ComputeCOVID19+: Accelerating COVID-19 Diagnosis and Monitoring via High-Performance Deep Learning on CT Images

Garvit Goel
garvit217@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Atharva Gondhalekar
atharva1@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Jingyuan Qi
jingyq1@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Zhicheng Zhang
zzc623@stanford.edu
Stanford University
Palo Alto, California, USA

Guohua Cao
ghcao@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Wu-chun Feng
feng@cs.vt.edu
Virginia Tech
Blacksburg, Virginia, USA

ABSTRACT

The COVID-19 pandemic has highlighted the importance of diagnosis and monitoring as early and accurately as possible. However, the reverse-transcription polymerase chain reaction (RT-PCR) test results in two issues: (1) protracted turnaround time from sample collection to testing result and (2) compromised test accuracy, as low as 67%, due to *when* and *how* the samples are collected, packaged, and delivered to the lab to conduct the RT-PCR test. Thus, we present ComputeCOVID19+, our computed tomography-based framework to improve the testing speed and accuracy of COVID-19 (*plus* its variants) via a deep learning-based network for CT image enhancement called DDnet, short for DenseNet and Deconvolution network. To demonstrate its speed and accuracy, we evaluate ComputeCOVID19+ across several sources of computed tomography (CT) images and on many heterogeneous platforms, including multi-core CPU, many-core GPU, and even FPGA. Our results show that ComputeCOVID19+ can significantly shorten the turnaround time from days to minutes and improve the testing accuracy to 91%.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; **Artificial intelligence**; **Machine learning**.

KEYWORDS

AI, biomedical imaging, COVID-19, computed tomography, coronavirus, deep learning, FPGA, GPU, neural network

1 INTRODUCTION

Since the discovery of COVID-19 in December 2019, it has resulted in 184,015,446 confirmed cases and 3,980,350 deaths worldwide, as of 4 July 2021 [7, 21]. Even more worrisome is that the total number of *actual* cases is unknown. As much as 50% of the population is asymptomatic [34], and, in turn, unwittingly serve as contagious transmitters. Furthermore, Johns Hopkins University reports that 59% of COVID-19 spread comes from *asymptomatic transmission* (i.e., 35% from presymptomatic individuals and 24% from individuals who never develop symptoms) [20]. To compound matters further, the accuracy of the standard COVID-19 RT-PCR test is mediocre with *one in three* producing a *false negative*, i.e., 67% sensitivity [24].

Thus, we present ComputeCOVID19+, a deep-learning (DL) framework that delivers much higher sensitivity (91%) than the RT-PCR test (67%) and much faster turnaround time (≈ 5 *minutes*) than RT-PCR (≈ 4 *hours* per test with multi-day turnaround time). The improved sensitivity is due to our enhanced imaging and analysis of lung CT images while the faster turnaround time is due to better accessibility and less dependency on materials and labor.

Patients with COVID-19 possess lung CT scans that exhibit a spectrum of distinguishing hallmark features (*a.k.a.* radiological or CT abnormalities), such as ground-glass opacities (GGOs), linear opacities, vascular consolidation, reversed halo signs, and crazy-paving patterns. Figure 1 provides visual examples of some of these hallmark features found in COVID-19 patients.

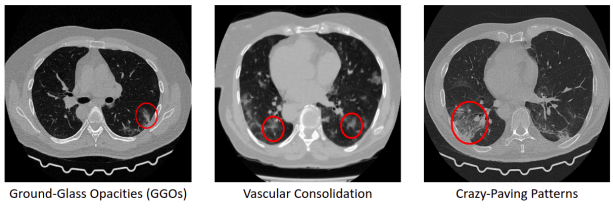


Figure 1: Abnormalities in chest CT scans of COVID-19 patients

Continued Importance of COVID-19 Testing. Despite the rollout of COVID-19 vaccines resulting in 47% of the U.S. population being fully vaccinated (but only 11% globally), as of 4 July 2021, 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). In the United Kingdom (UK), for example, the number of confirmed cases per million, as shown in Figure 2, is exponentially increasing again, due to the partial easing of restrictions and the enormous growth of the Delta variant (B.1.617.2) to 98% of the confirmed cases in the UK (as of 14 June 2021), thus marking the start of the 4th wave for the UK [21, 36].

ICPP '21, August 9–12, 2021, Lemont, IL, USA

ACM ISBN 978-1-4503-9068-2/21/08…$15.00
https://doi.org/10.1145/3472456.3473523

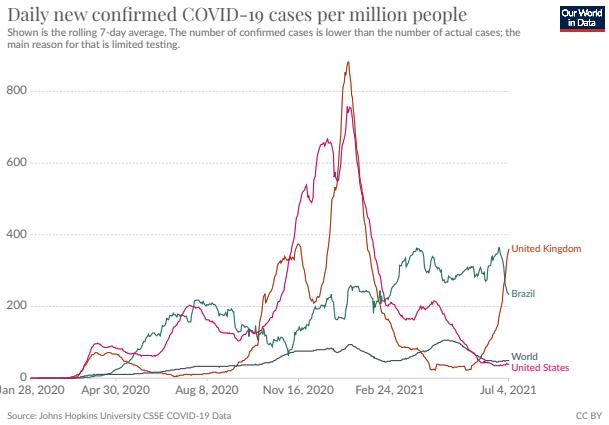


Figure 2: Confirmed Cases of COVID-19 Per Million People [21, 36]

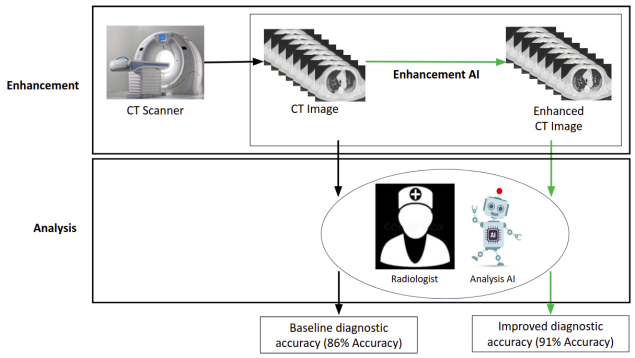


Figure 3: ComputeCOVID19+ framework. The green arrows represent the ComputeCOVID19+ workflow, where our image enhancement measurably improves the accuracy of COVID-19 diagnosis. (Analysis AI consists of Segmentation AI and Classification AI.)

