CUDNN LIBRARY

DU-06702-001_v6.5 | August 2014

User Guide

Chapter 1. INTRODUCTION

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

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 multithreading and (optional) interoperability with CUDA streams.

Chapter 2. GENERAL DESCRIPTION

2.1. Programming Model

The cuDNN Library exposes a Host API but assumes that for operations using the GPU the data is directly accessible from the device.

The application must initialize the handle to the cuDNN library context by calling the **`cudnnCreate()`** function. Then, the handle is explicitly passed to every subsequent library function call that operate on GPU data. Once the application finishes using the library, it must call the function **`cudnnDestroy()`** to release the resources associated with the cuDNN library context. This approach allows the user to explicitly control the library setup when using multiple host threads and multiple GPUs. For example, the application can use **`cudaSetDevice()`** to associate different devices with different host threads and in each of those host threads it can initialize a unique handle to the cuDNN library context, which will use the particular device associated with that host thread. Then, the cuDNN library function calls made with different handle will automatically dispatch the computation to different devices. The device associated with a particular cuDNN context is assumed to remain unchanged between the corresponding **`cudnnCreate()`** and **`cudnnDestroy()`** calls. In order for the cuDNN library to use a different device within the same host thread, the application must set the new device to be used by calling **`cudaSetDevice()`** and then create another cuDNN context, which will be associated with the new device, by calling **`cudnnCreate()`**.

2.2. Thread Safety

The library is thread safe and its functions can be called from multiple host threads, even with the same handle. When multiple threads share the same handle, extreme care needs to be taken when the handle configuration is changed because that change will affect potentially subsequent cuDNN calls in all threads. It is even more true for the destruction of the handle. So it is not recommended that multiple threads share the same cuDNN handle.

2.3. Reproducibility

By design, most of cuDNN API routines from a given version generate the same bit-wise results at every run when executed on GPUs with the same architecture and the same number of SMs. However, bit-wise reproducibility is not guaranteed across versions, as the implementation of a given routine may change. With the current release, the following routines do not guarantee reproducibility because they use atomic add operations:

- **`cudnnConvolutionBackwardFilter`**
- **`cudnnConvolutionBackwardData`**

2.4. Requirements

cuDNN supports NVIDIA GPUs of compute capability 3.0 and higher and requires an NVIDIA Driver compatible with CUDA Toolkit 6.5.

Chapter 3. CUDNN DATATYPES REFERENCE

This chapter describes all the types and enums of the cuDNN library API.

3.1. cudnnHandle_t

cudnnHandle_t is a pointer to an opaque structure holding the cuDNN library context. The cuDNN library context must be created using **cudnnCreate()** and the returned handle must be passed to all subsequent library function calls. The context should be destroyed at the end using **cudnnDestroy()**. The context is associated with only one GPU device, the current device at the time of the call to **cudnnCreate()**. However multiple contexts can be created on the same GPU device.

3.2. cudnnStatus_t

cudnnStatus_t is an enumerated type used for function status returns. All cuDNN library functions return their status, which can be one of the following values:

Value	Meaning
CUDNN_STATUS_SUCCESS	The operation completed successfully.
CUDNN_STATUS_NOT_INITIALIZED	The cuDNN library was not initialized properly. This error is usually returned when a call to **cudnnCreate()** fails or when **cudnnCreate()** has not been called prior to calling another cuDNN routine. In the former case, it is usually due to an error in the CUDA Runtime API called by **cudnnCreate()** or by an error in the hardware setup.
CUDNN_STATUS_ALLOC_FAILED	Resource allocation failed inside the cuDNN library. This is usually caused by an internal **cudaMalloc()** failure. To correct: prior to the function call, deallocate previously allocated memory as much as possible.

Value	Meaning
`CUDNN_STATUS_BAD_PARAM`	An incorrect value or parameter was passed to the function. To correct: ensure that all the parameters being passed have valid values.
`CUDNN_STATUS_ARCH_MISMATCH`	The function requires a feature absent from the current GPU device. Note that cuDNN only supports devices with compute capabilities greater than or equal to 3.0. To correct: compile and run the application on a device with appropriate compute capability.
`CUDNN_STATUS_MAPPING_ERROR`	An access to GPU memory space failed, which is usually caused by a failure to bind a texture. To correct: prior to the function call, unbind any previously bound textures. Otherwise, this may indicate an internal error/bug in the library.
`CUDNN_STATUS_EXECUTION_FAILED`	The GPU program failed to execute. This is usually caused by a failure to launch some cuDNN kernel on the GPU, which can occur for multiple reasons. To correct: check that the hardware, an appropriate version of the driver, and the cuDNN library are correctly installed. Otherwise, this may indicate a internal error/bug in the library.
`CUDNN_STATUS_INTERNAL_ERROR`	An internal cuDNN operation failed.
`CUDNN_STATUS_NOT_SUPPORTED`	The functionality requested is not presently supported by cuDNN.
`CUDNN_STATUS_LICENSE_ERROR`	The functionality requested requires some license and an error was detected when trying to check the current licensing. This error can happen if the license is not present or is expired or if the environment variable NVIDIA_LICENSE_FILE is not set properly.

3.3. cudnnTensor4dDescriptor_t

`cudnnCreateTensor4dDescriptor_t` is a pointer to an opaque structure holding the description of a generic 4D dataset. **`cudnnCreateTensor4dDescriptor()`** is used to create one instance, and **`cudnnSetTensor4dDescriptor()`** or **`cudnnSetTensor4dDescriptorEx()`** must be used to initialize this instance.

