Supercharge your development time with the Arm NN and Arm Compute Library

Ronan Naughton, Jim Flynn, Gian Marco Iodice
9th February 2021
## AI Virtual Tech Talks Series

<table>
<thead>
<tr>
<th>Date</th>
<th>Title</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 9(^{th})</td>
<td>Supercharge your development time with the new Arm NN 20.11 release</td>
<td>Arm</td>
</tr>
<tr>
<td>February 23(^{rd})</td>
<td>Hands-on with PyArmNN for object detection</td>
<td>Arm Workshop</td>
</tr>
<tr>
<td>March 9(^{th})</td>
<td>Automate tinyML Development &amp; Deployment with Qeexo AutoML</td>
<td>Qeexo</td>
</tr>
</tbody>
</table>

Visit: developer.arm.com/solutions/machine-learning-on-arm/ai-virtual-tech-talks
Presenters

Ronan Naughton
Senior Product Manager,
MLG, Arm

Jim Flynn
Arm NN Engineering Manager,
MLG, Arm

Gian Marco Iodice
ACL Staff Software Engineer,
MLG, Arm
Arm Software Stack

Applications

High Level Frameworks
(e.g. TensorFlow Lite, PyTorch, Android NNAPI)

Tools

Dynamic Workload Compilers & Drivers

CPU
Cortex-A Neoverse

GPU
Mali

NPU
Ethos-N
Why Use the Arm NN and Arm Compute Library?

Versatile and Portable:
- Easily target multiple platforms (CPU, GPU and NPU) from a single code base
- Reduce overall development time, keep using existing framework and tools
- Deployable for Android, Linux and ‘bare metal’ applications

Superior Performance:
- Best in class across a wide range of popular networks
- Uses advanced network optimization techniques, workload tuning and GEMM heuristics

Arm Specific Optimizations:
- Outperforms generic math and ML libraries due to Arm specific optimization
- Specific architectures (e.g. dot product for Armv8.2A) and micro architecture optimizations (e.g. Cortex-A53)
- Quick adoption of new Arm technologies e.g. SVE, SVE2
Relentless Performance Optimization

Arm NN performance improvements*

- Continuous optimization of the key operators in the most popular networks
- New operators added to support each release of Android
- Advanced network manipulation techniques used to improve performance

*Mean performance improvements of Arm NN relative to up to six different industry software libraries

Source: Arm benchmarking 20.11
Arm NN Integration Options with Neural Network Frameworks

- Multiple framework integration options for Linux and Android

- Model file support via parser
- Runtime support
Arm NN TFLite Delegate

Accelerating inference on Android and Linux

- Unlocks Arm specific CPU and GPU optimizations for TFLite users
- All TFLite models can be accelerated through Arm NN with unsupported ops handled by the TFLite runtime
- Enables performance and great operator coverage
Debian Packages for Arm NN and ACL

• 20.08 Release of Arm SW stack available on Ubuntu Launchpad PPA. Formal release in Bullseye (Debian 11)

• Benefits:
  • Reliable: All build dependencies are taken care of by the robust Debian packaging infrastructure
  • Quick setup: Ready for prototyping with just a few ‘apt-get’ commands
  • Accessible: Ubuntu ‘Groovy’ now available for Raspberry Pi with full Aarch64 desktop experience
Create a parser object and load your model file.

```python
import pyarmnn as ann
import imageio
parser = ann.ITfLiteParser()
network = parser.CreateNetworkFromBinaryFile('./model.tflite')
```

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

```python
input_binding_info = parser.GetNetworkInputBindingInfo(0, 'model/input')
options = ann.CreationOptions()
runtime = ann.IRuntime(options)
```

Choose preferred backends for execution and optimize the network.

```python
preferredBackends = [ann.BackendId('CpuAcc'), ann.BackendId('CpuRef')]
opt_network, messages = ann.Optimize(network, preferredBackends,
runtime.GetDeviceSpec(),
ann.OptimizerOptions())
```

Make workload tensors using input and output binding information.

```python
img = imageio.imread('./image.png')
input_tensors = ann.make_input_tensors([input_binding_info], [img])
output_binding_info = parser.GetNetworkOutputBindingInfo(0, 'model/output')
output_tensors = ann.make_output_tensors([output_binding_info])
```

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

```python
runtime.EnqueueWorkload(net_id, input_tensors, output_tensors)
results = ann.workload_tensors_to_ndarray(output_tensors)
print(results)
```
Arm Public Model Zoo

