The Art of Losing to Win: Using Lossy Image Compression to Improve Data Loading in Deep Learning Pipelines

Lennart Behme[1] Saravanan Thirumuruganathan[2] Alireza Rezaei Mahdiraji[3*]
Jorge-Arnulfo Quiané-Ruiz[1,4] Volker Markl[1,4]

[1]*Technische Universität Berlin* [2]*Qatar Computing Research Institute* [3]*Glooko GmbH* [4]*DFKI GmbH*

***Abstract*—Training deep learning (DL) models often takes a significant amount of time and is thus typically performed on expensive GPUs to speed up the process. However, data loading has recently been identified as one of the main performance bottlenecks in DL, resulting in GPU under-utilization. Looking forward, the combination of larger datasets and faster GPUs will exacerbate the problem. The data management community has started to address this by proposing data loading optimization techniques, including lossy image compression. While lossy compression is a conceptually promising approach for mitigating data loading bottlenecks in DL, there is only limited understanding of its efficacy in terms of impact on model throughput and accuracy. In this paper, we present an extensive experimental analysis of lossy image compression as a means to improve the performance of neural network training. We find that lossy compression can improve both throughput and accuracy of DL pipelines if resources such as time or storage capacity are limited. Furthermore, the choice of compression quality and codec are important hyperparameters when training deep neural networks.**

***Index Terms*—deep learning, data loading, lossy compression**

I. Introduction

Deep learning (DL) has achieved great success in various domains, such as computer vision [1], natural language processing [2], and speech recognition [3]. However, the training of DL models can take a very long time – days, weeks, or even months! For instance, training one out of almost 900 AlphaStar agents required as much as 44 days using a TPU pod with 32 third-generation TPUs [4]. Achieving state-of-the-art performance requires increasingly larger datasets and models, resulting in even longer training times. The onerous training and optimization of DL models slows down AI innovation.

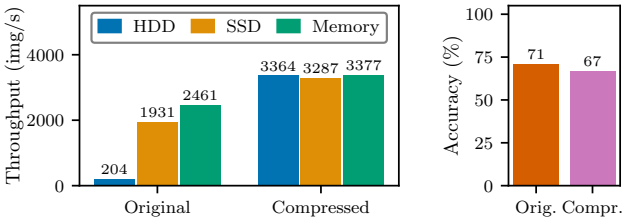
There has been extensive work – including from the data management community [5]–[11] – on improving the resource-intensive nature of deep neural network (DNN) training. Recently, data loading has been identified as a major performance bottleneck in DNN training [9], [12]. Despite an increasing amount of works that investigate the direction of accelerating data loading to speed up DNN training [6], [7], [9]–[15], efficient data loading remains an open problem. To better understand this problem, we briefly outline the end-to-end DNN training process.

DL Training Pipelines. A deep learning pipeline (DLP) consists of three major steps: fetching, preprocessing, and processing. First, data items are fetched from local or remote storage, typically involving random access. Next, the items are preprocessed by applying decoding followed by transformations such as cropping, scaling, or perturbations. Finally, the data are used to train the DNN, which involves a forward pass and a backward pass for updating model parameters. With this primer on DLPs in mind, let us revisit the data loading problem.

Data Stalls in DNN Training. The processing phase of DLPs is compute-intensive and typically performed on accelerators, such as GPUs or TPUs. Modern accelerators are very powerful and even a data bandwidth of 1 GiB/s results in data stalls where the GPU finishes processing a batch of data before the next one is loaded [7]. We subsume all steps from fetching a data item to placing it on a GPU under the term data loading. Data loading bottlenecks can be caused by data fetching (IO), preprocessing (CPU), or network communication. A recent study showed that DNN training time is dominated by data loading [9]. As much as 30% of DNN training is spent on data loading in industrial workloads [16]. For research workloads, this number even goes up to 65% [9]. Going forward, this phenomenon will only get amplified by larger datasets and increasingly more powerful accelerators that outpace advances in storage and general-purpose computing hardware.

Mitigating Data Stalls – The Status Quo. There is a growing number of investigations on data loading optimization techniques to mitigate GPU under-utilization [6], [7], [9]–[15]. We categorize the existing work into domain-agnostic and domain-aware optimization approaches. The majority of contributions focus on domain-agnostic optimization: techniques that are widely applicable but view data items as chunks of bytes and thus do not consider the unique attributes of a data domain. In contrast, domain-aware optimization leverages specific data characteristics. To the best of our knowledge, computer vision in the form of image or video data is the only domain that has been considered for domain-aware optimizations yet. Two recent papers have identified lossy image compression as a

*work done while at DFKI GmbH



(a) Throughput (b) Accuracy

Fig. 1: Impact of lossy image compression on the (a) throughput and (b) learning performance of deep learning pipelines.

technique to mitigate data loading bottlenecks. Intuitively, lossy compression offers promising data loading optimization potential, as it aims to retain the important information of a data item while reducing its file size. A novel data format that applies progressive image compression to minimize the overhead of fetching data during DNN *training* was proposed in [7]. Another work [6] showed that using lossy compression for *inference* can reduce data loading time without reducing accuracy by more than a predefined acceptable level.

Lossy Compression – The Unknowns. Fig. 1a presents an excerpt from our experiment in Fig. 3c and shows that compression can indeed help to mitigate storage-related throughput bottlenecks in an exemplary DLP by fitting the data into memory. Interestingly, the in-memory throughput exhibits that compression also reduces CPU bottlenecks, as smaller image files are decoded faster. This highlights the potential of lossy compression to alleviate data loading bottlenecks in DLPs. Nevertheless, the question of *when and how* lossy compression benefits DLPs is not fully answered yet. Fig. 1b shows ResNet18 trained on two variants of ImageNet and stresses that naively applying compression can drastically reduce the learning performance (see Section V-A for details). Thus, it is crucial to systematically study under which conditions lossy compression is a useful data loading optimization technique.

Outline of Contributions. We empirically investigate lossy image compression as an optimization technique and analyze how it can improve end-to-end DLP throughput without compromising prediction quality. For this, we introduce the experiment sandbox $(DL)^2$ (Deep Learning Data Loading Analysis), which allows us to study both domain-aware and domain-agnostic data loading optimization techniques in a variety of settings. Our contributions are as follows:

(1) We measure the impact of data loading bottlenecks on end-to-end DLP throughput and highlight the importance of addressing the data loading bottleneck problem (Section IV). To this end, we show that moderate image compression can achieve substantial file size and decoding time savings, thus offering a large potential to mitigate data loading bottlenecks.

(2) We study how lossy image compression as a drop-in replacement for the original data impacts the performance of DLPs. Using two popular large computer vision datasets with different characteristics, we show that moderately compressed data still achieve benchmark accuracy but take up less than 30% of the space. We evaluate how compression can improve the training when resources such as training time or storage capacity are limited (Section V).

(3) We analyze if dedicated training regimens using multiple levels of data quality can optimize the trade-off between throughput and learning performance further. We find that using a single appropriate compression quality for the entire training outperforms simple training regimens (Section VI).

(4) We show that domain-aware optimization techniques, such as lossy image compression codecs, are complementary to domain-agnostic optimization techniques, such as the DL-specific software cache MinIO [9] (Section VII).

(5) We investigate the applicability of learned compression codecs in DLPs, which have not yet been thoroughly analyzed for their practicality and runtime performance. We particularly compare the compression factor, encoding time, as well as decoding time of seven learned compression codecs to conventional codecs (Section VIII).

In short, we are the first to find that lossy image compression can improve the training throughput *and* learning performance of DLPs suffering from data loading bottlenecks if the training time or storage capacity is limited. We also found that the choice of image compression codec is a key hyperparameter of many DLPs. We summarize our study with a discussion of lessons learned that addresses both open research problems for scholars and hands-on guidance for practitioners, distilled in an easy to use flowchart (Section IX).

II. Background and Related Work

Lossless versus Lossy Compression. The goal of data (and image) compression is to reduce the size of a given bitstream [17]. This goal creates a trade-off for any compression codec between the compression factor, loss of information, and required computing resources for the encoding and decoding phases. Lossless compression plays an important role in areas where compression artifacts are unacceptable (e.g., medical imaging). However, lossless techniques offer a far more limited trade-off between compression factor and computing resources as they disregard any information loss. Therefore, we focus our work on lossy image compression methods. A new differentiation between perceptual and non-perceptual image reconstruction was introduced in [18]. Conventional methods aim to reduce the perceived loss of image quality to humans. For example, JPEG leverages the fact that the human eye has a lower chromatic visual acuity [19]. Non-perceptual compression techniques reconstruct the image in such a way that downstream machine learning (ML) tasks do not suffer any loss in accuracy. Hence, the encoder is able to remove more information at the cost that the resulting compressed image can only be used for DLPs that are invariant to the compressor in terms of the preprocessing operations it was trained on [18].

