Deep Learning Optimizations
## Intel® AI Analytics Toolkit

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

### Who needs this product?
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

### Features/Benefits
- **Deep Learning**
  - Intel® Optimization for TensorFlow
  - Intel® Optimization for PyTorch
  - Intel® Neural Compressor
  - Model Zoo for Intel® Architecture
- **Machine Learning**
  - Intel® Extension for Scikit-learn
  - Intel-optimized XGBoost
- **Data Analytics**
  - Intel® Distribution of Modin
  - OmniSci Backend
- **Intel-optimized Python**
  - NumPy
  - SciPy
  - Numba
  - Pandas
  - Data Parallel Python

### Hardware Support
- CPU
- GPU

Hardware support varies by individual tool. Architecture support will be expanded over time.

### Get the Toolkit
- [HERE](#) or via these locations
  - Intel Installer
  - Docker
  - Apt, Yum
  - Conda
  - Intel® DevCloud

Back to Domain-specific Toolkits for Specialized Workloads
**Deep learning and AI ecosystem** includes edge and datacenter applications.

- **Open source frameworks** (TensorFlow\textsuperscript{*}, PyTorch\textsuperscript{*}, ONNX Runtime\textsuperscript{*})
- OEM applications (Matlab\textsuperscript{*}, DL4J\textsuperscript{*})
- Cloud service providers internal workloads
- **Intel deep learning products** (OpenVINO™, BigDL)

**oneDNN** is an open source performance library for deep learning applications.

- Includes **optimized** versions of **key deep learning functions**
- **Abstracts out** instruction set and other complexities of performance optimizations
- **Same API for both Intel CPU’s and GPU’s**, use the best technology for the job
- **Open** for community contributions

---

**Optimization Notice:** 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
Intel-optimized Deep Learning Frameworks

- Intel-optimized DL frameworks are drop-in replacement
  - No front code change for the user
- Optimizations are upstreamed automatically
- Latest optimizations in extension libraries
~6 years of close collaboration between Intel and Google
Intel® Extension for TensorFlow* Architecture
Major Optimization Methodologies

- oneDNN Integration with TensorFlow
  - Replaces compute-intensive standard TF ops with highly optimized custom oneDNN ops
  - Aggressive op fusions to improve performance of Convolutions and Matrix Multiplications

- bfloat16 and 8-bit low precision data types supported by SPR
  - New matrix-based instructions set, Intel AMX
BF16 API

1. Train with BF16 with AVX-512

```python
# BF16 without AMX
os.environ["ONEDNN_MAX_CPU_ISA"] = "AVX512_BF16"
tf.config.optimizer.set_experimental_options({"auto_mixed_precision_onednn_bfloat16":True})

transformer_layer = transformers.TFDistilBertModel.from_pretrained("distilbert-base-uncased")
tokenizer = transformers.DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
model = build_model(transformer_layer, max_len=160)

# fine tune model according to disaster tweets dataset
if is_tune_model:
    train_input = bert_encode(train.values, tokenizer, max_len=160)
    train_labels = train.target.values
    start_time = time.time()
    train_history = model.fit(train_input, train_labels, validation_split=0.2, epochs=1, batch_size=16)
    end_time = time.time()
    # save model weights so we don’t have to fine tune it every time
    os.makedirs(save_weights_dir, exist_ok=True)
    model.save_weights(save_weights_dir + "/bf16_model_weights.h5")
```

2. Train with BF16 with AMX

```python
# BF16 without AMX
os.environ["ONEDNN_MAX_CPU_ISA"] = "AVX512_BF16"

# BF16 with AMX
os.environ["ONEDNN_MAX_CPU_ISA"] = "AMX_BF16"
```

Turned on by default after TF 2.11
BF16 API (cont.)

