CUDNN

RN-08667-001_v07 | May 2018

Release Notes

TABLE OF CONTENTS

Chapter 1. CUDNN OVERVIEW

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks. It provides highly tuned implementations of routines arising frequently in DNN applications:

- Convolution forward and backward, including cross-correlation
- Pooling forward and backward
- Softmax forward and backward
- Neuron activations forward and backward:
  - Rectified linear (ReLU)
  - Sigmoid
  - Hyperbolic tangent (TANH)
- Tensor transformation functions
- LRN, LCN and batch normalization forward and backward

cuDNN's convolution routines aim for performance competitive with the fastest GEMM (matrix multiply) based implementations of such routines while using significantly less memory.

cuDNN features customizable data layouts, supporting flexible dimension ordering, striding, and subregions for the 4D tensors used as inputs and outputs to all of its routines. This flexibility allows easy integration into any neural network implementation and avoids the input/output transposition steps sometimes necessary with GEMM-based convolutions.

cuDNN offers a context-based API that allows for easy multi-threading and (optional) interoperability with CUDA streams.

Chapter 2. CUDNN RELEASE NOTES V7.1.1

Key Features and Enhancements

The following enhancements have been added to this release:

- Added new API **`cudnnSetRNNProjectionLayers`** and **`cudnnGetRNNProjectionLayers`** to support Projection Layer for the RNN LSTM cell. In this release only the inference use case will be supported. The bi-directional and the training forward and backward for training is not supported in 7.1.1 but will be supported in the upcoming 7.1.2 release without API changes. For all the unsupported cases in this release, **`CUDNN_NOT_SUPPORTED`** is returned when projection layer is set and the RNN is called.
- The **`cudnnGetRNNLinLayerMatrixParams()`** function was enhanced and a bug was fixed without modifying its prototype. Specifically:
    - The **`cudnnGetRNNLinLayerMatrixParams()`** function was updated to support the RNN projection feature. An extra linLayerID value of 8 can be used to retrieve the address and the size of the “recurrent” projection weight matrix when "mode" in **`cudnnSetRNNDescriptor()`** is configured to **`CUDNN_LSTM`** and the recurrent projection is enabled via **`cudnnSetRNNProjectionLayers()`**.
    - Instead of reporting the total number of elements in each weight matrix in the “linLayerMatDesc” filter descriptor, the **`cudnnGetRNNLinLayerMatrixParams()`** function returns the matrix size as two dimensions: rows and columns. This allows the user to easily print and initialize RNN weight matrices. Elements in each weight matrix are arranged in the row-major order. Due to historical reasons, the minimum number of dimensions in the filter descriptor is three. In previous versions of the cuDNN library, **`cudnnGetRNNLinLayerMatrixParams()`** returned the total number of weights as follows: **`filterDimA[0]=total_size, filterDimA[1]=1, filterDimA[2]=1`**. In v7.1.1, the format was changed to: **`filterDimA[0]=1, filterDimA[1]=rows, filterDimA[2]=columns`**. In both cases, the

"format" field of the filter descriptor should be ignored when retrieved by **cudnnGetFilterNdDescriptor()**.

- A bug in **cudnnGetRNNLinLayerMatrixParams()** was fixed to return a zeroed filter descriptor when the corresponding weight matrix does not exist. This occurs, for example, for linLayerID values of 0-3 when the first RNN layer is configured to exclude matrix multiplications applied to RNN input data (**inputMode=CUDNN_SKIP_INPUT** in **cudnnSetRNNDescriptor()** specifies implicit, fixed identity weight matrices for RNN input). Such cases in previous versions of the cuDNN library caused **cudnnGetRNNLinLayerMatrixParams()** to return corrupted filter descriptors with some entries from the previous call. A workaround was to create a new filter descriptor for every invocation of **cudnnGetRNNLinLayerMatrixParams()**.

