IXPUG Workshop at HPC Asia 2021

Distributed MLPerf ResNet50 Training on Intel Xeon Architectures with TensorFlow

Wei Wang, Niranjan Hasabnis (Intel Corporation)

01/22/2021

intel®

Introduction & Background

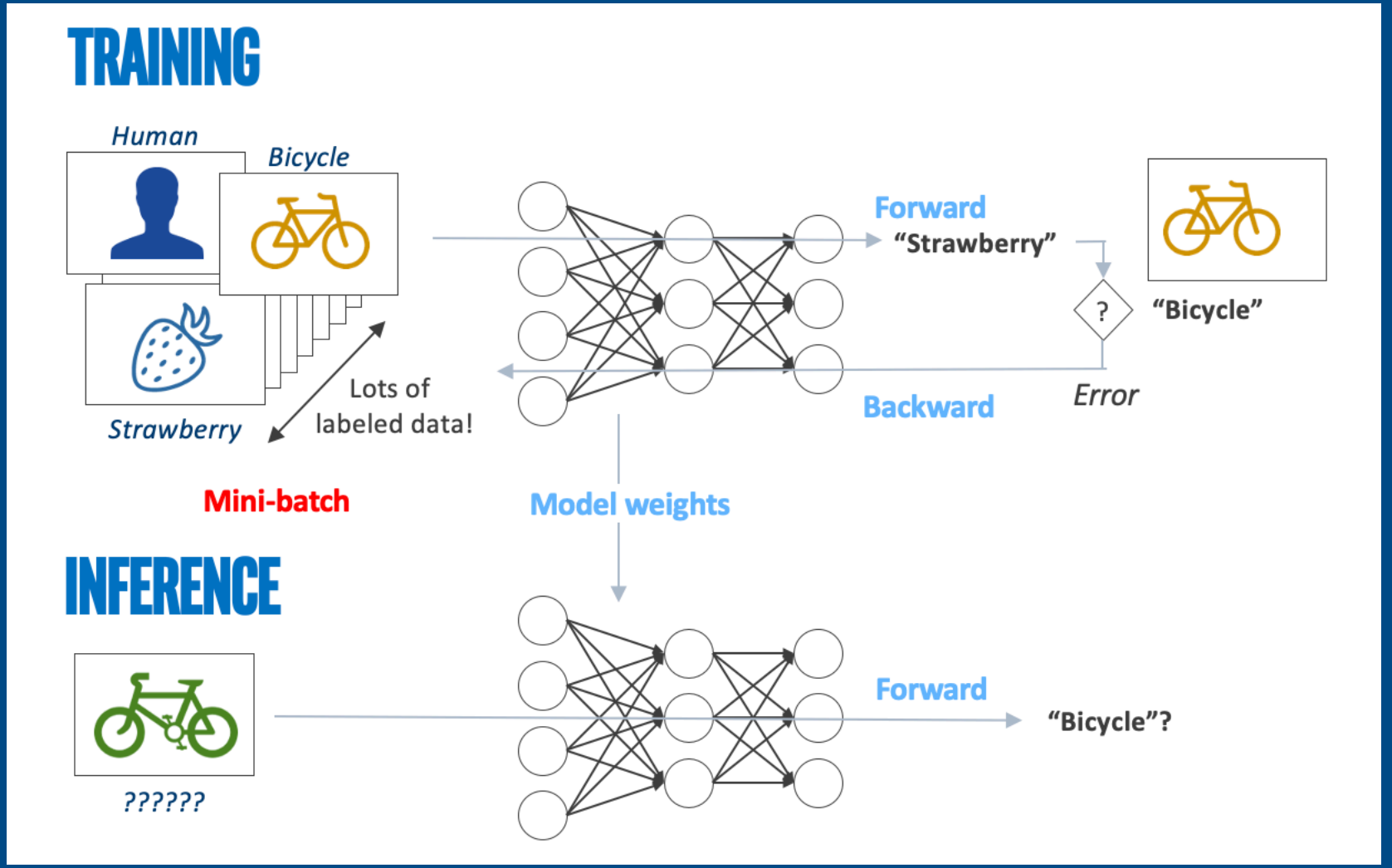
MLPerf/MLCommons, Deep Learning, and Intel-Optimized TensorFlow

