TENSORFLOW

RN-08467-001_v20.03 | April 2020

Release Notes
# TABLE OF CONTENTS

Chapter 1. TensorFlow Overview.......................................................................................... 1
Chapter 2. Pulling A Container............................................................................................. 2
Chapter 3. Running TensorFlow.......................................................................................... 3
Chapter 4. TensorFlow Release 20.03................................................................................. 5
Chapter 5. TensorFlow Release 20.02............................................................................... 12
Chapter 6. TensorFlow Release 20.01............................................................................... 19
Chapter 7. TensorFlow Release 19.12............................................................................... 26
Chapter 8. TensorFlow Release 19.11............................................................................... 33
Chapter 9. TensorFlow Release 19.10............................................................................... 41
Chapter 10. TensorFlow Release 19.09............................................................................. 47
Chapter 11. TensorFlow Release 19.08............................................................................. 53
Chapter 12. TensorFlow Release 19.07............................................................................. 59
Chapter 13. TensorFlow Release 19.06............................................................................. 65
Chapter 14. TensorFlow Release 19.05............................................................................. 71
Chapter 15. TensorFlow Release 19.04............................................................................. 76
Chapter 16. TensorFlow Release 19.03............................................................................. 81
Chapter 17. TensorFlow Release 19.02............................................................................. 85
Chapter 18. TensorFlow Release 19.01............................................................................. 88
Chapter 19. TensorFlow Release 18.12............................................................................. 91
Chapter 20. TensorFlow Release 18.11............................................................................. 94
Chapter 21. TensorFlow Release 18.10.......................................................................... 101
Chapter 22. TensorFlow Release 18.09.......................................................................... 105
Chapter 23. TensorFlow Release 18.08.......................................................................... 108
Chapter 24. TensorFlow Release 18.07.......................................................................... 110
Chapter 25. TensorFlow Release 18.06.......................................................................... 113
Chapter 26. TensorFlow Release 18.05.......................................................................... 116
Chapter 27. TensorFlow Release 18.04.......................................................................... 118
Chapter 28. TensorFlow Release 18.03.......................................................................... 120
Chapter 29. TensorFlow Release 18.02.......................................................................... 122
Chapter 30. TensorFlow Release 17.12.......................................................................... 124
Chapter 31. TensorFlow Release 17.11.......................................................................... 126
Chapter 32. TensorFlow Release 17.10.......................................................................... 128
Chapter 33. TensorFlow Release 17.09.......................................................................... 130
Chapter 34. TensorFlow Release 17.07.......................................................................... 132
Chapter 35. TensorFlow Release 17.06.......................................................................... 133
Chapter 36. TensorFlow Release 17.05.......................................................................... 134
Chapter 37. TensorFlow Release 17.04.......................................................................... 136
Chapter 38. TensorFlow Release 17.03.......................................................................... 138
Chapter 39. TensorFlow Release 17.02.......................................................................... 139
Chapter 41. TensorFlow Release 17.01................................................................. 140
Chapter 42. TensorFlow Release 16.12................................................................. 141
Chapter 1.
TENSORFLOW OVERVIEW

The NVIDIA Deep Learning SDK accelerates widely-used deep learning frameworks such as TensorFlow™.

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture lets you deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code.

TensorFlow was originally developed by researchers and engineers working on the Google Brain team within Google’s Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

See /workspace/README.md inside the container for information on customizing your TensorFlow image. For more information about TensorFlow, including tutorials, documentation, and examples, see:

- TensorFlow tutorials
- TensorFlow API

This document describes the key features, software enhancements and improvements, any known issues, and how to run this container.
Chapter 2.
PULLING A CONTAINER

Before you can pull a container from the NGC container registry, you must have Docker installed. For DGX users, this is explained in Preparing to use NVIDIA Containers Getting Started Guide.

For users other than DGX, follow the NVIDIA® GPU Cloud™ (NGC) container registry installation documentation based on your platform.

You must also have access and be logged into the NGC container registry as explained in the NGC Getting Started Guide.

The deep learning frameworks are stored in the following repository where you can find the NGC Docker containers.

nvcr.io/nvidia

The deep learning framework containers are stored in the nvcr.io/nvidia repository.
Chapter 3.  
RUNNING TENSORFLOW

Before you can run an NGC deep learning framework container, your Docker environment must support NVIDIA GPUs. To run a container, issue the appropriate command as explained in the Running A Container chapter in the NVIDIA Containers And Frameworks User Guide and specify the registry, repository, and tags.

On a system with GPU support for NGC containers, the following occurs when running a container:

- The Docker engine loads the image into a container which runs the software.
- You define the runtime resources of the container by including additional flags and settings that are used with the command. These flags and settings are described in Running A Container.
- The GPUs are explicitly defined for the Docker container (defaults to all GPUs, but can be specified using NVIDIA_VISIBLE_DEVICES environment variable). Starting in Docker 19.03, follow the steps as outlined below. For more information, refer to the nvidia-docker documentation here.

The method implemented in your system depends on the DGX OS version installed (for DGX systems), the specific NGC Cloud Image provided by a Cloud Service Provider, or the software that you have installed in preparation for running NGC containers on TITAN PCs, Quadro PCs, or vGPUs.

1. Issue the command for the applicable release of the container that you want. The following command assumes you want to pull the latest container.

   **For TensorFlow version 2.x**

   ```bash
   docker pull nvcr.io/nvidia/tensorflow:20.03-tf2-py3
   ```

   Or

   **For TensorFlow version 1.x**

   ```bash
   docker pull nvcr.io/nvidia/tensorflow:20.03-tf1-py3
   ```
2. Open a command prompt and paste the pull command. The pulling of the container image begins. Ensure the pull completes successfully before proceeding to the next step.

3. Run the container image.

If you have Docker 19.03 or later, a typical command to launch the container is:

```
docker run --gpus all -it --rm -v local_dir:container_dir nvcr.io/nvidia/tensorflow:<xx.xx>-tf<x>-py<x>
```

If you have Docker 19.02 or earlier, a typical command to launch the container is:

```
nvidia-docker run -it --rm -v local_dir:container_dir nvcr.io/nvidia/tensorflow:<xx.xx>-tf<x>-py<x>
```

TensorFlow is run by importing it as a Python module:

```
$ python
   >>> import tensorflow as tf
   >>> print(tf.__version__)
1.15.0
```

You might want to pull in data and model descriptions from locations outside the container for use by TensorFlow or save results to locations outside the container. To accomplish this, the easiest method is to mount one or more host directories as Docker® data volumes.

In order to share data between GPUs, NVIDIA® Collective Communications Library™ (NCCL) may require shared system memory for IPC and pinned (page-locked) system memory resources. The operating system's limits on these resources may need to be increased accordingly. Refer to your system's documentation for details.

In particular, Docker containers default to limited shared and pinned memory resources. When using NCCL inside a container, it is recommended that you increase these resources by issuing:

```
--shm-size=1g --ulimit memlock=-1
```

in the command line to:

```
docker run --gpus all
```
Chapter 4.
TENSORFLOW RELEASE 20.03

The NVIDIA container image of TensorFlow, release 20.03, is available on NGC.

**Contents of the TensorFlow container**

This container image includes the complete source of the NVIDIA version of TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- **Ubuntu 18.04**

  Container image `20.03-tf1-py3` and `20.03-tf2-py3` contains **Python 3.6**

  - NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
  - NVIDIA cuDNN 7.6.5
  - NVIDIA NCCL 2.6.3 (optimized for NVLink™)
  - Horovod 0.19.0
  - OpenMPI 3.1.4
  - TensorBoard
    - `20.03-tf1-py3` includes version 1.15.0+nv
    - `20.03-tf2-py3` includes version 2.1.1
  - MLNX_OFED
  - OpenSeq2Seq at commit a81babd
    - Included only in `20.03-tf1-py3`
  - TensorRT 7.0.0
  - DALI 0.19.0
- DLProf 0.10.0
  - Included only in 20.03-tf1-py3
- Nsight Compute 2019.5.0
- Nsight Systems 2020.1.1
- XLA-Lite
- Tensor Core optimized examples: (Included only in 20.03-tf1-py3)
  - U-Net Medical
  - SSD320 v1.2
  - Neural Collaborative Filtering (NCF)
  - BERT
  - U-Net Industrial Defect Segmentation
  - GNMT v2
  - ResNet-50 v1.5
- Jupyter and JupyterLab:
  - Jupyter Client 6.0.0
  - Jupyter Core 4.6.1
  - Jupyter Notebook
    - 20.03-tf1-py3 includes version 6.0.2
    - 20.03-tf2-py3 includes version 5.7.8
  - JupyterLab
    - 20.03-tf1-py3 includes version 1.2.2
    - 20.03-tf2-py3 includes version 1.0.2
  - JupyterLab Server
    - 20.03-tf1-py3 includes version 1.0.7
    - 20.03-tf2-py3 includes version 1.0.0
  - Jupyter-TensorBoard

**Driver Requirements**

Release 20.03 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.33.01. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

**GPU Requirements**

Release 20.03 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this
compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

**Key Features and Enhancements**

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 20.03 is based on TensorFlow 1.15.2 and TensorFlow 2.1.0.
- The latest version of NVIDIA NCCL 2.6.3 (optimized for NVLink™)
- The latest version of DALI 0.19.0
- The latest version of DLProf 0.10.0
- When the environment variable TF_DETERMINISTIC_OPS is set to ‘true’ or ‘1’, bilinear resizing will operate deterministically in both the forward and backward directions. In the TF 1 container image variant, the default way of accessing this functionality is via tf.image.resize_bilinear. In the TF 2 container image variant, the default way of accessing this functionality is via tf.image.resize with method=ResizeMethod.BILINEAR (which is the default method setting). This feature is also exposed through tf.keras.layers.UpSampling2D with interpolation='bilinear' (which is not the default interpolation setting). Enabling determinism may reduce performance. For more information, see NVIDIA's tensorflow-determinism repository on GitHub.
- Ubuntu 18.04 with February 2020 updates

**Announcements**

- Python 2.7 is no longer supported in this TensorFlow container release.
- The TF_ENABLE.AUTO_MIXED_PRECISION environment variables are no longer supported in the tf2 container because it is not possible to automatically enable loss scaling in many cases in the tf 2.x API. Instead tf.train.experimental.enable_mixed_precision_graph_rewrite() should be used to enable AMP.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

**NVIDIA TensorFlow Container Versions**

The following table shows what versions of Ubuntu, CUDA, TensorFlow, and TensorRT are supported in each of the NVIDIA containers for TensorFlow. For older container versions, refer to the Frameworks Support Matrix.
Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Known Issues

- We have observed a regression in the performance of certain TF-TRT benchmarks in TensorFlow 1.15 including image classification models with precision INT8. We are still investigating this. Since 19.11 comes with a new version of TensorFlow (1.15), which includes a lot of changes in the TensorFlow backend, it’s very possible that the regression is caused by a change in the TensorFlow backend.
- CUDA 10.2 and NCCL 2.5.x libraries require slightly more device memory than previous releases. As a result, some models that ran previously may exhaust device memory.
- The accuracy of Faster RCNN with the backbone ResNet-50 using TensorRT6.0 INT8 calibration is lower than expected. This will be fixed in future releases of TensorRT.
- The following warning is issued when the method `build()` from the API is not called. This warning can be ignored.

```
OP_REQUIRES failed at trt_engine_resource_ops.cc:183 : Not found: Container TF-TRT does not exist. (Could not find resource: TF-TRT/TRTEngineOp_...
```
The following warning is issued because internally TensorFlow calls the TensorRT optimizer for certain objects unnecessarily. This warning can be ignored.

```
 TensorRTOptimizer is probably called on funcdef! This optimizer must *NOT* be called on function objects.
```

- We have seen failures when using INT8 calibration (post-training) within the same process that does FP32/FP16 conversion. We recommend using separate processes for different precisions until this issue gets resolved.
- We have seen failures when calling the TensorRT optimizer on models that are already optimized by TensorRT. This issue will be fixed in a future release.
- In case you import nets from `models.slim`, you might see the following error:

```
 AttributeError: module 'tensorflow_core.contrib' has no attribute 'tensorrt'
```

Changing the order of imports can fix the issue. Therefore, import TensorRT before importing nets as follows:

```
import tensorflow.contrib.tensorrt as trt
import nets.nets_factory
```

**Automatic Mixed Precision (AMP)**

Automatic mixed precision converts certain float32 operations to operate in float16, which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using an optimizer from `tf.train` or `tf.keras.optimizers` for both `compute_gradients()` and `apply_gradients()` operations (for example, by calling `optimizer.minimize()` or `model.fit()`), automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`.

For more information on this function, see the TensorFlow documentation [here](https://www.tensorflow.org/api_docs/python/tf/train/GraphOptions#mixed_precision_graph_rewrite).


**Tensor Core Examples**

The [tensor core examples provided in GitHub](https://github.com/nvidia/tensorflow) focus on achieving the best performance and convergence by using the latest [deep learning example](https://www.tensorflow.org/api_docs/python/tf/keras/examples) networks and [model scripts](https://www.tensorflow.org/api_docs/python/tf/keras/models) for training.
Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper [U-Net: Convolutional Networks for Biomedical Image Segmentation](https://arxiv.org/abs/1505.04597), without any alteration. This model script is available on [GitHub](https://github.com) as well as [NVIDIA GPU Cloud (NGC)](https://ngc.nvidia.com).

- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the [SSD: Single Shot MultiBox Detector](https://arxiv.org/abs/1512.02325) paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on [GitHub](https://github.com) as well as [NVIDIA GPU Cloud (NGC)](https://ngc.nvidia.com).

- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item. This model script is available on [GitHub](https://github.com) as well as [NVIDIA GPU Cloud (NGC)](https://ngc.nvidia.com).

- **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](https://arxiv.org/abs/1810.04805) paper. NVIDIA's BERT is an optimized version of [Google’s official implementation](https://github.com/google-research/bert), leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on [GitHub](https://github.com) as well as [NVIDIA GPU Cloud (NGC)](https://ngc.nvidia.com).

- **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: [U-Net: Convolutional Networks for Biomedical Image Segmentation](https://arxiv.org/abs/1505.04597). This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset [DAGM2007](http://dagm2007.mi.tum.de). This model script is available on [GitHub](https://github.com) as well as [NVIDIA GPU Cloud (NGC)](https://ngc.nvidia.com).

- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the [Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](https://www.aclweb.org/anthology/W16-2200) paper. The most important difference between the two
models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

**Known Issues**

- There are known issues since the 19.11 release for NCF inference with XLA and VGG16 training without XLA; these benchmarks have performance that is lower than expected.
- There are known issues regarding TF-TRT INT8 accuracy issues. See the *Accelerating Inference In TensorFlow With TensorRT (TF-TRT)* section above for more information.
- TensorFlow has a bug in its IPv6 support which can result in the following error: *Tensorboard could not bind to unsupported address family ::*. To workaround this error, pass the `--host <IP>` flag when starting TensorBoard.
- A known issue in TensorFlow results in the error *Cannot take the length of Shape with unknown rank* when training variable sized images with the Keras `model.fit` API. Details are provided [here](#) and a fix will be available in a future release.
- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.
- There is a known issue when your NVIDIA driver release is older than 418.xx since the 19.10 release, the Nsight Systems profiling tool (for example, the `nsys`) might cause **CUDA runtime API error.** A fix will be included in a future release.
Chapter 5.
TENSORFLOW RELEASE 20.02

The NVIDIA container image of TensorFlow, release 20.02, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 18.04

  Container image 20.02-tf1-py3 and 20.02-tf2-py3 contains Python 3.6

- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- Horovod 0.19.0
- OpenMPI 3.1.4
- TensorBoard
  - 20.02-tf1-py3 includes version 1.15.0+nv
  - 20.02-tf2-py3 includes version 2.1.0
- MLNX_OFED
- OpenSeq2Seq at commit a81babd
  - Included only in 20.02-tf1-py3
- TensorRT 7.0.0
- DALI 0.18.0 Beta
- **DLProf 20.02**
  - Included only in `20.02-tf1-py3`
- **Nsight Compute 2019.5.0**
- **Nsight Systems 2020.1.1**
- **XLA-Lite**
- **Tensor Core optimized examples:** (Included only in `20.02-tf1-py3`)
  - **U-Net Medical**
  - **SSD320 v1.2**
  - **Neural Collaborative Filtering (NCF)**
  - **BERT**
  - **U-Net Industrial Defect Segmentation**
  - **GNMT v2**
  - **ResNet-50 v1.5**
- **Jupyter and JupyterLab:**
  - **Jupyter Client 5.3.4**
  - **Jupyter Core 4.6.1**
  - **Jupyter Notebook**
    - `20.02-tf1-py3` includes version **6.0.2**
    - `20.02-tf2-py3` includes version **5.7.8**
  - **JupyterLab**
    - `20.02-tf1-py3` includes version **1.2.2**
    - `20.02-tf2-py3` includes version **1.0.2**
  - **JupyterLab Server**
    - `20.02-tf1-py3` includes version **1.0.6**
    - `20.02-tf2-py3` includes version **1.0.0**
  - **Jupyter-TensorBoard**

**Driver Requirements**

Release 20.02 is based on **NVIDIA CUDA 10.2.89**, which requires **NVIDIA Driver** release 440.33.01. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the **CUDA Application Compatibility** topic. For more information, see **CUDA Compatibility and Upgrades**.

