

Intel Performance optimizations for Deep Learning

Dr. Séverine Habert– AI Engineering Manager

Severine.habert@intel.com

21 April 2022



intel®



lrz



TUM

Agenda

- Data precision
- Optimized DL frameworks
 - oneDNN
 - Tensorflow
 - PyTorch
 - Intel Extension for PyTorch
- Intel[®] Neural Compressor

Before we start

- What are data precision?

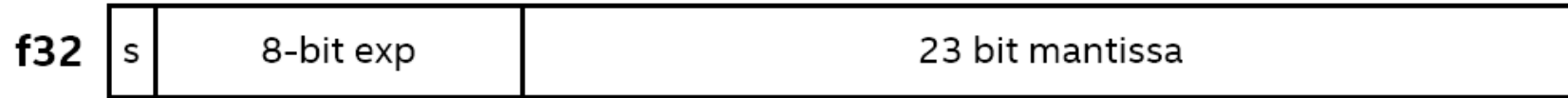
Number of bits used in memory to store a value

- Commonly found data precisions in Deep Learning are:

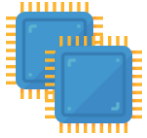


FP32 for Training

- The standard for Deep Learning training is to remain on FP32



The motivation for lower precision



**Lower memory
bandwidth**



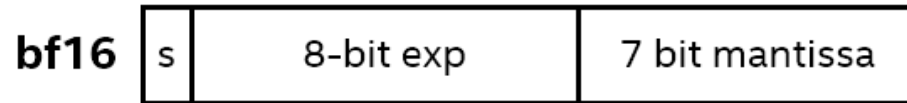
**Lower
storage**



**Higher
performance**

While retaining an
acceptable accuracy loss

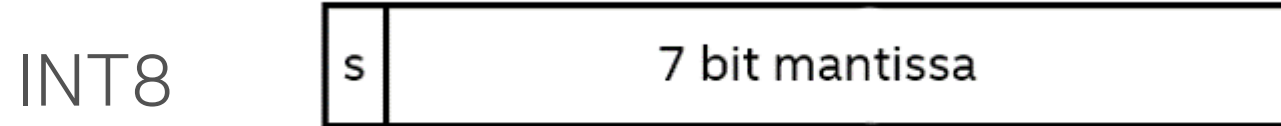
16-bits data precision



Benefit of BF16:

- Performance 2x up
- Comparable accuracy loss against fp32
- No loss scaling, compared to fp16
- Can be used for training (mixed-precision training)

8-bits data precision



- Not suitable for training but recommended for inference when a small loss of accuracy is acceptable
- Significant speed-up compared to FP32 inference

Intel Hardware takes great advantage of this precision

Intel® Xeon® Scalable Processors

The **Only** Data Center CPU with Built-in AI Acceleration

Intel Advanced Vector Extensions 512
Intel Deep Learning Boost (Intel DL Boost)
Intel Optane Persistent Memory

Shipping

Cascade Lake

New Intel DL Boost (VNNI)
New memory storage hierarchy

Cooper Lake

Intel DL Boost (BFLOAT16)

April 2021

Ice Lake

Intel DL Boost (VNNI) and new
Intel Software Guard Extensions
(Intel® SGX) that enable new
AI use cases like federated learning

2022

Sapphire Rapids

Intel Advanced Matrix Extensions (AMX)
extends built-in AI acceleration
capabilities on Xeon Scalable

Leadership performance

Optimized Deep Learning FW

Intel's oneAPI Ecosystem

Built on Intel's Rich Heritage of CPU Tools Expanded to XPU

oneAPI

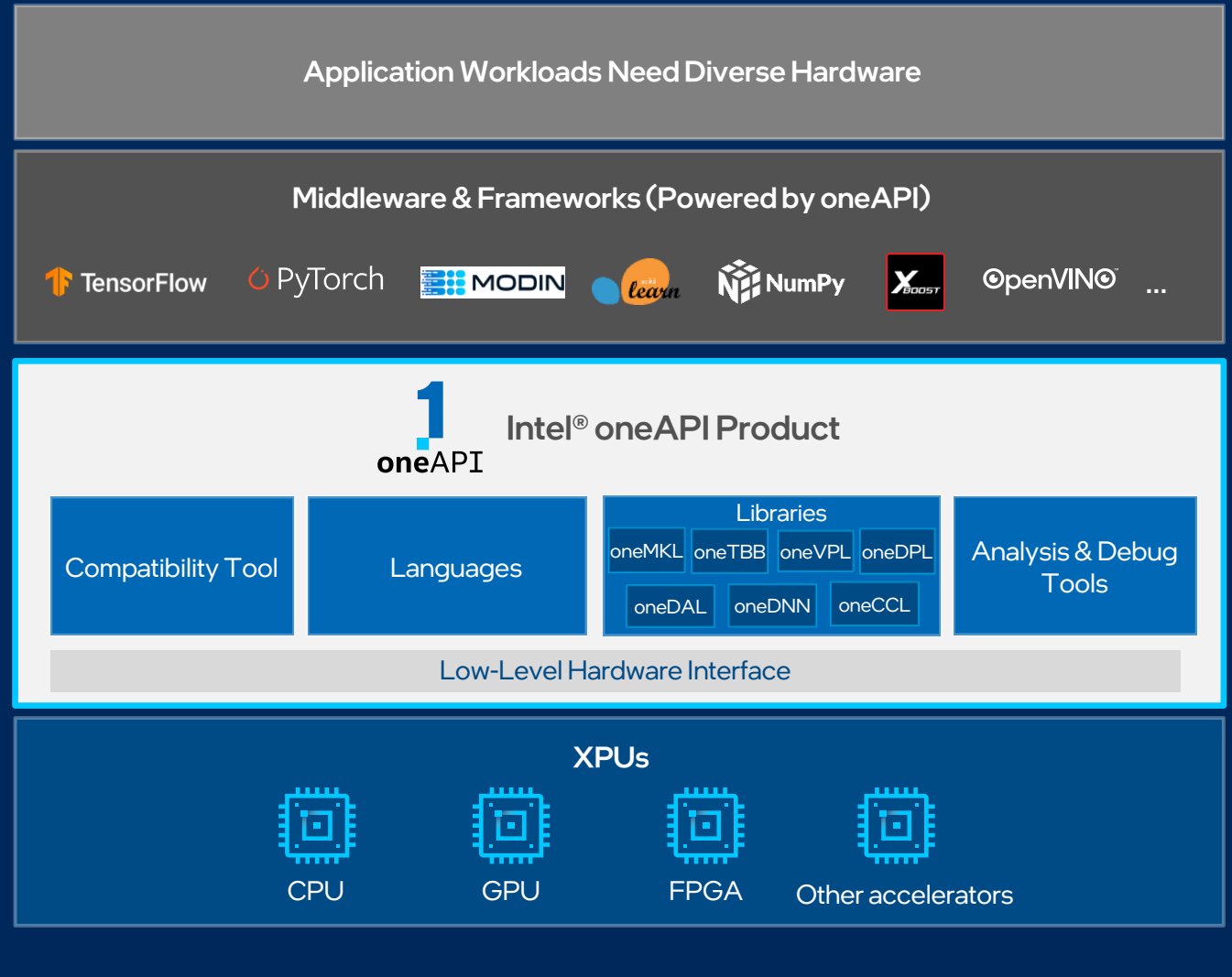
A cross-architecture language based on C++ and SYCL standards

Powerful libraries designed for acceleration of domain-specific functions

A complete set of advanced compilers, libraries, and porting, analysis and debugger tools

Powered by oneAPI

Frameworks and middleware that are built using one or more of the oneAPI industry specification elements, the DPC++ language, and libraries listed on oneapi.com.



[Available Now](#)

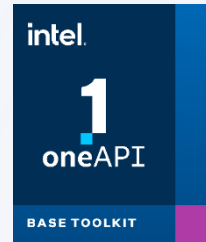
Intel® oneAPI Toolkits

A complete set of proven developer tools expanded from CPU to XPU



Intel® oneAPI Base Toolkit

Native Code Developers



A core set of high-performance tools for building C++, Data Parallel C++ applications & oneAPI library-based applications

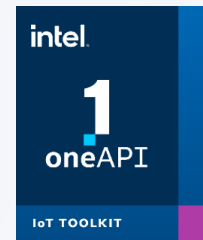
Add-on Domain-specific Toolkits

Specialized Workloads



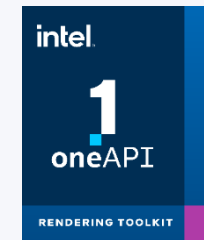
Intel® oneAPI Tools for HPC

Deliver fast Fortran, OpenMP & MPI applications that scale



Intel® oneAPI Tools for IoT

Build efficient, reliable solutions that run at network's edge



Intel® oneAPI Rendering Toolkit

Create performant, high-fidelity visualization applications

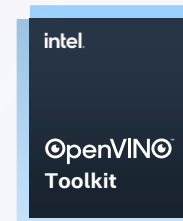
Toolkits powered by oneAPI

Data Scientists & AI Developers



Intel® AI Analytics Toolkit

Accelerate machine learning & data science pipelines with optimized DL frameworks & high-performing Python libraries



