Inference Optimization Using TensorRT with Use Cases

Jack Han / 한재근 | Solutions Architect | NVIDIA
TensorRT 4 Adoption

Use Cases

- Video
- Maps
- Image
- NLP
- Search
- Speech

AI CONFERENCE
AI Inference is exploding

1 Billion
Videos Watched Per Day
Facebook

LIVE VIDEO

1 Billion
Voice Searches Per Day
Google, Bing, etc.

SPEECH

1 Trillion
Ads/Rankings Per Day
Impressions

RECOMMENDATIONS
Bigger and More Compute Intensive in quest for accuracy

- **Image** (GOP * Bandwidth)
  - AlexNet
  - ResNet-50
  - GoogleNet
  - Inception-v2
  - Inception-v4
  - 350X

- **Speech** (GOP * Bandwidth)
  - DeepSpeech
  - DeepSpeech 2
  - DeepSpeech 3
  - 30X

- **Translation** (GOP * Bandwidth)
  - MoE
  - OpenNMT
  - GNMT
  - 10X
Cambrian Explosion

Convolutional Networks
- Encoder/Decoder
- ReLu
- BatchNorm
- Concat
- Dropout
- Pooling

Recurrent Networks
- LSTM
- GRU
- Beam Search
- WaveNet
- CTC
- Attention

Generative Adversarial Networks
- 3D-GAN
- MedGAN
- Conditional GAN
- Coupled GAN
- Speech Enhancement GAN

Reinforcement Learning
- DQN
- Simulation
- DDPG

New Species
- Mixture of Experts
- Neural Collaborative Filtering
- Block Sparse LSTM
- Capsule Nets
Inefficiency Limits Innovation

Single Model Only

Some systems are overused while others are underutilized

Single Framework Only

Solutions can only support models from one framework

Custom Development

Developers need to reinvent the plumbing for every application
More Accuracy = Time

Speed/accuracy trade-offs for modern convolutional object detectors, Google Research
Inference Performance

- Latency
- Throughput
- Efficiency
- Easy Deploy
## Inference Optimization

<table>
<thead>
<tr>
<th>Lightweight Model</th>
<th>Deep Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Computation</td>
<td>No Network Modification</td>
</tr>
<tr>
<td>Suitable Lightweight Hardware</td>
<td>Continuous the Enovation</td>
</tr>
<tr>
<td>E.g; MobileNet</td>
<td>E.g; Quantization and Binarization</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>Network Pruning and Sharing</td>
</tr>
<tr>
<td>SEP-Nets</td>
<td>Distillation / Factorization</td>
</tr>
</tbody>
</table>
TensorRT
From framework to target

FRAMEWORKS
- Caffe2
- PaddlePaddle
- Chainer
- PyTorch
- TensorFlow
- mxnet
- Theano

GPU PLATFORMS
- TESLA T4
- JETSON TX2
- DRIVE PX 2
- NVIDIA DLA
- TESLA V100
TensorRT 5

More Layers / Plugin / APIs
Inference Server Integration
TensorRT Workflow

Step 1. Optimize trained model

Model → Optimize → Plan
TensorRT Workflow

Step 2. Deploy optimized plan

Plan → Runtime Engine → Data center/Embedded/Automotive
Step-by-Step TensorRT Plan Build

https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html#support_op
### 7 Steps to Deployment with TensorRT

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Convert trained model into TensorRT format</td>
<td><code>uff_model = uff.from_tensorflow_frozen_model(&quot;frozen_model_file.pb&quot;, OUTPUT_LAYERS)</code></td>
</tr>
<tr>
<td>2</td>
<td>Create a model parser</td>
<td><code>parser = uffparser.create_uff_parser()</code></td>
</tr>
<tr>
<td>3</td>
<td>Register inputs and outputs</td>
<td><code>parser.register_input(INPUT_LAYERS[0], (INPUT_C,INPUT_H,INPUT_W),0)</code> <code>parser.register_output(OUTPUT_LAYERS[0])</code></td>
</tr>
<tr>
<td>4</td>
<td>Optimize model and create a runtime engine</td>
<td><code>engine = trt.utils.uff_to_trt_engine(G_LOGGER, </code>uff_model, <code>parser, INPUT_LAYERS[0], </code>INSTRUCTION_BATCH_SIZE, 1&lt;&lt;20, trt.infer.DataType.FLOAT)`</td>
</tr>
<tr>
<td>5</td>
<td>Serialize optimized engine</td>
<td><code>trt.utils.write_engine_to_file(save_path, engine.serialize())</code></td>
</tr>
<tr>
<td>6</td>
<td>De-serialize engine</td>
<td><code>engine = Engine(PLAN=plan, postprocessors=\{&quot;output_layer_name&quot;:post_processing_function\})</code></td>
</tr>
<tr>
<td>7</td>
<td>Perform inference</td>
<td><code>result = engine_single.infer(image)</code></td>
</tr>
</tbody>
</table>
Custom Layer Support using Plugin Layer
Plugin Layer Procedure

Phase 1. Initialize

Phase 2. Inference

Phase 3. Store/Load

Layer?

