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. cudnnTensorDescriptor_t

cudnnCreateTensorDescriptor_t is a pointer to an opaque structure holding the description of a generic n-D dataset. **cudnnCreateTensorDescriptor()** is used to create one instance, and one of the routrines **cudnnSetTensorNdDescriptor()**, **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. **cudnnCreateConvolutionDescriptor()** is used to create one instance, and **cudnnSetConvolutionNdDescriptor()** or **cudnnSetConvolution2dDescriptor()** 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 **cudnnSetPoolingNdDescriptor()** or **cudnnSetPooling2dDescriptor()** 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 **cudnnAddTensor()** 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. cudnnConvolutionFwdPreference_t

cudnnConvolutionFwdPreference_t is an enumerated type used by **cudnnGetConvolutionForwardAlgorithm()** to help the choice of the algorithm used for the forward convolution.

Value	Meaning
CUDNN_CONVOLUTION_FWD_NO_WORKSPACE	In this configuration, the routine **cudnnGetConvolutionForwardAlgorithm()** is guaranteed to return an algorithm that does not require any extra workspace to be provided by the user.
CUDNN_CONVOLUTION_FWD_PREFER_FASTEST	In this configuration, the routine **cudnnGetConvolutionForwardAlgorithm()** will return the fastest algorithm regardless how much workspace is needed to execute it.
CUDNN_CONVOLUTION_FWD_SPECIFY_WORKSPACE_LIMIT	In this configuration, the routine **cudnnGetConvolutionForwardAlgorithm()** will return the fastest algorithm that fits within the memory limit that the user provided.

3.12. cudnnConvolutionFwdAlgo_t

cudnnConvolutionFwdAlgo_t is an enumerated type that exposes the different algorithm available to execute the forward convolution operation.

Value	Meaning
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM	This algorithm expresses the convolution as a matrix product without actually explicitly form the matrix that holds the input tensor data.
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM	This algorithm expresses the convolution as a matrix product without actually explicitly form the matrix that holds the input tensor data, but still needs some memory workspace to precompute some indices in order to facilitate the implicit construction of the matrix that holds the input tensor data
CUDNN_CONVOLUTION_FWD_ALGO_GEMM	This algorithm expresses the convolution as an explicit matrix product. A significant memory

Value	Meaning
	workspace is needed to store the matrix that holds the input tensor data.
`CUDNN_CONVOLUTION_FWD_ALGO_DIRECT`	This algorithm expresses the convolution as a direct convolution (e.g without implicitly or explicitly doing a matrix multiplication).

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. cudnnCreateTensorDescriptor

```
cudnnStatus_t cudnnCreateTensorDescriptor(cudnnTensorDescriptor_t *tensorDesc)
```

This function creates a generic Tensor 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( cudnnTensorDescriptor_t   tensorDesc,
                            cudnnTensorFormat_t format,
                            cudnnDataType_t dataType,
                            int n,
                            int c,
                            int h,
                            int w )
```

This function initializes a previously created generic Tensor descriptor object into a 4D tensor. 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( cudnnTensorDescriptor_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 generic Tensor descriptor object into a 4D tensor, 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; Those routines will return CUDNN_STATUS_NOT_SUPPORTED if a Tensor4D object with an unsupported stride is used. **cudnnTransformTensor** 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( cudnnTensorDescriptor_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. cudnnSetTensorNdDescriptor

```
cudnnStatus_t
cudnnSetTensorNdDescriptor( cudnnTensorDescriptor_t   tensorDesc,

                            cudnnDataType_t dataType,
                            int nbDims,
                            int dimA[],
                            int strideA[])
```

This function initializes a previously created generic Tensor descriptor object.

Param	In/out	Meaning
tensorDesc	input/ output	Handle to a previously created tensor descriptor.
datatype	input	Data type.
nbDims	input	Dimension of the tensor.
dimA	input	Array of dimension **nbDims** that contain the size of the tensor for every dimension.
strideA	input	Array of dimension **nbDims** that contain the stride of the tensor for every dimension.

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 elements of the array **dimA** was negative or **dataType** has an invalid enumerant value.
CUDNN_STATUS_NOT_SUPPORTED	the parameter **nbDims** exceeds the maximum supported dimension.

4.10. cudnnGetTensorNdDescriptor

