RUNNING ACCELERATED NEURAL NETWORKS USING PYTHON AND ARM NN

Pavel Macenauer
Sr. Software Engineer, NXP Semiconductors
SEPTEMBER 2020
OVERVIEW

• Introduction to Arm NN
  • About Arm NN
  • Linaro & project infrastructure
• New Python Interface in Arm NN 20.05 (PyArmNN)
  • About PyArmNN
  • SWIG
• NN Acceleration & Arm NN Backends
  • What is it?
  • Backends available
  • 3rd party backends
Introduction to Arm NN
WHAT IS ARM NN?

- A **middleware** inference engine for machine learning on the edge
  - **Single API** integrating popular high-level ML frameworks (TensorFlow, TF Lite, Caffe, ONNX – MXNet, PyTorch)
  - Connects high-level ML frameworks to compute engines, drivers, HW through **Arm NN backends**
  - Optimized for **ARM and NXP** hardware
    - Cortex-A CPUs, Mali GPUs, Ethos-N NPUs
    - i.MX8 microprocessors (Cortex-A CPUs + GPU/NPU for acceleration)

https://github.com/ARM-software/android-nn-driver
ARM NN, LINARO, RELEASES AND DEVELOPMENT

• Open source project donated mid-2018 to Linaro AI initiative (https://www.mlplatform.org/) by ARM
  - Development lead by ARM (pull requests are merged by ARM core developers)
  - Community and NXP is participating as well

• Releases on GitHub (https://github.com/ARM-software/armnn)
  - Quarterly releases (latest 20.08)
  - See also for release notes, to report bugs or start discussion

• Development and pull requests on mlplatform.org (https://review.mlplatform.org/admin/repos/ml/armnn)

• Synchronized releases with Arm Compute Library (https://github.com/ARM-software/ComputeLibrary)
PyArmNN

(Python interface to Arm NN since 20.05)
 PYARMNN

• Wrapper around the C++ interface
  - Does not implement any computational kernels
    ▪ A few convenient functions
  - Built from the Arm NN repository
    ▪ CMake or standalone scripts can be used
  - Not available on PyPi
  - Python API is very similar to C++
  - Generated using SWIG

• Contributed by the joint forces of ARM and NXP to Arm NN 20.05
  - Available on Arm NN GitHub since 20.05
  - Standalone project for older versions (19.08, 20.02) on NXP GitHub
    ▪ User must provide prebuilt Arm NN libraries and headers
    ▪ i.MX8 Yocto Linux images support 19.08 and 20.02

CMake

$ cmake .. -DBUILD_PYTHON=1

Scripts and commands

$ export SWIG_EXECUTABLE=<path_to_swig>
$ export ARMNN_INCLUDE=<path_to_armnn_include>
$ export ARMNN_LIB=<path_to_armnn_libraries>

$ python setup.py clean --all
$ python swig_generate.py -v
$ python setup.py build_ext --inplace

$ python setup.py sdist
$ python setup.py bdist_wheel

https://github.com/ARM-software/armnn/blob/branches/armnn_20_08/python/pyarmnn/README.md
https://github.com/NXPmicro/pyarmnn-release
HOW TO GENERATE A PYTHON WRAPPER USING SWIG

• What is SWIG? ([http://www.swig.org/](http://www.swig.org/))
  - A tool connecting C/C++ with a wide range of other programming languages (Python, JavaScript, Perl, etc.)
  - It’s here for 20+ years
  - Variety of other options Boost.Python, pyrex, ctypes, pybind, etc.
  - Need to write 2 files - SWIG template and setup.py (setuptools, distutils)
    - setuptools.Extension in setup.py
    - Expose the interface in SWIG templates + memory management
  - PyArmNN requires SWIG 4 due to C++ STL support
    - May be required to install from sources (typically SWIG 3 is available in package managers)
    - SWIG_EXECUTABLE variable in the build system
# Whatever Python imports are necessary
import numpy as np
import cv2
import pyarmnn as ann

# Create a parser and load the model (ONNX, TF or Caffe can be used as well)
parser = ann.ITfLiteParser()  # Other parsers may required additional inputs
network = parser.CreateNetworkFromBinaryFile('my_model.tflite')

# Initialize options, runtime and set backends to run the model on
options = ann.CreationOptions()
rt = ann.IRuntime(options)
preferredBackends = [ann.BackendId('VsiNpu'), ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]

# Load into Arm NN internal format/engine
opt_network, _ = ann.Optimize(network, preferredBackends, rt.GetDeviceSpec(), ann.OptimizerOptions())
net_id, _ = rt.LoadNetwork(opt_network)
image = ...  # Load a frame, image, process using numpy, cv2, etc.

