PROFILING DEEP LEARNING NETWORKS AND AUTOMATIC MIXED PRECISION FOR OPTIMIZATION

Sheng Luo NVIDIA 深度学习解决方案架构师
Training Deep Learning Models is a time and compute intensive process

Optimizing Deep Learning Models involves iterating through

- Locating optimization opportunities (Profiling)
- Visualizing and Analyzing key areas of improvement
- Taking Action

Profiling helps to find where to update code to accelerate training time on GPUs without loss of accuracy
CONCISE PROFILING PERSONAS IN DEEP LEARNING

Researchers
Goal: Fast development, optimize threads, kernels, easy optimization tools

Data Scientists & Applied Researchers
Goal: Reduced Training time Visualization, intuitive tools

Sysadmins & IT DevOps
Goal: Maximum uptime Understanding TCO health checking tools

Skills in Algorithms -- Skills in Domains & Applications -- Skills in Systems
PROFILING TOOLS & TECHNOLOGIES

Researchers

- NVTX
- Nsight Systems
- Nsight Compute

Data Scientists & Applied Researchers

- DLProf
- Tensorboard
- <Nsight Systems w/ NVTX>

Sysadmins & DevOps

- Data Center Monitoring Tools
- DCGM, NVML
- <Nsight Systems>

Skills in Algorithms --------> Skills in Domains & Applications-----------> Skills in Systems
DATA SCIENTIST & RESEARCHER CHALLENGES

- Limited CUDA skills
- Using TensorBoard but no GPU time or usage
- How to locate opportunities for speed ups using reduced precision
- Reason why some operations are not using Tensor cores could be as easy as dimensions of the matrix are wrong, need to be divisible by 8

How to discover where performance can be improved?
NVIDIA PROFILING TOOLS FOR DEEP LEARNING

Deep Learning Profiler (DLProf)

Nsight Systems

Nsight Compute

NVTX for TensorFlow

NVTX for PyTorch

NVTX for MXNet

*Nsight Systems and Nsight Compute have been built using CUDA Profiling Tools Interface (CUPTI). They rely on NVTX markers to focus on sections of code.

*NVTX Nvidia Tools Extension Library is a way to annotate source code with markers.

*DLProf calls Nsight systems to collect the profile data and correlate with the graph.
DEEP LEARNING PROFILER (DLPROF)

New Profiling Tool for Data Scientists and Deep learning Researchers

Target user:
Data scientist/deep learning researcher

Solution:
Tool to understand and visualize DNN GPU utilization and timeline correlated with graph, which operations ran for how long and where optimization is possible
Goal: Easily understand & locate model operations that could be optimized especially to use or validate mixed precision use

Value Prop:
Two easy steps, no additional installs
Visualization in an already familiar TensorBoard as well as text reports
$ dlprof --reports=summary,detail,iteration /usr/bin/python main.py --mode=train --iter_unit=batch --num_iter=100 --batch_size=128 --warmup_steps=10 --use_cosine_lr --label_smoothing=0.1 --lr_init=0.256 --lr_warmup_epochs=1 --momentum=0.875 --weight_decay=3.0517578125e-05 --use_tf_amp --data_dir=/data/train-image --results_dir=./results

Prefix training script with dlprof

Visualize with Tensorboard or text reports

Graphdef file generated in Tensorflow

Use Nsight tools to gather kernel and timing profile data

Correlate profile data with Tensorflow model

Generate TensorBoard event files and detailed reports

Analyze in TensorBoard or other 3rd party tools
Use the TensorFlow Container

Generate graphdef and run dlprof

Collection of timing data occurs leveraging Nsight Systems behind the scenes

Visualize on TensorBoard

- Check Model summary report to understand how much time was spent in operations that were eligible to use Tensor cores but did not, what was the performance like in each iteration
- Check iterations report to understand each iteration of the training run
- Read top 10 ops that were consuming the most GPU and CPU time
- Switch to csv reports and back to TensorBoard as needed

Find clues to where optimization may be possible
DEEP LEARNING PROFILER

TensorBoard Support  ●  GPU Summary  ●  Support AMP and TF-TRT

Visualization with TensorBoard
GPU Lens on TensorBoard, Tensor Core compatibility

