DEEP LEARNING AT SCALE USING DISTRIBUTED FRAMEWORKS

DR. FABIO BARUFFA

SENIOR TECHNICAL CONSULTING ENGINEER, INTEL IAGS

Legal Disclaimer & Optimization Notice

Performance results are based on testing as of September 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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.

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

LEGAL NOTICES & DISCLAIMERS

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo, Pentium, Celeron, Atom, Core, Xeon, Movidius and others are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

© 2019 Intel Corporation.

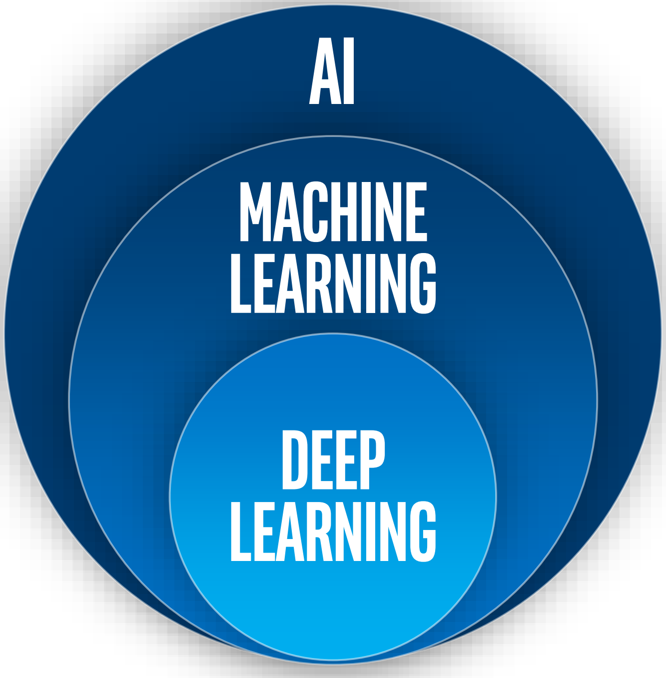
Outline

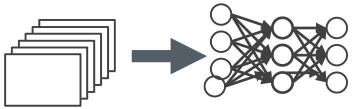
- Introduction to Intel® DL Boost
- Intel® AI optimized frameworks
- Integration with the popular AI/ML frameworks:
  - Tensorflow accelerated with Intel® MKL-DNN
  - Intel® Machine Learning Scaling Library
  - Horovod
- Hands-on

INTRODUCTION

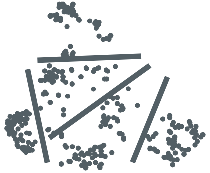
WHAT IS AI?

Regression
Classification
Clustering
Decision Trees
Data Generation
Image Processing
Speech Processing
Natural Language Processing
Recommender Systems
Adversarial Networks
Reinforcement Learning





SUPERVISED LEARNING



UNSUPERVISED LEARNING

REINFORCEMENT LEARNING

No one size fits all approach to AI

MACHINE VS. DEEP LEARNING

MACHINE LEARNING

How do you engineer the best features?

$N \times N$

$(f_1, f_2, \dots, f_K)$

Roundness of face
Dist between eyes
Nose width
Eye socket depth
Cheek bone structure
Jaw line length
...etc.

CLASSIFIER ALGORITHM

SVM
Random Forest
Naïve Bayes
Decision Trees
Logistic Regression
Ensemble methods

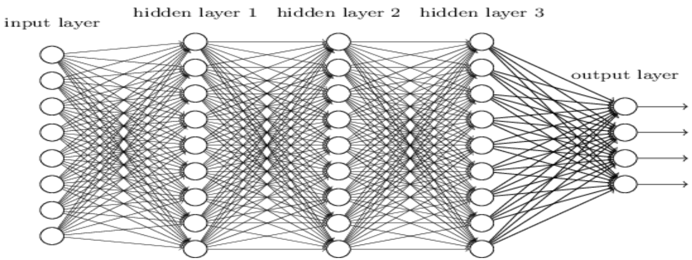
Arjun

DEEP LEARNING

How do you guide the model to find the best features?

$N \times N$

NEURAL NETWORK



Arjun

DEEP LEARNING GLOSSARY

LIBRARY

MKL-DNN
DAAL
Spark MlLib
Scikit-Learn
Intel® Distribution for Python
Mahout
NumPy
Pandas

Hardware-optimized mathematical and other primitive functions that are commonly used in machine & deep learning algorithms, topologies & frameworks

FRAMEWORK

TensorFlow
BigDL for Spark
mxnet
PaddlePaddle
PYTORCH
Caffe

Open-source software environments that facilitate deep learning model development & deployment through built-in components and the ability to customize code

TOPOLOGY

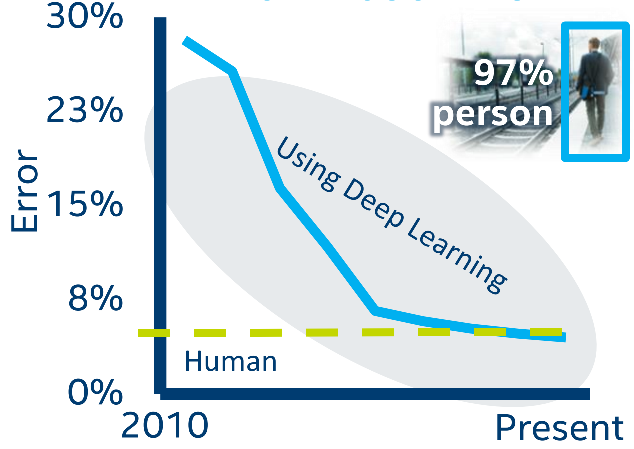
Wide variety of algorithms modeled loosely after the human brain that use neural networks to recognize complex patterns in data that are otherwise difficult to reverse engineer

Translating common deep learning terminology

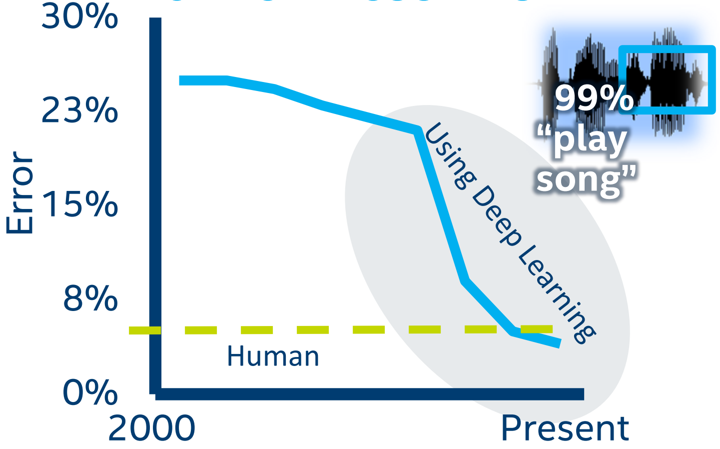
DEEP LEARNING BREAKTHROUGHS

Machines able to meet or exceed human image & speech recognition

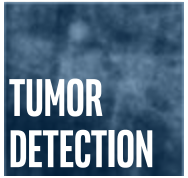
IMAGE RECOGNITION



SPEECH RECOGNITION



e.g.

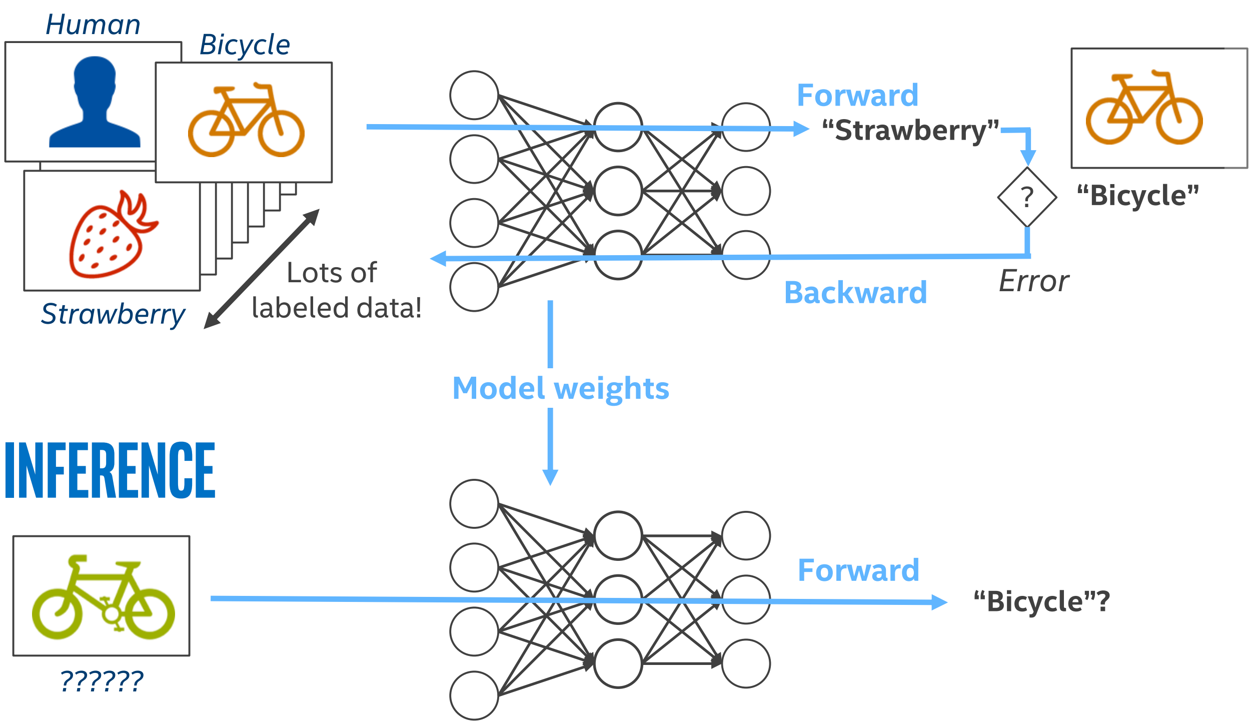


Source: ILSVRC ImageNet winning entry classification error rate each year 2010-2016 (Left), https://www.microsoft.com/en-us/research/blog/microsoft-researchers-achieve-new-conversational-speech-recognition-milestone/ (Right)

DEEP LEARNING BASICS

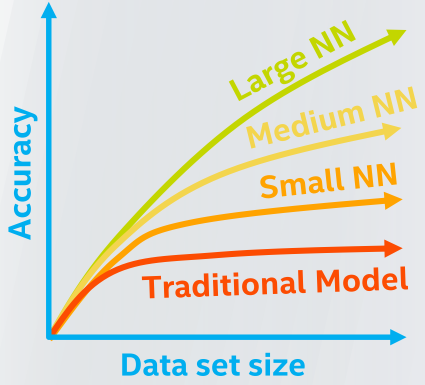
TRAINING

INFERENCE



DID YOU KNOW?