MLPerf Training Benchmarks

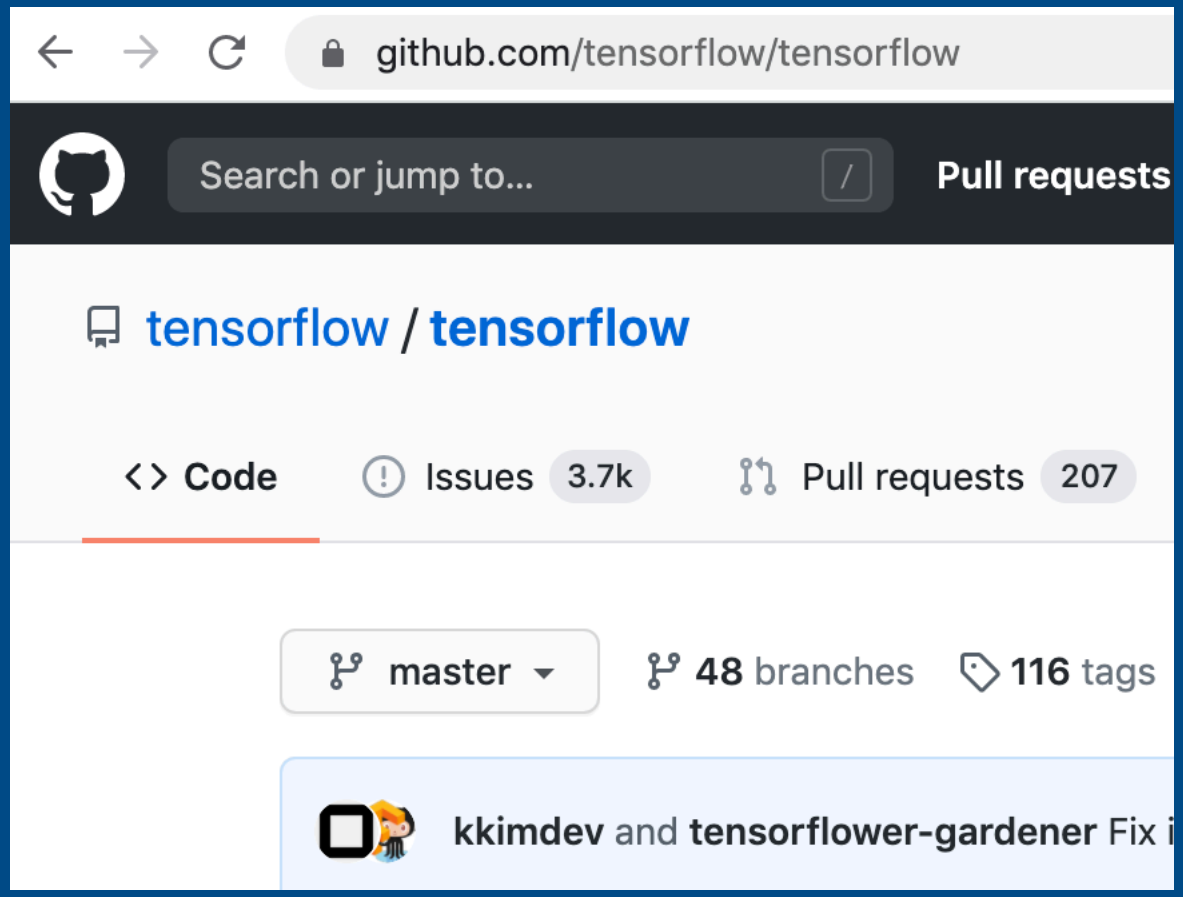
Area	Benchmark	Dataset	Quality Target	Reference Implementation Model
Vision	Image classification	ImageNet	75.90% classification	ResNet-50 v1.5
Vision	Object detection (light weight)	COCO	23.0% mAP	SSD
Vision	Object detection (heavy weight)	COCO	0.377 Box min AP and 0.339 Mask min AP	Mask R-CNN
Language	Translation (recurrent)	WMT English-German	24.0 Sacre BLEU	NMT
Language	Translation (non-recurrent)	WMT English-German	25.00 BLEU	Transformer
Language	NLP	Wikipedia 2020/01/01	0.712 Mask-LM accuracy	BERT
Commerce	Recommendation	1TB Click Logs	0.8025 AUC	DLRM
Research	Reinforcement learning	Go	50% win rate vs. checkpoint	Mini Go (based on Alpha Go paper)

