Choose the Best Accelerated Technology

Intel Performance optimizations for Deep Learning

Laurent Duhem – HPC/AI Solutions Architect (Laurent.duhem@intel.com)
Shailen Sobhee - AI Software Technical Consultant (shailen.sobhee@intel.com)
January 2021
Quick recap of oneAPI
Overview of oneDNN
Training:
  • Overview of performance-optimized DL frameworks
Inferencing:
  • Intro to Intel Distribution of OpenVINO
  • Intel Low Precision Optimization Tool
Hands-on demos
Intel’s oneAPI Ecosystem

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

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.

Visit [software.intel.com/oneapi](https://software.intel.com/oneapi) for more details.

Some capabilities may differ per architecture and custom-tuning will still be required. Other accelerators to be supported in the future.

Available Now
Intel® oneAPI Toolkits
A complete set of proven developer tools expanded from CPU to XPU

Intel® oneAPI Base Toolkit
Native Code Developers

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

Latest version is 2021.1

IAGS Intel Architecture, Graphics, and Software
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
Enabling Intel Deep Learning deployment toolkit, major open source deep learning networks and customer applications

Intel Deep Neural Network Library (oneDNN) is an open source IA optimized Deep Learning library for scalable, high-velocity integration with ML/DL frameworks.

- Includes open source implementations of new DNN functionality
- Delivers new algorithms ahead of MKL releases
- Open for community contributions

Intel oneMKL is commercial software Performance Library to extract max Intel HW performance and provide a common interface to all Intel processors and accelerators.

*Other names and brands may be claimed as property of others.*
Develop Fast Neural Networks on Intel® CPUs & GPUs
with Performance-optimized Building Blocks

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

- An open source 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
  - Open source for community contributions

- Get it here: software.intel.com/oneapi/dnnl
- Distribution: github.com/intel/mkl-dnn
Intel® oneAPI Deep Neural Network Library

Basic Information

- **Features**
  - API: C, C++, SYCL\(^{(4)}\)
  - Training: float32, bfloat16\(^{(1)}\)
  - Inference: float32, bfloat16\(^{(1)}\), float16\(^{(1)}\), and int8\(^{(1)}\)
  - MLPs, CNNs (1D, 2D and 3D), RNNs (plain, LSTM, GRU)

- **Support Matrix**
  - Compilers: Intel, GCC, CLANG, MSVC\(^{(2)}\), DPC++\(^{(4)}\)
  - OS: Linux, Windows\(^{(2)}\), macOS\(^{(2,3)}\)
  - CPU
    - Hardware: Intel® Atom, Intel® Core™, Intel® Xeon™
    - Runtimes: OpenMP, TBB, DPC++\(^{(4)}\)
  - GPU
    - Runtimes: OpenCL\(^{(2)}\), DPC++\(^{(4)}\)

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

(1) Low precision data types are supported only for platforms where hardware acceleration is available
(2) Not available in the oneAPI Beta binary distribution
(3) GPU support, OpenCL runtime and DPC++ GPU target are not available on macOS
(4) Targeting oneAPI Beta, not available in DNNL v1.x
Overview of Intel-optimizations for TensorFlow*
Intel® TensorFlow* optimizations

1. **Operator optimizations**: Replace default (Eigen) kernels by highly-optimized kernels (using Intel® oneDNN)
2. **Graph optimizations**: Fusion, Layout Propagation
3. **System optimizations**: Threading model
Setting up the TensorFlow environments

- We create two environments for our benchmarks
- One with “stock” TensorFlow
- The other with Intel-optimized TensorFlow
Operator optimizations

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

- Replace default (Eigen) kernels by highly-optimized kernels (using Intel® oneDNN)
- Intel® oneDNN has optimized a set of TensorFlow operations.
- Library is open-source (https://github.com/intel/mkl-dnn) and downloaded automatically when building TensorFlow.

