A 3D Deep Learning Architecture for Denoising Low-Dose CT Scans

Armen Kasparian[1], Guohua Cao[2], and Wu-chun Feng[1]

[1] Dept. of Computer Science, Virginia Tech — {armen,wfeng}@vt.edu
[2] School of Biomedical Engg., ShanghaiTech U. — caogh@shanghaitech.edu.cn

Abstract. Low-dose computed tomography (LDCT) scans reduce the radiation dose of computed tomography (CT) scans but come at the expense of image quality. Deep-learning (DL) image denoising techniques can enhance these LDCT images to match the quality of their regular-dose CT counterparts. To achieve better denoising performance than the current state of the art, we present a novel 3D DL architecture for LDCT image denoising called 3D-DDnet. The architecture leverages the inter-slice correlation in volumetric CT scans to obtain better denoising performance and employs distributed data parallel (DDP) strategies along with transfer learning to achieve faster training. The DDP training strategy enables a scalable multi-GPU approach on Nvidia A100 GPUs, which allows the training of previously prohibitively large volumetric samples. Our results show that 3D-DDnet achieves 10% better mean square error (MSE) on LDCT scans than its 2D predecessor (i.e., 2D-DDnet). In addition, the transfer learning in 3D-DDnet leverages existing trained 2D models to "jump start" the weights and biases of our 3D DL model and reduces training time by 50% while maintaining accuracy.

Keywords: deep learning · distributed data parallel · transfer learning · computed tomography · image enhancement

1 Introduction

Radiologists rely on non-invasive CT scans to obtain images of internal organs for the diagnosis and treatment of patients [1]. Unfortunately, it exposes the patient to significant radiation. Fazel et al. found that 75.4% of patients' total effective radiation dose in the USA can be attributed to CT and nuclear medicine scans and used 50 millisieverts (mSv) as the "high" exposure range for radiation [2]. A standard chest CT scan exposes patients to 4.55 mSv of radiation. With the need to minimize radiation exposure, LDCT scans have emerged to reduce radiation exposure to a mere 0.5 mSv but at the expense image fidelity [3].

Both statistical methods and deep learning-based methods have been used to combat the noise and artifacts found in lower-quality LDCT scans. For example, statistical model-based iterative reconstruction (MBIR) enhances image quality and reduces noise in CT imaging Deep learning (DL) architectures like RED-CNN [4] and DDnet [5] learn features from standard-dose CT scans and use that

information to denoise LDCT scans. These DL architectures focus on denoising these three-dimensional (3D) CT scans via two-dimensional (2D) slices as 3D denoising has been computationally impractical. Enabling the transition from today's 2D-slice denoising architectures to a 3D architecture requires training of a parameter space that is nearly 5× larger, i.e., 4,712,723 vs. 1,021,619.

Our 3D denoising architecture, 3D-DDnet, consists of two main components: (1) distributed data parallel training to improve performance and (2) data loader modifications to improve accuracy. The parallel training approach uses a volumetric representation of CT scans by selectively choosing the slices from the dataset. The number of slices selected for a volume can now be explored as a hyperparameter to fit into GPU VRAM. The increase in GPU VRAM size allows larger sample volumes to be offloaded from CPU to GPU for faster training.

2 Related Work

2D vs. 3D Deep-Learning Architectures. Existing 3D deep-learning (DL) architectures for CT imaging have explored segmentation, classification, and detection techniques versus their 2D counterparts. Avesta et al. compare 2D and 3D U-Net-based DL architectures for image segmentation in brain MRIs [6] and show that 3D models need 20× more memory than 2D models. The 3D models also converge to a more accurate state but at the expense of significantly more computational overhead due to the extra dimension. Our 3D-DDnet approach reduces this increased computational overhead of 3D approaches while improving the accuracy of state-of-the-art U-Net-based models.

Challenges with 3D Deep-Learning Networks. Singh et al. find that relative to classification, segmentation, and detection for 3D deep-learning networks, common challenges include increased difficulty in training and hyperparameter tuning and the inability to utilize smaller datasets [7]. Crespi et al. study the transition from 2D convolution neural networks (CNN) to 3D CNNs and find the larger dimension space problematic due to the lack of available datasets [8]. In contrast, our 3D-DDnet approach proposes a data-loading strategy that creates 3D volumes from existing 2D slice-based CT scan datasets. This strategy reduces the overhead of curating a custom dataset by introducing the ability to reuse previous datasets designed for 2D CT scan denoising.

