DEEP LEARNING DEPLOYMENT WITH NVIDIA TENSORRT

Ashish Sardana | Deep Learning Solutions Architect

NVIDIA

AGENDA

Deep Learning in Production

- Current Approaches
- Deployment Challenges

NVIDIA TensorRT

- Programmable Inference Accelerator
- Performance, Optimizations and Features

Example

- Import, Optimize and Deploy
  TensorFlow Models with TensorRT

Key Takeaways and Additional Resources

Q&A

DEEP LEARNING IN PRODUCTION

Speech Recognition

Recommender Systems

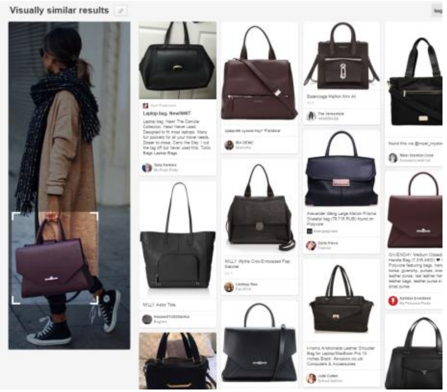
Autonomous Driving

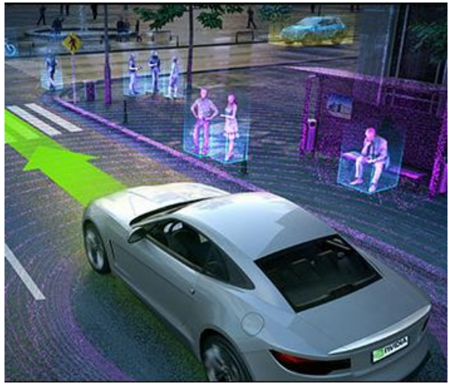
Real-time Object Recognition

Robotics

Real-time Language Translation

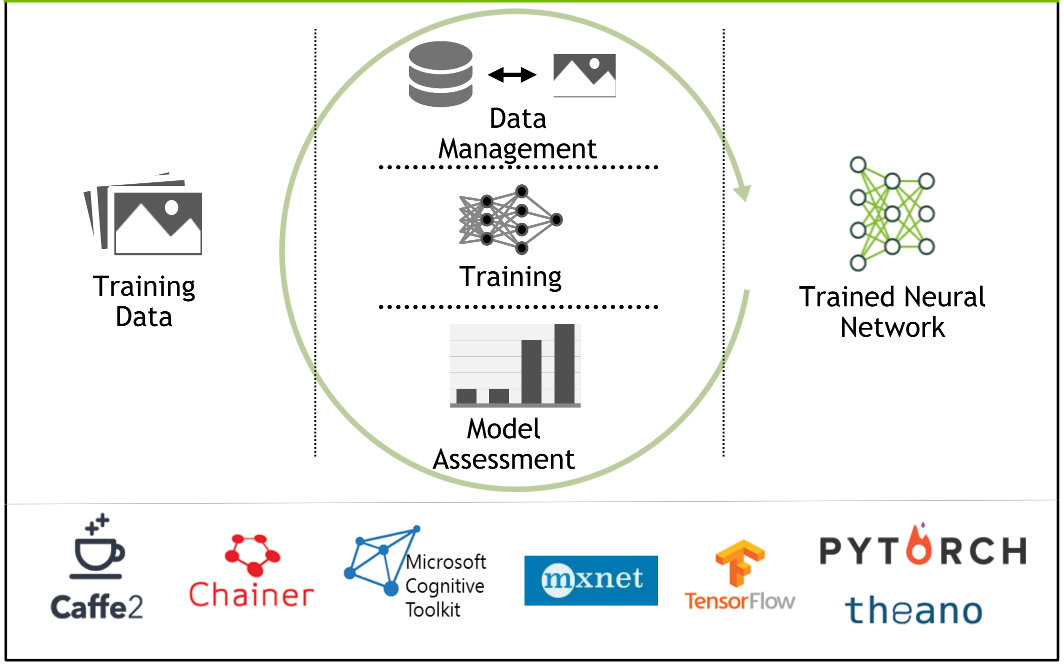
Many More…





CURRENT DEPLOYMENT WORKFLOW

TRAINING



UNOPTIMIZED DEPLOYMENT

1. **Deploy training framework**
2. **Deploy custom application using NVIDIA DL SDK**
3. **Framework or custom CPU-Only application**

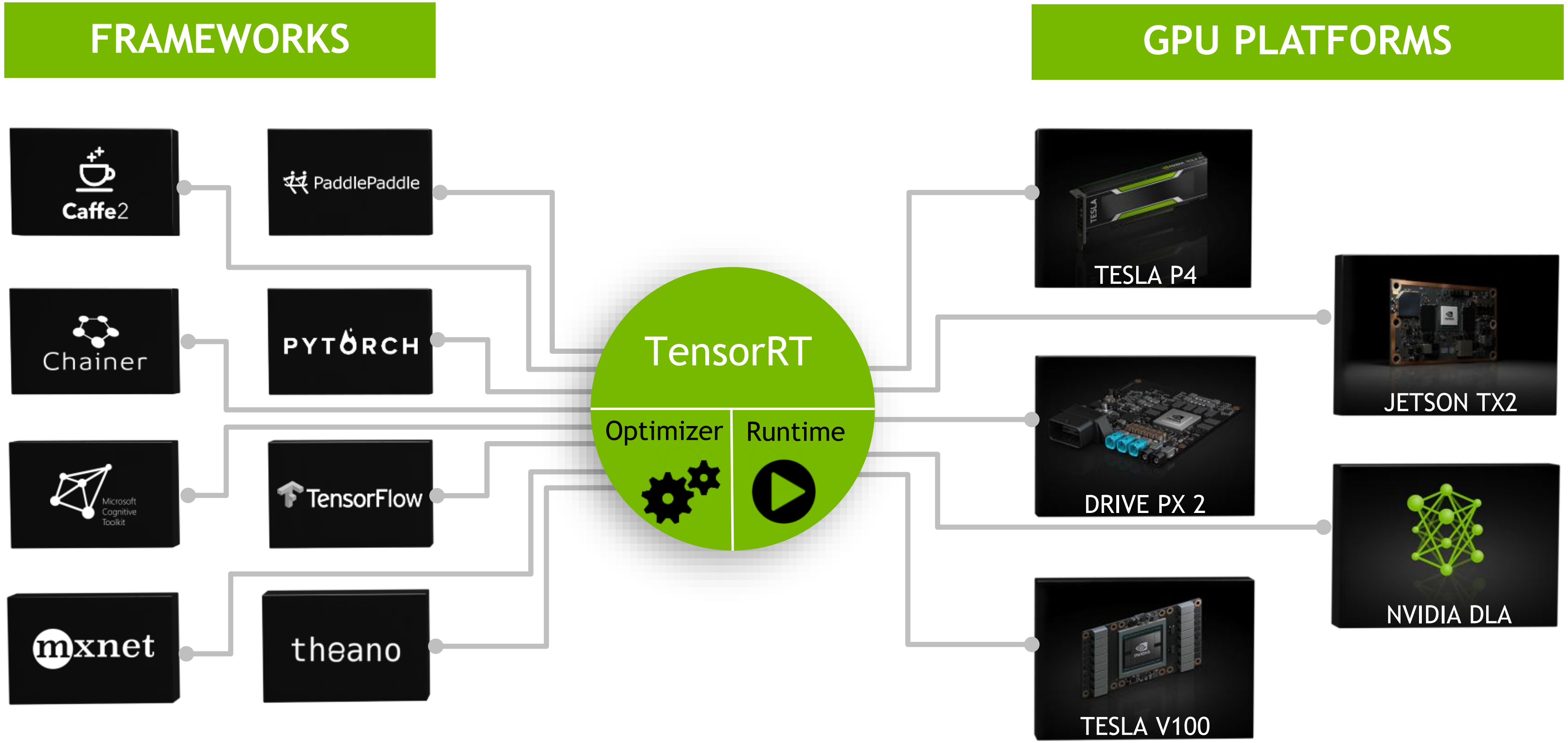
CUDA, NVIDIA Deep Learning SDK (cuDNN, cuBLAS, NCCL)

