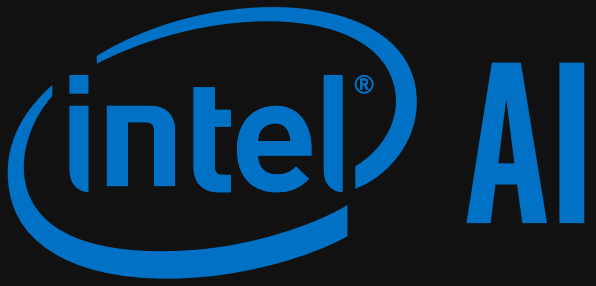
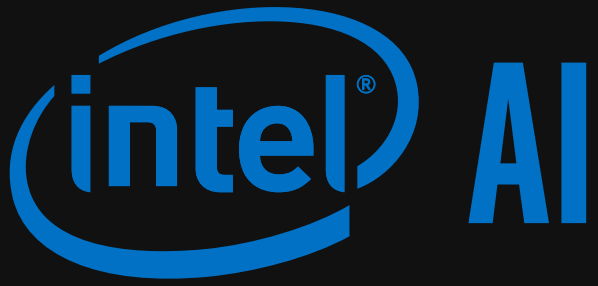
DEEP LEARNING

on the inside

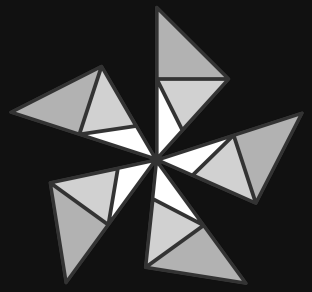

intel®
AI



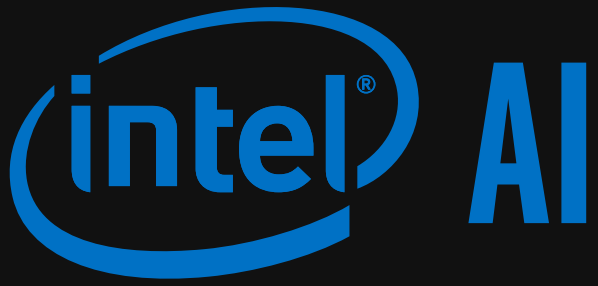
nGraph

ONNX

OpenVINO



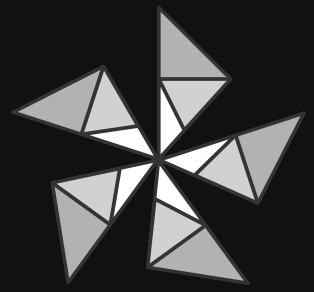
ONNXRuntime



nGraph

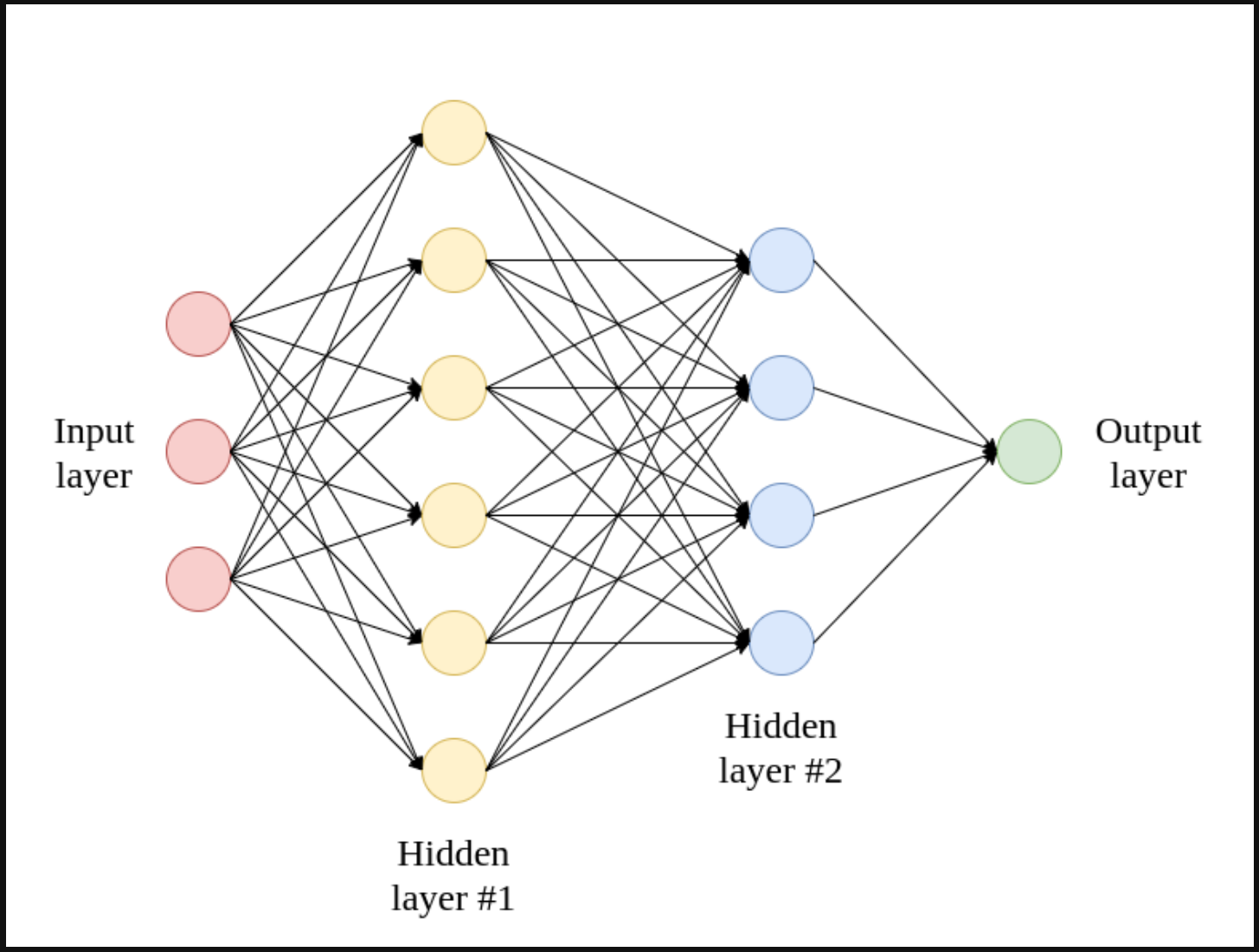
ONNX

OpenVINO

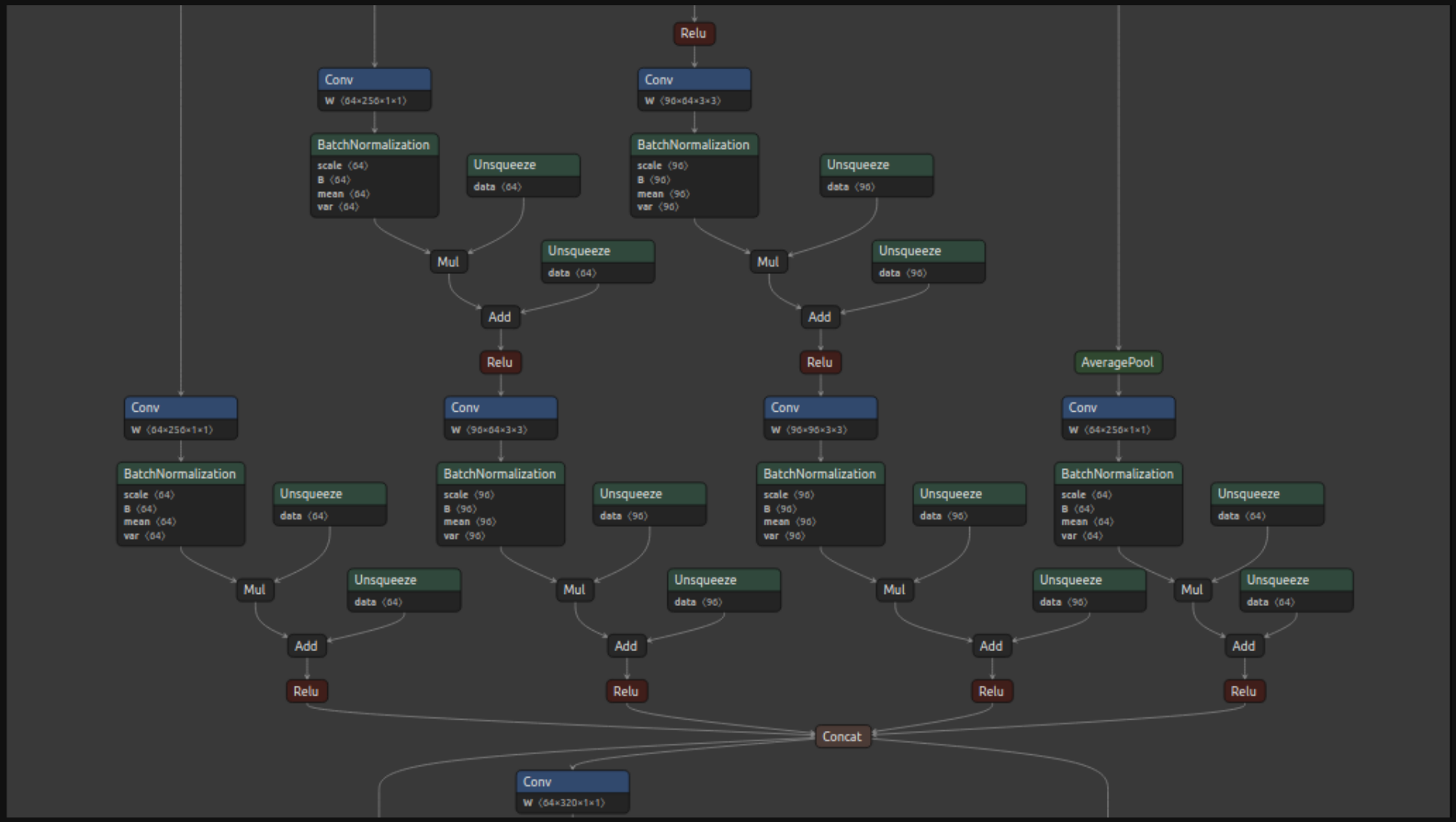


ONNXRuntime

Artificial neural networks

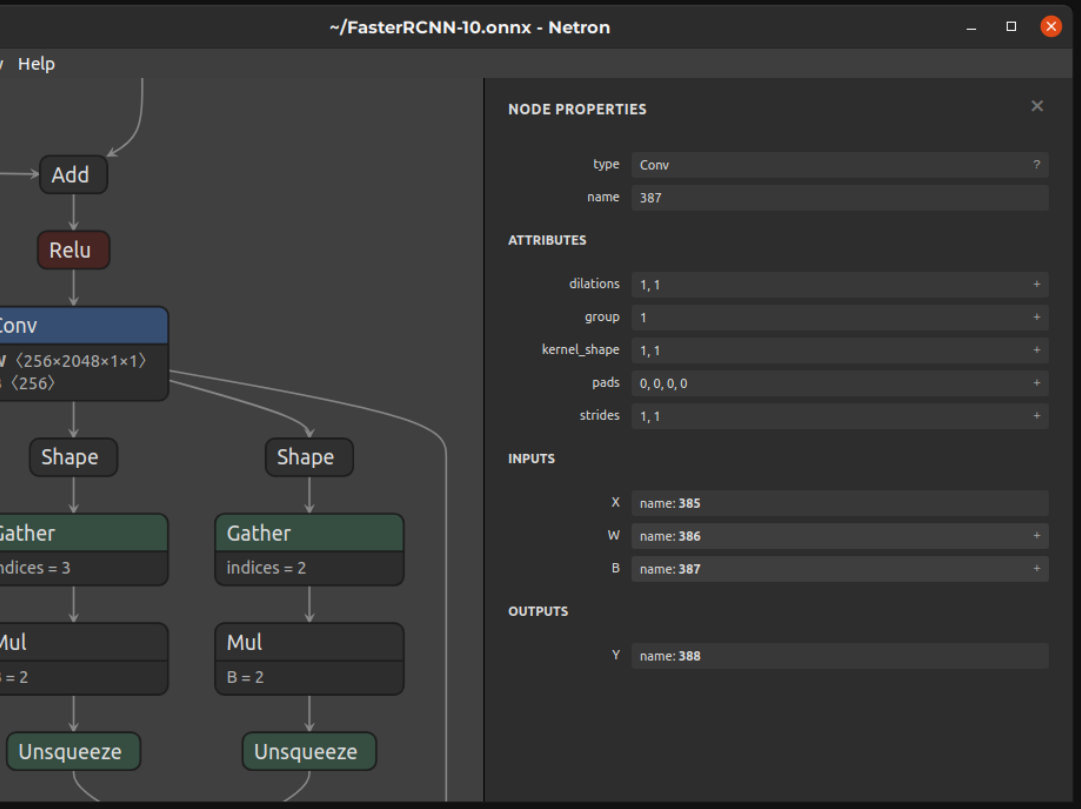


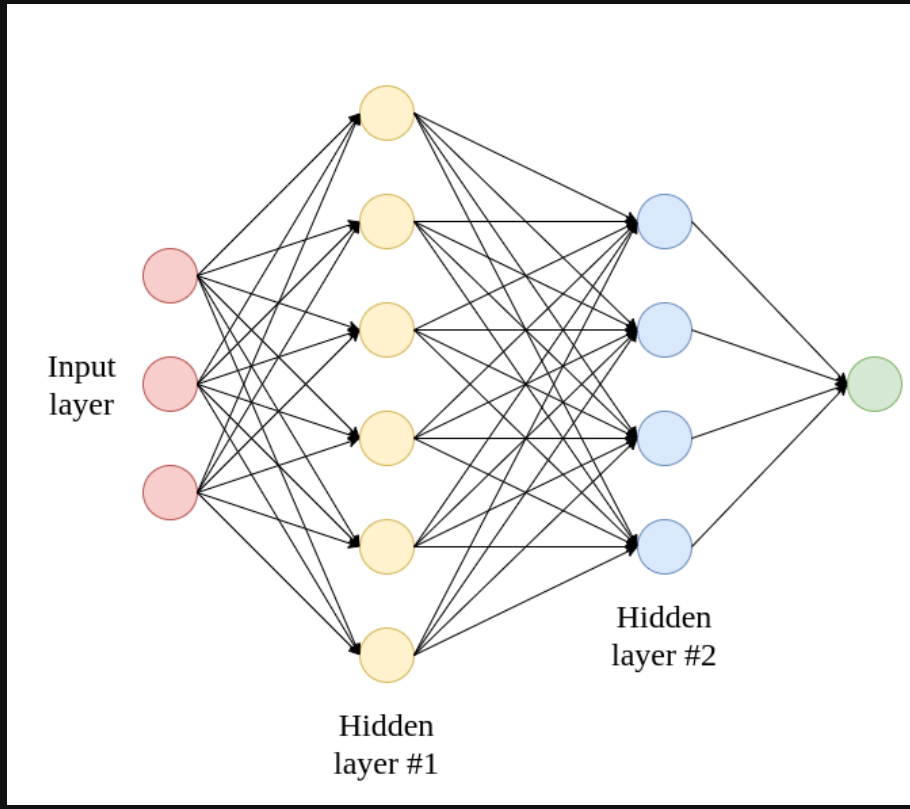
Real-life models



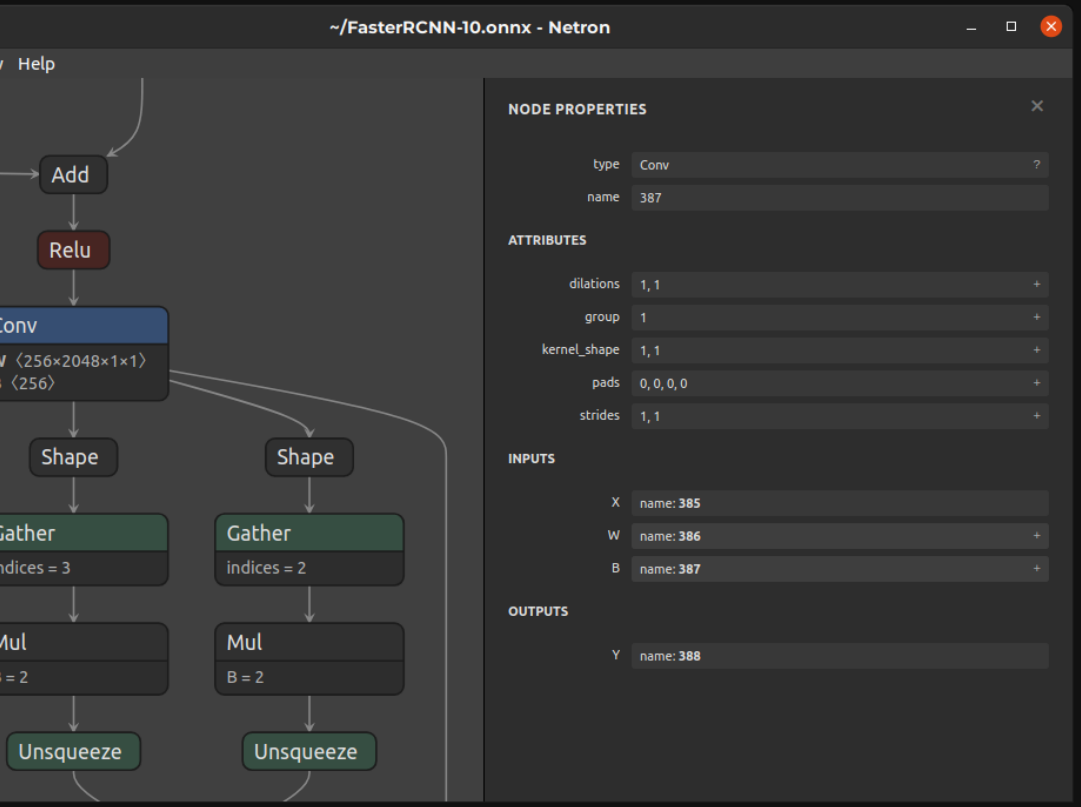
(fragment of the Faster R-CNN model)

