



TENSORRT

SWE-SWDOCTR-001-RELN_vTensorRT 5.0.4 | November 2018

Release Notes



TABLE OF CONTENTS

Chapter 1. TensorRT Overview	1
Chapter 2. TensorRT Release 5.x.x	2
2.1. TensorRT Release 5.0.4.....	2
2.2. TensorRT Release 5.0.3.....	4
2.3. TensorRT Release 5.0.2.....	5
2.4. TensorRT Release 5.0.1 Release Candidate (RC).....	12
2.5. TensorRT Release 5.0.0 Release Candidate (RC).....	16
Chapter 3. TensorRT Release 4.x.x	21
3.1. TensorRT Release 4.0.1.....	21
3.2. TensorRT Release 4.0 Release Candidate (RC) 2.....	24
3.3. TensorRT Release 4.0 Release Candidate (RC).....	25
Chapter 4. TensorRT Release 3.x.x	29
4.1. TensorRT Release 3.0.4.....	29
4.2. TensorRT Release 3.0.2.....	29
4.3. TensorRT Release 3.0.1.....	31
4.4. TensorRT Release 3.0 Release Candidate (RC).....	36
4.5. TensorRT Release 3.0 Early Access (EA).....	39
Chapter 5. TensorRT Release 2.x.x	42
5.1. TensorRT Release 2.1.....	42

Chapter 1.

TENSORRT OVERVIEW

The core of NVIDIA TensorRT is a C++ library that facilitates high performance inference on NVIDIA graphics processing units (GPUs). TensorRT takes a trained network, which consists of a network definition and a set of trained parameters, and produces a highly optimized runtime engine which performs inference for that network.

You can describe a TensorRT network using either a C++ or Python API, or you can import an existing Caffe, ONNX, or TensorFlow model using one of the provided parsers.

TensorRT provides API's via C++ and Python that help to express deep learning models via the Network Definition API or load a pre-defined model via the parsers that allows TensorRT to optimize and run them on a NVIDIA GPU. TensorRT applies graph optimizations, layer fusion, among other optimizations, while also finding the fastest implementation of that model leveraging a diverse collection of highly optimized kernels. TensorRT also supplies a runtime that you can use to execute this network on all of NVIDIA's GPU's from the Kepler generation onwards.

TensorRT also includes optional high speed mixed precision capabilities introduced in the Tegra X1, and extended with the Pascal, Volta, and Turing architectures.

Chapter 2.

TENSORRT RELEASE 5.X.X

2.1. TensorRT Release 5.0.4

This is the TensorRT 5.0.4 release notes for Windows users. This release includes fixes from the previous TensorRT 5.0.x releases as well as the following additional fixes. For previous TensorRT 5.0.x release notes, see [TensorRT Release Notes](#).

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements for the Windows platform.

- ▶ ONNX model parsing support has been added.
- ▶ Two new samples showcasing ONNX model parsing functionality have been added:
 - ▶ `sampleOnnxMNIST`
 - ▶ `sampleINT8API`
- ▶ CUDA 9.0 support has been added.

Compatibility

- ▶ TensorRT 5.0.4 supports Windows 10
- ▶ TensorRT 5.0.4 supports CUDA 10.0 and CUDA 9.0
- ▶ TensorRT 5.0.4 supports CUDNN 7.3.1
- ▶ TensorRT 5.0.4 supports Visual Studio 2017

Limitations In 5.0.4

- ▶ TensorRT 5.0.4 does not support Python API on Windows.

Known Issues

- ▶ NVIDIA's Windows display driver sets timeout detection recovery to 2 seconds by default. This can cause some timeouts within TensorRT's builder and cause crashes. For more information, see [Timeout Detection & Recovery \(TDR\)](#) to increase the default timeout threshold if you encounter this problem.
- ▶ TensorRT Windows performance is slower than Linux due to the operating system and driver differences. There are two driver modes:
 - ▶ WDDM (around 15% slower than Linux)
 - ▶ TCC (around 10% slower than Linux.) TCC mode is generally not supported for GeForce GPUs, however, we recommend it for Quadro or Tesla GPUs. Detailed instructions on setting TCC mode can be found here: [Tesla Compute Cluster \(TCC\)](#).
 - ▶ Volta FP16 performance on CUDA 9.0 may be up to 2x slower than on CUDA 10.0. We expect to mitigate this issue in a future release.
 - ▶ Most README files that are included with the samples assume that you are working on a Linux workstation. If you are using Windows and do not have access to a Linux system with an NVIDIA GPU, then you can try using [VirtualBox](#) to create a virtual machine based on [Ubuntu](#). You may also want to consider using a [Docker](#) container for Ubuntu. Many samples do not require any training, therefore the CPU versions of TensorFlow and PyTorch are enough to complete the samples.
 - ▶ For `sample_ssd` and `sample_uff_ssd`, the INT8 calibration script is not supported natively on Windows. You can generate the INT8 batches on a Linux machine and copy them over in order to run `sample_ssd` in INT8 mode.
 - ▶ For `sample_uff_ssd`, the Python script `convert-to-uff` is not packaged within the .zip. You can generate the required `.uff` file on a Linux machine and copy it over in order to run `sample_uff_ssd`. During INT8 calibration, you may encounter a file reading error in `TensorRT/data/samples/ssd/VOC2007/list.txt`. This is due to line-ending differences on Windows. To work around this, open `list.txt` in a text editor and ensure that the file is using Unix-style line endings.
 - ▶ For `sample_int8_api`, the `legacy` runtime option is not supported on Windows.
 - ▶ When issuing `-h` for `sampleINT8API`, the `--write_tensors` option is missing. The `--write_tensors` option generates a file that contains a list of network tensor names. By default, it writes to the `network_tensors.txt` file. For information about additional options, issue `--tensors`.

2.2. TensorRT Release 5.0.3

This is the TensorRT 5.0.3 release notes for Automotive and L4T users. This release includes fixes from the previous TensorRT 5.0.x releases as well as the following additional fixes. For previous TensorRT 5.0.x release notes, see [TensorRT Release Notes](#).

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- ▶ For this TensorRT release, JetPack L4T and Drive D5L are supported by a single package.

See the [TensorRT Developer Guide](#) for details.

Compatibility

TensorRT 5.0.3 supports the following product versions:

- ▶ CUDA 10.0
- ▶ cuDNN 7.3.1
- ▶ NvMedia DLA version 2.2
- ▶ NvMedia VPI Version 2.3

Known Issues

- ▶ For multi-process execution, and specifically when executing multiple inference sessions in parallel (for example, of `trtexec`) target different accelerators, you may observe a performance degradation if `cudaEventBlockingSync` is used for stream synchronization.

One way to work around this performance degradation is to use the `cudaEventDefault` flag when creating the events which internally uses the spin-wait synchronization mechanism. In `trtexec`, the default behavior is to use blocking events, but this can be overridden with the `--useSpinWait` option to specify spin-wait based synchronization.



The spin-wait mechanism can increase CPU utilization on the system.

For more information about CUDA blocking sync semantics, refer to [Event Management](#).

- ▶ There is a known issue when attempting to cross compile samples for mobile platforms on an `x86_64` host machine. As cross-platform CUDA packages

are structured differently, the following changes are required for `samples/Makefile.config` when compiling cross platform.

Line 80

Add:

```
-L"${CUDA_INSTALL_DIR}/targets/${TRIPLE}/${CUDA_LIBDIR}/stubs"
```

Line 109

Remove:

```
-lnvToolsExt
```

2.3. TensorRT Release 5.0.2

This is the TensorRT 5.0.2 release notes for Desktop users. This release includes fixes from the previous TensorRT 5.0.x releases as well as the following additional fixes. For previous TensorRT 5.0.x release notes, see [TensorRT Release Notes](#).

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

Platforms

Added support for CentOS 7.5, Ubuntu 18.04, and Windows 10.

Turing

You must use CUDA 10.0 or later if you are using a Turing GPU.

DLA (Deep Learning Accelerator)

The layers supported by DLA are Activation, Concatenation, Convolution, Deconvolution, ElementWise, FullyConnected, LRN, Pooling, and Scale. For layer specific constraints, see [DLA Supported Layers](#). AlexNet, GoogleNet, ResNet-50, and LeNet for MNIST networks have been validated on DLA. Since DLA support is new to this release, it is possible that other CNN networks that have not been validated will not work. Report any failing CNN networks that satisfy the layer constraints by submitting a bug via the [NVIDIA Developer](#) website. Ensure you log-in, click on your name in the upper right corner, click **My account** > **My Bugs** and select **Submit a New Bug**.

The `trtexec` tool can be used to run on DLA with the `--useDLACore=N` where `N` is 0 or 1, and `--fp16` options. To run the MNIST network on DLA using `trtexec`, issue:

```
./trtexec --deploy=data/mnist/mnist.prototxt --output=prob --useDLACore=0 --fp16 --allowGPUIFallback
```

`trtexec` does not support ONNX models on DLA.

Redesigned Python API

The Python API has gone through a thorough redesign to bring the API up to modern Python standards. This fixed multiple issues, including making it possible to support serialization via the Python API. Python samples using the new API include parser samples for ResNet-50, a Network API sample for MNIST, a plugin sample using Caffe, and an end-to-end sample using TensorFlow.

INT8

Support has been added for user-defined INT8 scales, using the new `ITensor::setDynamicRange` function. This makes it possible to define dynamic range for INT8 tensors without the need for a calibration data set. `setDynamicRange` currently supports only symmetric quantization. A user must either supply a dynamic range for each tensor or use the calibrator interface to take advantage of INT8 support.

Plugin Registry

A new searchable plugin registry, `IPluginRegistry`, is a single registration point for all plugins in an application and is used to find plugin implementations during deserialization.

