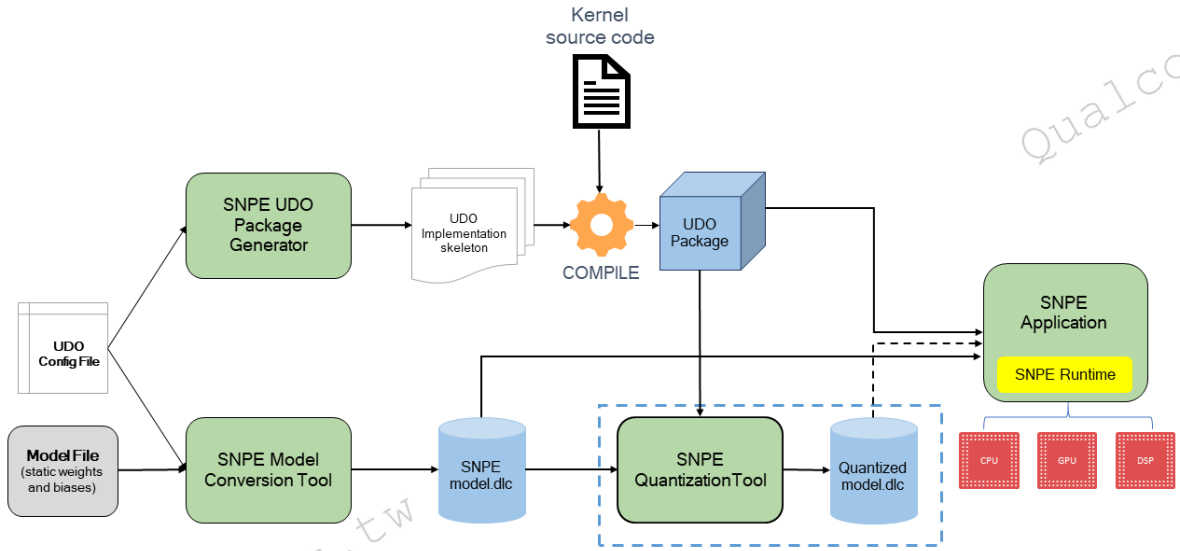
Qualcomm to Attendees of leochen@ntu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow

Workflow in developing and integrating a UDO into the runtime is shown on the right:

- 1st step, Define UDO config file
- Identify the operations in the model that need to be expressed as User-defined operations and describing their attributes through a configuration file.



Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow (cont.)

- Define UDO config file

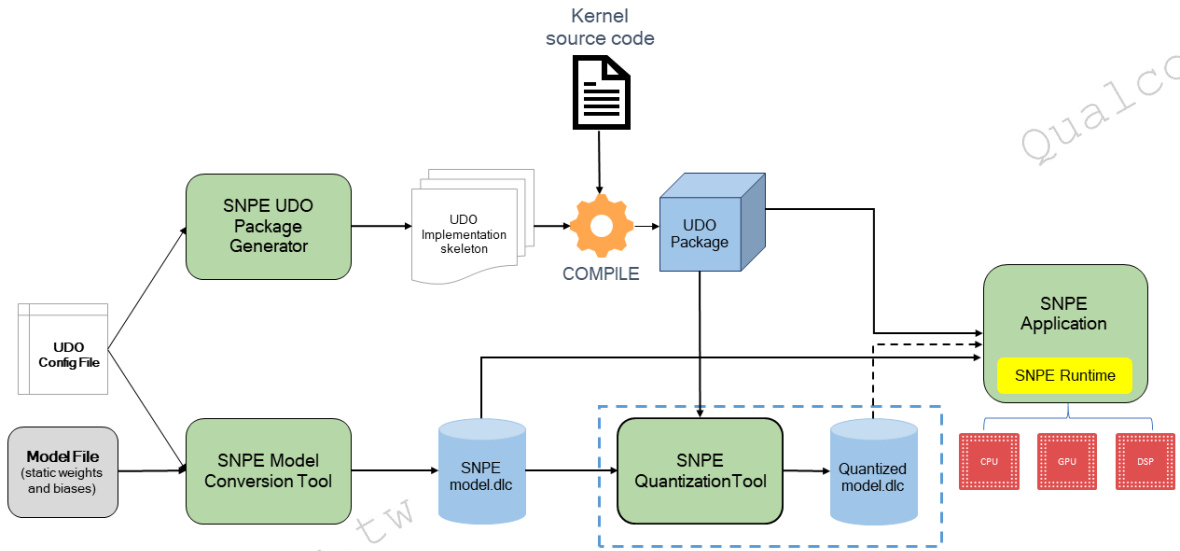
```
{
    "UdoPackage_0":
    {
        "Operators": [
            {
                "type": "",
                "inputs":[
                    {"name":"", "data_type": "FLOAT_32", "static": true,
                        "tensor_layout": "NHWC", "quantization_mode":"",
                        "quantization_params":{"min":0, "max":1}},
                ],
                "outputs":[
                    {"name":"","data_type": "FLOAT_32"}
                ],
                "scalar_params": [
                    {"name":"scalar_param_1", "data_type": "INT_32"}
                ],
                "tensor_params": [
                    {"name":"tensor_param_1", "data_type": "FLOAT_32", "tensor_layout": "NHWC",
                    "quantization_mode":"",
                    "quantization_params":{"min":0, "max":1}},
                ],
                "core_types": ["CPU", "GPU", "DSP"]
            }
        ],
        "UDO_PACKAGE_NAME": "MyCustomUdoPackage",
    }
}
```

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow (cont.)

- 2nd step generate UDO package
- Produce the components of a UDO package by creating source files for the UDO kernels and compiling them against appropriate tool-chains to generate dynamic libraries specific to hardware cores such as the GPU and DSP.
- SNPE provides a tool called snpe-udo-package-generator that assists users in creating common skeleton code for interfacing with SNPE UDO APIs and leaves placeholders for users to fill in the kernel implementation.
- It also generates makefiles for common targets such as X86 and Android and for runtimes per target specified in the config file.



Qualcomm t
Qualcomm to Attendees of leochen@nctu.edu.tw
to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow (cont.)