<table>
<thead>
<tr>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv2D</td>
<td>Conv2DGrad</td>
</tr>
<tr>
<td>Relu, TanH, ELU</td>
<td>ReLUGrad, TanHGrad, ELUGrad</td>
</tr>
<tr>
<td>MaxPooling</td>
<td>MaxPoolingGrad</td>
</tr>
<tr>
<td>AvgPooling</td>
<td>AvgPoolingGrad</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>BatchNormGrad</td>
</tr>
<tr>
<td>LRN</td>
<td>LRNGrad</td>
</tr>
<tr>
<td>MatMul, Concat</td>
<td></td>
</tr>
</tbody>
</table>
Fusing computations

- On Intel processors a high % of time is typically spent in BW-limited ops
  - ~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 # of memory accesses
  - Conv+ReLU+Sum, BatchNorm+ReLU, etc
  - Done for inference, WIP for training
- The frameworks are expected to be able to detect fusion opportunities
  - IntelCaffe already supports this
- Major impact on implementation
  - All the implementations must be made aware of the fusion to get max performance
  - Intel oneDNN team is looking for scalable solutions to this problem
Graph optimizations: fusion
All oneDNN operators use highly-optimized layouts for TensorFlow tensors.
More on memory channels: Memory layouts

- Most popular memory layouts for image recognition are **nhwc** and **nchw**
  - Challenging for Intel processors either for vectorization or for memory accesses (cache thrashing)
- Intel oneDNN convolutions use blocked layouts
  - Example: **nhwc** with channels blocked by 16 – **nChw16c**
  - Convolutions define which layouts are to be used by other primitives
  - Optimized frameworks track memory layouts and perform reorders **only** when necessary
Data Layout has a BIG Impact

- Continuous access to avoid gather/scatter
- Have iterations in inner most loop to ensure high vector utilization
- Maximize data reuse; e.g. weights in a convolution layer
- Overhead of layout conversion is sometimes negligible, compared with operating on unoptimized layout

\[
\begin{align*}
\text{for } i &= 1 \text{ to } N \ # \text{ batch size} \\
\text{for } j &= 1 \text{ to } C \ # \text{ number of channels}, \text{ image RGB = 3 channels} \\
\text{for } k &= 1 \text{ to } H \ # \text{ height} \\
\text{for } l &= 1 \text{ to } W \ # \text{ width} \\
\text{dot_product}( \ldots )
\end{align*}
\]
System optimizations: load balancing

- TensorFlow graphs offer opportunities for parallel execution.
- Threading model
  1. `inter_op_parallelism_threads` = max number of operators that can be executed in parallel
  2. `intra_op_parallelism_threads` = max number of threads to use for executing an operator
  3. `OMP_NUM_THREADS` = MKL-DNN equivalent of `intra_op_parallelism_threads`
tf.ConfigProto is used to set the inter_op_parallelism_threads and intra_op_parallelism_threads configurations of the Session object.

```python
>>> config = tf.ConfigProto()
>>> config.intra_op_parallelism_threads = 56
>>> config.inter_op_parallelism_threads = 2
>>> tf.Session(config=config)
```

https://www.tensorflow.org/performance/performance_guide#tensorflow_with_intel_mkl_dnn
Performance Guide


Example setting MKL variables with python `os.environ`:

```python
os.environ['KMP_BLOCKTIME'] = "1"

os.environ['KMP_AFFINITY'] = "granularity=fine,compact,1,0"

os.environ['KMP_SETTINGS'] = "0"
```

Tuning MKL for the best performance

This section details the different configurations and environment variables that can be used to tune the MKL to get optimal performance. Before tweaking various environment variables make sure the model is using the `NCHW` (channels_first) data format. The MKL is optimized for `NCHW` and Intel is working to get near performance parity when using `NHWC`.

MKL uses the following environment variables to tune performance:

- `KMP_BLOCKTIME` - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.
- `KMP_AFFINITY` - Enables the run-time library to bind threads to physical processing units.
- `KMP_SETTINGS` - Enables (true) or disables (false) the printing of OpenMP* run-time library environment variables during program execution.
- `OMP_NUM_THREADS` - Specifies the number of threads to use.