C++ Samples

sampleSSD

This sample demonstrates how to perform inference on the Caffe SSD network in TensorRT, use TensorRT plugins to speed up inference, and perform INT8 calibration on an SSD network. To generate the required `prototxt` file for this sample, perform the following steps:

1. Download `models_VGGNet_VOC0712_SSD_300x300.tar.gz` from: https://drive.google.com/file/d/0BzKzrI_SkD1_WVVTsmQxU0dVRzA/view
2. Extract the contents of the tar file;

```
tar xvf
~/Downloads/models_VGGNet_VOC0712_SSD_300x300.tar.gz
```

3. Edit the `deploy.prototxt` file and change all the `Flatten` layers to `Reshape` operations with the following parameters:

```
reshape_param {
  shape {
    dim: 0
    dim: -1
    dim: 1
    dim: 1
  }
}
```

4. Update the `detection_out` layer by adding the `keep_count` output, for example, add:

```
top: "keep_count"
```

5. Rename the `deploy.prototxt` file to `ssd.prototxt` and run the sample.

- To run the sample in INT8 mode, install Pillow first by issuing the `$ pip install Pillow` command, then follow the instructions from the README.

sampleINT8API

This sample demonstrates how to perform INT8 Inference using per-tensor dynamic range. To generate the required input data files for this sample, perform the following steps:

Running the sample:

- Download the [Model files](#) from GitHub, for example:

```
wget https://s3.amazonaws.com/download.onnx/models/opset_3/resnet50.tar.gz
```

- Unzip the tar file:

```
tar -xvzf resnet50.tar.gz
```

- Rename `resnet50/model.onnx` to `resnet50/resnet50.onnx`, then copy the `resnet50.onnx` file to the `data/int8_api` directory.
- Run the sample:

```
./sample_int8_api [-v or --verbose]
```

Running the sample with a custom configuration:

- Download the [Model files](#) from GitHub.
- Create an input image with a PPM extension. Resize it with the dimensions of 224x224x3.
- Create a file called `reference_labels.txt`. Ensure each line corresponds to a single imagenet label. You can download the imagenet 1000 class human readable labels from [here](#). The reference label file contains only a single label name per line, for example, 0: 'tench, Tinca tinca' is represented as `tench`.
- Create a file called `dynamic_ranges.txt`. Ensure each line corresponds to the tensor name and floating point dynamic range, for example `<tensor_name> : <float dynamic range>`. In order to generate tensor names, iterate over the network and generate the tensor names. The dynamic range can either be obtained from training (by measuring the min/max value of activation tensors in each epoch) or using custom post processing techniques (similar to TensorRT calibration). You can also choose to use a dummy per tensor dynamic range to run the sample.

Python Samples

yolov3_onnx

This sample demonstrates a full ONNX-based pipeline for inference with the network [YOLOv3-608](#), including pre- and post-processing.

uff_ssd

This sample demonstrates a full UFF-based inference pipeline for performing inference with an SSD (InceptionV2 feature extractor) network.

IPluginV2

A plugin class `IPluginV2` has been added together with a corresponding `IPluginV2` layer. The `IPluginV2` class includes similar methods to `IPlugin` and `IPluginExt`, so if your plugin implemented `IPluginExt` previously, you will change the class name to `IPluginV2`. The `IPlugin` and `IPluginExt` interfaces are to be deprecated in the future, therefore, moving to the `IPluginV2` interface for this release is strongly recommended.

See the [TensorRT Developer Guide](#) for details.

Breaking API Changes

- ▶ The choice of which DLA core to run a layer on is now made at runtime. You can select the device type at build time, using the following methods:

```
IBuilder::setDeviceType(ILayer* layer, DeviceType deviceType)
IBuilder::setDefaultDeviceType(DeviceType deviceType)

where DeviceType is:
{
    kGPU,    //!< GPU Device
    kDLA,    //!< DLA Core
};
```

The specific DLA core to execute the engine on can be set by the following methods:

```
IBuilder::setDLACore(int dlaCore)
IRuntime::setDLACore(int dlaCore)
```

The following methods have been added to get the DLA core set on `IBuilder` or `IRuntime` objects:

```
int IBuilder::getDLACore()
int IRuntime::getDLACore()
```

Another API has been added to query the number of accessible DLA cores as follows:

```
int IBuilder::getNbDLACores()
int IRuntime::getNbDLACores()
```

- ▶ The `--useDLA=<int>` on `trtexec` tool has been changed to `--useDLACore=<int>`, the value can range from 0 to `N-1`, `N` being the number of DLA cores. Similarly, to run any sample on DLA, use `--useDLACore=<int>` instead of `--useDLA=<int>`.

Compatibility

- ▶ TensorRT 5.0.2 has been tested with cuDNN 7.3.1.
- ▶ TensorRT 5.0.2 has been tested with TensorFlow 1.9.
- ▶ This TensorRT release supports CUDA 10.0 and CUDA 9.0. CUDA 8.0 and CUDA 9.2 are no longer supported. On Windows only, CUDA 10.0 is supported for TensorRT 5.0.1 RC.

Limitations In 5.0.2

- ▶ TensorRT 5.0.2 does not include support for DLA with the INT8 data type. Only DLA with the FP16 data type is supported by TensorRT at this time. DLA with INT8 support is planned for a future TensorRT release.
- ▶ Android is not supported in TensorRT 5.0.2.
- ▶ The Python API is only supported on x86-based Linux platforms.
- ▶ The `create*Plugin` functions in the `NvInferPlugin.h` file do not have Python bindings.
- ▶ ONNX models are not supported on DLA in TensorRT 5.0.2.
- ▶ The included `resnet_v1_152`, `resnet_v1_50`, `lenet5`, and `vgg19` UFF files do not support FP16 mode. This is because some of the weights fall outside the range of FP16.
- ▶ The ONNX parser is not supported on Windows 10. This includes all samples which depend on the ONNX parser. ONNX support will be added in a future release.
- ▶ Tensor Cores supporting INT4 were first introduced with Turing GPUs. This release of TensorRT 5.0 does not support INT4.
- ▶ The `yolov3_onnx` Python sample is not supported on Ubuntu 14.04 and earlier.
- ▶ The `uff_ssd` sample requires `tensorflow-gpu` for performing validation only. Other parts of the sample can use the CPU version of `tensorflow`.
- ▶ The Leaky ReLU plugin (`LReLU_TRT`) allows for only a parameterized slope on a per tensor basis.

Deprecated Features

The following features are deprecated in TensorRT 5.0.2:

- ▶ The majority of the old Python API, including the Lite and Utils API, are deprecated. It is currently still accessible in the `tensorrt.legacy` package, but will be removed in a future release.

- ▶ The following Python examples are deprecated:
 - ▶ `caffe_to_trt`
 - ▶ `pytorch_to_trt`
 - ▶ `tf_to_trt`
 - ▶ `onnx_mnist`
 - ▶ `uff_mnist`
 - ▶ `mnist_api`
 - ▶ `sample_onnx`
 - ▶ `googlenet`
 - ▶ `custom_layers`
 - ▶ `lite_examples`
 - ▶ `resnet_as_a_service`
- ▶ The `detectionOutput` Plugin has been renamed to the **NMS** Plugin.
- ▶ The old ONNX parser will no longer be packaged with TensorRT; instead, use the open-source ONNX parser.
- ▶ The `DimensionTypes` class is deprecated.
- ▶ The plugin APIs that return `INvPlugin` are being deprecated and they now return `IPluginV2`. These APIs will be removed in a future release. Refer to `NvInferPlugin.h` inside the TensorRT package.
- ▶ The `nvinfer1::IPluginFactory`, `nvuffparser1::IPluginFactory`, and `nvuffparser1::IPluginFactoryExt` plugins are still available for backward compatibility. However, it is still recommended to use the Plugin Registry and implement `IPluginCreator` for all new plugins.
- ▶ The `libnvinfer.a`, `libnvinfer_plugin.a`, and `libnvparsers.a` libraries have been renamed to `libnvinfer_static.a`, `libnvinfer_plugin_static.a`, and `libnvparsers_static.a` respectively. This makes TensorRT consistent with CUDA, cuDNN, and other NVIDIA software libraries. It also avoids some ambiguity between dynamic and static libraries during linking.

Known Issues

- ▶ Only AlexNet, GoogleNet, ResNet-50, and MNIST are known to work with DLA. Other networks may work, but they have not been extensively tested.
- ▶ For this TensorRT release, there are separate JetPack L4T and Drive D5L packages due to differences in the DLA library dependencies. In a future release, this should become unified.
- ▶ The static library `libnvparsers_static.a` requires a special build of `protobuf` to complete static linking. Due to filename conflicts with the official `protobuf`

packages, these additional libraries are only included in the tar file at this time. The two additional libraries that you will need to link against are `libprotobuf.a` and `libprotobuf-lite.a` from the tar file.

- ▶ The ONNX static libraries `libnnonxparser_static.a` and `libnnonxparser_runtime_static.a` require static libraries that are missing from the package in order to complete static linking. The two static libraries that are required to complete linking are `libonnx_proto.a` and `libnnonxparser_plugin.a`, as well as the protobuf libraries mentioned earlier. You will need to build these two missing static libraries from the [open source ONNX project](#). This issue will be resolved in a future release.
- ▶ The C++ API documentation is not included in the TensorRT zip file. Refer to the online documentation if you want to view the [TensorRT C++ API](#).
- ▶ Most README files that are included with the samples assume that you are working on a Linux workstation. If you are using Windows and do not have access to a Linux system with an NVIDIA GPU, then you can try using [VirtualBox](#) to create a virtual machine based on [Ubuntu](#). Many samples do not require any training, therefore the CPU versions of TensorFlow and PyTorch are enough to complete the samples.
- ▶ The [TensorRT Developer Guide](#) has been written with Linux users in mind. Windows specific instructions, where possible, will be added in a future revision of the document.
- ▶ If you have multiple networks with different layer device type configurations, `IBuilder::reset(nvinfer1::INetworkDefinition& network)` will reset the layer device types for all the networks in that builder.
- ▶ If `sampleMovieLensMPS` crashes before completing execution, an artifact (`/dev/shm/sem.engine_built`) will not be properly destroyed. If the sample complains about being unable to create a semaphore, remove the artifact by running `rm /dev/shm/sem.engine_built`.
- ▶ To create a valid UFF file for `sampleMovieLensMPS`, the correct command is:


```
python convert_to_uff.py sampleMovieLens.pb -p preprocess.py
```

 where `preprocess.py` is a script that is shipped with `sampleMovieLens`. Do not use the command specified by the README.
- ▶ The `trtexec` tool does not currently validate command-line arguments. If you encounter failures, double check the command-line parameters that you provided.

