PROGRESSIVE COMPRESSED RECORDS: TAKING A BYTE OUT OF DEEP LEARNING DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning training accesses vast amounts of data at high velocity, posing challenges for datasets retrieved over commodity networks and storage devices. We introduce a way to dynamically reduce the overhead of fetching and transporting training data with a method we term *Progressive Compressed Records (PCRs)*. PCRs deviate from previous formats by leveraging progressive compression to split each training example into multiple examples of increasingly higher fidelity, without adding to the total data size. Training examples of similar fidelity are grouped together, which reduces both the system overhead and data bandwidth needed to train a model. We show that models can be trained on aggressively compressed representations of the training data and still retain high accuracy, and that PCRs can enable a $2\times$ speedup on average over baseline formats using JPEG compression. Our results hold across deep learning architectures for a wide range of datasets: ImageNet, HAM10000, Stanford Cars, and CelebA-HQ.

1 INTRODUCTION

Today's wealth of image, audio, and text data enables novel machine learning applications, but also presents practical concerns, as modern models can require hundreds of thousands of machine hours to train (Chilimbi et al., 2014; Sun et al., 2017; Goyal et al., 2017; Brock et al., 2018; Mahajan et al., 2018). This is particularly true for deep learning workloads, where the training time can be roughly split into three components: the data pipeline (storage), the forward/backward computation (compute), and the variable synchronization (network). A plethora of work has investigated scaling deep learning from a computation or networking bound perspective (e.g., Dean et al., 2012; Cui et al., 2016; Abadi et al., 2015; Cui et al., 2014; Jouppi et al., 2017; Lim et al., 2018; Zhu et al., 2018; Alistarh et al., 2017; Lin et al., 2017; Wen et al., 2017; Wangni et al., 2018; Zhang et al., 2017) However, little attention has been paid toward scaling training from a storage perspective.

Unfortunately, with the amount of data being stored, system bandwidth limits how much data can be processed at any given time. Hardware trends point to an increasing divide between compute and networking or storage bandwidth (Li et al., 2016; Lim et al., 2018; Kurth et al., 2018). For example, the transportation of data for machine learning is a key factor in the design of the data centers themselves, such that they're built close to the data, which is expected to be serviced by slow storage media for the foreseeable future (David Reinsel, 2018). This, combined with the memory wall—a lack of bandwidth between compute and memory—suggests that, while computation may be sufficient moving forward, the mechanisms for moving data to the compute may not (Wulf & McKee, 1995; Kwon & Rhu, 2018; Hsieh et al., 2017). The storage pipeline is therefore a natural area to seek improvements in overall training time, which manifest from the storage medium, through the network, and into the compute nodes.

In this work, we propose a novel on-disk format called *Progressive Compressed Records* (PCRs) as a way to reduce the bandwidth cost associated with training over massive datasets. Our approach leverages a compression technique that decomposes each data item into *deltas*, each of which increases data fidelity. Naively storing replicas of the data at various compression levels could in fact increase storage costs. However, by carefully accounting for the layout of these deltas, we allow applications to control the tradeoff between dataset size (and, thus, bandwidth) and fidelity—all without additional overhead. As a result, we find that for a variety of popular deep learning models and datasets, bandwidth can be easily reduced by $2\times$ on average relative to JPEG compression without affecting model accuracy. Overall, this paper makes the following contributions:

1. In experiments with multiple architectures and several large-scale image datasets, we show that neural network training is robust to data compression in terms of test accuracy and training loss, but that the amount of compression that can be tolerated varies both by dataset and architecture.
2. We introduce *Progressive Compressed Records* (PCRs), a novel on-disk format for training data. PCRs combine progressive compression and careful data placement to enable applications to *dynamically* choose the fidelity of the data they consume, in order to reduce their data bandwidth.
3. We demonstrate that using PCRs, training speed can be improved by $2\times$ on average over standard formats using JPEG compression. This is achieved by selecting a lower data fidelity, which, in turn, reduces the amount of read data without significantly impairing model performance.

2 Background

Two complimentary concepts make up the process of storing training data: the layout of the data on the storage medium, and the representation of the data. Data layout is important because it can help fully utilize the potential of the underlying storage system. Data representation is important because it can allow the data to be processed faster. One example of data representation within the scope of this work is compression, which increases the useful computation per bit, a key property to consider as computation increases faster than bandwidth to storage.

Record Layouts. Learning from data requires sampling points from a training set, which will cause small, random accesses that are detrimental to storage device performance. Record layouts, such as TensorFlow's TFRecords (TFR) or MXNet's ImageRecord (Ima, a), attempt to alleviate this problem by batching data points together to increase access locality. Batches of training data are then accessed together, amortizing delays in access time across multiple data points. These batches of data are called *records*. The key to any record layout is the *serialization*, which is the conversion of data structures into byte streams. Record designs have different performance properties when written to disk, as shown in Figure 1.

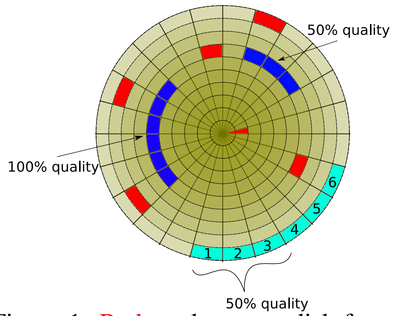
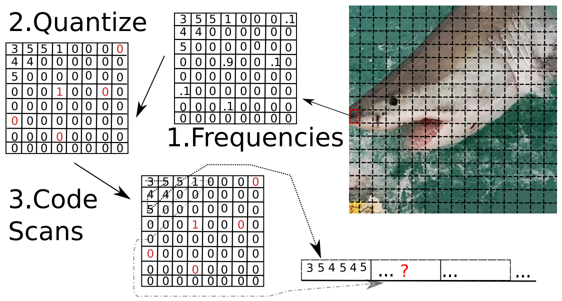


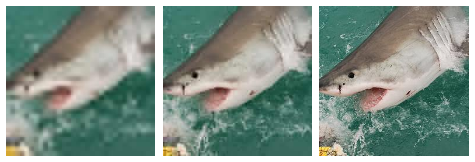
Figure 1: Red marks an on-disk format without records and has highly random behavior. Blue has records at various data qualities and has sequential behavior for a given quality. Cyan represents our proposed PCR format, which provides sequential behavior without the space overhead.