Training with a large data set AND deep (many layered) neural network often leads to the highest accuracy inference



Visit:
www.intel.ai/technology

SPEED UP DEVELOPMENT

using open AI software

MACHINE LEARNING | DEEP LEARNING

TOOLKITS
App developers

ANALYTICS ZOO
Open source platform for building E2E Analytics & AI applications on Apache Spark* with distributed TensorFlow*, Keras*, BigDL

OpenVINO™
Deep learning inference deployment on CPU/GPU/FPGA/VPU for Caffe*, TensorFlow*, MXNet*, ONNX*, Kaldi*

NAUTA
Open source, scalable, and extensible distributed deep learning platform built on Kubernetes (BETA)

LIBRARIES
Data scientists

Python
- Scikit-learn
- Pandas
- NumPy

R
- Cart
- Random Forest
- e1071

Distributed
- MlLib (on Spark)
- Mahout

Intel-optimized Frameworks

TensorFlow*, Caffe2*, ONNX*, mxnet*, PyTorch*, BigDL

And more framework optimizations underway including PaddlePaddle*, Chainer*, CNTK* & others

KERNELS
Library developers

Intel® Distribution for Python*
Intel distribution optimized for machine learning

Intel® Data Analytics Acceleration Library (DAAL)
High performance machine learning & data analytics library

Intel® Math Kernel Library for Deep Neural Networks (MKL-DNN)
Open source DNN functions for CPU / integrated graphics

nGraph
Open source compiler for deep learning model computations optimized for multiple devices (CPU, GPU, NNP) from multiple frameworks (TF, MXNet, ONNX)

INTRODUCTION TO INTEL® DEEP LEARNING BOOST

FAST EVOLUTION OF AI CAPABILITY ON INTEL® XEON® PLATFORM

Grantley

- Haswell (HSX): E5 V3
- Broadwell (BDX): E5 V4

Intel® AVX2 (256 bit)

Purley

- Skylake (SKX): SP (Scalable Processor)
  E.g. Gold 8180
  Gold 5117
  Silver 4110
- Cascade Lake (CLX): SP (Scalable Processor)
  E.g. Gold 8280
  Gold 5218

Intel® AVX512 (512 bit)
FP32, INT8, …

Intel® AVX512 **VNNI** (512 bit)
FP32, VNNI INT8, …

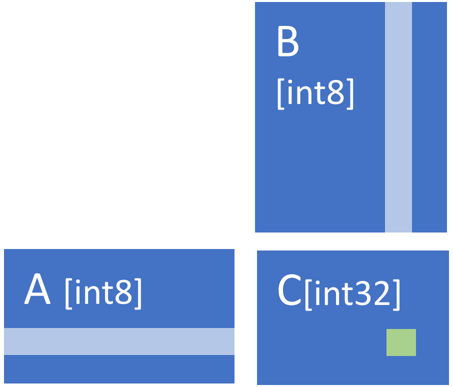
Intel® Deep Learning Boost

2015 — 2016 — 2017 — 2019

Intel® Deep Learning Boost **is a new set of AVX-512 instructions** ***designed to deliver significant, more efficient Deep Learning (Inference) acceleration on second generation Intel® Xeon® Scalable processor (codename "Cascade Lake")***

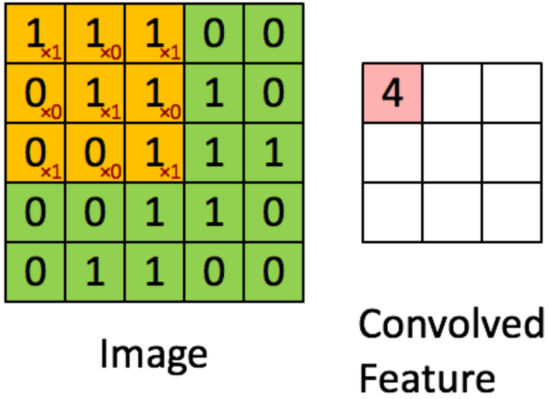
DEEP LEARNING FOUNDATIONS

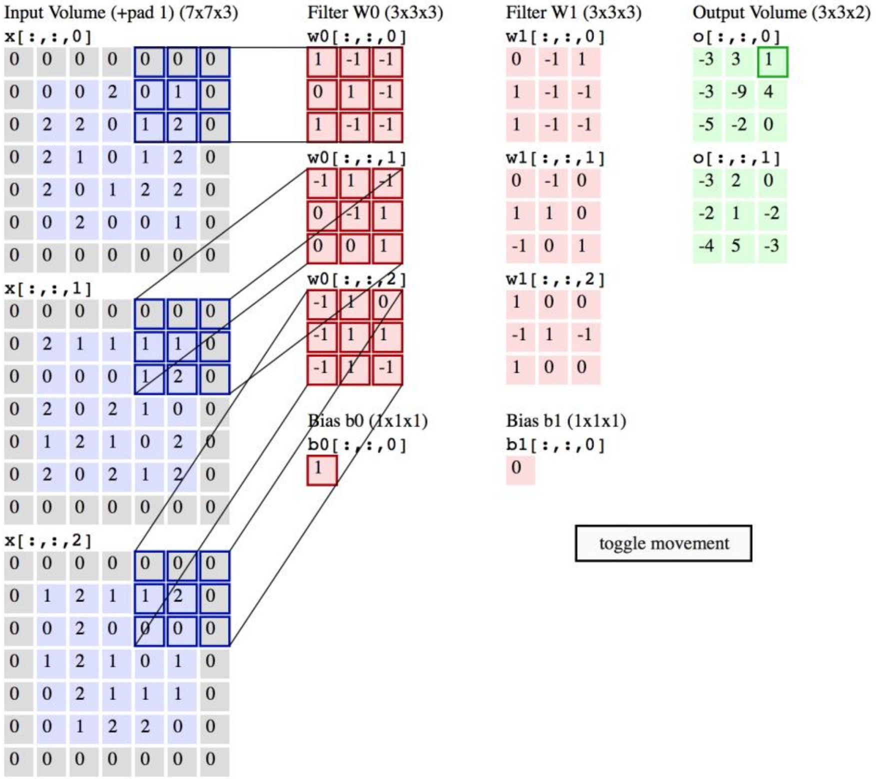
- **Matrix Multiplies are the foundation of many DL applications**
  - **Multiply** a row*column values, **accumulate** into a single value
- **Traditional HPC and many AI training workloads use floating point**
  - Massive dynamic range of values (FP32 goes up to ~2^128)
- **Why INT8 for Inference?**
  - More power efficient per operation due to smaller multiplies
  - Reduces pressure on cache and memory subsystem
  - Precision and dynamic range sufficient for many models
- **What's different about INT8?**
  - Much smaller dynamic range than FP32: 256 values
    - Requires *accumulation into INT32* to avoid overflow
      (FP handles this "for free" w/ large dynamic range)



Matrix Multiply
A x B = C

CONVOLUTION = MULTIPLY – ADD OP.





REDUCED PRECISION FOR INFERENCE

- Data types for different phases of deep learning
  - Training: fp32, fp16, bfloat16, …
  - Inference: fp32, fp16, **int8**, …

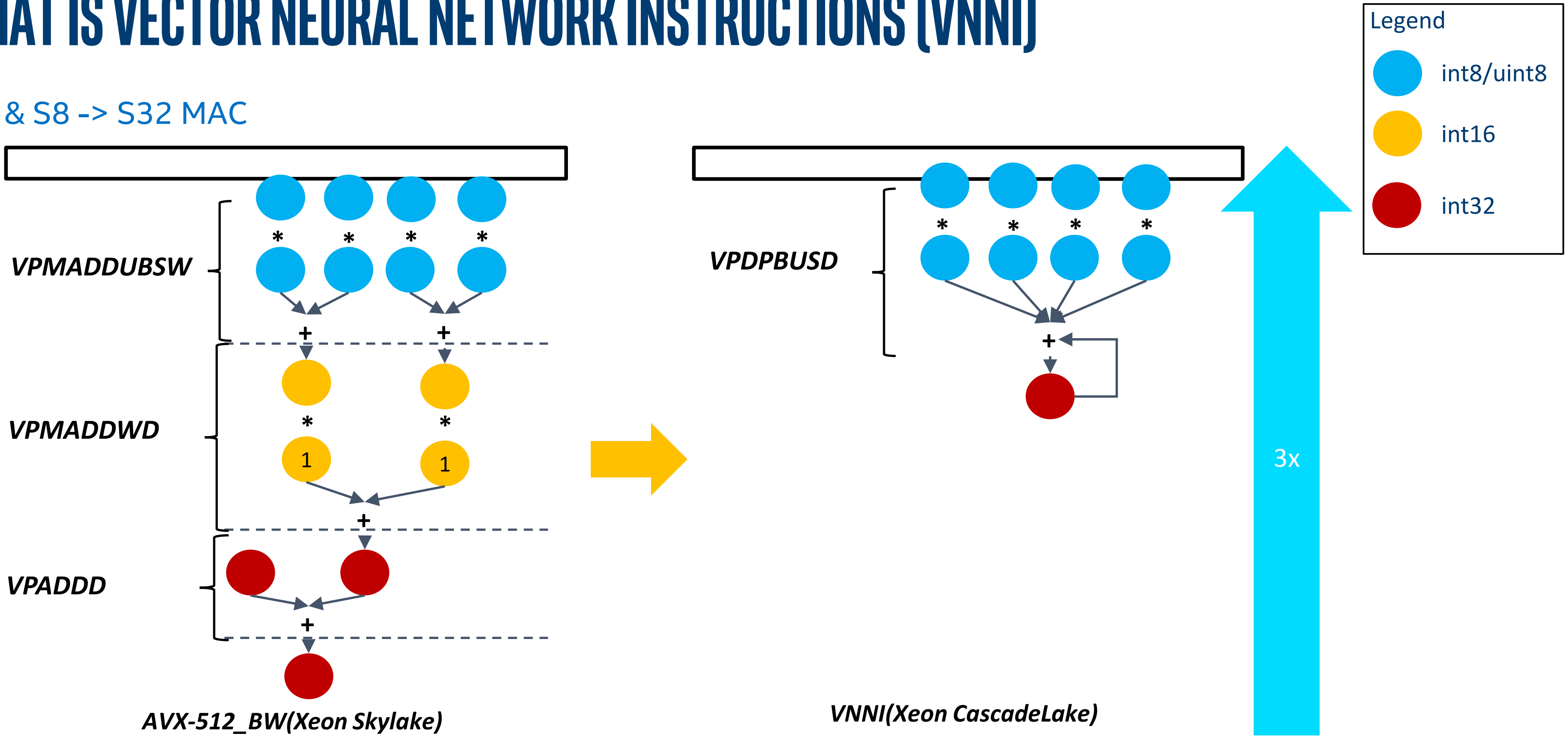
	Dynamic Range	Min Positive Value
FP32	$-3.4 \times 10^{38} \sim +3.4 \times 10^{38}$	1.4×10^{-45}
FP16	$-65504 \sim +65504$	5.96×10^{-8}
INT8	$-128 \sim +127$	1

