Real-Time Computed Tomography-based Medical Diagnosis Using Deep Learning

Garvit Goel

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 Feng, Chair
Changwoo Min
Jia-Bin Huang
Thomas L. Martin
Guohua Cao

January 31, 2022
Blacksburg, Virginia

Keywords: AI, biomedical imaging, corona virus, COVID-19, deep learning, diagnosis, neural networks, GPU, FPGA

Real-Time Computed Tomography-based Medical Diagnosis Using Deep Learning

Garvit Goel

(ABSTRACT)

Computed tomography has been widely used in medical diagnosis to generate accurate images of the body's internal organs. However, cancer risk is associated with high X-ray dose CT scans, limiting its applicability in medical diagnosis and telemedicine applications. CT scans acquired at low X-ray dose generate low-quality images with noise and streaking artifacts. Therefore we develop a deep learning-based CT image enhancement algorithm for improving the quality of low-dose CT images. Our algorithm uses a convolution neural network called DenseNet and Deconvolution network (DDnet) to remove noise and artifacts from the input image. To evaluate its advantages in medical diagnosis, we use DDnet to enhance chest CT scans of COVID-19 patients. We show that image enhancement can improve the accuracy of COVID-19 diagnosis ($\approx 5\%$ improvement), using a framework consisting of AI-based tools. For training and inference of the image enhancement AI model, we use heterogeneous computing platform for accelerating the execution and decreasing the turnaround time. Specifically, we use multiple GPUs in distributed setup to exploit batch-level parallelism during training. We achieve approximately $7\times$ speedup with 8 GPUs running in parallel compared to training DDnet on a single GPU. For inference, we implement DDnet using OpenCL and evaluate its performance on multi-core CPU, many-core GPU, and FPGA. Our OpenCL implementation is at least $2\times$ faster than analogous PyTorch implementation on each platform and achieves comparable performance between CPU and FPGA, while FPGA operated at a much lower frequency.

Real-Time Computed Tomography-based Medical Diagnosis Using Deep Learning

Garvit Goel

(GENERAL AUDIENCE ABSTRACT)

Computed tomography has been widely used in the medical diagnosis of diseases, such as cancer/tumor, viral pneumonia, and more recently, COVID-19. However, the risk of cancer associated with X-ray dose in CT scans limits the use of computed tomography in biomedical imaging. Therefore we develop a deep learning-based image enhancement algorithm that can be used with low X-ray dose computed tomography scanners to generate high-quality CT images. The algorithm uses a state-of-the-art convolution neural network for increased performance and computational efficiency. Further, we use image enhancement algorithm to develop a framework of AI-based tools to improve the accuracy of COVID-19 diagnosis. We test and validate the framework with clinical COVID-19 data. Our framework applies to the diagnosis of COVID-19 and its variants, and other diseases that can be diagnosed via computed tomography. We utilize high-performance computing techniques to reduce the execution time of training and testing AI models in our framework. We also evaluate the efficacy of training and inference of the neural network on heterogeneous computing platforms, including multi-core CPU, many-core GPU, and field-programmable gate arrays (FPGA), in terms of speed and power consumption.

Dedication

To my parents and brother

Acknowledgments

First and foremost, I would like to express my gratitude towards my advisor, Professor Wu Feng, for giving me the opportunity to work with him and providing invaluable guidance during the last two years. I am indebted to his infallible support and encouragement. I again thank Professor Feng for our countless discussions about the progress of projects, presentation of research for knowledgeable and general audiences, and conversations about the career opportunities in computer engineering. I would also like to thank Dr. Gouhua Cao for giving me challenging projects and providing helpful comments and suggestions along the way. The computing resources provided to me by Advanced Computing Resources and Synergy Lab at Virginia Tech, and Intel via Devcloud, have enabled me to carry out my experiments needed for the research. I also want to acknowledge the contributions of individuals, agencies, and societies who were involved in data collection and compilation. I want to thank and acknowledge my collaborators Jingyuan Qi and Atharva Gondhalekar, with whom I collaborated on many projects. I am also grateful for the support of my colleagues at Synergy Lab for their technical insights and constructive feedback. Last but not least, I would like to acknowledge my friends and colleagues at Virginia Tech: Namrata Panji, Chinmay Nagori, Chinmay Sapre, Shashank Thakalapally, Nithya Gopinath, Anila Moorthy, Sourav Sinha, for their unfailing support and love, and for making my time at graduate school very enjoyable. Finally, I would like to thank my parents and brother for their unconditional love, support, and encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial intelligence

AUC-ROC Area under receiver operating characteristic curve

CNN Convolution neural network

COVID-19 Coronavirus disease of 2019

CPU Central processing unit

CT Computed tomography

DDnet DenseNet and deconvolution network

DL Deep learning

ED product Energy-delay product

FBP Filtered back projection

FLOPs Floating point operations

FPGA Field programmable gate array

FPR False positive rate

GFLOPs Giga-floating point operations per second

GOPs Giga-operations per second

GPU Graphic processing unit

ML Machine learning

MS-SSIM Multi scale-structural similarity index

MSE Mean-square error

RT-PCR Reverse transcriptase-polymerase chain reaction

TPR True positive rate

Chapter 1

Introduction

1.1 Motivation

Computed tomography (CT) has been a widely used procedure to generate three-dimensional (3-D) tomographic images of the body. However, the risk of cancer associated with the high X-ray dosage per CT scan has been a concern in medical imaging. The research study conducted by Brenner et al. [6] showed that abdominal CT scans can increase the attributable risk of death from cancer by up to 0.1%. So, researchers have used low-dose X-ray CT scans, coupled with image reconstruction algorithms like filtered back projection (FBP) [67], to deliver the 3-D tomographic images. Unfortunately, this results in degraded image quality with missing-view artifacts and blurring. Consequently, advanced techniques, such as iterative image reconstruction [49], sinogram completion [43], and deep learning-based image enhancement [88] have been proposed to reconstruct higher-quality images from these low-dose X-ray projections.

Mainly, deep learning-based image enhancement has gained a lot of traction in recent years. With the advances in high-performance computing (HPC), and availability of a large amount of open-source CT data, deep convolution neural networks can be trained with high efficiency and can generate superior results as compared to other advanced image reconstruction algorithms mentioned above [88].

Computed tomography can also be used for diagnosis of COVID-19. Patients with COVID-19 possess chest CT scans that exhibit a wide spectrum of distinguishing hallmark features (*a.k.a.* radiological abnormalities), including, but not limited to, ground-glass opacities (GGOs), linear opacities, vascular consolidation, reversed halo signs, and crazy-paving patterns. Figure 1.1 provides visual examples of these hallmark features found in COVID-19 patients.

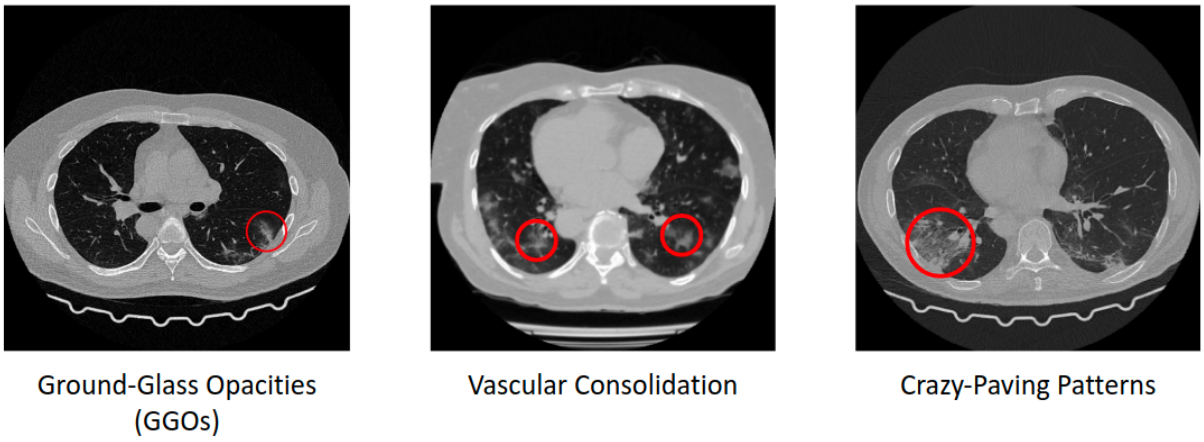


Figure 1.1: Abnormalities in chest CT scans of COVID-19 patients

The literature published in last one year suggests that CT-based medical diagnosis of COVID-19 [20, 22, 41, 68, 73, 76, 90] can outperform ubiquitously deployed RT-PCR test in terms of accuracy [38]. In addition to the improved accuracy, CT-based COVID-19 diagnosis has a faster turnaround time because of better accessibility and less dependency on materials and labor.

To that end, we develop and test deep learning-based algorithms for CT image enhancement. We validate the applicability of image enhancement in medical diagnosis by developing a compute-based deep learning (DL) framework for diagnosing COVID-19 using chest CT scans. The framework delivers much higher sensitivity (91%) than the RT-PCR test (67%) and much faster turnaround time ($\approx$ 5 *minutes*) than RT-PCR ($\approx$ 4 *hours* per test with

multi-day turnaround time). The improved sensitivity is due to the enhanced imaging and analysis of chest CT images and faster turnaround time is because of less dependency on materials and labor.

Continued Importance of COVID-19 Testing Despite the rollout of COVID-19 vaccines resulting in 60% of the U.S. population being vaccinated (but only 50% globally), as of January 10, 2022, there still exists the need for a rapid, accurate, and accessible test for diagnosing COVID-19 *plus* its variants (e.g., B.1.1.7 – Alpha, B.1.351 – Beta, B.1.617.2 – Delta, Omicron – B.1.1.529) and beyond. The Delta variant (B.1.617.2) and Omicron variant (B.1.1.529), for example, have demonstrated increased transmissibility and increased severity of disease [11] resulting in the fourth and fifth wave in the UK, and the USA. Figure 1.2 shows that the number of confirmed cases per day around the world [33, 66].

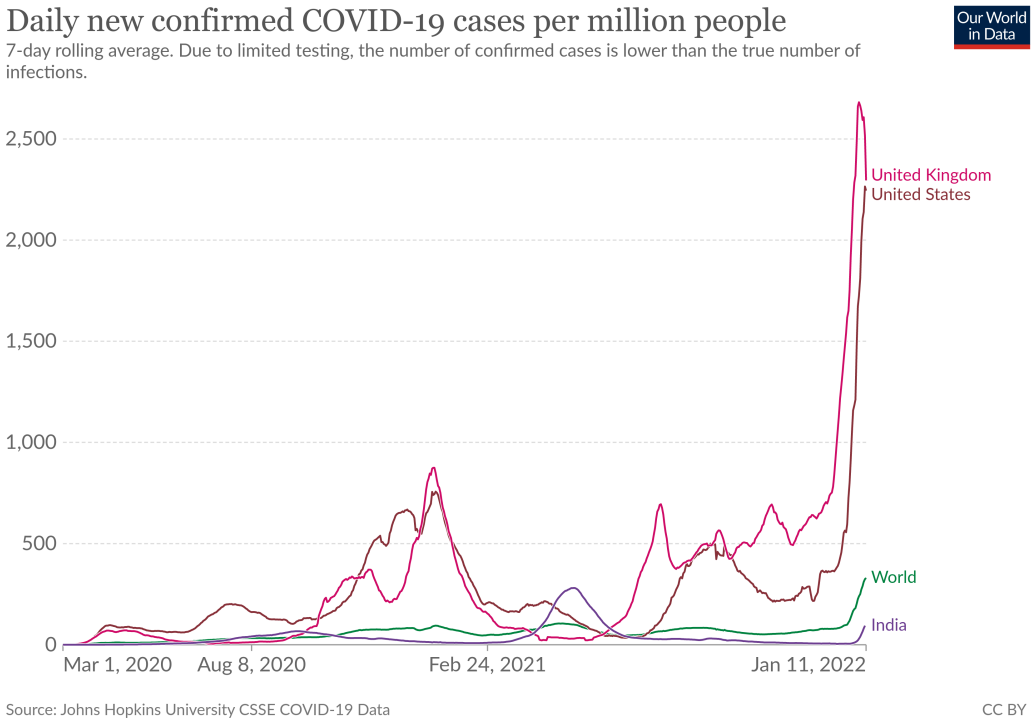


Figure 1.2: Confirmed cases of COVID-19 per million people [33, 66]

1.2 Contributions

This project aims to develop a fast, accurate, and cost-effective algorithm for deep learning-based image enhancement and to verify its applicability in medical diagnosis using a case study of COVID-19 diagnosis using chest CT scans. This work also explores the implementation of AI-based tools on heterogeneous platforms, such as multi-core CPU, many-core GPU, and FPGA. Each contribution is briefly described below.

Fast and Accurate CT Medical Diagnosis using Image Enhancement We research and develop a novel deep learning-based algorithm for high-fidelity CT image construction. The algorithm is based on a convolution neural network (CNN) that uses densely connected convolution operations, *a.k.a.* dense block [26], and deconvolution operations to enhance the quality of CT images. We also modify our deep learning-based 2D image enhancement algorithm to perform a 3D volumetric enhancement. We train both 2D and 3D networks with the same CT data and compare the results of enhancement.

Deep Learning-based COVID-19 Diagnosis using Chest CT Scans To test the applicability of deep learning-based image enhancement and evaluate its advantages, we develop a framework of AI-based tools, called ComputeCOVID19+, that combines image enhancement algorithm and image classification algorithm for high-precision interpretation of COVID-19 CT scans. The framework achieves an overall classification accuracy of 91%. In the process, we also create a dataset containing more than 500 chest CT scans (100,000 CT images) of patients showing COVID-19 symptoms.

Implementation of CNN on Heterogeneous Platforms Training and inference of deep neural networks are computationally expensive processes. Therefore we use hetero-

geneous computing platforms to accelerate training and inference. Specifically, we train the image enhancement neural network on a multi-GPU distributed system. For inference, which is comparatively less computationally expensive than training, we implement image enhancement on multi-core CPU, many-core GPU, and FPGA and evaluate each platform in terms of their power consumption and energy efficiency.

In summary, our work improves medical diagnosis using computed tomography, by making the following contributions:

- Development and evaluation of novel image enhancement algorithms for high-precision interpretation of CT scans.
- Evaluation of applicability of deep learning-based image enhancement in COVID-19 diagnosis.
- Performance evaluation of deep learning algorithms on heterogeneous platforms for speed and accuracy.

1.3 Related Work

The section presents related work from four areas: (1) CT image enhancement, (2) AI with computed tomography (CT), (3) RT-PCR genetic testing vs. CT-based COVID-19 testing, and (4) deep learning on FPGA.

1.3.1 CT Image Enhancement

With the increased use of computed tomography (CT) in medical diagnosis, low-dose X-ray CT has gained popularity recently due to its fast data acquisition and reduced radiation

exposure. However, simple image reconstruction techniques like filtered back projection (FBP) [67] generate low-quality CT images from low-dose X-ray projections. Thus, techniques like iterative image reconstruction [4], sinogram completion [1, 43], and image enhancement based on deep learning (DL) are used to reconstruct high-quality CT images. In particular, DL-based image enhancement has been shown to be the most effective in producing high-quality CT images [8, 10, 19, 31, 42, 81].

Image Enhancement using 2D CNNs

2D CNNs analyze and process each slice in CT scan individually. Therefore, computational and memory requirements of 2D CNNs are much lower as compared to 3D CNNs. The enhancement of CT images to remove streaking artifacts and noise, using both deep learning and deep learning combined with iterative reconstruction algorithms, has been extensively studied in the past.

Würfl et al. emulate FBP using a convolution neural network (CNN) [81]. Cheng et al. combine DL and iterative reconstruction to accelerate the algorithmic convergence using a leapfrogging strategy [10]. Han et al. use a deep residual network to estimate streaking artifacts in low-dose X-ray images [19]. Jin et al. [31] and Chen et al. [8] use FBP for image reconstruction from projection data, followed by applying a U-Net-like CNN for image enhancement.

Volumetric Enhancement using 3D CNNs

Modern computers' computational efficiency and processing ability have enabled the processing and analysis of three-dimensional (3D) images using 3D CNNs. 3D CNNs extract and analyze correlating features between 2D image slices (3D image features), which 2D CNNs

ignore. Some examples of 3D CNNs classifiers for COVID-19 diagnosis are given in §1.3.2.

3D image enhancement using 3D CNNs has not been studied extensively in the past. Le et al. [42] use a simple 3D convolution neural network for volumetric enhancement of CT scans. Their results show improved PSNR (Peak signal-to-noise ratio) and SSIM (structural similarity index measure) for enhanced images compared to the existing state-of-the-art 2D deep learning-based image enhancement algorithms. Similar 3D CNNs, that use convolution and deconvolution operations, are commonly used for volumetric segmentation [9, 34, 35, 35, 56, 91] and 3D super-resolution applications [9].

1.3.2 AI with Computed Tomography

AI-based medical diagnosis is often used in computed tomography and radiology. For example, the use of a convolution neural network (CNN) for the diagnosis of diseases, such as lung cancer, viral pneumonia COVID-19 pneumonia, using CT scans has been extensively studied in the recent past [20, 22, 41, 68, 73, 76, 90].

Two-Dimensional (2D) CNNs with 2D Images as Inputs

For COVID-19 diagnosis, 2D images must be manually selected from 3D CT scans because the associated abnormalities, such as GGO, consolidation, and crazy-paving patterns are present in only some segments of the lungs. Despite this limitation, 2D CNNs still achieve fairly high accuracy in COVID-19 diagnosis. He et al. [22] use VGG-16, ResNet, and DenseNet deep-learning (DL) architectures to classify 2D CT images. They leverage transfer learning, coupled with momentum contrastive learning [21], to make these models agnostic to the size of the datasets. Ultimately, they achieve an accuracy of 86% with a training dataset of 425 2D CT images. Similarly, Wang et al. [76] achieve 89.5% accuracy using an

M-inception network with a dataset containing 1065 CT images. Ying et al. [73] pre-process the 2D images to segment the lung region using OpenCV. The segmented CT images with their *Details Relation Extraction neural network* (DRE-Net) achieve 86% accuracy. Li et al. [41] use U-Net-based lung segmentation and classify the images using ResNet50. They report 90% sensitivity and 96% specificity for COVID-19 diagnosis with their framework.

Three-Dimensional (3D) CNNs with 3D Volumes as Inputs

3D CNNs extract 3D features from the input volume, and thus, COVID-19 diagnosis using 3D CNNs does *not* require any manual data preparation. Harmon et al. [20] demonstrate this by using a 3D version of AH-Net and DenseNet-121 to segment and classify the image, respectively; they achieve 90% accuracy. However, the accuracy of their framework drops to 86% when using our real-world datasets. Zheng et al. [90] combine image segmentation using 2D U-Net and classification using 3D deep convolutional neural network (DeCoVNet) to detect COVID-19 from CT volumes. Their framework achieves 90% accuracy with 540 CT scans.

1.3.3 RT-PCR vs. CT-based COVID-19 Testing

Testing, diagnosing, and monitoring COVID-19 (also known as the SARS-CoV-2 virus) as early and as accurately as possible is essential in mitigating the spread of COVID-19. However, the RT-PCR test has high sensitivity and specificity [77], the accuracy of the test *varies* with the time at which the test is taken. In particular, Johns Hopkins University study in 2020 showed that the false-negative rate of an infected person is 67% on the 4th day (i.e., 33% sensitivity) and improves to 38% (i.e., 62% sensitivity) with the onset of symptoms [38]. Furthermore, given the error-prone (human) process of sample collection, packaging, and de-

livery, the actual end-to-end sensitivity can be much worse. Moreover, the turnaround time and cost of the RT-PCR test is quite high [39], limiting testing capacity on top of the already mediocre sensitivity of the test.

COVID-19 testing based on computed tomography (CT) is a compelling alternative. Research conducted in China with 877 patients [18] shows that 84% of COVID-19 patients exhibited radiographic or CT abnormalities. A larger study in China with 1014 COVID-19 patients [2] shows that 88% of patients (from a biased pool of those who were already showing symptoms of COVID-19) had evidence of CT abnormalities, such as ground-glass opacity (GGO) and consolidation, in their chest CT scans, while only 59% of those same patients tested positive with the initial RT-PCR test. Similar results are reported in [16, 17].

1.3.4 Deep Learning on FPGA

Most of the prior work on FPGA acceleration of DL focus on implementing classification using CNN and optimizing the expensive convolution operations [44, 80, 84, 86]. However, deconvolution (convolution transpose) operation is more computationally expensive than convolution operation for applications such as image enhancement, super-resolution, image segmentation, etc. This shifts the performance bottleneck to deconvolution operation in such CNNs [7, 45, 87].

CNNs on FPGA

CNNs are widely used for image classification applications. Implementing a CNN on FPGA poses many challenges due to limited resources and low memory bandwidth. Solovyey et al. [72] mitigate the challenges as mentioned above by using fixed-point arithmetic instead of floating-point arithmetic to implement a CNN for handwritten digit recognition. Majumdar

et al. [50] use data caching to enhance the performance of convolution layers in a CNN. Cheng et al. [48] use low-precision data types for computation and batch-level parallelization to implement their CNN. They also use tiling by partitioning the input data into smaller tiles that can fit into on-chip memory to improve memory bandwidth and resource utilization. Alawad et al. [3] implement convolution in the stochastic domain and show that their architecture outperforms conventional CNNs with respect to scalability. Qiu et al. [64] realize a complex CNN, VGG16 with 138-million parameters, for large-scale image classification on a Xilinx Zynq FPGA; they also use singular vector decomposition before the fully connected neural network layers to reduce the memory footprint and achieve 86.66% classification accuracy. Caffeine [85] efficiently implements CNN on FPGA with the underlying focus on bandwidth optimization and memory access reorganization.

CNNs with Deconvolution on FPGA

CNNs with deconvolution are widely used in applications, such as image segmentation, super-resolution imaging, and pattern generation. Zhang et al. [87] implement a deconvolution neural network (DCNN), consisting of only the deconvolution operation, on FPGA using 12-bit fixed-point arithmetic and achieve 2.6 GOPs at 100 MHz. Liu et al. [45] propose a significantly optimized solution for deconvolution using a partial result buffer. Their U-Net implementation achieves 107 GOPs for the overall CNN and 29 GOPs for the deconvolution operation with 16-bit fixed precision data types. Chang et al. [7] implement a fast super-resolution convolutional neural network (FSRCNN) with five convolutions and one deconvolution operation and achieve 780 GOPs with 13-bit fixed-point numbers.

1.4 Thesis Organization

The rest of the document is organized as follows. Chapter 2 talks about the image enhancement in computed tomography (CT) using a deep neural network. Specifically, the chapter talks about the CT image enhancement using 2D and 3D version of DenseNet and Deoncolution Network (DDnet) and its application in medical diagnosis of COVID-19 using ComputeCOVID19+ framework. Chapter 3 evaluates the accuracy of 2D and 3D image enhancement quantitatively and qualitatively. Accuracy of ComputeCOVID19+ and the impact of image enhancement in medical diagnosis are discussed in §3.2. Chapter 4 talks about the implementation of DDnet on high performance computing platform, specifically on multi-core CPU, many-core GPU and FPGA, and evaluates each platform in terms of performance, and power efficiency. Chapter 5 analyzes some practical considerations of deploying deep learning-based algorithms for medical diagnosis and also proposes implementation plans for deploying DDnet in CT scanner machines, which can aid radiologists in real-time.

Chapter 2

CT Image Enhancement using a Deep Neural Network

Computed tomography plays an essential role in medical diagnosis by generating three-dimensional images of the body's internal organs. These images are generated from X-ray projections acquired at discrete angles around the body. As mentioned earlier, low X-ray dose computed tomography reduces the risk of cancer associated with CT scans but degrades the quality of reconstructed CT images. Therefore we develop and evaluate deep learning-based image enhancement algorithms to improve the quality of CT images.

Specifically, we use a convolution neural network, called DenseNet and Deconvolution network (DDnet) [88] to enhance CT images reconstructed via filtered back projection (FBP) [67]. The algorithm is also modified to input and enhance 3D CT scans, which considers the correlation between individual 2D image slices. The applicability of CT image enhancement in medical diagnosis is evaluated using a framework for diagnosing COVID-19 using computed tomography.

2.1 2D Image Enhancement

In this work, we use DenseNet and Deconvolution network (DDnet) [88] for CT image enhancement. DDnet consists of a convolution network with 37 convolution layers and a de-

convolution network with eight (8) deconvolution layers. Figure 2.1 shows the architecture of DDnet. The convolution network, deconvolution network, and shortcut connections distinguish DDnet's image enhancement from the existing state-of-the-art. Each distinguishing feature is explained below.

Convolution Network It consists of four dense blocks [26] for feature extraction from the input image. Each dense block, shown in Figure 2.2, consists of four densely connected layers, i.e., the input to each layer is concatenated with the inputs of all the previous layers. These dense connections in the network facilitate feature reuse and mitigate the exploding and vanishing gradient problem. The dense connections are known as local shortcut connections.

Each dense block is followed by a pooling and convolution layer. The pooling layer reduces the size of the feature maps by a factor of two (2) in both the `x` and `y` dimensions, resulting in a network that is more memory efficient and less sensitive to input variations.

Deconvolution Network This is used to reconstruct images from the extracted features. It consists of eight deconvolution layers and four un-pooling layers. (Deconvolution can be viewed as the transpose of convolution.) The un-pooling operation in DDnet scales the feature maps by a factor of two (2) in both the `x` and `y` dimensions using bi-linear interpolation. Table 2.1 shows the size of the input and output feature maps as well as the filters used for each convolution and deconvolution layer.

Shortcut Connections These refer to the concatenation of outputs from different layers in the network. Shortcut connections facilitate feature reuse and better information flow through the network [46], resulting in a better-trained network.

In addition to the local shortcut connections in the convolution network, DDnet uses short-

cut connections from the output of each dense block in the convolution network to the corresponding output of the un-pooling layer in the deconvolution network. These shortcut connections are called global shortcut connections.

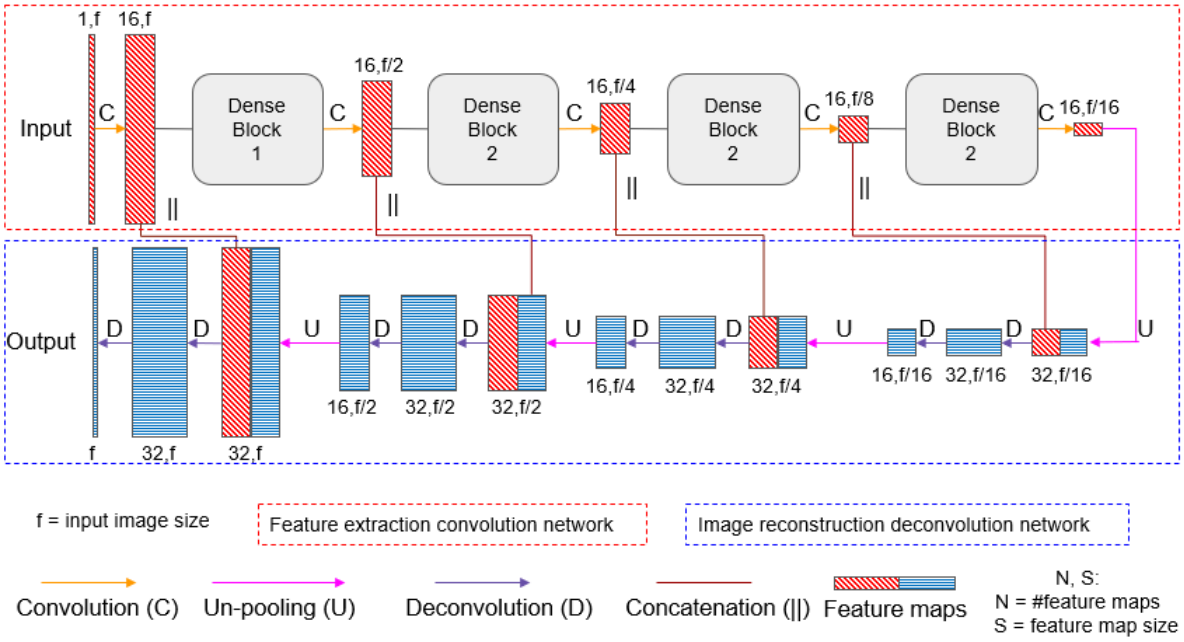


Figure 2.1: The architecture of DDnet

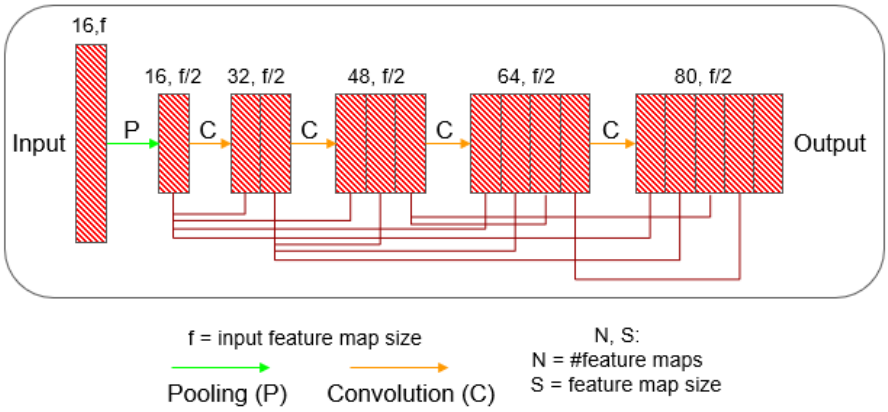


Figure 2.2: The architecture of dense block

Table 2.1: Size of output feature maps and implementation details of each layer in DDnet

Layers	**Output Size**	**Details**
Convolution 1	512×512×16	filter size=7×7, stride=1
Pooling 1	256×256×16	filter size=3×3, stride=2
Dense Block 1	256×256×80	filter size=$\begin{bmatrix} 1\times 1 \\ 5\times 5 \end{bmatrix}$ × 4, stride=1
Convolution 2	256×256×16	filter size=1×1, stride=1
Pooling 2	128×128×16	filter size=3×3, stride=2
Dense Block 2	128×128×80	filter size=$\begin{bmatrix} 1\times 1 \\ 5\times 5 \end{bmatrix}$ × 4, stride=1
Convolution 3	128×128×16	filter size=1×1, stride=1
Pooling 3	64×64×16	filter size=3×3, stride=2
Dense Block 3	64×64×80	filter size=$\begin{bmatrix} 1\times 1 \\ 5\times 5 \end{bmatrix}$ × 4, stride=1
Convolution 4	64×64×16	filter size=1×1, stride=1
Pooling 5	32×32×16	filter size=3×3, stride=2
Dense Block 4	32×32×80	filter size=$\begin{bmatrix} 1\times 1 \\ 5\times 5 \end{bmatrix}$ × 4, stride=1
Convolution 5	32×32×16	filter size=1×1, stride=1
Un-pooling 1	64×64×16	scale factor=2
Deconvolution 1	64×64×32	filter size=5×5, stride=1
Deconvolution 2	64×64×16	filter size=1×1, stride=1
Un-pooling 2	128×128×16	scale factor=2
Deconvolution 3	128×128×32	filter size=5×5, stride=1
Deconvolution 4	128×128×16	filter size=1×1, stride=1
Un-pooling 3	256×256×16	scale factor=2
Deconvolution 3	256×256×32	filter size=5×5, stride=1
Deconvolution 5	256×256×16	filter size=1×1, stride=1
Un-pooling 4	512×512×16	scale factor=2
Deconvolution 6	512×512×32	filter size=5×5, stride=1
Deconvolution 7	512×512×1	filter size=1×1, stride=1

2.1.1 Network Parameters

To find the optimal mapping function that enhances the quality of CT images, DDnet is trained with CT images of size 512×512 pixels. To avoid integer overflow, CT image data, which is usually expressed in the Hounsfield unit (HU), is converted to floating-point data within the data range [0, 1], inclusive, before feeding to the network.

The network uses a composite loss function L for back propagation that combines the mean square error (MSE) and multi-scale structural similarity index metric (MS-SSIM). The MS-SSIM [78] compares the luminance, contrast, and structural similarity between two images. The loss function, L, is given by Equation (2.1).

$$\mathcal{L} = ||y - f(x)||_2^2 + 0.1 \times (1 - L_{MS-SSIM}(Y, f(X))) \tag{2.1}$$

where $||y - f(x)||_2^2$ is the MSE term and $L_{MS-SSIM}$ is the MS-SSIM term.

Network weights are updated using the Adam optimizer [37]. All filters are initialized with a random Gaussian distribution with a mean of zero and a standard deviation of 0.01. The hyper-parameters are tuned by perturbing one parameter while keeping others fixed and analyzing the quantitative results. Training loss curves with varying learning rate, exponential decay, and batch size are shown in Figure 2.3. Optimal values of hyper-parameters are listed in Table 2.4

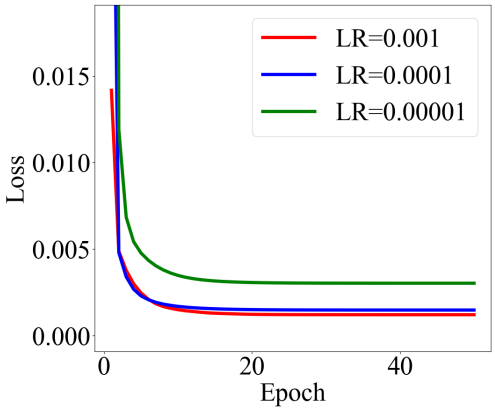
Table 2.2: Optimal values of hyper-parameters for training 2D-DDnet

Parameter	**Value**
Epoch	50
Learning rate	0.001
Learning rate decay	0.95
Batch size	2

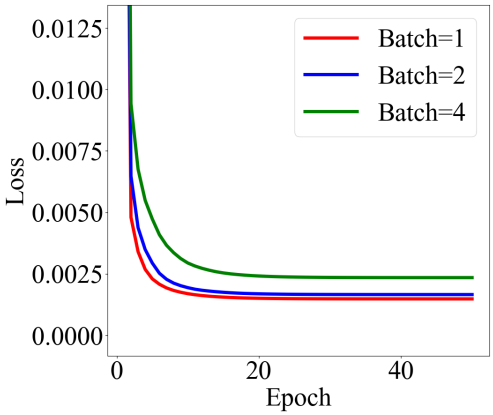
2.1.2 Data Collection

To train our DDnet, we obtained 5120 chest CT images from the following data sources.

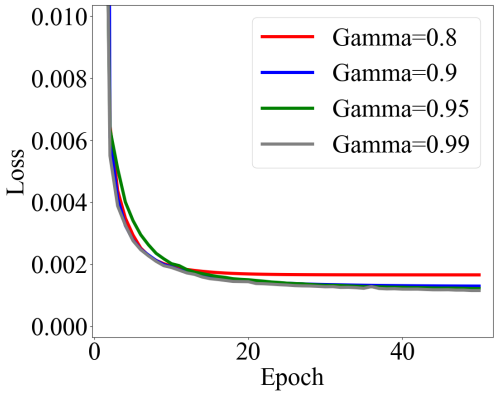
Mayo Clinic Data This data includes chest CT scans, acquired at full and quarter X-ray dosages, of eight patients. The number of projections acquired per CT image is 2304. We



(a) Training loss with varying learning rate. #Epoch=50, Gamma=0.8, Batch=1



(b) Training loss with varying batch size. #Epoch=50, Learning rate=0.0001, Gamma=0.8



(c) Training loss with varying learning rate decay. #Epoch=50, Learning rate=0.0001, Batch=2

Figure 2.3: Hyper-parameters tuning for training 2D-DDnet

used 2286, 300, and 300 images for training, validation, and testing, respectively.

Low X-ray Dose CT Images (Simulated) Though there is an abundance of CT data available for download, low X-ray dose CT images are *not* readily available. Thus, we simulated low X-ray dose chest CT scans for the training and testing of DDnet based on CT scans from the Medical Imaging Databank of the Valencia Region (BIMCV). The dataset contains chest CT scans and X-ray scans of 34 patients that tested positive for COVID-19.

To generate low X-ray dose CT images, we generated projection data from the original CT images using Beer's law and Siddon's ray-driven forward-projection method [71]. The X-ray source was monochromatic at 60 keV. We added Poisson noise according to projection data using the formula $P_i \sim \text{Poisson}\ \{b_i \times e^{l^i}\}$, $i = 1, 2, \ldots, N$, where P_i is the detector measurement along the i^{th} ray path, b_i is the blank scan factor, and l^i is the line integral of attenuation coefficients along the i^{th} ray path. No electronic readout noise was assumed. The Poisson noise (and hence dose) level can be adjusted by setting the number of photons per ray for the blank scan factor b_i. In this study, we uniformly set b_i to 10^6 photons for each ray.

The other CT geometry parameters are summarized below:

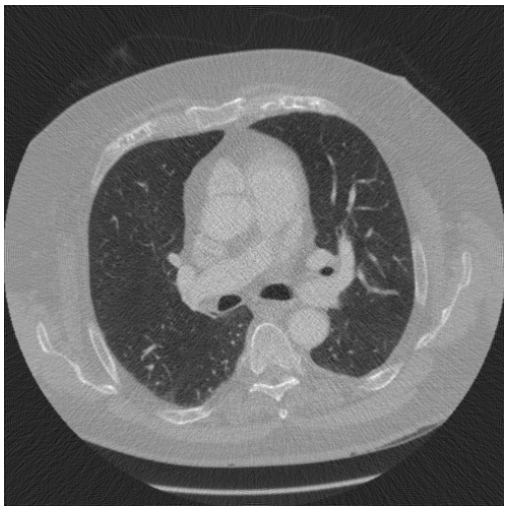
- The distance between source and detector and source and center of the object was set at 1500 mm and 1000 mm, respectively.

- A total of 1440 projections were evenly acquired across a 360-degree scan, i.e., a projection every 0.25°.

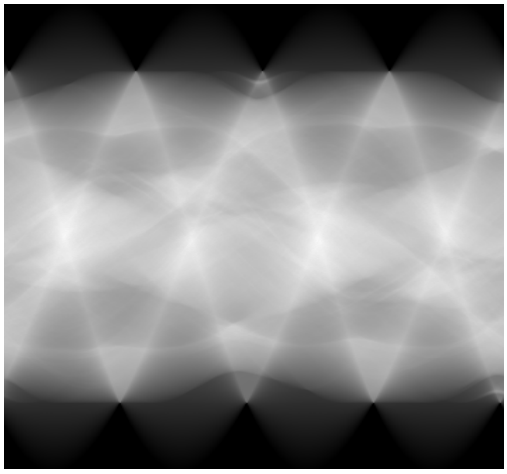
- 1024 pixels were used for X-ray detection.

Low X-ray dose CT images were then reconstructed using filter back projection (FBP) from the simulated projection data. Figure 2.4 shows a sample simulated sinogram and associated CT image reconstructed using FBP. We used 2816, 484, and 484 CT images from the simulated dataset for training, validation, and testing, respectively.

(a) Original CT image



(b) Simulated low-dose CT image



(c) Sinogram from projection data

Figure 2.4: Low X-Ray dose CT image simulation. CT Data source: BIMCV [54]

2.2 3D Volumetric Enhancement

The CT image enhancement using a 2D convolution neural network [10, 82] ignores the correlation between individual 2D image slices. 3D convolution neural networks use 3D cognition to extract and analyze features present in 3D volumes. To that end, we implement 3D volumetric enhancement for CT scans using a 3D adaptation of DDnet. The architecture of the 3D-DDnet is similar to 2D-DDnet, explained in §2.1, except each layer is modified to

process 3D/4D data. These modifications to each layer are explained Table 2.3.

Table 2.3: Summary of operations used in 2D and 3D CNN

Operation	Version	Defiition
Convolution	2D	$output(x,y,z) = bias(z) + \sum_{z_i=0}^{Z_i}\sum_{i=0}^{f\_size}\sum_{j=0}^{f\_size} input(x+i, y+j, z_i) \times filter(i,j,z,z_i)$
	3D	$output(w,x,y,z) = bias(z) + \sum_{z_i=0}^{Z_i}\sum_{i=0}^{f\_size}\sum_{j=0}^{f\_size}\sum_{k=0}^{f\_size} input(w+i, x+j, y+k, z_i) \times filter(i,j,k,z,z_i)$
Non-linear activation (Leaky-relu)	2D	$output(x,y,z) = \begin{cases} input(x,y,x) & \text{if } input(x,y,z) \geq 0 \\ 0.1 \times input(x,y,z) & \text{otherwise} \end{cases}$
	3D	$output(w,x,y,z) = \begin{cases} input(w,x,y,x) & \text{if } input(w,x,y,z) \geq 0 \\ 0.1 \times input(w,x,y,z) & \text{otherwise} \end{cases}$
Batch-normalization*	2D	$output(x,y,z) = \frac{input(x,y,z) - E(z)}{\sqrt{Var(z)+\epsilon}} \times \alpha + \beta$
	3D	$output(w,x,y,z) = \frac{input(w,x,y,z) - E(z)}{\sqrt{Var(z)+\epsilon}} \times \alpha + \beta$
Pooling (Max pool)	2D	$output(x,y,z) = \max_{\{i=0..f\_size-1, j=0..f\_size-1\}} input(x+i, y+j, z)$
	3D	$output(w,x,y,z) = \max_{\{i=0..f\_size-1, j=0..f\_size-1, k=0..f\_size-1\}} input(w+i, x+j, y+k, z)$
Pooling (Linear)	2D	$output(x,y) = f(x_2,y_2) \times \frac{(x_2-x)}{(x_2-x_1)} + f(x_1,y_1) \times \frac{(x-x_1))}{(x_2-x_1)}$
	3D	$output(x,y,z) = f(x_2,y_2,z_2) \times \frac{(x_2-x)}{(x_2-x_1)} + f(x_1,y_1,z_1) \times \frac{(x-x_1))}{(x_2-x_1)}$
Deconvolution	2D	Algorithm 1
	3D	Algorithm 2

* E[x] and Var[x] represent the mean and standard deviation of the inputs, respectively. α and β are the learnable parameters. These parameters are constant in the trained network.

2.2.1 Network Parameters

For training, 3D-DDnet uses the same loss function as 2D-DDnet, defined in Equation (2.1).

For calculating MS-SSIM for 3D volumes, luminance, contrast, and structural similarity

Algorithm 1: 2D-Deconvolution algorithm

Input : $input[X_i][Y_i][Z_i]$, $filter\_bias[Z_o]$, $filter[Z_0][Z_i][filter\_size][filter\_size]$
Output: $out[X_o][Y_o][Z_o]$
Data: y′, x′

```
for z ← 0 to Z_o do
    for y ← 0 to Y_o do
        for x ← 0 to X_o do
            out[z][y][x] ← filter_bias[z]
        end
    end
end
for z ← 0 to Z_o do
    for y ← 0 to Y_o do
        for x ← 0 to X_o do
            for z_1 ← 0 to Z_i do
                for j ← 0 to filter_size do
                    for i ← 0 to filter_size do
                        (y′, x′) ← map_to_output(y, x, j, i)
                        out[z][y′][x′] ← out[z][y′][x′] + input[z_1][y][x] × filter[z][z_1][j][i]
                    end
                end
            end
        end
    end
end
```

are calculated using a moving 3D window [83], in contrast to 2D MS-SSIM, which uses 2D moving window.

Network weights are updated using the Adam optimizer [37]. All filters are initialized with a random Gaussian distribution with a mean of zero and a standard deviation of 0.01. The hyper-parameters are tuned by perturbing one parameter while keeping others fixed and analyzing the quantitative results. Training loss curves with varying learning rate, exponential decay, and batch size is shown in Figure 2.5. Optimal values of hyper-parameters are shown in Table 2.4.

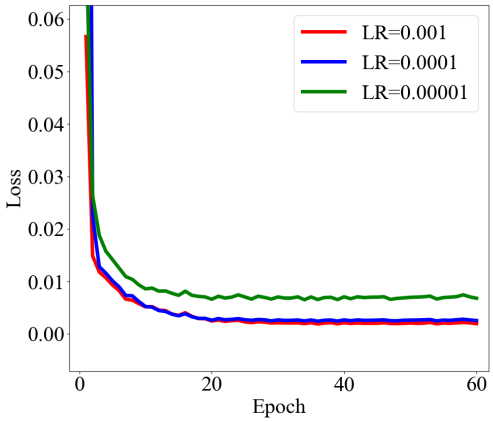
Algorithm 2: 3D-Deconvolution algorithm

Input : $input[W_i][X_i][Y_i][Z_i]$, $filter\_bias[Z_o]$,
$filter[Z_0][Z_i][filter\_size][filter\_size][filter\_size]$
Output: $out[W_o][X_o][Y_o][Z_o]$
Data: y′, x′, w′

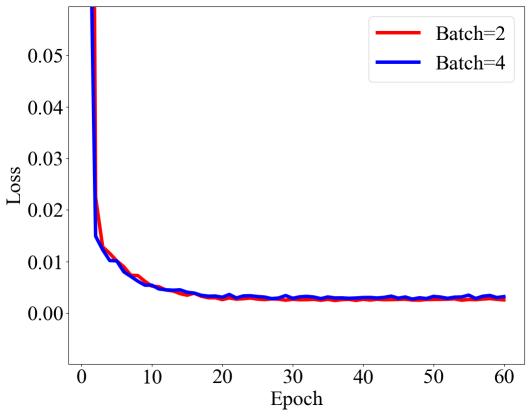
```
for z ← 0 to Z_o do
    for y ← 0 to Y_o do
        for x ← 0 to X_o do
            out[z][y][x] ← filter_bias[z]
        end
    end
end
for z ← 0 to Z_o do
    for y ← 0 to Y_o do
        for x ← 0 to X_o do
            for w ← 0 to W_o do
                for z_1 ← 0 to Z_i do
                    for k ← 0 to filter_size do
                        for j ← 0 to filter_size do
                            for i ← 0 to filter_size do
                                (y', x', w') ← map_to_output(y, x, w, j, i)out[z][y'][x'][w'] ←
                                  out[z][y'][x'][w'] + input[z_1][y][x][w] × filter[z][z_1][k][j][i]
                            end
                        end
                    end
                end
            end
        end
    end
end
```

Table 2.4: Optimal values of hyper-parameters for training 3D-DDnet

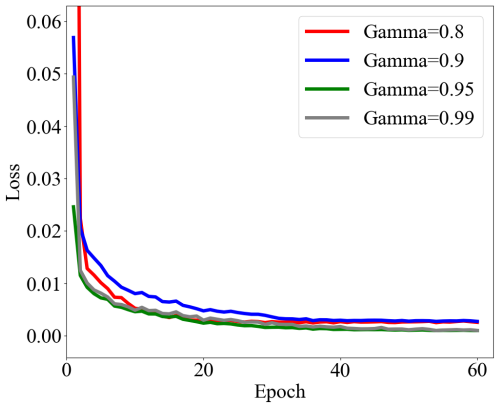
Parameter	**Value**
Epoch	60
Learning rate	0.0001
Learning rate decay	0.95
Batch size	2



(a) Training loss with varying learning rate. #Epoch=60, Gamma=0.8, Batch=2.



(b) Training loss with varying batch size. #Epoch=60, Learning rate=0.0001, Gamma=0.8.



(c) Training loss with varying learning rate decay. #Epoch=60, Learning rate=0.0001, Batch=2.

Figure 2.5: Hyper-parameter tuning for training 3D-DDnet

3D-DDnet has ≈ 5 million training parameters, i.e., $5\times$ the number of training parameters in 2D-DDnet. A large number of training parameters makes the network very sensitive to varying inputs and noise. This leads to an unstable network resulting in exploding and vanishing gradients. We solve this problem by using a batch size of two (2) or more CT scans as input to the network. Batch normalization used in the network normalizes the inputs in each batch so that the network is less sensitive to the noise in input scans [29].

2.2.2 Data Collection

CT scans for training 3D-DDnet are collected from three data sources listed in Table 2.5. These CT scans are used as high-quality target CT scans during the training process. Data simulation is used for generating low-quality 3D CT data. The parameters used for data simulation are same as described in §2.1.2.

Table 2.5: Data sources for training and testing 3D-DDnet

Data Source	**# Scans used for Training Dataset**	**# Scans used for Test Dataset**
Medical Imaging Databank of the Valencia Region (BIMCV)	371	20
Medical Imaging and Data Resource Center (MIDRC)	69	30
Lung Image Database Consortium Image Collection (LIDC)	500	131
Total	940	181

2.2.3 Training Strategies

The computational requirement of 3D-DDnet restricts the size of 3D CT scans that the network can enhance. Enhancement of CT scan containing 2^{22} pixels using 3D-DDnet requires $\approx$ 8GBs of memory. Keeping this in mind, we use various training strategies with different sizes of CT scans for training 3D-DDnet, as mentioned below.

- CT scans of size $256 \times 256 \times 64$ and $512 \times 512 \times 16$ pixels are used for training the network as is.

- CT scans of size $512 \times 512 \times 64$ are divided into four volumes of size $256 \times 256 \times 64$ each. These volumes are used as four (4) independent inputs for training the network. In order to remove the artifacts generated due to zero padding in convolution and deconvolution operation, as shown in Figure 2.6, we pad these tiles in both x and y directions, with 16 additional pixels from original CT scan data.

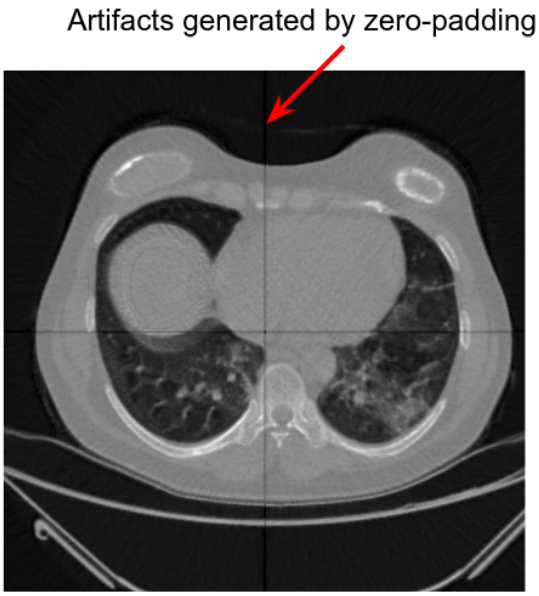


Figure 2.6: Artifacts present in enhanced images due to zero-padding in convolution and deconvolution operations

To maintain a constant number of image slices per input CT scan, image slices in input CT scans are homogeneously selected at equal distances from each other. For example, for a CT scan containing 512 image slices, the input CT scan of 64 image slices consists of every 8^{th} image slice from the original CT scan.

2.3 AI-based COVID-19 Diagnosis using Image Enhancement

Since the discovery of COVID-19 in December 2019, it has rapidly spread and resulted in 313 million confirmed cases and 5.5 million deaths in 192 countries [33], as of January 16, 2022. While symptoms are severe for some, as much as 50% of the population is asymptomatic [62], who unwittingly serve as contagious transmitters. Therefore, the total number of cases reported is arguably less than the actual COVID cases. Specifically, medical experts from

Johns Hopkins University reports that 59% of COVID-19 spread comes from *asymptomatic transmission*, comprising 35% from pre-symptomatic individuals and 24% from individuals who never develop symptoms [32]. Moreover, the accuracy of the traditional COVID-19 RT-PCR test[1] (from collecting samples to packaging and/or handling to testing results) is mediocre with *one in three* producing a *false negative*, i.e., 67% sensitivity [38]. In China, the medical literature cites an even lower 59% sensitivity rate [2, 16].

To that end, we develop a deep learning-based framework, called ComputeCOVID19+ for diagnosing and monitoring COVID-19. ComputeCOVID19+ adapts and extends the DenseNet and Deconvolution neural network [88] to realize high-quality CT imaging of COVID-19. ComputeCOVID19+ can deliver better and more timely diagnostic monitoring for progressing COVID-19 patients. Figure 2.7 provides a high-level overview of the ComputeCOVID19+ framework. Our results show that enhanced CT images can improve the accuracy of CT-based diagnosis of COVID-19 from 86% to 91%.

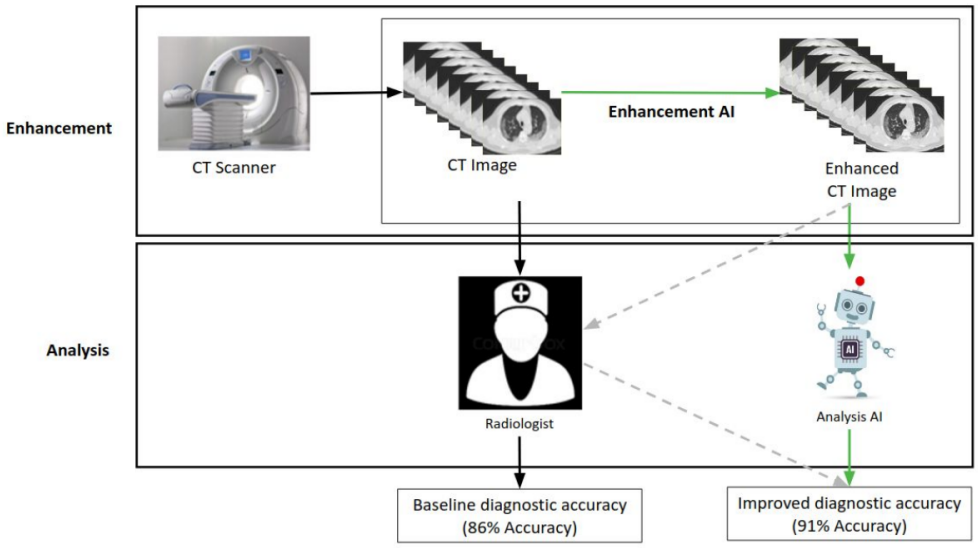


Figure 2.7: ComputeCOVID19+ framework. The green arrows represent the ComputeCOVID19+ workflow, where our image enhancement measurably improves the accuracy of COVID-19 diagnosis.

[1]RT-PCR is a laboratory-based test that stands for reverse-transcriptase polymerase chain reaction.

ComputeCOVID19+ can be downloaded from GitHub [2] and, in turn, deployed to existing CT scanners via a software update. ComputeCOVID19+ will transcend the current COVID-19 outbreak and be applicable to its inevitable variants (as the COVID-19 is an easy-to-mutate RNA virus), which can potentially be even deadlier. Furthermore, the software is not only applicable as a testing and diagnosis tool but also as a monitoring tool for COVID-19.

2.3.1 Approach

The ComputeCOVID19+ framework, based on chest CT and image enhancement algorithm [88], consists of three AI-based tools: (1) Enhancement AI, (2) Segmentation AI, and (3) Classification AI.

We evaluate ComputeCOVID19+ using the workflow shown in Figure 2.8. The first step prepares the data for the training and testing of each AI tool. Next comes Enhancement AI, which enhances CT images using a DenseNet and Deconvolution-based deep neural network (DDnet). The enhanced images are then fed to Segmentation AI for further pre-processing and finally categorized by Classification AI as either a positive or negative COVID-19 scan. The rest of this section elaborates on the ComputeCOVID19+ workflow: data preparation, image enhancement, and image analysis.

2.3.2 Data Preparation

In order to train the Enhancement AI and Classification AI tools, we collected CT scans from four data sources: (1) Mayo Clinic, (2) BIMCV: Medical Imaging Databank of the Valencia Region, (3) MIDRC: Medical Imaging and Data Resource Center, hosted by RSNA, and (4) LIDC: Lung Image Database Consortium Image Collection. These radiological data sources

[2] ComputeCOVID19+ is available at https://github.com/vtsynergy/DL-FACT

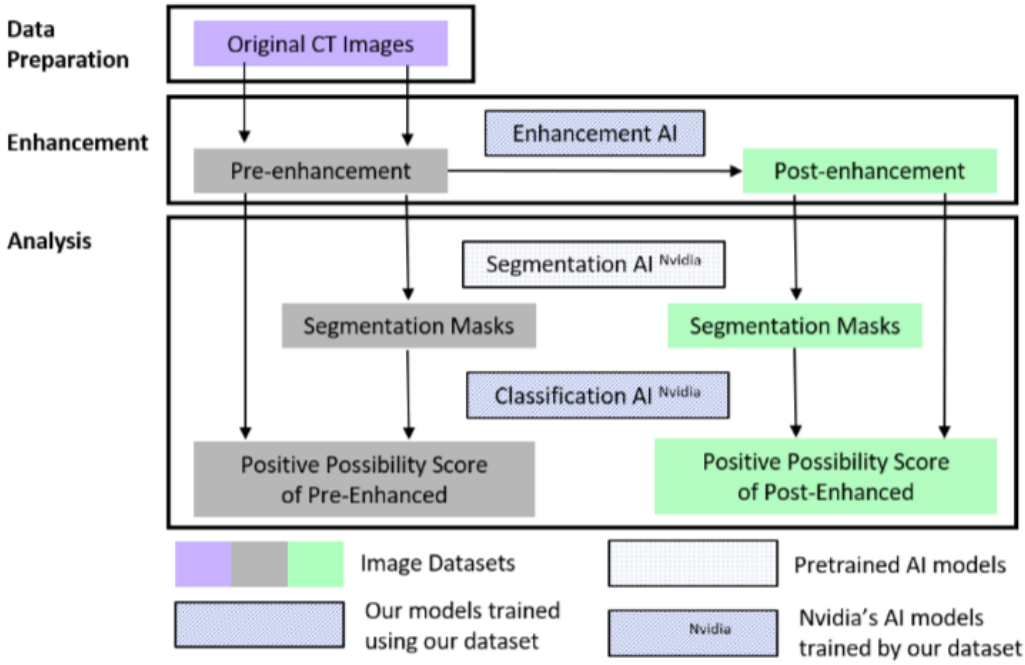


Figure 2.8: Workflow for testing the ComputeCOVID19+ framework

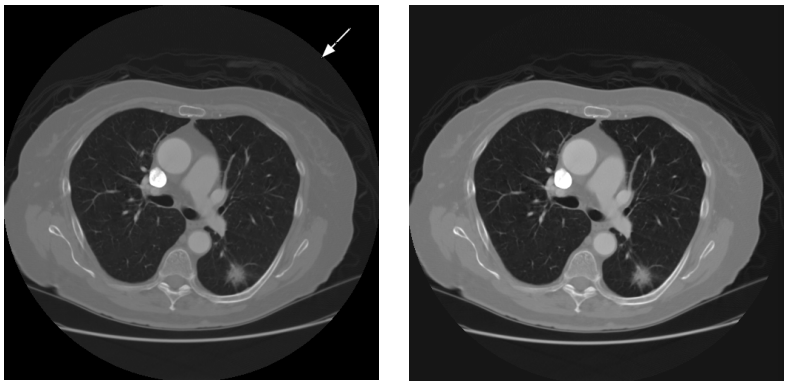
contain 3D chest CT scans composed of 2D image slices, each size 512×512 pixels. Table 2.6 provides a brief description of each data source.

To maintain consistency in CT scans collected from multiple data sources, we performed the following data preparation:

Table 2.6: Data source description

Data Source	Contents
Mayo Clinic	Eight (8) healthy chest CT scans and associated projection data at full-dose & quarter-dose radiation
Medical Imaging Databank of the Valencia Region (BIMCV)	X-ray scans and CT scans of 34 COVID-19 patients
Medical Imaging and Data Resource Center (MIDRC)	229 CT scans of COVID-19 patients
Lung Image Database Consortium Image Collection (LIDC)	1301 healthy chest CT scans

- Retaining only the chest CT scans from the BIMCV dataset, which contains a mixture of CT scans and X-ray images.
- Removal of circular segmentation at the boundary of CT scans from the BIMCV and MIDRC data sets, as shown in Figure 2.9.
- Filtering for CT scans that have more than 128 2D image slices to maintain isotropy in CT scans for better segmentation and classification with 3D networks.



(a) Original CT image with circular segmentation (b) Simulated CT image

Figure 2.9: Removal of circular segmentation in CT images. CT Data source: BIMCV [54]

2.3.3 Image Enhancement

Enhancement AI in ComputeCOVID19+ uses 2D-DDnet for improving the quality of input images. The architecture of the network, the training data and network parameters used for training the network are explained in §2.1

2.3.4 Image Analysis

The ComputeCOVID19+ framework leverages the workflow presented in [20] for the classification of CT images into positive and negative COVID-19 test cases. In particular, we use

the Segmentation AI and Classification AI tools to improve COVID-19 diagnosis using chest CT scans.

Segmentation AI The AI classifies each pixel in the image as foreground or background. In contrast to direct classification methods, segmentation-based classification categorizes an image based on the image and its segmentation mask to change the characteristics of the image to be more meaningful, thus facilitating better interpretation and classification.

The trained Segmentation AI model inputs 3D CT scans and generates a binary map of pixel-wise classification. The lung region in a CT scan is predicted as foreground while the rest of the regions in the scan, including heart, bones, torso, and everything outside the body, are classified as background. The binary map is multiplied with the input CT scan to generate the segmented CT scan.

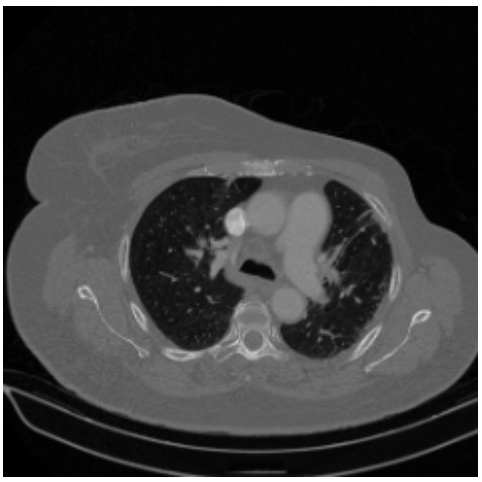
Classification AI To distinguish CT scans with COVID-19 symptoms, Classification AI from the ComputeCOVID19+ framework uses the DenseNet-121 [26] network but adapted for 3D volume classification.

The network uses four densely connected blocks for feature extraction. Each dense block is followed by maximum pooling and a transition convolution layer. Finally, fully connected layers classify the CT scan based on the extracted features.

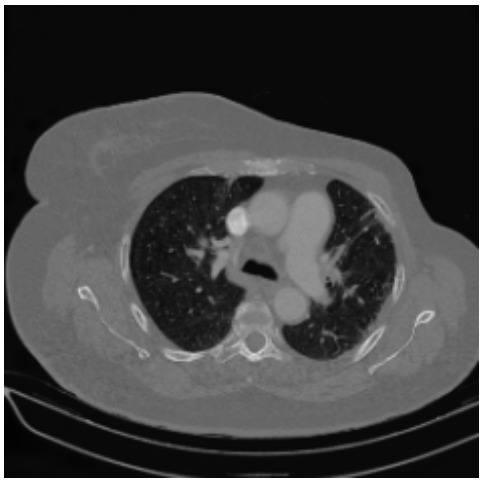
Compared to state-of-the-art classification CNNs (e.g., VGG and ResNet), DenseNet uses fewer parameters and needs less training time because the densely connected convolution layers facilitate feature reuse and better information flow through the network.

ComputeCOVID19+ leverages the pre-trained Segmentation AI and Classification AI model from Nvidia [60].

Complete workflow of ComputeCOVID19+ framework, starting from the original CT image to image classification, is shown in Figure 2.10.



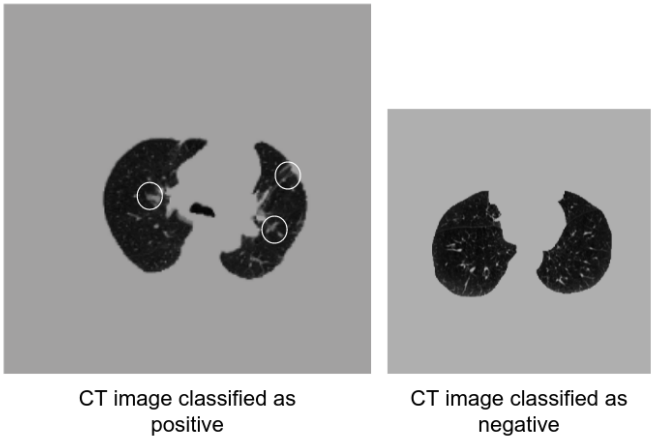
(a) Original CT image



(b) Enhanced CT image

(c) Segmentation mask

(d) Segmented CT image



(e) Classification on the basis of distinct features (highlighted by white circles) present in CT image

Figure 2.10: ComputeCOVID19+ workflow. CT Data source: BIMCV [54] and LIDC [47]

Chapter 3

Evaluating the Accuracy of CT Image Enhancement

In this chapter, we evaluate the efficiency of CT image enhancement using Enhancement AI, and COVID-19 diagnosis using ComputeCOVID19+ framework.

Accuracy of CT image enhancement is evaluated using the mean square error (MSE) and multi-scale structural similarity index metric (MS-SSIM) between the images. Mean square error is calculated using the absolute difference between individual pixels in the image, while MS-SSIM is calculated using the luminance, contrast, and structural similarity within a region of the image.

Accuracy of image classification is evaluated using absolute accuracy of correctly classified images, and Area Under Curve (AUC) of Receiver Operating Characteristic (ROC). ROC curve illustrates the diagnostic ability of classification AI at different threshold values.

3.1 Accuracy of **Enhancement AI**

In this section, we evaluate the accuracy of trained networks for 2D image enhancement and 3D volumetric enhancement separately, and compare both networks trained and tested on the same dataset.

3.1.1 2D Image Enhancement

For evaluating the efficiency of Enhancement AI in improving the quality of CT images, we quantified the network accuracy using the mean square error (MSE) and multi-scale structural similarity index metric (MS-SSIM) between the original CT image and enhanced CT image.

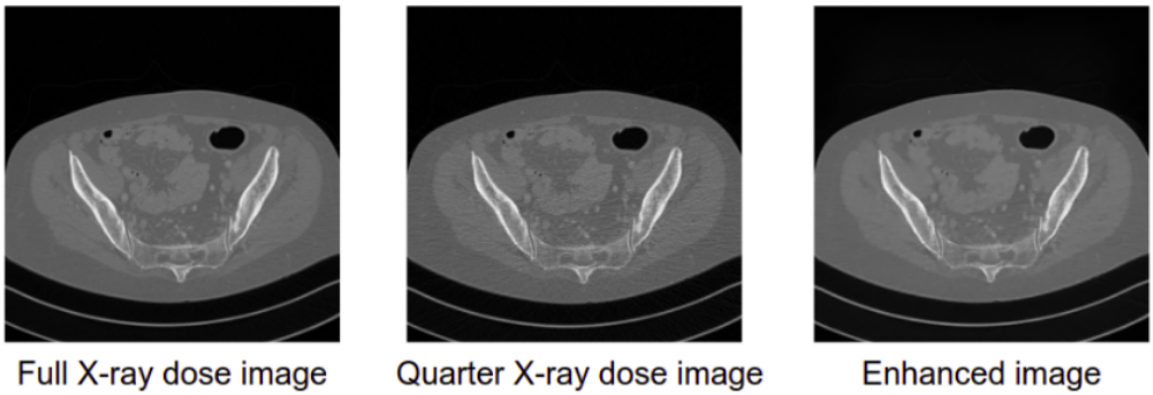
Figure 3.1a(a) shows the result of enhancing chest CT images from the Mayo Clinic dataset. CT images in Mayo Clinic dataset are very high quality even at low X-ray dose. Therefore, there is a minimal difference between the low X-ray dose CT image and the enhanced CT image. Even then, enhancement AI removes some noise present in low X-Ray dose CT images while retaining finer details. Figure 3.1b(b) shows the results of enhancing CT images from a simulated dataset. Enhancement AI removes the streaking artifacts and noise present in the image, reconstructed using FBP from the simulated projection data. The enhanced images for both datasets have well-defined boundaries and fine details.

Quantitatively, Enhancement AI achieved an average of 98.7% multi-scale structural similarity between the high-quality target image and enhanced image for CT images in the testing dataset. Table 3.1 summarizes the quantitative results of Enhancement AI.

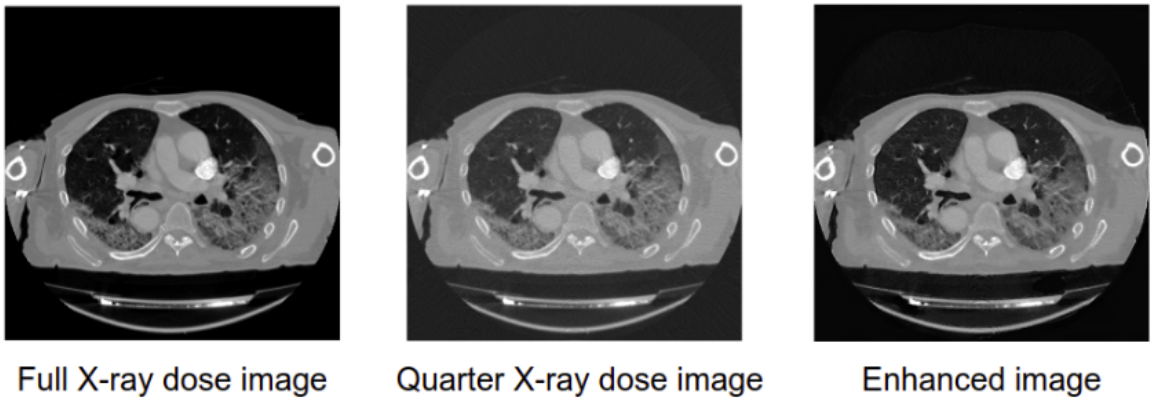
To illustrate the efficacy of CT image enhancement, Figure 3.2 shows the absolute difference maps between a low-quality input image and high-quality target image, and enhanced image and high-quality target image for two datasets. The enhancement AI effectively removes the missing projection artifact and noise present in low X-Ray dose CT images.

Table 3.1: Quantitative results of 2D-DDnet enhancement. Y and X refers to high dose and low dose CT images. f(x) is the image enhanced by DDnet.

	MSE	**MS-SSIM**
Y-X	0.00715	96.2 %
Y-f(X)	0.00091	98.7 %



(a) Mayo clinic dataset. CT Data source: Mayo clinic [52]



(b) Simulated dataset. CT Data source: BIMCV [54]

Figure 3.1: Image enhancement using 2D-DDnet

3.1.2 3D Volumetric Enhancement

Similar to 2D-DDnet, we quantify the accuracy of 3D enhancement using the mean square error (MSE) and multi-scale structural similarity index metric (MS-SSIM) between the original CT image and the enhanced CT image.

Results of 3D enhancement using one image slice from the CT scan of size $256 \times 256 \times 64$ pixels (originally $512 \times 512 \times 64$ pixels in size size and resized to $256 \times 256 \times 64$ pixels), $512 \times 512 \times 16$ pixels, and $512 \times 512 \times 64$ pixels (input to the network as four (4) separate tiles of size $256 \times 256 \times 64$), are shown in Figure 3.3a, 3.3b and 3.3c, respectively. The

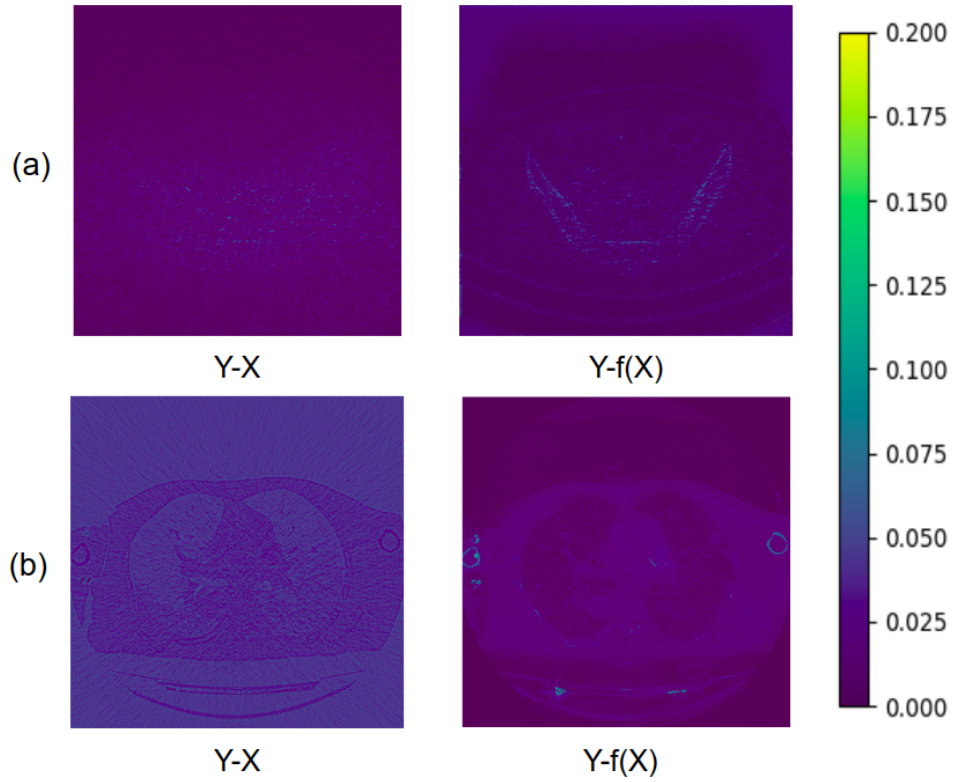
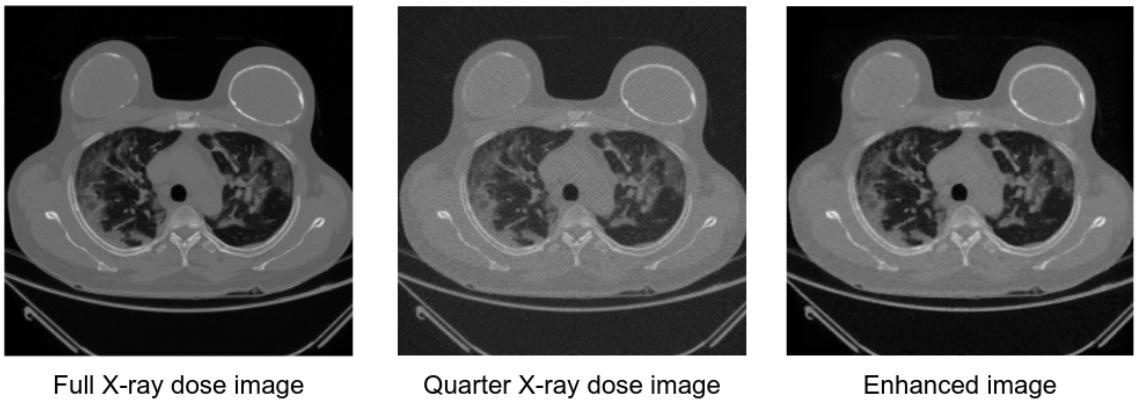


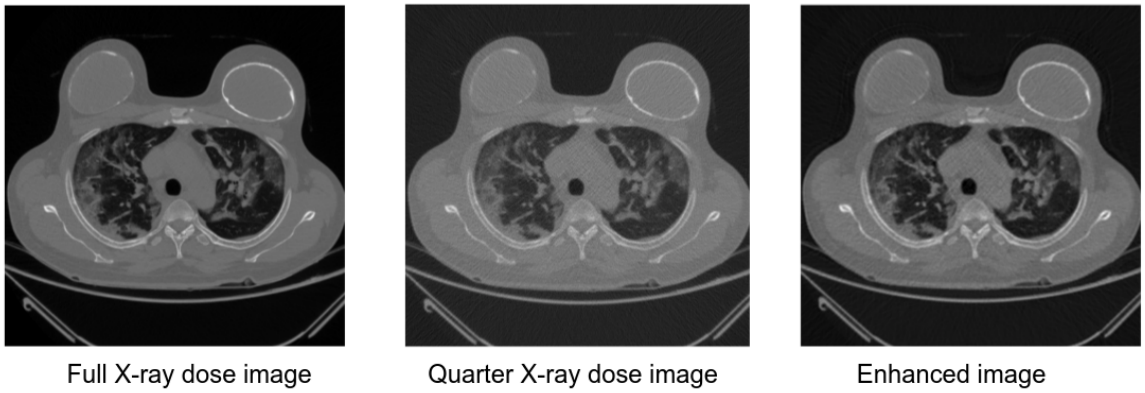
Figure 3.2: Absolute difference maps for (a) Mayo Clinic dataset, and (b) Simulated dataset. Y and X refers to full X-ray dose and quarter X-ray dose CT images. f(x) is the image enhanced by DDnet.

Enhancement AI reconstructs high quality CT scan by removing noise present in the input CT scan while retaining finer details with each dataset. The removal of noise and artifacts from quarter X-ray dose CT scan is clearly visible in absolute difference maps, shown in Figure 3.4.

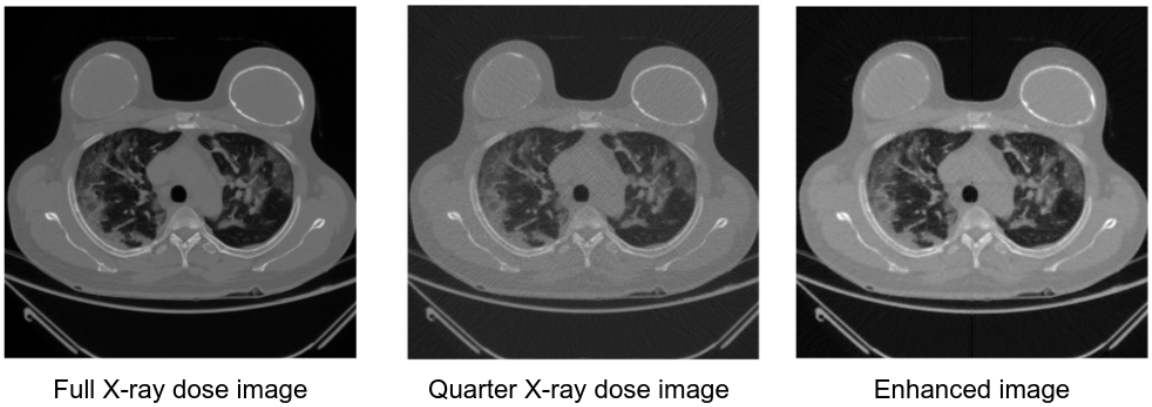
Quantitatively, 3D image Enhancement AI reduces MSE and improves MS-SSIM between high quality target CT scan and enhanced CT scan, as shown in Table 3.2. 3D-DDnet achieves highest accuracy with dataset containing CT scans of size $256 \times 256 \times 64$ pixels, as compared to CT scans of size $512 \times 512 \times 16$ pixels and $512 \times 512 \times 64$ pixels. This is because of the following reasons.



(a) Dataset containing CT scans of size $256 \times 256 \times 64$ pixels



(b) Dataset containing CT scans of size $512 \times 512 \times 16$ pixels



(c) Dataset containing CT scans of size $512 \times 512 \times 64$ pixels

Figure 3.3: 3D volumetric enhancement using 3D-DDnet. CT Data source: MIDRC [53]

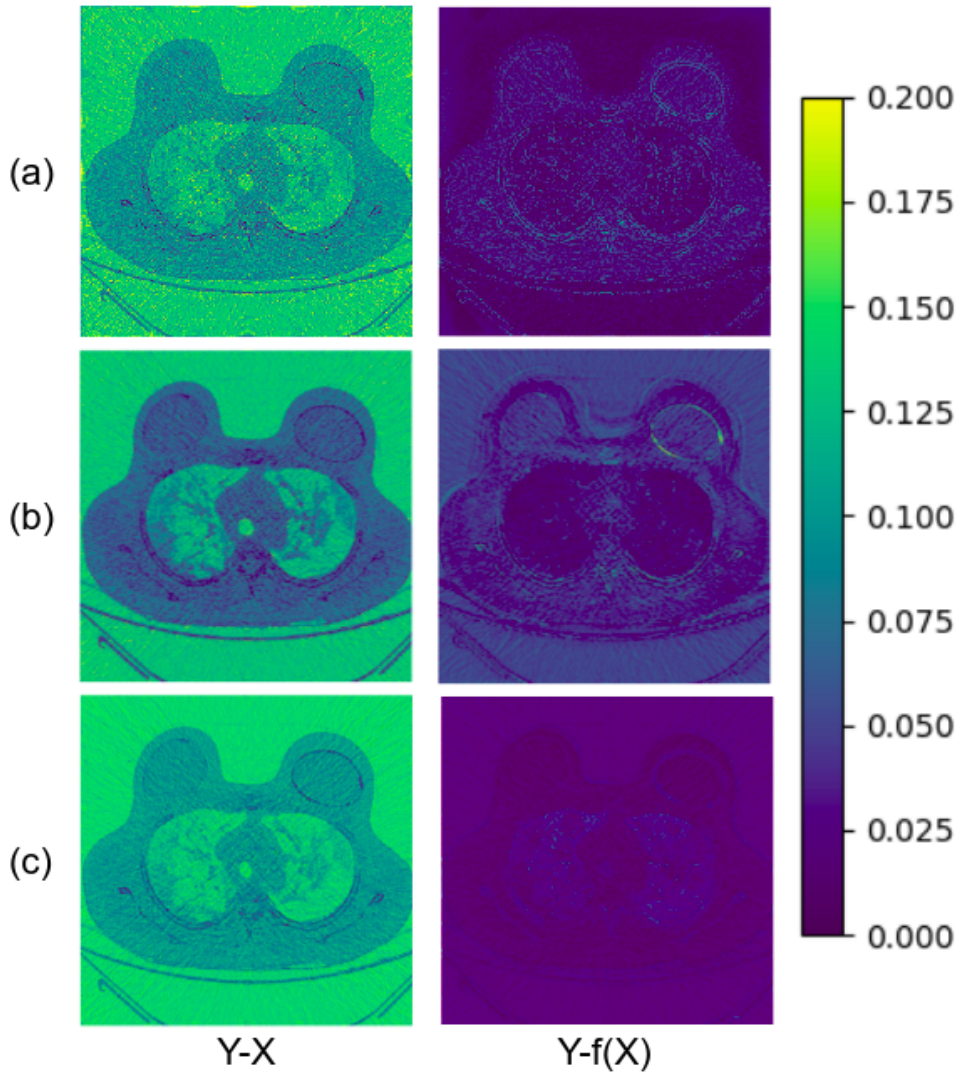


Figure 3.4: Absolute difference maps for CT image slice in (a) $256 \times 256 \times 64$ pixel-sized CT scans, (b) $512 \times 512 \times 16$ pixel-sized CT scans, and (c) $512 \times 512 \times 512$ pixel-sized CT scans. Y and X refers to high dose and low dose CT images. f(x) is the image enhanced by DDnet.

- MS-SSIM between high-quality target and low-quality input CT scans of size $256 \times 256 \times 64$ pixels is higher than that of $512 \times 512 \times 16$ and $512 \times 512 \times 64$ pixel-sized CT scans. Convolution network extracts finer features with higher quality CT scan as input for volumetric reconstruction via deconvolution network.

- Dataset containing CT scans of size $256 \times 256 \times 16$ pixels are more isotropic in terms

Table 3.2: Quantitative results of 3D-DDnet enhancement. Y and X refer to high-dose and low dose CT images. f(x) is the image enhanced by DDnet.

Scan Size	Comparison	MSE	MS-SSIM
$256 \times 256 \times 64$	Y-X	0.01069	94.0%
	Y-f(X)	0.00134	98.6%
$512 \times 512 \times 16$	Y-X	0.01095	88.3%
	Y-f(X)	0.00342	92.2%
$512 \times 512 \times 64$ (4 Tiles)	Y-X	0.01142	89.5%
	Y-f(X)	0.00309	93.1%

of scan size, as compared to the dataset containing CT scans of size $512 \times 512 \times 16$ pixels.

- Tiling of CT scans, of size $512 \times 512 \times 64$ pixels, generates more variability in input than inputs containing whole CT scans. The variability in input requires more complex functional mapping from input to output. Since the architecture of the network is fixed, simpler inputs achieve higher accuracy with the trained network.

Table 3.3: Comparison image and volumetric enhancement using DDnet. Y and X refer to high-dose and low-dose CT scans. f(x) is the CT scan enhanced by DDnet.

Network	Scan Size	Comparison	MSE	MS-SSIM
2D-DDnet	$512 \times 512 \times 64$	Y-X	0.0114	89.7%
		Y-f(x)	0.0074	91.4%
	$512 \times 512 \times 16$	Y-X	0.0109	88.3%
		Y-f(x)	0.0067	90.8%
3D-DDnet	$256 \times 256 \times 64$	Y-X	0.0107	94.0%
		Y-f(x)	0.0013	98.6%
	$512 \times 512 \times 64$ (4 Tiles)	Y-X	0.0114	89.7%
		Y-f(x)	0.0031	93.1%
	$512 \times 512 \times 16$	Y-X	0.0109	88.3%
		Y-f(x)	0.0034	92.1%

3.1.3 Comparison of 2D Image and 3D Volumetric Enhancement

The quantitative comparison of CT scan enhancement using 2D and 3D-DDnet, trained and tested on the same dataset, is shown in Table 3.3. 3D-DDent outperforms 2D-DDnet, for all sizes of CT scans, in terms of accuracy, i.e., in reducing the MSE and improving MS-SSIM between high-quality target CT scans and enhanced CT scans. This is because, in addition to enhancing 2D features in CT image slices, 3D-DDnet enhances 3^{rd} dimensional correlating features between the image slices present in the CT scans, which 2D-DDnet ignores. The enhancement of features in y-z plane, by trained networks, is shown in Figure 3.5. 3D-DDnet enhances the features along the z-direction, while there is minimal enhancement along the z-direction by 2D-DDnet.

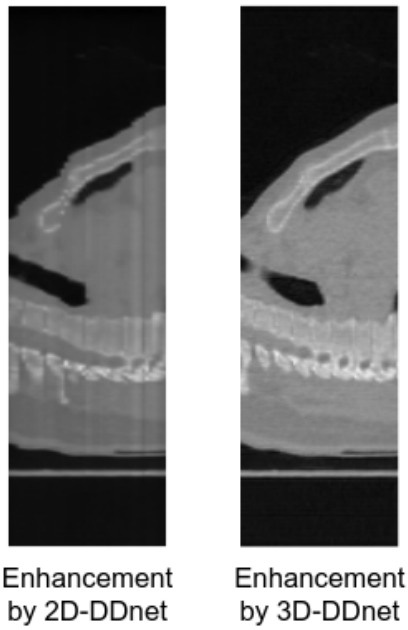


Figure 3.5: Enhancement of features in y-z plane by 2D and 3D-DDnet. CT Data source: MIDRC [53]

3.2 Accuracy of ComputeCOVID19+

For evaluating our ComputeCOVID19+ framework we measure accuracy of classification, as defined by Equation (3.1), and AUC-ROC. The ROC curve graph is plotted using the true-positive rate (TPR), i.e., Equation (3.2), and the false-positive rate (FPR), i.e., Equation (3.3), at different thresholds.

$$Accuracy = (TP + TN)/(TP + FP + FN + TN) \tag{3.1}$$

$$TPR = \frac{TP}{N} = \frac{TP}{TP + FN} \tag{3.2}$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{3.3}$$

where TP is the number of true positives, FN is the number of false negatives, FP is the number of false positives, TN is the number of true negatives, and N is the total number of negatives.

In order to understand the impact of image enhancement in COVID-19 diagnosis, we evaluate the accuracy of classification both with original CT images (i.e., Segmentation AI + Classification AI) and enhanced CT images (i.e., Enhancement AI + Segmentation AI + Classification AI) and compare the results.

Classification with Original Images

The accuracy of segmentation AI and classification AI is evaluated using a dataset containing 95 CT scans, of which 36 are of COVID-19 patients, and 59 have no abnormalities, i.e., healthy. The grey curves in Figures 3.7a and 3.7b show the accuracy and ROC curve, respectively, for the Classification AI tool. When applied to the original CT scans, our

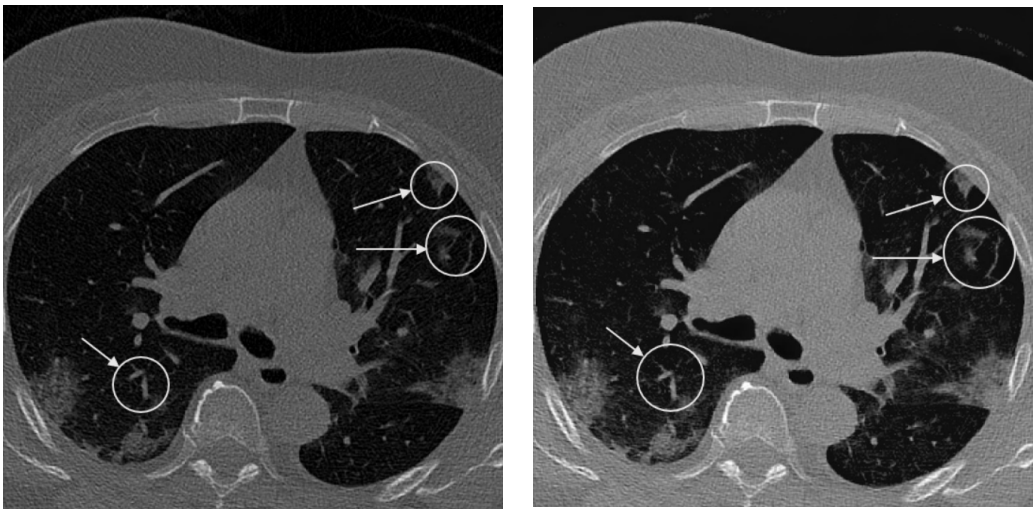
Classification AI tool achieves an accuracy of 86.32% and an AUC-ROC value of 0.890. The accuracy and AUC-ROC jump to 90.53% and 0.942, respectively, when it is applied to the enhanced images from Enhancement AI, as discussed further below.

Classification with Enhanced Images

The inclusion of Enhancement AI distinguishes our ComputeCOVID19+ framework from the existing state of the art for deep learning-based medical diagnosis. The use of Enhancement AI enables the framework to be suitable for low-dose X-ray CT applications.

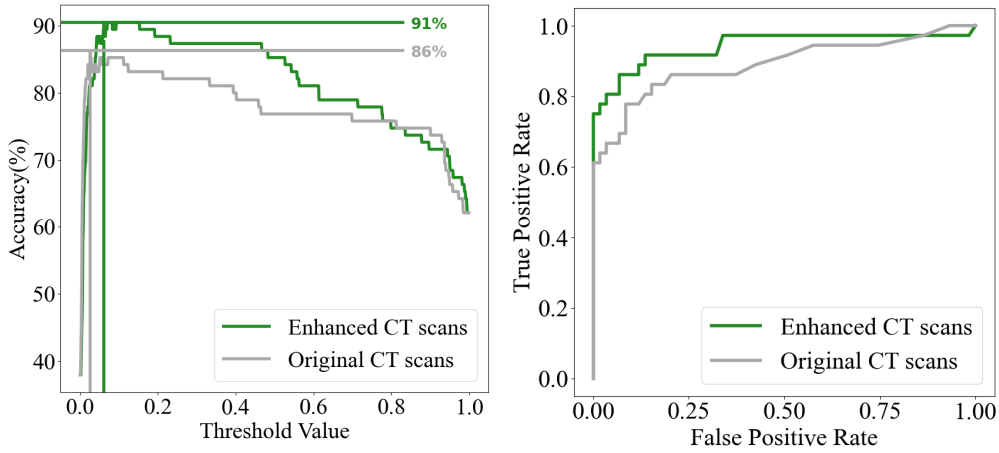
Classification AI outputs the probability of manifestation of distinctive COVID-19 features in the CT scan. Image enhancement facilitates image classification by enabling easier interpretation of high-quality, distinctive features present in enhanced CT scans. This improves the average output probability of *COVID-19 scans* to be correctly classified by 0.1136. Figure 3.6 shows a pair of original and enhanced chest CT images, in which small ground-glass opacities (GGOs) are more clearly visible and easier to interpret in enhanced CT image than in original CT image.

The efficacy of Enhancement AI is also demonstrated by the improved accuracy and ROC curves for classification from the original CT scans to the enhanced CT scans in Figure 3.7. The improved accuracy and ROC curves of ComputeCOVID19+'s classification are shown in green. Using the enhanced CT scans from Enhancement AI, the absolute accuracy of classification improved from 86% to 91%, and the AUC-ROC value increased from 0.890 to 0.942, as shown in Figures 3.7a and 3.7b respectively. Table 3.4 shows the result of the classification of the test dataset using a confusion matrix at an optimal threshold value of 0.061.



(a) Original CT image (b) Enhanced CT image

Figure 3.6: A CT image of a COVID-19 patient before enhancement (left) and its counterpart after enhancement (right). Smaller ground-glass opacities (GGOs) become easier to interpret in enhanced image. CT Data source: BIMCV [54]



(a) Classification accuracy (b) ROC curve for classification

Figure 3.7: ComputeCOVID19+ evaluation

Table 3.4: Confusion matrix for classification of test data set

		Ground-Truth Class	
		Positive	**Negative**
Predicted Class	**Positive**	True Positive 31	False Positive 4
	Negative	False Negative 5	True Negative 55

Chapter 4

Evaluating the Performance of CT Image Enhancement

Training and inference of deep neural networks are computationally expensive processes. High-performance computing with heterogeneous platforms allows us to exploit parallelization during training and inference to reduce the runtime. However, even with parallel execution, network training can take days to execute. In addition, the memory requirement of deep neural networks and the limited bandwidth available on the executing platform is a challenging problem.

To that end, we exploit batch-level parallelism to train Enhancement AI on multiple GPUs in a distributed setup. For inference, we implement Enhancement AI using OpenCL, and apply a set of application-specific and architecture-specific optimizations to maximize the utilization of available resources on the executing platform. We then evaluate the efficacy of our Enhancement AI inference on multi-core CPU, many-core GPU, and FPGA.

4.1 Approach

Training and inference of convolution neural networks are computationally expensive and require large computational and memory bandwidth. Accelerating these processes on parallel computing devices requires knowledge of the underlying hardware and processing de-

mands for adequate utilization of the available computing and memory resources. In §4.1.1 and §4.1.2, we describe the optimization of AI training on a multi-GPU system using PyTorch and the implementation and optimization of AI inference on heterogeneous platforms using OpenCL, respectively.

4.1.1 Training of Enhancement AI

We implemented Enhancement AI using PyTorch and parallelized it for a multi-GPU system using the DistributedDataParallel package [63], which exploits batch-level parallelism and parallelizes AI training by spawning one process per GPU. During training, forward propagation is executed independently, while the gradients are synchronized during back propagation to maintain consistency in the model present on each GPU. We used the gloo communication backend [15] to synchronize processes.

4.1.2 Inference of Enhancement AI

Inference with our Enhancement AI tool is not as computationally expensive as the training and can thus be performed on a single node containing multi-core CPU(s), many-core GPU(s), or FPGAs. Inference involves all the steps used in training except for the back propagation and weight updates. Thus, we performed inference by removing the back propagation and weight update steps from our PyTorch implementation. Along with our PyTorch implementation, we created and evaluated the performance of an equivalent inference implementation in OpenCL [57].

Inference requires seven operations for image enhancement: convolution, non-linear activation, batch normalization, pooling, deconvolution, un-pooling, and concatenation. Each operation is defined below.

Convolution Convolution extracts features from a given input image. It requires a multiply and accumulation operation over a moving window of a given size. Mathematically, convolution operation is defined in Equation (4.1). The corresponding algorithm for convolution is shown in Algorithm 3.

$$output(x,y,z) = bias(z) + \sum_{z_i=0}^{Z_i} \sum_{i=0}^{f\_size} \sum_{j=0}^{f\_size} input(x+i, y+j, z_i) \times filter(i,j,z,z_i) \tag{4.1}$$

Algorithm 3: Convolution algorithm

Input : $input[X_i][Y_i][Z_i]$, $filter\_bias[Z_o]$, $filter[Z_0][Z_i][filter\_size][filter\_size]$
Output: $out[X_o][Y_o][Z_o]$
Data: y', x', res

```
for z ← 0 to Z_o do
    for y ← 0 to Y_o do
        for x ← 0 to X_o do
            for z_1 ← 0 to Z_i do
                for j ← 0 to filter_size do
                    for i ← 0 to filter_size do
                        (y', x') ← map_to_input(y, x, j, i)
                        res ← res + input[z_i][y'][x'] × filter[z][z_1][j][i]
                    end
                end
            end
            out[z][y][x] ← res + filter_bias[z]
        end
    end
end
```

Non-linear Activation Activation functions incorporate non-linearity into the neural network. Equation (4.2) defines the leaky rectified linear unit (ReLU), which is used as an activation function in DDnet.

$$output(x,y,z) = \begin{cases} input(x,y,x) & \text{if } input(x,y,z) \geq 0 \\ 0.1 \times input(x,y,z) & \text{otherwise} \end{cases} \tag{4.2}$$

Batch Normalization Batch normalization, given by Equation (4.3), normalizes the input to the network. The operation speeds up the training process by regularizing the inputs.

$$output(x,y,z) = \frac{input(x,y,z) - E(z)}{\sqrt{Var(z) + \epsilon}} \times \alpha + \beta \tag{4.3}$$

E[x] and Var[x] represent the mean and standard deviation of the inputs, respectively. α and β are the learnable parameters. These parameters are constant in the trained network.

Pooling Pooling downsamples the input feature maps, making the network more memory-efficient and less sensitive to input variations. Pooling works over a moving window of a given size. In our implementation, we use *max pooling*, as defined by Equation (4.4), for implementing DDnet.

$$output(x,y,z) = \max_{\{i=0..f\_size-1, j=0..f\_size-1\}} input(x+i, y+j, z) \tag{4.4}$$

Un-pooling Un-pooling, the reverse of pooling, restores the size of feature maps. We used linear interpolation to implement un-pooling in DDnet. Linear interpolation in the x-direction is given by Equation (4.5).

$$output(x,y) = f(x_2,y_2)(x_2{-}x)/(x_2{-}x_1) + f(x_1,y_1)(x{-}x_1)/(x_2{-}x_1) \tag{4.5}$$

where $f(x_1, y_1)$ and $f(x_2, y_2)$ are the pixel values at (x_1, y_1) and (x_2, y_2) in the given feature map respectively.

Deconvolution Deconvolution, also known as transpose convolution, reconstructs high-quality images from the features extracted by convolution network. Pseudo-code for the deconvolution operation is given in Algorithm 1.

Concatenation Shortcut connections used in DDnet are implemented using concatenation operation. This operation concatenates feature maps output from different layers along the z-axis.

Implementation Each operation is implemented as an independent OpenCL kernel. The data exchange between the host (CPU) and device (GPU or FPGA) is minimized using the available global memory. The C++ wrapper classes provide PyTorch-like APIs for implementing deep neural networks using OpenCL.

Due to the size of DDnet in terms of the number of layers in the network and the size of feature maps in intermediate layers, convolution and deconvolution operations require a large number of global memory accesses resulting in under-utilization of available resources due to memory bandwidth bottleneck. DDnet uses 37 convolution layers and eight deconvolution layers. Table 4.1 summarizes the sizes of the input and output feature maps and filter sizes used in the convolution and deconvolution layers of DDnet.

Table 4.2 shows the absolute global load and store operations and floating-point operations executed in each kernel for 512×512×32 inputs. For evaluation, we use a 5×5 filter for the convolution and deconvolution operations. Pooling and un-pooling operations reduce and scale the size of feature maps by a factor of two, respectively. A large number of

Table 4.1: Input and output size, and size of filters used in convolution and deconvolution layers in DDnet

Layer	1	2	3	4	5	6	7	8	9
Kernels	conv1	conv2	conv3	conv4	conv5	conv6	conv7	conv8	conv9
Input size	512×512×1	256×256×16	256×256×64	256×256×32	256×256×64	256×256×48	256×256×64	256×256×64	256×256×64
Output size	512×512×16	256×256×64	256×256×16	256×256×64	256×256×16	256×256×64	256×256×16	256×256×64	256×256×16
Filter size	7×7	1×1	5×5	1×1	5×5	1×1	5×5	1×1	5×5

Layer	10	11	12	13	14	15	16	17	18
Kernels	conv10	conv11	conv12	conv13	conv14	conv15	conv16	conv17	conv18
Input size	256×256×80	128×128×16	128×128×64	128×128×32	128×128×64	128×128×48	128×128×64	128×128×64	128×128×64
Output size	256×256×16	128×128×64	128×128×16	128×128×64	128×128×16	128×128×64	128×128×16	128×128×64	128×128×16
Filter size	1×1	1×1	5×5	1×1	5×5	1×1	5×5	1×1	5×5

Layer	19	20	21	22	23	24	25	26	27
Kernels	conv19	conv20	conv21	conv22	conv23	conv24	conv25	conv26	conv27
Input size	128×128×80	64×64×16	64×64×64	64×64×32	64×64×64	64×64×48	64×64×64	64×64×64	64×64×64
Output size	128×128×16	64×64×64	64×64×16	64×64×64	64×64×16	64×64×64	64×64×16	64×64×64	64×64×16
Filter size	1×1	1×1	5×5	1×1	5×5	1×1	5×5	1×1	5×5

Layer	28	29	30	31	32	33	34	35	36
Kernels	conv28	conv29	conv30	conv31	conv32	conv33	conv34	conv35	conv36
Input size	64×64×80	32×32×16	32×32×64	32×32×32	32×32×64	32×32×48	32×32×64	32×32×64	32×32×64
Output size	64×64×16	32×32×64	32×32×16	32×32×64	32×32×16	32×32×64	32×32×16	32×32×64	32×32×16
Filter size	1×1	1×1	5×5	1×1	5×5	1×1	5×5	1×1	5×5

Layer	37	38	39	40	41	42	43	44	45
Kernels	conv37	deconv1	deconv2	deconv3	deconv4	deconv5	deconv6	deconv7	deconv8
Input size	32×32×80	64×64×32	64×64×32	128×64×32	128×128×32	256×256×32	256×256×32	512×512×32	512×512×32
Output size	32×32×16	64×64×32	64×64×16	128×64×32	128×128×16	256×256×32	256×256×16	512×512×32	512×512×1
Filter size	1×1	5×5	1×1	5×5	1×1	5×5	1×1	5×5	1×1

floating-point operations, coupled with a significant number of load-store operations make DDnet computationally expensive. In §4.1.3, we discuss a number of optimizations aimed at improving the overall performance.

Table 4.2: Global memory load/store and floating-point operations count for individual kernels with an input of size 512×512×32 floats

Kernels	**Global Memory Loads Operations** (10^6)	**Global Memory Store Operations** (10^6)	**Floating-point Operations** (10^6)
Convolution	13421.7	8.4	13421.7
Deconvolution	13421.7	6710.8	13421.7
Pooling	18.9	2.1	0
Un-pooling	134.3	33.5	469.7
Leaky-ReLU	8.4	8.4	8.4
Batch Normalization	41.9	8.4	41.9

4.1.3 Optimizing Inference

To reduce the runtime by maximizing the hardware utilization on each platform, we applied a set of application-specific and architecture-aware optimizations and FPGA-specific optimizations for each OpenCL kernel. These are explained below.

Application-specific Optimizations

Refactoring with Inverse Coefficient Mapping Recurring load and store operations in deconvolution, as shown in line 15 of Algorithm 1, degrade the performance of the kernel. To overcome this, we use inverse coefficient mapping as explained in [7, 87].

In a refactored kernel for deconvolution, we determine the input blocks needed to calculate each output element instead of directly deconvolving the input. Then, each element in the input block and corresponding weight coefficient are multiplied and added at once before the result is written to the global memory. Figure 4.1 illustrates the optimized operation while Algorithm 4 realizes the corresponding pseudo-code.

Data Reorganization The data in global memory stored in row-major format limits memory coalescing. For example, with 5x5 convolution and deconvolution, a maximum of 5 memory accesses can be coalesced when the feature maps are stored in column-major format.

To address this, the feature maps and network parameters are stored in the memory as shown in Figure 4.2. VEC_SIZE in the figure represents the maximum size of vectorization that can be applied in the modified kernel. We use 16-way vectorization in our implementation because it is the maximum vectorization implicitly defined in OpenCL for floating-point operations and memory accesses. For memory alignment and consistency, few feature maps are padded with zero as needed. This introduces a little memory overhead, but data reor-

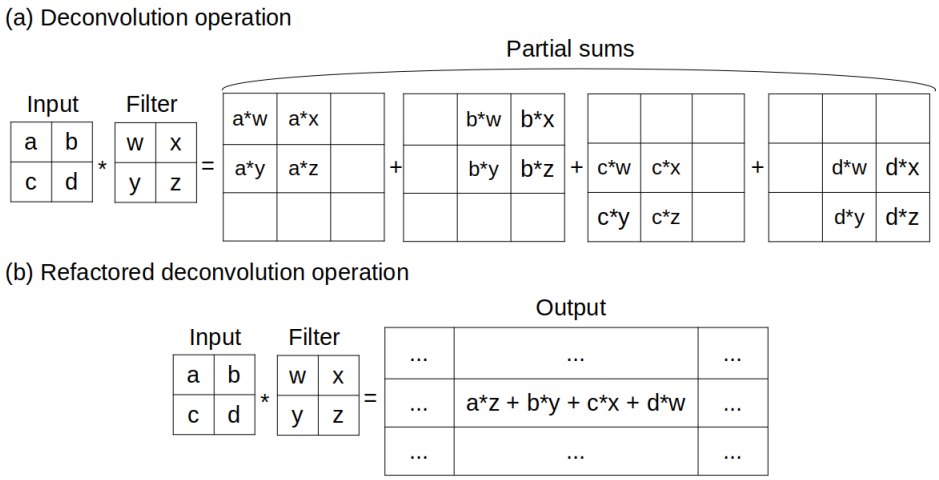


Figure 4.1: Deconvolution optimization (a) Deconvolution operation: Partial sums are calculated by multiplying an element in input and each element in filter. These partial sums are then added to get the final output. (b) Refactored deconvolution operation: Each output is calculated by determining which input elements affects that output and applying multiply and add operations before being written.

ganization considerably reduces the memory bandwidth requirement. The pseudo-code for reorganizing the data and modified convolution operation with reorganized data are given in Algorithm 5 and 6.

The advantages of using reorganized data are three folds: (1) Memory coalescing: Multiple memory accesses can be combined into a single transaction, (2) Vectorization: The reorganization of data allows vectorized memory access and floating-point operations, (3) Data reuse: With reorganized data, feature map data is prefetched and reused for calculating the entire output feature map at once.

Architecture-aware Optimizations

Kernel-launch Configuration OpenCL allows programmers to invoke the kernels in two configurations: NDRange, and single task. On GPU and CPU, NDRange kernel execution launches a group of work items on available compute units. These work items run indepen-

Algorithm 4: Refactored deconvolution algorithm

Input : $input[X_i][Y_i][Z_i]$, $filter\_bias[Z_o]$,
$filter[Z_0][Z_i][filter\_size][filter\_size]$
Output: $out[X_o][Y_o][Z_o]$
Data: sum

```
1  for z ← 0 to Z_o do
2      for y ← 0 to Y_o do
3          for x ← 0 to X_o do
4              sum ← 0
5              input_block ← block of input elements that determines out[z][y][x]
6              for z_1 ← 0 to Z_i do
7                  for j ← 0 to filter_size do
8                      for i ← 0 to filter_size do
9                          sum ← sum + input_block[z_1][j][i] × filter[z][z_1][j][i]
10                     end
11                 end
12             end
13             out[z][y][x] ← sum + filter_bias[z]
14         end
15     end
16 end
```

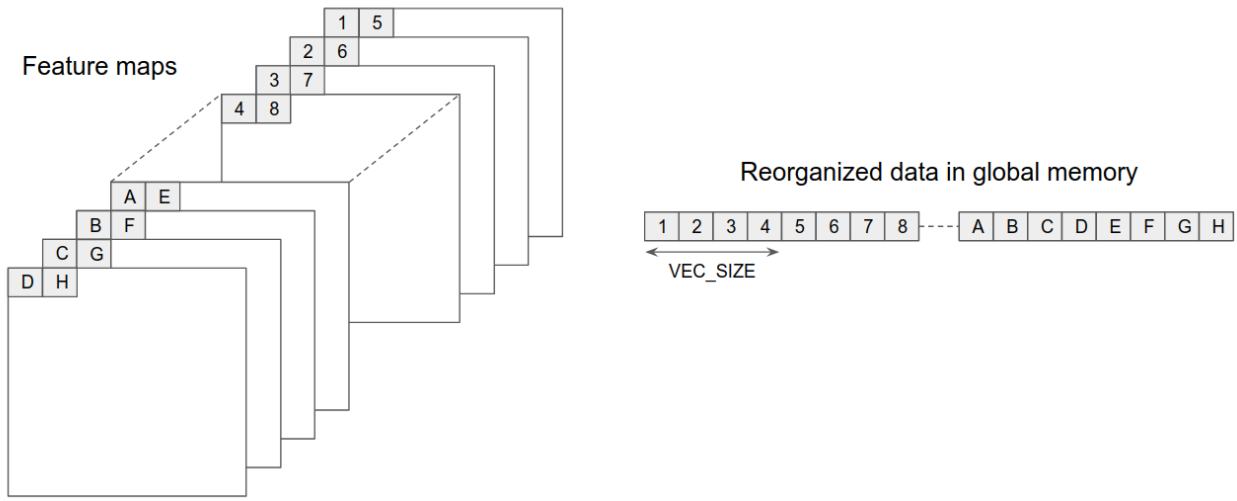


Figure 4.2: Reorganization of data in feature maps to maximize memory coalescing

Algorithm 5: Data reorganization algorithm

Input : $input[X_i][Y_i][Z_i]$, $filter[Z_0][Z_i][filter\_size][filter\_size]$, VEC_SIZE
Output: $input\_reorder[X_i \times Y_i \times Z_i]$, $filter\_reorder[Z_0 \times Z_i \times filter\_size \times filter\_size]$
Data: sum

```
for z ← 0 to Z_i/VEC_SIZE do
    for y ← 0 to Y_i do
        for x ← 0 to X_i do
            for w ← 0 to VEC_SIZE do
                input_reorder[(z × Y_i × X_i × VEC_SIZE) + (y × X_i × VEC_SIZE) + (x ×
                  VEC_SIZE) + w] ← input[x][y][z ∗ VEC_SIZE) + w]
            end
        end
    end
end
for z ← 0 to Z_o/VEC_SIZE do
    for z_1 ← 0 to Z_i/VEC_SIZE do
        for y ← 0 to filter_size do
            for x ← 0 to filter_size do
                for w ← 0 to VEC_SIZE do
                    for v ← 0 to VEC_SIZE do
                        filter_reorder[(z × Z_i × filter_size × filter_size ×
                          VEC_SIZE) + (z_1 × filter_size × filter_size × VEC_SIZE ×
                          VEC_SIZE) + (y × filter_size × VEC_SIZE × VEC_SIZE) +
                          (x × VEC_SIZE × VEC_SIZE) + (w × VEC_SIZE) + v] ←
                          filter[x][y][z_1 ∗ VEC_SIZE) + v][z × VEC_SIZE) + w]
                    end
                end
            end
        end
    end
end
```

dently unless synchronized. On the other hand, a single work items kernel launches one work item that runs on one compute unit.

The NDRange kernel launches a group of work items that execute the kernel as a pipeline of work items on FPGA. While, in a single-task kernel configuration, only a single work-item executes the kernel. Single-task configuration involves pipelined execution of loop iterations.

Algorithm 6: Convolution algorithm with data reorganization

Input : $input[X_i * Y_i * Z_i]$, $filter\_bias[Z_o]$,
$filter[Z_0 * Z_i * filter\_size * filter\_size]$, VEC_SIZE
Output: $out[X_o * Y_o * Z_o]$
Data: y', x', res

```
1  for z ← 0 to Z_o/VEC_SIZE do
2      for y ← 0 to Y_o do
3          for x ← 0 to X_o do
4              for z_1 ← 0 to Z_i/VEC_SIZE do
5                  for j ← 0 to filter_size do
6                      for i ← 0 to filter_size do
7                          for w ← 0 to VEC_SIZE do
8                              for v ← 0 to VEC_SIZE do
9                                  (y', x') ← map_to_input(y, x, j, i)
                                   res ← res + input[(z_1 × Y_i × X_i × VEC_SIZE) + (y' × X_i ×
                                   VEC_SIZE) + (x' × VEC_SIZE) + v)] × filter[z × Z_i ×
                                   filter_size × filter_size × VEC_SIZE) + (z_1 filter_size ×
                                   filter_size × VEC_SIZE × VEC_SIZE) + (j ×
                                   filter_size × VEC_SIZE × VEC_SIZE) + (i ×
                                   VEC_SIZE × VEC_SIZE) + (w × VEC_SIZE) + v]
10                             end
11                         end
12                     end
13                 end
14             end
15             for u ← 0 to VEC_SIZE do
16                 out[z × Y_i × X_i × VEC_SIZE) + (y × X_i × VEC_SIZE) + (x ×
                   VEC_SIZE) + u] ← res + filter_bias[z × VEC_SIZE + u]
17             end
18         end
19     end
20 end
```

In this work, we evaluate the impact of using both single-task and NDRange kernels.

Prefetching Memory prefetching is a technique of caching a load in local memory or registers prior to their usage. We prefetch the loop bounds (size of input, size of output, size of filters) by storing these values in a local integer variable.

Loop Unrolling On CPU and GPU, loop unrolling improves performance by reducing expensive branch instructions in execution. While, On FPGA, unrolled loops generate deeper pipelines to support multiple iterations of a loop [27]. In our implementation, we unroll the innermost loops of the convolution and deconvolution kernels by a factor of two (2) and innermost loops of the concatenation kernel by a factor of four (4).

Vectorization Vectorization executes SIMD instructions on arrays of data. Vector data types can improve the efficiency of the kernels by mitigating the bandwidth bottlenecks in the hardware [27]. As mentioned earlier, we use 16-way vectorization in our implementation for memory accesses and floating-point operations.

Low-precision Data Type Training and inference of deep neural networks with lower precision data types improve the performance by decreasing the memory and computational requirements [59], with a marginal decrease in the accuracy of trained network [12]. In our implementation, half-precision floats are used for storing feature maps and network parameters.

FPGA-specific Optimizations

Compute-unit Replication Replication of compute units improves the performance of kernels by increasing the computational bandwidth of the hardware [27, 28]. In our implementation, we use two compute units for convolution kernel.

4.2 Evaluation

We used Virginia Tech's Advanced Research Computing (ARC) Infer cluster, consisting of 18 compute nodes, for the compute-intensive training of Enhancement AI . Each node contains two Intel Xeon Gold 6130 CPUs and one Nvidia Tesla T4 GPU, coupled with 192 GB of system memory. Inference of Enhancement AI is evaluated on each of the following heterogeneous platforms:

- Many-core GPU, i.e., AMD Radeon Vega Frontier
- Multi-core CPU, i.e., Intel Xeon Gold 6128
- FPGA, i.e., Intel Arria 10 GX 1150

4.2.1 Training of Enhancement AI on a Multi-GPU System

Table 4.3 shows how our PyTorch implementation of Enhancement AI scales as the number of nodes increases. On a single node with a single Nvidia T4 GPU, the training for the Enhancement AI tool of ComputeCOVID19+ took approximately 15 hours.

The DistributedDataParallel container in Python parallelizes forward and backward propagation during AI training (since these processes are independent and load-balanced). Updating weights after forward and backward propagation requires synchronization at the end of every iteration. The speedup improves as the number of nodes increases but remains sub-linear due to the synchronization.

Increasing the batch size enables better utilization of the compute nodes, but it reduces the accuracy of the trained network. To date, the sensitivity of neural networks to batch size is not fully understood. Some explanations include (1) large batch-size training does not

converge to global minima; (2) large batch-size training tends to minimize the optimizer closer to the initial point; and (3) training samples in each batch interfere with each other's gradient [36].

4.2.2 Inference of Enhancement AI on Heterogeneous Platforms

The portability of OpenCL enables us to measure the inference runtime across a diverse set of platforms, as shown in Table 4.4. The best performance comes from the AMD Radeon Vega Frontier GPU, followed by the Intel Xeon Gold 6128 CPU and the Intel Arria 10 GX 1150 FPGA.

The breakdown of execution time for optimized kernels in DDnet is shown in Table 4.5. As shown in the table, the convolution operation is the most expensive operation in DDnet inference on each platform. This is because the number of floating-point operations required in convolution layers far exceeds the number of floating-point operations in deconvolution layers (there are 37 convolution layers and eight deconvolution layers).

Table 4.6 shows the runtime for inference using DDnet on HPC platforms with different

Table 4.3: Runtime for the enhancement AI training for 50 epochs

#Nodes*	Batch size	#Epochs	Training Runtime (hh:mm:ss)	MS-SSIM (Avg.)
1	1	50	15:14:46	98.71%
4	8	50	2:27:49	96.35%
4	8	100	4:58:52	96.30%
4	16	50	2:07:58	95.18%
8	8	50	2:21:49	95.46%
8	8	100	4:43:26	95.78%
8	32	50	1:17:25	92.04%
8	64	50	1:12:24	88.02%

* Each node has one NVIDIA® T4 GPU
(hh:mm::ss)=(hours:minutes:seconds)

optimizations, as described in §4.1.3. The impact of each optimization on performance and resource utilization on FPGA is discussed below.

Impact of Optimizations on Performance

Application-specific Optimizations Multiple threads executing in parallel in NDRange kernels repeatedly invalidate the cache lines due to recurring load and store to the same memory location, resulting in poor performance with naive deconvolution kernels. Refactoring the deconvolution kernel reduces the number of recurring loads and stores to the same global memory location. This improves the execution multi-folds ($\approx 10\times$ for CPU, $\approx 1000\times$ for GPU, $\approx 100\times$ for FPGA) on each platform with NDRange kernels. On the other hand, there is an abundance of on-chip memory, i.e., cache, available for executing thread in single-task kernel execution. Additionally, the write-back mechanism used for cache coherence in modern processors does not require the data to be written to global memory unless required by other processors. The abundance of on-chip memory and write-back cache coherence mechanism minimizes the execution time due to recurring loads and stores in the original single-task deconvolution kernel. However, the complex memory access in single-task refactored deconvolution kernels degrades the execution time for all three platforms.

Table 4.4: Runtime of the enhancement AI inference

Platform	**No. of Cores**	**Frequency**	**Runtime** (Seconds)
AMD Radeon™ Vega GFX 900	4096 stream processors	1600 MHz	0.05
Intel® Xeon® Gold 6128 CPU	24 CPU cores	3400 MHz	0.30
Intel® Arria 10 GX 1150 FPGA	2 compute units*	184 MHz	0.41

* Two compute units generated using __attribute__((num_compute_units(2)) which is a vendor-specific attribute

Table 4.5: Event-based time of the optimized OpenCL kernels for Enhancement AI inference. Execution time is reported in seconds

Platform	Convolution	Deconvolution	Other Kernels
CPU	0.14	0.09	0.07
GPU	0.027	0.009	0.004
FPGA	0.17	0.10	0.13

Table 4.6: Execution time profile of entire DDnet with different optimizations. Execution time is reported in seconds. REF: Refactoring, DR: Data Reorganization, PF: Prefetching, VEC: Vectorization, LU: Loop Unrolling, CUR: Compute Unit Replication, and HP: Half Precision Data

Kernels	CPU	GPU	FPGA
Single-task kernels	84.76	6550.26	10974.50
Single-task kernels + REF	390.99	7629.83	16875.57
Single-task kernels + REF + DR	14.52	825.90	384.58
NDRange kernels	6.51	219.60	636.30
NDRange + REF	1.95	0.25	69.89
NDRange + REF + DR	0.39	0.13	106.82
NDRange + REF + DR + PF	0.30	0.1	98.65
NDRange + REF + DR + PF + VEC	1.01	0.08	0.76
NDRange + REF + DR + PF + VEC + UNR	1.02	0.12	0.55
NDRange + REF + DR + PF + VEC + CUR	-	-	0.61
NDRange + REF + DR + PF + VEC + HP	1.11	0.048	0.66
NDRange + REF + DR + PF + VEC + UNR + HP	1.12	0.05	0.41
NDRange + REF + DR + PF + VEC + CUR + HP	-	-	0.48

OpenCL kernels with data reorganization reduce the independent memory accesses by maximizing memory coalescing. This reduces the bandwidth requirement for convolution, deconvolution, pooling, and un-pooling kernels (kernels where memory accesses were not linear, i.e., irregular). We see multi-folds speed up in CPU and GPU execution with data reorganization in both single task and NDRange kernels. However, data reorganization requires additional nested loops for consistent execution. Because of complex hardware generated

for nested loops (nested pipelines) in FPGA, the data reorganization degrades the performance of NDRange kernels on FPGA. The generation of FPGA's reconfigurable hardware for single task kernels with multiple nested loops is comparatively simpler because of relaxed constraints on the amount of memory required. Hence, data reorganization improves the performance of single task kernels by a factor of ≈ 40.

Architecture-aware Optimizations NDRange kernels exploit the massive parallelization available in the execution of each kernel in DDnet to improve the performance on each heterogeneous platform. The speedups with NDRange kernels follow the trend of parallel processing units/logic available on each platform. GPU with 4096 streaming processors gains the highest speedup, followed by FPGA and CPU ($\approx 30\times$, $\approx 17\times$ and $\approx 13\times$ speedups on GPU, FPGA and GPU respectively with the naive single task and NDRange kernels).

Memory prefetching relatively few variables (size of the input, output, and parameters arrays in each kernel) marginally improves the performance on each computing platform.

Vectorizing each kernel marginally improves the performance on GPU. This is because the SIMD compute units exploit inter-thread vectorization in the executing kernel. This leaves a little margin for improvement with manual vectorization, i.e., intra-thread vectorization, since the compute units and memory bandwidth are maximally utilized by inter-thread vectorization. We get $\approx 130\times$ via manual vectorization on FPGA. This is due to the generation of extra hardware for handling vectored floating-point operations and coalesced memory accesses due to vectored global memory accesses. On CPU, which does not have SIMD compute units or reconfigurable logic, vectorized kernels increase execution complexity and register pressure (i.e., the count of registers required to store local variables), which degrade the performance by a factor of ≈ 3.3.

Manual vectorization of kernels increases the resource utilization on FPGA and register

pressure on CPU and GPU. The hardware constraints on FPGA leave little scope for loop unrolling. Therefore we unroll the innermost loop of convolution, deconvolution, and concatenation kernel (most expensive kernels in DDnet) by a factor of two. This generates deeper pipelines on FPGA, which speeds up the execution by a factor of ≈ 1.4. However, due to the increased register pressure on CPU and GPU, vectorization and loop unrolling implemented simultaneously marginally degrade the performance of kernels.

The implicit APIs defined in OpenCL for reading half-precision floating-point data from global memory converts half-precision floating-point data to single-precision floating-point data after the global memory is read. We use this implementation in our OpenCL kernels with no additional type casting to reduce the complexity. Therefore, the implementation with half-precision floating-point data reduces only the memory bandwidth requirement, but the computational requirement of OpenCL kernels remains the same. This improves GPU and FPGA implementation performance by a factor of $\approx 1.7\times$ and $\approx 1.3\times$ respectively. On CPU, the execution time remains almost the same with half-precision floating data because the performance on this platform is not limited by memory bandwidth bottleneck.

FPGA-specific Optimizations The compiler generates wider and deeper pipelines with loop unrolled and vectorized kernels. These optimizations are expensive in terms of resource utilization. Therefore, simultaneous application of these optimizations leaves no room for compute unit replication. To apply compute unit replication to the most expensive kernel, i.e., convolution, we reduced the loop unrolling factor in the concatenation kernel and replicated the convolution kernel by a factor of two (2). This improves the execution time of convolution operation by 38% (0.17 seconds to 0.10 seconds with half-precision floating-point data) but degrades the performance of concatenation kernel by more than 100%(0.98 seconds to 0.20 seconds with half-precision floating-point data). This degrades the overall

Table 4.7: Resource utilization with multiple combinations of optimization on FPGA REF: Refactoring, DR: Data Reorganization, PF: Prefetching, VEC: Vectorization, LU: Loop Unrolling, CUR: Compute Unit Replication, and HP: Half Precision Data

Kernels	**ALM Utilization (%)**	**RAM Utilization (%)**	**DSP Utilization (%)**	**Frequency (MHz)**
Single-task kernels	35	15	3	222
Single-task kernels + REF	34	12	3	218
Single-task kernels + REF + DR	46	18	2	235
NDRange kernels	33	21	3	218
NDRange + REF	32	17	3	275
NDRange + REF + DR	37	27	3	223
NDRange + REF + DR + PF	36	27	2	231
NDRange + REF + DR + PF + VEC	40	32	16	236
NDRange + REF + DR + PF + VEC + UNR	46	56	2	191
NDRange + REF + DR + PF + VEC + CUR	51	73	36	153
NDRange + REF + DR + PF + VEC + HP	40	28	16	265
NDRange + REF + DR + PF + VEC + UNR + HP	50	48	26	216
NDRange + REF + DR + PF + VEC + CUR + HP	54	66	36	190

performance of DDnet inference on FPGA.

Impact of Optimizations on FPGA Resource Utilization

Table 4.7 summarizes the resources consumed by all kernels combined with corresponding optimizations. The impact of each optimization applied to FPGA implementation on resource utilization is discussed below.

Application-specific Optimizations Refactorizing deconvolution kernel marginally reduces the logic and RAM utilization in both single-task and NDRange kernels because of

simpler logic and fewer local variables used to store intermediate results.

Data reorganization requires additional nested loops and arrays to store intermediate results for consistent execution of kernels. This increases the logic utilization for implementing nested loops and RAM utilization for storing results in arrays in FPGA.

Architecture-aware Optimizations Logic utilization is slightly lower with NDRange kernels than single-task kernels because of the fewer iterative loops and input-dependent branches in the former. However, memory utilization increases with NDRange kernels since each work item requires independent memory for storing local variables.

There is no significant impact on resource utilization with the implementation of prefetching since only a few variables were prefetched from the global memory.

The increased ALM (Adaptive Logic Module), DSP, and RAM utilization with the vectorized kernel is attributed to the generation of SIMD compute units required for carrying out vectored floating-point operations and vectored global memory accesses. SIMD compute units significantly increases the DSP utilization (by 14%) and marginally increases the logic and RAM utilization (by 4% and 5% respectively)

Loop unrolling generates deeper pipelines by replicating the hardware for unrolled loops. This increases the ALM, DSP, and RAM utilization by 5%, 14%, and 10%, respectively, on FPGA.

As discussed in §4.2.2, the implementation OpenCL kernels with half-precision floating data does not impact computational requirement. Therefore, there is no significant change in ALM and DSP utilization. The RAM utilization decreases by a small margin because of smaller buffers used for storing half-precision floating-point data read from global memory.

FPGA-specific Optimizations There is a significant increase in resource utilization with replicated convolution kernel. ALM, DSP, and RAM utilization on FPGA increase by a factor of 7%, 15%, and 10%, respectively.

The absolute count of resources utilized on FPGA with most optimal implementation are presented in Table 4.8

Table 4.8: Absolute resource utilization of most optimal of implementation on FPGA REF: Refactoring, DR: Data Reorganization, PF: Prefetching, VEC: Vectorization, LU: Loop Unrolling, CUR: Compute Unit Replication, and HP: Half Precision Data

Kernels	**#ALM Utilized/ #ALM Available**	**#RAM Utilized/ #RAM Available**	**#DSP Utilized/ #DSP Available**
NDRange + REF + DR + PF + VEC + HP	469,283 / 933,120	5,655 / 11,721	1,503 / 5,760

4.2.3 Power Consumption on Heterogeneous Platforms

Along with performance efficiency, power consumed by computing platforms is an essential specification for selecting a platform to deploy in the real world, especially in embedded and real-time applications. Therefore, we measure the power consumed by DDnet inference on each heterogeneous platform and evaluate their efficiency in terms of the following metrics.

- Dynamic power consumed in Watts
- Total energy consumed per DDnet inference
- Performance per Watt i.e., FLOPS per Watt
- Energy efficiency in terms of energy-delay (ED) product

Since each computing platform is embedded in compact and functional servers, it was difficult to measure the static power consumption of individual computing platforms alone.

Table 4.9: Power consumption and power efficiency evaluation of DDnet inference on heterogeneous platforms

Platform	Idle Power (W)	Running Power (W)	Dynamic Power (W)	Execution Time (seconds)	Energy Consumed (KJ)	FLOPS per Watt (10^8)	ED Product (W.second^2)
CPU	255.1	304.5	49.4	30.8	1.52	0.25	46862.816
GPU	215.4	256.6	41.2	5.4	0.22	1.7	1001.16
FPGA	254.8	262.3	7.5	41.9	0.31	1.2	13167.08

Therefore, we measure the total power consumed by the server and estimate the dynamic power consumed by individual platforms by measuring the power consumption in idle and running states. For measuring the power consumed on computing platforms, we use Watts Up PRO Digital Electric Meter. The power meter logs one reading per second and has an accuracy of ±3%. Since the runtime of one inference using DDNet is less than one second on each platform, we measure the power consumption with 100 sequential input images. Table 4.9 shows power metrics described above measured on CPU, GPU, and FPGA.

FPGA operating at the lowest frequency consumes minimum dynamic power as compared to CPU and GPU. In terms of performance per Watt, FPGA achieves comparable energy efficiency with respect to GPU, and outperforms CPU by a factor of 4.8. In terms of energy consumption, GPU consumes the lowest energy, even though the power consumed by GPU is $\approx 5.5\times$ higher than the power consumed by FPGA. This is because the execution of DDnet inference is approximately an order of magnitude faster on GPU than CPU and FPGA. GPU also achieves the highest energy efficiency in terms of energy delay (ED) product.

Low power consumption is a desirable characteristic and sometimes a requirement in embedded applications. Specifically, in medical diagnosis, low-power computing devices, such as FPGAs, can benefit computed tomography applications in telemedicine, where the CT scanner machines are fitted in constrained spaces. On the other hand, GPUs are the appropriate choice of computing platforms in hospitals and clinics, where more focus is on speed

than power consumption. The implications of power and energy efficiency on deploying deep learning algorithms for computed tomography are discussed in §5.2.

Chapter 5

Practical Considerations of using AI for Biomedical Diagnosis

In the past decade, machine learning has gained tremendous popularity in every engineering domain, including but not limited to computer vision, robotics, healthcare, internet-of-things, and social media. In particular, machine learning in healthcare has the potential to enhance biomedical practices.

Machine learning algorithms in biomedical science has been extensively researched and developed for making predictions from the input data for medical diagnosis using neural networks. These algorithms use data from magnetic resonance imaging [30], retinal scan [51, 65], endoscopic images [5], computed tomography [20, 22, 41, 68, 73, 76, 90], electrocardiography [79] etc. Another category of machine learning-based algorithms aid clinicians by highlighting or extracting more meaningful features in the input data. These include image segmentation [69], image enhancement [88], super resolution [74] lesion segmentation [92] etc.

These algorithms, however efficient in theory, pose applicability questions on real-world data. Validation of these algorithms is a massive challenge for practical applications. The acceptance of results from these algorithms without human intervention is questionable, given high stakes in biomedical applications. Another obstacle is the availability of appropriate hardware in clinics for deploying these compute-intensive deep learning algorithms. This chapter addresses these challenges and comments on the applicability of image enhancement

in computed tomography.

5.1 Verification of Machine Learning Algorithms

An optimally trained machine learning algorithm works as a black box during inference. The mathematical functional mapping from input to output of a deep neural network is unknown to the user as well as to the developer. Furthermore, the size of the input and network complexity makes it impossible to formally verify machine learning algorithms. Therefore, the successful deployment of machine learning algorithms requires verification on a large dataset that is representative of real-world data. However, the limited amount of data for testing DL algorithms is often a challenge in their thorough verification [61, 70].

For biomedical applications, where accuracy correctness and accuracy are most consequential, generating erroneous results can have adverse implications. For example, a misdiagnosis of a COVID-19 patient can exacerbate illness from delays in the correct treatment [55]. Misdiagnosed patients also unwittingly serve as SARS-CoV-2 carriers and transmitters.

Keeping this in mind, we test and validate Enhancement AI and ComputeCOVID19+ across multiple data sources, described in Table 2.5. These data sources contain chest CT scans, each comprising of more than 100 image slices, of healthy and COVID-19 patients, generating enough variability inputs for training and testing Enhancement AI and ComputeCOVID19+ framework. High performance in terms of accuracy metrics for each neural network shows that the networks can generalize well to the variations in CT images across different datasets. These metrics also imply that the network and framework will likely perform well with the unseen real-world data.

In addition, we also propose an alternative workflow for ComputeCOVID19+ in which a ra-

diologist or a clinician replaces the Analysis AI, as shown in Figure 2.7. The software is packaged in individual modules, that is Enhancement AI and Analysis AI, which can be separately downloaded and installed from GitHub. This alternative workflow aids radiologists by enhancing the quality of CT images for better interpretation and ensures that the radiologists or clinicians are responsible for the decision-making of medical diagnosis.

Shadow Deployment Shadow deployment is an effective way of testing machine learning algorithms with real-world data before actual deployment. A shadow deployment is a process of running production data through the model under test. The results from the model under test do not serve customers but are used for its verification [40].

Shadow deployment of ComputeCOVID19+ can have two-folds advantages: (1) The model will be verified with real-world data with the help of radiologists and clinicians, (2) The results from the framework will aid radiologists and clinicians in decision-making,

5.2 Deploying AI for Computed Tomography in Clinics

Artificial intelligence and machine learning have reformed healthcare and biomedical techniques by aiding clinicians and radiologists, and providing expedited diagnosis to patients in the last decade. However, in practice, the use of computer-aided diagnosis (CAD) has not been routinely adopted by radiologists and clinicians [75]. This is because the development of healthcare IT infrastructure is substantially slower than the advances in research and development of CAD [24, 58]. Sparsity in the availability of high-performance computers in clinics is a huge bottleneck for radiologists and clinicians to adopt CAD methods.

To that end, we propose two strategies for deploying our deep learning algorithms for computed tomography, as explained below.

- Post-CT Processing: Training machine learning algorithms are comparatively more computationally expensive than inference. Therefore, we train our networks on supercomputers with multiple GPUs. These trained models can run even on a single-core CPU or GPU for real-time inference. These tools, i.e., our trained networks, can be installed via a simple software update and provide inference results with just one click. This provides seamless integration with the preexisting CT software tools and our trained neural networks.

- During-CT Processing: Image enhancement can be accomplished during the acquisition of X-ray projections by installing a processor inside the CT scanner machines. Having an embedded processor can reform telemedicine where CT machines are fitted in constrained spaces.

Considering the power consumption of CT scanner machines in the order of 10 to 100 kW (10 kW idle power and 100 kW peak power) [14, 23], it is favorable to embed low power computing device in the CT scanner machine. From Table 4.9, it is evident that FPGAs, with low power requirement, are appropriate processors for such applications. The reconfigurability of logic in FPGA also makes it easier to update software with time.

Chapter 6

Conclusions and Future Work

In this chapter, we present our work's conclusion and provide some perspectives of the future work that can be built upon this thesis.

6.1 Conclusion

In this work, we researched and developed deep learning based high-fidelity image enhancement algorithms and software for computed tomography applications. Specifically, we used DenseNet and Deconvolution network (DDnet) to enhance 2D images and 3D volumes. To validate the applicability of image enhancement in medical diagnosis and evaluate its advantages, we developed a CT-based framework, called ComputeCOVID19+, for COVID-19 diagnosis and monitoring. ComputeCOVID19+ contains novel algorithms and software for high-quality CT image construction and high-precision classification of COVID-19 CT scans. Our ComputeCOVID19+ can speed up the COVID-19 inference time from hours to minutes, while at the same time improving the diagnostic accuracy from 86% to 91%.

Furthermore, to reduce the turnaround time for training and testing AI models, we implement and accelerate the complex deep-learning algorithms across a multitude of heterogeneous platforms, including multi-core CPU, many-core GPU, and even FPGA, using OpenCL. Our OpenCL implementation is at least 2× faster than the analogous PyTorch implementation and can execute on multi-core CPU, many-core GPU and FPGA within a

fraction of a second. We also evaluate each platform in terms of power efficiency.

Finally, we present some practical considerations for deploying AI-based framework for medical diagnosis. These considerations include validation of AI models on clinical data, which is representative of real world data, and the availability of computing platform for implementing trained AI models in clinics.

6.2 Future Work

In this section, we discuss some future work and limitations of this work.

Improving the Accuracy of Image Enhancement Our Enhancement AI tool only leverages data from the image domain, which limits the extent to which the quality of image and accuracy of CT-based COVID-19 diagnosis can be improved ($\approx 5\%$ accuracy improvement in this work). Therefore, as part of future work, we seek to address this limitation by using data available from the projection domain and combining it with knowledge from medical imaging physics to reconstruct even higher-quality CT images. We intend to use super resolution and deblur-based iterative reconstruction (SADIR) algorithm [89] for this work. SADIR outperforms both iterative CT image construction and deep learning based CT image enhancement, and we expect to improve the classification accuracy of ComputeCOVID19+ framework by 2-3% using SADIR for CT image enhancement.

Optimizing DDnet's Architecture Our work on image enhancement extends the work by Zhang et al. in [88] on sparse view reconstruction of CT images. Therefore we use the architecture of DDnet as described in the research paper. Our work explores the optimization space of hyper-parameters, such as learning rate, batch size, etc. In addition to this,

some architectural parameters, such as the number of dense blocks, the growth rate in the dense block, number of convolution layers in each dense block, the size of convolution, and deconvolution filters, can be optimized to further improve the functional mapping from input to output. The exploration of these architectural parameters remains a part of future study.

Automating Hyper-parameter Tuning for Optimizing Network Training In this work, we tuned hyper-parameters of DDnet using perturb and observe method. This method is laborious and sometimes generate sub-optimal trained network. In future, we intend to use iterative machine learning (IterML) [13] for tuning the hyper-parameters of DDnet. IterML statistically prunes and optimizes the hyper-parameter search space to achieve better performance. Using IterML, we expect to reduce the time required for hyper-parameter tuning, as well as improve the efficiency of trained network.

Exploring Sparsity in Deep Neural Network Hoefler et al., in [25], show that the sparsity in a deep neural network leads to efficient inference and training with an acceptable compromise in the accuracy of the network. By carefully pruning convolution and deconvolution layers in DDnet, we seek to reduce the computational and memory bandwidth requirement of training and inference of the network. This will improve execution efficiency and reduce turnaround time.

Evaluating the Performance of Enhancement AI on Embedded CPU and Embedded GPU In this project, we used a high performance CPU and GPU for evaluating the performance of Enhancement AI inference in terms of speed and power efficiency. These computing devices deliver high performance in terms of speed at the expense of high power consumption. For applications, such as mobile Computed Tomography, where power consumption and the price of hardware are major concerns, embedded CPU or GPU are preferred

choice of computing hardware over a high performance CPU or GPU. Therefore, we plan to evaluate the performance of Enhancement AI inference on embedded CPU and GPU, and compare them against FPGA's performance.

Evaluating Framework with Real-world Low-dose CT Data As a subject of future study, we plan to evaluate the framework with low-dose CT image data. Low-dose CT technology benefits from reduced risk of cancer, but there is an associated loss in the quality of CT images. Analyzing the accuracy of diagnosis with such low-quality images would be an ideal stress test for our framework.

Evaluating the Applicability of CT Image Enhancement for Medical Diagnosis With the help of radiologists and clinicians, we intend to analyze the feasibility of deploying Enhancement AI and ComputeCOVID19+ in CT scanner machines for diagnosis of COVID-19 and other maladies, such as viral pneumonia and cancer. We also want to evaluate ComputeCOVID19+ framework by replacing Analysis AI with a radiologist/clinician for COVID-19 diagnosis. This will validate the efficacy of Enhancement AI for biomedical imaging and medical diagnosis.

Bibliography

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.

[2] Tao Ai, Zhenlu Yang, Hongyan Hou, Chenao Zhan, Chong Chen, Wenzhi Lv, Qian Tao, Ziyong Sun, and Liming Xia. Correlation of chest CT and RT-PCR testing for coronavirus disease 2019 (covid-19) in china: a report of 1014 cases. *Radiology*, 296(2): E32–E40, 2020.

[3] Mohammed Alawad. Scalable FPGA accelerator for deep convolutional neural networks with stochastic streaming. *IEEE Transactions on Multi-Scale Computing Systems*, 4(4): 888–899, 2018.

[4] Marcel Beister, Daniel Kolditz, and Willi A Kalender. Iterative Reconstruction Methods in X-ray CT. *Physica Medica*, 28(2):94–108, 2012.

[5] Mustain Billah and Sajjad Waheed. Gastrointestinal polyp detection in endoscopic images using an improved feature extraction method. *Biomedical engineering letters*, 8 (1):69–75, 2018.

[6] David J Brenner and Eric J Hall. Computed tomography—an increasing source of radiation exposure. *New England Journal of Medicine*, 357(22):2277–2284, 2007.

[7] Jung-Woo Chang, Keon-Woo Kang, and Suk-Ju Kang. An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(1):281–295, 2018.

[8] 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.

[9] Yuhua Chen, Yibin Xie, Zhengwei Zhou, Feng Shi, Anthony G Christodoulou, and Debiao Li. Brain MRI super resolution using 3D deep densely connected neural networks. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 739–742. IEEE, 2018.

[10] Lishui Cheng, Sangtae Ahn, Steven G Ross, Hua Qian, and Bruno De Man. Accelerated iterative image reconstruction using a deep learning based leapfrogging strategy. In *International conference on fully three-dimensional image reconstruction in radiology and nuclear medicine*, pages 715–720, 2017.

[11] Sarah Cherian, Varsha Potdar, Santosh Jadhav, Pragya Yadav, Nivedita Gupta, Mousmi Das, Partha Rakshit, Sujeet Singh, Priya Abraham, Samiran Panda, et al. Convergent evolution of SARS-CoV-2 spike mutations, L452R, E484Q and P681R, in the second wave of COVID-19 in Maharashtra, India. *bioRxiv*, 2021.

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[13] Xuewen Cui and Wu-chun Feng. Iterative machine learning (iterml) for effective parameter pruning and tuning in accelerators. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 16–23, 2019.

[14] Mohammad Amin Esmaeili, Ashkan Jahromi, Janet Twomey, Bayram Yildirim, Michael Overcash, Tyson Elsken, Fernando Dominquez, Nicholas Thomas, and Ashlee Mcadam. Energy consumption of VA hospital CT scans. In *Proceedings of the 2011 IEEE international symposium on sustainable systems and technology*, pages 1–5. IEEE, 2011.

[15] Facebook. GLOO Communication Backend. URL https://github.com/facebookincubator/gloo/.

[16] Yicheng Fang, Huangqi Zhang, Jicheng Xie, Minjie Lin, Lingjun Ying, Peipei Pang, and Wenbin Ji. Sensitivity of chest CT for COVID-19: comparison to RT-PCR. *Radiology*, 296(2):E115–E117, 2020.

[17] Hester A Gietema, Noortje Zelis, J Martijn Nobel, Lars JG Lambriks, Lieke B van Alphen, Astrid ML Oude Lashof, Joachim E Wildberger, Irene C Nelissen, and Patricia M Stassen. CT in Relation to RT-PCR in Diagnosing COVID-19 in The Netherlands: A Prospective Study. *PLOS One*, 15(7):e0235844, 2020.

[18] Wei-jie Guan, Zheng-yi Ni, Yu Hu, Wen-hua Liang, Chun-quan Ou, Jian-xing He, Lei Liu, Hong Shan, Chun-liang Lei, David SC Hui, et al. Clinical characteristics of coronavirus disease 2019 in China. *New England journal of medicine*, 382(18):1708–1720, 2020.

[19] Yo Seob Han, Jaejun Yoo, and Jong Chul Ye. Deep Residual Learning for Compressed Sensing CT Reconstruction via Persistent Homology Analysis. *arXiv preprint arXiv:1611.06391*, 2016.

[20] Stephanie A. Harmon, Thomas H. Sanford, Sheng Xu, Evrim B. Turkbey, Holger Roth, Ziyue Xu, Dong Yang, Andriy Myronenko, Victoria Anderson, Amel Amalou, Maxime Blain, Michael Kassin, Dilara Long, Nicole Varble, Stephanie M. Walker, Ulas Bagci, Anna Maria Ierardi, Elvira Stellato, Guido Giovanni Plensich, Giuseppe Franceschelli, Cristiano Girlando, Giovanni Irmici, Dominic Labella, Dima Hammoud, Ashkan Malayeri, Elizabeth Jones, Ronald M. Summers, Peter L. Choyke, Daguang Xu, Mona Flores, Kaku Tamura, Hirofumi Obinata, Hitoshi Mori, Francesca Patella, Maurizio Cariati,

Gianpaolo Carrafiello, Peng An, Bradford J. Wood, and Baris Turkbey. Artificial Intelligence for the Detection of COVID-19 Pneumonia on Chest CT using Multinational Datasets. *Nature Communications*, 11(1):1–7, Dec. 2020.

[21] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *IEEE Conf. on Computer Vision & Pattern Recognition*, pages 9729–9738, 2020.

[22] Xuehai He, Xingyi Yang, Shanghang Zhang, Jinyu Zhao, Yichen Zhang, Eric Xing, and Pengtao Xie. Sample-Efficient Deep Learning for COVID-19 Diagnosis Based on CT Scans. *MedRxiv*, 2020.

[23] Tobias Heye, Roland Knoerl, Thomas Wehrle, Daniel Mangold, Alessandro Cerminara, Michael Loser, Martin Plumeyer, Markus Degen, Rahel Lüthy, Dominique Brodbeck, et al. The energy consumption of radiology: energy-and cost-saving opportunities for CT and MRI operation. *Radiology*, 295(3):593–605, 2020.

[24] Neset Hikmet, Anol Bhattacherjee, Nir Menachemi, Varol O Kayhan, and Robert G Brooks. The role of organizational factors in the adoption of healthcare information technology in Florida hospitals. *Health care management science*, 11(1):1–9, 2008.

[25] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[26] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[27] *Intel FPGA SDK for OpenCL Pro Edition: Best Practices Guide.* Intel, .

URL https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807516407.html.

[28] *Intel FPGA SDK for OpenCL Pro Edition: Programming Guide*. Intel, . URL https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[30] Sajid Iqbal, M Usman Ghani Khan, Tanzila Saba, and Amjad Rehman. Computer-assisted brain tumor type discrimination using magnetic resonance imaging features. *Biomedical Engineering Letters*, 8(1):5–28, 2018.

[31] Kyong Hwan Jin, Michael T McCann, Emmanuel Froustey, and Michael Unser. Deep Convolutional Neural Network for Inverse Problems in Imaging. *IEEE Transactions on Image Processing*, 26(9):4509–4522, 2017.

[32] Michael A Johansson, Talia M Quandelacy, Sarah Kada, Pragati Venkata Prasad, Molly Steele, John T Brooks, Rachel B Slayton, Matthew Biggerstaff, and Jay C Butler. SARS-CoV-2 Transmission from People Without COVID-19 Symptoms. *J. American Medical Assoc.*, Jan. 2021.

[33] Johns Hopkins Coronavirus Resource Center. COVID-19 Map, 2020. URL https://coronavirus.jhu.edu/map.html.

[34] Konstantinos Kamnitsas, Christian Ledig, Virginia FJ Newcombe, Joanna P Simpson, Andrew D Kane, David K Menon, Daniel Rueckert, and Ben Glocker. Efficient multi-

scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78, 2017.

[35] Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. CNN-based segmentation of medical imaging data. *arXiv preprint arXiv:1701.03056*, 2017.

[36] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[37] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Lauren M Kucirka, Stephen A Lauer, Oliver Laeyendecker, Denali Boon, and Justin Lessler. Variation in False-Negative Rate of Reverse Transcriptase Polymerase Chain Reaction-based SARS-CoV-2 Tests by Time Since Exposure. *Annals of Internal Medicine*, 173(4):262–267, 2020.

[39] Daniel B Larremore, Bryan Wilder, Evan Lester, Soraya Shehata, James M Burke, James A Hay, Milind Tambe, Michael J Mina, and Roy Parker. Test Sensitivity is Secondary to Frequency and Turnaround Time for COVID-19 Screening. *Science Advances*, 7(1):eabd5393, 2021.

[40] Alexander Lavin, Ciarán M Gilligan-Lee, Alessya Visnjic, Siddha Ganju, Dava Newman, Sujoy Ganguly, Danny Lange, Atılım Güneş Baydin, Amit Sharma, Adam Gibson, et al. Technology readiness levels for machine learning systems. *arXiv preprint arXiv:2101.03989*, 2021.

[41] Lin Li, Lixin Qin, Zeguo Xu, Youbing Yin, Xin Wang, Bin Kong, Junjie Bai, Yi Lu, Zhenghan Fang, and Qi Song. Using Artificial Intelligence to Detect COVID-19 and

Community-Acquired Pneumonia Based on Pulmonary CT: Evaluation of the Diagnostic Accuracy. *Radiology*, 296(2):E65–E71, 2020.

[42] Meng Li, Shiwen Shen, Wen Gao, William Hsu, and Jason Cong. Computed tomography image enhancement using 3D convolutional neural network. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 291–299. Springer, 2018.

[43] Si Li, Qing Cao, Yang Chen, Yining Hu, Limin Luo, and Christine Toumoulin. Dictionary Learning Based Sinogram Inpainting for CT Sparse Reconstruction. *Optik*, 125 (12):2862–2867, 2014.

[44] Bing Liu, Danyin Zou, Lei Feng, Shou Feng, Ping Fu, and Junbao Li. An FPGA-based cnn accelerator integrating depthwise separable convolution. *Electronics*, 8(3):281, 2019.

[45] Shuanglong Liu, Hongxiang Fan, Xinyu Niu, Ho-cheung Ng, Yang Chu, and Wayne Luk. Optimizing CNN-based segmentation with deeply customized convolutional and deconvolutional architectures on FPGA. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(3):1–22, 2018.

[46] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[47] Lung Image Database Consortium Image Collection. LIDC-IDRI - The Cancer Imaging Archive (TCIA) Public Access – Cancer Imaging Archive Wiki. URL https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI.

[48] Cheng Luo, Man-Kit Sit, Hongxiang Fan, Shuanglong Liu, Wayne Luk, and Ce Guo.

Towards efficient deep neural network training by FPGA-based batch-level parallelism. *Journal of Semiconductors*, 41(2):022403, 2020.

[49] Halgurd S Maghdid, Aras T Asaad, Kayhan Zrar Ghafoor, Ali Safaa Sadiq, and Muhammad Khurram Khan. Diagnosing COVID-19 pneumonia from X-ray and CT images using deep learning and transfer learning algorithms. *arXiv preprint arXiv:2004.00038*, 2020.

[50] Kingshuk Majumder and Uday Bondhugula. A flexible FPGA accelerator for convolutional neural networks. *arXiv preprint arXiv:1912.07284*, 2019.

[51] Romany F Mansour. Deep-learning-based automatic computer-aided diagnosis system for diabetic retinopathy. *Biomedical engineering letters*, 8(1):41–57, 2018.

[52] McCollough, C.H., Chen, B., Holmes, D., III, Duan, X., Yu, Z., Yu, L., Leng, S., Fletcher, J. (2020). Data from Low Dose CT Image and Projection Data [Data set]. URL https://doi.org/10.7937/9npb-2637.

[53] Medical Imaging & Data Resource Ctr. MIDRC. URL https://www.midrc.org/.

[54] Medical Imaging Databank of the Valencia Region. BIMCV-COVID19 – BIMCV. URL https://bimcv.cipf.es/bimcv-projects/bimcv-covid19/.

[55] Jill C Muhrer. Risk of misdiagnosis and delayed diagnosis with COVID-19: A Syndemic Approach. *The Nurse Practitioner*, 46(2):44, 2021.

[56] Dominik Müller, Iñaki Soto Rey, and Frank Kramer. Automated Chest CT Image Segmentation of COVID-19 Lung Infection based on 3D U-Net. *arXiv preprint arXiv:2007.04774*, 2020.

[57] A. Munshi. The OpenCL Specification. In *IEEE Hot Chips 21 Symp.*, pages 1–314, 2009.

[58] Arsalan Nadeem and Ryan Cantrell. The importance of it infrastructure in the healthcare industry, Dec 2020. URL https://www.senderoconsulting.com/blogs/the-importance-of-it-infrastructure-in-the-healthcare-industry/.

[59] Nvidia. Mixed-Precision Programming with CUDA 8, 2016. URL https://developer.nvidia.com/blog/mixed-precision-programming-cuda-8/.

[60] NVIDIA GPU Cloud (NGC). Clara Train SDK. URL https://ngc.nvidia.com/catalog/containers/nvidia:clara-train-sdk.

[61] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D Lawrence. Challenges in deploying machine learning: a survey of case studies. *arXiv preprint arXiv:2011.09926*, 2020.

[62] R. Plater. As Many as 80 Percent of People with COVID-19 Aren't Aware They Have the Virus. *Healthline*, May 2020.

[63] PyTorch. Distributed Data Parallel, 2021. URL https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html.

[64] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 26–35, 2016.

[65] D Relan and R Relan. Multiscale self-quotient filtering for an improved unsupervised retinal blood vessels characterisation. *Biomedical engineering letters*, 8(1):59–68, 2018.

[66] M. Roser, H. Ritchie, E. Ortiz-Ospina, and J. Hasell. Coronavirus Pandemic (COVID-19). *Our World in Data*, 2020. https://ourworldindata.org/coronavirus.

[67] R Schofield, L King, U Tayal, I Castellano, J Stirrup, F Pontana, James Earls, and E Nicol. Image Reconstruction: Part 1 – Understanding Filtered Back Projection,

Noise and Image Acquisition. *Journal of Cardiovascular Computed Tomography*, 14(3): 219–225, 2020.

[68] Sertan Serte and Hasan Demirel. Deep Learning for Diagnosis of COVID-19 using 3D CT Scans. *Computers in Biology and Medicine*, page 104306, 2021.

[69] Neeraj Sharma and Lalit M Aggarwal. Automated medical image segmentation techniques. *Journal of medical physics/Association of Medical Physicists of India*, 35(1):3, 2010.

[70] Lin Shen, Benjamin H. Kann, R. Andrew Taylor, and Dennis L. Shung. The clinician's guide to the machine learning galaxy. *Frontiers in Physiology*, 12:414, 2021. ISSN 1664-042X. doi: 10.3389/fphys.2021.658583. URL https://www.frontiersin.org/article/10.3389/fphys.2021.658583.

[71] Robert L Siddon. Fast Calculation of the Exact Radiological Path for a Three-Dimensional CT Array. *Medical Physics*, 12(2):252–255, 1985.

[72] Roman A Solovyev, Alexandr A Kalinin, Alexander G Kustov, Dmitry V Telpukhov, and Vladimir S Ruhlov. FPGA implementation of convolutional neural networks with fixed-point calculations. *arXiv preprint arXiv:1808.09945*, 2018.

[73] Ying Song, Shuangjia Zheng, Liang Li, Xiang Zhang, Xiaodong Zhang, Ziwang Huang, Jianwen Chen, Huiying Zhao, Yusheng Jie, and Ruixuan Wang. Deep Learning Enables Accurate Diagnosis of Novel Coronavirus (COVID-19) with CT Images. *MedRxiv*, 2020.

[74] Dinh-Hoan Trinh, Marie Luong, Francoise Dibos, Jean-Marie Rocchisani, Canh-Duong Pham, and Truong Q Nguyen. Novel example-based method for super-resolution and denoising of medical images. *IEEE Transactions on Image processing*, 23(4):1882–1895, 2014.

[75] Bram Van Ginneken, Cornelia M Schaefer-Prokop, and Mathias Prokop. Computer-aided diagnosis: how to move from the laboratory to the clinic. *Radiology*, 261(3): 719–732, 2011.

[76] Shuai Wang, Bo Kang, Jinlu Ma, Xianjun Zeng, Mingming Xiao, Jia Guo, Mengjiao Cai, Jingyi Yang, Yaodong Li, and Xiangfei Meng. A Deep Learning Algorithm Using CT Images to Screen for Corona Virus Disease (COVID-19). *MedRxiv*, 2020.

[77] Wenling Wang, Yanli Xu, Ruqin Gao, Roujian Lu, Kai Han, Guizhen Wu, and Wenjie Tan. Detection of SARS-CoV-2 in Different Types of Clinical Specimens. *J. American Medical Assoc.*, 323(18):1843–1844, 2020.

[78] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image Quality Assessment: from Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[79] Ran Wei, Xinghua Zhang, Jinhai Wang, and Xin Dang. The research of sleep staging based on single-lead electrocardiogram and deep neural network. *Biomedical engineering letters*, 8(1):87–93, 2018.

[80] Ning Wu, Tao Jiang, Lei Zhang, Fang Zhou, and Fen Ge. A reconfigurable convolutional neural network-accelerated coprocessor based on RISC-V instruction set. *Electronics*, 9 (6):1005, 2020.

[81] Tobias Würfl, Florin C Ghesu, Vincent Christlein, and Andreas Maier. Deep Learning Computed Tomography. In *Int'l Conf. on Medical Image Computing and Computer-Assisted intervention*, pages 432–440. Springer, 2016.

[82] Tobias Würfl, Mathis Hoffmann, Vincent Christlein, Katharina Breininger, Yixin Huang, Mathias Unberath, and Andreas K Maier. Deep learning computed tomogra-

phy: Learning projection-domain weights from image domain in limited angle problems. *IEEE transactions on medical imaging*, 37(6):1454–1463, 2018.

[83] Kai Zeng and Zhou Wang. 3D-SSIM for video quality assessment. In *2012 19th IEEE International Conference on Image Processing*, pages 621–624. IEEE, 2012.

[84] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.

[85] Chen Zhang, Guangyu Sun, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(11):2072–2085, 2018.

[86] Hui Zhang, Mingxin Xia, and Guangshu Hu. A multiwindow partial buffering scheme for FPGA-based 2-D convolvers. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54(2):200–204, 2007.

[87] Xinyu Zhang, Srinjoy Das, Ojash Neopane, and Ken Kreutz-Delgado. A design methodology for efficient implementation of deconvolutional neural networks on an FPGA. *arXiv preprint arXiv:1705.02583*, 2017.

[88] 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.

[89] Zhicheng Zhang, Shaode Yu, Wenjian Qin, Xiaokun Liang, Yaoqin Xie, and Guohua Cao. Ct super resolution via zero shot learning. *arXiv preprint arXiv:2012.08943*, 2020.

[90] Chuansheng Zheng, Xianbo Deng, Qing Fu, Qiang Zhou, Jiapei Feng, Hui Ma, Wenyu Liu, and Xinggang Wang. Deep Learning-based Detection for COVID-19 from Chest CT using Weak Label. *MedRxiv*, 2020.

[91] Chenhong Zhou, Changxing Ding, Xinchao Wang, Zhentai Lu, and Dacheng Tao. One-pass multi-task networks with cross-task guided attention for brain tumor segmentation. *IEEE Transactions on Image Processing*, 29:4516–4529, 2020.

[92] Yi Zhou, Xiaodong He, Lei Huang, Li Liu, Fan Zhu, Shanshan Cui, and Ling Shao. Collaborative learning of semi-supervised segmentation and classification for medical images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2079–2088, 2019.

Appendices

Appendix A

CT Images from Different Datasets

In order to train, validate and test different AIs used in the project, CT images were collected from different sources. The complete dataset consists of 1000 CT scans of COVID-19 and healthy patients, as mention in Table 2.5 and 2.6. Some CT images from each data sources are shown in this Appendix.

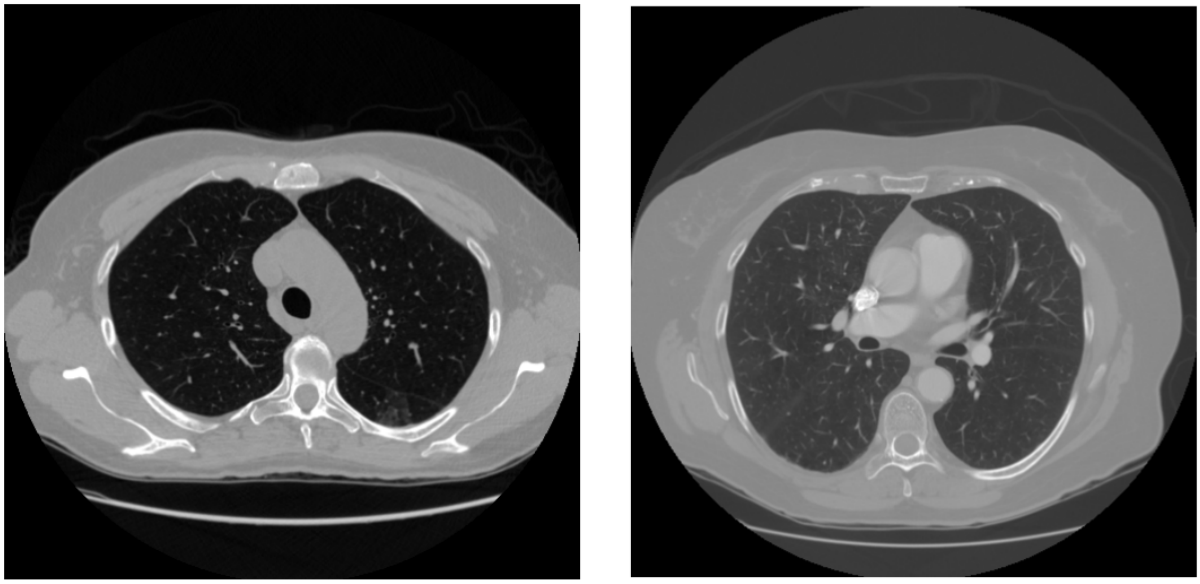
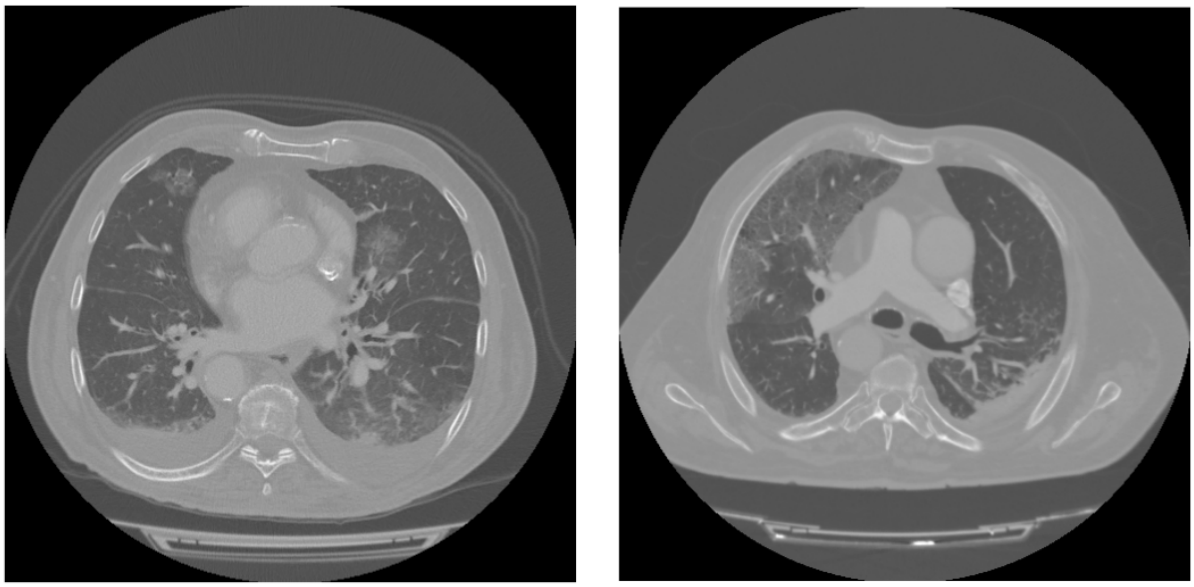
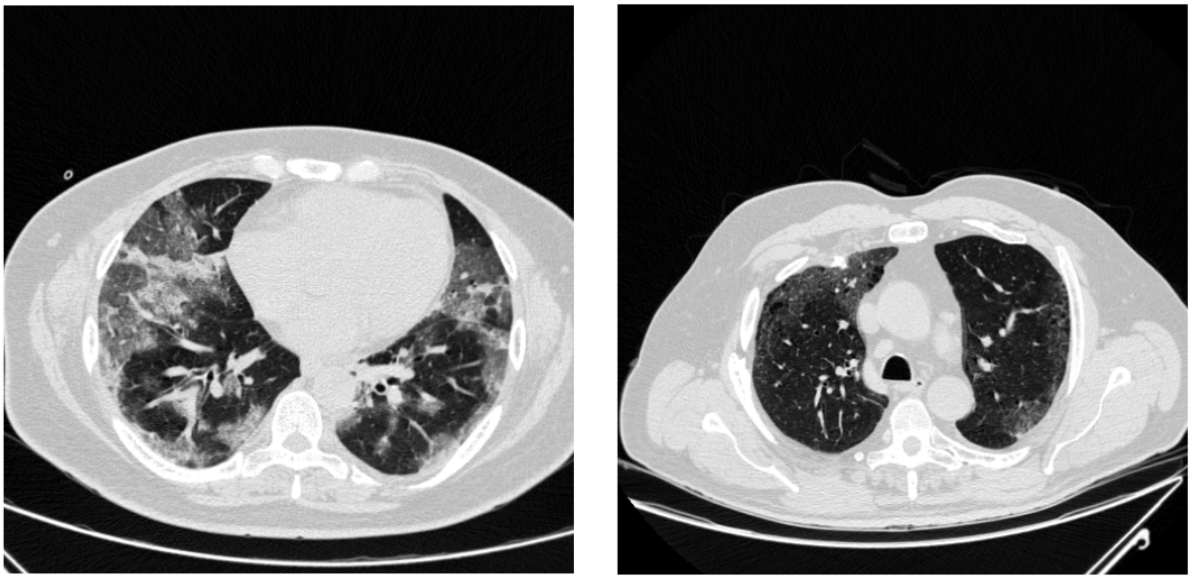


Figure A.1: CT images from LIDC dataset. CT Data source: LIDC [47]

Figure A.2: CT images of healthy patients

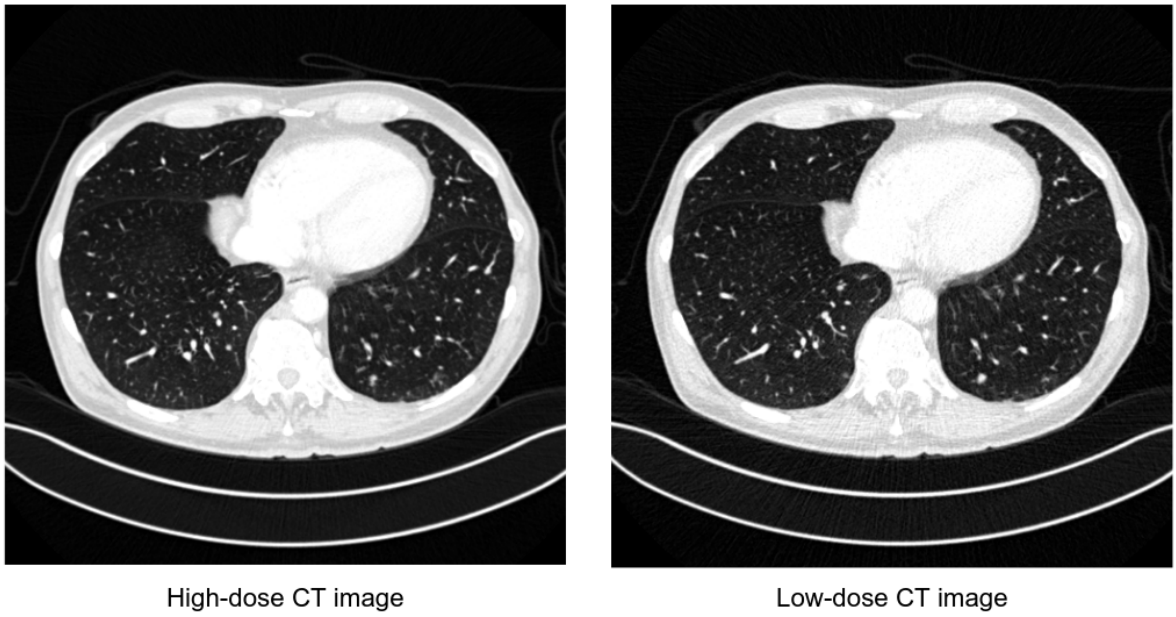


(a) CT images from BIMCV dataset. CT Data source: BIMCV [54]

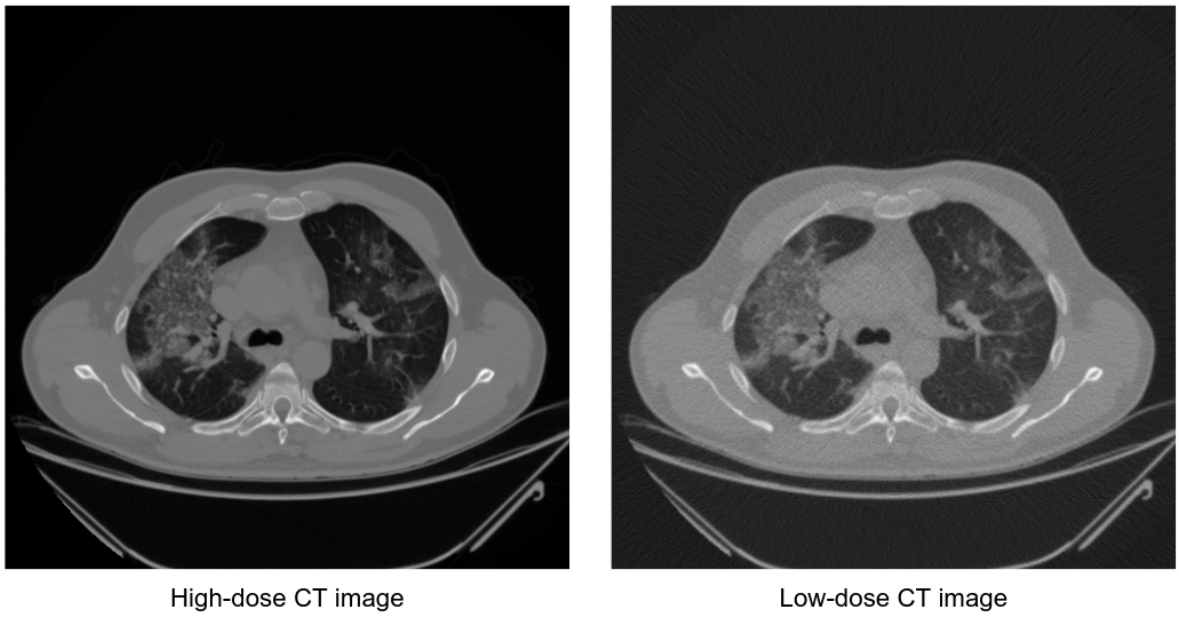


(b) CT images from MIDRC dataset. CT Data source: MIDRC [53]

Figure A.3: CT images of COVID-19 patients



(a) CT images from mayo clinic dataset. CT Data source: Mayo clinic [52]



(b) Simulated CT images from BIMCV dataset. CT Data source: BIMCV [54]

Figure A.4: High and low-dose CT images