Image Compression. Compression is commonly used to represent training data. JPEG (Wallace, 1992) is one of the most popular formats for image compression, and is used ubiquitously in machine learning (e.g., Sajjadi et al., 2017; Ima, b;c; MSC). Most compression formats (including JPEG) only allow for the compression level, i.e., the trade-off between data size and fidelity, to be set at encoding time, which often results in choosing this level independent of the application. This can result in over-compressing the data, which may affect application performance, or under-compressing the data, which results in excess data size, and thus, slower performance. Worse, deep learning pipelines often involve an application-defined post-processing step (e.g., data augmentation), which may further distort an image and obscure the relationship between image quality and model accuracy (Bishop, 1995; Karras et al., 2017; Dziugaite et al., 2016; Arnab et al., 2018). While setting encoding-time parameters is unavoidable, the ability to decompress data as it becomes available (dynamic compression) provides a means to avoid some of the bandwidth expenses of under-compression by simply terminating decompression once sufficient quality is reached.

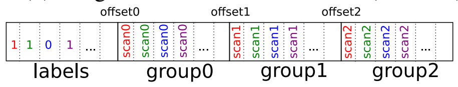
We provide a high-level illustration of the JPEG algorithm in Figure 2a, which provides the grounds to understanding how dynamic compression can be achieved. First, an image is split into blocks of size 8×8. Each block is converted into the frequency domain, such that frequency 0 is the average



(a) JPEG Algorithm (Before Entropy Coding)



(b) Images from scans: 1, 3, and 10 (baseline)



(c) PCR Encoding

Figure 2: **Left:** The JPEG algorithm proceeds by carving an image into blocks, which are then converted into frequencies, quantized, and then serialized. Progressive compression writes out a subset of important coefficients from each block before re-visiting the block. **Right Top:** A visualization of various scan levels from a 10-scan JPEG. Each scan progressively deblurs the image further. **Right Bottom:** Our PCR format progressively encodes images by scan. Data can be retrieved at a certain quality by sequentially reading up to the desired quality's offset (saving commensurate bandwidth).

color of the block, and higher frequencies encode rapid changes in the block. The low frequencies, such as the average value of the block, store the bulk of the perceptually-relevant content in the image (e.g., knowing the block is mostly blue is more important than knowing a white wave is rippling through it). Quantization, which discards information from the block and results in compression, thus prioritizes discarding higher frequencies. The resulting quantized table is then serialized into a flat form. Since data is rendered on a screen from left to right, top to bottom, it makes sense to encode the data in this manner, which results in a *sequential* format[1]. Decoding the resulting data is simply a matter of inverting (albeit losslessly) the process which we just described.

Progressive Image Compression. *Progressive* formats allow data to be read at varying degrees of compression without duplication. With the sequential case, data is order by blocks, and thus, partially reading the data results in "holes" in the image for those blocks (Wallace, 1992). Dynamic compression ensures that all spatial locations (i.e., blocks) get some information (*deltas*) before revising them (with more deltas). As progressive formats are simply a different traversal of the quantization matrix, with all else being equal, they contain the same information as sequential JPEG (JPE). Progressive JPEG, combined with an additional rearrangement of data, forms the basis of the idea behind PCRs. In Figure 2a, non-progressive formats serialize the image matrix in one pass, while progressive formats serialize the matrix in disjoint groups of deltas which are called *scans* (see Figure 2b). Scans are ordered by importance (e.g., the first scan contains higher quality deltas than subsequent scans). Thus, any references to images generated from scan n will implicitly assume that the image decoder had access to all prior scans (i.e., $\{$scan 1,scan 2,...,scan $(n-1)\}$). The image from scan 3 in Figure 2b makes use of scan 1 and scan 2 data, and scan 10 uses them all.

3 Progressive Compressed Records

In this section, we introduce a novel record format for machine learning training called Progressive Compressed Records (PCRs). PCRs are a combination of both layout and data representation. The reason for this is simple—having bandwidth savings is pointless if the hardware isn't operating efficiently in the first place. To this end, we introduce the concept of *scan groups* in Section 3.1, which leverage both layout and progressive compression to obtain dynamic decompression, allowing both high performance reads while reducing the amount of data read. Leveraging progressive compression, scan groups break images into deltas, which are then rearranged in order to faciliate reduced, yet sequential, data access. In Section 3.2, we discuss how PCRs are implemented, which covers both creating PCRs (encoding) and reading them (decoding). The PCR encoding process starts with the training dataset stored in JPEG format. PCRs then rearrange (twice) the bits associated with this training dataset to yield a bandwidth-optimized format. The benefits of the PCR implementation boiling down to a bit shuffle is that: 1) PCRs are easy to implement, 2) they're fundamentally loss-

[1]"Sequential" refers to in-memory, and should not be confused with sequential on-disk access.

less, 3) processing them is fast. Although PCRs can be implemented easily, they manifest in large speedups for a variety of scenerios.

3.1 Scan Groups

Scan groups are a collection of scans (deltas) of the same quality. Scan groups combine layout with progressive compression to allow reading subsets of the compressed data with high hardware efficiency. PCRs make the assumption that the entire training data will be read at the same quality. Using this assumption, scan groups rearrange the data such that all deltas of the same quality are grouped together. This is turn enables groups of deltas to be read together sequentially, which creates dynamicity in the decoding process. Since scans are sorted by importance, and scan groups are a set of scans, the scan groups are also sorted by importance.

To paint a clear representation of how scan groups work, we point the reader to Figure 2c. PCRs begin with some metadata which is assumed to be needed by all machine learning tasks, such as labels or bounding boxes. This metadata is small, and thus, not much of a concern if not all of it is used. The metadata is followed by scan groups, which consist of scans. The scan 1 representation of the shark in Figure 2b will be available in its record once data is read up to offset 1. Likewise, the scan 3 representation will be available once the record is read up to offset 3, and the representation will be more crisp as 3 scans were used per image, rather than 1. Reading up to the end of the record yields the most complete representation of the image. Since scan groups consist of groups of the same quality, every image contained in a record is available at the same quality at the same group offset. Users of PCRs can read data at a certain scan quality by simply reading the on-disk byte stream from the start of the PCR (i.e., offset 0) to the byte offset corresponding to the corresponding scan group. Avoiding reading the entire record results in bandwidth savings.