CHALLENGES WITH CURRENT APPROACHES

Requirement	Challenges
High Throughput	**Unable to processing high-volume, high-velocity data** ➢ Impact: Increased cost ($, time) per inference
Low Response Time	**Applications don't deliver real-time results** ➢ Impact: Negatively affects user experience (voice recognition, personalized recommendations, real-time object detection)
Power and Memory Efficiency	**Inefficient applications** ➢ Impact: Increased cost (running and cooling), makes deployment infeasible
Deployment-Grade Solution	**Research frameworks not designed for production** ➢ Impact: Framework overhead and dependencies increases time to solution and affects productivity

NVIDIA TENSORRT

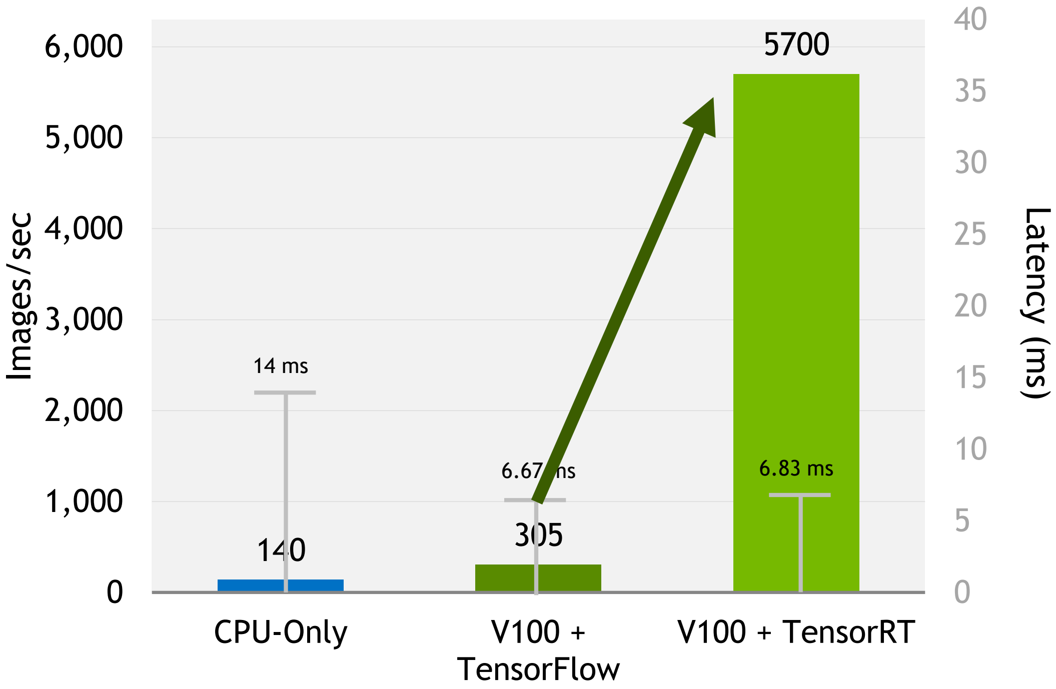
Programmable Inference Accelerator



developer.nvidia.com/tensorrt

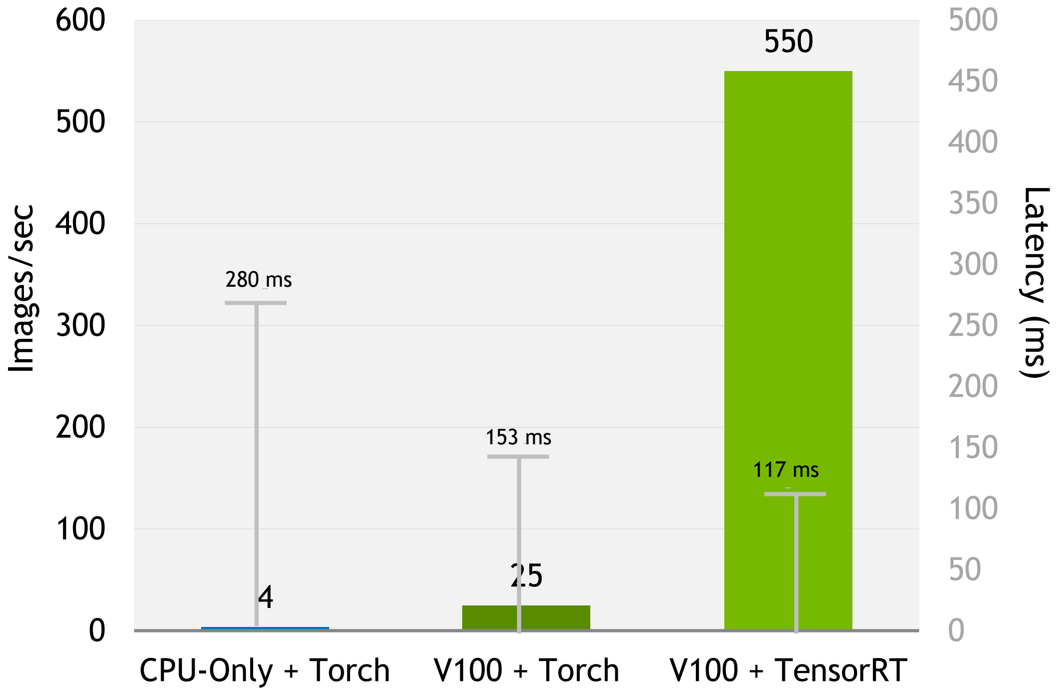
TENSORRT PERFORMANCE

40x Faster CNNs on V100 vs. CPU-Only Under 7ms Latency (ResNet50)



Inference throughput (images/sec) on ResNet50. **V100 + TensorRT**: NVIDIA TensorRT (FP16), batch size 39, Tesla V100-SXM2-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On **V100 + TensorFlow**: Preview of volta optimized TensorFlow (FP16), batch size 2, Tesla V100-PCIE-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On. **CPU-Only:** Intel Xeon-D 1587 Broadwell-E CPU and Intel DL SDK. Score doubled to comprehend Intel's stated claim of 2x performance improvement on Skylake with AVX512.

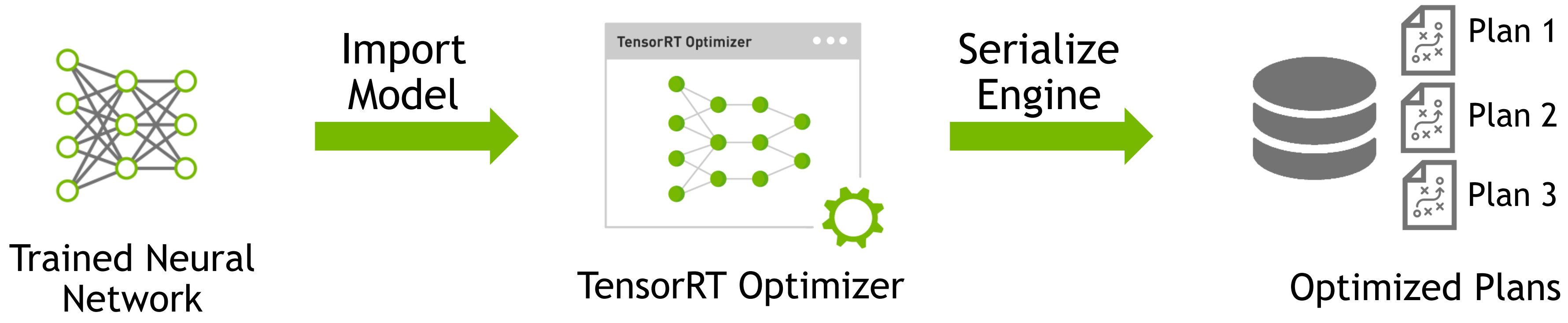
140x Faster Language Translation RNNs on V100 vs. CPU-Only Inference (OpenNMT)