Demo: Review TF Benchmark results
PyTorch Performance Benefit with Intel® AVX-512 VDPBF16PS Instruction

Inference Performance

<table>
<thead>
<tr>
<th>Scenario</th>
<th>DLRM</th>
<th>ResNet-50</th>
<th>ResNeXt-101 32x4d</th>
</tr>
</thead>
<tbody>
<tr>
<td>samples/s (FP32)</td>
<td>71061</td>
<td>243</td>
<td>120</td>
</tr>
<tr>
<td>samples/s (bfloat16)</td>
<td>99321</td>
<td>399</td>
<td>193</td>
</tr>
<tr>
<td>Speedup Ratio</td>
<td>1.4</td>
<td>1.64</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Intel Optimizations for PyTorch

- Accelerated operators
- Graph optimization
- Accelerated communications
Motivation for Intel Extension for PyTorch (IPEX)

- Provide customers with the up-to-date Intel software/hardware features
- Streamline the work to enable Intel accelerated library

**Operator Optimization**
- Auto dispatch the operators optimized by the extension backend
- Auto operator fusion via PyTorch graph mode

**Mix Precision**
- Accelerate PyTorch operator by bfloat16
- Automatic mixed precision
How to get IPEX

1. oneAPI AI Analytics Toolkit
2. Install from source
IPEX from the oneAPI AI Analytics Toolkit

Intel Optimizations for PyTorch

Intel-Optimized PyTorch
- PyTorch back-end optimizations
- Up-streamed to regular PyTorch
- Same front-end code as regular PyTorch

Intel Extension for PyTorch (IPEX)
- Additional optimizations and Mixed Precision support
- Different front-end

Torch-CCL
- For distributed learning
- PyTorch bindings for oneCCL
Installing IPEX from source

https://github.com/intel/intel-extension-for-pytorch
License - Apache 2.0

**Build and install**

1. Install PyTorch from source
   1. Get PyTorch v1.5.0-rc3 source
2. Download and install Intel PyTorch Extension source
3. Add new backend for Intel Extension for PyTorch

Note: Binary installation is coming soon
Data types

- Benefit of bfloat16
  - Performance 2x up
  - Comparable accuracy loss against fp32
  - No loss scaling, compared to fp16

* bfloat16 intrinsic support starts from 3rd Generation Intel® Xeon® Scalable Processors

Automatic Mixed Precision Feature (FP32 + BF16)

1. import ipex
2. Enable Auto-Mix-Precision by API
3. Convert the input tensors to the extension device
4. Convert the model to the extension device

```python
import torch
import intel_pytorch_extension as ipex
ipex.enable_auto_optimization(mixed_dtype = torch.bfloat16, train = True)

EPOCH = 20
BATCH_SIZE = 128
LR = 0.001

def main():
    train_loader = ...
    test_loader = ...
    net = topology()
    net = net.to(ipex.DEVICE)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(net.parameters(), lr = LR, momentum=0.9)
    for epoch in range(EPOCH):
        net.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            data = data.to(ipex.DEVICE)
            target = target.to(ipex.DEVICE)
            optimizer.zero_grad()
            output = net(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

    net.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data = data.to(ipex.DEVICE)
            target = target.to(ipex.DEVICE)
            output = net(data)
            test_loss += criterion(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
            test_loss /= len(test_loader.dataset)

if __name__ == '__main__':
    main()
```
PyTorch Demo
WRITE once, deploy & scale diversely

TensorFlow
ONNX
mxnet
KALDI
Caffe

Model Optimizer
Inference Engine

CPU
FPGA
Edge
GPU

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

All products, computer systems, dates, and figures are preliminary based on current
From a bird’s eye-view
Advanced capabilities to streamline deep learning deployments

1. Build

Trained Model
- TensorFlow
- Caffe
- Kaldi
- mxnet
- DNNX

Open Model Zoo
100+ open sourced and optimized pre-trained models; 80+ supported public models

2. Optimize

Model Optimizer
Converts and optimizes trained model using a supported framework

3. Deploy

Inference Engine
Common API that abstracts low-level programming for each hardware

- CPU Plugin
- GPU Plugin
- GNA Plugin
- Myriad Plugin
- HDDL Plugin
- FPGA Plugin
- CPU Plugin

Deep Learning Streamer
OpenCV | OpenCL™

Post-Training Optimization Tool

Deep Learning Workbench

Code Samples & Demos (e.g. Benchmark app, Accuracy Checker, Model Downloader)

Deployment Manager

Read, Load, Infer

Intermediate Representation (.xml, .bin)

IR Data
Get Started
Typical workflow from development to deployment

- Train a model
- Find a trained model
- Run the Model Optimizer
- Intermediate Representation: .bin, .xml
- Deploy using the Inference Engine
Supported Frameworks