3.2 Implementation

There are two fundamental PCR implementation details: the encoding process and the decoding process. The encoding process transforms a set of JPEG files into a directory, which contains 1) a database for PCR metadata and 2) at least one `.pcr` file. The decoding process, which takes the directory as input and yields a set of JPEG images, does the opposite. The dataset is split into many PCRs, and thus the training process is processing tens to hundreds of `.pcr` files per epoch. The data loader is where the PCR decoding library interfaces with the inputs provided to deep learning librares (e.g., TensorFlow (Abadi et al., 2015), MXNet (Chen et al., 2015), PyTorch (Paszke et al., 2017)). Below, we describe how each of these steps is done, and we will open source our implementation upon paper acceptance.

Encoding. Given a set of images, the PCR encoder must break the images into scans, group the scans into scan groups, and sort the scan groups by quality. Once the groups are sorted, the PCR encoder can serialize the groups while taking note of their offsets (so that they may later be decoded). The metadata (e.g., labels) is pre-pended to the serialized representation, and the serialized representation is written to disk.

Our implementation uses `JPEGTRAN` (jpe) to (losslessly) transform the set of JPEG images into a set of progressive JPEG images. With the default settings, each JPEG is broken up into 10 scans. The encoder scans the binary representation of the progressive JPEG files, searching for the markers that designate the end of a scan group. The encoder thus has access to all 10 offsets within the JPEG files that can be used to determine the boundaries between scan regions. Forming scan groups requires grouping the scan regions with the same quality together, which can be done in one pass over the set of images corresponding to that PCR. This grouping must be reversable, as the decoding process will un-group the scans to reconstruct the original images. This grouping can be done with existing serialization libraries. We use Protobuf (Pro) to serialize the groups as well as the labels. However, it is key that every group (and the metadata) be serialized as a seperate message, as Protobuf can rearrange the contents within a message, and thus can rearrange the ordering of the groups themselves. We finally concatenate the contents of the messages and write them out as one file. Our PCR converter script converts 100k images in a matter of minutes.

Decoding. To decode a PCR, one has to first lookup the offsets in the database corresponding to that file. The offsets provide sufficient information to do a partial read of the file (e.g., instead of reading the entire file, we read only enough bytes to read up to the desired scan group). Decoding the JPEGs requires inverting the PCR scan-group grouping process for the available scan-groups. Since we are missing scan-groups, we terminate the byte stream with an End-of-Image (EOI) JPEG token — this allows most JPEG decoders to render the byte stream with only the available subset of scans. The bulk of the inverse conversion is done in 150 lines of C++ code.

Loader. We implemented PCR loaders using PyTorch's dataloader as well as DALI (DAL)'s `ExternalSource` operator. Ultimately, we found that a *pipeline* abstraction simplified loader design, since record-based datasets can be easily iterated sequentially. This abstraction was more convenient that the PyTorch Dataloader abstraction, which assumes that we can index randomly into an in-memory data structure (e.g., `i = RandInt(0, n); (x, y) = data[i];`), and is therefore hard to use for constantly fetching record formats off disk. Our implementation, while being only several hundred lines of code, obtains image rates that are competitive (e.g., faster/slower depending on number of scans) with the included DALI TFRecord loader, showing that PCRs can be implemented efficiently (i.e., fast enough to rarely bottleneck data loading) with a low amount of engineering effort. Writing the entire PCR pipeline in native code is future work.

4 EXPERIMENTS

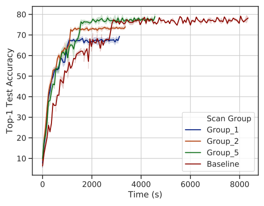
This section presents our evaluation of PCRs using a suite of large-scale image datasets. Large images are more straining on a system's network and storage, thus our evaluation focuses on high-resolution datasets. Overall, we find that PCRs are most effective on larger images due to the large amount of redundant information found in them, which results in better compression. We describe our experimental setup in Section 4.1. We present our evaluation results in Section 4.2. In Section 4.3, we analyze the relationship between image quality and time-to-accuracy. Finally, in Section 4.4, we show results that explain the cause of the speedups to the data loader. To avoid measuring implementation differences with other loaders, our evaluation focuses on the differences obtained by reading various amounts of scan groups. Reading all the data is the baseline.

4.1 EVALUATION SETUP

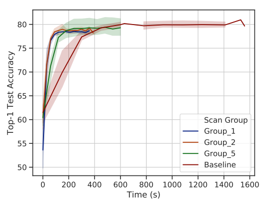
Our evaluation uses the ImageNet ILSVRC (Deng et al., 2009; Russakovsky et al., 2015), HAM10000 (Tschandl et al., 2018), Stanford Cars (Krause et al., 2013), and CelebA-HQ Karras et al. (2017) datasets which we describe next.

Datasets. ImageNet provides a wide diversity of classes, of which we focus on the first 100 to make training times more tractable. We refer to this subset of ImageNet as **ImageNet-100**. Since classes are roughly ordered by ImageNet categories, this results in a fine-grained, i.e., hard to classify, multiclass task. We convert the dataset into PCRs in batches of 1024, which results in 126 PCRs. We split the **HAM10000** dataset randomly 80%/20% between train and test. We convert the dataset into PCRs in batches of 64, which results in 125 PCRs of similar size as the ImageNet ones. The Stanford **Cars** dataset is another fine-grained classification dataset, since all images are cars and there are 196 classes spread over 16k images. We believe these datasets highlight some of the worst-case training scenarios, as it is considerably easier to predict highly compressed variants of unrelated images by exploiting low frequency image statistics (e.g., planes vs. frogs). Cars has 63 PCRs. CelebA-HQ is a high-resolution derivative of the CelebA dataset (Liu et al., 2015), which consists 30k celebrity faces at 1024^2. We use the annotations provided by CelebA to construct a smiling or not smiling dataset, which we term **CelebAHQ-Smile**. We split the 30k dataset into 80%/20% train/test and we convert the training set into 93 PCRs. All datasets except for HAM10000 use resizing, crop, and horizontal-flip augmentations, as is standard for ImageNet training. We provide examples of scan groups for these datasets, as well as how they look at various scans, in the appendix.

