53021 - DEPLOYMENT OF SEMANTIC SEGMENTATION NETWORK USING TENSORRT

Joohoon Lee
Chethan Ningaraju
OUTLINE

Basic background information and hands-on sessions

Semantic segmentation for automotive use case
Cityscapes dataset
Pre-trained sample network - FCN variant
Inference performance on DrivePX2 using Caffe and CUDNN
Introduction to TensorRT
FP32 Deployment using TensorRT
INT8 Deployment using TensorRT
“Semantic segmentation is the task of clustering parts of images together which belong to the same object class”

Martin Thoma - A Survey of Semantic Segmentation
SEGMENTATION FOR AUTOMOTIVE USE CASE

OpenRoadNet from NVIDIA
SEGMENTATION FOR AUTOMOTIVE USE CASE

OpenRoadNet from NVIDIA
CITYSCAPES DATASET

https://www.cityscapes-dataset.com/

CITYSCAPES DATASET

19 CLASS

| road      | person            |
| building  | rider             |
| wall      | car               |
| fence     | truck             |
| pole      | bus               |
| traffic light | train         |
| traffic sign | motorcycle    |
| vegetation | bicycle         |
| terrain   | sidewalk          |
CITYSCAPES DATASET

19 CLASS

- road
- building
- wall
- fence
- pole
- traffic light
- traffic sign
- vegetation
- terrain
- sky
- person
- rider
- car
- truck
- bus
- train
- motorcycle
- bicycle
- sidewalk
EVALUATION METRIC

19 CLASS
- road
- building
- wall
- fence
- pole
- traffic light
- traffic sign
- vegetation
- terrain
- sky
- person
- rider
- car
- truck
- bus
- train
- motorcycle
- bicycle
- sidewalk

7 CATEGORY
- flat
- nature
- object
- sky
- construction
- human
- vehicle

PER PIXEL METRIC

\[
\text{IoU} = \frac{TP}{TP + FP + FN}
\]

- TP = True Positive
- FP = False Positive
- FN = False Negative

Average IoU class
Average IoU category
PRETRAINED SAMPLE NETWORK

FCN Variant

VGG16 based FCN with modification
Trained using Cityscapes train dataset
60000 iterations starting from VGG weights
Average IoU class = 48.4
Average IoU category = 76.9
INFERENC PERFORMANCE USING CAFFE

Performance measured using Caffe on DrivePX2 dGPU

<table>
<thead>
<tr>
<th></th>
<th>CAFFE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (ms)</td>
<td>242.2</td>
</tr>
<tr>
<td>Images/sec</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Batch Size = 1, Input/Output Resolution = 512 x 1024
TensorRT

High performance neural network inference engine for production deployment

Generate optimized and deployment ready models for datacenter, embedded and automotive platforms

Deliver high-performance, low-latency inference demanded by real-time services

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

developer.nvidia.com/tensorrt
Step 1: Optimize trained model

Training Framework → NEURAL NETWORK → TensorRT Optimizer → PLAN → Validation USING TensorRT

Batch Size
Precision

Serialize to disk

developer.nvidia.com/tensorrt
TensorRT

Step 2: Deploy optimized plans with runtime

Serialized PLAN

TensorRT runtime engine

developer.nvidia.com/tensorrt
FP32 DEPLOYMENT USING TensorRT
OUTLINE - FP32
What you will implement today

Use Caffe parser to load a pre-trained model
Create TensorRT engine for FP32
Serialize engine to plan file
Measure performance of inferencing using TensorRT with FP32
Test inference output and visually inspect
GETTING STARTED

Basic information

Lab files are located under /home/nvidia/GTC2017-53021

$ cd /home/nvidia/GTC2017-53021

Recommended text editor

$ gedit <filename> &

You need to do something

For your reference

Expected output on console
DIRECTORY
Files for the lab

data
→ Pre-trained Caffe model

sampleCityscapes
→ Step 1: Optimize trained model

sampleCityscapesInference
→ Step 2: Deploy optimized plans with runtime
LET’S WORK ON THE CODE TOGETHER
USE CAFFE PARSER TO LOAD A MODEL

sampleCityscapes.cpp - TODO #1

TODO #1 : Create a Caffe parser object by calling createCaffeParser() function

~Line 231

```cpp
IBuilder* builder = createInferBuilder(gLogger);
INetworkDefinition* network = builder->createNetwork();
ICaffeParser* parser = /* TODO */
```

```cpp
NvCaffeParser.h
{
    ICaffeParser* createCaffeParser();
}
```
CREATE TensorRT ENGINE FOR FP32

sampleCityscapes.cpp - TODO #2

TODO #2: Create optimized TensorRT engine by calling buildCudaEngine on the builder object

~Line 271

ICudaEngine* engine = /* TODO */

NvInfer.h

class Ibuilder {
    virtual nvinfer1::ICudaEngine* buildCudaEngine(nvinfer1::INetworkDefinition& network) = 0;
};
TODO #3: Serialize the engine to a plan file and save by calling serialize() on the engine

