Optimization Guide

Intel® Xeon® Scalable Processor

借助基于第 3 代英特尔® 至强® 可扩展处理器的英特尔® AVX-512 和英特尔® 深度学习加速调优指南进行深度学习

Revision Record

Date	Revision	Description
04/06/2021	1.0	Initial draft
11/08/2021	1.2	Corrected an error by replacing byte with bit
07/20/2022	1.3	Updated the Neural Compressor section

概述

本用户指南旨在介绍如何使用第 3 代英特尔® 至强® 可扩展处理器平台（代号 Ice lake/Whitley）执行机器学习和深度学习的相关任务。在英特尔® 至强® 可扩展处理器平台上运行机器学习和深度学习工作负载具有以下优点:

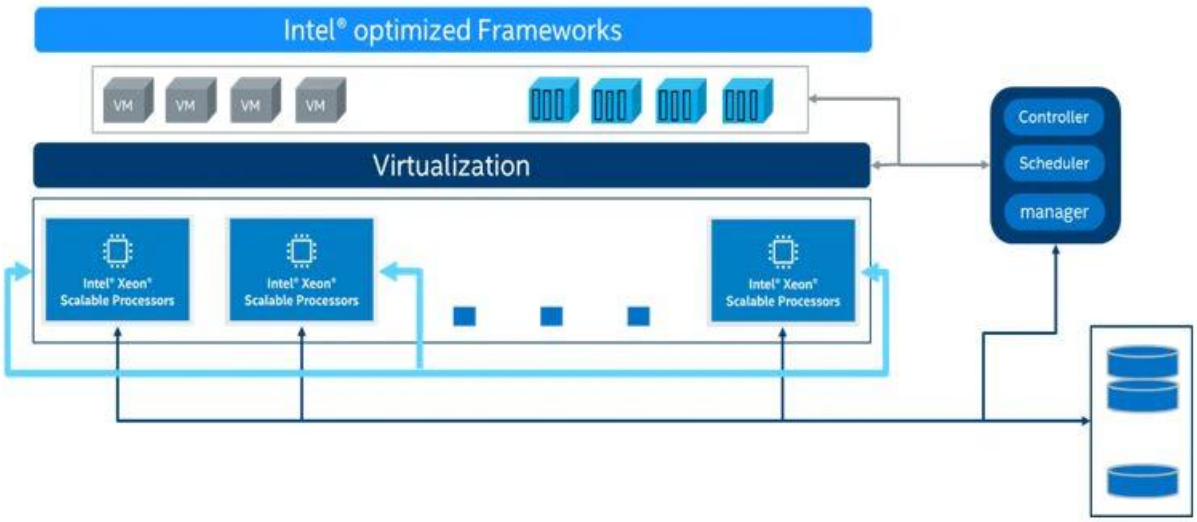
- 非常适合处理大内存型工作负载和医学成像，GAN，地震分析，基因组测序等中使用的 3D-CNN 拓扑。
- 可以使用简单的 numactl 命令进行灵活的核心控制，即使在批量较小时也非常适用于实时推理。
- 强大的生态系统支持，可直接在大型集群上进行分布式训练（例如直接基于数据源处进行计算）。避免了额外添加大容量存储和昂贵的缓存机制来进行规模化架构的训练
- 可以在同一个集群上支持多种工作负载（HPC / BigData / AI）获取更优的 TCO。
- 通过 SIMD 加速，满足许多实际深度学习应用程序的计算要求。
- 同一套基础架构直接用于训练和推理。

典型的深度学习应用程序开发和部署涉及以下阶段:

这些不同的阶段需要配置以下的资源，选择合适的资源可以大大加速你的 AI 业务的效率：

- 计算能力
- 内存
- 数据集的存储设备
- 计算节点之间的互联网络
- 优化的软件

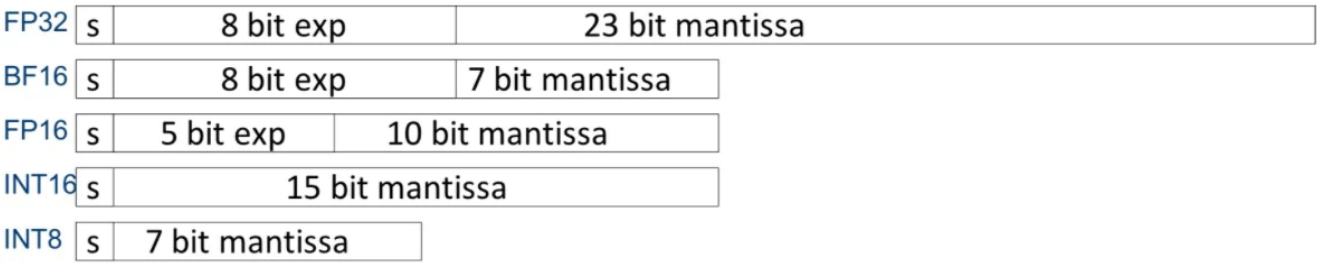
从数据集准备、模型训练、模型优化及部署等各个环节都可以在基于英特尔® 至强® 可扩展处理器平台的基础架构上实现，同时支持训练和推理的机器学习/深度学习平台。基础架构示意图如下:



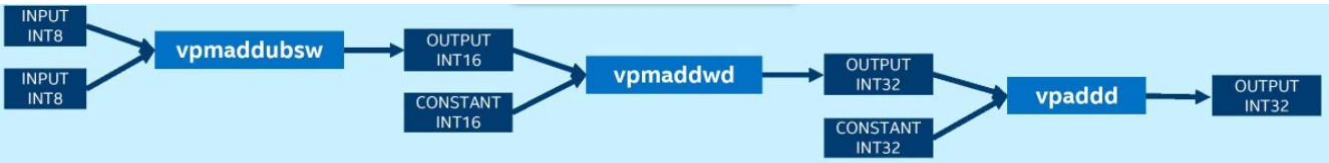
英特尔® AVX-512 和英特尔® 深度学习加速介绍

英特尔® 高级矢量扩展指令集 512 (英特尔® AVX-512) 是 x86 处理器上一套单指令多数据（SIMD）指令集。相较于传统的单指令单数据指令，SIMD 指令使得一条指令可以完成多组数据的操作。Intel® AVX-512 顾名思义寄存器位宽是 512 b，可以支持 16 路 32b 单精度浮点数或 64 路 8b 整型数。

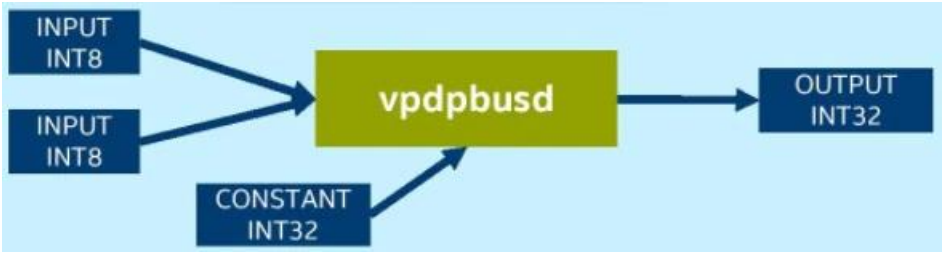
英特尔® 至强® 可扩展处理器可支持多种工作负载，包括复杂的 AI 工作负载； 英特尔® 至强® 可扩展处理器通过英特尔® 深度学习加速 (英特尔® DL Boost) 进一步提升了 AI 计算性能。英特尔® 深度学习加速包含英特尔® AVX-512 VNNI（Vector Neural Network Instructions），是对标准英特尔® AVX-512 指令集的扩展。可以将三条指令合并成一条指令执行，更进一步的发挥新一代英特尔® 至强® 可扩展处理器的计算潜能，提升 INT8 模型的推理性能。目前第 2 代和第 3 代英特尔® 至强 ® 可扩展处理器均支持英特尔® VNNI。



未使用 VNNI 的平台需要 vpmaddubsw、vpmaddwd 和 vpaddd 指令才能完成 INT8 卷积运算中的乘累加:



而拥有 VNNI 的平台上则可以使用一条指令 vpdpbusd 完成 INT8 卷积操作:



开发环境

我们用这个 i 硬件和软件考这本调优指南的建议:

硬件

这个配置是第 3 代英特尔® 至强® 可扩展处理器的硬件。 根据您的使用要求，选择服务器, 内存, 硬盘和网卡。

硬件	型
服务器	Intel® Coyote Pass Server Platform
CPU	Intel® Xeon® PLATINUM 8380 CPU @ 2.30GHz
内存	8*64 GB DDR4, 3200 MT/s