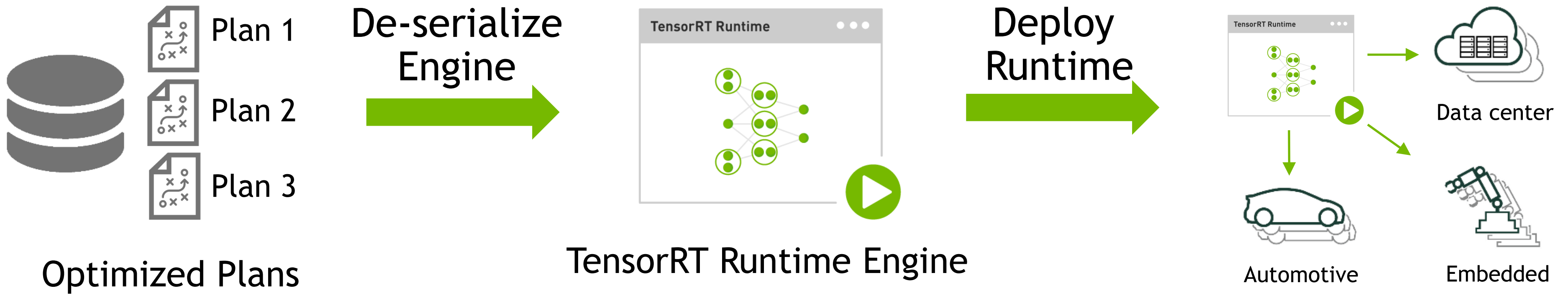
Inference throughput (sentences/sec) on OpenNMT 692M. **V100 + TensorRT**: NVIDIA TensorRT (FP32), batch size 64, Tesla V100-PCIE-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On. **V100 + Torch**: Torch (FP32), batch size 4, Tesla V100-PCIE-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On. **CPU-Only:** Torch (FP32), batch size 1, Intel E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On

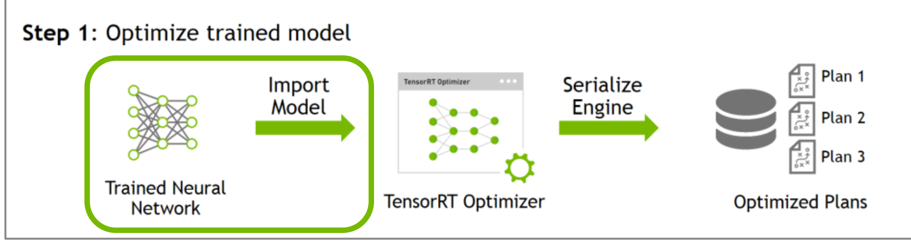
TENSORRT DEPLOYMENT WORKFLOW

Step 1: Optimize trained model



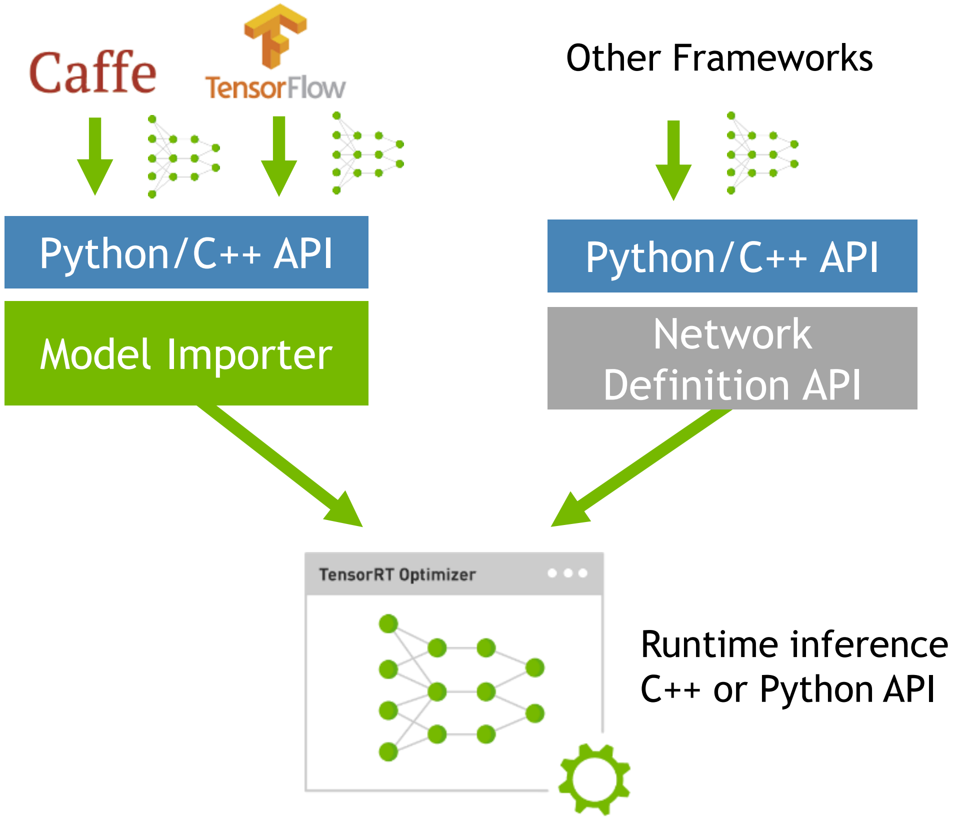
Step 2: Deploy optimized plans with runtime





MODEL IMPORTING

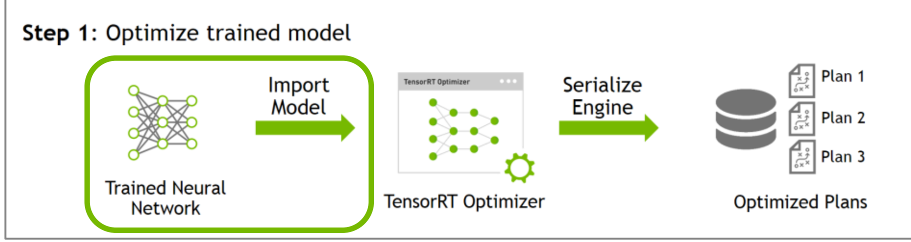
- AI Researchers
- Data Scientists



Example: Importing a TensorFlow model

```
import tensorrt as trt
import uff
from tensorrt.parsers import uffparser

G_LOGGER = trt.infer.ConsoleLogger(trt.infer.LogSeverity.INFO)

uff_model = uff.from_tensorflow_frozen_model("frozen_model.pb",
                                             "dense_2/Softmax")

parser = uffparser.create_uff_parser()
parser.register_input("input_1", (3,224,224),0)
parser.register_output("dense_2/Softmax")

engine = trt.utils.uff_to_trt_engine(G_LOGGER,
                                     uff_model,
                                     parser,
                                     INFERENCE_BATCH_SIZE,
                                     1<<20,
                                     trt.infer.DataType.FLOAT)

runtime = trt.infer.create_infer_runtime(G_LOGGER)
context = engine.create_execution_context()
```