- Creating a UDO Package
  - Generating UDO Skeleton Code

    snpe-udo-package-generator -p <my_config.json> -o <my-dir>

  - Completing the Registration Skeleton Code

    UDO header files located at: $SNPE_ROOT/share/SnpeUdo/include

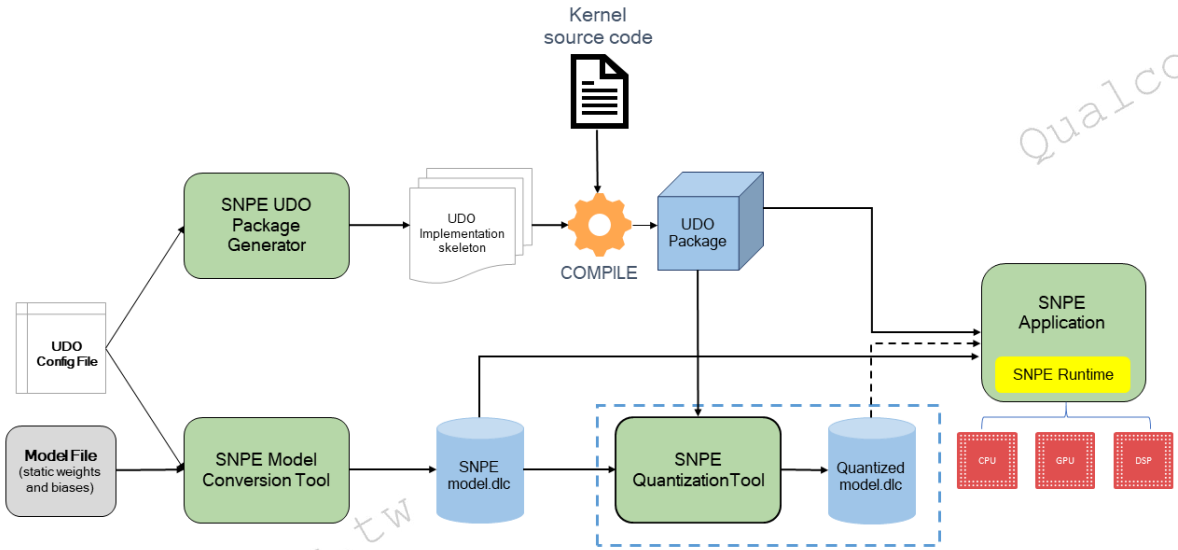
- Compiling a UDO package
  - Implementing a UDO for CPU
  - Implementing a UDO for GPU
  - Implementing a UDO for DSP

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow (cont.)

- 3rd step, model conversion and quantization
- The config file created in the first step is also required to be used by the SNPE model conversion tools along with the actual trained model to allow interpretation of the user-defined operations using definitions from the file.
- The resulting DLC files can then be inspected using tools like snpe-dlc-info to probe the attributes of the UDOs in the model, which is not possible with opaque representations such as with UDLs.

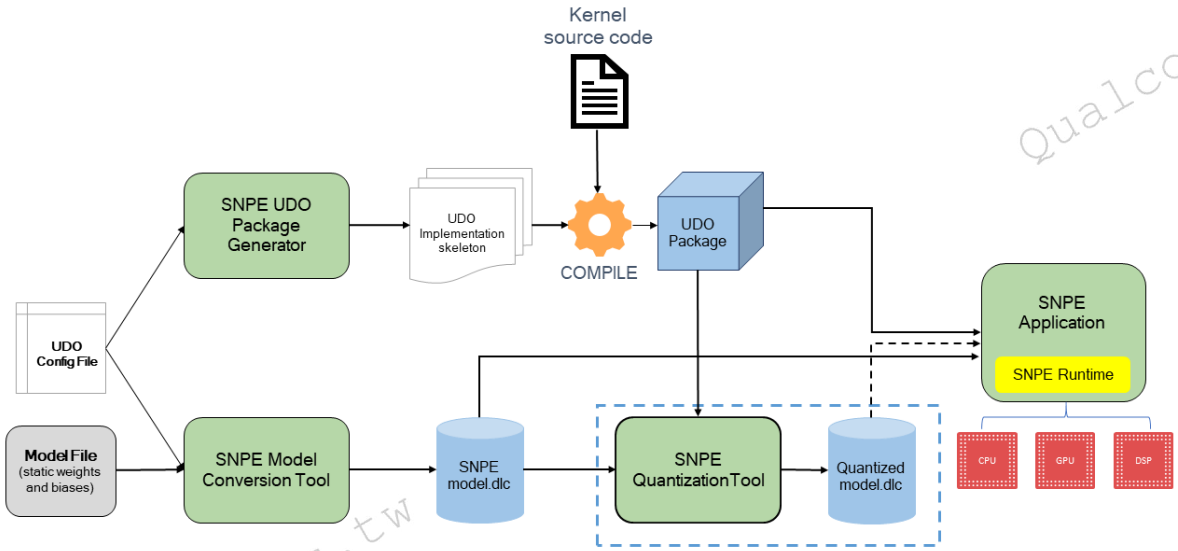


Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow (cont.)

- 3rd step, model conversion and quantization
- Optionally, models with UDOs can also be quantized using SNPE quantization tools to be used with fixed-point runtimes such as DSP. The quantizer tool estimates quantization ranges for activations from all layers in the network including UDOs.
- Since the tool runs offline on an x86 host machine it is required to have a CPU implementation for the UDO in order to perform inference through the entire network. This is also illustrated in dotted lines in the workflow diagram.



Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow (cont.)

- Preparing a model with UDO
  - Converting a network model with UDO into DLC

```
snpe-tensorflow-to-dlc -i <input-tensorflow-model>
                       -d <input-name> <input-dim>
                       --out_node <output-node-name>
                       --udo_config_paths <input-model.json>
                       -o <output-model.dlc>
```

  - Quantizing a DLC with UDO

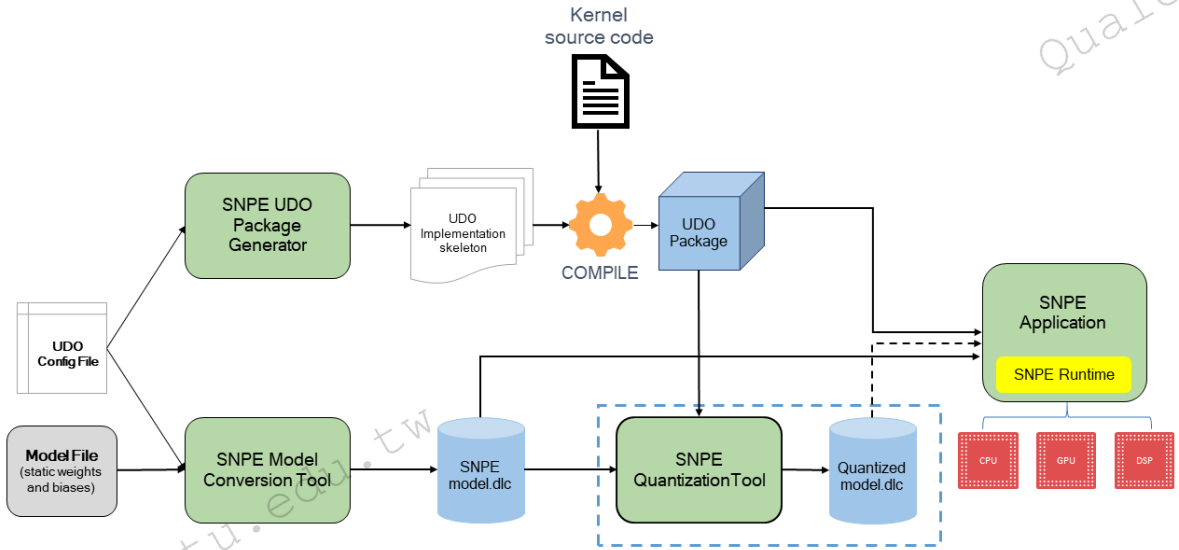
```
snpe-dlc-quantize --input_dlc <model.dlc>
                  --input_list <input-list.txt>
                  --udo_package_path <udo-registration-lib>
                  --output_dlc <quantized-model.dlc>
```

Qualcomm to Attendees of leochen@ncku.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

UDO Workflow (cont.)

