https://doi.org/10.2352/EI.2024.36.12.HPCI-196

High-Performance Data Loader for Large-Scale Data Processing

Edgar Josafat Martinez-Noriega[1], Chen Peng[1], and Rio Yokota[2]; National Institute of Advanced Industrial Science and Technology[1] and Tokyo Institute of Technology[2]; Japan

Abstract

The utilization of supercomputers and large clusters for big-data processing has recently gained immense popularity, primarily due to the widespread adoption of Graphics Processing Units (GPUs) to execute iterative algorithms, such as Deep Learning and 2D/3D imaging applications. This trend is especially prominent in the context of large-scale datasets, which can range from hundreds of gigabytes to several terabytes in size. Similar to the field of Deep Learning, which deals with datasets of comparable or even greater sizes (e.g. LION-3B), these efforts encounter complex challenges related to data storage, retrieval, and efficient GPU utilization. In this work, we benchmarked a collection of high-performance general dataloaders used in Deep Learning with a dual focus on user-friendliness (Pythonic) and high-performance execution. These dataloaders have become crucial tools. Notably, advanced dataloading solutions such as Webdatasets, FFCV, and DALI have demonstrated significantly superior performance when compared to traditional PyTorch general data loaders. This work provides a comprehensive benchmarking analysis of high-performance general dataloaders tailored for handling extensive datasets within supercomputer environments. Our findings indicate that DALI surpasses our baseline PyTorch dataloader by up to 3.4x in loading times for datasets comprising one million images.

Introduction

The employment of large clusters and supercomputers[1] for the processing of big data has garnered significant attention, primarily driven by the widespread integration of Graphics Processing Units (GPUs) for the execution of iterative algorithms, including Deep Learning and 2D/3D imaging applications. This phenomenon is particularly pronounced in the realm of large-scale datasets, spanning from hundreds of gigabytes to several terabytes in magnitude. Analogous to the domain of Deep Learning, which contends with datasets of comparable or larger scales (e.g., JFT-300M[2] and LION-5B[3]), these endeavors confront intricate challenges pertaining to data storage, retrieval, and the effective utilization of GPUs. Moreover, state of the art architecture such as the Vision Transformer is data-hungry, prevailing the challenge encountered by this architectural framework on to the need for a substantial volume of data. In order to attain cutting-edge performance levels through transfer learning, such models conventionally mandate in excess of 100 million images [4].

Moreover, dataloaders are integral components in the deep learning workflow, facilitating data ingestion, pre-processing, and augmentation. The primary objective of a dataloader lies in executing the operations essential for transferring data instances from a storage repository to the memory space situated alongside the processing units, facilitating their utilization during training to construct a batch of samples intended for input. The execution of these operations is limited or constrained by the bandwidth of the storage system, particularly its I/O. Consequently, contingent upon the hardware specifications of the system, the filesystem supporting it, and the data transfer rate of the connection with the computational units, it can exert a significant impact on the overall duration required to finalize the training. As the size of datasets continues to grow exponentially, the efficient loading of data onto GPUs becomes a paramount concern.

In this study, we present a benchmarking analysis of high-performance general dataloaders tailored for managing extensive datasets within supercomputer environments. Our investigation focuses on two critical dimensions: user-friendliness, particularly within the Python programming context, and high-performance execution. We conduct benchmarking evaluations across a spectrum of general dataloader solutions, encompassing established options from the PyTorch[5] library and advanced alternatives such as Webdatasets[6], FFCV [7], and DALI [8]. The objective is to evaluate their efficacy in terms of data loading speed, especially when confronted with datasets comprising millions of images or more. Through comparative analysis, we elucidate the strengths and limitations of these dataloaders in supercomputing environments. Initial findings from our benchmarking experiments demonstrate that DALI, an open-source library developed by NVIDIA, outperforms our baseline PyTorch dataloader by up to 3.4x in loading large datasets. This substantial enhancement underscores the potential of DALI in expediting the data ingestion phase within deep learning workflows on supercomputers. Additionally, we explore the user-friendliness aspect of these dataloaders, emphasizing the importance of seamless integration with popular deep learning frameworks like PyTorch. Moreover, we discuss the implications of our findings in the context of contemporary deep learning applications, underscoring the burgeoning need for handling massive datasets across domains such as computer vision. Efficient data loading and pre-processing are paramount for minimizing training duration and maximizing the efficient utilization of supercomputing resources.

