Arm NN plug-in framework

Les Bell, Matteo Martincigh, Narumol Prangnawarat
Agenda

• Arm NN Overview
• Arm NN Graph Builder
• Optimizing and running the network
• Backends and workloads
• Custom backends
Arm NN Overview
Arm NN Overview

• C++ 14 ML inference API for Linux

• Developed as Open Source software
  • Contributions are welcome from anyone and are reviewed before acceptance

• Synchronised release with Compute Library and Android NNAPI driver libraries
  • Compute Library
    – Arm CPU with NEON acceleration (ARMv7 and v8x)
    – Arm Mali GPU with OpenCL acceleration (Midgard and Bifrost architectures)
  • Android NNAPI driver
    – Forwards Android NNAPI HAL calls to the Arm NN API

Arm NN  https://review.mlplatform.org/#/admin/projects/ml/armnn
Arm Compute Library  https://review.mlplatform.org/#/admin/projects/ml/ComputeLibrary
Arm Android NN Driver  https://review.mlplatform.org/#/admin/projects/ml/android-nn-driver
Arm NN Overview (continued)

- NN applications and high level libraries can use Arm NN as a single API to access many NN accelerated devices
- Arm Android NN HAL driver provides access to Arm NN for Android applications
- Arm NN provides the backends for the lower level libraries
  - Third-party partners can add their own backends for Arm NN
- Some applications use the Compute Library directly, they can still do so
- NN Inference only (training is not supported)
Arm NN Components

- **Arm NN Core**
  - Graph builder API
  - Optimizer
  - Runtime
    - Reference and Neon/Cl via Compute Library
    - New backend planned

- **Parsers**
  - Tensorflow
  - Tensorflow Lite
  - Caffe
  - ONNX

- **Android NNAPI Driver**
Arm NN Supported Layers

... more to come

- Activation
- Addition
- BatchNormalization
- BatchToSpaceND
- Constant
- Convolution2d
- DepthwiseConvolution2d
- DetectionPostProcess
- Division
- Equal
- Floor
- FullyConnected

- Gather
- Greater
- Input
- L2Normalization
- LSTM
- Maximum
- Mean
- MemCopy
- Merger
- Minimum
- Multiplication
- Normalization

- Output
- Pad
- Permute
- Pooling2d
- Reshape
- ResizeBilinear
- Rsqrt
- Softmax
- SpaceToBatchND
- Splitter
- StridedSlice
- Subtraction
Arm NN Graph Builder
Graph Builder

- Provides step-by-step API for building a complete model
  - Sometimes also called the model builder

- Interface header:
  - “armnn/include/armnn/INetwork.hpp”

- Main classes
  - INetwork
  - IConnectableLayer
    - IInputSlot
    - IOutputSlot

```c++
/// Add a pooling layer to the network.
/// @param pooling2DDescriptor Pooling2DDescriptor to configure the pooling
/// @param name Optional name for the layer
/// @return Interface for configuring the layer.
virtual IConnectableLayer* AddPooling2DLayer(const Pooling2DDescriptor& pooling2DDescriptor, const char* name = nullptr) = 0;

/// Add an activation layer to the network.
/// @param activationDescriptor ActivationDescriptor to configure the activation
/// @param name Optional name for the layer
/// @return Interface for configuring the layer.
virtual IConnectableLayer* AddActivationLayer(const ActivationDescriptor& activationDescriptor, const char* name = nullptr) = 0;

/// Add a normalization layer to the network.
/// @param normalizationDescriptor NormalizationDescriptor to configure the normalization
/// @param name Optional name for the layer
/// @return Interface for configuring the layer.
virtual IConnectableLayer* AddNormalizationLayer(const NormalizationDescriptor& normalizationDescriptor, const char* name = nullptr) = 0;

/// Add a softmax layer to the network.
/// @param softmaxDescriptor SoftmaxDescriptor to configure the softmax
/// @param name Optional name for the layer
/// @return Interface for configuring the layer.
virtual IConnectableLayer* AddSoftmaxLayer(const SoftmaxDescriptor& softmaxDescriptor, const char* name = nullptr) = 0;
```
Connecting layers

- Layers are connected via the IConnectableLayer interface using a slot mechanism.
- Each layer has \([0-n]\) input slots and \([0-n]\) output slots.
  - Number of input/output slots depends on the layer type.
  - Most are fixed but some can be variable (e.g., Merger/Splitter layer).
- Output slots are connected to 1 or more input slots.
- Input slots are connected to only 1 output slot.
A simple example
A simple example

- Create an empty Model object

```cpp
INetworkPtr model = INetwork::Create();
```

- Add layers to the model

```cpp
IConnectableLayer* input0 = model->AddInputLayer(bindingIn0, "Input0");
IConnectableLayer* input1 = model->AddInputLayer(bindingIn1, "Input1");
IConnectableLayer* addition = model->AddAdditionLayer("Addition");
IConnectableLayer* output = model->AddOutputLayer(bindingOut, "Output");
```
A simple example

Input0

Addition

Input1

Output
Establishing connections

```cpp
IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
```
Establishing connections

```cpp
IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);```

Diagram:
- Input0
- Input1
- Addition
- Output
Establishing connections

```
IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
```
Establishing connections

```cpp
IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
```
Establishing connections

IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
Establishing connections

IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
Establishing connections

IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
Establishing connections

IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
Establishing connections

IOutputSlot* input0Out = input0->GetOutputSlot(0);
IInputSlot* addIn0 = addition->GetInputSlot(0);
input0Out->Connect(addIn0);

IOutputSlot* input1Out = input1->GetOutputSlot(0);
IInputSlot* addIn1 = addition->GetInputSlot(1);
input1Out->Connect(addIn1);

IOutputSlot* addOut = addition->GetOutputSlot(0);
IInputSlot* outputIn = output->GetInputSlot(0);
addOut->Connect(outputIn);
TensorInfo

• Set the tensor info for each output slot
  • Shape
  • Data format
  • Quantization parameters

```cpp
TensorShape shape({1, 1, 3, 4});
TensorInfo info(shape, DataType::Float32);

input0Out->SetTensorInfo(info);
input1Out->SetTensorInfo(info);
addOut->SetTensorInfo(info);
```
Optimizing and running the network
Optimizer

- Takes the completed `INetwork` object (contains a `Graph`) as input
  - `IOptimizedNetworkPtr Optimize(INetwork, { BackendId, ... }, IDeviceSpec, OptimizerOptions, errMessages)`

- Performs basic validation of the input network

- Modifies the graph for correctness
  - Inserts Copy layers between backends
  - Inserts FP32/FP16 conversion layers if necessary (specified in `OptimizerOptions`)
  - Adds debug layers, if required (specified in `OptimizerOptions`)

- Performs backend-independent optimizations
  - Removes redundant operations
  - Optimizes permutes/reshapes where possible (inverse permutes, permutes as reshapes, consecutive reshapes, ...)

© 2019 Arm Limited
**Optimizer (continued)**

- Decides which backend to assign to each layer
  - Uses the `ILayerSupport` interface, `IsXXXLayerSupported(…)` call to the preferred backend
    - e.g. `IsActivationSupported(...), IsConvolution2dSupported(...), IsLstmSupported(...)`
  - If the layer is not supported, it asks the next preferred backend, and so on...

- Runs backend-specific optimizations
  - For each selected backend, extracts the subgraphs that can be executed on that backend
  - For each subgraph, call `OptimizeSubGraph` on the selected backend
    - More details are included later

- Produces an `IOptimizedNetwork` object as output
Optimizer (continued)

• To ensure your backend is assigned to a layer
  • It must be the first BackendId in the list of preferred backends passed to the Optimize() function
  • Your backend must return true from the IsXXXLayerSupported(...) call
  • The OptimizeSubGraph(...) call must not fail
Runtime

- Loads an `IOptimizedNetwork` object
- Manages runtime memory
- Executes inference/predictions using `Runtime::EnqueueWorkload()` method
  - Low-overhead dispatcher
  - Dispatches workloads to desired backend
  - Manages synchronisation
Run inference

• Create the Runtime

```cpp
CreationOptions options; // GPU tuning and profiling options
IRuntimePtr runtime = IRuntime::Create(options);
```

• Optimize the completed model

```cpp
DeviceSpec spec = runtime->GetDeviceSpec(); // A set of registered BackendId's
std::vector<BackendId> backendPreferences = {
    armnn::BackendId("GpuAcc"),
    armnn::BackendId("CpuAcc"),
    armnn::BackendId("CpuRef")};

IOptimizedNetworkPtr optimizedModel = Optimize(model, backendPreferences, spec);
```
Run inference (continued)

• Load the optimized network in the runtime

```c
NetworkId networkId;
runtime->LoadNetwork(networkId, optimizedModel);
```

• A network identifier (`networkId`) is returned to use later for running the optimized network
• Creates a `LoadedNetwork` object and adds it to the runtime
• `LoadNetwork` creates a list of workloads (one per layer) using the backend’s `IWorkloadFactory` object
  - `IWorkload CreatePooling2d(...)`
  - `IWorkload CreateConvolution2d(...)`
  - ...

Run inference (continued)

• Create the input and output tensors

  InputTensors inputTensors{{LayerBindingId, 
    ConstTensor(TensorInfo, inputData)}, …}

  OutputTensors outputTensors{{LayerBindingId, 
    ConstTensor(TensorInfo, outputBuffer)}, …}

  • LayerBindingId (an int) is a unique id among inputs and outputs

• Execute the network

  • runtime->EnqueueWorkload(NetworkId, InputTensors, OutputTensors)

  • The NetworkId to use is the one you got from the call to LoadNetwork

  • Executes all workloads sequentially on the assigned backends

  • EnqueueWorkload will block until execution is finished

  • The result will be in the output tensor buffers

• Sample app: armnn/samples/SimpleSample.cpp
Backends and workloads
Backends

• A Backend is an abstraction that maps the layers of a network graph to the hardware that is responsible for executing those layers
• Backends support one, or more, layers from the graph
• They may optimize the subgraphs assigned to them
• They create backend-specific workloads for the layers they support
• They execute the workloads they create
Subgraphs

- A subgraph is a *view* of the original graph
- It holds a list of layers (of the original graph)
- Subgraphs are selected by Arm NN based on the backends assigned to the layers
  - Largest set of contiguous layers that can be executed independently on a single backend
- In this example:
  - Subgraph2 is allocated to a different backend to Subgraph1 and Subgraph3
  - Subgraph1 and Subgraph3 may be allocated to the same, or different, backends
Workloads

- Each layer will be executed using a workload
- A workload is used to enqueue a layer for computation
- Each workload is created by a WorkloadFactory
- Each backend needs its own WorkloadFactory
  - Creates workloads specific to each layer
Workload Factory

• A Workload Factory implements the IWorkloadFactory interface
  • GetBackendId
  • SupportsSubTensors
  • CreateSubTensorHandle
  • CreateTensorHandle
  • CreateInput
  • CreateOutput
  • Create<Layer> Functions to create each workload to compute the specific layer
Workload Implementation

- All workloads need to...
  - Implement the IWorkload interface
  - Implement the Execute method to execute the operator on the backend hardware
    - Read the input tensors
    - Write the result to the output tensors

```cpp
class IWorkload {
public:
  virtual ~IWorkload() {}  
    virtual void Execute() const = 0;
};
```
Custom backends
Custom backends

- Arm NN allows adding new backends through the *Pluggable Backend* mechanism
- Backends reside under armnn/src/backends in separate subdirectories
- CMake build system
- How to add your backend (*my_backend*):
  - Prepare a *backend.cmake* file (armnn/src/backends/my_backend/backend.cmake)
    - *add_subdirectory*(armnn/src/backends/my_backend)
    - *list*(APPEND armnnLibraries armnnMyBackend armnnMyBackendWorkloads)
    - *list*(APPEND armnnUnitTestLibraries armnnMyBackendUnitTests)
  - Prepare a *CMakeLists.txt* file (armnn/src/backends/my_backend/<tests/workloads>/CMakeLists.txt)
    - Provide rules to build armnnMyBackend, armnnMyBackendWorkloads and armnnMyBackendUnitTests
Custom backends (continued)

• How to add your backend (*my_backend)*:
  
  • Prepare a `backend.mk` file (`armnn/src/backends/my_backend/backend.mk`)
    
    - Arm NN on Android uses the native Android build system
    - BACKEND_SOURCES: list all `cpp` files to be built as part of the Arm NN shared library
      
      ```
      BACKEND_SOURCES := \\
      MyBackend.cpp \\
      workloads/MyBackendWorkload.cpp \\
      ...
      ```
    
    - BACKEND_TEST_SOURCES: list all `cpp` test files to add to Arm NN’s unit tests
      
      ```
      BACKEND_TEST_SOURCES := \\
      tests/MyBackendTests.cpp \\
      ...
      ```
    
    - The files will be automatically picked up by Arm NN’s main `Android.mk` file
Custom backends (continued)

- This shows the Arm NN backends directory structure
- The aclCommon, cl and neon directories contain code used for the Arm OpenCL and NEON backends
- The reference directory provides the reference backend
- backendsCommon contains some header files you need
- README.md explains how to add a backend
- my_backend is the example directory for the custom backend, with the make files described in the previous slides
  - You create this directory, add the make files and your source files
  - Choose your own name for this directory
Custom backends (continued)

- All backends must be uniquely identified by a `BackendId`
  - `armnn::BackendId myBackendId = armnn::BackendId("myBackendName");`
  - "myBackendName" must be unique amongst all backends

- There is a single instance of the `BackendRegistry` class, obtained with the function
  ```cpp
  BackendRegistry& BackendRegistryInstance();
  ```

- All backends must register with the `BackendRegistry`
  ```cpp
  using FactoryFunction = std::function<PointerType()>
  void BackendRegistry::Register(const BackendId& myBackendId, FactoryFunction factory);
  ```
  - Where the factory function returns an object supporting the `IBackendInternal` interface.

- A helper structure is provided to register a backend statically
  ```cpp
  BackendRegistry::StaticRegistryInitializer
  ```
Custom backends (continued)

Example registration for the reference backend:

```cpp
constexpr const char * RefBackendId() { return "CpuRef"; }

const BackendId& RefBackend::GetIdStatic()
{
    static const BackendId s_Id{RefBackendId()};
    return s_Id;
}

static BackendRegistry::StaticRegistryInitializer g_RegisterHelper
{
    BackendRegistryInstance(),
    RefBackend::GetIdStatic(),
    [](){}
    {
        return IBackendInternalUniquePtr(new RefBackend);
    }
};
```
Custom backends (continued)

• All backends need to implement the IBackendInternal interface:
  • IMemoryManagerUniquePtr CreateMemoryManager()
  • IWorkloadFactoryPtr CreateWorkloadFactory(IMemoryManagerSharedPtr)
    – The returned IWorkloadFactory object will be used to create the workloads (layer computation units)
  • IBackendContextPtr CreateBackendContext(IRuntime::CreationOptions)
  • ILayerSupportSharedPtr GetLayerSupport()
    – During optimization Arm NN needs to decide which layers are supported by the backend
    – IsLayerXXXSupported(...) functions return if the backend supports the specified layer
  • SubGraphUniquePtr OptimizeSubGraph(SubGraph, bool& optimizationAttempted)
    – Input: subgraph to optimize
    – Output: a valid, optimized subgraph and a flag to indicate that the optimization has been attempted
    – Replaces the now deprecated: Optimizations GetOptimizations()
**Backend optimization**

- The Optimizer calls `OptimizeSubGraph` on the selected backend, for each `SubGraph`
  - From class `IBackendInternal`
    
    ```
    using SubGraphUniquePtr = std::unique_ptr<SubGraph>;
    SubGraphUniquePtr OptimizeSubGraph(const SubGraph& subGraph,
                                       bool& optimizationAttempted) const = 0;
    ```
  - If no optimization is done, just set `optimizationAttempted` to false and return a null pointer
  - Example optimizations:
    - Merge layers, for more efficient execution
    - Add permute layers to modify the data layout for execution on the backend
  - Returns an optimized subgraph and a bool indicating optimisation was attempted
  - If optimization was not attempted, the subgraph is not changed and Arm NN will move on to the next subgraph
  - If the new optimized subgraph is invalid (e.g. null), Arm NN falls back to the next backend, for this subgraph
  - If the new optimized subgraph is valid, it replaces the original subgraph in the network
Backend optimization PreCompiled layers

- Custom backends may create new layers when optimizing a subgraph
  - For example, to merge two layers (such as a convolution and an activation) for more efficient processing
- The new layer can be supported by Arm NN as a PreCompiledLayer
  - The `PreCompiledLayer` includes a `PreCompiledDescriptor` with a pointer to an opaque layer object
  - The opaque layer object is responsible for storing any information it needs to create the workload
- The backend must support the `IWorkloadFactory::CreatePreCompiled` method
  - The `PreCompiledDescriptor` is passed to this method
  - The backend replaces the merged layers with the `PreCompiledLayer` in the subgraph it returns
Memory Management

• The purpose of the memory management is to minimize the memory usage
  • Allocate memory just before it is needed, release it as soon as it is no longer required

• All backends must support the `IBackendInternal interface CreateMemoryManager` method

• This returns a unique pointer to an `IMemoryManager` object
  
  ```cpp
  IBackendInternal::IMemoryManagerUniquePtr MyBackend::CreateMemoryManager() const
  {
    return std::make_unique<MyMemoryManager>(...);
  }
  ```

• Where `MyMemoryManager` is a class derived from `IBackendInternal::IMemoryManager`

• A backend that does not support a memory manager should return an empty pointer
  
  ```cpp
  IBackendInternal::IMemoryManagerUniquePtr MyBackend::CreateMemoryManager() const
  {
    return IBackendInternal::IMemoryManagerUniquePtr{};
  }
  ```
Memory Management (continued)

- The IMemoryManager interface defines two pure virtual methods to be implemented by the derived class for the backend:
  
  ```
  virtual void Acquire() = 0;
  virtual void Release() = 0;
  ```

- Acquire() is called by the LoadedNetwork before the model is executed
  - The backend memory manager should allocate any memory it needs for running the inference

- Release() is called by the LoadedNetwork, in its destructor, after the model is executed
  - The backend memory manager should free any memory it previously allocated

- The backend memory manager may use internal memory management to further optimize memory usage
  - For example: reuse tensor buffer memory internally when the tensor is no longer used
  - Unfortunately, Arm NN cannot help you with this
Backend Context

- The `IBackendContext` interface defines virtual methods to be implemented by the derived class for the backend:

  - Is this useful?

    ```cpp
    class IBackendContext
    {
      protected:
        IBackendContext(const IRuntime::CreationOptions&) {}

      public:
        // Before and after Load network events
        virtual bool BeforeLoadNetwork(NetworkId networkId) = 0;
        virtual bool AfterLoadNetwork(NetworkId networkId) = 0;

        // Before and after Unload network events
        virtual bool BeforeUnloadNetwork(NetworkId networkId) = 0;
        virtual bool AfterUnloadNetwork(NetworkId networkId) = 0;

        virtual ~IBackendContext() {};
    };
    ```
Questions
Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद
Bonus slides
ArmnnConverter Application

./ArmnnConverter --help
Convert a neural network model from provided file to ArmNN format.

Options:

--help
Display usage information

-f [ --model-format ] arg
Format of the model file, caffe-binary, caffe-text, onnx-binary, onnx-text, tensorflow-binary, tensorflow-text, tflite-binary.

-m [ --model-path ] arg
Path to model file.

-i [ --input-name ] arg
Identifier of the input tensors in the network, separated by whitespace.

-s [ --input-tensor-shape ] arg
The shape of the input tensor in the network as a flat array of integers, separated by comma. Multiple shapes are separated by whitespace. This parameter is optional, depending on the network.

-o [ --output-name ] arg
Identifier of the output tensor in the network.

-p [ --output-path ] arg
Path to serialize the network to.
ArmnnQuantizer Application

- Quantizes a Float 32 model to produce a Quantized Asymm 8 model

  ```
  ./ArmnnQuantizer --help
  Options:
  -h [ --help ]    Display help messages
  -f [ --infile ] arg Input file containing float 32 ArmNN Input Graph
  -d [ --outdir ] arg Directory that output file will be written to
  -o [ --outfile ] arg Output file name
  ```

- The quantized model can be loaded using the Arm NN Serialize/Deserialize API

- Primarily intended as an offline tool
Building the converter and quantizer applications

• To build the ArmnnConverter and ArmnnQuantizer applications you must turn on some options in the build control file armnn/cmake/GlobalConfig.cmake

  • ArmnnConverter
    - option(BUILD_ARMNN_SERIALIZER "Build Armnn Serializer" ON)
    - Plus, at least one of:
      - option(BUILD_CAFFE_PARSER "Build Caffe parser" ON)
      - option(BUILD_TF_PARSER "Build Tensorflow parser" ON)
      - option(BUILD_TF_LITE_PARSER "Build Tensorflow Lite parser" ON)
      - option(BUILD_ONNX_PARSER "Build Onnx parser" ON)

  • ArmnnQuantizer
    - option(BUILD_ARMNN_QUANTIZER "Build ArmNN quantizer" ON)
Useful links

https://review.mlplatform.org/#/admin/projects/ml/armnn
https://review.mlplatform.org/#/admin/projects/ml/ComputeLibrary
https://review.mlplatform.org/#/admin/projects/ml/android-nn-driver
https://github.com/ARM-software/armnn/blob/branches/armnn_19_02/src/armnnCaffeParser/README.md
https://github.com/ARM-software/armnn/blob/branches/armnn_19_02/src/armnnCaffeParser/CaffeSupport.md
https://github.com/ARM-software/armnn/blob/branches/armnn_19_02/src/armnnOnnxParser/README.md
https://github.com/ARM-software/armnn/blob/branches/armnn_19_02/src/armnnOnnxParser/OnnxSupport.md
https://github.com/ARM-software/armnn/blob/branches/armnn_19_02/src/armnnTfLiteParser/README.md
https://github.com/ARM-software/armnn/blob/branches/armnn_19_02/src/armnnTfLiteParser/TensorFlowLiteSupport.md
https://github.com/ARM-software/armnn/blob/branches/armnn_19_02/src/armnnTfParser/README.md