INTEGRATION OF DALI WITH TENSORRT ON XAVIER

Josh Park (joshp@nvidia.com), Manager - Automotive Deep Learning Solutions Architect at NVIDIA
Anurag Dixit(anuragd@nvidia.com), Deep Learning SW Engineer at NVIDIA

Contents

Backgrounds

Backgrounds

Massive amount of computation in DNN

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

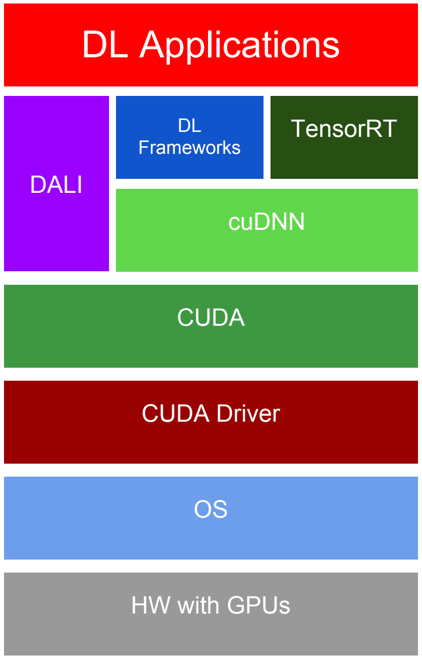
Parameter layers in billions FLOPS (mul/add)

[1] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

GPU: High Performance Computing Platform

	Tesla V100 PCle	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.5 TFLOPS
Single-Precision Performance	14 TFLOPS	15 TFLOPS
Tensor Performance	112 TFLOPS	120 TFLOPS
GPU Memory	16 GB HBM2	
Memory Bandwidth	900 GB/sec	
ECC	Yes	
Interconnect Bandwidth*	32 GB/sec	300 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

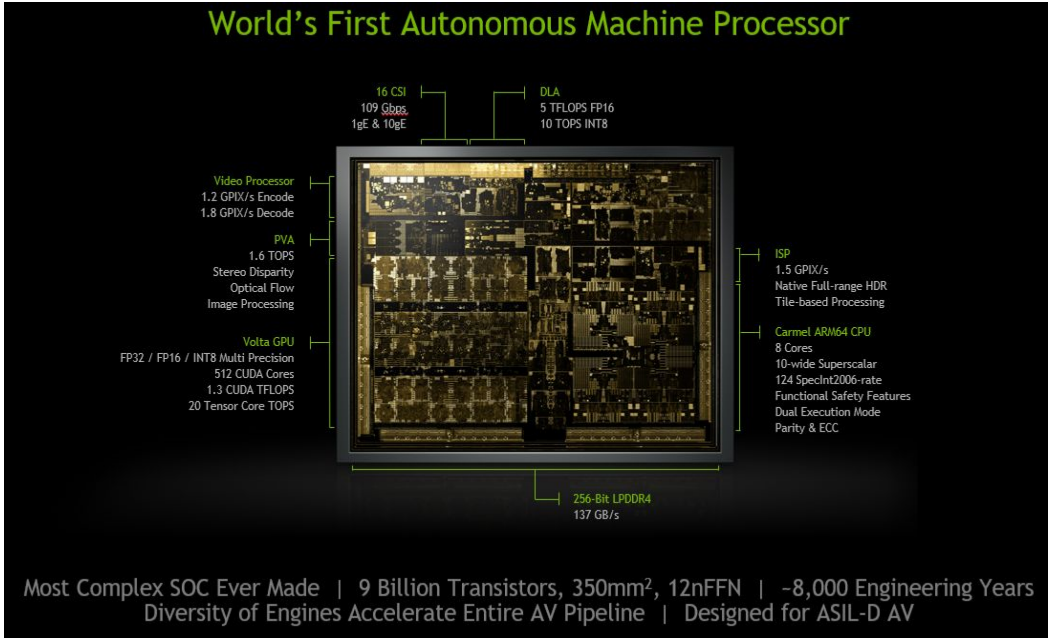
SW Libraries



NVIDIA DRIVE AGX Platform

Xavier - aarch64 based on SoC w/ CPU + GPU + MEM

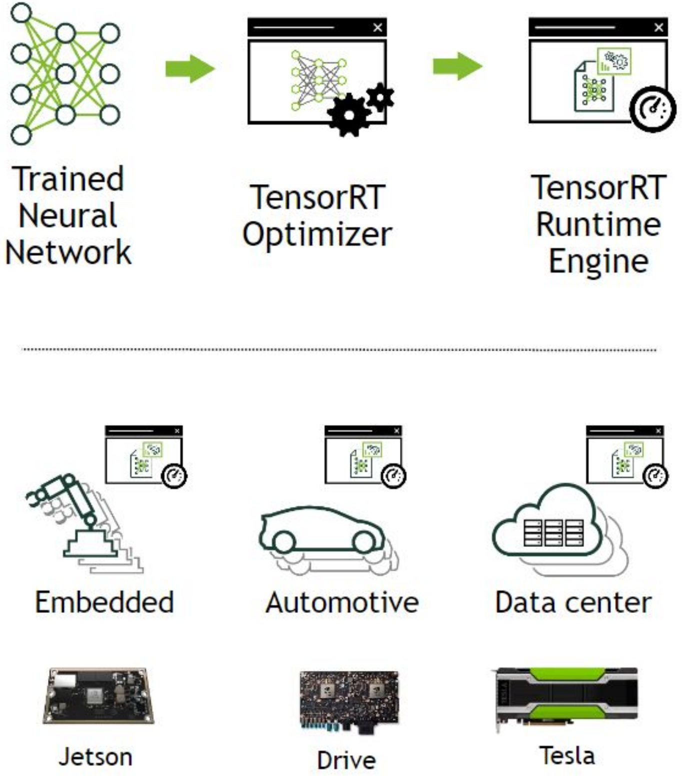
- **iGPU**
- **8 Volta SMs**
- **512 CUDA cores**
- **64 Tensor Cores**
- **20 TOPS INT8, 10 TOPS FP16**
- **CUDA Compute Capability 7.2**



