CUDNN

RN-08667-001_v7.6.4 | September 2019

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 applied 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 that is 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 in 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 V7.6.4

This is the cuDNN v7.6.4 release notes. This release includes fixes from the previous cuDNN v7.x.x releases as well as the following additional changes. For previous cuDNN release notes, see the cuDNN Archived Documentation.

Key Features and Enhancements

The following features and enhancements have been added to this release:

- Gained significant speed-up in multihead-attention forward training and inference.

Compatibility

For the latest compatibility software versions of the OS, CUDA, the CUDA driver, and the NVIDIA hardware, see the cuDNN Support Matrix for v7.6.4.

Limitations

- When launching a CUDA graph constructed via a stream capture that includes a **`cudnnConvolutionForward`** operation, the subsequent synchronization point reports a **`cudaErrorLaunchFailure`** error. This error appears when cuDNN is set to use a non-default stream.

Fixed Issues

The following issues have been fixed in this release:

- Earlier versions of cuDNN v7.6 contained symbols which would conflict with those of in TensorRT 5.1 and later. In some cases, these conflicts could lead to application crashes when applications linked against cuDNN and TensorRT. This issue is fixed in cuDNN 7.6.4.
- Addressed the regressions that were introduced in the **`cudnnConvolutionBiasActivationForward`** function in cuDNN 7.6.3. Previously, if this API had different values in destination data buffer and zData

buffer, then incorrect results were computed. This issue has been resolved and now the API will compute correct results even if users provide an arbitrary set of values to the destination data and zData.

- Multi-head attention will now return **`CUDNN_STATUS_ARCH_MISMATCH`** for true-half configuration on devices with compute capability less than 5.3 (for example, most of Maxwell and all of Kepler, etc..), which do not have native hardware support for true half computation. Previously, an error like **`CUDNN_STATUS_EXECUTION_FAILED`** may be triggered or inaccurate results may be produced.

Chapter 3. CUDNN RELEASE NOTES V7.6.3

Key Features and Enhancements

The following features and enhancements have been added to this release:

- The cuDNN 7.6.3 library now supports auto-padding for NHWC layout. The functional behavior, and the benefits of auto-padding as follows:
  - For use cases where C and K dimensions of input and filter Tensors are not multiples of 8, the auto-padding feature increases the Tensor size so that the Tensor dimensions are multiples of 8.
  - With auto-padding the cuDNN library invokes faster kernels, thereby improving the performance.
  - With auto-padding, the performance with NHWC data layout is now comparable to that of the NCHW layout.
- Added support for **`dataType`**=CUDNN_DATA_HALF and **`computePrec`**=CUDNN_DATA_HALF in multi-head attention forward (cudnnMultiHeadAttnForward) and backward (gradient) (cudnnMultiHeadAttnBackwardData and cudnnMultiHeadAttnBackwardWeights) API functions.
- Multi-head attention API now supports bias after the projections on Q, K, V, and O in the cudnnMultiHeadAttnForward() call (backward bias gradient is not yet supported).

  The new feature required a small API change in cudnnSetAttnDescriptor(): the **`cudnnAttnQueryMap_t queryMap`** argument is replaced with **`unsigned attnMode`** to pass various on and off options. This change is backward compatible with earlier API versions.
- Significantly improved the performance in typical multi-head attention use cases in forward inference and training, especially when the vector length of each head is a multiple of 32 up to 128.

- Tensor Core support is added for true half and single precision use cases in multi-head attention. Users may utilize it by setting the mathType argument in cudnnSetAttnDescriptor() to CUDNN_TENSOR_OP_MATH or CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION.
- The multiHeadAttention sample code is added. The sample code includes a compact NumPy/Autograd reference model of the multi-head attention block that computes the forward response and all first-order derivatives. The test code demonstrates how to use the multi-head attention API, access attention weights, and sequence data.
- Improved depth-wise convolution for forward, dgrad, and wgrad under the following conditions:
  - Algorithm is algo1
  - Tensor format for filter is NCHW (wgrad supports NHWC also)
  - Input and outputs are in FP16 and computation is in FP32
  - Filter size: 1x1, 3x3, 5x5, 7x7 (dgrad only supports stride 1)
  - Math type is CUDNN_DEFAULT_MATH
- Improved grouped convolution for **`cudnnConvolutionBackwardFilter()`** in the configuration below:
  - Algorithm is CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1
  - Math type is CUDNN_DEFAULT_MATH
  - Tensor format for filter is NCHW
  - Input and outputs are in FP16 and computation is in FP32
  - Filter size: 1x1, 3x3, 5x5, 7x7
- Improved the performance of grouped convolution, for **`cudnnConvolutionForward()`** in the configuration below:
  - Algorithm is CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM
  - Math type is CUDNN_TENSOR_OP_MATH or CUDNN_TENSOROP_MATH_ALLOW_CONVERSION
  - Tensor format for filter is NHWC
  - Input and outputs are in FP16 and computation is in FP16/ FP32
  - Per group C & K == 4/8/16/32
  - Filter size: 3x3
- Improved the performance of grouped convolution, for **`cudnnConvolutionBackwardFilter()`** in the configuration below:
  - Algorithm is CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1
  - Math type is CUDNN_TENSOR_OP_MATH or CUDNN_TENSOROP_MATH_ALLOW_CONVERSION
  - Tensor format for filter is NHWC
  - Input and outputs are in FP16 and computation is in FP32

- On NVIDIA Volta (compute capability 7.0)
- Per group C & K == 4/8/16/32
- Filter size: 1x1, 3x3

Fixed Issues

The following issues have been fixed in this release:

- Fixed an issue where cudnnMultiHeadAttnBackwardData was producing incorrect results when K sequence length is longer than 32.
- Fixed a race condition in cudnnMultiHeadAttnBackwardData that was producing intermittent incorrect results.
- The function cudnnCTCLoss() produced incorrect gradient result for label whose length is smaller than the maximal sequence length in the batch. This is fixed in cuDNN 7.6.3.

Chapter 4. CUDNN RELEASE NOTES V7.6.2

Key Features and Enhancements

The following features and enhancements have been added to this release:

- Enhanced the performance of 3D deconvolution using cudnnConvolutionBackwardData(), for the following configuration:
  - 2x2x2 filter and 2x2x2 convolution stride.
  - For FP16 for data input and output, and for accumulation.
  - For FP32 for data input and output, and for accumulation.
- Enhanced the performance of 3D convolution using cudnnConvolutionForward(), for the following configuration:
  - Tensor Core for FP16 for data input and output and FP32 accumulation when CUDNN_TENSOR_OP_MATH is set.
  - Tensor Core for FP32 for data input and output and FP32 accumulation when CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION is set.
- Enhanced the functionality of the data type cudnnFusedOps_t by adding the below three enums:
  - CUDNN_FUSED_CONV_SCALE_BIAS_ADD_ACTIVATION
  - CUDNN_FUSED_SCALE_BIAS_ADD_ACTIVATION_GEN_BITMASK, and
  - CUDNN_FUSED_DACTIVATION_FORK_DBATCHNORM

Fixed Issues

The following issues have been fixed in this release:

- In cuDNN 7.6.1, on Volta architecture only, there may be a performance degradation when the function cudnnConvolutionBackwardFilter() is used for 3D convolutions with CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1. This is fixed in cuDNN 7.6.2.