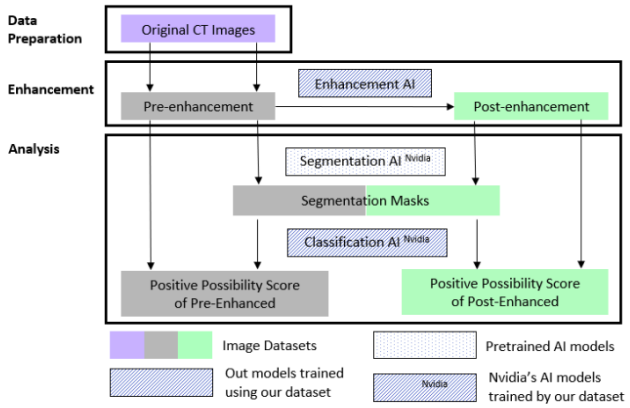


Figure 4: Workflow for testing the ComputeCOVID19+ framework

In summary, our ComputeCOVID19+ framework improves the testing speed and accuracy of COVID-19 (plus its variants) by making the following contributions:

- Novel algorithms and software for high-fidelity CT image construction and high-precision interpretation of COVID-19.
- Performance evaluation of our ComputeCOVID19+ framework on CT images with respect to speed and accuracy.
- Validation of ComputeCOVID19+ with clinical COVID-19 data.

The rest of the paper is organized as follows. In §2, we present the ComputeCOVID19+ framework and its underlying software architecture, followed by details of our network training for ComputeCOVID19+ in §3. In §4, we describe the optimizations applied for accelerating the parallelized training and inference of AI on a given heterogeneous platform. Then, we present an evaluation of the performance and accuracy of our framework in §5. In §6, we compare our ComputeCOVID19+ framework with the current state of the art for diagnosing COVID-19 and, in turn, further articulate and delineate the contributions of this paper. Finally, we provide future directions for this work in §7 and conclude with §8.

2 ComputeCOVID19+ FRAMEWORK

ComputeCOVID19+ is our computationally-based deep-learning (DL) diagnosis and monitoring framework for COVID-19. It adapts and extends the DenseNet & Deconvolution neural network (DDnet), initially developed for sparse-view CT reconstruction [45], to realize high-quality CT imaging and high-accuracy diagnosis of COVID-19. By deploying ComputeCOVID19+ to the widely available CT scanners nationwide,[1] we seek to enable more rapid, more accessible, and more accurate detection of COVID-19 with higher sensitivity and accuracy. Furthermore, ComputeCOVID19+ can deliver better and more timely diagnostic monitoring for progressing COVID-19 patients. Figure 3 provides an overview of the ComputeCOVID19+ framework. Our results show that it can improve the accuracy of CT-based diagnosis of COVID-19 from 86% to 91%.

Due to complexity of our DL algorithms for CT image enhancement and analysis, we leverage high-performance computing (HPC) via multi-core CPUs, many-core GPUs, and FPGAs to reduce the turnaround time for COVID-19 diagnosis from days via the RT-PCR test to only minutes using CT scanning and our ComputeCOVID19+ framework, where inference completes in less than *one second*.

ComputeCOVID19+, which is based on a chest CT and image enhancement algorithm [45], consists of three AI-based tools: (1) Enhancement AI, (2) Segmentation AI, and (3) Classification AI. We evaluate them across many computing devices, using the workflow shown in Figure 4. 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.

2.1 Data Preparation

In order to train our 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

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

Consortium Image Collection. These radiological data sources contain 3D chest CT scans composed of 2D image slices, each of size 512×512 pixels. Table 1 provides a description of each data source.

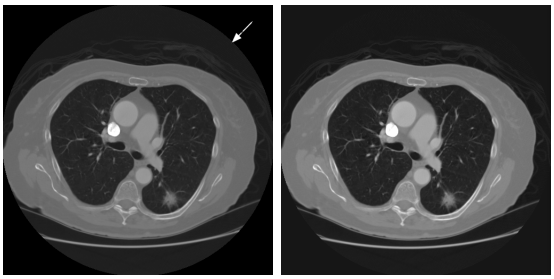
Table 1: Description of data sources

Data Source	Contents
Mayo Clinic	Eight (8) healthy chest CT scans & assoc. projection data at full & quarter dosage
Medical Imaging Databank of the Valencia Region (BIMCV)	X-ray scans & 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

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

- 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 datasets, as shown in Figure 5.
- Keeping CT scans with at least 128 two-dimensional (2D) images slices, in order to maintain isotropy in CT scans for better segmentation and classification with 3D networks.

Additional details on the CT data sources are in §3.1.2 and §3.3.2.



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

Figure 5: Removal of circular segmentation in CT images

2.2 Image Enhancement

Our Enhancement AI tool from ComputeCOVID-19+ uses DDnet for CT image enhancement [45]. DDnet consists of a convolution network with 37 convolution layers and a deconvolution network with eight deconvolution layers, as shown in Figure 6. The convolution network, deconvolution network, and shortcut connections distinguish DDnet from existing state of the art.

2.2.1 Convolution Network. This consists of four dense blocks for feature extraction from the input image [16], as shown in Figure 7. Each dense block contains four densely connected layers (i.e., the input to each layer is concatenated with the inputs of all the previous layers), which facilitate feature reuse and mitigate the exploding and vanishing gradient problems. 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 in both the x and y dimensions,

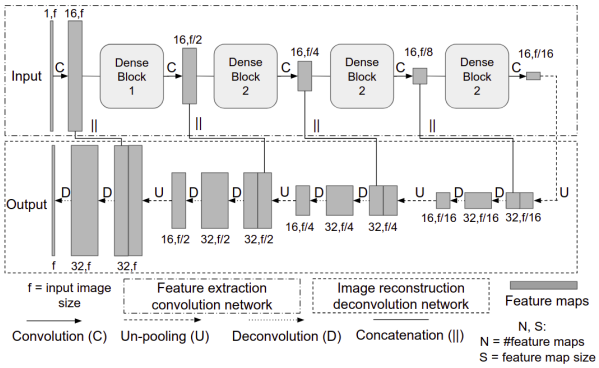


Figure 6: The architecture of DDnet

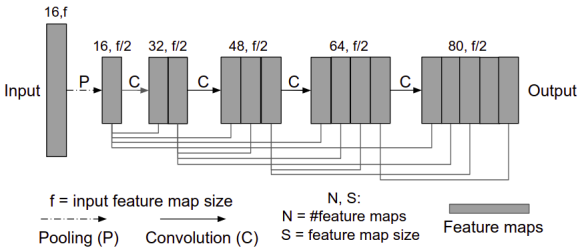


Figure 7: The architecture of dense block

resulting in better memory efficiency and less sensitivity to input variations.

