Fast Neural Network Inference with TensorRT on Autonomous Vehicles

Zejia Zheng (zheng@zoox.com)
Josh Park (josh@nvidia.com)
Jeff Pyke (jpyke@zoox.com)

Table of Contents

Background

Massive amount of computation in DNN

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

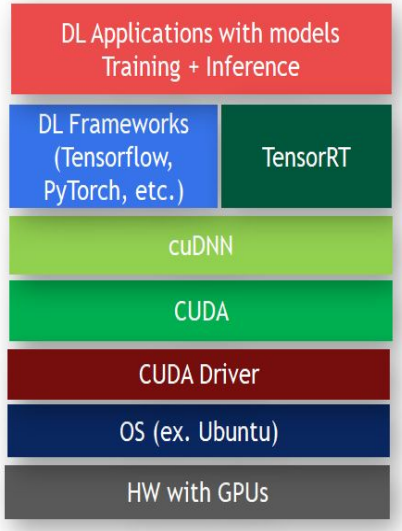
Parameter layers in billions FLOPS (mul/add)

[1] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

GPU: High Performance Computing Platform

	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.5 TFLOPS
Single-Precision Performance	14 TFLOPS	15 TFLOPS
Tensor Performance	112 TFLOPS	120 TFLOPS
GPU Memory	16 GB HBM2	
Memory Bandwidth	900 GB/sec	
ECC	Yes	
Interconnect Bandwidth*	32 GB/sec	300 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

SW Libraries



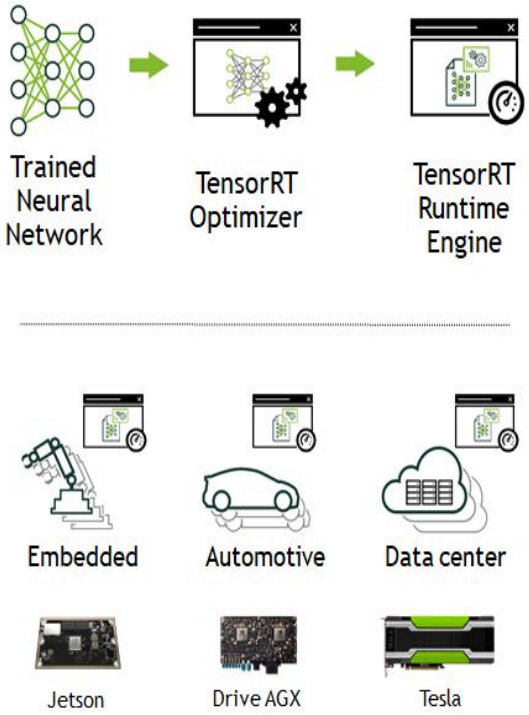
Nvidia TensorRT - Programmable Inference Accelerator

A sw platform for high-performance deep learning inference

TensorRT-based applications perform up to 40x faster than CPU-only platforms during inference

Deploy to hyperscale data centers, embedded, or **automotive** product platforms.

Speed up recommender, speech, video and translation in production

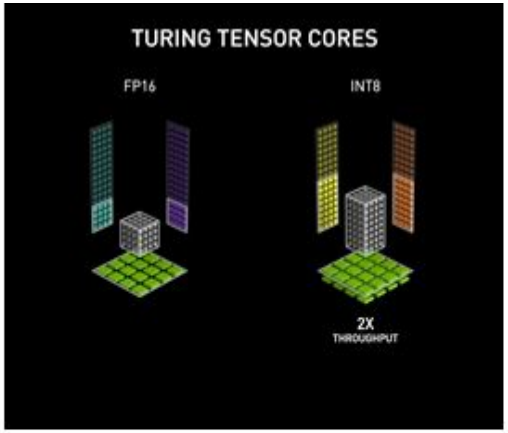


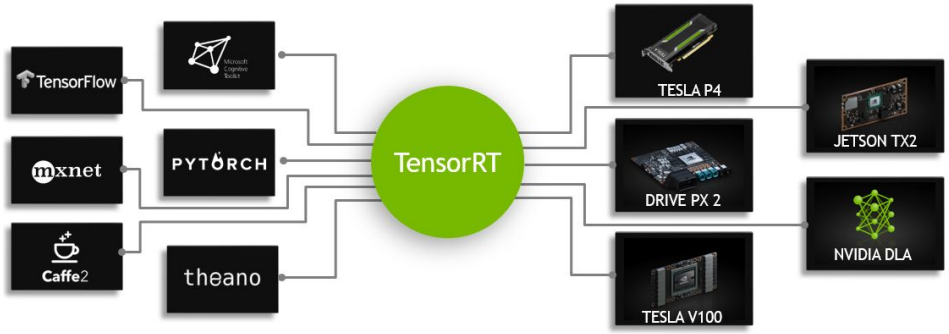
TensorRT 5 support Turing GPUs

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

Control precision per-layer with new APIs

Optimizations for depth-wise convolution operation





From Every Framework, Optimized For Each Target Platform

What TensorRT Does

Layer & Tensor Fusion:

Fuse several layers/ops into one layer

Auto-Tuning:

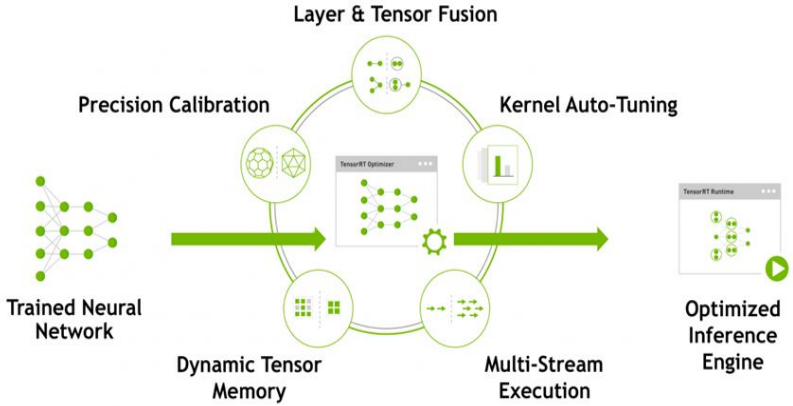
Platform specific kernels to maximize performance