- In cuDNN 7.6.1, on Turing and Pascal architectures, performance may be degraded for cudnnConvolutionBackwardData(), when used with the following conditions:
  - CUDNN_CONVOLUTION_BWD_DATA_ALGO_0 for 3D convolutions
  - **`wDesc`**, **`dyDesc`** and **`dxDesc`** are all in NCDHW
  - Data type configuration is FLOAT_CONFIG (i.e., single precision data and compute)

  This is fixed in cuDNN 7.6.2.
- In cuDNN 7.6.1, in some cases the function cudnnConvolutionBackwardData() may fail with "disallowed mismatches" error on Turing (T4) and Volta(V100) architectures, when used with the configuration below:
  - Algorithm is CUDNN_CONVOLUTION_BWD_DATA_ALGO_1
  - Math type is CUDNN_TENSOR_OP_MATH or CUDNN_TENSOROP_MATH_ALLOW_CONVERSION
  - Tensor format for filter is NCHW
  - Input and outputs are in FP16 and computation is in FP32

  This is fixed in cuDNN 7.6.2.

Chapter 5.
CUDNN RELEASE NOTES V7.6.1

Key Features and Enhancements

The following features and enhancements have been added to this release:

- Performance is enhanced for 3D convolutions using TensorCore for FP16 input and output data types, whenever they are supported. Moreover, for single-precision (FP32) input/output, cuDNN 7.6.1 will use these enhanced kernels whenever possible, and only when cudnnMathType_t is set to CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION. See cudnnConvolutionForward() and cudnnConvolutionBackwardData() and cudnnConvolutionBackwardFilter().
- On Maxwell and Pascal architectures only, the performance of 3D convolutions with the kernel size of 128^3, when used with CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1, is enhanced.
- API logging is fully implemented for the experimental multihead attention API, namely, for the following functions:
  - cudnnCreateAttnDescriptor
  - cudnnDestroyAttnDescriptor
  - cudnnSetAttnDescriptor
  - cudnnGetAttnDescriptor
  - cudnnGetMultiHeadAttnBuffers
  - cudnnGetMultiHeadAttnWeights
  - cudnnMultiHeadAttnForward
  - cudnnMultiHeadAttnBackwardData
  - cudnnMultiHeadAttnBackwardWeights
  - cudnnSetSeqDataDescriptor
  - cudnnGetSeqDataDescriptor
  - cudnnCreateSeqDataDescriptor
  - cudnnDestroySeqDataDescriptor

- Performance of the experimental multihead attention forward API is enhanced. See cudnnMultiHeadAttnForward().
- Performance is enhanced for the fused convolution and fused wgrad fallback path. See cudnnFusedOps_t.

Fixed Issues

The following issues have been fixed in this release:

- In cuDNN 7.6.0, the function cudnnGetConvolutionBackwardDataWorkspaceSize() returns a value for which cudnnConvolutionBackwardData(), when used with CUDNN_CONVOLUTION_BWD_DATA_ALGO_0, returns CUDNN_STATUS_NOT_SUPPORTED. This is fixed in cuDNN 7.6.1 so that now **`cudnnGetConvolutionBackwardDataWorkspaceSize()`** returns a proper value for **`cudnnConvolutionBackwardData()`**.
- In cuDNN 7.6.0 and earlier versions, when all the following conditions are true,
  - RNN model is bi-directional,
  - Cell type is LSTM,
  - cudnnRNNAlgo_t= CUDNN_RNN_ALGO_STANDARD, and
  - Dropout probability was greater than zero,

  then the cudnnRNNBackwardWeights() function produces inaccurate and occasionally non-deterministic results.

  This is fixed in cuDNN 7.6.1.

  An underlying issue, where the same buffer was used for left-to-right and right-to-left directions when re-computing forward dropout results passed from one RNN layer to the next, was the cause of the bug.
- A bug in cuDNN 7.6.0 and earlier versions, in the cudnnRNNForwardTraining() function, related to dropout, is fixed in cuDNN 7.6.1.

  When all the following conditions are true:

  - cudnnRNNAlgo_t=CUDNN_RNN_ALGO_PERSIST_STATIC,
  - cudnnMathType_t is CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION, and
  - input data type is CUDNN_DATA_FLOAT,

  then the FP32-to-FP16 conversion might be applied as a performance optimization.

  When this downconversion is scheduled, a GPU kernel invoked by cudnnDropoutForward() would crash due to incorrect parameters being passed. In this case CUDA runtime reports the "misaligned address" error when reading the data from global memory.

- In cuDNN 7.6.0, on RHEL7 only, the **`/usr/src/cudnn_samples_v7/samples_common.mk`** file is missing. This requires a workaround to compile the cuDNN samples. This is fixed in cuDNN 7.6.1 and the workaround is not needed for cuDNN 7.6.1 .
- In cuDNN 7.6.0, on pre-Volta hardware only, the function cudnnGetConvolutionBackwardFilterWorkspaceSize can erroneously return CUDNN_STATUS_SUCCESS for cudnnConvolutionBackwardFilter for 3D convolutions, using CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1 with NDHWC layout. When this occurs, the **`cudnnConvolutionBackwardFilter`** function will process the data using a kernel that expects the data in NCDHW layout (the only format supported by **`wDesc`** in this case), leading to incorrect results. In cuDNN 7.6.1, this is fixed so that **`cudnnGetConvolutionBackwardFilterWorkspaceSize`** will now return CUDNN_STATUS_NOT_SUPPORTED.
- In cuDNN 7.5.x and 7.6.0 for Jetson platform, in some cases the function cudnnConvolutionBackwardData , when used with CUDNN_CONVOLUTION_BWD_DATA_ALGO_WINOGRAD, might return incorrect results. This is fixed in cuDNN 7.6.1.
- When the data type configuration is FLOAT_CONFIG, then **`cudnnGetConvolution*Algorithm()`**, for a few convolution sizes, incorrectly returns a slow algorithm for the Pascal architecture. This is fixed in cuDNN 7.5.0 and later versions.
- When using the fusedOps API with the enum CUDNN_FUSED_SCALE_BIAS_ACTIVATION_CONV_BNSTATS or CUDNN_FUSED_SCALE_BIAS_ACTIVATION_WGRAD, and when input tensor is in NCHW format or is not fully-packed, then incorrect results may be produced. This is now fixed in cuDNN 7.6.1.

Known Issues

The following issues and limitations exist in this release:

- Algorithms returned by **`cudnnGetConvolution*Algorithm()`** may, in some limited use cases, fail to execute when they are actually run. This is a cuDNN library-wide issue and applies for convolution forward, convolution backward data, and convolution backward filter operations. This issue is also present in versions prior to cuDNN 7.6.1.
- When the input and output tensors are in NHWC and the filter is 1x1 and NCHW, the performance of the function cudnnConvolutionBackwardData() might be degraded.
- In cuDNN 7.6.1, when using the experimental multi-head attention API, it is possible that the forward and backward paths produce different results for the BERT model, when the batch size is greater than one and/or the number of heads is greater than one.

