TensorRT Inference with TensorFlow

Pooya Davoodi (NVIDIA)
Chul Gwon (Clarifai)
Guangda Lai (Google)
Trevor Morris (NVIDIA)

March 20, 2019

TensorFlow

An end-to-end open source machine learning platform

- Powerful experimentation for research
- Easy model building
- Robust ML production anywhere

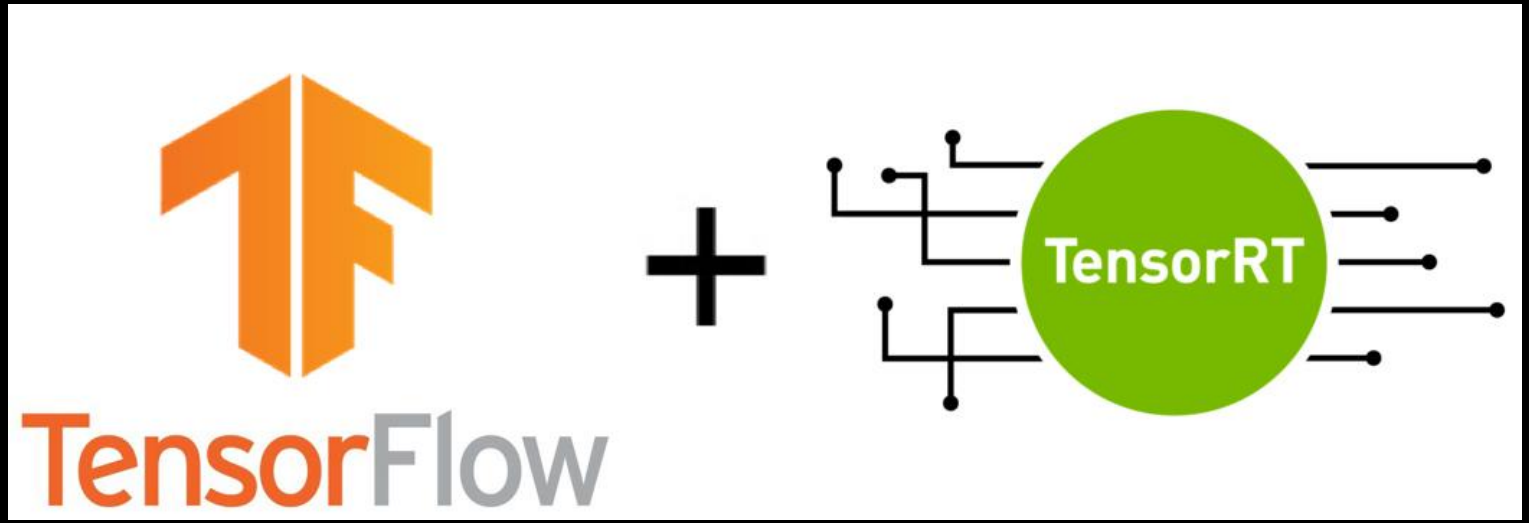
41m Downloads

NVIDIA TensorRT

Platform for High-Performance Deep Learning Inference

- Optimize and Deploy neural networks in production environments
- Maximize throughput for latency-critical apps with optimizer and runtime
- Deploy responsive and memory efficient apps with INT8 & FP16

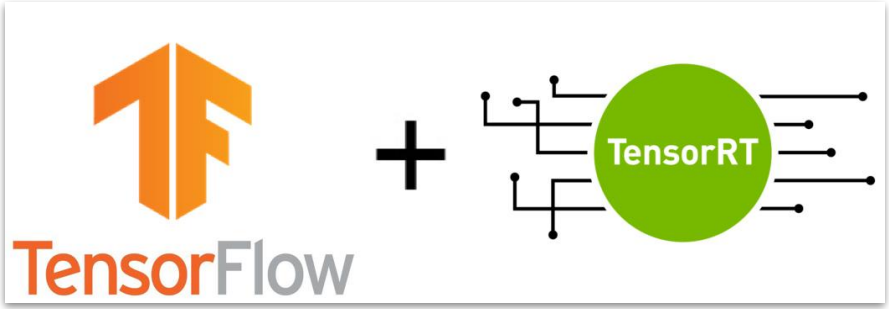
300k Downloads in 2018



TF-TRT = TF + TRT

Why to use TF-TRT

- Optimize TF inference
- Simple API
- Possible to optimize even if parts of model are not supported by TRT
- Can still use TF echosystem
- Extract TRT optimized parts out of TF model, and execute standalone

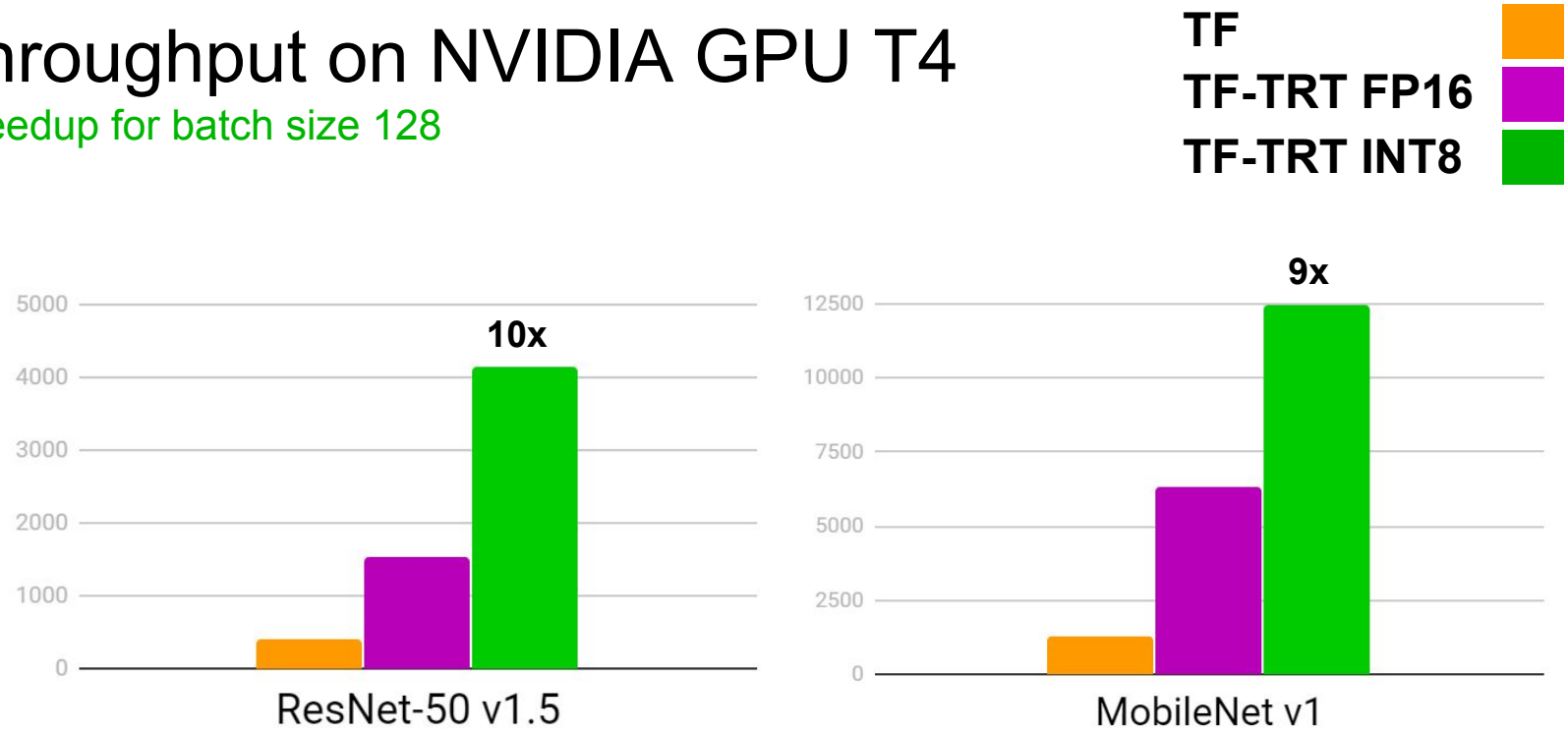


AGENDA

- Performance & Accuracy
- How to use TF-TRT
- How TF-TRT works
- Customer experience: Clarifai

Throughput on NVIDIA GPU T4

Speedup for batch size 128



Benchmark inference only (no I/O or preprocessing)
TensorFlow 1.13 in NVIDIA TensorFlow 19.03 containers
Scripts: https://github.com/tensorflow/tensorrt

Optimized models

- ResNet **10x**
- MobileNet **9x**
- Inception **8x**
- VGG **7x**
- NASNet L/M **4x**
- SSD MobileNet v1 **3x**

Coming soon:

- Faster-RCNN, Mask-RCNN
- Neural Collaborative Filtering
- NLP: Transformer, BERT

SSD: available soon in NVIDIA containers and github.com/tensorflow/tensorflow/
Scripts: https://github.com/tensorflow/tensorrt

Accuracy of FP16

Models	TF FP32	TF-TRT FP16
Mobilenet V2	74.08	74.07
NASNet Mobile	73.97	73.87
ResNet 50 V2	76.43	76.40
VGG 16	70.89	70.91
Inception V3	77.99	77.97
SSD Mobilenet v1	23.062	23.073

FP16 accuracy is within 0.1% of FP32 accuracy.

Top1 metric for classification models.
mAP for detection models.
Complete data: https://docs.nvidia.com/deeplearning/dgx/integrate-tf-trt/index.html#verified-models

Accuracy of INT8

Models	TF FP32	TF-TRT INT8
Mobilenet V2	74.08	73.90
NASNet Mobile	73.97	73.55
ResNet 50 V2	76.43	76.30
VGG 16	70.89	70.78
Inception V3	77.99	77.85

INT8 accuracy is within 0.2% of FP32 accuracy, except one model that's within 0.5%.

Top1 metric for classification models.
Complete data: https://docs.nvidia.com/deeplearning/dgx/integrate-tf-trt/index.html#verified-models

Supported TensorFlow operators

Most of important ops are supported

67 operators are supported

Not all types of inputs or attributes are supported.

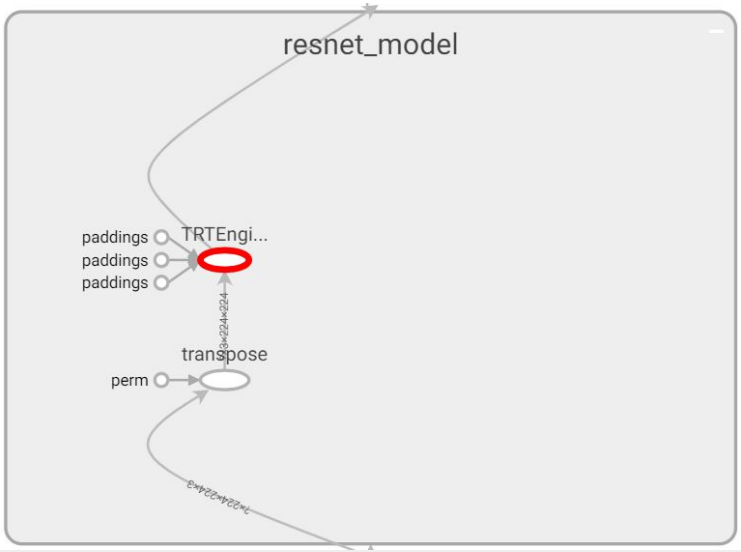
Examples of supported operators:

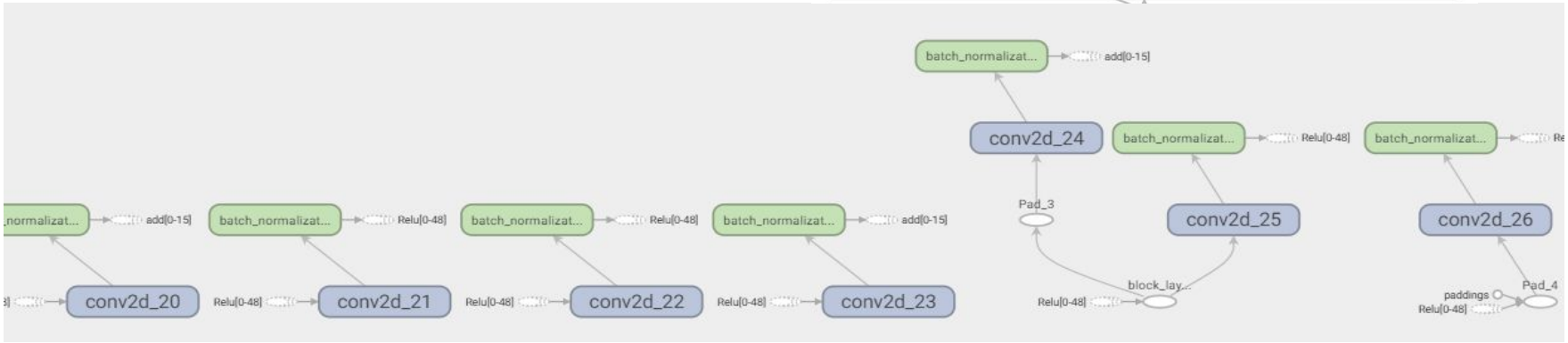
- Gather, (Strided)Slice, Topk
- Convolution: depthwise, dilated convolution
- Shape related: ExpandDims, Reshape, Squeeze
- NMS (Non-Max Suppression): highly effective in performance

List of supported ops: https://docs.nvidia.com/deeplearning/dgx/integrate-tf-trt/index.html#support-ops