- The **cudnnGetRNNLinLayerBiasParams()** function was updated to report the bias column vectors in "linLayerBiasDesc" in the same format as **cudnnGetRNNLinLayerMatrixParams()**. In previous versions of the cuDNN library, **cudnnGetRNNLinLayerBiasParams()** returned the total number of adjustable bias parameters as follows: **filterDimA[0]=total_size, filterDimA[1]=1, filterDimA[2]=1**. In v7.1.1, the format was changed to: **filterDimA[0]=1, filterDimA[1]=rows, filterDimA[2]=1** (number of columns). In both cases, the "format" field of the filter descriptor should be ignored when retrieved by **cudnnGetFilterNdDescriptor()**. The recurrent projection GEMM does not have a bias so the range of valid inputs for the "linLayerID" argument remains the same.
- Added support for use of Tensor Core for the **CUDNN_RNN_ALGO_PERSIST_STATIC**. This required cuda cuDNN v7.1 build with CUDA 9.1 and 387 or higher driver. It will not work with CUDA 9.0 and 384 driver.
- Added RNN search API that allows the application to provide an RNN descriptor and get a list of possible algorithm choices with performance and memory usage, to allow applications to choose between different implementations. For more information, refer to the documentation of: **cudnnFindRNNForwardInferenceAlgorithmEx**, **cudnnFindRNNForwardTrainingAlgorithmEx**, **cudnnFindRNNBackwardDataAlgorithmEx**, and **cudnnFindRNNBackwardWeightsAlgorithmEx**. In this release, the search will operate on STANDARD algorithm and will not support PERSISTENT algorithms of RNN.
- Added uint8 for support for the input data for **cudnnConvolutionBiasActivationForward** and **cudnnConvolutionForward**. Currently the support is on Volta (sm 70) and later architectures. Support for older architectures will be gradually added in the upcoming releases.

- Suport for CUDNN_ACTIVATION_IDENTITY is added to **cudnnConvolutionBiasActivationForward**. This allows users to perform Convolution and Bias without Activation.
- All API functions now support logging. User can trigger logging by setting environment variable "CUDNN_LOGINFO_DBG=1" and "CUDNN_LOGDEST_DBG= <option>" where <option> (i.e., the output destination of the log) can be chosen from "stdout", "stderr", or a file path. User may also use the new Set/GetCallBack functions to install their customized callback function. Log files can be added to the reported bugs or shared with us for analysis and future optimizations through partners.nvidia.com.
- Improved performance of 3D convolution on Volta architecture.
- The following algo-related functions have been added for this release: **cudnnGetAlgorithmSpaceSize**, **cudnnSaveAlgorithm**, **cudnnRestoreAlgorithm**, **cudnnCreateAlgorithmDescriptor**, **cudnnSetAlgorithmDescriptor**, **cudnnGetAlgorithmDescriptor**, **cudnnDestroyAlgorithmDescriptor**, **cudnnCreateAlgorithmPerformance**, **cudnnSetAlgorithmPerformance**, **cudnnGetAlgorithmPerformance**, **cudnnDestroyAlgorithmPerformance**.
- All algorithms for convolutions now support groupCount > 1. This includes **cudnConvolutionForward()**, **cudnnConvolutionBackwardData()**, and **cudnnConvolutionBackwardFilter()**.

Known Issues

Following are known issues in this release:

- RNN search Algorithm is restricted to STANDARD algorithm.
- Newly added projection Layer supported for inference and one directional RNN cells.
- uint8 input for convolution is restricted to Volta and later.
- cudnnGet picks a slow algorithm that doesn't use Tensor Cores on Volta when inputs are FP16 and it is possible to do so.
- There may be a small performance regression on multi-layer RNNs using the STANDARD algorithm with Tensor Core math in this release compared to 7.0.5.

Fixed Issues

The following issues have been fixed in this release:

- 3D convolution performance improvements for Volta.
- Added support for Algorithm 0 data gradients to cover cases previously not supported.
- Removed the requirement for dropout Descriptor in RNN inference. Before application had to set a non point for the dropout Descriptor which was not used.

- Use of CUDNN_TENSOR_NCHW_VECT_C with non-zero padding resulted in a return status of CUDNN_STATUS_INTERNAL_ERROR. This issue is now fixed.

Chapter 3. CUDNN RELEASE NOTES V7.0.5

Key Features and Enhancements

The following enhancements have been added to this release:

- None.

Known Issues

Following are known issues in this release:

- cuDNN library may trigger a CPU floating point exception when FP exceptions are enabled by user. This issue exists for all 7.0.x releases.
- There are heavy use cases of RNN layers that might hit a memory allocation issue in the CUDA driver when using cuDNN v7 with CUDA 8.0 and R375 driver on pre-Pascal architectures (Kepler and Maxwell). In these cases, subsequent CUDA kernels may fail to launch with an Error Code 30. To resolve the issue, it is recommended to use the latest R384 driver (from NVIDIA driver downloads) or to ensure that the persistence daemon is started. This behavior is observed on all 7.0.x releases.
- When using TENSOR_OP_MATH mode with **`cudnnConvolutionBiasActivationForward`**, the pointer to the bias must be aligned to 16 bytes and the size of allocated memory must be multiples of 256 elements. This behavior exists for all 7.0.x releases.