Multi-Stream Execution:

Execute CUDA streams for independent batch/inference

Dynamic Tensor Memory:

Reuse activation from already used layers

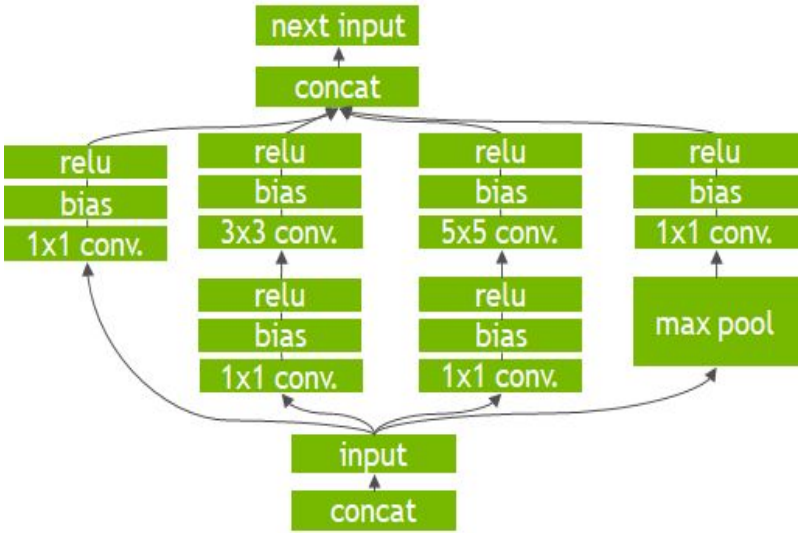


Precision Calibration:

Calibrate computations on lower precision (FP16/INT8) tensor operations

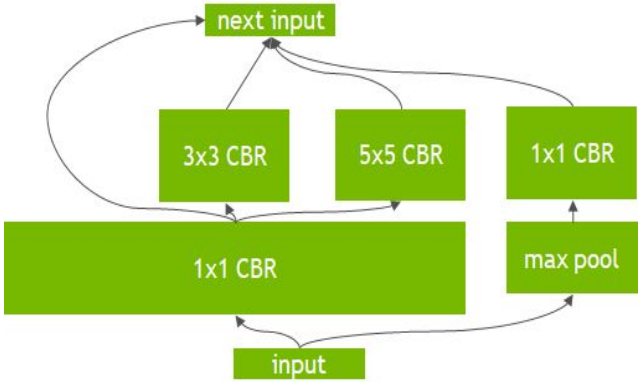
Layer & Tensor Fusion

Unoptimized Network



e.g

TensorRT Optimized Network



Networks	Number of layers (Before)	Number of layers (After)
VGG19	43	27
Inception v3	309	113
ResNet-152	670	159

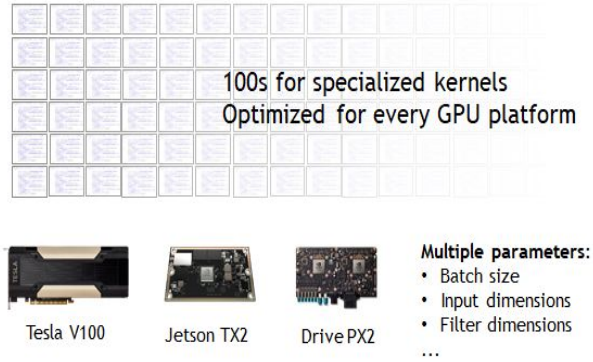
Kernel Auto-Tuning

Maximize kernel performance

Select the best performance for target GPU

Parameters

- Input data size,
- Batch,
- Tensor layout,
- Input dimension,
- Memory,
- Etc.




```
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Relu(14)
[V] [TRT] Tactic 5144900180810042977 time 0.209112
[V] [TRT] Tactic 7995089803833173579 time 0.147572
[V] [TRT] Tactic -9018593586369639207 time 0.229336
[V] [TRT] Tactic -8109134417363526151 time 0.210388
[V] [TRT] Tactic -3998689942157953075 time 0.228312
[V] [TRT] Tactic -3448915218022119398 time 0.14638
[V] [TRT] Tactic -3120440365138571960 time 0.147588
[V] [TRT] Tactic -242550410056126076 time 0.228548
[V] [TRT]
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Relu(1)
[V] [TRT] Tactic 0 time 0.547884
[V] [TRT] Tactic 1 time 0.424728
[V] [TRT] Tactic 2 scratch requested: 36864000, available: 16777216
[V] [TRT] Tactic 5 scratch requested: 131045440, available: 16777216
[V] [TRT]
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv1/Relu(33)
[V] [TRT] --------------- Chose 14 (-3448915218022119398)
[V] [TRT]
[V] [TRT] --------------- Timing resnet_v1_50/block1/unit_1/bottleneck_v1/conv2/Conv2D + resnet_v1_50/block1/unit_1/bottleneck_v1/conv2/Relu(3)
[V] [TRT]
```

Lower Precision - FP16

FP16 matches the results closely to FP32

TensorRT automatically converts FP32 weights to FP16 weights

```
builder->setFp16Mode(true);
```

No guarantee that 16-bit kernels will be used when building the engine

```
builder->setStrictTypeConstraints(true);
```

Tensor Core kernels (HMMA) for FP16 (supported on Volta and Turing GPUs)

Lower Precision - INT8 Quantization

Setting the builder flag enables INT8 precision inference.

```
builder->setInt8Mode(true);
IInt8Calibrator* calibrator;
builder->setInt8Calibrator(calibrator);
```