2.2.2 Deconvolution Network. This reconstructs images from the extracted features. It has eight deconvolution layers and four un-pooling layers. The un-pooling operation scales the feature maps by a factor of two in both the x and y dimensions using bi-linear interpolation. Table 2 shows the size of the input and output feature maps and the filters for each convolution and deconvolution layer.

2.2.3 Shortcut Connections. These concatenate the outputs from different layers in the network. Shortcut connections facilitate feature reuse and better information flow through the network [28], resulting in a better-trained network. In addition to the local shortcut connections, DDnet uses shortcut 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 shortcuts are called global connections.

2.3 Image Classification

ComputeCOVID19+ leverages the workflow in [13] — specifically, our Segmentation AI and Classification AI tools — to classify CT images into positive and negative COVID-19 test cases.

2.3.1 Segmentation AI. This 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 with the goal of changing the characteristics of the image to be more meaningful, thus facilitating better interpretation and classification. For chest CT images, isolating the lungs via segmentation provides better feature

Table 2: Input and output sizes and filter size of feature maps for the convolution and deconvolution layers 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=$\begin{bmatrix}1\times1\\5\times5\end{bmatrix}\times4$, 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=$\begin{bmatrix}1\times1\\5\times5\end{bmatrix}\times4$, 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=$\begin{bmatrix}1\times1\\5\times5\end{bmatrix}\times4$, 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=$\begin{bmatrix}1\times1\\5\times5\end{bmatrix}\times4$, 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

extraction and, in turn, higher accuracy for COVID-19 detection. Specifically, ComputeCOVID19+ uses an anisotropic hybrid network (AH-Net) [27], adapted for 3D CT image segmentation, and maintains consistency between slices in 3D volumes.

2.3.2 Classification AI. To distinguish CT scans with COVID-19 symptoms, Classification AI from our ComputeCOVID19+ framework uses the DenseNet-121 network [16] 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 on the basis of 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.

3 NETWORK TRAINING

The loss function, hyperparameters, and CT data used for training each AI model are explained in this section. For optimal training, the hyperparameters are tuned by perturbing one parameter while keeping others fixed and analyzing the quantitative results.

3.1 Enhancement AI

3.1.1 Network Parameters. To find the optimal mapping function that enhances the quality of CT images, our Enhancement AI tool is trained with CT images of size 512×512 pixels. To avoid integer overflow, CT image data, which is usually expressed in hounsfield units (HU), is converted to floating-point data within the data range [0, 1], inclusive, before feeding it into the network.

For back propagation, the network uses a composite loss function L that combines the mean square error (MSE) and multi-scale structural similarity index metric (MS-SSIM). The MS-SSIM [42] compares the luminance, contrast, and structure similarity between two images. The loss function, L, is given by Equation (1) below.

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

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

Network weights are updated via the Adam optimizer [23]. The learning rate is set to 10^{-4} and exponentially reduced by a factor of 0.8 each epoch. The network is trained with one CT image per batch for 50 epochs. All filters are initialized with a random Gaussian distribution with a mean of zero and standard deviation of 0.01.

3.1.2 Data Collection. To train our DDnet in Enhancement AI, we used 5120 chest CT images from two sources, as described below.

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

Low X-ray Dose CT Images (Simulated Data). While there is plenty of CT data available, low X-ray dose CT images are *not* readily available. Thus, we simulated such 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 who tested positive for COVID-19.

To create 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 [39]. 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.

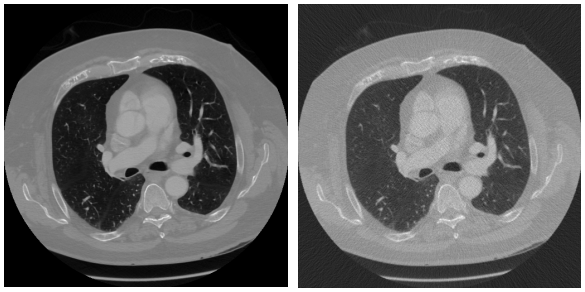
The other CT geometry parameters are summarized below:

- The distance between source and detector and source and center of object were set at 1500 mm and 1000 mm, respectively.
- 720 projections were evenly acquired across a 360-degree scan.
- 1024 pixels were used for X-ray detection.

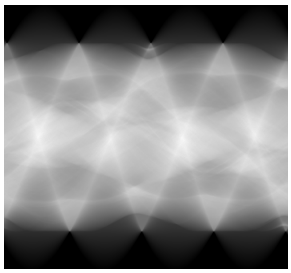
Low X-ray dose CT images were then reconstructed using filter back projection (FBP) from the simulated projection data. Figure 8 shows a sample simulated sinogram and associated CT image reconstructed using FBP. From the simulated dataset, we used 2816, 484, and 484 CT images for training, validation, and testing, respectively.

3.2 Segmentation AI

CT data for training Segmentation AI requires labeled CT scans, where each pixel in the scan is classified as either background or foreground. Such labelling is tedious and time-consuming and



(a) Original CT image (b) Simulated low-dose CT image



(c) Sinogram from projection data

Figure 8: Low X-Ray dose CT image simulation

requires radiological expertise. To address this, ComputeCOVID19+ leverages the pre-trained Segmentation AI model from Nvidia [33].

The trained Segmentation AI model ingests our 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, torso, and everything outside the body, are classified as background. The binary map is then multiplied with the input CT scan to generate the segmented CT scan.

3.3 Classification AI

3.3.1 Network Parameters. Using Nvidia's Clara Train pipeline [33], we train our own Classification AI model with 3D CT images of size 512×512×n, where n is the number of 2D image slices in one 3D CT scan. Unlike Enhancement AI, Classification AI uses CT image data represented in hounsfield units (HU) as inputs. The back propagation uses binary cross-entropy as loss. The loss function is given by Equation (2):

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log[p(y_i)] + (1-y_i)\cdot \log[1-p(y_i)] \quad (2)$$

where y is the target label (1 for the positive case and 0 for the negative case in this framework) and $p(y)$ is the predicted probability of the 3D scan being classified as positive for all N scans in a batch.

Weights are updated using the Adam optimizer [23]. The learning rate is initialized to 10^{-6}. Gaussian noise is added with probability 0.75 and variance of 0.1. Image contrast is adjusted with 0.5 probability. The scale of image intensity oscillates with 0.1 magnitude.