NVIDIA TensorRT

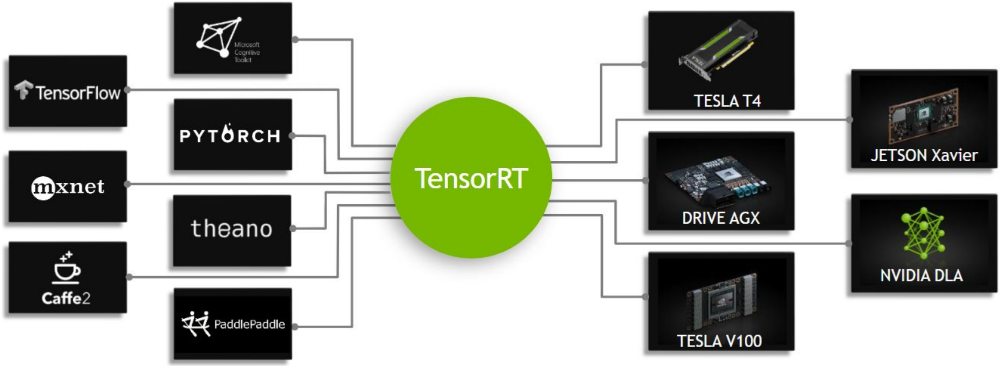
NVIDIA TensorRT - Programmable Inference Accelerator

- Optimize and Deploy neural networks in production environments
- Maximize throughput for latency critical apps with optimizer and runtime
- Deploy responsive and memory efficient apps with INT8 & FP16 optimizations
- Accelerate every framework with TensorFlow integration and ONNX support
- Run multiple models on a node with containerized inference server

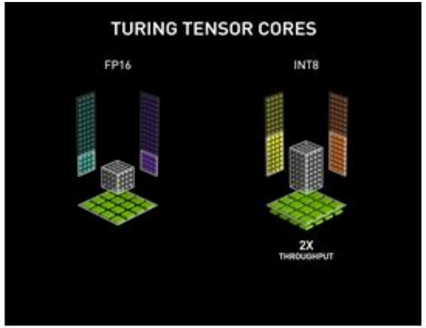


TensorRT 5 supports Turing GPUs

- Optimized kernels for mixed precision (FP32, FP16, INT8) workloads on Turing GPUs
- Control precision per-layer with new APIs
- Optimizations for depth-wise convolution operation



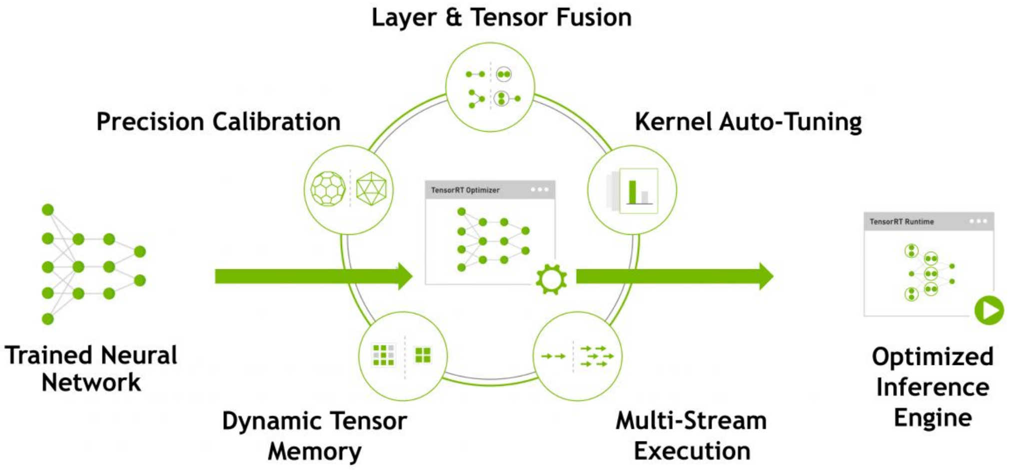
From Every Framework, Optimized For Each Target Platform