***INT8** has significantly lower precision and dynamic range than FP32*

- **int8** vs fp32
  - Better performance (instruction throughput)
  - Low memory consumption (high bandwidth, better cache usage)
  - Acceptable accuracy loss

WHAT IS VECTOR NEURAL NETWORK INSTRUCTIONS (VNNI)

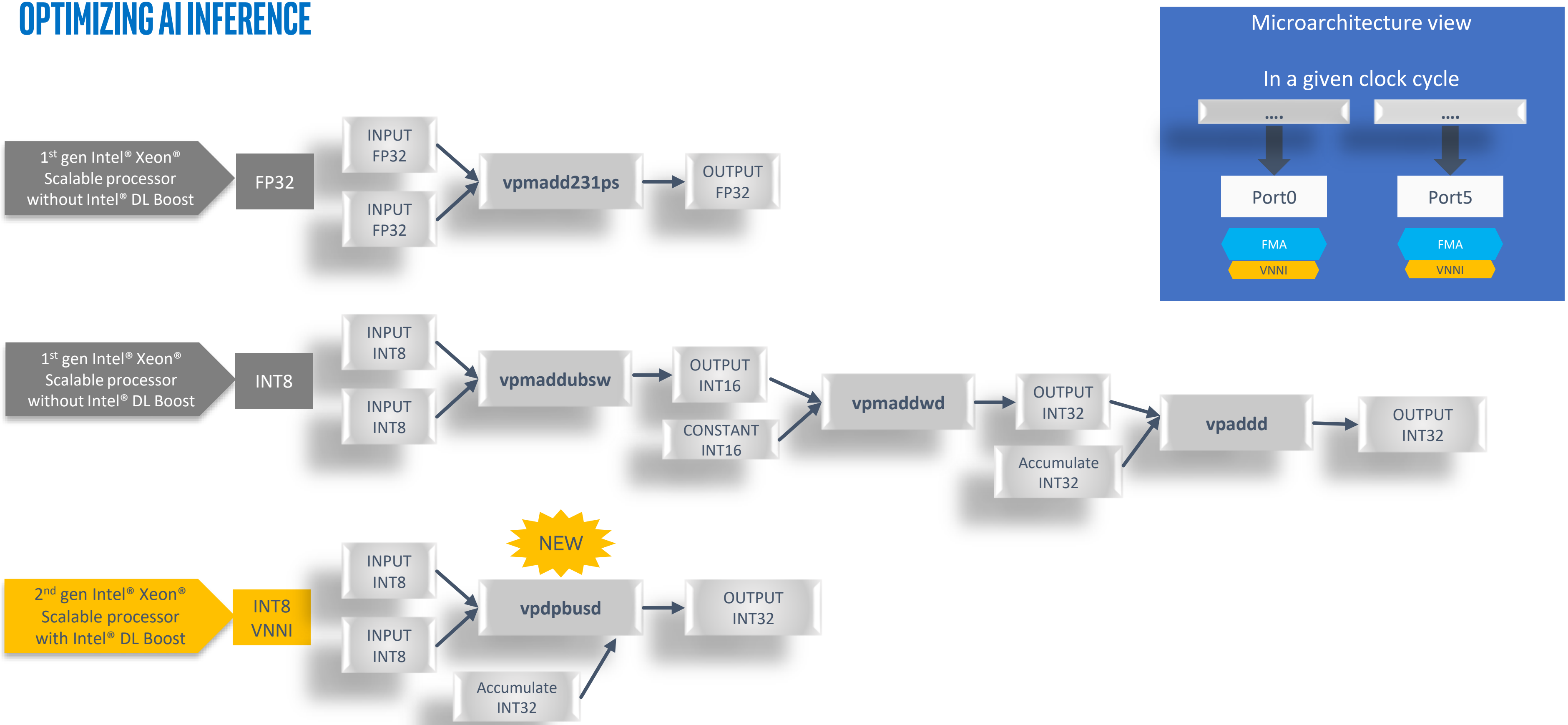
- U8 & S8 -> S32 MAC



VNNI can boost INT8 MAC from 1.33x of FP32 with AVX-512_BW to 4x of FP32 while iso-frequency.

INTEL® DEEP LEARNING BOOST

OPTIMIZING AI INFERENCE



SO WHAT?

AVX512_VNNI is a new set of of AVX-512 instructions to boost Deep Learning performance

- VNNI includes FMA instructions for:
  - 8-bit multiplies with 32-bit accumulates (u8 x s8 ⇒ s32)
  - 16-bit multiplies with 32-bit accumulates (s16 x s16 ⇒ s32)
- Theoretical peak compute gains are:
  - 4x int8 OPS over fp32 OPS and ¼ memory requirements
  - 2x int16 OPS over fp32 OPS and ½ memory requirements
- Ice Lake and future microarchitectures will have AVX512_VNNI

ENABLING INTEL® DL BOOST ON CASCADE LAKE

THEORETICAL IMPROVEMENTS: FP32 VS. INT8 & DL BOOST

UP TO 4X BOOST IN MAC/CYCLE

UP TO 4X IMPROVED PERFORMANCE / WATT

DECREASED MEMORY BANDWIDTH

IMPROVED CACHE PERFORMANCE

Workloads

Topologies

TensorFlow, mxnet, Caffe2, Caffe, PaddlePaddle

Frameworks

Intel® MKL-DNN Libraries

Intel® Processors

UP NEXT: MICROBENCHMARKING WITH INTEL® MKL-DNN'S

INTEL® AI OPTIMIZED FRAMEWORKS

INTEL® AI OPTIMIZED FRAMEWORKS

Popular DL Frameworks are now optimized for CPU!

CHOOSE YOUR FAVORITE FRAMEWORK

TensorFlow™* | Caffe* | mxnet* | BigDL FOR APACHE Spark™* | PYTORCH

See installation guides at ai.intel.com/framework-optimizations/

More under optimization: Caffe2* PYTORCH* PaddlePaddle*

SEE ALSO: Machine Learning Libraries for Python (Scikit-learn, Pandas, NumPy), R (Cart, randomForest, e1071), Distributed (MlLib on Spark, Mahout)
**Limited availability today*
Other names and brands may be claimed as the property of others.

AI (ML & DL) SOFTWARE STACK FOR INTEL® PROCESSORS

Deep learning and AI ecosystem includes edge and datacenter applications.

- Open source frameworks (Tensorflow*, MXNet*, PyTorch*, PaddlePaddle*)
- Intel deep learning products (, BigDL, OpenVINO™ toolkit)
- In-house user applications

Intel® MKL and Intel® MKL-DNN optimize deep learning and machine learning applications for Intel® processors :

- Through the collaboration with framework maintainers to upstream changes (Tensorflow*, MXNet*, PyTorch, PaddlePaddle*)
- Through Intel-optimized forks (Caffe*)
- By partnering to enable proprietary solutions

Intel® MKL-DNN is an open source performance library for deep learning applications (available at https://github.com/intel/mkl-dnn)

- Fast open source implementations for wide range of DNN functions
- Early access to new and experimental functionality
- Open for community contributions

Intel® MKL is a proprietary performance library for wide range of math and science applications
Distribution: Intel Registration Center, package repositories (apt, yum, conda, pip), Intel® Parallel Studio XE, Intel® System Studio

INTEL® MATH KERNEL FOR DEEP NEURAL NETWORKS (INTEL® MKL-DNN)

For developers of deep learning frameworks featuring optimized performance on Intel hardware

Distribution Details

- Open Source
- Apache* 2.0 License
- Common DNN APIs across all Intel hardware.
- Rapid release cycles, iterated with the DL community, to best support industry framework integration.
- Highly vectorized & threaded for maximal performance, based on the popular Intel® Math Kernel Library.

github.com/01org/mkl-dnn

Examples:

Direct 2D Convolution

Local response normalization (LRN)

Rectified linear unit neuron activation (ReLU)

Maximum pooling

Inner product

Accelerate Performance of Deep Learning Models

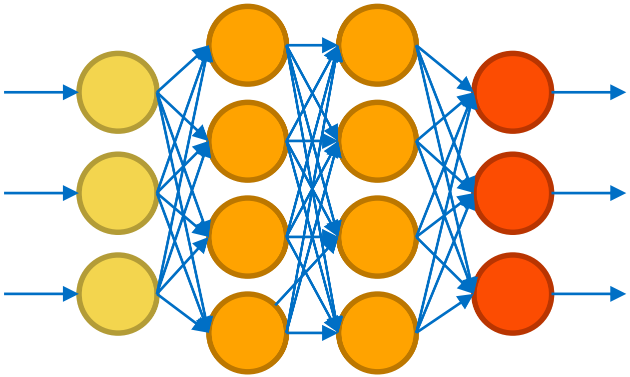
INTEGRATION WITH THE POPULAR AI/ML FRAMEWORKS

NEURAL NETWORKS

Use biology as inspiration for math model

Neurons:

- Get signals from previous neurons
- Generate signal (or not) according to inputs
- Pass that signal on to future neurons



