Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Svea Victoria Strassburger

born in Munich (Germany)

May 2024

Investigating the effectivity of deep learning, specifically nnU-Net, in enhancing in vivo Fluorescence Molecular Tomography through semantic segmentation as an alternative to resolving the ill-posed inverse problem

This Bachelor Thesis has been carried out by Svea Victoria Strassburger at the
German Cancer Research Center (DKFZ)
under the supervision of
Dr. Jörg Peter and Prof. Dr. Mark E. Ladd

Abstract

Fluorescence Molecular Tomography (FMT) and Fluorescence Molecular Imaging (FMI) are powerful non-invasive optical imaging techniques that enable the visualization of cellular and molecular activities within biological tissues. With their strong specificity and sensitivity they show potential for various applications in tumor diagnosis, drug development and the study of biological and medical processes.

However, its effective application in research and clinical settings is affected by reconstruction challenges arising from the strong scattering of photons propagating through tissue. The scattering leads to an ill-posed definition of the inverse problem, which forms the basis of every reconstruction process.

This bachelor thesis seeks to bypass the ill-posed inverse problem by taking advantage of deep learning techniques and implementing a neural network. The goal is to improve in vivo Fluorescence Optical Tomography through semantic segmentation of tomography images. Which is done by training and testing the neural network architecture nnU-Net on datasets based on simulated fluence data related to simple phantom geometries.

Zusammenfassung

Die Fluoreszenz-Molekulartomographie (FMT) und die Fluoreszenz-Molekularbildgebung (FMI) sind vielversprechende nicht-invasive optische Bildgebungstechniken, die die Visualisierung zellulärer und molekularer Aktivitäten innerhalb biologischer Gewebe ermöglichen. Aufgrund ihrer starken Spezifität und Sensitivität sind sie sehr vielversprechend für verschiedene Anwendungen in der Tumordiagnose, der Arzneimittelentwicklung sowie der Erforschung biologischer und medizinischer Prozesse.

Die effektive Anwendung in Forschungs- und klinischen Umgebungen wird jedoch durch Rekonstruktionsherausforderungen beeinträchtigt, die sich aus der starken Streuung von Photonen im Gewebe ergeben. Diese Streuung führt zu einer unbestimmten und schlecht konditionierten Definition des inversen Problems, das die Grundlage für jeden Rekonstruktionsprozess bildet.

Diese Bachelorarbeit zielt darauf ab, einen alternativen Ansatz zur Lösung des inversen Problems durch die Verwendung von Deep-Learning-Techniken und die Implementierung eines neuronalen Netzwerks zu finden. Ziel ist es, die in-vivo-Fluoreszenzoptische Tomographie durch semantische Segmentierung von Tomographiebildern zu verbessern. Dies erfolgt durch Trainieren und Testen der neuronalen Netzwerkarchitektur nnU-Net an Datensätzen, die auf simulierten Fluenzdaten basieren und sich auf einfache Phantomgeometrien beziehen.

Contents

Chapter 1

Introduction

The following chapter will provide motivational insights into the topic before offering a brief overview of the theoretical background, with a special focus on Optical Imaging modalities and Neural Networks Methods. Tools that were essential for this work are listed in Chapter 2. The experimental setup and implemented configurations will be displayed in Chapter 3. Chapter 4 will be rounded up by the examination of the results through analysing the efficiency of the network, the usefulness of the tools, and the quality of the output results of the network. Finally in Chapter 5, a conclusion and an outlook on the further paths of research build upon the examined method are provided. As the research work is fundamentally based on self-written python code, that code as well as more examples of the resulting datasets can be found in the very end of the Appendix.

1.1 Motivation

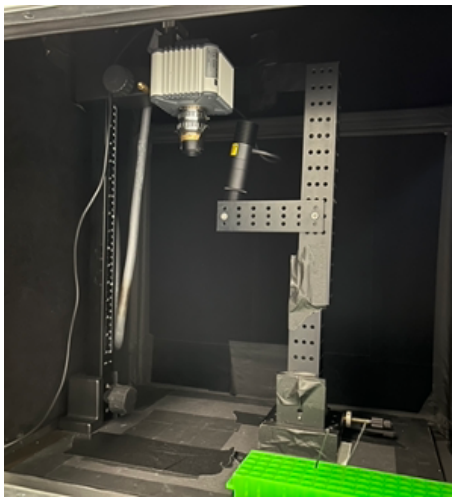
Molecular Imaging (MI) is an emerging technology that enables non-invasive visualization of physiological and pathological activities at cellular and molecular levels within biological tissues. This ability allows the study of biological and medical processes, as well as the diagnosis and management of diseases. In comparison to non-optical imaging modalities like Computed Tomography (CT), Magnetic Resonance Imaging (MRI), and Ultrasound Imaging, MI can detect disease occurrences before noticeable morphological changes manifest in the early stages of pathological processes. Moreover, MI enables real-time, non-invasive dynamic monitoring in vivo. As a result, MI aims to contribute to tumor detection, gene therapy, drug development, therapeutic evaluation, and intraoperative navigation [20, 49].

A significant part of Molecular Imaging (MI) technology is the field of Optical Molecular Imaging (OMI). Among various OMI techniques, two-dimensional Fluorescence Molecular Imaging (FMI) is very common and comes along with strong specificity and sensitivity.

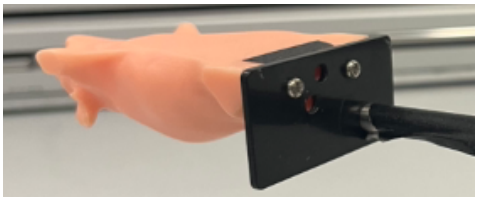
To extend the field of application from two dimensions to three dimensions, Fluorescence Molecular Tomography (FMT) was introduced. This three-dimensional tomography enables non-invasive reconstruction of three-dimensional spatial information and the concentration distribution of internal fluorescent targets in small animals [49].

Acquiring Basic FMI Images

Practical experience with reference to FMI was experimentally obtained. Therefor a basic experimental setup at the DKFZ was used. Consisting of a darkbox with Laser, CCD-Camera, Filter (see Fig. 1.1a) and fluorophores probes inside either a cylinder or a mouse phantom (see Fig. 1.1b) for obtaining basic fluorescence molecular images.

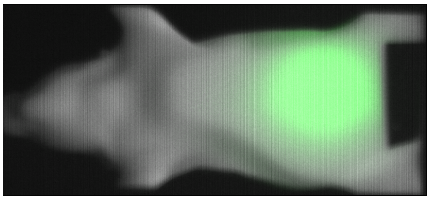


(a) Experimental Setup

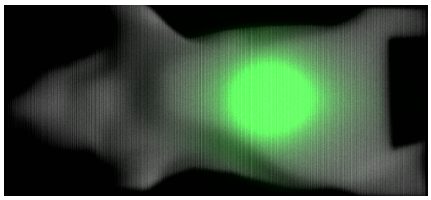


(b) Mouse phantom with two tumor inclusions

Figure 1.1: Experimental setup box with CCD-Camera, filter and laserpointer (left). Mouse phantom with two tumor inclusions that is placed inside the experimental box underneath the camera (right).



(a) Probe deep inside of body



(b) Probe shallow to the surface

Figure 1.2: Basic Fluorescence Molecular images obtained in the lab by using a CCD-Camera and a mouse phantom with two inclusion possibilities. The fluorophore probe is located deep inside of the body (left); the fluorophore probe is located shallow to the surface of the body (right). The intensities of the probes in both images are independently scaled.

Observing figure 1.2, it is shown that the shallow probe is displayed with much higher intensity in the recorded image than the deeper probe. The research presented in [31]

underlines this observation by demonstrating the impact of photon scattering on Fluorescence Imaging in living tissue (see Fig. 1.3).

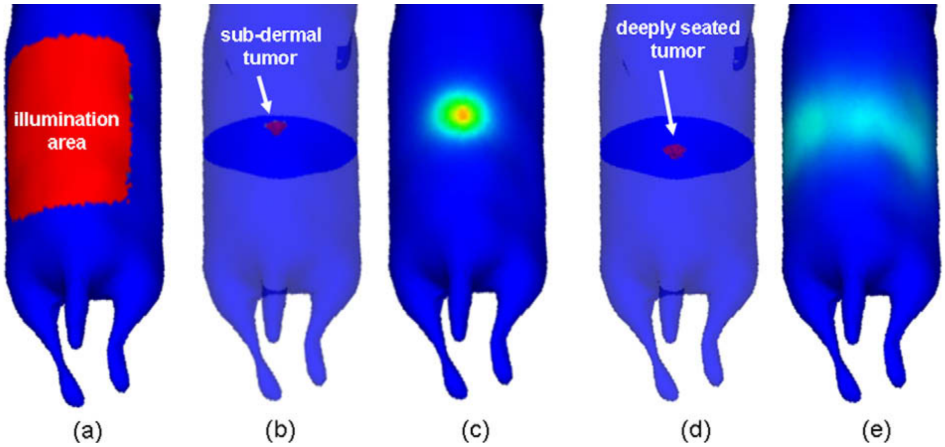
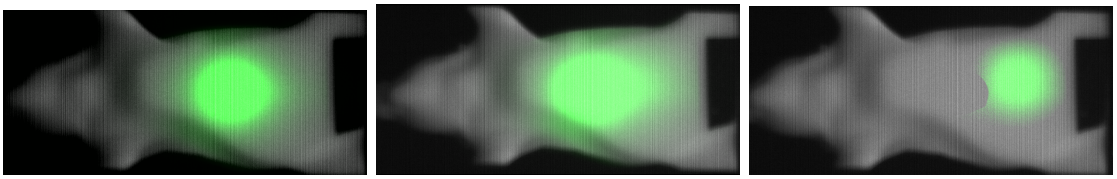


Figure 1.3: Mouse phantom with two tumor inclusions used to demonstrate the impact of photon scattering on Fluorescence Imaging in living tissue [31].

Two numerically simulated mouse models were utilized, with a spherical tumor positioned near the surface (b) and close to the center of the mouse (d). Figure (a) maps the illuminated area on the surface of the animal, while figures (c) and (e) illustrate the outgoing light at the surface for the tumors shown in (b) and (d), respectively. The intensity of the outgoing light in (e) is approximately one-thousandth of that shown in (c). Also, the distribution at light fluence on the surface is very diverse.

Shifting focus back to experimentally conducted FMI implementation, when both probes are simultaneously placed inside the phantom and an image is captured, there appears to be no visually distinguishable difference between the image of the two probes (Fig. 1.4b) and the image captured with only the shallow probe present (Fig. 1.2b).



(a) Single probe located shallow to the surface of the body

(b) Two probes, one located deep inside, one shallow to the surface

(c) Visualization of deeper probe by subtracting the shallow probe off the image

Figure 1.4: Basic Fluorescence Molecular image displaying two Fluorophores probes of which one is located deep inside and one shallow to the surface of the body (b). The deeper seated probe is not detectable, as it is overshadowed by the surface probe (b), only when latter is subtracted off the image, the deeper probe can be seen in the image (c).

This simple application already characterizes the challenges inherent in optical imaging. As the probe is situated deeper, its detection becomes increasingly difficult, leading to a

decline in imaging quality. Moreover, the presence of multiple probes within the body reduces the ability to differentiate between them and to detect probes located deeper, resulting in **limited detection accuracy**. These challenges encountered in Fluorescence Molecular Imaging directly impact the reconstruction challenges encountered in Fluorescence Molecular Tomography.

FMT promises great research potential for various practical applications related to tumor detection. However, it still faces major challenges in biological tissue imaging, especially limited detection accuracy and image quality which negatively impacts reconstruction. There are several approaches to solve these current reconstruction challenges associated with FMT. So far, there is no completely satisfactory approach, but many different, partially effective methods (review in [49], [20], [2]). As the capability of deep learning techniques has shown improvements in various fields of (medical) research, this work aims to present a new deep learning-based approach to solve the challenges of FMT reconstruction.

1.2 Imaging Modalities

When dealing with imaging techniques, they can be divided into the following areas. On the one hand, there are Computed Tomography (CT) and Magnetic Resonance Imaging (MRI), which detect anatomical structures. In contrast, nuclear medicine techniques make functional molecular parameters visible. Examples of these are Positron Emission Tomography (PET) and Single Photon Emission Computed Tomography (SPECT). In practice, it is common to combine anatomical and nuclear medicine techniques (e.g. SPECT/CT or PET/MR). At last in the field of small animals, optical imaging can be considered an alternative to nuclear medicine techniques [1]. The following subsection provides a brief introduction to the imaging modalities used in this project. To understand the concept of regular reconstruction, which will be used later, the theory of Computed Tomography is first briefly presented.

1.2.1 Computer Tomography

X-ray Computed Tomography (CT) utilizes the penetrating ability of X-rays to capture a series of two-dimensional **radiographs** of the object from various angles (Fig. 1.5). A radiograph obtained at a specific angle of illumination is called **projection** or **X-ray image**. The digital unit of such a two-dimensional image is called **Pixel**, an abbreviation for picture element. A computer reconstruction algorithm is then used to generate a stack of cross-sectional slices from these two-dimensional X-ray images of the object. This process provides a digital three-dimensional grayscale representation (called **tomogram**) of the internal structure of the object. The basic unit of such a three-dimensional digital representation of an image or object is described as **Voxels**, an abbreviation for volume elements. This tomogram provides the necessary data for numerically reconstructing the object including a volume structure.

Figure 1.5: Virtual slice through an attenuation contrast tomography (CT-Scan) of an iodine-stained mouse embryo [46].

The intensity of the X-ray beam (I) at distance x within a material is described by the initial unattenuated X-ray intensity (I_0) and the linear attenuation coefficient (μ) as [45]:

$$I = I_0 \exp^{-\mu x} \tag{1.1}$$

Thus, when an electromagnetic X-ray wave passes through an object, its decrease in intensity, also described as **attenuation**, is measured. The decrease depends on the type of tissue that is being penetrated, generating different transmission data. The average **linear attenuation coefficient** (μ) of material between the source and detector element is calculated by using the relationship of intensities shown in equation 1.1:

$$\mu = \frac{1}{x} \ln \left(\frac{I_0}{I} \right) \tag{1.2}$$

The measured value of μ alone is limited because the line between the source and detector may pass through varying amounts of matter with different attenuation coefficients. For example, a highly attenuated X-ray signal could pass through a small but dense bone region or a thicker soft tissue region, making it impossible to determine the exact cause with one measurement. The solution is to acquire many X-ray transmission measurements from different angles covering the entire volume of interest. This data is then combined to create the CT-Scan [45].

Highly attenuating features are displayed brightest in CT-Scans. This is why attenuation contrast is well suited to distinguish materials with large differences in electron density. It refers to objects containing very different materials, for example, bone fractures. The attenuation coefficient rises with an increasing atomic number (due to scattering by the electrons) and falls with increasing X-ray energy. That is why for example soft tissues, due to a low atomic number, show poor attenuation contrast. Poor attenuation contrast can also origin in specimen that consist of materials that attenuate similarly (again for example, soft tissue). Because of both factors there are many materials, that are better suited for phase contrast imaging or other methods such as Magnetic Resonance Imaging (MRI) instead of attenuation contrast [46].

Configurations

The three basic physical components of a CT scanner are the X-ray source, the X-ray detector and the sample stage (core). In the medical field, common X-ray Computed Tomography configurations consists of a gantry system where the source and the detector rotate in tandem around the patient, animal or specimen. In order to reconstruct an object accurately, a parallel beam acquisitions projection data from 180 degrees is required.

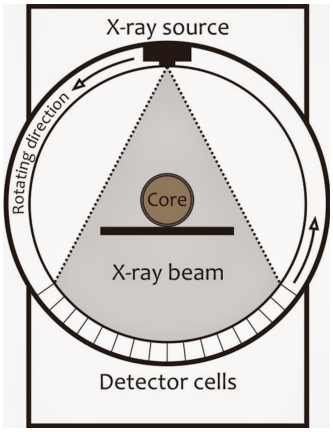


Figure 1.6: Common X-ray Computed Tomography configurations for medical use [47].

After the collection of the projection data the analysis requires only visual inspection of these virtual greyscale slices cut through the tomogram. Nevertheless, there is often a need to quantify and three-dimensionally visualize specific regions within the volume e.g. for radio therapy. This is achieved by segmentation, analysis and volume rendering [46].

Segmentation is an image-processing procedure of assigning a label to every voxel in a volume such that voxels with the same label share certain characteristics.

Volume rendering is a three-dimensional representation of data, often with certain segmented regions colourized or rendered transparent.

1.2.2 Fluorescence Molecular Imaging

In order to non-invasively differentiate between in vivo (referring to processes taking place in living organism) physiological and pathological activities at cellular and molecular levels, Molecular Imaging (MI) has become an important tool for biomedical research. MI can be either used for studying biological and medical processes or diagnosing and managing diseases, especially for tumor research and cancer diagnosis [20]. A sub-field of MI is referred to as Optical Molecular Imaging (OMI). Among others OMI technologies generally include diffusion optical tomography (DOT), Bioluminescence Imaging and Fluorescence Molecular Imaging (FMI) [49]. Latter will be discussed in the following in detail.

Physical Definition

Fluorescence

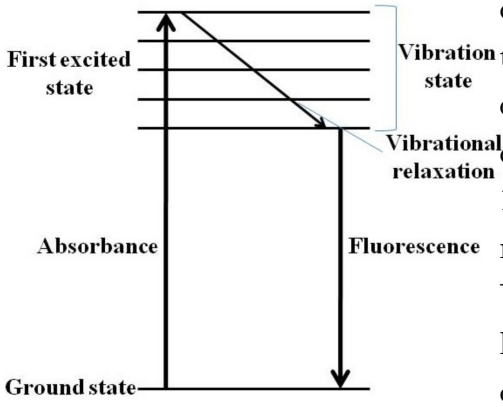


Figure 1.7: Jablonski diagram of fluorescence [34].

Fluorescent molecules, called **fluorophores**, are capable of being excited from their ground state to their first excited state, via absorption of light energy at a certain wavelength. This process is commonly illustrated in a Jablonski diagram (Fig. 1.7). While being in their excited state, the excited molecules can relax to lower vibrational state due to **Vibrational relaxation** 1.7. Which has an excited lifetime only ranging from $10^{-15}s$ to $10^{-9}s$ [40]. The decrease of vibrational states results in the emission of light energy on a different wavelength, while the molecules return to their ground state. The difference between the excitation and emission energy maxima is called the **Stokes shift**. On the other hand, the ratio of the number of fluorescence photons emitted to the number of photons absorbed, describing the emission efficiency, is called **Quantum yield**. Stokes shift and Quantum yield together represent the total amount of absorbed light.

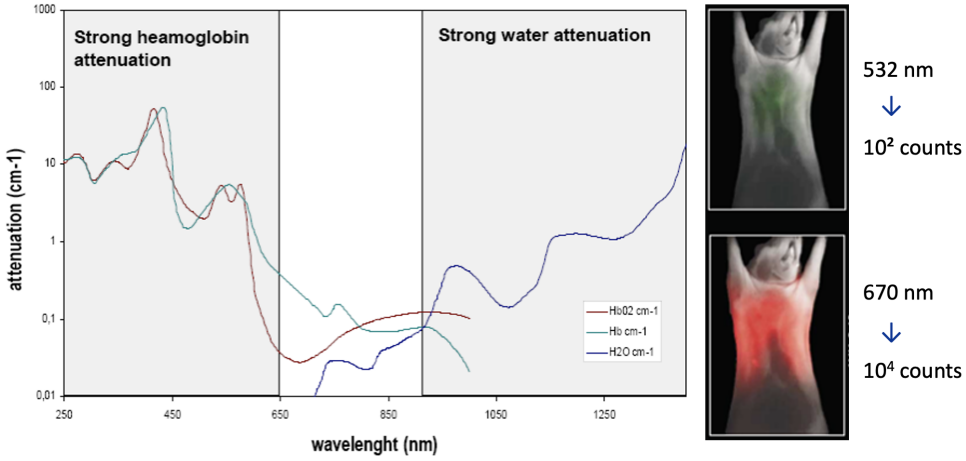


Figure 1.8: Absorption of light in tissue [11].

The entire process as well as the emitted ray itself, is called **fluorescence** [40]. The emitted light is of lower energy, and thus lower frequency and longer wavelength, than the absorbed light. This means that the color of the light that is emitted or fluoresced is different from the color or area of the light that has been absorbed. Most commonly the activating light is in the range between 400 nanometers (nm) to 700 nanometers, marking the visible spectrum of light. A common example of fluorophores include fluorescent dyes and proteins like green fluorescent protein (GFP). The activating light source for a GFP

ranges between 480 and 540 (visualized in figure 1.8 with the top right mouse). However, as can be seen in at the in figure 1.8, the excitation and emission wavelengths should be higher than 650 nm and lower than 900 nm to minimize absorption due to the strong water and haemoglobin attenuation.

Fluorescence Spectra: Attributes of any fluorophore, can be described through its fluorescence spectra (see figure 1.9). Fluorescence spectra describe the emission (blue line) of light by a fluorophore after it has been excited by absorbing light (green line) at a specific wavelength. Different filters (yellow, pink lines), including the wavelength of the excitation ray (red line), can be displayed as well. More detailed, as a fluorophore is excited most efficiently by light of a particular wavelength, this wavelength only marks the excitation maximum for the fluorophore. Light with a wavelength near the excitation maximum can also cause excitation, but it does so less efficient. The absorption spectrum (green line) indicates the wavelengths of light that the fluorophore absorbs most strongly. The absorption spectrum typically consists of one or more peaks, each corresponding to a specific electronic transition within the molecule.

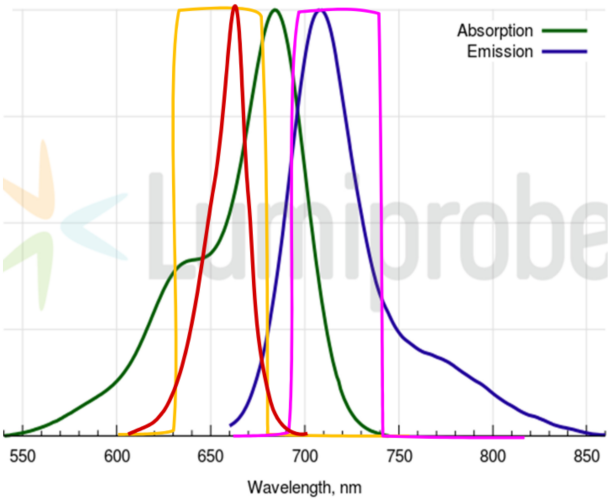


Figure 1.9: Absoprtion (green) and Emission (blue) Spectra of Cy5.5 fluorophore. With Bandpass Filter: Exiter (yellow), Emitter (pink), Excitation light source (red) [34]. This fluorophore was used for figure 1.2 and 1.4.

Fluorescence microscope

Fluorescence microscopes are an optical microscope used to study properties of organic or inorganic substances using the phenomena of fluorescence and phosphorescence instead of, or in addition to, reflection and absorption.

So called **contrast markers** or **smart probes** are used to either correctly label an organism using the fluorophore for further observation or to activate/manipulate the fluorophore at its final position as needed. For example due to the **Auto-Synthesis**, some

fluorescent proteins can be synthesized by cells, therefore being able to infiltrate e.g. genes in order to represent gene activation. Another example is **Enzymatically Activated Fluorescence**, which can be used to represent enzymatic activity, as it enzymatically cleaves molecules that have two fluorophores attached at the same time [37].

Spectrofluorometers mainly consists of four parts: light sources, monochromators, optical filters and a detector (Fig. 1.10) ([34], [40]).

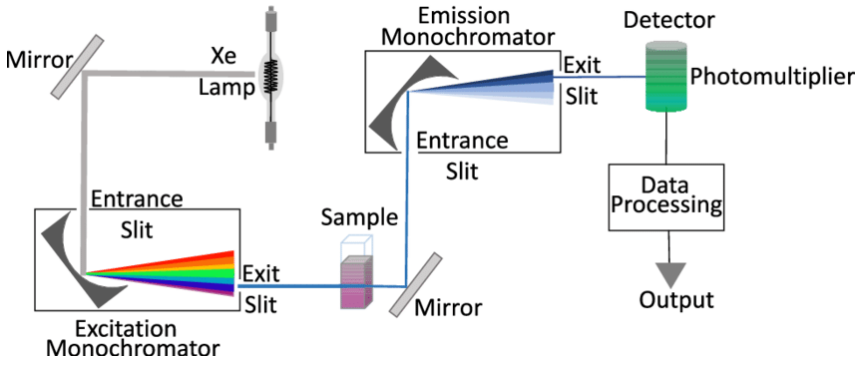


Figure 1.10: Schematic representation of a fluorescence spectrometer [12].

The light of the **light sources** (xenon lamps, Xe-Hg arc lamps or even LED light sources) needs to be in the visible range, that can provide the needed excitation energy. Prisms and diffraction gratings are two mainly used types of **monocharomators**, converting polychromatic light into monochromatic light. In addition to monocharomatore, **Optical filters** (monochromatic filter and long-pass filter (Fig. 1.9)) are used in order to further purifying the light. Those types of optical filters are also applied as **Excitation** and **Emission filters**. For selective excitation, a filter that transmits a narrow range of wavelengths is typically used. For example a basic excitation filter blocks all light up to a specific wavelength and allows all light above that wavelength to transmit. Whereas a bandpass excitation filter allows only a small range of light to be transmitted while blocking all light on either side of that range. On the other hand detecting only the fluorescence emission of a sample is complicated by the presence of stray light arising from sources. **Emission filter** such as either Longpass or Bandpass can be utilized. For a single colors, a **Longpass filter** is used, that blocks out the excitation light to reduce background noise but transmits everything else, maximizing the signal. If instead multiple colors are used in the sample, a **Bandpass emission filter** can be used here too, for isolating the emission from each dye.

In the end the emitting light has to be detected and outputted, for example through sensors in a camera device. For FMT in particular usually a CCD-Sensor is used.

1.2.3 Cameras, Sensors and Pixel

The emitted photons are being captured by a detector. For optical imaging, this detector can be a camera, as wavelengths of the visible spectrum need to be captured. The core of every camera are sensors. Sensors however consist of light receptors, which can also be described as individual photodiodes. They convert light energy into electrical charge. The amount of charge at each pixel is directly proportional to the intensity - brightness - of the light hitting it. In addition to capturing brightness information, the receptors are equipped with red, green, and blue color filters to capture different colors, through recording the intensity of red, green or blue light.

The electrical signals from all photodiodes on the sensor are forwarded to the camera's image processor, which interprets all this information and determines the color and brightness values of each individual light receptor, composing a digital image. The core unit of this image is called pixel - which is a single two-dimensional picture element. Often a sensor is described with a total number of megapixels, referring to millions of picture elements. But actually there is not necessarily a direct correspondence between the individual photodiodes on the sensor and the pixels in the resulting digital image, as some of the sensor data is used for technical processes inside of a camera. Hence it is more precise to describe the sensor with a certain number of "effective pixels". Nevertheless, for further references, the effective pixel will equally be addressed as pixel, disregarding the details just described.

If two sensors have the same number of megapixels but one sensor is arranged over a larger area, then the individual pixels (right cylinder in figure 1.11) must be larger and can therefore capture more light, allowing more photons to be converted into photoelectrons. This increases the sensitivity of the sensor. However, this is at the cost of resolution. As on the other side, smaller pixels are able to provide higher spatial resolution but yet capture less photons per pixel [7].

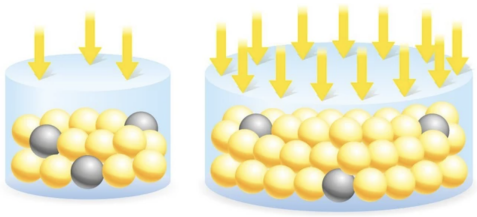


Figure 1.11: A larger pixel (right) captures more photons (depicted in yellow) and relatively less noise (gray) [7].

When talking about resolution, it can be distinguished between different types in different contexts. Firstly the **Temporal resolution** describes the time required to acquire enough projections ("time per scan"), of sufficient signal to noise ratio, to reconstruct an image of the desired quality. The time per scan can often be shortened by using fewer projections combined with iterative reconstruction techniques.
Secondly, the most commonly referred-to resolution is the **spatial resolution**, which in-

dicates the smallest linear distance between two points that can be distinguished in the reconstructed image, which does not necessarily lead to an equate to pixel size. Consequently images having higher spatial resolution are composed with a greater number of pixels than those of lower spatial resolution [38].

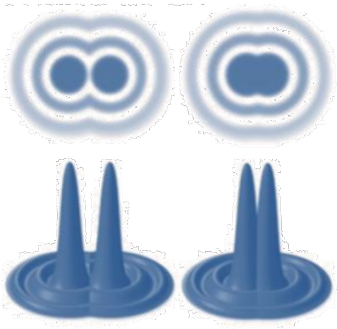


Figure 1.12: Left: Airy disks that can be distinguished. Right: Airy disks that cannot be distinguished from each other as they are below the Nyquist limit [32].

In order to determine the overall system's resolution, two other resolution have to be taken into account. The **Camera resolution** is the ability of the imaging device to resolve two points that are close together. It is defined through the Rayleigh Criterion (RC), implying whether or not two neighboring Airy disks (central bright spot of the diffraction pattern from a light source - see figure 1.13) can be distinguished from one another, determining the smallest point that can be resolved. The Nyquist limit marks that smallest point, because if the distance between two objects is greater than the Nyquist limit, a sensor can indeed distinguish between the two objects [7].

To conclude, the higher the resolution, the smaller the detail that can be resolved from an object. Still, it is schematic showing (fig. 1.13) that there needs to be at least one pixel width between two objects for the Nyquist limit to be overcome, allowing the two objects to be resolved. This is why smaller pixels provide higher resolution as they are able to distinguish between smaller objects.

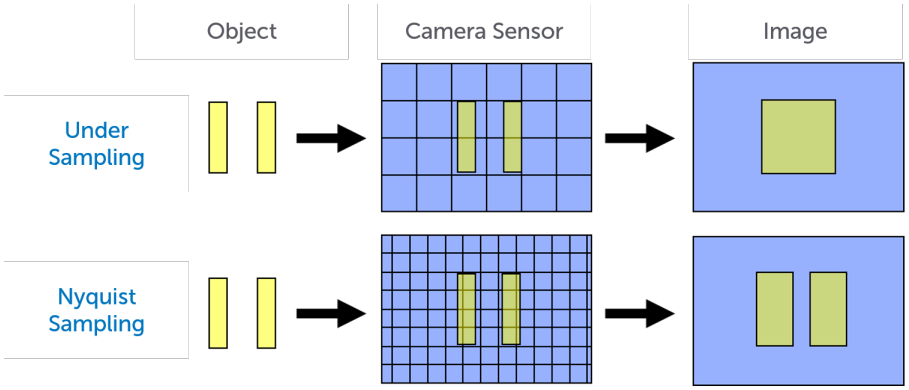


Figure 1.13: PixelSize-Sensor-Image correlation [32].

Every camera also consist of an included lens. With a similar concept to the camera resolution **Lens resolution** equally restrained by the concepts of the Rayleigh's Criterion. Here, the ability for a lens to resolve an object is limited by diffraction, created when the light emitted from an object travels through a lens aperture. Similar to the camera resolution, two different points on an object being imaged produce two different Airy

patterns. If the angular separation between the two points is greater than the angular radius of their Airy disk, the two objects can be resolved. If the angular separation on the contrary is smaller the two distinct points on the object merge [7].

Types of Sensors

A simple version of a sensor is the Photomultiplier Tube (PMT), which detects light by converting photons into an electrical signal, making it a device capable of sensing light. On the other side for higher sensitivity and lower noise there are more advanced sensors such as the Charge-Coupled Device (CCD) and Complementary Metal Oxide Semiconductor (CMOS) Sensor.

CCD

A CCD sensor transfers charges from the entire sensor surface to a single output node, hence the term "charge-coupled". Each pixel follows the same path to output its signal, which ensures a high level of image quality and uniformity, and is the core attribute of a CCD sensor. On the other, hand this results in a much more power-intensive readout process of a CCD sensor compared to a CMOS sensor.

CMOS

In contrast to a CCD sensor, a CMOS sensor contains transistors at each pixel, allowing the charge to be processed directly at the location, which requires less power, making CMOS sensors more energy-efficient. They are not just more energy-efficient but have the capability to read electrical charges at a much faster rate, on top of that additional features like noise reduction and image processing directly on the sensor can be included. As they share the fundamental structure of computer microprocessors, this enables cost-effective mass production, reducing the costs [7].

Types of Cameras

There are not only different sensors to choose from, but also different forms of camera optics. For this study the orthographic camera is being used. However there are also two other important forms of camera optics that will be mentioned for completeness.

Orthographic Camera

This camera is based on the concept of orthographic projection, which is a variant of parallel projection. It is defined through an affine transformation, entailing projection lines that are orthogonal to the projection plane, thus reducing a three-dimensional object to a two-dimensional representation. Because of this affine transformation an object's size in the rendered image stays constant regardless of its distance from the camera. Orthographic cameras are utilized in technical and engineering fields, including CAD (computer-aided design) and scientific visualization, where precise measurements and geometric accuracy

are important [24].

Projection Camera
In a projection camera, light from the scene passes through a lens and is projected onto a photosensitive surface, such as a film or digital sensor. The perspective bias of the image is based on the geometry of the lens and the position of objects relative to the camera. Projection cameras are employed in photography, videography, and conventional imaging applications that require accurate depiction of a scene's geometry and perspective [30].

Plenoptical Camera
The plenoptic camera is a multi-lens device that captures three-dimensional scene details. Similar to the structure of the compound eye of an insect, a plenoptic image consists of numerous partial images, each of which captures a different perspective of the scene. Resulting in facilitating the recovery of structures in any orientation as it can offer perspectives along both the vertical and horizontal axes. Plenoptic cameras are applied e for example in computational photography and 3D imaging, offering the capability to refocus images post-capture or to produce depth information from a single shot, which is highly advantageous [26].

1.2.4 Noise

Noise is a byproduct of irregular signal fluctuations that accompany a transmitted signal, increasing measurement uncertainty and limiting imaging resolution, speed and sensitivity [21]. It can be produced differently, either by the sensor, the electronics, temperature of the system or by fluctuation phenomena ([33], [6]).

Read noise (σ_R) refers to the accumulation of all noise produced by each system component during the conversion of the charge on each pixel into a signal. The lower the read noise, the easier it is to detect weak signals. It depends on the sensor design as well as the design of the camera electronics but is independent of signal level and temperature of the sensor.
Dark noise (σ_D) is caused by dark current, that flows even when the sensor is not exposed to light, thus no photons are incident on the camera. It is a thermal phenomenon resulting from electrons spontaneously generated within the silicon chip. The variation in the amount of dark electrons collected during the exposure is the dark shot noise. It is only dependent on the temperature and can be reduced by for example deep cooling of the camera. Dark noise is the most severe form if noise for in vivo imaging.
Photon shot noise (σ_S) is the statistical noise associated with the arrival of photons at the pixel, generated from the natural fluctuation of photons and emitted randomly. As it only depends on signal levels measured, it increases proportionally with the signal;

however, it becomes more apparent at lower signal levels.
Clock induced charge (CIC) (σ_C) is generated by the transfer of charge through the device, due to small probability ionization. CIC does not contribute much to overall noise but is evident in some specific forms of sensors.
Fixed pattern noise (σ_F) is caused by the variations in the responsivity of individual pixels on a sensor due to spatial non-uniformities of the pixels, and can be found within CMOS sensors for example.

Signal to Noise Ratio (SNR)

Signal-to-noise ratio compares the level of a desired signal to the level of background noise, the unwanted signal. More precise, it measures the ratio between an arbitrary signal level (not necessarily the most powerful signal possible) and noise. Hence it requires the selection of a representative or reference signal. Physically, it is defined as the ratio of signal power to the noise power while being measured in milliwatt (mW):

$$SNR_{mW} = \frac{P_{signal}}{P_{noise}} \tag{1.3}$$

Signal and noise power must be measured at equivalent positions within the same system bandwidth. If additionally the signal and the noise are measured across the same impedance, then the SNR can be obtained by calculating the square of the root mean square (RMS) amplitude [**p.1939**, 10]:

$$SNR_{mW} = \left(\frac{A_{signal}}{A_{noise}}\right)^2 \tag{1.4}$$

The ratio between, not a random signal level - as for measuring the SNR - but between the strongest undisturbed signal on a channel and the minimum identifiable signal is measured by the **dynamic range**. As many signals have a very wide dynamic range, SNRs are often expressed using the logarithmic decibel scale (see Tab. 1.1). Hence the SNR is measured in decibel-milliwatt (dBm):

$$SNR_{dB} = 10log_{10}(SNR) \tag{1.5}$$

A ratio higher than 1:1 indicates more signal than noise. Consequently the higher value of SNR the better the technical quality of the signal. The Rose criterion (named after Albert Rose) states that an SNR of at least 5dBm is needed to be able to distinguish image features at 100% certainty [**p.2007**, 10].

dBm	0	5	10	20	40
mW	1	3.1628	10	100	1000

Table 1.1: Comparison of SNR values in decibel-milliwatt (dBm) vs. milliwatt (mW)

1.2.5 Application of FMI in clinic and research

FMI takes along many advantages such as providing **cellular and molecular information**, **no radiation**, **highly reproducible and fast measurements**, **intuitive results**, **technical simplicity**, **low cost**, **high specificity** and **high sensitivity** ([2], [20], [49]).

Sensitivity describes the ability to detect contrast given a certain amount of probe. With regard to instrumentation, high sensitivity light detectors are capable of detecting very low levels of a light signal. They can differentiate the presence of an element above the background and noise. Applied to FMT, it means that FMT systems can detect even weak fluorescence signals. [29]. Specificity, on the other hand, characterizes the probe. High specificity, for example, refers to the ability to generate contrast where the concentration to be detected is present. FMT has a high specificity, although there is a fluorescence background signal due to the autofluorescence of the tissue.

FMI can monitor two-dimensional (2D) fluorescence signals by external laser irradiation of fluorescent molecular probes that are used to mark specific molecules or cells to monitor physiological and pathological changes in vivo. In contrary to PET, FMI does not quantify three-dimensional (3D) distribution information about internal fluorescent targets.