Source: https://mlcommons.org/en/training-normal-07/

Image Classification Training & Inference



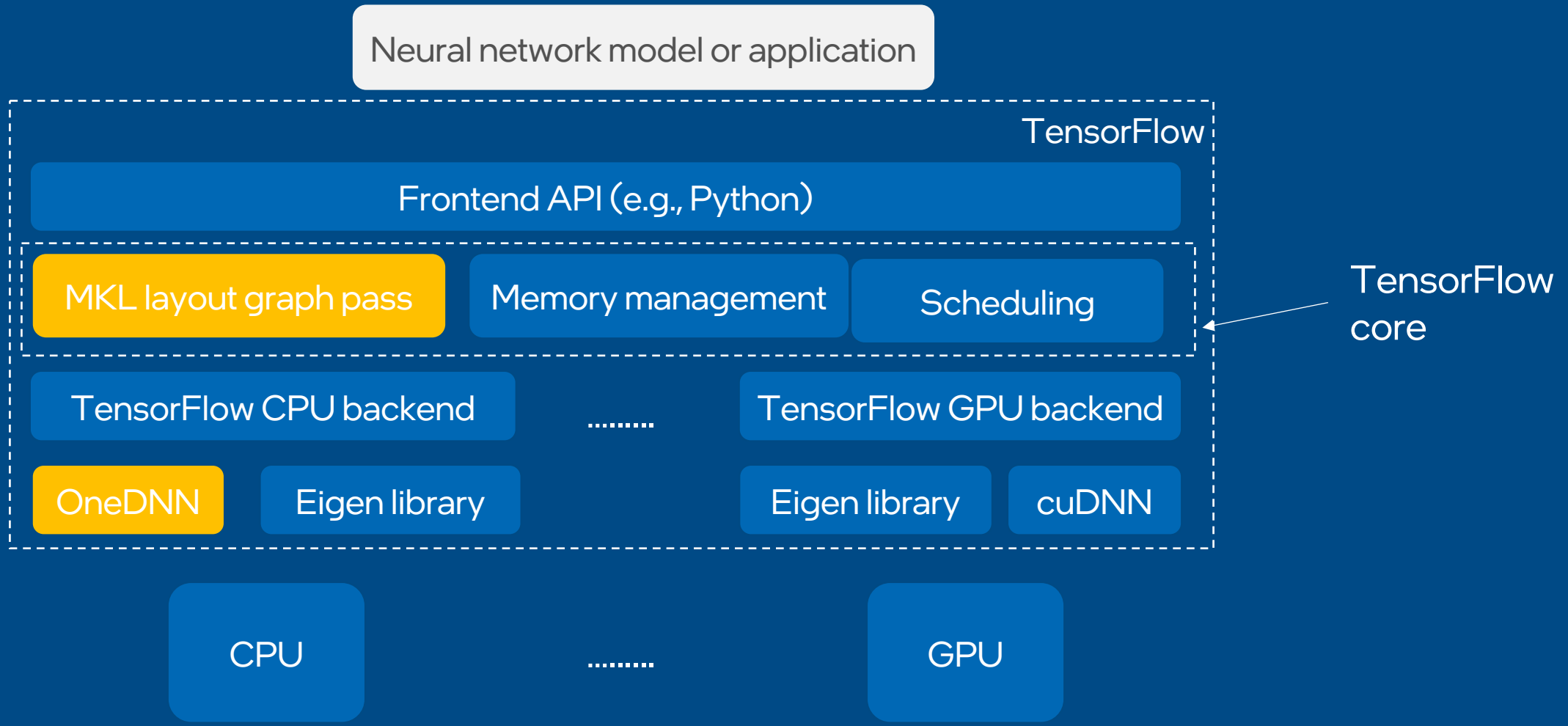
Intel Optimized TensorFlow



Intel optimizations are part of public TensorFlow github repo

- PyPI Wheels
  - `pip install intel-tensorflow`
- Docker Hub
  - `docker pull intel/intel-optimized-tensorflow`
- Anaconda
  - `conda install tensorflow -c intel`
- Build from Source
  - `bazel build --config=mkl …`