3.3.2 Data Collection. We used 305 3D chest CT scans for the training and validation of our Classification AI model. These CT scans were obtained from three radiological data sources.

- Medical Imaging Databank of the Valencia Region (BIMCV) [31]
- Medical Imaging and Data Resource Center (MIDRC) [30]
- Lung Image Database Consortium Image Collection (LIDC) [29]

The BIMCV and MIDRC datasets provide 3D chest CT scans of COVID-19 patients. These CT scans are labeled as positive ground truth. CT scans showing symptoms of COVID-19 are manually filtered for selection.

4 OPTIMIZING PARALLEL TRAINING AND INFERENCE OF AI

Training a CNN and doing inference on the trained network are computationally expensive and require large computational and memory bandwidth. The acceleration of these processes on parallel computing devices requires knowledge of the underlying hardware and processing demands for adequate utilization of the available computing and memory resources. In §4.1 and §4.2, 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 Training of Enhancement AI

We implemented Enhancement AI using PyTorch and parallelized it for a multi-GPU system using the DistributedDataParallel package [35], 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 [8] to synchronize processes.

4.2 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), and/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 [32].

Inference requires six operations for image enhancement: convolution, non-linear activation, batch normalization, pooling, deconvolution, and un-pooling. The data exchange between the host (CPU) and device (GPU) is minimized by using the memory available on the device platform. We also applied a set of application-specific and architecture-aware optimizations as well as FPGA-specific optimizations for each OpenCL kernel.

4.2.1 Application-Specific Optimizations. Recurring load and store operations in the deconvolution kernel require high memory bandwidth and result in multiple cache misses, thus degrading performance. To overcome this, we used inverse coefficient mapping [4, 44] to refactor the kernel for deconvolution. Instead of directly deconvolving the input, we first determined the input blocks needed for calculating each output element, and then, each element in

the input block and corresponding weight coefficient was multiplied and added at once before the result was written to the global memory. Figure 9 illustrates this optimization.

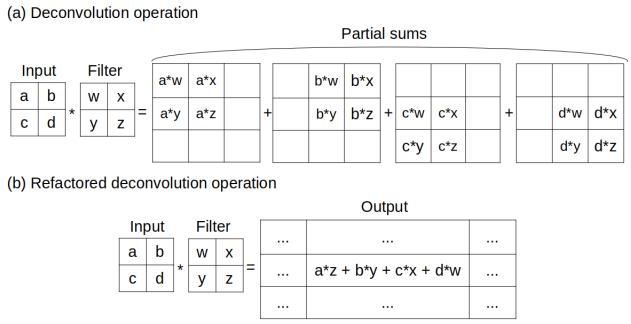


Figure 9: Deconvolution optimization. (a) Deconvolution operation: Partial sums are calculated by multiplying an element in input and each element in filter. These partial sums are then added to get the final output. (b) Refactored deconvolution operation: Each output is calculated by determining which input elements affect that output and applying multiply and add operations before being written.

4.2.2 Architecture-Aware Optimizations. The architecture-aware optimizations that are then applied to the aforementioned refactored kernels include memory prefetching and loop unrolling.

- *Memory Prefetching:* This standard optimization caches a load in local memory or registers prior to its usage. We prefetch the loop bounds (size of input, size of output, size of filters) by storing these values in local integer variables.
- *Loop Unrolling:* This optimization improves the performance of kernels by reducing the number of branch instructions, whether on the CPU, GPU, or FPGA. For the FPGA, unrolled loops improve performance by generating extra hardware to support multiple iterations of a loop and resolving data dependencies between iterations [17]. In our implementation, we unrolled the multiply-and-adder loop in DDnet by a factor of five in the convolution and deconvolution kernels, respectively. Because the size of the filters used in convolution and deconvolution is less than or equal to 5, this unrolling factor fully unrolls the loop and achieves the best performance.

4.2.3 FPGA-Specific Optimizations. Unlike the CPU and GPU, the underlying computing hardware in an FPGA is *not* fixed. The ability to reconfigure compute logic allows additional optimizations to be implemented on FPGA. These optimizations are explained below.

- *Compute-Unit Replication*: Replication of compute units improves the performance of kernels by increasing the computational bandwidth of the hardware (at the expense of using more silicon hardware). In our implementation, we identified two compute units each for the convolution and deconvolution kernels, respectively [17, 18], as ideal.
- *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 [17]. The FPGA OpenCL compiler generates the hardware to support SIMD instructions. In our convolution and deconvolution kernels, we used vector load and vector multiply operations.
- *Dedicated Kernels*: Multiple dedicated kernels that operate with fixed inputs and parameters can sometimes lead to higher throughput at the expense of extra hardware. Having fixed parameters in the kernels results in better pipelines with low initialization intervals. In our application, we used dedicated kernels for the convolution and deconvolution operations with a 5×5 filter size.
- *Runtime Reconfiguration*: Optimizations such as vectorization, loop unrolling, compute-unit replication, and dedicated kernels create extra hardware to achieve higher compute bandwidth. However, simultaneous application of these optimizations leads to excessive resource utilization on the target FPGA, resulting in compilation failures. To address this problem of excessive resource utilization, we used *runtime reconfiguration* if the overhead of FPGA reconfiguration was less than the gain in performance with optimized kernels. The ability to configure FPGA hardware at runtime provides room for extra hardware configuration, yielding higher throughput and, in turn, better performance.

To make use of FPGA runtime reconfiguration (and combine it with the other aforementioned optimizations), we split the execution of DDnet into two kernels: convolution and deconvolution. As shown in Figure 6, the convolution kernel consists of convolution, batch normalization, non-linear activation, and pooling operations; the deconvolution kernel consists of deconvolution, batch normalization, non-linear activation, and un-pooling operations. Figure 10 shows the runtime reconfiguration of DDnet for the FPGA.

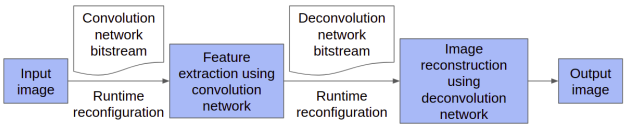


Figure 10: Runtime reconfiguration of DDnet

5 EVALUATION

We evaluate our ComputeCOVID19+ framework with respect to both *computing performance* and *accuracy*. For computing performance, we first evaluate the performance of *training* and then the performance of *inference* in the ComputeCOVID19+ framework. The training is conducted in a distributed computing setup with multiple GPUs. We analyze the impact of parallelizing the training on execution time and accuracy.

