NVIDIA TensorRT

Release Notes | NVIDIA Docs

RN-08624-001_v8.0.1 | July 2021

Table of Contents

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.

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 an 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:

- NVIDIA® Ampere GPU architecture
- NVIDIA® Turing™ GPU architecture
- NVIDIA® Volta™ GPU architecture
- NVIDIA® Pascal™ GPU architecture

Chapter 2. TensorRT Release 8.x.x

2.1. TensorRT Release 8.0.1

This is the TensorRT 8.0.1 release notes and is applicable to x86 Linux and Windows users, as well as PowerPC and ARM Server Base System Architecture (SBSA) users on Linux only.

These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

This release includes several fixes from the previous TensorRT 8.x.x release as well as the following additional changes. For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

- Added support for RedHat/CentOS 8.3, Ubuntu 20.04, and SUSE Linux Enterprise Server 15 Linux distributions. Only a tar file installation is supported on SLES 15 at this time. For more information, refer to the TensorRT Installation Guide.
- Added Python 3.9 support. Use a tar file installation to obtain the new Python wheel files. For more information, refer to the TensorRT Installation Guide.
- Added `ResizeCoordinateTransformation`, `ResizeSelector`, and `ResizeRoundMode`; three new enumerations to `IResizeLayer`, and enhanced `IResizeLayer` to support more resize modes from TensorFlow, PyTorch, and ONNX. For more information, refer to the IResizeLayer section in the *TensorRT Developer Guide*.
- Builder timing cache can be serialized and reused across builder instances. For more information, refer to the Builder Layer Timing Cache and trtexec sections in the *TensorRT Developer Guide*.
- Added convolution and fully-connected tactics which support and make use of structured sparsity in kernel weights. This feature can be enabled by setting the `kSPARSE_WEIGHTS` flag in `IBuilderConfig`. This feature is only available on NVIDIA Ampere GPUs. For more information, refer to the Structured Sparsity section in the *Best Practices For TensorRT Performance* guide. *(not applicable for Jetson platforms)*

- Added two new layers to the API: `IQuantizeLayer` and `IDequantizeLayer` which can be used to explicitly specify the precision of operations and data buffers. ONNX's `QuantizeLinear` and `DequantizeLinear` operators are mapped to these new layers which enables the support for networks trained using Quantization-Aware Training (QAT) methodology. For more information, refer to the Explicit-Quantization, `IQuantizeLayer`, and `IDequantizeLayer` sections in the *TensorRT Developer Guide* and Q/DQ Fusion in the *Best Practices For TensorRT Performance* guide.
- Achieved QuartzNet optimization with support of 1D fused depthwise + pointwise convolution kernel to achieve up to 1.8x end-to-end performance improvement on A100. *(not applicable for Jetson platforms)*
- Added support for the following ONNX operators: `Celu`, `CumSum`, `EyeLike`, `GatherElements`, `GlobalLpPool`, `GreaterOrEqual`, `LessOrEqual`, `LpNormalization`, `LpPool`, `ReverseSequence`, and `SoftmaxCrossEntropyLoss`. For more information, refer to the Supported Ops section in the *TensorRT Support Matrix*.
- Added Sigmoid/Tanh INT8 support for DLA. It allows DLA sub-graph with Sigmoid/Tanh to compile with INT8 by auto-upgrade to FP16 internally. For more information, refer to the DLA Supported Layers section in the *TensorRT Developer Guide*.
- Added DLA native planar format and DLA native gray-scale format support.
- Allow to generate reformat-free engine with DLA when `EngineCapability` is `EngineCapability::kDEFAULT`.
- TensorRT now declares API's with the `noexcept` keyword to clarify that exceptions must not cross the library boundary. All TensorRT classes that an application inherits from (such as `IGpuAllocator`, `IPluginV2`, etc...) must guarantee that methods called by TensorRT do not throw uncaught exceptions, or the behavior is undefined.
- TensorRT reports errors, along with an associated `ErrorCode`, via the `ErrorRecorder` API for all errors. The `ErrorRecorder` will fallback to the legacy logger reporting, with `Severity::kERROR` or `Severity::kINTERNAL_ERROR`, if no error recorder is registered. The `ErrorCodes` allow recovery in cases where TensorRT previously reported non-recoverable situations.
- Improved performance of the `GlobalAveragePooling` operation, which is used in some CNNs like EfficientNet. For transformer based networks with INT8 precision, it's recommended to use a network which is trained using Quantization Aware Training (QAT) and has `IQuantizeLayer` and `IDequantizeLayer` layers in the network definition.
- TensorRT now supports refit weights via names. For more information, refer to Refitting An Engine in the *TensorRT Developer Guide*.
- Refitting performance has been improved. The performance boost can be evident when the weights are large or a large number of weights or layers are updated at the same time.
- Added the following new samples.
  - This sample, engine_refit_onnx_bidaf, builds an engine from the ONNX BiDAF model, and refits the TensorRT engine with weights from the model. The new refit APIs allow users to locate the weights via names from ONNX models instead of layer names and

weights roles. For more information, refer to the Refitting An Engine Built From An ONNX Model In Python in the *TensorRT Sample Support Guide*.

- This sample, efficientdet, demonstrates the conversion and execution of Google EfficientDet models with TensorRT. For more information, refer to the Scalable And Efficient Object Detection With EfficientDet Networks In Python in the *TensorRT Sample Support Guide*.

- Improved performance for the transformer based networks such as BERT and other networks that use Multi-Head Self-Attention.
- Added cuDNN to the `IBuilderConfig::setTacticSources` enum. Use of cuDNN as a source of operator implementations can be enabled or disabled using the `IBuilderConfig::setTacticSources` API function.
- The following C++ API functions were added:
  - `class IDequantizeLayer`
  - `class IQuantizeLayer`
  - `class ITimingCache`
  - `IBuilder::buildSerializedNetwork()`
  - `IBuilderConfig::getTimingCache()`
  - `IBuilderConfig::setTimingCache()`
  - `IGpuAllocator::reallocate()`
  - `INetworkDefinition::addDequantize()`
  - `INetworkDefinition::addQuantize()`
  - `INetworkDefinition::setWeightsName()`
  - `IPluginRegistry::deregisterCreator()`
  - `IRefitter::getMissingWeights()`
  - `IRefitter::getAllWeights()`
  - `IRefitter::setNamedWeights()`
  - `IResizeLayer::getCoordinateTransformation()`
  - `IResizeLayer::getNearestRounding()`
  - `IResizeLayer::getSelectorForSinglePixel()`
  - `IResizeLayer::setCoordinateTransformation()`
  - `IResizeLayer::setNearestRounding()`
  - `IResizeLayer::setSelectorForSinglePixel()`
  - `IScaleLayer::setChannelAxis()`
  - `enum ResizeCoordinateTransformation`
  - `enum ResizeMode`
  - `BuilderFlag::kSPARSE_WEIGHTS`
  - `TacticSource::kCUDNN`

- TensorFormat::kDLA_HWC4
- TensorFormat::kDLA_LINEAR
- TensorFormat::kHWC16

- The following Python API functions were added:
  - class IDequantizeLayer
  - class IQuantizeLayer
  - class ITimingCache
  - Builder.build_serialized_network()
  - IBuilderConfig.get_timing_cache()
  - IBuilderConfig.set_timing_cache()
  - IGpuAllocator.reallocate()
  - INetworkDefinition.add_dequantize()
  - INetworkDefinition.add_quantize()
  - INetworkDefinition.set_weights_name()
  - IPluginRegistry.deregister_creator()
  - Refitter.get_all_weights()
  - Refitter.get_missing_weights()
  - Refitter::set_named_weights()
  - IResizeLayer.coordinate_transformation
  - IResizeLayer.nearest_rounding
  - IResizeLayer.selector_for_single_pixel
  - IScaleLayer.channel_axis
  - enum ResizeCoordinateTransformationDoc
  - enum ResizeMode
  - BuilderFlag.SPARSE_WEIGHTS
  - TacticSource.CUDNN
  - TensorFormat.DLA_HWC4
  - TensorFormat.DLA_LINEAR
  - TensorFormat.HWC16
- The memory reporting mechanism on Linux platforms has been improved to include large allocations that were not previously tracked due to being memory mapped instead of heap allocated.

Breaking API Changes

- Support for Python 2 has been dropped. This means that TensorRT will no longer include wheels for Python 2, and Python samples will not work with Python 2.

- All API's have been marked as `noexcept` where appropriate. The `IErrorRecorder` interface has been fully integrated into the API for error reporting. The Logger is only used as a fallback when the `ErrorRecorder` is not provided by the user.
- Callback changes are now marked `noexcept`, therefore, implementations must also be marked `noexcept`. TensorRT has never catered to exceptions thrown by callbacks, but this is now captured in the API.
- Methods that take parameters of type `void**` where the array of pointers is unmodifiable are now changed to take type `void*const*`.
- `Dims` is now a type alias for `class Dims32`. Code that forward-declares `Dims` should forward-declare `class Dims32; using Dims = Dims32;`.
- Between TensorRT 8.0 EA and TensorRT 8.0 GA the function prototype for `getLogger()` has been moved from `NvInferRuntimeCommon.h` to `NvInferRuntime.h`. You may need to update your application source code if you're using `getLogger()` and were previously only including `NvInferRuntimeCommon.h`.

Compatibility

- TensorRT 8.0.1 has been tested with the following:
  - cuDNN 8.2.1
  - TensorFlow 1.15.5
  - PyTorch 1.8.1
  - ONNX 1.8.0
- This TensorRT release supports CUDA:
  - 11.3 update 1
  - 11.2 update 2
  - 11.1 update 1
  - 11.0 update 1
  - 10.2
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used, however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- For QAT networks, TensorRT 8.0 supports per-tensor and per-axis quantization scales for weights. For activations, only per-tensor quantization is supported. Only symmetric quantization is supported and zero-point weights may be omitted or, if zero-points are provided, all coefficients must have a value of zero.

- Loops and `DataType::kBOOL` are not supported when the static TensorRT library is used. Performance improvements for transformer based architectures such as BERT will also not be available when using the static TensorRT library.
- When using reformat-free I/O, the extent of a tensor in a vectorized dimension might not be a multiple of the vector length. Elements in a partially occupied vector that are not within the tensor are referred to here as *vector-padding*. For example:
  - On GPU
    - for input tensors, the application shall set vector-padding elements to zero.
    - for output tensors, the value of vector-padding elements is undefined. In a future release, TensorRT will support setting them to zero.
  - On DLA
    - for input tensors, vector-padding elements are ignored.
    - for output tensors, vector-padding elements are unmodified.
- When running INT8 networks on DLA using TensorRT, operations must be added to the same subgraph to reduce quantization errors across the subgraph of the network that runs on the DLA by allowing them to fuse and retain higher precision for intermediate results. Breaking apart the subgraph in order to inspect intermediate results by setting the tensors as network output tensors, can result in different levels of quantization errors due to these optimizations being disabled.
- If both `kSPARSE_WEIGHTS` and `kREFIT` flags are set in `IBuilderConfig`, the convolution layers having structured sparse kernel weights cannot be refitted with new kernel weights which do not have structured sparsity. The `IRefitter::setWeights()` will print an error and return `false` in that case.
- Samples which require TensorFlow in order to run, which typically also use UFF models, are not supported on ARM SBSA releases of TensorRT 8.0. There is no good source for TensorFlow 1.15.x for AArch64 that also supports Python 3.8 which can be used to run these samples.
- Using CUDA graph capture on TensorRT execution contexts with CUDA 10.2 on NVIDIA K80 GPUs may lead to graph capture failures. Upgrading to CUDA 11.0 or above will solve the issue. *(not applicable for Jetson platforms)*
- On RHEL and CentOS, the TensorRT RPM packages for CUDA 11.3 cannot be installed alongside any CUDA 11.x Toolkit packages, like the Debian packages, due to RPM packaging limitations. The TensorRT runtime library packages can only be installed alongside CUDA 11.2 and CUDA 11.3 Toolkit packages and the TensorRT development packages can only be installed alongside CUDA 11.3 Toolkit packages. When using the TAR package, the TensorRT CUDA 11.3 build can be used with any CUDA 11.x Toolkit.

Deprecated And Removed Features

The following features are deprecated in TensorRT 8.0.1:

- Deprecation is used to inform developers that some APIs and tools are no longer recommended for use. TensorRT has the following deprecation policy:
  - This policy comes into effect beginning with TensorRT 8.0.
  - Deprecation notices are communicated in the release notes. Deprecated API elements are marked with the `TRT_DEPRECATED` macro where possible.
  - TensorRT provides a 12-month migration period after the deprecation. For any APIs and tools deprecated in TensorRT 7.x, the 12-month migration period starts from the TensorRT 8.0 GA release date.
  - APIs and tools will continue to work during the migration period.
  - After the migration period ends, we reserve the right to remove the APIs and tools in a future release.
- `IRNNLayer` was deprecated in TensorRT 4.0 and has been removed in TensorRT 8.0. `IRNNv2Layer` was deprecated in TensorRT 7.2.1. `IRNNv2Layer` has been deprecated in favor of the loop API, however, it is still available for backwards compatibility. For more information about the loop API, refer to the sampleCharRNN sample with the `--Iloop` option as well as the Working With Loops chapter.
- `IPlugin` and `IPluginFactory` interfaces were deprecated in TensorRT 6.0 and have been removed in TensorRT 8.0. We recommend that you write new plugins or refactor existing ones to target the `IPluginV2DynamicExt` and `IPluginV2IOExt` interfaces. For more information, refer to the Migrating Plugins From TensorRT 6.x Or 7.x To TensorRT 8.x.x section.
- We removed samplePlugin since it was meant to demonstrate the `IPluginExt` interface, which is no longer supported in TensorRT 8.0.
- We have deprecated the Caffe Parser and UFF Parser in TensorRT 7.0. They are still tested and functional in TensorRT 8.0, however, we plan to remove the support in the future. Ensure you migrate your workflow to use `tf2onnx`, `keras2onnx` or TensorFlow-TensorRT (TF-TRT) for deployment.

  If using UFF, ensure you migrate to the ONNX workflow through the enablement of a plugin. ONNX workflow is not dependent on plugin enablement. For plugin enablement of a plugin on ONNX, refer to Estimating Depth with ONNX Models and Custom Layers Using NVIDIA TensorRT.
  - For TensorFlow to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorFlow, ONNX, and TensorRT.
  - For PyTorch to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorRT.

  Caffe and UFF-specific topics in the Developer Guide have been moved to the Appendix section until removal in the subsequent major release.
- Interface functions that provided a destroy function are deprecated in TensorRT 8.0. The destructors will be exposed publicly in order for the delete operator to work as expected on these classes.

- `nvinfer1::NetworkDefinitionCreationFlag::kEXPLICIT_PRECISION` is deprecated. Networks that have `QuantizeLayer` and `DequantizeLayer` layers will be automatically processed using Q/DQ-processing, which includes explicit-precision semantics. Explicit precision is a network-optimizer constraint that prevents the optimizer from performing precision-conversions that are not dictated by the semantics of the network. For more information, refer to the Working With QAT Networks section in the *TensorRT Developer Guide.*
- `nvinfer1::IResizeLayer::setAlignCorners` and `nvinfer1::IResizeLayer::getAlignCorners` are deprecated. Use `nvinfer1::IResizeLayer::setCoordinateTransformation`, `nvinfer1::IResizeLayer::setSelectorForSinglePixel` and `nvinfer1::IResizeLayer::setNearestRounding` instead.
- Destructors for classes with `destroy()` methods were previously protected. They are now public, enabling use of smart pointers for these classes. The `destroy()` methods are deprecated.
- The `CgPersistentLSTMPlugin_TRT` plugin is deprecated.
- sampleMovieLens and sampleMovieLensMPS have been removed from the TensorRT package.
- The following C++ API functions, types, and a field, which were previously deprecated, were removed:

  Core Library:

  - `DimensionType`
  - `Dims::Type`
  - `class DimsCHW`
  - `class DimsNCHW`
  - `class IOutputDimensionFormula`
  - `class IPlugin`
  - `class IPluginFactory`
  - `class IPluginLayer`
  - `class IRNNLayer`
  - `IBuilder::getEngineCapability()`
  - `IBuilder::allowGPUFallback()`
  - `IBuilder::buildCudaEngine()`
  - `IBuilder::canRunOnDLA()`
  - `IBuilder::createNetwork()`
  - `IBuilder::getAverageFindIterations()`
  - `IBuilder::getDebugSync()`
  - `IBuilder::getDefaultDeviceType()`

- IBuilder::getDeviceType()
- IBuilder::getDLACore()
- IBuilder::getFp16Mode()
- IBuilder::getHalf2Mode()
- IBuilder::getInt8Mode()
- IBuilder::getMaxWorkspaceSize()
- IBuilder::getMinFindIterations()
- IBuilder::getRefittable()
- IBuilder::getStrictTypeConstraints()
- IBuilder::isDeviceTypeSet()
- IBuilder::reset()
- IBuilder::resetDeviceType()
- IBuilder::setAverageFindIterations()
- IBuilder::setDebugSync()
- IBuilder::setDefaultDeviceType()
- IBuilder::setDeviceType()
- IBuilder::setDLACore()
- IBuilder::setEngineCapability()
- IBuilder::setFp16Mode()
- IBuilder::setHalf2Mode()
- IBuilder::setInt8Calibrator()
- IBuilder::setInt8Mode()
- IBuilder::setMaxWorkspaceSize()
- IBuilder::setMinFindIterations()
- IBuilder::setRefittable()
- IBuilder::setStrictTypeConstraints()
- ICudaEngine::getWorkspaceSize()
- IMatrixMultiplyLayer::getTranspose()
- IMatrixMultiplyLayer::setTranspose()
- INetworkDefinition::addMatrixMultiply()
- INetworkDefinition::addPlugin()
- INetworkDefinition::addPluginExt()
- INetworkDefinition::addRNN()
- INetworkDefinition::getConvolutionOutputDimensionsFormula()
- INetworkDefinition::getDeconvolutionOutputDimensionsFormula()
- INetworkDefinition::getPoolingOutputDimensionsFormula()

- `INetworkDefinition::setConvolutionOutputDimensionsFormula()`
- `INetworkDefinition::setDeconvolutionOutputDimensionsFormula()`
- `INetworkDefinition::setPoolingOutputDimensionsFormula()`
- `ITensor::getDynamicRange()`
- `TensorFormat::kNHWC8`
- `TensorFormat::NCHW`
- `TensorFormat::kNC2HW2`

Plugins: The following plugin classes were removed:

- `class INvPlugin`
- `createLReLUPlugin()`
- `createClipPlugin()`
- `PluginType`
- `struct SoftmaxTree`

Plugin interface methods: For plugins based on `IPluginV2DynamicExt` and `IPluginV2IOExt`, certain methods with legacy function signatures (derived from `IPluginV2` and `IPluginV2Ext` base classes) which were deprecated and marked for removal in TensorRT 8.0 will no longer be available. Plugins using these interface methods must stop using them or implement the versions with updated signatures, as applicable.

Unsupported plugin methods removed in TensorRT 8.0:

- `IPluginV2DynamicExt::canBroadcastInputAcrossBatch()`
- `IPluginV2DynamicExt::isOutputBroadcastAcrossBatch()`
- `IPluginV2DynamicExt::getTensorRTVersion()`
- `IPluginV2IOExt::configureWithFormat()`
- `IPluginV2IOExt::getTensorRTVersion()`

Use updated versions for supported plugin methods:

- `IPluginV2DynamicExt::configurePlugin()`
- `IPluginV2DynamicExt::enqueue()`
- `IPluginV2DynamicExt::getOutputDimensions()`
- `IPluginV2DynamicExt::getWorkspaceSize()`
- `IPluginV2IOExt::configurePlugin()`

Use newer methods for the following:

- `IPluginV2DynamicExt::supportsFormat()` has been removed,use `IPluginV2DynamicExt::supportsFormatCombination()` instead.
- `IPluginV2IOExt::supportsFormat()` has been removed,use `IPluginV2IOExt::supportsFormatCombination()` instead.