Impact of Multislice Inputs on Accuracy. Multislice inputs for 3D CNN noise reduction have previously been explored on the accuracy front. Zhou et al. show that the improved accuracy of a 3D network comes at the expense of longer training time per epoch [9]. The 2D network takes 25 minutes to train per epoch while the 3D equivalent takes 270. In addition, due to hardware limitations, their data loader is limited to sampling 64×64 patches of the source 512×512 slices. An earlier implementation of 3D-DDnet [10] was also limited by hardware, resulting in the use of a sample patching and stitching technique.

3 Approach and Design of 3D-DDnet

Increasing the dimensions of an architecture from 2D to 3D dramatically increases the parameters of the network (in our case, from 1,021,619 to 4,712,723), allowing for more feature extraction but at the expense of more computation [11]. To alleviate these costs, we present an overhaul to the training strategy and enhancements to the data loader for our 3D-DDnet architecture.

3.1 Architecture

The 3D-DDnet architecture extends our previous "2D DenseNet and Deconvolutional neural network" (i.e., DDnet) [5], as shown in Fig. 1. This extended architecture generalizes more information from the source data by using the correlations found between slices and delivers better accuracy.

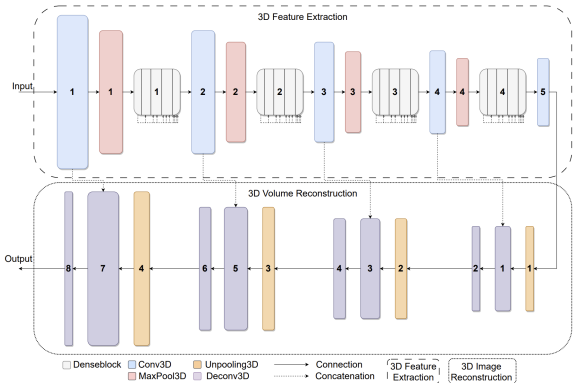


Fig. 1: Architecture of 3D-DDnet. Layer numbers correspond to Table 1.

The convolution layers in the upper half of 3D-DDnet now consist of 3D dense blocks and 3D convolutions. The 3D dense blocks contain internal 3D convolution layers that are followed by 3D max pooling layers, which work like their 2D counterparts but reduce the latent space by a factor of two in the x, y, and z dimensions of the sample volume.

Table 1 shows the input and output sizes of each network layer, where a volume consists of 32 slices. The architecture allows for the number of slices utilized in a sample to be a hyperparameter. This hyperparameter is then set at runtime, facilitating further research into the effects of different volume sizes.

3.2 Datasets

The data used to train and test 3D-DDnet came from three sources: (1) Mayo Clinic with 100 healthy CT scans at full and quarter dosage, (2) Lung Image Database Consortium (LIDC) with 722 CT scans, and (3) Medical Imaging Databank of the Valencia region (BIMCV) with 397 CT scans. If the dataset did *not* contain LDCT images, we ran the high-dose images through an algorithm to simulate LDCT scans. These numerically simulated low-dose images were then paired with the corresponding high-dose CT images to create a uniform dataset containing source and target images. All the CT images from the dataset are of size 512×512 and rotated to be the same orientation.