	Dynamic Range	Mininum Positive Value
FP32	$-3.4\times10^{38} \sim +3.4\times10^{38}$	1.4×10^{-45}
FP16	-65504 ~ +65504	5.96×10^{-8}
INT8	-128 ~ +127	1

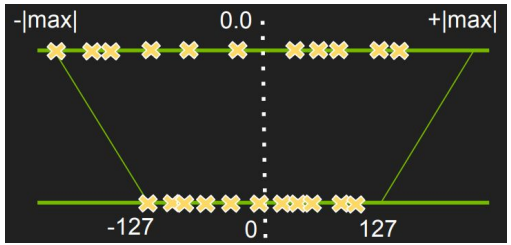
Quantization of FP32 weights and activation tensors

(weights) Int8_weight = ROUND_To_Nearest (scaling_factor * FP32_weight_in_the_filters)

* scaling_factor = 127.0 f / max (| all_FP32_weights |)

(activation) Int8_value = if (value > threshold); threshold; else scaling_factor * FP32_value

* Activation range unknown (input dependent) => calibration is needed



Dynamic range of each activation tensor => the appropriate quantization scale

TensorRT: symmetric quantization with quantization scale calculated using absolute maximum dynamic range values

Control precision per-layer with new APIs

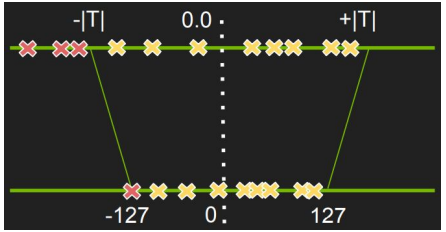
Tensor Core kernel (IMMA) for INT8 (supported on Drive AGX Xavier iGPU and Turing GPUs)

Lower Precision - INT8 Calibration

Run FP32 inference on Calibration

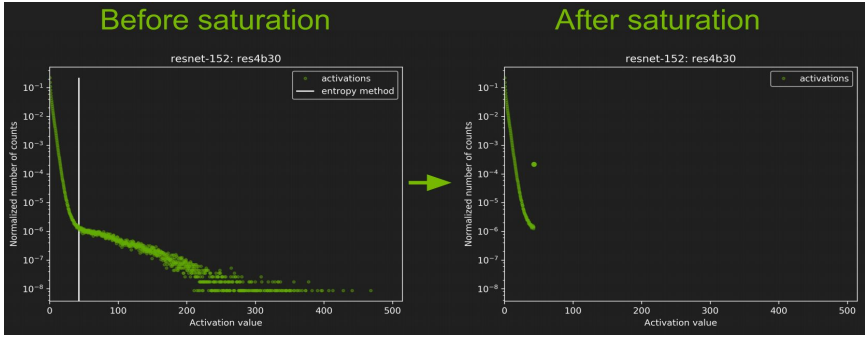
Per Layer:

- Histograms of activations
- Quantized distributions with different saturation thresholds.



Two ways to set saturation thresholds (dynamic ranges) :

- manually set the dynamic range for each network tensor using `setDynamicRange` API
  - * Currently, only symmetric ranges are supported
- use INT8 calibration to generate per tensor dynamic range using the calibration dataset (*i.e. 'representative' dataset)*
  - *pick threshold which minimizes KL_divergence (entropy method)



Plugin for Custom OPs in TensorRT 5

Custom op/layer: op/layer not supported by TensorRT => need to implement plugin for TensorRT engine

Plugin Registry

- stores a pointer to all the registered Plugin Creators / look up a specific Plugin Creator
- Built-in plugins: `RPROI_TRT, Normalize_TRT, PriorBox_TRT, GridAnchor_TRT, NMS_TRT, LReLU_TRT, Reorg_TRT, Region_TRT, Clip_TRT`

Register a plugin by calling `REGISTER_TENSORRT_PLUGIN(pluginCreator)` which statically registers the Plugin Creator to the Plugin Registry

Benchmark Tool: trtexec

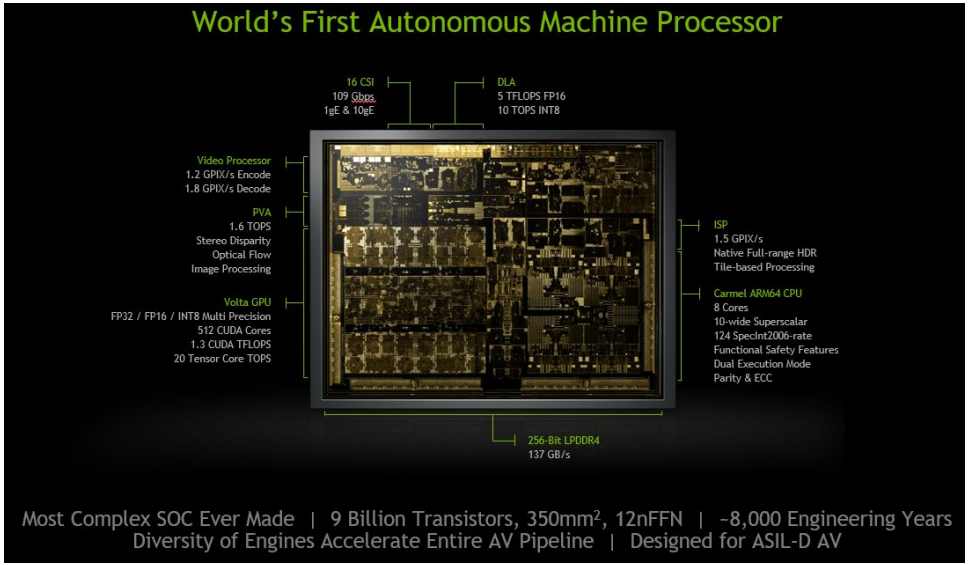
Useful tool to measure performance (latency, not accuracy)