The three-dimensional extension of FMI is called fluorescence molecular tomography (FMT). It enables noninvasive **real-time** 3D visualization and quantitative analysis of different molecular processes in small animal studies in vivo. This technology seeks to reconstruct 3D spatial information and the concentration distribution of internal fluorescent targets. FMT has been extensively applied in preclinical diagnostics and various small animal model-based studies, including the early detection and diagnosis of tumors, the analysis of pathological tumor characteristics, and drug development [49].

1.2.6 Problems of Fluorescence Molecular Tomography

As seen, despite the significant potential of FMT in numerous practical applications, it continues to encounter substantial challenges in imaging biological tissues, primarily constrained by detection accuracy and imaging quality limitations [49]. Those challenges include poor penetration, scattering effects, depth-dependent signal, inability to use with larger species, and lack of detailed anatomical information.

Before explaining the origin of some of those challenges, two phenomena should be briefly

discussed beforehand. Theoretically fluorophores can repeatedly undergo the fluorescence process indefinitely, thereby producing signals multiply times, which enables for example real-time observation. Unfortunately due to the fluorophore's structural instability during the excited lifetime, continuous illumination generates reactive oxygen species (ROS), which damage the fluorophores, causing them to change their structure so that they can no longer fluoresce. That means even when the required light energy is supplied, the molecule are no longer put into an excited state. This is called **photobleaching** ([40], [14]). Similarly, continuous exposure to excitation light can induce a harmful another phenomena called **Phototoxicity**. Phototoxic effects include cell damage, cell death, or alteration of cellular processes [14]. It should be noted, though, that this is a greater problem in microscopy than it is in vivo imaging.

The wavelengths of the visible spectrum, which is applied in FMI, increase harmfully the effect of photobleaching as well as induce higher levels of phototoxicity.

Photon Propagation in biological tissues

Optical Molecular Imaging is based on the basis of **photon propagation in biological tissues**. The interaction between photons and biological tissues includes absorption, reflection, scattering, refraction and transmission (see Fig. 1.14).

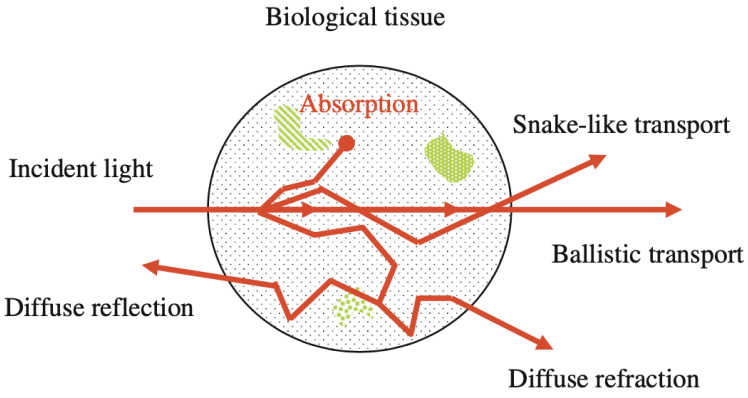


Figure 1.14: Several forms of photon propagation in biological tissues [20].

Out of all interactions, strong photon scattering predominates over absorption in biological tissues at near-infrared and visible wavelengths. This stronger scattering limits the penetration depth of excitation and emission light, leading to reduced image quality and resolution, particularly in thick or densely scattering samples. Penetration depth refers to the depth at which light can effectively penetrate into biological tissue [14]. In detail, due to scattering the number of excitation photons decreases, before they can be absorbed by the fluorophore. On the other hand, due to the scattering of emitted photons nonlinear changes in fluorescence signals are detected on the surface, manipulating the detection accuracy and presenting a serious challenge for applying the simplified linear

photon propagation model, which is usually used for simulating photon transport. In other words, there are several source, as well as light, constellations that might produce the same photon distribution at the surface. Especially if the signals originate from deeply seated fluorophore sources.

When this nonlinear relationship, caused by scattering, is combined with the process of recovering 3D data from 2D data for a FMT reconstruction, the **reconstruction process** turns out to be an **ill-posed problem**. This leads to the degradation of FMT image quality including spatial resolution, positioning accuracy and adequate recovery.

Reconstruction processes are linked to solving the **inverse problem** of photon propagation. In order to solve a system of equation adequately, only a certain number of unknown variables is allowed. Otherwise the problem is mathematically defined as **ill-conditioned**, making the solution process sensitive to noise and model errors, thus decreasing the system's robustness. Unfortunately this is the case for most fluorescence molecular imaging systems, as the diversity of fluorescent source distribution patterns, that need to be reconstructed (number of unknowns) is larger and more complex than the number of source–detector pairs (number of equations) resulting in an ambiguous mapping.

Summarized, the amount of light detected depends on wavelength of the excitation and emission wavelengths, depth in tissue and tissue type. Thus fluorescence molecular imaging faces significant limitations, described through the highly ill-posed and ill-conditioned inverse problem, attributed to multiple photon scatterings within heterogeneous biological tissues, along with the uncertainties associated with light propagation deep within the tissue [2].

The mathematical attributes regarding FMI will be discussed in the following chapter 1.2.7.

1.2.7 Mathematical Context

To provide a clear introduction to the mathematical context of Optical Molecular Imaging, let's start by examining a figure. Different optical modalities may have distinct physical imaging conditions, but they share the same mathematical context 1.15.

The **forward problem** is concerned with modeling photon propagation within biological tissues, while the **inverse problem** aims to reconstruct the internal source distribution from the signals detected on the external surface.

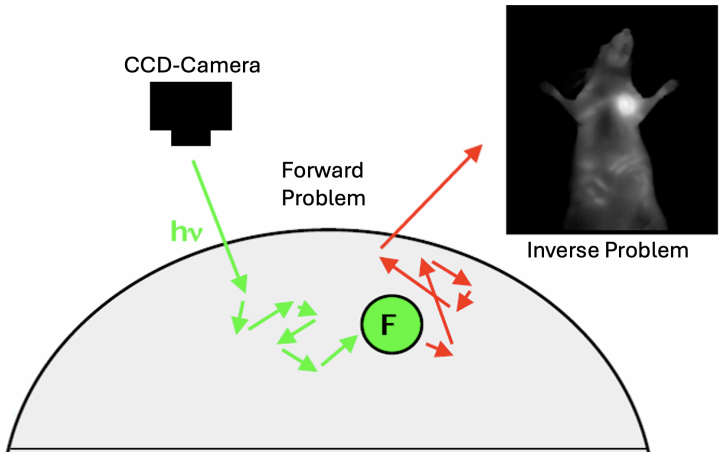


Figure 1.15: Concept diagram of Optical Molecular Imaging modalities [20]

.

Forward Problem

In order to approximate of photon propagation in biological tissues, the photon transport, also called the transfer equation, is firstly explained through the radiative transfer equation (RTE), also known as the Boltzmann equation:

$$(s \cdot \nabla + \mu_a(r) + \mu_s(r))L(r,s,t) = \mu_s(r) \int_\Omega f(s \cdot s`)L(r,s,t)ds` + Q(r,s,t) \tag{1.6}$$

where $s`$ represents the the solid Euler angle pointing in the direction of s, ∇ denotes the gradient operator, $L(r,s,t)$ is the radiance, which represents the photon flow energy per unit solid angle at point r, time t and direction s, $\mu_a(r)$ represents the absorption coefficient, $\mu_s(r)$ represents the scattering coefficient, $f(s \cdot s`)$ represents the scattering phase function, which indicates the probability of photons being scattered from $s`$ to s, and $Q(r,s,t)$ denotes the Emission, representing the fluorescent source term, which describes the spatial distribution of the fluorescent source [49].
When dealing with x-rays in CT imaging, the ratio between absorption and scattering changes, due to the decrease in scattering of x-ray photons. Hence the forward problem can be reduced to

$$(s \cdot \nabla + \mu_a(r))L(r,s,t) = 0 \tag{1.7}$$

which makes solving the RTE a much simpler task. In regards of optical tomography this reduction is not possible, therefore a more complex sequence of steps need to be conducted to solve the RTE. By approximating the RTE to the **diffusion equation (DE)** and introducing **boundary condition**, the **forward problem** can be **discretised** [49].

Diffusion Equation

The diffusion equation is obtained based a simplified first- order diffusion approximation model of the RTE and describes the the photon propagation process as followed:

$$\nabla(D_x(r)\nabla\Phi_x(r)) - \mu_{ax}(r)\Phi_x(r) = -\delta(r - r_s) \qquad \forall r \in \Omega \tag{1.8}$$

$$\nabla(D_m(r)\nabla\Phi_m(r)) - \mu_{am}(r)\Phi_m(r) = -\Phi_x(r)x \qquad \forall r \in \Omega \tag{1.9}$$

with Ω denoting the solid angle, x and m represent the excitation and emission wavelengths, $\delta(r - r_s)$ represents the excitation light source, $\Phi_x(r)$ and $\Phi_m(r)$ denote the optical field intensity of excitation and fluorescence, $\mu_{ax}(r)$ and $\mu_{am}(r)$ denote the absorption coefficient of excitation and fluorescence in biological tissues, $\mu_{af}(r)$ denotes the absorption coefficient of fluorophores, $D_x(r)$ and $D_m(r)$ denote the diffusion coefficient of excitation and fluorescence. Lastly x is the fluorescent source distribution to be reconstructed [49].

Boundary Conditions

According to the theory of differential equations, the solution of 1.9 requires an appropriate boundary condition, such as the Dirichlet boundary condition for the coupled diffusion equation:

$$\Phi_{x,m}(r) + 2\rho D_{x,m}(r)\frac{\partial\Phi_{x,m}(r)}{\partial n} = 0 \qquad \forall r \in \partial\Omega \tag{1.10}$$

Here $\partial\Omega$ denotes the boundary of Ω, n denotes the outward normal of the boundary $\partial\Omega$ and ρ is the boundary mismatch parameter and accounts for the light reflection on the boundary surface.

Discretization of the Forward Problem

In Optical Molecular Imaging, the solution of the forward problem is the basis for the inverse problem. In order to study the inverse problem, the forward problem of the diffusion equation needs to be solved. The methods for solving the diffusion equation can be classified into three categories: **analytical**, **statistical** and **numerical methods**. Important for this project, as it forms the foundation of the simulation software "Lipros," is the **Monte Carlo** (MC) method, which is most commonly used to simulate photon propagation in biological tissues, and for obtaining significant physical information from biological tissues [20].

Therefore, the forward problem can be discretized into a simplified linear problem by

omitting the nonlinear excitation and emission processes of fluorophores, and solving 1.9 and 1.2.7 as followed:

$$\Phi = Wx \tag{1.11}$$

where Φ is the fluorescence signal intensity obtained from tissue surface measurement, x is the fluorescent source distribution to be reconstructed and W is the physical equation describing the relationship between Φ and x, also described as weight matrix [49].

Inverse Problem

The inverse problem aims to obtain the 3D distribution of fluorescent targets based on the mathematical model of photon propagation constructed in the forward problem.

The inverse problem of FMT focuses on reconstructing fluorescent targets, by recovering the fluorescent distribution x, as per equation 1.2.7. This can be solved by inverting the weight matrix W.

$$x = W^{-1}\Phi \tag{1.12}$$

where $W \in R^{s\times t}$ represents the system matrix, s denotes the number of surface fluorescence measurements and t denotes the number of internal nodes. $x \in R^t$ denotes the unknown vector of discrete node and $\Phi \in R^s$ represents the surface fluorescence measurements. This inversion often presents ill-posed attributes due to the typically lower dimension of Φ compared to x ($s < t$), resulting in W not being a square matrix and the null space of W not being zero-dimensional. Moreover, equation (1.2.7) is also ill-conditioned due to the large number of conditions, which contains a larger amount of unknown parameters compared to the number of equations making the reconstructed images incredibly susceptible to noise and errors [49].

In summary, the inverse problem of FMI is inherently ill-posed and ill-conditioned, primarily because of the significant scattering coefficient of optical photons in biological tissue. Addressing the ill-posed and ill-conditioned nature of the inverse problem is the most critical issue in attaining meaningful solutions. As a result, there is currently no exact mathematical solution without resorting to some form of approximation, prior assumption, or regularization. This sets it apart from imaging modalities based on high-energy photons like PET or SPECT, where due to the high-energy photons, the scattering is reduced and the emission data model can be utilized to solve the transport problem. Mathematical solution for solving the inverse problem by either using some kind of approximation, prior assumption or regularisation will be analysed in the next section.

1.2.8 Inverse Problem Solving - Different Approaches

Several efforts have been made to effectively reconstruct internal source distribution from the signal detected on the external surface, by addressing the above-mentioned challenges.

The main fields of possible approaches are the **iterative solution algorithms**, **reconstruction methods**, **regularization methods** and application of **deep learning technology** for tackling the inverse problem thus for FMT reconstruction [20].

Iterative Solution Algorithms

In order to continuously improve efficiency and accuracy for reconstruction, iterative solution methods can help to gather meaningful solutions. Optimization algorithms such as the conjugated gradient method, Gauss–Newton method, interior-point method and augmented Lagrangian method can be used ([20], [2]).

Reconstruction Methods

Next to the iterative classes of reconstruction algorithms, the most common analytical reconstruction method is called Filtered Back Projection (FBP). FBP is especially considered as a standard CT reconstruction algorithm, that will also be used for this work. For better overall understanding of the entire process of applying the forward and inverse problem to an actual CT-Scan process, the different steps will be explained in the following. When source and detector are rotating through 180 around an object, they plot the signal recorded by a line of voxels thus create a graph called **sinogram**. Differently phrased, a sinogram is a collection of projections at several angles, which can be understood as a linear transform of the original image. This two-dimensional sinogram needs to be transformed into a for human clearly readable form, which is called a CT-scan or projection but might also be referred to as CT-image in the following. The transformation from sinogram to CT-scan is called reconstruction process and more precisely is done through a back projection reconstruction algorithm.

To better understand this process, considering a one cross-sectional slice (Fig. 1.16a), recorded by a row of pixels on the detector, the attenuation can be represented as a line profile. For a given projection angle, θ, each pixel on the detector sums the X-ray photons passing through the specimen slice along a given beam path. Figure 1.16b shows three projection angles and the corresponding projection (a yellow line profile) of absorptivity. The accumulation of those projection lines of absoptivity form a sinogram (Fig. 1.16c), showing the variation in attenuation across the row of pixels as the projection angle varies.

In figure 1.16d the **backprojection reconstruction algorithm** takes each projection -

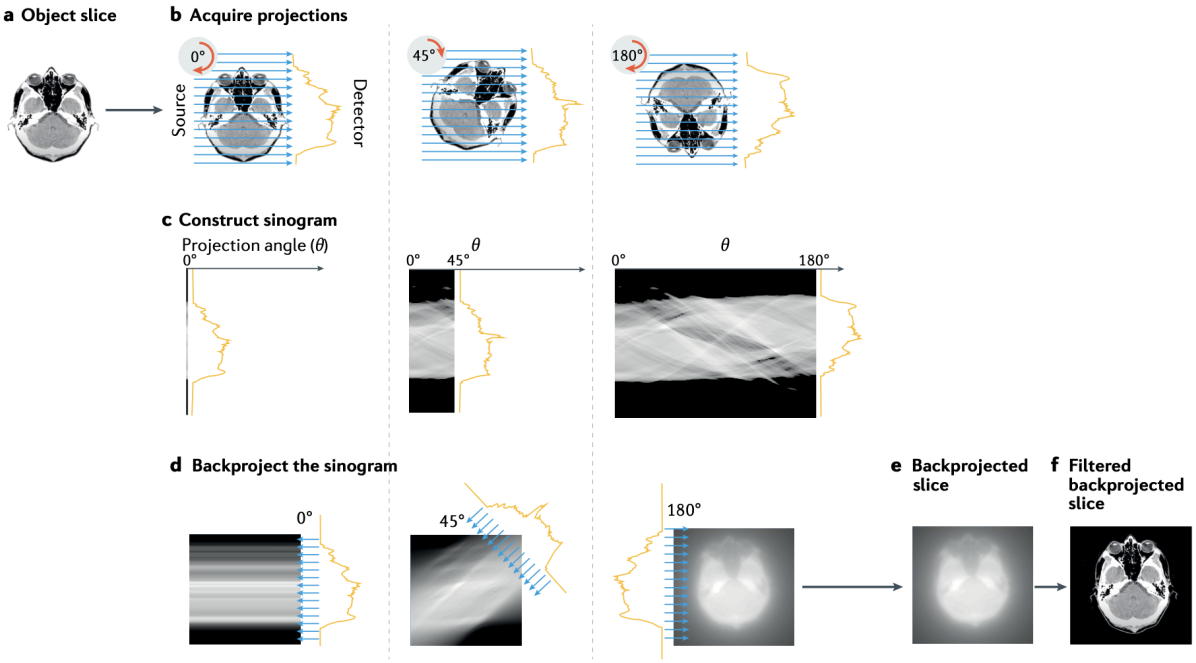


Figure 1.16: Backprojection reconstruction method for a single slice obtained by parallel beam Computed Tomography [46]. In this study, parallel beam CT is employed instead of the more commonly used cone-beam CT.

each attenuation profile - making up the sinogram and mathematically projects it back along the angle θ at which it was recorded. Thereby, the attenuation value in the sinogram is divided by the number of image pixels along the direction of the projection from source to detector, and the average attenuation value obtained is assigned to these pixels. If the pixel values were added up along the projection direction, the original attenuation value in the sinogram could be reconstructed. This process is repeated for each gantry angle. The sum of all the back-projected attenuation profiles describes the final back-projected image [36].

More theoretically explained, its mathematical foundation is called Fourier slice theorem (see Fig. 1.17). It uses Fourier transform of the projection and interpolation in Fourier space to obtain the 2D Fourier transform of the image, which is then inverted to form the reconstructed image.
The quality of the final images does depend on the amount of projections given as this is resulting in an increasing number of line profiles backprojected.

Nevertheless while comparing figure 1.16a with figure 1.16e it is evident that backprojection distributes mass incorrectly, introducing blurring into the reconstruction. Another step, called **Filtered backprojection** corrects this blurring by applying a filter (most commonly a ramp filter) to the projections. In order to compensate for the high-frequency components in Fourier space that are missing, the ramp filter is a mathematical function that suppresses low spatial frequency components of the attenuation profiles. On the

other side, this filter also potentially enhances image noise. Nevertheless applying the FBP leads to a sharper final image (Fig. 1.16f) [46].

The just discussed relationship between the slices and projections can be described mathematically for example by the **Radon transform**. Which is an integral transform that projects a cross-sectional slice along a given direction to give the one-dimensional profile. In x-ray Computed Tomography, this one-dimensional profile is equivalent to the sinogram [46].

As the inverse Radon transform reconstructs the object from a set of projections, the (forward) Radon transform can be used to simulate a tomography experiment. And that is exactly what can be observed in figure 1.16 and also will be an essential tool of this work, an application of the Radon transform to simulate a tomography experiment and reconstructs the input image based on the resulting sinogram formed by the simulation [39].

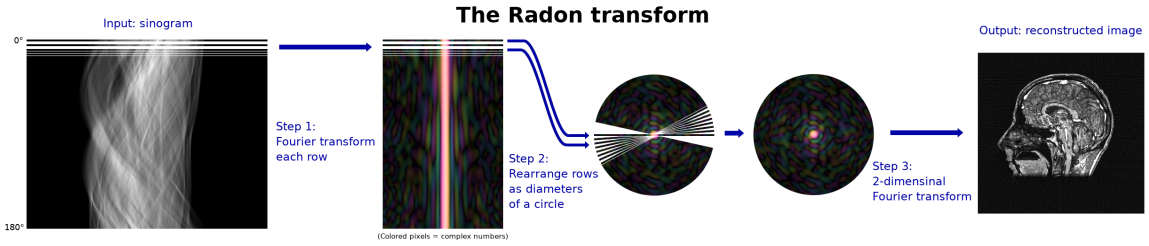


Figure 1.17: Radon transform via Fourier transform [44].

Regularization Methods

Numerical solutions to the inverse problem are for example the singular value decomposition (SVD) methods, the least squares methods and the **regularization-based methods** [49].

The main concept of the regularization method is to introduce initial knowledge into the inverse problem in the form of a penalty function so as to constrain the solution process. The challenge here is to introduce the right kind of prior knowledge and how to describe them mathematically [49]. The underlying principle is to either reduce the number of unknown variables or increase the amount of measurements [2].

Based on the regularization method, the solution for 1.2.7 determines source power density x^* by minimizing the following objective function as followed [49]:

$$x^* = min_x(||\Phi - Wx||_2^2 + \lambda T(x)) \tag{1.13}$$

where $T(x)$ is the regularization term and λ is the regularization parameter.

Often Tikhonov regularization for example is used to improve reconstruction accuracy. In order to enhance the reconstruction efficiency on the other side, sparse-promoting regularization (L0, L1, L2) and total variation regularization (TV) have been introduced ([49], [20], [2]).

Deep Learning Technology

In recent years various deep learning technology in FMT reconstruction has been implemented and proven themselves as impressive reconstruction performances. Shown deep neural network-based (DNN-based) approaches focus especially on **image restoration** and **inverse problem solving** [49]. so far, there are two deep learning methods for solving the FMT inverse problem, the first is called **postprocessing** and the second is called **end-to-end deep neural network (ETE-DNN)** method.

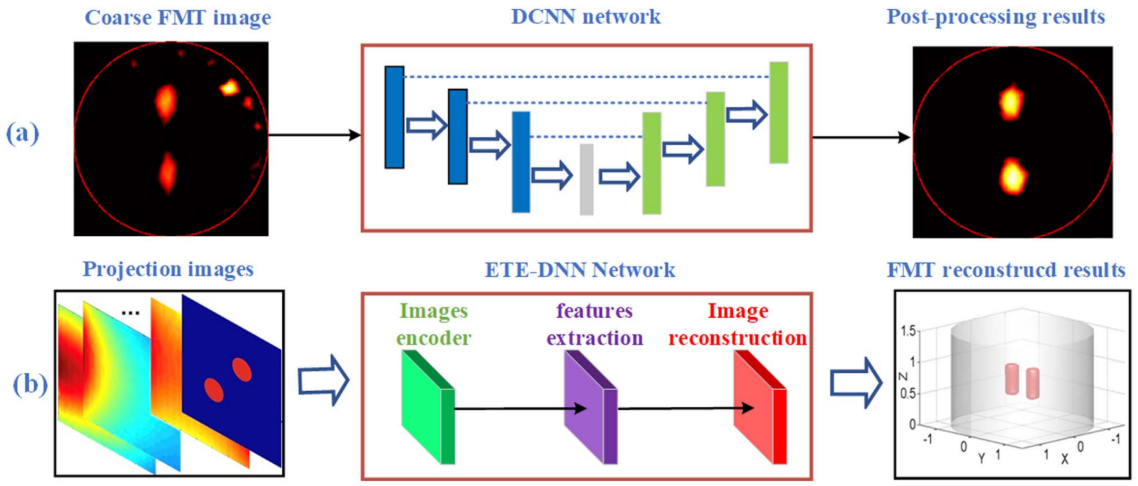


Figure 1.18: Different DNN-based methods for FMT reconstruction: (a) post-processing (b) end-to-end deep neural networ (ETE-DNN) [49]

ETE-DNN Method

The ETE-DNN method establishes a direct nonlinear mapping relationship from surface fluorescence measurements to interior fluorescence distribution. Hence it adopts a large amount of data to learn and seek the unknown solutions of the inverse problem. It achieves reconstruction by encoding the projection images and extracting relevant features (see figure 1.18b)).

Post-processing methods based on DNNs

Since reconstructed images are often influenced by artifacts, resulting in a biased estimation of the ground truth, a CNN can be employed for post-processing. It reduces errors between the reconstructed images and the actual FMT images, thus enhancing the quality of the FMT images.(see figure 1.18a)). Usually the core concept specifies at first the obtainment of preliminary reconstruction images and secondly the adoption of advanced

deep network models (e.g. CNN) to further optimize the preliminary reconstruction results.

To illustrate this important concept, let's consider Long's work (Long, 2018 [22]), where the reconstruction process is broadly split into two phases. Initially, the conventional Tikhonov regularization method is applied to acquire provisional reconstruction outcomes. Afterwards a CNN, usually the architecture of U-net, further optimizes the reconstruction results of the first stage.

Instead of splitting the process into two phases, it can also be split into two sub-networks, one being a fully connected (FC) and the other one a local connection (LC) network. Latter otimizes the preliminary reconstruction results to further improve the morphological reconstruction of FMT, whereas the initial network obtains preliminary reconstruction results. This technology is called "K-nearest neighbor–local connection (KNN-LC) network model " [49].

Most importantly, as of state of knowledge right now, these approaches are only applicable to reconstructions within homogeneous tissue models, posing challenges for reconstructing fluorescent targets within heterogeneous tissues.

Until now, the primary focus of most DNN approaches has been to enhance the reconstruction algorithm from FMT sinograms to the object phantom itself. This has been achieved by applying various techniques to the input of FMT sinograms, with the final object slices being the direct output. However, this thesis aims to challenge this conventional approach and propose an alternative solution. Before delving into this new approach, it is essential to discuss some theoretical basics.

1.3 Artificial Neural Networks

Introduction to the concepts of Artificial Intelligence

Artificial intelligence (AI), can be understood as technology that allows any kind of machine to simulate human intelligence and problem-solving capabilities. An absolute clear definition is difficult to offer, as the term AI is still used quite loosely. Machine Learning (ML) and its subcategory Deep Learning (DL) can be seen as a part of Artificial intelligence. Many popular algorithms in AI are based on Neural Networks, whose structure is inspired by the structure of the neurons in the brain. Those algorithms allow to "learn" from available data by enhancing their accuracy in classifications or predictions progressively. For validation new data is used to assess the trained algorithm's performance.

The "learning"-period is called training phase, during which the training data set is preprocessed and meaningful features are extracted. Preprocessing tries to put the available data into a clean format, by e.g. performing noise reduction or image repair. Feature extraction is a crucial step in building a good model (i.e. AI algorithm). Information is aggregated from the preprocessed dataset in a smart way, to make it as easy as possible for the model to make a decision (e.g. **classification**). $\hat{f}$ is a way of representing the AI algorithm (e.g. a **classifier function**). ($\hat{f}$) is able to extract a distinctive and complete **classification results** (i.e. feature representation), which can be defined mathematically as $\hat{y}$. Those features, influencing the classification results, differ according to each individual application and are "hand-crafted" for every new task. Mathematically those different features can be represented in terms of the **feature vector** $x \in R_n$. The classifier ($\hat{f}$) itself has its own parameters, represented as the **classifier's parameter** vector θ, representing the set of hyperparameters required for training and applying the method. θ is fitted during the training phase and later evaluated on an independent test data set. All previously mentioned quantities can be summarized with the following equation, describing the core principle, that every model in ML (i.e. AI algorithms) follows.

$$\hat{f}_\theta(x) = \hat{y} \tag{1.14}$$

1.3.1 Neural Network

Neural networks describe a certain type of ML model (AI algorithm). They can be an example for a previously described classifier. The internal structure of a neural network, consists of many similar building blocks. The core unit of those components is called a **neuron**. A neuron itself is already capable of classification. Originating from 1.14 the explicit expression is:

$$\hat{f}_\theta(x) = h(w^T x + w_0) \tag{1.15}$$

Where the bias term $w_0 \in \mathbb{R}$ and the weight vector $w^T = (w_1, ..., w_n) \in \mathbb{R}^n$ are adjustable parameters, which are jointly represented by the classification's parameter $\theta 0(w_0, w_1, ..., w_n) \in \mathbb{R}^{n+1}$. w_0 is called the bias term. Further, $h(x)$ is the so-called activation function. In theory this can be any non-linear function, and serves the purpose to introduce non-linearity into an otherwise completely linear neuron.

Activation functions

Without the restriction of non-linearity of the activation function, the network would be merely a sequence of linear transformations, not being able to approximate arbitrary smooth functions with the network [8]. There is a huge variety of functions that can be chosen from, as different functions are expedient for different applications. Classical activations are $sign(x)$, $\sigma(x)$, and $tanh(x)$ but as lot of research is conducted on implementing more suitable functions there are many other functions that turned out to be useful to train Neural Network, such as [8]:

Rectified Linear Unit	The ReLU vanishes for all negative numbers, while it is a linear growing function for positive numbers.
Sigmoid	The sigmoid function is a smoothed version of the step function.
Hyperbolic Tangent	The hyperbolic tangent has similar behavior to sigmoid, but can take both positive and negative values.
Softmax	The softmax function is commonly used for the last layer in a neural network for classification tasks

For example is the mathematical definition [23] of the Rectified Linear Unit function, shown in figure 1.19, described as:

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \mathbf{0}, & else \end{cases} \tag{1.16}$$

While a slight modification of latter results in the *Leaky ReLU*:

$$LReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \boldsymbol{\alpha x}, & else \end{cases} \tag{1.17}$$

Both functions are especially important for this work, as ReLU is being used for a special network architecture called U-Net and LReLu was chosen for a modification of U-Net called nnU-Net, which is the out-of-box-network chosen for this work.

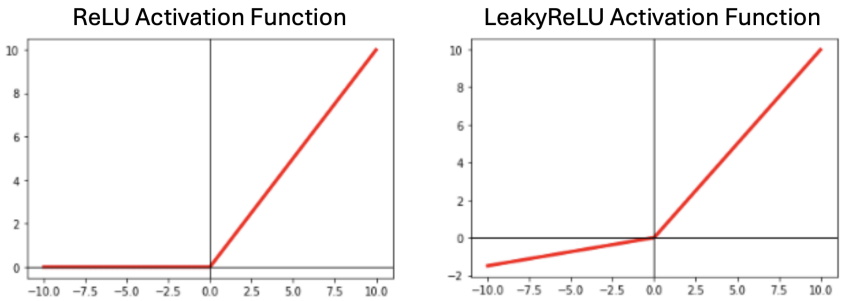


Figure 1.19: Plot of different activation functions [19].

Single Layer Network

A single Layer of a Neural Network of size N is simply a linear combination of the output of N different neurons.

$$\hat{f}(x) = \sum_{i=0}^{N-1} v_i h(w_i^T x + w_{0,i}) \tag{1.18}$$

this time introducing combination weights $v_i \in \mathbb{R}$ for $i = 0, ..., N-1$. v_i is also part of θ.

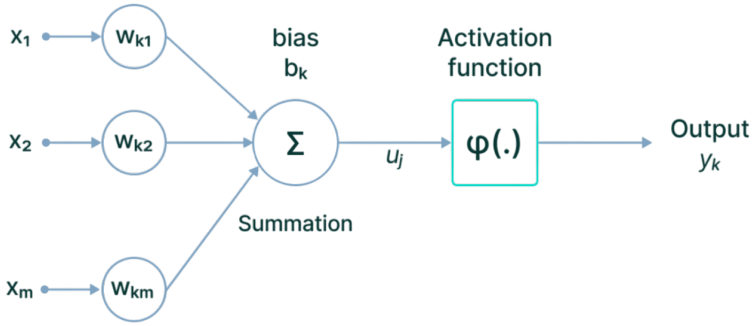


Figure 1.20: Description of the k-th neuron, representing the possible structure of a Single Layer Network [5].

Looking at the classifier function $\hat{f}$ as an approximation of the true unknown function $f(x)$, $\epsilon > 0$ can be used to quantify their divergence:

$$\left| f(x) - \hat{f}(x) \right| \leq \epsilon \qquad \forall x \tag{1.19}$$

It is well known that for increasing N, ϵ can be reduced [23].

This leads to the conclusion that the more neurons implemented, the better the approximating model will be.

Multilayer Network

Instead of accumulating many neurons parallel to each other in one simple layer, they can also be placed in different layers, that are connected by taking the outputs of neurons from the previous layer as inputs for the neurons of the current layer. Notice that this added inter-connectivity can greatly increase the complexity of such a model. Creating such networks, that consist of multiply layers stacked on top of each other, the field of **Deep Learning** is introduced. In Deep Learning, approaches are explored which are based on neural networks with several such stacked layers. Conceptually, each output/input

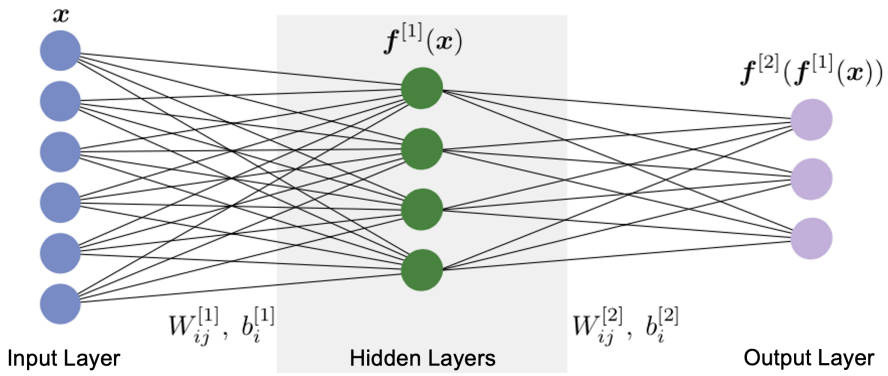


Figure 1.21: Structure of Multilayer Network [8].

component can be viewed as a neuron (but without weights). That way the input (i.e. feature vector x) and output (i.e. classification results $\hat{y}$) each make up their own layer. All layers in between are referred to as hidden layers. Modifying the weights of each neuron in the hidden layers can improve the prediction/classification performance of the neural network as a whole, for example when analysing an image.

In a basic hidden layer, called **fully connected layer**, each input neuron is connected to every neuron in the next layer, which means that the network does not know which two neurons (possibly representing pixels in the image) are related (e.g. adjacent). In this case important information such as local correlations or geometric structures are lost. To retain this information, a so-called **convolutional layer** can be used.

Convolution

The basic idea behind a convolutional layer is that local patterns of data are written as weights within a **feature-map**. Hence, they are stored in a **filter**. The convolutional layer matches these filter patterns again with a certain region of the input data, restoring the information. The matching of the convolutional layer with the filter patterns is mathematically described as **Convolution** of the filter f with the input data x (which is

also the origin of the layer's name):

$$(f * x)(t) = \int f(a)x(t-a)da \tag{1.20}$$

Thanks to the introduction of a feature-map, **convolution layers** only consider a local neighborhood of an input component (e.g. pixel) as an input for each neuron, saving the same weights (i.e. "fitters") across all neurons of the same layer in the feature-map (1.22(left)), which reduces the amount of parameters and therefore memory required to store such a layer - which is an intended and welcoming attribute, since images often correspond to very high dimension/data which can make the number of parameters explode [23].

Networks that are based on such convolution layers are called "Convolutional Neural Networks" (CNN). In typical CNNs, multiple filters are used simultaneously and independently in each layer, with each filter automatically specializing in extracting certain features. For example, one filter may aim for edge detection, while another may be sensitive to the brightness of an area [8].

Applying a convolutional layer will often reduce the dimension of its input. This might involuntarily lead to information loss. To prevent this, so-called **downsampling** reduces spatial as well as pixel-level information, as the network is applying convolutional layers. Downsampling one the other hand is not neccesarily always desired. An immediate action while applying the convolutional layers would be to add zeros around the edge of the data so that the feature map has the same size as the input data. This process is called **"padding"**. As another option, image enhancement can be performed afterwards by expanding, or **upsampling**, the feature map, in order to create it back to the shape of the input image.

Pooling

Pooling is a way of reducing the dimension of the data. It summarised the output of several neurons into one singular output (e.g. by taking its maximum), passing forward only the most important information (1.22(left)). In case of a CNN this is done separately for each feature map which additionally can help to reduce the spatial size of the input.

The determination of what information is considered important may vary from case to case: **Max-pooling** for example chooses the element with maximal value in a region to be analysed. Its output can result in a feature map that contains the most distinctive features from the previous feature map [43].

Summary

Most of the previously described attributes are core pieces of a Convolutional Neural Network, and can be otherwise also found in various constellations in different networks (see Fig. 1.22 (right)). Ending this section, it is worth emphasizing that any distribution of tasks between the filters and layers occurs automatically and is not predetermined by the network structure. It can vary from application to application, from training to training. This is exactly what makes the functioning of neural networks so appealing for research and fascinating for application.

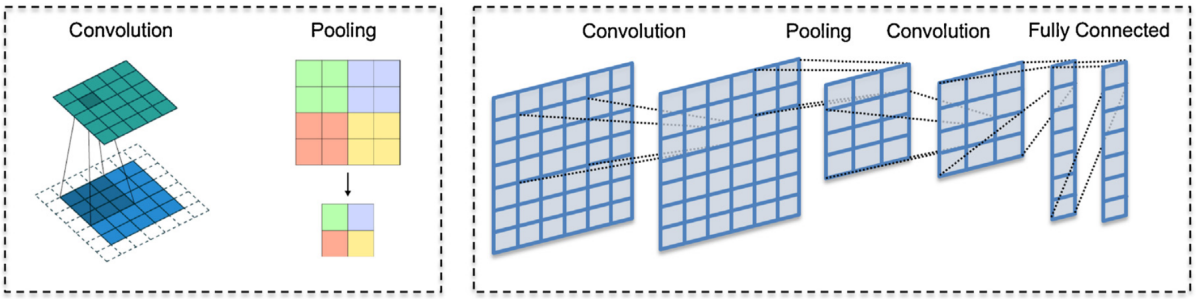


Figure 1.22: Convolutional layers only face a restricted field and all neurons share the same weight. Pooling layers reduce the total input size. Both are typically combined in an alternating manner to construct convolutional neural networks (CNNs) - on the right [23].

1.3.2 Semantic Segmentation

In medical image segmentation, it is the aim to determine the classifications of an organ or anatomical structure or even a source distribution as accurately as possible. Simultaneously through the adaptation of ML, image segmentation and the preparation of segmentation maps play an important role in training computers to recognize structures such as landscapes, photos of people, as well as medical images.

Hence looking at medical image segmentation in regard of deep learning is an algorithm that assigns labels to pixels. More specific, it identifies collections of pixels and classifies them regarding different characteristics such as color, contrast, placement within the image and other attributes. Pixels associated with a specific class are categorized solely based on that class without considering any additional information or context [43]. Other than image classification, segmentation is not understanding what information the image contains but lets the machine identify the precise locations of different kinds of visual information including their local boundaries.

