Master thesis

Aritra Mazumdar

Reconstruction of 3D CT Volume from 2D X-ray Image using Deep Learning

July 27, 2021

supervised by:
Prof. Dr.-Ing. Tobias Knopp
Prof. Dr.-Ing. Rolf-Rainer Grigat

Hamburg University of Technology
Institute for Biomedical Imaging
Schwarzenbergstraße 95
21073 Hamburg

University Medical Center Hamburg-Eppendorf
Section for Biomedical Imaging
Martinistraße 52
20246 Hamburg

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Quellen und Hilfsmittel angefertigt zu haben.

Hamburg, den 27.07.2021

Abstract

X-ray (X-radiation) is the cheapest and the most easily accessible form of medical imaging modality. But with the complete 3D anatomy of human body being collapsed in a single 2D X-ray plate, it is hard for physicians to do medical diagnosis from an X-ray and needs extremely skilled radiologists. CT (Computed Tomography) scan on the other hand obtains a high quality cross sectional images of the human body, with the help of multiple X-rays cast from different angle, thereby providing a lot more detail compared to X-rays. But CT scan is much more expensive, much less accessible and exposes patients to a higher degree of radiation. This thesis aims at reconstructing this 3D CT volume image from the 2D X-ray image of a patient, using Deep Learning, thereby combining the best aspects of the two different imaging modalities along with Artificial Intelligence. A patient can easily obtain an X-ray at low cost and from any medical centre, which are mostly equipped with X-ray machines, but not necessarily with CT scan setups. Then the patient can generate the CT scan from his/her X-ray using our application. In this thesis, we discuss in detail different approaches in achieving the aforesaid goal, by taking into account different data pre-processing methods and network architectures apart from other associated parameters and hyperparamters. It describes in stages the story of implementation and lays out a comparative analysis of the impact of change of every important constituent in this complete setup on the training process and network evaluation. This thesis entails a groundbreaking research using Deep Learning, in achieving the above-mentioned goal, and marks the beginning of a novel methodology, which can revolutionise this particular aspect of medical imaging, and paves the way for further research and contributions in this direction in future.

Contents

List of Figures

List of Tables

1 Introduction

1.1. Medical Imaging

Medical imaging [35] is the mechanism of imaging the internal complicated structures of a human body for the purpose of diagnosis and treatment. It is a type of medical imaging, which makes use of radiology. There are different medical imaging techniques that have been in practice for a long time. The most common of them are:

- Computed Tomography (CT) Scan
- Magnetic Resonance Imaging (MRI)
- X-radiation (X-ray)
- Endoscopy
- Ultrasonography
- Fluoroscopy
- Nuclear Medicine

The types of medical imaging which are of concern for this thesis are X-rays and CT scan. Hence, let's have a brief overview of these two modalities from end user's perspective.

X-ray : X-rays [33, 53, 77] are a form of electromagnetic radiation that can pass through any kind of solid object, including human body. The penetration capability of X-rays in any object depends on the density of the object. So, when it comes to X-ray penetration in human body, it will lead to more absorption by the bones and much lesser by the soft tissues. So structures of concern like soft tissues, which are overlaid by bones, can easily be obscured by them. Not only that, the entire structure of the human body gets collapsed in a single 2 Dimensional (2D) X-ray, because of it being a projection based imaging technique. Both these factors lead to analysis of anatomy of concern extremely difficult and needs really skilled radiologists. But it still remains the most cheap and the most easily accessible imaging modality [25] till date.

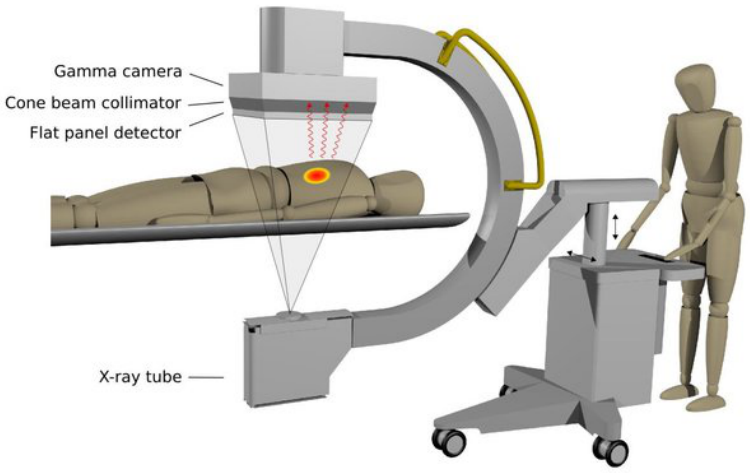


Figure 1.1.: X-ray imaging technique. X-ray travels from the generator (X-ray tube) through the body of the patient to the X-ray detector. The detector converts the radiation into a signal, which is the final raw X-ray image. This diagram shows a dual-layer detector, consisting of a flat panel detector, a cone beam collimator and a gamma camera. [image source: [54]]

The figure 1.1 shows a classical X-ray imaging technique. As depicted nicely in the diagram, the X-ray radiation travels from the generator (X-ray tube) through the body of the patient to a dual-layer detector, consisting of a flat panel detector, a cone beam collimator and a gamma camera. . After the attenuation of the radiation, that takes place while travelling through the body, the detector converts the radiation into a signal. This signal is the raw X-ray image.

CT Scan : CT scan [84, 68, 78], on the other hand, is a medical imaging technique that makes use of multiple X-ray images captured from several angles to produce cross-sectional images of a human body. Hence, there is a lot of information or in other words detailing in anatomy in a 3 Dimensional (3D) CT scan, compared to 2D X-rays [25], thereby making medical diagnosis much easier. It can be used to produce detailed images of many structures inside the body, like blood vessels, soft tissues and of course bones. Apart from detailing, it also provides physicians access into various views by simple manipulation of the data and without the need for additional imaging. Hence, it allows physicians to work with anatomy of concern and throw away the rest. So overall, CT scan is a much more powerful imaging technique compared to X-rays. But it is also much more expensive and not so easily available as the conventional X-rays. Moreover, exposure to radiation is considerably higher in CT scan compared to X-rays.

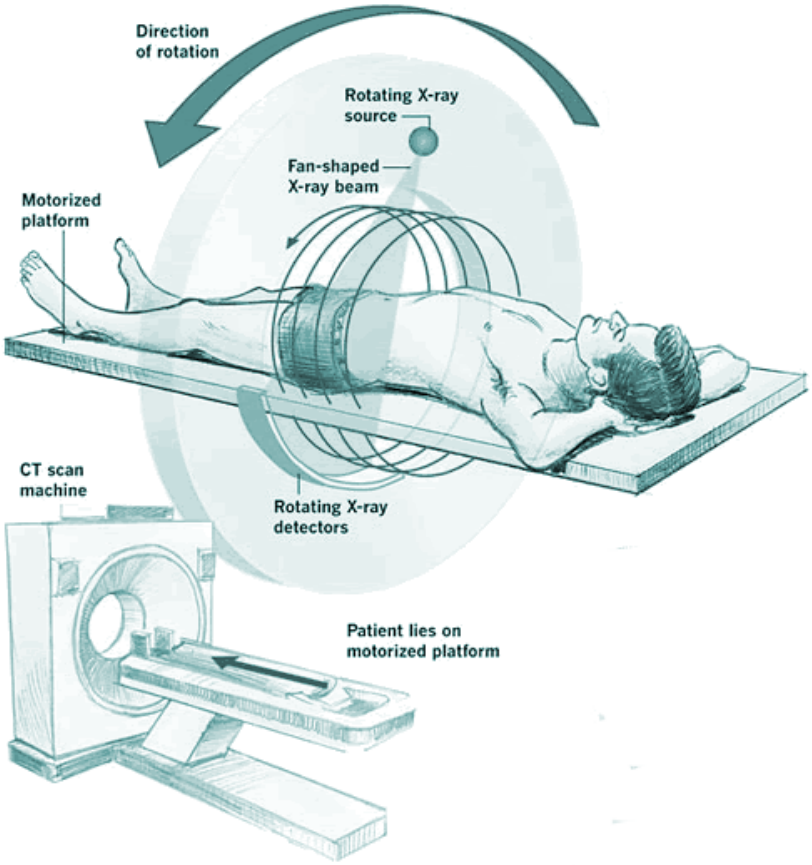


Figure 1.2.: CT scan imaging technique. Patient lies on a motorized platform, which slides inside the CT scan machine. The X-ray tube rotates around the patient and takes multiple images of very thin slices of the human body. After that, reconstruction method like filterd backprojection is applied to obtain a complete CT volume information of the anatomy of patient's body. [image source: [79]]

The figure 1.2 shows a typical CT scan imaging technique. The patient lies on a table that can slide through the opening into the CT scanner. When the patient enters into the scanner by virtue of the sliding table, the X-ray tube rotates around the patient and takes multiple images of very thin slices of the human body. After that, reconstruction method like filterd backprojection is applied to obtain a complete volume of slices, which captures the internal anatomical details of the body.
Having formed a basic idea of these two imaging modalities, let us try to understand the motivation behind the formulation of our thesis topic.

1.2. Motivation

This thesis aims at bringing the best from the two imaging techniques mentioned above. There are applications and scripts which are able to generate X-ray from CT scan and not the other way round. Given the easy availability of X-ray imaging and the detailed analytical capability of CT scan imaging, the thesis is premised on a novel idea of generating 3D CT scan volume from 2D X-ray images by making use of Deep Learning algorithms.

Deep Learning [11, 44, 56, 75] is a sub-class of Machine Learning algorithms. From Sentiment Analysis [83] to Anomaly Detection [15] and from Online Fraud Detection [82] to Autonomous Vehicles [30], Machine Learning has entered every sector of human lives and has made life easier. Similarly, Deep Learning has made immense advancements in many areas of computer vision including object detection [110], image classification [112], image segmentation [66] and many others. This has been made possible due to the easy availability of labeled data of common world objects, which do not need expert labeling. But Deep Learning is yet to make similar progress in the field of medical imaging [62] due to the lack of labeled data, which unlike many other domains of computer vision, needs expert labeling.

1.3. Thesis Goal

The final goal of the thesis is to apply Deep learning algorithms to enable a computer to construct a 3D CT volume of slices from a single X-ray image of a patient, thereby saving the expenses for a CT scan and the need to be subject to a lot more radiation and overcoming the problem of less availability of CT scan machines.

1.4. Thesis Structure

The report consists of 5 more chapters after this current chapter of Introduction.

Chapter 2 (Background) : This chapter starts with explaining the relation between Artificial Intelligence, Machine Learning and Deep Learning in section 2.1 and then unfolding different learning methods in section 2.2. It then points out the similarities between biological and artificial neurons in section 2.3. After giving an overview of broader types of neural networks in section 2.4, it dives straight into explaining the detailed architecture of UNet in 2.5, whose different versions have been used all through the thesis work, and then explains the basic concepts of transfer learning in section 2.6 and how it fits into our project. While training a neural network, bias, variance, overfitting and underfitting are the very basic concepts that everyone should be accustomed to and have been delineated in next section 2.7. Afterwards, it narrates the complete process of optimisation of weights of neural networks during the process of training in section 2.8. Finally, it gives an overview of the already existing related work in section 2.9, which was used as a cornerstone for further research for this thesis.

Chapter 3 (Experiments) : This chapter explains the complete experimental setup used for the whole work of the project. That includes the different network architectures used for different setups and their corresponding experiments in section 3.3, list of 20 experiments, conducted based on 16 different parameters and 4 other experiments, as a part of subsequent evaluations in section 3.2, general setup, namely remote connection, IDE, Python framework and system GPU, used for all experiments in section 3.1, dataset preparation method for training, which includes raw data and dataset acquisition method from same, pre-processing of the acquired dataset and finally the dataset splitting in section 3.4 and finally the deep learning setup in section 3.5, which entails discussions about the most important hyperparameters, batch size and epochs, regularisation, optimiser and loss functions and accuracy metrics used for the process of training.

Chapter 4 (Results and Discussion) : 16 different parameters were tested in 16 different stages through 24 different experiments, including a re-run, so as to provide to the readers a nicely explained comparative analysis of impact of different parameters in the training process. Additionally, 3 different experiments were conducted in the section of subsequent evaluations so as to provide an insight into performance of the complete setup in similar reconstruction tasks. So this chapter presents the learning curves of the trained networks from each of these 27 experiments, their corresponding discussions providing the analysis for each stage and phase and conclusion for each describing what we learnt from each and scopes of betterment for same. Alongside, it presents the performance of each of the networks in generating output 3D CT scan slices on test data of 2D Digitally Reconstructed Radiograph (DRR)s. Finally, it analyses the performance of the networks on real world X-rays in the last (28th) experiment.

Chapter 5 (Conclusion and Future Work) : This chapter concludes the report outlining the final achievement with respect to our set goal in section 5.1 along with explanation of possible future work in section 5.2 that can be taken forward from here to fill in the gaps between that set goal and our best accomplishment.

Chapter Appendix (Phases of Failed Implementation) : This chapter takes the reader into a journey of a sequence of failed implementations, which formed the basis for Research and Development and also acted as a cornerstone for laying the foundation of the final phase of successful implementation. Each and every phase has a new story to tell. Right from setting the initial objective set in the first phase discussed in section A.1, to it's evolution in section A.2, from analysing the reasons of complete failure of the network in section A.2 to spotting the scope of improvement in section A.3, from betterment of results in section A.4 to trying out data augmentation techniques in section A.5 and finally looking out for different methods of dataset pre-processing as well as relatively different network architectures through phases 6-10 in sections A.6 - A.10, this chapter story tells you the complete course of the unsuccessful implementations in the beginning.

2

Background

This chapter starts with introducing the concepts of Artificial Intelligence, Machine Learning and Deep Learning and the different types of learning methods. Then it explains the relation between biological and artificial neurons. Following that, it gives an overview of the two broader categories of already existing neural networks and discusses in detail the architecture of U-net [86] along with outlining the basics of transfer learning. After that, it explains the concepts of bias, variance, overfitting, underfitting and optimisation in neural networks. Eventually it presents an overview of a related work in this field, the drawback associated with it and the way our work counters that drawback.

2.1. Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence (AI) : It [87, 75] refers to any technique that enables machines to imitate human behavior, including learning, reasoning and self-correction.

Application : It [6] can be used for daily weather forecast.

Machine Learning (ML) : It [24, 67] is a subset of AI techniques which uses statistical methods to enable machines to improve with past experience and without the need for human intervention.

Application : It can be used to give recommendations [9] to online users of applications like YouTube based on their past choices.

Deep Learning (DL) : It [44, 39] is a subset of ML in which multi-layered neural networks learn to extract features from vast amount of data.

Application : It can be used in self-driving cars [13], which will then detect congestion and reroute automatically.

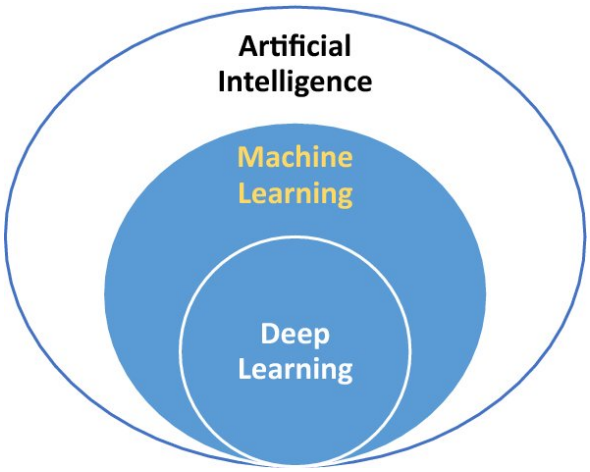


Figure 2.1.: Relation among Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL). DL is a subset of ML and it extracts features from data by virtue of multi-layered neural networks. ML is a subset of AI techniques and it enables machines to improve from past experience by use of statistical methods. AI provides solutions to machines to imitate human behaviour. [image source: [22]]

The figure 2.1 shows the interrelation among Artificial Intelligence, Machine Learning and Deep Learning [100] [75].

The branch that is of concern for us in this thesis is the area of Deep Learning as deep neural networks have been used for the task of reconstruction.

2.2. Types of Learning Methods

Machine Learning and subsequently Deep Learning methods can be broadly divided into three different categories:

Supervised learning : [20] Supervised learning mechanism tries to finds a mapping between inputs and labels based on a training dataset, and then makes predictions on inputs from test dataset, which is not a part of the training process.

Unsupervised learning : Unsupervised learning [14] is inspired by the brain's ability to extract patterns and recognise complex visual scenes and sounds. It has roots in neuroscience and is based on information theory and statistics. An unsupervised learner never receives any feedback from model at all. It only responds to the received inputs.

Reinforcement learning : Inspired by animal and human learning, the reinforcement learning [97] phase enables an agent to learn a mapping from states to actions by trial and error so that the expected cumulative reward in the future is maximised and the cumulative risk gets minimised.

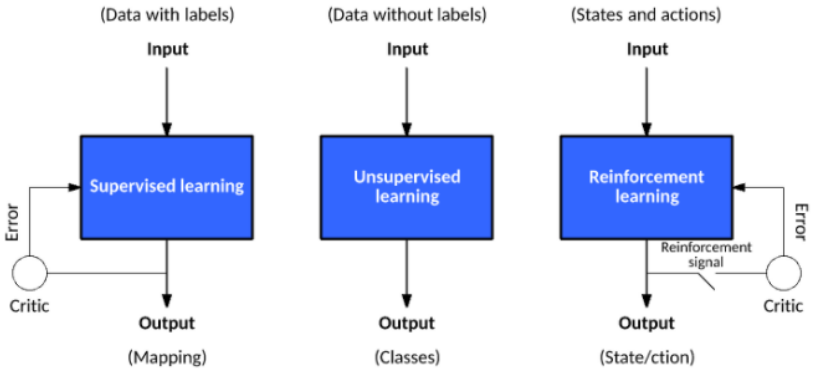


Figure 2.2.: Supervised (left), unsupervised (middle) and reinforcement learning (right). Supervised learning finds a mapping between inputs and labels from training dataset and makes predictions on inputs from test dataset. Unsupervised learning enables the machine to figure out the unknown patterns in the data without being told. Reinforcement learning learns a mapping from states to actions by trial and error to discover the actions that yield the greatest rewards. [image source: [59]]

The figure 2.2 depicts the basic workflow for these three different types of Machine Learning/Deep Learning methods discussed above.

The type of learning algorithm which has been used for this thesis is Supervised Deep Learning as we have a dataset containing paired inputs and labels.

2.3. Biological and Artificial Neurons

Neurons [12] are the fundamental units of brain and nervous system. Artificial Neurons [49] draw inspiration from Biological Neurons. This section shows the striking resemblance between two of them.

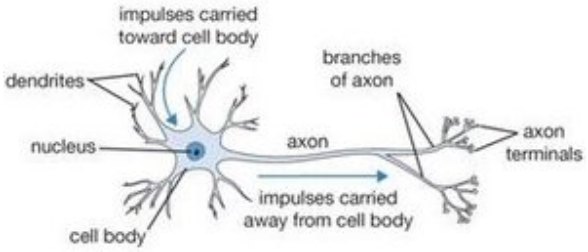


Figure 2.3.: Biological neuron. Dendrites receive input signal. Nucleus processes the received input signal and decides activation of a neuron. Axon passes on the information to the concerned cell. Axon terminals act as interconnection between the axon and dendrites of other neurons. [image source: [69]]

Figure 2.3 shows a Biological Neuron, consisting of dendrites, which is in charge of receiving input signal, nucleus, which processes the received input signal and decides whether a neuron should be activated or not, axon, which is responsible for passing on the information to the concerned cell and finally axon terminals or synapses, which acts as as interconnection between the axon and dendrites of other neurons.

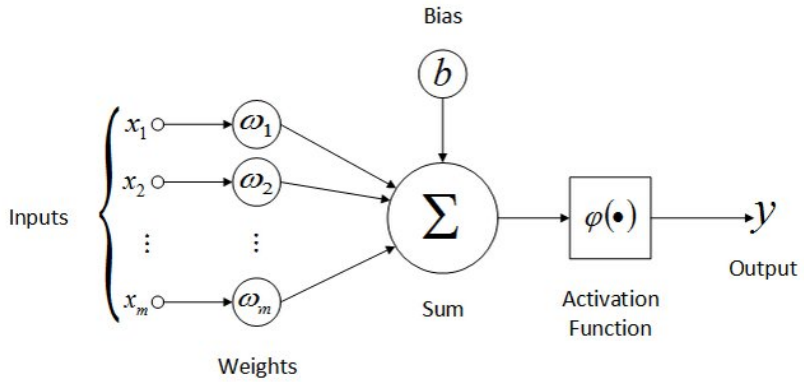


Figure 2.4.: Artificial Neuron of a Perceptron. The inputs (i.e., x_1 to x_n) are incoming signals from other neurons. The weights (i.e., w_1 to w_n) are used for weighing the received input values. Afterwards, the summation function is employed to add up the weighted inputs along with the bias b and, finally, an activation function ϕ, which decides whether a neuron should be activated or not, is employed to receive the final output, which can be then passed on to another neuron in the neural network. [image source: [23]]

Figure 2.4 shows a Artificial Neuron of a Perceptron, consisting of inputs, which receive incoming values to a neuron, weights, used for weighing the received input values, summation function, which adds up the weighted inputs along with the bias, an activation function, which decides whether a neuron should be activated or not and finally output, which can be then passed on to another neuron in the neural network. Hence the output of an artificial neuron can be modelled by the equation 2.1.

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b)[23] \tag{2.1}$$

where x_i are the inputs, w_i are the weights, n is the number of inputs, ϕ is the activation function and b is the bias, which are discussed later in detail in this chapter.

Hence dendrites, nucleus, axon and axon terminals (synapses) of a biological neuron can be equated to input, node, output and interconnections of an artificial neuron respectively.

2.4. Types of Neural Networks

There are two broad types of neural networks [108] based on how information is fed and processed across the network in between input and output nodes:

Convolutional Neural Network (CNN) : It [76] is a multi-layer feed-forward network, designed to recognise features from a 2D image data. It is a derivation from multi-layer perceptrons, consisting of convolutional layers. These convolutional layers can be used to extract more and more different kinds of features as the image moves through the network. In simple words, edges are detected in first layer and simple shapes in second layer and so on.

Application : It can be used in Image Recognition [16] and Object Detection [34].

Recurrent Neural Network (RNN) : It [91] is a class of neural networks which not only has´the feed-forward connections like the two variants described above, but also has recurrent or feed-back connections. These recurrent connections are used to feed the information of the output node back into the network. In other words, the output of an RNN is dependant not only on the current input but also on the last output, thereby turning each node into a kind of memory storage.

Application : It can be used in Speech Recognition [40] and Machine Translation [63].

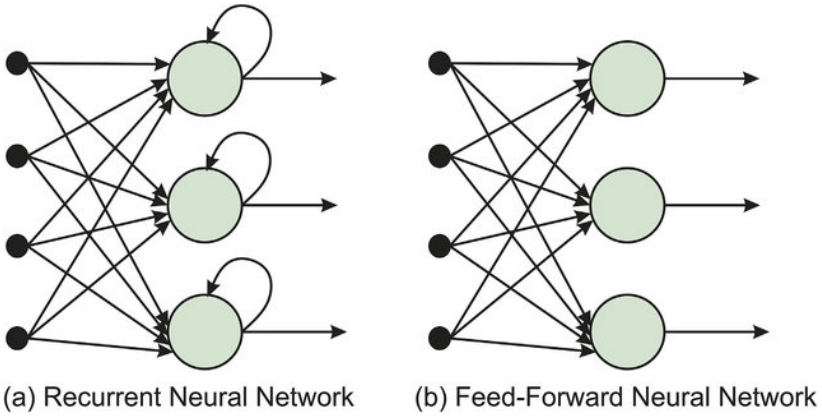


Figure 2.5.: Recurrent (left) vs. feed-forward (right) neural network. Recurrent Neural Network has connections feeding output back into the network, apart from input to output connections whereas Feed-Forward Neural Network has only uni-directional connections from input to output. [image source: [29]]

Figure 2.5 shows the clear difference between CNN (Feed Forward Neural Network) and RNN (Recurrent Neural Networks). In a Feed-Forward Neural Network, connections are uni-directional from input to output, whereas in a Recurrent Neural Network, apart from the input to output connections, there are connections feeding output back into the network.

Our task uses a network, which is a variation of the CNN and it is called U-Net, which is described in detail in the next section.

2.5. U-Net - Architecture Details

A typical Convolutional Neural Network usually consists of a flatten layer and one or many fully-connected layers apart from convolutional, pooling and activation layers. U-Net [86] is a derivation of CNN, consisting of only convolutional layers. Hence it is also referred to as a fully convolutional network. It was primarily modelled for segmentation of biomedical images. It can be used for reconstruction purpose too.

It consists of a contracting path, also called the encoder, and an expanding path, also known as the decoder, which are quite symmetrical to each other, thereby giving it a “U” like shape. Hence the name of the network follows from the architecture of the same, which looks like a “U”.

The encoder part consists of repetition of convolutions, with activation after each convolution and

following pooling operations. The encoder extracts spatial information from an input 2D image and keeps passing it through it's subsequent layers. As the spatial information moves through the encoder, it keeps getting reduced in resolution through max-pooling and the number of channels keep increasing. The information is finally stored in the latent space, with reduced resolution and increased channels, of the encoder.

The symmetrical decoder part similarly consists of repetition of convolutions, activation and up-sampling operations, in place of pooling, and finally skip connections, arriving from the encoder part. The decoder passes the spatial information, available in the latent space, through it's succeeding layers. As the spatial information passes through the decoder, it keeps increasing in resolution through bilinear up-sampling and also gets concatenated with similar high resolution features, made available by skip connections from the encoder part, till it completes the reconstruction of the original or segmented version of original image.

The next section will describe in detail every aspect related to the architecture, functionalities and training of the network.

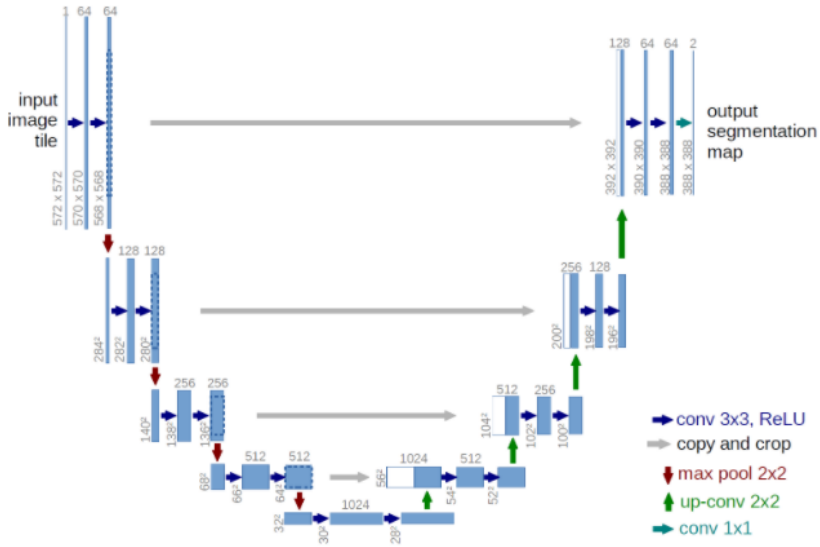


Figure 2.6.: U-Net architecture. It consists of a contracting path, the encoder, indicated by decrease in image dimension in stages, and an expanding path, the decoder, depicted by decrease in image dimension, with a dropout in between. The encoder extracts spatial information from an input 2D image and stores in the latent space, with reduced resolution and increased channel dimension. The symmetrical decoder uses the spatial information, available in the latent space, and reconstructs the original image or a requirement-specific version (e.g., segmented) version of the image. There are also skip connections, originating from the encoder and terminating in the decoder. They help to concatenate higher resolution features from the encoder with the same at the decoder. [image source: [86]]

Figure 2.6 shows a classical U-Net architecture consisting of encoder, decoder and skip connections.

Let's dig a bit deep into details of architecture of U-Net. We will discuss about each and every layer which makes up the U-Net architecture.

2.5.1. Convolution Layer

Since U-Net is a variant of CNN and is also referred to as a fully convolutional [107] network, so the most important component of a U-Net architecture is the convolution layer.

The convolution layer [4] is used to extract features from a 2D input image and it achieves that by making use of what is known as convolution kernels. Each of these kernels at a particular convolution layer is tailored in a way that it is used to identify the presence or absence of one particular feature in the selected section of the image. This process of feature extraction is achieved by a convolution operation between each of the kernels and the region of interest of the image, which in turn condenses the image into feature maps by throwing away the redundant data. As the input image moves through the different convolution layers in the network, the features which keep getting detected also move from lower order to higher. In the first convolution layer, simpler or low-level features like edges will get detected. It is inspired from the fact that a single pixel in a 2D image has local correlation with all of it's neighbouring pixels. In the next convolution layer, mid-level features like shapes can be detected. Similarly, high-level features like objects will be possible to extract in higher layers of convolution. This follows from the fact that later part of the network learns the global correlation and sees the bigger picture of the image.

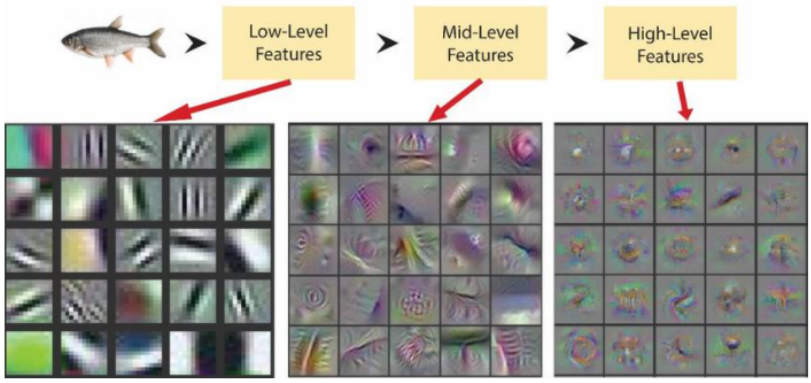


Figure 2.7.: Feature extraction by convolution layers in stages. In the first convolution layer, low-level features like edges might get detected. In the next layer, mid-level features like shapes can be detected. Similarly, in higher layers, high-level features like objects will be possible to get extracted. [image source: [94]]

Figure 2.7 depicts stage-wise extraction of features from lower to higher order from an input image.

Mathematically a 2D convolution is represented by equation 2.2.

$$y[i,j] = \sum_{n}\sum_{m}[m,n] \cdot x[i-m, j-n][28,\ 4] \tag{2.2}$$

where h is the convolution kernel, x is the input image, and the output y is the condensed feature map. The process of convolution starts with element-wise multiplication of pixels between the image patch and the convolution kernel and then subsequent summation of these multiplied values, which forms a single pixel value in the feature map. Thereafter, the center of the convolution kernel moves exactly by those many pixels which is given by the value of stride. This computation is carried on until and

unless each and every pixel of the feature map has been computed. Size of the image can be altered from one layer to another with the value of stride and also with zero padding, which involves addition of border of pixels of zeros around the corners of the input image.

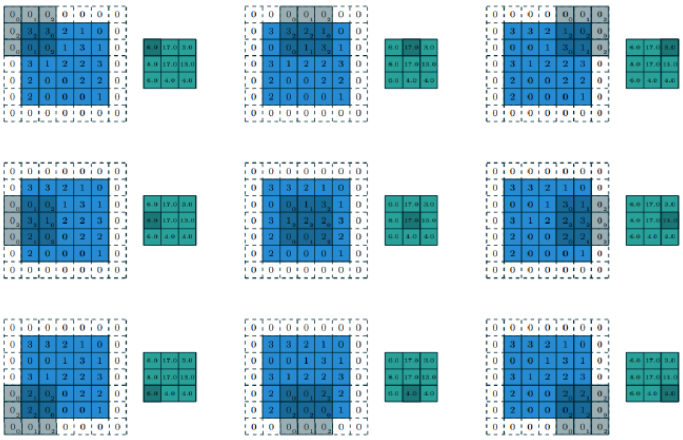


Figure 2.8.: Convolution operation. It shows a convolution operation between a convolution kernel and a 2D zero-padded input image with kernel size of 3×3 and stride of 2. [image source: [28]]

Figure 2.8 models a convolution operation between a convolution kernel and a 2D zero-padded input image with kernel size of 3×3 and stride of 2.

Different distribution of convolution layers have been used in the encoder and the decoder for different variants of U-Net architecture used for the thesis task.

2.5.2. Activation Function

The second most important unit of this neural network is the activation function [4, 32], which determines whether a neuron should be fired or not. It is used after every repetition of convolution and batch normalisation (discussed later in this section) layers throughout the network. It is used to introduce non-linear properties into the output of a neuron This non-linearity is essential for a neural network, as without it, the network will simply behave as a linear regression thereby reducing the learning power of the network and making it unsuitable to real world-complex situations. That explains the need for non-linear activation functions.

There are several kinds of non-linear activation functions which are in use for quite some time. But we will focus only on the four major ones. Those are:

Sigmoid : It [32] is one of the earliest activation functions, which maps linear input in the range of $[-\infty, +\infty]$ to non-linear output in the range of $[0, 1]$. With reference to the figure 2.9, in the input range of $[-2, 2]$, where the activation curve has a really steep gradient, network will perform good and learning will happen smoothly. But towards both ends of the function curve at $[-\infty, -2]$ and $[2, \infty]$, when the graph tends to saturate, the gradient reduces drastically making it tougher for the network

to learn anymore – a situation known as vanishing gradient, where the weights of the network stop getting updated anymore. The sigmoid function is represented by equation 2.3.

$$\phi(z) = \frac{1}{1 + e^{-z}}[32] \tag{2.3}$$

Tanh : Hyperbolic Tangent or Tanh [32] operates in a very similar way as the sigmoid activation function, except for mapping the output in the range of $[-1, 1]$. Hence it allows the derivative to be more steep, facilitating quicker learning, but the problem of flattened gradient at the saturation zone of the function curve still persists like the sigmoid function. The function and the plot for tanh is given by equation 2.4.

$$T(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}[32] \tag{2.4}$$

ReLu : Rectified Linear Unit or ReLu [32] is one of the most common or widely used activation functions in CNN. It maps input in the range of $[-\infty, \infty]$ to output in the range of $[0, \infty]$, thereby mapping any negative value to zero and retaining any positive value as it is. CNNs working with ReLu tend to converge much faster. But the drawback associated with ReLu is that if the input to a lot of activations get below zero, most of the neurons' output will be zero thereby making it hard for the network to learn because of improper mapping of negative values. The ReLu function is given by equation 2.5.

$$R(z) = max(0, z)[32] \tag{2.5}$$

Leaky ReLu : To address the dying ReLu problem, scientists came up with the leaky ReLu activation function [32], which allows a leak of 0.1, thereby increasing the output range of ReLu to $[-\infty, +\infty]$, but by keeping the gradient in the negative region really small. The Leaky ReLu function can be modelled by the equation 2.6.

$$L(z) = \begin{cases} 0.01, & \text{for } z < 0 \\ z, & \text{for } z \geq 0 \end{cases} [32] \tag{2.6}$$

The plots for these four activation functions are shown in figure 2.9.

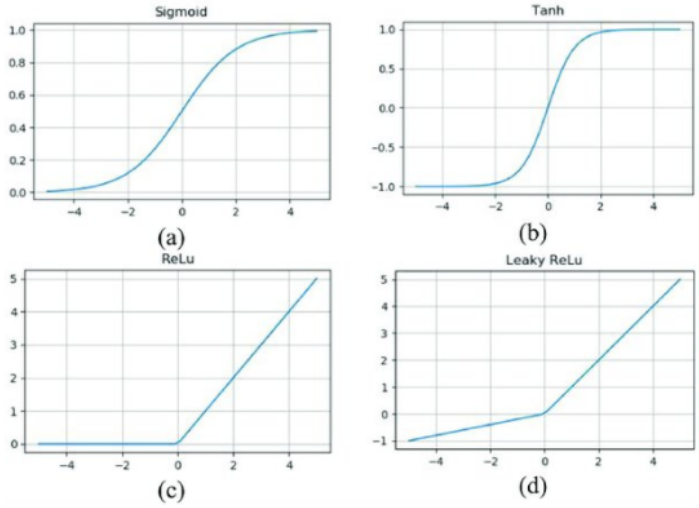


Figure 2.9.: Activation functions. (a) represents Sigmoid activation function. It maps linear input in the range of $[-\infty, +\infty]$ to non-linear output in the range of $[0, 1]$. (b) shows Tanh activation. It maps linear input in the range of $[-\infty, +\infty]$ to non-linear output in the range of $[-1, 1]$. (c) depicts ReLu function. It maps input in the range of $[-\infty, \infty]$ to output in the range of $[0, \infty]$. (d) is Leaky ReLu activation function. It allows a leak of 0.1, thereby increasing the output range of ReLu to $[-\infty, +\infty]$. [image source: [55]]

In our implementation ReLu has been used after every repeated combination of convolutional and normalization layers (wherever applicable) across different experimental setups.

In the final layer, sigmoid gave better results than ReLu, because the pixel values were already normalised between 0 and 1. Hence, with no negative values around, ReLu will not have any effect on the non-negative values. Also in few of the setups, Leaky ReLu has been used in the final layer.

2.5.3. Pooling Layer

Pooling layer [4] is used to reduce data dimensionality of each feature map, while retaining the most important information. This helps the network in two different ways. Firstly, with reduced volume of data, it becomes easy for the network to learns from the training data quickly, thereby reducing the training time. Secondly, it stops the network from learning too much from the known data so that it fails to generalise on the unseen data – a condition, known as overfitting. The reason for overfitting without a pooling layer is, since the feature maps capture the location of features in the input image very precisely, so it means even with a very little movement of features, like rotation, shifting or similar, will result in an altogether different feature map. This is broader sense means that the network will have a tough time generalising with a rotated version of the image although it has seen the unrotated version already. Pooling layer addresses this problem by extracting the summarised version of features from a feature map such that even with small changes in the position of the feature in the input, the pooled feature map will still have the feature in the same location. Pooling function is defined by a function, filter size and stride length, just like the convolutional layer. There are two different broader pooling techniques:

Max Pooling : It [19] only captures the strongest activation and disregards every other unit in the pooling patch of the feature map. In other words, it takes the maximum pixel value in the pooling region. It is represented by the equation 2.7.

$$f_{MaxPooling}(X) = max_{i,j}(X(i,j))[19] \tag{2.7}$$

Average Pooling : It [60] takes all activations in the pooling patch of the feature map into consideration with equal contribution. In other words, it takes the average value of all the pixels in the pooling region. It is represented by the equation 2.8.

$$f_{AveragePooling}(X) = \frac{1}{n+m}\sum_{n}\sum_{m}X(i,j) \tag{2.8}$$

In our task, Max Pooling has been used. The two different types of pooling has been diagrammatically explained in the figure 2.10 with a stride length of 2 and filter size of 2×2 pixel.