- 4th step, actually execute network models with UDOs.
- SNPE applications use the UDO package to register UDO implementations within the process that runs inference on select network models.
- It should be noted that these UDOs can be exercised by multiple instances of SNPE simultaneously without race conditions, which increases the overall throughput for network inference.



Qualcomm to attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running a Native UDO Demo

- Step 1 : the tutorial for execution on Android targets will use the arm64-v8a architecture. This portion of the slides is generic to CPU runtime, other runtimes (GPU, DSP, AIP) are similar to CPU runtime.

```
# architecture: arm64-v8a - compiler: clang - STL: libc++
export SNPE_TARGET_ARCH=aarch64-android-clang6.0
export SNPE_TARGET_STL=libc++_shared.so
```

- Step 2 : push SNPE binaries and libraries to the target device:

```
adb shell "mkdir -p /data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/bin"
adb shell "mkdir -p /data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib"
adb push $SNPE_ROOT/lib/$SNPE_TARGET_ARCH/$SNPE_TARGET_STL
/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib
adb push $SNPE_ROOT/lib/$SNPE_TARGET_ARCH/*.so /data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib
adb push $SNPE_ROOT/bin/$SNPE_TARGET_ARCH/snpe-net-run
/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/bin
```

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running a Native UDO Demo

- Step 3 : push the Inception-V3 UDO model and input data to the device:

```
cd $SNPE_ROOT/models/inception_v3
mkdir data/rawfiles && cp data/cropped/*.raw data/rawfiles/
adb shell "mkdir -p /data/local/tmp/inception_v3_udo"
adb push data/rawfiles /data/local/tmp/inception_v3_udo/cropped
adb push data/target_raw_list.txt /data/local/tmp/inception_v3_udo
adb push dlc/inception_v3_udo.dlc /data/local/tmp/inception_v3_udo
rm -rf data/rawfiles
```

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running a Native UDO Demo

- Step 4 : once the model and data have been placed on the device, place the UDO libraries on the device, take android CPU execution as an example

```
cd $SNPE_ROOT/models/inception_v3

adb shell "mkdir -p /data/local/tmp/inception_v3_udo/cpu"

adb push SoftmaxUdoPackage/libs/arm64-v8a/libUdoSoftmaxUdoPackageImplCpu.so
/data/local/tmp/inception_v3_udo/cpu

adb push SoftmaxUdoPackage/libs/arm64-v8a/libUdoSoftmaxUdoPackageReg.so
/data/local/tmp/inception_v3_udo/cpu

adb push SoftmaxUdoPackage/libs/arm64-v8a/libc++_shared.so /data/local/tmp/inception_v3_udo/cpu
```

Qualcomm to Attendees of leochen@ntu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running a Native UDO Demo

- Step 5: now set required environment variables and run snpe-net-run on device:

```
adb shell
export SNPE_TARGET_ARCH=aarch64-android-clang6.0
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib
export PATH=$PATH:/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/bin
cd /data/local/tmp/inception_v3_udo/
export LD_LIBRARY_PATH=/data/local/tmp/inception_v3_udo/cpu/:$LD_LIBRARY_PATH
snpe-net-run --container inception_v3_udo.dlc --input_list target_raw_list.txt --
udo_package_path cpu/libUdoSoftmaxUdoPackageReg.so
```

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running a Native UDO Demo

- Execute process:

```
aikit:/data/local/tmp/inception_v3_udo # export SNPE_TARGET_ARCH=aarch64-android-clang6.0
aikit:/data/local/tmp/inception_v3_udo # export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib
aikit:/data/local/tmp/inception_v3_udo # export PATH=$PATH:/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/bin
aikit:/data/local/tmp/inception_v3_udo # export LD_LIBRARY_PATH=/data/local/tmp/inception_v3_udo/cpu/:$LD_LIBRARY_PATH
aikit:/data/local/tmp/inception_v3_udo # snpe-net-run --container inception_v3_udo.dlc --input_list target_raw_list.txt --udo_package_path cpu/libUdoSoftmaxUdoPackageReg.so
WARNING: linker: Warning: unable to normalize "data/local/tmp/inception_v3_udo/cpu/"
WARNING: linker: Warning: unable to normalize "data/local/tmp/inception_v3_udo/gpu/"
WARNING: Library libUdoSoftmaxUdoPackageImplGpu.so could not be loaded: 'dlopen failed: library "libUdoSoftmaxUdoPackageImplGpu.so" not found.' Runtime may be impacted.
--------------------------------------------------------------------------------
Model String: N/A
SNPE v1.36.0.746
--------------------------------------------------------------------------------
Processing DNN input(s):
cropped/plastic_cup.raw
Processing DNN input(s):
cropped/notice_sign.raw
Processing DNN input(s):
cropped/chairs.raw
Processing DNN input(s):
cropped/trash_bin.raw
```

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running a Native UDO Demo

- The executable will create the results folder: /data/local/tmp/inception_v3/output. To pull the output:

```
adb pull /data/local/tmp/inception_v3/output output_android
```

- Check the classification results by running the following python script:

```
python3 scripts/show_inceptionv3_classifications.py -i data/target_raw_list.txt -o output_android/
-l data/imagenet_slim_labels.txt
```

- The output should look like the following, showing classification results for all the images.

```
Classification results
cropped/plastic_cup.raw 0.990612 648 measuring cup
cropped/notice_sign.raw 0.167454 459 brass
cropped/chairs.raw      0.382221 832 studio couch
cropped/trash_bin.raw   0.684573 413 ashcan
```

- For more details and sample codes, please refer to UDO tutorial chapter:
  - https://developer.qualcomm.com/docs/snpe/udo_overview.html

Please keep this document for personal use only
DO NOT DISTRIBUTE IT
Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Limitations

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Limitations

- General SNPE Limitations
  - SNPE currently supports 4D input data, where the first dimension is batch.
  - Only batch of 1 is supported for RCNN networks like Faster-RCNN. See Layer Limitations below.
- General CPU Runtime Limitations
  - Not all layers have been optimized for the CPU runtime. For example, deconvolution is dramatically slower than convolution.