Conventional versus Learned Codecs. Image codecs like JPEG, PNG, or WebP were designed by experts who hand-picked algorithm parameters to achieve a good performance. Such codecs could be suboptimal though, wasting either storage capacity or image quality. Recently, learned compression

codecs [18], [20]–[26] have surpassed conventional codecs in terms of their rate-distortion trade-off. While the strong compression performance of learned codecs has been demonstrated, their runtime performance is under-investigated to date.

Domain-agnostic Optimizations. Prior work focuses either on caching or efficient use of preprocessing results. DeepIO [27] and DIESEL [14] propose caching techniques for ML on HPC clusters using RDMA and a storage-caching co-design for distributed file systems. Quiver [13] is a distributed cache that runs on the secondary storage (SSD) of compute nodes while the data are assumed to be located on cloud object storage. The data loading library CoorDL, based on a differential analysis of data stalls in DLPs, was introduced in [9]. CoorDL combines a software cache called MinIO together with coordinated caching across nodes and inter-job coordinated preprocessing. We complement their domain-agnostic insights by studying the new class of domain-aware optimization. Other works on domain-agnostic optimization include [15], [28].

Domain-aware Optimizations. The effects of lossy image compression on DNN classification performance have been investigated in [29]–[35]. However, only two recent publications have started to partially analyze lossy compression as a data loading optimization technique for DLPs. Smol [6] is a DNN inference engine that uses low-resolution image or video data to mitigate preprocessing bottlenecks in *inference* pipelines. The system introduces a new cost model to properly account for preprocessing costs and jointly optimizes the preprocessing and DNN execution stages of a DLP. Kuchnik *et al.* [7] introduce a new data format called Progressive Compressed Record (PCR). PCR leverages progressive JPEG encoding to enable sequential read access to different quality levels of the same data without increasing their size. However, PCR comes with a potentially significant increase in decoding cost depending on the used quality level. Our work complements Smol and PCR by contributing an extensive experimental analysis on how lossy image compression can improve data loading in training DLPs and especially in situations where resources such as time or storage are limited.

DL Benchmarks. There is extensive work on benchmarking DL performance (e.g., [36]–[39]). Beyond that, Mohan *et al.* [9] and Isenko *et al.* [12] as well as projects from the open source community [40], [41] started to investigate and optimize DLP performance from a (domain-agnostic) data loading perspective. Our work continues this exploration with a specific focus on lossy image compression as a means to mitigate data loading bottlenecks via domain-aware optimization techniques.

III. Methodology

We now identify the key guiding hypotheses and outline our experimental setup. The setup is generic and represents realistic and diverse DL workloads, allowing us to obtain meaningful insights about the impact of lossy image compression on DLPs.

Research Hypotheses. Our experimental study was driven by *five* central research hypotheses. Table I shows an overview of our hypotheses and references the corresponding result sections. We begin by independently reproducing and verifying the data loading bottlenecks identified by [9]. Then, we investigate the potential of lossy image compression as a drop-in replacement (i.e., using compressed instead of original data). Comparing DLPs trained on data of different quality, we evaluate peak accuracy and end-to-end pipeline throughput in three different scenarios: (a) training without restrictions, (b) training with a limited time budget, and (c) training with limited storage capacity. Next, we investigate how various training regimens that combine different levels of data quality affect the performance. Supporting our categorization of data loading optimization techniques, we demonstrate that image compression is orthogonal to domain-agnostic techniques [9], [13]–[15]. We also conduct an in-depth comparison of conventional and learned codecs and find that the runtime performance of learned codecs is not competitive for a practical use.

Hardware Setup. We split our analysis across two different servers. The first server $\mathcal{S}_1$ is equipped with two Intel Xeon Gold 5115 CPUs (40 vCores), 192 GiB main memory, an NVIDIA V100 (16 GB) GPU, and runs Ubuntu 16.04 with CUDA 10.2. It has locally mounted hard disk[1] and solid state drives. $\mathcal{S}_1$ represents an average ML server that is within the range of what most DL practitioners have at their disposal. Thus, we used $\mathcal{S}_1$ for all experiments in which we analyze runtime bottlenecks of our DLPs unless otherwise noted. Specifically, Figures 1a, 3, 6, and 10 as well as Table III were done using $\mathcal{S}_1$. We also used a state-of-the-art ML server $\mathcal{S}_2$ to speed up long-running experiments in which we do not directly benchmark the runtime. Specifically, Figures 1b, 2, 4, 5, 7, 8, and 9 as well as Table IV were done using $\mathcal{S}_2$. $\mathcal{S}_2$ has two AMD EPYC 7742 CPUs (256 vCores), 2 TiB main memory, three NVIDIA A100 (40 GB) GPUs, an SAS hybrid storage array, and runs Ubuntu 20.04 with CUDA 11.1.

$(DL)^2$ Experiment Sandbox. We built $(DL)^2$, a Deep Learning Data Loading Analysis experiment sandbox, to analyze data loading in DLPs. Our sandbox is flexible enough to reproduce all the optimization techniques we cover in this paper and in our work beyond. To implement a flexible and powerful sandbox for data loading analysis, we took an existing code base [42] and extensively refactored as well as extended it. $(DL)^2$ aims at addressing two main problems of previously proposed optimization techniques. First, prior works are difficult to compare with each other because they are implemented in different languages and frameworks. Such a general comparability of techniques is crucial as it allows us to evaluate the effectiveness of different optimization techniques and not only their engineering prowess. Second, $(DL)^2$ encourages researchers to use standard DL framework interfaces instead of customized solutions for new proposals. Solutions that rely on highly-customized software are arguably harder to adopt. Thus, our sandbox encourages contributions that are easy to integrate for others, thereby benefiting the entire community.

[1] Although HDDs are known to not match state-of-the-art GPU performance, terabyte-sized datasets are, nevertheless, often stored on HDDs or cloud storage buckets with limited I/O bandwidth for economical reasons.

TABLE I: Research hypotheses overview.

H1	Fetch as well as image decoding bottlenecks throttle the throughput of DLPs and image compression helps to mitigate them.	§IV
H2	Compressed images can act as a drop-in replacement for the original data to optimize DLP performance.	§V
H3	Dedicated training regimens that combine different levels of image quality can help to optimize the trade-off between pipeline throughput and learning performance.	§VI
H4	Lossy image compression as a domain-aware optimization technique can be combined with domain-agnostic approaches to achieve better DLP throughput.	§VII
H5	Learned compression codecs achieve better compression factors but currently achieve too little (de-)compression throughput to replace conventional image codecs.	§VIII

$(DL)^2$ is written in Python 3.8, using PyTorch 1.9 [43] as the core DL framework. Next to the native PyTorch data loader, we employed DALI 1.5 [44] to implement additional data loaders for our DLPs. For the analysis of learned codecs, we used CompressAI 1.1.8 [45]. The set of experiments presented in this paper uses DNN architecture implementations from the Torchvision model zoo. However, $(DL)^2$ generally supports arbitrary extensions to PyTorch's `Module` class. We also designed $(DL)^2$ to be extensible with new implementations of the PyTorch `DataLoader` and `Dataset` interfaces or custom data augmentation operators. These interfaces offer the flexibility to analyze diverse data loading optimizations.

Datasets. We conducted the core of our experiments on ImageNet [46]. In addition, we used Places365 [47] to cover a broader range of possible image data distributions and extend the generalizability of our findings. There are three reasons why Places365 is an interesting addition to ImageNet. (1) It is large enough (110 GiB) to not fit into every reasonable amount of main memory. (2) It is a dataset for scene recognition, meaning that an ML model has to learn rather abstract image categories such as "alley" or "beach", opposed to comparably specific categories from ImageNet (e.g., "lion" or "ant"). Therefore, using Places365 allows us to analyze if lossy image compression has a different effect on model performance depending on whether the data show more specific objects or abstract concepts. (3) All images of Places365 have a compression quality of 75 on a scale from 1 (low) to 100 (high) while the ImageNet samples are compressed at different but generally high quality levels (median quality level 96). This difference enables us to investigate how strongly the impact of compression depends on the original image quality. Beyond ImageNet and Places365, using $(DL)^2$ allows to benchmark additional datasets with little human effort (while the compute time investment depends on the dataset size).

We re-compress our two benchmark datasets at five different quality factors (QFs) using JPEG and WebP, which gives us a total of eleven different variants of each dataset to analyze the impact of lossy compression on DLP performance (see Table II). Next to JPEG, WebP is the second-most common lossy image compression codec, well-supported by DL software stacks, and growing in popularity. Next-generation image codecs, such as AVIF, are an interesting direction of future work but currently out of scope due to the missing support in image processing libraries like OpenCV [48]. For ImageNet, we choose the QFs 85, 75, 50, 25, and 10. We select QF 85 as our first QF because it is the recommended quality for web sites by Google [49].

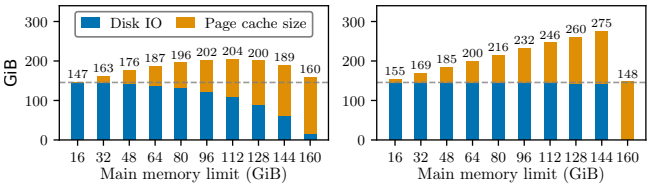
TABLE II: Overview of dataset variants and their size.

