

Importance of data loading pipeline in training deep neural networks

M. Zolnouri,
X. Li, V. Partovi Nia

G-2020-23-EIW08

April 2020

La collection *Les Cahiers du GERAD* est constituée des travaux de recherche menés par nos membres. La plupart de ces documents de travail a été soumis à des revues avec comité de révision. Lorsqu'un document est accepté et publié, le pdf original est retiré si c'est nécessaire et un lien vers l'article publié est ajouté.

Citation suggérée : M. Zolnouri, X. Li, V. Partovi Nia (Avril 2020). Importance of data loading pipeline in training deep neural networks, *In* C. Audet, S. Le Digabel, A. Lodi, D. Orban and V. Partovi Nia, (Eds.). Proceedings of the Edge Intelligence Workshop 2020, Montréal, Canada, 2-3 Mars, 2020, pages 51-58. Les Cahiers du GERAD G-2020-23, GERAD, HEC Montréal, Canada.

Avant de citer ce rapport technique, veuillez visiter notre site Web (<https://www.gerad.ca/fr/papers/G-2020-23-EIW08>) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

The series *Les Cahiers du GERAD* consists of working papers carried out by our members. Most of these pre-prints have been submitted to peer-reviewed journals. When accepted and published, if necessary, the original pdf is removed and a link to the published article is added.

Suggested citation: M. Zolnouri, X. Li, V. Partovi Nia (April 2020). Importance of data loading pipeline in training deep neural networks, *In* C. Audet, S. Le Digabel, A. Lodi, D. Orban and V. Partovi Nia, (Eds.). Proceedings of the Edge Intelligence Workshop 2020, Montreal, Canada, March 2-3, 2020, pages 51-58. Les Cahiers du GERAD G-2020-23, GERAD, HEC Montréal, Canada.

Before citing this technical report, please visit our website (<https://www.gerad.ca/en/papers/G-2020-23-EIW08>) to update your reference data, if it has been published in a scientific journal.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2020
– Bibliothèque et Archives Canada, 2020

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2020
– Library and Archives Canada, 2020

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Télec. : 514 340-5665
info@gerad.ca
www.gerad.ca

Importance of data loading pipeline in training deep neural networks

Mahdi Zolnouri ^a

Xinlin Li ^a

Vahid Partovi Nia ^{a,b}

^a Huawei Noah's Ark Lab, Montréal (Québec),
Canada, H3N 1X9

^b GERAD, HEC Montréal, Montréal (Québec),
Canada, H3T 2A7

mahdi.zolnouri@huawei.com

xinlin.li1@huawei.com

vahid.partovinia@huawei.com

April 2020

Les Cahiers du GERAD

G–2020–23–EIW08

Copyright © 2020 GERAD, Zolnouri, Li, Partovi Nia

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs. Les auteurs conservent leur droit d'auteur et leurs droits moraux sur leurs publications et les utilisateurs s'engagent à reconnaître et respecter les exigences légales associées à ces droits. Ainsi, les utilisateurs:

- Peuvent télécharger et imprimer une copie de toute publication du portail public aux fins d'étude ou de recherche privée;
- Ne peuvent pas distribuer le matériel ou l'utiliser pour une activité à but lucratif ou pour un gain commercial;
- Peuvent distribuer gratuitement l'URL identifiant la publication.

Si vous pensez que ce document enfreint le droit d'auteur, contactez-nous en fournissant des détails. Nous supprimerons immédiatement l'accès au travail et enquêterons sur votre demande.

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

- May download and print one copy of any publication from the public portal for the purpose of private study or research;
- May not further distribute the material or use it for any profit-making activity or commercial gain;
- May freely distribute the URL identifying the publication.

