Inference Engines Effect on Neural Networks

Brady Gehrman, Dylan Nicholson, Christopher Moore

Department of Compter Science and Engineering
University of Louisville
Louisville, USA
{btgehr01@louisville.edu}
{dlnich07@louisville.edu}
{ckmoor04@louisville.edu}

***Abstract* — Deep neural networks (DNNs) have emerged as powerful tools in various domains, including computer vision and natural language processing. This paper presents an exploration of the TensorRT optimization engine's impact on DNN models implemented in popular frameworks, PyTorch and TensorFlow. Building upon the foundational work of Demystifying TensorRT, the research investigates the optimization effects across different machine learning software packages and hardware configurations. Evaluation metrics such as accuracy and inference latency are employed to compare optimized and unoptimized models on different GPU testing devices.**

Results indicate that TensorRT optimization consistently improves inference latency, providing a notable speedup in model execution. While accuracies remain relatively constant. This research contributes valuable insights into the practical implications of TensorRT optimization for DNNs, emphasizing its potential to enhance performance across diverse machine learning applications and hardware setups.

I. Introduction

Deep neural networks have become the most popular deep learning model in recent times. They have been widely used in applications that deal with computer vision, natural language processing, and speech recognition. There have been several deep learning libraries created to ease the creation of several different types of machine learning models including the deep neural network; the two most popular frameworks being PyTorch and TensorFlow. These deep learning models are commonly trained in the cloud and then deployed on specialized pieces of hardware such as the GPU. More recently there has been a push to find different optimization techniques that allow for more efficient model storage and deployment on these devices, this is especially useful for models used in real-time environments stored on computational edge devices. One such technology from Nvidia, coined TensorRT, was created to optimize the deployment of deep neural networks on Nvidia GPUs. Nvidia's TensorRT optimization engine looks to decrease inference times for GPU-stored deep neural networks while keeping the accuracy of the models constant.

Brady Gehrman was responsible for creating the necessary code to apply Nvidia's TensorRT optimization engine to the generated PyTorch and TensorFlow deep neural network models. This process involved understanding which libraries and or packages needed to be downloaded or installed and creating a plan to implement those packages as well as creating the code needed to test the optimized models.

Christopher Moore handled the TensorFlow implementation of the Neural Network as well as the TensorRT optimization of the that model. The TensorFlow model was based on a collection of TensorFlow and Nvidia examples. Christopher was also responsible for collecting data from the Lab computer stationed in Duthie Cryptography lab.

Dylan Nicholson took charge of the PyTorch implementation of the neural network as well as applying the TensorRT optimization engine to the created models. This process involved training the created neural network as well as creating the testing function that was used to generate the time improvements and accuracy calculations of the PyTorch model.

II. Related Work

Demystifying TensorRT [1] is the foundation on which this project is built off. [1] tests TensorRT optimizations for NN models on older Nvidia NX and AGX platforms. TensorRT is an inference library developed by NVIDIA to optimize and accelerate deep neural networks on their devices. One of the major features of the software tool is the ability to fuse layers of the neural network together to reduce computational overhead. This allows for devices to spend less time performing computations that are needed by each additional layer of the neural network. The engine is also able to dynamically reuse the various tensors created during inferencing, which can cut down on the memory footprint utilized on the various GPUs. The tensors are the pieces of data that are loaded onto the GPU, so that the model can be tested there. By being able to reuse the tensors being passed to GPU, less memory needs to be copied over to the device allowing for more time to be focused on other operations. The researchers found that classification models run through the inference engine saw increases of roughly 23x the number of frames that can be processed by the model in a second [1]. The decrease in execution time is extremely important to the overall functionality of the engine as it creates the baseline on how the models should perform on our testing systems. They cover many more NN models from more packages than tested here. They're findings are to be validated on newer hardware, so fewer models can be selected.