Then, the inference of the trained network, which is computationally less expensive than the training, is evaluated on many heterogeneous platforms, including multi-core CPU, many-core GPU, and FPGA.

The accuracy of ComputeCOVID19+ framework is evaluated by analyzing the enhancement and analysis modules, individually and together. To further understand the impact of ComputeCOVID19+'s image enhancement via Enhancement AI, we compare the results from using the original CT scans (i.e., Segmentation AI + Classification AI) to the results from using the enhanced CT scans (i.e., Enhancement AI + Segmentation AI + Classification AI).

5.1 Performance of ComputeCOVID19+

For the training and inference of Classification AI and Segmentation AI, we used a high-performance workstation equipped with an Intel Core i9-10900K CPU and Nvidia GeForce RTX 3090 GPU, coupled with 32 GB of system memory.

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

- Many-core GPUs, including Nvidia V100, Nvidia P100, Nvidia T4, and AMD Radeon Vega Frontier
- Multi-core CPU, i.e., Intel Xeon Gold 6128
- FPGA, i.e., Intel Arria 10 GX 1150

5.1.1 Training & Inference of Segmentation and Classification. Both the Segmentation AI and Classification AI tools run in the Nvidia Clara environment. For the former, we use the pre-trained model from NVIDIA "as is"; for the latter, we train the NVIDIA classification model with our CT scans. In all, the performance of the tools is in line with the performance reported in [13]. On an Nvidia GeForce RTX 3090 GPU, the training of Classification AI for 100 epochs and 305 CT scans took 4 hours and 28 minutes. On the same GPU platform, the runtime for inference of Segmentation AI and Classification AI took 45.88 seconds and 5.90 seconds, respectively.

5.1.2 Training of Enhancement AI on a Multi-GPU System. Table 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 [22].

5.1.3 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. The best performance comes from the Nvidia V100 GPU, followed by the Nvidia P100, AMD Radeon Vega Frontier, and Nvidia T4 GPUs.

To better understand how we achieved the above runtimes for inference, we profiled the (serial) kernel code for convolution, deconvolution, and other kernels to be 31.50, 299.86, and 0.46 seconds, respectively, on an Intel Xeon Gold 6128 CPU. Clearly, the deconvolution kernel is the most computationally expensive, followed by the convolution kernel. Thus, for the CPU (and GPUs), parallelizing and optimizing the deconvolution and convolution kernels delivered the most benefit, as shown in Table 5. Convolution went from 31.50 seconds down to 0.495 seconds (i.e., speedup ≈ 64×), and deconvolution dropped from 299.86 seconds to only 1.078 seconds (i.e., speedup ≈ 278×) on the Intel Xeon Gold 6128 CPU.

Comparing deconvolution operations and convolution operations in DDnet, the convolution uses approximately 1.87× floating point operations and global memory accesses (there are 37 convolution layers and 8 deconvolution layers in DDnet). However, due to the irregular memory accesses and expensive integer division operations in deconvolution kernel, the deconvolution kernel has higher execution time than convolution kernel on CPU and GPU. Vectorization of deconvolution kernel simplifies the memory accesses and reduces the count of integer division operations in

Table 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 an Nvidia T4 GPU. (hh:mm::ss) = (hours:minutes:seconds).

Table 4: Inference runtime for the Enhancement AI tool

Platform	Number of Cores	Maximum Bandwidth (GB/s)	Maximum Frequency (MHz)	PyTorch Runtime (seconds)	OpenCL Runtime (seconds)
Nvidia V100 GPU	5120 (CUDA cores)	900	1380	0.22	0.10
Nvidia P100 GPU	3584 (CUDA cores)	732	1328	0.73	0.25
AMD Radeon Vega Frontier GPU	4096 (Stream Proc.)	480	1600	–	0.25
Nvidia T4 GPU	2560 (CUDA cores)	320	1590	1.29	0.29
Intel Xeon Gold 6128 CPU	24 (CPU cores)	119	3400	5.52	1.64
Intel Arria 10 GX 1150 FPGA	2 (CUs)*	< 3	184	–	16.74

* Two compute units (CUs) are generated using vendor-specific attribute, __attribute__((num_compute_units(2))
– : The PyTorch implementation is not portable to this platform.

Table 5: Event-based time of the optimized OpenCL kernels for Enhancement AI inference. Execution time is reported in seconds.

Platform	Kernel runtime (seconds)		
	Convolution	Deconvolution	Other kernels
Nvidia V100 GPU	0.036	0.059	0.004
Nvidia P100 GPU	0.075	0.169	0.005
AMD Radeon Vega Frontier GPU	0.082	0.170	0.005
Nvidia® T4	0.123	0.153	0.016
Intel Xeon Gold 6128 CPU	0.495	1.078	0.057
Intel Arria 10 GX 1150 FPGA	9.819	2.839	3.991

deconvolution kernel. This reduces the execution time of deconvolution kernel on FPGA significantly and makes the convolution kernel more expensive on FPGA.

Because the Enhancement AI network is dense and the number of load and store operations is just as significant as the number of floating-point operations, particularly in the convolution and deconvolution kernels, as shown in Table 6, the combination of row-major and column-major accesses in the convolution and deconvolution kernels provides little opportunity for coalesced memory accesses. As a consequence, the performance of our optimized OpenCL kernels across the various platforms from Table 4 tracks with the memory bandwidth of the platforms. The Nvidia V100 GPU has the highest bandwidth (as well as a significantly large number of CUDA cores). Thus, the V100 outperforms the other platforms, as expected, due to the aforementioned memory-bound nature of the Enhancement AI tool.

Table 7 shows the runtime for inference using DDnet on HPC platforms with different optimizations, as described in §4.2. Refactoring the kernel reduced the number of recurring loads and stores from/to the global buffer in the deconvolution kernel and delivered significant performance improvement across all platforms. Loop unrolling and prefetching the relatively few filter parameters achieved only marginal speedup because the problem is memory-bound. Compared to our PyTorch inference implementation, our OpenCL implementation is approximately 3.4× faster on the CPU and at least 2.0× faster on the Nvidia GPUs.

The OpenCL kernels designed for GPUs are functionally portable, but not performance portable, on FPGAs. Extracting competitive performance from the reconfigurable logic in FPGAs via OpenCL necessitates the use of vendor-specific attributes. To that end, we implemented the optimizations proposed in §4.2.3, which include loop unrolling, compute-unit (CU) replication, and manual vectorization. OpenCL kernels get mapped to the underlying hardware resources of FPGA, which include RAM blocks, registers, and arithmetic logic units. Limited availability of these resources impacts the extent to which the optimizations mentioned above can be applied. Keeping this limitation in mind, we used appropriate attributes for loop unrolling (by a factor of five) and compute-unit (CU) replication (by a factor of two) in the convolution and deconvolution kernels. Even after loop unrolling and CU replication, deconvolution remained the most computationally-expensive kernel, so we also applied vectorization (by a factor of five) to the deconvolution and used a 5×5 filter size and constant values for stride and padding.