Caffe Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

UFF Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

- The following Python API functions, which were previously deprecated, were removed:

Core library:

- `class DimsCHW`
- `class DimsNCHW`
- `class IPlugin`
- `class IPluginFactory`
- `class IPluginLayer`
- `class IRNNLayer`
- `Builder.build_cuda_engine()`
- `Builder.average_find_iterations`
- `Builder.debug_sync`
- `Builder.fp16_mode`
- `IBuilder.int8_mode`
- `Builder.max_workspace_size`
- `Builder.min_find_iterations`
- `Builder.refittable`
- `Builder.strict_type_constraints`
- `ICudaEngine.max_workspace_size`
- `IMatrixMultiplyLayer.transpose0`
- `IMatrixMultiplyLayer.transpose0`
- `INetworkDefinition.add_matrix_multiply_deprecated()`
- `INetworkDefinition.add_plugin()`
- `INetworkDefinition.add_plugin_ext()`
- `INetworkDefinition.add_rnn()`
- `INetworkDefinition.convolution_output_dimensions_formula`

- `INetworkDefinition.deconvolution_output_dimensions_formula`
- `INetworkDefinition.pooling_output_dimensions_formula`
- `ITensor.get_dynamic_range()`
- `Dims.get_type()`
- `TensorFormat.HWC8`
- `TensorFormat.NCHW`
- `TensorFormat.NCHW2`

Caffe Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

UFF Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

Plugins:

- `class INvPlugin`
- `createLReLUPlugin()`
- `createClipPlugin()`
- `PluginType`
- `struct SoftmaxTree`

- The following Python API functions were removed:

Core library:

- `class DimsCHW`
- `class DimsNCHW`
- `class IPlugin`
- `class IPluginFactory`
- `class IPluginLayer`
- `class IRNNLayer`
- `Builder.build_cuda_engine()`
- `Builder.average_find_iterations`
- `Builder.debug_sync`

- `Builder.fp16_mode`
- `IBuilder.int8_mode`
- `Builder.max_workspace_size`
- `Builder.min_find_iterations`
- `Builder.refittable`
- `Builder.strict_type_constraints`
- `ICudaEngine.max_workspace_size`
- `IMatrixMultiplyLayer.transpose0`
- `IMatrixMultiplyLayer.transpose0`
- `INetworkDefinition.add_matrix_multiply_deprecated()`
- `INetworkDefinition.add_plugin()`
- `INetworkDefinition.add_plugin_ext()`
- `INetworkDefinition.add_rnn()`
- `INetworkDefinition.convolution_output_dimensions_formula`
- `INetworkDefinition.deconvolution_output_dimensions_formula`
- `INetworkDefinition.pooling_output_dimensions_formula`
- `ITensor.get_dynamic_range()`
- `Dims.get_type()`
- `TensorFormat.HWC8`
- `TensorFormat.NCHW`
- `TensorFormat.NCHW2`

Caffe Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `CaffeParser.plugin_factory`
- `CaffeParser.plugin_factory_ext`

UFF Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `UffParser.plugin_factory`
- `UffParser.plugin_factory_ext`

Fixed Issues

- The diagram in IRNNv2Layer was incorrect. The diagram has been updated and fixed.

- Improved build times for convolution layers with dynamic shapes and large range of leading dimensions.
- TensorRT 8.0 no longer requires `libcublas.so.*` to be present on your system when running an application which was linked with the TensorRT static library. The TensorRT static library now requires cuBLAS and other dependencies to be linked at link time and will no longer open these libraries using `dlopen()`.
- TensorRT 8.0 no longer requires an extra Identity layer between the ElementWise and the Constant whose rank is > 4. For TensorRT 7.x versions, cases like Convolution and FullyConnected with bias where ONNX decomposes the bias to ElementWise, there was a fusion which didn't support per element scale. We previously inserted an Identity to workaround this.
- There was a known performance regression compared to TensorRT 7.1 for Convolution layers with kernel size greater than 5x5. For example, it could lead up to 35% performance regression of the VGG16 UFF model compared to TensorRT 7.1. This issue has been fixed in this release.
- When running networks such as Cortana, LSTM Peephole, MLP, and Faster RCNN, there was a 5% to 16% performance regression on GA102 devices and a 7% to 36% performance regression on GA104 devices. This issue has been fixed in this release. *(not applicable for Jetson platforms)*
- Some RNN networks such as Cortana with FP32 precision and batch size of 8 or higher had a 20% performance loss with CUDA 11.0 or higher compared to CUDA 10.2. This issue has been fixed in this release.
- There was an issue when compiling the TensorRT samples with a GCC version less than 5.x and using the static libraries which resulted in the error message `munmap_chunk(): invalid pointer`. RHEL/CentOS 7.x users were most likely to have observed this issue. This issue has been fixed in this release.
- cuTENSOR, used by TensorRT 8.0 EA, was known to have significant performance regressions with the CUDA 11.3 compiler. This regression has been fixed by the CUDA 11.3 update 1 compiler.
- The installation of PyTorch Quantization Toolkit requires Python version >=3.7, GCC version >=5.4. The specific version of Python may be missing from some operating systems and will need to be separately installed. Refer to the README instructions for the workaround.
- On platforms with Python >= 3.8, TensorFlow 1.x must be installed from the NVIDIA Python package index. For example:

```
pip install --extra-index-url https://pypi.ngc.nvidia.com nvidia-tensorflow;
        python_version==3.8
```

- There is an up to 15% performance regression compared to TensorRT 7.2.3 for QuartzNet variants on Volta GPUs.
- MNIST images used by the samples previously had to be downloaded manually. These images are now shipped with the samples.
- You may observe relocation issues during linking if the resulting binary exceeds 2 GB. This can occur if you are linking TensorRT and all of its dependencies into your application

statically. A workaround for this linking issue has been documented in the *TensorRT Sample Support Guide* under Limitations.

- `IProfiler` would not correctly call user-implemented methods when used from the Python API. This issue has been fixed in this release.
- TensorRT memory usage has improved and can be better managed via `IGpuAllocator::reallocate` when more memory is required.
- TensorRT refitting performance has been improved, especially for large weights and when multiple weights are refitted at the same time. Refitting performance will continue to be optimized in later releases.
- The interfaces that took an argument of type `void**` (for example, `enqueueV2`) now declare it as `void*const*`.
- There was an up to 24% performance regression in TensorRT 8.0.0 compared to TensorRT 7.2.3 for networks containing Slice layers on Turing GPUs. This issue has been fixed.
- There was an up to 8% performance regression in TensorRT 8.0.0 compared to TensorRT 7.2.3 for DenseNet variants on Volta GPUs. This issue has been fixed in this release.
- If input tensors with dynamic shapes were found to be inconsistent with the selected optimization profile during engine building or during inference, an error message is issued with graceful program exit instead of assertion failure and abnormal exit.
- When running TensorRT 8.0.0 with cuDNN 8.2.0, there is a known performance regression for the deconvolution layer compared to running with previous cuDNN releases. For example, some deconvolution layers can have up to 7x performance regression on Turing GPUs compared to running with cuDNN 8.0.4. This has been fixed in the latest cuDNN 8.2.1 release.

Announcements

- TensorRT 8.0 will be the last TensorRT release that will provide support for Ubuntu 16.04. This also means TensorRT 8.0 will be the last TensorRT release that will support Python 3.5.
- Python samples use a unified data downloading workflow. Each sample has a YAML (`download.yml`) describing the data files that are required to download via a link before running the sample, if any. The download tool parses the YAML and downloads the data files. All other sample code assumes that the data has been downloaded before the code is invoked. An error will be raised if the data is not correctly downloaded. Refer to the Python sample documentation for more information.

Known Issues

- The TensorRT ARM SBSA cross packages in the CUDA network repository cannot be installed because cuDNN ARM SBSA cross packages are not available, which is a dependency of the TensorRT cross packages. The TensorRT ARM SBSA cross packages may be removed in the near future. You should use the native TensorRT ARM SBSA packages instead.

- There is a known issue that graph capture may fail in some cases for `IExecutionContext::enqueue()` and `IExecutionContext::enqueueV2()`. For more information, refer to the documentation for IExecutionContext::enqueueV2(), including how to work around this issue.
- On PowerPC, some RNN networks have up to a 15% performance regression compared to TensorRT 7.0. *(not applicable for Jetson platforms)*
- Some fusions are not enabled when the TensorRT static library is used. This means there is a performance loss of around 10% for networks like BERT and YOLO3 when linking with the static library compared to the dynamic library. The performance loss depends on precision used and batch size and it can be up to 60% in some cases.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit those weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.
- There is a known performance regression compared to TensorRT 7.1 for some networks dominated by FullyConnected with activation and bias operations:
  - up to 12% in FP32 mode. This will be fixed in a future release.
  - up to 10% in FP16 mode on Maxwell and Pascal GPUs.
- There is an up to 8% performance regression compared to TensorRT 7.1 for some networks with heavy FullyConnected operation like VGG16 on Nano.
- There is a known issue that TensorRT selects `kLINEAR` format when the user uses reformat-free I/O with vectorized formats and with input/output tensors which have only 3 dimensions. The workaround is to add an additional dimension to the tensors with size 1 to make them 4 dimensional tensors.
- As DLA Deconvolution layers with square kernels and strides between 23 and 32 significantly slow down compilation time, they are disabled by TensorRT to run on DLA.
- There are some known false alarms reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommendation to suppress the false alarms is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool.

```
{
  Memory leak errors with dlopen.
   Memcheck:Leak
   match-leak-kinds: definite
   ...
   fun:*dlopen*
   ...
}

{
```

```
  Tegra ioctl false alarm
  Memcheck:Param
  ioctl(TCGETA)
  fun:ioctl
  ...
  obj:*libnvrm_gpu.so*
  ...
  obj:*libcuda.so*
}
```

The suppression file can resolve the false alarms about definite loss related to `dlopen()` and `ioctl()` definite loss on the Tegra platform. The other false alarm which can not be added to the suppression file is a sole `malloc()` call without any call stack.

- There is an up to 150% performance regression compared to TensorRT 7.2.3 for 3D U-Net variants on NVIDIA Ampere GPUs, if the optimal algorithm choice is constrained by the available workspace. To work around this issue, enlarge the workspace size. *(not applicable for Jetson platforms)*
- `PluginFieldCollection` in the Python API may prematurely deallocate `PluginFields`. To work around this, assign the list of plugin fields to a named variable:

```
plugin_fields = [trt.PluginField(...), ...]
plugin_field_collection = trt.PluginFieldCollection(plugin_fields)
```

- The tactic source cuBLASLt cannot be selected on SM 3.x devices for CUDA 10.x. If selected, it will fallback to using cuBLAS. *(not applicable for Jetson platforms)*
- On Jetson devices, the power consumption may increase for the sake of performance improvement when compared against TensorRT 7.1. No significant drop in the performance per watt has been observed.
- There is an up to 15% performance regression compared to TensorRT 7.2.3 for path perception network (Pathnet) in FP32.
- There is an up to 10-11% performance regression on Xavier:
  - compared to TensorRT 7.2.3 for ResNet-152 with batch size 2 in FP16.
  - compared to TensorRT 6 for ResNeXt networks with small batch (1 or 2) in FP32.
- For networks that use deconv with large kernel size, the engine build time could drop a lot for this layer on Xavier. It could also lead to `the launch timed out and was terminated` error message on Jetson Nano/TX1.
- For some networks with large amounts of weights and activation data, DLA may fail compiling the subgraph, and that subgraph will fallback to GPU.
- There is an up to 10% performance regression when TensorRT is used with CUDNN 8.1 or 8.2. When CUDNN 8.0 is used, the performance is recovered. *(not applicable for Jetson platforms)*
- There is an up to 6% performance regression compared to TensorRT 7.2.3 for WaveRNN in FP16 on Volta and Turing platforms.
- On embedded devices, TensorRT attempts to avoid testing kernel candidates whose memory requirements would trigger the Out of Memory (OOM) killer. If it does trigger, consider reducing the memory requirement for the model by reducing index dimensions, or maximize the available memory by closing other applications.

- There is a known accuracy issue of GoogLeNet variants with NVIDIA Ampere GPUs where TF32 mode is enabled by default on windows. *(not applicable for Jetson platforms)*
- There is an up to 40% regression compared to TensorRT 7.2.3 for DenseNet with CUDA 11.3 on P100 and V100. When CUDA 11.0 is used, the regression is recovered. *(not applicable for Jetson platforms)*
- There is an up to 10% performance regression compared to TensorRT 7.2.3 in JetPack 4.5 for ResNet-like networks on NVIDIA DLA when the dynamic ranges of the inputs of the ElementWise `ADD` layers are different. This is due to a fix for a bug in DLA where it ignored the dynamic range of the second input of the ElementWise `ADD` layers and caused some accuracy issues.
- There is a known 4% accuracy regression with Faster R-CNN NasNet network with NVIDIA Ampere and Turing GPUs. *(not applicable for Jetson platforms)*
- Under some conditions, `RNNv2Layer` can require a larger workspace size than previous versions of TensorRT in order to run all supported tactics. Consider increasing the workspace size to work around this issue.
- Engine build times for TensorRT 8.0 may be slower than TensorRT 7.2 due to the engine optimizer being more aggressive.
- There is an up to 30% performance regression with QAT (quantization-aware-training) EfficientNet networks on V100 compared to TensorRT 7.2. *(not applicable for Jetson platforms)*
- The new Python sample efficientdet is only available in the OSS release and will be added in the core package in the next release.

2.2. TensorRT Release 8.0.0 Early Access (EA)

This is the TensorRT 8.0.0 Early Access (EA) release notes and is applicable to Linux x86 users.

These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

This release includes several fixes from the previous TensorRT 7.x.x release as well as the following additional changes. For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

- Added support for RedHat/CentOS 8.3, Ubuntu 20.04, and SUSE Linux Enterprise Server 15 Linux distributions. Only a tar file installation is supported on SLES 15 at this time. For more information, refer to the TensorRT Installation Guide.

- Added Python 3.9 support. Use a tar file installation to obtain the new Python wheel files. For more information, refer to the TensorRT Installation Guide.
- Added `ResizeCoordinateTransformation`, `ResizeSelector`, and `ResizeRoundMode`; three new enumerations to `IResizeLayer`, and enhanced `IResizeLayer` to support more resize modes from TensorFlow, PyTorch, and ONNX. For more information, refer to the IResizeLayer section in the *TensorRT Developer Guide*.
- Builder timing cache can be serialized and reused across builder instances. For more information, refer to the Builder Layer Timing Cache and trtexec sections in the *TensorRT Developer Guide*.
- Added convolution and fully-connected tactics which support and make use of structured sparsity in kernel weights. This feature can be enabled by setting the `kSPARSE_WEIGHTS` flag in `IBuilderConfig`. This feature is only available on NVIDIA Ampere GPUs. For more information, refer to the Structured Sparsity section in the *Best Practices For TensorRT Performance* guide. *(not applicable for Jetson platforms)*
- Added two new layers to the API: `IQuantizeLayer` and `IDequantizeLayer` which can be used to explicitly specify the precision of operations and data buffers. ONNX's `QuantizeLinear` and `DequantizeLinear` operators are mapped to these new layers which enables the support for networks trained using Quantization-Aware Training (QAT) methodology. For more information, refer to the Explicit-Quantization, `IQuantizeLayer`, and `IDequantizeLayer` sections in the *TensorRT Developer Guide* and Q/DQ Fusion in the *Best Practices For TensorRT Performance* guide.
- Achieved QuartzNet optimization with support of 1D fused depthwise + pointwise convolution kernel to achieve up to 1.8x end-to-end performance improvement on A100. *(not applicable for Jetson platforms)*
- Added support for the following ONNX operators: `Celu`, `CumSum`, `EyeLike`, `GatherElements`, `GlobalLpPool`, `GreaterOrEqual`, `LessOrEqual`, `LpNormalization`, `LpPool`, `ReverseSequence`, and `SoftmaxCrossEntropyLoss`. For more information, refer to the Supported Ops section in the *TensorRT Support Matrix*.
- Added Sigmoid/Tanh INT8 support for DLA. It allows DLA sub-graph with Sigmoid/Tanh to compile with INT8 by auto-upgrade to FP16 internally. For more information, refer to the DLA Supported Layers section in the *TensorRT Developer Guide*.
- Added DLA native planar format and DLA native gray-scale format support.
- Allow to generate reformat-free engine with DLA when `EngineCapability` is `EngineCapability::kDEFAULT`.
- TensorRT now declares API's with the `noexcept` keyword to clarify that exceptions must not cross the library boundary. All TensorRT classes that an application inherits from (such as `IGpuAllocator`, `IPluginV2`, etc...) must guarantee that methods called by TensorRT do not throw uncaught exceptions, or the behavior is undefined.
- TensorRT reports errors, along with an associated `ErrorCode`, via the `ErrorRecorder` API for all errors. The `ErrorRecorder` will fallback to the legacy logger reporting, with `Severity::kERROR` or `Severity::kINTERNAL_ERROR`, if no error recorder is registered.

The `ErrorCodes` allow recovery in cases where TensorRT previously reported non-recoverable situations.

- Improved performance of the `GlobalAveragePooling` operation, which is used in some CNNs like EfficientNet. For transformer based networks with INT8 precision, it's recommended to use a network which is trained using Quantization Aware Training (QAT) and has `IQuantizeLayer` and `IDequantizeLayer` layers in the network definition.
- TensorRT now supports refit weights via names. For more information, refer to Refitting An Engine in the *TensorRT Developer Guide*.
- Refitting performance has been improved. The performance boost can be evident when the weights are large or a large number of weights or layers are updated at the same time.
- Added a new sample.This sample, engine_refit_onnx_bidaf, builds an engine from the ONNX BiDAF model, and refits the TensorRT engine with weights from the model. The new refit APIs allow users to locate the weights via names from ONNX models instead of layer names and weights roles. For more information, refer to the Refitting An Engine Built From An ONNX Model In Python in the *TensorRT Sample Support Guide*.
- Improved performance for the transformer based networks such as BERT and other networks that use Multi-Head Self-Attention.
- Added cuDNN to the `IBuilderConfig::setTacticSources` enum. Use of cuDNN as a source of operator implementations can be enabled or disabled using the `IBuilderConfig::setTacticSources` API function.
- The following C++ API functions were added:
  - `class IDequanzizeLayer`
  - `class IQuantizeLayer`
  - `class ITimingCache`
  - `IBuilder::buildSerializedNetwork()`
  - `IBuilderConfig::getTimingCache()`
  - `IBuilderConfig::setTimingCache()`
  - `IGpuAllocator::reallocate()`
  - `INetworkDefinition::addDequantize()`
  - `INetworkDefinition::addQuantize()`
  - `INetworkDefinition::setWeightsName()`
  - `IPluginRegistry::deregisterCreator()`
  - `IRefitter::getMissingWeights()`
  - `IRefitter::getAllWeights()`
  - `IRefitter::setNamedWeights()`
  - `IResizeLayer::getCoordinateTransformation()`
  - `IResizeLayer::getNearestRounding()`
  - `IResizeLayer::getSelectorForSinglePixel()`

- `IResizeLayer::setCoordinateTransformation()`
- `IResizeLayer::setNearestRounding()`
- `IResizeLayer::setSelectorForSinglePixel()`
- `IScaleLayer::setChannelAxis()`
- `enum ResizeCoordinateTransformation`
- `enum ResizeMode`
- `BuilderFlag::kSPARSE_WEIGHTS`
- `TacticSource::kCUDNN`
- `TensorFormat::kDLA_HWC4`
- `TensorFormat::kDLA_LINEAR`
- `TensorFormat::kHWC16`

- The following Python API functions were added:
  - `class IDequanzizeLayer`
  - `class IQuantizeLayer`
  - `class ITimingCache`
  - `Builder.build_serialized_network()`
  - `IBuilderConfig.get_timing_cache()`
  - `IBuilderConfig.set_timing_cache()`
  - `IGpuAllocator.reallocate()`
  - `INetworkDefinition.add_dequantize()`
  - `INetworkDefinition.add_quantize()`
  - `INetworkDefinition.set_weights_name()`
  - `IPluginRegistry.deregister_creator()`
  - `IRefitter.get_missing_weights()`
  - `IRefitter.get_all_weights()`
  - `IRefitter::set_named_weights()`
  - `IResizeLayer.coordinate_transformation`
  - `IResizeLayer.nearest_rounding`
  - `IResizeLayer.selector_for_single_pixel`
  - `IScaleLayer.channel_axis`
  - `enum ResizeCoordinateTransformation`
  - `enum ResizeMode`
  - `BuilderFlag.SPARSE_WEIGHTS`
  - `TacticSource.CUDNN`
  - `TensorFormat.DLA_HWC4`
  - `TensorFormat.DLA_LINEAR`

- `TensorFormat.HWC16`

Breaking API Changes

- Support for Python 2 has been dropped. This means that TensorRT will no longer include wheels for Python 2, and Python samples will not work with Python 2.
- All API's have been marked as noexcept where appropriate. The `IErrorRecorder` interface has been fully integrated into the API for error reporting. The Logger is only used as a fallback when the `ErrorRecorder` is not provided by the user.
- Callback changes are now marked `noexcept`, therefore, implementations must also be marked `noexcept`. TensorRT has never catered to exceptions thrown by callbacks, but this is now captured in the API.
- Methods that take parameters of type `void**` where the array of pointers is unmodifiable are now changed to take type `void*const*`.
- `Dims` is now a type alias for `class Dims32`. Code that forward-declares `Dims` should forward-declare `class Dims32; using Dims = Dims32;`.

