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

Armen C. Kasparian

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Wu-chun Feng, Chair
Thinh Doan
Sharath Raghvendra

February 26, 2023
Blacksburg, Virginia

Keywords: Deep Learning, Distributed Data Parallelism, Transfer Learning, Computed Tomography, Image Enhancement, Image Denoising

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

Armen C. Kasparian

(ABSTRACT)

This paper introduces 3D-DDnet, a cutting-edge 3D deep learning (DL) framework designed to improve the image quality of low-dose computed tomography (LDCT) scans. Although LDCT scans are advantageous for reducing radiation exposure, they inherently suffer from reduced image quality. Our novel 3D DL architecture addresses this issue by effectively enhancing LDCT images to achieve parity with the quality of standard-dose CT scans. By exploiting the inter-slice correlation present in volumetric CT data, 3D-DDnet surpasses existing denoising benchmarks. It incorporates distributed data parallel (DDP) and transfer learning techniques to significantly accelerate the training process. The DDP approach is particularly tailored for operation across multiple Nvidia A100 GPUs, facilitating the processing of large-scale volumetric data sets that were previously unmanageable due to size constraints. Comparative analyses demonstrate that 3D-DDnet reduces the mean square error (MSE) by 10% over its 2D counterpart, 2D-DDnet. Moreover, by applying transfer learning from pre-trained 2D models, 3D-DDnet effectively 'jump starts' the learning process, cutting training times by half without compromising on model accuracy.

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

Armen C. Kasparian

(GENERAL AUDIENCE ABSTRACT)

This research focuses on improving the quality of low-dose CT scans using advanced technology. CT scans are medical imaging techniques used to see inside the body. Low-dose CT (LDCT) scans use less radiation than standard CT scans, making them safer, but the downside is that the images are not as clear. To solve this problem, we developed a new deep learning method to make these low-dose images clearer and as good as regular CT scans. Our approach, called 3D-DDnet, is unique because it looks at the scans in 3D, considering how slices of the scan are related, which helps remove the noise and improve the image quality. Additionally, we used a technique called distributed data parallel (DDP) with advanced GPUs (graphics processing units, which are powerful computer components) to speed up the training of our system. This means our method can learn to improve images faster and work with larger data sets than before. Our results are promising: 3D-DDnet improved the image quality of low-dose CT scans significantly better than previous methods. Also, by using what we call "transfer learning" (starting with a pre-made model and adapting it), we cut the training time in half without losing accuracy. This development is essential for making low-dose CT scans more effective and safer for patients.

Dedication

This work is dedicated to my grandfather Dr. Raffy Hovanessian, for his everlasting push for education.

Acknowledgments

I would like to first thank my advisor, Dr. Wu Feng, for always providing the environment for growth and knowledge. His excellence as an advisor has taught me the ins and outs of research in academia and national labs. I must also thank Dr. Guohua Cao for being a guiding ear from the beginning of my work on 3D image denoising. His guidance and understanding of the biomedical community provided me with the resources to build in this field. I also acknowledge the work done by Garvit Goel, a previous Synergy Lab member and immediate predecessor of this research endeavor. His efforts to bring the 3D model to fruition were pivotal in my research. On the computational front, I must thank *Advanced Research Computing* at Virginia Tech for providing computational resources and technical support that have contributed to the results reported in this thesis. I would also like to acknowledge individuals in my personal life who have been pivotal in completing my degree and thesis. I would like to thank my Synergy Lab colleagues for providing a workplace of collaboration and conversation that only strengthened my research. I am also grateful for the support of my friends Alex, Matt, Sengal, Clancy, and Kevin, and many others for making my graduate school experience better than I could have imagined. My family has also been enormously influential in supporting me in all my endeavors. Ara, Alec, Alexei, and Alessandra, for always bringing a smile into my life. My Shomama for being the apple of my eye and always reassuring me that my path is just beginning. Most importantly, I must thank my Mother for being there and always providing a listening ear to guide me to my successes.

Contents

List of Figures

List of Tables

List of Abbreviations

2D-DDnet Two-Dimensional DenseNet and Deconvolution Network

3D-DDnet Three-Dimensional DenseNet and Deconvolution Network

AI Artificial Intelligence

CONVnet Convolutional Network

COVID-19 Coronavirus Disease of 2019

CPU Central Processing Unit

CT Computed Tomography

DDP Distributed Data Parallelism

DL Data Loader

DL Deep Learning

DP Distributed Parallelism

GPU Graphics Processing Unit

LDCT Low-Dose Computed Tomography

ML Machine Learning

MLP Multilayer Perceptron

MRI Magnetic Resonance Imaging

MS-SSIM Multi Scale Structural Similarity Index

MSE Mean squared Error

NN Neural Network

SSIM Structural Similarity Index

TF Transfer Learning

Chapter 1

Introduction

CT scans have proven to be one of the most valuable resources doctors utilize for diagnosing and managing medical conditions. Radiologists rely on CT imaging techniques to get sensitive images of internal organs for initial diagnosis and management of patients. The quantitative data and detail they provide allow for a noninvasive technique to monitor the progression of diseases and the effectiveness of treatments over time [7]. Unfortunately, this noninvasive technique comes with its downsides. Radiation exposure is a factor doctors must think about when ordering the procedure. Fazel et al. found that over 75.4% of patients' total effective radiation dose in the United States can be attributed to CT and nuclear medicine scans [5]. This study used 50 millisieverts (mSv) as the "high" exposure range. A standard chest CT scan typically effectively delivers 4.55 mSv of radiation. In diagnostic imaging, efforts should be made to minimize radiation doses and follow the principle of ALARA to protect patients from unnecessary exposure [1]. With the need to continually minimize radiation exposure, low-dose CT (LDCT) scans have become an avenue for doctors to order scans while being more conscious of patient exposure. The LDCT scans reduced radiation exposure of, on average, 0.5 mSv comes at the cost of image fidelity [16]. Compared to their standard dose counterparts, these scans introduce a large amount of noise and artifacts, inhibiting doctors' ability to utilize the scan.

Our method for transitioning into three-dimensional architectures leverages a parallel architecture for training the more parameter-rich space. The approach can be broken into

two main categories: efficiency improvement via distributed data parallel training strategies and accuracy improvement via data loader modifications. This implementation of parallel training allows our approach to utilize the entire resolution CT scan by selectively choosing the volumes from the CT scans. The number of slices selected for a volume is an important hyperparameter that can now be explored due to recent improvements in GPU VRAM. This capacity increase allows larger sample volumes to be offloaded from the CPU, allowing for feasible training times. This paper will explore the effects of utilizing this parallel approach in both efficiency and accuracy and will provide experimental results to back these claims.

1.1 Motivation

This work is an extension of a completed NSF project titled "ComputeCOVID19+: Accelerating COVID-19 Diagnosis and Monitoring via High-Performance Deep Learning on CT Images" by Garvit Goel et al. [6]. The project began as a way to utilize CT scans for diagnosis of brain diseases pre-pandemic. The project's focus shifted to apply the prior techniques from the brain to the lungs to quickly and accurately diagnose COVID-19 with CT scans of the lung during the 2019 pandemic. The project comprises a full stack pipeline containing an image-denoising, deep-learning model and a classification AI. In the setup, the image-denoising model is piped into the classification AI, which was trained to determine if COVID-19 is present in the patient.

The need for the ComputeCOVID19+ project is evident when looking at research from Johns Hopkins University on the efficacy of the RT-PCR test as a method of diagnosing COVID-19 patients [6]. In 2020, they showed that the false-negative rate (the rate at which the test incorrectly claims the patient is not infected) was 67% on the 4th day. This fact is further problematic as the prolonged turnaround time from administering the test to the result

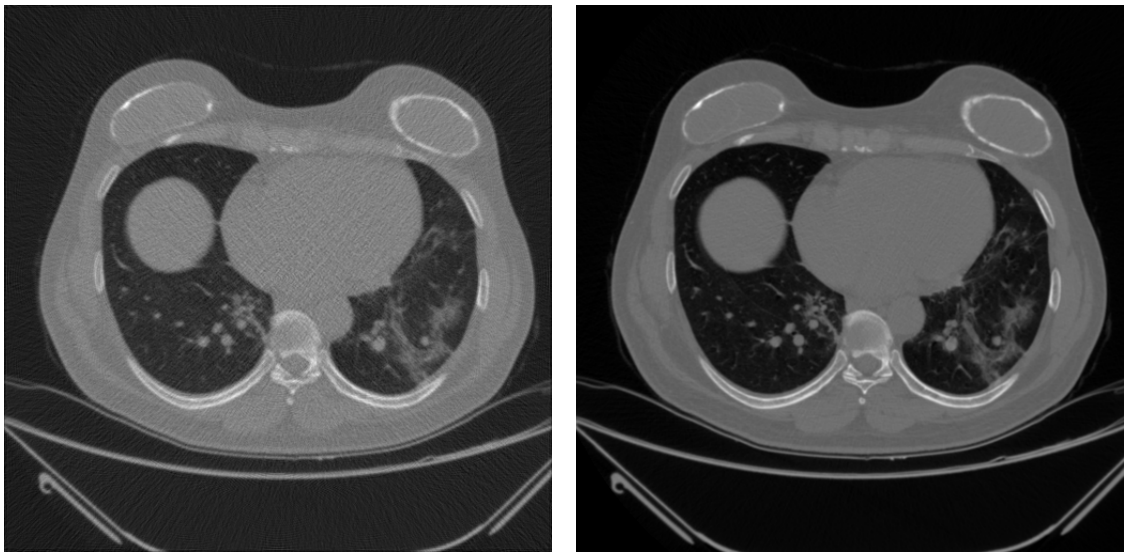
increases this time frame. This extended turnaround time, along with the time taken for the samples to be collected, packaged, and delivered to the lab, almost turns the RT-PCR test into a coin flip as to whether it will present a false negative.

This means that the solution to fast, reliable testing requires a new approach separate from the RT-PCR test that can improve upon the presently available while using technology currently accessible. The ComputeCOVID19+ project determined that utilizing CT scans of the lung could provide a trained classification AI to diagnose patients accurately. Unfortunately, there are downsides to CT scans that need to be addressed to make this a feasible replacement and solution; the primary one is radiation.

Radiation exposure is a factor that doctors must think about when ordering the procedure. Fazel et al. found that over 75.4% of patients' total effective radiation dose in the United States can be attributed to CT and nuclear medicine scans [5]. This study used 50 millisieverts (mSv) as the "high" exposure range. A standard chest CT scan typically effectively delivers 4.55 mSv of radiation. In diagnostic imaging, efforts should be made to minimize radiation doses and follow the principle of ALARA to protect patients from unnecessary exposure [1].