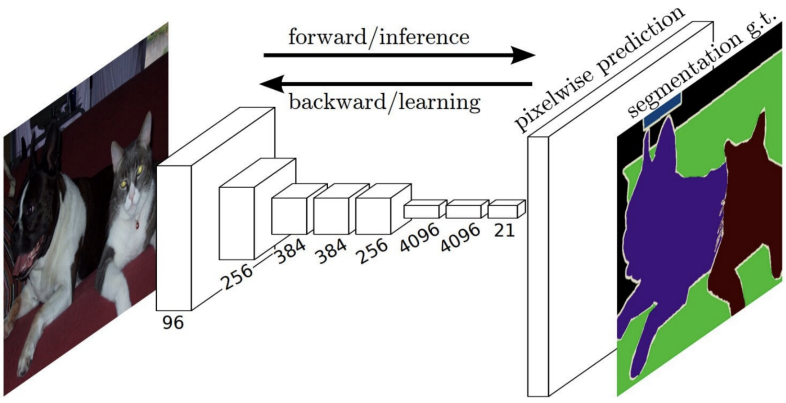


Figure 1.23: Concept of an image segmentation deep learning [25].

DeepLearning based Image Segmentation Models

Fully convolutional networks (FCNs)

FCN is a type of network that only consists of convolutional layers. More than that, next to convolution it takes advantage of pooling and upsampling layers, instead of using flat, dense layers. This makes the networks easier to train and enables possibility for applications that take an image of arbitrary size and produce a segmentation map of the same size [25].

Encoder-Decoder models

The Encoder-Decoder model, as illustrated in figure 1.24, is based on a two-stage concept, of reducing the dimension of the input, and then trying to reconstruct it. First stage, the encoder, is represented by an encoding function $z = f(x)$, similarly the decoder of the second stage is represented by the decoding function $y = g(z)$. z compresses the input into a latent-space representation, whereas y aims to predict the output from the latent space representation. The latent or feature representation itself is a vector, that captures the underlying semantic information of the input that is useful for predicting the output [25].

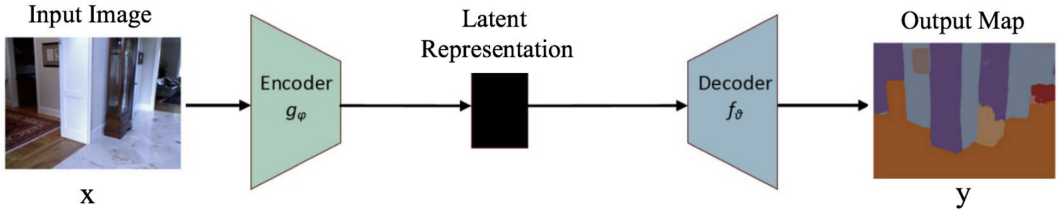


Figure 1.24: Architecture of an encoder-decoder model [25].

U-Net

U-Net is a model initially developed for medical image segmentation, which combines the concepts of FCNs and the encoder-decoder model. Their network and training strategy relies on the use of data augmentation to learn from the available annotated images more effectively ([35], [25]).

As seen in figure 1.25, the U-Net architecture is based on two parts. While the encoder stacks convolutional layers that are downsampling the image to extract information from it, the decoder rebuilds the image features using the process of deconvolution.

The U-Net architecture is primarily used in the medical field to identify cancerous and non-cancerous tumors in the lungs and brain [25].

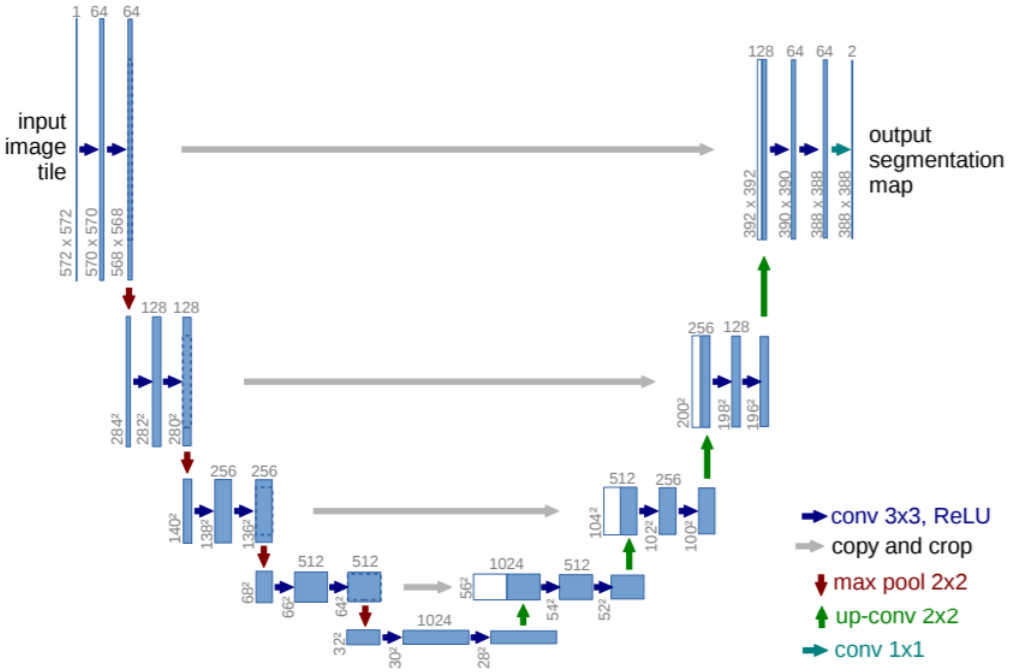


Figure 1.25: Architecture of the U-Net [35].

1.3.3 Supervised Learning

Learning tasks for deep learning can be split into two categories. On one hand, the tasks that involve datasets made up of input-output pairs or data-label pairs. On the other hand, there are tasks where information needs to be extracted from unlabeled data [**p.47**, 8]. The latter is called **unsupervised learning**, which will not be explained further in this work. The former concept is called **supervised learning** and represents the learning concept used in this work.

Configuration of supervised learning

As already discussed in chapter 1.3 a basic segmentation algorithm can be formalized as a function

$$f_\theta(x) = \hat{y} \tag{1.21}$$

where x represents an image, $\hat{y}$ denotes the corresponding predicted segmentation, and θ represents the set of hyperparameters adopted during training. Similar to the classification function $\hat{f}$ in chapter 1.3.1, the results of the classification $\hat{y}$ - representing the features - has to be seen only as an approximation of the true value y, also described as ground-truth output.

Training is about determining good values for the elements of the set of parameters (θ). Technically there can be many "good" configurations of θ. A function that measures the quality of θ (i.e. the quality of the predictions) needs to be defined. This function is called the loss function $L(\theta)$.

Loss Function

As $\hat{f}$ estimates the true classifier function f, $\hat{y}$ estimates the true class labels y. The loss function quantifies the discrepancy between $\hat{y}$ and y, as the goal of the learning process is to minimize the difference between $\hat{y}$ and y for all tuples in the data set by adjusting the parameters of θ. Thus, $L(\theta)$ needs to be minimized with respect to θ.

$$\theta_{best} \in \operatorname{argmin} L_\theta(\hat{y}, y) \tag{1.22}$$

As $L(\theta)$ is minimized, the quality of parameters is high if their loss is low. Similar to the activation functions, depending on the task, there are various loss functions to choose from. There are loss functions for regression or for deep learning for different fields such as object detection, face recognition or classification cases [42]. A simple example for a single sample is e.g. the "Square loss":

$$L(y, \hat{y}) = (y - \hat{y})^2 \tag{1.23}$$

or Absolute Loss:

$$L(y, \hat{y}) = ||y - \hat{y}|| \tag{1.24}$$

As segmentation is part of classification cases, the cross entropy loss function [13] is usually used for training multi-label classification models, and is therefore important to mention, here as the loss for N individual samples $(y_1, \hat{y}_1), ..., (y_N, \hat{y}_N)$:

$$L_{binary} = -\frac{1}{N}\sum_{i=1}^{N}(y_i log\hat{y}_i + (1 - y_i)log(1 - \hat{y}_i)) \tag{1.25}$$

Gradient Descent

The iterative method called **gradient descent** is used to minimize $L(\theta)$ [**p. 31-32**, 8]. Mathematically the gradient shows the direction of the steepest positive slope (i.e. the negative gradient points in the direction of steepest descent) of a function and is therefore a powerful tool to use in order to find the minimum of the loss function. The gradient, a derivatives of the loss function with respect to weights and biases (i.e. θ) is used to determine the direction of descending down the parameter landscape, taking small steps to find a minimum (Fig. 1.26(b)).

$$\theta_{new} \rightarrow \theta_{old} - \eta\frac{\partial L(\theta)}{\partial \theta_{old}} \tag{1.26}$$

The parameter $\eta > 0$ is called the learning rate and controls the step size in the parameter space. If chosen too small at the beginning of optimization, a local minimum instead of the global minimum might be reached (Fig. 1.26(a)) or if chosen too large it might jump across a minimum (Fig. 1.26(c)). It is never guaranteed that the global minimum is found, sometimes even neither a local minimum. To efficiently calculate the gradient a special algorithm needs to be applied. For example the **back-propagation algorithm** which is commonly used [23].

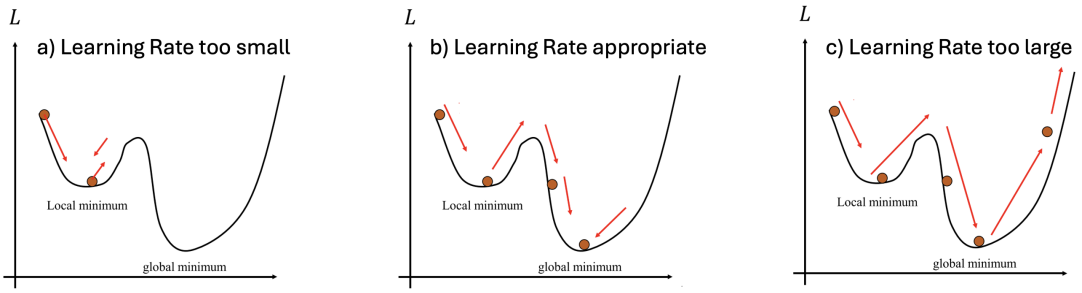


Figure 1.26: Different settings of the Learning Rate [41].

As the gradient descent is based on an iterative algorithm, the network parameter have to be updated iteratively. After the iterative process is over, the network needs to be

evaluated. Applying this to the actual network architecture works as followed. Typically, the entire training data is divided into batches, and then the network is evaluated for each batch. The random subdivision of the training data into batches, where larger batches lead to greater training efficiency but also higher computational effort, is maintained for several iteration steps before being reselected. The successive iteration steps with a fixed batch division are referred to as one epoch [**p. 33f**, 8].

Chapter 2

Methods and Tools

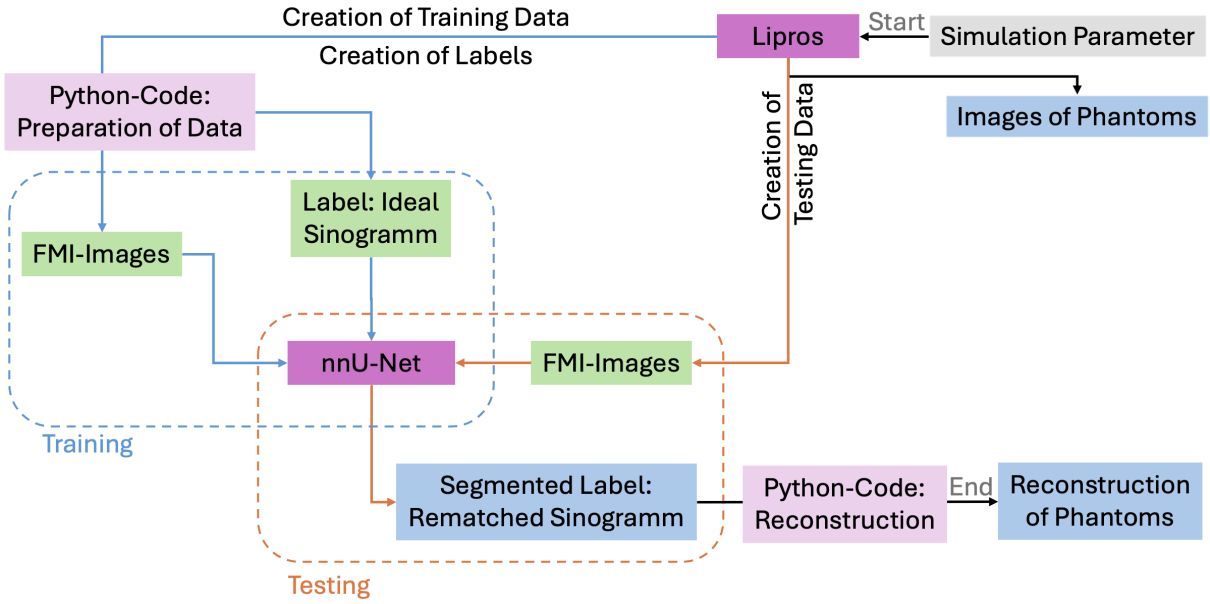


Figure 2.1: Core-Workflow-Chart with essential components of the conducted research. Tools and codes are in shades of purple, input data in light green and output data (sinogram or phantom) in bright blue

To gain an understanding of the concepts in this work, one can break down the workflow into three stages: simulation of essential data, training and testing of the neural network, and ultimately, reconstructing, analysing, and refining the method. Figure 2.1 can offer guidance through the structural explanation. The core idea of this work is to enhance in vivo fluorescence optical tomography through semantic segmentation as an alternative to resolving the ill-posed inverse problem. This involves preparing, training, and validating the compatibility of augmenting FMI images, which depict an incomplete sinogram riddled with artifacts, using a trained neural network. The objective is to obtain a completed sinogram devoid of artifacts, facilitating straightforward image reconstruction through the standard Filtered Back Projection technique.

Initially, the FMI data intended for training and testing purposes needs to be generated. Given the focus on maximizing trial diversity to train successfully and to properly test the efficacy of the proposed method, it was determined that all required data would be simulated. Actual user cases may be considered in the future, once the method is fully developed.

For any supervised neural network, the training data comprises specific images, each paired with a corresponding label representing the desired outcome. Throughout the research, the type of both the training and testing images remain consistent, consisting of Fluorescence Molecular Imaging (FMI) induced light fluence simulations. However, the labels are derived from either Bioluminescence Imaging (BLI) induced light fluence simulations representing **Ideal-Source-Projections (ISP)** without attenuation and scattering or Computed Tomography (CT) simulated data.

All simulations are based on simple phantom geometries. FMI and BLI/ISP data are simulated using the Lipros program, while CT labels are generated using Python code implementing the Radon transformation from the Scikit-Image Library.

Following simulation, the data required certain preparation for optimal input, as the network cannot be trained on projection data. This was particularly crucial as the neural network necessitated a specific infrastructure and data setup for proper utilization, which will later be discussed in detail.

Subsequently, this data was employed to train nnU-Net, a pre-programmed neural network. Utilizing the trained network, newly generated FMI-Images, again produced by Lipros and prepared using Python code, could be accurately segmented and supplemented with missing information. This desired outcome is then used to reconstruct the phantom, ensuring its integrity.

Throughout the ongoing research, the training and testing data have been updated or modified. This includes replacing BLI/ISP labels with CT images, adding White-Gaussian Noise to the FMI images, generating phantom data containing multiple tumors or creating heterogeneous data. These adjustments were mostly made by manipulating the initial simulated data using custom Python code.

For data simulation, preparation, manipulation, and analysis, various tools have been employed. The concepts of these tools will be described in the subsequent sections.

2.1 Lipros

The FMI and BLI/ISP images utilized in this thesis originate from the simulation program "Lipros," developed, maintained, and operated by Dr. Jörg Peter from the DKFZ. This simulation program employs the Monte Carlo algorithm to simulate light fluence induced by both BLI/ISP and FMI.

The Monte Carlo Method simulates the interaction of photons as they undergo scattering and absorption events based on local parameter values. This process continues until the photons either have negligible contribution, or escape the surface, thereby being detected by the camera [3].

The simulation begins with the creation of an excitation photon using predefined light source parameters. The propagation of this excitation photon from the surface to the interior of the phantom, where the tumor is situated, results in the creation of an emission photon. After modeling the photon's propagation back to the surface, where it can be detected by the camera, the simulation is completed. This process yields simulated models of excitation, emission, and actual camera images, approximating real camera images captured in in-vivo research. Although Lipros outputs these three modalities, only the camera view will be utilized in this thesis, as it offers a realistic representation and is the only applicable modality in practical scenarios.

2.2 nnU-Net

The neural network employed in this study is known as nnU-Net, short for "No New U-Net". U-Net, as discussed in Chapter 1.3.2, is a convolutional network designed specifically for biomedical image segmentation. Hence, nnU-Net builds upon the foundation of the U-Net model, utilizing a series of optimization strategies to fully exploit its capabilities [35].

This framework automatically adjusts to new datasets by performing self-configuration tasks, including preprocessing steps, determining the optimal patch size and batch size, and configuring inference settings based on the characteristics of the dataset at hand. Moreover, the network architecture, training process, and post-processing techniques are also adapted to suit the requirements of each new task [17].

This involves modifying the original segmentation algorithm (chapter 1.3.2), particularly the computation of the set of hyperparameters θ. Until now, there hasn't been guidance on adjusting θ when transitioning to a new dataset with different characteristics, necessitating manual updates to the hyperparameter set for each new dataset.

This lack of adaptability has led to the exploration of a generalized function for θ that

can perform well across diverse datasets, expressed as:

$$g(X, Y) = \theta \tag{2.1}$$

Now, θ is categorized into distinct groups. Parameters that define a robust segmentation architecture and training scheme, requiring minimal adaptation, are identified. Conversely, dynamic parameters, such as heuristic rules governing normalization, resampling, patch size, batch size, and network geometries, require adjustment based on the characteristics of the dataset, leading to the formulation of the function $g(X, Y)$ as described above [17].

In conclusion, nnU-Net is now available as a readily deployable tool. By providing a robust architecture, nnU-Net makes state-of-the-art segmentation accessible to a broader audience without the need for specialized expertise; only standard computing resources are required [16].

Furthermore, the recent finalization of nnU-Net Version 1 and ongoing development of Version 2 offer notable improvements. Version 2 introduces smarter disk storage, streamlined support for 2D data formats (such as .png, .mhd, .tiz), and an improved internal structure for easier usage. Consequently, nnU-Net-v2 aligns perfectly with the objectives of this study, providing the necessary support and functionality [18].

Method Configuration

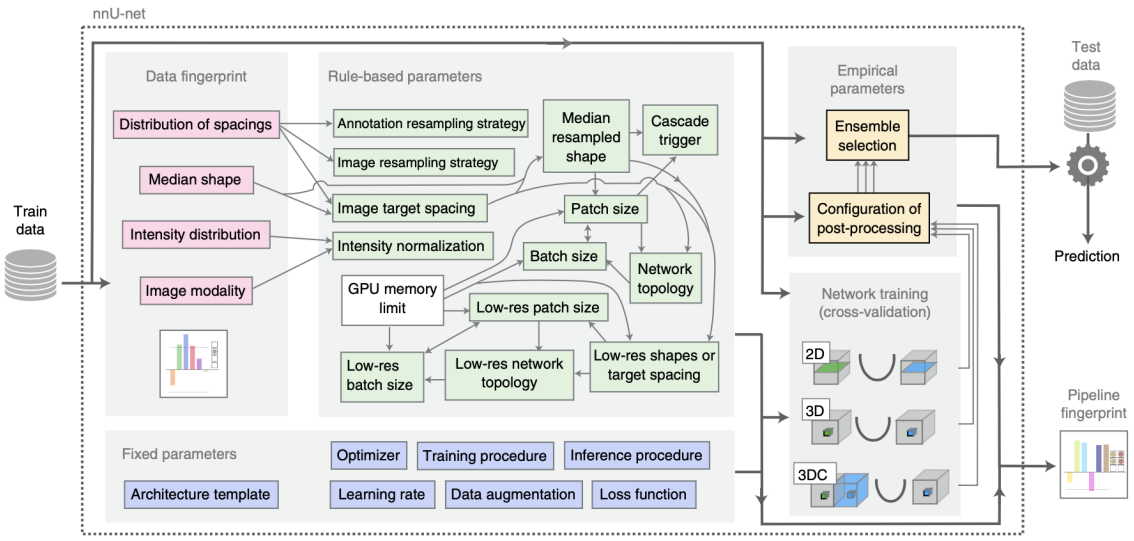


Figure 2.2: Workflow of nnU-Net [16]

Figure 2.2 illustrates that nnU-Net simplifies user interaction by requiring only proper preparation of input data (both training and testing) in the correct format, instead of setting every parameter manually. Most tasks are automated within the nnU-Net framework.[17]

Initially, the training dataset is entered, from which a data fingerprint (displayed in pink) is extracted. This fingerprint is utilized to generate empirical parameters and facilitate network training. Subsequently, the data fingerprint influences the adjustment of various rule-based parameters (highlighted in green). These rule-based parameters are then combined with a set of fixed parameters (emphasized in blue), proven effective across diverse contexts of (medical) image segmentation. The initial input data, along with the parameter sets, is employed to train a neural network, yielding a specific model. Finally, the model undergoes post-processing based on empirical parameters (indicated in yellow), ensuring optimal performance. This well-trained model is then applied to test data, enabling predictions based on the input data.

For a more comprehensive understanding of each attribute section and configuration step beyond what is outlined below, please consult the document [16], beginning on page 10.

Preprocessing

Image Normalization: For proper intensity value normalization, nnU-Net requires knowledge of the modality of its input channels.

Voxel Spacing: After gathering spacing information within the training data, all training cases undergo resampling using third-order spline interpolation. Corresponding segmentation labels are resampled using linear interpolation.

Training Procedure

Network Architecture: Three U-Net models are configured, designed, and trained independently: a 2D U-Net, a 3D U-Net, and a cascade of two 3D U-Net models. In the cascade model, the first U-Net generates a segmentation at a low resolution, which is then refined by the second model. Apart from the U-Net architecture, padded convolutions are utilized to maintain identical output and input shapes, and instance normalization and Leaky ReLUs are employed instead of ReLUs.

Network Hyperparameters: All U-Net architectures begin with 30 convolutional filters in the first layer, doubling this number with each pooling operation. Pooling along each axis continues until further pooling would reduce the spatial size of the axis below 4 voxels.

Network Training: The standard training schedule involves 1000 epochs for each training process, with one epoch defined as an iteration over 250 mini-batches. All U-Net architectures undergo training in a five-fold cross-validation. During training, the learning rate is reduced by a factor of 0.2, if the average of the training loss does not improve within the

last 30 epochs. Training halts when the learning rate drops below 10^{-6} or exceeds 1000 epochs.

Inference

nnU-Net ensembles combinations of two U-Net configurations (2D, 3D, and cascade) and automatically selects the best single model or ensemble for test set prediction based on cross-validation results. If only one configuration is trained, for example when the input data is solely two-dimensional, there is no need to select the best single model for test set prediction.

Fingerprints

Dataset fingerprints (pink): Initially, the data undergoes cropping to its non-zero region, thereby minimizing the image size. This process captures all relevant parameters and properties, including the image size (number of voxels per spatial dimension) before and after cropping, image spacing (physical size of the voxels), modalities (derived from metadata), and the total number of training cases.

Pipeline fingerprints: nnU-Net automates the design of deep learning methods, condensing the design choices to the most critical ones and storing these rule-based parameters in a pipeline fingerprint. Additionally, fixed parameters, which remain constant across datasets, and empirical parameters, optimized during training, are captured in the pipeline fingerprint.

2.3 Python Libraries

Throughout this work, Python has been utilized for many different tasks, each taking advantage of different libraries.

Python is a universal programming language widely adopted across various fields. Its straightforward syntax makes it beginner-friendly while offering versatility and flexibility. Python libraries consist of predefined algorithms and functions that extend the capabilities of the language, providing a wide array of tools and modules for diverse tasks, thus simplifying programming endeavors. Additionally, some libraries may be built upon combinations of others.

For instance, libraries like **NumPy** and **SciPy** are fundamental for scientific computation and form the backbone of every data science and analytics code. **Matplotlib** is essential for data visualization, while **Scikit-Learn** enables the utilization of machine learning algorithms. The **Pillow library**, **OpenCV**, and the Python toolbox **Scikit-Image**

are indispensable for image processing, and were used for the preparation of simulated data.

SimpleITK is employed to handle MHD files effectively, allowing for their reading into Python arrays and conversion back into images of any data type.
For drawing phantoms, **PIL functions** such as Image, ImageDraw, and ImageFont prove useful.

Python offers numerous methods for automating tasks, including interacting with system files when necessary, providing an alternative to **bash scripting**.

Bash, a command-line interface shell program extensively used in Linux, allows direct control over a computer's operating system through a command-line interface (CLI). It is commonly utilized to automate file and application operations to save time. By utilizing Python modules such as 'shutil', 'os' and 'subprocess', tasks, typically performed in Bash, can be performed by Python.

In the context of implementing CT images based on a phantom, the Radon-Transformation-Library from the Scikit-Images-Package is a suitable choice. While the detailed concept of simulating a Computer Tomography experiment is covered in Chapter 1.2.8, only the essential implementation details are highlighted here. Functions like 'radon', 'rescale', and 'resize' from skimage.transform, as well as '_get_fourier_filter' from 'skimage.transform.radon_transform', are used for simulating the CT label.

The Radon Transform also conveniently offers the capability to reconstruct the input image, the phantom, based on the resulting sinogram outputted by nnUNet, by importing **iradon** from **skimage.transform**. For performing the inverse Radon transform and reconstructing the original image, two methods can be included: Filtered Back Projection (FBP) and Simultaneous Algebraic Reconstruction Technique (SART). In this work, the Filtered Back Projection Method (FBP) of the Radon Transform is applied.

Lastly, when developing Deep Neural Networks or AI-related applications, three common libraries are available: TensorFlow, Keras, and PyTorch. nnU-Net specifically utilizes **PyTorch** as its core library.

2.4 Image Software

The output generated by the simulation algorithm "Lipros" is typically in the form of mhd-files, a common data type for medical and scientific images. Each image is split into a pair of files: an .mhd file and a .raw file. The mhd-file acts as a reference to the pixel values stored in the raw file. The format of the mhd-file specifies how the image data is organized within the file and how the pixel data should be interpreted for correct loading

and visualization. Additionally, mha-files combine both the raw and mhd files into a single file, making it less straightforward to access their settings. nnUNet is capable of working with images of either png or mha type.

However, mha or mhd images are not readily displayable. Specialized image editing and processing software, such as MITK, ImageJ, and Fiji, designed for medical and scientific analysis, can be used to handle them.

ImageJ is a Java-based image processing toolkit that is compatible with Linux, Mac OS X, and Windows. It is freely available without any licensing requirements. **Fiji**, a distribution of ImageJ, includes useful plugins and can be easily installed on Mac OS X [15].

According to the documentation provided by MITK, the Medical Imaging Interaction Toolkit (MITK) is a free open-source software designed for developing interactive medical image processing applications. It serves as a C++ toolkit or application framework for software development [27].

MITK was readily accessible on the Linux distribution at DKFZ in Heidelberg, while Fiji was installed on the personal Mac OS X system to allow for more flexible usage.

Chapter 3

Creating Data

Apart from simulating FMI and ISP data, all other tasks in this project were tackled through the development of custom applications using the Python programming language. The primary tasks executed with Python are summarized below, and the associated code is available in Appendix E.

- Preparing of simulated data, necessary for alignment with the formal expectations of the neuronal network [E.10 and E.11].
- Drawing phantoms for better display and further use for CT-simulation [E.4].
- Simulating CT-labels based on the input phantoms [E.1].
- Adding noise to the input images [E.2].
- Creating composite phantoms containing multiple tumors through super-positioning input phantoms [E.3].
- Reconstruction of the output data generated by nnU-Net [E.5].
- Applying simple filter for reducing artefact in CT-Output [E.5].
- Creating difference image for analysing the results [E.7].
- Automating manipulation of images and implementing repetitive tasks [E.12 and E.13].

3.1 Simulation using Lipros

The simulation code operates based on various parameters that can be adjusted independently. These parameters mimic the setup of an optical imaging modality and include elements such as the Gantry, Cameras, Light Source, and the Phantom, which can be

defined as either homogeneous or heterogeneous tissue, along with the specific induced light fluence simulation modalities.

Camera Description

Initially, the parameters for the gantry, which acts as a rotating disk, must be defined. At first the number of cameras is set, which remains constant at 30 throughout this project. Lipros offers two camera setups: orthographic and plenoptic rendering of surface fluences. For simplicity, the orthographic camera setup is chosen for this work. As discussed in chapter 1.2.3, this setup involves an affine transformation, ensuring that an object's size in the rendered image remains constant regardless of its distance from the camera. Thus, the distance of the cameras to the phantom is inconsequential.
Each camera is characterized by its size, defined by the size of sensors, which in turn are determined by sensor pixels. Both, voxel size and pixel size within the setup are set to 0.1mm and need to be defined in the code. Each sensor with a size of (0.1 x 11)mm equals therefore to (1 x 110)pixel, so the cameras are intact line sensors. This reduction of image size is done due to efficiency reasons. However, this takes nothing away from the validity of the method. It's worth noting that when adjusting the number of cameras in the gantry parameter, the size of the cameras themselves must also be updated accordingly.

Furthermore, the simulation allows for sensor noise simulation, which is controlled by parameters such as the "Signal to Noise Ratio" and the "Noise Gain" of the camera sensor. The SNR is consistently set to 5dB according to the "Rose Criterion" (explained in chapter 1.2.4). For the primary simulation using Lipros, the Noise Gain, which is based on the concept of the "Additive White Gaussian Noise model" is set to 0. However, there is a scripted option to apply Noise Gain to final images later (see 3.2.3 and E.2).

Parameter	**Value**
gantry	gantryNumberOfCameras=30
cameras	cameraSensorMinSize=0.1mm 11mm
	cameraSensorSNR=5dB
	cameraSensorNoiseGain=0
cameraOrthographic	

Light Source Description

The excitation light sources consist of lasers positioned between each camera. These sources are characterized by parameters that define the filter properties applied to the laser light. A filter is designed to permit a specific range of wavelengths of light to pass through. The Wavelength Center (CW) denotes the wavelength at the center of this range, while the Full Width at Half Maximum (FWHM) represents the bandwidth, where the filter transmits at least 50% of light at the CW. For instance, in this scenario, a

'660-20' filter has a CW of 660nm and a bandwidth of 20nm, resulting in an overall range of 650nm to 670nm where light transmission is relatively unimpeded. Beyond this range, the filter significantly attenuates light, allowing only visible red light to pass through the air and the phantom, thus exciting the fluorophore.

Parameter	**Value**
gantry	gantryNumberofLights=30
lights	excitationConfocalWavelengthCenter=660nm
	excitationConfocalWavelengthFWHM=20.0nm

Lipros offers the flexibility to define phantom tissues using either 3D surface polygonals from .ply files or 3D triangulation meshes from .mesh files. Tissue inserts, such as tumor lesions, can alternatively be defined using point clouds from .pcl files. However, for simplicity, this study opts for basic geometries, utilizing cylinder and sphere shapes to represent the phantom with a tumor.

Phantom Description

The phantom body is modeled as a cylinder resembling muscle tissue, with a diameter of 10.5mm and a length of 20.0mm, although the length is inconsequential as the subsequent analysis will be simplified to two dimensions. Point clouds are currently avoided due to their unnecessary complexity, given the early stage of the research.

The tumor is represented as a sphere with a diameter of 2.0mm, positioned within the phantom body. Its fluorophore type is defined as Cy5. Invitrogen Cyanine5 (Cy5) dye is a highly luminescent, far-red-fluorescent dye. One notable advantage of using long-wavelength dyes like Cy5 is their minimal autofluorescence interference from biological specimens in this spectral region [9].

Parameter	**Value**
phantom	phantomExportVoxelSize=0.1mm 0.1mm 0.1mm
tissue	tisGeometry=CYLINDER
	tisScale=10.5 10.5 20.0
	tisType=MUSCLE
tissue	tisGeometry=SPHERE
	tisScale=2.0 2.0 2.0
	tisTranslate=$xmm $ymm 0.0mm
	tisType=TUMOR
	tisFluorophore=CY5

To gain a better understanding for the experimental setup the 2-dimensional layout is sketched in Fig. 3.1 below.

In addition to simulating FMT emission and excitation, the FMT emission light fluence is utilized to reconstruct camera data within the phantom. To simulate specific fluence sce-

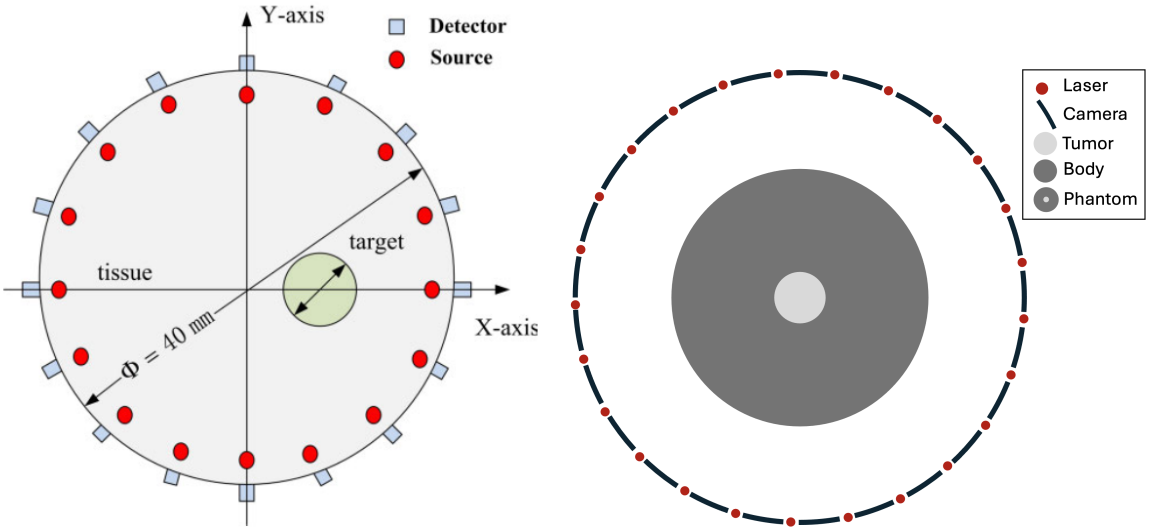


Figure 3.1: Experimental Setup of Cameras, Lasers and Target in general theory (left) and specifically for Lipros-Simulation (right)

narios accurately, the behavior of scattered fluences must be separately simulated within a defined geometric range. This range consists of a 4.0mm window located at the central z-position of 0.0mm, precisely in the middle of the cylinder.

The excitation of photons per position, on which the fluorescence is simulated, is set to 1000000 photons. Each photon is excited for a pulse duration of 1 second, with the laser is starting off at the right side of the phantom at 0.0 degrees to rotate around the phantom.

Parameter	**Value**
simulate	simulateFluenceVoxelSize=0.1mm 0.1mm 0.1mm
	simulateTotalAbsorberZmin=-2.0mm
	simulateTotalAbsorberZmax=2.0mm
	simulateCameraAxialCenterPositionZ=0.0mm
simulateFmi	simulateLightStartAngleOffset=0.0deg
	simulateLightExcitationPhotonsPerPos=1000000
	simulateLightExcitationPulseDuration=1s

If simulating Bioluminescence Imaging (BLI), which will represent **Ideal-Source-Projections (ISP)** without attenuation and scattering, instead of FMI, the code requires minimal adjustments. Specifically, the phantom's body, represented by the muscle cylinder, is removed, and BLI-specific parameter values are assigned instead of FMI parameters. For instance, the number of gantry lights is set to zero.

Additionally, tissue parameters are modified to reflect the simulation mode change from Fluorophore to Bioluminescence, with RedFluc designated as the type of Bioluminescence tissue. Red-shifted bioluminescence reporters, like RedFluc, are preferred for biological

imaging due to their enhanced performance. Luciferase enzymes, which catalyze the bioluminescent reaction, are commonly utilized in various bioluminescent organisms to emit light [48].

Parameter	**Value**
gantry	gantryNumberOfLights=0
tissue	tisBioluminescence=REDFLUC
	tisBioluminescencePhotons=1000000
simulateBli	

It was observed that the required number of photons for satisfactory image quality in the simulations is lower than initially anticipated. Despite this reduction, the images remain clear and of high quality, and nnUnet effectively processes the provided data. Consequently, this reduction in photon count results in shorter overall simulation runtimes for Lipros which is beneficial for a possible increase in running and varying simulations.

To generate an adequate amount of training and testing data, a for-loop was utilized to vary the tumor tissue's position and create various phantom configurations. A superposition of all these positions is illustrated in figure 3.2, showcasing the diversity of phantom types for simulation purposes.

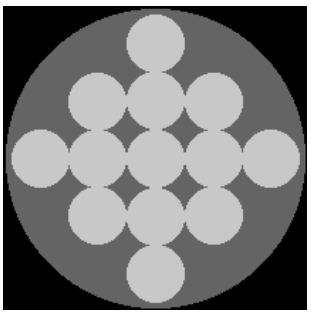


Figure 3.2: Superposition of different tumor positions inside phantom

The completed Lipros setup for the simulation can be seen in Appendix [E.8].

3.2 Preparation of the FMI-Camera-Images

As discussed previously, the imaging setup comprises 30 lasers and 30 cameras. Typically, each camera captures the output from each laser beam, resulting in 30 different angle images, each with dimensions of $x \times y$ pixels, for every camera. This yields a total of 900 images (30 lasers $\times$ 30 cameras) with dimensions $x \times y$. To optimize network training time without sacrificing quality, as well as to reduce the final dimension of the image, the image size was restricted to 1mm wide and 11mm long images (1×110 pixels).

Consequently, each image outputted by Lipros has dimensions (1, 110, 30), representing 1 pixel $\times$ 110 pixels $\times$ 30 lasers. There are 30 such images, each corresponding to one

camera (as shown in the second left section of Fig. 3.3). To create a single image for network input, these camera images were concatenated, forming a block with dimensions (30, 110, 30), denoting 30 cameras × 110 pixels × 30 lasers (as depicted in the second left section of Fig. 3.3).