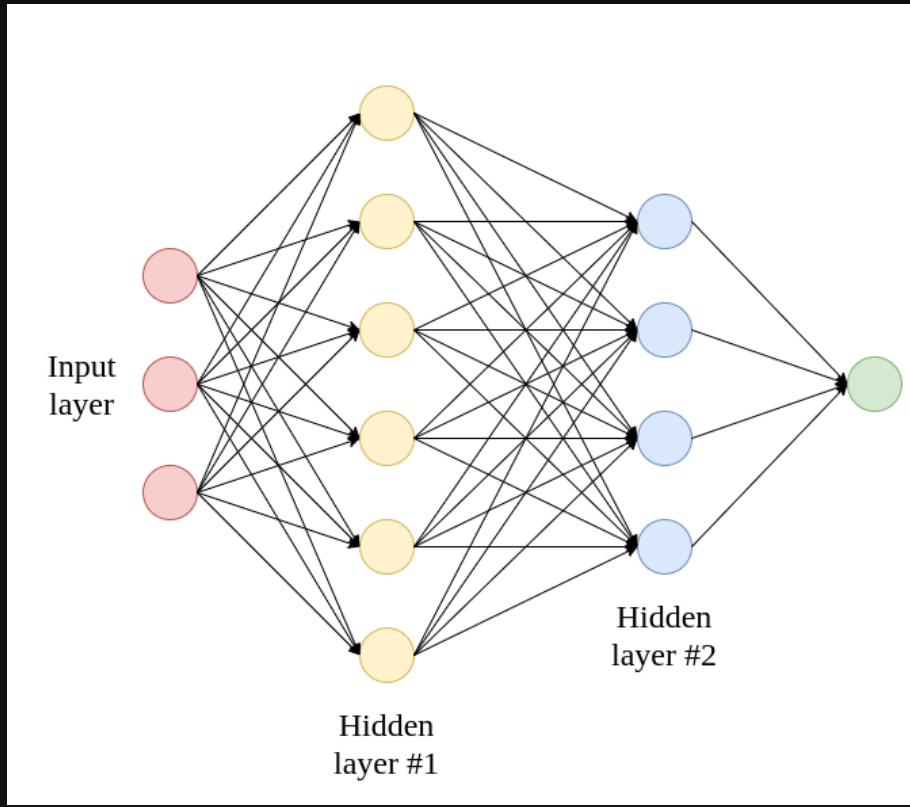
Graph representations





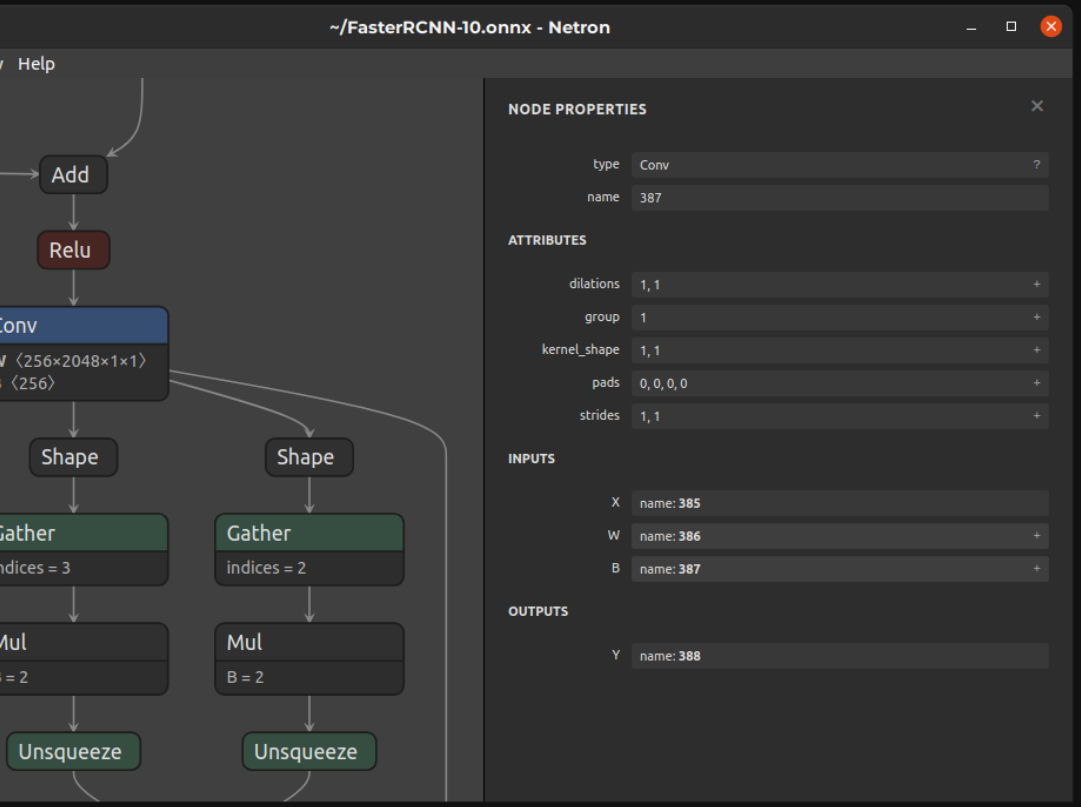
Graph representations

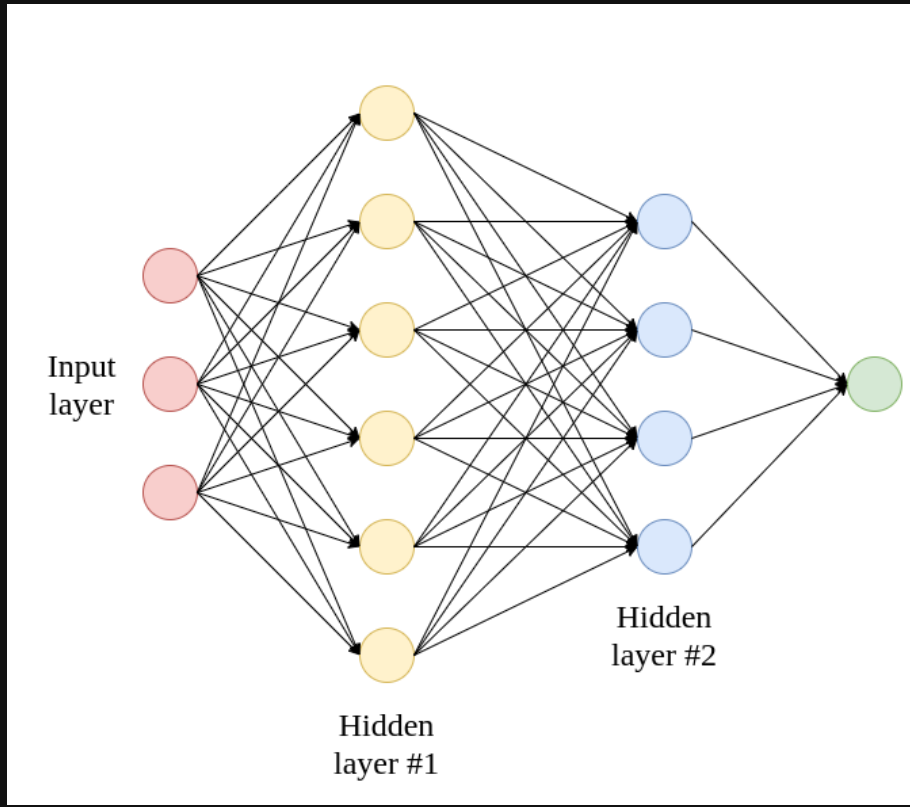




- nodes: operations *vs* individual neurons

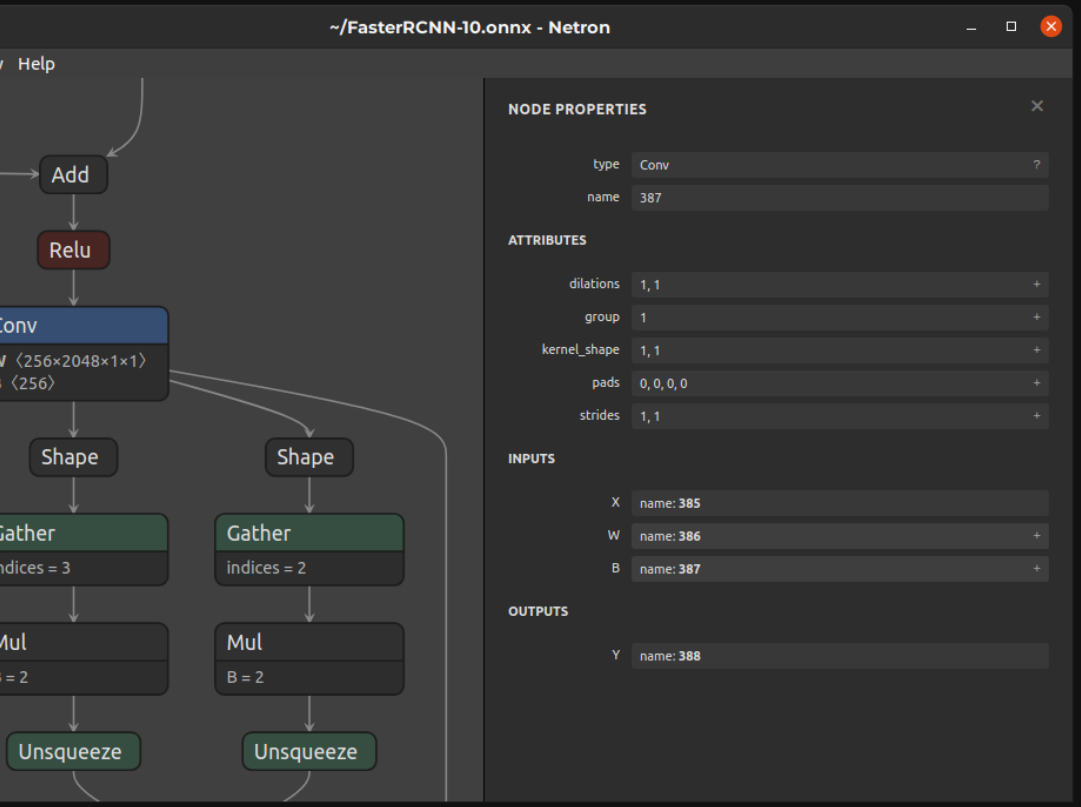
Graph representations

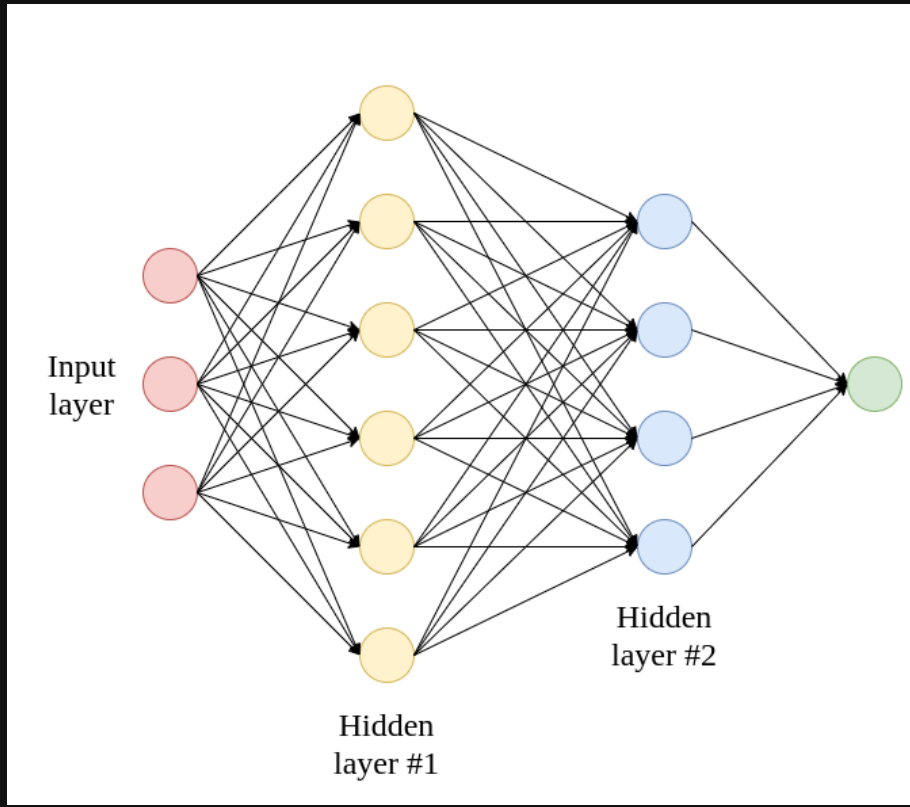




- nodes: operations *vs* individual neurons
- edges: tensors *vs* individual weights

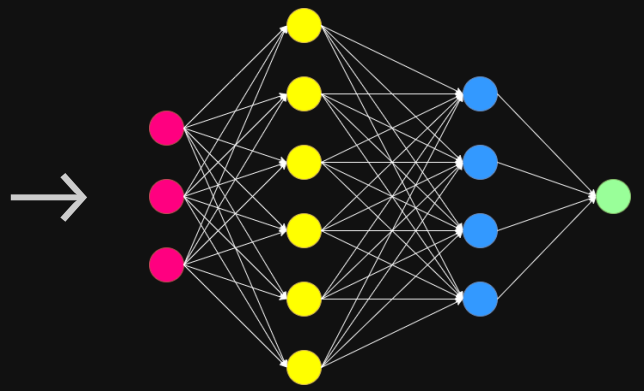
Graph representations





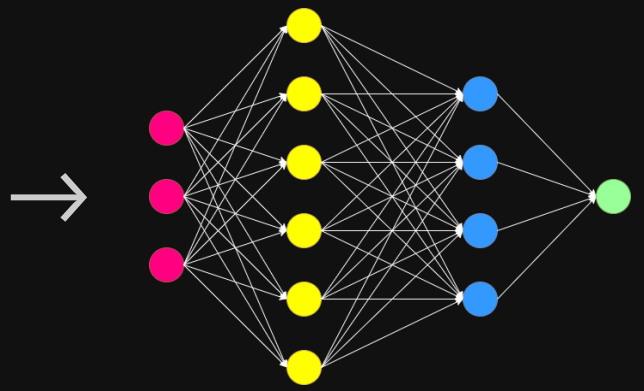
- nodes: operations *vs* individual neurons