2.4. TensorRT Release 5.0.1 Release Candidate (RC)

This is the release candidate (RC) for TensorRT 5.0.1 release notes. This release is for Windows users only. It includes several enhancements and improvements compared to the previously released TensorRT 4.0.1. This preview release is for early testing and feedback, therefore, for production use of TensorRT, continue to use [TensorRT 4.0.1](#).

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

Platforms

Added support for CentOS 7.5, Ubuntu 18.04, and Windows 10.

Turing

You must use CUDA 10.0 or later if you are using a Turing GPU.

DLA (Deep Learning Accelerator)

The layers supported by DLA are Activation, Concatenation, Convolution, Deconvolution, ElementWise, FullyConnected, LRN, Pooling, and Scale. For layer specific constraints, see [DLA Supported Layers](#). Networks such as AlexNet, GoogleNet, ResNet-50, and MNIST work with DLA. Other CNN networks may work, but they have not been extensively tested and may result in failures including segfaults.

The `trtexec` tool can be used to run on DLA with the `--useDLA=N` and `--fp16` options. To run the AlexNet network on DLA using `trtexec`, issue:

```
./trtexec --deploy=data/AlexNet/AlexNet_N2.prototxt --output=prob --useDLA=1
--fp16 --allowGPUFallback
```

`trtexec` does not support ONNX models to run on DLA.

Redesigned Python API

The Python API has been rewritten from scratch and includes various improvements. In addition to several bug fixes, it is now possible to serialize and deserialize an engine to and from a file using the Python API. Python samples using the new API include parser samples for ResNet-50, a Network API sample for MNIST, a plugin sample using Caffe, and an end-to-end sample using TensorFlow.

INT8

Support for user-defined INT8 scales, using the new `ITensor::setDynamicRange` function. This makes it possible to provide custom INT8 calibration without the need for a calibration data set. `setDynamicRange` currently supports only symmetric

quantization. Furthermore, if no calibration table is provided, calibration scales must be provided for each layer.

Plugin Registry

A new searchable plugin registry, **IPluginRegistry**, that is a single registration point for all plugins in an application and is used to find plugin implementations during deserialization.

sampleSSD

This sample demonstrates how to preprocess the input to the SSD network, perform inference on the SSD network in TensorRT, use TensorRT plugins to speed up inference, and perform INT8 calibration on an SSD network.

See the [TensorRT Developer Guide](#) for details.

Breaking API Changes

- ▶ The **IPluginExt** API has 4 new methods, **getPluginType**, **getPluginVersion**, **destroy** and **clone**. All plugins of type **IPluginExt** will have to implement these new methods and re-compile. This is a temporary issue; we expect to restore compatibility with the 4.0 API in the GA release. For more information, see [Migrating Plugins From TensorRT 5.0.0 RC To TensorRT 5.0.x](#) for guidance on migration.

Compatibility

- ▶ TensorRT 5.0.1 RC has been tested with cuDNN 7.3.0.
- ▶ TensorRT 5.0.1 RC has been tested with TensorFlow 1.9.
- ▶ TensorRT 5.0.1 RC for Windows has been tested with Visual Studio 2017.
- ▶ This TensorRT release supports CUDA 10.0 and CUDA 9.0. CUDA 8.0 and CUDA 9.2 are no longer supported. On Windows only, CUDA 10.0 is supported for TensorRT 5.0.1 RC.

Limitations In 5.0.1 RC

- ▶ For this release, there are separate JetPack L4T and Drive D5L packages due to differences in the DLA library dependencies. In a future release, this should become unified.
- ▶ Android is not supported in TensorRT 5.0.1 RC.
- ▶ The Python API does not support DLA.
- ▶ The **create*Plugin** functions in the **NvInferPlugin.h** file do not have Python bindings.

- ▶ The choice of which DLA device to run on is currently made at build time. In GA, it will be selectable at runtime.
- ▶ ONNX models are not supported on DLA in TensorRT 5.0.1 RC.
- ▶ The included `resnet_v1_152`, `resnet_v1_50`, `lenet5`, and `vgg19` UFF files do not support FP16 mode. This is because some of the weights fall outside the range of FP16.
- ▶ Python is not supported on Windows 10. This includes the `graphsurgeon` and UFF Python modules.
- ▶ The ONNX parser is not supported on Windows 10. This includes all samples which depend on the ONNX parser. ONNX support will be added in a future release.

Deprecated Features

The following features are deprecated in TensorRT 5.0.1 RC:

- ▶ Majority of the old Python API, including the Lite and Utils API, is deprecated. It is currently still accessible in the `tensorrt.legacy` package, but will be removed in a future release.
- ▶ The following Python examples:
 - ▶ `caffe_to_trt`
 - ▶ `pytorch_to_trt`
 - ▶ `tf_to_trt`
 - ▶ `onnx_mnist`
 - ▶ `uff_mnist`
 - ▶ `mnist_api`
 - ▶ `sample_onnx`
 - ▶ `googlenet`
 - ▶ `custom_layers`
 - ▶ `lite_examples`
 - ▶ `resnet_as_a_service`
- ▶ The `detectionOutput` Plugin has been renamed to the `NMS` Plugin.
- ▶ The old ONNX parser will no longer be packaged with TensorRT; instead, use the open-source ONNX parser.
- ▶ The `DimensionTypes` class.
- ▶ The plugin APIs that return `IPlugin` are being deprecated and they now return `IPluginExt`. These APIs will be removed in a future release. Refer to the `NvInferPlugin.h` file inside the package.

- ▶ `nvinfer1::IPluginFactory`, `nvuffparser1::IPluginFactory`, and `nvuffparser1::IPluginFactoryExt` (still available for backward compatibility). Instead, use the Plugin Registry and implement `IPluginCreator` for all new plugins.
- ▶ `libnvinfer.a`, `libnvinfer_plugin.a`, and `libnvparsers.a` have been renamed to `libnvinfer_static.a`, `libnvinfer_plugin_static.a`, and `libnvparsers_static.a` respectively. This makes TensorRT consistent with CUDA, cuDNN, and other NVIDIA software libraries. It also avoids some ambiguity between dynamic and static libraries during linking.

Known Issues

- ▶ The Plugin Registry will only register plugins with a unique `{name, version}` tuple. The API for this is likely to change in future versions to support multiple plugins with same name and version.
- ▶ Only AlexNet, GoogleNet, ResNet-50, and MNIST are known to work with DLA. Other networks may work, but they have not been extensively tested.
- ▶ The static library `libnvparsers_static.a` requires a special build of `protobuf` to complete static linking. Due to filename conflicts with the official `protobuf` packages, these additional libraries are only included in the tar file at this time. The two additional libraries that you will need to link against are `libprotobuf.a` and `libprotobuf-lite.a` from the tar file.
- ▶ The ONNX static libraries `libnnonnxparser_static.a` and `libnnonnxparser_runtime_static.a` require static libraries that are missing from the package in order to complete static linking. The two static libraries that are required to complete linking are `libonnx_proto.a` and `libnnonnxparser_plugin.a`, as well as the `protobuf` libraries mentioned earlier. You will need to build these two missing static libraries from the [open source ONNX project](#). This issue will be resolved in a future release.
- ▶ If you upgrade only `uff-converter-tf`, for example using `apt-get install uff-converter-tf`, then it will not upgrade `graphsurgeon-tf` due to inexact dependencies between these two packages. You will need to specify both packages on the command line, such as `apt-get install uff-converter-tf graphsurgeon-tf` in order to upgrade both packages. This will be fixed in a future release.
- ▶ The `fc_plugin_caffe_mnist` python sample cannot be executed if the sample is built using `pybind11 v2.2.4`. We suggest that you instead clone `pybind11 v2.2.3` using the following command:

```
git clone -b v2.2.3 https://github.com/pybind/pybind11.git
```

- ▶ The C++ API documentation is not included in the TensorRT zip file. Refer to the online documentation if you want to view the [TensorRT C++ API](#).
- ▶ Most README files that are included with the samples assume that you are working on a Linux workstation. If you are using Windows and do not have access to a Linux system with an NVIDIA GPU, then you can try using [VirtualBox](#) to create a virtual machine based on [Ubuntu](#). Many samples do not require any training, therefore the CPU versions of TensorFlow and PyTorch are enough to complete the samples.
- ▶ The [TensorRT Developer Guide](#) has been written with Linux users in mind. Windows specific instructions, where possible, will be added in a future revision of the document.

2.5. TensorRT Release 5.0.0 Release Candidate (RC)

This is the release candidate (RC) for TensorRT 5.0.0. It includes several enhancements and improvements compared to the previously released TensorRT 4.0.1. This preview release is for early testing and feedback, therefore, for production use of TensorRT, continue to use [TensorRT 4.0.1](#).

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

Platforms

Added support for CentOS 7.5 and Ubuntu 18.04.

Turing

You must use CUDA 10.0 or later if you are using a Turing GPU.