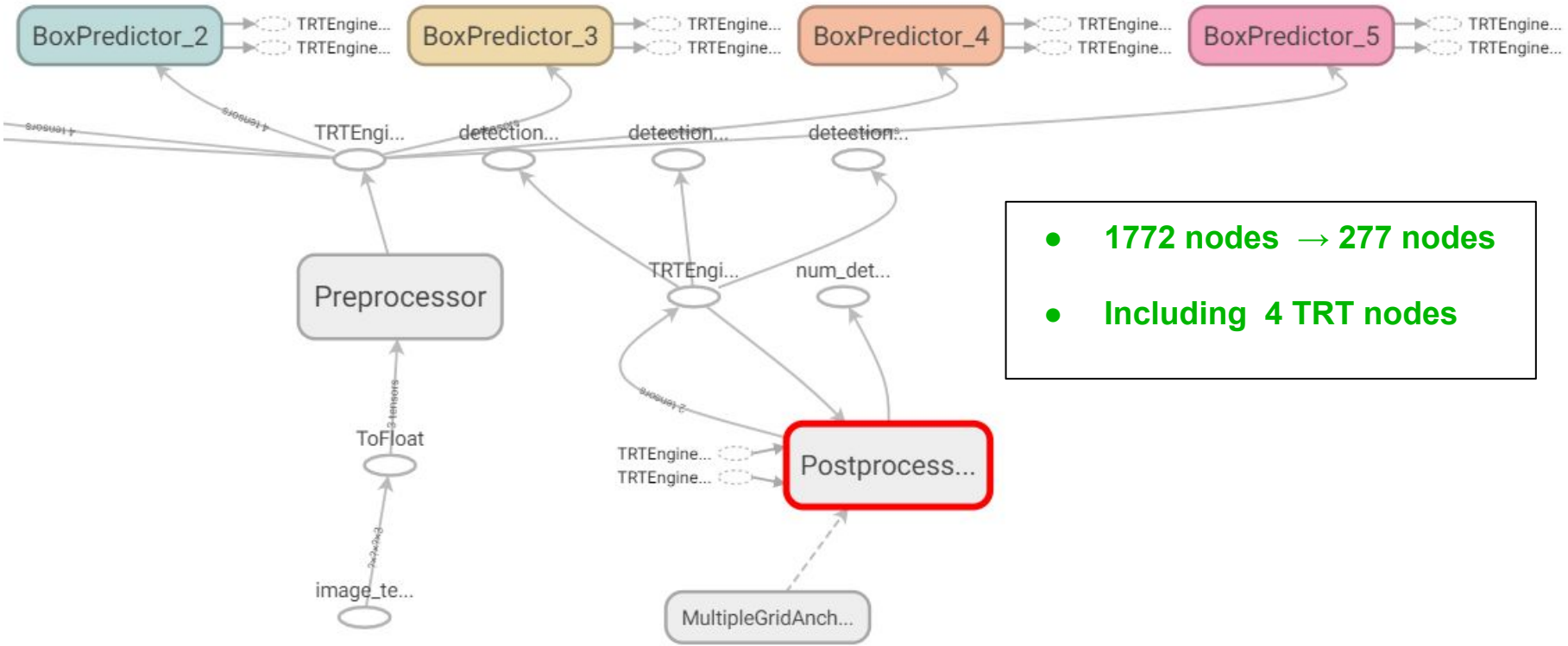
ResNet-50 v1.5

- **741 nodes → 12 nodes**
- **Including 1 TRT node**





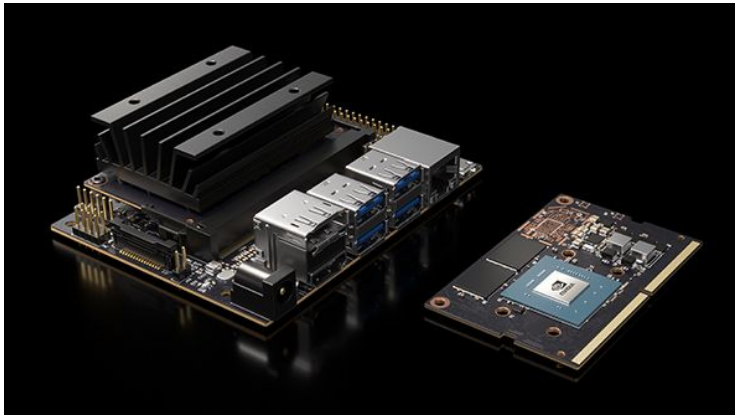
SSD Mobilenet v1



- **1772 nodes → 277 nodes**
- **Including 4 TRT nodes**


Where to use TF-TRT

TF-TRT on Jetson Platform



Monthly release of Tensorflow

- Nano, Xavier, TX2

How to setup

- Install Jetpack
- Install TF dependencies (numpy, libjpeg8-dev, requests, h5py, etc)
- Install TF
  - `pip install --extra-index-url https://developer.download.nvidia.com/compute/redist/jp/v42 tensorflow-gpu`

https://docs.nvidia.com/deeplearning/dgx/index.html#installing-frameworks-for-jetson

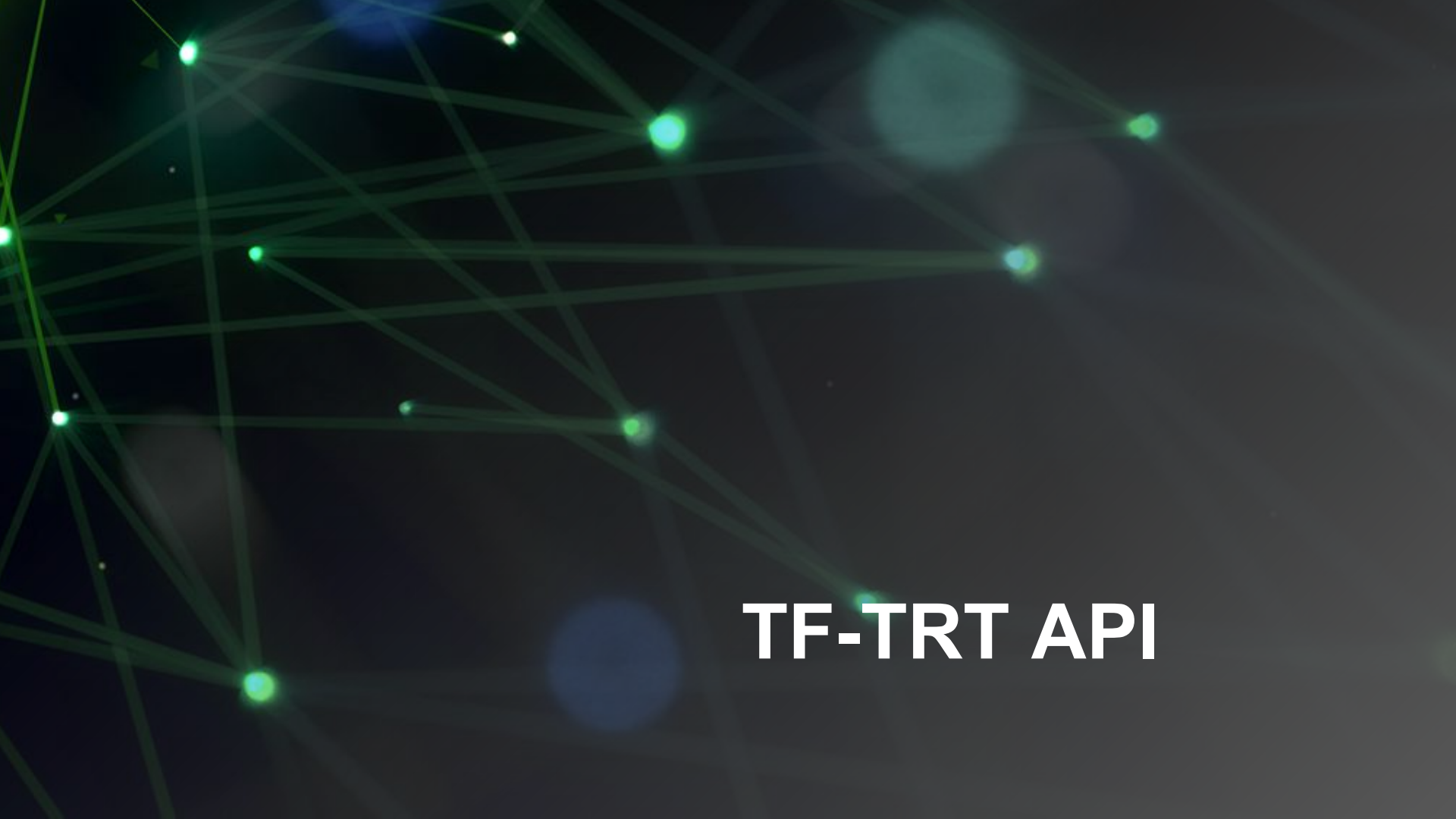
Cloud inferencing solutions

Multiple models scalable across GPUs

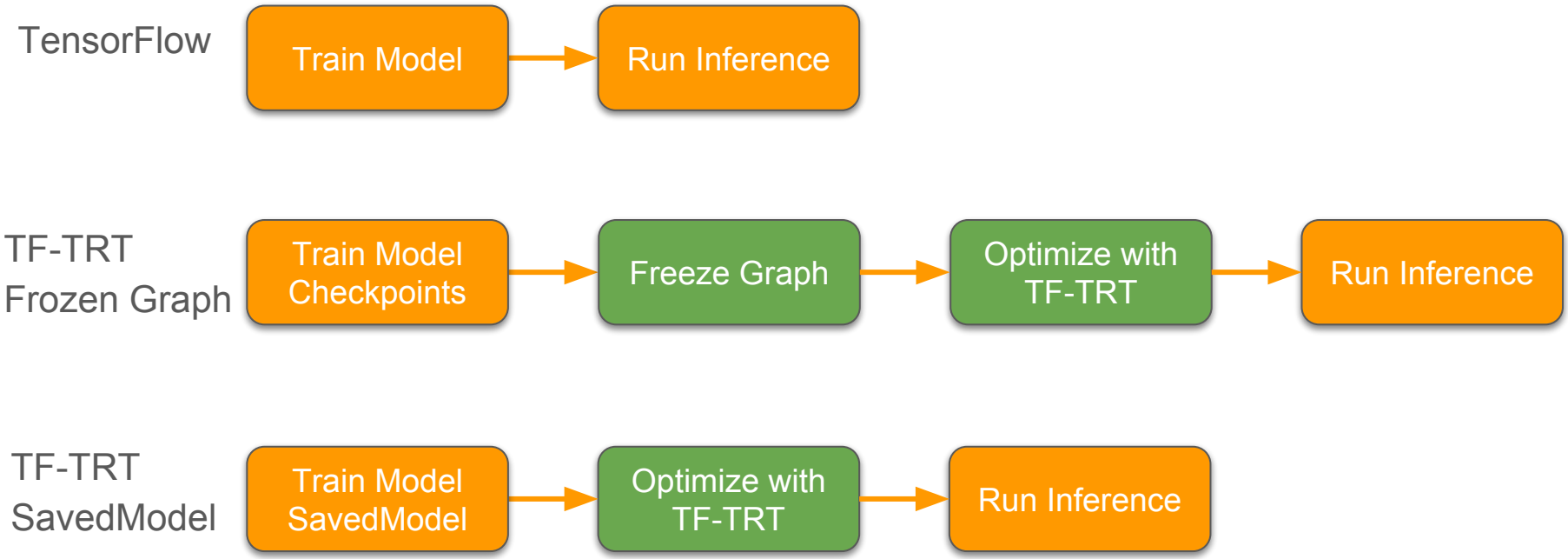
- TensortRT Inference Server (TRTIS)
  - TensorRT, TensorFlow, and other inferencing engines
  - Monthly release in containers
  - github.com/NVIDIA/tensorrt-inference-server

- TensorFlow Serving (TFS)
  - TF-TRT with TensorFlow >=1.13
  - TRT 5.0
  - tensorflow.org/serving

- Maximizing Utilization for Data Center Inference with TRTIS, Wed 11am 220C, 12pm Hall3
- TensorFlow Extended: How to Take AI from Experimentation to Production, Wed 11am 210F


TF-TRT API

Inference workflow



TF-TRT API in TensorFlow <=1.13

One API call returns a TF-TRT optimized graph

```
import tensorflow.contrib.tensorrt as trt
converted_graph_def = trt.create_inference_graph(
    input_graph_def=frozen_graph,
    outputs=['logits', 'classes'])
```

```
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)
```

