ISSN: 2664-9640

Research on the Acceleration Effect of Tensorrt in Deep Learning

Zhen Song[1], Ke Shui[1]

[1]Southwest Petroleum University, China.

Abstract

Deep learning is widely used in a variety of programs, such as natural language processing, recommendation systems, image and video analysis. With the development of deep learning technology, the requirements for precision and performance are getting higher and higher, which leads to the multiplication of model complexity and size, which greatly increases the time required for reasoning. In order to solve this problem, this paper explores the process of neural network optimization by TensorRT and the acceleration effect of several common neural network models.

Keywords

Deep learning, TensorRT, neural network.

1. Introduction

At present, many enterprises deploy neural networks in cloud servers, receive concurrent requests from users, calculate the inference results of each request, and respond to users. In addition to the internet speed delay, the main time-consuming process is in the inference phase of the neural network. In some areas, such as autonomous driving, target tracking, and high latency requirements, it is necessary to increase the speed of inference. NVIDIA TensorRT is the platform for high-performance deep learning reasoning. Mainly to optimize and run the deep learning reasoning part, which can greatly reduce the delay of deep learning program and improve throughput, and maximize the efficiency of GPU.

2. Acceleration Principle

Generally, the model used in inference is consistent with that used in training, but there is no need for this. Compared with the training process, only a forward calculation is needed in inference, and the prediction results are obtained through neural network. At this time, the parameters of the model have been fixed, the bandwidth requirement is smaller than that in training, and the removal of redundant structure has no great impact on the inference results. so the reasoning process has a great room for optimization.

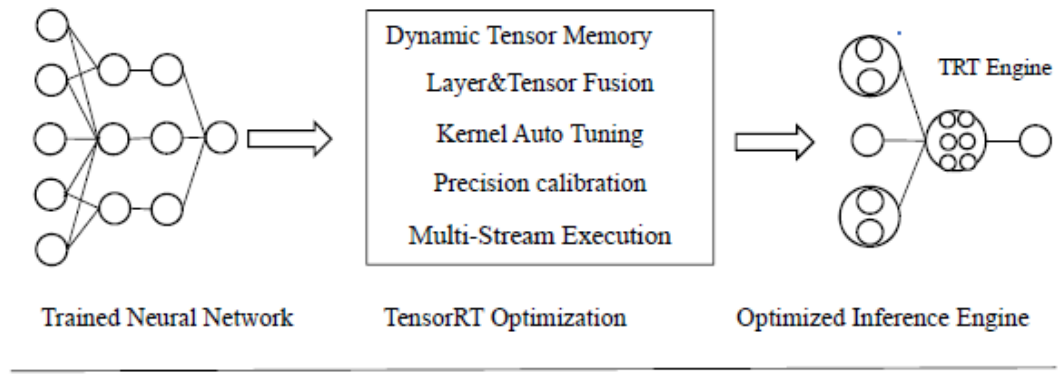

FIg 1. Optimization diagram

ISSN: 2664-9640

2.1. Merge Network Layer

TensorRT can decompose and reconstruct neural network networks1, merge and optimize compatible layers, and implement model compression. Figure 1 below shows how to optimize:

First, eliminate unused layers and avoid unnecessary calculations.

Vertical layer fusion on neural networks: Combine compatible layers into a single layer. For example, the convolution layer, the ReLU layer and the bias layer can be combined into one CBR layer, as shown in Figure 2(b).

Neural network horizontal layer fusion: merge similar layers of the same input. As shown in Figure 2 (c), three 1x1 CBR layers and two 3x3 CBR layers are fused respectively.

Finally, a highly compressed neural network model is obtained.