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Abstract: *Training large-scale deep neural networks is a long, time-consuming operation, often requiring many GPUs to accelerate. In large models, the time spent loading data takes a significant portion of model training time. As GPU servers are typically expensive, tricks that can save training time are valuable. Slow training is observed especially on real-world applications where exhaustive data augmentation operations are required. Data augmentation techniques include: padding, rotation, adding noise, down sampling, up sampling, etc. These additional operations increase the need to build an efficient data loading pipeline, and to explore existing tools to speed up training time. We focus on the comparison of two main tools designed for this task, namely binary data format to accelerate data reading, and NVIDIA DALI to accelerate data augmentation. Our study shows improvement on the order of 20% to 40% if such dedicated tools are used.*

1 Introduction

Deep neural networks have achieved great successes in various domains such as computer vision [3, 2], natural language processing [10, 9], and speech recognition [17] among others. This is a result of deeper and wider models, which allow modeling large and complex data. As computing hardware has improved, larger data sets are analyzed. It appears that processing power always falls behind data volume and model size. In order to make the training process more efficient, several fields are developing new techniques such as providing dedicated tools to accelerate training and inference, as well as neural model compression to deploy a simpler model with comparable accuracy but fewer operations.

Training acceleration is a difficult task. Let's understand the core of the problem using an a very simple neural network, e.g. logistic regression. Suppose N pairs of input features of dimension d , say \mathbf{x}_i and binary output data, say y_i are available ($\mathbf{x}_i, y_i, i = 1, \dots, N$). A logistic regression model is equivalent to a fully-connected network with a single hidden layer and a single neuron. As the data size gets bigger in terms of N and d , training requires more computation.

A training process optimizes a loss function, here

$$\mathcal{L}(\mathbf{w}) = - \sum_{i=1}^n y_i \log \sigma(\mathbf{x}_i^\top \mathbf{w}) + (1 - y_i) \log(1 - \sigma(\mathbf{x}_i^\top \mathbf{w})), \quad (1)$$

where

$$\sigma(x) = \{1 + \exp(-x)\}^{-1}$$

is the sigmoid activation. For the case of logistic regression iterative re-weighted least squares is often used to optimize \mathcal{L} , which is equivalent to Newton's method. Newton's method starts from an initial estimate \mathbf{w}_0 and updates

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{H}^{-1} \nabla \mathcal{L}(\mathbf{w}) \big|_{\mathbf{w}=\mathbf{w}_t} \quad (2)$$

where \mathbf{H} is the $d \times d$ Hessian of \mathcal{L} and $\nabla \mathcal{L} = \sum_{i=1}^n \nabla \mathcal{L}_i(\mathbf{w})$ is the sum of individual gradients, each of length d . Computing the local approximation of $\mathbf{H} = \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top$ is of $\mathcal{O}(nd^2)$ and factorizing it is of $\mathcal{O}(nd^3)$, if not impossible.

Increasing the number of data points $N \rightarrow \infty$ theoretically makes the optimization easier, because $\mathcal{L}(\mathbf{w})$ has more curvature as N increases, a *blessing*. However, computation of $\mathcal{L}(\mathbf{w})$, $\nabla \mathcal{L}(\mathbf{w})$, and \mathbf{H} becomes a *curse* since all of these quantities are in the form of a sum and their computation may lead to memory overflow. Optimization using Newton's method becomes increasingly hard with large feature size d . Large features are common in almost all machine learning challenges. The remedy for large N is to break computations into smaller sub-operations. The remedy for large d is to switch from a second-order approximation, i.e., Newton's method, to a first-order approximation, i.e., the gradient method.

Computing partial sums is a simple way to overcome the large N issue, so that each partial sum remains within the memory resource constraints. Then the final quantity is computed by summing the partial sums, perhaps with a proper re-scaling. The idea of partial sum is somehow a re-shape of the mini batch training approach.

