INT8 INFERENCE OF QUANTIZATION-AWARE TRAINED MODELS USING ONNX-TENSORRT

Presenters: Dheeraj Peri

CONTENTS

DEEP NEURAL NETWORKS QUANTIZATION

INTRODUCTION TO QUANTIZATION

0.34	3.75	5.64
1.12	2.7	-0.9
-4.7	0.68	1.43

FP32
(pre-quantized)

quantize

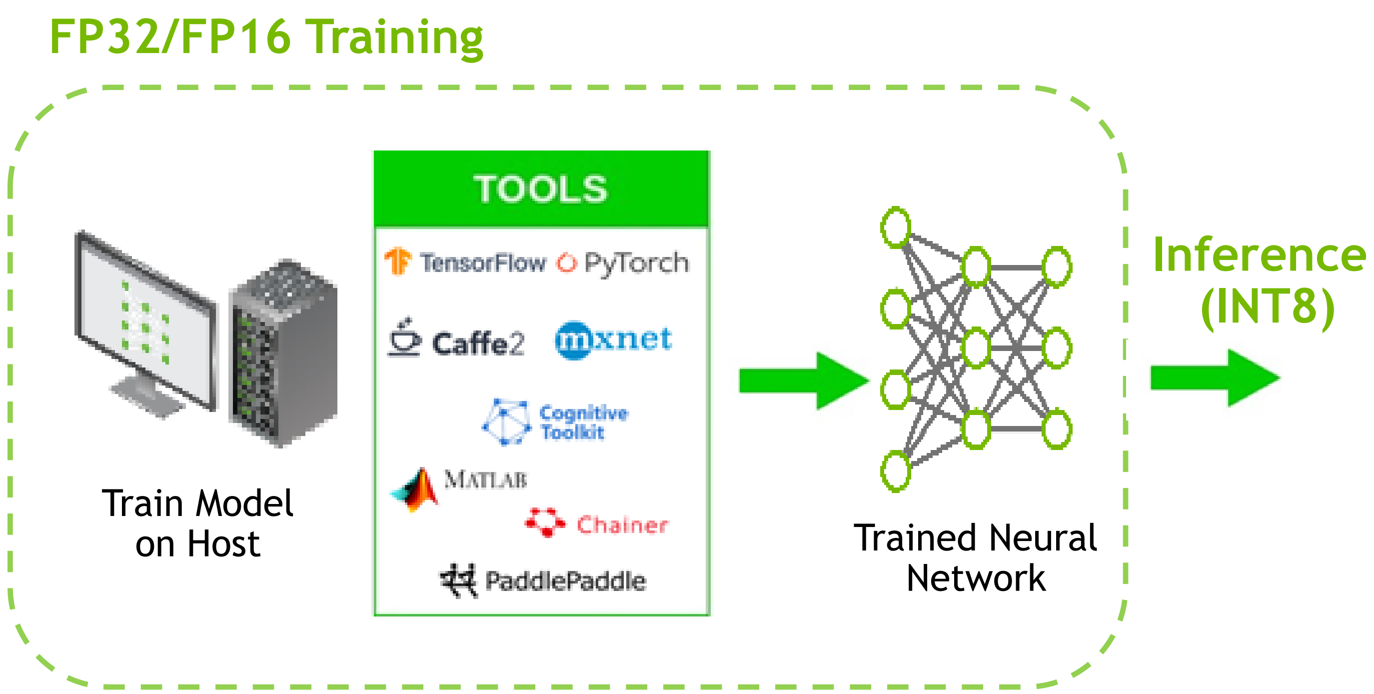
64	134	217
76	119	21
3	81	99

INT8
(quantized)

dequantize

0.41	3.62	5.29
1.3	2.8	-0.92
-4.5	0.71	1.39

FP32
(dequantized)



- What is Quantization?
    - Quantization is the process of converting continuous values to discrete set of values using linear/non-linear scaling techniques.
- High precision is necessary during training for fine-grain weight updates.
- High precision is not usually necessary during inference and may hinder the deployment of AI models in real-time and/or in resource-limited devices.
- INT8 is computationally less expensive and has lower memory footprint.
- INT8 precision results in faster inference with similar performance.