- In cuDNN 7.6.1, on Volta architecture only, there may be a performance degradation when the function cudnnConvolutionBackwardFilter() is used for 3D convolutions with CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1.
- In cuDNN 7.6.1, on Turing and Pascal architectures, performance may be degraded for cudnnConvolutionBackwardData(), when used with the following conditions:
  - CUDNN_CONVOLUTION_BWD_DATA_ALGO_0 for 3D convolutions
  - **`wDesc`**, **`dyDesc`** and **`dxDesc`** are all in NCDHW
  - Data type configuration is FLOAT_CONFIG (i.e., single precision data and compute)

Chapter 6.
CUDNN RELEASE NOTES V7.6.0

Key Features and Enhancements

The following features and enhancements have been added to this release:

- A new API is introduced for fused ops, which can accelerate many use cases in ResNet-like networks. With this new API it is now possible to execute various fused operations such as apply per channel scale and bias, perform activation, compute convolution, and generate batchnorm statistics. Below is a list of supported datatype and functions in this API:

  Datatypes:

  1. cudnnFusedOpsVariantParamPack_t
  2. cudnnFusedOpsConstParamPack_t
  3. cudnnFusedOpsPlan_t
  4. cudnnFusedOps_t
  5. cudnnFusedOpsConstParamLabel_t
  6. cudnnFusedOpsPointerPlaceHolder_t
  7. cudnnFusedOpsVariantParamLabel_t

  Functions:

  1. cudnnCreateFusedOpsConstParamPack
  2. cudnnDestroyFusedOpsConstParamPack
  3. cudnnSetFusedOpsConstParamPackAttribute
  4. cudnnGetFusedOpsConstParamPackAttribute
  5. cudnnCreateFusedOpsVariantParamPack
  6. cudnnDestroyFusedOpsVariantParamPack
  7. cudnnSetFusedOpsVariantParamPackAttribute
  8. cudnnGetFusedOpsVariantParamPackAttribute
  9. cudnnCreateFusedOpsPlan

10. cudnnDestroyFusedOpsPlan
11. cudnnMakeFusedOpsPlan
12. cudnnFusedOpsExecute

- Improved the performance of grouped convolution layers in ResNeXt-50, for cudnnConvolutionBackwardData() in the configuration below:
  - On NVIDIA Volta (compute capability 7.0)
  - Algorithm is CUDNN_CONVOLUTION_BWD_DATA_ALGO_1
  - Stride of 1
  - Math type is CUDNN_TENSOR_OP_MATH or CUDNN_TENSOROP_MATH_ALLOW_CONVERSION
  - Tensor format for filter is NHWC
  - Input and outputs are in FP16 and computation is in FP32
- A new API is introduced to enhance the inference time. With this new API it is now possible to separate the filter layout transformation that was applied on every call, which in turn leads to inference time enhancement. Below is a list of supported datatype and functions in this API.
  1. cudnnReorderType_t
  2. cudnnReorderFilterAndBias
  3. cudnnSetConvolutionReorderType
  4. cudnnGetConvolutionReorderType
- Performance is enhanced (by selecting a faster kernel) on NVIDIA T4 cards for INT8x4 and INT8x32.

Fixed Issues

The following issues have been fixed in this release:

- In cuDNN 7.5.0 and cuDNN 7.5.1, a bug in the cudnnRNNBackwardData() function affected the thread synchronization. This effect is limited to only the first iteration of the loop, and only in some paths. This occurs when using the function with the CUDNN_RNN_ALGO_PERSIST_STATIC method. This is fixed in cuDNN 7.6.0.

Known Issues

The following issues and limitations exist in this release:

- The **`cudnnConvolutionBackwardData()`** function for CUDNN_CONVOLUTION_BWD_DATA_ALGO_0 fails with CUDNN_STATUS_NOT_SUPPORTED when the input size is large.
- A general known issue for cuDNN library: the Tensor pointers and the filter pointers require at a minimum 4-byte alignment, including for FP16 or INT8 data.
- On RHEL7 only, the /**`usr/src/cudnn_samples_v7/samples_common.mk`** file is missing. This will prevent compiling the cuDNN samples. The workaround

is to copy the below contents into “samples_common.mk” text file and place this file in the “/usr/src/cudnn_samples_v7/” directory, so that the **/usr/src/cudnn_samples_v7/samples_common.mk** file exists.

```
# Setting SMS for all samples
# architecture

ifneq ($(TARGET_ARCH), ppc64le)
CUDA_VERSION := $(shell cat $(CUDA_PATH)/include/cuda.h |grep "define
 CUDA_VERSION" |awk '{print $$3}')
else
CUDA_VERSION := $(shell cat $(CUDA_PATH)/targets/ppc64le-linux/include/
cuda.h |grep "define CUDA_VERSION" |awk '{print $$3}')
endif

#Link against cublasLt for CUDA 10.1 and up.
CUBLASLT:=false
ifeq ($(shell test $(CUDA_VERSION) -ge 10010; echo $$?),0)
CUBLASLT:=true
endif
$(info Linking agains cublasLt = $(CUBLASLT))

ifeq ($(CUDA_VERSION),8000 )
SMS_VOLTA =
else
ifneq ($(TARGET_ARCH), ppc64le)
ifeq ($(CUDA_VERSION), $(filter $(CUDA_VERSION), 9000 9010 9020))
SMS_VOLTA ?= 70
else
ifeq ($(TARGET_OS), darwin)
SMS_VOLTA ?= 70
else
SMS_VOLTA ?= 70 72 75
endif #ifneq ($(TARGET_OS), darwin)
endif #ifeq ($(CUDA_VERSION), $(filter $(CUDA_VERSION), 9000 9010 9020))
else
SMS_VOLTA ?= 70
endif #ifneq ($(TARGET_ARCH), ppc64le)
endif #ifeq ($(CUDA_VERSION),8000 )
SMS ?= 30 35 50 53 60 61 62 $(SMS_VOLTA)
```

Chapter 7. CUDNN RELEASE NOTES V7.5.1

Key Features and Enhancements

The following features and enhancements have been added to this release:

- The function cudnnMultiHeadAttnForward() is now enabled to sweep through all the time-steps in a single API call. This is indicated by a negative value of the **`currIdx`** argument in the inference mode, i.e., when **`reserveSpace=NULL`** so that either cudnnMultiHeadAttnBackwardData() or cudnnMultiHeadAttnBackwardWeights() will not be invoked. This sweep mode can be used to implement self-attention on the encoder side of the transformer model.

Fixed Issues

The following issues have been fixed in this release:

- In cuDNN 7.5.0, using the static link for cudnnConvolutionBiasActivationForward() function may result in CUDNN_STATUS_NOT_SUPPORTED error message. The workaround is to perform a whole-archive link. This issue is fixed in cuDNN 7.5.1.
- In cuDNN 7.5.0 and 7.4.x, in some cases of input images with large dimensions, the 3D forward convolution operations with CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM will cause a crash with “illegal memory access” error. This is fixed in cuDNN 7.5.1.
- In cuDNN 7.5.0, setting **`attnDropoutDesc=NULL`** in cudnnSetAttnDescriptor() triggered a segmentation fault in **`cudnnMultiHeadAttnForward()`**, even though the user is required to set it to NULL in the inference mode. This is fixed in cuDNN 7.5.1.