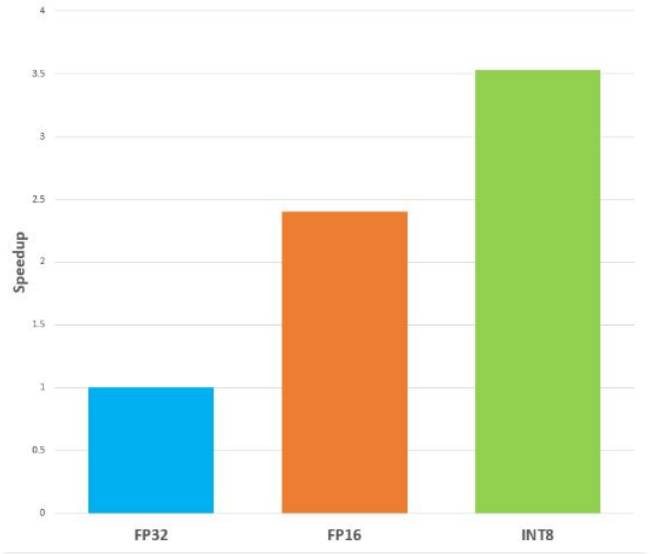
Source and prebuilt binary are provided.

```
Mandatory params:
  --deploy=<file>      Caffe deploy file
  OR --uff=<file>      UFF file
  --output=<name>      Output blob name (can be specified multiple times)

Mandatory params for onnx:
  --onnx=<file>        ONNX Model file

Optional params:
  --uffInput=<name>,C,H,W Input blob names along with their dimensions for UFF parser
  --model=<file>       Caffe model file (default = no model, random weights used)
  --batch=N            Set batch size (default = 1)
  --device=N           Set cuda device to N (default = 0)
  --iterations=N       Run N iterations (default = 10)
  --avgRuns=N          Set avgRuns to N - perf is measured as an average of avgRuns (default=10)
  --percentile=P       For each iteration, report the percentile time at P percentage (0<P<=100, default = 99.0%)
  --workspace=N        Set workspace size in megabytes (default = 16)
  --fp16               Run in fp16 mode (default = false). Permits 16-bit kernels
  --int8               Run in int8 mode (default = false). Currently no support for ONNX model.
  --verbose            Use verbose logging (default = false)
  --hostTime           Measure host time rather than GPU time (default = false)
  --engine=<file>      Generate a serialized TensorRT engine
  --calib=<file>       Read INT8 calibration cache file.  Currently no support for ONNX model.
  --useDLA=N        Enable execution on DLA for all layers that support DLA. Value can range from 1 to N, where N is the number of DLA engines on the platform. Set the --fp16 flag as well for DLA
  --allowGPUFallback   If --useDLA flag is present and if a layer cannot run on DLA, then run it on GPU.
```

TensorRT Performance on Xavier

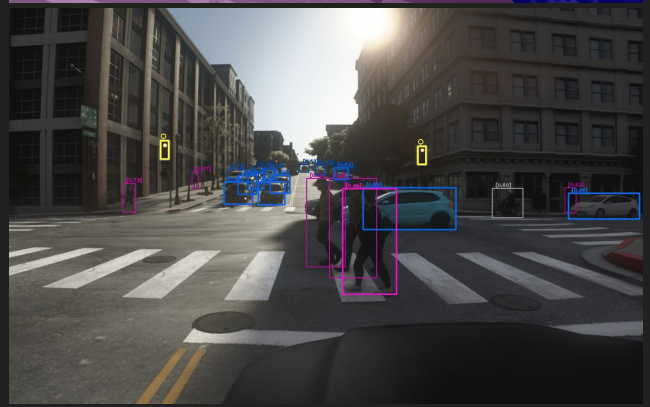
8x Volta SM, 512 CUDA cores, 64 Tensor Cores, 20 TOPS INT8, 10 TFLOPS FP16, 8x larger L1 cache size, 4x faster L2 cache access, CUDA compute capability 7.2