```
cudnnStatus_t
cudnnGetTensorNdDescriptor( const cudnnTensorDescriptor_t   tensorDesc,

                            int nbDimsRequested,
                            cudnnDataType_t *dataType,
                            int *nbDims,
                            int dimA[],
                            int strideA[])
```

This function queires a previously initialized generic Tensor descriptor object.

Param	In/out	Meaning
tensorDesc	input	Handle to a previously initialized tensor descriptor.
nbDimsRequested	input	Dimension of the expected tensor descriptor. It is also the minimum size of the arrays **dimA** and **strideA** in order to be able to hold the results
datatype	output	Data type.
nbDims	output	Actual dimension of the tensor.
dimA	output	Array of dimension of at least **nbDimsRequested** that will be filled with the size parameters from the provided tensor descriptor.
strideA	input	Array of dimension of at least **nbDimsRequested** that will be filled with the stride parameters from the provided tensor descriptor.

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 elements of the array **dimA** was negative or **dataType** has an invalid enumerant value.
CUDNN_STATUS_NOT_SUPPORTED	the parameter **nbDims** exceeds the maximum supported dimension

4.11. cudnnDestroyTensorDescriptor

```
cudnnStatus_t cudnnDestroyTensorDescriptor(cudnnTensorDescriptor_t tensorDesc)
```

This function destroys a previously created Tensor descriptor object.

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was destroyed successfully.

4.12. cudnnTransformTensor

```
cudnnStatus_t
cudnnTransformTensor( cudnnHandle_t                  handle,
                      const void                    *alpha,
                      const cudnnTensorDescriptor_t  srcDesc,
                      const void                    *srcData,
                      const void                    *beta,
                      const cudnnTensorDescriptor_t  destDesc,
                      void                          *destData )
```

This function copies the scaled 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.
alpha	input	Scalar factor to be applied to every element of the input tensor before it is added to the output tensor.
srcDesc	input	Handle to a previously initialized tensor descriptor.
srcData	input	Pointer to data of the tensor described by the **srcDesc** descriptor.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the operation Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

4.13. cudnnAddTensor

```
cudnnStatus_t
cudnnAddTensor(  cudnnHandle_t                      handle,
                 cudnnAddMode_t                     mode,
                 const void                        *alpha,
                 const cudnnTensorDescriptor_t      biasDesc,
                 const void                        *biasData,
                 const void                        *beta,
                 const cudnnTensorDescriptor_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.
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.
biasDesc	input	Handle to a previously initialized tensor descriptor.
biasData	input	Pointer to data of the tensor described by the **biasDesc** descriptor.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the operation Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

4.14. cudnnSetTensor

```
cudnnStatus_t
cudnnSetTensor( cudnnHandle_t                 handle,
                const cudnnTensorDescriptor_t srcDestDesc,
                void                          *srcDestData,
                const void                    *value
```

This function sets all the elements of a tensor to a given value

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
srcDestDesc	input	Handle to a previously initialized tensor descriptor.
srcDestData	input/ output	Pointer to data of the tensor described by the **srcDestDesc** descriptor.
value	input	Pointer in Host memory to a value that all elements of the tensor will be set to.

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	one of the provided pointers is nil
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.15. cudnnScaleTensor

```
cudnnStatus_t
cudnnScaleTensor( cudnnHandle_t                 handle,
                  const cudnnTensorDescriptor_t srcDestDesc,
                  void                          *srcDestData,
                  const void                    *alpha
```

This function scale all the elements of a tensor by a give factor.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.