Intel® Distribution of OpenVINO™ Toolkit

Deploy high performance inference & applications from edge to cloud

Intel® AI Analytics Toolkit

Powered by oneAPI

Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

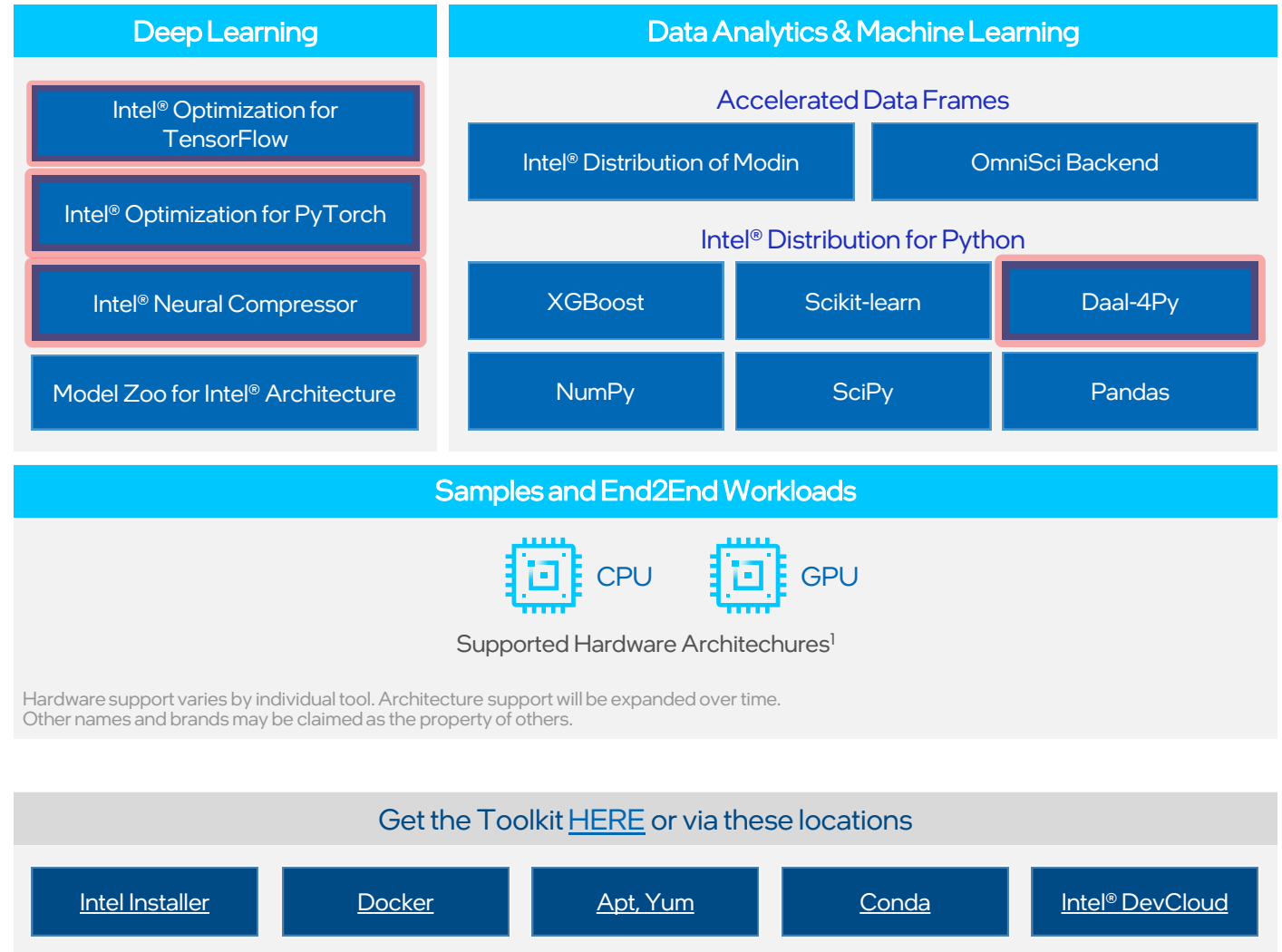
Who Uses It?

Data scientists, AI researchers, ML and DL developers, AI application developers

Top Features/Benefits

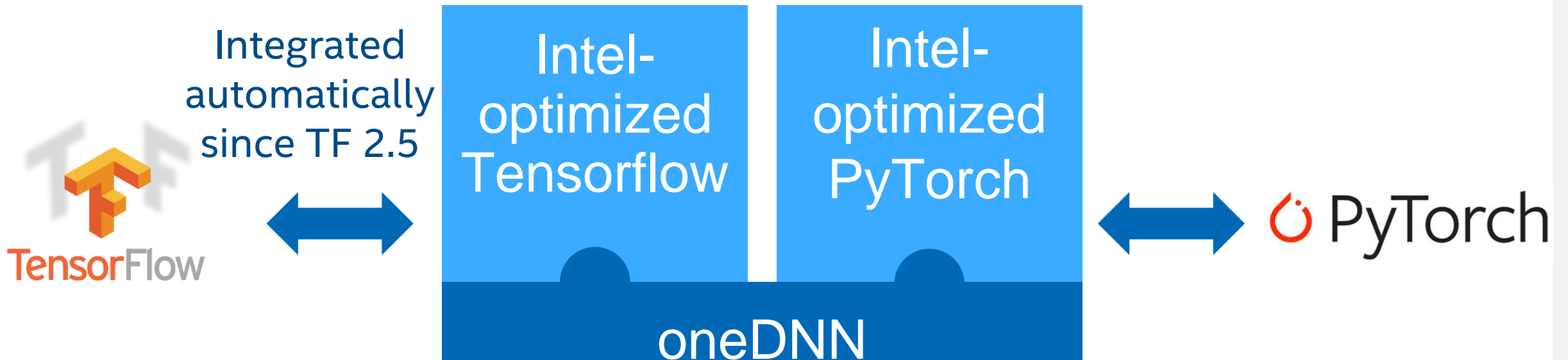
- Deep learning performance for training and inference with Intel optimized DL frameworks and tools
- Drop-in acceleration for data analytics and machine learning workflows with compute-intensive Python packages

Learn More: software.intel.com/oneapi/ai-kit



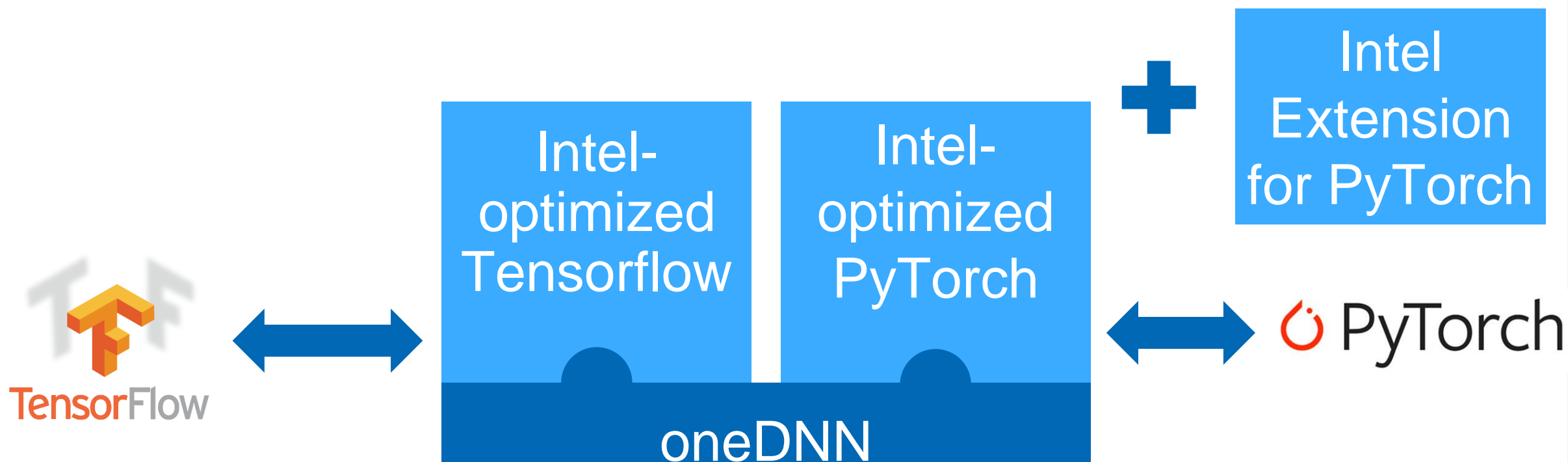
Intel-optimized Deep Learning Frameworks

- Intel-optimized DL frameworks are drop-in replacement,
 - No front code change for the user
- Optimizations are upstreamed automatically (TF) or on a regular basis (PyTorch) to stock frameworks
 - TF: Optimizations are integrated automatically since TF 2.5 and are activated after setting up `TF_ENABLE_ONEDNN_OPTS=1`



Intel-optimized Deep Learning Frameworks

- Intel Extension for PyTorch is an additional module for functions not supported in standard PyTorch (such as mixed precision and dGPU support)
- As they offer more aggressive optimizations, they offer bigger speed-up for training and inference



Intel[®] oneAPI Deep Neural Network Library (oneDNN)

Develop Fast Neural Networks on Intel[®] CPUs & GPUs with Performance-optimized Building Blocks

Intel® oneAPI Deep Neural Network Library (oneDNN)

An **open-source cross-platform** performance library for deep learning applications

- Helps developers create high performance deep learning frameworks
- Abstracts out instruction set and other complexities of performance optimizations
- **Same API for both Intel CPUs and GPUs, use the best technology for the job**
- Supports Linux, Windows and macOS
- Open source for community contributions

More information as well as sources:

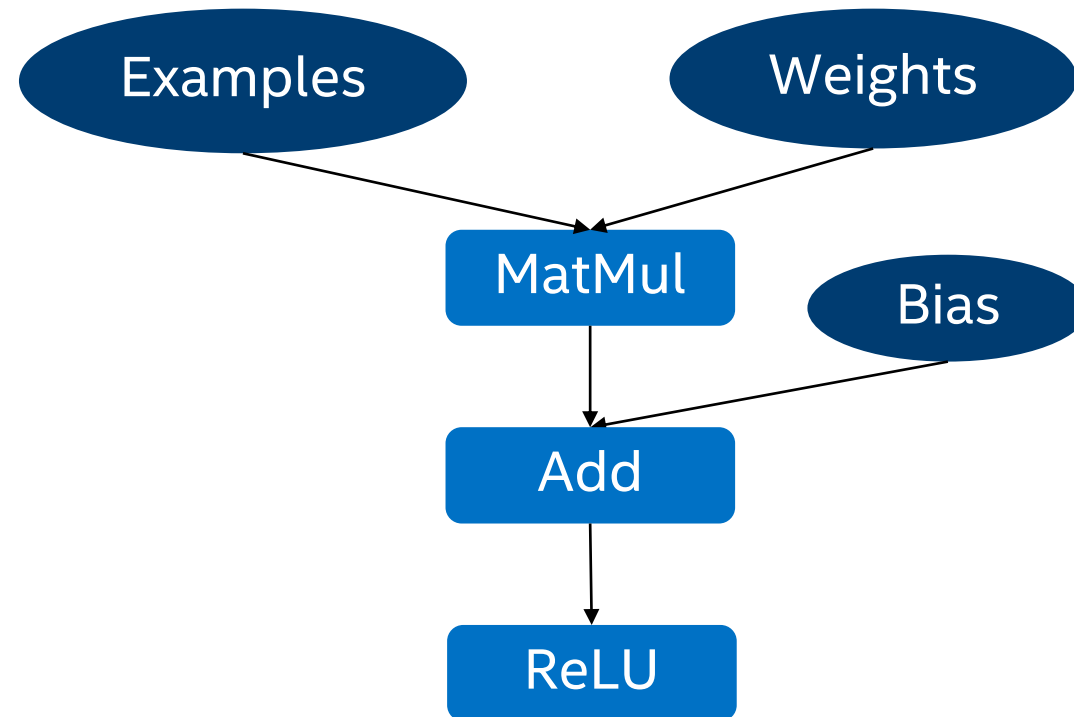
<https://github.com/oneapi-src/oneDNN>

oneDNN

1. Operator optimizations: Replace default kernels by highly-optimized kernels (using Intel® oneDNN)
2. Memory Layout optimization: set optimal layout for each kernel, while minimizing memory changes in between kernels
3. Graph optimizations: Fusion, Layout Propagation

Operator optimizations

In TensorFlow, computation graph is a data-flow graph.



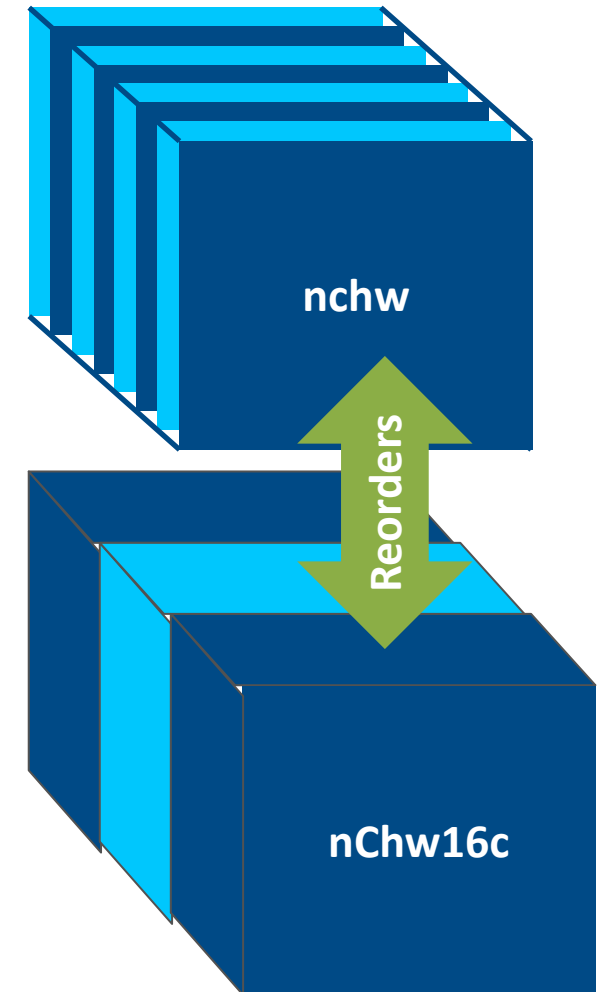
Operator optimizations

- Replace default kernels by highly-optimized kernels (using Intel® oneDNN)
- Adapt to available instruction sets (AVX-512, AVX2, VNNI)
- Adapt to required precision:
 - Training: FP32, BF16
 - Inference: FP32, BF16, FP16, and INT8

	Intel® oneDNN
Convolution	2D/3D Direct Convolution/Deconvolution, Depthwise separable convolution 2D Winograd convolution
Inner Product	2D/3D Inner Production
Pooling	2D/3D Maximum 2D/3D Average (include/exclude padding)
Normalization	2D/3D LRN across/within channel, 2D/3D Batch normalization
Eltwise (Loss/activation)	ReLU(bounded/soft), ELU, Tanh; Softmax, Logistic, linear; square, sqrt, abs, exp, gelu, swish
Data manipulation	Reorder, sum, concat, View
RNN cell	RNN cell, LSTM cell, GRU cell
Fused primitive	Conv+ReLU+sum, BatchNorm+ReLU
Data type	f32, bfloat16, s8, u8

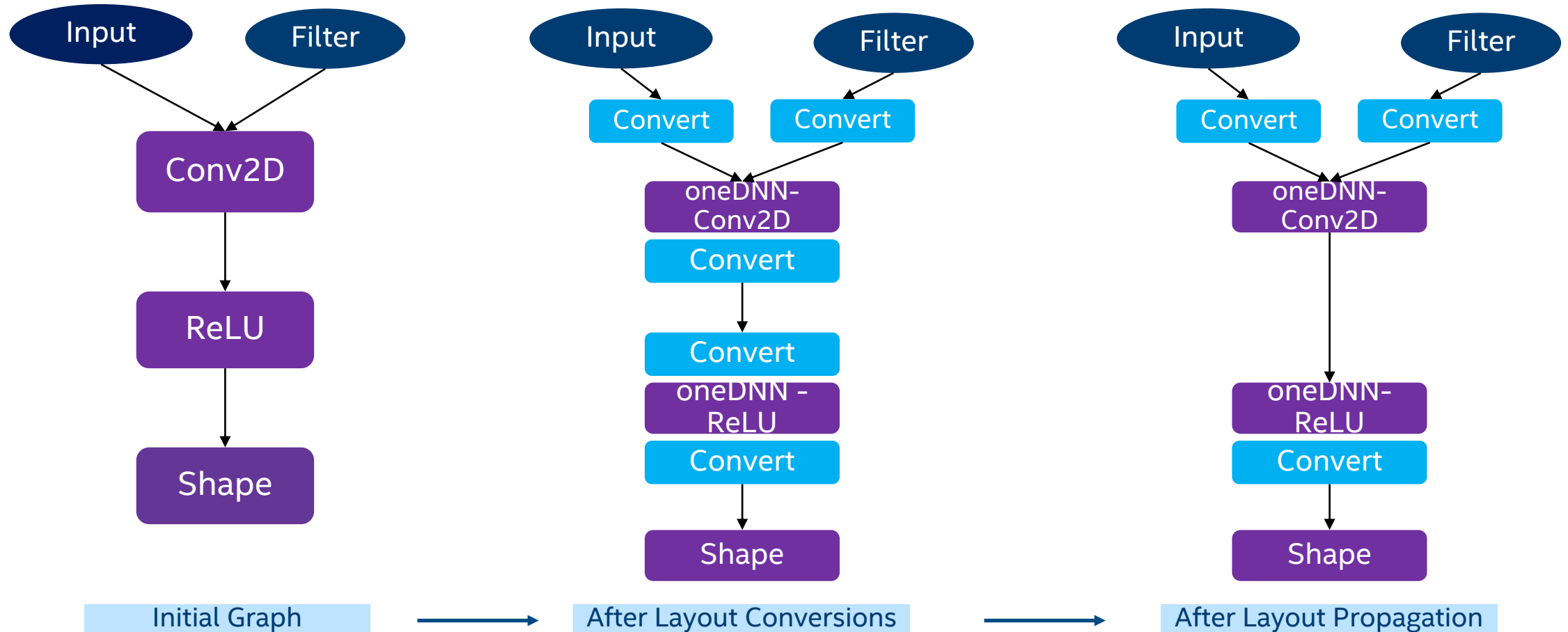
More on memory channels: Memory layouts

- Most popular memory layouts for image recognition are **NHWC** and **NCHW**
 - Challenging for Intel processors both for vectorization or for memory accesses
- Intel oneDNN convolutions use blocked layouts
 - Most popular oneDNN data format is **nChw16c** on AVX512+ systems and **nChw8c** on SSE4.1+ systems



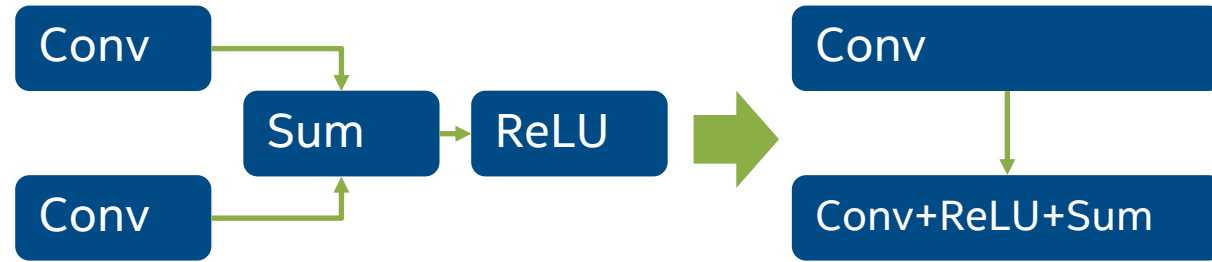
More details: https://oneapi-src.github.io/oneDNN/understanding_memory_formats.html