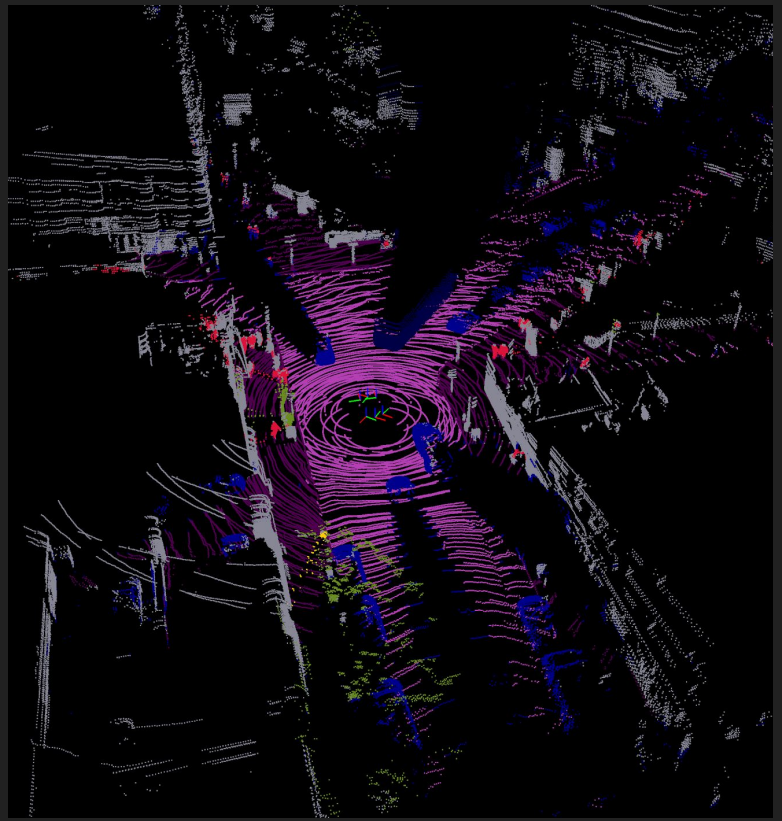




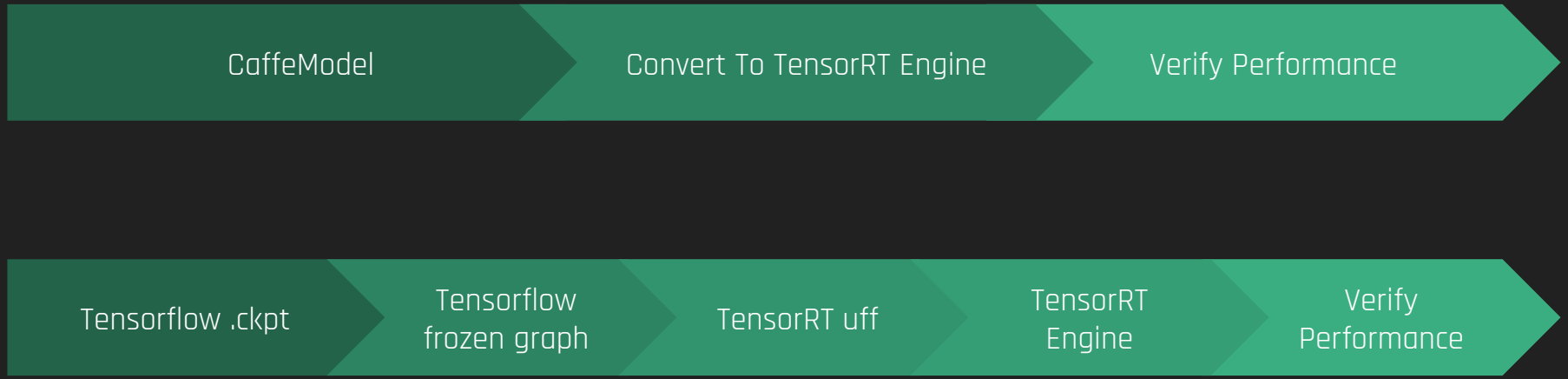
TensorRT SpeedUp Per Precision (resnet-18)

TensorRT at Zoox





TensorRT Conversion Pipeline



TensorRT at Zoox

Almost all of neural network models are deployed with TensorRT at Zoox

- Use cases include various vision/prediction/lidar models
- 2-6x speedup compared to Caffe/TensorFlow in Fp32.
- 6-13x speedup in Fp16.
- 9-19x speedup in Int8.

Benchmark results obtained on RTX 2080 Ti.

Fp16 Inference with TensorRT

Latency (Tesla V100, Resnet 50, Input Size: 224x224x3)

Batch Size	Fp32 (ms)	Fp16 (ms)	Speedup
4	4.356	2.389	1.8x
16	11.154	3.956	2.8x
32	20.090	6.439	3.1x
64	37.566	11.445	3.3x

Activation Overflow with Fp16

Backbone

Conv

Conv

...

Activation Overflow with Fp16

Backbone

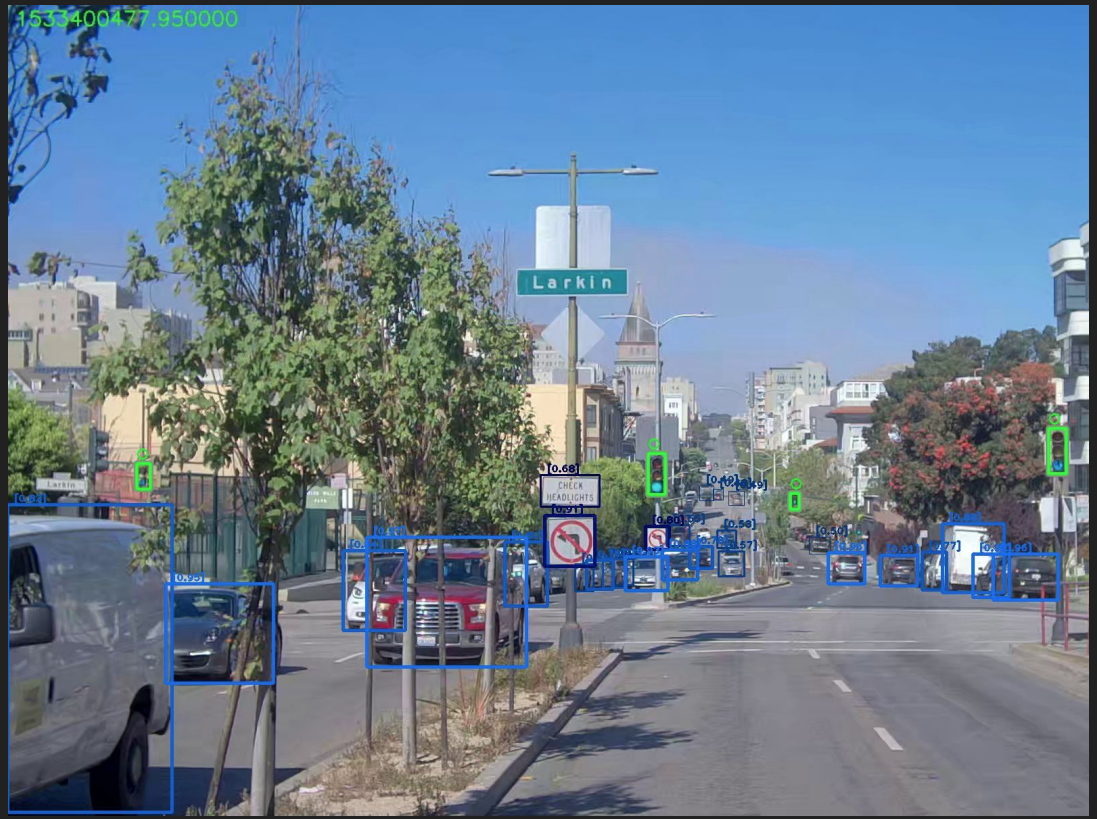
Conv

BN

Conv

BN

...