Known Issues

The following issues and limitations exist in this release:

- In cuDNN7.5 and cudnn7.5.1, image size smaller than filter size is unsupported, even with sufficient padding.

Chapter 8. CUDNN RELEASE NOTES V7.5.0

Key Features and Enhancements

The following features and enhancements have been added to this release:

- In cudnnConvolutionForward() for 2D convolutions, for **`wDesc`** NCHW, the IMPLICIT_GEMM algorithm (algo 0) now supports the Data Type Configuration of INT8x4_CONFIG, and INT8x4_EXT_CONFIG also.
- A new set of APIs are added to provide support for Multi-Head Attention computation. The following is a list of the new functions and data types:

  Datatypes:

  - cudnnSeqDataAxis_t
  - cudnnMultiHeadAttnWeightKind_t
  - cudnnSeqDataDescriptor_t
  - cudnnWgradMode_t
  - cudnnAttnQueryMap_t
  - cudnnAttnDescriptor_t

  Functions:

  - cudnnCreateAttnDescriptor
  - cudnnDestroyAttnDescriptor
  - cudnnSetAttnDescriptor
  - cudnnGetAttnDescriptor
  - cudnnGetMultiHeadAttnBuffers
  - cudnnGetMultiHeadAttnWeights
  - cudnnMultiHeadAttnForward
  - cudnnMultiHeadAttnBackwardData
  - cudnnMultiHeadAttnBackwardWeights
  - cudnnSetSeqDataDescriptor

  - cudnnGetSeqDataDescriptor
  - cudnnCreateSeqDataDescriptor
  - cudnnDestroySeqDataDescriptor
- A new set of APIs for general tensor folding is introduced. The following is a list of the new functions and data types:

  Datatypes:

  - cudnnTensorTransformDescriptor_t
  - cudnnFoldingDirection_t

  Functions:

  - cudnnTransformTensorEx
  - cudnnCreateTensorTransformDescriptor
  - cudnnDestroyTensorTransformDescriptor
  - cudnnInitTransformDest
  - cudnnSetTensorTransformDescriptor
  - cudnnGetTensorTransformDescriptor
- A new set of APIs, and enhancements for the existing APIs, are introduced for RNNs. The following is the list of the new and enhanced functions and data types:

  Datatypes:

  - cudnnRNNBiasMode_t (new)
  - cudnnRNNMode_t (enhanced)

  Functions:

  - cudnnSetRNNBiasMode (new)
  - cudnnGetRNNBiasMode (new)
  - cudnnGetRNNLinLayerBiasParams (enhanced)
- All **`cudnnRNNForward/Backward*`** functions are enhanced to support FP16 math precision mode when both input and output are in FP16. To switch to FP16 math precision, set the **`mathPrec`** parameter in **`cudnnSetRNNDescriptor`** to CUDNN_DATA_HALF. To switch to FP32 math precision, set the **`mathPrec`** parameter in **`cudnnSetRNNDescriptor`** to CUDNN_DATA_FLOAT. This feature is only available for CUDNN_ALGO_STANDARD and for the compute capability 5.3 or higher.
- Added support for INT8x4 and INT8x32 data type for cudnnPoolingForward. Using these will provide improved performance over scalar data type.

Fixed Issues

The following issues have been fixed in this release:

- When the following is true for the **`cudnnConvolutionBackwardData()`** function:

- used with CUDNN_CONVOLUTION_BWD_DATA_ALGO_FFT_TILING, and
- **convDesc**'s vertical stride is exactly 2, and
- the vertical padding is a multiple of 2, and
- the filter height is a multiple of 2

OR

- used with CUDNN_CONVOLUTION_BWD_DATA_ALGO_FFT_TILING, and
- **convDesc**'s horizontal stride is exactly 2, and
- the horizontal padding is a multiple of 2, and
- the filter width is a multiple of 2

then the resulting output is incorrect. This issue was present in cuDNN 7.3.1 and later. This is fixed in cuDNN 7.5.0.

- The **mathPrec** parameter in **cudnnSetRNNDescriptor** is reserved for controlling math precision in RNN, but was not checked or enforced. This parameter is now strictly enforced. As a result, the following applies:
  - For the input/output in FP16, the parameter **mathPrec** can be CUDNN_DATA_HALF or CUDNN_DATA_FLOAT.
  - For the input/output in FP32, the parameter **mathPrec** can only be CUDNN_DATA_FLOAT, and
  - For the input/output in FP64, double type, the parameter **mathPrec** can only be CUDNN_DATA_DOUBLE.
- Users upgrading to cuDNN 7.4 may see insufficiently small values returned from the function cudnnGetConvolutionBackwardFilterWorkspaceSize () for dimensions 5 and greater, resulting in a CUDNN_STATUS_EXECUTION_FAILED error message. In cuDNN 7.4, the workaround for this issue is to calculate the workspace by using the formula below:

```
Let M be the product of output tensor (gradDesc) dimensions starting at 1.
Let N be the output tensor dimension 0.
Let Mp = (M+31)/32
Let Np = (N+31)/32
W = 2 * M * N * sizeof(int) is the workspace that should be used.
```

  This is fixed.
- In earlier cuDNN versions, when all the conditions below are true:
  - 3-D convolution
  - Batch size > 1
  - Algorithm is "CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1"
  - **convDesc**'s **dataType** is CUDNN_DATA_HALF,

  then, calls to **cudnnConvolutionBackwardFilter()** may produce incorrect (and non-deterministic) results. This is fixed in cuDNN 7.5.0.

- In cuDNN 7.4.2, for some cases the 3D convolution resulted in a reduced performance on Turing GPUs, compared to the previous cuDNN releases. This is fixed.
- For **int8x32** datatype, the function cudnnSetTensor4dDescriptorEx erroneously returns CUDNN_STATUS_BAD_PARAM. Now it is fixed in cuDNN 7.5 so it no longer returns bad param.
- In cuDNN 7.4.1 and 7.4.2, when cudnnBatchNormMode_t is set to CUDNN_BATCHNORM_SPATIAL_PERSISTENT and the input/output tensors are in NHWC format and of CUDNN_DATA_HALF datatype, then, on Windows only, the **cudnnBatchNormalization*Ex** functions are supported only with the device in TCC mode. See Tesla Compute Cluster Mode for Windows .