QUANTIZATION SCHEMES

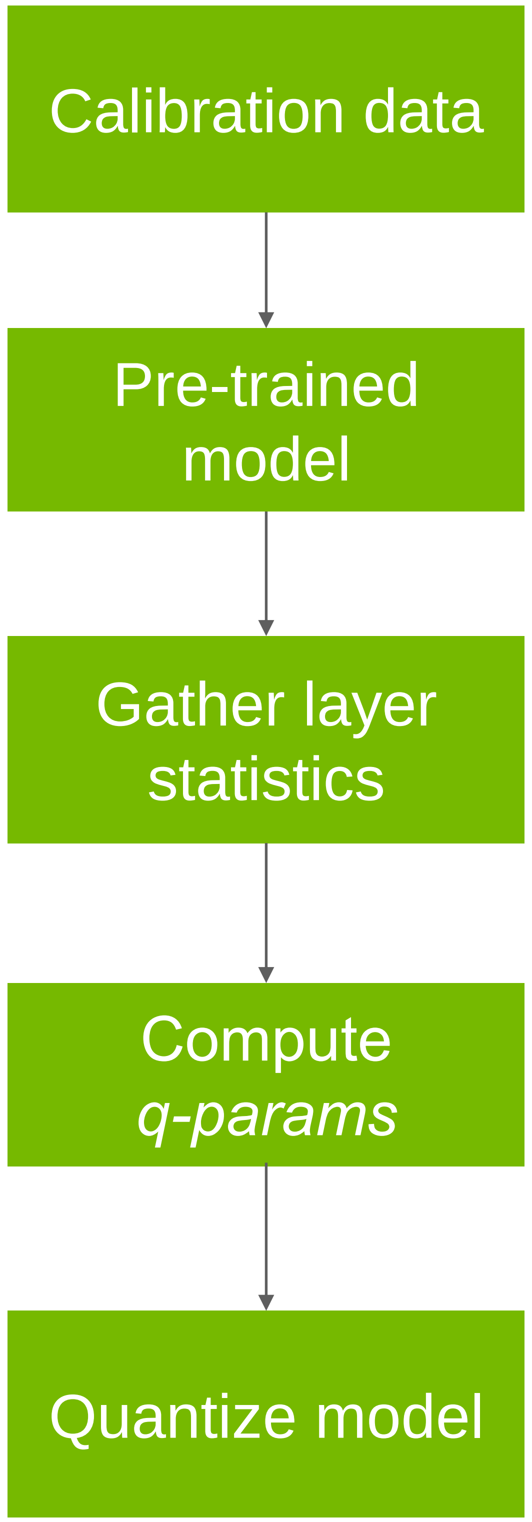
- Floating point tensors can be converted to lower precision tensors using a variety of quantization schemes.

  Example: R = **s * (**Q **- z**),
  where R is the real value,
  Q is the quantized value,
  s (scale) and z (zero-point) are the quantization parameters (*q-params*) to be determined

- *Q-params* can be determined with:
  - Post-Training Quantization (PTQ)
  - Quantization-Aware Training (QAT)

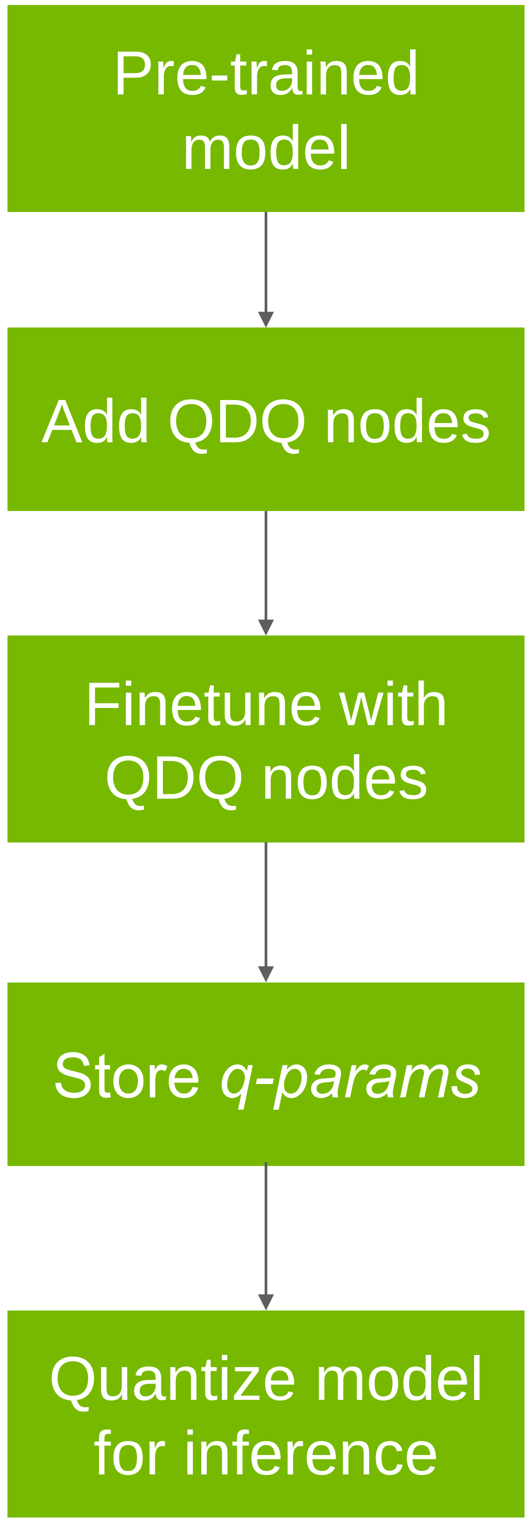
POST TRAINING QUANTIZATION (PTQ)

- Start with a pre-trained model and run it on a calibration dataset.
- Calibration data can be a subset of training or validation data.
- Calculate dynamic ranges of weights and activations in the network to compute quantization parameters (*q-params*).
- Quantize the network using *q-params* and run inference.



QUANTIZATION AWARE TRAINING (QAT)

- Start with a pre-trained model and introduce quantize and dequantize (QDQ) nodes at desired layers.
- Finetune it for a small number of epochs.
- Simulates the quantization process that occurs during inference.
- The goal is to learn the *q-params/model parameters* which can help to reduce the accuracy drop between the quantized model and pre-trained model.



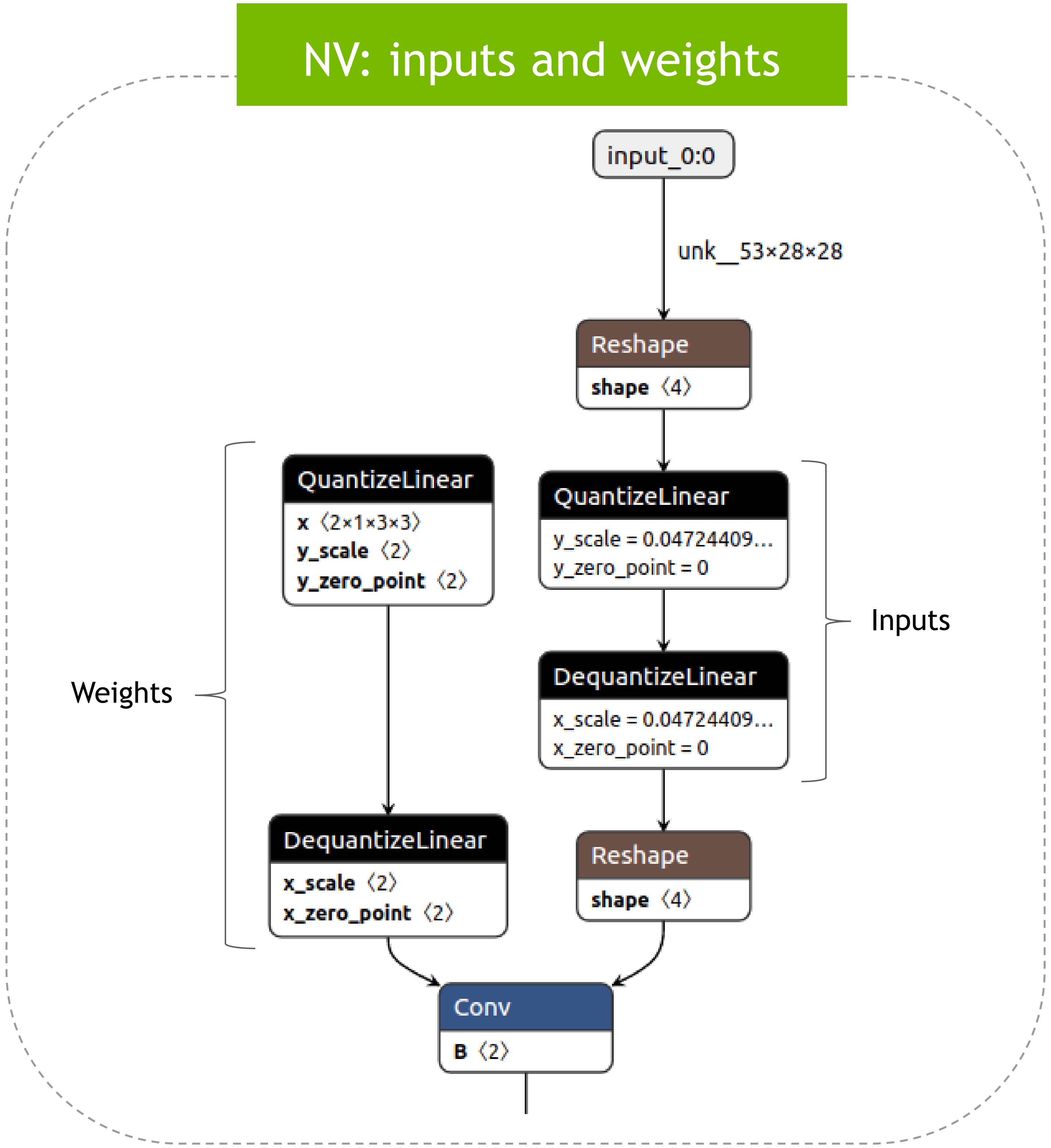
QAT FOR TENSORFLOW 2.0

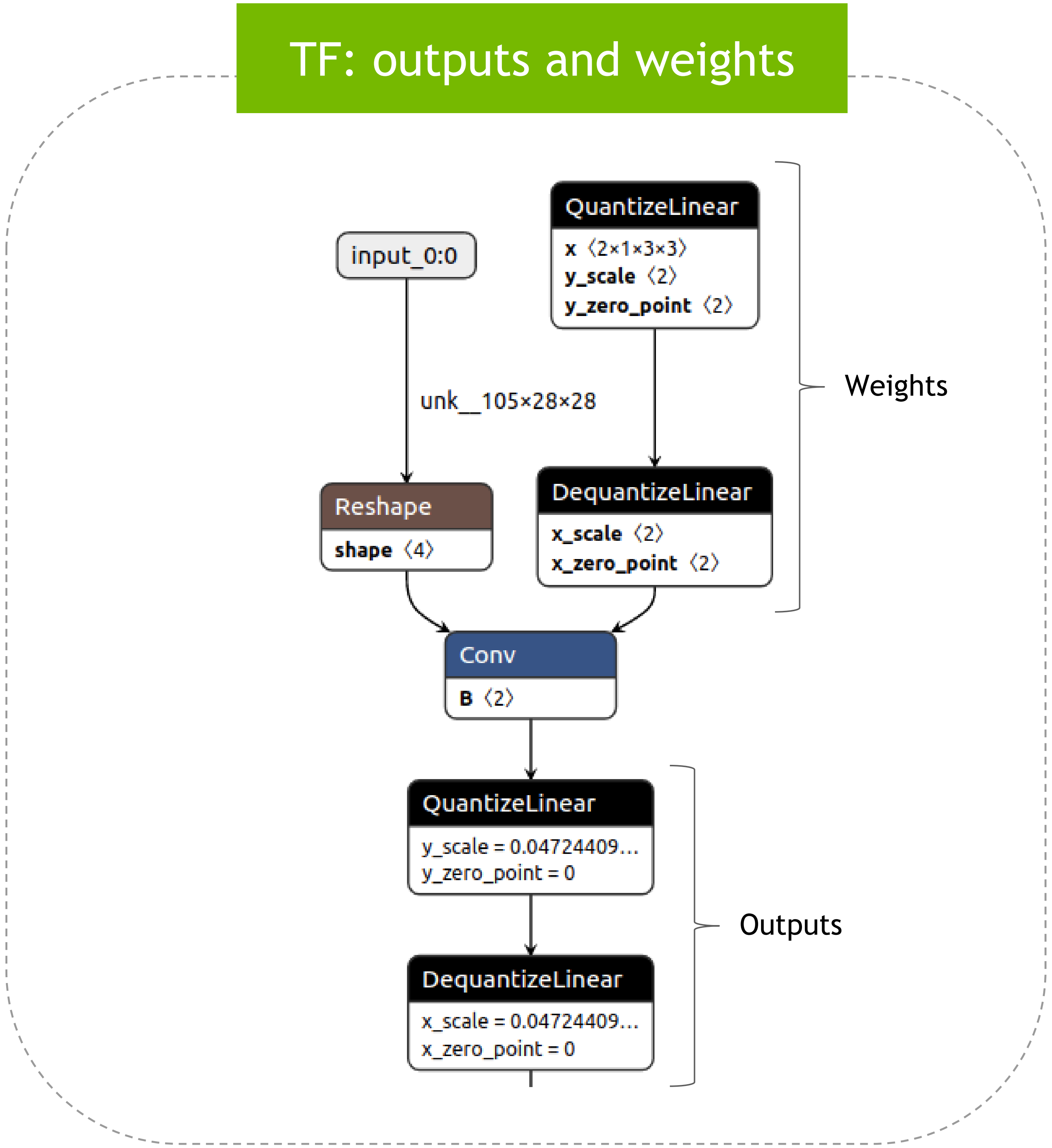
- **TFMOT (TensorFlow Model Optimization Toolkit)**
  - Implements TensorFlow quantization recipe designed for TensorFlow lite.
  - Supports quantization of the whole model (full) and of some layers (partial by layer class).
  - Quantization is performed using **tf.quantization.fake_quant_with_min_max_vars** op.

- **NVIDIA TF2 Quantization Toolkit**
  - Implements NVIDIA recommended quantization recipe optimized for TensorRT needed for model acceleration.
  - Offers new features of top of what TFMOT offers:
    - Quantize layers with both layer name or layer class as attributes.
    - Programmable, pattern-based quantization
    - Quantization is performed using **tf.quantization.quantize_and_dequantize_v2** op.
- **Note**: To get the best performance for a QAT model on a GPU using ONNX-TensorRT, we recommend using our **NVIDIA TF2 Quantization toolkit.**

