Developing Custom Plugins for Unsupported Layers in TensorRT

Kuldeep Singh, Ramanan, Kiran M. Sabu,Nagavardhan Reddy, Mr.Tejalal Choudhary (Mentor)

Abstract:

A deep learning model is optimized to its maximum extent via specifying the optimizers and loss functions. The Purpose of TensorRT is to optimize the deep learning model without compromising the performance of the model. This whole process is carried out before the deployment phase by converting a Network built using the common frameworks like Tensorflow, pytorch etc to a TensorRT network. Conversion to TensorRT network involves converting the layers/operation native to the framework to layers/operations of a TensorRT network. Currently TensorRT does not support all layers , we propose to create a custom plugin for one such unsupported layer thus optimizing the TensorRT model to the maximum.

Index terms: Nvidia SDK, Deep learning , TensorRT inference

1.Introduction:

TensorRT is a Software Development Kit(SDK) developed by NVIDIA for inference of high-performance deep learning models.TensorRT is available as an API for both C++ and Pyhton.It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications.TensorRT can be used to improve the performance of a wide variety of applications such as video streaming, speech recognition, recommendation and natural language processing.The basic idea behind tensorRT is to optimize the pretrained model before deployment so that it is efficient even in the simplest of devices such as Jetson nano [1].TensorRT optimizes the neural network by combining layers and optimizing kernel selection for improved latency, throughput, power efficiency and memory consumption. TensorRT has been integrated with Tensorflow making tensorflow the ideal framework for TensorRT optimization and inference.while troubleshooting.TensorRT optimizes trained neural network models to produce a deployment-ready runtime inference engine. The pre-trained neural network is optimized using several techniques like mixed-precision, layer fusion, kernel autotuning and Dynamic Tensor memory.[1]

· TensorRT has its advantages and disadvantages. Some of the advantages are, it allows deployment of a deep learning model in a device with low computing power such as Jetson nano thus improves the scalability. The memory usage and computation time during Inference are significantly reduced, which is of at most importance in applications related to Autonomous vehicles [1]. The GraphSurgeon

feature provides the potential to map TensorFlow nodes to custom layers in TensorRT, thus enabling Inference for many TensorFlow networks with TensorRT..

The Limitations of TensorRT are as follows, TensorRT does not support all layers of specific architectures. If it encounters an unsupported layer, its functionality is obtained from the corresponding framework used. To optimize new layers, a custom plugin is created to alter its operation in terms of tensorRT. This process is called Registering of custom plugins.

·The creation of custom plugins can be done in C++ or Python. The resources for creating custom plugins using Python is limited [1].

The rest of the paper is organized as follows: Section 2 describes the deployment of a TensorRT model. Section 3 includes the implementation and experimentation. The architecture of SPP is described in Section 5. The supported and unsupported layers are mentioned in the Section 6. Finally , the section 7

2. TENSORRT DEPLOYMENT :

TensorRT allows developers to import, calibrate, generate, and deploy optimized networks. Deep learning Networks can be imported directly from Caffe, or from other frameworks via the UFF or ONNX formats. Users can also run custom layers through TensorRT using the Plugin interface. The GraphSurgeon feature provides the capability to map tensorflow nodes to the corresponding custom layers in tensorRT, thus linking many TensorFlow networks with the tensorRT network. TensorRT provides an implementation of C++ on all supported platforms, and implementation of Python on x86, aarch64, and ppc64le. The key interfaces in the TensorRT core library are as follows

2.1 Network Definition:

The Network Definition interface provides methods to specify the definition of a network. Input Tensors and output tensors can be specified, layers can be added or modified , and there is an interface for designing each upheld layer type. As well as layer types, such as convolutional and fully connected layers, and a Plugin layer type allows the application to implement functionality not natively supported by TensorRT.TensorRT provides parsers like CAFFE, UFF and ONNX for importing trained networks to create network definitions.We used UFF parser to parse our network.

2.2 Builder:

The Builder interface allows the creation of an optimized engine from a network definition. It allows the application to specify the maximum batch size and workspace size, the lowest acceptable level of precision, timing iteration counts for autotuning, an interface for quantizing systems to run in 8-piece exactnessIt is possible to build multiple engines based on the same network definition, but with different builder configurations

2.3 Engine:

The Engine interface allows the application to execute Inference. It supports synchronous and asynchronous execution, profiling, and enumeration and querying of the bindings for the engine inputs and outputs. A single-engine can have multiple execution contexts, allowing a single set of trained parameters to be used for the simultaneous execution of multiple batches.