  Starting with cuDNN 7.5.0, the following checks are added for the driver mode on Windows. If on Windows and not in TCC mode:

  - The functions will fallback to a slower implementation if **bnOps** in the **cudnnBatchNormalization*Ex** function is set to CUDNN_BATCHNORM_OPS_BN.
  - If **bnOps** is set to CUDNN_BATCHNORM_OPS_BN_ACTIVATION, or CUDNN_BATCHNORM_OPS_BN_ADD_ACTIVATION, the CUDNN_STATUS_NOT_SUPPORTED is returned.
- In cuDNN 7.4.2, in some cases the cudnnConvolutionBackwardData() function, when used with NHWC tensor format, resulted in the “disallowed mismatches” error. This is fixed.
- In some cases, using cudnnConvolutionBiasActivationForward() with **GroupCount() > 1** and **xDesc**'s data type is CUDNN_DATA_HALF will produce incorrect results for all groups except the first. This is fixed.
- When using cuDNN 7.3.1 on Quadro P4000, when calling the cudnnConvolutionForward() function with CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED algorithm, there was a small chance of seeing intermittent inaccurate results. This is fixed.
- When cudnnConvolutionForward() is called with these settings: Datatype is CUDNN_DATA_INT8x4, Convolution is 2D, architecture is **sm_61**, filter size is larger than 8x8, then incorrect result and potential illegal memory access error occurs. This is fixed.
- For **sm_72** and **sm_75**, the function cudnnConvolutionBiasActivationForward(), when used with INT8x32, failed to run. This is fixed.
- In the function **cudnnSetRNNDataDescriptor** , if API logging is turned on, the **seqLengthArray** field in the log may not display the correct number of array elements. This is fixed.
- For the batchNorm functions **cudnnBatchNormalization{Backward|BackwardEx|ForwardInference|ForwardTraining|ForwardTrainingEx}**, the value of **epsilon** is required to be greater or equal to CUDNN_BN_MIN_EPSILON

which was defined in the **cudnn.h** file to the value 1e-5. This threshold value is now lowered to 0.0 to allow a wider range of **epsilon** value. However, users should still choose the **epsilon** value carefully, since a too small a value of **epsilon** may cause batchNormalization to overflow when the input data's standard deviation is close to 0.

- Some Grouped Convolutions (particularly those used in Depthwise-Separable convolutions) may return INTERNAL_ERROR if they have all inputs/outputs as NHWC-packed and do not match one of the following criteria:
  - filter_height = 1, filter_width = 1, vertical_conv_stride = 1, horizontal_conv_stride = 1
  - filter_height = 3, filter_width = 3, vertical_conv_stride = 1, horizontal_conv_stride = 1
  - filter_height = 3, filter_width = 3, vertical_conv_stride = 2, horizontal_conv_stride = 2

Known Issues

The following issues and limitations exist in this release:

- The RNN persist-static algorithm returns incorrect results for GRU problems in backwards mode, when the hidden size is greater than 1024. Due to this, RNN persist-static algorithm is disabled in cuDNN 7.5.0. Users with such GRU problems are advised to use the standard or persist-dynamic RNN algorithms. See cudnnRNNAlgo_t(). This note applies to all previous cuDNN 7 releases.
- The function cudnnConvolutionBackwardFilter(), when used with CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1, returns the error **"Uninitialized __global__ memory read of size 4".**

Chapter 9. CUDNN RELEASE NOTES V7.4.2

Fixed Issues

The following issues have been fixed in this release:

- In some cases when the data is in CUDNN_DATA_HALF and NHWC, illegal memory access may occur for **`cudnnBatchNormalization*`** functions in the cuDNN 7.4.1 library. This is now fixed.
- When the data is in CUDNN_DATA_HALF and NHWC, for **`cudnnBatchNormalization*`** functions when (N*H*W) is large and odd number, the output may contain wrong results. This is fixed.
- When calling the **`cudnnConvolutionBiasActivationForward()`** function with the **`algo`** parameter set to CUDNN_CONVOLUTION_FWD_ALGO_FFT and the **`activationDesc`** parameter set to CUDNN_ACTIVATION_RELU and sufficiently large inputs, the ReLU operation is not applied and negative values are passed through to the output. This issue is now fixed. This issue was present in all previous cuDNN versions.
- Performance regression was introduced in cuDNN 7.4.1 for **`cudnnConvolutionBwdFilterAlgo_t()`** function with CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1 algorithm. This is fixed.

Known Issues

The following issues and limitations exist in this release:

- When **`cudnnBatchNormMode_t`** is set to CUDNN_BATCHNORM_SPATIAL_PERSISTENT and the input/output tensors are in NHWC format and of CUDNN_DATA_HALF datatype, then, **on Windows only**, the **`cudnnBatchNormalization*Ex`** functions are supported only with the device in TCC mode. See Tesla Compute Cluster Mode for Windows. This issue is not present on Linux systems. This issue is present in cuDNN 7.4.1 and this current version.

- In some cases the 3D convolution will have a reduced performance on Turing GPUs, compared to the previous cuDNN releases.
- The functions **`cudnnGetConvolutionForwardAlgorithm_v7()`** and **`cudnnGetConvolutionForwardWorkspaceSize()`** will return CUDNN_STATUS_SUCCESS, but the execution of the convolution returns CUDNN_STATUS_NOT_SUPPORTED. This issue is present in cuDNN 7.2.2 library and later versions.

Chapter 10. CUDNN RELEASE NOTES V7.4.1

Key Features and Enhancements

The following enhancements have been added to this release:

- Added a new family of fast NHWC batch normalization functions. See the following five new functions and one new type descriptor:
  - cudnnGetBatchNormalizationForwardTrainingExWorkspaceSize(),
  - cudnnBatchNormalizationForwardTrainingEx,
  - cudnnGetBatchNormalizationBackwardExWorkspaceSize(),
  - cudnnBatchNormalizationBackwardEx(),
  - cudnnGetBatchNormalizationTrainingExReserveSpaceSize() functions, and
  - cudnnBatchNormOps_t type descriptor
- For API Logging, a conversion specifier for the process id is added. With this, the process id can be included in the log file name. See API Logging.
- Performance of **`cudnnPoolingBackward()`** is enhanced for the average pooling when using NHWC data format--for both the CUDNN_POOLING_AVERAGE_COUNT_INCLUDE_PADDING and CUDNN_POOLING_AVERAGE_COUNT_EXCLUDE_PADDING cases of **`cudnnPoolingMode_t`**.
- Performance of the strided convolution in **`cudnnConvolutionBackwardData()`** is enhanced when the filter is in NHWC format and the data type is TRUE_HALF_CONFIG or PSEUDO_HALF_CONFIG or FLOAT_CONFIG. For strides **`u,v < r,s`** the performance is further enhanced.
- Significantly improved the performance of cudnnConvolutionForward(), cudnnConvolutionBackwardData() & cudnnConvolutionBackwardFilter() functions on RCNN models such as Fast RCNN, Faster RCNN, & Mask RCNN.

Fixed Issues

The following issues have been fixed in this release:

- The following set up was giving “Misaligned Address” error in cuDNN 7.3.x. This is fixed in cuDNN 7.4.1: For the cudnnConvolutionForward() function with the CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM algorithm, in the data type configuration of PSEUDO_HALF_CONFIG, when the input and output tensors are in in NHWC and the filter is 1x1 and NCHW, and Tensor Op is enabled.
- For a few convolution sizes for ALGO_0 and ALGO_1, the performance of the function cudnnConvolutionBackwardFilter() was degraded in cuDNN 7.3.1. This is now fixed.
- Fixed. In cuDNN 7.3.1 the function cudnnAddTensor was computing incorrect results when run on GPUs with the compute capability < 6.0 (prior to Pascal).

Known Issues

The following issues and limitations exist in this release:

- When calling the **`cudnnConvolutionBiasActivationForward()`** function with the **`algo`** parameter set to CUDNN_CONVOLUTION_FWD_ALGO_FFT and the **`activationDesc`** parameter set to CUDNN_ACTIVATION_RELU and sufficiently large inputs, the ReLU operation is not applied and negative values are passed through to the output. This issue is present in all previous cuDNN versions.