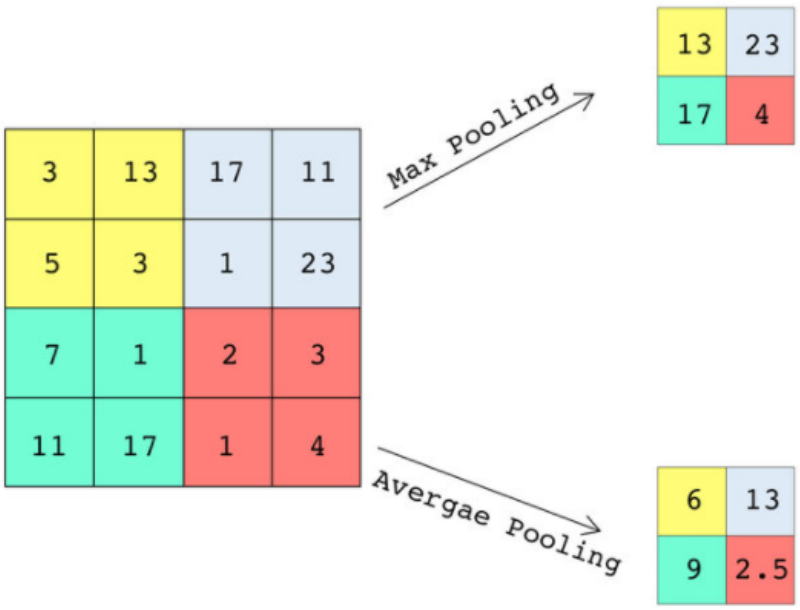


Figure 2.10.: Max and average pooling. Max pooling takes the maximum pixel value in the pooling region. Average pooling takes the average value of all the pixels in the pooling region. [image source: [5]]

2.5.4. Normalisation

Normalisation [21] is a technique in which data arriving from different sources and having entirely different range is standardised into a similar range before being fed to the network, such that every

feature for a single data point and for the whole dataset has an equal contribution. Not only that, it also brings in a bit of regularisation in the network and helps in the reduction of internal covariance shift resulting from change of network parameters. All of these make the training process easier and facilitate quicker convergence. Apart from the data pre-processing, which is explained in detail in the Data Pre-processing subsection of Experimental Setup, there are other normalisation techniques which are applied to data in between layers of neural network. In our task, three different types of normalisation [89] have been used:

Batch Normalisation : The most common form of normalisation technique used in between layers of neural network is batch normalisation [60, 89]. In order to avoid costly calculations of covariance matrices to decorralate and whiten the data at every layer, batch normalisation normalizes the distribution of each input feature in each layer across each mini-batch to have zero mean and standard deviation of one. Additionally it tries to learn a scale factor and a bias value in order to find the necessary covariance shift. Batch Normalisation allows flexibility to choosing learning rate and weights initialisation but it becomes apparently useless for a batch size of 1 because of zero variance.

Layer Normalisation : It [10, 89] is a relatively newer technique than batch normalisation. On contrary to normalization of input features across the whole batch, layer normalisation does normalisation of the input across the features.

Group Normalisation : Group Normalization [105, 89] is implemented along the direction of features but unlike in layer normalisation, it splits up the features into different groups and then does the normalisation over each group separately.

In our task, across different network architectures, datasets and system configurations, no normalisation, batch normalisation, layer normalisation and group normalisation have been used.

2.5.5. Skip Connections

An important aspect of U-Net architecture are the skip connections[26] [27] originating at different layers of the encoder and ending up in the decoder at equivalent layers. It helps the network in two different ways:

- It helps to capture the spatial information from the early layers of the encoder without any degradation and pass it on to the decoder to facilitate the reconstruction process during upsampling, while taking into consideration minute details from the early layers.

- It helps gradient information from the encoder, which might have been lost in subsequent layers due to vanishing gradient problem, travel faster through the deep U-Net and this helps in reduction of convergence time.

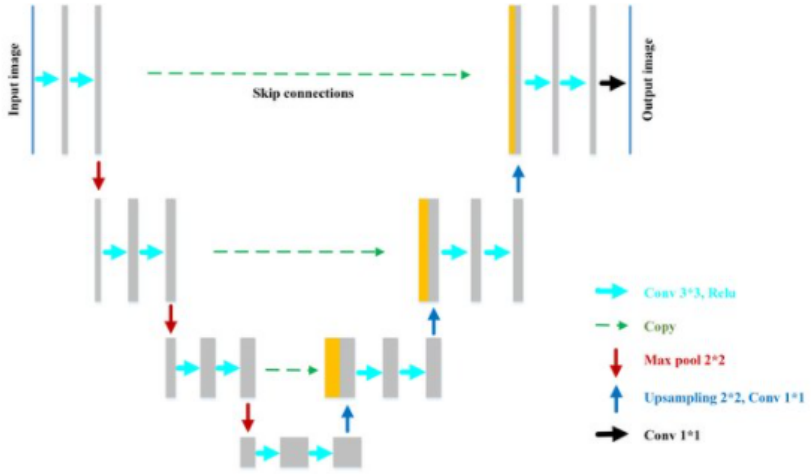


Figure 2.11.: UNet skip connections. They capture the spatial information from the early layers of the encoder and pass them on to the later layers of decoder to facilitate the reconstruction process. [image source: [96]]

Figure 2.11 shows skip connections architecture in U-Net.

2.6. Transfer learning

There are three practical problems associated with training a deep neural network, built from scratch. Firstly, if we have a large dataset, it might take many days and even weeks to train the network even with high end system configuration. After training for such a long time, if the results are not up to the mark or if the training encounters any sort of error, then the whole effort, in terms of investment of time and resources, is wasted. Secondly, in all domains or in every application, there is not always abundance of labelled dataset. That is a great challenge for the purpose of training too as deep neural network is extremely data hungry and it's performance goes up with more data being fed into the network. This is because with more and more seen examples, the network fails to fit perfectly to the seen huge and complex data and is forced to start generalising, thereby yielding good results on unseen data. Thirdly, even with data and everything else in place, it might be challenging to fine tune a network to the specific task, while taking into consideration all kinds of parameters and hyperparameters and network architecture.

Transfer Learning[11, 98] aims to counter all three of the problems in one go. As the name suggests, it is basically transferring the knowledge that a network learnt from a domain for a specific task to make predictions for another or similar domain for a different task. Transfer learning entails pre-trained deep neural networks. These neural networks have been trained by experts over a long period of time in different phases for different or likely domains. Hence the weights are already tuned to the vast amount of data that the network has seen as a result of which the network is expected to have learnt generalisation within the likely domains. So in order to fine tune this pre-trained network to a different task in a different domain, it does not need a really big dataset. It can perform well on a small dataset as well because as explained above, it has already been exposed to huge amount of data to which it has aligned it's weights. So the small dataset, that it needs to make predictions on, is to fine tune the network to that particular task. With transfer learning, usually all layers are kept

frozen except a few first ones, in case there is a difference in input channels for input data, and a few last ones, which are used for fine tuning to the new dataset and also to output the required number of channels as needed for the task. So with most of the layers being frozen and with the need to train only a few layers on a much smaller dataset, it overcomes the need to have high end system configuration and the training is much faster. Finally since the network has been mostly optimised in terms of it's architecture or in terms of parameters there is very little to experiment in those regard and people can achieve great results with very little effort. Hence this process of transferring the learning of a highly optimised network, built and trained by experts, thereby outdoing the need for large labelled dataset or high end system configuration for the purpose of achieving good results in short time is known as transfer learning.

In our task, we have explored the scope of transfer learning, but to little success and the reasons are mentioned in section 3.9.

2.7. Bias, Variance, Overfitting and Underfitting

Before getting into the concept of overfitting and underfitting, it is important to understand the concept of bias, variance and the tradeoff [70] between them.

Bias : It [74] is basically the difference between the output predicted by a network and the output that is expected of it. If a network has high bias, it means it is oversimplified and it fails to fit to the training data well.

Variance : It [46] happens when the network fails to generalise on the data that it has not seen before due to the network being over-complicated as a result of which it learns even the noise from the known training data.

Bias-Variance tradeoff, overfitting and underfitting : Figure 2.12 represents perfectly the trade-off [70] between bias and variance.

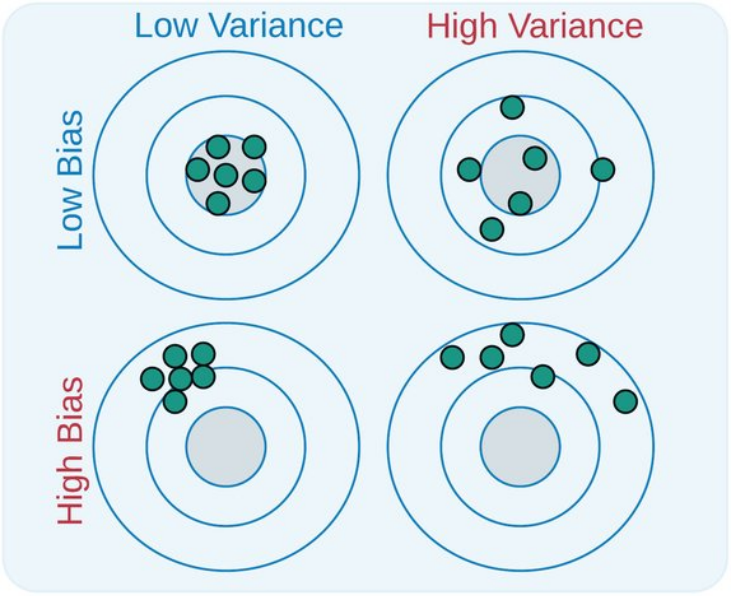


Figure 2.12.: Bias variance trade-off. A combination of High Bias and Low Variance leads to a situation called underfitting. A combination of Low Bias and High Variance is referred to as overfitting. A combination of Low Bias and Low Variance is a good fit, a situation in which network predictions lie in the bull's eye. [image source: [41]]

The center of the target is the region where the network does a good job with prediction and it gets worse as we move away from the center.

A combination of High Bias and Low Variance leads to a situation called underfitting in which the network performs bad even on the training data. This can happen due to oversimplified model or less data as a result of which the network fails to learn anything.

A combination of Low Bias and High Variance is referred to as overfitting, in which the network performs good on known data but fails to do so on unseen data. This happens due to a really big network, with lots of parameters, which has learnt also the noise from known data.

A combination of Low Bias and Low Variance is a good fit and it is the situation when the network predictions is in the bull's eye.

2.7.1. Tackle Underfitting

The ways of countering underfitting [2, 73, 36] problem in neural network area as follows:

Training Data : If the network is not able to learn [36] properly even from the known training dataset and the hypothesis curve fails to generate a good fit for being forced to generalise, due to the

large size of the dataset having vast amount of complex data, so removing a certain amount of training data (preferably redundant ones) helps to deal with underfitting.

Network Complexity : Since one [36] of the reasons for underfitting is an oversimplified network, increasing the model complexity helps to address the problem. This can be done either by increasing the number of channels for the hidden layers or by increasing the number of hidden layers itself.

Training Time : Simply increasing the training time [36] or the number of epochs will also help in this situation. In our task, underfitting was countered by increasing the training time and by removing the redundant examples present in the dataset. The network complexity was set at the maximum limit which the system GPU can handle. It was still underfitting due to system configuration constraints and the level of complexity entailed by a highly complex dimensionality of $256 \times 256 \times 256$ pixels volume data, as a result of which, we spent a longer time on training for every experiment in every phase, which reduced considerably though after pruning of the dataset.

2.7.2. Tackle Overfitting

The ways [48] to mitigate overfitting are as follows:

Training Data : When the dataset is small, there is a high probability of the network to tune it's weights completely to the dataset, even to it's specifics and thereby move away from the capability of generalising. So an usual and easy apporach to overcoming overfitting is to provide more and more data to the network. When the network has to deal with considerably larger amount of data, it fails to overfit ny more and starts generalising.

Network Complexity : It [36] is exactly opposite to handling underfitting. If the network is too much complex with a lot of trainable parameter, then those huge number of parameters learn each and every detail of the training data, even the noise and hence fails to generalise. So simply making the network less complex may solve the problem.

Early Stopping : [80] In a situation when the network's training accuracy still keeps increasing but the test (or validation) accuracy starts to decrease because the network is tuning in more to the specifics of the training data, thereby moving away from the capability of generalisation, it makes no sense to keep training the data. So the phenomenon of stopping training at this point is known as early stopping. We have used a similar variant of early stopping and it is called early saving. This is explained in detail in section 3.11.

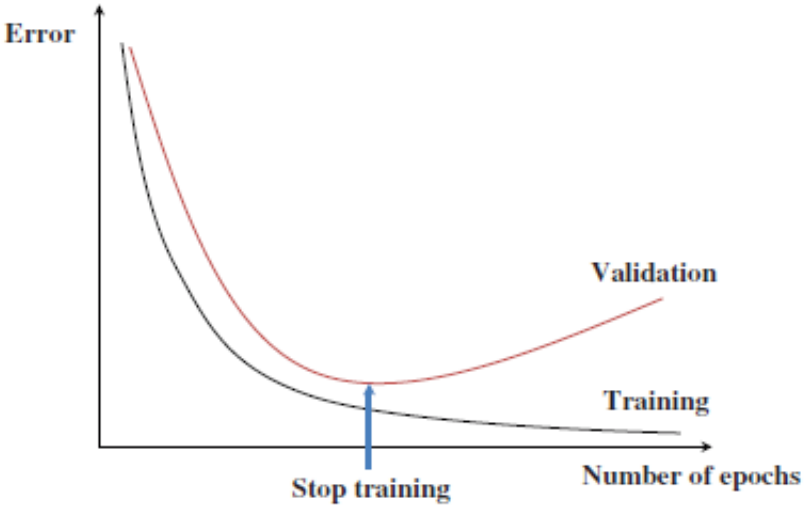


Figure 2.13.: Early stopping. It is a situation when validation error starts to increase, but training error keeps on decreasing. This happens when the neural network has started to overfit to training data and has moved away from generalisation. [image source: [92]]

Figure 2.13 shows the process of stopping training at early stopping point.

Regularisation : Regularisation [71] is a set of techniques which overcomes the problem of overfitting by penalising the weight matrices of the neurons because a network with large network weights can be really unstable, where small changes in the input can lead to large changes in the output. There are three different types of regularisation techniques.

L1 – L1 Regularisation[88, 45] or Lasso Regression adds a weighted summation over absolute values of coefficients as a penalty to the loss function. It is given by:

$$.Loss_{L1} = Loss_{normal} + \lambda * \sum ||w||[88] \tag{2.9}$$

where the weighted term λ is the regularisation parameter

L2 – L2 Regularisation [99] or Ridge Regression adds a weighted summation over squared values of coefficients as a penalty to the loss function. It is given by:

$$.Loss_{L2} = Loss_{normal} + \lambda * \sum ||w||^2[88] \tag{2.10}$$

Dropout – It [95] is a very simple yet powerful regularisation technique. It drops out neurons in an absolute randomised fashion during training and weights, associated with these dropped neurons, do not get updated during back propagation, a concept which is explained later in this section. The effect of dropout on a neural network is explained in the figure 2.14.

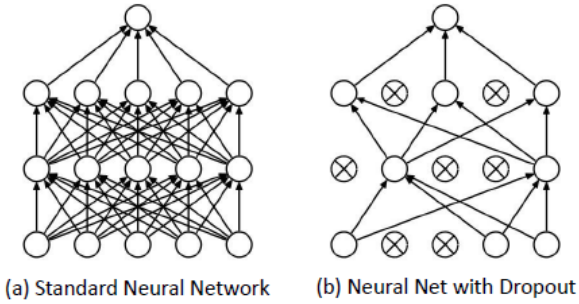


Figure 2.14.: Neural network dropout. Left (a) shows a standard neural network. Right (b) shows the same with dropout functionality. It drops tout neurons in a randomised fashion during training and weights, associated with these dropped neurons, do not get updated during back propagation. [image source: [85]]

In our task, we have used L2 regularisation and a dropout layer right after the encoder to tackle overfitting.

Data Augmentation : Increasing the size of the dataset can also be one of the ways to deal with overfitting. In simpler terms, more and more data the network sees, more is the probability that the network has seen a lot of variance already in the training data and less in the chance that it faces a high variance in the unseen data. In technical terms, if there is more data, the network will fail to fit into the specifics of it and it will start generalising more. When there is a shortage of data, the most popular technique which is used to virtually increase the dataset by adding modified versions of copies of existing data is data augmentation [93]. The most popular data augmentation techniques are rotation, shifting, scaling etc. It is described more in detail in sections 3.5 and 4.6.

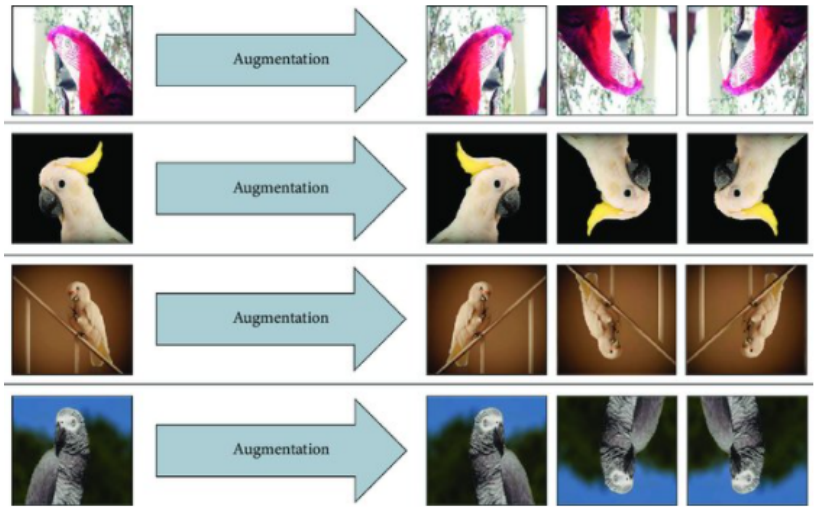


Figure 2.15.: Data Augmentation. It virtually increases the size of the dataset by adding modified versions of copies of existing data, when the existing dataset is small. The most popular data augmentation techniques are rotation, shifting and scaling. [image source: [17]]

Figure 2.15 shows data augmented images of a endangered parrot species.

Overfitting was addressed by all the methods discussed above, except for reducing model complexity, as it was already a situation of underfitting.

2.8. Optimisation

Before going deep into optimisation strategies in a neural network, it is important to know the two basic terminologies, loss function and accuracy metric.

Loss Function : The purpose of loss function [50] is to calculate the difference between the predicted output of the neural network and the supposed actual output (labels). There are different kinds of loss functions which are is use like cross-entropy or mean squared error. The four different kinds of loss functions which have been used in our task are decomposition loss (L_D), reconstruction loss (L_R), latent space loss (L_L) and total loss (L). Those can be explained by the equations 2.11 and 2.12.

$$L_D = \frac{1}{N}(\lambda_d \sum_{i=0}^{N-1}(\hat{y}_i - y_i)^2 + (1 - \lambda_d)\sum_{i=0}^{N-1}|\hat{y}_i - y_i|) \tag{2.11}$$

$$L_R = L_L = \frac{1}{N}\sum_{i=0}^{N-1}(\hat{y}_i - y_i)^2 \tag{2.12}$$

where N is the number of total number of pixels in a 2D image and λ_d is the trade-off factor which is altered between 0.1 and 0.9. L_D or the decomposition loss represents the loss between the 3D volume, decomposed by the network and their corresponding labels. L_R or the reconstruction loss is the loss between the 2D DRR, reconstructed from 3D CT volume, produced by the network and the input to the network. L_L or the latent space loss is the loss in the output of the latent space right after the encoder between the network, which is supposed to generate 3D CT volume from 2D DRR image and the network which was pre-trained for reconstructing 3D CT volume.

There are three different variants of loss functions which have been used in different setups and they are given by equations 2.13, 2.14 and 2.15.

$$L = L_D \tag{2.13}$$

which represents the total loss for the network which used only the decomposition perspective.

$$L = L_D + \lambda_r * L_R \tag{2.14}$$

gives the total loss for the network which also performed the reconstruction part other than the decomposition bit. Here λ_r is a regularisation parameter and it's value is set to 0.5.

$$L = L_D + L_R + L_L \tag{2.15}$$

This version of total loss was used, when all the factors were taken into consideration.

Accuracy Metrics : The purpose of the accuracy metric is to find out how good the network is performing in terms of generating the output when compared to the expected output. Our task used two

different accuracy metrics, Peak Signal-to-Noise Ratio (PSNR)[101, 31, 47] and Structural Similarity Index Measure (SSIM) [102, 72], which are explained in equations 2.16 and 2.19.

$$PSNR = 10 * \log_{10}(\frac{I^2}{MSE}) \tag{2.16}$$

where I is the maximum value of a pixel (1 for our task) and MSE is mean square error and is given by equation 2.17.

$$MSE = \frac{1}{m * n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [A(i,j) - B(i,j)]^2 \tag{2.17}$$

where m and n are the dimensions of a 2D image.

$$SSIM = \frac{(2 * \mu_x * \mu_y + c_1) * (2 * \sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_2) * (\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.18}$$

where x and y are two different 2D images, μ_x and μ_y represent average values of pixels for x and y respectively, σ_x^2 and σ_y^2 are variances of same, σ_{xy} is covariance between x and y. c_1 and c_2 are stabilising factors and are given by $c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$, where k_1 and k_2 are set to default values of 0.01 and 0.03 respectively and L is the dynamic range of pixel values.
So the final objective of the neural network is to minimise the loss value in loss function to an optimal minimum such that the accuracy metric reaches it's maximum achievable value after having found the optimised value of weights.

The optimisation is an iterative loop, which consists of three distinct steps:

Forward Pass : It [57] is the process of calculating the output of a network by taking into consideration all the weights associated with all the nodes in the network from the input layer through the hidden layers to the output layer. In simple terms, it is represented by equation 2.19

$$y = f(W_L \cdot f(W_{L-1} \cdot f(...(f(W_2 \cdot (f(W_1 \cdot x)))...) \tag{2.19}$$

where x and y are the input and output matrices respectively, f is the activation function for each layer and W_1 represents the the weight matrix for first layer, W_2 for the second layer and so on. The weight matrix for each layer consists of weights for each and every connection to the neurons for that particular layer and is represented by $w_{i,j}^l$ is the weight for the connection between i-th input and j-th neuron for layer l.

Backpropagation : [43, 81] In this stage, the gradient of the loss calculated between the output, generated after a forward pass, and the expected output (target), with respect to every parameter, is backpropagated in reverse direction across the network from the output to the input layer for each parameter. Equation 2.20 shows the gradient calculation with respect to loss function for each weight.

$$\nabla L = (\frac{\partial L}{\partial w_{1,1}^1}, \frac{\partial L}{\partial w_{1,2}^1}, \ldots, \frac{\partial L}{\partial w_{i,j}^l}) \tag{2.20}$$

where L is the loss function.

Weights Updation : In the last step the weight for each parameter get updated according to the rule mentioned in equation 2.21

$$w^l_{i,j_{t+1}} = w^l_{i,j_t} - \eta \nabla L \tag{2.21}$$

where $w^l_{i,j_{t+1}}$ and w^l_{i,j_t} represent the weights at time $t+1$ and t respectively, or in other words after and before updation and η is the learning rate.

This whole optimisation process of adjusting the gradient of the loss function to find a global minima (so as to have the most optimised weights of the parameters associated with the neural networks, thereby enabling the neural network to produce the most optimised output) is known as gradient descent. There are three different modes to it:

Gradient Descent :[111] This computes the gradient of the loss function with respect to parameters and also updates the parameters for entire training dataset. Although it is slow in computation, it has the problem of hitting a local minima and never getting out of it to steer towards the global minima.

Stochastic Gradient Descent (SGD): This method [1] performs gradient calculation and parameter updation for each training example. Hence this method is faster than batch gradient descent but it has a high probability to overshoot a global minima.

Mini Batch Gradient Descent : It [58] combines the best of the both worlds. Here gradient calculation and weights updation do not take place for a whole dataset or an individual training sample, but for a small batch from the training set. Although it performs better than the above two, choose an appropriate learning rate and batch size can be tricky.

Learning Rate : Learning rate [38] is a hyperparameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. Choosing the right learning rate is extremely essential for convergence. If the learning rate is too low, then it means that it will take a long time to find a global minima, thereby making convergence slower, and if it is chosen too high then, it might miss the global minima and never lead to convergence. Figure 2.16 shows a diagrammatic representation of effects of choosing too high, too low and just right learning rates.

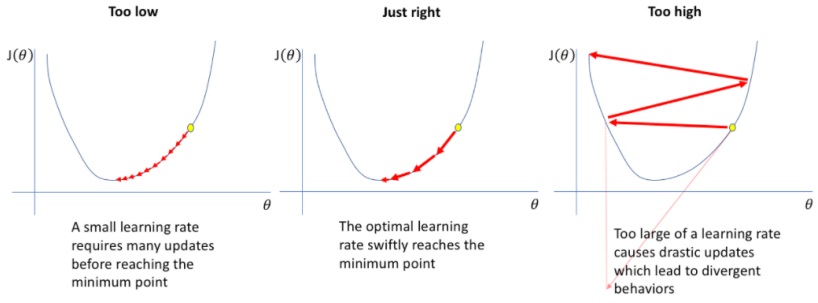


Figure 2.16.: Choice of learning rates. Too low learning rate can mean longer time to find the global minima. Too high learning rate might mean skipping the global minima every time. Just right learning rate will help in finding the minima swiftly. [image source: [109]]

Since the choice of just right learning rate can be tricky and completely dependant on a particular situation and also since the choice of a constant learning rate is not so wise, it would be really great if there is an algorithm which takes care of how the learning rate [37] should change as the training progresses. In other words, the algorithm should be able to assign a higher learning rate when it is far from the global minima and change it to a lower value when it is close to the minima. Here comes the role of learning rate scheduler [106]. There are different types of schedulers like time based decay, step based decay or exponential decay which can do the job. Apart from them, there are some adaptive optimisers which also adjust the learning rate according to situation. One of them is Adam or Adaptive Moment Estimation, which has been used for the task.

Adaptive Moment Estimation (Adam) : Adam [52] combines the adaptive learning rate strategy of AdaGrad and momentum strategy of RMSProp, both of which are also well-known optimisers. While the momentum aspect of it accelerates the search in the direction of global minima by damping the oscillations, adaptive learning rate aspect of it takes care of learning rate scheduling. The first and second moments of momentum are given by equation 2.22 and 2.24.

$$\hat{m_u} = \frac{m_u}{1 - {\beta_1}^u}[52] \tag{2.22}$$

$$\hat{v_u} = \frac{v_u}{1 - {\beta_2}^u} \tag{2.23}$$

Parameter update is done as shown in the equation 2.24.

$$\theta_{u+1} = \theta_u - \frac{\eta}{\sqrt{\hat{v_u}} + \epsilon}\hat{v_u} \tag{2.24}$$

where β_1 is 0.9 , β_2 is 0.999 and ϵ is a very small value.

In our experiments, we have majorly used Adam except in few cases, where SGD was also used.

2.9. Related Work

In this section, we will discuss about a similar work [90] that has already been done in this area.

Liyue Shen, Wei Zhao and Lei Xing from the Departments of Randiation Oncology and Electrical Engineering of Stanford University in USA published a research on a related topic in the year 2019, but with a twist. The name of his topic is 'Patient-specific reconstruction of volumetric computed tomography images from a single projection view via deep learning'.

For understanding what has already been done in their work, why it needs improvement and how it has been upgraded in this thesis, we will go briefly through the objective of their research, dataset acquisition process, network architecture, results and finally the drawback associated with the implementation.

2.9.1. Objective

For obtaining a flawless CT volume image, multiple projections from several angles are required so as to take care of the detailed anatomy of the inside of human body.

Their work aims to train a deep neural network in a way such that it learns the relationship between multiple DRRs (input) of a patient and their corresponding CT volume image (label), so that it can eventually predict the CT volume for the same patient from a single projection image in the evaluation phase. They chose upper-abdomen, lungs and head-and-neck separately as their regions of interest. They use the word 'Patient-specific' in their title, because the network uses a priori knowledge of a patient in the training phase.

2.9.2. Dataset

At first 3D CT volume image was acquired for a patient. Then this volume image was subjected to a series of translations, rotations and organ deformations through data augmentation. Each of these individual cases of data augmented volumetric images formed a single label for a single data point. Corresponding to this single data point, DRR was generated either from a single or multiple projection views by using TrueBeam imaging system and each of these DRRs or set of DRRs (for multiple projections) formed a single input for the same single data point. Hence each data point in the training and validation sets consists of a 3D CT volume image and it's corresponding 2D projected DRR from one or multiple views. For test set, they followed the exactly similar procedures for generating DRRs (input) from fresh set of augmented volume images (target).

2.9.3. Network Architecture

Figure 2.17 shows the architecture of the deep neural network used for their research.

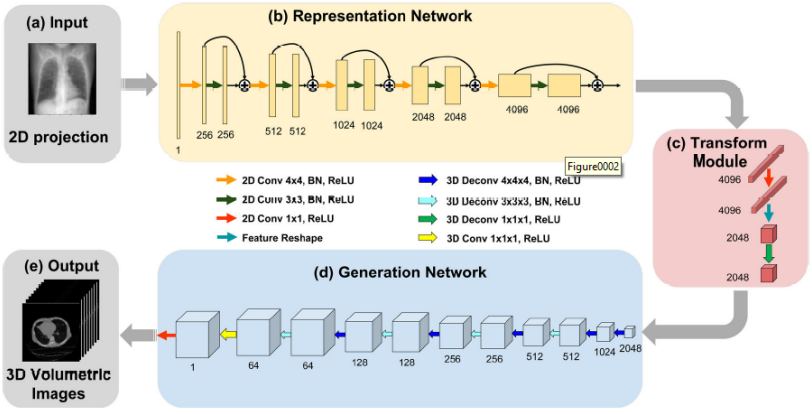


Figure 2.17.: Network architecture of related work by authors. It consists of 3 different parts, Representation Network, which learns the semantic relationship between the 2D input projection DRRs and the 3D volumetric CT image, Transformation Module, which bridges the gap between the 2D and 3D feature representations and Generation Network, which provides structures to the learned representations. [image source: [90]]

The network consists of a encoder-decoder framework doing a feature-space transformation and as can be seen from the figure 2.17, it has three major constituents:

Representation Network : It is used to extract features and to also learn the semantic relationship between the 2D input projection DRRs and the 3D volumetric CT image. It has residual connections.

Transformation Module : It is used to bridge the gap between the 2D and 3D feature representations through redistribution of features.

Generation Network : It is used to provide structures to the learned representations and finally accomplish the feature-image space conversion.

2.9.4. Results

Figure 2.18 shows 4 slices from the ground truth of abdominal and lung CT of a patient in the first row. In each of a, b, c and d, the first row gives the prediction of the trained network for the corresponding ground-truth slices (labels), shown in the row above a, and the second row is the difference signal between the predictions and their ground truths (labels) for the number of input projection views.

It is quite evident from predictions and difference signals that DRRs generated for a single data augmented CT volume image from multiple views (2, 5 and 10) did not give better results than single view projection DRR. This is because the semantic representations of the underlying 3D scene is same for every 2D projection image for a particular CT volume taken from different angles or in other words, the representation in feature space does not change across transformation of multiple 2D projections, taken from different angles, into the same 3D CT image. Hence they concluded that it is possible to generate a 3D CT volume image from a single view projection DRR with the a priori knowledge of the patient.

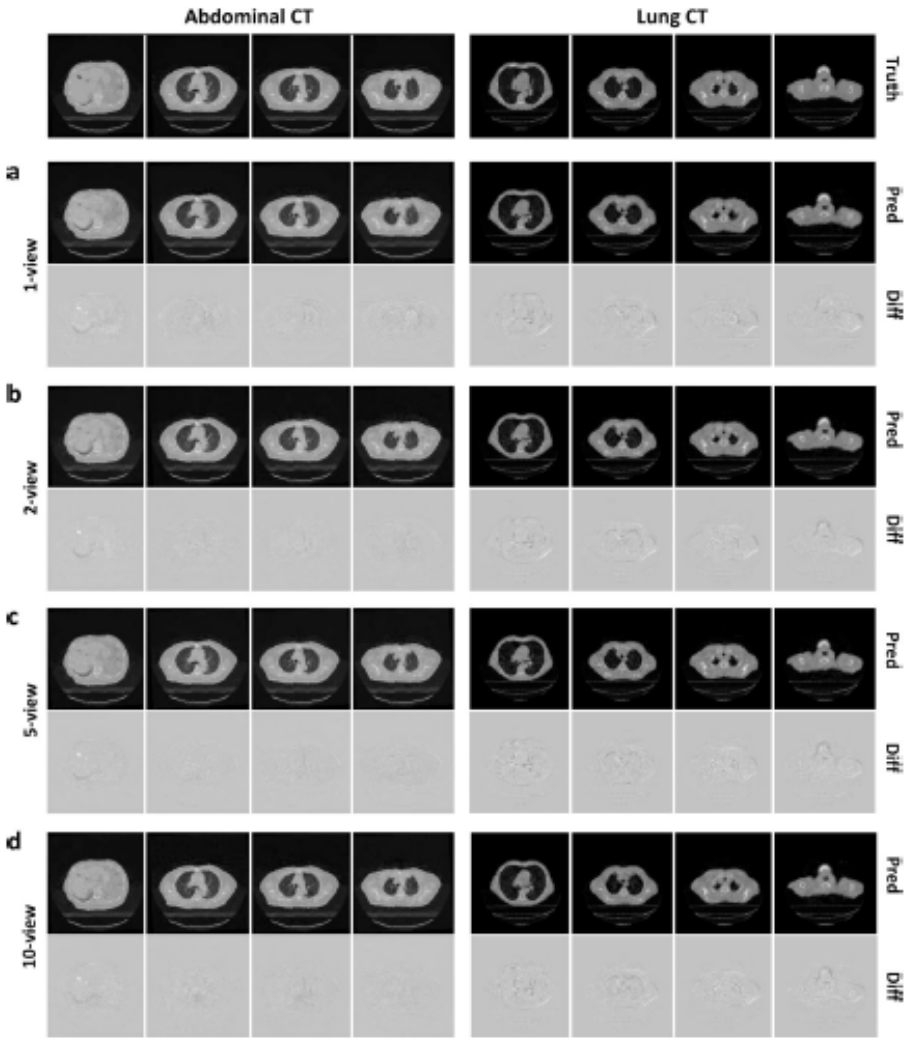


Figure 2.18.: CT slices with different views by authors. Top row represents ground truth of CT slices. First rows of **a**, **b**, **c** and **d** shows prediction on CT slices with 1-view, 2-views, 5-views and 10-views input DRR respectively. Second row for every view shows the difference signal between prediction and ground truth. [image source: [90]]

2.9.5. Drawback

Although the predictions show striking resemblance with the labels, this training method has a major drawback and it lies with the a priori aspect of it. For every new patient, the 3D CT volume image has to be acquired and the dataset needs to be prepared exactly in the same process described above. Following that, the network needs to be trained on this a priori information of the patient with a single view projection, so that the network can output the corresponding same CT volume in evaluation phase when another 2D DRR, acquired from any fresh angle, is fed into the network. In other words, to obtain a CT volume information for a patient from the network, the patient has to do the CT scan in the first place and then generate the same CT scan from a projection DRR or X-ray, thereby making no sense eventually as the network trained on generating 3D CT volume for a dataset of known

patients cannot be used to generalise for predicting the output for an unknown patient. But the main objective of deep learning or machine learning is to make predictions for unknown data based on the learning that the networks have acquired from known data and not make predictions on only known data.

Our task aims to counter this particular drawback and come up with a deep learning model which can generate 3D CT volume image from their corresponding 2D X-ray image for unknown patients based on the learning from known patients of training dataset.

3 Experiments

This chapter illustrates the entire experimental setup used for the project. From explaining the different forms of network architecture, associated with multiple experiments of the thesis task, to giving an overview of the list of conducted experiments for generating results and also for subsequent evaluations, and from describing the deep learning and the general experimental setup, related to these experiments, to narrating the complete dataset preparation method used for training, right from acquiring the raw dataset till it's final pre-processing, it does not leave out any detail related to the conducted experiments.

3.1. General setup

The general setup is described below.

3.1.1. Remote Connection

There are several applications which could have been used to have a remote access to my UKE system. But primarily AnyDesk, Team Viewer and Chrome-remote-desktop were used at different points in time depending upon circumstances and availability. Initially, we used AnyDesk until the time, when AnyDesk server suffered a week-long downtime, after which we decided to shift to using Team Viewer. It was working fine too till it's license for free use for academic purposes expired. In the last phase, we decided to use Chrome-remote-desktop. All of them provide an easy and user-friendly interface but AnyDesk has been the best in terms of smoothness in user experience and Team Viewer is the easiest when it comes to setting up the connection and Chrome-remote-desktop has an unique advantage over the rest two by allowing remote access without requiring the monitor of the remote system to remain on. How to set up the connection is beyond the scope of this report and can be found in the respective websites from where the softwares can also be downloaded.

3.1.2. Integrated Development Environment

PyCharm has been used as the Integrated Development Environment (IDE). It works really great across all platforms of Windows or Linux or macOS. Both the Professional and Community versions have been used, depending on availability of license. It [103] has a very user-friendly interface and is extremely intuitive. Also, it provides a nice look and feel and an enjoyable coding experience. This is because of the multiple in-built features that it has including coding assistance and analysis. It is very sharp in highlighting error and providing with easy and quick fixes. It can provide similar fixes for uninstalled packages. Not only that, it provides a nice structuring among files which makes it easy to navigate across the project. These things also contribute to easy refactoring of codes. Due to the user coding experience in terms of ease and fixes and several other feel good factors and technical

assistance, PyCharm has been chosen for the project over similar other IDEs, like Jupyter Notebook or Anaconda, available in the market.

3.1.3. Framework

PyTorch is an open source machine learning library, used in computer vision and natural language processing. In recent years, PyTorch has gained immense popularity among academic researchers and that's why it offers a great community support in forums like Stack Overflow and similar.

There are 3 most popular frameworks in use today and those are Keras, Tensorflow and PyTorch. But PyTorch is making good advancements in terms of popularity. It might not provide easy readability like Keras in terms of writing code. But it offers a lot of flexibility in coding or in writing custom layers of architecture since it has a much lower level API than Keras. Hence not only customisation but also debugging is easier, once someone has understood the flow in the framework. Moreover it is much faster than Keras and provides easy access to system GPU for training deep neural nets with many layers and parameters. From user experience, writing a custom metric was much easier in PyTorch than in Keras.

3.1.4. System GPU

Two different kinds of systems were used during the whole process of thesis work and they had two different GPU configurations.

The first system was exclusively for the thesis work at the Institute at UKE and has two GPUs, namely Nvidia Geforce GTX 1080 Ti, each with 11 GB of memory. Even with this high GPU configuration, it took between 3 and 5 days to run a significant number of epochs, from which it can be concluded that the network completed the learning process. Unfortunately, multi-GPU training was not implemented, thus it took a few days longer than expected.

The second system which was used at the TUHH and it is a cluster system which was not completely allocated for this work. This system has four GPUs, namely Nvidia V100 Tensor Core GPU, each with 32 GB of memory. Even with this system, it took between 2 and 5 days to train the bigger/complicated networks. Not only that, it failed to run 2D-3D UNet training on CT volume of $512 \times 512 \times 512$ pixels, thereby indicating that the cluster system has also hit it's capacity. Here, multi-GPU training was implemented to complete the training a little quicker.

3.2. List of experiments

This section is broadly divided into 2 parts - Experiments and Subsequent Evaluations.

3.2.1. Experiments

We chose 7 different parameters, apart from the baseline, for training the neural networks so as to find out how distinctly these trained networks will perform while trying to do evaluations on unseen test data. Additionally, for some of these parameters, there were a few options, that were required to be further tested. This complete set of parameters and their related options will provide to the reader a nice

and well sought out comparative analysis among the impact of each of these parameters on network training and prediction. In case, if every possible permutation among each of these 7 parameters and their corresponding options, along with baseline, would have to be tested then that would entail a much longer time for concluding the report. Hence we chose elimination method for declaring the winner at every stage. The winner from the current stage is carried over to the next stage where the next parameter would be tested based on it's own options, and the winning combination for that stage will be carried over to the next similarly and so on. So we carried out these experiments based on these 7 parameters, including the baseline, in 8 different stages. Eventually we had the most idealistic combination at the end of the 8th stage which yielded the best results on paper, but would not fulfil the feasibility check, when it comes to practical implementation, due to one particular aspect of the entire setup - view points of DRRs. Hence, we introduced a final Miscellaneous stage (stage 9) and conducted a couple of experiments, where we retained every other aspect of the idealistic combination from the last (8th) stage, except the view-points of DRRs. These couple of realistic experiments, along with the idealistic winner from the last stage are the 3 final setups to be rolled out as a part of the thesis task. Including the Miscellaneous stage, we have altogether 15 experiments, spanned over 9 stages. A brief overview of the experimental setup related to these 9 stages along with the winner for every stage is provided below, but the network evaluations and their corresponding discussions are presented in chapter 4.

