Abhay Chaturvedi Follow
Oct 11, 2020 · 5 min read · Listen

Understanding Nvidia TensorRT for deep learning model optimization

Posted by: Abhay Chaturvedi

What is TensorRT?

TensorRT is a library developed by NVIDIA for faster inference on NVIDIA graphics processing units (GPUs). TensorRT is built on CUDA, NVIDIA's parallel programming model. It can give around 4 to 5 times faster inference on many real-time services and embedded applications. While as per the documentation, It does give 40 times faster inference as compared to CPU only performance.

In this article, we will look into two things. First, how does TRT performs optimization on deep learning models? Secondly, how to use TensorFlow-TensorRT module for optimizing deep learning models.

How does TensorRT perform optimization?

TensorRT performs five types of optimization for increasing throughput of deep learning models. We will be discussing all five types of optimizations in this article.

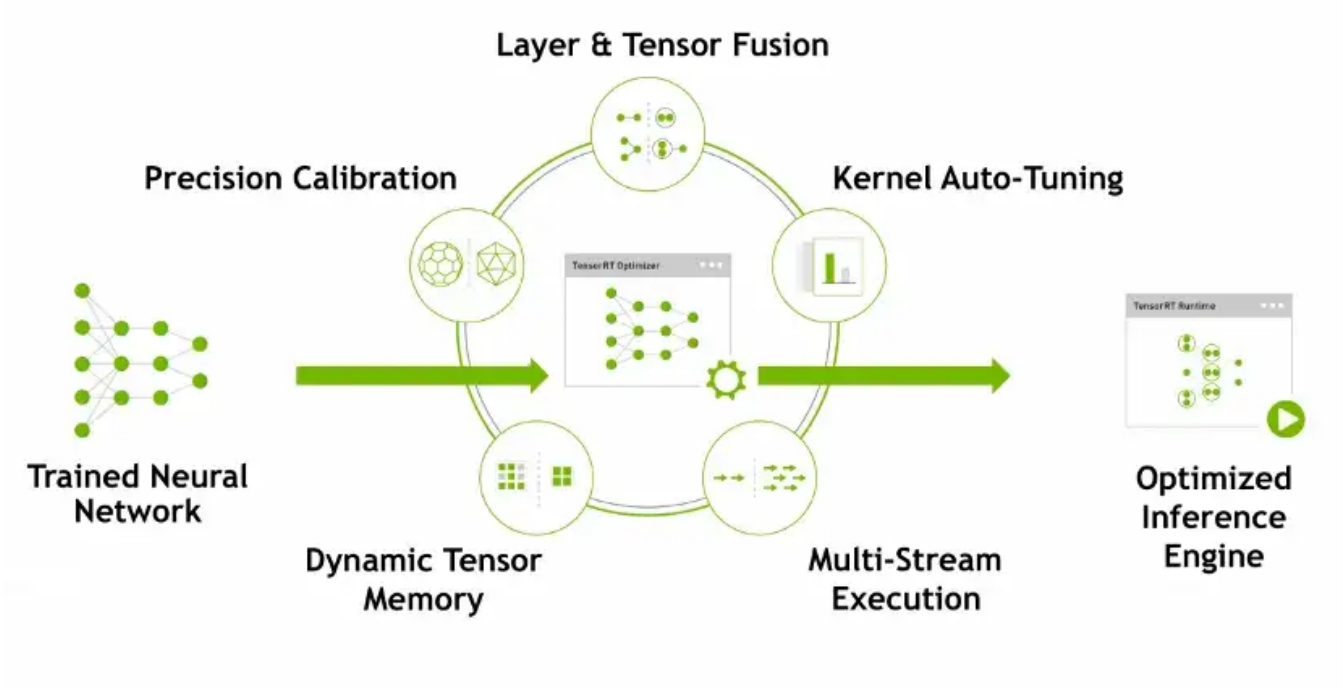

Figure 1. Types of optimizations performed by TensorRT to optimize models.

1) Weight and Activation Precision Calibration

During the process of training, parameters and activations are in FP32(Floating Point 32) precision. So to convert them in FP16 or INT8 precision. This optimization not only reduces latency but also gives a significant reduction in model size because FP32 precision get converted into FP16 or and INT8.

	Dynamic Range	Minimum Positive Value
FP32	-3.4×1038 ~ +3.4×1038	1.4 × 10−45
FP16	65504 ~ +65504	5.96 x 10-8
INT8	-128 ~ +127	1

Figure 2. Dynamic range of different precisions.

Now, while converting to FP16 (lower precision) some of our weights will get shrunk due to overflow as the dynamic range of FP16 is lower than that of FP32. But experimentally, this doesn't affect the accuracy significantly.

But, how do we justify that? We know FP32 is high precision. Weights and activation values, in general, are resilient to noise. While training, the model tries to preserve the features necessary for inference. So, throwing out unnecessary stuff is a kind of built-in process. So while we convert the model in lower precision we assume that noise is thrown out by the model.