Breadth of supported frameworks to enable developers with flexibility

- TensorFlow
- Caffe
- mxnet
- ONNX
- KALDI
- OpenVINO

(and other tools via ONNX* conversion)

Supported Frameworks and Formats ➔ https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Introduction.html#SupportedFW
Configure the Model Optimizer for your Framework ➔ https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_Config_Model_Optimizer.html
Core Components

Model optimization to deployment

Model Optimizer

- A Python-based tool to import trained models and convert them to Intermediate Representation
- Optimizes for performance or space with conservative topology transformations
- Hardware-agnostic optimizations

Development Guide ➔

Inference Engine

- High-level, C, C++ and Python, inference runtime API
- Interface is implemented as dynamically loaded plugins for each hardware type
- Delivers best performance for each type without requiring users to implement and maintain multiple code pathways

Development Guide ➔
Model Optimization
Breadth of supported frameworks to enable developers with flexibility

Model Optimizer loads a model into memory, reads it, builds the internal representation of the model, optimizes it, and produces the Intermediate Representation.

Optimization techniques available are:

— Linear operation fusing
— Stride optimizations
— Group convolutions fusing

Note: Except for ONNX (.onnx model formats), all models have to be converted to an IR format to use as input to the Inference Engine.

.xml – describes the network topology
.bin – describes the weights and biases binary data
Inference Engine
Common high-level inference runtime for cross-platform flexibility
Post-Training Optimization Tool
Conversion technique that reduces model size into low-precision without re-training

Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.

Different optimization approaches are supported: quantization algorithms, sparsity, etc.
Deep Learning Workbench

Web-based UI extension tool for model analyses and graphical measurements

- **Visualizes performance data for** topologies and layers to aid in model analysis
- **Automates analysis** for optimal performance configuration (streams, batches, latency)
- **Experiment with INT8 or Winograd calibration** for optimal tuning using the Post Training Optimization Tool
- Provide **accuracy information** through accuracy checker
- **Direct access to models** from public set of Open Model Zoo
- Enables **remote profiling**, allowing the collection of performance data from multiple different machines without any additional set-up.
Compounding Effect of Hardware and Software

Use Intel® Xe Graphics + CPU combined for maximum inferencing

Tiger Lake + Intel® Distribution of OpenVINO™ toolkit vs Coffee Lake CPU

Using the Multi-device plugin

The above is preliminary performance data based on pre-production components. For more complete information about performance and benchmark results, visit [www.intel.com/benchmarks](http://www.intel.com/benchmarks). See backup for configuration details.
Pre-Trained Models and Public Models

Open-sourced repository of pre-trained models and support for public models

Use free Pre-trained Models to speed up development and deployment

Take advantage of the Model Downloader and other automation tools to quickly get started

Iterate with the Accuracy Checker to validate the accuracy of your models

100+ Pre-trained Models
Common AI tasks
Object Detection
Object Recognition
Reidentification
Semantic Segmentation
Instance Segmentation
Human Pose Estimation
Image Processing
Text Detection
Text Recognition
Text Spotting
Action Recognition
Image Retrieval
Compressed Models
Question Answering

100+ Public Models
Pre-optimized external models
Classification
Segmentation
Object Detection
Human Pose Estimation
Monocular Depth Estimation
Image Inpainting
Style Transfer
Action Recognition
Colorization
DEMO
Demos and Reference Implementations

Quickly get started with example demo applications and reference implementations

Take advantage of pre-built, open-sourced example implementations with step-by-step guidance and required components list

- Face Access Control - C++
- Intruder Detector - C++
- Machine Operator Monitor - C++
- Machine Operator Monitor - Go
- Motor Defect Detector - Python
- Object Flaw Detector - C++
- Object Size Detector - C++
- Object Size Detector - Go
- Parking Lot Counter - C++
- Parking Lot Counter - Go
- People Counter - C++
- Restricted Zone Notifier - Go
- Shopper Gaze Monitor - C++
- Shopper Mood Monitor - Go
- Store Aisle Monitor - C++
- Store Traffic Monitor - C++
- Store Traffic Monitor - Python

IAGS Intel Architecture, Graphics, and Software
Intel Low Precision Optimization Tool Tutorial
The motivation for low precision