Chapter 11. CUDNN RELEASE NOTES V7.3.1

Key Features and Enhancements

The following enhancements have been added to this release:

- The FFT tiling algorithms for convolution have been enhanced to support strided convolution. In specific, for the algorithms CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING and CUDNN_CONVOLUTION_BWD_DATA_ALGO_FFT_TILING, the **`convDesc`**'s vertical and horizontal filter stride can be 2 when neither the filter width nor the filter height is 1.
- The CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD algorithm for **`cudnnConvolutionForward()`** and **`cudnnConvolutionBackwardData()`** now give superior performance for Volta architecture. In addition, the mobile version of this algorithm in the same functions gives superior performance for Maxwell and Pascal architectures.
- Dilated convolutions now give superior performance for **`cudnnConvolutionForward()`**, **`cudnnConvolutionBackwardData()`**, and **`cudnnConvolutionBackwardFilter()`** on Volta architecture, in some cases.

Known Issues and Limitations

The following issues and limitations exist in this release:

- For the **`cudnnConvolutionForward()`**, when using a 1x1 filter with input and output tensors of **`NHWC`** format and of CUDNN_DATA_HALF (half precision) type, and the filter format is **`NCHW`**, with compute type of float, cuDNN will generate incorrect results.
- On Quadro P4000, when calling **`cudnnConvolutionForward()`** function with CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED algorithm, there may be a small chance of seeing intermittent inaccurate results.

- When using **`cudnnConvolutionBackwardFilter()`** with CUDNN_CONVOLUTION_BWD_FILTER_ALGO_0 in mixed precision computation, with input/output in CUDNN_DATA_HALF (half precision) and compute type of float, when the number of batches (N) is larger than 1 the results might include INF due to an intermediate down-convert to half float. In other words, with an accumulation of float for all intermediate values (such as in CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1) the result will be a finite half precision float. This limitation also exists in all previous cuDNN versions.

Fixed Issues

The following issues have been fixed in this release:

- Fixed a pointer arithmetic integer overflow issue in RNN forward and backward functions, when sequence length and mini-batch size are sufficiently large.
- When tensor cores are enabled in cuDNN 7.3.0, the **`cudnnConvolutionBackwardFilter()`** calculations were performing an illegal memory access when K and C values are both non-integral multiples of 8. This issue is fixed.
- For the CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1 algorithm in **`cudnnConvolutionBackwardFilter()`**, on Volta, the tensor operations were occasionally failing when the filter spatial size (filter **`h`** * filter **`w`**) was greater than 64. This issue is fixed.
- While running cuDNN 7.3.0 on Turing with CUDA 10.0, r400 driver, the functions **`cudnnRNNForwardTraining(Ex)`** and **`cudnnRNNForwardInference(Ex)`** errored out returning CUDNN_STATUS_NOT_SUPPORTED. This issue is fixed.
- In cuDNN 7.3.0, when using CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1 with tensor data or filter data in **`NHWC`** format, the function might have resulted in a silent failure. This is now fixed.

Chapter 12. CUDNN RELEASE NOTES V7.3.0

Key Features and Enhancements

The following enhancements have been added to this release:

- Support is added to the following for the dilated convolution, for **`NCHW`** and **`NHWC`** filter formats:
  - **`cudnnConvolutionForward()`** for 2D, CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM,
  - **`cudnnConvolutionBackwardData()`** for 2D, CUDNN_CONVOLUTION_BWD_DATA_ALGO_1, and
  - **`cudnnConvolutionBackwardFilter()`** for 2D, CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1

  For these supported cases, the dilated convolution is expected to offer superior speed, compared to the existing dilated convolution with algo 0.
- Grouped convolutions for depth-wise separable convolutions are optimized for the following NHWC formats: HHH (input: Half, compute: Half, output: Half), HSH, and SSS.
- While using CUDNN_TENSOR_OP_MATH or CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION, with the tensor cores, the **`c`** and **`k`** dimensions of the tensors are now padded to multiples of 8 (as needed), to allow a tensor core kernel to run.
- The CUDNN_BATCHNORM_SPATIAL_PERSISTENT algo is enhanced in **`cudnnBatchNormalizationForwardTraining()`** and **`cudnnBatchNormalizationBackward()`** to propagate NaN-s or Inf-s as in a pure floating point implementation (the "persistent" flavor of the batch normalization is optimized for speed and it uses integer atomics for inter thread-block reductions). In earlier versions of cuDNN we recommended invoking **`cudnnQueryRuntimeError()`** to ensure no overflow was encountered. When it happened, the best practice was to discard the results, and use

CUDNN_BATCHNORM_SPATIAL instead, as some results generated by CUDNN_BATCHNORM_SPATIAL_PERSISTENT could be finite but invalid. This behavior is now corrected: NaN-s and/or Inf-s are consistently output when intermediate results are out of range. The refined implementation simulates math operations on special floating point values, for example, +Inf-Inf=NaN.

Known Issues and Limitations

Following issues and limitations exist in this release:

- When tensor cores are enabled in cuDNN 7.3.0, the wgrad calculations will perform an illegal memory access when K and C values are both non-integral multiples of 8. This will not likely produce incorrect results, but may corrupt other memory depending on the user buffer locations. This issue is present on Volta & Turing architectures.
- Using **`cudnnGetConvolution*_v7`** routines with **`cudnnConvolutionDescriptor_t`** set to CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION leads to incorrect outputs. These incorrect outputs will consist only of CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION cases, instead of also returning the performance results for both DEFAULT_MATH and CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION cases.

Fixed Issues

The following issues have been fixed in this release:

- Using **`cudnnConvolutionBackwardData()`** with CUDNN_CONVOLUTION_BWD_DATA_ALGO_WINOGRAD algorithm produced incorrect results due to an incorrect filter transform. This issue was present in cuDNN 7.2.1.
- For INT8 type, with **`xDesc`** and **`yDesc`** of NHWC format, the **`cudnnGetConvolutionForwardAlgorithm_v7`** function was incorrectly returning CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM as a valid algorithm. This is fixed.
- **`cudnnConvolutionForward()`** using CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD intermittently produced incorrect results in cuDNN 7.2, due to a race condition. This issue is fixed.
- When running **`cudnnConvolutionBackwardFilter()`** with NHWC filter format, when **`n`**, **`c`**, and **`k`** are all multiple of 8, and when the **`workSpace`** input is exactly as indicated by **`cudnnGetConvolutionBackwardFilterWorkspaceSize()`**, leads to error in cuDNN 7.2. This is fixed.
- When the user runs **`cudnnRNNForward`*** or **`cudnnRNNBackward`*** with FP32 input/output on sm_70 or sm_72, with RNN descriptor's **`algo`** field set to CUDNN_RNN_ALGO_PERSIST_STATIC, and **`cudnnMathType_t`** type set to

CUDNN_TENSOR_OP_MATH via **cudnnSetRNNMatrixMathType**, then the results were incorrect. This is fixed.

