Author: Pavan Shetty, Northeastern University, Shepherd Lab

* This Document explains the procedure of converting a Tensorflow trained .h5 model to a .trt model.  
    
    
  **Folder Information: H5 to TRT**  
  Files for Training:
* testing\_max\_code.ipynb
* ml\_util.py

Files for trt conversion:

* tf\_trt.py

Please find H5 to TRT folder at: <https://github.com/Pavan-r-shetty/tensorflow_2_TensorRT.git>

Some Reading:

When you train your model, the model by default has INT64 weights. When you try to convert the model to a TensorRT model, it will try to cast the layers with INT64 to INT32 or Float  
In order for you to set float conversion while you train, include the following in your training script as seen below, you can fins this in testing\_max\_code.ipynb file

#trt stuff

from tensorflow.keras.mixed\_precision import set\_global\_policy, Policy

mixed\_precision\_policy = Policy('mixed\_float16')

set\_global\_policy(mixed\_precision\_policy)

You can avoid the above, it wouldn’t cause any problem, but the trt convertor warns you that it will try to convert whatever possible to INT32.   
Also INT32, INT64, INT8, FP16 are different types of quantization. In the end there is a tradeoff between speed vs accuracy.  
  
Typical .h5 to .trt conversion includes the following:  
  
.h5 trained model -> .onnx model -> .trt model  
  
  
The training script trains model with None Shape, which means you train with flexible window of Data (Time series window)  
But while you convert to trt, you need to fix the input shape by passing Optimization parameters as described below, you can also find this in tf\_trt.py:

# Optimization profile for dynamic input shapes

profile = builder.create\_optimization\_profile()

# Assuming the input tensor name is "input\_tensor" and its dynamic shape is [batch\_size, channels, height, width]

# Here's an example where we assume the batch size can vary between 1 and 32, and we optimize for a batch size of 16:

min\_shape = (1, 80, 8)

opt\_shape = (1, 80, 8)

max\_shape = (1, 80, 8)

You would need to use the set shape as above in all the future scripts, like inference test of the trt model or while utilizing the .trt file in general

profile.set\_shape("my\_input\_layer", min\_shape, opt\_shape, max\_shape)

config.add\_optimization\_profile(profile)

The above code with “my\_input\_layer” is specific to your trained model, it is the the name of the input tensor

Use the below to specify FP16 quantization model for your trt conversion, you can find this in tf\_trt.py too:  
 # Check if the platform supports FP16 mode

if builder.platform\_has\_fast\_fp16:

config.set\_flag(trt.BuilderFlag.FP16) # Set FP16 mode

else:

print("Warning: This platform does not have fast FP16 support.")

return None  
  
Whether you should use FP16 (half-precision floating point) or INT32 (32-bit integer) for training, especially with the intention of using TensorRT for inference, depends on several factors:

* Memory Consumption:
  + FP16: Uses 16 bits, so it consumes half the memory of a 32-bit representation. This means you can potentially double the batch size, fit larger models into memory, and benefit from faster memory access.
  + INT32: Uses 32 bits, which will consume more memory than FP16.
* Computational Speed:
  + FP16: Many modern GPUs have specialized hardware for FP16 computations, which can accelerate the training process. NVIDIA's Volta and newer architectures, for example, have Tensor Cores optimized for FP16 arithmetic, which can significantly speed up training.
  + INT32: It's less common for deep learning training to be optimized for INT32. Typically, INT8 (8-bit integer) is the target for integer optimizations, especially in the context of inference.
* Precision and Numeric Stability:
  + FP16: Has lower precision compared to FP32 and might introduce numerical instability in training, especially during gradient computations. Some models, especially those with batch normalization layers, may not train well with naive FP16 arithmetic due to this reduced precision. Mixed precision training, which uses both FP16 for the forward and backward passes and FP32 for weight updates, is a common workaround to this problem.
  + INT32: Isn't typically used for training neural networks because of the lack of precision in representing values between integers.
* TensorRT Compatibility:
  + TensorRT does support FP16 mode, and using it can offer substantial performance gains. This is often the preferred mode for deploying deep learning models for inference on platforms like NVIDIA's Jetson and recent GPUs.
  + For INT32, TensorRT primarily uses it for indices or certain layer parameters. The primary integer precision mode that TensorRT optimizes for is INT8 quantization, which requires a calibration step.
* Overall Recommendation for Training and TensorRT:
  + If you aim for faster training and intend to use TensorRT for inference, FP16 (mixed precision training) would be a more logical choice than INT32. FP16 can give you the speed benefits during training while still maintaining model accuracy.
  + Post-training, you can then convert and optimize the model using TensorRT in FP16 mode for faster inference.
  + If aiming for even faster inference and smaller model sizes and if you're willing to potentially trade-off some accuracy, you can consider further quantizing the model to INT8 using TensorRT's calibration mechanism.

Always remember to test the accuracy of your models after any precision changes to ensure they still meet your application's requirements.