3.4. cudnnFilterDescriptor_t

cudnnFilterDescriptor_t is a pointer to an opaque structure holding the description of a filter dataset. **cudnnCreateFilterDescriptor()** is used to create one instance, and **cudnnSetFilterDescriptor()** must be used to initialize this instance.

3.5. cudnnConvolutionDescriptor_t

cudnnConvolutionDescriptor_t is a pointer to an opaque structure holding the description of a convolution operation. **cudnnCreateFilterDescriptor()** is used to create one instance, and **cudnnSetFilterDescriptor()** must be used to initialize this instance.

3.6. cudnnPoolingDescriptor_t

cudnnPoolingDescriptor_t is a pointer to an opaque structure holding the description of a pooling operation. **cudnnCreatePoolingDescriptor()** is used to create one instance, and **cudnnSetPoolingDescriptor()** must be used to initialize this instance.

3.7. cudnnDataType_t

cudnnDataType_t is an enumerated type indicating the data type to which a tensor descriptor or filter descriptor refers.

Value	Meaning
CUDNN_DATA_FLOAT	The data is 32-bit single-precision floating point (**float**).
CUDNN_DATA_DOUBLE	The data is 64-bit double-precision floating point (**double**).

3.8. cudnnTensorFormat_t

cudnnTensorFormat_t is an enumerated type used by **cudnnSetTensor4dDescriptor()** to create a tensor with a pre-defined layout.

Value	Meaning
CUDNN_TENSOR_NCHW	This tensor format specifies that the data is laid out in the following order: image, features map, rows, columns. The strides are implicitly defined in such a way that the data are contiguous in memory with no padding between images, feature maps, rows, and columns; the columns are the

Value	Meaning
	inner dimension and the images are the outermost dimension.
CUDNN_TENSOR_NHWC	This tensor format specifies that the data is laid out in the following order: image, rows, columns, features maps. The strides are implicitly defined in such a way that the data are contiguous in memory with no padding between images, rows, columns, and features maps; the feature maps are the inner dimension and the images are the outermost dimension.

3.9. cudnnAddMode_t

cudnnAddMode_t is an enumerated type used by **cudnnAddTensor4d()** to specify how a bias tensor is added to an input/output tensor.

Value	Meaning
CUDNN_ADD_IMAGE or **CUDNN_ADD_SAME_HW**	In this mode, the bias tensor is defined as one image with one feature map. This image will be added to every feature map of every image of the input/output tensor.
CUDNN_ADD_FEATURE_MAP or **CUDNN_ADD_SAME_CHW**	In this mode, the bias tensor is defined as one image with multiple feature maps. This image will be added to every image of the input/output tensor.
CUDNN_ADD_SAME_C	In this mode, the bias tensor is defined as one image with multiple feature maps of dimension 1x1; it can be seen as an vector of feature maps. Each feature map of the bias tensor will be added to the corresponding feature map of all height-by-width pixels of every image of the input/output tensor.
CUDNN_ADD_FULL_TENSOR	In this mode, the bias tensor has the same dimensions as the input/output tensor. It will be added point-wise to the input/output tensor.

3.10. cudnnConvolutionMode_t

cudnnConvolutionMode_t is an enumerated type used by **cudnnSetConvolutionDescriptor()** to configure a convolution descriptor. The filter used for the convolution can be applied in two different ways, corresponding mathematically to a convolution or to a cross-correlation. (A cross-correlation is equivalent to a convolution with its filter rotated by 180 degrees.)

Value	Meaning
CUDNN_CONVOLUTION	In this mode, a convolution operation will be done when applying the filter to the images.
CUDNN_CROSS_CORRELATION	In this mode, a cross-correlation operation will be done when applying the filter to the images.

3.11. cudnnConvolutionPath_t

cudnnConvolutionPath_t is an enumerated type used by the helper routine **cudnnGetOutputTensor4dDim()** to select the results to output.

Value	Meaning
CUDNN_CONVOLUTION_FWD	**cudnnGetOutputTensor4dDim()** will return dimensions related to the output tensor of the forward convolution.
CUDNN_CONVOLUTION_WEIGHT_GRAD	**cudnnGetOutputTensor4dDim()** will return the dimensions of the output filter produced while computing the gradients, which is part of the backward convolution.
CUDNN_CONVOLUTION_DATA_GRAD	**cudnnGetOutputTensor4dDim()** will return the dimensions of the output tensor produced while computing the gradients, which is part of the backward convolution.

3.12. cudnnAccumulateResult_t

cudnnAccumulateResult_t is an enumerated type used by **cudnnConvolutionForward()**, **cudnnConvolutionBackwardFilter()** and **cudnnConvolutionBackwardData()** to specify whether those routines accumulate their results with the output tensor or simply write them to it, overwriting the previous value.

Value	Meaning
CUDNN_RESULT_ACCUMULATE	The results are accumulated with (added to the previous value of) the output tensor.
CUDNN_RESULT_NO_ACCUMULATE	The results overwrite the output tensor.

3.13. cudnnSoftmaxAlgorithm_t

cudnnSoftmaxAlgorithm_t is used to select an implementation of the softmax function used in **cudnnSoftmaxForward()** and **cudnnSoftmaxBackward()**.

Value	Meaning
`CUDNN_SOFTMAX_FAST`	This implementation applies the straightforward softmax operation.
`CUDNN_SOFTMAX_ACCURATE`	This implementation applies a scaling to the input to avoid any potential overflow.

3.14. cudnnSoftmaxMode_t

`cudnnSoftmaxMode_t` is used to select over which data the **`cudnnSoftmaxForward()`** and **`cudnnSoftmaxBackward()`** are computing their results.