软件

软件	版本
操作系统	Ubuntu 20.04.4 LTS
Kernel	5.4.0

BIOS 设置和硬件选型配置

BIOS 设置

设置的影响	建议
Processor configuration: Hyper-Threading	Enable
SNC (Sub NUMA)	Disable
Boot performance mode	Max Performance
Turbo Mode	Enable
Power and Performance: Hardware P-States	Native Mode

硬件配置建议

机器学习的工作负载尤其是深度学习工作负载，通常都是计算密集型的应用。因此需要对内存, CPU 及硬盘等计算资源都需要选择合适的类型，以便获取最佳的计算性能。总结一些常用设置建议如下:

内存设置

建议将所有的内存通道用满，以便获取到所有内存通道的带宽。

CPU 配置

英特尔处理器中的英特尔 AVX-512 加速单元 FMA 是发挥计算性能的主要部件，人工智能相关的工作负载通常为计算密集的应用。建议采用每个物理核配置 2 个英特尔 AVX512 计算单元的第 3 代英特尔® 至强® 处理器 Gold 6 系列及以上的处理器，以获取更好的计算性能。

网络配置

如果需要搭建跨节点的训练集群，选择高速的网络如 25G/100G 网络能够提供更好的扩展性。

硬盘配置

为了提供工作负载的 IO 效率，建议配置 SSD 及更高读写速度的快速硬盘。

Linux* 系统优化

OpenMP*参数的设置

主要参数的推荐配置如下所示：

- OMP_NUM_THREADS = “容器中的 CPU 内核数”
- KMP_BLOCKTIME = 1 或者 0（具体需要根据模型的类型进行设置）
- KMP_AFFINITY=granularity=fine, verbose, compact,1,0
- CPU 核数的调整

CPU 核数对推理服务性能的影响主要是：

- batchsize 较小时（例如在线类服务），推理吞吐量的提升会随着 CPU 内核数的增加而逐渐减弱，实践中根据使用模型的不同，推荐 8-16 核 CPU 进行服务部署。
- batchsize 较大时（例如离线类服务），推理吞吐量会随着 CPU 内核数的增加呈线性增长，实践中推荐使用 20 核以上的 CPU 进行服务部署。

taskset -C xxx-xxx –p pid (limits the number of CPU cores used in service)

NUMA 设置的影响

对于 NUMA 架构的服务器，NUMA 配置在同一 node 上相比不同 node 上性能通常会有 5%-10% 的提升。

#numactl -N NUMA_NODE -l command args ... (controls NUMA nodes running in service)

Linux Performance Governor 的设置

性能：顾名思义只注重效率，将 CPU 频率固定工作在其支持的最高运行频率上，从而取得最好的性能。

cpupower frequency-set -g performance

CPU C-States 设置

CPU C-States：为了在 CPU 空闲的时候降低功耗，CPU 可以被命令进入低功耗模式。每个 CPU 都有几种功耗模式，这些模式被统称为 C-states 或者 C-modes。

禁用 C-States 可以带来性能的提升。

#cpupower idle-set -d 2,3

使用英特尔® Optimization for TensorFlow* Deep Learning Framework

TensorFlow 是用于大规模机器学习 (ML) 和深度学习 (DL) 的最受欢迎的深度学习框架之一。自 2016 年以来，英特尔和 Google 工程师一直在合作，使用英特尔® oneAPI Deep Neural Network Library (英特尔® OneDNN) 优化 TensorFlow 性能，加速在英特尔® 至强® 可扩展处理器平台上的训练和推理性能。

部署英特尔® Optimization for TensorFlow* Deep Learning Framework

参考：https://www.intel.com/content/www/cn/zh/developer/articles/guide/optimization-for-tensorflow-installation-guide.html

第一步：安装 python3.x 的环境。这里以 anaconda 建立 python3.6 为例

参考网址：https://www.anaconda.com/products/individual

下载 anaconda* 最新版本并安装

```
# wget https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh

# sh Anaconda3-2020.02-Linux-x86_64.sh

# source /root/.bashrc

# conda install python=3.6 (create a Python3.6 environment)

#(base) [root@xx]# python -V

Python 3.6.10
```

第二步：安装英特尔® Optimization for TensorFlow* 版本：intel-tensorflow。

安装最新版本 (2.x)

```
# pip install intel-tensorflow
```

如果您需要安装 tensorflow1.x，推荐安装以下版本，以便利用第 3 代英特尔® 至强® 可扩展处理器平台上的性能加速优势：

```
# pip install https://storage.googleapis.com/intel-optimized-tensorflow/intel_tensorflow-1.15.0up2-cp36-cp36m-manylinux2010_x86_64.whl
```

第三步：设置运行时优化参数

参考网址：

https://github.com/IntelAI/models/blob/master/docs/general/tensorflow/GeneralBestPractices.md

通常有以下两种方式运行推理，从而采用不同的优化设置

批量推理：批次大小 >1，测量每秒可处理多少个输入张量。通常，对于批量推理，可以通过使用同一个 CPU socket 上的所有物理核心来实现最佳性能。

在线推理（也称为实时推理）：批次大小 = 1，处理单个输入张力（批次大小为 1 时）的所需时间测量。在实时推理方案中，通过多实例并发运行来获取最佳的吞吐量。

1：获取系统的物理核个数：

确认目前的物理核配置个数，建议用以下命令获取

```
# lscpu | grep "Core(s) per socket" | cut -d':' -f2 | xargs
```

确认全的物理核配置，建议用以下命令获取：

(You may also use this command to list all physical cores for all sockets)

```
$ lscpu -b -p=Core,Socket | grep -v '^#' | sort -u | wc -l
```

本例子假定为 8 个物理核 (physical cores)。

2：设置优化参数：

通过以下方式设置优化运行参数，有两种方式。请按照您的需求选择一种设置即可。

方法一：直接设置环境运行参数：

```
export OMP_NUM_THREADS=physical cores

export KMP_AFFINITY="granularity=fine,verbose,compact,1,0"

export KMP_BLOCKTIME=1

export KMP_SETTINGS=1
```

方法二：在运行的 python 代码中加入环境变化设置：

```
import os

os.environ["KMP_BLOCKTIME"] = "1"

os.environ["KMP_SETTINGS"] = "1"

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

if FLAGS.num_intra_threads &gt; 0:

os.environ["OMP_NUM_THREADS"]= # &lt;physical cores&gt;

config = tf.ConfigProto()

config.intra_op_parallelism_threads = # &lt;physical cores&gt;
```

```
config.inter_op_parallelism_threads = 1



tf.Session(config=config)
```

运行基于英特尔® Optimization for TensorFlow* 深度学习模型的推理, 支持 FP32/ INT8

本文主要介绍如何运行 ResNet50 的推理基准测试。您可以参考运行您的机器学习/深度学习模型推理。

参考网址：https://github.com/IntelAI/models/blob/master/docs/image_recognition/tensorflow/Tutorial.md

这里以 ResNet50 的推理基准测试为例, 支持 FP32/BFloat16/Int8 三种精度的模型推理。

参考网
址： https://github.com/IntelAI/models/blob/master/benchmarks/image_recognition/tensorflow/resnet50v1_5/README.md

支持 FP32 精度的模型推理:

https://github.com/IntelAI/models/blob/master/benchmarks/image_recognition/tensorflow/resnet50v1_5/README.md#fp32-inference-instructions

支持 INT8 精度的模型推理:

https://github.com/IntelAI/models/blob/master/benchmarks/image_recognition/tensorflow/resnet50v1_5/README.md#int8-inference-instructions

运行基于址：英特尔® Optimization for TensorFlow* 深度学习模型的训练, 支持 FP32

本文主要介绍如何运行 ResNet50 的训练基准测试。您可以参考运行您的机器学习/深度学习模型训练。

基于 FP32 精度的训练:

