Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Kristin Schive Hjelde

# Lungs and Lobes Semantic Segmentation in Mediastinal CT Scans Using 3D Convolutional Neural Networks

Master's thesis in Electronics Systems Design and Innovation
Supervisor: David Bouget, André Pedersen and Ilangko Balasingham
June 2020

NTNU
Kunnskap for en bedre verden

Kristin Schive Hjelde

# Lungs and Lobes Semantic Segmentation in Mediastinal CT Scans Using 3D Convolutional Neural Networks

Master's thesis in Electronics Systems Design and Innovation
Supervisor: David Bouget, André Pedersen and Ilangko Balasingham
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

# Summary

Pulmonary CT image analysis is a vital part in the assessment and treatment planning of different lung diseases. The method often requires the lungs to be separated from the surrounding structures, a process known as lung segmentation. Fissures divide the lungs into smaller compartments known as lobes, with their own independent vessels and airways system. Some diseases develop only in one of these lobes, and some treatments are performed on a lobar level. It is thus necessary to also segment the lobes in many cases. Doing lungs and lobes segmentation by hand is a tedious and time-consuming process, requiring expert radiologists. A variety of different methods to automate the segmentation process have been proposed throughout the years, but many of them struggle on lungs with high amounts of abnormalities, which often is the case for diseased lungs.

This thesis aims at contributing further towards the full automation of lung and lobe segmentation by investigating two state-of-the-art neural network architectures, and their ability to generate accurate segmentation models with short training and inference time. The two studied approaches are the 3D U-Net, and the relatively new PLS-Net. Both are 3D fully convolutional networks whereby the U-Net is more traditional with a large number of parameters while the PLS-Net is extremely lightweight in comparison. The two networks were tested for the lung segmentation task, using different combinations of deep learning frameworks (e.g., Tensorflow and PyTorch), training precision, batch sizes and input resolutions. In addition, the PLS-Net was trained and tested for the lobe segmentation task, together with a simple post-processing step.

The results showed that using smaller input volumes with batch size 2 gave higher accuracy than using larger input volumes with batch size 1. PyTorch and TensorFlow gave equally good Dice scores on the lung segmentation task, and PyTorch gave better training performance, but was slower during inference. Using mixed precision over full precision reduced the memory footprint by 40%, without reducing the accuracy. The PLS-Net used 30% less memory than the U-Net for the same input data and batch size, with a reduction of 0.3% in Dice score. For the lobe segmentation task, the PLS-Net gave a Dice score of 92.6% before post-processing, and 92.9% after. The lobe segmentation model performed equally well on data from another data set and the data used for training, but struggled on samples containing abnormalities.

# Sammendrag

Pulmonal CT-bildeanalyse er en viktig del av vurderingen og behandlingsplanleggingen av forskjellige lungesykdommer. Metoden krever ofte at lungene skilles fra de omkringliggende strukturene, en prosess kjent som lungesegmentering. Dype spalter (fissurer) deler lungene i mindre volumer kjent som lapper, med sine egne uavhengige åre- og luftveisystemer. Noen sykdommer utvikler seg bare i en av disse lappene, og noen behandlinger utføres på et slikt lappenivå. Det er derfor også nødvendig å segmentere lappene i mange tilfeller. Å gjøre segmentering av lunger og lapper for hånd er en langtekkelig og tidkrevende prosess, som krever eksperter innen feltet. Det er foreslått en rekke forskjellige metoder for å automatisere segmenteringsprosessen gjennom årene, men mange av dem sliter på lunger med store mengder abnormaliteter, noe som ofte er tilfelle for syke lunger.

Denne oppgaven tar sikte på å bidra ytterligere mot full automatisering av lunge- og lappesegmentering ved å undersøke to toppmoderne nevrale nettverksarkitekturer, og deres evne til å generere nøyaktige segmenteringsmodeller med kort trening- og inferenstid. De to nettverkene som skal undersøkes er 3D U-Net, og det relativt nye PLS-Net. Begge er 3D konvolusjonelle nettverk der U-Net er tradisjonelt med et høyt antall parametere mens PLS-Net er ekstremt lett til sammenligning. De to nettverkene ble testet for lungesegmenteringsoppgaven ved bruk av forskjellige kombinasjoner av rammeverk for dyp læring (f.eks. Tensorflow og PyTorch), treningspresisjon (16-bit vs. 32-bit), batchstørrelser og volumstørrelse. I tillegg ble PLS-Net trent og testet for lappesegmenteringsoppgaven, etterfulgt av et enkelt postprosseseringssteg.

Resultatene viste at bruk av mindre volum med batchstørrelse 2 ga høyere nøyaktighet enn å bruke større volum med batchstørrelse 1. PyTorch og TensorFlow ga like gode Dice-verdier på lungesegmenteringsoppgaven, og PyTorch ga bedre treningsytelse, men var tregere under inferens. Ved å bruke blandet presisjon over full presisjon, ble minneavtrykket redusert med 40%, uten å redusere nøyaktigheten. PLS-Net brukte 30% mindre minne enn U-Net for samme volum og batchstørrelse, med en reduksjon på 0,3% i Dice-verdi. For lappesegmenteringsoppgaven ga PLS-Net en Dice-verdi på 92,6% før postprossesering og 92,9 % etter. Lappesegmenteringsmodellen presterte like bra for data fra et annet datasett som på dataene brukt til trening, men slet på lunger som inneholdt abnormaliteter.

# Preface

This thesis represents the end of my master's degree in Electronics System Design and Innovation at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. These five years as a student have given me the opportunity to work on many different projects, and I have gotten to know many interesting people.

I started working on this task as part of my specialization project in the fall, and chose to continue the work through my master thesis, as I found deep learning in combination with the medical field very interesting. After two months of working, the COVID-19 pandemic shut down the country, and employees and students were ordered to continue the work from home. Even though the distance to my supervisors changed from ten short meters away, to being an internet connection away, the situation did not affect the progress much. Seeing CT images of lungs on the news in connection with the pandemic, only increased my motivation to continue the work, and truly showed me the importance of the field.

Looking back, I have learned a lot during this process, both about lung anatomy, deep learning and not to mention how to wrap all this into a final report.

## Acknowledgements

Abdolreza Sabzi Shahrebabaki deserves a thank you for helping me set up the remote machine used to produce the results for this thesis. He always responded quickly to my emails whenever I ran into an issue with the system.

I am also grateful to the Health Research department at SINTEF Digital, and Trond Røvik Størseth, not only for letting me use the precious time of my two supervisors, but also for supplying me with an office space and computer equipment. Thomas Langø made sure to include me in his Lung group, where I was invited to participate in meetings, with experts from a variety of fields. This gave me better insight behind the motivation and the common goal everyone is working towards.

Most of all, I am grateful for my main supervisor David Bouget, for all the guidance and support through both the specialization project and now the master thesis, and for co supervisor André Pedersen, who officially joined us at the beginning of the year. They have always made time for me, and their genuine interest in seeing this project succeed, have inspired me through the entire process.

Lastly I would like to thank my boyfriend and fellow student Kristoffer Røise, for continuously listening to my thoughts, problems and frustrations, and for help with proofreading and corrections. For moral and practical support, I would like to thank my parents.

Kristin Schive Hjelde
Trondheim, June 13, 2020

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

| | |
|---|---|
| **CNN** | Convolutional Neural Network |
| **CT** | Computer Tomography |
| **DAG** | Directed Acyclic Graph |
| **DC** | Dice Coefficient |
| **DSC** | Dice Score |
| **DRDB** | Dilated Residual Dense Block |
| **DS** | Depthwise Separable |
| **HU** | Hounsfield Unit |
| **MLP** | Multilayer Perceptron |
| **PLS-Net** | Pulmonary Lobe Segmentation Network |
| **ReLU** | Rectified Linear Unit |
| **SGD** | Stochastic Gradient Descent |

# 1 | Introduction

## 1.1 Background

Lung diseases are the cause of millions of deaths worldwide [1]. By detecting these diseases at an early stage, the chance of surviving will be increased. In pulmonary image analysis, computer tomography (CT) images of the lungs are analysed for abnormalities. A vital part of pulmonary image analysis is to separate the lungs from the surrounding structures [2], a process known as lung segmentation.

Pulmonary lobes are different regions of the lungs, divided by boundary surfaces known as pulmonary fissures. Each of the lobes have their independent vessels and airway system, which leads to some diseases only developing at a lobar level. Examples of such diseases are centrilobular emphysema and tuberculosis [3]. Some treatments of lung diseases, such as lung cancer surgery, are also performed on a lobar level. Thus, segmenting the lobes is of clinical interest both in disease diagnostics and treatment planning.

All lungs and lobes suffer from inter-patient variability in shape and structure. Diseased lungs are referred to as *pathological lungs*, and often contain abnormalities as many lung pathologies change the tissue density. In turn, this changes the intensity in the CT image, making it hard to do segmentation based on intensity only [4]. The fissures that divide the lobes have similar Hounsfield unit (HU) as vessels, and can be challenging to distinguish between each other. In addition, it is common that one or more fissures are incomplete. This makes lobe segmentation a challenging task, especially in pathological lungs.

Manual delineation of lungs and lobes is time consuming and require extensive knowledge of the anatomical structure of the lungs. Different approaches have been proposed to automate the process throughout the years, including a variety of different algorithms, both semi-automatic [5] and fully automatic methods [2, 6]. While many of them have proved to perform well on healthy lungs, and lungs with minimal abnormalities, the general trend is that they struggle in cases with moderate to high amounts of abnormalities [7]. In most recent efforts, deep neural networks using supervised learning have been utilized for the segmentation tasks. In 2007, Harrison et al. proposed a 2D network that proved to give good results on a large variety of pathological lungs [8]. Later, several 3D networks have also been proposed for both the lung and lobe segmentation task [9, 10].

Training deep neural networks may be time consuming, and require a lot of computational power and memory. In addition, a lot of annotated data is needed to achieve good results using this method. Such data is often hard to obtain due to strict privacy regulations of patient information, and the need of an expert radiologist to create proper annotated data. However, applying fully trained deep learning models is significantly faster than manual delineation, and have proven to outperform the traditional algorithm based methods in cases with high amount of abnormalities [7].

## 1.2 Aim and Method

Segmentation of the lung and its lobes in CT images is an important part in both disease assessment and treatment planning of lung diseases. While there has been made great efforts in the field of automating the process in the recent years, there is still a way to go before these methods can fully reach the accuracy of experienced radiologists. This thesis aims to contribute further towards the automation of lung- and lobe segmentation through investigating the feasibility of training accurate segmentation models with shorter training and inference time. The already well established 3D U-Net by [11] was used together with the relatively new, but successful PLS-Net by Lee et al. [12], which is extremely lightweight compared to the U-Net.

The impact on training and inference performance when using PyTorch over TensorFlow will be tested for lung segmentation using the U-Net, in addition to investigating the accuracy performance of different batch sizes and input resolutions, and the effect of using mixed precision over full precision. A comparison of the PLS-Net to the U-Net will be made with respect to training, inference and accuracy performance. The PLS-Net will also be used to investigate the lobe segmentation problem, in combination with a simple post-processing step.

This master's thesis is a continuation of a specialization project that started last semester. The theory chapter is an extended version of the one in the project. In the section on human lungs, lobes are now covered. Depthwise separable and dilated convolutions, residual learning and dense connectivity is added to the section on deep learning, and an introduction to the deep learning frameworks TensorFlow and PyTorch is provided at the end of the chapter.

## 1.3 Outline of Thesis

The motivation behind this thesis has now been introduced in this first chapter. In the second chapter, theory covering the basics of the human lungs, an overview of the imaging technique used to obtain the data, and an introduction to deep learning techniques are presented. The data and details behind the two network architectures used in the experiments are presented in Chapter 3, together with a description of the validation study and implementation details. In Chapter 4, the results from the experiments are presented, before they are discussed in Chapter 5. A conclusion based on the results and discussion is given in Chapter 6.

# 2 | Theory

In this chapter, some theory needed to understand the purpose and methods of this thesis will be presented. In the first section, a basic introduction of the anatomical structure of the human lungs will be given. The second section presents theory behind the method used to obtain the data used in the experiments. Together, these two sections will help the reader understand and interpret what the data is showing as it is being presented later in the thesis. In the third section, some general deep learning principles will be introduced. Some understanding of these principles will be necessary to understand the methods used later on.

## 2.1 The Human Lungs

The lungs are part of the human respiratory system, which main task is to do gas exchange with the blood. During cellular respiration the cells release chemical energy from oxygen molecules to fuel cellular activity. As part of this process, carbon dioxide is released as a waste product. The oxygen molecules are transported to the cells through the bloodstream, which also picks up the carbon dioxide and transports it to the lungs. The lungs then transfer oxygen from the atmosphere into the bloodstream, and releases the carbon dioxide from the bloodstream into the atmosphere [13].