- edges: tensors *vs* individual weights
- graph: computational graph *vs* linear graph with layers

Model as a function



DOG

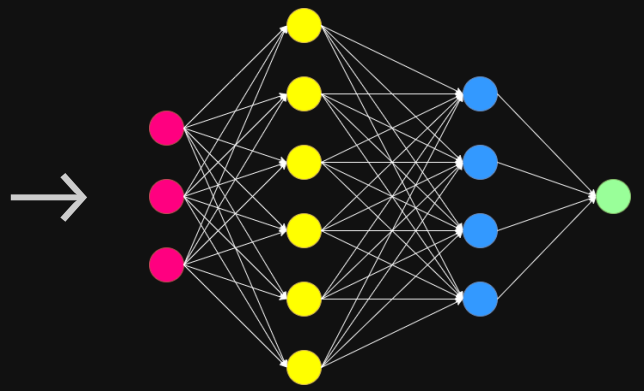
Model as a function



DOG

$$y = f(x)$$

Model as a function

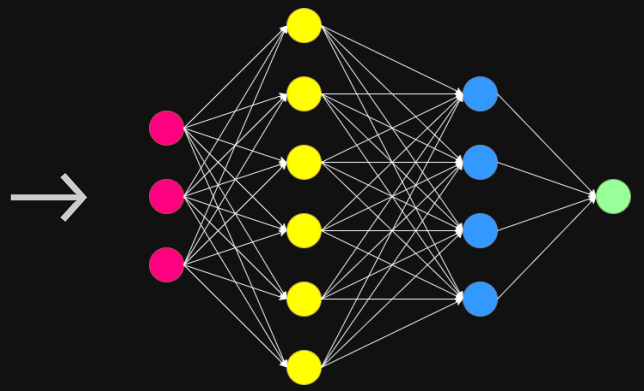


DOG

$$y = f(x)$$

1 ⇒ DOG

Model as a function



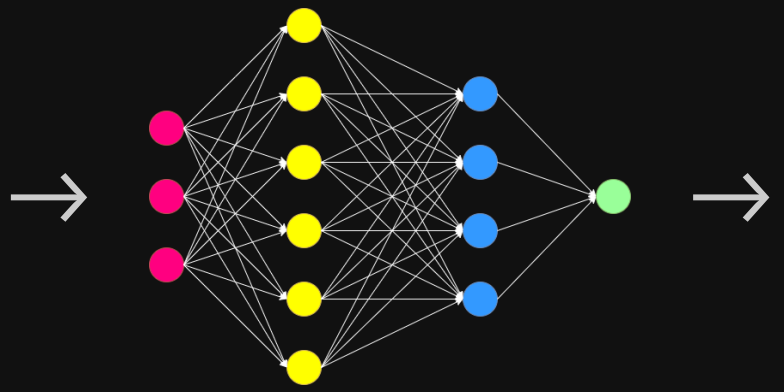
DOG

$$y = f(x)$$

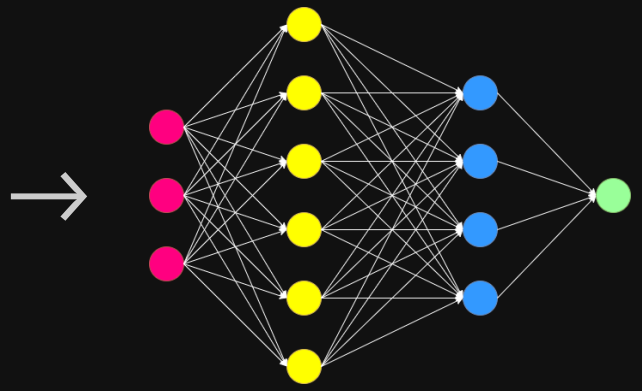
1 ⇒ DOG

0 ⇒ NOT A DOG

Model as a function

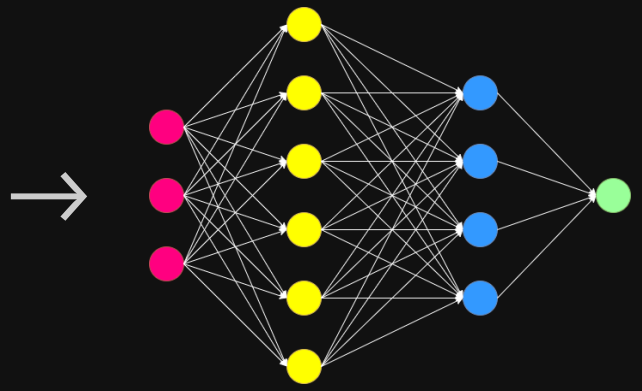


Model as a function



WTF?