Param	In/out	Meaning
srcDestDesc	input	Handle to a previously initialized tensor descriptor.
srcDestData	input/ output	Pointer to data of the tensor described by the **srcDestDesc** descriptor.
alpha	input	Pointer in Host memory to a value that all elements of the tensor will be scaled with.

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	one of the provided pointers is nil
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.16. 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.

4.17. cudnnSetFilter4dDescriptor

```
cudnnStatus_t
cudnnSetFilter4dDescriptor( cudnnFilterDescriptor_t filterDesc,
                            cudnnDataType_t dataType,
                            int k,
                            int c,
                            int h,
                            int w )
```

This function initializes a previously created filter descriptor object into a 4D filter. 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.

Param	In/out	Meaning
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.

4.18. cudnnGetFilter4dDescriptor

```
cudnnStatus_t
cudnnGetFilter4dDescriptor( 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.

4.19. cudnnSetFilterNdDescriptor

```
cudnnStatus_t
cudnnSetFilterNdDescriptor( cudnnFilterDescriptor_t filterDesc,
                            int nbDims,
                            int filterDimA[])
```

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.
nbDims	input	Dimension of the filter.
filterDimA	input	Array of dimension **nbDims** containing the size of the filter for each dimension.

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 elements of the array **filterDimA** is negative or **dataType** has an invalid enumerant value.
CUDNN_STATUS_NOT_SUPPORTED	the parameter **nbDims** exceeds the maximum supported dimension.

4.20. cudnnGetFilterNdDescriptor

```
cudnnStatus_t
cudnnGetFilterNdDescriptor( const cudnnFilterDescriptor_t filterDesc,
                            int nbDimsRequested,
                            cudnnDataType_t *dataType,
                            int *nbDims,
                            int filterDimA[])
```

This function queries a previously initialized filter descriptor object.

Param	In/out	Meaning
filterDesc	input	Handle to a previously initialized filter descriptor.
nbDimsRequested	input	Dimension of the expected filter descriptor. It is also the minimum size of the arrays **filterDimA** in order to be able to hold the results
datatype	input	Data type.
nbDims	input	Actual dimension of the filter.
filterDimA	input	Array of dimension of at least **nbDimsRequested** that will be filled with the filter parameters from the provided filter descriptor.

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	The parameter **nbDimsRequested** is negative.

4.21. 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.

4.22. 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.

4.23. cudnnSetConvolution2dDescriptor

```
cudnnStatus_t
cudnnSetConvolution2dDescriptor( cudnnConvolutionDescriptor_t convDesc,
                                 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 into a 2D correlation. 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.
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.

Param	In/out	Meaning
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.

4.24. cudnnGetConvolution2dDescriptor

```
cudnnStatus_t
cudnnGetConvolution2dDescriptor( const cudnnConvolutionDescriptor_t convDesc,
                                 int* pad_h,
                                 int* pad_w,
                                 int* u,
                                 int* v,
                                 int* upscalex,
                                 int* upscaley,
                                 cudnnConvolutionMode_t *mode )
```

This function queries a previously initialized 2D convolution descriptor object.

Param	In/out	Meaning
convDesc	input/ output	Handle to a previously created convolution descriptor.
pad_h	output	zero-padding height: number of rows of zeros implicitly concatenated onto the top and onto the bottom of input images.
pad_w	output	zero-padding width: number of columns of zeros implicitly concatenated onto the left and onto the right of input images.
u	output	Vertical filter stride.
v	output	Horizontal filter stride.
upscalex	output	Upscale the input in x-direction.
upscaley	output	Upscale the input in y-direction.
mode	output	convolution mode.

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.

4.25. cudnnGetConvolution2dForwardOutputDim

```
cudnnStatus_t
cudnnGetConvolution2dForwardOutputDim( const cudnnConvolutionDescriptor_t
 convDesc,
                                       const cudnnTensorDescriptor_t
 inputTensorDesc,
                                       const cudnnFilterDescriptor_t filterDesc,
                                       int *n,
                                       int *c,
                                       int *h,
                                       int *w )
```

This function returns the dimensions of the resulting 4D tensor of a 2D convolution, given the convolution descriptor, the input tensor descriptor and the filter descriptor 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.
inputTensorDesc	input	Handle to a previously initialized tensor descriptor.
filterDesc	input	Handle to a previously initialized filter descriptor.
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.

