A Trip Through the NGC TensorFlow Container

GTC 2019 S9256
AGENDA
A Trip Through the TensorFlow Container

► Getting our bearings…where am I? What is NGC?
► Lazily strolling through the NGC TensorFlow container contents
► Examples!? Check those out!
► Moving in, and using the NGC TensorFlow container daily
THE NGC CONTAINER REGISTRY
Simple Access to GPU-Accelerated Software

Discover over 40 GPU-Accelerated Containers
Spanning deep learning, machine learning, HPC applications, HPC visualization, and more

Innovate in Minutes, Not Weeks
Pre-configured, ready-to-run

Run Anywhere
The top cloud providers, NVIDIA DGX Systems, PCs and workstations with select NVIDIA GPUs, and NGC-Ready systems
THE DESTINATION FOR GPU-ACCELERATED SOFTWARE

<table>
<thead>
<tr>
<th>HPC</th>
<th>Deep Learning</th>
<th>Machine Learning</th>
<th>Inference</th>
<th>Visualization</th>
<th>Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigDFT</td>
<td>Caffe2</td>
<td>H2O Driverless AI</td>
<td>DeepStream</td>
<td>Index</td>
<td>Kubernetes on NVIDIA GPUs</td>
</tr>
<tr>
<td>CANDLE</td>
<td>Chainer</td>
<td>Kinetica</td>
<td>DeepStream 360d</td>
<td>ParaView</td>
<td></td>
</tr>
<tr>
<td>CHROMA</td>
<td>CUDA</td>
<td>MATLAB</td>
<td>TensorRT</td>
<td>ParaView Holodeck</td>
<td></td>
</tr>
<tr>
<td>GAMESS</td>
<td>Deep Cognition Studio</td>
<td>OmniSci (MapD)</td>
<td>TensorRT Inference Server</td>
<td>ParaView Index</td>
<td></td>
</tr>
<tr>
<td>GROMACS</td>
<td>DIGITS</td>
<td>RAPIDS</td>
<td></td>
<td>ParaView Optix</td>
<td></td>
</tr>
<tr>
<td>LAMMPS</td>
<td>Microsoft Cognitive Toolkit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lattice Microbes</td>
<td>MXNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MILC</td>
<td>NVCaffe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMD</td>
<td>PaddlePaddle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PGI Compilers</td>
<td>PyTorch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PicOnGPU</td>
<td>TensorFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QMCPACK</td>
<td>Theano</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RELION</td>
<td>Torch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SOFTWARE ON THE NGC CONTAINER REGISTRY

10 containers

10 containers

October 2017

42 containers

November 2018
CONTINUOUS IMPROVEMENT
NVIDIA Optimizations Delivers Better Performance on the Same Hardware

Over 12 months, up to 1.8X improvement with mixed-precision on ResNet-50
EASY TO FIND CONTAINERS
Streamlines the NGC User Experience
GET STARTED WITH NGC
Explore the NGC Container Registry

To learn more about all of the GPU-accelerated software available from the NGC container registry, visit: nvidia.com/ngc

Technical information: developer.nvidia.com

Training: nvidia.com/dli

Get Started: ngc.nvidia.com
THE TENSORFLOW CONTAINER CONTENTS
TOOLS YOU NEED FOR AN E2E WORKFLOW

Our session today will cover these items...

Data Loading & Preprocessing
- DALI

Interactive R&D
- Jupyter
- Tensorboard

Training Compute
- CUDA
- cuDNN
- cuBLAS
- Python (2 or 3)

Training Communication
- NCCL
- Horovod
- OpenMPI
- Mellanox OFED

Production Inference
- TensorRT
- TF-TRT
- TRT/IS

As we tour the container, we will point out items that might be of interest

DATA LOADING & PREPROCESSING

NVIDIA Data Loading Library (DALI)

► Full input pipeline acceleration including data loading and augmentation
► Drop-in integration with direct plugins to DL frameworks and open source bindings
► Portable workflows through multiple input formats and configurable graphs
  ► Input Formats - JPEG, LMDB, RecordIO, TFRecord, COCO, H.264, HEVC

Framework Pre-processing – With DALI & nvJPEG

INTERACTIVE R&D

Jupyter and TensorBoard

Welcome to Jupyter!