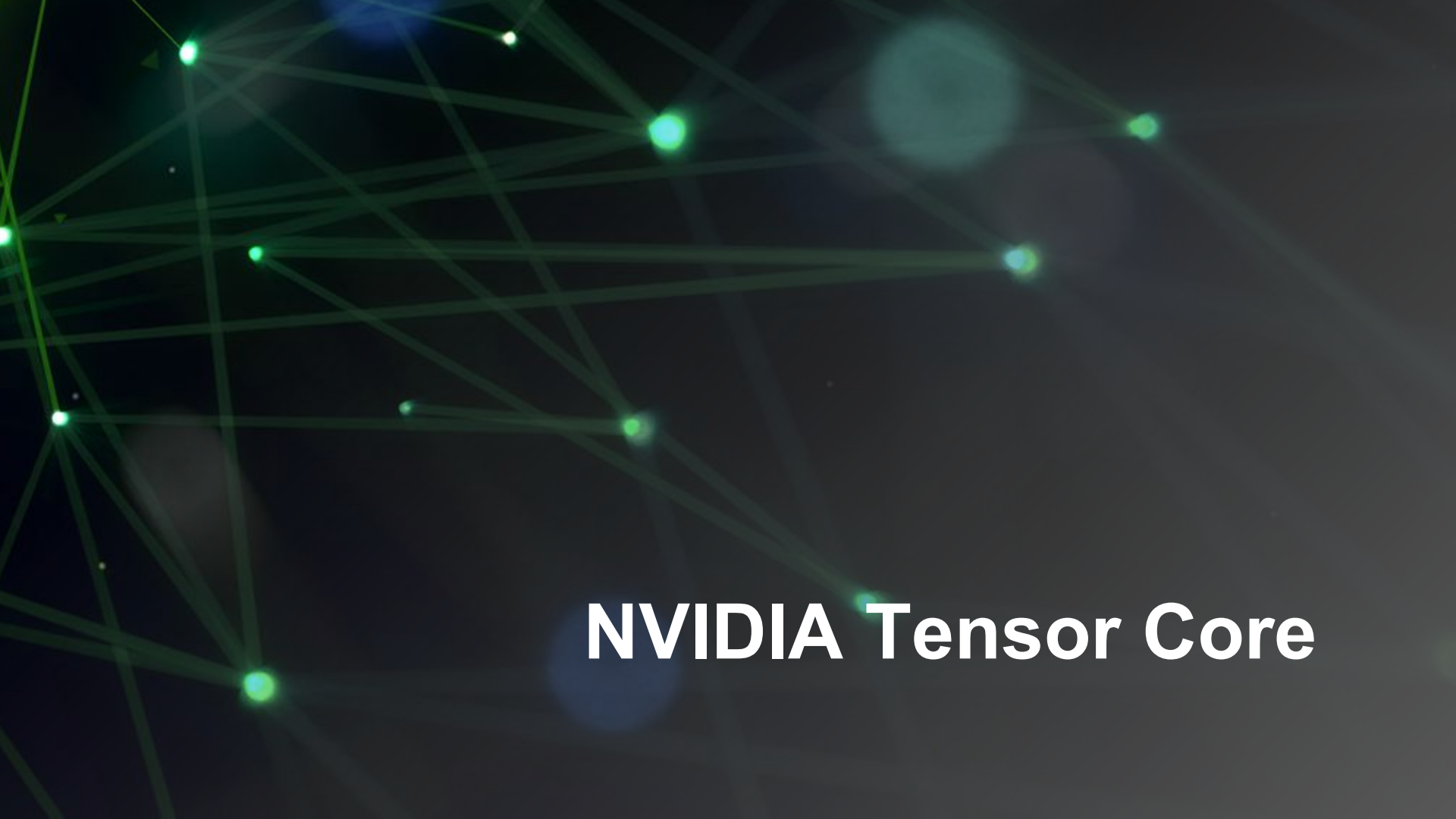
TF-TRT API in TensorFlow > 1.13

contrib → compiler

Python class

```
from tensorflow.python.compiler.tensorrt import trt_convert as trt
converter = trt.TrtGraphConverter(
   input_saved_model_dir=input_saved_model_dir)
converter.convert()
converter.save(output_saved_model_dir)
```

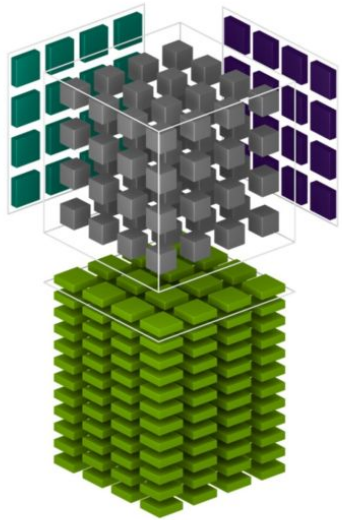
```
pip install tensorflow-gpu==2.0.0-alpha0
```


NVIDIA Tensor Core

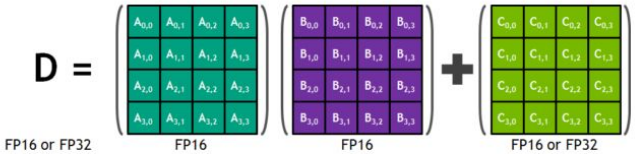
Tensor Cores in GPU Volta/Turing

Easy to enable

- TensorRT enables Tensor Cores automatically



```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=input_saved_model_dir,
    precision_mode="FP16" OR "INT8")
```



Profile to verify Tensor Core usage

Multiple profilers

- nvprof
- NVIDIA NSight Systems
- NVIDIA NSight Compute
- NVIDIA DLProf
- TensorFlow Profiler

GTC

- Profiling Deep Learning Networks, Tuesday, Poonam Chitale, David Zier
- Deep Learning Developer Tools for Network Optimization, Wed 4-6pm Hall 3

nvprof for verifying Tensor Core usage

h884, h1688, i8816

```
$ nvprof python run_inference.py
...
==87== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   20.85%  1.41948s     46080  30.804us  14.688us  694.17us  trt_turing_h1688cudnn_128x128_ldg8_relu_exp_interior_nhwc_tn_v1
                   17.88%  1.21692s     32104  37.905us  13.120us  127.78us  trt_turing_h1688cudnn_128x128_ldg8_relu_exp_small_nhwc_tn_v1
                   10.91%  742.33ms     34034  21.811us  6.3680us  58.335us  void cuScale::scale<__half, __half, bool=1, cuScale::Mode, bool=0, ...
                    7.77%  528.65ms     10080  52.445us  13.184us  437.02us  trt_turing_h1688cudnn_256x64_sliced1x2_ldg8_relu_exp_interior_nhwc_...
                    5.75%  391.27ms      8104  48.280us  13.216us  127.01us  trt_turing_h1688cudnn_256x64_sliced1x2_ldg8_relu_exp_small_nhwc_tn...
                    4.27%  290.90ms      4736  61.423us     672ns  9.1938ms  [CUDA memcpy HtoD]
                    4.19%  284.93ms      2080  136.99us  26.847us  367.39us  trt_volta_scudnn_128x64_relu_interior_nn_v1
                    2.59%  176.06ms      4106  42.878us  14.112us  702.43us  trt_turing_h1688cudnn_128x128_ldg8_relu_exp_medium_nhwc_tn_v1
                    2.53%  172.25ms      1152  149.53us  75.807us  263.33us  volta_cgemm_32x32_tn
                    2.44%  165.84ms      8010  20.703us  2.3040us  48.575us  void cuPad::pad<__half, int4, int=128, bool=0>...
                    2.16%  146.81ms      2218  66.189us  2.2400us  72.767us  void cuInt8::nchwTonhwc<float, int=32, int=32, int=2>...
                    1.30%  88.795ms      2000  44.397us  43.679us  62.111us  void Eigen::internal::EigenMetaKernel<Eigen::TensorEvaluator...
                    1.20%  81.957ms      2106  38.916us  13.664us  449.08us  trt_turing_h1688cudnn_256x64_sliced1x2_ldg8_relu_exp_medium_nhwc...
                    1.16%  78.870ms      2034  38.775us  30.880us  452.12us  trt_turing_h1688cudnn_256x64_sliced1x2_ldg8_relu_exp_large_nhwc_tn...
                    1.06%  71.838ms      2002  35.883us  22.176us  45.888us  trt_volta_h884gemm_64x64_ldg8_relu_nn_v1
                    0.99%  67.413ms      2002  33.673us  31.200us  35.104us  void nvinfer1::poolCoalescedC<nvinfer1::PoolingType, int=3, bool=0>...
```

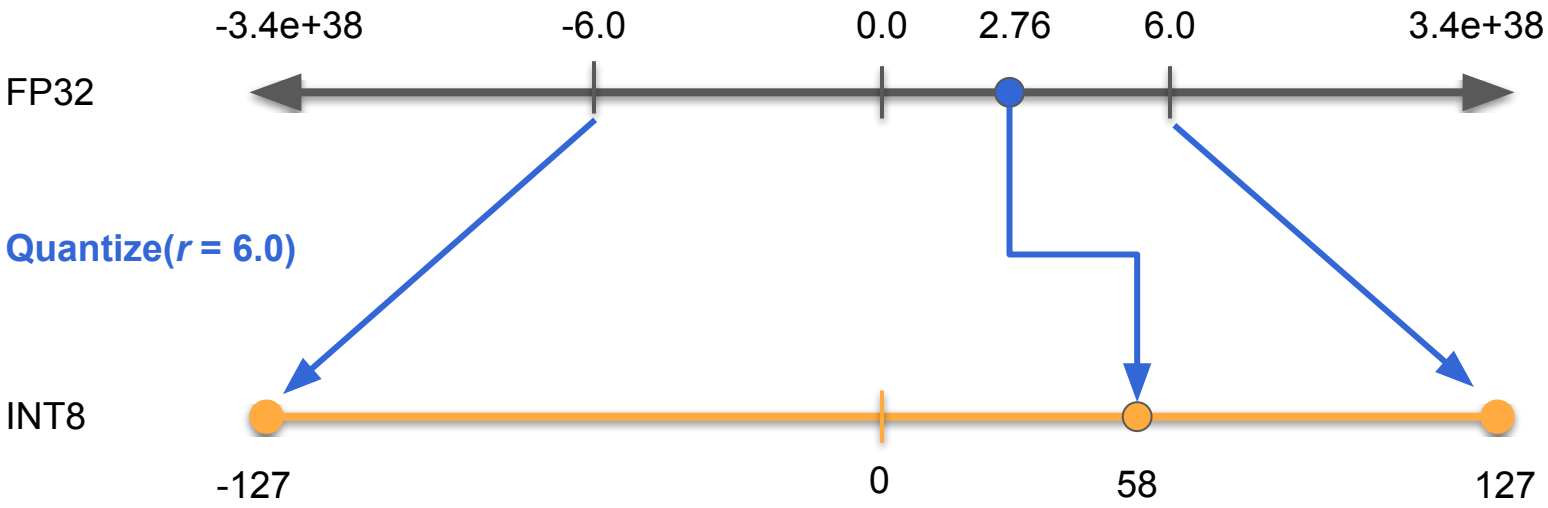
What if not using Tensor Core

- Hardware: GPU Volta or Turing
- Configuration
  - precision_mode: FP16 or INT8
  - Dimensions must be multiples of 8
- Tensor Core may not be the fastest
- Unsupported case
- Report to NVIDIA

https://docs.nvidia.com/deeplearning/dgx/integrate-tf-trt/index.html


INT8 Quantization

TensorRT's INT8 Quantization Approach



$$Quantize(x, r) = round(s * clip(x, -r, r))$$
$$\text{where } s = 127 / r$$

Two Methods for Determining Quantization Ranges

1. Calibration
   - Recommended method
   - Works with most models with minimal accuracy loss (<1%)

2. Quantization-Aware Training
   - Model the quantization error during training
   - Quantization ranges are learned
   - Can provide better accuracy than calibration

TF-TRT calibration API in TensorFlow <=1.13

```
import tensorflow.contrib.tensorrt as trt
calib_graph = trt.create_inference_graph(...
    precision_mode='INT8',
    use_calibration=True)
```

TF-TRT calibration API in TensorFlow <=1.13

```
import tensorflow.contrib.tensorrt as trt
calib_graph = trt.create_inference_graph(...
    precision_mode='INT8',
    use_calibration=True)
with tf.session() as sess:
    tf.import_graph_def(calib_graph)
    for i in range(10):
        sess.run('output:0', {'input:0': my_next_data()})
```

TF-TRT calibration API in TensorFlow <=1.13

```
import tensorflow.contrib.tensorrt as trt
calib_graph = trt.create_inference_graph(...
    precision_mode='INT8',
    use_calibration=True)
with tf.session() as sess:
    tf.import_graph_def(calib_graph)
    for i in range(10):
        sess.run('output:0', {'input:0': my_next_data()})
converted_graph_def = trt.calib_graph_to_infer_graph(calib_graph)
```

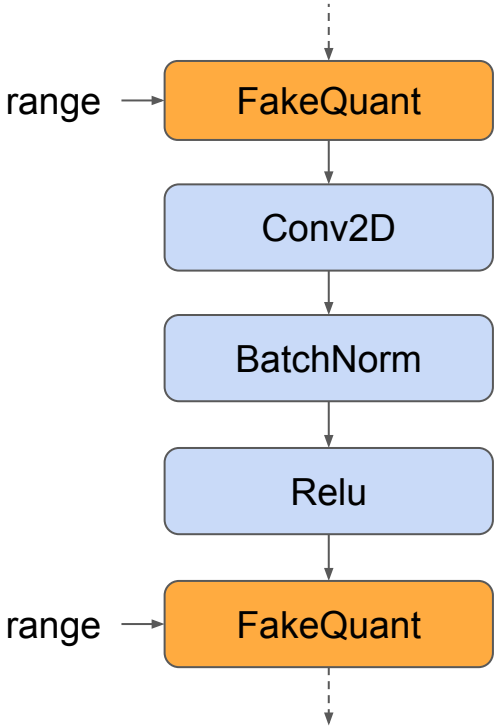
TF-TRT calibration API in TensorFlow > 1.13

```
from tensorflow.python.compiler.tensorrt import trt_convert as trt
converter = trt.TrtGraphConverter(...)
converter.convert()
converter.calibrate(
    fetch_names=['output:0'],
    num_runs=10,
    feed_dict_fn=lambda: {'input:0': my_next_data()})
converter.save(output_saved_model_dir)
```

Quantization-Aware Training

- Can increase accuracy beyond calibration
- Insert quantization nodes into your pretrained model
  - Experimental
- Finetune model to adapt for quantization error
- Give model to TF-TRT

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=input_saved_model_dir,
    precision_mode="INT8",
    use_calibration=False)
converter.convert()
converter.save(output_saved_model_dir)
```



How TF-TRT Works

How TF-TRT works

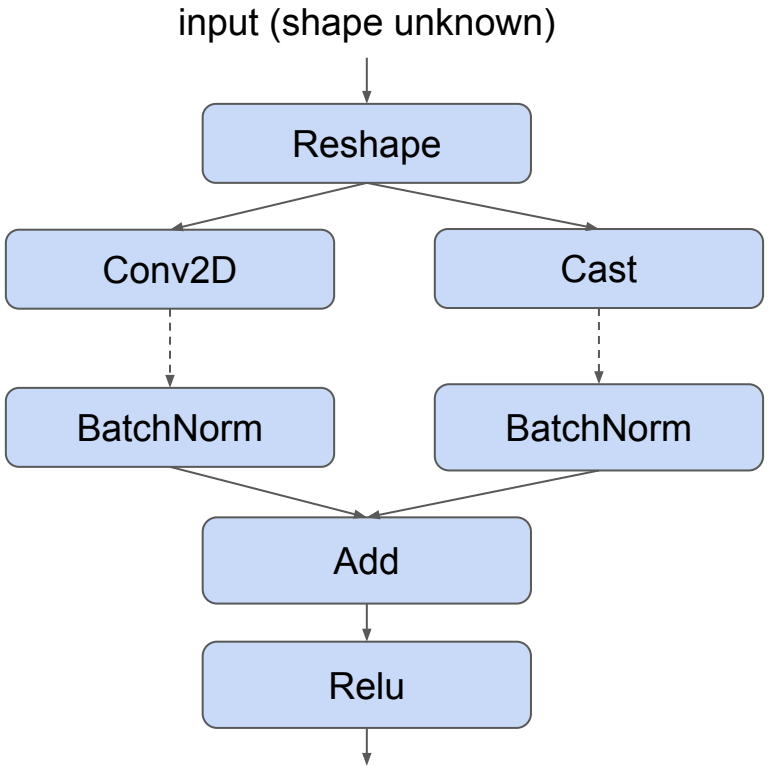
Under the hood:

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

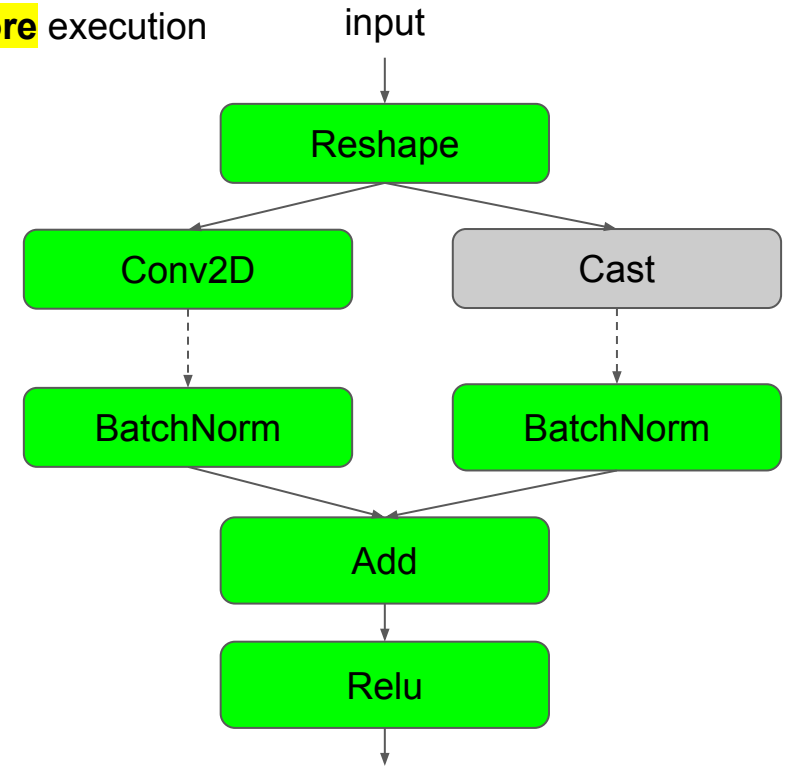
Example




```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: mark TRT-compatible nodes

Before execution



- Visit all nodes
- Mark them as TRT-compatible or TRT-incompatible based on:
  - Operation type
  - Attribute settings

Legend

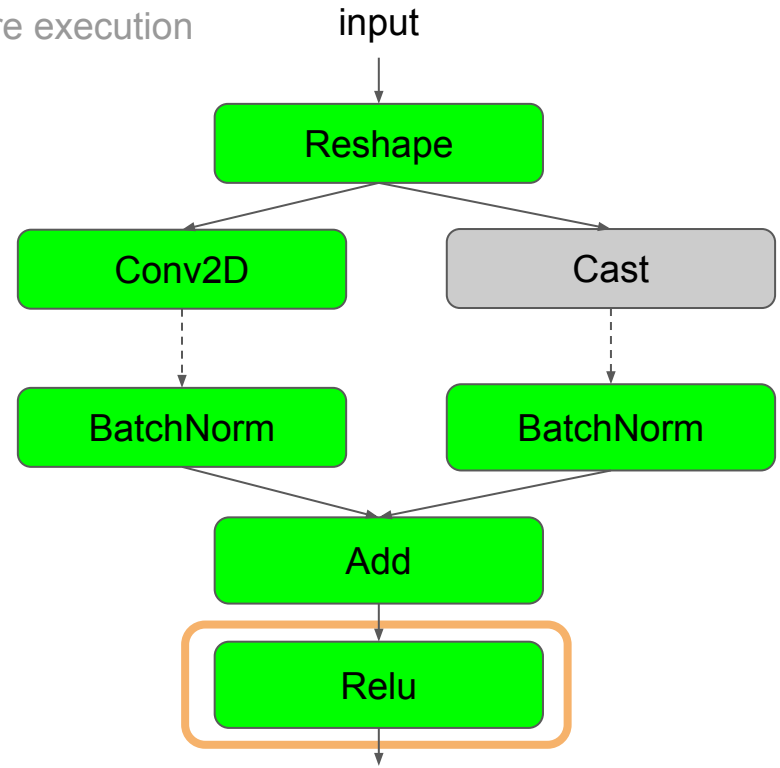
TRT-compatible

TRT-incompatible

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: cluster marked nodes

Before execution

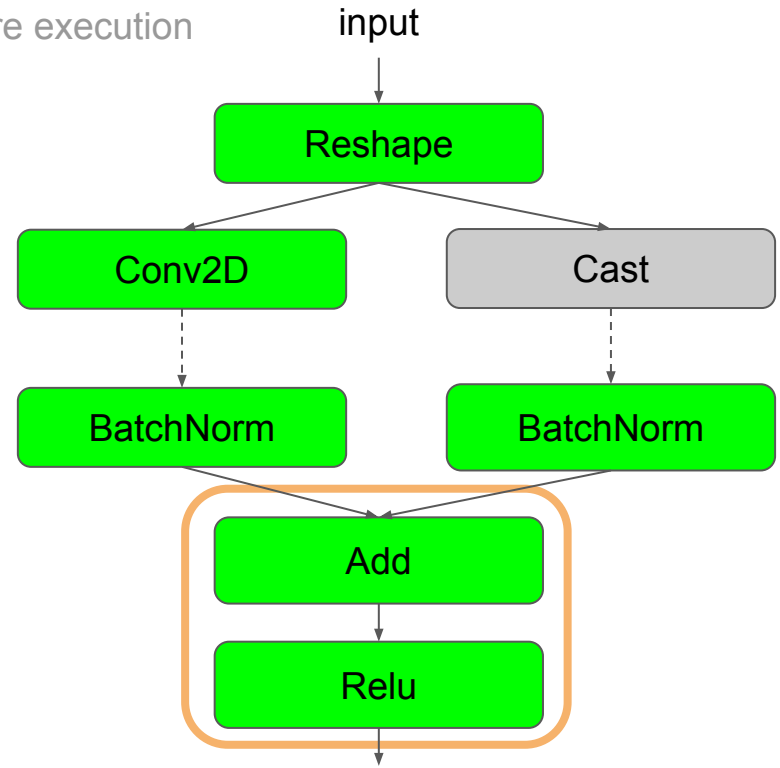


- Cluster nodes into TRT-compatible subgraphs
- The result should be a direct acyclic graph (DAG)
- Doesn't create circular dependency

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: cluster marked nodes

Before execution

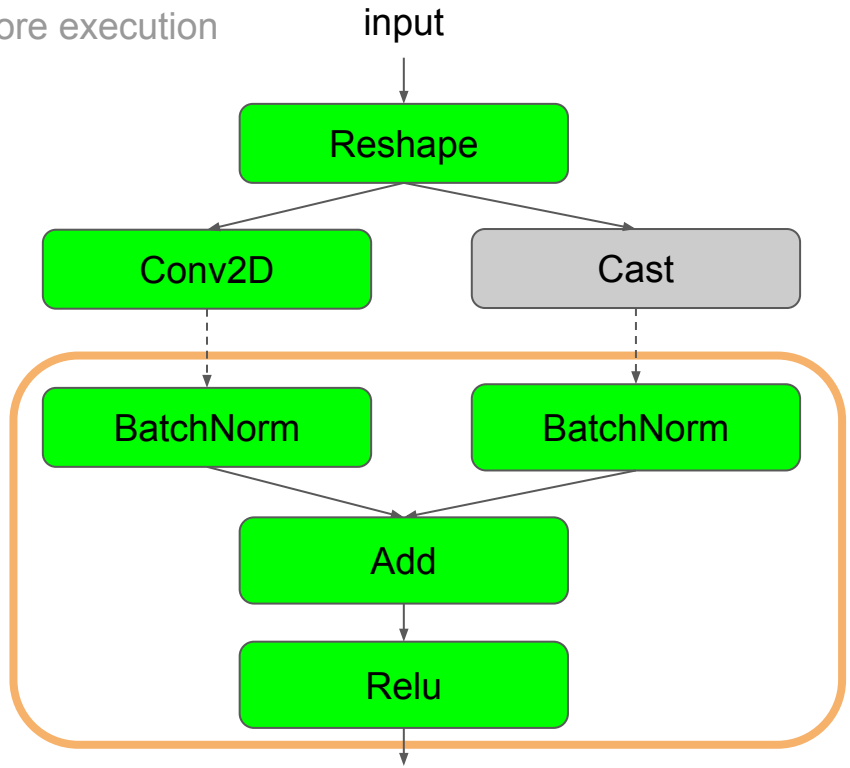


- Cluster nodes into TRT-compatible subgraphs
- The result should be a direct acyclic graph (DAG)
- Doesn't create circular dependency

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: cluster marked nodes

Before execution

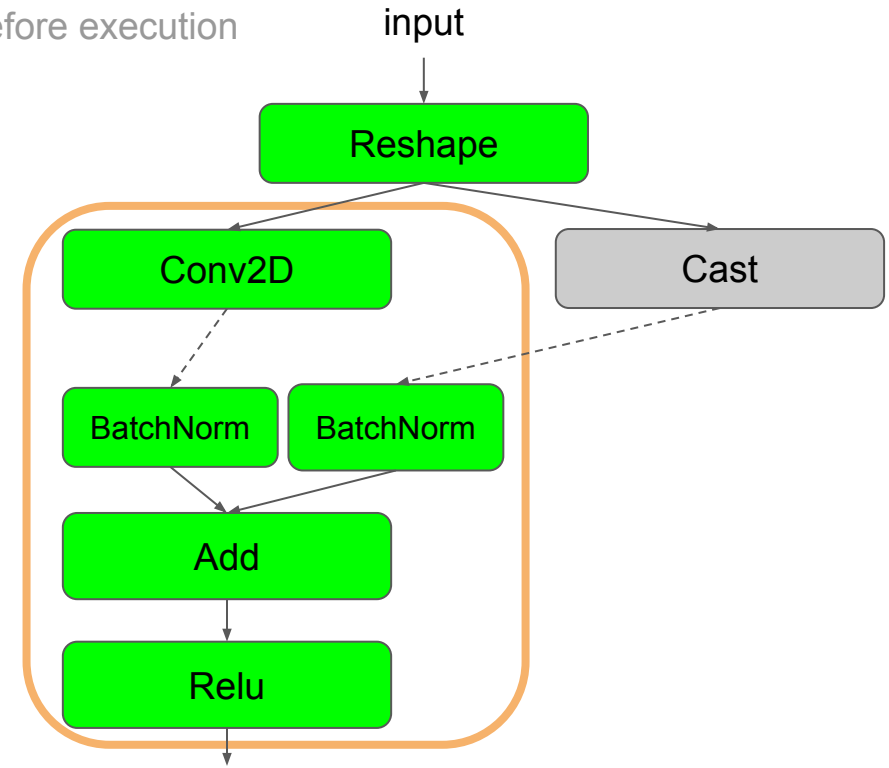


- Cluster nodes into TRT-compatible subgraphs
- The result should be a direct acyclic graph (DAG)
- Doesn't create circular dependency