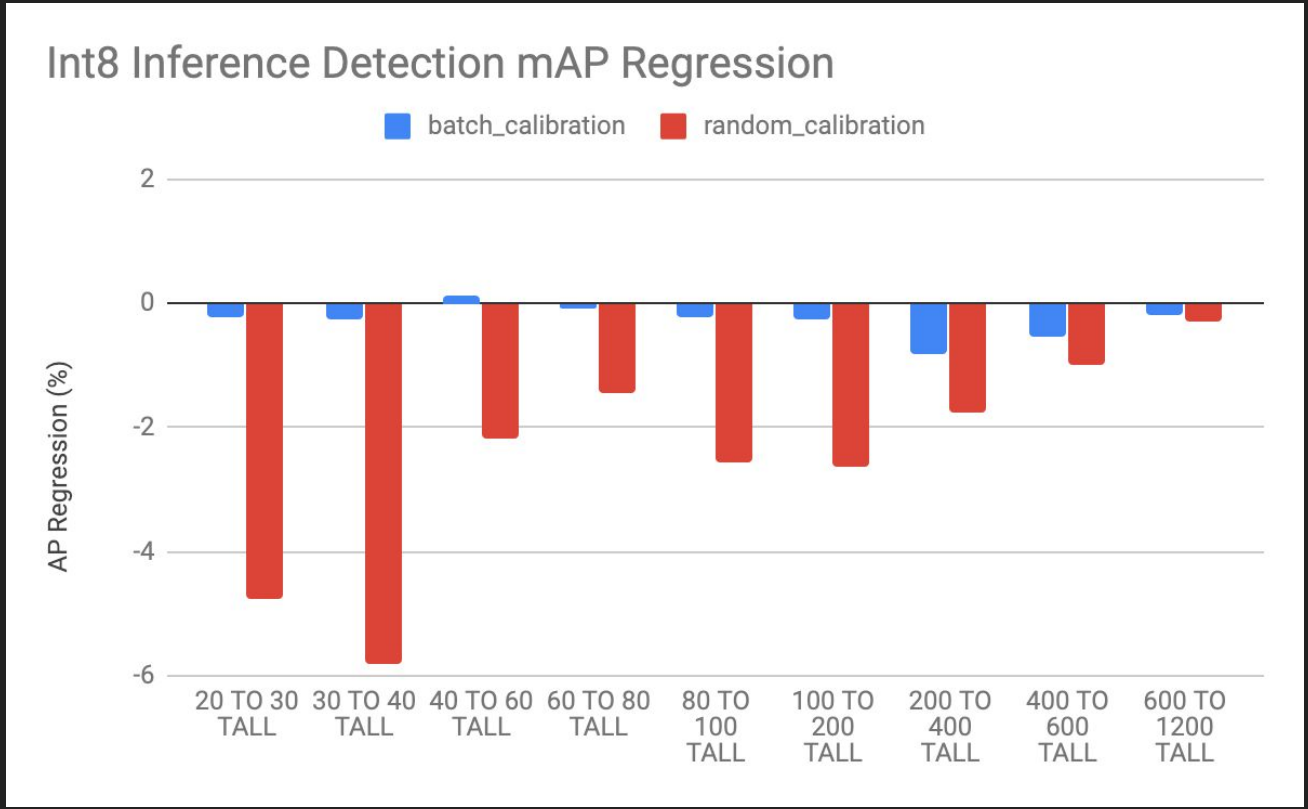


Int8 Inference: Latency

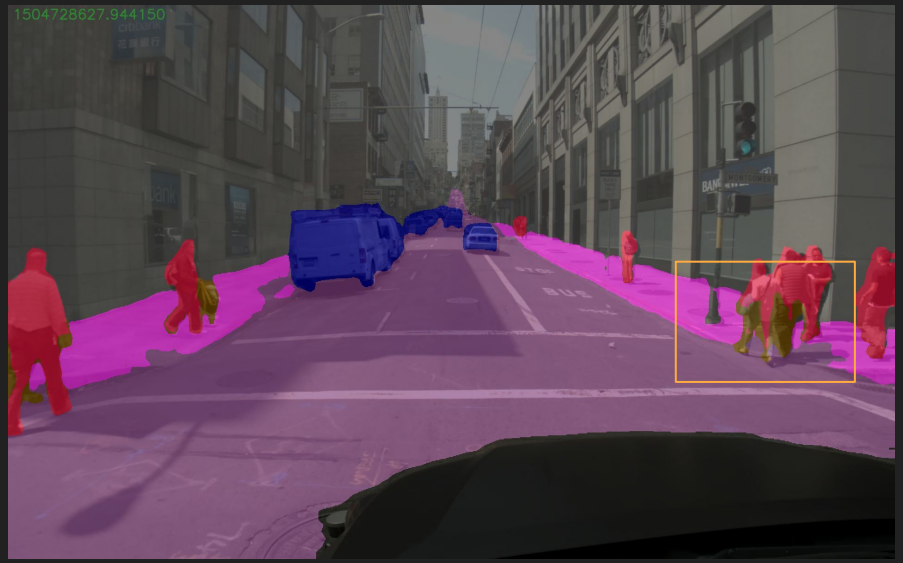
Latency (RTX 2080 Ti, Standard Resnet50, Input Size: 224x224x3)

Batch Size	Fp32 (ms)	Fp16 (ms)	Int8 (ms)	Fp16 Speedup	Int8 Speedup
4	3.800	1.722	1.212	2.2x	3.1x
16	11.305	3.631	2.121	3.1x	5.3x
32	21.423	6.473	3.629	3.3x	5.9x
64	40.938	12.497	6.636	3.3x	6.2x