4.26. cudnnSetConvolutionNdDescriptor

```
cudnnStatus_t
cudnnSetConvolutionNdDescriptor( cudnnConvolutionDescriptor_t convDesc,
                                 int arrayLength,
                                 int padA[],

                                 int filterStrideA[],
                                 int upscaleA[],
                                 cudnnConvolutionMode_t mode )
```

This function initializes a previously created generic convolution descriptor object into a n-D correlation. 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.
arrayLength	input	Dimension of the convolution.
padA	input	Array of dimension **arrayLength** containing the zero-padding size for each dimension. For every dimension, the padding represents the number of extra zeros implicitly concatenated at the start and at the end of every element of that dimension .
filterStrideA	input	Array of dimension **arrayLength** containing the filter stride for each dimension. For every dimension, the fitler stride represents the number of elements to slide to reach the next start of the filtering window of the next point.
upscaleA	input	Array of dimension **arrayLength** containing the upscale factor for each dimension.
mode	input	Selects between **CUDNN_CONVOLUTION** and **CUDNN_CROSS_CORRELATION**.

4.27. cudnnGetConvolutionNdDescriptor

```
cudnnStatus_t
cudnnGetConvolutionNdDescriptor( const cudnnConvolutionDescriptor_t convDesc,
                                 int arrayLengthRequested,
                                 int *arrayLength,
                                 int padA[],

                                 int filterStrideA[],
                                 int upscaleA[],
                                 cudnnConvolutionMode_t *mode )
```

This function queries a previously initialized convolution descriptor object.

Param	In/out	Meaning
convDesc	input/ output	Handle to a previously created convolution descriptor.

Param	In/out	Meaning
arrayLengthRequested	input	Dimension of the expected convolution descriptor. It is also the minimum size of the arrays **padA**, **filterStrideA** and **upsacleA** in order to be able to hold the results
arrayLength	output	actual dimension of the convolution descriptor.
padA	output	Array of dimension of at least **arrayLengthRequested** that will be filled with the padding parameters from the provided convolution descriptor.
filterStrideA	output	Array of dimension of at least **arrayLengthRequested** that will be filled with the filter stride from the provided convolution descriptor.
upscaleA	output	Array of dimension at least **arrayLengthRequested** that will be filled with the upscaling parameters from the provided convolution descriptor.
mode	output	convolution mode of the provided descriptor.

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

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The query was successfully.
CUDNN_STATUS_BAD_PARAM	The **arrayLenghtRequest** is negative
CUDNN_STATUS_NOT_SUPPORTED	The **arrayLenghtRequest** is greater than the maximum supported dimension

4.28. cudnnGetConvolutionNdForwardOutputDim

```
cudnnStatus_t
cudnnGetConvolutionNdForwardOutputDim( const cudnnConvolutionDescriptor_t
 convDesc,
                                       const cudnnTensorDescriptor_t
 inputTensorDesc,
                                       const cudnnFilterDescriptor_t filterDesc,
                                       int nbDims,
                                       int tensorOuputDimA[] )
```

This function returns the dimensions of the resulting n-D tensor of a **nbDims-2**-D convolution, given the convolution descriptor, the input tensor descriptor and the filter descriptor 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.
inputTensorDesc	input	Handle to a previously initialized tensor descriptor.
filterDesc	input	Handle to a previously initialized filter descriptor.
nbDims	input	Dimension of the output tensor

Param	In/out	Meaning
tensorOuputDimA	output	Array of dimensions **nbDims** that contains on exit of this routine the sizes of the output tensor

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

Return Value	Meaning
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ The dimension of the convolution descriptor is different from **nbDims-2**. ▸ The dimension of the input tensor descriptor is different from **nbDims**. ▸ The dimension of the filter descriptor is different from **nbDims**.
CUDNN_STATUS_SUCCESS	The routine exits successfully.