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: cluster marked nodes

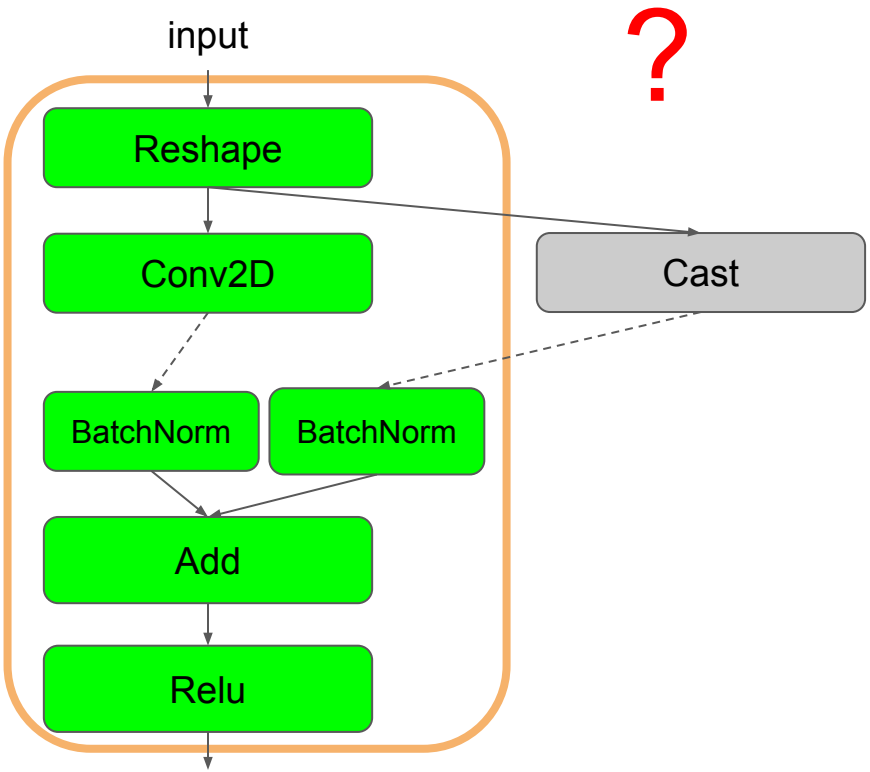
Before execution



- Cluster nodes into TRT-compatible subgraphs
- The result should be a direct acyclic graph (DAG)
- Doesn't create circular dependency

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

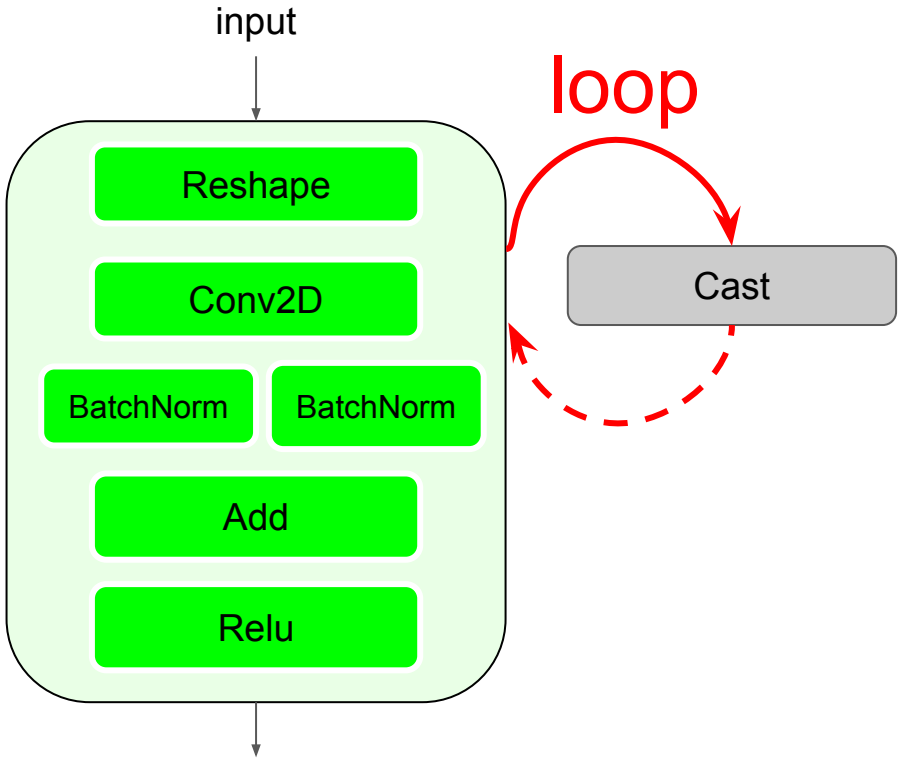
Phase 1: cluster marked nodes



- Cluster nodes into TRT-compatible subgraphs
- The result should be a direct acyclic graph (DAG)
- Doesn't create circular dependency

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: cluster marked nodes

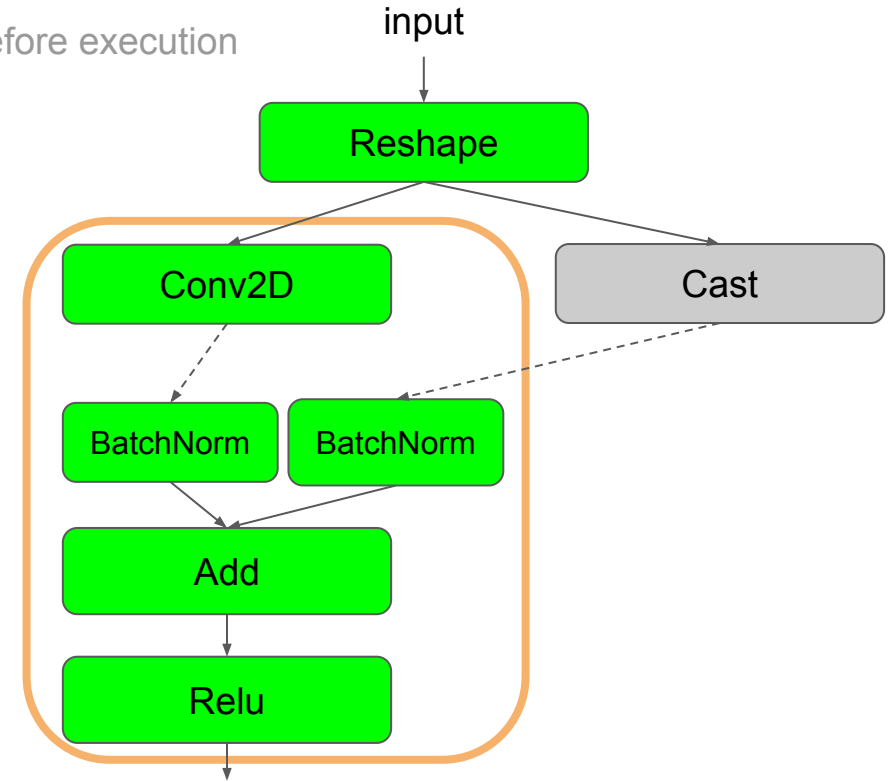


- Cluster nodes into TRT-compatible subgraphs
- The result should be a direct acyclic graph (DAG)
- Doesn't create circular dependency

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: cluster marked nodes

Before execution

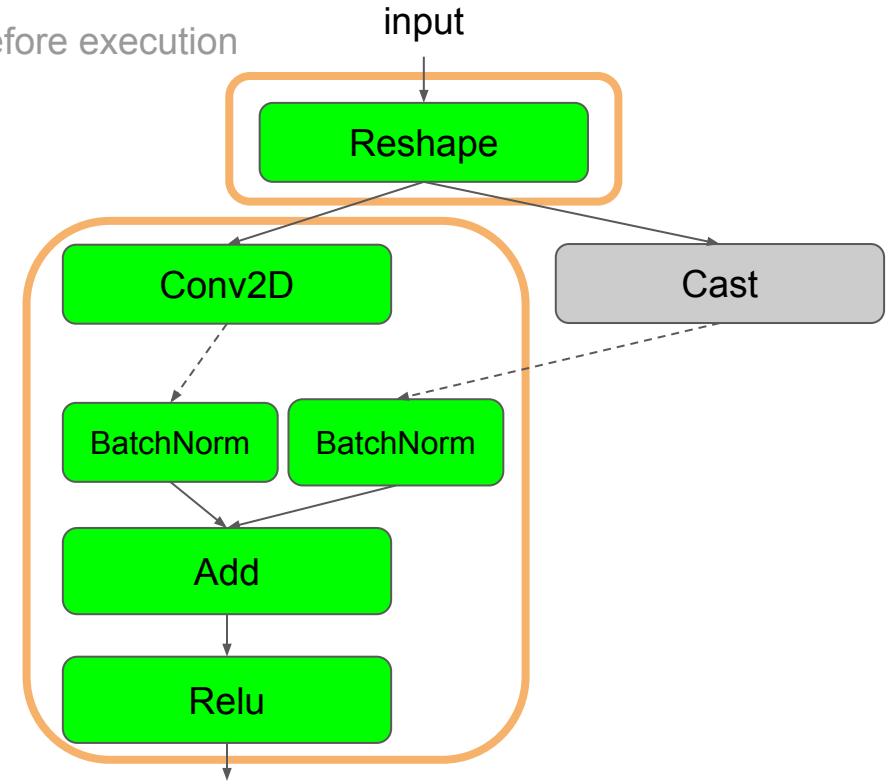


- Cluster nodes into TRT-compatible subgraphs
- The result should be a direct acyclic graph (DAG)
- Doesn't create circular dependency

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: cluster marked nodes

Before execution

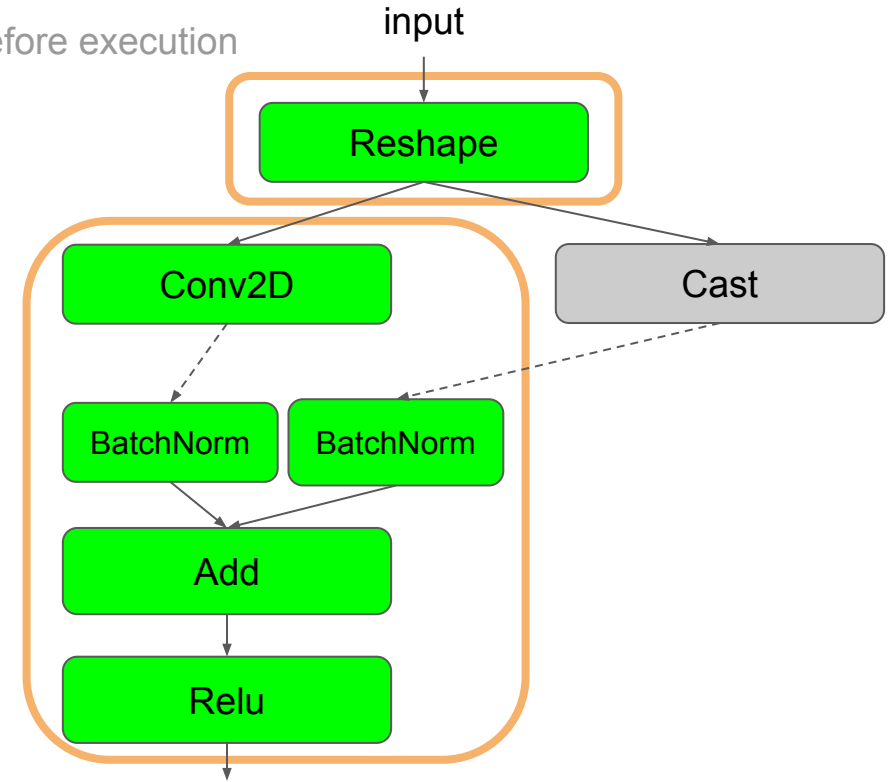


To break the loop: create separate clusters

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: remove small clusters

Before execution



Drop clusters with #nodes less than minimum_segment_size.