Fixed Issues

The following issues have been fixed in this release:

- Corrected the algorithm fallback behavior in RNN when user set to use **`CUDNN_TENSOR_OP_MATH`** when using compute card without HMMA. Instead of returning **`CUDNN_STATUS_NOT_SUPPORTED`**, the RNN algorithm will now continue to run using **`CUDNN_DEFAULT_MATH`**. The correct behavior is to fall back to using default math when Tensor Core is not supported. Fixed to the expected behavior.

- On Volta hardware, **`BWD_FILTER_ALGO_1`** and **`BWD_DATA_ALGO_1`** convolutions using a number of filter elements greater than 512 were causing **`CUDA_ERROR_ILLEGAL_ADDRESS`** and **`CUDNN_STATUS_INTERNAL_ERROR`** errors. Logic was added to fall back to a generic kernel for these filter sizes.
- cuDNN v7 with CUDA 8.0 produced erroneous results on Volta for some common cases of Algo 1. Logic was added to fall back to a generic kernel when cudnn v7 with CUDA 8.0 is used on Volta.

Chapter 4. CUDNN RELEASE NOTES V7.0.4

Key Features and Enhancements

Performance improvements for grouped convolutions when input channels and output channels per group are 1, 2, or 4 for the following algorithms:

- **`CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM`**
- **`CUDNN_CONVOLUTION_BWD_DATA_ALGO0`**
- **`CUDNN_CONVOLUTION_BWD_DATA_ALGO_1`**
- **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO_0`**
- **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1`**

Known Issues

Following are known issues in this release:

- The CUDA 8.0 build of cuDNN may produce incorrect computations when run on Volta.
- cuDNN library triggers CPU floating point exception when FP exceptions are enabled by user. This issue exists for all 7.0.x releases.
- There are heavy use cases of RNN layers that might hit a memory allocation issue in the CUDA driver when using cuDNN v7 with CUDA 8.0 and R375 driver on pre-Pascal architectures (Kepler and Maxwell). In these cases, subsequent CUDA kernels may fail to launch with an Error Code 30. To resolve the issue, it is recommended to use the latest R384 driver (from NVIDIA driver downloads) or to ensure that the persistence daemon is started. This behavior is observed on all 7.0.x releases.
- When using TENSOR_OP_MATH mode with **`cudnnConvolutionBiasActivationForward`**, the pointer to the bias must be aligned to 16 bytes and the size of allocated memory must be multiples of 256 elements. This behavior exists for all 7.0.x releases.

Fixed Issues

The following issues have been fixed in this release:

- Fixed out-of-band global memory accesses in the 256-point 1D FFT kernel. The problem affected convolutions with 1x1 filters and tall but narrow images, e.g., 1x500 (WxH). In those cases, the workspace size for the **`FFT_TILING`** algo was computed incorrectly. There was no error in the FFT kernel.
- Eliminated a source of floating point exceptions in the **`CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED`** algorithm. The host code to generate a negative infinity floating point value was substituted with a different logic. By default, FP exceptions are disabled. However, a user program enabled them by invoking **`feenableexcept()`**. There are at least two other sources of FP exceptions in the cuDNN library, affecting for example **`BATCHNORM_SPATIAL_PERSISTENT`**. Those sources of FP exceptions will be eliminated in future releases of the cuDNN library.

Chapter 5. CUDNN RELEASE NOTES V7.0.3

Key Features and Enhancements

Performance improvements for various cases:

- Forward Grouped Convolutions where input channel per groups is 1, 2 or 4 and hardware is Volta or Pascal.
- **`cudnnTransformTensor()`** where input and output tensor is packed.

> This is an improved fallback, improvements will not be seen in all cases.

Known Issues

The following are known issues in this release:

- **`CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING`** may cause **`CUDA_ERROR_ILLEGAL_ADDRESS`**. This issue affects input images of just one 1 pixel in width and certain **`n`**, **`c`**, **`k`**, **`h`** combinations.