Initialize

enqueue

Serialize

Deserialize
### Supporting Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>C++</th>
<th>Python (x86 only)</th>
<th>NvCaffeParser</th>
<th>NvUffParser</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNNs</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RNNs</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>INT8 Calibration</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Asymmetric Padding</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Define your network using C/C++ or Python

```cpp
INetworkDefinition* network = builder->createNetwork();

auto data = network->addInput(INPUT_BLOB_NAME, DataType::kFLOAT,Dims);
auto rnn = network->addRNNv2(*data, LAYER_COUNT, HIDDEN_SIZE, SEQ_SIZE);

convertRNNWeights, convertRNNBias
setWeightsForGate, setBiasForGate

auto fcwts = network->addConstant(Dims2(VOCAB_SIZE, HIDDEN_SIZE), weights);
auto matrixMultiply = network->addMatrixMultiply(*fcwts->getOutput(0),
auto fc biases = network->addConstant(Dims2(VOCAB_SIZE, 1), weightMap[FC],
auto addBiasLayer = network->addElementWise(*matrixMultiply->getOutput(),
auto pred = network->addTopK(*addBiasLayer->getOutput(0), nvinfer1::T
t network->markOutput(*pred->getOutput(1));
```
TensorRT RNN Programming

Dump Weights → Load Weights → Convert Weights → Set Weights

/usr/src/tensorrt/samples/common/dumpTFWts.py

ckpt

wts

TF format

TensorRT RNN Layer

Get started with sampleCharRNN in documentation
NMT with Transformer Inference

Deploy highly-optimized language translation apps in production environments

Running 3 binded engines (Encoder, Generator, Suffle Engine)

Modular Network Merge

Support NMT layers such as Gather, Softmax, Batch GEMM and Top K

Get started with [NMT sample](#) in documentation

Runs on CPU
GPU-Accelerated

* CPU-Only: TensorFlow on SKL 6140 18 core, FP32
  GPU: V100, TensorRT 5, FP16; Sorted data, Batch=128, English to German
Recommendation Inference

Deploy multi-layer perceptron (MLP) based recommendation apps in production

Predict events (click, purchase, interest) accurately based on input items (user, query, observed activity)

Get started with examples on MNIST character recognition and movie recommender system in the documentation

* CPU-Only: TensorFlow on Intel Xeon E5-2698 v4 CPU at 2.20 GHz; GPU: V100 with custom networks
Inferencing with Low precision

I8_weight = Round_to_nearest_int(scaling_factor * F32_weight)
scaling_factor = 127.0f / max( abs(all_F32_weights_in_the_filter) )
Post Training Quantization

Minimize information loss between FP32 and INT8 inference on calibration dataset

100 mini-batches are sufficient to determine quantization parameter (~500 samples)
New INT8 APIs and Optimizations

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

Getting Start with [INT8 sample](#) in the documentation

**Auto Calibration**
- FP32 weights
- Optimize to INT8
- Calibration Data O(1000) Images

**Custom Calibration**
- FP32 weights
- Custom Calibration
- FP32 or INT8 weights
- Optimize to INT8
- Custom Activation ranges

**Quantization Aware Training**
- Custom Activation Ranges
- Optimize to INT8
Inference Performance

**Throughput**

- P4 - INT8
- V100 - Mixed

**Performance/WATT**

- P4 - INT8
- V100 - Mixed

*Higher is better!!*
WIDELY ADOPTED
NVIDIA INFERENC MOMENTUM

Image Tagging
Video Analysis
Advertising Impact
Video Captioning
Cybersecurity
Visual Search
Finding Music
Sports Performance
Customer Service
Visual Search
Industrial Inspection
Voice Recognition
“Using GPUs made it possible to enable media understanding of the Twitter platform, by drastically reducing media deep learning model training time and by allowing us to derive real-time understanding of live videos at inference time.”

Nicolas Koumchatzky, Head of Twitter Cortex
“At Microsoft, we are working hard to deliver the most advanced AI-powered services to our customers. Using NVIDIA GPUs in real-time inference workloads has improved Bing’s advanced search offerings, enabling us to reduce object detection latency for images. We look forward to working with NVIDIA’s next generation inference hardware and software to expand the way people benefit from AI products and services.”

-Jordi Ribas CVP, Bing and Al Products, Microsoft

“AI is becoming increasingly pervasive, and inference is a critical capability customers need to successfully deploy their AI models, so we’re excited to support NVIDIA’s Turing Tesla T4 GPUs on Google Cloud Platform soon.”

-Chris Kleban, product manager at Google Cloud
AI CONFERENCE

SEOUL | NOVEMBER 7 - 8, 2018

www.nvidia.com/ko-kr/ai-conference/