ARM Compute Library introduction

Nesterov Alexander, Obolenskiy Arseniy

ITLab

November 15, 2024

Contents

ARM Compute Library

arm COMPUTE LIBRARY

Compute Library latest release 24.09

The Compute Library is a collection of low-level machine learning functions optimized for Arm® Cortex®-A, Arm® Neoverse® and Arm® Mali™ GPUs architectures.

The library provides superior performance to other open source alternatives and immediate support for new Arm® technologies e.g. SVE2.

Key Features:

- Open source software available under a permissive MIT license
- Over 100 machine learning functions for CPU and GPU
- Multiple convolution algorithms (GeMM, Winograd, FFT, Direct and indirect-GeMM)
- Support for multiple data types: FP32, FP16, INT8, UINT8, BFLOAT16
- Micro-architecture optimization for key ML primitives
- Highly configurable build options enabling lightweight binaries
- Advanced optimization techniques such as kernel fusion, Fast math enablement and texture utilization
- Device and workload specific tuning using OpenCL tuner and GeMM optimized heuristics

Repository	Link
Release	https://github.com/arm-software/ComputeLibrary
Development	https://review.mlplatform.org/#/admin/projects/ml/ComputeLibrary

Source: https://github.com/ARM-software/ComputeLibrary

Supported Architectures/Technologies

Supported Architectures/Technologies

- Arm® CPUs:
  - Arm® Cortex®-A processor family using Arm® Neon™ technology
  - Arm® Neoverse® processor family
  - Arm® Cortex®-R processor family with Armv8-R AArch64 architecture using Arm® Neon™ technology
  - Arm® Cortex®-X1 processor using Arm® Neon™ technology
- Arm® Mali™ GPUs:
  - Arm® Mali™-G processor family
  - Arm® Mali™-T processor family
- x86

Source: https://github.com/ARM-software/ComputeLibrary

Supported Systems

Supported Systems

- Android™
- Bare Metal
- Linux®
- OpenBSD®
- macOS®
- Tizen™

Source: https://github.com/ARM-software/ComputeLibrary

Build ACL

Building for macOS

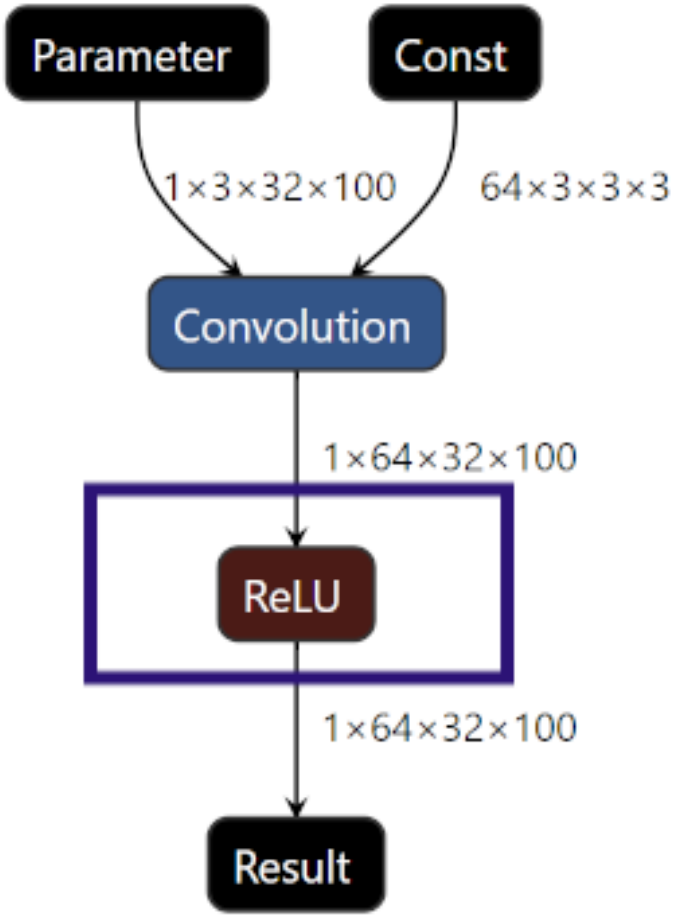
To natively compile the library with accelerated CPU support:

```
scons Werror=1 -j8 neon=1 opencl=0 os=macos arch=armv8.2-a build=native
```

```
nesterov@gkleinma-mobl ACL %
nesterov@gkleinma-mobl ACL % scons Werror=1 -j10 neon=1 opencl=0 os=macos arch=armv8.2-a build=native examples=1
```

Source:
https://artificial-intelligence.sites.arm.com/computelibrary/latest/how_to_build.xhtml

Return to examples



Source: docs.openvino.ai

ACL operators

Supported Operators

Supported Operators

Compute Library supports operators that are listed in below table.