Value	Meaning
`CUDNN_SOFTMAX_MODE_INSTANCE`	The softmax operation is computed per image (N) across the dimensions C,H,W.
`CUDNN_SOFTMAX_MODE_CHANNEL`	The softmax operation is computed per spatial location (H,W) per image (N) across the dimension C.

3.15. cudnnPoolingMode_t

`cudnnPoolingMode_t` is an enumerated type passed to **`cudnnSetPoolingDescriptor()`** to select the pooling method to be used by **`cudnnPoolingForward()`** and **`cudnnPoolingBackward()`**.

Value	Meaning
`CUDNN_POOLING_MAX`	The maximum value inside the pooling window will be used.
`CUDNN_POOLING_AVERAGE`	The values inside the pooling window will be averaged.

3.16. cudnnActivationMode_t

`cudnnActivationMode_t` is an enumerated type used to select the neuron activation function used in **`cudnnActivationForward()`** and **`cudnnActivationBackward()`**.

Value	Meaning
`CUDNN_ACTIVATION_SIGMOID`	Selects the sigmoid function.
`CUDNN_ACTIVATION_RELU`	Selects the rectified linear function.
`CUDNN_ACTIVATION_TANH`	Selects the hyperbolic tangent function.

3.17. cudnnDataType_t

cudnnDataType_t is an enumerated type indicating the data type to which a tensor descriptor or filter descriptor refers.

Value	Meaning
CUDNN_DATA_FLOAT	The data is 32-bit single-precision floating point (float).
CUDNN_DATA_DOUBLE	The data is 64-bit double-precision floating point (double).

Chapter 4. CUDNN API REFERENCE

This chapter describes the API of all the routines of the cuDNN library.

4.1. cudnnCreate

```
cudnnStatus_t cudnnCreate(cudnnHandle_t *handle)
```

This function initializes the cuDNN library and creates a handle to an opaque structure holding the cuDNN library context. It allocates hardware resources on the host and device and must be called prior to making any other cuDNN library calls. The cuDNN library context is tied to the current CUDA device. To use the library on multiple devices, one cuDNN handle needs to be created for each device. For a given device, multiple cuDNN handles with different configurations (e.g., different current CUDA streams) may be created. Because **cudnnCreate** allocates some internal resources, the release of those resources by calling **cudnnDestroy** will implicitly call **cudaDeviceSynchronize**; therefore, the recommended best practice is to call **cudnnCreate/cudnnDestroy** outside of performance-critical code paths. For multithreaded applications that use the same device from different threads, the recommended programming model is to create one (or a few, as is convenient) cuDNN handle(s) per thread and use that cuDNN handle for the entire life of the thread.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The initialization succeeded.
CUDNN_STATUS_NOT_INITIALIZED	CUDA Runtime API initialization failed.
CUDNN_STATUS_ALLOC_FAILED	The resources could not be allocated.

4.2. cudnnDestroy

```
cudnnStatus_t cudnnDestroy(cudnnHandle_t handle)
```

This function releases hardware resources used by the cuDNN library. This function is usually the last call with a particular handle to the cuDNN library. Because **cudnnCreate** allocates some internal resources, the release of those resources by

calling **cudnnDestroy** will implicitly call **cudaDeviceSynchronize**; therefore, the recommended best practice is to call **cudnnCreate/cudnnDestroy** outside of performance-critical code paths.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The cuDNN context destruction was successful.
CUDNN_STATUS_NOT_INITIALIZED	The library was not initialized.

4.3. cudnnSetStream

```
cudnnStatus_t cudnnSetStream(cudnnHandle_t handle, cudaStream_t streamId)
```

This function sets the cuDNN library stream, which will be used to execute all subsequent calls to the cuDNN library functions with that particular handle. If the cuDNN library stream is not set, all kernels use the default (**NULL**) stream. In particular, this routine can be used to change the stream between kernel launches and then to reset the cuDNN library stream back to **NULL**.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The stream was set successfully.

4.4. cudnnGetStream

```
cudnnStatus_t cudnnGetStream(cudnnHandle_t handle, cudaStream_t *streamId)
```

This function gets the cuDNN library stream, which is being used to execute all calls to the cuDNN library functions. If the cuDNN library stream is not set, all kernels use the *default* **NULL** stream.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The stream was returned successfully.

4.5. cudnnCreateTensor4dDescriptor

```
cudnnStatus_t cudnnCreateTensor4dDescriptor(cudnnTensor4dDescriptor_t
 *tensorDesc)
```

This function creates a Tensor4D descriptor object by allocating the memory needed to hold its opaque structure.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was created successfully.
CUDNN_STATUS_ALLOC_FAILED	The resources could not be allocated.

4.6. cudnnSetTensor4dDescriptor

```
cudnnStatus_t
cudnnSetTensor4dDescriptor( cudnnTensor4dDescriptor_t tensorDesc,
                            cudnnTensorFormat_t format,
                            cudnnDataType_t dataType,
                            int n,
                            int c,
                            int h,
                            int w )
```

This function initializes a previously created Tensor4D descriptor object. The strides of the four dimensions are inferred from the format parameter and set in such a way that the data is contiguous in memory with no padding between dimensions.

Param	In/out	Meaning
tensorDesc	input/ output	Handle to a previously created tensor descriptor.
format	input	Type of format.
datatype	input	Data type.
n	input	Number of images.
c	input	Number of feature maps per image.
h	input	Height of each feature map.
w	input	Width of each feature map.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the parameters **n,c,h,w** was negative or **format** has an invalid enumerant value or **dataType** has an invalid enumerant value.

