NVIDIA cuDNN

Best Practices | NVIDIA Docs

BPG-09678-001_v8.2.4 | August 2021

Table of Contents

List of Tables

Chapter 1. Introduction

> **ATTENTION:** These guidelines are applicable to 3D convolution and deconvolution functions starting in NVIDIA® CUDA® Deep Neural Network library™ (cuDNN) v7.6.3.

This document provides guidelines for setting the cuDNN library parameters to enhance the performance of 3D convolutions. Specifically, these guidelines are focused on settings such as filter sizes, padding and dilation settings. Additionally, an application-specific use-case, namely, medical imaging, is presented to demonstrate the performance enhancement of 3D convolutions with these recommended settings.

Specifically, these guidelines are applicable to the following functions and their associated data types:

- cudnnConvolutionForward()
- cudnnConvolutionBackwardData()
- cudnnConvolutionBackwardFilter()

For more information, see the cuDNN Developer Guide and cuDNN API.

Chapter 2. Best Practices For Medical Imaging

To optimize your performance in your model, ensure you meet the following general guidelines:

Layout

The layout is in NCHW format.

Filter size

The filter size is `Tx1x1`, `Tx2x2`, `Tx3x3`, `Tx5x5`, where `T` is a positive integer. There are additional limits for the value of `T` in `wgrad` and strided `dgrad`.

Stride

Arbitrary for forward and backward filter; `dgrad`/`deconv`: 1x1x1 or 2x2x2 with 2x2x2 filter.

Dilation

The dilation is 1x1x1.

Platform

The platform is Volta, Turing, and Ampere with input/output channels divisible by 8.

Batch/image size

cuDNN will fallback to non-Tensor Core kernel if it determines that the workspace required is larger than 256MB of GPU memory. The workspace required depends on many factors. For the Tensor Core kernels, the workspace size generally scales linearly with output tensor size. Therefore, this can be mitigated by using smaller image sizes or mini-batch sizes.

2.1. Recommended Settings In cuDNN While Performing 3D Convolutions

The following tables show the specific improvements that were made in each release.

2.1.1. cuDNN 8.x.x Recommended Settings

Recommended settings while performing 3D convolutions for cuDNN 8.x.x.

		8.0.3 - 8.2.4	8.0.0 and 8.0.1 Preview - 8.0.2
Platform		NVIDIA Ampere GPU architecture NVIDIA Turing GPU architecture NVIDIA Volta GPU architecture	
Convolution (3D or 2D)		3D and 2D	

Convolution or deconvolution (`fprop`, `dgrad`, or `wgrad`)		`fprop`
		`dgrad`
		`wgrad`	

Grouped convolution	Yes or No	Yes	
	Group size	`C_per_group == K_per_group == {4,8,16,32,64,128,256}`	`C_per_group == K_per_group == {4,8,16,32}`
Data layout format (`NHWC`/`NCHW`)[1]		NDHWC	
Input/output precision (FP16, FP32, or FP64)		FP16 and FP32[2]	
Accumulator (compute) precision (FP16, FP32, or FP64)		FP32	
Filter (kernel) sizes		No limitation	
Padding		No limitation	
Image sizes		2GB limitation for a tensor	
Number of channels	C	`0 mod 8`	
	K	`0 mod 8`	
Convolution mode		Cross-correlation and convolution	
Strides		`dgrad`: 1x1x1 or 2x2x2	
Dilation		No limitation	
Data pointer alignment		All data pointers are 16-bytes aligned.	

2.1.2. cuDNN 7.6.x Recommended Settings

Recommended settings while performing 3D convolutions for cuDNN 7.6.x.

	7.6.5	7.6.4	7.6.2	7.6.1
Platform	Turing Volta	Volta		
Convolution (3D or 2D)	3D and 2D	3D		
Convolution or deconvolution (`fprop`, `dgrad`, or `wgrad`)	`fprop` `dgrad`		`fprop` `dgrad`	`fprop` `dgrad`

[1] `NHWC`/`NCHW` corresponds to `NDHWC`/`NCDHW` in 3D convolution.

[2] With `CUDNN_TENSOROP_MATH_ALLOW_CONVERSION` pre-Ampere. Default TF32 math in Ampere.

		7.6.5	7.6.4	7.6.2	7.6.1
		`wgrad`			`wgrad`
Grouped convolution	Yes or No	Yes		No	
	Group size	`C_per_group == K_per_group == {4,8,16,32}`		NA	
Data layout format (`NHWC`/`NCHW`)[3]		NCDHW			NCDHW[4]
Input/output precision (FP16, FP32, or FP64)		FP16		FP16 or FP32	FP16[5] or FP32[6]
Accumulator (compute) precision (FP16, FP32, or FP64)		FP32		Better to be the same with input/output precision.	FP32
Filter (kernel) sizes		2x2x2 T[7]x1x1 Tx2x2 Tx3x3 Tx5x5			1x1x1 2x2x2 3x3x3 5x5x5 Tx1x1 Tx2x2 Tx3x3 Tx5x5 Tx1x1 Tx2x2 Tx3x3 Tx5x5
Padding		No limitation			Filter // 2[8]
Image sizes		256 MB WS limit		No limitation	256 MB WS limit
Number of channels	C	Arbitrary			0 mod 8
	K	Arbitrary			0 mod 8
Convolution mode		Cross-correlation for `dgrad`; otherwise, both modes		No limitation Cross-correlation	
Strides		1x1x1 and 2x2x2 strides for `dgrad`		2x2x2	1x1x1

[3] `NHWC`/`NCHW` corresponds to `NDHWC`/`NCDHW` in 3D convolution.
[4] With NCHW <> NHWC format transformation.
[5] FP16: `CUDNN_TENSOROP_MATH`
[6] FP32: `CUDNN_TENSOROP_MATH_ALLOW_CONVERSION`
[7] An arbitrary positive value.
[8] `padding = filter // 2`

	7.6.5	7.6.4	7.6.2	7.6.1
			Arbitrary stride	
Dilation	1x1x1			

Chapter 3. Medical Imaging Performance

The following table shows the average speed-up of **unique cuDNN 3D convolution calls** for each network on V100 and A100 GPUs that satisfies the conditions in Best Practices For Medical Imaging. The end-to-end training performance will depend on a number of factors, such as framework overhead, kernel run time, and model architecture type.

3.1. Average Speedup Of Unique cuDNN 3D Convolutions API Calls

3.1.1. cuDNN 8.x.x Average Speedup

cuDNN version 8.2.4 compared to 7.6.5

Table 1. Average speed-up of unique cuDNN (version 8.2.4 compared to 7.6.5) 3D convolution API calls on V100 and A100 for both FP16 and FP32

Model	Batchsize	A100 8.2.4 vs V100 7.6.5		V100 8.2.4 vs V100 7.6.5	
		FP16	FP32	FP16	FP32
V-Net (3D-Image segmentation)	2	2.4x	7.4x	2.3x	2.5x
	8	3.6x	6.3x	2.6x	1.7x
	16	4.4x	7.5x	2.7x	2.1x
	32	6.5x	5.7x	3.5x	1.6x
3D-UNet (3D-Image Segmentation)	2	8.0x	7.1x	3.9x	1.5x
	4	12.6x	6.3x	5.9x	1.5x

cuDNN version 8.2.2 compared to 7.6.5

Table 2. Average speed-up of unique cuDNN (version 8.2.2 compared to 7.6.5) 3D convolution API calls on V100 and A100 for both FP16 and FP32.