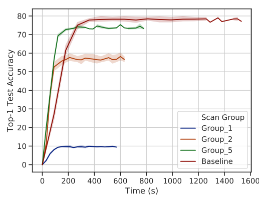
Training Regime. We use pretrained ImageNet weights for HAM10000 and Cars due to the limited amount of training data. We use standard ImageNet training, starting the learning rate at 0.1 and dropping it on epoch 30 and 60 by $10\times$. After augmentations, all inputs are of size 224×224. The pretrained experiments (HAM10000 and Cars) start at a learning rate of 0.01 to avoid changing the initialization too aggressively. We use `fp16` training as it gives us an additional 18% images per second (from 3200 to 3900). We use a ResNet18 (He et al., 2016) and ShuffleNetv2 (Ma et al., 2018)



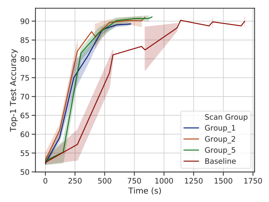
(a) ImageNet-100



(b) HAM10000



(c) Stanford Cars



(d) CelebAHQ-Smile

Figure 3: ImageNet, HAM10000, Stanford Cars, and CelebA-HQ top-1 test performance with ResNet-18. Top-5 accuracies mirror the top-1 accuracies trends. Time is the x-axis (seconds) and is relative to first epoch. 95% confidence intervals are shown.

architecture for our experiments with a batch size of 128 per each worker. We run each experiment at least 3 times to obtain confidence intervals.

System Setup. We run distributed experiments on a 16-node Ceph (Weil et al., 2006) cluster connected with a Cisco Nexus 3264-Q 64-port QSFP+ 40GbE switch. Each node has a 16–core Intel E5–2698Bv3 Xeon 2GHz CPU, 64GiB RAM, NVIDIA TitanX, 4TB 7200RPM Seagate ST4000NM0023 HDD, and a Mellanox MCX314A-BCCT 40GbE NIC. All nodes run Linux kernel 4.15 on Ubuntu 18.04, CUDA10, and the Luminous release (v12.2.12) of Ceph. We use 6 of the nodes as Ceph nodes; 5 nodes are dedicated as storage nodes in the form of Object Storage Devices (OSDs), and one node is used as a Ceph metadata server (MDS). The rest of the 10 nodes are used as machine learning workers for the training process. This means there is a 2:1 ratio between compute and storage nodes. The exception to this is Figure 5a, which was run with 8 additional worker nodes in the interest of time. We use PyTorch (Paszke et al., 2017) with NVIDIA Apex (ape). We use 4 threads to prefetch data in the loader.

4.2 Time to Accuracy

The time-to-accuracy results for ResNet18 training are presented in Figure 3, while those of ShuffleNetv2 are presented in Figure 5. See the appendix for the corresponding training loss results. All scan groups within a dataset were run for the same number of epochs (at least 90), so lower scan groups finish earlier. We sample test accuracy every 15 epochs for non-ImageNet datasets to reduce interference with training, since test set evaluation can take significant time. For visualization, we group ImageNet data points by 60 seconds, and the rest by 30 seconds.

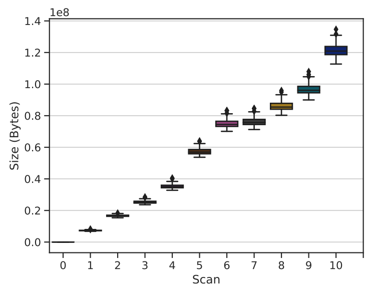
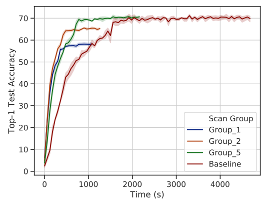


Figure 4: The size in bytes of scan groups for ImageNet-100.

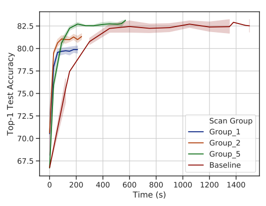
First, we make note that for all datasets, except for Cars, *PCRs provide a* $2\times$ *boost to time-to-accuracy compared to the baseline*. The reason for this speedup is that lower scan groups cost less. As shown in Figure 4, scan group 5 is roughly half the size of the baseline, and scan group 1 is a fifth of scan group 5. This trend holds across datasets (see appendix). As we will discuss in Section 4.4, the space savings manifest in reduced data loader latencies.

Second, we note that *there is an inherent trade-off between convergence quality and the speedup attained by using less storage resources*. In general, although lower quality scan groups allow the system to operate more efficiently, they do so at the expense of model convergence. Scan group 1, the lowest quality scan, performs poorly, especially on Cars, where fine-grain details are important. Scan groups limit the maximum achievable accuracy on a task; if learning plateaus prematurely, applications should raise the scan group in a manner similar to learning rate drops.

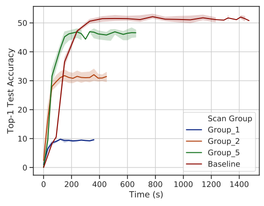
Third, *the relative rankings of scan groups is relatively stable across models*. We further relate these rankings to the quality of the scan groups in Section 4.3. Our conclusion is that, for most datasets, scan group 5 costs half as much in terms of bandwidth, but reaches the same level of test accuracy as the baseline — thus, it is a good group to default to. This is most clear for ImageNet and HAM10000, which are challenging enough for small variations in image quality to make a



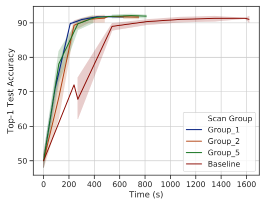
(a) ImageNet-100



(b) HAM10000



(c) Stanford Cars

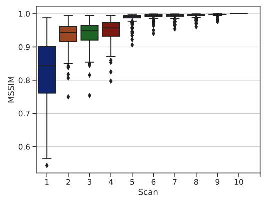


(d) CelebAHQ-Smile

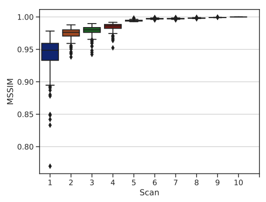
Figure 5: ImageNet, HAM10000, and Stanford Cars top-1 test performance with ShuffleNetv2. ImageNet-100 was run on 8 additional ML nodes, yielding 18 worker nodes. Time is the x-axis (seconds) and is relative to first epoch. 95% confidence intervals are shown.

commensurate difference in test accuracy. This is in contrast to Cars, which is too fine-grain to allow images to be degraded, and CelebAHQ-Smile, which is too coarse-grained.

