L3: Accelerator-Friendly Lossless Image Format for High-Resolution, High-Throughput DNN Training

Jonghyun Bae, Woohyeon Baek, Tae Jun Ham, and Jae W. Lee

Seoul National University, Seoul, Korea
{jonghbae, baneling100}@snu.ac.kr, ham.taejun@gmail.com, jaewlee@snu.ac.kr

Abstract. The training process of deep neural networks (DNNs) is usually pipelined with stages for data preparation on CPUs followed by gradient computation on accelerators like GPUs. In an ideal pipeline, the end-to-end training throughput is eventually limited by the throughput of the accelerator, not by that of data preparation. In the past, the DNN training pipeline achieved a near-optimal throughput by utilizing datasets encoded with a lightweight, lossy image format like JPEG. However, as high-resolution, *losslessly*-encoded datasets become more popular for applications requiring high accuracy, a performance problem arises in the data preparation stage due to low-throughput image decoding on the CPU. Thus, we propose L3, a custom lightweight, lossless image format for high-resolution, high-throughput DNN training. The decoding process of L3 is effectively parallelized on the accelerator, thus minimizing CPU intervention for data preparation during DNN training. L3 achieves a 9.29× higher data preparation throughput than PNG, the most popular *lossless* image format, for the Cityscapes dataset on NVIDIA A100 GPU, which leads to 1.71× higher end-to-end training throughput. Compared to JPEG and WebP, two popular *lossy* image formats, L3 provides up to 1.77× and 2.87× higher end-to-end training throughput for ImageNet, respectively, at equivalent metric performance.

Keywords: DNN training, Data preparation, Image processing

1 Introduction

The recent development of deep neural networks (DNNs) has been incited by the large-scale, publicly accessible datasets such as ImageNet [10], CIFAR [19], and SVHN [30]. These datasets are conventionally encoded using a lossy encoding format (e.g., JPEG). However, for emerging application domains requiring high accuracies, such as autonomous driving [6, 13, 26, 58], image generation and restoration [22, 23, 52, 53], denoising [45, 55], and medical diagnosis [15, 44, 57], the use of lossy image format can potentially result in accuracy loss. Thus, the demands for *losslessly* encoded datasets continue to increase for more accurate pixel-wise segmentation and object representations.

Today's end-to-end DNN training pipeline is composed of the data preparation stage on the CPU, followed by the gradient computation stage on the accelerator (e.g., GPU, TPU). Since the next mini-batch data preparation stage overlaps with the current batch's gradient computation stage, the longest stage limits end-to-end training throughput. In the traditional setting of DNN training pipeline employing lossy-encoded datasets, the gradient computation stage is the primary bottleneck. Thus, most of the research has been conducted to improve the model training throughput on the accelerator by reducing inter-node communication overhead [16, 29, 41], optimizing GPU memory [25, 46, 50], and compiler optimization for DNN operators [5, 24].

However, data preparation has recently become the critical performance bottleneck, especially for datasets with high-resolution, lossless images, which require a complex decoding process. According to a recent study [38], the throughput ratio of the gradient computation to the data preparation can be as high as 54.9× even with well-optimized data preparation. Balancing the pipeline stages, in this case, would require 50× more CPU cores. There are several recent proposals to address stalls in the data preparation, such as NVIDIA's Data Loading Library (DALI) [33] and TrainBox [38]. However, they only target lossy-encoded datasets with custom hardware support using a dedicated hardware JPEG decoder on NVIDIA A100 GPU [31] and an FPGA accelerator [38].

Thus, we propose L3, a new lightweight, lossless image format, whose decoding can be fully accelerated on data-parallel architectures such as GPUs. L3 eliminates the data preparation stalls by replacing the complex decoding algorithm running on the CPU to achieve the end-to-end training throughput close to the ideal pipeline with zero overhead for data preparation. For standalone execution L3 on NVIDIA A100 GPU delivers a 9.29× higher decoding throughput for the Full HD (1920×1080) Cityscapes dataset than the lossless PNG format on Intel Xeon CPU while maintaining at most 9% loss in the compression ratio. Furthermore, L3 achieves a 1.25× (1.77×) and 1.93× (2.87×) higher geomean (maximum) training throughput than JPEG and lossy WebP, respectively, for seven state-of-the-art object detection and semantic segmentation models at equivalent metric performance.

2 Related Work

Optimized Data Preparation. There exist several recent proposals on improving the data preparation performance for DNN training. DIESEL [51] splits files into two parts—in-memory metadata and on-disk objects, to fully utilize the I/O bandwidth by reducing redundant metadata accesses. CoorDL [27] uses host memory as a cache for the storage and utilizes a new caching policy named MinIO, which does not replace the once-cached dataset elements. *tf.data* [28] proposes an optimized input data pipeline for TensorFlow with optimized parallel I/O, caching, and automatic resource (e.g., CPU, I/O) management to minimize the data preparation latency. These proposals reduce the latency of `Load` but do not address the `Decode` which can be the main bottleneck, especially when the

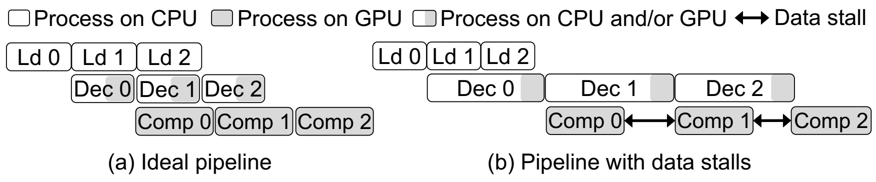

Fig. 1. (a) Ideal DNN training pipeline (b) DNN training Pipeline with data stalls (Ld: `Load`, Dec: `Decode`, Comp: `Compute`)

input images employ lossless encoding. In contrast, L3 presents a new lossless, accelerator-friendly image format to provide superior `Decode` throughput, hence effectively eliminating the bottleneck in the `Decode` stage.