4.29. 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.

4.30. cudnnGetConvolutionForwardAlgorithm

```
cudnnStatus_t
cudnnGetConvolutionForwardAlgorithm( cudnnHandle_t                       handle,
                                     const cudnnTensorDescriptor_t       srcDesc,
                                     const cudnnFilterDescriptor_t
 filterDesc,
                                     const cudnnConvolutionDescriptor_t
 convDesc,
                                     const cudnnTensorDescriptor_t
 destDesc,
                                     cudnnConvolutionFwdPreference_t
 preference,
                                     size_t
 memoryLimitInbytes,
                                     cudnnConvolutionFwdAlgo_t          *algo
                                   )
```

This function advised the best algorithm to choose for the forward convolution depending on the critera expressed in the **cudnnConvolutionFwdPreference_t** enumerant.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.

Param	In/out	Meaning
srcDesc	input	Handle to the previously initialized input tensor descriptor.
filterDesc	input	Handle to a previously initialized filter descriptor.
convDesc	input	Previously initialized convolution descriptor.
destDesc	input	Handle to the previously initialized output tensor descriptor.
preference	input	Enumerant to express the preference criteria in terms of memory requirement and speed.
memoryLimitInbytes	input	It is used when enumerant **preference** is set to **CUDNN_CONVOLUTION_FWD_SPECIFY_WORKSPACE_LIMIT** to specify the maximum amount of GPU memory the user is willing to use as a workspace
algo	output	Enumerant that specifies which convolution algorithm should be used to compute the results according to the specified preference

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

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The query was successful.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ The numbers of feature maps of the input tensor and output tensor differ. ▸ The **dataType** of the two tensor descriptors or the filter are different.

4.31. cudnnGetConvolutionForwardWorkspaceSize

```
cudnnStatus_t
cudnnGetConvolutionForwardWorkspaceSize( cudnnHandle_t   handle,
                                         const   cudnnTensorDescriptor_t
 srcDesc,
                                         const   cudnnFilterDescriptor_t
 filterDesc,
                                         const   cudnnConvolutionDescriptor_t
 convDesc,
                                         const   cudnnTensor4dDescriptor_t
 destDesc,
                                         cudnnConvolutionFwdAlgo_t
 algo,
                                         size_t
 *sizeInBytes
                                 )
```

This function returns the amount of GPU memory workspace the user needs to allocate to be able to call **cudnnConvolutionForward** with the specified algorithm. The workspace allocated will then be passed to the routine **cudnnConvolutionForward**. The specified algorithm can be the result of the call to **cudnnGetConvolutionForwardAlgorithm** or can be chosen arbitrarily by the user. Note that not every algorithm is available for every configuration of the input tensor and/or every configuration of the convolution descriptor.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
filterDesc	input	Handle to a previously initialized filter descriptor.
convDesc	input	Previously initialized convolution descriptor.
destDesc	input	Handle to the previously initialized output tensor descriptor.
algo	input	Enumerant that specifies the chosen convolution algorithm
sizeInBytes	output	Amount of GPU memory needed as workspace to be able to execute a forward convolution with the sepcified **algo**

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

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The query was successful.
CUDNN_STATUS_BAD_PARAM	At least one of the following conditions are met: ▸ The numbers of feature maps of the input tensor and output tensor differ. ▸ The **dataType** of the two tensor descriptors or the filter are different.
CUDNN_STATUS_NOT_SUPPORTED	The combination of the tensor descriptors, filter descriptor and convolution descriptor is not supported for the specified algorithm.