By layering many neurons, can create complex model

MAIN TENSORFLOW API CLASSES

Graph

- Container for operations and tensors

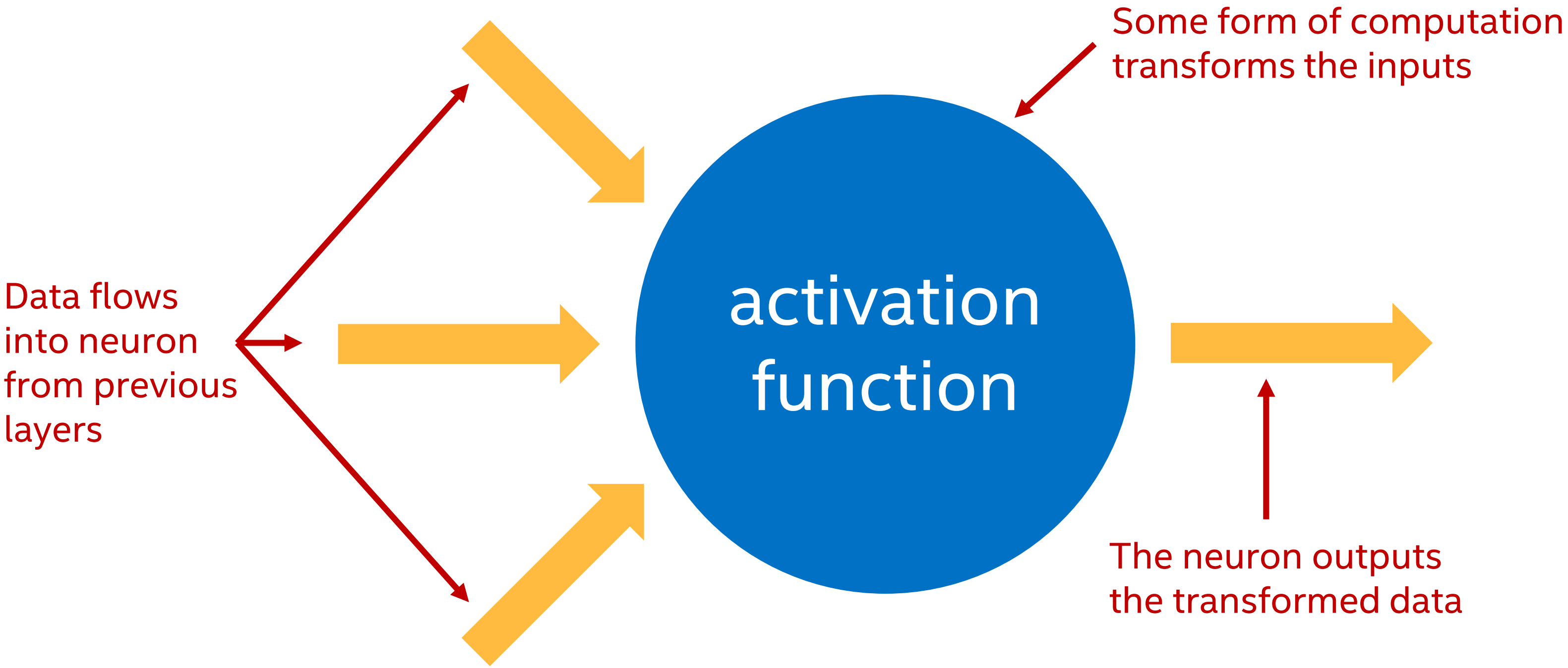
Operation

- Nodes in the graph
- Represent computations

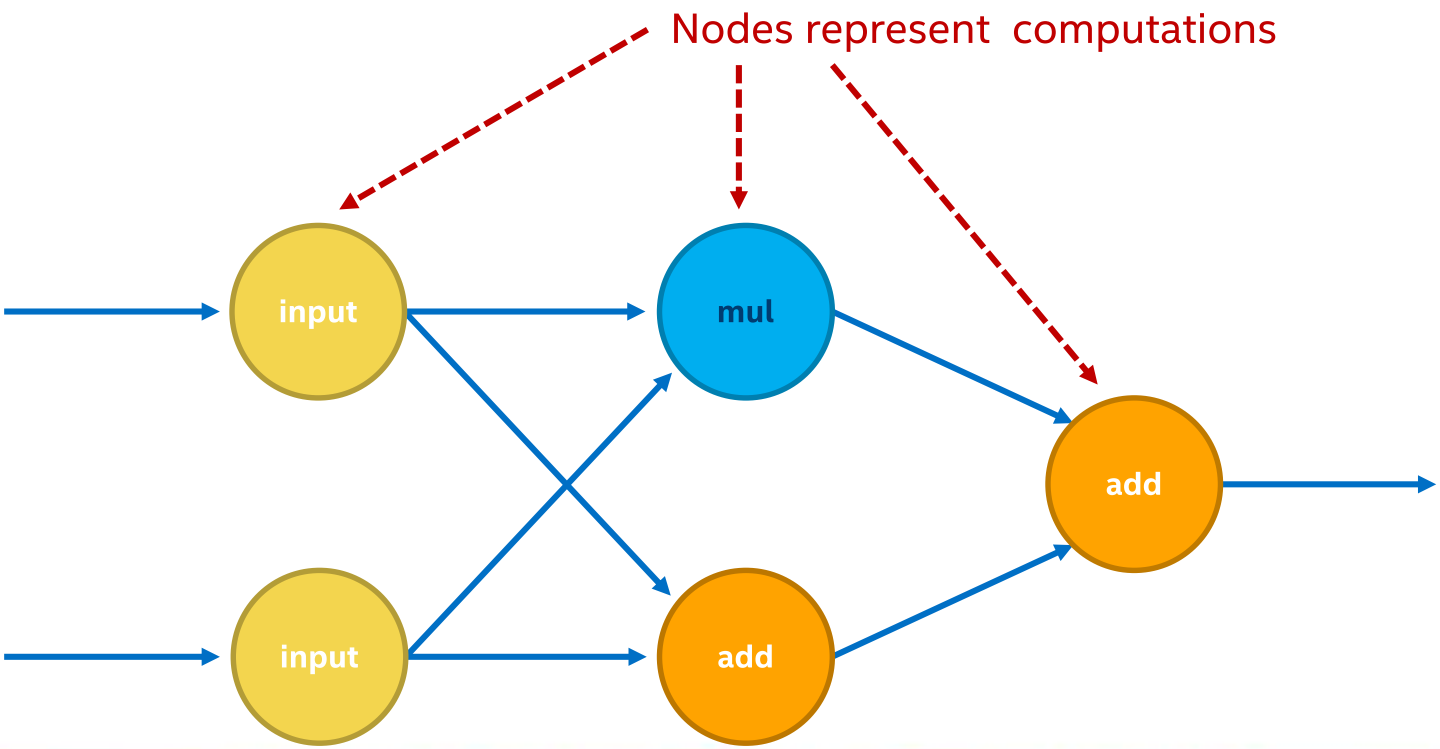
Tensor

- Edges in the graph
- Represent data

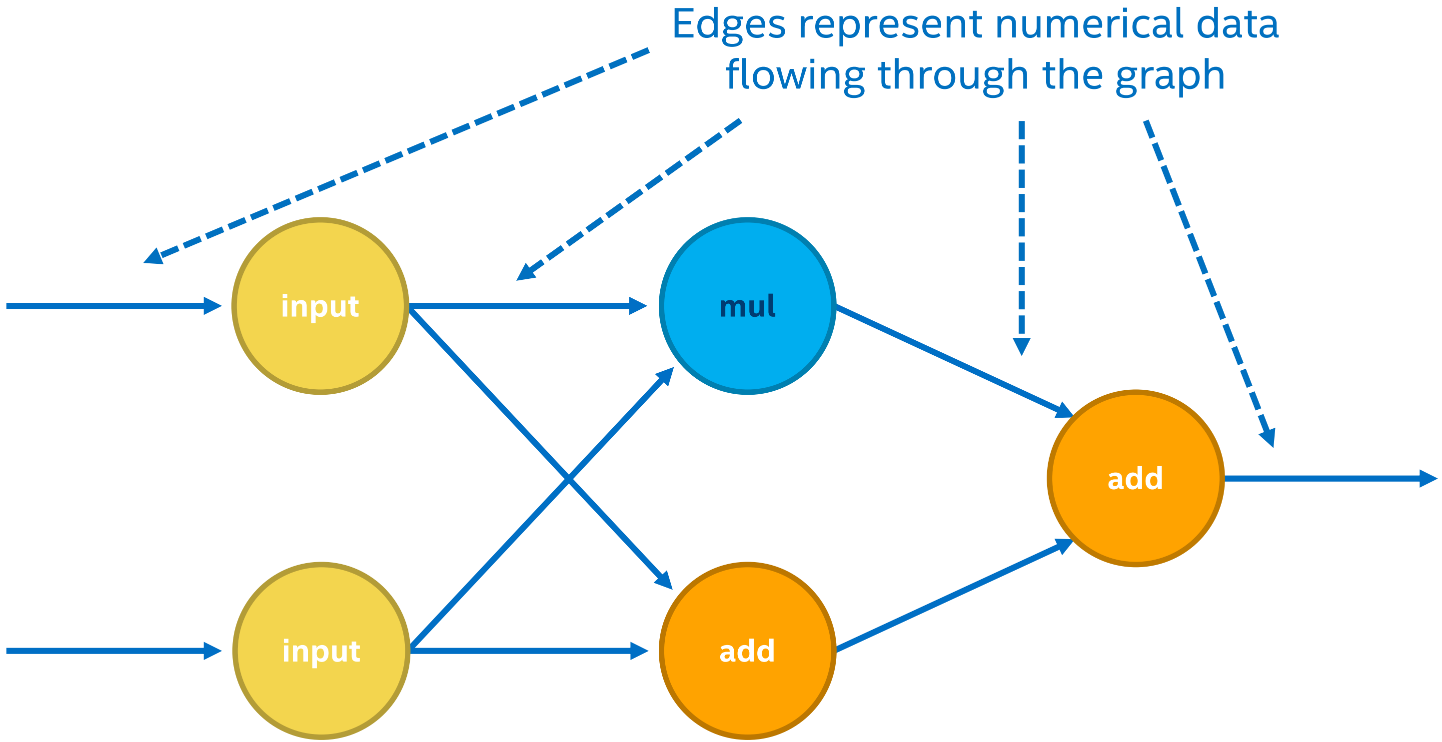
READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH



COMPUTATION GRAPH

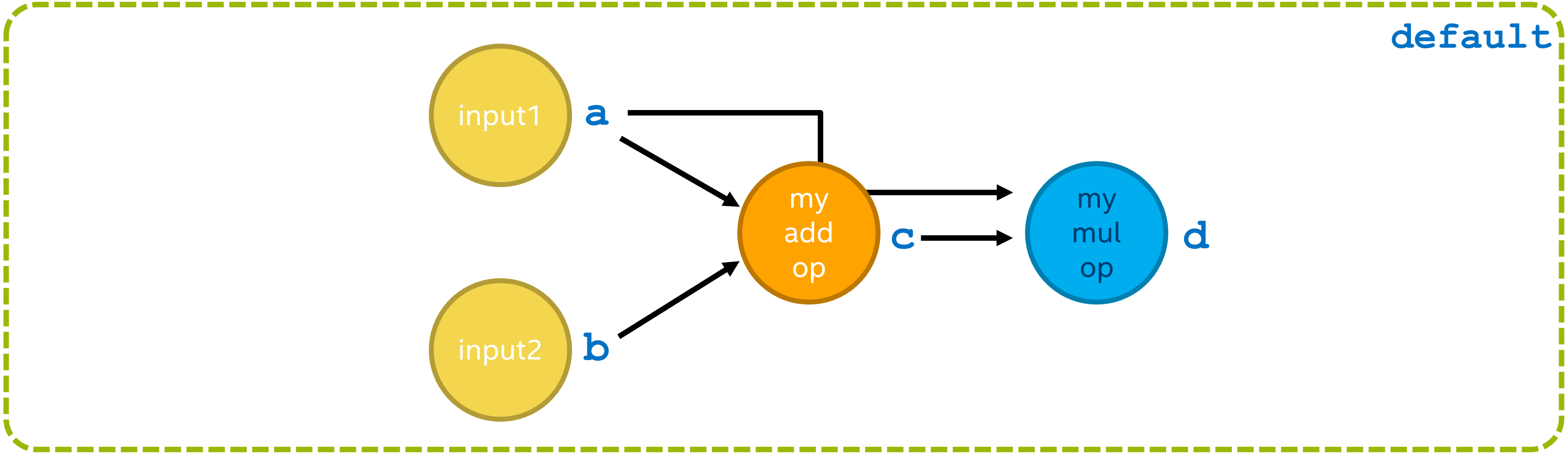


COMPUTATION GRAPH



`tf.constant()` creates an `Operation` that returns a fixed value
`tf.placeholder()` defines explicit input that vary run-to-run

```
>>> a = tf.placeholder(tf.float32, name="input1")
>>> c = tf.add(a, b, name="my_add_op")
```