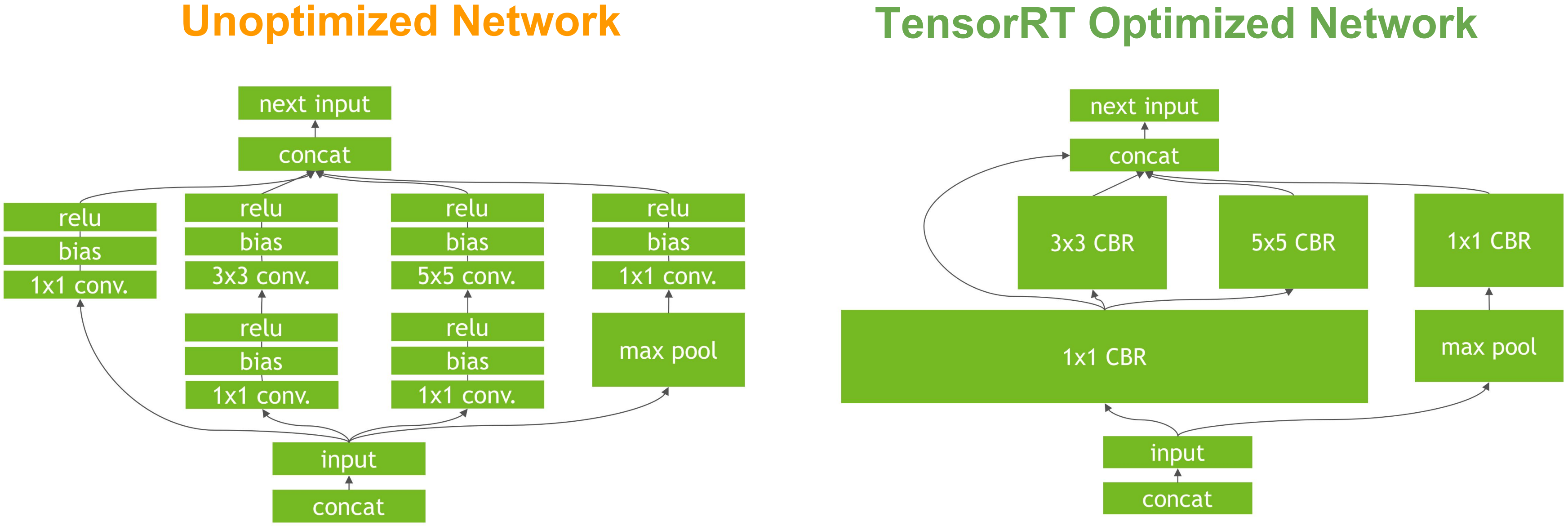
Turing Tensor Core

How TensorRT Works?

- Layer & Tensor Fusion
- Auto-Tuning
- Precision Calibration
- Multi-Stream Execution
- Dynamic Tensor Memory



Layer & Tensor Fusion



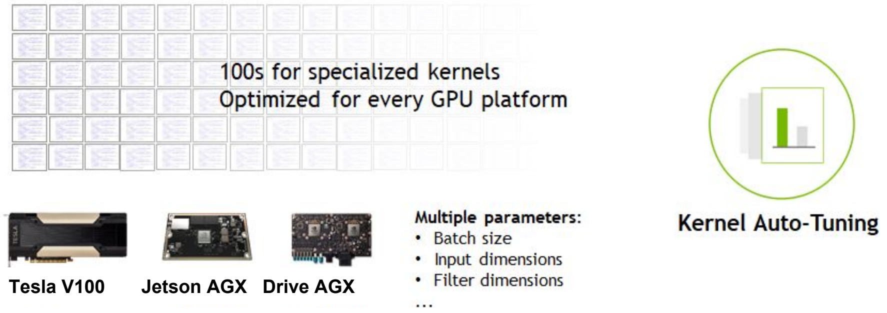
e.g

Networks	Number of layers (Before)	Number of layers (After)
VGG19	43	27
Inception v3	309	113
ResNet-152	670	159

Kernel Auto-Tuning

- Maximize kernel performance
- Select the best performance

  for target GPU
- Parameters
  - Input data size
  - Batch
  - Tensor layout
  - Input dimension
  - Memory
  - Etc.




```
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Relu(14)
[V] [TRT] Tactic 5144900180810042977 time 0.209112
[V] [TRT] Tactic 7995089803833173579 time 0.147572
[V] [TRT] Tactic -9018593586369639207 time 0.229336
[V] [TRT] Tactic -8109134417363526151 time 0.210388
[V] [TRT] Tactic -3998689942157953075 time 0.228312
[V] [TRT] Tactic -3448915218022119398 time 0.14638
[V] [TRT] Tactic -3120440365138571960 time 0.147588
[V] [TRT] Tactic -242550410056126076 time 0.228548
[V] [TRT]
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Relu(1)
[V] [TRT] Tactic 0 time 0.547884
[V] [TRT] Tactic 1 time 0.424728
[V] [TRT] Tactic 2 scratch requested: 36864000, available: 16777216
[V] [TRT] Tactic 5 scratch requested: 131045440, available: 16777216
[V] [TRT]
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Relu(33)
[V] [TRT] --------------- Chose 14 (-3448915218022119398)
[V] [TRT]
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv2/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv2/Relu(3)
[V] [TRT]
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv2/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv2/Relu(2)
[V] [TRT] Tactic 7 time 1.13859
[V] [TRT] Tactic 10 time 1.10559
[V] [TRT] Tactic 14 time 1.075
[V] [TRT] Tactic 15 time 0.989992
[V] [TRT] Tactic 25 time 0.970556
[V] [TRT] Tactic 26 time 1.2982
[V] [TRT] Tactic 29 time 1.28358
[V] [TRT] Tactic 30 time 1.21168
[V] [TRT] Tactic 33 time 1.05864
```

Lower precision - FP16

- FP16 matches the results quite closely to FP32
- TensorRT automatically converts FP32 weights to FP16 weights

```
builder->setFp16Mode(true);
```