QUANTIZATION RECIPES

NVIDIA vs. TensorFlow

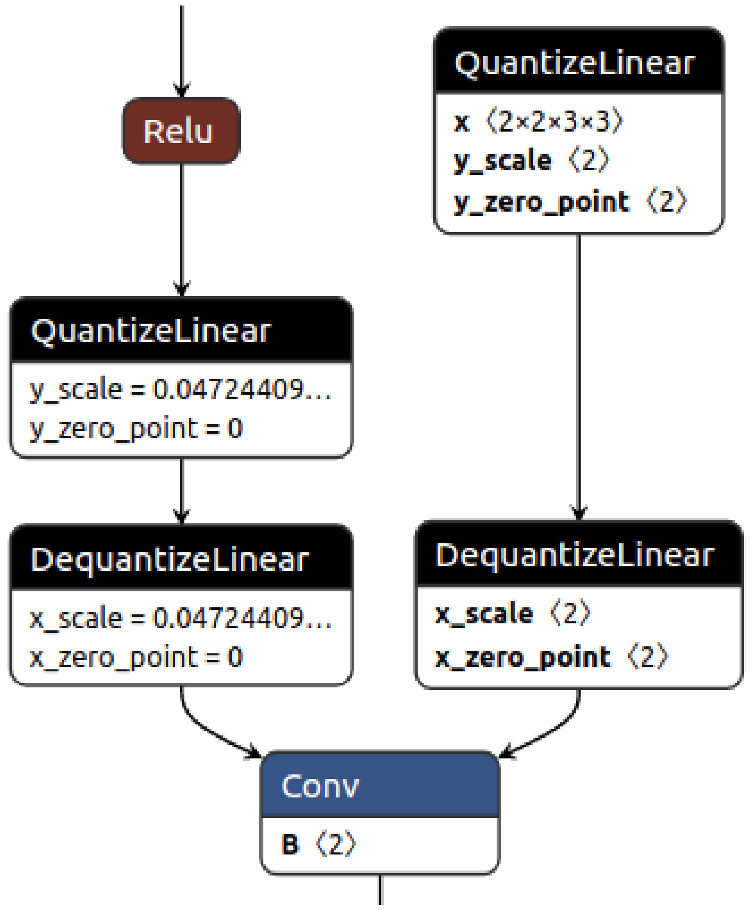




QDQ PLACEMENTS

According to TensorRT's recommendation

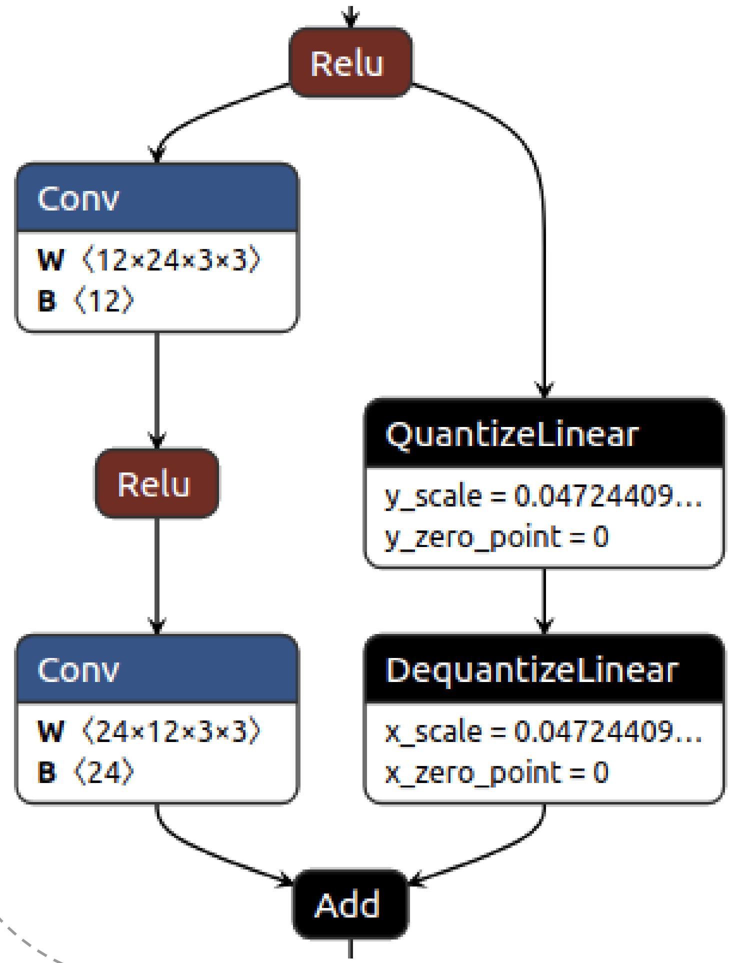
Conv/Dense



Weighted layers

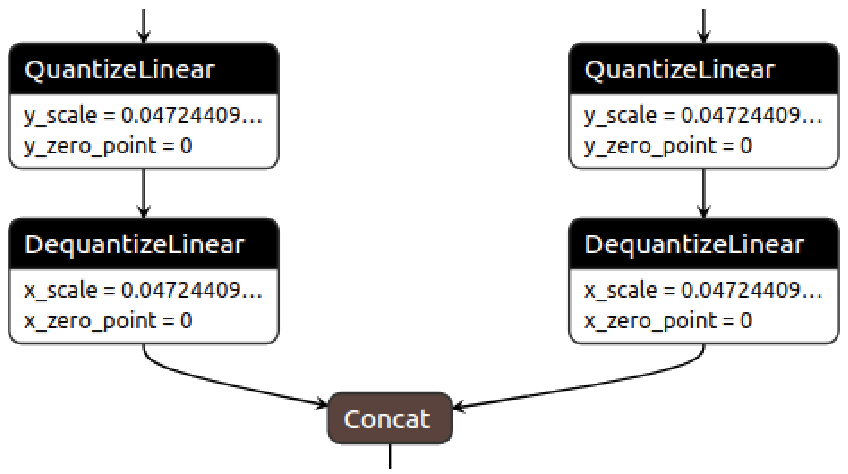
Add

Add to inputs without Conv (don't add to bias)



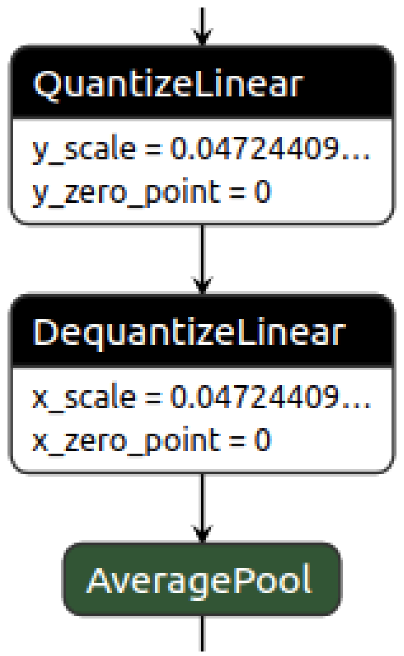
Concat

Add to all inputs



Non-weighted layers

Pooling



AvgPool, GlobalAvgPool, MaxPool

END-TO-END WORKFLOW

WORKFLOW FOR DEPLOYMENT

TF 2.0 Pretrained Model to INT8 TensorRT Engine

Pre-trained TensorFlow 2.0 model → Quantize model with Nvidia toolkit → Fine-tune and save quantized model → Convert TensorFlow 2.0 model to ONNX → Generate TensorRT INT8 engine from ONNX