4.7. cudnnSetTensor4dDescriptorEx

```
cudnnStatus_t
cudnnSetTensor4dDescriptorEx( cudnnTensor4dDescriptor_t tensorDesc,
                              cudnnDataType_t dataType,
                              int n,
                              int c,
                              int h,
                              int w,
                              int nStride,
                              int cStride,
                              int hStride,
                              int wStride )
```

This function initializes a previously created Tensor4D descriptor object, similarly to **cudnnSetTensor4dDescriptor** but with the strides explicitly passed as parameters. This can be used to lay out the 4D tensor in any order or simply to define gaps between dimensions.

At present, some cuDNN routines have limited support for strides; for example, **wStride==1** is sometimes required. Those routines will return CUDNN_STATUS_NOT_SUPPORTED if a Tensor4D object with an unsupported stride is used. **cudnnTransformTensor4d** can be used to convert the data to a supported layout.

Param	In/out	Meaning
tensorDesc	input/ output	Handle to a previously created tensor descriptor.
datatype	input	Data type.
n	input	Number of images.
c	input	Number of feature maps per image.
h	input	Height of each feature map.
w	input	Width of each feature map.
nStride	input	Stride between two consecutive images.
cStride	input	Stride between two consecutive feature maps.
hStride	input	Stride between two consecutive rows.
wStride	input	Stride between two consecutive columns.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the parameters **n,c,h,w** or **nStride,cStride,hStride,wStride** is negative or **dataType** has an invalid enumerant value.

4.8. cudnnGetTensor4dDescriptor

```
cudnnStatus_t
cudnnGetTensor4dDescriptor( cudnnTensor4dDescriptor_t tensorDesc,
                            cudnnDataType_t *dataType,
                            int *n,
                            int *c,
                            int *h,
                            int *w,
                            int *nStride,
                            int *cStride,
                            int *hStride,
                            int *wStride )
```

This function queries the parameters of the previouly initialized Tensor4D descriptor object.

Param	In/out	Meaning
tensorDesc	input	Handle to a previously insitialized tensor descriptor.
datatype	output	Data type.
n	output	Number of images.
c	output	Number of feature maps per image.
h	output	Height of each feature map.
w	output	Width of each feature map.
nStride	output	Stride between two consecutive images.
cStride	output	Stride between two consecutive feature maps.
hStride	output	Stride between two consecutive rows.
wStride	output	Stride between two consecutive columns.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
`CUDNN_STATUS_SUCCESS`	The operation succeeded.

4.9. cudnnDestroyTensor4dDescriptor

```
cudnnStatus_t cudnnDestroyTensor4dDescriptor(cudnnTensor4dDescriptor_t
 tensorDesc)
```

This function destroys a previously created Tensor4D descriptor object.

Return Value	Meaning
`CUDNN_STATUS_SUCCESS`	The object was destroyed successfully.

4.10. cudnnTransformTensor4d

```
cudnnStatus_t
cudnnTransformTensor4d( cudnnHandle_t handle,
                        cudnnTensor4dDescriptor_t  srcDesc,
                        const void                *srcData,
                        cudnnTensor4dDescriptor_t  destDesc,
                        void                      *destData )
```

This function copies the data from one tensor to another tensor with a different layout. Those descriptors need to have the same dimensions but not necessarily the same strides. The input and output tensors must not overlap in any way (i.e., tensors cannot be transformed in place). This function can be used to convert a tensor with an unsupported format to a supported one.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
srcDesc	input	Handle to a previously initialized tensor descriptor.
srcData	input	Pointer to data of the tensor described by the **srcDesc** descriptor.
destDesc	input	Handle to a previously initialized tensor descriptor.
destData	output	Pointer to data of the tensor described by the **destDesc** descriptor.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function launched successfully.
CUDNN_STATUS_BAD_PARAM	The dimensions **n,c,h,w** or the **dataType** of the two tensor descriptors are different.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.11. cudnnAddTensor4d

```
cudnnStatus_t
cudnnAddTensor4d( cudnnHandle_t               handle,
                  cudnnAddMode_t              mode,
                  const void                 *alpha,
                  cudnnTensor4dDescriptor_t   biasDesc,
                  const void                 *biasData,
                  cudnnTensor4dDescriptor_t   srcDestDesc,
                  void                       *srcDestData )
```

This function adds the scaled values of one tensor to another tensor. The **mode** parameter can be used to select different ways of performing the scaled addition. The amount of data described by the **biasDesc** descriptor must match exactly the amount of data needed to perform the addition. Therefore, the following conditions must be met:

- Except for the **CUDNN_ADD_SAME_C** mode, the dimensions **h,w** of the two tensors must match.
- In the case of **CUDNN_ADD_IMAGE** mode, the dimensions **n,c** of the bias tensor must be 1.
- In the case of **CUDNN_ADD_FEATURE_MAP** mode, the dimension **n** of the bias tensor must be 1 and the dimension **c** of the two tensors must match.
- In the case of **CUDNN_ADD_FULL_TENSOR** mode, the dimensions **n,c** of the two tensors must match.
- In the case of **CUDNN_ADD_SAME_C** mode, the dimensions **n,w,h** of the bias tensor must be 1 and the dimension **c** of the two tensors must match.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
biasDesc	input	Handle to a previously initialized tensor descriptor.

Param	In/out	Meaning
mode	input	Addition mode that describe how the addition is performed.
alpha	input	Scalar factor to be applied to every data element of the bias tensor before it is added to the output tensor.
srcData	input	Pointer to data of the tensor described by the **biasDesc** descriptor.
srcDestDesc	input/ output	Handle to a previously initialized tensor descriptor.
srcDestData	input/ output	Pointer to data of the tensor described by the **srcDestDesc** descriptor.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function executed successfully.
CUDNN_STATUS_BAD_PARAM	The dimensions **n,c,h,w** of the bias tensor refer to an amount of data that is incompatible with the **mode** parameter and the output tensor dimensions or the **dataType** of the two tensor descriptors are different.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.12. cudnnCreateFilterDescriptor

```
cudnnStatus_t cudnnCreateFilterDescriptor(cudnnFilterDescriptor_t *filterDesc)
```

This function creates a filter descriptor object by allocating the memory needed to hold its opaque structure,

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was created successfully.
CUDNN_STATUS_ALLOC_FAILED	The resources could not be allocated.

