TensorRT 实现深度网络模型推理加速

Sherry wang, Nov.
AGENDA

Deep Learning in Production
- Current Approaches
- Deployment Challenges
- TensorRT Solutions and Performance

NVIDIA TensorRT
- Programmable Inference Accelerator
- Performance, Optimizations and Features

What is new in TensorRT5

Example
- TensorFlow Models with TensorRT
- Caffe inference vs TensorRT inference

Q&A
Deep Learning in Production

✓ Challenges
✓ Solutions
✓ Performance
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

Data Management

Training

Model Assessment

Trained Neural Network

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

UNOPTIMIZED DEPLOYMENT

1. Deploy training framework, such as Caffe, TensorFlow

2. Deploy custom application using NVIDIA DL SDK

3. Framework or custom CPU-Only application
## CHALLENGES WITH CURRENT APPROACHES

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Throughput</td>
<td>Unable to processing high-volume, high-velocity data</td>
</tr>
<tr>
<td></td>
<td>➢ Impact: Increased cost ($, time) per inference</td>
</tr>
<tr>
<td>Low Response Time</td>
<td>Applications don’t deliver real-time results</td>
</tr>
<tr>
<td></td>
<td>➢ Impact: Negatively affects user experience (voice recognition, personalized recommendations, real-time object detection)</td>
</tr>
<tr>
<td>Power and Memory Efficiency</td>
<td>Inefficient applications</td>
</tr>
<tr>
<td></td>
<td>➢ Impact: Increased cost (running and cooling), makes deployment infeasible</td>
</tr>
<tr>
<td>Elegant Deployment Solution</td>
<td>Research frameworks not designed for production</td>
</tr>
<tr>
<td></td>
<td>➢ Impact: Framework overhead and dependencies increases time to solution and affects productivity</td>
</tr>
</tbody>
</table>
## CHALLENGES ADDRESSED BY TENSORRT

<table>
<thead>
<tr>
<th>Requirement</th>
<th>TensorRT Delivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Throughput</td>
<td>Maximizes inference performance on NVIDIA GPUs</td>
</tr>
<tr>
<td></td>
<td>➢ INT8, FP16 Precision Calibration, Layer &amp; Tensor Fusion, Kernel Auto-Tuning</td>
</tr>
<tr>
<td>Low Response Time</td>
<td>➢ Up to 40x Faster than CPU-Only inference and 18x faster inference of TensorFlow models</td>
</tr>
<tr>
<td></td>
<td>➢ Under 7ms real-time latency</td>
</tr>
<tr>
<td>Power and Memory Efficiency</td>
<td>Performs target specific optimizations</td>
</tr>
<tr>
<td></td>
<td>➢ Platform specific kernels</td>
</tr>
<tr>
<td></td>
<td>➢ Dynamic Tensor Memory management improves memory re-use</td>
</tr>
<tr>
<td>Elegant Deployment Solution</td>
<td>Designed for production environments</td>
</tr>
<tr>
<td></td>
<td>➢ No framework overhead, minimal dependencies</td>
</tr>
<tr>
<td></td>
<td>➢ Multiple frameworks, Network Definition API</td>
</tr>
<tr>
<td></td>
<td>➢ C++, Python API, Customer Layer API</td>
</tr>
</tbody>
</table>
30X HIGHER THROUGHPUT

Tesla P4 and TensorRT: Highest Performance Inferencing Platform for Scale-out Data Center

- **AlexNet**:
  - BDW-E5-2690 CPU: 2,370 images/second
  - Tesla P4 (FP32): 8,780 images/second
  - Tesla P4 (INT8): 3,420 images/second

- **GoogleNet**:
  - BDW-E5-2690 CPU: 1069 images/second
  - Tesla P4 (FP32): 2,370 images/second
  - Tesla P4 (INT8): 1,193 images/second

- **VGG**:
  - BDW-E5-2690 CPU: 22 images/second
  - Tesla P4 (FP32): 126 images/second
  - Tesla P4 (INT8): 409 images/second

- **ResNet**:
  - BDW-E5-2690 CPU: 21 images/second
  - Tesla P4 (FP32): 189 images/second
  - Tesla P4 (INT8): 645 images/second

**System Configs:**
- GPU tests run on NVIDIA Tesla P4 on same server platform running TensorRT 2.0.0.0, batch size = 128.
15X FASTER RESPONSIVENESS

Tesla P4 and TensorRT: Inferencing Platform for Real-Time Responsiveness

Latency Required for Real-Time Video @30FPS

System Configs: dual/socket Intel® Xeon® E5-2690v4 (12-core/24-thread, 2.6GHz base, 3.5GHz turbo, 35MB cache), 256GB system RAM, Ubuntu 14.04.5, Intel Math Kernel Library 2017 gold release. GPU tests run on NVIDIA Tesla P4 on same server platform running TensorRT 2.0.c, batch size = 2
60X HIGHER POWER EFFICIENCY

