The Image Calculator: 10x Faster Image-AI Inference by Replacing JPEG with Self-designing Storage Format

UTKU SIRIN and STRATOS IDREOS, Harvard University, USA

Numerous applications today rely on artificial intelligence over images. Image AI is, however, extremely expensive. In particular, the inference cost of image AI dominates the end-to-end cost. We observe that the image storage format lies at the root of the problem. Images today are predominantly stored in JPEG format. JPEG is a storage format designed for the human eye; it maximally compresses images without distorting the components of an image that are visible to the human eye. However, our observation is that during image AI, images are "seen" by algorithms, not humans. In addition, every AI application is different regarding which data components of the images are the most relevant.

We present the Image Calculator, a self-designing image storage format that adapts to the given AI task, i.e., the specific neural network, the dataset, and the applications' specific accuracy, inference time, and storage requirements. Contrary to the state-of-the-art, the Image Calculator does not use a fixed storage format like JPEG. Instead, it designs and constructs a new storage format tailored to the context. It does so by constructing a massive design space of candidate storage formats from first principles, within which it searches efficiently using composite performance models (inference time, accuracy, storage). This way, it leverages the given AI task's unique characteristics to compress the data maximally. We evaluate the Image Calculator across a diverse set of data, image analysis tasks, AI models, and hardware. We show that the Image Calculator can generate image storage formats that reduce inference time by up to 14.2x and storage by up to 8.2x with a minimal loss in accuracy or gain, compared to JPEG and its state-of-the-art variants.

CCS Concepts: • **Information systems → Record storage alternatives**; • **Computing methodologies → Image representations**.

Additional Key Words and Phrases: Image Storage; AI; JPEG; AI Inference Time; Self-designing Systems

ACM Reference Format:
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 (SIGMOD), Article 52 (February 2024), 31 pages. https://doi.org/10.1145/3639307

1 TOWARDS EFFICIENT IMAGE AI STORAGE

Numerous and Diverse Applications Rely on Image AI. Image AI solutions create AI models for image analysis tasks, which learn from past behaviour and perform predictions for future behaviour. Today, hospitals use image AI to predict sepsis and to improve surgical plans. Governments use image AI to detect traffic violations. Farmers use AI to detect plant diseases and get a higher yield. Big technology companies use AI to build self-driving cars. The benefits come from automating tasks that are otherwise too complex, error-prone, and time-consuming for humans.

Problem: Image AI is Expensive. Images occupy vast space on persistent storage, consume large network bandwidth, and require significant processing power. At the same time, convolutional

Authors' address: Utku Sirin, utkusirin@seas.harvard.edu; Stratos Idreos, stratos@seas.harvard.edu, Harvard University, USA.

ACM 2836-6573/2024/2-ART52
https://doi.org/10.1145/3639307

neural networks (CNNs) and visual transformers [56, 139, 145], which are state-of-the-art image AI models, are computationally complex and expensive mathematical structures where successive layers of transformers/convolutions perform linear and non-linear transformations over the input image repeatedly.

Data Management Concepts to Speedup AI. There is an increasing trend in utilizing database management concepts in AI. For example, more than 50 papers have been published at SIGMOD and VLDB in the past four years in the areas of training pipeline optimization [1, 41–44, 68, 69, 71, 78, 84, 89, 90, 106, 107, 113, 113, 118, 120, 121, 123, 125, 126, 131–133, 136, 147, 151, 159, 164–166, 168–170], automated training data generation [10, 28, 117, 128, 146, 155], scalable inference time [38, 76, 106, 135, 157, 162], and video analytics using ML [6, 14, 26, 49, 58, 74, 75, 167]. Our work utilizes data management concepts in storage to speed up image AI inference.

Inference Cost Exceeds Training Cost at Scale. The AI lifecycle includes two parts: training and inference. During training, the AI model learns from past data. During inference, the trained model is used to analyze a new image. Since inference happens frequently, inference cost dramatically exceeds training cost at scale. For example, Amazon and Google report that the inference costs are nearly 90% of their customers' AI applications [4, 31, 39, 57].

Where Does the Inference Time Go? Inference time is typically defined by the execution time of the AI model over a single image [11, 13, 91, 92, 142, 148, 150, 156]. In Figure 1, we examine the end-to-end inference time of four state-of-the-art AI models, ResNet [51], EfficientNet [143], ShuffleNet [108], and MobileNet [56][1]. For this experiment, the server is an A100 GPU machine. The data resides on disk as JPEG files. The model resides on the GPU. Each inference call requires reading the data from disk to main memory, decoding it from the bytes stream into a visually recognizable image, transferring it from main memory to GPU, and executing the AI model on the GPU. The graph on the left-hand side presents the normalized number of floating point operations (flops) that require executing each model over an image, and the graph on the right-hand side presents the end-to-end inference time. The figure shows that, although models get smaller and smaller, the number of flops decreases by as much as 98%, the inference time remains the same. The reason is that reducing the number of flops reduces the model execution time, i.e., the GPU time, but not the other time components. When the GPU time is reduced, the other time components surface up, and the inference time remains the same[2]. In this way, inference time is a complex function of multiple time components, and reducing inference time requires reducing all time components.

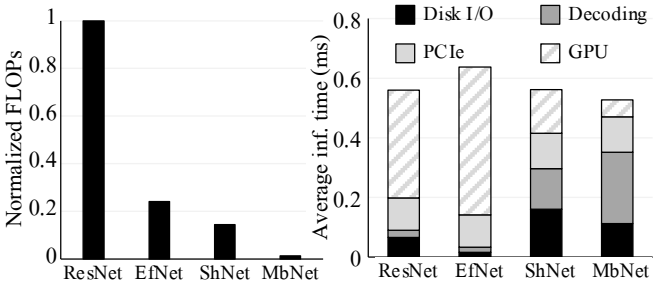


Fig. 1. Fast inference requires improving all cost parts.

Intuition 1: Storage is Key. We observe that all inference time components depend on one main factor: the amount of data moved/processed. This is determined by the storage format used for images. First, the storage format defines how we compress and store the data and how much data we read from the disk/network. Therefore, the storage format defines disk I/O and decoding times.

[1]Please see Section 5.1 for the details of the models.

[2]EfficientNet has a higher GPU time than ResNet, although it requires only a quarter of flops due to its more on expensive pointwise operations (ReLU & BatchNorm) [13, 142].

Second, PCIe time depends on how much data we transfer over the PCIe link, which depends on image size. Third, storage also determines the GPU time. The state-of-the-art AI models for images, i.e., CNNs and visual transformers, are mainly composed of transformers/convolutions, where inference time depends on the height and width, i.e., the spatial dimensions of the image.

Intuition 2: JPEG is for the Human Eye, Not Image AI. The vast majority of images today are stored in JPEG. JPEG lossily compresses images, where the lost information minimally distorts the quality of the image perceived by the human eye. We observe that Image AI tasks do not have to keep images in a human-recognizable format, as the images will be "seen" by neural networks. In addition, JPEG represents images in the human-recognizable red-green-blue (RGB) format in main memory. Hence, it suffers from constant decoding, PCIe, and GPU times, no matter how deeply it compresses images. Finally, every image AI application and task has unique characteristics regarding how much data needs to be maintained, how much storage/inference-time budget the use case has, and how much error the AI solution can tolerate.

Our thesis is that no single storage format is sufficient, and dramatically more efficient performance is only possible if the storage format is tailored for the given context: dataset, AI model, and the AI task.

The Solution: An Adaptive Storage Format with Scalable Image Representation. We introduce *the Image Calculator (IC)*, a self-designing storage format with scalable image representation in main memory. IC is a storage-format generator that produces new and tailored image storage formats. While current state-of-the-art storage formats are fixed and designed for a single purpose, such as minimally interfering with the human eye, IC is a storage-format generator that automatically produces a different storage format for every scenario. This way, it adapts the storage format to the given context and aims for maximal efficiency. IC takes the image dataset, AI model, hardware, and performance budgets as input and outputs the most efficient storage format for this AI task. IC achieves that by creating a rich design space of storage formats and efficient and accurate performance models that allow searching within this design space. IC represents images in a semi-compressed, non-visual format in main memory that scales inference time with data size. Finally, IC allows users to explore the design space interactively, i.e., perform what-if design questions to decide on the desired inference-time/storage/accuracy balance.

Contributions. In summary, our contributions are as follows.

- We introduce a design space of image storage formats. The design space is constructed by all combinations of fine-grained design decisions on how to store image data. Such decisions include subsampling strategy, block size, discrete-cosine transformation coefficient selection, and quantization factor.
- We show that this design space creates a massive set of practically infinite candidate image storage formats. We show how to dramatically reduce the infinite set into a valuable set of six thousand candidates by performing sensitivity analysis for each design decision and its domain and maintaining only the most performant designs (inference-time/storage/accuracy).
- We show how to search within the design space by introducing efficient performance models that take into account the balance of inference time, accuracy, and storage. Our performance models utilize sampling, interpolation, and transfer learning. We show that our method is 2.7-21.3x faster than brute force with 1-5% error.
- We evaluate the Image Calculator across diverse datasets, AI models, AI tasks, hardware, and baselines. We demonstrate that the Image Calculator can find storage formats with 2.8-8.2x reduced storage and 2.5-14.2x lower inference time than JPEG and state-of-the-art variants, proving the benefits of using an adaptive storage format tailored for the given context instead of a fixed design such as JPEG.

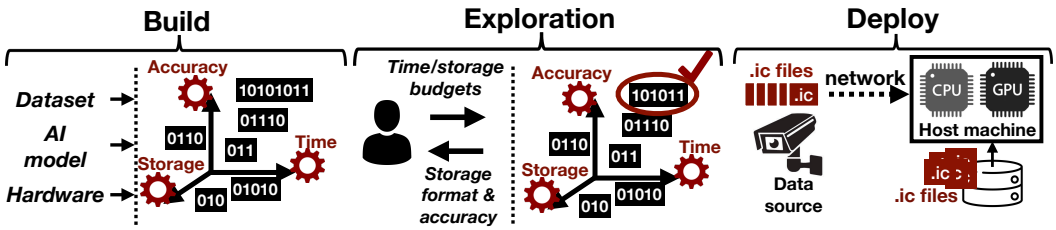


Fig. 2. The Image Calculator creates a tailored storage format given the AI model, the data, and the hardware.

The paper is curated to be self-contained with the most critical material, and we also accompany it with a technical report with numerous additional experiments and details [138].

2 BACKGROUND

All standard image/video storage formats, such as JPEG, JPEG2000, H.264/AVC, and H.265/HEVC, use Discrete-Cosine Transformation (DCT) [8]. DCT is a version of Fourier transformation, where the image is transformed from the pixel domain to the frequency domain. The frequency domain allows fine-grained control over the information in the image.

DCT is usually used for a specific block size. JPEG, for example, uses a fixed block size of 8x8. It, then, performs DCT over each block independently based on the following formula:

$$DCT[i,j] = \frac{2}{B}C(i)C(j)\sum_{k=0}^{B-1}\sum_{l=0}^{B-1} f(k,l) \cos\left(\frac{(2k+1)i\pi}{2B}\right)\cos\left(\frac{(2l+1)j\pi}{2B}\right) \quad (1)$$

where B is the block size, $f(k,l)$ is the pixel value at the $[k,l]$th position, and

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } x = 0 \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

This way, images are represented in the structured frequency domain rather than the unstructured pixel domain. Each value in a DCT-transformed block is called a frequency coefficient and represents how much specific frequency information contributes to the actual pixel values of that block. DCT is a reversible operation.

3 OVERVIEW

First, we present a high-level overview of the Image Calculator based on Figure 2. The Image Calculator has three execution modes.

Build: It takes as input the dataset, the AI model, and the hardware, and it constructs the performance models for the metrics of interest: accuracy, inference time, and storage.

Exploration: Users can interact with the Image Calculator in real time and explore the tradeoffs between the performance metrics. It takes as input the user's inference time and storage budgets and returns a storage format and accuracy. The accuracy represents how much the accuracy of the AI model would be if the dataset had been stored using the given storage format.

Deploy: The Image Calculator is deployed and used in real-time. It uses the storage format chosen in the exploration mode. It performs the final training of the AI model and moves it to the host machine for inference. It receives images at the data source, e.g., a camera, stores them in the

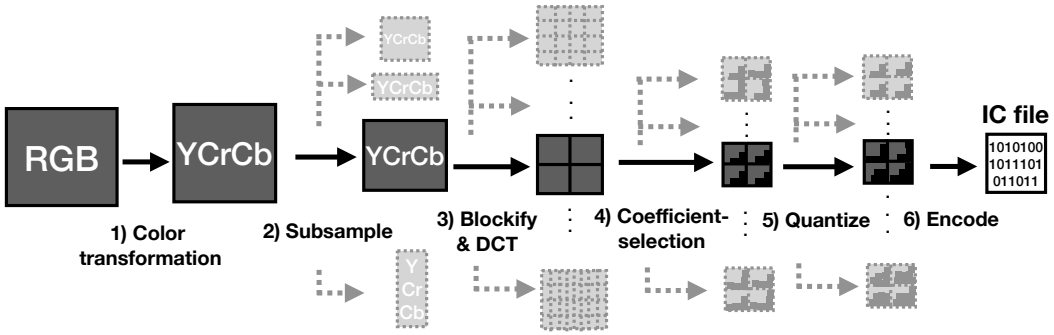


Fig. 3. The Image Calculator storage template generates entirely new image storage formats as combinations of fundamental design primitives.

chosen storage format, transfers them to the host machine, and performs the inference. Inference requires decoding the byte stream into an image representation in the main memory, transferring the decoded image to the GPU, and executing the AI model. The deploy mode might also run directly on the host machine if the data is stored on the local disk.