In neural networks with a large d and n , numerical optimization is simplified to gradient descent in which the hessian \mathbf{H} is replaced by the identity matrix with a positive scalar learning rate

$$\mathbf{H} = \frac{1}{\eta} \mathbf{I}, \eta > 0$$

so the weight update is simplified to

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla \mathcal{L}(\mathbf{w}) \big|_{\mathbf{w}=\mathbf{w}_t} . \quad (3)$$

Furthermore, to benefit from parallel computation, N data points are arranged in n random mini batches, each of size B , i.e. $N = nB$. Each batch has its own gradient

$$\nabla \mathcal{L}_b(\mathbf{w}) = \sum_{i=1}^B \nabla \mathcal{L}_{bi}(\mathbf{w}),$$

which is equivalent to scaling η by n , on average. This allows computations to be run in parallel for each batch [4, 16].

Even if computations are run in parallel, all data still needs to be fed to the optimizer in several rounds of *epochs*, similar to Newton’s method. Therefore, investing in an efficient data loading pipeline plays an important role in training speed [18].

The rest of the paper focuses on clarifying the benefit of a dedicated tool such as DALI¹ for managing data loading implemented by NVIDIA in PyTorch, while a using a convenient data reading format such as Hierarchical Data Format 5 (HDF5) [14] or TensorFlow Record [1] to accelerate file reading.

Data loading is a crucial part of model training in neural networks. It begins by reading the data from a secondary memory storage, such as *Solid State Drive*, then caches it into a primary memory storage, such as *Random Access Memory*. This data transfer includes extra operations like data augmentation to feed the data to the model. See Figure 1.

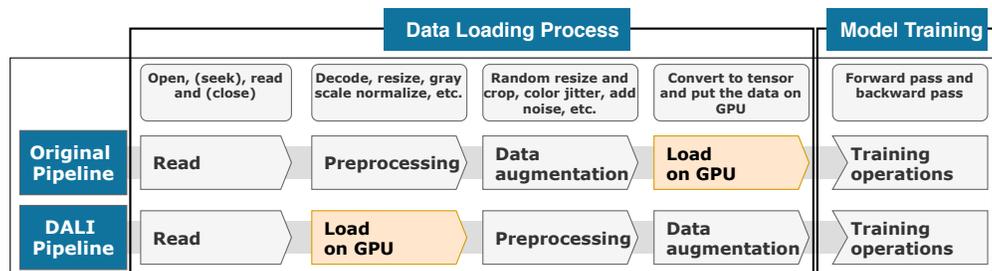


Figure 1: Batch training sequence performed on a GPU with and without DALI

There are two main issues in the data loading process i) reading data directly from files is inefficient ii) resource allocation for extra operations on data overloads the CPU.

Reading data from individual files directly is slow. This happens when the entire data set is not cached in the local memory. Every single file is opened, is read, and is closed sequentially. These sequential operations add considerable overhead to the file retrieval time. One solution to prevent

¹<https://github.com/NVIDIA/DALI>

this overhead is to use the Hierarchical Data Format version 5 (HDF5) [14], which has an open-source library to store, manipulate, and manage the large data set. HDF5 format stores multiple data sets in one file as a multidimensional array of binary data. It also groups storage layout by storing data in fixed-size chunks on disk. Another alternative to HDF5 is the TensorFlow Record (TFRecord) [1], which uses a sequence of binary strings to store data. It allows large data sets to be sequentially loaded to the local memory.

The whole process of data loading is managed by CPU, which can create a bottleneck for model training. This bottleneck happens normally in the case of multi-node-multi-GPU training, as loading batches of data takes more time than forward-backward propagation. To prevent the CPU bottleneck issue, NVIDIA Data Loading Library (DALI) helps by sharing some data loading tasks between the CPU and GPU, to prevent the CPU bottleneck issue. Figure 2 shows how employing this library improves the data loading process considerably. DALI is a collection of highly optimized building blocks and execution engines which provides a full data accelerated pipeline: from reading the data to preparing for training and inference.