To further reduce the data complexity to two dimensions, each set of corresponding laser images was summed up, resulting in an image with dimensions (30, 110), representing 30 cameras × 110 pixels. Rotating this image yields the final dataset suitable for network training (as shown in the two left sections of Fig. 3.3).

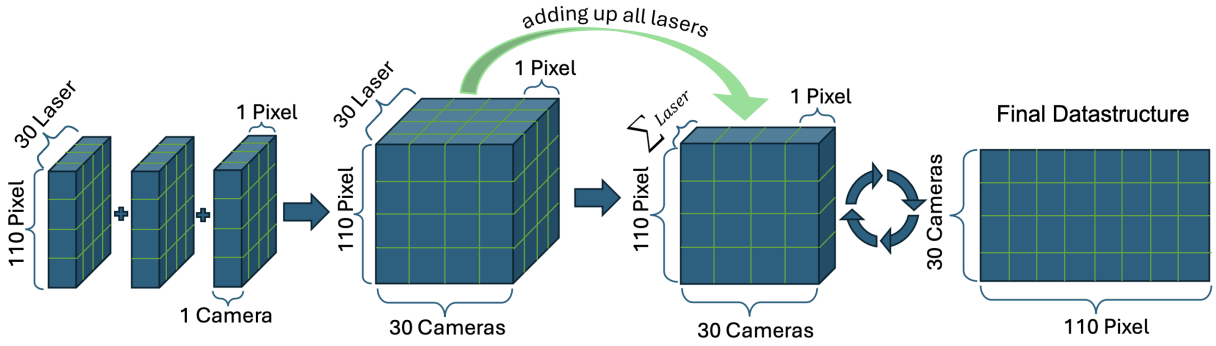


Figure 3.3: Preparation of FMI Images: on the left side is the output of Lipros, which is being manipulated to get the final **data structure** of input image for the network dataset according to nnU-Net standards (right)

In fact, all experiments were initially conducted with a series of images of dimension 1×55. However, since the complete sinogram could not be depicted through inverse Radon reconstruction due to information being cut off at the sides already during simulation, a completely new dataset was trained, for which the individual image size was increased to 1×110. As a result, a camera detector is now slightly longer than the phantom itself, restoring the spatial overall image and improving the performance of inverse Radon reconstruction. Instead of investing effort into restructuring the image size, consideration could have been given to rescaling the reconstruction process. However, since the research does not only focus on reconstructing nearly perfect sinograms, but rather on achieving them, the decision was made to create a new simulation with an image size of 1×110.

Nevertheless, to incorporate the images into the existing workflow, the 1×110 images were rescaled to 1×55, while preserving all their information. Each experiment was then conducted again using this dataset.

Understanding the correlation of the sinogram with the phantom based on the imaging procedure

To grasp the relationship between the sinogram and the imaging process, figure 3.4 offers a simplified explanatory framework. In each simulation, the rotation begins on the right side (3 o'clock) of the phantom, with the projections rotating clockwise. Each camera

detects emitted photons within a specific section of its sensor, registering light only in that particular area and displaying it in the images (as depicted on the left). The FMI-Images (bottom left) will reveal no illumination in sections where excessive scattering prevents emitted photons from reaching the detector.

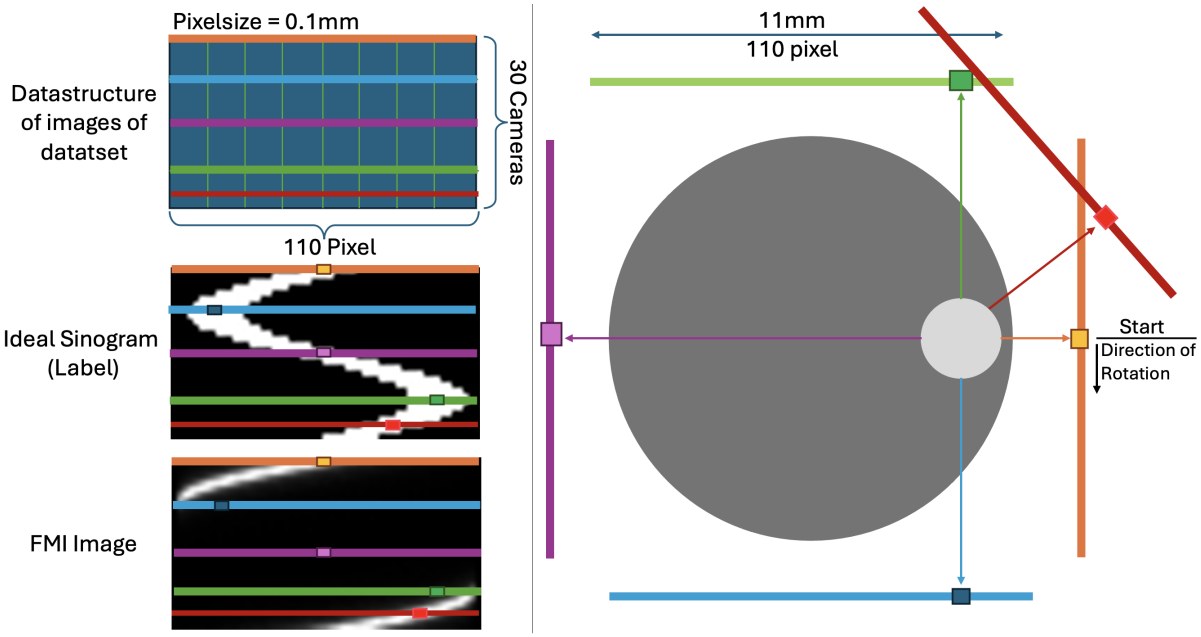


Figure 3.4: Explanation of creation process of Input/FMI Images (left bottom) and source-only ideal sinogram Images (left middle) of nnU-Net

In order to gain an intuitive understanding and internalize the structure of the images in correlation to the phantom, a few more input/output pairs are displayed:

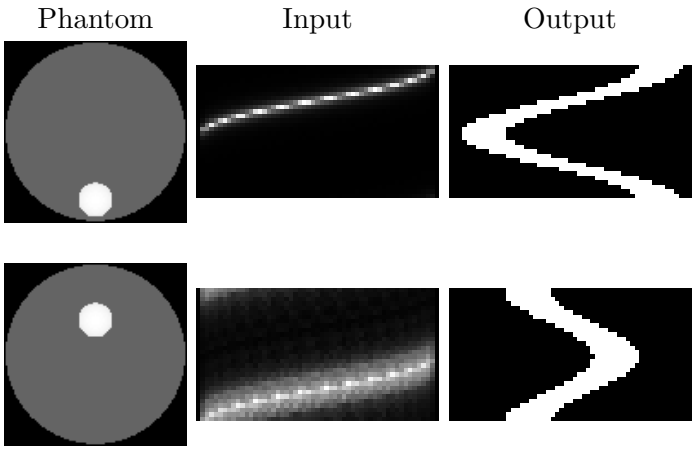


Figure 3.5: Examples of input (FMI image given to the network model for testing) / output (predicted image from testing the network model) pairs in correlation to the phantom

3.2.1 Model 1: single tumor per phantom

Lipros simulates the single-tumor setup directly. In addition to the camera sequences that are then combined to form the final sinogram, it simulates phantom images.

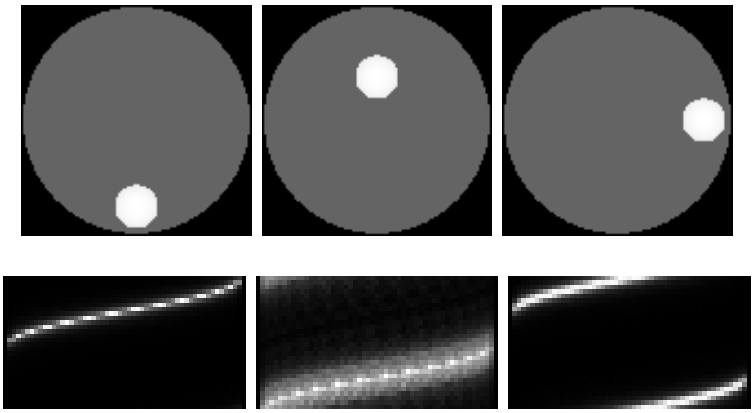


Figure 3.6: Examples of Single-Tumor-Phantom (top row) with related FMT-Image Sinograms (bottom row), respectively.

3.2.2 Model 2: multiply tumors per phantom

Instead of conducting an entirely new simulation to incorporate two tumors per phantom, the new data was supposed to be produced through the superposition of single-tumor phantoms. Since the images are independent of each other, the processes involved are also independent. Thus, they represent additive processes, allowing for the creation of new multi-tumor phantom images through superposition and normalization of the existing data.

Naturally, there are distinctions between the superposition approach (its results seen in figure 3.7) and the method of simulating new images. These differences originate from the slight absorption of the excitation light by the other fluorophore concentration in each case. However, this discrepancy is minimal, accounting for only about 5% difference between simulating two-tumors or creating two-tumors with superposition. Hence, the resulting sinograms of two-tumor phantoms appear as follows:

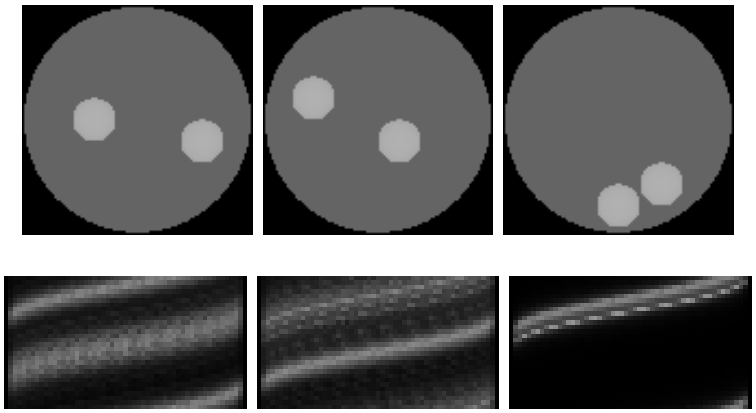
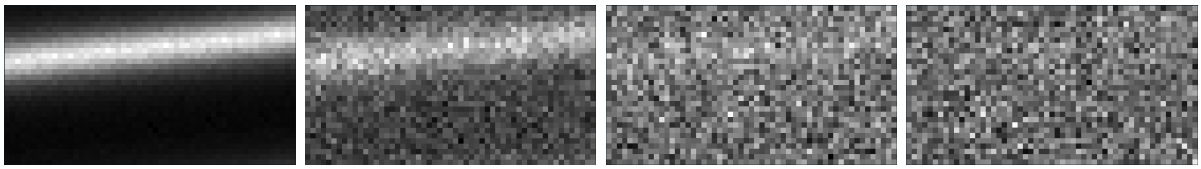


Figure 3.7: Examples of Two-Tumor-Phantom (top row) with related FMT-Image Sinogram (bottom row)

3.2.3 Adding Noise - Noise Gain

As explained earlier (see Chap. 1.2.4), image noise refers to the presence of unwanted artifacts that are not part of the original scene, often manifesting as a grainy texture overlaying the image. This statistical variation in the measurement arises from random processes [28], allowing for the replication of these random effects to introduce controlled levels of noise to an image. By intentionally introducing noise through modulation and coding, the presence of artifacts can be increased, subsequently diminishing the technical quality of the signal and overall image. Integrating noise into the initial images enhances their realism.



(a) No Noise (b) NoiseGain=1 (c) NoiseGain=5 (d) NoiseGain=10

Figure 3.8: NoiseGain applied to Images

3.3 Preparation of ISP-Labels

The preparation of Ideal-Source-Projections (ISP) without attenuation and scattering was considerably simpler compared to FMI images. The objective of the required labels was to represent the idealized version of a sinogram of the tumor inside the cylinder, assuming unobstructed photon propagation through the tissue. Hence, these labels were based on the premise that the phantom solely comprised the tumor without any other tissue types like muscle, organs, or skin that could interfere with light propagation. This was achieved by taking advantage of Bioluminescence Imaging (BLI) of only the tumor itself in space.

The Bioluminescence Images (BLI) were simulated in a similar way to FMI simulation using Lipros, with each of the 30 cameras capturing an image from only one single light source, resulting in reduced output data compared to FMI. To generate the final sinogram image as the label input for network training, the one-dimensional 1×110 images from each camera were summed together and eventually rotated (see Fig. 3.9).

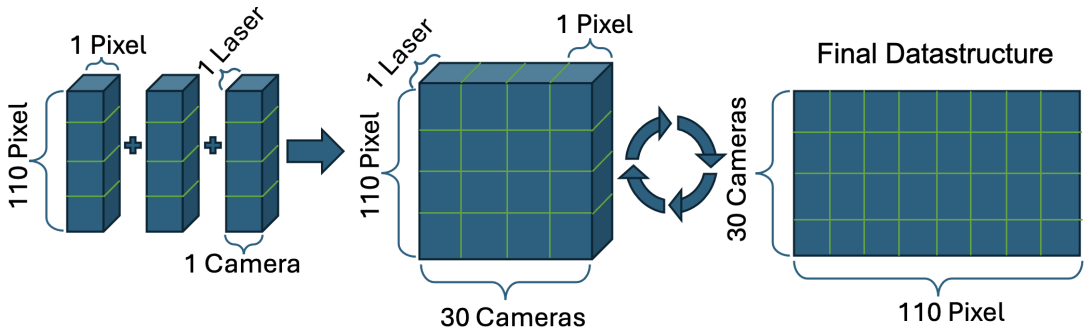


Figure 3.9: Preperation of Images: on the left side is output of Lipros, which is being manipulated to get the final input image for nnU-Net standards (right)

The same concept outlined for the creation of multiple tumors FMI-Images applies to the BLI-Simulation's Two-Phantom-Images as well.

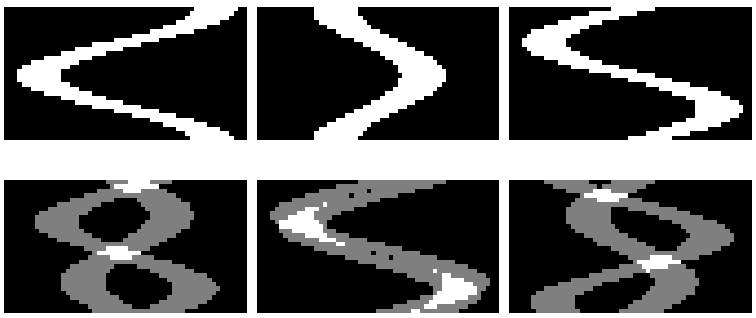


Figure 3.10: Examples of ISP-Label Sinogram; single-tumor-phantom (top row) and two-tumor-phantom (bottom row)

3.4 Creating CT-Labels

As detailed in chapters 1.2.1 and 1.2.8, utilizing phantom slices (cylinder with tumor) and the Radon Transform library from **skimage.transform**, both the forward and inverse transformation and reconstruction can be performed. Specifically, the Filtered Back Projection Method (FBP) was applied here. Further details regarding the implementation can be found in Appendix E.1.

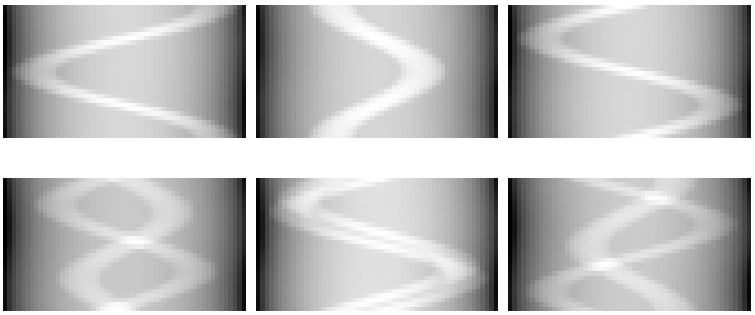


Figure 3.11: Example for CT-label of single-tumor-phantom (top row) and two-tumor-phantom (bottom row)

3.5 Analysing Reconstruction Process

After testing the trained network model, further steps of modification are applied to either further improve the sinogram or conduct the process of reconstruction of the phantom. The core step of the reconstruction transformation (applied in all cases) is the Filtered Back Projection (see code snippet E.5), labeled as 'FBP-Recon.' in figure 3.12. Afterwards it is possible to apply a self-written filter, that addresses fringed edges and small artifacts in the reconstructed phantoms. This is done through conducting segmentation based on pixel value thresholds, that delineate segmented regions in the image, labeled as 'PhantomFilter' in figure 3.12 (see code snippet E.5). In regards of any CT-configuration, some network outputs show artifacts (black pixel spots like in image 3.12e). Those numerical artefacts probably originate from the trained network model itself, possibly due to an overflow of maximal values. They are repairable though applying a specific type of filter, smothering the black pixel spots. This filter is applied before reconstruction and is labeled as 'ArtefactFilter' in 3.12 (refer to code E.6).

Before continuing, the quality of the just described reconstruction methods used in this study will be accurately assessed, by visualising their efficiency in regards of every type of singoram occurring in this work.

The first row in figure 3.12a shows the bad quality of the reconstructed phantom when not applying any framework to the simulated FMI sinogram. Next, the applied modifications are observed on ideal sinograms (see Fig. 3.12b and 3.12d), which will be treated as labels for training. Figures 3.12c and 3.12e show the same for the framework's output, the segmented sinogram. This is done for both cases, ISP as well as CT configurations. It provides a basis for understanding the reconstruction quality of any sinogram type.

Nevertheless it is import to highlight, that this thesis does **not focus on the process of reconstruction**. There are far more developed methods than only the simple parallel FBP presented here. It is only suppose to offer a impression for the assessment of the developed framework.

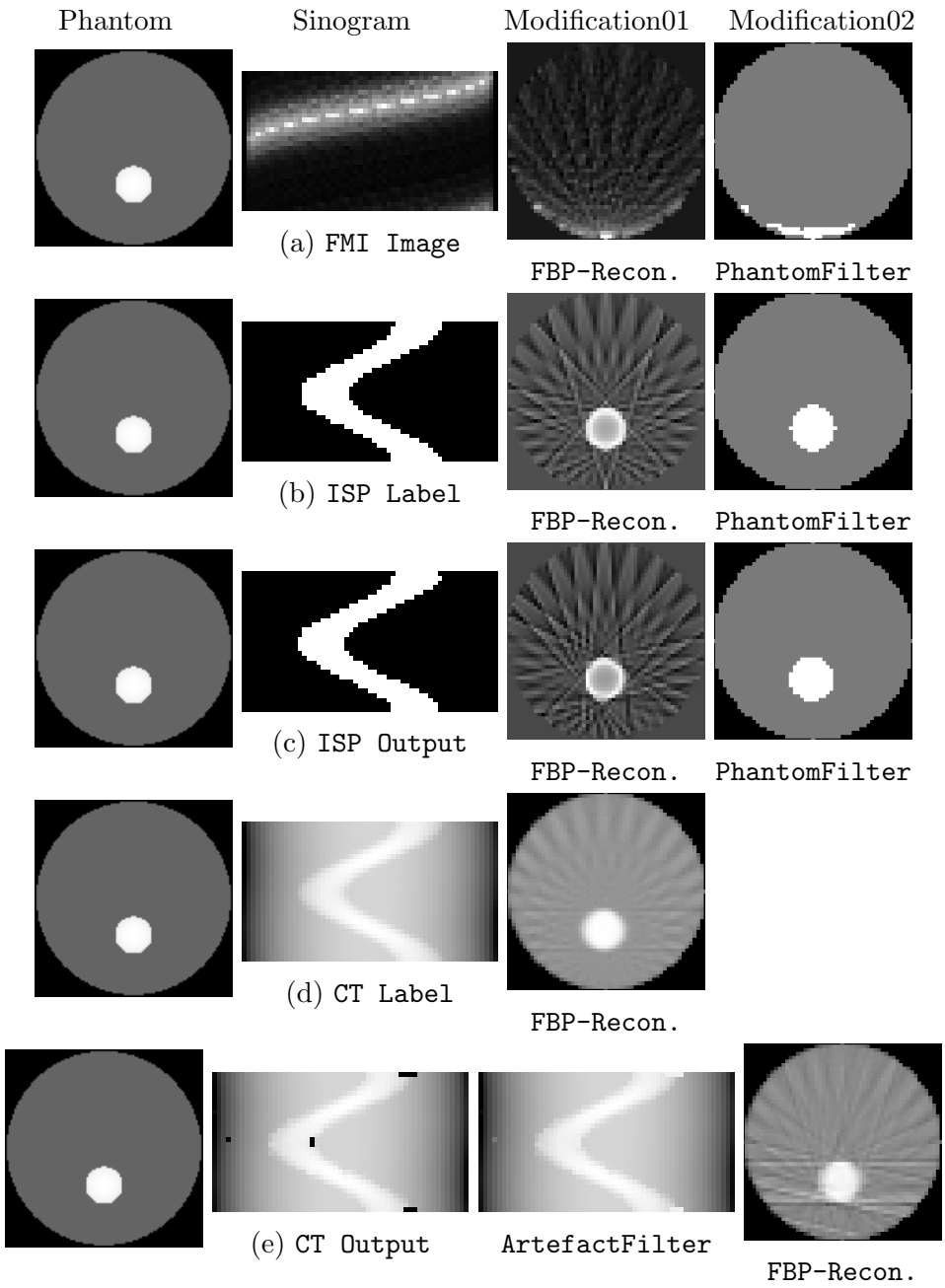


Figure 3.12: Analysis of quality of reconstruction of phantom slice based on a incomplete FMI image (a); based on ideal ISP-data used as Label (b); based on an output image after testing the network model on ISP (c); based on ideal CT-data used as Label (d); based on an output image after testing the network model on CT (e). For each configuration the initial phantom, the corresponding sinogram, as well as the first and second type of modification are shown.

3.6 Implementing nnU-Net

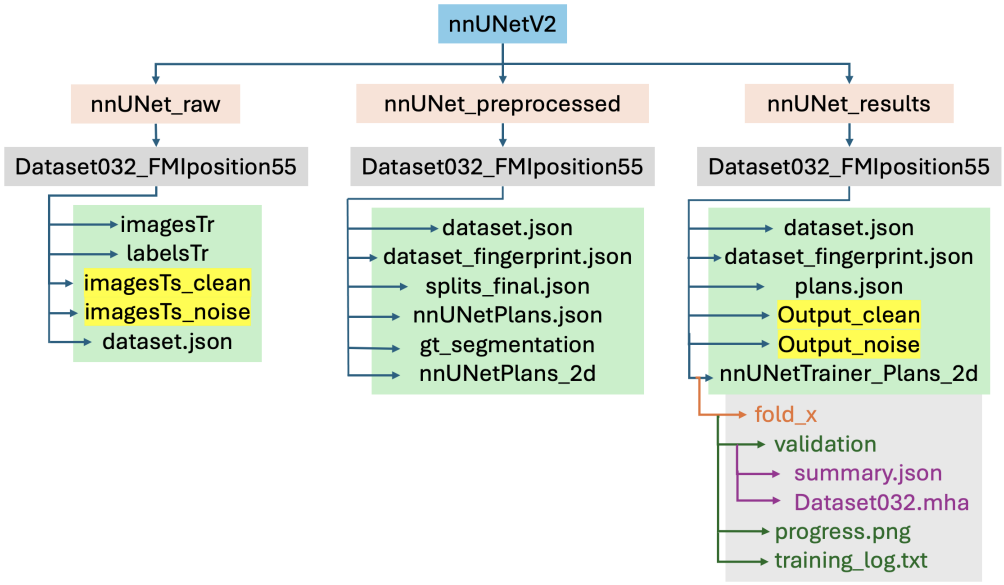


Figure 3.13: Local folder structure for nnU-Net application

Preparation of Raw Data

To prepare the datasets, various Python scripts are employed to manipulate images as required and to establish the necessary structure for integrating the dataset into nnU-Net. The finalized prepared dataset, comprising training images, associated labels, diverse types of testing images, and a .json-file, is then stored within the "nnUNet_raw" folder (Fig. 3.13). The dataset.json file is crucial for the nnU-Net to recognize the data type provided. Within the dataset.json file, the segmentation labels have to be given. ISP labels are encoded in binary code, represented as "0" and "1". In contrast, for the CT label, which necessitates multi-variable information, the labels range from 0 to 255, corresponding to all pixel values within an image.

Preprocessing

The following commands were utilized to plan and preprocess the training setup (see theory in chapter 2.2). Upon successful execution of these commands, a new folder named "nnUNet_preprocessed" is generated, housing all necessary information for actual training (see Fig. 3.13). If any input data is in an incorrect format, error messages will guide the user to correctly set up the data.

```
1 nnUNetv2_plan_and_preprocess -d DATASET_ID --
     verify_dataset_integrity
2 nnUNetv2_plan_and_preprocess -d 32 --verify_dataset_integrity
```

where "verify_dataset_integrity" will check for some of the most common error sources.

In detail *plan_and_preprocess* is calling

```
nnUNetv2_extract_fingerprint
nnUNetv2_plan_experiment
nnUNetv2_preprocess
```

Training

Once all the necessary configurations are set up, you can initiate the training process with the following commands:

```
nnUNetv2_train DATASET_NAME_OR_ID UNET_CONFIGURATION FOLD [
   additional options , see -h]
nohup nnUNetv2_train 32 2d x --c --npz &
```

In this command, "x" specifies the specific fold to be trained (0 to 4) or if only a prediction with a single model should be made, "x = all". While "2d" represents one of the three U-Net configurations available (2d, 3d_fullres and 3d_cascade_fullres). The options "–c" and "–npz" are among the additional options that can be specified. Using "nohup" and "&" at the end of the command allows the process to run in the background even when the screen is on standby, ensuring uninterrupted long-term training.

For parallel training of all 5 folds or multiple datasets, the following command can be used, which calls the script "run_nnUNetv2_39_40_41.sh" E.14:

```
nohup run_nnUNetv2_39_40_41.sh &
```

Once the training is complete, a folder named "nnUNet_results" will be created, containing all the data generated during the training process (see Fig. 3.13).

Evaluation

With a new set of input data (testing images) located in the "nnU-Net_raw" folder, the trained model can be tested, and the final predicted images will be stored in the output folder inside "nnUNet_results".

```
nnUNetv2_predict -i INPUT_FOLDER -o OUTPUT_FOLDER -d
   DATASET_NAME_OR_ID -c CONFIGURATION --save_probabilities

nnUNetv2_predict -i /home/svea/Documents/nnUNetv2/nnUNet_raw/
   Dataset032_FMIposition55/imagesTs -o /home/svea/Documents/
   nnUNetv2/nnUNet_results/Dataset032_FMIposition55/output -d 32 -
   c 2d --save_probabilities
```

Per default, inference will be done with all 5 folds from the cross-validation as an ensemble - considering all 5 folds were trained prior to running inference. We very strongly recommend you use all 5 folds. It is also possible to make predictions with a single model, thus the all fold needed to be trained and specified in nnUNetv2_predict with "-f all".

Once all folds have been trained in all three different training modes (2d, 3d, 3d-fullres), the best configuration can be selected.

```
nnUNetv2_find_best_configuration DATASET_NAME_OR_ID -c
   CONFIGURATIONS
```

Since the input images are all 2-dimensional, only the 2d-training will be applied, making the selection step unnecessary for this work.

Those are the following datasets that were created, trained and used:

- Dataset030_CTpositionRadon_noise5_self
- Dataset031_FMIposition110n5
- Dataset032_FMIposition55
- Dataset033_FMInoise5
- Dataset034_FMIposition110
- Dataset035_CTpositionRadon
- Dataset036_CTpositionRadon_noise5
- Dataset037_FMInoise5_self
- Dataset038_addTwo
- Dataset039_addTwo_ct
- Dataset040_addTwo_ct_noise
- Dataset041_addTwo_noise

Chapter 4

Results and Discussion

In summary, the aim of this research is to develop a framework for predicting FMI images efficiently based on training. Due to numerous artifacts caused by heavy scattering in these images, the inverse problem of photon propagation through tissue is ill-posed and ill-conditioned. Conventional approaches, such as iterative solution algorithms, reconstruction methods and regularization methods of solving the inverse problem for medical image reconstruction are impractical. Instead, the proposed approach involves semantic segmentation to predict the corresponding ideal sinogram, using a trained neural network. This prediction process aims to convert incomplete sinograms into rematched versions based on ideal complete sinograms. Since the ill-posed nature of the inverse problem no longer applies to these sinograms, simpler reconstruction become feasible.

Upon validating the effectiveness of this method, the primary focus shifts to evaluating how well the predicted results align with the corresponding ideal sinogram. This can be analysed by visually comparing the quality of both sinograms and, additionally, by applying filtered back projection (FBP) to the predicted sinogram to evaluate how accurately the tumor's position can be reconstructed.

The workflow for each configuration of a new dataset (e.g., Fig. 4.1) can be analysed to understand the specific setup. These workflows are all constructed based on the core structure delineated in Chapter 2. While the fundamental process of training and subsequent testing with new test images remains largely consistent, the nature of the data utilized and its preparation methods vary. Detailed explanations and illustrations of all types of input data are provided in Chapter 3.

4.1 Configuration 1: FMI-Data + ISP-Label

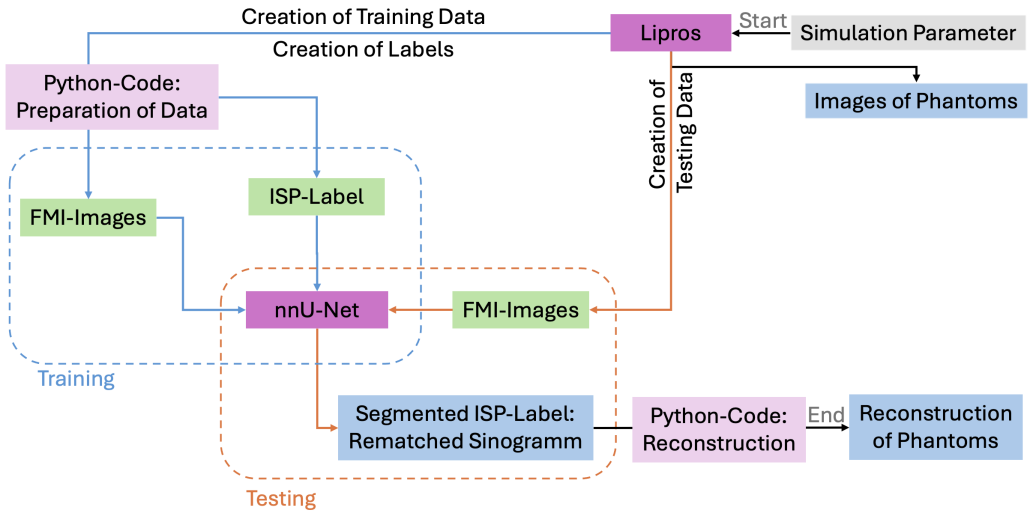


Figure 4.1: Workflow for this setup

Initially, the primary objective was to determine the feasibility of the conceptual framework. After investing significant time in experimenting with simulated data and configuring images to align with nnU-Net requirements, the first dataset was created. All attributes of this dataset meet nnU-Net guidelines.

As illustrated in the workflow (Fig. 4.1) for this initial model, single-tumor FMI images were utilized as input, with ISP images serving as corresponding labels (see figure 4.2).

Following training and initial prediction with this dataset, a major success was achieved (referring to figure 4.2). The core concept upon which the entire research was based proved to be effective, demonstrating remarkable performance. Visual assessment of segmentation quality, comparing the output image with the label, revealed that the predicted sinograms ('Network Output') and reconstructed phantoms ('Reconstructed') show high quality and alignment with the ideal labels and phantoms. Despite slight differences in the last row, where the tumor is located in the center of the body, this approach represents a notable improvement, because accurate localization of deeply embedded tumors is typically challenging, as was deduced in the motivation of this paper (see section 1.1).

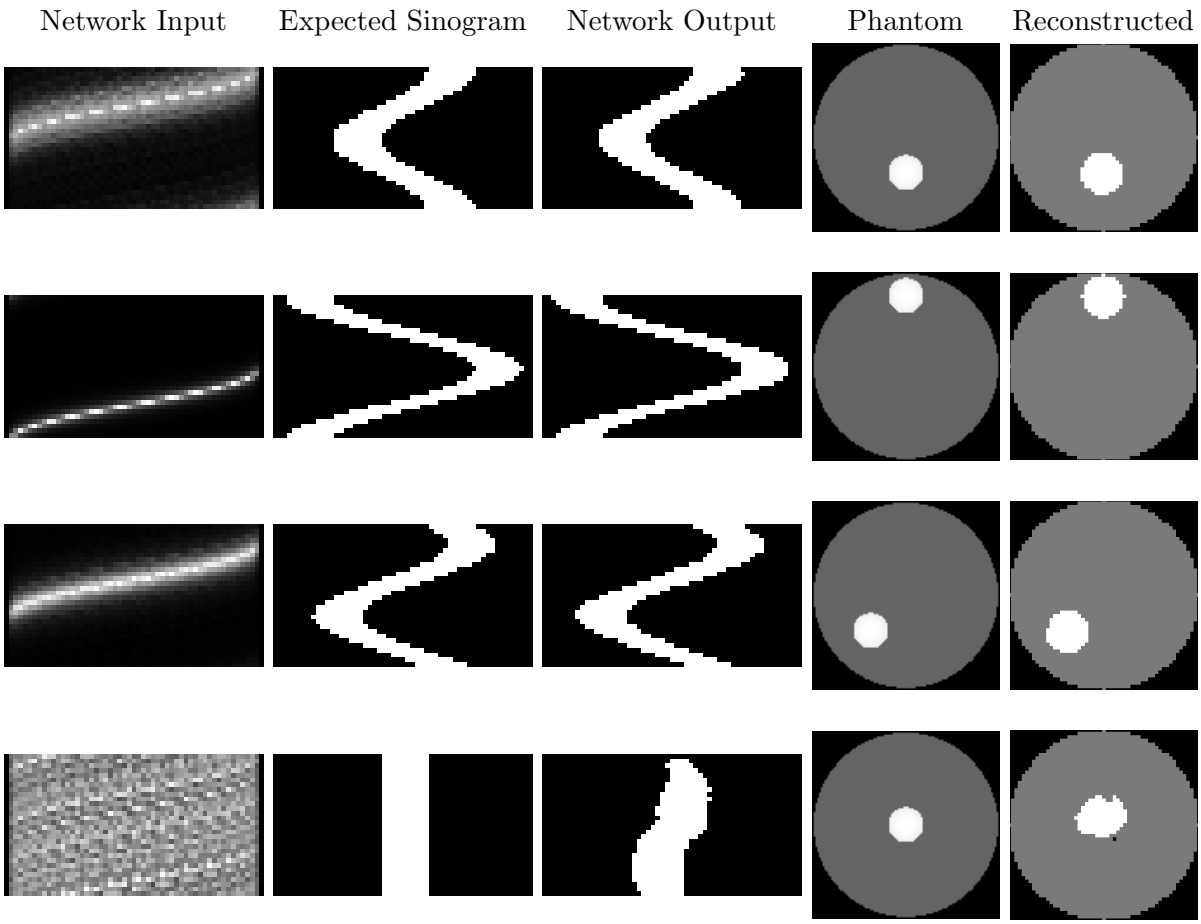


Figure 4.2: Configuration 01

4.2 Configuration 2: FMI-Data + CT-Label

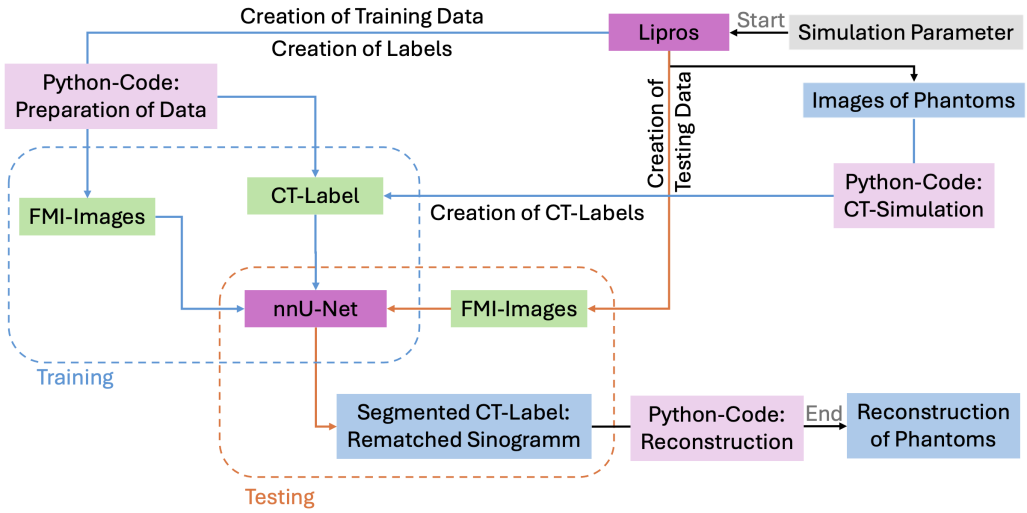


Figure 4.3: Workflow for this setup

After validating the core framework, efforts were directed towards further development. It became evident that while the framework excelled with **simulated ideal ISP images** as labels and **noise-free FMI images**, this setup would proved impractical for real medical research applications. Recognizing the necessity for more realistic labels, it became apparent that obtaining ideal ISP images in real-world scenarios would be unfeasible. An alternative label was searched, preferably derived from another imaging modality, a modality such as Computed Tomography (CT). Commonly used, easy to generate, efficient to reconstruct, and able to provide anatomical information, CT labels can offer a feasible solution for in vivo research applications.

Observing the output of CT images from the trained network model (see Fig. 4.4) yields satisfaction. Furthermore, when comparing the body-center tumor of the CT configuration with the ISP configuration (see Fig. 4.2), a precise result was achieved for configuration 02.