The lungs are located beneath the rib cage in the chest, surrounded by a membrane known as the pulmonary pleurae. At some points, this pleura folds in to the lungs as pulmonary fissures, dividing the lungs into smaller regions. The resulting sections are known as pulmonary lobes, hereby referred to as the lobes. The left lung is divided into a superior lobe and a lower lobe, by the left oblique fissure. The right lung is divided by both a right oblique fissure and a horizontal fissure. This results in three lobes for the right lung: the superior lobe, the middle lobe and the lower lobe, as shown in Figure 2.1.1.

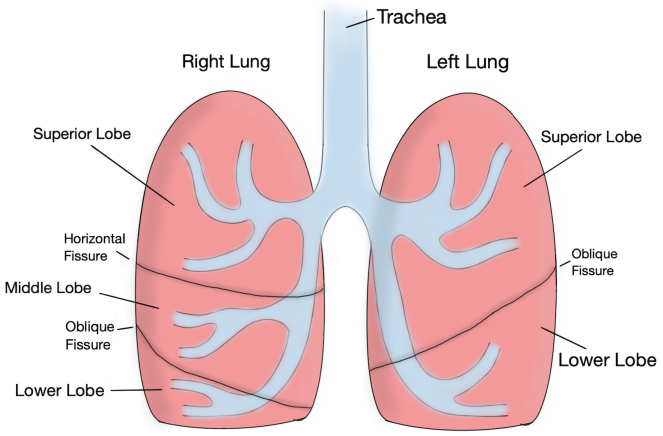



**Figure 2.1.1:** Simplified illustration of the human repository system and the lungs.

During inhaling, air from the atmosphere is transported to the lungs through the trachea. When the trachea reaches the lungs, it branches into a right and a left mainstream bronchi. These mainstream bronchi are further divided into five lobar bronchi, where each of the branches supplies one lobe with oxygen. Inside the lobes, the lobar bronchi divides into bronchioles.

As each lobe is supplied with air from their own, independent lobar bronchi, some diseases only develops at a lobar level. These types of diseases may include lung cancer [14] and lobar pneumonia [15]. Therefore, it is often interesting to examine and treat diseases at a lobar level. In a lobectomy for example, one or several lobes are removed together with nearby lymph nodes, through surgery. In the planning of such surgeries, knowledge of the patient's lobes is vital.

## 2.2 Computer Tomography

In a CT scan, an X-ray source emitting narrow X-ray beams is rotated around the patient. A detector picks up the transmitted signal at the other side of the patient. When the source has done a full rotation, a cross-sectional image or *slice*, is computed. Then, the patient is moved forward to obtain a new slice. By repeating this process, and then stacking the resulting slices, a three dimensional image of the scan region can be obtained. This gives images of sections of the human body with high quality [16].

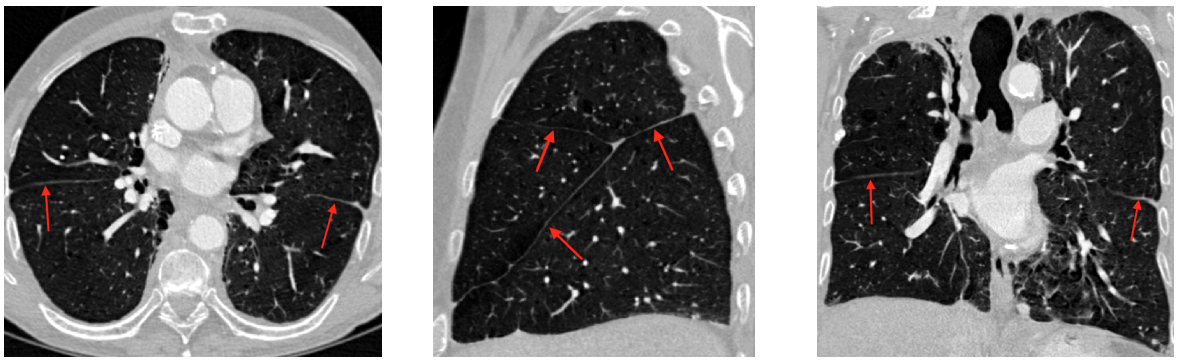
X-rays have different attenuation depending on the material they propagates through. The attenuation coefficient $\mu$ within the material is used during the reconstruction of the CT scan to produce a grayscale image. A linear transformation of the attenuation coefficient is done to calculate the corresponding HU, defined by (2.2.1), where $\mu_{water}$ and $\mu_{air}$ are the linear attenuation coefficients of distilled water and air. The HU is a measure of the radio density, and is used by radiologists to interpret CT images. Dense materials, such as bone, have positive HU and appear bright, while sparse materials, such as air, have negative values and appear dark [17].

$$HU = 1000 \times \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}} \tag{2.2.1}$$

In some cases, a radiocontrast agent is administered to the patient before the scan, to enhance the visibility of certain internal structures that would prove impossible to distinguish otherwise. The resulting images are known as contrast CT, and can also be useful to obtain functional information about tissues.

The lungs contain a lot of air, which is sparse. This makes the lungs appear dark in the CT images, as shown in Figure 2.2.1, with an intensity value between [-700, -600] HU. The contrast to the surrounding structures is high, except to the trachea and bronchi, which is also filled with air, and thus have intensity values close to that of the lungs. The fissures appear as light, blurred lines in the CT images and can be seen in Figure 2.2.1, where the red arrows are pointing at the fissures. The thickness of the line may vary, and in some cases, the fissure

is incomplete and not possible to see in a CT image.



**(a)** Axial **(b)** Sagittal **(c)** Coronal

**Figure 2.2.1:** CT slices of the lungs seen from different views. The red arrows point at the lung fissures.

## 2.3 Deep Learning

Machine learning utilizes algorithms and statistical modelling to enable computer systems to perform specific tasks without being explicitly programmed to do so. It is based on the idea that a system can learn from raw data by identifying patterns, and use this knowledge to make subjective decisions on its own. The performance of these machine learning algorithms are highly dependent on the representation of the data it is given. Each piece of information in the representation is known as a feature, and it is the choice of these features that affects the performance the most [18].

Different branches of machine learning exists, depending on the learning technique. In supervised learning, the model is given a set of training data $x$ with corresponding ground truths $y$. It is assumed that there exists an unknown mapping between the training data and its ground truth, given by $y = f(x)$. The goal of the model is to approximate this mapping as $\hat{y} = \hat{f}(x)$. The model use the training data to make a prediction $\hat{y}$, and compares this prediction to the ground truth $y$, as shown in Figure 2.3.1. Based on this comparison, some model parameters are updated to get the prediction closer to the ground truth [18].

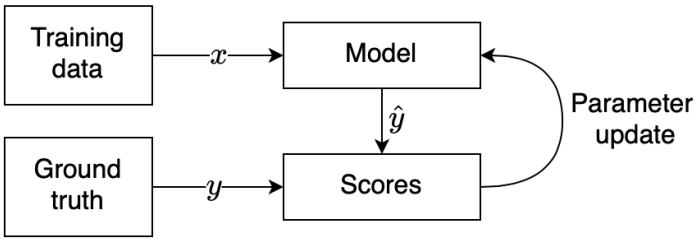



**Figure 2.3.1:** Flow chart illustrating the steps behind supervised learning.

Designing a feature extractor to transform the raw input data to suitable feature vectors, is

a difficult task that requires expertise and careful engineering [19]. Deep learning methods solves this problem by automatically recognizing the representation that best suit the task. Complex functions are learned by transforming the representations into simpler, more abstract functions. This process is usually repeated several times, resulting in multiple levels of different abstract representations.

### 2.3.1 Feed-forward Neural Networks

The most basic example of a deep learning model is the feed-forward neural network, also known as a multilayer perceptron (MLP). Feed-forward neural networks consists of an input layer with known inputs, one or more hidden layers, and an output layer where the outputs can be observed. These layers, with exception of the input layer, are built by so-called units or neurons, working in parallel, as illustrated in Figure 2.3.2.

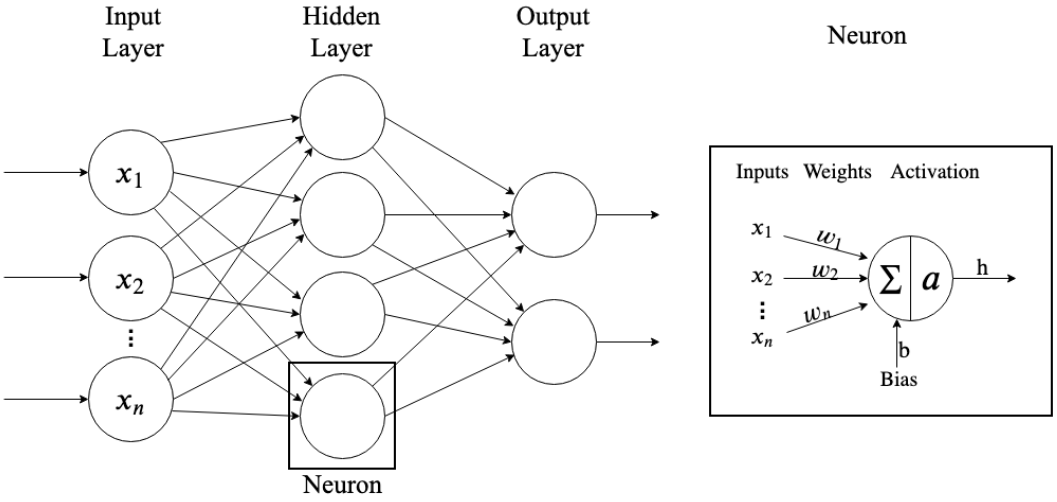



**Figure 2.3.2:** A feed-forward neural network with one hidden layer. One of the neurons in the hidden layers are highlighted to show its function in the network.

Each neuron receives inputs $\boldsymbol{x}$ from the neurons in the previous layer, and adds a weight $\boldsymbol{w}$ to these values. The weighted inputs are summed with a potential bias $b$, before an activation function $a$ is applied to make a non-linear mapping. The output $h$ of a neuron, is then given by (2.3.1), where $x_i$ and $w_i$ denotes the input and weight of the $i$-th neuron in the previous layer [20]. An illustration of the neuron is also shown in Figure 2.3.2.

$$h = a\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.3.1}$$

In theory, it is possible to represent any function with a simple feed-forward network consisting of only one hidden layer with a finite number of neurons. For many cases, this would in practice mean an infeasibly large layer that is not able to learn and generalize properly. Instead, many

simpler layers are used to represent the function. Neural networks with many hidden layers all called *deep*, and are better at generalizing [21].

For networks with many layers, the rectified linear unit (ReLU) is commonly used as the activation function, as it typically learns faster than most other activation functions [19]. The function of this *artificial* neuron resembles how a neuron behaves in the human neural system, hence the name *neural* networks [17].

### 2.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are commonly used for data represented by multiple arrays, such as time series and images. In these types of data, neighbouring values are often highly correlated, and they often contain features that are invariant to the location in the input data [19]. This makes CNNs a popular choice for most computer vision applications [22].

CNNs use convolutional operations to extract features from the given input. In image processing, 3D convolutions are used to filter a 3D image with a fixed 3D filter, known as a *kernel*. Depending on its filter coefficients, the kernel can extract different features such as edges and corners. The filtered image is produced by sliding the kernel with a certain stride value over the input image, where the sum of the products is calculated. The result is a filtered image or *feature map*, typically of reduced size, as shown in Figure 2.3.3. This allows the network to be deeper with fewer parameters. For an input image of size $L \times M \times N$ and a kernel of size $K \times K \times K$, the filtered image will be of size $L-(K-1) \times M-(K-1) \times N-(K-1)$. The kernel does not have to be cubic, but for this thesis we assume that it is, for simplicity.

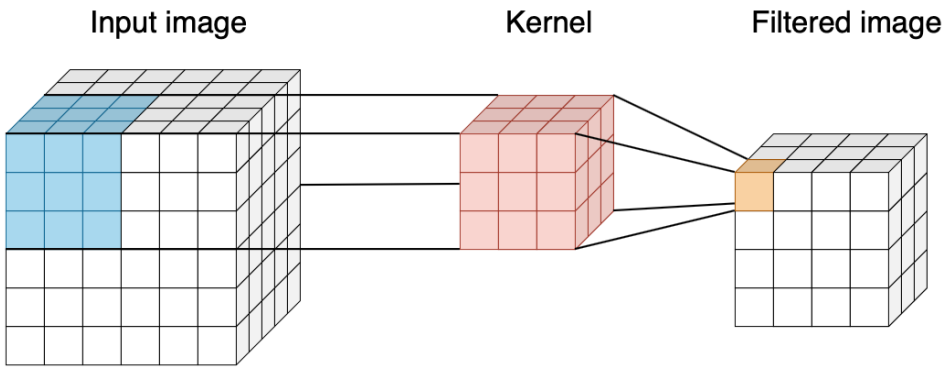
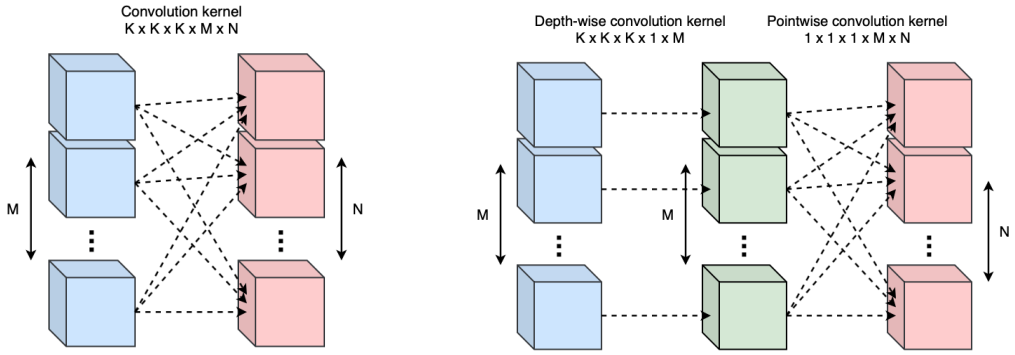



**Figure 2.3.3:** Filtering of a 3D input image of size 6x6x6 with a kernel of size 3x3x3. The result is a filtered image of size 4x4x4, assuming no padding is applied at the edges.

