Fast Training of an AI Radiologist

Leveraging Data Pipelining to Efficiently Utilize GPUs

Rakshith Vasudev
rakshith_vasudev@dell.com
Dell EMC HPC & AI Innovation Lab
Austin, Texas, U.S.A.

John A. Lockman III
john_lockman@dell.com
Dell EMC HPC & AI Innovation Lab
Austin, Texas, U.S.A.

Lucas A. Wilson
luke_wilson@dell.com
Dell EMC HPC & AI Innovation Lab
Austin, Texas, U.S.A.

Srinivas Varadharajan
s_varadharajan@dell.com
Dell EMC HPC & AI Innovation Lab
Austin, Texas, U.S.A.

Frank Han
Frank_Han1@dell.com
Dell EMC HPC & AI Innovation Lab
Austin, Texas, U.S.A.

Rengan Xu
rengan_xu@dell.com
Dell EMC HPC & AI Innovation Lab
Austin, Texas, U.S.A.

Quy Ta
quy_ta@dell.com
Dell EMC HPC & AI Innovation Lab
Austin, Texas, U.S.A.

ABSTRACT

In a distributed deep learning training setting, using accelerators such as GPUs can be challenging to develop a high throughput model. If the accelerators are not utilized effectively, then this could mean more time to solution and thus the model's throughput is low. To use accelerators effectively across multiple nodes, we need to utilize an effective data pipelining mechanism that handles scaling gracefully so GPUs can be exploited of their parallelism. We study the effect of using the correct pipelining mechanism that is followed by tensorflow official models vs a naive pipelining mechanism that doesn't scale well, on two image classification models such as ResNet50 and DenseNet121 as pretrained base models for CheXNet. Both the models using the optimized data pipeline demonstrate effective linear scaling when GPUs are added. Further, we also show that converting raw images to TF Records is not always necessary if there is an extensive work needed to convert raw images to TF Records and if there is a permissible bandwidth to take approximately 8% - 15% throughput loss.

KEYWORDS

data pipeline, effective GPU utilization, DensNet121, ResNet50, GPUs, horovod, distributed deep learning

ACM Reference Format:
Rakshith Vasudev, John A. Lockman III, Lucas A. Wilson, Srinivas Varadharajan, Frank Han, Rengan Xu, and Quy Ta. 2019. Fast Training of an AI Radiologist: Leveraging Data Pipelining to Efficiently Utilize GPUs. In *Proceedings of Supercomputing 2019 (SC'19).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

SC'19, November 18–21 2019, Denver, CO, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As many organizations start to adopt AI in their business, the need to quickly train models continues to grow. The healthcare domain is one example where continuous improvement in models is necessary. GPUs offer the potential to train deep learning models more quickly, often times by orders of magnitude when compared to unaccelerated methods.

As an use case, we consider CheXNet [1] model from Stanford that is an AI Radiologist which takes in a Chest X-ray image and outputs the probability of pneumonia with upto 14 different thoracic diseases. We explored ways to develop high throughput neural network based models for identifying pneumonia, emphysema, and a host of other thoracic pathologies. Our goal was to develop high throughput model which could be trained in parallel that utilizes GPUs effectively on HPC clusters.

2 MODEL ARCHITECTURE

Stanford's CheXNet is the starting point, that uses Chest-Xray14 dataset from NIH. DenseNet121 [2] is used as the pretrained base model with ImageNet [3] weights, Adam optimizer is wrapped around by the horovod [4] optimizer to support distributed training, while the local mini batch size is 64.

3 HPC EXPERIMENT SETUP

Our experiments were performed on the Dell EMC PowerEdge C4140 dense compute platform [5]. Each node contains four Tesla V100 data center GPUs [6], two Intel Xeon Gold 6148 processors, 384 GB of 2666MHz DDR4 RAM inter connected with Mellanox EDR HCA. The storage is a Lustre file system.

The deep learning framework is Tensorflow-GPU: 1.12.0, Horovod: 0.16.4, MPI: 4.0.0 with CUDA and UCX support, CUDA version: 10.1.105, CUDNN: 7.6.0, NCCL Version: 2.4.7, Python: 3.6.8 and OS: RHEL-7.4.

4 IMPROVING THE DATA PIPELINE

Pipelining [7] overlaps the preprocessing and model execution of a training step. While the accelerator is performing training step N, the CPU is preparing the data for step N+1. Improving the input data pipeline involves aspects like sharding, enabling shuffling only during training, parallel interleaving which is preprocessing multiple files, prefetching batch size number of files such that they are available to be used for the next cycle by the GPUs, the dataset is then subjected to simultaneous map and batch upon which it is prefetched again. This new approach is effective to be able to scale to multiple GPUs on distributed nodes. Close to linear scaling in throughput can be expected as more GPUs are added with both, raw images and TF Records (Figure 3). We tried exploring to see if the same holds good for different base models such as DenseNet121 and Resnet50, which proved to be effective (Figures 1 and 2). The naive approach here used incorrect sequence of operations while building the TF data pipeline such as not prefetching twice and not using bulk of elements at the same time for preprocessing and not parallel interleaving. In an ideal setup, the GPUs should hover around 100% utilization when the correct optimized pipeline technique is used. We use the TensorFlow's Official models [8] approach to build the TF data pipeline to utilize accelerators effectively.

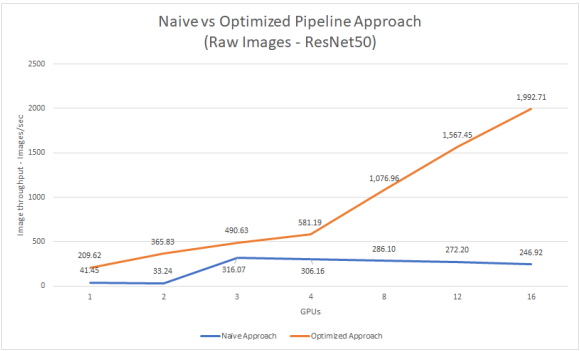

Figure 1: Naive vs Optimized Pipeline - ResNet50

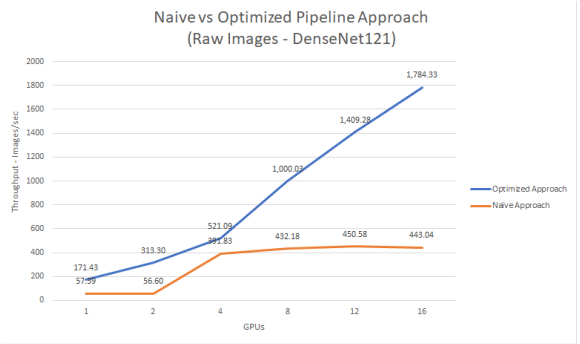

Figure 2: Naive vs Optimized Pipeline - DenseNet121

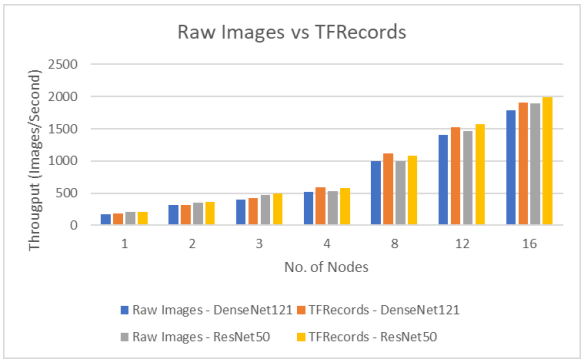

Figure 3: Raw Image vs TF Records

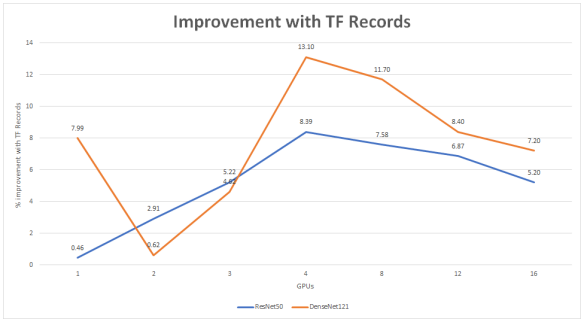

Figure 4: Improvement with TF Records

4.1 Raw Images VS TF Records

Having the right data pipeline enabled us to almost close the gap in performance with raw images and TF records (Figure 3). The most performance gain with ResNet50 model was seen at 4GPUs with 8.39% and that with DenseNet121 was seen at 4 GPUs with 13.1% (Figure 4). Another aspect to consider is the effort to create TFRecords, in particular, when there is a continuous change in the data or if the time taken to encode and decode these images is too high or even the number of times the model is retrained. Unless training is done for benchmarking or other similar purposes, conversion to TF Records are not always required if there's more effort required to achieve the same.

5 CONCLUSION

We presented an approach to scale the workload and parallelize in order to effectively engage accelerators such as GPUs to fully utilize them by building the correct optimized pipelining mechanism. We also indicated that using TF Records may not be necessary when there is a good pipeline that can scale well in place.

6 FUTURE WORK

In the future, we will extend our tests to see if there is a significant performance hit when using raw input data vs TF Records on different workloads such as text, video, audio, etc.

REFERENCES

[1] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Yi Ding, Aarti Bagul, Curtis Langlotz, Katie S. Shpanskaya, Matthew P. Lungren, and Andrew Y. Ng. 2017. Chexnet: radiologist-level pneumonia detection on chest x-rays with deep learning. *CoRR*, abs/1711.05225. arXiv: 1711.05225. http://arxiv.org/abs/1711.05225.

[2] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely connected convolutional networks. *CoRR*, abs/1608.06993. arXiv: 1608.06993. http://arxiv.org/abs/1608.06993.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

[4] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*.

[5] Dell EMC. 2019. Dell EMC PowerEdge C4140 Server. https://www.dell.com/en-us/work/shop/povw/poweredge-c4140. (2019).

[6] NVIDIA. 2019. Nvidia Tesla V100 Data Center GPU. https://www.nvidia.com/en-us/data-center/tesla-v100/. (2019).

[7] Tensorflow. 2019. Data Input Pipeline Performance. https://www.tensorflow.org/guide/performance/datasets. (2019).

[8] Tensorflow. 2019. Tensorflow official Models. https://github.com/tensorflow/models/tree/master/official/resnet/keras. (2019).