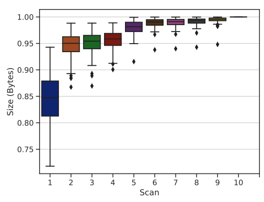
4.3 The Relationship Between Image Quality and Test Accuracy



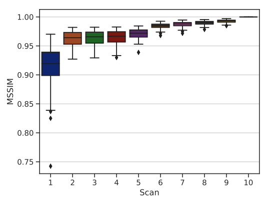
(a) ImageNet-100



(b) HAM10000



(c) Stanford Cars



(d) CelebAHQ

Figure 6: The reconstruction quality of using various amounts of scans using the MSSIM metric.

We use MSSIM (Wang et al., 2003), a standard measure of image similarity, to compare how various scans approximate the reference image, and we show the results in Figure 6. We find that there is a strong connection between MSSIM and the resulting final test accuracy, especially when comparing scan groups *within* a task. Our preliminary tests found that scan groups that have very similar MSSIM perform very similarly, so we show only groups 1, 2, 5, and the baseline, which corresponds to scan group 10. Due to the way progressive JPEG is coded by default, groups tend to cluster (e.g., 2, 3, and 4 are usually similar, while 5 introduces a difference). We found that MSSIM being poor, such as in scan 1 for Cars, or MSSIM being close to baseline, such as scan 5 for HAM10000, were good predictors of relative test accuracy. As a concrete example, Cars has a slightly higher MSSIM on scan 6 than scan 5, and using scan 6 results in about 2% accuracy boost over scan 5 for ResNet18.

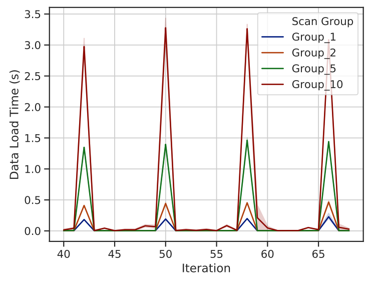


Figure 7: Data loading stalls are periodic and followed by extents of prefetched data. Lower scan groups result in lower magnitude stalls.

4.4 The Relationship Between Scans and Data Stalls

The datasets we evaluated have shown that data loading can slow down the training process. To highlight these slowdowns, and the improvements PCRs achieve by not using all scan groups, we present the loading time of data for the ResNet18 ImageNet-100 run in Figure 7. We obtain similar results for the other datasets. The baseline of using all scan group results in high periodic loading stalls, where the prefetching queue was drained. Upon blocking, training cannot proceed until the worker threads obtain a full batch of data. Because there are multiple threads fetching the data, future batches are also fetched, resulting in periods of (mostly) no stalls. Using less scan groups reduces the amount of data read, which results in lower magnitude stalls. These stalls occur with both DALI and PyTorch loaders.

5 Related Work

Data Reduction. Prior work focused on training machine learning models on reduced datasets. Data reduction techniques, such as sketching, coresets, clustering, and sampling, have been used to reduce the size of a training set (Karnin & Liberty, 2019; Feldman et al., 2013; Liberty, 2013; Woodruff et al., 2014; Daniely et al., 2017; Kabkab et al., 2016). Other work develops reformulated large-scale SVM training into a storage-friendly form (Matsushima et al., 2012). Recently, dataset distillation was proposed (Wang et al., 2018), where a model's parameters were compressed into in a few training examples, such that the original model could be recovered by training on the reduced dataset. Our work takes a different approach to these methods, as we reduce the size of the dataset with compression techniques (i.e., JPEG) that are already in use.

Compression. Prior work on compressed neural network models to reduce their transmission cost or allow the model to be efficiently implemented in hardware Han et al. (2015a; 2016; 2015b); Cheng et al. (2017); Xu et al. (2018); Hwang & Sung (2014); Anwar et al. (2015); Denton et al. (2014). Our work attempts to compress data for training, and not the network itself. Prior work has also looked into compressing neural network training network traffic (Lim et al., 2018; Alistarh et al., 2017; Lin et al., 2017; Wen et al., 2017; Wangni et al., 2018; Zhang et al., 2017). While our work focuses on mainly on storage, this line of work is complementary to ours, as both gradients and training data can use the same network resources. There is a line of work in using compressed formats to reduce system bandwidth requirements, such as in the context of databases and computer memories Zukowski et al. (2006); Abadi et al. (2006); Pekhimenko et al. (2018); Houston & Koh; Pekhimenko et al. (2012). Our work differs in that it exploits ML-specific resilience to lossy transformations. Training models directly on compressed representation for the purpose of avoiding decoding or reducing model complexity has been done (Gueguen et al., 2018; Torfason et al., 2018; Fu & Guimaraes, 2016; Ulicny & Dahyot, 2017). Our work differs in motivation, as we do not focus on model computation or make modifications to the models.

Training Over Large Datasets. Training with massive amounts of parallelism (thus stressing system bandwidth) while achieving near-linear speedup has been the focus of previous work. A common objective is training models over ImageNet in a record amount of time (Goyal et al., 2017; You et al., 2017; 2018; Jia et al., 2018; Ying et al., 2018; You et al.). This line of work, while demonstrating immense bandwidth needs, typically keeps data in memory, avoiding storage problems altogether. Recently, the high performance computing community has seen an interest in training models at massive scale, training deep models over two cluster with 5300 P100 and 27360 V100 GPUs (Kurth et al., 2018). This work had a large emphasis on I/O, as each GPU can match or exceed the bandwidth of a disk; to get around this bandwidth problem, each node was given a subset of images to store locally rather than performing a true sampling of data. Our work attempts to reduce the storage bottleneck altogether, such that a single disk could service potentially many GPUs. A seperate line of work shows that I/O is a significant bottleneck for certain tasks, and proposes optimizing I/O via a set of deep-learning specific optimization to LMDB (Pumma et al., 2019). In contrast, our focus is more on data representation, which is agnostic of the storage system.