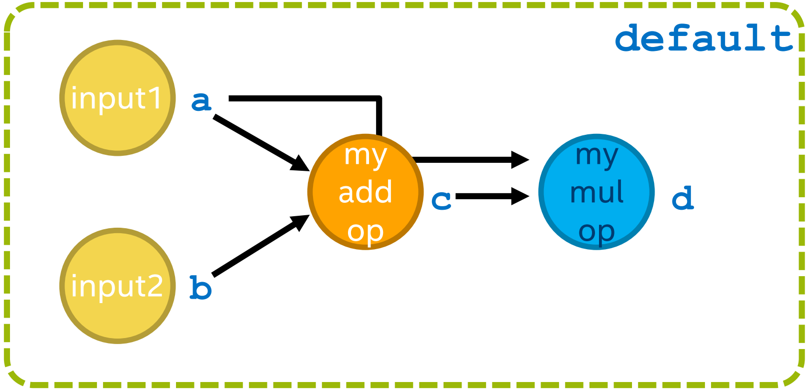
We use a `Session` object to execute graphs.
Each `Session` is dedicated to a single graph.

```
>>> sess = tf.Session()
```

Session `sess`

Graph: `default`

Variable values:



`ConfigProto` is used to set configurations of the `Session` object.

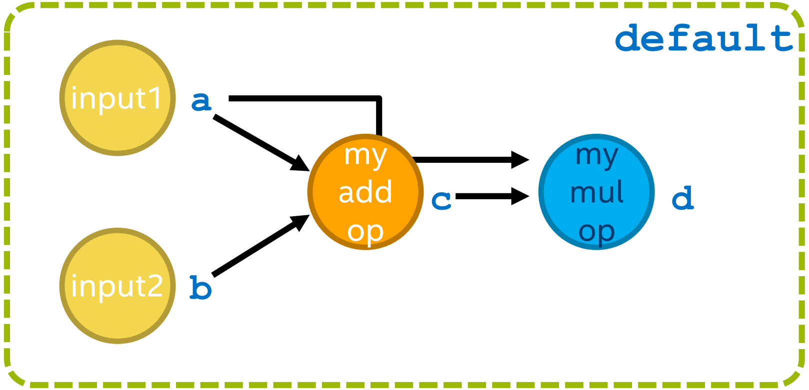
```
>>> config = tf.ConfigProto(inter_op_parallelism_threads=2,
                intra_op_parallelism_threads=44)

>>> tf.Session(config=config)
```

Session `sess`

Graph: `default`

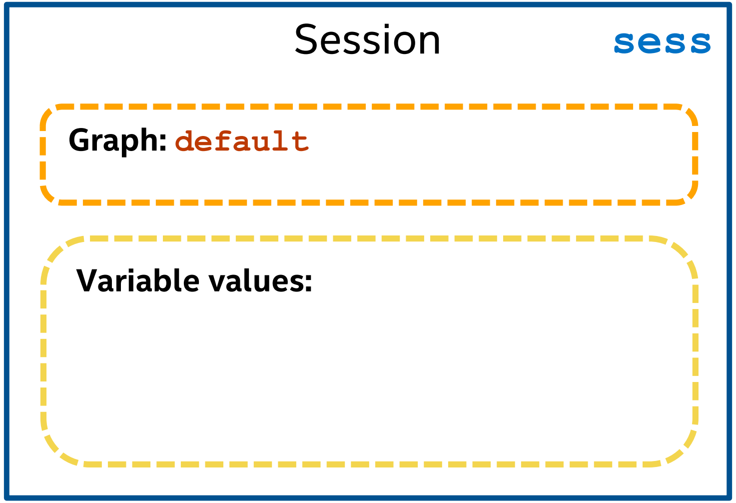
Variable values:

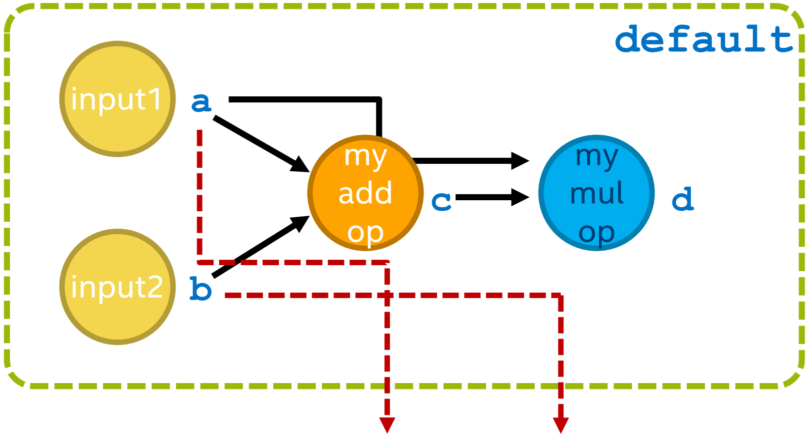


`placeholders` require data to fill them in when the graph is run

We do this by creating a dictionary mapping `Tensor` keys to numeric values

```
>>> feed_dict = {a: 3.0, b: 2.0}
```



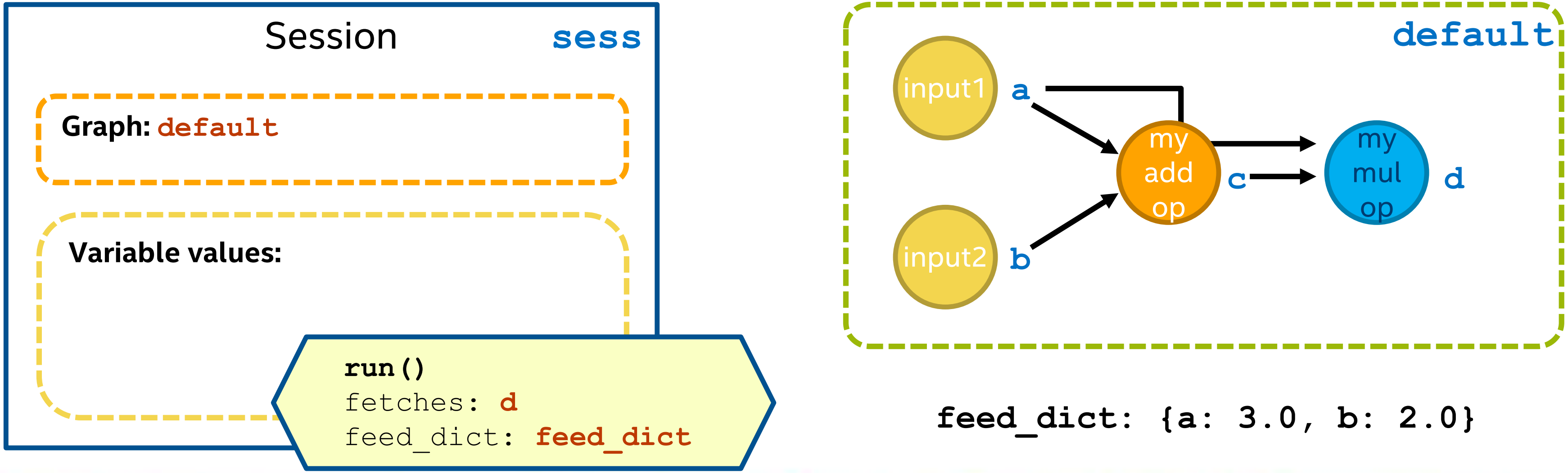


feed_dict: {a: 3.0, b: 2.0}

We execute the graph with `sess.run(fetches, feed_dict)`

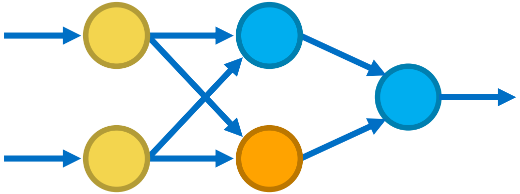
`sess.run` returns the fetched values as a NumPy array

```
>>> out = sess.run(d, feed_dict=feed_dict)
```

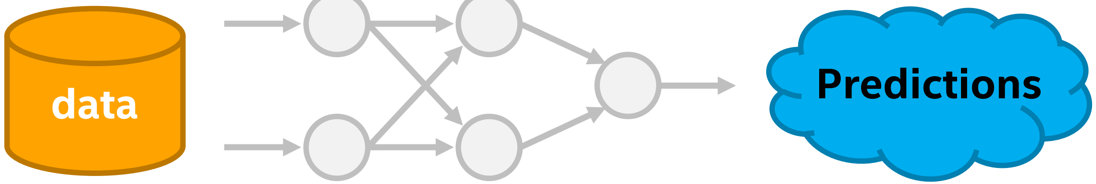


TWO-STEP PROGRAMMING PATTERN

1. Define a computation graph



2. Run the graph

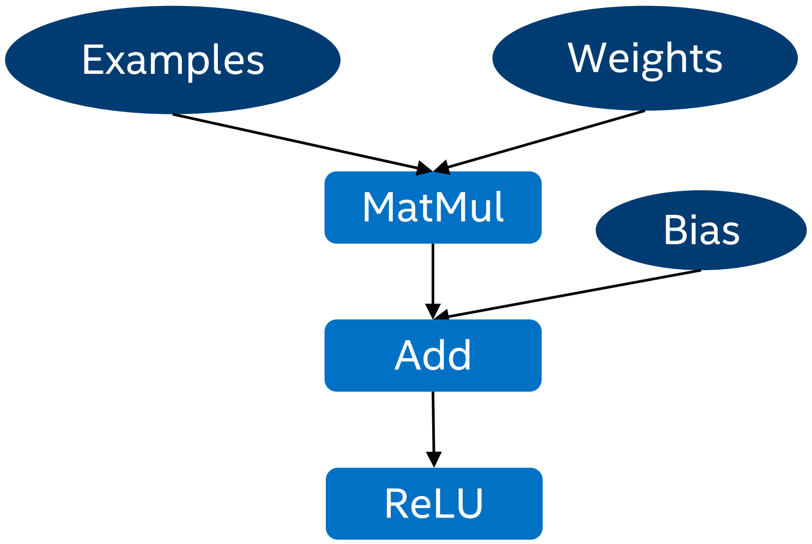


INTEL® TENSORFLOW* OPTIMIZATIONS

1. Operator optimizations: Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)
2. Graph optimizations: Fusion, Layout Propagation
3. System optimizations: Threading model