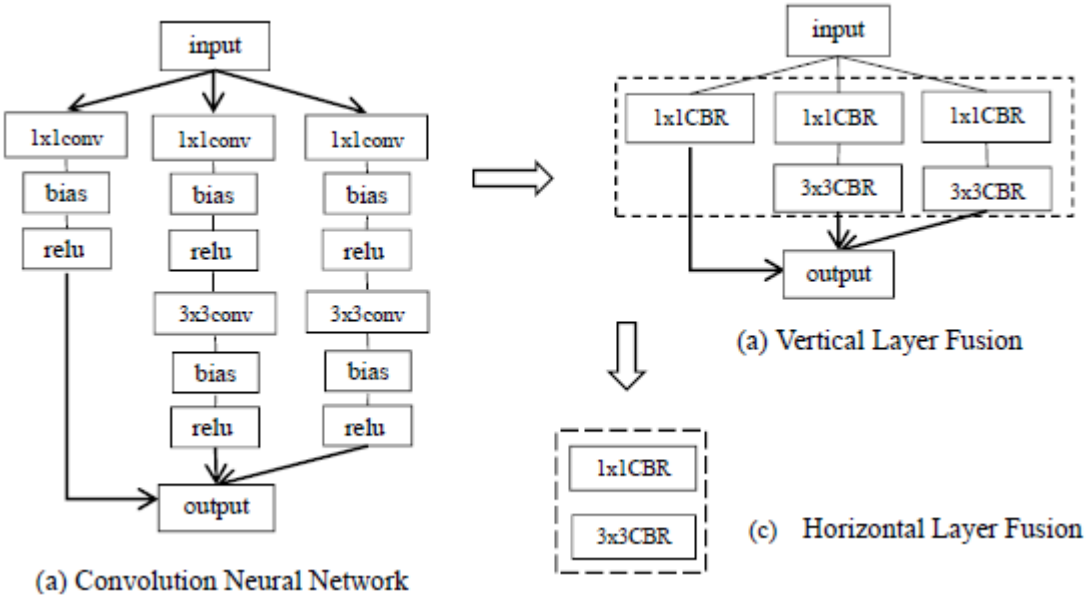

Fig 2. TRT Layer fusion diagram

2.2. Low Precision Inference

Usually the training model we get is FP32 precision (single-precision), but after engineering verification, the neural network can still get the correct result even if the calculation precision is reduced during the reasoning process. TensorRT supports inferences using low-computation precision, such as FP16 (half-precision), INT8.

In theory, if a NVIDIA GPU Tensor Core can perform one FP32 calculation, two FP16 calculations can be performed with half precision, and the throughput can be doubled. With INT8 precision, four INT8 calculations can be performed, which greatly reduces the use of computing resources. Combined with the combined model, the inference speed of neural network model can be doubled.

3. Environment Building and GPU Arithmetic Support

In this paper, the acceleration process and the acceleration effect of three common neural network models were studied by using the TenorRT, and the results of the conventional deployment and the TensorRT deployment were compared. The neural network model used was ResNet50, YOLOV3, and CTPN.

ISSN: 2664-9640

Table 1

Request	TensorRT	TensorFlow	PyTorch	CUDA	Cudnn
Version	6.0.1.5	1.13	1.2	10.0	7.6.3

Using low-precision inference requires support for GPU performance, and some older graphics cards may not be able to make low-precision inferences. The following are the calculation accuracy supported by different computing power graphics cards. For common common GPUs, the GTX10 series GPU has a computing power of 6.1, which cannot be used for FP16 inference. The RTX20 series GPU has a computing power of 7.5, which supports full-precision inference. So it is recommended to use the latest graphics card or professional computing GPU

Table 2

GPU compute capability	Example	FP32	FP16	INT8
7.5	Tesla T4	Yes	Yes	Yes
7.2	Jetson AG	Yes	Yes	Yes
7.0	Tesla V100	Yes	Yes	Yes
6.2	Jetson TX2	Yes	Yes	No
6.1	Tesla P4	Yes	No	Yes
6.0	Tesla P100	Yes	Yes	No
5.3	Jetson TX1	Yes	Yes	No
5.2	Tesla M4	Yes	No	No

4. Experimental Process and Results

4.1. TensorRT + ResNet50

ResNet503, also known as the residual network, is the backbone of many complex neural network models and is proven to be fully compatible with TensorRT. The network model is mainly composed of four groups of blocks, each of which is 3, 4, 6, and 3 blocks, each block contains 5 convolutional layers, and the whole model has a total of 49 convolutional layers and a fully connected layer. The original framework used is PyTorch, When TensorRT optimization is performed, it is first exported to onnx format, and then read and executed by parser. After TensorRT compression, as shown in Figure 3, all OPs are directly compressed into a TRT engine (a TRT engine with multiple steps)

