Frequency-Store: Scaling Image AI by A Column-Store for Images

Utku Sirin, Victoria Kauffman, Aadit Saluja, Florian Klein, Jeremy Hsu, Stratos Idreos
Harvard University

ABSTRACT

Artificial intelligence over images improves every aspect of modern human life and has shown great success across numerous applications. However, it is costly to perform image AI. Image AI pipelines need to move heavy image files over the network so that many applications can concurrently process the images with varying resource budgets and performance requirements. As a result, data movement dominates the end-to-end image AI cost.

This work presents Frequency-Store, the first column-store for images. Our intuition is that images do not need to be consumed by image AI one whole image at a time. Instead, there are "components" of data within each image that can be consumed separately and thus also can be stored separately. This decomposition allows the sharing of data movement across image AI processing pipelines both for training and inference.

Frequency-Store breaks images into columns and stores batches of images column by column rather than storing individual images file by file. It utilizes the inherent blocks and frequencies-based structure in image data and defines a novel column abstraction. Column-wise storage allows applications with various characteristics and resource demands to share data efficiently. Columns store data items with similar characteristics, allowing tight data representations and efficient compression. We show that Frequency-Store improves inference/training time by up to 11x and compression ratio by up to 2.2x compared to state-of-the-art image AI storage.

1 EFFICIENT IMAGE AI BY COLUMNAR STORAGE

Image AI Improves Every Aspect of Modern Human Life. Image AI has shown great success in numerous areas, from medical imaging to self-driving cars. Medical doctors now use image AI to get help in their decision-making process in the early/late detection of diseases. Companies deploy image AI tools to enhance worker safety conditions, thereby increasing productivity. Farmers use image AI to efficiently monitor the conditions of their crops and take preventive actions against any potential disease, further improving their yield [9, 11, 19].

Problem: Image AI is Expensive. Every year, billions of digital cameras capture trillions of images. These images consume thousands of petabytes of storage in the cloud. Numerous AI applications process these massive datasets for various purposes, such as personalized content management, image reconstruction, and visual captioning. Applications access data over remote storage servers, and need to move and process data in compute nodes using high-end processors. This incurs an immense dollar cost for the cloud vendors and application developers, limiting the accessibility of image AI [7].

Inference and Training are Equally Costly. The lifetime of an AI model includes two stages: training and inference. Training is where the model learns how to perform the task. Inference is where the model is deployed and performs a learned task. Both training and inference are important cost components. While some applications perform frequent re-trainings by auto-generated data, some applications are deployed across billions of devices concurrently and continuously performing inference, and some other applications perform both training and inference equally frequently.

Data Movement Dominates the Cost. Most images are captured today by a network-attached device, such as a mobile phone, or are stored in a remote storage server for later analysis. Therefore, performing inference/training or analysis of the images requires transferring data over the network. In Figure 1, we break down the inference time of an AI model when applications move images over the network. Inference requires (i) reading images over the network, (ii) CPU-decoding them, (iii) moving them over PCIe, and (iv) finally processing them on GPU by the AI model[1]. We perform the analysis for two storage formats: image calculator (IC), which is a recently proposed self-designing storage format for image AI [16], and standard JPEG [4]. As the figure shows, over 95% of the time goes to reading the data over the network. Analyzing training costs leads to similar results. Therefore, data movement dominates image AI cost.

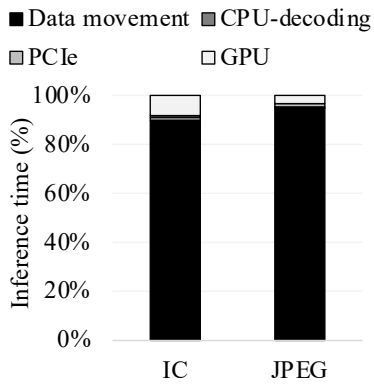

Figure 1: Data movement dominates image AI inference cost.

Intuition 1: Applications Need Only A Part of the Data. Images today are stored file by file. Hence, once an application reads an image, it reads it as a whole from the source to the node it runs. Recent work has shown that applications can perform the same processing by using only some part of the data, which allows reduced inference and training times by sacrificing little or no loss in how successful the AI task is performed [16, 18]. Applications decide on what part of the data is necessary. Every application has unique characteristics and budgets regarding how much data it needs.

 15th Annual Conference on Innovative Data Systems Research (CIDR '25). January 19-22, Amsterdam, The Netherlands

[1]ImageNet data, ResNet50 model, A100 GPU, and average network bandwidth of 50Mbps. Please see Section 6.1 for further details.

Intuition 2: Image Data is Multimodal. Furthermore, we observe various data ranges and distributions in image data. Some parts of the image have wide ranges with significant variance, whereas others have narrow ranges at varying levels with variances. Storing images file by file requires using a single data width for all images wide enough to cover the entire range. This wastes many bits, results in inefficient storage, and increases data movement costs.