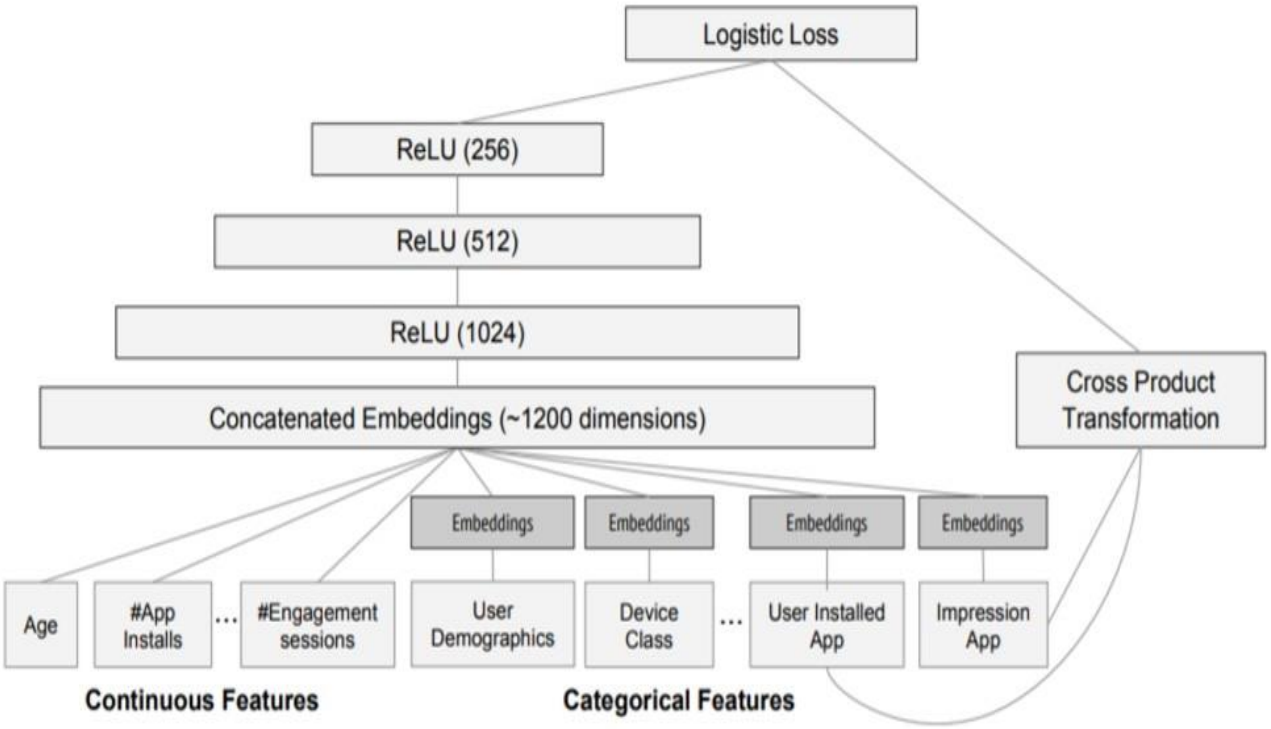
https://github.com/IntelAI/models/blob/master/benchmarks/image_recognition/tensorflow/resnet50v1_5/README.md#fp32-training-instructions

典型应用 - 基于英特尔® Optimization for TensorFlow* 在 Wide & Deep 模型上的推理和训练

在数据中心的众多业务中，使用推荐系统为用户匹配他们感兴趣的内容是一个典型的应用。推荐系统是一种信息过滤系统，能根据用户的档案或者历史行为记录，学习出用户的兴趣爱好，预测出用户对给定物品的评分或偏好。它改变了商家与用户的沟通方式，加强了和用户之间的交互性。

利用深度学习，我们从海量而复杂的原始数据中学习到传统机器使用人工特征工程难以表达的特征间的深度交互。相关的研究成果包括 Wide & Deep、DeepFM、FNN、DCN 等模型。

以 Wide & Deep 模型为例，它的核心思想是结合线性模型的记忆能力（memorization）和 DNN 模型的泛化能力（generalization），在训练过程中同时优化 2 个模型的参数。从而达到整体模型的预测能力最优。其结构如下所示：



Wide 部分

Wide 部分就是一个广义线性模型，输入主要由两部分，一部分是原始特征，另一部分是交互特征。我们可以通过 cross-product transformation 的形式来构造 K 组交互特征：

$$\phi_k(\mathbf{x}) = \prod_{i=1}^{d} x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$

Deep 部分

Deep 部分就是一个 DNN 的模型，每一层计算如下：

$$a^{(l+1)} = f\left(W^{(l)}a^{(l)} + b^{(l)}\right)$$

联合训练

Wide & Deep 模型采用的是联合训练的形式，而非集成。二者的区别就是联合训练共用一个损失函数，然后同时更新各个部分的参数，而集成方法是独立训练 N 个模型，然后进行融合。因此，模型的输出为：

$$P(Y=1|\mathbf{x}) = \sigma\left(\mathbf{w}_{wide}^{T}[\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^{T}a^{(l_f)} + b\right)$$

以上是 Wide & Deep 模型的相关背景。接下来介绍如何运行 Wide & Deep 的推理基准测试。

参考网址：

https://github.com/IntelAI/models/blob/master/docs/recommendation/tensorflow/Tutorial.md

准备数据集

https://github.com/IntelAI/models/tree/master/benchmarks/recommendation/tensorflow/wide_deep_large_ds#Prepare-dataset

支持 FP32 精度的模型推理：

https://github.com/IntelAI/models/tree/master/benchmarks/recommendation/tensorflow/wide_deep_large_ds#fp32-inference-instructions

支持 INT8 精度的模型推理：

https://github.com/IntelAI/models/tree/master/benchmarks/recommendation/tensorflow/wide_deep_large_ds#int8-inference-instructions

基于 FP32 精度的训练：

https://github.com/IntelAI/models/tree/master/benchmarks/recommendation/tensorflow/wide_deep_large_ds#fp32-training-instructions

基于 MKL Threadpool* 的 Tensorflow*（可选）

TensorFlow 从 2.3.0 开始，增加了一个新的功能。TensorFlow 多线程支持可以选择使用 Eigen threadpool 而不是 OpenMP*，具体做法是在编译 Tensorflow 源代码时，使用编译选项 --config=mkl_threadpool 而不是之前的 --config=mkl。

如用户想在 TensorFlow 1.15 上尝试此功能，则需下载经过英特尔移植和优化的源代码，并进行编译（特别强调这里需要安装 Bazel* 0.24.1 ）：

```
# git clone https://github.com/Intel-tensorflow/tensorflow.git

# git checkout -b tf-1.15-maint remotes/origin/tf-1.15-maint

# bazel --output_user_root=$BUILD_DIR build --config=mkl_threadpool -c opt --copt=-O3
//tensorflow/tools/pip_package:build_pip_package

bazel-bin/tensorflow/tools/pip_package/build_pip_package $BUILD_DIR
```

上述步骤顺利完成后，TensorFlow wheel 文件可在 $BUILD_DIR 路径下找到。例如： tensorflow-1.15.0up2-cp36-cp36m-linux_x86_64.whl。安装步骤如下：

```
# pip uninstall tensorflow

# pip install $BUILD_DIR/&lt;filename&gt;.whl --user
```

使用深度学习框架 PyTorch*

部署 PyTorch*

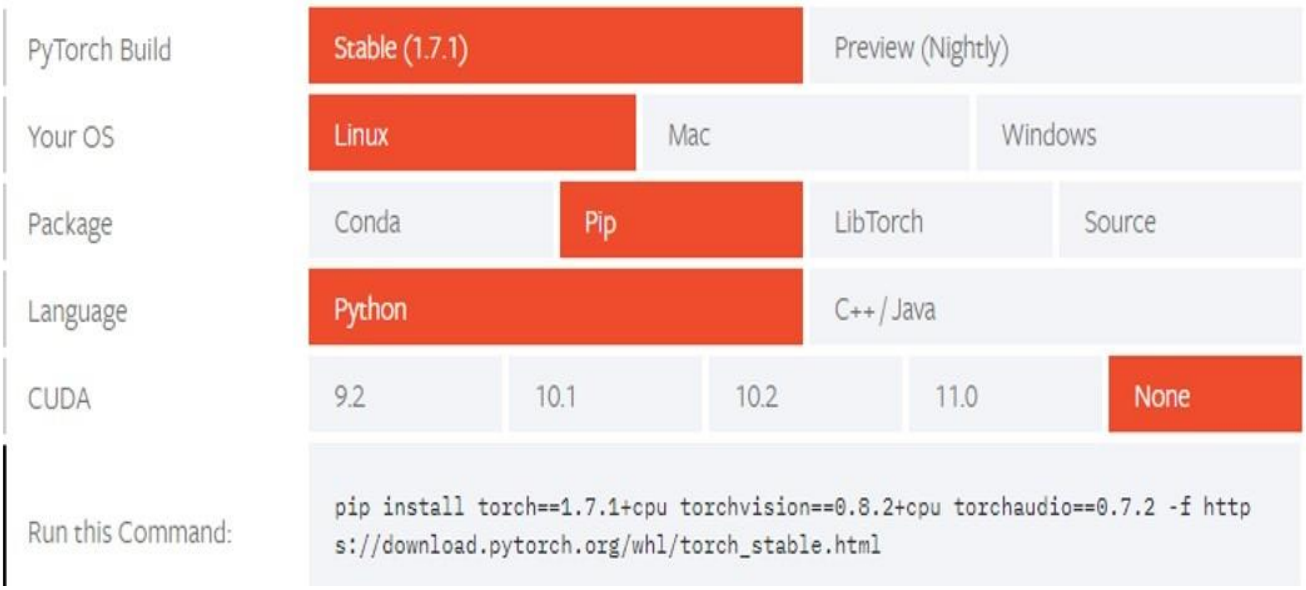
参考：https://www.intel.com/content/www/cn/zh/developer/articles/guide/getting-started-with-intel-optimization-of-pytorch.html