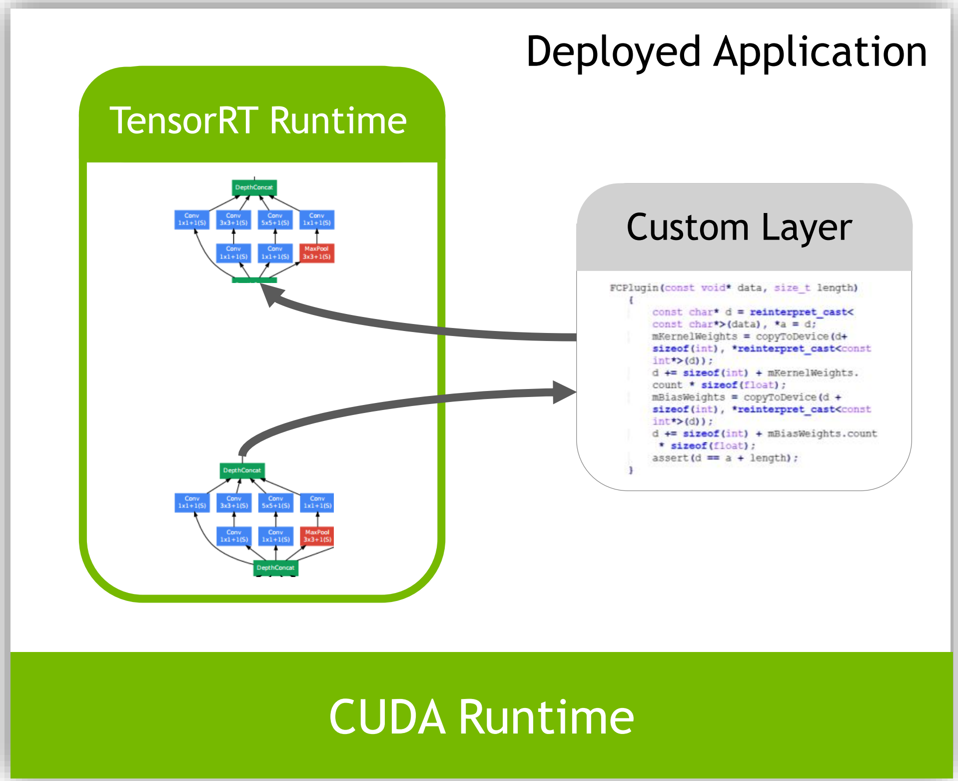
TENSORRT LAYERS



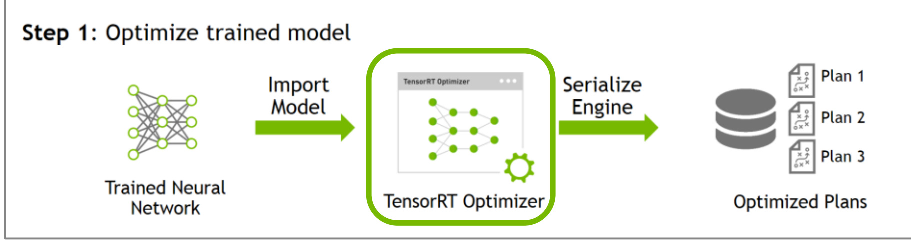
Built-in Layer Support

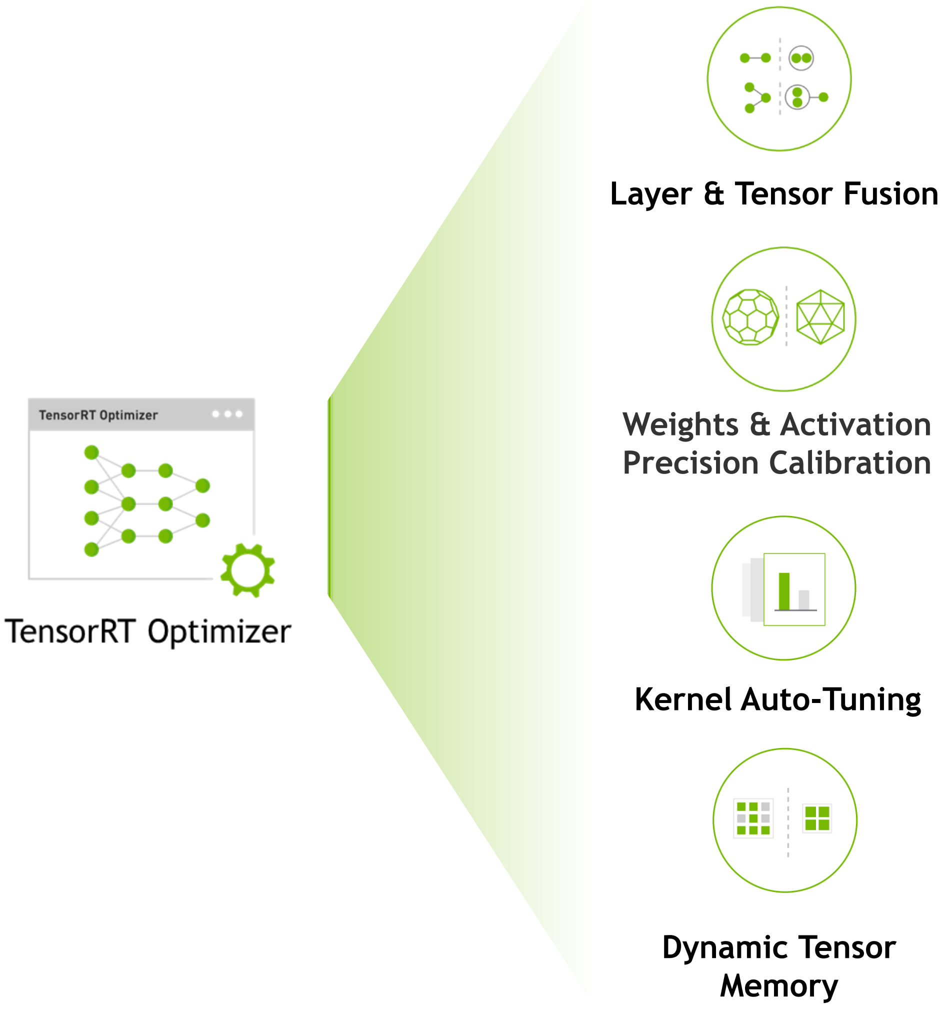
- Convolution
- LSTM and GRU
- Activation: ReLU, tanh, sigmoid
- Pooling: max and average
- Scaling
- Element wise operations
- LRN
- Fully-connected
- SoftMax
- Deconvolution

Custom Layer API



TENSORRT OPTIMIZATIONS

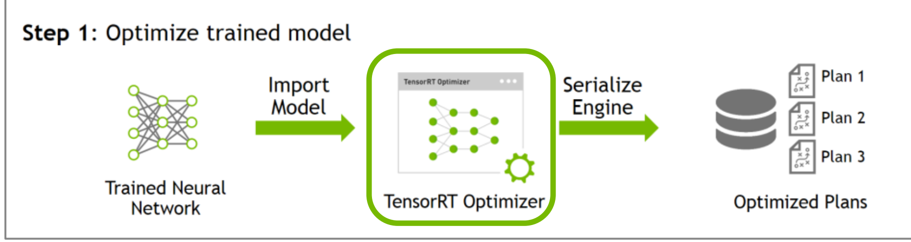




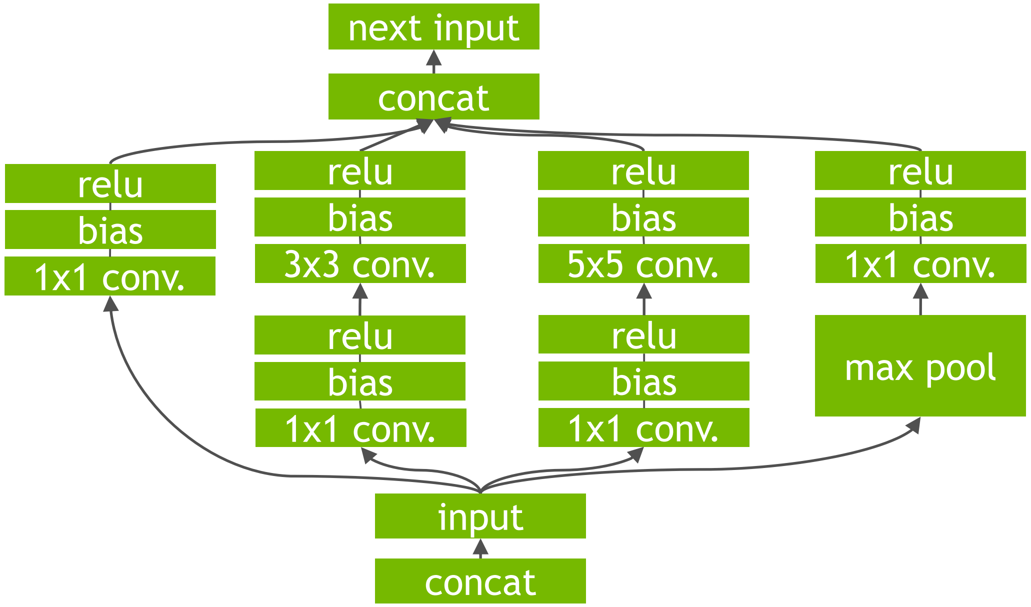
- Optimizations are completely automatic
- Performed with a single function call

```
engine = trt.utils.uff_to_trt_engine(G_LOGGER,
                                     uff_model,
                                     parser,
                                     INFERENCE_BATCH_SIZE,
                                     1<<20,
                                     trt.infer.DataType.FLOAT)
```