The computational cost of convolutional layers is high. To extract all relevant features from the input, one requires several kernels at each convolutional layer, resulting in a stack of filtered images from each of the different kernels to represent the feature map. A regular convolution can be factorised into a depthwise and a pointwise operation. Together, these two steps makes a so called depthwise separable convolution (DS convolution). While a regular convolution calculates both spatial and cross-channel correlations simultaneously, the DS convolution factors these operations into two simpler calculations. First, a spatial convolution is performed

over each of the input channels, before a pointwise convolution performs a $1 \times 1 \times 1$ convolution to combine the depthwise convolution outputs and project them onto a new channel space [12].

While a regular 3D convolution use a 5D convolution kernel tensor $W \in \mathbb{R}^{K \times K \times K \times M \times N}$, the DS convolutions factorise this kernel into a depthwise kernel tensor $D \in \mathbb{R}^{K \times K \times K \times M}$ and a pointwise kernel $P \in \mathbb{R}^{M \times N}$. This is illustrated in Figure 2.3.4, where K is the size of the cube shaped kernel, M is the number of input channels, N number of output channels, and the blocks represent the feature maps. By factoring the operations of a regular convolution into simpler steps, the number of parameters and the computational cost of the necessary calculations will decrease. This is beneficial as one of the main challenges with deep learning is the computational cost, especially when dealing with large data such as 3D data.



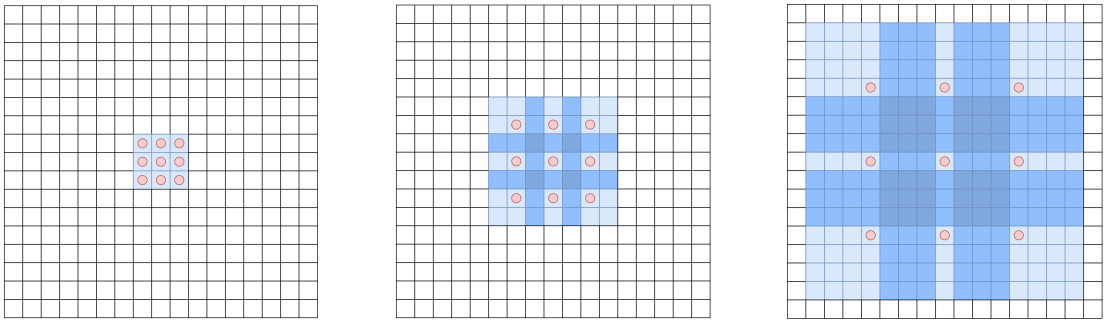

**(a)** Regular convolution. **(b)** DS convolution.

**Figure 2.3.4:** Comparison of the different operations for a regular convolution and a depthwise separable convolution.

The range of context that can be seen in each convolutional layer is known as the receptive field. To capture both low-level and high-level features of the input image, several convolutional layers in series are commonly used. The first convolutional layers captures the low-level features, such as color, edges and corners, with small receptive fields. The receptive field then increases with the downsampled input, resulting in high-level features. This way of increasing the receptive field may however result in loss of details due to the frequent downsampling. Another way to extract global context is by increasing the size of the kernel, but this comes at a memory and computational cost. A third option is to use so called dilated convolution, where the receptive field grows faster than the number of parameters.

In a dilated convolution, the kernels are upsampled (or dilated) by adding zeros between the kernel values. The number of zeros between each value is called the dilation rate $r$, and a regular convolution is simply a dilated convolution with $r = 1$. Figure 2.3.5 shows how the receptive field is expanded with the dilation rate, while the number of parameters at each convolutional layer stays the same. The receptive field grows exponentially with each layer, while the number of parameters grows linearly. This makes it an effective way of expanding the receptive field, without increasing the number of parameters significantly.

A downsampling operation known as *pooling*, is often performed between every few convolutional layers, and results in fewer parameters in the network. This helps reduce the number of feature-map coefficients to process, as well as making successive convolution layers look at



**(a)** $r = 1$ **(b)** $r = 2$ **(c)** $r = 4$

**Figure 2.3.5:** Illustration of how the receptive field of a convolution is expanded with the dilation rate $r$ for a $3 \times 3$ kernel. The red dots illustrates the kernel values of the upsampled kernel. The blue fields shows the range of the receptive field, where the different shades indicates the overlap between between the kernel values as they are swept over the data. (a) $r = 1$ gives a receptive field of $3 \times 3$. (b) $r = 2$ gives a receptive filed of $7 \times 7$. (c) $r = 4$ gives a receptive field of $15 \times 15$.

increasingly larger windows to induce spatial-filter hierarchies [22]. The pooling layer takes a group, or a *pool*, of neighboring values in the feature map and outputs only one value. In deep learning contexts, this value is typically the average or maximum of the pool. Figure 2.3.6 illustrates the process of 2D max pooling.

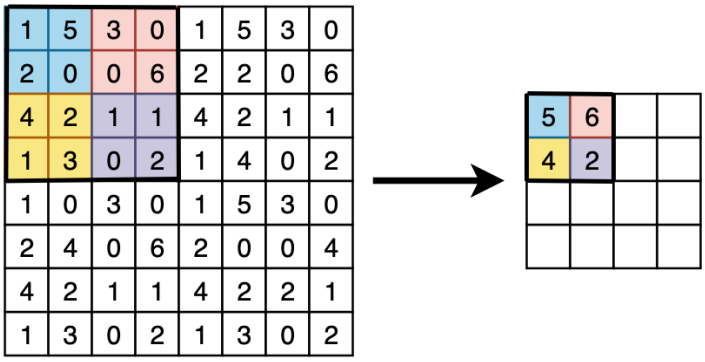



**Figure 2.3.6:** 2D max pooling with a pool size of 2x2.

### 2.3.3 Training Deep Neural Networks

During training of a neural network, the desired response of every neuron in the output layer is known. The desired outputs of the hidden neurons however, are unknown. A learning algorithm decides how the weights and bias of the hidden layers should be adjusted to minimize the error between the actual output and the desired output with the help of a *cost function* [18]. The learning algorithm uses a gradient vector to see how adjusting the weights will result in an increase or decrease in this cost function. The weight vector is then adjusted in the opposite direction. This is known as *backpropagation*. The most common form of doing this, is by drawing a batch of random samples from the training set to feed to the network. This process is known as stochastic gradient descent (SGD) [19].

An issue that often occurs during training is *overfitting* [20], where the model performs significantly better on the training set than on the validation and test set. This means that the network is failing in generalizing the information from the training data. One way of reducing the risk of overfitting is by adding more data to the training set, or reducing the complexity of the model. In cases where this is not possible, adding regularization to the network may be useful.

One of the most frequently used regularization techniques is *dropout* [23]. During training, this technique randomly drops nodes and their connections in the network, given a probability. This prevent the network from adapting to much to the given training set, as it forces the network to use a wider range of neurons. *Batch normalization* [24] is a technique where the unit values in the hidden layers are normalized. This speeds up training and allows for a higher learning rate, in addition to acting as a regularizer because it adds some noise to each hidden layers activations.

Another useful regularization technique is *data augmentation*. This is a way of adding more variability to the data set, by adding different transforms to the data. For images, this could be different degrees of flipping, rotation, scaling, color transformations, etc. These transforms are usually assigned a probability of how likely they are to be added to an image, so that different combinations of the transforms are added to the different samples.

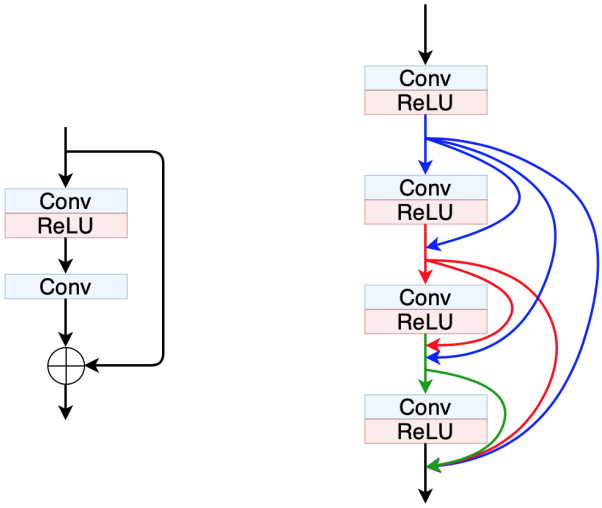
### 2.3.4 Residual Learning and Dense Connections

For deep neural networks with many layers, the gradient of the error may become vanishingly small during backpropagation. When the gradient of the error is propagated to the shallower layers, the gradient at each shallow layer becomes smaller and smaller, until it vanishes completely. This is known as the vanishing gradient problem, and stops the network from training properly when gradient-based learning methods are being used.

Batch normalization and ReLUs as activation functions are used to deal with the vanishing gradients problem, but for deep networks, this might not be enough. Another way of dealing with this problem, is by using deep residual learning, a method introduced by Kaiming He et al. in their ResNet [25]. The method uses *identity shortcut connections*, which promotes gradient propagation by connecting the output of one layer to the input of another layer, while skipping one or more layers inbetween. Element-wise addition is used to connect the layers, as shown in Figure 2.3.7a. The impact of the vanishing gradient problem is reduced as the activations from previous layers are reused.

If a block of layers is connected in a way where each layer receives feature maps directly from all preceding layers, the block is said to be densely connected. Gao Huang et al. used a dense block like that in their well known DenseNet [26], which proved to achieve higher accuracy with fewer parameters compared to the ResNet of Kaiming He et al. Dense connections use channel wise concatenation instead of element-wise addition, as shown in Figure 2.3.7b.

Due to the direct connections to different layers, dense networks benefits from a strong gradient flow, where the error can be easily propagated to the early layers. This further helps with the vanishing gradient problem. Dense networks also makes it possible for the classifier to




**(a)** Residual learning **(b)** Dense connection

**Figure 2.3.7:** Illustrations of residual learning and dense connections. Residual learning typically use element-wise addition, while dense connections use concatenation.

utilize features of different complexity levels, and not only the most complex (high level) features. This is especially useful when training data is sparse. In addition, receiving inputs from previous layers will result in more diversified features.

### 2.3.5 Deep Learning Libraries

For implementing deep neural networks, there are several frameworks to choose from. TensorFlow [27] and PyTorch [28] are two popular choices, both open-source. TensorFlow is based on Theano, and is developed by Google Brain, while PyTorch is based on Torch, and is developed by Facebook, Inc.

Both TensorFlow and PyTorch operate on tensors, and view the models as directed acyclic graphs (DAGs). The main difference between the two frameworks is that TensorFlow use a static computation graph while PyTorch use a dynamic computation graph. For static graph frameworks, the graph is only defined once, which means that it cannot be changed on the go. With dynamic graph frameworks however, a different graph is constructed from scratch for every training sample. This makes it possible to define and change nodes for every training instance, which in turn makes it more suitable for variable input sizes.

Normally, the training of neural networks is done with 32-bit floating point type, so called *full precision training*. By reducing the floating point to 16-bit, so called *half precision training* can be achieved. This results in benefits like computational speed up due to faster math operations with the reduced data, and a reduction in required memory, which may be very useful for larger neural networks. Using half precision may come at the expense of numeric instability and lower accuracy. By using 32-bit floating point for precision-sensitive components such as optimizers and batch normalization layers while using 16-bit everywhere else, one gets the benefits of memory reduction and speed up, while keeping the numeric stability and accuracy

from training with full precision. This is known as *mixed precision training*, as a combination of different precision are being used.