**GPU Requirements**

Release 20.02 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this
compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 20.02 is based on TensorFlow 1.15.2 and TensorFlow 2.1.0.
- Latest version of DLProf 20.02
- Latest version of DALI 0.18.0 Beta
- 20.02-tf2-py3 includes version 2.1.0
- Latest version of Nsight Systems 2020.1.1
- Latest version of Horovod 0.19.0
- Ubuntu 18.04 with January 2020 updates
- Improved AMP logging messages to include instructions for tweaking AMP lists.
- Added nvtx markers in TF 2.1 eager execution path for improved profiling with nvtx.

Announcements

- Python 2.7 is no longer supported in this TensorFlow container release.
- The TF_ENABLE_AUTO_MIXED_PRECISION environment variables are no longer supported in the tf2 container because it is not possible to automatically enable loss scaling in many cases in the tf 2.x API. Instead tf.train.experimental.enable_mixed_precision_graph_rewrite() should be used to enable AMP.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

NVIDIA TensorFlow Container Versions

The following table shows what versions of Ubuntu, CUDA, TensorFlow, and TensorRT are supported in each of the NVIDIA containers for TensorFlow. For older container versions, refer to the Frameworks Support Matrix.

<table>
<thead>
<tr>
<th>Container Version</th>
<th>Ubuntu</th>
<th>CUDA Toolkit</th>
<th>TensorFlow</th>
<th>TensorRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.02</td>
<td>18.04</td>
<td>NVIDIA CUDA 10.2.89</td>
<td>2.1.0</td>
<td>TensorRT 7.0.0</td>
</tr>
<tr>
<td></td>
<td>16.04</td>
<td></td>
<td>1.15.2</td>
<td></td>
</tr>
<tr>
<td>20.01</td>
<td></td>
<td></td>
<td>2.0.0</td>
<td></td>
</tr>
<tr>
<td>19.12</td>
<td></td>
<td></td>
<td>1.15.0</td>
<td>TensorRT 6.0.1</td>
</tr>
</tbody>
</table>
Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see [Accelerating Inference In TensorFlow With TensorRT User Guide](#).

**Known Issues**

- We have observed a regression in the performance of certain TF-TRT benchmarks in TensorFlow 1.15 including image classification models with precision INT8. We are still investigating this. Since 19.11 comes with a new version of TensorFlow (1.15), which includes a lot of changes in the TensorFlow backend, it's very possible that the regression is caused by a change in the TensorFlow backend.

- CUDA 10.2 and NCCL 2.5.x libraries require slightly more device memory than previous releases. As a result, some models that ran previously may exhaust device memory.

- The accuracy of Faster RCNN with the backbone ResNet-50 using TensorRT6.0 INT8 calibration is lower than expected. This will be fixed in future releases of TensorRT.

- The following warning is issued when the method build() from the API is not called. This warning can be ignored.

  ```
  OP_REQUIRES failed at trt_engine_resource_ops.cc:183 : Not found: Container TF-TRT does not exist. (Could not find resource: TF-TRT/TRTEngineOp_...
  ```

- The following warning is issued because internally TensorFlow calls the TensorRT optimizer for certain objects unnecessarily. This warning can be ignored.

  ```
  TensorRTOptimizer is probably called on funcdef! This optimizer must *NOT* be called on function objects.
  ```

- We have seen failures when using INT8 calibration (post-training) within the same process that does FP32/FP16 conversion. We recommend using separate processes for different precisions until this issue gets resolved.

- We have seen failures when calling the TensorRT optimizer on models that are already optimized by TensorRT. This issue will be fixed in a future release.

- In case you import nets from `models/slim`, you might see the following error:
AttributeError: module 'tensorflow_core.contrib' has no attribute 'tensorrt'

Changing the order of imports can fix the issue. Therefore, import TensorRT before importing nets as follows:

```
import tensorflow.contrib.tensorrt as trt
import nets.nets_factory
```

**Automatic Mixed Precision (AMP)**

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using an optimizer from `tf.train` or `tf.keras.optimizers` for both `compute_gradients()` and `apply_gradients()` operations (for example, by calling `optimizer.minimize()` or `model.fit()`), automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`.

For more information on this function, see the TensorFlow documentation [here](#).

For backward compatibility with previous container releases, AMP can also be enabled for `tf.train` optimizers by defining the following environment variable:

```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the TensorFlow User Guide, along with Training With Mixed Precision.

**Tensor Core Examples**

The tensor core examples provided in GitHub focus on achieving the best performance and convergence by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation,
without any alteration. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUS for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck
blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

**Known Issues**

- There are known issues since the 19.11 release for NCF inference with XLA and VGG16 training without XLA; these benchmarks have performance that is lower than expected.
- There are known issues regarding TF-TRT INT8 accuracy issues. See the Accelerating Inference In TensorFlow With TensorRT (TF-TRT) section above for more information.
- For BERT Large training with the 19.08 release on Tesla V100 boards with 16 GB memory, performance with batch size 3 per GPU is lower than expected; batch size 2 per GPU may be a better choice for this model on these GPUs with the 19.08 release. 32 GB GPUs are not affected.
- TensorBoard has a bug in its IPv6 support which can result in the following error: ***Tensorboard could not bind to unsupported address family ::*** To workaround this error, pass the **--host <IP>** flag when starting TensorBoard.
- A known issue in TensorFlow results in the error **Cannot take the length of Shape with unknown rank** when training variable sized images with the Keras model.fit API. Details are provided [here](#) and a fix will be available in a future release.
- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.
- There is a known issue when your NVIDIA driver release is older than 418.xx since the 19.10 release, the Nsight Systems profiling tool (for example, the nsys) might cause **CUDA runtime API error**. A fix will be included in a future release.
Chapter 6.
TENSORFLOW RELEASE 20.01

The NVIDIA container image of TensorFlow, release 20.01, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 18.04
- Container image:
  - 20.01-tf1-py2 contains Python 2.7
  - 20.01-tf1-py3 and 20.01-tf2-py3 contains Python 3.6
- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- Horovod 0.18.2
- OpenMPI 3.1.4
- TensorBoard
  - 20.01-tf1-py2 and 20.01-tf1-py3 include version 1.15.0+nv
  - 20.01-tf2-py3 includes version 2.0.1
- MLNX_OFED
- OpenSeq2Seq at commit a81babd
  - Included only in 20.01-tf1-py2 and 20.01-tf1-py3
TensorRT 7.0.0
DALI 0.17.0 Beta
DLProf 20.01
  • Included only in 20.01-tf1-py2 and 20.01-tf1-py3
Nsight Compute 2019.5.0
Nsight Systems 2019.6.1
XLA-Lite
Tensor Core optimized examples: (Included only in 20.01-tf1-py2 and 20.01-tf1-py3)
  • U-Net Medical
  • SSD320 v1.2
  • Neural Collaborative Filtering (NCF)
  • BERT
  • U-Net Industrial Defect Segmentation
  • GNMT v2
  • ResNet-50 v1.5
Jupyter and JupyterLab:
  • Jupyter Client 5.3.4
  • Jupyter Core 4.6.1
  • Jupyter Notebook
    • 20.01-tf1-py2 and 20.01-tf1-py3 includes version 6.0.2
    • 20.01-tf2-py3 includes version 5.7.8
  • JupyterLab
    • 20.01-tf1-py2 and 20.01-tf1-py3 includes version 1.2.2
    • 20.01-tf2-py3 includes version 1.0.2
  • JupyterLab Server
    • 20.01-tf1-py2 and 20.01-tf1-py3 includes version 1.0.6
    • 20.01-tf2-py3 includes version 1.0.0
  • Jupyter-TensorBoard

Driver Requirements
Release 20.01 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.33.01. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.
GPU Requirements

Release 20.01 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

‣ TensorFlow container image version 20.01 is based on TensorFlow 1.15.0 and TensorFlow 2.0.0.
‣ Latest version of TensorRT 7.0.0
‣ Latest version of DALI 0.17.0 Beta
‣ Latest version of DLProf 20.01
‣ XLA-Lite, reduced featured version of XLA focused on stable GPU performance
‣ Ubuntu 18.04 with December 2019 updates

Announcements

‣ We will stop support for Python 2.7 in the next TensorFlow container release.
‣ Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

NVIDIA TensorFlow Container Versions

The following table shows what versions of Ubuntu, CUDA, TensorFlow, and TensorRT are supported in each of the NVIDIA containers for TensorFlow. For older container versions, refer to the Frameworks Support Matrix.

<table>
<thead>
<tr>
<th>Container Version</th>
<th>Ubuntu</th>
<th>CUDA Toolkit</th>
<th>TensorFlow</th>
<th>TensorRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.01</td>
<td>18.04</td>
<td>NVIDIA CUDA 10.2.89</td>
<td>2.0.0</td>
<td>TensorRT 7.0.0</td>
</tr>
<tr>
<td>19.12</td>
<td>16.04</td>
<td></td>
<td>1.15.0</td>
<td>TensorRT 6.0.1</td>
</tr>
<tr>
<td>19.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19.10</td>
<td></td>
<td>NVIDIA CUDA 10.1.243</td>
<td>1.14.0</td>
<td>TensorRT 5.1.5</td>
</tr>
<tr>
<td>19.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Fixed Issues

‣ Fixed a bug in Deconvolution (conv2d_transpose) that caused wrong outputs.

Known Issues

‣ We have observed a regression in the performance of certain TF-TRT benchmarks in TensorFlow 1.15 including image classification models with precision INT8. We are still investigating this. Since 19.11 comes with a new version of TensorFlow (1.15), which includes a lot of changes in the TensorFlow backend, it’s very possible that the regression is caused by a change in the TensorFlow backend.

‣ CUDA 10.2 and NCCL 2.5.x libraries require slightly more device memory than previous releases. As a result, some models that ran previously may exhaust device memory.

‣ The accuracy of Faster RCNN with the backbone ResNet-50 using TensorRT6.0 INT8 calibration is lower than expected. This will be fixed in future releases of TensorRT.

‣ The following warning is issued when the method `build()` from the API is not called. This warning can be ignored.

    OP_REQUIRES failed at trt_engine_resource_ops.cc:183 : Not found: Container TF-TRT does not exist. (Could not find resource: TF-TRT/TRTEngineOp...)

‣ The following warning is issued because internally TensorFlow calls the TensorRT optimizer for certain objects unnecessarily. This warning can be ignored.

    TensorRTOptimizer is probably called on funcdef! This optimizer must *NOT* be called on function objects.

‣ We have seen failures when using INT8 calibration (post-training) within the same process that does FP32/FP16 conversion. We recommend using separate processes for different precisions until this issue gets resolved.

‣ We have seen failures when calling the TensorRT optimizer on models that are already optimized by TensorRT. This issue will be fixed in a future release.

‣ In case you import nets from models/slim, you might see the following error:

    AttributeError: module 'tensorflow_core.contrib' has no attribute 'tensorrt'

Changing the order of imports can fix the issue. Therefore, import TensorRT before importing nets as follows:

    import tensorflow.contrib.tensorrt as trt
    import nets.nets_factory
Automatic Mixed Precision (AMP)

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using an optimizer from `tf.train` or `tf.keras.optimizers` for both `compute_gradients()` and `apply_gradients()` operations (for example, by calling `optimizer.minimize()` or `model.fit()`), automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`.

For more information on this function, see the TensorFlow documentation here.

For backward compatibility with previous container releases, AMP can also be enabled for `tf.train` optimizers by defining the following environment variable:

```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the TensorFlow User Guide, along with Training With Mixed Precision.

Tensor Core Examples

The tensor core examples provided in GitHub focus on achieving the best performance and convergence by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation, without any alteration. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
Known Issues

- There are known issues since the 19.11 release for NCF inference with XLA and VGG16 training without XLA; these benchmarks have performance that is lower than expected.
- There are known issues regarding TF-TRT INT8 accuracy issues. See the *Accelerating Inference In TensorFlow With TensorRT (TF-TRT)* section above for more information.
- For BERT Large training with the 19.08 release on Tesla V100 boards with 16 GB memory, performance with batch size 3 per GPU is lower than expected; batch size 2 per GPU may be a better choice for this model on these GPUs with the 19.08 release. 32 GB GPUs are not affected.
- TensorBoard has a bug in its IPv6 support which can result in the following error: `Tensorboard could not bind to unsupported address family ::`. To workaround this error, pass the `--host <IP>` flag when starting TensorBoard.
- A known issue in TensorFlow results in the error `Cannot take the length of Shape with unknown rank` when training variable sized images with the Keras `model.fit` API. Details are provided [here](#) and a fix will be available in a future release.
- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.
- There is a known issue when your NVIDIA driver release is older than 418.xx since the 19.10 release, the Nsight Systems profiling tool (for example, the `nsys`) might cause CUDA runtime API error. A fix will be included in a future release.
Chapter 7.
TENSORFLOW RELEASE 19.12

The NVIDIA container image of TensorFlow, release 19.12, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 18.04

  Container image:
  - 19.12-tf1-py2 contains Python 2.7
  - 19.12-tf1-py3 and 19.12-tf2-py3 contains Python 3.6

- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- Horovod 0.18.2
- OpenMPI 3.1.4
- TensorBoard
  - 19.12-tf1-py2 and 19.12-tf1-py3 include version 1.15.0+nv
  - 19.12-tf2-py3 includes version 2.0.1
- MLNX_OFED
- OpenSeq2Seq at commit a81babd
  - Included only in 19.12-tf1-py2 and 19.12-tf1-py3
TensorRT 6.0.1
DALI 0.16.0 Beta
DLProf 19.12
  › Included only in 19.12-tf1-py2 and 19.12-tf1-py3
Nsight Compute 2019.5.0
Nsight Systems 2019.6.1
Tensor Core optimized examples: (Included only in 19.12-tf1-py2 and 19.12-tf1-py3)
  › U-Net Medical
  › SSD320 v1.2
  › Neural Collaborative Filtering (NCF)
  › BERT
  › U-Net Industrial Defect Segmentation
  › GNMT v2
  › ResNet-50 v1.5
Jupyter and JupyterLab:
  › Jupyter Client 5.3.4
  › Jupyter Core 4.6.1
  › Jupyter Notebook
    › 19.12-tf1-py2 and 19.12-tf1-py3 includes version 6.0.2
    › 19.12-tf2-py3 includes version 5.7.8
  › JupyterLab
    › 19.12-tf1-py2 and 19.12-tf1-py3 includes version 1.2.2
    › 19.12-tf2-py3 includes version 1.0.2
  › JupyterLab Server
    › 19.12-tf1-py2 and 19.12-tf1-py3 includes version 1.0.6
    › 19.12-tf2-py3 includes version 1.0.0
  › Jupyter-TensorBoard

**Driver Requirements**

Release 19.12 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.30. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410, 418.xx or 440.30. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.
GPU Requirements
Release 19.12 supports CUDA compute capability 6.0 and higher. This corresponds to
gPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this
compute capability corresponds to, see CUDA GPUs. For additional support details, see
Deep Learning Frameworks Support Matrix.

Key Features and Enhancements
This TensorFlow release includes the following key features and enhancements.

‣ TensorFlow container image version 19.12 is based on TensorFlow 1.15.0 and
TensorFlow 2.0.0.
‣ Latest version of DALI 0.16.0 Beta
‣ Latest version of DLProf 19.12
‣ Latest version of Horovod 0.18.2
‣ Latest version of Nsight Systems 2019.6.1
‣ Latest version of TensorBoard for 19.12-tf2-py3 includes version 2.0.2
‣ Jupyter Notebook, JupyterLab, and JupyterLab Server versions are now specific to
which TensorFlow container version you choose to use.
‣ Added optimized GenerateBoxProposals op for object detection models.
‣ Deterministic cuDNN convolutions, enabled via TF_CUDNN_DETERMINISTIC
or TF_DETERMINISTIC_OPS are now available on a wider range of layer
configurations. Prior to this version, some layer configurations would result in an
exception with the message No algorithm worked!
‣ Ubuntu 18.04 with November 2019 updates

Announcements
‣ We will stop support for Python 2.7 in a future TensorFlow container release.
‣ Deep learning framework containers 19.11 and later include experimental support
for Singularity v3.0.

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)
For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In
TensorFlow With TensorRT User Guide.

Key Features And Enhancements
‣ Per channel and QDQ op support for Quantization API in TensorFlow 1.15
container

Known Issues
‣ We have seen a performance regression in SSD Mobilenet V1 in 19.12 with both
native TensorFlow and TF-TRT, mostly with batch size 8 but also 1 and 2, and
with all types of GPUs. This could be due to a change in the SSD graph. We are still investigating this issue.

- We have observed a regression in the performance of certain TF-TRT benchmarks in TensorFlow 1.15 including image classification models with precision INT8. We are still investigating this issue. Since 19.11 comes with a new version of TensorFlow (1.15), which includes a lot of changes in the TensorFlow backend, it’s very possible that the regression is caused by a change in the TensorFlow backend.

- We have observed a regression in the performance of certain TF-TRT benchmarks in TensorFlow 1.15 including image classification models with precision INT8. We are still investigating this. Since 19.11 comes with a new version of TensorFlow (1.15), which includes a lot of changes in the TensorFlow backend, it’s very possible that the regression is caused by a change in the TensorFlow backend.