GPU Summary
Top 10 Ops, Duration, Iteration Report, Node Report

Support AMP and TF-TRT
Understand AMP in Training, TF-TRT in Inference

Available in the NGC TensorFlow container
https://ngc.nvidia.com/catalog/containers/nvidia:tensorflow
DLPROF ON TRAINING WITH AMP
MASSIVE PARALLEL ACCELERATION

Volta and Turing GPUs

Volta V100 GPU
640 Tensor Cores
5,120 CUDA cores
640 NEW Tensor cores
7.8 FP64 TFLOPS | 15.7 FP32 TFLOPS
| 125 Tensor TFLOPS
20MB SM RF | 16MB Cache
32 GB HBM2 @ 900GB/s | 300GB/s NVLink

Turing T4 GPU
320 Turing Tensor Cores
2,560 CUDA Cores
65 FP16 TFLOPS | 130 INT8 TOPS | 260 INT4 TOPS
16GB | 320GB/s
70 W
AUTOMATIC MIXED PRECISION
Easy to Use, Greater Performance and Boost in Productivity

Insert ~ two lines of code to introduce Automatic Mixed-Precision and get up to 3X speedup

AMP uses a graph optimization technique to determine FP16 and FP32 operations
Support for TensorFlow, PyTorch and MXNet

Unleash the next generation AI performance and get faster to the market!
## ENABLING AUTOMATIC MIXED PRECISION

Add Just A Few Lines of Code, Get up to 3X Speedup

### TensorFlow

```python
os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'
OR
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

NVIDIA Container 19.07+ and TF 1.14+, explicit optimizer wrapper available:

```python
opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)
```

### PyTorch