Network dictionary was saved for both current epoch and also for that epoch which gave the best result on validation data so that the saved network can be used for further evaluation on test data later. This saving of the network weights for the best state of validation data is known as early saving or early stopping, in case the network was stopped from further training after the point of overfitting. This is done because beyond the early saving/stopping point the network starts to overfit to the training data and stops generalising on the validation data. The training was not stopped at the point of early stopping so as to ensure that the network does not achieve better state in future epochs. Out

Stage 1 - Baseline

In the first stage, we conducted an experiment on a cleaner and smaller version of the LIDC-IDRI dataset, consisting of only 125 good samples. This dataset comprises of a list of top-view 3D CT volume and their corresponding front-view 2D DRR as label-input pairs. MeVisLab was used to normalize each of these CT volumes to a fixed dimension of $200 \times 256 \times 256$ pixels and also to generate front-view DRR of the dimension of 256×256 pixels, by projecting the CT volumes. The generated DRRs lacked in appearance a realistic look and feel, when compared to real-worldd X-rays/DRRs. This experiment, where we tried to generate a 3D CT volume, of 200 top-view slices from front-view DRR, has been treated as the baseline, against which all other parameters will be tested in subsequent stages. The purpose of extracting a cleaner version of the dataset is that, it can cause the neural network to overfit to the specifics of noise of the noisy training examples and thereby perform poorly on the unseen test data. Results were not impressive for this stage for obvious reasons, which have been discussed in detail in chapter 4.

Stage 2 - Dataset Pre-processing Method

In this stage, a dataset, exactly similar to the one discussed in the previous stage, in terms of dimensions and view-points, was generated by Python script. This time the DRRs looked much more realistic. Moreover, the Python script took into account the factor of isotropy/isometry across the CT slices,

while resampling and in their corresponding DRRs, while projecting, and also across different patients. We wanted to test at this stage that whether these two factors, realistic-looking DRRs and aspect of isotropy in pre-processing of the dataset, would have an advantage over the baseline setup. The script created dataset gave an inevitable edge over the baseline one, created by MeVisLab. Hence to test the further parameters in later stages, we move on with the winning combination from this stage, which is basically the dataset created by Python script.

Stage 3 - CT-DRR Combination

In this stage, three different experiments were run on the Python script created dataset, but from this stage onward, we have used CT volumes of dimension of $256 \times 256 \times 256$ pixels and DRRs of dimension of 256×256 pixels. The purpose of this stage was to find tshe best DRR-CT combination for training. In the first one, we tried to generate frontal CT slices from front-view DRR and in the second one, we wanted to generate lateral CT slices from lateral DRR. The final experiment involved generation of top-view CT volume from top-view DRRs. This was compared with the the output from the last stage which generated top-view CT volume from frontal DRR. The top-view CT along with top-view DRR was the winner among all and also gave better results than the CT-DRR combination from the last stage.

Stage 4 - View Point

The last stage rendered best results, while generating top-view CT slices from a single top-view DRR. In this stage, two more experiments were conducted. The first one involved generation of 3D volume with top-view slices from double-view DRRs, frontal and top, and the second one was about generating the same from triple-view DRRs, frontal, lateral and top. This was done so as to find out if multiple views can considerably improve results or not. Triple view DRRs was the winner here, but did not improve the results substantially.

Stage 5 - Data Augmentation

For this stage, data augmentation was performed in an experiment as till now every experiment was run without data augmentation. Every other aspect remained just the same. This was done so as to find out if data augmentation, which helps to avoid overfitting with a smaller dataset, could prove to be useful for our task or not. It did not improve the results. As there was no formal winner for this stage, data augmentation was still kept as a part of the training process with a hope that it might help in some other combination at some other stage.

Stage 6 - Loss

Till now only decomposition loss was used. Now in one of the experiments, reconstruction loss was combined with the decomposition loss and in the next experiment, latent loss was combined with reconstruction and decomposition loss, put together. This was done so as to find out if backpropagating the reconstruction or latent losses could render better results over the baseline network. Decomposition plus reconstruction loss improved the results over mere decomposition loss in this phase.

Stage 7 - Network

Till now 2D UNet was used for all experiments. Now the winning combination from last stage was tested with a different network architecture of 2D-3D UNet, so as to find out whether the network architecture was posing a constraint in improvement of results or not. But as the 2D-3D UNet failed to improve the results, we go on with 2D UNet as the other network puts burden on the GPU due to it's high memory requirement. Also in the next stage, where we have to test the volume dimension, it will be already stressful for the GPU and additionally this new network would surely result in a memory overload.

Stage 8 - Dimension

Till now $256 \times 256 \times 256$ CT volume of top-view slices was used along with $3 \times 256 \times 256$ pixels DRR dimension, as triple view DRR was the winner of stage 4. Now in this stage we conducted two different experiments - first one, with a CT dimension of $128 \times 128 \times 128$ pixels and DRR of $3 \times 128 \times 128$ pixels and second one, with a dimension of $512 \times 512 \times 512$ pixels for CT and $3 \times 512 \times 512$ pixels for DRRs. We want to find out whether increasing or decreasing the resolution will help in the overall accuracy. Decreasing the dimension means decreasing the detailing and thereby forcing the network to not remember every specifics of a particular data entry. It helps in avoiding overfitting. Increasing the resolution means exactly the opposite. So this was a really interesting parameter to test. Interestingly, $512 \times 512 \times 512$ slices improved the accuracy at this point. The explanation for this has been provided in chapter 4.

Stage 9 - Miscellaneous

Till the last stage, we have been able to find out the most idealistic combination, which has excelled in terms of performance, based on 8 different parameters. But those ideal combinations on paper would not be feasible in reality in some cases or in some other cases would incur more costs and expose patients to a greater amount of radiation, thereby going back on the primary objectives of the thesis. To explain the first case of non-feasibility, top view DRR is a perfect example. There is no way in which the existing health care system would allow extracting a top view DRR. Also, triple view DRRs fit rightly into the perspective of extra cost and extra radiation. So in this stage, we have tried with the winning combination, two different experiments. The first one was generation of frontal CT slices from frontal DRR, an experiment which has already been run in stage 3, but this time with the final winning combination for comparison of the realistic results with the ideal ones. The second experiment was with just one additional view – lateral DRR. This experiment was not run before. The double view DRR did not improve the results. So with an additional view of DRR not helping the training process and incurring just an extra cost for patient for no purpose, the single view DRR is the winner for exclusively this particular stage.

But for the final set of results, we have put the idealistic winner from the last stage and also both the realistic ones from this stage. So essentially, we can verify the impact of all three view-points of DRRs on network evaluations. With the idealist winner from the last stage, we can input the complete set of triple-view DRRs and expect top view CT volume at the output. For the first experiment of this stage, we will input single view frontal DRR and for the second one, double view DRRs - frontal and lateral, and expect frontal CT volume at the output. Thereby the impact of 1/2/3 view-points, which was one of the major tasks of the thesis, will be examined in detail.

Summary

The summary for this list of experiments is provided below:

Stage 1 : Baseline was achieved.

Stage 2 : Dataset acquisition by Python script gave superior results.

Stage 3 : Top-view DRR –> Top-view CT turned out to be the best CT - DRR combination.

Stage 4 : Triple-view input DRRs gave a slightly more edge over the double/single-view DRRs.

Stage 5 : With no formal winner, data augmentation was retained.

Stage 6 : Reconstruction loss along with decomposition loss was carried over as the winner for further stages of experiments.

Stage 7 : 2D-3D UNet did not improve or deteriorate results and 2D UNet was preferred from now on.

Stage 8 : Triple-view DRR of $3 \times 512 \times 512$ pixels and CT volume of $512 \times 512 \times 512$ pixels turned out to be the winning input-label combination.

Stage 9 : Double-view input DRR is considered the winner with network outputs a little better visually.

Results for these 15 different experiments from 9 distinct stages have been discussed in the next chapter, which also includes discussions about 4 more experiments, carried out as a part of subsequent evaluations, which have been narrated in the 3.2.2 section.

3.2.2. Subsequent Evaluations

There were three evaluations that were further conducted and evaluated in addition to the previously listed experiments. Those are explained below.

3D CT –> 2D DRR

The thesis objective was to be able to generate 3D CT volume of patients from their corresponding 2D DRR/X-ray images. But this time the opposite target was set. In other words, we wanted to generate 2D DRR image from a 3D CT volume of a patient. Results were almost close to reality, thereby indicating that it is much easier to render the projection 2D image from a 3D volume than to interpolate a high dimensional and complex 3D volume information from a simple 2D image array.

3D CT –> 3D CT

In the next experiment, we try to generate 3D CT volume from itself. The purpose of this stage was also the same like the last one. It is to show that apart from the task of generating 2D DRR from 3D CT volume being an easy one, the task of generating the volume from itself is also much easier for the

network than generating the same from 2D DRR, given the amount of information lost in the process of 3D volume getting collapsed in those 2D DRRs.

Triple-view Noisy DRRs –> Triple-view Noiseless DRRs

In the third experiment , we tried to find out whether the network is suitable for performing a typical task, for which the network was designed for. This is to clarify that every aspect related to pre-processing the dataset, training the network and architecture for this particular network is in place and the only way to improve the accuracy further is to try out a different approach that entails a different network architecture and also a different dataset – a cleaner one probably. So this experiment was all about trying to generate a noiseless version of DRRs from it's noisy counterpart. The results prove that the network is performing in an excellent manner in it's typical tasks.

The results related to the experiments for all the 9 stages and also for subsequent evaluations have been presented in the next chapter.

3.3. Network Architectures

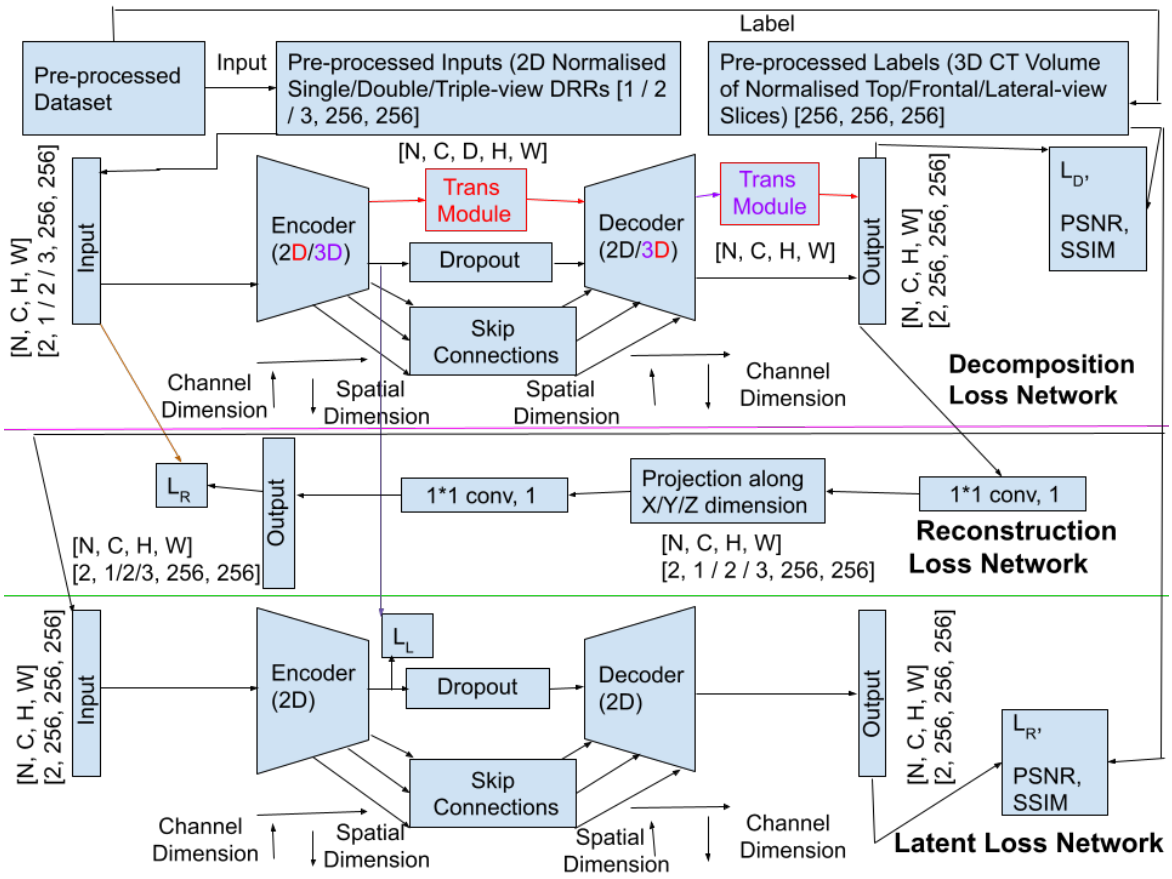


Figure 3.1.: Complete network architecture

Figure 3.1 shows the architecture in $[N, C, H, W]$ ([Batch Size, Channel Dimension, Height of Image, Width of Image]) format, which is the format for PyTorch tensors for neural network operations. In this representation, we are inputting either single/double/triple-view DRRs, each of dimension 256×256 pixels, and we are expecting at the output a decomposed CT volume of dimension $256 \times 256 \times 256$ pixels. Also we have considered a batch size of 2. So the input will have a dimension of [2, 1/2/3, 256, 256]. Correspondingly, the output dimension will be [2, 256, 256, 256]. The second dimension of the input is kept optional as (1/2/3) due to the fact that user can input either a single or double or triple-view DRRs, depending on the requirement. The network can be broadly divided into 3 different sections: Decomposition Loss Network, Reconstruction Loss Network and Latent Loss Network.

3.3.1. Decomposition Loss Network

With reference to the figure 3.1, decomposition loss network (DLN) has been used in all stages of the experiments. It consists of input, output, encoder, decoder, skip connections, originating from the encoder and terminating in the decoder, and a mandatory dropout module, in between the encoder and the decoder. The encoder is responsible for extracting the features from a 2D DRR and converting them into a latent space information with reduced spatial dimension and increased channel dimension, in a way that enables the decoder in it's process of decomposition. The decoder does exactly the opposite job of converting that compressed and useful latent space information into a 3D CT volume, with increased spatial dimension and reduced channel dimension. The skip connections capture the high resolution spatial features from the early layers of the encoder without any degradation and concatenates the same with the features from the latter layers of the decoder. This useful information, from the early layers, which might have been lost in the subsequent layers, provides an additional and constructive insight to the decoder to help in it's overall process of decomposition. Dropout module helps in adding a bit of regularisation. The encoder and the decoder consist of repeated blocks of convolution and max pooling (encoder)/upsampling (decoder) layers. Also after each of the convolution layers, ReLu activation has been used, except for the last layer, which uses sigmoid as activation. Also, batch normalisation have been used in each of these repeated blocks for only a few experiments. For most of the experiments, which use 2D UNet, both encoder and decoder use 2D convolutions and 2D Max pooling (encoder)/bilinear upsampling (decoder). For those experiments, which used 2D-3D UNet or in other words a combination of 2D encoder and 3D decoder in UNet, 2D convolutions were used along with 2D Max pooling at the encoder and 3D convolutions along with trilinear upsampling at the decoder. Also group normalisation and leaky ReLu were used in place of batch normalisation and ReLu respecively in the repeated blocks and dropout module as well as skip connections were taken down for this particular setup. Since this network architecture makes use of 2D encoder and 3D decoder, this transformation from 2D to 3D feature space by redistribution of features is achieved by a transformation module in between the encoder and the decoder. This module consists of three blocks of convolution layers and reshape operations. The torch tensor, after this transformation, looks like $[N, C, D, H, W]$, with an additional D (Depth) dimension included after transformation. Hence, the features need to be reshaped from $[N, C, D, H, W]$ to $[N, C, H, W]$ format before the output is generated for the network. This is done by another transformation module at the end of the decoder. Going by the name (i.e., decomposition loss network), this network uses decomposition loss (L_D) for backpropagation and the SSIM and PSNR metric for evaluation. This is the baseline network, where the single/double/triple view 2D DRR is decomposed into 3D CT volume which is compared with the 3D CT labels for the decompositional loss calculation. This section is placed right above the magenta line in the diagram.

3.3.2. Reconstruction Loss Network

Reconstruction loss network (RLN) has been from stage 5 onward. This part of the network tries to reconstruct the 2D DRR (single or double or triple view) from the decomposed CT volume. It has been implemented so as to find out if trying to reconstruct the input from the output can help to improve the results of decomposition used in our baseline in the decomposition loss network. Since decomposition loss is always the baseline, so all those network architectures which used reconstruction loss, used it on top of the decomposition loss. It has a sequence of 1×1 convolution layers which projects the generated 3D CT volume along the C dimension (i.e., C will be one, two, or three depending on whether we are trying to generate a single, double, or triple view DRR, respectively). The reconstruction loss is calculated between the single or double or triple view 2D input DRRs and the newly generated projection DRRs. The network adds the reconstruction loss (L_R) to the decomposition loss (L_D) to form the total collective loss. This section is depicted in between the magenta line below the decomposition network and the green line above the latent space network.

3.3.3. Latent Loss Network

The last network uses the latent space loss aspect. The purpose of this network is to first try to reconstruct the 3D CT volume from the same so that the encoder knows how to tune it's weights for a successful reconstruction of its slices. At first this network is trained separately using reconstruction loss, calculated between the 3D CT volume generated at the output and the same fed into the input of the network. Once the network is trained, now the encoder knows how to optimise itself for generating a perfect latent space at the output of the encoder. Now while training the primary network (i.e., the DLN+RLN), the output from the encoder part of the latent loss network is fed into the same of the primary network for latent space loss (L_L) calculation. This information from the latent space of the latent loss network, which is optimised for a perfect reconstruction of 3D CT volume from same, guides the primary network to reshape it's latent space so that it can perform the decomposition process in a more efficient way. This part of the network was used with the baseline and reconstruction part of the network put together to find out if it improves the results over the baseline. It uses reconstruction loss during the pretraining. It is used only in stage 5. It is below the green line in Figure 3.1.

3.4. Dataset preparation for training

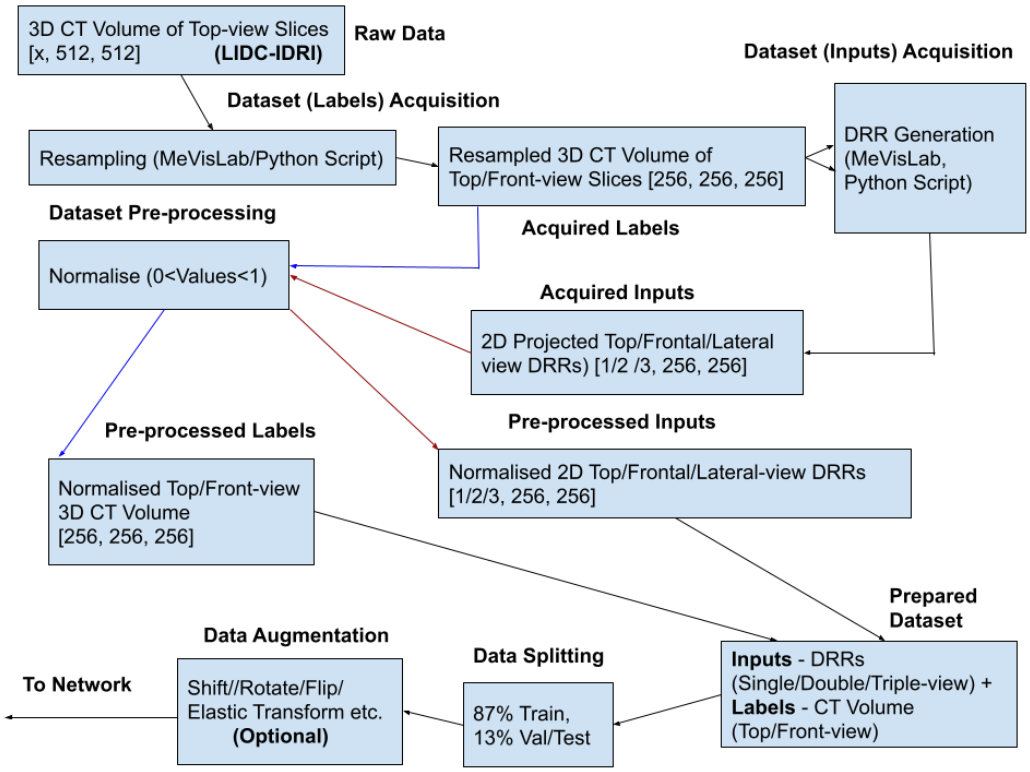


Figure 3.2.: Complete dataset preparation workflow

The entire workflow of processing of the raw data, right from the point of downloading it, into it's final processed version, after which it is fed into the network is shown in the figure 3.2 and is described below.

3.4.1. Raw Data

We used the Lung Image Database Consortium-Image Database Resource Initiative (LIDC-IDRI) dataset as our raw data. It has 1012 entries of patients and each has a corresponding CT volume. These samples had non-normalised number of top-view CT slices and each slice had the spatial size of 512×512 pixels. A cleaner version of the same dataset, consisting of only 125 good samples, was used.

3.4.2. Dataset Acquisition

The 3D CT volume of top-view CT slices was resampled to a standard CT volume of $x \times y \times z$ pixels depending on the different stages of our experiments. We used three different spatial sizes. Firstly, we resampled each CT volume to $256 \times 256 \times 256$ for the stages 1 to 7. Secondly, we resampled each CT volume to $512 \times 512 \times 512$ for the stages 8 to 9. Finally, we resampled the same to $128 \times 128 \times 128$ for the stage 8. Apart from the dimension, 3 different view-points, namely top, frontal and lateral,

were taken into consideration while normalising the CT volume. Stages 1-8 used top-view CT volume. Stages 3 and stages 4 and 9 used front-view CT volume. Finally, lateral-view CT volume was used only in stage 3. This normalised 3D CT volume with stage-specific dimension and view-point (e.g., $256 \times 256 \times 256$ of top-view CT slices), formed the label part of the dataset.

The 2D DRRs were obtained from the normalised 3D CT volume. The dimension for a single 2D DRR was inline with the dimension of the normalised CT volume (e.g., the $256 \times 256 \times 256$ CT volume was projected to a 256×256 DRR). Hence, dimensions of the generated DRRs, for stages 1 to 7, were 256×256 and for 8 to 9, were both 512×512 and 128×128. Same as the CT volume, the input DRRs also had three different kinds of views - top, frontal and lateral. Stage 1 used top-view, whereas stage 3 used also front and lateral-view DRRs. We also combined the DRRs in double-view and triple-view formats with respective dimensions being $2 \times 256 \times 256$ and $3 \times 256 \times 256$ pixels. Stages 1, 9 and 7 used single, double and triple-view DRRs respectively. Hence the 2D DRRs with stage-specific dimension, view-point and number of view-points (e.g., $3 \times 256 \times 256$ of triple-view DRRs) formed the input part of the dataset.

At this point, as an example, the dataset consists of 3D NumPy arrays for inputs with dimension $3 \times 256 \times 256$, for triple view 2D DRRs and 3D NumPy arrays for labels with dimension $256 \times 256 \times 256$ for top-view 3D CT volume.

This whole process of acquisition of dataset was accomplished either by using MeVisLab or by Python script. MeVisLab was used only in stage 1, whereas Python script was used for all stages afterwards.

3.4.3. Dataset Pre-processing

Following this, both inputs and labels were normalised to values ranging from 0 to 1 during training using Python script. Normalisation of labels is shown by blue arrows and of input by red arrows.

3.4.4. Data Split

We randomly split the 125 samples in 87% and 13% for training and validation/test data.

Once the dataset is pre-processed and splitted completely, it is loaded using dataloader of Pytorch with a pre-determmed batch size.

3.4.5. Data Augmentation

Once the dataset is loaded as input-label pair by the dataloader, exclusively for stage 5, data augmentation is performed for both inputs and labels, using a PyTorch library, Albumentation, and the different types of data augmentation that were implemented are Shift Scale Rotate, Random Crop and Reshape, Horizontal Flip, Vertical Flip, Elastic Transform, Random Brightness, Random Contrast, Median Blur and Gaussian Noise.

Now the dataset consists of Pytorch specific torch tensors in $[N, C, H, W]$ format, where N stands for batch size, C is the channel dimension, H represents height of the image and W is the width of the image of respective dimensions, which is finally fed into the network.

3.5. Deep learning setup

The deep learning setup is described below.

3.5.1. Batch Size and Epochs

Batch size was set to 1 for all the stages from 1 to 6. This is due to the fact that the output that we are expecting from the network should be a 3D CT volume, usually of the dimension of $256 \times 256 \times 256$ pixels. So with such a high dimension required to be reproduced at the output, the channel dimensions for all the hidden layers of the UNet had to be kept subsequently higher, thereby reaching up to a maximum of 3072 at the end of the encoder. This requires training of a huge number of parameters and that resulted in hitting the maximum system capacity already. If the batch size had to be increased, then the channel dimensions in all the hidden layers had to be reduced which would then give us a sub-optimum result. Therefore, the channel dimensions were set, keeping in mind what would generate optimum results and also what the system can handle at most. That resulted in a trade-off with the batch size, which could be set only to 1. But for stages 7-9, which were run on the high configuration cluster system having 4 GPUs, the batch size could be set to 4, after enabling multi-GPU support in the script.

For every experiment, different number of epochs were run ranging from 200 to 1800 depending on the criticality of the task.

3.5.2. Regularisation

L2 regularisation was implemented for all the stages. The weight decay was tested for all decimal values ranging from 10^{-3} to 10^{-6}.

A consistent dropout of 0.5 was used only in between the encoder and the decoder and not in any other of the hidden layers.

Additionally, early stopping was used in all of the experiments to save the network weights for it's best state other than saving the network weights for the current state at the end of every epoch in order to use the best weights for making prediction. This was done so as to avoid the situation of overfitting by missing out on the exact moment when the training needs to be stopped to avoid the same. This is done because beyond the early saving/stopping point, the network starts to overfit to the training data and stops generalising on the validation data. In our setup, the training was not stopped at the point of early stopping so as to ensure that the network does not achieve better state in future epochs.

3.5.3. Optimiser

SGD was used as an optimiser only in one of the stages with learning rate scheduling. In rest of the experiments, Adam was used.

Any learning rate higher than the optimum would result in the training being stuck in a local minimum, as a result of which after repeated epochs of training, the validation accuracy would not reach it's optimum level and fluctuate around a sub-optimum. Any lower learning rate would slow down the training process even further thereby requiring a lot more time for completion of training, sometimes overshooting even by a week. Every learning rate[51] between 0.01 and 10^{-6} was tried and tested so as to find the optimum one.

Learning rate schedulers are used to adapt the learning rate according to situations. In the initial phase of the training when the optimisation is far away from the global minimum, learning rate can be kept high so as to speed up the training process and while it is close to the optima, it should reduce it's stride so that it does not skip the optima. So finding the optimum initial learning rate is an ideal solution for an optimiser like Adam which adapts it's learning rate with epochs depending on the situation. But for an optimiser like SGD, which do not have adaptive learning rate mechanism, it would either slow down the training or never let the network reach it's state of optima. Learning rate schedulers are therefore useful to tackle this situation. So we used reduce on plateau learning rate scheduler in experiments where we used SGD as an optimiser, which we mentioned in the previous subsection.

3.5.4. Loss and Metric

Decomposition loss (L_D) was used in all of the experiments. It is a weighted combination of L1 and L2 losses. This was treated as the baseline. Additionally, stage 6 made use of the reconstruction (L_R) and the latent space loss (L_L) so as to find out if those would improve the results over the baseline. They comprise solely of L2 loss.

For metric, structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR) were used. For PSNR, code was written manually and for SSIM a library was invoked for the purpose of implementation.

Out of the three different measurements: loss, SSIM and PSNR, the SSIM was chosen as the main measurement for early stopping. The other two measurements have their best states in close enough epochs but not necessarily always at the same epoch. So all the three measurements (SSIM, loss and PSNR) are saved along with their network dictionary for the current epoch. The highest stored validation SSIM value is always compared with the next epoch SSIM value. In case the current SSIM value has a higher value than the highest saved one, then the current SSIM value is stored as the highest SSIM value and the highest stored SSIM value and the corresponding state dictionary for this best state is also stored. Even when the network starts to overfit, it will still kept on training for a considerable number of epochs so as to find out that at any state at the end of future epochs, the validation SSIM for that current epoch overshoots the maximum attained previous value. In case it does, best network dictionary will be updated and in case it does not the best network dictionary is retained in the form of the last saved one. So that explains early saving in place of early stopping.

The reason why validation SSIM is chosen as the parameter is because it talks a lot about the structural similarity between outputs and labels which is of more importance compared to the loss values or PSNR values. So even if the validation loss is minimum and validation PSNR is maximum at another epochs, it is the structural visual similarity that we are looking at for our training.

4
Results and Discussion

In the 3.2 subsection of previous chapter, we have explained the list of experiments that we have conducted in different stages. In this chapter, we have presented, in stages, the results for each of these experiments. It is broadly divided into 2 parts:

- **Training Curves :** It includes plots of three different training curves, namely the loss curve, the SSIM curve and the PSNR curve for both training and validation data.
- **Visual Network Performance :** It shows the performance of the trained network in it's evaluation phase. We have included an example input - 2D DRRs (single/double/triple-view) - from the test dataset and the corresponding network output - 3D CT scan (top/front-view). We saved the network dictionary for it's best state during training. Later on, we used this saved network and it's dictionary for doing evaluation on test data.

Additionally, we have also provided a discussion on the results for each of these stages, where we analyse both the training curves and the network performance for the conducted experiments.

We have also included experiments carried out under subsequent evaluations, mentioned in the 3.2.2 section of the previous chapter. Apart from that, we have listed out the loss, PSNR and SSIM values for each of these experiments in a table for doing a comparative analysis. Finally, we have put together the accuracy scores for the SSIM metric for the 3 best setups from our thesis work.

4.1. Baseline

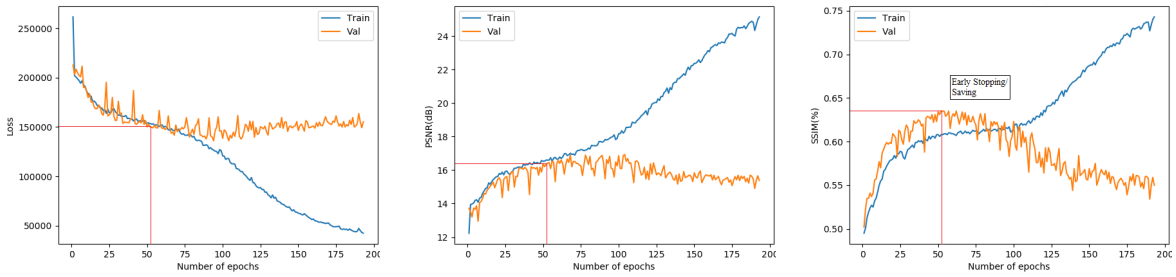
A baseline experiment was conducted in this stage. As the name suggests, this experiment will be treated as the baseline, against which results of all other experiments/stages will be compared.

4.1.1. Unrelated CT-DRR+Non-Isotropic CT+Unrealistic DRR+MeVisLab

For this experiment, the cleaner version of the dataset, comprising of only 125 good samples, was used. We used only single-view (frontal) DRRs and top-view CT volume of 200 slices as input-label pair. As a result, the output and the input dimensions for this phase were $200 \times 256 \times 256$ and 256×256 pixels respectively. The pre-processing of the dataset was achieved completely by MeVisLab. For this stage only, early stopping/saving has been explained through the learning curves, the core idea of which remains same for all other experiments.

Training Curves

The loss, PSNR and SSIM curves for this stage have been presented in Figure 4.1 in (a), (b) and (c) respectively.

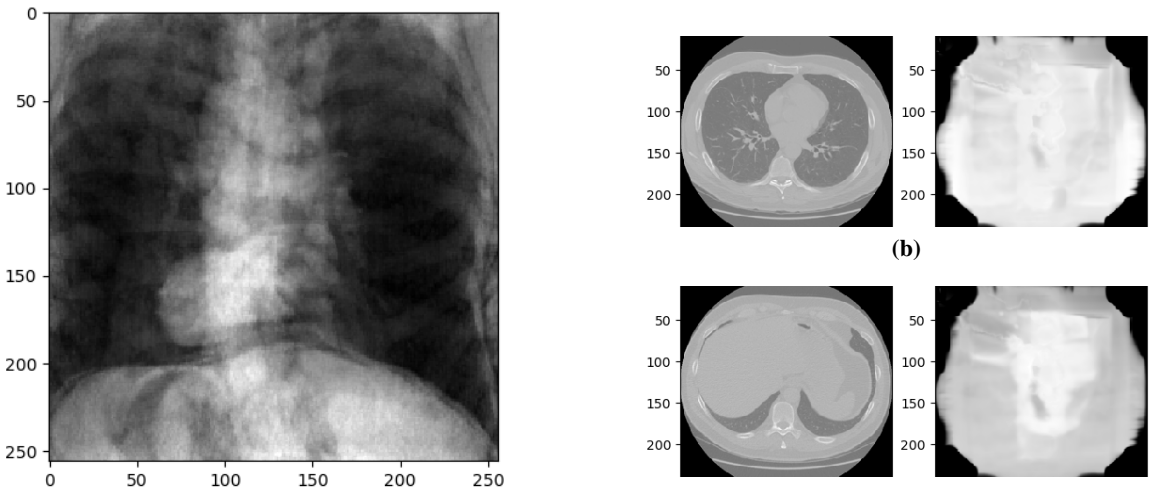


(a) Loss **(b)** PSNR (in dB) **(c)** SSIM

Figure 4.1.: Loss and metric results for stage 1 experiment 1 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

Let's have a look at the input DRR and also at the performance of the trained network on the test data of patient number 920 of the LIDC-IDRI dataset by putting out a comparison between the original and decomposed CT slices. Figure 4.2 shows in (a), the input DRR and in (b) and (c), the CT slices. Patient number 920 has been used for the purpose of evaluation for all stages except for subsequent evaluations.



(a) **(b)** **(c)**

Figure 4.2.: Visual example results for stage 1 experiment 1 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.1.2. Discussion

The maximum values of SSIM and PSNR achieved for this experiment are 0.635 and 16.935 dB, respectively. The lowest loss for this setup is 136080.714.

The network dictionary could have been saved for any of the three parameters but the parameter which has been chosen as the cornerstone is SSIM. This is because we are more concerned in the visual structural similarity of the generated output when compared with the actual labels. With reference to the diagram 4.1c, the best state which has been saved has been pointed out by the red lines interesting the validation SSIM curve at it's maximum achieved value of 0.635. So the values of loss and PSNR corresponding to the highest achieved value of SSIM, pointed out similarly by red intersecting lines in diagrams 4.1b and 4.1b, are 150214.282 and 16.339 dB respectively. These values of loss and PSNR are very close to their self achieved optimum values. In every other stage this method of early saving/stopping has not been illustrated, but the basic principle followed has been the same.

The reason for such low accuracy on the validation data can be attributed to 3 facts.

First of all, in this setup, we are trying to generate top view 3D CT slices from frontal view DRRs, which do not look same or even close. So the network is not learning anything from the training data that it can put to use to make predictions on the test dataset. Hence, the output slices look nothing more than a random smudge. In other words, we are inputting an image of an aeroplane to the network and expecting images of an ocean in each of it's output slices. There is hardly any structural useful information in the image of the aeroplane for the encoder of the network to fetch and forward to the decoder which in turn can reproduce the decomposed slices of absolutely unrelated images of oceans. An usual application of this network is to reconstruct a clearer image from a noisy version of the same where both the input and the label have correlated structural information which enables the network to learn the mapping correctly and perform the process of reconstruction successfully - an experiment which has been tested in the section of subsequent evaluations. With that correlation missing in this setup, it is no wonder that the network is performing so poorly.

The second issue with this setup is that the MeVisLab does not take into account the aspect of isotropy while resampling the CT volume to a fixed dimension. Isotropy provides a way of normalisation along the CT slices and also across the patients. We know that how much important is the factor of normalisation when it comes to training neural networks. Normalisation helps to limit the entire dataset in a similar range of values. For our case, the CT images are derived from completely different setups and that means that they can span over entirely different ranges of values. Isotropy can potentially fix the problem by bringing them in a desired range. With that aspect of normalisation by isotropy missing in our setup in this phase, it becomes extremely tough for the neural networks to learn from different range of output labels.

The third reason behind this low accuracy is that the DRRs generated in this setup using MeVisLab lacks the realistic look and feel. So again, the network fails to learn the semantic correlation between the DRR images and the CT volume. Even if the first problem would have been overcome by creating a dataset of visually related frontal DRR and front-view CT volume as input-label pair, the unrealistic DRRs would still force the network to struggle with finding the semantic mapping between the DRRs and the CT volume.

Hence in the next stage, we try to generate DRRs which look much closer to reality. Additionally, we also try to resample the CT volume by keeping in mind the factor of isotropy. This will be achieved by pre-processing the dataset by using Python script. This will hopefully counter the last two

problems discussed in this phase and also help us find out if it gives us as an edge over the baseline. The winner of the next stage will be carried over further to the third stage.

4.2. Dataset Pre-processing Method

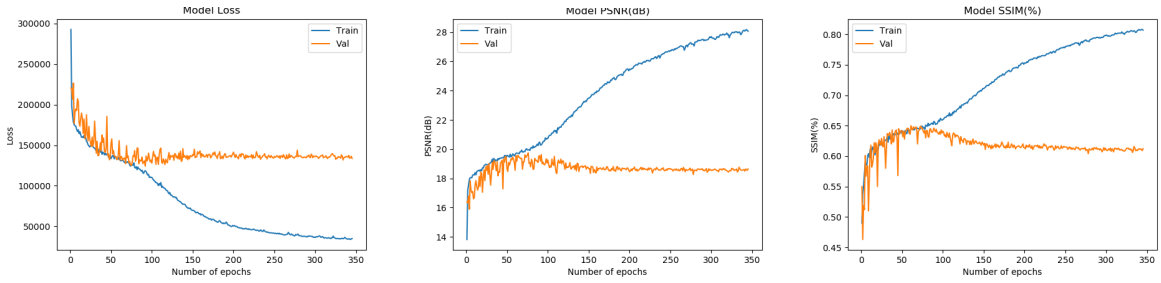
Also for this stage, we carried out a single experiment. It was conducted on exactly the same setup as that of baseline from last stage. The only difference was that, this time the dataset was pre-processed completely by Python script. The purpose of this stage was to find out whether the dataset pre-processed by the script could generate better results compared to the ones generated by MeVisLab, as it takes into account the factor of isotropy of CT volume and realistic looking DRRs while pre-processing the dataset. The results presented in this section will be compared with that of the baseline and the setup, whose results look better, will be carried over to the next stage, where the next parameter will be tested.

4.2.1. Python Script

In this stage also, front-view DRR was used as the input and 200 slices of top view CT volume was used as the label. As mentioned above, other than pre-processing the dataset by Python script, instead of MeVisLab, every other aspect of the setup remained exactly the same as the stage before.

Training Curves

The training curves generated for this setup, of data pre-processing by Python script, are available in the figure 4.3.



(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.3.: Loss and metric results for stage 2 experiment 1 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

Since we have generated the input DRRs in a much more realistic fashion in this setup, so let us have a look at the same in the figure 4.4a so that we can spot the realistic aspect of the DRRs visually, which was missing in the previous setup. Figures 4.4b and 4.4c show the comparison between the original and the decomposed the CT slices. As we have mentioned before, patient number 920 has been used for evaluation for this phase and for every other phase afterwards.