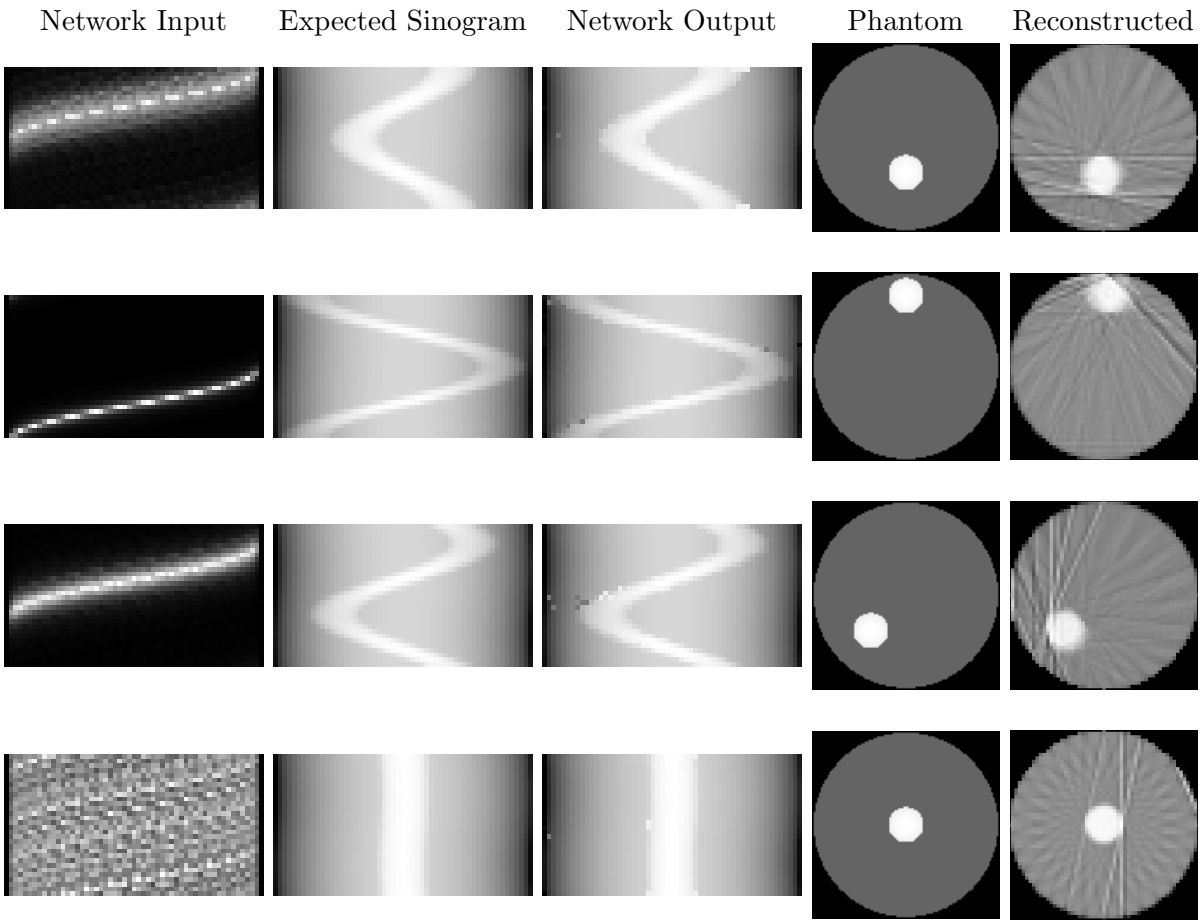


Figure 4.4: Configuration 02

4.3 Configuration 3: FMI-Data + Noise

The generation of real in vivo labels became feasible through CT imaging. Thus attention is directed towards **noise-free FMI images** for further development now. In real-world applications, these images are typically not as devoid of noise as those provided in configuration 01 and 02. Overall any type of noise has been absent so far. Therefore, for the next step, a layer of noise was added to the input images (for both training and testing).

4.3.1 Model 3.1: FMI + Noise + ISP-Label

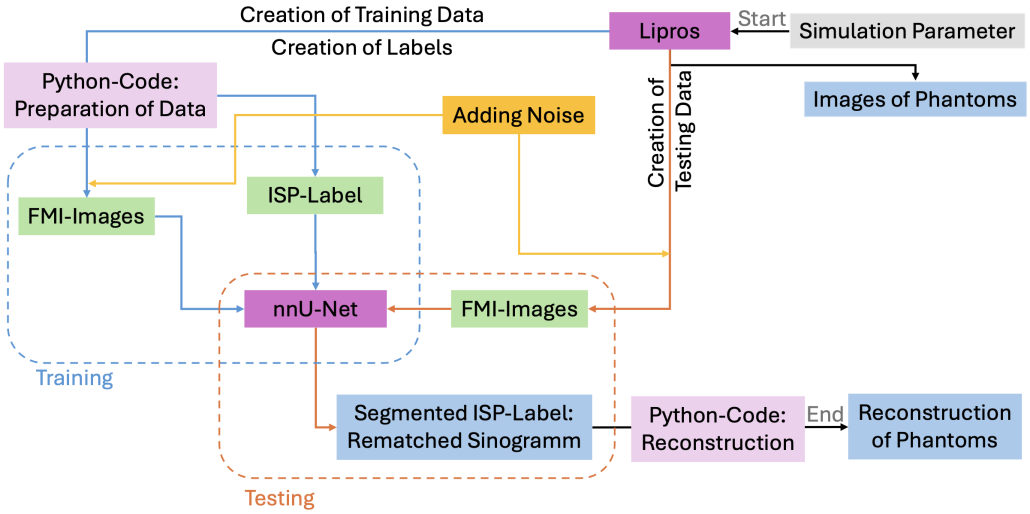


Figure 4.5: Workflow for this setup

The initial dataset for this configuration utilizes ISP labels once more. However, in this iteration, both the training and testing images are noised (refer to figure 4.6). The network deals perfectly with the increase of noise of the input data. Only when the tumor is positioned right in the center, this model is not able to predict the sinogram correctly.

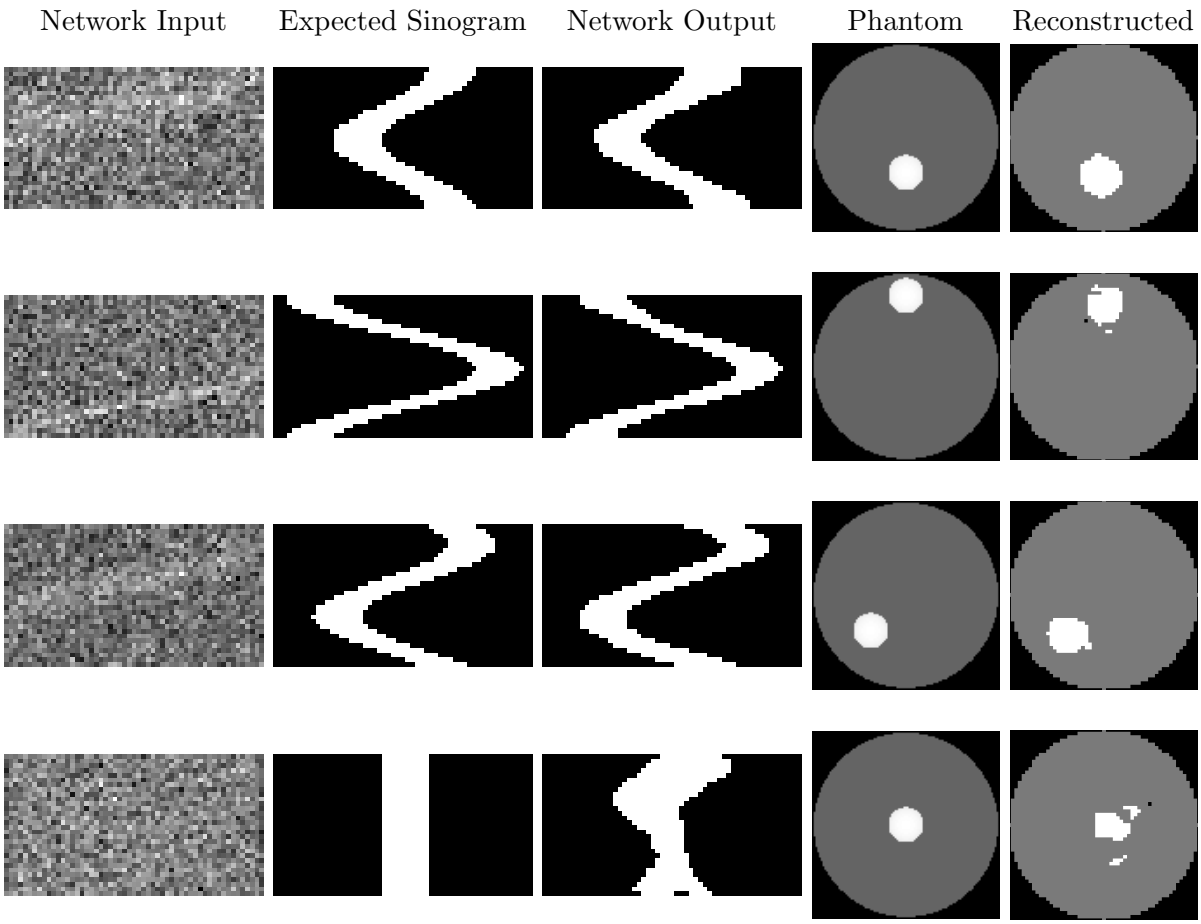


Figure 4.6: Configuration 3.1

Testing Noiseless-Data on "Noised-Images"-Trained Model

To gain a comprehensive understanding of a trained model's performance in relation to noise, two configurations are being evaluated. Testing data of both noised and noiseless images, and models trained on both noisy and noise-free data.

At first the model trained on noisy images was selected for testing using noiseless data (see figure 4.18).

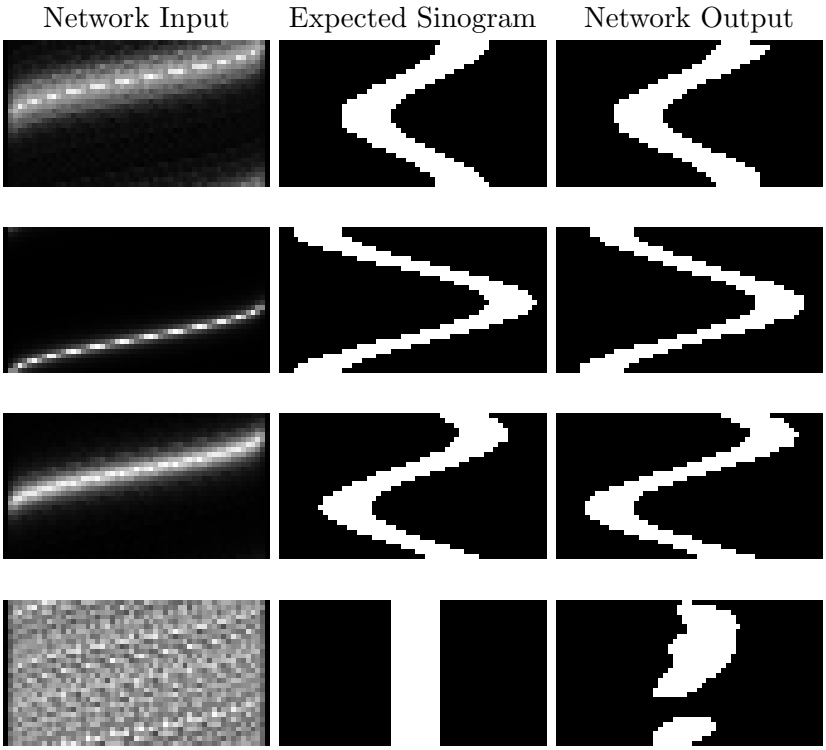


Figure 4.7: Testing Noiseless-Data on "Noised-Images"-Trained Model

This demonstrates that, despite being only trained on noisy data, the model is capable of generating quality images from noiseless/clean testing data. It indicates a level of adaptability and flexibility within the model.

Testing Noised-Data on "Noiseless-Images"-Trained Model

Secondly, the model trained on noiseless images was selected for testing using noised data (see figure 4.8).

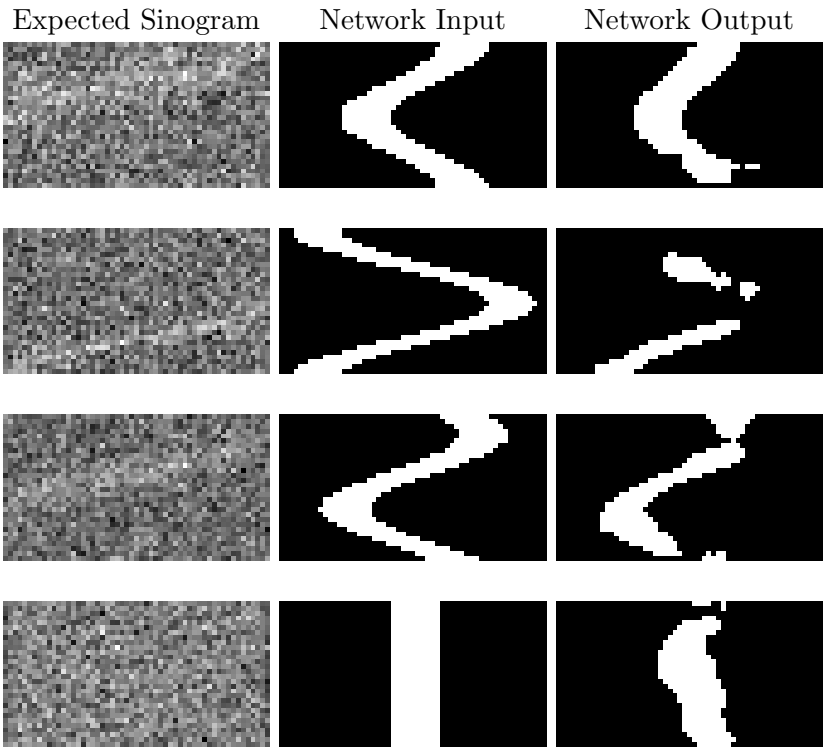


Figure 4.8: Testing Noised-Data on "Noiseless-Images"-Trained Model

As expected, the results of the testing are unsatisfying, displaying a high degree of errors and missing information. This is understandable, as noise in the data increases the complexity of the problem to be solved. Consequently, a network that has not been specifically tested on the "most difficult case" cannot predict accurately.

4.3.2 Model 3.2: FMI + Noise + CT-Label

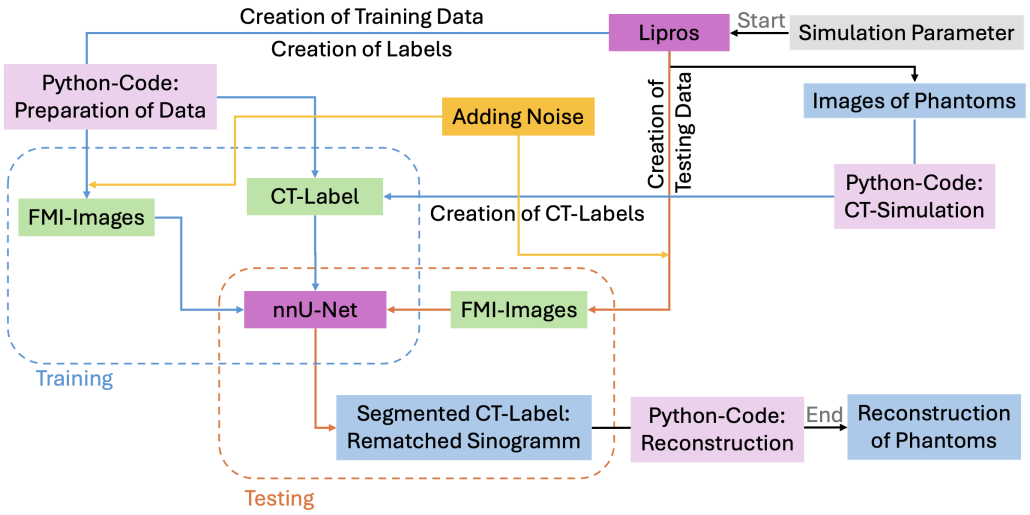


Figure 4.9: Workflow for this setup

Drawing a preliminary conclusion, both attributes were combined for testing. Only if noise-affected FMI images trained on CT labels yield complete CT sinograms, can the potential utility of this framework be further enhanced (see figure 4.10).

Comparing the images in figure 4.10, it becomes evident that this application is promising for yielding further positive results. Even after applying the 'ArtefactFilter' discussed in section 3.5, the predicted CT sinograms and reconstructed phantoms show small artefacts to some degree. Those could be even further remedied by improving or adding another artefact-filter to the process of reconstruction. Nevertheless the filtered reconstruction, can still predict the position accurately. Effort towards improving reconstruction will enable even more accurate reconstruction. Consequently, further testing will be conducted to explore the potential range of application scenarios for the framework.

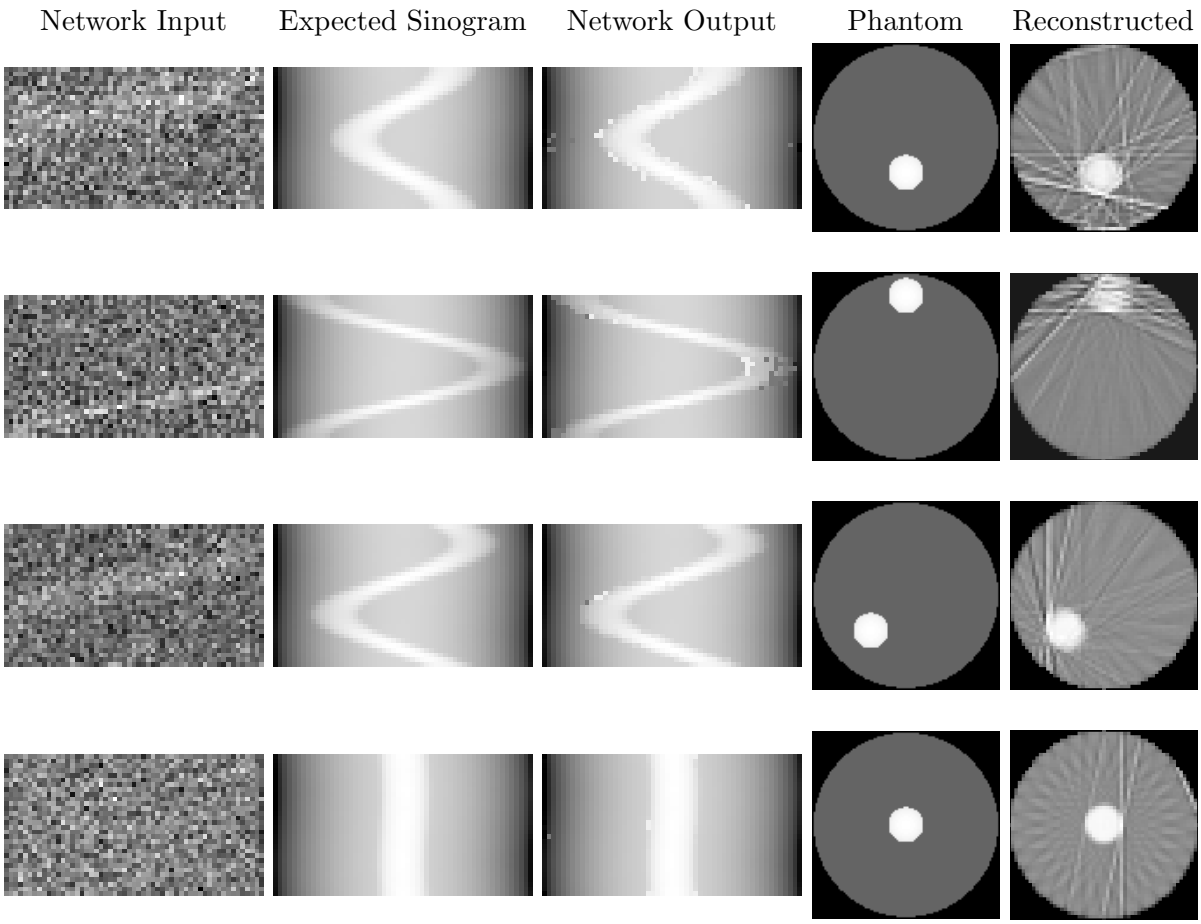


Figure 4.10: Configuration 3.2

4.4 Configuration 4: Two Tumors in one Phantom (addTwo)

Numerous advanced use cases can be considered for implementation. Nevertheless reflecting the motivation behind this work (see section 1.1), two main issues were highlighted. Firstly, it became apparent that the deeper a tumor is situated, the more challenging it is to reconstruct. Secondly, the issue of poor detectability of multiple tumors was raised. Therefore, as the next configuration, the number of tumor inserts within the body will be modified. Consequently, the superposition of two FMI images was conducted and trained for three different models. This time the analyses of the results does not include the reconstruction of the phantom slices, but the introduction of **differentiation images** for better comparison.

4.4.1 Configuration 4.1: addTwo + ISP-Label

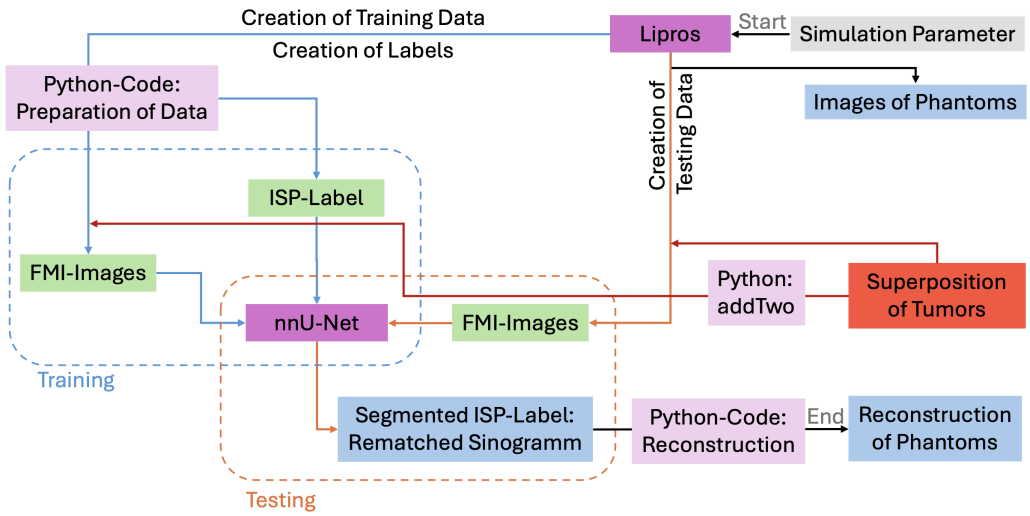


Figure 4.11: Workflow for this setup

Following the same procedure as with the single tumor phantom, the superposition of two tumors (an FMI image) was initially trained with ISP labels. The results proved to be good, displaying quality similar to that of configuration01 (see figure 4.2). This outcome is logical, as the only additional complexity lies in the partial overlap of two sinograms within the phantom.

Comparing the output in figure 4.12 to their ideal sinograms, a good correlation is visible. This can be emphasised through creating a difference image, by subtracting the 'Ideal Sinogram' from the 'Output' (as can be seen in the last column of Fig. 4.12). The **red** pixel represent the space where only the ideal sinogram is present, the **blue** pixel on the contrary mark the space where only the sinogram of the network output is present. The **white** background represents the area where the images of expected sinogram and

network output match with each other versus the **green** pixel, representing mismatching regarding the overlapping sinogram outlines.

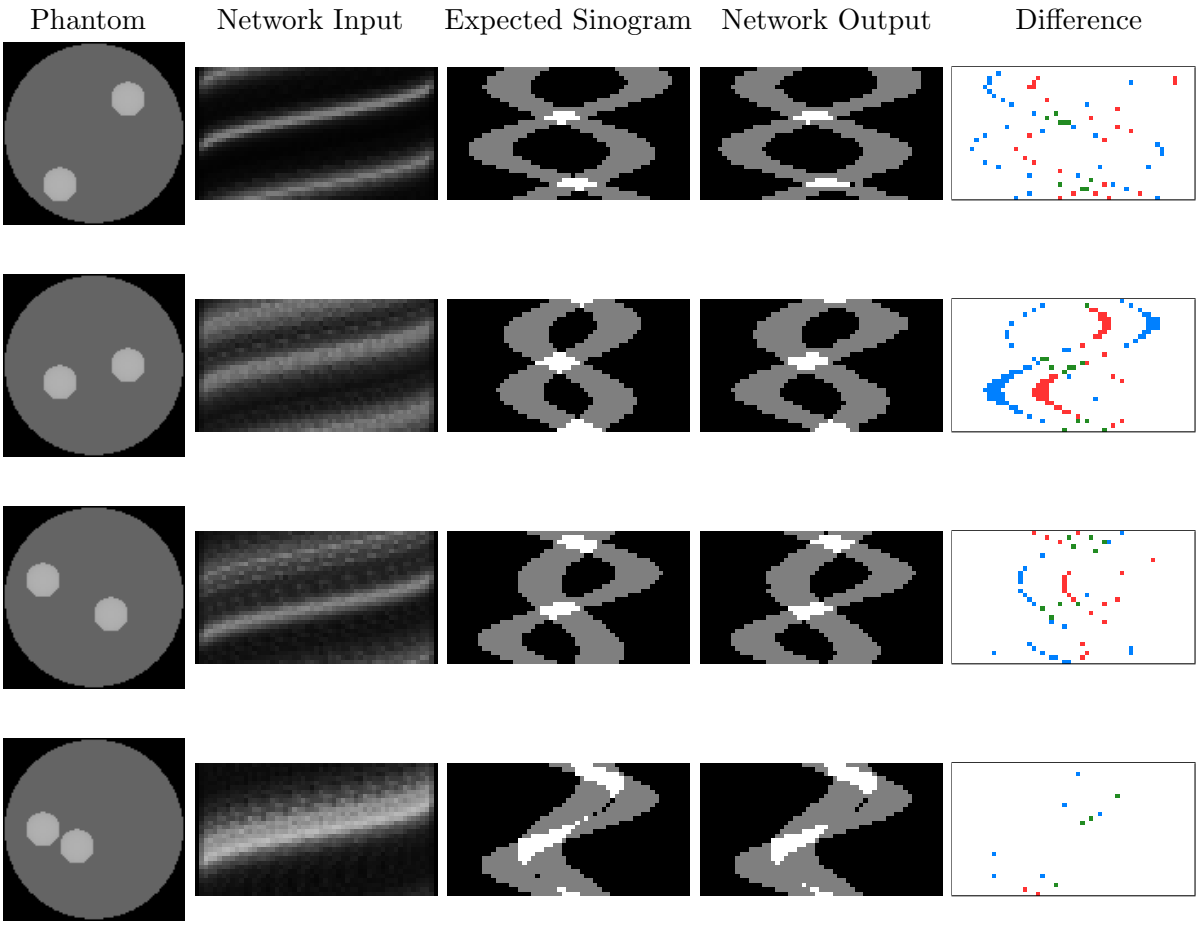
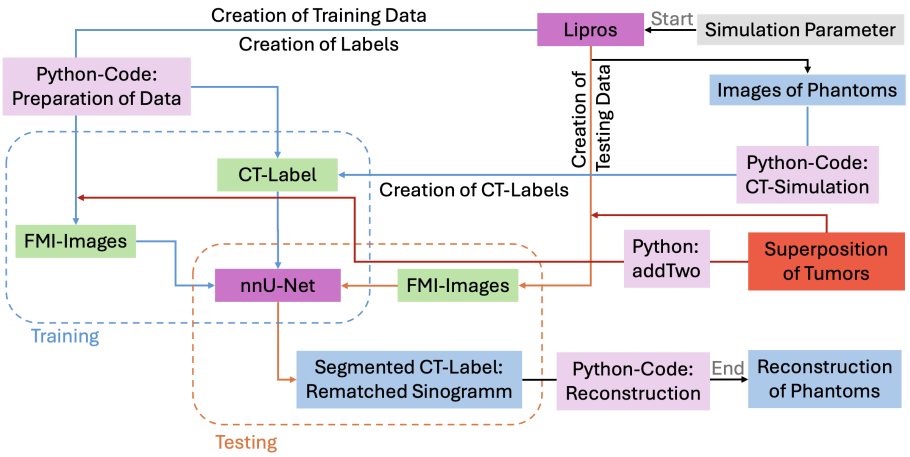


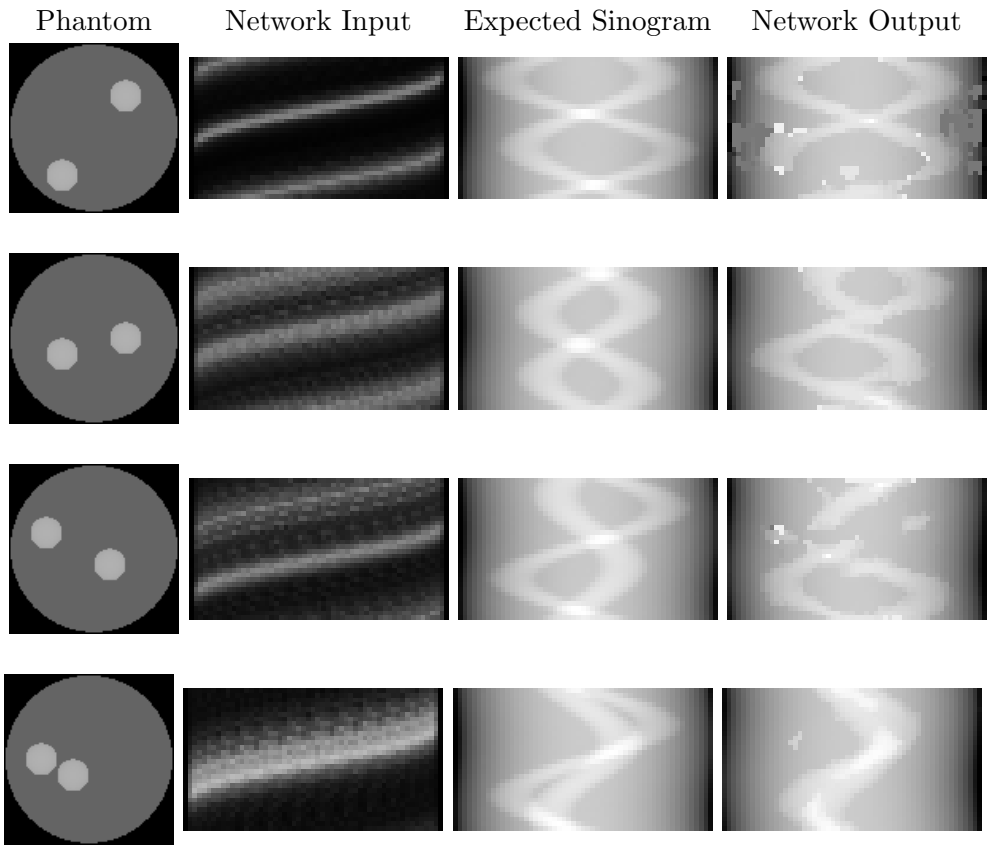
Figure 4.12: Configuration 4.1; Difference Image presenting the area where the images of expected sinogram and network output match with each other (white background), the area where only the ideal sinogram is present (red pixel), the area where only the sinogram of the network output is present (blue pixel) and the mismatching regarding the overlapping sinogram outlines (green pixel).

Looking ahead, having demonstrated the feasibility of this advanced use case, the ultimate goal of this study is to determine whether it is possible to develop a model based on multiple tumors, CT labels, and noised input data. To comprehensively understand both the potential and limitations of this objective, it is divided into various sub-configurations, each integrating a new form of model sequentially.

4.4.2 Configuration 4.2: addTwo + CT-Label

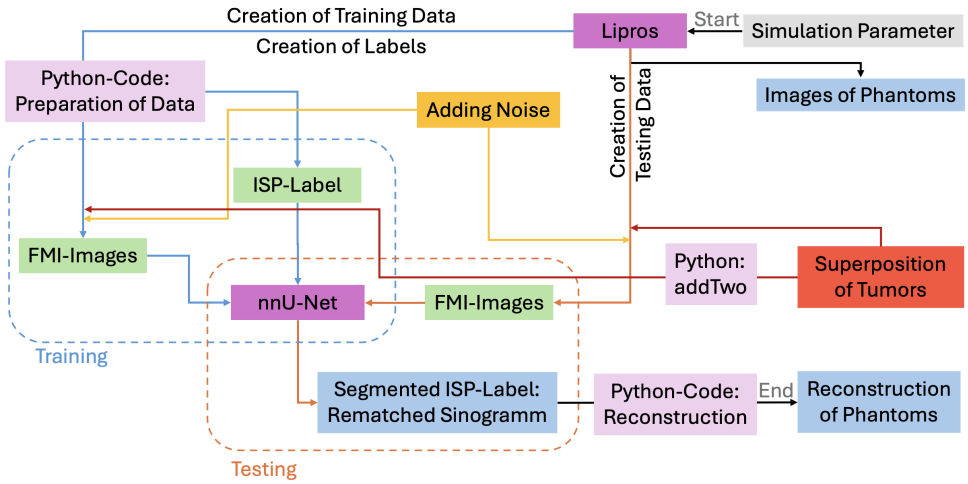


Continuing the same approach as with single tumors, the next step involves exchanging the ISP labels to CT labels for the dataset (see figure below).

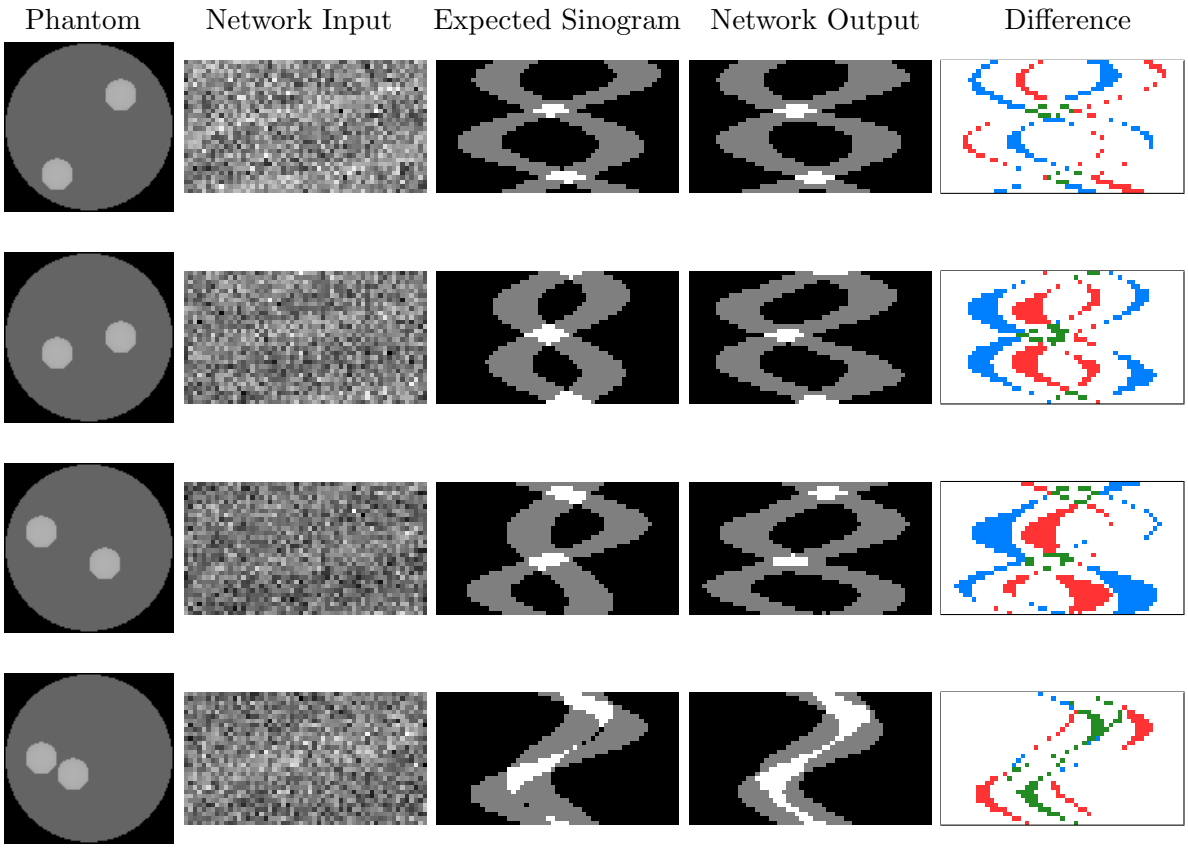


In this configuration, the quality of the predicted data is reduced significantly. More artifacts are present, and there is inconsistency in predicting the shape of the sinogram.

4.4.3 Configuration 4.3: addTwo + Noise + ISP-Label



Instead of using CT labels, noise is applied to the input images, while the labels remain ISP images.



Although the results of the Noise-ISP configuration (4.3) is of better visible quality than the CT configuration (4.2), the misprediction of the shape outline is evident in both. However, it is better visible in the Noise model, particularly due to the absence of distracting artifacts and can therefore be displayed using a difference image (see right column above). The legend for the Difference Image is displayed in Fig. 4.12.

4.4.4 Configuration 4.4: addTwo + Noise + CT-Label

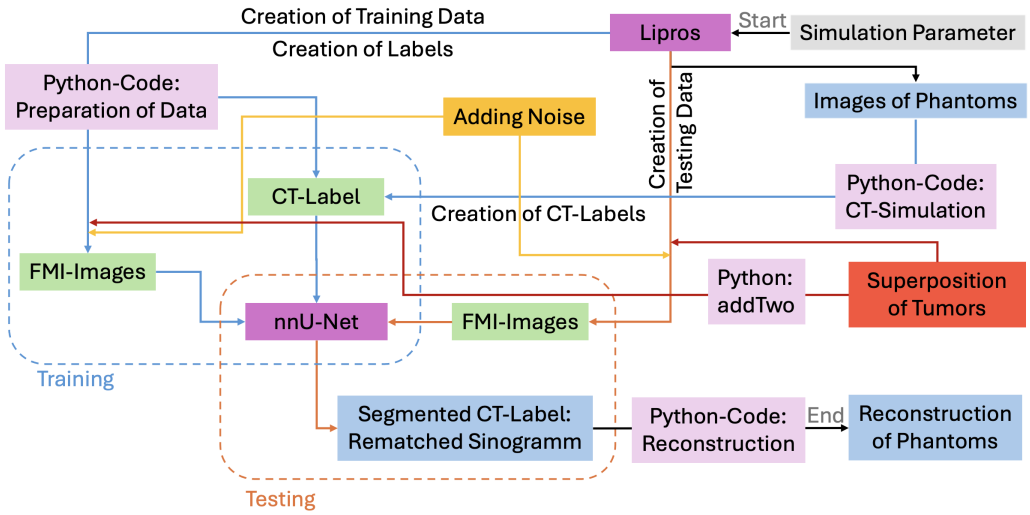


Figure 4.15: Workflow for this setup

In the final round, all three attributes tested separately are now intended to be trained together to evaluate the performance of this multi-attributed network (see figure 4.16).