2.4 Serializing and deserializing:

We can either serialize the engine or we can use the engine directly for Inference. Serializing and deserializing a model is an optional step before using it for Inference - if desirable, the engine object can be used for Inference directly.

During Serialization,the engine is transformed into a format to store and use at a later time for Inference.We the apply deserialization on the engine to use it for Inference. Serializing and deserializing are optional.

2.5 Performing Inference:

This engine is fed with data to perform Inference.The inference part can be done from devices like Jetson nano with at most efficiency.

3.Implementation and Experiments:

In order to get a better understanding of the Mixed Precision technique a tensorflow model for detecting the handwritten numbers (MNIST dataset) was trained on the three different precisions individually. The tensorflow model(TF 32bit) model was built first and then converted to a TensorRT model with 32 bit precision. The tensorflow model was then converted to an INT 16 precision and INT 8 precision and was trained using the same dataset.Due to the Reduction in the precision ,the models used less memory and higher computation speed. When the precision is decreased the Throughput of the model increases. The throughput of the INT 32 model was 277 images/s and the throughput of the INT 16 precision model was 741 images/s.

The custom plugins present in the Github repository off NVIDIA was also executed and its impact on the model was observed. The custom plugins are written in C++ and executed in a linux environment. The sample codes of Custom plugins were studied and then the skeleton structure for the custom plugin was built.

The functionality of the Spatial Pyramid Layer was written in C++ and the plugin was registered under the TensorRT plugin library.A registered Plugin can be used when it is needed. The only limitation during the registration process is that the plugin should have a unique name.

5. Spatial Pyramid Pooling:

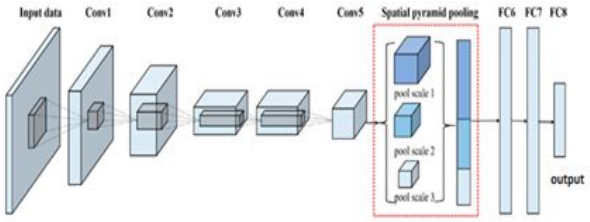

Fig 5.1: architecture of SPP-Net

Existing deep convolutional neural networks (CNNs) require a fixed-size (e.g. 64X64) input image. This requirement may decrease the recognition accuracy for the images or sub-images of an arbitrary size/scale.

Using Spatial Pyramid Pooling is more significant in Convolutional Neural Network. Using SPP layer , we compute the feature maps from the entire image only once, and then pool features in arbitrary regions (sub-images) to generate fixed-length representations for training the detectors.Spatial Pyramid Pooling allows the model to do manipulation on images of any size thus eliminating the necessity of maintaining a fixed input image size.

The layer is used between convolution and and fully connected layers so that we can generate output of fixed size from the output of convolution layers for the convolution layers.

The aim of this project is to create a custom plugin to map the functionality of the Spatial Pyramid Pooling to the tensorRT network thus reducing the time taken to perform the optimization process.

6.Supported and Unsupported layers:

Supported Layers in TensorRT	
Layer Name	**Layer Name in TensorRT**
Activation Layer	IActivationLayer
Concatenation Layer	IConcatenationLayer
Convolution layer	IConvolutionLayer
Arithmetic layer	IElementWiseLayer
Fill Layer	IFillLayer
Fully connected Layer	IFullyConnectedLayer
Padding Layer	IPaddingLayer

Unsupported Layers in TensorRT
Spatial Pyramid Pooling Layer

7. Conclusion:

TensorRT is a relatively new Software Development Kit thus, addition of support to new layers allows optimization of wide variety of architectures using the SPP layer. TensorRT is optimizing the pre-trained models of different frameworks.It's giving less inference timeAfter optimization model is generating more throughput. The .CPP and .h files are ready for SPP plugin

References:

[1] NVIDIA TensorRT Developer Guide
Available: https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html.
Access date: 10-12-2019.
[2] Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition [Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun Microsoft Research, China ,Xi'an Jiaotong University, China, University of Science and Technology of China] Access date: 20-12-2019