Model as a function



WTF?

model = approximation of a function

with a certain accuracy

Universal approximation theorem

Universal approximation theorems imply that neural networks can represent a wide variety of interesting functions when given appropriate weights.

en.wikipedia.org/wiki/Universal_approximation_theorem

Deep Learning model

from a C++ developer's perspective

Deep Learning model

from a C++ developer's perspective

```
// generates a class of functions
template <float A, float B>
float linear_function(const float x)
{
    return A * x + B;
}

float linear_function(const float x) {
    return 3.14 * x + 1.618;
}
```

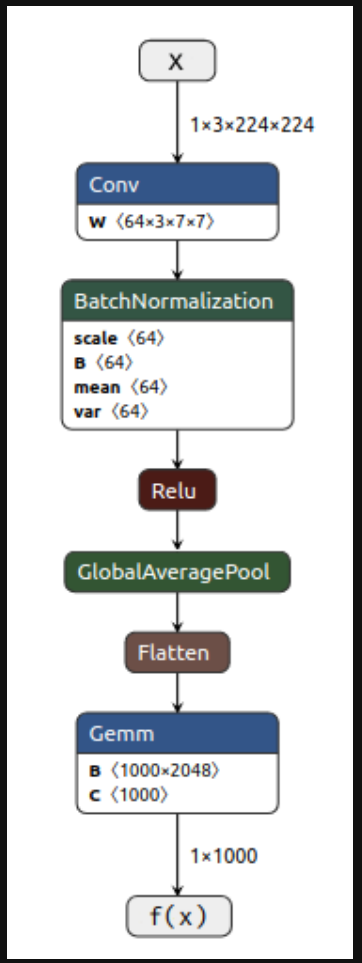
Deep Learning model

from a *generic* developer's perspective

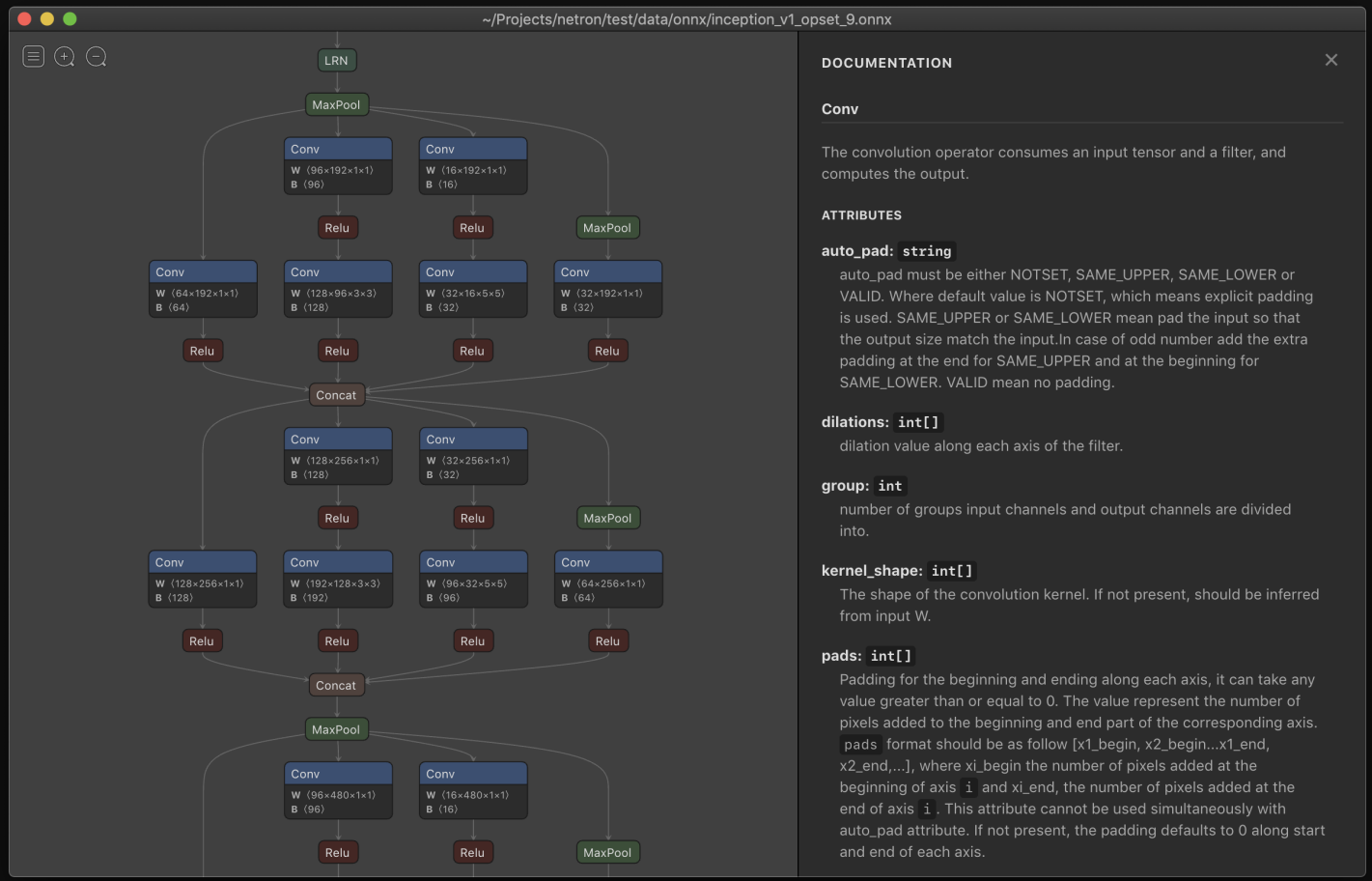
Deep Learning model

from a *generic* developer's perspective

$$f(x) = Gemm(Flatten(GlobalAveragePool(Relu(BatchNormalization(Conv(x)$$


X
1×3×224×224
Conv
W 〈64×3×7×7〉
BatchNormalization
scale 〈64〉
B 〈64〉
mean 〈64〉
var 〈64〉
Relu
GlobalAveragePool
Flatten
Gemm
B 〈1000×2048〉
C 〈1000〉
1×1000
f(x)

Graphs in Deep Learning



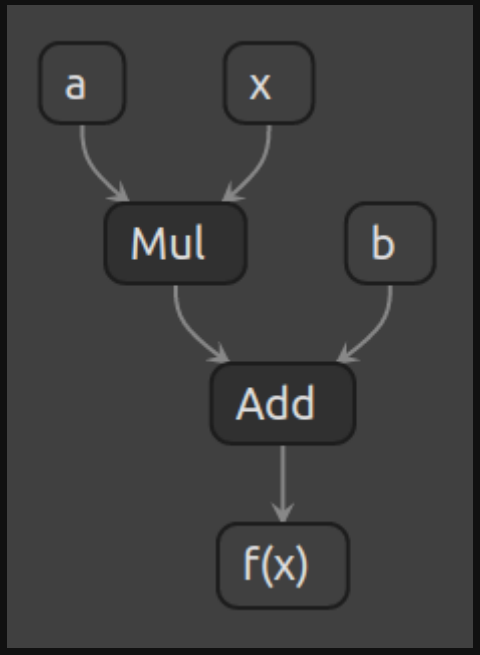
lutzroeder/netron https://netron.app

model → function → graph

$$f(x) = ax + b$$

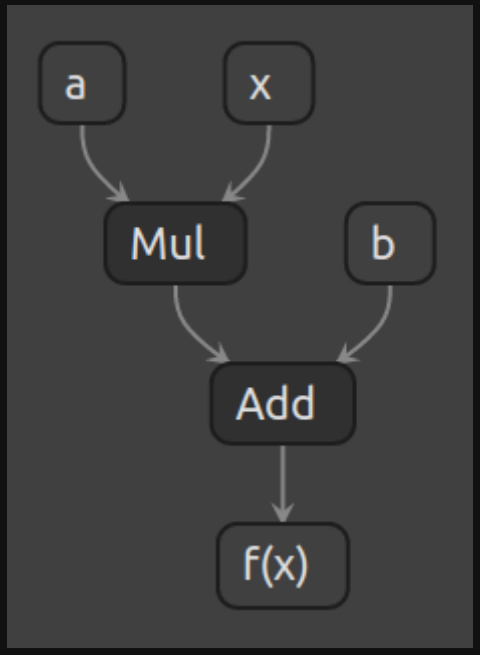
model → function → graph

$$f(x) = ax + b$$



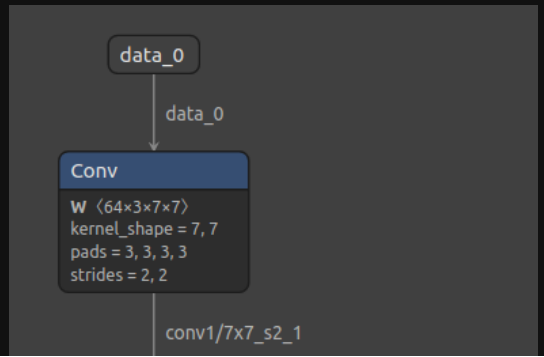
model → function → graph

$$f(x) = ax + b$$

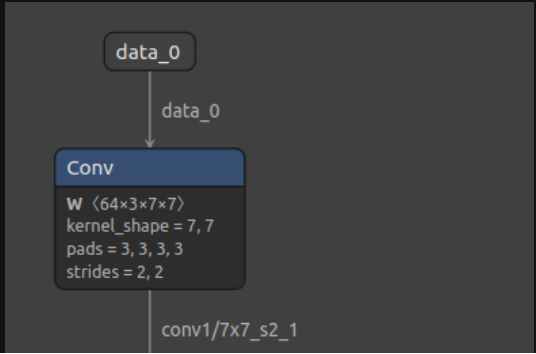


computational graphs

Operators


data_0
data_0
Conv
W 〈64×3×7×7〉
kernel_shape = 7, 7
pads = 3, 3, 3, 3
strides = 2, 2
conv1/7x7_s2_1

Operators


data_0
data_0
Conv
W 〈64×3×7×7〉
kernel_shape = 7, 7
pads = 3, 3, 3, 3
strides = 2, 2
conv1/7x7_s2_1

Tensors

Tensors

- N-dimentional data arrays (float, int, int8, int64, ...)