安装环境：python3.6 及以上的版本

第 1 步：访问 PyTorch 官网：https://pytorch.org/

第 2 步：选择 CPU

目前，英特尔 oneDNN 已经集成到 PyTorch 的正式版本中，因此无需额外的安装步骤就可以在英特尔® 至强® 可扩展处理器平台上获取加速性能，在选择 CUDA 的版本时。请将 CUDA 选项设置为 “None”。具体见下图。



第 3 步：安装

```
pip install torch==1.7.1+cpu torchvision==0.8.2+cpu torchaudio==0.7.2 -f
https://download.pytorch.org/whl/torch_stable.html
```

运行基于 PyTorch* 的深度学习模型的训练和推理的优化建议

您可参考以下网址在英特尔® 至强® 可扩展处理器平台上进行 PyTorch* 的优化参数设置

参考：https://www.intel.com/content/www/cn/zh/developer/articles/technical/how-to-get-better-performance-on-pytorchcaffe2-with-intel-acceleration.html

英特尔® Extension for PyTorch 介绍和使用

英特尔® Extension for PyTorch 是一个 PyTorch 的扩展 python 包，旨在提升 PyTorch 在英特尔® 至强® 处理器平台上的计算性能。这个扩展包不仅包括一些函数，还提供一些优化以加速英特尔新的硬件性能。

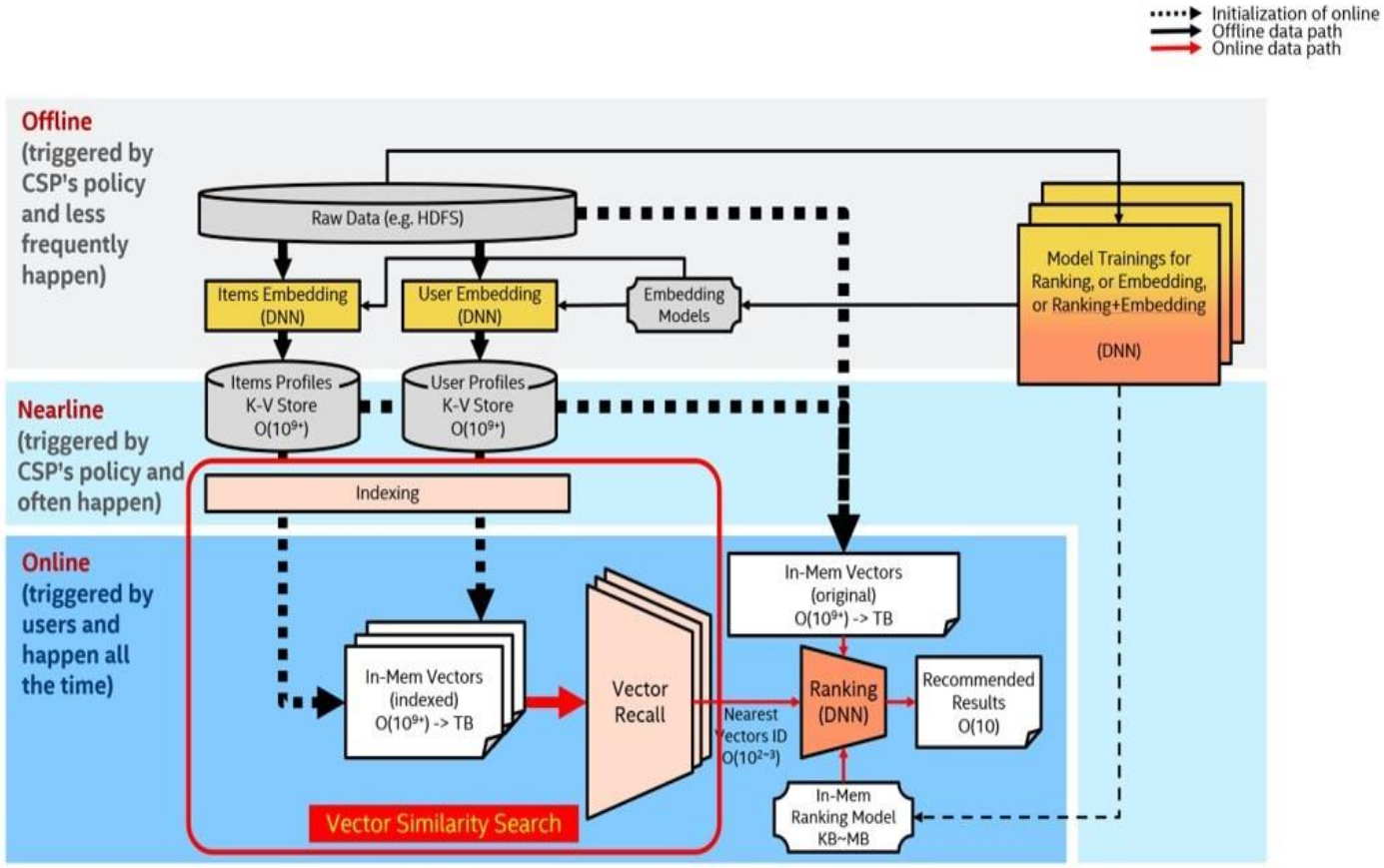
英特尔 Extension for PyTorch 的 github 链接如下：

https://github.com/intel/intel-extension-for-pytorch

https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/Features-and-Functionality/IntelPyTorch_Extensions_AutoMixedPrecision

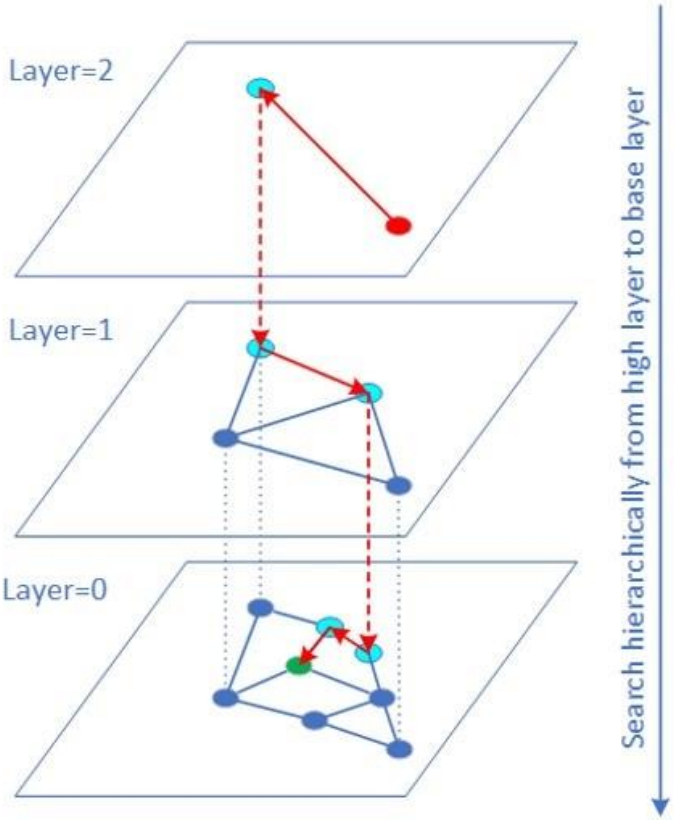
使用英特尔® 深度学习加速 VNNI 加速推荐系统中的矢量召回

推荐系统需要解决的问题是：如何为既定用户生成一个长度为 K 的推荐列表，并使该推荐列表尽量（高准确性）、尽快（低延迟）地满足用户的兴趣和需求？常规的推荐系统包含两部分：矢量召回（vector recall）和重排（ranking）。前者从庞大的推荐池里粗筛出当前用户最可能感兴趣的几百或几千条内容，并将结果交由后者的排序模块进一步排序，得到最终推荐结果。