With the need to continually reduce radiation exposure, low-dose CT (LDCT) scans have become an avenue for doctors to order scans while being more conscious of patient exposure. The LDCT scans reduced radiation exposure of, on average, 0.5 mSv comes at the cost of image fidelity [16]. Compared to their standard dose counterparts, these scans introduce a large amount of noise and artifacts, inhibiting doctors' ability to utilize the scan. This can be seen in Fig. 1.1, where the LDCT scan shows significant degradation compared to the full-dose scan.

This hurdle is overcome by using an image denoising algorithm called 2D-DDnet. This



(a) Low-dose CT Scan Slice (b) Full-dose CT Scan Slice

Figure 1.1: Comparison between low-dose and standard-dose CT scan showing the artifacts and streaking caused by the lower radiation amount.

algorithm is designed to input a single slice of an LDCT scan and denoise it to look more closely like its full dose counterpart. Overall, the 2D-DDnet approach was trained to achieve a 78% reduction in overall noise and artifacts.

1.2 Contributions

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, allowing for feasible training times.

1.3 Related Work

Existing 3D deep learning architectures focusing on CT images have explored segmentation, classification, and detection applications against their respective 2D counterparts. Avesta et al. provide comparisons between two-dimensional and three-dimensional U-Net based deep learning architectures for image segmentation in brain MRIs [2]. Their work focused on each model's performance and accuracy characteristics, showing that the 3D models required approximately 20 times more computational memory. The 3D models could also converge to a more accurate and better performing state at the expense of the significant computational overhead introduced by the extra dimension. The approach presented here in this paper aims to solve the computational increase introduced by utilizing modern training techniques. This will provide the ability to keep the more accurate U-Net based model without the limitations of computation training time.

Recent research into 3D deep learning in the medical field has shown a recent undertaking into the higher-dimension networks. Singh et al. reviewed the biomedical field's adoption of 3D deep learning networks and found that in areas of classification, segmentation, and detection, common challenges plague acceptance [11]. These challenges involve the increased difficulty introduced in training and hyperparameter tuning and the inability to utilize smaller datasets.

Data has been and will be an integral part of deep learning. Dataset selection and curating a diverse and unbiased training set is a significant undertaking. Crespi et al. researched the transition from 2D convolution neural networks (CNN) to 3D CNNs. They found that a large problem with expanding to the larger-dimension space is the datasets available [4]. Our paper utilizes a data-loading strategy that can create volumes from the existing slice-based CT scan datasets. These data-loader changes reduce the overhead of curating a custom

dataset for the problem by introducing the ability to reuse previous datasets designed for 2D CT scan denoising.

Multislice inputs for 3D CNN noise reduction have previously been explored on the accuracy front. Zhou et al. presented a survey outlining the effects of modifying the number of slices in the sample volume. They show that with the increase in accuracy of the 3D network comes the downside of training time per epoch [17]. In their results, the 2D network variant takes 25 minutes to train per epoch, while the 3D equivalent takes 270 minutes. Additionally, due to hardware limitations, their data loader is limited to sampling 64x64 patches of the 512x512 source slices. These results leave an opportunity for acceleration via parallelism to decrease training time and allow the use of full-resolution slices.

Both statistical and deep learning-based methods have been introduced to combat the noise and artifacts found in LDCT scans. MBIR is a statistical model-based iterative reconstruction method that has been demonstrated to enhance image quality and reduce noise in clinical CT systems at the cost of increased computational requirements [9]. Along with the statistical approaches are encoder-decoder deep learning architectures like RED-CNN [3]. Networks like RED-CNN and 2D-DDnet are designed to learn features from target standard dosage CT scans and utilize that information to denoise LDCT scans. These deep neural network architectures have previously focused on denoising these three-dimensional CT scans via two-dimensional slice-based approaches, as three-dimensional denoising has been computationally intensive.

1.4 Thesis Organization

The rest of the thesis is organized as follows. The central architecture along with the enhancements made to the network will be presented in Chapter 2. This chapter outlines the

3D-DDnet architecture along with the optimizations made to datasets in Section 2.2, data loaders in Section 2.3, training strategy in Section 2.4, and finally the transfer learning strategy in Section 2.5. Chapter 3 presents the evaluation metrics utilized for training and testing, primarily focusing on the multi-part loss function. It also contains Section 3.2, which focuses on the evaluation architecture designed to compare dimensional mismatched architectures equally. Chapter 4 presents the training and testing results over the tailored dataset between 2D-DDnet and 3D-DDnet. This chapter analyzes how each of the optimizations affects the training and testing of the three-dimensional architecture and presents both quantitative and qualitative results in Sections 4.1 and 4.2, respectively. Chapter 5 concludes the thesis, along with a presentation of avenues for future work.

Chapter 2

CT Scan Denoising

Increasing the dimensions of a deep-learning (DL) architecture from two to three dramatically increases the parameters of the network — in our case, from 1,021,619 to 4,712,723 — and allows for more features to be extracted but at the expense of more computation [12]. This added computing cost is mainly seen in the training phase of the network, where the increase in parameters requires more time to converge to an optimal solution.

Along with the rise in network parameters comes an increase in data. With the network now ingesting 3D sample volumes comes an increase in computing costs. To alleviate the transition costs, we present an overhaul to the training strategy and enhancements to the data loader for our 3D-DDnet architecture.

2.1 Architecture

Our 3D-DDnet architecture extends our previous "2D DenseNet and Deconvolutional neural network" (i.e., DDnet, also referred to as 2D-DDnet in this thesis in order to explicitly distinguish it from 3D-DDnet) [15], as shown in Fig. 2.1. This extended architecture generalizes more information from the source data by using the correlations found between slices and delivers better accuracy.

Our new 3D-DDnet overhauls 2D-DDnet to support three-dimensional (3D) data. In the top

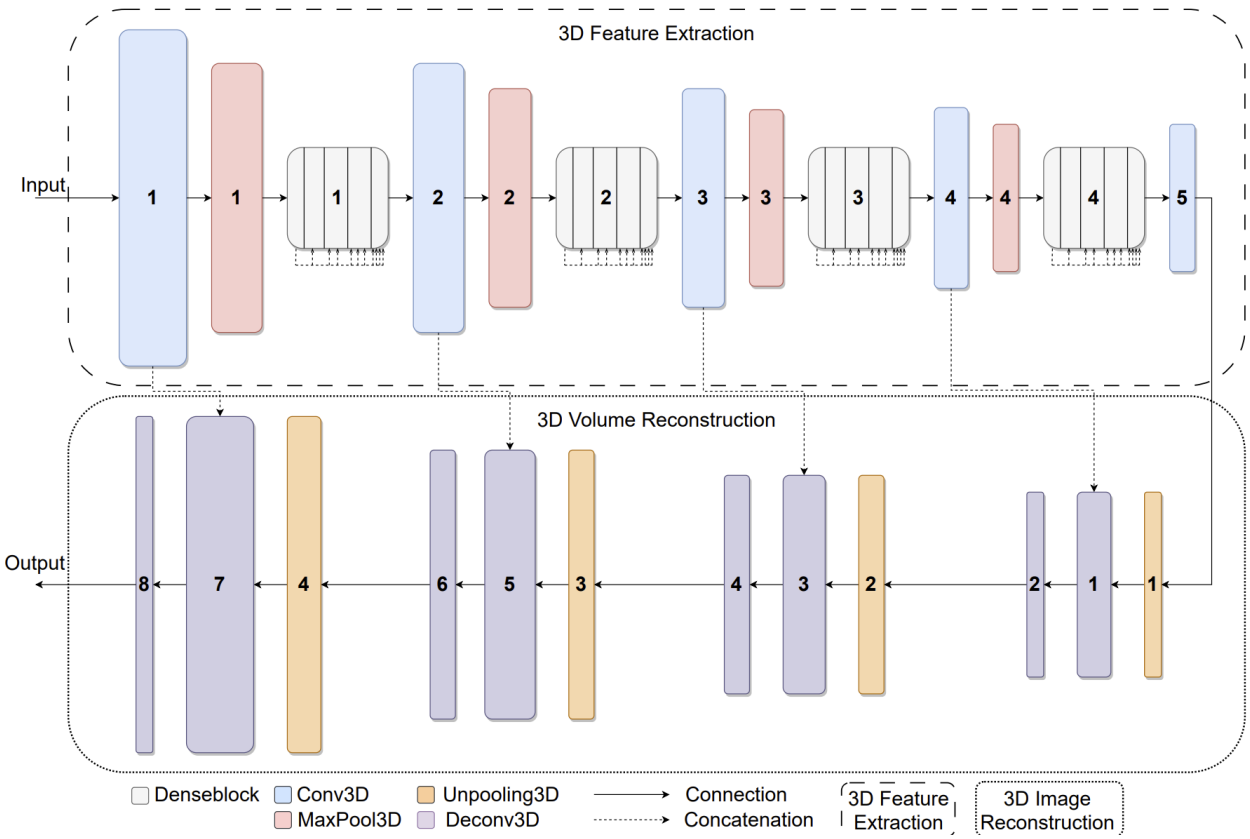


Figure 2.1: Architecture of 3D-DDnet. Layer numbers correspond to Table 2.1.

half of 3D-DDnet, the convolution layers now consist of 3D dense blocks and 3D convolutions. The 3D dense blocks contain internal 3D convolution layers, which are followed by 3D max pooling layers. The 3D max pooling layers 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 2.1 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, allowing for research into the effects of different volume sizes.

A detailed explanation of the original two-dimensional layers and the corresponding higher dimension three-dimensional layers are provided to understand the architecture fully and

Table 2.1: 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

show why the move to a three-dimensional architecture can extract more information than its two-dimensional predecessor.

2.1.1 Convolution Layers

In the feature-extraction phase of our architecture, the convolution layers are pivotal to learning. These layers apply various filters to extract progressively complex features. As data progresses through these layers, the spatial resolution decreases while the feature depth increases, allowing the network to capture a broader spatial context crucial for understand-

ing the image's global structure. Non-linear activation functions, like ReLU, follow each convolution operation, enabling the model to learn complex patterns by introducing nonlinearity. This use of convolution layers in the feature extraction portion is fundamental for providing detailed, multi-level feature information necessary for precise image denoising.

2D Convolution Layers

In deep learning (DL), a 2D convolutional block primarily performs feature extraction from 2D data, such as images. It consists of a 2D convolutional layer, where a filter (or kernel) slides over the 2D input data (image) and performs element-wise multiplication followed by a summation, producing a feature map.