Upon looking at the final composition 4.16, the malfunction characteristics of the last two configurations became apparent. The integration of all attributes presents a significant increase in complexity that the current network cannot effectively manage. This serves as a limitation that may not be permanent but requires further refinement through adjustments to the input dataset and potentially the network parameter settings.

It is apparent that the network effectively predicts detailed sinograms (as proved for single-tumors 4.4). The misprediction of the configuration now only clearly visualizes that its model was not trained on enough variation in the images, in order to draw accurate conclusions from the input images. Consequently, addressing the misalignment of the predicted shape underscores the necessity for additional training data. This requirement becomes especially significant, considering that, unlike a single tumor within a body, the overlay of any two phantoms exponentially multiplies the potential arrangements of tumors thus possible types of sinograms.

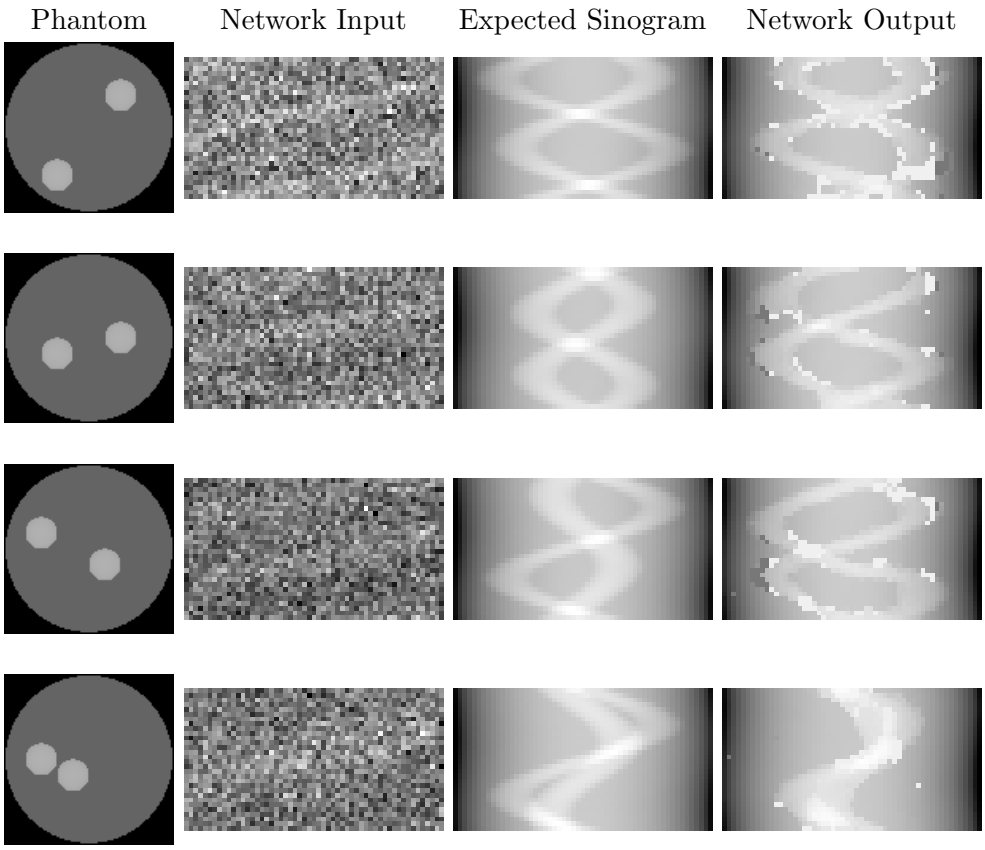


Figure 4.16: Configuration 4.4

4.5 Performance of nnUNet

Each testing dataset (FMI - clean and noised) comprised the same images, with variations based on the configuration type. While the training images (FMI - clean and noised) and labels (ISP and CT) differed in the quantity available for each attribute, the training process of any training session took approximately the same amount of time (around 4 hours as seen in Table 4.1). This consistency in training time can be attributed to the small size of each individual image and the relatively minor differences in the number of images, as training sets for other scenarios can often be much larger.

However, it's worth noting that if the output results, such as those for CT training, are not as successful, increasing the dataset size significantly may potentially improve the quality of the results.

Configuration	ISP	CT	ISP-noise	CT-noise
number of images in training data	32	13	32	13
runtime for training [h:min]	3:58	3:55	4:12	3:56
Configuration	add2	add2-CT	add2-noise	add2-CT-noise
number of images in training data	60	32	60	32
runtime for training [h:min]	4:16	4:12	4:09	4:16

Table 4.1: Evaluation of network performance according to amount of training data and training runtime (training either 'all-fold' or one single fold)

A network's performance can be analysed after training by evaluating its metrics on the tumor label. To provide an overview, here are brief explanations of those metrics (further details can be found in the Appendix B). In this context, "**positive**" refers to the sinogram being appropriately rematched to a complete sinogram, while "**negative**" indicates the occurrence of artifacts and the misalignment of the predicted sinogram with the corresponding label. **Recall (sensitivity / true positive)** measures the proportion of actual positives correctly identified by the model. Contrarily, the **true negative rate (specificity)** measures the proportion of actual negative cases correctly classified as negative. A high value (close to 1) for both metrics suggests that the model effectively captures most positive and negative cases. Similar high values are desired for dice and precision. The **Dice coefficient** measures the similarity between predicted and true positive regions (tumor labels), with a value closer to 1 indicating a high degree of similarity. **Precision** indicates that the model's positive predictions are mostly correct. The only metric that should ideally be low (close to 0) is the **false negative rate**. This rate measures the proportion of actual positive cases incorrectly classified as negative, with a low value indicating that the model correctly identifies most positive cases.

In typical segmentation tasks, each attribute class that is supposed to be segmented gets

its own specific label assigned. In this scenario, these specific labels include the tumor in sinogram shape and the background. The values of these labels must be assigned before training the network. For ISP labels of a single tumor, this task was straightforward. Since they were binarized, the labels chosen were simply 0 for background and 1 for the tumor sinogram. Based on this classification, the network's performance can be analysed after training by evaluating its metrics on the tumor label. Therefore, the network performance for the first configuration, which includes **ISP labels** and **noiseless data**, as well as the third configuration, which also includes **ISP labels** but utilizes **noised data** instead, is evaluated at first 4.2.

Statistics	ISP-noiseless	ISP-noised	addtwo-noiseless
Precision	0.922	0.826	0.886
Sensitivity/TruePositive	0.922	0.836	0.898
Specificity/TrueNegative	0.982	0.959	0.999
False Negative Rate	0.078	0.164	0.102
Dice	0.984	0.831	0.892

Table 4.2: Metrics for the network performance of "ISP-noiseless" (conf01; DS32), "ISP-noised" (conf3.1 - DS37) and "addtwo-noiseless" (conf4.1; DS38)

In contrast, selecting labels for CT-Labels and "AddTwo"-Images was not as straightforward. Here, the chosen values for the labels did not address a single class, such as tumor or organ. When examining the CT-label, the entire sinogram consists of different pixel values corresponding to detailed anatomical information of the CT image. Consequently, there are 255 assigned labels, each representing a pixel color, allowing for the creation of the entire sinogram by predicting these different pixel values. Therefore, analysing any metric in this case is **not indicative**, as these instances do not adhere to the conventional method of semantic segmentation. Thus, it is even more interesting that this implementation proves successful. Hence, this is what metrics might resemble for any dataset trained on CT data:

Statistics	CT-noiseless	CT-noised	addtwo-CT-noised
Precision	0.138	0.102	0.035
Sensitivity/TruePositive	0.138	0.009	0.027
Specificity/TrueNegative	0.997	0.997	0.997
False Negative Rate	0.868	0.907	0.972

Table 4.3: Metrics for the network performance of "CT-noiseless" (conf02; DS35), "CT-noised" (conf3.2; DS30) and "addtwo-CT-noised" (conf4.4; DS40)

An examination of Table 4.2 reveals that the model's segmentation performance is notably high. In essence, all metrics, except for the desired false negative rate, approach a levels close to 1. In summary, based on the calculated values and given the training and test cases, it appears that the model is performing well, demonstrating its effectiveness

in correctly classifying instances and achieving a balanced trade-off between minimizing false positives and false negatives. When comparing training with noise to training without noise, some degradation is observed, which is to be expected. The same trend applies when comparing "addtwo-noiseless" to "ISP-noiseless". Both datasets are noiseless, but "addtwo-noiseless" includes additional complexity due to the overlapping of parts of the superpositioned tumor sinograms. Nevertheless, both "ISP-noised" and "addtwo-noiseless" still exhibit reasonably good performance.

In assessing the network performance of any CT-trained model, only the true negative metric, specificity, offers an expected value. It indicates the model's effectiveness in capturing most negative cases. However, as previously explained, analysing metrics for CT models is not expedient. It is worth repeating that despite this limitation, the quality of the predicted image demonstrates good performance. Visual quality assessment is enough to confirm this statement.

Analyse Learning Curve

By analysing the learning curve in figure 4.17, the training process can be evaluated. The blue line represents the training loss, indicating how well the model is fitting the training data. The red line represents the validation loss during training, indicating how well the model fits new data. For evaluating the graphs only the understanding of the overall trend of each graph line is important.

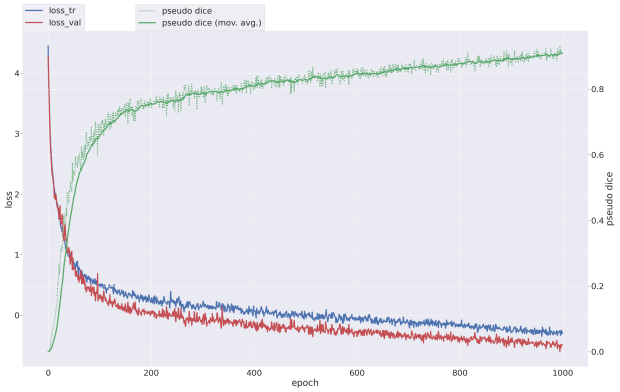
There are two options for setting up the network configuration for training. The first option is to use a 5-fold configuration, where the dataset is divided into five subsets, each used for training and re-validation in turn. The sets used for training and the ones used for re-validation are independent to each other. After training all five folds, the optimal model is selected for prediction. Analysing the progress across the 0-fold (Fig. 4.17a and 4.17b) illustrates the network's ability to be applied to new datasets. Training and validation loss can either align (good learning curve), or differ from each other (bad learning curve - overfitting), various developments are possible. Datasets trained on ISP labels (as shown in Fig. 4.17a) consistently show a favorable learning curve. Conversely, datasets trained on CT labels (as depicted in Fig. 4.17b) display no correlation of training and validation loss. This behavior of the learning curve underscores once more that models trained on CT images cannot be accurately analysed using standard methods such as metrics or learning curves due to the "unconventional method of pixel label assignment" in the training model, as previously described.

The second option is to train the dataset only once, a setup known as "all-folds". In this approach, the data is not divided into subsets, resulting in the network training on the entire dataset very precisely. However, the data used for validation is drawn from

the same set used for training, leading to the validation loss being approximately equal to the train loss. Consequently, the learning curve and resulting metrics do not provide an accurate evaluation (see Fig. 4.17c), as they cannot compare new results to unseen data.

(a) Trained on ISP data, fold0

(b) Trained on CT data, fold0



(c) Trained on all-folds

Figure 4.17: Learning Curve for 1000 epochs, displaying training loss (blue), validation loss (red) and pseudo dice (green).

For this work, each dataset was trained twice: once using the 5-fold configuration (five independent training runs) and once using the "all-fold" configuration, training the entire final model at an instance. Interestingly, while the 5-fold configuration offered insights into the performance evaluation of the training process, it was found that the "all-fold" configuration actually yielded **better results** in terms of visual quality when testing any trained model. Hence, throughout the study, all images displayed are predicted using the "all-fold" model. An example highlighting this difference is provided below:

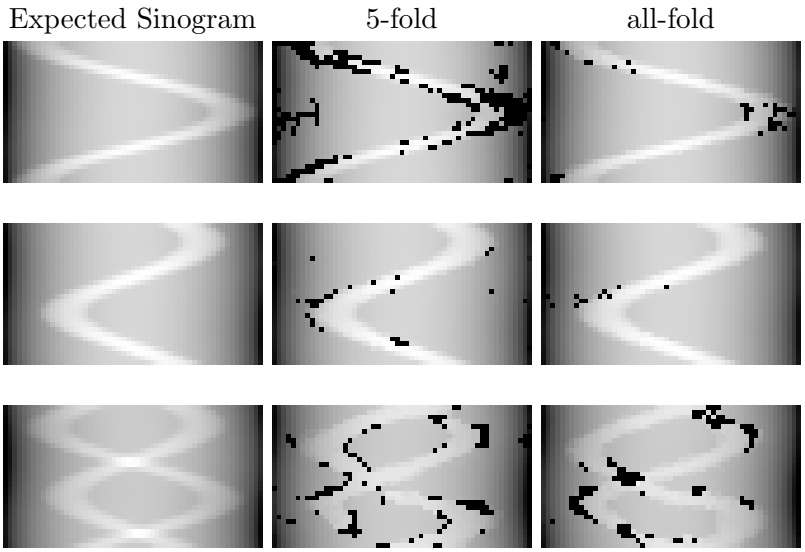


Figure 4.18: Difference of prediction of either 5-fold training or all-fold training

4.5.1 Rating of the Tools

Though initially requiring some time to install, run, and become familiar with nnU-Net's workflow, mastering its operations has made handling nnU-Net not only very pleasant but it even worked much better as expected. Much appreciation is owed to the Applied Computer Vision Lab (ACVL) team at Helmholtz Imaging and the Division of Medical Image Computing at the German Cancer Research Center (DKFZ) for their profound job in developing this 'out-of-the-box' network. nnU-Net represents a significant tool that has to offer invaluable support in AI-driven segmentation research across various domains. Its capability to handle CT-label configurations, not inherently tailored for a specific semantic-class segmentation task, signals promising flexibility for nnU-Net's applications.

Similarly, the efficiency and quality of simulations produced by the 'Lipros' simulation program are truly impressive. **Dr. Jörg Peter** deserves immense recognition as he has put a lot of knowledge and effort into developing and improving Lipros. Implementing simulations within Lipros has been a straightforward task, thanks to Dr. Jörg Peter's realisation of ideas for simulation and support.

Chapter 5

Conclusion and Outlook

5.1 Outlook

Up until now, traditional iteration-based regularization methods have been the common and standard approach for addressing the inverse problem in Fluorescence Molecular Tomography (FMT). They have demonstrated good reconstruction performance in terms of image contrast, positioning accuracy, and spatial resolution. Nevertheless, these methods pose challenges in numerous biomedical applications due to their high computational demands, dependence on prior knowledge, uncertainty of solutions, and significant time investment [49].

Research has been done in the field of deep learning, in order to find a different approach to reconstruct the tomography without those posed challenges. There have been attempts in implementing a deep neural network-based method for solving the reconstruction problem of FMT data (see chapter 1.2.8). But those as well as the conventional, iteration based algorithms all focus on how to improve the reconstruction algorithm based upon the FMT data. This is done by either improving the FMT data through decreasing the errors/artefacts, or feeding the algorithm more information than only the FMT images in order to reduce the unknown parameter. Zang (2022) states, that in general, the accuracy of the photon propagation model can be improved by combining anatomical information acquired from CT or MRI scans in the FMT reconstructions. Up until now, those details are only being used to influence the FMT reconstruction. Long (2018) for example, has used the U-Net architecture as well, focusing on post-processing as he stayed within the range of FMT-images, by training his network to improve the quality of the FMT-Images by reducing the artifacts.

Extracting the two core ideas, using segmentation networks as well as including valuable CT information, the decision was made to take a step backwards, in order to look at the

main source of the problem. If the reconstruction itself is so problematic, why not try to skip this step and move it to an application that is already known and well functioning, such as image reconstruction of a CT image? The concept of CT reconstruction is well known and easily done, as it is already implemented in daily life of medical research and clinical life. It shifts the challenge of FMT handling to the intersection, where those FMT-data need to be transferred to CT data before they can be reconstructed. This is where the use of a segmentation tool, such as the U-Net comes in hand. Of course, it would have also been possible to proceed differently and first reconstruct the FMI sinograms into incomplete phantoms, and then train them with a network to generate complete phantom slices. However, this approach was put aside because it would result in further loss of information and further complicate the problem. Additionally, reconstruction is a separate area of research that can be further improved and adjusted even after the application of the trained network like with this method.

Another benefit of working with a deep learning-based framework instead of an iteration-based algorithm is present in regards of the amount of computation done for each image reconstruction. When applying the iteration-based algorithm for every reconstruction, thus minimizing the loss function, a large amount of computation is needed. On the other hand, when implementing a neural network, the initial training to generate a network model takes some time (here around 4 to 20 hours depending on the training setup), but afterwards for a single reconstruction, the testing - reconstruction - is done instantly.

Of course it is also clear that iterative methods are based on physical and mathematical theory, whereas with deep learning based on statistics and probabilities, weak interpretability comes along. Inside a network any distribution of tasks between the filters and layers occurs automatically and is not predetermined by the network structure. It can vary from application to application, from training to training. It is impossible to know based on what factors the trained models conduct their decisions for segmenting the input images. This is exactly what makes neural networks so appealing for research and fascinating for application. Contrary because of this attribute the "framework" can not be repeated with gaining the exact same results. Which marks a clear problem in medical imaging.
The power of this type of research is that even if there were some limits found, for example spatial resolution of two tumors close to each other, it does not necessarily mean that this is a final limit. It all depends on what type of data the network is trained on. For example a new dataset could be generated, focusing on teaching the network with various images of close tumors of every positional orientation, to solve the spatial resolution issue.

The primary ambition for this work was to effectively verifying the functionality of the investigated method. After successfully doing so, there seem to be no limitations on how to continue exploring its possibilities and limitations. So far, the attributes implemented

have only scratched at the surface of this problem space, which is much bigger. For example looking back at the research done by Long (2018), the view of attention gets drawn to his conclusion that his U-Net solution is only suitable for reconstruction in homogeneous tissue models addressing the disability to reconstruct fluorescent targets in heterogeneous tissues. From rating the results of the method used in this work so far, this increase in complexity should not be an obstacle for it. That is why as a next step of research a new configuration could be implemented, with simulated heterogeneous phantoms. At first it would be a dataset with the same heterogeneous general tissue and only a position changing tumor. Its compatibility with noised input data and CT-labels would have to be tested of course. Afterwards the further investigation of interest would be to see if not only the tumor position could change but also the set up of the heterogeneous tissue, changing the position, size and amount of organs, muscle/air ration etc.

Further development needs to test threshold values and limiting factors for any type of application for this framework. What is the threshold for the amount of noise in the image, up until which the network can still do satisfying predictions? What is the threshold of interaction between smallest tumor size and contrary larger phantom that is still possible to detect? Obviously in a more advanced stage of the research, the type of tumor could be changed, too. Up until now, only a point source, formed as a sphere, represented the tumor. Instead tumor lesions for example, can alternatively be defined using point clouds in Lipros.

Furthermore it could be tested, how the network is dealing with changes in shape of the entire phantom itself. Lipros offers either 3D surface polygonals or 3D triangulation meshes to define more phantom tissues that are more complex than a geometrical cylinder. How is the quality of the resulting predictions of such sinograms? How is the network dealing with different shapes and body outlines in the same training dataset? Is it necessary to keep the same shape for one trainingset? Ideally only one trained model on different sizes, shapes, types of tissues etc. should be needed, other wise for every new proband a new network model must be trained.

An interesting step would be to compare this framework directly to classical approaches of reconstructions. In the end, this is the essential question. How well is this method performing in real life compared to present methods? In this field of action, categorizing, labeling, and preparing real patient data would be of essential use. Applying all these steps, that have been conducted with simulated data, to real data (small animal research or any type of actual patient data) could be a interesting investigation.

5.2 Conclusion

The motivation for this study was to find an alternative approach for overcoming the deduced challenges of FMT (see section 1.1), such as decrease in image quality and limited detection accuracy, that result in an unsuccessful reconstruction. Implementing the "FMT-to-CT-then-Reconstruction" method, a new framework was therefore created. It is supposed to overcome the complexity of solving the ill-posed inverse problem for the FMT by simply transforming the FMI to a CT-Sinogram, which then can be properly reconstructed to the final phantom. This is done by implementing and training a neural network on FMI images corresponding to complete sinograms, that can simply be reconstructed, as labels. Thus to reduce the sources of errors in order to increase the reconstruction reliability and efficiency, it is not necessary anymore to focus research on solving the inverse problem of the FMT but to further improve the transition from FMT to CT, as the details that go missing here, will be also the details that will be missing in the final phantom.

Even though only a small problem space was investigated in this three-month work and there was no perfect transition from FMT to CT, the idea for this new approach was taken up, implemented and actually proved that this framework is feasible. Hence, the doors are opening for many possible research opportunities, by putting the focus not on improving the accuracy and efficiency for reconstruction, but on improving the neuronal network as such it aces the transition from FMT-data to CT-data. Therefore, the boundaries of this attempt need to be investigated. Which parameters of the setup of the neuronal network need to be set and which network architecture needs to be chosen in order to improve the transformation. Other modalities for the training can be found that might improve the efficiency of the pre-trained neural network. Of course, the training data needs to be variable as different shapes, different densities, different sizes etc. need to be taken into account too. To conclude, as it was not possible to find any similar attempts towards this particular approach of reconstructing FMT-Data, it is legitimate to state, that the concept of this approach is valuable, and further investigations are very worth to be considered.

Appendix A

Data Summary

This entire study, including thesis and codes can be found in a GitHub repository named "Enhancing-Fluorescene-Molecular-Tomography-through-Deep-Learning-based-on-nnUNet".

To get a better insight into the results of nnU-Net's predictions, many more examples of the results of the datasets trained and predicted by nnU-Net are shown in this section.

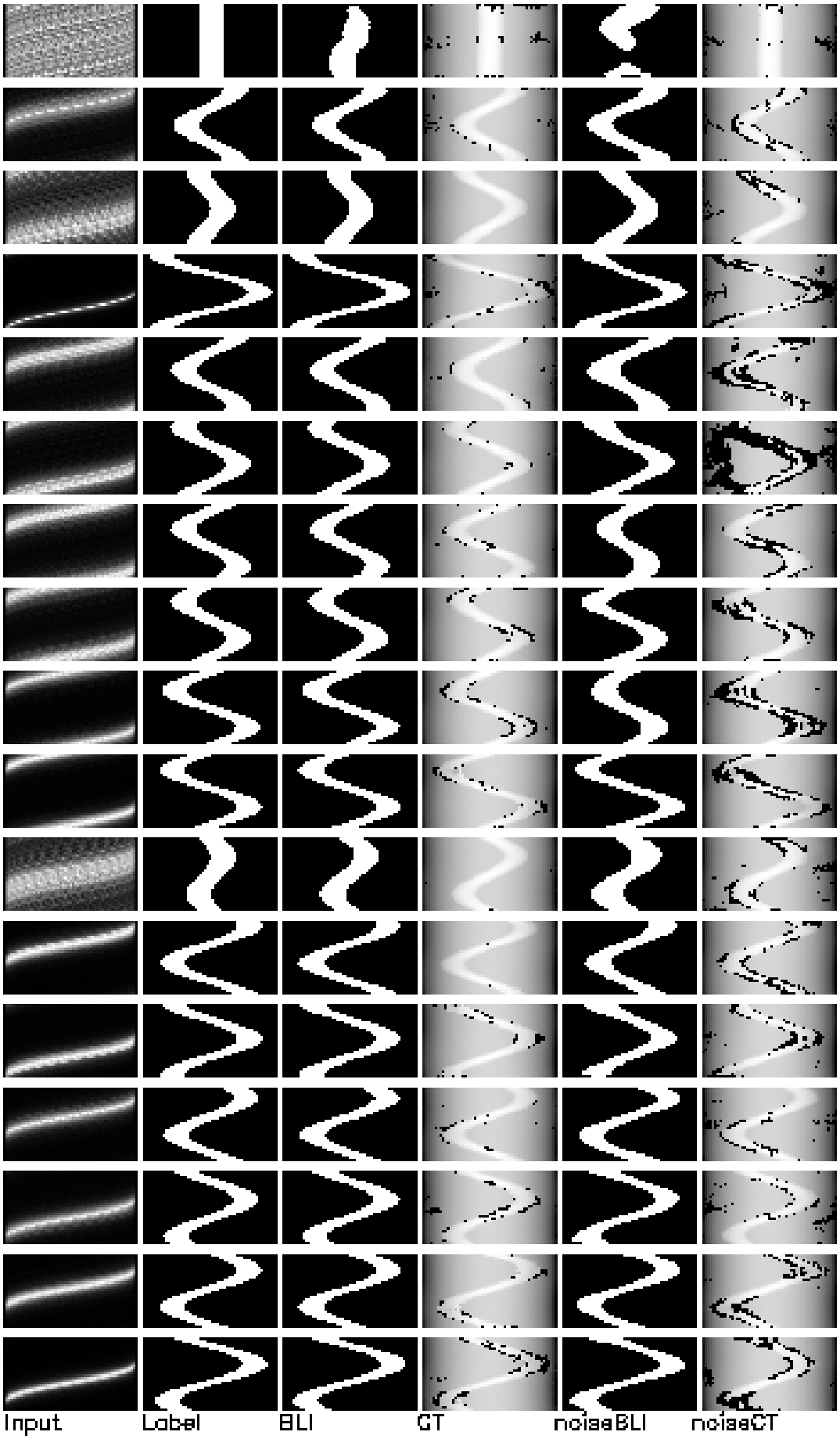


Figure A.1: single tumor, fold 0-4 trained

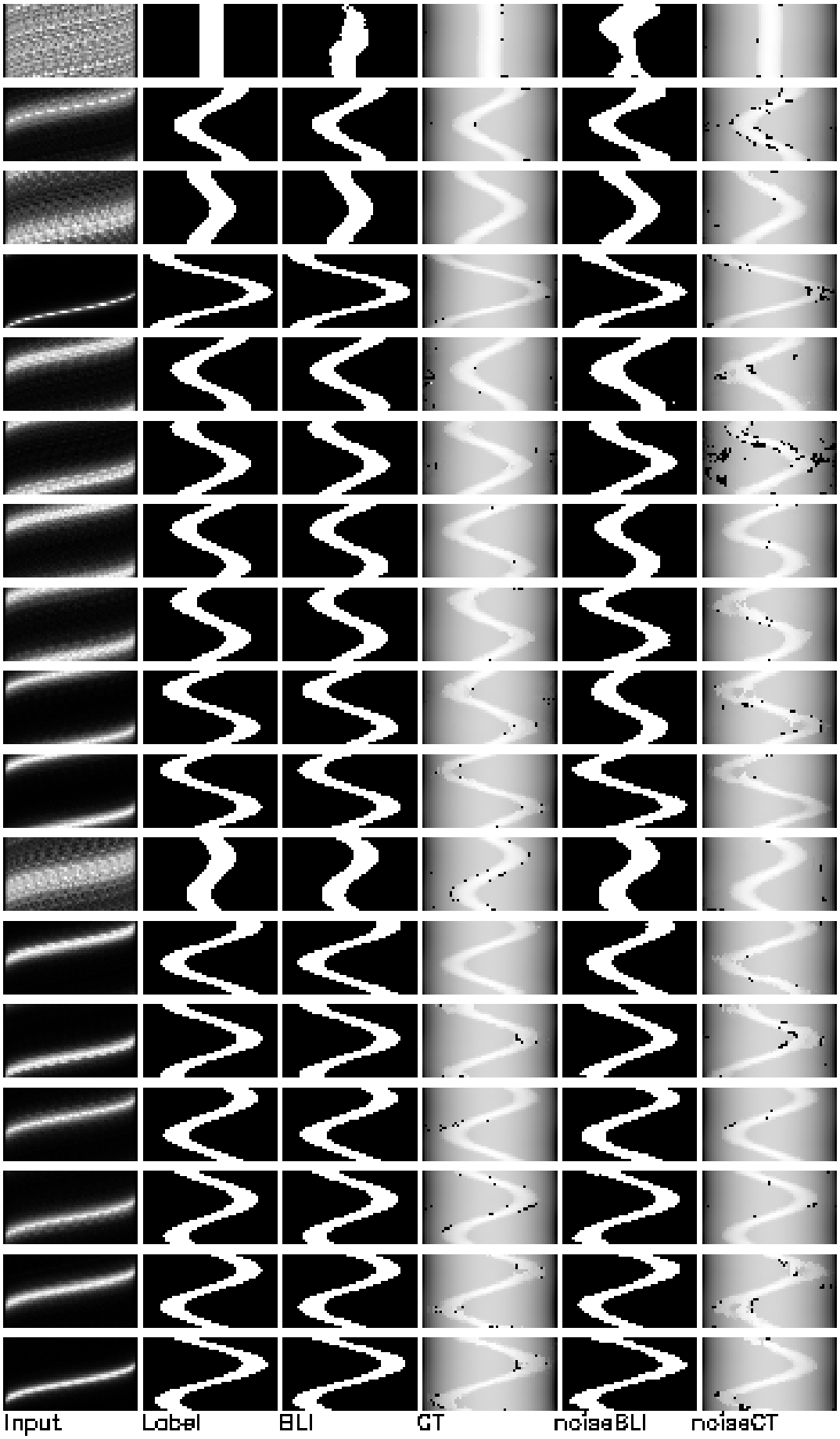


Figure A.2: single tumor, all-fold trained

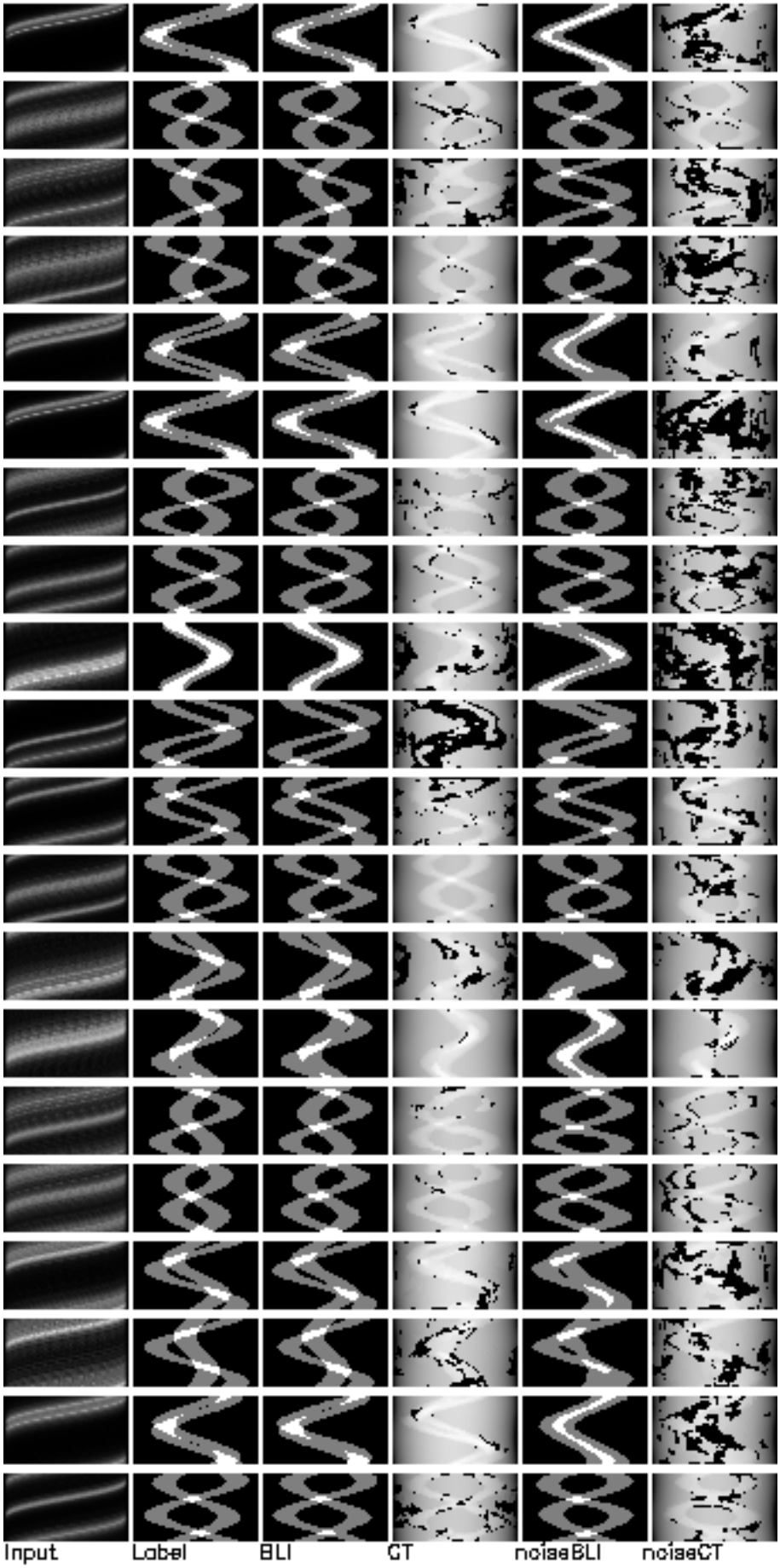


Figure A.3: multiply tumors, fold 0-4 trained

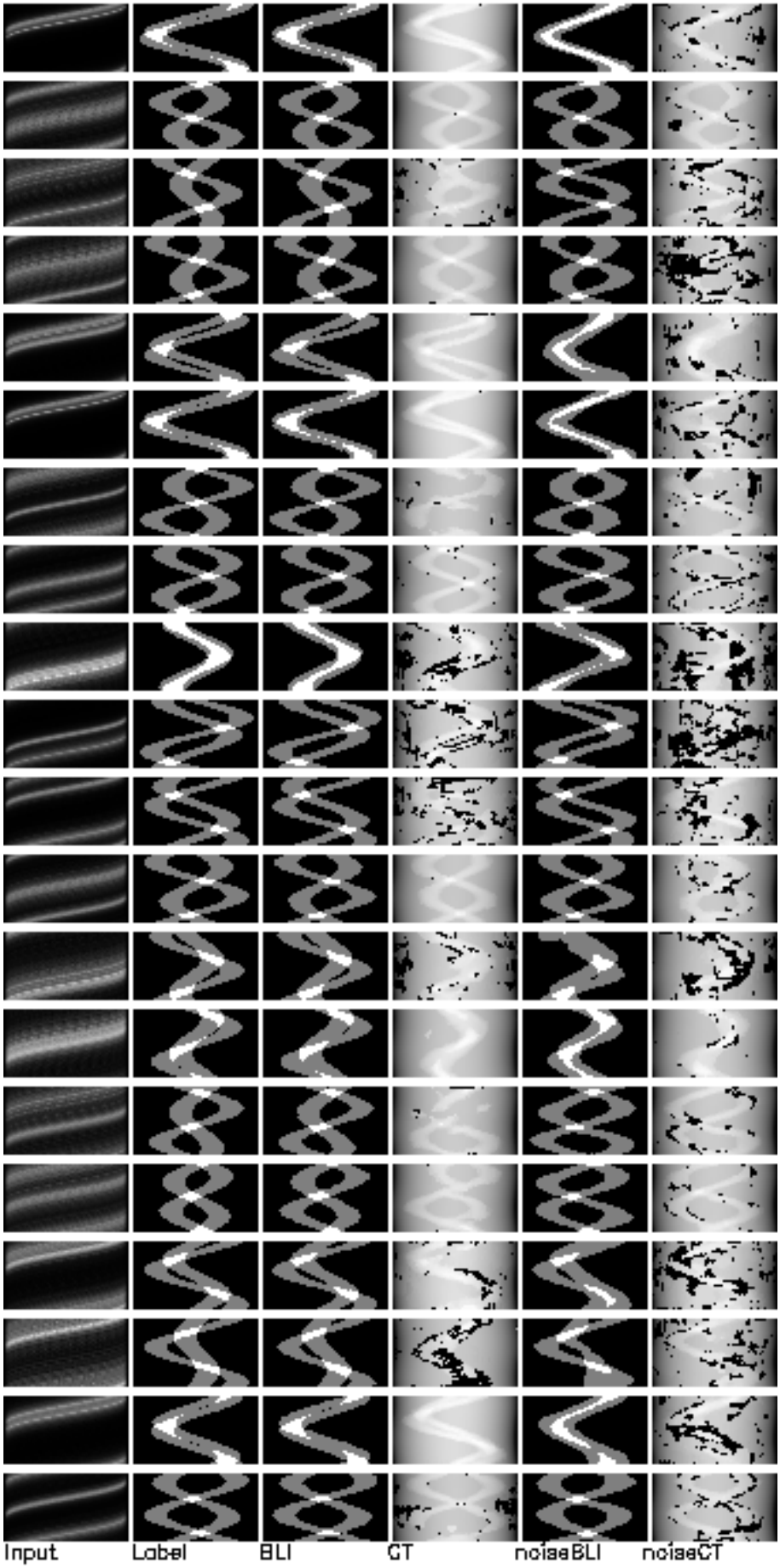


Figure A.4: multiply tumors, all-fold trained

Appendix B

Analysis tools for the evaluation of network training

B.1 Performance of Segmentation - Review

After the segmentation is done through testing the network model, the performance of the segmentation model should to be evaluated. This evaluation should be done in multiple respects, such as quantitative accuracy, speed (inference time), and storage requirements (memory footprint). But as the segmentation is dealing with images the value of visual quality assessment should not be put aside either. The speed depends on the data size and experimental conditions, whereas the storage is based among others on the setup of the data. Only the accuracy of segmentation algorithms can be assessed by applying certain metrics.

These values can be used to calculate various performance metrics such as accuracy, recall, precision, false negative rate, true negative rate, F1-score and dice, by using the following formulas ([25], [**p.35f**, 8]).