4 SELF-DESIGNING IMAGE AI STORAGE

The Image Calculator is based on a storage template, i.e., a set of design primitives. Each design decision has a domain. Every template instance is a different image storage format. To create a new instance, one has to give value to all primitives. We first describe the storage template at a high level (Sec. 4.1). Then, we describe each design primitive and their domains in detail (Sec. 4.2). Lastly, we describe how we search within the space (Sec. 4.3).

4.1 Storage Template

Figure 3 presents the storage template. It contains design primitives that we identified in state-of-the-art image/video storage formats and recent research variants, such as JPEG, JPEG2000, H.264/AVC, H.265/HEVC, PNG, and BMP [8, 141]. Below, we describe the primitives and how we combine them to form a storage template.

4.1.1 Design Primitives. The Image Calculator (IC) comprises six steps. There is one design primitive for steps (2), (3), (4), and (5), whereas we make fixed design choices for steps (1) and (6). IC first transforms the images from the RGB colour space to the luma-chroma (YCrCb) colour space. It then performs subsampling, splits the image into equal-sized blocks, and performs discrete-cosine transformation (DCT) over each block. Each DCT block contains a set of values called DCT coefficients. In step (4), IC performs the DCT-Coefficient-Selection: it identifies the most valuable coefficients and removes the rest. As we remove more DCT coefficients, the image gets smaller, allowing scalable image representation in the main memory. Section 4.2.3 discusses this in more detail. In step (5), IC quantizes the chosen coefficients by dividing them by a specific value and rounding the result into its nearest integer. At step (6), IC encodes the quantized coefficients into a byte stream.

4.1.2 Step (1) and (6). We make fixed design choices for steps (1) and (6), as they contribute very little to data size. Step (1) is merely a linear, colour-space transformation. Step (6) encodes the quantized DCT values into a byte stream using lossless compression such that the most frequently occurring values are encoded with the least number of bits. We use NumPy's lossless compression.

		Definition	Domain	A new design
Design primitives	Subsampling Strategy	It defines how to sample the images in YCrCb format.	{1x1, 2x2-r, 2x2-c, 4x4}	1x1
	Block Size (B)	It defines the granularity of the blocks to split the image and apply DCT.	{8x8, 16x16, 32x32, 64x64, 128x128, 256x256}	128x128
	DCT Coefficients	It defines which DCT coefficients to maintain and which DCT coefficients to remove from the dataset.	Top-left triangles of size {1x1, ..., BxB}.	Top-left 3x3 triangle.
	Quantization Factor	It defines the integral value that the chosen DCT coefficients are quantized with.	{20, 50, 100}	50

Table 1. The design space of the Image Calculator.

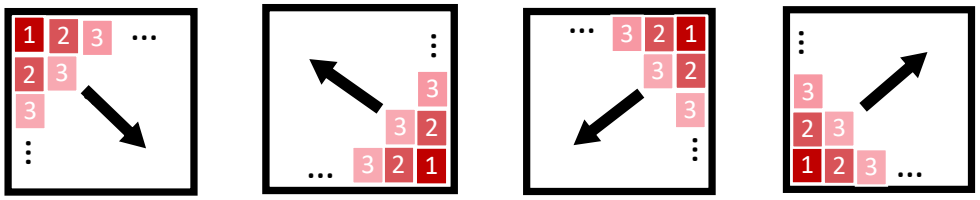
4.1.3 Infinitely Large Space. Having defined the design primitives, we now estimate the size of the design space. Each design primitive can take any value that satisfies the primitive. For example, we can subsample an image in every possible way to reduce the number of pixels in the image. Similarly, we can split the image into any size of blocks, choose any subset of the coefficients, and quantize each value with any possible value. This design space of combinatorial choices is practically infinite. More specifically, for an 8x8 block, there are 256^{8x8} possible quantization matrices, assuming that quantization factors are integers in $[1, 256]$. For 8x8, 16x16, ..., 256x256 block sizes, this number is $256^{8x8} + 256^{16x16} + \ldots + 256^{256x256}$, which is $> 10^{150K}$, making it impossible to search within this space. The following section describes how we reduce this infinitely large space into six thousand valuable candidates.

4.2 Reducing the Design Space Size

The domains of the primitives, when combined, are practically creating an infinitely large space, making searching within the design space impossible. In this section, we describe how we reduce the size of the space by investigating in detail the domain of each design primitive and the value it brings in end-to-end performance. The challenge is that every removed value of every design primitive potentially excludes useful storage formats. Hence, the goal here is to balance the domain size for each design primitive with a tractable search time and practical final storage formats. Our approach to solving this challenge is to perform sensitivity analysis for each design primitive and reduce their domains such that the overall size of the search space is manageable to search within. Our final design space size is reduced to six thousand storage formats. Table 1 summarises the design primitives and gives examples of generated storage formats. Now, we describe the design primitives and the sensitivity analysis for their reduced domains.

4.2.1 Subsampling Strategy. One of the Image Calculator's main decisions is how to sample the image. This decision has two aspects: which sampling strategy to use and which channels to sample. For the first aspect, we use four possible sampling strategies: (i) sampling every other row and every other column (4x4), (ii) sampling only every other row (2x2-r), and (iii) sampling only every other column (2x2-c), and (iv) no sampling (1x1).

Brightness and Color Information Are Equally Important for AI Models. For the second aspect, we perform the following analysis. We train an AI model by using only the Y, the Cr, and the



(a) Strategy 1: Increasing-sized upper-left triangle (b) Strategy 2: Increasing-sized lower-right triangle (c) Strategy 3: Increasing-sized upper-right triangle (d) Strategy 4: Increasing-sized lower-left triangle

Fig. 4. Four coefficient-selection strategies.

Cb channel, and we examine whether the accuracy is consistently higher for any of the channels. We used a state-of-the-art ResNet50 model and three image-classification datasets[3]. We observed that none of the channels consistently achieved a higher accuracy than the others. Hence, we conclude that all three channels are equally important. We subsample all the channels at the same rate. This way, we reduce the dimension of this domain into four. More details can be found in our technical report [138].

4.2.2 Block Size. Block size defines how we split an image into smaller-sized blocks. It allows for controlling lost information due to quantization and other design primitives.

Fixed Block Size Limits Capacity of the Design Space. As the Image Calculator targets a wide variety of AI tasks, using a single, fixed block size limits the representation capacity of the space of storage formats in terms of offering successful tradeoffs across its target metrics. Thus, we expand this choice into six possible block sizes: 8x8, 16x16, 32x32, 64x64, 128x128, and 256x256.

4.2.3 DCT Coefficients. AI models need only a certain amount of information to be successful. IC compresses the data by removing coefficients that are not useful for the given AI task.

Frequencies of the DCT Coefficients Increase Over The Spatial Dimensions. To identify what values are helpful for learning, we observe that DCT coefficients represent frequency components whose frequencies increase over the horizontal and vertical dimensions of the image. As we go from left to right in a block of DCT coefficients, the vertical frequencies of the DCT coefficients gradually increase. As we go from top to bottom in a block of DCT coefficients, the horizontal frequencies of the DCT coefficients gradually increase.

The DCT coefficient with the lowest horizontal and vertical frequency is stored in a block's $[0, 0]$th cell. The DCT coefficient with the highest vertical but lowest horizontal frequency is stored at the $[0, B-1]$th cell in a block of size BxB. Similarly, the DCT coefficient with the highest horizontal but lowest vertical frequency is stored at the $[B-1, 0]$th cell. Finally, the DCT coefficient with the highest horizontal and vertical frequency is stored in $[B-1, B-1]$th cell. Therefore, as we go from top-left to bottom-right in a block, both the vertical and horizontal frequencies increase. As we go from upper-right to bottom-left, the horizontal frequency increases while the vertical frequency decreases.

Low-Frequency Coefficients Are Much Useful Than High-Frequency Coefficients. For a block of DCT coefficients, we first choose an increasing-sized upper-left triangle of each block in the image as shown in Figure 4a. Then, we choose the increasing-sized lower-right triangle of each block in the image. Figure 4b depicts this strategy. Third and fourth, we choose the increasing-sized upper-right and lower-left triangles of each block in the image. Figure 4c and 4d depicts this

[3]Please see Section 5.1 for the details of the datasets and the AI model.

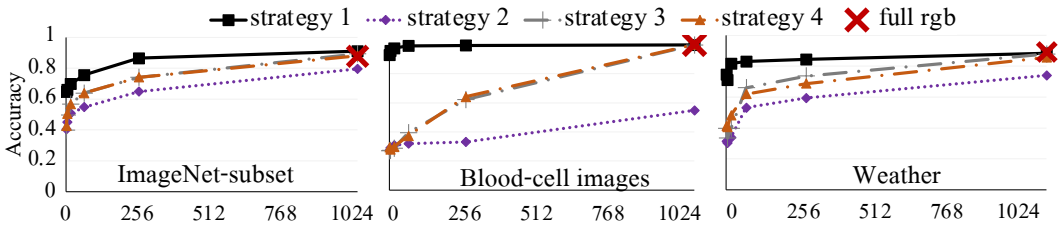


Fig. 5. Choosing increasing-sized upper-left triangles (strategy 1) provides a consistent advantage.

strategy. We set all the unchosen coefficients to zero and remove those coefficients outside the chosen triangle's boundaries. Figure 6 presents an example where we choose an upper-left 3x3 triangle for 32x32 blocks over a 256x256 image for a single channel. For each strategy, we analyze how the accuracy of the AI model changes as we gradually increase the size of the chosen triangle. We perform this with our three image classification datasets and the ResNet50 AI model. Figure 5 presents the results.

The figure shows that strategy 1 has a clear advantage over all other strategies for all datasets. Low horizontal and low vertical frequency coefficients are more valuable than the others for the AI models. Based on this result, we use this primitive to remove unuseful DCT coefficients from the data. For a block of BxB, we choose one of the upper-left triangles of size $1x1$, $2x2$, ..., BxB and eliminate the unchosen coefficients. The possible subsets of a BxB block of DCT coefficients are 2^{BxB}. By performing the sensitivity analysis above and identifying the most useful DCT coefficients for each block, we reduce this number to only B without sacrificing the representation capacity of the space of storage formats.

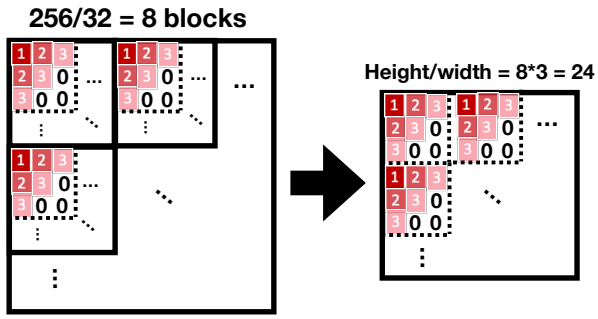
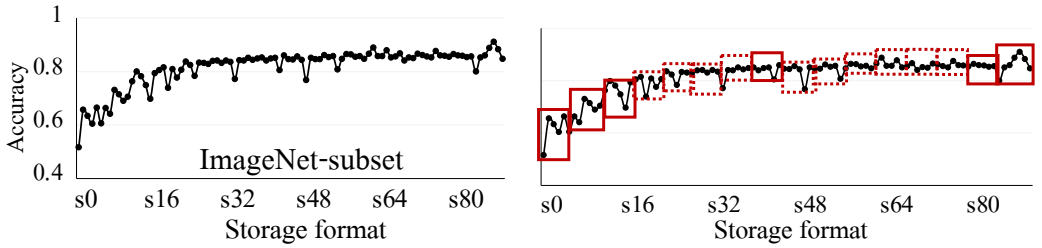


Fig. 6. IC removes DCT coefficients outside the boundary of the chosen triangle.

Removing DCT Coefficients Allows Scalable Representation of Images. Eliminating the unuseful DCT coefficients has two advantages. First, it allows for significantly reducing the data size, as we physically remove some values from the dataset. Secondly, it provides a scalable representation of images in the main memory. We eliminate the unuseful DCT coefficients outside the boundaries of the chosen triangle, as shown in Figure 6. As a result, the spatial dimensions of the image are reduced. For example, the image size in Figure 6 is reduced from 256x256 to 24x24 after we remove the DCT coefficients outside the chosen triangle. This significant reduction in terms of the spatial dimensions of the image allows for reducing the decoding, PCIe, and GPU times.

4.2.4 Quantization Factor. While removing unuseful DCT coefficients is an efficient means to reduce the data, more is necessary to compress the data by orders of magnitude. The reason is that the DCT coefficients are large floating point numbers. They require a large number of bits to encode (see Step (6) in Figure 3). Quantization allows for reducing the DCT coefficients' magnitude and encoding them as integers. It refers to dividing each value by a specific constant and rounding them into their nearest integers.



(a) The storage formats in our design space are highly correlated. X-axis: list of storage formats from the most to the least compressing one. Observation: The less a storage format compresses the data, the higher its accuracy.

(b) We utilize the relationship among the candidate storage formats by partitioning the design space into buckets, where each bucket contains similarly performing designs. Solid and dashed red lines show sampled and unsampled buckets.

Fig. 7. We discover and leverage the performance relationships among the candidate storage formats in the design space to efficiently build models that predict the accuracy.