Table 1: Input/output sizes and filter sizes of the layers in 3D-DDnet

Layers	Output Size	Details
Conv3D 1	$512 \times 512 \times 32 \times 16$	Filter size=$7 \times 7 \times 7$, Stride=1
MaxPool3D 1	$256 \times 256 \times 16 \times 16$	Filter size=$3 \times 3 \times 3$, Stride=3
DenseBlock3D 1	$256 \times 256 \times 16 \times 80$	Filter size=$(5 \times 5 \times 5) \times 4$
Conv3D 2	$256 \times 256 \times 16 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=1
MaxPool3D 2	$128 \times 128 \times 8 \times 16$	Filter size=$3 \times 3 \times 3$, Stride=3
DenseBlock3D 2	$128 \times 128 \times 8 \times 80$	Filter size=$(5 \times 5 \times 5) \times 4$
Conv3D 3	$128 \times 128 \times 8 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=1
MaxPool3D 3	$64 \times 64 \times 4 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=2
DenseBlock3D 3	$64 \times 64 \times 4 \times 80$	Filter size=$(5 \times 5 \times 5) \times 4$
Conv3D 4	$64 \times 64 \times 4 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=1
MaxPool3D 4	$32 \times 32 \times 2 \times 16$	Filter size=$3 \times 3 \times 3$, Stride=2
DenseBlock3D 4	$32 \times 32 \times 2 \times 80$	Filter size=$(5 \times 5 \times 5) \times 4$
Conv3D 5	$32 \times 32 \times 2 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=1
Unpooling3D 1	$64 \times 64 \times 4 \times 16$	Scale factor = 2
Deconv3D 1	$64 \times 64 \times 4 \times 32$	Filter size=$5 \times 5 \times 5$, Stride=1
Deconv3D 2	$64 \times 64 \times 4 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=1
Unpooling3D 2	$128 \times 128 \times 8 \times 16$	Scale factor = 2
Deconv3D 3	$128 \times 128 \times 8 \times 32$	Filter size=$5 \times 5 \times 5$, Stride=1
Deconv3D 4	$128 \times 128 \times 8 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=1
Unpooling3D 3	$256 \times 256 \times 16 \times 16$	Scale factor = 2
Deconv3D 5	$256 \times 256 \times 16 \times 32$	Filter size=$5 \times 5 \times 5$, Stride=1
Deconv3D 6	$256 \times 256 \times 16 \times 16$	Filter size=$1 \times 1 \times 1$, Stride=1
Unpooling3D 4	$512 \times 512 \times 32 \times 16$	Scale factor = 2
Deconv3D 7	$512 \times 512 \times 32 \times 32$	Filter size=$5 \times 5 \times 5$, Stride=1
Deconv3D 8	$512 \times 512 \times 32 \times 1$	Filter size=$1 \times 1 \times 1$, Stride=1

3.3 Data Loaders

2D data loaders supply batches of image slices to models, with a batch size of one meaning one image per epoch for training. Larger batch sizes send more slices per epoch. 3D data loaders, on the other hand, send volumes—stacked sets of 2D slices—as single samples. Batches with 3D loaders contain multiple volumes processed separately, increasing computational demands. For instance, a 512×512 slice requires 1.5 GB of memory, while a 512×512×16 volume needs nearly 20 GB. This memory constraint makes volume slice selection critical, determined by both desired accuracy and hardware capabilities. Our paper introduces and evaluates two data loader types (see Fig. 2): strided and grouped.

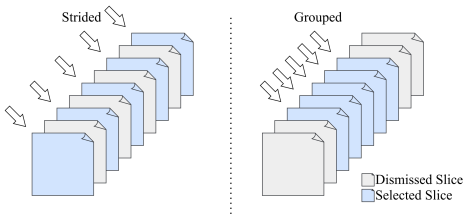


Fig. 2: Strided vs. grouped data loader. A selection of a five-slice volume from a seven-slice sample with a stride length of size two.

Strided Data Loader The strided data loader selects slices at uniform intervals. The strided loader's advantage is its consistent slice selection across the entire CT scan, useful when crucial data is spread throughout. However, it uniformly values all slices, which might not be ideal. For example, the initial and final slices of LDCT may offer limited relevant data; yet, this loader picks slices uniformly, potentially including less valuable ones for training.

Center-Grouped Data Loader The center-grouped data loader minimizes the distance between slices selected for the volume to help the model learn the correlation between slices in the third dimension. This data loader works by selecting a group of slices centered around the middle of the sample. The main benefit of this data loader is that the selected slices for the volume contain the highest correlation between them.

3.4 Training Strategy

Current datacenter GPUs offer up to 80GB of HMB2e VRAM, a significant upgrade over the previous 32GB of HBM2, thus supporting the loading of larger datasets onto GPUs and the shift from 2D slices to 3D volumetric data. For context, a single 512×512 slice occupies about 1.5GB, while 16-slice and 32-slice volumes require 20GB and 40GB, respectively, underscoring the demand for enhanced GPU capacity.