3. Inference with BF16 without AMX

```python
# Reload the model as the bf16 model with AVX512 to compare inference time
os.environ['ONEDNN_MAX_CPU_ISA'] = "AVX512_BF16"
tf.config.optimizer.set_experimental_options({‘auto_mixed_precision_onednn_bfloat16’:True})
bf16_model_noAmx = tf.keras.models.load_model('models/my_saved_model_fp32')

bf16_model_noAmx_export_path = "models/my_saved_model_bf16_noAmx"
bf16_model_noAmx.save(bf16_model_noAmx_export_path)
```

4. Inference with BF16 with AMX

```python
# Reload the model as the bf16 model with AMX to compare inference time
os.environ['ONEDNN_MAX_CPU_ISA'] = "AMX_BF16"
tf.config.optimizer.set_experimental_options({‘auto_mixed_precision_onednn_bfloat16’:True})
bf16_model_withAmx = tf.keras.models.load_model('models/my_saved_model_fp32')

bf16_model_withAmx_export_path = "models/my_saved_model_bf16_with_amx"
bf16_model_withAmx.save(bf16_model_withAmx_export_path)
```
TensorFlow Benchmark: SPR Inference (Batch Size = 1)

Inference latency speedup: the higher the better

SPR TENSORFLOW INFERENCCE LATENCY W/AMX

BF16: 2.00-6.90X
INT8: 3.18-13.61X

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
Intel® Optimization for PyTorch® upstream
Intel® Optimization for PyTorch*

ECOSYSTEM
- torchvision
- TorchServe
- Hugging Face
- DeepSpeed
- PyTorch Lightning
- PyG
- ...

FRAMEWORKS
- PyTorch
- Intel® Extension for PyTorch*

LIBRARIES
- oneDNN
- oneCCL

* Other names and brands may be claimed as the property of others
Intel® Extension for PyTorch* Architecture

- **Eager-Mode**
  - Custom Modules, Optimizers, Quantization

- **Graph-Mode**
  - Custom Fusion Passes
  - oneDNN Fusion Passes

- **ATen Ops**
  - Custom Ops

- **Graph Ops**
  - Custom Fused Ops
  - oneDNN Fused Ops

**Kernels**
- GPU Custom & ATen Kernels
- oneDNN GPU Kernels
- oneMKL GPU Kernels
- CPU Custom & ATen Kernels
- oneDNN CPU Kernels
- oneMKL CPU Kernels

**OneAPI**
- SYCL Language
- oneDNN GPU
- oneMKL GPU
- SYCL Runtime
- LevelZero Runtime

**OpenMP**
- Thread Runtime

**Device-agnostic**
- GPU related
- CPU related
Major Optimization Methodologies

- General performance optimization and Intel new feature enabling in PyTorch upstream
- Additional performance boost and early adoption of aggressive optimizations through Intel® Extension for PyTorch*

**Operator Optimization**
- Vectorization
- Parallelization
- Memory Layout
- Low Precision

**Graph Optimization**
- Operator fusion
- Constant folding

**Runtime Extension**
- Thread affinity
- Memory allocation
Training w/AMX BF16 on Intel Extension for PyTorch

```python
import torch
import torchvision
import intel_extension_for_pytorch as ipex

LR = 0.001
DOWNLOAD = True
DATA = 'datasets/cifar10/'

transform = torchvision.transforms.Compose([  
    torchvision.transforms.Resize((224, 224)),  
    torchvision.transforms.ToTensor(),  
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = torchvision.datasets.CIFAR10(  
    root=DATA,  
    train=True,  
    transform=transform,  
    download=DOWNLOAD,  
)

train_loader = torch.utils.data.DataLoader(  
    dataset=train_dataset,  
    batch_size=128
)

model = torchvision.models.resnet50()
criterion = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=LR, momentum=0.9)
model.train()

model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.bfloat16)

for batch_idx, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    with torch.cpu.amp.autocast():
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        print(batch_idx)
    torch.save({  
        'model_state_dict': model.state_dict(),  
        'optimizer_state_dict': optimizer.state_dict(),  
    }, 'checkpoint.pth')
```
Inference w/AMX BF16 on Intel Extension for PyTorch

```python
import torch
import torchvision.models as models

model = models.resnet50(weights='ResNet50_Weights.DEFAULT')
model.eval()
data = torch.rand(1, 3, 224, 224)

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

with torch.no_grad(), torch.cpu.amp.autocast():
    model = torch.jit.trace(model, torch.rand(1, 3, 224, 224))
    model = torch.jit.freeze(model)

model(data)
```