- To enforce that 16-bit kernels will be used when building the engine

```
builder->setStrictTypeConstraints(true);
```

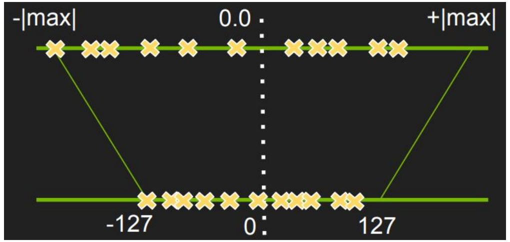
- Tensor Core kernels (HMMA) for FP16 (supported on Volta and Turing GPUs)

Lower Precision - INT8 Quantization

- Setting the builder flag enables INT8 precision inference.
  - `builder->setInt8Mode(true);`
  - `IInt8Calibrator* calibrator;`
  - `builder->setInt8Calibrator(calibrator);`

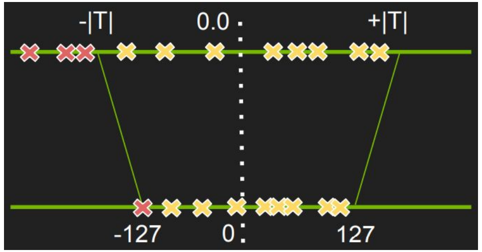
	Dynamic Range	Mininum Positive Value
FP32	$-3.4\times10^{38} \sim +3.4\times10^{38}$	1.4×10^{-45}
FP16	$-65504 \sim +65504$	5.96×10^{-8}
INT8	$-128 \sim +127$	1

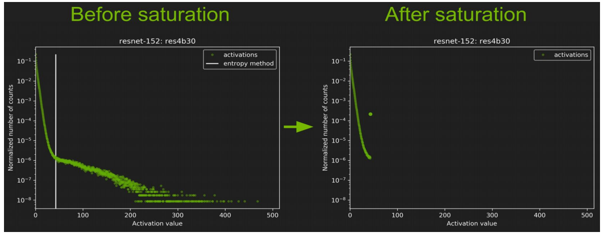
- Quantization of FP32 weights and activation tensors
  - (**weights**) Int8_weight = ROUND_To_Nearest (scaling_factor * FP32_weight_in_the_filters)
    - * scaling_factor = 127.0 f / max (| all_FP32_weights |)
  - (**activation**) Int8_value = if (value > threshold): threshold; else scaling_factor * FP32_value
    - * Activation range unknown (input dependent) => calibration is needed
- Dynamic range of each activation tensor => the appropriate quantization scale
- TensorRT: symmetric quantization with quantization scale calculated using absolute maximum dynamic range values
- Control precision per-layer with new APIs
- Tensor Core kernel (IMMA) for INT8 (supported on Drive AGX Xavier iGPU and Turing GPUs)



Lower Precision - INT8 Calibration

- Calibration Solutions in TensorRT
    - Run FP32 inference on Calibration
    - Per Layer:
        - Histograms of activations
        - Quantized distributions with different saturation thresholds.
    - Two ways to set saturation thresholds (dynamic ranges) :
        - manually set the dynamic range for each network tensor using setDynamicRange API
            - * Currently, only symmetric ranges are supported
        - use INT8 calibration to generate per tensor dynamic range using the calibration dataset (*i.e. 'representative' dataset)*
            - *pick threshold which minimizes KL_divergence (entropy method)





** INT8 and FP16 mode, both if the platform supports. TensorRT will choose the most performance optimal kernel to perform inference.*

Plugin for Custom OPs in TensorRT 5

- Custom op/layer: op/layer not supported by TensorRT => need to implement plugin for TensorRT engine
- Plugin Registry
  - stores a pointer to all the registered Plugin Creators / look up a specific Plugin Creator
  - Built-in plugins: `RPROI_TRT`, `Normalize_TRT`, `PriorBox_TRT`, `GridAnchor_TRT`, `NMS_TRT`, `LReLU_TRT`, `Reorg_TRT`, `Region_TRT`, `Clip_TRT`
- Register a plugin by calling `REGISTER_TENSORRT_PLUGIN(pluginCreator)` which statically registers the Plugin Creator to the Plugin Registry

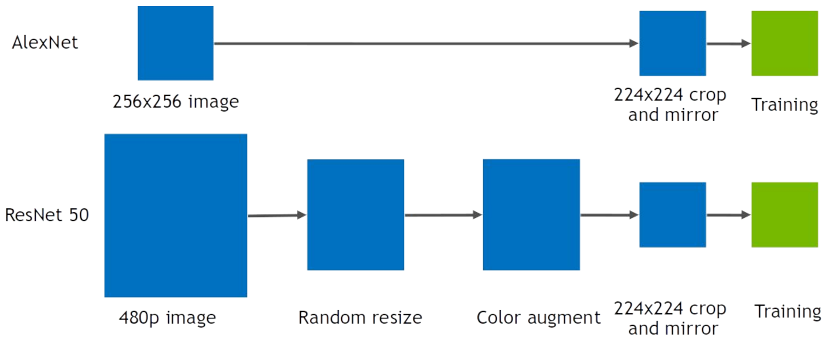
How can we further optimize end-to-end inference pipeline on NVIDIA DRIVE Xavier?