DLA (Deep Learning Accelerator)

The layers supported by DLA are Activation, Concatenation, Convolution, Deconvolution, ElementWise, FullyConnected, LRN, Pooling, and Scale. For layer specific constraints, see [DLA Supported Layers](#). Networks such as AlexNet, GoogleNet, ResNet-50, and MNIST work with DLA. Other CNN networks may work, but they have not been extensively tested and may result in failures including segfaults.

The `trtexec` tool can be used to run on DLA with the `--useDLA=N` and `--fp16` options. To run the AlexNet network on DLA using `trtexec`, issue:

```
./trtexec --deploy=data/AlexNet/AlexNet_N2.prototxt --output=prob --useDLA=1
--fp16 --allowGPUFallback
```

`trtexec` does not support ONNX models to run on DLA.

Redesigned Python API

The Python API has been rewritten from scratch and includes various improvements. In addition to several bug fixes, it is now possible to serialize and deserialize an engine to and from a file using the Python API. Python samples using the new API include parser samples for ResNet-50, a Network API sample for MNIST, a plugin sample using Caffe, and an end-to-end sample using TensorFlow.

INT8

Support for user-defined INT8 scales, using the new `ITensor::setDynamicRange` function. This makes it possible to provide custom INT8 calibration without the need for a calibration data set. `setDynamicRange` currently supports only symmetric quantization. Furthermore, if no calibration table is provided, calibration scales must be provided for each layer.

Plugin Registry

A new searchable plugin registry, `IPluginRegistry`, that is a single registration point for all plugins in an application and is used to find plugin implementations during deserialization.

See the [TensorRT Developer Guide](#) for details.

Breaking API Changes

- ▶ The `IPluginExt` API has 4 new methods, `getPluginType`, `getPluginVersion`, `destroy` and `clone`. All plugins of type `IPluginExt` will have to implement these new methods and re-compile. This is a temporary issue; we expect to restore compatibility with the 4.0 API in the GA release. For more information, see [Migrating Plugins From TensorRT 4.0.x To TensorRT 5.0 RC](#) for guidance on migration.
- ▶ Upcoming changes in TensorRT 5.0 GA for plugins
 - ▶ A new plugin class `IPluginV2` and a corresponding `IPluginV2` layer will be introduced. The `IPluginV2` class includes similar methods to `IPlugin` and `IPluginExt`, so if your plugin implemented `IPluginExt` previously, you will change the class name to `IPluginV2`.
 - ▶ The `IPluginCreator` class will create and deserialize plugins of type `IPluginV2` as opposed to `IPluginExt`.
 - ▶ The `create*Plugin()` methods in `NvInferPlugin.h` will return plugin objects of type `IPluginV2` as opposed to `IPluginExt`.

Compatibility

- ▶ TensorRT 5.0.0 RC has been tested with cuDNN 7.3.0.
- ▶ TensorRT 5.0.0 RC has been tested with TensorFlow 1.9.
- ▶ This TensorRT release supports CUDA 10.0 and CUDA 9.0. CUDA 8.0 and CUDA 9.2 are no longer supported.

Limitations In 5.0.0 RC

- ▶ For this release, there are separate JetPack L4T and Drive D5L packages due to differences in the DLA library dependencies. In a future release, this should become unified.
- ▶ Android is not supported in TensorRT 5.0.0 RC.
- ▶ The Python API does not support DLA.
- ▶ The `create*Plugin` functions in the `NvInferPlugin.h` file do not have Python bindings.
- ▶ The choice of which DLA device to run on is currently made at build time. In GA, it will be selectable at runtime.
- ▶ ONNX models are not supported on DLA in TensorRT 5.0 RC.
- ▶ The included `resnet_v1_152`, `resnet_v1_50`, `lenet5`, and `vgg19` UFF files do not support FP16 mode. This is because some of the weights fall outside the range of FP16.

Deprecated Features

The following features are deprecated in TensorRT 5.0.0:

- ▶ Majority of the old Python API, including the Lite and Utils API, is deprecated. It is currently still accessible in the `tensorrt.legacy` package, but will be removed in a future release.
- ▶ The following Python examples:
 - ▶ `caffe_to_trt`
 - ▶ `pytorch_to_trt`
 - ▶ `tf_to_trt`
 - ▶ `onnx_mnist`
 - ▶ `uff_mnist`
 - ▶ `mnist_api`
 - ▶ `sample_onnx`
 - ▶ `googlenet`

- ▶ `custom_layers`
- ▶ `lite_examples`
- ▶ `resnet_as_a_service`
- ▶ The **detectionOutput** Plugin has been renamed to the **NMS** Plugin.
- ▶ The old ONNX parser will no longer be packaged with TensorRT; instead, use the open-source ONNX parser.
- ▶ The **DimensionTypes** class.
- ▶ The plugin APIs that return **IPlugin** are being deprecated and they now return **IPluginExt**. These APIs will be removed in a future release. Refer to the **NvInferPlugin.h** file inside the package.
- ▶ **nvinfer1::IPluginFactory**, **nvuffparser1::IPluginFactory**, and **nvuffparser1::IPluginFactoryExt** (still available for backward compatibility). Instead, use the Plugin Registry and implement **IPluginCreator** for all new plugins.
- ▶ **libnvinfer.a**, **libnvinfer_plugin.a**, and **libnvparsers.a** have been renamed to **libnvinfer_static.a**, **libnvinfer_plugin_static.a**, and **libnvparsers_static.a** respectively. This makes TensorRT consistent with CUDA, cuDNN, and other NVIDIA software libraries. It also avoids some ambiguity between dynamic and static libraries during linking.

Known Issues

- ▶ The Plugin Registry will only register plugins with a unique **{name, version}** tuple. The API for this is likely to change in future versions to support multiple plugins with same name and version.
- ▶ Only AlexNet, GoogleNet, ResNet-50, and MNIST are known to work with DLA. Other networks may work, but they have not been extensively tested.
- ▶ The static library **libnvparsers_static.a** requires a special build of **protobuf** to complete static linking. Due to filename conflicts with the official **protobuf** packages, these additional libraries are only included in the tar file at this time. The two additional libraries that you will need to link against are **libprotobuf.a** and **libprotobuf-lite.a** from the tar file.
- ▶ The ONNX static libraries **libnvonnxparser_static.a** and **libnvonnxparser_runtime_static.a** require static libraries that are missing from the package in order to complete static linking. The two static libraries that are required to complete linking are **libonnx_proto.a** and **libnvonnxparser_plugin.a**, as well as the **protobuf** libraries mentioned earlier.

You will need to build these two missing static libraries from the [open source ONNX project](#). This issue will be resolved in a future release.

- ▶ If you upgrade only `uff-converter-tf`, for example using `apt-get install uff-converter-tf`, then it will not upgrade `graphsurgeon-tf` due to inexact dependencies between these two packages. You will need to specify both packages on the command line, such as `apt-get install uff-converter-tf graphsurgeon-tf` in order to upgrade both packages. This will be fixed in a future release.
- ▶ The `fc_plugin_caffe_mnist` python sample cannot be executed if the sample is built using pybind11 v2.2.4. We suggest that you instead clone pybind11 v2.2.3 using the following command:

```
git clone -b v2.2.3 https://github.com/pybind/pybind11.git
```

Chapter 3.

TENSORRT RELEASE 4.X.X

3.1. TensorRT Release 4.0.1

This TensorRT 4.0.1 General Availability release includes several enhancements and improvements compared to the previously released TensorRT 3.0.4.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- ▶ TensorRT 4.0.1 GA has been tested with cuDNN 7.1.3 and now requires cuDNN 7.1.x.
- ▶ Support for ONNX 1.0 (Open Neural Network Exchange) has been implemented. ONNX is a standard for representing deep learning models that enable models to be transferred between frameworks. TensorRT can now parse the network definitions in ONNX format, in addition to NVCaffe and UFF formats.
- ▶ The Custom Layer API now supports user-defined layers that take half precision, or FP16, inputs and return FP16 outputs.
- ▶ Added support for the MatrixMultiply, Constant, Gather, Ragged SoftMax, Reduce, RNNv2 and TopK layers (for K up to 25).
- ▶ This release has optimizations which target recommender systems like Neural Collaborative Filtering.
- ▶ Many layers now support the ability to broadcast across the batch dimension.
- ▶ In TensorRT 3.0, INT8 had issues with rounding and striding in the Activation layer. This may have caused INT8 accuracy to be low. Those issues have been fixed.
- ▶ The C++ samples and Python examples were tested with TensorFlow 1.8 and PyTorch 0.4.0 where applicable.

- ▶ Added `sampleOnnxMNIST`. This sample shows the conversion of an MNIST network in ONNX format to a TensorRT network.
- ▶ Added `sampleNMT`. Neural Machine Translation (NMT) using sequence to sequence (seq2seq) models has garnered a lot of attention and is used in various NMT frameworks. `sampleNMT` is a highly modular sample for inferencing using C++ and TensorRT API so that you can consider using it as a reference point in your projects.
- ▶ Updated `sampleCharRNN` to use RNNv2 and converting weights from TensorFlow to TensorRT.
- ▶ Added `sampleUffSSD`. This sample converts the TensorFlow Single Shot MultiBox Detector (SSD) network to a UFF format and runs it on TensorRT using plugins. This sample also demonstrates how other TensorFlow networks can be preprocessed and converted to UFF format with support of custom plugin nodes.
- ▶ Memory management improvements (see the Memory Management section in the Developer Guide for details.)
 - ▶ Applications may now provide their own memory for activations and workspace during inference, which is used only while the pipeline is running.
 - ▶ An allocator callback is available for all memory allocated on the GPU. In addition, model deserialization is significantly faster (from system memory, up to 10x faster on large models).

Using TensorRT 4.0.1

Ensure you are familiar with the following notes when using this release.