矢量召回可以转换成高纬度的矢量相似性搜索问题。

HNSW (Hierarchical Navigable Small World) 算法是基于图结构的 ANN (Approximate Nearest Neighbor) 矢量相似度搜索算法之一。也是速度最快精度最高的算法之一。



矢量原始数据的数据类型常常是 FP32。对于很多业务（如图片检索），矢量数据是可以用 INT8/INT16 表示而且量化误差对最终搜集结果影响有限。这时可以使用 VNNI intrinsic 指令实现矢量 INT8/INT16 的内积计算。大量实验表明 QPS 性能有较大的提升，而且召回率几乎不变。QPS 提升的原因一方面是 INT8/INT16 访问带宽比 FP32 少很多，另一方面距离计算部分由于使用 VNNI 指令得以加速。

目前优化代码是基于 HNSWLib [1] 开源项目实现的。我们已经将它移植到业内广泛使用的 Faiss [2] 框架上。

为得到最佳性能，部署建议如下:

- 绑定 NUMA
- 每个 CPU 物理核运行一个查询进程

参考命令（以一个 socket 24 核为例）:

```
# numactl -C 0-23 <test_program>
```

当数据集比较大时（如 1 亿到 10 亿数据量级范围），传统的做法是将数据集切片，变成几个较小的数据集，每个数据集单独获取 topK，最后再合并。由于增加了多个机器之间的通信，增加延迟的同时降低了 QPS。我们在大数据集上使用 HNSW 方案的心得是：尽量不切片，在完整的数据集上建立索引和执行搜索，可获得最佳性能。当数据集过大，DDR 空间不够时，可以考虑使用英特尔® 傲腾™ 持久内存解决。

通过将 HNSW layer0 数据保存到 PMEM 上，可以显著扩大能支持的数据集的大小（单插槽可支持高达 40 亿条记录的 @d=100 的 INT8 数据库）。同时利用持久化特性避免大量数据的加载过程，使得初始化时间大为缩短。

AI 神经网络模型量化

AI 神经网络量化流程

神经网络的计算主要集中在卷积层和全连接层。这两层的计算都是可以表示为：Y = X * 权重 + 偏差。因此在做性能优化时也就自然而然地把焦点集中到了矩阵乘法上。神经网络模型量化的出发点是以有限的精度损失为代价换取性能上的提升。用低精度的整数代替 32 位浮点数做矩阵运算，不仅能够加速计算，同时也压缩了模型，节约了内存访问带宽。

神经网络模型量化有三种做法：

- 训练后量化，大多数人工智能框架均支持。
- 量化感知训练，在训练融合时将 FakeQuantization 节点插入到 FP32 模型中。增加了量化引入的噪声。在训练的反向传播中使模型权重落入一个有限区间，从而达到更好的量化后精度。
- 动态量化，与 PTQ 非常类似。都是训练后的量化。不同的是激活层的量化因子是神经网络模型在运行时根据数据范围动态决定的，而 PTQ 则是预先在小规模数据集上采样，获取激活层数据分布和范围信息，并永久记录在新生成的量化模型中。我们稍后将介绍的英特尔® AI Quantization Tools for TensorFlow* 中，目前只有 onnxruntime 可在后端支持此方法。

神经网络模型训练后量化的基本流程如下：

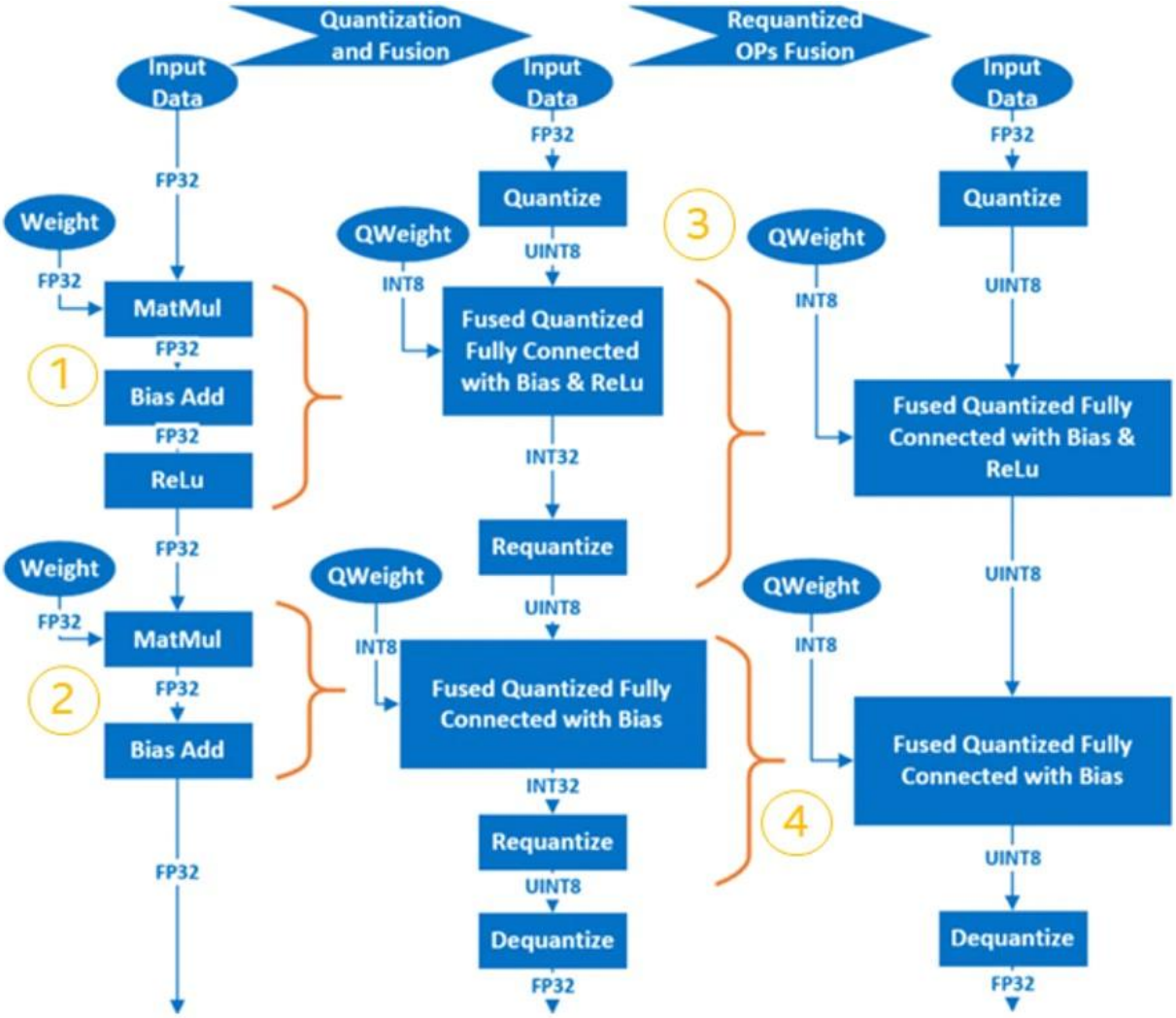
1.融合 FP32 OP 并转换为 INT8 OP。例如 MatMul、BiasAdd 和 ReLU 可以融合为单个量化的全连接层 OP，即 QuantizedMatMulWithBiasAndRelu。不同的神经网络框架所支持的可融合的 OP 不完全相同。稍后将介绍的英特尔® AI Quantization Tools for TensorFlow* 中，可以看到 TensorFlow 支持的可融合 OP 列表如下：
https://github.com/intel/lpot/blob/master/lpot/adaptor/tensorflow.yaml#L190。
pyTorch 支持的可融合 OP 请参阅：
https://github.com/intel/lpot/blob/master/lpot/adaptor/pytorch_cpu.yaml#L124

2.量化权重并保存在量化模型中。

3.量化输入/激活层，方法是在校准数据集上采样，获取激活层数据分布和范围信息，并记录在新生成的量化模型中。

4.将 Requantize 操作融合到其对应的 INT8 OP 中，以生成最终的量化模型。

以包含两层 MatMul 的简单模型为例，我们可以观察到量化的过程如下：



英特尔® Neural Compressor 介绍

你会借助英特尔®Neural compressor 实现自动化。它支持调优策略以快速找到最佳模型。 在通用框架上更快地使用此解决方案而不会降低效率。它是一个开源软件 Python 库，可在英特尔 CPU 和 GPU 上运行。 对于流行的网络压缩技术，比如 quantization, pruning, and knowledge distillation，我们可以用 Neural Compressor 和深度学习框架

参考网址：https://github.com/intel/neural-compressor

目前英特尔® Neural Compressor 支持以下英特尔优化的深度学习框架：

- Tensorflow*
- PyTorch*
- Apache* MXNet

- ONNX Runtime

当前已验证的各框架版本如下:

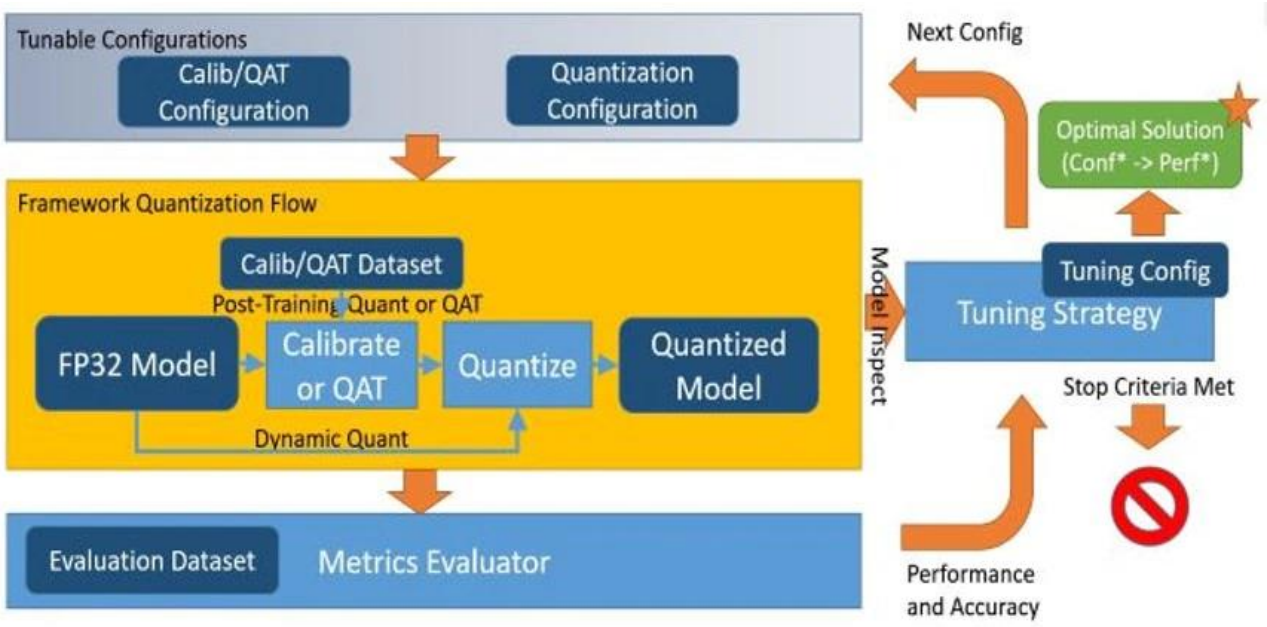
OS	Python	Framework	Version
CentOS 7.8	3.6	TensorFlow	2.2.0
Ubuntu 18.04	3.7		1.15.0 UP1
			1.15.0 UP2
			2.3.0
			2.1.0
			1.15.2
		PyTorch	1.5.0 + cpu
		Apache* MXNet	1.7.0
			1.6.0
		ONNX Runtime	1.6.0

英特尔® Neural Compressor 支持的调优策略有:

- Basic
- Bayesian
- MSE
- TPE
- Exhaustive

Random

Intel® Neural Compressor 工作流程是。根据调优策略自动选择与精度损失目标匹配的模型量化参数。然后生成量化模



型。

安装英特尔® Neural Compressor

安装详情请参阅：https://github.com/intel/lpot/blob/master/README.md

第 1 步：使用 Anaconda 创建名为 lpot 的 Python3.x 虚拟环境。这里我们使用 Python 3.7 为例：

```
# conda create -n lpot python=3.7


# conda activate lpot
```

第 2 步：安装 lpot；可选择以下 2 种安装方式：

从二进制文件安装：

```
# pip install lpot
```

从源代码安装：

```
# git clone https://github.com/intel/neural-compressor.git
```

```
# cd lpot

# pip install –r requirements.txt

# python setup.py install
```

运行英特尔® Neural Compressor

我们以 ResNet50 v1.0 为例，说明如何使用此工具进行量化。

准备数据集

第一步，下载并解压缩 ImageNet validation 数据集：

```
# mkdir –p img_raw/val &amp;amp;&amp;amp; cd img_raw

# wget http://www.image-net.org/challenges/LSVRC/2012/dd31405981ef5f776aa17412e1f0c112/ILSVRC2012_img_val.tar

# tar –xvf ILSVRC2012_img_val.tar -C val
```

第二步，把 image 文件移动进按标签分类的子目录：

```
# cd val

# wget -qO- https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.sh | bash
```

第 3 步：使用脚本 prepare_dataset.sh 将原始数据转化为 TFrecord 格式：

```
# cd examples/tensorflow/image_recognition
```

```
# bash prepare_dataset.sh --output_dir=./data --raw_dir=/PATH/TO/img_raw/val/ --subset=validation
```

参考网址： https://github.com/intel/lpot/tree/master/examples/tensorflow/image_recognition#2-prepare-dataset

准备模型

```
# wget https://storage.googleapis.com/intel-optimized-tensorflow/models/v1_6/resnet50_fp32_pretrained_model.pb
```

运行 Tuning:

编辑文件：examples/tensorflow/image_recognition/resnet50_v1.yaml，确保 quantization\calibration、evaluation\accuracy 和 evaluation\performance 的数据集路径是用户的真实本地路径。应为之前数据准备阶段生成的 TFrecord 数据的所在位置。

```
# cd examples/tensorflow/image_recognition/tensorflow_models/quantization/ptq

# bash run_tuning.sh --config=resnet50_v1.yaml \n
--input_model=/PATH/TO/resnet50_fp32_pretrained_model.pb \n
--output_model=./lpot_resnet50_v1.pb
```

参考网址： https://github.com/intel/lpot/tree/master/examples/tensorflow/image_recognition#1-resnet50-v10

运行 Benchmark:

```
# bash run_benchmark.sh --input_model=./lpot_resnet50_v1.pb --config=resnet50_v1.yaml
```

输出如下。性能数据仅供参考：

精确模式基准测试结果：

精确度为 0.739

批次大小 = 32

延迟：（结果可能会有不同）

吞吐量：（结果可能会有不同）

性能模式基准测试结果：

精确度为 0.000

批次大小 = 32

延迟：（结果可能会有不同）

吞吐量：（结果可能会有不同）

使用英特尔® 发行版 OpenVINO™ 工具套件进行推理加速

英特尔® 发行版 OpenVINO™ 工具套件

英特尔® 发行版 OpenVINO™ 工具套件官网和下载网站：

https://www.intel.com/content/www/cn/zh/developer/tools/openvino-toolkit/overview.html

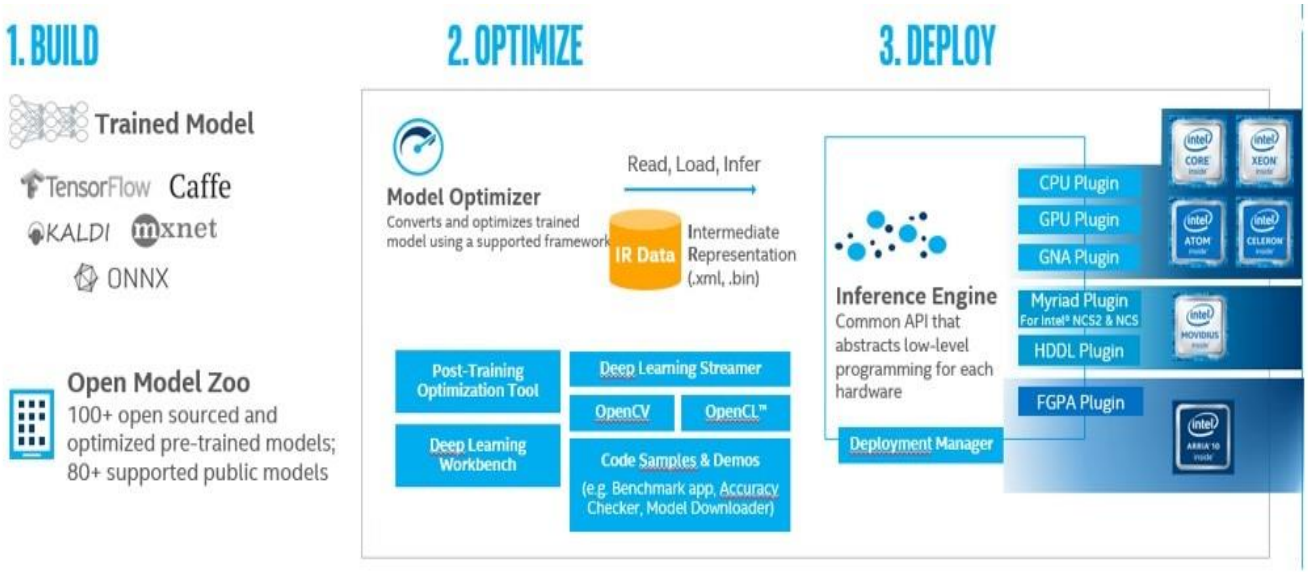
在线文档地址：

https://docs.openvino.ai/latest/index.html

在线中文文档地址：

https://docs.openvino.ai/cn/latest/index.html

英特尔® 发行版 OpenVINO™ 工具套件可用于加快计算机视觉和深度学习应用程序的开发。它能够支持各种加速器的深度学习应用程序，包括英特尔® 至强® 处理器平台上的 CPU、GPU、FPGA 以及英特尔® Movidius™ CPU，同时能够直接支持异构的执行。