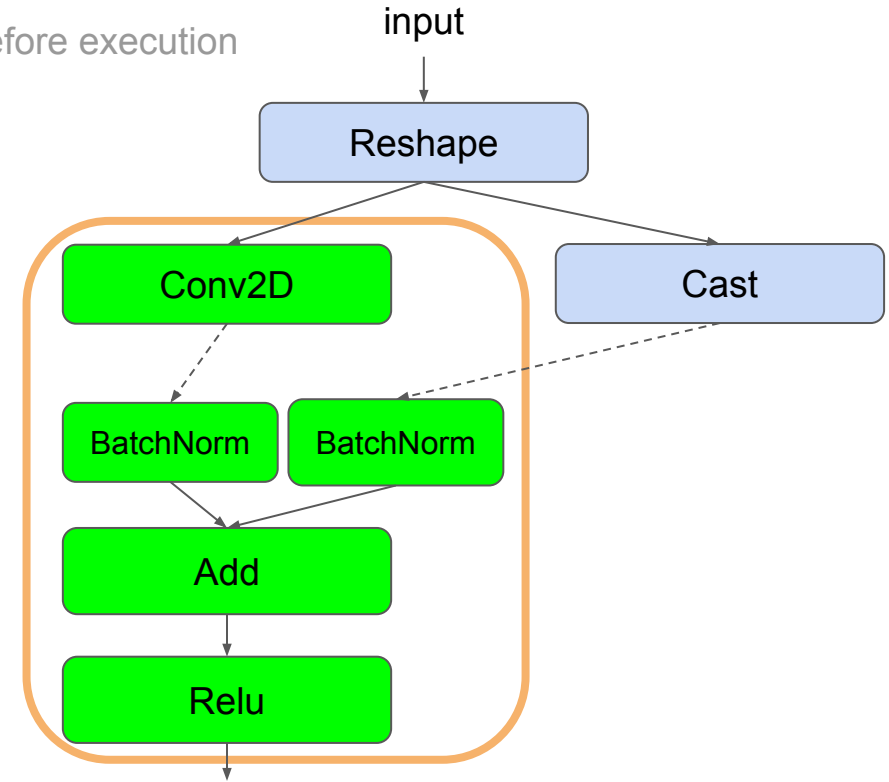
Trade-off:

- Too small: overheads of too many clusters (e.g. extra memcpy to cast dtype)
- Too large: missing TRT optimizations

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: partition result

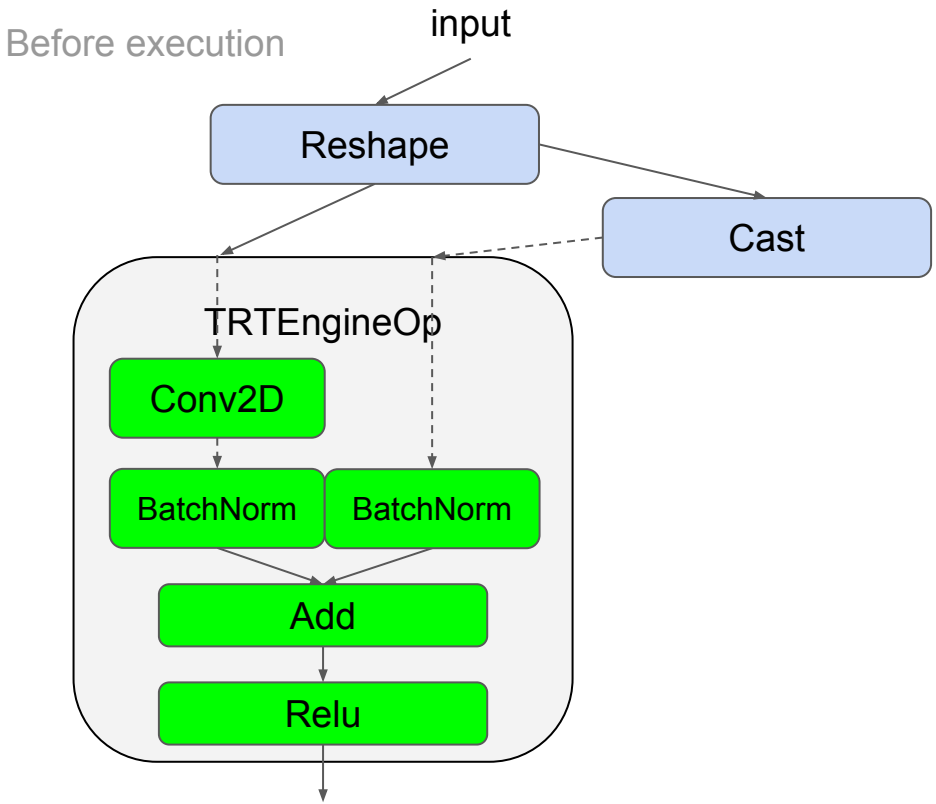
Before execution



The cluster with Reshape is dropped.

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

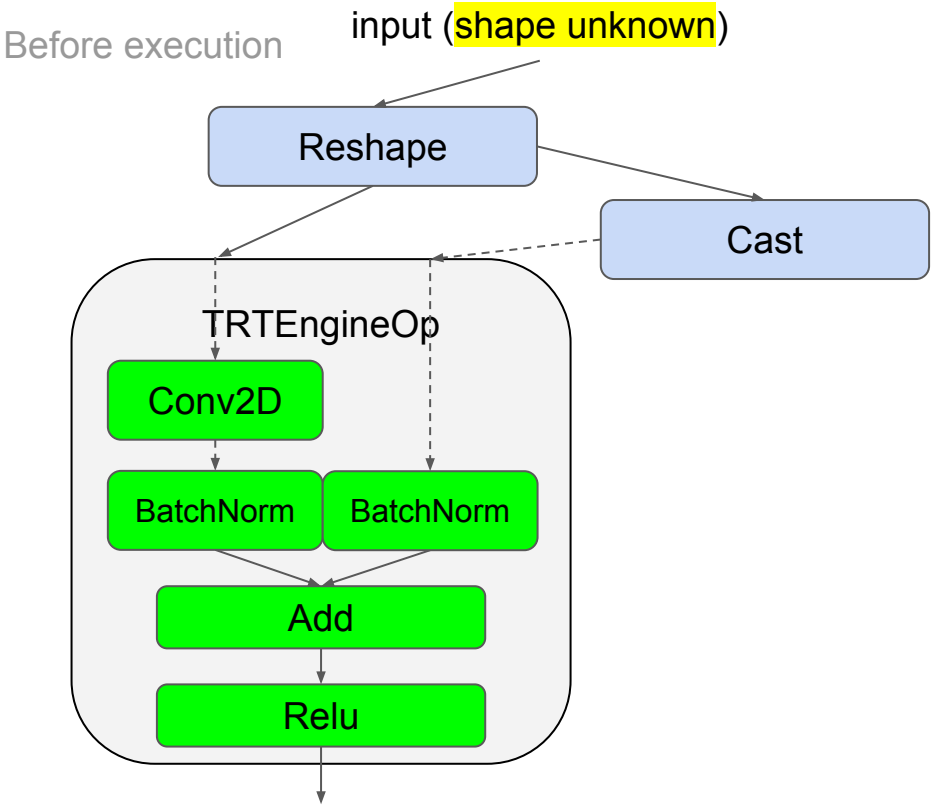
Phase 1: create TRTEngineOp



- Wrap the TRT-compatible subgraph in a custom op called TRTEngineOp
- Use the new op to replace the subgraph

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

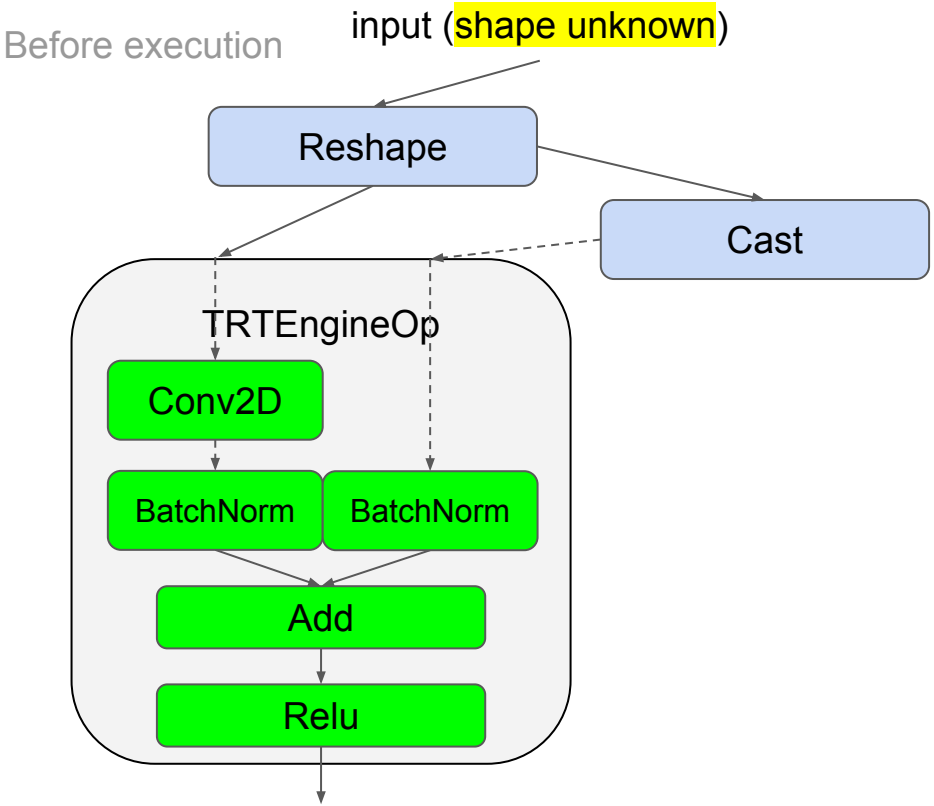
Phase 1: handle unknown shapes



- Input shape are still unknown
- Unknown shapes are common in TensorFlow graphs, e.g.
  input = tf.placeholder(
    tf.float32, shape=[None, None])
- Challenge: TRT requires known shapes when building the network

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 1: handle unknown shapes



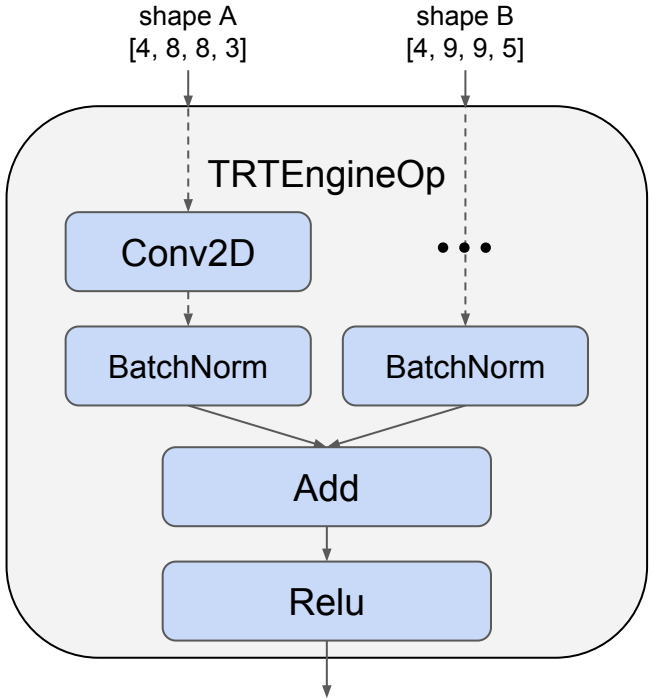
Two solutions:

- Make all the shapes known (use graph with full shapes specified, may require extra work)
- Postpone TensorRT optimization to execution phase, when shapes will be fully specified (is_dynamic_op=True. Default is False)

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 2: create TRT network

During execution

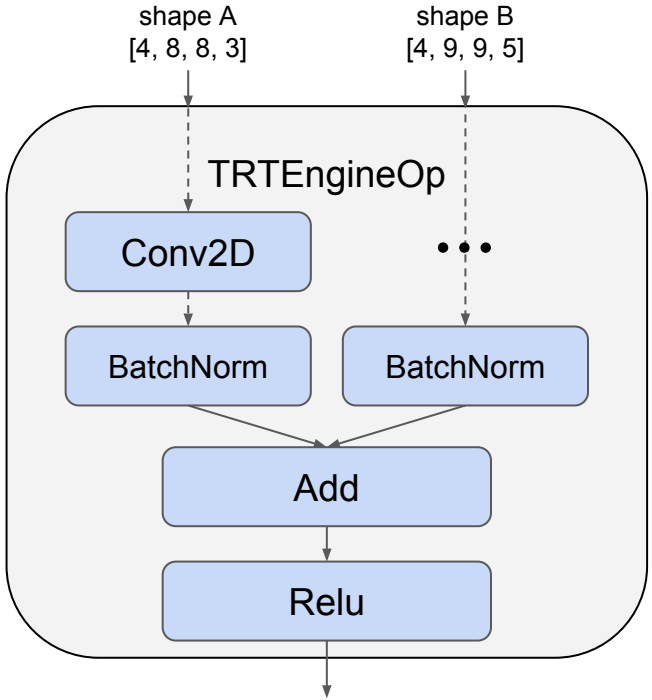


Input shapes are fully specified at runtime

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 2: TRT engine cache

During execution

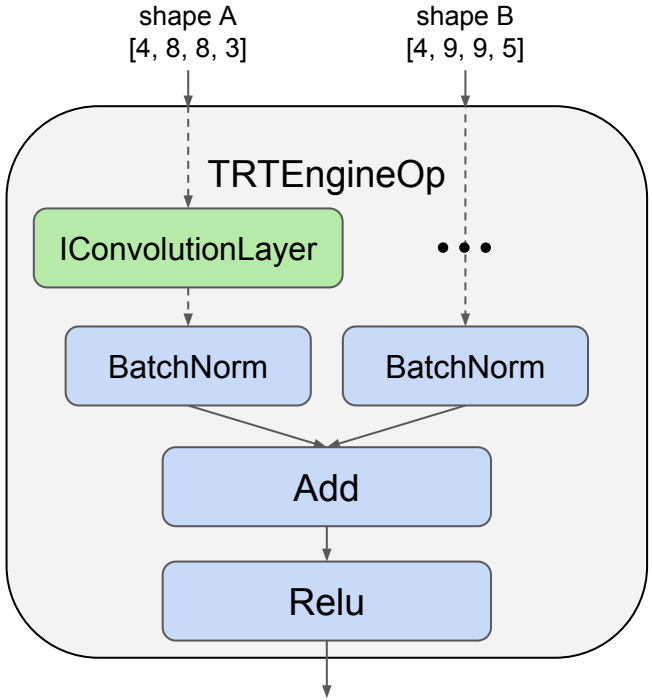


- There is an LRU engine cache in TRTEngineOp
- Keys of the cache are input shapes
- If cache miss, build a new engine
- If cache is full, evict an old engine

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 2: TF ops to TRT layers conversion

During execution

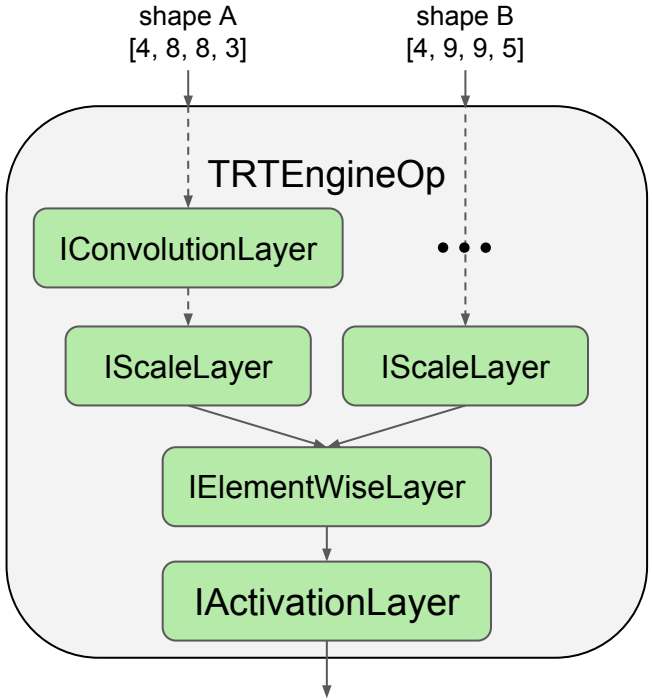


- Traverse the nodes in topological order
- Each TF node is converted to one or more TRT layers

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 2: TF ops to TRT layers conversion

During execution



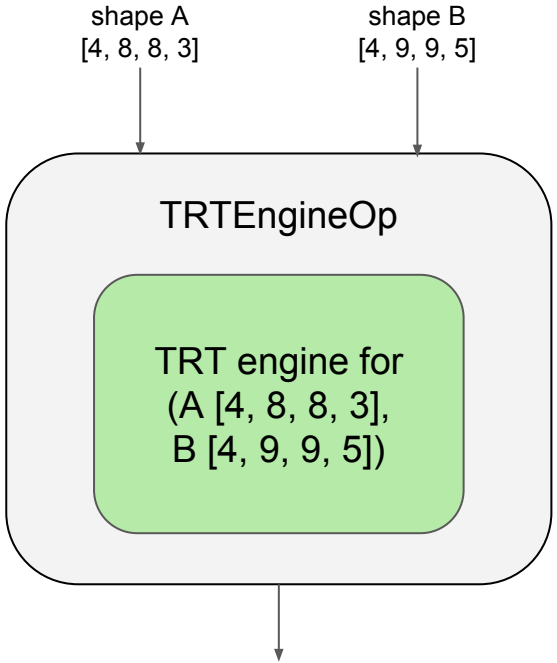
Finishing TRT network creation.

Next: build TRT engine (phase 3)

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Phase 3: build TRT engine

During execution



Optimization from TensorRT library

- Layer & Tensor fusion
- Precision calibration
- Kernel auto-tuning

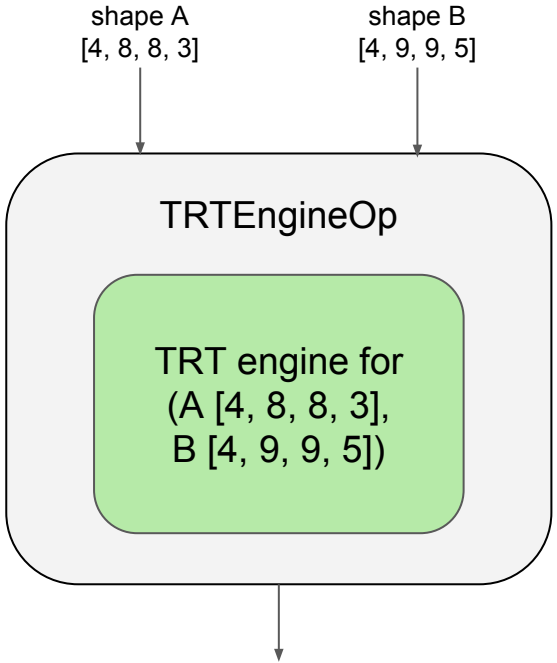
These optimizations:

- Invisible to user
- Applied to current GPU

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

TRT batch dimension

During execution



TF tensors: all dimensions are treated similarly

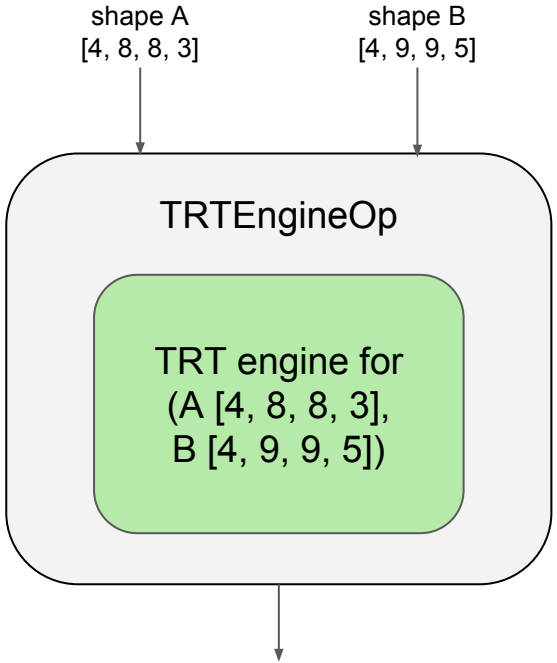
TRT:

- First dimension is special, called “batch dimension”
- TRT uses batch dim for optimizations

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

TRT batch dimension

During execution



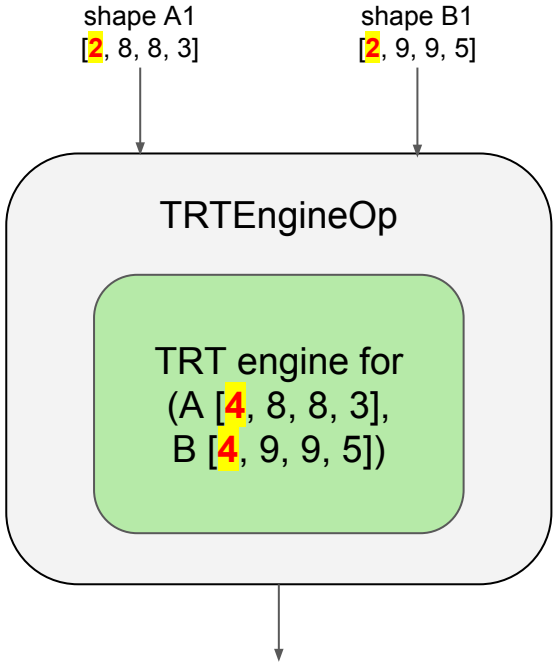
Batch dimension is determined by:

- Input shapes during execution (when is_dynamic_op=True, like this case)
- max_batch_size parameter (when is_dynamic_op=False, not listed here)

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Handle different batch dimensions

During execution



New inputs with a different batch dimension.

We can reuse an engine for a new input, if:

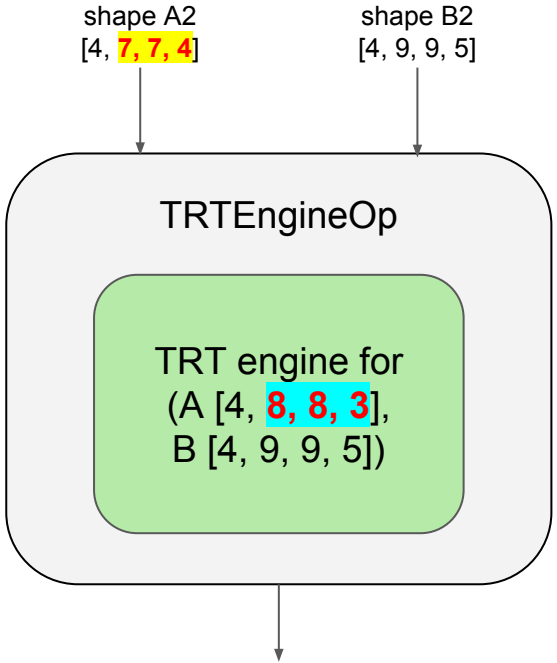
- engine batch size >= batch dim of new input, and
- non-batch dims match the new input

Otherwise: redo phase 2&3

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Handle different input shapes

During execution

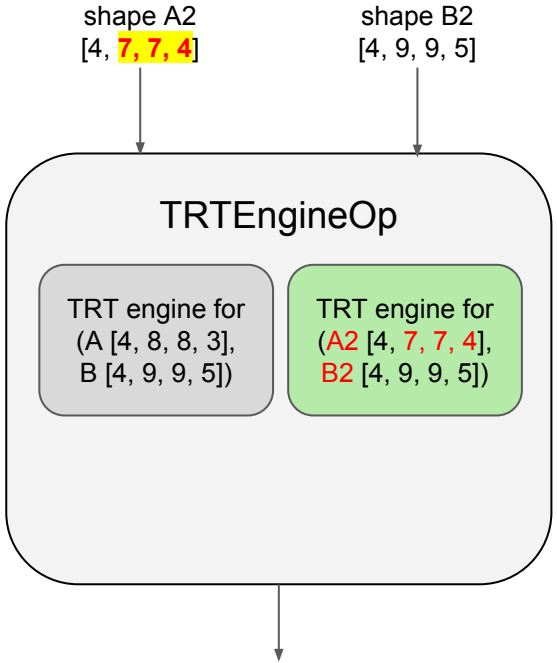


New inputs with different shapes
(different non-batch dimensions)

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Handle different input shapes

During execution



- Cache is full, evict old engine
- Use larger maximum_cached_engines to avoid that.
- Will consume more CPU/GPU resource, but usually not a problem in practice

```
converter = trt.TrtGraphConverter(
    input_saved_model_dir=model_dir,
    minimum_segment_size=3,
    is_dynamic_op=True,
    maximum_cached_engines=1)
```

Future of TF-TRT

- Dynamic shapes
  - Certains tensors have variable shape (NLP)
- TF 2.0 for calibration
- Support for more TF ops and models
  - Faster-RCNN, Mask-RCNN
  - Neural Collaborative Filtering
  - NLP: Transformer, BERT

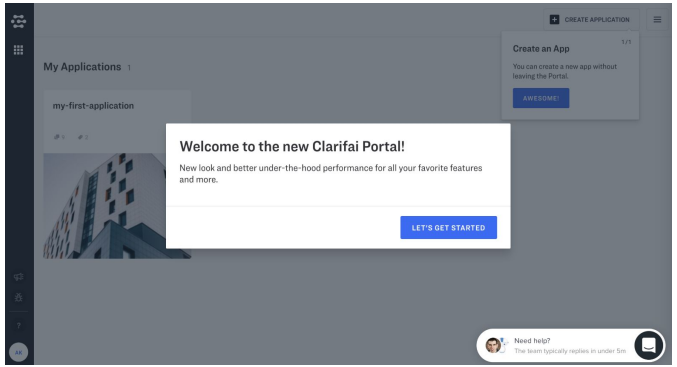

clarifai

About Clarifai

- Founded by Matt Zeiler in 2013
- SF Office - Clarifai Research
- DC Office - Public Sector
- 90+ employees

- $40M+ in Venture Capital Funding

Menlo · USV · LUX · Qualcomm Ventures · NVIDIA · Google Ventures



- Image and video recognition
- Clarifai Portal NEW
- On-prem deployment
- Edge/ Mobile SDK

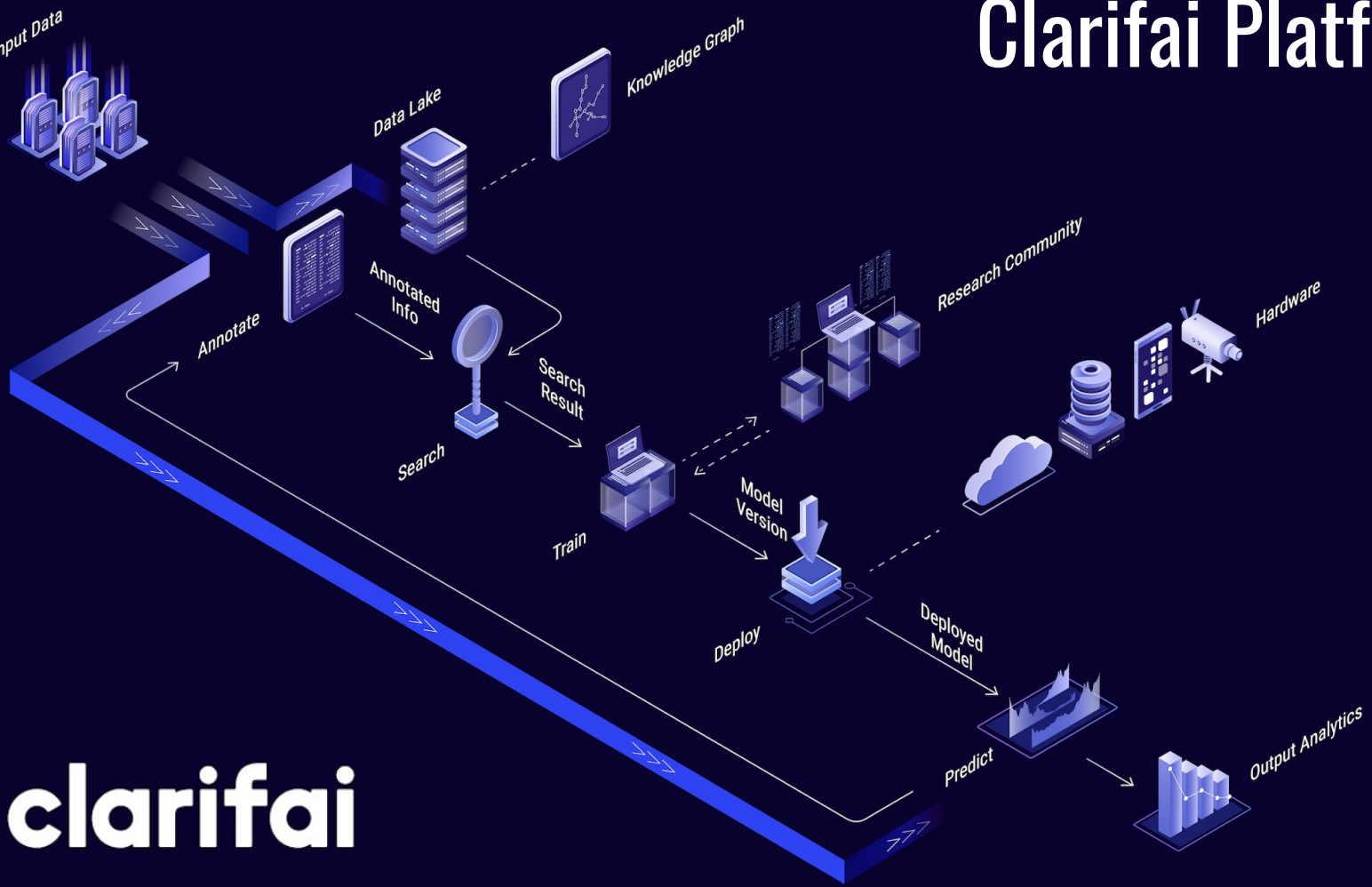
clarifai

Clarifai Models

- General Model - v1.5
- Demographics
- Color
- Moderation / NSFW
- Retail Analytics
- Public Safety
- Face Detection/Recognition
- Aerial
- Satellite

clarifai

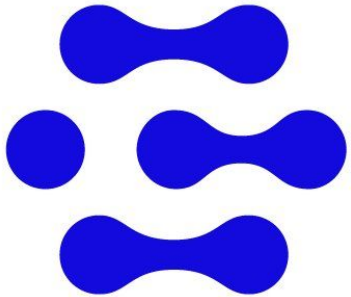
Clarifai Platform



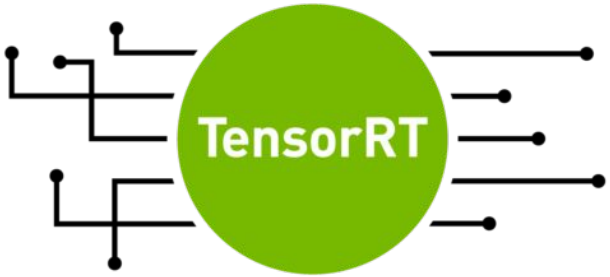
clarifai

Why TensorRT?

- Process images faster! Often need to trade off between speed and accuracy
  - Use case for public sector work: Need object detectors to work real-time for full motion video



- Take advantage of NVIDIA suite of tools, including DeepStream, NVIDIA Inference Engine



- Edge processing with NVIDIA Xavier
- Started with our latest General Model (version 1.5)

clarifai

Speed Performance using (TF-)TRT

- Started with TF-TRT
- Converted our General v1.5 model
- Over 3x speedup over our native TF frozen graph with minimal modifications
- Over 3x decrease in latency

Frames Per Second				
Batch Size	Native TF	TF-TRT fp32	TF-TRT fp16	TF-TRT int8
1	67.5 (1x)	187.0 (2.8x)	225.6 (3.3x)	303.9 (4.5x)
4	226.0 (1x)	464.0 (2.1x)	718.6 (3.2x)	721.7 (3.2x)
8	319.2 (1x)	590.5 (1.8x)	949.2 (3.0x)	1017.0 (3.2x)
16	410.6 (1x)	743.9 (1.8x)	1220.3 (3.0x)	1334.0 (3.2x)
Latency (ms)				
1	14.8 (1x)	5.35 (2.8x)	4.43 (3.3x)	3.29 (4.5x)
4	17.7 (1x)	8.62 (2.1x)	5.57 (3.2x)	5.54 (3.2x)
8	25.1 (1x)	13.6 (1.8x)	8.43 (3.0x)	7.87 (3.2x)
16	39.0 (1x)	21.5 (1.8x)	13.1 (3.0x)	12.0 (3.2x)

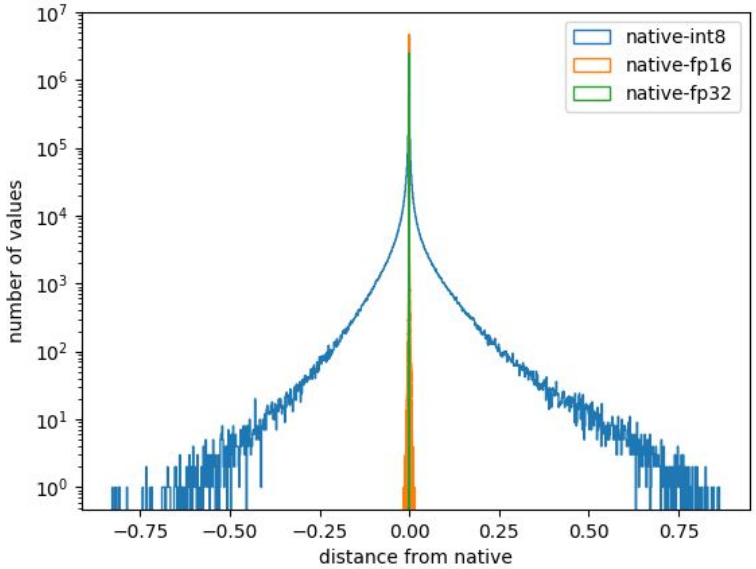
Speed Performance using TRT

- Converted our General v1.5 model directly to TRT via Universal Framework Format (UFF)
- Required 2 custom plugins (courtesy of NVIDIA)
  - StridedSlice
  - Pad
- ~5x speedup over our native TF frozen graph

Batch Size	Native TF	TRT fp32	TRT fp16
1	67.5 (1x)	257.2 (3.8x)	332.7(4.9x)
4	226.0 (1x)	592.4 (2.6x)	1050.1 (4.6x)
8	319.2 (1x)	805.7 (2.5x)	1591.2 (5.0x)
16	410.6 (1x)	972.4 (2.3x)	2046.7 (5.0x)

Results Metrics using (TF-)TRT

- Compared effects on accuracy from using TRT
- Comparison of values from each element of the sigmoid layer (11k per image)
- ~550 images

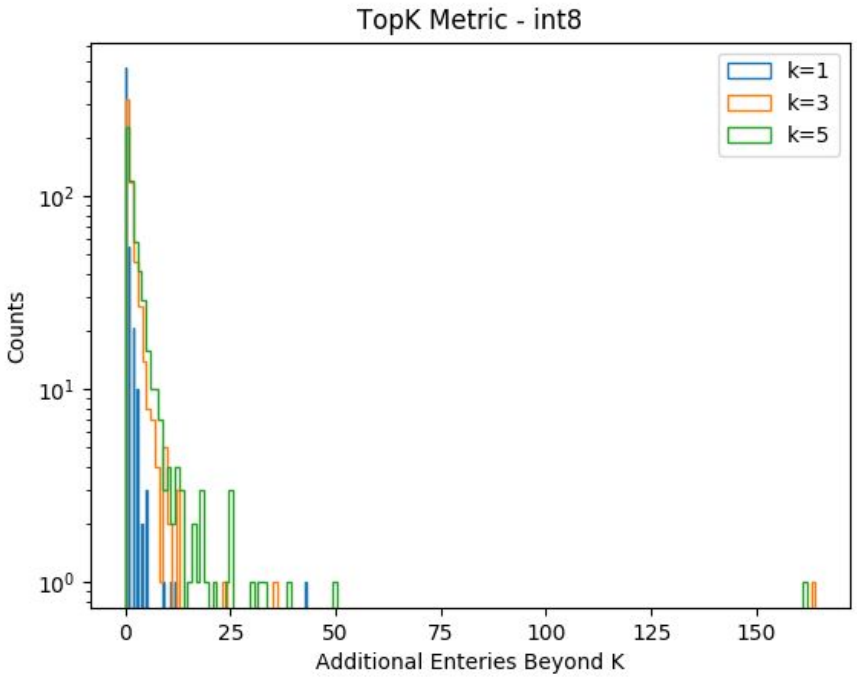


	Min	Max	Mean
Native-FP32	-6.4e-6	5.6e6	5.5e-8
Native-FP16	-0.016	0.016	8.4e-5
Native-INT8	-0.83	0.86	0.0050

Results Metrics using (TF-)TRT (cont')

- Top-K recall - how many elements do we need to include from the TRT result to obtain the Top-K from our native TF graph
- FP32 results were identical
- FP16 mostly agreed, with +3 as the largest discrepancy
- Int8 had the most discrepancy

Int8	Max	Mean
Top-1	55	0.4
Top-3	118	1.4
Top-5	122	2.7



clarifai

Example Results

Jon Howe
NVIDIA

Clarifai fp32	TFTRT fp32	TFTRT fp16	TFTRT int8
child: 0.990 cute: 0.988 cheerful: 0.972 outdoors: 0.970 fun: 0.969 portrait: 0.968 summer: 0.949 happiness: 0.946 people: 0.925 nature: 0.922	child: 0.990 cute: 0.988 cheerful: 0.972 outdoors: 0.970 fun: 0.969 portrait: 0.968 summer: 0.949 happiness: 0.946 people: 0.925 nature: 0.921	child: 0.990 cute: 0.988 cheerful: 0.972 outdoors: 0.969 fun: 0.968 summer: 0.948 portrait: 0.948 happiness: 0.945 people: 0.924 nature: 0.922	child: 0.991 outdoors: 0.980 portrait: 0.976 cute: 0.975 fun: 0.974 nature: 0.966 summer: 0.959 happiness: 0.958 cheerful: 0.955 people: 0.950

clarifai

More Example Results

Eran Nussinovitch
Clarifai

Clarifai fp32	TFTRT fp32	TFTRT fp16	TFTRT int8
market: 1.000	market: 1.000	market: 1.000	market: 1.000
stall: 1.000	stall: 1.000	stall: 1.000	merchant: 0.999
merchant: 1.000	merchant: 1.000	merchant: 1.000	stall: 0.999
sell: 0.999	sell: 0.999	sell: 0.999	people: 0.998
people: 0.999	people: 0.999	people: 0.999	sell: 0.998
grow: 0.998	grow: 0.998	grow: 0.998	grow: 0.997
vendors: 0.996	vendors: 0.996	vendors: 0.996	vendors: 0.993
marketplace: 0.993	marketplace: 0.993	marketplace: 0.993	shopping: 0.990
shopping: 0.993	shopping: 0.993	shopping: 0.993	booth: 0.989
booth: 0.992	booth: 0.992	booth: 0.992	*stock*: 0.986

clarifai

Conclusions / Future Work

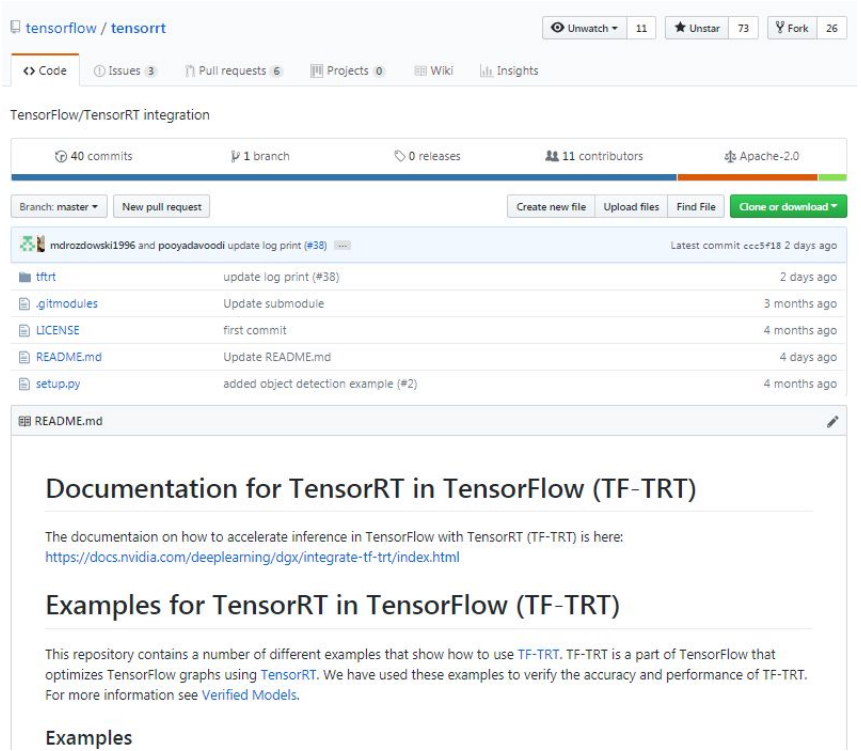
- Over 3x speed up and 3x decrease in latency with our General Model v1.5 using TF-TRT
  - Minimal effort/impact on existing setup
  - Greater speed up possible with some degradation in accuracy
- ~5x speed up with our General Model using TRT
  - More effort vs TF-TRT - needed some custom plugins
- Next steps - conversion of object detection model to TRT

TF-TRT Examples and documentation

Examples repository,
with links to documentation

https://github.com/tensorflow/tensorrt

- Image classification
  - MobileNet, NASNet, ResNet, VGG, Inception
- Object detection
  - SSD, Faster-RCNN, Mask-RCNN




Thank You