Int8 Inference: Detection Performance



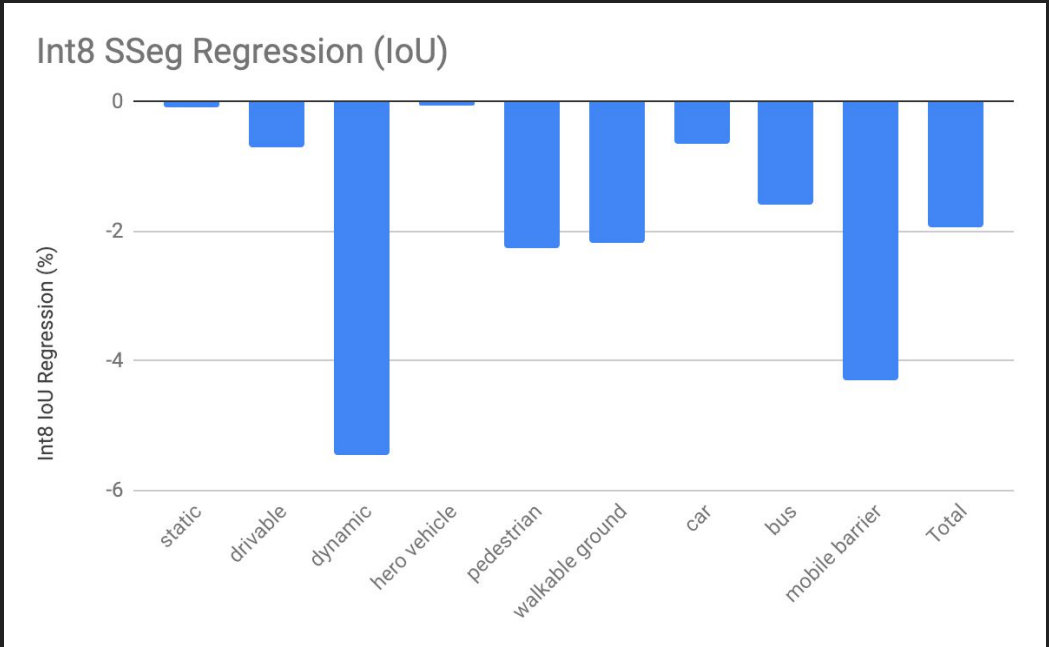
Int8 Inference: Semantic Segmentation Visualization



Int8 SSeg

Fp32 SSeg

Int8 Inference: Semantic Segmentation Performance



IoU = (target ∩ prediction) / (target ∪ prediction)

Next Steps on Int8 Inference

To resolve the regression:

- Inference with mixed precision
- Manually set the dynamic range (see slide 10)

	Fp32	Int8	Mixed (7 Fp32 layers, 27 int8 layers)
Area Under Curve (regression)	0	-0.006	-0.003
Latency (relative)	1.0	0.61	0.69

Summary: TensorRT at Zoox

Almost all of neural network models are deployed with TensorRT at Zoox

- 2-4x speedup compared to Caffe/TensorFlow in Fp32.

Reduced precision inference

- Fp16 inference works with no regression.
- Int8 inference needs calibration and might yield regression.
- 6-13x speedup in Fp16.
- 9-19x speedup in Int8.

Example: Converting a Tensorflow LeNet

```
def network(self, X):
    # Convolution Layer with 32 filters and a kernel size of 5
    conv1 = tf.layers.conv2d(X, 32, 5, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

    # Convolution Layer with 64 filters and a kernel size of 3
    conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

    # Fully connected layer (in tf contrib folder for now)
    fc1 = tf.contrib.layers.flatten(conv2)
    fc1 = tf.layers.dense(fc1, 1024)
    logits = tf.layers.dense(fc1, 10)
    output = tf.identity(logits, name = "output")
    return output
```

Two Steps

$ convert_to_uff --input_graph lenet5.pb --input-node input --output-node output --output lenet5.uff

available after installing `uff-****-py2.py3-none-any.whl`

$ convert_and_validate --uff_model lenet5.uff --output_engine lenet5.trt5p0p1 --input_dims 1,32,32 --original_graph lenet5.pb

modified from `loadModelAndCreateEngine` function in `samples/sampleUffSSD`

First Modification

```python
def network(self, X):
    # Convolution Layer with 32 filters and a kernel size of 5
    conv1 = tf.layers.conv2d(X, 32, 5, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

    # Convolution Layer with 64 filters and a kernel size of 3
    conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

    # Fully connected layer (in tf contrib folder for now)
    fc1 = tf.contrib.layers.flatten(conv2)
    fc1 = tf.layers.dense(fc1, 1024)
    logits = tf.layers.dense(fc1, 10)
    output = tf.identity(logits, name = "output")
    return output
```

Use output node name:

`dense_2/BiasAdd`

Let's convert it!

Well it converts, but ... (verification step is important!)

```
======
I0826 20:15:02.995867   576 convert_and_validate_tensorflow.cpp:152] Min diff 0.243159
I0826 20:15:02.995896   576 convert_and_validate_tensorflow.cpp:153] Max diff 5.09985
I0826 20:15:02.995903   576 convert_and_validate_tensorflow.cpp:154] 1/4 error 0.808826
I0826 20:15:02.995908   576 convert_and_validate_tensorflow.cpp:155] Median error 1.25298
I0826 20:15:02.995913   576 convert_and_validate_tensorflow.cpp:156] 3/4 error 2.02286
```

Diff is sky-high. Why?

- Tensorflow defaults to channel last (NHWC).
- TensorRT does not fully support this format.
  - Avoid changes in dimension if possible. (4D to 2D, or axis operations like slice, reshape, or split)

Getting Rid of Dimension Changes

```
def network(self, X):
    # Convolution Layer with 32 filters and a kernel size of 5
    conv1 = tf.layers.conv2d(X, 32, 5, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

    # Convolution Layer with 64 filters and a kernel size of 3
    conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

    # Fully connected layer (in tf contrib folder for now)
    fc1 = tf.contrib.layers.flatten(conv2)
    fc1 = tf.layers.dense(fc1, 1024)
    logits = tf.layers.dense(fc1, 10)
    output = tf.identity(logits, name = "output")
    return output
```

After Modification

```
def network(self, X):
    # Convolution Layer with 32 filters and a kernel size of 5
    conv1 = tf.layers.conv2d(input, 32, 5, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv1 = tf.layers.max_pooling2d(conv1, 2, 2)

    # Convolution Layer with 64 filters and a kernel size of 3
    conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv2 = tf.layers.max_pooling2d(conv2, 2, 2)

    # Use conv2d instead of fc for TensorRt
    fc1 = tf.layers.conv2d(conv2, 1024, 6, activation=None)
    fc2 = tf.layers.conv2d(fc1, 10, 1, activation=None, name="fc2")
    logits = tf.contrib.layers.flatten(fc2)
    output = tf.identity(logits, name="output")
    return output
```

We only need the output here in trt. Output node: fc2/BiasAdd

In our network conv2 outputs a ?x6x6x64 tensor (nhwc).