Accuracy = $\frac{True}{allCases} = \frac{TP+TN}{TP+TN+FP+FN}$
Accuracy measures the ratio of correctly predicted instances among the total instances. Generally, an accuracy above 0.9 is considered very good.

Specifically suitable for binary classification tasks (with the classes "positive" and "negative") are the metrics: precision, sensitivity, and specificity. They can be expressed through the true and false "positive" as well as true and false "negative" classification results.

Precision = $\frac{TP}{TP+FP}$
Precision measures the proportion of true positives among the instances that are predicted as positive by the model. A high value, indicates that the model's positive predictions

are mostly correct.

When examining patients for a disease for example, sensitivity is the ability of the method to yield a positive result for the person that has the disease. Sensitivity on other hand is the ability of the technology to obtain normal range or negative results for a person who does not have a disease. Highly sensitive tests will lead to positive findings for patients with a disease, whereas highly specific tests will show patients without a finding having no disease.

Recall/Sensitivity = $\frac{TP}{TP+FN}$
Recall measures the proportion of actual positives that are correctly identified by the model. A high value, suggests that the model is effectively capturing most of the positive cases.

Specificity / True negative rate = $\frac{TN}{TN+FP}$
The true negative rate (specificity) measures the proportion of actual negative cases that were correctly classified as negative. A high true negative rate (close to 1) is desirable, as it indicates that the model is correctly identifying most of the negative cases.

False negative rate = $\frac{FN}{TP+FN}$
The false negative rate measures the proportion of actual positive cases that were incorrectly classified as negative. A low false negative rate (close to 0) is desirable, as it indicates that the model is correctly identifying most of the positive cases.

Usually a combined version of precision and recall rates is interested to look at. A popular metric is called the F1 score, which is defined as the harmonic mean of precision and recall, providing a balance between the two metrics.
F1-score = $\frac{2*Precision*Recall}{Presision+Recall}$
A high F1-score (close to 1) indicates good performance in both precision and recall.

Dice coefficient is another popular metric for image segmentation. It can be defined as twice the overlap area of predicted and ground-truth maps, divided by the total number of pixels in both images:
Dice = $\frac{2*TP}{2*TP+FP+FN} = \frac{2|A \cap B}{|A|+|B|}$
The Dice coefficient measures the similarity between the predicted and true positive regions. A value closer to 1 indicates a high degree of overlap between the predicted and true positive regions, a desirable output.

Intersection over Union (IoU) or the Jaccard Index is defined as the area of intersection between the predicted segmentation map and the ground truth, divided by the area of union between the predicted segmentation map and the ground truth:
IoU= $\frac{|A \cap B|}{|A \cup B|}$
where A and B denote the ground truth and the predicted segmentation maps, respec-

tively. It ranges between 0 and 1. As this coefficient is very similar to the Dice it is neglected in the further analysis.

B.2 Analysing the Progress during Training

If the test data is included in the trained data, it is referred to as training loss. If new data is tested that the network had not previously trained on, this is called validation loss. It provides an estimation of the error on unseen data. Examining the loss throughout the training iterations can be useful. A rapid increase in loss early on often indicates that the learning rate, denoted as η, is set too high, a phenomenon known as the exploding gradient. Conversely, setting η too low may lead to a stagnation of loss over iterations, resulting in vanishing gradients. Therefore, the precise selection of η and other training hyperparameters is essential for effective training [23].

B.3 Overfitting

Apart from the training set, a validation set serves to detect overfitting. Unlike the training set, the validation set is not utilized to adjust the parameter weights. Thus, the loss of the validation set provides an estimation of the error on unseen data. While the loss on the training set decreases consistently during optimization, the loss on the validation set may begin to rise at a certain stage of training. This usually indicates a suitable point to stop model updates to prevent overfitting to the training data [23].

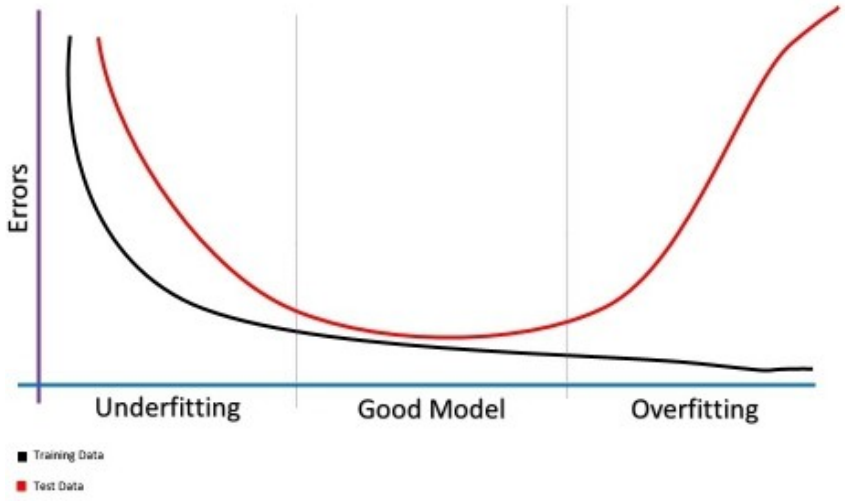


Figure B.1: Display of validation and training loss curve, referring to concepts of under- and overfitting [4]. The black curve represents the Training Data and the red curve represents the Test Data.

Appendix C

Abbreviation

FMI	Flourescence molecular Imaging
FMT	Flourescence molecular Tomography
BLI	Bioluminescence Imaging
ISP	Ideal Source Project without attenuation and scattering
CT	Computer Tomography
MRI	magnetic resonance imaging
PET	Positron Emission Tomography
SPECT	Single Photon Emission Computed Tomography
OMI	optical molecular imaging
MI	molecular imaging
CNN	Convolutional Neural Network
UNet	Convolutional Networks for Biomedical Image Segmentation
DDN	deep neural network-based
ETE-DNN	end-to-end deep neural network
FC	fully connected network
LC	local connected network
KNN-LC	K-nearest neighbor–local connection network
FCN	Fully convolutional network
nnUNet	no-new UNet
FBP	Filtered Back Projection
GFP	green fluorescent protein
NIR	near-infrared light
RTE	radiative transfer equation
DE	diffusion equation
MC	Monte Carlo

FBP	Filtered Back Projection
SVD	singular value decomposition method
TV	total variation regularization
RC	Rayleigh Criterion
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
SNR	Signal to Noise Ratio
RMS	root mean square
AI	Artificial Intelligence
DL	Deep Learning
ML	Machine Learning
ReLU	Rectified Linear Unit
LReLU	Leaky Rectified Linear Unit
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Appendix D

Acknowledgement

First and foremost, I would like to express my gratitude to my supervisor, Dr. Jörg Peter. I think there is no other advisor to find with whom the spectrum of conversation can range from Kandinsky's art over sacrificial art in Finland to self-crafting record players, from the art of ligature to the concept of correct sequencing of biblical chapters, all while seamlessly weaving back to the fields of medical physics and machine learning. Thank you for all the time you spend and effort you took on helping me through the past months.

A heartfelt thank you to Prof. Dr. Mark E. Ladd for being my primary examiner and to Prof. Dr. Oliver Jäkel for agreeing to serve as the second examiner. My gratitude goes also to Sabine Fritz, who handled all the administrative tasks at DKFZ.

I also want thank Daniel Richter and Jessica Silas for the hours we spent at the blue couch talking and supporting each other. Thank you to Pablo, Consti, Beat, Lennart, Flo, Eric and Hannah for our profound discussions! My family gave me the emotional support I needed (obviously throughout my entire degree). And my Rugby team played a vital role in keeping me sane and reminding me to take breaks. I am grateful to the family Firnhaber for providing me with a place to stay in Heidelberg, and to the team at Energeering for their flexibility and interesting tasks.

During my studies, I have made lifelong friends and found homes in unexpected places. I am endlessly grateful to my Wanderersatz, whose boundless enthusiasm for physics and shared love for chocolate fueled my own passion in physics, and to my Zurich crew, whose late-night gatherings replenished my spirit. My time in Zurich, Heidelberg, and the vibrant energy of India, where physics education intertwined with foreign but familial bonds, have shaped my pathway profoundly. More than anything I am grateful for each and every moment that I was taken away by the spell of studying physics, and the aspects of mathematics and computer science, a field of interest that continues to challenge and inspire me.

Bibliography

[1] B. Abhisheka, S.K. Biswas, and B. et al. Purkayastha. "Recent trend in medical imaging modalities and their applications in disease diagnosis: a review". In: *Multimedia Tools and Applications* 83 (2024), pp. 43035–43070. DOI: 10.1007/s11042-023-17326-1. URL: https://doi.org/10.1007/s11042-023-17326-1.

[2] Y. An et al. "A Novel Region Reconstruction Method for Fluorescence Molecular Tomography". In: *IEEE Transactions on Biomedical Engineering* 62.7 (2015). Epub 2015 Feb 20, pp. 1818–1826. DOI: 10.1109/TBME.2015.2404915.

[3] S. R. Arridge. "Optical tomography in medical imaging". In: *Inverse Problems* 15(2) (1999), R41–R93. DOI: 10.1088/0266-5611/15/2/022.

[4] Baeldung. *Understanding the Learning Curve in Machine Learning*. https://www.baeldung.com/cs/learning-curve-ml. Accessed: 2024-03-18. Mar. 2024.

[5] Pragati Baheti. *The Essential Guide to Neural Network Architectures*. Web page. Accessed on April 18, 2024. 2021. URL: https://www.v7labs.com/blog/neural-network-architectures-guide.

[6] *Camera Noise and Temperature Tutorial*. Camera Noise and Temperature Tutorial. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=10773.

[7] Canon. *Image Sensors Explained*. 2024. URL: https://www.canon.de/pro/infobank/image-sensors-explained/.

[8] Kenny Choo et al. *Machine Learning kompakt: Ein Einstieg für Studierende der Naturwissenschaften*. Springer Nature, 2020. ISBN: 978-3-658-32267-0. DOI: 10.1007/978-3-658-32268-7.

[9] *Cy5 Dye*. Thermo Fisher Scientific. 2024. URL: https://www.thermofisher.com/de/de/home/life-science/cell-analysis/fluorophores/cy5-dye.html.

[10] W. Dubitzky et al., eds. *Encyclopedia of Systems Biology*. Springer Science+Business Media LLC, 2013. DOI: 10.1007/978-1-4419-9863-7.

[11] Dr. Jörg Peter / E0204. "Plenoptische Bildgebung im Kontext synchromodaler Anwendung". powerpoint presentation in the event of the Workshop-Sensors-Fischer. The research on multimodal medical imaging is based at E0204 of the DKFZ, Heidelberg.

[12] Anderson Gomes et al. "Experimental Methods in Chemical Engineering: Fluorescence Emission Spectroscopy". In: *The Canadian Journal of Chemical Engineering* 97 (May 2019). DOI: 10.1002/cjce.23506.

[13] Qingwen Guo et al. "A novel multi-label pest image classifier using the modified Swin Transformer and soft binary cross entropy loss". In: *Engineering Applications of Artificial Intelligence* 126 (2023), p. 107060. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2023.107060.

[14] Kaori Hirayama. *Lösung der Probleme bei der Biofluoreszenz-Bildgebung durch Photonenaufwärtskonversion*. Olympus Life Science. 2023. URL: https://www.olympus-lifescience.com/de/discovery/solving-the-problems-of-biofluorescence-imaging-with-photon-upconversion/.

[15] *ImageJ Home*. 2024. URL: https://imagej.net/ij/index.html.

[16] F. Isensee et al. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation". In: *Nature methods* 18 (2021), pp. 203–211. DOI: 10.1038/s41592-020-01008-z.

[17] Fabian Isensee and et al. "nnU-Net: Breaking the Spell on Successful Medical Image Segmentation". In: (Apr. 2019).

[18] Florian Isensee et al. *Welcome to the new nnU-Net!* https://github.com/MIC-DKFZ/nnUNet.

[19] Ibrahem Kandel and Mauro Castelli. "Transfer Learning with Convolutional Neural Networks for Diabetic Retinopathy Image Classification. A Review". In: *Applied Sciences* 10 (Mar. 2020), p. 2021. DOI: 10.3390/app10062021.

[20] C. C. Leng and J. Tian. "Mathematical method in optical molecular imaging". In: *Sci China Inf Sci* 58 (2015), 031101(13). DOI: 10.1007/s11432-014-5222-5.

[21] X. Li, Y. Li, Y. Zhou, et al. "Real-time denoising enables high-sensitivity fluorescence time-lapse imaging beyond the shot-noise limit". In: *Nature Biotechnology* 41 (2023), pp. 282–292. DOI: 10.1038/s41587-022-01450-8. URL: https://doi.org/10.1038/s41587-022-01450-8.

[22] F. Long. "Deep learning-based mesoscopic fluorescence molecular tomography: an in silico study". In: *J Med Imaging (Bellingham)* 5.3 (2018). Epub 2018 Sep 4, p. 036001. DOI: 10.1117/1.JMI.5.3.036001.

[23] Andreas Maier et al. "A gentle introduction to deep learning in medical image processing". In: *Zeitschrift für Medizinische Physik* 29.2 (2019). Special Issue: Deep Learning in Medical Physics, pp. 86–101. ISSN: 0939-3889. DOI: https://doi.org/10.1016/j.zemedi.2018.12.003. URL: https://www.sciencedirect.com/science/article/pii/S093938891830120X.

[24] Patric Maynard. *Drawing Distinctions: The Varieties of Graphic Expression*. Cornell University Press, 2005, p. 22. ISBN: 0-8014-7280-6.

[25] Shervin Minaee et al. “Image Segmentation Using Deep Learning: A Survey”. In: (Jan. 2020). arXiv: 2001.05566.

[26] MIT BCS Perceptual Science Group. *Demo by John Y. A. Wang*. https://persci.mit.edu/demos/jwang/plenoptic/plenoptic.html. Copyright 1995. All rights reserved. 1995.

[27] *MITK Documentation*. 2024. URL: https://docs.mitk.org/nightly/index.html.

[28] *Noise, Image Quality Factors*. Image Engineering, 2024. URL: https://www.image-engineering.de/library/image-quality/factors/1080-noise.

[29] PerkinElmer, Inc. *Sensitivity, Background, Noise, and Calibration in Atomic Spectroscopy: Effects on Accuracy and Detection Limits*. WHITE PAPER. 2017-2018.

[30] Bryan Poling. “A Tutorial On Camera Models”. In: ().

[31] “Pre-clinical whole-body fluorescence imaging: Review of instruments, methods and applications”. In: *Journal of Photochemistry and Photobiology B: Biology* 98.1 (2010), pp. 77–94. ISSN: 1011-1344. DOI: https://doi.org/10.1016/j.jphotobiol.2009.11.007. URL: https://www.sciencedirect.com/science/article/pii/S1011134409002061.

[32] Princeton Instruments. *Pixel Size and Camera Resolution*. 2024. URL: https://www.princetoninstruments.com/learn/camera-fundamentals/pixel-size-and-camera-resolution.

[33] Princeton Instruments. *Scientific Camera Noise Sources*. 2024. URL: https://www.princetoninstruments.com/learn/camera-fundamentals/scientific-camera-noise-sources.

[34] Pavan M. V. Raja and Andrew R. Barron. “1.11: Fluorescence Spectroscopy”. In: *LibreTexts - Chemistry* (). URL: https://chem.libretexts.org/Bookshelves/Analytical_Chemistry/Physical_Methods_in_Chemistry_and_Nano_Science_(Barron)/01%3A_Elemental_Analysis/1.11%3A_Fluorescence_Spectroscopy.

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[36] R. Schofield et al. “Image reconstruction: Part 1 – understanding filtered back projection, noise and image acquisition”. In: *Journal of Cardiovascular Computed Tomography* 14.3 (2020), pp. 219–225. ISSN: 1934-5925. DOI: 10.1016/j.jcct.2019.04.008. URL: https://www.sciencedirect.com/science/article/pii/S1934592519300607.

[37] Ralf B. Schulz and Wolfhard Semmler. “Grundlagen optischer und fluoreszenzgestützter Tomographie in diffusen Medien”. In: *Z. Med. Phys.* 15 (2005), pp. 177–186. URL: http://www.elsevier.de/zmedphys.

[38] *Spatial Resolution, Pixel Size, and Scale*. last modified: 2015-11-20, last accessed: April, 2024. Natural Resources Canada. URL: https://natural-resources.canada.ca/maps-tools-and-publications/satellite-imagery-and-air-

photos/tutorial-fundamentals-remote-sensing/satellites-and-sensors/spatial-resolution-pixel-size-and-scale/9407.

[39] Chris Stolk. "Radon Transform Lecture Notes". In: (2014). December 18. URL: https://staff.fnwi.uva.nl/c.c.stolk/FourierAnalysis2013/notes_Radon1.pdf.

[40] Thermo Fisher Scientific. *Fluorescence Fundamentals*. URL: https://www.thermofisher.com/de/de/home/life-science/cell-analysis/cell-analysis-learning-center/molecular-probes-school-of-fluorescence/fluorescence-basics/fluorescence-fundamentals.html.

[41] Shinji Wakimoto, Yoshiya Matsukawa, and Hideyuki Aoki. "New criteria to select reasonable hyperparameters for kinetic parameter estimation in distributed activation energy model (DAEM) by using neural network". In: *Chemical Engineering Science* 285 (2024), p. 119597. ISSN: 0009-2509. DOI: 10.1016/j.ces.2023.119597.

[42] Qiang Wang, Yufei Ma, Kun Zhao, et al. "A Comprehensive Survey of Loss Functions in Machine Learning". In: *Annals of Data Science* 9 (2022), pp. 187–212. DOI: 10.1007/s40745-020-00253-5.

[43] *What is Semantic Segmentation?* IBM. 2024. URL: https://www.ibm.com/topics/semantic-segmentation.

[44] Wikipedia contributors. *Radon transform — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Radon_transform&oldid=1210821887. [Online; accessed 25-April-2024]. 2024.

[45] T. Willson. "Principles of CT". In: *Clinical Atlas of Bone SPECT/CT*. Ed. by T. Van den Wyngaert, G. Gnanasegaran, and K. Strobel. Springer, Cham, 2023. DOI: 10.1007/978-3-030-32256-4_4-1. URL: https://doi.org/10.1007/978-3-030-32256-4_4-1.

[46] P.J. Withers, C. Bouman, S. Carmignato, et al. "X-ray computed tomography". In: *Nature Reviews Methods Primers* 1 (2021), p. 18. DOI: 10.1038/s43586-021-00015-4. URL: https://doi.org/10.1038/s43586-021-00015-4.

[47] *X-ray computed tomography (CT scans)*. https://quakerecnankai.blogspot.com/2014/12/x-ray-computed-tomography-ct-scans.html. Accessed on Tuesday 16 December 2014.

[48] H. W. Yeh, O. Karmach, A. Ji, et al. "Red-shifted luciferase–luciferin pairs for enhanced bioluminescence imaging". In: *Nature Methods* 14 (2017), pp. 971–974. DOI: 10.1038/nmeth.4400. URL: https://doi.org/10.1038/nmeth.4400.

[49] Peng Zhang et al. "A review of advances in imaging methodology in fluorescence molecular tomography". In: *Phys. Med. Biol.* 67 (2022), 10TR01. DOI: 10.1088/1361-6560/ac694c.

Appendix E

Code

Contents

E.1 Creating CT-Images as labels based on phantoms

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import cv2
import PIL as pillow, numpy
import os

from skimage.data import shepp_logan_phantom
from skimage.transform import radon, rescale, resize
from skimage.transform import iradon
from skimage.transform.radon_transform import _get_fourier_filter
from skimage import io

def do(path, filename):
    fig, ((ax1, ax2, ax5), (ax3, ax4, ax6)) = plt.subplots(2,3, figsize=(9, 8))
    image = io.imread(path)
    dimensions = image.shape
    # Check if the array is three-dimensional
    if len(dimensions) == 3:
        # Convert the image to grayscale - in case shape is (x, y, 3)
        image = np.dot(image[..., :3], [0.2989, 0.5870, 0.1140])
    image4 = Image.fromarray(image)
    #image4.save('output_addtwo/'+filename[:-4]+'init.png')
    ax1.set_title(filename[:4]+"\ninitial phantom")
    ax1.imshow(image4, cmap=plt.cm.Greys_r)
    print("image shape: ", dimensions)
    print("image type: ", type(image))
    print("image shape after: ", image.shape)
    image1 = rescale(image, scale=0.75, mode='reflect', channel_axis=None)# 0.75 for 160
    theta = np.linspace(0., 360., max(image1.shape), endpoint=False)
    sinogram_init = radon(image1, theta=theta)
    sino= np.transpose(sinogram_init, axes=(1, 0))
    sino1 = resize(sino, (30, 55),anti_aliasing=True)
    sino1 = cv2.normalize(sino1, None, 255, 0, cv2.NORM_MINMAX,
```

```
cv2.CV_8U)
    image1 = Image.fromarray(sino1)
    image1.save('output_addtwo/'+filename[:-4]+'_0.png')
    ax3.set_title("initial sinogram (transposed)\ninput for
reconstruction")
    ax3.imshow(image1, cmap=plt.cm.Greys_r)
    image_orig = image


    # roation 1
    rotated_array = np.rot90(image_orig, k=1)
    image3 = Image.fromarray(rotated_array)
    #image3.save('output_addtwo/'+filename[:-4]+'turned.png')
    ax2.set_title("turned phantom")
    ax2.imshow(image3, cmap=plt.cm.Greys_r)
    image = rescale(rotated_array, scale=0.75, mode='reflect',
channel_axis=None)#oder 0.75 for 160 ; 214 auf 360
    print("image shape RESCALED: ", image.shape)
    theta = np.linspace(0., 360., max(image.shape), endpoint=
False)
    sinogram = radon(image, theta=theta)
    sino= np.transpose(sinogram, axes=(1, 0))
    sino1 = resize(sino, (30, 55),anti_aliasing=True)
    sino1 = cv2.normalize(sino1, None, 255, 0, cv2.NORM_MINMAX,
cv2.CV_8U)
    print("sinogram shape: ", sino1.shape)
    print("sino type: ", type(sino1))
    image1 = Image.fromarray(sino1)
    image1.save('output_addtwo/'+filename[:-4]+'_1.png')
    ax4.set_title("turned sinogram (transposed)\ninput for
reconstruction")
    ax4.imshow(image1, cmap=plt.cm.Greys_r)

    # roation -1
    rotated_array = np.rot90(image_orig, k=-1)
    image3 = Image.fromarray(rotated_array)
    #image3.save('output_addtwo/'+filename[:-4]+'turned.png')
    ax2.set_title("turned phantom")
    ax2.imshow(image3, cmap=plt.cm.Greys_r)
    image = rescale(rotated_array, scale=0.75, mode='reflect',
channel_axis=None)#oder 0.75 for 160 ; 214 auf 360
```

```
69      print("image shape RESCALED: ", image.shape)
70      theta = np.linspace(0., 360., max(image.shape), endpoint=
    False)
71      sinogram = radon(image, theta=theta)
72      sino= np.transpose(sinogram, axes=(1, 0))
73      sino1 = resize(sino, (30, 55),anti_aliasing=True)
74      sino1 = cv2.normalize(sino1, None, 255, 0, cv2.NORM_MINMAX,
    cv2.CV_8U)
75      print("sinogram shape: ", sino1.shape)
76      print("sino type: ", type(sino1))
77      image1 = Image.fromarray(sino1)
78      image1.save('output_addtwo/'+filename[:-4]+'_2.png')
79
80      # roation -2
81      rotated_array = np.rot90(image_orig, k=-2)
82      image3 = Image.fromarray(rotated_array)
83      #image3.save('output_addtwo/'+filename[:-4]+'turned.png')
84      ax2.set_title("turned phantom")
85      ax2.imshow(image3, cmap=plt.cm.Greys_r)
86      image = rescale(rotated_array, scale=0.75, mode='reflect',
    channel_axis=None)#oder 0.75 for 160 ; 214 auf 360
87      print("image shape RESCALED: ", image.shape)
88      theta = np.linspace(0., 360., max(image.shape), endpoint=
    False)
89      sinogram = radon(image, theta=theta)
90      sino= np.transpose(sinogram, axes=(1, 0))
91      sino1 = resize(sino, (30, 55),anti_aliasing=True)
92      sino1 = cv2.normalize(sino1, None, 255, 0, cv2.NORM_MINMAX,
    cv2.CV_8U)
93      print("sinogram shape: ", sino1.shape)
94      print("sino type: ", type(sino1))
95      image1 = Image.fromarray(sino1)
96      image1.save('output_addtwo/'+filename[:-4]+'_3.png')
97      return 0
98
99
100 # Specify the directory path
101 directory_path = "input_addtwo/"
102
103 # Iterate through the files in the directory
104 for filename in os.listdir(directory_path):
```

```
105     if filename.endswith(".png"):
106         # Get the full file path
107         full_path = os.path.join(directory_path, filename)
108         # Call the function with the file path
109         do(full_path, filename)
```

Listing E.1: Creating CT-Images as labels based on phantoms

E.2 Adding WhiteGaussian Noise on Images

```
1  import subprocess
2  import SimpleITK as sitk
3  import numpy as np
4  import cv2
5  from PIL import Image
6  import os
7
8  def execute_command(command):
9      try:
10         # Execute the command
11         result = subprocess.run(command, shell=True, check=True,
       stdout=subprocess.PIPE, stderr=subprocess.PIPE,
       universal_newlines=True)
12         # Print the output
13         print("Command output:")
14         print(result.stdout)
15     except subprocess.CalledProcessError as e:
16         # Print the error message if the command fails
17         print("Command failed with error:", e)
18         print("Error output:")
19         print(e.stderr)
20
21 def load_itk(filename):
22     itkimage = sitk.ReadImage(filename)
23     image = sitk.GetArrayFromImage(itkimage)
24     origin = np.array(list(reversed(itkimage.GetOrigin())))
25     spacing = np.array(list(reversed(itkimage.GetSpacing())))
26     return image, spacing, origin
27
28 def mhd_to_mha(image_path, filename_output_mha):
29     image_arr, spacing, origin  = load_itk(image_path)
30     image = sitk.GetImageFromArray(image_arr)
```

```
    sitk.WriteImage(image,filename_output_mha)
    return 0

def mh_to_png(path, path_saving, filename):
    mha = sitk.ReadImage(path)
    array0 = sitk.GetArrayFromImage(mha)
    #print("sh", array0.shape)
    array = np.reshape(array0, (30, 55))
    image = Image.fromarray(cv2.normalize(array, None, 255, 0,
   cv2.NORM_MINMAX, cv2.CV_8U))
    image.save(path_saving+filename[:-9]+".png")
    return 0

def iterate_folder(filepath_input, path_saving):
    filepath = os.listdir(filepath_input)
    filepath.sort()
    for filename in filepath:
        if filename.endswith("0.mhd"):
            image_path = os.path.join(filepath_input, filename)
            execute_command('mhd-AWGN ' +filename+' 5 5')
            mh_to_png(image_path, path_saving, filename)
    return 0

def iterate_folder1(filepath_input, path_saving):
    filepath = os.listdir(filepath_input)
    filepath.sort()
    for filename in filepath:
        if filename.endswith("AWGN.mhd"):
            image_path = os.path.join(filepath_input, filename)
            mh_to_png(image_path, path_saving, filename)
            path_saving1 = image_path[:-9]+".mha"
            image_path1 = path_saving1[:-14]+'7'+path_saving1
   [-13:]
            mhd_to_mha(image_path, image_path1)
    return 0


test_data_src = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
   Dataset024_FMInoise/imagesTs'
train_data_src = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
   Dataset024_FMInoise/imagesTr'
```

```
test_data_dest = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset024_FMInoise/noise/test/data/'
train_data_dest = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset024_FMInoise/noise/train/data/'

train_data_src = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset037_FMInoise5_self/mhd_data/train/data'
train_data_dest = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset037_FMInoise5_self/mhd_data/train/data/noise/'
test_data_src = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset037_FMInoise5_self/mhd_data/test/data'
test_data_dest = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset037_FMInoise5_self/mhd_data/test/data/noise/'

#first execute this
iterate_folder(test_data_src, test_data_dest)
#then execute this
iterate_folder1(test_data_src, test_data_dest)
```

Listing E.2: Adding WhiteGaussian Noise on Images

E.3 Superposition of phantoms to one phantom with multiply tumors inside

```
import os
import random
from PIL import Image
import numpy as np

# Function to get a list of PNG files in a directory
def get_png_files_in_directory(directory):
    files = []
    filepath = os.listdir(directory)
    filepath.sort()
    for file in filepath:
        if file.endswith(".png"):
            files.append(os.path.join(directory, file))
    return files

# Function to choose two random files from a list
def choose_random_files(files):
    return random.sample(files, 2)
```

```
19
20 # Function to read PNG images and convert them to arrays
21 def png_to_array(image_path):
22     image = Image.open(image_path)
23     return np.array(image)
24
25 # Function to add arrays together
26 def add_arrays(arrays):
27     result = arrays[0]/2
28     for array in arrays[1:]:
29         result = np.add(result, array/2)
30     return result
31
32 # Function to save array as PNG image
33 def array_to_png(array, output_path):
34     image = Image.fromarray(array.astype('uint8'))
35     image.save(output_path)
36
37 # Example directories
38 directory1 = "/home/svea/Documents/nnUNetv2/nnUNet_raw/adding_two
    /imagesTr_Add"
39 directory2 = "/home/svea/Documents/nnUNetv2/nnUNet_raw/adding_two
    /labelsTr_Add"
40 directory3 = "/home/svea/Documents/nnUNetv2/nnUNet_raw/adding_two
    /phantom_Add"
41
42 # Get list of PNG files in each directory
43 files1 = get_png_files_in_directory(directory1)
44 files2 = get_png_files_in_directory(directory2)
45 files3 = get_png_files_in_directory(directory3)
46
47 # Check if both directories have the same number of images
48 if len(files1) != len(files2) != len(file3):
49     raise ValueError("Both directories must have the same number
    of images.")
50
51 # Run the loop 20 times
52 for i in range(20):
53     random_files1 = []
54     random_files2 = []
55     random_files3 = []
```

```
56
57      # Choose two random files from the first directory
58      random_files1 = choose_random_files(files1)
59      print(type(random_files1))
60
61      # Corresponding files from the second directory
62      index = files1.index(random_files1[0])
63      index2 = files1.index(random_files1[1])
64
65      #files1.remove(random_files1[0])
66      random_files2 = [files2[index], files2[index2]]
67      #files2.remove(random_files2[0])
68      random_files3 = [files3[index], files3[index2]]
69      #files3.remove(random_files3[0])
70
71      print(f"Iteration {i+1}:")
72      print("Randomly chosen files from directory 1:")
73      print(random_files1)
74      print("Randomly chosen files from directory 2:")
75      print(random_files2)
76      print("Randomly chosen files from directory 3:")
77      print(random_files3)
78      print()  # Add an empty line for readability
79
80      # Read PNG images as arrays
81      arrays1 = [png_to_array(image_path) for image_path in
    random_files1]
82      arrays2 = [png_to_array(image_path) for image_path in
    random_files2]
83      arrays3 = [png_to_array(image_path) for image_path in
    random_files3]
84
85      # Add arrays together
86      sum_array1 = add_arrays(arrays1)
87      sum_array2 = add_arrays(arrays2)
88      sum_array3 = add_arrays(arrays3)
89
90      directory_aim1 = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset039_add_two/imagesTr'
91      directory_aim2 = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset039_add_two/labelsTr'
```

```
    directory_aim3 = '/home/svea/Documents/nnUNetv2/nnUNet_raw/Dataset039_add_two/phantoms'
    # Save the result as a PNG image
    if i < 10:
        output_path1 = f"{directory_aim1}/Dataset038_00{70+i}_0000.png"
        output_path2 = f"{directory_aim2}/Dataset038_00{70+i}.png"
        output_path3 = f"{directory_aim3}/Dataset038_00{70+i}.png"
    if i < 20:
        output_path1 = f"{directory_aim1}/Dataset038_0{80+(i-10)}_0000.png"
        output_path2 = f"{directory_aim2}/Dataset038_0{80+(i-10)}.png"
        output_path3 = f"{directory_aim3}/Dataset038_0{80+(i-10)}.png"

    array_to_png(sum_array1, output_path1)
    array_to_png(sum_array2, output_path2)
    array_to_png(sum_array3, output_path3)
    print(f"Sum image saved as {output_path1}")
    print()
```

Listing E.3: Superposition of phantoms to one phantom with multiply tumors inside

E.4 Drawing the phantoms for simulation and proper display

```
from PIL import Image, ImageDraw, ImageFont
import numpy as np
import math

big_circle_center = (107, 107)
big_circle_radius = 105
#small_circle_center = (110, 110)
small_circle_radius = 20
y = 107
big_circle_color = (100, 100, 100)  # gray
small_circle_color = (200, 200, 200)  # lighter gray


```

```
14 def draw_all_in_one(big_circle_center,big_circle_radius,
   small_circle_radius, big_circle_color, small_circle_color,
   title, y):
15     image_size = (214, 214)
16     image = Image.new("RGB", image_size, "black")
17     draw = ImageDraw.Draw(image)
18 
19     draw.ellipse((big_circle_center[0] - big_circle_radius,
20                   big_circle_center[1] - big_circle_radius,
21                   big_circle_center[0] + big_circle_radius,
22                   big_circle_center[1] + big_circle_radius),
23                   fill=big_circle_color)
24 
25     x = 2+small_circle_radius+4
26     draw.ellipse((x - small_circle_radius, y -
   small_circle_radius, x + small_circle_radius, y +
   small_circle_radius), fill=small_circle_color)
27     draw.ellipse((y - small_circle_radius, x -
   small_circle_radius, y + small_circle_radius, x +
   small_circle_radius), fill=small_circle_color)
28 
29     x = (2+small_circle_radius+4)+small_circle_radius*2
30     draw.ellipse((x - small_circle_radius, y -
   small_circle_radius, x + small_circle_radius, y +
   small_circle_radius), fill=small_circle_color)
31     draw.ellipse((y - small_circle_radius, x -
   small_circle_radius, y + small_circle_radius, x +
   small_circle_radius), fill=small_circle_color)
32     y1 = y+small_circle_radius*2
33     draw.ellipse((x - small_circle_radius, y1 -
   small_circle_radius, x + small_circle_radius, y1 +
   small_circle_radius), fill=small_circle_color)
34     y1 = y-small_circle_radius*2
35     draw.ellipse((x - small_circle_radius, y1 -
   small_circle_radius, x + small_circle_radius, y1 +
   small_circle_radius), fill=small_circle_color)
36 
37     x = big_circle_center[0]
38     draw.ellipse((x - small_circle_radius, y -
   small_circle_radius, x + small_circle_radius, y +
   small_circle_radius), fill=small_circle_color)
```

```
39      draw.ellipse((y - small_circle_radius, x -
        small_circle_radius, y + small_circle_radius, x +
        small_circle_radius), fill=small_circle_color)
40
41      x = (212-small_circle_radius-4)-small_circle_radius*2
42      draw.ellipse((x - small_circle_radius, y -
        small_circle_radius, x + small_circle_radius, y +
        small_circle_radius), fill=small_circle_color)
43      draw.ellipse((y - small_circle_radius, x -
        small_circle_radius, y + small_circle_radius, x +
        small_circle_radius), fill=small_circle_color)
44      y1 = y+small_circle_radius*2
45      draw.ellipse((x - small_circle_radius, y1 -
        small_circle_radius, x + small_circle_radius, y1 +
        small_circle_radius), fill=small_circle_color)
46      y1 = y-small_circle_radius*2
47      draw.ellipse((x - small_circle_radius, y1 -
        small_circle_radius, x + small_circle_radius, y1 +
        small_circle_radius), fill=small_circle_color)
48
49      x = 212-small_circle_radius-4
50      draw.ellipse((x - small_circle_radius, y -
        small_circle_radius, x + small_circle_radius, y +
        small_circle_radius), fill=small_circle_color)
51      draw.ellipse((y - small_circle_radius, x -
        small_circle_radius, y + small_circle_radius, x +
        small_circle_radius), fill=small_circle_color)
52      image = image.convert("L")
53      # Save the image
54      image.save(f"draw/{title}.png")
55      image.save(f"draw/{title}.jpg")
56      # Show the image (optional)
57      image.close()
58
59      return 0
60
61  def draw(x,y,big_circle_center,big_circle_radius,
        small_circle_radius, big_circle_color, small_circle_color,
        title):
62      image_size = (214, 214)
63      image = Image.new("RGB", image_size, "black")
```

```
64      draw = ImageDraw.Draw(image)
65
66      draw.ellipse((big_circle_center[0] - big_circle_radius,
67                    big_circle_center[1] - big_circle_radius,
68                    big_circle_center[0] + big_circle_radius,
69                    big_circle_center[1] + big_circle_radius),
70                    fill=big_circle_color)
71
72      draw.ellipse((x - small_circle_radius, y -
    small_circle_radius, x + small_circle_radius, y +
    small_circle_radius), fill=small_circle_color)
73      #draw.ellipse((x , y , x + 2*small_circle_radius, y + 2*
    small_circle_radius), fill=small_circle_color)
74
75      # Save the image
76      image = image.convert("L")
77      image.save(f"draw/{title}.png")
78      image.save(f"draw/{title}.jpg")
79      image.close()
80
81      return 0
82
83
84  x = 2+small_circle_radius+4
85  draw(x,y,big_circle_center,big_circle_radius,small_circle_radius,
        big_circle_color, small_circle_color,f'x-{x}_y-{y}') # f'x-{y}
    _y-{x}'
86  draw(y,x,big_circle_center,big_circle_radius,small_circle_radius,
        big_circle_color, small_circle_color,f'x-{y}_y-{x}')
87
88  x = (2+small_circle_radius+4)+small_circle_radius*2
89  draw(y,x,big_circle_center,big_circle_radius,small_circle_radius,
        big_circle_color, small_circle_color,f'x-{y}_y-{x}')
90  draw(x,y,big_circle_center,big_circle_radius,small_circle_radius,
        big_circle_color, small_circle_color,f'x-{x}_y-{y}')
91  y1 = y+small_circle_radius*2
92  draw(x,y1,big_circle_center,big_circle_radius,small_circle_radius
    , big_circle_color, small_circle_color,f'x-{x}_y-{y1}')
93  y1 = y-small_circle_radius*2
94  draw(x,y1,big_circle_center,big_circle_radius,small_circle_radius
    , big_circle_color, small_circle_color,f'x-{x}_y-{y1}')
```