~Line 279

tensorRTModelStream = /* TODO */

NvInfer.h
class ICudaEngine{
  virtual IHostMemory* serialize() const = 0;
};
BUILD & TEST YOUR CODE

Output is optimized, serialized engine file

In the GTC2017-53021/sampleCityscapes/ directory, build the sample

```bash
$ cd /home/nvidia/GTC2017-L53021/sampleCityscapes

$ make
```

Run the sample_cityscapes program and check the output file

```bash
$ cd /home/nvidia/GTC2017-L53021/bin

$ ./sample_cityscapes

$ ls -alsh ../output
```
MEASURE INFERENCE PERFORMANCE IN FP32

sampleCityscapes.cpp - TODO #4

TODO #4: Timing routine has been written in comment.

Please uncomment.

~Line 331

```cpp
// std::cout << "Avg execution time over " << TIMING_ITERATIONS << " iteration is " << total/TIMING_ITERATIONS << " ms." << std::endl;
```
BUILD & TEST YOUR CODE

Measure average execution time

In the GTC2017-53021/sampleCityscapes/ directory, build the sample

```
$ cd /home/nvidia/GTC2017-53021/sampleCityscapes

$ make
```

Run the sample_cityscapes program to measure average execution time

```
$ cd /home/nvidia/GTC2017-53021/bin

$ ./sample_cityscapes
```

Avg execution time over 10 iterations is 170.756 ms.
MEASURE PERFORMANCE USING PROFILER

sampleCityscapes.cpp - TODO #5

TODO #5 : Set profiler for the context, to get per layer performance
~Line 365-366

IExecutionContext *context = engine->createExecutionContext();
context->setProfiler(&gProfiler);

TODO #6 : Call printLayerTimes() at the end
~Line 378-379

gProfiler.printLayerTimes();
BUILD & TEST YOUR CODE
Measure per layer execution time using profiler

Build and run sample_cityscapes again

<table>
<thead>
<tr>
<th>Layer</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1_1 + relu1_1</td>
<td>2.447ms</td>
</tr>
<tr>
<td>conv1_2 + relu1_2</td>
<td>11.816ms</td>
</tr>
<tr>
<td>pool1</td>
<td>2.625ms</td>
</tr>
<tr>
<td>conv2_1 + relu2_1</td>
<td>6.054ms</td>
</tr>
<tr>
<td>conv2_2 + relu2_2</td>
<td>11.784ms</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>upscore_pool4</td>
<td>0.082ms</td>
</tr>
<tr>
<td>score_pool3 + fuse_pool3</td>
<td>0.180ms</td>
</tr>
<tr>
<td>upscore8</td>
<td>2.485ms</td>
</tr>
</tbody>
</table>

Time over all layers: 170.756
PREPARE A TEST IMAGE

In the GTC2017-53021/scripts/ directory, run script

```
$ cd /home/nvidia/GTC2017-53021/scripts/
$ python batch_preprocessor.py test
```

Location of dataset = /home/nvidia/GTC2017-53021-
Data/Cityscapes/leftImg8bit/train/*//*.png
Processing batches for test
Total number of images = 2975
NUM_PER_BATCH = 1
NUM_BATCHES = 1
Adding image: aachen_000000_000019_leftImg8bit.png in batch_test0
GENERATE PREDICTION OUTPUT

In the GTC2017-53021/sampleCityscapesInference/ directory, build the sample

```bash
$ cd /home/nvidia/GTC2017-53021/sampleCityscapesInference/

$ make
```

Run the `sample_cityscapes_inference` program

```bash
$ cd /home/nvidia/GTC2017-53021/bin

$ ./sample_cityscapes_inference test
```

Saving output prediction to `./output/aachen_000000_000019_leftImg8bit_pred.png`
VISUALLY INSPECT THE PREDICTION

Open the prediction file using ‘display’ command

$ display ../output/aachen_000000_000019_leftImg8bit_pred.png
VISUALLY INSPECT THE PREDICTION

In the GTC2017-53021/scripts/ directory, run script

```bash
$ cd /home/nvidia/GTC2017-53021/scripts/

$ python display_color.py ../output/aachen_000000_000019_leftImg8bit_pred.png
```
SUMMARY OF FP32 DEPLOYMENT
Performance comparison against Caffe on DrivePX2 dGPU

<table>
<thead>
<tr>
<th></th>
<th>CAFFE</th>
<th>TENSORRT FP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (ms)</td>
<td>242.2</td>
<td>170.7</td>
</tr>
<tr>
<td>Images/sec</td>
<td>4.1</td>
<td>5.9</td>
</tr>
</tbody>
</table>

Batch Size = 1, Input/Output Resolution = 512 x 1024
INT8 DEPLOYMENT USING TensorRT
TensorRT