4.13. cudnnSetFilterDescriptor

```
cudnnStatus_t
cudnnSetFilterDescriptor( cudnnFilterDescriptor_t filterDesc,
                          cudnnDataType_t dataType,
                          int k,
                          int c,
                          int h,
                          int w )
```

This function initializes a previously created filter descriptor object. Filters layout must be contiguous in memory.

Param	In/out	Meaning
filterDesc	input/ output	Handle to a previously created filter descriptor.
datatype	input	Data type.
k	input	Number of output feature maps.
c	input	Number of input feature maps.
h	input	Height of each filter.
w	input	Width of each filter.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the parameters **k,c,h,w** is negative or **dataType** has an invalid enumerant value.

4.14. cudnnGetFilterDescriptor

```
cudnnStatus_t
cudnnGetFilterDescriptor( cudnnFilterDescriptor_t filterDesc,
                          cudnnDataType_t *dataType,
                          int *k,
                          int *c,
                          int *h,
                          int *w )
```

This function queries the parameters of the previouly initialized filter descriptor object.

Param	In/out	Meaning
filterDesc	input	Handle to a previously created filter descriptor.
datatype	output	Data type.
k	output	Number of output feature maps.
c	output	Number of input feature maps.
h	output	Height of each filter.
w	output	Width of each filter.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.

4.15. cudnnDestroyFilterDescriptor

```
cudnnStatus_t cudnnDestroyFilterDescriptor(cudnnFilterdDescriptor_t filterDesc)
```

This function destroys a previously created Tensor4D descriptor object.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was destroyed successfully.

4.16. cudnnCreateConvolutionDescriptor

```
cudnnStatus_t cudnnCreateConvolutionDescriptor(cudnnConvolutionDescriptor_t
 *convDesc)
```

This function creates a convolution descriptor object by allocating the memory needed to hold its opaque structure,

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was created successfully.
CUDNN_STATUS_ALLOC_FAILED	The resources could not be allocated.

4.17. cudnnSetConvolutionDescriptor

```
cudnnStatus_t
cudnnSetConvolutionDescriptor( cudnnConvolutionDescriptor_t convDesc,
                               cudnnTensor4dDescriptor_t inputTensorDesc,
                               cudnnFilterDescriptor_t filterDesc,
                               int pad_h,
                               int pad_w,
                               int u,
                               int v,
                               int upscalex,
                               int upscaley,
                               cudnnConvolutionMode_t mode )
```

This function initializes a previously created convolution descriptor object, according to an input tensor descriptor and a filter descriptor passed as parameter. This function assumes that the tensor and filter descriptors corresponds to the formard convolution path and checks if their settings are valid. That same convolution descriptor can be reused in the backward path provided it corresponds to the same layer.

Param	In/out	Meaning
convDesc	input/ output	Handle to a previously created convolution descriptor.
inputTensorDesc	input	Input tensor descriptor used for that layer on the forward path.
filterDesc	input	Filter descriptor used for that layer on the forward path.

Param	In/out	Meaning
pad_h	input	zero-padding height: number of rows of zeros implicitly concatenated onto the top and onto the bottom of input images.
pad_w	input	zero-padding width: number of columns of zeros implicitly concatenated onto the left and onto the right of input images.
u	input	Vertical filter stride.
v	input	Horizontal filter stride.
upscalex	input	Upscale the input in x-direction.
upscaley	input	Upscale the input in y-direction.
mode	input	Selects between **CUDNN_CONVOLUTION** and **CUDNN_CROSS_CORRELATION**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ One of the parameters **u,v** is negative. ▸ The **dataType** of the tensor and filter descriptors differ or have invalid enumerant values. ▸ The number of feature maps of the tensor descriptor and the number of input feature maps of the filter differ. ▸ The parameter **mode** has an invalid enumerant value.
CUDNN_STATUS_NOT_SUPPORTED	The parameter **upscalex** or **upscaley** is not 1.

4.18. cudnnSetConvolutionDescriptorEx

```
cudnnStatus_t
cudnnSetConvolutionDescriptorEx( cudnnConvolutionDescriptor_t convDesc,
                                 int n,
                                 int c,
                                 int h,
                                 int w,
                                 int k,
                                 int r,
                                 int s,
                                 int pad_h,
                                 int pad_w,
                                 int u,
                                 int v,
                                 int upscalex,
                                 int upscaley,
                                 cudnnConvolutionMode_t mode )
```

This function initializes a previously created convolution descriptor object. It is similar to **cudnnSetConvolutionDescriptor** but every parameter of the convolution must be passed explicitly.

Param	In/out	Meaning
convDesc	input/ output	Handle to a previously created convolution descriptor.
n	input	Number of images.
c	input	Number of input feature maps.
h	input	Height of each input feature map.
w	input	Width of each input feature map.
k	input	Number of output feature maps.
r	input	Height of each filter.
s	input	Width of each filter.
pad_h	input	zero-padding height: number of rows of zeros implicitly concatenated onto the top and onto the bottom of input images.
pad_w	input	zero-padding width: number of columns of zeros implicitly concatenated onto the left and onto the right of input images.
u	input	Vertical filter stride.
v	input	Horizontal filter stride.
upscalex	input	Upscale the input in x-direction.
upscaley	input	Upscale the input in y-direction.
mode	input	Selects between **CUDNN_CONVOLUTION** and **CUDNN_CROSS_CORRELATION**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ One of the parameters **u,v** is negative. ▸ The parameter **mode** has an invalid enumerant value.
CUDNN_STATUS_NOT_SUPPORTED	The parameter **upscalex** or **upscaley** is not 1.