oneAPI Deep Neural Network Library (oneDNN) + TensorFlow



BFloat16 Enablement and Distributed Training

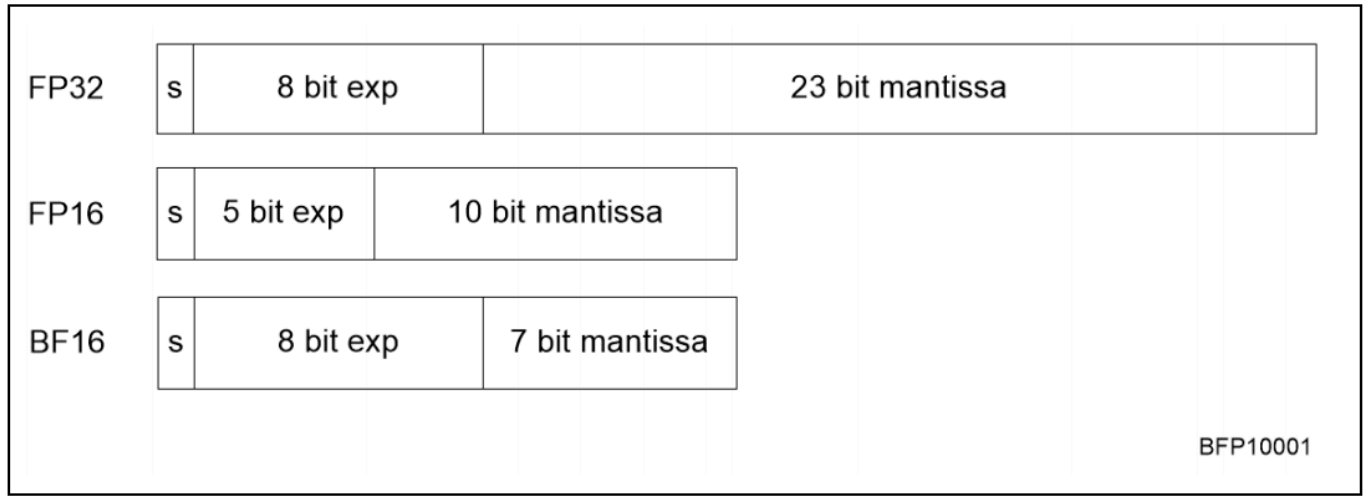
Intel Optimized TensorFlow with BFloat16 + Horovod

BFloat16 DataType

1.1 Bfloat16 Floating-point Format

Intel® Deep Learning Boost (Intel® DL Boost) uses bfloat16 format (BF16). Figure 1-1 illustrates BF16 versus FP16 and FP32.

Figure 1-1. Comparison of BF16 to FP16 and FP32.

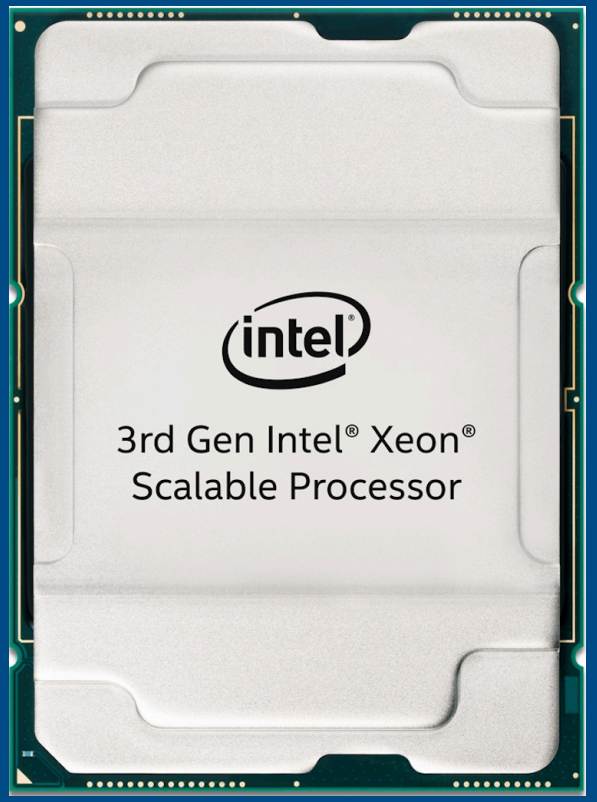


BFloat16: same range as FP32, precision not as high, does not affect Deep Learning training accuracy, speeds up calculation, saves memory bandwidth

https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numerics-definition-white-paper.pdf

3rd Generation Intel Xeon Scalable Processors

- Enhanced Intel Deep Learning Boost (Intel DL Boost) with VNNI (Vector Neural Network Instructions) and new bfloat16 (Brain Floating Point 16-bit) support
- VCVTNE2PS2BF16 — Convert Two Packed Single Data to One Packed BF16 Data
- VCVTNEPS2BF16 — Convert Packed Single Data to Packed BF16 Data
- VDPBF16PS — Dot Product of BF16 Pairs Accumulated into Packed Single Precision



Integrating BFloat16 oneDNN API in TensorFlow's MKL backend

```
namespace tensorflow {
typedef Eigen::ThreadPoolDevice CPUDevice;

template <typename Device, typename T>
class MklAddNOp : public OpKernel {
 public:
  ~MklAddNOp() {}
  explicit MklAddNOp(OpKernelConstruction* context) : OpKernel(context) {}
```

Operator kernel definition in TensorFlow MKL backend

```
#define REGISTER_MKL_CPU(T)                                       \
  REGISTER_KERNEL_BUILDER(Name("_MklAddN")                        \
                              .Device(DEVICE_CPU)                 \
                              .TypeConstraint<T>("T")             \
                              .Label(mkl_op_registry::kMklOpLabel), \
                          MklAddNOp<CPUDevice, T>);

TF_CALL_float(REGISTER_MKL_CPU);
+ TF_CALL_bfloat16(REGISTER_MKL_CPU);
#undef REGISTER_MKL_CPU
}  // namespace tensorflow
```

Registering kernel for different types

C++ templates simplified the enabling process.