Related Work

The emergence of deep learning has engendered an escalating demand for training larger models with expansive datasets, prompting extensive investigations into the bottlenecks encountered during the training process. For instance, Mohan *et al.* [9] conducted a comprehensive examination of data stalls within various models, revealing that these stalls could account for up to 65% of the total training time in certain scenarios. The insights gleaned from their research informed the development of a coordinated caching and pre-processing library named CoorDL, which has demonstrated the capacity to accelerate training by up to 15-fold in distributed training scenarios across two servers. Notably, this study introduced a sizable dataset for Audio Classification,

denoted as M5 [10], with a size of approximately 950 GB, along with performance evaluations on ImageNet-21k, boasting a size of 1.3 TB [11]. Additionally, Matson *et al.* [12] presented an extensive investigation into machine learning workloads, focusing on both training and inference tasks across various AI benchmarks . Their study primarily aimed to assess the time required to attain specific accuracy thresholds, a metric demanding incremental resource allocation and unsuitable for evaluating data loaders and associated parameters. Moreover, Wu *et al.* [13] conducted a study on memory and CPU utilization, scrutinizing access and usage patterns through an array of parallel libraries . Their analysis included investigations into the impact of batch sizes on training efficiency and accuracy, providing valuable insights into optimizing computational resources for deep learning tasks.

On another hand, reaching large datasets such as JFT-300M and LION-5B represents some challenges. In the firs place JFT and its variants are proprietary and not open to the public. On the later one, LION-5B is open to the public consisting in 5B links. Even employing high-end CPUs downloading the whole dataset from their servers over the internet would take some months and also some computing resources to check consistency. Moreover, the use of these large datasets are not exempt from ethical concerns, such as societal biases, privacy issues, and copyright violations. Thus, synthetic datasets are gained traction to perform transfer learning. For instance, Barad *et al.* [14] presented a synthetic dataset consisting of 21,000 programs, each tasked with generating a varied collection of synthetic images. These images are generated using OpenGL Shading Language (GLSL) programs, recognized for their versatility and adjustability. These scrtipts are executed on the GPU utilizing the OpenGL API. Furthermore, Formula-Driven Supervised Learning (FDSL) was originally introduced by Kataoka *et al.* [15], where synthetic images and their corresponding labels are generated using mathematical formulas. These unique fractal images offer a distinctive approach to constructing extensive, diverse, and dynamically evolving datasets suitable for training purposes. Numerous endeavors have been proposed in the FDSL domain [16, 17], encompassing the development of innovative datasets such as Visual Atom [18] and MV-Fractal [19]. In this study, we utilize a modified version of the FractalDB, which facilitates the generation of large datasets for our experiments.

Method

This section furnishes a comprehensive overview of the dataloader and the challenges it encounters when processing extensive datasets within distributed computing environments. Additionally, we elucidate the assortment of dataloaders subjected to benchmarking, along with their impact on the dataset characteristics both prior to and subsequent to their utilization.

The Deep Learning Dataloader

As previously mentioned, the dataloader assumes a pivotal role in deep learning workflows. This indispensable tool is entrusted with the responsibility of loading and preprocessing datasets, thereby rendering them accessible to the model during both training and inference phases. Serving as a conduit between the raw data and the deep learning model, it facilitates the seamless ingestion and manipulation of data. Furthermore, the dataloader retrieves samples from the dataset, encompassing various

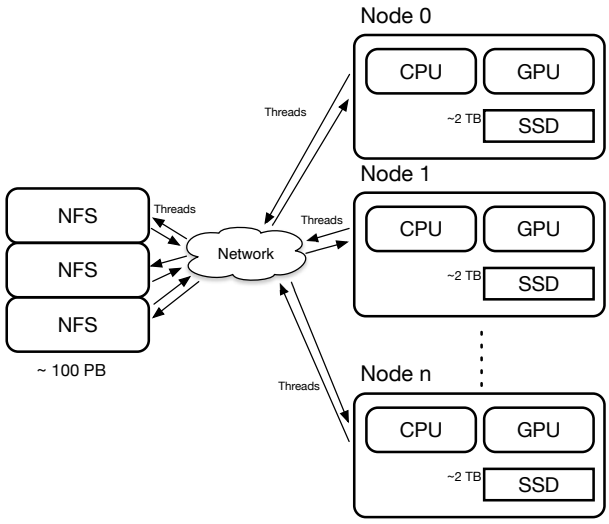

Figure 1. *The high traffic on a distributed system when datasets are retrive from network file system.*

data modalities such as images, text, audio, and more, and efficiently loads them into memory, subsequently transferring them to the GPU memory. In addition to this primary function, it undertakes ancillary tasks such as image resizing, pixel value normalization, text tokenization, and data conversion into numerical format. The dataloader orchestrates the organization of data into batches, wherein multiple samples are amalgamated, thus functioning as the primary scheduler for batch processing. This batching mechanism enhances computational efficiency by enabling the model to concurrently process multiple samples, thereby harnessing the power of parallelism. Lastly, the dataloader introduces randomness by shuffling the data, thereby thwarting any potential memorization of sample order by the model during training.

Moreover, within the domain of deep learning, various data types with distinct characteristics necessitate handling, including Image, Text, Numerical, Audio, and Video data. These data types serve specific purposes, such as time series forecasting, image pattern recognition, or video description tasks. Consequently, acquiring and pre-processing such diverse data pose unique challenges for a single dataloader to provide an optimal solution. While a general-purpose dataloader is available in PyTorch [5], purportedly supporting all data types, research by Hambardzumyan *et al.* [20] underscores that handling numerous small datasets of SQL datatypes presents disparate challenges compared to processing an equivalent volume of items in images or videos. Additionally, as depicted in Figure 1, the scalability of distributed training across nodes amplifies the network load required to access datasets for multiple CPU threads on each node. Given the focus of this study on large-scale vision transformers or image processing, we opt to benchmark dataloaders that prioritize compression or exploit alternate hardware resources, such as GPUs, for pre-processing. This strategy aims to mitigate the computational bottlenecks during distributed training.

Webdataset

The Webdatasets data loader was introduced [6] to streamline the acquisition of datasets directly from online repositories for integration into machine learning pipelines. It compresses the

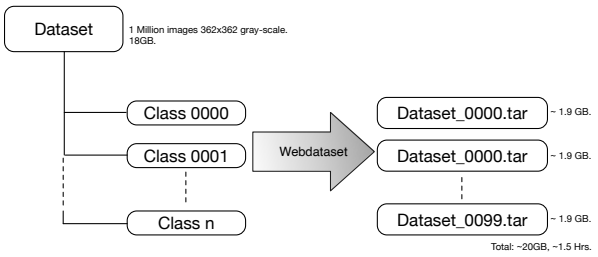

Figure 2. *Conversion process when a normal dataset is processed by Webdataset. Original size and time for pre-processing are included for a million image dataset.*

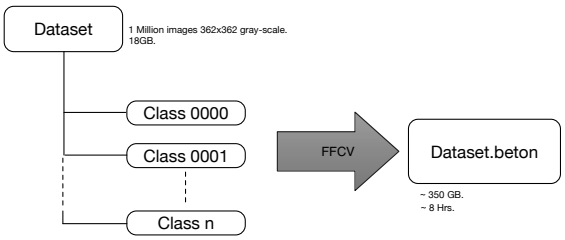

Figure 3. *Conversion process when a normal dataset is processed by FFCV. Original size and time for pre-processing are included for a million image dataset.*

whole dataset into POSIX tar archives. Furthermore, offers a flexible and adaptable interface for accessing diverse datasets available on the web, eliminating the need for physical presence within the local network and circumventing the manual downloading and pre-processing steps. Notably, one of its notable features is dynamic dataset retrieval, empowering users to fetch datasets from online repositories dynamically, encompassing public datasets hosted on websites, cloud storage platforms, or data APIs. Additionally, this dataloader boasts on-the-fly pre-processing capabilities, enabling users to apply transformations and manipulations to the data during the loading process. Furthermore, Webdatasets facilitates streaming data loading, enabling efficient processing of large datasets without requiring the entire dataset to be loaded into memory simultaneously. This streaming functionality is particularly advantageous for handling datasets exceeding available memory resources. Moreover, the library offers support for parallel data loading, capitalizing on multi-core processors or distributed computing environments to expedite the loading process. Furthermore, Webdatasets seamlessly integrates with prominent deep learning frameworks such as PyTorch and TensorFlow, enabling users to seamlessly incorporate web-based datasets into their machine learning models and experiments.

As depicted in Figure 2, the left side illustrates the conventional structure of datasets loaded by PyTorch, whereas the right side showcases the resulting format subsequent to pre-processing with Webdataset, culminating in the conversion of the dataset into POSIX tar archives. It is noteworthy to mention that the conversion of the original dataset to the Webdataset format necessitates certain computational resources. While this operation imposes minimal burden for smaller datasets such as CIFAR100 [21] or Stanford Cars [22], it may become considerable for larger datasets.

FFCV

The FFCV dataloader was proposed [7] to efficiently manage image datasets, emphasizing the optimization of data loading performance through methodologies such as caching and parallel processing. Introducing a proprietary dataset format termed the ”beton” extension, this dataloader encapsulates data in a binary-like representation, optionally compressed in JPEG format for network transmission, thus maximizing throughput and performance. A notable feature of the FFCV dataloader is its file caching mechanism, which preserves pre-processed image data in memory or on disk to mitigate the overhead associated with recurrent data loading and pre-processing tasks. This approach significantly augments data loading speed and overall training efficiency, particularly in scenarios involving numerous augmentation techniques. Moreover, FFCV employs parallel processing strategies to concurrently load and pre-process image data, leveraging multi-core processors to expedite the loading process. By distributing computational tasks across multiple cores, FFCV enhances throughput and reduces loading times, particularly when handling extensive image datasets. Furthermore, FFCV facilitates efficient image augmentation capabilities, enabling the application of diverse transformations and augmentations to image data during the loading phase. These augmentations bolster the diversity and resilience of the training data, thereby enhancing model generalization and performance.

Figure 3 illustrates the conventional structure of a dataset on the left, contrasted with the converted output produced by the FFCV conversion library on the right. Notably, the converted output consists of a single file encapsulating the entire dataset, enhancing bandwidth utilization when accessed by CPUs on the nodes due to its binary-like data format representation. However, akin to Webdataset, the creation of the ”beton” file by FFCV necessitates additional computational resources. While this may not pose a significant challenge for smaller datasets, it can present considerable difficulties for larger datasets, particularly those incorporating numerous augmentation techniques inside the images.

NVIDIA Data Loading Library

The NVIDIA Data Loading Library (DALI) was proposed [8] to efficiently manage large-scale datasets for deep learning, particularly for computer vision applications. Its primary objective is to optimize the data loading process, thereby maximizing GPU and CPU utilization to accelerate training and inference workflows. DALI incorporates GPU-accelerated data loading, leveraging the GPU to minimize CPU-GPU data transfer overhead and enhance overall performance. It supports parallel data loading and augmentation, allowing multiple CPU threads to concurrently process data, thus enhancing throughput. Additionally, DALI offers a comprehensive suite of image pre-processing operations, including resizing, cropping, rotation, and color augmentation, all performed on-the-fly during data loading to mitigate memory overhead. Furthermore, DALI seamlessly integrates with popular deep learning frameworks such as TensorFlow and PyTorch. Its GPU acceleration, parallel execution capabilities, and deep learning framework compatibility collectively contribute to enhanced training efficiency and model performance.

In contrast to the Webdataset and FFCV dataloaders examined in this study, DALI does not need the transformation of the original dataset format or structure. DALI optimizes performance

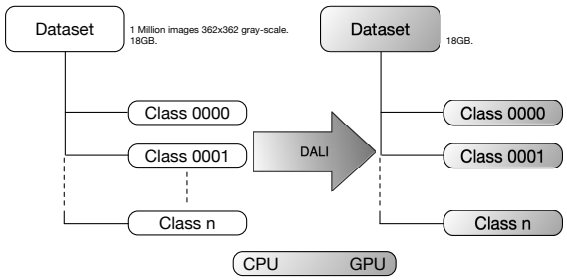

Figure 4. *Conversion process when a normal dataset is processed by DALI. Original size for pre-processing is included.*

by leveraging the GPU during data retrieval, encompassing tasks such as decoding JPEG format and performing augmentations that necessitate image normalization, resizing or other operations. As illustrated in Figure 4, the dataset remains unaltered, preserving its original size and format.

Evaluation

In this section, we present the outcomes of our experiments evaluating three distinct dataloaders. We outline the experimental setup and the environment in which the experiments were conducted. We provide a brief description of the FractalDB [15] use in our experiments. Our analysis primarily focuses on measuring image retrieval time without considering the whole training. For the main tests presented herein, we offer various configurations for the dataloaders capable of utilizing compression techniques. Moreover, we compare these findings against those obtained using the PyTorch dataloader, which serves as our baseline reference.

Experimental Environment

We leveraged the AI Bridging Cloud Infrastructure (ABCI) supercomputer [23], renowned for its specialization in AI computing tasks. This supercomputer comprises two distinct node configurations: those featuring A100 GPUs and those housing V100 GPUs. The Volta-equipped nodes, are equipped of 1,088 compute nodes, each integrating 2 Intel Xeon Gold 6148 CPUs (a total of 40 cores), 384 GiB of DRAM, 4 NVidia V100 GPUs, and InfiniBand EDR NICs. Additionally, each node includes with 1.6TB of local storage and shares access to a 35PB Lustre parallel filesystem. On the other hand, the Ampere nodes encompass 120 compute nodes, each with 2 Intel Xeon Platinum 8360Y Processors (resulting in a total of 76 cores), 512 GiB of DRAM, 8 NVidia A100 GPUs, and InfiniBand HDR connectivity. Furthermore, every node includes 2.0TB of local storage and is linked to the Lustre NFS for shared filesystem access.

The Fractal Dataset

Fractals are complex geometric shapes that exhibits self-similarity at different scales. Firstly introduced by Kataoka *et al.* [15], the FractalDB is a collection of fractal images generated by a render method that uses the Iterated Function System (IFS). In this sense, we can search and generate an arbitrary number of fractals and its corresponded labels, thus, we can form any dataset size desired. Figure 5 shows an example of the different patterns generated on each class. More specifically, each C class has a hyperparameter set θ_y to generate fractals as $F_y(s) = R(\theta_y, s)$, where R is the rendering routine and s is a random seed that creates variations within the class C. The hyperparameter $\theta_y = \{(w_i, p_i)\}_{i=1}^{n}$

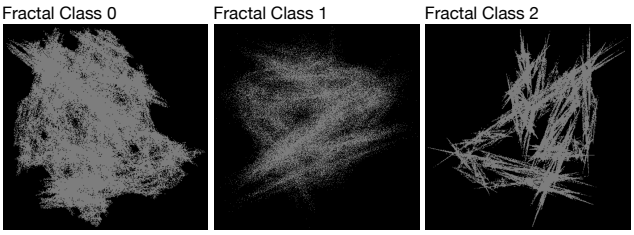

Figure 5. *Sample fractal images from FractalDB.*

consist of functions $w_i : \mathbb{R}^2 \to \mathbb{R}^2$ with probability mass function p_i. The original FractalDB comprises a million images divided in a C =1k classes, and 1k instances per class. In this study we utilized such dataset.

Performance per Batch Step

In this experiment, we assessed the performance of all dataloaders exclusively during a large batch step, which is a common practice in Deep Learning for training data delivery. The batch size was set to 500 for consistency. We quantified the total time elapsed from file retrieval to its aggregation into batches and the application of basic transformations (*Resize(), ToTensor(), and Normalize()*) until it is transferred to the GPU using the operation *ToDevice()*. To emphasize raw dataloader performance, only a subset of essential transformations was employed for this experiment. The experiments were conducted on Volta nodes, and the dataloader configurations were as follows:

- For FFCV, all three transformations were included during beton file creation, with three different configurations (10%,50%, and 90%) utilized for compressing images using JPEG format.
- For Webdataset, shards were divided into 100 and 1,000, with the inclusion of a file pair inside each shard, comprising the image and its corresponding Class ID.
- DALI incorporated identical transformation routines, executed from both CPU and GPU.

Figure 6 illustrates the performance comparison among all dataloaders. It is evident that the baseline approach using PyTorch alone to retrieve 500 images via the file system exhibits the poorest performance, even exceeding the time required by Webdatasets, FFCV, or DALI by a considerable margin. As for the FFCV loader, the compression ratio significantly influences loading times, as it introduces computational overhead during decoding. However, FFCV demonstrates improved performance when the compression ratio is set to 10%. Webdatasets exhibit the shortest loading times for both configurations, with the optimal performance observed when shards are set to 1,000. Furthermore, we observed performance variations based on the number of shards used to partition the original dataset. DALI outperforms FFCV and achieves comparable results to Webdatasets. The observed variations in loading times suggest that network traffic influences performance when accessing images. Consequently, we conducted additional measurements by relocating the dataset to the SSD of the node, thereby minimizing the impact of network IO, which we assume to be negligible.

Figure 7 depicts a comparable experiment to the previously described one. In this instance, the dataset was allocated on the SSD. The considerable reduction in loading time observed in the

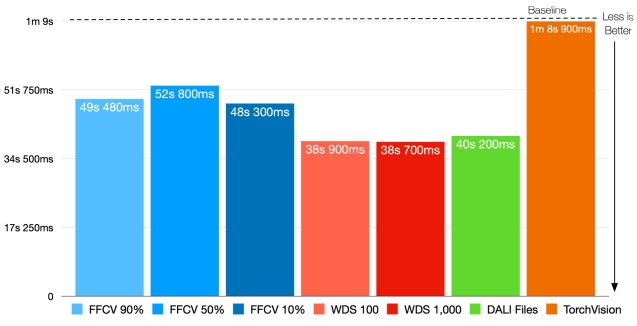

Figure 6. *Time to load one batch step (500 images). Location of the dataset is on the file system NFS.*

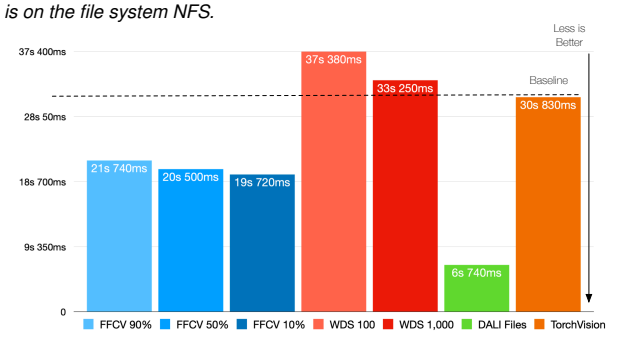

Figure 7. *Time to load one batch step (500 images). Location of the dataset is on the local SSD.*

baseline, or when using the PyTorch dataloader, is noteworthy, decreasing from 68 seconds to 30 seconds. This emphasizes the substantial burden imposed by IO during training, particularly when employing multiple nodes. Interestingly, the Webdatasets loader exhibits the poorest performance among all other dataloaders, even with the 1,000 shards configuration. FFCV follows a similar trend as in the preceding experiment, with the 10% compression ratio yielding the best loading time. Notably, DALI demonstrates the most remarkable improvement in loading time, decreasing from 40 seconds to 6 seconds. This represents an order of magnitude enhancement in performance, attributed to the concurrent transformations performed by both CPU and GPU. Directly comparing both experiments reveals a reduction of more than half the time for all dataloaders, except for Webdatasets, which performs similarly to when the dataset was on the NFS.

Performance per Epoch

In this experiment, we assessed the performance of all dataloaders across a full epoch, encompassing the entire dataset. This evaluation extends beyond previous results, involving the measurement of the complete time required to load 1 million images. Additionally, we introduced a more intricate set of transformations, including AutoAugment [24], which imposes greater CPU load during batch formation. It is noteworthy that integrating AutoAugment into FFCV necessitated implementing this algorithm within their custom transformations, which proved to be a non-trivial task. Consequently, for FFCV, AutoAugment was not included when creating the beton file. In contrast, DALI, which also offers custom transformations, supports algorithms such as AutoAugment. The complete list of transformations includes *AutoAugment()*, *ToTensor()*, and *Normalize()*, with the time taken to reach the GPU measured using *ToDevice()*. In order to provide a better insight on how much would take to load 1 Million images as fast as possible we conducted the experiment allocating the whole dataset on the SSD. The experiments were conducted on Volta nodes on ABCI.

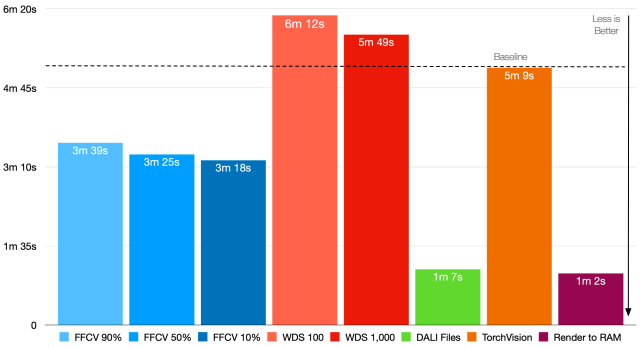

Figure 8. *Time to load one full epoch. The location of the dataset is on the local SSD.*

Figure 8 presents the performance of the dataloaders when loading the entire 1 million-image dataset. The results shows a similar trend to those observed during the batch step, as the primary challenge in this experiment lies in handling the larger volume of images. The baseline PyTorch loader required 5 minutes to load the entire dataset, a performance closely matched by Webdataset with the 1,000 shards configuration. However, reducing the number of shards in Webdataset increased the loading time by one minute. FFCV demonstrated the best performance with the 10% compression ratio, loading the entire dataset in under three minutes, which is more than half the time compared to the baseline. DALI excelled in loading time, completing the task in close to 1 minute and 7 seconds which outperforms up to 3.4x to the baseline. This represents the most efficient option for loading large images, particularly when combined with relocating the dataset to a local SSD and utilizing DALI. Additionally, we conducted an additional measurement to assess the time required for a dataset to be loaded directly into RAM memory when created on-the-fly. We observed that it was faster that even DALI under 1 minute and 2 seconds. This measurement is part of our future work, focusing on enhancing dataset access and creation speed.

Conclusion

We have presented a benchmark that provides a valuable reference for researchers and practitioners involved in deep learning tasks on supercomputing platforms. Our benchmark offers insights and evaluations of high-performance general dataloaders, with a primary focus on their loading speed. Our findings underscore DALI as a promising solution for accelerating data loading on supercomputers. Additionally, we have observed that a straightforward solution to enhance loading times is to relocate datasets to local storage. In our experiments, leveraging the SSD within each node significantly reduced loading times, halving them compared to using the file system. Furthermore, we noted variations among dataloaders, with Webdatasets maintaining consistent performance regardless of whether the dataset is stored on the SSD or over NFS. FFCV may be a suitable option, particularly when extensive augmentation or preprocessing is not required. Moreover, DALI demonstrates superior loading times, making it the optimal choice for handling large datasets. As datasets continue to expand in size and complexity, efficient

data management becomes increasingly critical task.

Acknowledgements

This paper is based on results obtained from a project, JPNP20006, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] TOP500, The List, https://www.top500.org/, 2023, [November. 2023].

[2] Sun. C, Shrivastava. A, Singh. S, Gupta. A. Revisiting unreasonable effectiveness of data in deep learning era. In Proceedings of the IEEE international conference on computer vision 2017 (pp. 843-852).

[3] Schuhmann. C, Beaumont. R, Vencu. R, Gordon. CW, Wightman. R, Cherti. M, Coombes. T, Katta. A, Mullis. C, Wortsman. M, Schramowski. P. LAION-5B: An open large-scale dataset for training next generation image-text models. In Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

[4] Kolesnikov. A, Beyer. L, Zhai. X, Puigcerver. J, Yung. J, Gelly. S, Houlsby. N. Big transfer (bit): General visual representation learning. InComputer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16 2020 (pp. 491-507). Springer International Publishing.

[5] PyTorch Core Team, PyTorch Vision Docs, https://pytorch.org/vision/stable/datasets.html, [January 2024].

[6] Aizman. A, Maltby. G, Breuel. T. High performance I/O for large scale deep learning. In 2019 IEEE International Conference on Big Data (Big Data) 2019 Dec 9 (pp. 5965-5967). IEEE.

[7] Leclerc. G, Ilyas. A, Engstrom. L, Park. SM, Salman. H, Madry. A. FFCV: Accelerating training by removing data bottlenecks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2023 (pp. 12011-12020).

[8] Nvidia Data Loading Library - DALI. https://developer.nvidia.com/dali, 2024, [January 2024].

[9] Mohan. J, Phanishayee. A, Raniwala. A, Chidambaram. V. Analyzing and mitigating data stalls in DNN training. In Proceedings of the VLDB Endowment, Volume 14, Issue 5, pp 771–784.

[10] Defferrard. M, Benzi. K, Vandergheynst. P, Bresson. X. FMA: A dataset for music analysis. 18th International Society for Music Information Retrieval Conference (ISMIR), 2017.

[11] ImageNet Dataset, https://www.image-net.org/index.php, 2024, [January 2024].

[12] Mattson. P, Cheng. C, Diamos. G, Coleman. C, Micikevicius. P, Patterson. D, Tang. H, et al. Mlperf training benchmark. Proceedings of Machine Learning and Systems, 2:336–349, 2020.

[13] Wu. Y, Liu. L, Pu. C, Cao. W, Sahin. S, Wei. W, Zhang. Q. A comparative measurement study of deep learning as a service framework. IEEE Transactions on Services Computing. 2019 Jul 18;15(1):551-66.

[14] Baradad. M, Chen. CF, Wulff. J, Wang. T, Feris. R, Torralba. A, Isola. P. Procedural Image Programs for Representation Learning. In Advances in Neural Information Processing Systems 2022 Nov 26.

[15] Kataoka. H, Okayasu. K, Matsumoto. A, Yamagata. E, Yamada. R, Inoue. N, Nakamura. A, Satoh. Y. Pre-training without natural images. In Proceedings of the Asian Conference on Computer Vision 2020.

[16] Kataoka. H, Hayamizu. R, Yamada. R, Nakashima. K, Takashima. S, Zhang. X, Martinez-Noriega. EJ, Inoue. N, Yokota. R. Replacing Labeled Real-image Datasets with Auto-generated Contours. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2022 (pp. 21232-21241).

[17] Nakashima. K, Kataoka. H, Matsumoto. A, Iwata. K, Inoue. N, Satoh. Y. Can vision transformers learn without natural images?. In Proceedings of the AAAI Conference on Artificial Intelligence 2022 Jun 28 (Vol. 36, No. 2, pp. 1990-1998).

[18] Takashima. S, Hayamizu. R, Inoue. N, Kataoka. H, Yokota. R. Visual atoms: Pre-training vision transformers with sinusoidal waves. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2023 (pp. 18579-18588).

[19] Yamada. R, Takahashi. R, Suzuki. R, Nakamura. A, Yoshiyasu. Y, Sagawa. R, Kataoka. H. MV-FractalDB: formula-driven supervised learning for multi-view image recognition. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2021 Sep 27 (pp. 2076-2083). IEEE.

[20] Hambardzumyan. S, Tuli. A, Ghukasyan. L, Rahman. F, Topchyan. H, Isayan. D, McQuade. M, Harutyunyan. M, Hakobyan. T, Stranic. I, Buniatyan. D. Deep lake: A lakehouse for deep learning. arXiv preprint arXiv:2209.10785. 2022 Sep 22.

[21] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. 2009.

[22] Krause. J, Stark. M, Deng. J, Fei-Fei. L. 3d object representations for fine-grained categorization. In Proceedings of the IEEE international conference on computer vision workshops 2013 (pp. 554-561).

[23] National Institute of Advanced Industrial Science and Technology, ABCI Supercomputer, https://abci.ai, 2023, [January 2023].

[24] Cubuk. ED, Zoph. B, Mane. D, Vasudevan. V, Le. QV. Autoaugment: Learning augmentation policies from data. arXiv preprint arXiv:1805.09501. 2018 May 24.

Author Biography

Edgar Josafat Martinez-Noriega obtained his Doctorate in Computer Science from the University of Electro-Communications, Tokyo in 2022. Following this, he has been employed as a Post-Doctoral Researcher at the National Institute of Advanced Industrial Science and Technology (AIST), working on the application of synthetic datasets for large-scale deep learning. His research focuses on parallel computing, computer graphics, and deep learning.

Peng Chen is a researcher at National Institute of Advanced Industrial Science and Technology (AIST). Also, he is working as a visiting scientist at RIKEN Center for Computational Science (RIKEN-CCS), Japan. He received the B.E. degree in navigation from Dalian Maritime University, China, in 2005; the M.E. degree in traffic information engineering and control from Shanghai Maritime University, China, in 2007; the Ph.D. from Tokyo Institute of Technology, Japan, in 2020. His research interests include parallel computing, image processing, and machine learning.

Rio Yokota is a professor at the Global Scientific Information and Computing Center, Tokyo Institute of Technology. His research focuses on high performance computing, linear algebra, and machine learning. He has developed several libraries, including ExaFMM for fast multipole methods, and Hatrix for hierarchical low-rank algorithms. He has received the Gordon Bell prize in 2009 using the first GPU supercomputer. Rio is a member of ACM, IEEE, and SIAM.