TENSORRT

RN-08624-030_v01 | January 2017

Release Notes

TABLE OF CONTENTS

Chapter 1. TENSORRT OVERVIEW

NVIDIA TensorRT™ is a C++ library that facilitates high performance inference on NVIDIA GPUs. TensorRT takes a network definition and optimizes it by merging tensors and layers, transforming weights, choosing efficient intermediate data formats, and selecting from a large kernel catalog based on layer parameters and measured performance.

TensorRT consists of import methods to help you express your trained deep learning model for TensorRT to optimize and run. It is an optimization tool that applies graph optimization and layer fusion and finds the fastest implementation of that model leveraging a diverse collection of highly optimized kernels, and a runtime that you can use to execute this network in an inference context.

TensorRT includes a full infrastructure that allows you to leverage high speed reduced precision capabilities of Pascal GPUs as an optional optimization.

TensorRT is built with gcc 4.8.

Chapter 2. 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<<30)`** to **`((1<<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_try.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.

Chapter 3. 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, 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 Unites States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

www.nvidia.com