This similar technique of clipping overflowing weights won't work while converting to INT8 precision. Because INT8 values are very small ranging from [-127 to +127] and most of our weights will get modified and overflow in lower precision resulting in a significant drop in accuracy of our model. So what we do is, we map those weights in INT8 precision using scaling and bias term.

How to find a scaling factor and bias term?

The bias term doesn't add value so we will focus on the scaling factor. To get the scaling factor we directly apply map from -|max| and |max| in FP32 value to INT8 -127 and 127 values respectively. The other values get assigned accordingly as per the linear scale. As shown in the left image of figure 3.

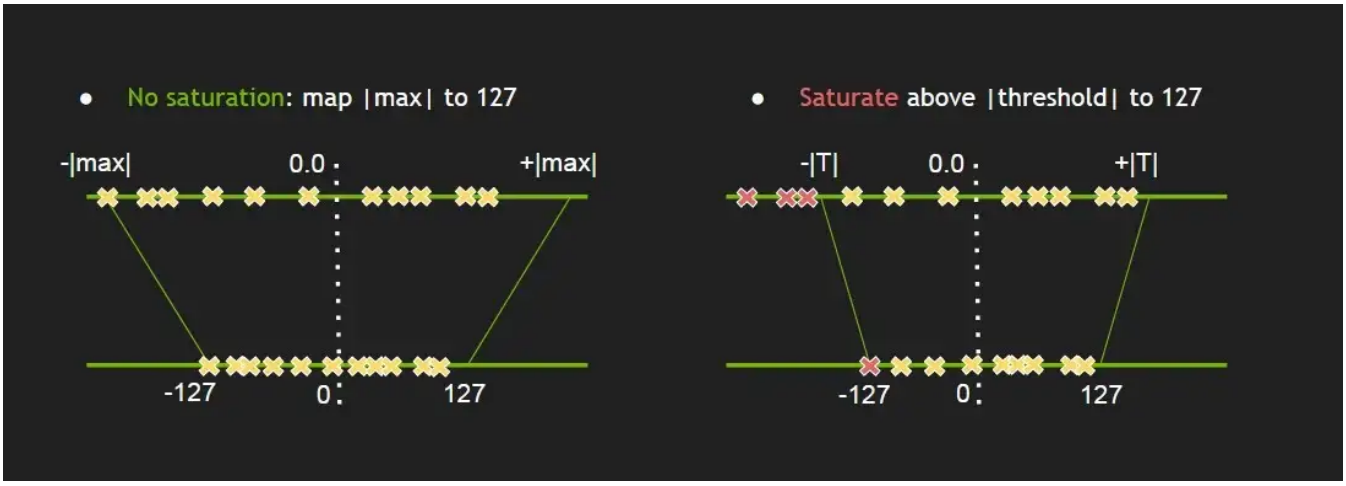

Figure 3. INT8 Calibration.

This thing can also result in a significant drop in accuracy. Now, instead of mapping from |max| values. What we do is, we map the threshold value range to -127 or 127 and all other outliers to the threshold get mapped to extreme range value i.e. -127 or 127. As shown in the right image of figure 3.

How to get the optimal value of the threshold? So, to represent FP32 distribution in INT8 TensorRT uses KL-divergence to measures the difference and minimizes it. TensorRT uses iterative search instead of gradient descent based optimization for

finding threshold. Pseudo-code steps for KL-divergence is given below. While you can read it here in detail.

```
1) Run FP32 inference on the calibration dataset.
2) For each layer:
    ~ collect histogram of activations.
    ~ generate many quantized distributions with different saturation thresholds.
    ~pick threshold that minimizes KL_divergence.
```

KL-divergence hosted with ❤ by **GitHub** **view raw**

Calibration data:

Is a set of a representative dataset, which gives histogram activation values used to find the threshold for minimum kl-divergence. This process is called calibration and the dataset is called the calibration dataset.

2) Layers and Tensor Fusion

While executing a graph by any deep learning framework a similar computation needs to be performed routinely. So, to overcome this TensorRT uses layer and tensor fusion to optimize the GPU memory and bandwidth by fusing nodes in a kernel vertically or horizontally(or both), which reduces the overhead and the cost of reading and writing the tensor data for each layer. This is a simple analogy of instead buying three items from the market in three trips we do a single trip and buy all three items.