- ▶ The builder methods `setHalf2Mode` and `getHalf2Mode` have been superseded by `setFp16Mode` and `getFp16Mode` which better represent their intended usage.
- ▶ The sample utility `giexec` has been renamed to `trtexec` to be consistent with the product name, TensorRT, which is often shortened to TRT. A compatibility script for users of `giexec` has been included to help users make the transition.

Deprecated Features

- ▶ The RNN layer type is deprecated in favor of RNNv2, however, it is still available for backwards compatibility.
- ▶ Legacy GIE version defines in `NvInfer.h` have been removed. They were `NV_GIE_MAJOR`, `NV_GIE_MINOR`, `NV_GIE_PATCH`, and `NV_GIE_VERSION`. The correct alternatives are `NV_TENSORRT_MAJOR`, `NV_TENSORRT_MINOR`, `NV_TENSORRT_PATCH`, and `NV_TENSORRT_VERSION` which existed in TensorRT 3.0.4 as well.
- ▶ Dimension types are now ignored in the API, however, they are still available for backwards compatibility.

Known Issues

- ▶ If the ONNX parser included with TensorRT is unable to parse your model, then try updating to the latest [open source ONNX parser](#), which may resolve your issue.
- ▶ PyTorch no longer supports Python 3.4 with their current release (0.4.0). Therefore, the TensorRT PyTorch examples will not work when using Python 3 on Ubuntu 14.04.
- ▶ Reshape to a tensor that has a larger number of dimensions than the input tensor is not supported.
- ▶ Reformat has a known memory overwrite issue on Volta when FP16 is used with the Concatenation layer and the Reformat layer.
- ▶ If you have two different CUDA versions of TensorRT installed, such as CUDA 8.0 and CUDA 9.0, or CUDA 9.2 using local repos, then you will need to execute an additional command to install the CUDA 8.0 version of TensorRT and prevent it from upgrading to the CUDA 9.0 or CUDA 9.2 versions of TensorRT.

```
sudo apt-get install libnvinfer4=4.1.2-1+cuda8.0 \
  libnvinfer-dev=4.1.2-1+cuda8.0
sudo apt-mark hold libnvinfer4 libnvinfer-dev
```

- ▶ sampleNMT
 - ▶ Performance is not fully optimized
- ▶ sampleUffSSD
 - ▶ Some precision loss was observed while running the network in INT8 mode, causing some objects to go undetected in the image. Our general observation is that having at least 500 images for calibration is a good starting point.
- ▶ Performance regressions
 - ▶ Compared to earlier TensorRT versions, a 5% slowdown was observed on AlexNet when running on GP102 devices with batch size 2 using the NvCaffeParser.
 - ▶ Compared to earlier TensorRT versions, a 5% to 10% slowdown was observed on variants of inception and some instances of ResNet when using the NvUffParser.
- ▶ The NvUffParser returns the output tensor in the shape specified by the user, and not in NCHW shape as in earlier versions of TensorRT. In other words, the output tensor shape will match the shape of the tensor returned by TensorFlow, for the same network.
- ▶ The Python 3.4 documentation is missing from the Ubuntu 14.04 packages. Refer to the Python 2.7 documentation or view the online Python documentation as an alternative.

- ▶ Some samples do not provide a `-h` argument to print the sample usage. You can refer to the `README.txt` file in the sample directory for usage examples. Also, if the data files for some samples cannot be found it will sometimes raise an exception and abort instead of exiting normally.
- ▶ If you have more than one version of the CUDA toolkit installed on your system and the CUDA version for TensorRT is not the latest version of the CUDA toolkit, then you will need to provide an additional argument when compiling the samples. For example, you have CUDA 9.0 and CUDA 9.2 installed and you are using TensorRT for CUDA 9.0.

```
make CUDA_INSTALL_DIR=/usr/local/cuda-9.0
```

- ▶ When you `pip uninstall tensorrtplugins` Python package, you may see the following error which can be ignored.

```
OSError: [Errno 2] No such file or directory: '/usr/local/lib/python2.7/dist-packages/tensorrtplugins-4.0.1.0-py2.7-linux-x86_64.egg'
```

- ▶ Due to a bug in cuDNN 7.1.3, which is the version of cuDNN TensorRT has been validated against, using RNNs with half precision on Kepler GPUs will cause TensorRT to abort. FP16 support is non-native on Kepler GPUs, therefore, using any precision other than FP32 is discouraged except for testing.
- ▶ `sampleMovieLens` is currently limited to running a maximum of 8 concurrent processes on a Titan V and may result in suboptimal engines during parallel execution. The sample will be enhanced in the near future to support a greater degree of concurrency. Additionally, to ensure compatibility with TensorRT, use TensorFlow $\leq 1.7.0$ to train the model. There may be a conflict between the versions of CUDA and/or cuDNN used by TensorRT and TensorFlow 1.7. We suggest that you install TensorFlow 1.7 CPU in order to complete the sample.

```
python -m pip install tensorflow==1.7.0
```

3.2. TensorRT Release 4.0 Release Candidate (RC) 2

This TensorRT 4.0 Release Candidate (RC) 2 includes several enhancements and improvements compared to the previously released TensorRT 3.0.4. TensorRT 4.0 RC2 supports desktop and Tegra platforms. This release candidate is for early testing and feedback, for production use of TensorRT, continue to use 3.0.4.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- ▶ TensorRT 4.0 RC2 for mobile supports cuDNN 7.1.2.
- ▶ TensorRT 4.0 RC2 for desktop supports cuDNN 7.1.3.
- ▶ Support for ONNX 1.0 (Open Neural Network Exchange) has been implemented. TensorRT can now parse the network definitions in ONNX format, in addition to NVCaffe and UFF formats.
- ▶ The Custom Layer API now supports user-defined layers that take half precision, or FP16, inputs and return FP16 tensors.
- ▶ Added support for the MatrixMultiply, Constant, Gather, Ragged SoftMax, Reduce, RNNv2 and TopK layers (for K up to 25).
- ▶ Added SampleONNXMNIST sample. Open Neural Network Exchange (ONNX) is a standard for representing deep learning models that enable models to be transferred between frameworks. This sample shows the conversion of an MNIST network in ONNX format to a TensorRT network.

Deprecated Features

- ▶ The RNN layer type is deprecated in favor of RNNv2, however, it is still available for backwards compatibility.
- ▶ Legacy GIE version defines in `NvInfer.h` have been removed. They were `NV_GIE_MAJOR`, `NV_GIE_MINOR`, `NV_GIE_PATCH`, and `NV_GIE_VERSION`. The correct alternatives are `NV_TENSORRT_MAJOR`, `NV_TENSORRT_MINOR`, `NV_TENSORRT_PATCH`, and `NV_TENSORRT_VERSION` which existed in TensorRT 3.0.4 as well.
- ▶ Dimension Types are now ignored in the API, however, they are still available for backwards compatibility.

Known Issues

SampleMLP and SampleNMT are included in this release, however, they are beta samples. They are currently not optimized for mobile platforms.

3.3. TensorRT Release 4.0 Release Candidate (RC)

This TensorRT 4.0 Release Candidate (RC) includes several enhancements and improvements compared to the previously released TensorRT 3.0.4. TensorRT 4.0 RC supports x86 desktop platforms only. This release candidate is for early testing and feedback, for production use of TensorRT, continue to use 3.0.4.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- ▶ Support for ONNX 1.0 (Open Neural Network Exchange) has been implemented. TensorRT can now parse the network definitions in ONNX format, in addition to NVCaffe and UFF formats.
- ▶ The Custom Layer API now supports user-defined layers that take half precision, or FP16, inputs and return FP16 tensors.
- ▶ Added support for the MatrixMultiply, Constant, Gather, Ragged SoftMax, Reduce, RNNv2 and TopK layers (for K up to 25).
- ▶ The samples were tested with TensorFlow 1.6. You must be using cuDNN 7.0.x in order to use both TensorRT and TensorFlow at the same time since TensorFlow 1.6 does not support cuDNN 7.1.x yet.
- ▶ Added SampleMLP sample for multi-layer perceptrons.
- ▶ Added SampleONNXMNIST sample. Open Neural Network Exchange (ONNX) is a standard for representing deep learning models that enable models to be transferred between frameworks. This sample shows the conversion of an MNIST network in ONNX format to a TensorRT network.
- ▶ Added SampleNMT sample. Neural Machine Translation (NMT) using sequence to sequence (seq2seq) models has garnered a lot of attention and is used in various NMT frameworks. SampleNMT is a highly modular sample for inferencing using C++ and TensorRT API so that you can consider using it as a reference point in your projects.
- ▶ Updated SampleCharRNN sample to use RNNv2 and converting weights from TensorFlow to TensorRT.

Deprecated Features

- ▶ The RNN layer type is deprecated in favor of RNNv2, however, it is still available for backwards compatibility.
- ▶ Legacy GIE version defines in `NvInfer.h` have been removed. They were `NV_GIE_MAJOR`, `NV_GIE_MINOR`, `NV_GIE_PATCH`, and `NV_GIE_VERSION`. The correct alternatives are `NV_TENSORRT_MAJOR`, `NV_TENSORRT_MINOR`, `NV_TENSORRT_PATCH`, and `NV_TENSORRT_VERSION` which existed in TensorRT 3.0.4 as well.
- ▶ Dimension Types are now ignored in the API, however, they are still available for backwards compatibility.