This repo contains an introduction to Jupyter and Python.

Outline of some basics:
- Notebook Basics
- Python (overview, python)
- Markdown Cells
- Rich Display System
- Custom Display logo
- Running a Secure Public Notebook Server
- How Jupyter works to run code in different languages.

You can also get this tutorial and run it on your laptop:

git clone https://github.com/ipython/ipython-in-depth

Install Python and Jupyter:

with conda:
```bash
conda install ipython jupyter
```

with pip:
```bash
# first, always upgrade pip!
pip install --upgrade pip
pip install --upgrade ipython jupyter
```

Start the notebook in the tutorial directory:
```
cd ipython-in-depth
jupyter notebook
```
CUDA
- The CUDA architecture supports OpenCL and DirectX Compute, C++ and Fortran
- Use GPU to perform general-purpose mathematical calculations increasing computing performance.

cuDNN
- Provides highly tuned implementations for standard routines
- Forward and backward convolution, pooling, normalization, and activation layers.

cuBLAS
- GPU-accelerated implementation of the standard basic linear algebra subroutines
- Speed up applications with compute-intensive operations
- Single GPU or multi-GPU configurations

Python
- Python 2 or Python 3 environments
- Compile Python code for execution on GPUs with Numba from Anaconda
- Speed of a compiled language targeting both CPUs and NVIDIA GPUs
TRAINING COMMUNICATION

NVIDIA Collective Communications Library (NCCL)

- Maximizes performance of collective operations (allreduce, etc.)
- Topology aware for multi-GPU and multi-node

Check out https://devblogs.nvidia.com/scaling-deep-learning-training-nccl/ for more detail!

https://developer.nvidia.com/nccl
TRAINING COMMUNICATION

Horovod

- Open Source, developed by Uber
- Improves communication performance vs Distributed TensorFlow
- Installed into /opt/tensorflow/third_party

More data and graphs like this from Uber at the URL below!

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

- Easily launch multiple instances of a single program!
- HPC standard for distributed computing
- Used by Horovod and NCCL

Mellanox OFED

- Standard for low-latency connections
  - Enables InfiniBand and RDMA!
- Used by MPI and NCCL
- Not typically directly used by users

https://www.open-mpi.org/

PRODUCTION INFERENCE
TensorRT and TensorFlow Integration

- Model optimization right in TensorFlow

...more on this later

https://developer.nvidia.com/tensorrt
THE TENSORFLOW CONTAINER EXAMPLES
LAYOUT

How Container Contents are Organized

► Default directory is /workspace

► README.md files in most places

► Example Dockerfiles in docker-examples
  ► How to add new packages
  ► How to patch TensorFlow

► Additional software installed to /usr/local
  ► /usr/local/bin/jupyter-lab
  ► /usr/local/bin/tensorboard
  ► /usr/local/mpi/bin/mpi

► Examples in /usr/local/nvidia-examples
  ► Runnable TensorFlow examples!
CNN EXAMPLES
/workspace/nvidia-examples/cnn

- Examples implement popular CNN models for single-node training on multi-GPU systems
- Used for benchmarking, or as a starting point for training networks
- Multi-GPU support in scripts provided using Horovod/MPI
- Common utilities for defining CNN networks and performing basic training in nvutils
- /workspace/nvidia-examples/cnn/nvutils is demonstrated in the model scripts.

```python
from __future__ import print_function
from builtins import range
import nvutils
import tensorflow as tf
import argparse

nvutils.init()

default_args = 
```
CNN EXAMPLES - ALEXNET

alexnet.py

► Trivial example of AlexNet

► Uses synthetic data (no dataset needed!)