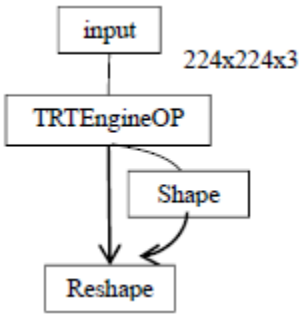

Fig 3. Optimized ResNet model 505050model

ISSN: 2664-9640

The selected test set has a total of 10,000 images, and the speed and accuracy of the ResNet50 optimized by TensorRT under different precisions are tested. The results are taken as the average of 10 tests. The data is shown in Table 3.

Table 3

	PyTorch	TRT FP32	TRT FP16	TRT INT8
total time (min)	20.74	11.08	5.72	4.27
Average time (s)	0.1245	0.0664	0.0343	0.0256
Precision	0.8142	0.8136	0.8102	0.7864

It can be seen that the ResNet50 model is optimized (FP32), the speed is nearly doubled. With half precision (FP16), the speed is nearly doubled on the basis of FP32. But the speed of INT8 is only 25% higher than that of half-precision. After the final optimization, the speed is 5 times that of the original model.

4.2. TensorRT + YOLOV3

YOLOV34 is a commonly used network architecture in target detection, consisting of darknet-53 and YOLO parts. The YOLO part consists of a DBL component (convolution + BN + Leaky relu) and a fully connected layer. For darknet-53, TensorRT can merge it into a TRT engine (see TRTEngineOP0 in Figure 4), but for DBL structure, TensorRT does not support Leaky ReLU, only a part of it can be merged (see TRTEngineOP1, 2 in Figure 4). Model structure shown in Figure 4.

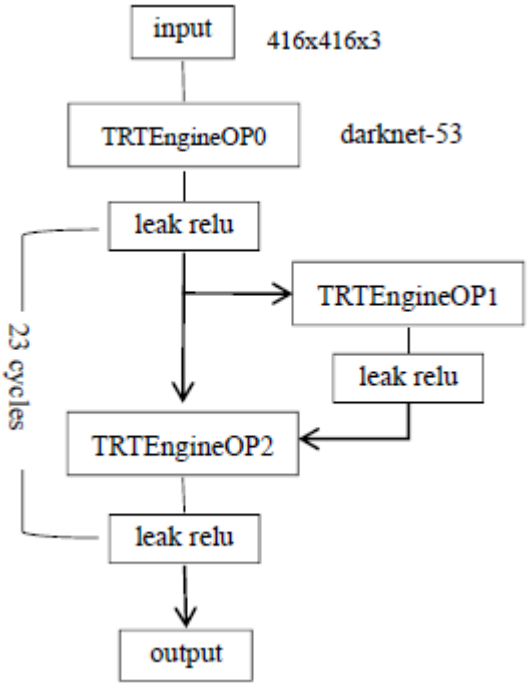

Fig 4. Optimized YOLOv3 model

For the deployment of YOLOV3, the COCO dataset and VOC dataset were used to study the acceleration process and acceleration effect.

ISSN: 2664-9640

The experiment found that in the process of reducing FP32 to INT8, part of the image detection effect will be better than the original model, and some pictures will have detection errors. Studies have shown that the loss of precision leads to a change in the degree of model fitting, which improves the over-fitting phenomenon to some extent, but may also result in under-fitting.

In the experiment, the speed of YOLOV3 was improved by about 50%, and the theoretical speed was not reached. There are two main reasons: 1. TensorRT has poor compatibility with YOLOV3, and there are fewer structures that can be optimized. 2. TensorFlow's integrated TensorFlow does not achieve optimal efficiency.