Optimized Lossless Image Decoding. Many proposals attempt to improve the decoding throughput of the lossless image format, such as Huffman decoding [47, 54, 56], bzip2 [39], and LZSS [36] by utilizing GPUs or specialized hardware. However, there are trade-offs between decoding throughput and compression ratio. So, achieving both high decoding throughput and high compression ratio simultaneously is challenging. Instead, Gompresso [48] utilizes a slightly modified LZ77 algorithm and partition-based Huffman coding to improve the compression/decompression throughput on GPUs. Similarly, Adaptive LossLess (ALL) data compression [12] exploits run-length and adaptive dictionary-based coding for GPUs, as well as a partition-based compression/decompression scheme to improve throughput. However, both research target general-purpose applications; hence, they are sub-optimal for DNN training. Instead, L3 is a more specialized format for DNN training to achieve high throughput while maintaining a competitive compression ratio.

3 Background and Motivation

3.1 DNN Training Pipeline

The DNN training operates as a three-stage pipeline: (1) `Load` image, (2) `Decode` image, and (3) `Compute` gradients. For each iteration, a single batch of images are loaded from a storage device to the host main memory (`Load`), and then those images are decoded with CPU and/or GPU [34, 35] (`Decode`). These decoded images are transferred to the GPU device, and then a gradient computing iteration is performed (`Compute`). In most DNN frameworks such as PyTorch [43], TensorFlow [1], and MXNet [4], these three steps execute in a pipelined manner.

Since those three stages are pipelined, the overall training throughput is determined by the stage that takes the longest time among `Load`, `Decode`, and `Compute`. Ideally, the `Load` and `Decode` time should be shorter than `Compute` time so that the time spent on `Load` and `Decode` is completely hidden, as shown in Figure 1(a). However, as shown in Figure 1(b), either `Decode` or `Load` stage may take longer to bottleneck the pipeline. Such cases are likely to happen when high-resolution images are decoded on CPU.

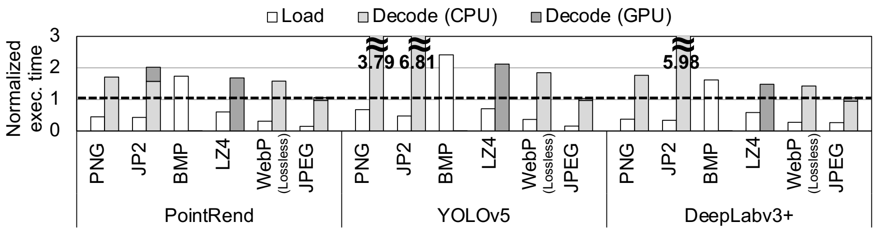

Fig. 2. `Load` and `Decode` execution time normalized to the `Compute` time (in the dotted line) for three representative models in Table 2 and six image formats.

3.2 Data Preparation Bottleneck

Profiling Data Preparation Bottleneck. Figure 2 compares the `Load` and `Decode` time with the `Compute` time for two segmentation models (PointRend [18], DeepLabv3+ [3]) and one detection model (YOLOv5 [49]). The models run on a platform with Intel Xeon Platinum 8275CL CPU, NVIDIA A100 GPU with 40GB HBM2 DRAM, and 1TB NVMe SSD. The datasets and mini-batch size used for each model are described in Section 5.1. The graph shows that the `Load` and `Decode` time varies greatly across input image formats, even for the same model. For example, the PNG, JP2, and lossless WebP spend more time on `Decode` than `Compute`; this implies that the DNN training pipeline is bottlenecked by `Decode`. On the other hand, the BMP has a longer `Load` time than `Compute` time, indicating that the pipeline is bottlenecked by `Load`. This is because the uncompressed BMP data tends to be much larger than other compressed image formats. Among those measured, the JPEG is the only one that is not bottlenecked in data preparation (i.e., `Load` and `Decode`). However, JPEG is a lossy image format that loses some information in the original data.
Decoding Bottleneck for Lossless Image Formats. JPEG format requires much less time to decode as it utilizes GPU to perform most of the decoding process. Since modern high-end GPUs have substantially higher computation power than CPUs, this greatly helps to prevent `Decode` from being a bottleneck. However, much less attention has been paid to accelerating *lossless* image formats. NVIDIA proposes the use of LZ4-based format with its library to accelerate the decoding of LZ4-compressed images using GPU [32]; however, as shown in Figure 2, its decoding speed is much slower than that of JPEG. Also, it is much more challenging to accelerate `Decode` of other lossless image formats. For example, it is well-known that PNG decoding, especially a process of decoding data compressed with dynamic Huffman coding, is difficult to parallelize and thus not well suited for GPU implementation [12, 48, 56].

3.3 Comparison of Lossless and Lossy Image Formats

Image formats can be lossy or lossless. A lossless image format (e.g., PNG, BMP, JPEG2000, lossless WebP) keeps all information in the raw image. In contrast,

Table 1. Test set accuracy and its standard deviation for seven object detection and semantic segmentation models on PNG (Lossless), JPEG (Lossy), and WebP (Lossy) encoded datasets

Model	**Lossless**	**Lossy**	
	PNG (Stdev.)	**JPEG (Stdev.)**	**WebP (Stdev.)**
DDRNet23-slim (mIoU)	0.729 (0.001)	0.693 (0.003)	0.689 (0.001)
DeepLabv3+ (mIoU)	0.801 (0.004)	0.801 (0.007)	0.808 (0.009)
MaskFormer (mIoU)	0.685 (0.008)	0.669 (0.004)	0.651 (0.007)
PointRend (mIoU)	0.725 (0.001)	0.711 (0.003)	0.712 (0.002)
EgoNet (PCK@0.1)	0.323 (0.005)	0.300 (0.002)	0.305 (0.002)
PointPillar (mAP)	0.611 (0.006)	0.593 (0.004)	0.582 (0.009)
YOLOv5 (mAP@50)	0.633 (0.004)	0.605 (0.003)	0.614 (0.007)

lossy image formats (e.g., JPEG, lossy WebP) often require less storage space but lose some information in the raw image. In the context of DNNs, high-resolution, lossless image formats are commonly utilized for domains where model accuracy is critical such as autonomous driving [6, 13, 26, 58], image generation and restoration [22, 23, 52, 53], denoising [45, 55], and medical diagnosis [15, 44, 57]. Table 1 shows the impact of lossy image format on the test set accuracy and its standard deviation of seven object detection and semantic segmentation models, which are often deployed for autonomous driving. We use the default JPEG and lossy WebP quality factor of the Python Pillow package, which is 75. The reported accuracy number is an average of three training runs, and we observe negligible variations in the accuracy. Other than the image format, we use the same hyperparameters taken from the original publication without tuning for both lossless and lossy image formats. The use of the JPEG and lossy WebP results in degradation of the test set accuracy in all models except DeepLabv3+, compared to a case where the lossless PNG format is used for encoding.

4 L3 Design

4.1 Design Goal

L3 is a new image format specialized for a specific use case (i.e., ML/DL training) with a lightweight, lossless encoding/decoding algorithm. Our design goal is to eliminate the data preparation bottleneck by (i) maximizing decoding throughput by leveraging the GPU while (ii) providing a good-enough compression ratio not to introduce a new bottleneck in the `Load` stage. The conventional lossless image formats feature different tradeoffs. If the compression ratio is most important, WebP would be the best option. If compatibility is most important, PNG would be the one. Instead, we carefully design L3 to efficiently accelerate its decoding process on the GPU, unlike existing lossless image formats that are not suitable for GPU acceleration due to their limited parallelism. With L3, the training system can eliminate the CPU bottleneck to improve the DNN training throughput for datasets with lossless, high-resolution images.

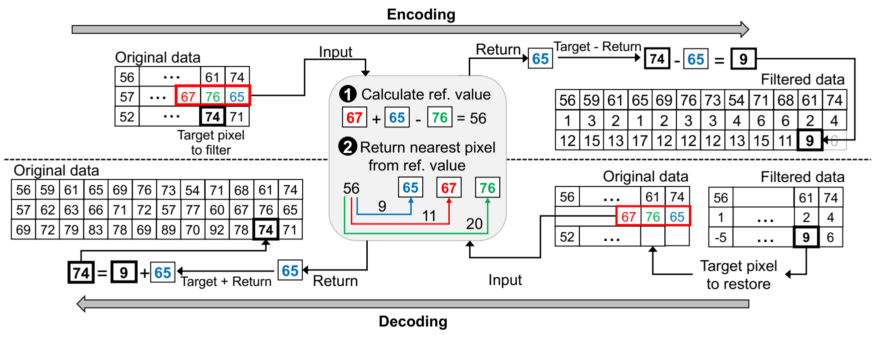

Fig. 3. Example of the custom Paeth filter in L3

4.2 Encoding/Decoding Algorithm

Encoding and decoding for L3 is a two-stage process: Paeth filtering followed by the row-wise base-delta encoding/decoding. Essentially, L3 utilizes a customized Paeth filter to significantly reduce the number of bits required to represent a pixel, exploiting the spatial redundancy (i.e., nearby pixels have similar values). After Paeth filtering, L3 performs base-delta encoding/decoding to reduce further the number of bits representing each pixel and uses a packed representation to store the image in a compressed format. The compressed image can be decoded by performing the reverse operation of each stage. In what follows, we describe each stage in greater detail.

Custom Paeth Filter. The Paeth filter [37], best known for its usage in PNG encoding, is a popular method to reduce the range of values for each pixel. The original Paeth filter encodes each pixel based on three neighboring pixels (left, top, top-left). However, this selection creates both *row-wise* dependency along the row dimension (left) as well as *column-wise* dependency (top) along the column dimension, thus making it challenging to exploit fine-grained (i.e., pixel-level) parallelism. Therefore, L3 customizes the Paeth filter to make it more amenable to parallel execution by inspecting a different set of neighboring pixels (top-left, top, top-right), thus eliminating column-wise data dependency.

Figure 3 shows the encoding/decoding process of the custom Paeth filter in L3. In the rest of this paper, we refer to this as the Paeth filter for brevity. First, the Paeth filter calculates the *reference value* using those three neighboring pixels as follows: $Top\_Left + Top\_Right - Top$ (Step ❶). Then, among the three neighboring pixels, the filter selects the one whose value is the closest to the reference value (Step ❷). Finally, the difference between the original pixel value (i.e., 74 in the example) and the selected neighboring pixel value (i.e., 65 in the example) is stored. As an exception, the first row does not have a preceding row, so this row's filter operation is skipped.

The decoding process is similar to the encoding process. Since the first row is stored in a raw data format, we can start decoding from the second row towards the bottom row. The reference value can be computed by inspecting the three

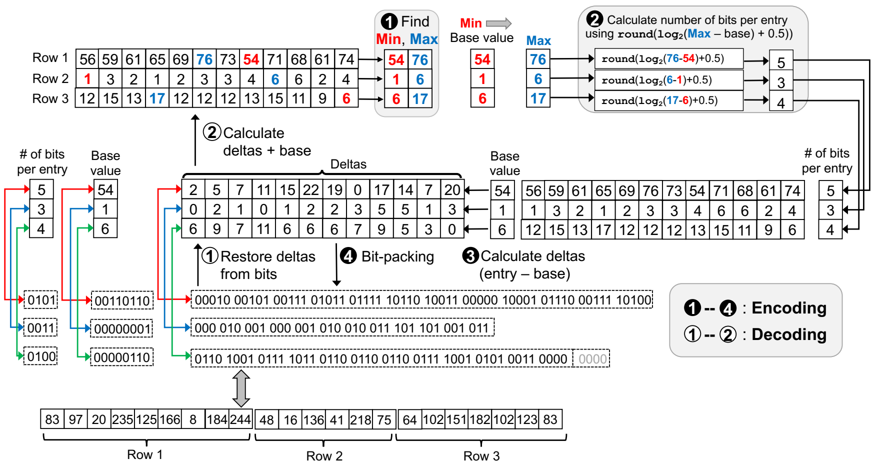

Fig. 4. Example of base-delta encoding/decoding

neighboring pixels in the preceding row to identify the pixel whose value is the closest to the reference. Then we add the stored residual (i.e., 9 in the example) to the pixel value (i.e., 65 in the example) to recover the original pixel value.

Base-Delta Encoding/Decoding. The outcome of the custom Paeth filter has a reduced value range. L3 applies the base-delta encoding [11, 40] to each row to reduce the number of bits representing each pixel. Figure 4 illustrates this base-delta encoding and decoding process. Encoding is a four-step process. First, each row's minimum and maximum values are found (Step ❶). The minimum value is selected as the *base value* of each row. Also, the minimum number of bits required to cover all delta values is computed (Step ❷). Then, the deltas from the base value for all elements are calculated (Step ❸). Finally, a compressed bitstream is generated by representing each delta using the minimum number of bits (Step ❹). Specifically, the first four bits represent the number of bits per delta for each row, and the following eight bits the base value. Then, the deltas of the row, each represented using the minimum number of bits, are appended. All the rows are concatenated to generate the final encoded stream, as shown at the bottom of Figure 4.

The decoding process is quite simple. From the compressed stream, the decoder first reads the four bits as well as the following eight bits to identify the number of bits per entry and the base value for the row (Step ①). From this point, the decoder extracts the delta one by one and adds the base value to reconstruct the original value (Step ②). Once the first row is completed, the same process is repeated for the rest of the rows.

4.3 L3 File Format

The L3 encoding/decoding algorithm can work with any large-sized image. In practice, to maximize the parallelism and improve the decoding throughput in

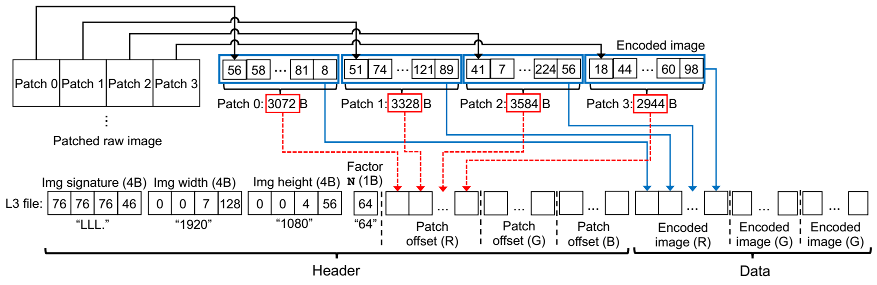

Fig. 5. Example of L3 file formatting

GPUs, L3 divides a large image into multiple square *patches* of size $N \times N$, where N is the user-specified parameter. We empirically set N=32 for images whose resolution is below 1080×720 (HD), N=64 for images whose resolution is between 1080×720 (HD) and 1920×1080 (FHD), and N=128 for images whose resolution is between 1920×1080 (FHD) and 3840×2160 (UHD). Once N is set, the image is first separated into three channels (R, G, and B). Then, the image is divided into square-sized patches for each channel, and the encoding algorithm is applied to each patch. The encoded patch is concatenated, and the offset for each patch is recorded and stored in a separate array.

Figure 5 shows the L3 file format. The file consists of the header and the data sections. The header section contains a 4-byte file type signature (also called magic bytes), 4-byte image width, 4-byte image height, 1-byte patch size, and three patch offset arrays corresponding to the three color channels. The data section contains encoded patch data for patches.

4.4 Optimizing L3 Decoder on GPU

Patch-level Parallelism. Figure 6(a) illustrates patch-level parallelism in L3. Without parallelism, a single thread block, a unit of scheduling for GPU devices, decodes the entire image over multiple iterations, underutilizing GPU resources. Instead, the L3 decoder first reads the header of each image and then splits it into multiple patches based on the header. Then, the task of decoding a single patch is assigned to a distinct thread block. This enables the GPU to process multiple patches in parallel to exploit the massive parallelism of GPU devices.

Row-wise Parallel Paeth Filter. Even within a single patch, it is necessary for GPU devices to exploit fine-grained parallelism to maximize performance. For the Paeth filter, we implement the decoder to process a set of pixels within a single row in parallel. Figure 6(b) presents the row-wise parallel processing of the Paeth filter. Because of both row-wise and column-wise data dependencies between three neighbor pixels (Section 4.2), the original Paeth filter should decode the pixels in a single patch sequentially from top-left to bottom-right. Our use of the custom Paeth filter makes parallelization easier as it inspects three top pixels in the preceding row for decoding, instead of the one left pixel and two top

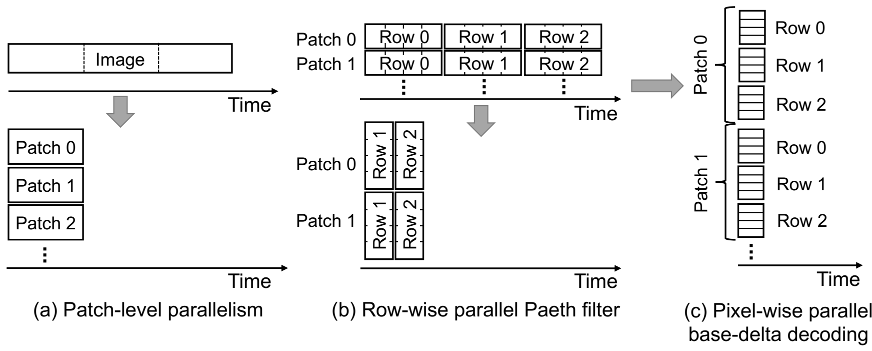

Fig. 6. Parallel execution example of (a) Patch-level parallelism (b) Row-wise parallel Paeth filter (c) Pixel-wise parallel base-delta decoding.

pixels (i.e., top, top-left) as in the original Paeth filter. Starting from the second row, pixels within a single row are decoded in parallel (i.e., each CUDA thread processes a single pixel in a row). Once the decoding for a row is completed, the decoding for the next row within the same patch begins. This process is repeated for all rows in the patch.

Pixel-wise Parallel Base-Delta Decoding. Figure 6(c) shows the pixel-level parallel base-delta decoding in L3. The original execution takes the same order with the original Paeth filter–it reconstructs pixels from the first pixel to the last one sequentially in the patch. The L3 decoder assigns a thread for base-delta decoding each pixel to exploit pixel-level parallelism within a row. Basically, each thread extracts the delta corresponding to the pixel and then adds the base value to reconstruct the original value.

Overlapping Decoding and Gradient Computing on GPU. To execute both computing and decoding concurrently on the GPU, we allocate both processes to separate *CUDA streams*. We prioritize the computing stream over the decoding stream to prevent the computing (processing the current batch) from being interfered with by the decoding (processing the subsequent batch).

5 Evaluation

5.1 Methodology

Experimental Setup. We implement the decoder of L3 by extending NVIDIA Data Loading Library (DALI) [33] (version 1.6.0) by registering L3 as a new image format. We initiate the training for each model by running DALI-integrated PyTorch (version 1.9.0) [43]. For experiments, we use `p4d.24xlarge` AWS EC2 instance with 8×NVIDIA A100 GPUs with 40GB HBM2 DRAM per each, 96 vCPUs on Intel Xeon Platinum 8275CL CPU, and 8×1TB NVMe SSD.

Models and Datasets. We compare L3 with various lossless (PNG, JP2, BMP, LZ4, lossless WebP) and lossy (JPEG, lossy WebP) image formats and decoding method. Note that WebP support both lossless and lossy compression. For

Table 2. Detailed descriptions about models, datasets, and mini-batch sizes

Dataset (Resolution) # of train/val/test	**Model (Backbone)**	**Batch size**
Cityscapes (1920×1080) 2975/500/1525	DDRNet23-slim	36
	DeepLabv3+ (MobileNetv2)	48
	MaskFormer (ResNet50)	32
	PointRend (SemanticFPN)	40
KITTI (1024×720) 3519/3462/500	EgoNet	24
	PointPillars	48
	YOLOv5	128
DIV2K (2040×1200)	For compression ratio only	
FFHQ (5760×3840)	For compression ratio only	
RAISE-1K (4928×3264)	For compression ratio only	

PNG, JP2, BMP, WebP, and JPEG, we use the implementation of the NVIDIA DALI framework [33]. For LZ4 decoding we use NVIDIA `nvcomp` library [32]. We use seven object detection and semantic segmentation models (`DDR`: DDRNet23-slim [14], `DL`: DeepLabv3+ [3], `MF`: MaskFormer [7], `PR`: PointRend [18], `EN`: EgoNet [21], `PP`: PointPillars [20], and `YOLO`: YOLOv5 [49]) and two datasets (Cityscapes [8] and KITTI [13]) for throughput comparison. For the comparison of compression ratios of the lossless formats, we use three additional high-resolution datasets: DIV2K [2], FFHQ [17], and RAISE-1K [9]. Table 2 summarizes the model, dataset, and batch size used for training. The batch size is chosen to the maximum value for each model without causing an out-of-memory error. For the hyperparameters not shown in the table, we use the default values suggested in the papers or open-source implementations.

Quality Factor of Lossy Image Formats. As we discuss in Section 3.3, a lossy image format may yield a lower metric performance than the lossless image format at the default configuration. For fair comparison, we tune the quality factor (Q-factor) of a lossy image format such that it achieves an equivalent metric performance to the lossless image format. Specifically, we set the Q-factor to be the minimum value while its accuracy falls within 1% of that of the lossless image format. Table 3 reports the accuracy of L3, JPEG (Lossy), and WebP (Lossy) with the corresponding quality factors for the latter two. We use these values of Q-factor to compare the training throughput of JPEG and WebP (Lossy) with that of L3.

5.2 Compression Ratio

Table 4 reports the compression ratio of the lossless formats, including L3, where lower is better. For this experiment, we use five image datasets, as well as two additional synthetic images. The random image (Random) selects a random value for each pixel in the range of 0 through 255. The black image (Black) sets all pixel values to zero. Except for the black image, the compression ratio of L3 is worse than PNG and WebP (Lossless), falling within 6-9% of the PNG and 9-30% of

Table 3. Test set accuracy of L3 (Lossless), JPEG (Lossy), and WebP (Lossy) and its quality factor

Model	**L3 acc.**	**JPEG acc. (Q-factor)**	**WebP acc. (Q-factor)**
DDR	0.729	0.730 (95)	0.726 (95)
DL	0.801	0.801 (75)	0.808 (75)
MF	0.685	0.686 (90)	0.685 (95)
PR	0.725	0.721 (85)	0.722 (85)
EN	0.323	0.323 (80)	0.324 (80)
PP	0.611	0.613 (85)	0.616 (95)
YOLO	0.633	0.635 (85)	0.635 (85)

Table 4. Data compression ratio (compressed size/decompressed size). Lower is better.

	Cityscapes	**KITTI**	**DIV2K**	**FFHQ**	**RAISE1K**	**Random**	**Black**
Raw (MB)	5.93	2.24	7.01	63.28	46.02	5.93	5.93
PNG	0.39×	0.62×	0.67×	0.25×	0.57×	1.01×	0.001×
JP2	0.38×	0.57×	0.59×	0.28×	0.56×	1.09×	0.005×
BMP	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
LZ4	0.79×	0.85×	0.83×	0.44×	0.69×	1.01×	0.026×
WebP	0.29×	0.55×	0.49×	0.15×	0.37×	1.00×	0.000002×
L3	0.44×	0.64×	0.76×	0.33×	0.63×	1.02×	0.13×

the WebP (Lossless) format. However, we reiterate that the design objective of L3 is not to maximize the compression ratio but to balance the DNN training pipeline with more balanced resource utilization. In particular, L3 eliminates the data preparation bottleneck by (i) maximizing decoding throughput while (ii) providing a good-enough compression to not cause data stalls on the `Load` stage. L3's compression ratio is substantially worse in Black because L3 does not incorporate a recurring patterns compression (e.g., run-length encoding, Huffman coding). However, we find that containing many trivially repeated patterns is not common in lossless, high-resolution images.

5.3 Throughput Comparison with Lossless Decoders

Data Preparation Throughput. Figure 7(a) compares the data preparation throughput (`Load`+`Decode`) of L3 and other lossless decoding formats with varying resolutions. We utilize the FHD Cityscapes dataset and its scaled-down/up versions using the Lanczos filter in Python Pillow package, which provides the highest quality for image down/upscaling [42]. Compared to PNG, L3 improves the data preparation throughput by 5.67×, 9.29×, and 15.71× for HD, FHD, and UHD images, respectively. Furthermore, L3 outperforms WebP (Lossless), a state-of-the-art lossless image format, by a factor of 2.41×, 3.25×, and 4.51× for the same datasets. Thus, L3 achieves substantially higher data preparation throughput than all the other lossless formats, regardless of the resolution.

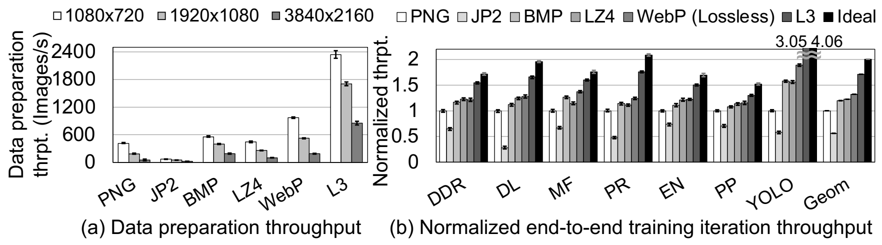

Fig. 7. Throughput comparison with lossless-encoded datasets (a) Data preparation (`Load`+`Decode`) throughput (b) Normalized end-to-end training iteration throughput

DNN Training Throughput. Figure 7(b) presents the normalized end-to-end training throughput (iterations/sec) of different image formats on various models. The training throughput is normalized to PNG. *Ideal* refers to the case when the data preparation stage is completely hidden by the `Compute` stage, thus yielding zero overhead. The choice of the image format has a significant impact on the training throughput as the data preparation time can potentially bottleneck the DNN training pipeline.

Overall, L3 achieves the best end-to-end training throughput with a 1.71× geomean speedup compared to PNG. PNG and JP2 achieve lower throughput mainly because their complex algorithms running on CPU make them slow. The training throughput of BMP is limited by the I/O bandwidth (`Load`). LZ4 decoding utilizes GPUs, but its throughput is much lower than L3, achieving only 1.23× throughput speedup compared to PNG. WebP (Lossless) is the most recently proposed lossless image format, which achieves a 1.32× higher geomean training throughput than PNG. In contrast, L3 achieves a 1.71× higher throughput than PNG, which outperforms WebP (Lossless) by a significant margin.

5.4 Throughput Comparison with Lossy Decoders

Impact of Q-Factor on Data Preparation Throughput. Figure 8 shows the trade-off between quality factor and data preparation throughput of JPEG (Lossy) and WebP (Lossy). We use the scaled-down/up versions of Cityscapes dataset and convert it to JPEG and WebP format. The black dotted line marks the data preparation throughput of L3 (Lossless) for a given image resolution. WebP (Lossy) decoding is done on CPU, whereas JPEG decoding on the dedicated hardware JPEG accelerator on NVIDIA A100 GPU using `nvJPEG` library. Even with the lowest quality factor (i.e., highest decoding throughput), WebP (Lossy) has a lower decoding throughput than L3 for all three image resolutions. This leads to a substantially lower end-to-end training throughput of WebP (Lossy) than L3.

In case of JPEG, there is a crossover point of the quality factor, where JPEG starts to outperform L3 in terms of decoding throughput. As we increase the image resolution from HD to UHD, L3 observes diminishing returns in throughput gains (in terms of pixels/sec) as the utilization of CUDA cores becomes

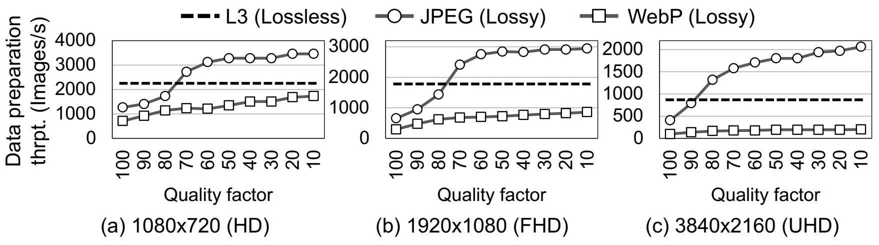

Fig. 8. Data preparation throughput of JPEG (Lossy) and WebP (Lossy)-encoded Cityscapes datasets on various quality factors. (a) 1080×720 (HD) (b) 1920×1080 (FHD) (c) 3840×2160 (UHD) resolution

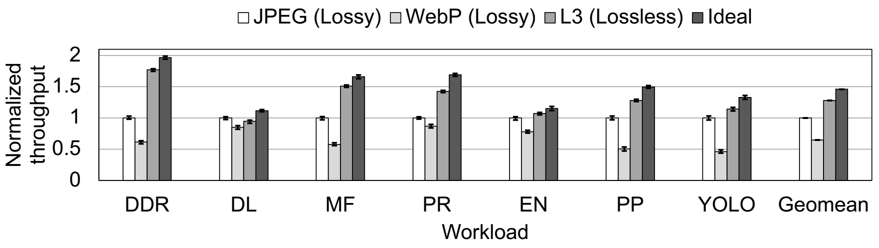

Fig. 9. Normalized training iteration throughput with JPEG (Lossy), WebP (Lossy), and L3 (Lossless)-encoded dataset

high. JPEG performance scales a bit more gracefully to image resolution than L3 as the entire decoder runs on a custom JPEG hardware accelerator which is available only on NVIDIA A100 GPU. However, L3 can provide more robust performance on various data-parallel accelerators without requiring format-specific hardware support. In contrast, JPEG performance depends highly on the existence of custom hardware and enough CPU cores to sustain high throughput.

DNN Training Throughput. Figure 9 shows the end-to-end training iteration throughput for JPEG and WebP (Lossy) at equivalent test set accuracy. For comparable metric performance the quality factor of JPEG and WebP is set to the value in Table 3). The throughput is normalized to that of JPEG. Overall, L3 achieves up to 1.77× and 2.87× higher end-to-end training throughput than JPEG and WebP (Lossy), respectively. The geomean throughput gains are 1.25× for JPEG and 1.93× for WebP. Decreasing the quality factor may boost training throughput, but this comes with the cost of degrading the metric performance.

5.5 Ablation Study: Execution Time of L3 Decoder

L3 customizes the Paeth filter to make it more amenable to parallel execution by inspecting a different set of neighboring pixels (top-left, top, top-right), thus eliminating column-wise data dependency. Also, L3 unleashes pixel-level parallelism within the base-delta decoding step. Finally, L3 processes the decoding tasks in parallel with patch-level parallelism to exploit the massive parallelism of

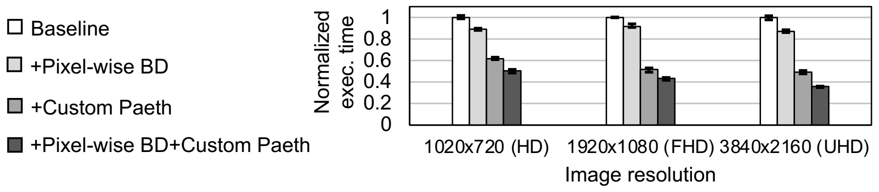

Fig. 10. Normalized L3 decoder execution time on various image resolution (`Baseline`: Seq. original Paeth filter+Seq. base-delta decoding, `+Pixel-wise BD`: Baseline+pixel-wise parallel base-delta decoding, `+Custom Paeth`: Baseline+row-wise parallel custom Paeth filter, `+Pixel-wise BD+Custom Paeth`: Pixel-wise parallel base-delta decoding+Row-wise parallel custom Paeth filter)

GPU devices. The patch-level parallelism reduces the decoding time by 76.8%, 83.9%, and 86.5% for HD, FHD, and UHD resolutions, respectively. We set the state in which patch-level parallelism is applied as baseline.

Figure 10 shows an ablation study to quantify the benefits of each component. The execution time is normalized by the baseline with sequential base-delta decoding and the original Paeth filter (Baseline). We use the FHD Cityscapes dataset and scale up/down the images with the Lanczos filter in Python Pillow package. The figure shows that pixel-wise parallel base-delta decoding (+Pixel-wise BD) and the custom Paeth filter (+Custom Paeth) reduces the decoding time by an average of 10.8% and 46.0%, respectively, over the baseline for the three image resolutions. With both optimizations applied (+Pixel-wise BD+Custom Paeth), the overall decoding time is reduced by 49.5%, 56.7%, and 59.1% from the Baseline for HD, FHD, and UHD images, respectively.

6 Conclusion

We propose L3, a new lightweight, lossless image format for high-resolution, high-throughput DNN training. The decoding algorithm of L3 is accelerated on data-parallel architectures such as GPUs. Thus, L3 yields much faster decoding than the existing lossless image formats whose decoders are mostly running on the CPU. L3 effectively eliminates data preparation bottlenecks in the DNN training pipeline. L3 can be readily deployed on GPU without requiring any support for specialized hardware. L3 can significantly reduce the end-to-end training time by providing higher throughput, higher accuracy, or both, compared to the existing lossless and lossy image formats at equivalent test set accuracy.

Acknowledgements. This work was supported by SNU-SK Hynix Solution Research Center (S3RC), and National Research Foundation of Korea (NRF) grant funded by Korea government (MSIT) (NRF-2020R1A2C3010663). The source code is available at `https://github.com/SNU-ARC/L3.git`. Jae W. Lee is the corresponding author.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A system for large-scale machine learning. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation. pp. 265–283. USENIX Association (November 2016)
2. Agustsson, E., Timofte, R.: NTIRE 2017 challenge on single image super-resolution: Dataset and study. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (July 2017)
3. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Proceedings of the European Conference on Computer Vision (September 2018)
4. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., Zhang, Z.: MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274 (2015)
5. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A.: TVM: An automated end-to-end optimizing compiler for deep learning. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation. pp. 578–594. USENIX Association (October 2018)
6. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3D object detection network for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (July 2017)
7. Cheng, B., Schwing, A., Kirillov, A.: Per-pixel classification is not all you need for semantic segmentation. In: Proceedings of the Advances in Neural Information Processing Systems. vol. 34, pp. 17864–17875. Curran Associates, Inc. (December 2021)
8. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (June 2016)
9. Dang-Nguyen, D.T., Pasquini, C., Conotter, V., Boato, G.: RAISE: A raw images dataset for digital image forensics. In: Proceedings of the 6th ACM Multimedia Systems Conference. p. 219–224. Association for Computing Machinery (March 2015)
10. Deng, J., Dong, W., Socher, R., jia Li, L., Li, K., Fei-fei, L.: ImageNet: A large-scale hierarchical image database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE (June 2009)
11. Farrens, M., Park, A.: Dynamic base register caching: A technique for reducing address bus width. In: Proceedings of the 18th Annual International Symposium on Computer Architecture. p. 128–137. Association for Computing Machinery (May 1991)
12. Funasaka, S., Nakano, K., Ito, Y.: Adaptive loss-less data compression method optimized for GPU decompression. Concurrency and Computation: Practice and Experience **29**(24), e4283 (August 2017)
13. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? The KITTI vision benchmark suite. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (June 2012)

14. Hong, Y., Pan, H., Sun, W., Jia, Y.: Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. arXiv preprint arXiv:2101.06085 (September 2021)
15. Hou, L., Cheng, Y., Shazeer, N., Parmar, N., Li, Y., Korfiatis, P., Drucker, T.M., Blezek, D.J., Song, X.: High resolution medical image analysis with spatial partitioning. arXiv preprint arXiv:1909.03108 (September 2019)
16. Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q.V., Wu, Y., Chen, z.: GPipe: Efficient training of giant neural networks using pipeline parallelism. In: Proceedings of the Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (December 2019)
17. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (June 2019)
18. Kirillov, A., Wu, Y., He, K., Girshick, R.: PointRend: Image segmentation as rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (June 2020)
19. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. In: Technical report. Citeseer (April 2009)
20. Lang, A.H., Vora, S., Caesar, H., Zhou, L., Yang, J., Beijbom, O.: PointPillars: Fast encoders for object detection from point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (June 2019)
21. Li, S., Yan, Z., Li, H., Cheng, K.T.: Exploring intermediate representation for monocular vehicle pose estimation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1873–1883 (June 2021)
22. Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., Timofte, R.: SwinIR: Image restoration using swin transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops. pp. 1833–1844 (October 2021)
23. Lim, B., Son, S., Kim, H., Nah, S., Mu Lee, K.: Enhanced deep residual networks for single image super-resolution. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 136–144 (July 2017)
24. Ma, L., Xie, Z., Yang, Z., Xue, J., Miao, Y., Cui, W., Hu, W., Yang, F., Zhang, L., Zhou, L.: Rammer: Enabling holistic deep learning compiler optimizations with rTasks. In: Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation. pp. 881–897. USENIX Association (November 2020)
25. Markthub, P., Belviranli, M.E., Lee, S., Vetter, J.S., Matsuoka, S.: DRAGON: Breaking GPU memory capacity limits with direct NVM access. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. pp. 32:1–32:13. IEEE (November 2018)
26. Menze, M., Geiger, A.: Object scene flow for autonomous vehicles. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (June 2015)
27. Mohan, J., Phanishayee, A., Raniwala, A., Chidambaram, V.: Analyzing and mitigating data stalls in DNN training. Proceedings of the VLDB Endowment **14**(5), 771–784 (Jan 2021)
28. Murray, D.G., Simsa, J., Klimovic, A., Indyk, I.: tf.data: A machine learning data processing framework. Proceedings of the VLDB Endowment **14**(12), 2945–2958 (July 2021)
29. Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N.R., Ganger, G.R., Gibbons, P.B., Zaharia, M.: PipeDream: Generalized pipeline parallelism for DNN training. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. p. 1–15. Association for Computing Machinery (October 2019)

30. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning (December 2011)
31. NVIDIA: NVIDIA A100 tensor core GPU architecture. `https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf` (2020)
32. NVIDIA: nvcomp: A library for fast lossless compression/decompression on the GPU. `https://github.com/NVIDIA/nvcomp` (2021)
33. NVIDIA: The NVIDIA data loading library (DALI). `https://github.com/NVIDIA/DALI` (2021)
34. NVIDIA: nvJPEG libraries: GPU-accelerated JPEG decoder, encoder and transcoder. `https://developer.nvidia.com/nvjpeg` (2021)
35. NVIDIA: nvJPEG2000 libraries. `https://docs.nvidia.com/cuda/nvjpeg2000` (2021)
36. Ozsoy, A., Swany, M.: CULZSS: LZSS lossless data compression on CUDA. In: Proceedings of the 2011 IEEE International Conference on Cluster Computing. pp. 403–411 (September 2011)
37. Paeth, A.W.: II.9 - Image file compression made easy. In: Graphics Gems II, pp. 93–100. Morgan Kaufmann (1991)
38. Park, P., Jeong, H., Kim, J.: TrainBox: An extreme-scale neural network training server architecture by systematically balancing operations. In: Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture. pp. 825–838 (October 2020)
39. Patel, R.A., Zhang, Y., Mak, J., Davidson, A., Owens, J.D.: Parallel lossless data compression on the GPU. In: Proceedings of the 2012 Innovative Parallel Computing. pp. 1–9 (May 2012)
40. Pekhimenko, G., Seshadri, V., Mutlu, O., Gibbons, P.B., Kozuch, M.A., Mowry, T.C.: Base-delta-immediate compression: Practical data compression for on-chip caches. In: Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques. p. 377–388. Association for Computing Machinery (September 2012)
41. Peng, Y., Zhu, Y., Chen, Y., Bao, Y., Yi, B., Lan, C., Wu, C., Guo, C.: A generic communication scheduler for distributed DNN training acceleration. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. p. 16–29. Association for Computing Machinery (October 2019)
42. Pillow: Python pillow filters. `https://pillow.readthedocs.io/en/stable/handbook/concepts.html\#filters` (2021)
43. PyTorch: PyTorch. `https://pytorch.org` (2021)
44. Rebsamen, M., Suter, Y., Wiest, R., Reyes, M., Rummel, C.: Brain morphometry estimation: From hours to seconds using deep learning. Frontiers in Neurology **11**, 244 (April 2020)
45. Ren, C., He, X., Wang, C., Zhao, Z.: Adaptive consistency prior based deep network for image denoising. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8596–8606 (June 2021)
46. Rhu, M., Gimelshein, N., Clemons, J., Zulfiqar, A., Keckler, S.W.: vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 18:1–18:13. IEEE (October 2016)
47. Sarangi, S., Baas, B.: Canonical huffman decoder on fine-grain many-core processor arrays. In: Proceedings of the 2021 26th Asia and South Pacific Design Automation Conference. pp. 512–517 (January 2021)

48. Sitaridi, E., Mueller, R., Kaldewey, T., Lohman, G., Ross, K.A.: Massively-parallel lossless data decompression. In: Procceedings of the 2016 45th International Conference on Parallel Processing. pp. 242–247 (August 2016)
49. Ultralytics: Yolov5. `https://github.com/ultralytics/yolov5/` (2021)
50. Wang, L., Ye, J., Zhao, Y., Wu, W., Li, A., Song, S.L., Xu, Z., Kraska, T.: SuperNeurons: Dynamic GPU memory management for training deep neural networks. In: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 41–53. ACM (February 2018)
51. Wang, L., Ye, S., Yang, B., Lu, Y., Zhang, H., Yan, S., Luo, Q.: DIESEL: A dataset-based distributed storage and caching system for large-scale deep learning training. In: Proceedings of the 49th International Conference on Parallel Processing. Association for Computing Machinery (August 2020)
52. Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., Qiao, Y., Change Loy, C.: ESRGAN: Enhanced super-resolution generative adversarial networks. In: Proceedings of the European Conference on Computer Vision Workshops (September 2018)
53. Wang, Z., Cun, X., Bao, J., Zhou, W., Liu, J., Li, H.: Uformer: A general u-shaped transformer for image restoration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 17683–17693 (June 2022)
54. Weißenberger, A., Schmidt, B.: Massively parallel huffman decoding on GPUs. In: Proceedings of the 47th International Conference on Parallel Processing. Association for Computing Machinery (August 2018)
55. Xu, L., Zhang, J., Cheng, X., Zhang, F., Wei, X., Ren, J.: Efficient deep image denoising via class specific convolution. Proceedings of the AAAI Conference on Artificial Intelligence **35**(4), 3039–3046 (May 2021)
56. Yamamoto, N., Nakano, K., Ito, Y., Takafuji, D., Kasagi, A., Tabaru, T.: Huffman coding with gap arrays for GPU acceleration. In: Proceedings of the 49th International Conference on Parallel Processing. Association for Computing Machinery (August 2020)
57. Zhou, S., Nie, D., Adeli, E., Yin, J., Lian, J., Shen, D.: High-resolution encoder–decoder networks for low-contrast medical image segmentation. IEEE Transactions on Image Processing **29**, 461–475 (2020)
58. Zhou, Y., Tuzel, O.: VoxelNet: End-to-end learning for point cloud based 3D object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (June 2018)