Compatibility

- TensorRT 8.0.0 EA has been tested with the following:
  - cuDNN 8.2.0
  - TensorFlow 1.15.5
  - PyTorch 1.8.0
  - ONNX 1.8.0
- This TensorRT release supports CUDA:
  - 10.2
  - 11.0 update 1
  - 11.1 update 1
  - 11.2 update 2
  - 11.3

Note: There are two TensorRT binary builds for CUDA 11.0 and CUDA 11.3. The build for CUDA 11.3 is compatible with CUDA 11.1 and CUDA 11.2 libraries. For both builds, CUDA driver compatible with the runtime CUDA version is required (see Table 2 here). For the CUDA 11.3 build, driver version 465 or above is suggested for best performance.

- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used, however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- For QAT networks, TensorRT 8.0 supports per-tensor and per-axis quantization scales for weights. For activations, only per-tensor quantization is supported. Only symmetric quantization is supported and zero-point weights may be omitted or, if zero-points are provided, all coefficients must have a value of zero.
- Loops and `DataType::kBOOL` are not supported when the static TensorRT library is used. Performance improvements for transformer based architectures such as BERT will also not be available when using static TensorRT library.
- When using reformat-free I/O, the extent of a tensor in a vectorized dimension might not be a multiple of the vector length. Elements in a partially occupied vector that are not within the tensor are referred to here as *vector-padding*. For example:
  - On GPU
    - for input tensors, the application shall set vector-padding elements to zero.
    - for output tensors, the value of vector-padding elements is undefined. In a future release, TensorRT will support setting them to zero.
  - On DLA
    - for input tensors, vector-padding elements are ignored.
    - for output tensors, vector-padding elements are unmodified.
- When running INT8 networks on DLA using TensorRT, operations must be added to the same subgraph to reduce quantization errors across the subgraph of the network that runs on the DLA by allowing them to fuse and retain higher precision for intermediate results. Breaking apart the subgraph in order to inspect intermediate results by setting the tensors as network output tensors, can result in different levels of quantization errors due to these optimizations being disabled.
- If both `kSPARSE_WEIGHTS` and `kREFIT` flags are set in `IBuilderConfig`, the convolution layers having structured sparse kernel weights cannot be refitted with new kernel weights which do not have structured sparsity. The `IRefitter::setWeights()` will print an error and return `false` in that case.

Deprecated And Removed Features

The following features are deprecated in TensorRT 8.0.0:

- Deprecation is used to inform developers that some APIs and tools are no longer recommended for use. TensorRT has the following deprecation policy:
  - This policy comes into effect beginning with TensorRT 8.0.
  - Deprecation notices are communicated in the release notes. Deprecated API elements are marked with the `TRT_DEPRECATED` macro where possible.

- TensorRT provides a 12-month migration period after the deprecation. For any APIs and tools deprecated in TensorRT 7.x, the 12-month migration period starts from the TensorRT 8.0 GA release date.
- APIs and tools will continue to work during the migration period.
- After the migration period ends, we reserve the right to remove the APIs and tools in a future release.

- `IRNNLayer` was deprecated in TensorRT 4.0 and has been removed in TensorRT 8.0. `IRNNv2Layer` was deprecated in TensorRT 7.2.1. `IRNNv2Layer` has been deprecated in favor of the loop API, however, it is still available for backwards compatibility. For more information about the loop API, refer to the sampleCharRNN sample with the `--Iloop` option as well as the Working With Loops chapter.
- `IPlugin` and `IPluginFactory` interfaces were deprecated in TensorRT 6.0 and have been removed in TensorRT 8.0. We recommend that you write new plugins or refactor existing ones to target the `IPluginV2DynamicExt` and `IPluginV2IOExt` interfaces. For more information, refer to the Migrating Plugins From TensorRT 6.x Or 7.x To TensorRT 8.x.x section.
- We removed samplePlugin since it was meant to demonstrate the `IPluginExt` interface, which is no longer supported in TensorRT 8.0.
- We have deprecated the Caffe Parser and UFF Parser in TensorRT 7.0. They are still tested and functional in TensorRT 8.0, however, we plan to remove the support in the future. Ensure you migrate your workflow to use `tf2onnx`, `keras2onnx` or TensorFlow-TensorRT (TF-TRT) for deployment.

  If using UFF, ensure you migrate to the ONNX workflow through enablement of a plugin. ONNX workflow is not dependent on plugin enablement. For plugin enablement of a plugin on ONNX, refer to Estimating Depth with ONNX Models and Custom Layers Using NVIDIA TensorRT.

  - For TensorFlow to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorFlow, ONNX, and TensorRT.
  - For PyTorch to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorRT.

  Caffe and UFF-specific topics in the Developer Guide have been moved to the Appendix section until removal in the subsequent major release.
- Interface functions that provided a destroy function are deprecated in TensorRT 8.0. The destructors will be exposed publicly in order for the delete operator to work as expected on these classes.
- `nvinfer1::NetworkDefinitionCreationFlag::kEXPLICIT_PRECISION` is deprecated. Networks that have `QuantizeLayer` and `DequantizeLayer` layers will be automatically processed using Q/DQ-processing, which includes explicit-precision semantics. Explicit precision is a network-optimizer constraint that prevents the optimizer from performing precision-conversions that are not dictated by the semantics of the network. For more

information, refer to the Working With QAT Networks section in the *TensorRT Developer Guide*.

- `nvinfer1::IResizeLayer::setAlignCorners` and `nvinfer1::IResizeLayer::getAlignCorners` are deprecated. Use `nvinfer1::IResizeLayer::setCoordinateTransformation`, `nvinfer1::IResizeLayer::setSelectorForSinglePixel` and `nvinfer1::IResizeLayer::setNearestRounding` instead.
- Destructors for classes with `destroy()` methods were previously protected. They are now public, enabling use of smart pointers for these classes. The `destroy()` methods are deprecated.
- The following C++ API functions, types, and a field, which were previously deprecated, were removed:

  Core Library:

  - `DimensionType`
  - `Dims::Type`
  - `class DimsCHW`
  - `class DimsNCHW`
  - `class IOutputDimensionFormula`
  - `class IPlugin`
  - `class IPluginFactory`
  - `class IPluginLayer`
  - `class IRNNLayer`
  - `IBuilder::getEngineCapability()`
  - `IBuilder::allowGPUFallback()`
  - `IBuilder::buildCudaEngine()`
  - `IBuilder::canRunOnDLA()`
  - `IBuilder::createNetwork()`
  - `IBuilder::getAverageFindIterations()`
  - `IBuilder::getDebugSync()`
  - `IBuilder::getDefaultDeviceType()`
  - `IBuilder::getDeviceType()`
  - `IBuilder::getDLACore()`
  - `IBuilder::getFp16Mode()`
  - `IBuilder::getHalf2Mode()`
  - `IBuilder::getInt8Mode()`
  - `IBuilder::getMaxWorkspaceSize()`
  - `IBuilder::getMinFindIterations()`

- IBuilder::getRefittable()
- IBuilder::getStrictTypeConstraints()
- IBuilder::isDeviceTypeSet()
- IBuilder::reset()
- IBuilder::resetDeviceType()
- IBuilder::setAverageFindIterations()
- IBuilder::setDebugSync()
- IBuilder::setDefaultDeviceType()
- IBuilder::setDeviceType()
- IBuilder::setDLACore()
- IBuilder::setEngineCapability()
- IBuilder::setFp16Mode()
- IBuilder::setHalf2Mode()
- IBuilder::setInt8Calibrator()
- IBuilder::setInt8Mode()
- IBuilder::setMaxWorkspaceSize()
- IBuilder::setMinFindIterations()
- IBuilder::setRefittable()
- IBuilder::setStrictTypeConstraints()
- ICudaEngine::getWorkspaceSize()
- IMatrixMultiplyLayer::getTranspose()
- IMatrixMultiplyLayer::setTranspose()
- INetworkDefinition::addMatrixMultiply()
- INetworkDefinition::addPlugin()
- INetworkDefinition::addPluginExt()
- INetworkDefinition::addRNN()
- INetworkDefinition::getConvolutionOutputDimensionsFormula()
- INetworkDefinition::getDeconvolutionOutputDimensionsFormula()
- INetworkDefinition::getPoolingOutputDimensionsFormula()
- INetworkDefinition::setConvolutionOutputDimensionsFormula()
- INetworkDefinition::setDeconvolutionOutputDimensionsFormula()
- INetworkDefinition::setPoolingOutputDimensionsFormula()
- ITensor::getDynamicRange()
- TensorFormat::kNHWC8
- TensorFormat::NCHW
- TensorFormat::kNC2HW2

Plugins: The following plugin classes were removed:

- `class INvPlugin`
- `createLReLUPlugin()`
- `createClipPlugin()`
- `PluginType`
- `struct SoftmaxTree`

Plugin interface methods: For plugins based on `IPluginV2DynamicExt` and `IPluginV2IOExt`, certain methods with legacy function signatures (derived from `IPluginV2` and `IPluginV2Ext` base classes) which were deprecated and marked for removal in TensorRT 8.0 will no longer be available. Plugins using these interface methods must stop using them or implement the versions with updated signatures, as applicable.

Unsupported plugin methods removed in TensorRT 8.0:

- `IPluginV2DynamicExt::canBroadcastInputAcrossBatch()`
- `IPluginV2DynamicExt::isOutputBroadcastAcrossBatch()`
- `IPluginV2DynamicExt::getTensorRTVersion()`
- `IPluginV2IOExt::configureWithFormat()`
- `IPluginV2IOExt::getTensorRTVersion()`

Use updated versions for supported plugin methods:

- `IPluginV2DynamicExt::configurePlugin()`
- `IPluginV2DynamicExt::enqueue()`
- `IPluginV2DynamicExt::getOutputDimensions()`
- `IPluginV2DynamicExt::getWorkspaceSize()`
- `IPluginV2IOExt::configurePlugin()`

Use newer methods for the following:

- `IPluginV2DynamicExt::supportsFormat()` has been removed,use `IPluginV2DynamicExt::supportsFormatCombination()` instead.
- `IPluginV2IOExt::supportsFormat()` has been removed,use `IPluginV2IOExt::supportsFormatCombination()` instead.

Caffe Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

UFF Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

- The following Python API functions, which were previously deprecated, were removed:

  Core library:

  - `class DimsCHW`
  - `class DimsNCHW`
  - `class IPlugin`
  - `class IPluginFactory`
  - `class IPluginLayer`
  - `class IRNNLayer`
  - `Builder.build_cuda_engine()`
  - `Builder.average_find_iterations`
  - `Builder.debug_sync`
  - `Builder.fp16_mode`
  - `IBuilder.int8_mode`
  - `Builder.max_workspace_size`
  - `Builder.min_find_iterations`
  - `Builder.refittable`
  - `Builder.strict_type_constraints`
  - `ICudaEngine.max_workspace_size`
  - `IMatrixMultiplyLayer.transpose0`
  - `IMatrixMultiplyLayer.transpose0`
  - `INetworkDefinition.add_matrix_multiply_deprecated()`
  - `INetworkDefinition.add_plugin()`
  - `INetworkDefinition.add_plugin_ext()`
  - `INetworkDefinition.add_rnn()`
  - `INetworkDefinition.convolution_output_dimensions_formula`
  - `INetworkDefinition.deconvolution_output_dimensions_formula`
  - `INetworkDefinition.pooling_output_dimensions_formula`
  - `ITensor.get_dynamic_range()`
  - `Dims.get_type()`
  - `TensorFormat.HWC8`
  - `TensorFormat.NCHW`

- `TensorFormat.NCHW2`

Caffe Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

UFF Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `setPluginFactory()`
- `setPluginFactoryExt()`

Plugins:

- `class INvPlugin`
- `createLReLUPlugin()`
- `createClipPlugin()`
- `PluginType`
- `struct SoftmaxTree`

- The following Python API functions were removed:

Core library:

- `class DimsCHW`
- `class DimsNCHW`
- `class IPlugin`
- `class IPluginFactory`
- `class IPluginLayer`
- `class IRNNLayer`
- `Builder.build_cuda_engine()`
- `Builder.average_find_iterations`
- `Builder.debug_sync`
- `Builder.fp16_mode`
- `IBuilder.int8_mode`
- `Builder.max_workspace_size`
- `Builder.min_find_iterations`
- `Builder.refittable`
- `Builder.strict_type_constraints`

- `ICudaEngine.max_workspace_size`
- `IMatrixMultiplyLayer.transpose0`
- `IMatrixMultiplyLayer.transpose0`
- `INetworkDefinition.add_matrix_multiply_deprecated()`
- `INetworkDefinition.add_plugin()`
- `INetworkDefinition.add_plugin_ext()`
- `INetworkDefinition.add_rnn()`
- `INetworkDefinition.convolution_output_dimensions_formula`
- `INetworkDefinition.deconvolution_output_dimensions_formula`
- `INetworkDefinition.pooling_output_dimensions_formula`
- `ITensor.get_dynamic_range()`
- `Dims.get_type()`
- `TensorFormat.HWC8`
- `TensorFormat.NCHW`
- `TensorFormat.NCHW2`

Caffe Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `CaffeParser.plugin_factory`
- `CaffeParser.plugin_factory_ext`

UFF Parser:

- `class IPluginFactory`
- `class IPluginFactoryExt`
- `UffParser.plugin_factory`
- `UffParser.plugin_factory_ext`

Fixed Issues

- Improved build times for convolution layers with dynamic shapes and large range of leading dimensions.
- TensorRT 8.0 no longer requires `libcublas.so.*` to be present on your system when running an application which was linked with the TensorRT static library. The TensorRT static library now requires cuBLAS and other dependencies to be linked at link time and will no longer open these libraries using `dlopen()`.
- TensorRT 8.0 no longer requires an extra Identity layer between the ElementWise and the Constant whose rank is > 4. For TensorRT 7.x versions, cases like Convolution and FullyConnected with bias where ONNX decomposes the bias to ElementWise, there was

a fusion which didn't support per element scale. We previously inserted an Identity to workaround this.

- There was a known performance regression compared to TensorRT 7.1 for Convolution layers with kernel size greater than 5x5. For example, it could lead up to 35% performance regression of the VGG16 UFF model compared to TensorRT 7.1. This issue has been fixed in this release.
- When running networks such as Cortana, LSTM Peephole, MLP, and Faster RCNN, there was a 5% to 16% performance regression on GA102 devices and a 7% to 36% performance regression on GA104 devices. This issue has been fixed in this release. *(not applicable for Jetson platforms)*
- Some RNN networks such as Cortana with FP32 precision and batch size of 8 or higher had a 20% performance loss with CUDA 11.0 or higher compared to CUDA 10.2. This issue has been fixed in this release.

Announcements

- TensorRT 8.0 will be the last TensorRT release that will provide support for Ubuntu 16.04. This also means TensorRT 8.0 will be the last TensorRT release that will support Python 3.5.
- Python samples use a unified data downloading workflow. Each sample has a YAML (`download.yml`) describing the data files that are required to download via a link before running the sample, if any. The download tool parses the YAML and downloads the data files. All other sample code assumes that the data has been downloaded before the code is invoked. An error will be raised if the data is not correctly downloaded. Refer to the Python sample documentation for more information.

Known Issues

- The diagram in IRNNv2Layer is incorrect. This will be fixed in a future release.
- There is a known issue that graph capture may fail in some cases for `IExecutionContext::enqueue()` and `IExecutionContext::enqueueV2()`. For more information, refer to the documentation for IExecutionContext::enqueueV2(), including how to work around this issue.
- Some fusions are not enabled when the TensorRT static library is used. This means there is a performance loss of around 10% for networks like BERT and YOLO3 when linking with the static library compared to the dynamic library. The performance loss depends on precision used and batch size and it can be up to 60% in some cases.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit those weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.

- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.
- There is a known performance regression compared to TensorRT 7.1 for some networks dominated by FullyConnected with activation and bias operations:
  - up to 12% in FP32 mode. This will be fixed in a future release.
  - up to 10% in FP16 mode on Maxwell and Pascal GPUs.
- There is an up to 8% performance regression compared to TensorRT 7.1 for some networks with heavy FullyConnected operation on Nano.
- There is an up to 15% performance regression compared to TensorRT 7.2.3 for QuartzNet variants on Volta GPUs.
- There is an up to 150% performance regression compared to TensorRT 7.2.3 for 3D U-Net variants on NVIDIA Ampere GPUs if the workspace size is limited to 1GB. Enlarging the workspace size (for example, to 2GB) can workaround this issue.
- There is a known issue that TensorRT selects `kLINEAR` format when the user uses reformat-free I/O with vectorized formats and with input/output tensors which have only 3 dimensions. The workaround is to add an additional dimension to the tensors with size 1 to make them 4 dimensional tensors.
- CuTensor based algorithms on TensorRT 8.0 EA are known to have significant performance regressions due to an issue with the CUDA 11.3 compiler (5x-10x slower than CUDA 11.0 builds). This is due to a compiler regression and the performance should be recovered with a future CUDA release.
- When running TensorRT 8.0.0 with cuDNN 8.2.0, there is a known performance regression for the deconvolution layer compared to running with previous cuDNN releases. For example, some deconvolution layers can have up to 7x performance regression on Turing GPUs compared to running with cuDNN 8.0.4. This will be fixed in a future cuDNN release.
- There is a known false alarm reported by the Valgrind memory leak check tool when detecting potential memory leaks from TensorRT applications. The recommended way for suppressing the false alarm is to provide a Valgrind suppression file with the following contents when running the Valgrind memory leak check tool.

```
{
   Ignore the dlopen false alarm.
   Memcheck:Leak
   ...
   fun:_dl_open
   ...
}
```

- There is an up to 8% performance regression compared to TensorRT 7.2.3 for DenseNet variants on Volta GPUs.
- There is an up to 24% performance regression compared to TensorRT 7.2.3 for networks containing Slice layers on Turing GPUs.

- While using the TensorRT static library, users are still required to have the cuDNN/cuBLAS dynamic libraries installed at runtime. This issue will be resolved in the GA release so that cuDNN/cuBLAS static libraries will always be used instead.
- An issue was discovered while compiling the TensorRT samples using the TensorRT static libraries with a GCC version older than 5.x. When using RHEL/CentOS 7.x, you may observe a crash with the error message `munmap_chunk(): invalid pointer` if the patch below is not applied. More details regarding this issue with a workaround for your own application can be found in the TensorRT Sample Support Guide.

```
--- a/samples/Makefile.config
+++ b/samples/Makefile.config
@@ -331,13 +331,13 @@ $(OUTDIR)/$(OUTNAME_DEBUG) : $(DOBJS) $(CUDOBJS)
 else
 $(OUTDIR)/$(OUTNAME_RELEASE) : $(OBJS) $(CUOBJS)
   $(ECHO) Linking: $@
- $(AT)$(CC) -o $@ $^ $(LFLAGS) -Wl,--start-group $(LIBS) -Wl,--end-group
+ $(AT)$(CC) -o $@ $(LFLAGS) -Wl,--start-group $(LIBS) $^ -Wl,--end-group
   # Copy every EXTRA_FILE of this sample to bin dir
   $(foreach EXTRA_FILE,$(EXTRA_FILES), cp -f $(EXTRA_FILE)$(OUTDIR)/$(EXTRA_FILE); )

 $(OUTDIR)/$(OUTNAME_DEBUG) : $(DOBJS) $(CUDOBJS)
   $(ECHO) Linking: $@
- $(AT)$(CC) -o $@ $^ $(LFLAGSD) -Wl,--start-group $(DLIBS) -Wl,--end-group
+ $(AT)$(CC) -o $@ $(LFLAGSD) -Wl,--start-group $(DLIBS) $^ -Wl,--end-group
 endif

 $(OBJDIR)/%.o: %.cpp
```

- The tactic source cuBLASLt cannot be selected on SM 3.x devices for CUDA 10.x. If selected, it will fallback to using cuBLAS.

Chapter 3. TensorRT Release 7.x.x

3.1. TensorRT Release 7.2.3

ATTENTION:

This is the TensorRT 7.2.3 GA release notes for Windows and Linux x86 users. For NVIDIA Jetson Linux for Tegra users, TensorRT 7.2.3 is an Early Access (EA) release specifically for MLPerf Inference. For production use of TensorRT, we recommend using the TensorRT 7.1.3 GA.