A 6 by 6 conv with 1024 conv filters it's the same as a fully connected layer.

Let's Convert it Again!

```
I0826 20:34:45.630513  3969 convert_and_validate_tensorflow.cpp:152] Min diff 7.15256e-07
I0826 20:34:45.630527  3969 convert_and_validate_tensorflow.cpp:153] Max diff 5.72205e-06
I0826 20:34:45.630533  3969 convert_and_validate_tensorflow.cpp:154] 1/4 error 9.53674e-07
I0826 20:34:45.630537  3969 convert_and_validate_tensorflow.cpp:155] Median error 2.86102e-06
I0826 20:34:45.630542  3969 convert_and_validate_tensorflow.cpp:156] 3/4 error 3.8147e-06
```

```
=======
B=1 TensorRT avg per image 0.0002904s over 1000 iter.
=======
```

```
=======
B=1 Tensorflow avg per image 0.0007458s over 1000 iter.
=======
```

~2.5x speedup with TensorRT

Some Other Tips

Use Tensorflow tools/graph_transforms/summarize_graph to verify frozen graph.

Use Identity op to control input node.

Use graphsurgeon package to manipulate Tensorflow graphs.

Use tensorflow transform_graph to fold BatchNorms.

Thanks!

Special thanks to:

Perception team and Infra team members from Zoox

Joohoon Lee's team from Nvidia

Q & A

Extra Materials

Converting BatchNorms

```
bn = layers.batch_norm(pool, scope='bn', is_training = is_training)
```

Issue 1: *is_training* creates a Select op that's not supported in TensorRT.

Solution: Find all Select op and replace them with Identity.

```
# Find the select op in the graph and replace it with identity op.
def remove_select(input_graph):
    no_select_graph = copy.deepcopy(input_graph)
    for i in range(len(no_select_graph.node)):
        if (no_select_graph.node[i].op == "Select"):
            node_name = no_select_graph.node[i].input[0]
            curr_node = no_select_graph.node[i]

            # is_training is always False during inference.
            evaluated_value = False

            # Replace the select op with Identity.
            no_select_graph.node[i].op = "Identity"
            if (evaluated_value == False):
                input_name = no_select_graph.node[i].input[2]
            else:
                input_name = no_select_graph.node[i].input[1]

            no_select_graph.node[i].ClearField("input")
            no_select_graph.node[i].input.extend([input_name])
    return no_select_graph
```

Converting BatchNorms

```
bn = layers.batch_norm(pool, scope='bn', is_training = is_training)
```

Issue 2: batch_norm involves a series of operations that's not supported in TensorRT.

Solution: Fold the batch_norm into convolution.

```
from tensorflow.tools.graph_transforms import TransformGraph

# Optimize graph by removing unused nodes and fold batch norm.
transforms = ["strip_unused_nodes", "fold_constants",
              "remove_nodes(op=Identity, op=CheckNumerics)",
              "fold_batch_norms", "fold_old_batch_norms"]
transformed_graph = TransformGraph(no_select_graph, input_layers,
                                   output_layers, transforms)
```

Verify Frozen Graph

There should be no variables, all weights are frozen

This is your input node

This is your output node.

```
INFO: Running command line: bazel-bin/external/org_tensorflow/tensorflow/tools/graph_transforms/summarize_graph '--in_graph
=/mnt/flashblade/zejia/tf_to_trt_example/output_graph.pb'
Found 1 possible inputs: (name=input, type=float(1), shape=[?,32,32,1])
No variables spotted.
Found 1 possible outputs: (name=output, op=Identity)
Found 2389902 (2.39M) const parameters, 0 (0) variable parameters, and 0 control_edges
Op types used: 12 Const, 9 Identity, 4 BiasAdd, 2 Conv2D, 2 MatMul, 2 MaxPool, 2 Relu, 1 Pack, 1 Placeholder, 1 Reshape, 1
Shape, 1 StridedSlice
To use with tensorflow/tools/benchmark:benchmark_model try these arguments:
bazel run tensorflow/tools/benchmark:benchmark_model -- --graph=/mnt/flashblade/zejia/tf_to_trt_example/output_graph.pb --s
how_flops --input_layer=input --input_layer_type=float --input_layer_shape=-1,32,32,1 --output_layer=output
```

What if I only want to convert part of the network?

E.g., input queues are a lot faster than naive placeholder.

Solution: use tf.identity.

```
def network(self, X):
    input = tf.identity(X, name = "input")
    # Convolution Layer with 32 filters and a kernel size of 5
    conv1 = tf.layers.conv2d(input, 32, 5, activation=tf.nn.relu)
    # Max Pooling (down-sampling) with strides of 2 and kernel size of 2
    conv1 = tf.layers.max_pooling2d(conv1, 2, 2)
```

Then in tf_to_uff and convert_and_validate_tensorflow use this as your input layer

TensorRT Graphsurgeon

For Tensorflow -> Uff conversion, sometimes the graph needs to be processed first in order to be successfully converted to TensorRT.

Example: Tensorflow inserts chain of Shape, Slice, ConcatV2, Reshape before Softmax.
Slice is not supported by TensorRT.
Solution: Use the TensorRT graphsurgeon API to remove this chain and pass the inputs directly to Softmax.