./alexnet.py 2>/dev/null
CNN EXAMPLES - ALEXNET

alexnet.py

<table>
<thead>
<tr>
<th>Step</th>
<th>Epoch</th>
<th>Img/sec</th>
<th>Loss</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>76.7</td>
<td>6.922</td>
<td>9.996</td>
</tr>
<tr>
<td>10</td>
<td>10.0</td>
<td>844.0</td>
<td>5.879</td>
<td>8.943</td>
</tr>
<tr>
<td>20</td>
<td>20.0</td>
<td>6629.0</td>
<td>5.683</td>
<td>8.728</td>
</tr>
<tr>
<td>30</td>
<td>30.0</td>
<td>6630.8</td>
<td>5.605</td>
<td>8.619</td>
</tr>
<tr>
<td>40</td>
<td>40.0</td>
<td>6665.7</td>
<td>5.585</td>
<td>8.559</td>
</tr>
<tr>
<td>50</td>
<td>50.0</td>
<td>6638.2</td>
<td>5.553</td>
<td>8.489</td>
</tr>
<tr>
<td>60</td>
<td>60.0</td>
<td>6643.6</td>
<td>5.529</td>
<td>8.434</td>
</tr>
<tr>
<td>70</td>
<td>70.0</td>
<td>6640.0</td>
<td>5.520</td>
<td>8.380</td>
</tr>
</tbody>
</table>
CNN EXAMPLES - ALEXNET WITH DATA

alexnet.py

- Run with `-h` to get arguments

- Can specify `--data_dir` to point to ImageNet data

  ```
  ./alexnet.py --data_dir /datasets/imagenet_TFrecords 2>/dev/null
  ```
CNN EXAMPLES - ALEXNET WITH DATA

alexnet.py

```
In [4]: !./alexnet.py --data_dir /datasets/imagenet_TFrecords 2>/tmp/errors

Py 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609]
TF 1.12.0
Script arguments:
   --num_iter 91
   --batch_size 256
   --predict False
   --precision fp16
   --data_dir /datasets/imagenet_TFrecords
   --display_every 10
   --iterator_unit epoch
Training
   Step  Epoch  Img/sec  Loss    LR
   1     0.0    80.1     6.919  9.993  1.00000
   10    0.0    778.1   6.911  9.951  0.99996
   20    0.0    4273.9  6.916  9.836  0.99992
   30    0.0    4476.0  6.909  9.620  0.99987
   40    0.0    3934.0  6.902  9.379  0.99983
   50    0.0    4118.3  6.909  9.157  0.99978
   60    0.0    4016.0  6.909  9.019  0.99971
```
CNN EXAMPLES - INCEPTIONV3

inception_v3.py

► Train InceptionV3 on ImageNet

► Identical invocation to AlexNet example (use -h for help)

`./inception_v3.py --data_dir /datasets/imagenet_TFrecords 2>/dev/null`
CNN EXAMPLES - INCEPTIONV3

inception_v3.py

In [5]: 1./inception_v3.py --data_dir /datasets/imagenet_TFrecords 2>/tmp/errors

```
Patched TensorFlow 1.12.0, revision 0a13f1d8


TF 1.12.0

Script arguments:
  --predict False
  --data_dir /datasets/imagenet_TFrecords
  --precision fp16
  --display_every 10
  --num_iter 90
  --iter_unit epoch
  --batch_size 128

Training

<table>
<thead>
<tr>
<th>Step</th>
<th>Epoch</th>
<th>Img/sec</th>
<th>Loss</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>9.5</td>
<td>7.037</td>
<td>8.149 1.00000</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>69.2</td>
<td>7.241</td>
<td>8.355 0.99998</td>
</tr>
<tr>
<td>20</td>
<td>0.0</td>
<td>460.8</td>
<td>7.128</td>
<td>8.244 0.99996</td>
</tr>
<tr>
<td>30</td>
<td>0.0</td>
<td>451.4</td>
<td>7.027</td>
<td>8.142 0.99994</td>
</tr>
<tr>
<td>40</td>
<td>0.0</td>
<td>451.4</td>
<td>6.953</td>
<td>8.062 0.99991</td>
</tr>
<tr>
<td>50</td>
<td>0.0</td>
<td>455.1</td>
<td>6.984</td>
<td>8.082 0.99989</td>
</tr>
<tr>
<td>60</td>
<td>0.0</td>
<td>465.9</td>
<td>6.963</td>
<td>8.044 0.99987</td>
</tr>
</tbody>
</table>
```
CNN EXAMPLES - RESNET

resnet.py

► Really-really similar to AlexNet and InceptionV3! (and –h works too)