TENSORFLOW MODEL QUANTIZATION

Quantization Aware Training using NVIDIA toolkit

NVIDIA Quantization and Fine-tuning

```
import tensorflow as tf
from tensorflow_quantization import quantize_model

x_train, x_test, y_train, y_test = DataLoader(....)          # Load dataset

model = MyModelClass(*args, **kwargs)                        # Define model structure
model.load_weights("pretrained_ckpt")

model = quantize_model(model)

# Compile and fine-tune quantized model
model.compile(optimizer="adam",
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
model.fit(x_train, y_train, batch_size=128, epochs=5, validation_split=0.1)

model.save_weights("quantized_finetuned_ckpt")               # Save quantized and fine-tuned model weights
```

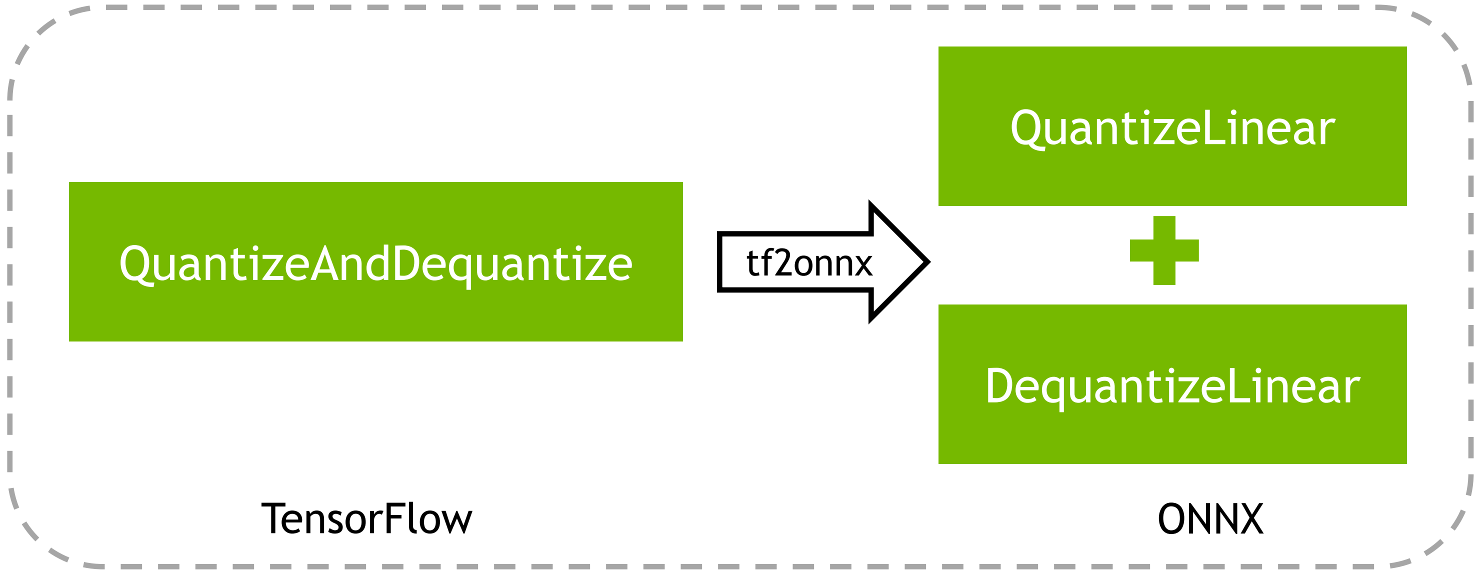
4 lines!

DEPLOYMENT WITH ONNX-TENSORRT

Conversion via ONNX

- Convert your finetuned QAT model to ONNX using *tf2onnx*.

- *tf2onnx* is a standard way to convert TF models into ONNX.
- It has conversion support for many standard DL operators. Support for quantization operators has been added.