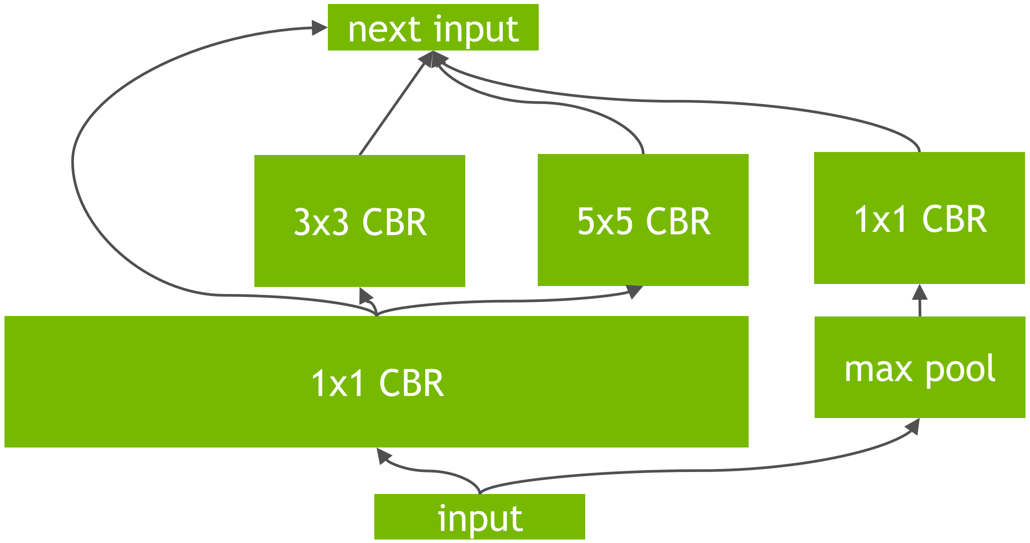
LAYER & TENSOR FUSION



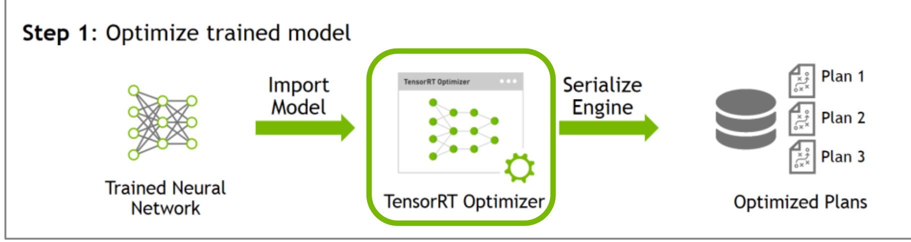
Un-Optimized Network



TensorRT Optimized Network



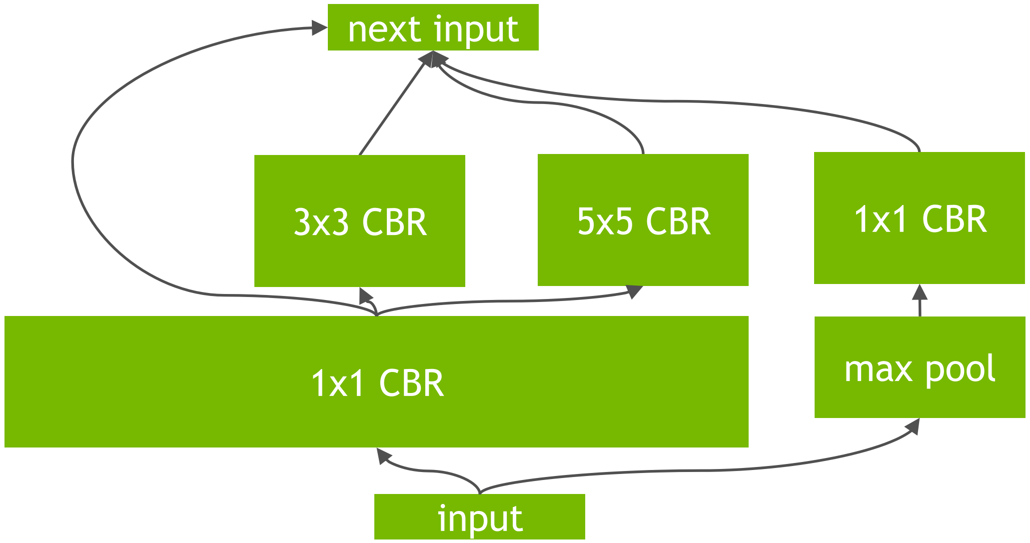
LAYER & TENSOR FUSION



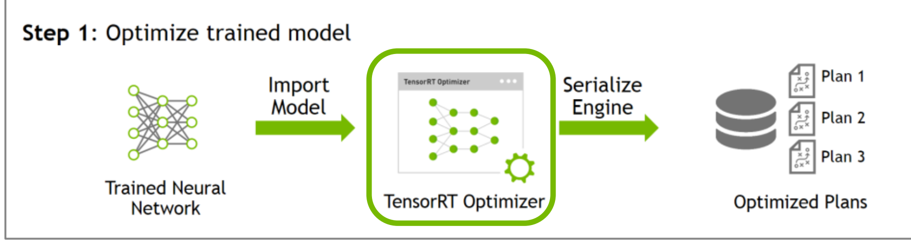
- Vertical Fusion
- Horizonal Fusion
- Layer Elimination

Network	Layers before	Layers after
VGG19	43	27
Inception V3	309	113
ResNet-152	670	159

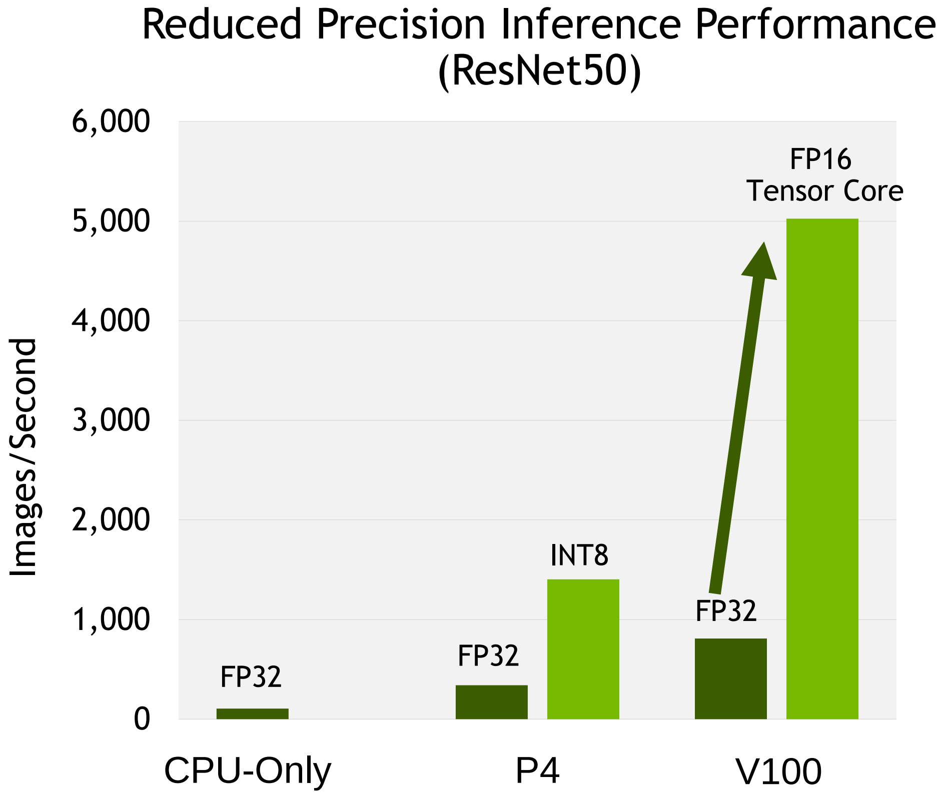
TensorRT Optimized Network



FP16, INT8 PRECISION CALIBRATION



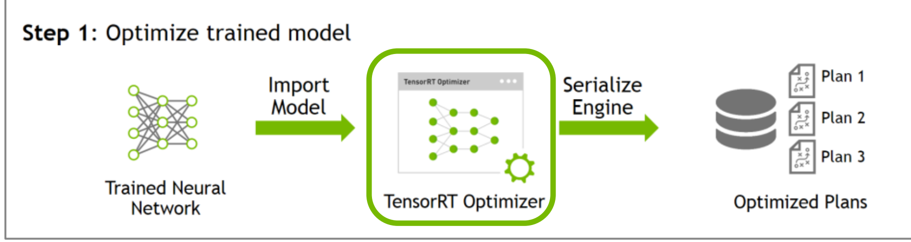
Precision	Dynamic Range	
FP32	-3.4×10^{38} ~ $+3.4\times10^{38}$	⟵ Training precision
FP16	-65504 ~ +65504	⟵ No calibration required
INT8	-128 ~ +127	⟵ **Requires calibration**