These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

Note: The release schedule for the TensorRT 7.2.3 Python `pip` packages is aligned to the Deep Learning Frameworks 21.03 release and may not be available at the same time as the TensorRT 7.2.3 general release.

This release includes several fixes from the previous TensorRT 7.x.x release as well as the following additional changes. For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

- Updated the list of supported TensorFlow ops. See Supported Ops for more information.

Breaking API Changes

- When building the TensorRT samples statically using the `TRT_STATIC=1` make option, the suffix `_static` will be appended to the output binary file name. For more information, refer to the Building Samples Using Static Libraries section.

Compatibility

- TensorRT 7.2.3 has been tested with the following:
  - cuDNN 8.1.1
  - TensorFlow 1.15.3
  - PyTorch 1.5.0
  - ONNX 1.6.0
- This TensorRT release supports CUDA 10.2, 11.0 update 1, 11.1 update 1, and 11.2 update 1.

> **Note:** If you are developing an application that is being compiled with CUDA 11.2 or you are using CUDA 11.2 libraries to run your application, then you must install CUDA 11.1 using either the Debian/RPM packages or using a CUDA 11.1 tar/zip/exe package. NVRTC from CUDA 11.1 is a runtime requirement of TensorRT and must be present to run TensorRT applications. If you are using the network repo installation method, this additional step is not needed.

- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section. Other semantically compatible releases of cuDNN and cuBLAS can be used, however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- TensorRT 7.2 only supports per-tensor quantization scales for both activations and weights in explicit precision mode. No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, refer to the Working With Explicit Precision Using C++ section.
- Loops and `DataType::kBOOL` are not supported when the static TensorRT library is used.
- When using reformat-free I/O, the extent of a tensor in a vectorized dimension might not be a multiple of the vector length. Elements in a partially occupied vector that are not within the tensor are referred to here as *vector-padding*. For example:
  - On GPU
    - for input tensors, the application shall set vector-padding elements to zero.
    - for output tensors, the value of vector-padding elements is undefined. In a future release, TensorRT will support setting them to zero.
  - On DLA
    - for input tensors, vector-padding elements are ignored.
    - for output tensors, vector-padding elements are unmodified.

- When running INT8 networks on DLA using TensorRT, operations must be added to the same subgraph to reduce quantization errors across the subgraph of the network that runs on the DLA by allowing them to fuse and retain higher precision for intermediate results. Breaking apart the subgraph in order to inspect intermediate results by setting the tensors as network output tensors, can result in different levels of quantization errors due to these optimizations being disabled.
- The `IExecutionContext` contains shared resources, therefore, calling `enqueue` or `enqueueV2` in from the same `IExecutionContext` object with different CUDA streams concurrently results in undefined behavior. To perform inference concurrently in multiple CUDA streams, use one `IExecutionContext` per CUDA stream.

Deprecated Features

The following features are deprecated in TensorRT 7.2.3:

- `IRNNLayer` was deprecated in TensorRT 4.0 and will be removed in TensorRT 8.0. `IRNNv2Layer` was deprecated in TensorRT 7.2.1 and will be removed in TensorRT 9.0. `IRNNv2Layer` has been deprecated in favor of the loop API, however, it is still available for backwards compatibility. For more information about the loop API, refer to the sampleCharRNN sample with the `--loop` option as well as the Working With Loops chapter.
- We have deprecated the Caffe Parser and UFF Parser in TensorRT 7.0. They will be tested and functional in the next major release of TensorRT 8.0, however, we plan to remove the support in the subsequent major release. Ensure you migrate your workflow to use `tf2onnx`, `keras2onnx` or TensorFlow-TensorRT (TF-TRT) for deployment.

  If using UFF, ensure you migrate to the ONNX workflow through enablement of a plugin. ONNX workflow is not dependent on plugin enablement. For plugin enablement of a plugin on ONNX, refer to Estimating Depth with ONNX Models and Custom Layers Using NVIDIA TensorRT.

  - For TensorFlow to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorFlow, ONNX, and TensorRT.
  - For PyTorch to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorRT.
- We have deprecated `TensorFormat::kCHW4` as the DLA color image format. Instead, use `TensorFormat::kDLA_HWC4` to specify the DLA color image formats.
- Interface functions that provided a destroy function will be deprecated in TensorRT 8.0 and will be removed in TensorRT 10.0. The destructors will be exposed publicly in order for the delete operator to work as expected on these classes.

Fixed Issues

- `IIdentityLayer` was broken prior to TensorRT 7.0, such that it worked only for identity operations and FP16 to FP32 conversions. The problem was fixed in TensorRT 7.0,

however, the fix was omitted from the Release Notes. See comments in `NvInfer.h` about class `IIdentityLayer` for a list of supported conversions.

- Fixed a bug in the builder where networks with depthwise separable convolution layers whose output was also an FP32 output of the network would have failed earlier.

Announcements

- Support for Python 2 will be dropped in the next major TensorRT release. This means that TensorRT will no longer include wheels for Python 2, and Python samples will not work with Python 2. Ensure you migrate your application to Python version 3.

Known Issues

- The diagram in IRNNv2Layer is incorrect. This will be fixed in a future release.
- There is a known issue that graph capture may fail in some cases for `IExecutionContext::enqueue()` and `IExecutionContext::enqueueV2()`. For more information, refer to the documentation for IExecutionContext::enqueueV2(), including how to work around this issue.
- Some fusions are not enabled when the TensorRT static library is used. This means there is a performance loss of around 10% for networks like BERT and YOLO3 when linking with the static library compared to the dynamic library. The performance loss depends on precision used and batch size and it can be up to 60% in some cases.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit those weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.
- Convolution layers with dynamic shapes and large range of possible index dimensions in the profile have a known build time performance issue. This can be bypassed by using `IAlgorithmSelector` and disabling `cudnnConvolution` tactics.
- There is a known performance regression compared to TensorRT 7.1 for some networks dominated by FullyConnected with activation and bias operations:
  - up to 12% in FP32 mode. This will be fixed in a future release.
  - up to 10% in FP16 mode on Maxwell and Pascal GPUs.
- There is a known performance regression compared to TensorRT 7.1 for Convolution layers with kernel size greater than 5x5. For example, it can lead up to 35% performance regression of the VGG16 UFF model compared to TensorRT 7.1. This will be fixed in a future release.

- If the network contains an ElementWise layer, where one operand is a constant and the const rank is > 4, there is going to be a fusion to the Scale layer which doesn't support per element scale. The case can be seen for the Convolution and FullyConnected layers with bias where ONNX decomposes the bias to be ElementWise. To workaround this issue, add an Identity layer between the ElementWise and the const to prevent the fusion.
- Due to limitations with how requirements can be specified with the RPM application version supported by RHEL/CentOS 7.x, the cuBLAS development package from CUDA 11.1 is required when you are developing applications using TensorRT and CUDA 11.2. Your build environment can reference cuBLAS 11.2 without issues; this is only a packaging issue. This issue will be resolved with the next major CUDA version. Ubuntu does not have this limitation, therefore, cuBLAS 11.1 is not required for CUDA 11.2 development on those OS's.
- Some RNN networks such as Cortana with FP32 precision and batch size of 8 or higher have up to a 20% performance loss with CUDA 11.0 or higher compared to CUDA 10.2.
- You must have `libcublas.so.*` present on your system while running an application linked with the TensorRT static library. TensorRT now links to cuBLAS using `dlopen()` rather than at compiler link time for both the dynamic and static libraries. A solution to this problem will be worked out in a future release so that cuBLAS can be statically linked once again for applications which require the TensorRT static library.
- There is an up to 8% performance regression compared to TensorRT 7.1 for some networks with heavy FullyConnected operation on Nano.
- There is a known issue that TensorRT selects `kLINEAR` format when the user uses reformat-free I/O with vectorized formats and with input/output tensors which have only 3 dimensions. The workaround is to add an additional dimension to the tensors with size 1 to make them 4 dimensional tensors.
- When running networks such as Cortana, LSTM Peephole, MLP, and Faster RCNN you may observe a 5% to 16% performance regression on GA102 devices and a 7% to 36% performance regression on GA104 devices.
- When using TensorRT 7.2.3 with DLA, deconvolution layers with square kernels and strides greater than 23 are known to slow down build time by hours. Use `IBuilderConfig::setDeviceType(const ILayer* layer, DeviceType deviceType)` to run these layers on the GPU. [*JetPack issue*]

3.2. TensorRT Release 7.2.2

These are the TensorRT 7.2.2 release notes and are applicable to Windows and Linux x86 users.

These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

This release includes several fixes from the previous TensorRT 7.x.x release as well as the following additional changes. For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

- Added support for Python 3.8. The Linux tar packages now include TensorRT Python binding wheel files that support Python 3.8.

  Note: TensorFlow 1.15.x does not support Python 3.8. Continue to use an earlier Python version if you require UFF support. Updating the TensorRT samples to support TensorFlow 2.x will be done in a future release.

- Added the following debugging tools:

  Note: Although these tools are shipped with TensorRT, their utility extends beyond the TensorRT workflow.

  ONNX GraphSurgeon API Reference
  ONNX GraphSurgeon provides a convenient way to create and modify ONNX models. For more information, see ONNX GraphSurgeon API Reference.

  Polygraphy API Reference
  Polygraphy is a toolkit designed to assist in running and debugging deep learning models in various frameworks. For more information, refer to the Polygraphy API.

  PyTorch-Quantization Toolkit User Guide
  PyTorch-Quantization is a toolkit for training and evaluating PyTorch models with simulated quantization. Quantization can be added to the model automatically, or manually, allowing the model to be tuned for accuracy and performance. The quantized model can be exported to ONNX and imported to an upcoming version of TensorRT. For more information, refer to the PyTorch-Quantization Toolkit User Guide.

- Added instructions and a list of limitations for how to build the TensorRT samples using the TensorRT static libraries, including cuDNN and other CUDA libraries that are statically linked. For more information, refer to the Building Samples Using Static Libraries section in the *Working With TensorRT Samples*.
- Added a Quick Start Guide. This guide is a starting point for users who want to try out TensorRT; specifically, this document enables users to quickly deploy and run inference on a finished TensorRT engine.

Compatibility

- TensorRT 7.2.2 has been tested with the following:
  - cuDNN 8.0.5

- TensorFlow 1.15.3
- PyTorch 1.5.0
- ONNX 1.6.0

- This TensorRT release supports CUDA 10.2, 11.0 update 1, 11.1 update 1, and 11.2.

> **Note:** If you are developing an application that is being compiled with CUDA 11.2 or you are using CUDA 11.2 libraries to run your application, then you must install CUDA 11.1 using either the Debian/RPM packages or using a CUDA 11.1 tar/zip/exe package. NVRTC from CUDA 11.1 is a runtime requirement of TensorRT and must be present to run TensorRT applications. If you are using the network repo installation method, this additional step is not needed.

- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section in the *TensorRT Support Matrix*. Other semantically compatible releases of cuDNN and cuBLAS can be used, however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- TensorRT 7.2 only supports per-tensor quantization scales for both activations and weights in explicit precision mode. No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, refer to the Working With Explicit Precision Using C++ in the *TensorRT Developer Guide* for more information.
- Loops and `DataType::kBOOL` are not supported when the static TensorRT library is used.
- When using reformat-free I/O, the extent of a tensor in a vectorized dimension might not be a multiple of the vector length. Elements in a partially occupied vector that are not within the tensor are referred to here as *vector-padding*. For example:
  - On GPU
    - for input tensors, the application shall set vector-padding elements to zero.
    - for output tensors, the value of vector-padding elements is undefined. In a future release, TensorRT will support setting them to zero.
  - On DLA
    - for input tensors, vector-padding elements are ignored.
    - for output tensors, vector-padding elements are unmodified.
- When running INT8 networks on DLA using TensorRT, operations must be added to the same subgraph to reduce quantization errors across the subgraph of the network that runs on the DLA by allowing them to fuse and retain higher precision for intermediate results. Breaking apart the subgraph in order to inspect intermediate results by setting the tensors as network output tensors, can result in different levels of quantization errors due to these optimizations being disabled.

- There is a known issue that TensorRT selects `kLINEAR` format when the user uses reformat-free I/O with vectorized formats and with input/output tensors which have only 3 dimensions. The workaround is to add an additional dimension to the tensors with size 1 to make them 4 dimensional tensors.
- The `IExecutionContext` contains shared resources, therefore, calling `enqueue` or `enqueueV2` in from the same `IExecutionContext` object with different CUDA streams concurrently results in undefined behavior. To perform inference concurrently in multiple CUDA streams, use one `IExecutionContext` per CUDA stream.

Deprecated Features

The following features are deprecated in TensorRT 7.2.2:

- `IRNNLayer` was deprecated in TensorRT 4.0 and will be removed in TensorRT 8.0. `IRNNv2Layer` was deprecated in TensorRT 7.2.1 and will be removed in TensorRT 9.0. `IRNNv2Layer` has been deprecated in favor of the loop API, however, it is still available for backwards compatibility. For more information about the loop API, refer to the sampleCharRNN sample with the `--loop` option as well as the Working With Loops chapter in the *TensorRT Developer Guide*.
- We have deprecated the Caffe Parser and UFF Parser in TensorRT 7.0. They will be tested and functional in the next major release of TensorRT 8.0, however, we plan to remove the support in the subsequent major release. Ensure you migrate your workflow to use `tf2onnx`, `keras2onnx` or TensorFlow-TensorRT (TF-TRT) for deployment.

  If using UFF, ensure you migrate to the ONNX workflow through enablement of a plugin. ONNX workflow is not dependent on plugin enablement. For plugin enablement of a plugin on ONNX, refer to Estimating Depth with ONNX Models and Custom Layers Using NVIDIA TensorRT.

  - For TensorFlow to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorFlow, ONNX, and TensorRT.
  - For PyTorch to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorRT.

Fixed Issues

- If you had started with a clean system installation and you had not installed the CUDA Toolkit prior to installing the TensorRT samples, then you may of needed to manually install `cuda-nvcc-XX-Y` and `cuda-nvprof-XX-Y`, where `XX-Y` matches the CUDA major and minor version for your desired setup. Without these additional packages, you may have encountered compile errors while building the TensorRT samples. This issue has been fixed in this release.
- There was up to 23% performance regression on Volta GPUs for some RNN networks. This issue has been fixed in this release. *(not applicable for Jetson platforms)*

- There was a known accuracy issue of 3D U-Net networks on NVIDIA Ampere GPUs where TF32 mode is enabled by default. This issue has been fixed in this release.

Announcements

- Support for Python 2 will be dropped in a future TensorRT release. This means that TensorRT will no longer include wheels for Python 2, and Python samples will not work with Python 2. Ensure you migrate your application to Python version 3.

Known Issues

- The diagram in IRNNv2Layer is incorrect. This will be fixed in a future release.
- There is a known issue that graph capture may fail in some cases for `IExecutionContext::enqueue()` and `IExecutionContext::enqueueV2()`. For more information, refer to the documentation for `IExecutionContext::enqueueV2()`, including how to work around this issue.
- Some fusions are not enabled when the static library is used. This means there is a performance loss of around 10% for networks like BERT and YOLO3 when linking with the static library compared to the dynamic library. The performance loss depends on precision used and batch size and it can be up to 60% in some cases.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit those weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.
- Convolution layers with dynamic shapes and large range of possible index dimensions in the profile have a known build time performance issue. This can be bypassed by using `IAlgorithmSelector` and disabling `cudnnConvolution` tactics.
- There is a known performance regression compared to TensorRT 7.1 for some networks dominated by FullyConnected with activation and bias operations:
  - up to 12% in FP32 mode. This will be fixed in a future release.
  - up to 10% in FP16 mode on Maxwell and Pascal GPUs.
- There is a known performance regression compared to TensorRT 7.1 for Convolution layers with kernel size greater than 5x5. For example, it can lead up to 35% performance regression of the VGG16 UFF model compared to TensorRT 7.1. This will be fixed in a future release.
- If the network contains an ElementWise layer, where one operand is a constant and the const rank is > 4, there is going to be a fusion to the Scale layer which doesn't support per element scale. The case can be seen for the Convolution and FullyConnected layers with

bias where ONNX decomposes the bias to be ElementWise. To workaround this issue, add an Identity layer between the ElementWise and the const to prevent the fusion.

- Due to limitations with how requirements can be specified with the RPM version supported by RHEL/CentOS 7.x, the cuBLAS development package from CUDA 11.1 is required when you are developing applications using TensorRT and CUDA 11.2. Your build environment can reference cuBLAS 11.2 without issues; this is only a packaging issue. This issue will be resolved with future CUDA versions. Ubuntu does not have this limitation, therefore, cuBLAS 11.1 is not required for CUDA 11.2 development on those OS's.
- Some RNN networks such as Cortana with FP32 precision and batch size of 8 or higher have up to a 20% performance loss with CUDA 11.0 or higher compared to CUDA 10.2.
- You must have `libcublas.so.*` present on your system while running an application linked with the TensorRT static library. TensorRT now links to cuBLAS using `dlopen()` rather than at compiler link time for both the dynamic and static libraries. A solution to this problem will be worked out in a future release so that cuBLAS can be statically linked once again for applications which require the TensorRT static library.

3.3. TensorRT Release 7.2.1

These are the TensorRT 7.2.1 release notes and are applicable to Linux x86, Windows x64 and Linux ARM Server Base System Architecture (SBSA) users.

These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

This release includes several fixes from the previous TensorRT 7.x.x release as well as the following additional changes. For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

- Added support for CUDA 11.1 and GeForce devices with compute capability version 8.6.
- Added support for Linux ARM Server Base System Architecture (SBSA) users on Ubuntu 18.04.
- Added instructions for installing TensorRT from a `pip` wheel file. For step-by-step instructions, refer to the `pip` Wheel File Installation section in the *TensorRT Installation Guide*.

Compatibility

- TensorRT 7.2.1 has been tested with the following:
  - cuDNN 8.0.4
  - TensorFlow 1.15.3

- PyTorch 1.5.0
- ONNX 1.6.0

- This TensorRT release supports CUDA 10.2, 11.0 update 1, and 11.1.
- It is suggested that you use TensorRT with a software stack that has been tested; including cuDNN and cuBLAS versions as documented in the Features For Platforms And Software section in the *TensorRT Support Matrix*. Other semantically compatible releases of cuDNN and cuBLAS can be used, however, other versions may have performance improvements as well as regressions. In rare cases, functional regressions might also be observed.

Limitations

- TensorRT 7.2 only supports per-tensor quantization scales for both activations and weights in explicit precision mode. No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, refer to the Working With Explicit Precision Using C++ in the *TensorRT Developer Guide* for more information.
- Replace `IRNNLayer` and `IRNNv2Layer` with loops. `IRNNLayer` was deprecated in TensorRT 4.0 and will be removed in TensorRT 8.0. `IRNNv2Layer` was deprecated in TensorRT 7.2.1 and will be removed in TensorRT 9.0. Use the loop API to synthesize a recurrent subnetwork. For an example, see sampleCharRNN sample, method `SampleCharRNNLoop::addLSTMCell`. The loop API lets you express general recurrent networks instead of being limited to the prefabricated cells in `IRNNLayer` and `IRNNv2Layer`.
- When using reformat-free I/O, the extent of a tensor in a vectorized dimension might not be a multiple of the vector length. Elements in a partially occupied vector that are not within the tensor are referred to here as *vector-padding*. For example:
  - On GPU
    - for input tensors, the application shall set vector-padding elements to zero.
    - for output tensors, the value of vector-padding elements is undefined. In a future release, TensorRT will support setting them to zero.
  - On DLA
    - for input tensors, vector-padding elements are ignored.
    - for output tensors, vector-padding elements are unmodified.
- Loops and `DataType::kBOOL` are not supported when the static TensorRT library is used.
- When running INT8 networks on DLA using TensorRT, operations must be added to the same subgraph to reduce quantization errors across the subgraph of the network that runs on the DLA by allowing them to fuse and retain higher precision for intermediate results. Breaking apart the subgraph in order to inspect intermediate results by setting the tensors as network output tensors, can result in different levels of quantization errors due to these optimizations being disabled.

- There is a known issue that TensorRT selects `kLINEAR` format when the user uses reformat-free I/O with vectorized formats and with input/output tensors which have only 3 dimensions. The workaround is to add an additional dimension to the tensors with size 1 to make them 4 dimensional tensors.