4.32. cudnnConvolutionForward

```
cudnnStatus_t
cudnnConvolutionForward( cudnnHandle_t                          handle,
                         const void                            *alpha,
                         const cudnnTensorDescriptor_t          srcDesc,
                         const void                            *srcData,
                         const cudnnFilterDescriptor_t          filterDesc,
                         const void                            *filterData,
                         const cudnnConvolutionDescriptor_t     convDesc,
                         cudnnConvolutionFwdAlgo_t              algo,
                         void                                  *workSpace,
                         size_t
 workSpaceSizeInBytes,
                         const void                            *beta,

                         const cudnnTensorDescriptor_t          destDesc,
                         void                                  *destData )
```

This function executes convolutions or cross-correlations over **src** using the specified **filters**, returning results in **dest**. Scaling factors **alpha** and **beta** can be used to scale the input tensor and the output tensor respectively.

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
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.
algo	input	Enumerant that specifies which convolution algorithm shoud be used to compute the results
workSpace	input	Data pointer to GPU memory to a workspace needed to able to execute the specified algorithm. If no workspace is needed for a particular algorithm, that pointer can be nil
workSpaceSizeInBytes	input	Specifies the size in bytes of the provided **workSpace**
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the convolution. Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

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.

4.33. cudnnConvolutionBackwardBias

```
cudnnStatus_t
cudnnConvolutionBackwardBias( cudnnHandle_t                    handle,
                              const void                      *alpha,
                              const cudnnTensorDescriptor_t    srcDesc,
                              const void                      *srcData,
                              const void                      *beta,
                              const cudnnTensorDescriptor_t    destDesc,
                              void                            *destData
                               )
```

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.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the convolution gradient. Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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 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.

4.34. cudnnConvolutionBackwardFilter

```
cudnnStatus_t
cudnnConvolutionBackwardFilter( cudnnHandle_t                          handle,
                                const void                            *alpha,
                                const cudnnTensorDescriptor_t          srcDesc,
                                const void                            *srcData,
                                const cudnnTensorDescriptor_t          diffDesc,
                                const void                            *diffData,
                                const cudnnConvolutionDescriptor_t     convDesc,
                                const void                            *beta,

                                const cudnnFilterDescriptor_t          gradDesc,
                                void                                  *gradData )
```

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.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
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.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the convolution gradient. Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

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. The descriptor **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.

4.35. cudnnConvolutionBackwardData

```
cudnnStatus_t
cudnnConvolutionBackwardData( cudnnHandle_t                       handle,
                              const void                         *alpha,
                              const cudnnFilterDescriptor_t       filterDesc,
                              const void                         *filterData,
                              const cudnnTensorDescriptor_t       diffDesc,
                              const void                         *diffData,
                              const cudnnConvolutionDescriptor_t  convDesc,
                              const void                         *beta,
                              const cudnnTensorDescriptor_t       gradDesc,
                              void                               *gradData

                            );
```

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.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
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.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the convolution gradient. Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

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. The descriptor **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.

4.36. cudnnSoftmaxForward

```
cudnnStatus_t
cudnnSoftmaxForward( cudnnHandle_t                    handle,
                     cudnnSoftmaxAlgorithm_t          algorithm,
                     cudnnSoftmaxMode_t               mode,
                     const void                      *alpha,
                     const cudnnTensorDescriptor_t    srcDesc,
                     const void                      *srcData,
                     const void                      *beta,
                     const cudnnTensorDescriptor_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.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the softmac function. Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

4.37. cudnnSoftmaxBackward

```
cudnnStatus_t
cudnnSoftmaxBackward( cudnnHandle_t                    handle,
                      cudnnSoftmaxAlgorithm_t          algorithm,
                      cudnnSoftmaxMode_t               mode,
                      const void                      *alpha,
                      const cudnnTensorDescriptor_t    srcDesc,
                      const void                      *srcData,
                      const cudnnTensorDescriptor_t    srcDiffDesc,
                      const void                      *srcDiffData,
                      const void                      *beta,
                      const cudnnTensorDescriptor_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.
alpha	input	Scaling factor with which every element of the input tensors is multiplied.
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**.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the softmax gradient. Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

4.38. 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.