Fixed Issues

The following issues have been fixed in this release:

- **`AddTensor`** and **`TensorOp`** produce incorrect results for half and INT8 inputs for various use cases.
- **`cudnnPoolingBackward()`** can produce incorrect values for rare cases of non-deterministic MAX pooling with **`window_width > 256`**. These rare cases are when the maximum element in a window is duplicated horizontally (along width) by a stride of **`256*k`** for some **`k`**. The behavior is now fixed to accumulate derivatives for the duplicate that is left-most.
- **`cudnnGetConvolutionForwardWorkspaceSize()`** produces incorrect workspace size for algorithm **`FFT_TILING`** for 1d convolutions. This only occurs for large sized

convolutions where intermediate calculations produce values greater than 2^31 (2 to the power of 31).

- **`CUDNN_STATUS_NOT_SUPPORTED`** returned by **`cudnnPooling*()`** functions for small **`x`** image (**`channels * height * width < 4`**).

Chapter 6. CUDNN RELEASE NOTES V7.0.2

Key Features and Enhancements

This is a patch release of cuDNN 7.0 and includes bug fixes and performance improvements mainly on Volta.

Algo 1 Convolutions Performance Improvements

Performance improvements were made to **`CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM`**, **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1`**, and **`CUDNN_CONVOLUTION_BWD_DATA_ALGO_1`**. These improvements consist of new SASS kernels and improved heuristics. The new kernels implement convolutions over various data sizes and tile sizes. The improved heuristics take advantage of these new kernels.

Known Issues

The following are known issues in this release:

- **`cudnnGetConvolutionForwardWorkspaceSize()`** returns overflowed size_t value for certain input shape for **`CUDNN_CONVOLUTION_*_ALGO_FFT_TILING`**.
- **`cudnnPoolingBackward()`** fails for pooling window size > 256.

Fixed Issues

The following issues have been fixed in this release:

- Batch Norm **`CUDNN_BATCHNORM_SPATIAL_PERSISTENT`** might get into race conditions in certain scenarios.
- cuDNN convolution layers using **`TENSOR_OP_MATH`** with fp16 inputs and outputs and fp32 compute will use “round to nearest” mode instead of “round to zero” mode as in 7.0.1. This rounding mode has proven to achieve better results in training.

- Fixed synchronization logic in the **`CUDNN_CTC_LOSS_ALGO_DETERMINISTIC`** algo for CTC. The original code would hang in rare cases.
- Convolution algorithms using **`TENSOR_OP_MATH`** returned a workspace size from **`*GetWorkspaceSize()`** smaller than actually necessary.
- The results of int8 are inaccurate in certain cases when calling **`cudnnConvolutionForward()`** in convolution layer.
- **`cudnnConvolutionForward()`** called with **`xDesc's channel = yDesc's channel = groupCount`** could compute incorrect values when vertical padding > 0.

Chapter 7. CUDNN RELEASE NOTES V7.0.1

cuDNN v7.0.1 is the first release to support the Volta GPU architecture. In addition, cuDNN v7.0.1 brings new layers, grouped convolutions, and improved convolution find as error query mechanism.

Key Features and Enhancements

This cuDNN release includes the following key features and enhancements.

Tensor Cores

Version 7.0.1 of cuDNN is the first to support the Tensor Core operations in its implementation. Tensor Cores provide highly optimized matrix multiplication building blocks that do not have an equivalent numerical behavior in the traditional instructions, therefore, its numerical behavior is slightly different.

`cudnnSetConvolutionMathType`, `cudnnSetRNNMatrixMathType`, and `cudnnMathType_t`

The **`cudnnSetConvolutionMathType`** and **`cudnnSetRNNMatrixMathType`** functions enable you to choose whether or not to use Tensor Core operations in the convolution and RNN layers respectively by setting the math mode to either **`CUDNN_TENSOR_OP_MATH`** or **`CUDNN_DEFAULT_MATH`**.

Tensor Core operations perform parallel floating point accumulation of multiple floating point products.

Setting the math mode to **`CUDNN_TENSOR_OP_MATH`** indicates that the library will use Tensor Core operations.

The default is **`CUDNN_DEFAULT_MATH`**. This default indicates that the Tensor Core operations will be avoided by the library. The default mode is a serialized operation whereas, the Tensor Core is a parallelized operation, therefore, the two might result in slightly different numerical results due to the different sequencing of operations.

> The library falls back to the default math mode when Tensor Core operations are not supported or not permitted.

cudnnSetConvolutionGroupCount

A new interface that allows applications to perform convolution groups in the convolution layers in a single API call.

cudnnCTCLoss

cudnnCTCLoss provides a GPU implementation of the Connectionist Temporal Classification (CTC) loss function for RNNs. The CTC loss function is used for phoneme recognition in speech and handwriting recognition.

CUDNN_BATCHNORM_SPATIAL_PERSISTENT

The **CUDNN_BATCHNORM_SPATIAL_PERSISTENT** function is a new batch normalization mode for **cudnnBatchNormalizationForwardTraining** and **cudnnBatchNormalizationBackward**. This mode is similar to **CUDNN_BATCHNORM_SPATIAL**, however, it can be faster for some tasks.

cudnnQueryRuntimeError

The **cudnnQueryRuntimeError** function reports error codes written by GPU kernels when executing **cudnnBatchNormalizationForwardTraining** and **cudnnBatchNormalizationBackward** with the **CUDNN_BATCHNORM_SPATIAL_PERSISTENT** mode.

cudnnGetConvolutionForwardAlgorithm_v7

This new API returns all algorithms sorted by expected performance (using internal heuristics). These algorithms are output similarly to **cudnnFindConvolutionForwardAlgorithm**.

cudnnGetConvolutionBackwardDataAlgorithm_v7

This new API returns all algorithms sorted by expected performance (using internal heuristics). These algorithms are output similarly to **cudnnFindConvolutionBackwardAlgorithm**.

cudnnGetConvolutionBackwardFilterAlgorithm_v7

This new API returns all algorithms sorted by expected performance (using internal heuristics). These algorithms are output similarly to **cudnnFindConvolutionBackwardFilterAlgorithm**.

CUDNN_REDUCE_TENSOR_MUL_NO_ZEROS

The **MUL_NO_ZEROS** function is a multiplication reduction that ignores zeros in the data.

CUDNN_OP_TENSOR_NOT

The **OP_TENSOR_NOT** function is a unary operation that takes the negative of (alpha*A).

cudnnGetDropoutDescriptor

The **cudnnGetDropoutDescriptor** function allows applications to get dropout values.

Using cuDNN v7.0.1

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

- Multi-threading behavior has been modified. Multi-threading is allowed only when using different cuDNN handles in different threads.
- In **cudnnConvolutionBackwardFilter**, dilated convolution did not support cases where the product of all filter dimensions was odd for half precision floating point. These are now supported by **CUDNN_CONVOLUTION_BWD_FILTER_ALGO1**.
- Fixed bug that produced a silent computation error for when a batch size was larger than 65536 for **CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM**.
- In **getConvolutionForwardAlgorithm**, an error was not correctly reported in v5 when the output size was larger than expected. In v6 the **CUDNN_STATUS_NOT_SUPPORTED**, error message displayed. In v7, this error is modified to **CUDNN_STATUS_BAD_PARAM**.
- In **cudnnConvolutionBackwardFilter**, cuDNN now runs some exceptional cases correctly where it previously erroneously returned **CUDNN_STATUS_NOT_SUPPORTED**. This impacted the algorithms **CUDNN_CONVOLUTION_BWD_FILTER_ALGO0** and **CUDNN_CONVOLUTION_BWD_FILTER_ALGO3**.

Deprecated Features

The following routines have been removed:

- **cudnnSetConvolution2dDescriptor_v4**
- **cudnnSetConvolution2dDescriptor_v5**
- **cudnnGetConvolution2dDescriptor_v4**
- **cudnnGetConvolution2dDescriptor_v5**

> Only the non-suffixed versions of these routines remain.

The following routines have been created and have the same API prototype as their non-suffixed equivalent from cuDNN v6:

- **`cudnnSetRNNDescriptor_v5`** - The non-suffixed version of the routines in cuDNN v7.0.1 are now mapped to their **`_v6`** equivalent.

> **Attention** It is strongly advised to use the non-suffixed version as the **`_v5`** and **`_v6`** routines will be removed in the next cuDNN release.

- **`cudnnGetConvolutionForwardAlgorithm`**, **`cudnnGetConvolutionBackwardDataAlgorithm`**, and **`cudnnGetConvolutionBackwardFilterAlgorithm`** - A **`_v7`** version of this routine has been created. For more information, see the *Backward compatibility and deprecation policy* chapter of the cuDNN documentation for details.

Known Issues

- cuDNN pooling backwards fails for pooling window size > 256.

Notice

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

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

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

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

Trademarks

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

Copyright

www.nvidia.com