NVIDIA DALI

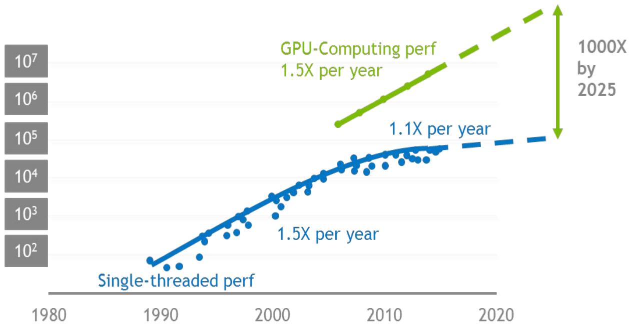
Motivation: CPU BOTTLENECK OF DL TRAINING

CPU ops and CPU to GPU ratio

- Operations are performed mainly on CPUs before the input data is ready for inference/training
- Half precision arithmetic, multi-GPU, dense systems are now common (e.g., DGX1V, DGX2)
- Can't easily scale CPU cores (expensive, technically challenging)
- Falling CPU to GPU ratio:
  - DGX1: 40 cores, 8 GPUs, 5 cores/ GPU
  - DGX2: 48 cores ↑, 16 GPUs ↑, 3 cores/ GPU ↓



Complexity of I/O pipeline



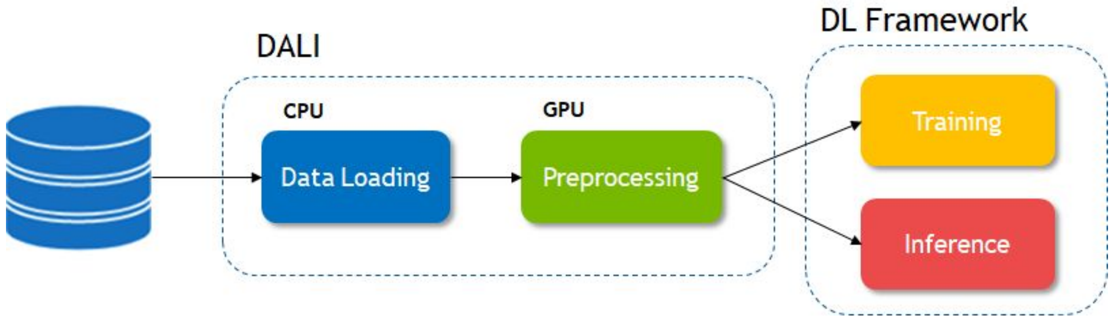
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten New plot and data collected for 2010-2015 by K. Rupp

NVIDIA

Data Loading Library (DALI)

High Performance Data Processing Library

“Originally on X86_64”



Why DALI?

- Running DNN models requires input data pre-processing
- Pre-processing involves
    - Decoding, Resize, Crop, Spatial augmentation, Format conversions (NCHW ⟺ NHWC)
- DALI supports
    - the feature to accelerate pre-processing on GPUs
    - configurable graphs and custom operators
    - multiple input formats (e.g. JPEG, LMDB, RecordIO, TFRecord)
    - serializing a whole graph (portable graph)
- Easily integrates with framework plugins and open source bindings

Integration: Our Effort on DALI

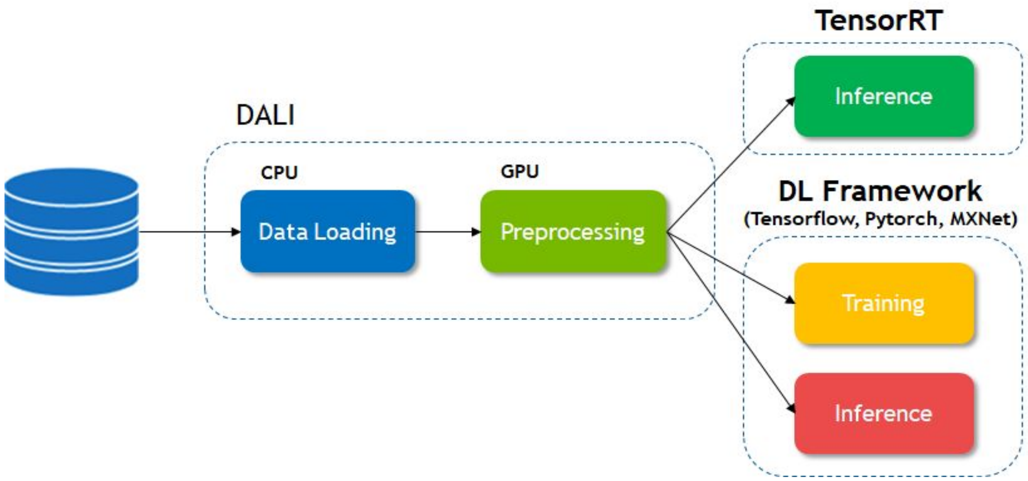
Extension to aarch64 and Inference engine