- Lower Power
- Lower memory bandwidth
- Lower storage
- Higher performance

Important:
Acceptable accuracy loss
The key term:

▪ Quantization
Quantization in a nutshell

Floating Point
96.1924
32-bit

Integer
96
8 bit

Floating Point
96.1924
32-bit

Integer
96
8 bit
Challenge & Solution of Low Precision Optimization Tool (for Inferencing in Deep Learning)

- Low Precision Inference can speed up the performance by reducing the computing, memory and storage of AI model.
- Intel provides solution to cover the challenge of it:

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Intel Solution</th>
<th>How</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware support</td>
<td>Intel® Deep Learning Boost supported by the Second-Generation Intel® Xeon® Scalable Processors and later.</td>
<td>VNNI intrinsic. Support INT8 MulAdd.</td>
</tr>
<tr>
<td>Complex to convert the FP32 model to INT8/BF16 model</td>
<td>Intel® Low Precision Optimization Tool (LPOT)</td>
<td>Unified quantization API</td>
</tr>
<tr>
<td>Accuracy loss in converting to INT8 model</td>
<td>Intel® Low Precision Optimization Tool (LPOT)</td>
<td>Auto tuning</td>
</tr>
</tbody>
</table>
Product Definition

- Convert the FP32 model to INT8/BF16 model. Optimize the model in same time.
- Support multiple Intel optimized DL frameworks (TensorFlow, PyTorch, MXNet) on both CPU and GPU.
- Support automatic accuracy-driven tuning, along with additional custom objectives like performance, model size, or memory footprint.
- Provide the easy extension capability for new backends (e.g., PDPD, ONNX RT) and new tuning strategies/metrics (e.g., HAWQ from UCB).
## Tuning Zoo

The followings are the models supported by Intel® Low Precision Optimization Tool for auto tuning.

<table>
<thead>
<tr>
<th>TensorFlow Model</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50 V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet50 V1.5</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet101</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V2</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V3</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V4</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNetV2_50</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNetV2_101</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNetV2_152</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception ResNet V2</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>SSD ResNet50 V1</td>
<td>Object Detection</td>
</tr>
<tr>
<td>Wide &amp; Deep</td>
<td>Recommendation</td>
</tr>
<tr>
<td>VGG16</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>VGG19</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Style_transfer</td>
<td>Style Transfer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PyTorch Model</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-Large RTE</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Large QNLI</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Large CoLA</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base SST-z</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base RTE</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base STS-B</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base CoLA</td>
<td>Language Translation</td>
</tr>
<tr>
<td>BERT-Base MRPC</td>
<td>Language Translation</td>
</tr>
<tr>
<td>DLRM</td>
<td>Recommendation</td>
</tr>
<tr>
<td>BERT-Large MRPC</td>
<td>Language Translation</td>
</tr>
<tr>
<td>ResNet50 V1.5 32x8d</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>BERT-Large SQUAD</td>
<td>Language Translation</td>
</tr>
<tr>
<td>ResNet50 V1.5</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet18</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V3</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>YOLO V3</td>
<td>Object Detection</td>
</tr>
<tr>
<td>Peleenet</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNest50</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>SE_ResNet50_32x4d</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet50 V1.5 QAT</td>
<td>Image Recognition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MxNet Model</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50 V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>MobileNet V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>MobileNet V2</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>SSD-ResNet50</td>
<td>Object Detection</td>
</tr>
<tr>
<td>SqueezeNet V1</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>ResNet18</td>
<td>Image Recognition</td>
</tr>
<tr>
<td>Inception V3</td>
<td>Image Recognition</td>
</tr>
</tbody>
</table>
Auto-tuning Flow

Tunable Configurations

Quantization

Quantizer

FP32 Model → Low-precision Model

Evaluator

(Accuracy metrics, Performance etc.)

Next Config

Model Inspect

Tuning Strategy

Optimal Solution
System Requirements

- **Hardware**
  Intel® Low Precision Optimization Tool supports systems based on Intel 64 architecture or compatible processors.
  The quantization model could get acceleration by Intel® Deep Learning Boost if running on the Second-Generation Intel® Xeon® Scalable Processors and later:
  Verified:
  - Cascade Lake & Cooper Lake, with Intel DL Boost VNNI
  - Skylake, with AVX-512 INT8

- **OS: Linux**
  Verified: CentOS 7.3 & Ubuntu 18.04

- **Software**
  Intel® Low Precision Optimization Tool requires to install Intel optimized framework version for TensorFlow, PyTorch, and MXNet.