- When the user runs **cudnnRNNForward*** or **cudnnRNNBackward*** with FP32 input/output on sm_70 or sm_72, with RNN descriptor's **algo** field set to CUDNN_RNN_ALGO_PERSIST_STATIC, and **cudnnMathType_t** type set to CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION via **cudnnSetRNNMatrixMathType**, then the resulting performance was suboptimal. This is fixed.
- Convolution routines with filter format as NHWC require both input and output formats to be NHWC. However, in cuDNN 7.2 and earlier, this condition was not being checked for, as a result of which silent failures may have occurred. This is fixed in 7.3.0 to correctly return CUDNN_STATUS_NOT_SUPPORTED.

Chapter 13. CUDNN RELEASE NOTES V7.2.1

Key Features and Enhancements

The following enhancements have been added to this release:

- The following new functions are added to provide support for the padding mask for the **cudnnRNN*** family of functions:
  - **cudnnSetRNNPaddingMode()**: Enables/disables the padded RNN input/output.
  - **cudnnGetRNNPaddingMode()**: Reads the padding mode status.
  - **cudnnCreateRNNDataDescriptor()** and **cudnnDestroyRNNDataDescriptor()**: Creates and destroys, respectively, **cudnnRNNDataDescriptor_t**, an RNN data descriptor.
  - **cudnnSetRNNDataDescriptor()** and **cudnnGetRNNDataDescriptor()**: Initializes and reads, respectively, the RNN data descriptor.
  - **cudnnRNNForwardTrainingEx()**: An extended version of the **cudnnRNNForwardTraining()** to allow for the padded (unpacked) layout for the input/output.
  - **cudnnRNNForwardInferenceEx()**: An extended version of the **cudnnRNNForwardInference()** to allow for the padded (unpacked) layout for the input/output.
  - **cudnnRNNBackwardDataEx()**: An extended version of the **cudnnRNNBackwardData()** to allow for the padded (unpacked) layout for the input/output.
  - **cudnnRNNBackwardWeightsEx()**: An extended version of the **cudnnRNNBackwardWeights()** to allow for the padded (unpacked) layout for the input/output.
- Added support for cell clipping in cuDNN LSTM. The following new functions are added:
  - **cudnnRNNSetClip()** and **cudnnRNNGetClip()**: Sets and retrieves, respectively, the LSTM cell clipping mode.

- Accelerate your convolution computation with this new feature: When the input channel size **c** is a multiple of 32, you can use the new data type CUDNN_DATA_INT8x32 to accelerate your convolution computation.

  
  This new data type CUDNN_DATA_INT8x32 is only supported by sm_72.

- Enhanced the family of **cudnnFindRNN*** functions. The **findIntensity** input to these functions now enable the user to control the overall runtime of the RNN find algorithms, by selecting a percentage of a large Cartesian product space to be searched.
- A new mode CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION is added to **cudnnMathType_t**. The computation time for FP32 tensors can be reduced by selecting this mode.
- The functions **cudnnRNNForwardInference()**, **cudnnRNNForwardTraining()**, **cudnnRNNBackwardData()**, and **cudnnRNNBackwardWeights()** will now perform down conversion of FP32 input/output only when CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION is set.
- Improved the heuristics for **cudnnGet*Algorithm()** functions.

Known Issues and Limitations

Following issues and limitations exist in this release:

- For FP16 inputs, the functions **cudnnGetConvolutionForwardAlgorithm()**, **cudnnGetConvolutionBackwardDataAlgorithm()**, and **cudnnGetConvolutionBackwardFilterAlgorithm()** will obtain a slower algorithm.
- For cases where **beta** is not equal to zero, and when the input channel size is greater than 65535, then the below **cudnnConvolutionBackwardFilter()** algorithms may return EXECUTION_FAILED error:
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_0,
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1, and
  - CUDNN_CONVOLUTION_BWD_FILTER_ALGO_3
- **This is a rare occurrence:** When **beta** is not equal to zero, the function **cudnnFindConvolutionBackwardFilterAlgorithm()** may not return the fastest algorithm available for **cudnnConvolutionBackwardFilter()**.
- Grouped convolutions are not supported in the TRUE_HALF_CONFIG (**convDesc** is CUDNN_DATA_HALF) data type configuration. As a workaround, the PSEUDO_HALF_CONFIG (**convDesc** is CUDNN_DATA_FLOAT) data type configuration can be used without losing any precision.
- For the **cudnnConvolutionBiasActivationForward()** function, if the input **cudnnActivationMode_t** is set to enum value CUDNN_ACTIVATION_IDENTITY,

then the input **cudnnConvolutionFwdAlgo_t** must be set to the enum value CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM.

- When the user runs **cudnnRNNForward*** or **cudnnRNNBackward*** with FP32 input/output, on sm_70 or sm_72, with RNN descriptor's **algo** field set to **CUDNN_RNN_ALGO_PERSIST_STATIC**, and math type set to **CUDNN_TENSOR_OP_MATH** via **cudnnSetRNNMatrixMathType()**, then the results are incorrect.
- When the user runs **cudnnRNNForward*** or **cudnnRNNBackward*** with FP32 input/output, on sm_70 or sm_72, with RNN descriptor's **algo** field set to **CUDNN_RNN_ALGO_PERSIST_STATIC**, and math type set to **CUDNN_TENSOR_OP_MATH_ALLOW_CONVERSION** via **cudnnSetRNNMatrixMathType()**, then the resulting performance is suboptimal.

Fixed Issues

The following issues have been fixed in this release:

- The **cudnnConvolutionBackwardData()** function produced incorrect result under these conditions:
  - The **algo** input is set to **CUDNN_CONVOLUTION_BWD_DATA_ALGO_1** in **cudnnConvolutionBwdDataAlgo_t**, and
  - **CUDNN_TENSOR_OP_MATH** is selected.

  Under above conditions, the dgrad computation was giving incorrect results when the data is not packed and the data format is NCHW. This is fixed.
- When the **cudnnConvolutionFwdAlgo_t()** was set to **CONVOLUTION_FWD_ALGO_FFT_TILING** then the function **cudnnConvolutionForward()** was leading to illegal memory access. This is now fixed.
- **cudnnPoolingBackward()** was failing when using a large kernel size used for 'global_pooling' with NHWC I/O layout. This is fixed.
- The below two items are fixed: If you set RNN mathtype to CUDNN_TENSOR_OP_MATH, and run RNN on **sm6x** or earlier hardware:
  - a. You may have received CUDNN_STATUS_NOT_SUPPORTED when algo selected is CUDNN_RNN_ALGO_STANDARD or CUDNN_RNN_ALGO_PERSIST_STATIC.
  - b. You may have received incorrect results when algo selected is CUDNN_RNN_ALGO_PERSIST_DYNAMIC.
- If you passed in variable sequence length input tensor to **cudnnRNNForwardInference()**, **cudnnRNNForwardTraining()**, **cudnnRNNBackwardData()**, and used CUDNN_RNN_ALGO_PERSIST_STATIC or CUDNN_RNN_ALGO_PERSIST_DYNAMIC, then you may have received incorrect

results. Now this is being checked, and CUDNN_STATUS_NOT_SUPPORTED will be returned.