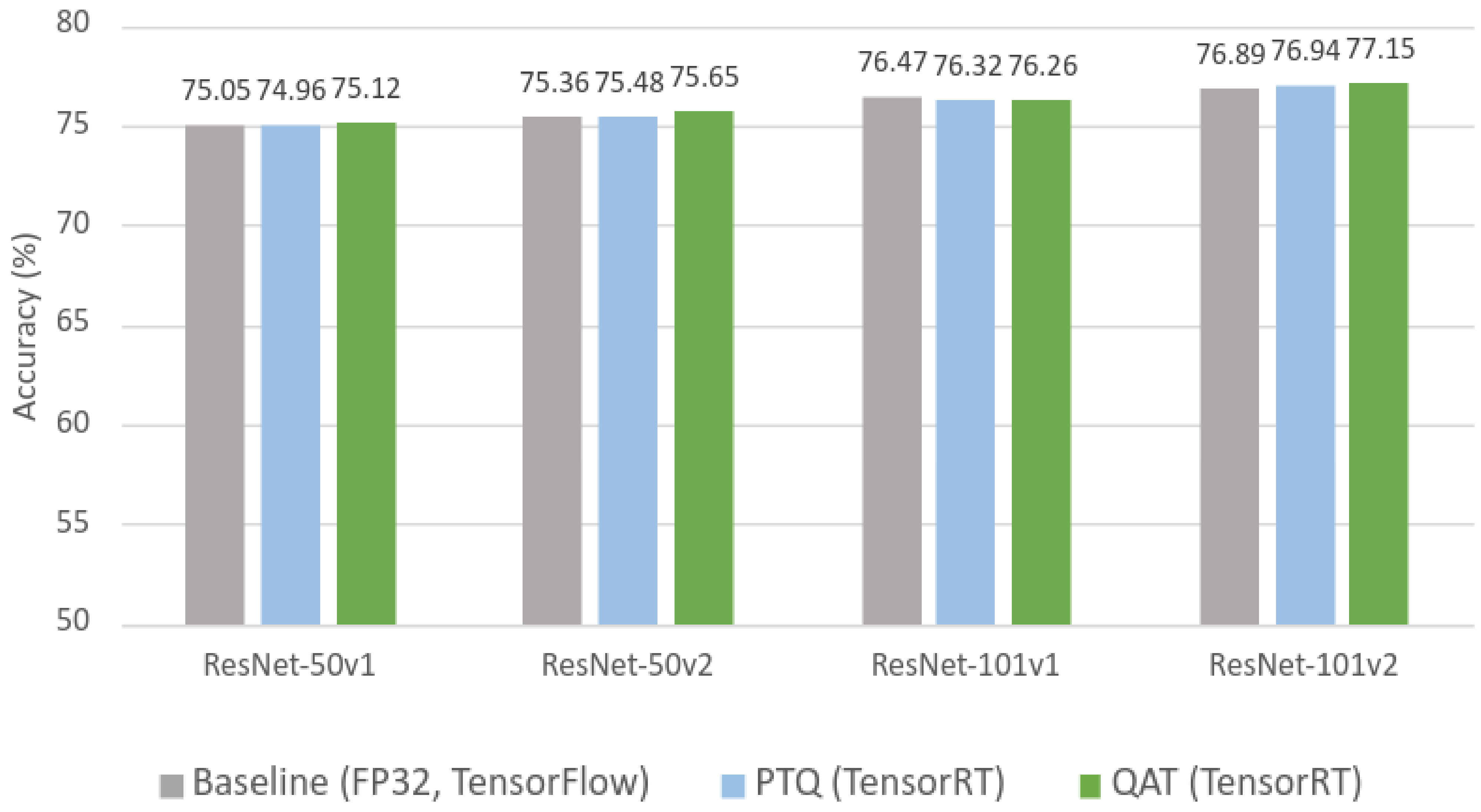
- Deploy the ONNX graph with TensorRT:
  - ONNX Parser in TensorRT parses the ONNX graph and converts into a TensorRT network definition. https://github.com/onnx/onnx-tensorrt
  - QDQ optimizations and fusions are performed to build an optimized TensorRT engine.