<table>
<thead>
<tr>
<th>Verified Release</th>
<th>Installation Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Optimization for TensorFlow: v1.15 (up1), v2.1, v2.2, v2.3</td>
<td>pip install intel-tensorflow==2.3.0</td>
</tr>
<tr>
<td>PyTorch: v1.5</td>
<td>pip install torch==1.5.0+cpu****</td>
</tr>
<tr>
<td>MXNet: v1.6, v1.7</td>
<td>pip install mxnet-mkl==1.6.0</td>
</tr>
</tbody>
</table>
Installation

▪ **Install from Intel AI Analytics Toolkit (Recommended)**

source /opt/intel/oneapi/setvars.sh
conda activate tensorflow
cd /opt/intel/oneapi/iLiT/latest
sudo ./install_iLiT.sh

▪ **Install from source**

git clone https://github.com/intel/lpot.git
cd lpot
python setup.py install

▪ **Install from binary**

# install from pip
pip install lpot
# install from conda
conda install lpot -c intel -c conda-forge

For more detailed installation info, please refer to https://github.com/intel/lpot
Usage: Simple Python API + YAML config

LPOT is designed to reduce the workload of user and keep the flexibility.

Python API
- Simple API is easy to integrated in original training/inference script.

YAML
- Common functions are integrated and controlled by parameters;
- Templates are easy to refer;
- Lots of advance parameters provide powerful tuning capability.

FP32 model
- YAML file (template-based)

Launcher code based on API

Training/Inference script

INT8 model
- Coding-free (80%): template-based configs
- Coding-needed (20%): user providing callback functions
Python API

- Core User-facing API:
  - Quantization()
    - Follow a specified tuning strategy to tune a low precision model through QAT or PTQ which can meet pre-defined accuracy goal and objective.

```python
class Quantization(object):
    def __init__(self, conf_fname):
        ...

    def __call__(self, model, q_dataloader=None, q_func=None,
                 eval_dataloader=None, eval_func=None):
        ...
```
Intel LPOT YAML Configure

Intel LPOT YAML config consists of 6 building blocks:

- model
- device
- quantization
- evaluation
- tuning

```yaml
# imit yaml building block
model: # model specific info, such as model name, framework, input/output node name required for tensorflow.
  ...

device: ... # the device imit runs at, cpu or gpu. default is cpu.

quantization: # the setting of calibration/quantization behavior. only required for PTQ and QAT.
  ...

evaluation: # the setting of how to evaluate a model.
  ...

tuning: # the tuning behavior, such as strategy, objective, accuracy criterion.
  ...
```
Intermediate: TensorFlow HelloWorld

```python
import tensorflow as tf

(train_images, train_labels), (test_images, test_labels) =
kerns.datasets.fashion_mnist.load_data()

train_images = train_images.astype(np.float32) / 255.0
test_images = test_images.astype(np.float32) / 255.0

model = tf.keras.models.load_model("../models/simple_model")
```

This example shows how to create LPOT calibration and evaluation dataloader by code and pass them to LPOT for tune.

Full example: https://github.com/intel/lpot/tree/master/examples/hello_world
DEMO
Which Toolkit should I use
## Which Toolkit to Use When?