Step 1: Optimize trained model for INT8
TensorRT

Step 2: Deploy optimized plans with runtime

 Serialized PLAN

 TensorRT runtime engine

developer.nvidia.com/tensorrt
OUTLINE - INT8
What you will implement today

- Prepare calibration dataset for INT8 inferencing
- Create TensorRT engine for INT8 with entropy calibrator
- Measure performance of inferencing using TensorRT with INT8
- Validate the accuracy of INT8 model using Cityscapes validation dataset
Before we move to INT8, please perform the following first

$ cd /home/nvidia/GTC2017-53021/output

$ rm *.png
PREPARE CALIBRATION DATASET

In the GTC2017-53021/scripts/ directory, run script

```bash
$ cd /home/nvidia/GTC2017-53021/scripts/
$ python batch_preprocessor.py calibration
```

Location of dataset = /home/nvidia/GTC2017-53021-
Data/Cityscapes/leftImg8bit/train/*//*.png
Processing batches for calibration
Total number of images = 2975
NUM_PER_BATCH = 1
NUM_BATCHES = 50
TODO #7 : Please uncomment.

Line 257~261

```cpp
// TODO #7: Uncomment the below 4 lines
BatchStream calibrationStream(CAL_BATCH_SIZE, NB_CAL_BATCHES, 
"../batches/batch_calibration");
Int8EntropyCalibrator calibrator(calibrationStream, FIRST_CAL_BATCH);
builder->setInt8Mode(true);
builder->setInt8Calibrator(&calibrator);
```
BUILD & TEST YOUR CODE
Measure per layer execution time using profiler

Build and run sample_cityscapes again

<table>
<thead>
<tr>
<th>Layer Description</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1_1 + relu1_1 input reformatter 0</td>
<td>0.168ms</td>
</tr>
<tr>
<td>conv1_1 + relu1_1</td>
<td>1.013ms</td>
</tr>
<tr>
<td>conv1_2 + relu1_2</td>
<td>4.241ms</td>
</tr>
<tr>
<td>pool1</td>
<td>0.700ms</td>
</tr>
<tr>
<td>conv2_1 + relu2_1</td>
<td>2.066ms</td>
</tr>
<tr>
<td>conv2_2 + relu2_2</td>
<td>3.851ms</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>upscore_pool4</td>
<td>0.047ms</td>
</tr>
<tr>
<td>score_pool3 + fuse_pool3</td>
<td>0.066ms</td>
</tr>
<tr>
<td>upscore8</td>
<td>2.197ms</td>
</tr>
</tbody>
</table>

Time over all layers: 50.237
PREPARE VALIDATION DATASET

In the GTC2017-53021/scripts/ directory, run script

```
$ cd /home/nvidia/GTC2017-53021/scripts/

$ python batch_preprocessor.py validation
```

Location of dataset = /home/nvidia/GTC2017-53021-
Data/Cityscapes/leftImg8bit/val/*/*.png
Processing batches for validation
Total number of images = 500
NUM_PER_BATCH = 1
NUM_BATCHES = 500
In the GTC2017-53021/sampleCityscapesInference/ directory, build the sample

$ cd /home/nvidia/GTC2017-53021/sampleCityscapesInference/

$ make

Run the sample_cityscapes_inference program for all 500 validation images

$ cd /home/nvidia/GTC2017-53021/bin

$ ./sample_cityscapes_inference validation

Saving output prediction to
./output/frankfrut_000000_000294_leftImg8bit_pred.png
VALIDATE ACCURACY
scripts/eval_tensorrt_cityscapes.py

In the GTC2017-53021/scripts/ directory, run script

```
$ cd /home/nvidia/GTC2017-53021/scripts/

$ python eval_tensorrt_cityscapes.py
```

Evaluating 500 pairs of images...
Images processed: 500

<table>
<thead>
<tr>
<th>classes</th>
<th>IoU</th>
<th>nIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score Average</td>
<td>0.481</td>
<td>0.236</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>categories</th>
<th>IoU</th>
<th>nIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score Average</td>
<td>0.768</td>
<td>0.565</td>
</tr>
</tbody>
</table>
# SUMMARY OF INT8 DEPLOYMENT

Performance and IoU comparison against Caffe on DrivePX2 dGPU

<table>
<thead>
<tr>
<th></th>
<th>CAFFE</th>
<th>TENSORRT FP32</th>
<th>TENSORRT INT8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (ms)</td>
<td>242.2</td>
<td>170.7</td>
<td>50.2</td>
</tr>
<tr>
<td>Images/sec</td>
<td>4.1</td>
<td>5.9</td>
<td>19.9</td>
</tr>
<tr>
<td>Class IoU</td>
<td>48.4</td>
<td>48.4</td>
<td>48.1</td>
</tr>
<tr>
<td>Category IoU</td>
<td>76.9</td>
<td>76.9</td>
<td>76.8</td>
</tr>
</tbody>
</table>

Batch Size = 1, Input/Output Resolution = 512 x 1024