III. Proposed Research

To explore the full capabilities of TensorRT and its effects on Neural Networks, implementations across a variety of packages and systems will need to be tested and compared.

The first stage was selecting which machine learning software packages will be used to implement the NN's. As TensorFlow is designed for synergy with Nvidia devices and TensorRT itself, it proved to be an obvious choice and one of the packages to implement. The second package selected was PyTorch. PyTorch has garnered a positive reputation and has set itself as an industry favorite. Considering its popularity and extensive online community support, it was a clear second choice. Initial plans were made for use of a third package tested in [1], such as caffe or Darknet. Time constraints and relative difficult learning curve of the already selected packages limited the scope of the project.

The next step was to gather machines with compatible edge devices (Nvidia GPU's) and collect their hardware information. Table 1 shows which GPUs were used to test the models.

Table 1. GPU Test Devices

GPU Device	NVIDIA GeForce RTX 3080	NVIDIA GeForce RTX 3060	NVIDIA GeForce MX150
Memory Size	10 GB	12 GB	2 GB
Memory Type	GDDR6X	GDDR6	GDDR5
Memory Clock	19.01 GHz	15 GHz	1502 MHz
Memory Bandwidth	760.32 GB/s	360 GB/s	6 GB/s
Cuda Cores	8704	3584	384
Base Clock	1.440 GHz	1.32 GHz	1469 MHz
Boost Clock	1.710 GHz	1.78 GHz	1532 MHz
Architecture	Ampere	Ampere	Pascal

The machine leaning task that the networks need to conduct is arbitrary. What is being classified or what prediction is made is of little importance as only the improvement made when optimizing is to be considered. Original plans were made to use the same data set as in [1] which is a subset of the *Imagenet* dataset consisting of 1000 classes each with 50 images. This data set lead to the use of the *DeepLake* Python package for data loading as it had the set in their cloud servers for easy distribution. [1] created an object detection network for this data which proved to be a difficult task. The complexity of the task as well as the variety of the data was cause for a pivot to a new data set.

The Fashion-MNIST [2] became the new data set for the NN classification. The Fashion-MNIST set contains 60,000 training samples and 10,000 test samples of 10 different classes. Each sample is a 28 x 28 pixel grayscale image of different types of apparel. [2] is a popular data set for benchmarking algorithms and models and will be an appropriate set for this project. The goal for the NN is to process the 784 pixels of the image into one output classification.

Using the selected deep learning toolkits and dataset several neural networks could be trained and then optimized using TensorRT. The unoptimized deep neural networks and optimized deep neural networks could then be compared by collecting the total amount of time it takes each model to label all the test images while also comparing the accuracy of each model.

IV. Methodology

Regardless of Software or Hardware, the experiment will follow these steps:

1. Load the data from *DeepLake* onto the edge device.
2. Flatten the 28 X 28 images into a 1D vector and normalize the values.
3. Create and Train the NN
4. Save NN model to a file.
5. Compile model on edge device
6. Pass data through model
7. Output model accuracy and execution times
8. Pass the Saved model into TensorRT converter.
9. Saved Converted model to file.
10. Repeat Steps 5 – 7 for optimized model.

V. Evaluation

After the TensorRT optimized models were generated for the ResNet-50 deep neural networks created using PyTorch and TensorFlow, the optimized models and unoptimized models were then tested against each other with different testing metrics such as accuracy and inference latency. Each test script was executed ten times for each GPU testing device that the team had access to. The results of those ten trials were then averaged and then tabulated. The results can be seen in table 2 and table 3.