ResNet50 model changes for BFloat16

1. Use BFloat16 for activations and gradients, keep weights in FP32

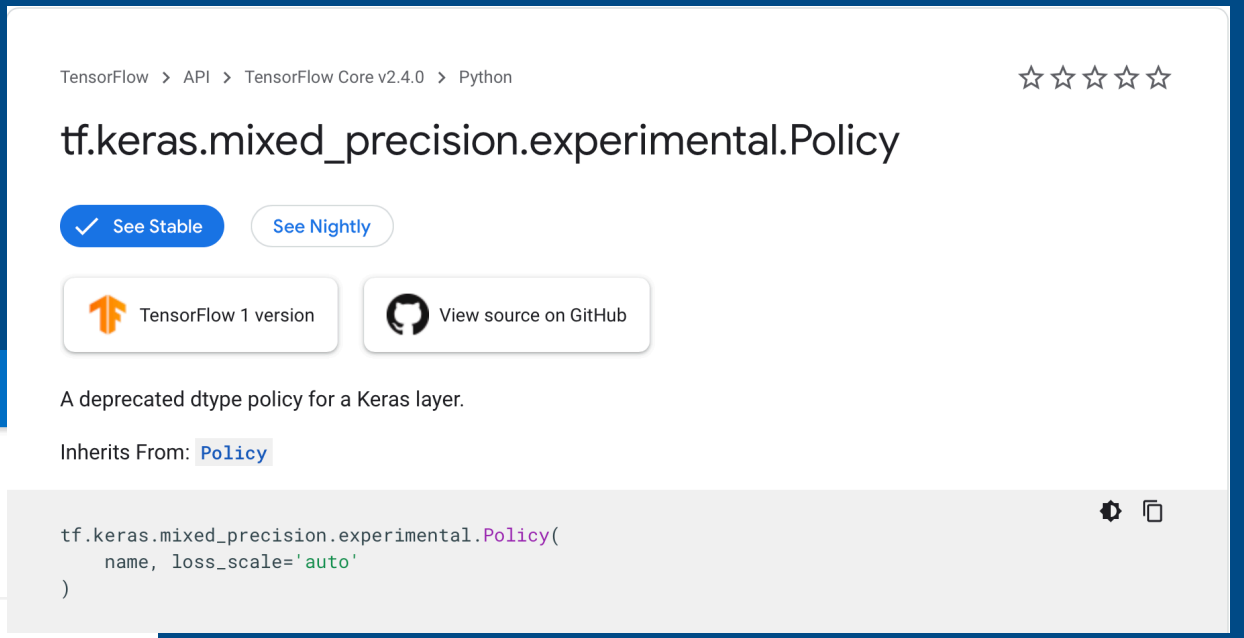
```
if dtype in CASTABLE_TYPES:
   var = getter(name, shape, tf.float32, *args, **kwargs)
   return tf.cast(var, dtype=dtype, name=name + '_cast')
else:
   return getter(name, shape, dtype, *args, **kwargs)
```