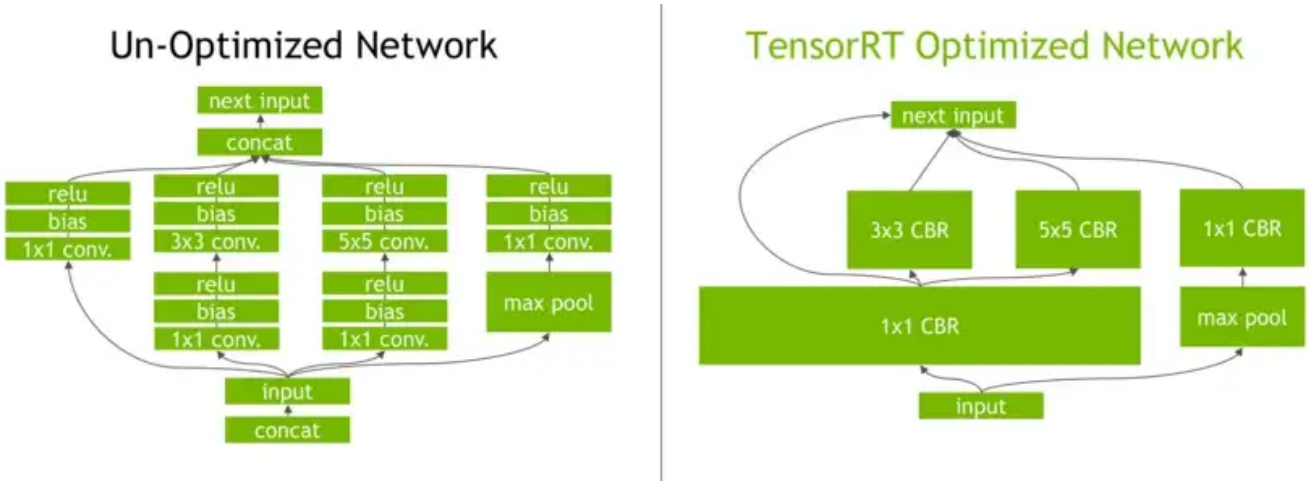

Figure 4. TensorRT's vertical and horizontal layer fusion on the GoogLeNet Inception module graph, reducing computation and memory overhead.

As shown above, TensorRT recognizes all layers with similar input and filter size but with different weights and combines them to form a single 1x1 CBR layer as shown

in the right side of figure 4.

3) Kernel auto-tuning

While optimizing models, there is some kernel specific optimization which can be performed during the process. This selects the best layers, algorithms, and optimal batch size based on the target GPU platform. For example, there are multiple ways of performing convolution operation but which one is the most optimal way on this selected platform, TRT opts that automatically.

4) Dynamic Tensor Memory

TensorRT improves the memory reuse by allocating memory to tensor only for the duration of its usage. It helps in reducing the memory footprints and avoiding allocation overhead for fast and efficient execution.

5) Multiple Stream Execution

TensorRT is designed to process multiple input streams in parallel. This is basically Nvidia's CUDA stream.

Using Tensorflow-TensorRT (TF-TRT) API

Now we have seen the process, how TRT optimizes the model for faster inference and lower latency. Next, we will look into Tensorflow-TRT API for optimizing our deep learning models for faster inference.

Setting up environment for TensorRT:

To setup TensorRT on your system execute the below commands in terminal to setup TensorRT on 64 bit Ubuntu 18.04 and TensorFlow≥2.0

We need to install NVIDIA machine learning package, libnvinfer5 and TensorFlow-GPU only if we want to use NVIDIA GPU while optimization.

```
pip install tensorflow-gpu==2.0.0

wget https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/x86

dpkg -i nvidia-machine-learning-repo-*.deb
apt-get update

sudo apt-get install libnvinfer5
```

Installation libraries for tensorrt.py hosted with ❤ by **GitHub** **view raw**

If you are not sure if Tensor-core GPU is present on your system. You can check it using following command.

```
from tensorflow.python.client import device_lib

def check_tensor_core_gpu_present():
    local_device_protos = device_lib.list_local_devices()
    for line in local_device_protos:
        if "compute capability" in str(line):
            compute_capability = float(line.physical_device_desc.split("compute capabi
            if compute_capability>=7.0:
                return True

print("Tensor Core GPU Present:", check_tensor_core_gpu_present())
tensor_core_gpu = check_tensor_core_gpu_present()
```

Is tensor-core gpu is present.py hosted with ❤ by **GitHub** **view raw**

First of all we import trt_converter and a per-trained resnet50 model, on which the optimization needs to be performed. If we want our model to get optimized in FP16 or Fp32 precision then we can just change precision_mode parameter in conversion parameters. Also, we can setup max_workspace_size_bytes parameter which indicates your maximum RAM capacity on your system.

But, while converting precision to INT8 we also need calibration data as described previously in this post and we can pass this calibration input data while calling the convert function. The code snippet is shown below.

The above implementation is done for TensorFlow version≥2.0. While if you want to perform optimization on TensorFlow1.x, you can use TrtGraphConverterV1 instead of TrtGraphConverterV2.

Concluding remarks

All the content of this article, including images are taken from the official documentation and website of NVIDIA TensorRT.

LinkedIn: https://www.linkedin.com/in/abhay-chaturvedi-39196b166/

GitHub: https://github.com/chaturvediabhay24

Tensorrt Optimization Deep Learning Models Model Optimization Nvidia