► Can specify --layers to select resnet
  ► E.g., --layers 50 gives ResNet-50

./resnet.py --layers=50 --data_dir=/datasets/imagenet_TFrecords 2>/dev/null

Let’s explore this one in more depth!
CNN EXAMPLES - RESNET

resnet.py

In [7]: $./resnet.py --layers=50 --data_dir=/datasets/imagenet_TFrecords 2>/tmp/errors

PY 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609]
TF 1.12.0
Script arguments:
   --num_iter 90
   --layers 50
   --display_every 10
   --iter_unit epoch
   --data_dir /datasets/imagenet_TFrecords
   --predict False
   --batch_size 256
   --precision fp16

Training

<table>
<thead>
<tr>
<th>Step</th>
<th>Epoch</th>
<th>Img/sec</th>
<th>Loss</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>25.0</td>
<td>7.601</td>
<td>8.572</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>226.7</td>
<td>7.568</td>
<td>8.540</td>
</tr>
<tr>
<td>20</td>
<td>0.0</td>
<td>765.2</td>
<td>7.120</td>
<td>8.095</td>
</tr>
<tr>
<td>30</td>
<td>0.0</td>
<td>861.9</td>
<td>7.060</td>
<td>8.039</td>
</tr>
<tr>
<td>40</td>
<td>0.0</td>
<td>862.1</td>
<td>7.085</td>
<td>8.062</td>
</tr>
<tr>
<td>50</td>
<td>0.0</td>
<td>859.0</td>
<td>7.073</td>
<td>8.047</td>
</tr>
</tbody>
</table>
CNN EXAMPLES - RESNET FP32

resnet.py

► Modern GPUs can use reduced precision
  ► Less memory usage
  ► Higher performance
  ► Can use Tensor Cores!

► --precision Select single or half precision arithmetic. (default:fp16)

./resnet.py --layers=50 --data_dir=/datasets/imagenet_TFrecords
--precision=fp32 2>/dev/null
CNN EXAMPLES - RESNET FP32

resnet.py

In [11]: !./resnet.py --layers=50 --batch_size=256 --data_dir=/datasets/imagenet_TFrecords --precision=fp32 2>/tmp/errors

Python 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609]
TF 1.12.0

Script arguments:
--predict False
--layers 50
--batch_size 256
--precision fp32
--display_every 10
--num_iter 90
--iter_unit epoch
--data_dir /datasets/imagenet_TFrecords

Training
Step Epoch Img/sec Loss LR

Error!??!! Why?
CNN EXAMPLES - RESNET FP32

resnet.py

- Modern GPUs can use reduced precision
  - Less memory usage
  - Higher performance
  - Can use Tensor Cores!

- `--batch_size` Size of each minibatch (default: 256)

```
./resnet.py --layers=50 --batch_size=128
--data_dir=/datasets/imagenet_TFrecords --precision=fp32
2>/dev/null
```
CNN EXAMPLES - RESNET FP32

resnet.py

```
In [9]: !./resnet.py --layers=50 --batch_size=128 --data_dir=/datasets/imagenet_TFrecords --precision=fp32 2>/tmp/errors

Py 3.5.2 (default, Nov 12 2018, 13:43:14)
GCC 5.4.0 20160609
TF 1.12.0
Script arguments:
   --precision fp32
   --display_every 10
   --predict False
   --layers 50
   --num_iter 90
   --batch_size 128
   --iter_unit epoch
   --data_dir /datasets/imagenet_TFrecords

Training
Step Epoch Img/sec Loss  LR
 1   0.0  16.3  7.571  8.543  2.00000
 10  0.0  121.3  8.120  9.092  1.99996
 20  0.0  363.9  7.674  8.650  1.99992
 30  0.0  392.8  7.400  8.380  1.99987
 40  0.0  392.8  7.103  8.085  1.99983
 50  0.0  392.8  6.294  7.201  1.99979
```