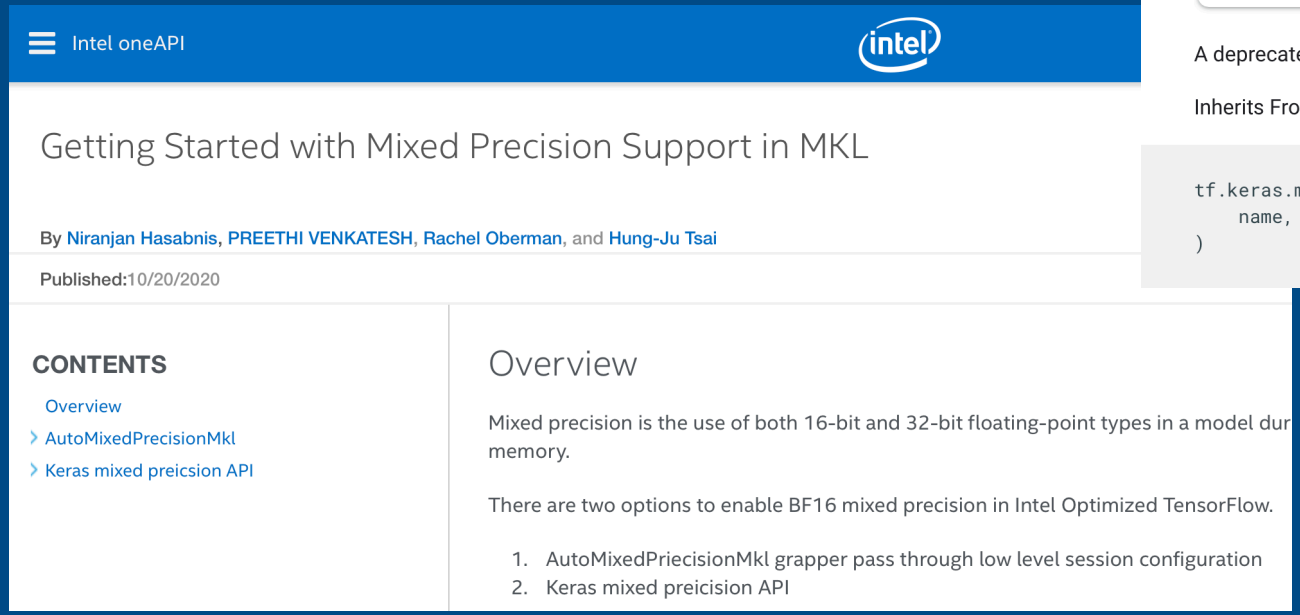
Manual changes

2. Convert input images from FP32 to BFloat16 using tf.cast()

```
# features: tensor representing input images
if use_bfloat16 == True:
   dtype = tf.bfloat16
features = tf.cast(features, dtype)
```

AutoMixedPrecision: enable BFloat16 in a model automatically using TensorFlow's Keras API





AutoMixedPrecision works with TensorFlow's MKL backend also.

Distributed Intel TensorFlow (Intel Optimized TensorFlow + Horovod)

- Horovod (created by Uber), communication library sits on top of TensorFlow, efficient Ring-Allreduce gradient communication
- No source code changes within TF
- Minimal amount of model instrumentation
- Good scalability
- Good alternative to native distributed strategy within TensorFlow

ResNet50v1.5 Model Changes for Distributed Training

```
import horovod.tensorflow as hvd
# initialize Horovod
hvd.init()

Global_batch_size = params['batch_size'] * hvd.size()

dataset = dataset.shard(hvd.size(), hvd.rank())

optimizer = hvd.DistributedOptimizer(optimizer)

steps_per_epoch_per_worker = steps_per_epoch // hvd.size()

hvd.BroadcastGlobalVariablesHook(0)
```

Distributed Inference With Horovod

Evenly distribute validation dataset (TFRecord files) among Horovod workers, explicit allreduce call to aggregate results

```
    files_per_worker = 128//hvd.size()
    return [
        os.path.join(data_dir, 'validation-%05d-of-00128' % (i+files_per_worker*hvd.rank()))
        for i in range(files_per_worker)]
```

```
eval_classifier = tf.estimator.Estimator(
    model_fn=model_function, model_dir=model_dir.rsplit('/', 1)[0]+'/main', config=run_config,
```

```
eval_results = eval_classifier.evaluate(input_fn=input_fn_eval,
                                        steps=flags.max_train_steps)
eval_results_per_worker = eval_results['accuracy']
allreduced_results = hvd.allreduce(eval_results_per_worker)
```