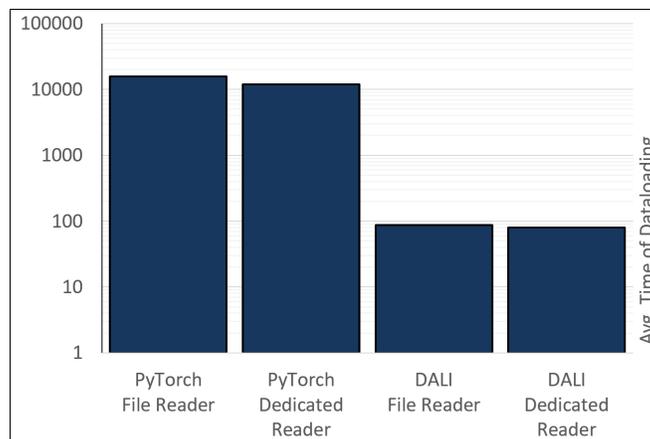


Figure 2: Logarithmic scale of average time of data loading (in micro second) for ResNet-50 [12] on ImageNet data set with batch-size $B = 256$

2 Data format

In order to explore the effect that data format and resource allocation has on data loading performance, we present four pipelines for data loading: with/without dedicated reader, and with/without resource allocation. We used PyTorch version 1.2.0 to implement these pipelines on a computer vision task.

The PyTorch *file reader* pipeline is the PyTorch *DataLoader* class which combines *data set* object with a *sampler* object, to provide a single or multi-process iterators over the data set. This pipeline reads data from individual JPEG files on the storage and uses the PyTorch *Transforms* class to chain several image transformation operations together. This prepares the data for training by doing the data augmentation. By default, all these operations are directed to the CPU.

The *dedicated reader* pipeline is also based on the PyTorch *DATALOADER* class. This pipeline stores data differently, i.e. instead of reading data from individual JPEG files, the entire train and validation data sets are stored as two HDF5 files.

3 Resource allocation

This pipeline uses the NVIDIA DALI library to read data from JPEG files, process and then feed GPUs for training. In the DALI pipeline, the data loading process is shared between CPU and GPU. This means that all operations on data, such as resizing, cropping and data augmentation can be run

on CPU, GPU, or a mix of both. To measure the effect of file formatting, data file retrieval has been done in two cases: reading directly from JPEG files and using a dedicated file reader to read data from TFRecord data set.

4 Training time improvement

We summarize our experiments using four configurations, i.e., with/without a dedicated file reader, and with/without DALI. In order to compare these pipelines, we performed several experiments in two use cases: with few or with extensive *data augmentation* operations. By few operations, we mean resize with random crop and random horizontal flip operations and by extensive, we mean resize with random crop operation, random horizontal flip operation and random adjustment of the brightness, contrast and saturation of an image. Simple data augmentation is applied in most deep learning prototypes, while extensive data augmentation is very common in industry to ensure model robustness in real products. Our experiments are run twice, once on small subset of InsightFace [6, 7, 8, 11] and once on large ImageNet data set [5].

Figure 3 shows that a dedicated data reader is enough to improve epoch time, but only if data is small. However, the DALI pipeline still is a winner for small data requiring extensive data augmentation. This is because DALI distributes the data augmentation operation between CPU and GPU to avoid CPU overcharge. This can be seen by comparing the top left panel, with the top right panel.

If extensive data preprocessing operations are performed in a larger ImageNet data, CPU workload becomes the bottleneck even with few data augmentation operations. Consequently, epoch time increases in data loading and DALI can avoid the CPU overcharge by performing some of these operations on GPU. Figure 3 (bottom right) shows that DALI improves data loading from 2200 seconds to