6 Conclusion

In order to continue making advances in machine learning, researchers will need access to larger and larger datasets, which will eventually spill into storage. Storage and networking bandwidth, which are precious resources, can be better utilized with efficient compression formats. We introduce a novel record format, Progressive Compressed Records, that trades off data fidelity with storage and network demands, allowing the same model to be trained with $2\times$ less storage bandwidth while retaining model accuracy. PCRs leverage progressive compression to split training examples into multiple examples of increasingly higher fidelity without the overheads of naive approaches. PCRs avoid duplicating space, are easy to implement, and can be applied to a broad range of tasks dynamically. While we apply our format in this work specifically to images with JPEG compression, PCRs are general enough to handle various data modalities or additional compression techniques; future work will include exploring these directions in fields outside visual classification, such as audio generation or video segmentation.

REFERENCES

Dali. `https://github.com/NVIDIA/DALI`. Accessed: 08-30-2019.

Imagerecord. `https://gluon-cv.mxnet.io/build/examples_datasets/recordio.html`, a. Accessed: 09-22-2019.

Imagenet large scale visual recognition challenge 2017 (ilsvrc2017). `http://image-net.org/challenges/LSVRC/2017/download-images-1p39.php`, b. Accessed: 08-30-2019.

List of all image urls of fall 2011 release. `http://image-net.org/imagenet_data/urls/imagenet_fall11_urls.tgz`, c. Accessed: 08-30-2019.

Jpegtran libjpeg.txt.txt. `https://github.com/cloudflare/jpegtran/blob/master/libjpeg.txt`. Accessed: 08-30-2019.

Tensorflow build_mscoco_data. `https://github.com/tensorflow/models/blob/master/research/im2txt/im2txt/data/build_mscoco_data.py`. Accessed: 08-30-2019.

Protocol buffers. `https://developers.google.com/protocol-buffers/`. Accessed: 08-29-2019.

Using tfrecords and tf.example. `https://www.tensorflow.org/tutorials/load_data/tf_records/`. Accessed: 08-29-2019.

apex. `https://github.com/NVIDIA/apex`. Accessed: 09-22-2019.

jpegtran(1) - linux max page. `https://linux.die.net/man/1/jpegtran`. Accessed: 09-22-2019.

Daniel Abadi, Samuel Madden, and Miguel Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 671–682. ACM, 2006.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2017.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1131–1135. IEEE, 2015.

Anurag Arnab, Ondrej Miksik, and Philip HS Torr. On the robustness of semantic segmentation models to adversarial attacks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 888–897, 2018.

Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 571–582, 2014.

Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. Exploiting bounded staleness to speed up big data analytics. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 37–48, Philadelphia, PA, 2014. USENIX Association. ISBN 978-1-931971-10-2. URL `https://www.usenix.org/conference/atc14/technical-sessions/presentation/cui`.

Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*, pp. 4. ACM, 2016.

Amit Daniely, Nevena Lazic, Yoram Singer, and Kunal Talwar. Short and deep: Sketching and neural networks. 2017. URL `https://openreview.net/pdf?id=r1br_2Kge`.

John Rydning David Reinsel, John Gantz. Data age 2025: The digitization of the world from edge to core. Technical report, 2018. URL `https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf`.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1223–1231. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf`.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pp. 1269–1277, 2014.

Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M Roy. A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853*, 2016.

Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1434–1453. Society for Industrial and Applied Mathematics, 2013.

Dan Fu and Gabriel Guimaraes. Using compression to speed up image classification in artificial neural networks. 2016.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. Faster neural networks straight from jpeg. In *Advances in Neural Information Processing Systems*, pp. 3933–3944, 2018.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254. IEEE, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Mike Houston and Wei Koh. Compression in the graphics pipeline.

Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching {LAN} speeds. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pp. 629–647, 2017.

Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6. IEEE, 2014.

Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12. IEEE, 2017.

Maya Kabkab, Azadeh Alavi, and Rama Chellappa. Dcnns on a diet: Sampling strategies for reducing the training set size. *arXiv preprint arXiv:1606.04232*, 2016.

Zohar Karnin and Edo Liberty. Discrepancy, coresets, and sketches in machine learning. In Alina Beygelzimer and Daniel Hsu (eds.), *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *Proceedings of Machine Learning Research*, pp. 1975–1993, Phoenix, USA, 25–28 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v99/karnin19a.html`.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, et al. Exascale deep learning for climate analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, pp. 51. IEEE Press, 2018.

Youngeun Kwon and Minsoo Rhu. Beyond the memory wall: A case for memory-centric hpc system for deep learning. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 148–161. IEEE, 2018.

Jing Li, Hung-Wei Tseng, Chunbin Lin, Yannis Papakonstantinou, and Steven Swanson. Hippogriffdb: Balancing i/o and gpu bandwidth in big data analytics. *Proceedings of the VLDB Endowment*, 9(14):1647–1658, 2016.

Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 581–588. ACM, 2013.

Hyeontaek Lim, David G Andersen, and Michael Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning. *arXiv preprint arXiv:1802.07389*, 2018.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 116–131, 2018.

Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 181–196, 2018.

Shin Matsushima, SVN Vishwanathan, and Alexander J Smola. Linear support vector machines via dual cached loops. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 177–185. ACM, 2012.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Base-delta-immediate compression: practical data compression for on-chip caches. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 377–388. ACM, 2012.

Gennady Pekhimenko, Chuanxiong Guo, Myeongjae Jeon, Peng Huang, and Lidong Zhou. Tersecades: Efficient data compression in stream processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 307–320, 2018.

Sarunya Pumma, Min Si, Wu-Chun Feng, and Pavan Balaji. Scalable deep learning via i/o analysis and optimization. *ACM Trans. Parallel Comput*, 1(1), 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Mehdi SM Sajjadi, Bernhard Scholkopf, and Michael Hirsch. Enhancenet: Single image super-resolution through automated texture synthesis. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4491–4500, 2017.

Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Robert Torfason, Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Towards image understanding from deep compression without decoding. *arXiv preprint arXiv:1803.06131*, 2018.

Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5: 180161, 2018.

Matej Ulicny and Rozenn Dahyot. On using cnn with dct based image data. 2017.

Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pp. 1398–1402. Ieee, 2003.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pp. 1299–1309, 2018.

Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320. USENIX Association, 2006.

Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in neural information processing systems*, pp. 1509–1519, 2017.

David P Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, March 1995. ISSN 0163-5964. doi: 10.1145/216585.216588. URL `http://doi.acm.org/10.1145/216585.216588`.

Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992*, 2018.

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. 100-epoch imagenet training with alexnet in 24 minutes.

Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. 2017.

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pp. 1. ACM, 2018.

Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pp. 181–193, 2017.

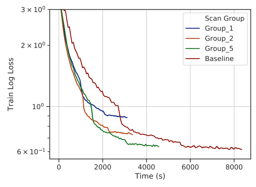
Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. Tbd: Benchmarking and analyzing deep neural network training. *arXiv preprint arXiv:1803.06905*, 2018.

Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. Super-scalar ram-cpu cache compression. In *22nd International Conference on Data Engineering (ICDE'06)*, pp. 59–59. IEEE, 2006.

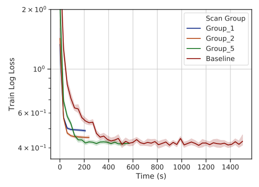
A Appendix

A.1 Additional Experiment Plots

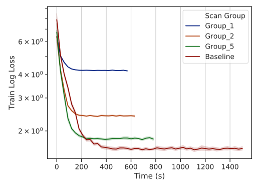
Below, we provide additional experiment plots (e.g., loss and scan size) that were omitted in the main text.



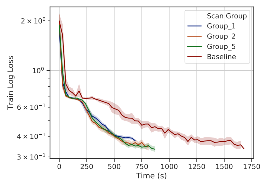
(a) ImageNet-100



(b) HAM10000

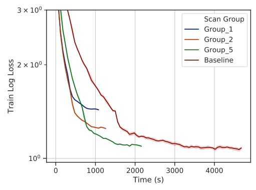


(c) Stanford Cars

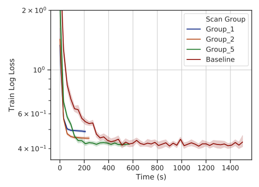


(d) CelebAHQ-Smile

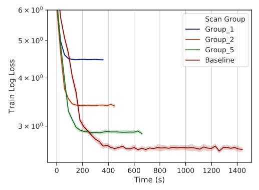
Figure 8: ImageNet-100, HAM10000, Stanford Cars, and CelebA-HQ train loss with ResNet-18. Time is the x-axis (seconds) and is relative to first epoch. All runs were completed on the Ceph cluster.



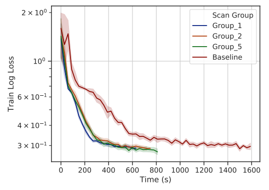
(a) ImageNet-100



(b) HAM10000

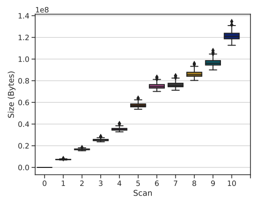


(c) Stanford Cars

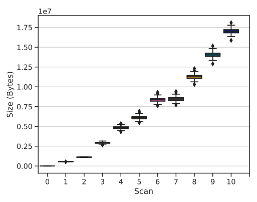


(d) CelebAHQ-Smile

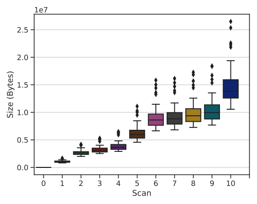
Figure 9: ImageNet, HAM10000, Stanford Cars, and CelebA-HQ train loss with ShuffleNetv2. Time is the x-axis (seconds) and is relative to first epoch. All runs were completed on the Ceph cluster.



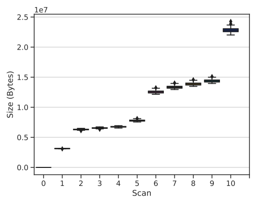
(a) ImageNet-100



(b) HAM10000



(c) Stanford Cars

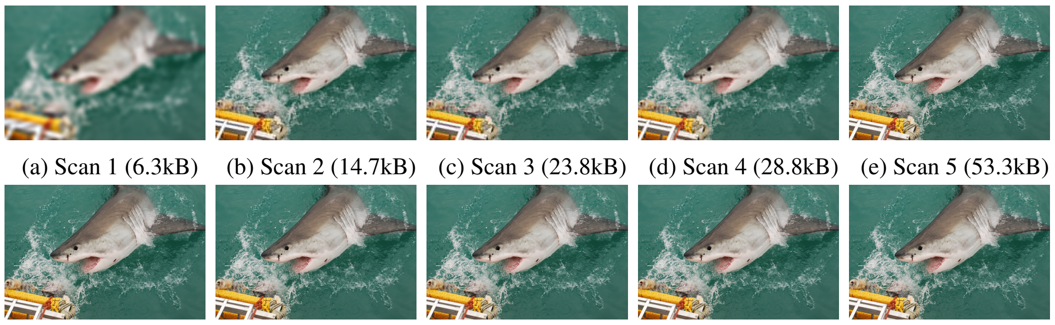


(d) CelebAHQ-Smile

Figure 10: The size in bytes of various levels of scans read. Scan group 0 is shown, which contains only labels and is typically $\sim$ 100 Bytes. Each scan adds roughly a linear amount of data, although certain scans do add considerably more than others due to techniques like chroma subsampling. Using all 10 scans can require over an order of magnitude more bandwidth than 1–2 scans.