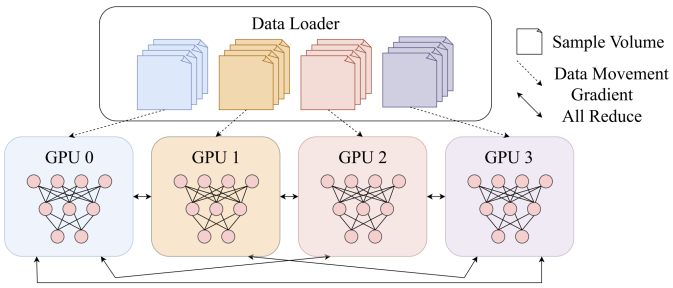


Fig. 3: Overview of the distributed data parallel architecture. The data loader individually sends unique batches of data to each GPU while each of the models loaded onto the GPUs are synchronized during the gradient, all-reduce step found in backpropagation. The models receive different data from the data loader but average the gradients per epoch, leaving the models synchronized.

Past generations of datacenter GPUs could not even load a single sample volume onto the GPU. With this hardware limitation now lifted, we need to be smarter with how we load the dataset onto the GPU. Since we are unable to load *multiple* sample volumes onto a single GPU, our approach to solve this problem uses a distributed data parallel (DDP) approach. Distributed data parallelism (DDP) enables data parallelism at the model level. It allows for a synchronized model on each device while different data is passed for training during each epoch. This approach, built into PyTorch, works by creating a replica of the model architecture on each GPU. For each epoch, each replica model running on its corresponding GPU receives a different batch of data to run through the forward pass. Gradients are then computed locally for each process. These local gradients are then synchronized during the backward pass with an all-reduce gradient synchronization that calculates the mean of all the gradients across all the processes. These average gradients are then distributed to the individual processes for the backward pass. After this backward pass completes, all the models across each of the individual processes are the same and prepared for the

next epoch and batch of data. A visualization of how the data loader works in tandem with the synchronized models can be seen in Fig. 3.

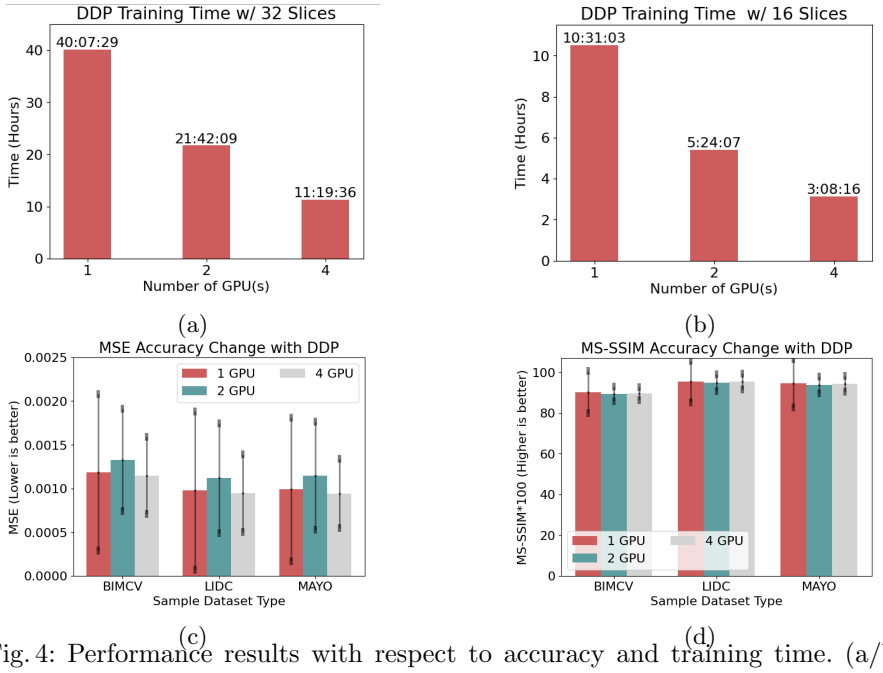


Fig. 4: Performance results with respect to accuracy and training time. (a/b) Show training time scaling with respect to the number of GPUs. (c/d) Show accuracy variance between datasets with respect to the number of GPUs.

Data Parallelism with Batches When using the DDP training strategy, we need to understand how it affects the batch size and learning during training. With DDP, we modify the batch size to be an effective batch size that relates to the number of processes being run. This effective batch size is equivalent to the local batch size (the number of samples being processed per epoch per process) multiplied by the number of processes (GPUs) being utilized.

In our application with volumetric CT image enhancement, this now removes the VRAM limitation of the GPU and increases the previous maximum batch size from one to equal the number of GPUs available for training. It is important to note that this is only possible since we can load, at minimum, a single 3D volumetric sample onto the individual GPU. Then, by utilizing multiple GPUs, we can increase our effective batch size and scale with data parallelism.

The performance speedup shown in Fig. 4.(a/b) is due to the data parallelism leveraged by utilizing the DDP training algorithm. The use of DDP also facilitates reasonable, effective batch sizes that allow for faster training times while maintaining accuracy, as shown in the Fig. 4.(c/d).

When looking at Fig. 4.(c/d), we can see the variance bands continually decrease as more GPUs are utilized during training. Using more GPUs in distributed data parallel training reduces variance in model training by allowing for larger batch sizes and more consistent gradient estimates, leading to smoother, more stable updates and better generalization. This is achieved through parallel processing, averaging gradients across GPUs, and efficient utilization of computational resources.

3.5 Transfer Learning

Leveraging pre-existing 2D networks can offset the computational challenges of transitioning to 3D networks. Transfer learning applies the weights and biases of a pre-trained model to new settings, providing an advantage in training. Specifically for 3D networks, it enhances efficiency by strategically employing the 2D architecture's weights and biases. Numerous 2D models in the biomedical sector allow us to harness their computational investment to amplify the 3D counterparts through the reuse of older network structures.

Typically, in transfer learning, trained layers from a donor network are integrated directly into a recipient network, ensuring the pre-learned parameters initiate the recipient's training. However, this is feasible only when both architectures have matching dimensions. When dimensions differ, as in our study, an adapted transfer learning strategy is needed. Transferring from 2D to 3D involves embedding the weights of a 2D image kernel into a 3D volume kernel, as visualized in Fig. 5.

Unoccupied values in the 3D kernel are filled with zeros from the pretrained 2D kernel. We tested alternative fill methods, such as consistently using the 2D kernel for the entire 3D space or introducing random noise. All variations displayed comparable performance, with negligible differences in metrics like MSE and MS-SSIM.

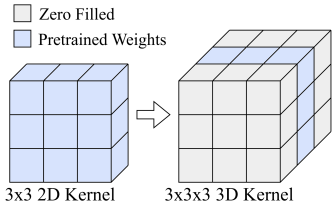


Fig. 5: Transfer learning diagram showing insert of 2D kernel into 3D kernel

4 Evaluation Metrics for the Loss Function

The loss function of the neural network dictates the metrics that backpropagation is trying to enhance. Mean squared error (MSE) is a direct pixel-to-pixel comparison of the differences between two images or volumes – see Eq. (1). The MSE equation iterates over N pixels, where X is the source image, and Y is the image after being denoised.

$$MSE(X,Y) = \frac{1}{N}\sum_{n=1}^{N}[X-Y]^2, \tag{1}$$

The structural similarity (SSIM) index, shown in Eq. (2), measures the similarity between two images based on their structural information in the spatial domain [12]. The SSIM index considers luminance $\mathrm{l}(X,Y)$, contrast $\mathrm{c}(X,Y)$, and structure $\mathrm{s}(X,Y)$ and returns a value between -1 and 1, where a value of 1 indicates perfect similarity. In Eq. 2, μ_x and μ_y are the mean intensities of windows, σ_x^2 and σ_y^2 represent the variances of windows, σ_{xy} is the covariance between windows which provides a measure of the strength and direction of the relationship between two sets of variables, $C1$ and $C2$ are constants added to stabilize for the dynamic range of the images where $C1 = (k_1 L)^2$ and $C2 = (k_2 L)^2$ where L is the dynamic range of the images and k_1 and k_2 are small constants (set to default values $k_1 = 0.01$ and $k_2 = 0.03$).

$$\mathrm{SSIM}(X,Y) = \frac{(2\mu_x\mu_y + C1)}{(\mu_x^2 + \mu_y^2 + C1)} \times \frac{(2\sigma_{xy} + C2)}{(\sigma_x^2 + \sigma_y^2 + C2)} = \mathrm{l}(X,Y) \times \mathrm{cs}(X,Y) \quad (2)$$

The multi-scale structural similarity index (MS-SSIM), defined in Eq. (3), is built upon the SSIM by images considering multiple scales (M and j) to better capture structural similarities across different levels of detail. MS-SSIM divides the image or volume into multiple smaller sub-regions and computes the SSIM value at different levels of image detail [13]. Like SSIM, MS-SSIM ranges from -1 to 1, with higher values indicating a higher similarity between the two samples. α and β_j are weights for the luminance, contrast, and structure terms. These weights are set to one to equally weigh all the terms.

$$\text{MS-SSIM}(X,Y) = l_M^{\alpha}(X,Y) \cdot \prod_{j=1}^{M} cs_j^{\beta_j}(X,Y) \quad (3)$$

Eq. (4) presents a composite loss function, $Loss(X,Y)$ that combines the higher-quality MS-SSIM with the MSE. This allows us to have a loss function where the MSE focuses on pixel-wise accuracy and the MS-SSIM emphasizes preserving structural and perceptual quality. Combining them offers a holistic approach, ensuring both pixel-level precision and high perceptual quality in the denoised images. In our loss function, a γ of 0.1 is selected to balance the magnitude of these two terms properly.

$$\mathrm{Loss}(X,Y) = \ \mathrm{MSE}(X,Y) + \gamma(1 - \text{MS-SSIM}(X,Y)), \quad (4)$$

5 Results

Here we present both the quantitative results of 3D-DDnet with respect to accuracy and the qualitative results of 3D-DDnet with respect to the CT images and the difference maps between the original high-quality CT image and the denoised CT images produced by 2D-DDnet and 3D-DDnet. These results are run with the Adam optimizer and a learning rate of 0.001. An ablation study is done between the different optimizations which can be seen in the quantitative results below.

5.1 Quantitative Results

The baseline 2D-DDnet model sets the benchmark for our transition to 3D-DDnet. Training on a single GPU for about 8 hours with a batch size of one, it achieves an average MSE of 0.003998 and an MS-SSIM of 0.788877. In contrast, our 3D-DDnet variants show improvements in both MSE and MS-SSIM, highlighting the benefits of the 3D approach.

Comparisons between using 32 and 16 slices per volume in 3D-DDnet (Fig. 6 and Table 2) reveal negligible differences (± 1) in performance, suggesting that using more than 16 slices may lead to unnecessary computational expense. Therefore, for optimal efficiency and accuracy, a 16-slice volume is recommended.

Speed enhancements are evident with the distributed data parallel strategy. Scaling from 1 to 2 GPUs nearly doubles the speed, while increasing from 2 to 4 GPUs offers a 1.75× speedup.

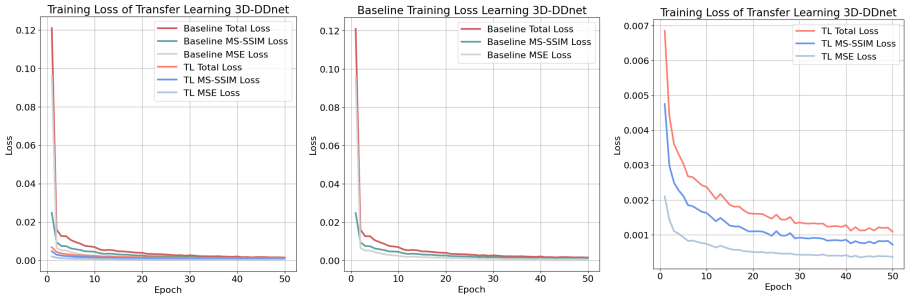
To verify that parallel architecture doesn't compromise the model's stability, we examined loss curves for single and multi-GPU setups (Fig. 7). The loss consistently decreases over 50 epochs, indicating stable training. Notably, transfer learning provides a substantial acceleration in learning, allowing the model to rapidly approach optimal loss values. In instances where surpassing the 2D model's performance is the goal, training can be shorter. At 15 epochs, the 3D model already outperforms its 2D counterpart. The loss function plateaus at 50 epochs (Fig. 7(c)), signifying that the network has reached its optimal point.

Fig. 6: Training loss similarities between the 16- and 32-slice volume data loader

Table 2: Quantitative Results showing the training time, average MSE, and average MS-SSIM of testing. The architecture section contains GDL (Grouped Data Loader), SDL (Strided Data Loader), and TL (Transfer Learning) optimizations present. Batch Size = GPU Count

Architecture	Epochs	Slice Count	Batch Size	Training Time (H:M:s)	Average MSE	Average MS-SSIM
2D-DDnet	50	N/A	1	8:44:32	0.0039±0.0015	0.7888±0.0857
3D-DDnet-SDL	50	32	2	21:47:50	0.0022±0.0014	0.9165±0.0455
3D-DDnet-GDL	50	32	2	21:42:09	0.0012±0.0009	0.9233±0.0779
3D-DDnet-TL	15	32	2	1:56:13	0.0014±0.0008	0.9201±0.0596
3D-DDnet-TL-GDL	50	16	1	10:31:03	0.0010±0.0006	**0.9339±0.0572**
3D-DDnet-TL-GDL	50	16	2	5:24:07	0.0011±0.0006	0.9258 ±0.0603
3D-DDnet-TL-GDL	50	16	4	**3:08:17**	**0.0010±0.0006**	0.9303±0.0667
3D-DDnet-TL-GDL	50	32	2	21:43:28	0.0021±0.0080	0.9246±0.0634
3D-DDnet-TL-GDL	50	32	4	11:19:36	0.0011±0.0009	0.9257±0.0713
3D-DDnet-TL-GDL	25	32	2	10:52:32	0.0016±0.0017	0.9247±0.0866
3D-DDnet-TL-GDL	25	32	4	5:40:45	0.0015±0.0013	0.9076±0.0807

Our results show that there is very little difference between the utilization of 32 slices per volume compared to the 16 slices per volume in both training (Fig. 6) and testing. From Table 2, it is apparent that while holding all hyperparameters the same but the number of slices per volume there is only a ±1% difference which is within the margin of error between the two models. This leads to the conclusion that the utilization of more slices in the volume can be considered wasted computation time as the training time is almost double. For maximum parallel efficiency while maintaining accuracy, selecting 16 slices is critical.



(a) Baseline overlaid with transfer learning

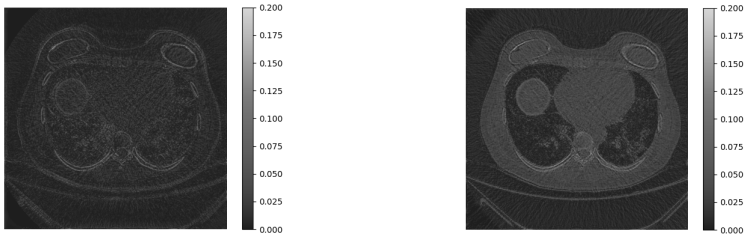
(b) Baseline without transfer learning

(c) Transfer learning training graph

Fig. 7: Graph (a) shows the order of magnitude difference in the loss function between utilizing transfer learning and not. Graph (b) shows the loss function without transfer learning, and graph (c) shows the loss function with transfer learning. Both graphs (b) and (c) show that transfer learning does not affect the ability of the network to optimize the loss function.

5.2 Qualitative Results

The difference maps in Fig. 8 also give an insight into what the network focuses on changing. At the same time, the 2D network is more focused on minor changes that pertain to the streaking. The 3D network focuses on the lung tissue, with the difference map showing the most changes in that area.

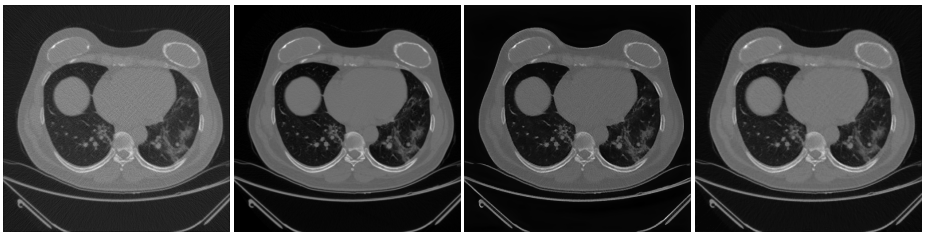


(a) 2D-DDnet

(b) 3D-DDnet

Fig. 8: Difference maps of the same randomly selected sample CT slice

The direct comparison between the 2D and 3D reconstructed images shows how the two architectures differ in what they aim to enhance and denoise. Looking at the 2D reconstructed image in Fig. 9(c), much of the streaking is removed from the source LDCT slice found in Fig. 9(a). While this may seem positive initially, many features found in the target CT slice shown in Fig. 9(b) are lost in this process. Looking at the 3D reconstructed image in Fig. 9(d), we can see that the streaking is overall reduced from the LDCT scan. With this reduction comes the retention of features, which can be seen on the top portion of the scan.



(a) Source Low Dose CT | (b) Target Full Dose CT | (c) 2D-DDnet Reconstructed | (d) 3D-DDnet Reconstructed

Fig. 9: Comparison between the LDCT, target slice, 2D reconstructed slice, and 3D reconstructed slice. The sample CT slice was randomly selected from the dataset. 3D-DDnet can maintain features that are lost in the 2D reconstruction. Retaining these features comes at the cost of some streaking and artifacts still present in the 3D reconstruction.

6 Conclusion

This work presents a method for adopting 3D architectures in biomedical image denoising, demonstrating effective training of larger models on single-node multi-GPU systems using distributed data parallelism. We adapt existing 2D data loaders and datasets for 3D modeling and employ transfer learning for efficient reuse of datasets and pretrained models.

While focused on LDCT scan denoising, our 3D-DDnet architecture is versatile, applicable to fields like image segmentation and classification, and shows potential in MRI imaging. This study indicates the need for scaling to multi-node setups for larger datasets due to the correlation between batch size and GPU utilization.

The shift to 3D introduces hyperparameter sensitivity, addressed through manual tuning in this study. Future research could explore automated optimization methods. Overall, 3D-DDnet effectively outperforms its 2D predecessor, highlighting the advantages of 3D models in biomedical imaging.

Acknowledgments. This research was supported in part by NSF IIS-2027607 and NSF CCF-2031215 as well as Dr. Cynthia McCollough, the Mayo Clinic, and the Amer-

ican Association of Physicists in Medicine and grants EB017095 and EB017185 from the NIBIB, for providing the Mayo Clinic dataset. We acknowledge Garvit Goel for his initial work on the 2D model and its evolution to the 3D model as well as Advanced Research Computing at Virginia Tech for providing the computational resources and technical support that contributed to the results in this paper.

References

1. J. Hsieh, "Computed tomography: principles, design, artifacts, and recent advances," vol. 1, 2015.
2. R. Fazel, H. M. Krumholz, Y. Wang, J. S. Ross, J. Chen, H. H. Ting, N. D. Shah, K. Nasir, A. J. Einstein, and B. K. Nallamothu, "Exposure to low-dose ionizing radiation from medical imaging procedures," *New England Journal of Medicine*, vol. 361, pp. 849–857, 2009.
3. Y. Zhou, Y. Zheng, Y. Wen, X. Dai, X. Wang, Q. Gong, C. Huang, F. Lv, and J. Wu, "Radiation dose levels in chest computed tomography scans of coronavirus disease 2019 pneumonia," *Medicine*, vol. 100, p. e26692, 2021.
4. H. Chen, Y. Zhang, M. K. Kalra, F. Lin, Y. Chen, P. Liao, J. Zhou, and G. Wang, "Low-dose ct with a residual encoder-decoder convolutional neural network," *IEEE Transactions on Medical Imaging*, vol. 36, no. 12, p. 2524–2535, 2017.
5. Z. Zhang, X. Liang, X. Dong, Y. Xie, and G. Cao, "A sparse-view ct reconstruction method based on combination of densenet and deconvolution," *IEEE Transactions on Medical Imaging*, vol. 37, no. 6, pp. 1407–1417, 2018.
6. A. Avesta, S. Hossain, M. Lin, M. Aboian, H. M. Krumholz, and S. Aneja, "Comparing 3d, 2.5d, and 2d approaches to brain image segmentation," 2022.
7. S. P. Singh, L. Wang, S. Gupta, H. Goli, P. Padmanabhan, and B. Gulyás, "3d deep learning on medical images: A review," 2020.
8. L. Crespi, D. Loiacono, and P. Sartori, "Are 3d better than 2d convolutional neural networks for medical imaging semantic segmentation?" in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8.
9. Z. Zhou, N. R. Huber, A. Inoue, C. H. McCollough, and L. Yu, "Multislice input for 2D and 3D residual convolutional neural network noise reduction in CT," *Journal of Medical Imaging*, vol. 10, no. 1, p. 014003, 2023.
10. G. Goel, A. Gondhalekar, J. Qi, Z. Zhang, G. Cao, and W. Feng, "Compute-covid19+: Accelerating covid-19 diagnosis and monitoring via high-performance deep learning on ct images," in *Proceedings of the 50th International Conference on Parallel Processing*. Association for Computing Machinery, 2021.
11. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.
12. Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
13. Z. Wang, E. Simoncelli, and A. Bovik, "Multiscale structural similarity for image quality assessment," in *37th Asilomar Conf. on Signals, Systems, and Computers, 2003*, vol. 2, 2003, pp. 1398–1402 Vol.2.