Precision calibration for INT8 inference:

- Minimizes information loss between FP32 and INT8 inference on a calibration dataset
- Completely automatic

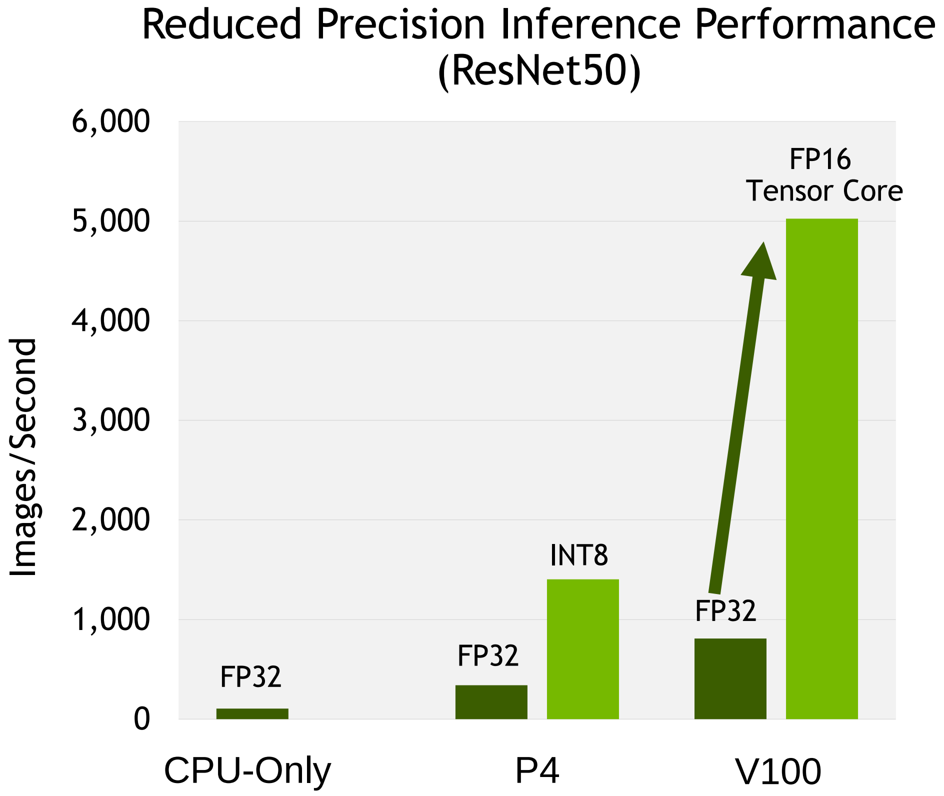
FP16, INT8 PRECISION CALIBRATION

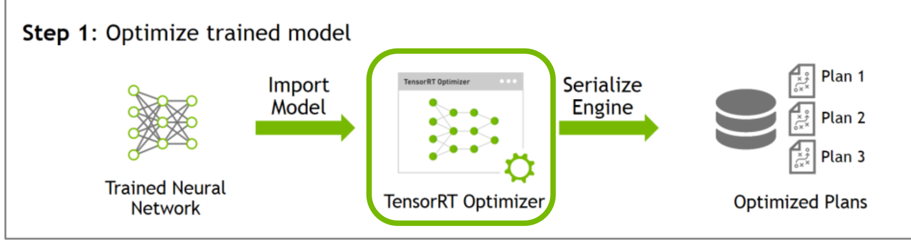


	FP32 Top 1	INT8 Top 1	Difference
Googlenet	68.87%	68.49%	0.38%
VGG	68.56%	68.45%	0.11%
Resnet-50	73.11%	72.54%	0.57%
Resnet-152	75.18%	74.56%	0.61%

Precision calibration for INT8 inference:

- Minimizes information loss between FP32 and INT8 inference on a calibration dataset
- Completely automatic





KERNEL AUTO-TUNING
DYNAMIC TENSOR MEMORY

Kernel Auto-Tuning

100s for specialized kernels
Optimized for every GPU platform

Tesla V100

Jetson TX2

Drive PX2

Multiple parameters:
- Batch size
- Input dimensions
- Filter dimensions

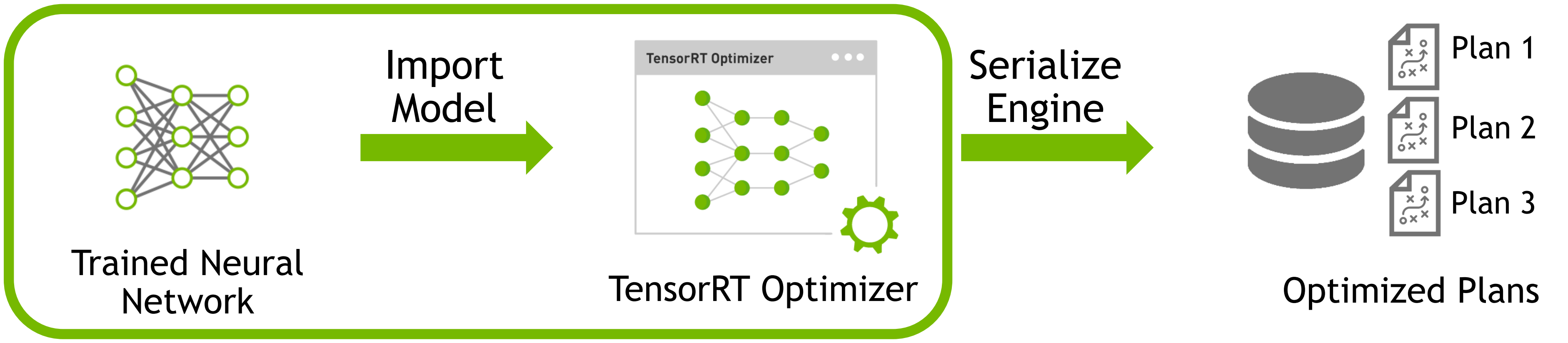
...