Known Issues

- ▶ If you were previously using the machine learning debian repository, then it will conflict with the version of `libcudnn7` that is contained within the local repository for TensorRT. The following commands will downgrade `libcudnn7` to version 7.0.5.15, which is supported and tested with TensorRT, and hold the package at

this version. If you are using CUDA 8.0 for your application, ensure you replace `cuda9.0` with `cuda8.0`.

```
sudo apt-get install libcudnn7=7.0.5.15-1+cuda9.0 libcudnn7-
dev=7.0.5.15-1+cuda9.0
sudo apt-mark hold libcudnn7 libcudnn7-dev
```

If you would like to later upgrade `libcudnn7` to the latest version, then you can use the following commands to remove the hold.

```
sudo apt-mark unhold libcudnn7 libcudnn7-dev
sudo apt-get dist-upgrade
```

- ▶ If you have both the CUDA 8.0 and CUDA 9.0 local repos installed for TensorRT, then you will need to execute an additional command to install the CUDA 8.0 version of TensorRT and prevent it from upgrading to the CUDA 9.0 version of TensorRT.

```
sudo apt-get install libnvinfer4=4.1.0-1+cuda8.0 libnvinfer-
dev=4.1.0-1+cuda8.0
sudo apt-mark hold libnvinfer4 libnvinfer-dev
```

- ▶ If you installed the dependencies for the TensorRT python examples using `pip install tensorrt[examples]` then it could replace the GPU accelerated version of TensorFlow with the CPU accelerated version of TensorFlow. You will need to remove the version of TensorFlow installed as a TensorRT dependency and install the GPU accelerated version in its place.

```
pip uninstall tensorflow
pip install tensorflow-gpu
```

- ▶ **SampleNMT**
 - ▶ Performance is not fully optimized
 - ▶ SampleNMT does not support FP16
 - ▶ The vocabulary files are expected to be in the `../../../../data/samples/nmt/deen` directory from the executable. The sample doesn't print usage if vocabulary files are not present in the above mentioned path. For more information, see the `README.txt` file for usage details.
- ▶ **SampleMLP**
 - ▶ Performance is not fully optimized
 - ▶ SampleMLP does not support FP16
 - ▶ The accuracy of MLPs for handwritten digit recognition is lower than CNNs, therefore, the sample may give an incorrect prediction in some cases.
 - ▶ SampleMLP usage has incorrect details on the `-a` parameter. It should be `-a <#>`. The activation to use on the layers, defaults to `1`. Valid values are `1 [ReLU]`, `2 [Sigmoid]`, and `3 [TanH]`; instead of `-a <#>`. The activation to use in on the layers, defaults to `1`. Valid values are `0 [ReLU]`, `1 [Sigmoid]`, and `2 [TanH]`.
 - ▶ The timing information printed by the sample may not be accurate.

- ▶ Performance regressions
 - ▶ A 5% slowdown was observed on AlexNet when running on GP102 devices with batch size 2 using the Caffe parser.
 - ▶ A 5% to 10% slowdown was observed on variants of inception, some instances of ResNet, and some instances of SSD when using the UFF parser.

Chapter 4.

TENSORRT RELEASE 3.X.X

4.1. TensorRT Release 3.0.4

This TensorRT 3.0.4 General Availability release is a minor release and includes some improvements and fixes compared to the previously released TensorRT 3.0.2.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- ▶ Fixed an issue with INT8 deconvolution bias. If you have seen an issue with deconvolution INT8 accuracy especially regarding TensorRT. 2.1, then this fix should solve the issue.
- ▶ Fixed an accuracy issue in FP16 mode for NVCaffe models.

Using TensorRT 3.0.4

Ensure you are familiar with the following notes when using this release.

- ▶ The UFF converter script is packaged only for x86 users. If you are not an x86 user, and you want to convert TensorFlow models into UFF, you need to obtain the conversion script from the x86 package of TensorRT.

4.2. TensorRT Release 3.0.2

This TensorRT 3.0.2 General Availability release is a minor release and includes some improvements and fixes compared to the previously released TensorRT 3.0.1.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- ▶ Fixed a bug in one of the INT8 deconvolution kernels that was generating incorrect results. This fixed accuracy regression from 2.1 for networks that use deconvolutions.
- ▶ Fixed a bug where the builder would report out-of-memory when compiling a low precision network, in the case that a low-precision version of the kernel could not be found. The builder now correctly falls back to a higher precision version of the kernel.
- ▶ Fixed a bug where the existence of some low-precision kernels were being incorrectly reported to the builder.

Using TensorRT 3.0.2

Ensure you are familiar with the following notes when using this release.

- ▶ When working with large networks and large batch sizes on the Jetson TX1 you may see failures that are the result of CUDA error 4. This error generally means a CUDA kernel failed to execute properly, but sometimes this can mean the CUDA kernel actually timed out. The CPU and GPU share memory on the Jetson TX1 and reducing the memory used by the CPU would help the situation. If you are not using the graphical display on L4T you can stop the X11 server to free up CPU and GPU memory. This can be done using:

```
$ sudo systemctl stop lightdm.service
```

Known Issues

- ▶ INT8 deconvolutions with biases have the bias scaled incorrectly. U-Net based segmentation networks typically have non-zero bias.
- ▶ For TensorRT Android 32-bit, if your memory usage is high, then you may see TensorRT failures. The issue is related to the CUDA allocated buffer address being higher or equal to 0x80000000 and it is hard to know the exact memory usage after which this issue is hit.
- ▶ If you are installing TensorRT from a tar package (instead of using the .deb packages and `apt-get`), you will need to update the `custom_plugins` example to point to the location that the tar package was installed into. For example, in the `<PYTHON_INSTALL_PATH>/tensorrt/examples/custom_layers/tensorrtplugins/setup.py` file change the following:
 - ▶ Change `TENSORRT_INC_DIR` to point to the `<TAR_INSTALL_ROOT>/include` directory.
 - ▶ Change `TENSORRT_LIB_DIR` to point to `<TAR_INSTALL_ROOT>/lib` directory.
- ▶ If you were previously using the machine learning debian repository, then it will conflict with the version of `libcudnn7` that is contained within the local repository for TensorRT. The following commands will downgrad `libcudnn7` to the CUDA 9.0 version, which is supported by TensorRT, and hold the package at this version.


```
sudo apt-get install libcudnn7=7.0.5.15-1+cuda9.0
libcudnn7-dev=7.0.5.15-1+cuda9.0
sudo apt-mark hold libcudnn7 libcudnn7-dev
```

If you would like to later upgrade **libcudnn7** to the latest version, then you can use the following commands to remove the hold.

```
sudo apt-mark unhold libcudnn7 libcudnn7-dev
sudo apt-get dist-upgrade
```

4.3. TensorRT Release 3.0.1

This TensorRT 3.0.1 General Availability release includes several enhancements and improvements compared to the previously released TensorRT 2.1.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

NvCaffeParser

NVCaffe 0.16 is now supported.

New deep learning layers or algorithms

- ▶ The TensorRT deconvolution layer previously did not support non-zero padding, or stride values that were distinct from kernel size. These restrictions have now been lifted.
- ▶ The TensorRT deconvolution layer now supports groups.
- ▶ Non-determinism in the deconvolution layer implementation has been eliminated.
- ▶ The TensorRT convolution layer API now supports dilated convolutions.
- ▶ The TensorRT API now supports these new layers (but they are not supported via the NvCaffeParser):
 - ▶ unary
 - ▶ shuffle
 - ▶ padding
- ▶ The Elementwise (eltwise) layer now supports broadcasting of input dimensions.
- ▶ The Flatten layer flattens the input while maintaining the `batch_size`. This layer was added in the UFF converter and NvUffParser.
- ▶ The Squeeze layer removes dimensions of size 1 from the shape of a tensor. This layer was added in the UFF converter and NvUffParser.

Universal Framework Format 0.2

UFF format is designed to encapsulate trained neural networks so that they can be parsed by TensorRT. It's also designed in a way of storing the information about a

neural network that is needed to create an inference engine based on that neural network.

Performance

- ▶ Performance regressions seen from v2.1 to 3.0.1 Release Candidate for INT8 and FP16 are now fixed.
 - ▶ The INT8 regression in LRN that impacted networks like GoogleNet and AlexNet is now fixed.
 - ▶ The FP16 regression that impacted networks like AlexNet and ResNet-50 is now fixed.
- ▶ The performance of the Xception network has improved, for example, by more than 3 times when batch size is 8 on Tesla P4.
- ▶ Changed how the CPU synchronizes with the GPU in order to reduce the overall load on the CPU when running inference with TensorRT.
- ▶ The deconvolution layer implementation included with TensorRT was, in some circumstances, using significantly more memory and had lower performance than the implementation provided by the cuDNN library. This has now been fixed.
- ▶ **MAX_TENSOR_SIZE** changed from $(1 \ll 30)$ to $((1 \ll 31) - 1)$. This change enables the user to run larger batch sizes for networks with large input images.

Samples

- ▶ All Python examples now import TensorRT after the appropriate framework is imported. For example, the `tf_to_trt.py` example imports TensorFlow before importing TensorRT. This is done to avoid cuDNN version conflict issues.
- ▶ The `tf_to_trt` and `pytorch_to_trt` samples shipped with the TensorRT 3.0 Release Candidate included network models that were improperly trained with the MNIST dataset, resulting in poor classification accuracy. This version has new models that have been properly trained with the MNIST dataset to provide better classification accuracy.
- ▶ The `pytorch_to_trt` sample originally showed low accuracy with MNIST, however, data and training parameters were modified to address this.
- ▶ The `giexec` command line wrapper in earlier versions would fail if users specify workspace ≥ 2048 MB. This issue is now fixed.

Functionality

The **AverageCountExcludesPadding** attribute has been added to the pooling layer to control whether to use inclusive or exclusive averaging. The default is **true**, as used by most frameworks. The NvCaffeParser sets this to **false**, restoring compatibility of padded average pooling between NVCaffe and TensorRT.

TensorRT Python API

TensorRT 3.0.1 introduces the TensorRT Python API, which provides developers interfaces to:

- ▶ the NvCaffeParser
- ▶ the NvUffParser
- ▶ The nvinfer graph definition API
- ▶ the inference engine builder
- ▶ the engine executor
- ▶ the perform calibration for running inference with INT8
- ▶ a workflow to include C++ custom layer implementations

TensorRT Lite: A simplified API for inference

TensorRT 3.0.1 provides a streamlined set of API functions (`tensorrt_lite`) that allow users to export a trained model, build an engine, and run inference, with only a few lines of Python code.