- CUDA 10.2 and NCCL 2.5.x libraries require slightly more device memory than previous releases. As a result, some models that ran previously may exhaust device memory.

- The accuracy of Faster RCNN with the backbone ResNet-50 using TensorRT6.0 INT8 calibration is lower than expected. This will be fixed in future releases of TensorRT.

- The following sentence that appears in the log of TensorRT 6.0 can be safely ignored. This will be removed in the future releases of TensorRT.

    Calling isShapeTensor before the entire network is constructed may result in an inaccurate result.

- The following warning is issued when the method `build()` from the API is not called. This warning can be ignored.

    `OP_REQUIRES failed at trt_engine_resource_ops.cc:183 : Not found: Container TF-TRT does not exist. (Could not find resource: TF-TRT/TRTEngineOp...`  

- The following warning is issued because internally TensorFlow calls the TensorRT optimizer for certain objects unnecessarily. This warning can be ignored.

    `OP_REQUIRES failed at trt_engine_resource_ops.cc:183 : Not found: Container TF-TRT does not exist. (Could not find resource: TF-TRT/TRTEngineOp...`  

- We have seen failures when using INT8 calibration (post-training) within the same process that does FP32/FP16 conversion. We recommend to use separate processes for different precisions until this issue gets resolved.

- We have seen failures when calling the TensorRT optimizer on models that are already optimized by TensorRT. This issue will be fixed in a future release.

- In case you import nets from `models/slim`, you might see the following error:

    `AttributeError: module 'tensorflow_core.contrib' has no attribute 'tensorflow'`
Changing the order of imports can fix the issue. Therefore, import TensorRT before importing nets as follows:

```python
import tensorflow.contrib.tensorrt as trt
import nets.nets_factory
```

**Automatic Mixed Precision (AMP)**

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using an optimizer from `tf.train` or `tf.keras.optimizers` for both `compute_gradients()` and `apply_gradients()` operations (for example, by calling `optimizer.minimize()` or `model.fit()`), automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`.

For more information on this function, see the TensorFlow documentation [here](https://www.tensorflow.org/api_docs/python/tf/train/experimental/enable_mixed_precision_graph_rewrite).

For backward compatibility with previous container releases, AMP can also be enabled for `tf.train` optimizers by defining the following environment variable:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```


**Tensor Core Examples**

The [tensor core examples provided in GitHub](https://github.com/NVIDIA/TensorRT) focus on achieving the best performance and convergence by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper [U-Net: Convolutional Networks for Biomedical Image Segmentation](https://arxiv.org/abs/1505.04597), without any alteration. This model script is available on [GitHub](https://github.com) as well as [NVIDIA GPU Cloud (NGC)](https://ngc.nvidia.com).
- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the [SSD: Single Shot MultiBox Detector](https://arxiv.org/abs/1512.02325) paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on [GitHub](https://github.com/) as well as [NVIDIA GPU Cloud (NGC)](https://nvidia.co.uk/).

- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item. This model script is available on [GitHub](https://github.com/) as well as [NVIDIA GPU Cloud (NGC)](https://nvidia.co.uk/).

- **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](https://arxiv.org/abs/1810.04805) paper. NVIDIA's BERT is an optimized version of [Google's official implementation](https://github.com/google-research/bert), leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on [GitHub](https://github.com/) as well as [NVIDIA GPU Cloud (NGC)](https://nvidia.co.uk/).

- **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the [U-Net model](https://arxiv.org/abs/1505.04597) which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: [U-Net: Convolutional Networks for Biomedical Image Segmentation](https://arxiv.org/abs/1505.04597). This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset [DAGM2007](https://www.robots.ox.ac.uk/~guyw/dagm.html). This model script is available on [GitHub](https://github.com/) as well as [NVIDIA GPU Cloud (NGC)](https://nvidia.co.uk/).

- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the [Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](https://arxiv.org/abs/1409.0473) paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on [GitHub](https://github.com/) as well as [NVIDIA GPU Cloud (NGC)](https://nvidia.co.uk/).

- **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with
Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There are known issues since the 19.11 release for NCF inference with XLA and VGG16 training without XLA; these benchmarks have performance that is lower than expected.
- There are known issues regarding TF-TRT INT8 accuracy issues. See the Accelerating Inference In TensorFlow With TensorRT (TF-TRT) section above for more information.
- For BERT Large training with the 19.08 release on Tesla V100 boards with 16 GB memory, performance with batch size 3 per GPU is lower than expected; batch size 2 per GPU may be a better choice for this model on these GPUs with the 19.08 release. 32 GB GPUs are not affected.
- TensorBoard has a bug in its IPv6 support which can result in the following error: Tensorboard could not bind to unsupported address family ::. To workaround this error, pass the --host <IP> flag when starting TensorBoard.
- A known issue in TensorFlow results in the error Cannot take the length of Shape with unknown rank when training variable sized images with the Keras model.fit API. Details are provided here and a fix will be available in a future release.
- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.
- There is a known issue when your NVIDIA driver release is older than 418.xx since the 19.10 release, the Nsight Systems profiling tool (for example, the nsys) might cause CUDA runtime API error. A fix will be included in a future release.
Chapter 8.
TENSORFLOW RELEASE 19.11

The NVIDIA container image of TensorFlow, release 19.11, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 18.04
- NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89
- NVIDIA cuDNN 7.6.5
- NVIDIA NCCL 2.5.6 (optimized for NVLink™)
- Horovod 0.18.1
- OpenMPI 3.1.4
- TensorBoard
  - 19.11-tf1-py2 and 19.11-tf1-py3 include version 1.15.0+nv
  - 19.11-tf2-py3 includes version 2.0.1
- MLNX_OFED
- OpenSeq2Seq at commit a81babd
  - Included only in 19.11-tf1-py2 and 19.11-tf1-py3
- TensorFlow Release 19.11
- www.nvidia.com
- TensorFlow
- RN-08467-001_v20.03 | 34

‣ TensorRT 6.0.1
‣ DALI 0.15.0 Beta
‣ DLProf 19.11
  ‣ Included only in 19.11-tf1-py2 and 19.11-tf1-py3
‣ Nsight Compute 2019.5.0
‣ Nsight Systems 2019.5.2
‣ Tensor Core optimized examples: (Included only in 19.11-tf1-py2 and 19.11-tf1-py3)
  ‣ U-Net Medical
  ‣ SSD320 v1.2
  ‣ Neural Collaborative Filtering (NCF)
  ‣ BERT
  ‣ U-Net Industrial Defect Segmentation
  ‣ GNMT v2
  ‣ ResNet-50 v1.5
‣ Jupyter and JupyterLab:
  ‣ Jupyter Client 5.3.4
  ‣ Jupyter Core 4.6.1
  ‣ Jupyter Notebook 6.0.1
  ‣ JupyterLab 1.0.2
  ‣ JupyterLab Server 1.0.0
  ‣ Jupyter-TensorBoard

**Driver Requirements**

Release 19.11 is based on NVIDIA CUDA 10.2.89, which requires NVIDIA Driver release 440.30. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+, 410 or 418.xx. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

**GPU Requirements**

Release 19.11 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

**Key Features and Enhancements**

This TensorFlow release includes the following key features and enhancements.
TensorFlow container image version 19.11 is based on TensorFlow 1.15.0 and TensorFlow 2.0.0.

Added a TensorFlow 2.x container.

Latest version of NVIDIA CUDA 10.2.89 including cuBLAS 10.2.2.89

Latest version of NVIDIA cuDNN 7.6.5

Latest version of NVIDIA NCCL 2.5.6

Latest version of TensorFlow. We now provide:

- 19.11-tf1-py2 and 19.11-tf1-py3 include version 1.15.0+nv
- 19.11-tf2-py3 includes version 2.0.1

Latest version of OpenSeq2Seq at commit a81babd

- Included only in 19.11-tf1-py2 and 19.11-tf1-py3

Latest version of Nsight Compute 2019.5.0

Latest version of Nsight Systems 2019.5.2

Latest version of DLProf 19.11

- Included only in 19.11-tf1-py2 and 19.11-tf1-py3

Latest versions of Jupyter Client 5.3.4 and Jupyter Core 4.6.1

Support added for the fast cuDNN CTC loss function via nn.ctc_loss (for 19.11-tf2-py3) or nn.ctc_loss_v2 (for 19.11-tf1-py3). To enable it, define the export TF_CUDNN_CTC_LOSS=1 environment variable.

Ubuntu 18.04 with October 2019 updates

Announcements

- We will stop support for Python 2.7 in a future TensorFlow container release.
- Deep learning framework containers 19.11 and later include experimental support for Singularity v3.0.

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

- New converters were added. Refer to the Supported Operators section in the Accelerating Inference In TensorFlow With TensorRT User Guide for the list of new converters.

- TensorFlow 2.0:
  - A new API is introduced for TF-TRT in TensorFlow 2.0. This new API can only be used in TensorFlow 2.0. Refer to the User Guide for more information about the new API.
Introduced a new API method (`converter.build()` for optimizing TensorFlow engines during graph optimization. Previously, the optimization during preprocessing (before deployment) was possible by using `is_dynamic_op=False`.

- `converter.convert()` no longer returns a `tf.function`. Now, the function must be accessed from the saved model. This encapsulates the function in the converter for better safety.

- The `converter.calibrate()` method has been removed. To trigger calibration, a `calibration_input_fn` should be provided to `converter.convert()`.

**Deprecated Features**

- The old API of TF-TRT is deprecated. It still works in TensorFlow 1.14 and 1.15, however, it is removed in TensorFlow 2.0. The old API is a Python function named `create_inference_graph` which is now replaced by the Python class `TrtGraphConverter` in TensorFlow 1.x and `TrtGraphConverterV2` in TensorFlow 2.0 with a number of methods. Refer to [TF-TRT User Guide](#) for more information about the API and how to use it.

**Known Issues**

- We have observed a regression in the performance of certain TF-TRT benchmarks in TensorFlow 1.15 including image classification models with precision INT8. We are still investigating this. Since 19.11 comes with a new version of TensorFlow (1.15), which includes a lot of changes in the TensorFlow backend, it’s very possible that the regression is caused by a change in the TensorFlow backend.

- CUDA 10.2 and NCCL 2.5.x libraries require slightly more device memory than previous releases. As a result, some models that ran previously may exhaust device memory.

- TensorRT INT8 calibration algorithm (see the [TF-TRT User Guide](#) for more information about how to use INT8) is very slow for certain models such as NASNet and Inception. We are working on optimizing the calibration algorithm in TensorRT.

- The accuracy of Faster RCNN with the backbone ResNet-50 using TensorRT6.0 INT8 calibration is lower than expected. We are investigating the issue.

- The following sentence that appears in the log of TensorRT 6.0 can be safely ignored. This will be removed in the future releases of TensorRT.

> Calling `isShapeTensor` before the entire network is constructed may result in an inaccurate result.

- The following warning is issued when the method `build()` from the API is not called. This warning can be ignored.
The following warning is issued because internally TensorFlow calls the TensorRT optimizer for certain objects unnecessarily. This warning can be ignored.

We have seen failures when using INT8 calibration (post-training) within the same process that does FP32/FP16 conversion. We recommend to use separate processes for different precisions until this issue gets resolved.

We have seen failures when calling the TensorRT optimizer on models that are already optimized by TensorRT. This issue will be fixed in a future release.

In case you import nets from `models/slim`, you might see the following error:

```
AttributeError: module 'tensorflow_core.contrib' has no attribute 'tensorrt'
```

Changing the order of imports can fix the issue. Therefore, import TensorRT before importing nets as follows:

```
import tensorflow.contrib.tensorrt as trt
import nets.nets_factory
```

### Automatic Mixed Precision (AMP)

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using an optimizer from `tf.train` or `tf.keras.optimizers` for both `compute_gradients()` and `apply_gradients()` operations (for example, by calling `optimizer.minimize()` or `model.fit()`), automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`.

For more information on this function, see the TensorFlow documentation [here](https://www.tensorflow.org/api_docs/python/tf/train/enable_mixed_precision_graph_rewrite).

For backward compatibility with previous container releases, AMP can also be enabled for `tf.train` optimizers by defining the following environment variable:

```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

Tensor Core Examples

The tensor core examples provided in GitHub focus on achieving the best performance and convergence by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

‣ **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation, without any alteration. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

‣ **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

‣ **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

‣ **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

‣ **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the
industrial anomaly dataset DAGM2007. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

**Known Issues**

- There are known issues in the 19.11 release for NCF inference with XLA and VGG16 training without XLA; these benchmarks have performance that is lower than expected.

- There are known issues regarding TF-TRT INT8 accuracy issues. See the Accelerating Inference In TensorFlow With TensorRT (TF-TRT) section above for more information.

- For BERT Large training with the 19.08 release on Tesla V100 boards with 16 GB memory, performance with batch size 3 per GPU is lower than expected; batch size 2 per GPU may be a better choice for this model on these GPUs with the 19.08 release. 32 GB GPUs are not affected.

- TensorBoard has a bug in its IPv6 support which can result in the following error: Tensorboard could not bind to unsupported address family ::. To workaround this error, pass the --host <IP> flag when starting TensorBoard.

- Automatic Mixed Precision (AMP) does not support the Keras LearningRateScheduler in the 19.08 release. A fix will be included in a future release.

- A known issue in TensorFlow results in the error Cannot take the length of Shape with unknown rank when training variable sized images with the Keras model.fit API. Details are provided here and a fix will be available in a future release.
Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.

There is a known issue when your NVIDIA driver release is older than 418.xx in the 19.10 release, the Nsight Systems profiling tool (for example, the `nsys`) might cause CUDA runtime API error. A fix will be included in a future release.
Chapter 9.
TENSORFLOW RELEASE 19.10

The NVIDIA container image of TensorFlow, release 19.10, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 18.04


- NVIDIA CUDA 10.1.243 including cuBLAS 10.2.1.243
- NVIDIA cuDNN 7.6.4
- NVIDIA NCCL 2.4.8 (optimized for NVLink™)
- Horovod 0.18.1
- OpenMPI 3.1.4
- TensorBoard 1.14.0+nv
- MLNX_OFED
- OpenSeq2Seq at commit 2e0b1d8
- TensorRT 6.0.1
- DALI 0.14.0 Beta
- DLProf 19.10
- Nsight Compute 2019.4.0
- Nsight Systems 2019.5.1
Tensor Core optimized example:

- **U-Net Medical**
- **SSD320 v1.2**
- **Neural Collaborative Filtering (NCF)**
- **BERT**
- **U-Net Industrial Defect Segmentation**
- **GNMT v2**
- **ResNet-50 v1.5**

Jupyter and JupyterLab:

- **Jupyter Client 5.3.3**
- **Jupyter Core 4.5.0**
- **Jupyter Notebook 6.0.1**
- **JupyterLab 1.0.2**
- **JupyterLab Server 1.0.0**
- **Jupyter-TensorBoard**

**Driver Requirements**

Release 19.10 is based on [NVIDIA CUDA 10.1.243](https://docs.nvidia.com/cuda/index.html), which requires [NVIDIA Driver](https://developer.nvidia.com/cuda-toolkit) release 418.xx. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+ or 410. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the [CUDA Application Compatibility](https://docs.nvidia.com/cuda/index.html) topic. For more information, see [CUDA Compatibility and Upgrades](https://docs.nvidia.com/cuda/index.html).

**GPU Requirements**

Release 19.10 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see [CUDA GPUs](https://docs.nvidia.com/cuda/index.html). For additional support details, see [Deep Learning Frameworks Support Matrix](https://docs.nvidia.com/cuda/index.html).

**Key Features and Enhancements**

This TensorFlow release includes the following key features and enhancements.

- **TensorFlow** container image version 19.10 is based on [TensorFlow 1.14.0](https://www.tensorflow.org/)
- Latest version of [NVIDIA cuDNN 7.6.4](https://docs.nvidia.com/deeplearning/cudnn/index.html)
- Latest version of [Horovod 0.18.1](https://github.com/uber/horovod)
- Latest version of [DALI 0.14.0 Beta](https://docs.nvidia.com/deeplearning/dali/index.html)
- Latest version of [DLProf 19.10](https://software.nvidia.com/dlprof)
- Latest versions of [Nsight Systems 2019.5.1](https://docs.nvidia.com/nsight/index.html)
- Latest versions of Jupyter Client 5.3.3
- Dilated convolutions will now be evaluated using cuDNN by default.
- Automatic Mixed Precision will correctly handle TensorList ops.
- Automatic Mixed Precision can now evaluate softmax and activation functions in FP16.
- Ubuntu 18.04 with September 2019 updates

**Announcements**

We will stop support for Python 2.7 in a future TensorFlow container release.

**Accelerating Inference In TensorFlow With TensorRT (TF-TRT)**

For step-by-step instructions on how to use TF-TRT, see [Accelerating Inference In TensorFlow With TensorRT User Guide](#).

**Deprecated Features**

- The old API of TF-TRT is deprecated. It still works in TensorFlow 1.14 and 1.15, however, it is removed in TensorFlow 2.0. The old API is a Python function named `create_inference_graph` which is now replaced by the Python class `TrtGraphConverter` with a number of methods. Refer to [TF-TRT User Guide](#) for more information about the API and how to use it.

**Known Issues**

- TensorRT INT8 calibration algorithm (see the [TF-TRT User Guide](#) for more information about how to use INT8) is very slow for certain models such as NASNet and Inception. We are working on optimizing the calibration algorithm in TensorRT.
- The pip package of TensorFlow 1.14 released by Google is missing TensorRT. This will be fixed in the next release of TensorFlow by Google. In the meantime, you can use the more recent versions of TensorFlow pip packages released by Google (1.15 and 2.0) or the [NVIDIA container](#) for TensorFlow.
- The accuracy of Faster RCNN with the backbone ResNet-50 using TensorRT6.0 INT8 calibration is lower than expected. We are investigating the issue.
- The following sentence that appears in the log of TensorRT 6.0 can be safely ignored. This will be removed in the future releases of TensorRT.