Deprecated Features

The following features are deprecated in TensorRT 7.2.1:

- Documented the deprecation policy of TensorRT. For details, see TensorRT Deprecation Policy in the *TensorRT Developer Guide*.
- `IRNNLayer` was deprecated in TensorRT 4.0 and will be removed in TensorRT 8.0. `IRNNv2Layer` was deprecated in TensorRT 7.2.1 and will be removed in TensorRT 9.0. `IRNNv2Layer` has been deprecated in favor of the loop API, however, it is still available for backwards compatibility. For more information about the loop API, refer to the sampleCharRNN sample with the `--Iloop` option as well as the Working With Loops chapter in the *TensorRT Developer Guide*.
- We have deprecated the Caffe Parser and UFF Parser in TensorRT 7. They will be tested and functional in the next major release of TensorRT 8, however, we plan to remove the support in the subsequent major release. Ensure you migrate your workflow to use `tf2onnx`, `keras2onnx` or TensorFlow-TensorRT (TF-TRT) for deployment.

  If using UFF, ensure you migrate to the ONNX workflow through enablement of a plugin. ONNX workflow is not dependent on plugin enablement. For plugin enablement of a plugin on ONNX, refer to Estimating Depth with ONNX Models and Custom Layers Using NVIDIA TensorRT.

  - For TensorFlow to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorFlow, ONNX, and TensorRT.
  - For PyTorch to ONNX and then to TensorRT, refer to Speeding up Deep Learning Inference Using TensorRT.

Fixed Issues

- A symbol conflict between the cuBLAS static library and the TensorRT plugin static library has been resolved. The `Logger` class used internally by the TensorRT plugin library has been moved to a namespace to avoid symbol conflicts. You may experience unexpected crashes during initialization or when exiting your application if linking with TensorRT static libraries prior to this fix.
- There was a known performance regression on P100:

  - 30% regression on 3D networks like 3D U-Net in FP32 mode

  This issue has been fixed in this release. *(not applicable for Jetson platforms)*
- For Windows users with CUDA 11.0, some fusions were not enabled. This means there was a performance loss of around 10% - 60% for networks like BERT and YOLO3. The

performance loss depends on the precision used and batch size. This issue has been fixed in this release.

- There was up to a 10% performance regression for Inception V4 networks in FP32 mode on P100 and V100. This issue has been fixed in this release. *(not applicable for Jetson platforms)*
- MobileNetV1 and MobileNetV2 networks had up to a 14% performance regression in FP32 mode. This issue has been fixed in this release.

Announcements

- Support for Python 2 will be dropped in a future TensorRT release. This means that TensorRT will no longer include wheels for Python 2, and Python samples will not work with Python 2. Ensure you migrate your application to Python version 3.

Known Issues

- The diagram in IRNNv2Layer is incorrect. This will be fixed in a future release.
- There is a known issue that graph capture may fail in some cases for `IExecutionContext::enqueue()` and `IExecutionContext::enqueueV2()`. For more information, refer to the documentation for `IExecutionContext::enqueueV2()`, including how to work around this issue.
- There is up to 23% performance regression on Volta GPUs for some RNN networks. *(not applicable for Jetson platforms)*
- Some fusions are not enabled when the static library is used. This means there is a performance loss of around 10% for networks like BERT and YOLO3. The performance loss depends on precision used and batch size and it can be up to 60% in some cases.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit those weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.
- There is a known accuracy issue of 3D U-Net networks on NVIDIA Ampere GPUs where TF32 mode is enabled by default. To workaround this issue, TF32 mode can be disabled via TensorRT or by setting the environment variable `NVIDIA_TF32_OVERRIDE=0` when an engine is built. For more information and how to control TF32, see Enabling TF32 Inference Using C++ in the *TensorRT Developer Guide*.
- Convolution layers with dynamic shapes and large range of possible index dimensions in the profile have a known build time performance issue. This can be bypassed by using `IAlgorithmSelector` and disabling `cudnnConvolution` tactics.

- If you are starting with a clean system installation and you have not installed the CUDA Toolkit prior to installing the TensorRT samples, then you may need to manually install `cuda-nvcc-XX-Y` and `cuda-nvprof-XX-Y`, where `XX-Y` matches the CUDA major and minor version for your desired setup. Without these additional packages, you may encounter compile errors while building the TensorRT samples. These additional dependencies will be corrected in a future release.

3.4. TensorRT Release 7.2.0

ATTENTION:

This is the TensorRT 7.2.0 release notes. We recommend PowerPC users download the TensorRT 7.2.0 build for production use. For Linux and JetPack users, TensorRT 7.2.0 is a Release Candidate (RC). As an RC release, this is a Preview for early testing and feedback. For production use of TensorRT for Linux and JetPack users, we recommend downloading TensorRT 7.1.3. The RC release is subject to change based on ongoing performance tuning and functional testing. For feedback, submit a bug on the NVIDIA Developer website.

These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

This release includes several fixes from the previous TensorRT 7.x.x release as well as the following additional changes. For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.
FullyConnected Layer optimization

- Improved performance with Tensor Core in INT8 mode.
- TensorRT now uses cuBLASLt internally instead of cuBLAS. This decreases the overall runtime memory footprint. Users can revert to the old behavior by using the new `setTacticSources` API in `IBuilderConfig`.

Compatibility

- TensorRT 7.2.0 has been tested with the following:
  - cuDNN 8.0.2 for x86 and Jetson and 8.0.3 for PowerPC
  - TensorFlow 1.15.3
  - PyTorch 1.5.1
  - ONNX 1.6.0

- This TensorRT release supports CUDA 10.2 for Jetson and 11.0 update 1 for x86 and PowerPC.

Limitations

- TensorRT 7.2 only supports per-tensor quantization scales for both activations and weights in explicit precision mode. No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, refer to the Working With Explicit Precision Using C++ in the *TensorRT Developer Guide* for more information.
- When using reformat-free I/O, the extent of a tensor in a vectorized dimension might not be a multiple of the vector length. Elements in a partially occupied vector that are not within the tensor are referred to here as *vector-padding*. For example:
  - On GPU
    - for input tensors, the application shall set vector-padding elements to zero.
    - for output tensors, the value of vector-padding elements is undefined. In a future release, TensorRT will support setting them to zero.
  - On DLA
    - for input tensors, vector-padding elements are ignored.
    - for output tensors, vector-padding elements are unmodified.

Fixed Issues

- When using an RPM file on RedHat for a cuDNN installation, upgrading from cuDNN v7 to cuDNN v8 directly or indirectly via TensorRT 7.1.3 would cause installation errors. This issue has been fixed in the cuDNN 8.0.2 release.

Known Issues

- There is a known package dependency issue when installing the `python-libnvinfer` RPM package on RHEL/CentOS 8.x. You will encounter the following error:

```
- nothing provides python >= 2.7 needed by python-libnvinfer-7.2.0-1.cuda11.0.ppc64le
```

  Listed below are two options you can choose from to workaround this packaging issue:

  Option 1: Install the RPM package by ignoring the missing dependency.

```
# Install TensorRT and Python 2.x first
sudo yum install tensorrt python2
# Download the RPM package and install the package directly
sudo yum install yum-utils
yumdownloader python-libnvinfer
sudo rpm -Uvh --nodeps python-libnvinfer-*.rpm
```

  Option 2: Install the TensorRT Python bindings using the Python wheel file.

  An alternative to installing the RPM package for the Python bindings is to instead install the Python wheel file from the TAR package using `pip`. Refer to step 6 within the Tar File Installation section of the TensorRT Installation Guide.

The Python 3.x RPM packages are not affected by this dependency issue. This issue will be resolved in the next release.

(not applicable for Jetson platforms)

- There is a known performance regression on some RNN networks:
  - up to 12% on Pascal and Turing GPUs
  - up to 20% on Volta GPUs

  (not applicable for Jetson platforms)
- There is a known performance regression on P100:
  - 30% regression on 3D networks like 3D U-Net in FP32 mode

  (not applicable for Jetson platforms)
- There is up to a 10% performance regression for Inception V4 networks in FP32 mode on P100 and V100. *(not applicable for Jetson platforms)*
- The diagram in IRNNv2Layer is incorrect. This will be fixed in a future release.
- There is a known issue that graph capture may fail in some cases for `IExecutionContext::enqueue()` and `IExecutionContext::enqueueV2()`. For more information, refer to the documentation for `IExecutionContext::enqueueV2()`, including how to work around this issue.
- On PowerPC, some RNN networks have up to a 15% performance regression compared to TensorRT 7.0. *(not applicable for Jetson platforms)*
- MobileNetV1 and MobileNetV2 networks have up to a 14% performance regression in FP32 mode.
- Some fusions are not enabled in the following cases:
  - Windows with CUDA 11.0
  - When the static library is used

  This means there is a performance loss of around 10% for networks like BERT and YOLO3. The performance loss depends on precision used and batch size and it can be up to 60% in some cases.
- Loops and `DataType::kBOOL` are not supported when the static TensorRT library is used.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit the weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.

- When using concat on the DLA, all inputs to concat must be exact multiples of the vector size (16 for FP16, 32 for INT8). This will be fixed in a future release of TensorRT.

3.5. TensorRT Release 7.1.3

ATTENTION:

This is the TensorRT 7.1.3 GA release notes. For production use of TensorRT, we recommend using the TensorRT 7.1.3 build for CUDA 10.2. The CUDA 11.0 RC build is a Preview release for early testing and feedback on NVIDIA A100. This release is subject to change based on ongoing performance tuning and functional testing. For feedback, submit a bug on the NVIDIA Developer website.

These release notes are applicable to JetPack users of TensorRT unless appended specifically with *(not applicable for Jetson platforms)*.

This release includes several fixes from the previous TensorRT 7.x.x release as well as the following additional changes. For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

Working with empty tensors

TensorRT supports empty tensors. A tensor is an empty tensor if it has one or more dimensions with length zero. Zero-length dimensions usually get no special treatment. If a rule works for a dimension of length `L` for an arbitrary positive value of `L`, it usually works for `L=0` too. For more information, see Working With Empty Tensors in the *TensorRT Developer Guide*.

Builder layer timing cache

The layer timing cache will cache the layer profiling information during the builder phase. If there are other layers with the same input/output tensor configuration and layer params, then the TensorRT builder will skip profiling and reuse the cached result for the repeated layers. Models with many repeated layers (for example, BERT, WaveGlow, etc...) will see a significant speedup in builder time. The builder flag `kDISABLE_TIMING_CACHE` can be set if you want to disable this feature. For more information, see Builder Layer Timing Cache in the *TensorRT Developer Guide* and Initializing The Engine in the *Best Practices For TensorRT Performance*.

Pointwise fusion based on code generation

Pointwise fusion was introduced in TensorRT 6.0.1 to fuse multiple adjacent pointwise layers into one single layer. In this release, its implementation has been updated to use code generation and runtime compilation to further improve performance. The code generation and runtime compilation happen during execution plan building. For more information, see the TensorRT Best Practices Guide.

Dilation support for deconvolution

`IDeconvolutionLayer` now supports a dilation parameter. This is accessible through the C++ API, Python API, and the ONNX parser (see `ConvTranspose`). For more information, `IDeconvolutionLayer` in the *TensorRT Developer Guide*.

Selecting FP16 and INT8 kernels

TensorRT supports Mixed Precision Inference with FP32, FP16, or INT8 as supported precisions. Depending on the hardware support, you can choose to enable either of the above precision to accelerate inference. You can also choose to execute `trtexec` with the "`--best`" option directly, which would enable all supported precisions for inference resulting in best performance. For more information, see Mixed Precision in the *Best Practices For TensorRT Performance*.

Calibration with dynamic shapes

INT8 calibration with dynamic shapes supports the same functionality as a standard INT8 calibrator but for networks with dynamic shapes. You will need to provide a calibration optimization profile that would be used to set the dimensions for calibration. If a calibration optimization profile is not set, the first network optimization profile will be used as a calibration optimization profile. For more information, see INT8 Calibration With Dynamic Shapes in the *TensorRT Developer Guide*.

Algorithm selection

Algorithm selection provides a mechanism to select and report algorithms for different layers in a network. This can also be used to deterministically build TensorRT engine or to reproduce the same implementations for layers in the engine. For more information, see the Algorithm Selection and Determinism And Reproducibility In The Builder topics in the *TensorRT Developer Guide*.

INT8 calibration

The Legacy class `IInt8LegacyCalibrator` is un-deprecated. It is provided as a fallback option if the other calibrators yield poor results. A new `kCALIBRATION_BEFORE_FUSION` has been added which allows calibration before fusion. For more information, see INT8 Calibration Using C++ in the *TensorRT Developer Guide*.

Quantizing and dequantizing scale layers

A quantizing scale layer can be specified as a scale layer with output precision type of INT8. Similarly, a dequantizing scale layer can be specified as a scale layer with output precision type of FP32. Networks must be created with Explicit Precision mode to use these layers. Quantizing and dequantizing (QDQ) scale layers only support per-tensor quantization scales i.e. a single scale per tensor. Also, No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, see Working With Explicit Precision Using C++ in the *TensorRT Developer Guide*.

Samples compilation

A new Makefile option `TRT_STATIC=1` has been added which allows you to build the TensorRT samples with TensorRT and most dependent libraries statically linked into the sample binary.

Group normalization plugin

A new group normalization plugin has been added. For details on group normalization, refer to the Group Normalization paper.

TF32 support

TF32 is enabled by default for `DataType::kFLOAT`. On the NVIDIA Ampere architecture-based A100/GA100 GPU, TF32 can speed up networks using FP32, typically with no loss of accuracy. It combines FP32 dynamic range and format with FP16 precision. TF32 can be disabled via TensorRT or by setting the environment variable `NVIDIA_TF32_OVERRIDE=0` when an engine is built. For more information and how to control TF32, see Enabling TF32 Inference Using C++ in the *TensorRT Developer Guide*. *(not applicable for Jetson platforms)*

New plugins

Added new plugins for common operators in the BERT model, including embedding layer normalization, skip layer normalization and multi-head attention.

`embLayerNormPlugin`

This plugin performs the following two tasks:

- Embeds an input sequence consisting of token IDs and segment IDs. This consists of token embedding lookup, segment embedding lookup, adding positional embeddings and finally, layer normalization.
- Preprocesses input masks that are used to mark valid input tokens in sequences that are padded to the target sequence length. It assumes contiguous input masks and encodes the masks as a single number denoting the number of valid elements. This plugin supports FP32 mode and FP16 mode.

`skipLayerNormPlugin`

This plugin adds a residual tensor, and applies layer normalization, meaning, transforming the mean and standard deviation to beta and gamma, respectively. Optionally, it can add a bias vector before layer normalization. This plugin supports FP32

mode, FP16 mode, and INT8 mode. It may bring a negative impact on the end-to-end prediction accuracy when running under INT8 mode.

`bertQKVToContextPlugin`

This plugin takes query, key, and value tensors and computes scaled multi-head attention, that is to compute scaled dot product attention scores `SoftMax(K' * Q / sqrt(HeadSize))` and return values weighted by these attention scores. This plugin supports FP32 mode, FP16 mode, and INT8 mode. It is optimized for sequence lengths 128 and 384, and INT8 mode is only available for those sequence lengths.

These plugins only support GPUs with compute capability >= 7.0. For more information about these new BERT-related plugins, see TensorRT Open Source Plugins.

New sample

sampleAlgorithmSelector

sampleAlgorithmSelector shows an example of how to use the algorithm selection API based on sampleMNIST. This sample demonstrates the usage of `IAlgorithmSelector` to deterministically build TensorRT engines. It also shows the usage of `IAlgorithmSelector::selectAlgorithms` to define heuristics for selection of algorithms. For more information, see Algorithm Selection in the *TensorRT Developer Guide*, Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT in the *TensorRT Samples Support Guide*.

onnx_packnet

onnx_packnet is a Python sample which uses TensorRT to perform inference with the PackNet network. PackNet is a self-supervised monocular depth estimation network used in autonomous driving. For more information, refer to TensorRT Inference Of ONNX Models With Custom Layers in the *TensorRT Sample Support Guide*.

Multi-Instance GPU (MIG)

Multi-instance GPU, or MIG, is a new feature in NVIDIA Ampere GPU architecture that enables user-directed partitioning of a single GPU into multiple smaller GPUs. This improves GPU utilization by enabling the GPU to be shared effectively by parallel compute workloads on bare metal, GPU pass through, or on multiple vGPUs. For more information, refer to Working With Multi-Instance GPU in the *TensorRT Developer Guide*. *(not applicable for Jetson platforms)*

Improved ONNX Resize operator support

The ONNX resize modes `asymmetric`, `align_corners`, `half_pixel`, and `pytorch_half_pixel` are now supported. For more information on these resize modes, see the ONNX Resize Operator Specification.

Compatibility

- TensorRT 7.1.3 has been tested with the following:

- cuDNN 8.0.0 Preview
- TensorFlow 1.15.2
- PyTorch 1.4.0

> **Note:** Due to a known issue in PyTorch (#32983), you need to use the CPU version of PyTorch if you intend to load it with TensorRT; just as the TensorRT samples do.

- ONNX 1.6.0

- This TensorRT release supports CUDA 10.2 and CUDA 11.0 RC.

Limitations

- TensorRT 7.1 only supports per-tensor quantization scales for both activations and weights in explicit precision mode. No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, refer to the Working With Explicit Precision Using C++ in the *TensorRT Developer Guide* for more information.

Deprecated Features

The following features are deprecated in TensorRT 7.1.3:

- The fc_plugin_caffe_mnist Python sample has been deprecated. The FCPlugin is not selected by fc_plugin_caffe_mnist which was intended to demonstrate its usage. This is because there is no default importer for FCPlugin in the Caffe parser.
- Python 2.7 support has been deprecated. A warning will be emitted when you import the TensorRT bindings for Python 2.7. You should update your application to support Python 3.x to prevent issues with future TensorRT releases. In addition, the legacy Python bindings have been removed. You will need to migrate your application to the new Python bindings if you haven't done so already. Refer to the Python Migration Guide for more information.
- Support for CUDA Compute Capability version 3.0 has been removed. Support for CUDA Compute Capability versions 5.0 and lower may be removed in a future release. Specifically:

CUDA Compute Capability Version	Status
Maxwell SM 5.0 (2014-2017): - GM10X - GeForce 745 - GM10X - GeForce 750 - GM10X - GeForce 830 - GM10X - GeForce 840 - Quadro K620 - Quadro K1200 - Quadro K2200	Supported

CUDA Compute Capability Version	Status
▶ M5XX ▶ M6XX ▶ M1XXX ▶ M2000	
Kepler SM 3.7 (2014): ▶ GK210 - K8	Deprecated
Kepler SM 3.5 (2013): ▶ GK110 - K20 ▶ GeForce GTX 780 family ▶ GTX Titan	Deprecated
Kepler SM 3.0 (2012): ▶ GK10X GPUs ▶ GeForce 600 series ▶ K10 ▶ GRID K1/K2 ▶ Quadro K series	Removed

- Many methods of class `IBuilder` have been deprecated. The following table shows deprecated methods of class `IBuilder` that have replacements in `IBuilder`:

Deprecated `IBuilder` Method	`IBuilder` Replacement
`createNetwork()`	`createNetworkV2(0)`
`buildCudaEngine(`*network*`)`	`buildEngineWithConfig(`*network*`,`*config*`)`
`reset(`*network*`)`	`reset()`

The next table shows the deprecated methods of `IBuilder` that have direct equivalents in class `IBuilderConfig` with the same name.

Deprecated `IBuilder` Methods with Direct Equivalents in `IBuilderConfig`
▶ `setMaxWorkspaceSize` ▶ `getMaxWorkspaceSize`
`setInt8Calibrator`
▶ `setDeviceType` ▶ `getDeviceType`

Deprecated `IBuilder` Methods with Direct Equivalents in `IBuilderConfig`
▶ isDeviceTypeSet
▶ resetDeviceType ▶ setDefaultDeviceType ▶ getDefaultDeviceType
canRunOnDLA
▶ setDLACore ▶ getDLACore
▶ setEngineCapability ▶ getEngineCapability

Timing methods in `IBuilder` also have replacements in `IBuilderConfig`, with new names.

Deprecated `IBuilder` Method	Replacement In `IBuilderConfig`
setMinFindIterations	setMinTimingIterations
getMinFindIterations	getMinTimingIterations
setAverageFindIterations	setAvgTimingIterations
getAverageFindIterations	getAvgTimingIterations

Finally, some `IBuilder` methods related to boolean properties have been replaced with methods for setting/getting flags. For example, these calls on an `IBuilder`:

```
builder.setHalf2Mode(true);
builder.setInt8Mode(false);
```

can be replaced with these calls on a `IBuilderConfig`:

```
config.setFlag(BuilderFlag::kFP16);
config.clearFlag(BuilderFlag::kINT8);
```

The following table lists the deprecated methods and the corresponding flag.