Solution: Columnar Storage for Images. This work presents Frequency-Store, the first image storage that uses columns for images. Frequency-Store breaks a batch of images into a set of columns and stores each column as a separate compressed data file instead of storing images file by file as traditionally done. Columnar storage allows Frequency-Store to read data files that are only necessary for the application, dramatically reducing data movement cost and, hence, the inference and training time of image AI applications. Furthermore, Frequency-Store defines columns based on data items with similar characteristics, enabling tight data representations and efficiently compressing the images.

Contributions. In summary, our contributions are as follows.

- We introduce the concept of shareable storage formats. Shareable storage formats are a family of storage formats that allow sharing data among each other and trading AI model quality with various levels of resource budgets. Every shareable storage format is either a subset or superset of another storage format in the design space.
- We show that image data is multimodal. It contains multiple data distributions, some of which have high magnitudes with wide ranges and significant variances and some of which have low magnitudes with varying levels of narrow ranges and variances.
- We present a column abstraction for storing images. Our column abstraction allows efficient data sharing across applications and efficient data encoding by grouping data items with similar data characteristics.
- We designed and implemented the Frequency-Store, the first column-store for images. We demonstrate that Frequency-Store improves inference and training times of image AI applications by up to 11x and compression ratio by up to 2.2x, compared to state-of-the-art storage formats.

2 MOTIVATION

Application-specific Storage. Applications have unique problem characteristics with varying resource budgets and performance requirements. As a result, the amount of data they need to perform their task successfully depends on the task-specific features. For example, some applications work on a simple problem where only a little data is enough, whereas others might need all the data. Other applications might need all the data but only have a small resource budget and hence need to trade AI model quality for a lower data movement cost.

Breaking the Trade-Off. Current storage formats force applications to choose between the following two: either (i) choose a fixed storage, such as JPEG, and use it across all the applications sharing the data, or (ii) choose an application-specific storage, such as image calculator [16] and use a different data for each application. The first option allows applications to share data but is a general-purpose storage format with heavy image files. The second option stores images much more efficiently thanks to specialized storage but does not allow applications to share data. In this work, we ask the following question:

Can we use application-specific storage and also share data across different applications?

We show that the answer to this question is yes, and the solution relies on two main concepts: shareable storage formats and columnar storage. Shareable storage formats define a family of storage formats, where storage formats can share data among each other and each storage format offers a different trade-off between the data cost and AI model quality. Columnar storage breaks image data into pieces, so every storage format in the shareable storage formats can be easily reconstructed without creating a new copy of the data. Columnar storage further allows tight data representations and efficient compressing of the data. The following sections present an overview of Frequency-Store and introduce the concept of shareable storage formats and how we implement it using columnar data organization.

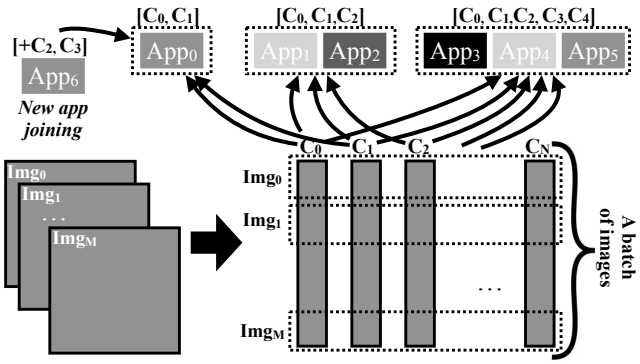

Figure 2: Frequency-Store keeps images column by column rather than file by file. As a result, it allows applications to share data and read only the data they need.

3 OVERVIEW

Figure 2 presents an overview of Frequency-Store. Frequency-Store keeps a single copy of the data, where images are stored column by column instead of file by file. It breaks a batch of images into a set of columns and keeps each column in a separate data file. An image file in this representation is spread across different columns, i.e., data files, as shown by Figure 2. Columnar storage allows modular access to image data such that only the columns that applications need will be accessed, and applications will share columns that moved to the same or closely-located machines. New applications with different resource budgets can flexibly join the execution and share data with applications colocated in the same machine. As a result, Frequency-Store provides both application-specific storage, which brings *efficiency*, and sharing data across applications, which brings *scalability*.

4 SHAREABLE STORAGE FORMATS

Shareable storage formats are a family of storage formats where every storage format contains data that is either superset or subset of another storage format. Most image storage formats lossily compress the data [4, 16]. Given an image, they perform operations, such as pruning and quantization, to reduce data size and efficiently store the images.

Each storage format in the set of shareable storage formats lossily compresses the data at a different level. While some storage formats highly compress the data and retain only a tiny amount of information, some lightly compress the data and retain almost all information. Having a wide range of storage formats with varying levels of data allows applications to choose the right storage formats that fits their budget.

The critical feature of shareable storage formats is that they can use the same underlying storage. Achieving this requires diving deep into details of how images are stored. There are four main steps that storage formats perform when storing an image: (i) partitioning, (ii) transformation, (iii) pruning, and (iv) quantization[2].