Beyond x86_64

- Extension of targeted platform to "**aarch64**": Drive AGX Platform

High level TensorRT runtime within DALI

- TensorRTInfer op via a plugin



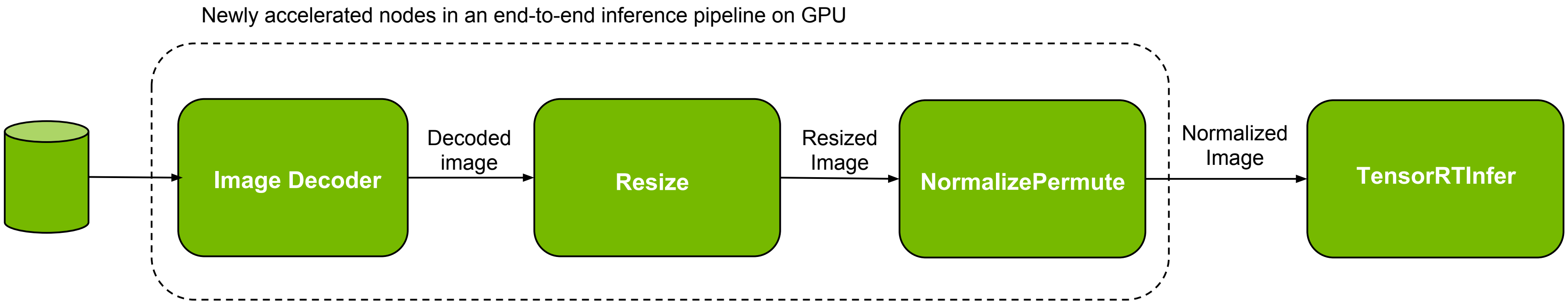
Dependency

Components	On x86_64	On aarch64
gcc	4.9.2 or later	5.4
Boost	1.66 or later	N/A
Nvidia CUDA	9.0 or later	10.0 or later
protobuf	version 2.0 or later	version 2.0
cmake	3.5 or later	3.5 later
libnvjpeg	Included in cuda toolkit	Included in cuda toolkit
opencv	version 3.4 (recommended) 2.x (unofficial)	version 3.4
TensorRT	5.0 / 5.1	5.0 / 5.1

How we Integrate TensorRT with DALI?

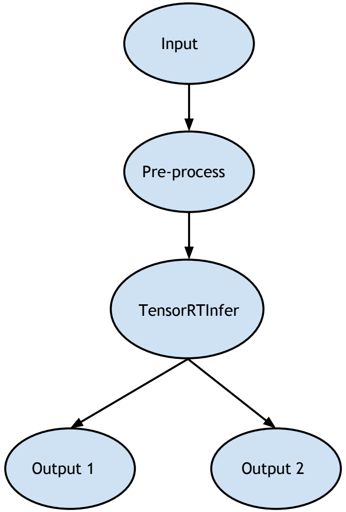
- DALI supports custom operator in C++
- Custom operator library can be loaded in the runtime
- TensorRT inference is treated as a custom operator
- TensorRT Infer schema
    - serialized engine
    - TensorRT plugins
    - input/output binding names
    - batch size for inference