A.2 Image Examples by Scan

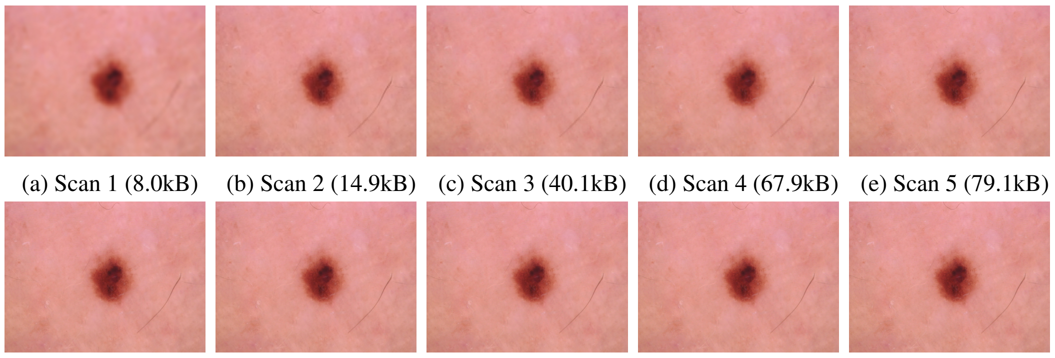
Below, we provide image examples from each dataset that show what each scan group looks like. Images can use a remarkably low amount of scan groups without impacting visual quality, which can manifest is space savings if used accordingly.



(a) Scan 1 (6.3kB) (b) Scan 2 (14.7kB) (c) Scan 3 (23.8kB) (d) Scan 4 (28.8kB) (e) Scan 5 (53.3kB)

(f) Scan 6 (72.0kB) (g) Scan 7 (73.1kB) (h) Scan 8 (84.6kB) (i) Scan 9 (91.0kB) (j) Scan 10 (115.3kB)

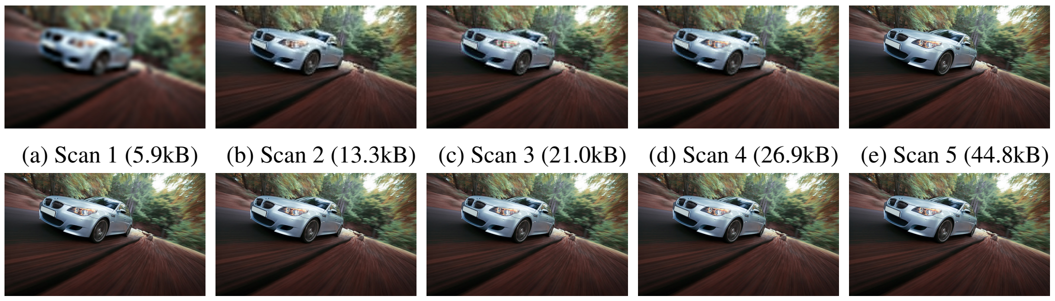
Figure 11: Examples of scans for ImageNet. The amount of scans needed to hit an acceptable level of quality is small. Having a larger final size (high quality compression) results in more savings for earlier scans.



(a) Scan 1 (8.0kB) (b) Scan 2 (14.9kB) (c) Scan 3 (40.1kB) (d) Scan 4 (67.9kB) (e) Scan 5 (79.1kB)

(f) Scan 6 (116.2kB) (g) Scan 7 (117.8kB) (h) Scan 8 (165.3kB) (i) Scan 9 (213.2kB) (j) Scan 10 (260.9kB)

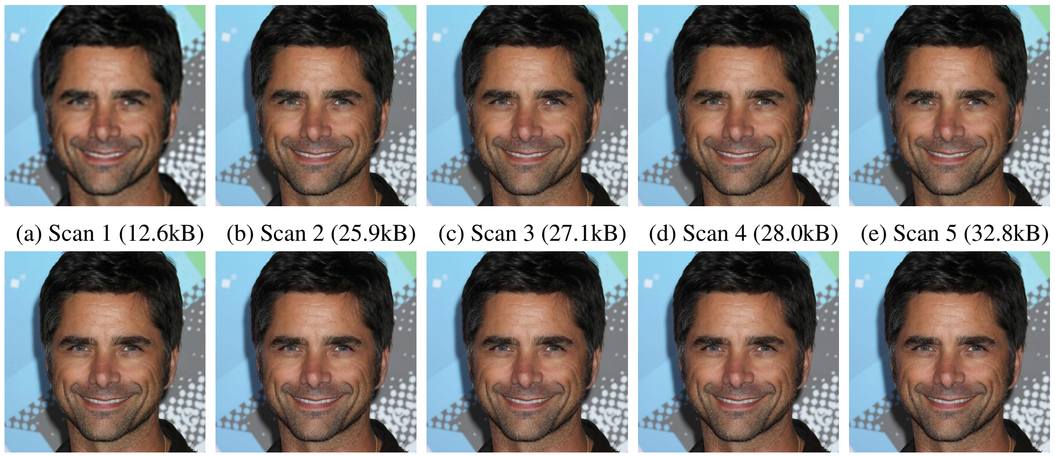
Figure 12: Examples of scans for HAM10000. The amount of scans needed to hit an acceptable level of quality is small. Having a larger final size (high quality compression) results in more savings for earlier scans.



(a) Scan 1 (5.9kB) (b) Scan 2 (13.3kB) (c) Scan 3 (21.0kB) (d) Scan 4 (26.9kB) (e) Scan 5 (44.8kB)

(f) Scan 6 (58.8kB) (g) Scan 7 (59.7kB) (h) Scan 8 (69.7kB) (i) Scan 9 (77.0kB) (j) Scan 10 (96.6kB)

Figure 13: Examples of scans for Stanford Cars. The amount of scans needed to hit an acceptable level of quality is small. Having a larger final size (high quality compression) results in more savings for earlier scans.



(a) Scan 1 (12.6kB) (b) Scan 2 (25.9kB) (c) Scan 3 (27.1kB) (d) Scan 4 (28.0kB) (e) Scan 5 (32.8kB)

(f) Scan 6 (57.5kB) (g) Scan 7 (60.6kB) (h) Scan 8 (62.9kB) (i) Scan 9 (65.1kB) (j) Scan 10 (110.7kB)

Figure 14: Examples of scans for CelebA-HQ. The amount of scans needed to hit an acceptable level of quality is small. Having a larger final size (high quality compression) results in more savings for earlier scans.