For the 2D convolution formula in Eq. (2.1), f represents the input image and g denotes the kernel or filter applied to the image. The indices i and j specify the location in the output feature map. The double summation iterates over the kernel dimensions, where m and n run through the range of the kernel size; M and N represent half the kernel size. The term *f(m,n)f(m,n)* accesses the pixel value at position *(m,n)(m,n)* in the input image, and *g(i−m,j−n)g(i−m,j−n)* gives the corresponding kernel value, effectively applying a weighted sum of the pixel values under the kernel to produce each element of the output feature map.

$$(f * g)(i, j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f(m, n) \cdot g(i - m, j - n) \tag{2.1}$$

3D Convolution Layers

In deep learning (DL), a 3D convolutional block is designed to process 3D data, such as volumetric images or videos, by extracting spatial-temporal features. It consists of a 3D convolutional layer, where a 3D filter moves across the input data in three dimensions —

height, width, and depth. It performs element-wise multiplication and summarizes the results to produce a 3D feature map. This process effectively captures patterns across the three dimensions. This is crucial for medical imaging applications methods like CT scans.

For the 3D convolution formula in Eq. (2.2), f represents the 3D input, and g is the 3D kernel or filter being applied. The indices i, j, and k refer to the coordinates in the output feature map. The triple summation loops over the kernel's dimensions with m, n, and o indexing through the extent of the kernel; M, N, and O represent half the size of the kernel in each dimension. *f(m,n,o)f(m,n,o)* accesses the value in the input at position *(m,n,o)(m,n,o)*, while *g(i−m,j−n,k−o)g(i−m,j−n,k−o)* provides the kernel's corresponding value, which is used to calculate a weighted sum across the kernel's volume to produce each element of the output feature map. This process allows the 3D convolution to capture the data's complex spatial and temporal patterns.

$$(f * g)(i, j, k) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} \sum_{o=-O}^{O} f(m, n, o) \cdot g(i - m, j - n, k - o) \tag{2.2}$$

2.1.2 Max Pooling Layers

Max pooling serves two primary purposes: dimensionality reduction and feature enhancement with noise suppression. Max pooling reduces the dimensionality of the input feature maps by utilizing a sliding window. The largest value in the window is then selected. For example, in a 2D 3×3 window, the largest value out of the nine values found in the window would be selected, and the window would then move to the next set of values and repeat. This method of downsampling reduces the size of the input feature map by the stride size. By selecting the maximum value within each window, the max pooling operation helps highlight the important features of the input while suppressing less essential details like noise.

2D Max Pooling

For the MaxPooling2D equation, shown in Eq. (2.3), P_{ij} represents the value of the output feature map at the position (i, j). The input feature map is denoted by I, and the max pooling operation is applied. The size of the pooling window is represented by K and set to three in our use case, which defines the height and width of the square window used for pooling. The stride of the pooling operation is given by s, which determines the step size for moving the pooling window across the input feature map. In our use case, we also use a value of three to reduce the dimension after each pooling layer to 1/3. The terms a and b are indices that iterate over the window dimensions, with the double maximization operation $\max_{a=0}^{K-1} \max_{b=0}^{K-1}$ calculating the maximum value within the $K \times K$ window at each position $(s \cdot i, s \cdot j)$ on the input feature map.

$$\mathrm{P}_{ij} = \max_{a=0}^{K-1} \max_{b=0}^{K-1} I_{s \cdot i+a, s \cdot j+b} \tag{2.3}$$

3D Max Pooling

In the context of the MaxPooling3D equation, as shown in Eq. (2.4), the term P_{ijk} signifies the value at the location (i, j, k) on the output feature map. The symbol I represents the input feature map, to which the max pooling operation is applied. The dimension of the pooling window is given by K, which in this instance is three, corresponding to a $3 \times 3 \times 3$ window. The stride, denoted as s, specifies the displacement of the pooling window across the input feature map for each iteration. For our specific implementation, we chose a stride of three, effectively reducing the spatial dimensions of the features to a third of their initial size. The variables a, b, and c are employed to traverse the dimensions of the pooling window, and the process of identifying the maximum value within the $K \times K \times K$ window at every

coordinate $(s \cdot i, s \cdot j, s \cdot k)$ on the input feature map is described by the triple maximization operation $\max_{a=0}^{K-1} \max_{b=0}^{K-1} \max_{c=0}^{K-1}$.

$$\mathrm{P}_{ijk} = \max_{a=0}^{K-1} \max_{b=0}^{K-1} \max_{c=0}^{K-1} I_{s \cdot i+a, s \cdot j+b, s \cdot k+c} \tag{2.4}$$

2.1.3 Deconvolution Layers

Deconvolution is the key to the reconstruction portion of the 3D-DDnet architecture. Deconvolution layers are crucial for upsampling and reconstructing the feature map to its original resolution. The deconvolution layers in the expansive or reconstruction phase gradually restore the spatial resolution from the rich feature output space of the first portion of the architecture. These layers transform the condensed, high-level feature information obtained in the earlier layers into a higher resolution space. Deconvolution layers achieve this by reversing the effect of convolution and pooling operations: they map the lower-resolution, deep feature representations to higher-resolution, shallower feature maps. This process is accompanied by concatenation with correspondingly cropped feature maps from the contracting path called skip connections, which helps to recover spatial information lost during downsampling. The result is a detailed, high-resolution feature map that combines high-level contextual information from the deeper layers and fine-grained spatial information from the earlier layers, enabling precise segmentation or reconstruction of the original image.

2D Deconvolution

For the Deconv2D operation, shown in Eq. (2.5), $O(i, j)$ denotes the output value at the position (i, j). The deconvolution kernel is represented by $K(m, n)$ with dimensions M and N. The input feature map is denoted by I, and the operation iteratively applies the kernel

to this input. The stride of the operation is indicated by s, determining the step size for each kernel application. The padding applied to the input feature map is represented by p, adjusting the effective spatial dimensions of the input. This deconvolution operation is commonly used in neural networks for upsampling 2D data.

$$\mathrm{O}(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(m,n) \cdot I(s \cdot i + m - p, s \cdot j + n - p) \tag{2.5}$$

3D Deconvolution

In the Deconv3D operation, shown in Eq. (2.6), $O(i,j,k)$ denotes the output value at the position (i,j,k). The kernel of the deconvolution, $K(m,n,o)$, has dimensions M, N, and O. This function works in the same manner as explained in §2.1.3 explanation but adds an extra dimension to account for the z axis. This operation is used for upsampling in 3D space.

$$\mathrm{O}(i,j,k) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \sum_{o=0}^{O-1} K(m,n,o) \cdot I(s \cdot i + m - p, s \cdot j + n - p, s \cdot k + o - p) \tag{2.6}$$

2.1.4 Unpooling Layers

In the reconstruction phase of both 2-DDnet and 3D-DDnet, unpooling layers are required to upsample the feature maps to rebuild the image to its original resolution. After the feature extraction phase, which reduces the spatial dimensions and increases the feature depth, the unpooling layers in the reconstruction phase work to reverse this process. They incrementally increase the spatial dimensions of the feature maps while reducing their depth. Unpooling involves expanding the reduced feature maps by duplicating or rearranging their elements to form a larger map. This step is often combined with concatenation operations, where

feature maps from the feature extraction phase are merged with the upsampled maps in the reconstruction phase. This merging is done with skip connections to reintroduce spatial information lost during downsampling—this aids in restoring fine details and context to the output. The result is a high-resolution, detailed feature map that synthesizes the contextual information from the network's deeper layers with the spatial information from its initial layers, enabling accurate and detailed restoration of the important features of the original image.

2D Unpooling

In the Unpooling2D operation, shown in Eq. (2.7), $O(i, j)$ denotes the output value at the position (i, j). The input feature map is indicated by I. The unpooling process involves upsampling the input feature map, where s is the stride of the unpooling operation, determining the upsampling factor in each dimension. The floor function $\lfloor \cdot \rfloor$ is used to map the coordinates of the output feature map back to the corresponding coordinates of the input feature map. This unpooling operation increases the size of the input feature map by duplicating its values in the x and y dimensions.

$$\mathrm{O}(i, j) = I(\lfloor \frac{i}{s} \rfloor, \lfloor \frac{j}{s} \rfloor) \tag{2.7}$$

3D Unpooling

In the Unpooling3D operation, shown in Eq. (2.8), $O(i, j, k)$ denotes the output value at the position (i, j, k). The input feature map is denoted by I. The operation involves upsampling the input feature map, where s represents the stride of the unpooling operation, dictating the upsampling factor in each dimension. In our application, the value of three is selected

to match the downsampling factor used in the feature extraction portion of the architecture. The floor function $\lfloor \cdot \rfloor$ maps the coordinates of the output feature map back to the corresponding coordinates of the input feature map. This process increases the size of the input feature map by duplicating its values in the x, y and z dimensions.

$$\mathrm{O}(i,j,k) = I(\lfloor \frac{i}{s} \rfloor, \lfloor \frac{j}{s} \rfloor, \lfloor \frac{k}{s} \rfloor) \tag{2.8}$$

2.2 Datasets

Understanding the two-dimensional dataset can provide context regarding the scale required to switch to three dimensions. In our application, we designed our 3D dataset to be built off the original 2D data. This allows for the reuse of the existing data and makes adding an extra dimension significantly less resource-intensive with respect to data collection.

We created our dataset from three sources, as shown in Table 2.2. The original dataset utilized for training the prior 2D-DDnet model consists of the same dataset but is formatted and loaded into the model differently. The variation in the dimensions of the models calls for adaptations to the dataset along with modifications to the data loader. The adaptations to the dataset are explained below by outlining the differences between the two-dimensional and three-dimensional settings.

Table 2.2: Dataset Sources

Database	Samples
Mayo Clinic	100 Healthy CT Scans at Full and Quarter Dosage with variable slices per scan
Lung Image Database Consortium (LIDC)	722 CT Scans with 256/512 slices per scan
Medical Imaging Databank of the Valencia Region (BIMCV)	397 CT Scans with variable slices per scan

2.2.1 Two-Dimensional Datasets

In the two-dimensional setting, a dataset that is comprised of slices of biomedical images can be looked at individually. In this setting, each slice has a source and target version that the network can use to learn the transformation required to convert the source image to look like its target. Each slice is then denoised and run through the network in batches.

2.2.2 Three-Dimensional Datasets

In the three-dimensional setting, a dataset is comprised of collections of slices known as sample volumes. These volumes contain slices that are ordered and correlated with each other to create the three-dimensional structure of the sample. In this setting, each sample volume has a source and target version that the network can use to learn the transformation required to convert the source volume into the target volume. In this case, the network is not trained to denoise individual slices. Instead, it is trained to denoise the volume in its entirety.

2.3 Data Loader

Two-dimensional data loaders are designed to feed slices in batches to the model. In this setting, a batch size of one would pass a single image to the model for each epoch for training. Increasing the batch size in a two-dimensional data loader increases the number of image slices passed through each epoch.

Three-dimensional data loaders are designed to load in a volume instead of a single slice. In this space, a volume is considered a collection of two-dimensional slices that are stacked together and sent into the model as a single sample. A batch utilizing a three-dimensional

data loader consists of multiple volumes that are each passed through the network individually. Computationally, this is very expensive as we are now multiplying the amount of data found in the two-dimensional setting by the number of slices found in the volume.

The computational overhead of the extra dimension creates a computational limitation that is not as prevalent in the two-dimensional setting. For example, in a slice of size 512x512, 2^{18} pixels need to be enhanced, whereas a volume of 16 images of a sample of 512x512x16 brings the total number of pixels that need to be passed through the network to 2^{22}. In hardware, the single slice of the two-dimensional slice requires about 1.5 GB of memory to be loaded. In comparison, the three-dimensional example above would require nearly 20 GB of memory. This makes selecting the correct number of slices for the volume into a hyperparameter of the three-dimensional model, which has an upper limit depending on the hardware of the system you are running the model on.

Along with the limitation of the number of slices that can be used comes the question of which slices to choose from the sample to create the three-dimensional volume. The number of slices possible for a sample is limited to the hardware resources available. Selecting which slices from the complete CT scan then becomes a two-pronged question: (1) Which slices do we choose? (2) How many slides are sufficient for the accuracy gains desired? In our research, we present and test two different versions of the data loader shown in Fig. 2.2. Each of which has benefits and drawbacks. These different data loaders can be switched and selected case by case, depending on what the problem set calls for. The following discussion explains the differences and benefits of the strided data loader and the grouped data loader.

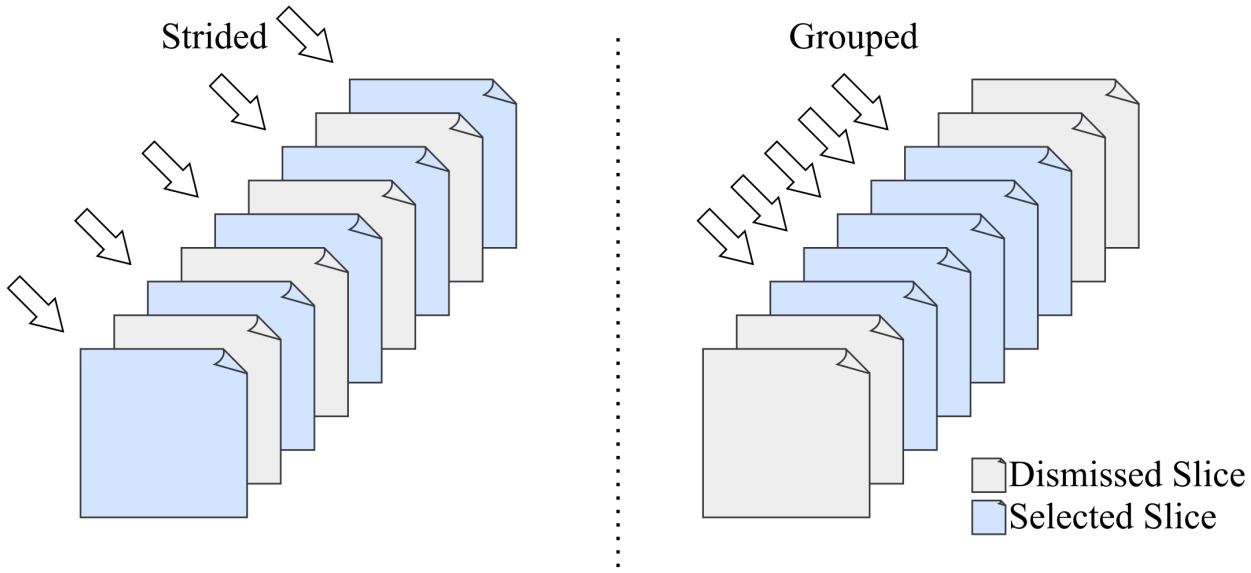


Figure 2.2: Strided vs. grouped data loader. The figure shows a selection of a five-slice volume selected from a seven-slice sample. A stride length of size two is used in this example.

2.3.1 Strided Data Loader

A strided approach maintains a constant number of evenly spaced image slices per input sample volume. In this situation, a CT scan consists of an unknown number of slices that pertain to a particular patient. Sample slices for the volume in the input scan are homogeneously selected at equal distances from each other. For example, in an LDCT scan containing 512 image slices, a desired volume of size 64 consists of every 8th image slice from the scan. This data loader approach allows for a dataset containing CT scans that contain differing numbers of slices. This process works as long as the number of slices determined for the hyperparameter is less than the minimum number of slices in the smallest LDCT scan.

The benefits of using this data loader are that an equidistant selection of slices spanning the entire CT scan represents the total sample. This data loader selection is suitable in a situation where vital information can be found throughout the complete sample. There are situations where this data loader is detrimental to the model's learning. This approach places

equal value on all the slices inside the sample, which is not always optimal. An example of this is in our biomedical imaging case, where the beginning and end of the LDCT samples may contain limited information about the target. In this case, the strided data loader assigns each slice an equal value by selecting equally through the sample and including the miscellaneous slices that may not aid the overall training goal.

2.3.2 Center-Grouped Data Loader

Introducing the third dimension into the model allows the architecture to learn the correlation between the slices of the data. 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 benefits of using a grouped data loader are that the collection of slices selected for the volume contains the highest correlation between them. For example, if using the strided data loader, the stride distance is eight, and the correlation between slice zero and slice one in the volume will be low as it is separated by eight slices in the sample. Using the grouped data loader, we guarantee maximum correlation between the slices that comprise the sample volume to try and maximize the gain found when switching from two-dimensional architectures.

Two different versions of this system were implemented, with the original grouped data loader always selecting from the center of the dataset sample. The second iteration of this data loader randomly selects a grouped volume from the slices at an undetermined location in the dataset sample.

2.3.3 Future Data Loaders

Countless variations of this system can be implemented. There are many combinations of what data to pick and from where. Future considerations for large 3D models can look at the data loader for more performance gains as computing resources continue to increase. Another iteration of this data loader could randomly select a grouped volume from the slices at an undetermined location in the dataset sample until all the potential groups are chosen. This could be a possible design for more accuracy. This approach would be very computationally intensive and increase training time immensely but would ensure that the model is trained on the full CT scans. This would also provide the full use of the dataset.

2.4 PyTorch DDP Training Strategy

Modern GPU hardware is essential for transitioning from 2D to 3D network architectures. Current datacenter GPUs offer up to 80GB of HMB2e VRAM, a significant upgrade from the previous 32GB of HBM2. This increase, coupled with 50% performance gains, supports loading larger datasets onto GPUs, and, in turn, facilitates the shift from 2D slices to 3D volumetric data in image enhancement.

Using the hardware resources of modern GPU hardware is critical to the transition from 2D to 3D network architectures. Modern datacenter GPUs currently "max out" at 80GB of HMB2e VRAM per card, allowing more data to be loaded onto the GPU for training. Past generations of GPUs comparable hardware have at most 32 GB of slower HBM2. With more than double the on-chip memory capacity and 50% performance gains over past generations, we can now load larger datasets onto the GPUs, allowing for the move to 3D volumetric data over 2D slices for image enhancement. The need for larger capacity accelerators can be seen

by looking at the sizes of a single sample of data. A single 512×512 slice is approximately 1.5GB. Compared with the 16-slice and 32-slice volumes, which are 20GB and 40GB respectively, this shows just how much data needs to be loaded for training a volume. Table 2.3 shows the respective sizes of each slice/volume of data.

Table 2.3: Approximate size in GB of sample image/volume

Sample Dimensions	Approximate Size (GB)
512×512	1.5
512×512×16	20
512×512×32	40

Past generations of datacenter GPUs could not even load a single sample volume onto the GPU (i.e., 20GB and 40GB, as noted in Table 2.3). 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 solving this problem uses PyTorch's 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. 2.3.

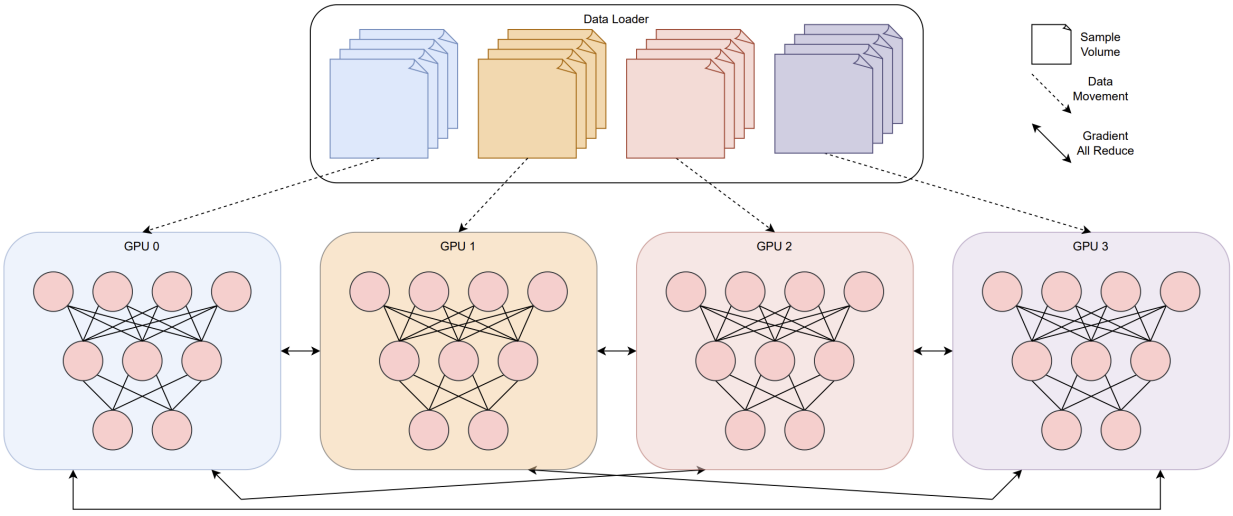


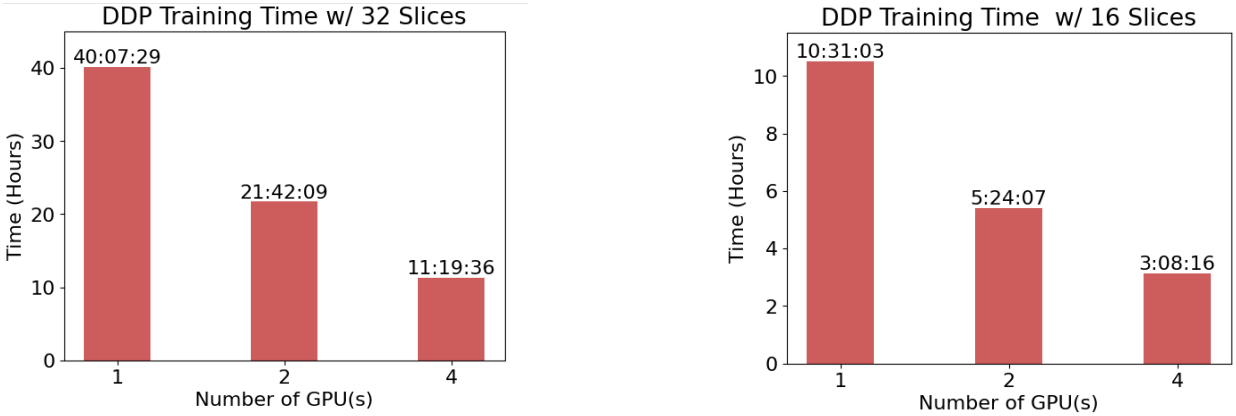
Figure 2.3: Overview of the distributed data parallel architecture. The data loader individually sends unique batches of data to each GPU while each of the models loaded onto the GPUs are synchronized during the gradient, all-reduce step found in backpropagation. The models receive different data from the data loader but average the gradients per epoch, leaving the models synchronized.

2.4.1 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.

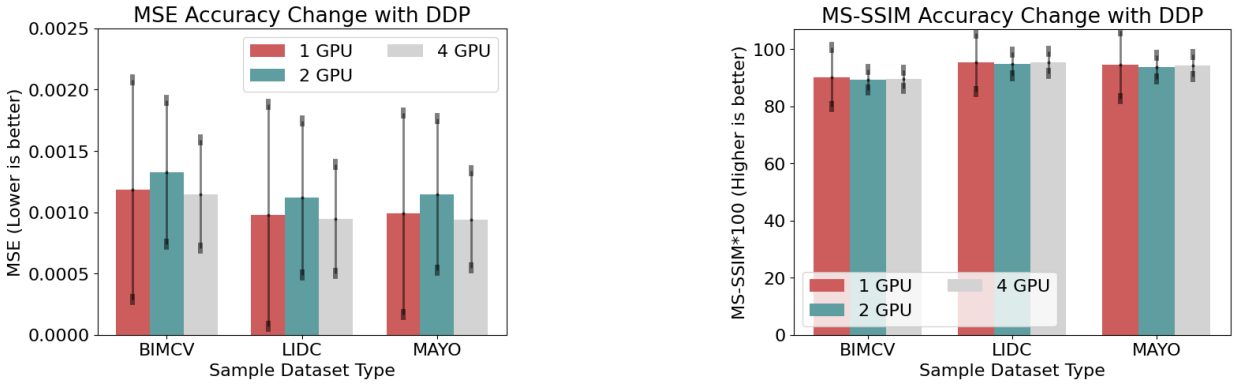
In our application with volumetric CT image enhancement, this now removes the VRAM limitation of the GPU and increases the previous maximum batch size from one to the number of GPUs available for training. It is important to note that this is only possible since we can load, at minimum, a single 3D volumetric 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. 2.4 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. 2.4. Another unexpected benefit from increasing the number of GPUs during training is the decrease in the variance of the accuracy. When looking at Fig. 2.5, we can see the variance bands continually decrease as more GPUs are utilized during training.



(a) (b)

Figure 2.4: Training time scaling graph for 3D-DDnet utilizing DDP: (a) 32-Slice Volume, (b) 16-Slice Volume



(a) (b)

Figure 2.5: Accuracy variance while utilizing 1/2/4 GPUs. The figure outlines the variance between the different datasets (a): MSE, (b) MS-SSIM

2.5 Transfer Learning

With many trained 2D architectures already in the biomedical field, we can leverage the computational costs associated with training these 2D models to enhance the 3D variants. Our transfer learning algorithm allows the reuse of the older network architectures to "boost" the new architectures. Leveraging pre-existing 2D networks can also 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. 2.6.

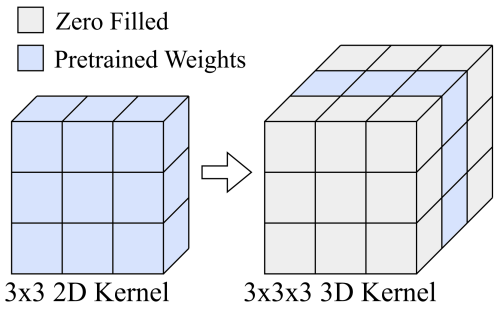


Figure 2.6: Transfer learning diagram showing insert of 2D kernel into 3D kernel

Unoccupied values in the 3D kernel are filled with zeros from the pre-trained 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.

2.5.1 2D-to-2D Transfer Learning Algorithm

In the default approach to transfer learning, the trained layers of the donor network are directly grafted onto the recipient network. The donor layers are spliced and placed in the desired location of the recipient network. This ensures that the learned weights and biases from the donor architecture are present at the beginning of the training of the recipient network. This approach only works if the dimensions of the architectures match, allowing for a direct transplant. If the dimensions are different between the networks, as in our case, we must adapt our transfer learning approach.

Algorithm 1 Transfer Learning Algorithm

1: **Input:** Network 1 with weights $W^{(1)}$ and biases $B^{(1)}$
2: **Input:** Network 2 with weights $W^{(2)}$ and biases $B^{(2)}$
3: **for** each selected layer l **do**
4: Transfer weights: $W_l^{(2)} \leftarrow W_l^{(1)}$
5: Transfer biases: $B_l^{(2)} \leftarrow B_l^{(1)}$
6: **end for**
7: **Output:** Updated Network 2 with transferred weights and biases

In the presented transfer learning algorithm, the notation $W^{(1)}$ and $B^{(1)}$ represent the weights and biases of the source neural network, referred to as Network 1. Similarly, $W^{(2)}$ and $B^{(2)}$ denote the weights and biases of the target neural network, Network 2. The algorithm involves transferring weights and biases from selected layers of Network 1 to the corresponding layers in Network 2. This process is iterative and conducted for each selected layer, denoted by l. Specifically, $W_l^{(2)} \leftarrow W_l^{(1)}$ signifies the assignment of weights from layer l of Network 1 to layer l of Network 2, and $B_l^{(2)} \leftarrow B_l^{(1)}$ represents a similar transfer for biases. The algorithm aims to leverage learned features from Network 1 to enhance or expedite the learning

process in Network 2.

2.5.2 2D-to-3D Transfer Learning Algorithm

Following the approach from [10] provided the key to applying transfer learning between models of different dimensions. Transfer learning from a smaller network architecture to a larger one is easily understood when looking at the network layer by layer. When transferring a lower-dimensional layer into a higher-dimensional one, the weights and biases of the lower-dimension kernel must be placed into the proper channel. For example, let us look at moving from a 2D image kernel to a 3D volume kernel. In the 2D kernel, let $K1$ and $K2$ represent the x and y dimensions of the kernel. In the 3D kernel, let $K1$ and $K2$ still represent the x and y dimensions and $K3$ represent the z dimension of the kernel. Since the larger 3D kernel is a stack of the smaller 2D kernel, we can insert the weights of the 2D kernel into a specific location in the 3D kernel.

Algorithm 2 Transfer Learning from 2D Network to 3D Network

1: **Input:** 2D Network with weights $W^{(2D)}$ and biases $B^{(2D)}$
2: **Input:** 3D Network with weights $W^{(3D)}$ and biases $B^{(3D)}$
3: **for** each selected layer l in 2D Network **do**
4: Initialize $W_l^{(3D)}$ and $B_l^{(3D)}$ with zeros
5: Determine the center channel c of $W_l^{(3D)}$ and $B_l^{(3D)}$
6: Transfer weights: $W_{l,c}^{(3D)} \leftarrow W_l^{(2D)}$
7: Transfer biases: $B_{l,c}^{(3D)} \leftarrow B_l^{(2D)}$
8: **end for**
9: **Output:** Updated 3D Network with weights and biases from 2D Network

Modifications of the original algorithm were tested, including an all-fill modification, where all layers of the 3D kernel are instantiated to be the same 2D kernel. Another modified variant was a random fill where instead of the zero-filled values, random Gaussian noise is used to fill the kernel. Testing showed that all modifications to the original algorithm performed within

the margin of error when comparing mean squared error (MSE) and multi-scale structural similarity (MS-SSIM). Furthermore, the decision was made to utilize the center-fill algorithm moving forward.

Chapter 3

Evaluation Metrics

In deep learning, the selection and formulation of appropriate evaluation metrics are necessary to assess model performance comprehensively. This chapter explores the intricacies of our chosen loss function, which stands as a summation of multiple metrics tailored to encapsulate the image-denoising objective of our model. By integrating these metrics, we aim to create a more nuanced and holistic loss function that aligns closely with our specific problem domain and goals. The following discussion explains the rationale behind the composite choice of our loss function, detailing how each component metric contributes to the overall model efficacy and guides the optimization process toward achieving superior generalization and predictive accuracy.

3.1 Metrics

In the world of deep learning, the success and efficiency of a model are not merely determined by its ability to make predictions or classifications. Instead, the accurate measure of a model's effectiveness lies in its performance, rigorously assessed using various evaluation metrics. These metrics are tools to quantitatively analyze and interpret the accuracy, precision, recall, and other aspects of a model's predictions. They not only facilitate an understanding of how well the model is performing in its current state but also provide critical insights into areas requiring improvement, thereby guiding the iterative process of model refinement. In

essence, evaluation metrics are indispensable for validating the reliability and robustness of deep learning models, ensuring they function optimally in real-world scenarios. Three metrics are used to train the network and quantify the effectiveness of our denoising model: MSE, SSIM, MS-SSIM. The following subsections will outline the use case of each of the metrics and provide a mathematical explanation of how each metric works. These metrics are then combined to create a loss function that equally weights each of the values.

3.1.1 MSE Metric

Mean squared error (MSE) is a direct pixel-to-pixel comparison of the differences between two images or volumes – see Eq. (3.1). The MSE equation iterates over N pixels, where X is the source image, and Y is the image after being denoised.

$$MSE(X,Y) = \frac{1}{N}\sum_{n=1}^{N}[X-Y]^2, \tag{3.1}$$

3.1.2 SSIM Metric

The structural similarity (SSIM) index, shown in Eq. (3.2), is a widely used metric to assess image quality and similarity. It measures the similarity between two images based on their structural information in the spatial domain [14]. 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. (3.2), 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 $k_2 = 0.03$).

$$\text{SSIM}(X, Y) = \frac{(2\mu_x\mu_y + C1)}{(\mu_x^2 + \mu_y^2 + C1)} \times \frac{(2\sigma_{xy} + C2)}{(\sigma_x^2 + \sigma_y^2 + C2)} = \text{l}(X, Y) \times \text{cs}(X, Y) \tag{3.2}$$

3.1.3 MS-SSIM Metric

The multi-scale structural similarity index (MS-SSIM), defined in Eq. (3.3), 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 weigh all the terms equally.

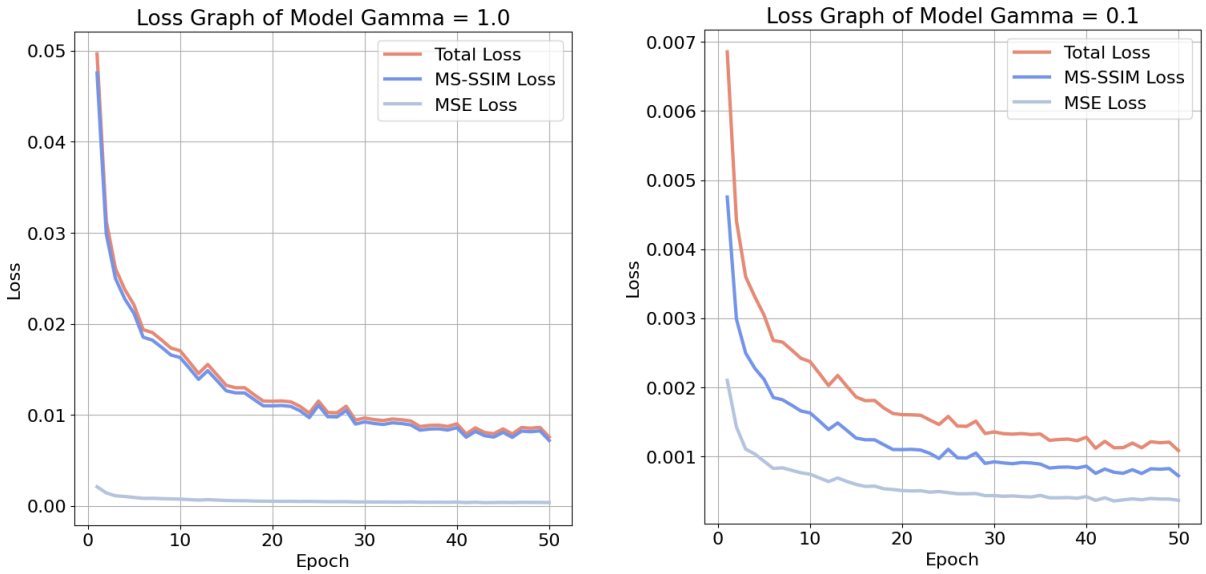
$$\text{MS-SSIM}(X, Y) = l_M^{\alpha}(X, Y) \cdot \prod_{j=1}^{M} cs_j^{\beta_j}(X, Y) \tag{3.3}$$

3.1.4 Loss Function

The loss function of the neural network dictates the metrics the backpropagation is trying to enhance. The loss function of the neural network is two prong. The loss function combines the higher quality MS-SSIM and 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.

$$\text{Loss}(X, Y) = \text{MSE}(X, Y) + \gamma(1 - \text{MS-SSIM}(X, Y)) \tag{3.4}$$

In our loss function, a γ of 0.1 is selected to properly balance the magnitude of these two terms. This value was chosen through manual exploration of the loss function. Fig. 3.1(a) shows the original loss function with both the MSE and MS-SSIM equally weighted. In this figure, the MS-SSIM function output overwhelms the MSE function output, rendering the MSE's weight in the total loss almost non-existent. In this regard, the loss function would primarily focus on optimizing the MS-SSIM function without optimizing the MSE metric. In Fig. 3.1(b), the gamma value set to 0.1 shows how the two metrics can be of the same magnitude. This allows for the loss function to be more evenly balanced and allows for the optimization function to weight the importance of each metric equally.



(a) Gamma = 1.0 (b) Gamma = 0.1

Figure 3.1: Graph showing the selected value of gamma. Selection evens the weight of each metric in the loss function

3.2 Comparison of Mismatched Dimensions

Comparing the performance gain from a two-dimensional model to its three-dimensional counterpart is challenging as both models try to optimize a different overall metric. In the two-dimensional setting, the network is trained and optimized for enhancing the individual slices of the sample stack. In contrast, the three-dimensional network is designed to enhance the complete sample volume in one pass. The decision was made to compare the three-dimensional network to the two-dimensional metrics to overcome this hurdle. To achieve this, we created a pipeline for producing reconstructed CT scans from both the 2D and 3D networks after training. This would then take the 3D volume and split it back into its respective slices. These 3D sourced 2D reconstructed slices would then be fed through the same metrics as the 2D sourced slices to produce the comparison. A representation of this pipeline can be seen in Fig. 3.2.

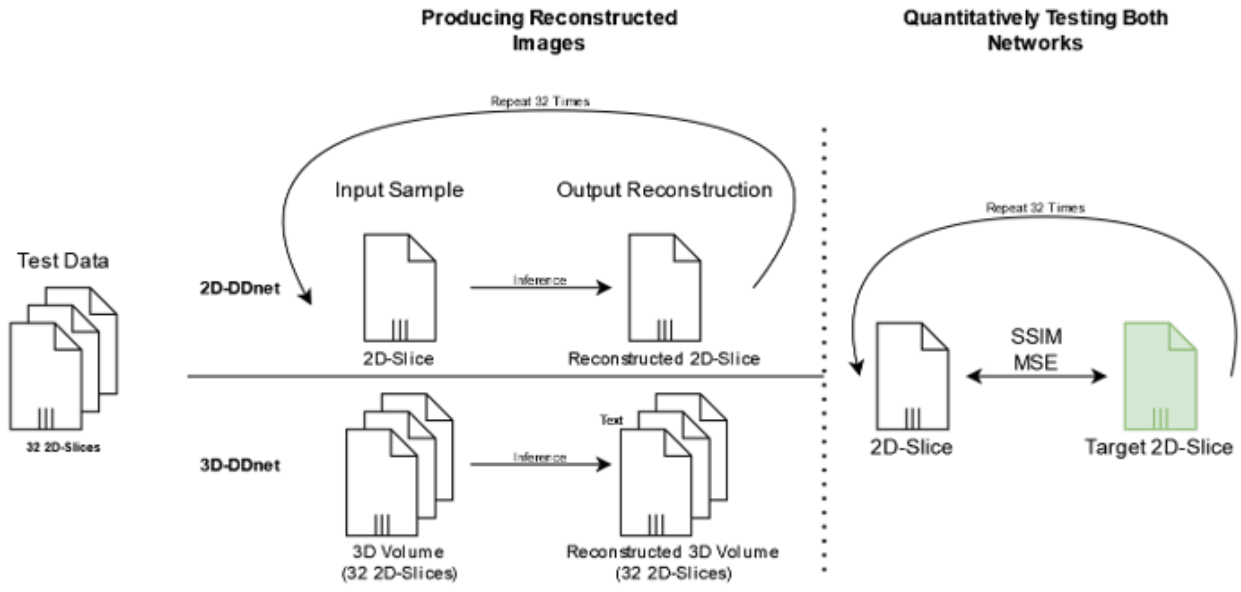


Figure 3.2: Testing pipeline for 2D vs 3D networks

Chapter 4

Results

The results are categorized into two main sections: quantitative and qualitative. Quantitatively, we will discuss timing profiles for different hyperparameters, specifically highlighting runs with the Adam optimizer and a learning rate of 0.001, and showcase training speedup via distributed data parallel methods with corresponding loss graphs to verify stability. Qualitatively, we will compare denoised CT images produced by the 2D-DDnet and 3D-DDnet networks. This comparison will be set against a high-quality target image, using the LDCT source image for reference. Additionally, difference maps will be presented to illustrate the denoising techniques of both 2D and 3D models on LDCT images.

We will present multiple different results pertaining to the architecture's performance regarding parallel scalability and accuracy. There will be two main portions of the results, the first being quantitative and the second being qualitative. For the quantitative results, we will first present the timing profiles of each run with different hyperparameters. The left untouched hyperparameters include using the Adam optimizer [8] with a learning rate of 0.001. Then, we will show the training speedup with distributed data parallel approaches with graphs of the loss confirming the stability of the implementation.

For the qualitative results, we will present the denoised CT images from the 2D-DDnet network compared to the denoised CT images from the 3D-DDnet network. This direct comparison will be presented alongside the target high quality image with the LDCT source image as a reference. We will then present the difference maps between the LDCT image and

the target images from both the 2D architecture and the 3D architecture. This will provide a qualitative view into how each dimensionally different model approaches the denoising effort of the LDCT images.

4.1 Quantitative Results

First and foremost, it is important to look at the baseline 2D-DDnet approach to see what reference we are trying to achieve as a baseline. The 2D model can be trained on a single GPU in approximately 8 hours and a batch size of one. Its average MSE over training is 0.003998, and the average MS-SSIM is 0.788877. These values represent the benchmarks we are trying to overcome with a three-dimensional approach. Looking at Table 4.1, we can see that all versions of the 3D-DDnet architecture tested were able to drop the MSE and increase the MS-SSIM. This shows that the 3D architecture is worth switching to if we can control the training time.

Table 4.1: 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. 4.1) and testing. From Table 4.1, it is apparent that while holding all hyperparameters 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.

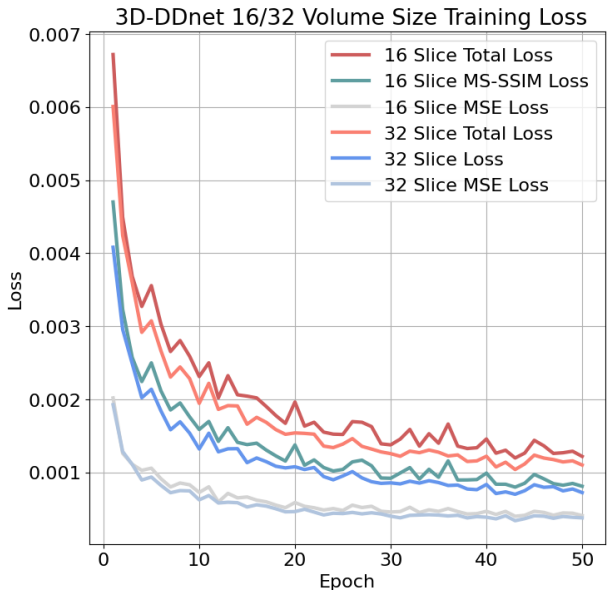
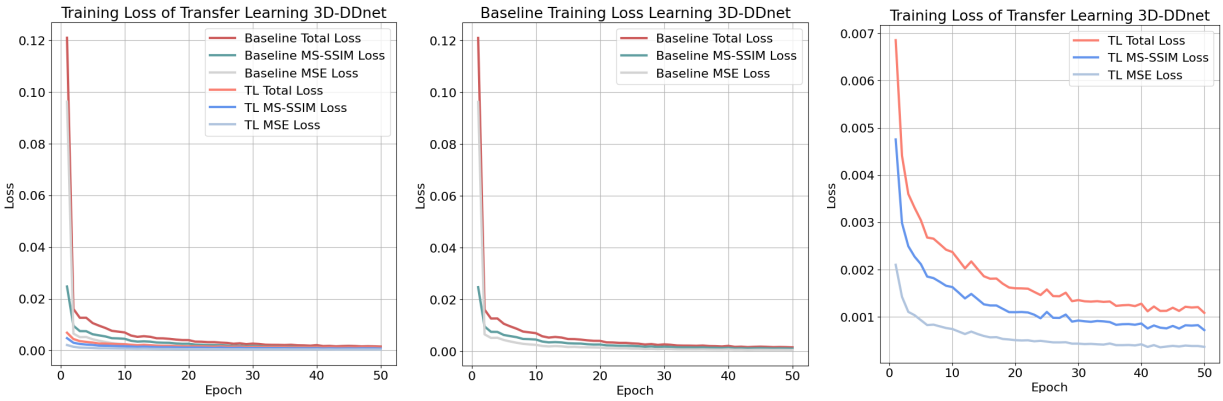


Figure 4.1: Training loss similarities between the 16- and 32-slice volume data loader

We can see the desired speedup from the distributed data parallel approach when looking at the training time for both the 16-slice runs and the 32-slice runs. When scaling from 1/2/4 GPUs, which corresponds to 1/2/4 batch sizes (each GPU receives a single batch of data), we see nearly a 2× speedup from one to two GPUs. From two to four GPUs, the scaling factor starts to drop off with the training time speedup only at 1.75×.

To ensure the parallel architecture is not affecting the training stability of the 3D network, we look at the loss curves for a single GPU approach vs. a multi-GPU approach. In Fig. 4.2, we show that the loss function continually decreases over the 50 training epochs without exhibiting any signs of divergence in the neural network. Along with this, the transfer learning (TL) algorithm gives a massive head start in learning compared to without it. Both



(a) Baseline overlaid with transfer learning

(b) Baseline without transfer learning

(c) Transfer learning training graph

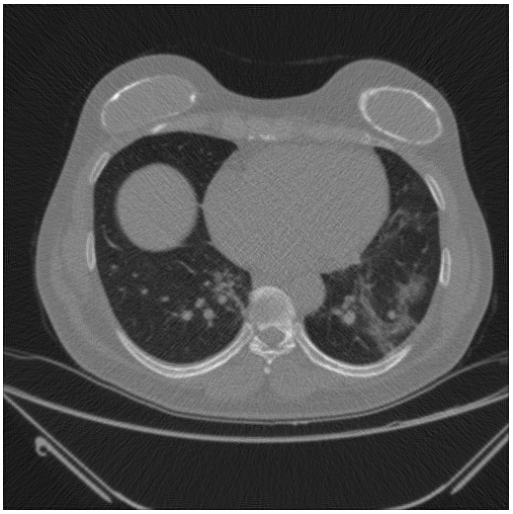
Figure 4.2: Graph (a) shows the order of magnitude difference in the loss function between utilizing transfer learning and not. Graph (b) shows the loss function without transfer learning, and graph (c) shows the loss function with transfer learning. Both graphs (b) and (c) show that transfer learning does not affect the ability of the network to optimize the loss function.

models can converge to the same loss value, but the speed at which the TL optimized model can achieve that value is much quicker, as the graph shows. Fig. 4.2 shows that the TL-optimized model can continually optimize the loss function even though the scale at which it is doing so is drastically decreased compared to the non-optimized model. When stopping the transfer learning optimized run early, we see that even at 15 epochs of training the 3D model outperforms its 2D predecessor. If the desired goal is to outperform the 2D predecessor, then training does not have to proceed longer. To achieve the maximum performance out of 3D-DDnet, Fig. 4.2(c) shows us that the loss function plateaus at 50 epochs. The plateau of the loss function shows that at that point in the training process, the network has converged to an optimum.

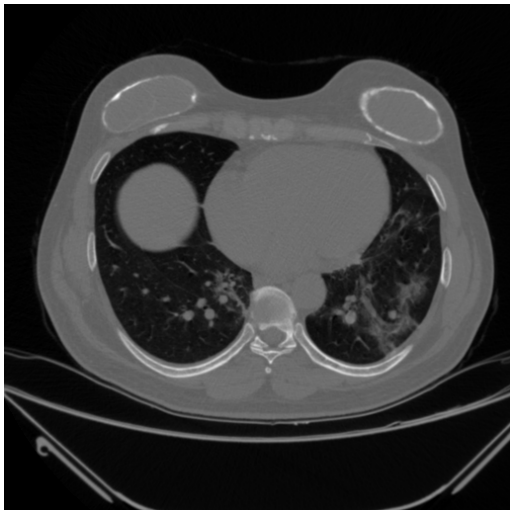
4.2 Qualitative Results

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. 4.3(c), much of the streaking is removed from the source LDCT slice found in Fig. 4.3(a). While this may seem positive initially, many features found in the target CT slice shown in Fig. 4.3(b) are lost in this process. Looking at the 3D reconstructed image in Fig. 4.3(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.

The difference maps in Fig. 4.4 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.

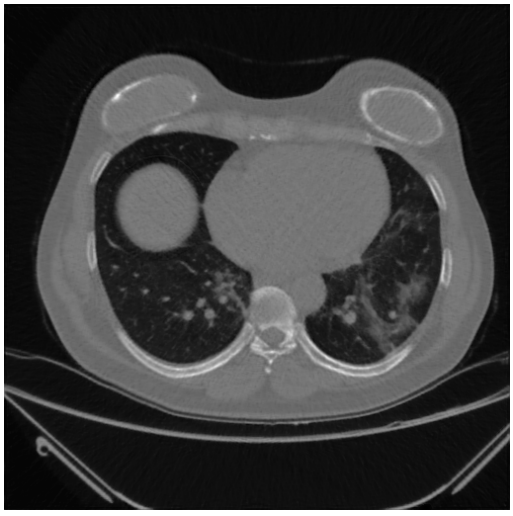


(a) Source low-dose CT



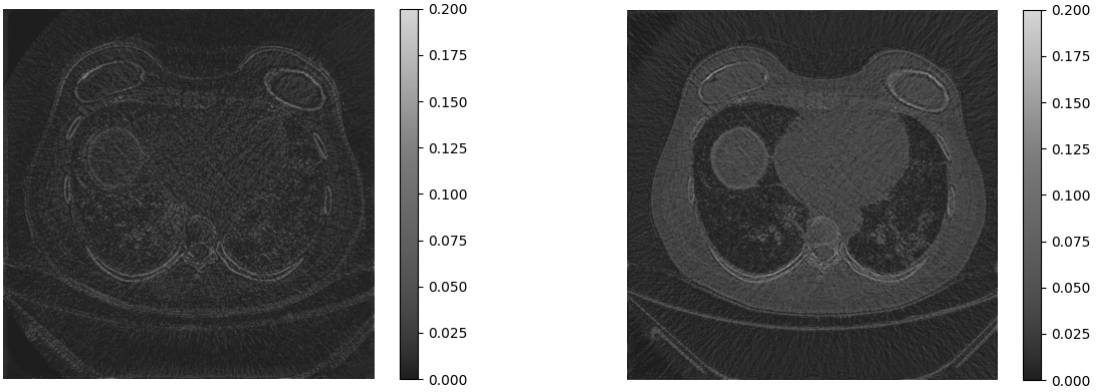
(b) Target full-dose CT

(c) 2D-DDnet reconstructed



(d) 3D-DDnet reconstructed

Figure 4.3: 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.



(a) 2D-DDnet (b) 3D-DDnet

Figure 4.4: Difference maps of the same randomly selected sample CT slice

Chapter 5

Discussion and Conclusion

5.1 Discussion and Future Work

This work shows a guideline for the biomedical image denoising field to embrace three-dimensional architectures fully. With the distributed data parallel approach, larger models have become feasible to train on single node multi-GPU setups. We have also shown that already existing two-dimensional data loaders and datasets can be converted to support three-dimensional architectures. This, along with the transfer learning algorithm shown in this paper, allows the reuse of datasets and pretrained models to achieve better accuracy models in shorter amounts of time.

While this work focused on denoising LDCT scans, its framework can be applied to different fields. Image segmentation and classification are two main areas where larger three-dimensional architectures can be applied, utilizing their two-dimensional counterparts as a springboard to boost efficiency. This work can also be applied to other three-dimensional scans traditionally augmented in two-dimensional settings. Another significant area of exploration could be in the MRI imaging applications, with denoising being a prime use case for the field.

In our application, the network design was not large enough or parameter dense enough to warrant using a multi-nodal environment. The 3D-DDnet architecture is tailored for its

specific use case. A more extensive, generalized network utilized for MRI and CT scans could require more computation hardware. In our use case, we also see a direct correlation between the batch size and GPU utilization due to the size of an individual batch of volumes being large enough that, at max, a batch size of two could fit into a single GPU. In a scenario where the dataset size is much larger, scaling up to a multi-nodal environment would be required to leverage the speedups shown here.

The drawbacks to moving to three dimensions do exist and are important to discuss. A large portion of this work was devoted to the hyperparameter tuning of the more extensive network. With the increase in architecture dimension comes the increase in sensitivity to hyperparameters. In our case, manual hyperparameter exploration considerably slowed the search for optimal solutions. A future consideration could be the introduction of auto-hyperparameter exploration to help discover more optimal values efficiently.

5.2 Conclusions

In this work, we presented the 3D-DDnet architecture as a novel approach to LDCT scan denoising. To realize this architecture, we implemented a distributed data parallel training strategy to parallelize the training step of the deep learning network. We also implemented an approach for utilizing transfer learning between structurally different models. We also presented the efforts to modify existing two-dimensional data loaders to support three-dimensional volumes. This allowed for the reuse of the preexisting two-dimensional dataset.

We showed that the 3D-DDnet architecture can outperform its 2D-DDnet predecessor in all aspects, including overall training time. We can show that introducing the spatial correlation between slices produces an accuracy increase of over 10%. In this work, we did not compare

against other CT denoising architectures as the main study of this work is not to outperform the competition. The primary research of this work is to present the steps required to take the leap into the extra dimension provided by the data and show the implications and effects that come with it. In this regard, the steps taken in this thesis can be applied to competitor neural networks working in two dimensions to bring a similar uplift in performance and accuracy. We can now show that this parallel approach benefits the biomedical imaging space and should be fully explored.

Bibliography

[1] URL https://www.nrc.gov/reading-rm/basic-ref/glossary/alara.html.

[2] Arman Avesta, Sajid Hossain, MingDe Lin, Mariam Aboian, Harlan M. Krumholz, and Sanjay Aneja. Comparing 3d, 2.5d, and 2d approaches to brain image segmentation, 2022.

[3] Hu Chen, Yi Zhang, Mannudeep K. Kalra, Feng Lin, Yang Chen, Peixi Liao, Jiliu Zhou, and Ge Wang. Low-dose ct with a residual encoder-decoder convolutional neural network. *IEEE Transactions on Medical Imaging*, 36(12):2524–2535, 2017. doi: 10.1109/tmi.2017.2715284.

[4] Leonardo Crespi, Daniele Loiacono, and Pierandrea Sartori. Are 3d better than 2d convolutional neural networks for medical imaging semantic segmentation? In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022. doi: 10.1109/IJCNN55064.2022.9892850.

[5] 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*, 361:849–857, 2009. doi: 10.1056/nejmoa0901249.

[6] Garvit Goel, Atharva Gondhalekar, Jingyuan Qi, Zhicheng Zhang, Guohua Cao, and Wu Feng. Computecovid19+: Accelerating covid-19 diagnosis and monitoring via high-performance deep learning on ct images. In *Proceedings of the 50th International Conference on Parallel Processing*, ICPP '21, New York, NY, USA, 2021. Association for

Computing Machinery. ISBN 9781450390682. doi: 10.1145/3472456.3473523. URL https://doi.org/10.1145/3472456.3473523.

[7] J. Hsieh. Computed tomography: principles, design, artifacts, and recent advances. 1, 2015. doi: 10.1117/3.2197756.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Ke Li, Jie Tang, and Guang-Hong Chen. Statistical model based iterative reconstruction (mbir) in clinical ct systems: Experimental assessment of noise performance. *Medical Physics*, 41(4):041906, 2014. doi: https://doi.org/10.1118/1.4867863. URL https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.4867863.

[10] Hongming Shan, Yi Zhang, Qingsong Yang, Uwe Kruger, Mannudeep K. Kalra, Ling Sun, Wenxiang Cong, and Ge Wang. 3-d convolutional encoder-decoder network for low-dose ct via transfer learning from a 2-d trained network. *IEEE Transactions on Medical Imaging*, 37(6):1522–1534, 2018. doi: 10.1109/TMI.2018.2832217.

[11] Satya P. Singh, Lipo Wang, Sukrit Gupta, Haveesh Goli, Parasuraman Padmanabhan, and Balázs Gulyás. 3d deep learning on medical images: A review, 2020.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. doi: 10.1109/CVPR.2015.7298594.

[13] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, volume 2, pages 1398–1402 Vol.2, 2003. doi: 10.1109/ACSSC.2003.1292216.

[14] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

[15] Zhicheng Zhang, Xiaokun Liang, Xu Dong, Yaoqin Xie, and Guohua Cao. A sparse-view ct reconstruction method based on combination of densenet and deconvolution. *IEEE Transactions on Medical Imaging*, 37(6):1407–1417, 2018. doi: 10.1109/TMI.2018.2823338.

[16] 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*, 100:e26692, 2021. doi: 10.1097/md.0000000000026692.

[17] Zhongxing Zhou, Nathan R. Huber, Akitoshi Inoue, Cynthia H. McCollough, and Lifeng Yu. Multislice input for 2D and 3D residual convolutional neural network noise reduction in CT. *Journal of Medical Imaging*, 10(1):014003, 2023. doi: 10.1117/1.JMI.10.1.014003. URL https://doi.org/10.1117/1.JMI.10.1.014003.

Appendices

Appendix A

First Appendix

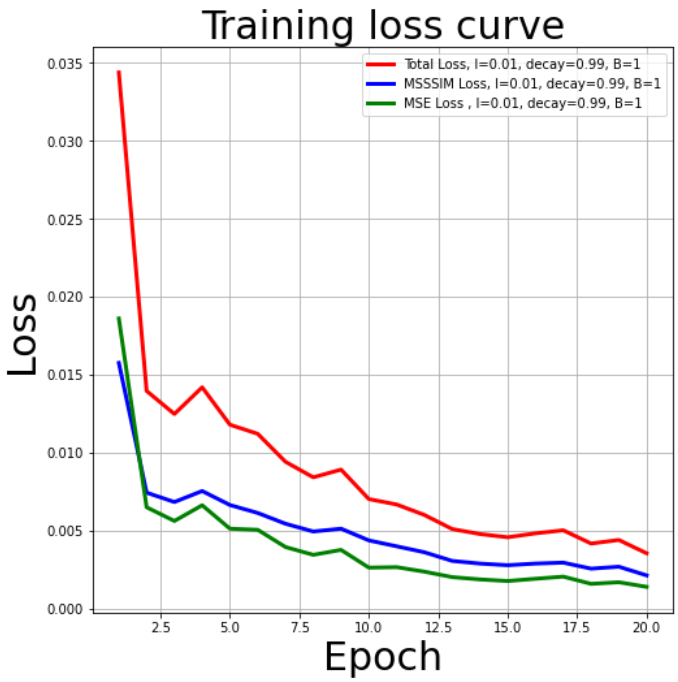


Figure A.1: Loss function training graph showing hyperparameter tuning with learning rate set to 0.01. A batch size of 1 and a decay rate of 0.99 are held as static variables.

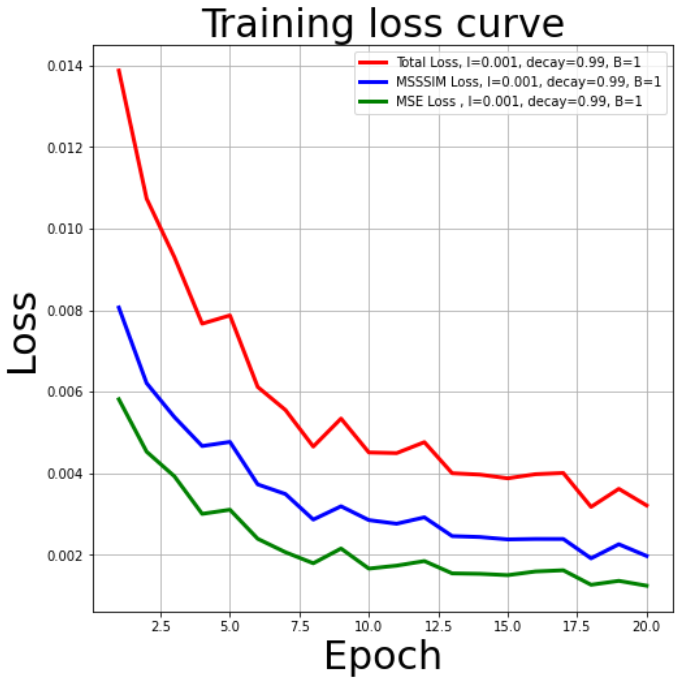


Figure A.2: Loss function training graph showing hyperparameter tuning with learning rate set to 0.001. A batch size of 1 and a decay rate of 0.99 are held as static variables.

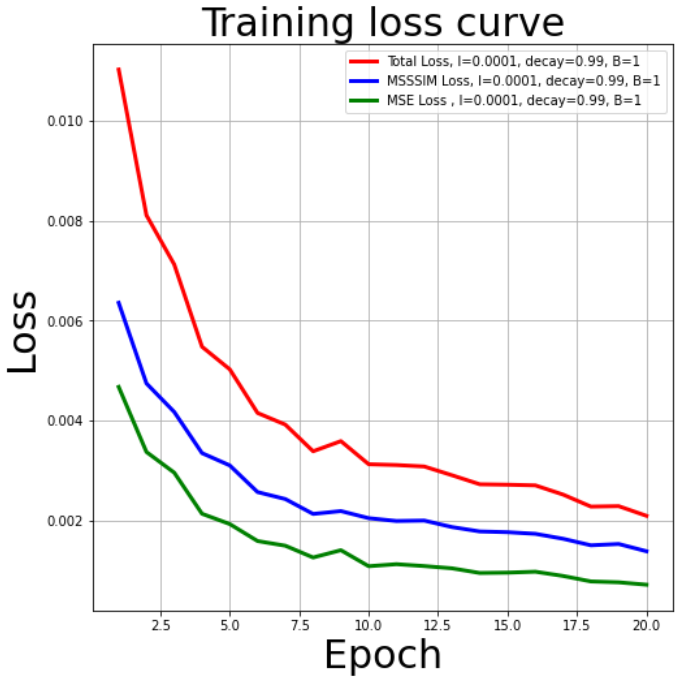
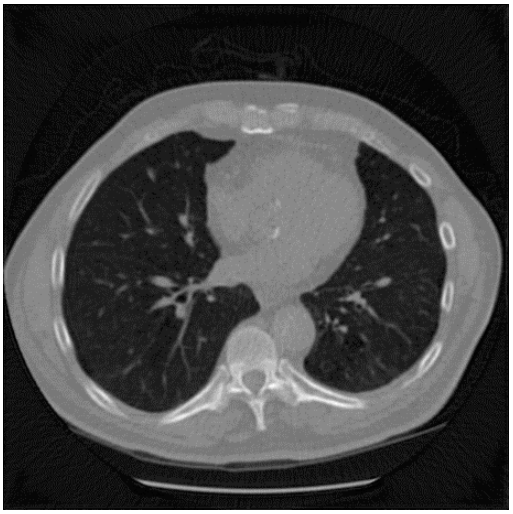
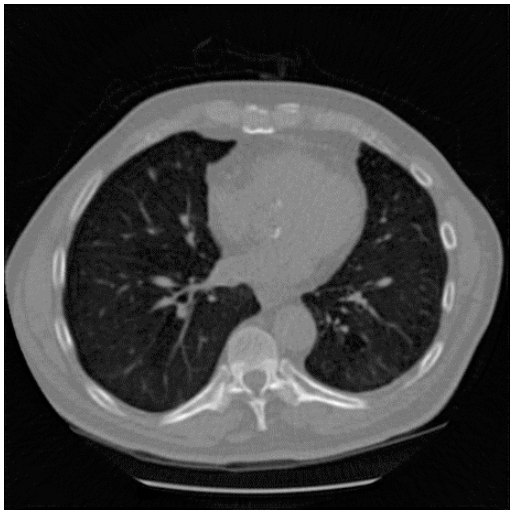


Figure A.3: Loss function training graph showing hyperparameter tuning with learning rate set to 0.0001. A batch size of 1 and a decay rate of 0.99 are held as static variables.



(a) Batch Size: 1



(b) Batch Size: 2

(c) Batch Size: 4

(d) Target Image

Figure A.4: Visualization of how batch size affects the reconstructed slices of the CT scans. A learning rate of 0.001 and a decay rate of 0.99 are held as static variables. A larger batch size helps the reconstruction of images by reducing the amount of streaking and artifacts.