Pipeline Example of TensorRT within DALI

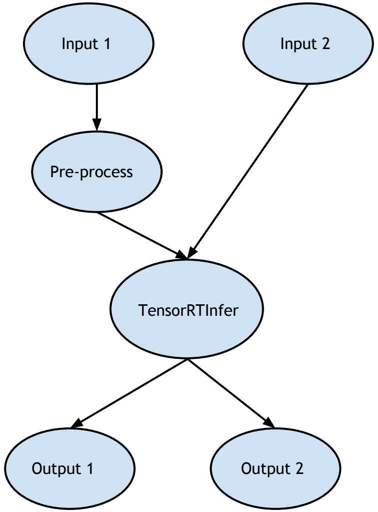


Use Cases

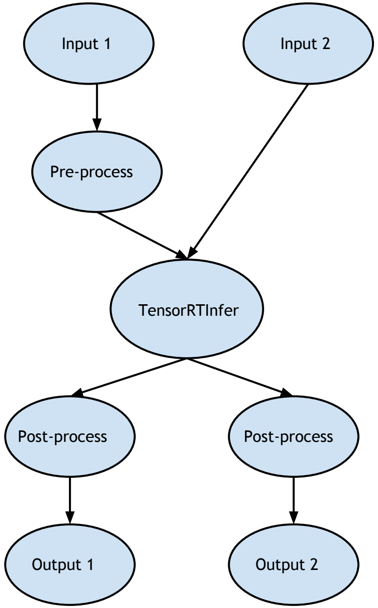
Single Input, Multi Outputs



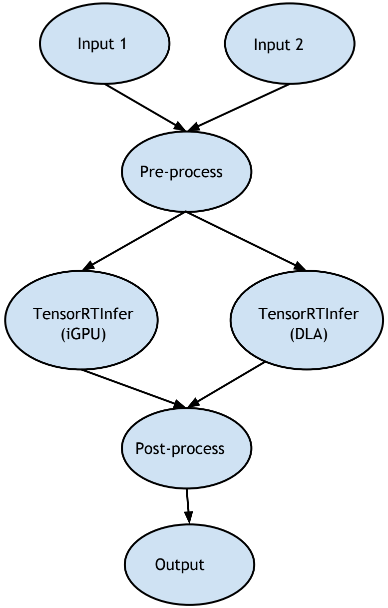
Multi Inputs, Multi Outputs



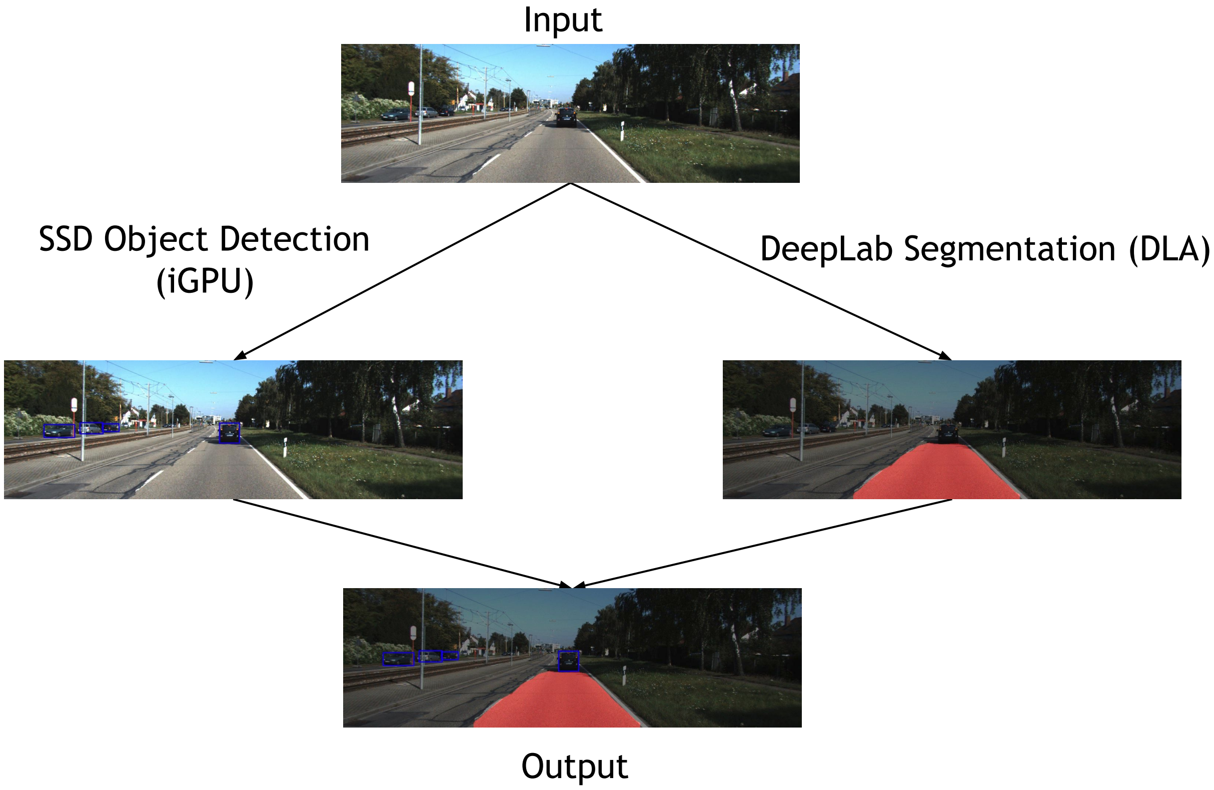
Multi Inputs, Multi Output with Post processing