Chapter 14. CUDNN RELEASE NOTES V7.1.4

Key Features and Enhancements

The following enhancements have been added to this release:

- Improved performance for some cases of data-gradient convolutions and maxpooling. This is expected to improve performance of ResNet-50 like networks.
- The runtime of the RNN Find algorithm suite is improved in v7.1.4 resulting in slightly improved runtime of **`cudnnFindRNN***AlgorithmEx`**.

Known Issues

Following are known issues in this release:

- **`cudnnGet`** picks a slow algorithm that does not use Tensor Cores on Volta when inputs are FP16 and it is possible to do so.
- The **`cudnnConvolutionBackwardFilter()`** function may output incorrect results for **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO_FFT_TILING`** when the convolution mode is **`CUDNN_CONVOLUTION`**. This function should not be used in this mode.

Fixed Issues

The following issues have been fixed in this release:

- **`cudnnAddTensorNd`** might cause a segmentation fault if called with bad arguments (e.g. null pointer), this issue is in 7.1.3 only and fixed in 7.1.4.
- **`cudnnRNNBackwardData`** LSTM cell with fp16 (half) inputs might generate wrong values (silently), this issue exists in cudnn 7.1.3 binaries compiled with cuda toolkit 9.0 and toolkit cuda 9.2, and does not exist in cudnn 7.1.3 binaries compiled with toolkit 9.1.
- **`cudnnGetRNNLinLayerMatrixParams`** wrongly returns CUDNN_STATUS_BAD_PARAM when **`cudnnSetRNNDescriptor`** is called with dataType == CUDNN_DATA_FLOAT. This is an issue in 7.1.3 only and will be fixed

in 7.1.4. The dataType argument as of today supports only **CUDNN_DATA_FLOAT** and we plan to support additional compute types in the future.

- There is a small memory leak issue when calling **cudnnRNNBackwardData** with **CUDNN_RNN_ALGO_STANDARD**. This issue also affects previous cuDNN v7 releases. This is fixed in 7.1.4.
- RNN with half precision returns **CUDNN_EXECUTION_FAILED** on Kepler gpu in 7.1.3. This is fixed in 7.1.4 to use pseudo-fp16 computation
- The RNN Find algorithm suite mistakenly did not test **CUDNN_RNN_ALGO_PERSIST_STATIC** and **CUDNN_RNN_ALGO_PERSIST_DYNAMIC** kernels with tensor operations enabled when it was possible to do so. This is fixed in v7.1.4.

Chapter 15. CUDNN RELEASE NOTES V7.1.3

Known Issues

Following are known issues in this release:

- **`cudnnGet`** picks a slow algorithm that does not use Tensor Cores on Volta when inputs are FP16 and it is possible to do so.
- The **`cudnnConvolutionBackwardFilter()`** function may output incorrect results for **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO_FFT_TILING`** when the convolution mode is CUDNN_CONVOLUTION and the product "n*k" (n - batch size, k - number of output feature maps) is large, i.e., several thousand or more. It appears that the CUDNN_CROSS_CORRELATION mode is not affected by this bug.
- There is a small memory leak issue when calling **`cudnnRNNBackwardData`** with **`CUDNN_RNN_ALGO_STANDARD`**. This issue also affects previous cuDNN v7 releases.
- RNN with half precision will not work on Kepler GPUs and will return **`CUDNN_EXECUTION_FAILED`**. This will be fixed in future releases to return **`CUDNN_STATUS_UNSUPPORTED`**.

Fixed Issues

The following issues have been fixed in this release:

- **`cudnnRNNbackwardData`** for LSTM with recurrent projection in half precision may fail in rare cases with misaligned memory access on Pascal and Maxwell.
- **`cudnnRNNbackwardData`** for bidirectional LSTM with recurrent projection may produce inaccurate results, or **`CUDNN_STATUS_UNSUPPORTED`**.
- Algo 1 for forward convolution and dgrad may produce erroneous results when the filter size is greater than the input size. This issue is fixed in 7.1.3.
- For very large RNN networks, the function **`cudnnGetRNNWorkspaceSize`** and **`cudnnGetRNNTrainingReserveSize`** may internally overflow and give incorrect results.

- The small performance regression on multi-layer RNNs using the STANDARD algorithm and Tensor Core math in 7.1.2, as compared to 7.0.5, is fixed in this release.
- Fixed an issue with Persistent LSTM backward pass with a hidden state size in the range 257 to 512 on GPUs with number of SMs between 22 and 31 might hang. This issue also exists in 7.1.1. This is fixed in 7.1.3.
- Fixed an issue Persistent GRU backward pass with a hidden state size in the range 513->720 on GPUs with exactly 30 SMs would hang. This issue also exists in 7.1.1. This is fixed in 7.1.3.

Chapter 16. CUDNN RELEASE NOTES V7.1.2

Key Features and Enhancements

The following enhancements have been added to this release:

- RNN search API extended to support all RNN algorithms.
- Newly added projection Layer supported for inference bidirectional RNN cells and for backward data and gradient.
- Support IDENTITY Activation for all **`cudnnConvolutionBiasActivationForward`** data types for **`CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM`**.
- Added documentation to clarify RNN/LSTM weight formats.

Known Issues

Following are known issues in this release:

- cudnnGet picks a slow algorithm that does not 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 v7.0.5.
- LSTM projection dgrad half precision may fail in rare cases with misaligned memory access on Pascal and Maxwell.
- Dgrad for bidirectional LSTM with projection should not be used, may produce inaccurate results, or **`CUDNN_STATUS_UNSUPPORTED`**.
- The **`cudnnConvolutionBackwardFilter()`** function may output incorrect results for **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO_FFT_TILING`** when the convolution mode is **`CUDNN_CONVOLUTION`** and the product "n*k" (n - batch size, k - number of output feature maps) is large, i.e., several thousand or more. It appears that the **`CUDNN_CROSS_CORRELATION`** mode is not affected by this.

- Persistent LSTM backward pass with a hidden state size in the range 257 to 512 on GPUs with number of SMs between 22 and 31 might hang. This issue also exists in 7.1.1 and will be fixed in 7.1.3.
- Persistent GRU backward pass with a hidden state size in the range 513 to 720 on GPUs with exactly 30 SMs would hang. This issue also exists in 7.1.1 and will be fixed in 7.1.3.
- Algo 1 for forward convolution and dgrad may produce erroneous results when the filter size is greater than the input size.

Fixed Issues

The following issues have been fixed in this release:

- The uint8 input for convolution is restricted to Volta and later. We added support for older architectures, for algo: **`CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM`**.
- In some cases when algorithm **`CUDNN_CONVOLUTION_BWD_FILTER_ALGO1`** was selected, the routine **`cudnnConvolutionBackwardFilter`** could fail at runtime and return **`CUDNN_STATUS_EXECUTION_FAILED`**. It now returns **`CUDNN_STATUS_NOT_SUPPORTED`**.
- **`cudnnSetRNNDescriptor`** no longer needs valid Dropout Descriptor in inference mode, user can pass NULL for Dropout Descriptor in inference mode.

Chapter 17. 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 18. 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 Tensor Cores. 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 19. 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 20. 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 21. 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 22. 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

Trademarks

Copyright

www.nvidia.com