```python
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
```

### MXNet

```python
amp.init()
amp.init_trainer(trainer)
with amp.scale_loss(loss, trainer) as scaled_loss:
    autograd.backward(scaled_loss)
```

VISUALIZATION

NVIDIA Modifications to TensorBoard to Reflect GPU Details

- Puts a **GPU Lens** on TensorBoard
- Tensor Core compatibility showing
  - Eligible operations that could use Tensor cores
- Drill down to examine deeper on the green colored nodes
  - Ops that used Tensor cores
  - Ops that did not

Visualization inside your favorite tool.

(Note: DLProf version of TensorBoard is included in the container in addition to the existing one)
GPU Summary tab in TensorBoard

Top 10 operations

Duration: All of the operations in your code with respect to time, sortable, filterable and whether Tensor cores were used

Iterations report: Kernels executed and time used per iterations

Nodes report: Kernels executed and time used per iterations per Nodes

Model Summary: GPU time summary for all iterations
BEFORE & AFTER

No Tensor core info

Tensor core candidates and usage details per node operation
### GENERATING OTHER REPORTS BESIDES TENSORBOARD

DLProf Can Generate Reports in CSV and Json formats if Specified

For example, `dlprof --reports=detail --file Formats=csv`

<table>
<thead>
<tr>
<th>Name</th>
<th>Node Op</th>
<th>Origin</th>
<th>No. C.</th>
<th>TC Elig.</th>
<th>Using T</th>
<th>Total CI</th>
<th>Avg. CP</th>
<th>Min CP</th>
<th>Max CP</th>
<th>Total GPU Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>resnet50_v1.5/conv2d/conv2d</td>
<td>GraphDef</td>
<td>103</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>2.72E+09</td>
<td>26441154</td>
<td>151463</td>
<td>2.7E+09</td>
<td>3581490134</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/conv2d</td>
<td>GraphDef</td>
<td>103</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>80321528</td>
<td>779820.7</td>
<td>144240</td>
<td>819686</td>
<td>2674308744</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/ma</td>
<td>GraphDef</td>
<td>103</td>
<td>no</td>
<td>no</td>
<td></td>
<td>9893090</td>
<td>96049.4</td>
<td>51955</td>
<td>120934</td>
<td>667652739</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlr</td>
<td>GraphDef</td>
<td>103</td>
<td>no</td>
<td>no</td>
<td></td>
<td>24970708</td>
<td>242434.1</td>
<td>101461</td>
<td>275133</td>
<td>345946313</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlrBlock</td>
<td>GraphDef</td>
<td>103</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>12356031</td>
<td>119961.5</td>
<td>98613</td>
<td>211255</td>
<td>279960455</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlrBlock</td>
<td>GraphDef</td>
<td>103</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>26610762</td>
<td>258356.9</td>
<td>142673</td>
<td>288300</td>
<td>274993113</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlrBlock</td>
<td>GraphDef</td>
<td>103</td>
<td>no</td>
<td>no</td>
<td></td>
<td>49759885</td>
<td>483105.7</td>
<td>102745</td>
<td>516828</td>
<td>260050878</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlrBlock</td>
<td>GraphDef</td>
<td>103</td>
<td>no</td>
<td>no</td>
<td></td>
<td>7309938</td>
<td>70970.3</td>
<td>52081</td>
<td>90557</td>
<td>248203511</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlrBlock</td>
<td>GraphDef</td>
<td>103</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>60398150</td>
<td>586389.8</td>
<td>120131</td>
<td>630714</td>
<td>242249977</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlrBlock</td>
<td>GraphDef</td>
<td>103</td>
<td>no</td>
<td>no</td>
<td></td>
<td>61872633</td>
<td>600705.2</td>
<td>113473</td>
<td>749725</td>
<td>237655066</td>
</tr>
<tr>
<td>gradients/resnet50_v1.5/btlrBlock</td>
<td>GraphDef</td>
<td>103</td>
<td>no</td>
<td>no</td>
<td></td>
<td>12573131</td>
<td>122069.2</td>
<td>105913</td>
<td>156985</td>
<td>225558285</td>
</tr>
</tbody>
</table>
EXAMPLE: DLPROF WITH & WITHOUT AMP

Impact of Enabling Automatic Mixed Precision

Command for training run of MobileNetV2 with FP32 default

$ dlprof --reports=summary,detail,iteration --key_node=tower_0/v/add --iter_start=11 /usr/bin/python tf_cnn_benchmarks.py --num_gpus=1 --batch_size=1024 --model=mobilenet --device=gpu --gpu_indices=1 --data_name=imagenet --data_dir=/data --num_batches=50

Turn on AMP mode:

$ export TF_ENABLE_AUTO_MIXED_PRECISION=1