Tesla P4 and TensorRT: Most Power Efficient Inferencing Platform

System Configs: dual-socket Intel® Xeon® E5-2690v4 (12-core/24-thread, 2.6GHz base, 3.5GHz turbo, 35MB cache), 256GB system RAM, Ubuntu 14.04.5, Intel Math Kernel Library 2017 gold release. GPU tests run on NVIDIA Tesla P4 on same server platform running TensorRT 2.0.0, batch size = 128.
NVIDIA TensorRT

✓ Deploy Workflow
✓ Optimization Strategy
✓ Model Types Supported
NVIDIA TensorRT
Deep Learning Inference Optimizer and Runtime

High performance neural network inference optimizer and runtime engine for production deployment

Maximize inference throughput for latency-critical services in hyperscale datacenters, embedded, and automotive production environments

Optimize TensorFlow and ONNX-framework models to generate high-performance runtime engines

Deploy faster, more responsive and memory efficient deep learning applications with INT8 and FP16 optimized precision support

developer.nvidia.com/tensorrt
TENSORRT DEPLOYMENT WORKFLOW

Step 1: Optimize trained model

1. Import Model
2. TensorRT Optimizer
3. Serialize Engine
4. Optimized Plans

Trained Neural Network

Step 2: Deploy optimized plans with runtime

1. De-serialize Engine
2. TensorRT Runtime Engine
3. Deploy Runtime
4. Optimized Plans

Embedded
Automotive
Data center
TENSORRT OPTIMIZER

Layer & Tensor Fusion

Precision Calibration

Kernel Auto-Tuning

Trained Neural Network

Dynamic Tensor Memory

Multi-Stream Execution

Optimized Inference Engine
**FP16, INT8 PRECISION CALIBRATION**

<table>
<thead>
<tr>
<th>Precision</th>
<th>Dynamic Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP32</td>
<td>-3.4x10^{38} ~ +3.4x10^{38}</td>
</tr>
<tr>
<td>FP16</td>
<td>-65504 ~ +65504</td>
</tr>
<tr>
<td>INT8</td>
<td>-128 ~ +127</td>
</tr>
</tbody>
</table>

- **Training precision**
- **No calibration required**
- **Requires calibration**

**Precision calibration for INT8 inference:**
- Minimizes information loss between FP32 and INT8 inference on a calibration dataset
- Completely automatic

**Reduced Precision Inference Performance (ResNet50)**

- **FP32**
- **FP16**
- **INT8**

- **Images/Second**
  - CPU-Only
  - P4
  - V100

**Step 1: Optimize trained model**
- Import Model
- TensorRT Optimizer
- Serialize Engine
- Optimized Plans
FP16, INT8 PRECISION CALIBRATION

<table>
<thead>
<tr>
<th></th>
<th>FP32 Top 1</th>
<th>INT8 Top 1</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Googlenet</td>
<td>68.87%</td>
<td>68.49%</td>
<td>0.38%</td>
</tr>
<tr>
<td>VGG</td>
<td>68.56%</td>
<td>68.45%</td>
<td>0.11%</td>
</tr>
<tr>
<td>Resnet-50</td>
<td>73.11%</td>
<td>72.54%</td>
<td>0.57%</td>
</tr>
<tr>
<td>Resnet-152</td>
<td>75.18%</td>
<td>74.56%</td>
<td>0.61%</td>
</tr>
</tbody>
</table>

Precision calibration for INT8 inference:
- Minimizes information loss between FP32 and INT8 inference on a calibration dataset
- Completely automatic/custom calibration/Quantization Aware Training

Reduced Precision Inference Performance (ResNet50)
Graph Optimizer
Horizontal Layer Fusion (Layer Aggregation)

CBR = Convolution, Bias and ReLU
Graph Optimizer
eliminate the concatenation layers
# Graph Optimizer

Table: Number of layers before and after vertical and horizontal fusions and unused layer elimination

<table>
<thead>
<tr>
<th>Network</th>
<th>Layers</th>
<th>Layers after fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG19</td>
<td>43</td>
<td>27</td>
</tr>
<tr>
<td>Inception V3</td>
<td>309</td>
<td>113</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>670</td>
<td>159</td>
</tr>
</tbody>
</table>
KERNEL AUTO-TUNING
DYNAMIC TENSOR MEMORY

Kernel Auto-Tuning

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

Specific Algorithm:
- Direct gemm,
- Winograd,
- FFT

Specific GPU platform:
- Tesla P4/40
- Tesla V100
- Jetson TX2
- Drive PX2

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 provides model importers for Caffe and TensorFlow. Other framework models can be imported using the Network Definition API.
TENSORRT LAYERS