<table>
<thead>
<tr>
<th>Intel® AI Analytics Toolkit</th>
<th>OpenVINO™ Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Value Prop</strong></td>
<td><strong>Key Value Prop</strong></td>
</tr>
<tr>
<td>• Provide performance and easy integration across end-to-end data science pipeline for efficient AI model development</td>
<td>• Provide leading performance and efficiency for DL inference solutions to deploy across any Intel HW (cloud to edge).</td>
</tr>
<tr>
<td>• Maximum compatibility with opensource FWKs and Libs with drop-in acceleration that require minimal to no code changes</td>
<td>• Optimized package size for deployment based on memory requirements</td>
</tr>
<tr>
<td>• Audience: Data Scientists; AI Researchers; DL/ML Developers</td>
<td>• Audience: AI Application Developers; Media and Vision Developers</td>
</tr>
<tr>
<td><strong>Use Cases</strong></td>
<td><strong>Use Cases</strong></td>
</tr>
<tr>
<td>• Data Ingestion, Data pre-processing, ETL operations</td>
<td>• Inference apps for vision, Speech, Text, NLP</td>
</tr>
<tr>
<td>• Model training and inference</td>
<td>• Media streaming / encode, decode</td>
</tr>
<tr>
<td>• Scaling to multi-core / multi-nodes / clusters</td>
<td>• Scale across HW architectures – edge, cloud, datacenter, device</td>
</tr>
<tr>
<td><strong>HW Support</strong></td>
<td><strong>HW Support</strong></td>
</tr>
<tr>
<td>• CPUs - Datacenter and Server segments – Xeons, Workstations</td>
<td>• CPU - Xeons, Client CPUs and Atom processors</td>
</tr>
<tr>
<td>• GPU - ATS and PVC (in future)</td>
<td>• GPU - Gen Graphics; DG1 (current), ATS, PVC (in future)</td>
</tr>
<tr>
<td>• VPU - NCS &amp; Vision Accelerator Design Products,</td>
<td>• VPU - NCS &amp; Vision Accelerator Design Products,</td>
</tr>
<tr>
<td>• FPGA</td>
<td>• FPGA</td>
</tr>
<tr>
<td>• GNA</td>
<td>• GNA</td>
</tr>
</tbody>
</table>

### Use Intel® Low Precision Optimization Tool when using AI Analytics Toolkit
- Supports BF16 for training and FP16, Int8 and BF16 for Inference
- Seamlessly integrates with Intel optimized frameworks
- Available in the AI toolkit and independently

### Use Post Training Optimization Tool when using OpenVINO™ Toolkit
- Supports FP16, Int8 and BF16 for inference
- Directly works with Intermediate Representation Format
- Available in the Intel Distribution of OpenVINO toolkit
- Provides Training extension via NNCF for PyTorch with FP16, Int8

**Exception:** If a model is not supported by OpenVINO™ toolkit for Inference deployment, build custom layers for OV or fall back to the AI Analytics Toolkit and use optimized DL frameworks for inference.
AI Development Workflow

**Determine Use Case**

**Data Analytics**
- Data ingestion and Pre-processing
- Use AI Kit (Modin, Omnisci, Pandas, Numpy, Scipy)

**Machine Learning**
- Classical ML Training and Prediction
- Use AI Kit (Scikit-learn+Daal4py, XGBoost)
- Optimize primitives for DL FWKs
- Use Base Kit (oneDNN, oneCCL)
- Train DL model on Intel (CPU, dGPU)
- Use AI Kit (Intel-optimized TF, Pytorch)
- Re-train a model on custom data
- Use AI Kit (Intel-optimized TF, Pytorch)
- Pick a Pre-trained model optimized by Intel
- Use AI Kit (Model Zoo for IA)

**Deep Learning**
- Run DL Inference on trained model
- Use AI Kit (Intel-optimized TF, Pytorch)
- Further optimize
- Use AI Kit (Low precision Opt Tool + Intel-optimized TF, Pytorch)
- Convert to Low Precision and run inference
- Use AI Kit (Intel-optimized TF, Pytorch)

**Public models trained with any FWK – TF, Caffe, ONNX, MXNet, etc.**

**Deploy DL Models on Intel® platforms**

**Pick a pre-trained model in IR format (Open Model Zoo)**

**Use OpenVINO™ Toolkit**
- Use AI Kit (Low precision Opt Tool + Intel-optimized TF, Pytorch)
- Deploy DL Models on Intel® platforms

**Alternately there are options to directly download any of the Intel optimized FWKs, ML libraries & Tools independently. Our recommendation is to get them through the toolkit for seamless interoperability and good out of box experience.**

**Native Code developers, Framework Developers**
- Data Scientists, AI Researchers, ML/DL Developers
- AI, Media and Computer Vision Application Developers

**Native Code developers, Framework Developers**
- CPU
- GPU
- VPU
- FPGA
- GNA
A comprehensive workflow to optimize your DL model for the Intel Hardware that will be used for running inference.
1) We run the demo on DC
   - TF demo
   - PyTorch demo
   - future: (ATS demo)