---

**BERT**

```python
import torch
from transformers import BertModel

model = BertModel.from_pretrained("bert-base-uncased")
model.eval()
vocab_size = model.config.vocab_size
batch_size = 1
seq_length = 512
data = torch.randint(vocab_size, size=[batch_size, seq_length])

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

with torch.no_grad(), torch.cpu.amp.autocast():
    d = torch.randint(vocab_size, size=[batch_size, seq_length])
    model = torch.jit.trace(model, (d,), check_trace=False, strict=False)
    model = torch.jit.freeze(model)

model(data)
```
Runtime Optimizations with Launch Script

- Automates configuration settings to optimize on topology
  - OpenMP library: Intel OpenMP library, GNU OpenMP library
  - Memory allocator: PyTorch default, Jemalloc, TCMalloc
  - Number of instances: single, multiple
  - Number of cores per instance
- **Usage Guide** with options and examples
- **Sample Command**
  - `ipexrun --ninstances 4 --ncore_per_instance 4 --enable_tcmalloc
    ${SCRIPT_PATH}`
PyTorch Benchmark: SPR Inference (Batch Size = 1)

Inference latency speedup: the higher the better

BF16: 4.29-6.91X
INT8: 7.33-13.66X
Intel® Neural Compressor (INC)

<table>
<thead>
<tr>
<th>Model</th>
<th>TensorFlow</th>
<th>PyTorch</th>
<th>ONNX</th>
<th>MXNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Facing APIs</td>
<td>Quantization, Pruning, Knowledge Distillation, Graph Optimization, ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compressions</td>
<td>Quantization</td>
<td>Pruning</td>
<td>Mix Precision</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post training static quantization</td>
<td>magnitude pruning</td>
<td>FP32 -&gt; INT8/BF16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post training dynamic quantization</td>
<td>Gradient sensitivity pruning</td>
<td>FP32 -&gt; BF16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quantization-aware training</td>
<td>Knowledge Distillation</td>
<td>FP32 -&gt; Opt Fp32</td>
<td></td>
</tr>
<tr>
<td>Backends</td>
<td>TensorFlow, PyTorch, ONNX Runtime, MXNet, Engine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware platforms</td>
<td>Intel CPU</td>
<td>Intel GPU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://github.com/intel/neural-compressor

Installation:
- pip install neural-compressor
- conda install neural-compressor -c conda-forge -c intel
INT8 Quantized Inference Performance
Uses Intel® Optimization for Tensorflow and Intel® Neural Compressor

INT8 Inference Throughput Scaling up to 2.8x and Accuracy Drop within 0.6%

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
**Model Zoo for Intel® Architecture**