OPERATOR OPTIMIZATIONS

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



OPERATOR OPTIMIZATIONS

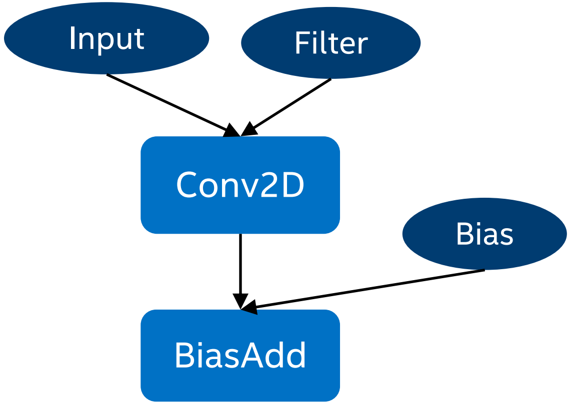
Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)

Intel® MKL-DNN has optimized a set of TensorFlow operations.

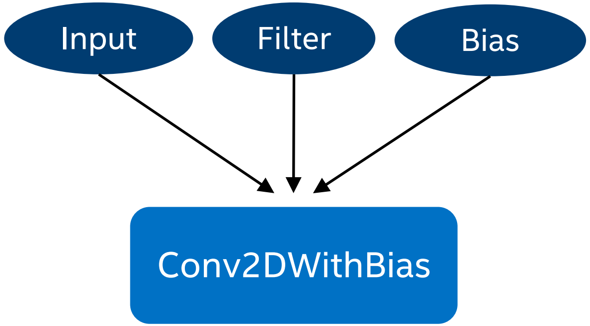
Library is open-source (https://github.com/intel/mkl-dnn) and downloaded automatically when building TensorFlow.

Forward	Backward
Conv2D	Conv2DGrad
Relu, TanH, ELU	ReLUGrad, TanHGrad, ELUGrad
MaxPooling	MaxPoolingGrad
AvgPooling	AvgPoolingGrad
BatchNorm	BatchNormGrad
LRN	LRNGrad
MatMul, Concat	

GRAPH OPTIMIZATIONS: FUSION



Before Merge

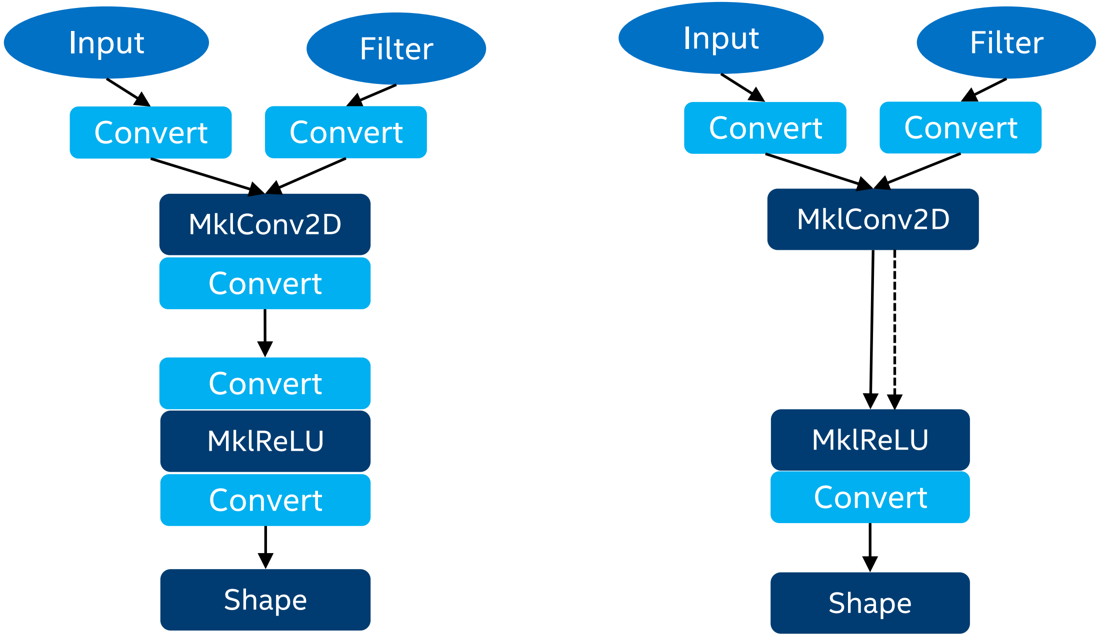


After Merge

GRAPH OPTIMIZATIONS: LAYOUT PROPAGATION

Converting to/from optimized layout can be less expensive than operating on un-optimized layout.

All MKL-DNN operators use highly-optimized layouts for TensorFlow tensors.



After Layout Conversion

After Layout Propagation

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

21	18	32	6	3
1				
40				
23				
5				

8	92	37	29	44
11	9	22	3	26
3	47	29	88	1
15	16	22	46	12
29	9	13	11	1

21	18	…	1	..	8	92	..	Channel based (NCHW)
21	8	18	92	..	1	11	..	Pixel based (NHWC)

```
for i= 1 to N # batch size
    for j = 1 to C # number of channels, image RGB = 3 channels
        for k  = 1 to H  # height
            for l = 1 to W # width
                dot_product( …)
```

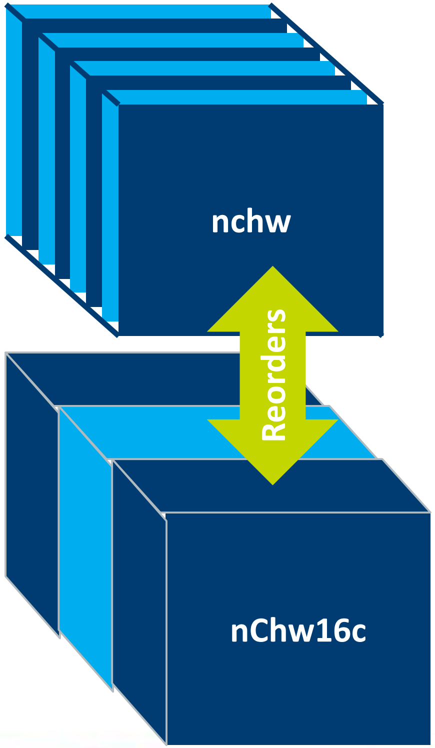
MORE ON MEMORY CHANNELS: MEMORY LAYOUTS

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 MKL-DNN 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

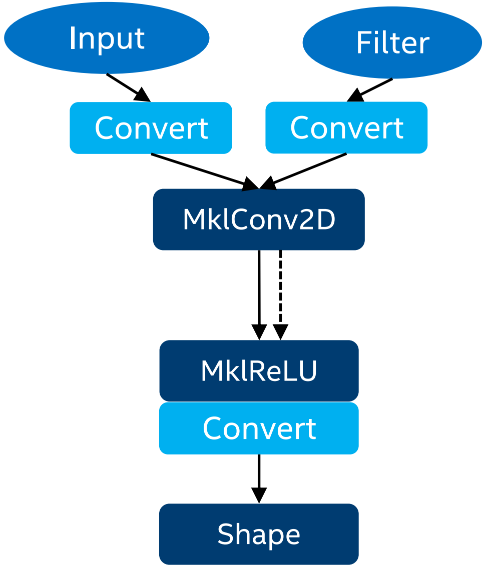


SYSTEM OPTIMIZATIONS: LOAD BALANCING

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`**

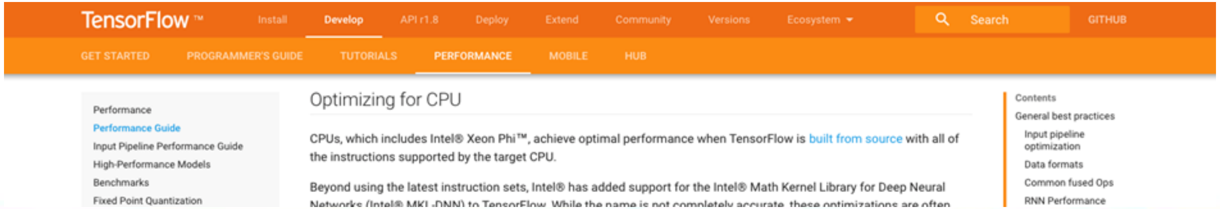


PERFORMANCE GUIDE

`tf.ConfigProto` is used to set the `inter_op_parallelism_threads` and `intra_op_parallelism_threads` configurations of the `Session` object.

```
>>> 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

Maximize TensorFlow* Performance on CPU: Considerations and Recommendations for Inference Workloads: https://software.intel.com/en-us/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference

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