4.19. cudnnGetOutputTensor4dDim

```
cudnnStatus_t
cudnnGetOutputTensor4dDim( const cudnnConvolutionDescriptor_t convDesc,
                           cudnnConvolutionPath_t path,
                           int *n,
                           int *c,
                           int *h,
                           int *w )
```

This function returns the dimensions of a convolution's output, given the convolution descriptor and the direction of the convolution. This function can help to setup the output tensor and allocate the proper amount of memory prior to launch the actual convolution.

Param	In/out	Meaning
convDesc	input	Handle to a previously created convolution descriptor.
path	input	Enumerant to specify the direction of the convolution.
n	output	Number of output images.
c	output	Number of output feature maps per image.
h	output	Height of each output feature map.
w	output	Width of each output feature map.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_BAD_PARAM	The **path** parameter has an invalid enumerant value.
CUDNN_STATUS_SUCCESS	The object was set successfully.

4.20. cudnnDestroyFilterDescriptor

```
cudnnStatus_t cudnnDestroyConvolutionDescriptor(cudnnConvolutionDescriptor_t
 convDesc)
```

This function destroys a previously created convolution descriptor object.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was destroyed successfully.

4.21. cudnnConvolutionForward

```
cudnnStatus_t
cudnnConvolutionForward( cudnnHandle_t handle,
                         cudnnTensor4dDescriptor_t     srcDesc,
                         const void                   *srcData,
                         cudnnFilterDescriptor_t       filterDesc,
                         const void                   *filterData,
                         cudnnConvolutionDescriptor_t  convDesc,
                         cudnnTensor4dDescriptor_t     destDesc,
                         void                         *destData,
                         cudnnAccumulateResult_t       accumulate )
```

This function executes convolutions or cross-correlations over **`src`** using the specified **`filters`**, returning results in **`dest`**.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
srcDesc	input	Handle to a previously initialized tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **`srcDesc`**.
filterDesc	input	Handle to a previously initialized filter descriptor.
filterData	input	Data pointer to GPU memory associated with the filter descriptor **`filterDesc`**.
convDesc	input	Previously initialized convolution descriptor.
destDesc	input	Handle to a previously initialized tensor descriptor.
destData	input/ output	Data pointer to GPU memory associated with the tensor descriptor **`destDesc`** that carries the result of the convolution.
accumulate	input	Enumerant that specifies whether the convolution accumulates with or overwrites the output tensor.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
`CUDNN_STATUS_SUCCESS`	The operation was launched successfully.
`CUDNN_STATUS_MAPPING_ERROR`	An error occured during the texture binding of the filter data.
`CUDNN_STATUS_EXECUTION_FAILED`	The function failed to launch on the GPU.

4.22. cudnnConvolutionBackwardBias

```
cudnnStatus_t
cudnnConvolutionBackwardBias( cudnnHandle_t handle,
                              cudnnTensor4dDescriptor_t  srcDesc,
                              const void                *srcData,
                              cudnnTensor4dDescriptor_t  destDesc,
                              void                      *destData,
                              cudnnAccumulateResult_t    accumulate )
```

This function computes the convolution gradient with respect to the bias, which is the sum of every element belonging to the same feature map across all of the images of the input tensor. Therefore, the number of elements produced is equal to the number of features maps of the input tensor.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	output	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.
accumulate	input	Enumerant that specifies whether the convolution accumulates with or overwrites the output tensor.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The operation was launched successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ One of the parameters **n,h,w** of the output tensor is not 1. ▸ The numbers of feature maps of the input tensor and output tensor differ. ▸ The **dataType** of the two tensor descriptors are different.
CUDNN_STATUS_NOT_SUPPORTED	At least one of the following conditions are met: ▸ The width stride of the input tensor is not 1. ▸ The height stride and the width of the input tensor differ. ▸ The feature map stride of the output tensor is not 1.

4.23. cudnnConvolutionBackwardFilter

```
cudnnStatus_t
cudnnConvolutionBackwardFilter( cudnnHandle_t handle,
                                cudnnTensor4dDescriptor_t     srcDesc,
                                const void                   *srcData,
                                cudnnTensor4dDescriptor_t     diffDesc,
                                const void                   *diffData,
                                cudnnConvolutionDescriptor_t  convDesc,
                                cudnnFilterDescriptor_t       gradDesc,
                                void                         *gradData,
                                cudnnAccumulateResult_t       accumulate )
```

This function computes the convolution gradient with respect to the filter coefficients.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
srcDesc	input	Handle to a previously initialized tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
diffDesc	input	Handle to the previously initialized input differential tensor descriptor.
diffData	input	Data pointer to GPU memory associated with the input differential tensor descriptor **diffDesc**.
convDesc	input	Previously initialized convolution descriptor.
gradDesc	input	Handle to a previously initialized filter descriptor.
gradData	input/ output	Data pointer to GPU memory associated with the filter descriptor **gradDesc** that carries the result.
accumulate	input	Enumerant that specifies whether the convolution accumulates with or overwrites the output tensor.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The operation was launched successfully.
CUDNN_STATUS_NOT_SUPPORTED	The requested operation is not currently supported in cuDNN. Your diffDesc is likely not in NCHW format.
CUDNN_STATUS_MAPPING_ERROR	An error occurs during the texture binding of the filter data.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.24. cudnnConvolutionBackwardData

```
cudnnStatus_t
cudnnConvolutionBackwardData( cudnnHandle_t handle,
                              cudnnFilterDescriptor_t       filterDesc,
                              const void                    *filterData,
                              cudnnTensor4dDescriptor_t     diffDesc,
                              const void                    *diffData,
                              cudnnConvolutionDescriptor_t  convDesc,
                              cudnnTensor4dDescriptor_t     gradDesc,
                              void                          *gradData,
                              cudnnAccumulateResult_t       accumulate
                            );
```