### Language Modeling

<table>
<thead>
<tr>
<th>Model</th>
<th>Framework</th>
<th>Mode</th>
<th>Model Documentation</th>
<th>Benchmark/Test Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT large</td>
<td>TensorFlow</td>
<td>Inference</td>
<td>FP32 BF8t16 FP16</td>
<td>SQuAD</td>
</tr>
<tr>
<td>BERT large</td>
<td>TensorFlow</td>
<td>Training</td>
<td>FP32 BF8t16 FP16</td>
<td>SQuAD and MRPC</td>
</tr>
<tr>
<td>BERT large Sapphire Rapids</td>
<td>TensorFlow</td>
<td>Inference</td>
<td>FP32 BF8t16 int8 BF8t32</td>
<td>SQuAD</td>
</tr>
<tr>
<td>BERT large Sapphire Rapids</td>
<td>TensorFlow</td>
<td>Training</td>
<td>FP32 BF8t16 BF8t32</td>
<td>SQuAD</td>
</tr>
<tr>
<td>DistilBERT base</td>
<td>TensorFlow</td>
<td>Inference</td>
<td>FP32 BF8t16 int8 FP16</td>
<td>SST-2</td>
</tr>
<tr>
<td>BERT base</td>
<td>PyTorch</td>
<td>Inference</td>
<td>FP32 BF8t16</td>
<td>BERT Base SQuAD1.1</td>
</tr>
<tr>
<td>BERT large</td>
<td>PyTorch</td>
<td>Inference</td>
<td>FP32 int8 BF8t16</td>
<td>BERT Large SQuAD1.1</td>
</tr>
<tr>
<td>BERT large</td>
<td>PyTorch</td>
<td>Training</td>
<td>FP32 BF8t16 BF8t32</td>
<td>preprocessed text dataset</td>
</tr>
<tr>
<td>DistilBERT base</td>
<td>PyTorch</td>
<td>Inference</td>
<td>FP32 Int8 BF8t16</td>
<td>DistilBERT Base SQuAD1.1</td>
</tr>
<tr>
<td>RNN-T</td>
<td>PyTorch</td>
<td>Inference</td>
<td>FP32 BF8t16 BF8t32</td>
<td>RNN-T dataset</td>
</tr>
<tr>
<td>RoBERTa base</td>
<td>PyTorch</td>
<td>Inference</td>
<td>FP32 BF8t16</td>
<td>RoBERTa Base SQuAD 2.0</td>
</tr>
<tr>
<td>T5</td>
<td>PyTorch</td>
<td>Inference</td>
<td>FP32 Int8</td>
<td></td>
</tr>
</tbody>
</table>

### TensorFlow BERT Large inference

**Description**

This document has instructions for running BERT Large inference using Intel-optimized Tensorflow.

**Datasets**

**BERT Large Data**

Download and unzip the BERT Large uncased (whole word masking) model from the [google bert repo](https://huggingface.co/models). Then, download the Stanford Question Answering Dataset (SQuAD) dataset file `dev-v1.1.json` into the `~/.unsaved_data/SQuAD/` directory that was just unzipped.

```
wget https://storage.googleapis.com/bert_models/2020_02_05/uncased_L-24_H-1024_A-16.zip
unzip uncased_L-24_H-1024_A-16.zip
```

```
```

Set the `SQUAD_DIR` to point to that directory when running BERT Large inference using the SQuAD data.

**Quick Start Scripts**

<table>
<thead>
<tr>
<th>Script name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>profile.sh</code></td>
<td>This script runs inference in profile mode with a default batch size of 128.</td>
</tr>
<tr>
<td><code>inference.sh</code></td>
<td>Runs realtime inference using a default batch size of 128, for the specified precision (fp32, bf8t16 or fp16). To run inference for throughput, set <code>BATCH_SIZE</code> environment variable.</td>
</tr>
</tbody>
</table>
Key Features & Benefits

- **Accelerate end-to-end AI and Data Science pipelines and achieve drop-in acceleration** with optimized Python tools built using oneAPI libraries (i.e. oneMKL, oneDNN, oneCCL, oneDAL, and more)

- **Achieve high-performance for deep learning training and inference** with Intel-optimized versions of TensorFlow and PyTorch, and low-precision optimization with support for **int8 and bfloat16**

- **Expedite development** by using the open-source **pre-trained deep learning models optimized by Intel** for best performance via Model Zoo for Intel® Architecture

- **Seamlessly scale Pandas workflows across multi-node dataframes** with Intel® Distribution of Modin, **accelerate analytics** with performant backends such as OmniSci

- **Increase machine learning model accuracy and performance** with algorithms in Scikit-learn and XGBoost optimized for Intel architectures

- **Supports cross-architecture development** (Intel® CPUs/GPUs) and compute
Useful Links

- Intel® AI Analytics Toolkit (AI Kit)
- Intel® Extension for PyTorch*
- Intel® Extension for TensorFlow*
- Intel® Neural Compressor
- oneAPI-samples GitHub
- Model Zoo for Intel® Architecture GitHub
Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

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 does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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