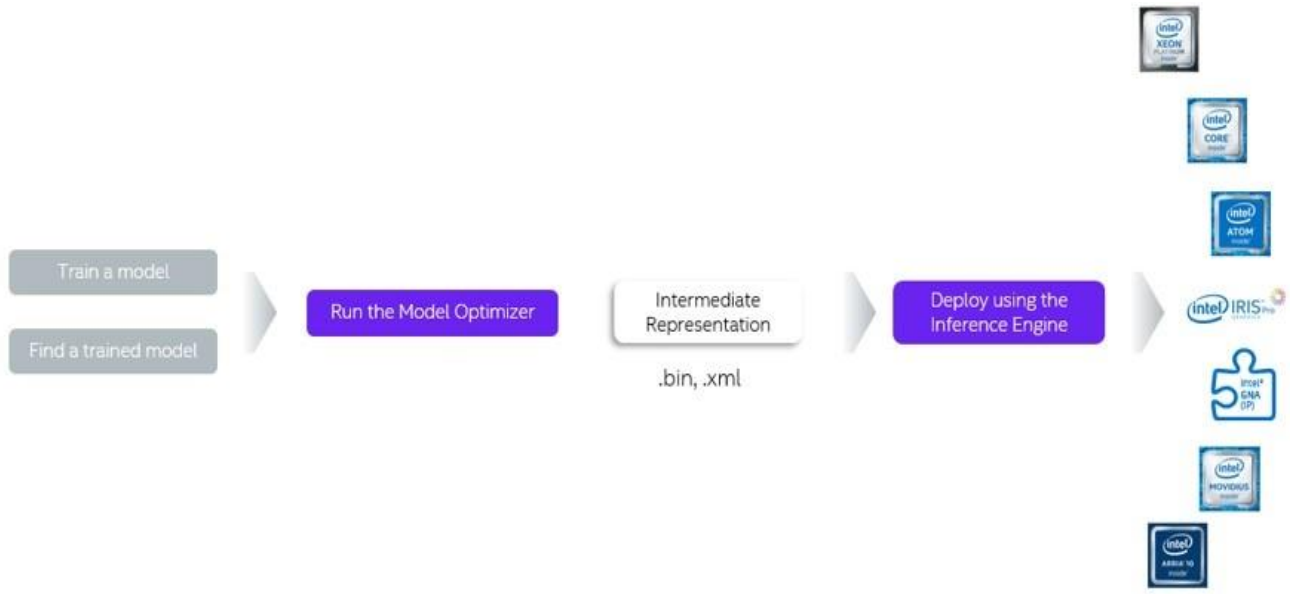


英特尔® 发行版 OpenVINOTM 工具套件旨在提高计算机视觉处理及深度学习推理解决方案的性能，并缩短开发时间。包括计算机视觉和深度学习开发套件两部分。

Deep Learning Deployment Toolkit (DLDT) 是一个加速深度学习推理性能的跨平台工具，包括：

- ●模型优化器：将通过 Caffe*、TensorFlow、Mxnet 及其他框架训练的模型转化为中间表示 (IR)。
- ●推理引擎：在 CPU、GPU、FPGA、VPU 及其他硬件上执行 IR。自动调用硬件的加速套件实现推理性能的加速。

英特尔® 发行版 OpenVINOTM 工具套件的工作流程：



部署英特尔® 发行版 OpenVINO™ 工具套件

请参考中文安装文档：

安装适用于 Linux* 的英特尔® 发行版 OpenVINO™ 工具套件：

使用英特尔® 发行版 OpenVINO™ 工具套件的 Deep Learning Deployment Toolkit (DLDT) 英特尔® 深度学习部署工具套件简介

图像分类 C++ 示例 (Async)

对象检测 C++ 示例 (SSD)

自动语音识别 C++ 样本

动作识别 Python* 演示

十字路口摄像头 C++ 演示

人体姿态估计 C++ 演示

交互式人脸检测 C++ 演示

使用英特尔® 发行版 OpenVINO™ 工具套件加速 INT8 的推理

使用 INT8 精度的模型运行推理，并通过英特尔® 至强® 可扩展处理器平台的英特尔® 深度学习加速进行加速，可以大幅提升推理效率。同时节约模型推理的计算资源和电力消耗。OpenVINO 的 2020 版本及之后的版本都提供 INT8 量化工具支持 FP32 模型的量化。

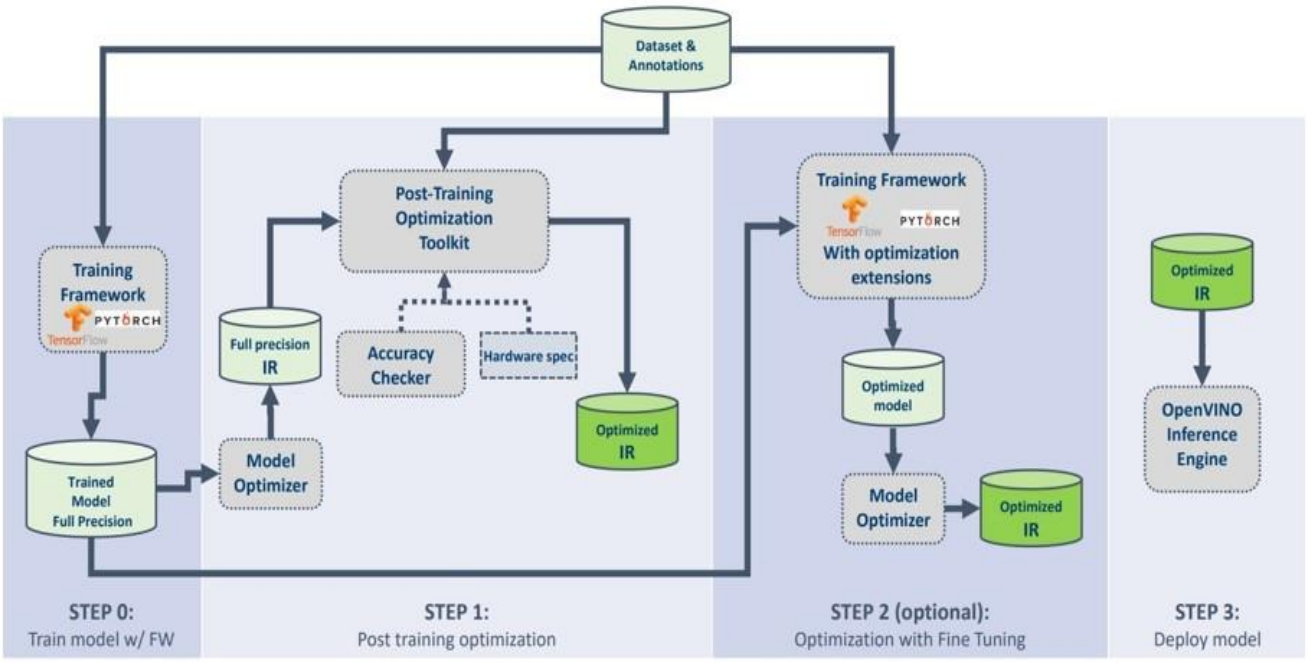
OpenVINO 提供的 INT8 模型量化工具是一种训练后优化工具套件 (POT)，用于优化和量化训练后的模型。无需对模型进行重新训练或者微调，无需更改模型结构。下图展示了通过 OpenVINO 对新模型进行优化的主要流程。