```
os.environ["KMP_BLOCKTIME"] = "1"
os.environ["KMP_AFFINITY"] = "granularity=fine,compact,1,0"
os.environ["KMP_SETTINGS"] = "0"
os.environ["OMP_NUM_THREADS"] = “56"
```

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.

Intel Tensorflow* install guide is available → https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide

DISTRIBUTED TENSORFLOW™ COMPARE

Distributed Tensorflow with Parameter Server

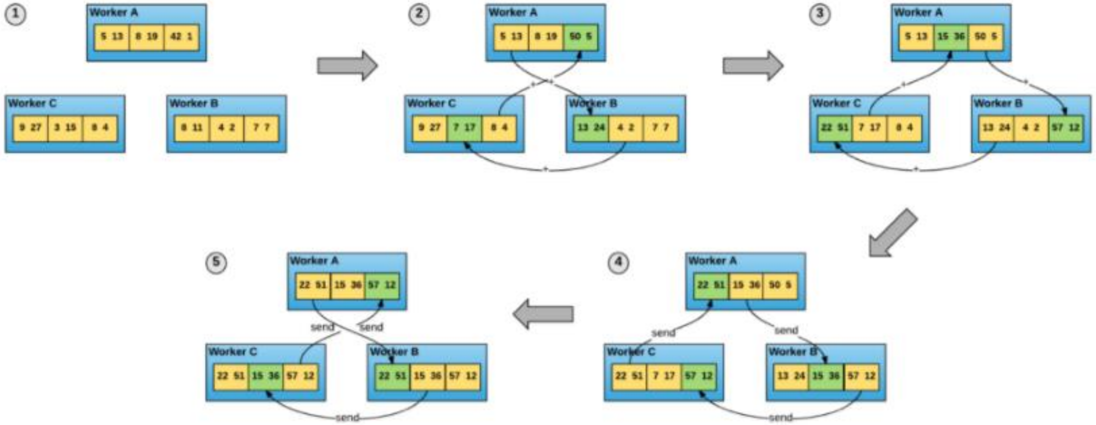
With Parameter Server

The parameter server model for distributed training jobs can be configured with different ratios of parameter servers to workers, each with different performance profiles.

HOROVOD

Uber's open source Distributed training framework for TensorFlow

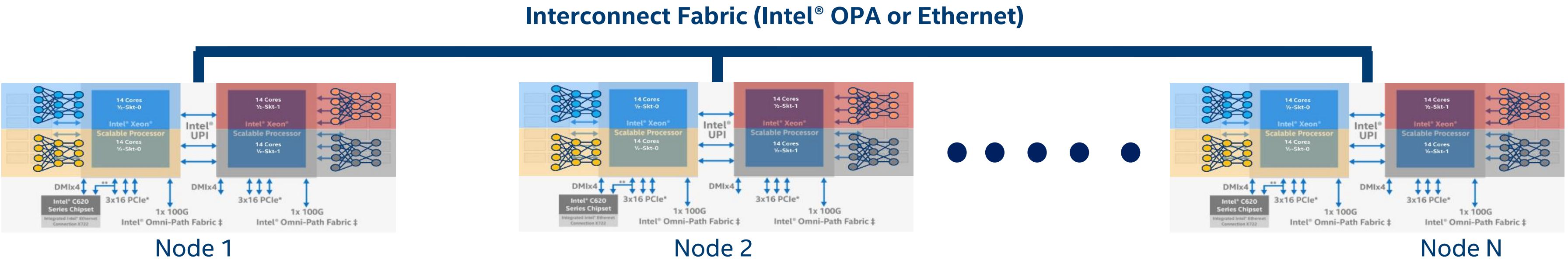
No Parameter Server



The ring all-reduce algorithm allows worker nodes to average gradients and disperse them to all nodes without the need for a parameter server.

Source: https://eng.uber.com/horovod/

DISTRIBUTED TRAINING WITH HOROVOD* MPI LIB



Distributed Deep Learning Training Across Multiple nodes

Each node running multiple workers/node

Uses optimized MPI Library for gradient updates over network fabric

Caffe – Use Optimized Intel® MPI ML Scaling Library (Intel® MLSL)

TensorFlow* – Uber horovod MPI Library

Intel Best Known Methods: https://ai.intel.com/accelerating-deep-learning-training-inference-system-level-optimizations/

HOROVOD: HOW TO CHANGE THE CODE

Usage

To use Horovod, make the following additions to your program. This example uses TensorFlow.

1. Run `hvd.init()`.
2. Pin a server GPU to be used by this process using `config.gpu_options.visible_device_list`. With the typical setup of one GPU per process, this can be set to *local rank*. In that case, the first process on the server will be allocated the first GPU, second process will be allocated the second GPU and so forth.
3. Scale the learning rate by number of workers. Effective batch size in synchronous distributed training is scaled by the number of workers. An increase in learning rate compensates for the increased batch size.
4. Wrap optimizer in `hvd.DistributedOptimizer`. The distributed optimizer delegates gradient computation to the original optimizer, averages gradients using **allreduce** or **allgather**, and then applies those averaged gradients.
5. Add `hvd.BroadcastGlobalVariablesHook(0)` to broadcast initial variable states from rank 0 to all other processes. This is necessary to ensure consistent initialization of all workers when training is started with random weights or restored from a checkpoint. Alternatively, if you're not using `MonitoredTrainingSession`, you can simply execute the `hvd.broadcast_global_variables` op after global variables have been initialized.
6. Modify your code to save checkpoints only on worker 0 to prevent other workers from corrupting them. This can be accomplished by passing `checkpoint_dir=None` to `tf.train.MonitoredTrainingSession` if `hvd.rank() != 0`.

Example (see the examples directory for full training examples):

```
import tensorflow as tf
import horovod.tensorflow as hvd


# Initialize Horovod
hvd.init()
```

HOROVOD

Guidelines and example on github:

https://github.com/horovod/horovod#usage

GitHub

HOROVOD 101 QUICK START

```
import horovod.tensorflow as hvd
hvd.init()

#Scale the optimizer
opt = tf.train.AdagradOptimizer(0.01 * hvd.size())

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Save checkpoints only on worker 0 to prevent other workers from
corrupting them.
checkpoint_dir = '/tmp/train_logs' if hvd.rank() == 0 else None
```

HOROVOD* FOR MULTI-NODE

from Parameter server (PS):

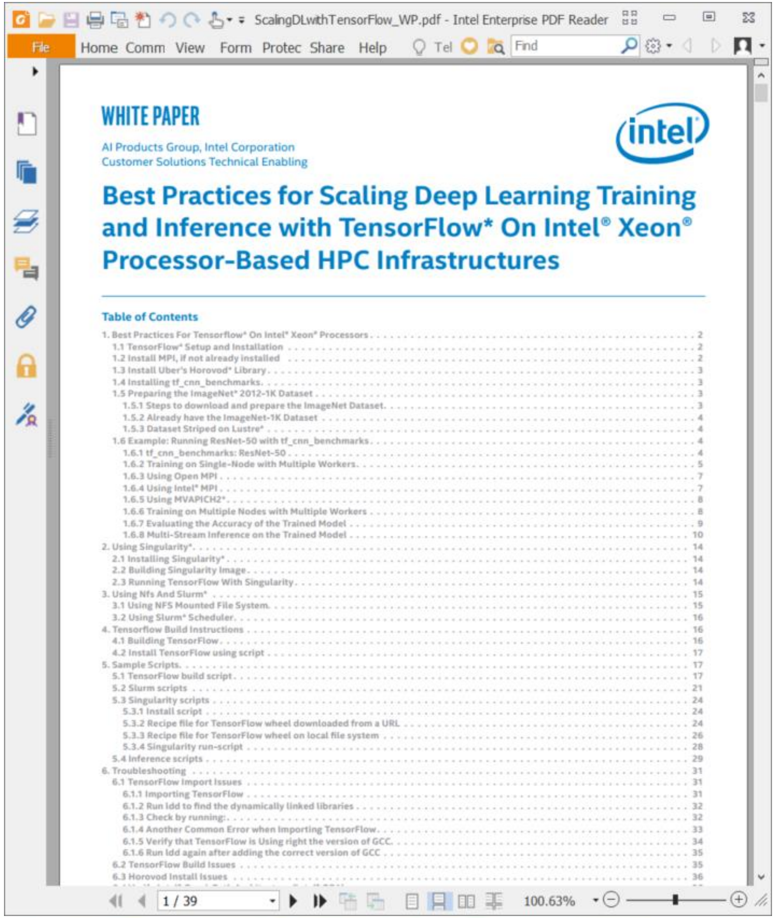
```
NP=4
PER_PROC=10
HOSTLIST=192.168.10.110
MODEL=inception3
BS=64
BATCHES=100
INTRA=10
INTER=2
```

```
/usr/lib64/openmpi/bin/mpirun --allow-run-as-root -np $NP -cpus-per-proc $PER_PROC –
map-by socket -H $HOSTLIST --report-bindings --oversubscribe -x LD_LIBRARY_PATH python
./tf_cnn_benchmarks.py --model $MODEL --batch_size $BS --data_format NCHW –
num_batches $BATCHES --distortions=True --mkl=True --local_parameter_device cpu –
num_warmup_batches 10 --optimizer rmsprop --display_every 10 --kmp_blocktime 1 –
variable_update horovod --horovod_device cpu --num_intra_threads $INTRA –
num_inter_threads $INTER  --data_dir /home/tf_imagenet --data_name imagenet
```

SCALING TENSORFLOW*

There is way more to consider when striking for peak performance on distributed deep learning training.:

https://ai.intel.com/white-papers/best-known-methods-for-scaling-deep-learning-with-tensorflow-on-intel-xeon-processor-based-clusters/



WHITE PAPER

AI Products Group, Intel Corporation
Customer Solutions Technical Enabling

intel

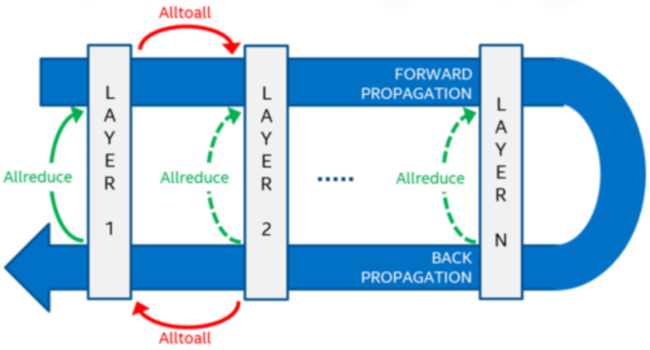
Best Practices for Scaling Deep Learning Training and Inference with TensorFlow* On Intel® Xeon® Processor-Based HPC Infrastructures

Table of Contents

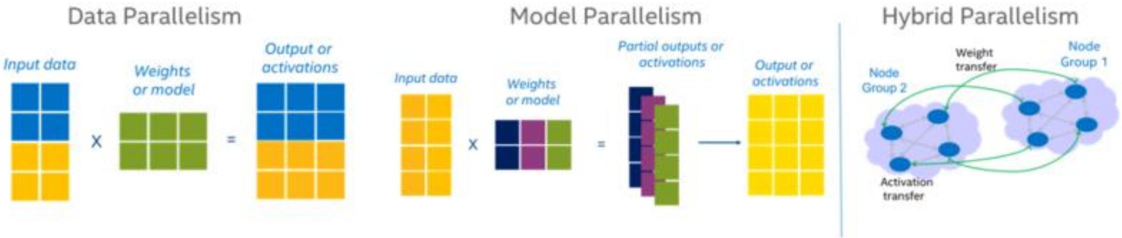
INTEL® MACHINE LEARNING SCALING LIBRARY (MLSL)

Distributed Deep Leaning Requirements:

✓Compute/communication overlap
✓Choosing optimal communication algorithm
✓Prioritizing latency-bound communication
✓Portable / efficient implementation
✓Ease of integration with quantization algorithms
✓Integration with Deep Learning Frameworks



Communication dependent on work partitioning strategy
Data parallelism = Allreduce (or) Reduce_Scatter + Allgather
Model parallelism = AlltoAll



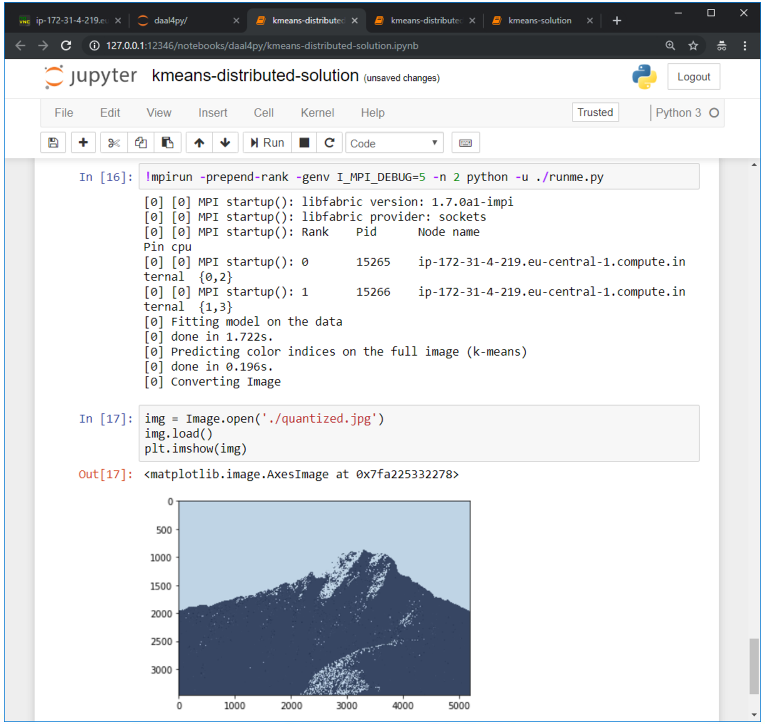
HANDS-ON

daal4py/kmeans-distr.ipynb

1) Performs a pixel-wise Vector Quantization (VQ) using K-Means
2) Implemented the domain decomposition according to:
   - d4p.num_procs()
   - d4p.my_procid()
3) Using the distributed algorithm from Daal4Py
   - d4p.kmeans_init(n_colors, method="plusPlusDense", distributed=True)
4) What is the meaning of d4p.daalinit() & d4p.daalfini()?
5) How does threading compare to multiprocessing in terms of performance?

Distributed K-Means Demo Summary

- Each process (MPI rank) get's a different chunk of data
- Only process #0 reports results
- Inference is using the same routines as training with 0 maximum iterations and centroid assignment
- There is no oversubscription since DAAL only sees the cores “owned” by the corresponding MPI rank



benchmarks/tf_bench.sh