Partitioning & Transformation. When an image, say of size 256x256, is stored, storage formats first partition them into a set of blocks of a specific size, e.g., 8x8 or 16x16. Then, they transform every block from its visually recognizable spatial domain into the so-called frequency domain. They use the discrete cosine transformation, a version of the Fourier transformation. Representing images in the frequency domain introduces a structure to the image data. Every value in the frequency domain is called a frequency coefficient and is responsible for carrying the weight of a specific signal with a particular frequency. As the signals' frequency increases, the coefficients' magnitude decreases. Therefore, low-frequency coefficients usually have a high value, whereas high-frequency coefficients usually have a low value.

Pruning & Quantization. Image data is pruned based on the frequencies. Recent work has shown that low-frequency coefficients are more important for image AI than high-frequency coefficients [16, 18]. Hence, image storage algorithms start pruning out coefficients with higher frequencies, as they are less valuable. The number of coefficients to prune depends on the application. The fourth step quantizes the remaining coefficients. Quantization refers to dividing each data value with a specific constant and rounding them to their nearest integer, which allows significantly reducing magnitudes of the data values and hence encoding them with a low number of bits.

Sharing Data. We observe that storage formats with the same block size, the same transformation, the same quantization policy, and different pruning strategies can share data. To illustrate, consider two storage formats. The first uses a 4x4 block size, keeping data values in the frequency domain using discrete cosine transformation, quantizing every value by a fixed quantization factor of 50, and pruning the 10 highest frequency coefficients. The second one uses the same block size, transformation, and quantization factor but pruning the 13 highest frequency coefficients. Since there is a strict ordering between frequencies of the pruned coefficients, the first storage format contains all data needed by the second storage format. Figure 3 illustrates this scenario for images of 8x8 size.

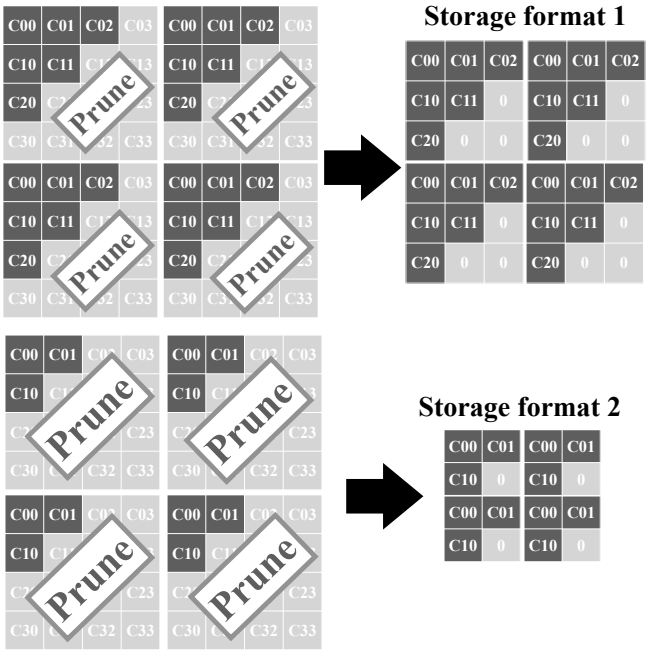

Figure 3: Storage formats with same block size, transformation, quantization policy, and different pruning strategies can share data. First storage format prunes 10 highest frequency coefficients, whereas, second storage format prunes 13 highest frequency coefficients. As first storage format contains all the coefficients that second storage format needs, they can share data.

Based on this observation, we create a set of shareable storage formats. We choose a fixed block size, transformation, and quantization factor. We then vary the pruning strategy and create a set of formats that prune data at different levels. As pruning only happens from highest to lowest frequency, storage formats in this set can share their data. The following sections describe how we choose a fixed block size, transformation, and quantization factor.

4.1 Choosing A Block Size & Transformation

Empirical Analysis. We use discrete cosine transformation, as other image storage formats do [4, 16, 18]. To choose the block size, we perform an empirical analysis across different block sizes: 8x8, 16x16, ..., 256x256. Each block size represents images differently in the frequency domain and provides a different model quality. We aim to find the block size that provides the highest model quality.

Block Size of 32 and 64 are Closest to the Pareto. We prune increasing number of coefficients for every block size and measure AI model quality for three quantization factors: 20, 50, and 100, as also used by [16]. We examine average accuracy loss for each block size across different number of coefficients and quantization factors. We compute the loss compared to Pareto-optimal accuracy over all block sizes and quantization factors. Figure 4 presents the results for three datasets, a 5-class subset of ImageNet, Weather,

[2]Sampling is another common operation. We exclude sampling, as a recent work has shown that it is the least effective operation for image storage [16].

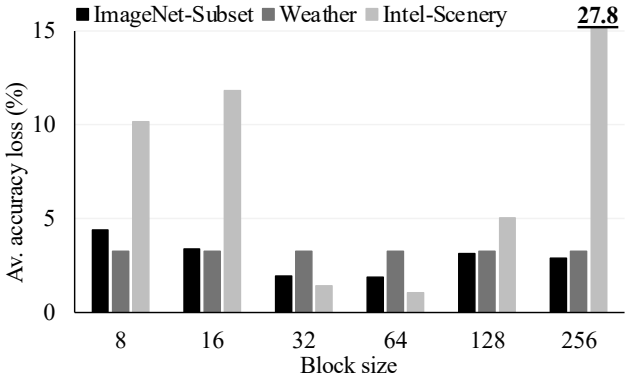

Figure 4: We chose 32 as the common block size for our shareable storage formats, as it has the lowest accuracy loss compared to the Pareto-optimal curve with about 2% difference and provides a more efficient columnar storage than block size of 64 with smaller number of diagonal columns (see Section 5.2.3 for details).

and Intel-Scenery datasets[3]. As the figure shows, block size of 32 and 64 has the lowest accuracy loss, less than 2% on average. We chose block size of 32 among the two block sizes, as it allows a more efficient columnar storage with smaller number of diagonal columns (see Section 5.2.3 for details).

4.2 Choosing A Quantization Factor

We performed a similar experiment to choose the quantization factor. Quantization factors reduce data size while also losing some model quality. We aim to choose a value that loses a small model quality but significantly reduces data size. We tested three quantization factors, as also done by [16]: 20, 50, and 100. We fix the block size to 32 and report average accuracy loss across different number of frequency coefficients. Figure 5 presents the results. As the figure shows, quantization factor of 50 provides the minimum average accuracy loss. We found out that it also provides a substantial reduction in data size (>50%). Hence, we chose quantization factor of 50 as the common quantization factor for our shareable storage formats.

5 COLUMNAR STORAGE

Every storage format in the shareable storage formats contains data that is either superset or subset of another storage format in the set. State-of-the-art storage formats store images file by file [4, 16, 18]. Therefore, even though shareable storage formats can share data, data is packed within image files and inaccessible to others.

Sharing Data Across Columns. We implement shareable storage formats by using columnar data organization. We break a batch of images into columns, where each storage format can read its data by only reading the necessary columns. This way, storage formats can share their data and read only what they need, bringing scalability and efficiency. The columnar organization further allows for storing similar valued data items together and, hence, efficient encoding.

[3]Using ResNet50 AI model. Please see Section 6.1 for more details.

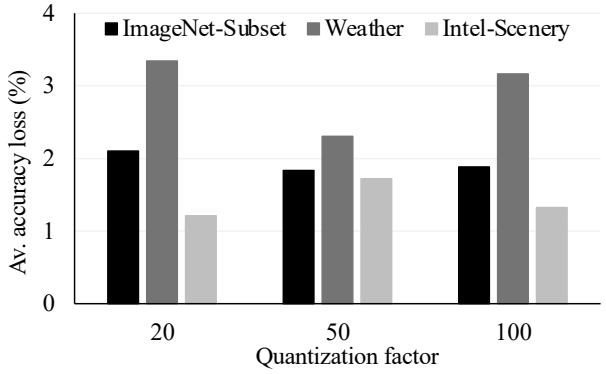

Figure 5: We chose 50 as the common quantization factor for our shareable storage formats, as it provides a good balance is accuracy loss (<2%) and reduced data size (>50%).

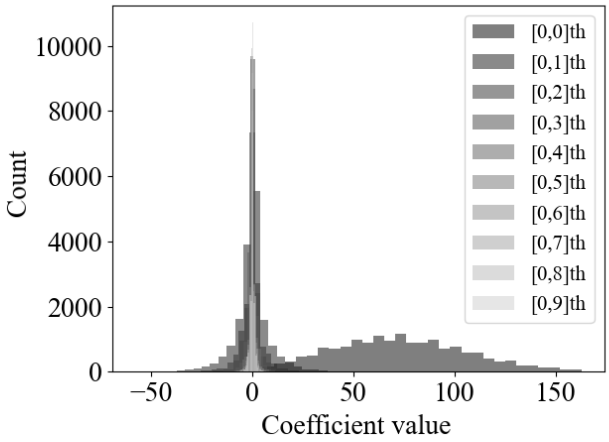

Figure 6: Image data contains values with different distributions and ranges.

5.1 Image Data Analysis

Images today are stored as integers. However, integers can have varying widths, ranging from 8 to 64 bits. Storing images file by file requires having a data width that is wide enough to cover all integers in the dataset. In Figure 6, we take a block of frequency coefficients (by using a block size of 32x32) across a thousand images of 256x256 size and examine the distribution of each coefficient[4].

Image Data is Multimodal. We observe that while some coefficients have a large mean and variance, others have a small mean with an increasingly smaller variance. Hence, using one data width for all images wastes a large number of bits. Storage formats can store coefficients using varying data widths based on their needs.

5.2 A Column Abstraction for Images

Structure within the Image Data. Relational data defines columns based on attributes of tables, which have a clear two-dimensional structure. We seek a similar structure in the image data. We observe

[4]We use a 5-class subset of ImageNet data.

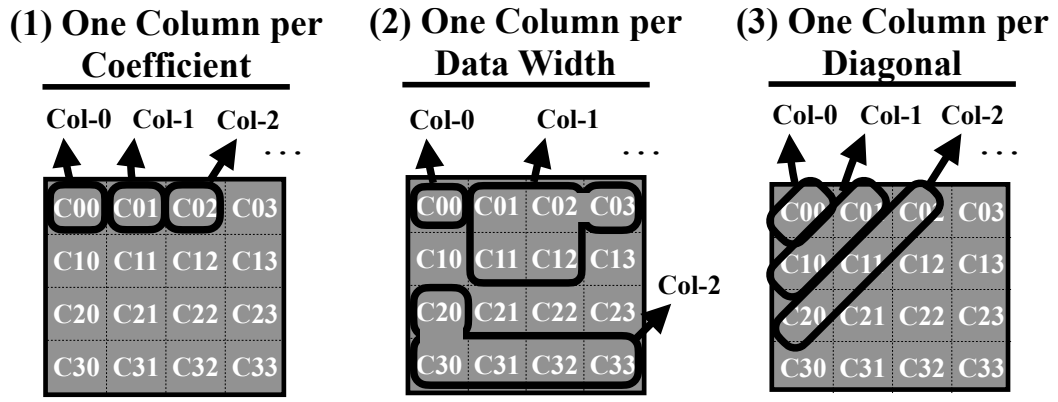

Figure 7: We analyzed three different column abstraction methods to efficiently reconstruct images by reading only data that is needed and efficiently encoding them. Figures show an example block of 4x4 in frequency domain. Columns with diagonals of coefficients proved to be best choice.

that blocks constitute the first level of structure in images. Each image is partitioned into a set of blocks when stored. An image of 256x256, for example, is partitioned into 64 32x32 blocks. Each block is transformed into the frequency domain separately and hence contains an independent unit of information. The second level of structure is frequency. Within each block, each value is a coefficient for a different frequency. Lower frequencies contain large values with high variance, whereas higher frequencies contain small values with low variance.

We use block and frequency-based structures in image data to define a column abstraction for images. Our goal is to have an abstraction that allows (i) efficiently sharing columns across applications and (ii) bringing similar-valued data items together to efficiently encode them. We examine three different column abstractions shown in Figure 7.

5.2.1 One Column per Coefficient. We first examined keeping one column for each coefficient, as shown by the left-hand side of Figure 7. For a block size of 32x32, this would mean 1024 columns. Each column contains data items from all blocks of all images in the batch. If there are 1000 images in the batch, the block size is 32x32, and the image size is 256x256, meaning each column includes 1000x64=64K data items, as there are 64 blocks in a single image.

Efficient Sharing and Encoding, but Too Many Data Files. This method allows efficient sharing across storage formats, as storage formats can independently access any coefficient they need. Furthermore, as shown in Section 5.1, coefficients have distinct ranges, allowing tight data representations. However, this method suffers from one major problem: a large number of data files. Data files create a fragmentation on the disk. Hence, having a large number of data files results in poor storage and data access efficiency.

5.2.2 One Column per Data Width. The next strategy combines different frequency coefficients into a single column to reduce the number of data files, as shown by the middle of Figure 7. We combine coefficients that we can encode with the same number of bits. This method impedes reading only the coefficients that a storage format needs, as now different coefficients will be bundled together. Our intuition is that if we encode coefficients so tightly with the minimal number of bits, the overhead of reading unnecessary coefficients could be paid off by efficient encoding. We create one column for all coefficients that we can encode with 2 bits, one for 3 bits, etc. We then bitpack each column into unsigned 8-bit integers.

Small Number of Data Files, but Inefficient Sharing and Encoding. We observe that this method encodes data worse than the previous method. We save arrays using Python's internal encoding method `savez_compressed`, which exploits repetitions in the data. Original image data contains highly repeated values, as shown in Figure 6. Bitpacking values into 8-bit unsigned integers damage this repetition, providing a worse compression ratio than using one column per coefficient. Hence, while reducing the number of data files, this strategy resulted in reading many unnecessary coefficients and high data movement costs.

5.2.3 One Column per Diagonal of Coefficients. Lastly, we examined keeping the diagonal of coefficients as one column, as shown by the right-hand side of Figure 7. Frequencies increase diagonally, from the upper-left corner to the lower-right corner, in a block of coefficients. The frequencies of [1,0]th and [0,1]st coefficients are the same. The frequencies of [2,0]th, [1,1]st, and [0,2]nd coefficients are also the same. We follow this structure and create one column per diagonal of coefficients.

Small Number of Data Files, Efficient Sharing and Encoding. One column per diagonal significantly reduces the number of data files, as it keeps one data file per diagonal of coefficients. It follows directly how storage formats reconstruct images. Storage formats prune a diagonal of coefficients when pruning. A storage format that keeps the ten lowest frequency coefficients keeps ten diagonals of coefficients from the upper-left to the lower-right corner of each block. Therefore, this column abstraction allows reading only coefficients that each storage format needs. Lastly, coefficients exhibit similar ranges and distributions along the diagonal. We can store each diagonal with 8- or 16-bit integers, allowing an efficient encoding with column-specific data widths. Hence, we define our column abstraction based on diagonals of coefficients.

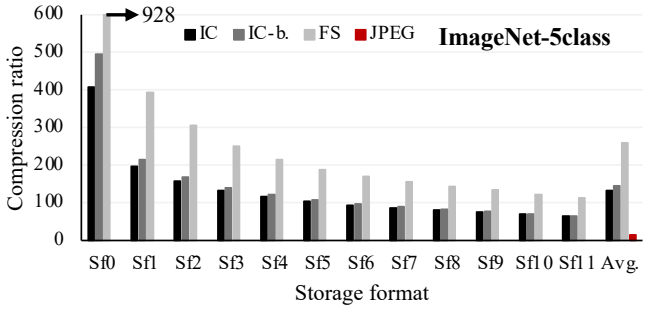

Figure 8: Frequency-Store (FS) improves the compression ratio by up to 2.2x thanks to storing similar-valued items together.

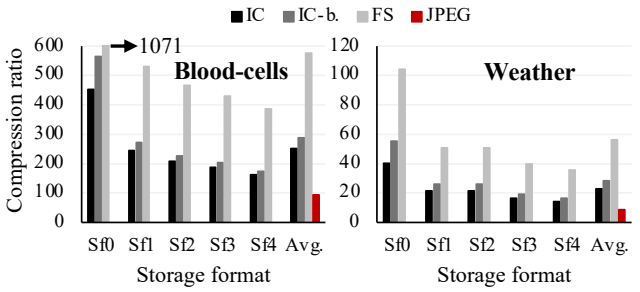

Figure 9: Frequency-Store (FS) improves the compression ratio by up to 2x for blood-cell images and weather datasets, similar to the ImageNet dataset.

6 EXPERIMENTS

We now move to evaluation. Our evaluation shows the following.

We show that Frequency-Store compresses the data by up to 2.2x higher than state-of-the-art image storage formats.
We show that Frequency-Store does not introduce any image reconstruction overhead compared to file-by-file image reconstruction.
We show that Frequency-Store improves end-to-end inference and training time by up to 11x when the bottleneck is the network and up to 4.75x when the bottleneck is storage.

6.1 Setup

Datasets. We use four image classification datasets: a 5-class subset of ImageNet data [10], blood-cell identification dataset [15], weather prediction dataset [2], and Intel scenery dataset [3].
AI Models. For Figure 1, 4, and 5, we use the popular convolutional neural network ResNet50 [12]. For all other experiments, we use the popular convolutional neural network MobileNet-V3 [14].
Training. We use PyTorch v2.0.0+cu117 with torchvision v0.15.1+cu117. The optimizer is stochastic gradient descent (SGD), which has a learning rate of 0.001 and a momentum of 0.9.
Hardware. We use a GPU server with 4 Nvidia A100 GPUs, each with 80GB of memory, an Intel Xeon CPU with 64 cores, and 512GB of main memory. The server has a network-attached storage with a 100Gbps InfiniBand connection.
Baselines. We use image calculator (IC), a recently proposed self-designing storage format for image AI applications [16] and JPEG [4] as the baselines.
Methodology. When reading the data from the disk, we drop the OS caches. We measure end-to-end inference/training time for the whole dataset for five minutes with multiple iterations and take the average over the number of images processed. We use a batch size of 256 when storing images as columns.

6.2 Results

6.2.1 Compression & Reconstruction Cost. As Frequency-Store combines similar-valued data items and uses column-specific data widths, it can efficiently compress the data. Figure 8 compares Frequency-Store (FS) with IC, JPEG, and batched version of IC (IC-b.) for the ImageNet dataset we use. IC-b saves a batch of image files as a single file, using the same batch size FS uses (256). We use it to isolate any improvement that would trivially come from merely batching the images[5]. We evaluate the same 12 storage formats for IC, IC-b, and FS with varying levels of prunings and report the average. This is possible as IC's space of storage formats subsumes that of FS. X-axis lists the twelve storage formats from Sf0 to Sf11, as well as average of the twelve results at the end. Y-axis presents the compression ratio.

FS Improves Compression Ratio by up to 2.2x. As the figure shows, FS has, on average, 1.8x higher compression ratio than IC and IC-b, ranging from 1.74x to 1.87x. FS stores the same amount of data as IC and IC-b and uses the same underlying encoding algorithm (Python's `savez_compressed`). Hence, improvement is purely due to storing data column by column rather than file by file. IC and IC-b are similar, which shows that just batching a set of files does not improve the compression ratio. JPEG's compression ratio, shown by the red column at the right-end of the figure, is dramatically low, 17x lower than FS since it is a fixed storage format without any specialization to the AI problem.

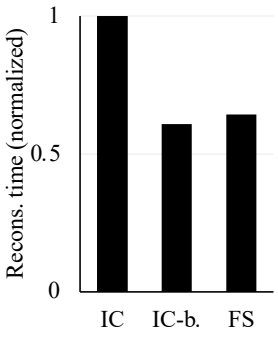

Figure 10: Despite a deeper compression level, FS does not introduce any reconstruction cost, as decompression works better over batches of files than over individual files.

Figure 9 performs a similar analysis using five storage formats with varying levels of prunings for the two other datasets we use: blood-cell images and weather dataset. We observe similar results. FS compresses data on average by 2x better than IC/IC-b, ranging from 1.9x to 2.2x savings. Once again, JPEG's compression ratio is dramatically lower than FS, by 6-6.5x, due to being a fixed storage format for all AI problems.

[5]JPEG follows a block-based encoding algorithm. Hence, batching JPEG files does not bring any benefit in terms of compression ratio

FS does not Introduce Any Reconstruction Cost. A higher compression ratio raises the question of whether a deeper compression level causes any decompression, i.e., reconstruction cost. Figure 10 presents normalized image-reconstruc- tion times for IC, IC-b, and FS for the ImageNet dataset we use, averaged across five storage formats with varying pruning levels. As the figure shows, IC-b and FS are about 40% faster than IC. Image reconstruction requires a costly CPU-decoding process, which is more efficient when the data is in batches than file by file.

6.2.2 End-to-end Inference/Training Time Reduction. Frequency-Store allows applications to share data. We consider 12 applications with different resource budgets sharing the same dataset (i) when applications access data remotely over the network and (ii) when data is on a fast network-attached storage with 100Gbps InfiniBand. We use the same 12 storage formats used in previous section, Section 6.2.1. We use the ImageNet-5class dataset.

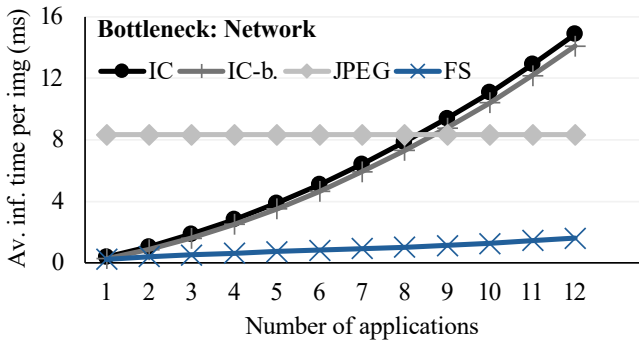

Figure 11: Frequency-Store scales much better than other storage formats and reduces inference time by up to 11x, thanks to reading only columns that applications need, sharing data across the applications, and compressing data better.

FS Improves Inference/Training Time by up to 11x when the Network is Bottleneck. Figure 11 shows average inference time per image when applications access data remotely over a network. X-axis presents number of applications sharing the data. We assume an average bandwidth of 50Mbps [13]. We measure GPU server time by profiling inference when data is in the main memory (/dev/shm). We add profiled server time to the estimated network time based on the assumed bandwidth to obtain the final inference time. Network time overwhelmingly dominates the inference time by consuming more than 95% of it. We assume perfect data sharing for JPEG and FS. They bring data once for the storage format with most data and share it with all the others.

As can be seen, FS's inference time scales much better than other storage formats with up to 11x reduced inference time, thanks to reading only columns that applications need, sharing data across the applications, and compressing data better. IC-b does not improve the compression ratio or provide any sharing, so its time is similar to IC. Despite its low compression ratio, JPEG is better than IC and IC-b for nine or more applications. This is because JPEG allows applications share the data and hence its high data movement cost pays off for large number of applications. For small number of applications, however, its inference time is significantly worse than all other storage formats.

We repeated a similar experiment for training time. Figure 12 (left) presents average training time per image when twelve applications sharing the data. As the figure shows FS improves training time by up to 9x, thanks to data sharing and efficiently compressing columns. Compared to inference, training has a higher GPU time, which is a small portion of the end-to-end inference time. Therefore, data-movement savings still contribute significantly to the overall results and bring up to 9x improvement. JPEG achieves a lower training time than IC and IC-b, thanks sharing data across all the twelve applications. Yet, JPEG's training time is 5x higher than FS, showing FS's efficient sharing and encoding mechanisms.

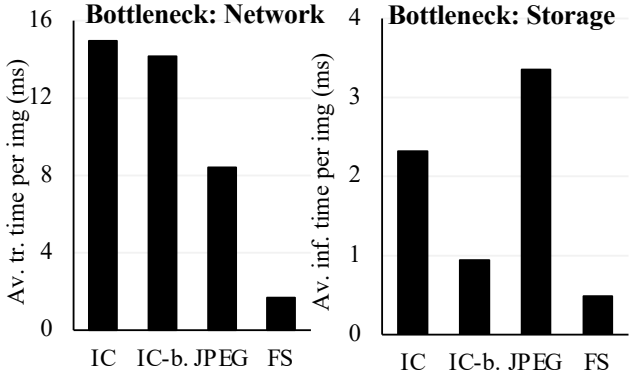

Figure 12: Frequency-Store reduces training time by up to 9x when the network is the bottleneck (left) and inference time by up to 4.75x when storage is the bottleneck (right), thanks to sharing data across the applications and compressing data better.

FS Improves Inference Time by up to 4.75x when Storage is Bottleneck. Figure 12 (right) presents average inference time per image when storage is the main bottleneck for twelve applications sharing the data. We observe that the batched storage format, IC-b is significantly more successful than its unbatched version, IC. The reason is that storing hundreds of thousands of image files one by one causes high disk fragmentation, resulting in a very inefficient disk access time. Frequency-Store improves all the baselines thanks to sharing data across applications and compressing data better. It is 4.75x faster than IC and 1.9x faster than IC-b. As storage access is faster than network access, Frequency-Store's savings are reduced compared to when the network is the bottleneck. JPEG is worse than IC since its other time components, such as GPU time, overshadow the savings due to sharing.

7 RELATED WORK

Application-Specific Storage for Image AI. Application-specific storage formats tailor storage format to the AI problem at hand. These studies range from self-designing storage formats [16] to learning a JPEG-like encoding [18]. The goal is to store as little data as possible to perform the AI task successfully enough. Our work builds on these studies, proposing a novel storage layout for efficiently sharing data across applications and better compression.

Learned Compression. Another line of work trains a neural network jointly optimizing the AI task and compression-ratio [8, 17]. These studies learn a data representation in main memory whose storage on disk is low. Our work could also extend these studies for more efficient storage access.

Column-Stores. Column-stores are one of the fundamental building blocks of modern database systems [1, 5, 6]. They inspired our work. We show that their motivations for relational data, such as reading only what you need and using tight data representations, also apply to image data.

8 CONCLUSION

This work presents Frequency-Store, the first column-store for images. Frequency-Store relies on the structure in image data and defines a novel column abstraction for storing images. It breaks a batch of image files into a set of columns and stores data column by column rather than file by file. Columns allow applications to share data efficiently and efficiently storing data by using tight data representations. We show that Frequency-Store improves image AI's inference/training time by up to 11x and compression ratio by up to 2.2x, compared to state-of-the-art image AI storage formats.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and members of the Data Systems Lab at Harvard for their constructive and insightful feedback. This work is partially funded by the Swiss National Science Foundation Postdoc Mobility scholarship P500PT_217934, the USA Department of Energy project DE-SC0020200, and a Sloan Research Fellowship.

REFERENCES

[1] Daniel Abadi, Peter A. Boncz, Stavros Harizopoulos, Stratos Idreos, and Samuel Madden. 2013. The Design and Implementation of Modern Column-Oriented Database Systems. *Found. Trends Databases* 5, 3 (2013), 197–280.

[2] Gbeminiyi Ajayi. 2018. Multi-class Weather Dataset for Image Classification . *Mendeley Data* 1 (2018). https://doi.org/10.17632/4drtyfjtfy.1

[3] Puneet Bansal. 2019. Intel Scenery Dataset. https://www.kaggle.com/datasets/puneet6060/intel-image-classification Accessed on May 16, 2023.

[4] Vasudev Bhaskaran and Konstantinos Konstantinides. 1995. *Image and Video Compression Standards*. Springer.

[5] Peter Boncz. 2002. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph. D. Dissertation. Universiteit van Amsterdam.

[6] Peter A. Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR*. 225–237.

[7] Matic Broz. 2024. How Many Pictures are There (2024): Statistics, Trends, and Forecasts. https://photutorial.com/photos-statistics/ Accessed on July 31, 2024.

[8] Lahiru D. Chamain, Siyu Qi, and Zhi Ding. 2022. End-to-End Image Classification and Compression With Variational Autoencoders. *IEEE Internet of Things Journal* 9, 21 (2022), 21916–21931.

[9] Chooch. 2023. How to use Computer Vision AI for Detecting Workplace Hazards? https://www.chooch.com/blog/computer-vision-ai-safety-technology-to-detect-workplace-hazards Accessed on Nov 14, 2024.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-scale Hierarchical Image Database. In *CVPR*. 248–255.

[11] Alice Gomstyn and Alexandra Jonker. 2023. What is Smart Farming? https://www.ibm.com/topics/smart-farming Accessed on Nov 14, 2024.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[13] Aleksander Hougen. 2024. What Is a Good Internet Speed in 2024 for Browsing, Streaming Online Gaming? https://www.cloudwards.net/good-internet-speed Accessed on July 31, 2024.

[14] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. 2019. Searching for MobileNetV3. In *ICCV*. 1314–1324.

[15] Paul Mooney. 2018. Blood Cell Images. https://www.kaggle.com/datasets/paultimothymooney/blood-cells Accessed on May 16, 2023.

[16] Utku Sirin and Stratos Idreos. 2024. The Image Calculator: 10x Faster Image-AI Inference by Replacing JPEG with Self-designing Storage Format. *Proc. ACM Manag. Data* 2, 1, Article 52 (2024).

[17] Róbert Torfason, Fabian Mentzer, Eiríkur Ágústsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. 2018. Towards Image Understanding from Deep Compression Without Decoding. In *ICLR*. 1–17.

[18] Kai Xu, Minghai Qin, Fei Sun, Yuhao Wang, Yen-Kuang Chen, and Fengbo Ren. 2020. Learning in the Frequency Domain. In *CVPR*. 1740–1749.

[19] Gang Yu, Kai Sun, Chao Xu, Xing-Hua Shi, Chong Wu, Ting Xie, Run-Qi Meng, Xiang-He Meng, Kuan-Song Wang, Hong-Mei Xiao, and Hong-Wen Deng. 2021. Accurate Recognition of Colorectal Cancer with Semi-supervised Deep Learning on Pathological Images. *Nature Communications* 12, 6311 (2021).