Tensors

- N-dimentional data arrays (float, int, int8, int64, ...)
- Flat, contiguous blocks of allocated memory

Tensors

- N-dimentional data arrays (float, int, int8, int64, ...)
- Flat, contiguous blocks of allocated memory
- Shape - {1, 3, 224, 224} or {3, 3}

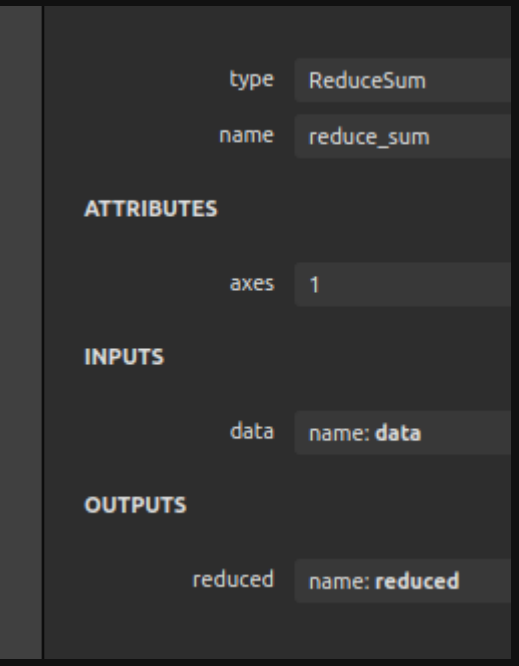
Tensors

- N-dimentional data arrays (float, int, int8, int64, ...)
- Flat, contiguous blocks of allocated memory
- Shape - {1, 3, 224, 224} or {3, 3}
- Dimensions (static or dynamic)

Tensors

- N-dimentional data arrays (float, int, int8, int64, ...)
- Flat, contiguous blocks of allocated memory
- Shape - {1, 3, 224, 224} or {3, 3}
- Dimensions (static or dynamic)
- Rank

ReduceSum of a 2D tensor




```
const auto data = Tensor({1, 2, 3, 4, 5, 6}, {
// [[1 2 3],
//  [4 5 6]]
const auto over_axis_1 = ReduceSum(data, 1);
// [6, 15]
const auto over_axis_0 = ReduceSum(data, 0);
// [5, 7, 9]
```

Inference

~/Projects/netron/test/data/onnx/inception_v1_opset_9.onnx

DOCUMENTATION

Conv

The convolution operator consumes an input tensor and a filter, and computes the output.

ATTRIBUTES

auto_pad: `string`

auto_pad must be either NOTSET, SAME_UPPER, SAME_LOWER or VALID. Where default value is NOTSET, which means explicit padding is used. SAME_UPPER or SAME_LOWER mean pad the input so that the output size match the input.In case of odd number add the extra padding at the end for SAME_UPPER and at the beginning for SAME_LOWER. VALID mean no padding.

dilations: `int[]`

dilation value along each axis of the filter.

group: `int`

number of groups input channels and output channels are divided into.

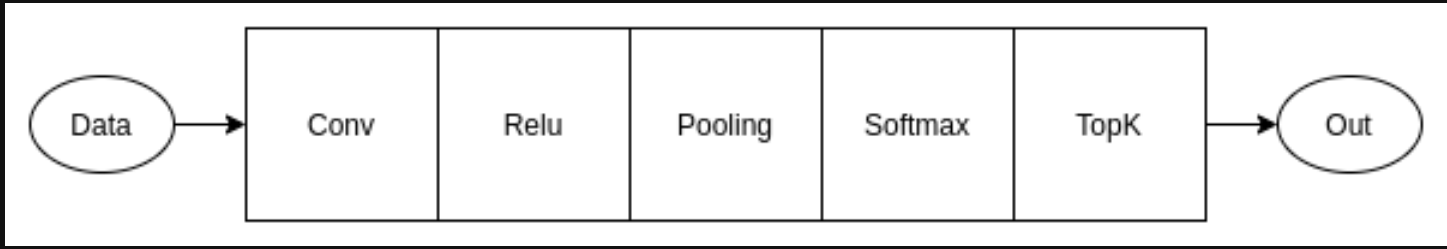
kernel_shape: `int[]`

The shape of the convolution kernel. If not present, should be inferred from input W.

pads: `int[]`

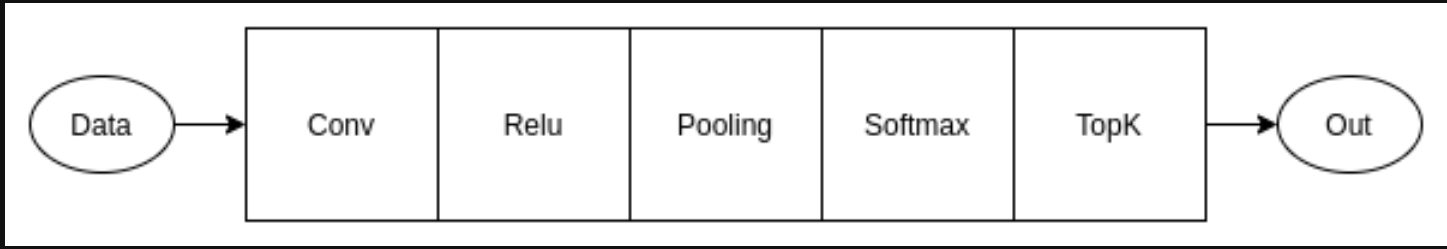
Padding for the beginning and ending along each axis, it can take any value greater than or equal to 0. The value represent the number of pixels added to the beginning and end part of the corresponding axis. `pads` format should be as follow [x1_begin, x2_begin...x1_end, x2_end,...], where xi_begin the number of pixels added at the beginning of axis `i` and xi_end, the number of pixels added at the end of axis `i`. This attribute cannot be used simultaneously with auto_pad attribute. If not present, the padding defaults to 0 along start and end of each axis.

Naïve implementation



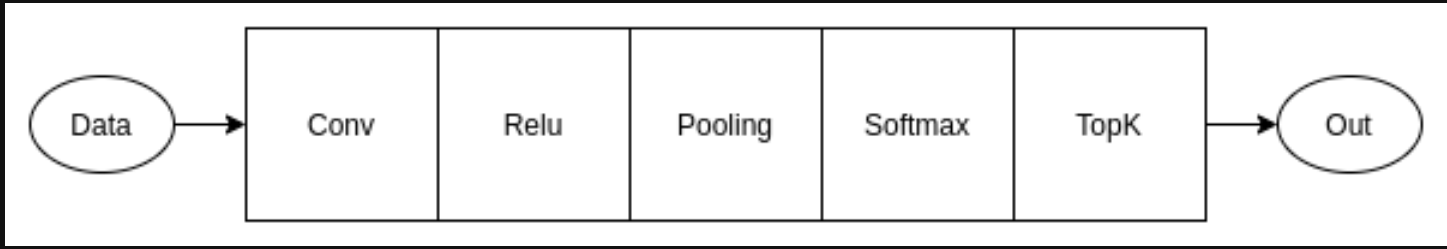
- Topological sort of the graph

Naïve implementation



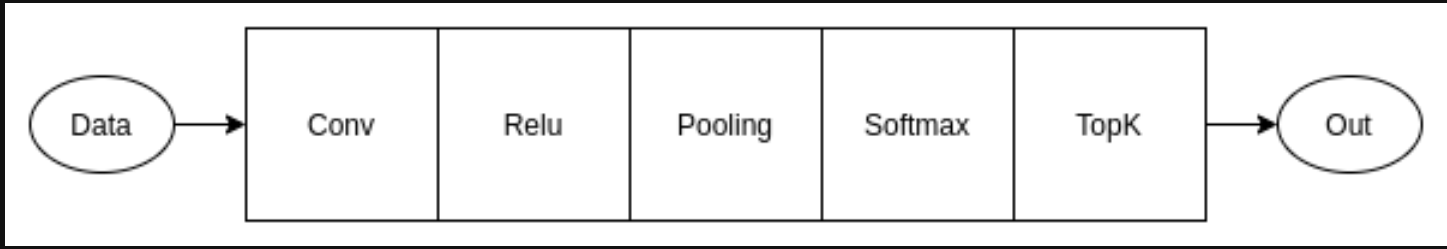
- Topological sort of the graph
- Implementation of the operators

Naïve implementation



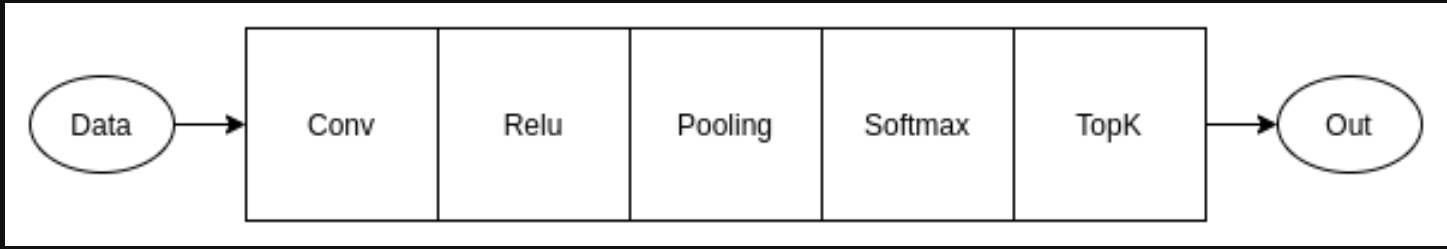
- Topological sort of the graph
- Implementation of the operators
- Sequential execution of sorted nodes

Naïve implementation



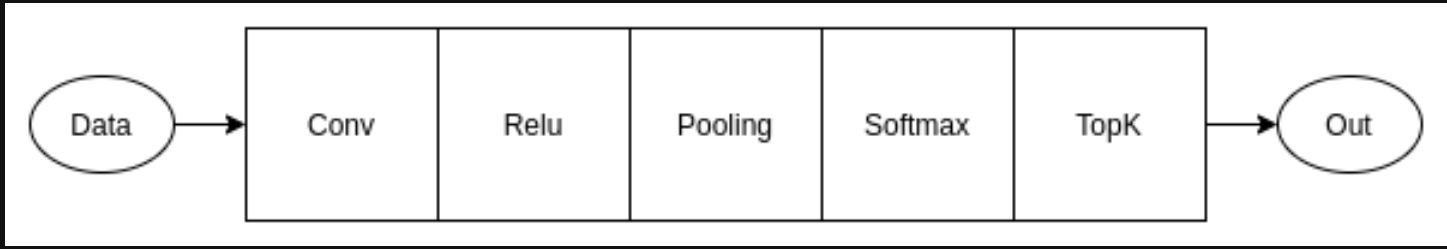
- Topological sort of the graph
- Implementation of the operators
- Sequential execution of sorted nodes
- Probably wasting a lot of HW potential

Naïve implementation



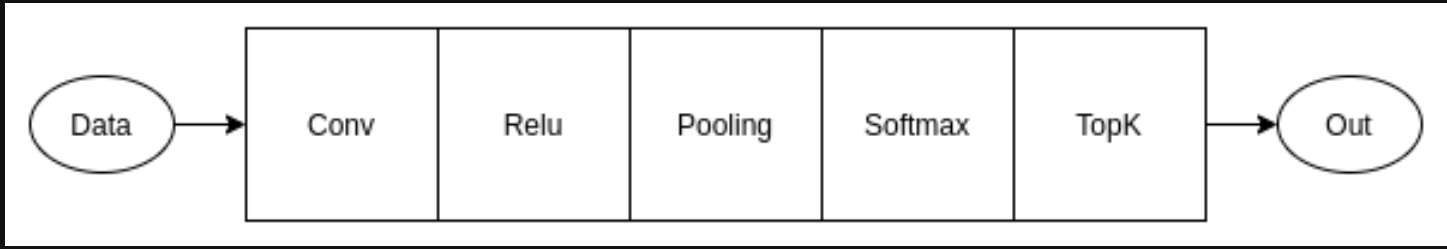
- Topological sort of the graph
- Implementation of the operators
- Sequential execution of sorted nodes
- Probably wasting a lot of HW potential
- Cores not utilized

Naïve implementation



- Topological sort of the graph
- Implementation of the operators
- Sequential execution of sorted nodes
- Probably wasting a lot of HW potential
- Cores not utilized
- No vectorization

Naïve implementation



- Topological sort of the graph
- Implementation of the operators
- Sequential execution of sorted nodes
- Probably wasting a lot of HW potential
- Cores not utilized
- No vectorization
- Not good enough for big, real-life models

Kernels

Kernels

- Highly specialized functions/primitives

Kernels

- Highly specialized functions/primitives
- Hardware specific

Kernels

- Highly specialized functions/primitives
- Hardware specific
- Architecture & available instructions dependent

Kernels

- Highly specialized functions/primitives
- Hardware specific
- Architecture & available instructions dependent
- Data type, data shape (rank)

Kernels

- Highly specialized functions/primitives
- Hardware specific
- Architecture & available instructions dependent
- Data type, data shape (rank)
- oneDNN - Intel CPU & GPU

Kernels

- Highly specialized functions/primitives
- Hardware specific
- Architecture & available instructions dependent
- Data type, data shape (rank)
- oneDNN - Intel CPU & GPU

 oneapi-src/oneDNN

Example: Binary Convolution

Direct Optimization

Direct Optimization

- DL framework + kernels = performance

Direct Optimization

- DL framework + kernels = performance
- N frameworks * M hardware platforms = a lot of work

Direct Optimization

- DL framework + kernels = performance
- N frameworks * M hardware platforms = a lot of work
- K operations

Direct Optimization

- DL framework + kernels = performance
- N frameworks * M hardware platforms = a lot of work
- K operations in each framework

Direct Optimization

- DL framework + kernels = performance
- N frameworks * M hardware platforms = a lot of work
- K operations in each framework = even more work

Direct Optimization

- DL framework + kernels = performance
- N frameworks * M hardware platforms = a lot of work
- K operations in each framework = even more work
- HW independent optimizations?