Graph optimizations: layout propagation



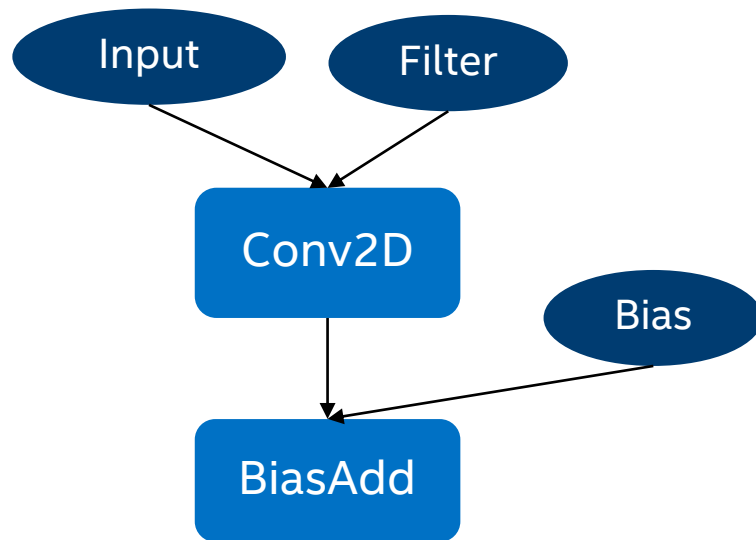
oneDNN track memory layouts and perform reorders only when necessary

Fusing computations

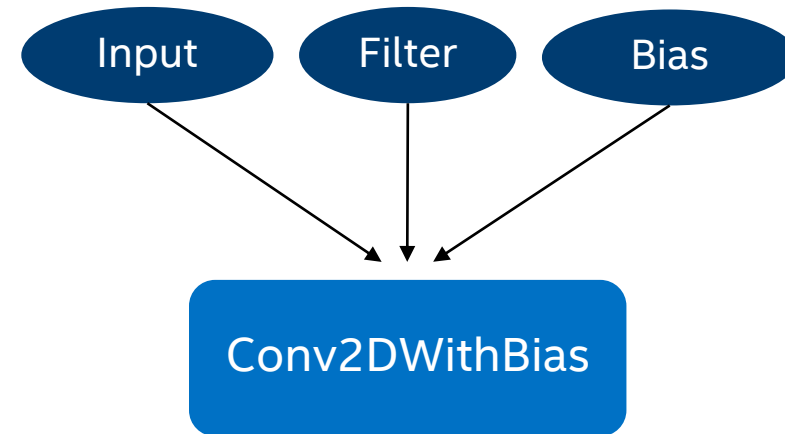


- On Intel processors a high percentage of time is typically spent in bandwidth-limited ops such activation functions
 - ~40% of ResNet-50, even higher for inference
- The solution is to fuse BW-limited ops with convolutions or one with another to reduce the number of memory accesses
 - We fuse patterns: Conv+ReLU+Sum, BatchNorm+ReLU, etc...

Graph optimizations: fusion



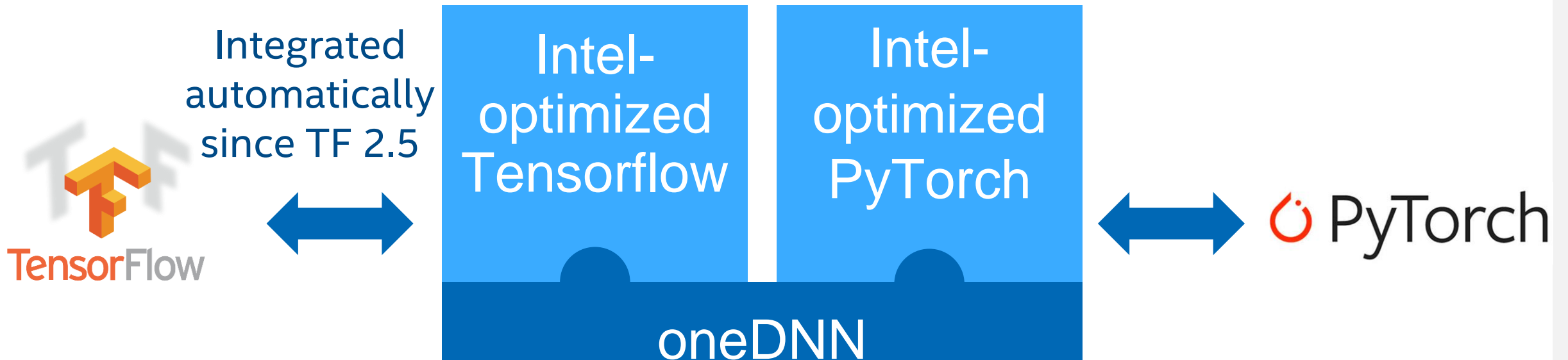
Before Merge



After Merge

Intel-optimized Deep Learning Frameworks

- Intel-optimized DL frameworks are drop-in replacement,
 - No front code change for the user
- Optimizations are upstreamed automatically (TF) or on a regular basis (PyTorch) to stock frameworks
 - TF: Optimizations are integrated automatically since TF 2.5 and are activated after setting up `TF_ENABLE_ONEDNN_OPTS=1`



How to get the optimized frameworks

- In the Intel AI Analytics toolkit

No need to call the flag for Tensorflow



- Through the framework pip/conda wheel:

PyTorch Build	Stable (1.11.0)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 10.2	CUDA 11.3	ROCm 4.5.2 (beta)	CPU
Run this Command:	<code>pip3 install torch torchvision torchaudio</code>			

TensorFlow > Install

Was this helpful?

Install TensorFlow with pip

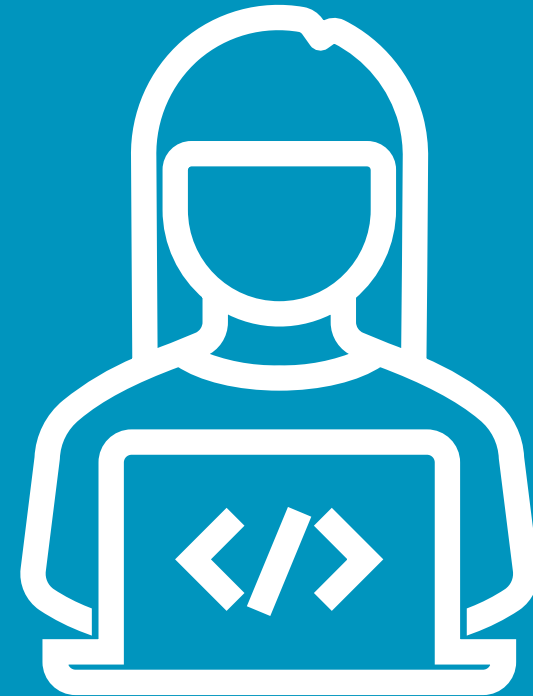
TensorFlow 2 packages are available

- `tensorflow` –Latest stable release with CPU and GPU support (Ubuntu and Windows)
- `tf-nightly` –Preview build (unstable). Ubuntu and Windows include GPU support.

Flag `TF_ENABLE_ONEDNN_OPTS=1` required

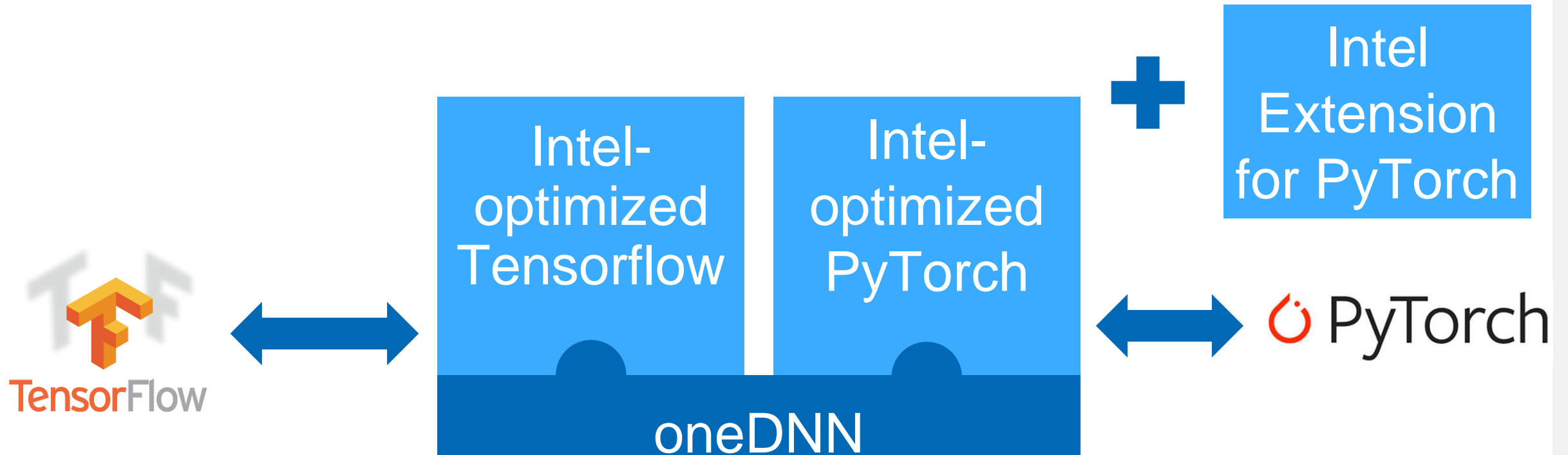


Benchmark demo with Tensorflow



Intel-optimized Deep Learning Frameworks

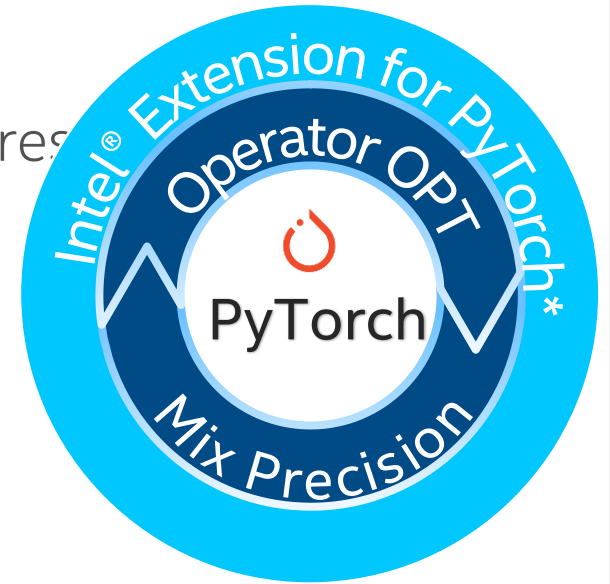
- Intel Extension for PyTorch is an additional module for functions not supported in standard PyTorch (such as mixed precision and dGPU support)
- As they offer more aggressive optimizations, they offer bigger speed-up for training and inference