- Highly optimized networks specific to Arm architectures
- Clustering, pruning and quantization aware training used to produce the most efficient models
- Model, meta data and test data available
- Supported by code samples and How-To guides

<table>
<thead>
<tr>
<th>Model</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD MobileNet v1</td>
<td>Revised for 30% Arm NN performance uplift</td>
</tr>
<tr>
<td>MobileNet v2</td>
<td>Clustering and pruning for better accuracy at the same sized</td>
</tr>
<tr>
<td>Wav2Letter</td>
<td>10x faster, tuned to mobile / embedded voice UI</td>
</tr>
<tr>
<td>DS-CNN</td>
<td>Revised for updated tooling and clustering</td>
</tr>
<tr>
<td>Yolo V3</td>
<td>Simplified classes and BB, detector quantization compatible</td>
</tr>
</tbody>
</table>
Demo 1: Inference on Linux using Raspberry Pi4
Examining the performance on Cortex-A72 CPU using TFLite Delegates

### Test 1

<table>
<thead>
<tr>
<th>Accelerator</th>
<th>Min Exec. time - ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFLite CPU Ref (Test 1a)</td>
<td>505</td>
</tr>
<tr>
<td>TFLite CPU XNNPACK (Test 1b)</td>
<td>372.2</td>
</tr>
<tr>
<td>TFLite CPU ArmNN (Test 2)</td>
<td>246</td>
</tr>
</tbody>
</table>

### Test 2

<table>
<thead>
<tr>
<th>Accelerator</th>
<th>Min Exec. time - ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFLite Model Benchmark Tool</td>
<td></td>
</tr>
<tr>
<td>Arm NN TFLite Delegate</td>
<td></td>
</tr>
<tr>
<td>Compute Library</td>
<td></td>
</tr>
<tr>
<td>Cortex-A72 CPU</td>
<td></td>
</tr>
</tbody>
</table>

Yolo V3 (Arm - fp32) | Inception V3 (TFLite - fp32) |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>505</td>
<td>793.5</td>
</tr>
<tr>
<td>372.2</td>
<td>600.9</td>
</tr>
<tr>
<td>246</td>
<td>385.3</td>
</tr>
</tbody>
</table>

~ 1.5x - 1.6x Perf Speed Up over XNNPACK
Demo 2: Inference on Android
Examining the performance on Mali G77 GPU using TFLite Delegates

Test 1
- TFLite Model Benchmark Tool
  - TFLite CPU Ref
  - TFLite GPU Delegate
- Mali G77 GPU

Test 2
- TFLite Model Benchmark Tool
  - Arm NN, TFLite Delegate
  - Arm NN
  - Compute Library
- Mali G77 GPU

<table>
<thead>
<tr>
<th>Accelerator</th>
<th>Min. Exec. time - ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFLite GPU delegate (Test 1)</td>
<td>44.8</td>
</tr>
<tr>
<td>TFLite GPU ArmNN w. tuner (Test 2)</td>
<td>26.5</td>
</tr>
<tr>
<td>Yolo V3 (Arm - fp32)</td>
<td></td>
</tr>
<tr>
<td>Inception V3 (TFLite - fp32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>~1.7x - 2.2x Perf Speed Up over TFLite GPU Delegate</td>
</tr>
</tbody>
</table>
Useful Links

- Arm NN HAL for Android: https://github.com/ARM-software/android-nn-driver
- Arm NN TFLite Delegate: https://github.com/ARM-software/armnn/tree/branches/armnn_20_11/delegate
- Arm Model Zoo: https://github.com/ARM-software/ML-zoo/tree/master/models
- Arm ML Examples: https://github.com/ARM-software/ML-examples/
- How to Guides: https://developer.arm.com/solutions/machine-learning-on-arm/
- Ubuntu for Raspberry Pi: https://ubuntu.com/raspberry-pi
- Contributions: https://www.mlplatform.org/contributing/

SHA versions

- TensorFlow: 1b215470642efd86e927d1c15ba026b4ff45dfa7
- ArmNN: 97bf84f6e162307fc3e8c53045ef0bc60a3e3289
- ACL: b309fc249e4383b4d40ae03e377c3cbad3f9f5f7
Thank you!

Tweet us: @ArmSoftwareDev

Check out our Arm YouTube channel and our Arm Software Developers YouTube channel

Signup now for our next AI Virtual Tech Talk here

Attendees: don’t forget to fill out the survey to be in with a chance of winning an Arduino Nano 33 BLE board
Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה