CNN INFERENCE WITH cuDNN

Chris Hebert, Sven Middelberg, March 21, 2019

AGENDA

Introduction to cuDNN

cuDNN Best Practices:

- Memory Management Done Right
- Choosing the Right Convolution Algorithm & Tensor Layout
- Tensor Cores: Low Precision Inference at Speed of Light

From Research to Production: It just works ... or not?!

Summary

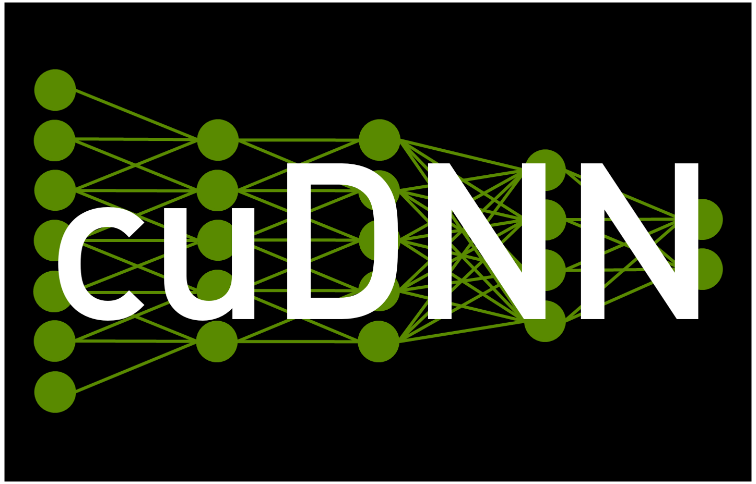
cuDNN ?

cuDNN Provides a set of common network operations

- Convolution
- Activation
- Tensor Ops – Add, multiply etc

Highly optimized for respective HW architectures

cuDNN is the backend for most DL frameworks that target NVIDIA Hardware



HELLO CONVOLUTION

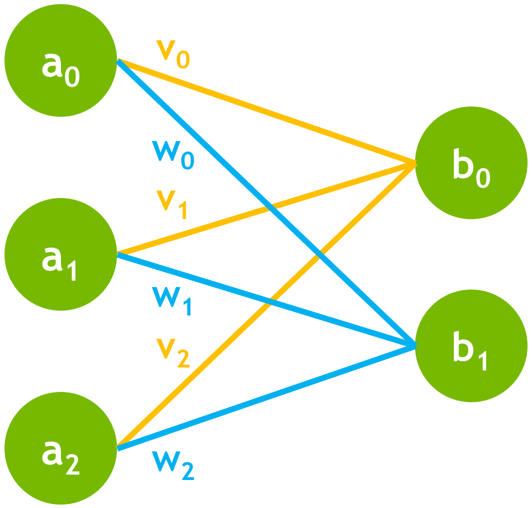
- ✓ Context Management
- ✓ Input and Output Tensors
- ✓ Convolution Filter
- ✓ Convolution Descriptor & Algorithm
- ✓ Convolution Workspace
- ✓ Convolution Computation

```
cudnnHandle_t ctx;
cudnnCreate(&ctx);

cudnnTensorDescriptor_t in_desc;
cudnnCreateTensorDescriptor(&in_desc);
cudnnSetTensorDescriptor(in_desc, CUDNN_TENSOR_NCHW, CUDNN_DATA_FLOAT, 1, 1920, 1080, 4);
// Same for out_desc

cudnnFilterDescriptor_t filter_desc;
cudnnCreateFilterDescriptor(&filter_desc);
cudnnSetFilter4dDescriptor(filter_desc, CUDNN_DATA_FLOAT, CUDNN_TENSOR_NCHW, 8, 4, 3, 3);

cudnnConvolutionDescriptor_t conv_desc;
cudnnCreateConvolutionDescriptor(&conv_desc);
cudnnSetConvolution2dDescriptor(conv_desc, 1, 1, 2, 2, 1, 1, CUDNN_CONVOLUTION, CUDNN_DATA_FLOAT);
cudnnConvolutionFwdAlgo_t conv_alg = CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM;

size_t workspace_size; void* workspace;
cudnnGetConvolutionForwardWorkspaceSize(ctx, in_desc, filter_desc, conv_desc, out_desc, conv_alg, &workspace_size);
cudaMalloc(&workspace, workspace_size);

// Allocate and initialize device data ...

float alpha = 1.0f; float beta = 0.0f;
cudnnStatus_t status = cudnnConvolutionForward(ctx, &alpha, in_desc, in_dev, filter_desc, weights_dev, conv_desc, conv_alg, workspace, workspace_size,
                                               &beta, out_desc, out_dev);
```

FULLY-CONNECTED LAYERS

cuDNN has no native support for fully-connected layers

Forward pass of fully-connected layers is basically a matrix vector multiply



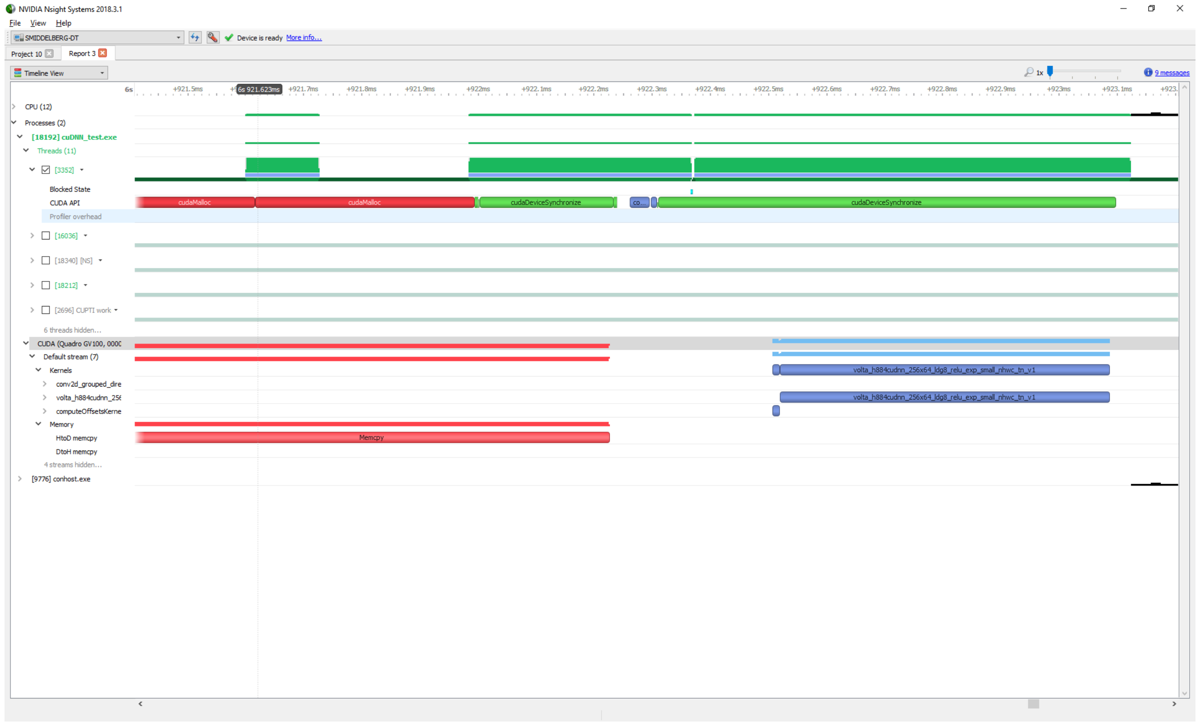
$$\begin{pmatrix} v_0 & v_1 & v_2 \\ w_0 & w_1 & w_2 \end{pmatrix} * \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$$

➢ cuBLAS

cuDNN PROFILING

Profiling Tools

- NVIDIA Nsight Systems / NVVP
- nvprof
- NVIDIA Nsight Compute
  Custom CUDA kernel profiling



GPU timing using CUDA events:

```
cudaEvent_t start, stop;
cudaEventCreate(&start); cudaEventCreate(&stop);

cudaEventRecord(start);
// Do something on GPU
cudaEventRecord(stop);

cudaEventSynchronize(stop);

float ms;
cudaEventElapsedTime(&ms, start, stop);
```

TensorRT & cuDNN

TensorRT	cuDNN

TensorRT & cuDNN

TensorRT		cuDNN
High: Uses cuDNN internally	Level of abstraction	Low: Basic DL primitives
Low: Only custom layers must be implemented manually	Programming effort	High: Network must be assembled manually

TensorRT & cuDNN

TensorRT		cuDNN
High: Uses cuDNN internally	Level of abstraction	Low: Basic DL primitives
Low: Only custom layers must be implemented manually	Programming effort	High: Network must be assembled manually
High: Allows to import models from many training frameworks	Compatibility support for other APIs	None: Models must be imported manually

TensorRT & cuDNN

TensorRT		cuDNN
High: Uses cuDNN internally	Level of abstraction	Low: Basic DL primitives
Low: Only custom layers must be implemented manually	Programming effort	High: Network must be assembled manually
High: Allows to import models from many training frameworks	Compatibility support for other APIs	None: Models must be imported manually
Low: Fully-automatic inference optimization	Optimization effort	High: Needs extensive profiling and optimization

TensorRT & cuDNN

TensorRT		cuDNN
High: Uses cuDNN internally	Level of abstraction	Low: Basic DL primitives
Low: Only custom layers must be implemented manually	Programming effort	High: Network must be assembled manually
High: Allows to import models from many training frameworks	Compatibility support for other APIs	None: Models must be imported manually
Low: Fully-automatic inference optimization	Optimization effort	High: Needs extensive profiling and optimization
Good	Performance	Potentially better (depending on effort)

TensorRT & cuDNN

TensorRT		cuDNN
High: Uses cuDNN internally	Level of abstraction	Low: Basic DL primitives
Low: Only custom layers must be implemented manually	Programming effort	High: Network must be assembled manually
High: Allows to import models from many training frameworks	Compatibility support for other APIs	None: Models must be imported manually
Low: Fully-automatic inference optimization	Optimization effort	High: Needs extensive profiling and optimization
Good	Performance	Potentially better (depending on effort)
Low: Re-optimization needed for any change in architecture or input dimensions	Mutability	High: Usually requires very few optimizations after reasonable changes in input / architecture

AGENDA

Introduction to cuDNN

cuDNN Best Practices:

- Memory Management Done Right
- Choosing the Right Convolution Algorithm & Tensor Layout
- Tensor Cores: Low Precision Inference at Speed of Light

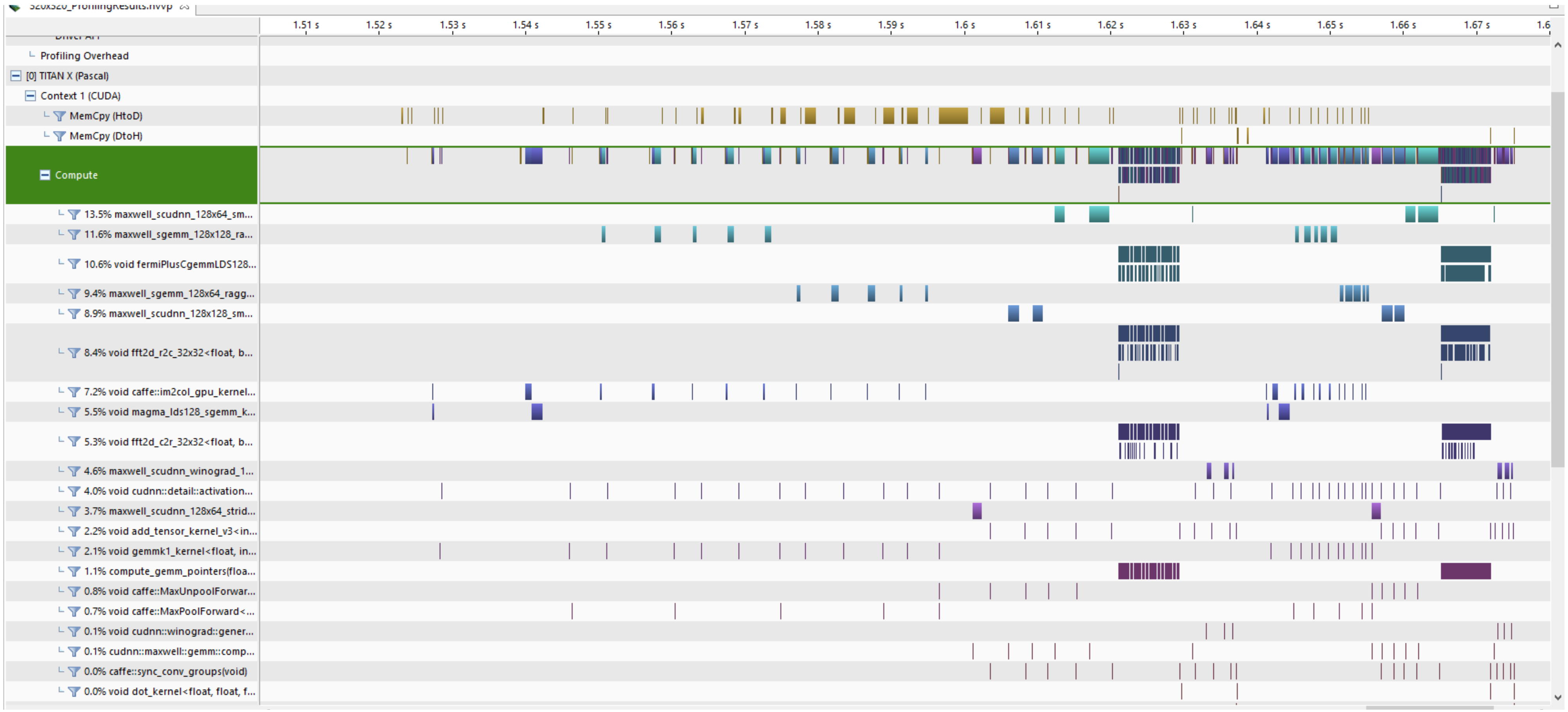
From Research to Production: It just works ... or not?!

Summary

AUTOENCODER IN CAFFE

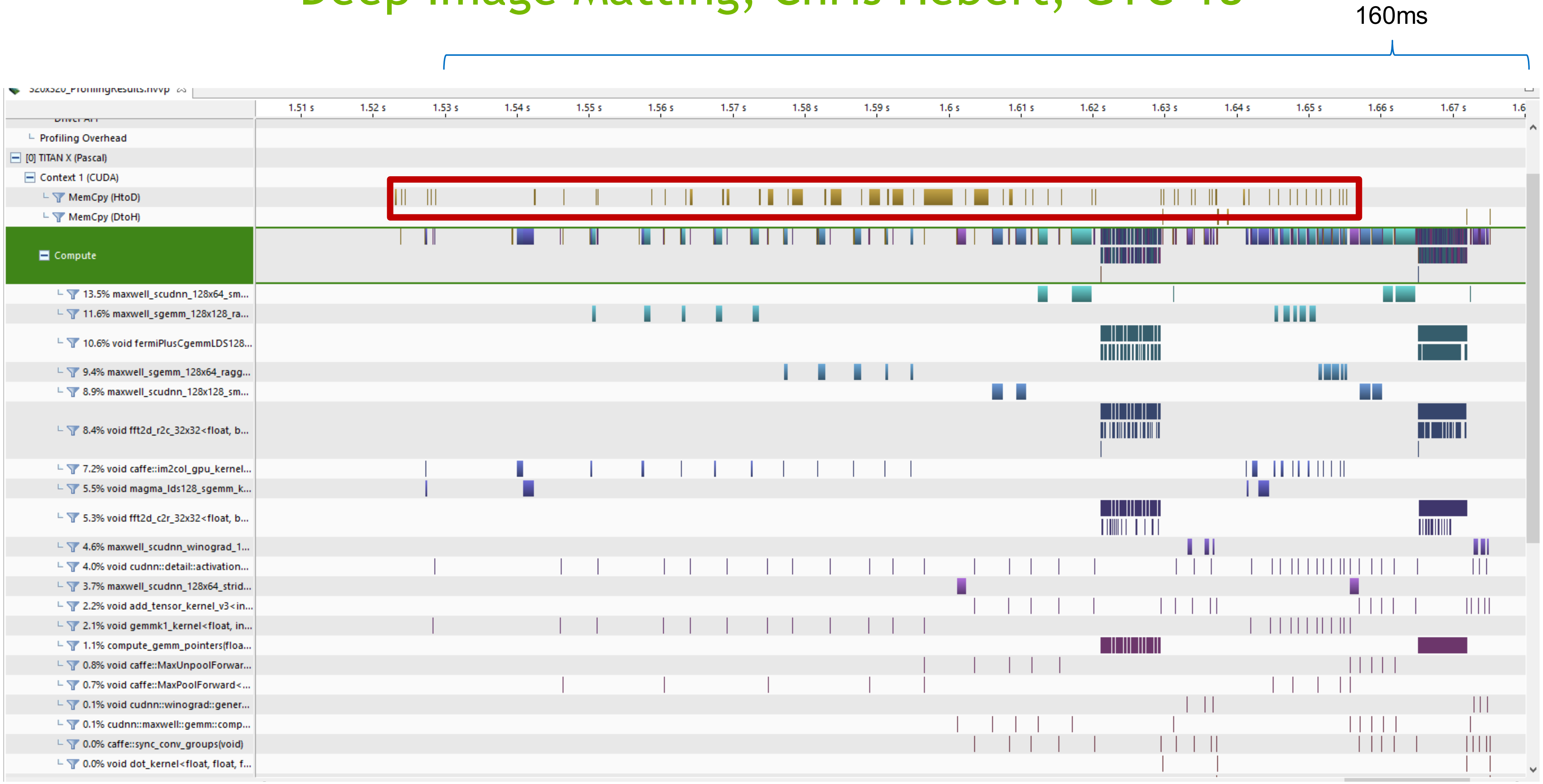
Deep Image Matting, Chris Hebert, GTC'18

160ms



AUTOENCODER IN CAFFE

Deep Image Matting, Chris Hebert, GTC'18



CUDNN DEVICE MEMORY MANAGEMENT

Minimize Footprint - Maximize Performance

Don‘ts

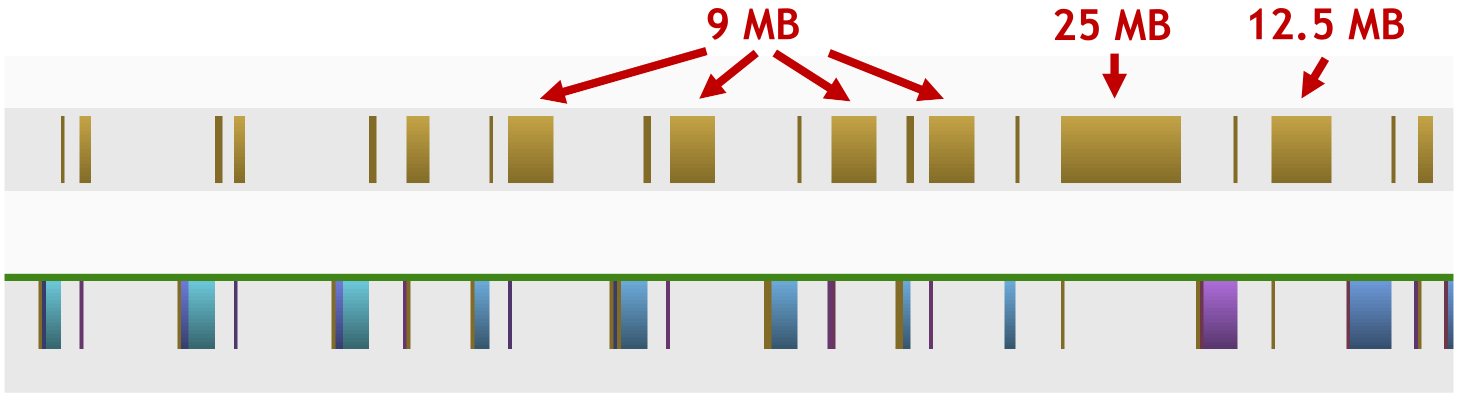
- ✗ Be wasteful with allocations
- ✗ Interleaving allocations with kernel execution
- ✗ Memcpy layer weights on-the-fly, one H2D-copy per layer per forward pass
- ✗ Copy synchroneously from pageable host memory

Dos

- ✓ Maximize reuse
- ✓ Allocate once at startup, use every forward pass
- ✓ Initialize layer weights once, copy only the input and output tensors per forward pass
- ✓ Copy asynchroneously from page-locked host memory, maximize overlap and resource utilization

INITIALIZING WEIGHTS AT STARTUP

Deep Image Matting, Chris Hebert, GTC'18

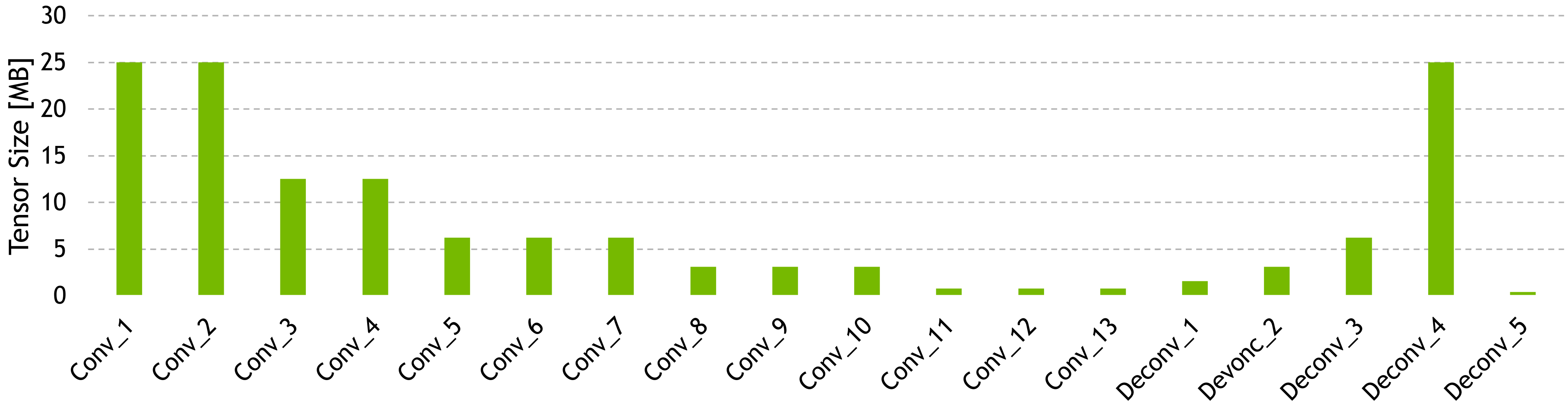


Overall weights data transferred for **each inference** is **~89 MB ⇒ 8.7 ms @10Gb/s**

Allocate and initialize **once at startup**, **reuse each inference**!

TENSOR DEVICE MEMORY

Deep Image Matting, Chris Hebert, GTC'18



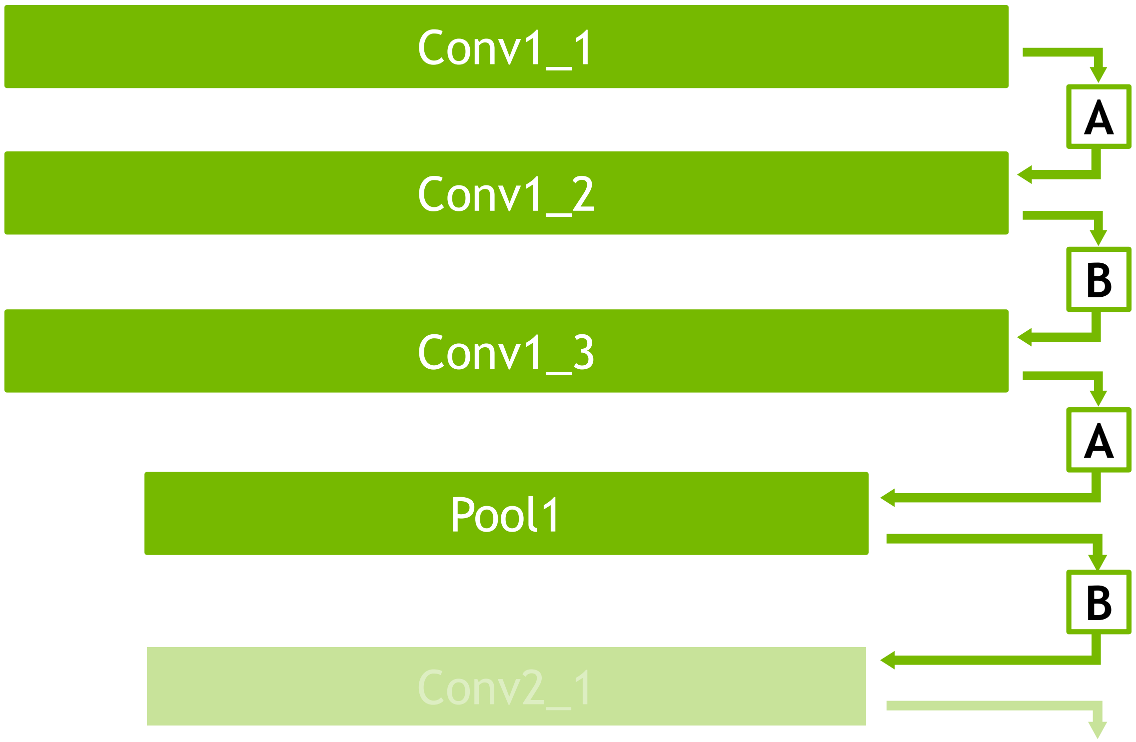
Combined size of all output tensors is **141.8 MB**

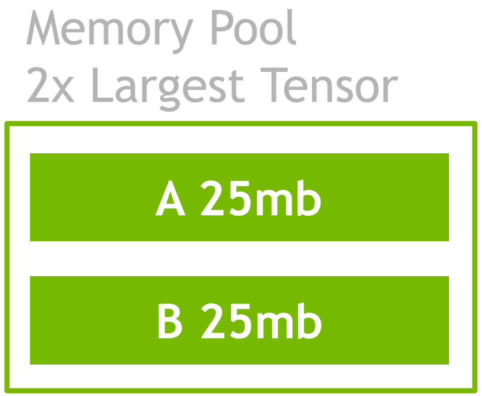
Only two tensors used at a time: input & output tensor

➢ Allocate two times the maximum tensor size: **50 MB**

MINIMIZING MEMORY FOOTPRINT

“Ping-Pong” Tensor Memory





Doesn‘t work for cached tensors, e.g., skip links!

MINIMIZING MEMORY FOOTPRINT

Workspace Memory

Size of a convolution workspace varies, depending on multiple parameters:

- input and output tensor dimensions
- Precisions
- Convolution algorithm
- ...

But: workspace can be shared among layers

➢ **Allocate maximum workspace size!**

OPTIMIZING RESOURCE UTILIZATION

Maximum Inference Throughput by Asynchronous Copies

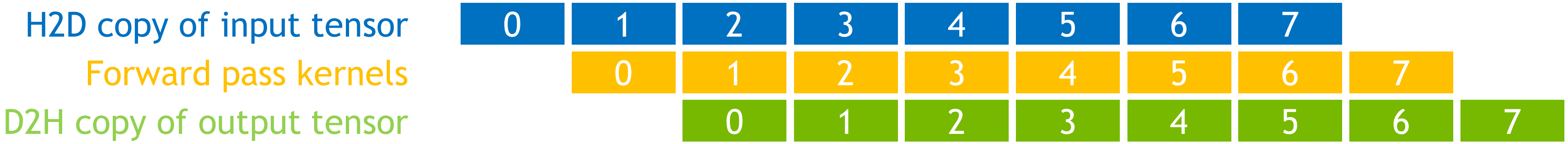
Use page-locked host memory for H2D and D2H memcpys:

`cudaMallocHost(…)` instead of `malloc(…)`

Use asynchronous memcpys:

`cudaMemcpyAsync(…, stream)` instead of `cudaMemcpy(…)`

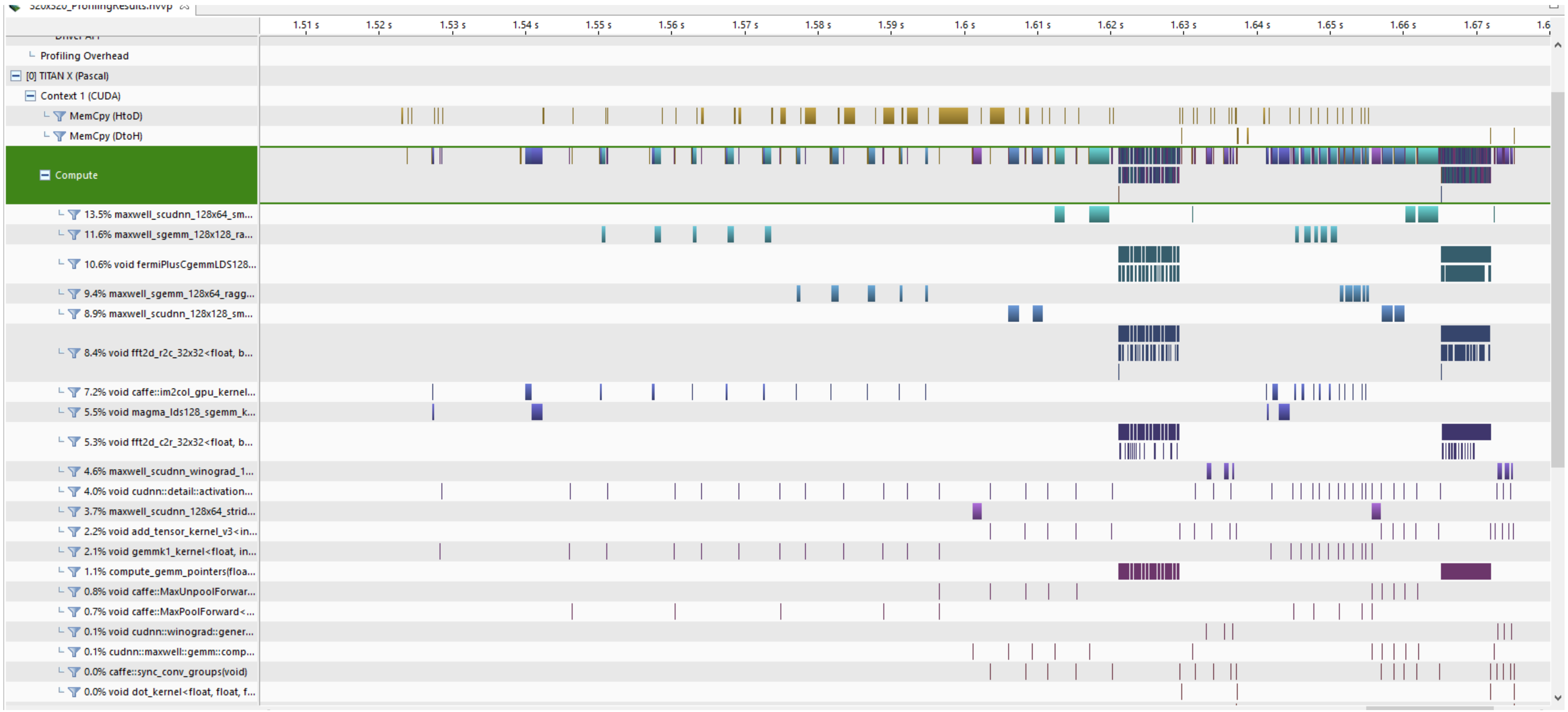
Use multiple CUDA streams to overlap memcpys and kernel executions:



AUTOENCODER IN CAFFE

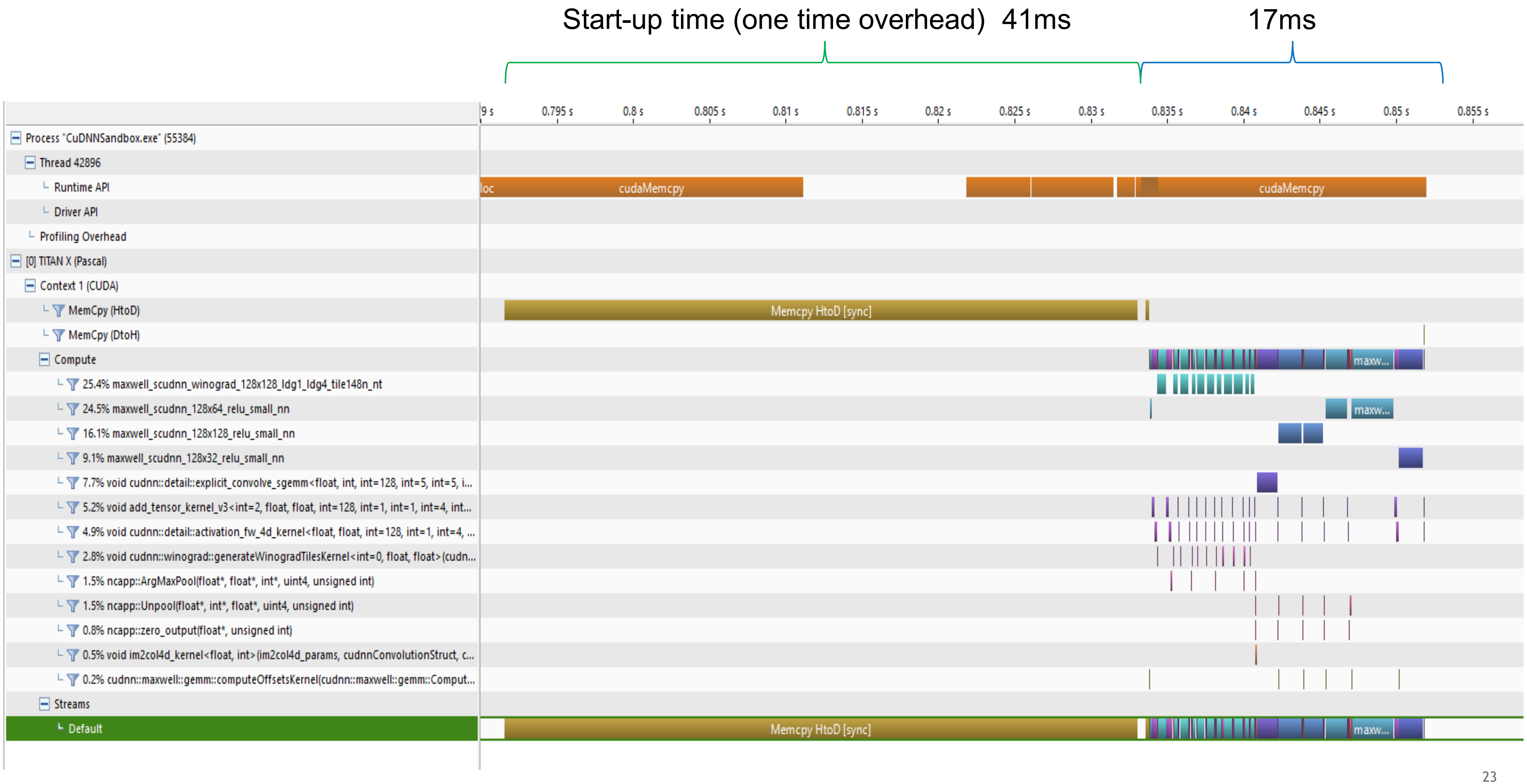
Deep Image Matting, Chris Hebert, GTC'18

160ms



AUTOENCODER IN cuDNN

Deep Image Matting, Chris Hebert, GTC'18



AGENDA

Introduction to cuDNN

cuDNN Best Practices:

- Memory Management Done Right
- **Choosing the Right Convolution Algorithm & Tensor Layout**
- Tensor Cores: Low Precision Inference at Speed of Light

From Research to Production: It just works ... or not?!

Summary

CONVOLUTION ALGORITHMS

128x128x128x128 convolution, FP32, NCHW, Quadro GV100

CUDNN_CONVOLUTION_FWD_ALGO_ …	3 x 3		11 x 11	
	Performance	Workspace	Performance	Workspace
CUDNN_CONVOLUTION_FWD_ALGO_GEMM				
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM				
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM				
CUDNN_CONVOLUTION_FWD_ALGO_FFT				
CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING				
CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD				
CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED				

CONVOLUTION ALGORITHMS

128x128x128x128 convolution, FP32, NCHW, Quadro GV100

CUDNN_CONVOLUTION_FWD_ALGO_ …	3 x 3		11 x 11	
	Performance	Workspace	Performance	Workspace
CUDNN_CONVOLUTION_FWD_ALGO_GEMM	0.76 ms	72 MB		
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM	0.62 ms	None		
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM	0.47 ms	0.01 MB		
CUDNN_CONVOLUTION_FWD_ALGO_FFT	45.3 ms	8322 MB		
CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING	3.69 ms	70 MB		
CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD	0.26 ms	1.56 MB		
CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED	2.73 ms	578 MB		

CONVOLUTION ALGORITHMS

128x128x128x128 convolution, FP32, NCHW, Quadro GV100

CUDNN_CONVOLUTION_FWD_ALGO_ …	3 x 3		11 x 11	
	Performance	Workspace	Performance	Workspace
CUDNN_CONVOLUTION_FWD_ALGO_GEMM	0.76 ms	72 MB	8.47 ms	968 MB
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM	0.62 ms	None	6.82 ms	None
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM	0.47 ms	0.01 MB	6.58 ms	0.01 MB
CUDNN_CONVOLUTION_FWD_ALGO_FFT	45.3 ms	8322 MB	44.7 ms	8328 MB
CUDNN_CONVOLUTION_FWD_ALGO_FFT_TILING	3.69 ms	70 MB	5.13 ms	70 MB
CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD	0.26 ms	1.56 MB	Unsupported	
CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED	2.73 ms	578 MB	Unsupported	

CONVOLUTION ALGORITHMS

Choice depends on memory and performance requirements

Some algorithms not supported for certain convolution configurations

Chosing the most suitable and supported algorithm, layer by layer, is a tedious job

- `cudnnGetConvolutionForwardAlgorithm_v7(...)`

  Based on heuristics: List of algorithms sorted by expected runtime

- `cudnnFindConvolutionForwardAlgorithm(...)`

  More accurate results based on exhaustive experiments

TENSOR LAYOUT

NCHW vs NHWC

Input Tensor Size	Output Tensor Size	Filter Size	NCHW	NHWC
32 x 32 x 64	16 x 16 x 128	3 x 3	0.05 ms	0.06 ms
128 x 128 x 128	128 x 128 x 128	3 x 3	0.26 ms	0.50 ms
512 x 512 x 32	256 x 256 x 64	5 x 5	0.56 ms	1.06 ms
1920 x 1080 x 3	1920 x 1080 x 32	5 x 5	0.97 ms	1.36 ms
16 x 16 x 128	8 x 8 x 256	7 x 7	0.40 ms	0.41 ms
1920 x 1080 x 4	960 x 540 x 32	9 x 9	1.22 ms	1.34 ms
128 x 128 x 128	128 x 128 x 128	11 x 11	5.13 ms	5.72 ms

Convolution algorithm selected using cudnnFindConvolutionForwardAlgorithm(...)

TENSOR LAYOUT

Usually NCHW is faster than NHWC for tensor data

Might be reasonable to pre-convert NHWC input tensor to NCHW and back after the inference to achieve optimal throughput

⇒ **Profile!**

One exception ...

AGENDA

Introduction to cuDNN

cuDNN Best Practices:

- Memory Management Done Right
- Choosing the Right Convolution Algorithm & Tensor Layout
- Tensor Cores: Low Precision Inference at Speed of Light

From Research to Production: It just works ... or not?!

Summary

LOW PRECISION INFERENCE

In most cases FP16 / half provides more than adequate precision for image processing

Volta and Turing have hardware for FAST fp16 – TRUE_HALF_CONFIG

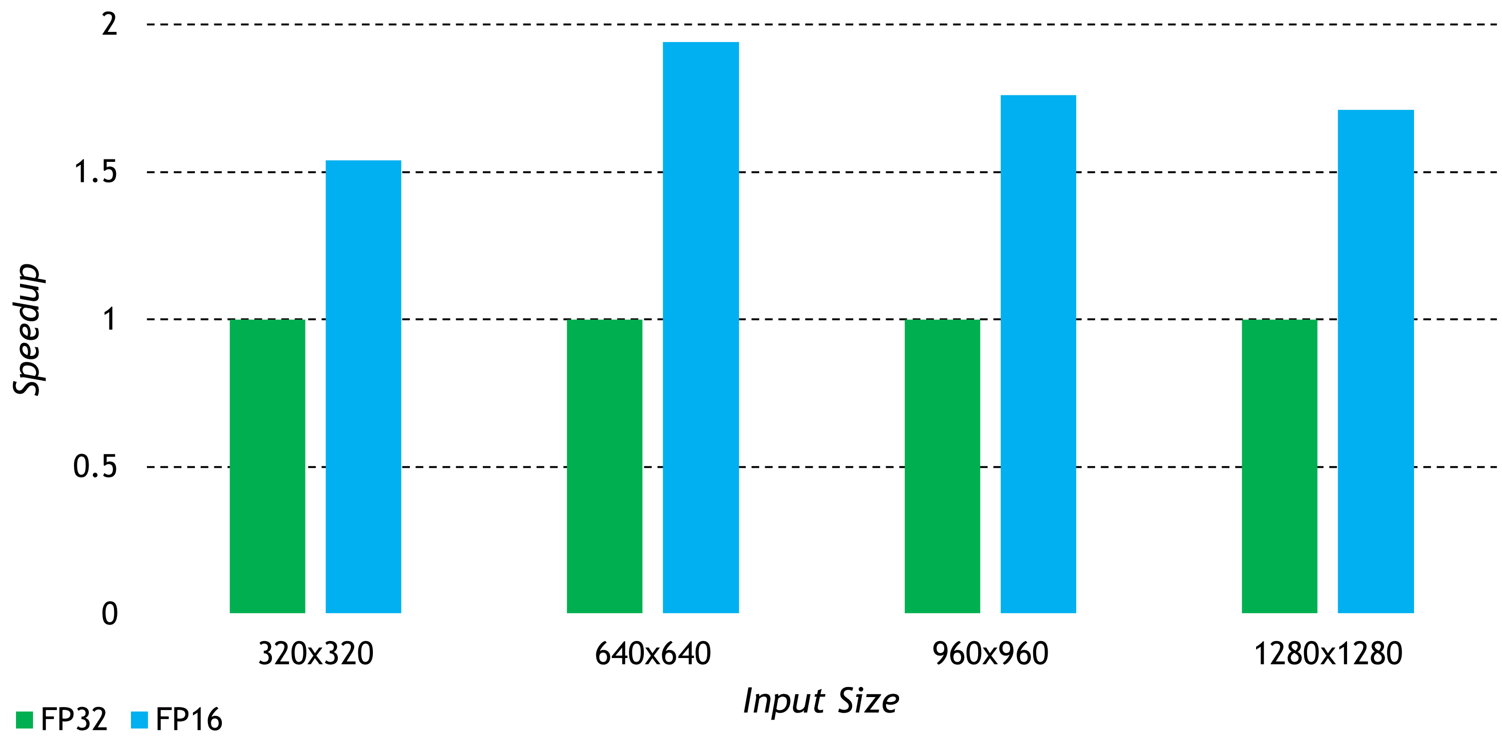
On Pascal and below, store in fp16 but process in fp32 – PSEUDO_HALF_CONFIG

Given an FP32 model, simply converting the weights to FP16 often retains decent quality

For best results ⇒ Retrain with FP16 precision

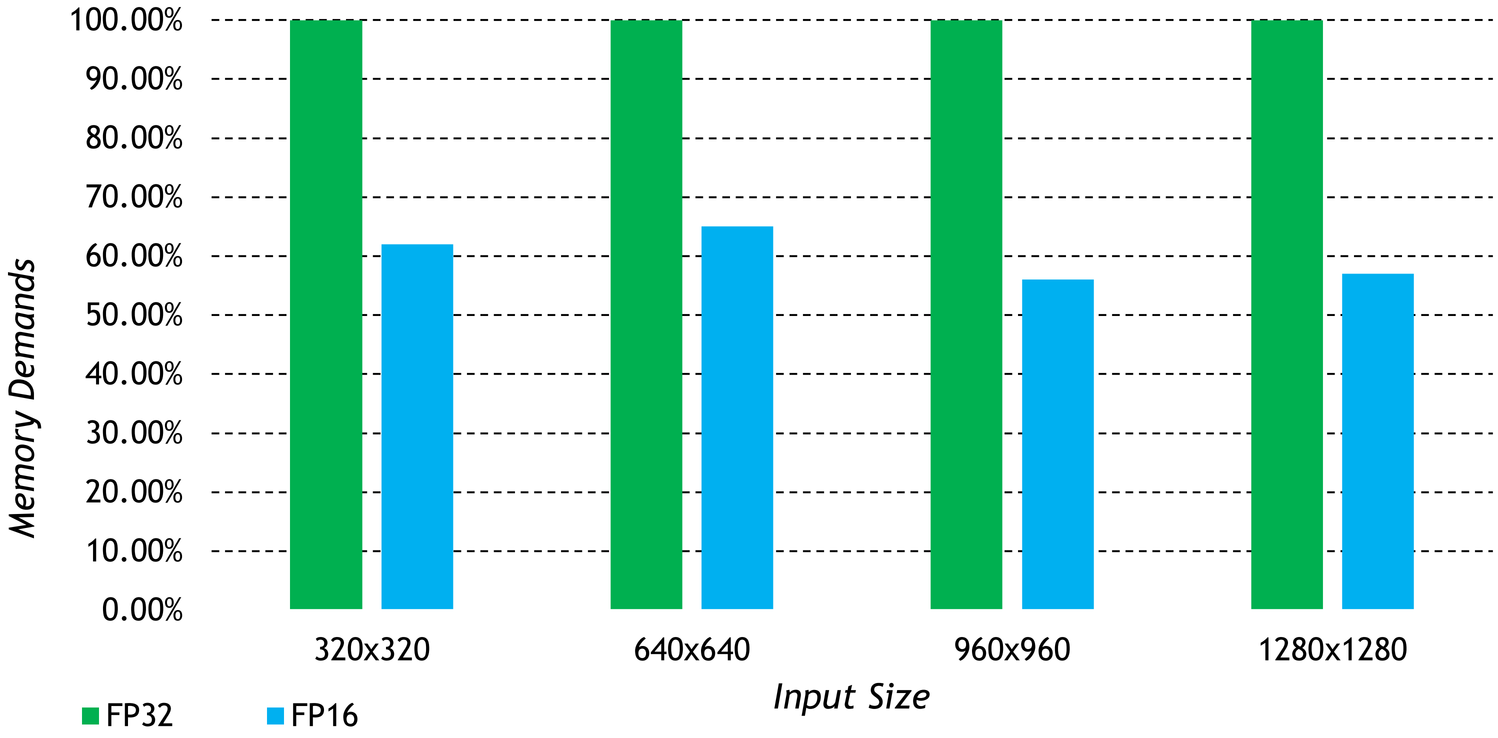
LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC‘18



LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC‘18



TENSOR CORES ON VOLTA & TURING

Tensor Cores perform FP16 matrix multiply accumulate (HMMA)

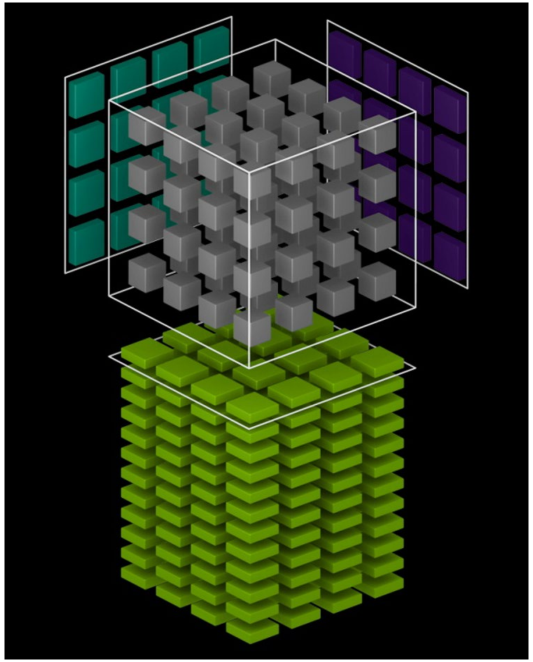
Turing also supports INT8 and INT4

Only two algorithms supported:

- CUDNN_CONVOLUTION_FWD_ALGO_WINOGRAD_NONFUSED
- CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM

Number of Input and output channels must be multiple of eight!

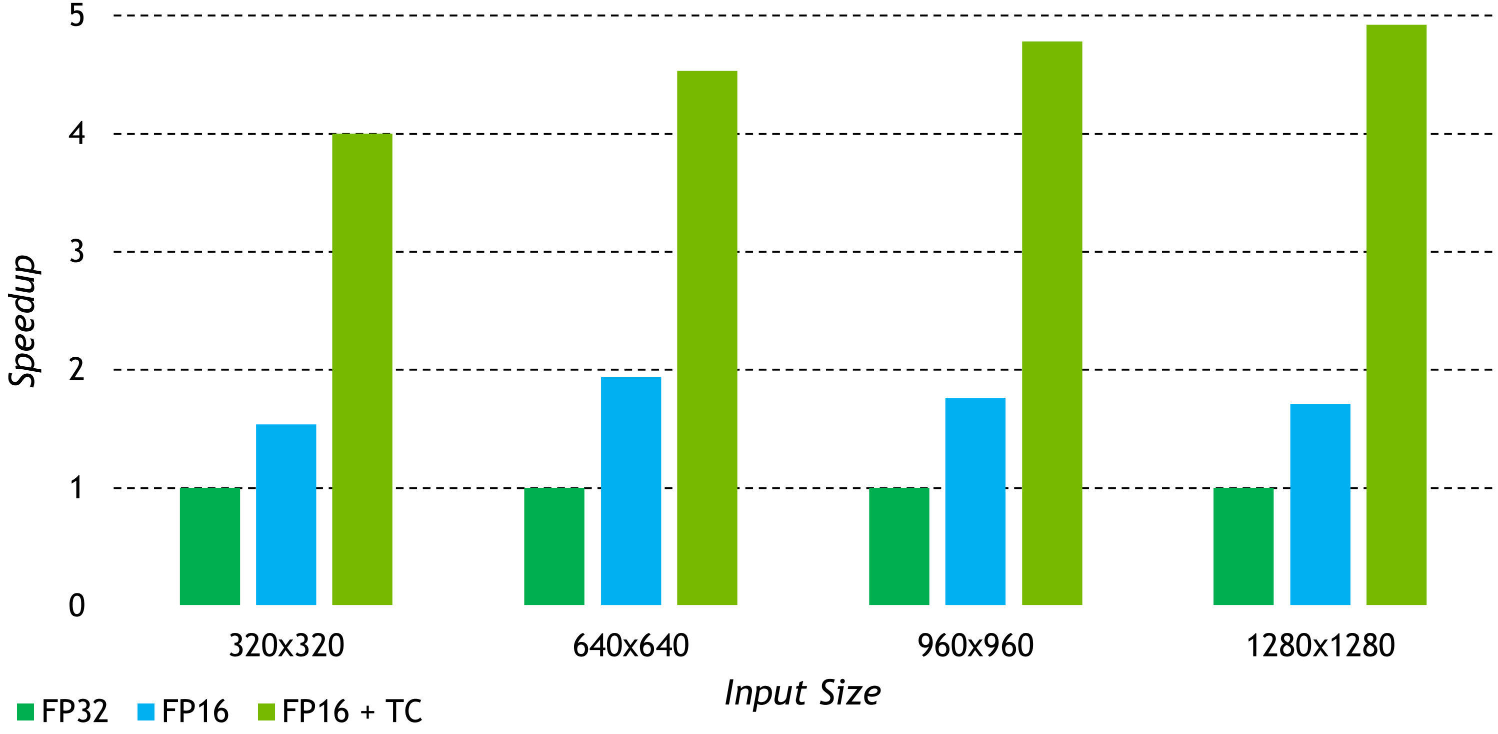
Convolution math type must be set to CUDNN_TENSOR_OP_MATH



And it will be much MUCH faster!

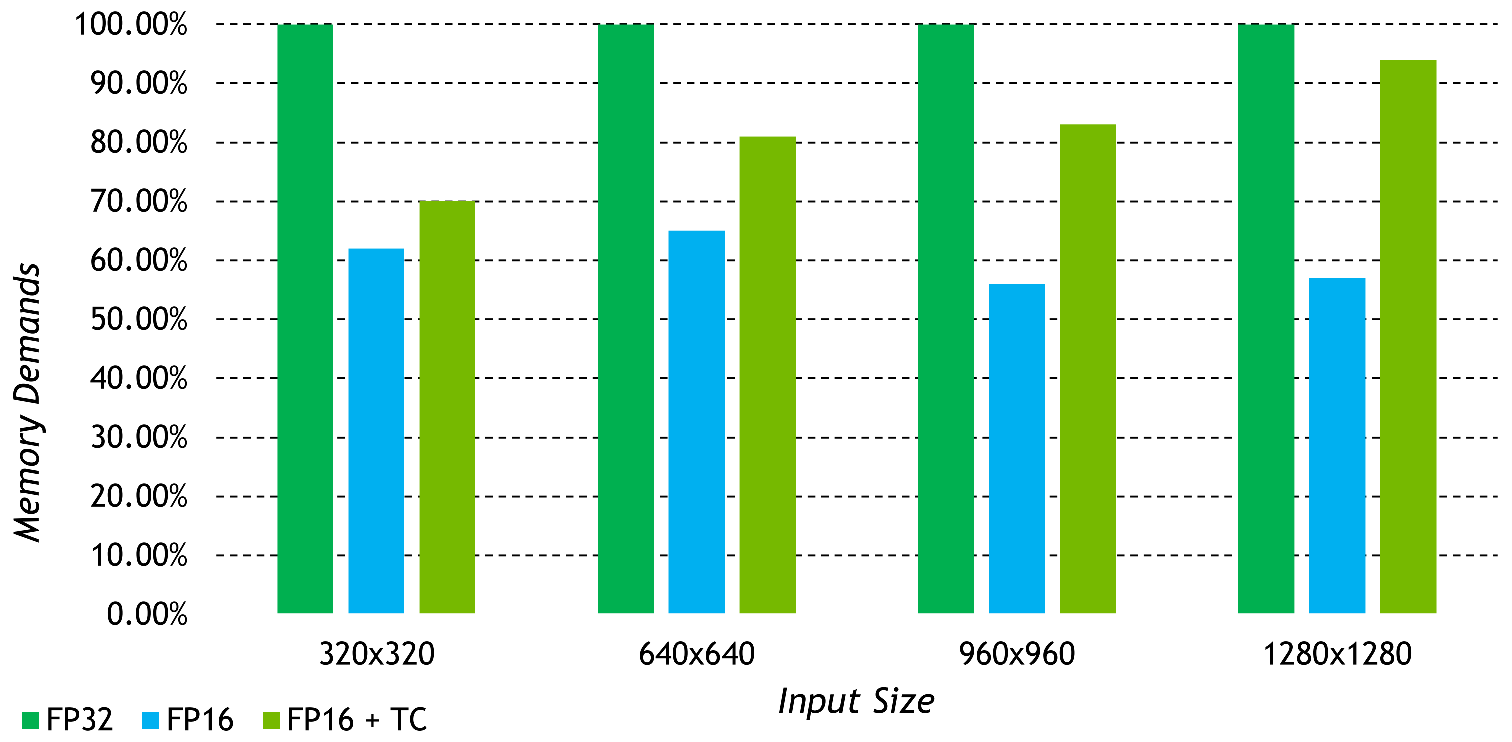
LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC'18



LOW PRECISION INFERENCE

Deep Image Matting, Chris Hebert, GTC'18



TENSOR CORES ON VOLTA AND TURING

NCHW vs NHWC

Input Tensor Size	Output Tensor Size	Filter Size	NCHW	NHWC
32 x 32 x 64	16 x 16 x 128	3 x 3	0.05 ms	0.04 ms
128 x 128 x 128	128 x 128 x 128	3 x 3	0.11 ms	0.08 ms
512 x 512 x 32	256 x 256 x 64	5 x 5	0.25 ms	0.15 ms
1920 x 1080 x 8	1920 x 1080 x 32	5 x 5	3.00 ms	2.31 ms
16 x 16 x 128	8 x 8 x 256	7 x 7	0.26 ms	0.23 ms
128 x 128 x 128	128 x 128 x 128	7 x 7	0.37 ms	0.34 ms
800 x 800 x 8	400 x 400 x 8	9 x 9	1.20 ms	2.53 ms

Convolution algorithm selected using cudnnFindConvolutionForwardAlgorithm(...)

CHANNEL PADDING

Some tensors might don‘t have a channel count that is a multiple of eight, e.g., three-channel RGB input tensors

- Cannot use Tensor Core acceleration

512x512x**3** → 512x512x32 convolution, 3x3 kernel, FP16, NHWC, GV100: **0.84 ms**

512x512x**8** → 512x512x32 convolution, 3x3 kernel, FP16, NHWC, GV100: **0.18 ms**

- Pad input tensor, zero-pad filter weights

CHANNEL FOLDING

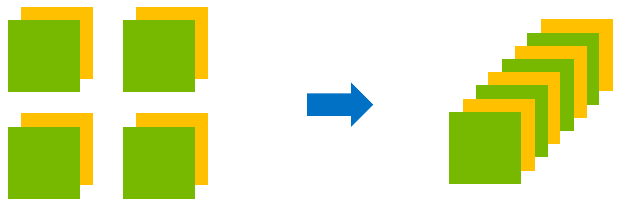
Tensor Core convolution performance varies with channel count

Both of these 3x3 convolutions uses input and output tensors that have the same size:

2048 x 2048 x 8 → 2048 x 2048 x 8 ⇒ **2.01 ms** **24 MB**

1024 x 1024 x 32 → 1024 x 1024 x 32 ⇒ **0.6 ms** **6 MB**

- Fold 2x2xN slices into 1x1x4N slices



Increases receptive field

Requires re-training

AGENDA

Introduction to cuDNN

cuDNN Best Practices:

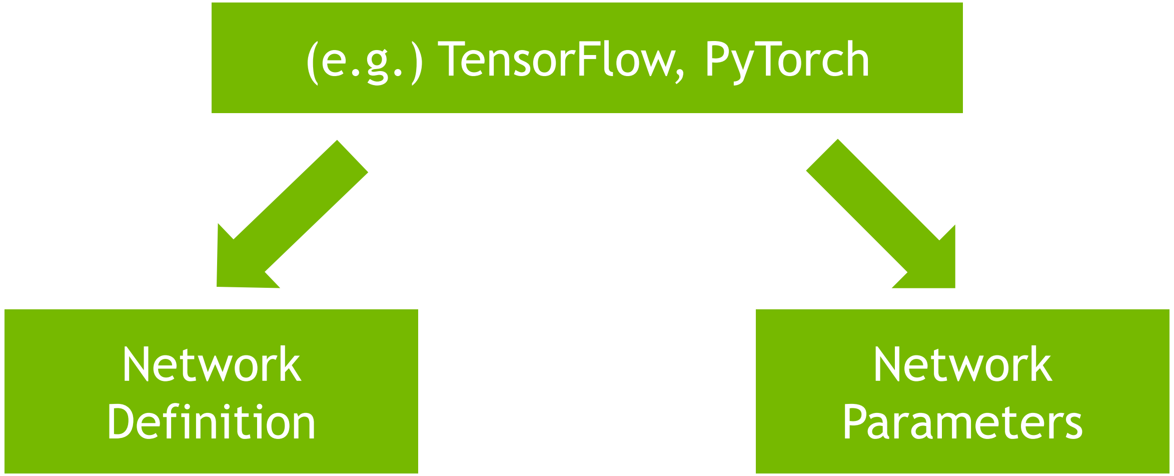
- Memory Management Done Right
- Choosing the Right Convolution Algorithm & Tensor Layout
- Tensor Cores: Low Precision Inference at Speed of Light

From Research to Production: It just works ... or not?!

Summary

Research To Production

What do we need?



Research To Production

What do we need?

(e.g.) TensorFlow, PyTorch

Network Definition

Network Parameters

A few ways to do this.

Research To Production

C++ Parse The Protocol Buffers

- Most checkpoint/model file formats based on Protocol Buffers from Google
  - Check em out, they're awesome.
- Message format defined in the .proto file
- Compiled with the Protocol Buffer Compiler
- Manipulate the contents with the Protocol Buffer API
- Good tutorials for this at
  - https://developers.google.com/protocol-buffers/docs/cpptutorial

Research To Production

Write the params and arch straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

```
……
head_0_spade_0_mlp_shared_0,Weights,  176198144,2342400,     128,183,5,5
head_0_spade_0_mlp_shared_0,biases,   176197632,512,         1,1,1,128
……
```

Tensor Name

Offset,Size

Shape

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

```
input_file_path = <Path to Pytorch checkpoint>

ckpnt = torch.load(input_file_path,map_location="cpu")

out_path = “netG_params.txt".format(item_key)

        with open(out_path,"w") as f:
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Iterate the model, find the weights and biases

```
input_file_path = <Path to Pytorch checkpoint>

ckpnt = torch.load(input_file_path,map_location="cpu")

out_path = “netG_params.txt".format(item_key)

with open(out_path,"w") as f:

          for model_key in ckpnt :
                if model_key.find("weight") > 0 or model_key.find("bias") > 0:
                    cur_var = Variable(ckpnt[model_key])
                    var_size = cur_var.size()
                    size_len = len(var_size)
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Replace ‘.’ with ‘_’ (personal preference)

```
tensor_name = model_key.replace(".","_")
tensor_type = “”
if model_key.find("weight") > 0:
          tensor_type = "Weights"
          tensor_name = tensor_name.replace("_weight","")
if model_key.find("bias") > 0:
          tensor_type = "biases"
          tensor_name = tensor_name.replace("_bias","")

tensor_dims = ""
tensor_total_size = 1
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Record the tensor shape in a consistent manner

```
 if size_len < 4:
         size_len_delta = 4-size_len
         for s in range(size_len_delta):
                  tensor_dims += ",1"

         for s in range(size_len):
                  tensor_dims += ",{}".format(var_size[s])
                  tensor_total_size *= var_size[s]

tensor_total_file_size = tensor_total_size * 4

tensor_size_data = ",{},{}".format(tensor_offset,tensor_total_file_size)
```

Research To Production

Write the model architecture straight from Python

PyTorch example (simplified)

Write the name, offset, size and shape to the text file.

```
tensor_total_file_size = tensor_total_size * 4

tensor_size_data = ",{},{}".format(tensor_offset,tensor_total_file_size)
tensor_offset += tensor_total_file_size
tensor_name += ",{}".format(tensor_type)
f.write(tensor_name)
f.write(tensor_size_data)
f.write(tensor_dims)
f.write("\n")
```

Research To Production

Write the params and arch straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

```
……
head_0_spade_0_mlp_shared_0,Weights,  176198144,2342400,     128,183,5,5
head_0_spade_0_mlp_shared_0,biases,   176197632,512,         1,1,1,128
……
```

Tensor Name

Offset,Size

Shape

Research To Production

Write the params and arch straight from Python

PyTorch example (simplified)

Load the model in Python, open output file for writing

out,in,filter H/W

```
……
head_0_spade_0_mlp_shared_0,Weights,  176198144,2342400,     128,183,5,5
head_0_spade_0_mlp_shared_0,biases,   176197632,512,         1,1,1,128
……
```

Tensor Name

Offset,Size

Shape

Research To Production

Write the model params straight from Python

PyTorch example (simplified)

Similar loop as before but extract the weights from the .data member and write to a single file

```
(iterate model as before)

data = ckpt[model_key]
cur_var = Variable(data)
var_size = cur_var.size()
np_data = data.cpu().numpy()
f.write(np_data.tobytes())
```

Research To Production

Scientist vs Engineer

- Scientists express their models in an algebraically correct manner.
  - That’s their job.
  - But algebraically correct does not necessarily mean performant.
- Engineers need to identify when an algorithm can be restructured for performance.
  - That’s our job.

Research To Production

Scientist vs Engineer

Example 1. Wx+b when you ONLY want the bias.

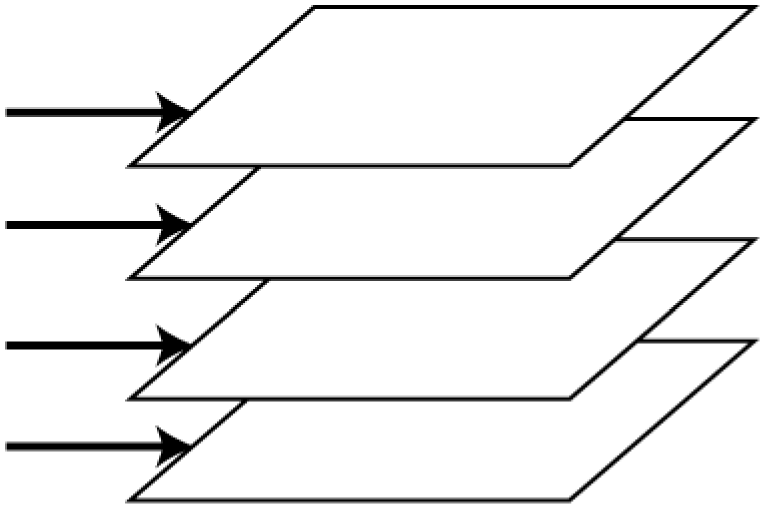
$$\begin{bmatrix} 0.1762 & 0.0365 & -0.0102 & 0.9191 \\ -0.2388 & -0.0010 & 0.7723 & -0.5400 \\ 0.0012 & -0.3333 & -0.0001 & 0.0838 \\ 0.0019 & -0.0095 & 0.0200 & 0.0211 \end{bmatrix} \times \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} + \begin{bmatrix} 0.1762 \\ -0.2388 \\ 0.0012 \\ 0.0019 \end{bmatrix}$$

Research To Production

Scientist vs Engineer

Example 1. Wx+b when you ONLY want the bias.

$$\begin{bmatrix} 0.1762 & 0.0365 & -0.0102 & 0.9191 \\ -0.2388 & -0.0010 & 0.7723 & -0.5400 \\ 0.0012 & -0.3333 & -0.0001 & 0.0838 \\ 0.0019 & -0.0095 & 0.0200 & 0.0211 \end{bmatrix} \times \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} + \begin{bmatrix} 0.1762 \\ -0.2388 \\ 0.0012 \\ 0.0019 \end{bmatrix}$$



Research To Production

Scientist vs Engineer

Example 1. Wx+b when you ONLY want the bias.

In this particular case:

Write custom kernel to write the bias values.

And if possible fuse with previous and/or next step.

Research To Production

Scientist vs Engineer

Example 2. Downsample called many times on the same data.

Layer Block (e.g. Resnet)

- Sub Layer 0 – (e.g. Normalization)
- Sub Layer 1 –
- Sub Layer S – (ResNet Shortcut)

Sub Layer

- Downsample original Input
- Do Layer Specific Stuff

Research To Production

Scientist vs Engineer

This is the same
operation on the
same data 3 times

Example 2. Downsample called many times on the same data.

Layer Block (e.g. Resnet)

- Sub Layer 0 – (e.g. Normalization)
- Sub Layer 1 –
- Sub Layer S – (ResNet Shortcut)

Sub Layer

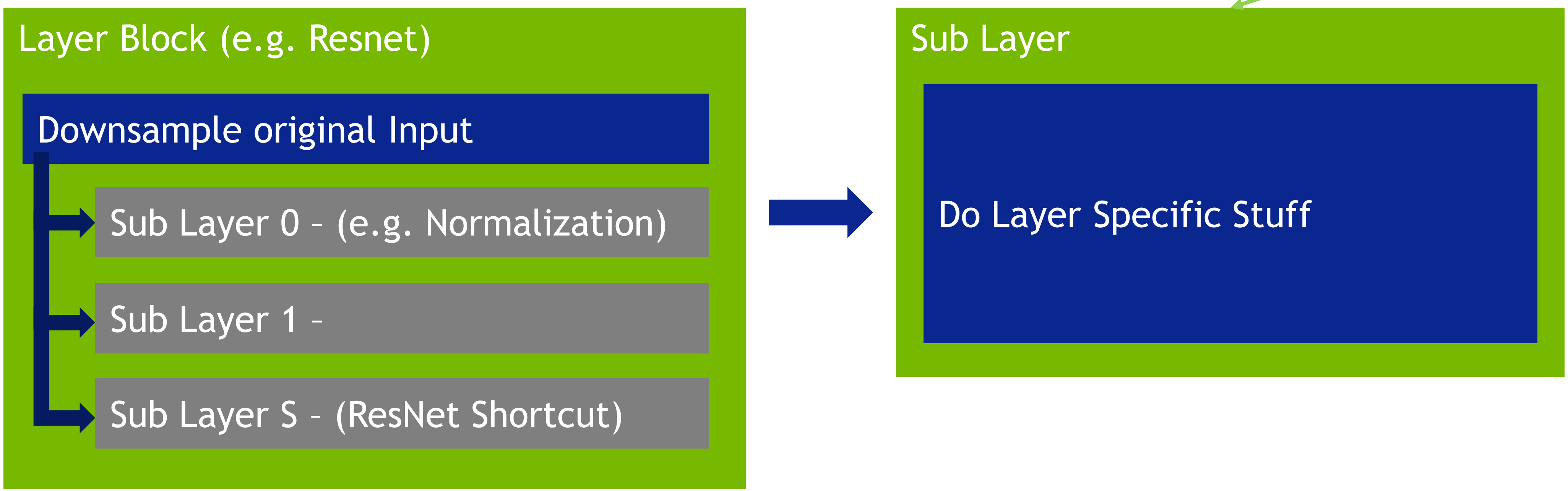
- Downsample original Input
- Do Layer Specific Stuff

Research To Production

Scientist vs Engineer

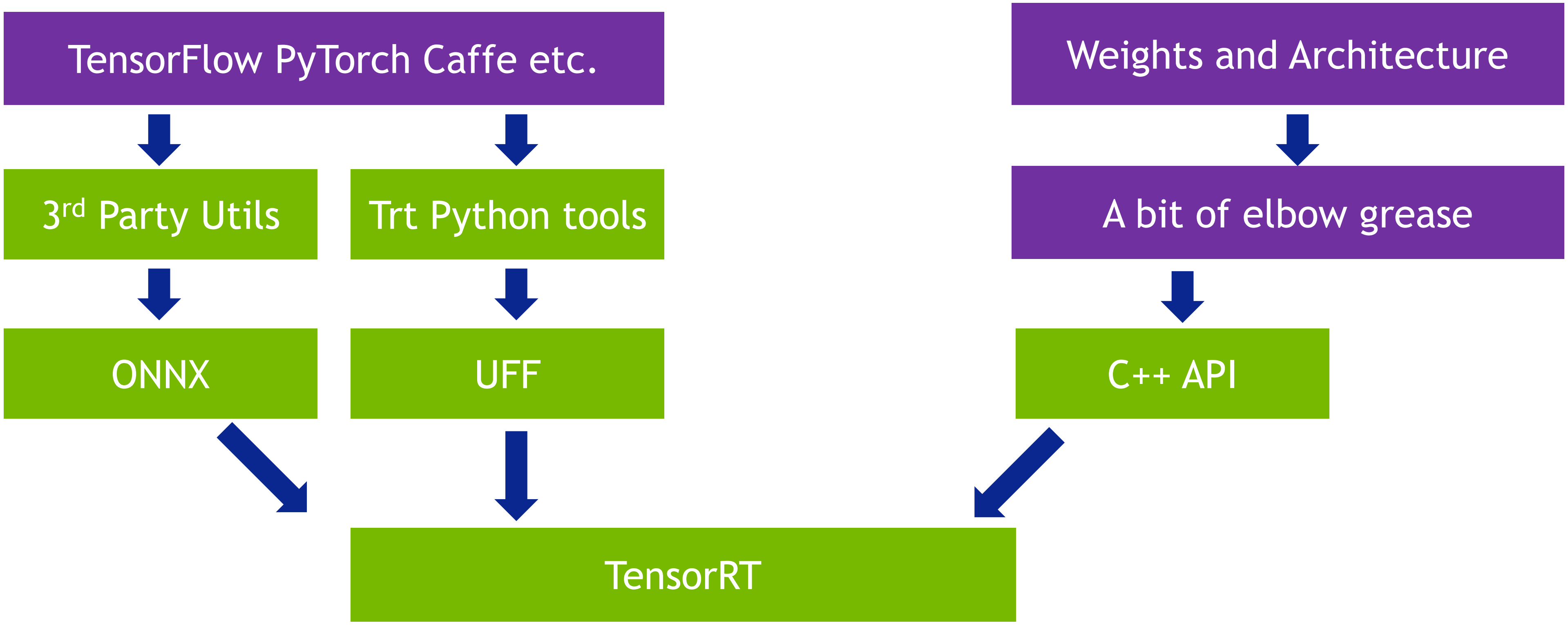
Example 2. Re order the operations….

This is the same operation on the same data 3 times



Research To Production

Porting to TensorRT



Research To Production

UFF, ONNX or API …. Which to use….

- Most common architectures will import directly from TensorFlow/PyTorch etc
- Most common operations are already supported in TensorRT