This function computes the convolution gradient with respect to the ouput tensor.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
filterDesc	input	Handle to a previously initialized filter descriptor.
filterData	input	Data pointer to GPU memory associated with the filter descriptor **filterDesc**.
diffDesc	input	Handle to the previously initialized input differential tensor descriptor.
diffData	input	Data pointer to GPU memory associated with the input differential tensor descriptor **diffDesc**.
convDesc	input	Previously initialized convolution descriptor.
gradDesc	input	Handle to the previously initialized output tensor descriptor.
gradData	input/ output	Data pointer to GPU memory associated with the output tensor descriptor **gradDesc** that carries the result.
accumulate	input	Enumerant that specifies whether the convolution accumulates with or overwrites the output tensor.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The operation was launched successfully.
CUDNN_STATUS_NOT_SUPPORTED	The requested operation is not currently supported in cuDNN. Your diffDesc is likely not in NCHW format.
CUDNN_STATUS_MAPPING_ERROR	An error occurs during the texture binding of the filter data or the input differential tensor data
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.25. cudnnSoftmaxForward

```
cudnnStatus_t
cudnnSoftmaxForward( cudnnHandle_t handle,
                     cudnnSoftmaxAlgorithm_t    algorithm,
                     cudnnSoftmaxMode_t         mode,
                     cudnnTensor4dDescriptor_t  srcDesc,
                     const void                *srcData,
                     cudnnTensor4dDescriptor_t  destDesc,
                     void                      *destData )
```

This routine computes the softmax function.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
algorithm	input	Enumerant to specify the softmax algorithm.
mode	input	Enumerant to specify the softmax mode.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	output	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function launched successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ The dimensions **n,c,h,w** of the input tensor and output tensors differ. ▸ The **datatype** of the input tensor and output tensors differ. ▸ The parameters **algorithm** or **mode** have an invalid enumerant value.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.26. cudnnSoftmaxBackward

```
cudnnStatus_t
cudnnSoftmaxBackward( cudnnHandle_t handle,
                      cudnnSoftmaxAlgorithm_t    algorithm,
                      cudnnSoftmaxMode_t         mode,
                      cudnnTensor4dDescriptor_t  srcDesc,
                      const void                *srcData,
                      cudnnTensor4dDescriptor_t  srcDiffDesc,
                      const void                *srcDiffData,
                      cudnnTensor4dDescriptor_t  destDiffDesc,
                      void                      *destDiffData )
```

This routine computes the gradient of the softmax function.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
algorithm	input	Enumerant to specify the softmax algorithm.
mode	input	Enumerant to specify the softmax mode.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
srcDiffDesc	input	Handle to the previously initialized input differential tensor descriptor.
srcDiffData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDiffData**.
destDiffDesc	input	Handle to the previously initialized output differential tensor descriptor.
destDiffData	output	Data pointer to GPU memory associated with the output tensor descriptor **destDiffDesc**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function launched successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ The dimensions **n,c,h,w** of the **srcDesc**, **srcDiffDesc** and **destDiffDesc** tensors differ. ▸ The strides **nStride, cStride, hStride, wStride** of the **srcDesc** and **srcDiffDesc** tensors differ. ▸ The **datatype** of the three tensors differs.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.27. cudnnCreatePoolingDescriptor

```
cudnnStatus_t cudnnCreatePoolingDescriptor( cudnnPoolingDescriptor_t*
 poolingDesc )
```

This function creates a pooling descriptor object by allocating the memory needed to hold its opaque structure,

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was created successfully.
CUDNN_STATUS_ALLOC_FAILED	The resources could not be allocated.

4.28. cudnnSetPoolingDescriptor

```
cudnnStatus_t
cudnnSetPoolingDescriptor( cudnnPoolingDescriptor_t poolingDesc,
                           cudnnPoolingMode_t mode,
                           int windowHeight,
                           int windowWidth,
                           int verticalStride,
                           int horizontalStride )
```

This function initializes a previously created pooling descriptor object.

Param	In/out	Meaning
poolingDesc	input/ output	Handle to a previously created pooling descriptor.
mode	input	Enumerant to specify the pooling mode.
windowHeight	input	Height of the pooling window.
windowWidth	input	Width of the pooling window.
verticalStride	input	Pooling vertical stride.
horizontalStride	input	Pooling horizontal stride.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the parameters **windowHeight**, **windowWidth**, **verticalStride**, **horizontalStride** is negative or **mode** has an invalid enumerant value.

4.29. cudnnGetPoolingDescriptor

```
cudnnStatus_t
cudnnGetPoolingDescriptor( cudnnPoolingDescriptor_t poolingDesc,
                           cudnnPoolingMode_t *mode,
                           int *windowHeight,
                           int *windowWidth,
                           int *verticalStride,
                           int *horizontalStride )
```

This function queries a previously created pooling descriptor object.

Param	In/out	Meaning
poolingDesc	input	Handle to a previously created pooling descriptor.
mode	output	Enumerant to specify the pooling mode.
windowHeight	output	Height of the pooling window.
windowWidth	output	Width of the pooling window.
verticalStride	output	Pooling vertical stride.
horizontalStride	output	Pooling horizontal stride.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was set successfully.