Intel[®] Extension for PyTorch (IPEX)

Intel® Extension for PyTorch* (IPEX)

- Buffer the PRs for stock Pytorch
- Provide users with the up-to-date Intel software/hardware features
- Streamline the work to integrate oneDNN
- Unify user experiences on Intel CPU and GPU



Operator Optimization

- Customized operators
- Auto graph optimization

Mix Precision

- Accelerate PyTorch operator by LP
- Simplify the data type conversion

Optimal Optimizer

- Split Optimizer (e.g., split-sgd)
- Fused Optimizer

Ease-of-Use User-Facing API (v1.10.x~)

For Float32

```
import torch
import torchvision.models as models

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

model = model.to(memory_format=torch.channels_last)
data = data.to(memory_format=torch.channels_last)

##### code changes #####
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model)
#####

with torch.no_grad():
    model(data)
```



Ease-of-Use User-Facing API (v1.10.x~)

For BFloat16

```
import torch
import torchvision.models as models

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

model = model.to(memory_format=torch.channels_last)
data = data.to(memory_format=torch.channels_last)

##### code changes #####
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)
#####

with torch.no_grad():
    with torch.cpu.amp.autocast():
        model(data)
```

Usage

```
import torch
import intel_pytorch_extension

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv2d = torch.nn.Conv2d(3, 5, 5)

    def forward(self, input):
        res = self.conv2d(input)
        return res

input = torch.randn(5, 3, 9, 9)
model = Model()
model = model.to('xpu')
input = input.to('xpu')
res = model(input)
```

Prior to v1.10

```
import torch
import intel_extension_for_pytorch as ipex

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv2d = torch.nn.Conv2d(3, 5, 5)

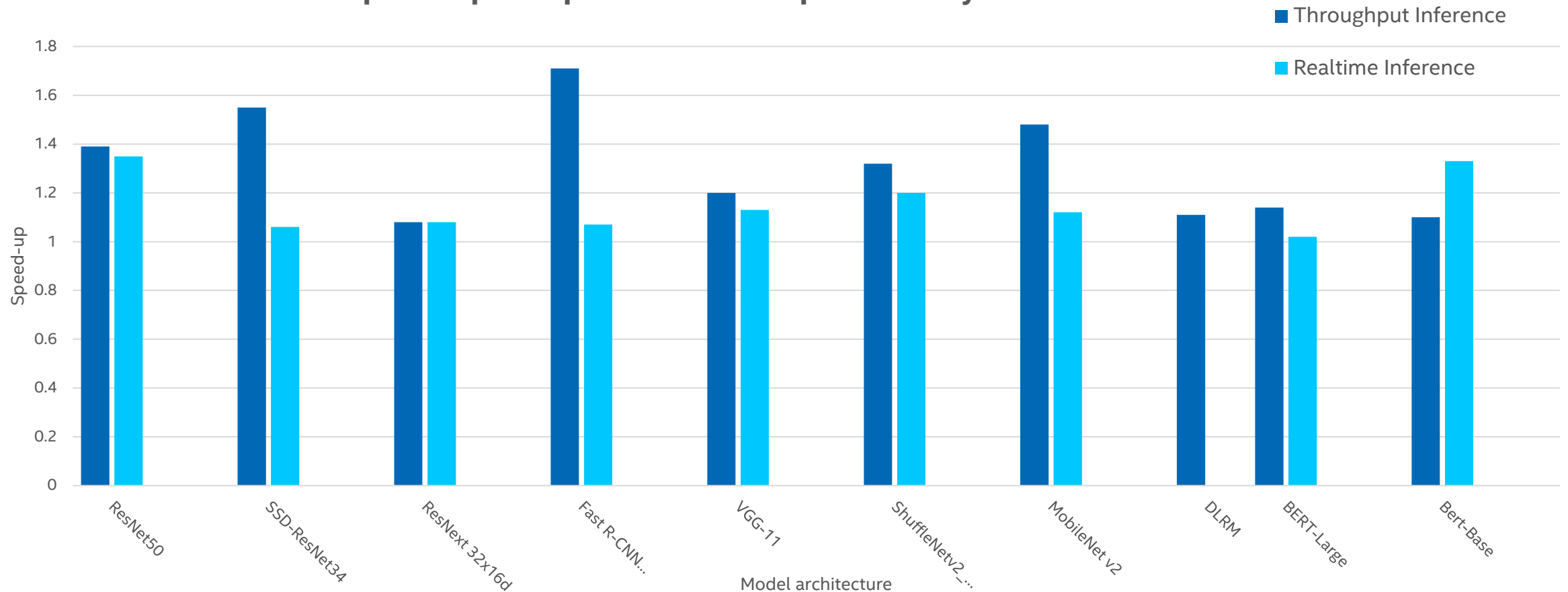
    def forward(self, input):
        res = self.conv2d(input)
        return res

input = torch.randn(5, 3, 9, 9)
model = Model()
model = ipex.optimize(model, dtype=torch.float32, level="O1")
input = input.to(memory_format=torch.channels_last)
res = model(input)
```

v1.10

Intel Extension for PyTorch benchmark

Speed-up compared to Intel-optimized PyTorch for Float32



How to get the Intel Extension for PyTorch

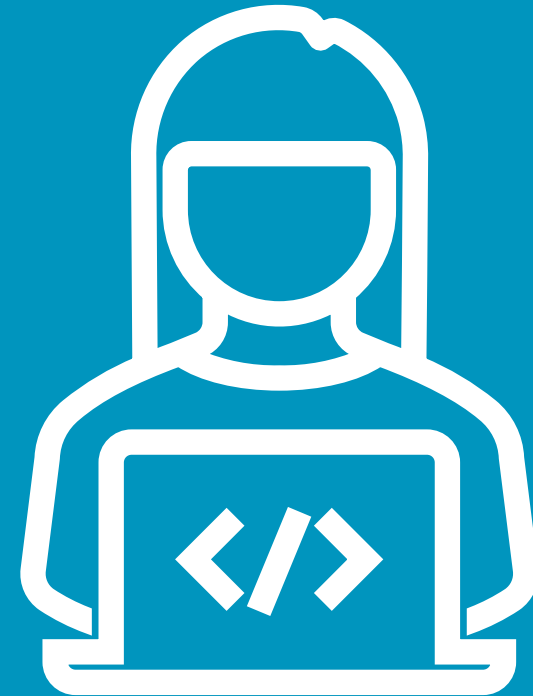
- In the Intel AI Analytics toolkit (API v1.8)
- Through the pip wheel (API v1.11 matching latest PyTorch 1.11)



```
python -m pip install  
intel_extension_for_pytorch
```



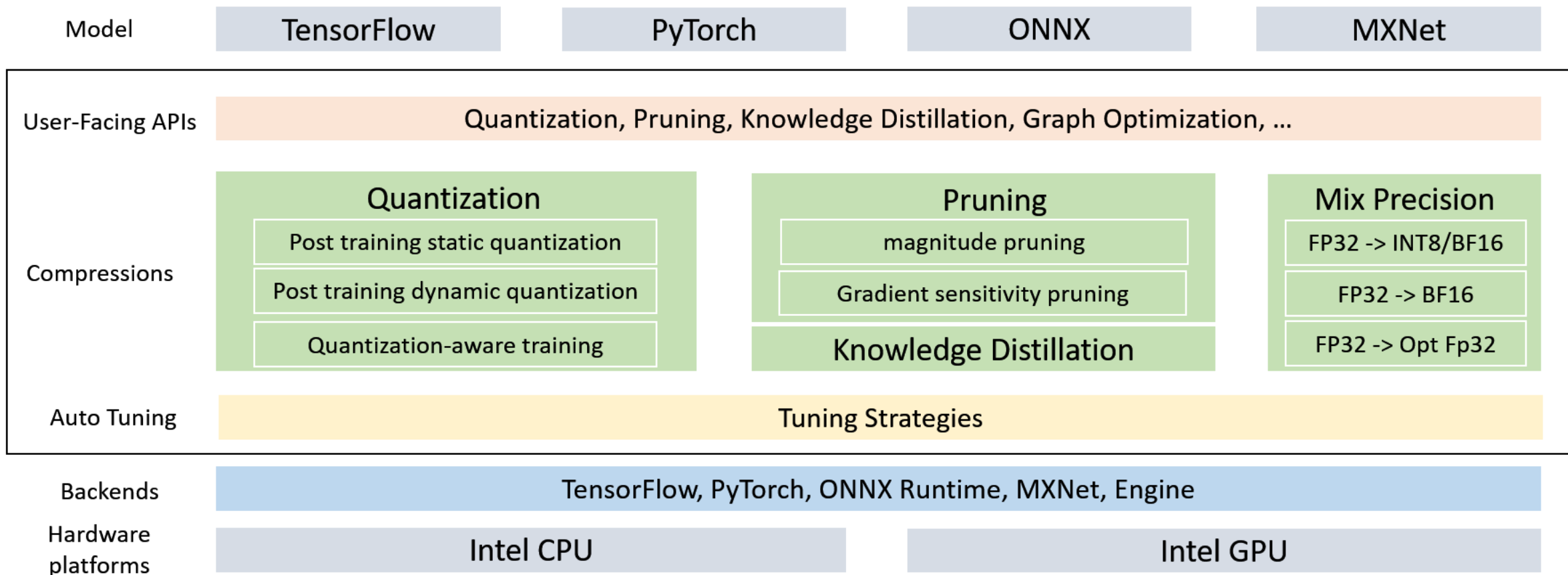
Demo with Intel Extension for PyTorch



Intel Neural Compressor

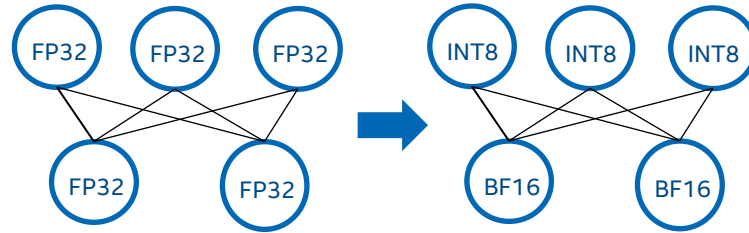
INC: Intel Neural Compressor

- Intel Neural Compressor is an open-source Python library to create low-precision inference solutions on popular deep-learning frameworks
- It supports quantization to BF16/INT8, pruning, knowledge distillation and graph optimizations