```
Calling isShapeTensor before the entire network is constructed may result in an inaccurate result.
```

**Automatic Mixed Precision (AMP)**

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:
a loss scaling optimizer  
graph rewriter

For models already using an optimizer from `tf.train` or `tf.keras.optimizers` for both `compute_gradients()` and `apply_gradients()` operations (for example, by calling `optimizer.minimize()` or `model.fit()`), automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`.

For more information on this function, see the TensorFlow documentation here.

For backward compatibility with previous container releases, AMP can also be enabled for `tf.train` optimizers by defining the following environment variable:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the TensorFlow User Guide, along with Training With Mixed Precision.

Tensor Core Examples

The tensor core examples provided in GitHub focus on achieving the best performance and convergence by using the latest deep learning example networks and model scripts for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation, without any alteration. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a
rating to or clicked on, the specified item. This model script is available on GitHub as well as **NVIDIA GPU Cloud (NGC)**.

- **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as **NVIDIA GPU Cloud (NGC)**.

- **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007. This model script is available on GitHub as well as **NVIDIA GPU Cloud (NGC)**.

- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on GitHub as well as **NVIDIA GPU Cloud (NGC)**.

- **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as **NVIDIA GPU Cloud (NGC)**.

**Known Issues**

- There are known issues regarding TF-TRT INT8 accuracy issues. See the *Accelerating Inference In TensorFlow With TensorRT (TF-TRT)* section above for more information.

- There is a known performance regression in TensorFlow 1.14.0 affecting a variety of models. Affected models include GNMT, SSD, and NCF. Performance regressions can be as high as 20% compared to TensorFlow 1.13.1 in the 19.06 release.
- For BERT Large training with the 19.08 release on Tesla V100 boards with 16 GB memory, performance with batch size 3 per GPU is lower than expected; batch size 2 per GPU may be a better choice for this model on these GPUs with the 19.08 release. 32 GB GPUs are not affected.
- TensorBoard has a bug in its IPv6 support which can result in the following error: `Tensorboard could not bind to unsupported address family ::`. To workaround this error, pass the `--host <IP>` flag when starting TensorBoard.
- Automatic Mixed Precision (AMP) does not support the Keras `LearningRateScheduler` in the 19.08 release. A fix will be included in a future release.
- A known issue in TensorFlow results in the error `Cannot take the length of Shape with unknown rank` when training variable sized images with the Keras `model.fit` API. Details are provided [here](#) and a fix will be available in a future release.
- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.
- There is a known issue when your NVIDIA driver release is older than 418.xx in the 19.10 release, the Nsight Systems profiling tool (for example, the `nsys`) might cause CUDA runtime API error. A fix will be included in a future release.
Chapter 10.
TENSORFLOW RELEASE 19.09

The NVIDIA container image of TensorFlow, release 19.09, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 18.04
- NVIDIA CUDA 10.1.243 including cuBLAS 10.2.1.243
- NVIDIA cuDNN 7.6.3
- NVIDIA NCCL 2.4.8 (optimized for NVLink™)
- Horovod 0.18.0
- OpenMPI 3.1.4
- TensorBoard 1.14.0+nv
- MLNX_OFED
- OpenSeq2Seq at commit 2e0b1d8
- TensorRT 6.0.1
- DALI 0.13.0 Beta
- DLProf 19.09
- Nsight Compute 2019.4.0
- Nsight Systems 2019.4.2
Tensor Core optimized example:

- U-Net Medical
- SSD320 v1.2
- Neural Collaborative Filtering (NCF)
- BERT
- U-Net Industrial Defect Segmentation
- GNMT v2
- ResNet-50 v1.5

Jupyter and JupyterLab:

- Jupyter Client 5.3.1
- Jupyter Core 4.5.0
- Jupyter Notebook 6.0.1
- JupyterLab 1.0.2
- JupyterLab Server 1.0.0
- Jupyter-TensorBoard

Driver Requirements

Release 19.09 is based on NVIDIA CUDA 10.1.243, which requires NVIDIA Driver release 418.xx. However, if you are running on Tesla (for example, T4 or any other Tesla board), you may use NVIDIA driver release 396, 384.111+ or 410. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.09 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 19.09 is based on TensorFlow 1.14.0.
- Latest version of NVIDIA cuDNN 7.6.3
- Latest version of Horovod 0.18.0
- Latest version of TensorRT 6.0.1
- Latest version of DALI 0.13.0 Beta
- Latest version of DLPprof 19.09
- Latest versions of **Nsight Compute 2019.4.0** and **Nsight Systems 2019.4.2**
- Latest version of **Jupyter Notebook 6.0.1**
- Ubuntu 18.04 with August 2019 updates

**Announcements**

We will stop support for Python 2.7 in a future TensorFlow container release.

**Accelerating Inference In TensorFlow With TensorRT (TF-TRT)**

For step-by-step instructions on how to use TF-TRT, see **Accelerating Inference In TensorFlow With TensorRT User Guide**.

**Deprecated Features**

- The old API of TF-TRT is deprecated. It still works in TensorFlow 1.14, however, it may be removed in TensorFlow 2.0. The old API is a Python function named `create_inference_graph` which is not replaced by the Python class `TrtGraphConverter` with a number of methods. Refer to **TF-TRT User Guide** for more information about the API and how to use it.

**Known Issues**

- Precision mode in the TF-TRT API is a string with one of the following values: **FP32**, **FP16** or **INT8**. In TensorFlow 1.13, these strings were supported in lowercase, however, in TensorFlow 1.14 only uppercase is supported.
- INT8 calibration (see the **TF-TRT User Guide** for more information about how to use INT8) is a very slow process that can take 1 hour depending on the model. We are working on optimizing this algorithm in TensorRT.
- The pip package of TensorFlow 1.14 released by Google is missing TensorRT. This will be fixed in the next release of TensorFlow by Google. In the meantime, you can use the **NVIDIA container** for TensorFlow.

**Automatic Mixed Precision (AMP)**

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using a **tf.train.Optimizer** or **tf.keras.optimizers.Optimizer** for both **compute_gradients()** and **apply_gradients()** operations, automatic mixed precision can be enabled by wrapping the optimizer with **tf.train.experimental.enable_mixed_precision_graph_rewrite()**. For backward compatibility with AMP in previous containers, AMP can also be enabled by
defining the following environment variable before calling the usual float32 training script:

```bash
export TF_ENABLEAUTO_MIXED_PRECISION=1
```

Models implementing their own optimizers can use the graph rewriter on its own (while implementing loss scaling manually) by setting the following flag in the `tf.session` config:

```python
config.graph_options.rewrite_options.auto_mixed_precision=1
```

Or equivalently for backward compatibility with AMP in previous NGC containers, by setting the following environment variable:

```bash
export TF_ENABLEAUTO_MIXED_PRECISION_GRAPH_REWRITE=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the TensorFlow User Guide, along with Training With Mixed Precision.

**Tensor Core Examples**

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper [U-Net: Convolutional Networks for Biomedical Image Segmentation](https://arxiv.org/abs/1505.04597), without any alteration. This model script is available on [GitHub](https://github.com) as well as NVIDIA GPU Cloud (NGC).

- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on [GitHub](https://github.com) as well as NVIDIA GPU Cloud (NGC).

- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item. This model script is available on [GitHub](https://github.com) as well as NVIDIA GPU Cloud (NGC).
• **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

• **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

• **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

• **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

**Known Issues**

• There is a known performance regression in TensorFlow 1.14.0 affecting a variety of models. Affected models include GNMT, SSD, and UNet. Performance regressions can be as high as 20% compared to TensorFlow 1.13.1 in the 19.06 release.

• For BERT Large training with the 19.08 release on Tesla V100 boards with 16 GB memory, performance with batch size 3 per GPU is lower than expected; batch size 2 per GPU may be a better choice for this model on these GPUs with the 19.08 release. 32 GB GPUs are not affected.
TensorBoard has a bug in its IPv6 support which can result in the following error: `Tensorboard could not bind to unsupported address family ::`. To workaround this error, pass the `--host <IP>` flag when starting TensorBoard.

In previous containers, `libtensorflow_framework.so` was available in the `/usr/local/lib/tensorflow` directory. This was redundant with the libs installed with the TensorFlow pip package. To find the TensorFlow lib directory, use `tf.sysconfig.get_lib()`.

Automatic Mixed Precision (AMP) does not support the Keras `LearningRateScheduler` in the 19.08 release. A fix will be included in a future release.

A known issue in TensorFlow results in the error *Cannot take the length of Shape with unknown rank* when training variable sized images with the Keras `model.fit` API. Details are provided here and a fix will be available in a future release.

Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.

There is a known issue when your NVIDIA driver release is older than 418.xx in the 19.09 release, the Nsight Systems profiling tool (for example, the `nsys`) might cause CUDA runtime API error. A fix will be included in a future release.
The NVIDIA container image of TensorFlow, release 19.08, is available on NGC.

**Contents of the TensorFlow container**

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- **Ubuntu 18.04**
- **NVIDIA CUDA 10.1.243** including cuBLAS 10.2.1.243
- **NVIDIA cuDNN 7.6.2**
- **NVIDIA NCCL 2.4.8** (optimized for NVLink™)
- **Horovod 0.16.2**
- **OpenMPI 3.1.4**
- **TensorBoard 1.14.0+nv**
- **MLNX_OFED +4.0**
- **OpenSeq2Seq at commit 2e0b1d8**
- **TensorRT 5.1.5**
- **DALI 0.12.0 Beta**
- **DLProf 19.08**
- **Nsight Compute 10.1.168**
- **Nsight Systems 2019.3.7.9**
Tensor Core optimized example:

- U-Net Medical
- SSD320 v1.2
- Neural Collaborative Filtering (NCF)
- BERT
- U-Net Industrial Defect Segmentation
- GNMT v2
- ResNet-50 v1.5

Jupyter and JupyterLab:

- Jupyter Client 5.3.1
- Jupyter Core 4.5.0
- Jupyter Notebook 6.0.0
- JupyterLab 1.0.2
- JupyterLab Server 1.0.0
- Jupyter-TensorBoard

Driver Requirements

Release 19.08 is based on NVIDIA CUDA 10.1.243, which requires NVIDIA Driver release 418.87. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.08 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 19.08 is based on TensorFlow 1.14.0.
- Latest version of NVIDIA CUDA 10.1.243 including cuBLAS 10.2.1.243
- Latest version of NVIDIA cuDNN 7.6.2
- Latest version of NVIDIA NCCL 2.4.8
- Latest version of OpenMPI 3.1.4
- Latest commit of OpenSeq2Seq
- Latest version of Nsight Systems 2019.3.7.9
Latest version of DALI 0.12.0 Beta
Latest version of MLNX_OFED +4.0
Latest version of DLPprof 19.08
Latest version of Jupyter Notebook 6.0.0
Ubuntu 18.04 with July 2019 updates

Announcements

We will stop support for Python 2.7 in a future TensorFlow container release.

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

- Migrated TensorRT conversion sources from the contrib directory to the compiler directory in preparation for TensorFlow 2.0. The Python code can be found at //tensorflow/python/compiler/tensorrt.
- Added a user friendly TrtGraphConverter API for TensorRT conversion.
- Expanded support for TensorFlow operators in TensorRT conversion (for example, Gather, Slice, Pack, Unpack, ArgMin, ArgMax, DepthSpaceShuffle). Refer to the TF-TRT User Guide for a complete list of supported operators.
- Support added for TensorFlow operator CombinedNonMaxSuppression in TensorRT conversion which significantly accelerates SSD object detection models.
- Integrated TensorRT 5.1.5 into TensorFlow. See the TensorRT 5.1.5 Release Notes for a full list of new features.

Deprecated Features

- The old API of TF-TRT is deprecated. It still works in TensorFlow 1.14, however, it may be removed in TensorFlow 2.0. The old API is a Python function named create_inference_graph which is not replaced by the Python class TrtGraphConverter with a number of methods. Refer to TF-TRT User Guide for more information about the API and how to use it.

Known Issues

- Precision mode in the TF-TRT API is a string with one of the following values: FP32, FP16 or INT8. In TensorFlow 1.13, these strings were supported in lowercase, however, in TensorFlow 1.14 only uppercase is supported.
- INT8 calibration (see the TF-TRT User Guide for more information about how to use INT8) is a very slow process that can take 1 hour depending on the model. We are working on optimizing this algorithm in TensorRT.
The pip package of TensorFlow 1.14 released by Google is missing TensorRT. This will be fixed in the next release of TensorFlow by Google. In the meantime, you can use the NVIDIA container for TensorFlow.

**Automatic Mixed Precision (AMP)**

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using a `tf.train.Optimizer` or `tf.keras.optimizers.Optimizer` for both `compute_gradients()` and `apply_gradients()` operations, automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`. For backward compatibility with AMP in previous containers, AMP can also be enabled by defining the following environment variable before calling the usual float32 training script:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

Models implementing their own optimizers can use the graph rewriter on its own (while implementing loss scaling manually) by setting the following flag in the `tf.session` config:

```python
config.graph_options.rewrite_options.auto_mixed_precision=1
```

Or equivalently for backward compatibility with AMP in previous NGC containers, by setting the following environment variable:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see [Automatic Mixed Precision Training In TensorFlow](#) from the TensorFlow User Guide, along with [Training With Mixed Precision](#).

**Tensor Core Examples**

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.
- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper *U-Net: Convolutional Networks for Biomedical Image Segmentation*, without any alteration. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the *SSD: Single Shot MultiBox Detector* paper, which describes an SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUS for faster training times while maintaining target accuracy. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: *U-Net: Convolutional Networks for Biomedical Image Segmentation*. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the *Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).
ResNet-50 v1.5 model. The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training. This model script is available on GitHub as well as NVIDIA GPU Cloud (NGC).

Known Issues

- There is a known performance regression in TensorFlow 1.14.0 affecting a variety of models. Affected models include GNMT, SSD, and UNet. Performance regressions can be as high as 20% compared to TensorFlow 1.13.1 in the 19.06 release.
- For BERT Large training with the 19.08 release on Tesla V100 boards with 16 GB memory, performance with batch size 3 per GPU is lower than expected; batch size 2 per GPU may be a better choice for this model on these GPUs with the 19.08 release. 32 GB GPUs are not affected.
- TensorBoard has a bug in its IPv6 support which can result in the following error: `Tensorboard could not bind to unsupported address family :::`. To workaround this error, pass the `--host <IP>` flag when starting TensorBoard.
- In previous containers, `libtensorflow_framework.so` was available in the `/usr/local/lib/tensorflow` directory. This was redundant with the libs installed with the TensorFlow pip package. To find the TensorFlow `lib` directory, use `tf.sysconfig.get_lib()`.
- Automatic Mixed Precision (AMP) does not support the Keras `LearningRateScheduler` in the 19.08 release. A fix will be included in a future release.
- A known issue in TensorFlow results in the error `Cannot take the length of Shape with unknown rank` when training variable sized images with the Keras `model.fit` API. Details are provided here and a fix will be available in a future release.
- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.
Chapter 12.
TENSORFLOW RELEASE 19.07

The NVIDIA container image of TensorFlow, release 19.07, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 18.04
- NVIDIA CUDA 10.1.168 including cuBLAS 10.2.0.168
- NVIDIA cuDNN 7.6.1
- NVIDIA NCCL 2.4.7 (optimized for NVLink™)
- Horovod 0.16.2
- OpenMPI 3.1.3
- TensorBoard 1.14.0+nv
- MLNX_OFED +3.4
- OpenSeq2Seq at commit 27346d1
- TensorRT 5.1.5
- DALI 0.11.0 Beta
- DLProf 19.07
- Nsight Compute 10.1.168
- Nsight Systems 2019.3.6.30

Tensor Core optimized example:

- **U-Net Medical**
- **SSD320 v1.2**
- **Neural Collaborative Filtering (NCF)**
- **BERT**
- **U-Net Industrial Defect Segmentation**
- **GNMT v2**
- **ResNet-50 v1.5**

Jupyter and JupyterLab:

- **Jupyter Client 5.3.1**
- **Jupyter Core 4.5.0**
- **Jupyter Notebook 5.7.8**
- **JupyterLab 1.0.1**
- **JupyterLab Server 1.0.0**
- **Jupyter-TensorBoard**

Driver Requirements

Release 19.07 is based on NVIDIA CUDA 10.1.168, which requires NVIDIA Driver release 418.67. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.07 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- **TensorFlow** container image version 19.07 is based on TensorFlow 1.14.0.
- Automatic Mixed Precision updated with latest upstream changes (see below)
- Latest version of Nsight Systems 2019.3.6.30
- Latest version of Python 3.6
- Latest version of TensorBoard 1.14.0 with additional updates from NVIDIA
- Latest version of NVIDIA cuDNN 7.6.1
Latest versions of Jupyter Client 5.3.1, Jupyter Core 4.5.0, JupyterLab 1.0.1 and JupyterLab Server 1.0.0, including Jupyter-TensorBoard integration.