Fig 5. TensorFlow inference

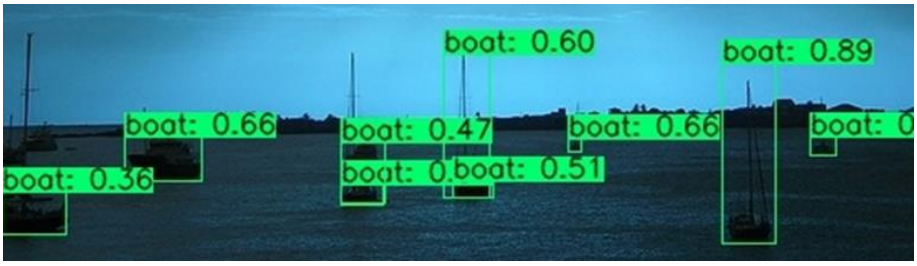

Fig 6. TensorRT INT8 inference results

4.3. TensorRT + CTPN

CTPN5 is a neural network model for natural scene text box detection, which consists of a convolution part, a BLSTM part and an FC feature layer. In TensorRT, we mainly optimize the convolution part. As shown in the figure, the convolution part is a fine-tuned VGG16 model. Due to TensorRT's good optimization of convolution, the entire convolution part can be merged into one TRT engine. The rest is not compatible with TensorRT, and is not optimized. The model is shown in Figure 7.

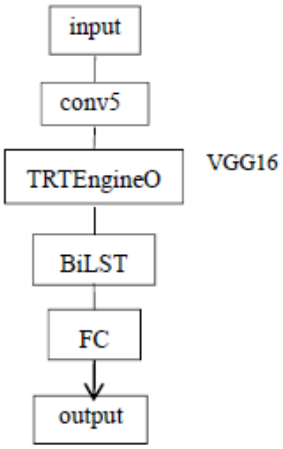

Fig 7. Optimized CTPN model

ISSN: 2664-9640

The CTPN is deployed in a production environment, the GPU memory footprint is maximized, and the maximum throughput is tested using multiple processes. Comparing the optimized TensorRT FP16 model with the original TensorFlow model, the results are as follows:

Table 4

Number of processes	model	GPU usage	Average one picture time (s)	One minute Throughput	Optimized speed / original speed
1	TRT FP16	24%	0.1120	534	166.6%
	TensorFlow	52%	0.1863	321	
2	TRT FP16	49%	0.1490	826	189.4%
	TensorFlow	72%	0.2748	436	
3	TRT FP16	68%	0.1739	1034	207.5%
	TensorFlow	99%	0.3607	498	
5	TRT FP16	99%	0.2277	1316	234.6%
	TensorFlow	99%	0.5339	561	

The RTX2080ti has 11GB of memory and can only boot up to 5 CTPN processes. According to the data in Table 4, the more processes, the more obvious the advantages of TensorRT. When the number of processes reaches the maximum, the inferred speed of TensorRT FP16 is 234.6% of the original model speed, and the acceleration effect is obvious.

5. Conclusion

The use of TensorRT to accelerate the inference process of neural networks. The acceleration effects of different network models are different, even huge. The specific acceleration effect is related to the layer that can be optimized. The greater the proportion of structures that can be optimized, the more pronounced the increase in speed.

Using TensorRT can reduce the amount of GPU computing resources used and can run more processes under the same resources. Therefore, the maximum use of the GPU is of great significance to production deployment.

After optimization by TensorRT, the combined effect of using FP16 is the best, and the speed is significantly improved compared with the original model, but the loss of precision is negligible. Due to the different calibration quality, INT8 may have various problems. FP16 is recommended.

References

[1] Production Deep Learning with NVIDIA GPU Inference Engine. https: // devblogs. nvidia.com/ production-deep-learning-nvidia-gpu-inference-engine/

[2] CUDA C Programming Guide. https: // docs. nvidia. com/ cuda/ cuda-c-programming-guide/ index.html#arithmetic-instructions/

[3] K. He, X. Zhang, S. Ren and J. Sun. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778.

[4] Redmon J , Farhadi A . YOLOv3: An Incremental Improvement[J]. 2018.

[5] Tian Z , Huang W , He T , et al. Detecting Text in Natural Image with Connectionist Text Proposal Network[J]. 2016.