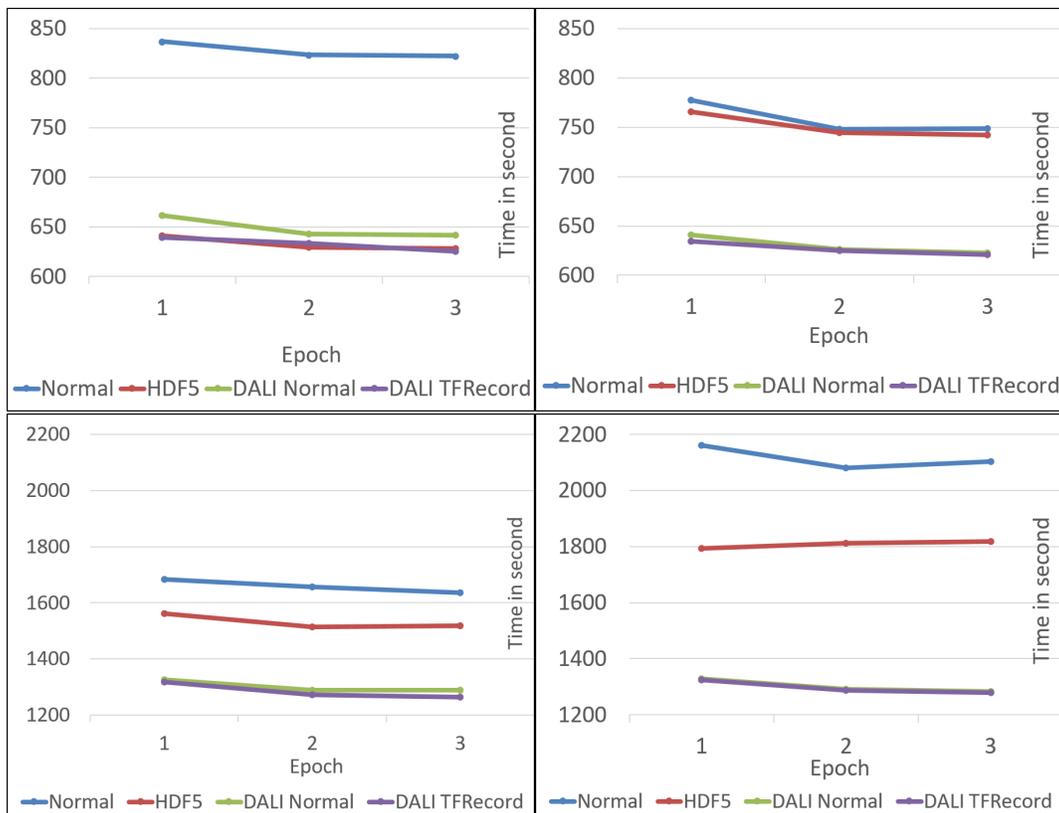


Figure 3: Epoch time for the small subset of InsightFace (top panels) and large ImageNet data (bottom panels), while few data processing operations are performed (left panels) and while extensive preprocessing operations are performed (right panel). The experiments in top panels were performed on 4 GPUs (NVIDIA TITAN 12 GB memory) and in bottom panels on 8 GPUs (NVIDIA TESLA V100 32GB memory)

1300 seconds, to gain performance benefit of about 40%. Using only a dedicated reader without DALI improves the training time from 1800 seconds to 1300 seconds, giving a performance improvement of 30%. This effect is also visible in Figure 4 on a single epoch time. DALI fuses multiple operations such as cropping and normalizing and run it on one GPU CUDA kernel. This speeds up data augmentation process by reducing the number of memory access.

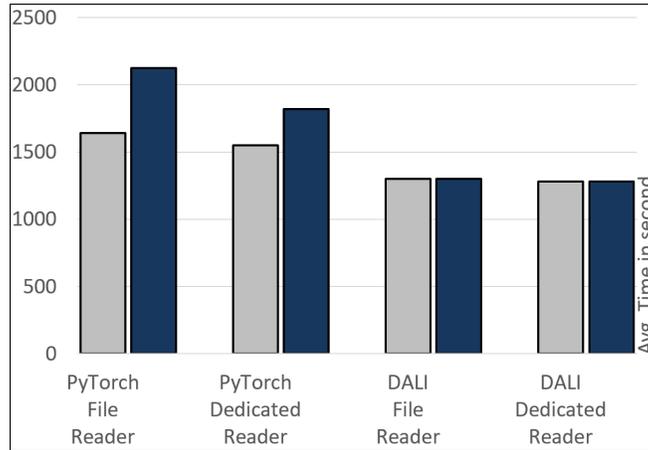


Figure 4: Time comparison for few (gray) and extensive (dark) data augmentations

5 Data loading improvement

Let’s move from epoch time to data loading time by removing forward and backward pass from training time, see Figure 1. Data loading contributes about 40% to the epoch time, see Figure 5.

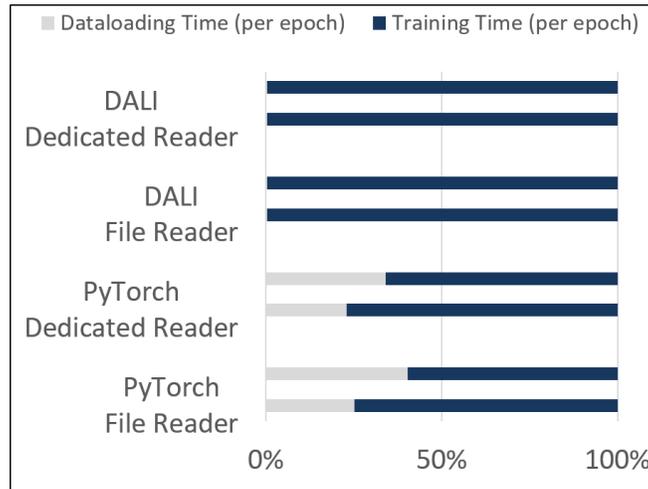


Figure 5: Percentage of data loading time (gray) added on top of training time (dark) in a single training epoch. Each data loader test case has two bar charts for showing few and extensive data augmentations

Figure 6 confirms that the DALI pipeline considerably improves data loading by a factor of 100x. However, for very large models, the GPU is only required to perform forward and backward passes, so loading the CPU for data is wiser. Therefore, it is important to keep the CPU and GPU load well balanced through DALI load option. Figure 7 confirms the same message when training time is stacked on data loading time to measure the epoch time overall.

In Figure 7 data loading is stacked on training to reflect the epoch time overall. There is a small difference between the common data loader pipeline and NVIDIA DALI data loader pipeline. Using

the GPU resources for data loading may slow down the overall model training time if it fails to balance CPU and GPU load.

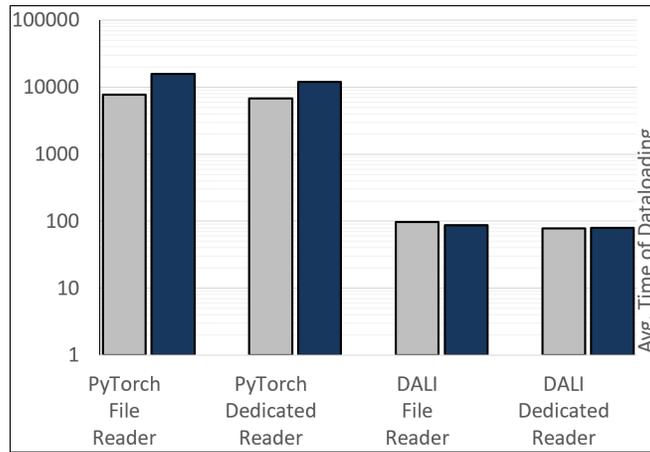


Figure 6: Logarithmic scale plot of average data loading time (per epoch), four data loader pipelines: with/without DALI, and with/without a dedicated reader. Time for few (gray) and extensive (dark) data augmentations