Intel® nGraph™

NervanaSystems/ngraph

openvinotoolkit/openvino

from the technical standpoint

from the technical standpoint

- a library

from the technical standpoint

- a library
-  inside

from the technical standpoint

- a library
-  inside
-  python API

from the functional standpoint

from the functional standpoint

- HW independent IR of DL models

from the functional standpoint

- HW independent IR of DL models
- graph compiler

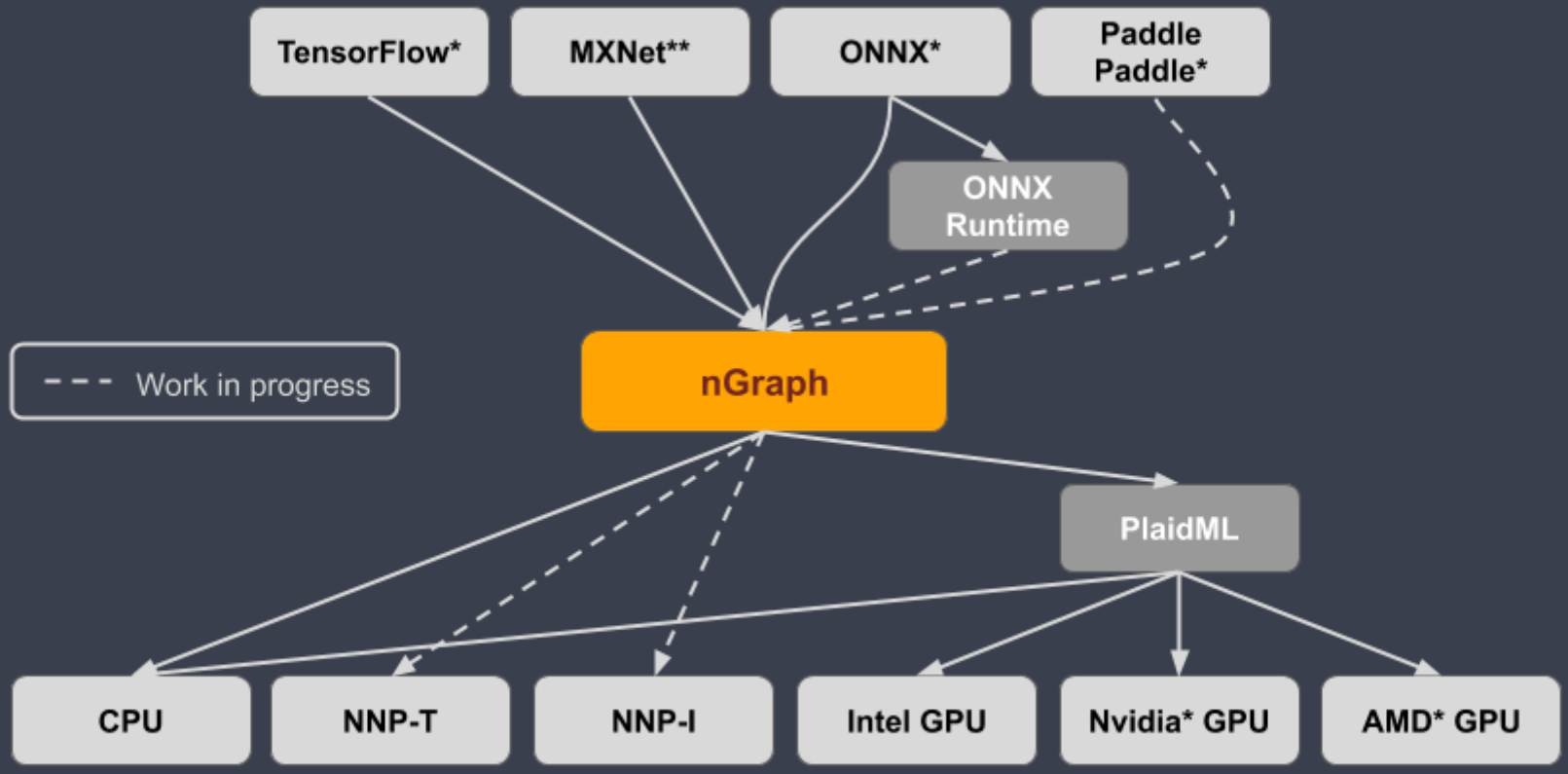
from the functional standpoint

- HW independent IR of DL models
- graph compiler
- an optimizing compiler

from the functional standpoint

- HW independent IR of DL models
- graph compiler
- an optimizing compiler
- `ngraph::Function`

nGraph


TensorFlow*
MXNet**
ONNX*
Paddle Paddle*
ONNX Runtime
Work in progress
nGraph
PlaidML
CPU
NNP-T
NNP-I
Intel GPU
Nvidia* GPU
AMD* GPU

Graph optimizations

Graph optimizations

- HW independent

Graph optimizations

- HW independent
- Operate on an `ngraph::Function`

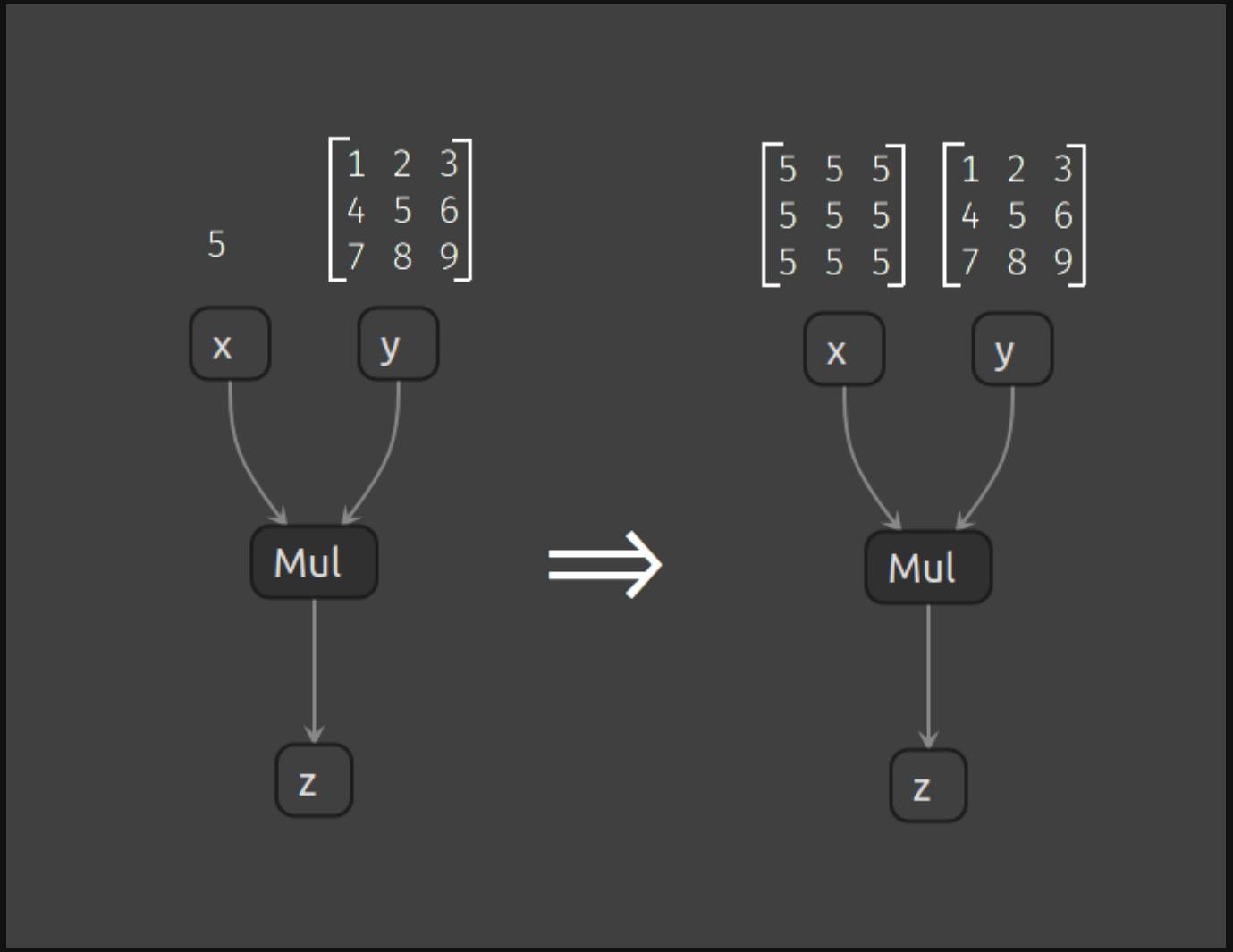
Graph optimizations

- HW independent
- Operate on an `ngraph::Function`
- Graph pattern matching & replacement

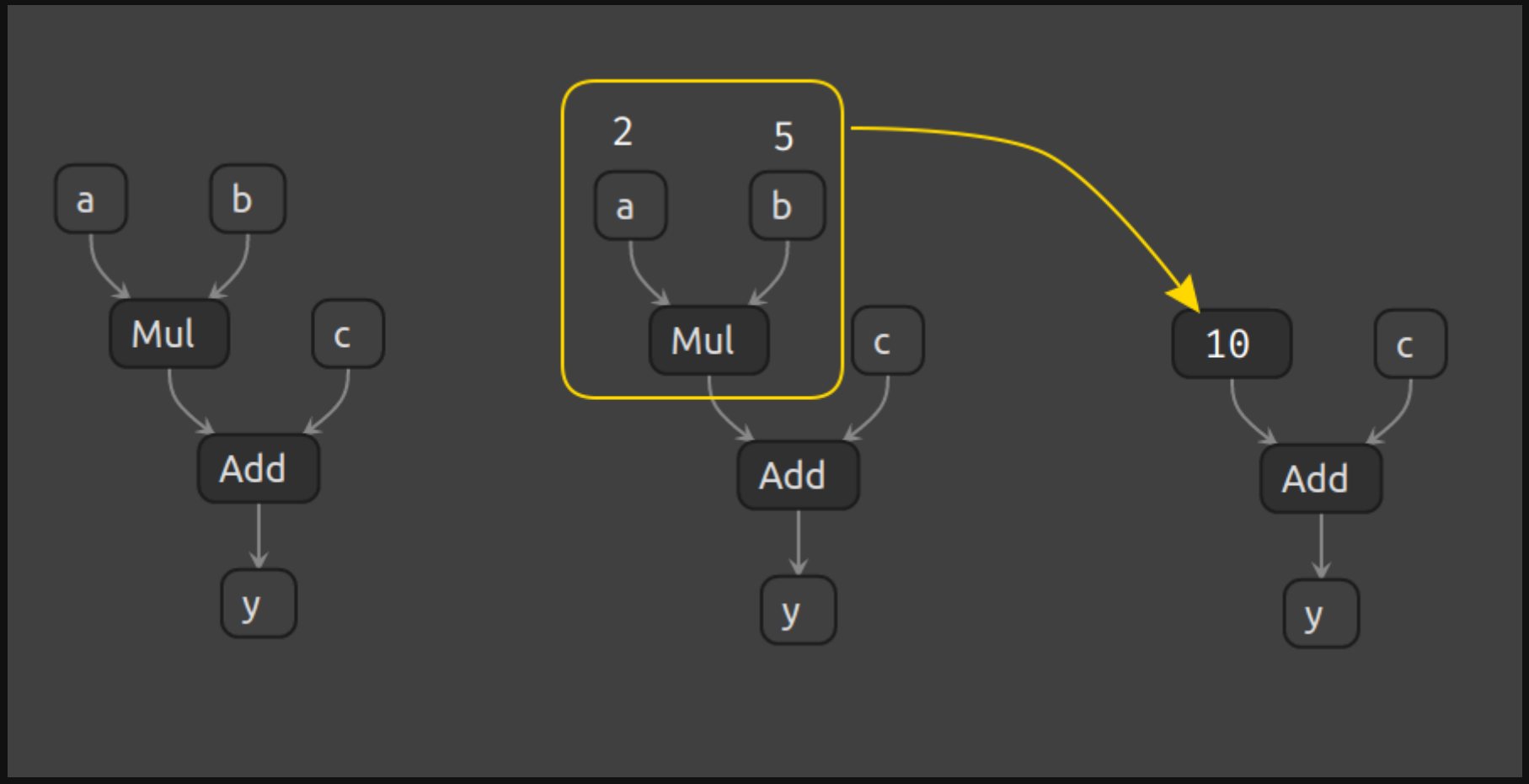
Graph optimizations

- HW independent
- Operate on an `ngraph::Function`
- Graph pattern matching & replacement
- "Function passes"

Implicit broadcast



Constant folding



NOP elimination

- Elimination of "useless" nodes

NOP elimination

- Elimination of "useless" nodes
- Convert $int \to int$

NOP elimination

- Elimination of "useless" nodes
- Convert $int \rightarrow int$
- Broadast or Reshape when

  $target\_shape == input\_shape$

NOP elimination

- Elimination of "useless" nodes
- Convert $int \rightarrow int$
- Broadast or Reshape when

  $target\_shape == input\_shape$
- Reduction without a given axis

NOP elimination

- Elimination of "useless" nodes
- Convert $int \rightarrow int$
- Broadast or Reshape when

  $target\_shape == input\_shape$
- Reduction without a given axis
- Addition of zero or multiplication by one

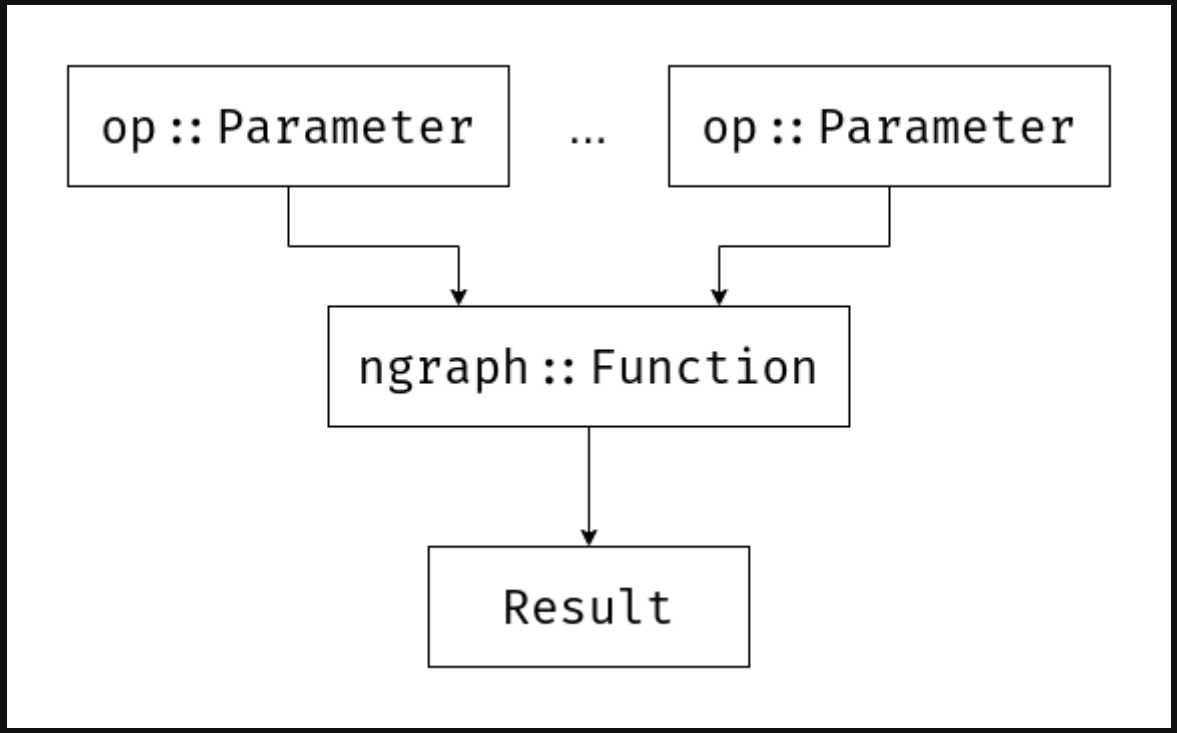
NOP elimination

- Elimination of "useless" nodes
- Convert $int \rightarrow int$
- Broadast or Reshape when

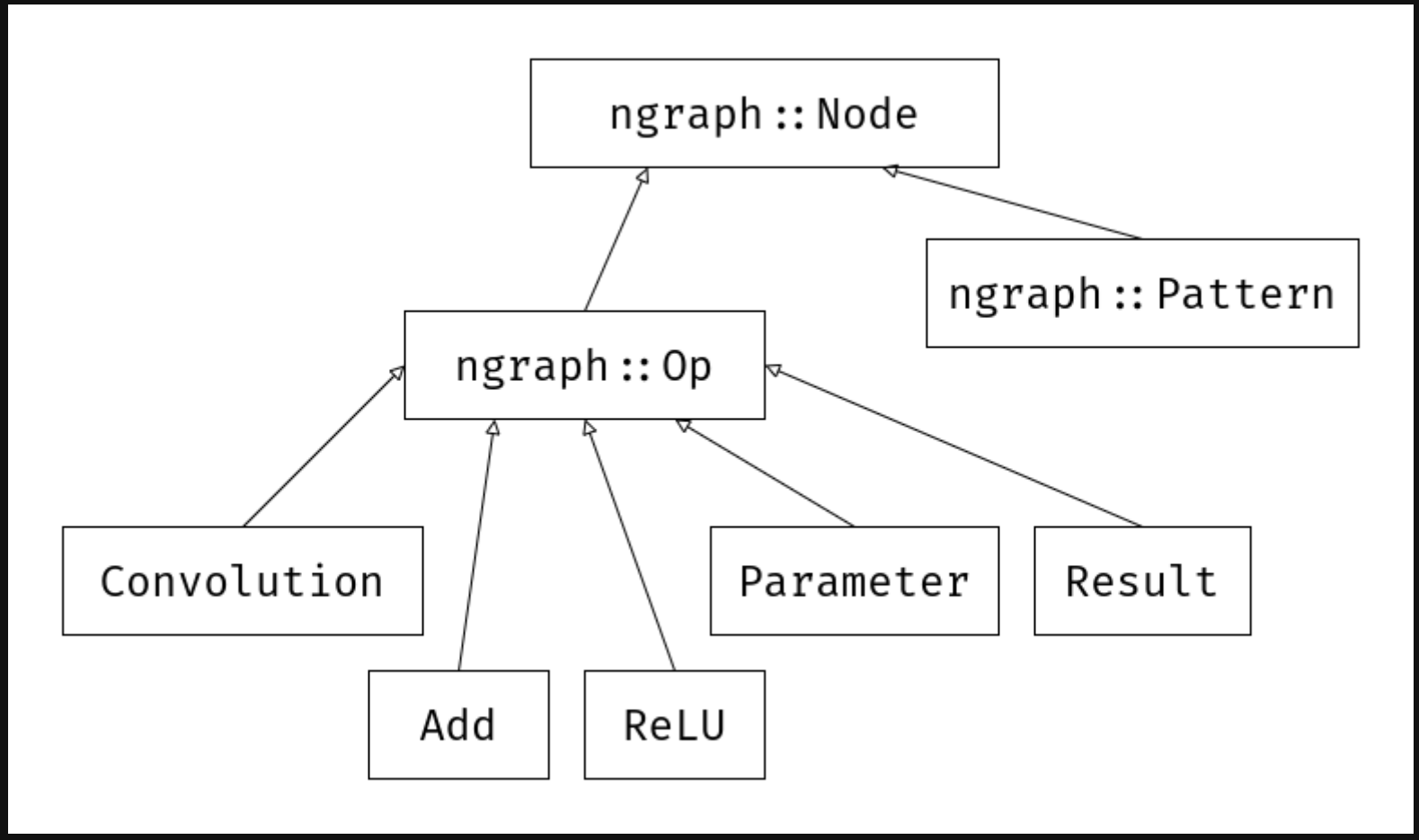
  $target\_shape == input\_shape$
- Reduction without a given axis
- Addition of zero or multiplication by one

common_optimizations.cpp

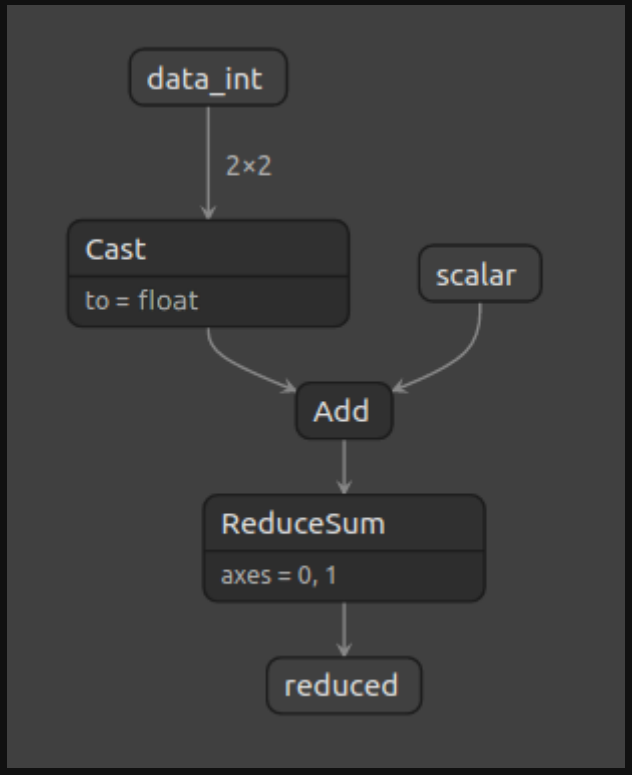
ngraph::Function



ngraph::Node


ngraph::Node
ngraph::Pattern
ngraph::Op
Convolution
Add
ReLU
Parameter
Result

Creating a simple Function



Creating a simple Function

```
const auto data_int = make_shared<op::Parameter>(element::i32, Shape{2, 2});
const auto scalar = make_shared<op::Parameter>(element::f32, Shape{});

const auto cast = make_shared<op::Convert>(data_int, element::f32);
const auto add = make_shared<op::Add>(cast, scalar);

const auto reduction_axes = op::Constant::create(element::i32, Shape{2}, {0, 1});
const auto reduce_sum = make_shared<op::ReduceSum>(add, reduction_axes);

const auto function = make_shared<Function>(reduce_sum, ParameterVector{data_int, scalar});
```