Model	Batchsize	A100 8.2.2 vs V100 7.6.5		V100 8.2.2 vs V100 7.6.5	
		FP16	FP32	FP16	FP32
V-Net (3D-Image segmentation)	2	2.4x	7.5x	2.2x	2.5x
	8	3.6x	6.3x	2.6x	1.7x
	16	4.4x	7.5x	2.7x	2.1x
	32	6.5x	5.7x	3.5x	1.6x
3D-UNet (3D-Image Segmentation)	2	8.0x	7.1x	3.9x	1.5x
	4	12.6x	6.3x	5.9x	1.5x

cuDNN version 8.2.1 compared to 7.6.5

Table 3. Average speed-up of unique cuDNN (version 8.2.1 compared to 7.6.5) 3D convolution API calls on V100 and A100 for both FP16 and FP32.

Model	Batchsize	A100 8.2.1 vs V100 7.6.5		V100 8.2.1 vs V100 7.6.5	
		FP16	FP32	FP16	FP32
V-Net (3D-Image segmentation)	2	2.5x	7.7x	2.2x	2.5x
	8	3.7x	6.4x	2.6x	1.7x
	16	4.5x	7.5x	2.7x	2.1x
	32	6.5x	5.7x	3.6x	1.6x
3D-UNet (3D-Image Segmentation)	2	8.3x	7.3x	3.8x	1.5x
	4	12.7x	6.4x	5.8x	1.5x

cuDNN version 8.2.0 compared to 7.6.5

Table 4. Average speed-up of unique cuDNN (version 8.2.0 compared to 7.6.5) 3D convolution API calls on V100 and A100 for both FP16 and FP32.

Model	Batchsize	A100 8.2.0 vs V100 7.6.5		V100 8.2.0 vs V100 7.6.5	
		FP16	FP32	FP16	FP32
V-Net (3D-Image segmentation)	2	2.3x	7.3x	2.2x	2.5x
	8	3.4x	5.9x	2.4x	1.8x
	16	4.1x	6.8x	2.5x	2.1x
	32	5.8x	5.1x	3.3x	1.6x
3D-UNet (3D-Image Segmentation)	2	6.8x	5.9x	3.4x	1.5x
	4	10.5x	2.6x	5.1x	1.6x

cuDNN version 8.1.1 compared to 7.6.5

Table 5. Average speed-up of unique cuDNN (version 8.1.1 compared to 7.6.5) 3D convolution API calls on V100 and A100 for both FP16 and FP32.

Model	Batchsize	A100 8.1.1 vs V100 7.6.5		V100 8.1.1 vs V100 7.6.5	
		FP16	FP32	FP16	FP32
V-Net (3D-Image segmentation)	2	2.3x	6.8x	2.1x	2.4x
	8	3.2x	5.1x	2.3x	1.8x
	16	3.8x	5.9x	2.3x	2.1x
	32	5.4x	4.4x	3.1x	1.6x
3D-UNet (3D-Image Segmentation)	2	7.2x	6.3x	3.4x	1.5x
	4	11x	2.6x	4.9x	1.6x

cuDNN version 8.1.0 compared to 7.6.5

Table 6. Average speed-up of unique cuDNN (version 8.1.0 compared to 7.6.5) 3D convolution API calls on V100 and A100 for both FP16 and FP32.

Model	Batchsize	A100 8.1.0 vs V100 7.6.5		V100 8.1.0 vs V100 7.6.5	
		FP16	FP32	FP16	FP32
V-Net (3D-Image segmentation)	2	2.4x	7.3x	2.2x	2.4x
	8	3.4x	5.3x	2.3x	1.8x
	16	3.9x	6x	2.3x	2.1x
	32	5.5x	4.4x	3.1x	1.6x
3D-UNet (3D-Image Segmentation)	2	7.3x	6.4x	3.5x	1.5x
	4	11.2x	2.6x	5x	1.6x

Chapter 4. Medical Imaging Limitations

Your application will be functional but slow if the model has:

- Channel counts lower than 32 (gets worse the lower it is)
- Data gradients for convolutions with stride

If the above is in the network, use `cuDNNFind` to get the best option.

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