Run the same command:

$ dlprof --reports=summary,detail,iteration --key_node=tower_0/v/add --iter_start=11 /usr/bin/python tf_cnn_benchmarks.py --num_gpus=1 --batch_size=1024 --model=mobilenet --device=gpu --gpu_indices=1 --data_name=imagenet --data_dir=/data --num_batches=50
VISUALIZATION

Before using FP32

After using AMP
## COMPARING RESULTS

### Model Summary

<table>
<thead>
<tr>
<th></th>
<th>GPU Time</th>
<th>#Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FP32</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All Nodes</td>
<td>4.6 s</td>
<td>2082</td>
</tr>
<tr>
<td>Nodes Using TC</td>
<td>0 µs</td>
<td>0</td>
</tr>
<tr>
<td>Nodes Eligible For TC, But Not Using</td>
<td>2.16 s</td>
<td>110</td>
</tr>
<tr>
<td>All Other Nodes</td>
<td>2.54 s</td>
<td>1972</td>
</tr>
<tr>
<td><strong>AMP ON</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All Nodes</td>
<td>3.81 s</td>
<td>2924</td>
</tr>
<tr>
<td>Nodes Using TC</td>
<td>756 ms</td>
<td>105</td>
</tr>
<tr>
<td>Nodes Eligible For TC, But Not Using</td>
<td>61.1 ms</td>
<td>6</td>
</tr>
<tr>
<td>All Other Nodes</td>
<td>3.13 s</td>
<td>2813</td>
</tr>
</tbody>
</table>

Number of Found Iterations: 50
## TOP 10 NODES

<table>
<thead>
<tr>
<th>GPU Time (µs)</th>
<th>CPU Time (µs)</th>
<th>Op Name</th>
<th>Op Type</th>
<th>Origin</th>
<th>Calls</th>
<th>TC Eligible</th>
<th>Using TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>300537</td>
<td>10781</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/Conv2DBackpropFilter</td>
<td>Conv2DBackpropFilter</td>
<td>GraphDef</td>
<td>50</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>259489</td>
<td>44517</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/Conv2D</td>
<td>Conv2D</td>
<td>GraphDef</td>
<td>49</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>107338</td>
<td>9770</td>
<td>tower_0/v/transpose</td>
<td>Transpose</td>
<td>GraphDef</td>
<td>49</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>98710</td>
<td>1073621</td>
<td>MultiDeviceIteratorGetNextFromShard</td>
<td>MultiDeviceIteratorGetNextFromShard</td>
<td>Optimized</td>
<td>40</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>73550</td>
<td>4944</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/Conv2D-0-Transpose</td>
<td>Transpose</td>
<td>GraphDef</td>
<td>49</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>68149</td>
<td>11874</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/Conv2D</td>
<td>Conv2DBackpropFilter</td>
<td>GraphDef</td>
<td>50</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>67795</td>
<td>9749</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/expan</td>
<td>Conv2DBackpropFilter</td>
<td>GraphDef</td>
<td>50</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>67767</td>
<td>7563</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/expan</td>
<td>Conv2DBackpropFilter</td>
<td>GraphDef</td>
<td>50</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>64576</td>
<td>5301</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/expan</td>
<td>DepthwiseConv2dNative8</td>
<td>GraphDef</td>
<td>50</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>58088</td>
<td>4906</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/expan</td>
<td>DepthwiseConv2dNative8</td>
<td>GraphDef</td>
<td>50</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

### Before AMP

### With AMP

<table>
<thead>
<tr>
<th>GPU Time (µs)</th>
<th>CPU Time (µs)</th>
<th>Op Name</th>
<th>Op Type</th>
<th>Origin</th>
<th>Calls</th>
<th>TC Eligible</th>
<th>Using TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>310325</td>
<td>231252</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/expan</td>
<td>DepthwiseConv2dNative8</td>
<td>GraphDef</td>
<td>50</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>195665</td>
<td>13826</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/Conv2DBackpropFilter</td>
<td>Conv2DBackpropFilter</td>
<td>GraphDef</td>
<td>50</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>184808</td>
<td>51753</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/Conv2D</td>
<td>Conv2D</td>
<td>GraphDef</td>
<td>49</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>139014</td>
<td>1330803</td>
<td>MultiDeviceIteratorGetNextFromShard</td>
<td>MultiDeviceIteratorGetNextFromShard</td>
<td>Optimized</td>