Deprecated `IBuilder` Method	Corresponding Flag
▶ setHalf2Mode ▶ setFp16Mode ▶ getHalf2Mode ▶ getFp16Mode	BuilderFlag::kFP16
▶ setInt8Mode ▶ getInt8Mode	BuilderFlag::kINT8

Deprecated `IBuilder` Method	Corresponding Flag
`setDebugSync`	`BuilderFlag::kDEBUG`
▶ `setRefittable` ▶ `getRefittable`	`BuilderFlag::kREFIT`
▶ `setStrictTypeConstraints` ▶ `getStrictTypeConstraints`	`BuilderFlag::kSTRICT_TYPES`
`allowGPUFallback`	`BuilderFlag::kGPU_FALLBACK`

- The `INvPlugin` creator function has been deprecated since TensorRT 5.1.x and has now been fully removed. We recommend that users upgrade their plugins to one of the later plugin interfaces, refer to Extending TensorRT With Custom Layers section in the *TensorRT Developer Guide* for more information.

Fixed Issues

- Fixed memory leaks in engine serialization when UFF models are used.
- Fixed a crash during engine build for networks with RNNv2 on Windows.
- Statically linking with TensorRT library resulted in segfault in certain cases. The issue is now fixed.
- Fixed multiple bugs related to dynamic shapes, specifically:
  - padding modes for convolution and deconvolution,
  - engines with multiple optimization profiles, and
  - empty tensors (tensors with zero volume).

Announcements

- Boolean shape tensors now supported:
  - `IElementwiseLayer` with `kLESS`, `kEQUAL`, `kGREATER`, `kAND`, `kOR`, `kXOR` can operate on shape tensors.
  - `ISelectLayer` can operate on shape tensors.
  - `IUnaryLayer` with `kNOT` is not supported for shape tensors.
- NVIDIA TensorRT Inference Server has been renamed to NVIDIA Triton Inference Server. For more information, refer to the Triton Inference Server documentation.

Known Issues

- In the CUDA 11.0 RC release, there is a known performance regression on some RNN networks:
  - up to 50% on Turing GPUs

- up to 12% on Pascal and Volta GPUs
- There is known performance regression between 30-80% for networks like ResNet-50 and MobileNet when run in FP16 mode on SM50 devices.
- The Windows library size is 600 MB bigger than the Linux library size. This will be fixed in the next release.
- Static compiling of samples with the CentOS7 CUDA 11.0 RC build fails.
- There is a known performance regressions on P100:
  - 50-100% regression on 3D networks like 3D U-Net
  - 17% on Xception in FP16 mode
- There is a known performance regression for Inception V3 and V4 networks in FP32 mode:
  - up to 60% on V100
  - up to 15% on RTX6000
- Some fusions are not enabled on Windows with CUDA 11. This would mean performance loss of around 10% for networks like YOLO3.
- The diagram in IRNNv2Layer is incorrect. This will be fixed in a future release.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit the weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.
- Some fusions are not enabled in the following cases:
  - Windows with CUDA 11
  - When the static library is used

  This means there is a performance loss of around 10% for networks like BERT and YOLO3. The performance loss depends on precision used and batch size and it can be up to 60% in some cases.
- Loops and `DataType::kBOOL` are not supported when the static TensorRT library is used.
- There is an error in the `config.py` file included in the `sampleUffFasterRCNN` sample. Specifically, line 34 in the config file should be changed from: `dynamic_graph.remove('input_2')` to`dynamic_graph.remove(dynamic_graph.find_nodes_by_name('input_2'))`
- *Updated: June 25, 2020*

  When using an RPM file on RedHat for installation, installing cuDNN v8 directly or via TensorRT 7.1.3 will enable users to build their application with cuDNN v8. However, in

order for the user to compile an application with cuDNN v7 after cuDNN v8 is installed, the user will need to perform the following steps:

1. Issue `sudo mv /usr/include/cudnn.h /usr/include/cudnn_v8.h`.
2. Issue `sudo ln -s /etc/alternatives/libcudnn /usr/include/cudnn.h`.
3. Switch to cuDNN v7 by issuing `sudo update-alternatives --config libcudnn` and choose cuDNN v7 from the list.

Steps 1 and 2 are required for the user to be able to switch between v7 and v8 installations. After steps 1 and 2 are performed once, step 3 can be used repeatedly and the user can choose the appropriate cuDNN version to work with. For more information, refer to the Installing From An RPM File and Upgrading From v7 To v8 sections in the *cuDNN Installation Guide*.

3.6. TensorRT Release 7.1.2 Release Candidate (RC)

These are the TensorRT 7.1.2 Release Candidate (RC) release notes and are applicable to data center and workstation Linux users. This release includes several fixes from the previous TensorRT 7.x.x release as well as the following additional changes. These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

INT8 calibration

The Legacy class `IInt8LegacyCalibrator` is un-deprecated. It is provided as a fallback option if the other calibrators yield poor results. A new `kCALIBRATION_BEFORE_FUSION` has been added which allows calibration before fusion. For more information, see INT8 Calibration Using C++ in the *TensorRT Developer Guide*.

Quantizing and dequantizing scale layers

A quantizing scale layer can be specified as a scale layer with output precision type of INT8. Similarly, a dequantizing scale layer can be specified as a scale layer with output precision type of FP32. Networks must be created with Explicit Precision mode to use these layers. Quantizing and dequantizing (QDQ) scale layers only support per-tensor quantization scales i.e. a single scale per tensor. Also, No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, see Working With Explicit Precision Using C++ in the *TensorRT Developer Guide*.

Samples compilation

A new Makefile option `TRT_STATIC=1` has been added which allows you to build the TensorRT samples with TensorRT and most dependent libraries statically linked into the sample binary.

Group normalization plugin

A new group normalization plugin has been added. For details on group normalization, refer to the Group Normalization paper.

TF32 support

TF32 is enabled by default for `DataType::kFLOAT`. On the NVIDIA Ampere architecture-based A100/GA100 GPU, TF32 can speed up networks using FP32, typically with no loss of accuracy. It combines FP32 dynamic range and format with FP16 precision. TF32 can be disabled via TensorRT or by setting the environment variable `NVIDIA_TF32_OVERRIDE=0` when an engine is built. For more information and how to control TF32, see Enabling TF32 Inference Using C++ in the *TensorRT Developer Guide*. *(not applicable for Jetson platforms)*

New plugins

Added new plugins for common operators in the BERT model, including embedding layer normalization, skip layer normalization and multi-head attention.

`embLayerNormPlugin`

This plugin performs the following two tasks:

- Embeds an input sequence consisting of token IDs and segment IDs. This consists of token embedding lookup, segment embedding lookup, adding positional embeddings and finally, layer normalization.
- Preprocesses input masks that are used to mark valid input tokens in sequences that are padded to the target sequence length. It assumes contiguous input masks and encodes the masks as a single number denoting the number of valid elements. This plugin supports FP32 mode and FP16 mode.

`skipLayerNormPlugin`

This plugin adds a residual tensor, and applies layer normalization, meaning, transforming the mean and standard deviation to beta and gamma, respectively. Optionally, it can add a bias vector before layer normalization. This plugin supports FP32 mode, FP16 mode, and INT8 mode. It may bring a negative impact on the end-to-end prediction accuracy when running under INT8 mode.

`bertQKVToContextPlugin`

This plugin takes query, key, and value tensors and computes scaled multi-head attention, that is to compute scaled dot product attention scores `SoftMax(K' * Q / sqrt(HeadSize))` and return values weighted by these attention scores. This plugin supports FP32 mode, FP16 mode, and INT8 mode. It is optimized for sequence lengths 128 and 384, and INT8 mode is only available for those sequence lengths.

These plugins only support GPUs with compute capability >= 7.0. For more information about these new BERT-related plugins, see TensorRT Open Source Plugins.

Compatibility

- TensorRT 7.1.2 has been tested with the following:
  - cuDNN 8.0.0 Preview
  - TensorFlow 1.15.2
  - PyTorch 1.4.0
  - ONNX 1.6.0
- This TensorRT release supports CUDA 10.2 and 11.0 RC.
- Linux x86

Limitations

- TensorRT 7.1 only supports per-tensor quantization scales for both activations and weights in explicit precision mode. No shift weights are allowed for the QDQ scale layer as only symmetric quantization is supported. For more information, refer to the Working With Explicit Precision Using C++ in the *TensorRT Developer Guide* for more information.

Deprecated Features

The following features are deprecated in TensorRT 7.1.2:

- The fc_plugin_caffe_mnist Python sample has been deprecated. The FCPlugin is not selected by fc_plugin_caffe_mnist which was intended to demonstrate its usage. This is because there is no default importer for FCPlugin in the Caffe parser.

Announcements

- NVIDIA TensorRT Inference Server has been renamed to NVIDIA Triton Inference Server. For more information, refer to the Triton Inference Server documentation.

Known Issues

- There is a known issue that graph capture may fail in some cases for `IExecutionContext::enqueue()` and `IExecutionContext::enqueueV2()`. For more information, refer to the documentation for `IExecutionContext::enqueueV2()`, including how to work around this issue.
- There is a known ~40% performance regression on 3D networks like 3D Unet.
- There is a known ~50% performance regression on LSTM autoencoder with `BS=8`.
- There is a minor performance regression across a variety of networks that will be fixed in TensorRT 7.1.x GA.
- The diagram in IRNNv2Layer is incorrect. This will be fixed in TensorRT 7.1.x GA.
- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any

attempt to refit the weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.

- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.

3.7. TensorRT Release 7.1.0 Early Access (EA)

These are the TensorRT 7.1.0 Early Access (EA) release notes and are applicable to NVIDIA® Jetson™ Linux for Tegra™ users. This release includes several fixes from the previous TensorRT 6.0.0 and later releases as well as the following additional changes. These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

This preview release is for early testing and feedback, therefore, for production use of TensorRT, continue to use TensorRT 7.0.0.

For previous TensorRT documentation, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

Working with empty tensors

TensorRT supports empty tensors. A tensor is an empty tensor if it has one or more dimensions with length zero. Zero-length dimensions usually get no special treatment. If a rule works for a dimension of length `L` for an arbitrary positive value of `L`, it usually works for `L=0` too. For more information, see Working With Empty Tensors in the *TensorRT Developer Guide*.

Builder layer timing cache

The layer timing cache will cache the layer profiling information during the builder phase. If there are other layers with the same input/output tensor configuration and layer params, then the TensorRT builder will skip profiling and reuse the cached result for the repeated layers. Models with many repeated layers (for example, BERT, WaveGlow, etc...) will see a significant speedup in builder time. The builder flag `kDISABLE_TIMING_CACHE` can be set if you want to disable this feature. For more information, see Builder Layer Timing Cache in the *TensorRT Developer Guide* and Initializing The Engine in the *Best Practices For TensorRT Performance*.

Pointwise fusion based on code generation

Pointwise fusion was introduced in TensorRT 6.0.1 to fuse multiple adjacent pointwise layers into one single layer. In this release, its implementation has been updated to use

code generation and runtime compilation to further improve performance. The code generation and runtime compilation happen during execution plan building. For more information, see the TensorRT Best Practices Guide.

Dilation support for deconvolution

`IDeconvolutionLayer` now supports a dilation parameter. This is accessible through the C++ API, Python API, and the ONNX parser (see `ConvTranspose`). For more information, `IDeconvolutionLayer` in the *TensorRT Developer Guide*.

Selecting FP16 and INT8 kernels

TensorRT supports Mixed Precision Inference with FP32, FP16, or INT8 as supported precisions. Depending on the hardware support, you can choose to enable either of the above precision to accelerate inference. You can also choose to execute `trtexec` with the "`--best`" option directly, which would enable all supported precisions for inference resulting in best performance. For more information, see Mixed Precision in the *Best Practices For TensorRT Performance*.

Calibration with dynamic shapes

INT8 calibration with dynamic shapes supports the same functionality as a standard INT8 calibrator but for networks with dynamic shapes. You will need to provide a calibration optimization profile that would be used to set the dimensions for calibration. If a calibration optimization profile is not set, the first network optimization profile will be used as a calibration optimization profile. For more information, see INT8 Calibration With Dynamic Shapes in the *TensorRT Developer Guide*.

Algorithm selection

Algorithm selection provides a mechanism to select and report algorithms for different layers in a network. This can also be used to deterministically build TensorRT engine or to reproduce the same implementations for layers in the engine. For more information, see the Algorithm Selection and Determinism And Reproducibility In The Builder topics in the *TensorRT Developer Guide*.

New sample

sampleAlgorithmSelector shows an example of how to use the algorithm selection API based on sampleMNIST. This sample demonstrates the usage of `IAlgorithmSelector` to deterministically build TensorRT engines. It also shows the usage of `IAlgorithmSelector::selectAlgorithms` to define heuristics for selection of algorithms. For more information, see Algorithm Selection in the *TensorRT Developer Guide*, Algorithm Selection API Usage Example Based On sampleMNIST In TensorRT in the *TensorRT Samples Support Guide*.

Compatibility

- TensorRT 7.1.0 has been tested with the following:
  - cuDNN 8.0.0 Preview

- TensorFlow 1.15.2
- PyTorch 1.4.0
- ONNX 1.6.0

- This TensorRT release supports CUDA 10.2.
- JetPack 4.4

Deprecated Features

The following features are deprecated in TensorRT 7.1.0:

- Python 2.7 support has been deprecated. A warning will be emitted when you import the TensorRT bindings for Python 2.7. You should update your application to support Python 3.x to prevent issues with future TensorRT releases. In addition, the legacy Python bindings have been removed. You will need to migrate your application to the new Python bindings if you haven't done so already. Refer to the Python Migration Guide for more information.
- Support for CUDA Compute Capability version 3.0 has been removed. Support for CUDA Compute Capability versions 5.0 and lower may be removed in a future release. Specifically:

CUDA Compute Capability Version	Status
Maxwell SM 5.0 (2014-2017): - GM10X - GeForce 745 - GM10X - GeForce 750 - GM10X - GeForce 830 - GM10X - GeForce 840 - Quadro K620 - Quadro K1200 - Quadro K2200 - M5XX - M6XX - M1XXX - M2000	Supported
Kepler SM 3.7 (2014): - GK210 - K8	Deprecated
Kepler SM 3.5 (2013): - GK110 - K20 - GeForce GTX 780 family - GTX Titan	Deprecated

CUDA Compute Capability Version	Status
Kepler SM 3.0 (2012): ▶ GK10X GPUs ▶ GeForce 600 series ▶ K10 ▶ GRID K1/K2 ▶ Quadro K series	Removed

- Many methods of class `IBuilder` have been deprecated. The following table shows deprecated methods of class `IBuilder` that have replacements in `IBuilder`:

Deprecated `IBuilder` Method	`IBuilder` Replacement
`createNetwork()`	`createNetworkV2(0)`
`buildCudaEngine(`*network*`)`	`buildEngineWithConfig(`*network*`,`*config*`)`
`reset(`*network*`)`	`reset()`

The next table shows the deprecated methods of `IBuilder` that have direct equivalents in class `IBuilderConfig` with the same name.

Deprecated `IBuilder` Methods with Direct Equivalents in `IBuilderConfig`
▶ `setMaxWorkspaceSize` ▶ `getMaxWorkspaceSize`
`setInt8Calibrator`
▶ `setDeviceType` ▶ `getDeviceType` ▶ `isDeviceTypeSet`
▶ `resetDeviceType` ▶ `setDefaultDeviceType` ▶ `getDefaultDeviceType`
`canRunOnDLA`
▶ `setDLACore` ▶ `getDLACore`
▶ `setEngineCapability` ▶ `getEngineCapability`

Timing methods in `IBuilder` also have replacements in `IBuilderConfig`, with new names.

Deprecated `IBuilder` Method	Replacement In `IBuilderConfig`
`setMinFindIterations`	`setMinTimingIterations`
`getMinFindIterations`	`getMinTimingIterations`
`setAverageFindIterations`	`setAvgTimingIterations`
`getAverageFindIterations`	`getAvgTimingIterations`

Finally, some `IBuilder` methods related to boolean properties have been replaced with methods for setting/getting flags. For example, these calls on an `IBuilder`:

```
builder.setHalf2Mode(true);
builder.setInt8Mode(false);
```

can be replaced with these calls on a `IBuilderConfig`:

```
config.setFlag(BuilderFlag::kFP16);
config.clearFlag(BuilderFlag::kINT8);
```

The following table lists the deprecated methods and the corresponding flag.

Deprecated `IBuilder` Method	Corresponding Flag
▶ `setHalf2Mode` ▶ `setFp16Mode` ▶ `getHalf2Mode` ▶ `getFp16Mode`	`BuilderFlag::kFP16`
▶ `setInt8Mode` ▶ `getInt8Mode`	`BuilderFlag::kINT8`
`setDebugSync`	`BuilderFlag::kDEBUG`
▶ `setRefittable` ▶ `getRefittable`	`BuilderFlag::kREFIT`
▶ `setStrictTypeConstraints` ▶ `getStrictTypeConstraints`	`BuilderFlag::kSTRICT_TYPES`
`allowGPUFallback`	`BuilderFlag::kGPU_FALLBACK`

- The `INvPlugin` creator function has been deprecated since TensorRT 5.1.x and has now been fully removed. We recommend that users upgrade their plugins to one of the later plugin interfaces, refer to Extending TensorRT With Custom Layers section in the *TensorRT Developer Guide* for more information.

Fixed Issues

- DLA has restrictions on usage that were previously undocumented. Some programs that might have worked, but violated these restrictions, are now expected to fail at build time. For more information, see Restrictions With DLA and FAQs in the *TensorRT Developer Guide*.

Announcements

- NVIDIA TensorRT Inference Server has been renamed to NVIDIA Triton Inference Server. For more information, refer to the Triton Inference Server documentation.

Known Issues

- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so any attempt to refit the weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.

3.8. TensorRT Release 7.0.0

These are the TensorRT 7.0.0 release notes for Linux and Windows users. This release includes fixes from the previous TensorRT 6.0.1 release as well as the following additional changes. These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

For previous TensorRT release notes, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

Working with loops

TensorRT supports loop-like constructs, which can be useful for recurrent networks. TensorRT loops support scanning over input tensors, recurrent definitions of tensors, and both "scan outputs" and "last value" outputs. For more information, see Working With Loops in the *TensorRT Developer Guide*.

ONNX parser with dynamic shapes support

The ONNX parser supports full-dimensions mode only. Your network definition must be created with the `explicitBatch` flag set. For more information, see Importing An ONNX

Model Using The C++ Parser API and Working With Dynamic Shapes in the *TensorRT Developer Guide* for more information.

TensorRT container with OSS

The TensorRT monthly container release now contains pre-built binaries from the TensorRT Open Source Repository. For more information, refer to the monthly released TensorRT Container Release Notes starting in 19.12+.

BERT INT8 and mixed precision optimizations

Some GEMM layers are now followed by GELU activation in the BERT model. Since TensorRT doesn't have IMMA GEMM layers, you can implement those GEMM layers in the BERT network with either `IConvolutionLayer` or `IFullyConnectedLayer` layers depending on what precision you require. For example, you can leverage `IConvolutionLayer` with `H == W == 1` (CONV1x1) to implement a FullyConnected operation and leverage IMMA math under INT8 mode. TensorRT supports the fusion of Convolution/FullyConnected and GELU. For more information, refer to TensorRT Best Practices Guide and Adding Custom Layers Using The C++ API in the *TensorRT Developer Guide*.

Working with Quantized Networks

TensorRT now supports quantized models trained with Quantization Aware Training. Support is limited to symmetrically quantized models, meaning `zero_point = 0` using `QuantizeLinear` and `DequantizeLinear` ONNX ops. For more information, see Working With Quantized Networks in the *TensorRT Developer Guide* and QDQ Fusions in the *Best Practices For TensorRT Performance Guide*.

New layers

`IFillLayer`

The `IFillLayer` is used to generate an output tensor with the specified mode. For more information, see the C++ class `IFillLayer` or the Python class `IFillLayer`.

`IIteratorLayer`

The `IIteratorLayer` enables a loop to iterate over a tensor. A loop is defined by loop boundary layers. For more information, see the C++ class `IIteratorLayer` or the Python class `IIteratorLayer` and Working With Loops in the *TensorRT Developer Guide*.

`ILoopBoundaryLayer`

Class `ILoopBoundaryLayer` defines a virtual method `getLoop()` that returns a pointer to the associated `ILoop`. For more information, see the C++ class `ILoopBoundaryLayer` or the Python class `ILoopBoundaryLayer` and Working With Loops in the *TensorRT Developer Guide*.