- Convolution/Cross Correlation
- Activation
  - Sigmoid, Relu, Clipped Relu, TanH, ELU
- Batch Norm
  - Spatial, Spatial_persistent, Per Activation
- Pooling
  - Max, Average

Research To Production

UFF, ONNX or API …. Which to use….

- Sometimes it's not that easy
- Sometimes some graph surgery is required.
  - Edit the graph to strip out e.g. pre/post processing at either end of the graph
- TensorRT provides a plug-in interface for custom layers
  - Name custom layers as per the incoming model (e.g. LeakyRelu)
  - From TrT 5.1 : The IPluginV2 interface supports optimization.
- There is a simpler option

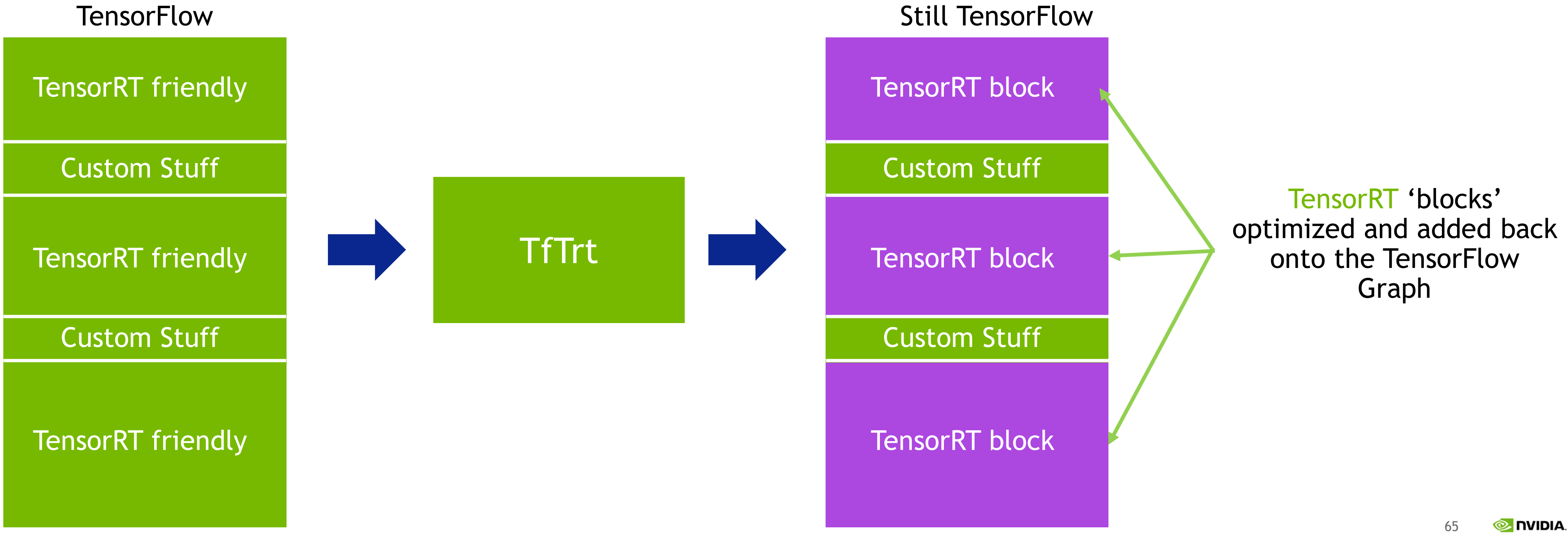
Research To Production

Porting to TensorRT Using TfTrt

- Converts TensorFlow graph into 1 or more TensorRT ‘blocks’
- Adds these blocks back onto TensorFlow graph
- Inference of these blocks performed with TensorRT
- The rest use TensorFlow
- Workflow:
  - Load TensorFlow graph
  - Prepare for inference (freeze layers, convert variables to constants etc)
  - Call trt.create_inference_graph(input_graph_def, outputs, max_batch_size,max_workspace_size,precision_mode)

Research To Production

Porting to TensorRT Using TfTrt



Research To Production

Porting ‘Funky’ networks to TensorRT

Important takeaways from this

You don’t need generate a single monolithic graph with TensorRT

Generate graph snippets from TensorRT interleaved with custom CUDA

You can do this with the TensorRT API

Run them in whatever sequence you need at run time.

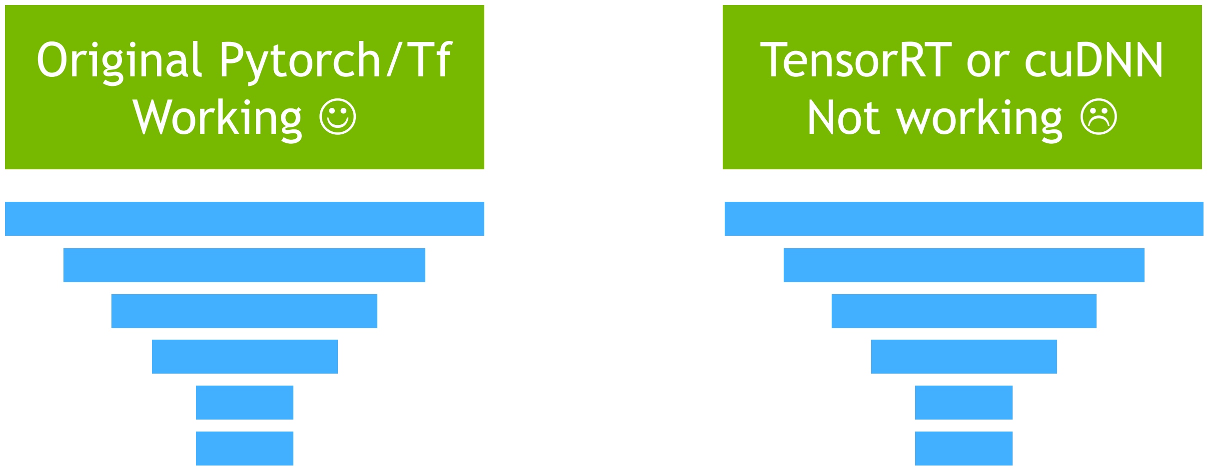
Allows you to create inference solutions with dynamic runtime behavior.

Keep all data on the GPU whenever possible.

Research To Production

When it doesn't ……. Just work.

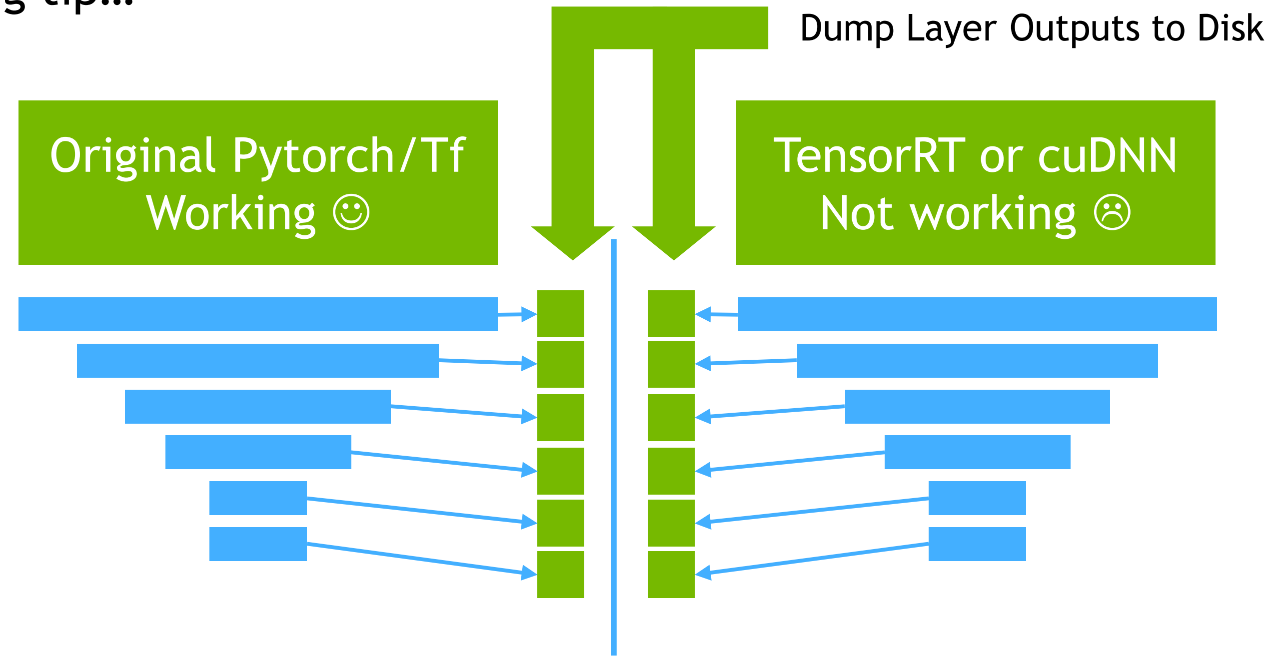
Here's a debugging tip…



Research To Production

When it doesn't ……. Just work.

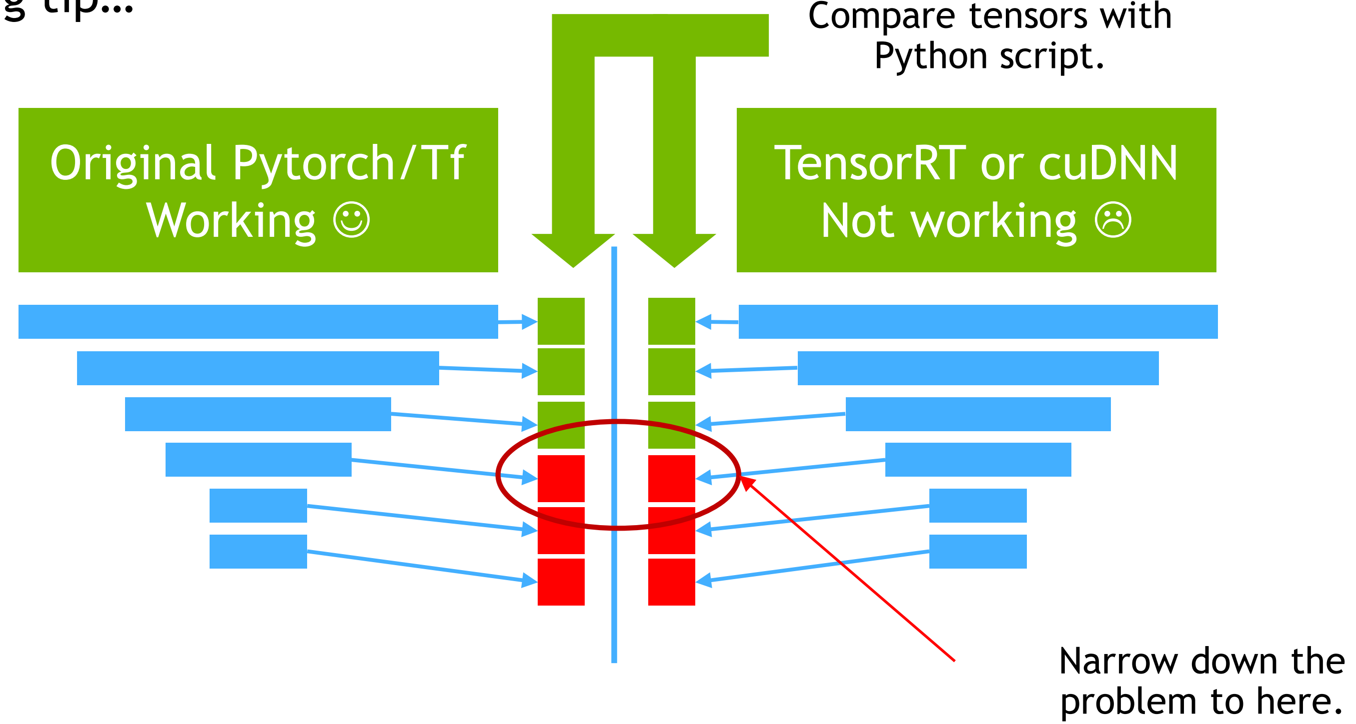
Here's a debugging tip…



Research To Production

When it doesn't .... Just work.

Here's a debugging tip…



AGENDA

Introduction to cuDNN

cuDNN Best Practices:

- Memory Management Done Right
- Choosing the Right Convolution Algorithm & Tensor Layout
- Tensor Cores: Low Precision Inference at Speed of Light

From Research to Production: It just works ... or not?!

Summary

SUMMARY

Common DL frameworks often far from optimized for inference on GPUs

- Use cuDNN (or TensorRT) if you care about performance & memory!

Memory Management matters!

Lower your precision if possible!

Use hardware-specific optimizations, e.g. Tensor Cores on Volta & Turing!

You can never profile too much!

Download, documentation, discussion board, …

www.developer.nvidia.com/cudnn

NVIDIA®