NVIDIA®
CNN EXAMPLES - RESNET DALI

resnet.py

► DALI can speed data loading and augmentation
► Also possible to reduce CPU usage for CPU-bound applications
► Needs tfrecords indexed (so DALI can parallelize) with tfrecord2idx

mkdir /imagenet_idx
for x in `ls /datasets/imagenet_TFrecords`; do
tfrecord2idx /datasets/imagenet_TFrecords/$x /datasets/imagenet_idx/$x.idx;
done

► Argument --use_dali enables DALI
► Can specify CPU or GPU to be used by DALI

./resnet.py --layers=50 --data_dir=/datasets/imagenet_TFrecords
--precision=fp16 --data_idx_dir /datasets/imagenet_idx --use_dali
GPU 2>/dev/null
CNN EXAMPLES - RESNET DALI

```python
PY 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609]
TF 1.12.0
Script arguments:
  --num_iter 90
  --display_every 10
  --data_idx_dir /datasets/imagenet_idx
  --iter_unit epoch
  --use_dali GPU
  --predict False
  --layers 50
  --batch_size 256
  --data_dir /datasets/imagenet_TFrecords
  --precision fp16
Training
<table>
<thead>
<tr>
<th>Step</th>
<th>Epoch</th>
<th>Img/sec</th>
<th>Loss</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>22.4</td>
<td>7.700</td>
<td>8.671</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>248.6</td>
<td>7.694</td>
<td>8.666</td>
</tr>
<tr>
<td>20</td>
<td>0.0</td>
<td>810.5</td>
<td>7.415</td>
<td>8.390</td>
</tr>
<tr>
<td>30</td>
<td>0.0</td>
<td>845.8</td>
<td>7.405</td>
<td>8.383</td>
</tr>
</tbody>
</table>
CNN EXAMPLES - A DALI DISCUSSION

resnet.py vs alenet.py

► DALI can speed data loading and augmentation
   ► Resnet-50 without DALI: ~830 images/sec
   ► Resnet-50 with DALI: ~825 images/sec

WHAT? Isn’t DALI supposed to speed things up?

► What about AlexNet?
   ► AlexNet without DALI: ~5100 images/sec
   ► AlexNet with DALI: ~5800 images/sec

./alexnet.py --data_dir=/datasets/imagenet_TFrecords
   --precision=fp16 --data_idx_dir /imagenet_idx --use_dali GPU
   2>/dev/null
CNN EXAMPLES - A DALI DISCUSSION

resnet.py vs alenet.py

In [16]:!./alexnet.py --batch_size 256 --data_dir=/datasets/imagenet_TFrecords --precision=fp16 --data_idx_dir /datasets/imagenet

Py 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609]
TF 1.12.0

Script arguments:
--num_iter 91
--data_dir /datasets/imagenet_TFrecords
--display_every 10
--predict False
--use_dali GPU
--batch_size 256
--iter_unit epoch
--data_idx_dir /datasets/imagenet_idx
--precision fp16

Training

<table>
<thead>
<tr>
<th>Step</th>
<th>Epoch</th>
<th>Img/sec</th>
<th>Loss</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>61.6</td>
<td>6.912</td>
<td>9.986 1.000000</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>745.1</td>
<td>6.927</td>
<td>9.968 0.99996</td>
</tr>
<tr>
<td>20</td>
<td>0.0</td>
<td>5591.9</td>
<td>6.910</td>
<td>9.802 0.99992</td>
</tr>
<tr>
<td>30</td>
<td>0.0</td>
<td>5590.1</td>
<td>6.912</td>
<td>9.583 0.99987</td>
</tr>
<tr>
<td>40</td>
<td>0.0</td>
<td>5599.6</td>
<td>6.915</td>
<td>9.449 0.99963</td>
</tr>
</tbody>
</table>
CNN EXAMPLES - RESNET MULTIGPU

resnet.py

► MPI and Horovod enable multi-GPU training

► Use mpiexec to start processes
  ▶ Being root in the container means some extra mpiexec flags

► Specify number of GPUs to use with --np argument

```bash
mpiexec --allow-run-as-root --bind-to socket -np 8 ./resnet.py
--layers=50 --data_dir=/datasets/imagenet_TFrecords
--precision=fp16 --data_idx_dir /datasets/imagenet_idx --use_dali
gpu 2>/dev/null
```
CNN EXAMPLES - RESNET Multigpu

resnet.py