Built-in support

- Convolution(3D), Deconvolution
- Activation: relu, tanh, sigmoid
- Pooling: max and average
- Scaling
- Element wise operations
- LRN
- Fully-connected
- SoftMax
- Gather
  - Flatten
  - TopK
  - Const
  - Concatenation
  - BatchGemm
  - Padding
  - RNN
  - RNNv2(GRU, and LSTM)
  - Shuffle
  - Squeeze

Custom Layer API

Application

TensorRT

CUDA Runtime

Custom Layer

PLUGIN LAYERS

TensorRT1

IExecutionContext *contextA = engineA->createExecutionContext();
IExecutionContext *contextB = engineB->createExecutionContext();
<...

contextA.enqueue(batchSize, buffersA, stream, nullptr);
myLayer(outputFromA, inputToB, stream);
contextB.enqueue(batchSize, buffersB, stream, nullptr);

TensorRT Runtime

Deployed Application

Custom Layer

CUDA Runtime

TensorRT2/3/4/5
What’s new in TensorRT5
TENSORRT 5 & TENSORRT INFERENCE SERVER

Turing Support ● Optimizations & APIs ● Inference Server

World’s Most Advanced Inference Accelerator

Up to 40x faster inference for apps such as translation using mixed precision on Turing Tensor Cores

New optimizations & flexible INT8 APIs

Achieve highest throughput at low latency with newly optimized operations, INT8 workflows, and support for Win and CentOS

TensorRT inference server

Maximize GPU utilization by executing multiple models from different frameworks on a node via API

Free download to members of NVIDIA Developer Program soon at developer.nvidia.com/tensorrt
TENSORRT 5 SUPPORTS TURING GPUs

Fastest inference using mixed precision (FP32, FP16, INT8) and Turing Tensor Cores

Speed up recommender, speech, video and translation inference in production

Optimized kernels for mixed precision (FP32, FP16, INT8) workloads on Turing GPUs

Up to 40x faster inference for apps vs CPU-only platforms

MPS maximizes utilization and delivers higher overall throughput when you want to run multiple separate inference processes

developer.nvidia.com/tensorrt
NEW INT8 APIs AND OPTIMIZATIONS

High-performance optimizations and flexible APIs for mixed precision inference

Maximize throughput at low latency with mixed precision compute in production

- Apply INT8 quantization aware training or custom calibration algorithms with new APIs
- Control precision per-layer with new APIs
- Optimizations for depth wise convolution operation

[Diagram]

Auto Calibration

FP32 Training → Optimize to INT8
FP32 weights
Calibration Data O(1000) Images

Custom Calibration

FP32 Training → Custom Calibration → Optimize to INT8
FP32 weights → Custom Activation ranges → FP32 or INT8 weights

Quantization Aware Training

Quantization Aware Training in FP32 → Optimize to INT8
Custom Activation Ranges → FP32 or accurate INT8 weights

[Link]
developer.nvidia.com/tensorrt
TENSORRT INFEERENCE SERVER

Containerized Microservice for Data Center Inference

Multiple models scalable across GPUs

- Supports all popular AI frameworks
- Seamless integration into DevOps deployments leveraging Docker and Kubernetes
- Ready-to-run container, free from the NGC container registry

![Diagram of TENSORRT INFEERENCE SERVER components: DNN Models, NV DL SDK, NV Docker, TensorRT Inference Server, Kubernetes]
TENSORRT INTEGRATED WITH TENSORFLOW

Speed up TensorFlow inference with TensorRT optimizations

Speed up TensorFlow model inference with TensorRT with new TensorFlow APIs

- Simple API to use TensorRT within TensorFlow easily
- Sub-graph optimization with fallback offers flexibility of TensorFlow and optimizations of TensorRT
- Optimizations for FP32, FP16 and INT8 with use of Tensor Cores automatically

# Apply TensorRT optimizations
trt_graph = trt.create_inference_graph(frozen_graph_def, 
output_node_name,
max_batch_size=batch_size, 
max_workspace_size_bytes=workspace_size, 
precision_mode=precision)

# INT8 specific graph conversion
trt_graph = trt.calib_graph_to_infer_graph(calibGraph)

Available from TensorFlow 1.7
https://github.com/tensorflow/tensorflow

developer.nvidia.com/tensorrt
Sub-Graph Optimizations within TensorFlow
Sub graph is replaced by a single TensorRT "op" in TensorFlow

This is the sub graph that can be accelerated by TensorRT

The rest of the graph runs in TensorFlow as before!
Examples
7 STEPS TO DEPLOYMENT WITH TENSORRT

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