Typical Distributed TensorFlow Environment Setup

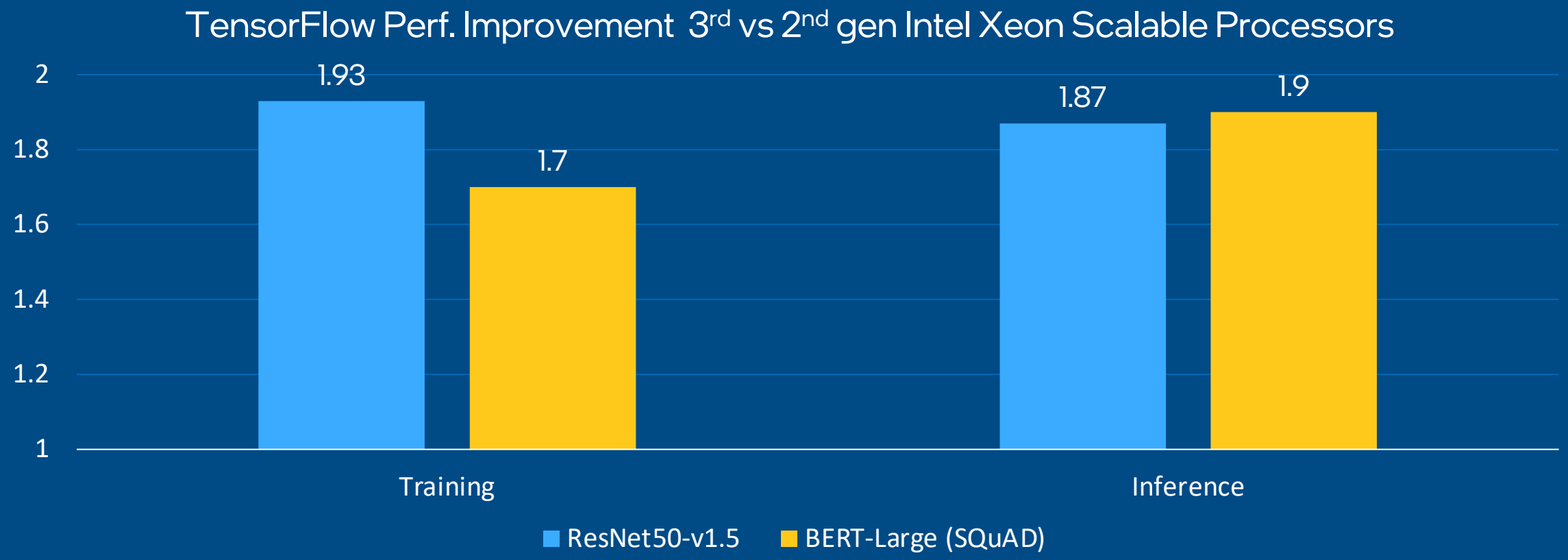
1. Install OpenMPI or IntelMPI
2. virutalenv -p /usr/bin/python3 Intel-TF-venv
3. . ./Intel-TF-venv/bin/activate
4. pip install intel-tensorflow==2.3.0
5. pip install --no-cache-dir horovod==0.19.1 or latest
6. Run distributed model: mpirun --map-by socket -n number_of_sockets*number_of_nodes –H hostlist python resnet_main.py … (or --bind-to socket -map-by slot)

Experimental Results

Single-Node, Multi-Socket Benchmarking & Convergence

BFloat16 vs. FP32 Performance Benefits

BFloat16 on 3rd gen Intel Xeon Processors accelerates ResNet50 performance by up to 1.93x



Reference: https://blog.tensorflow.org/2020/06/accelerating-ai-performance-on-3rd-gen-processors-with-tensorflow-bfloat16.html

TensorFlow ResNet50 Training on 3rd Gen Intel Xeon Scalable Processors

Repeated Runs	Time[1] To Train (Baseline[2])	Time To Train (Improved)	Total Evaluation Time (Baseline)	Total Evaluation Time (Improved)
1	1146.58	1097.36	52.67	4.98
2	1145.04	1100.02	52.55	4.98
3	1144.91	1098.05	52.58	4.97
4	1145.83	1098.82	52.84	4.97
5	1148.02	1100.73	53.21	4.96
Olympic Average	1145.82	1098.97	52.70	4.97

[1]All time shown in minutes and with BF16 precision
[2]MLPerf v0.7 Training Closed ResNet-50, Retrieved from https://mlcommons.org/en/training-normal-07/, 19 January 2021, entry 0.7-63. MLPerf name and logo are trademarks. See www.mlperf.org for more information.