We divide all DCT coefficients with the same value. We call this value the quantization factor. We vary it from 2 to 5, 10, 20, 50, and 100 and analyze how the accuracy changes as we increase the quantization factor. The larger the quantization factor is, the more data we save. We used our three image classification datasets and the ResNet50 model. We observed that the accuracy remains constant until 20 and increases at 50 and 100. Hence, we set the domain of this primitive to 20, 50, and 100. More details are in our technical report [138].

4.2.5 Size of the Design Space. Having covered each design primitive and domain, we now count the total storage formats in our reduced design space. The subsampling strategy primitive has four possible values. The block size and the DCT coefficients primitives have 8 + 16 + 32 + 64 + 128 + 256 = 504 possible values. The quantization factor primitive has three possible values. It makes 4 x 504 x 3 = 6048 storage formats. Thanks to our sensitivity analyses, we reduce the infinite design space into six thousand valuable storage formats.

4.2.6 Standard Storage Formats versus The Image Calculator. The Image Calculator (IC) is a generator of storage formats, whereas standard storage formats are existing fixed designs for minimizing interference with the human eye. To illustrate some of the differences between the formats in the design space of IC vs. JPEG, when IC eliminates coefficients, the reduced size of each block can be any value between 1 and B, B being the original block size. In Figure 6 in Section 4.2.3, the block size 32x32 is effectively reduced to 3x3 after the coefficient elimination. JPEG's quantization matrix could not be used for any such scenario, as it is of fixed size and rigid, whereas IC's quantization factor can be.

4.3 Efficient Search

Even with the reduced design space, if we did a brute-force search, we would need to train a new AI model from scratch for each storage format (so we know its accuracy). In this section, we describe how we efficiently search within this design space by building accurate performance models that predict the value of each performance metric for all candidate storage formats. The challenge is achieving high accuracy of the performance model, i.e., accurately predicting the value of each performance metric (accuracy, inference time, and storage), yet being efficient in avoiding training a new AI model from scratch every time.

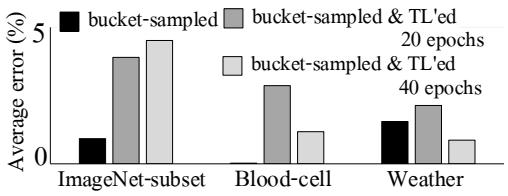


Fig. 8. Bucket sampling, interpolation, and transfer learning reduce the accuracy-model construction time by more than an order of magnitude with little error.

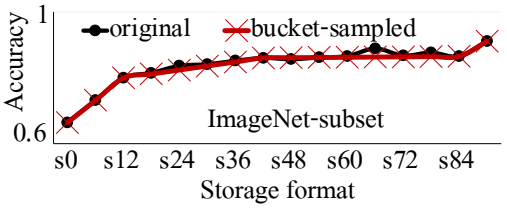


Fig. 9. Bucket sampling and interpolation provide strong results in building a highly accurate accuracy model. The sampled and interpolated accuracy values almost precisely match the original accuracy.

4.3.1 Accuracy Model. Our intuition towards building the models is that the storage formats in the design space are correlated. Consider the storage format that uses 16x16 blocks, the upper-left 4x4 triangle of each block, with no sampling and quantization factor of 2. This storage format stores the data only a little differently than the storage format that uses the same block size, same subsampling strategy, and same quantization factor but the upper-left 5x5 triangle. Thus, we first discover the relationship among the storage formats. We then explain how the Image Calculator exploits this relationship to build the accuracy model fast.

Storage Formats Have Diminishing Returns. We perform the following sensitivity analysis to understand the relationship between the storage formats in the design space. We fix the quantization factor to 1 and the subsampling strategy to no-sampling. We uniformly sample 96 storage formats with different block sizes and DCT coefficients. We train the ResNet50 model from scratch for each storage format and plot their accuracies in Figure 7a. As we go from left to right in Figure 7a, the storage formats compress the data less and less. The figure shows that, as the storage formats compress the data less and less, their accuracy values increase substantially in the beginning and then grow very slowly towards the end, i.e., have diminishing returns.

Exploiting Diminishing-Returns Reduces Model-Construction Time By 2.7x. The Image Calculator exploits this structure in the design space by using bucket sampling and interpolation. It first partitions the space of storage formats into equal-sized buckets. It then samples the critical buckets that define the shape of the accuracy curve and predicts the accuracy values of the others by interpolation. We use sixteen buckets, as it provides a reasonable estimation of the shape of the curve. Figure 7b presents the buckets, where the solid-line buckets are the sampled ones, and the dashed-line buckets are those that are not. The Image Calculator takes the maximum accuracy within each sampled bucket. It then interpolates the maximum accuracy of the unsampled buckets based on the maximum accuracy of the sampled buckets. For an unsampled bucket i, the interpolation between the sampled buckets j and k, where $j < k$, would be as in Equation 3:

$$acc_{bucket_i} = acc_{bucket_j} + \frac{acc_{bucket_k} - acc_{bucket_j}}{k - j} * (i - j) \tag{3}$$

By sampling buckets and interpolating, the Image Calculator predicts the accuracy values of the unsampled storage formats without actually training an AI model for them. In Figure 8, left-most bar, we quantify the error that bucket sampling introduces and show that it is less than 1%. In Figure 9, we compare the original and sampled accuracies line-by-line for the ImageNet subset and show that they are nearly identical. We obtained similar results for the Blood-cell and Weather datasets. More details are in our technical report [138]. Sampling and interpolation reduce the model-construction time by 16/6 = ~2.7x.

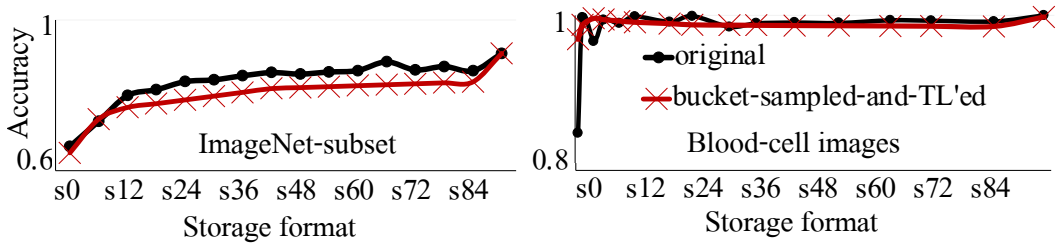


Fig. 10. The Image Calculator predicts accuracy values that closely match the actual accuracy.

Transfer Learning Together with Sampling Reduces the Construction Time by 12-21.3x. Transfer learning refers to training an already-trained AI model with a different dataset performing a similar task, such as image classification. It allows reaching a similar accuracy level with fewer epochs [11, 34, 46, 124]. In our case, the AI models trained with different storage formats use the same datasets with different amounts of information. We use transfer learning to accelerate the learning of the AI models for the sampled storage formats. We train the AI model from scratch only for the least-compressing storage format. We transfer the AI model from the least compressing to the most compressing format one after the other. We use 20 and 40 epochs of transfer learning.

Figure 8 quantifies how much error transfer-learning introduces compared to training all the sampled AI models from scratch for our three datasets and the ResNet50 model[4]. The figure shows that transfer learning produces less than 5% error for all cases. With 40 epochs of transfer learning, the accuracy values of the Blood-cell and Weather datasets almost precisely match the original accuracy values, as shown on the right side of Figure 10 (Weather graph omitted for brevity). The accuracy values of the ImageNet-subset also closely follow the original values by always being ~3-4% lower, as shown on the left side of Figure 10. More details are in our technical report [138].

We are inspired by the weight-sharing neural architecture search (NAS) studies in our idea of using transfer learning. Weight-sharing NAS studies try to find a neural network architecture by sharing the learned weights across different architectures. This way, the newly-tried neural architectures can quickly be examined whether providing a successful or unsuccessful result [11, 124]. In our work, we share the learned AI model among the storage formats to estimate what storage format provides how much accuracy.

By sharing a learned AI model across storage formats, the Image Calculator trains a single model from scratch and uses transfer learning for the rest. For 36 models and 40 epochs of transfer learning (210/40=~5x reduction in training time), this would make 1 + 35/5 = 8 models to train, 96/8 = 12x cheaper than the brute-force approach. With 20 epochs of transfer learning, this would make 1 + 35/10 = 4.5 model training, 96/4.5 = 21.3x cheaper than the brute-force approach. Transfer-learning duration is a user-specified parameter (budget-related). At 210 epochs, it would be equal to performing bucket sampling without transfer learning, where each storage format in the sampled buckets is trained from scratch. As we examined in the previous section, this would have a highly accurate accuracy model with less than 1% error. Once the Image Calculator constructs the accuracy model, the user can use it repeatedly for all possible use cases where performance budgets differ.

4.3.2 Inference Time & Storage Models. Training an AI model, even for a small-sized dataset, typically takes several hours with one GPU, whereas profiling a single storage format to obtain the average inference time or consumed space on disk takes only a few minutes. Hence, given the relative performance difference, which becomes even more significant for real-life large data sets,

[4]Please see Section 5.1 for the details of the datasets and the AI model.

the Image Calculator profiles on the fly to obtain efficiently the actual end-to-end inference time and the amount of on-disk storage that the various storage formats are expected to consume.

An alternative would be to build mathematical models that predict the behavior of each candidate storage format even faster. However, this is impossible in practice due to the hardware environment's variability in a cloud setup and the ever-changing collection of machines over time. In addition, as we have shown in previous research with the self-designing concept, it is highly error-prone to capture mathematical end-to-end performance, including both memory and disk, due to the vast number of components required in such models [18, 66, 77]. The disadvantage of building the time model by profiling the hardware device is that the user/application needs access to the hardware device before deployment. However, it is straightforward to do in a cloud environment.

4.4 Total Development Cost

In this section, we provide insights into the total development cost. As inference time and storage models are cheap to compute, the cost of accuracy-model construction dominates the total development cost. Thus, we provide the total development cost in terms of the number of AI model trainings required to construct the accuracy model. The brute-force approach would require training 6048 AI models, one for each candidate storage format in the design space. As shown in Section 4.3.1, we reduce this cost by 2.7-21.3x, depending on the number of epochs used for transfer learning. This results in the overall cost of 284-2240 AI model trainings.

Having constructed the accuracy model, users can then explore the accuracy-latency and accuracy-storage tradeoffs interactively in real-time and make an informed decision on what storage format to use (see the Exploration phase in Section 3 and Figure 2). To build the accuracy model efficiently and achieve interactivity, the Image Calculator partitions its design space into sixteen buckets. It performs transfer learning only for a sample of buckets enough to understand the relative behavior across all buckets. Suppose the users choose a storage format in one of the unsampled buckets at the end of the exploration phase. In that case, the Image Calculator performs one final set of transfer learnings for the chosen bucket to compare the storage formats within the bucket. This adds 38-378[5] additional AI model trainings to the overall cost, bounding the overall cost at 322-2618 AI-model trainings.

4.5 Reading Images

To read an image stored using an Image Calculator generated storage format, we need to perform the inverse of step (6). It decodes the byte stream into the quantized DCT coefficients and uses the DCT coefficients as is when learning and performing the inference. It significantly reduces the decoding time, as constructing the image ready to perform the inference takes only one step.

5 EXPERIMENTAL ANALYSIS

We now show that the Image Calculator improves inference time and required storage. Our evaluation consists of seven parts:

- We show that the Image Calculator successfully trades off accuracy for a 9.2x lower inference time and 8.2x lower storage than JPEG (and its state-of-the-art variants) across different AI models and datasets for the image classification task.
- We show that the Image Calculator can reduce individual inference-time components, such as the PCIe time, by up to 271x.

[5]The cost of transfer-learning for one bucket depends on the number of epochs used for transfer learning. Each bucket contains 6048/16=378 storage formats with sixteen buckets. For 210 epochs of transfer learning, which equals training a single AI model from scratch, the cost of transfer learning for one bucket is 378*1=378 AI-model trainings. For 20 epochs, which equals ~0.1 training, the cost of transfer-learning for one bucket is 378*0.1=~38. See Section 4.3.1 for more details.

- We show that these results scale with dataset size.
- We evaluate the Image Calculator on different hardware devices and show that it is even more successful on less powerful machines.
- We show that the Image Calculator works successfully on recently proposed visual transformers by achieving a lower inference time by 3.8x and higher accuracy by 2.5%.
- We show that the Image Calculator works well on further and diverse image analysis tasks such as object detection and segmentation by achieving up to 3.9x lower object detection time with no and 10% loss of mean-average precision at 50% and 75% thresholds.
- We show that the Image Calculator maintains a reasonable level of visual quality up to ~200x compression ratio, allowing the use of the Image Calculator for high-level visual inspections.

5.1 Setup & Methodology

Datasets. We use three image classification datasets. We scale all images to 256x256 for a more fair comparison. The first dataset is a five-class subset of the standard ImageNet dataset [29]. We chose three random subsets and observed that they all behaved similarly. We use one of them. Each class has ~1300 samples. The second dataset is the Blood-cell images. It is a medical imaging dataset including four blood cell types. It is for identifying blood-based diseases. Each class has ~2500 samples [116]. The third dataset is the Weather dataset. It is for predicting weather conditions based on image classification. There are four classes in the dataset. Each class has ~280 samples [2]. All the datasets are JPEG files. We use a 50-class subset of ImageNet and the full ImageNet datasets in Section 5.5 for the large dataset experiment. We use a cigarette-filter object detection dataset with ~2000 samples in total [72]. We use micro-controller detection and segmentation datasets [73] with 150 samples in total.