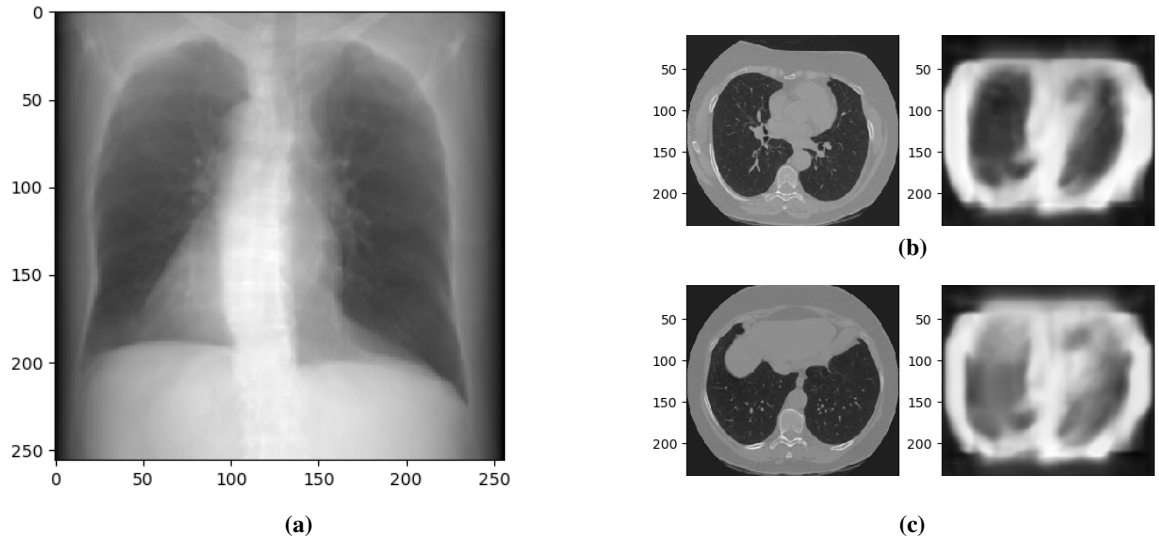


Figure 4.4.: Visual example results for stage 2 experiment 1 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.2.2. Discussion

The highest SSIM achieved for this stage was 0.65 and the corresponding PSNR and loss values are 19.56 dB and 131659.455. The results have improved only slightly, with respect to the accuracy metrics and the learning curves. But the corresponding increase in terms of visual clarity of the slices, shown in figure 4.4, is huge when compared to the same from the previous stage.

This immense improvement in appearance can be attributed to the different ways used by these two platforms for pre-processing the dataset. There were 2 major flaws in course of pre-processing of the dataset - one related to the pre-processing of the input DRRs and another one associated with the pre-processing of the output CT labels - in the last stage. The current setup addressed both these flaws in one go.

First of all, with this new form of pre-processing, the generated DRRs look much different and realistic than the previous setup, when you compare them to real world X-rays. Definitely, an SSIM score of 0.635 (from last stage) is not an impressive figure, but atleast it is a good enough value for the trained network to reproduce something better than just a mere smudge in it's evaluation phase. Two things are clear from the network evaluation of previous phase. There is hardly any information in the unrealistic looking DRRS or the network would be able to atleast reproduce a noisy CT volume with some form of structuring in it. Additionally, there is a missing semantic correlation between these input-label pairs or the network would be able to render CT slices which would still be noisy but would have patient specific information embedded in it. Hence when this issue gets addressed in this stage, it is no wonder that the network performs a lot better.

Second of all, during resampling of the CT volume to a normalised dimension for each patient, the Python script takes care of maintaining the isotropy of CT voxels, which MeVisLab simply ignores. That is why there is also a clear difference in appearance of the same original slices for these two

stages. The importance of isotropy becomes vivid when the labels and the outputs are juxtaposed to each other. We can clearly see that even though the network cannot reproduce the slices closely enough, but still formation of initial structuring compared to a random smudge is a major step up when compared to improvement in accuracy. There is of course much more scope of improvement because even though we can see a formation of an outline, but still the transition of pixels between the grey anatomical elements and the black non-anatomical ones is kind of smooth, which is required to be sharp. Also the decomposed slices are far away visually from the original slices. This is because we still have structurally unrelated CT-DRR pairs, as a result of which there has not been a sharp improvement in the accuracy metrics. With reference to the figure A.20, during the process of resampling by MeVisLab, any one factor out of image, size, voxel size and scale factor can be locked. So to achieve isotropy, if the voxel sizes are set to [1, 1, 1], that would mean that either the z or both x and y dimensions of the image would be different. That will throw away the main intention of resampling which is to have same dimension across all 3 coordinates as with changing dimensions (of either inputs or labels) across data entries, it would be impossible to train a neural network. If the image size is kept constant, as can be seen in the diagram, isotropy would be compromised. As we have discussed before that, isotropy provides a means of normalisation across CT slices for the same patient and across different patients and it is an important criteria for our clinical dataset, which have CT scans of patients derived with completely different setups. When MeVisLab failed to address this issue, Python script could successfully achieve it alongside maintaining a constant CT dimension for all patients. In case of pre-processing by script, the ideal voxel size has been set to [1, 1, 1], where the first 2 dimensions represent the voxel and the last one represents slice thickness. This resulted in generating a dataset of isotropically resampled CT volume. This newly achieved isotropy of voxels has made a huge difference for the network while it tries to reproduce the CT slices from test dataset because during learning, it now learns from a normalised data unlike in previous setup.

In this setup we addressed the last two drawbacks, related to pre-processing method, from the first stage. But we are yet to address the issue of semantically unrelated CT-DRR pairs. In the next stage, we are going to address this issue by generating multiple datasets, but all having relatable input–label pairs.

4.3. CT - DRR Combination

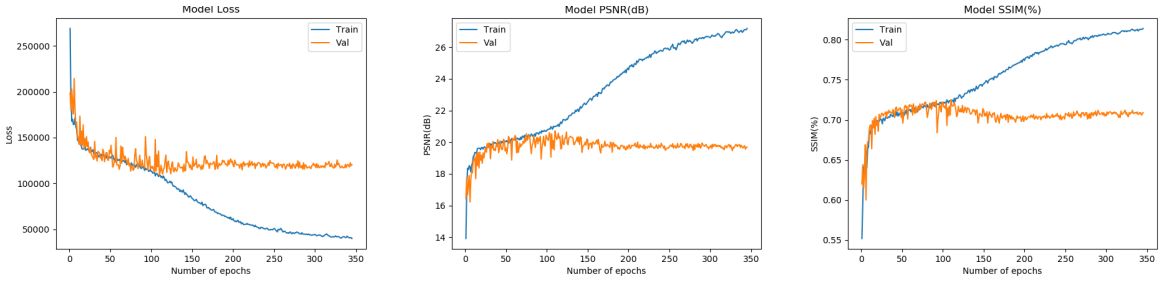
In this stage, visually and structurally related combinations of CT and DRR were generated as input-label pairs by Python script and compared with the winner from the last stage - unrelated top view CT - front view DRR combination. We considered 3 different viewing angles for formation of the related pairs - frontal, lateral and top. The motive was to find out which particular viewing angle would be the best for generation of structurally related CT-DRR pairs.

4.3.1. Frontal CT - Frontal DRR

In this experiment frontal CT volume and frontal DRR image were used as input-label combination.

Training Curves

The loss, PSNR and SSIM curves for training and validation data for this experiment are shown in figure 4.5a, 4.5b and 4.5c respectively.

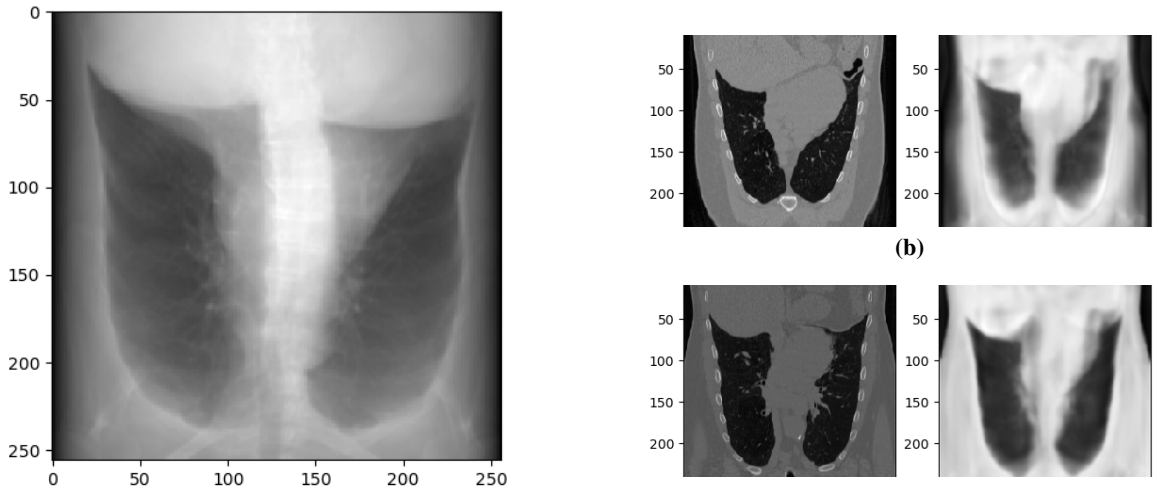


(a) Loss **(b)** PSNR (in dB) **(c)** SSIM

Figure 4.5.: Loss and metric results for stage 3 experiment 1 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

The input DRR used for this experiment is still the same, but we have just used vertically flipped versions of both DRRs and CT volumes from this stage onward and there is no special reason for that. The flipped frontal DRR used for this stage is available in figure 4.6a and the corresponding comparison of CT slices are in figures (b) and (c).



(a) **(b)** **(c)**

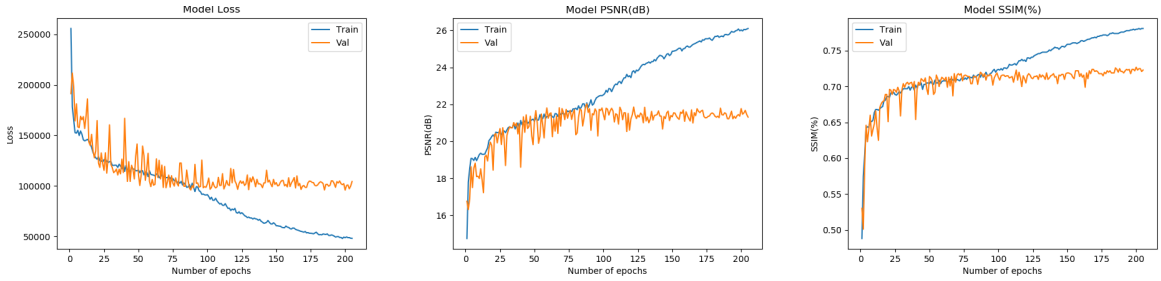
Figure 4.6.: Visual example results for stage 3 experiment 1 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.3.2. Lateral CT - Lateral DRR

In this particular combination vertically flipped lateral CT volume was used along with vertically flipped lateral DRR image as label-input pair.

Training Curves

The learning curves for this section are available in figure 4.7.

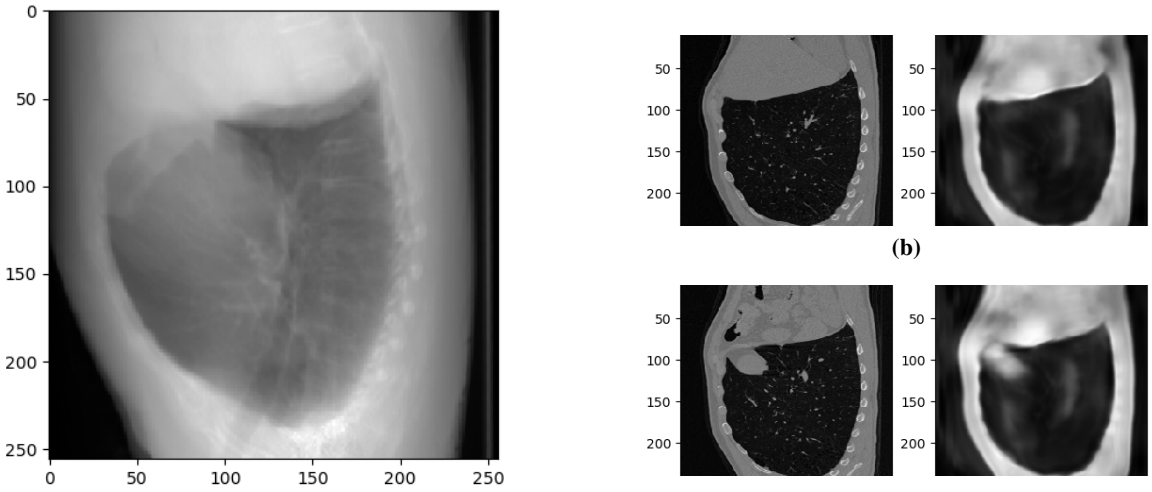


(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.7.: Loss and metric results for stage 3 experiment 2 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

Lateral DRR was fed at the input of the network which was expected to produce a lateral view of the CT volume. The DRR and CT slices comparison for this experiment of this stage are shown in figures 4.8 (a), (b) and (c).



(a) **(b)** **(c)**

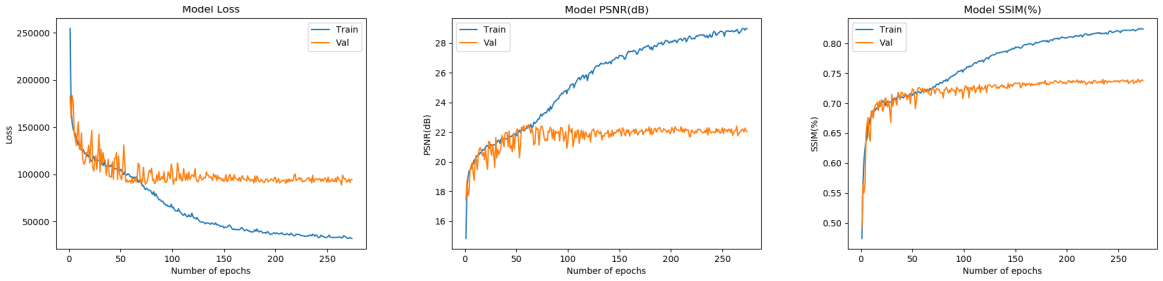
Figure 4.8.: Visual example results for stage 3 experiment 2 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.3.3. Top CT - Top DRR

For this experiment top CT volume and top DRR image were used for training.

Training Curves

Figures 4.9a, 4.9b and 4.9c depict the training curves for this experiment.

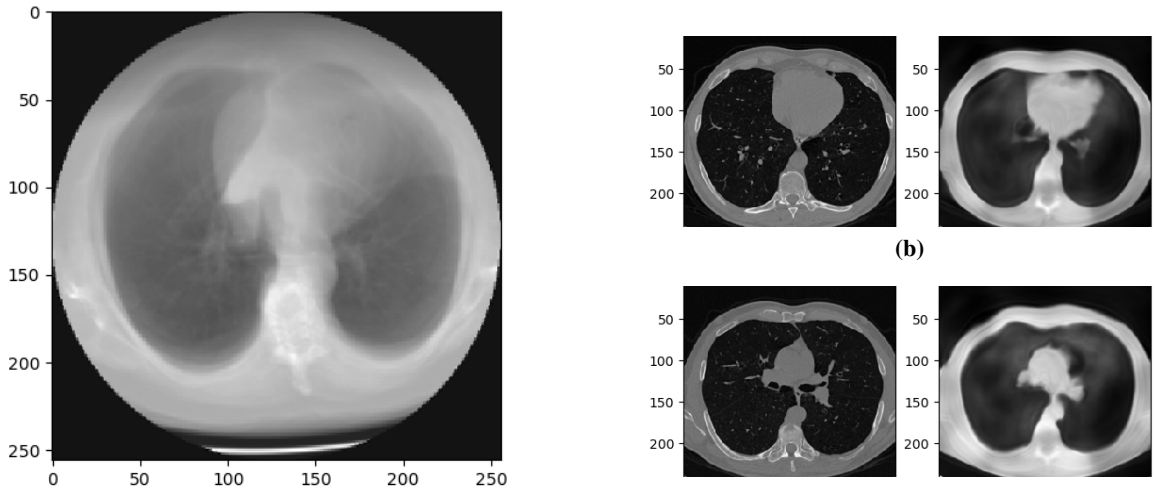


(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.9.: Loss and metric results for stage 3 experiment 3 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

To the input of the network was fed the top view DRR and top view CT slices were expected at the output of the same. The top view DRR and corresponding top view CT slices 115 and 165 for this stage for the same patient 920 are shown in figure 4.10.



(a) **(b)** **(c)**

Figure 4.10.: Visual example results for stage 3 experiment 3 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.3.4. Discussion

The highest achieved SSIM for the frontal, lateral and top view experiments of this stage are 0.725, 0.727 and 0.74 respectively. Likewise the corresponding values of PSNR for these 3 experiments are 20.704 dB, 21.77 dB and 22.27 respectively. Finally the loss values also follow a similar pattern and stand at values 109224.857, 95849.402 and 90918.762 respectively.

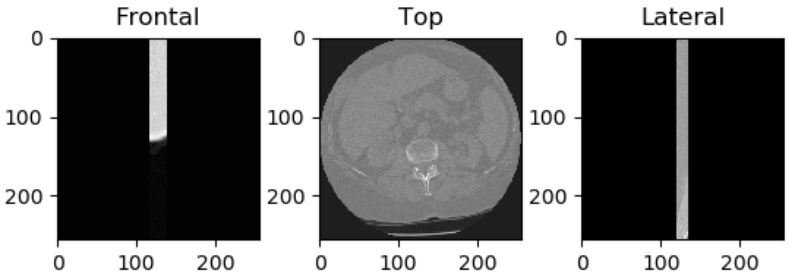
The reason for increased accuracy at this stage for all 3 experiments is nothing but the semantically related input-label combination. The 2D convolutional layers do not look at the volume as a

whole as a result of which it lacks the spatial information across the slices. Instead they look at the individual slice which were initially top view CT slices while they were trying to reconstruct from the seemingly different frontal DRR in the previous 2 stages. Hence now frontal, lateral and top-view DRRs have the necessary information that can be fed to the encoder of the UNet, which enables the decoder to reproduce the decomposed frontal, lateral and top-view CT slices respectively at the output, which look similar to their individual DRRs. The network makes use of the semantic relation and positional correlation to improve the process of decomposition as compared to the previous stages. This explains why all 3 experiments of this stage yielded far more superior results than the last stages.

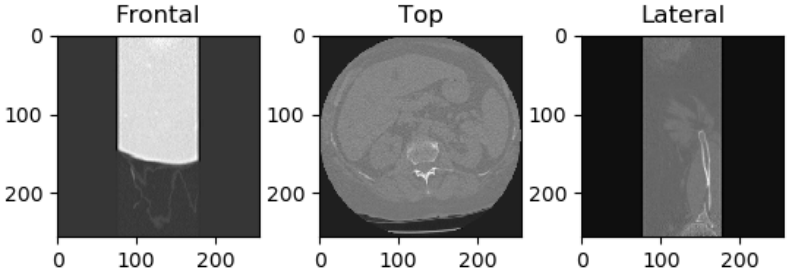
Now let us try to argue why the lateral-view pairs produced better results than front-view ones. Apart from the anatomical components of the human body, the CT volume also has some additional black areas surrounding the anatomy which are nothing but noise and hence will be referred to as noisy border areas in this discussion. The reason for a small increase in accuracy of the lateral pairs over the frontal ones is because of more amount of border areas around the anatomy when the volume is viewed frontally as compared to when viewed laterally. In other words, when the network sees the volume in frontal direction, it sees more slices towards the beginning and also towards the end which comprise of only noisy borders and no components of anatomy as compared to lateral view-point. So with with the lesser number of slices contributing to noisy borders in lateral direction, the network performs better. This is simply because these noisy borders have no useful information worth the task of decomposition and confuses the network, while it tries to solve such a complex problem of reproducing 3D CT scans with 256 dimension from 2D DRRs with a single dimension in the z direction.

Using the same logic, we can explain the further improvement with the top-view pairs compared to the lateral-view ones. The human anatomy extends vertically right from the top till the bottom in the projected CT volume or the DRR, indicating that the top view slices have components of human anatomy right from the first slice till the last one in the non projected CT volume. We found out in the previous comparison that the network performs better with lateral-view pairs which have lesser number of slices with low anatomical content and more noisy border areas. Going by the same logic, with top-view pairs, which have absolutely zero slice with no anatomical content and completely noisy border areas, it performs even better than the lateral-view pairs.

With respect to the figures 4.11a and 4.11b which capture the 1st and the 10th CT slices of every viewing angle for patient 920, there is well structured anatomical content in the top view, whereas the lateral view has relatively much lesser content of anatomy. The anatomical content becomes even lesser in frontal view. The anatomical content increased from 1st to 10th slice in both frontal and lateral views, but in the same proportion as that of the 1st slice. When well structured anatomical parts start from slice 1 in top view, the same starts from near about 35 and 45 in lateral and frontal views respectively for this particular patient. As we have already mentioned before, since more amount of noisy borders confuses the network with useless information, while it tries to solve a complex problem, hence that explains the slight improvement in accuracy in every stage where those borders also keep decreasing slightly.



(a) Frontal, Top and Lateral CT slice 1



(b) Frontal, Top and Lateral CT slice 10

Figure 4.11.: Frontal, Top and Lateral CT slices (patient 920). (a) shows slice number 1. (b) shows slice number 10

Hence two things are clear from this stage. First one is that any structurally related pair of CT-DRR will yield better results than the unrelated ones due to the ability of the network to find the positional correlation and the semantic mapping. Secondly, even within the related pairs, the pair which has lesser number of slices with noisy border areas, will perform better because of less confusion and more clarity thrown at the network from the dataset. So the dataset, pre-processed by Python script, containing top view DRR and top view CT volume as input-label pair, turns out to be a clear winner of this stage. In the next stage, we are going to test the parameter of view-points of DRRs to find out if additional view-points can help to improve the decomposition process even further or not.

4.4. View Point

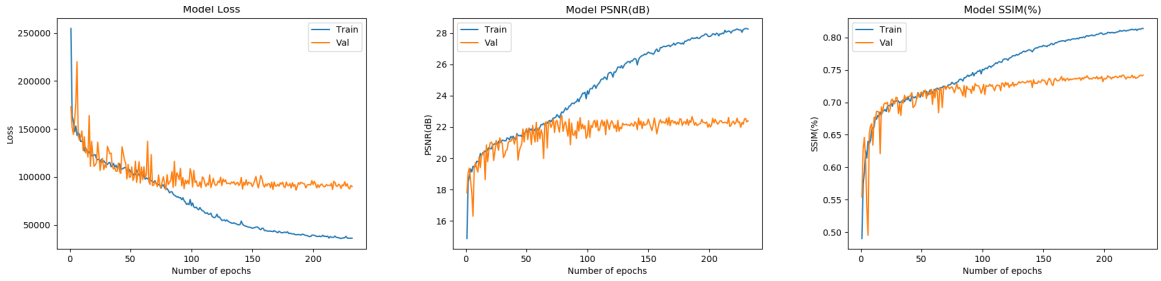
In this section we will try to find out in two different experiments, whether addition of a second (frontal) and afterwards a third (lateral) view DRR helps in the process of evaluation.

4.4.1. Top+Frontal

In this experiment, we will be feeding into the network, both top and frontal DRRs and try to reproduce top view CT volume at the output.

Training Curves

The 3 different curves for this experiment are presented in diagrams 4.12a, 4.12b and 4.12c respectively.

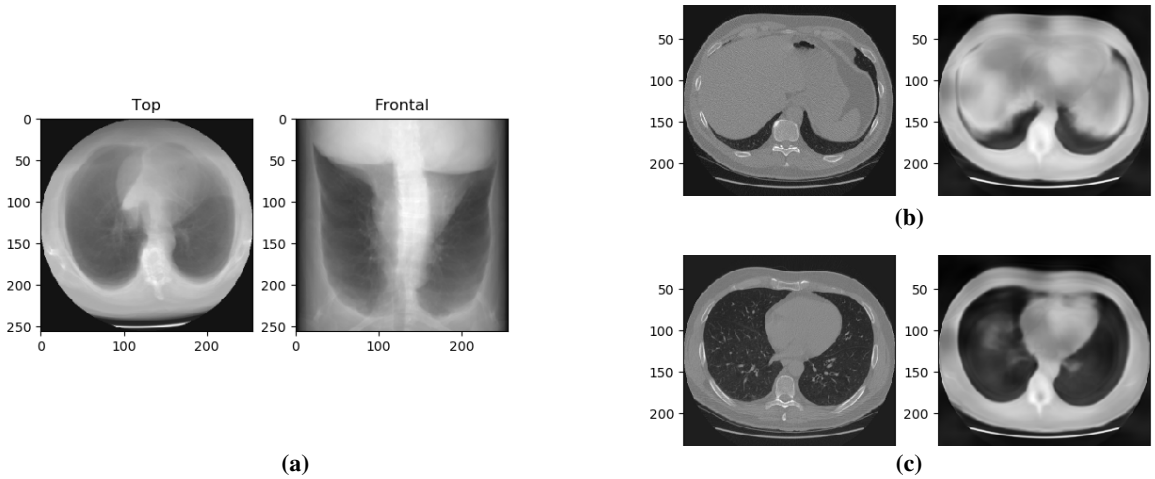


(a) Loss **(b)** PSNR (in dB) **(c)** SSIM

Figure 4.12.: Loss and metric results for stage 4 experiment 1 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

We have already put together the frontal and the top-view DRRs in the figure 4.13a and the corresponding comparison among decomposed and original slices in 4.13b and 4.13c.



(a) **(b)** **(c)**

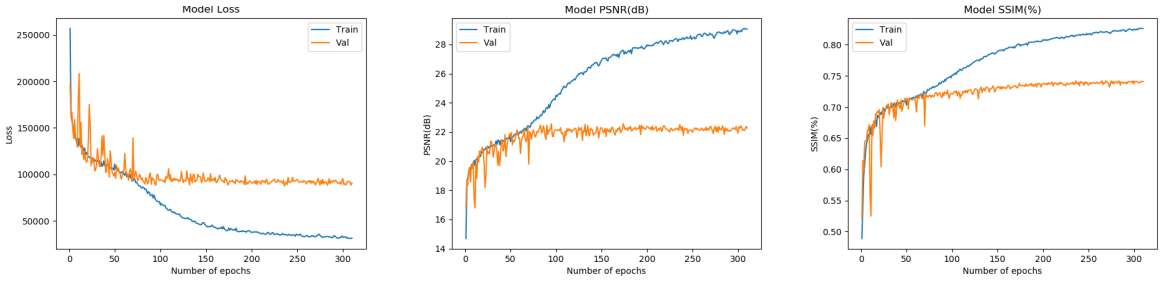
Figure 4.13.: Visual example results for stage 4 experiment 1 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.4.2. Top+Frontal+Lateral

For this one, all 3 view points of top, frontal and lateral DRRs will be used as input and top view CT slices will be used as label.

Training Curves

The loss, PSNR and SSIM curves for this experiment are presented in diagrams 4.14a, 4.14b and 4.14c respectively.



(a) Loss **(b)** PSNR (in dB) **(c)** SSIM

Figure 4.14.: Loss and metric results for stage 4 experiment 2 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

The triple view DRRs which have been used for this stage are shown in figure 4.15a. Figures 4.15b and 4.15c show the CT slices.

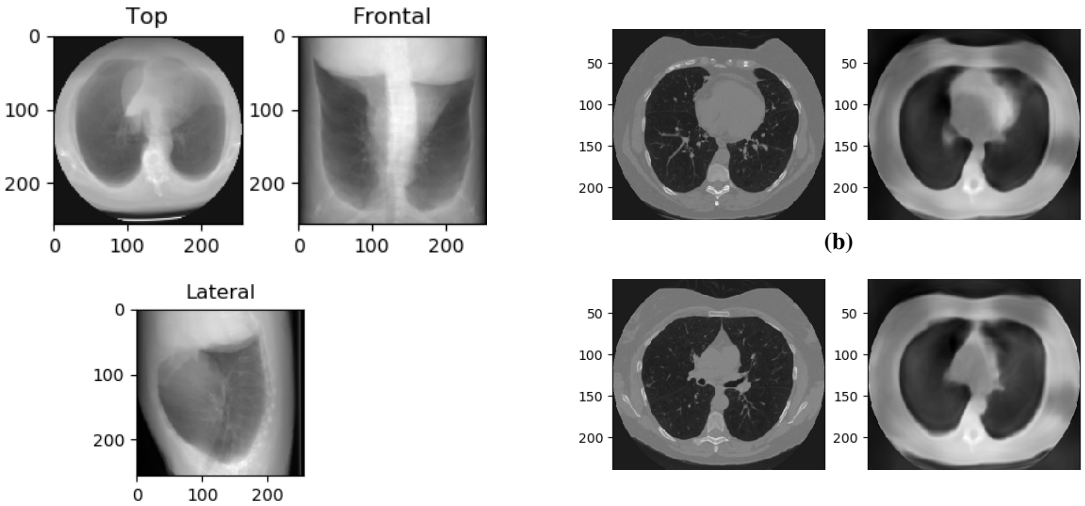


Figure 4.15.: Visual example results for stage 4 experiment 2 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.4.3. Discussion

The SSIM, loss and PSNR values stand at 0.741, 93246.988 and 22.227 dB respectively for the first experiment and at 0.742, 87643.957 and 22.497 dB respectively for the second experiment.

Although in terms of loss and PSNR, the results have deteriorated a bit in the first experiment compared to the last stage, but in terms of SSIM, it has shown better results. But we can see an improvement in all the three metrics in the second experiment when compared with the first experiment or with the last stage. Since we are concerned primarily about the SSIM metric as it focuses on the visual

structural similarity, so we will base our arguments based on the same.

This small increase in SSIM metric in course of these two experiments will not be visible in the network outputs but will still be considered as a small step up from the last stage. The reason for such an insignificant increase in SSIM metric of the first experiment over the experiment from the last stage or of the second experiment over the first experiment of this stage can be attributed to the fact, every time an additional view point is providing some perspective in shaping the latent space information of the encoder which in turn is helping the decoder in it's process of decoding. Hence even though the impact of more than one view point is not immense, but it does help a tiny bit to improve the process of evaluation on unseen test data. We know that the whole 3D CT volume gets collapsed in a single 2D plate while generating the DRR. Hence the DRR captured from a particular view point will always miss out on the depth information along a particular axis, which depends on the view point that we have considered. If we consider the length and breadth of a front-view CT volume as the X and Y axes respectively and it's depth as the Z axis, then the top view DRR will lack the depth information along the X axis whereas the front view DRR will have the same knowledge gap along the Z axis. Similarly, lateral view DRR will miss out on the depth information along the Y axis. These missing depth information are partially compensated by a very small amount every time we add one more view point in the input DRRs.

But intuitively, adding more view points should have improved the accuracy by a considerable amount. It fails to do so for the following reason. Even though we are providing additional information along different axes, these information get cascaded in the input just as additional input channels. In other words, the network definitely receives extra information along different dimension but it does not know how to read these separate channel information with respect to one another as it does not know the relationship among these input channels. If there was a way to not only provide additional information to the network for more than 1 axis through more than 1 input channel, but also to specify the relationship among these input channels, then we could have expected a considerable jump in the metrics. With our current setup it is not possible and to achieve that we either need to have a different network architecture and/or feed the data to the network in a completely different setup, We have discussed different state-of-the-art networks in Future Work, which also processes the input data in a different manner than our current setup.

Hence we realised that if we keep on adding more and more view points, it will help the network to interpolate the whole volume better at every step. But it is kind of impractical for two primary reasons. More number of X-rays will increase the expenses for the patient and will also expose him/her to an excess amount of radiation. These were also the same challenges that we wanted to overcome with our thesis task and we do not simply want to go back on our primary objectives. Hence we stop with triple-view DRRs and move ahead with it along with top view CT volume as the winning input-label combination from this stage to the next one, where we test the impact of data augmentation on accuracy.

4.5. Data Augmentation

For this stage, data augmentation was applied on the winning combination from the last stage.

4.5.1. Augmentation

Since data augmentation did not improve the SSIM metric over 0.742, so results have been skipped for this stage and we will concentrate only on the discussion.

4.5.2. Discussion

When the training dataset is small and it does not have an enough representation of all the possibilities of real world data, data augmentation helps to avoid overfitting by virtually increasing the size of it. In other words, when the network keeps seeing more and more data, it fails to fit into the specifics of such a complex ideally infinite dataset and starts generalising.

With less number of channels in the hidden layers, a network can try to learn only from a dataset of limited size. When the size of the dataset keeps increasing but not the channel dimensions in the hidden layers, then the network struggles to learn even from the training data - situation known as underfitting. So when you have an ideally infinite dataset to remember, performance increases rapidly with increased number of channels in hidden layers.

In our setup, we have used a maximum channel dimension of 3097 in the last layer of encoder and beyond that, the system threw memory error. So we have used the optimum channel dimension what the system could handle. Even with that high number of channels it took almost 600 epochs for the network to reach a training accuracy of 85% and would require near about 500 more epochs to achieve more than 95%. Without data augmentation and with optimum channel dimension, the setup is already an underfitted one due to the problem being extremely complex of interpolating from a single z dimension to a dimension of the size of 256. A setup which is underfitted will tend towards an increased amount of underfitting with data augmentation, because now we a larger volume of complex data to remember but still the same lesser of hidden nodes. But since we have reached the maximum system capacity and we cannot increase the hidden channel dimensions further so the data augmentation approach is confusing the network more than helping it in solving it's complex problem of interpolation along 1–>256 channel dimension.

So with data augmentation not helping in the current setup and only increasing the GPU load and the overall training time, we will not include it in our training from the next stage onward. Next we want to check if tweaking the network architecture can bring in some positive influence on the validation accuracy or not.

4.6. Loss

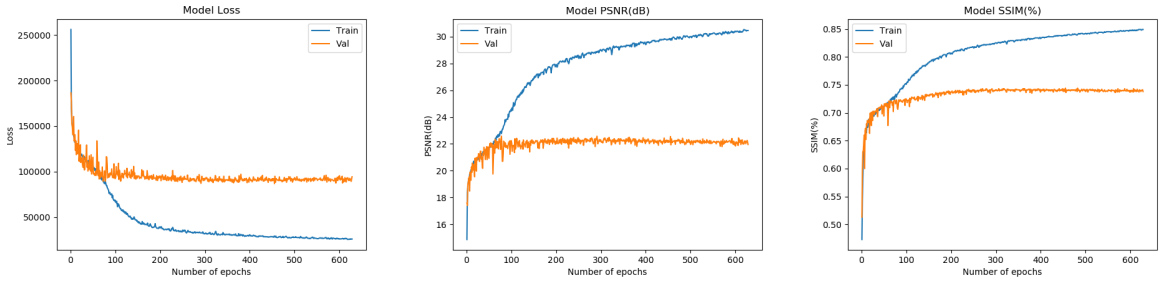
Till this stage, we have only been using the decomposition loss, calculated between the CT slices, decomposed by the network and the CT slices which were derived as labels, post resampling and processing by Python script. For this stage we have considered reconstruction loss along with the decomposition one for one experiment and latent space loss on these two losses for another experiment.

4.6.1. Decomposition + Reconstruction

Until now the network was trying to decompose the single/double/triple view DRRs into a CT volume of size $256 \times 256 \times 256$ pixels. But now we have added an additional step of trying to reconstruct the triple view DRRs of dimension $3 \times 256 \times 256$ from the decomposed volume. We have already discussed how the decomposition loss is calculated. The reconstruction loss is calculated between the reconstructed triple-view DRRs and the ones which were fed as input into the network.

Training Curves

The learning curves for this experiment are presented in diagrams 4.16a, 4.16b and 4.16c respectively.

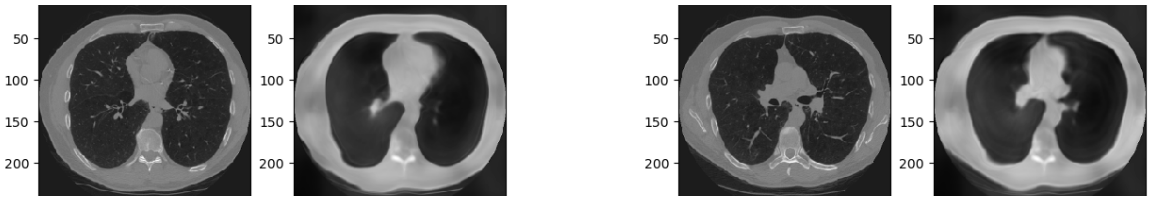


(a) Loss **(b)** PSNR **(c)** SSIM

Figure 4.16.: Loss and metric results for stage 6 experiment 1 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

Since we have already seen the triple view DRRs in the last stage, so we will look directly into the CT slices in figures 4.17a and 4.17b.



(a) **(b)**

Figure 4.17.: Visual example results for stage 6 experiment 1 (patient 920). (a) and (b) show a comparison between the original and decomposed CT slices.

4.6.2. Decomposition + Reconstruction + Latent Space

For this experiment of this stage, we also took into account another loss, latent space loss, apart from the decomposition and reconstruction loss. For obtaining this loss, at first a network, also referred to as latent space network, needs to be trained based on reconstruction of 3D CT volume from itself. Once the network is trained to a sufficient level, which is 93% for our case, then we train our main network, where we decompose triple view 2D DRRs into 3D CT volume and then reconstruct the

same DRRs from the decomposed CT volume. We calculate the decomposition and the reconstruction losses in the same ways that we have already discussed. Additionally, we calculate the latent space loss between the encoder outputs of the primary and the latent space network, while putting the latent space network into evaluation mode and the primary network into training mode.

But since this loss did not improve the result over the last experiment of reconstruction loss on top of decomposition, so we have skipped plotting the results for this section and will dive straight into the discussion for this stage.

4.6.3. Discussion

For the first experiment, there has been a little increase in the value of SSIM and it stands at 0.744 with the corresponding values for loss and PSNR being 86805.614 and 22.55 dB respectively. And we have already mentioned that the latent space loss did not contribute at all in improvement of the SSIM metric by even a marginal amount in the second experiment.

When the decomposed volume is reconstructed into the triple view input DRRs, the reconstruction loss also backpropagates along with the decomposition loss. This helps to fine-tune the weights one step further compared to what it was doing during decomposition loss. It clearly means that when we try to reconstruct the input to the network from the decomposed volume, it gives a little more perspective to the network on how to decompose it better compared to the baseline network. This explains the reason behind this small increase in SSIM value.

No improvement of results with latent space loss can be explained in similar line as that of reconstruction loss. The latent space network is not able to provide any additional perspective to the primary network, which is useful for the decomposition task. Hence the encoder of the primary network already has the optimum information it needs on how to accomplish the decomposition process from it's own training.

Since reconstruction loss contributed to 0.004 increase in accuracy, it will be a part of the training process from this stage onward. Since in latent space network we have to at first train the latent space network before training the primary network, so it only increases the training time with no positive impact on accuracy. So we will not consider this loss for next stages. In the next stage, we will try to find out if we can achieve accuracy improvement with a different network architecture or not.

4.7. Network

Till now we have tested multiple setups using the same network architecture of 2D UNet. For this stage, we wanted to test if a different network architecture could have a positive impact on the accuracy metrics or not. So we decided to use 2D(encoder)-3D(decoder) UNet in this stage.

4.7.1. 2D-3D UNet

In a 2D UNet, the 2D convolution layers look at the individual CT slices separately. But these 256 slices are not all stand alone slices having no relation among each other. Since these slices make up a volume, there is always a certain order (order of human anatomy) in which they will appear which

follows from the spatial connection among them. It is extremely important for a neural network to understand that context. In a separate context where we use Machine Learning to translate a complete sentence from one language to another, if a neural network looks at the words individually, it would still be able to translate word-by-word successfully, but if it misses out on the context of sentence formation, the translated set of words would not make any sense when put together in a sentence.

With 2D-3D UNet, the 2D encoder acts similarly as that of a 2D UNet. Following the encoder, the transformation module redistributes the spatial features in a 3D feature space. But unlike the 2D decoder of 2D UNet, which simply tries to fine-tune each of channel's weights separately to minimise the loss, the 3D decoder here also takes into consideration the spatial relation among the channels.

Intuitively, we understand that this network is supposed to perform way better than the basic 2D one. But we did not see any improvement visually. So we will avoid showing the results and will jump straight into the discussion.

4.7.2. Discussion

The validation SSIM score did not shoot above the last achieved best score of 0.744.

With reference to the section 3.1.4, we could use a maximum channel dimension of 3072 while training a 2D UNet, without encountering a GPU memory error, using the first system configuration. But while training the 2D-3D UNet that number was 128. This indicates how much of resource (GPU memory and computation power) is consumed by 3D convolutional layers. So it is evident that this network did not perform at it's optimum level, as it did not have the width which is required for solving such a complex task. To see the optimum performance of this network, we require a system with much higher configuration. Hence this has been put as a part of the possible future work in the next chapter.

In the next stage we will be testing the dimension parameter and one of it's test cases will involve testing the dimension of 512 pixels, which is already taxing enough. As a result, since we did not have the optimum extract from the network in this stage due to system configuration limitations, so we move on to the next stage with our older setup using 2D UNet, as we want to avoid unnecessary taxing of the system.

4.8. Dimension

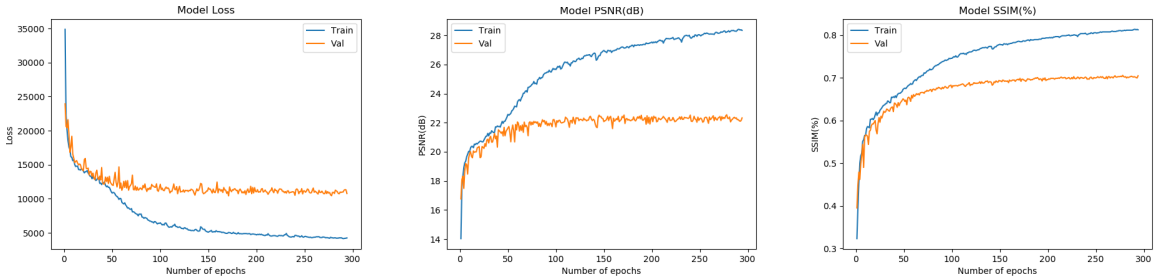
In this section, we want to find out how the accuracy changes with changing dimensions of the input-label pairs. This is also the last parameter that we are going to test. Till now a CT volume of $256 \times 256 \times 256$ pixels a was used as label. Likewise we have used triple view DRRs of $3 \times 256 \times 256$ pixels as input. Now in this stage, 2 different experiments will be conducted, the first one with 128 pixels and the second one with 512 pixels.

4.8.1. Dimension 128

Triple-view DRRs are fed at the input. Hence the input is of dimension of $3 \times 128 \times 128$ pixels and the label is of the dimension of $128 \times 128 \times 128$ pixels.

Training Curves

Let's have a look at the loss, PSNR and SSIM curves for this experiment from this stage in diagrams 4.18a, 4.18b and 4.18c respectively.



(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.18.: Loss and metric results for stage 8 experiment 1 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

Although we have seen the triple view DRRs before, still let's again have a look at it with reduced dimensions in the figure 4.19a and also at it's corresponding slices in figures 4.19b and 4.19c.

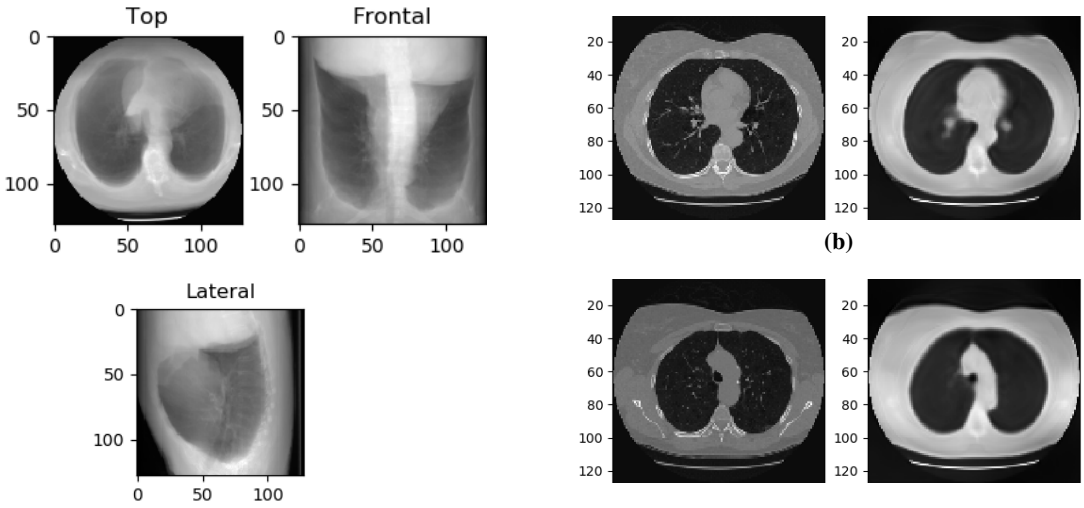


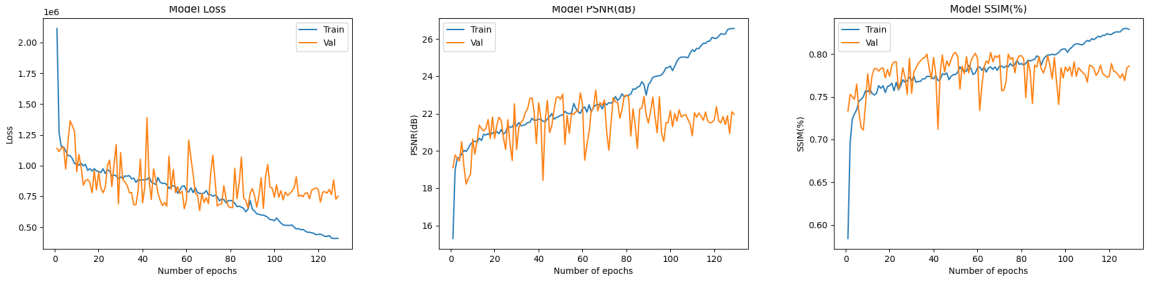
Figure 4.19.: Visual example results for stage 8 experiment 1 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.8.2. Dimension 512

For this experiment, we have chosen a CT volume of $512 \times 512 \times 512$ pixels as label and triple view DRRs of $3 \times 512 \times 512$ pixels as input.

Training Curves

Diagrams 4.20a, 4.20b and 4.20c depict the learning curve for this experiment.



(a) Loss **(b)** PSNR (in dB) **(c)** SSIM

Figure 4.20.: Loss and metric results for stage 15 experiment 2 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

We have already seen the triple-view DRRs with two different resolutions in 2 previous experiments. So let's just have a look at the original and the decomposed slices in figures 4.21a and 4.21b.

(a) **(b)**

Figure 4.21.: Visual example results for stage 8 experiment 2 (patient 920). (a) and (b) show a comparison between the original and decomposed CT slices.

4.8.3. Discussion

The SSIM scores for the first and second experiments are 0.706 and 0.802 respectively and the corresponding PSNR values are 22.338 dB and 22.767 dB respectively. Although this increase in accuracy is not visible to human eyes distinctly in the network outputs, it is still a big step up from the last stage whose best validation SSIM score was 0.744. But if we try to look closely, we can see that the outer structuring is a bit more prominent in the generated slices with the 512 dimension compared to the slices with 256 or 128 dimension. The inner detailing still remains the same to bare human eyes. But it is quite evident with the slices of dimension 128 that it has not only compromised on outer structuring, but has also lost a lot on inner detailing of the slices.

This increase in accuracy with the increase in resolution and vice-versa can be explained with reference to one of the ways (training data) mentioned in section 2.7 as a means to counter overfitting. It clearly stated that increase in information helps to counter overfitting by forcing a neural network to start generalising.

Hence it is clear that with increase in resolution the neural network receives a lot more information, which helps it to generalise much better on the unseen test data and that reflects both in the training curves and also in the generated slices. Since this was the last parameter that we tested, so we can declare the setup of this current stage with the dimension of 512 pixels as the final winning combination.

But in-spite of the current setup being the final winning combination, there is one unrealistic factor associated with it. Hence in the next stage we want to get rid of that factor and find out the realistic winners apart from the idealistic ones from this stage and in doing so we do not change any other aspect of the current setup.

4.9. Miscellaneous

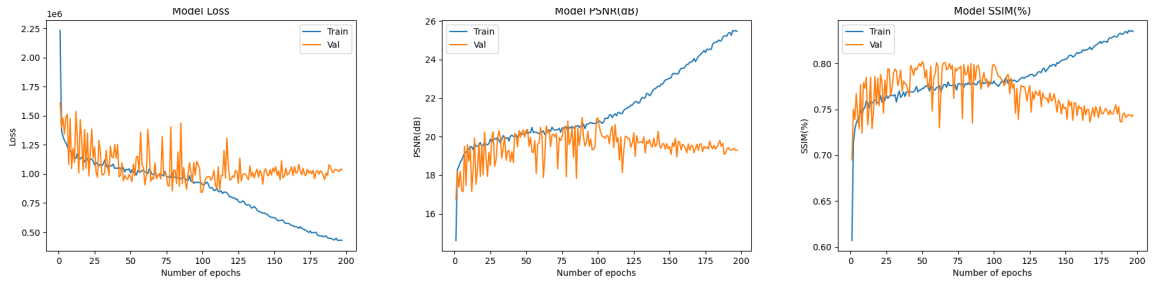
We have found the ideal combination and setup till this stage which yielded the best results. That ideal combination uses triple view DRR as input. But as we have discussed earlier, top view DRR/X-ray is not feasible with current health care system. Also if a patient has to obtain 3 different X-rays, he/she might be subject to more radiation and also spend more than what he/she would with a single view DRR. So in this subsection, we retest one of the experiments which we already conducted in experiment 1 in stage 3 where we try to generate frontal CT slices from frontal DRR, but this time with the ideal combination, which we explored through different stages, and in another experiment, we input an additional lateral view DRR to the network too.

4.9.1. Frontal DRR –> Frontal CT + Ideal Combination

In this experiment we kept the ideal combination in place and we just changed the input to simply frontal DRR, instead of triple views. Apart from that, we used the dimension 512 in contrast with 256 (from 3rd stage) for input-label combination, as that turned out to be a part of the winning combination from the last stage.

Training Curves

The learning curves for this section are available in 4.22a, 4.22b and 4.22c.

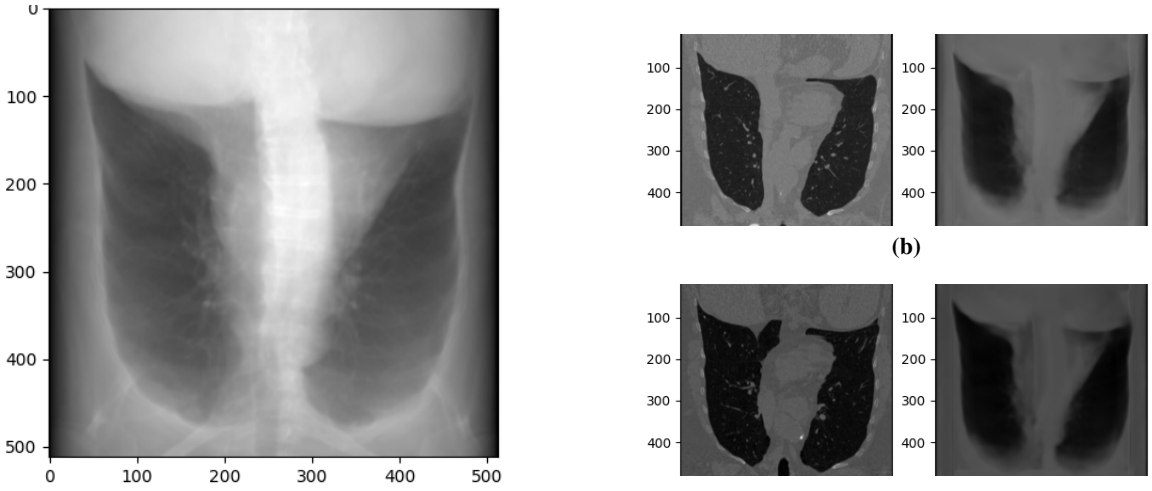


(a) Loss **(b)** PSNR (in dB) **(c)** SSIM

Figure 4.22.: Loss and metric results for stage 16 experiment 1 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

Let's visualise the frontal DRR of dimension 512×512 pixels in figure 4.23a and comparison between original and decomposed slices in figures 4.23b and 4.23c.



(a) **(b)** **(c)**

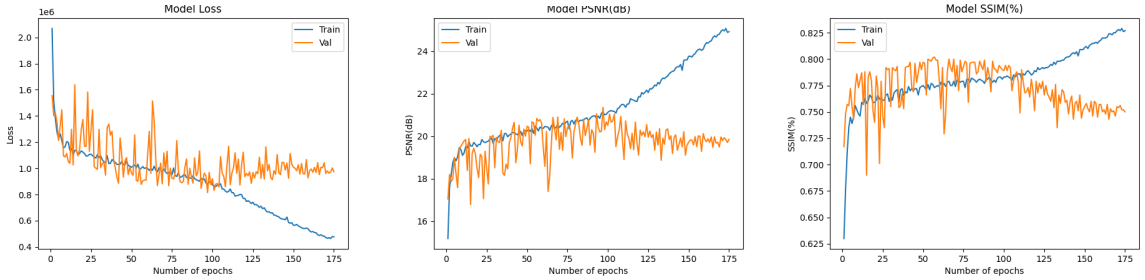
Figure 4.23.: Visual example results for stage 9 experiment 1 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.9.2. Frontal + Lateral DRR –> Frontal CT + Ideal Combination

For this experiment we just added an additional lateral view DRR to the input and rest of it still remains the same.

Training Curves

The learning curves for this section are available in 4.24a, 4.24b and 4.24c.

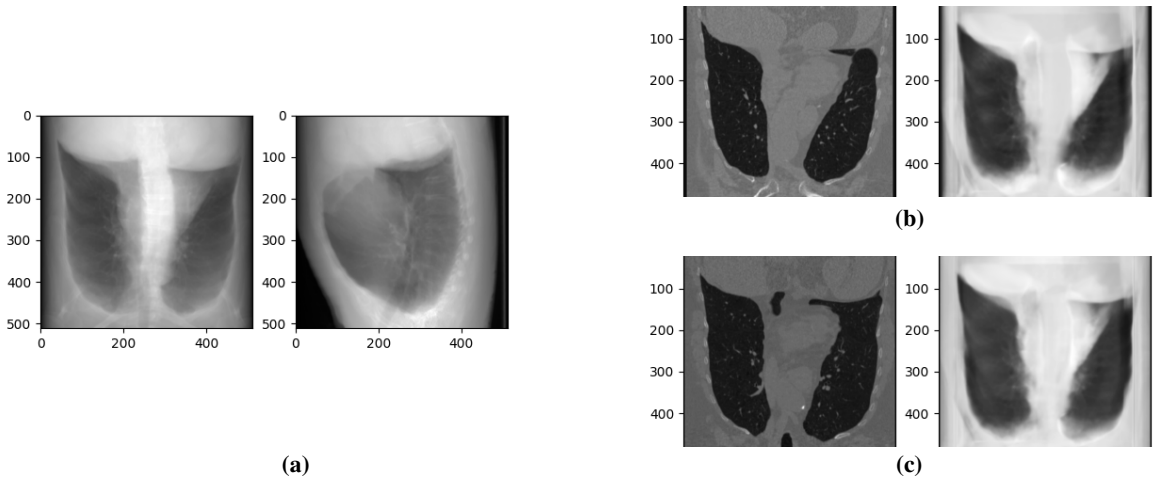


(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.24.: Loss and metric results for stage 9 experiment 2 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

The double view DRR with increased resolution is shown in figure 4.25a and the corresponding slices in 4.25b and 4.25c.



(a) **(b)** **(c)**

Figure 4.25.: Visual example results for stage 9 experiment 2 (patient 920). (a) shows the input DRR. (b) and (c) show a comparison between the original and decomposed CT slices.

4.9.3. Discussion

The SSIM scores for both these experiments are same and it is 0.802. The PSNR values for these two stand at 21.308 dB and 22.714 dB respectively.

The improvement of accuracy for this stage over stage 3 for the same experiment have already been explained in later stages post stage 3. It is basically due to the changing winning combination which got evolved with further stages. The two main contributors to improvement are basically the introduction of reconstruction loss and also an increased resolution. The reasons for contribution for each of these two factors have already been discussed in stage 6 and 8 respectively. Unlike in stage 4, an additional view did not contribute to any improvement in metrics. But visually results look a little better with double view input DRR. So we can declare the second experiment from this stage to be the winner of the realistic setups.

4.10. Subsequent Evaluations

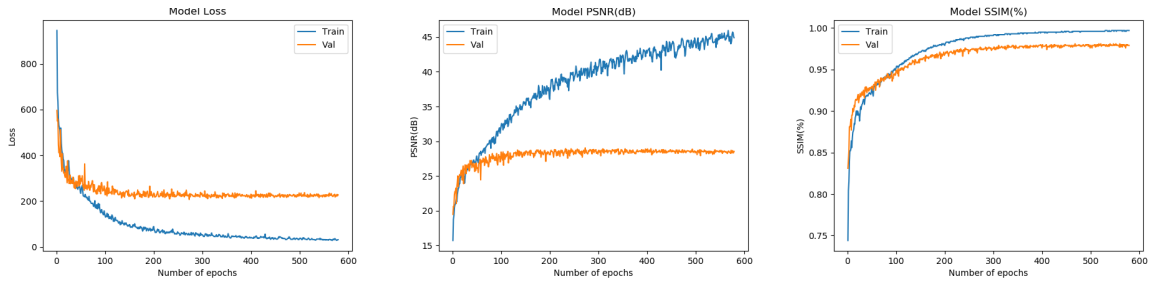
We wanted to research at this stage that if our setup is performing as per expectation for the typical tasks (like denoising, single/triple channel image reconstruction) for which the network has been exclusively designed. Then that will reinstate that there is no flaw in our current setup, but the task itself is extremely complicated for this particular setup compared to it's typical tasks. Hence this is the optimum that we can expect with our setup and the only way to improve is to explore different network architectures or different ways of processing or inputting data into the network.

4.10.1. 3D CT –> 2D DRR

The whole report describes different ways and approaches to achieve one final goal of generating 3D CT volume from single/double/triple view DRRs. But in this subsection we will try to achieve just the opposite - generate single view frontal DRR from CT volume.

Training Curves

The learning curves for this section have been listed out in 4.26a, 4.26b and 4.26c depict the learning curve for this experiment.



(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.26.: Loss and metric results for subsequent evaluations experiment 1 (patient 926). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

Let's compare the original and the generated frontal DRRs in the figure 4.27.

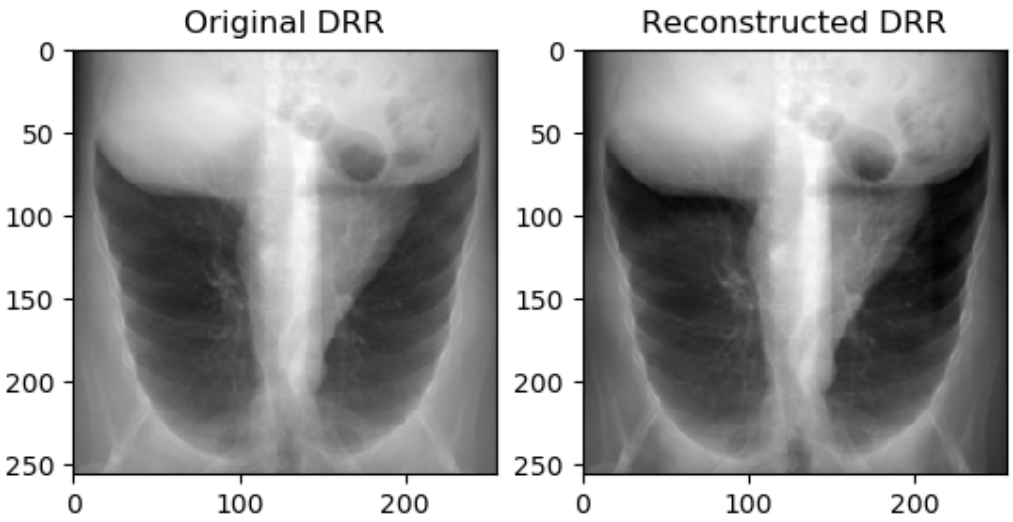


Figure 4.27.: Visual example results for subsequent evaluations experiment 1 (patient 926). It shows a comparison between the original and reconstructed frontal DRR.

Discussion

Validation SSIM for this experiment is 0.981 and loss and PSNR values are 217.8 and 38.549 dB respectively.

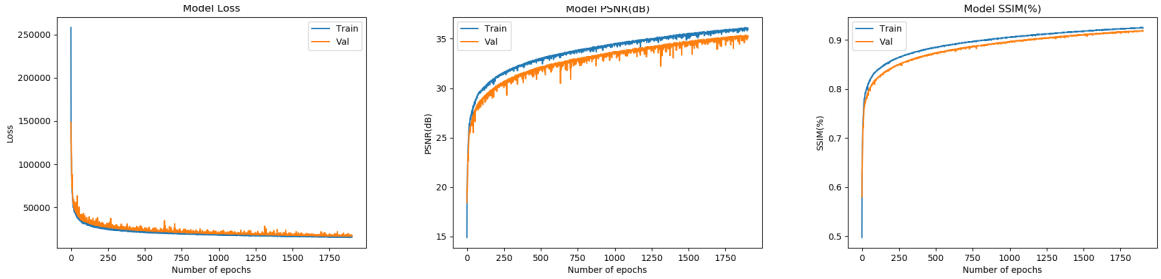
It is much easier to interpolate from dimension of higher order (256) to lower one (1) than the opposite and that is evident in the metric values. This follows from the fact that the appearance of DRR in a single 2D image for different persons do not bring in a lot of variations, compared to the individual CT slice of a patient. For example, the anatomical component of the CT volume might be placed in different set of coordinates for different patients with respect to the whole volume. For first patient the thorax volume might stretch from 100-200, 150-230 and 120-220 pixels along x, y and z axes respectively. Rest of the pixels might be simply black borders or zero padding. These dimensions maybe different for a different patient and it depends entirely on the setup decided by the radiologist while obtaining the CT scan. Additionally, on a different context, the slice number 190 for an underweight patient (or a patient with athletic build) might correspond to a section of spine whereas the same for an overweight patient (or a patient with muscular build) might be a portion of the vasculature. So we can understand that there is a lot more chance of variation in inner detailing in an individual CT slice across patients. But when that entire volume is projected on to a single 2D plate for obtaining a DRR/X-ray, those minute details get collapsed/obscured in the projection. Although there will be patient specific footprints in the DRRs too, but intuitively those every single minute variation is not captured in the 2D DRR, which makes it much more generic and easy to reproduce by a neural network.

4.10.2. 3D CT –> 3D CT

In 6th stage, to analyse the impact of latent space loss, we ran one experiment in which we tried to generate the 3D volume of 256 pixels from itself. We will present the results in this section so as to prove the same point that we tried to in the last subsection. The point is like in the last experiment, where we saw it is really easy for the network in terms of accuracy and training time to reconstruct the frontal 2D DRR from the 3D CT volume, we will see a similar phenomenon in this stage too.

Training Curves

The loss, PSNR and SSIM curves are shown in the figures 4.28a, 4.28b and 4.28c.

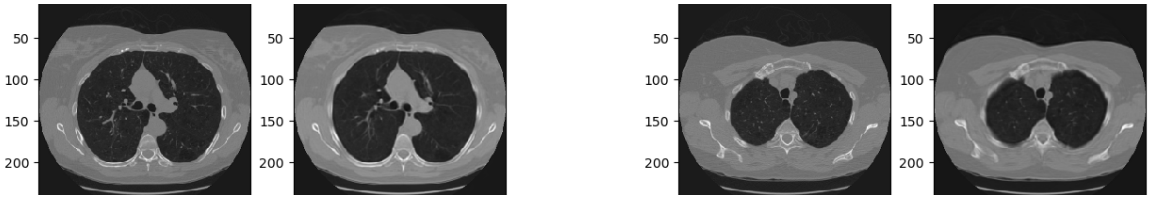


(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.28.: Loss and metric results for stage subsequent evaluations experiment 2 (patient 920). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

The original and reconstructed slices for patient 920 are available in the figure 4.29.



(a) **(b)**

Figure 4.29.: Visual example results for subsequent evaluations experiment 2 (patient 920). (a) and (b) show a comparison between the original and decomposed CT slices.

Discussion

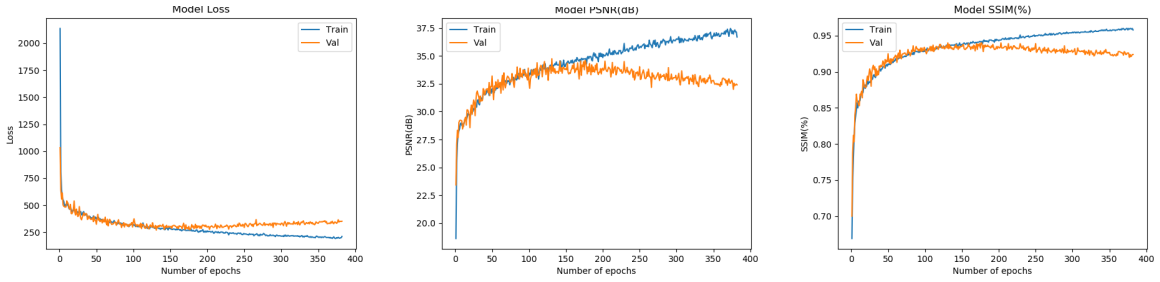
Validation SSIM for this experiment is 0.919 and loss and PSNR values are 17072.163 and 35.316 dB respectively. It is evident from the learning curves that these numbers would have improved even further both for training and for validation data, but the training has been stopped after the network has achieved considerable level of accuracy. Since the training involved more than 1750 epochs, so beyond this, the system would give a memory error. These numbers are not as promising as the last experiment thereby proving that it is a little more difficult of a task than reconstruction of a single channel image. But still it is way easier than reconstructing the volume from 2D DRR.

4.10.3. Triple-view Noisy DRRs –> Triple-view Noiseless DRRs

We do one final experiment in this subsection, where we try to reconstruct the triple-view DRR from a noisy version of itself - a typical usecase for which the network has been designed.

Training Curves

The loss, PSNR and SSIM curves are shown in the figures out 4.30a, 4.30b and 4.30c.



(a) Loss training curve **(b)** PSNR training curve **(c)** SSIM training curve

Figure 4.30.: Loss and metric results for subsequent evaluations experiment 3 (patient 945). (a) shows the loss over the training epochs. (b) shows the PSNR and (c) shows the SSIM.

Visual network performance

The original. noisy and reconstructed versions of each of the 3 views of DRRs -frontal, lateral and top - are available in the figures 4.31a, 4.31b and 4.31c respectively.

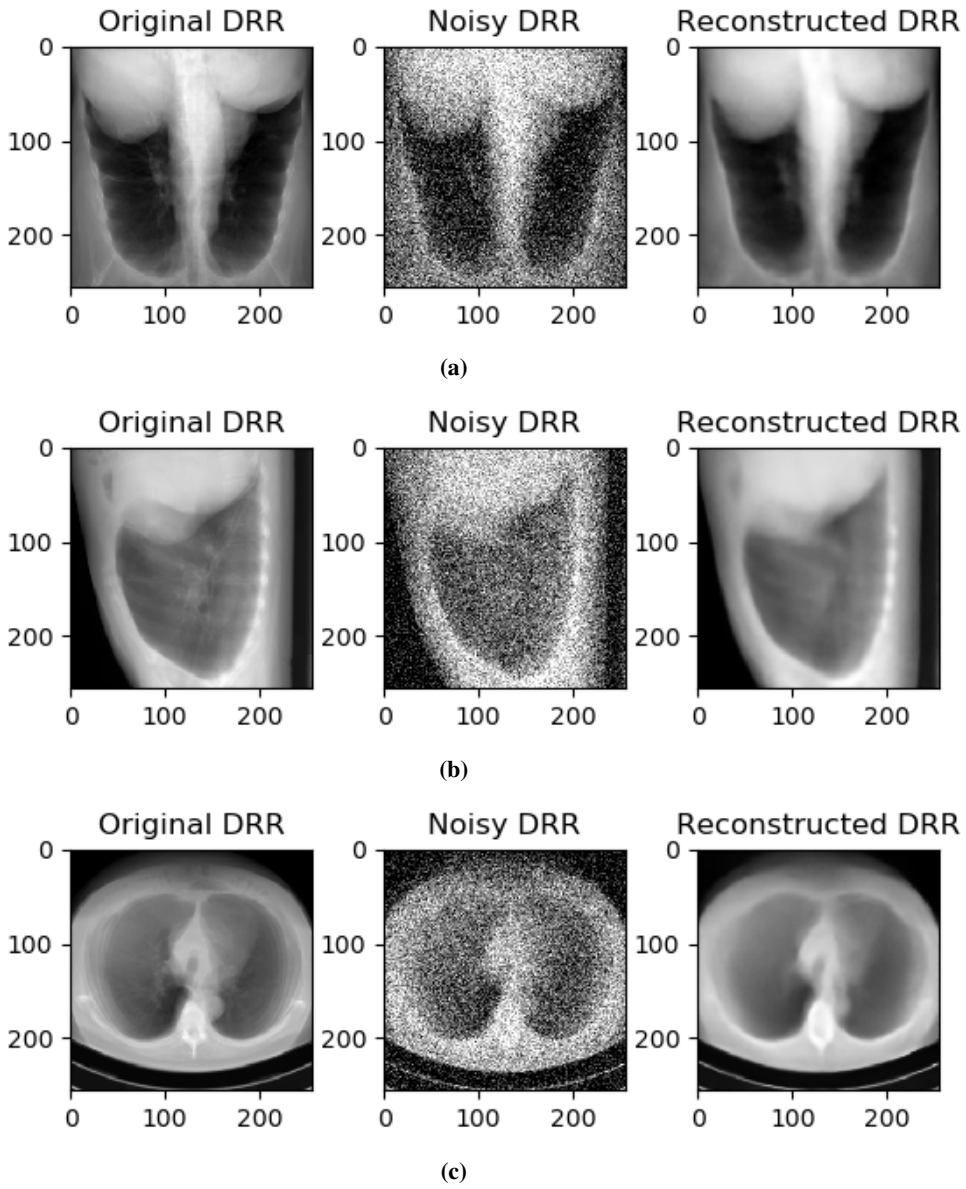


Figure 4.31.: Visual example results for subsequent evaluations experiment 3 (patient 920). (a) shows a comparison among the original, noisy and reconstructed frontal DRRs, (b) shows a comparison among the original, noisy and reconstructed lateral DRRs and (c) shows a comparison among the original, noisy and reconstructed top DRR.

Discussion

With reference to the DRRs presented in the results section, we trained the network on the noisy versions of the DRRs to reconstruct noiseless versions of the same. The original DRRs were noiseless. We then added Gaussian Noise through data augmentation methods of albumentations which we fed to the network and then finally fine-tuned the network by calculating the loss between the network output and the noiseless versions of the DRRs so as to achieve the objective of reconstructing those triple

view DRRs in noiseless format from their noisy versions. Validation SSIM for this experiment is 0.94 and loss and PSNR values are 269.91 and 36.746 dB respectively. So from the learning curves and the metrics, we can see that the network is also efficient in it's typical use case. That proves the point that the network architecture, the parameters, hyperparameters setting and dataset pre-prcoessing are all in place.

This bring us to the end of experiments. In the next subsection, we will list the accuracy metrics for all these 18 experiments.

4.11. Comparative Analysis

In this section we have put together the accuracy metrics, SSIM and PSNR, from each of the test stages, including the miscellaneous section and the subsequent evaluations in a single table 4.1 for visual comparison and for analysing the impact of each parameter on accuracy metrics.

No.	**Input**	**Output**	**Parameter Test**	**Option**	**PSNR (dB)**	**SSIM**
1	Front-view DRR	Top-view CT Volume	Baseline (Non Isotropic CT and Unrealistic DRR)	MeVisLab	16.339	0.635
2	Front-view DRR	Top-view CT Volume	Dataset Pre-processing Method	Python Script	19.56	0.65
3	Front-view DRR	Front-view CT Volume	CT-DRR Combina-tion	Frontal CT-Frontal DRR	20.704	0.725
4	Lateral-view DRR	Lateral-view CT Volume		Lateral CT-Lateral DRR	21.77	0.727
5	Top-view DRR	Top-view CT Volume		Top CT-Top DRR	22.27	0.74
6	Top+Front-view DRR	Top-view CT Volume	View Point	Top+Frontal DRR	22.227	0.741
7	Top+Front +Lateral-view DRR	Top-view CT Volume		Top+Frontal +Lateral DRR	22.497	0.742
8	Top+Front +Lateral-view DRR	Top-view CT Volume	Data Aug-mentation	True	22.561	0.742
9	Top+Front +Lateral-view DRR	Top-view CT Volume	Loss	Decomp.+ Recons.	22.55	0.744

No.	Input	Output	Parameter Test	Option	PSNR (dB)	SSIM
10	Top+Front +Lateral-view DRR	Top-view CT Volume		Decomp.+ Recons.+ Lat.	22.57	0.744
11	Top+Front +Lateral-view DRR	Top-view CT Volume	Network	2D-3D UNet	22.61	0.744
12	Top+Front +Lateral-view DRR	Top-view CT Volume	Dimension	128	22.338	0.706
13	Top+Front +Lateral-view DRR	Top-view CT Volume		512	22.767	0.802
14	Front-view DRR	Front-view CT Volume	Misc.	Ideal Combination (1 View)	21.308	0.802
15	Front +Lateral-view DRR	Front-view CT Volume		Ideal Combination (2 Views)	22.714	0.802
16	Front-view CT Volume	Front-view DRR	Subsequent Evaluations	1	38.549	0.981
17	Front-view CT Volume	Front-view CT Volume		2	35.316	0.919
18	Triple-view Noisy DRR	Triple-view Noiseless DRR		3	36.746	0.94

Table 4.1.: PSNR & SSIM Table

From the table above, we can have a clear idea about how with every parameter change, accuracy metrics changed. Now in the next subsection, we will list out the final 3 setups of the project based on our main metric.

4.12. Final Setups

One of the main aspects of the thesis was to test the impact of number of view points of DRRs. Here we have listed accordingly in table , keeping in mind the ones which yielded best SSIM values, as we are concerned more about visual structural similarity than a random loss value. We have used the final ideal combination that eventually won at the end of stage 8, which took in as input triple view DRRs. Apart from this ideal combination, we also included the couple of experiments from the Miscellaneous stage (stage 9) keeping in mind realistic point of view in terms of view points of DRRs. So these two experiments used 1 (frontal) and 2 (frontal+lateral) practically feasible view points.

So values of accuracy metric, SSIM, have been listed out below in the table 4.2 for these three setups.

Setup	**Input**	**Input Dimension**	**Output**	**Output Dimension**	**Net**	**SSIM**
Idealistic	Front+Top +Lateral-view DRR	$3\times512\times512$	Top-view CT Volume	$512\times512\times512$	2D UNet	0.802
Realistic - Basic	Front-view DRR	512×512	Front-view CT Volume	$512\times512\times512$	2D UNet	0.802
Realistic - Advanced	Front +Lateral-view DRR	$2\times512\times512$	Front-view CT Volume	$512\times512\times512$	2D UNet	0.802

Table 4.2.: Best Setups

These 3 setups produced the best results in terms of SSIM metric. On the basis of different view-points, we named them as Idealistic, Realistic - Basic and Realistic - Advanced.

Since the Idealistic setup is not feasible in reality and since the Realistic - Advanced generated visually more appropriate CT slices than the Realistic - Basic one, so the final setup of choice would be the Realistic - Advanced one.

5
Conclusion and Future Work

This chapter concludes the report explaining our final accomplishment and also outlining the scopes of future work in this topic which can lead to a probable improvement of the expected results and drive us closer to the goal, set for this thesis work.

5.1. Conclusion

We have arrived at the last section of the report. Right from introducing the basic concepts of medical imaging in Introduction to providing the technical context in the chapter of Background and from giving an overview of the conducted experiments along with their respective setups in the chapter of Experiments to visualising network outputs and analysing implementations in Results and Discussion, the reader will have a great time assimilating the complete workflow of the project. Following this chapter, there is Appendix (A), which explains the journey of implementation through different phases. These stages of failed implementations formed the cornerstone for the final successful execution of the project.

Value of validation SSIM for first implementation was 0.532 and the same for the final phase stands at 0.802. This track explaining the step by step increase in accuracy in every stage must have provided the reader an interesting outlook on what all, that needs to be done, to achieve something similar to this, with the existing setup. With an accuracy of 80.2%, it is already promising enough and the results are visually quite satisfactory. If further research is conducted on this topic and a break-through can be achieved in the same, then there might come a time when it will be possible to extract complete volume information of internal anatomy of human body from a very simple, cheap and easily available imaging modality, X-ray. This will be able to save patients the expenses for obtaining expensive CT scan and also help them avoid the need to be subjected to excessive radiation, associated with medical imaging with CT modality. Hence we need to have an insight into possible future work that can be done on this project so as to maximise the volume information that can be extracted from a collapsed 2D projection image. Since it brings us to the end of the report, we will have a discussion in the section of future work in order to provide a basic guideline to the reader on how to improve the accuracy for achievement of more reliability during evaluation.
There are multiple sources which have implemented generation of DRR or X-ray from existing CT scan data either through normal programming or with the help of a tool, like MeVisLab, which has been used for this project as well, or through deep learning approaches. But there is hardly any rightful implementation for achieving the exact opposite - generation of CT scan from DRR. We have already seen in subsequent evaluations that the training process of generation of DRR from CT scan volume or the CT scan volume from itself or noiseless triple-view DRRs from their noisy versions take much lesser number of epochs and crosses 90% very easily in every instance. Hence the reason is clear. It is much easier to interpolate from higher dimension of information to a lower or an equal one but

the opposite is equally tough. In one of the implementations, described in related work section of 2 chapter, we have seen that it is possible to train the network on known set of input-label pairs and then generate the CT volume from DRR or X-ray for patient whose data has already been a part of the training process. In other words, you need to obtain the CT scan volume and DRR for the patient and train the network with that data and then finally input the obtained DRR into the network to extract CT volume information, as a result of which they were able to attain a much higher accuracy than us. Although the results look pretty good, but it makes the whole purpose meaningless if you have to extract both CT volume information and DRR/X-ray beforehand for a patient so as to obtain the same CT volume information from the generated DRR/X-ray. Not only that, it also goes against the basic principle of machine learning / deep learning which requires evaluation to be carried out on unknown data or data which is not a part of the training dataset. Otherwise, evaluation on known data, whose information has been used to fine-tune the network, would always give great results. So this thesis entails the first ever R&D conducted thoroughly in this regard. So let's have a look into the probable future work, that can bring in ground breaking results in this topic and set a new milestone in the same.

5.2. Future Work

There are some key areas which have been identified as future area of work for further improvement, following the foundation laid by this project.

- We saw that 2D-3D UNet could not improve the results over the baseline 2D UNet. One of the major reasons was that the 2D-3D UNet was not as much wde as the 2D UNet. This was because the 2D-3D UNet stresses the system GPU way more than the 2D UNet, as a result of which the network cannot be set to eual width as that of 2D UNet. For our case, the network width for 2D-3D UNet was set at maximum of what the system can handle and beyond that, system would throw a memory error. Hence as a possible future work, while using 2D-3D UNet, a much wider network can be used while training. That can probably improve the results quite a lot.

- There has already been an implementation on generation of a 3D object from images taken for the object from multiple angles by making use of a state-of-the-art network, 3D-R2N2 [18]. When for this case, the network tries to generate the object from it's own images taken from different angles, for ours the network has to generate the non projected CT volume from weighted projection of the same. Generation of a 3D object from it's image and the same from it's weighted projection are pretty different tasks in terms of appearance of the input and label. Still we can use that approach for our case to find out if it improves the accuracy or not.

- There is another implementation using 3D-GAN [104] in which it tries to generate the same volume from it's single image through learning of a probabilistic latent space of object shapes. We can use the same network and the setup and transfer it to our project to again look for scopes of improvement.

- There are multiple approaches clubbed together for Image-based 3D Object Reconstruction in the paper referred in [42]. Each of these approaches can be tried out for our task. Although again, each of these approaches work on reconstruction of the object from it's own image, as described in second and third points, still it is worthy analysing each of these approaches for our task. One of it might stand out and cause a considerable improvement in results.

- Also we can try out a different dataset which has more cleaner examples and test the entire similar setup of this project for training and evaluation.
- On a futuristic perspective, we can try to obtain other imaging modalities, like CT scan, similarly from X-rays/DRRs.

This lists out the possible future work in this area and subsequently brings us to the end of the report. I wish it was a pleasant experience for the readers, while going through the report.

List of Abbreviations

List of Softwares

Name	Version	URL	Comment
PyCharm	Professional Edition	https://www.jetbrains.com/pycharm/	Used for writing the complete Deep Learning algorithm and also for dataset pre-processing
MeVisLab	3.3 (VS2017-64 debug)	https://www.mevislab.de/	Used for dataset pre-processing
Fiji ImageJ	1.6	https://imagej.net/Fiji	Used for CT volume viewing

Table 5.1.: Software Table

Bibliography

[1] *IOP Conf. Series: Journal of Physics: Conf.*, 1004:012027, 2018.

[2] Wil Aalst, Vladimir Rubin, H. Verbeek, B. Dongen, E. Kindler, and C. Günther. Process mining: A two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9:87–111, 11 2010.

[3] Shadi Albarqouni, Javad Fotouhi, and Nassir Navab. X-ray in-depth decomposition: Revealing the latent structures. *Lecture Notes in Computer Science*, page 444–452, 2017.

[4] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[5] Nura Aljaafari. *Ichthyoplankton Classification Tool using Generative Adversarial Networks and Transfer Learning*. PhD thesis, 02 2018.

[6] T.R.V. Anandharajan, G. Abhishek Hariharan, K.K. Vignajeth, R. Jijendiran, and Kushmita. Weather monitoring using artificial intelligence. In *2016 2nd International Conference on Computational Intelligence and Networks (CINE)*, pages 106–111, 2016.

[7] Caner Imaging Archive. LIDC-IDRI Dataset. `https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI`.

[8] Caner Imaging Archive. LIDC-IDRI Dataset. `https://www.cancerimagingarchive.net/collections/`.

[9] Kaito Ariu, Narae Ryu, Se-Young Yun, and Alexandre Proutière. Regret in online recommendation systems, 2020.

[10] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[11] Ivo Matteo Baltruschat. Deep learning for advanced medical applications. 12th Oct, 2016.

[12] David GT Barrett, Ari S Morcos, and Jakob H Macke. Analyzing biological and artificial neural networks: challenges with opportunities for synergy? *Current Opinion in Neurobiology*, 55:55–64, 2019. Machine Learning, Big Data, and Neuroscience.

[13] Aman Bhalla, Munipalle Sai Nikhila, and Pradeep Singh. Simulation of self-driving car using deep learning. In *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, pages 519–525, 2020.

[14] M. Emre Celebi and Kemal Aydin. *Unsupervised Learning Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[15] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey, 2019.

[16] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. Convolutional neural network (cnn) for image detection and recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pages 278–282, 2018.

[17] Daegyu Choe, Eunjeong Choi, and Dongkeun Kim. The real-time mobile application for classifying of endangered parrot species using the cnn models based on transfer learning. *Mobile Information Systems*, 2020:1–13, 03 2020.

[18] Chris Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. volume 9912, pages 628–644, 10 2016.

[19] Vincent Christlein, Lukas Spranger, Mathias Seuret, Anguelos Nicolaou, Pavel Král, and Andreas Maier. Deep generalized max pooling, 2019.

[20] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. *Supervised Learning*, pages 21–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[21] Zhenwei Dai and Reinhard Heckel. Channel normalization in convolutional neural network avoids vanishing gradients, 2019.

[22] Ashraf Darwish, Aboul Ella Hassanien, and Swagatam Das. A survey of swarm and evolutionary computing approaches for deep learning. *Artificial Intelligence Review*, 53, 03 2020.

[23] Rodrigo De Oliveira, Ramon Cristian Fernandes Araújo, Fabrício Barros, Adriano Segundo, Ronaldo Zampolo, Wellington Fonseca, Victor Dmitriev, and Fernando Brasil. A system based on artificial neural networks for automatic classification of hydro-generator stator windings partial discharges. *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, 16:628–645, 09 2017.

[24] Ayon Dey. Machine learning algorithms: A review. *International Journal of Computer Science and Information Technologies*, 7, 2016.

[25] Bio Differences. X-ray and CT scan comparison 4. `https://biodifferences.com/difference-between-x-ray-and-ct-scan.html`.

[26] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. 08 2016.

[27] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation, 2016.

[28] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. cite arxiv:1603.07285.

[29] Ashkan Eliasy and Justynsa Przychodzen. The role of ai in capital structure to enhance corporate funding strategies. *Array*, 6:100017, 07 2020.

[30] Lance Eliot and Michael Eliot. *Autonomous Vehicle Driverless Self-Driving Cars and Artificial Intelligence: Practical Advances in AI and Machine Learning*. LBE Press Publishing, 1st edition, 2017.

[31] Fernando A. Fardo, Victor H. Conforto, Francisco C. de Oliveira, and Paulo S. Rodrigues. A formal evaluation of psnr as quality measurement parameter for image segmentation algorithms, 2016.

[32] Jianli Feng and Shengnan Lu. Performance analysis of various activation functions in artificial neural networks. *Journal of Physics: Conference Series*, 1237:022030, 06 2019.

[33] Andreas Fouras, Marcus Kitchen, Stephen Dubsky, R. Lewis, Stuart Hooper, and Kerry Hourigan. The past, present, and future of x-ray technology for in vivo imaging of function and form. *Journal of Applied Physics*, 105:102009 – 102009, 06 2009.

[34] Reagan Galvez, Argel Bandala, Elmer Dadios, Ryan Vicerra, and Jose Martin Maningo. Object detection using convolutional neural networks. pages 2023–2027, 10 2018.

[35] Debashis Ganguly, Srabonti Chakraborty, Maricel Balitanas, and Tai-hoon Kim. *Medical Imaging: A Review*, volume 78, pages 504–516. 09 2010.

[36] Andrei Gavrilov, Alex Jordache, Maya Vasdani, and Jack Deng. Preventing model overfitting and underfitting in convolutional neural networks. *International Journal of Software Science and Computational Intelligence*, 10:19–28, 10 2018.

[37] Spiros Georgakopoulos and V. Plagianakos. A novel adaptive learning rate algorithm for convolutional neural network training. pages 327–336, 08 2017.

[38] Spiros V. Georgakopoulos and Vassilis P. Plagianakos. Efficient learning rate adaptation for convolutional neural network training. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[40] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[41] Venkat Gudivada, Amy Apon, and Junhua Ding. Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. *International Journal on Advances in Software*, 10:1–20, 07 2017.

[42] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1578–1604, May 2021.

[43] CASPER HANSEN. Setting the learning rate of your neural network. `https://mlfromscratch.com/neural-networks-explained/#/`, Aug 5, 2019.

[44] William Grant Hatcher and Wei Yu. A survey of deep learning: Platforms, applications and emerging research trends. *IEEE Access*, 6:24411–24432, 2018.

[45] Juncai He, Xiaodong Jia, Jinchao Xu, Lian Zhang, and Liang Zhao. Make ℓ_1 regularization effective in training sparse cnn, 2020.

[46] Takahiko Henmi, Esmeraldo Ronnie Rey Zara, Yoshihiro Hirohashi, and Tsuyoshi Kato. Adaptive signal variances: Cnn initialization through modern architectures, 2020.

[47] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[48] Vinita Silaparasetty-Data Driven Investor. How to Handle Overfitting and Underfitting in Machine Learning. `https://medium.datadriveninvestor.com/how-to-handle-overfitting-and-underfitting-470a1f7389fes`, Nov 27, 2019.

[49] A.K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, 1996.

[50] Katarzyna Janocha and Wojciech Czarnecki. On loss functions for deep neural networks in classification. *ArXiv*, abs/1702.05659, 2017.

[51] JEREMY JORDAN. Setting the learning rate of your neural network. `https://www.jeremyjordan.me/nn-learning-rate/`, Mar 1, 2018.

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

[53] Paul Kirkpatrick and A. V. Baez. Formation of optical images by x-rays. *J. Opt. Soc. Am.*, 38(9):766–774, Sep 1948.

[54] Wilco Koppert, Martijn Dietze, Sandra van der Velden, Leo Steenbergen, and Hugo Jong. A comparative study of nai(tl), cebr3, and czt for use in a real-time simultaneous nuclear and fluoroscopic dual-layer detector. *Physics in Medicine and Biology*, 64, 06 2019.

[55] Ping-Huan Kuo, Ssu-Ting Lin, and Jun Hu. Dnae-gan: Noise-free acoustic signal generator by integrating autoencoder and generative adversarial network. *International Journal of Distributed Sensor Networks*, 16:155014772092352, 05 2020.

[56] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[57] Hongsheng Li, Rui Zhao, and Xiaogang Wang. Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification, 2014.

[58] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 661–670, New York, NY, USA, 2014. Association for Computing Machinery.

[59] Ying Siu Liang. *End-user Robot Programming in Cobotic Environments*. PhD thesis, 06 2019.

[60] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014.

[61] Laurent Louis, Patrick Baud, and Teng-fong Wong. Characterization of pore-space heterogeneity in sandstone by x-ray computed tomography. *Geological Society, London, Special Publications*, 284:127–146, 01 2007.

[62] Alexander Selvikvåg Lundervold and Arvid Lundervold. An overview of deep learning in medical imaging focusing on mri. *Zeitschrift für Medizinische Physik*, 29(2):102–127, 2019. Special Issue: Deep Learning in Medical Physics.

[63] Sainik Kumar Mahata, Dipankar Das, and Sivaji Bandyopadhyay. Mtil2017: Machine translation using recurrent neural network on statistical machine translation. *Journal of Intelligent Systems*, 28, 05 2018.

[64] Fraunhofer MEVIS. MeVisLab. `https://mevislabdownloads.mevis.de/docs/current/FMEwork/ReleaseMeVis/Documentation/Publish/Overviews/DICOMInMeVisLab.html`.

[65] Fraunhofer MEVIS. MeVisLab 1. `https://mevislabdownloads.mevis.de/docs/current/MeVisLab/Standard/Documentation/Publish/ModuleReference/DicomRescale.html`.

[66] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey, 2020.

[67] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition, 1997.

[68] Mahmoud Moustafa. Ct paper. 06 2013.

[69] Brian Mwandau and Matunda Nyanchama. *Investigating Keystroke Dynamics as a Two-Factor Biometric Security*. PhD thesis, 06 2018.

[70] Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks, 2019.

[71] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 78, New York, NY, USA, 2004. Association for Computing Machinery.

[72] Jim Nilsson and Tomas Akenine-Möller. Understanding ssim, 2020.

[73] Abhinav Sagar KD Nuggets. 5 Techniques to Prevent Overfitting in Neural Networks. `https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html`, Dec, 2019.

[74] Daniel Omeiza. A step towards exposing bias in trained deep convolutional neural network models, 2019.

[75] Pariwat Ongsulee. Artificial intelligence, machine learning and deep learning. In *2017 15th International Conference on ICT and Knowledge Engineering (ICT KE)*, pages 1–6, 2017.

[76] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[77] Goodman PC. The new light: discovery and introduction of the x-ray. June 12,2017.

[78] Robert Pelberg. *Basic Principles in Computed Tomography (CT)*, pages 19–58. Springer London, London, 2015.

[79] Cyber Physics. CT Scan Imaging Diagram. https://www.cyberphysics.co.uk/topics/medical/CTScanner.htm, 2018.

[80] Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[81] H. Qi. Derivation of backpropagation in convolutional neural network (cnn). 2016.

[82] Pradheepan Raghavan and Neamat Gayar. Fraud detection using machine learning and deep learning. pages 334–339, 12 2019.

[83] Kumar Ravi, Srirangaraj Setlur, Ravi Vadlamani, and Venu Govindaraju. Article citation sentiment analysis using deep learning. pages 78–85, 07 2018.

[84] Get Revising. X-ray and CT scan comparison 3. https://getrevising.co.uk/grids/x_rays_ct_scans_and_ultrasound, 21st May, 2013.

[85] Giorgio Roffo. Ranking to learn and learning to rank: On the role of ranking in pattern recognition applications. 06 2017.

[86] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[87] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[88] Artem Oppermann Towards Data Science. Regularization in Deep Learning — L1, L2, and Dropout. https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036, Feb 19, 2020.

[89] Nilesh Vijayrania Towards Data Science. Different Normalization Layers in Deep Learning. https://towardsdatascience.com/different-normalization-layers-in-deep-learning-1a7214ff71d6, Dec 10, 2020.

[90] Liyue Shen, Wei Zhao, and Lei Xing. Patient-specific reconstruction of volumetric computed tomography images from a single projection view via deep learning. *Nature Biomedical Engineering*, 3, 11 2019.

[91] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, Mar 2020.

[92] Sangmun Shin, Thanh-Tra Hoang, Tuan-Ho Le, and Moo-Yeon Lee. A new robust design method using neural network. *Journal of Nanoelectronics and Optoelectronics*, 11:68–78, 02 2016.

[93] Connor Shorten and Taghi Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 07 2019.

[94] Shoaib Siddiqui, Imran Malik, Faisal Shafait, Ajmal Mian, Mark Shortis, and Euan Harvey. Automatic fish species classification in underwater videos: Exploiting pretrained deep neural network models to compensate for limited labelled data. *ICES Journal of Marine Science*, 75, 05 2017.

[95] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[96] Baikai Sui, Tao Jiang, and Zhen Zhang. Ecgan: An improved conditional generative adversarial network with edge detection to augment limited training data for the classification of remote sensing images with high spatial resolution. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, PP:1–1, 10 2020.

[97] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[98] Lisa Torrey and Jude Shavlik. Transfer learning.

[99] Twan van Laarhoven. L2 regularization versus batch and weight normalization, 2017.

[100] Hans-Dieter Wehle. Machine learning, deep learning, and ai: What's the difference? 07 2017.

[101] Wikipedia. PSNR. `https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio`.

[102] Wikipedia. SSIM. `https://en.wikipedia.org/wiki/Structural_similarity`.

[103] Wikipedia. PyCharm IDE. `https://en.wikipedia.org/wiki/PyCharm`, 16 May 2021.

[104] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[105] Yuxin Wu and Kaiming He. Group normalization, 2018.

[106] Minglong Xue, Jian Li, and Qingli Luo. Toward optimal learning rate schedule in scene classification network. *IEEE Geoscience and Remote Sensing Letters*, pages 1–5, 2020.

[107] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights Imaging*, 06 2018.

[108] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. 02 2017.

[109] Faisal Zaman. *Automated Liver Segmentation from MR-Images Using Neural Networks*. PhD thesis, 11 2019.

[110] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review, 2018. cite arxiv:1807.05511.

[111] XueFei Zhou. Understanding the convolutional neural networks with gradient descent and backpropagation. *Journal of Physics: Conference Series*, 1004:012028, 04 2018.

[112] Òscar Lorente, Ian Riera, and Aditya Rana. Image classification with classic and deep learning techniques, 2021.

A

Phases of Failed Implementation

In this chapter we will take you through the list of unsuccessful phases of implementation. In each of these phases, you will get acquainted with the background motivation that drove the research for that phase, how the work was carried out post the research, how the final goal kept getting evolved, the findings of the respective phase and finally the associated drawbacks, which allowed areas of improvement in their consequent phases.

A.1. Phase 1

In the first phase, we set up a goal, very different from the work associated with the thesis topic. It was similar to the work, done by the authors in [3]. The complete course of this thesis, right from setting up of initial objective to achievement of the final goal and the research and the work, that followed along, is premised upon/inspired from the work done in paper [3]. In other words, it is the reference paper, that we used for our thesis. A brief idea of the initial objective, dataset acquisition process, network architecture and drawbacks associated with the method will help to form a clearer idea about the initial task description.

Initial Objective : As discussed in the chapter 1 (Introduction) of the report, while generating an X-ray for a selected section of a human body, the whole 3D anatomical structures of the body gets collapsed in a single 2D X-ray image and anatomy of concern, like soft tissues might get obscured by overlaying structures of bones with higher absorption power. Hence a lot of detail is lost in the process and needs skilled clinicians to focus on the anatomy of interest. But the accuracy still remains questionable with this process of human intervention and it might be interesting to explore the improvement based on a computer-aided assistance.

So in the first phase of implementation, the goal was to split a 2D chest X-ray/DRR, obtained from projection of the complete CT volume of a patient, into a set of 3 independent non-overlapping 2D X-rays/DRRs, obtained by projecting 3 clipped and subsequently zero-padded (for retaining the original CT volume's dimension) sub-volumes (ribcage, vasculature and spine) of the CT scan for the same patient. As mentioned above, this task has already been implemented in a research paper 'X-ray In-Depth Decomposition: Revealing The Latent Structures'[3] by Shadi Albarqouni, Javad Fotouhi, and Nassir Navab on 6 different clinical datasets. Our task was to achieve similar results with the LIDC-IDRI[7] dataset.

Dataset : This section is broadly divided into 3 sub-sections - raw data, data pre-processing and processed dataset.

Raw Data : LIDC-IDRI was used as the raw data. It consists of 1012 entries of patients. Each of these entries consists of CT volume of $z \times 512 \times 512$ pixels with non-normalised z dimension. It means that each of these volumes has unequal number of top-view slices (z dimension), but each of these slices for each of the volumes has a dimension of $512(x) \times 512(y)$ pixels. Every single entry forms a single data point (input-label pair) in the final dataset, obtained through appropriate pre-processing of this raw dataset.

Let's have a look at the complete 3D CT volume, for patient 2 from the dataset, rendered by VTK Plotter in the figure A.1, from frontal, top and lateral view point along.

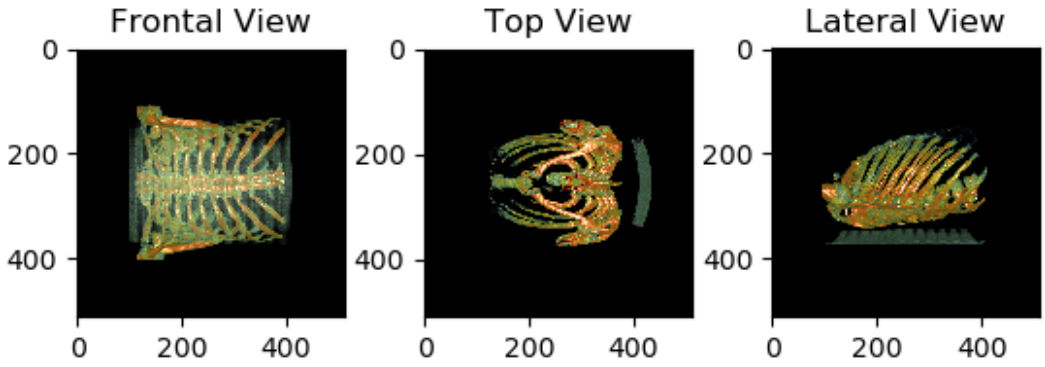


Figure A.1.: VTK plotter CT volume. Complete 3D CT volume plotted by VTK plotter - frontal (left), top (middle) and lateral (right) views - patient 2

Data Pre-processing : The target of pre-processing was to obtain 4 frontal DRRs - 3 (labels) for ribcage, vasculature and spine and also another one (input) for the whole volume for each of these non-normalised (z dimension) CT volume entries in the raw data.

Let's have a look at the 75th, 125th and 175th slices for patient 2 from the dataset.

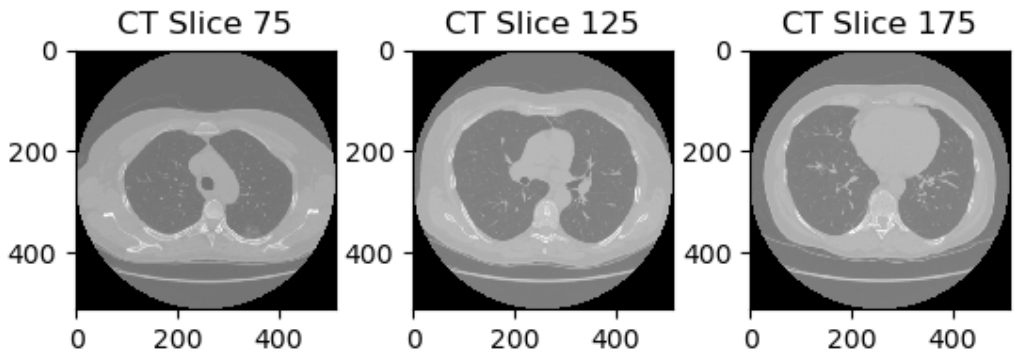


Figure A.2.: Patient 2 complete volume: slices 75 (left), 125 (middle) and 175 (right)

This entire CT volume was loaded using a Python script a NumPy 3D NumPy array. Since the raw CT volume for this patient consists of 261 slices, so the dimension of array is given by $261 \times 512 \times 512$

pixels. Then this NumPy volume was clipped into three sub-volumes along the x axis with indices $[0 : 154]$ for ribcage, $[155 : 326]$ for vasculature and $[327 : 512]$ for spine. Each of these clipped sub-volumes was zero-padded so as to retain the original dimension of $261 \times 512 \times 512$ pixels.

The slices 75, 125 and 175 for the clipped and then zero-padded sub-volume of ribcage are shown in figure A.3.

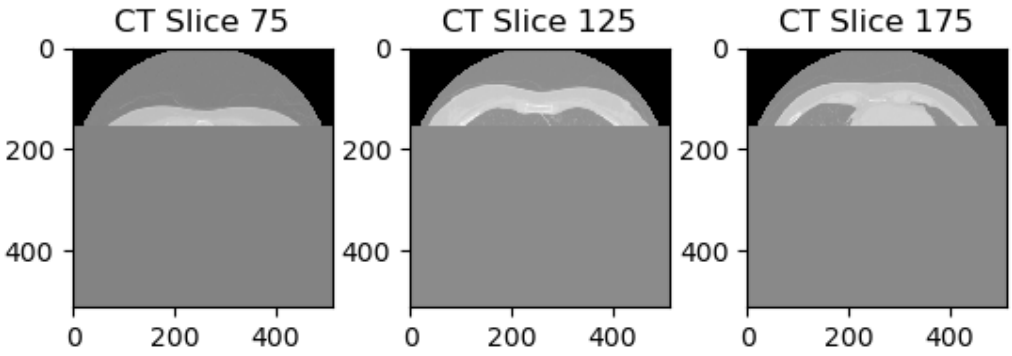


Figure A.3.: Patient 2 sub-volume ribcage: slices 75 (left), 125 (middle) and 175 (right)

The same slices for the clipped sub-volume of vasculature are available in figure A.4

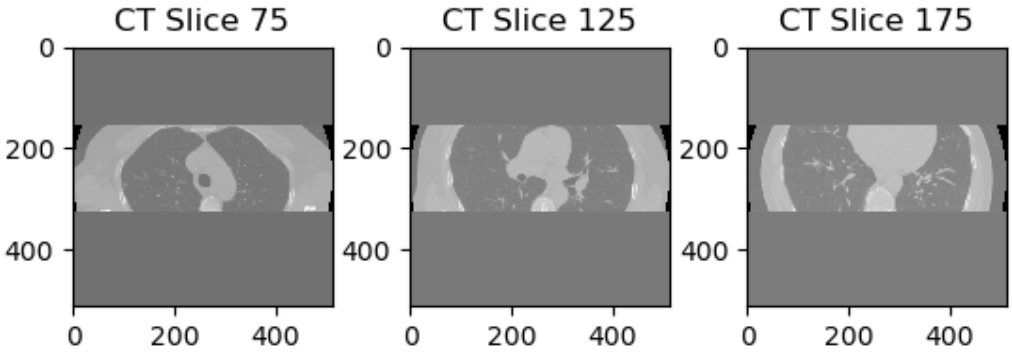


Figure A.4.: Patient 2 sub-volume vasculature: slices 75 (left), 125 (middle) and 175 (right)

Finally, the figure A.5 shows the clipped slices for the spine sub-volume.

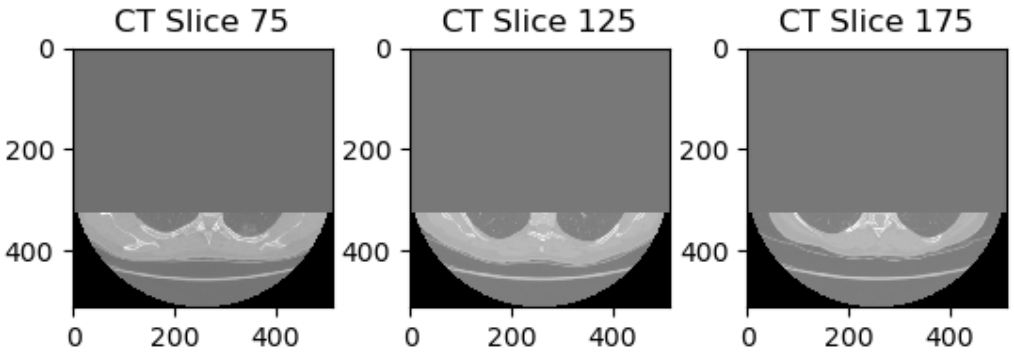


Figure A.5.: Patient 2 sub-volume spine: slices 75 (left), 125 (middle) and 175 (right)

After this, we generate 2D frontal DRRs of dimensions 256×256 pixels using Python script. Figure A.6 shows the DRRs generated for the whole and clipped sub-volumes for patient 1 from the dataset.

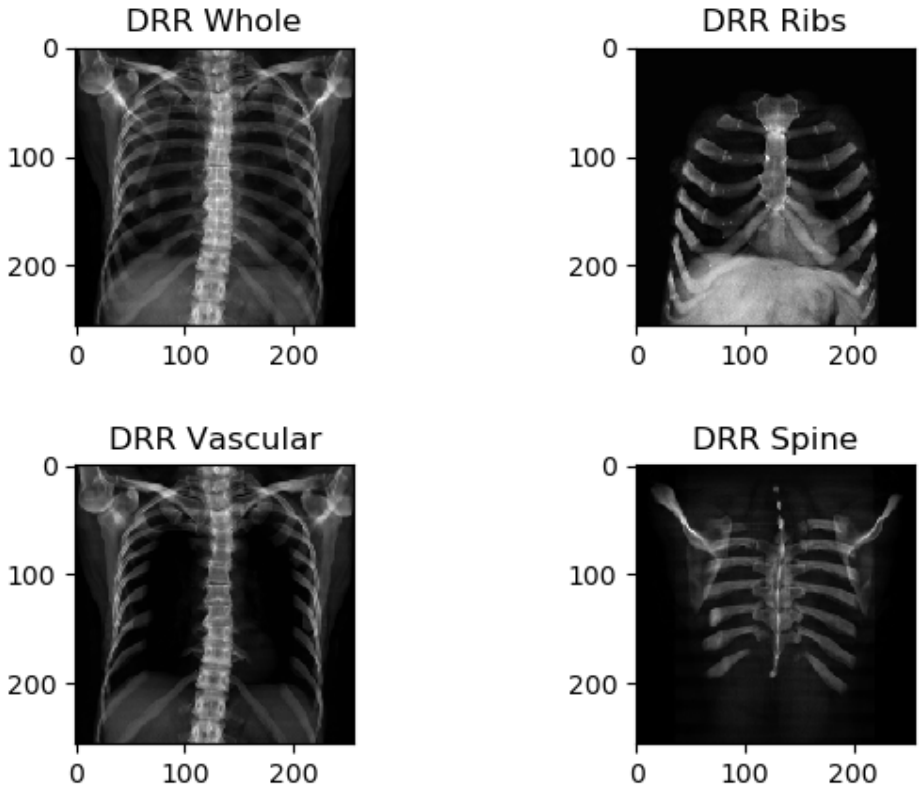


Figure A.6.: DRRs for whole and clipped volumes Python: whole (top left), ribs (top right), vasculature (bottom left), spine (bottom right)

But these DRRs hardly look like real world X-rays. In fact, they look more like the bone mask structure extracted from the complete CT volume for patient 2 of the dataset and when juxtaposed with the

bone mask structure extracted by the authors, mentioned in phase 4 of Appendix, they look strikingly similar. The comparison is shown in the figure A.7.

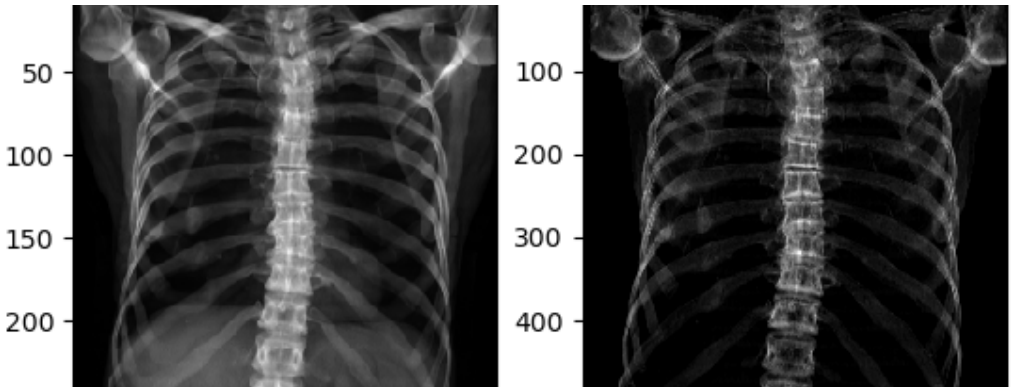


Figure A.7.: DRR masks self-generated (left) and generated by author (right) for patient 2

This can be explained by the fact that these DRRs were generated by adding up pixel values along the direction of projection (z). The major flaw with the script is that instead of merely adding up the pixel values along the projection path, it should be adding up weighted pixel values along the DRR ray cast, depending upon the distance of the pixel from the X-ray/DRR source, and also on the quantity of absorption by each constituent or in other words the attenuation factor of the ray along it's path. So in the process, the method of extracting bone mask structure from CT volume was found out but the way to generate DRR was yet to be explored.

In the reference paper, the authors used a software name ImFusion for DRR generation. Hence we dropped the idea of DRR generation using Python script and instead decided to use either an online application or an offline desktop tool for the purpose. We wanted to use the same tool as that of the authors, but it was not possible because it needed an activation license to be purchased. So with that option being out of scope, we researched and came across several other already available desktop applications, apart from ImFusion, which are specialised for tackling DICOM image format. Those are Plastimatch, MeVisLab, 3D Slicer and ImageJ. But after trying out each and every application, MeVisLab was chosen as the suitable one, and the reasons were very clear. The other applications are either developed solely for the purpose of viewing by physicians and other concerned authorities or were too simple to support advanced operations required for our task or did not render the required results in expected manner or needed purchase of licenses for implementing those advanced operations or had incomplete modules. In either way, none of the other applications could single-handedly support the advanced modules to complete the workflow of DRR generation. ImageJ was used to view the slices resampled by MeVisLab and it was discovered that that the slices are produced by resampling module of MeVisLab are in reverse order. Besides that, no other DICOM tool was used for any other purpose. Apart from the modules required for generating DRR MeVisLab have a complete list of already working modules for several other kinds of tasks and also offers prompt and quality community support. It has one drawback of executing operations for a single patient at a time and not allowing automation for multiple patients. Hence it increases the total time and effort for pre-processing a large dataset to a huge extent but at least it can get the job done, unlike the other tools. Parallel beam of simulated X-ray was used for the purpose. The setup for generating DRR for any CT volume in MeVisLab is shown in the figure A.8.

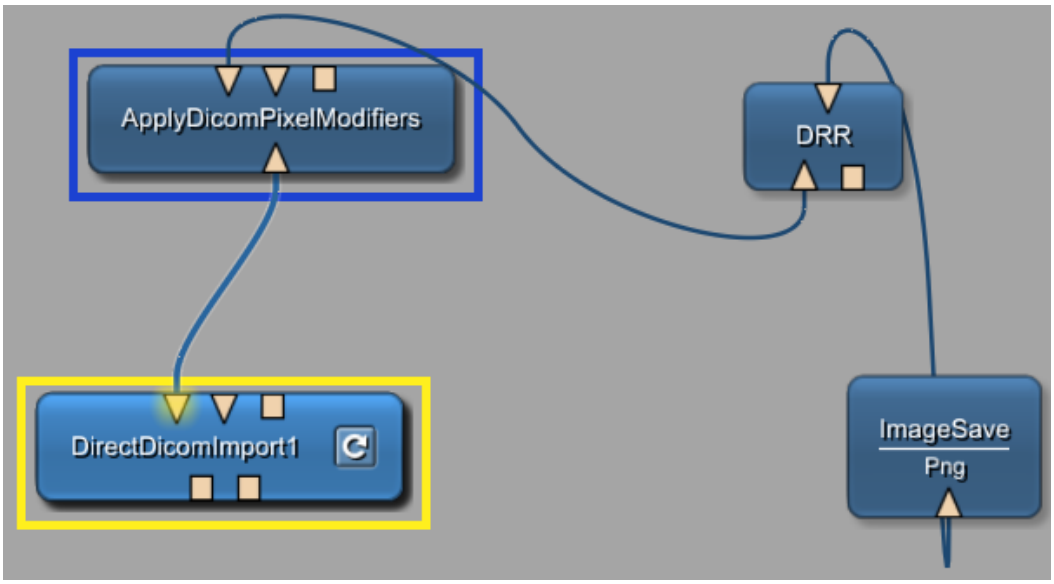


Figure A.8.: MeVisLab setup for DRR generation

It consists of a cascade of modules put together to generate the DRR. The functionalities of each of these modules are described below.

Direct Dicom Import : Figure A.9 shows the first module (Direct Dicom Import) involved in the workflow.

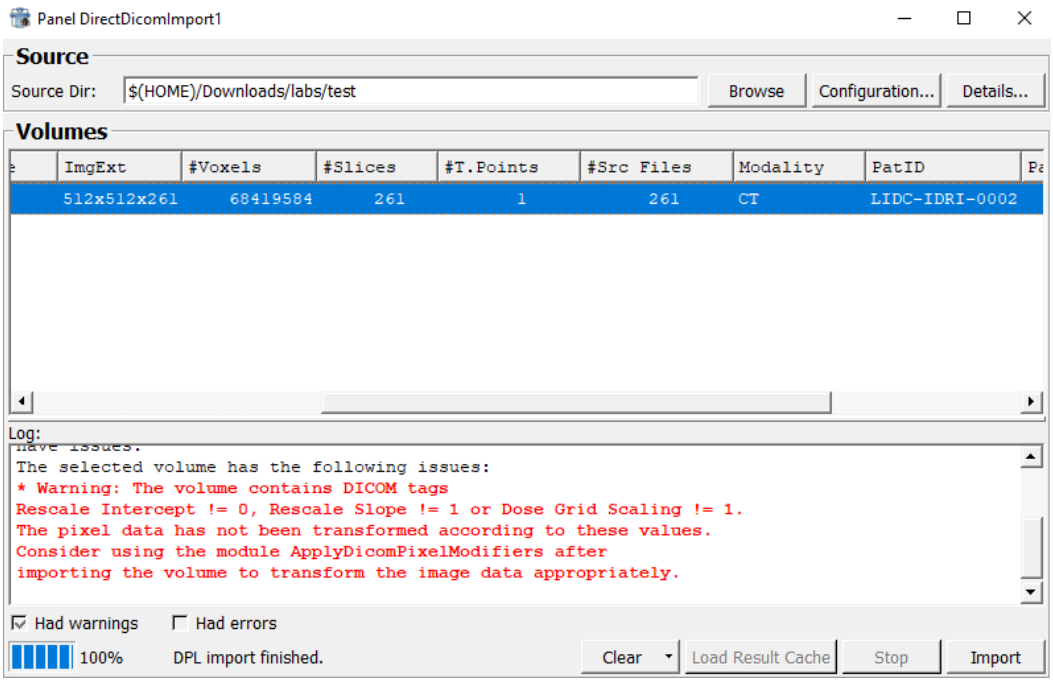


Figure A.9.: Direct dicom import module MeVisLab [image source: [64]]

It is used to import the whole volume of CT slices for each patient. It provides multiple information through it's different fields as can be seen in the image A.9 above. Imaging modality, mentioned as CT in the "Modality" field, has been imported for patient 2 of the dataset, as is shown in the "PatID" field. The total volume is of $512 \times 512 \times 261$ pixels as mentioned in "ImgExt" field. It means the CT volume consists of 261 top view slices, also mentioned in "Src Files", each of 512×512 pixels. More details and additional configuration are available under the "Details" and "Configuration" tabs, respectively, but further discussion on those two tabs are out of scope for this thesis. After successfully importing the CT volume information for patient 2, a warning is displayed that "Rescale Intercept" and "Rescale Slope" or "Dose Grid Scaling" are not 0 and 1, respectively, for which the next module, "Apply Dicom Pixel Modifiers", is suggested.

Apply Dicom Pixel Modifiers : Figure A.10 depicts this pixel modifier module.

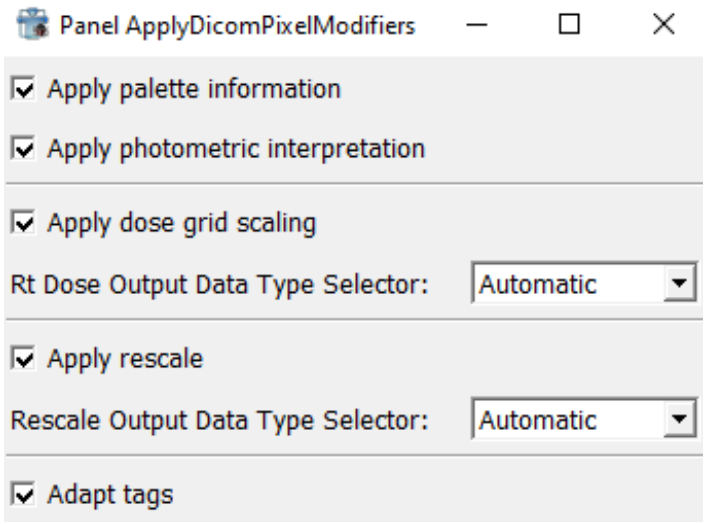


Figure A.10.: Apply dicom pixel modifiers module MeVisLab

This module is used to convert Intensity Values (IV) for each of the 261 CT slices for patient 2 to Hounsfield Units (HU)[65], by normalising intercept and slope to 0 and 1, respectively, in equation A.1. It provides standardisation or normalisation based on the image tags for pixel values of CT volumes that may have come from different vendors who may have used different methods to acquire them. Hence it is extremely essential to convert them to HU to have normalised CT slices across the whole volume and across different patients.

$$HU = IV * slope + intercept \tag{A.1}$$

"Apply Dose Grid Scaling" and "Apply Rescale" fields in the module achieve rescaling of slope and intercepts to 1 and 0, respectively.

DRR : In the figure A.11, we can see details of the DRR module.

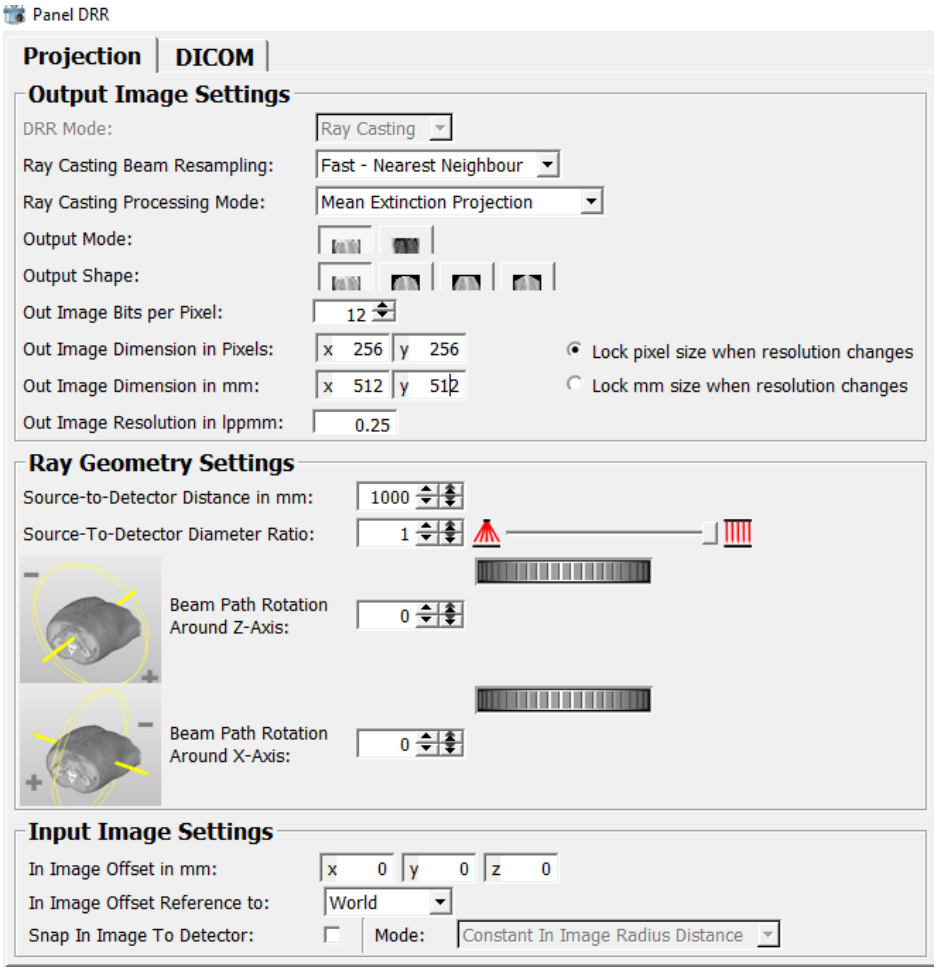


Figure A.11.: DRR module MeVisLab

The fields which are of concern are “Output Mode”, “Output Shape” and “Out Image Dimensios in Pixels”. “Output Mode” and “Output shape” have been selected as the ones which are highlighted in the image due to them rendering the final DRR with the look and feel as expected. “Out Image Dimension in Pixels” have been locked as the deciding factor in case of change of resolution and has been set to 256×256 (x,y) pixels because we wanted to generate a 2D DRR of 256×256 dimension, as mentioned in the beginning of step 3. Additionally, “Source-To-Detector Diameter Ratio” has been set to 1, which would generate DRR using parallel beam, in contrast to using 0 from 2nd phase onward, which generates the same by making use of conical beam. The difference between these two beams are discussed more in details in the 2nd phase. Apart from the combination of “Fast - Nearest Neighbour” and “Mean Extinction Projection”, “Exact - Siddon” and “LUT Based Projection” have also been tried for “Ray Casting Beam Resampling” and “Ray Casting Processing Mode” fields, respectively, but the combination displayed in the figure A.11 yielded the best results in terms of look and feel of what is expected.

Image Save : This module is simply used to save the generated DRR in different formats as shown in figure A.12. Images were saved in both PNG format for the sake of viewing and also in DICOM

format for data loading in Python script. The DICOMs preserve the header information, in case those information is later required to be appended to the image for retaining the patient information.

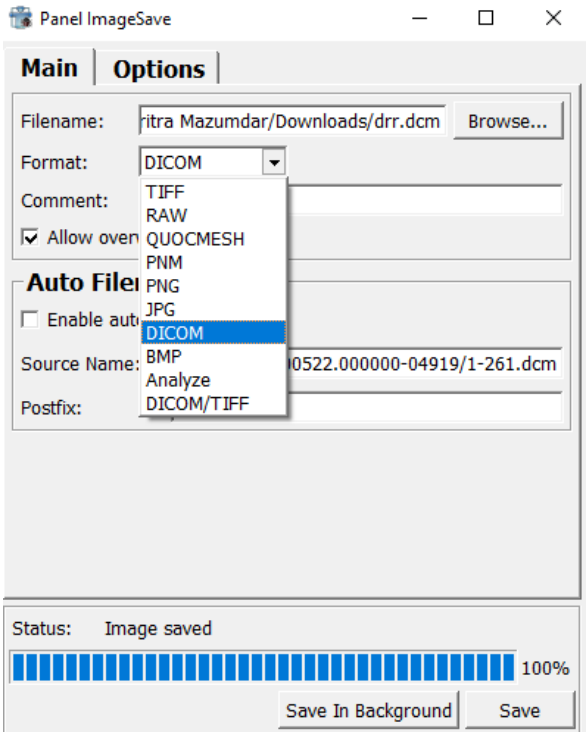


Figure A.12.: Image save module MeVisLab

The connectors in between the modules basically carry the output from each module to the input of the next module. Figure A.13 shows the output inspector when the outgoing connection of the "Direct Dicom Import" Module is selected for patient 2 of the dataset.

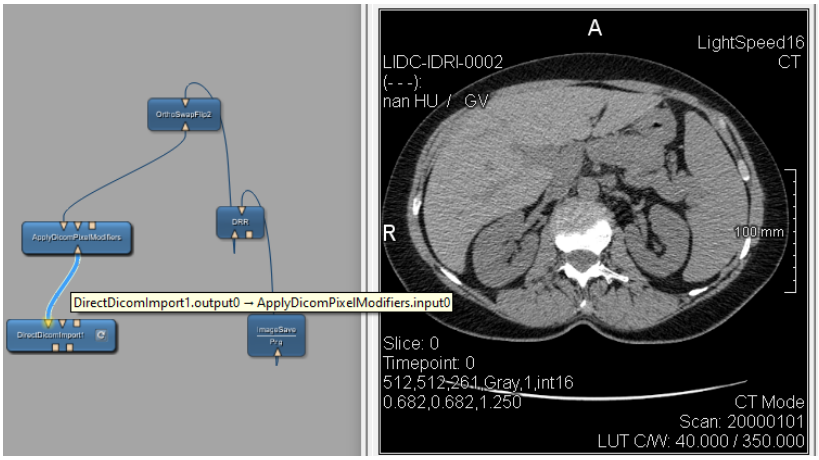


Figure A.13.: Connector output MeVisLab

Processed Dataset : The generated DRRs for the whole CT volume (which form the input to the network) and for the sub-volumes of ribcage, vasculature and spine (which form the labels or the expected output from the network) for first three Patient (P)s of the dataset are available in figures A.14, A.15, A.16 and A.17, respectively.

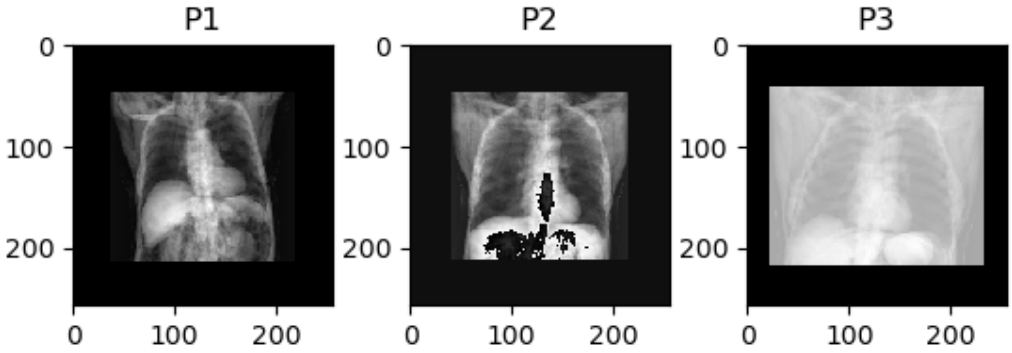


Figure A.14.: Input authors

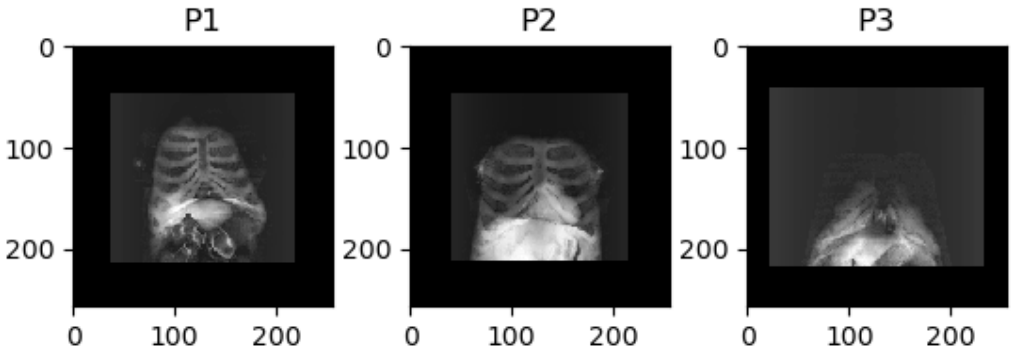


Figure A.15.: Labels ribs authors

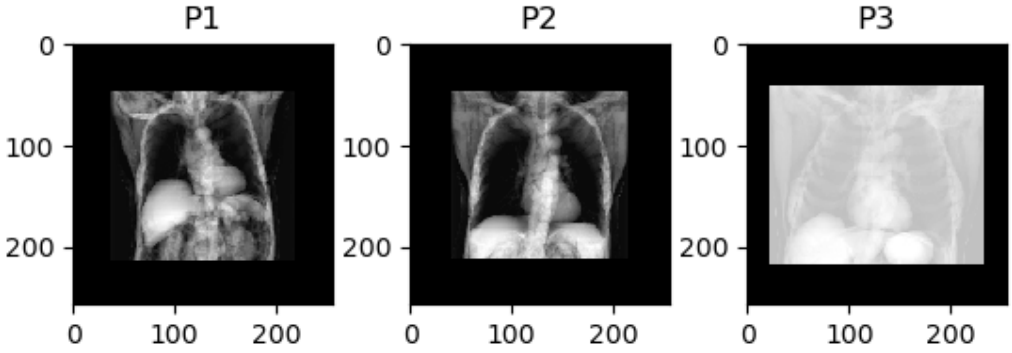


Figure A.16.: Labels vasculature authors

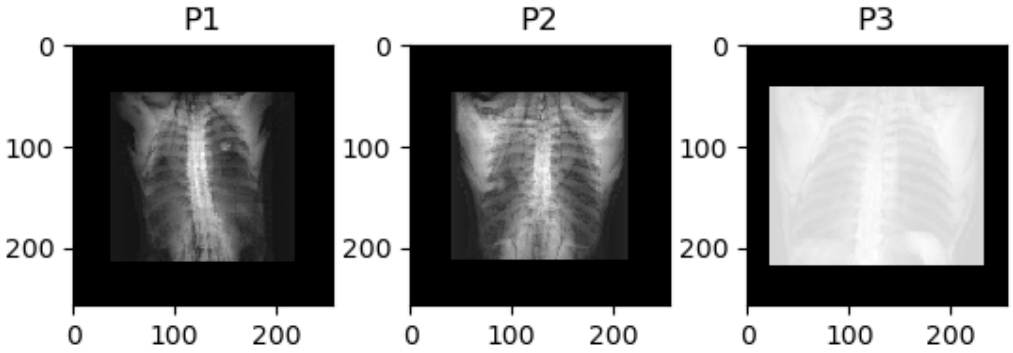


Figure A.17.: Labels spine authors

So, now a single data point in the dataset, pre-processed from the raw data, consists of a quadruple of DRRs, generated for the whole volume (input) and also for 3 different sub-volumes (labels) explained above.

Network Architecture : Figure A.18 shows the complete network architecture used by the authors in their paper. We planned to use the same for our task too.

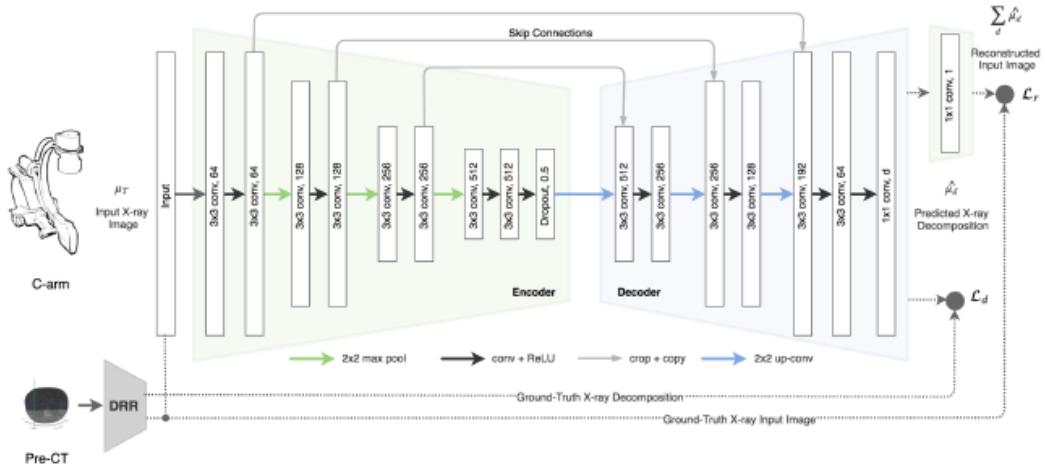


Figure A.18.: Network architecture authors

The network's input is a single 2D X-ray images/DRR, for the whole volume (also referred to as the Ground-Truth X-ray Input Image in the diagram) and is expected to output d (3 in our case) 2D X-ray images/DRRS (also referred to as the Ground-Truth X-ray Decomposition in the diagram), for the 3 sub-volumes, before the last layer. So the decomposition loss (L_d) is calculated by comparing these three outputs with the Ground-Truth X-ray Decomposition (or simply labels).

Also, in the final layer, they tried to reconstruct the original X-ray image/DRR from the decomposed output so as to explore any improvement in the process of decomposition, over the baseline consisting of only the decomposition module. Hence the reconstruction loss (L_r) is calculated by comparing this reconstructed single 2D X-ray/DRR with the Ground-Truth X-ray Input Image (or simply input).

The combined loss is used to optimise the network during the training process. In training phase, they obtained the DRRs by ImFusion software and in the testing phase they used real-world X-rays. They used SSIM and PSNR as metric for evaluation.

We also planned to use the similar metric and loss functions.

Drawbacks : The problem with this whole setup was that, in the paper, authors took help of radiologists to clip the sub-volumes of the individual patients. But that was out of scope for us. Furthermore, using a common template for clipping the whole CT volumes for the above-mentioned three regions of interest will not scale for all patients due to difference in lighting, positioning, volume of human bodies, method of acquisitions and several other artefacts.

Hence with use of template and seeking help from radiologists out of way, the topic was dropped. Also since the topic was already implemented by the authors, a newer and more interesting topic was designed. This newer topic also got evolved in phases.

A.2. Phase 2

In this phase, a new goal was set. Instead of trying to decompose an X-ray/DRR, captured by projecting the complete anatomy of a region of a human body, into 3 separate X-rays/DRRs, obtained by projection of non-overlapping anatomies of interest for the same region, it was decided to reconstruct a 3D CT volume image from a 2D X-ray image, as this novel concept has not been implemented before. Hence a successful implementation of same would mean a break through in the field of medical imaging and opening up several other scopes of research in related areas.

It would also mean countering the problems of expenses and high amount of radiation associated with a 3D computer-assisted CT imaging. Hence if a patient has an access to the most easily available and cheap form of medical imaging - the X-ray, this implementation could generate the detailed 3D volume image from the same and provide the necessary insight to physicians to look through the obscured regions of interest in a conventional X-ray and aid the process of medical intervention.

A similar version of the network architecture of 2D U-Net, described in phase 1, was used with a higher number of channels all through the hidden layers and 200 number of channels in the final layer, as it was decided that we would be generating 200 output slices for CT volume from 2D DRR.

For this task, the raw data of LIDC-IDRI, which has also been used in the 1st phase, was pre-processed before feeding into the network. Since every patient's entry in the raw data had unequal number of top-view slices along with higher dimension for each slice (512×512 (x,y)), each of them was required to be re-sampled to a definite number of top-view slices (200) with a reduced dimension of 256×256 (x,y) for the feasibility and ease of the training process, respectively.

This was accomplished using a setup of MeVisLab, which is described in the figure A.19.

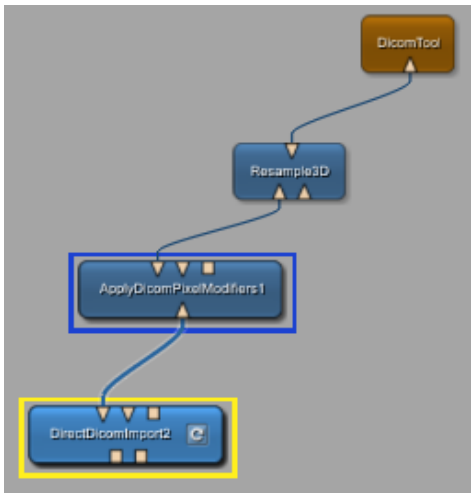


Figure A.19.: MeVisLab setup for CT volume resampling

The modules “DirectDicomImport” and “ApplyDicomPixelModifiers” have already been discussed in the first phase.

Resample 3D : The next new module in this setup is the “Resample3D” module. The details of the module are available in the figure A.20.

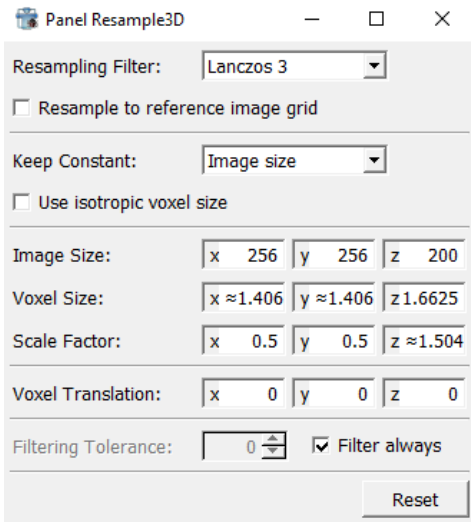


Figure A.20.: Resample 3D module MeVisLab

Either “Image Size” or “Voxel Size” or “Scale Factor” can be kept constant. Of course, “Image Size” was kept constant so as to have a required constant CT volume of $200 \times 256 \times 256$ (z, x, y) pixels. Hence,it resamples e.g. the CT volume of $133 \times 512 \times 512$ pixels of patient 1 to $200 \times 256 \times 256$ pixels, which is the final dimension for every resampled CT volume for each patient. Isotropic voxel size could not be selected as then it changes either the z dimension or both the x and y dimensions differently for each patient which would then negate the need for the resampling purpose to have a constant volume. The Resampling filter was by default chosen as Lanczos3 as other options like

Hamming, Hanning, Gaussian etc. did not give better results. Also the CT slices, resampled by this filter, are quite similar in terms of look and feel to the original slices as shown in the figure A.21. In this figure first (1) and last (133) slices of original CT volume of patient 1 from the dataset are compared with first (1) and last (200) slices of resampled CT volume respectively.

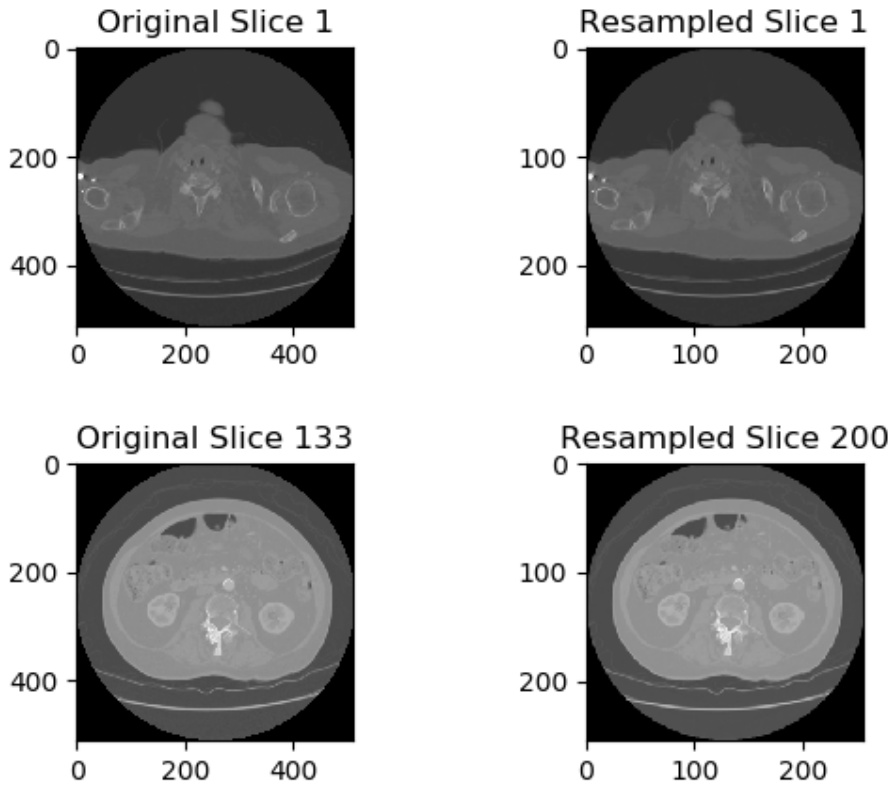


Figure A.21.: Original slices 1 (top left), 133 (bottom left) and resampled slices 1 (top right), 200 (bottom right) patient 1 MeVisLab

From the figure it is evident that the resampling filter did a good job in terms of resampling the whole CT volume with respect to the visual similarity between the original and resampled slices.

Dicom Tool : The next new module in this section is the "DicomTool", which is simply used for saving the resampled slices. Figure A.22 shows the inner details of this module.

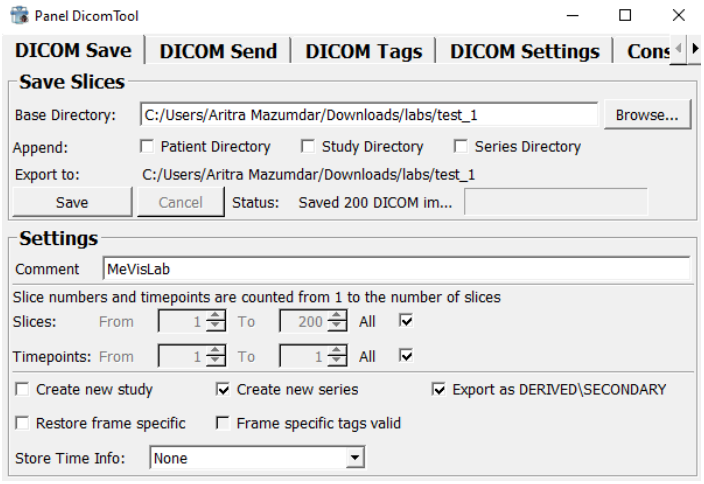


Figure A.22.: Dicom tool module MeVisLab

Once the labels or the CT volumes have been pre-processed to a normalised dimension of 200×256×256 pixels for the patients, frontal DRRs of dimension 256×256 pixels are acquired using the same setup of MeVisLab, described in phase 1, but with a little modification. We use the conical beam as opposed to the parallel beam this time. The difference between these two types of beams used for DRR generation is shown in the figure A.23.

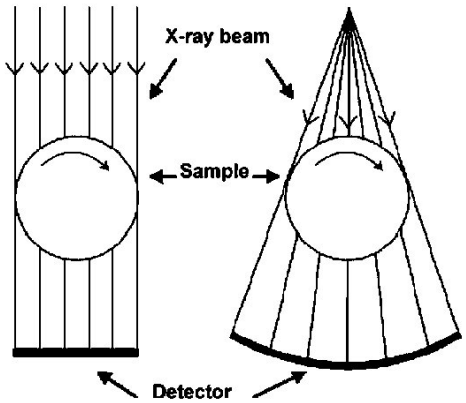


Figure A.23.: Parallel and conical beam of x-ray [image source: [61]]

This was done so as to have much more information of anatomy for the region of concern (thorax and abdomen) on the generated DRR, as a DRR generated by conical beam spans over the whole image in contrast with the ones generated by parallel beam. It helps a little bit with obscured anatomical features. This is clearly visible in the two types of DRRs generated for the same CT volume of same patient in figure A.24.

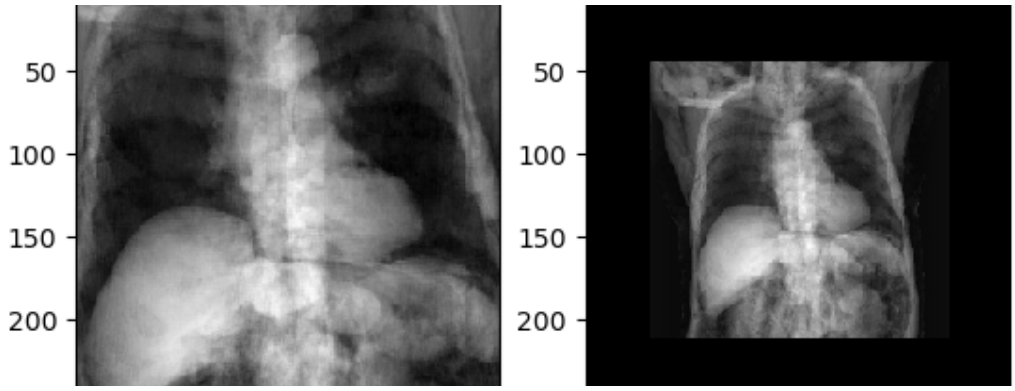


Figure A.24.: Frontal DRRs parallel (left) and conical (right) beam patient 1 MeVisLab

So now the pre-processed dataset consists of a frontal 2D DRR of 256×256 pixels as the input and a 3D resampled CT volume of $200 \times 256 \times 256$ pixels, which needs to be saved in reverse order post resampling, as the labels. The resaving of the CT slices post resampling is essential, as without that, we will be destroying the positional correlation between the resampled CT volume and the frontal DRR, thereby making it hard for the network to learn the seemingly non-existent relationship between vertically inverted top view CT slices and the non-inverted frontal view DRR and eventually to reproduce the same. Instead of resaving the CT slices for each patient, which would take an awful lot of time while considering the dataset of 1012 patients, it was decided to instead flip the whole input set of frontal DRRs vertically in one go using Python script and that would take less than a minute. Each of these vertically flipped frontal 2D DRRs and their corresponding resampled 3D CT volumes formed an entry for a single data point in the final dataset. An input-label pair of generated and vertically flipped frontal DRR and pixel normalised slices of CT volume corresponding to patient 1 in the dataset is shown in figure A.25. Slices 1, 100 and 200 are displayed in the picture.

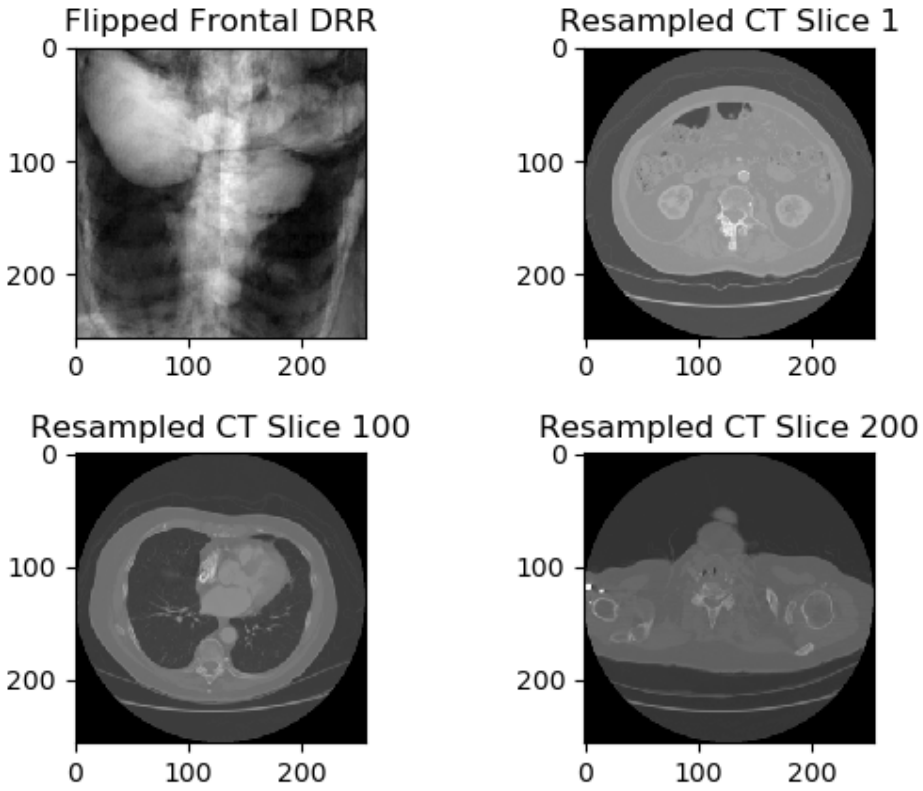


Figure A.25.: Flipped frontal DRR (top left) and resampled CT slices 1 (top right), 100 (bottom left), 200 (top right) patient 1 MeVisLab

Drawbacks : There were a major drawback associated with the process. With reference to the figure A.25, frontal view DRR was fed into the input of the network and top view CT scan slices were expected at the output, thereby forcing the network to learn the mapping between seemingly absolutely unrelated input and label. It is like inputting a 2D image of a mountain into the network and expecting a 3D volume object of a car at the output. There is very little correlation between the input and expected output and as a result there is hardly any useful information semantic representation in the input which the network can extract and make use of, while trying to reproduce an unrelated output volume. Hence the results were not satisfying at all as the network could barely learn anything and so they did not also improve with different kinds network architectures, parameters and hyperparameters that were tried and tested later in this phase.

A.3. Phase 3

The third phase was exactly similar to the second phase, except for the generation of DRR. An extra module was added in the setup which could generate a top view DRR instead of a frontal one. In that way, it would be much easy for the network to correlate and reproduce top view CT slices from seemingly similar top view DRR. The extended setup in MeVisLab is shown in the figure A.26.

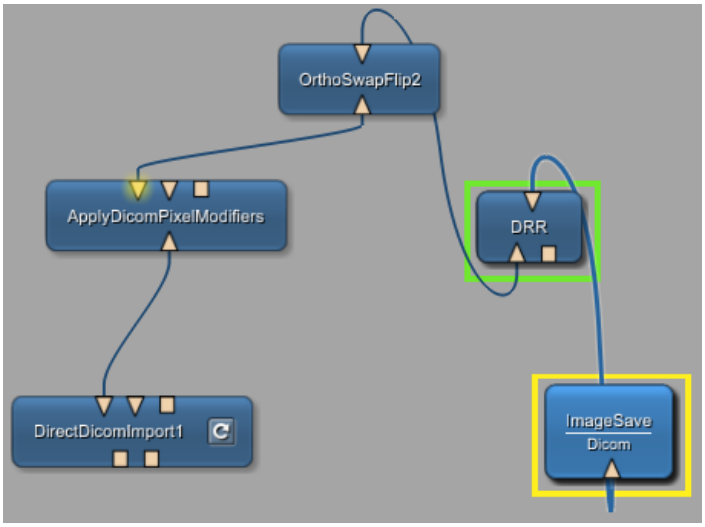


Figure A.26.: Extended MeVisLab setup for DRR generation

OrthoSwapFlip : The "OrthoSwapFlip" module is used to rotate the CT volume in such a way that top view DRR can be captured by passing a beam along the coronal direction. Figure A.27 shows the details of the module.

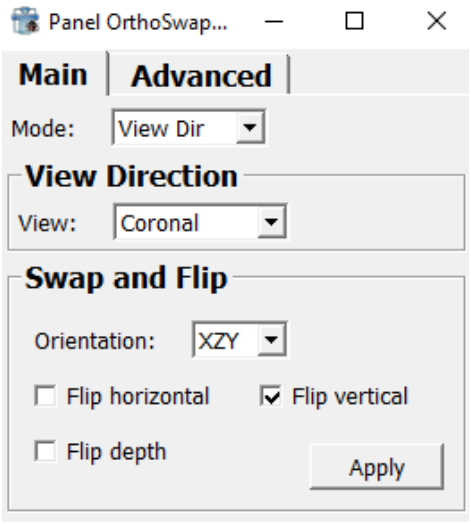


Figure A.27.: Orthogonal swap & flip module MeVisLab

Three different options are available under the view tab and those are transversal, saggital, coronal and others. These are basically directions of DRR trajectory. Coronal was selected from the options so as to get the top view DRR. There are similar other custom options available in orientation tab. The top view CT slices with the top view DRR for the same patient discussed in phase 2 now got transformed into what is visible in figure A.28.

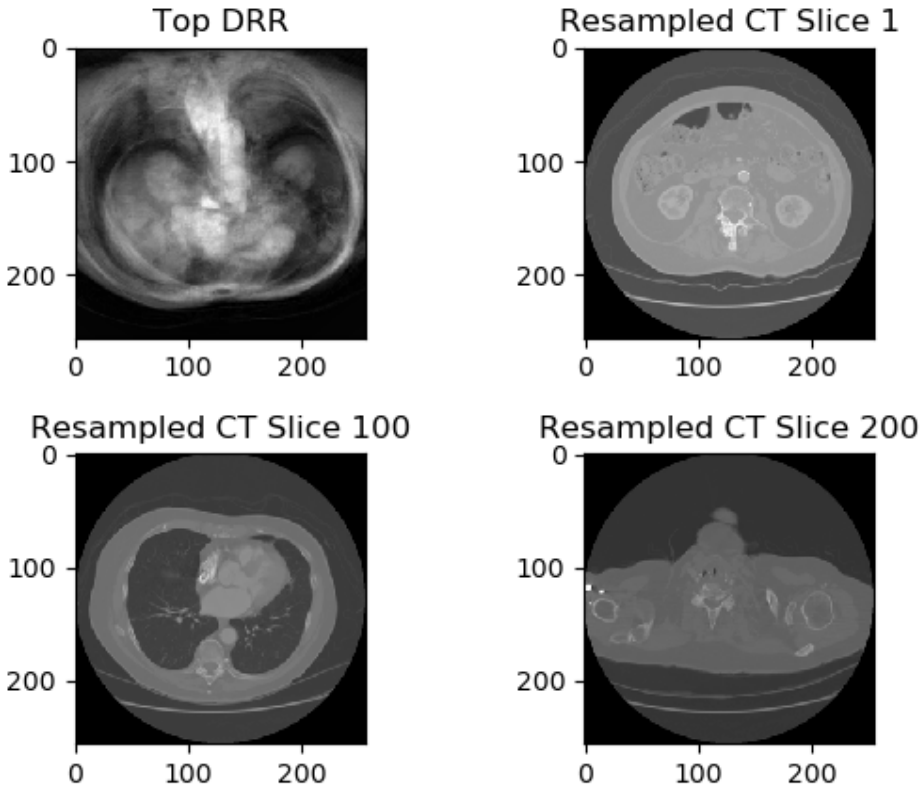


Figure A.28.: Top DRR (top left) and CT slices 1 (top right), 100 (bottom left), 200 (bottom right) patient 1 MeVisLab

It is evident that the generated top view DRR is vertically flipped (similar to second phase which needed resaving of the slices in reverse order) which needed manipulation in the settings tab “OrthoSwapFlip” module. In the “Swap and Flip” section, “Orientation” was set to “XZY” and Flip depth and Flip vertical were both selected and then it generated the appropriate input-label pair for this phase. Figure A.29 shows the corrected input-label pair.

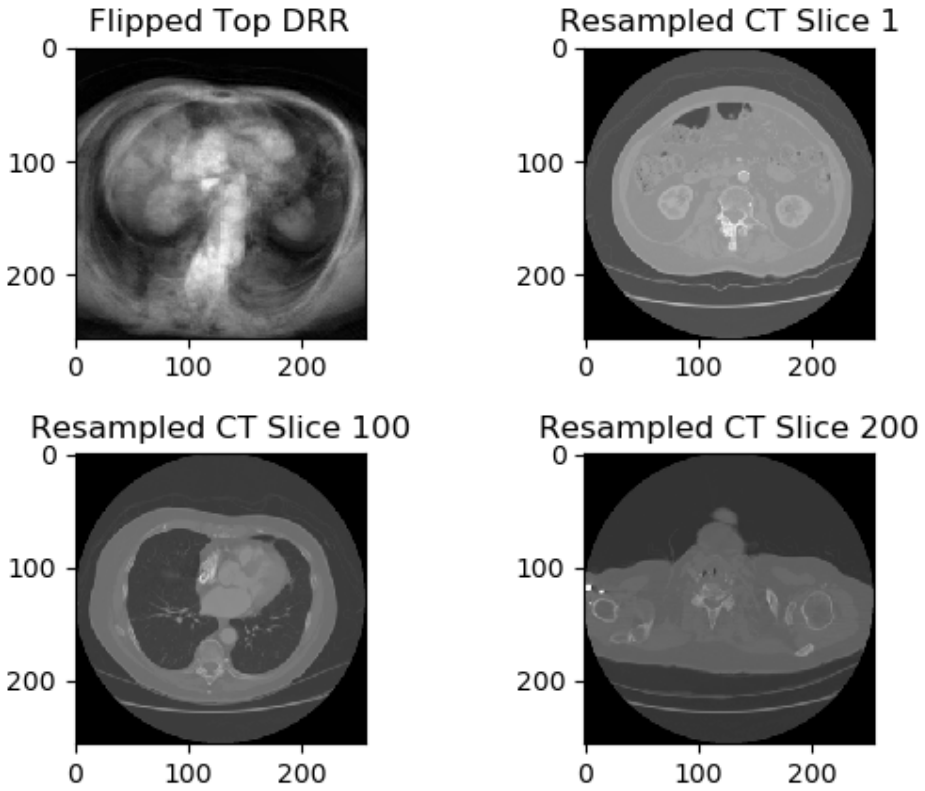


Figure A.29.: Flipped top DRR (top left) and CT slices 1 (top right), 100 (bottom left), 200 (bottom right) patient 1 MeVisLab

During this time, results got better compared to the previous setup due to the fact that the network can now extract useful features from 2D DRR and try to reproduce a similar looking CT volume. In the second stage for this phase, both the top and frontal DRRs were fed into the network and it improved the results by a bit, thereby indicating that an additional view could provide a little more perspective to the network to help it in it's task of decomposition. In the third phase, it was decided to explore the impact of triple view DRRs, which meant inclusion of the lateral view too. But the work for this phase was halted due to the reason mentioned as the last drawback in the next subsection and also because we luckily came across the dataset used in the next phase, which countered this particular problem.

Drawback : Although results got much better compared to the previous phase, because of the new found correlation among seemingly related input-label pair, it is practically impossible to obtain a top view DRR/X-ray for a patient with the existing X-ray imaging medical setup. Second drawback was that the dataset had a lot of noisy data points, which does not provide any fruitful information or might even misguide the network, when it is trying to find the perfect semantic correlation between the DRRs and the CT scans. The last one was that the DRRs simulated electronically by MeVisLab lacked the real look and feel as a real-world X-ray, obtained physically from a radiologist. So in the next phase, these three drawbacks were countered.

A.4. Phase 4

To counter the last two drawbacks in the last phase, a new dataset was used. This dataset[8] was a combination of five different datasets, LIDC-IDRI, Clinical Proteomic Tumor Analysis Consortium Lung Squamous Cell Carcinoma (CPTAC-LSCC), Clinical Proteomic Tumor Analysis Consortium Lung Adenocarcinoma (CPTAC-LUAD), The Cancer Genome Atlas Lung Squamous Cell Carcinoma (TCGA-LUSC) and The Cancer Genome Atlas Lung Adenocarcinoma (TCGA-LUAD) and has been used by O. Gozes and H. Greenspan in their own task 'Bone Structures Extraction and Enhancement in chest radiographs via CNN trained on Synthetic Data'. This dataset had already generated frontal DRRs, which had a much realistic look and feel. Also it took care of the noisy examples by keeping only the good ones and removing the noisy ones. To counter the first drawback, now frontal CT slices were resampled using the same MeVisLab setup, discussed in the first phase with just an additional module as shown in figure A.30.

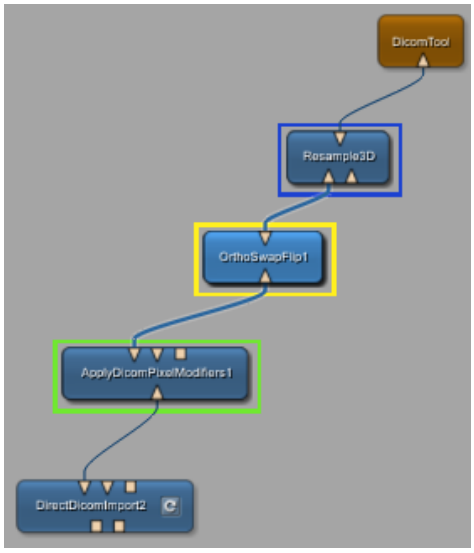


Figure A.30.: Extended MeVisLab setup for CT volume resampling

The additional module of "OrthoSwapFlip" rotates the CT volume in such a way that resampling would now produce frontal slices instead of top ones in the same way frontal DRR is transformed into top view DRR. We initially used 30 vertical slices. Figure A.31 shows the newly generated front view 2D DRR already available in the dataset and the corresponding frontal CT slices.

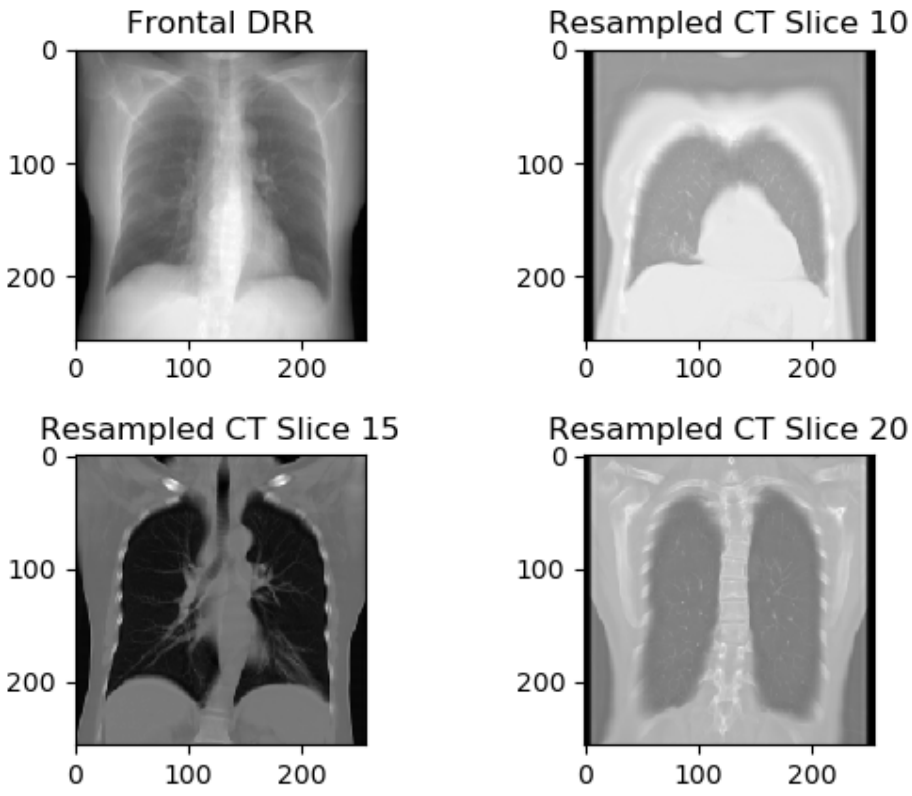


Figure A.31.: Frontal DRR (top left) authors and CT slices 10 (top right), 15 (botthom left), 20 (bottom right) MeVisLab patient 2 MeVisLab

Drawbacks : The already existing DRRs looked much more realistic and free of artefacts compared to the DRRs generated by the MeVisLab. This is clearly visible in the figure A.32.

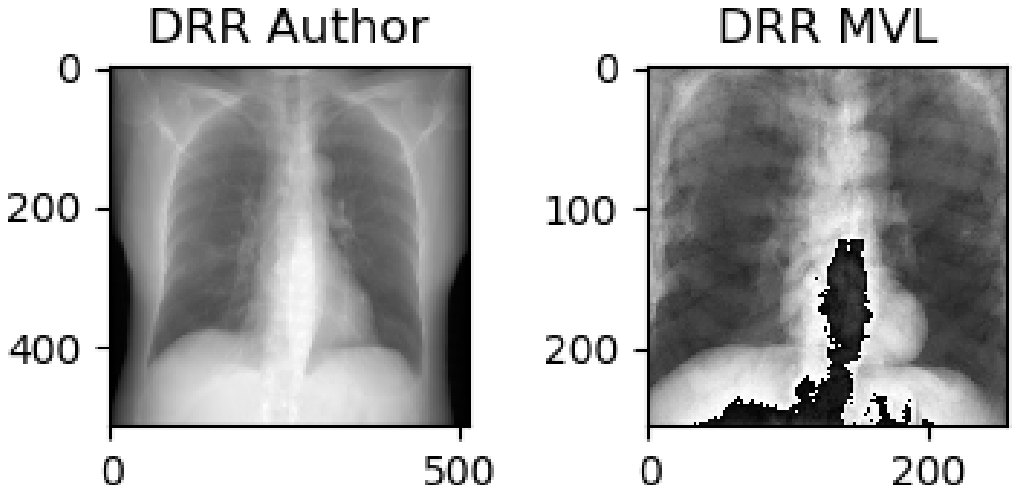


Figure A.32.: DRR generated by author (left) and DRR generated by MevisLab (right)

But the generated CT slices did not conform to isotropy, which is required to have uniformity across the dataset for both inputs and labels. Additionally, the lesser number of 30 slices which now made up the complete CT volume has much less information to detail compared to the original larger volume. Hence it included some noise while trying to compress the information.

A.5. Phase 5

To deal with the problem of isotropy, in this phase, finally a Python script was formulated which accomplished the resampling of the CT to a dimension of $256 \times 256 \times 256$ pixels, for each patient, while maintaining the isotropy of pixels across the slices and across all the patients in the dataset. It took care of the drawbacks related to the missing details and non-isotropic aspect in 30 slices CT volume data in previous phase. It can even generate DRR for the corresponding CT volume for all the patients in one go, thereby immensely saving time and effort and also providing the flexibility to make any change to the raw data as per requirement for generation of multiple datasets easily and quickly. Also, for this phase, DRR was generated from three different view points: frontal, top and lateral.

Isotropy and 256 slices led to some advancement in generated results when compared to the last phase. The raw data that was used from this phase onward was the cleaner version of the LIDC-IDRI dataset used by the authors in the last phase and the rest 4 datasets were excluded. Figure A.33 shows the Original CT Slices (OCS) 75, 125 and 175 in first row and the triple view Original DRRs (OD) in second row for patient 206 from the above-mentioned dataset.

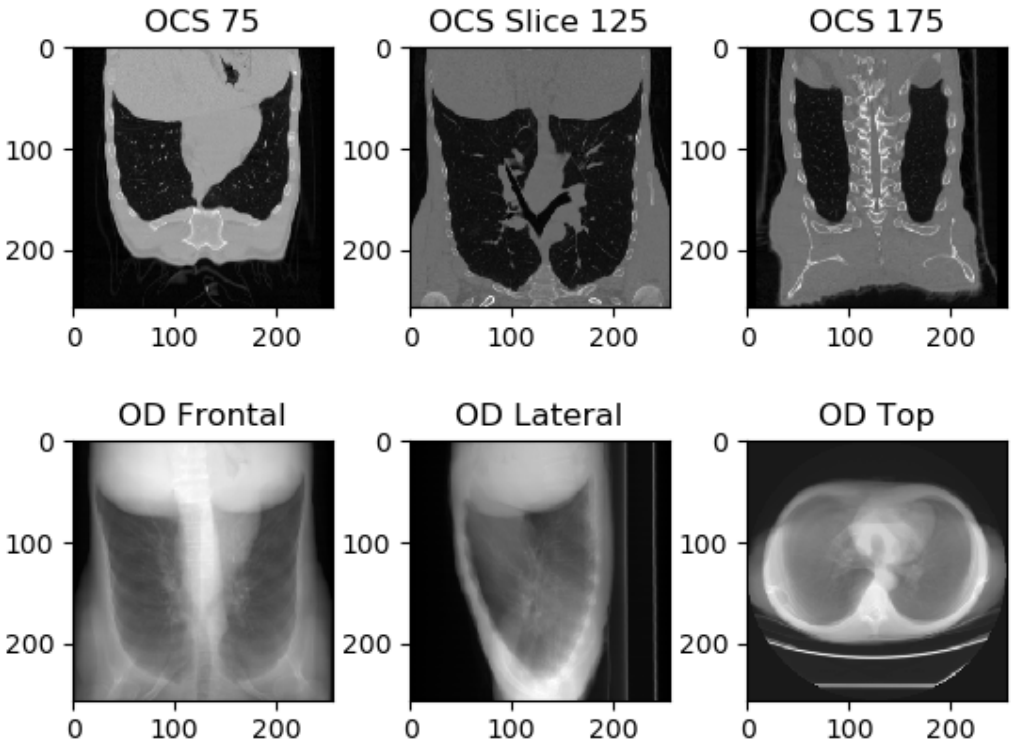


Figure A.33.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Original DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) patient 206 Python script

Different data augmentation techniques were used in this phase and afterwards too, by means of Albumentations, which is a library specialised for image augmentation for input and label 2D or 3D, at the same time, with the same augmentation states for both. These techniques include operations like “Shift Scale Rotate”, “Random Crop and Reshape”, “Horizontal Flip”, “Vertical Flip”, “Elastic Transform”, “Random Brightness”, “Random Contrast” and “Gaussian Noise”. These data augmentation techniques were applied on inputs and labels were generated afterwards. The augmentation was applied only on the generated CT volume and triple-view DRRs were generated on the fly while data loading for the augmented CT volume.

A.5.1. Random Crop and Reshape

In this augmentation technique, we have cropped randomly the CT volume and their corresponding triple-view DRRs to shapes of $256 \times 180 \times 180$ pixels and 180×180 pixels respectively and then reshaped them to original shapes of $256 \times 256 \times 256$ and 256×256 pixels respectively, as a result of both the augmented CT volume and DRRs seem to have been cropped and enlarged. Figures A.34 and A.35 show the comparison between original and augmented CT volumes and also between original and augmented DRRs respectively before and after cropping and reshaping.

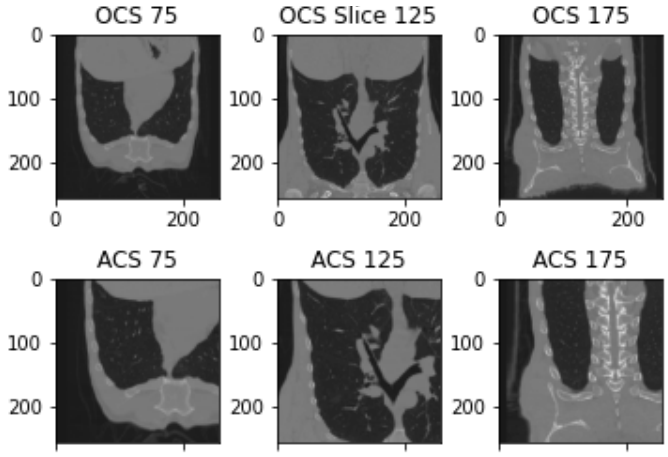


Figure A.34.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Random Cropping and Reshaping patient 206 Python script

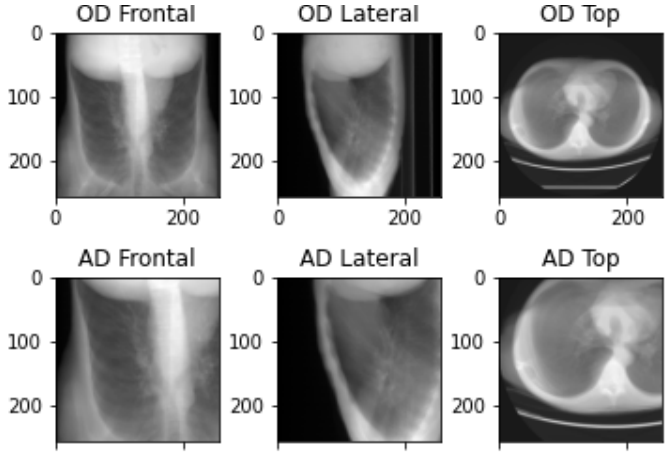


Figure A.35.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) with Random Cropping and Reshaping patient 206 Python script

A.5.2. Shift, Scale and Rotate

In this method, the CT volume and the triple-view DRRs were subjected to three different transformations, namely shift, scale and rotate, with limits set at a value of 0.2 for shift and scale functions and at 45 degrees for rotate operation. The comparison between the original and augmented CT volumes pre and post shift, scale and rotate transformations is shown in figures A.36 and the same between the original and augmented DRRs in A.37.

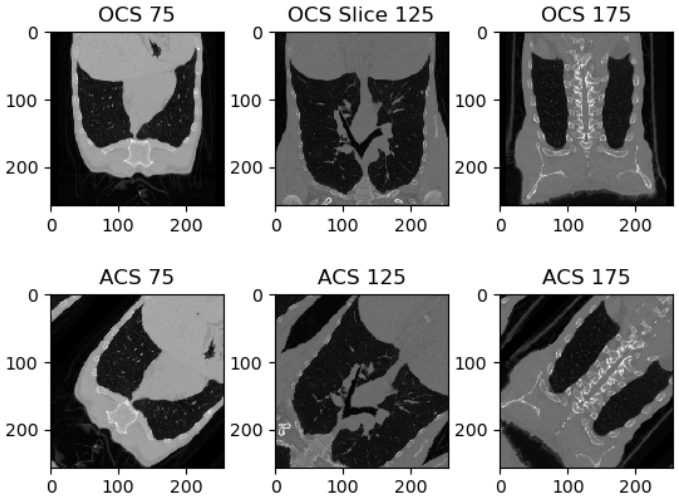


Figure A.36.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Shift, Scale and Rotate patient 206 Python script

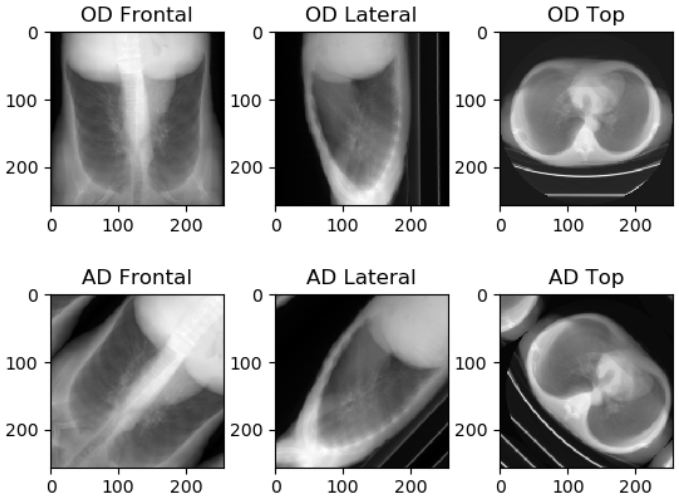


Figure A.37.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) with Shift, Scale and Rotate patient 206 Python script

A.5.3. Horizontal Flip

This augmentation technique flips the CT volume and the DRRs horizontally. Figures A.38 and A.39 show the impact of this transformation on the CT volumes and the DRRs respectively.

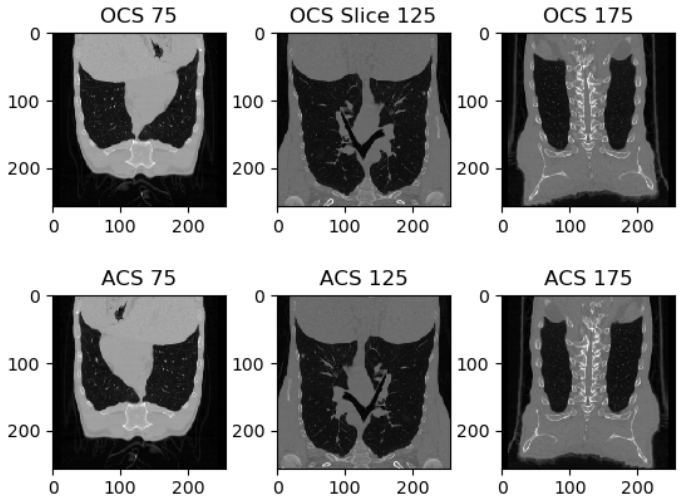


Figure A.38.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Horizontal Flip patient 206 Python script

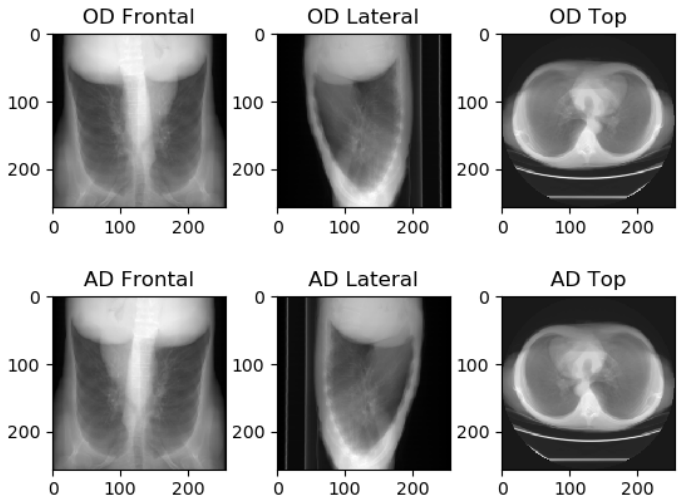


Figure A.39.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) with Horizontal Flip patient 206 Python script

A.5.4. Vertical Flip

This is a similar augmentation method to previous one and instead of flipping horizontally, it flips vertically the CT volume and also the DRRs. Figures A.40 and A.41 show the transformation vividly.

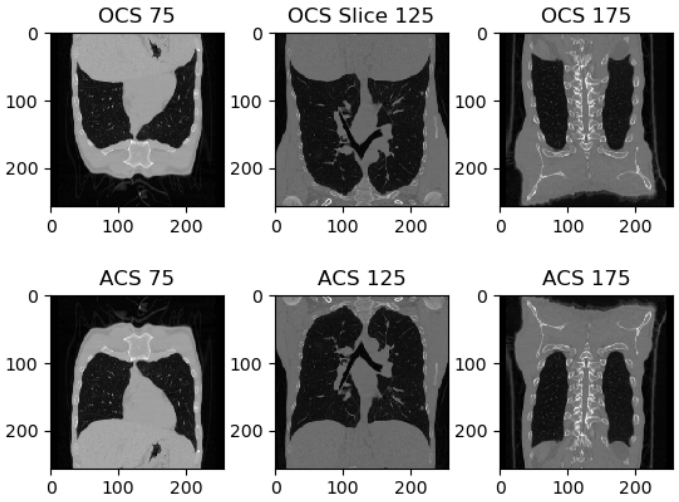


Figure A.40.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Vertical Flip patient 206 Python script

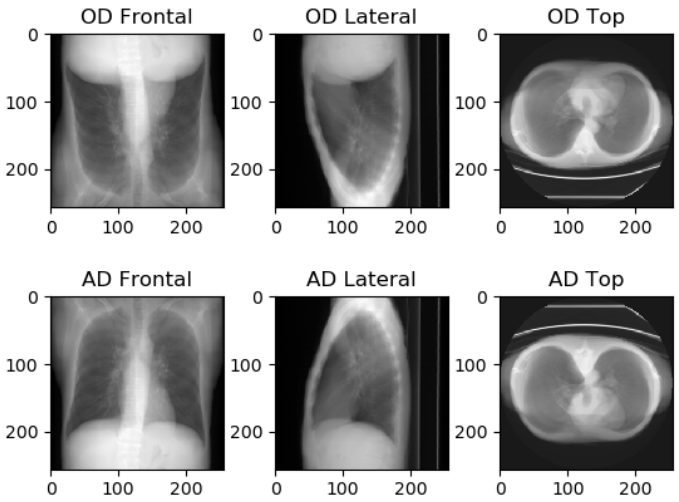


Figure A.41.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) with Vertical Flip patient 206 Python script

A.5.5. Random Brightness

It brightens up the entire CT volume and all the DRRs till an upper limit of 1. Figures A.42 and A.43 show the impact of random brightness operation on the input and the labels.

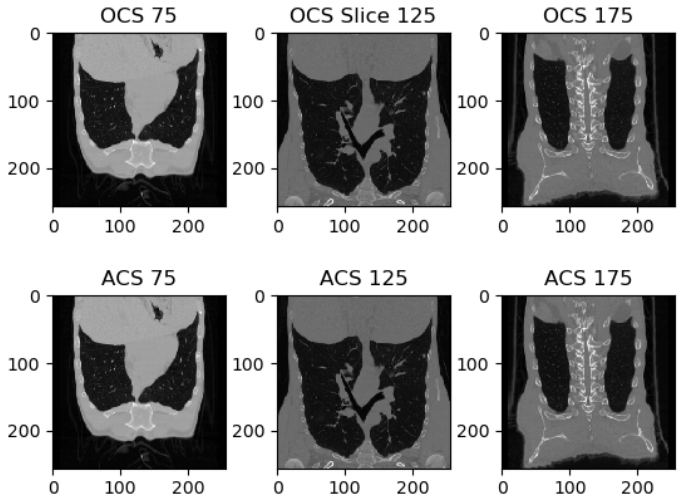


Figure A.42.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Random Brightness patient 206 Python script

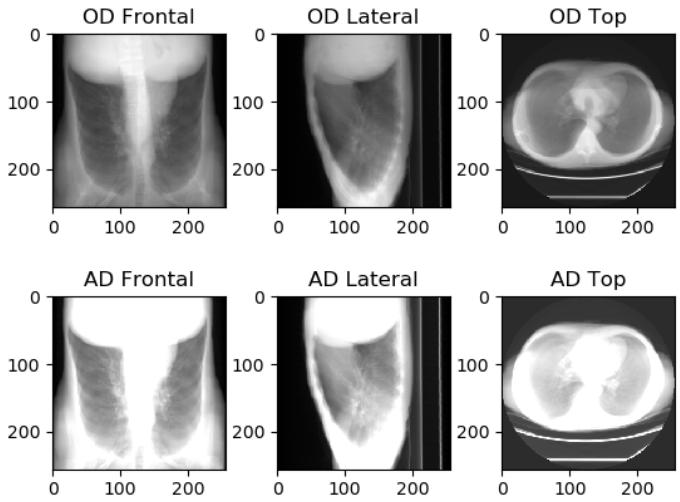


Figure A.43.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) with Random Brightness patient 206 Python script

A.5.6. Random Contrast

It adds or subtracts an additional contrast factor for the CT volume and the DRRs till an upper limit of 1. Figures A.44 and A.45 show the transformation of the CT volume and the DRRs respectively.

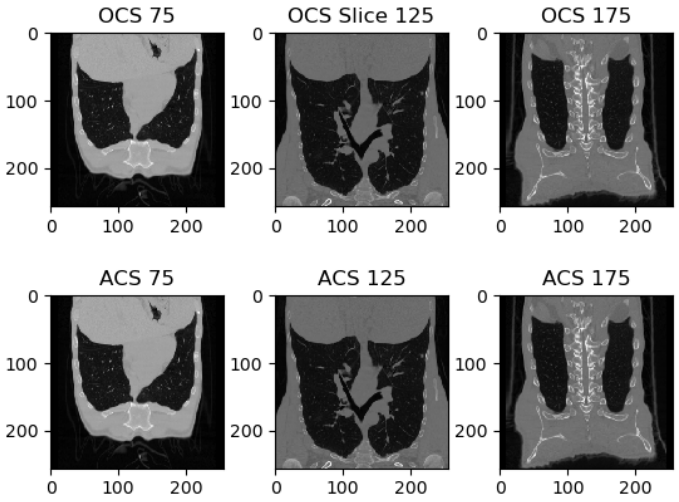


Figure A.44.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Random Contrast patient 206 Python script

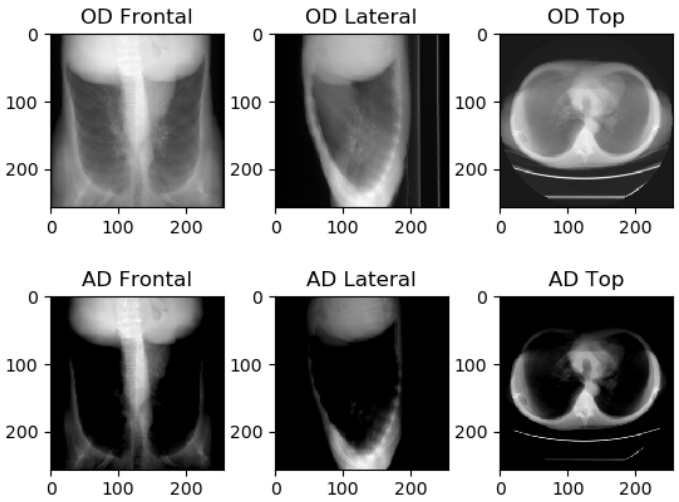


Figure A.45.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) Random Contrast patient 206 Python script

A.5.7. Gaussian Noise

It adds Gaussian noise to the complete CT volume and also for each of the 3 DRRs using a Gaussian filter with a variation limit for noise addition set in between 0 and 0.1 in our case. Figures A.46 and A.47 show the data augmentation effect.

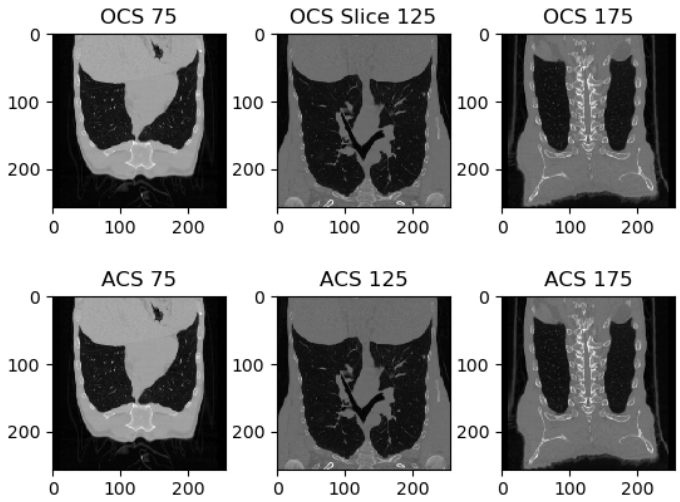


Figure A.46.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Gaussian Noise patient 206 Python script

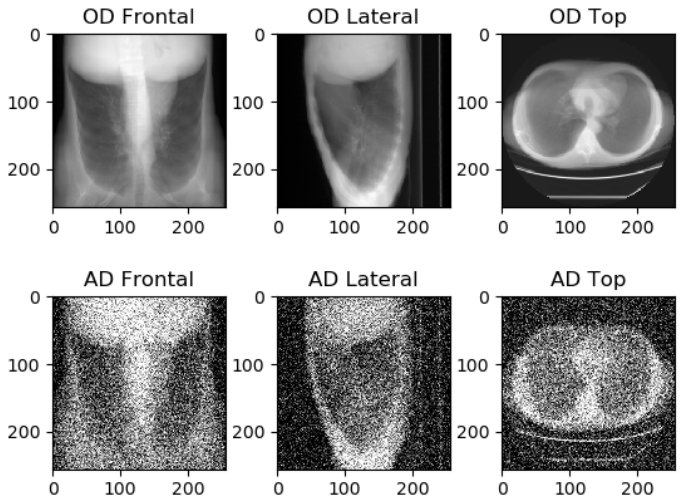


Figure A.47.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) with Gaussian Noise patient 206 Python script

A.5.8. Elastic transform

It provides an elastic transform to the CT volume and DRRs with upper and lower limits specified by the factors alpha and gamma which are set to 1 and 50, respectively in our experiments. Figure A.48 and A.49 show the comparison between pre and post transformation clearly.

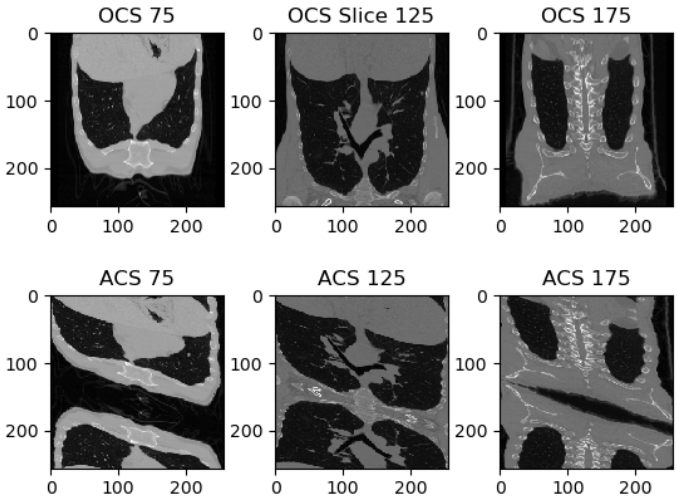


Figure A.48.: Original CT Slices 75 (top left), 125 (top middle), 175 (top right) and Augmented CT Slices 75 (bottom left), 125 (bottom middle), 175 (bottom right) with Elastic Transform patient 206 Python script

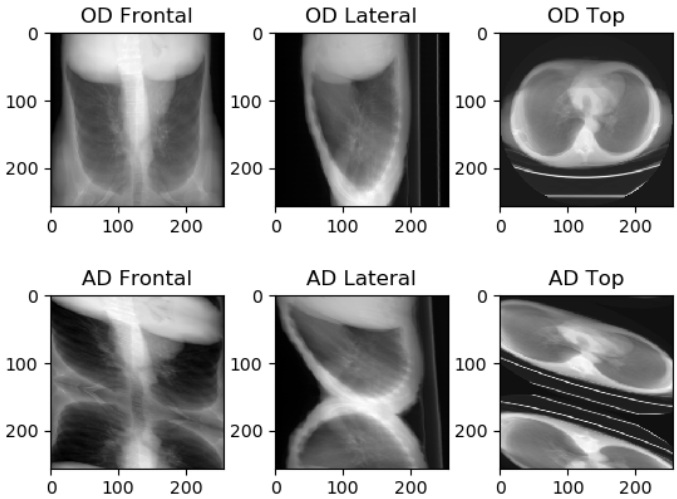


Figure A.49.: Original DRRs Frontal (top left), Lateral (top middle), Top (top right) and Augmented DRRs Frontal (bottom left), Lateral (bottom middle), Top (bottom right) with Elastic Transform patient 206 Python script

There was a little improvement with multiple views over single view DRR. Similar version of 2D U-Net was used. This time the network had 256 output channels, three input channels (for triple view DRR) and consequently higher number of channels in hidden layers.

Improvement : We completed this experiment only with decomposition loss. So we wanted to verify if the reconstruction loss network and the latent loss network could improve results over the baseline in the next phase.

A.6. Phase 6

In this phase, every aspect of the dataset pre-processing remained same. From the network perspective, reconstruction loss and latent space loss were also taken into account. The decomposition part of the network, which calculates the loss between the decomposed CT volume and the 3D CT labels, is still the same.

Right after the decomposition process, the reconstruction part of the network tries to reconstruct the triple view 2D DRRs from the decomposed CT volume by projecting the volume along it's respective dimension. This is where the reconstruction loss is calculated between these generated DRRs from the output of the reconstruction network and the original DRRs which were fed into the network.

For including the latent space loss, at first a separate network is trained to reconstruct the 3D CT volume from itself using the same network as that of decomposition. Once the latent loss network is trained, it's encoder is equipped well enough to guide the encoder of the primary decomposition network, with the additional latent space information, in the process of re-aligning it's weights in a way that facilitates the decoder with the task of decomposition of 2D DRRs into 3D CT volume. The latent space loss is calculated between the encoder outputs of the primary and latent loss network. The backpropagated error now updates the weight of the encoder part of the primary network with the information of the latent space loss. Except including these two different losses on top of the baseline, there was no other change made in this phase.

Improvement : The reconstruction loss helped in a little improvement over the baseline but contribution of the latent space loss was absolutely zero. Beyond this point there was no further scope of improvement except trying different network architectures, which brings us to the next phase.

A.7. Phase 7

As an improvement, it was decided that trying to reproduce the 256 layers (slices) at the output of the neural network separately might not be the best idea to go about it. The optimisation process might optimise the 2D layers individually, but eventually the optimised weights would miss out on the volumetric correlation among the layers. Hence it was decided to use a 3D U-Net instead of a 2D one. This way the network will try to reproduce a 3D CT volume as a whole taking into consideration that there is a relationship also among the layers on contrary to tuning 2D weights individually of 2D U-Net. For this phase, right after loading the triple view 2D DRRs, features were redistributed to go from 2D to 3D using a transformation module so to obtain the final expected 3D shape of output ($256 \times 256 \times 256$). Then a 4th dimension was introduced so as to incorporate the channel dimension.

Improvement : With no improvement in results, instead of using a completely 3D convolutional network right after the first input layer, another version of 3D U-Net was thought to be explored which is discussed briefly in next section.

A.8. Phase 8

In this phase, instead of using 3D convolutional layers all through the network, the step up from 2D to 3D took place right after the encoder and just before the decoder. In other words after the features, extracted from a 2D DRR which got transformed to a higher channel dimension but lower spatial dimension at the end of encoder, were now redistributed to give it a shape of a 3D volume before being fed to the decoder, which would then try to reproduce the CT volume at it's output. There was hardly any change in any other aspect. The results got visually a little better. The outer structures were prominently captured this time, but it missed out on inner detailing.

Improvement : Since, this was the maximum that could be achieved in this task with networks trained from the beginning, so transfer Learning was thought off as the next best alternative.

A.9. Phase 9

When there is not enough data points in the dataset or when there is a system constraint, then transfer learning helps to achieve miles never achieved before. By transferring the knowledge learnt in different domain and different datasets to another relatable domain and dataset and by tweaking a few initial and final layers, transfer learning is bound to give much better results than networks trained from scratch. Years of research in modifying the network and training it to different kinds of datasets by experts have been put in to gain the extra miles in pre-trained neural networks that leaves networks, trained from scratch, far behind. So transfer learning was thought of as a worthy alternative. So we made use of ResNet-152 weighted 2D U-Net, where we changed the first input layer to have three channels due to three projection DRRs and also the last layer to have 256 output channels so as to match the output z dimension. Additionally, we set the first and last two layers to trainable, while freezing the rest of the network. It gave worse results than any of the setups seen above with 256 slices.

Improvement : There were two basic reasons for results turning worse. Primarily, it was because of the weights being tuned to datasets which are totally unrelatable, as this network has not been trained on any clinical dataset for performing a relevant decomposition/reconstruction process using UNet. Hence with no modification of weights to this particular dataset for the whole network except for the first and last two layers, there was hardly any scope for the network to learn from the new dataset, while not having acquired the knowledge from similar datasets either. Increasing the number of trainable layers would mean meeting system constraints. Secondarily, the output channel was already reduced to a dimension, much lesser than 256, needed at the final layer, before the last two layers. So it has already lost a lot of information in reconstructing the output. So it made no sense in going from lower to higher channel dimension in the encoder and then from higher to lower and eventually from lower to higher in the last two layers of the decoder. It is like extracting the channel information in the encoder and then loosing out on the same to shape the spatial dimension correctly and then finally trying to rebuild the channel dimension in the end. Hence this idea was totally dropped which brings us to the next phase of implementation.

A.10. Phase 10

In this last phase of unsuccessful approaches, pre-processed dataset was once again acquired, but this time for 512 slices, each of 512×512 pixels. Hence, the newly produced CT volume for each patient was of the size $512 \times 512 \times 512$ pixels and the DRRs, generated on the fly, from the augmented version of this CT volume, was also of dimension 512 each. This was done so as to find out if increase of detail could help improve the results because in one of the past phases, 30 slices gave worse results compared to a similar setup with 256 slices. So we thought of giving it a try with increased number of slices and with increased dimension of each slice. Figure A.50 shows the triple view DRR along with front-view slices 150, 250 and 350 for the CT volume of patient 2 from the dataset.

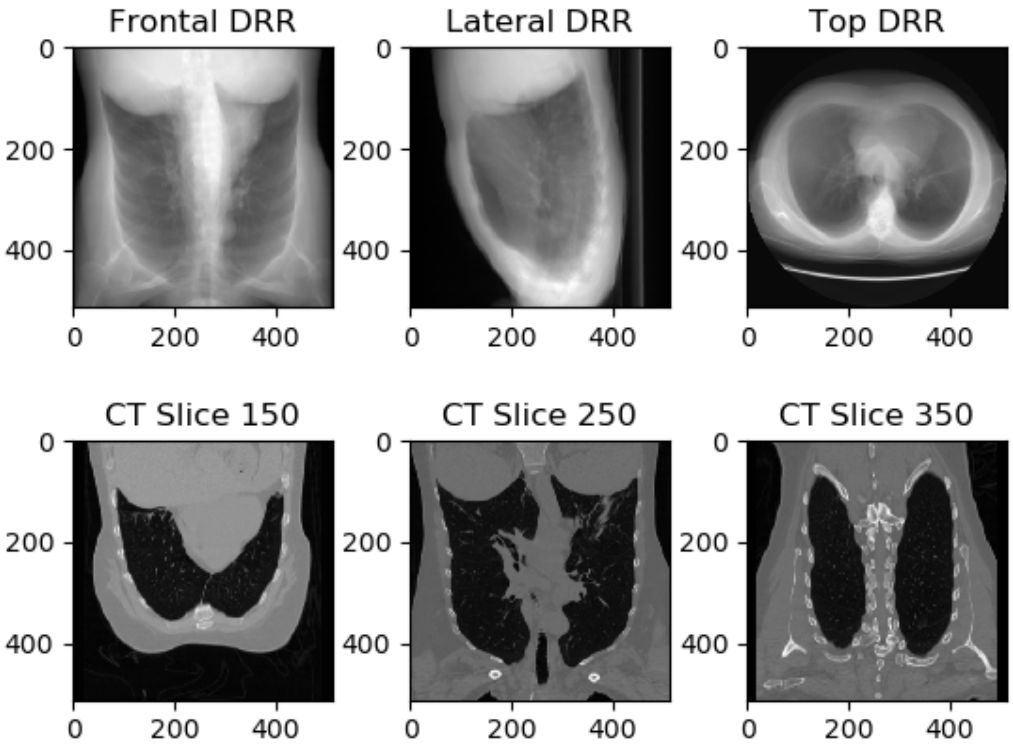


Figure A.50.: Triple-view DRR and frontal slices

Improvement : It did improve the results, indicating that we have reached the last level of what could have been achieved with this dataset and with this setup.

In the next phase, experiments were conducted with a well directed goal of identifying the impact of every parameter on the training process so as to include them in Results and Discussion chapter. This phase is discussed under 3.2 subsection of chapter 3 and the respective results are presented in chapter 4. More and more interesting observations appeared in this last phase, which is discussed in detail in section 3.2 in Experiments chapter.