Liquid: Mix-and-Match Multiple Image Formats to Balance DNN Training Pipeline

Woohyeon Baek*
baneling100@snu.ac.kr
Seoul National University

Jonghyun Bae*
jbae2@lbl.gov
Lawrence Berkeley National Laboratory

Donghyun Lee
eudh1206@snu.ac.kr
Seoul National University

Hyunwoong Bae
hwbae0326@snu.ac.kr
Seoul National University

Yeonhong Park
ilil96@snu.ac.kr
Seoul National University

Jae W. Lee
jaewlee@snu.ac.kr
Seoul National University

ABSTRACT

Today's deep neural network (DNN) training pipeline utilizes hardware resources holistically, including host CPUs and storage devices for preprocessing the input data and accelerators like GPUs for computing gradients. As the performance of the accelerator scales rapidly, the frontend data preparation stages are becoming a new performance bottleneck to yield suboptimal training throughput. Since the bottleneck in the pipeline may vary depending on hardware configurations, DNN models, and datasets, overprovisioning hardware resources for data preparation such as CPU cores and disk bandwidth is not a cost-effective solution. Instead, we make a case for leveraging multiple data formats, possibly with opposing characteristics in resource utilization, to balance the training pipeline. This idea is realized by Liquid, a new system for building an efficient training pipeline with multi-format datasets. Our evaluation on three distinct execution environments demonstrates that Liquid achieves up to 3.05× and 1.54× higher data preparation throughput on Cityscapes/CityPersons (PNG) and ImageNet (JPEG) datasets, respectively, over the baseline single-format pipeline. This leads up to 2.02× and 1.25× higher end-to-end geomean training throughput with no accuracy drop.

*Both authors contributed equally to this research. This work was conducted while Jonghyun Bae was with Seoul National University.

APSys '23, August 24–25, 2023, Seoul, Republic of Korea

ACM ISBN 979-8-4007-0305-8/23/08…$15.00
https://doi.org/10.1145/3609510.3609811

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**.

KEYWORDS

DNN training, data preparation, image processing

ACM Reference Format:
Woohyeon Baek, Jonghyun Bae, Donghyun Lee, Hyunwoong Bae, Yeonhong Park, and Jae W. Lee. 2023. Liquid: Mix-and-Match Multiple Image Formats to Balance DNN Training Pipeline. In *14th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '23), August 24–25, 2023, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3609510.3609811

1 INTRODUCTION

Deep neural networks (DNNs) enable a wide variety of visual analytics tasks and systems for their high accuracy. State-of-the-art DNNs are computation-intensive; thus, hardware accelerators such as GPU and TPU are commonly employed to train those models whose input data is often prepared on the host device (i.e., CPU) to form a training pipeline. Specifically, the data preparation process consists of reading a large number of input images from the disk, decoding, and transferring them to the hardware accelerator. This whole process is pipelined to maximize training throughput.

Ideally, the throughput should be limited by GPU as the data preparation pipeline continuously feeds input data to keep the accelerator busy for gradient computation. However, these data preparation often fails to sustain sufficient throughput and causes *data stalls* on the GPU. Recent studies report that even a well-optimized frontend stage can be much slower than the gradient computation stage by up to 54.9× [19] and that 10—70% of idle time on the accelerator is caused by blocking I/O for data preparation [17].

For visual analytics tasks, various image formats are utilized according to their objectives, each of which has different trade-offs in terms of resource utilization. An input image may be compressed using a lossy or lossless encoding

scheme or in a raw format. Due to complex decoding algorithms, the data preparation pipeline for compressed images is often compute-bound. In contrast, the same pipeline for uncompressed images can be I/O-bound as the input image is already decoded to have a large size while taking zero computation for decoding. Since a DNN training system is called upon to handle diverse sets of DNN models and datasets, it is extremely challenging to balance the training pipeline and sustain consistently high throughput over such a myriad of use cases.

The research community has proposed various solutions to this problem including data caching methods [17, 18] and new file systems [27, 23, 3, 8]. However, these proposals are only a half solution at best, as it only addresses I/O bottlenecks. Likewise, custom hardware decoders [16, 7, 12] only address computing bottlenecks.

To this end, we make a case for leveraging multiple image formats. Our premise is that, by finding the right mix of input images encoded with different formats, we can balance the utilization of hardware resources, including CPU, GPU, and disk, hence sustaining high training throughput. Although the idea is simple, it requires answers to the following two research questions. First, *which image formats should be used together for optimal performance*? Second, *what is the optimal ratio of mixture among those image formats*? In addition to answering to these conceptual questions, we need to implement an efficient runtime system that supports training with multi-format dataset.

Thus, we propose Liquid, a system that balances the data preparation pipeline utilizing multiple image formats. Liquid conducts offline profiling to find an optimal mixture ratio and generates a multi-format dataset. Then Liquid runtime prepares input data using this dataset maintaining high CPU and disk utilization. We prototype Liquid on NVIDIA DALI [6], a popular data loading library, and evaluate it using a wide variety of vision tasks on three different execution environments. According to our experiments, Liquid improves the data preparation throughput by 2.24–3.04× for PNG-encoded datasets and 5–54% for JPEG-encoded ImageNet which leads to an increase of geomean training throughput by up to 2.02× and 1.25×, respectively.

2 BACKGROUND AND MOTIVATION

2.1 DNN Training Pipeline

The DNN training pipeline consists of three stages: (1) `Load` images, (2) `Decode` images, and (3) `Compute` gradients. Figure 1 shows the DNN training process. Input data are first fetched from storage (`Load`) and then decompressed (`Decode`) in host memory. The decoded items are sent to the training devices (e.g., GPU, TPU) for gradient computation (`Compute`). Ideally, the data preparation and the gradient computation steps can be overlapped, as they are typically executed on different hardware devices. However, the recent hardware accelerators have dramatically reduced the `Compute` time, making the data preparation stages a new performance bottleneck of DNN training to cause *data stalls*.

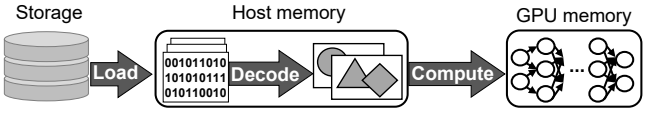

Figure 1: DNN training process

2.2 Characteristics of Image Formats

Visual analytics tasks such as image classification and object detection show specific characteristics of data preparation depending on the image format of the dataset.

The image format can be first divided into two categories based on whether the pixel value is stored in a raw or in an encoded format. The encoded format can be further divided into two sub-categories: lossless and lossy. Figure 2 shows the resource utilization of three representative image formats. We use MobileNetv3 [11] running on a platform with 24-core CPU, 4 NVIDIA RTX 3090 GPUs, and 2TB NVMe SSD. We utilize the JPEG (Lossy)-encoded ImageNet dataset and its converted PNG (Lossless) and BMP (Raw) versions. The utilization triangle is drawn *to scale*. `CPU util` and `Disk BW util` refer to the average CPU time and disk bandwidth utilization. `GPU goodput` refers to the effective GPU utilization for gradient computation. A 100% GPU goodput is the *ideal* training throughput when there is zero overhead for data preparation.

Lossless Image Format. Decompression of lossless image formats is usually a highly sequential operation; thus the DNN training pipeline is often bottlenecked by the `Decode` time. Figure 2(a) presents resource utilization of PNG format, one of the well-known lossless image format, showing that the GPU goodput is only 24.5% (of the ideal throughput) as the CPU utilization is peaked at 100%.

Raw Image Format. Raw image formats (e.g., BMP, PNM) store uncompressed pixel values; thus performance is often bottlenecked by `Load`. In Figure 2(b), the GPU goodput for BMP is only 27.4% as the disk bandwidth utilization is peaked at 100% becasue of a large volume of reads.

Lossy Image Format. In one particular case, `nvJPEG` library can offload some of the CPU-dependent decoding operations of the JPEG format to GPU. Figure 2(c) shows the resource utilization of this case. Both CPU and disk bandwidth are not saturated thanks to the high compression ratio of JPEG and GPU-assisted decoding process, the GPU goodput is only 69.4%. This is due to the resource contention between decoding and gradient computation on GPU.

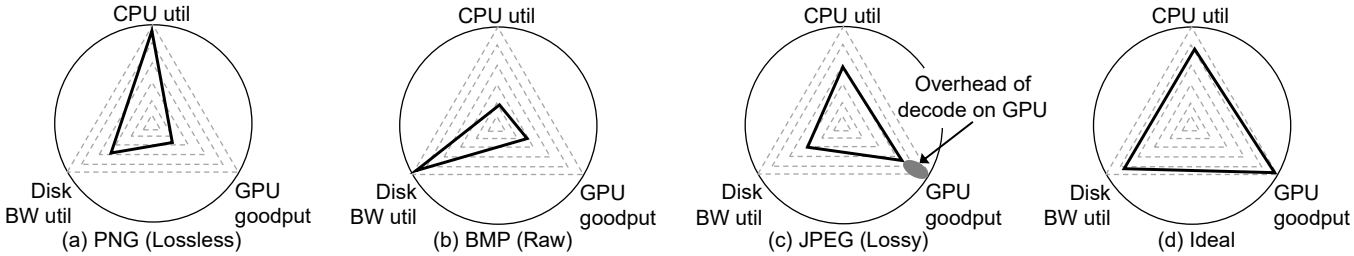

Figure 2: Resource utilization of DNN training on (a) PNG (Lossless) (b) BMP (Raw) (c) JPEG (Lossy) and (d) ideal format

Ideal Image Format. Figure 2(d) shows a hypothetical case of an ideal image format. It should maintain a high resource utilization for both CPU and disk bandwidth to eliminate data stalls and enable the GPU to achieve near 100% GPU goodput without interruption. One way to achieve this is to introduce a novel image format. Instead, we advocate a less intrusive approach of leveraging multiple existing image formats with opposing characteristics.

2.3 Prior Work

Several studies to address data stalls in DNN training propose caching techniques in host DRAM [17, 18] or more efficient data reads [3, 27, 8, 23] to improve the Load throughput. However, due to the uniform access pattern to the dataset (i.e., each image is used just once per epoch), the effectiveness of caching is limited. Furthermore, some proposals require intrusive system modifications like the file system. Still, the Decode stage can cause data stalls. Alternatively, there are proposals to accelerate the decoding process with a novel hardware accelerator [7, 12, 16, 15] and/or a software library [26, 9, 24, 28]. However, to provide robust performance over a variety of hardware configurations, both Load and Decode stages must be jointly optimized.

3 DNN TRAINING WITH MULTI-FORMAT DATASETS

We aim to maximize the data preparation throughput by leveraging multiple image formats. By converting a subset of dataset into a different format, we attempt to reduce the load of the bottlenecked resource at the cost of increasing that of an underutilized resource. For this goal, we should first answer the two research questions: *which image formats should be mixed together?* and *what is the optimal ratio of mixture among the image formats?*

Finding Set of Image Formats to Mix. To get insights into the first question, we investigate the changes in loading and decoding throughput over a spectrum of mixing ratios when a portion of the PNG-encoded Cityscapes dataset is converted into one of the following three formats: BMP (raw), JPEG (lossy) and WebP (lossless). Figure 3(a) shows the results. The tail of each arrow points to the baseline throughput (PNG only), and as we move towards the head of the arrow, the portion of the images encoded with the second format increases (up to 100%).

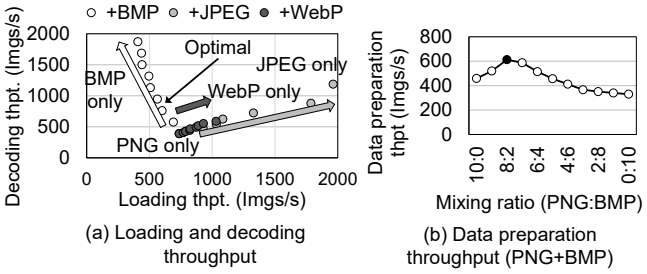

Figure 3: (a) Loading and decoding throughput over a spectrum of mixing ratios between two formats; (b) Data preparation throughput of mixing PNG and BMP with varying the mixing ratio

A takeaway from this experiment is that it is desirable to mix image formats with *opposing characteristics* and find an optimal mix that *balances loading and decoding throughput*. The changes are not significant when mixing PNG and WebP (dark gray dots), as the decoding is the bottleneck for both. When mixed with JPEG (light gray dots), the overall data preparation throughput improves. However, as shown in Figure 2(c), the GPU goodput cannot get close to the ideal throughput due to resource contentions on GPU with JPEG-encoded dataset. Moreover, there might be noticeable drops in model accuracy due to the loss of information while converting PNG images to lossy formats [1]. In contrast, mixing PNG with BMP (white dots) shows a curve heading upper-left. Adding more BMP-formatted images increases the decoding throughput at the cost of loading throughput, which is the direction that we want.

Finding Optimal Mix of Images. The data preparation throughput is determined by the minimum value of loading and decoding throughput. Therefore, the mixing ratio that balances loading and decoding throughput yields the

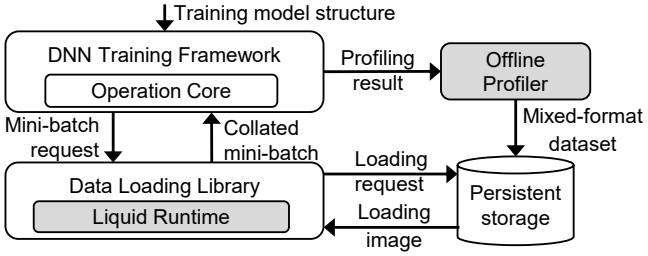

Figure 4: High-level overview of Liquid

best performance. Figure 3(b) shows the data preparation throughput when mixing PNG and BMP. In this case, 8:2 (PNG:BMP) is the best.

4 LIQUID SYSTEM

This section presents Liquid, a system that maximizes data preparation throughput in the DNN training pipeline by balancing loading and decoding throughput with a multi-format dataset. Figure 4 is a high-level overview of Liquid, which consists of two primary components: an offline profiler and a runtime. The offline profiler finds the optimal mixing ratio for a given task and creates a multi-format dataset accordingly. The Liquid runtime builds input data for each mini-batch by drawing images from the multi-format dataset while maintaining the balance between loading and decoding throughput. Section 4.1 and Section 4.2 describe each component of Liquid.

4.1 Offline Profiler

The offline profiler determines the optimal format mixing ratio for a given input dataset and training environment by measuring loading and decoding throughput with different mixing ratios. The offline profiler uses an iterative binary serach for efficiency. Initially, the search space consists of 11 mixing ratios ranging from 10:0 (raw images only) to 0:10 (encoded images only). Starting with the midpoint of the search space, which is 5:5, the offline profiler measures loading and decoding throughput. If the loading throughput is lower (higher) than decoding throughput, the offline profiler discards the right (left) half of the search space. The offline profiler profiles the midpoint of the remaining search space, and this process continues until the search space converges to one value. Once the optimal ratio is determined, the offline profiler generates a multi-format dataset accordingly.

4.2 Runtime System

Basically, Liquid runtime follows the same principle of the existing DNN frameworks [25, 21]. Liquid runtime also maintains an in-memory staging area called shuffle buffer for batch sampling to avoid random disk reads. Instead of directly sampling images from disks, images are sequentially loaded into shuffle buffer from which mini-batch input data are sampled. However, the batch sampling mechanisms of the existing DNN frameworks cannot be directly applied to Liquid runtime for two reasons. First, they do not support multi-format datasets. Liquid runtime should build multi-format mini-batch input data according to the optimal mixing ratio. Second, they rely on OS page cache for inter-epoch data reuse, which may incur non-deterministic fluctuation in the amount of disk accesses across mini-batches. It may break the balance in data preparation pipeline, thus resulting in performance degradation. Liquid runtime should be able to keep loading throughput constant across mini-batches by maintaining constant in-memory cache hit ratio.

To this end, Liquid runtime introduces a new batch sampling mechanism, which is illustrated in Figure 5. To support multi-format datasets, Liquid runtime maintains a separate shuffle buffer for each format. This allows Liquid runtime to sample images based on the mixing ratio. In addition, unlike existing DNN frameworks, Liquid runtime bypasses page cache and uses all available memory space as shuffle buffers, which also serve as a user-level in-memory cache. Then to maintain a constant in-memory cache hit ratio, each shuffle buffer is divided into two regions: current and next epoch. Current epoch region is where the images are sampled. Once images are sampled at each mini-batch, instead of evicting them and fetching new ones from the disk as in existing DNN frameworks, Liquid runtime selectively moves a part of the sampled images to the next epoch region for reuse in the next epoch. For each mini-batch, the number of images moved to the next epoch region and the number of images evicted remain constant. This ensures constant loading throughput. As each image in dataset is sampled exactly once per epoch, an achievable in-memory cache hit ratio is the ratio of the shuffle buffer size to the dataset size. To consistently achieve this ratio, Liquid runtime randomly selects images to move to the next epoch region as many as this ratio from the sampled images at each mini-batch and evicts the rest. The next epoch region starts empty at the beginning of each epoch and grows steadily to reach the size of the shuffle buffer by the end of the epoch. In next epoch, the next epoch region in the current epoch becomes the new current epoch region. This process repeats throughout the training.

5 EVALUATION

5.1 Methodology

Implementation. We prototype Liquid in C++ with approximately 500 lines of code by extending NVIDIA DALI [6].
Comparison Baselines. We compare Liquid with the following two designs: DALI and CoorDL [17]. DALI refers

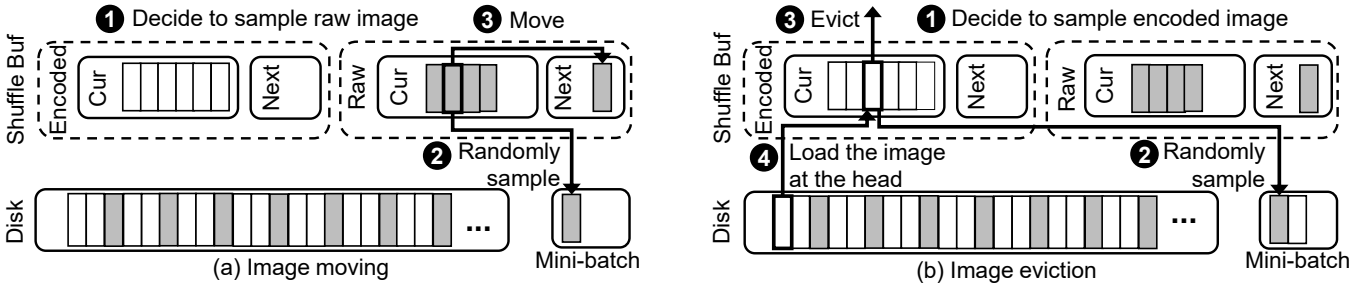

Figure 5: Illustration of runtime operations of Liquid. Once an image is sampled from the current epoch regions of the shuffle buffers, the image is selectively moved to the next epoch regions (a) or evicted and replaced by the new one from the disk (b).

Table 1: System configurations

	T4	V100	RTX3090
CPU	Intel Xeon 8 cores@2GHz	Intel Xeon 8 cores@2GHz	2×AMD EPYC 7452 32 cores@2.35GHz
DRAM	104GB	104GB	448GB
SSD	NVMe 4×375GB	NVMe 4×375GB	NVMe 2×1.8TB
GPU	4×T4 (16GB)	4×V100 (16GB)	4×RTX3090 (24GB)

Table 2: Details about models, datasets, and batch sizes

Dataset (Format)	Model(Abbr.)	Batch size / GPU		
		T4	V100	RTX3090
Cityscapes (PNG) [5]	DeepLabv3+(DL) [2]	32	32	56
	PointRend(PR) [14]	16	16	24
CityPersons (PNG) [30]	YOLOv5(YL) [29]	96	96	160
ImageNet (JPEG) [22]	MobileNetv3(MN) [11]	768	768	1280
	ResNet-18(RN) [10]	512	512	768
	ShuffleNetv2(SN) [31]	768	768	1024
	SqueezeNetv1.1(SQ) [13]	512	512	768

to vanilla DALI with the file reader. CoorDL caches images randomly as they are fetched from storage at the first epoch. These items are never evicted. We use the public opened version on Github [4].

System Configurations. We conduct experiments in three different execution environments. The platform setups are summarized in Table 1. All SSDs are configured to RAID 0. The same versions of software packages are used for all three: Python 3.7, PyTorch 1.12 [20], and NVIDIA DALI 1.15.

Models and Datasets. Among various DNN models and datasets, we choose two semantic segmentation models with Cityscapes [5], one object detection model with CityPersons [30], and four image classification models with ImageNet [22]. Table 2 summarizes the target models, datasets, and batch size. We use the default values suggested in the original paper or implementations for the hyperparameters not shown in the table.

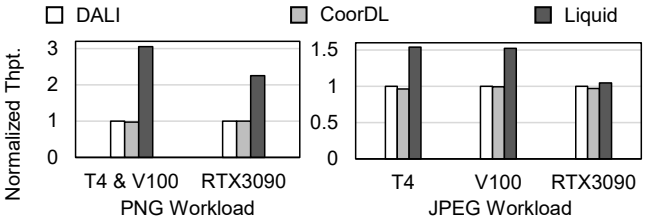

Figure 6: Normalized PNG-encoded and JPEG-encoded data preparation throughput comparison across various training environments

Training Environment. For fair evaluation, we control the available DRAM capacity using `cgroup` to be the same in all cases. DALI uses the same total DRAM capacity for the OS page cache. CoorDL uses 2GB for the OS page cache, and the remaining is allocated for the user-level cache. Finally, Liquid uses the entire DRAM capacity for the user-level shuffle buffer.

5.2 Results

Data Preparation Throughput. Figure 6 shows the PNG-encoded and JPEG-encoded data preparation throughput of DALI, CoorDL, and Liquid on three execution environments normalized by the baseline. We utilize the PNG-encoded Cityscapes with 30.3GB (30% of original dataset) and JPEG-encoded ImageNet with 41.7GB (30% of original dataset) for shuffle buffer. Note that T4 and V100 of PNG workload show the same results by their identical CPU and SSD configurations, as the data preparation stages do not utilize GPUs.

Compared to DALI, Liquid improves the PNG-encoded data preparation throughput by 3.05× on T4 and V100, and 2.25× on RTX3090. The optimal mixing ratio (PNG:BMP) found in each environment is 3:7 and 7:3, respectively. T4 and V100 supports high read bandwidth (680K IOPS) with

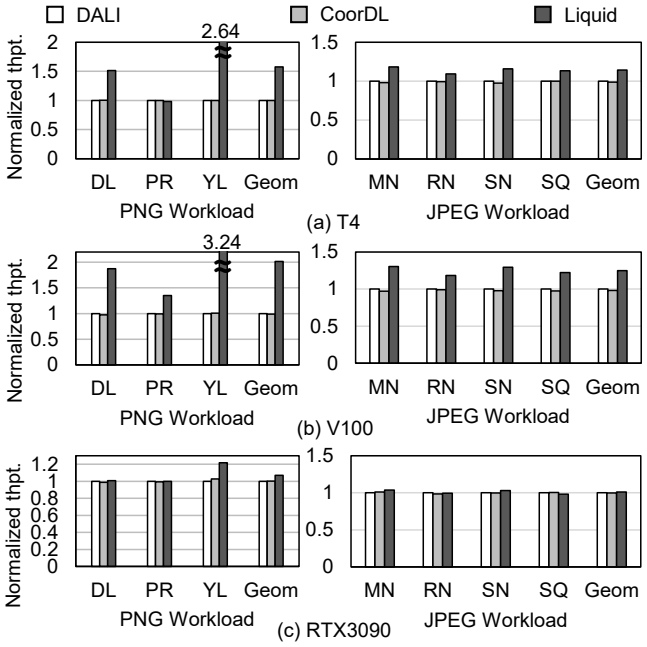

Figure 7: End-to-end training throughput comparison across various models with PNG-encoded and JPEG-encoded dataset (a) T4 (b) V100 (c) RTX3090

RAID 0 on four SSDs, while only eight vCPU cores are available. Therefore, Liquid selects the optimal mixing ratio that BMP has a higher ratio than PNG, to reduce the CPU's load at the cost of an increase in disk reads. At the same time, both DALI and CoorDL are bottlenecked by the decode stage. On the other hand, the RTX3090 can achieve higher decode throughput than T4 and V100 because of the sufficient number of CPU cores. Therefore, Liquid finds an optimal ratio with more PNG than BMP images.

Unlike the lossless image dataset, JPEG decode stage can be processed by both CPU and GPU. Moreover, JPEG often has no I/O bottlenecks due to its high compression ratio. Thus, the runtime profiler of Liquid finds an optimal mixing ratio highly skewed towards JPEG and generates a multi-format dataset consisting mostly of JPEG images. T4 and V100, which have lower FLOPS, achieve 54% and 52% preparation throughput improvement with the ratio of 6:4 (JPEG:BMP) for both. However, RTX3090, having higher compute capabilities, shows only marginal performance gains at the optimal ratio of 9:1 (JPEG:BMP) over the baselines.

DNN Training Throughput. Figure 7 presents the normalized end-to-end training throughput of different data preparation systems with various models on three execution environments. The throughput is normalized to DALI. The mixing ratio that yields the highest data preparation throughput on each training environment in Figure 6 is used.

Overall, Liquid improves end-to-end geomean training throughput with PNG-encoded dataset by 1.58×, 2.02×, and 1.07× over DALI on T4, V100, and, RTX3090, respectively. The throughput gains are most pronounced for the models with a relatively short gradient computing time, such as YOLOv5 (YL) for which Liquid achieves 2.64×, 3.24×, and 1.22× speedups on T4, V100, and RTX3090. In contrast, PointRend (PR) shows the lowest throughput gains with 1.35× at maximum (on V100) due to the long gradient compute time, providing more headroom to hide the data preparation time. CoorDL stores a part of dataset images in memory, but without eviction from/promotion to the buffer. Also, unlike Liquid, it does not support format-aware batch sampling nor consistent cache hit, to cause a pipeline imbalance.

Liquid improves the geomean end-to-end training throughput with JPEG-encoded dataset by 14% and 25% on T4 and V100 over DALI, but has marginal impact on RTX3090. In summary, the performance gains of Liquid are marginal for lossy image formats on high-performance GPUs to which part of the decode operation can be offloaded. However, it still improves performance substantially for relatively low-performance GPUs such as T4 and V100.

6 CONCLUSION

The DNN training process adopts a pipelined structure with three distinct stages, `Load`, `Decode` and `Compute`. Thus, the end-to-end training throughput is eventually determined by the slowest of the three. As the the performance of the accelerator like GPU scales more rapidly than the host CPU or storage device, data stalls in the frontend `Load` and `Decode` stages are becoming a more serious concern these days. Our premise is that, by finding a right mix of input images encoded with different formats, we can balance the utilization of hardware resources including CPU, GPU, and the disk, hence sustaining high training throughput on a variety of execution environments with different hardware configurations. Liquid realizes this premise with format-aware runtime batch sampling, hence boosting data preparation throughput substantially. Also, Liquid flexibly supports various execution environments without requiring intrusive hardware/software changes, to deliver robust training performance regardless of the hardware configuration.

ACKNOWLEDGMENTS

This work was supported by SNU-SK Hynix Solution Research Center (S3RC), and Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by the Korea government (MSIT) (20210-00105, Development of Model Compression Framework for Scalable On-device AI Computing on Edge Applications). Jae W. Lee is the corresponding author.

REFERENCES

[1] Jonghyun Bae, Woohyeon Baek, Tae Jun Ham, and Jae W. Lee. 2022. L3: accelerator-friendly lossless image format for high-resolution, high-throughput dnn training. In *European Conference on Computer Vision (ECCV 22)*. Springer. European Computer Vision Association, Tel-Aviv, Israel, (Oct. 2022), 171–188.

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European Conference on Computer Vision*. Springer, Munich, Germany, (Sept. 2018), 18 pages.

[3] Fahim Chowdhury, Yue Zhu, Todd Heer, Saul Paredes, Adam Moody, Robin Goldstone, Kathryn Mohror, and Weikuan Yu. 2019. I/O characterization and performance evaluation of BeeGFS for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing* Article 80. Association for Computing Machinery, New York, NY, USA, 10 pages. ISBN: 9781450362955.

[4] Project Fiddle. 2022. Coordinated data loader. https://github.com/msr-fiddle/CoorDL. (2022).

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Las Vegas, NV, USA, (June 2016), 11 pages.

[6] NVIDIA. 2021. The NVIDIA data loading library (DALI). https://github.com/NVIDIA/DALI. (2021).

[7] Antonin Descampe, Franois-Olivier Devaux, Gal Rouvroy, Jean-Didier Legat, Jean-Jacques Quisquater, and Benot Macq. 2006. A flexible hardware JPEG 2000 decoder for digital cinema. *IEEE Transactions on Circuits and Systems for Video Technology*, 16, 11, 1397–1410. DOI: 10.1109/TCSVT.2006.884573.

[8] Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. 2021. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* Article 92. Association for Computing Machinery, New York, NY, USA, 15 pages. ISBN: 9781450384421. DOI: 10.1145/3458817.3476181.

[9] Shunji Funasaka, Koji Nakano, and Yasuaki Ito. 2017. Adaptive lossless data compression method optimized for GPU decompression. *Concurrency and Computation: Practice and Experience*, 29, 24, e4283. e4283 cpe.4283.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385. (2015).

[11] Andrew Howard et al. 2019. Searching for mobilenetv3. arXiv preprint arXiv:1905.02244. (2019).

[12] Shizhen Huang and Tianyi Zheng. 2008. Hardware design for accelerating PNG decode. In *Proceedings of the 2008 IEEE International Conference on Electron Devices and Solid-State Circuits*. IEEE, Hong Kong, China, 1–4. DOI: 10.1109/EDSSC.2008.4760652.

[13] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. Squeezenet: alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. arXiv preprint arXiv:1602.07360. (2016).

[14] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. 2020. Pointrend: image segmentation as rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Virtual, (June 2020), 10 pages.

[15] Morgan Ledwon, Bruce F. Cockburn, and Jie Han. 2019. Design and evaluation of an FPGA-based hardware accelerator for deflate data decompression. In *Proceedings of the 2019 IEEE Canadian Conference of Electrical and Computer Engineering*. IEEE, Edmonton, AB, Canada, 1–6. DOI: 10.1109/CCECE.2019.8861851.

[16] Mihir Mody, Vipul Paladiya, and Kapil Ahuja. 2013. Efficient progressive JPEG decoder using JPEG baseline hardware. In *Proceedings of the 2013 IEEE Second International Conference on Image Information Processing*. IEEE, Shimla, India, 369–372. DOI: 10.1109/ICIIP.2013.6707617.

[17] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2021. Analyzing and mitigating data stalls in DNN training. *Proceedings of the VLDB Endowment*, 14, 5, (Jan. 2021), 771–784.

[18] Derek G. Murray, Jiri Simsa, Ana Klimovic, and Ihor Indyk. 2021. Tf.data: a machine learning data processing framework. *Proceedings of the VLDB Endowment*, 14, 12, (Aug. 2021), 2945–2958.

[19] Pyeongsu Park, Heetaek Jeong, and Jangwoo Kim. 2020. TrainBox: an extreme-scale neural network training server architecture by systematically balancing operations. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, Athens, Greece, 825–838. DOI: 10.1109/MICRO50266.2020.00072.

[20] PyTorch. 2021. PyTorch. https://pytorch.org. (2021).

[21] Apache Mesos. 2022. RecordIO Data Format. https://mesos.apache.org/documentation/latest/recordio. (2022).

[22] Olga Russakovsky et al. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115, 3, 211–252. DOI: 10.1007/s11263-015-0816-y.

[23] Frederic Schimmelpfennig, Marc-André Vef, Reza Salkhordeh, Alberto Miranda, Ramon Nou, and André Brinkmann. 2021. Streamlining distributed deep learning I/O with ad hoc file systems. In *Proceedings of the 2021 IEEE International Conference on Cluster Computing*. IEEE, Portland, OR, USA, 169–180. DOI: 10.1109/Cluster48925.2021.00062.

[24] Evangelia Sitaridi, Rene Mueller, Tim Kaldewey, Guy Lohman, and Kenneth A. Ross. 2016. Massively-parallel lossless data decompression. In *Procceedings of the 2016 45th International Conference on Parallel Processing*. IEEE, Philadelphia, PA, USA, 242–247. DOI: 10.1109/ICPP.2016.35.

[25] TensorFlow. 2022. TFRecord and tf.train.Example. https://www.tensorflow.org/tutorials/load_data/tfrecord. (2022).

[26] Jiannan Tian et al. 2020. Cusz: an efficient gpu-based error-bounded lossy compression framework for scientific data. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. Association for Computing Machinery, New York, NY, USA, 3–15. ISBN: 9781450380751.

[27] Lipeng Wang, Songgao Ye, Baichen Yang, Youyou Lu, Hequan Zhang, Shengen Yan, and Qiong Luo. 2020. DIESEL: a dataset-based distributed storage and caching system for large-scale deep learning training. In *Proceedings of the 49th International Conference on Parallel Processing* Article 20. Association for Computing Machinery, New York, NY, USA, 11 pages. ISBN: 9781450388160.

[28] André Weißenberger and Bertil Schmidt. 2018. Massively parallel huffman decoding on GPUs. In *Proceedings of the 47th International Conference on Parallel Processing* Article 27. Association for Computing Machinery, New York, NY, USA, 10 pages. ISBN: 9781450365109.

[29] Ultralytics. 2021. Yolov5. https://github.com/ultralytics/yolov5/. (2021).

[30] S. Zhang, R. Benenson, and B. Schiele. 2017. Citypersons: a diverse dataset for pedestrian detection. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Los Alamitos, CA, USA, (July 2017), 4457–4465.

[31] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: an extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Salt Lake City, UT, USA, (June 2018), 6848–6856.