# 3 | Materials and Methods

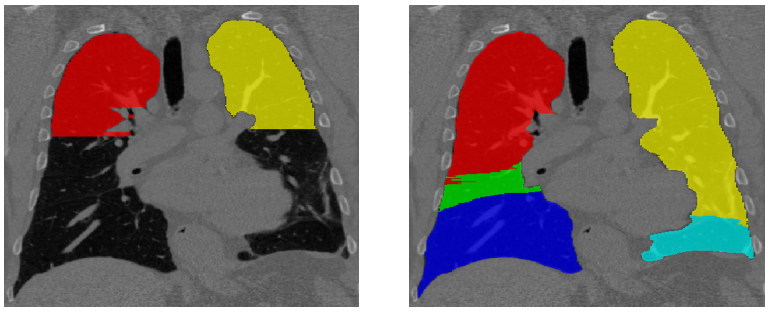
## 3.1 Data

For this thesis, a total of 176 CT volumes were collected from four different sources: 90 CTs were obtained from the publicly available database *DeepLesion* [29] from the National Institutes of Health (NIH), 20 from the *VESsel SEgmentation in the Lung 2012* challenge (VESSEL12) [30], 51 from the *LUng Nodule Analysis 2016* challenge (LUNA16) [31] and the remaining 15 from St. Olavs Hospital (STO) in Trondheim (Norway), previously used in [32, 33].

The CT volumes from LUNA16, VESSEL12 and STO were accompanied by corresponding ground truths of the lungs. In addition, lung segmentations for the NIH data were produced by applying an already existing lung segmentation model to the CT volumes [34].

The LUNA16 data was also obtained together with ground truths of the lobes. For the volumes from VESSEL12, lobe segmentations were obtained by applying a lobe segmentation model that was already trained on the VESSEL12 data. The result was a data set with 176 CTs with lung annotations and 71 CTs with lobe annotations.

During the study, the lobe annotations for one patient from the VESSEL12 data set was found to be incomplete. However, the models produced in the experiments for this thesis were already trained using this data at the point of discovery. The correct lobe annotations were produced and used for the validation studies. Figure 3.1.1 shows the incomplete segmentation and the corrected segmentation.



**(a)** Incomplete lobe segmentation. **(b)** Corrected lobe segmentation.

**Figure 3.1.1:** The incomplete and the corrected lobe annotation for one CT from our data set.

In our data set, we have a mix of 86 chest CT volumes and 90 full body CT volumes, with scan regions as shown in Figure 3.1.2. The dimensions of the CT volumes in the x, y and z dimension cover $[487; 512] \times [441; 512] \times [56; 854]$voxels, where the size of each voxel ranges from $[0.52; 0.98] \times [0.52; 0.98] \times [0.5; 5.0]\, mm^3$ along each axis.

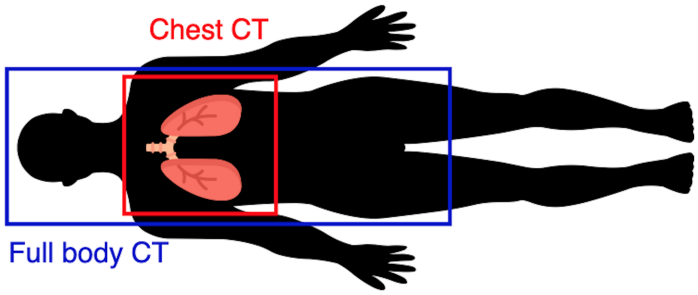



**Figure 3.1.2:** The red box indicates the scan region for a chest CT, while the blue box indicates the scan region for a full body CT.

## 3.2 Method

In this section, the basic principles behind the U-Net and the PLS-Net architecture will be introduced. Following the introduction to each of the networks, is a section describing some architecture and implementation choices made for this study, together with the training strategy. The implementation details used to perform the experiments are then summarized, before the final validation study is presented.

### 3.2.1 U-Net Architecture

The 3D U-Net proposed by Çiçek et al. [11], is a modified version of the original 2D U-Net introduced by Ronneberger et al. [35]. An example of the network architecture and its distinctive U-shape, can be seen in Figure 3.2.1. The U-Net is divided into a symmetric contracting and expanding path, hereby referred to as the encoder and the decoder. While the encoder aims at capturing the context of the input data, the decoder aims at achieving precise localization of the segmentation mask.

The U-Net consist of multiple resolution levels, where a set of operations is repeated at every level. An important building block in the network is the convolution block, that includes a 3D convolution operator with kernel size $3 \times 3 \times 3$, followed by an activation function. In the encoder, the convolution block is applied twice at every resolution level, to increase the number feature channels from the input. The feature maps are then downsampled by a $2 \times 2 \times 2$ max pooling with stride two in each direction. No max pooling is done in the last layer.

At the decoder, a simple upsampling of the feature maps with factor two is performed for every resolution level. A copy from the feature maps at the encoder is concatenated with the upsampled feature maps, to regain the high resolution details. The convolution block used at the encoder is then applied twice to reduce the number of feature channels, before the same process is repeated at the next resolution level. At the output, a $1 \times 1 \times 1$ convolution followed by a softmax operation is applied to reduce the number of output channels to the desired number of classes, $C$. An overview of the network can be seen in Figure 3.2.1.

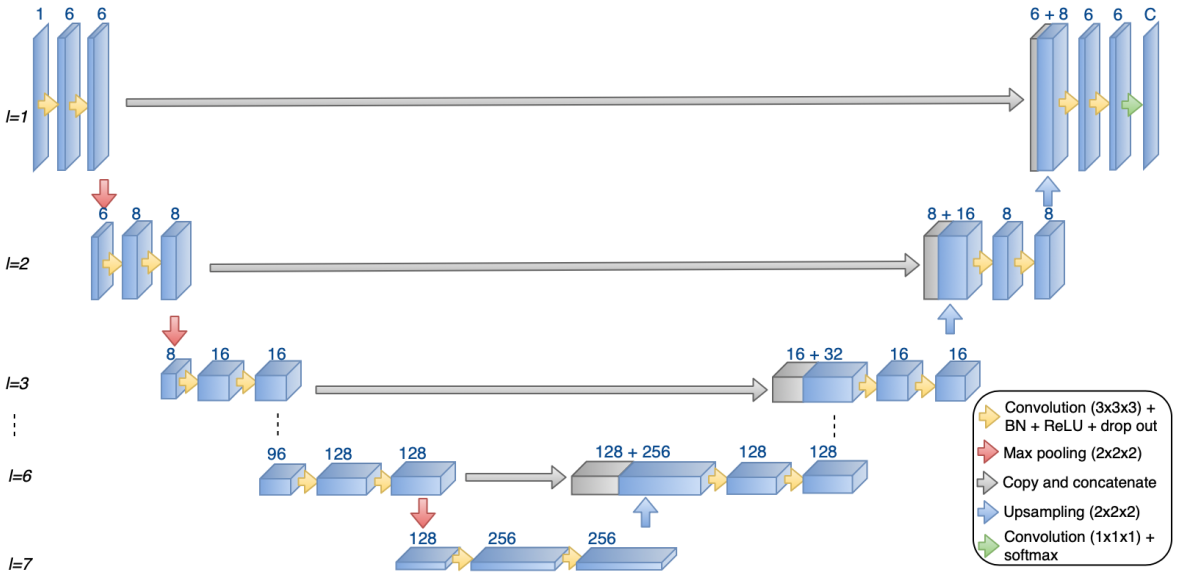



**Figure 3.2.1:** Overview of the U-net architecture. The blue boxes illustrate the feature maps at each resolution level. The numbers above the box indicate the number of channels in the feature maps.

### 3.2.2 Training Strategy for the U-Net

In this thesis, the U-Net was implemented with seven resolution levels, $l$, to ensure that both high-level and low-level features from the input data were captured. The number of kernels applied during the convolution at each resolution level was $\{6, 8, 16, 32, 96, 128, 256\}$, from the lowest to the highest level. To speed up the training, a batch normalization layer was applied after each convolutional layer, followed by a spatial dropout of 0.2 for regularization. ReLU was chosen as the activation function for the hidden layers, as it has proven to work well for networks with many layers [19]. In the final layer, softmax was used as the activation function to output a vector representing the probability of each class.

The U-Net was trained using the Adam optimizer [36] with a learning rate of $10^{-3}$. For the loss function, the soft Dice coefficient (DC) presented in (3.2.1) was used, where $y$ is the ground truth, $\hat{y}$ is the prediction and $v$ is the number of voxels in the volume. The coefficient is calculated for all classes, and averaged. To formulate a function that can be minimized, the final loss is then given by $1 - DC$.

$$DC(y, \hat{y}) = 1 - \frac{2\sum_v y\hat{y} + 1}{\sum_v y^2 + \sum_v \hat{y}^2 + 1} \tag{3.2.1}$$

Training was either done with batch size 1 on samples of size $256 \times 256 \times 256$, or batch size 2 on samples of size $192 \times 192 \times 192$. The U-Net was only trained for the lung segmentation task, with $C = 2$. Training was done from scratch, and stopped after 20 epochs with no improvement in the validation loss. The model from the epoch with lowest validation loss was saved and evaluated.

### 3.2.3 PLS Architecture

The second network architecture investigated in this work is the Pulmonary Lobe Segmentation Network (PLS-Net), as introduced by Lee et al. [12]. The PLS-Net is a 3D fully convolutional network, with an asymmetric encoder-decoder structure. The network aims at leveraging information from the whole input volume at once, while keeping the number of network parameters low.

The PLS-Net was designed to exploit the spatial and contextual information in high-resolution CT volumes efficiently. To achieve this, Lee et al. introduced the dilated residual dense block (DRDB) as an important building block in their network. To extract both local and global contexts from the high resolution input volume, a series of four $3 \times 3 \times 3$ depthwise separable convolutional layers with different dilation rates were implemented in the DRDB-block. This ensures a large diversity of the receptive field. An illustration of the operations and the resulting feature maps of the DRDB-block is shown in Figure 3.2.2.

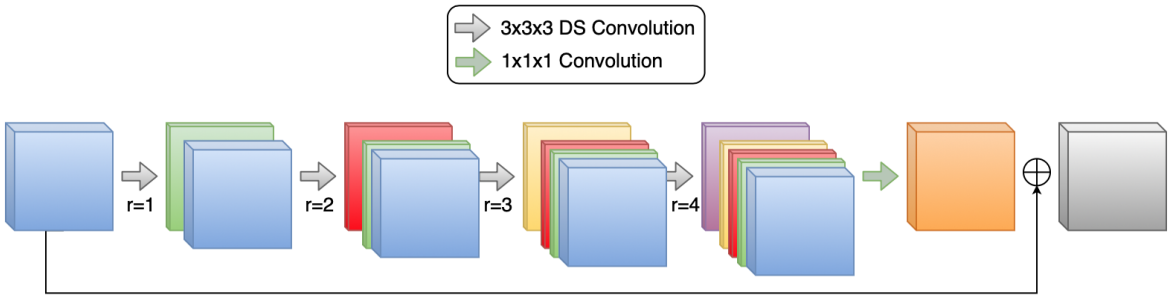



**Figure 3.2.2:** Illustration of the DRDB-block. $r$ indicates the dilation rate of the convolution, and the blocks represent the feature maps.

When using dilated convolutions, gridding artefacts may occur, as the input of the convolutions is sampled in a checkerboard manner. When several convolutional layers are following in cascade, these artefacts may be propagated to the consecutive layers, if the dilation rates have a common factor relationship. Thus, Lee et al. set the dilation rates of the four consecutive convolutions to $r = \{1, 2, 3, 4\}$. This series of dilated convolutions exponentially enlarges the receptive field with each following layer.

To ensure multi-scale context, the cascade of dilated convolutions in the PLS-Net was implemented with dense connections and residual learning. The four dilated convolutions were all densely connected through concatenations, to receive feature maps from the previous layers. Each dilated convolution layer adds 12 new features to the cumulative feature maps from the previous layers. A skip connection from the input of the block was then added to the final concatenated features, after a $1 \times 1 \times 1$ convolution to ensure compatible shapes, for residual learning.

The DRDB-block requires a lot of memory during training. After the forward pass, all intermediate activations for each layer are usually stored in memory to compute the gradients in the backwards pass. These intermediate activations require a lot of memory to store, but are cheap to compute. By recomputing all intermediate activations in the backward pass

instead of storing them, a reduction in required memory can be traded for a small increase in training time. This memory efficient implementation of the DRDB-block changes the memory consumption for the feature maps from quadratic to linear, with the network depth.

Lee et al. implemented their network with depthwise separable convolutional layers, to obtain feature maps with a reduced number of parameters and computational cost. The DS convolution was followed by batch normalization to help speed up training. The encoder aims at representing the input by feature maps. Each resolution level $l$ in the encoder included a $3 \times 3 \times 3$ DS convolution which downsamples the input with a stride of 2. To bring back some of the spatial information lost during the convolutional and downsampling operations, Lee et al. added input reinforcement by concatenating the feature maps from the DS convolutional layer with the a downsampled version of the input image. The image was downsampled with a factor of $2^l$, using trilinear interpolation. The concatenation was further followed by a series of $2^{l-1}$ DRDB-blocks. An overview of the network is shown in Figure 3.2.3.

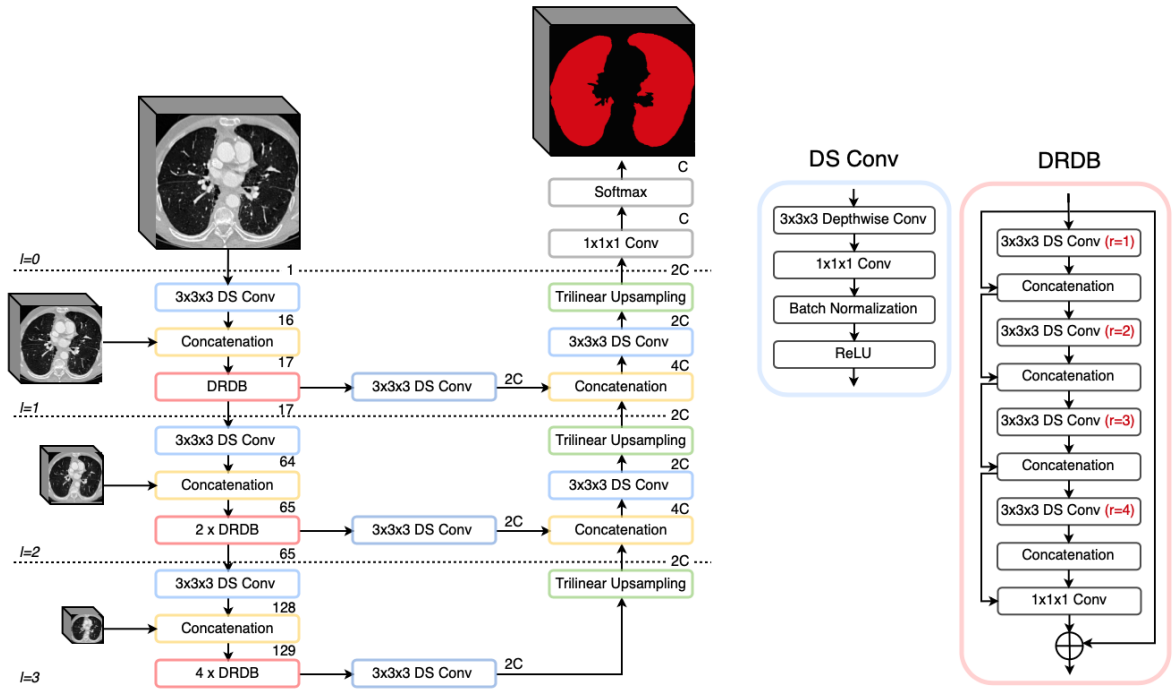



**Figure 3.2.3:** Overview of the PLS-Net architecture. The different resolution levels are marked with $l$, and the number above each block in the network indicates the number of feature channels at that point. $C$ is the number of desired classes and $r$ is the dilation rate.

Due to the downsampling operations at the encoder, the resolution is gradually reduced to 1/8 of the input resolution. At the decoder, the feature maps were upsampled to the original input size. A simple upsampling operation will not be able to regain the high resolution details of the input, and the result would be a coarse segmentation mask. To avoid this, a decoder with convolutional layers and concatenation with feature maps from the encoder, were used together with trilinear upsampling with factor 2.

$3 \times 3 \times 3$ DS convolutional layers were used to generate $2C$ feature maps, where $C$ is the

number of desired classes. The upsampled feature maps were concatenated with the $2C$ feature maps from the DRDB-block at the corresponding resolution block. This way, the high resolution features from the encoder were obtained at the decoder. When the feature maps were back to the original input size, a $1 \times 1 \times 1$ convolution followed by a softmax operation was applied to produce the $C$ probabilities representing each class. An illustration of the final PLS-Net architecture is shown in Figure 3.2.3.

### 3.2.4 Training Strategy for the PLS-Net

In this thesis, the PLS-Net was implemented with three resolution levels, where the number of feature channels produced at each resolution level was {16, 64, 128}, same as for the network of Lee et al. The number of resolution levels can be lower for the PLS-Net compared to the U-Net, because of the large receptive field of the DRDB-block. Number of output channels, $C$, was 2 for the lung segmentation task, and 6 for the lobe segmentation task.

Same as for the U-Net, the PLS-Net was trained using the Adam optimizer with a learning rate of $10^{-3}$, the Dice loss function introduced in (3.2.1), ReLU as the activation function for the hidden layers and softmax for the output layer. The network was trained with a batch size of two, once for samples with a fixed size of $256 \times 256 \times 256$ for lung segmentation, and once for variable size of $256 \times 192 \times [181; 382]$ for lobe segmentation. Because of the variable sizes of the lobe samples, all samples in one batch were padded to the same size before being fed to the network. Training was done from scratch, and stopped after 20 epochs with no improvement in the validation loss. The model from the epoch with lowest validation loss was saved and evaluated.

### 3.2.5 Implementation Details

All original CT volumes were pre-processed to obtain standardized samples for training, starting with resampling to a uniform spacing of 1mm. Resizing of the data was then performed to reduce the required memory during training. The intensity range was clipped to $[-1000, 1000]\,HU$ and normalized to zero mean and unit variance. To avoid loosing too much details of the lobes and its fissures, the data used for lobe segmentation was cropped before being resized. All slices that did not contain any part of a lung were dropped.

A set of data augmentation transforms was applied at random during training for both networks: rotation between [-20°, 20°] and translate between [-5%, 5%]. For inference, the test samples were resampled and resized to match the training data used to obtain that specific model. The predicted segmentation mask was then resampled and resized back to the original size, to perform evaluation of the models accuracy.

The implementation was done in Python 3.6.9, using either Keras v.2.3.1 with TensorFlow v.1.15.0 backend, or PyTorch-lighting v.0.7.2 with PyTorch v.1.3.1 as backend. The models were trained on a remote Ubuntu 18.04.4 server with Intel Core i9-9900K CPU @3.60GHz and GeForce RTX 2080 with 11GB RAM.

### 3.2.6 Validation Studies

Four studies were conducted for this thesis: (i) a lung segmentation study with focus on choice of framework and training precision, (ii) an evaluation of the PLS-Net for lung segmentation, (iii) a lobe segmentation study, and (iv) evaluation of the lobe segmentation model on lungs with abnormalities.

**The lung segmentation study:** the U-Net presented in Section 3.2.1 was used to train five models with different combinations of libraries, batch size and input resolution, and training precision. Details of each model are presented in Table 3.2.1. The data set containing the 176 CT volumes and corresponding lung annotations, was divided into fixed training, validation and test sets, with the ratio 70-15-15, respectively.

**Table 3.2.1:** Lung segmentation study: overall U-Net parameters description where bs stands for batch size.

| Model name | Network | Library | Precision | Input size | bs |
|---|---|---|---|---|---|
| Model 1 | U-Net | TensorFlow | Full | 256x256x256 | 1 |
| Model 2 | U-Net | PyTorch | Full | 256x256x256 | 1 |
| Model 3 | U-Net | TensorFlow | Full | 192x192x192 | 2 |
| Model 4 | U-Net | PyTorch | Full | 192x192x192 | 2 |
| Model 5 | U-Net | PyTorch | Mixed | 192x192x192 | 2 |

**PLS-Net for lung segmentation study:** the PLS-Net introduced in Section 3.2.3 was trained for the lung segmentation task, using PyTorch and mixed precision and compared to U-Net performances. In addition, the impact of using a memory efficient DRDB-block was studied.

**The lobe segmentation task:** the PLS-Net was trained using the 71 CT volumes with corresponding lobe annotations. The network was trained using PyTorch and mixed precision, and to ensure that the model generalizes to independent data, a five-fold cross-validation regime was used for the limited data sample. The performances of the lobe models were evaluated for each of the five lobes independently, as well as in an overall manner. Post-processing using the predicted lungs mask was also attempted to improve the lobes segmentation quality by removing any potential excess labeling outside of the lungs.

**Lobes segmentation on heavily pathological lungs:** the lobe segmentation model from the third study was applied to some CT volumes containing heavier abnormalities (e.g., extreme emphysema, partially collapsed lung). These CT volumes were selected from the NIH data set used for training the lung segmentation models, and did not have any corresponding ground truths of the lungs. The validation is purely qualitative and based on visual observations.

Three different measures were used to assess the performances of the different trained networks are: the Dice score for segmentation quality (in %), the duration to compare training and inference speed (in s), and the GPU memory consumption (in GB). In this thesis, the memory consumption is measured by looking at how much GPU memory the network use during training of the model and inference. The training speed is a measure of how long it takes for the whole data set to pass through the network, and will be given as seconds per epoch. Inference time is the time it takes for a trained network to make a prediction. An average of the time it takes to make a prediction on each test sample will be given to assess the inference speed. The Dice score, detailed in Equation (3.2.2), is a measure of the similarity of the ground truth $X$ and the segmentation mask predicted by the model $Y$.

$$\mathrm{DSC} = \frac{2|X \cap Y|}{|X| + |Y|} \tag{3.2.2}$$

In addition to the three measurements mentioned above, visual inspection will be performed by the author, to further assess the accuracy performance of the models.

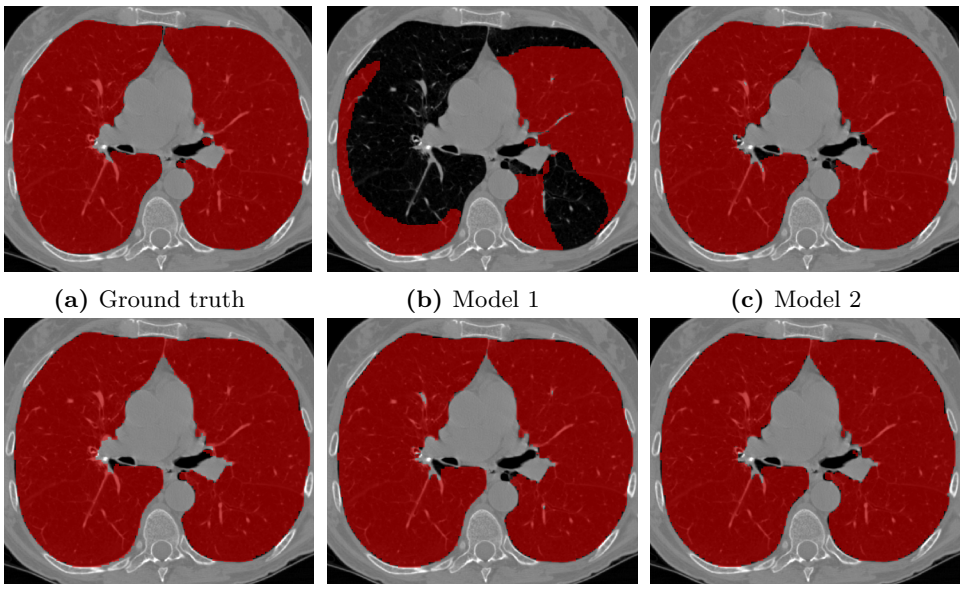
# 4 | Results

## 4.1 Lung Segmentation Study

The results from the experiment are summarized in Table 4.1.1, while details regarding each model are presented in Table 3.2.1.

**Table 4.1.1:** Comparison of the results from the lung segmentation study.

| | Training | | Inference | | Accuracy |
|---|---|---|---|---|---|
| | Memory (GB) | Speed (s/epoch) | Memory (GB) | Speed (ms/sample) | Dice (%) |
| Model 1 | 8.8 | 375 | 4.6 | 203 | 95.5 |
| Model 2 | 8.3 | 270 | 4.1 | 353 | 91.9 |
| Model 3 | 9.1 | 192 | 2.5 | 168 | 97.4 |
| Model 4 | 7.2 | 96.2 | 2.3 | 172 | 97.5 |
| Model 5 | 4.3 | 103.6 | 1.7 | 236 | 97.5 |

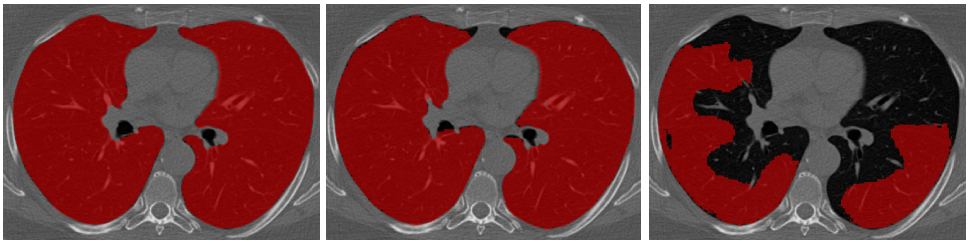


**(a)** Ground truth **(b)** Model 1 **(c)** Model 2

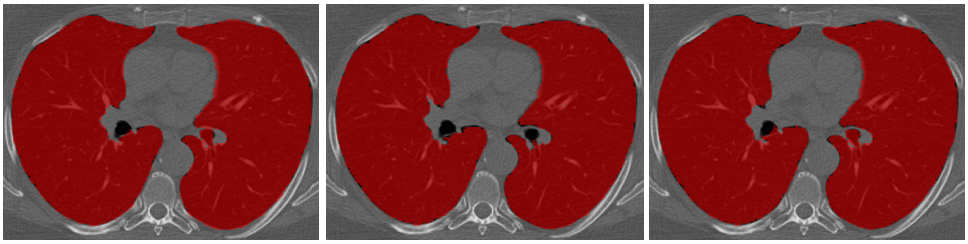
**(d)** Model 3 **(e)** Model 4 **(f)** Model 5

**Figure 4.1.1:** Predicted segmentation masks from all models on test sample A, together with the ground truth.

Figure 4.1.1 and 4.1.2 present two samples, A and B, of the predicted lung mask of all five models, together with the ground truth. Visual inspection of the models shows that the accuracy of Model 1 and Model 2 was very inconsistent. While they performed well on most of the CT volumes, there were some volumes where they struggled with both false positives and false negatives. Figure 4.1.1b and 4.1.2b shows the large variation in accuracy for Model 1 over two different samples. The same can be seen in Figure 4.1.1c and 4.1.2c for Model 2. These deviations were not found for Model 3, Model 4 or Model 5, which gave a more persistent performance.



**(a)** Ground truth **(b)** Model 1 **(c)** Model 2



**(d)** Model 3 **(e)** Model 4 **(f)** Model 5

**Figure 4.1.2:** Predicted segmentation masks from all models on test sample B, together with the ground truth.

## 4.2 Evaluation of the PLS-Net

Table 4.2.1 summarizes the results of training, inference and accuracy performance for the U-Net and the PLS-Net.

**Table 4.2.1:** Comparison of the U-Net and the PLS-Net. Both networks were trained with the PyTorch library, a batch size of 2, mixed precision and samples of size 256x256x256.

| Network | | Training | | Inference | | Accuracy |
|---|---|---|---|---|---|---|
| Network | # param. (M) | Memory (GB) | Speed (s/epoch) | Memory (GB) | Speed (ms/sample) | Dice (%) |
| U-Net | 6.6 | 8.8 | 247 | 2.7 | 482 | 97.7 |
| PLS-Net | 0.25 | 5.9 | 367 | 2.5 | 338 | 97.4 |

A comparison in training performances of the regular DRDB-block and the memory efficient DRDB-block is summarized in Table 4.2.2. The regular DRDB-block saves the intermediate

activations computed in the forward pass, while the memory efficient DRDB-block only stores the inputs and function parameter, and recomputes the rest during backwards pass.

**Table 4.2.2:** Comparison of speed and memory consumption for regular and memory efficient DRDB-block.

| | Memory (GB) | Speed (s/epoch) |
|---|---|---|
| Regular DRDB | 10.6 | 345 |
| Memory efficient DRDB | 5.9 | 367 |

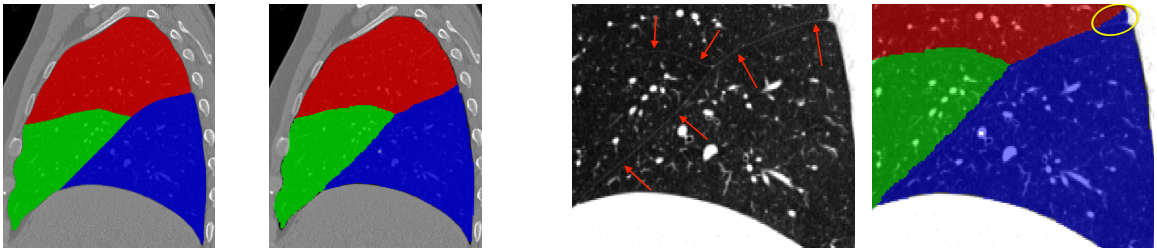
## 4.3 Lobe Segmentation Study

The results from the lobe segmentation task are presented in Table 4.3.1 together with the results obtained by Lee et al. in their PLS-Net experiments.

**Table 4.3.1:** Results from the lobe segmentation study. Dice scores are reported for each individual lobe and fold, as well as overall scores per lobe and epoch. The Dice scores obtained by Lee et al. are added for reference. Keep in mind that they used a different data set for training and testing, when comparing the results.

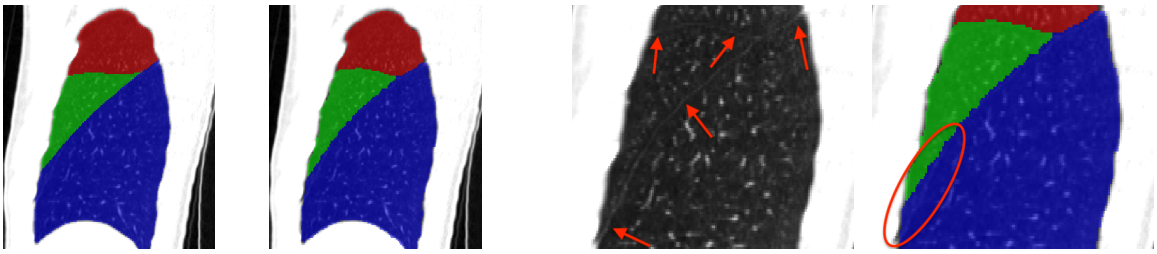
| | Dice score (%) | | | | | | |
|---|---|---|---|---|---|---|---|
| Fold | 1 | 2 | 3 | 4 | 5 | **Total** | Lee et al. |
| Right Superior | 92.6 | 91.5 | 92.1 | 94.0 | 89.8 | **92.0** | 96.2 |
| Right Middle | 86.5 | 81.7 | 85.5 | 87.9 | 82.6 | **84.8** | 93.6 |
| Right Lower | 96.6 | 92.2 | 93.8 | 95.8 | 94.5 | **94.6** | 96.3 |
| Left Superior | 97.3 | 95.8 | 95.8 | 96.3 | 96.1 | **96.3** | 96.8 |
| Left Lower | 96.8 | 95.6 | 94.5 | 95.8 | 94.1 | **95.4** | 96.1 |
| Overall | 94.0 | 91.4 | 92.3 | 94.0 | 91.4 | **92.6** | 95.8 |

Figure 4.3.1 and Figure 4.3.2 show slices from the right lung of two different test volumes, C and D. Figure 4.3.1a and 4.3.1b show a comparison of the ground truth and the predicted mask, while in Figure 4.3.1c and 4.3.1d, the same slice is shown when it is zoomed in on the borders of the lobes, where the red arrows are pointed at the fissures. The same goes for Figure 4.3.2. As can be seen from these figures, the model succeeds at locating the fissures, but has trouble following them closely to the borders of the lungs. This can also be seen in Figure 4.3.2d, where the gap between the true fissure and the predicted border of the right lower lobe (blue) and the right middle lobe (green), is marked with a circle. Figure 4.3.2a shows that the mask used as ground truth is also inaccurate around this fissure, and it can be argued that the predicted mask is more precise than the ground truth in this particular case.



**(a)** Ground truth **(b)** Prediction **(c)** Zoomed CT slice **(d)** Zoomed prediction

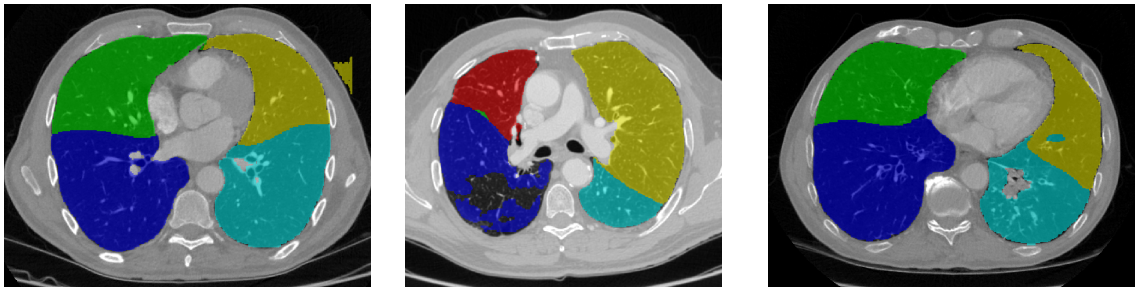
**Figure 4.3.1:** Ground truth and prediction for slice C for the lobe segmentation task.



**(a)** Ground truth **(b)** Prediction **(c)** Zoomed CT slice **(d)** Zoomed prediction

**Figure 4.3.2:** Ground truth and prediction for slice D for the lobe segmentation task.

The visual inspection of the predicted lobe masks also showed that on some volumes, voxels outside of the patients lungs were labeled as lobes, as seen in the upper right corner in Figure 4.3.3a. These errors occurred both outside of the patients' body, as well as inside, typically where air was found in the abdominal area. Figure 4.3.3b shows how the model in some cases failed at labeling all voxels inside the lungs as part of a lobe. For some volumes, the models also struggled with fragments of one lobe inside of another lobe, as seen in Figure 4.3.3c, where a patch of the left lower lobe (cyan) is found inside the left superior lobe (yellow).



**(a)** False positives. **(b)** False negatives. **(c)** Misplaced fragments.

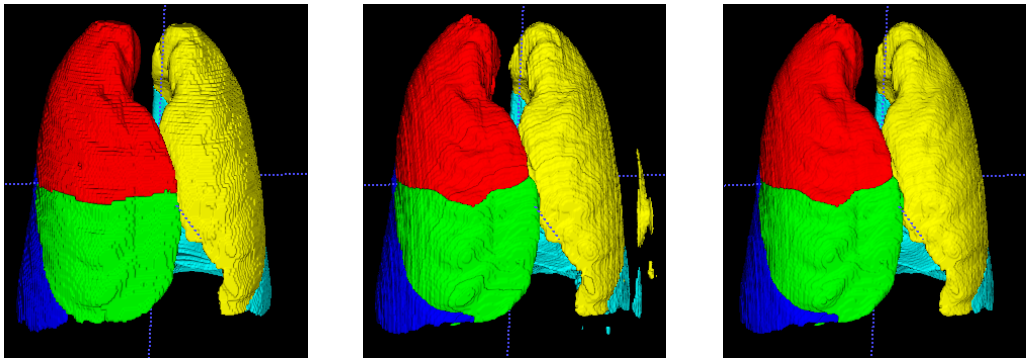
**Figure 4.3.3:** Examples of errors found in the predicted lobes masks, shown for three different samples.

The Dice scores obtained after applying the lung masks to the predicted lobe segmentations are presented in Table 4.3.2. The Dice scores displayed in the table are the average over all five folds.

**Table 4.3.2:** Comparison of Dice score before and after the post-processing step. The results are averaged over all five folds.

| | Dice score (%) | | |
|---|---|---|---|
| Fold | Raw prediction | **With lung mask** | Lee et al. |
| Right Superior | 92.0 | **92.2** | 96.2 |
| Right Middle | 84.8 | **85.0** | 93.6 |
| Right Lower | 94.6 | **94.9** | 96.3 |
| Left Superior | 96.3 | **96.5** | 96.8 |
| Left Lower | 95.4 | **95.9** | 96.1 |
| Overall | 92.6 | **92.9** | 95.8 |

Figure 4.3.4 shows a visual representation on the effect of applying the lung mask to a predicted lobe segmentation mask, containing a lot of redundant labeling outside of the patients body. The ground truth is shown in Figure 4.3.4a, with the predicted lobe mask in Figure 4.3.4b and the final result in Figure 4.3.4c.

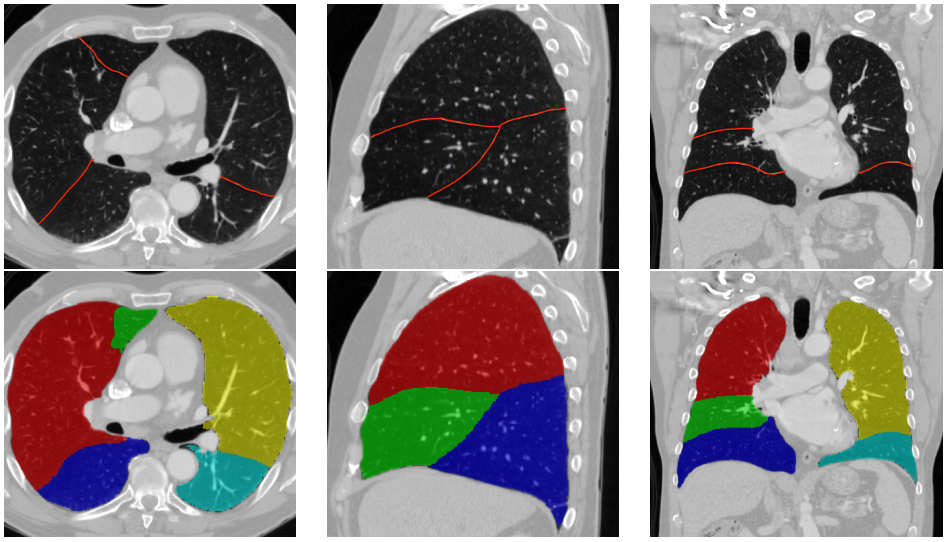


**(a)** Ground truth **(b)** Raw prediction **(c)** Post lung mask

**Figure 4.3.4:** Comparison of the predicted lobe mask before and after post processing with the lung mask.

## 4.4 Lobe Segmentation on Lungs with Abnormalities

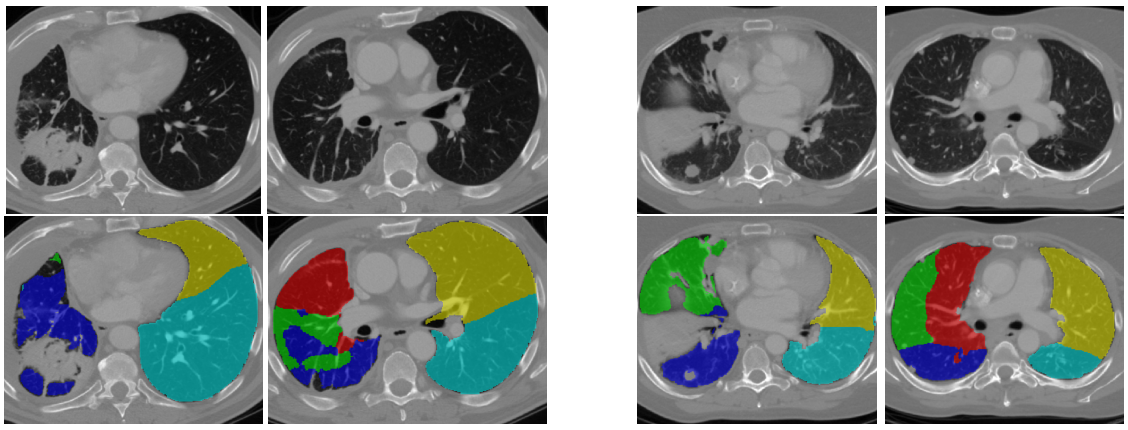
Figure 4.4.1 shows the resulting lobe segmentation mask when the model was applied to a random CT volume from the NIH data set, which did not contain any abnormalities. As we did not have any ground truths for this data set, an attempt of highlighting the fissure was done by the author, and can be seen as red lines in the slices displayed in the figure.



**(a)** Axial view. **(b)** Sagittal view. **(c)** Coronal view.

**Figure 4.4.1:** Predicted lobe segmentation mask for a sample from a different data-set than the one used for training. The red lines are the authors educated guess of where the true fissures are located.

Figure 4.4.2 shows how the lobe segmentation model from fold 1 in Section 4.3 performed on two different CT volumes, E and F, from the NIH data set, containing abnormalities. Two slices, 1 and 2, are shown for each of the volumes, where slice 1 shows a part of the volume where an abnormality is present, while slice 2 shows a part where the abnormality is not present.



**(a)** Sample E, slice 1. **(b)** Sample E, slice 2. **(c)** Sample F, slice 1. **(d)** Sample F, slice 2.

**Figure 4.4.2:** Predicted lobe segmentation mask for two different CT volumes from the NIH data-set containing abnormalities.

# 5 | Discussion

## 5.1 Lung Segmentation Study

The results in Table 4.1.1 show that the models trained with batch size 1 gave a lower accuracy performance than the models trained with batch size 2. The models using batch size 1 were trained and predicted on larger volumes with higher resolution. As these volumes contain more information, one could expect the models to perform better in this case. In addition, the checkerboard effects associated with upsampling the predicted mask back to the original size should be reduced for the larger volumes. This should in theory give a higher Dice score compared to using smaller volumes with lower resolution for training and prediction. When using batch size 1, the weights are updated after every single sample, which may give noisy training. As a result, the training lasts longer and reaching the optimum could be difficult. The lungs are relatively large, and with high contrast to most of the surrounding structures. Thus, using larger volumes during training might not be crucial for the lung segmentation task.

Visual inspection also shows that the models trained with batch size 1 gave very varying results over the different test volumes, as seen in Figure 4.1.1b and 4.1.2b, and Figure 4.1.1c and 4.1.2c. It is also interesting that there seems to be no correlation between which of the test volumes the two models performed well and bad on. While Model 1 predicted bad on the test volume displayed in Figure 4.1.1b, Model 2 performed well on this volume, as seen in Figure 4.1.1c. This is surprising as the two models are trained on the same data, using the same network. However, due to randomness in initialization and the sequential order of the data, the same optimum may not be reached for the two networks.

In Table 4.1.1, the Dice scores of Model 3 and Model 4 show that the models trained with TensorFlow and PyTorch performs equally well for batch size 2. This is expected, as the choice of framework should not have any impact on the models accuracy performance, as long as the network architecture and the training data is the same. The mathematical operations in the networks give the same result regardless of the framework. However, the way the operations are calculated, stored, and how the hardware is utilized, may differ between frameworks. This may give different results in training and inference performance, with respect to memory footprint and speed, as seen in table 4.1.1. During training, PyTorch seems to outperform TensorFlow in both training time and memory use. This is expected, as TensorFlow aims at being dynamic and user friendly, while PyTorch is more lightweight in comparison.

The results for Model 4 and Model 5 in Table 4.1.1 show that using mixed precision during training result in a 40% memory reduction. In spite of the reduction in memory footprint, the model accuracy does not seem to be reduced in any way. This indicates that no significant information is lost by reducing neither the input data nor most of the network layers to 16-bit. A memory reduction of 25% is also observed during inference with the use of mixed precision.

A speed-up in training and inference time is also expected, as using 16-bit floating point speeds up data transfer due to lower memory bandwith requirements, and results in faster math operations. As the results show however, using mixed precision actually increases the training and inference time. This should not be happening, and might be due to a bug in the implementation or the framework. The PyTorch models were trained with *benchmark* enabled, which is a function that optimizes the model for maximum performance on the GPU, based on the inputs of the network. With this function enabled, the training speed was reduced from 562 s/epoch to 96.2 s/epoch for Model 4, and 159 s/epoch to 103.6 s/epoch for Model 5. This indicates that the benchmark-function have a much larger effect on models with full precision than on models using mixed precision, at least for the hardware used in this study. However, this does not explain why the inference time was longer for Model 5, as the benchmark-function was not enabled during inference. The use of mixed precision is relatively new, as support for half-precision on commercial GPUs was not a given until recently. Even though the development is fast, not all software is optimized for half-precision, and it might explain the surprising result.

## 5.2 Evaluation of the PLS-Net

As we can see from Table 4.2.1, the PLS-Net only use ∼0.25 million parameters, to the U-Nets ∼6.6 million parameters. Because of the large reduction in parameters and computations, we expect the PLS-Net to have a lower memory footprint and to be faster. As the results shows, the PLS-Nets memory footprint is reduced by more than 30% during training compared to the U-Net. The memory use during inference is also reduced for the PLS-Net, and the inference time is shorter, as expected. Table 4.2.1 does however show that the PLS-Net is slower than the U-Net during training.

In the implementation of the DRDB-block, the block was implemented to be memory efficient by recomputing intermediate activations instead of storing them. This came at the cost of lower training speed, which may explain the unexpected result. However, Table 4.2.2 shows that while the memory use is reduced by 45% when using a memory efficient DRDB-block, the extra time used per epoch is relatively small, and does not explain by itself why the PLS-Net is so much slower during training. According to the developers at PyTorch, using so called *grouped convolutions* might not always trigger the 16-bit floating point path, and thus might not use TensorCores, which is what accelerates the mixed precision models. The DS-convolutions in the PLS-Net are implemented with such grouped convolutions, while the U-Net is not. This may explain why one epoch takes longer for the PLS-Net. Even though time per epoch is longer for the PLS-Net, the total training time for the model is 7% faster than the U-Net, as it uses fewer epochs to reach the minimum.

The large reduction in memory footprint during training allows PLS-Net to make use of larger input volumes at no batch size's expanse. Leveraging input volumes with a higher resolution provides refined and more detailed segmentation results, lessening the staircase effect around objects' edges. As a result, the PLS-Net architecture is well suited for applications requiring fine details such as the segmentation of pulmonary lobes.

## 5.3 Lobe Segmentation Study

As could be predicted, the segmentation models struggle more on the right lung than on the left one since there is one more lobe to take into account (cf. Table 4.3.1). The fissures separating the different lobes are relatively small and often incomplete, or at least inconsistent with accessory bits. Handling lobes segmentation in the right lung is thus more complex for the network. From Lee et al., incomplete fissures have been reported in 70% of their data set with a 9% occurrence ratio for accessory fissures, suggesting that a network would have a hard time relying on relevant information that is often of poor quality. From those observations, the segmentation results for the right middle lobe are understandably worse as it is defined by two fissures as opposed to any other lobe.

From Table 4.3.1, we can see that the results from this study do not reach the results of Lee et al., even though the same network architecture was used for both experiments. The inequality is likely due to the difference in the data used to train the network. While Lee et al. used 210 chest CT scans for their experiments, the experiments done for this thesis only had 71 CT scans available to use for the lobe segmentation task. This does not seem to be enough data for the model to generalize properly for this network. The quality of the ground truths used as target during training is also very important for the result, as the model only can be as good as these targets. While Lee et al. used ground truths that were checked and corrected by experienced radiologists, only 51 one of the targets used for this thesis was proper ground truths. The remaining 20 targets were obtained by using an existing lobe segmentation model, without being properly checked or corrected. As mentioned in 3.1, a consequence of this was that the models were trained on at least one incomplete target, which may have caused the model to underperform.

The overall Dice score of each fold in Table 4.3.1 shows that the results vary from 91.4% to 94.0%. These large variations also imply that the model does not generalize properly because of the limited data set used for training. Quantity is however not the only problem. The data used for training a model should also cover a large variety of different cases for the model to generalize. In the experiments done by Lee et al., the data set was obtained from patients with different lung diseases, including lungs with abnormalities like fibrosis, nodules, emphysema, ground-glass opacity and reticular opacity. The data set used in this thesis did not cover the same span of abnormalities, which may help explain the poor generalization.

Figure 4.3.4 shows that applying the lung mask over the predicted lobes successfully removes the falsely predicted voxels outside of the lungs. In Table 4.3.2, we can see that the overall Dice score improves by 0.3%, to 92.9%. This might not be a large improvement, but as long as the lung mask is available, it is a very simple and computationally inexpensive operation. The result is still far from that of Lee et al., which implies that a lot of the inaccuracy is due to the other types of errors found during the visual inspection, such as the "holes" in the lobes, inaccurate detection of the fissures, and fragments of one lobe inside another lobe.

## 5.4 Lobe Segmentation on Lungs with Abnormalities

Figure 4.4.1 shows that applying the lobe segmentation model to a CT volume from another data set, gives equally good results as for a CT volume from the data set used to train the model. This shows that the model generalizes well across data sets, as long as there are no significant abnormalities, which the model is not properly trained for. When the model is applied to a volume where one of the lungs contains abnormalities however, the result is worse, as seen in Figure 4.4.2. The data set used to train the model does not include lungs with abnormalities like this, and it is thus not very surprising that the models' accuracy is reduced in such cases.

From figure 4.4.2 we can see how abnormalities in one part of the lung also affects slices where the abnormalities are not present. This shows how the model use 3D global information when making the predictions, and that irregularities in one place propagate to other parts of the lungs. Abnormalities in the right lung will however not affect the left lung, as the model does not use any information from the right lung to make predictions in the left lung.

## 5.5 Limitations of Study and Future Work

A shortcoming of this thesis is the lack of proper ground truth for all CT volumes in the data set. While generating annotations by using existing models gave us more data to work with, learning from imperfect annotations limits the potential of training using supervised learning. Achieving proper ground truths is desirable, but is also very time consuming and requires the help of expert radiologists.

Collecting more data would also be of great use when trying to improve the model performance. For the models to generalize properly, a sufficient amount of data that covers many different cases is a requirement. As lung- and lobe segmentation tasks play an important role in computer-aided diagnostic and treatment planning of lung diseases, it is important that the models perform well on diseased lungs. These lungs often contain abnormalities, which we showed that the model did not perform well on, as it was not trained for it. Adding more data with different abnormalities would thus be an important step in making these models capable of segmenting diseased lungs and lobes.

There is a lot of potential in using post-processing to improve the segmentation masks of the lobes. In this thesis, only a simple multiplication of the lung mask was used to improve the results. Unwanted fragments inside the lobes could be removed by mapping groups of voxels with one label surrounded by another label, to that neighbouring label. By using a lot of the same techniques, holes in the lobes could also be filled with the value of the neighbouring labels.

# 6 | Conclusion

In this thesis, the 3D U-Net introduced by Çiçek et al. [11] and the PLS-Net introduced by Lee et al. [12], were trained and evaluated for automatic lung segmentation, with different combinations of deep learning frameworks, training precision, batch sizes and input resolutions. The PLS-Net was also trained and evaluated for automatic lobe segmentation, followed by a simple post processing step. The different models were evaluated with respect to training, inference and accuracy performance.

The test results for the lung segmentation showed that the models trained with batch size 2 outperformed the models trained on batch size 1 with respect to accuracy. For the models trained with batch size 2, the choice of framework and training precision did not seem to affect the models accuracy at all. PyTorch gave better training performance, but was slower during inference. However, the inference time was in the order of milliseconds, and the difference will not have any practical impact for the user. Using mixed precision over full precision reduced the memory footprint during training and inference, without affecting the accuracy. Training and inference time were longer for mixed precision than full precision, which was unexpected, and may be due to an implementation error, a bug in the framework or sub-optimal use of the hardware.

Comparison of the two networks showed that the PLS-Net gave a reduction of more than 30% in memory footprint compared to the U-Net, for the same batch size and training data. This reduction makes it possible to feed larger input volumes, or use larger batch sizes for the PLS-Net. The PLS-Net was faster during inference, but slower during training per epoch, which according to the developers at PyTorch, might be due to a bug with the depthwise separable convolution used in the PLS-Net.

Results from the lobe segmentation study showed that the Dice scores obtained in this study did not reach the scores of Lee et al., as they used almost three times as much data, with larger variability across the data set and proper ground truths checked by expert radiologists. Adding lung masks to the predicted lobe segmentation masks for post-processing successfully removed the false positives outside of the lungs, and improved the overall Dice score by 0.3%. The new scores did still not reach the result obtained by Lee et al., as the post-processing did not handle holes (false negatives), fragments and inaccurate fissure detection in the segmented mask.

Applying the lobe segmentation model to CT volumes from a different data set than the one used for training, showed that the model did well on volumes without any significant abnormalities, which means that the model generalizes well across data sets. On volumes containing abnormalities, the model did not perform well. This was not surprising, as the data set used to train the model did not include lungs with severe abnormalities. Overall, the results show that the lung- and lobe segmentation task really is a data diversity problem, more than a problem with the method.

# Bibliography


[1] T. Zhao *et al.*, "Lung segmentation in CT images using a fully convolutional neural network with multi-instance and conditional adversary loss", in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, Washington, DC: IEEE, Apr. 2018, pp. 505–509, ISBN: 978-1-5386-3636-7.

[2] S. Hu *et al.*, "Automatic lung segmentation for accurate quantitation of volumetric X-ray CT images", *IEEE Transactions on Medical Imaging*, vol. 20, no. 6, pp. 490–498, Jun. 2001, ISSN: 02780062.

[3] S. F. Nemec *et al.*, "Upper Lobe–Predominant Diseases of the Lung", en, *American Journal of Roentgenology*, vol. 200, no. 3, W222–W237, Mar. 2013, ISSN: 0361-803X, 1546-3141.

[4] P. Hua *et al.*, "Segmentation of pathological and diseased lung tissue in CT images using a graph-search algorithm", in *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, Chicago, IL, USA: IEEE, Mar. 2011, pp. 2072–2075, ISBN: 978-1-4244-4127-3.

[5] L. W. Hedlund *et al.*, "Two methods for isolating the lung area of a CT scan for density information.", en, *Radiology*, vol. 144, no. 2, pp. 353–357, Jul. 1982, ISSN: 0033-8419, 1527-1315.

[6] J. Pu *et al.*, "Adaptive border marching algorithm: Automatic lung segmentation on chest CT images", en, *Computerized Medical Imaging and Graphics*, vol. 32, no. 6, pp. 452–462, Sep. 2008, ISSN: 08956111.

[7] A. Mansoor *et al.*, "Segmentation and Image Analysis of Abnormal Lungs at CT: Current Approaches, Challenges, and Future Trends", en, *RadioGraphics*, vol. 35, no. 4, pp. 1056–1076, Jul. 2015, ISSN: 0271-5333, 1527-1323.

[8] A. P. Harrison *et al.*, "Progressive and Multi-path Holistically Nested Neural Networks for Pathological Lung Segmentation from CT Images", en, in *Medical Image Computing and Computer Assisted Intervention MICCAI 2017*, M. Descoteaux *et al.*, Eds., vol. 10435, Cham: Springer International Publishing, 2017, pp. 621–629, ISBN: 978-3-319-66178-0 978-3-319-66179-7.

[9] A.-A.-Z. Imran *et al.*, "Automatic Segmentation of Pulmonary Lobes Using a Progressive Dense V-Network", in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, D. Stoyanov *et al.*, Eds., vol. 11045, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2018, pp. 282–290, ISBN: 978-3-030-00888-8 978-3-030-00889-5.

[10] J. Park *et al.*, "Fully Automated Lung Lobe Segmentation in Volumetric Chest CT with 3D U-Net: Validation with Intra- and Extra-Datasets", en, *Journal of Digital Imaging*, vol. 33, no. 1, pp. 221–230, Feb. 2020, ISSN: 0897-1889, 1618-727X.

[11] Ö. Çiçek *et al.*, "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, S. Ourselin *et al.*, Eds., vol. 9901, Cham: Springer International Publishing, 2016, pp. 424–432, ISBN: 978-3-319-46722-1 978-3-319-46723-8.

[12] H. Lee *et al.*, "Efficient 3D Fully Convolutional Networks for Pulmonary Lobe Segmentation in CT Images", *arXiv:1909.07474 [cs, eess]*, Sep. 2019, arXiv: 1909.07474.

[13] J. G. Betts *et al.*, *Anatomy and physiology*, English. OpenStax, 2017, OCLC: 1001472383, ISBN: 978-1-947172-04-3.

[14] K. Bae *et al.*, "Severity of pulmonary emphysema and lung cancer: Analysis using quantitative lobar emphysema scoring", en, *Medicine*, vol. 95, no. 48, e5494, Dec. 2016, ISSN: 0025-7974.

[15] P. Pahal *et al.*, "Typical Bacterial Pneumonia", eng, in *StatPearls*, Treasure Island (FL): StatPearls Publishing, 2020.

[16] M. M. Woolfson, *The fundamentals of imaging from particles to galaxies*, English. Singapore; London: World Scientific, 2012, OCLC: 929736255, ISBN: 978-1-84816-686-8.

[17] S. K. Zhou *et al.*, Eds., *Deep Learning for Medical Image Analysis*, ser. Elsevier and MICCAI Society book series. London ; San Diego: Elsevier/Academic Press, 2017, OCLC: ocn957503470, ISBN: 978-0-12-810408-8.

[18] I. Goodfellow *et al.*, *Deep Learning*. MIT Press, 2016.

[19] Y. LeCun *et al.*, "Deep learning", en, *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 0028-0836, 1476-4687.

[20] R. C. Gonzalez *et al.*, *Digital Image Processing: Global Edition*, 4th Edition. Pearson Education Inc, 2018, ISBN: 978-93-5306-298-9.

[21] J. Li *et al.*, "Understanding Generalization in Deep Learning via Tensor Methods", *arXiv:2001.05070 [cs, stat]*, May 2020, arXiv: 2001.05070.

[22] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018, OCLC: ocn982650571, ISBN: 978-1-61729-443-3.

[23] N. Srivastava *et al.*, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958,

[24] S. Ioffe *et al.*, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *arXiv:1502.03167 [cs]*, Mar. 2015, arXiv: 1502.03167.

[25] K. He *et al.*, "Deep Residual Learning for Image Recognition", *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385.

[26] G. Huang *et al.*, "Densely Connected Convolutional Networks", *arXiv:1608.06993 [cs]*, Jan. 2018, arXiv: 1608.06993.

[27] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", *arXiv:1603.04467 [cs]*, Mar. 2016, arXiv: 1603.04467.

[28] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", *arXiv:1912.01703 [cs, stat]*, Dec. 2019, arXiv: 1912.01703.

[29] K. Yan *et al.*, "DeepLesion: Automated mining of large-scale lesion annotations and universal lesion detection with deep learning", *Journal of Medical Imaging*, vol. 5, no. 03, p. 1, Jul. 2018, ISSN: 2329-4302.

[30] R. D. Rudyanto *et al.*, "Comparing algorithms for automated vessel segmentation in computed tomography scans of the lung: The VESSEL12 study", en, *Medical Image Analysis*, vol. 18, no. 7, pp. 1217–1232, Oct. 2014, ISSN: 13618415.

[31] C. for Open Medical Image Computing, "LUng Nodule Analysis (LUNA16) All Images",

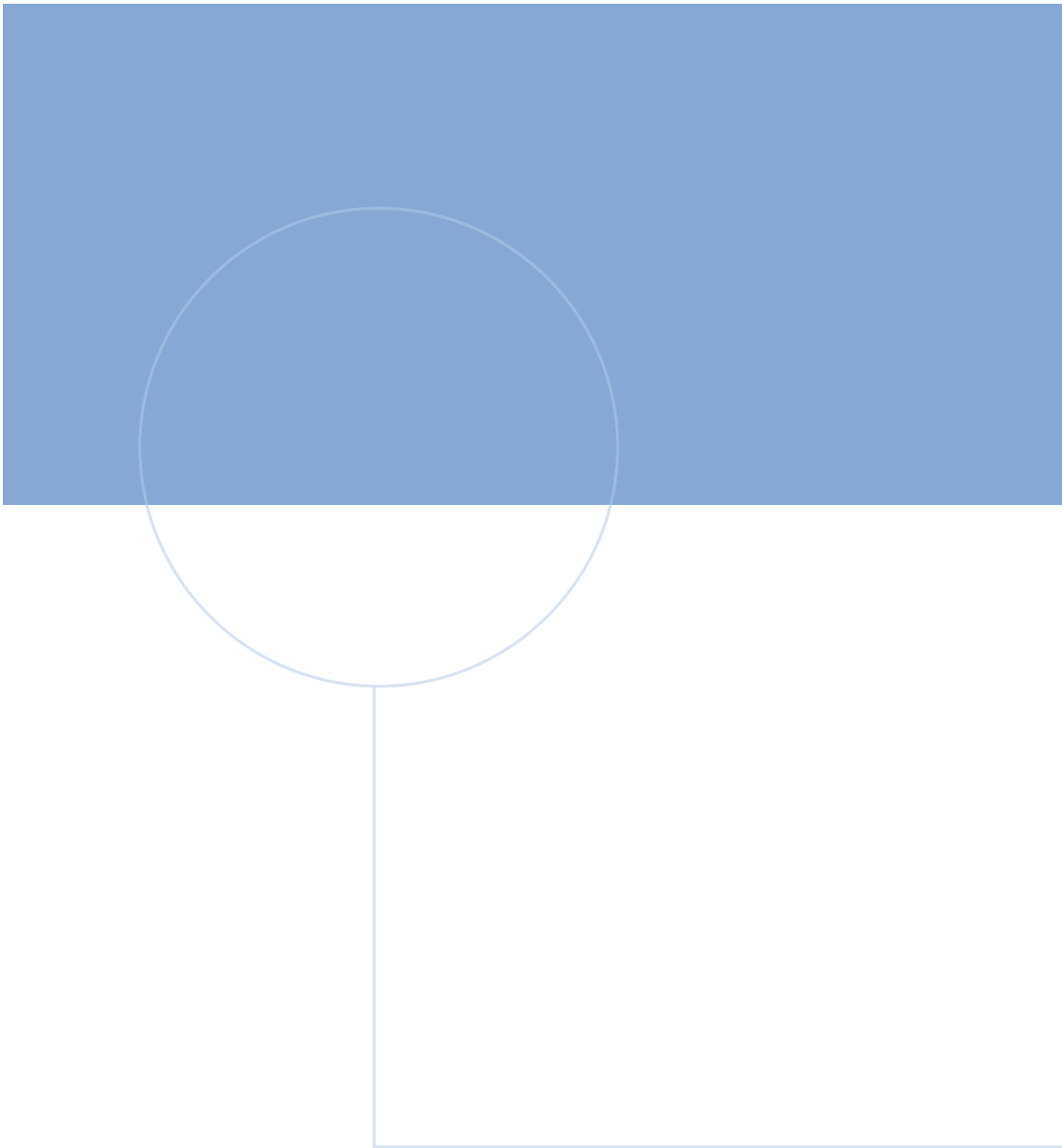
[32] E. F. Hofstad *et al.*, "Automatic registration of CT images to patient during the initial phase of bronchoscopy: A clinical pilot study: Automatic bronchoscopy registration", en, *Medical Physics*, vol. 41, no. 4, p. 041 903, Mar. 2014, ISSN: 00942405.

[33] D. Bouget *et al.*, "Semantic segmentation and detection of mediastinal lymph nodes and anatomical structures in CT data for lung cancer staging", en, *International Journal of Computer Assisted Radiology and Surgery*, vol. 14, no. 6, pp. 977–986, Jun. 2019, ISSN: 1861-6410, 1861-6429.

[34] J. Hofmanninger *et al.*, "Automatic lung segmentation in routine imaging is a data diversity problem, not a methodology problem", *arXiv:2001.11767 [physics, stat]*, Jan. 2020, arXiv: 2001.11767.

[35] O. Ronneberger *et al.*, "U-Net: Convolutional Networks for Biomedical Image Segmentation", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab *et al.*, Eds., vol. 9351, Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24573-7 978-3-319-24574-4.

[36] D. P. Kingma *et al.*, "Adam: A Method for Stochastic Optimization", *arXiv:1412.6980 [cs]*, Jan. 2017, arXiv: 1412.6980.



NTNU

Kunnskap for en bedre verden