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Limitations(cont.)

General GPU Runtime Limitations

- In GPU_FLOAT32_16_HYBRID mode, the GPU kernels use HALF_FLOAT precision for all intermediate data handling and FULL_FLOAT precision for all of its computations. While this does not typically affect mAP for networks that are being used for classification this can overflow/underflow which can impact use of the engine for uses other than classification. If an impact is observed, try running with the CPU runtime which is always FULL_FLOAT to validate any overflow/underflow issues.
- In GPU_FLOAT16 mode, the GPU kernels use HALF_FLOAT precision for all intermediate data handling and all of its computations. In this mode, due to lower computation precision comparing to GPU_FLOAT32_16_HYBRID, chances of negative impact on network's accuracy (e.g. mAP score) are higher. Users are encouraged to test accuracy performance of their network using this mode to ensure it meets requirements of their use case.
- For absolute size restrictions, the concept of “packed” channels refers to the number of channels divided by 4, and rounded up to the nearest integer:
- packed_channels = ceil(channels / 4.0)
- Whenever a layer has a 4-dimensional (i.e. batch x width x height x channels) component, such as input, output, or weight tensor, that component will have the following size restrictions:
- Number of packed channels * width < MaxPerGPUSize
- For all layers that have weights/biases, restrictions are:
- Filter size * filter size * 4 <= MaxPerGPUSize
- Number of output channels / 4 <= MaxPerGPUSize
- **The MaxPerGPUSize is dependent on Qualcomm Adreno™ GPU type and the values are given below**
- **A330: 8192**
- **A430, A530: 16384**
- While loading any network, GPU runtime may choose to merge (squash) few layers with the previous layers in the network, depending on the compatibility of the layers. This results in missing performance information for the squashed layers.

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Limitations(cont.)

- How to quickly check Snapdragon chipset's GPU type
- https://en.wikipedia.org/wiki/List_of_Qualcomm_Snapdragon_systems-on-chip

Model number	CPU (ARMv8)	GPU	DSP	Model number	CPU (ARMv8)	GPU	DSP
MSM8996 Lite (820)	2 + 2 cores (1.804 GHz + 1.363 GHz Kryo)	Adreno 530 510 MHz (407.4 GFLOPS)	Hexagon 680 1 GHz	MSM8998 (835)	4 + 4 cores (2.45 GHz + 1.9 GHz Kryo 280)	Adreno 540 710/670 MHz (737/686 GFLOPS)	Hexagon 682
MSM8996 (820)	2 + 2 cores (2.15 GHz + 1.593 GHz Kryo)	Adreno 530 624 MHz (498.5 GFLOPS)		SDM845	4 + 4 cores (2.8 GHz Kryo 385 Gold+ 1.8 GHz Kryo 385 Silver)	Adreno 540 710/670 MHz (737/686 GFLOPS)	Hexagon 685
MSM8996 Pro-AB(821)	2 + 2 cores (2.342 GHz + 1.6/2.188 GHz Kryo)	Adreno 530 653 MHz (519.2 GFLOPS)		SDM850	4 + 4 cores (2.95 GHz Kryo 385 Gold + 1.8 GHz Kryo 385 Silver)		
MSM8996 Pro-AC(821)				SM8150 (855)	Kryo 485 1 + 3 + 4 cores (2.84 GHz + 2.42 GHz + 1.80 GHz)	Adreno 640 585 MHz (954.7 GFLOPs)	Hexagon 690

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Limitations(cont.)

- General DSP Runtime Limitations
  - When using non-quantized models, the first network execution after network initialization may be significantly slower than subsequent executions. To avoid this, use a DLC file that has been quantized by snpe-dlc-quantize.
- General AIP Runtime Limitations
  - If the input layer of a network needs to be processed by HTA the input must be a 4D tensor with shape format as NHWC where the batch dimension N must be 1 and the number of channels C cannot exceed 16.
  - However, one could take advantage of manually partitioning a network to bypass this limitation by having the input layer be processed on the HVX instead. See Adding HTA sections for details on partitioning.

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Limitations(cont.)

- Layer Limitations
  - Batch normalization (+ Scaling)
    - Caffe: Scaling (scale_layer) is optional. If present, it extends functionality of Batch normalization (batch_norm_layer). If not present, batch_norm_layer will still be converted as per Caffe specification.
    - **scale_layer used anywhere else in the network but immediately after the batch_norm_layer is not supported.**
    - 1d (i.e. per-channel) batch normalization: support available only for caffe models. support not available in dsp runtime.
    - Starting in 1.15.0, the caffe converter distinguishes between a batch_norm_layer and an instance_norm_layer using the value of the batchnorm_param use_global_stats. If use_global_stats is set to True the converter will consume the layer as a batch_norm_layer. If use_global_stats is set to False the converter will consume the layer as an instance_norm_layer. It's important to ensure the prototxt used to convert a caffe model with a batch_norm_layer has the value of use_global_stats not defined as "False" (i.e. do not use a training prototxt for conversion of a caffe model with batch_norm_layers in them)

Qualcomm to Attendees of leochen.nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Limitations(cont.)

- Layer Limitations

Color space conversion

- For NV21 input image encoding type, width or height must be multiple of 2. The reason is 4 Y (2wx2h) is sharing one UV pair.

Concatenation

- For GPU runtime, the number of input channels in each of the inputs can assume arbitrary values. However, if one or more of these are not a multiple of 4, performance of the layer will be diminished.

Convolution

- For GPU runtime, when the number of groups is greater than 1, the number of output channels must be a multiple of 4 * the number of groups. For example, with 2 groups, the number of output channels must be a multiple of 8 (4*2=8).

Crop

- For GPU runtime, the number of input channels in each of the inputs must be a multiple of 4.
- Crop on the DSP is not optimized in all cases. Spatial cropping is optimized (cropping height and/or width, leaving other dimensions unchanged)

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Limitations(cont.)