```
In [17]: !mpiexec --allow-run-as-root --bind-to socket -np 8 /workspace/nvidia-examples/cnn/resnet.py --layers=50 --data_dir=/

--num_iter 90
--use_dali GPU
--iter_unit epoch
--batch_size 256
--predict False
--data_idx_dir /datasets/imagenet_idx
--data_dir /datasets/imagenet_TFrecords
--layers 50

Training
Training
Training
Training
Training
Training
Training
Training
Training
Training

<table>
<thead>
<tr>
<th>Step</th>
<th>Epoch</th>
<th>Img/sec</th>
<th>Loss</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>145.2</td>
<td>7.633</td>
<td>8.605</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1524.2</td>
<td>7.346</td>
<td>8.317</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>5221.9</td>
<td>7.185</td>
<td>8.156</td>
</tr>
</tbody>
</table>
```
OTHER EXAMPLES
Not just CNNs!

- OpenSeq2Seq
  - Sequence-to-Sequence toolkit
  - [https://nvidia.github.io/OpenSeq2Seq](https://nvidia.github.io/OpenSeq2Seq)

- Big LSTM
  - Language Modeling examples
  - [https://github.com/rafaljozefowicz/lm](https://github.com/rafaljozefowicz/lm)
BUILDING A WORKFLOW
TOOLS YOU NEED FOR AN E2E WORKFLOW

Our session today will cover these items...

Data Loading & Preprocessing
- DALI

Interactive R&D
- Jupyter
- Tensorboard

Training Compute
- CUDA
- cuDNN
- cuBLAS
- Python (2 or 3)

Training Communication
- NCCL
- Horovod
- OpenMPI
- Mellanox OFED

Production Inference
- TensorRT
- TF-TRT
- TRT/IS

As we tour the container, we will point out items that might be of interest

TENSORRT INTEGRATED WITH TENSORFLOW

Speed up TensorFlow inference with TensorRT optimizations

Speed up TensorFlow model inference with TensorRT with new TensorFlow APIs

Simple API to use TensorRT within TensorFlow easily

Sub-graph optimization with fallback offers flexibility of TensorFlow and optimizations of TensorRT

Optimizations for FP32, FP16 and INT8 with use of Tensor Cores automatically

# Apply TensorRT optimizations
trt_graph = trt.create_inference_graph(frozen_graph_def, output_node_name, max_batch_size=batch_size, max_workspace_size_bytes=workspace_size, precision_mode=precision)

# INT8 specific graph conversion
trt_graph = trt.calib_graph_to_infer_graph(calibGraph)

Available from TensorFlow 1.7
https://github.com/tensorflow/tensorflow

developer.nvidia.com/tensorrt
TENSORRT INFEERENCE SERVER

Containerized Microservice for Data Center Inference

Multiple models scalable across GPUs

Supports all popular AI frameworks

Seamless integration into DevOps deployments leveraging Docker and Kubernetes

Ready-to-run container, free from the NGC container registry
PRODUCTION INFERENC
Putting the trained model to work, monitor and scale

Data Volume

TensorFlow

Model Store
Config management and other tools

TensorBoard
Monitoring

TensorRT Inference Server
Server Container

TensorRT Inference Server
Client Container

Plus your own customization!
PRODUCTION INFERENCETensorFlow and TensorRT

1. Use TensorRT Inference Server to serve native TensorFlow model
   a. What is the performance?

2. Freeze TensorFlow model and optimize with TensorRT

3. Use TensorRT Inference Server to serve optimized TensorRT model
   a. What is the performance?
WHAT NEXT?
SUMMARY & NEXT STEPS

Where to go next?

► Login to the NVIDIA GPU Cloud
   https://nvidia.com/ngc

► Pull the TensorFlow container to your local system
   docker pull nvcr.io/nvidia/tensorflow:19.02-py3

► Explore the examples!
  ► The Jupyter notebook and these slides will be on the GTC Website

► Train a model on your own data

► Experiment with your model and the TensorRT Inference Server
   docker pull nvcr.io/nvidia/tensorrtserver:19.02-py3

Learn and have fun!