4.39. cudnnSetPooling2dDescriptor

```
cudnnStatus_t
cudnnSetPooling2dDescriptor( cudnnPoolingDescriptor_t poolingDesc,
                             cudnnPoolingMode_t mode,
                             int windowHeight,
                             int windowWidth,
                             int verticalPadding,
                             int horizontalPadding,
                             int verticalStride,
                             int horizontalStride )
```

This function initializes a previously created generic pooling descriptor object into a 2D description.

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.
verticalPadding	input	Size of vertical padding.
horizontalPadding	input	Size of horizontal padding
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.

4.40. cudnnGetPooling2dDescriptor

```
cudnnStatus_t
cudnnGetPooling2dDescriptor( const cudnnPoolingDescriptor_t poolingDesc,
                             cudnnPoolingMode_t *mode,
                             int *windowHeight,
                             int *windowWidth,
                             int *verticalPadding,
                             int *horizontalPadding,
                             int *verticalStride,
                             int *horizontalStride )
```

This function queries a previously created 2D 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.
verticalPadding	output	Size of vertical padding.
horizontalPadding	output	Size of horizontal padding.
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.

4.41. cudnnSetPoolingNdDescriptor

```
cudnnStatus_t
cudnnSetPoolingNdDescriptor( cudnnPoolingDescriptor_t poolingDesc,
                             cudnnPoolingMode_t mode,
                             int nbDims,
                             int windowDimA[],
                             int paddingA[],
                             int strideA[] )
```

This function initializes a previously created generic 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.

Param	In/out	Meaning
nbDims	input	Dimension of the pooling operation.
windowDimA	output	Array of dimension **nbDims** containing the window size for each dimension.
paddingA	output	Array of dimension **nbDims** containing the padding size for each dimension.
strideA	output	Array of dimension **nbDims** containing the striding size for each dimension.

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 elements of the arrays **windowDimA**, **paddingA** or **strideA** is negative or **mode** has an invalid enumerant value.

4.42. cudnnGetPoolingNdDescriptor

```
cudnnStatus_t
cudnnGetPoolingNdDescriptor( const cudnnPoolingDescriptor_t poolingDesc,
                             int nbDimsRequested,
                             cudnnPoolingMode_t *mode,
                             int *nbDims,
                             int windowDimA[],
                             int paddingA[],
                             int strideA[] )
```

This function queries a previously initialized generic pooling descriptor object.

Param	In/out	Meaning
poolingDesc	input	Handle to a previously created pooling descriptor.
nbDimsRequested	input	Dimension of the expected pooling descriptor. It is also the minimum size of the arrays **windowDimA**, **paddingA** and **strideA** in order to be able to hold the results
mode	output	Enumerant to specify the pooling mode.
nbDims	output	Actual dimension of the pooling descriptor.
windowDimA	output	Array of dimension of at least **nbDimsRequested** that will be filled with the window parameters from the provided pooling descriptor.
paddingA	output	Array of dimension of at least **nbDimsRequested** that will be filled with the padding parameters from the provided pooling descriptor.
strideA	output	Array of dimension at least **nbDimsRequested** that will be filled with the stride parameters from the provided pooling descriptor.

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

Return Value	Meaning
CUDNN_STATUS_SUCCESS	The object was queried successfully.
CUDNN_STATUS_BAD_PARAM	The parameter **nbDimsRequested** is negative.

4.43. 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.

4.44. cudnnPoolingForward

```
cudnnStatus_t
cudnnPoolingForward( cudnnHandle_t                    handle,
                     const cudnnPoolingDescriptor_t   poolingDesc,
                     const void                      *alpha,
                     const cudnnTensorDescriptor_t    srcDesc,
                     const void                      *srcData,
                     const void                      *beta,
                     const cudnnTensorDescriptor_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.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the pooling. Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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.

4.45. cudnnPoolingBackward

```
cudnnStatus_t
cudnnPoolingBackward( cudnnHandle_t handle,
                      const cudnnPoolingDescriptor_t   poolingDesc,
                      const void                      *alpha,
                      const cudnnTensorDescriptor_t    srcDesc,
                      const void                      *srcData,
                      const cudnnTensorDescriptor_t    srcDiffDesc,
                      const void                      *srcDiffData,
                      const cudnnTensorDescriptor_t    destDesc,
                      const void                      *destData,
                      const void                      *beta,
                      const cudnnTensorDescriptor_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.
alpha	input	Scaling factor with which every element of the input tensors is multiplied.
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**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	input	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the pooling gradient. Note that if **beta** is zero, the

Param	In/out	Meaning
		output is not read and can contain any uninitialized data (including Nan numbers)
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.