In summary, we presented a performance evaluation of our ComputeCOVID19+ framework, which demonstrated the following:

- The efficacy of our solution in accurately diagnosing COVID-19.
- The adaptability of our inference solution across a wide spectrum of heterogeneous platforms, including CPU, GPU, and FPGA.
- Inference performance that is competitive across heterogeneous platforms, even a low-power device such as an FPGA.

Table 6: Global memory load/store and floating-point operations count for individual kernels with an input of size 512×512×32

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	8.4	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

[1] 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.
[2] The number of floating-point operations and memory accesses is obtained by implementing counters in each kernel.

Table 7: Execution time profile of entire DDnet with different optimizations. Execution time is reported in seconds. REF: Refactoring, PF: Prefetching, and LU: Loop Unrolling.

Platform	Baseline	Baseline + REF	Baseline + REF + PF	Baseline + REF + PF + LU
Nvidia GPU V100	63.82	0.10	0.10	0.10
Nvidia GPU P100	152.08	0.29	0.26	0.25
AMD Radeon Vega Frontier GPU	219.60	0.25	0.25	0.25
Nvidia T4	59.30	0.32	0.31	0.29
Intel Xeon Gold 6128 CPU	6.51	1.95	1.69	1.64
Intel Arria 10 GX 1150 FPGA	278.53	130.62	127.72	65.83[1]

[1] The execution time reported in this table does not use the kernel with FPGA-specific optimizations described in §4.2.3.

5.2 Accuracy of ComputeCOVID19+

To evaluate accuracy of ComputeCOVID19+, we analyze the results from our Enhancement AI, Segmentation AI, and Classification AI tools that are shown in Figure 4. For Enhancement AI, we quantify 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. For CT classification, we measure accuracy as follows:

- Accuracy, as defined by Equation (3), is the percentage of CT scans classified correctly.
- AUC-ROC: Area Under the Curve (AUC) of Receiver Characteristic Operator (ROC). The ROC curve graph is plotted using the true-positive rate (TPR), i.e., Equation (4), and the false-positive rate (FPR), i.e., Equation (5), at different thresholds.

$$Accuracy = (TP + TN)/(TP + FP + FN + TN) \tag{3}$$

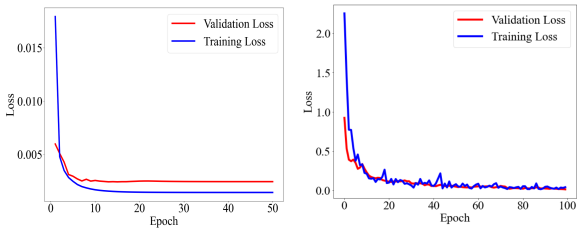
$$TPR = \frac{TP}{N} = \frac{TP}{TP + FN} \tag{4}$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{5}$$

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.

Figures 11a and 11b capture the training and validation loss curves for Enhancement AI and Classification AI, respectively.

5.2.1 Enhancement AI. Figure 12a shows the result of enhancing chest CT images from the Mayo Clinic dataset. The enhancement removed the noise present in the low X-Ray dose CT images while retaining finer details. Figure 12b shows the results of enhancing CT images from a simulated dataset. Enhancement AI removed the streaking and noise artifacts present in the image. The absolute difference maps between the low dose X-ray CT image and enhanced CT image show the efficacy of DDnet.



(a) Training and validation loss for Enhancement AI **(b) Training and validation loss for Classification AI**

Figure 11: Training loss curves

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 8 summarizes the accuracy results of Enhancement AI.

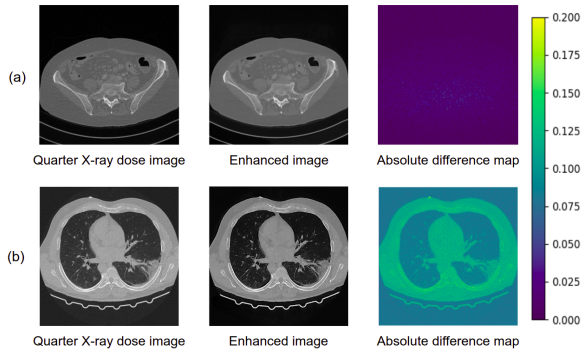


Figure 12: Image enhancement using DDnet for the (a) Mayo Clinic dataset and (b) simulated dataset.

Table 8: Accuracy results of Enhancement AI in DDnet. Y and X refers to high-dose and low-dose CT images. f(X) is the image enhanced by DDnet.

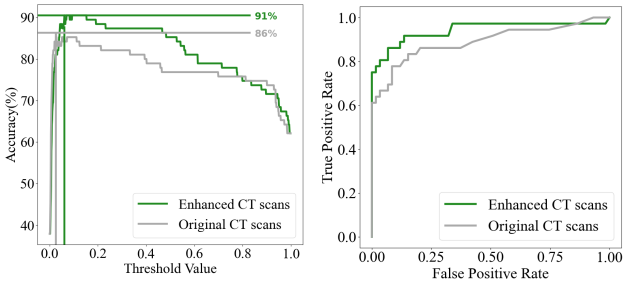
	MSE	MS-SSIM
Y-X	0.00715	96.2 %
Y-f(X)	0.00091	98.7 %

5.2.2 Segmentation AI + Classification AI. 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 13a and 13b 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.

5.2.3 Impact of Prepending Enhancement AI. 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. With enhanced CT scans, the convolution network in Classification AI extracts high quality distinctive features, enabling easier interpretation for classification. This improves the average output probability of *COVID-19 scans* to be correctly classified by 0.1136.

As noted in §5.2.2, 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 13. 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 13a and 13b respectively. Table 9 shows the result of the classification of the test dataset using a confusion matrix at an optimal threshold value of 0.061.



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

Figure 13: ComputeCOVID19+ evaluation

Table 9: Confusion matrix for classification of test data set

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

6 RELATED WORK

Here we present related work from three areas: (1) RT-PCR genetic testing vs. CT-based image testing (*a la* ComputeCOVID19+), (2) AI-based computed tomography (CT), and (3) CT image enhancement.

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

RT-PCR is the standard test for detecting COVID-19 (a.k.a. SARS-CoV-2 virus). However, a Johns Hopkins University study in 2020 showed that the accuracy of the test *varies* with the time at which the test is taken. Specifically, the false-negative rate of an infected person is 67% on the 4th day (i.e., 33% sensitivity) and only improves to 38% (i.e., 62% sensitivity) with the onset of symptoms [24].

COVID-19 testing based on CT is a compelling alternative. Research conducted in China with 877 patients [11] shows that 84% of COVID-19 patients exhibited 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 [9, 10].

6.2 AI with Computed Tomography (CT)

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 in CT scans has been extensively studied in the recent past [13, 15, 25, 38, 40, 41, 46].

6.2.1 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, like GGO, are present in only some segments of the lungs. He et al. [15] use VGG-16, ResNet, and DenseNet deep-learning (DL) networks to classify 2D CT images. They use transfer learning, coupled with momentum contrastive learning [14], to make these models agnostic to the dataset sizes and achieve 86% accuracy with a training dataset of 425 2D CT images. Similarly, Wang et al. [41] achieve 89% accuracy using an M-inception network and a dataset of 1065 CT images. Ying et al. [40] pre-process the 2D images to segment the lung region using OpenCV and achieve 86% accuracy. Li et al. [25] use U-Net-based lung segmentation and classify the images using ResNet50.

6.2.2 Three-Dimensional (3D) CNNs with 3D Volumes as Inputs. 3D CNNs extract 3D features from the input volume do *not* require any manual data preparation. Harmon et al. [13] demonstrate this by using a 3D version of AH-Net and DenseNet-121 to segment and classify the image, respectively. While they reported 90% accuracy, their accuracy drops to 86% when using our real-world datasets. Zheng et al. [46] combine image segmentation using 2D U-Net and classification using a 3D deep CNN to detect COVID-19 from CT volumes and achieve 90% accuracy with 540 CT scans.

6.3 CT Image Enhancement

With the increased use of computed tomography (CT) in medical diagnosis, low-dose X-ray CT has gained popularity due to its fast data acquisition and reduced radiation exposure. However, image reconstruction techniques like filtered back projection (FBP) [37] generate low-quality CT images from low-dose X-ray projections. Thus, techniques like iterative image reconstruction [3], sinogram completion [1, 26], and image enhancement based on deep learning (DL) are used to reconstruct high-quality CT images.

Würfl et al. [43] emulate FBP using a CNN. Cheng et al. [6] combine DL and iterative reconstruction to accelerate the algorithmic convergence using a leapfrogging strategy. Han et al. [12] use a deep residual network to estimate streaking artifacts in low-dose X-ray images. Jin et al. [19] and Chen et al. [5] use FBP for image reconstruction from projection data, followed by applying a U-Net-like CNN for image enhancement.

6.4 Comparison with Prior Work

Table 10 presents a tabular comparison of our ComputeCOVID19+ framework with other similar existing work. ComputeCOVID19+ differs from prior work as follows:

- The addition of deep learning-based CT image enhancement to medical diagnosis for improved accuracy.
- The acceleration of neural network training for CT image enhancement using GPUs.
- A hardware-agnostic realization of an image enhancement network using OpenCL, thus enabling "write-once, run-anywhere" capability on CPU, GPU, and FPGA.

7 PERSPECTIVE AND FUTURE WORK

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% improvement in this work). Therefore, as part of future work, we seek to address this limitation by also using data available from the projection domain and combining it with knowledge from medical imaging physics to reconstruct even higher-quality CT images.

While we evaluated the performance of inference in our Enhancement AI tool across a diverse set of platforms, the availability of heterogeneous platforms (e.g., FPGA or GPU) in clinical settings is limited, while the CPU is ubiquitous. As such, clinicians can make use of our trained AI models for the CPU and still achieve real-time performance, as shown in Table 4.

As a subject of future study, we plan to evaluate the framework with low-dose CT image data. Low-dose CT technology comes with the benefit of 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. Finally, with the help of radiologists and clinicians, we intend to analyze the applicability of ComputeCOVID19+ for diagnosing other maladies, such as viral pneumonia and cancer.

8 CONCLUSION

We present our research and development of ComputeCOVID19+, a CT-based framework 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. Furthermore, we implement and accelerate the complex deep-learning algorithms of ComputeCOVID19+ across a multitude of heterogeneous platforms, including multi-core CPU, many-core GPU, and even FPGA. 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%.

9 ACKNOWLEDGEMENT

This research was supported in part by NSF CCF-2031215.

We thank Dr. Cynthia McCollough, the Mayo Clinic, and the American Association of Physicists in Medicine for providing the Mayo Clinic dataset.

The authors acknowledge Advanced Research Computing at Virginia Tech for providing computational resources and technical support that have contributed to the results reported within this paper. URL: https://arc.vt.edu/. The authors also acknowledge and thank Intel for access to their Arria 10 FPGA via the Intel DevCloud.

Table 10: Comparison of ComputeCOVID19+ with existing similar work

Framework	CT scans pre-processing		2D/3D classification	Data labeling	Supported hardware for inference		
	Image enhancement	Image segmentation			CPU	GPU	FPGA
ComputeCOVID19+	✓	✓	3D	Not required	✓	✓	✓
He et al. [15]	×	×	2D	Manual[2]	✓	✓	×
M-inception [41]	×	✓	2D	Manual[2]	*	*	×
DRE-Net [40]	×	✓	2D	Manual[2]	*	*	×
Li et al. [25]	×	✓	2D	Manual[2]	*	✓	×
DeCoVNet [46]	×	✓	3D	Not required	*	✓	×
Harmon et al. [13]	×	✓	3D	Not required	×	✓	×
Serte et al. [38]	×	×	2D/3D	Not required	*	✓	×

[*]No information available.
[2]CT images showing symptoms of COVID-19 must be manually filtered.

REFERENCES

[1] M. Aharon et al. 2006. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Transactions on Signal Processing* 54, 11 (2006), 4311–4322.
[2] T. Ai et al. 2020. Correlation of Chest CT and RT-PCR Testing for Coronavirus Disease 2019 (COVID-19) in China: A Report of 1014 Cases. *Radiology* 296, 2 (2020), E32–E40.
[3] M. Beister et al. 2012. Iterative Reconstruction Methods in X-ray CT. *Physica Medica* 28, 2 (2012), 94–108.
[4] J. Chang et al. 2018. 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 (2018), 281–295.
[5] H. Chen et al. 2017. Low-Dose CT with a Residual Encoder-Decoder Convolutional Neural Network. *IEEE Trans. on Medical Imaging* 36, 12 (2017), 2524–2535.
[6] L. Cheng et al. 2017. 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*. 715–720.
[7] E. Dong et al. 2020. An Interactive Web-based Dashboard to Track COVID-19 in Real Time. *Lancet Infectious Diseases* 20, 5 (May 2020), 533–534.
[8] Facebook. [n.d.]. GLOO Communication Backend. https://github.com/facebookincubator/gloo/
[9] Y. Fang et al. 2020. Sensitivity of Chest CT for COVID-19: Comparison to RT-PCR. *Radiology* 296, 2 (2020), E115–E117.
[10] H. Gietema et al. 2020. CT in Relation to RT-PCR in Diagnosing COVID-19 in The Netherlands: A Prospective Study. *PLOS One* 15, 7 (2020), e0235844.
[11] W. Guan et al. 2020. Clinical Characteristics of Coronavirus Disease 2019 in China. *New England Journal of Medicine* 382, 18 (2020), 1708–1720.
[12] Y. Han et al. 2016. Deep Residual Learning for Compressed Sensing CT Reconstruction via Persistent Homology Analysis. *arXiv:1611.06391 preprint* (2016).
[13] S. Harmon et al. 2020. Artificial Intelligence for the Detection of COVID-19 Pneumonia on Chest CT using Multinational Datasets. *Nature Communications* 11, 1 (Dec. 2020), 1–7.
[14] K. He et al. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. In *IEEE Conf. on Computer Vision & Pattern Recognition*. 9729–9738.
[15] X. He et al. 2020. Sample-Efficient Deep Learning for COVID-19 Diagnosis Based on CT Scans. *MedRxiv* (2020).
[16] G. Huang et al. 2017. Densely Connected Convolutional Networks. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.
[17] Intel [n.d.]. *Intel FPGA SDK for OpenCL Pro Edition: Best Practices Guide*. Intel. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf
[18] Intel [n.d.]. *Intel FPGA SDK for OpenCL Pro Edition: Programming Guide*. Intel. https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html
[19] K. Jin et al. 2017. Deep Convolutional Neural Network for Inverse Problems in Imaging. *IEEE Transactions on Image Processing* 26, 9 (2017), 4509–4522.
[20] M. Johansson et al. 2021. SARS-CoV-2 Transmission from People Without COVID-19 Symptoms. *J. American Medical Assoc.* (Jan. 2021).
[21] Johns Hopkins Coronavirus Resource Center. 2020. COVID-19 Map. https://coronavirus.jhu.edu/map.html
[22] N. S. Keskar et al. 2016. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv:1609.04836 preprint* (2016).
[23] D. Kingma et al. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 preprint* (2014).
[24] L. Kucirka et al. 2020. 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 (2020), 262–267.
[25] L. Li et al. 2020. Using Artificial Intelligence to Detect COVID-19 and Community-Acquired Pneumonia Based on Pulmonary CT: Evaluation of the Diagnostic Accuracy. *Radiology* 296, 2 (2020), E65–E71.
[26] S. Li et al. 2014. Dictionary Learning Based Sinogram Inpainting for CT Sparse Reconstruction. *Optik* 125, 12 (2014), 2862–2867.
[27] S. Liu et al. 2017. 3D Anisotropic Hybrid Network: Transferring Convolutional Features from 2D Images to 3D Anisotropic Volumes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11071 LNCS (Nov. 2017), 851–858.
[28] J. Long et al. 2015. Fully Convolutional Networks for Semantic Segmentation. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.
[29] Lung Image Database Consortium Image Collection. [n.d.]. LIDC-IDRI - The Cancer Imaging Archive (TCIA) Public Access – Cancer Imaging Archive Wiki. https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI
[30] Medical Imaging & Data Resource Ctr. [n.d.]. MIDRC. https://www.midrc.org/
[31] Medical Imaging Databank of the Valencia Region. [n.d.]. BIMCV-COVID19 – BIMCV. https://bimcv.cipf.es/bimcv-projects/bimcv-covid19/
[32] A. Munshi. 2009. The OpenCL Specification. In *IEEE Hot Chips Symposium*. 1–314.
[33] NVIDIA GPU Cloud (NGC). [n.d.]. Clara Train SDK. https://ngc.nvidia.com/catalog/containers/nvidia:clara-train-sdk
[34] R. Plater. 2020. As Many as 80 Percent of People with COVID-19 Aren't Aware They Have the Virus. *Healthline* (May 2020).
[35] PyTorch 2021. *Distributed Data Parallel*. https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html
[36] M. Roser et al. 2020. Coronavirus Pandemic (COVID-19). *Our World in Data* (2020). https://ourworldindata.org/coronavirus.
[37] R. Schofield et al. 2020. Image Reconstruction: Part 1 – Understanding Filtered Back Projection, Noise and Image Acquisition. *Journal of Cardiovascular Computed Tomography* 14, 3 (2020), 219–225.
[38] S. Serte et al. 2021. Deep Learning for Diagnosis of COVID-19 using 3D CT Scans. *Computers in Biology and Medicine* (2021), 104306.
[39] R. Siddon. 1985. Fast Calculation of the Exact Radiological Path for a Three-Dimensional CT Array. *Medical Physics* 12, 2 (1985), 252–255.
[40] Y. Song et al. 2020. Deep Learning Enables Accurate Diagnosis of Novel Coronavirus (COVID-19) with CT Images. *MedRxiv* (2020).
[41] S. Wang et al. 2020. A Deep Learning Algorithm Using CT Images to Screen for Corona Virus Disease (COVID-19). *MedRxiv* (2020).
[42] Z. Wang et al. 2004. Image Quality Assessment: from Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.
[43] T. Würfl et al. 2016. Deep Learning Computed Tomography. In *Int'l Conf. on Medical Image Computing and Computer-Assisted intervention*. Springer, 432–440.
[44] X. Zhang et al. 2017. A Design Methodology for Efficient Implementation of Deconvolutional Neural Networks on an FPGA. *arXiv:1705.02583 preprint* (2017).
[45] Z. Zhang et al. 2018. A Sparse-View CT Reconstruction Method Based on Combination of DenseNet and Deconvolution. *IEEE Transactions on Medical Imaging* 37, 6 (2018), 1407–1417.
[46] C. Zheng et al. 2020. Deep Learning-based Detection for COVID-19 from Chest CT using Weak Label. *MedRxiv* (2020).