4.30. cudnnDestroyPoolingDescriptor

```
cudnnStatus_t cudnnDestroyPoolingDescriptor( cudnnPoolingDescriptor_t
 poolingDesc )
```

This function destroys a previously created pooling descriptor object.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was destroyed successfully.

4.31. cudnnPoolingForward

```
cudnnStatus_t
cudnnPoolingForward( cudnnHandle_t handle,
                     cudnnPoolingDescriptor_t   poolingDesc,
                     cudnnTensor4dDescriptor_t  srcDesc,
                     const void                *srcData,
                     cudnnTensor4dDescriptor_t  destDesc,
                     void                      *destData )
```

This function computes pooling of input values (i.e., the maximum or average of several adjacent values) to produce an output with smaller height and/or width.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
poolingDesc	input	Handle to a previously initialized pooling descriptor.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	output	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function launched successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ The dimensions **n,c** of the input tensor and output tensors differ. ▸ The **datatype** of the input tensor and output tensors differs.
CUDNN_STATUS_NOT_SUPPORTED	The **wStride** of input tensor or output tensor is not 1.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.32. cudnnPoolingBackward

```
cudnnStatus_t
cudnnPoolingBackward( cudnnHandle_t handle,
                      cudnnPoolingDescriptor_t   poolingDesc,
                      cudnnTensor4dDescriptor_t  srcDesc,
                      const void                *srcData,
                      cudnnTensor4dDescriptor_t  srcDiffDesc,
                      const void                *srcDiffData,
                      cudnnTensor4dDescriptor_t  destDesc,
                      const void                *destData,
                      cudnnTensor4dDescriptor_t  destDiffDesc,
                      void                      *destDiffData )
```

This function computes the gradient of a pooling operation.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
poolingDesc	input	Handle to the previously initialized pooling descriptor.
srcDesc	input	Handle to the previously initialized input tensor descriptor.

Param	In/out	Meaning
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
srcDiffDesc	input	Handle to the previously initialized input differential tensor descriptor.
srcDiffData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDiffData**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	input	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.
destDiffDesc	input	Handle to the previously initialized output differential tensor descriptor.
destDiffData	output	Data pointer to GPU memory associated with the output tensor descriptor **destDiffDesc**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function launched successfully.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ The dimensions **n,c,h,w** of the **srcDesc** and **srcDiffDesc** tensors differ. ▸ The strides **nStride, cStride, hStride, wStride** of the **srcDesc** and **srcDiffDesc** tensors differ. ▸ The dimensions **n,c,h,w** of the **destDesc** and **destDiffDesc** tensors differ. ▸ The strides **nStride, cStride, hStride, wStride** of the **destDesc** and **destDiffDesc** tensors differ. ▸ The **datatype** of the four tensors differ.
CUDNN_STATUS_NOT_SUPPORTED	The **wStride** of input tensor or output tensor is not 1.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.33. cudnnActivationForward

```
cudnnStatus_t
cudnnActivationForward( cudnnHandle_t handle,
                        cudnnActivationMode_t mode,
                        cudnnTensor4dDescriptor_t  srcDesc,
                        const void                *srcData,
                        cudnnTensor4dDescriptor_t  destDesc,
                        void                      *destData )
```

This routine applies a specified neuron activation function element-wise over each input value.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
mode	input	Enumerant to specify the activation mode.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	output	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function launched successfully.
CUDNN_STATUS_BAD_PARAM	The parameter **mode** has an invalid enumerant value.
CUDNN_STATUS_NOT_SUPPORTED	At least one of the following conditions are met: ▸ The dimensions **n,c,h,w** of the input tensor and output tensors differ. ▸ The **datatype** of the input tensor and output tensors differs.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.34. cudnnActivationBackward

```
cudnnStatus_t
cudnnActivationBackward( cudnnHandle_t handle,
                         cudnnActivationMode_t mode,
                         cudnnTensor4dDescriptor_t  srcDesc,
                         const void                *srcData,
                         cudnnTensor4dDescriptor_t  srcDiffDesc,
                         const void                *srcDiffData,
                         cudnnTensor4dDescriptor_t  destDesc,
                         const void                *destData,
                         cudnnTensor4dDescriptor_t  destDiffDesc,
                         void                      *destDiffData )
```

This routine computes the gradient of a neuron activation function.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
mode	input	Enumerant to specify the activation mode.
srcDesc	input	Handle to the previously initialized input tensor descriptor.

Param	In/out	Meaning
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
srcDiffDesc	input	Handle to the previously initialized input differential tensor descriptor.
srcDiffData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDiffData**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	input	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.
destDiffDesc	input	Handle to the previously initialized output differential tensor descriptor.
destDiffData	output	Data pointer to GPU memory associated with the output tensor descriptor **destDiffDesc**.

The possible error values returned by this function and their meanings are listed below.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The function launched successfully.
CUDNN_STATUS_BAD_PARAM	The parameter **mode** has an invalid enumerant value.
CUDNN_STATUS_NOT_SUPPORTED	At least one of the following conditions are met: ▸ The dimensions **n,c,h,w** of the four tensors differ. ▸ The strides **nStride, cStride, hStride, wStride** of the input tensor and the input differential tensor differ. ▸ The strides **nStride, cStride, hStride, wStride** of the output tensor and the output differential tensor differ. ▸ The **datatype** of the four tensors differs.
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

Chapter 5. ACKNOWLEDGMENTS

Some of the cuDNN library routines were derived from code developed by the University of Tennessee and are subject to the Modified Berkeley Software Distribution License as follows: