Optimizing Deep Learning for Biomedical Imaging

Ayush Chaturvedi[1], Guohua Cao[2], and Wu-chun Feng[1]

[1] Virginia Polytechnic Institute and State University, Blacksburg, VA, USA
{ayushchatur,wfeng}@vt.edu
[2] School of Biomedical Engineering, ShanghaiTech University, Shanghai, CHINA
caogh@shanghaitech.edu.cn

Abstract. With the significant increase in the use of deep learning (DL) for biomedical imaging, the corresponding DL models have become increasingly complex and computationally intensive to achieve high accuracy. This work presents both architecture-aware optimizations and sparsity optimizations to efficiently utilize underlying parallel hardware resources and reduce the computational demand of DL models while maintaining their accuracy. We demonstrate the efficacy of our optimization techniques on an existing DL model in the biomedical domain, i.e., DDNet, short for Densenet and Deconvolution Network, that is designed to enhance the quality of CT images. Overall, our optimization techniques in concert reduce the total training time by 1.94× while maintaining accuracy.

Keywords: AI · deep learning · image de-noising · chest CT · COVID-19

1 Introduction

Computed tomography (CT) has been pivotal in detecting abnormalities within the human body. With recent advances in artificial intelligence (AI) and the availability of open-source CT image datasets, deep learning (DL) models are actively being trained to help detect abnormalities in CT images [10, 15, 21]. The accuracy of these DL models depends heavily on the quality of CT images in the datasets, which, in turn, correlate to amount of radiation dosage from a CT scan. While a standard-dosage CT scan can generate high-quality CT images, it increases the attributable risk of death from cancer by up to 0.1% [2]. Thus, medical institutions worldwide use a low-dosage CT (LDCT) scan, resulting in low-quality CT images. In turn, scientists rely on DL models to improve the quality of CT images generated from low-dosage CT scans [9, 14, 23]. However, these LDCT images further exacerbate the computational needs of DL, as training these DL models requires substantial data and state-of-the-art computing resources. Thus, optimization techniques are needed to train these DL models more efficiently with fewer computational resources. To this end, we apply and demonstrate our optimization techniques on our DL image enhancement model called DDNet [27], short for Densenet and Deconvolution Network.

Fig. 1 shows the auto-encoder-decoder architecture of DDNet, consisting of a convolution network and deconvolution network, both connected via skip connections. The convolution network features four denseblocks [11], each containing five densely connected convolution layers for efficient feature extraction. Along with the dense blocks, the network consists of 37 convolution layers in total. The images used to train the network model consist of high-quality (HQ) and low-quality (LQ) chest CT images of size 512×512 in 32-bit grayscale. For the loss computation, the network uses a complex loss function that combines the mean square error (MSE) and multi-scale structural similarity index metric (MS-SSIM) [25].

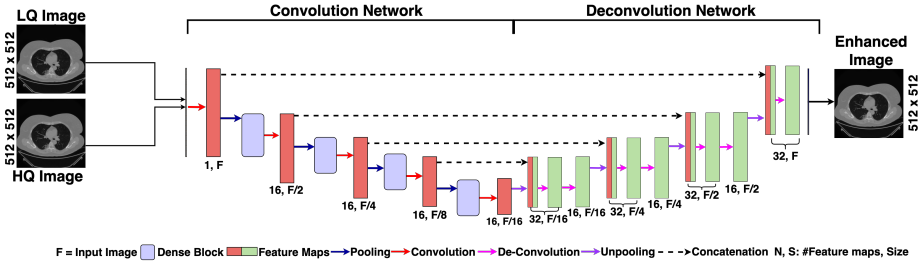


Fig. 1. Architecture of DDNet.

While DDNet improves the quality of chest CT scans and, in turn, results in better COVID-19 detection [8], its architecture requires very large GPU memory and extensive training time. Although both the human brain and a DL model consist of millions of interconnected neurons that serve as fundamental processing units, the human brain, unlike traditional DL models, exhibits a *sparse structure* [7, 12], i.e., not all the neurons in the brain are always interconnected. Hence, we present algorithmic strategies and, in turn, optimizations that incorporate *sparsity* into DL models to reduce their computational demands during training while maintaining accuracy.[3]

In summary, our work improves current state-of-the-art DL-based CT imaging via the following contributions:

- Novel realization of sparse algorithms to reduce complexity and training time of deep-learning (DL) models by accounting for their neural architecture.
- An optimized data loader that mitigates the data-movement latency associated with training DL models on small datasets in the PyTorch framework.
- A mixed-precision algorithm for convolution neural networks that leverages a specialized data format and 'tensor cores' in modern NVIDIA GPUs.

[3] These sparse techniques should *not* be confused with "sparse reconstruction techniques" in biomedical imaging [1]. Thus, for lucidity, we define sparsity in DL models as referring to *the magnitude of zero entries in their programmatic representation.*

2 Sparse Optimizations

Like the human brain, a deep learning (DL) model consists of many layers with millions or billions of neurons. These layers are represented as a combination of parameters (i.e., weights and biases) stored in huge multi-dimensional matrices, also called *tensors*. Having larger layers with more parameters improves accuracy, but operations on huge tensors require significant computational resources. Thus, to reduce the number of effective parameters, sparse techniques modify specific entries in a tensor to zero in a process called *pruning*, thus creating sparsity in tensors and the overall DL model. In this work, we leverage three kinds of sparse techniques for our DDNet model: random unstructured, structured, and magnitude-based sparse optimizations.

2.1 Unstructured and Structured Sparsity

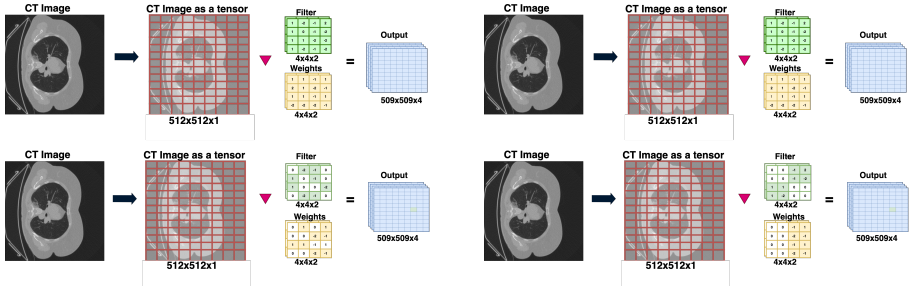
The sparse taxonomy is defined by the structure imposed on the tensors while pruning the values from a DL model. *Structured* sparsity addresses the dimensions of the tensors associated with weights, bias, and filter values, whereas *unstructured* sparsity only formulates the criteria for selecting values from these tensors in a DL model. We use both techniques and apply them to DDNet.

In DDNet, the convolution and deconvolution layers collectively form a significant part of the overall computation in the forward pass, i.e., matrix-multiply-add (MMA) operations on multi-dimensional tensors. Moreover, skip connections across the network and shortcut connections within the dense blocks require intermediate results to be held in memory, thus increasing memory requirements for DDNet. To reduce compute and memory overhead, we introduce sparsity by pruning tensors engaged in skip connections, convolution, and deconvolution.

Random Unstructured: Randomly pruning parameters of the DL layers falls under the class of unstructured sparse techniques [6, 17]. Fig. 2(a) contrasts convolution (red triangle) over a chest CT image without (above) and with (below) randomly pruned tensors. As depicted, random entries in the tensors of the convolution filter and weights are set to zero, thus ceasing to contribute to the model and reducing the number of effective parameters in DDNet.

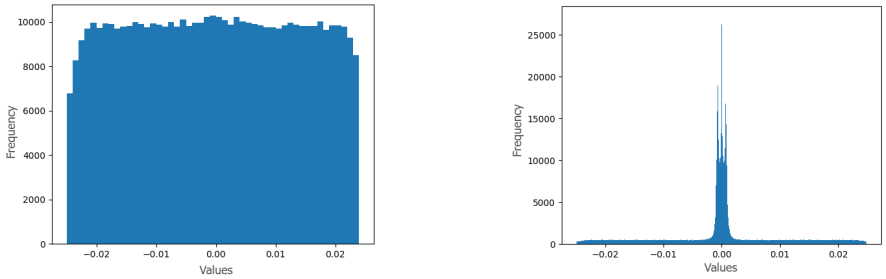
Structured: Fig. 2(b) incorporates structured sparsity into DDNet by pruning entire dimensions in the weights tensors and blocks in the convolution filters. We only prune those dimensions that are *not* associated with skip connections in the weights tensors. In filters, blocks on the upper left corner and lower right corner are pruned. The resulting tensors are modified to a dense representation so that only non-zero entries are brought to the memory. Moreover, pruning blocks or channels in these tensors reduce their effective dimensions and the total number of convolution operations. As a result, the convolution layers are now calculated faster because they require fewer MMA operations and less memory.

Magnitude-based Sparsity: To intelligently prune from a DL model, we employ another sparsification technique that imposes 'criteria' to select parameters



(a) Random sparsity (b) Structured sparsity

Fig. 2. Convolution operation with sparse filter and weights using (a) random sparse and (b) structured sparse techniques, respectively.



(a) (b)

Fig. 3. Normal distribution of parameters for (a) dense DDNet and (b) DDNet with 50% sparsity.

to be pruned. For example, 'Top-K' uses the absolute values of the model parameters as a proxy for their importance. The underlying assumption is that parameters with the smallest magnitude will contribute the least to the DL model. To select which parameters to prune, we analyze the normal distribution of all parameter values associated with every layer in DDNet in Fig. 3(a). By sorting these values based on their magnitude and then pruning K% of the values from the lower half of the sorted distribution, we get a distribution of parameter values, as shown in Fig. 3(b), where K is set to 50.

While sparsity provides significant performance benefits, it comes at the expense of information loss due to the removal of data via pruning. Consequently, the accuracy of the DDNet model suffers, as shown in Table 1. As remediation, the model needs to be *retrained*, which we articulate in §2.2 with our proposed hybrid training schedule.

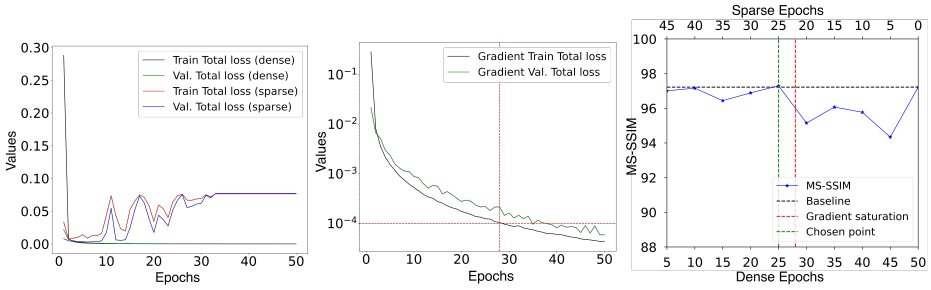
Model	% Sparsity	Sparsity Type	MS-SSIM	Training Time
DDNet (dense)	0	None	97.22 ± 1.49	235 minutes
DDNet (sparse)	50	Random	42.39 ± 2.10	175 minutes

Table 1. Accuracy with sparse and dense DDNet. Hyperparameters: batch size = 1; learning rate = 1e-4; decay rate = 0.95; training epochs = 50.

2.2 Hybrid Training Schedule

While sparse optimizations reduce the required number of parameters, there are certain parameters that encode patterns that are critical for the model's accuracy. Removing such parameters leads to a decrease in the model's ability to accurately predict the target variable (see MS-SSIM column in Table 1).

Fig. 4(a) compares the total training and validation loss for the dense DDNet model (black and green line) and 50% sparse DDNet model (red and blue line). The total training loss of the model with sparsity (blue line) starts at a lower value and keeps decreasing until 10 epochs, at which point it suddenly explodes to finally converge at a final value that is orders of magnitude worse than its dense DDNet counterpart. Thus, to recover the accuracy lost via pruning, the parameters that remain (after pruning) must be subject to a certain amount of re-training. To identify the point in time during training as to 'when' sparsity and re-training should be done, we visualized the gradients of the total loss values during the training and validation phase.



(a) (b) (c)

Fig. 4. Results of Hybrid Training: (a) Training Loss Comparison Between Dense and Sparse DDNet, (b) Progression of Gradients of the Total Loss During Training, (c) Accuracy with Different Hybrid Schedules: Dense Epochs Followed by Sparse Epochs.

Fig. 4(b) shows a logarithmic plot for the progression of the gradients of the total (training and validation) loss values for a dense DDNet model. The figure shows that the change in the gradients spans multiple orders of magnitude across the first 28 epochs of training (red dotted lines). Thereafter, the loss values continue to diminish but remain within the same order of magnitude, indicating that the overall loss value is approaching a global minima.

Removing parameters until the gradients saturate will hurt the model's accuracy. Thus, we propose a 'hybrid training schedule' wherein dense training is followed by pruning (i.e., sparsity) and then re-training, thus preserving accuracy while improving performance. Using such a hybrid schedule, Fig. 4(c) shows the variation in accuracy (i.e., MS-SSIM) with different combinations of dense (X-axis below) and sparse epochs (X-axis above) that total 50 epochs. The horizontal black dashed line represents the accuracy corresponding to complete dense training (baseline). The vertical green dashed line highlights the point having an ideal balance of dense and sparse epochs that results in the same accuracy.

3 Architecture-Aware Optimizations

To complement the sparsity optimizations, we propose three architecture-aware optimizations to fully utilize the underlying Nvidia Ampere GPU hardware.

3.1 DoLL: Efficient Data-Loader for Small Datasets

Distributed data parallelism (DDP) scales the training of DL models that utilize large datasets. In PyTorch, DDP capabilities are supported by a Distributed Data Loader (DDL) library that prefetches data to the GPUs via multi-threaded worker processes in the background. However, with small datasets, DDL is inefficient for two reasons: (1) worker threads can sit idle after working on their corresponding chunk of the dataset (or mini-batch) and consume resources for the rest of the training period and (2) operations on the respective mini-batches, e.g., index distribution, batch sample preparation, and transformations, occur on the CPU while training occurs on the GPU. To remediate these issues, we design an efficient data-loader for small datasets, i.e., DoLL.

Architecture of DoLL Fig. 5 shows the DoLL architecture, which uses the large GPU memory by staging the entire dataset on it, in parallel, for each replica process. Unlike DDL, DoLL does *not* initialize communication queues and inter-process communication (IPC) for index distribution or data preparation; instead, it leverages the NVIDIA Collective Communications Library (NCCL) so data is directly communicated between GPUs within each node over the NvLink interconnect, thus bypassing the CPU and PCIe bus. While staging the entire dataset on GPU memory takes significantly longer than moving a small mini-batch sample, the overall cost is significantly less than moving small mini-batches from CPU to GPU repeatedly during each training epoch. As a result, *more* operations can now be performed on the *faster* GPU with DoLL.

3.2 Mixed Precision and Tensor Cores

Mixed-precision training leverages a combination of higher- and lower-precision storage formats to accelerate computation in the training process by utilizing higher clock speeds for arithmetic operations in lower precision. In the forward pass, tensor cores perform matrix-multiply-add (MMA) operations in TF32 format [20] for convolution, as shown in Fig. 6(a), and deconvolution.

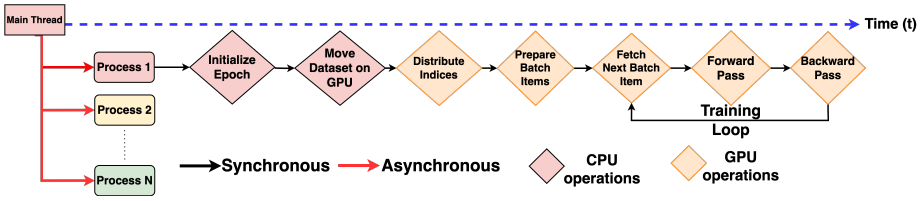
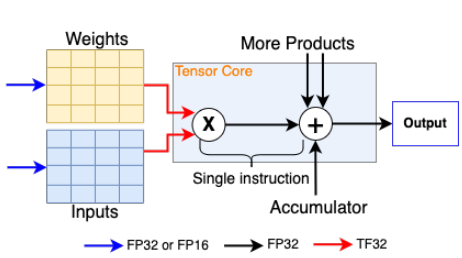
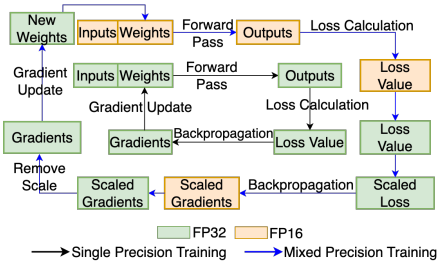


Fig. 5. Architecture of the new data loader, i.e., DoLL.



(a) Convolution operation in mixed precision on tensor cores.



(b) Backward pass in single and mixed precision.

Fig. 6. Mixed precision: (a) convolution in the forward pass and (b) backward pass.

Performing the entire MMA operation in mixed precision delivers high performance for the following two reasons: (1) tensor cores perform fused-matrix-multiply-add (FMMA) operations in a single clock cycle, delivering more throughput than a CUDA core, which takes two clock cycles to complete an FMMA, and (2) using a specialized format, TF32, the FMMA operations require less memory because of the reduced number of mantissa bits.

For the backward pass, loss calculations and gradient updates are computed in half-precision with values scaled with a scalar value. Simultaneously, the original values are stored and respectively updated in higher single-precision (with the same scale factor) to preserve accuracy. Fig. 6(b) contrasts the training workflow between single precision (black arrows) and mixed precision (blue arrows).

3.3 Graph Capture Optimization

Deep learning (DL) frameworks often use a loose coupling of low-level hardware-specific binaries (written in CUDA, HIP, C, and C++) for performance-critical operations with user-friendly APIs for programming productivity. This results in reduced performance and increased overhead, e.g., CUDA kernel launches.

In PyTorch, the CUDA kernel launch overhead is the latency experienced due to the repetitive launch of the same CUDA kernels (on Nvidia GPUs) to compute layers in the DL model during the training iterations. To address this, we leverage the CUDA Graphs API with PyTorch. Fig. 7 illustrates the optimization through an example of a DL model with two layers undergoing two training epochs with

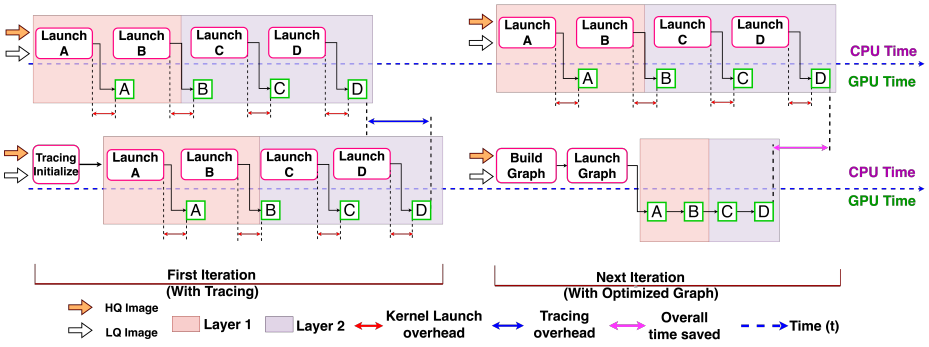


Fig. 7. An example demonstrating graph capture optimization (GCO).

(below) and without (above) our graph capture optimization (GCO). The two layers, Layer 1 and Layer 2, launch two GPU kernels, each containing (A, B) and (C, D), respectively, via the CUDA backend API; in reality, these layers may launch multiple GPU kernels. The GCO initializes an alternative CUDA stream to capture the runtime information of the DL workflow; this stage is called 'tracing.' The new CUDA stream records information about the control and data path in the main CUDA stream (where training happens), such as the execution order of the GPU kernels (A, B, C, and D), their input parameters, their sizes, and data types. As a consequence, tracing consumes additional GPU memory that is needed for secondary buffers to match the size, data type, and dimensions of the input and output tensors of each GPU kernel. As a result, the entire iteration is slower due to the 'tracing overhead' (blue arrow).

Once the tracing finishes, the alternative stream holds a serialized version of a static CUDA graph, which has the same data path and execution order of GPU kernels as the main CUDA stream. Then, at the start of the next epoch, PyTorch's just-in-time (JIT) compiler uses the captured information and instantiates a static CUDA graph containing the GPU kernels in the main CUDA stream. The alternate stream and memory buffers are discarded, and a static CUDA graph is launched on the GPU scheduler. The launched CUDA graph is replayed for the rest of the training duration, mitigating the need to repeatedly launch and destroy GPU kernels at each epoch. Thus, the subsequent epochs run faster, amortizing the latency of the tracing in the initial epochs.

4 Results and Analysis

In this section, we evaluate the *accuracy* of our optimizations with respect to MS-SSIM (multi-scale structural similarity index) [25] and *performance* with respect to total time to train DDNet with a target MS-SSIM value of 97.22 ± 1.49. We construct a dataset of chest CT scans from four public biomedical data sources: Mayo Clinic [18], BIMCV Medical Imaging Databank of the Valencia Region, MIDRC: Medical Imaging and Data Resource Center [19] and Lung

Image Database Consortium (LIDC) Image Collection. These radiological data sources contain 3D chest CT scans composed of 2D image slices, each of size 512× 512 pixels. The training setup for the dense and sparse DDNet hyperparameters is as follows: 50 epochs for training, batch size = 32, learning rate = 0.0001, and decay rate = 0.95. The original (dense) DDNet took 79 minutes to converge to the target MS-SSIM value, which, in turn, we use as our reference training time.

4.1 Sparse Optimizations

Table 2 shows the speedup and accuracy drop using different sparse techniques. With a hybrid training schedule of 25 dense epochs and 25 sparse epochs, all the sparsity optimizations provide a similar speedup of 1.14× with no loss in accuracy because all three sparse optimizations target only the same convolution and deconvolution layers in the DDNet model. Moreover, all optimizations prune 50% of entries in the weights and bias tensors associated with these layers to benefit from the 2:4 sparsity support in the 2nd generation of tensor cores on the Nvidia Ampere GPU [20]. With the same 50% sparsity in all three optimization techniques, an equal number of effective parameters translates to an equal number of FMMA operations in all sparse techniques. As a result, the final sparse models show the same degree of optimization and training time reduction.

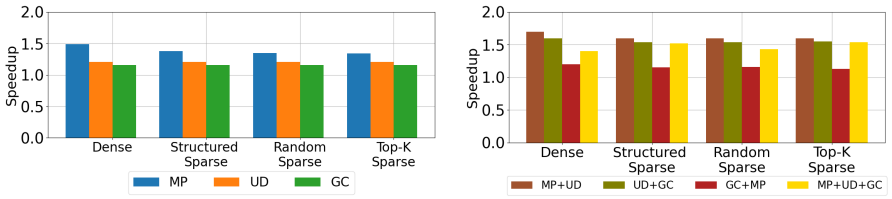
Sparsity Type	**Speedup**	**MS-SSIM**	**%Accuracy Drop**
Structured	1.14×	97.75 ± 1.62	0
Random Unstructured	1.14×	97.30 ± 2.17	0
Top-K	1.14×	96.98 ± 2.02	0.02

Table 2. Speedup vs. accuracy with different sparse optimizations.

4.2 Architecture-Aware Optimizations

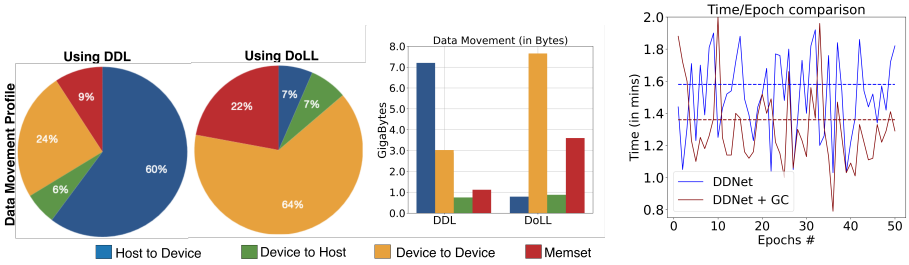
We apply and evaluate our architecture-aware optimizations individually for both sparse and dense models and then in concert. First, mixed precision delivers a speedup of 1.49× for the dense model and 1.34× for the sparse models (see Fig. 8(a)). Additionally, with mixed precision, the memory consumption of the two models reduces from 62 GB to 27 GB because using lower precision (TF32 in the forward pass and half-precision in the backward pass) requires fewer bytes.

Second, DoLL provides a constant speedup of 1.21× (see Fig. 8(a)) for the dense and sparse models since the amount of training data is the same for both models. Fig. 9(a) compares the profiles for data movement using two data loaders: PyTorch's DDL and our DoLL. With DDL, 60% of all data movement is due to the repetitive CPU-to-GPU, i.e., host-to-device (H2D), transfers of incremental portions of the dataset. DoLL reduces this H2D transfer overhead from 60% down to 7% by moving the entire dataset to the GPU before training starts, thus minimizing H2D transfers. In terms of the amount of data moved between



(a) Individual optimizations. (b) All optimizations in concert.

Fig. 8. Speedup with combinations of optimizations. MP: Mixed precision, UD: Using DoLL, GCO: Graph capture optimization.



(a) (b)

Fig. 9. Improvement with architecture-aware optimizations. (a) Data movement profile of DDNet by percentage and total bytes, (b) per epoch time comparison with GCO.

the CPU and GPU. With DoLL, the H2D transfer is less than 1 GB, which is in contrast to the 7 GB with DDL. However, the DL model now consumes more GPU memory, as expected and shown in Fig. 9(a), where the data movement for device-to-device increases from 24% to 64% and memset from 9% to 22%.

Third, graph capture optimization (GCO) delivers a 1.16× speedup. Fig. 9(b) shows a per-epoch time comparison of the baseline DDNet model (blue line) to one with GCO (maroon line). Due to the tracing overhead, the initial five epochs are slower, but the overall mean per epoch time is faster.

When all three optimizations are combined (see Fig. 8(b)), we observe an overall speedup of 1.7× for the baseline model and 1.6× each, for the three sparse models. A concerted overall speedup for all the optimizations is not realized because GCO does not work well with mixed precision. When GCO is combined with mixed precision, the data types and the size of the input tensors in GPU kernels vary at each iteration due to the mixed precision. As a result, the graph optimization skips such GPU kernels from the CUDA graph, and the CUDA kernel launch latency for such GPU kernels is thus not mitigated.

5 Related Work

This section presents related work from (1) deep learning-based CT image enhancement and (2) sparsity in deep learning (DL). Li et al. [16] present an ex-

tensive survey of DL-based image post-processing techniques for improving CT image quality. DNN architectures, such as auto-encoders [22], deep convolution neural networks (CNNs) [4], generative adversarial networks (GANs) [26], and transformers [28] have shown potential in improving CT image quality. However, these architectures generally ignore the computational cost of training and the hardware support needed in the CT equipment to deploy such DNNs.

To address the above, *sparse optimizations* reduce the complexity of DL models, including CNNs [3], GANs [24] and transformer-based neural networks [5]. Hoefler et al. present an extensive survey on sparsity in deep learning [13].

6 Conclusion

With the increasing use of deep learning (DL) in the biomedical domain, researchers need to focus on both the accuracy *and* performance of DL models. While an accurate model is essential, the high computational cost of training such a model limits its accessibility. Therefore, this work presents a combination of architecture-aware and sparse optimizations for DL models in the biomedical domain. Our architecture-aware optimizations deliver a speedup of up to 1.7× for the baseline (dense) DDNet model without losing any accuracy. By introducing a hybrid dense+sparse training schedule to the aforementioned architecture-aware optimizations, we achieve an additional 1.14× speedup, resulting in an aggregate speedup of 1.94× while maintaining accuracy.

References

1. Bian, J., Siewerdsen, J.H., Han, X., Sidky, E.Y., Prince, J.L., Pelizzari, C.A., Pan, X.: Evaluation of sparse-view reconstruction from flat-panel-detector cone-beam ct. Physics in Medicine and Biology **55**(22), 6575–6599 (2010)
2. Brenner, D.J., Hall, E.J.: Computed tomography — an increasing source of radiation exposure. New England Journal of Medicine **357**(22), 2277–2284 (2007)
3. Changpinyo, S., Sandler, M., Zhmoginov, A.: The power of sparsity in convolutional neural networks (2017)
4. Chen, H., Zhang, Y., Zhang, W., Liao, P., Li, K., Zhou, J., Wang, G.: A low-dose ct via convolutional neural network. Biomedical Optics Express **8**(2), 679 (2017)
5. Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers (Apr 2019)
6. Frankle, J., Dziugaite, G.K., Roy, D.M., Carbin, M.: Pruning neural networks at initialization: Why are we missing the mark? In: 9th Int'l. Conf. on Learning Representations (May 2021)
7. Friston, K.: Hierarchical models in the brain. PLoS Comp. Biology **4**(11) (2008)
8. Goel, G., Gondhalekar, A., Qi, J., Zhang, Z., Cao, G., Feng, W.: Computecovid19+: Accelerating covid-19 diagnosis and monitoring via high-performance deep learning on ct images. In: Proc. of the 50th Int'l Conf. on Parallel Processing. Association for Computing Machinery (2021)
9. Gong, W., Yao, Y., Ni, J., Jiang, H., Jia, L., Xiong, W., Zhang, W., He, S., Wei, Z., Zhou, J.: Deep learning-based low-dose ct for adaptive radiotherapy of abdominal and pelvic tumors. Frontiers in Oncology **12** (2022)

10. Gupta, R.K., Bharti, S., Kunhare, N., Sahu, Y., Pathik, N.: Brain tumor detection and classification using cycle generative adversarial networks. Interdisciplinary Sciences: Computational Life Sciences **14**(2), 485–502 (2022)
11. Hasan, N., Bao, Y., Shawon, A., Huang, Y.: Densenet convolutional neural networks application for predicting covid-19 using ct image. SN Computer Science **2**(5) (2021)
12. Herculano-Houzel, S., Mota, B., Wong, P., Kaas, J.H.: Connectivity-driven white matter scaling and folding in primate cerebral cortex. Proc. of the Nat'l Academy of Sciences **107**(44), 19008–19013 (2010)
13. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks. J. Mach. Learn. Res. **22**(1) (Jan 2021)
14. Jiang, B., Li, N., Shi, X., Zhang, S., Li, J., de Bock, G.H., Vliegenthart, R., Xie, X.: Deep learning reconstruction shows better lung nodule detection for ultra–low-dose chest ct. Radiology **303**, 202–212 (2022)
15. Lei, Y., He, X., Yao, J., Wang, T., Wang, L., Li, W., Curran, W.J., Liu, T., Xu, D., Yang, X.: Breast tumor segmentation in 3d automatic breast ultrasound using mask scoring r-cnn. Medical Physics **48**(1), 204–214 (2020)
16. Li, D., Ma, L., Li, J., Qi, S., Yao, Y., Teng, Y.: A comprehensive survey on deep learning techniques in ct image quality improvement. Medical and Biological Engineering and Computing **60**(10), 2757–2770 (2022)
17. Liu, S., Chen, T., Chen, X., Shen, L., Mocanu, D.C., Wang, Z., Pechenizkiy, M.: The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. arXiv preprint arXiv:2202.02643 (2022)
18. McCollough, C. et al: Data from Low Dose CT Image and Projection Data. (2021)
19. Medical Imaging & Data Resource Ctr.: (2021-04-09), https://www.midrc.org/
20. Mishra, A., Latorre, J.A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., Micikevicius, P.: Accelerating sparse deep neural networks (2021)
21. Nasrullah, N., Sang, J., Alam, M.S., Mateen, M., Cai, B., Hu, H.: Automated lung nodule detection and classification using deep learning combined with multiple strategies. Sensors **19**(17), 3722 (2019)
22. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation (May 2015)
23. Sanaat, A., Shiri, I., Arabi, H., Mainta, I., Nkoulou, R., Zaidi, H.: Deep learning-assisted ultra-fast/low-dose whole-body pet/ct imaging. European Journal of Nuclear Medicine and Molecular Imaging **48**(8), 2405–2415 (2021)
24. Wang, Y., Wu, J., Hovakimyan, N., Sun, R.: Double dynamic sparse training for gans (Feb 2023)
25. Wang, Z., Simoncelli, E., Bovik, A.: Multiscale structural similarity for image quality assessment. The 37th Asilomar Conf. on Signals, Systems; Computers, 2003
26. Wolterink, J.M., Leiner, T., Viergever, M.A., Išgum, I.: Generative adversarial networks for noise reduction in low-dose ct. IEEE Tran. on Medical Imaging **36**(12), 2536–2545 (2017)
27. Zhang, Z., Liang, X., Dong, X., Xie, Y., Cao, G.: A Sparse-View CT Reconstruction Method Based on Combination of DenseNet and Deconvolution. IEEE Tran. on Medical Imaging **37**(6), 1407–1417 (Jun 2018)
28. Zhao, J., Hou, X., Pan, M., Zhang, H.: Attention-based generative adversarial network in medical imaging: A narrative review. Computers in Biology and Medicine **149**, 105948 (2022)