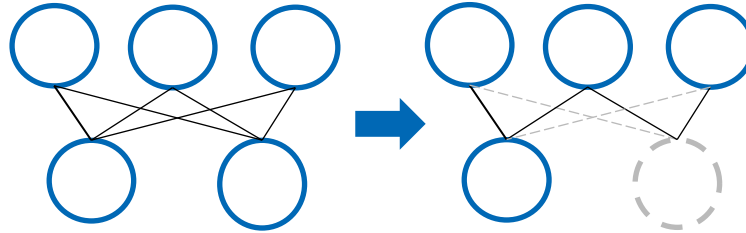


Deep Learning Inference Optimization

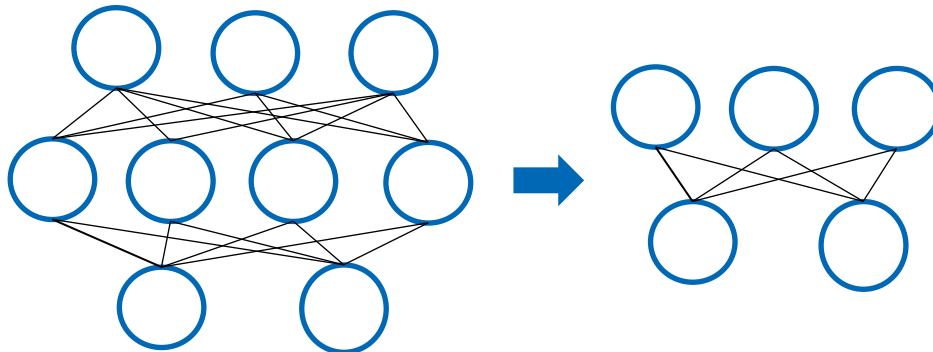
Quantization



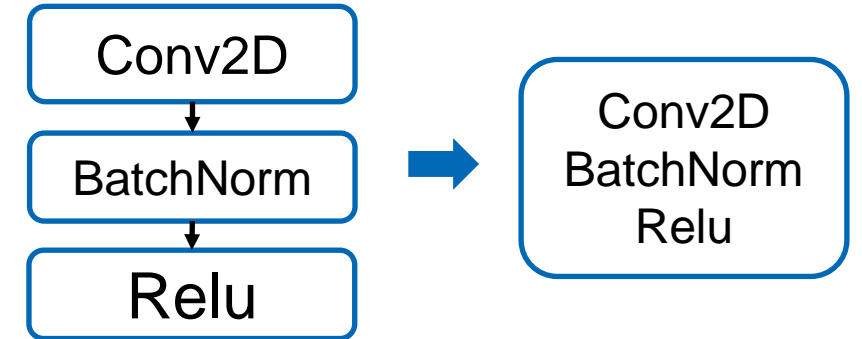
Pruning



Knowledge distillation



Graph Optimization

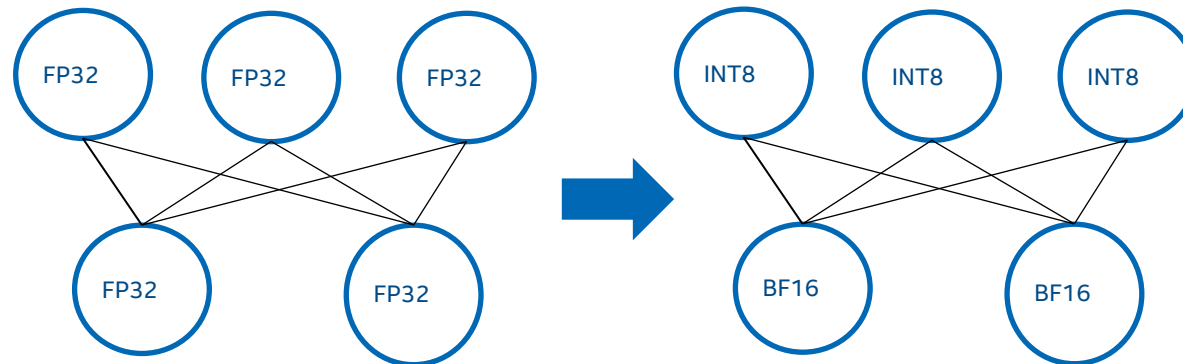


Intel® Neural Compressor

[Intel® Neural Compressor](https://github.com/intel/neural-compressor) (<https://github.com/intel/neural-compressor>) is designed to use automatic accuracy-driven tuning strategies to help user **easily & quickly** find out the best optimization methods above.

Quantization

- Quantization is to convert all/some weights, biases and activations in the neural network from their FP32 value to INT8 or BF16



Quantization

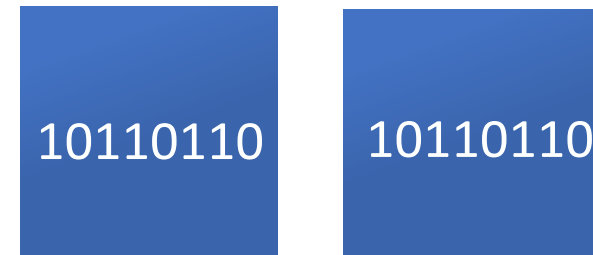
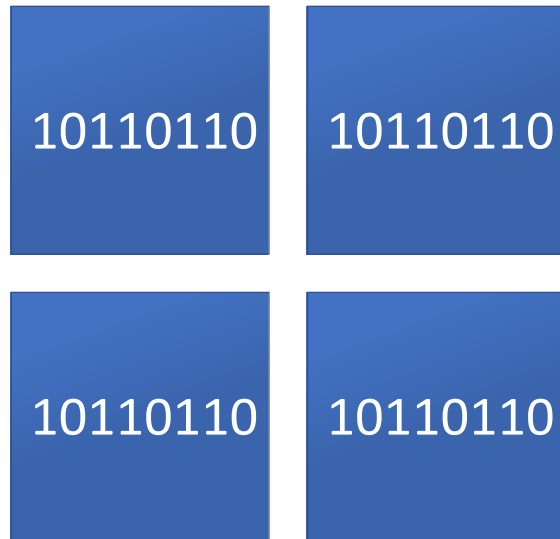
FP32
96.1924
32 bit



INT8
96
8 bit

or

BF16
96.19
16 bit



Quantization techniques possible

- **Post-training static quantization** adds prior to the quantization, an inference stage with a batch of data, that computes the resulting distributions of the different activations. The distribution range is then used for quantization by segmenting it into 256 bins.
- **Post-training dynamic quantization** follows similar principle, but the activation range is calculated for every new batch of data.

Quantization techniques possible

- In **Quantization-Aware Training (QAT)**, all weights and activations are “fake quantized” during both the training forward and backward passes
 - FP32 values are rounded to mimic INT8 values, but all computations are still done with floating point numbers.
 - Weight training is made while “aware” of the model will be quantized in the end
 - Leads to higher accuracy than previous 2 quantization techniques

Pruning

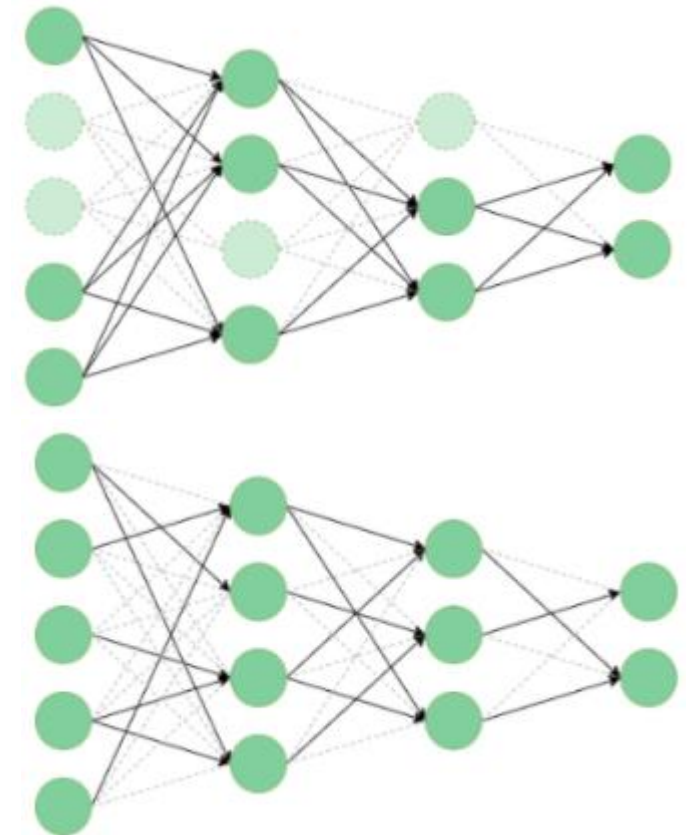
- Network pruning is one of popular approaches of network compression, which reduces the size of a network by removing parameters with minimal drop in accuracy.

- **Structured Pruning**

- Pruning defined patterns such as filters or layers

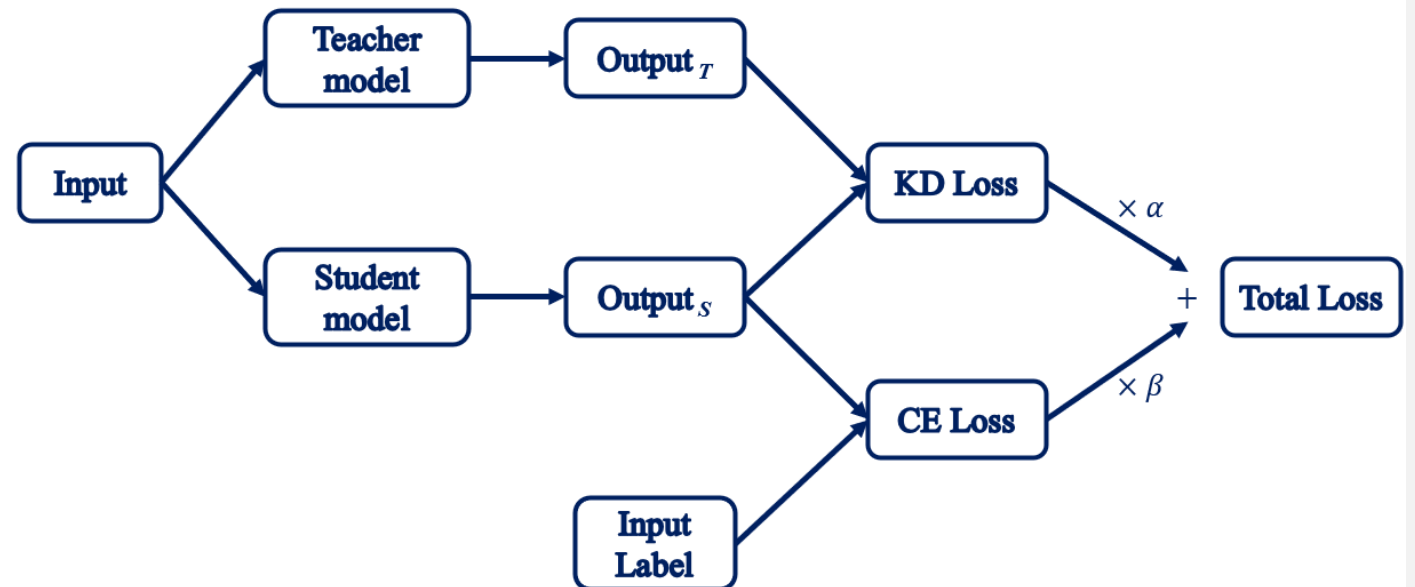
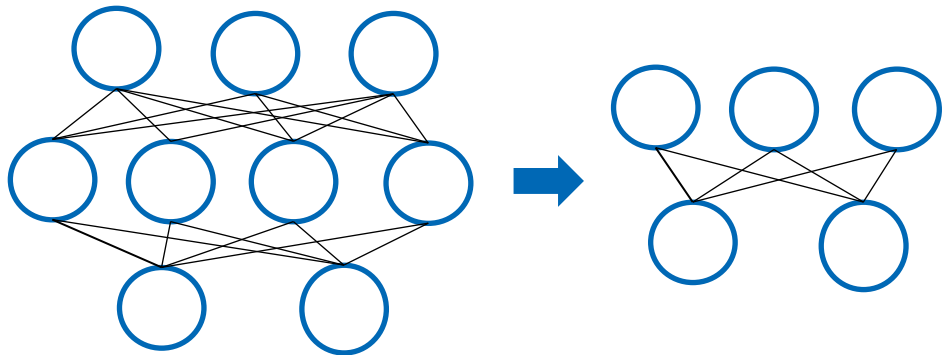
- **Unstructured Pruning**

- Pruning connection between layers on a random basis



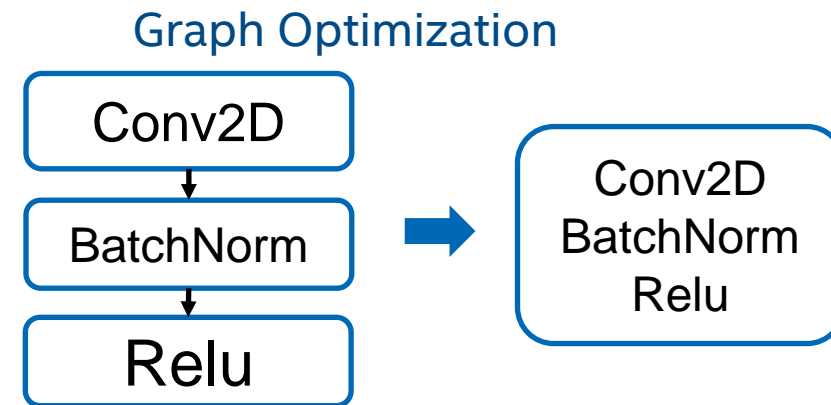
Knowledge Distillation

- Knowledge distillation is one of popular approaches of network compression
- Transfers knowledge from a large model to a smaller one without loss of validity.



Graph optimizations

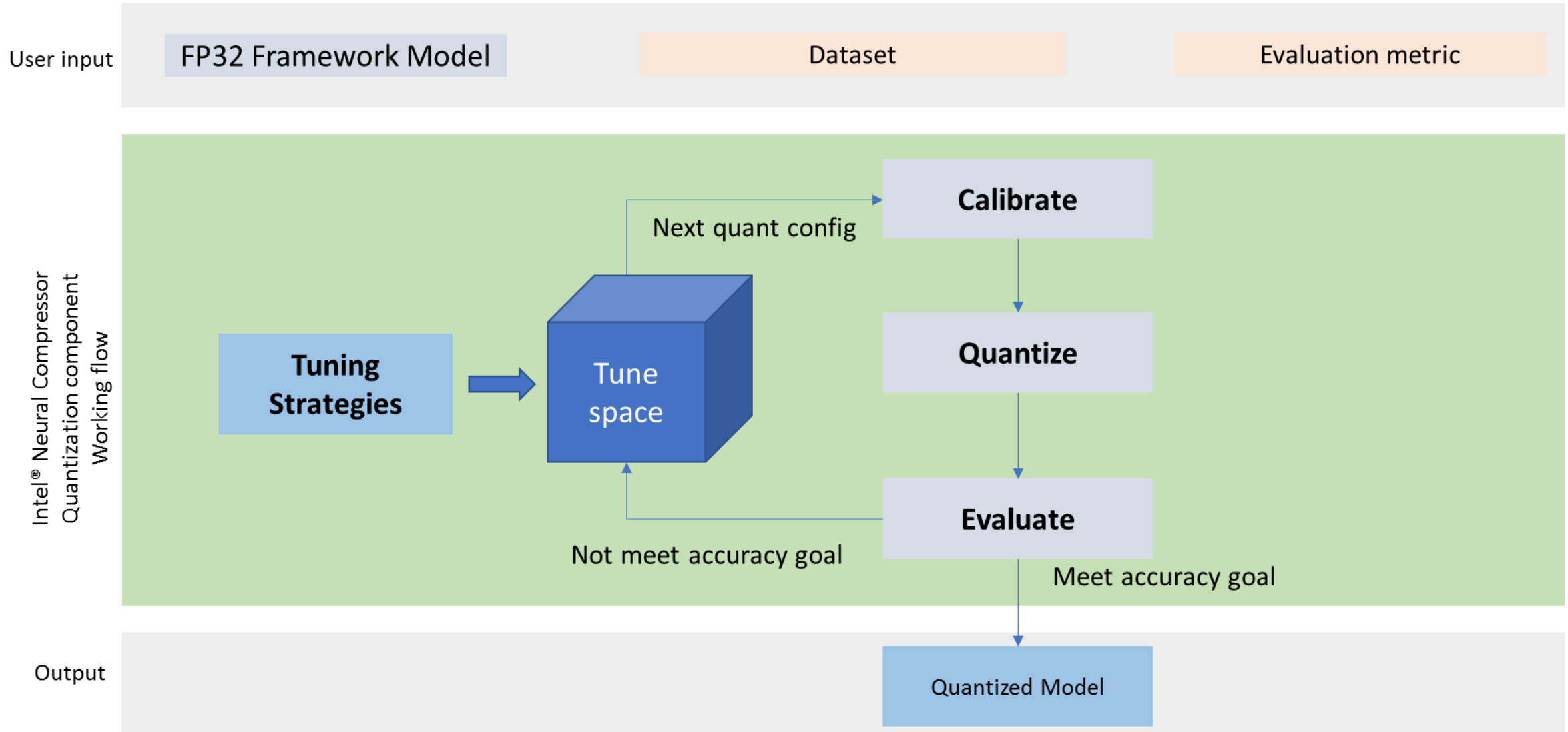
- Optimize for inference the model
- The inference prior allows to simplify and fuse certain layers together



How to quantize with INC?

- Quantization is an iterative process. There are multiple strategy that to define the best next iteration should be.
- **Tuning strategies:**
 - Basic: tries to quantize entire model and then, fallback to re-convert to FP32 certain layers if accuracy not met
 - Bayesian: use Bayesian optimization to define best next iteration
 - Random

Intel® Neural Compressor Quantization Flow



Quantization process

- All parameters such as Framework of origin, accuracy metric, tuning strategy are defined inside a yaml file.
- This is fed to INC API alongside the model and dataloaders

Quantization process

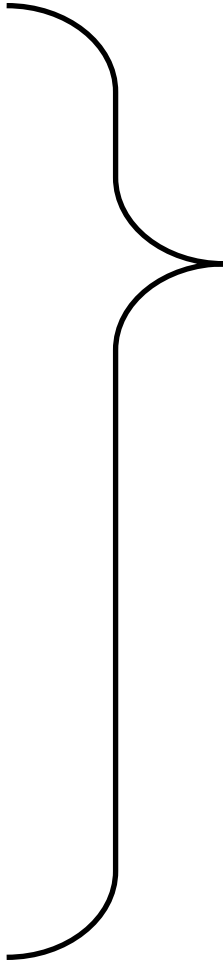
alexnet.yaml

```
version: 1.0

model:
  name: hello_world
  framework: tensorflow
  tensorflow, mxnet and pytorch
  inputs: x
  outputs: Identity


evaluation:
  accuracy:
    metric:
      topk: 1

tuning:
  accuracy_criterion:
    relative: 0.01
  accuracy loss percentage: 1%
  exit_policy:
    timeout: 0
  random_seed: 100
```

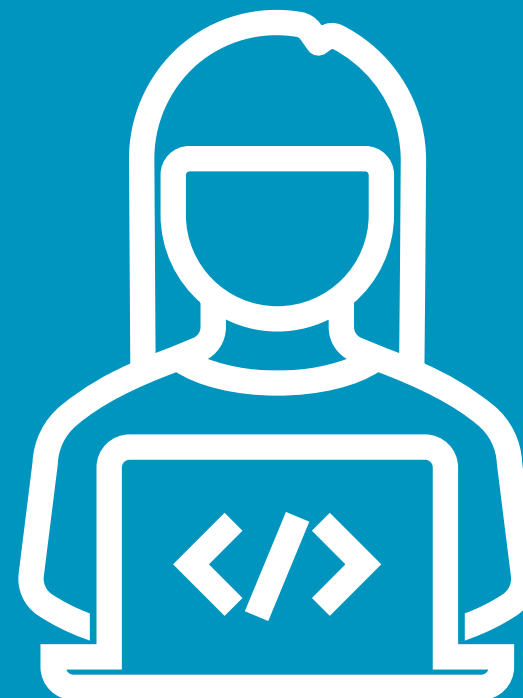


```
fp32_graph = alexnet.load_pb(input_graph_path)|
dataloader = Dataloader(batch_size)

quantizer = inc.experimental.Quantization(yaml_config)
quantizer.model = fp32_graph
quantizer.calib_dataloader = dataloader
quantizer.eval_dataloader = dataloader
q_model = quantizer.fit()
```



INC demo:
Quantize an
AlexNet trained
with Keras on
MNIST to INT8

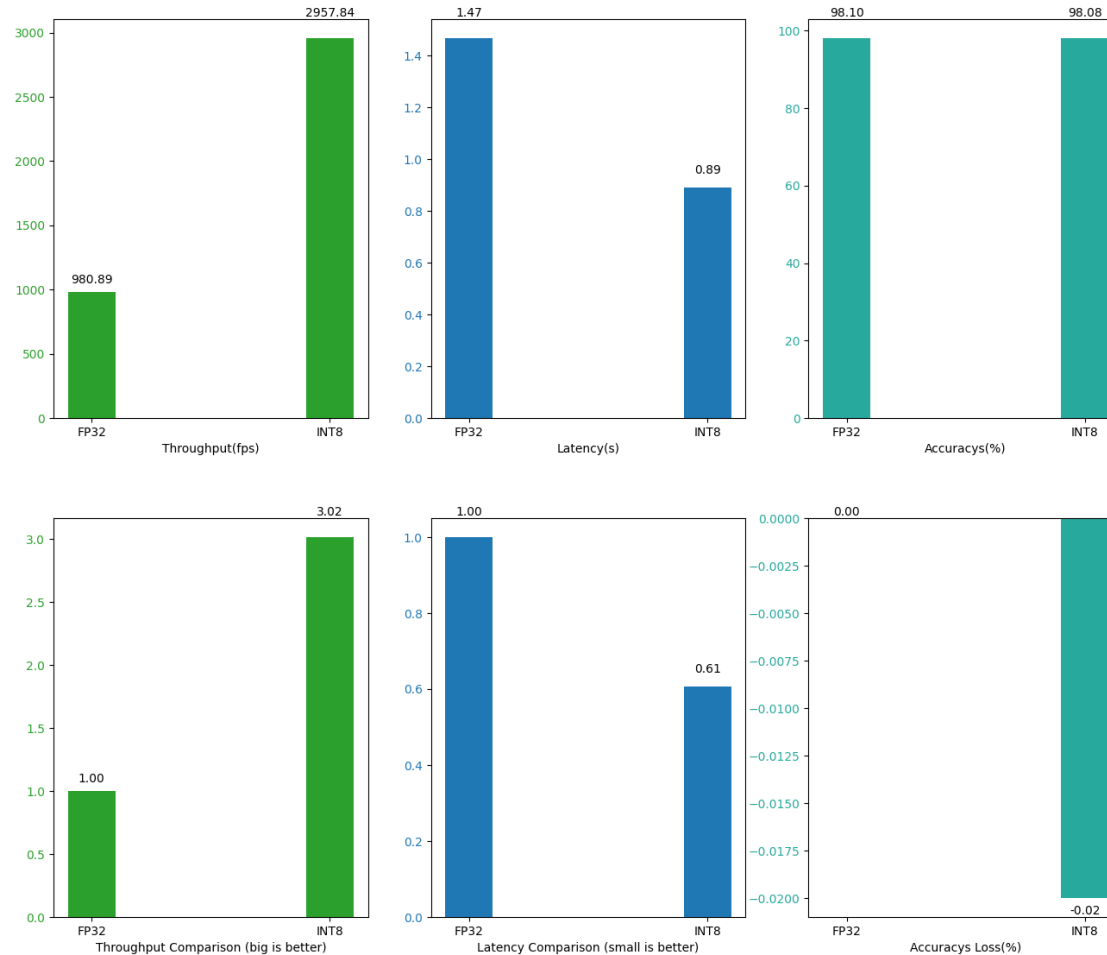


Demo

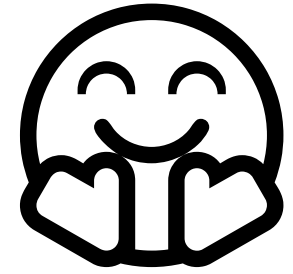
1. Quantize the frozen .pb model file to INT8 model using INC
2. Test & Compare the performance of FP32 and INT8 model by same script.

Check Result

Performance data



INC: Intel Neural Compressor



- Hugging Face 🤗 is an NLP-focused startup, which developed the very popular Transformers library that exposes state-of-art transformer architectures to end-users
- They released in 2021 Optimum, a toolkit for quantize Transformers based on Intel Neural Compressor

```
from optimum.intel.neural_compressor.quantization import IncQuantizerForSequenceClassification

# Create quantizer from config
quantizer = IncQuantizerForSequenceClassification.from_config(
    "echarlaix/bert-base-dynamic-quant-test",
    config_name="quantization.yml",
    eval_func=eval_func,
)

# Apply dynamic quantization
model = quantizer.fit_dynamic()
```

intel®

Notices and Disclaimers

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Optimization Notice

¹ Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.

² Software and workloads used in performance tests may have been optimized for performance only on microprocessors from Intel. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. Consult other information and performance tests while evaluating potential purchases, including performance when combined with other products. For more information, see Performance Benchmark Test Disclosure. Source: Intel measurements, as of June 2017.

Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at www.intel.com.

Intel, the Intel logo, Xeon™, Arria™ and Movidius™ are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© Intel Corporation.

Slide Reference	1	2	3
System Board	Intel® Server S2600 (Dual socket)	Supermicro / X11SPL-F	Supermicro / X11SPL-F
Product	Xeon Silver 4216	Intel(R) Xeon(R) Silver 4112	Intel(R) Xeon(R) Silver 4112
CPU sockets	2	-	1
Physical cores	2 x 16	4	4
Processor Base Frequency	2.10 GHz	2.60GHz	2.60GHz
HyperThreading	enabled	-	enabled
Turbo	On	-	On
Power-Performance Mode	Performance Mode	-	-
Total System Memory size	12 x 64GB	16384	16384
Memory speed	2400MHz	2400MHz	2400MHz
Software OS	Ubuntu 18.04	Ubuntu 16.04.3 LTS	Ubuntu 16.04.6 LTS
Software Kernel	4.15.0-66-generic x86_64	4.13.0-36-generic	4.15.0-29-generic
Test Date	27 September 2019	25 May 2018	18 April 2019
Precision (IntMode)	Int 8 (Throughput Mode)	FP32	Int 8 (Throughput Mode)
Power (TDP)	200W	85W	85W
Price Link on 30 Sep 2019 (Prices may vary)	\$2,024	\$483	\$483
Network	Mobilenet SSD	Mobilenet SSD	Mobilenet SSD

Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at www.intel.com.

Intel, the Intel logo, Xeon™, Arria™ and Movidius™ are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© Intel Corporation.

System Board	Intel prototype, TGL U DDR4 SODIMM RVP	ASUSTeK COMPUTER INC. / PRIME Z370-A
CPU	11 th Gen Intel® Core™ -5-1145G7E @ 2.6 GHz.	8 th Gen Intel® Core™ i5-8500T @ 3.0 GHz.
Sockets / Physical cores	1 / 4	1 / 6
HyperThreading / Turbo Setting	Enabled / On	Na / On
Memory	2 x 8198 MB 3200 MT/s DDR4	2 x 16384 MB 2667 MT/s DDR4
OS	Ubuntu* 18.04 LTS	Ubuntu* 18.04 LTS
Kernel	5.8.0-050800-generic	5.3.0-24-generic
Software	Intel® Distribution of OpenVINO™ toolkit 2021.1.075	Intel® Distribution of OpenVINO™ toolkit 2021.1.075
BIOS	Intel TGLIFUI1.R00.3243.A04.2006302148	AMI, version 2401
BIOS release date	Release Date: 06/30/2021	7/12/2019
BIOS Setting	Load default settings	Load default settings, set XMP to 2667
Test Date	9/9/2021	9/9/2021
Precision and Batch Size	CPU: INT8, GPU: FP16-INT8, batch size: 1	CPU: INT8, GPU: FP16-INT8, batch size: 1
Number of Inference Requests	4	6
Number of Execution Streams	4	6
Power (TDP Link)	<u>28 W</u>	<u>35W</u>
Price (USD) Link on Sep 22,2021 Prices may vary	<u>\$309</u>	<u>\$192</u>

- 1): Memory is installed such that all primary memory slots are populated.
- 2): Testing by Intel as of September 9, 2021