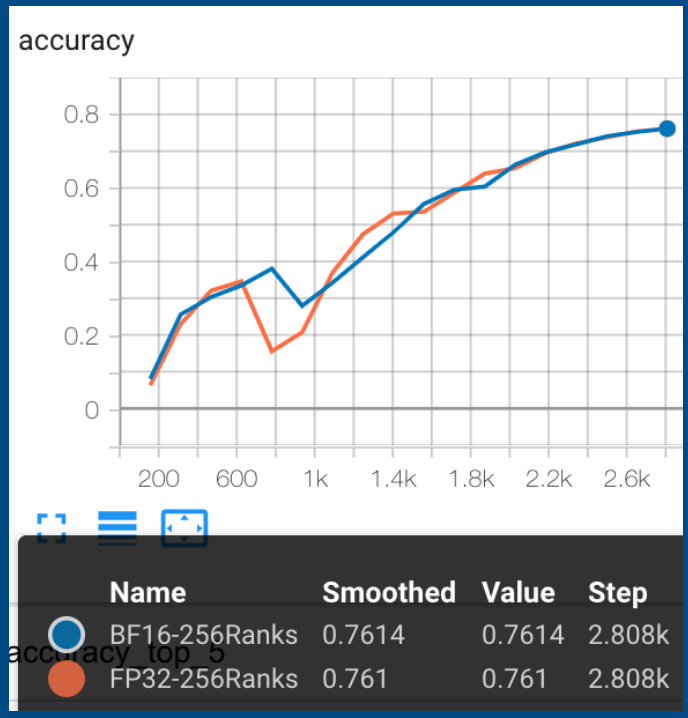
Experimental Results

Multi-Node Training Convergence

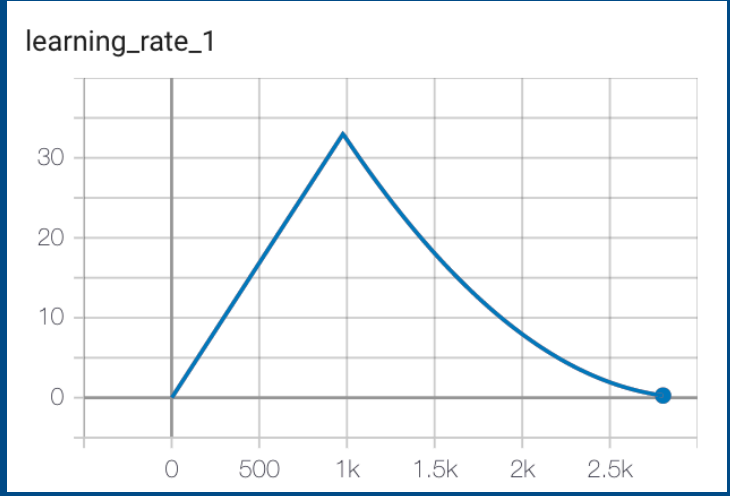
Large Batch Size Distributed Training (BFloat16 & FP32)

Top-1 accuracy with BFloat16 shows similar trend as FP32.

- Batch Size = 32K (128 * 256)
  - LARS (Layerwise Adaptive Rate Scaling)
  - Scaled Learning Rate: 33
  - Warmup Epochs: 25 (~1K steps)
  - Mini Batch Size: 128
  - Number of workers (ranks): 256
  - Number of machines: 64 Intel Xeon Platinum 9242 (FP32) , 128 Intel Xeon Platinum 8260L (BF16)
  - Number of sockets per machine: 2
  - Number of cores per socket: 48 (FP32), 24 (BF16)
  - Convergence: 72 epochs



Top-1 Accuracy



Learning Rate Schedule

Conclusion

- Intel-Optimized TensorFlow w/ oneDNN
- Bfloat16 data type and enablement
- Distributed TensorFlow training with Horovod
- 3rd Gen Intel Xeon Scalable Processors
- Latest Layer-wise Adaptive Rate Scaling (LARS) ResNet50v1.5 Optimizer

A successful MLPerf training v0.7 ResNet50 submission from Intel!

Questions?


intel

Appendix/Backup

Steps to try RN50 Training (FP32 & BF16) with Intel Optimized TensorFlow

https://github.com/mlperf/training_results_v0.7/tree/master/Intel/benchmarks/resnet/1-node-8s-cpx-1-tensorflow

Method 2: docker container approach (recommended because OpenMPI and Horovod has already been setup in the docker)

- Download the container Intel has released a docker container featuring BF16 integration and ResNet50 lars optimizer in TensorFlow and is available via the following command.

```
docker pull intel/intel-optimized-tensorflow:tensorflow-2.2-bf16-rn50-nightly
```

- MLPerf logging utility The TF ResNet50 code requires mlperf logging repo. Please make sure the following step is performed under the directory containing this README.md file.

```
git clone https://github.com/mlperf/logging.git
git checkout 71b2a076e9319c2dedb635dd8a34ef71e3a455e5
```

- Launch the container

After the above docker image is pulled and mlperf logging repo is downloaded, you can launch the docker container using a command similar to the following:

```
docker run -it --privileged  --name=mlperf-rn50  -v /dataset/:/dataset -v /MLPerfv0.7-code-base/:/workspace {docker_image_
```

The /dataset refers to the imagenet dataset, the MLPerfv0.7-code-base refers to the MLPerf training v0.7 code. The docker image id can be found by "docker images" command.

With the above steps done, TF+MPI+Horovod environment should be setup correctly. Note: the above "--privileged" is needed to avoid seeing the issue similar to "https://github.com/horovod/horovod/issues/653". If you do not have the privilege, suggest trying Method 1, i.e., running on bare-metal or without the container.

- Run the workload The code is tested on an 8S Xeon platform having 28 cores in each socket. You would need to configure the OpenMP related parameters if you are not using identical platforms. Also, you need to modify the related path in `run\_and\_time.sh` before you run the code.

Trying with general models with Intel Optimized TensorFlow

- pip install intel-tensorflow（in future: pip install tensorflow）
- Intel Model Zoo: https://github.com/intelai/models
- Container Portal: https://software.intel.com/content/www/us/en/develop/tools/containers/get-started.html