2) Slide on how to access DevCloud

2) What’s behind the DC
Intel DevCloud: Getting started with oneAPI
Objectives of the External DevCloud Strategy

1. Demonstrate the promise of oneAPI.
2. Provide developers easy access to oneAPI h/w & s/w environment
3. Get high value feedback on oneAPI tools, libraries, language.
4. Seed research, papers, curriculum, lighthouse apps (the metrics output).
5. Support two tracks with web front end for consistent experience:
   • oneAPI production hardware/software
   • NDA SDP hardware/software
One API DevCloud Architecture

Users

Access: ssh/Jupyter/Browser

Login Node

Disk TB1

Disk TB2

Disk TB3

Disk TB4

Disk TB5

Storage Disks

AI DevCloud: Xeon SKL/CLX Cluster

Login Node

SKL node11

SKL node12

SKL node13

SKL node14

CLX/AEP node21

CLX/AEP node22

CLX/AEP node23

FPGA/Arria 10 node31

FPGA/Arria 10 node32

FPGA/Arria 10 node33

Gen9 node1

Gen9 node2

Gen9 node3

Gen9 node4

Gen9 node5

DG! node1

DG! node2

ATS node3

ATS node4

PVC node5

Intel Opt Frameworks Available today

TensorFlow

Caffe

MxNet

Python, PyTorch

Direct Programming

DPC++

OpenVINO

ToolKit1

ToolKit2

OneAPI Programming

MKL

TBB

Media SDK

MKL-DNN

Parallel STL

DLDT

DAAL

.*

MLSL

New Additions for Under NDA

Container 1

Container 2

Container 3

Container 4

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

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](http://www.intel.com).

Intel, the Intel logo, Xeon™, Arria™ and Moovdis™ 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.
### 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](http://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.

---

<table>
<thead>
<tr>
<th>System Board</th>
<th>Intel prototype, TGL U DDR4 SODIMM RVP</th>
<th>ASUSTeK COMPUTER INC. / PRIME Z370-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>11th Gen Intel® Core™ -5-1145G7E @ 2.6 GHz.</td>
<td>8th Gen Intel ® Core™ i5-8500T @ 3.0 GHz.</td>
</tr>
<tr>
<td>Sockets / Physical cores</td>
<td>1 / 4</td>
<td>1 / 6</td>
</tr>
<tr>
<td>HyperThreading / Turbo Setting</td>
<td>Enabled / On</td>
<td>Na / On</td>
</tr>
<tr>
<td>Memory</td>
<td>2 x 8198 MB 3200 MT/s DDR4</td>
<td>2 x 16384 MB 2667 MT/s DDR4</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu* 18.04 LTS</td>
<td>Ubuntu* 18.04 LTS</td>
</tr>
<tr>
<td>Kernel</td>
<td>5.8.0-050800-generic</td>
<td>5.3.0-24-generic</td>
</tr>
<tr>
<td>Software</td>
<td>Intel® Distribution of OpenVINO™ toolkit 2021.1.075</td>
<td>Intel® Distribution of OpenVINO™ toolkit 2021.1.075</td>
</tr>
<tr>
<td>BIOS</td>
<td>Intel TGLIFI11.R00.3243.A04.2006302148</td>
<td>AMI, version 2401</td>
</tr>
<tr>
<td>BIOS release date</td>
<td>Release Date: 06/30/2021</td>
<td>7/12/2019</td>
</tr>
<tr>
<td>BIOS Setting</td>
<td>Load default settings</td>
<td>Load default settings, set XMP to 2667</td>
</tr>
<tr>
<td>Test Date</td>
<td>9/9/2021</td>
<td>9/9/2021</td>
</tr>
<tr>
<td>Precision and Batch Size</td>
<td>CPU: INT8, GPU: FP16-INT8, batch size: 1</td>
<td>CPU: INT8, GPU: FP16-INT8, batch size: 1</td>
</tr>
<tr>
<td>Number of Inference Requests</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Number of Execution Streams</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Power (TDP Link)</td>
<td>28 W</td>
<td>35W</td>
</tr>
<tr>
<td>Price (USD) Link on Sep 22,2021</td>
<td>$309</td>
<td>$192</td>
</tr>
</tbody>
</table>

1): Memory is installed such that all primary memory slots are populated.

2): Testing by Intel as of September 9, 2021