Streamlined export of models trained in TensorFlow into TensorRT

With this release, you can take a trained model in TensorFlow saved in a TensorFlow protobuf and convert it to run in TensorRT. The TensorFlow model exporter creates an output file in a format called UFF (Universal Framework Format), which can then be parsed by TensorRT.

Currently the export path is expected to support the following:

- ▶ TensorFlow 1.3
- ▶ FP32 CNNs
- ▶ FP16 CNNs

The TensorFlow export path is currently not expected to support the following:

- ▶ Other versions of TensorFlow (0.9, 1.1, etc.)
- ▶ RNNs
- ▶ INT8 CNNs

Volta

The NVIDIA Volta architecture is now supported, including the Tesla V100 GPU. On Volta devices, the Tensor Core feature provides a large performance improvement, and Tensor Cores are automatically used when the builder is set to `half2mode`.

QNX

TensorRT 3.0.1 runs on the QNX operating system on the Drive PX2 platform.

Release Notes 3.0.1 Errata

- ▶ Due to the cuDNN symbol conflict issues between TensorRT and TensorFlow, the `tf_to_trt` Python example works with TensorFlow 1.4.0 only and not prior versions of TensorFlow.
- ▶ If your system has multiple `libcudnnX-dev` versions installed, ensure that cuDNN 7 is used for compiling and running TensorRT samples. This problem can occur when you have TensorRT and a framework installed. TensorRT uses cuDNN 7 while most frameworks are currently on cuDNN 6.
- ▶ There are various details in the *Release Notes* and *Developer Guide* about the `pytorch_to_trt` Python example. This sample is no longer part of the package because of cuDNN symbol conflict issues between PyTorch and TensorRT.
- ▶ In the *Installation and Setup* section of the *Release Notes*, it is mentioned that `TENSORRT_LIB_DIR` should point to `<TAR_INSTALL_ROOT>/lib64`. Instead, `TENSORRT_LIB_DIR` should point to `<TAR_INSTALL_ROOT>/lib`.
- ▶ There are some known minor performance regressions for FP32 mode on K80 for large batch sizes on CUDA 8. Update to CUDA 9 if you see similar performance regression.

Using TensorRT 3.0.1

Ensure you are familiar with the following notes when using this release.

- ▶ Although networks can use NHWC and NCHW, TensorFlow users are encouraged to convert their networks to use NCHW data ordering explicitly in order to achieve the best possible performance.
- ▶ The `libnvcaffe_parsers.so` library file is now called `libnvparsers.so`. The links for `libnvcaffe_parsers` are updated to point to the new `libnvparsers` library. The static library `libnvcaffe_parser.a` is also linked to the new `libnvparsers`.

Known Issues

Installation and Setup

- ▶ If you are installing TensorRT from a tar package (instead of using the .deb packages and `apt-get`), you will need to update the `custom_plugins` example to point to the location that the tar package was installed into. For example, in the `<PYTHON_INSTALL_PATH>/tensorrt/examples/custom_layers/tensorrtplugins/setup.py` file change the following:
 - ▶ Change `TENSORRT_INC_DIR` to point to the `<TAR_INSTALL_ROOT>/include` directory.
 - ▶ Change `TENSORRT_LIB_DIR` to point to `<TAR_INSTALL_ROOT>/lib64` directory.

- ▶ The PyTorch based sample will not work with the CUDA 9 Toolkit. It will only work with the CUDA 8 Toolkit.
- ▶ When using the TensorRT APIs from Python, import the `tensorflow` and `uff` modules before importing the `tensorrt` module. This is required to avoid a potential namespace conflict with the `protobuf` library as well as the cuDNN version. In a future update, the modules will be fixed to allow the loading of these Python modules to be in an arbitrary order.
- ▶ The TensorRT Python APIs are only supported on x86 based systems. Some installation packages for ARM based systems may contain Python `.whl` files. Do not install these on the ARM systems, as they will not function.
- ▶ The TensorRT product version is incremented from 2.1 to 3.0.1 because we added major new functionality to the product. The `libnvinfer` package version number was incremented from 3.0.2 to 4.0 because we made non-backward compatible changes to the application programming interface.
- ▶ The TensorRT debian package name was simplified in this release to `tensorrt`. In previous releases, the product version was used as a suffix, for example `tensorrt-2.1.2`.
- ▶ If you have trouble installing the TensorRT Python modules on Ubuntu 14.04, refer to the steps on installing `swig` to resolve the issue. For installation instructions, see [Unix Installation](#).
- ▶ The Flatten layer can only be placed in front of the Fully Connected layer. This means that the Flatten layer can only be used if its output is directly fed to a Fully Connected layer.
- ▶ The Squeeze layer only implements the binary squeeze (removing specific size 1 dimensions). The batch dimension cannot be removed.
- ▶ If you see the `Numpy.core.multiarray failed to import` error message, upgrade your NumPy to version 1.13.0 or greater.
- ▶ For Ubuntu 14.04, use pip version `>= 9.0.1` to get all the dependencies installed.

TensorFlow Model Conversion

- ▶ The TensorFlow to TensorRT model export works only when running TensorFlow with GPU support enabled. The converter does **not** work if TensorFlow is running without GPU acceleration.
- ▶ The TensorFlow to TensorRT model export does **not** work with network models specified using the TensorFlow Slim interface, nor does it work with models specified using the Keras interface.
- ▶ The TensorFlow to TensorRT model export does **not** support recurrent neural network (RNN) models.
- ▶ The TensorFlow to TensorRT model export may produce a model that has extra tensor reformatting layers compared to a model generated directly using the C++ or Python TensorRT graph builder API. This may cause the model that originated from

TensorFlow to run slower than the model constructed directly with the TensorRT APIs.

- ▶ Although TensorFlow models can use either NHWC or NCHW tensor layouts, TensorFlow users are encouraged to convert their models to use the NCHW tensor layout explicitly, in order to achieve the best possible performance when exporting the model to TensorRT.
- ▶ The TensorFlow parser requires that input will be fed to the network in NCHW format.

Other known issues

- ▶ On the V100 GPU, running models with INT8 only works if the batch size is evenly divisible by 4.
- ▶ TensorRT Python interface requires NumPy 1.13.0 while the installing TensorRT using `pip` may only install 1.11.0. Use `sudo pip install numpy -U` to update if the NumPy version on the user machine is not 1.13.0.

4.4. TensorRT Release 3.0 Release Candidate (RC)

This is the second preview release of TensorRT. For production use of TensorRT, continue to use 2.1.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

Volta

The NVIDIA Volta architecture is now supported, including the Tesla V100 GPU. On Volta devices, the Tensor Core feature provides a large performance improvement, and Tensor Cores are automatically used when the builder is set to `half2mode`.

Streamlined export of models trained in TensorFlow into TensorRT

With this release you can take a trained model in TensorFlow saved in a TensorFlow protobuf and convert it to run in TensorRT. The TensorFlow model exporter creates an output file in a format called UFF (Universal Framework Format), which can then be parsed by TensorRT.

Currently the export path is expected to support the following:

- ▶ Tensorflow 1.3
- ▶ FP32 CNNs
- ▶ FP16 CNNs

The TensorFlow export path is currently not expected to support the following:

- ▶ Other versions of TensorFlow (0.9, 1.1, etc.)

- ▶ RNNs
- ▶ INT8 CNNs

TensorFlow convenience functions

NVIDIA provides convenience functions so that when using UFF and TensorRT to export a model and run inference, only a few lines of code is needed.

Universal Framework Format 0.1

UFF format is designed to encapsulate trained neural networks so they can be parsed by TensorRT.

Python API

TensorRT 3.0 introduces the TensorRT Python API, which provides developers interfaces to:

- ▶ the NvCaffeParser
- ▶ the NvUffParser
- ▶ The nvinfer graph definition API
- ▶ the inference engine builder
- ▶ the engine executor

TensorRT also introduces a workflow to include C++ custom layer implementations in Python based TensorRT applications.

New deep learning layers or algorithms

- ▶ The TensorRT deconvolution layer previously did not support non-zero padding, or stride values that were distinct from kernel size. These restrictions have now been lifted.
- ▶ The TensorRT deconvolution layer now supports groups.
- ▶ Non-determinism in the deconvolution layer implementation has been eliminated.
- ▶ The TensorRT convolution layer API now supports dilated convolutions.
- ▶ The TensorRT API now supports these new layers (but they are not supported via the NvCaffeParser):
 - ▶ unary
 - ▶ shuffle
 - ▶ padding
- ▶ The Elementwise (eltwise) layer now supports broadcasting of input dimensions.

QNX

TensorRT 3.0 runs on the QNX operating system on the Drive PX2 platform.

Known Issues

Installation and Setup

- ▶ If you are installing TensorRT from a tar package (instead of using the .deb packages and `apt-get`), then the `custom_plugins` example will need to be updated to point to the location that the tar package was installed to. For example, in the `<PYTHON_INSTALL_PATH>/tensorrt/examples/custom_layers/tensorrtplugins/setup.py` file change the following:
 - ▶ Change `TENSORRT_INC_DIR` to point to the `<TAR_INSTALL_ROOT>/include` directory.
 - ▶ Change `TENSORRT_LIB_DIR` to point to the `<TAR_INSTALL_ROOT>/lib` directory.
- ▶ The PyTorch based sample will not work with the CUDA 9 Toolkit. It will only work with the CUDA 8 Toolkit.
- ▶ When using the TensorRT APIs from Python, import the `tensorflow` and `uff` modules before importing the `tensorrt` module. This is required to avoid a potential namespace conflict with the `protobuf` library. In a future update, the modules will be fixed to allow the loading of these Python modules to be in an arbitrary order.
- ▶ The TensorRT Python APIs are only supported on x86 based systems. Some installation packages for ARM based systems may contain Python `.whl` files. Do not install these on the ARM systems, as they will not function.
- ▶ The TensorRT product version is incremented from 2.1 to 3.0 because we added major new functionality to the product. The `libnvinfer` package version number was incremented from 3.0.2 to 4.0 because we made non-backward compatible changes to the application programming interface.
- ▶ The TensorRT debian package name was simplified in this release to `tensorrt`. In previous releases, the product version was used as a suffix, for example `tensorrt-2.1.2`.
- ▶ If you have trouble installing the TensorRT Python modules on Ubuntu 14.04, refer to the steps on installing `swig` to resolve the issue. For installation instructions, see [Unix Installation](#).
- ▶ There is a performance regression in the LRN layer when the network is running in INT8 mode. It impacts networks like GoogleNet and AlexNet but not ResNet-50, VGG-19 etc.