Compute Library supports a wide list of data-types, information can been directly found in the documentation of each kernel/function. The main data-types that the Machine Learning functions support are the following:

- BFLOAT16: 16-bit non-standard brain floating point
- QASYMM8: 8-bit unsigned asymmetric quantized
- QASYMM8_SIGNED: 8-bit signed asymmetric quantized
- QSYMM8_PER_CHANNEL: 8-bit signed symmetric quantized (Used for the weights)
- QSYMM8: 8-bit unsigned symmetric quantized
- QSYMM16: 16-bit unsigned symmetric quantized
- F32: 32-bit single precision floating point
- F16: 16-bit half precision floating point
- S32: 32-bit signed integer
- U8: 8-bit unsigned char
- All: Agnostic to any specific data type

Compute Library supports the following data layouts (fast changing dimension from right to left):

- NHWC: The native layout of Compute Library that delivers the best performance where channels are in the fastest changing dimension
- NCHW: Legacy layout where width is in the fastest changing dimension
- NDHWC: New data layout for supporting 3D operators
- All: Agnostic to any specific data layout

where N = batches, C = channels, H = height, W = width, D = depth

Source:
https://artificial-intelligence.sites.arm.com/computelibrary/latest/operators_list.xhtml

ACL activation operator

Function	Description	Equivalent Android NNAPI Op	Backends	Data Layouts	
ActivationLayer	Function to simulate an activation layer with the specified activation function.	• ANEURALNETWORKS_ELU • ANEURALNETWORKS_HARD_SWISH • ANEURALNETWORKS_LOGISTIC • ANEURALNETWORKS_RELU • ANEURALNETWORKS_RELU1 • ANEURALNETWORKS_RELU6 • ANEURALNETWORKS_TANH	NEActivationLayer	• All	src / dst QASYMM8 / QASYMM8 QASYMM8_SIGNED / QASYMM8_SIGNED QSYMM16 / QSYMM16 F16 / F16 F32 / F32
			CLActivationLayer	• All	src / dst QASYMM8 / QASYMM8 QASYMM8_SIGNED / QASYMM8_SIGNED QSYMM16 / QSYMM16 F16 / F16 F32 / F32

Source:
https://artificial-intelligence.sites.arm.com/computelibrary/latest/operators_list.xhtml

Validate activation operator

◆ validate()

```
static Status validate ( const ITensorInfo *          input,
                         const ITensorInfo *          output,
                         const ActivationLayerInfo &  act_info
                       )
```

[NEActivationLayer snippet]

Static function to check if given info will lead to a valid configuration of NEActivationLayer

Parameters

- [in] **input** Source tensor info. In case of `output` tensor info = nullptr, this tensor will store the result of the activation function. Data types supported: QASYMM8/QASYMM8_SIGNED/QSYMM16/F16/F32.
- [in] **output** Destination tensor info. Data type supported: same as `input`
- [in] **act_info** Activation layer information.

Returns

a status

Source: Description of activation operator

TensorInfo for operators

TensorInfo() [10/11]

```
TensorInfo ( const TensorShape &  tensor_shape,
             size_t               num_channels,
             DataType             data_type,
             DataLayout           data_layout
           )
```

Constructor.

Parameters

[in]	**tensor_shape**	It specifies the size for each dimension of the tensor in number of elements.
[in]	**num_channels**	It indicates the number of channels for each tensor element
[in]	**data_type**	Data type to use for each tensor element
[in]	**data_layout**	The data layout setting for the tensor data.

Source: Description of TensorInfo

Configure activation operator

◆ configure()

```
void configure ( ITensor *          input,
                 ITensor *          output,
                 ActivationLayerInfo activation_info
               )
```

[NEActivationLayer snippet]

Set the input and output tensor.

Valid data layouts:

- All

Valid data type configurations:

src	dst
QASYMM8	QASYMM8
QASYMM8_SIGNED	QASYMM8_SIGNED
QSYMM16	QSYMM16
F16	F16
F32	F32

Note
If the output tensor is a nullptr or is equal to the input, the activation function will be performed in-place

Parameters

[in,out]	**input**	Source tensor. In case of `output` tensor = nullptr, this tensor will store the result of the activation function. Data types supported: QASYMM8/QASYMM8_SIGNED/QSYMM16/F16/F32.
[out]	**output**	Destination tensor. Data type supported: same as `input`
[in]	**activation_info**	Activation layer parameters.

Source: Description of activation operator

Tensor for operators

Public Member Functions

	Tensor (**IRuntimeContext** *ctx=nullptr) Constructor. More...
	~Tensor ()=default Destructor: free the tensor's memory. More...
	Tensor (**Tensor** &&)=default Allow instances of this class to be move constructed. More...
Tensor &	**operator=** (**Tensor** &&)=default Allow instances of this class to be moved. More...
TensorAllocator *	**allocator** () Return a pointer to the tensor's allocator. More...
ITensorInfo *	**info** () const override Interface to be implemented by the child class to return the tensor's metadata. More...
ITensorInfo *	**info** () override Interface to be implemented by the child class to return the tensor's metadata. More...
uint8_t *	**buffer** () const override Interface to be implemented by the child class to return a pointer to CPU memory. More...
void	**associate_memory_group** (**IMemoryGroup** *memory_group) override Associates a memory managable object with the memory group that manages it. More...

Source: Description of Tensor

Run activation operator

◆ run()

void run ( )

Run the kernels contained in the function.

For CPU kernels:

- Multi-threading is used for the kernels which are parallelisable.
- By default std::thread::hardware_concurrency() threads are used.

Note

CPPScheduler::set_num_threads() can be used to manually set the number of threads

For OpenCL kernels:

- All the kernels are enqueued on the queue associated with **CLScheduler**.
- The queue is then flushed.

Note

The function will not block until the kernels are executed. It is the user's responsibility to wait.

Will call **prepare()** on first run if hasn't been done

Implements **IFunction**.

Source: Description of activation operator

Get ONNX model

Usage

Python CLI

```python
from ultralytics import YOLO

# Load the YOLO11 model
model = YOLO("yolo11n.pt")

# Export the model to ONNX format
model.export(format="onnx")  # creates 'yolo11n.onnx'

# Load the exported ONNX model
onnx_model = YOLO("yolo11n.onnx")

# Run inference
results = onnx_model("https://ultralytics.com/images/bus.jpg")
```

Source: https://docs.ultralytics.com/integrations/onnx/