<td>43</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>100537</td>
<td>13325</td>
<td>tower_0/v/transpose</td>
<td>Transpose</td>
<td>GraphDef</td>
<td>49</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>78434</td>
<td>6748</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/expan</td>
<td>DepthwiseConv2dNative8</td>
<td>GraphDef</td>
<td>50</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>66449</td>
<td>6206</td>
<td>input_processing/iteratorGetNext-0-CastToFp16-AutoV</td>
<td>Cast</td>
<td>GraphDef</td>
<td>49</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>64571</td>
<td>10363</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/expan</td>
<td>DepthwiseConv2dNative8</td>
<td>GraphDef</td>
<td>50</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>55115</td>
<td>13455</td>
<td>tower_0/v/cg/MobilenetV2/expanded_conv_1/depthwise</td>
<td>DepthwiseConv2dNative</td>
<td>GraphDef</td>
<td>49</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>42621</td>
<td>4063</td>
<td>tower_0/v/grads/tower_0/v/cg/MobilenetV2/Logits AvgPoolGrad</td>
<td>GraphDef</td>
<td>50</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE: FURTHER INVESTIGATION

Profiling MobileNetV2

1. Visualize Tensorboard and locate the ops not using Tensor cores
2. Check model summary saw that 6 nodes were eligible but not using TCs
3. 16 nodes accounted for 61.1 miliseconds out of total GPU time of 3.81 s so about 1.6%
4. Checking the timing info for Top 10 operations shows among the Top10, the nodes eligible were already using TCs
5. Open up the detailed report csv file to filter those 6 nodes that were eligible and not using tensor cores
6. Drill down on the top 2 ops; they show following data:

<table>
<thead>
<tr>
<th>Name</th>
<th>Node Op</th>
<th>Origin</th>
<th>No. Calls</th>
<th>TC Eligible</th>
<th>Using TC</th>
<th>Total CPU Time (ns)</th>
<th>Avg. CPU Time (ns)</th>
<th>Min CPU Time (ns)</th>
<th>Max CPU Time (ns)</th>
<th>Total GPU Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tower_0/v/cg/MobilenetV2/Logits/Conv2d_1c_1x1/Conv2D</td>
<td>Conv2D</td>
<td>GraphDef</td>
<td>49</td>
<td>yes</td>
<td>no</td>
<td>5768334</td>
<td>117721.1</td>
<td>80603</td>
<td>383079</td>
<td>12081912</td>
</tr>
<tr>
<td>tower_0/v/gradients/tower_0/v/cg/MobilenetV2/Logits/Conv2d_1c_1x1/Conv2D_grad/Conv2DBackpropInput</td>
<td>Conv2DBackpropInput</td>
<td>GraphDef</td>
<td>50</td>
<td>yes</td>
<td>no</td>
<td>10183849</td>
<td>203677</td>
<td>57169</td>
<td>5048786</td>
<td>11520894</td>
</tr>
</tbody>
</table>
EXAMPLE CONTINUED

Visualise the Interesting Node in Tensorboard

Copy and paste the op node in TensorBoard: “tower_0/v/cg/MobilenetV2/Logits/Conv2d_1c_1x1/Conv2D”

- Check the input and output types
- There may be an operation that is not multiple of 8
- Get to the python code that was contributing to any slowness in training
- Workflow ends at this point, it is up to user to modify their python code
TENSOR CORE PERFORMANCE GUIDELINES

- Satisfy requirements to enable Tensor Cores
  - For linear layers: input size, output size, batch size need to be multiples of 8 (FP16) / 16 (INT8)
  - For convolutions: input and output channel counts need to be multiples of 8 (FP16) /16 (INT8)

- Ensure good Tensor Core GEMM efficiency
  - Choose the above dimensions as multiples of 64/128/256
  - (if the total number of tiles is small) Ensure that the tile count is a multiple of the SM count

- Be aware of bandwidth limited regimes
  - If any GEMM dimension is 128 or smaller, the operation is likely bandwidth limited
DLPROF ON INFERENCE WITH TF-TRT
TF-TRT API IN TENSORFLOW > 1.13

contrib → compiler

from tensorflow.python.compiler.tensorrt import trt

converter = trt.TrtGraphConverter(input_saved_model_dir=input_saved_model_dir)
converter.convert()
converter.save(output_saved_model_dir)

from tensorflow.python.compiler.tensorrt import trt

converter = trt.TrtGraphConverter(
    input_graph_def=frozen_graph,
    nodes_blacklist=['logits', 'classes'])
frozen_graph = converter.convert()
LEGACY TF-TRT API IN TENSORFLOW <=1.13

One API Call Returns a TF-TRT Optimized graph

Import from SavedModel

Import tensorflow.contrib.tensorrt as trt

trt.create_inference_graph(
    input_saved_model_dir=input_saved_model_dir,
    output_saved_model_dir=output_saved_model_dir)

Import from Frozen Graph

Import tensorflow.contrib.tensorrt as trt