4.46. cudnnActivationForward

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

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

In-place operation is allowed for this routine; i.e., **srcData** and **destData** pointers may be equal. However, this requires **srcDesc** and **destDesc** descriptors to be identical (particularly, the strides of the input and output must match for in-place operation to be allowed).

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
mode	input	Enumerant to specify the activation mode.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
srcDesc	input	Handle to the previously initialized input tensor descriptor.
srcData	input	Data pointer to GPU memory associated with the tensor descriptor **srcDesc**.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the activation Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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 parameter **mode** has an invalid enumerant value. ▸ The dimensions **n,c,h,w** of the input tensor and output tensors differ. ▸ The **datatype** of the input tensor and output tensors differs. ▸ The strides **nStride,cStride,hStride,wStride** of the input tensor and output tensors differ and in-place operation is used (i.e., **srcData** and **destData** pointers are equal).
CUDNN_STATUS_EXECUTION_FAILED	The function failed to launch on the GPU.

4.47. cudnnActivationBackward

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

This routine computes the gradient of a neuron activation function.

In-place operation is allowed for this routine; i.e., **srcData** and **destData** pointers may be equal and **srcDiffData** and **destDiffData** pointers may be equal. However, this requires the corresponding tensor descriptors to be identical (particularly, the strides of the input and output must match for in-place operation to be allowed).

Param	In/out	Meaning
handle	input	Handle to a previously created cuDNN context.
mode	input	Enumerant to specify the activation mode.
alpha	input	Scaling factor with which every element of the input tensor is multiplied.
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**.
destDesc	input	Handle to the previously initialized output tensor descriptor.
destData	input	Data pointer to GPU memory associated with the output tensor descriptor **destDesc**.
beta	input	Scaling factor which is applied on every element of the output tensor prior to adding the result of the activation gradient Note that if **beta** is zero, the output is not read and can contain any uninitialized data (including Nan numbers)
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 parameter **mode** has an invalid enumerant value. ▸ The dimensions **n,c,h,w** of the input tensor and output tensors differ. ▸ The **datatype** of the input tensor and output tensors differs. ▸ The strides **nStride, cStride, hStride, wStride** of the input tensor and output tensors differ and in-place operation is used.

Return Value	Meaning
`CUDNN_STATUS_NOT_SUPPORTED`	At least one of the following conditions are met: ▸ 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.
`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 others and are subject to the following:

5.1. University of Tennessee


```
Copyright (c) 2010 The University of Tennessee.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer listed in this license in the documentation and/or
      other materials provided with the distribution.
    * Neither the name of the copyright holders nor the names of its
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

5.2. University of California, Berkeley

```
COPYRIGHT

All contributions by the University of California:
Copyright (c) 2014, The Regents of the University of California (Regents)
All rights reserved.

All other contributions:
Copyright (c) 2014, the respective contributors
All rights reserved.

Caffe uses a shared copyright model: each contributor holds copyright over
their contributions to Caffe. The project versioning records all such
contribution and copyright details. If a contributor wants to further mark
their specific copyright on a particular contribution, they should indicate
their copyright solely in the commit message of the change when it is
committed.

LICENSE

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CONTRIBUTION AGREEMENT

By contributing to the BVLC/caffe repository through pull-request, comment,
or otherwise, the contributor releases their content to the
license and copyright terms herein.
```

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2014 NVIDIA Corporation. All rights reserved.