```
Generating training model
Initializing graph
Running warm up
Done warm up
Step    Img/sec total_loss
1       images/sec: 3.7 +/- 0.0 (jitter = 0.0)  7.780
10      images/sec: 3.8 +/- 0.0 (jitter = 0.1)  7.877
20      images/sec: 3.9 +/- 0.0 (jitter = 0.1)  7.744
30      images/sec: 3.8 +/- 0.0 (jitter = 0.1)  7.672
----------------------------------------------------------------
total images/sec: 3.84
----------------------------------------------------------------
```

Tensorflow*

```
Generating training model
Initializing graph
Running warm up
Done warm up
Step    Img/sec total_loss
1       images/sec: 17.3 +/- 0.0 (jitter = 0.0) 7.993
10      images/sec: 17.6 +/- 0.1 (jitter = 0.4) 7.854
20      images/sec: 17.6 +/- 0.1 (jitter = 0.5) 7.726
30      images/sec: 17.7 +/- 0.1 (jitter = 0.4) 7.360
----------------------------------------------------------------
total images/sec: 17.69
----------------------------------------------------------------
```

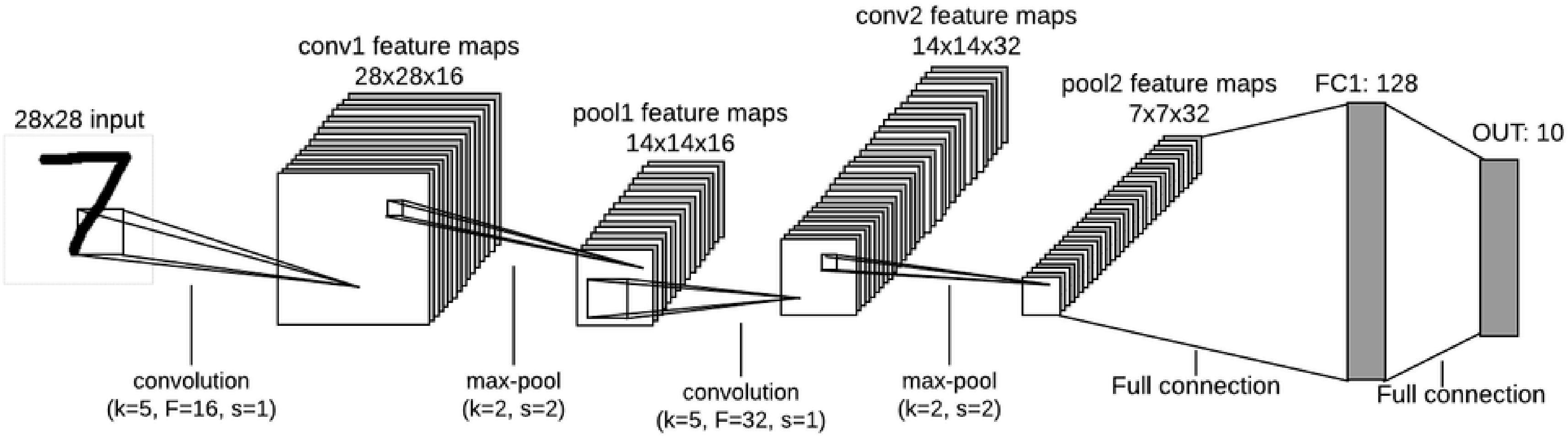
Tensorflow* with Intel® MKL-DNN

```
######### Executive Summary #########


Environment |  Network   | Batch Size | Images/Second
---------------------------------------------------------
Default     | resnet50 |     16     | 3.84
Optimized   | resnet50 |     16     | 17.69


#############################################
Average Intel Optimized speedup = 5X
#############################################
```

Tensorflow/tf_basics/cnn_mnist.ipynb



Source:https://www.easy-tensorflow.com/tf-tutorials/convolutional-neural-nets-cnns

- Implementation of a simple Convolutional Neural Network in TensorFlow with two convolutional layers, followed by two fully-connected layers at the end

Tensorflow/tf_basics/cnn_mnist.ipynb

Let's try to run this example a observe the performance

Standard Python and Tensorflow installation

```
- source activate python-3.6
- pip show tensorflow | grep Location
    - useful to locate the TF installation for see the library linked: ldd $Location/tensorflow,
- rm -rf mnist_convnet_model/*
- Run the sample: time python cnn_mnist.py
```

Intel Python and Optimized Tensorflow

```
- source activate intel-py
- pip show tensorflow | grep Location
    - useful to locate the TF installation for see the library linked: ldd $Location/tensorflow,
- rm-rf mnist_convnet_model/*
- export export MKLDNN VERBOSE=1
```

Tensorflow+Horovod/cnn_mnist-hvd.ipynb

Delete the checkpoint if needed, otherwise TF won't train any further

```
- rm -rf checkpoints
```

Let's start changing the numer of MPI tasks, what performance difference would you expect?

```
- mpirun -prepend-rank -genv OMP_NUM_THREADS=2 -genv I_MPI_DEBUG=5 -n 2 python -u cnn_mnist-hvd.py
- mpirun -prepend-rank -genv OMP_NUM_THREADS=2 -genv I_MPI_DEBUG=5 -n 4 python -u cnn_mnist-hvd.py
- check the size of the dataset:
    - ls -lha ~/.keras/datasets/
```

Intel Python and Optimized Tensorflow

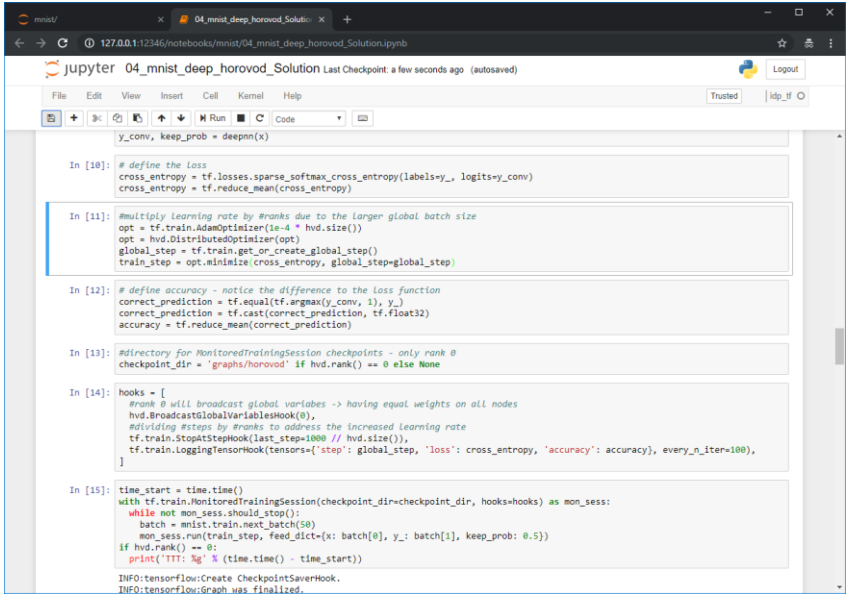
```
- source activate hvd-impi
- pip show tensorflow | grep Location
    - useful to locate the TF installation for see the library linked: ldd $Location/tensorflow/libtensorflow...so
- rm-rf /tmp/*
- export export MKLDNN_VERBOSE=1
```

Tensorflow+Horovod/cnn_mnist-hvd.ipynb

1) How to initialize Horovod and why is it necessary?
2) Why is it necessary to adept the learning rate with larger batches?
3) How can you dynamically adept the learning rate?
4) How to identify rank #1 (0)?
5) Why is it necessary to adept the number of training steps according to the number of workers / larger batches?
6) How can you dynamically adept the number of training steps?
7) How is the single process performance vs 2 ranks vs 4 ranks?

MNIST CNN Horovod Demo Summary

- Horovod initializes the MPI communication underneath and therefore defines rank() and size()
- In order to reduce the Time To Train with multiple workers, therefore increasing the batch size, the learning rate needs to scale
- Same for the # of steps for training
- 4 ranks can be faster since less threading efficiency is required in small convolutions



intel®