converted_graph_def = trt.create_inference_graph(
    input_graph_def = frozen_graph,
    outputs=['logits', 'classes'])

The original Python function create_inference_graph that was used in TensorFlow 1.13 and earlier is deprecated and planned to be removed to TensorFlow 2.0.
HOW TF-TRT WORKS

Under the hood

1. Phase 1: graph partition
   - Partition the TF Graph: TRT-compatible vs. TRT-incompatible
   - Wrap each TRT-compatible subgraph in a single node (TRTEngineOp)
   - Use the new node to replace the subgraph

2. Phase 2: layer conversion
   - For each new node, build a TensorRT network (a graph containing TensorRT layers)

3. Phase 3: engine optimization
   - Optimize the network and use it to build a TensorRT engine

TRT-incompatible subgraphs remain untouched and are handled by TF runtime do the inference with TF interface
Deserializing frozen graph → Convert with TF-TRT → Run inference and do profiling

PROFILING WORKFLOWS
import tensorflow as tf
import numpy as np
from tensorflow.python.compiler.tensorrt import trt_convert as trt

with tf.Session() as sess:
    with tf.gfile.GFile("/path/to/your/frozen/graph.pb", 'rb') as f:
        frozen_graph = tf.GraphDef()
        frozen_graph.ParseFromString(f.read())

    converter = trt.TrtGraphConverter(input_graph_def=frozen_graph, nodes_blacklist=['logits', 'classes'])
    trt_graph = converter.convert()

    features = np.random.normal(loc=112, scale=70, size=(8, 3, 224, 224)).astype(np.float32)
    features = np.clip(features, 0.0, 255.0)

    new_graph = tf.import_graph_def(trt_graph, input_map={'input': features}, return_elements=['logits', 'classes'])
    for i in range(0, 200):
        sess.run(new_graph)

dlprof --key_node= import/resnet_v1_50/TRTEngineOp_0 --iter_start 51 --iter_stop 200 python inference.py
FP16 EXAMPLE WITH RESNET50 V1.5

Reading CUDA API calls from Nsight Systems database

[WARNING] Detected TensorCore enabled kernel called from a non TensorCore compatible node:

  Node: import/resnet_v1_50/TRTEngineOp_0
  Node Op: TRTEngineOp
  Node Orig: 0
  Kernel: trt_turing_h1688cudnn_256x64_ldg8_relu_exp_interior_nhwc_tn_v1
INT8 EXAMPLE WITH RESNET50 V1.5

Reading CUDA API calls from Nsight Systems database

[WARNING] Detected TensorCore enabled kernel called from a non TensorCore compatible node:
  - Node: import/resnet_v1_50/TRTEngineOp_0
  - Node Op: TRTEngineOp
  - Node Orig: 0
  - Kernel: trt_turing_int8_i8816cudnn_int8_256x64_ldg16_relu_singleBuffer_interior_nt_v1
NVIDIA TOOLS EXTENSION LIBRARY (NVTX)

NVTX is a platform agnostic, tools agnostic API

Allows developers to annotate(mark) source code, events, code ranges etc.

NVIDIA optimized TensorFlow, PyTorch, MXnet have NVTX annotations built in

Easy to use plugin for TensorFlow is also available:

https://github.com/NVIDIA/nvtx-plugins/

*Nsight Systems, Nsight Compute and Deep Learning Profiler make use of NVTX markers

https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvtx
Nvidia framework containers on NGC already have NVTX markers which are used by DLProf.

For easily defining your own markers, use this library in addition

- Library developed specifically for annotating python code to help visualize network better in Nsight Systems

- Workflow:
  - Import nvtx_tf library
  - Annotate python code
  - Run TensorFlow
  - Get data through a profiler such as Nsight Systems
PYPROF

Deep Learning Profiler Will be Leveraging for PyTorch Support

Library for effectively using NVTX marker for PyTorch
Custom NVTX marker as a python dictionary with module name, function name, arguments (tensor shapes & type, scalar type & value).

Workflow:
• Import library
• Annotate python code
• Run with profiler

import torch.cuda.profiler as profiler
from apex import pyprof
pyprof.nvtx.init()

https://github.com/NVIDIA/apex/tree/master/apex/pyprof
GETTING STARTED: DEEP LEARNING PROFILER

- Try DLProf alpha included in Optimized Tensorflow container on NGC starting June
  docker pull nvcr.io/nvidia/tensorflow:19.11-py3

- Have graphdef definitions of your models ready for input

- Prefix training script with dlprof command and get the profile output

- Run a few iterations and smaller batch size to collect fast profile data of your entire model

Note: There are two versions of Tensorboard included in the docker container
DLProf version of Tensorboard is only used if user wishes to.