`ILoopOutputLayer`

The `ILoopOutputLayer` specifies an output from the loop. For more information, see the C++ class `ILoopOutputLayer` or the Python class `ILoopOutputLayer` and Working With Loops in the *TensorRT Developer Guide*.

IParametricReluLayer

The IParametricReluLayer represents a parametric ReLU operation, meaning, a leaky ReLU where the slopes for x < 0 can be different for each element. For more information, see the C++ class IParametricReluLayer or the Python class IParametricReluLayer.

IRecurrenceLayer

The IRecurrenceLayer specifies a recurrent definition. For more information, see the C++ class IRecurrenceLayer or the Python class IRecurrenceLayer and Working With Loops in the *TensorRT Developer Guide*.

ISelectLayer

The ISelectLayer returns either of the two inputs depending on the condition. For more information, see the C++ class ISelectLayer or the Python class ISelectLayer.

ITripLimitLayer

The ITripLimitLayer specifies how many times the loop iterates. For more information, see the C++ class ITripLayer or the Python class ITripLayer and Working With Loops in the *TensorRT Developer Guide*.

New operations

ONNX: Added ConstantOfShape, DequantizeLinear, Equal, Erf, Expand, Greater, GRU, Less, Loop, LRN, LSTM, Not, PRelu, QuantizeLinear, RandomUniform, RandomUniformLike, Range, RNN, Scan, Sqrt, Tile, and Where.

For more information, see the full list of Supported Ops in the Support Matrix.

Boolean tensor support

TensorRT supports boolean tensors which can be marked as network input and output. IElementWiseLayer, IUnaryLayer (only kNOT), IShuffleLayer, ITripLimit (only kWHILE) and ISelectLayer support the boolean datatype. Boolean tensors can be used only with FP32 and FP16 precision networks. For more information, refer to the Layers section in the *TensorRT Developer Guide*.

Compatibility

- TensorRT 7.0.0 has been tested with the following:
  - cuDNN 7.6.5
  - TensorFlow 1.14.0
  - PyTorch 1.3.0
  - ONNX 1.6.0
- This TensorRT release supports CUDA 9.0, 10.0, and 10.2.
- For PowerPC users, Tesla V100 and Tesla T4 GPUs are supported.

Limitations

- UFF samples, such as `sampleUffMNIST`, `sampleUffSSD`, `sampleUffPluginV2Ext`, `sampleUffMaskRCNN`, `sampleUffFasterRCNN`, `uff_custom_plugin`, and `uff_ssd`, support TensorFlow 1.x and not models trained with TensorFlow 2.0.
- Loops and `DataType::kBOOL` are supported on limited platforms. On platforms without loop support, `INetworkDefinition::addLoop` returns `nullptr`. Attempting to build an engine using operations that consume or produce `DataType::kBOOL` on a platform without support, results in validation rejecting the network. For details on which platforms are supported with loops, refer to the Features For Platforms And Software section in the *TensorRT Support Matrix*.
- Explicit precision networks with quantized and de-quantized nodes are only supported on devices with hardware INT8 support. Running on devices without hardware INT8 support results in undefined behavior.

Deprecated Features

The following features are deprecated in TensorRT 7.0.0:

- **Backward Compatibility and Deprecation Policy** - When a new function, for example `foo`, is first introduced, there is no explicit version in the name and the version is assumed to be `1`. When changing the API of an existing TensorRT function `foo` (usually to support some new functionality), first, a new routine `fooV<N>` is created where `N` represents the `N`th version of the function and the previous version `fooV<N-1>` remains untouched to ensure backward compatibility. At this point, `fooV<N-1>` is considered deprecated, and should be treated as such by users of TensorRT.

  Starting with TensorRT 7, we will be eliminating deprecated API per the following policy.

  - APIs already marked deprecated prior to TensorRT 7 (6 and older) will be removed in the next major release of TensorRT 8.
  - APIs deprecated in TensorRT `<M>`, where `M` is the major version greater than or equal to 7, will be removed in TensorRT `<M+2>`. This means that deprecated APIs remain functional for two major releases before they are removed.
- **Deprecation of Caffe Parser and UFF Parser** - We are deprecating Caffe Parser and UFF Parser in TensorRT 7. They will be tested and functional in the next major release of TensorRT 8, but we plan to remove the support in the subsequent major release. Plan to migrate your workflow to use `tf2onnx`, `keras2onnx` or TensorFlow-TensorRT (TF-TRT) for deployment.

Fixed Issues

- You no longer have to build ONNX and TensorFlow from source in order to workaround pybind11 compatibility issues. The TensorRT Python bindings are now built using pybind11 version 2.4.3.

- Windows users are now able to build applications designed to use the TensorRT refittable engine feature. The issue related to unresolved symbols has been resolved.
- A virtual destructor has been added to the `IPluginFactory` class.

Known Issues

- The UFF parser generates unused `IConstantLayer` objects that are visible via method `NetworkDefinition::getLayer` but optimized away by TensorRT, so an attempt to refit the weights with `IRefitter::setWeights` will be rejected. Given an `IConstantLayer* layer`, you can detect whether it is used for execution by checking: `layer->getOutput(0)->isExecutionTensor()`.
- The ONNX parser does not support RNN, LSTM, and GRU nodes when the activation type of the forward pass does not match the activation type of the reverse pass in bidirectional cases.
- The INT8 calibration does not work with dynamic shapes. To workaround this issue, ensure there are two passes in the code:
  1. Using a fixed shape input to build the engine in the first pass, allows TensorRT to generate the calibration cache.
  2. Then, create the engine again using the dynamic shape input and the builder will reuse the calibration cache generated in the first pass.

Chapter 4. TensorRT Release 6.x.x

4.1. TensorRT Release 6.0.1

This is the TensorRT 6.0.1 release notes for Linux and Windows users. This release includes fixes from the previous TensorRT 5.x.x releases as well as the following additional changes. These release notes are applicable to workstation, server, and JetPack users unless appended specifically with *(not applicable for Jetson platforms)*.

For previous TensorRT release notes, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

- **New layers:**
  IResizeLayer

  The IResizeLayer implements the resize operation on an input tensor. For more information, see IResizeLayer: TensorRT API and IResizeLayer: TensorRT Developer Guide.

  IShapeLayer

  The IShapeLayer gets the shape of a tensor. For more information, see IShapeLayer: TensorRT API and IShapeLayer: TensorRT Developer Guide.

  PointWise fusion

  Multiple adjacent pointwise layers can be fused into a single pointwise layer, to improve performance. For more information, see the TensorRT Best Practices Guide.
- **New operators:**
  3-dimensional convolution

  Performs a convolution operation with 3D filters on a 5D tensor. For more information, see addConvolutionNd in the TensorRT API and IConvolutionalLayer in the TensorRT Developer Guide.

3-dimensional deconvolution

Performs a deconvolution operation with 3D filters on a 5D tensor. For more information, see addDeconvolutionNd in the TensorRT API and IDeconvolutionLayer in the TensorRT Developer Guide.

3-dimensional pooling

Performs a pooling operation with a 3D sliding window on a 5D tensor. For more information, see addPoolingNd in the TensorRT API and IPoolingLayer in the TensorRT Developer Guide.

- **New plugins:**

Added a persistent LSTM plugin; a half precision persistent LSTM plugin that supports variable sequence lengths. This plugin also supports bi-direction, setting initial hidden/cell values, storing final hidden/cell values, and multi layers. You can use it through the PluginV2 interface, achieves better performance with small batch sizes, and is currently only supported on Linux. For more information, see Persistent LSTM Plugin in the TensorRT Developer Guide. *(not applicable for Jetson platforms)*

- **New operations:**

TensorFlow

Added `ResizeBilinear` and `ResizeNearest` ops.

ONNX

Added `Resize` op.

For more information, see the full list of Supported Ops in the Support Matrix.

- **New samples:**

sampleDynamicReshape

Added sampleDynamicReshape which demonstrates how to use dynamic input dimensions in TensorRT by creating an engine for resizing dynamically shaped inputs to the correct size for an ONNX MNIST model. For more information, see Working With Dynamic Shapes in the TensorRT Developer Guide, Digit Recognition With Dynamic Shapes in the TensorRT Samples Support Guide and the GitHub: sampleDynamicReshape directory.

sampleReformatFreeIO

Added sampleReformatFreeIO which uses a Caffe model that was trained on theMNIST dataset and performs engine building and inference using TensorRT. Specifically, it shows how to use reformat free I/O tensors APIs to explicitly specify I/O formats to `TensorFormat::kLINEAR`, `TensorFormat::kCHW2` and `TensorFormat::kHWC8` for Float16 and INT8 precision. For more information, see Specifying I/O Formats Using The Reformat Free I/O Tensors APIs in the TensorRT Samples Support Guide and the GitHub: sampleReformatFreeIO directory.

sampleUffPluginV2Ext

Added sampleUffPluginV2Ext which implements the custom pooling layer for the MNIST model (`data/samples/lenet5_custom_pool.uff`) and demonstrates how to extend INT8 I/O for a plugin. For more information, see Adding A Custom Layer That Supports INT8 I/O To Your Network In TensorRT in the TensorRT Samples Support Guide and the GitHub: sampleUffPluginV2Ext directory.

sampleNMT

Added sampleNMT which demonstrates the implementation of Neural Machine Translation (NMT) based on a TensorFlow seq2seq model using the TensorRT API. The TensorFlow seq2seq model is an open sourced NMT project that uses deep neural networks to translate text from one language to another language. For more information, see Neural Machine Translation (NMT) Using A Sequence To Sequence (seq2seq) Model in the TensorRT Samples Support Guide and Importing A Model Using The C++ API For Safety in the TensorRT Developer Guide and the GitHub: sampleNMT directory.

sampleUffMaskRCNN

This sample, sampleUffMaskRCNN, performs inference on the Mask R-CNN network in TensorRT. Mask R-CNN is based on the Mask R-CNN paper which performs the task of object detection and object mask predictions on a target image. This sample's model is based on the Keras implementation of Mask R-CNN and its training framework can be found in the Mask R-CNN Github repository. For more information, see sampleUffMaskRCNN in the TensorRT Sample Support Guide. This sample is available only in GitHub: sampleUffMaskRCNN and is not packaged with the product. *(not applicable for Jetson platforms)*

sampleUffFasterRCNN

This sample, sampleUffFasterRCNN, is a UFF TensorRT sample for Faster-RCNN in NVIDIA Transfer Learning Toolkit SDK. This sample serves as a demo of how to use pretrained Faster-RCNN model in Transfer Learning Toolkit to do inference with TensorRT. For more information, see sampleUffFasterRCNN in the TensorRT Sample Support Guide. This sample is available only in GitHub: sampleUffFasterRCNN and is not packaged with the product. *(not applicable for Jetson platforms)*

- **New optimizations:**
  Dynamic shapes

  The size of a tensor can vary at runtime. IShuffleLayer, ISliceLayer, and the new IResizeLayer now have optional inputs that can specify runtime dimensions. IShapeLayer can get the dimensions of tensors at runtime, and some layers can compute new dimensions. For more information, see Working With Dynamic Shapes and TensorRT Layers in the TensorRT Developer Guide, Digit Recognition With Dynamic Shapes in the TensorRT Samples Support Guide and the GitHub: sampleDynamicReshape directory.

Reformat free I/O

Network I/O tensors can be different to linear FP32. Formats of network I/O tensors now have APIs to be specified explicitly. The removal of reformatting is beneficial to many applications and specifically saves considerable memory traffic time. For more information, see Working With Reformat-Free Network I/O Tensors and Example 4: Add A Custom Layer With INT8 I/O Support Using C++ in the TensorRT Developer Guide.

Layer optimizations

Shuffle operations that are equivalent to identify operations on the underlying data will be omitted, if the input tensor is only used in the shuffle layer and the input and output tensors of this layer are not input and output tensors of the network. TensorRT no longer executes additional kernels or memory copies for such operations. For more information, see How Does TensorRT Work in the TensorRT Developer Guide.

New INT8 calibrator

`MinMaxCalibrator` - Preferred calibrator for NLP tasks. Supports per activation tensor scaling. Computes scales using per tensor absolute maximum value. For more information, see INT8 Calibration Using C++.

Explicit precision

You can manually configure a network to be an explicit precision network in TensorRT. This feature enables users to import pre-quantized models with explicit quantizing and dequantizing scale layers into TensorRT. Setting the network to be an explicit precision network implies that you will set the precision of all the network input tensors and layer output tensors in the network. TensorRT will not quantize the weights of any layer (including those running in lower precision). Instead, weights will simply be cast into the required precision. For more information about explicit precision, see Working With Explicit Precision Using C++ and Working With Explicit Precision Using Python in the TensorRT Developer Guide.

- **Installation:**
  - Added support for RPM and Debian packages for PowerPC users. *(not applicable for Jetson platforms)*

Compatibility

- TensorRT 6.0.1 has been tested with the following:
  - cuDNN 7.6.5
  - TensorFlow 1.14.0
  - PyTorch 1.1.0
  - ONNX 1.5.0
- This TensorRT release supports CUDA 9.0 *(not applicable for Jetson platforms)*, 10.0, and 10.1 update 2 *(not applicable for Jetson platforms)*, and 10.2.

- For PowerPC users, Tesla V100 Volta and Turing T4 GPUs are supported.

Limitations

- Upgrading TensorRT to the latest version is only supported when the currently installed TensorRT version is equal to or newer than the last two public releases. For example, TensorRT 6.x.x supports upgrading from TensorRT 5.0.x and TensorRT 5.1.x. *(not applicable for Jetson platforms)*
- Calibration for a network with INT8 I/O tensors requires FP32 calibration data.
- Shape tensors cannot be network inputs or outputs. Shape tensors can be created by `IConstantLayer`, `IShapeLayer`, or any of the following operations on shape tensors: `IConcatenationLayer`, `IElementWiseLayer`, `IGatherLayer`, `IReduceLayer` (`kSUM`, `kMAX`, `kMIN`, `kPROD`), `IShuffleLayer`, or `ISliceLayer`.

Deprecated Features

The following features are deprecated in TensorRT 6.0.1:

Samples changes

- The PGM files for the MNIST samples have been removed. A script, called `generate_pgms.py` (or `download_pgms.py` for CUDA 10.2), has been provided in the `samples/mnist/data` directory to generate the images using the dataset.
- `--useDLACore=0` is no longer a valid option for sampleCharRNN as DLA does not support FP32 or RNN's, and the sample is only written to work with FP32 in all cases.

Fixed Issues

- Logging level `Severity::kVERBOSE` is now fully supported. Log messages with this level of severity are verbose messages with debugging information.
- Deconvolution layer with stride > 32 is now supported on DLA.
- Deconvolution layer with kernel size > 32 is now supported on DLA.

Known Issues

- For Ubuntu 14.04 and CentOS7, in order for ONNX, TensorFlow and TensorRT to co-exist in the same environment, ONNX and TensorFlow must be built from source using your system's native compilers. It's especially important to build ONNX and TensorFlow from source when using the IBM Anaconda channel for PowerPC to avoid compatibility issues with pybind11 and protobuf. *(not applicable for Jetson platforms)*
- PointWise fusions will be disabled when the SM version is lower than 7.0 due to a performance issue. This includes all pre-Volta GPUs, for example, Pascal, Maxwell, Kepler, TX-1, TX-2, Nano.
- TensorRT assumes that all resources for the device it is building on are available for optimization purposes. Concurrent use of multiple TensorRT builders (for example, multiple `trtexec` instances) to compile on different targets (DLA0, DLA1 and GPU) may

oversubscribe system resources causing undefined behavior (meaning, inefficient plans, builder failure, or system instability).

It is recommended to use `trtexec` with the `--saveEngine` argument to compile for different targets (DLA and GPU) separately and save their plan files. Such plan files can then be reused for loading (using `trtexec` with the `--loadEngine` argument) and submitting multiple inference jobs on the respective targets (DLA0, DLA1, GPU). This two step process alleviates over-subscription of system resources during the build phase while also allowing execution of the plan file to proceed without interference by the builder.

- Windows users are currently unable to refit an engine due to some linking issues. You will encounter undefined symbols while building an application designed to use the TensorRT refittable engine feature. *(not applicable for Jetson platforms)*

Chapter 5. TensorRT Release 5.x.x

5.1. TensorRT Release 5.1.5

This is the TensorRT 5.1.5 release notes for Linux and Windows users. This release includes fixes from the previous TensorRT 5.1.x releases as well as the following additional changes.

For previously released versions of TensorRT, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

TensorRT Open Source Software (OSS)

The TensorRT GitHub repository contains the Open Source Software (OSS) components of NVIDIA TensorRT. Included are the sources for TensorRT plugins and parsers (Caffe and ONNX) libraries, as well as sample applications demonstrating usage and capabilities of the TensorRT platform. Refer to the README.md file for prerequisites, steps for downloading, setting-up the build environment, and instructions for building the TensorRT OSS components.

For more information, see the NVIDIA Developer news article NVIDIA open sources parsers and plugins in TensorRT.

Compatibility

- TensorRT 5.1.5 has been tested with the following:
  - cuDNN 7.5.0
  - TensorFlow 1.12.0
  - PyTorch 1.0
  - ONNX 1.4.1
- This TensorRT release supports CUDA 9.0, CUDA 10.0, and CUDA 10.1.

Deprecated Features

The following features are deprecated in TensorRT 5.1.5:

- `getDIGITS` has been removed from the TensorRT package.

Known Issues

- For Ubuntu 14.04 and CentOS7, there is a known bug when trying to import TensorRT and ONNX Python modules together due to different compiler versions used to generate their respective Python bindings. As a work around, build the ONNX module from source using your system's native compilers.
- You may see the following warning when running programs linked with TensorRT 5.1.5 and CUDA 10.1 libraries:

```
[W] [TRT] TensorRT was compiled against cuBLAS 10.2.0 but is linked against cuBLAS
 10.1.0.
```

  You can resolve this by updating your CUDA 10.1 installation to 10.1 update 1 here.
- There is a known issue in sample yolov3_onnx with ONNX versions > 1.4.1. To work around this, install version 1.4.1 of ONNX through:

```
pip uninstall onnx; pip install onnx==1.4.1
```

5.2. TensorRT Release 5.1.3

This is the TensorRT 5.1.3 release notes for PowerPC users. This release includes fixes from the previous TensorRT 5.1.x releases as well as the following additional changes.

For previously released versions of TensorRT, see the TensorRT Archived Documentation.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

Samples

The `README.md` files for many samples, located within each sample source directory, have been greatly improved. We hope this makes it easier to understand the sample source code and successfully run the sample.

ONNX parser

The ONNX parser now converts GEMMs and MatMuls using the MatrixMultiply layer, and adds support for scaling the results with the alpha and beta parameters.

Asymmetric padding

- `IConvolutionLayer`, `IDeconvolutionLayer` and `IPoolingLayer` directly support setting asymmetric padding. You do not need to add an explicit `IPaddingLayer`.
- The new APIs are `setPaddingMode()`, `setPrePadding()` and `setPostPadding()`. The `setPaddingMode()` method takes precedence over `setPaddingMode()` and `setPrePadding()` when more than one padding method is used.
- The Caffe, UFF, and ONNX parsers have been updated to support the new asymmetric padding APIs.

Precision optimization

TensorRT provides optimized kernels for mixed precision (FP32, FP16 and INT8) workloads on Turing GPUs, and optimizations for depthwise convolution operations. You can control the precision per-layer with the `ILayer` APIs.

Compatibility

- TensorRT 5.1.3 has been tested with the following:
  - cuDNN 7.5.0
  - TensorFlow 1.12.0
  - PyTorch 1.0
  - ONNX 1.4.1
- This TensorRT release supports CUDA 10.1.
- TensorRT will now emit a warning when the major, minor, and patch versions of cuDNN and cuBLAS do not match the major, minor, and patch versions that TensorRT is expecting.

Limitations

- For CentOS and RHEL users, when choosing Python 3:
  - Only Python version 3.6 from EPEL is supported by the RPM installation.
  - Only Python versions 3.4 and 3.6 from EPEL are supported by the tar installation.
- In order to run the UFF converter and its related C++ and Python samples on PowerPC, it's necessary to install TensorFlow for PowerPC. For more information, see Install TensorFlow on Power systems.
- In order to run the PyTorch samples on PowerPC, it's necessary to install PyTorch specifically built for PowerPC, which is not available from PyPi. For more information, see Install PyTorch on Power systems.

Deprecated Features

The following features are deprecated in TensorRT 5.1.3:

- `sampleNMT` has been removed from the TensorRT package. The public data source files have changed and no longer work with the sample.

Fixed Issues

The following issues have been resolved in TensorRT 5.1.3:

- Fixed the behavior of the Caffe crop layer when the layer has an asymmetric crop offset.
- `ITensor::getType()` and `ILayer::getOutputType()` now report the type correctly. Previously, both types reported `DataType::kFLOAT` even if the output type should

have been `DataType::kINT32`. For example, the output type of `IConstantLayer` with `DataType::kINT32` weights is now correctly reported as `DataType::kINT32`. The affected layers include:

- `IConstantLayer` (when weights have type `DataType::kINT32`)
- `IConcatentationLayer` (when inputs have type `DataType::kINT32`)
- `IGatherLayer` (when first input has type `DataType::kINT32`)
- `IIdentityLayer` (when input has type `DataType::kINT32`)
- `IShuffleLayer` (when input has type `DataType::kINT32`)
- `ISliceLayer` (when input has type `DataType::kINT32`)
- `ITopKLayer` (second output)

- When using INT8 mode, dynamic ranges are no longer required for INT32 tensors, even if you're not using automatic quantization.
- Using an INT32 tensor where a floating-point tensor is expected, or vice-versa, issues an error explaining the mismatch instead of asserting failure.
- The ONNX TensorRT parser now attempts to downcast INT64 graph weights to INT32.
- Fixed an issue where the engine would fail to build when asymmetric padding convolutions were present in the network.

Known Issues

- When running ShuffleNet with small batch sizes between 1 and 4, you may encounter performance regressions of up to 15% compared to TensorRT 5.0.
- When running ResNeXt101 with a batch size of 4 using INT8 precision on a Volta GPU, you may encounter intermittent performance regressions of up to 10% compared to TensorRT 5.0. Rebuilding the engine may resolve this issue.
- There is a known issue in sample yolov3_onnx with ONNX versions > 1.4.1. To work around this, install version 1.4.1 of ONNX through:

```
pip uninstall onnx; pip install onnx==1.4.1
```

5.3. TensorRT Release 5.1.2 Release Candidate (RC)

This is the release candidate (RC) for TensorRT 5.1.2 and is applicable to Linux and Windows users. This RC includes several enhancements and improvements compared to the previously released TensorRT 5.0.2.

This preview release is for early testing and feedback, therefore, for production use of TensorRT, continue to use TensorRT 5.0.2.

For previously released versions of TensorRT, see the TensorRT Documentation Archives.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

Improved performance of HMMA and IMMA convolution

The performance of Convolution, including Depthwise Separable Convolution and Group Convolution has improved in FP16 and INT8 modes on Volta and Turing. For example: ResNeXt-101 batch=1 INT8 3x speedup on Tesla T4.

Reload weights for an existing TensorRT engine

Engines can be refitted with new weights. For more information, see Refitting An Engine.

New supported operations

Caffe: Added `BNLL`, `Clip` and `ELU` ops. Additionally, the `leaky ReLU` option for the `ReLU` op (`negative_slope != 0`) was added.

UFF: Added `ArgMax`, `ArgMin`, `Clip`, `Elu`, `ExpandDims`, `Identity`, `LeakyReLU`, `Recip`, `Relu6`, `Sin`, `Cos`, `Tan`, `Asin`, `Acos`, `Atan`, `Sinh`, `Cosh`, `Asinh`, `Acosh`, `Atanh`, `Ceil`, `Floor`, `Selu`, `Slice`, `Softplus` and `Softsign` ops.

ONNX: Added `ArgMax`, `ArgMin`, `Clip`, `Cast`, `Elu`, `Selu`, `HardSigmoid`, `Softplus`, `Gather`, `ImageScaler`, `LeakyReLU`, `ParametricSoftplus`, `Sin`, `Cos`, `Tan`, `Asin`, `Acos`, `Atan`, `Sinh`, `Cosh`, `Asinh`, `Acosh`, `Atanh`, `Ceil`, `Floor`, `ScaledTanh`, `Softsign`, `Slice`, `ThresholdedRelu` and `Unsqueeze` ops.

For more information, see the TensorRT Support Matrix.

NVTX support

NVIDIA Tools Extension SDK (NVTX) is a C-based API for marking events and ranges in your applications. NVTX annotations were added in TensorRT to help correlate the runtime engine layer execution with CUDA kernel calls. NVIDIA Nsight Systems supports collecting and visualizing these events and ranges on the timeline. NVIDIA Nsight Compute also supports collecting and displaying the state of all active NVTX domains and ranges in a given thread when the application is suspended.

New layer

Added support for the Slice layer. The Slice layer implements a slice operator for tensors. For more information, see ISliceLayer.

RNNs

Changed RNNv1 and RNNv2 validation of hidden and cell input/output dimensions. This affects only bidirectional RNNs.

EntropyCalibrator2

Added Entropy Calibration algorithm; which is the preferred calibrator.

Python support

Python 3 is now supported for CentOS and RHEL users. The Python 3 wheel files have been split so that each wheel file now contains the Python bindings for only one Python version and follows `pip` naming conventions.

New Python samples

- INT8 Calibration In Python - This sample demonstrates how to create an INT8 calibrator, build and calibrate an engine for INT8 mode, and finally run inference in INT8 mode.
- Engine Refit In Python **-** This sample demonstrates the engine refit functionality provided by TensorRT. The model first trains an MNIST model in PyTorch, then recreates the network in TensorRT.

For more information, see the Samples Support Guide.

NVIDIA Machine Learning network repository installation

TensorRT 5.1 can now be directly installed from the NVIDIA Machine Learning network repository when only the C++ libraries and headers are required. The intermediate step of downloading and installing a local repo from the network repo is no longer required. This simplifies the number of steps required to automate the TensorRT installation. See the TensorRT Installation Guide for more information.

Breaking API Changes

- A `kVERBOSE` logging level was added in TensorRT 5.1, however, due to ABI implications, `kVERBOSE` is not currently being used. Messages at the `kVERBOSE` logging level may be emitted in a future release.

Compatibility

- TensorRT 5.1.2 RC has been tested with the following:
  - cuDNN 7.5.0
  - TensorFlow 1.12.0
  - PyTorch 1.0
- This TensorRT release supports CUDA 9.0, CUDA 10.0 and CUDA 10.1.

Limitations

- A few optimizations are disabled when building refittable engines:
  - `IScaleLayer` operations that have non-zero count of weights for shift or scale and are mathematically the identity function will not be removed, since a refit of the shift or scale weights could make it a non-identity function. `IScaleLayer` operations where the shift and scale weights have zero count are still removed if the power weights are unity.
  - Optimizations for multilayer perceptrons are disabled. These optimizations target serial compositions of `IFullyConnectedLayer`, `IMatrixMultiplyLayer`, and `IActivationLayer`.

Deprecated Features

The following features are deprecated in TensorRT 5.1.2 RC:

- The UFF Parser which is used to parse a network in UFF format will be deprecated in a future release. The recommended method of importing TensorFlow models to TensorRT is using TensorFlow with TensorRT (TF-TRT). For step-by-step instructions on how to accelerate inference in TF-TRT, see the TF-TRT User Guide and Release Notes. For source code from GitHub, see Examples for TensorRT in TensorFlow (TF-TRT).
- Deprecated `--engine=<filename>` option in `trtexec`. Use `--saveEngine=<filename>` and `--loadEngine=<filename>` instead for clarity.

Known Issues

- Using the current public data sources, sampleNMT produces incorrect results which results in a low BLEU score. This sample will be removed in the next release so that we can update the source code to work with the latest public data.
- There is a known multilayer perceptron (MLP) performance regression in TensorRT 5.1.2 compared to TensorRT 5.0. During the engine build phase the GPU cache state may lead to different tactic selections on Turing. The magnitude of the regression depends on the batch size and the depth of the network.
- On sampleSSD and sampleUffSSD during INT8 calibration, you may encounter a file read error in `TensorRT-5.1.x.x/data/samples/ssd/VOC2007/list.txt`. This is due to line-ending differences on Windows vs Linux. To workaround this problem, open `list.txt` in a text editor and ensure that the file is using Unix-style line endings.
- Python sample yolov3_onnx is functional only for ONNX versions greater than 1.1.0 and less than 1.4.0.

5.4. TensorRT Release 5.1.1 Release Candidate (RC)

This is the release candidate (RC) for TensorRT 5.1.1 and is applicable to automotive users on PDK version 5.1.3. This RC includes several enhancements and improvements compared to the previously released TensorRT 5.0.3.

This preview release is for early testing and feedback, therefore, for production use of TensorRT, continue to use TensorRT 5.0.3.

For previously released versions of TensorRT, see the TensorRT Documentation Archives.

Key Features And Enhancements

This TensorRT release includes the following key features and enhancements.

- CUDA 10.1 is now supported. For more information, see the CUDA 10.1 Release Notes.

Breaking API Changes

- A `kVERBOSE` logging level was added in TensorRT 5.1.0, however, due to ABI implications, `kVERBOSE` is no longer being used in TensorRT 5.1.1. It may be used again in a future release.

Compatibility

- TensorRT 5.1.1 RC has been tested with the following:
  - cuDNN 7.5.0
- This TensorRT release supports CUDA 10.1.

Limitations

- The Python API is not included in this package.

Known Issues

- When linking against CUDA 10.1, performance regressions may occur under Drive 5.0 QNX and Drive 5.0 Linux because of a regression in cuBLAS. This affects the FullyConnected layers in AlexNet, VGG19, and ResNet-50 for small batch sizes (between 1 and 4).
- Performance regressions of around 10% may be seen when using group convolutions caused by a CUDA mobile driver bug. These regressions might be seen in networks such as ResNext and ShuffleNet.

5.5. TensorRT Release 5.1.0 Release Candidate (RC)

This is the release candidate (RC) for TensorRT 5.1.0. It includes several enhancements and improvements compared to the previously released TensorRT 5.0.x.

This preview release is for early testing and feedback, therefore, for production use of TensorRT, continue to use TensorRT 5.0.2.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

Improved performance of HMMA and IMMA convolution

The performance of Convolution, including Depthwise Separable Convolution and Group Convolution has improved in FP16 and INT8 modes on Volta, Xavier and Turing. For example:

- ResNet50 INT8 batch=8 1.2x speedup on Jetson AGX Xavier
- MobileNetV2 FP16 batch=8 1.2x speedup on Jetson AGX Xavier
- ResNeXt-101 batch=1 INT8 3x speedup on Tesla T4

Reload weights for an existing TensorRT engine

Engines can be refitted with new weights. For more information, see Refitting An Engine.

DLA with INT8

Added support for running the AlexNet network on DLA using `trtexec` in INT8 mode. For more information, see Working With DLA.

New supported operations

Caffe: Added `BNLL`, `Clip` and `ELU` ops. Additionally, the `leaky ReLU` option for the `ReLU` op (`negative_slope != 0`) was added.

UFF: Added `ArgMax`, `ArgMin`, `Clip`, `Elu`, `ExpandDims`, `Identity`, `LeakyReLU`, `Recip`, `Relu6`, `Sin`, `Cos`, `Tan`, `Asin`, `Acos`, `Atan`, `Sinh`, `Cosh`, `Asinh`, `Acosh`, `Atanh`, `Ceil`, `Floor`, `Selu`, `Slice`, `Softplus` and `Softsign` ops.

ONNX: Added `ArgMax`, `ArgMin`, `Clip`, `Cast`, `Elu`, `Selu`, `HardSigmoid`, `Softplus`, `Gather`, `ImageScaler`, `LeakyReLU`, `ParametricSoftplus`, `Sin`, `Cos`, `Tan`, `Asin`, `Acos`, `Atan`, `Sinh`, `Cosh`, `Asinh`, `Acosh`, `Atanh`, `Ceil`, `Floor`, `ScaledTanh`, `Softsign`, `Slice`, `ThresholdedRelu` and `Unsqueeze` ops.

For more information, see the TensorRT Support Matrix.

NVTX support

NVIDIA Tools Extension SDK (NVTX) is a C-based API for marking events and ranges in your applications. NVTX annotations were added in TensorRT to help correlate the runtime engine layer execution with CUDA kernel calls. NVIDIA Nsight Systems supports collecting and visualizing these events and ranges on the timeline. NVIDIA Nsight Compute also supports collecting and displaying the state of all active NVTX domains and ranges in a given thread when the application is suspended.

New layer

Added support for the Slice layer. The Slice layer implements a slice operator for tensors. For more information, see ISliceLayer.

RNNs

Changed RNNv1 and RNNv2 validation of hidden and cell input/output dimensions. This affects only bidirectional RNNs.

EntropyCalibrator2

Added Entropy Calibration algorithm; which is the preferred calibrator. This is also the required calibrator for DLA INT8 because it supports per activation tensor scaling.

ILogger

Added verbose severity level in `ILogger` for emitting debugging messages. Some messages that were previously logged with severity level kINFO are now logged with severity level kVERBOSE. Added new `ILogger` derived class in samples and `trtexec`. Most messages should be categorized (using the severity level) as:

[V]
For verbose debug informational messages.

[I]
For "instructional" informational messages.

[W]
For warning messages.

[E]
For error messages.

[F]
For fatal error messages.

Python

- INT8 Calibration In Python - This sample demonstrates how to create an INT8 calibrator, build and calibrate an engine for INT8 mode, and finally run inference in INT8 mode.
- Engine Refit In Python - This sample demonstrates the engine refit functionality provided by TensorRT. The model first trains an MNIST model in PyTorch, then recreates the network in TensorRT.

For more information, see the Samples Support Guide.

Python bindings

Added Python bindings to the `aarch64-gnu` release package (debian and tar).

RPM installation

Provided installation support for Red Hat Enterprise Linux (RHEL) and CentOS users to upgrade from TensorRT 5.0.x to TensorRT 5.1.x. For more information, see the upgrading instructions in the Installation Guide.

Breaking API Changes

- A new logging level, `kVERBOSE`, was added in TensorRT 5.1.0. Messages are being emitted by the TensorRT builder and/or engine using this new logging level. Since the logging level did not exist in TensorRT 5.0.x, some applications might not handle the new logging level properly and in some cases the application may crash. In the next release, more descriptive messages will appear when using the `kINFO` logging level because the `kVERBOSE` messages will be produced using `kINFO`. However, the `kVERBOSE` logging level will remain in the API and `kVERBOSE` messages may be emitted in a future TensorRT release.

Compatibility

- TensorRT 5.1.0 RC has been tested with cuDNN 7.3.1.
- TensorRT 5.1.0 RC has been tested with TensorFlow 1.12.0.
- TensorRT 5.1.0 RC has been tested with PyTorch 1.0.
- This TensorRT release supports CUDA 10.0.

Limitations

- A few optimizations are disabled when building refittable engines.
  - `IScaleLayer` operations that have non-zero count of weights for shift or scale and are mathematically the identity function will not be removed, since a refit of the shift or scale weights could make it a non-identity function. `IScaleLayer` operations where the shift and scale weights have zero count are still removed if the power weights are unity.
  - Optimizations for multilayer perceptrons are disabled. These optimizations target serial compositions of `IFullyConnectedLayer`, `IMatrixMultiplyLayer`, and `IActivationLayer`.
- DLA limitations
  - FP16 LRN is supported with the following parameters:
    - `local_size` = 5
    - `alpha` = 0.0001
    - `beta` = 0.75
  - INT8 LRN, Sigmoid, and Tanh are not supported.

  For more information, see DLA Supported Layers.

Deprecated Features

The following features are deprecated in TensorRT 5.1.0 RC:

- Deprecated `--engine=<filename>` option in `trtexec`. Use `--saveEngine=<filename>` and `--loadEngine=<filename>` instead for clarity.

Known Issues

- When the tensor size is too large, such as a single tensor that has more than 4G elements, overflow may occur which will cause TensorRT to crash. As a workaround, you may need to reduce the batch size.

5.6. TensorRT Release 5.0.6

This is the release for TensorRT 5.0.6 and is applicable to JetPack 4.2.0 users.

This release includes several enhancements and improvements compared to the previously released TensorRT Release 5.0.5.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements for JetPack users.

- Python support for AArch64 Linux is included as an early access release. All features are expected to be available, however, some aspects of functionality and performance will likely be limited compared to a non-EA release.
- The UFF parser's memory usage was significantly reduced to better accommodate boards with small amounts of memory.

Compatibility

- TensorRT 5.0.6 has been tested with the following:
  - cuDNN 7.3.1
  - TensorFlow 1.12
  - PyTorch 1.0
- This TensorRT release supports CUDA 10.0.

Known Issues

- The default workspace size for sampleUffSSD is 1 GB. This may be too large for the Jetson TX1 NANO, therefore, change the workspace for the builder in the source file via the following code:
```
builder->setMaxWorkspaceSize(16_MB);
```
- In order to run larger networks or larger batch sizes with TensorRT, it may be necessary to free memory on the board. This can be accomplished by running in headless mode or killing processes with high memory consumption.
- Due to limited system memory on the Jetson TX1 NANO, which is shared between the CPU and GPU, you may not be able run some samples, for example, sampleFasterRCNN.
- Python sample `yolov3_onnx` is functional only for ONNX versions greater than 1.1.0 and less than 1.4.0.

5.7. TensorRT Release 5.0.5

This is the TensorRT 5.0.5 release notes for Android 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 Android users.

- TensorRT 5.0.5 has two sub-releases:
  - TensorRT 5.0.5.0 (without DLA support)
  - TensorRT 5.0.5.1 (with DLA support)

Compatibility

- TensorRT 5.0.5 supports CUDA 10.0
- TensorRT 5.0.5 supports cuDNN 7.3.1
- TensorRT 5.0.5 supports the Android platform with API level 26 or higher

Limitations In 5.0.5

- TensorRT 5.0.5.1 supports DLA while TensorRT 5.0.5.0 does not.

Known Issues

- For TensorRT 5.0.5.0, some sample programs have `--useDLACore` in their command line arguments, however, do not use it because this release does not support DLA.
- When running `trtexec` from a saved engine, the `--output` and `--input` command line arguments are mandatory. For example:

```
./trtexec --onnx=data/mnist/mnist.onnx --fp16 --engine=./mnist_onnx_fp16.engine
./trtexec --engine=./mnist_onnx_fp16.engine --input=Input3 --output=Plus214_Output_0
```

- When running applications that use DLA on Xavier based platforms that also contain a discrete GPU (dGPU), you may be required to select the integrated GPU (iGPU). This can be done using the following command:

```
export CUDA_VISIBLE_DEVICES=1
```

5.8. 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`.

5.9. 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.

  

  Note: 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
  ```

5.10. 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 --
allowGPUFallback
```

`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.
6. 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:

1. Download the Model files from GitHub, for example:

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

2. Unzip the tar file:

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

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

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

Running the sample with a custom configuration:

1. Download the Model files from GitHub.
2. Create an input image with a PPM extension. Resize it with the dimensions of 224x224x3.
3. 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`.
4. 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 `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.
- 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 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.

5.11. 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 `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
  ```
- 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.

5.12. 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 6. TensorRT Release 4.x.x

6.1. 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` the `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 <= 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
```

6.2. 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.

6.3. 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 7. TensorRT Release 3.x.x

7.1. 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.

7.2. TensorRT Release 3.0.3

This TensorRT 3.0.3 General Availability release is a minor release and includes some improvements and fixes compared to the previously released TensorRT 3.0.2. This release is for AArch64 only.

Key Features and Enhancements

This TensorRT release includes the following key features and enhancements.

- Added support for Xavier

Using TensorRT 3.0.3

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

- When building the samples in this release, it is necessary to specify `CUDA_INSTALL_DIR` as an argument to the Makefile.
- This release does not support TensorRT Python bindings.

Known Issues

- When building the samples on aarch64 natively, there is an issue in the `Makefile.config` file that requires you to provide an additional option to make, namely `CUDA_LIBDIR`.
- The `infer_caffe_static` test fails on D5L Parker dGPU. This is a regression from the previous release.
- QnX has known performance issues with the `mmap` and `malloc()` operating system memory allocation routines. These issues can affect the performance of TensorRT; up to 10X.

7.3. 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
```

7.4. 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_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.

7.5. 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.

7.6. 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 `ibnvcaffe_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 8. TensorRT Release 2.x.x

8.1. TensorRT Release 2.1

This TensorRT 2.1 General Availability release is a minor release and includes the following improvements and fixes.

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

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform 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 document. NVIDIA accepts no 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 document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ARM

ARM, AMBA and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. All other brands or product names are the property of their respective holders. "ARM" is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Limited.; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, AS and ARM Sweden AB.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com

Trademarks

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com