Dynamic Tensor Memory

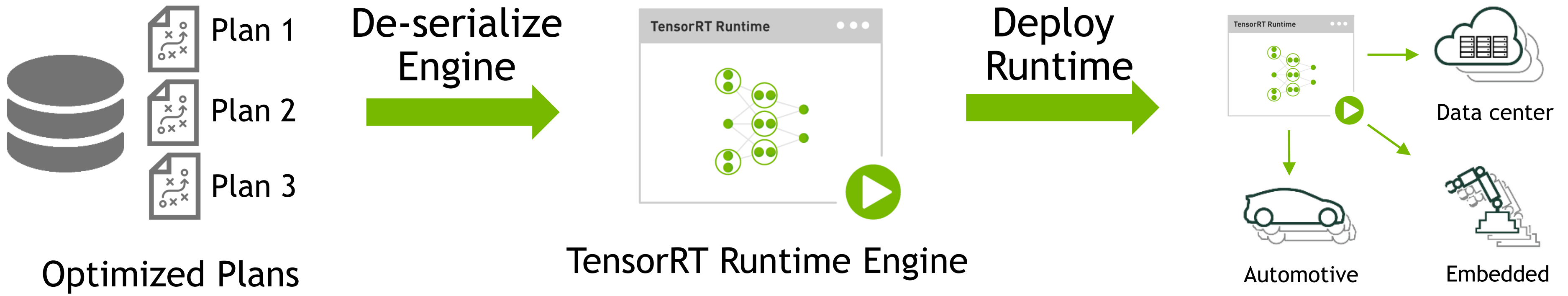
- Reduces memory footprint and improves memory re-use
- Manages memory allocation for each tensor only for the duration of its usage

TENSORRT DEPLOYMENT WORKFLOW

Step 1: Optimize trained model



Step 2: Deploy optimized plans with runtime

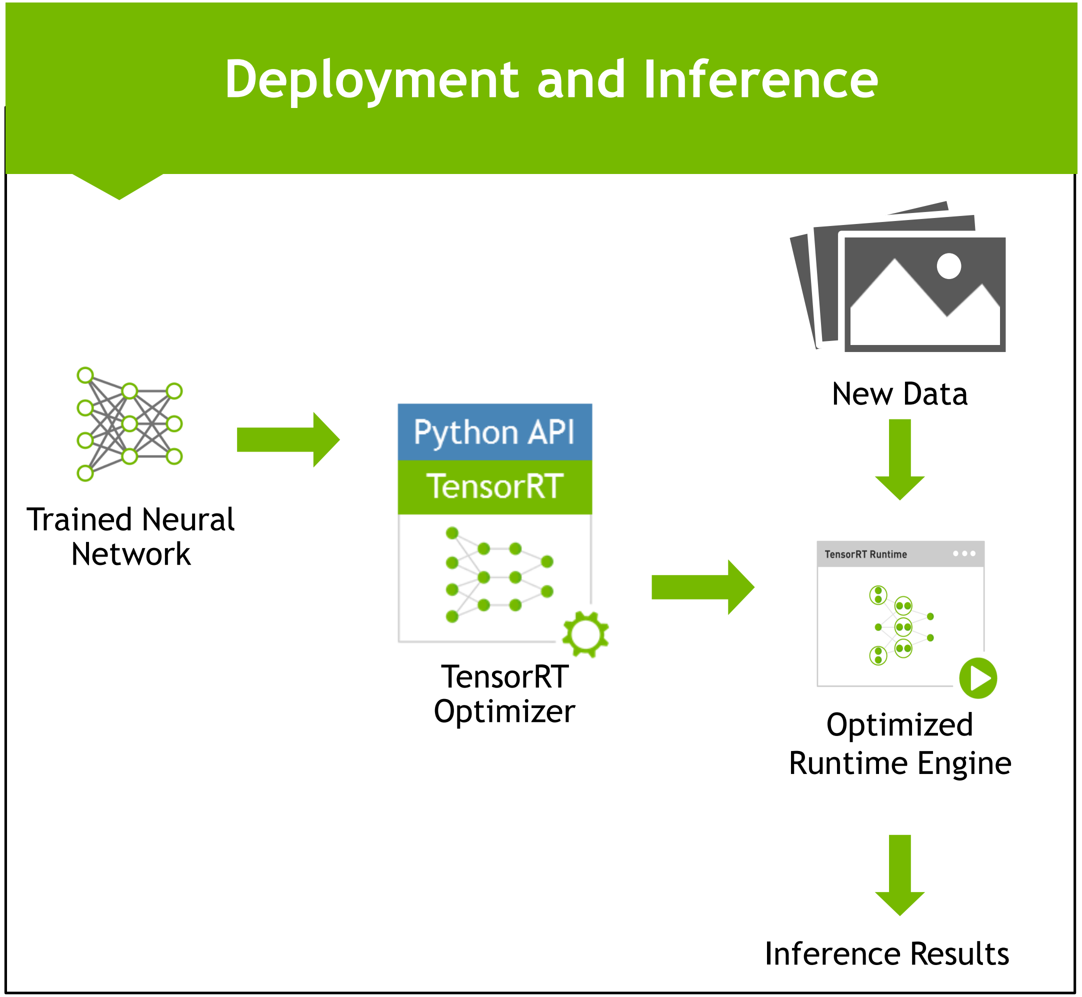


EXAMPLE: DEPLOYING TENSORFLOW MODELS WITH TENSORRT

Import, optimize and deploy TensorFlow models using TensorRT python API

Steps:

- Start with a frozen TensorFlow model
- Create a model parser
- Optimize model and create a runtime engine
- Perform inference using the optimized runtime engine



7 STEPS TO DEPLOYMENT WITH TENSORRT

```
uff_model = uff.from_tensorflow_frozen_model("frozen_model_file.pb",
                                             OUTPUT_LAYERS)

parser = uffparser.create_uff_parser()


parser.register_input(INPUT_LAYERS[0], (INPUT_C,INPUT_H,INPUT_W),0)
parser.register_output(OUTPUT_LAYERS[0])


engine = trt.utils.uff_to_trt_engine(G_LOGGER,
                                     uff_model,
                                     parser,
                                     INFERENCE_BATCH_SIZE,
                                     1<<20,
                                     trt.infer.DataType.FLOAT)


trt.utils.write_engine_to_file(save_path, engine.serialize())




engine = Engine(PLAN=plan,
                postprocessors={"output_layer_name":post_processing_function})

result = engine_single.infer(image)
```

Step 1: Convert trained model into TensorRT format

Step 2: Create a model parser

Step 3: Register inputs and outputs

Step 4: Optimize model and create a runtime engine

Step 5: Serialize optimized engine

Step 6: De-serialize engine

Step 7: Perform inference

developer.nvidia.com/tensorrt

RECAP: DEPLOYMENT WORKFLOW

Step 1: Optimize trained model

VGG19 → Import Model → TensorRT Optimizer → Serialize Engine → Plan file

VGG19

FP32, FP16, Batch Size 1

keras_vgg19_b1_fp32.engine

Step 2: Deploy optimized plans with runtime

Plan file → De-serialize Engine → TensorRT Runtime → Deploy Runtime → TensorRT Runtime

keras_vgg19_b1_fp32.engine

TensorRT Runtime Engine

New flower images

Prediction Results

CHALLENGES ADDRESSED BY TENSORRT

Requirement	*TensorRT Delivers*
High Throughput **Low Response Time**	**Maximizes inference performance on NVIDIA GPUs** ➢ INT8, FP16 Precision Calibration, Layer & Tensor Fusion, Kernel Auto-Tuning ➢ Up to 40x Faster than CPU-Only inference and 18x faster inference of TensorFlow models ➢ Under 7ms real-time latency
Power and Memory Efficiency	**Performs target specific optimizations** ➢ Platform specific kernels for Embedded (Jetson), Datacenter (Tesla GPUs) and Automotive (DrivePX) ➢ Dynamic Tensor Memory management improves memory re-use
Deployment-Grade Solution	**Designed for production environments** ➢ No framework overhead, minimal dependencies ➢ Multiple frameworks, Network Definition API ➢ C++, Python API, Customer Layer API

TENSORRT PRODUCTION USE CASES

"NVIDIA's AI platform, **using TensorRT software on Tesla GPUs**, is the best technology on the market to support SAP's requirements for inferencing. TensorRT and NVIDIA GPUs changed our business model **from an offline, next-day service to real-time**. We have maximum AI performance and versatility to meet our customers' needs, while substantially reducing energy requirements."