步骤 0：为获取训练后的模型，

步骤 1：为 POT 的组成和优化步骤，

步骤 2：为可选操作（根据实际情况选择是否对模型进行微调，达到更好的准确度），以及

步骤 3：是通过 OpenVINO 的 IE 对模型进行推理。



POT 提供一个独立的命令行工具和 Pythoh AIP，主要支持以下特性：

两种类型的训练后 INT8 量化算法：快速的 DefaultQuantization 和精准的 AccuracyAwareQuantization。 使用树状结构 Parzen 估算器进行训练后量化参数全局优化 支持对称和非对称量化方案 支持多种硬件平台（CPU，GPU）的压缩 量

化卷积层和全连接层的每个通道 支持多个领域：计算机视觉，推荐系统 通过提供的 API 提供客户化的优化方法 具体操作和使用请参考以下网址：

训练后优化工具套件介绍

低精确度优化指南

[POT 最佳实践] (https://docs.openvino.ai/latest/pot_docs_BestPractices.html)

[POT 常见问题] (https://docs.openvino.ai/latest/pot_docs_FrequentlyAskedQuestions.html)

[通过 DL Workbench 的 web 界面进行 INT8 量化优化]
(https://docs.openvino.ai/latest/workbench_docs_Workbench_DG_Int_8_Quantization.html)

使用英特尔® DAAL 加速机器学习

英特尔® Data Analytics Acceleration Library（英特尔® DAAL）

作为人工智能的一个分支，机器学习现在正获得极大的关注。基于机器学习的高级分析也越来越流行。其原因在于，与其它分析方法相比，机器学习能够帮助 IT 人员、数据科学家、各种业务团队及其组织迅速释放优势。并且机器学习提供了许多新的商用开源解决方案，为开发人员提供了一个丰富的生态系统。此外，开发人员可以从各种各样的开源机器学习库中进行选择，如 Scikit-learn、Cloudera* 和 Spark* MLlib。

英特尔® Distribution for Python 介绍 英特尔® Distribution for Python 是一个 Python 开发工具包，面向人工智能软件的开发人员。可以加速 Python 在英特尔® 至强® 可扩展处理器平台的计算速度。 可通过 Anaconda* 获取，也可以与 Conda*、PIP*、APT GET、YUM、Docker* 等一起安装使用。参考、下载网址：
https://www.intel.com/content/www/cn/zh/developer/tools/oneapi/distribution-for-python.html

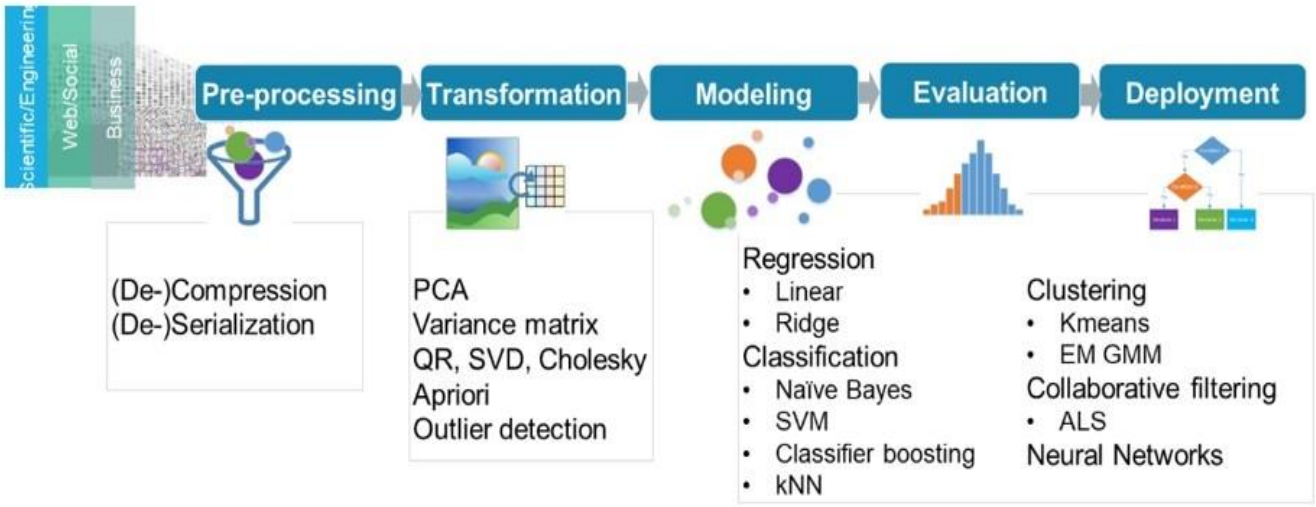
英特尔® Distribution for Python* 的特点：

- 开箱即用：只需很少或根本不需要更改代码，即可获得更快的 Python 应用程序性能。
- 借助集成的英特尔® 性能库（例如英特尔® 数学核心函数库和英特尔® Data Analytics Acceleration Library）来加速 NumPy，SciPy 和 scikit-learn*
- 结合最新的向量化和多线程指令，Numba* 和 Cython，提升并行化和向量化效率。

英特尔® DAAL

英特尔® Data Analytics Acceleration Library (英特尔® DAAL) 面向数据科学家，旨在加快数据分析和预测效率，尤其是海量数据。可充分利用向量化和多线程等优化技术，提升机器学习在英特尔平台上的整体性能。

英特尔® DAAL 是可以帮助数据科学家和分析师，快速建立从数据预处理，到数据特征工程、数据建模和部署的一整套端到端软件方案。它提供了建立机器学习和分析所需的各种数据分析及算法所需的高性能构建模块。目前已经支持线性回归、逻辑回归、LASSO、AdaBoost，贝叶斯分类器、支撑向量机、K 近邻、Kmeans 聚类、DBSCAN 聚类、 各种决策树、随机森林、 Gradient Boosting 等经典机器学习算法。这些算法经过高度优化，可在英特尔® 至强® 可扩展处理器上实现高性能。如中国一家领先的大数据分析技术和服务提供商，使用这些资源已将多个数据挖掘算法提高了数倍。



为了使开发人员在基于英特尔技术的环境下，在机器学习应用程序中更轻松地使用英特尔® DAAL，英特尔开源了整个项目 (https://github.com/intel/daal)，并针对不同的大数据使用场景提供了全内存流式和分布式的算法支持。比如 DAAL Kmeans 可以很好和 Spark 结合，在 Spark 集群上进行多节点聚类。另外，英特尔® DAAL 提供了 C++、Java 和 python 接口。

DAAL4py

为了更好地支持 python 应用最为广泛的 Scikitlearn, 英特尔® DAAL 提供了非常简便的 Python 接口 DAAL4py（更多详细信息请参见开源网站：https://github.com/IntelPython/daal4py ）。它可以和 Scikitlearn 无缝的结合，在底层提供机器学习的算法加速。

开发者可以无需修改 Scikitlearn 代码，即可利用自动向量化和多线程化的优势。目前 DAAL4py 在 Scikitlearn 中支持算法有：

- Sklearn 线性回归、Sklearn 岭回归和逻辑回归
- PCA
- KMeans
- pairwise_distance
- SVC (SVM 分类)

安装英特尔® Distribution for Python* 和英特尔® DAAL

下载和安装英特尔® Distribution for Python*（已包含 英特尔® DAAL)* (Intel® DAAL already included)

单独安装英特尔® DAAL

英特尔® DAAL 开发者指南

使用英特尔® DAAL

使用英特尔® DAAL 加速 scikit-learn，有两种方式

方法 1：命令行方式

```
# python -m daal4py &lt;your-scikit-learn-script&gt;
```

方法 2：在代码中加入

```
import daal4py.sklearn


daal4py.sklearn.patch_sklearn('kmeans')
```

参考资料

[1] HNSWLib 项目开源资源：https://github.com/nmslib/hnswlib

[2] Open source of Faiss project: https://github.com/facebookresearch/faiss

Additional References

Intel® AVX-512 info: https://colfaxresearch.com/skl-avx512/

Intel® Optimized AI Frameworks:
https://www.intel.com/content/www/us/en/developer/tools/frameworks/overview.html

Intel® Distribution of OpenVINO™ toolkit: https://docs.openvino.ai

Intel® Analytics Zoo: https://github.com/intel-analytics/analytics-zoo

通知和免责声明

英特尔技术可能需要支持的硬件、软件或服务激活。

没有任何产品或组件能保证绝对安全。

您的成本和结果可能会有所不同。

英特尔使用代号来标识正在开发且未公开的产品、技术或服务。这些并非“商用”名称，也不会被用作商标

所述产品可能包含设计缺陷或错误（即勘误表），这可能会使产品与已发布的规范有所偏差。可应要求提供当前的勘误表。