Table 2. Accuracy Comparisons

	NVIDIA GeForce RTX 3080		NVIDIA GeForce RTX 3060		NVIDIA GeForce MX150	
	Unoptimized	*TensorRT*	*Unoptimized*	*TensorRT*	*Unoptimized*	*TensorRT*
PT	81.74%	81.96%	81.3%	81.1%	81.45%	81.49%
TF	86.60%	86.60%	87.36%	87.36%	87.41%	87.41%

Table 3. Execution Time Comparisons

	NVIDIA GeForce RTX 3080		NVIDIA GeForce RTX 3060		NVIDIA GeForce MX150	
	Unoptimized	*TensorRT*	*Unoptimized*	*TensorRT*	*Unoptimized*	*TensorRT*
PT	5.742 s	4.602 s	5.322 s	5.291 s	18.547 s	10.013 s
TF	0.740 s	0.179 s	0.519 s	21.92 s *	1.706 s	0.879 s

Table 2 showcases the accuracies of the TensorRT optimized and unoptimized deep neural networks created using PyTorch and TensorFlow respectively. As documented, the accuracies among the unoptimized and optimized models were held relatively constant. However, it most cases you can even noticed that the TensorRT optimized models produced a higher testing accuracy, this is likely due to the unoptimized models commonly overfitting the data, which is countered when the model was optimized using the TensorRT optimization engine.

Table 3 illustrates the average inference latency for the TensorRT optimized model versus the unoptimized model built using PyTorch and TensorFlow respectively. These models were each given ten thousand images from our dataset for

testing. These images were batched to the GPU edge device one hundred at a time. The important thing to notice is that the cumulative time that it took these models to label each testing image from the testing set was lower for the TensorRT optimized models. However, the speedup was not uniform for each GPU testing device. We noticed that for newer GPUs such as the Nvidia GeForce RTX 3080 and the Nvidia GeForce RTX 3060 we only saw a fractional speedup from the optimized model; however, for older GPUs such as the Nvidia GeForce MX150 the speedup was almost two times for the optimized model. This is likely because our trained deep neural network was not very large in size compared to the ones deployed within industry. This means that the unoptimized neural networks likely still performed well on the newer GPUs that encompass more memory and Cuda threads, leaving little to be optimized when it comes down to the models themselves for those devices. However, the older GPU with lower memory bandwidth and less Cuda cores were impacted more when the model was optimized using TensorRT because of their lesser computational output. Another thing to note is that the models produced from the different deep learning libraries (PyTorch and TensorFlow) had different speedup results when they were optimized using TensorRT. The optimized TensorFlow model seemed to produce a higher speedup when compared to the optimized PyTorch model.

VI. Conclusions and Future Work

In conclusion, this paper has highlighted the importance of TensorRT in real world applications involving the use of neural networks. As demonstrated above, the optimizations applied by TensorRT improve the typical run time of the models tested on a variety of hardware devices. Every device that we tested showed improvements in the runtime for both TensorFlow and PyTorch models. The various improvements in runtime prove the effectiveness of TensorRT as an inference engine, but also demonstrate its ability to adapt to various neural network frameworks. This flexibility is extremely important in more critical time applications because any of the created models can be improved with the optimization methods of TensorRT. Potential future work could involve training a much larger neural network with more than 60000 images to see how well the optimization engine performs on much larger devices. In addition, further testing could be done to evaluate the exact frames per second and top-1 error metrics of the created models. This could help verify the improvements of the optimization engine. Lastly, potential tests could be done utilizing real-time environments such as autonomous vehicle navigation and modern robotics tasks. In summary, this research contributes insights into the usefulness of TensorRT and its ability to advance the development of neural network models in critical time applications.

References

[1] O. Shafi, C. Rai, R. Sen and G. Ananthanarayanan, "Demystifying TensorRT: Characterizing Neural Network Inference Engine on Nvidia Edge Devices," 2021 IEEE International Symposium on Workload Characterization (IISWC), Storrs, CT, USA, 2021, pp. 226-237, doi: 10.1109/IISWC53511.2021.00030.

[2] Xiao, H., Rasul, K. & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms (cite arxiv:1708.07747Comment: Dataset is freely available at https://github.com/zalandoresearch/fashion-mnist Benchmark is available at http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/)