RESULTS

ACCURACY

Evaluating models in FP32 vs INT8 precision

TensorFlow 2.8, TensorRT 8.4-EA

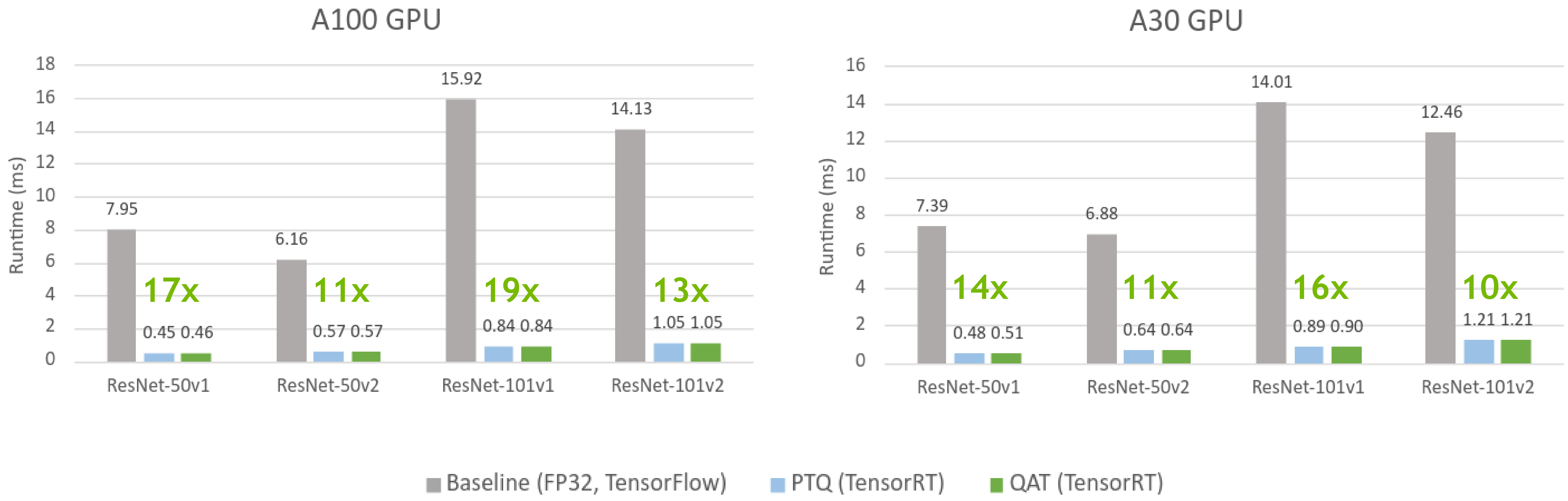


* See Table 7: https://arxiv.org/pdf/2004.09602.pdf

LATENCY

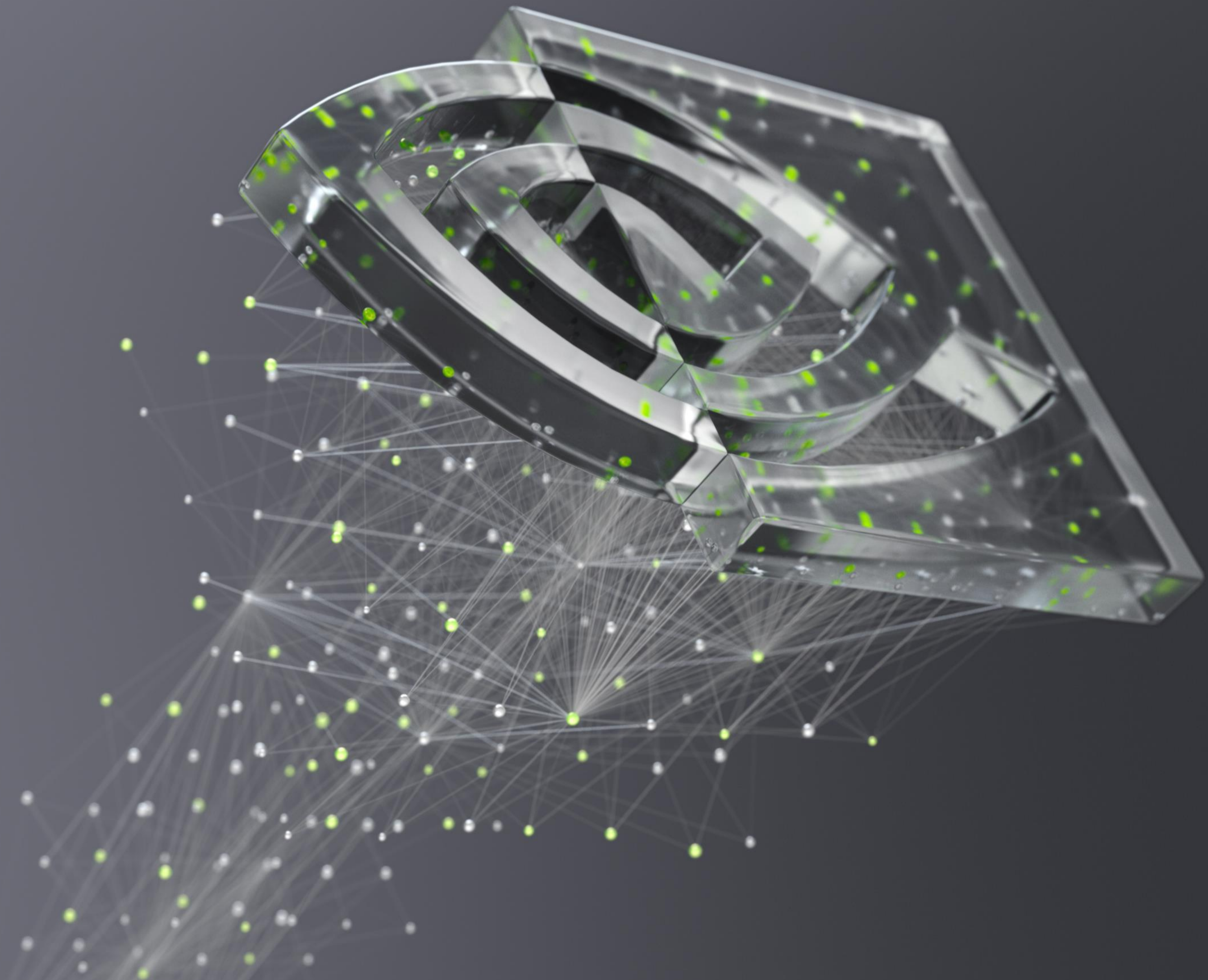
Evaluating models in FP32 vs INT8 precision

TensorFlow 2.8, TensorRT 8.4-EA



CONCLUSION

- Quantization-Aware Training provides an alternative to deploy deep neural networks in lower precision.
- QAT models might be less prone to accuracy drop during inference compared to PTQ models due to model-parameters fine-tuning.
- We demonstrated an end-to-end QAT workflow from TensorFlow to TensorRT deployment via ONNX. TF2ONNX enables converting TF models into ONNX graphs which is then optimized by TensorRT.
- Our experiments with ResNet models showed that the INT8 accuracy is on par with the FP32 baseline accuracy and that QAT latency is on par with PTQ



THANK YOU!