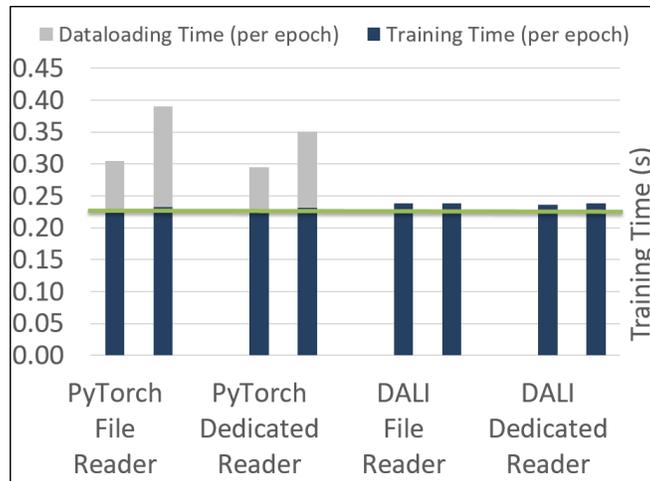


Figure 7: Training time on four data loader pipelines: with/without DALI, and with/without a dedicated reader. Each data loader pipeline has two bar charts for showing few and extensive data augmentations

By default, DALI uses the first GPU slot to perform data loading process. However, the NVIDIA APEX library uses multiple GPUs for this task. This flexibility becomes increasingly important for large models while all data and models cannot be loaded into a single GPU and multi-GPU operation becomes a necessity.

6 Conclusion

NVIDIA DALI provides an effective alternative to common data loading process. It provides a full pipeline of optimizations including data readers and tools to accelerate training and inference. It also enables most data augmentation operations to be performed on GPU and on CPU. In addition, DALI prepared a full pipeline for common data sets like MS-COCO data set [15] as well as provides a reader for TFRecord and CAFFE LMDB data formats [13]. DALI remains extremely flexible by supporting ExternalSource operator so that implementation of unsupported readers such as HDF5 becomes feasible.

Here, we focused on large models, in which DALI GPU improves training time. However, training is faster with DALI CPU for small networks.

Acknowledgment

The authors would like to thank Eyyüb Sari and Vanessa Courville for their assistance in this project. We appreciate fruitful technical discussions with Huawei Cloud Core Shanghai Gang Chi and Pengcheng Tang as well as Jiajin Zhang from Noah's Ark Shenzhen engineering team.

References

- [1] Martín Abadi. Tensorflow: learning functions at scale. In Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, pages 1–1, 2016.
- [2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.
- [3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1251–1258, 2017.
- [4] Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. In Advances in neural information processing systems, pages 1647–1655, 2011.
- [5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.
- [6] Jiankang Deng, Jia Guo, Xue Niannan, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In CVPR, 2019.
- [7] Jiankang Deng, Jia Guo, Zhou Yuxiang, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild. In arxiv, 2019.
- [8] Jiankang Deng, Anastasios Roussos, Grigorios Chrysos, Evangelos Ververas, Irene Kotsia, Jie Shen, and Stefanos Zafeiriou. The menpo benchmark for multi-pose 2d and 3d facial landmark localisation and tracking. *IJCV*, 2018.
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
- [10] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.
- [11] Jia Guo, Jiankang Deng, Niannan Xue, and Stefanos Zafeiriou. Stacked dense u-nets with dual transformers for robust face alignment. In BMVC, 2018.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
- [13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675–678, 2014.
- [14] Sandeep Koranne. Hierarchical data format 5: Hdf5. In Handbook of Open Source Tools, pages 191–200. Springer, 2011.
- [15] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.
- [16] Krishnamurty Muralidhar and Rathindra Sarathy. Data shuffling—a new masking approach for numerical data. *Management Science*, 52(5):658–670, 2006.
- [17] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In IEEE 2011 workshop on automatic speech recognition and understanding, number CONF. IEEE Signal Processing Society, 2011.
- [18] Chih-Chieh Yang and Guojing Cong. Accelerating data loading in deep neural network training. arXiv preprint arXiv:1910.01196, 2019.