- Layer Limitations
  - Deconvolution
    - For GPU and CPU runtime, the number of output channels (i.e. number of filters) can be any value (not necessarily a multiple of 4).
    - For GPU runtime the following limitations apply:
      - number of packed input channels * number output channels <= MaxPerGPUSize
      - Filter size-X * Filter size-Y <= MaxPerGPUSize
      - Stride <= filter size
  - For DSP runtime, deconvolutions with stride > 4 are not fully optimized.
  - Caffe parameter limitation: dilation and rectangular filters are not supported

More details about limitations, please refer to the website below:

https://developer.qualcomm.com/docs/snpe/limitations.html

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

3 Steps to Use SNPE API

- Step 1: Build SNPE instance
- Step 2: Fill input tensor with data
- Step 3: Inference(execute) and get results

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT
Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

CPU vs GPU vs DSP

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Performance Data on SDM845 & MSM8996 Examples

Test Goal: Inference Speed by network and chipset
Image SRC: ILSVRC_2015(ImageNet Large Scale Visual Recognition Challenge 2015) evaluation set(1-500)
Notice: SNPE 1.25.1 is used to test the data below

SNPE 1.25.1		GPU				DSP				CPU			
HW	Network	Load(ms/frame)	Inference		Total(Load+Inference)	Load(ms/frame)	Inference		Total(Load+Inference)	Load(ms/frame)	Inference		Total(Load+Inference)
			(ms/frame)	(FPS)	(FPS)		(ms/frame)	(FPS)	(FPS)		(ms/frame)	(FPS)	(FPS)
SDM845	AlexNet	12.37	21.39	46.75	29.62	15.28	15.28	65.45	32.72	5.1	104.34	9.58	9.14
	InceptionV3	20.43	89.32	11.20	9.11	34.44	34.44	29.04	14.52	7.73	720.47	1.39	1.37
	VGG16	16.54	274.85	3.64	3.43	86.69	86.69	11.54	5.77	8.9	1290.63	0.77	0.77
Turbox Platform-820	AlexNet	10.86	28.47	35.12	25.43	22.58	19.58	51.07	23.72	2.61	124.25	8.05	7.88
	InceptionV3	25.41	138.52	7.22	6.10	52.34	52.34	19.11	9.55	4.51	624.91	1.60	1.59
	VGG16	24.08	337.90	2.96	2.76	132.18	132.18	7.57	3.78	22.32	1350.9	0.74	0.73

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Run SNPE on Linux Machine

Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Overview

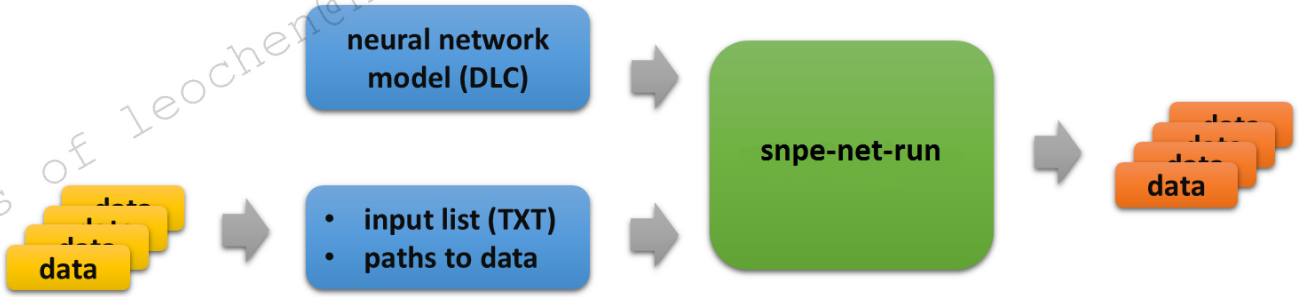
- The example C++ application in this tutorial is called snpe-net-run. It is a command line executable that executes a neural network using SNPE SDK APIs.
- The required arguments to snpe-net-run are:
  - A neural network model in the DLC file format
  - An input list file with paths to the input data.
- Optional arguments to snpe-net-run are:
  - Choice of GPU or DSP runtime (default is CPU)
  - Output directory (default is ./output)
  - Show help description

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Overview(cont.)

snpe-net-run creates and populates an output directory with the results of executing the neural network on the input data.



The SNPE SDK provides Linux and Android binaries of snpe-net-run under

- $SNPE_ROOT/bin/x86_64-linux-clang
- $SNPE_ROOT/bin/arm-android-clang6.0
- $SNPE_ROOT/bin/aarch64-android-clang6.0
- $SNPE_ROOT/bin/aarch64-linux-gcc4.9
- $SNPE_ROOT/bin/arm-linux-gcc4.9sf
- $SNPE_ROOT/bin/aarch64-oe-linux-gcc6.4
- $SNPE_ROOT/bin/arm-oe-linux-gcc6.4hf

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Prerequisites

- The SNPE SDK has been set up following the SNPE Setup chapter.
- Caffe is installed
- Alexnet assets is ready
  - The AlexNet imagenet classification model is trained to classify images with 1000 labels. The examples below shows the steps required to execute a pretrained AlexNet model with snpe-net-run to classify a set of sample images.
  - In order to get alexnet assets, setup_alexnet.py is recommanded :

```
$ python $SNPE_ROOT/models/alexnet/scripts/setup_alexnet.py -a ~/tmpdir -d
```

```
Copying Caffe model
Modiying prototxt to use a batch size of 1
Creating DLC
2020-03-11 11:07:06,620 - 169 - INFO - across channels=True
2020-03-11 11:07:06,620 - 169 - INFO - across channels=True
2020-03-11 11:07:07,497 - 169 - INFO - INFO_DLC_SAVE_LOCATION: Saving model at /home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/dlc/bvlc_alexnet.dlc
2020-03-11 11:07:09,017 - 169 - INFO - INFO_CONVERSION_SUCCESS: Conversion completed successfully
Getting imagenet aux data
Creating ilsvrc_2012_mean.npy
('Creating %s', 'ilsvrc_2012_mean_cropped.bin')
Creating ilsvrc_2012_labels.txt
Create SNPE alexnet input
processing /home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/chairs.jpg
processing /home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/trash_bin.jpg
processing /home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/plastic_cup.jpg
processing /home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/notice_sign.jpg
Create file lists
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/raw_list.txt created listing 4 files.
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/target_raw_list.txt created listing 4 files.
Setup alexnet completed.
```

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Prerequisites(cont.)

After the script is complete the prepared AlexNet assets are copied to the $SNPE_ROOT/models/alexnet directory, along with sample raw images, and converted SNPE DLC files.

```
.
├── caffe
│   ├── bvlc_alexnet.caffemodel
│   ├── deploy_batch_1.prototxt
│   ├── deploy.dlc
│   └── deploy.prototxt
├── data
│   ├── caffe_ilsvrc12.tar.gz
│   ├── chairs.jpg
│   ├── cropped
│   ├── det_synset_words.txt
│   ├── ilsvrc_2012_labels.txt
│   ├── ilsvrc_2012_mean_cropped.bin
│   ├── ilsvrc_2012_mean.npy
│   ├── imagenet.bet.pickle
│   ├── imagenet_mean.binaryproto
│   ├── notice_sign.jpg
│   ├── plastic_cup.jpg
│   ├── synsets.txt
│   ├── synset_words.txt
│   ├── target_raw_list.txt
│   ├── test.txt
│   ├── train.txt
│   ├── trash_bin.jpg
│   └── val.txt
├── dlc
│   └── bvlc_alexnet.dlc
└── scripts
    ├── create_alexnet_raws.py
    ├── create_file_list.py
    ├── setup_alexnet.py
    └── show_alexnet_classifications.py

5 directories, 26 files
```

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Run on Linux Host

- Go to the base location for the model and run snpe-net-run
  - cd $SNPE_ROOT/models/alexnet
  - snpe-net-run --container dlc/bvlc_alexnet.dlc --input_list data/cropped/raw_list.txt

- After snpe-net-run completes, verify that the results are populated in the $SNPE_ROOT/models/alexnet/output directory. There should be one or more .log files and several Result_X directories, each containing a prob.raw file.

```
--------------------------------------------------------------------------------
Model String: N/A
SNPE v1.35.0.698
--------------------------------------------------------------------------------
Processing DNN input(s):
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/plastic_cup.raw
Processing DNN input(s):
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/notice_sign.raw
Processing DNN input(s):
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/chairs.raw
Processing DNN input(s):
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/trash_bin.raw
```

Qualcomm to Attendees of leochen@ncu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Run on Linux Host(cont.)

- One of the inputs is data/cropped/handicap_sign.raw and it was created from data/cropped/handicap_sign.jpg which looks like the following.

- With this input file, snpe-net-run created the output file

  $SNPE_ROOT/models/alexnet/output/Result_0/prob.raw.

- It holds the output tensor data of 1000 probabilities for the 1000 categories. The element with the highest value represents the top classification.

Qualcomm to Attendees of leochen@ntu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Run on Linux Host(cont.)

- We can use a python script to interpret the classification results as follows.

```
python $SNPE_ROOT/models/alexnet/scripts/show_alexnet_classifications.py -i data/cropped/raw_list.txt -o output/ -l
data/ilsvrc_2012_labels.txt
```

- The output should look like the following, showing classification results for all the images.

```
(caffe-2-snpe-dlc) user@user-Z2-R-Series-GK5CP0Z:~/snpe-sdk/models/alexnet$ python $SNPE_ROOT/models/alexnet/scripts/show_alexnet_classificat
ions.py -i data/cropped/raw_list.txt -o output/ -l data/ilsvrc_2012_labels.txt
Classification results
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/plastic_cup.raw 0.720697 647 measuring cup
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/notice_sign.raw 0.666179 458 brass, memorial tablet, plaque
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/chairs.raw      0.363968 831 studio couch, day bed
/home/user/Workspace/Framework/SNPE/snpe-1.35.0.698/models/alexnet/data/cropped/trash_bin.raw   0.950433 412 ashcan, trash can, garbage can,
wastebin, ash bin, ash-bin, ashbin, dustbin, trash barrel, trash bin
```

Note: a) The <input_files_dir> above maps to a path such as /XXX/snpe-x.y.z/models/alexnet/data/cropped/

b) The AlexNet image classification model does not accept jpg files as input. The model expects its input tensor dimension to be 227x227x3 as a float array, see Input Images for more detail. The scripts/setup_alexnet.py script performed a jpg to binary data conversion by calling scripts/create_alexnet_raws.py. The scripts are an example of how jpg images can be preprocessed to generate input for the AlexNet model.

- The output shows the image was classified as “measuring cup" (index 647 of the labels) with a probability of 0.720697. Look at the rest of the output to see the model's classification on other images.

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Building and Running the C++ Application on ARM Android

Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Prerequisites

- You will need the Android NDK to build the Android C++ executable. The tutorial assumes that you can invoke 'ndk-build' from the shell.
- Android NDK r17c is recommended.
- You can download Android NDK by the method below:

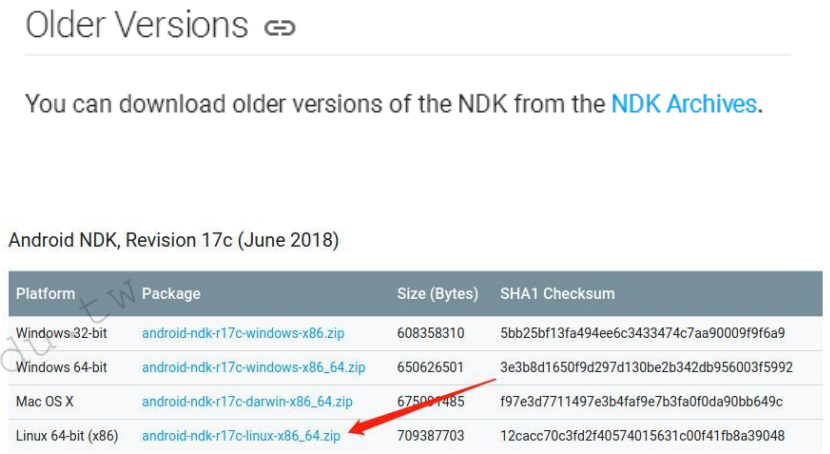
  a) https://developer.android.google.cn/ndk/downloads/

  b) Choose NDK Archieves

  c) Agree the terms and conditions appear

  d) Choose the specific version to download

- Click android-ndk-r17c-linux-x86_64.zip to download.



Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Prerequisites(cont.)

- E.g. the download link is:
  - https://dl.google.com/android/repository/android-ndk-r17c-linux-x86_64.zip
- Extract the package downloaded:
  - $unzip android-ndk-r17c-linux-x86_64.zip
  - Directory android-ndk-r17c is generated
- Add NDK binaries path to the environment
  - $export NDK_PATH=Path to android-ndk-r17c
  - $export PATH=$PATH:$NDK_PATH
- Test whether the ndk-build environment has been installed normally.
  - $which ndk-build
  - The path to android-ndk-r17c will be printed

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Building

- First move to snpe-sample's base directory.
  - $cd $SNPE_ROOT/examples/NativeCpp/SampleCode
- To build snpe-sample with clang/libc++ SNPE binaries (i.e., arm-android-clang6.0 and aarch64-android-clang6.0), use the following command:
  - $cd $SNPE_ROOT/examples/NativeCpp/SampleCode
  - $ndk-build NDK_TOOLCHAIN_VERSION=clang APP_STL=c++_shared
- The ndk-build command will build both armeabi-v7a and arm64-v8a binaries of snpe-sample.
  - $SNPE_ROOT/examples/NativeCpp/SampleCode/obj/local/armeabi-v7a/snpe-sample
  - $SNPE_ROOT/examples/NativeCpp/SampleCode/obj/local/arm64-v8a/snpe-sample

Qualcomm to Attendees of leochen@ctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Push Assets

- Push model data to Android target
- To execute the AlexNet classification model on your Android target follow these steps:
  - cd $SNPE_ROOT/models/alexnet
  - mkdir data/rawfiles && cp data/cropped/*.raw data/rawfiles/
  - adb shell "mkdir -p /data/local/tmp/alexnet"
  - adb push data/rawfiles /data/local/tmp/alexnet/cropped
  - adb push data/target_raw_list.txt /data/local/tmp/alexnet
  - adb push dlc/bvlc_alexnet.dlc /data/local/tmp/alexnet
  - rm -rf data/rawfiles

Note: It may take some time to push the AlexNet dlc file to your target.

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running on ARM Android

- To run the Android C++ executable, push the appropriate SNPE libraries and the executable onto the Android target.
  - $export SNPE_TARGET_ARCH=arm-android-clang6.0
  - $export SNPE_TARGET_ARCH_OBJ_DIR=armeabi-v7a
  - $adb shell "mkdir -p /data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/bin"
  - $adb shell "mkdir -p /data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib"
  - $adb shell "mkdir -p /data/local/tmp/snpeexample/dsp/lib"
  - $adb push $SNPE_ROOT/lib/$SNPE_TARGET_ARCH/ /data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib
  - $adb push $SNPE_ROOT/lib/dsp/ /data/local/tmp/snpeexample/dsp/lib
  - $adb push $SNPE_ROOT/examples/NativeCpp/SampleCode/obj/local/$SNPE_TARGET_ARCH_OBJ_DIR/snpe-sample /data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/bin

Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running on ARM Android(cont.)

- Run snpe-sample with the Alexnet model on the target. This assumes that you have done the setup steps for running Run on Android Target to push to the target all the sample data files and Alexnet model.
  - $adb shell
  - aikit:/ $export SNPE_TARGET_ARCH=arm-android-clang6.0
  - aikit:/ $export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/lib
  - aikit:/ $export PATH=$PATH:/data/local/tmp/snpeexample/$SNPE_TARGET_ARCH/bin
  - aikit:/ $cd /data/local/tmp/alexnet
  - aikit:/ $snpe-sample -b ITENSOR -d bvlc_alexnet.dlc -i target_raw_list.txt -o output_sample
  - aikit:/ $exit

- Pull the target output into a host side output directory.
  - cd $SNPE_ROOT/models/alexnet/
  - adb pull /data/local/tmp/alexnet/output_sample output_sample

```
Batch size for the container is 1
Processing DNN Input: cropped/plastic_cup.raw
Processing DNN Input: cropped/notice_sign.raw
Processing DNN Input: cropped/chairs.raw
Processing DNN Input: cropped/trash_bin.raw
```

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Running on ARM Android(cont.)

- Again we can run the interpret script to see the classification results.
- python $SNPE_ROOT/models/alexnet/scripts/show_alexnet_classifications.py -i data/target_raw_list.txt -o output_sample/ -l data/ilsvrc_2012_labels.txt

```
(caffe-2-snpe-dlc) user@user-Z2-R-Series-GK5CP0Z:~/snpe-sdk/models/alexnet$ python $SNPE_ROOT/models/alexne
t/scripts/show_alexnet_classifications.py -i data/target_raw_list.txt -o output_sample/ -l data/ilsvrc_2012
_labels.txt
Classification results
cropped/plastic_cup.raw 0.720697 647 measuring cup
cropped/notice_sign.raw 0.666181 458 brass, memorial tablet, plaque
cropped/chairs.raw      0.363962 831 studio couch, day bed
cropped/trash_bin.raw   0.950433 412 ashcan, trash can, garbage can, wastebin, ash bin, ash-bin, ashbin, du
stbin, trash barrel, trash bin
```

Qualcomm
Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT
Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Thermal Measurement

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Snapdragon Profiler Overview

- Snapdragon Profiler is profiling software that runs on the Windows, Mac, and Linux platforms. It connects with Android devices powered by Qualcomm® Snapdragon™ processors over USB. Snapdragon Profiler allows developers to analyze CPU, GPU, DSP, memory, power, thermal, and network data, so they can find and fix performance bottlenecks.

- Download method:
  - https://developer.qualcomm.com/software/snapdragon-profiler

Qualcomm to Attendees of leochen@ncu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

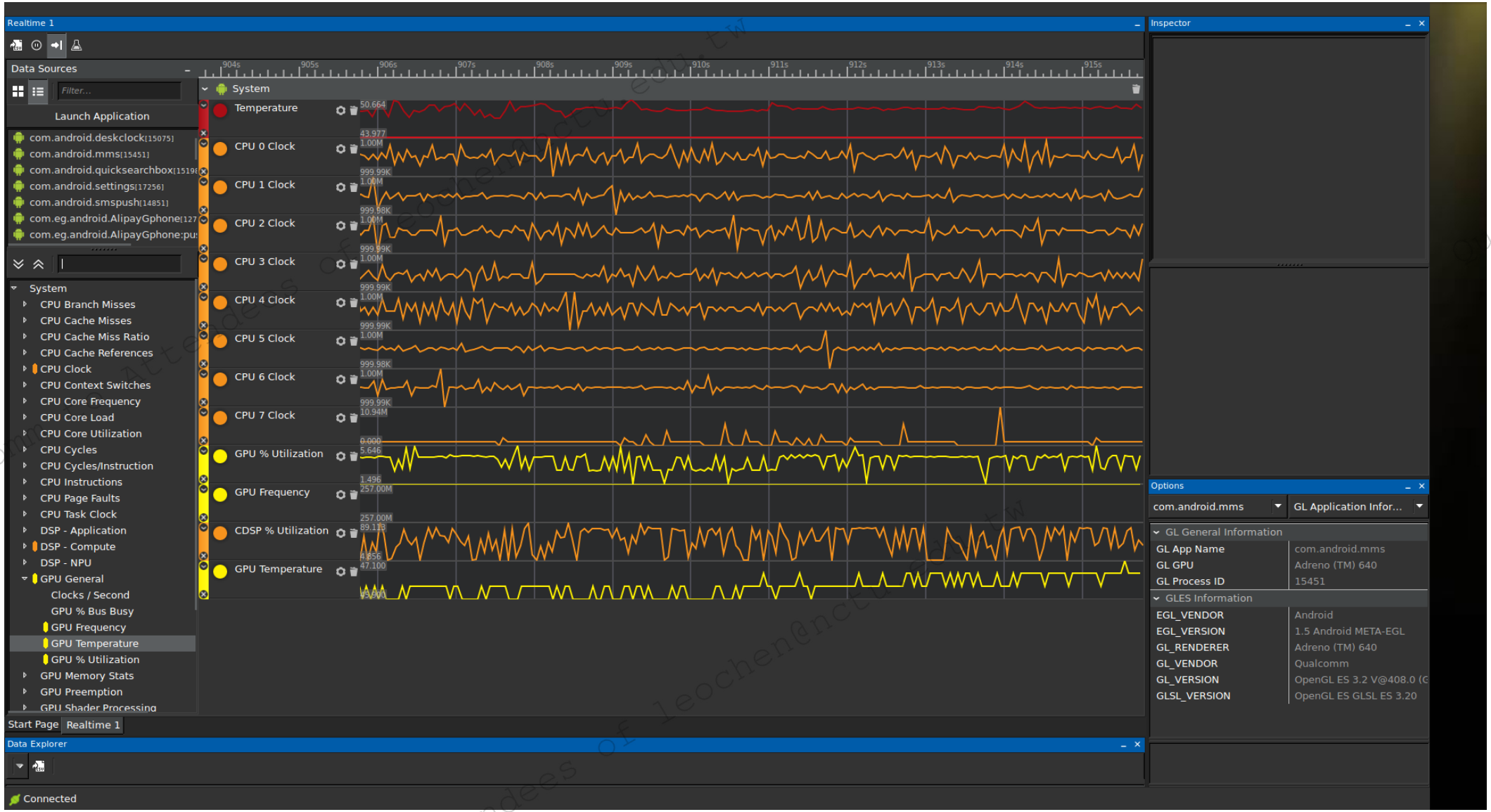
Snapdragon Profiler Features and Benefits

- Real-time view makes it easy to correlate system resource usage on a timeline
  - **Analyze CPU, GPU, DSP*, memory, power, thermal, and network data metrics**
  - Select from over 150 different hardware performance counters in 22 categories
- Trace Capture mode allows you to visualize kernel and system events on a timeline to analyze low-level system events across the CPU, GPU, and DSP.
  - View CPU scheduling and GPU stage data to see where your application is spending its time
- Snapshot Capture*** mode allows you to capture and debug a rendered frame from any OpenGL ES app
  - Step through and replay a rendered frame draw call-by-draw call
  - View and edit shaders and preview the results on your device
  - View and debug pixel history
  - Capture and view GPU metrics per draw call
- GPU APIs: OpenGL ES 3.1, OpenCL 2.1, and Vulkan 1.0**

* Requires a Snapdragon 820 (or later) processor
** Requires Android N (or an Android 6.0 device with a graphics driver that supports Vulkan)
*** Requires a Snapdragon 805 (or later) processor and Android 6.0 (or later)

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

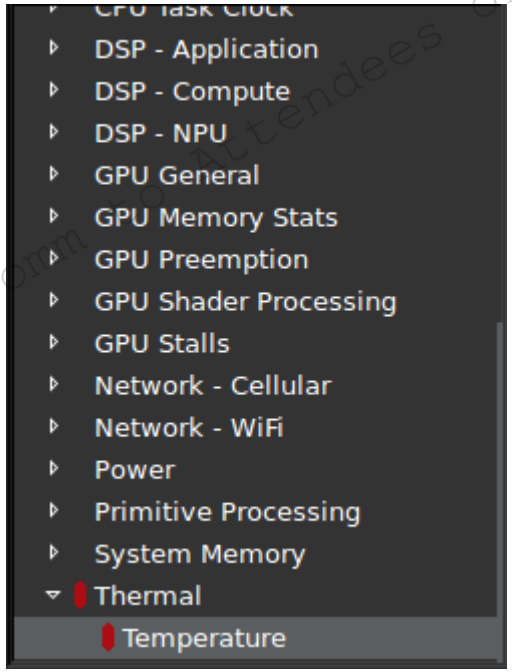
Snapdragon Profiler Appearence



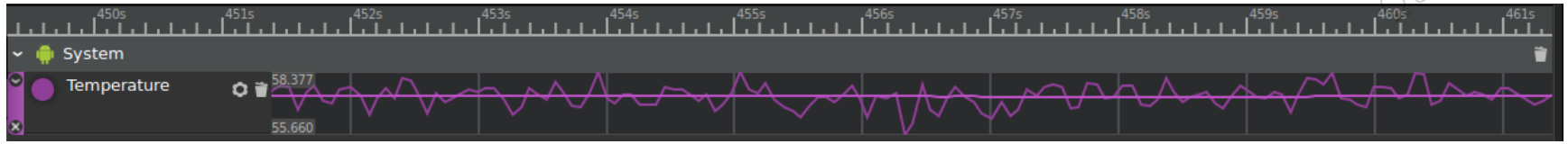
Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Example to Measure Temperature

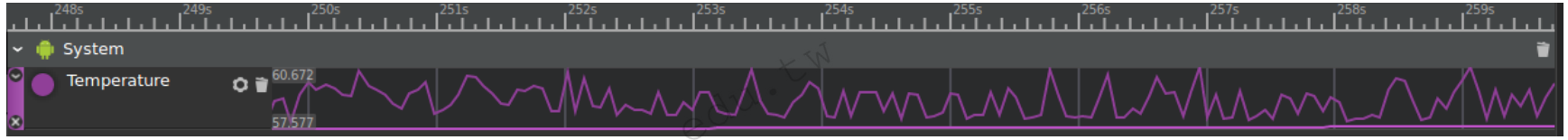
Test Device: Xiao Mi 9 (Snapdragon 855 Processor)
Test Method:
Step 1 : Reboot device and cool to room temperature.
Step 2: Run Snapdragon Profiler and Face Recognition Demo to test temperature for 5 minutes.
More details, please refer to https://developer.qualcomm.com/software/snapdragon-profiler



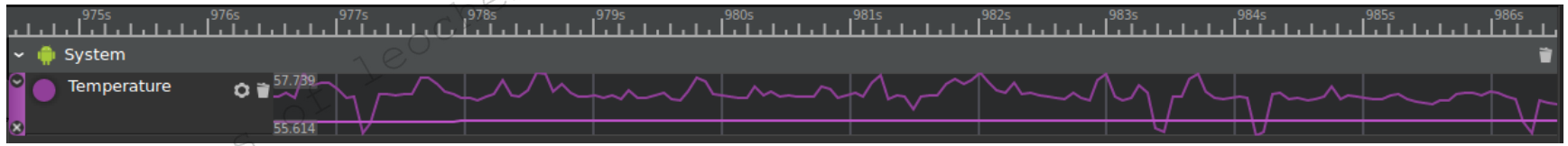
CPU 57.3 ℃



GPU 57.8 ℃



DSP 56.1 ℃



Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Solve Problem with SNPE

Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Color Enhancement Scheme

Input image

SNPE inference result image

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

Q & A

Qualcomm to Attendees of leochen@nctu.edu.tw
Qualcomm to Attendees of leochen@nctu.edu.tw

Please keep this document for personal use only
DO NOT DISTRIBUTE IT

ThunderSoft®

Edge Computing & On-Device AI Enabler

Qualcomm

Thank You

Qualcomm to Attendees of leochen@nctu.edu.tw