Snapdragon NPE Overview

Mark Charlebois
Director, Engineering
Qualcomm Technologies, Inc.
Snapdragon Neural Processing Engine

Software accelerated runtime for the execution of deep neural networks on device

Available at: developer.qualcomm.com

Efficient execution on Snapdragon
Model framework/network support
Developer Tools

What’s new?

Fixed and floating point optimizations
Supports Caffe2, CNTK, MxNet
New optimizations for networks

TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.
Elements of Snapdragon NPE SDK

API
• C++ library in binary form and header files
• Java library for Android integration
• C++ and Python API support for interacting with DLC

DLC
• Snapdragon NPE DNN model format
• Network is a collection of connected layers
• DNN models are stored in DLC files

Tools
• Model converters to create Snapdragon NPE compatible DNN models from popular training framework formats
• Optimization and debugging support tools

Support Assets
• Development host (x86 Ubuntu 14.04)
• User and reference documentation
• Tutorials and examples
• Benchmarking
Snapdragon NPE SW Diagram

SDK Productivity Components
- 3rd Party Apps
- Benchmarking
- Network Debug Tools
- Tutorial Samples
- User & Reference Docs

SDK API

Core Runtime
- Runtime Engine
  - Profiling Logging
  - Model Debug
  - User Defined Layers (UDL) API
- Compute Networks
  - CPU
  - GPU
  - DSP

OS Drivers
- OS: Android & Linux (x86_64, Armv7, Armv7hf, AArch64)
- QuRT

HW
- CPU
- Adreno GPU
- Hexagon DSP

DNN Model Conversion Tools
- Caffe/2 -> DLC fixed
- TensorFlow -> DLC fixed
- Caffe/2 -> DLC Float
- TensorFlow -> DLC Float
- UDL Plugin

Libraries
- libOpenCL.so
- libsnpe_[a,c]dsp*.so
- libsnpe_dsp_*skel.so

Model Debug
Profiling Logging
Network Debug Tools
User Defined Layers (UDL) API
Model loader
DL Container
Test Code
SDK API
Snapdragon NPE SDK

• SDK can be downloaded from Qualcomm Developer Network

The NPE SDK supports Qualcomm® Snapdragon™ 845, 820, 835, 625, 626, 650, 652, 653, 660, 630, 636, and 450 as well as the Qualcomm® Snapdragon™ 820Am automotive platform and Qualcomm Snapdragon™ Flight. For Qualcomm® Adreno™ GPU support, libOpenCL.so must be present on device.

Toolchains:
• Android (armv7, aarch64) - GCC and Clang toolchains
• Linux (armv7, armv7hf, aarch64, x86_64*) - GCC on ARM, Clang on x86_64

* CPU only
NPE SDK Developer Tools

- snpe-net-run
- snpe-caffe-to-dlc
- snpe-caffe2-to-dlc
- snpe-tensorflow-to-dlc
- snpe-onnx-to-dlc*
- snpe-diagview
- snpe-dlc-info
- snpe-dlc-quantize
- snpe_bench.py

*Coming soon
Using the Snapdragon NPE
Snapdragon NPE Workflow

GoogleNet
Inception
SSD
Alexnet
ResNet
MobileNet
SqueezeNet
Faster - RCNN

**SNPE Workflow**

User Defined Layer (UDL) - enables prototyping of layers not yet supported

**SNPE Workflow with UDL**

Caffe

Caffe2
In the Snapdragon NPE, images must be presented as a tensor of shape (height x width x channel), where channel is the fastest-changing dimension.

- See $SNPE_ROOT/models/alexnet/scripts/create_alexnet_raws.py in the SDK

For current Snapdragon NPE SDK release, N=1. Batch support coming in future release.
Quantized vs Non-Quantized Models

- Non-quantized DLC files use 32 bit floating point representations of network parameters.
- Quantized DLC files use 8 bit fixed point representations of network parameters and are smaller.
Making a Snapdragon NPE Enabled Application

App setup

```cpp
bool useUserSuppliedBuffers = false;

// Set the Runtime
static zdl::DlSystem::Runtime_t runtime =
  zdl::SNPE::SNPEFactory::isRuntimeAvailable(zdl::DlSystem::Runtime_t::GPU) ?
  zdl::DlSystem::Runtime_t::GPU : zdl::DlSystem::Runtime_t::CPU;

// Load DLC Container
std::unique_ptr<zdl::DlContainer::IDlContainer> container =
  zdl::DlContainer::IDlContainer::open(dlcPath);

// Build SNPE instance
zdl::SNPE::SNPEBuilder snpeBuilder(container);
std::unique_ptr<zdl::SNPE::SNPE> snpe = snpeBuilder.setOutputLayers({})
  .setRuntimeProcessor(runtime)
  .setUdlBundle(udlBundle)
  .setUseUserSuppliedBuffers(useUserSuppliedBuffers)
  .build();
```
Making a Snapdragon NPE Enabled Application
Running the network (ITensor)

```cpp
// Load the inputs
std::unique_ptr<zdl::DlSystem::ITensor> inputTensor = loadInputTensor(snpe, fileLine); // See SDK docs
static zdl::DlSystem::TensorMap outputTensorMap;

// Run the network
snpe.execute(inputTensor, outputTensorMap);

zdl::DlSystem::StringList tensorNames = outputTensorMap.getTensorNames();

// Access the results
std::for_each( tensorNames.begin(), tensorNames.end(), [&](const char* name){
    auto tensorPtr = outputTensorMap.getTensor(name);
    for ( auto it = tensorPtr->cbegin(); it != tensorPtr->cend(); ++it ){
        float f = *it;
        ...
    }
});
```
Making a SNPE Enabled Application

Running the network (UserBuffer)

```cpp
// Load the Inputs
loadInputUserBuffer(applicationInputBuffers, snpe, fileLine); // See SDK Docs

// Run the Network
snpe.execute(inputMap, outputMap);

const zdl::DlSystem::StringList& outputBufferNames = outputMap.getUserBufferNames();

// Access the results
std::for_each(outputBufferNames.begin(), outputBufferNames.end(), [&] (const char* name) { 
    auto buffer = applicationOutputBuffers.at(name).data();
    float *f;
    for (auto i=0; i< buffer.size(); i+=sizeof(float)) { 
        f = reinterpret_cast<float*>(&buffer[i]);
        ...
    }
});
```

...