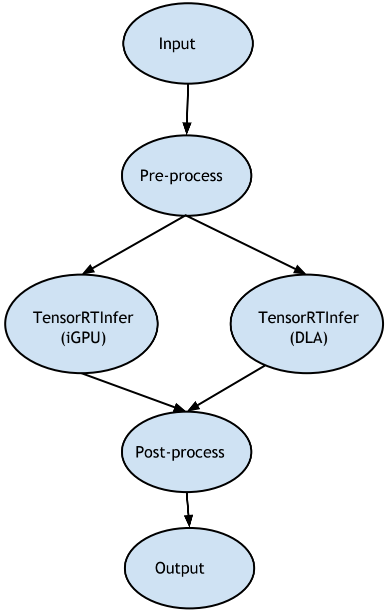
iGPU + DLA pipeline



Parallel Inference Pipeline



iGPU + DLA pipeline

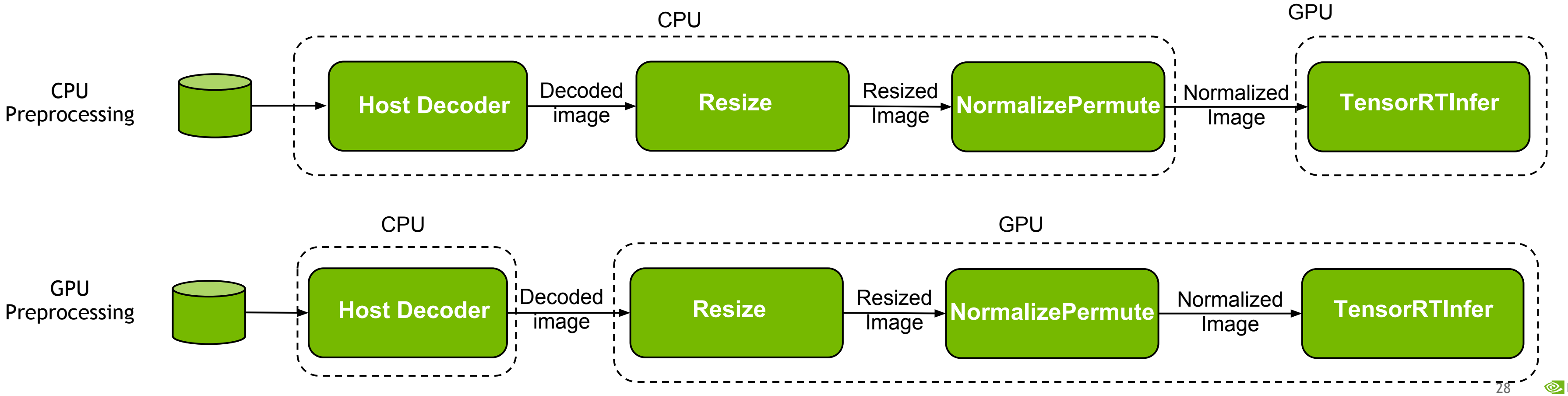


Performance

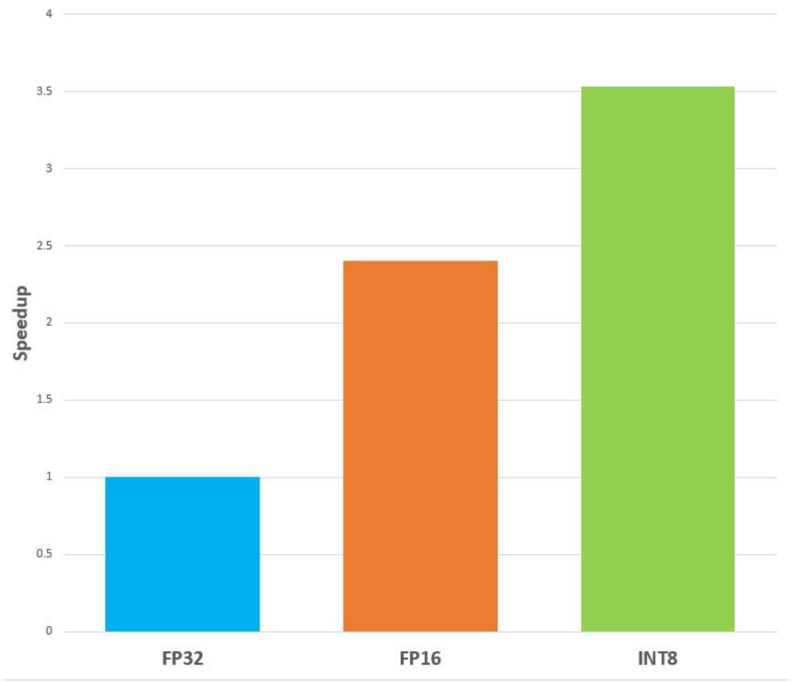
Object Detection Model on DALI

- Model Name: SSD (Backbone ResNet18)
- Input Resolution: 3x1024x1024
- Batch: 1
- HW Platform: TensorRT Inference on Xavier (iGPU)
- OS: QNX 7.0
- CUDA: 10.0
- cuDNN: 7.3.0
- TensorRT: 5.1.1
- Preprocessing: jpeg decoding, resizing, normalizing

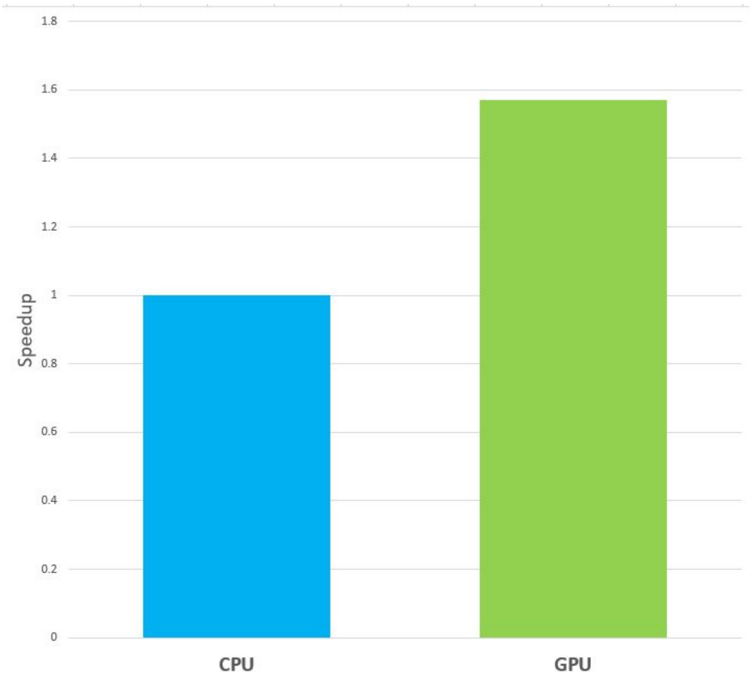
DALI Pipeline



Performance of DALI + TensorRT on Xavier



TensorRT Speedup per Precision (resnet-18)



Preprocessing Speedup via DALI

NVIDIA.

Stay Tuned!

NVIDIA DALI github: https://github.com/NVIDIA/DALI

[PR] Extend DALI for aarch64 platform: https://github.com/NVIDIA/DALI/pull/522

Acknowledgement

Special Thanks to

- NVIDIA DALI Team
  - @Janusz Lisiecki, @Przemek Tredak, @Joaquin Anton Guirao, @Michal Zientkiewicz
- NVIDIA TSE/ADLSA
  - @Muni Anda, @Joohoon Lee, @Naren Sivagnanadasan, @Le An, @Jeff Hetherly, @Yu-Te Cheng
- NVIDIA Developer Marketing
  - @Siddarth Sharma

NVIDIA

Thank You