	Total file size of train and validation set (GiB)			
	ImageNet [46]		Places365 [47]	
Quality factor	JPEG	WebP	JPEG	WebP
Original	145.7	-	109.2	-
85	67.5	56.2	-	-
75	51.3	38.0	-	-
50	35.3	28.4	87.4	57.2
25	23.6	19.3	52.6	38.6
10	14.5	13.6	30.4	28.0
5	-	-	21.8	22.2
1	-	-	17.3	17.1

The other four QFs are then distributed across the rest of the JPEG quality spectrum $(1-100)$. As Places365 has a fixed compression quality of 75, QFs larger than 74 do not have a file size reducing effect on the dataset. Therefore, we exclude QF 85 and QF 75 while extending our compressed variants of Places365 with QF 5 and QF 1. This has the added side benefit that we can investigate if QFs smaller than 10 indeed have a detrimental impact on learning performance as Dodge and Karam [31] indicate. Note that we do not consider the impact of encoding time outside of Section VIII because (1) encoding is a one-time effort opposed to repeated decoding in each epoch and (2) its overall impact is negligible – compressing ImageNet with JPEG QF 85 only takes 116 seconds on $\mathcal{S}_2$.

Resource Control. We chose our benchmark datasets with the goal of being reasonably large to cause data loading bottlenecks while still being trainable in a feasible amount of time. As both datasets nevertheless fit into the main memory of our servers, we limit the amount of available memory with a cgroup in some of our experiments to simulate larger-than-memory datasets. To understand the effects of the memory limit on the amount of IO per epoch (i.e., the severity of a fetch bottleneck), we trained ResNet50 on ImageNet with multiple memory limits and measured the amount of IO per epoch.

Fig. 2 shows a thrashing analysis for the native PyTorch data loader and DALI. We used the Linux kernel interface to obtain page cache statistics and subtract 1 GiB from the measured size to account for pages not related to our experiments. We observe that although DALI is more memory-efficient than PyTorch and could hence make better use of the page cache, it suffers from pathological thrashing. DALI sequentially reads the dataset in each epoch and then uses an in-memory shuffle buffer to randomize data ingestion. This approach is at odds with the LRU-based Linux page cache though. Therefore, DALI



(a) PyTorch (b) DALI

Fig. 2: Average amount of IO per epoch and page cache size of ResNet50 trained on ImageNet with different memory limits. The dashed line indicates the dataset size (145.7 GiB).

does not benefit at all from an increased main memory limit before the entire dataset fits into memory. PyTorch, on the other hand, randomly accesses data on disk and thus benefits from an increased cache. However, a larger cache does not result in an equivalent reduction in disk IO due to thrashing. Even though DALI has a worse cache usage than PyTorch, we use DALI as our default data loader since it still showed a better overall performance in preliminary experiments. Based on this decision, Fig. 2 shows that the choice of memory limit is negligible as long as it is smaller than the size of the dataset. Consequently, we chose the power of 2 that is closest to 50% of the dataset size (64 GiB) as our main memory limit.

Deep Learning Pipelines. We use three well-studied computer vision DNN architectures in our analysis: ResNet50 [1], ResNet18, and AlexNet [50]. Our goal is to cover a wide range of model complexity while keeping the time to train a model feasible. Conceptually (and supported by preliminary experiments), our findings extend to more recent architectures, such as EfficientNet [51] or MobileNet [52], that are based on the same DNN building blocks. Vision Transformers are a new type of DNN architectures but out of scope for this analysis since their (currently) low GPU throughput prevents them from data loading bottlenecks. To reduce the hyperparameter space of our analysis, we follow the recommendations for learning hyperparameters and preprocessing from the NVIDIA Deep Learning Examples repository [42]. We use throughput as well as accuracy to measure DLP performance.

Reproducibility. Our code, experiment logs, and reproducibility instructions are available at https://github.com/lbhm/dl2.

IV. Data Loading Bottlenecks

Let us start the result discussion with a performance evaluation of end-to-end DLPs to illustrate data loading bottlenecks (Section IV-A). Afterwards, we show the potential performance improvement that compression can achieve (Section IV-B).

A. Illustrating the Problem

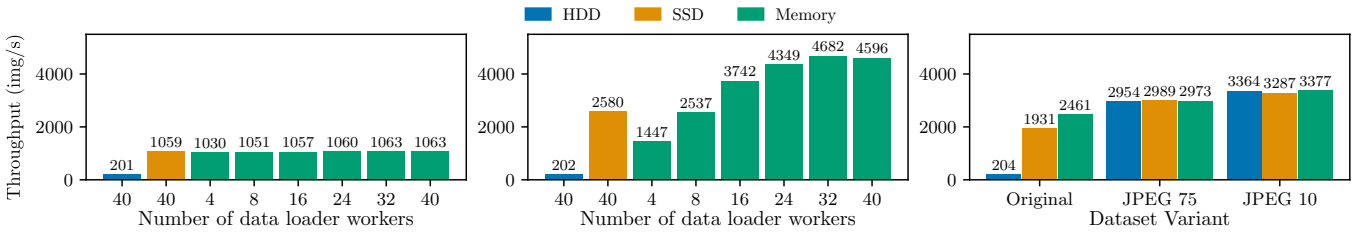
We begin by quantifying the impact of both IO- and CPU-related bottlenecks on end-to-end DLP throughput. We apply the differential analysis approach from [9] to measure the influence of storage type, number of data loader workers (i.e., CPU resources), and image file size on end-to-end throughput. The idea of differential analysis is to vary individual parameters between experiments so that a comparative analysis reveals their impact on a complex system.

Fig. 3 shows the throughput of three different pipelines averaged over five epochs (excluding warm-up). The first two subfigures highlight that storing data on HDD, even if about half of the data are cached in memory, has a detrimental effect on the throughput of ResNet50 and AlexNet pipelines. Comparing the throughput between HDD and SSD demonstrates that the two architectures suffer a $5\times$ and $13\times$ slowdown on HDD, respectively. Our third benchmark architecture ResNet18 exhibits the same behavior but is not depicted for brevity. Even though HDDs are still used to store large amounts of data, their throughput is too low for modern data processing tasks. More importantly, Fig. 3b shows that modern SSDs and high-end CPUs also throttle end-to-end DLP throughput if DNN throughput is high enough. While ResNet50 with data on SSD or in memory and any number of workers is bottlenecked by the GPU, AlexNet is bottlenecked both by the SSD and by the CPU. To quantify the impact, we conducted an additional experiment in which we locked a batch of preprocessed images in GPU memory to eliminate any data loading overhead. This "synthetic" data loader (excluded from Fig. 3 to not distort the scale) demonstrates that AlexNet can process up to 8500 images per second if not throttled by data loading, which is $2\times$ faster than the fastest real DLP in our setup. Although AlexNet is a simple architecture for today's DL standards, our finding translates to any more modern DNN architecture if the GPU processing power (i.e., DNN throughput) is high enough. For example, MobileNetV3 Small [52] achieves a GPU throughput of 4400 images per second on $\mathcal{S}_1$, surpassing the maximum throughput that the SSD and HDD can sustain.

> Data loading is a key performance bottleneck of DLPs that can be caused by different pipeline components.

B. Compression to the Rescue

We next study if lossy image compression is a viable technique to reduce data loading time in both IO- and CPU-bound scenarios. We implement a standalone data loader that only fetches and decodes the data to prevent masking effects of other DLP operations such as, for example, the GPU bottleneck in Fig. 3a. Fig. 3c shows the average throughput of our data loader when it has to process three variants of ImageNet compressed at different JPEG quality levels (original, QF 75, QF 10). Using QF 75 compression increases the throughput by 20% in memory, 56% on SSD, and 1348% on HDD as the data now completely fit into main memory. The results confirm our hypothesis H1 and demonstrate the two directions how lossy compression improves data loading. (1) Smaller dataset variants avoid fetch stalls by fitting into memory. Whereas the original data suffer from fetch stalls similar to the previous experiments, HDD and SSD exhibit the same throughput like in-memory on the other two dataset variants. The exact compression quality level at which the data fit into memory and thus eliminate fetch stalls is dependent on the dataset, amount of main memory, and compression codec. The overall effect generalizes beyond our specific experiment though. (2) Smaller files require less



(a) ResNet50 (b) AlexNet (c) Standalone data loader

Fig. 3: Differential analysis of data loading bottlenecks. (a) and (b) show the throughput of ResNet50 and AlexNet pipelines trained with different CPU and storage resources (higher is better). (c) depicts the data loading throughput of ImageNet compressed at three quality levels when using a standalone data loader (40 workers).

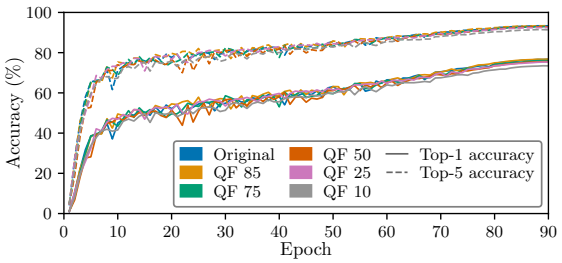


Fig. 4: Validation accuracy of ResNet50 on ImageNet.

time to decode. Comparing the in-memory scenario across all dataset variants highlights that the throughput increases by 20% and 14%, respectively. This performance increase stems from faster decoding times. Previous work has shown that image decoding is by far the most costly preprocessing operator in image DLPs [6], [9]. Therefore, savings in the decoding step have a large impact on the preprocessing costs.

> Image compression indeed helps to mitigate data loading bottlenecks by reducing the fetch and decoding time of DLPs.

V. Compression as a Drop-In Replacement

We now investigate whether compressed images can be used as a direct replacement of the original data without further DLP modifications. To evaluate the efficacy of lossy image compression, we study three dedicated scenarios: Can compressed data[2] achieve the same peak accuracy as the original data (Section V-A)? Can compressed data outperform the original data if training time is limited (Section V-B)? Can compression beat subsampling if storage limitations do not allow to use the entire original data (Section V-C)?

A. Peak Accuracy

To consider lossy compression as a data loading optimization technique, compressed data must achieve a prediction performance that is as close as possible to the original data. We evaluate the learning performance of our three benchmark DNN architectures on compressed data by training them on eleven ImageNet and Places365 variants over 90 epochs.

[2]For brevity, we use this as an abbreviation for "models trained on compressed data".

All models exhibit a consistent learning progress similar to ResNet50 trained on JPEG variants of ImageNet shown in Fig. 4. Therefore, we only consider the last epochs of training and the peak accuracy in our discussion.

Fig. 5 summarizes the final training progress for all JPEG and two WebP scenarios. The peak accuracy of compressed variants does not deviate from the original data by more than 5 percentage points across all configurations, corroborating earlier findings that convolutional neural networks are resilient to image compression [29]–[31], [33], [35]. Different to the previous work though, we also train and not just validate our models on compressed data, hence further extending the finding from inference to training situations. We can see that the next best compressed variant (QF 85 for ImageNet and QF 50 for Places365) is always within $0.1 - 0.2$ percentage points of the peak top-1 accuracy in any of the scenarios. This is within reasonable bounds given the randomness in DNN training. Furthermore, at least two compressed variants surpass the ResNet50 benchmark accuracy for ImageNet (75.9%, by MLPerf [53]) and Places365 (54.7%, from [47]). This means that for ResNet50, we can reduce the training dataset size by at least 74% without letting the prediction performance drop below the benchmark accuracy.

The peak accuracy of JPEG variants with a QF lower or equal to 10 shows a significant decrease in peak accuracy. For example, ResNet50 on Places365 QF 1 reaches an accuracy of 51.3%, which is four percentage points lower than the accuracy of the original data (55.4%). However, QF 1 represents the "worst possible" JPEG compression and the dataset size is reduced by 85% compared to the original dataset. It is noteworthy that the model nevertheless achieves 93% of the reference learning performance. Interestingly, the accuracy decrease of low-quality data is significantly smaller for WebP variants of Places365. As the better learning performance of WebP over JPEG data is consistent across the three lower quality variants (see Fig. 5e and 5f), this indicates a systematic impact of the compression codec on learning performance. The similar (ImageNet) or even better (Places365) accuracy of WebP variants is a strong argument for the use of WebP in image DLPs as WebP also produces 21% smaller datasets on average (see Table II). However, the advantages of WebP do not come without a cost. When we compare the average throughput of JPEG and WebP DLPs in a CPU-bottlenecked scenario

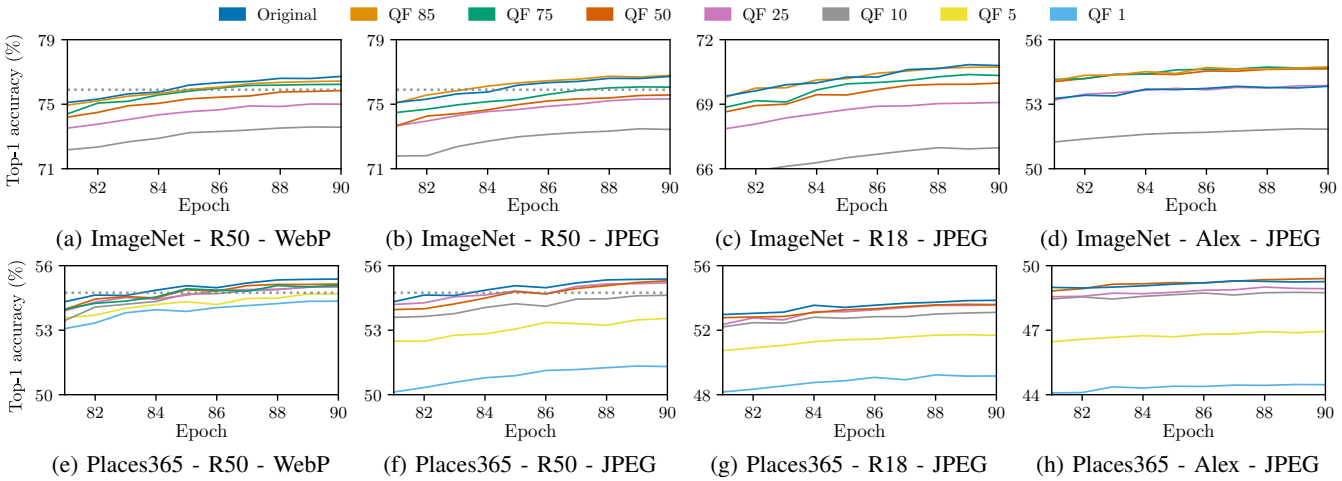


Fig. 5: Validation accuracy of ResNet50, ResNet18, and AlexNet trained on different dataset variants. (a) and (b) additionally show a dotted line with the 75.9% benchmark accuracy for ResNet50 on ImageNet from MLPerf [53]. (e) and (f) show the 54.7% top-1 accuracy benchmark for ResNet50 presented by the Places365 authors [47].

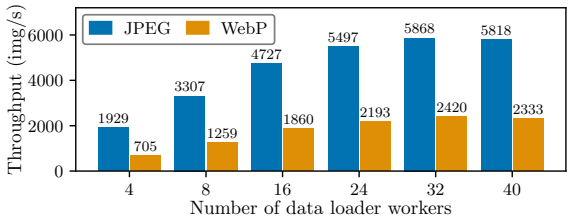


Fig. 6: Throughput comparison of AlexNet on ImageNet compressed with JPEG and WebP (both QF 85).

(Fig. 6), WebP decoding causes 2.6× lower throughput on average as it requires more CPU resources. Thus, WebP offers a different trade-off between compression factor, reconstruction quality, and CPU cycles than JPEG to the user. The choices of compression codec and quality level are important DLP hyperparameters that need to be tuned according to the given hardware setup and desired accuracy-throughput trade-off.

> Moderately compressed data can achieve the same peak accuracy as the original data.

B. Limited Training Time Budget

After analyzing the learning performance of compressed dataset variants without resource restrictions, we next consider the scenario where the training time for a DLP is limited, adding the important dimension of runtime performance. Fast training cycles are critical for situations such as model development, hyperparameter search, or when training on pay-as-you-go infrastructure. Our results in Section IV show that compression can significantly increase the throughput of DLPs with a data loading bottleneck. Thus, we analyze if compressed variants can offset the data quality disadvantage by completing more training epochs than the original data in the same time frame. In preliminary tests, we observed that the epoch times of our DLPs are stable after an initial warm-up epoch. To balance the breadth

TABLE III: Overview of representative epoch times.

Dataset variant	Epoch time (s) ResNet50 (HDD)	ResNet18 (HDD)	AlexNet (HDD)	AlexNet (SSD)
Original	6585	6585	6585	516
JPEG QF 85	2584	2584	2584	265
JPEG QF 75	1251	452	215	214
JPEG QF 50	1250	449	200	201
JPEG QF 25	1248	449	192	190
JPEG QF 10	1248	448	185	186

and runtime (i.e., feasibility) for this set of experiments, we therefore combine our servers $\mathcal{S}_1$ and $\mathcal{S}_2$: First, we determine a representative epoch time computed as a five-epoch average (excluding warm-up) for each combination of JPEG dataset variant and DNN architecture on our benchmark server $\mathcal{S}_1$. Based on that, we calculate how many epochs a given DLP can complete within a defined time budget and then simulate long-running experiments with $\mathcal{S}_2$. This approach allows us to keep the experiment time feasible, as we only need to run six data loading-bottlenecked epochs per configuration on $\mathcal{S}_1$ before leveraging the faster hardware of $\mathcal{S}_2$, while still being able to extrapolate how long an entire experiment would have taken on $\mathcal{S}_1$. Note that apart from a faster execution time, it makes no difference whether a DLP is executed on $\mathcal{S}_1$ or $\mathcal{S}_2$, thus not limiting the generality of our experiments.

Table III summarizes all epoch times we measure on $\mathcal{S}_1$. We compute the representative epoch time of each DNN architecture on HDD as all suffer from fetch bottlenecks in this scenario (see Section IV). For the larger-than-memory dataset variants (original data and QF 85), we average the epoch time by storage type as the DNN architecture does not influence the runtime in these scenarios. Furthermore, we measure the epoch time of AlexNet on SSD as it also suffers from data loading bottlenecks. We can see a clear differentiation in epoch times between in-memory (QF$\leq$ 75) and larger-than-

memory dataset variants (gray rows). On CPU-bottlenecked architectures, the results also exhibit a performance difference between in-memory variants due to the reduced decoding time. Overall, the slowdown from the original data to the fastest compressed variant varies from $3\times$ to $36\times$ (red cells).

We assume that a potential accuracy difference between dataset variants depends on the difference of completed epochs across pipelines. Therefore, we define three general time limits of {5, 10, 20} hours that allow us to investigate different scenarios: from configurations with few training epochs, such as ResNet50 on HDD over 10h (DLPs complete between 5 and 28 epochs), up to long training jobs, such as AlexNet on SSD over 5h (between 34 and 96 epochs). We also define another special time limit for ResNet50, derived from the time it takes the fastest compressed dataset variant to complete twice the default number of training epochs (180). The goal of this 62h time limit is to investigate the effect of compression when the original data can train for a reasonable number of epochs and the compressed variants train longer than usual.

Fig. 7 shows one DLP configuration for each of the time budgets we study. The original data achieve a worse accuracy than compressed variants. Comparing the performance penalty of larger-than-memory datasets, we see that the accuracy difference shrinks with a growing time budget as the bottlenecked pipelines are able to complete more training epochs. Nevertheless, a noticeable accuracy advantage (>3 %) between the original data and the best compressed variant remains. In each scenario, a variant that fits into memory (hence being able to complete more epochs) beats the two higher-quality dataset variants that are larger-than-memory. This is especially interesting in the AlexNet on SSD scenario (Fig. 7a), as the throughput advantage of compressed variants is much lower than on HDD. Here, QF 50 achieves a marginally better performance than QF 75 (54.18% vs. 54.04%), indicating that the performance increase due to reduced decoding time can mitigate worse data quality. Table 7d summarizes the peak accuracy of ResNet50 pipelines trained with a 62h time budget. We see that compressed data not only achieve higher accuracy than the original data when training for short periods but also in long-running experiments. All compressed variants but QF 10 achieve a higher peak accuracy than the original data. JPEG QF 75 (green row) even achieves better peak accuracy than the original data achieved after 90 training epochs (76.72%).

> Compression is a viable technique for achieving better learning performance whenever the training time budget is limited or fast training progress is paramount.

C. Limited Storage Capacity

We now turn our analysis to another potentially scarce resource: storage. Edge computing devices with limited compute and storage resources are becoming increasingly relevant for the training of ML models. For example, federated learning pushes model update computations to participating parties instead of gathering raw data on a central server [54]–[56]. Moreover, our previous set of experiments shows that being able to fit all training data into memory can yield significant runtime and accuracy-over-time improvements. Thus, an analysis of storage limits is an interesting question for a large share of DL practitioners (i.e., those without unlimited main memory).

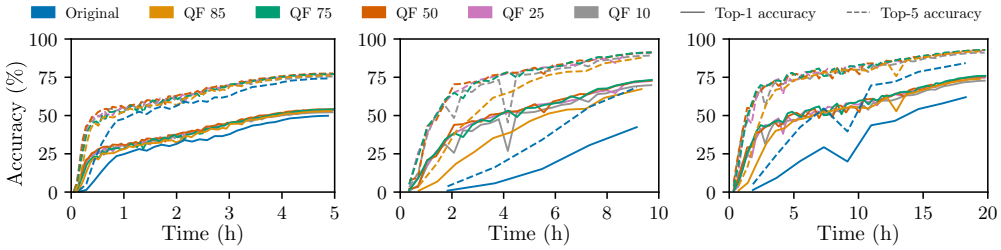
Compression opens up a compelling trade-off between the number of training samples and the quality per sample compared to simple approaches such as subsampling to meet storage limitations. We evaluate this trade-off for each of our architecture-dataset combinations, using five different storage limits. To make the storage limits independent of the datasets we use, we define our limits relative to the original dataset size and consider {50, 40, 30, 20, 10}%. For each storage limit, we compare a subsampled version of the original data to the highest-quality compressed variant that fits into the respective limit (see Table II). We make our subsampled datasets reproducible by selecting the first $x\%$ training images (based on file name as there is no particular ordering) and all validation images per class for a subsample that complies with an $x\%$ storage limit. We train each subsampled and compressed variant over 90 epochs and compare the validation accuracy.

Fig. 8 exhibits the five storage limits for ResNet50 on ImageNet (the other five architecture-dataset combinations show the same pattern). We see that it can be beneficial to give up on data quality if a larger amount of training data can be used in turn. As the storage limit becomes more restrictive, the performance difference between subsampled and compressed data grows. While the accuracy difference between compression and subsampling is 6 percentage points for a 50% storage limit, it grows to 29 points in Fig. 8e as the storage limit tightens. The accuracy difference between models trained on subsampled and compressed data shrinks as the architecture becomes less complex (ResNet50 > ResNet18 > AlexNet). Overall, the experiment highlights the importance of the number of training samples versus quality per sample trade-off and shows how compression can be used to optimize this trade-off. Still, researchers and DLP engineers must assess this trade-off for each individual use case as there is no general rule whether to prioritize quantity or quality of data.

> Compressing instead of subsampling training data when storage capacity is limited yields significantly better prediction quality in all of our experiments.

VI. Training Regimens

So far, we have assumed image compression quality to be fixed for the entire training procedure. We next study if combining multiple dataset variants into a *training regimen* can optimize the accuracy-throughput trade-off further. A training regimen consists of two or more consecutive stages. Each stage uses a different dataset variant and potentially modifies other DLP hyperparameters, such as the learning rate or regularization. The idea of training regimens is implied by a recent work [7], which evaluates a tuning mechanism that can change the data quality of the proposed Progressive

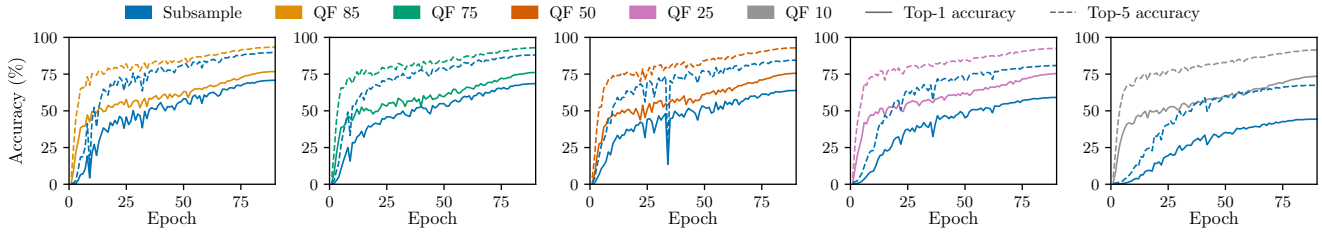


(a) 5h: AlexNet on SSD (b) 10h: ResNet50 on HDD (c) 20h: ResNet50 on HDD

Dataset	Epochs	Acc. (%)
Original	34	74.27
QF 85	86	76.37
QF 75	179	77.50
QF 50	179	76.97
QF 25	180	76.60
QF 10	180	73.81

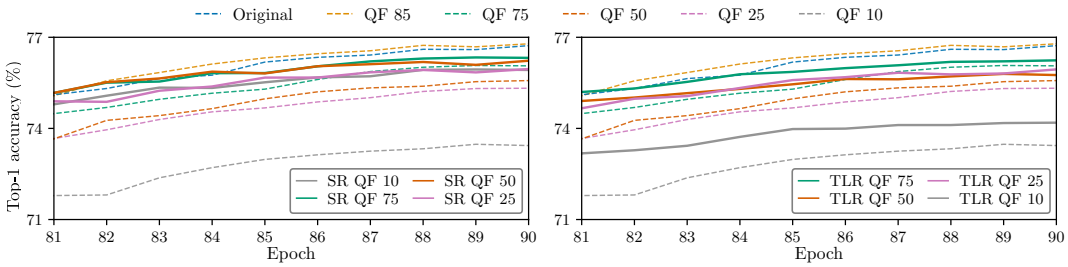
(d) 62h: ResNet50 on HDD

Fig. 7: Accuracy analysis of pipelines trained with different time budgets. Progress finishing before the time limit is due to the model not completing another full epoch. The pipelines with original data completed (a) 34, (b) 5, (c) 10, and (d) 34 epochs.



(a) 50% (b) 40% (c) 30% (d) 20% (e) 10%

Fig. 8: Accuracy of ResNet50 trained on subsampled and compressed data conforming to different storage capacity limits relative to the original dataset size.



(a) Simple regimens (SR) (b) Transfer learning regimens (TLR)

Regimen	Size (GiB)
SR QF 10	80.1
SR QF 25	84.7
SR QF 50	90.5
SR QF 75	98.5
TLR QF 10	29.1
TLR QF 25	37.2
TLR QF 50	47.6
TLR QF 75	61.8

(c) Average dataset size

Fig. 9: Overview of training regimens for ResNet50 on ImageNet. Dashed lines represent baselines from Fig. 5b.

Compressed Record format during training. We investigate the idea of training regimens further and compare them to our baselines from Section V-A. All configurations receive a training budget of 90 epochs, which we then split up into separate stages. We consider two categories of configurations. The first regimen type is called simple regimen (SR), as it equally divides the budget between stages (i.e., 45 epochs per stage in a two-stage configuration). The second type is motivated by the idea of transfer learning [57], thus called transfer learning regimen (TLR). TLR trains a model on lower quality data for 80 epochs during the first stage and then refines it by training on the highest quality data for 10 epochs. Intuitively, a model should learn a basic understanding of the data domain on low-quality samples and then fine-tune its classification performance on high-quality data.

Fig. 9 shows the performance results of the training regimens. Overall, all regimens achieve a peak accuracy that lies within the performance boundaries of the involved dataset variants. For example, SR QF 10 combines JPEG QF 10 with the original data and achieves a peak accuracy of 75.9%, which is better than the result of QF 10 (73.5%) and worse than the original data (76.7%, see Section V-A). To compare training regimens with the underlying dataset variants in terms of their runtime performance, we compute the average dataset size of a regimen based on Table II. The average dataset size is a reasonable proxy for the runtime performance of DLPs suffering from data loading bottlenecks as we showed in Section IV-B. For example, TLR QF 10 that trains on JPEG QF 10 and then switches to the original data has an average dataset size $\mathcal{D}$ of:

$$\mathcal{D}(\text{TLR QF 10}) = \frac{80 * 14.5 \text{ GiB} + 10 * 145.7 \text{ GiB}}{90} = 29.1 \text{ GiB}$$

Comparing the average dataset sizes from Table 9c and their corresponding learning performance to our baselines trained on a single dataset variant challenges the efficacy of our regimens-under-test. The dataset variants that are closest in accuracy to a regimen configuration have a very similar or smaller dataset size than the respective regimen. For instance, all SRs achieve a peak accuracy close to the prediction quality of QF 75 while having a significantly larger average dataset size. Hence, we claim that choosing the right compression quality yields the same or better results than combining multiple dataset variants in one

of the two regimen types that we analyze. Consequently, we have to decline our initial hypothesis about training regimens based on our experimental data. Nevertheless, we still see potential in the idea of training regimens. Combining multiple dataset variants introduces additional hyperparameters, such as the number or the length of stages. Thus, drawing a general conclusion about the efficacy of training regimens requires a more in-depth analysis of regimen types and hyperparameter configurations that is beyond the scope of this paper. For now, our results indicate that choosing the right compression quality up front might give a better throughput-accuracy trade-off than combining multiple quality levels as proposed in [7].

> Training regimens can achieve a trade-off between the accuracy and throughput of involved dataset variants.

VII. Orthogonality Analysis

We next examine if lossy image compression can be combined with domain-agnostic optimization techniques for further throughput improvements. For this, we reimplement the recently proposed MinIO software cache [9] in PyTorch. MinIO prevents thrashing in DLPs by caching a fixed set of training samples without replacement. To assess the orthogonality of MinIO and compression, we measure the average throughput of our three benchmark architectures in four scenarios: (1) without modifications, (2) using MinIO instead of the default page cache based on an LRU policy, (3) using JPEG QF 85 instead of the original data, and (4) combining MinIO with compression. We analyze each architecture with data stored on HDD and additionally evaluate AlexNet with data stored on SSD as it also suffers from data loading bottlenecks on SSD in our setup.

Fig. 10 shows that combining MinIO with compression yields up to a $3\times$ throughput improvement over the second-best scenario that only applies a single optimization technique. Across the models trained on HDD data, ResNet50 exhibits the lowest relative speed-up of $1.4\times$ as the throughput of ResNet50 in the combined scenario already reaches the maximum that our GPU on server $\mathcal{S}_1$ can sustain at about 1050 img/s. For ResNet18, MinIO and compression alone achieve a $2\times$ and $3.7\times$ throughput improvement over the default scenario, respectively. The combined scenario surpasses a linear combination of the individual speed-ups and yields a $10.2\times$ improvement over the default scenario. AlexNet on HDD exhibits a similar throughput pattern across scenarios, supporting our ResNet18 observations. AlexNet on SSD shows a different pattern though. There are two main observations here: the throughput is generally higher and MinIO has no positive effect on throughput. The better performance of SSDs causes the main bottleneck in AlexNet pipelines to shift from fetching to preprocessing. As MinIO only improves the fetch throughput of DLPs, it consequently does not have a positive effect on the end-to-end throughput anymore. Yet, AlexNet on SSD highlights that even if one technique has no positive impact, combining compression with domain-agnostic techniques such as MinIO does not hurt the performance.

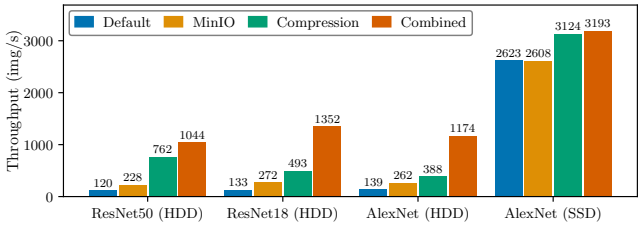


Fig. 10: Orthogonality of MinIO and lossy image compression.

Overall, this experiment confirms our hypothesis that MinIO and compression are orthogonal to each other, meaning that the improvements of one technique do not cannibalize the potential of the other technique. Of course, combining them may not increase end-to-end throughput in every DLP. As both techniques primarily (compression) or only (MinIO) aim to mitigate fetch bottlenecks, pipelines suffering from a large fetch-to-preprocessing throughput gap (e.g., HDD pipelines in our setup) will benefit the most. However, if another component in the DLP is not able to match fetch rate improvements caused by compression and MinIO, potential performance gains are masked when the primary DLP bottleneck shifts (e.g., to CPU). Nevertheless, this experiment substantiates that combining domain-agnostic and domain-aware techniques is key for achieving the optimal end-to-end throughput in DLPs.

> Compression is orthogonal to the MinIO software cache. This finding provides initial evidence for the general interaction between domain-agnostic and domain-aware data loading optimization techniques.

VIII. Learned Compression

Learned image codecs have achieved impressive compression results in the last years. However, they have not been thoroughly analyzed in terms of their practicality and runtime performance yet. Despite the recent attention, most proposed algorithms are either not publicly available or do not have a reusable code base yet. We used the CompressAI library [45], because it offers well-documented and pre-trained models for six previously published learned codecs [21]–[23]. In addition, we included lossyless [18] in our analysis as it follows a new non-perceptual image reconstruction approach (see Section II) and offers pretrained models as well. We consider three distinct metrics: compression factor, encoding time, and decoding time. With each metric, we evaluate three different quality parameters per codec on a benchmark dataset: For JPEG and WebP, we used the quality parameters {10, 50, 85}. For CompressAI, we used the parameters {1, 3, 6} or {1, 4, 8}, depending on how many pre-trained quality levels are available per model. For lossyless, we used the three available pre-trained models with beta values of {0.01, 0.05, 0.1}. In the encoding time analysis, we applied a batch encoding approach by default for learned codecs as we want to study their best-case behavior. To achieve a reasonable runtime for our experiments, we utilized a subsampled version of ImageNet with five classes (670 MiB). Table IV shows an overview of all codecs and their performance.

TABLE IV: Comparison of compression factor, encoding time, and decoding time of different image codecs. The compression factor shows the relative reduction in file size compared to our benchmark dataset (higher is better).

	Compression factor			Encoding time (s)			Decoding time (s)		
Compression quality	low	med.	high	low	med.	high	low	med.	high
JPEG	14×	5×	2×	4	5	4	5	4	4
WebP	16×	7×	3×	6	5	7	3	4	5
bmshj2018-factorized [21]	99×	35×	9×	54	64	122	1392	1727	1597
bmshj2018-hyperprior [21]	101×	37×	10×	93	99	127	1753	1907	2130
mbt2018-mean [23]	100×	36×	10×	95	100	127	1843	1871	2229
mbt2018 [23]	132×	42×	11×	3522	3540	3775	7200+	7200+	7200+
cheng2020-anchor [22]	123×	63×	23×	3516	3481	3516	7200+	7200+	7200+
cheng2020-attn [22]	131×	66×	24×	3494	3656	3564	7200+	7200+	7200+
lossyless [18]	559×	434×	259×	38	42	41	24	27	27

Compression Factor. The first section of Table IV highlights that learned compression codecs achieve significantly higher compression factors than conventional codecs. The CompressAI codecs and lossyless compress our benchmark dataset up to 1 and 2 orders of magnitude better than conventional codecs on the low and high quality setting, respectively (green cells). Batch encoding favors the compression factor of learned codecs though, because large images are scaled and cropped to a uniform size before the encoding step. In contrast, JPEG and WebP encode the entire original image. Thus, we also ran *bmshj2018-factorized* in single image mode without image preprocessing to measure the differences with batch encoding. We observed that compressing only a single image at a time on average takes $13.5\times$ longer than batch encoding across all three quality levels and results in a dataset that is $3.8\times$ larger. This shows that batch compression heavily reduces the processing time and dataset size at the cost of losing information, which could hurt the downstream prediction quality. Nevertheless, the single image encoding datasets are still $\{30\times, 10\times, 2\times\}$ smaller than the benchmark dataset across quality levels. This makes them about $2\times$ more effective than conventional codecs for medium and low image quality. Overall, learned image compression codecs are a promising replacement for conventional codecs from a compression standpoint.

Encoding Time. We next study learned codecs from a runtime perspective. For CompressAI codecs, we evaluate the encoding time on CPU as well as GPU (lossyless encoding is only supported on GPU). Although using GPUs for inference with auto-regressive models is not recommended since the entropy coder is run sequentially on CPU, Table IV reports encoding times on GPU for learned codecs as they are faster in all cases. The encoding time shows a clear performance advantage of conventional codecs. While JPEG and WebP take only 4-7 seconds to encode the dataset (and thus are likely dominated by overhead), learned codecs take up to 3775 seconds! Generally, JPEG is between $10\times$ and $944\times$ faster than learned codecs. Lossyless also is substantially faster than CompressAI codecs but still an order of magnitude slower than JPEG.

Decoding Time. The decoding time is an essential metric for any codec used in a DLP as it is performed every time an image is loaded by a data loader worker. As of now, CompressAI and lossyless do not support batch decoding, requiring every image to be processed separately. Note that we terminate a decoding job if it does not finish within 2 hours as this is more than a $1000\times$ runtime increase over the worst conventional codec runtime. Table IV summarizes that all CompressAI codecs are at least two orders of magnitude slower than JPEG and WebP (red cells), rendering them unusable in a production DLP.

As the runtime performance of lossyless is within one order of magnitude of the conventional codecs, we also encode and decode the entire ImageNet-1K dataset with JPEG, WebP, and lossyless for a more thorough comparison that mitigates the impact of overhead in short-running jobs. On ImageNet-1K, lossyless encoding takes 1663 seconds on average over all three quality levels, which is $15\times$ and $6\times$ slower compared to JPEG and WebP, respectively. This difference gets exacerbated when decoding. Lossyless takes 2801 seconds on average, being $255\times$ and $175\times$ slower than the conventional codecs. Note that a fair comparison of non-perceptual codecs like lossyless and perceptual codecs like JPEG is not trivial, because lossyless only requires a small DNN (i.e., less GPU computation time) for the classification after decoding [18]. On the other hand, we also do not consider the time required to train a learned codec in our evaluation. Thus, our runtime evaluation on a large and highly-used dataset is the best approximation of a performance comparison to date. Due to the missing support for batch decoding, we include lossyless in our verdict that learned codecs are an impractical replacement for conventional codecs at this time. Consequently, we defer an investigation of the achievable accuracy of data compressed with learned codecs to future work until the runtime performance has improved.

Implementation Limits. We identify two software bottlenecks in the DL landscape that need to be addressed: (1) In a typical DLP, the image decoding operator runs inside of a data loader worker. However, some DL frameworks, such as PyTorch, run workers in single-threaded mode. Limiting a DNN-based decoder to a single thread would further increase the performance gap between learned and conventional codecs. (2) Inference engines for DL, such as TensorRT, could reduce the runtime deficiencies of learned codecs. Unfortunately, the variable length input of learned decoders was not supported by TensorRT as of our analysis. Thus, more work towards efficient runtime engines for learned compression codecs is necessary.

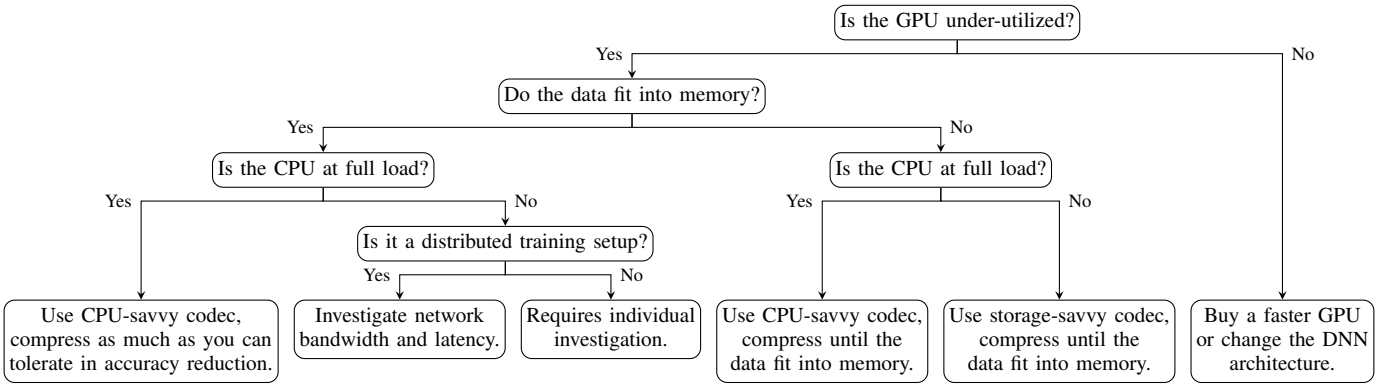


Fig. 11: Flowchart with a summary of our hands-on recommendations for practitioners.

> Learned compression codecs achieve a better compression performance compared to conventional codecs. However, their slow runtime performance currently makes them unfeasible for practical use in DLPs.

IX. Lessons Learned

Our work serves as a motivation for data management (DM) and ML researchers to consider lossy image compression as a data loading optimization technique. We argue that our findings also translate to other domains with lossy compression, such as video or audio. We believe that the DM community is qualified to address data stalls in DLPs by leveraging its existing expertise about data loading in large-scale systems.

Hands-on Guidance for Practitioners. Fig. 11 summarizes our recommendations for practitioners. If the training data do not fit into memory or a DLP requires complex preprocessing, data loading bottlenecks are likely an issue. By choosing the right compression codec and quality level, a DLP engineer can pick the optimal trade-off between prediction quality, dataset size, and CPU load. Thus, we encourage practitioners to take codecs other than JPEG into account for their work. A rule of thumb for the quality level is to use the highest QF that still fits the training data into memory to achieve a good trade-off. Here, lossy compression can be of great use when the training time budget is limited. For example, hyperparameter search strategies, such as successive halving [58], could iteratively use higher quality data in each stage to speed up the search or to execute more epochs in the same time. If there is an excessive amount of training data or storage is scarce, compressing the data instead of discarding (i.e., subsampling) them can be beneficial. In distributed training or data streams, compression also helps to control network traffic and latency. Lastly, one has to verify if the main DLP bottleneck shifts due to compression.

Open Research Problems for DM and ML Scholars.
(DM 1) Data loading optimization for DL and domain-aware optimization techniques in specific are not sufficiently explored yet. There is further potential for domain-aware optimization to keep up with the increasing speed of DL accelerators. For example, progressive image resizing has been studied in the context of GPU computations [59], [60], but could potentially also help to optimize data loading. Our orthogonality analysis shows that domain-aware optimizations are key to boosting existing domain-agnostic techniques. Beyond computer vision, domains such as audio or text remain unattended to date.
(DM 2) Our analysis, similar to related work [6], [7], [9], [12], highlights the significance of data loading for the end-to-end DLP throughput. So far, data loading and model training have been mostly viewed as separate problems with Kang *et al.* [6] starting to jointly optimize data loading and DNN training. Our results underscore this direction and call for more research in the direction of holistic DM for DLPs, such as data ecosystems where data are offered as a service.
(ML 1) We find that our training regimens can have a positive impact on the accuracy-throughput trade-off in some, but too few situations. Therefore, our observations are a call-to-arms for further exploration of training regimens: We need to understand when and how combining multiple variants of a dataset during training can optimize the accuracy-throughput trade-off.
(ML 2) Our preliminary analysis shows that the runtime performance of current learned codecs makes them impractical to use in DLPs. To overcome this gap, ML researchers must design more efficient codecs and systems researchers must design interfaces to integrate learned codec inference into the existing data loading libraries for model training.

X. Conclusion

The research community has identified data loading as a performance bottleneck in DLPs and shown that lossy image compression can help to mitigate it. However, no work has systematically studied under which conditions lossy compression is useful yet. This paper thoroughly investigated the impact of lossy compression on data loading in DLPs. We found that it is a strong lever to control the trade-off between accuracy and throughput for DNN training. Interestingly, compression can even improve learning performance and throughput at the same time when training time or storage capacity is limited.

Acknowledgments

This work was supported by the the German Ministry for Education and Research as BIFOLD (01IS18025A and 01IS18037A). We also thank Bonaventura Del Monte and Clemens Lutz for their helpful feedback.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," presented at CVPR '16, 2016.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, *et al.*, "Attention is all you need," presented at NIPS '17, 2017.

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, *et al.*, "Deep speech 2 : End-to-end speech recognition in english and mandarin," presented at ICML '16, 2016.

[4] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[5] Mijin An, In-Yeong Song, Yong-Ho Song, and Sang-Won Lee, "Avoiding read stalls on flash storage," presented at SIGMOD '22, 2022.

[6] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia, "Jointly optimizing preprocessing and inference for DNN-based visual analytics," *PVLDB*, vol. 14, no. 2, pp. 87–100, 2020.

[7] Michael Kuchnik, George Amvrosiadis, and Virginia Smith, "Progressive compressed records: Taking a byte out of deep learning data," *PVLDB*, vol. 14, no. 11, pp. 2627–2641, 2021.

[8] Arun Kumar, "Automation of data prep, ML, and data science: New cure or snake oil?," presented at SIGMOD '21, 2021.

[9] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram, "Analyzing and mitigating data stalls in DNN training," *PVLDB*, vol. 14, no. 5, pp. 771–784, 2021.

[10] Supun Nakandala, Yuhao Zhang, and Arun Kumar, "Cerebro: A data system for optimized deep learning model selection," *PVLDB*, vol. 13, no. 12, pp. 2159–2173, 2020.

[11] Yuhao Zhang, Frank McQuillan, Nandish Jayaram, *et al.*, "Distributed deep learning on data systems: A comparative analysis of approaches," *PVLDB*, vol. 14, no. 10, pp. 1769–1782, 2021.

[12] Alexander Isenko, Ruben Mayer, Jeffrey Jedele, and Hans-Arno Jacobsen, "Where is my training bottleneck? hidden trade-offs in deep learning preprocessing pipelines," presented at SIGMOD '22, 2022.

[13] Abhishek Vijaya Kumar and Muthian Sivathanu, "Quiver: An informed storage cache for deep learning," presented at FAST '20, 2020.

[14] Lipeng Wang, Songgao Ye, Baichen Yang, *et al.*, "DIESEL: A dataset-based distributed storage and caching system for large-scale deep learning training," presented at ICPP '20, 2020.

[15] Chih-Chieh Yang and Guojing Cong, "Accelerating data loading in deep neural network training," presented at HiPC '19, 2019.

[16] Derek G. Murray, Jiri Simsa, Ana Klimovic, and Ihor Indyk, "Tf.data: A machine learning data processing framework," *arXiv:2101.12127*, 2021.

[17] David Salomon, *A Concise Introduction to Data Compression*. Springer Science & Business Media, 2007.

[18] Yann Dubois, Benjamin Bloem-Reddy, Karen Ullrich, and Chris J. Maddison, "Lossy compression for lossless prediction," presented at Neural Compression Workshop at ICLR '21, 2021.

[19] Stefan Winkler, Murat Kunt, and Christian J. van den Branden Lambrecht, "Vision and video: Models and applications," in *Vision Models and Applications to Image and Video Processing*, Christian J. van den Branden Lambrecht, Ed., Boston, MA: Springer, 2001, pp. 201–229.

[20] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli, "End-to-end optimized image compression," presented at ICLR '17, 2017.

[21] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston, "Variational image compression with a scale hyperprior," presented at ICLR '18, 2018.

[22] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," presented at CVPR '20, 2020.

[23] David Minnen, Johannes Ballé, and George Toderici, "Joint autoregressive and hierarchical priors for learned image compression," presented at NIPS '18, 2018.

[24] Oren Rippel and Lubomir Bourdev, "Real-time adaptive image compression," presented at ICML '17, 2017.

[25] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár, "Lossy image compression with compressive autoencoders," presented at ICLR '17, 2017.

[26] George Toderici, Damien Vincent, Nick Johnston, *et al.*, "Full resolution image compression with recurrent neural networks," presented at CVPR '17, 2017.

[27] Yue Zhu, Fahim Chowdhury, Huansong Fu, *et al.*, "Entropy-aware I/O pipelining for large-scale deep learning on HPC systems," presented at MASCOTS '18, 2018.

[28] Aarati Kakaraparthy, Abhay Venkatesh, Amar Phanishayee, and Shivaram Venkataraman, "The case for unifying data loading in machine learning clusters," presented at HotCloud '19, 2019.

[29] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, *et al.*, "SHIELD: Fast, practical defense and vaccination for deep learning using JPEG compression," presented at KDD '18, 2018.

[30] Samuel Dodge and Lina Karam, "Quality resilient deep neural networks," *arXiv:1703.08119*, 2017.

[31] ——, "Understanding how image quality affects deep neural networks," presented at QoMEX '16, 2016.

[32] Sudeep Katakol, Basem Elbarashy, Luis Herranz, Joost van de Weijer, and Antonio M. López, "Distributed learning and inference with compressed images," *IEEE*

Transactions on Image Processing, vol. 30, pp. 3069–3083, 2021.

[33] Michał Koziarski and Bogusław Cyganek, "Impact of low resolution on image recognition with deep neural networks: An experimental study," *IJAMCS*, vol. 28, no. 4, pp. 735–744, 2018.

[34] Matt Poyser, Amir Atapour-Abarghouei, and Toby P. Breckon, "On the impact of lossy image and video compression on the performance of deep convolutional neural network architectures," presented at ICPR '21, 2021.

[35] Prasun Roy, Subhankar Ghosh, Saumik Bhattacharya, and Umapada Pal, "Effects of degradations on deep neural network architectures," *arXiv:1807.10108*, 2019.

[36] Cody Coleman, Deepak Narayanan, Daniel Kang, *et al.*, "DAWNBench: An end-to-end deep learning benchmark and competition," presented at NIPS '17, 2017.

[37] Peter Mattson, Vijay Janapa Reddi, Christine Cheng, *et al.*, "MLPerf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[38] Yu Emma Wang, Gu-Yeon Wei, and David Brooks, "A systematic methodology for analysis of deep learning hardware and software platforms," presented at MLSys '20, 2020.

[39] Hongyu Zhu, Mohamed Akrout, Bojian Zheng, *et al.*, "Benchmarking and analyzing deep neural network training," presented at IISWC '18, 2018.

[40] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry, *FFCV*, version Commit e97289f, 2022. [Online]. Available: https://github.com/libffcv/ffcv/.

[41] Webdataset contributors, *Webdataset: High performance I/O for large scale deep learning*, 2022. [Online]. Available: https://github.com/webdataset/webdataset.

[42] NVIDIA. "Deep learning examples." (2021), [Online]. Available: https://github.com/NVIDIA/DeepLearningExamples.

[43] Adam Paszke, Sam Gross, Francisco Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035.

[44] NVIDIA. "NVIDIA data loading library (DALI)." (2019), [Online]. Available: https://developer.nvidia.com/DALI.

[45] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja, "CompressAI: A PyTorch library and evaluation platform for end-to-end compression research," *arXiv:2011.03029*, 2020.

[46] Olga Russakovsky, Jia Deng, Hao Su, *et al.*, "ImageNet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[47] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba, "Places: A 10 million image database for scene recognition," *PAML*, vol. 40, no. 6, pp. 1452–1464, 2018.

[48] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.

[49] Google. "Optimize images: PageSpeed insights." (2018), [Online]. Available: https://developers.google.com/speed/docs/insights/OptimizeImages.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks," *CACM*, vol. 60, no. 6, pp. 84–90, 2017.

[51] Mingxing Tan and Quoc V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," presented at ICML '19, 2019.

[52] Andrew Howard, Mark Sandler, Grace Chu, *et al.*, "Searching for MobileNetV3," presented at ICCV '19, 2019.

[53] MLCommons Association. "MLPerf training v1.0 results," MLCommons. (2021), [Online]. Available: https://mlcommons.org/en/training-normal-10/.

[54] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, *et al.*, "Towards federated learning at scale: System design," presented at SysML '19, 2019.

[55] Peter Kairouz, H. Brendan McMahan, Brendan Avent, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1, pp. 1–210, 2021.

[56] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," presented at AISTATS '17, 2017.

[57] Sinno Jialin Pan and Qiang Yang, "A survey on transfer learning," *TKDE*, vol. 22, no. 10, pp. 1345–1359, 2010.

[58] Zohar Karnin, Tomer Koren, and Oren Somekh, "Almost optimal exploration in multi-armed bandits," presented at ICML '13, 2013.

[59] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," presented at ICLR '18, 2018.

[60] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee, "Enhanced deep residual networks for single image super-resolution," presented at CVPR '17, 2017.