TensorFlow Model Conversion

- ▶ The TensorFlow to TensorRT model export works only when running TensorFlow with GPU support enabled. The converter does **not** work if TensorFlow is running without GPU acceleration.

- ▶ The TensorFlow to TensorRT model export does **not** work with network models specified using the TensorFlow Slim interface, nor does it work with models specified using the Keras interface.
- ▶ The TensorFlow to TensorRT model export does not support recurrent neural network (RNN) models.
- ▶ The TensorFlow to TensorRT model export does not support convolutional layers that have asymmetric padding (a different number of zero-padded rows and columns).
- ▶ The TensorFlow to TensorRT model export may produce a model that has extra tensor reformatting layers compared to a model generated directly using the C++ or Python TensorRT graph builder API. This may cause the model that originated from TensorFlow to run slower than the model constructed directly with the TensorRT APIs.
- ▶ Although TensorFlow models can use either NHWC or NCHW tensor layouts, TensorFlow users are encouraged to convert their models to use the NCHW tensor layout explicitly, in order to achieve the best possible performance.

Other known issues

- ▶ The Inception v4 network models are not supported with this Release Candidate with FP16 on V100.
- ▶ On V100, running models with INT8 do not work if the batch size is not divisible by 4.
- ▶ The Average Pooling behavior has changed to exclude padding from the computation, which is how all other Pooling modes handle padding. This results in incorrect behavior for network models which rely on Average Pooling and which include padding, such as Inception v3. This issue will be addressed in a future release.
- ▶ In this Release Candidate, the arguments for the `tensorrt_exec.py` script are slightly different than the ones for the `giexec` executable, and can be a source of confusion for users. Consult the documentation carefully to avoid unexpected errors. The command-line arguments will be changed to match `giexec` in a future update.
- ▶ The INT8 Calibration feature is not available in the TensorRT Python APIs.
- ▶ The `examples/custom_layer` sample will not work on Ubuntu 14.04 x86_64 systems, however, it does work properly on Ubuntu 16.04 systems. This will be fixed in the next update of the software.

4.5. TensorRT Release 3.0 Early Access (EA)

This is a preview release of TensorRT. For production use of TensorRT, continue to use 2.1.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

Streamlined export for models trained in TensorFlow to TensorRT

With this release you can take a TensorFlow trained model saved in a TensorFlow protobuf and convert it to run in TensorRT. The TensorFlow to UFF converter creates an output file in a format called UFF (Universal Framework Format) which can then be read into TensorRT.

Currently the export path is expected to support the following:

- ▶ Tensorflow 1.0
- ▶ FP32 CNNs
- ▶ FP16 CNNs

The TensorFlow export path is currently not expected to support the following:

- ▶ Other versions of TensorFlow (0.9, 1.1, etc..)
- ▶ RNNs
- ▶ INT8 CNNs

TensorFlow convenience functions

NVIDIA provides convenience functions so that when using UFF and TensorRT to export a model and run inference, only a few lines of code is needed.

Universal Framework Format 0.1

UFF format is designed as a way of storing the information about a neural network that is needed to create an inference engine based on that neural network.

Python API

TensorRT 3.0 introduces the TensorRT Python API, allowing developers to access:

- ▶ the NvCaffeParser
- ▶ the NvUffParser
- ▶ The nvinfer graph definition API
- ▶ the inference engine builder
- ▶ the inference-time interface for engine execution within Python

TensorRT also introduces a workflow to include C++ custom layer implementations in Python based TensorRT applications.

Using TensorRT 3.0

Ensure you are familiar with the following notes when using this release.

- ▶ Although networks can use NHWC and NCHW, TensorFlow users are encouraged to convert their networks to use NCHW data ordering explicitly in order to achieve the best possible performance.

- ▶ Average pooling behavior changed to exclude the padding from the computation. The padding is now excluded from the computation in all of the pooling modes. This results in incorrect behavior for networks which rely on average pooling which includes padding, such as `inceptionV3`. This issue will be addressed in a future release.
- ▶ The `libnvcaffe_parsers.so` library file is now called `libnvparsers.so`. The links for `libnvcaffe_parsers` are updated to point to the new `libnvparsers` library. The static library `libnvcaffe_parser.a` is also linked to the new `libnvparsers`. For example:
 - ▶ Old structure: `libnvcaffe_parsers.4.0.0.so` links to `libnvcaffe_parsers.4.so` which links to `libnvcaffe_parsers.so`.
 - ▶ New structure: `libnvcaffe_parsers.4.0.0.so` links to `libnvcaffe_parsers.4.so` which links to `libnvcaffe_parsers.so` which links to `libnvparsers.so` (actual file).

Known Issues

- ▶ TensorRT does not support asymmetric padding.
- ▶ Some TensorRT optimizations disabled just for this Early Release (EA) to ensure that the UFF model runs properly. This will be addressed in TensorRT 3.0.
- ▶ The TensorFlow conversion path is not fully optimized.
- ▶ INT8 Calibration is not available in Python.
- ▶ Deconvolution is not implemented in the UFF workflow.

Chapter 5.

TENSORRT RELEASE 2.X.X

5.1. TensorRT Release 2.1

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

Custom Layer API

If you want TensorRT to use novel, unique or proprietary layers in the evaluation of certain networks, the Custom Layer API lets you provide a CUDA kernel function that implements the functionality you want.

Installers

You have two ways you can install TensorRT 2.1:

1. Ubuntu deb packages. If you have root access and prefer to use package management to ensure consistency of dependencies, then you can use the **apt-get** command and the deb packages.
2. Tar file based installers. If you do not have root access or you want to install multiple versions of TensorRT side-by-side for comparison purposes, then you can use the tar file install. The tar file installation uses target dep-style directory structures so that you can install TensorRT libraries for multiple architectures and then do cross compilation.

INT8 support

TensorRT can be used on supported GPUs (such as P4 and P40) to execute networks using INT8 rather than FP32 precision. Networks using INT8 deliver significant performance improvements.

Recurrent Neural Network

LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) are two popular and powerful variations of a Recurrent Neural Network cell. Recurrent neural networks are designed to work with sequences of characters, words, sounds, images, etc. TensorRT 2.1 provides implementations of LSTM, GRU and the original RNN layer.

Using TensorRT 2.1

Ensure you are familiar with the following notes when using this release.

- ▶ Running networks in FP16 or INT8 may not work correctly on platforms without hardware support for the appropriate reduced precision instructions.
- ▶ GTX 750 and K1200 users will need to upgrade to CUDA 8 in order to use TensorRT.
- ▶ If you have previously installed TensorRT 2.0 EA or TensorRT 2.1 RC and you install TensorRT 2.1, you may find that the old meta package is still installed. It can be safely removed with the `apt-get` command.
- ▶ Debian packages are supplied in the form of local repositories. Once you have installed TensorRT, you can safely remove the TensorRT local repository debian package.
- ▶ The implementation of deconvolution is now deterministic. In order to ensure determinism, the new algorithm requires more workspace.
- ▶ FP16 performance was significantly improved for batch size = 1. The new algorithm is sometimes slower for batch sizes greater than one.
- ▶ Calibration for INT8 does not require labeled data. `SampleINT8` uses labels only to compare the accuracy of INT8 inference with the accuracy of FP32 inference.
- ▶ Running with larger batch sizes gives higher overall throughput but uses more memory. When trying TensorRT out on GPUs with smaller memory, be aware that some of the samples may not work with batch sizes of 128.
- ▶ The included Caffe parser library does not currently understand the [NVIDIA/Caffe](#) format for batch normalization. The [BVLC/Caffe](#) batch normalization format is parsed correctly.

Deprecated Features

The parameterized calibration technique introduced in the 2.0 EA pre-release has been replaced by the new entropy calibration mechanism.

- ▶ The Legacy class `IInt8LegacyCalibrator` is deprecated.

Known Issues

- ▶ When using reduced precision, either INT8 or FP16, on platforms with hardware support for those types, pooling with window sizes other than 1,2,3,5 or 7 will fail.
- ▶ When using `MAX_AVERAGE_BLEND` or `AVERAGE` pooling in INT8 with a channel count that is not a multiple of 4, TensorRT may generate incorrect results.
- ▶ When downloading the Faster R-CNN data on Jetson TX1 users may see the following error:

```
ERROR: cannot verify dl.dropboxusercontent.com's certificate,  
issued by 'CN=DigiCert SHA2 High Assurance Server  
CA,OU=www.digicert.com,O=DigiCert Inc,C=US':  
Unable to locally verify the issuer's authority.  
To connect to dl.dropboxusercontent.com insecurely, use `--no-  
check-certificate`.
```

Adding the `--no-check-certificate` flag should resolve the issue.

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, cuFFT, cuSPARSE, DALI, DIGITS, DGX, DGX-1, Jetson, Kepler, NVIDIA Maxwell, NCCL, NVLink, Pascal, Tegra, TensorRT, and Tesla are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2018 NVIDIA Corporation. All rights reserved.