AI Models. We use three state-of-the-art convolutional neural networks and two visual transformers for the image-classification tasks: ResNet50 [51], EfficientNet-B3 [143], the small version of MobileNet-V3 [56], Swin Transformer [102], and MaxViT [145]. We vary the partition size for MaxViT from 2 to 4 and 8 and use the maximum partition size that works with the spatial dimensions of the given JPEG/IC image. We use the Faster R-CNN [130] model for the object detection task, and the Mask R-CNN [50] model for the object detection and segmentation task.

Training. For the image classification tasks, we use stochastic gradient descent (SGD) as the optimizer with a learning rate of 0.001, momentum of 0.9, and weight decay of $4e^{-5}$. We train models for 210 epochs when trained from scratch. We split each dataset into training and validation sets with an 80% to 20% ratio and report the validation accuracy. We use PyTorch (version 1.13.1+cu116) with torchvision (version 0.14.1+cu116). When training models over DCT coefficients of different block sizes, we reconstruct the image in main memory as red-green-blue (RGB) images and perform DCT by using the DCT implementation of the `torchjpeg` library [36, 37]. The library provides a GPU-based DCT implementation. We improved their implementation and integrated it into our main pipeline when constructing the accuracy model. We use 8 CPU worker threads with a prefetch factor of 2 when reading the data from the disk by PyTorch's DataLoader interface for all experiments unless otherwise stated. We chose this number as the CPU-to-GPU ratio in our main test server is 8. We tune the batch size for each model and use the highest-performing batch size for each. We use 32-bit integers when saving DCT coefficients and cast them to 32-bit floats when training models. We use RandomResizedCrop and RandomHorizontalFlip transformations over the RGB images as data augmentation in our training pipeline. We normalize the images after DCT with the mean and standard deviation of the used DCT coefficients. We use the same parameters as the classification tasks for the detection task. We use the same SGD except with a learning rate of 0.005 for the detection and segmentation task. We use the pre-trained Mask R-CNN model with the Microsoft COCO dataset and transfer-learn from it for 50 epochs using a batch size of 4.

Hardware. Our main test servers are GPU servers with Nvidia A100 or V100 GPUs. The A100 server has 4 GPUs, 80GB of memory, 64 Intel Xeon CPU cores (Platinum), 512GB of main memory, and 400GB of local SSD storage. The V100 server has 4 GPUs, 32GB of memory, 32 Intel Xeon CPU cores (Gold), 372GB of main memory, and 400GB of local SSD storage. We perform all inference experiments over a single GPU. We use A100 server for the experiments in Section 5.2, 5.3, 5.4, 5.5, and 5.7, while we use V100 for the rest of the experiments.
Baselines. We compare IC with JPEG and two state-of-the-art variants of JPEG: [45], [160, 161]. We also compare IC against other widely-used storage baselines such as numpy compressed (npz), GNU zip (gzip), and TFRecords.
Methodology. When reading the data from the disk, we pollute the caches. We warm up the execution for about a minute and perform the profiling for about three minutes. To provide the inference-time breakdown, we repeat the same experiment when the data is on /dev/shm, is already decoded in the main memory, and is already decoded and transferred in the GPU memory.
Feasible and Representative Sample of Design Space. We always use top-1 accuracy, except for the large dataset experiments in Section 5.5. We present the actual accuracy values rather than the predicted ones, as this is what users face at deployment. The accuracy model's accuracy is covered in Section 4.3.1. As training for six thousand storage candidates requires an overwhelming amount of GPU time and resources, we choose a subset of the six thousand candidates in the design space. We first observe that subsampling generally brings less benefit than coefficient selection. Hence, we fix the subsampling strategy to no-subsampling, i.e., $1x1$. We further observe that the quantization factor mostly contributes to reducing the amount of stored data rather than reducing the latency. Hence, we fix the quantization factor to its intermediate value: 50. Lastly, we observe that there is no specific pattern in which block size and the number of DCT coefficients are superior to the others. Hence, we uniformly sample the block size and DCT coefficients from their domains by 20%. Fixing the subsampling strategy and quantization factor values reduces the number of candidate storage formats to 6048/(4x3) = 504. Further sampling 20% for the block size and DCT coefficients reduces this number to 96, which is a feasible and well-representative sample of the complete design space to prove the benefits of the Image Calculator instead of a fixed design such as JPEG across a diverse set of datasets, AI tasks, hardware, and baselines. We show that, even with this sampled design space, we achieve strong results by up to 8.2x and 14.2x reduction in storage and inference time. It is possible to achieve even better results by using the complete design space.

5.2 Adapting Storage Format to the AI Task

This section shows the tradeoffs and flexibility the Image Calculator (IC) offers.
The Image Calculator Reduces Inference Time by 3-9.2x by Losing 1-5% accuracy. Figure 11 presents the accuracy-inference-time tradeoffs for three datasets and AI models and compares them with JPEG. IC can provide a similar level of accuracy to JPEG while significantly reducing inference time. By losing less than 1% of accuracy, IC can reduce inference time by 9.2x for the Blood-cell images over EfficientNet. Furthermore, IC allows a wide range of accuracy-inference-time tradeoffs from which different application requirements and performance budgets can benefit. For example, by losing 5% accuracy, IC can reduce inference time by 6.5x for the Weather dataset over MobileNet. By losing 3% accuracy, IC can reduce the inference time by 3x for the ImageNet subset over ResNet.
The Image Calculator Reduces Storage by 2.8-8.2x. Figure 12 presents the accuracy-storage tradeoff. We show the results only for ResNet, as the other models produce similar results. IC offers a wide range of accuracy-storage tradeoffs. For a 3% accuracy loss, IC reduces the storage by 6.5x for the ImageNet subset. For less than 1% accuracy loss and no-accuracy loss, IC can reduce storage by 4.8x and 2.8x for the Blood-cell and Weather datasets. By losing 5% accuracy, IC can reduce the storage by 8.2x for the Weather dataset.

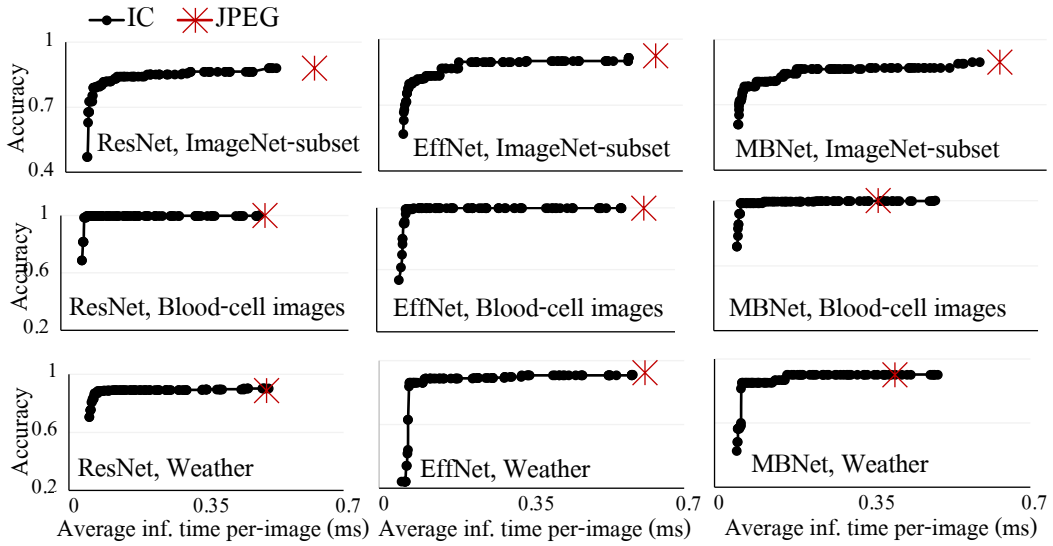


Fig. 11. The Image Calculator (IC) offers a wide range of accuracy-inference-time tradeoffs. It allows up to 9.2x lower inference time with 1-5% accuracy loss. As the user trades more and more accuracy, it achieves lower and lower inference time.

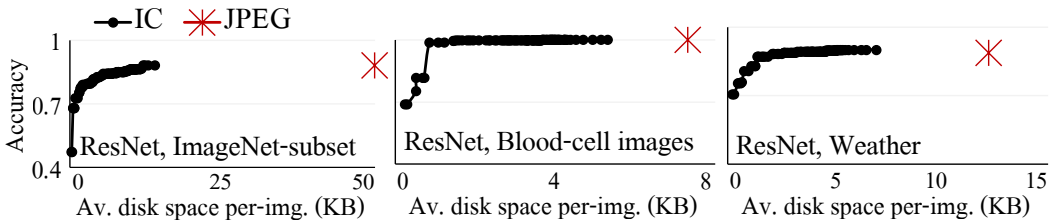


Fig. 12. The Image Calculator offers a wide range of accuracy-storage tradeoffs. It allows up to 6.5x reduced storage with 1-5% accuracy loss. As users trade more accuracy, they get lower storage.

5.3 Analyzing Individual Inference-Time Components

We now choose five specific scenarios and analyze where inference time goes. We chose two scenarios for the ImageNet subset with ResNet, one for the Blood-cell dataset with EfficientNet, and two for the Weather datasets with MobileNet. Figure 13 presents the results.

The Image Calculator Reduces Individual Inference-Time Components by up to 271x. IC reduces different inference-time components for different use cases at different rates. For the ImageNet subset dataset with the ResNet model, with 85% accuracy, it reduces the disk I/O time by 1.2x, CPU-decoding time by 2.5x, the PCIe time by 9.6x, and the GPU time by 2.5x, providing 3x overall improvement. Similarly, the Image Calculator reduces the CPU-decoding, PCIe, and GPU times by 17x, 271x, and 10x for 99% accuracy for the Blood-cell dataset and by 4.5x, 56x, and 7.3x for 85% accuracy for the Weather dataset.

The Image Calculator Provides Benefits For All Bottlenecks. Often, applications are bottlenecked by different inference-time components. Suppose the images are already decoded in the main memory buffers, and the GPU time is already low. IC can improve the inference time by

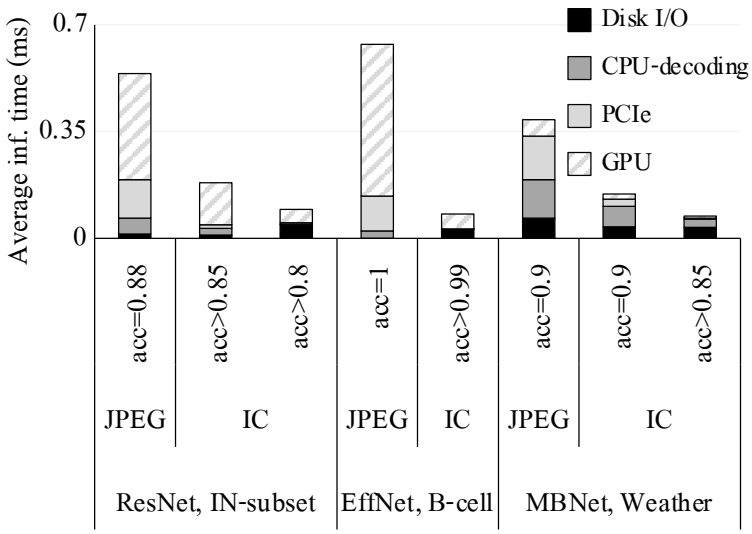


Fig. 13. As the Image Calculator reduces all inference time components, it helps reducing all bottlenecks. For example, it reduces the PCIe time by up to 271x, which can be the main bottleneck if the data is in memory buffers already decoded and GPU utilization is low.

up to 271x in this case, as PCIe is the main bottleneck. If the PCIe time decreases due to using a more advanced link, CPU-decoding time could be the bottleneck, which IC would improve the end-to-end inference time by up to 17x. Thanks to IC reducing the inference cost at its root by reducing the amount of data, its improvements apply to all inference-time components.

Data over Network. Network latency is the primary inference-time bottleneck when data is transferred over the network. We simulated this by Linux's `ping` command over five popular websites such as google.com and amazon.com. The average measured latency was ~38ms, which is way above the time it takes to perform the inference locally (<1ms). The more data we can pack in one network trip, the faster the inference is. It is directly proportional to the amount of data we send, which the Image Calculator significantly reduces, as shown in Figure 12.

5.4 Tuning JPEG

JPEG offers a tunable parameter, quality factor, which allows for reducing the visual quality for lower storage. We modify this parameter from its minimum value of 0 to its mid-range value of 70 to its maximum value of 100. This parameter is usually fixed and set at the data source for a particular visual quality[6].

Tuning JPEG Is Suboptimal. Figure 14 presents the accuracy-inference-time and accuracy-storage tradeoffs for IC, JPEG, and JPEG-tuned. Even JPEG's lowest quality factor has a very high inference time and misses many accuracy-storage tradeoffs that IC offers. Hence, tuning JPEG would only provide a suboptimal solution.

5.5 Scaling with Large Datasets

Image classification datasets have varying sizes from a few classes to hundreds of classes [29, 67]. This section presents how IC performs on large-sized datasets. We use a 50-class subset of ImageNet

[6]To illustrate, our used ImageNet, Blood-cell, and Weather datasets have quality factors of 96, 75, 80, on average.

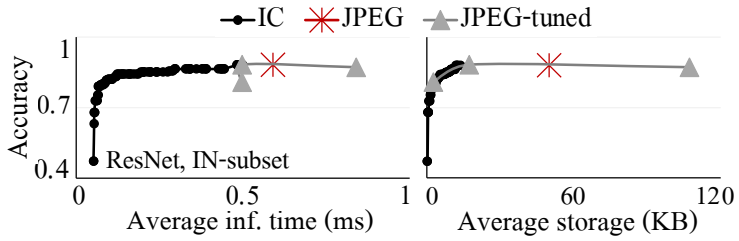


Fig. 14. The Image Calculator outperforms tuned JPEG.

and the complete ImageNet dataset with 1000 classes. We examine the accuracy-inference-time and accuracy-storage tradeoffs. We use top-5 accuracy instead of top-1 accuracy.

Powerful Accuracy/Performance Tradeoff Navigation. Figure 15 presents the results. IC allows for an informed and controlled tradeoff navigation for both class sizes. IC provides several possible tradeoffs that users can select depending on their application requirements. For example, two of the points in Figure 15 for the 50-class ImageNet are 1) achieving 2x inference speed up compared to JPEG while dropping 3.4% in accuracy, and 2) achieving 3.6x inference speed up compared to JPEG while dropping 4.6% in accuracy. For the full Imagenet, Figure 15 shows that IC achieves a similar tradeoff curve but this time with slightly lower accuracy compared to JPEG. In this way, IC provides a similar tradeoff curve regardless of class size, allowing users to make informed decisions based on the application needs in accuracy and inference time and balance their application costs, e.g., the cloud cost with accuracy.

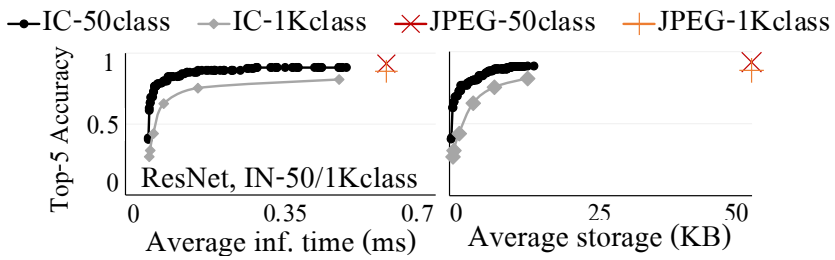


Fig. 15. The Image Calculator provides an informed and controlled tradeoff on small and large datasets to outperform JPEG.

5.6 Using Less Powerful Machines

Up to this section, we used A100 GPUs for all experiments. Inference today happens on a wide variety of hardware. This section shows that IC reduces the inference time for different computation devices. We use an Nvidia V100 GPU and an Intel Xeon Gold CPU at 2.6 GHz.

The Image Calculator Provides Even Higher Benefits for Smaller Hardware. Figure 16 presents the results for two specific use cases. IC reduces the inference time by 4.1x and 14.2x over JPEG on the Nvidia V100 GPU and the server CPU. These speedups were 3x and 6x on our central test server, which has a more powerful A100 GPU. As the computation capacity of the hardware gets smaller and smaller, our reduced computation and data size becomes more and more valuable. Hence, IC provides higher and higher benefits.

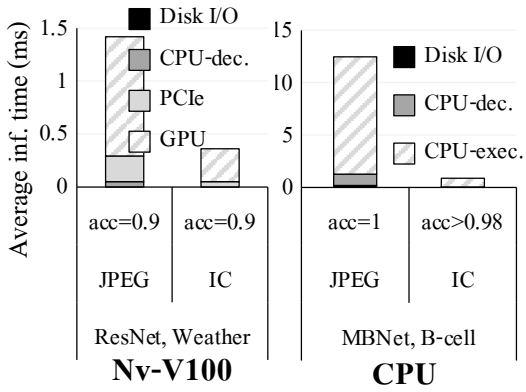


Fig. 16. The Image Calculator provides even higher speedups on less powerful hardware. Its speedup increases from 3x to 4.1x and 6x to 14.2x when we move from an A100 machine to a V100 and CPU.

5.7 Outperforming JPEG Variants

While JPEG was proposed 30 years ago, it is still the most widely used image storage format. Hence, modifying JPEG for AI tasks is an active research topic. This section compares our work with two promising studies.

5.7.1 Learning Straight On DCT [45]. This study performs the image classification directly over the DCT coefficients. Its goal is to reduce the decoding time. We use the authors' original codebase, profile their solution (the NumPy implementation and UpSampling-RFA version), and compare it with JPEG and our work. Figure 17 presents the inference time breakdown on the left-hand side.
Decoding Time is Increased; It is Data Representation That Reduces Time. The study only slightly improves JPEG and is much worse than the Image Calculator. First, the study does not reduce the PCIe time, as it does not change the image size in the main memory. Second, opposite to the study's primary goal, the decoding time is higher than that of JPEG. The study uses the libjpeg library over a Python interface, which has an overhead that exceeds the fast C++-based image decoding pipeline. The authors report a 1.9x improvement in inference time over JPEG. This is because the study represents DCT coefficients as 192x32x32 cuboids, which have low spatial dimensions and incur a low GPU time (as Figure 17 also shows). As GPU time was more dominant back in time than it is today, the authors saw a 1.9x improvement. The study does not provide a breakdown of their results. These results show that reducing inference time requires reducing all time components together.

5.7.2 Learning On A Subset of DCT [160, 161]. The second study represents images as 192x32x32 DCT coefficients, similar to [45]. It, however, uses only a subset of the DCT coefficients and learns which DCT coefficients are useful through a tunable parameter λ, which sets the balance between the accuracy and the number of DCT coefficients. The study fully reconstructs the RGB image in the main memory, performs DCT over it on CPUs, chooses the specific subset of the coefficients, and transfers only them to the GPU side. The authors aim to reduce the PCIe time only.
Increased Decoding Time Exceeds Reduced PCIe/GPU Time. We profile the authors' original codebase and compare it with JPEG and IC. We tune the λ parameter. We chose $\lambda = 0.75$, as it gave the highest reduction in DCT coefficients without significantly losing the accuracy (2% accuracy loss for 70% coefficient reduction). The right-hand side presents the results in Figure 17. We observe that the study suffers from a high decoding time. As the study performs

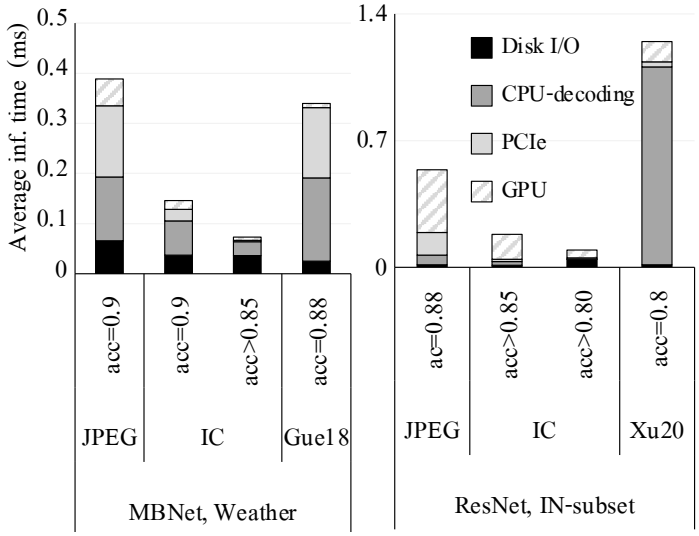


Fig. 17. The Image Calculator outperforms state-of-the-art variants of JPEG. Left-hand side: the study by Gueguen et al., 2018 [45]. Right-hand side: the study by Xu et al., 2020 [161]. The variants address only one of the inference time components. As a result, they are either slightly faster or even slower than JPEG. Both variants benefit from reduced GPU time.

the expensive DCT on CPUs, its decoding time is much higher than regular RGB-decoding time. It does improve the PCIe and the GPU times as expected and also shown by Figure 17, but the decoding time is so high that the savings from the PCIe and GPU times become negligible. These results again demonstrate the importance of reducing the end-to-end inference time for all time components together.

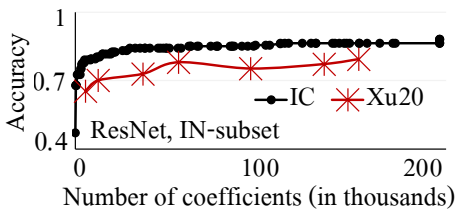


Fig. 18. The Image Calculator strongly outperforms Xu et al., 2020 [161] thanks to its higher spatial information and larger design space.

Lost Spatial Information and Small Design Space Produce Poor Results. Arguably, Xu et al., 2020's [161] is an adaptive technique that adapts the amount of data necessary to perform the inference. We tested this scenario by varying [161]'s λ parameter and examining the accuracy level and the number of DCT coefficients. We compare these results with the Image Calculator in Figure 18. IC strongly outperforms [161]. First, [161] represents images as transposed 192x32x32 cuboids, where much spatial information is lost. Second, IC relies on an ample and carefully designed design space, whereas [161] uses only JPEG's 8x8 blocks. Changing [161]'s data representation and expanding their design space could improve their results, but this requires re-designing their framework, which is outside the scope of our work.

5.8 Accelerating Transformers

Visual transformers (ViT) have gained significant attention in recent years [33]. This section examines the impact of IC on two popular ViT architectures: Swin Transformer [102] and MaxViT

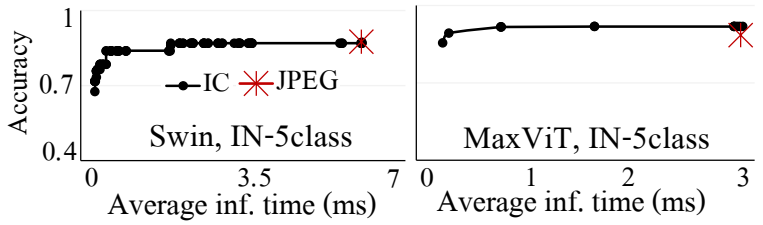


Fig. 19. The Image Calculator reduces inference time without losing any accuracy for Swin and provides even higher accuracy for MaxViT.

[145]. We use the 5-class subset of ImageNet and examine the accuracy-inference-time and accuracy-storage tradeoffs.

IC Achieves Higher or Same Accuracy and Faster Inference Time. Figure 19 presents the accuracy-inference-time trade-off. Again, IC achieves a complete tradeoff curve, allowing for informed decisions. In the case of transformers, we observe that there is, in fact, a net benefit. Specifically, for Swin, IC reduces the inference time by 3.1x compared to JPEG without losing accuracy. For MaxViT, IC achieves a lower inference time (3.8x) and a higher accuracy (2.5%). The benefits from an improved storage footprint are 8.2x for Swin and 6.7x for MaxViT. Our technical report includes more details [138].

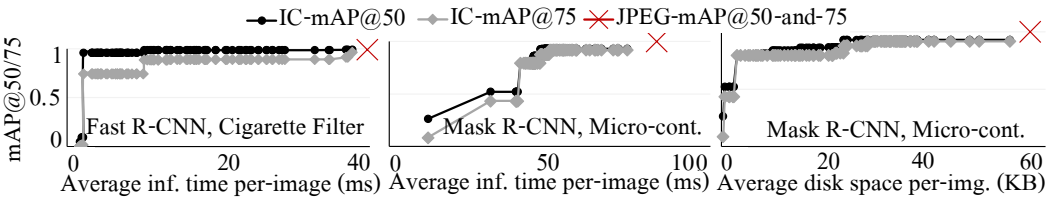


Fig. 20. The Image Calculator reduces inference time and storage by 1.8-3.9x for object detection and segmentation tasks by losing 0-10% mAP@50 and mAP@75.

5.9 Object Detection and Instance Segmentation

This section shows that the Image Calculator provides substantial performance benefits in image analysis tasks beyond classification, specifically object detection and instance segmentation.

Image Calculator Reduces Detection Time by 3.9x. We first analyze a pure object detection task, where the goal is to predict the boxes around cigarette filters [72]. We train a Faster R-CNN model [130] from scratch by using JPEG and the storage formats in our design space. Figure 20 (left) shows the mean average precision (mAP) at 50% and 75% intersection over union (IoU) thresholds versus the inference time trade-off. IC achieves the same mAP@50% as JPEG and 10% less mAP@75% than JPEG with 3.9x less time. Visual inspection of the resulting bounding boxes produced by IC and JPEG shows that they both detect the objects successfully. We performed the same study for the mAP-storage trade-off and observed similar trends. Our technical report includes more details on the visuals and mAP-storage trade-off [138].

Image Calculator Reduces Detection and Segmentation Time by 1.8x. Secondly, we analyze a joint object detection and instance segmentation task using the Mask R-CNN model [50]. We study

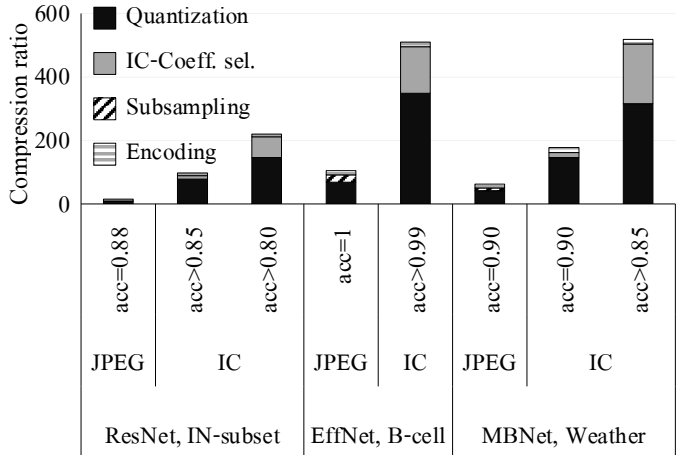


Fig. 21. Quantization and Coefficient Selection provide the highest benefits among the design primitives.

a micro-controller detection and segmentation task [73], where the goal is to predict the bounding boxes and segment the objects. We use a pre-trained model and transfer-learn, i.e., fine-tune the model using JPEG and IC storage formats. Figure 20 (middle and right) shows the mAP@50% and 75% versus inference time and storage tradeoffs for bounding box prediction. We leave the graphs for segmentation in our technical report, as they are similar to the bounding box results. IC can achieve 7% less mAP@50% and mAP@75% than JPEG with 1.8x less time and 2.5x less storage for detecting and segmenting the objects. Visual inspection of the resulting images shows successful results. Please see our technical report for the visuals [138].

5.10 Analysis of Design Primitives

This section examines the contribution of each design primitive to the overall compression ratio of IC. We incrementally turn on every design primitive and compare how much the compression ratio increases. Since IC contains numerous storage formats, we analyze the formats selected in Section 5.3. We combine the coefficient selection with the block size primitive, as they need to be tuned together.

Quantization and Coefficient Selection Provide the Highest Benefits. Figure 21 presents the results. Encoding represents the last stage of the IC pipeline, where we used a fixed lossless compression algorithm (see Section 4.1.2 and Figure 3). We start with only Encoding and gradually add Subsampling, IC-Coefficient-Selection, and Quantization. We perform a similar breakdown for JPEG without coefficient elimination and present the results side-by-side. As the figure shows, coefficient selection and quantization contribute the most to the overall compression ratio of IC. Similarly, for JPEG, quantization provides the highest benefit.

5.11 Conversion Costs Analysis

In this section, we study how the cost of converting a JPEG image into an IC image affects end-to-end costs. These conversion costs occur during training because data will naturally be in JPEG format when we build a new model.

We have two cases during inference: (1) First, high-performance model deployment is created with IC, and the end-points create new images directly in IC format to do inference. In this case, there are no conversion costs.

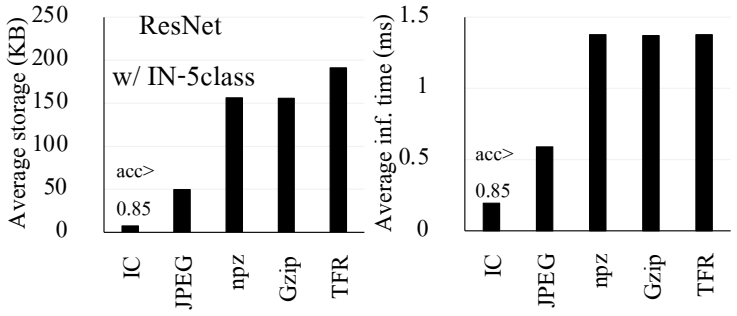


Fig. 22. The Image Calculator achieves 6.5-25x lower storage and 3-7x faster inference over alternative storage baselines to JPEG, such as numpy compressed and TFRecords.

(2) Second, JPEG hold-out data will be used for final model testing (inference) in successive iterations during development. These images can be cached in RGB and need to be converted to IC for iterations that test the network before the final deployment. We test this case below.

IC Conversion Cost is Small and Pays Off. The left side of Figure 23 shows the results for inference when RGB data needs to be converted to IC. IC improves the inference overall by 80% compared to using JPEG. While IC spends half of its time in the RGB-to-IC conversion, this pays off as it can perform the end-to-end inference dramatically faster in the IC format.

The right side of Figure 23 shows similar results for training. IC achieves a benefit with 83% faster training than JPEG. JPEG-to-IC conversion cost is only 23% of the IC's time; hence, there is a significant benefit from the conversion over using JPEG.

The reduced training time shows that IC could also improve the training time in various scenarios, such as hyper-parameter tuning, where there are repeated trainings with different hyper-parameters. As we identify the inference cost as the main bottleneck, we leave training-time optimizations as future work.

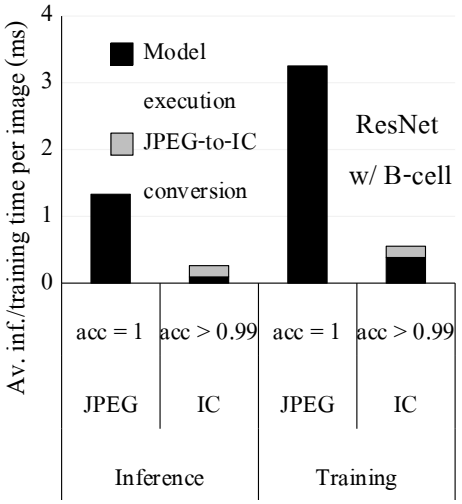


Fig. 23. JPEG-to-IC conversion brings dramatically lower end-to-end training and inference time.

5.12 Alternative Storage Baselines

This section examines alternative storage baselines, such as numpy compressed (npz), GNU zip (gzip), and TFRecords (using PyTorch's loader interface). Figure 22 presents the results for storage (left) and inference time (right). All baselines perform similarly, and IC successfully outperforms them by consuming ~20-24x less storage, providing ~7x faster inference.

5.13 Visual Quality of Image Calculator

While IC is designed for generating non-visual storage formats, it is possible to reconstruct an RGB image from an IC image by simply (i) extending the IC's main-memory representation to the

Fig. 24. IC maintains a reasonable level of visual quality until ~200x compression ratio. (The top-left is the original JPEG image; the others are reconstructed images of IC from least to most compressed.)

original size of the image, (ii) having zeros for all the eliminated coefficients, (iii) and performing the steps from (1) to (5) in Figure 3 in reverse order. This section examines the visual quality of such images.

The Image Calculator Maintains A Reasonable Level of Visual Quality until 200x Compression Ratio. Figure 24 presents the results from the highest (the original image, top-left corner) to the lowest visual quality (bottom-right corner). IC's compression ratios in the pictures are 59x, 103x, 197x, 489x, and 1135x. IC maintains relatively high visual quality until the compression ratio of ~200x. Thus, the storage formats IC generates can also be used for high-level visual inspections.

Additional Experiments. The paper is curated to be self-contained with the most critical experiments. Due to space constraints, this leaves out several experiments that could provide further insights. It includes (i) algorithmic analysis of discrete-cosine transformation and its effect on compression speed, (ii) analysis of a simulated networked system, (iii) reducing the number of worker threads, and (iv) top-1 accuracy-time/storage trade-offs for the large dataset, (v) further results on transformers, (vi) further results and visuals on the object detection and instance segmentation tasks. We include these results in the technical report [138].

6 RELATED WORK

Modifying JPEG for Image Classification. There are efforts to represent images using DCT coefficients. These studies range from simply using the DCT coefficients [45] to using a particular subset of the DCT coefficients [40, 104, 134, 160, 161]. Among those that use a subset of the DCT coefficients, [35, 40, 104, 134] use a fixed subset of coefficients, and [160, 161] learn which subset to choose. We compare IC with one study from each category. We show that IC outperforms both, as IC specializes in end-to-end storage format for the AI task and uses an ample and carefully designed design space.

Learned Compression. A large body of work on image compression for a specific vision task exists. These studies train a neural network jointly optimizing the vision task and the compression-ratio [3, 5, 9, 15–17, 20–24, 27, 30, 59, 60, 70, 83, 86, 95–98, 101, 105, 112, 122, 137, 140, 144, 149, 152–154, 158, 163]. These studies learn a data representation in main memory whose storage on disk is low. They do not target the inference time or propose an end-to-end storage format. They trade off accuracy with compression ratio through a parameter, similar to Xu et al., 2020 [161]. Hence,

they require learning a new representation for every tradeoff, which is very expensive to compute. Their representations are of fixed size in memory and hence suffer from JPEG's problem of high decoding, PCIe, and GPU times. Instead, IC creates a new custom storage format every time to match the context exactly and uses performance models to navigate the tradeoffs efficiently.

Network Compression/Platform-Aware Neural Architecture Search. There is a large body of work on reducing the size of the neural networks to reduce inference time [12, 25, 47, 48, 54, 55, 61, 85, 87, 93, 94, 99, 100, 103, 127, 171, 172]; and on platform-aware neural architecture search (NAS) that aims to find a neural network that fits into a budget [11, 13, 91, 92, 142, 148, 150, 156]. While these studies dramatically reduce the complexity of the AI model, their improvements are only within the GPU time and are therefore limited, as shown in Figure 1. The Image Calculator is complementary to this line of work and can be combined.

Self-Designing/Instance-Optimized Systems. There has been a growing interest in self-designing/instance-optimized systems for numerous problems such as data structure design [32, 63–66, 79–81, 88, 119], key-value storage engines [18, 19, 62], query optimization [110, 111], sorting [82], cloud storage format [109], and video analytics [7, 52, 53, 114, 115, 129]. Self-designing/instance-optimized systems automatically design, i.e., self-designs, themselves to maximally exploit the particular instance the system is surrounded with, e.g., the workload, hardware, data distribution, and time/quality requirements. Our work follows this and presents a self-designing storage format for image AI tasks.

7 SUMMARY

This paper introduces the Image Calculator, a self-designing storage format for image AI. The Image Calculator adapts itself to the given AI task to trade accuracy for faster inference and lower storage. It achieves this through an extensive design space of storage formats, within which it efficiently searches to balance inference time, accuracy, and storage. We evaluate the Image Calculator across various AI datasets, models, and hardware and show that it reduces storage by up to 8.2x and inference time by up to 14.2x, compared to JPEG and state-of-the-art variants.

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 Early Postdoc Mobility scholarship P2ELP2_199749 and by the USA Department of Energy project DE-SC0020200.

REFERENCES

[1] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J. Nair. 2021. Accelerating Recommendation System Training by Leveraging Popular Choices. *Proc. VLDB Endow.* 15, 1 (2021), 127–140.

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

[3] Baba Fakruddin Ali B H and Prakash Ramachandran. 2022. Compressive Domain Deep CNN for Image Classification and Performance Improvement Using Genetic Algorithm-Based Sensing Mask Learning. *Applied Sciences* 12, 14 (2022).

[4] AWS. 2019. Amazon EC2 Update – Inf1 Instances with AWS Inferentia Chips for High Performance Cost-Effective Inferencing. https://aws.amazon.com/blogs/aws/amazon-ec2-update-inf1-instances-with-aws-inferentia-chips-for-high-performance-cost-effective-inferencing/ Accessed on May 16, 2023.

[5] Yuanchao Bai, Xu Yang, Xianming Liu, Junjun Jiang, Yaowei Wang, Xiangyang Ji, and Wen Gao. 2022. Towards End-to-End Image Compression and Analysis with Transformers. In *AAAI*. 104–112.

[6] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *SIGMOD*. 1907–1921.

[7] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. 2020. MIRIS: Fast Object Track Queries in Video. In *SIGMOD*. 1907–1921.

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

[9] Imene Bouderbal, Abdenour Amamra, M. El-Arbi Djebbar, and Mohamed Akrem Benatia. 2022. Towards SSD Accelerating for Embedded Environments: A Compressive Sensing Based Approach. *Journal of Real-Time Image Processing* 19, 6 (2022), 1199–1210.

[10] Matthew Butrovich, Wan Shen Lim, Lin Ma, John Rollinson, William Zhang, Yu Xia, and Andrew Pavlo. 2022. Tastes Great! Less Filling! High Performance and Accurate Training Data Collection for Self-Driving Database Management Systems. In *SIGMOD*. 617–630.

[11] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *ICLR*.

[12] Han Cai, Ji Lin, Yujun Lin, Zhijian Liu, Haotian Tang, Hanrui Wang, Ligeng Zhu, and Song Han. 2022. Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications. *ACM Transactions Design Automation of Electronic Systems* 27, 3 (2022).

[13] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*.

[14] Jiashen Cao, Karan Sarkar, Ramyad Hadidi, Joy Arulraj, and Hyesoon Kim. 2022. FiGO: Fine-Grained Query Optimization in Video Analytics. In *SIGMOD*. 559–572.

[15] Lahiru D. Chamain, Siyu Qi, and Zhi Ding. 2021. An End-to-End Learning Architecture for Efficient Image Encoding and Deep Learning. In *29th European Signal Processing Conference (EUSIPCO)*. 691–695.

[16] 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.

[17] Lahiru D. Chamain, Fabien Racapé, Jean Bégaint, Akshay Pushparaja, and Simon Feltman. 2021. End-to-End Optimized Image Compression for Machines, A Study. In *31st Data Compression Conference (DCC)*. 163–172.

[18] Subarna Chatterjee, Meena Jagadeesan, Wilson Qin, and Stratos Idreos. 2022. Cosine: A Cloud-Cost Optimized Self-Designing Key-Value Storage Engine. *Proc. VLDB Endow.* 15, 1 (2022), 112–126.

[19] Subarna Chatterjee, Mark Pekala, Lev Kruglyak, and Stratos Idreos. 2024. Limousine: Blending Learned and Classical Indexes to Self-Design Larger-than-Memory Cloud Storage Engines. *Proc. ACM Manag. Data* (2024).

[20] Sien Chen, Jian Jin, Lili Meng, Weisi Lin, Zhuo Chen, Tsui-Shan Chang, Zhengguang Li, and Huaxiang Zhang. 2021. A New Image Codec Paradigm for Human and Machine Uses. *CoRR* abs/2112.10071 (2021).

[21] Hyomin Choi and Ivan V. Bajić. 2018. Deep Feature Compression for Collaborative Object Detection. In *25th IEEE International Conference on Image Processing (ICIP)*. 3743–3747.

[22] Hyomin Choi and Ivan V. Bajić. 2021. Latent-Space Scalability for Multi-Task Collaborative Intelligence. In *28th IEEE International Conference on Image Processing (ICIP)*. 3562–3566.

[23] Hyomin Choi and Ivan V. Bajić. 2022. Scalable Image Coding for Humans and Machines. *IEEE Transactions on Image Processing* 31, 1 (2022), 2739–2754.

[24] Hyomin Choi, Robert A. Cohen, and Ivan V. Bajić. 2020. Back-And-Forth Prediction for Deep Tensor Compression. In *45th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4467–4471.

[25] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *CoRR* abs/1805.06085 (2018).

[26] Pramod Chunduri, Jaeho Bang, Yao Lu, and Joy Arulraj. 2022. Zeus: Efficiently Localizing Actions in Videos Using Reinforcement Learning. In *SIGMOD*. 545–558.

[27] Felipe Codevilla, Jean Gabriel Simard, Ross Goroshin, and Chris Pal. 2021. Learned Image Compression for Machine Perception. *CoRR* abs/2111.02249 (2021).

[28] Nilaksh Das, Sanya Chaba, Renzhi Wu, Sakshi Gandhi, Duen Horng Chau, and Xu Chu. 2020. GOGGLES: Automatic Image Labeling with Affinity Coding. In *SIGMOD*. 1717–1732.

[29] 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.

[30] Yingpeng Deng and Lina J. Karam. 2021. Learning-based Compression for Material and Texture Recognition. *CoRR* abs/2104.10065 (2021).

[31] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. 2023. Trends in AI Inference Energy Consumption: Beyond the Performance-vs-Parameter Laws of Deep Learning. *Sustainable Computing: Informatics and Systems* 38 (2023), 100–857.

[32] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A Learned Multi-Dimensional Index for Correlated Data and Skewed Workloads. *Proc. VLDB Endow.* 14, 2 (2020), 74–86.

[33] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.

[34] Iddo Drori and Joaquin Vanschoren. 2021. AAAI 2021 Meta Learning Tutorial. https://sites.google.com/mit.edu/aaai2021metalearningtutorial Accessed on May 16, 2023.

[35] Lingyu Du and Guohao Lan. 2022. FreeGaze: Resource-efficient Gaze Estimation via Frequency Domain Contrastive Learning. *CoRR* abs/2209.06692 (2022).

[36] Max Ehrlich. 2020. TorchJPEG. https://torchjpeg.readthedocs.io/en/latest/ Accessed on May 16, 2023.

[37] Max Ehrlich, Larry Davis, Ser-Nam Lim, and Abhinav Shrivastava. 2020. Quantization Guided JPEG Artifact Correction. *ECCV* (2020).

[38] Jingzhi Fang, Yanyan Shen, Yue Wang, and Lei Chen. 2021. ETO: Accelerating Optimization of DNN Operators by High-Performance Tensor Program Reuse. *Proc. VLDB Endow.* 15, 2 (2021), 183–195.

[39] Forbes. 2019. Google Cloud Doubles Down On NVIDIA GPUs For Inference. https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference/ Accessed on May 16, 2023.

[40] Dan Fu and Gabriel Guimaraes. 2016. Using Compression to Speed Up Image Classification in Artificial Neural Networks. (2016). https://www.danfu.org/files/CompressionImageClassification.pdf

[41] Fangeheng Fu, Jiawei Jiang, Yingxia Shao, and Bin Cui. 2019. An Experimental Evaluation of Large Scale GBDT Systems. *Proc. VLDB Endow.* 12, 11 (2019), 1357–1370.

[42] Fangcheng Fu, Xupeng Miao, Jiawei Jiang, Huanran Xue, and Bin Cui. 2022. Towards Communication-Efficient Vertical Federated Learning Training via Cache-Enabled Local Updates. *Proc. VLDB Endow.* 15, 10 (2022), 2111–2120.

[43] Shay Gershtein, Tova Milo, Slava Novgorodov, and Kathy Razmadze. 2022. Classifier Construction Under Budget Constraints. In *SIGMOD*. 1160–1174.

[44] Stefan Grafberger, Paul Groth, and Sebastian Schelter. 2023. Automating and Optimizing Data-Centric What-If Analyses on Native Machine Learning Pipelines. *Proc. ACM Manag. Data* 1, 2 (2023).

[45] Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. 2018. Faster Neural Networks Straight from JPEG. In *NeurIPS*. 3937–3948.

[46] Isabelle Guyon, Jan N. van Rijn, Sébastien Treguer, and Joaquin Vanschoren (Eds.). 2021. *AAAI Workshop on Meta-Learning and MetaDL Challenge, MetaDL@AAAI 2021, virtual, February 9, 2021*. Proceedings of Machine Learning Research, Vol. 140. PMLR. https://proceedings.mlr.press/v140/

[47] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

[48] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *NuerIPS*. 1135–1143.

[49] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. VSS: A Storage System for Video Analytics. In *SIGMOD*. 685–696.

[50] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. 2017. Mask R-CNN. In *ICCV*. 2980–2988.

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

[52] Wenjia He, Michael R. Anderson, Maxwell Strome, and Michael Cafarella. 2020. A Method for Optimizing Opaque Filter Queries. In *SIGMOD*. 1257–1272.

[53] Wenjia He and Michael Cafarella. 2022. Controlled Intentional Degradation in Analytical Video Systems. In *SIGMOD*. 2105–2119.

[54] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In *ECCV*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). 815–832.

[55] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *ICCV*. 1398–1406.

[56] 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.

[57] HPCwire. 2019. AWS to Offer Nvidia's T4 GPUs for AI Inferencing. https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform/ Accessed on May 16, 2023.

[58] Bo Hu, Peizhen Guo, and Wenjun Hu. 2022. Video-Zilla: An Indexing Layer for Large-Scale Video Analytics. In *SIGMOD*. 1905–1919.

[59] Yueyu Hu, Shuai Yang, Wenhan Yang, Ling-Yu Duan, and Jiaying Liu. 2020. Towards Coding For Human And Machine Vision: A Scalable Image Coding Approach. In *21st IEEE International Conference on Multimedia and Expo (ICME)*. 1–6.

[60] Yueyu Hu, Wenhan Yang, Haofeng Huang, and Jiaying Liu. 2021. Revisit Visual Representation in Analytics Taxonomy: A Compression Perspective. *CoRR* abs/2106.08512 (2021).
[61] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks. In *NeurIPS*.
[62] Stratos Idreos, Niv Dayan, Wilson Qin, Mali Akmanalp, Sophie Hilgard, Andrew Ross, James Lennon, Varun Jain, Harshita Gupta, David Li, and Zichen Zhu. 2019. Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn. In *CIDR*.
[63] Stratos Idreos and Tim Kraska. 2019. From Auto-Tuning One Size Fits All to Self-Designed and Learned Data-Intensive Systems. In *SIGMOD*. 2054–2059.
[64] Stratos Idreos, Kostas Zoumpatianos, Manos Athanassoulis, Niv Dayan, Brian Hentschel, Michael S. Kester, Demi Guo, Lukas M. Maas, Wilson Qin, Abdul Wasay, and Yiyou Sun. 2018. The Periodic Table of Data Structures. *IEEE Data Eng. Bull.* 41, 3 (2018), 64–75.
[65] Stratos Idreos, Kostas Zoumpatianos, Subarna Chatterjee, Wilson Qin, Abdul Wasay, Brian Hentschel, Mike S. Kester, Niv Dayan, Demi Guo, Minseo Kang, and Yiyou Sun. 2019. Learning Data Structure Alchemy. *IEEE Data Eng. Bull.* 42, 2 (2019), 47–58.
[66] Stratos Idreos, Konstantinos Zoumpatianos, Brian Hentschel, Michael S. Kester, and Demi Guo. 2018. The Data Calculator: Data Structure Design and Cost Synthesis From First Principles, and Learned Cost Models. In *SIGMOD*. 535–550.
[67] iMerit. 2021. Top 13 Machine Learning Image Classification Datasets. https://imerit.net/blog/top-13-machine-learning-image-classification-datasets-all-pbm/ Accessed on Jan. 02, 2023.
[68] Alexander Isenko, Ruben Mayer, Jeffrey Jedele, and Hans-Arno Jacobsen. 2022. Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines. In *SIGMOD*. 1825–1839.
[69] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. 2021. Towards Demystifying Serverless Machine Learning Training. In *SIGMOD*. 857–871.
[70] Wenbin Jiao, Xuemin Cheng, Yao Hu, Qun Hao, and Hongsheng Bi. 2022. Image Recognition Based on Compressive Imaging and Optimal Feature Selection. *IEEE Photonics Journal* 14, 2 (2022), 1–12.
[71] Sian Jin, Chengming Zhang, Xintong Jiang, Yunhe Feng, Hui Guan, Guanpeng Li, Shuaiwen Leon Song, and Dingwen Tao. 2021. COMET: A Novel Memory-Efficient Deep Learning Training Framework by Using Error-Bounded Lossy Compression. *Proc. VLDB Endow.* 15, 4 (2021), 886–899.
[72] Kaggle. 2023. Cigarette Filter Detection Task. https://www.kaggle.com/datasets/estebanpacanchique/cigarette-butt Accessed on Sept. 20, 2023.
[73] Kaggle. 2023. Micro-controller Detection and Segmentation Task. https://www.kaggle.com/datasets/tannergi/microcontroller-segmentation Accessed on Sept. 20, 2023.
[74] Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. BlazeIt: Optimizing Declarative Aggregation and Limit Queries for Neural Network-Based Video Analytics. *Proc. VLDB Endow.* 13, 4 (2019), 533–546.
[75] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly Optimizing Preprocessing and Inference for DNN-Based Visual Analytics. *Proc. VLDB Endow.* 14, 2 (2020), 87–100.
[76] Barrie Kersbergen, Olivier Sprangers, and Sebastian Schelter. 2022. Serenade - Low-Latency Session-Based Recommendation in e-Commerce at Scale. In *SIGMOD*. 150–159.
[77] Michael S. Kester, Manos Athanassoulis, and Stratos Idreos. 2017. Access Path Selection in Main-Memory Optimized Data Systems: Should I Scan or Should I Probe?. In *SIGMOD*. 715–730.
[78] Alexandros Koliousis, Pijika Watcharapichat, Matthias Weidlich, Luo Mai, Paolo Costa, and Peter Pietzuch. 2019. Crossbow: Scaling Deep Learning with Small Batch Sizes on Multi-GPU Servers. *Proc. VLDB Endow.* 12, 11 (2019), 1399–1412.
[79] Tim Kraska. 2021. Towards Instance-Optimized Data Systems. *Proc. VLDB Endow.* 14, 12 (2021), 3222–3232.
[80] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H. Chi, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. 2019. SageDB: A Learned Database System. In *CIDR*.
[81] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD*. 489–504.
[82] Ani Kristo, Kapil Vaidya, Ugur Çetintemel, Sanchit Misra, and Tim Kraska. 2020. The Case for a Learned Sorting Algorithm. In *SIGMOD*. 1001–1016.
[83] Nikolina Kubiak and Simon Hadfield. 2021. TACTIC: Joint Rate-Distortion-Accuracy Optimisation for Low Bitrate Compression. *CoRR* abs/2109.10658 (2021).
[84] Andreas Kunft, Asterios Katsifodimos, Sebastian Schelter, Sebastian Breß, Tilmann Rabl, and Volker Markl. 2019. An Intermediate Representation for Optimizing Machine Learning Pipelines. *Proc. VLDB Endow.* 12, 11 (2019), 1553–1567.
[85] Andrew Lavin and Scott Gray. 2016. Fast Algorithms for Convolutional Neural Networks. In *CVPR*.

[86] Nam Le, Honglei Zhang, Francesco Cricri, Ramin Ghaznavi-Youvalari, and Esa Rahtu. 2021. Image Coding For Machines: An End-To-End Learned Approach. In *46th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1590–1594.

[87] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning Filters for Efficient ConvNets. In *ICLR*.

[88] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. 2020. LISA: A Learned Index Structure for Spatial Data. In *SIGMOD*. 2119–2133.

[89] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Jixiang Li, Ji Liu, Ce Zhang, and Bin Cui. 2022. Hyper-Tune: Towards Efficient Hyper-Parameter Tuning at Scale. *Proc. VLDB Endow.* 15, 6 (2022), 1256–1265.

[90] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, and Bin Cui. 2021. VolcanoML: Speeding up End-to-End AutoML via Scalable Search Space Decomposition. *Proc. VLDB Endow.* 14, 11 (2021), 2167–2176.

[91] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. 2021. Memory-efficient Patch-based Inference for Tiny Deep Learning. In *NeurIPS*. 2346–2358.

[92] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In *NeurIPS*.

[93] Ji Lin, Chuang Gan, and Song Han. 2019. Defensive Quantization: When Efficiency Meets Robustness. In *ICLR*.

[94] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime Neural Pruning. In *NeurIPS*.

[95] Jinming Liu, Heming Sun, and Jiro Katto. 2021. Learning in Compressed Domain for Faster Machine Vision Tasks. In *36th International Conference on Visual Communications and Image Processing (VCIP)*. 1–5.

[96] Jinming Liu, Heming Sun, and Jiro Katto. 2022. Improving Multiple Machine Vision Tasks in the Compressed Domain. In *26th International Conference on Pattern Recognition (ICPR)*. 331–337.

[97] Jinming Liu, Heming Sun, and Jiro Katto. 2022. Semantic Segmentation in Learned Compressed Domain. *CoRR* abs/2209.01355 (2022).

[98] Linfeng Liu, Tong Chen, Haojie Liu, Shiliang Pu, Li Wang, and Qiu Shen. 2022. 2C-Net: Integrate Image Compression and Classification via Deep Neural Network. *Multimedia Systems* (2022).

[99] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. 2018. Efficient Sparse-Winograd Convolutional Neural Networks. In *ICLR*.

[100] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning Efficient Convolutional Networks through Network Slimming. In *ICCV*. 2755–2763.

[101] Zihao Liu, Sicheng Li, Yen-kuang Chen, Tao Liu, Qi Liu, Xiaowei Xu, Yiyu Shi, and Wujie Wen. 2020. Orchestrating Medical Image Compression and Remote Segmentation Networks. In *23rd International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 406–416.

[102] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *ICCV*. IEEE, 9992–10002.

[103] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. 2019. MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning. In *ICCV*. 3295–3304.

[104] Shao-Yuan Lo and Hsueh-Ming Hang. 2019. Exploring Semantic Segmentation on the DCT Representation. In *1st ACM International Conference on Multimedia in Asia (MMASIA)*. 1–6.

[105] Guo Lu, Xingtong Ge, Tianxiong Zhong, Jing Geng, and Qiang Hu. 2022. Preprocessing Enhanced Image Compression for Machine Vision. *CoRR* abs/2206.05650 (2022).

[106] Shangyu Luo, Dimitrije Jankov, Binhang Yuan, and Chris Jermaine. 2021. Automatic Optimization of Matrix Implementations for Distributed Machine Learning and Linear Algebra. In *SIGMOD*. 1222–1234.

[107] Kaihao Ma, Xiao Yan, Zhenkun Cai, Yuzhen Huang, Yidi Wu, and James Cheng. 2023. FEC: Efficient Deep Recommendation Model Training with Flexible Embedding Communication. *Proc. ACM Manag. Data* 1, 2 (2023), 21 pages.

[108] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*. 122–138.

[109] Samuel Madden, Jialin Ding, Tim Kraska, Sivaprasad Sudhir, David Cohen, Timothy G. Mattson, and Nesime Tatbul. 2022. Self-Organizing Data Containers. In *CIDR*.

[110] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD*. 1275–1288.

[111] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *Proc. VLDB Endow.* 12, 11 (2019), 1705–1718.

[112] Yixin Mei, Fan Li, Li Li, and Zhu Li. 2021. Learn A Compression for Objection Detection - VAE With A Bridge. In *36th International Conference on Visual Communications and Image Processing (VCIP)*. 1–5.

[113] Xupeng Miao, Xiaonan Nie, Yingxia Shao, Zhi Yang, Jiawei Jiang, Lingxiao Ma, and Bin Cui. 2021. Heterogeneity-Aware Distributed Machine Learning Training via Partial Reduce. In *SIGMOD*. 2262–2270.

[114] Oscar Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. 2022. ExSample: Efficient Searches on Video Repositories through Adaptive Sampling. In *ICDE*.
[115] Oscar Moll, Manuel Favela, Samuel Madden, and Vijay Gadepally. 2022. SeeSaw: Interactive Ad-hoc Search Over Image Databases. *CoRR* abs/2208.06497 (2022).
[116] Paul Mooney. 2018. Blood Cell Images. https://www.kaggle.com/datasets/paultimothymooney/blood-cells Accessed on May 16, 2023.
[117] Supun Nakandala and Arun Kumar. 2022. Nautilus: An Optimized System for Deep Transfer Learning over Evolving Training Datasets. In *SIGMOD*. 506–520.
[118] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System for Optimized Deep Learning Model Selection. *Proc. VLDB Endow.* 13, 12 (2020), 2159–2173.
[119] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning Multi-Dimensional Indexes. In *SIGMOD*. 985–1000.
[120] Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. 2023. FlexMoE: Scaling Large-Scale Sparse Pre-Trained Model Training via Dynamic Device Placement. *Proc. ACM Manag. Data* 1, 1 (2023), 19 pages.
[121] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-End Optimization of Machine Learning Prediction Queries. In *SIGMOD*. 587–601.
[122] Neel Patwa, Nilesh Ahuja, Srinivasa Somayazulu, Omesh Tickoo, Srenivas Varadarajan, and Shashidhar Koolagudi. 2020. Semantic-Preserving Image Compression. In *27th IEEE International Conference on Image Processing (ICIP)*. 1281–1285.
[123] Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. 2022. Sancus: Staleness-Aware Communication-Avoiding Full-Graph Decentralized Training in Large-Scale Graph Neural Networks. *Proc. VLDB Endow.* 15, 9 (2022), 1937–1950.
[124] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *ICML*. 4095–4104.
[125] Arnab Phani, Lukas Erlbacher, and Matthias Boehm. 2022. UPLIFT: Parallelization Strategies for Feature Transformations in Machine Learning Workloads. *Proc. VLDB Endow.* 15, 11 (2022), 2929–2938.
[126] Arnab Phani, Benjamin Rath, and Matthias Boehm. 2021. LIMA: Fine-Grained Lineage Tracing and Reuse in Machine Learning Systems. In *SIGMOD*. 1426–1439.
[127] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *ECCV*. 525–542.
[128] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *Proc. VLDB Endow.* 11, 3 (2017), 269–282.
[129] Luis Remis and Chaunté W. Lacewell. 2021. Using VDMS to Index and Search 100M Images. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3240–3252.
[130] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*. 91–99.
[131] Alexander Renz-Wieland, Rainer Gemulla, Zoi Kaoudi, and Volker Markl. 2022. NuPS: A Parameter Server for Machine Learning with Non-Uniform Parameter Access. In *SIGMOD*. 481–495.
[132] Alexander Renz-Wieland, Rainer Gemulla, Steffen Zeuch, and Volker Markl. 2020. Dynamic Parameter Allocation in Parameter Servers. *Proc. VLDB Endow.* 13, 12 (2020), 1877–1890.
[133] Ties Robroek, Aaron Duane, Ehsan Yousefzadeh-Asl-Miandoab, and Pinar Tozun. 2023. Data Management and Visualization for Benchmarking Deep Learning Training Systems. In *DEEM Workshop*.
[134] Samuel Felipe dos Santos, Nicu Sebe, and Jurandy Almeida. 2020. The Good, The Bad, and The Ugly: Neural Networks Straight From JPEG. In *27th IEEE International Conference on Image Processing (ICIP)*. 1896–1900.
[135] Sebastian Schelter, Stefan Grafberger, and Ted Dunning. 2021. HedgeCut: Maintaining Randomised Trees for Low-Latency Machine Unlearning. In *SIGMOD*. 1545–1557.
[136] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. 2020. Learning to Validate the Predictions of Black Box Classifiers on Unseen Data. In *SIGMOD*. 1289–1299.
[137] Saurabh Singh, Sami Abu-El-Haija, Nick Johnston, Johannes Ballé, Abhinav Shrivastava, and George Toderici. 2020. End-to-End Learning of Compressible Features. In *27th IEEE International Conference on Image Processing (ICIP)*. 3349–3353.
[138] Utku Sirin and Stratos Idreos. 2024. The Image Calculator Technical Report. (2024).
[139] Samuel L. Smith, Andrew Brock, Leonard Berrada, and Soham De. 2023. ConvNets Match Vision Transformers at Scale. *CoRR* abs/2310.16764 (2023).
[140] Simeng Sun, Tianyu He, and Zhibo Chen. 2021. Semantic Structured Image Coding Framework for Multiple Intelligent Applications. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 9 (2021), 3631–3642.

[141] Peter Symes. 1998. *Video Compression: Fundamental Compression Techniques and an Overview of the JPEG and MPEG Compression Systems.* McGraw-Hill.

[142] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*. 2820–2828.

[143] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*. 6105–6114.

[144] 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.

[145] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan C. Bovik, and Yinxiao Li. 2022. MaxViT: Multi-axis Vision Transformer. In *ECCV*. 459–479.

[146] Paroma Varma and Christopher Ré. 2018. Snuba: Automating Weak Supervision to Label Training Data. *Proc. VLDB Endow.* 12, 3 (2018), 223–236.

[147] Xinchen Wan, Kaiqiang Xu, Xudong Liao, Yilun Jin, Kai Chen, and Xin Jin. 2023. Scalable and Efficient Full-Graph GNN Training for Large Graphs. *Proc. ACM Manag. Data* 1, 2 (2023), 23 pages.

[148] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *CVPR*. 8612–8620.

[149] Shurun Wang, Zhao Wang, Shiqi Wang, and Yan Ye. 2021. End-to-End Compression Towards Machine Vision: Network Architecture Design and Optimization. *IEEE Open Journal of Circuits and Systems* 2, 1 (2021), 675–685.

[150] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020. APQ: Joint Search for Network Architecture, Pruning and Quantization Policy. In *CVPR*. 2075–2084.

[151] Zeke Wang, Kaan Kara, Hantian Zhang, Gustavo Alonso, Onur Mutlu, and Ce Zhang. 2019. Accelerating Generalized Linear Models with MLWeaving: A One-Size-Fits-All System for Any-Precision Learning. *Proc. VLDB Endow.* 12, 7 (2019), 807–821.

[152] Zixi Wang, Fan Li, Jing Xu, and Pamela C. Cosman. 2022. Human–Machine Interaction-Oriented Image Coding for Resource-Constrained Visual Monitoring in IoT. *IEEE Internet of Things Journal* 9, 17 (2022), 16181–16195.

[153] Zhenzhen Wang, Minghai Qin, and Yen-Kuang Chen. 2022. Learning From the CNN-based Compressed Domain. In *22nd IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 4000–4008.

[154] Maurice Weber, Cedric Renggli, Helmut Grabner, and Ce Zhang. 2020. Observer Dependent Lossy Image Compression. In *8th DAGM German Conference on Pattern Recognition (GCPR)*. 130–144.

[155] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Cetintemel, and Carsten Binnig. 2020. DBPal: A Fully Pluggable NL2SQL Training Pipeline. In *SIGMOD*. 2347–2361.

[156] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *CVPR*. 10734–10742.

[157] Yuncheng Wu, Tien Tuan Anh Dinh, Guoyu Hu, Meihui Zhang, Yeow Meng Chee, and Beng Chin Ooi. 2022. Serverless Data Science - Are We There Yet? A Case Study of Model Serving. In *SIGMOD*. 1866–1875.

[158] Jiuhong Xiao, Lavisha Aggarwal, Prithviraj Banerjee, Manoj Aggarwal, and Gerard Medioni. 2022. Identity Preserving Loss for Learned Image Compression. In *CVPR Workshop on New Trends in Image Restoration and Enhancement and Challenges*. 517–526.

[159] Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, and Aditya Parameswaran. 2018. HELIX: Holistic Optimization for Accelerating Iterative Machine Learning. *Proc. VLDB Endow.* 12, 4 (2018), 446–460.

[160] Kai Xu. 2021. *Learning in Compressed Domains.* Ph. D. Dissertation. Arizona State University.

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

[162] Zhiqiang Xu, Dong Li, Weijie Zhao, Xing Shen, Tianbo Huang, Xiaoyun Li, and Ping Li. 2021. Agile and Accurate CTR Prediction Model Training for Massive-Scale Online Advertising Systems. In *SIGMOD*. 2404–2409.

[163] Shuai Yang, Yueyu Hu, Wenhan Yang, Ling-Yu Duan, and Jiaying Liu. 2021. Towards Coding for Human and Machine Vision: Scalable Face Image Coding. *IEEE Transactions on Multimedia* 23, 1 (2021), 2957–2971.

[164] Gyeong-In Yu, Saeed Amizadeh, Sehoon Kim, Artidoro Pagnoni, Ce Zhang, Byung-Gon Chun, Markus Weimer, and Matteo Interlandi. 2021. WindTunnel: Towards Differentiable ML Pipelines beyond a Single Model. *Proc. VLDB Endow.* 15, 1 (2021), 11–20.

[165] Binhang Yuan, Cameron R. Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. 2022. Distributed Learning of Fully Connected Neural Networks Using Independent Subnet Training. *Proc. VLDB Endow.* 15, 8 (2022), 1581–1590.

[166] Xin Zhang, Yanyan Shen, Yingxia Shao, and Lei Chen. 2023. DUCATI: A Dual-Cache Training System for Graph Neural Networks on Giant Graphs with the GPU. *Proc. ACM Manag. Data* 1, 2 (2023).

[167] Yuhao Zhang and Arun Kumar. 2019. Panorama: A Data System for Unbounded Vocabulary Querying over Video. *Proc. VLDB Endow.* 13, 4 (2019), 477–491.
[168] Yuhao Zhang, Frank McQuillan, Nandish Jayaram, Nikhil Kak, Ekta Khanna, Orhan Kislal, Domino Valdano, and Arun Kumar. 2021. Distributed Deep Learning on Data Systems: A Comparative Analysis of Approaches. *Proc. VLDB Endow.* 14, 10 (2021), 1769–1782.
[169] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: Efficient Graph Neural Network Training at Large Scale. *Proc. VLDB Endow.* 15, 6 (2022), 1228–1242.
[170] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. TGL: A General Framework for Temporal GNN Training on Billion-Scale Graphs. *Proc. VLDB Endow.* 15, 8 (2022), 1572–1580.
[171] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *CoRR* abs/1606.06160 (2016).
[172] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2017. Trained Ternary Quantization. In *ICLR*.

Received July 2023; revised October 2023; accepted November 2023