# Get input tensor from the model and load it with the image
input_names = parser.GetSubgraphInputTensorNames(0)
input_binding_info = parser.GetNetworkInputBindingInfo(0, input_names[0])
input_tensors = ann.make_input_tensors([input_binding_info], [image])

# Get output tensor from the model
output_names = parser.GetSubgraphOutputTensorNames(0)
output_binding_info = parser.GetNetworkOutputBindingInfo(0, output_names[0])
output_tensors = ann.make_output_tensors([output_binding_info])

rt.EnqueueWorkload(0, input_tensors, output_tensors)  # Run inference
out_tensor = ann.workload_tensors_to_ndarray(output_tensors)[0][0]

“A tabby is any domestic cat with a distinctive ‘M’ shaped marking on their forehead, stripes by their eyes and across their cheeks ...”

https://en.wikipedia.org/wiki/Tabby_cat
NN Acceleration & Arm NN Backends
**ARM NN BACKENDS**

- An abstraction connecting layers of a neural network graph to the underlying hardware through a driver or a compute engine
  - Enables hardware acceleration on a CPU, a GPU or an NPU
  - Existing backends –
    - **ARM Compute Library - OpenCL** (Mali GPU), **NEON** (Cortex-A CPU)
      - Can support any GPU with at least OpenCL 1.1, non-uniform workgroup size ext, fp16 ext, int64 base atomics ext
    - **ARM Ethos-N backend** using Ethos-N driver
    - **Reference backend** for testing, validation and as a default fallback
  - Easy to implement a 3rd party backend
    - E.g. VsiNpu backend for NXP i.MX8 microprocessors

[Links]
https://github.com/ARM-software/ethos-n-driver-stack
https://arm-software.github.io/ComputeLibrary/v20.08/
CPU ACCELERATION USING ARM NEON

• Accelerated using Arm NEON™ backend in ARM Compute Library

• **Mobilenet v1** quantized inference (Mobilenet_V1_1.0_224)
  - Popular image classification model for mobile platforms

  - Unoptimized (reference): **77441.685 ms**
    (i.MX8MP single A53 CPU core)

  - **NEON: 93.769 ms**
    (i.MX8MP 4x A53 CPU) = 825x faster than unoptimized implementation

https://github.com/ARM-software/ComputeLibrary
https://www.tensorflow.org/lite/guide/hosted_models

• Advanced Single Instruction Multiple Data (SIMD) architecture extension for the Arm Cortex-A and Cortex-R series processors

• Targeted for audio, video processing, computer vision, deep learning, etc.
HYBRID EXECUTION

- Layers are executed using **workloads** created by individual backends
  - Depends on layer support and which layers were specified by the user
- **Multiple backends** can be used
- Done internally in **Optimize** function

# VsiNpu is a custom backend for NXP i.MX8 devices
# CpuAcc is the ARM Neon accelerated backend
# CpuRef is the unoptimized CPU backend

```
preferredBackends = [ ann.BackendId('VsiNpu'),
                    ann.BackendId('CpuAcc'),
                    ann.BackendId('CpuRef') ]
```
3rd Party Backend

What is required to implement?

1. **Backend interface** - IWorkloadFactory, LayerSupportBase, IMemoryManager, etc.
2. **Workloads** – execute layers of a graph
3. **Unit Tests** – integrated into UnitTests binary
4. **Makefiles** (CMake)

- **Dynamic** (loaded during runtime) and **static** (compiled into Arm NN library) backends

Example directory structure

```
test
  - CMakeLists.txt
  - NpuLayerTests.cpp
  ...
workloads
  - CMakeLists.txt
  - NpuConvolution2dWorkload.hpp
  - NpuFullyConnectedWorkload.hpp
  ...
CMakeLists.txt
backend.cmake
NpuTensorHandler.cpp
NpuTensorHandler.hpp
NpuWorkloadFactory.cpp
NpuWorkloadFactory.hpp
...```
I.MX8 NPU Backend

- Delegates compute to the OpenVX™ driver (chooses GPU/NPU based on HW available)

- OpenVX™ is an open, royalty-free standard for cross-platform acceleration of computer vision applications by Khronos

- Mobilenet v1 quantized inference: **3.024 ms**
  - compared to **93.769 ms** using CpuAcc (4x A53 + NEON)

https://source.codeaurora.org/external/imx/armnn-imx/
SECURE CONNECTIONS
FOR A SMARTER WORLD