Source: JUERGEN MUELLER, SAP Chief Innovation Officer

"Real-time execution is very important for self-driving cars. Developing state of the art perception algorithms normally requires a painful trade-off between speed and accuracy, but **TensorRT brought our ResNet-151 inference time down from 250ms to 89ms.**"

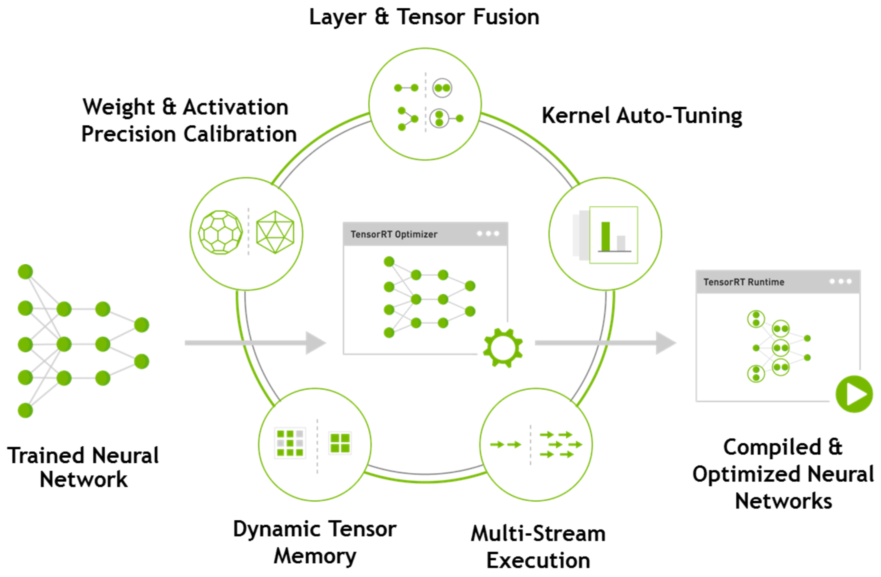
Source: Drew Gray – Director of Engineering, UBER ATG

"**TensorRT is a real game changer.** Not only does TensorRT make model deployment a snap but the resulting speed up is incredible: out of the box, BodySLAM™, our human pose estimation engine, now runs over **two times faster than using CAFFE GPU inferencing**."

Source: Paul Kruszewski, CEO - WRNCH

TENSORRT KEY TAKEAWAYS

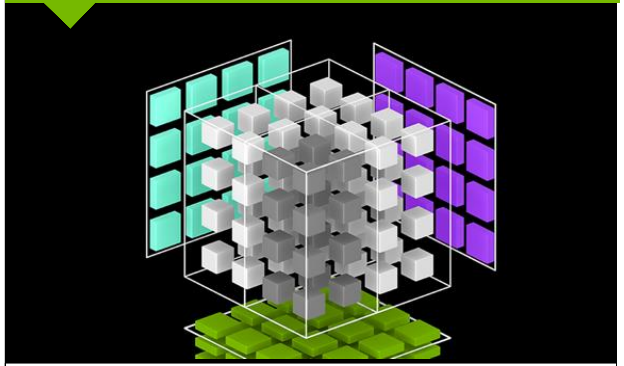
- ✓ Generate optimized, deployment-ready runtime engines for low latency inference
- ✓ Import models trained using Caffe or TensorFlow or use Network Definition API
- ✓ Deploy in FP32 or reduced precision INT8, FP16 for higher throughput
- ✓ Optimize frequently used layers and integrate user defined custom layers



NVIDIA TENSORRT 5 RC NOW AVAILABLE

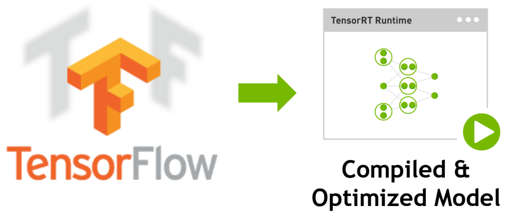
Volta TensorCore • TensorFlow Importer • Python API

Volta TensorCore Support



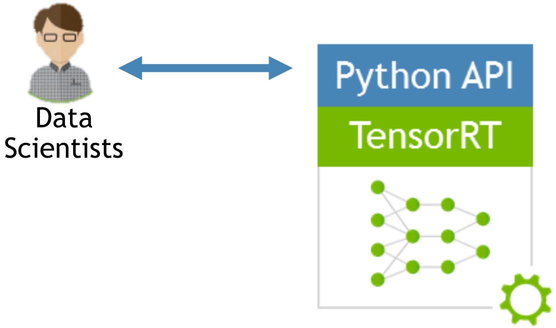
3.7x faster inference on Tesla V100 vs. Tesla P100 under 7ms real-time latency

Import TensorFlow Models



Optimize and deploy TensorFlow models up to **18x faster** vs. TensorFlow framework

Python API



Improved productivity with **easy to use** Python API for data science workflows

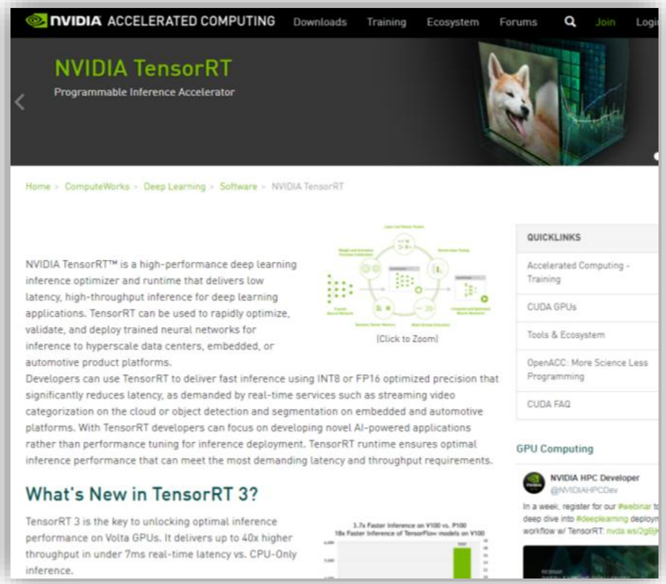
Free download to members of NVIDIA Developer Program

developer.nvidia.com/tensorrt

LEARN MORE

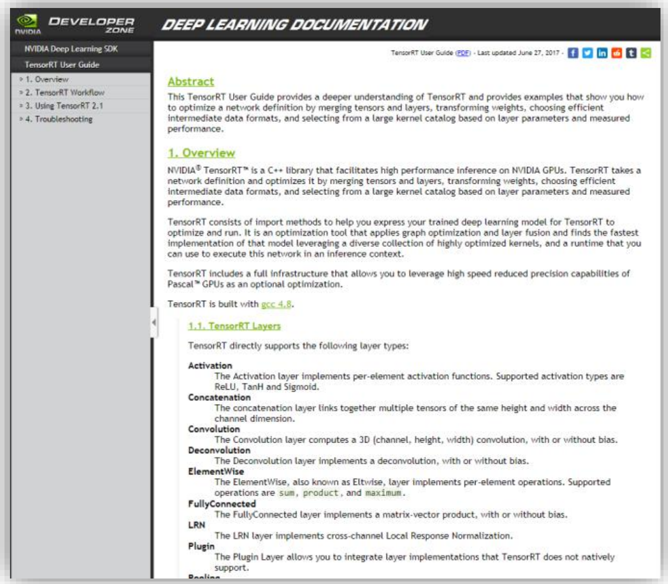
PRODUCT PAGE

developer.nvidia.com/tensorrt



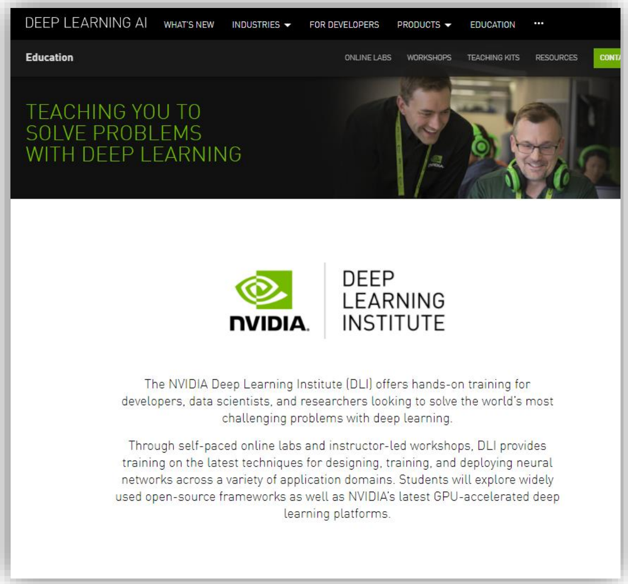
DOCUMENTATION

docs.nvidia.com/deeplearning/sdk



TRAINING

nvidia.com/dli



NVIDIA