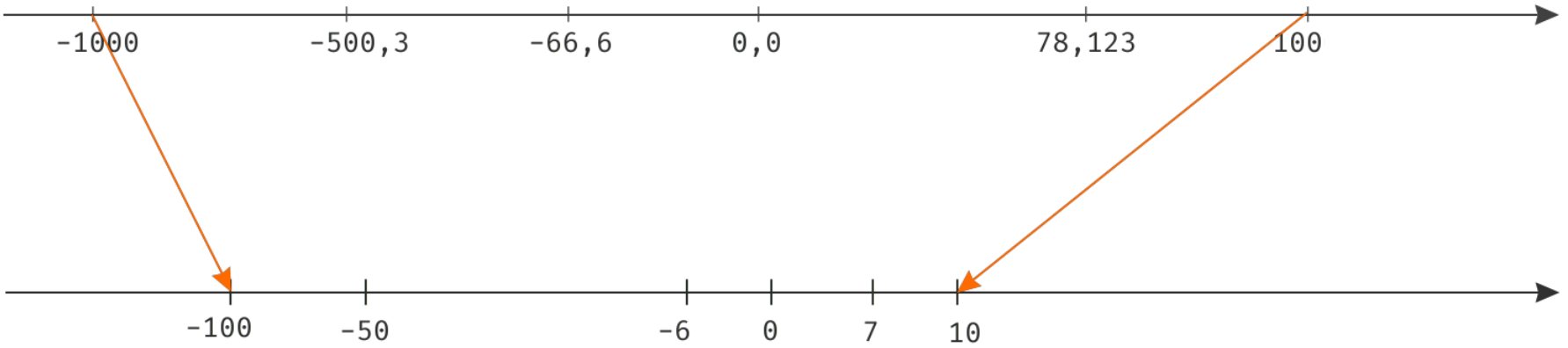
Creating a simple Function

```
const auto data_int = make_shared<op::Parameter>(element::i32, Shape{2, 2});
const auto scalar = make_shared<op::Parameter>(element::f32, Shape{});

const auto cast = make_shared<op::Convert>(data_int, element::f32);
const auto add = make_shared<op::Add>(cast, scalar);

const auto reduction_axes = op::Constant::create(element::i32, Shape{2}, {0, 1});
const auto reduce_sum = make_shared<op::ReduceSum>(add, reduction_axes);

const auto function = make_shared<Function>(reduce_sum, ParameterVector{data_int, scalar});
```

ideone.com/TJQ1IL

Quantization

$$q(r) = \lfloor r/s \rfloor + z$$



#include <limits>

- `std::numeric_limits<float>`

$$1 \times 10^{-38} \longleftrightarrow 3 \times 10^{38}$$

#include <limits>

- `std::numeric_limits<float>`

$$1 \times 10^{-38} \longleftrightarrow 3 \times 10^{38}$$

- `std::numeric_limits<int8_t>`

$$-128 \longleftrightarrow 127$$

#include <limits>

- `std::numeric_limits<float>`

$$1 \times 10^{-38} \longleftrightarrow 3 \times 10^{38}$$

- `std::numeric_limits<int8_t>`

$$-128 \longleftrightarrow 127$$

- `std::numeric_limits<uint8_t>`

$$0 \longleftrightarrow 255$$

Quantization

$$q(r) = \lfloor r/s \rfloor + z$$

REAL VALUE	SCALE	SCALE	SCALE	SCALE	ZERO POINT
	2	**4**	**8**	**50**	0
-1000,000	-500	-250	-125	-20	
-97,555	-49	-24	-12	-2	
-66,600	-33	-17	-8	-1	
0,000	0	0	0	0	
1,618	1	0	0	0	
3,142	2	1	0	0	
20,000	10	5	3	0	
1000,000	500	250	125	20	

Joules and m^2 per operation

Operation	Energy (pJ)	Area (μm^2)
`int8` addition	0.03	36
`int16` addition	0.05	67
`int32` addition	0.1	137
`float16` addition	0.4	1,360
`float32` addition	0.9	4,184
`int8` multiplication	0.2	282
`int32` multiplication	3.1	3,495
`float16` multiplication	1.1	1,640
`float32` multiplication	3.7	7,700

Quantization benefits

- Less power (edge devices)

Quantization benefits

- Less power (edge devices)
- Smaller chip

Quantization benefits

- Less power (edge devices)
- Smaller chip
- Better cache utilization

Quantization benefits

- Less power (edge devices)
- Smaller chip
- Better cache utilization
- Smaller weights == smaller model

Quantization benefits

- Less power (edge devices)
- Smaller chip
- Better cache utilization
- Smaller weights == smaller model
- Performance

HW support - VNNI

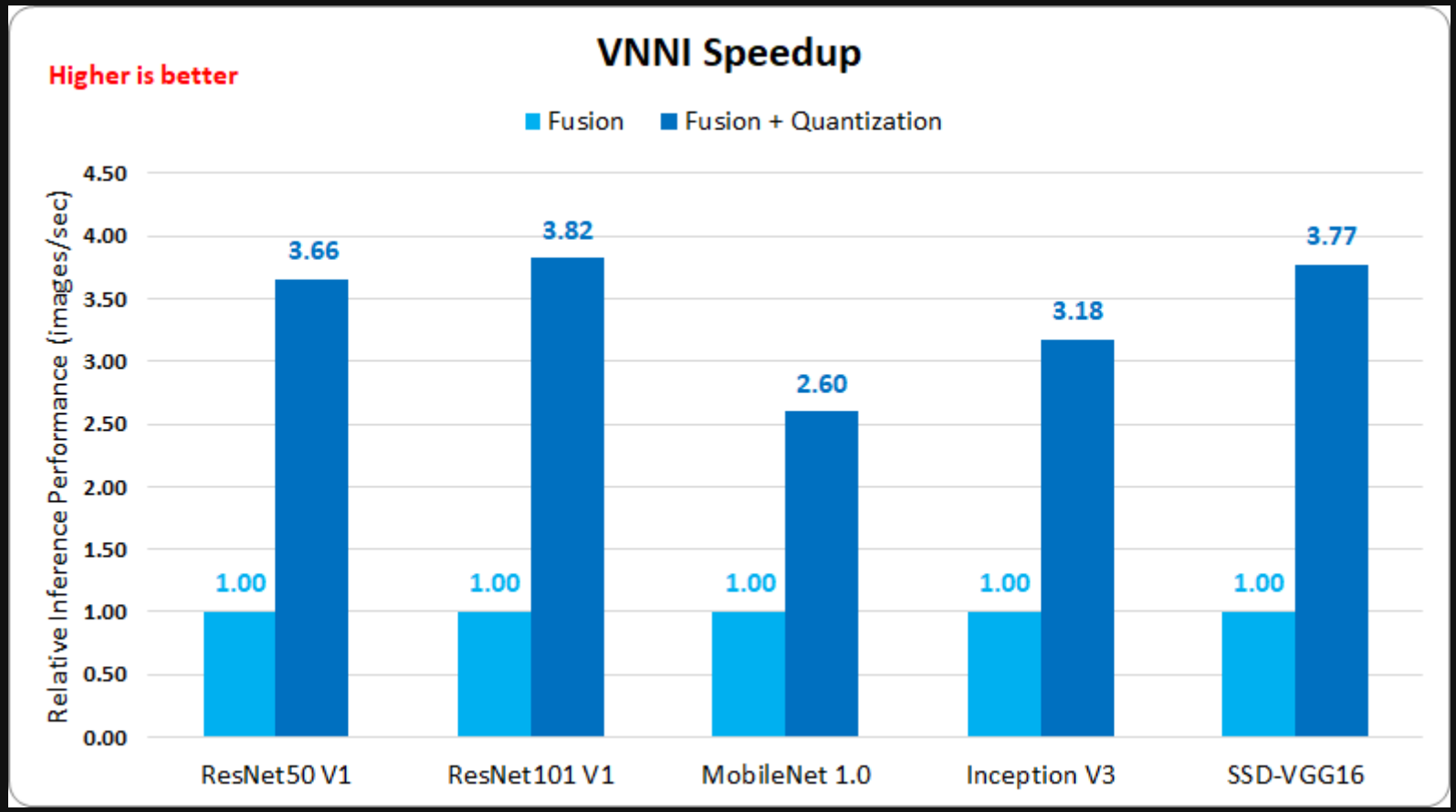
intel.ai/intel-deep-learning-boost

avx-512-animation.html

lowering-numerical-precision-increase-deep-learning-performance.html

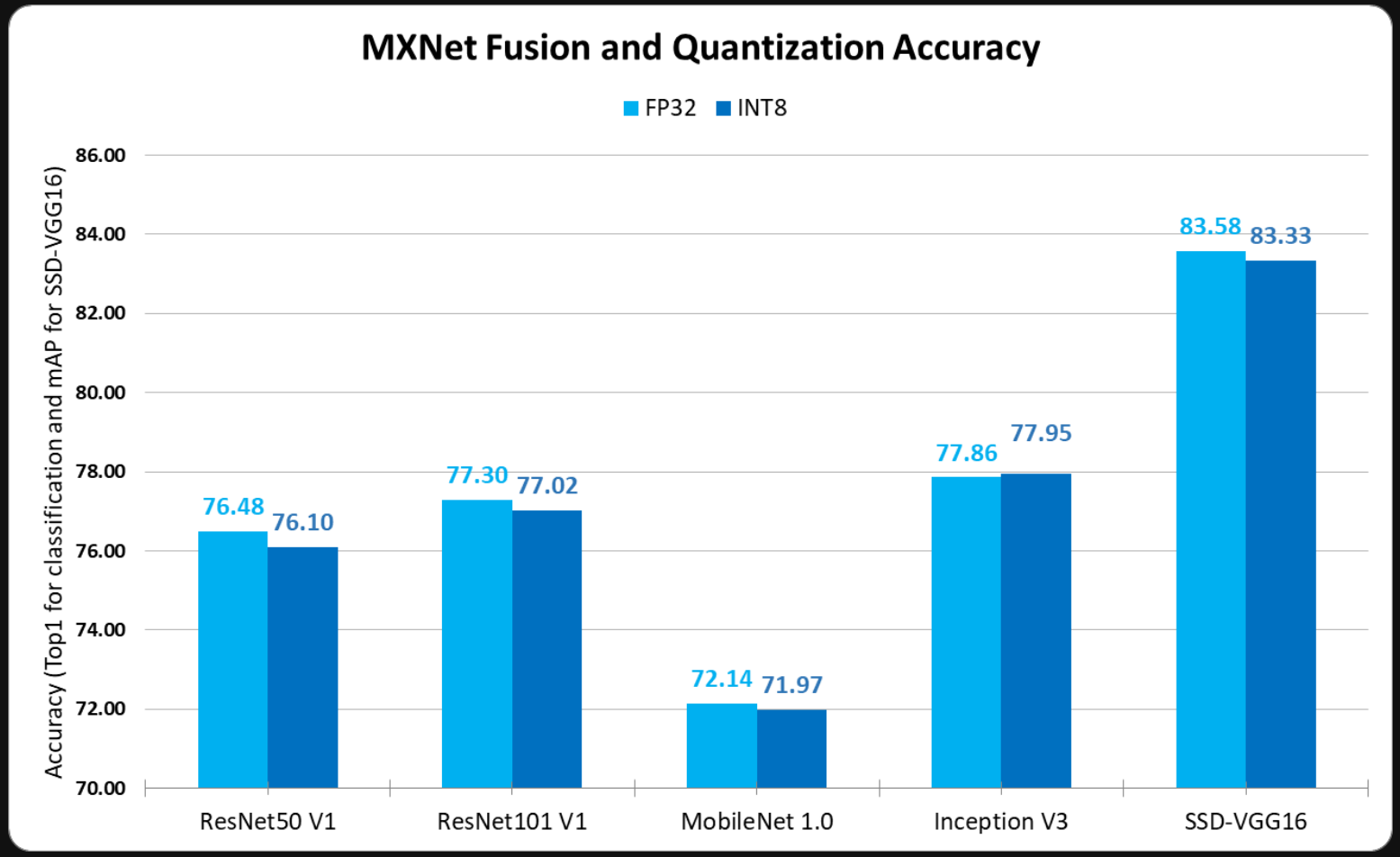
AVX512_VNNI

fp32 vs int8



medium.com/apache-mxnet/model-quantization-for-production-level-neural-network-inference-f54462ebba05

What about accuracy?



medium.com/apache-mxnet/model-quantization-for-production-level-neural-network-inference-f54462ebba05

DEEP LEARNING

on the inside

tomdol.github.io/dl-on-the-inside

www.youtube.com/watch?v=C9S0nmNS8bQ

Materials

- docs.openvinotoolkit.org/latest/index.html
- docs.openvinotoolkit.org/latest/omz_models_public_index.html
- docs.openvinotoolkit.org/latest/omz_demos.html
- github.com/openvinotoolkit/openvino
- github.com/openvinotoolkit/model_server
- github.com/openvinotoolkit/openvino_notebooks
- onnx.ai
- github.com/onnx/models
- 3Blue1Brown - neural nets
- coursera.org/specializations/deep-learning
- infoshare.pl