Latest version of DLProf 19.07

Latest version of DALI 0.11.0 Beta

Latest version of Ubuntu 18.04

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

- Migrated TensorRT conversion sources from the contrib directory to the compiler directory in preparation for TensorFlow 2.0. The Python code can be found at //tensorflow/python/compiler/tensorrt.
- Added a user friendly TrtGraphConverter API for TensorRT conversion.
- Expanded support for TensorFlow operators in TensorRT conversion (for example, Gather, Slice, Pack, Unpack, ArgMin, ArgMax, DepthSpaceShuffle). Refer to the TF-TRT User Guide for a complete list of supported operators.
- Support added for TensorFlow operator CombinedNonMaxSuppression in TensorRT conversion which significantly accelerates SSD object detection models.
- Integrated TensorRT 5.1.5 into TensorFlow. See the TensorRT 5.1.5 Release Notes for a full list of new features.

Deprecated Features

- The old API of TF-TRT is deprecated. It still works in TensorFlow 1.14, however, it may be removed in TensorFlow 2.0. The old API is a Python function named create_inference_graph which is not replaced by the Python class TrtGraphConverter with a number of methods. Refer to TF-TRT User Guide for more information about the API and how to use it

Known Issues

- Precision mode in the TF-TRT API is a string with one of the following values: FP32, FP16 or INT8. In TensorFlow 1.13, these strings were supported in lowercase, however, in TensorFlow 1.14 only uppercase is supported.
- INT8 calibration (see the TF-TRT User Guide for more information about how to use INT8) is a very slow process that can take 1 hour depending on the model. We are working on optimizing this algorithm in TensorRT.
- The pip package of TensorFlow 1.14 released by Google is missing TensorRT. This will be fixed in the next release of TensorFlow by Google. In the meantime, you can use the NVIDIA container for TensorFlow.
Announcements

We will stop support for Python 2.7 in a future TensorFlow container release. Once support has ended, the TensorFlow container will contain only one version of Python.

Automatic Mixed Precision (AMP)

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using a `tf.train.Optimizer` or `tf.keras.optimizers.Optimizer` for both `compute_gradients()` and `apply_gradients()` operations, automatic mixed precision can be enabled by wrapping the optimizer with `tf.train.experimental.enable_mixed_precision_graph_rewrite()`. For backward compatibility with AMP in previous containers, AMP can also be enabled by defining the following environment variable before calling the usual float32 training script:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

Models implementing their own optimizers can use the graph rewriter on its own (while implementing loss scaling manually) with the following environment variable:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the TensorFlow User Guide, along with Training With Mixed Precision.

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation, without any alteration.
SSD320 v1.2 model. The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository.

Neural Collaborative Filtering (NCF) model. The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item.

BERT model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

U-Net Industrial Defect Segmentation model. This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007.

GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep.

ResNet-50 v1.5 model. The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, tensor cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training.
Known Issues

- There is a known performance regression in TensorFlow 1.14.0 affecting a variety of models. Affected models include GNMT, SSD, and UNet. Performance regressions can be as high as 20% compared to TensorFlow 1.13.1 in the 19.06 release.

- There is an issue in TensorFlow 1.14 that increases the GPU memory footprint of certain models such as BERT. As a result, training may need to be performed with a reduced batch size.

- TensorBoard has a bug in its IPv6 support which can result in the following error: `Tensorboard could not bind to unsupported address family :::`. To workaround this error, pass the `--host <IP>` flag when starting TensorBoard.

- In previous containers, `libtensorflow_framework.so` was available in the `/usr/local/lib/tensorflow` directory. This was redundant with the libs installed with the TensorFlow pip package. To find the TensorFlow lib directory, use `tf.sysconfig.get_lib()`.

- Automatic Mixed Precision (AMP) does not support the Keras `LearningRateScheduler` in the 19.07 release. A fix will be included in the 19.08 release.

- A known issue in TensorFlow results in the error `Cannot take the length of Shape with unknown rank` when training variable sized images with the Keras `model.fit` API. Details are provided [here](#) and a fix will be available in a future release.

- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.

- Using `TF_ENABLE_NHWC=1` might cause memory leak (OOM) if `FusedBatchNormV3` is explicitly used. By default, `tf.nn.fused_batch_norm()` uses `FusedBatchNorm` and `FusedBatchNormV2`. The `FusedBatchNormV3` is set to be available after November 11th, 2019. A fix will be included in the 19.08 release.
Chapter 13.
TENSORFLOW RELEASE 19.06

The NVIDIA container image of TensorFlow, release 19.06, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 10.1.168 including cuBLAS 10.2.0.168
- NVIDIA cuDNN 7.6.0
- NVIDIA NCCL 2.4.7 (optimized for NVLink™)
- Horovod 0.16.2
- OpenMPI 3.1.3
- TensorBoard 1.13.1+nv
- MLNX_OFED 3.4
- OpenSeq2Seq at commit 27346d1
- TensorRT 5.1.5
- DALI 0.10.0 Beta
- DLProf 19.06
- Nsight Compute 10.1.168
- Nsight Systems 2019.3.1.94
Tensor Core optimized example:

- U-Net Medical
- SSD320 v1.2
- Neural Collaborative Filtering (NCF)
- BERT
- U-Net Industrial Defect Segmentation
- GNMT v2
- ResNet-50 v1.5

Jupyter and JupyterLab:

- Jupyter Client 5.2.4
- Jupyter Core 4.4.0
- Jupyter Notebook 5.7.8
- JupyterLab 0.35.6
- JupyterLab Server 0.2.0

Driver Requirements

Release 19.06 is based on NVIDIA CUDA 10.1.168, which requires NVIDIA Driver release 418.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.06 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 19.06 is based on TensorFlow 1.13.1.
- Latest version of NVIDIA CUDA 10.1.168 including cuBLAS 10.2.0.168
- Latest version of NVIDIA NCCL 2.4.7
- Latest version of DALI 0.10.0 Beta
- Latest version of JupyterLab 0.35.6
- Latest version of Horovod 0.16.2
- Latest version of Nsight Compute 10.1.168
- Latest OpenSeq2Seq at commit 27346d1
Added DLProf 19.06 software. Deep Learning Profiler (DLProf) is a tool for profiling deep learning models to help data scientists understand and improve performance of their models visually via TensorBoard or by analyzing text reports.

- Determinism - Setting the environment variable `TF_CUDNN_DETERMINISM=1` forces the selection of deterministic cuDNN convolution and max-pooling algorithms. When this is enabled, the algorithm selection procedure itself is also deterministic.

  Alternatively, setting `TF_DETERMINISTIC_OPS=1` has the same effect and additionally makes any bias addition that is based on `tf.nn.bias_add()` (for example, in Keras layers) operate deterministically on GPU. If you set `TF_DETERMINISTIC_OPS=1` then there is no need to also set `TF_CUDNN_DETERMINISM=1`.

  Selecting these deterministic options may reduce performance.

- Ubuntu 16.04 with May 2019 updates (see Announcements)

**Accelerating Inference In TensorFlow With TensorRT (TF-TRT)**


**Key Features And Enhancements**

- Integrated TensorRT 5.1.5 into TensorFlow. See the [TensorRT 5.1.5 Release Notes](https://www.nvidia.com) for a full list of new features.
- Improved examples at [GitHub: TF-TRT](https://www.nvidia.com), including README files, build scripts, benchmark mode, ResNet models from TensorFlow official model zoo, etc...

**Automatic Mixed Precision (AMP)**

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

- a loss scaling optimizer
- graph rewriter

For models already using a `tf.Optimizer()` for both `compute_gradients()` and `apply_gradients()` operations, automatic mixed precision can be enabled by defining the following environment variable before calling the usual float32 training script:

```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

Models implementing their own optimizers can use the graph rewriter on its own (while implementing loss scaling manually) with the following environment variable:

```
export TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE=1
```

**Tensor Core Examples**

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- **U-Net Medical model.** The U-Net model is a convolutional neural network for 2D image segmentation. This repository contains a U-Net implementation as described in the paper [U-Net: Convolutional Networks for Biomedical Image Segmentation](https://arxiv.org/abs/1505.04597), without any alteration.

- **SSD320 v1.2 model.** The SSD320 v1.2 model is based on the [SSD: Single Shot MultiBox Detector](https://arxiv.org/abs/1512.02325) paper, which describes SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing [model from the TensorFlow models repository](https://github.com/tensorflow/models).

- **Neural Collaborative Filtering (NCF) model.** The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item.

- **BERT model.** BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](https://arxiv.org/abs/1810.04805) paper. NVIDIA's BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUs for faster training times while maintaining target accuracy.

- **U-Net Industrial Defect Segmentation model.** This U-Net model is adapted from the original version of the [U-Net model](https://arxiv.org/abs/1505.04597) which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: [U-Net: Convolutional Networks for Biomedical Image Segmentation](https://arxiv.org/abs/1505.04597). This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset [DAGM2007](https://www.tensorflow.org/performance/mixed_precision#training_with_mixed_precision).
- **GNMT v2 model.** The GNMT v2 model is similar to the one discussed in the *Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep.

- **ResNet-50 v1.5 model.** The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, tensor cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training.

**Announcements**

In the next release, we will no longer support Ubuntu 16.04. Release 19.07 will instead support Ubuntu 18.04.

**Known Issues**

- There is a known performance regression with TensorFlow 1.13.1 for some networks when run with small batch sizes. As a workaround, increase the batch size.
- The AMP preview implementation is not compatible with Distributed Strategies. We recommend using Horovod for parallel training with AMP.
- AMP is not compatible with models the useResourceVariables for the global_step passed to the `tf.train.Optimizer.apply_gradients`. This will be fixed in the 19.07 NGC release.
- A known issue in TensorFlow results in the error **Cannot take the length of Shape with unknown rank** when training variable sized images with the Keras `model.fit` API. Details are provided [here](#) and a fix will be available in a future release.
- Support for CUDNN float32 Tensor Op Math mode first introduced in the 18.09 release is now deprecated in favor of Automatic Mixed Precision. It is scheduled to be removed after the 19.11 release.
- DLProf and Nsight Systems in the container will not work with GPU drivers newer than r418.
- There is a known issue when running the 19.06 TensorFlow container on a DGX-2 (or other systems having more than 8 GPUs) with RHEL 7.x (as opposed to Ubuntu) as the operating system. The known issue is that in some circumstances you will be shown the following message:
E tensorflow/stream_executor/cuda/cuda_driver.cc:300] failed call to cuInit:
CUDA_ERROR_OPERATING_SYSTEM: OS call failed or operation not supported on this OS
The NVIDIA container image of TensorFlow, release 19.05, is available on NGC.

**Contents of the TensorFlow container**

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- **Container image 19.05-py2 contains Python 2.7; 19.05-py3 contains Python 3.5.**
- NVIDIA CUDA 10.1 Update 1 including cuBLAS 10.1 Update 1
- NVIDIA cuDNN 7.6.0
- NVIDIA NCCL 2.4.6 (optimized for NVLink™)
- Horovod 0.16.1
- OpenMPI 3.1.3
- TensorBoard 1.13.1
- MLNX_OFED 3.4
- OpenSeq2Seq at commit 6e8835f
- TensorRT 5.1.5
- DALI 0.9.1 Beta
- Nsight Compute 10.1.163
- Nsight Systems 2019.3.1.94
- Tensor Core optimized example:
Driver Requirements

Release 19.05 is based on CUDA 10.1 Update 1, which requires NVIDIA Driver release 418.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.05 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 19.05 is based on TensorFlow 1.13.1.
- Latest version of NVIDIA CUDA 10.1 Update 1 including cuBLAS 10.1 Update 1
- Latest version of NVIDIA cuDNN 7.6.0
- Latest version of TensorRT 5.1.5
- Latest version of DALI 0.9.1 Beta
- Latest version of Nsight Compute 10.1.163
- Added the U-Net Medical Tensor Core example
- Added the NHWC plumbing to remove unnecessary format conversions between NHWC and NCHW. This feature is disabled by default, but can be enabled by setting the environment variable TF_ENABLE_NHWC=1.
Ubuntu 16.04 with April 2019 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)
For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In 
TensorFlow With TensorRT User Guide.

Key Features And Enhancements

- Integrated TensorRT 5.1.5 into TensorFlow. See the TensorRT 5.1.5 Release Notes 
  for a full list of new features.
- Improved examples at GitHub: TF-TRT, including README files, build scripts, 
  benchmark mode, ResNet models from TensorFlow official model zoo, etc...

Automatic Mixed Precision (AMP)
Automatic mixed precision converts certain float32 operations to operate in float16 
which can run much faster on Tensor Cores. Automatic mixed precision is built on two 
components:

- a loss scaling optimizer
- graph rewriter

For models already using a \texttt{tf.Optimizer()} for both \texttt{compute_gradients()} and 
\texttt{apply_gradients()} operations, automatic mixed precision can be enabled by defining 
the following environment variable before calling the usual float32 training script:

\begin{verbatim}
export TF_ENABLE_AUTO_MIXED_PRECISION=1
\end{verbatim}

Models implementing their own optimizers can use the graph rewriter on its own (while 
implementing loss scaling manually) with the following environment variable:

\begin{verbatim}
export TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE=1
\end{verbatim}

For more information about how to access and enable Automatic mixed precision 
for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the 

Tensor Core Examples
These examples focus on achieving the best performance and convergence from NVIDIA 
Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore 
you can get results much faster than training without tensor cores. This model is 
tested against each NGC monthly container release to ensure consistent accuracy and 
performance over time. This container includes the following tensor core examples.

- An implementation of the U-Net Medical model. The U-Net model is a 
  convolutional neural network for 2D image segmentation. This repository contains a
U-Net implementation as described in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation, without any alteration.

‣ An implementation of the SSD320 v1.2 model. The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository.

‣ An implementation of the Neural Collaborative Filtering (NCF) model. The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item.

‣ An implementation of the Bert model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUS for faster training times while maintaining target accuracy.

‣ An implementation of the U-Net Industrial Defect Segmentation model. This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007.

‣ An implementation of the GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep.

‣ An implementation of the ResNet-50 v1.5 model. The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel
multi-GPU training with Horovod, tensor cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training.

**Known Issues**

- There is a known performance regression with TensorFlow 1.13.1 for some networks when run with small batch sizes. As a workaround, increase the batch size.
- The AMP preview implementation is not compatible with Distributed Strategies. We recommend using Horovod for parallel training with AMP.
- A known issue in TensorFlow results in the error `Cannot take the length of Shape with unknown rank` when training variable sized images with the Keras `model.fit` API. Details are provided [here](#) and a fix will be available in a future release.
Chapter 15.
TENSORFLOW RELEASE 19.04

The NVIDIA container image of TensorFlow, release 19.04, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 10.1.105 including cuBLAS 10.1.0.105
- NVIDIA cuDNN 7.5.0
- NVIDIA NCCL 2.4.6 (optimized for NVLink™)
- Horovod 0.16.1
- OpenMPI 3.1.3
- TensorBoard 1.13.1
- MLNX_OFED 3.4
- OpenSeq2Seq at commit 6e8835f
- TensorRT 5.1.2
- DALI 0.8.1 Beta
- Nsight Compute 10.1.105
- Nsight Systems 2019.3.1.8
- Tensor Core optimized example:
Driver Requirements

Release 19.04 is based on CUDA 10.1, which requires NVIDIA Driver release 418.xx.x+. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver's compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.04 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Added the GNMT v2, U-Net Industrial Defect Segmentation, Bert, Neural Collaborative Filtering (NCF), and SSD320 v1.2 Tensor Core examples
- Latest version of NVIDIA NCCL 2.4.6
- Latest version of cuBLAS 10.1.0.105
- Latest version of DALI 0.8.1 Beta
- Latest version of Nsight Systems 2019.3.1.8
- Latest version of Horovod 0.16.1
- Improved stability for auto-tuning of fastest convolutional algorithms.
- Ubuntu 16.04 with March 2019 updates
Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

‣ Integrated TensorRT 5.1.2 RC into TensorFlow. See the TensorRT 5.1.2 RC Release Notes for a full list of new features.
‣ Improved examples at GitHub: TF-TRT, including README files, build scripts, benchmark mode, ResNet models from TensorFlow official model zoo, etc...

Automatic Mixed Precision (AMP)

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

‣ a loss scaling optimizer
‣ graph rewriter

For models already using a tf.Optimizer() for both compute_gradients() and apply_gradients() operations, automatic mixed precision can be enabled by defining the following environment variable before calling the usual float32 training script:

```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

Models implementing their own optimizers can use the graph rewriter on its own (while implementing loss scaling manually) with the following environment variable:

```
export TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the TensorFlow User Guide, along with Training With Mixed Precision.

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.

Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

‣ An implementation of the SSD320 v1.2 model. The SSD320 v1.2 model is based on the SSD: Single Shot MultiBox Detector paper, which describes SSD as “a method for detecting objects in images using a single deep neural network”. Our implementation is based on the existing model from the TensorFlow models repository.
- An implementation of the Neural Collaborative Filtering (NCF) model. The NCF model is a neural network that provides collaborative filtering based on implicit feedback, specifically, it provides product recommendations based on user and item interactions. The training data for this model should contain a sequence of user ID, item ID pairs indicating that the specified user has interacted with, for example, was given a rating to or clicked on, the specified item.

- An implementation of the Bert model. BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. This model is based on BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding paper. NVIDIA’s BERT is an optimized version of Google’s official implementation, leveraging mixed precision arithmetic and Tensor Cores on V100 GPUS for faster training times while maintaining target accuracy.

- An implementation of the U-Net Industrial Defect Segmentation model. This U-Net model is adapted from the original version of the U-Net model which is a convolutional auto-encoder for 2D image segmentation. U-Net was first introduced by Olaf Ronneberger, Philip Fischer, and Thomas Brox in the paper: U-Net: Convolutional Networks for Biomedical Image Segmentation. This work proposes a modified version of U-Net, called TinyUNet which performs efficiently and with very high accuracy on the industrial anomaly dataset DAGM2007.

- An implementation of the GNMT v2 model. The GNMT v2 model is similar to the one discussed in the Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation paper. The most important difference between the two models is in the attention mechanism. In our model, the output from the first LSTM layer of the decoder goes into the attention module, then the re-weighted context is concatenated with inputs to all subsequent LSTM layers in the decoder at the current timestep.

- An implementation of the ResNet-50 v1.5 model. The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, tensor cores (mixed precision) training, and static loss scaling for Tensor Cores (mixed precision) training.

**Known Issues**

- There is a known performance regression with TensorFlow 1.13.1 for some networks when run with small batch sizes. As a workaround, increase the batch size.
- The AMP preview implementation is not compatible with Distributed Strategies. We recommend using Horovod for parallel training with AMP.
- A known issue in TensorFlow results in the error `Cannot take the length of Shape with unknown rank` when training variable sized images with the Keras `model.fit` API. Details are provided [here](#) and a fix will be available in a future release.
Chapter 16.
TENSORFLOW RELEASE 19.03

The NVIDIA container image of TensorFlow, release 19.03, is available on NGC.

Contents of the TensorFlow container

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- **NVIDIA CUDA 10.1.105** including cuBLAS 10.1.105
- NVIDIA cuDNN 7.5.0
- **NVIDIA NCCL 2.4.3** (optimized for NVLink™)
- Horovod 0.16.0
- OpenMPI 3.1.3
- TensorFlow 1.13.1
- MLNX_OFED 3.4
- OpenSeq2Seq at commit 6e8835f
- TensorRT 5.1.2
- DALI 0.7 Beta
- Nsight Compute 10.1.105
- Nsight Systems 10.1.105
- Tensor Core optimized example:
- ResNet-50 v1.5
- Jupyter and JupyterLab:
  - Jupyter Client 5.2.4
  - Jupyter Core 4.4.0
  - JupyterLab 0.35.4
  - JupyterLab Server 0.2.0

**Driver Requirements**

Release 19.03 is based on CUDA 10.1, which requires NVIDIA Driver release 418.xx+. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384.111+ or 410. The CUDA driver’s compatibility package only supports particular drivers. For a complete list of supported drivers, see the CUDA Application Compatibility topic. For more information, see CUDA Compatibility and Upgrades.

**GPU Requirements**

Release 19.03 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

**Key Features and Enhancements**

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 19.03 is based on TensorFlow 1.13.1.
- Latest version of NVIDIA CUDA 10.1.105 including cuBLAS 10.1.105
- Latest version of NVIDIA cuDNN 7.5.0
- Latest version of NVIDIA NCCL 2.4.3
- Latest version of DALI 0.7 Beta
- Latest version of TensorRT 5.1.2
- Latest version of Horovod 0.16.0
- Latest version of TensorBoard 1.13.1
- Added the ResNet-50 v1.5 Tensor Core example
- Added Nsight Compute 10.1.105 and Nsight Systems 10.1.105 software
- Added support for TensorFlow Automatic Mixed Precision (TF-AMP); see below for more information.
- Ubuntu 16.04 with February 2019 updates
Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

‣ Integrated TensorRT 5.1.2 RC into TensorFlow. See the TensorRT 5.1.2 RC Release Notes for a full list of new features.
‣ Improved examples at GitHub: TF-TRT, including README files, build scripts, benchmark mode, ResNet models from TensorFlow official model zoo, etc...

Announcements

TensorRT 3.x is not longer supported, therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 5.x or later again.

For more information, see the Note in Serializing A Model In C++ or Serializing A Model In Python.

Automatic Mixed Precision (AMP)

Automatic mixed precision converts certain float32 operations to operate in float16 which can run much faster on Tensor Cores. Automatic mixed precision is built on two components:

‣ a loss scaling optimizer
‣ graph rewriter

For models already using a tf.Optimizer() for both compute_gradients() and apply_gradients() operations, automatic mixed precision can be enabled by defining the following environment variable before calling the usual float32 training script:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

Models implementing their own optimizers can use the graph rewriter on its own (while implementing loss scaling manually) with the following environment variable:

```bash
export TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE=1
```

For more information about how to access and enable Automatic mixed precision for TensorFlow, see Automatic Mixed Precision Training In TensorFlow from the TensorFlow User Guide, along with Training With Mixed Precision.

Tensor Core Examples

These examples focus on achieving the best performance and convergence from NVIDIA Volta Tensor Cores by using the latest deep learning example networks for training.
Each example model trains with mixed precision Tensor Cores on Volta, therefore you can get results much faster than training without tensor cores. This model is tested against each NGC monthly container release to ensure consistent accuracy and performance over time. This container includes the following tensor core examples.

- An implementation of the ResNet-50 v1.5 model. The ResNet-50 v1.5 model is a modified version of the original ResNet-50 v1 model. The difference between v1 and v1.5 is in the bottleneck blocks which requires downsampling, for example, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution. The following features were implemented in this model; data-parallel multi-GPU training with Horovod, Tensor Cores (mixed precision) training, and static loss scaling for tensor cores (mixed precision) training.

**Known Issues**

- There is a known performance regression with TensorFlow 1.13.1 for some networks when run with small batch sizes. As a workaround, increase the batch size.
- The AMP preview implementation is not compatible with Distributed Strategies. We recommend using Horovod for parallel training with AMP.
- If using or upgrading to a 3-part-version driver, for example, a driver that takes the format of \texttt{xxx.yy.zz}, you will receive a \texttt{Failed to detect NVIDIA driver version}. message. This is due to a known bug in the entry point script's parsing of 3-part driver versions. This message is non-fatal and can be ignored. This will be fixed in the 19.04 release.
The NVIDIA container image of TensorFlow, release 19.02, is available.

**Contents of TensorFlow**

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.2
- NVIDIA Collective Communications Library (NCCL) 2.3.7 (optimized for NVLink™)
- Horovod 0.15.1
- OpenMPI 3.1.3
- TensorFlow 1.12.2
- MLNX_OFED 3.4
- OpenSeq2Seq v18.12 at commit 59c70e7
- TensorRT 5.0.2
- DALI 0.6.1 Beta
- Jupyter and JupyterLab:
Driver Requirements

Release 19.02 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.02 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 19.02 is based on TensorFlow 1.13.0-rc0.
- Latest version of DALI 0.6.1 Beta
- Latest version of TensorBoard 1.12.2
- Added Jupyter and JupyterLab software in our packaged container.
- Latest version of jupyter_client 5.2.4
- Latest version of jupyter_core 4.4.0
- Ubuntu 16.04 with January 2019 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

- The following operators can now be converted from TensorFlow to TensorRT: ExpandDims, Reshape, Sigmoid, Sqrt, Square, Squeeze, StridedSlice and Tanh. For more information, see Supported Ops.
- You can manually insert quantization ranges (generated during quantization-aware training) to the graph, and then TF-TRT can use them during INT8 inference. That means calibration is not required with this feature. For more information, see INT8 Quantization.
Deprecated Features

‣ Support for TensorRT 3 has been removed.

Announcements

TensorRT 3.x is no longer supported, therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 5.x or later again.

For more information, see the Note in Serializing A Model In C++ or Serializing A Model In Python.

Known Issues

‣ Horovod and XLA cannot be used together due to a known issue in upstream TensorFlow. We expect this to be resolved in an upcoming release.

‣ There is a known performance regression with TensorFlow 1.13.0-rc0 for some networks when run with small batch sizes. As a workaround, increase the batch size.

‣ If using or upgrading to a 3-part-version driver, for example, a driver that takes the format of xxx.yy.zz, you will receive a Failed to detect NVIDIA driver version. message. This is due to a known bug in the entry point script's parsing of 3-part driver versions. This message is non-fatal and can be ignored. This will be fixed in the 19.04 release.
Chapter 18.
TENSORFLOW RELEASE 19.01

The NVIDIA container image of TensorFlow, release 19.01, is available.

Contents of TensorFlow
This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- Container image 19.01–py2 contains Python 2.7; 19.01–py3 contains Python 3.5.
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.2
- NCCL 2.3.7 (optimized for NVLink™)
- Horovod 0.15.1
- OpenMPI 3.1.3
- TensorBoard 1.12.0
- MLNX_OFED 3.4
- OpenSeq2Seq v18.12 at commit 59c70e7
- TensorRT 5.0.2
- DALI 0.6 Beta
Driver Requirements

Release 19.01 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 19.01 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

‣ TensorFlow container image version 19.01 is based on TensorFlow 1.12.0.
‣ Latest version of DALI 0.6 Beta
‣ Latest version of NVIDIA cuDNN 7.4.2
‣ Latest version of OpenMPI 3.1.3
‣ Ubuntu 16.04 with December 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Deprecated Features

‣ The image-classification examples were moved from /opt/tensorflow/nvidia-examples/inference/image-classification/scripts to https://github.com/tensorflow/tensorrt/tree/master/tftrt/examples/image-classification.
‣ The check_accuracy.py script, used to check whether the accuracy generated by the example matches with the expectation, was removed from the example. Refer to the published accuracy numbers to verify whether your generated accuracy numbers match with the expectation.

Announcements

Support for accelerating TensorFlow with TensorRT 3.x will be removed in a future release (likely TensorFlow 1.13). The generated plan files are not portable across platforms or TensorRT versions. Plans are specific to the exact GPU model they were built on (in addition to the platforms and the TensorRT version) and must be retargeted to the specific GPU in case you want to run them on a different GPU. Therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production
model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 4.x or later again.

For more information, see the Note in Serializing A Model In C++ or Serializing A Model In Python.

**Known Issues**

- Horovod and XLA cannot be used together due to a known issue in upstream TensorFlow. We expect this to be resolved in an upcoming release.
- If using or upgrading to a 3-part-version driver, for example, a driver that takes the format of `xxx.yy.zz`, you will receive a **Failed to detect NVIDIA driver version** message. This is due to a known bug in the entry point script’s parsing of 3-part driver versions. This message is non-fatal and can be ignored. This will be fixed in the 19.04 release.
Chapter 19.
TENSORFLOW RELEASE 18.12

The NVIDIA container image of TensorFlow, release 18.12, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

Container image 18.12-py2 contains Python 2.7; 18.12-py3 contains Python 3.5.

- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.1
- NCCL 2.3.7 (optimized for NVLink™)
- Horovod 0.15.1
- OpenMPI 3.1.2
- TensorBoard 1.12.0
- MLNX_OFED 3.4
- OpenSeq2Seq v18.12 at commit 59c70e7
- TensorRT 5.0.2
- DALI 0.5.0 Beta
Driver Requirements

Release 18.12 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

GPU Requirements

Release 18.12 supports CUDA compute capability 6.0 and higher. This corresponds to GPUs in the Pascal, Volta, and Turing families. Specifically, for a list of GPUs that this compute capability corresponds to, see CUDA GPUs. For additional support details, see Deep Learning Frameworks Support Matrix.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

‣ TensorFlow container image version 18.12 is based on TensorFlow 1.12.0.
‣ Latest version of DALI 0.5.0 Beta.
‣ OpenSeq2Seq’s custom CTC decoder is now pre-built in the container.
‣ The tensorflow.contrib.nccl module has been moved into core as tensorflow.python.ops.nccl_ops. User scripts may need to be updated accordingly. No changes are required for Horovod users. For an example of using Horovod, refer to the nvidia-examples/cnn/ directory.
‣ Inference image classification examples have been removed from the container and are now available at: GitHub: TensorFlow/TensorRT Integration.
‣ Ubuntu 16.04 with November 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Deprecated Features

‣ The image-classification examples were moved from /opt/tensorflow/nvidia-examples/inference/image-classification/scripts to https://github.com/tensorflow/tensorrt/tree/master/tftrt/examples/image-classification.
‣ The check_accuracy.py script, used to check whether the accuracy generated by the example matches with the expectation, was removed from the example. Refer to the published accuracy numbers to verify whether your generated accuracy numbers match with the expectation.

Announcements

Support for accelerating TensorFlow with TensorRT 3.x will be removed in a future release (likely TensorFlow 1.13). The generated plan files are not portable across
platforms or TensorRT versions. Plans are specific to the exact GPU model they were built on (in addition to the platforms and the TensorRT version) and must be retargeted to the specific GPU in case you want to run them on a different GPU. Therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 4.x or later again.

For more information, see the Note in Serializing A Model In C++ or Serializing A Model In Python.

Known Issues

- OpenSeq2Seq is only supported in the Python 3 container.
- Horovod and XLA cannot be used together due to a known issue in upstream TensorFlow. We expect this to be resolved in an upcoming release.
Chapter 20.
TENSORFLOW RELEASE 18.11

The NVIDIA container image of TensorFlow, release 18.11, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.1
- NCCL 2.3.7 (optimized for NVLink™)
- Horovod 0.15.1
- OpenMPI 3.1.2
- TensorBoard 1.12.0
- MLNX_OFED 3.4
- OpenSeq2Seq v18.11 at commit 4b95346
- TensorRT 5.0.2
- DALI 0.4.1 Beta
Driver Requirements

Release 18.11 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 18.11 is based on TensorFlow 1.12.0-rc2.
- Latest version of Horovod 0.15.1.
- Latest version of NCCL 2.3.7.
- Latest version of NVIDIA cuDNN 7.4.1.
- Latest version of TensorRT 5.0.2
- Latest version of DALI 0.4.1 Beta.
- Bug fixes and improvements for TensorFlow-TensorRT (TF-TRT) integration.
- Added an object detection example to workspace/nvidia-examples/inference/object-detection.
- Ubuntu 16.04 with October 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features and Enhancements

- Added support for dilated convolution.
- Fixed a bug in the Identity op.
- Fixed a bug in the Relu6 op.
- Support added to allow empty const tensor.
- Added object detection example to nvidia-examples/inference.

Known Issues

- In the TF-TRT API, the minimum_segment_size argument default value is 3. In the image classification examples under nvidia-examples/inference, we define a command line argument for minimum_segment_size which has its own default value. In 18.10, the default value was 7 and in 18.11 we changed it to 2. Smaller values for this argument would cause to convert more TensorFlow nodes to TensorRT which typically should improve the performance, however, we have observed cases where the performance gets worse. In particular, Resnet-50 with smaller batch sizes gets slower with minimum_segment_size=2 comparing to minimum_segment_size=7.
Announcements

Support for accelerating TensorFlow with TensorRT 3.x will be removed in a future release (likely TensorFlow 1.13). The generated plan files are not portable across platforms or TensorRT versions. Plans are specific to the exact GPU model they were built on (in addition to the platforms and the TensorRT version) and must be retargeted to the specific GPU in case you want to run them on a different GPU. Therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 4.x or later again.

For more information, see the Note in Serializing A Model In C++ or Serializing A Model In Python.

Known Issues

OpenSeq2Seq is only supported in the Python 3 container.
Chapter 21.
TENSORFLOW RELEASE 18.10

The NVIDIA container image of TensorFlow, release 18.10, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

Containers image 18.10-py2 contains Python 2.7; 18.10-py3 contains Python 3.5.

- NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.4.0
- NCCL 2.3.6 (optimized for NVLink™)
- Horovod 0.13.10
- OpenMPI 3.1.2
- TensorBoard 1.10.0
- MLNX_OFED 3.4
- OpenSeq2Seq v18.10 at commit 655eb65
- TensorRT 5.0.0 RC
- DALI 0.4 Beta
Driver Requirements

Release 18.10 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 18.10 is based on TensorFlow 1.10.0.
- Latest version of NCCL 2.3.6.
- Latest version of DALI 0.4 Beta
- Latest version of OpenMPI 3.1.2
- Fixed a bug in the ResNet example script when using NHWC data format.
- Fixed several issues when accelerating inference in TensorFlow with TensorRT including support for ReLu6, Identity, and dilated convolutions.
- Ubuntu 16.04 with September 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

- New examples at nvidia-examples/tftrt with good accuracy and performance.
- Built TF-TRT with TensorRT 5.0.0 which introduces the new TensorRT APIs into TF-TRT.
- Added support for the TensorFlow operator RELU6 (using Relu6(x) = min(ReLU(x), 6)).
- Made improvements in the image classification example, such as bug fixes and using the dynamic_op feature.

Limitations

- Not all the new TensorRT 5.0.0 features are supported yet in TF-TRT including INT8 quantization ranges and the plugins registry.
- We have only tested image classification models with TF-TRT including the ones we have provided in our examples inside the container (nvidia-examples/tftrt). This means object detection, translation (convolutional and recurrent based) are not yet supported due to either functionality or performance limitations.
- TF-TRT has an implementation of optimizing the TensorFlow graph by specifying appropriate TensorFlow session arguments without using the Python TF-TRT...
API (**create_inference_graph**), however, we have not thoroughly tested this functionality yet, therefore, we don’t support it.

**Known Issues**

- Running inference with batch sizes larger than the maximum batch size is not supported by TensorRT.
- Due to certain logs (errors or warnings) of TF-TRT, they could be misleading and point to the TensorRT graph as broken while it’s not. It is recommended to check whether there is any TensorRT op in the graph (the type of op is **TRTEngineOp**). If there is not TensorRT ops in the graph, that means no conversion has happened and the inference should fall back to the native TensorFlow. Currently, the best way to verify whether a frozen graph resulting from the conversion is not broken is to run inference on it and check the accuracy of the results.
- There are operators that are not supported by either TensorRT or the conversion algorithm. The convertor is supposed to skip these ops but this skip may not happen properly due to bugs. One way to get around this problem is to increase the value of the `minimum_segment_size` parameter and hope that the subgraphs that contain those ops are too small and remain out of the conversion.
- We have observed functionality problems in optimizing:
  - NASNet models with TF-TRT in FP16 precision mode.
  - ResNet, MobileNet, and NASNet models with TF-TRT in INT8 precision mode.

**Announcements**

Support for accelerating TensorFlow with TensorRT 3.x will be removed in a future release (likely TensorFlow 1.13). The generated plan files are not portable across platforms or TensorRT versions. Plans are specific to the exact GPU model they were built on (in addition to the platforms and the TensorRT version) and must be retargeted.
to the specific GPU in case you want to run them on a different GPU. Therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 4.x or later again.

For more information, see the Note in Serializing A Model In C++ or Serializing A Model In Python.

**Known Issues**

OpenSeq2Seq is only supported in the Python 3 container.
Chapter 22.
TENSORFLOW RELEASE 18.09

The NVIDIA container image of TensorFlow, release 18.09, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

<table>
<thead>
<tr>
<th>Container image 18.09–py2 contains Python 2.7; 18.09–py3 contains Python 3.5.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA CUDA 10.0.130 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 10.0.130</td>
</tr>
<tr>
<td>NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.3.0</td>
</tr>
<tr>
<td>NCCL 2.3.4 (optimized for NVLink™)</td>
</tr>
<tr>
<td>Horovod™ 0.13.10</td>
</tr>
<tr>
<td>OpenMPI 3.0.0</td>
</tr>
<tr>
<td>TensorFlow 1.10.0</td>
</tr>
<tr>
<td>MLNX_OFED 3.4</td>
</tr>
<tr>
<td>OpenSeq2Seq v18.09 at commit 694a230</td>
</tr>
<tr>
<td>TensorRT 5.0.0 RC</td>
</tr>
<tr>
<td>DALI 0.2 Beta</td>
</tr>
</tbody>
</table>
Driver Requirements

Release 18.09 is based on CUDA 10, which requires NVIDIA Driver release 410.xx. However, if you are running on Tesla (Tesla V100, Tesla P4, Tesla P40, or Tesla P100), you may use NVIDIA driver release 384. For more information, see CUDA Compatibility and Upgrades.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 18.09 is based on TensorFlow 1.10.0.
- Latest version of cuDNN 7.3.0.
- Latest version of CUDA 10.0.130 which includes support for DGX-2, Turing, and Jetson Xavier.
- Latest version of cuBLAS 10.0.130.
- Latest version of NCCL 2.3.4.
- Latest version of TensorRT 5.0.0 RC.
- Latest version of TensorBoard 1.10.0.
- Latest version of DALI 0.2 Beta
- Added support for CUDNN float32 Tensor Op Math mode, which enables float32 models to use Tensor Cores on supported hardware, at the cost of reduced precision. This is disabled by default, but can be enabled by setting the environment variables `TF_ENABLE_CUDNN_TENSOR_OP_MATH_FP32=1` (for convolutions) or `TF_ENABLE_CUDNN_RNN_TENSOR_OP_MATH_FP32=1` (for RNNs that use the `cudnn_rnn` op). This feature is currently considered experimental.
- Renamed the existing environment variable `TF_ENABLE_TENSOR_OP_MATH_FP32` to `TF_ENABLE_CUBLAS_TENSOR_OP_MATH_FP32`.

When using any of the `TF_ENABLE_*_TENSOR_OP_MATH_FP32` environment variables, it is recommended that models also use loss scaling to avoid numerical issues during training. For more information about loss scaling, see Training With Mixed Precision.

- Enhanced `tf.contrib.layers.layer_norm` by adding a `use_fused_batch_norm` parameter that improves performance. This parameter is disabled by default, but can be enabled by setting it to True.
- Ubuntu 16.04 with August 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.
Key Features And Enhancements

- New examples at nvidia-examples/tftrt with good accuracy and performance.
- Built TF-TRT with TensorRT 5.0.0 which introduces the new TensorRT APIs into TF-TRT.

Limitations

- Not all the new TensorRT 5.0.0 features are supported yet in TF-TRT including INT8 quantization ranges and the plugins registry.
- We have only tested image classification models with TF-TRT including the ones we have provided in our examples inside the container (nvidia-examples/tftrt). This means object detection, translation (convolutional and recurrent based) are not yet supported due to either functionality or performance limitations.
- TF-TRT has an implementation of optimizing the TensorFlow graph by specifying appropriate TensorFlow session arguments without using the Python TF-TRT API (create_inference_graph), however, we have not thoroughly tested this functionality yet, therefore, we don’t support it.
- TF-TRT has an implementation of the dynamic conversion of a TensorFlow graph, but we have not thoroughly tested this functionality yet, therefore, we don’t support it.

Known Issues

- Running inference with batch sizes larger than the maximum batch size is not supported by TensorRT.
- Due to certain logs (errors or warnings) of TF-TRT, they could be misleading and point to the TensorRT graph as broken while it’s not. It is recommended to check whether there is any TensorRT op in the graph (the type of op is TRT EngineOp). If there is not TensorRT ops in the graph, that means no conversion has happened and the inference should fall back to the native TensorFlow. Currently, the best way to verify whether a frozen graph resulting from the conversion is not broken is to run inference on it and check the accuracy of the results.
- There are operators that are not supported by either TensorRT or the conversion algorithm. The convertor is supposed to skip these ops but this skip may not happen properly due to bugs. One way to get around this problem is to increase the value of the minimum_segment_size parameter and hope that the subgraphs that contain those ops are too small and remain out of the conversion.
- We have observed functionality problems in optimizing:
  - NASNet models with TF-TRT in FP16 precision mode.
ResNet, MobileNet, and NASNet models with TF-TRT in INT8 precision mode.

TF-TRT cannot optimize certain models such as ResNet in INT8 precision mode because of a lacking feature in TensorRT regarding the dimensionality of tensors. Usually, increasing the value of `minimum_segment_size` is a workaround by removing those unsupported dimensions out of the TensorRT sub-graph.

TF-TRT doesn't work with TensorFlow Lite due to a TensorRT bug that causes Flatbuffer symbols to be exposed. This means you cannot import both `tf.contrib.tensorrt` and `tf.lite` in the same process.

We have observed a bit low accuracy on image classification models with TF-TRT on Jetson AGX Xavier.

INT8 calibration on `mobilenet_v1` and `mobilenet_v2` using TF-TRT fails if the calibration dataset has only one element.

**Announcements**

Support for accelerating TensorFlow with TensorRT 3.x will be removed in a future release (likely TensorFlow 1.13). The generated plan files are not portable across platforms or TensorRT versions. Plans are specific to the exact GPU model they were built on (in addition to the platforms and the TensorRT version) and must be retargeted to the specific GPU in case you want to run them on a different GPU. Therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 4.x or later again.

For more information, see the Note in [Serializing A Model In C++](#) or [Serializing A Model In Python](#).

**Known Issues**

- OpenSeq2Seq is only supported in the Python 3 container.
- The `build_imagenet_data` scripts have a missing dependency on the `axel` application. This can be resolved by issuing the following command:

```bash
apt-get update &
apt-get install axel
```
The NVIDIA container image of TensorFlow, release 18.08, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.425
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.2.1
- NCCL 2.2.13 (optimized for NVLink™)
- Horovod™ 0.12.1
- OpenMPI™ 3.0.0
- TensorFlow 1.9.0
- MLNX_OFED 3.4
- OpenSeq2Seq v0.5 at commit 83e96551.
- TensorRT 4.0.1
- DALI 0.1.2 Beta
Driver Requirements

Release 18.08 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

‣ TensorFlow container image version 18.08 is based on TensorFlow 1.9.0.
‣ Latest version of cuDNN 7.2.1.
‣ Latest version of DALI 0.1.2 Beta.
‣ Latest version of TensorBoard 1.9.0.
‣ Added experimental support for float16 data type in Horovod, allowing functions such as all_reduce to accept tensors in float16 precision. (This functionality is not yet integrated into multi-GPU training examples).
‣ Ubuntu 16.04 with July 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

‣ TensorRT conversion has been integrated into optimization pass. The tensorflow/contrib/tensorrt/test/test_tftrt.py script has an example showing the use of optimization pass.

Limitations

‣ TensorRT conversion relies on static shape inference, where the frozen graph should provide explicit dimension on all ranks other than the first batch dimension.
‣ Batchsize for converted TensorRT engines are fixed at conversion time. Inference can only run with batchsize smaller than the specified number.
‣ Current supported models are limited to CNNs. Object detection models and RNNs are not yet supported.
‣ Current optimization pass does not support INT8 yet.

Known Issues

‣ Input tensors are required to have rank 4 for quantization mode (INT8 precision).

Announcements

Starting with the next major version of CUDA release, we will no longer provide updated Python 2 containers and will only update Python 3 containers.
Known Issues

- The DALI integrated ResNet-50 samples in the 18.08 NGC TensorFlow container has lower than expected accuracy and performance results. We are working to address the issue in the next release.

- There is a known performance regression in the inference benchmarks for ResNet-50. We haven't seen this regression in the inference benchmarks for VGG or training benchmarks for any network. The cause of the regression is still under investigation.
Chapter 24.
TENSORFLOW RELEASE 18.07

The NVIDIA container image of TensorFlow, release 18.07, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

  Container image 18.07-py2 contains Python 2.7; 18.07-py3 contains Python 3.5.

- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.425
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.4
- NCCL 2.2.13 (optimized for NVLink™)
- Horovod™ 0.12.1
- OpenMPI™ 3.0.0
- TensorBoard 1.8.0
- MLNX_OFED 3.4
- OpenSeq2Seq v0.4 at commit 98ad236a.
- TensorRT 4.0.1
- DALI 0.1 Beta
Driver Requirements

Release 18.07 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 18.07 is based on TensorFlow 1.8.0.
- Added support for DALI 0.1 Beta.
- Latest version of CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.425.
- Ubuntu 16.04 with June 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

- Added TensorRT 4.0 API support with extended layer support. This support includes the FullyConnected layer and BatchedMatMul op.
- Resource management added, where memory allocation is uniformly managed by TensorFlow.
- Bug fixes and better error handling in conversion.

Limitations

- TensorRT conversion relies on static shape inference, where the frozen graph should provide explicit dimension on all ranks other than the first batch dimension.
- Batchsize for converted TensorRT engines are fixed at conversion time. Inference can only run with batchsize smaller than the specified number.
- Current supported models are limited to CNNs. Object detection models and RNNs are not yet supported.

Known Issues

- Input tensors are required to have rank 4 for quantization mode (INT8 precision).

Announcements

Starting with the next major version of CUDA release, we will no longer provide updated Python 2 containers and will only update Python 3 containers.

Known Issues

There are no known issues in this release.
Chapter 25.
TENSORFLOW RELEASE 18.06

The NVIDIA container image of TensorFlow, release 18.06, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

  Container image 18.06-py2 contains Python 2.7; 18.06-py3 contains Python 3.5.

- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.4
- NCCL 2.2.13 (optimized for NVLink™)
- Horovod™ 0.12.1
- OpenMPI™ 3.0.0
- TensorFlow 1.8.0
- MLNX_OFED 3.4
- OpenSeq2Seq v0.2 at commit a4f627e
- TensorRT 4.0.1

Driver Requirements

Release 18.06 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.
Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

‣ TensorFlow container image version 18.06 is based on TensorFlow 1.8.0.
‣ Updated scripts and README in nvidia-examples/cnn/ to use cleaner implementation with high-level TensorFlow APIs including Datasets, Layers, and Estimators. Multi-GPU support in these scripts is now provided exclusively using Horovod/MPI.
‣ Fixed incorrect network definition in resnet18 and resnet34 models in nvidia-examples/cnn/.
‣ Updated scripts and README in nvidia-examples/build_imagenet_data/ to improve usability and ensure that the dataset is correctly downloaded and resized.
‣ Added support for TensorRT 4 features to TensorFlow-TensorRT integration.
‣ Includes integration with TensorRT 4.0.1
‣ Optimized CPU bilinear image resize kernel to improve performance of input pipeline.
‣ Ubuntu 16.04 with May 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Key Features And Enhancements

‣ Added TensorRT 4.0 API support with extended layer support. This support includes the FullyConnected layer and BatchedMatMul op.
‣ Resource management added, where memory allocation is uniformly managed by TensorFlow.
‣ Bug fixes and better error handling in conversion.

Limitations

‣ TensorRT conversion relies on static shape inference, where the frozen graph should provide explicit dimension on all ranks other than the first batch dimension.
‣ Batchsize for converted TensorRT engines are fixed at conversion time. Inference can only run with batchsize smaller than the specified number.
‣ Current supported models are limited to CNNs. Object detection models and RNNs are not yet supported.

Known Issues

‣ Input tensors are required to have rank 4 for quantization mode (INT8 precision).
**Announcements**

Starting with the next major version of CUDA release, we will no longer provide updated Python 2 containers and will only update Python 3 containers.

**Known Issues**

There are no known issues in this release.
Chapter 26.
TENSORFLOW RELEASE 18.05

The NVIDIA container image of TensorFlow, release 18.05, is available.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

Container image 18.05-py2 contains Python 2.7; 18.05-py3 contains Python 3.5.

- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.2
- NCCL 2.1.15 (optimized for NVLink™)
- Horovod™ 0.12.1
- OpenMPI™ 3.0.0
- TensorBoard 1.7.0
- MLNX_OFED 3.4
- OpenSeq2Seq v0.2

Driver Requirements

Release 18.05 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.
Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- TensorFlow container image version 18.05 is based on TensorFlow 1.7.0.
- For developers needing more visibility between network layer calls and CUDA kernel calls, we’ve added support for basic NVTX ranges to the TensorFlow executor. Nsight Systems or the NVIDIA Visual Profiler, with NVTX ranges, are able to display each TensorFlow op demarcated by an NVTX range named by the op. NVTX ranges are enabled by default but can be disabled by setting the environment variable `TF_DISABLE_NVTX_RANGES=1`.
- Optimized input pipeline in `nvcnn.py` and `nvcnn_hvd.py` by casting back to uint8 immediately after image preprocessing.
- Added OpenSeq2Seq v0.2 to the base container.
- Includes integration with TensorFlow 3.0.4
- Ubuntu 16.04 with April 2018 updates

Accelerating Inference In TensorFlow With TensorRT (TF-TRT)

For step-by-step instructions on how to use TF-TRT, see Accelerating Inference In TensorFlow With TensorRT User Guide.

Attention

Support for accelerating TensorFlow with TensorRT 3.x will be removed in a future release (likely TensorFlow 1.13). The generated plan files are not portable across platforms or TensorRT versions. Plans are specific to the exact GPU model they were built on (in addition to the platforms and the TensorRT version) and must be retargeted to the specific GPU in case you want to run them on a different GPU. Therefore, models that were accelerated using TensorRT 3.x will no longer run. If you have a production model that was accelerated with TensorRT 3.x, you will need to convert your model with TensorRT 4.x or later again.

For more information, see the Note in Serializing A Model In C++ or Serializing A Model In Python.

Key Features And Enhancements

- TensorRT backend accelerates inference performance for frozen TensorFlow models.
- Automatic segmenter that recognizes TensorRT compatible subgraphs and converts them into TensorRT engines. TensorRT engines are wrapped with TensorFlow custom ops that moves the execution of the subgraph to TensorRT backend for optimized performance, while fall back to TensorFlow for non-TensorRT compatible ops.
Supported networks are slim classification networks including ResNet, VGG, and Inception.
- Mixed precision and quantization are supported.

Limitations
- Conversion relies on static shape inference, where the frozen graph should provide explicit dimension on all ranks other than the first batch dimension.
- Batchsize for converted TensorRT engines are fixed at conversion time. Inference can only run with batchsize smaller than the specified number.
- Current supported models are limited to CNNs. Object detection models and RNNs are not yet supported.
- Resource management is not integrated, therefore, ensure you limit the memory claimed by TensorFlow in order for TensorRT to acquire the necessary resource. To limit the memory, use `setting per_process_gpu_memory_fraction to < 1.0` and pass it to session creation, for example:

```python
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333) sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
```

Deprecated Features
In the 18.05 container, you need to create a TensorFlow session with the `per_process_gpu_memory_fraction` option. With the resource management fully integrated, you no longer need to reserve GPU memory from TensorFlow. Therefore, the option is not necessary for mixed TensorFlow-TensorRT (TF-TRT) model.

Known Issues
- The TensorRT engine only accepts input tensor with `rank == 4`.

Announcements
Starting with the next major version of CUDA release, we will no longer provide Python 2 containers and will only maintain Python 3 containers.

Known Issues
- There are no known issues in this release.
Chapter 27.
TENSORFLOW RELEASE 18.04

The NVIDIA container image of TensorFlow, release 18.04, is available.

**Contents of TensorFlow**

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

  Container image 18.04-py2 contains Python 2.7; 18.04-py3 contains Python 3.5.

- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)

- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.1

- NCCL 2.1.15 (optimized for NVLink™)

- Horovod™ 0.11.3

- OpenMPI™ 3.0.0

- TensorBoard 0.4.0-rc1

- MLNX_OFED 3.4

**Driver Requirements**

Release 18.04 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.
Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- **TensorFlow** container image version 18.04 is based on **TensorFlow 1.7.0**.
- Added the Mellanox user-space InfiniBand driver to the container.
- Latest version of MLNX_OFED 3.4
- Added support for TensorRT integration in TensorFlow. For functionality details, see **TensorRT Integration Speeds Up TensorFlow Inference** and the example in the **nvidia-examples/tftrt** directory.
- Improved **nvidia_examples/nvcnn.py** and **nvcnn_hvd.py** to ensure ResNet-50 model converges correctly out of the box. See **Changelog** at the top of **nvidia_examples/nvcnn.py** for more details.
- Enabled Tensor Op math for cuDNN-based RNNs in FP16 precision. This is enabled by default, but can be disabled by setting the environment variable **TF_DISABLE_CUDNN_RNN_TENSOR_OP_MATH=1**.
- Includes integration with **TensorRT 3.0.4**
- Latest version of NCCL 2.1.15
- Ubuntu 16.04 with March 2018 updates

Announcements

Starting with the next major version of CUDA release, we will no longer provide Python 2 containers and will only maintain Python 3 containers.

Known Issues

There is a degraded performance for graph construction time of grouped convolutions. For more information, see **Support for depthwise convolution by groups**.
Chapter 28.  
TENSORFLOW RELEASE 18.03

The NVIDIA container image of TensorFlow, release 18.03, is available.

**Contents of TensorFlow**

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

- NVIDIA CUDA 9.0.176 (see Errata section and 2.1) including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.333 (see section 2.3.1)
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.1.1
- NCCL 2.1.2 (optimized for NVLink™)
- Horovod™ 0.11.3
- OpenMPI™ 3.0.0
- TensorBoard 0.4.0-rc1

**Driver Requirements**

Release 18.03 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

**Key Features and Enhancements**

This TensorFlow release includes the following key features and enhancements.
TensorFlow container image version 18.03 is based on TensorFlow 1.4.0.

- Latest updates to OpenSeq2Seq module
- Latest version of cuBLAS 9.0.333
- Latest version of cuDNN 7.1.1
- Latest version of OpenMPI 3.0.0
- Latest version of Horovod 0.11.3
- Latest version of TensorBoard 0.4.0-rc1
- Ubuntu 16.04 with February 2018 updates

**Announcements**

Starting with the next major version of CUDA release, we will no longer provide Python 2 containers and will only maintain Python 3 containers.

**Known Issues**

There are no known issues in this release.
Chapter 29.
TENSORFLOW RELEASE 18.02

The NVIDIA container image of TensorFlow, release 18.02, is available.

TensorFlow container image version 18.02 is based on TensorFlow 1.4.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04

- NVIDIA CUDA 9.0.176 including:
  - CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.282 Patch 2 which is installed by default
  - cuBLAS 9.0.234 Patch 1 as a debian file. Installing Patch 1 by issuing the `dpkg -i /opt/cuda-cublas-9-0_9.0.234-1_amd64.deb` command is the workaround for the known issue described below.
  - NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.5
  - NVIDIA® Collective Communications Library™ (NCCL) 2.1.2 (optimized for NVLink™)
  - Horovod™ 0.11.2
Driver Requirements

Release 18.02 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

‣ Updated OpenSeq2Seq example to include latest bug fixes
‣ Latest version of cuBLAS
‣ Ubuntu 16.04 with January 2018 updates

Known Issues

‣ cuBLAS 9.0.282 regresses RNN seq2seq FP16 performance for a small subset of input sizes. This issue should be fixed in the next update. As a workaround, install cuBLAS 9.0.234 Patch 1 by issuing the `dpkg -i /opt/cuda-cublas-9-0_9.0.234-1_amd64.deb` command.

‣ The `broadcast` and `reduce` (but not `all_reduce`) functions in the `tf.contrib.nccl` module cause an error when executed as part of a graph. This issue should be fixed in the next update. The multi-GPU training example script `nvidia-examples/cnn/nvcnn.py` includes a workaround for the `nccl.broadcast` function so that the script still runs correctly.

The Horovod example script `nvidia-examples/cnn/nvcnn_hvd.py` is not affected by this issue.

‣ Some Python 3 codes may encounter errors when handling text strings containing non-Latin characters. This can be fixed by setting an environment variable with the following command:

```
$ export LC_ALL=C.UTF-8
```

This issue should be fixed in the next update.
The NVIDIA container image of TensorFlow, release 18.01, is available. TensorFlow container image version 18.01 is based on TensorFlow 1.4.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed as a system Python module.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 9.0.176 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.282
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.5
- NVIDIA® Collective Communications Library™ (NCCL) 2.1.2 (optimized for NVLink™)
- Horovod™ 0.11.2

Driver Requirements

Release 18.01 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.
- Addition of Python 3 package
- **Horovod** is now pre-installed in the container
- Updated OpenSeq2Seq example to include latest bug fixes
- Latest version of cuBLAS
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with December 2017 updates

**Known Issues**

cuBLAS 9.0.282 regresses RNN seq2seq FP16 performance for a small subset of input sizes. As a workaround, revert back to the 11.12 container.
Chapter 31.  
TENSORFLOW RELEASE 17.12

The NVIDIA container image of TensorFlow, release 17.12, is available.  
TensorFlow container image version 17.12 is based on TensorFlow 1.4.0.

Contents of TensorFlow  
This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/local/[bin,lib] directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:
- Ubuntu 16.04
- NVIDIA CUDA 9.0.176 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.234
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.5
- NVIDIA® Collective Communications Library™ (NCCL) 2.1.2 (optimized for NVLink™)

Driver Requirements  
Release 17.12 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements  
This TensorFlow release includes the following key features and enhancements.
- Latest version of CUDA
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with November 2017 updates

**Known Issues**

A corner case of float16 reductions is known to give the wrong result of Maxwell and earlier architectures. This will be fixed in a future release.
Chapter 32.
TENSORFLOW RELEASE 17.11

The NVIDIA container image of TensorFlow, release 17.11, is available.

TensorFlow container image version 17.11 is based on TensorFlow 1.3.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed into the `/usr/local/` [bin,lib] directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA 9.0.176 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) 9.0.234
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.4
- NVIDIA® Collective Communications Library™ (NCCL) 2.1.2 (optimized for NVLink™)

Driver Requirements

Release 17.11 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Added support for float16 data type and Tensor Core math in batched matrix multiply operations.
- Added example script `nvidia-examples/cnn/nvcnn_hvd.py`, which demonstrates use of the Horovod library for multi-node training.
- Added `Dockerfile.horovod` demonstrating how to build a Docker container with the Horovod library and MPI support.
- Added OpenSeq2Seq example demonstrating sequence-to-sequence model training in `nvidia-examples/OpenSeq2Seq`.
- Latest version of CUDA
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with October 2017 updates

**Known Issues**

There are no known issues in this release.
The NVIDIA container image of TensorFlow, release 17.10, is available. TensorFlow container image version 17.10 is based on TensorFlow 1.3.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/local/[bin,lib] directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 9.0
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.3
- NVIDIA® Collective Communications Library™ (NCCL) 2.0.5 (optimized for NVLink™)

Driver Requirements

Release 17.10 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Added PNG image support to nvcnn.py.
- Fixed issue with batchnorm op that broke backwards compatibility in the previous release.
Renamed the `TF_ENABLE_TENSOR_OP_MATH` (default=1) environment variable to `TF_DISABLE_TENSOR_OP_MATH` (default=0).

- Upgraded Bazel to version 0.5.4.
- Worked around hash mismatches in third-party source downloads.
- Enabled compilation flags `-march=sandybridge -mtune=broadwell`.
- Updated Eigen to the top of the tree and removed custom patches.
- Latest version of CUDA
- Latest version of cuDNN
- Latest version of NCCL
- Ubuntu 16.04 with September 2017 updates

**Known Issues**

There are no known issues in this release.
Chapter 34.
TENSORFLOW RELEASE 17.09

The NVIDIA container image of TensorFlow, release 17.09, is available. TensorFlow container image version 17.09 is based on TensorFlow 1.3.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/local/[bin,lib] directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 9.0
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 7.0.2
- NVIDIA® Collective Communications Library™ (NCCL) 2.0.5 (optimized for NVLink™)

Driver Requirements

Release 17.09 is based on CUDA 9, which requires NVIDIA Driver release 384.xx.

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Tensor Core operation support in TensorFlow is enabled by default on Volta for FP16 convolutions and matrix multiplies, which should give a speedup for FP16 models.
- Added experimental support for:
FP16 training in `nvidia-examples/cnn/nvcnn.py`
- FP16 input/output in the fused batch normalization operation (`tf.nn.fused_batch_norm`)
- Tensor Core operation in FP16 convolutions and matrix multiplications
  - Added the `TF_ENABLE_TENSOR_OP_MATH` parameter which enables and disables Tensor Core operation (defaults to enabled).
- Tensor Core operation in FP32 matrix multiplications
  - Added the `TF_ENABLE_TENSOR_OP_MATH_FP32` parameter which enables and disables Tensor Core operation for float32 matrix multiplications (defaults to disabled because it reduces precision).
- Increased the `TF_AUTOTUNE_THRESHOLD` parameter which improves auto-tune stability.
- Increased the `CUDADEVICE_MAXCONNECTIONS` parameter which solves performance issues related to streams on Tesla K80 GPUs.
- Enhancements to `nvidia-examples/cnn/nvcnn.py`
  - Fixed a bug where the final layer was wrong when running in evaluation mode.
  - Changed `is_training` to a constant instead of a placeholder for better performance and reduced memory use.
  - Merged gradients for all layers into a single NCCL call for better performance.
  - Disabled use of XLA by default for better performance.
  - Disabled `zero_debias_moving_mean` in batch normalization operation.

Latest version of CUDA
Latest version of cuDNN
Latest version of NCCL
Ubuntu 16.04 with August 2017 updates

**Known Issues**

There are no known issues in this release.
Chapter 35.
TENSORFLOW RELEASE 17.07

The NVIDIA container image of TensorFlow, release 17.07, is available. TensorFlow container image version 17.07 is based on TensorFlow 1.2.1.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/local/[bin,lib] directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61.2 including CUDA® Basic Linear Algebra Subroutines library™ (cuBLAS) Patch 2
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.21
- NVIDIA® Collective Communications Library™ (NCCL) 2.0.3 (optimized for NVLink™)

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Small bug-fixes in evaluation mode of nvidia-examples/cnn/nvcnn.py
- Ubuntu 16.04 with June 2017 updates

Known Issues

There are no known issues in this release.
The NVIDIA container image of TensorFlow, release 17.06, is available.

TensorFlow container image version 17.06 is based on TensorFlow 1.1.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/local/[bin,lib] directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.21
- NVIDIA® Collective Communications Library™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Ubuntu 16.04 with May 2017 updates

Known Issues

The inception_v4 model, with a batch size of 64 per GPU, and with large input images or resolution (for example, 480 pixels on the shortest side), are seen to run out of memory. To work around this in TensorFlow 17.06, reduce the resolution or reduce the batch size to allow the model to fit.
The NVIDIA container image of TensorFlow, release 17.05, is available.

TensorFlow container image version 17.05 is based on TensorFlow 1.0.1.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed into the `/usr/local/[bin,lib]` directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.21
- NVIDIA® Collective Communications Library™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Latest cuDNN release
- Ubuntu 16.04 with April 2017 updates

Known Issues

The `inception_v4` model, with a batch size of 64 per GPU, and with large input images or resolution (for example, 480 pixels on the shortest side), are seen to run out of
memory. To work around this in TensorFlow 17.05, reduce the resolution or reduce the batch size to allow the model to fit.
Chapter 38.
TENSORFLOW RELEASE 17.04

The NVIDIA container image of TensorFlow, release 17.04, is available. TensorFlow container image version 17.04 is based on TensorFlow 1.0.1.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/[local]/[bin,lib] directories in the container image.

To achieve optimum TensorFlow performance, for image based training, the container includes a sample script that demonstrates efficient training of convolutional neural networks (CNNs). The sample script may need to be modified to fit your application.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.20
- NVIDIA® Collective Communications Library™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- 2x improvement on 8GPUs; 1.5X on 4 GPUs
- Updated NCCL integration with support for NVLink
- Multi-GPU CNN examples that demonstrates efficient training of CNNs using NCCL
- XLA (Accelerated Linear Algebra) support enabled, allowing users to offload operations to TensorFlow experimental XLA back-end
- Ubuntu 16.04 with March 2017 updates
Known Issues

There are no known issues in this release.
Chapter 39.
TENSORFLOW RELEASE 17.03

The NVIDIA container image of TensorFlow, release 17.03, is available. TensorFlow container image version 17.03 is based on TensorFlow 1.0.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/local/[bin,lib] directories in the container image.

The container also includes the following:

- Ubuntu 16.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.20
- NVIDIA® Collective Communications Library™ (NCCL) 1.6.1

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Multi-GPU BigLSTM example that trains a recurrent neural network (RNN) to learn a language model
- Ubuntu 16.04 with February 2017 updates

Known Issues

There are no known issues in this release.
Chapter 40.
TENSORFLOW RELEASE 17.02

The NVIDIA container image of TensorFlow, release 17.02, is available.

TensorFlow container image version 17.02 is based on TensorFlow 0.12.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed into the `/usr/local/[bin,lib]` directories in the container image.

The container also includes the following:

- Ubuntu 14.04
- NVIDIA CUDA® 8.0.61
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.13
- NVIDIA® Collective Communications Library™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Fused image color adjustment kernels for improved preprocessing performance
- Ubuntu 14.04 with January 2017 updates

Known Issues

There are no known issues in this release.
Chapter 41.
TENSORFLOW RELEASE 17.01

The NVIDIA container image of TensorFlow, release 17.01, is available.

TensorFlow container image version 17.01 is based on TensorFlow 0.12.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in /opt/tensorflow. It is pre-built and installed into the /usr/local/[bin,lib] directories in the container image.

The container also includes the following:

- Ubuntu 14.04
- NVIDIA CUDA® 8.0.54
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.10
- NVIDIA® Collective Communications Library™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Ubuntu 14.04 with December 2016 updates

Known Issues

There are no known issues in this release.
The NVIDIA container image of TensorFlow, release 16.12, is available. TensorFlow container image version 16.12 is based on TensorFlow 0.12.0.

Contents of TensorFlow

This container image contains the complete source of the version of NVIDIA TensorFlow in `/opt/tensorflow`. It is pre-built and installed into the `/usr/local/[bin,lib]` directories in the container image.

The container also includes the following:

- Ubuntu 14.04
- NVIDIA CUDA® 8.0.54
- NVIDIA CUDA® Deep Neural Network library™ (cuDNN) 6.0.5
- NVIDIA® Collective Communications Library™ (NCCL) 1.6.1 (optimized for NVLink™)

Key Features and Enhancements

This TensorFlow release includes the following key features and enhancements.

- Supports multi-GPU training
  - [BETA] NCCL integration for improved multi-GPU scaling
  
  Requires explicit use by the model script.

- Supports recurrent neural networks
  - Support for cuDNN recurrent neural networks (RNN) layers
  
  Requires explicit use by the model script.
- Better I/O throughput via `libjpeg-turbo`, fast iDCT decoding
- Support for the non-fused Winograd algorithm for improved convolution performance.
- TensorBoard; a data visualization toolkit
- Several built-in TensorFlow examples
- Ubuntu 14.04 with November 2016 updates

**Known Issues**

There are no known issues in this release.
Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION
REFERENCED IN THIS GUIDE IS PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED,
STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY
DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A
PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever,
NVIDIA’s aggregate and cumulative liability towards customer for the product described in this guide shall
be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED,
MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE,
AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A
SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE
INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER
LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS
FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR
IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for
any specified use without further testing or modification. Testing of all parameters of each product is not
necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and
fit for the application planned by customer and to do the necessary testing for the application in order
to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect
the quality and reliability of the NVIDIA product and may result in additional or different conditions and/
or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any
default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA
product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license,
either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information
in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without
alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLProf,
Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NvCaffe, PerfWorks, Pascal, SDK Manager,
Tegra, TensorRT, TensorRT Inference Server, Triton Inference Server, Tesla, TF-TRT, and Volta are trademarks
and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and
product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2020 NVIDIA Corporation. All rights reserved.

www.nvidia.com