```
95
96 x = big_circle_center[0]
97 draw(y,x,big_circle_center,big_circle_radius,small_circle_radius,
       big_circle_color, small_circle_color,f'x-{y}_y-{x}')
98 draw(x,y,big_circle_center,big_circle_radius,small_circle_radius,
       big_circle_color, small_circle_color,f'x-{x}_y-{y}')
99
100 x = (212-small_circle_radius-4)-small_circle_radius*2
101 draw(x,y,big_circle_center,big_circle_radius,small_circle_radius,
       big_circle_color, small_circle_color,f'x-{x}_y-{y}')
102 draw(y,x,big_circle_center,big_circle_radius,small_circle_radius,
       big_circle_color, small_circle_color,f'x-{y}_y-{x}')
103 y1 = y+small_circle_radius*2
104 draw(x,y1,big_circle_center,big_circle_radius,small_circle_radius
       , big_circle_color, small_circle_color,f'x-{x}_y-{y1}')
105 y1 = y-small_circle_radius*2
106 draw(x,y1,big_circle_center,big_circle_radius,small_circle_radius
       , big_circle_color, small_circle_color,f'x-{x}_y-{y1}')
107
108 x = 212-small_circle_radius-4
109 draw(y,x,big_circle_center,big_circle_radius,small_circle_radius,
       big_circle_color, small_circle_color,f'x-{y}_y-{x}')
110 draw(x,y,big_circle_center,big_circle_radius,small_circle_radius,
       big_circle_color, small_circle_color,f'x-{x}_y-{y}')
111
112 draw_all_in_one(big_circle_center,big_circle_radius,
       small_circle_radius, big_circle_color, small_circle_color,"
       all_in_one", y)
113
114 print("Done")
115
116 for x in range(0, big_circle_center[0]*2, small_circle_radius):
117     for y in range(0, big_circle_center[1]*2, small_circle_radius
       ):
118         #Calculate the distance from the center of the smaller
       circle to the current point
119         distance = math.sqrt(x * x + y * y)
120         if distance <= small_circle_radius*2:
121         draw(x,y,big_circle_center,big_circle_radius,
       small_circle_radius, f'x-{x}_y-{y}')
```

Listing E.4: Drawing the phantoms for simulation and proper display

E.5 Reconstruction of phantom based on output sinogramm of nnUNet

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import cv2
import PIL as pillow, numpy
import os
import numpy as np
from skimage.data import shepp_logan_phantom
from skimage.transform import radon, rescale, resize
from skimage.transform import iradon
from skimage.transform.radon_transform import _get_fourier_filter
from skimage import io

def doCT(path, filename, dir_path, mode,mode1,vgl):
    image = io.imread(path)
    dimensions = image.shape
    if len(dimensions) == 3:
        image = np.dot(image[..., :3], [0.2989, 0.5870, 0.1140])
    print("image shape: ", dimensions)
    print("image shape after: ", image.shape)
    sinogram = image
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(8,
    4.5))
    ax1.set_title("input sinogram")
    ax1.imshow(sinogram, cmap=plt.cm.Greys_r)
    if mode == "transpose":
        image = np.transpose(image, axes=(1, 0))
        print("image shape TRANSPOSED: ")
    vgl = io.imread(vgl)
    #__________Reconstruction - Filtered back
   projection__________
    print("sinogram: ", type(sinogram))
    print("im: ", type(image))
    zero = np.zeros(image.shape[1])
    if mode1 == "not_ct":
        print("image vstsackdone")
        image = np.vstack((zero, image, zero))
        image = np.vstack((zero, image, zero))
```

```
37          image = np.vstack((zero, image, zero))
38          image = np.vstack((zero, image, zero))
39          image = np.vstack((zero, image, zero))
40          image = np.vstack((zero, image, zero))
41          image = resize(image, (55, 30),anti_aliasing=True)
42
43      print("image shape RESCALED: ", image.shape)
44      theta = np.linspace(0., 360., image.shape[1], endpoint=False)
45      print(image.dtype)
46      image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)
47      print("theta: ", theta.shape)
48      ax2.set_title("sinogram (transposed\ninput for reconstruction
    ")
49      ax2.imshow(image, cmap=plt.cm.Greys_r)
50      ax4.set_title(f"original phantom {vgl.shape}")
51      ax4.imshow(vgl, cmap=plt.cm.Greys_r)
52      fig.tight_layout()
53      # filters = ['ramp', 'shepp-logan', 'cosine', 'hamming', '
    hann']
54      reconstruction_fbp = iradon(image, theta=theta, filter_name='
    ramp')
55      ax3.set_title("Reconstruction\nFiltered back projection")
56      reconstruction_fbp = cv2.normalize(reconstruction_fbp, None,
    0, 255, cv2.NORM_MINMAX)
57      ax3.imshow(reconstruction_fbp, cmap=plt.cm.Greys_r)
58      fig.tight_layout()
59      rec55 = reconstruction_fbp
60      figrec, (ax1r) = plt.subplots(1,1, figsize=(8, 4.5))
61      ax1r.imshow(rec55, cmap=plt.cm.Greys_r)
62      figrec.tight_layout()
63      plt.savefig(dir_path+filename[:-4]+'recon.png')
64      #plt.show()
65      plt.close()
66      return 0
67
68
69  def doBLI(path, filename, dir_path, mode,mode1,vgl):
70      image = io.imread(path)
71      dimensions = image.shape
72      if len(dimensions) == 3:
73          image = np.dot(image[..., :3], [0.2989, 0.5870, 0.1140])
```

```
74      print("image shape: ", dimensions)
75      print("image shape after: ", image.shape)
76      sinogram = image
77      fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(8,
        4.5))
78      ax1.set_title("input sinogram")
79      ax1.imshow(sinogram, cmap=plt.cm.Greys_r)
80      if mode == "transpose":
81          image = np.transpose(image, axes=(1, 0))
82          print("image shape TRANSPOSED: ")
83      vgl = io.imread(vgl)
84      #__________Reconstruction - Filtered back
    projection__________
85      print("sinogram: ", type(sinogram))
86      print("im: ", type(image))
87      zero = np.zeros(image.shape[1])
88      if mode1 == "not_ct":
89          print("image vstsackdone")
90          image = np.vstack((zero, image, zero))
91          image = np.vstack((zero, image, zero))
92          image = np.vstack((zero, image, zero))
93          image = np.vstack((zero, image, zero))
94          image = np.vstack((zero, image, zero))
95          image = np.vstack((zero, image, zero))
96          image = resize(image, (55, 30),anti_aliasing=True)
97      print("image shape RESCALED: ", image.shape)
98      theta = np.linspace(0., 360., image.shape[1], endpoint=False)
99      print(image.dtype)
100     image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)
101     print("theta: ", theta.shape)
102     ax2.set_title("sinogram (transposed\ninput for reconstruction
    ")
103     ax2.imshow(image, cmap=plt.cm.Greys_r)
104     ax4.set_title(f"original phantom {vgl.shape}")
105     ax4.imshow(vgl, cmap=plt.cm.Greys_r)
106     fig.tight_layout()
107     reconstruction_fbp = iradon(image, theta=theta, filter_name='
    ramp')
108     ax3.set_title("Reconstruction\nFiltered back projection")
109     reconstruction_fbp = cv2.normalize(reconstruction_fbp, None,
    0, 255, cv2.NORM_MINMAX)
```

```
110     ax3.imshow(reconstruction_fbp, cmap=plt.cm.Greys_r)
111     fig.tight_layout()
112     rec55 = reconstruction_fbp
113 
114     reconstruction_fbp = iradon(image, theta=theta, filter_name='
    hann', interpolation='linear', preserve_range=True)
115     reconstruction_fbp = cv2.normalize(reconstruction_fbp, None,
    0, 255, cv2.NORM_MINMAX)
116     print('pre',reconstruction_fbp[0,0])
117     boarder = reconstruction_fbp[0,0]
118 
119     reconstruction_fbp[reconstruction_fbp > 100] = 250
120     #reconstruction_fbp[reconstruction_fbp > 150] = 250 #for DS37
121     for i in range(0, reconstruction_fbp.shape[0]):
122         k = 0
123         for j in range(0, reconstruction_fbp.shape[1]):
124             if k == 0:
125                 if not reconstruction_fbp[i, j] == boarder:
126                     reconstruction_fbp[i, :j] = 0
127                     k = 1
128     reconstruction_fbp = np.flip(reconstruction_fbp, axis=1)
129     for i in range(0, reconstruction_fbp.shape[0]):
130         k = 0
131         for j in range(0, reconstruction_fbp.shape[1]):
132             if k == 0:
133                 if not reconstruction_fbp[i, j] == boarder:
134                     reconstruction_fbp[i, :j] = 0
135                     k = 1
136     reconstruction_fbp = np.flip(reconstruction_fbp, axis=1)
137     print(reconstruction_fbp[0])
138     #reconstruction_fbp[reconstruction_fbp > 190] = 250
139     reconstruction_fbp[ (reconstruction_fbp > 0) & (
    reconstruction_fbp <= 100) ] = 100
140     #reconstruction_fbp[ (reconstruction_fbp > 0) & (
    reconstruction_fbp <= 150) ] = 100 #for DS37
141     ax3.set_title("Reconstruction\nFiltered back projection")
142     ax3.imshow(reconstruction_fbp, cmap=plt.cm.Greys_r)
143     fig.tight_layout()
144     #plt.savefig(dir_path+filename[:-4]+'ramp_recon_filter.png')
145 
146     fig2rec, (ax2r) = plt.subplots(1,1, figsize=(8, 4.5))
```

```
147     #reconstruction_fbp = np.rot90(reconstruction_fbp, k=2)
148     ax2r.imshow(reconstruction_fbp, cmap=plt.cm.Greys_r)
149     fig2rec.tight_layout()
150     plt.savefig(dir_path+filename[:-4]+'filterrecon.png')
151     figrec, (ax1r) = plt.subplots(1,1, figsize=(8, 4.5))
152     ax1r.imshow(rec55, cmap=plt.cm.Greys_r)
153     figrec.tight_layout()
154     plt.savefig(dir_path+filename[:-4]+'recon.png')
155     #plt.show()
156     plt.close()
157     return 0
158 
159 # Specify the directory path
160 directory_path = "singleall/BLI/37/"
161 mode = "transpose"#when input is straight a sinogram
162 mode1 = "nnot_ct"#when data is NOT ct data (not 026, 023)
163 vgl = 'practice/Dataset032_045.png'
164 for filename in os.listdir(directory_path):
165     if filename.endswith(".png") and not filename.endswith("recon
    .png"):
166         full_path = os.path.join(directory_path, filename)
167         doBLI(full_path, filename, directory_path, mode , mode1,
    vgl) #for DS37
168 
169 directory_path = "doubleall/BLI/"
170 vgl = 'practice/Dataset032_045.png'
171 for filename in os.listdir(directory_path):
172     if filename.endswith(".png") and not filename.endswith("recon
    .png"):
173         full_path = os.path.join(directory_path, filename)
174         doBLI(full_path, filename, directory_path, mode , mode1,
    vgl)
175 
176 directory_path = "singleall/CT/"
177 vgl = 'practice/Dataset032_045.png'
178 for filename in os.listdir(directory_path):
179     if filename.endswith(".png") and not filename.endswith("recon
    .png"):
180         full_path = os.path.join(directory_path, filename)
181         doCT(full_path, filename, directory_path, mode , mode1,
    vgl)
```

```
182
183 directory_path = "doubleall/CT/"
184 vgl = 'practice/Dataset032_045.png'
185 for filename in os.listdir(directory_path):
186     if filename.endswith(".png") and not filename.endswith("recon
    .png"):
187         full_path = os.path.join(directory_path, filename)
188         doCT(full_path, filename, directory_path, mode , mode1,
    vgl)
189
190 directory_path = "practice/ctvgl/"
191 vgl = 'practice/Dataset032_045.png'
192 for filename in os.listdir(directory_path):
193     if filename.endswith(".png") and not filename.endswith("recon
    .png"):
194         full_path = os.path.join(directory_path, filename)
195         doBLI(full_path, filename, directory_path, mode , mode1,
    vgl)
```

Listing E.5: Reconstruction of phantom based on output sinogramm of nnUNet

E.6 Applying simple filter for reducing artefact in CT-Output

```
1 from PIL import Image, ImageChops
2 import numpy as np
3 from PIL import Image
4 import os
5
6
7 def artefacts(image1, label):
8     image1 = Image.open(image1)
9     image1 = image1.convert("RGB")
10    array1 = np.array(image1)
11    positions = np.where(np.all(array1 == [0, 0, 0], axis=-1))
12    for pos in zip(*positions):
13        #print(f"Pixel with value [0, 0, 0] found at position: {
    pos[0]}, {pos[1]}")
14        if pos[1] != 0 and pos[1] != array1.shape[0] - 1:
15            #array1[pos[0], pos[1]] = [230, 230, 230]
16            print("here", pos)
17
```

```
18      mask = np.all(array1 == [0, 0, 0], axis=-1)
19      mask[:,0:10] = False  # Exclude first row
20      mask[:,-10:-1] = False  # Exclude last row
21      mask[:,-1] = False
22      array1[mask] = [240, 240, 240]
23      mask = np.all(array1 == [0, 0, 0], axis=-1)
24      mask[:,0] = False  # Exclude first row
25      mask[:,10:-10] = False  # Exclude last row
26      mask[:,-1] = False  # Exclude last row
27      array1[mask] = [120,120,120]
28      #print("ar2",array1[-16])
29      result_image = Image.fromarray(array1)
30      result_image.save(label)
31
32  directory_path1 = "doubleall/CT/"
33  images_folder1 = [f for f in os.listdir(directory_path1) if f.
        endswith('.png')]
34  #images_folder1 = [f for f in os.listdir(directory_path1) if f.
        endswith('.png') and not f.endswith("subtract.png")]
35  images_folder1.sort()
36  print(images_folder1)
37  for filename1 in images_folder1:
38      #label = directory_path1+filename1[:-4]+"subtract.png"
39      label = "artefacts/"+filename1
40      image1 = os.path.join(directory_path1, filename1)
41      artefacts(image1, label)
42      print(filename1)
43
44  directory_path1 = "singleall/CT/"
45  images_folder1 = [f for f in os.listdir(directory_path1) if f.
        endswith('.png')]
46  #images_folder1 = [f for f in os.listdir(directory_path1) if f.
        endswith('.png') and not f.endswith("subtract.png")]
47  images_folder1.sort()
48  print(images_folder1)
49  for filename1 in images_folder1:
50      #label = directory_path1+filename1[:-4]+"subtract.png"
51      label = "artefacts/"+filename1
52      image1 = os.path.join(directory_path1, filename1)
53      artefacts(image1, label)
```

```
54     print(filename1)
```

Listing E.6: Applying simple filter for reducing artefact in CT-Output

E.7 Creating difference image for analysing the results

```
1  from PIL import Image, ImageChops
2  import numpy as np
3  from PIL import Image
4  import os
5
6  def redhot(image, label):
7      image = Image.open(image)
8      image = image.convert("RGB")
9      array = np.array(image)
10     print(array[25])
11     #array[np.all(array == [255,255,255], axis=2)] = [183, 18,
       31]
12     array[np.all(array == [255,255,255], axis=2)] = [247, 98, 98]
13     result_image = Image.fromarray(array)
14     result_image.save(label)
15
16 def subtract(image1, image2, label):
17     image1 = Image.open(image1)
18     image2 = Image.open(image2)
19     image1 = image1.convert("RGB")
20     image2 = image2.convert("RGB")
21     array1 = np.array(image1)
22     array2 = np.array(image2)
23     #print("ar1",array1[-1])
24     #print("ar2",array2[-1])
25     if array1.shape != array2.shape:
26         raise ValueError("Images must have the same shape")
27     # Subtract array2 from array1
28     diff_array = np.abs(array2 - array1)
29     #print("shapediff",diff_array.shape)
30     #print("diff",diff_array[-15])
31     # Create a new image array for visualization
32     mask = np.all(diff_array == [0, 0, 0], axis=-1)
33     diff_array[mask] = [255, 255, 255]
34     mask1 = np.all(diff_array == [1, 1, 1], axis=-1)
```

```
    diff_array[mask1] = [34,139,34]
    mask4 = np.all(diff_array == [128, 128, 128], axis=-1)
    diff_array[mask4] = [34,139,34]
    mask2 = np.all(diff_array == [127, 127, 127], axis=-1)
    diff_array[mask2] = [255, 51, 51]
    mask3 = np.all(diff_array == [129,129,129], axis=-1)
    diff_array[mask3] = [0, 128, 255]
    print("diff5",diff_array[14])
    result_image = Image.fromarray(diff_array)
    result_image.save(label)

directory_path1 = "doubleallsubtract/38/"
#directory_path1 = "doubleallsubtract/41/"
directory_path2 = "labelsubtract/"
images_folder1 = [f for f in os.listdir(directory_path1) if f.
   endswith('.png') and not f.endswith("subtract.png")]
images_folder2 = [f for f in os.listdir(directory_path2) if f.
   endswith('.png')]
images_folder1.sort()
images_folder2.sort()
for filename1,filename2 in zip(images_folder1,images_folder2):
    #label = directory_path1+filename1[:-4]+"subtract.png"
    label = "doubleallsubtract/subtract/"+filename1[:-4]+"
   subtract.png"
    image1 = os.path.join(directory_path1, filename1)
    image2 = os.path.join(directory_path2, filename2)
    subtract(image1, image2, label)
    print(filename1,filename2)
```

Listing E.7: Creating difference image for analysing the results

E.8 LiprosCode for FMI-Simulation by Dr. Jörg Peter

```
#!/bin/bash

for ((x = -4 ; x <= 4 ; x++)); do
  for ((y = -4 ; y <= 4 ; y++)); do
    radius="$(bc -l <<<"sqrt($x*$x+$y*$y)")"
    if (( $(echo "$radius <= 4.0" |bc -l) )); then

      xx=$(printf "%+03d" $x)
```

```
9       yy=$(printf "%+03d" $y)
10
11      workingDir="${HOME}/cpp/lipros/output/lipros-30-sphere_${xx
    }mm-${yy}mm"
12      mkdir "$workingDir"
13      cp "$0" "$workingDir"
14      cd "$workingDir" || { echo "Could not change into
    $workingDir!"; exit 1; }
15      echo "$workingDir"
16
17      liprosArgs=(
18        lipros
19          --cpuThreads="$(cat /proc/cpuinfo | awk '/^processor/{
    print $3}' | wc -l)"
20          --verbosity=0
21        gantry
22          --gantryNumberOfCameras=30
23          --gantryCameraRadiusOfCurvature=32.0mm
24          --gantryNumberOfLights=30 # should be 0 (BLI) or equal
    to gantryNumberOfCameras
25          --gantryLightRadiusOfCurvature=32.0mm
26        cameras
27            --cameraSensorMinSize=0.1mm 5.5mm
28            --cameraSensorPixelSize=0.1mm 0.1mm
29            --cameraSensorSNR=5dB
30            --cameraSensorNoiseGain=0
31          cameraOrthographic
32        lights
33          excitationConfocal
34            --excitationConfocalWavelengthCenter=660nm
35            --excitationConfocalWavelengthFWHM=20.0nm
36            --excitationConfocalBeamDiameter=0.5mm
37        phantom
38          --phantomWriteAsMhd=1
39          --phantomExportVoxelSize=0.1mm 0.1mm 0.1mm
40          tissue
41            --tisGeometry=CYLINDER
42            --tisScale=10.5 10.5 20.0
43            --tisType=MUSCLE
44          tissue
45            --tisGeometry=SPHERE
```

```
            --tisScale=2.0 2.0 2.0
            --tisTranslate=${x}mm ${y}mm 0.0mm
            --tisType=TUMOR
            --tisFluorophore=CY55
            --tisFluorophoreMolarConcentration=1.0
         simulate
          --simulateFluenceVoxelSize=0.1mm 0.1mm 0.1mm
          --simulateTotalAbsorberZmin=-2.0mm
          --simulateTotalAbsorberZmax=2.0mm
          --simulateCameraAxialCenterPositionZ=0.0mm
           simulateFmi
            --simulateFluenceTimeFrames=1
            --simulateFluenceFrameDuration=1s
            --simulateLightAxialCenterPositionZ=0.0mm
            --simulateLightStartAngleOffset=0.0deg
            --simulateLightStepAngle=5.0deg
            --simulateLightAngleSteps=1
            --simulateLightExcitationPhotonsPerPos=1000000
            --simulateLightExcitationPulseDuration=1s
        )

      lipros "${liprosArgs[@]}"

    fi
  done
done
```

Listing E.8: LiprosCode for FMI-Simulation by Dr. Jörg Peter

E.9 LiprosCode for BLI/ISP-Simulation by Dr. Jörg Peter

```
#!/bin/bash

for ((x = -4 ; x <= 4 ; x++)); do
  for ((y = -4 ; y <= 4 ; y++)); do
    radius="$(bc -l <<<"sqrt($x*$x+$y*$y)")"
    if (( $(echo "$radius <= 4.0" |bc -l) )); then

      xx=$(printf "%+03d" $x)
      yy=$(printf "%+03d" $y)

```

```
11      workingDir="${HOME}/cpp/lipros/output/lipros-30-sphere-BLI-
    no-phantom_${xx}mm-${yy}mm"
12      mkdir "$workingDir"
13      cp "$0" "$workingDir"
14      cd "$workingDir" || { echo "Could not change into
    $workingDir!"; exit 1; }
15      echo "$workingDir"
16
17      liprosArgs=(
18        lipros
19          --cpuThreads="$(cat /proc/cpuinfo | awk '/^processor/{
    print $3}' | wc -l)"
20          --verbosity=0
21        gantry
22          --gantryNumberOfCameras=30
23          --gantryCameraRadiusOfCurvature=32.0mm
24        cameras
25            --cameraSensorMinSize=0.1mm 5.5mm
26            --cameraSensorPixelSize=0.1mm 0.1mm
27            --cameraSensorSNR=5dB
28            --cameraSensorNoiseGain=0
29          cameraOrthographic
30        phantom
31          --phantomWriteAsMhd=1
32          --phantomExportVoxelSize=0.1mm 0.1mm 0.1mm
33 #          tissue
34 #            --tisGeometry=CYLINDER
35 #            --tisScale=10.5 10.5 20.0
36 #            --tisType=MUSCLE
37          tissue
38            --tisGeometry=SPHERE
39            --tisScale=2.0 2.0 2.0
40            --tisTranslate=${x}mm ${y}mm 0.0mm
41            --tisType=TUMOR
42            --tisBioluminescence=RED_FLUC
43            --tisBioluminescencePhotons=1000000
44         simulate
45          --simulateFluenceVoxelSize=0.1mm 0.1mm 0.1mm
46          --simulateTotalAbsorberZmin=-2.0mm
47          --simulateTotalAbsorberZmax=2.0mm
48          --simulateCameraAxialCenterPositionZ=0.0mm
```

```
            simulateBli
        )

      lipros "${liprosArgs[@]}"

    fi
  done
done
```

Listing E.9: LiprosCode for BLI/ISP-Simulation by Dr. Jörg Peter

E.10 Preperation of data for nnUNet for Datasets 20 to 29

```
from matplotlib import pyplot as plt
import matplotlib as mpl
import matplotlib.pylab as plt
import SimpleITK as sitk
import numpy as np
import cv2
from PIL import Image
import os
import json
import shutil

def split_data_in_directory(src_dir_home, dest_dir_data,
    dest_dir_label):
    if not os.path.exists(src_dir_home):
        print("Source directory does not exist.")
        return
    if os.path.exists(dest_dir_data):
        print("Dest directory does exist.")
        return
    if os.path.exists(dest_dir_label):
        print("Dest directory does exist.")
        return
    os.makedirs(dest_dir_data, exist_ok=True)
    os.makedirs(dest_dir_label, exist_ok=True)

    dirs = [d for d in os.listdir(src_dir_home) if os.path.isdir(
    os.path.join(src_dir_home, d))]
    dirs.sort()
```

```
27      for i, directory in enumerate(dirs):
28          if directory.startswith("lipros-30-sphere_"):
29              src_directory = os.path.join(src_dir_home, directory)
30              dest_directory = os.path.join(dest_dir_data,
    directory)
31              shutil.copytree(src_directory, dest_directory)
32          else:
33              src_directory = os.path.join(src_dir_home, directory)
34              dest_directory = os.path.join(dest_dir_label,
    directory)
35              shutil.copytree(src_directory, dest_directory)
36      print(f"Copied {src_directory} to {dest_directory}")
37      print("split_data_in_directory - done")
38
39  def copy_every_third_directory(src_dir_data, src_dir_label,
    dest_dir_train, dest_dir_train_label, dest_dir_test,
    dest_dir_test_label):
40      # Check if the source directory exists
41      if not os.path.exists(src_dir_data):
42          print("Source directory data does not exist.")
43          return
44      if not os.path.exists(src_dir_label):
45          print("Source directory label does not exist.")
46          return
47      if os.path.exists(dest_dir_train):
48          print("Dest directory does exist.")
49          return
50
51      os.makedirs(dest_dir_train, exist_ok=True)
52      os.makedirs(dest_dir_train_label, exist_ok=True)
53      os.makedirs(dest_dir_test, exist_ok=True)
54      os.makedirs(dest_dir_test_label, exist_ok=True)
55
56      dirs = [d for d in os.listdir(src_dir_data) if os.path.isdir(
    os.path.join(src_dir_data, d))]
57      dirs.sort()
58      for i, directory in enumerate(dirs):
59              #if (i + 1) % 3 == 0:
60              #    src_directory = os.path.join(src_dir_data,
    directory)
61              #    dest_directory = os.path.join(dest_dir_test,
```

```
    directory)
62              #     shutil.copytree(src_directory, dest_directory)
63              #else:
64          src_directory = os.path.join(src_dir_data, directory)
65          dest_directory = os.path.join(dest_dir_train, directory)
66          shutil.copytree(src_directory, dest_directory)
67
68      dirs = [d for d in os.listdir(src_dir_label) if os.path.isdir
    (os.path.join(src_dir_label, d))]
69      dirs.sort()
70      for i, directory in enumerate(dirs):
71              #if (i + 1) % 3 == 0:
72              #     src_directory = os.path.join(src_dir_label,
    directory)
73              #     dest_directory = os.path.join(
    dest_dir_test_label, directory)
74              #     shutil.copytree(src_directory, dest_directory)
75              #else:
76          src_directory = os.path.join(src_dir_label, directory)
77          dest_directory = os.path.join(dest_dir_train_label,
    directory)
78          shutil.copytree(src_directory, dest_directory)
79
80      print(f"Copied {src_directory} to {dest_directory}")
81      print("copy_every_third_directory - done")
82
83  def add_laser_layer(image_arr, filename_output_label_split,
    filename):
84      arrays = []
85      label = filename_output_label_split+filename
86      for i in range(0,30):
87          arrays.append(image_arr)
88      arrays_combined = np.array(arrays)
89      image_combined = sitk.GetImageFromArray(arrays_combined)
90      sitk.WriteImage(image_combined,label)
91      image_arr, spacing, origin  = load_itk(label)
92      image_arr = np.transpose(image_arr, axes=(0, 2, 1))
93      return image_arr
94
95  def add_laser_layer_bin(image_arr, filename_output_label_split,
    filename):
```

```
96      arrays = []
97      label_split = filename_output_label_split+filename
98      for i in range(0,30):
99          arrays.append(image_arr)
100     arrays_combined = np.array(arrays)
101     image_combined = sitk.GetImageFromArray(arrays_combined)
102     sitk.WriteImage(image_combined,label_split)
103     image_arr, spacing, origin  = load_itk(label_split)
104     image_arr = np.transpose(image_arr, axes=(1, 0, 2))
105     return image_arr
106
107 def load_itk(filename):
108     itkimage = sitk.ReadImage(filename)
109     image = sitk.GetArrayFromImage(itkimage)
110     origin = np.array(list(reversed(itkimage.GetOrigin())))
111     spacing = np.array(list(reversed(itkimage.GetSpacing())))
112     return image, spacing, origin
113
114 def mh_to_png(filename, path):
115     mha = sitk.ReadImage(filename+".mha")
116     array0 = sitk.GetArrayFromImage(mha)
117     array = np.reshape(array0, (30, 55))
118     image = Image.fromarray((array * 255).astype(np.uint8))
119     image.save(path+filename+".png")
120     #image.save(filename+".jpg")
121     return 0
122
123 def mhd_to_mha(image_path, filename_output_mha,
    filename_output_label_split, filename):
124     image_arr, spacing, origin  = load_itk(image_path)
125     #image_arr = np.transpose(image_arr, (1, 2, 0))
126     #image_arr_added = add_laser_layer_bin(image_arr,
    filename_output_label_split, filename)
127     image = sitk.GetImageFromArray(image_arr)
128     sitk.WriteImage(image,filename_output_mha)
129     return image
130
131 def load_ALL_label(filepath_input, filename_output_mha,
    filename_output_mhd, type_mod, filename_output_label_split,
    filename_output_mha_nonbin, filename_output_mha_bin,
    filename_output_mha_self):
```

```
132     print("saved as: ", filename_output_mha)
133     print("load_ALL open")
134     arrays = []
135     filepath = os.listdir(filepath_input)
136     filepath.sort()
137     for filename in filepath:
138         if filename.endswith("oCams.mhd"):
139             image_path = os.path.join(filepath_input, filename)
140             image = mhd_to_mha(image_path,
    filename_output_mha_nonbin, filename_output_label_split,
    filename)
141         if filename.endswith("binarized-10.mhd"):
142             image_path = os.path.join(filepath_input, filename)
143             image = mhd_to_mha(image_path,
    filename_output_mha_bin, filename_output_label_split, filename)
144             sitk.WriteImage(image,filename_output_mha)
145             sitk.WriteImage(image,filename_output_mhd)
146         if filename.startswith(type_mod):
147             if filename.endswith(".mhd"):
148                 image_path = os.path.join(filepath_input,
    filename)
149                 image_arr, spacing, origin  = load_itk(image_path
    )
150                 array_laser = add_laser_layer(image_arr,
    filename_output_label_split, filename)
151                 arrays.append(array_laser)
152     arrays_combined = np.array(arrays)
153     image_combined = sitk.GetImageFromArray(arrays_combined)
154     sitk.WriteImage(image_combined,filename_output_mha_self)
155     #sitk.WriteImage(image_combined,filename_output_mhd)
156     print("load_ALL - done")
157     return 0
158
159 def phantom_to_mha_2d(filename_output_mha_phantom,
    filename_output_mhd_phantom, filepath_input):
160     filepath = os.listdir(filepath_input)
161     filepath.sort()
162     for filename in filepath:
163         if filename.startswith("phantom"):
164             if filename.endswith(".mhd"):
165                 image_path = os.path.join(filepath_input,
```

```
    filename)
166                 image_arr, spacing, origin  = load_itk(image_path
    )
167                 print("im.sh", image_arr.shape)
168                 summarized_array = np.sum(image_arr, axis=0)
169                 print("im.sh final", summarized_array.shape)
170                 image = sitk.GetImageFromArray(summarized_array)
171                 sitk.WriteImage(image,filename_output_mha_phantom
    )
172                 sitk.WriteImage(image,filename_output_mhd_phantom
    )
173                 #image = mhd_to_mha(image_path,
    filename_output_mha_phantom, filename_output_mha_phantom,
    filename)
174     return 0
175
176 def load_ALL_data(filepath_input, filename_output_mha,
    filename_output_mhd, type_mod):
177     print("saved as: ", filename_output_mha)
178     print("load_ALL open")
179     arrays = []
180     filepath = os.listdir(filepath_input)
181     filepath.sort()
182     for filename in filepath:
183         if filename.startswith(type_mod):
184             if filename.endswith(".mhd"):
185                 image_path = os.path.join(filepath_input,
    filename)
186                 image_arr, spacing, origin  = load_itk(image_path
    )
187                 summarized_array = np.sum(image_arr, axis=0)
188                 arrays.append(summarized_array)
189     arrays_combined = np.array(arrays)
190     array_final = np.transpose(arrays_combined, axes=(2, 0, 1))
191     image_combined = sitk.GetImageFromArray(array_final)
192     sitk.WriteImage(image_combined,filename_output_mha)
193     sitk.WriteImage(image_combined,filename_output_mhd)
194     print("load_ALL - done")
195     return 0
196
197 def create_label_string():
```

```
198     result_dict = {}
199     result_dict["background"] = 0
200     for i in range(1, 256):
201         result_dict[str(i)] = i
202     result_json_string = json.dumps(result_dict)
203     with open("output1.json", "w") as f:
204         f.write(result_json_string)
205     return result_json_string
206 
207 def update_json(name, train_ind, test_ind):
208     label_string = create_label_string()
209     with open("dataset.json", "r") as jsonFile:
210         data = json.load(jsonFile)
211     data["numTraining"] = train_ind
212     data["numTest"] = test_ind
213     with open("dataset.json", "w") as jsonFile:
214         json.dump(data, jsonFile)
215     dst = os.path.join(name, "/dataset.json")
216     shutil.copy2("dataset.json", name)
217     print("update_json - done")
218     return 0
219 
220 def for_loop(path_saving, name1, dest_dir_train,
    dest_dir_train_label, dest_dir_test, dest_dir_test_label):
221     type_mod_data = "FMI-cfLights-surfEm-oCam-"
222     type_mod_label = "BLI-oCam-c"
223     i = 0
224     #___________train___________
225     print("for_loop TRAIN - starts")
226     filenames = os.listdir(dest_dir_train)
227     filelabels = os.listdir(dest_dir_train_label)
228     filenames.sort()
229     filelabels.sort()
230     for filename, filelabel in zip(filenames, filelabels):
231         image_path_in_itr = os.path.join(dest_dir_train, filename
    )
232         image_path_in_ltr = os.path.join(dest_dir_train_label,
    filelabel)
233         # only defining the nnUNet-specific-names of the files
234         if i<10:
235             filename_output_itr = path_saving+"/imagesTr/"+name1+
```

```
"_00"+str(i)+"_0000.mha"
236             filename_output_ltr = path_saving+"/labelsTr/"+name1+
"_00"+str(i)+".mha"
237             filename_output_mhd_itr = path_saving+"/mhd_data/
train/data/"+name1+"_00"+str(i)+"_0000.mhd"
238             filename_output_mhd_ltr = path_saving+"/mhd_data/
train/label/"+name1+"_00"+str(i)+".mhd"
239             filename_output_mha_nonbin = path_saving+"/
labels_nonbinaryTr/"+name1+"_00"+str(i)+".mha"
240             filename_output_mha_bin = path_saving+"/
labels_binaryTr/"+name1+"_00"+str(i)+".mha"
241             filename_output_mha_self = path_saving+"/
labels_selfTr/"+name1+"_00"+str(i)+".mha"
242             filename_output_mha_phantom = path_saving+"/phantom/
train/"+name1+"_00"+str(i)+".mha"
243             filename_output_mhd_phantom = path_saving+"/phantom/
train/"+name1+"_00"+str(i)+".mhd"
244         if 10 <= i < 100:
245             filename_output_itr = path_saving+"/imagesTr/"+name1+
"_0"+str(i)+"_0000.mha"
246             filename_output_ltr = path_saving+"/labelsTr/"+name1+
"_0"+str(i)+".mha"
247             filename_output_mhd_itr = path_saving+"/mhd_data/
train/data/"+name1+"_0"+str(i)+"_0000.mhd"
248             filename_output_mhd_ltr = path_saving+"/mhd_data/
train/label/"+name1+"_0"+str(i)+".mhd"
249             filename_output_mha_nonbin = path_saving+"/
labels_nonbinaryTr/"+name1+"_0"+str(i)+".mha"
250             filename_output_mha_bin = path_saving+"/
labels_binaryTr/"+name1+"_0"+str(i)+".mha"
251             filename_output_mha_self = path_saving+"/
labels_selfTr/"+name1+"_0"+str(i)+".mha"
252             filename_output_mha_phantom = path_saving+"/phantom/
train/"+name1+"_0"+str(i)+".mha"
253             filename_output_mhd_phantom = path_saving+"/phantom/
train/"+name1+"_0"+str(i)+".mhd"
254         filename_output_label_split = path_saving+"/mhd_data/
split/"
255
256         print("image_Train starts")
257         print("Filename:", filename)
```

```
258         load_ALL_data(image_path_in_itr, filename_output_itr,
    filename_output_mhd_itr, type_mod_data)
259         phantom_to_mha_2d(filename_output_mha_phantom,
    filename_output_mhd_phantom, image_path_in_itr)
260         print("label_Train starts")
261         print("Filelabel:", filelabel)
262         load_ALL_label(image_path_in_ltr, filename_output_ltr,
    filename_output_mha_self, type_mod_label,
    filename_output_label_split, filename_output_mha_nonbin,
    filename_output_mha_bin, filename_output_mha_self)
263         i += 1
264
265     k = i
266     #__________test__________
267     print("for_loop TEST - starts")
268     filenames = os.listdir(dest_dir_test)
269     filelabels = os.listdir(dest_dir_test_label)
270     filenames.sort()
271     filelabels.sort()
272     for filename, filelabel in zip(filenames, filelabels):
273         image_path_in_its = os.path.join(dest_dir_test, filename)
274         image_path_in_lts = os.path.join(dest_dir_test_label,
    filelabel)
275         # only defining the nnUNet-specific-names of the files
276         if k<10:
277             filename_output_its = path_saving+"/imagesTs/"+name1+
    "_00"+str(k)+"_0000.mha"
278             filename_output_lts = path_saving+"/test_label/"+
    name1+"_00"+str(k)+".mha"
279             filename_output_mhd_its = path_saving+"/mhd_data/test
    /data/"+name1+"_00"+str(k)+"_0000.mhd"
280             filename_output_mhd_lts = path_saving+"/mhd_data/test
    /label/"+name1+"_00"+str(k)+".mhd"
281             filename_output_mha_nonbin = path_saving+"/
    labels_nonbinaryTr/"+name1+"_00"+str(k)+".mha"
282             filename_output_mha_bin = path_saving+"/
    labels_binaryTr/"+name1+"_00"+str(k)+".mha"
283             filename_output_mha_self = path_saving+"/
    labels_selfTr/"+name1+"_00"+str(k)+".mha"
284             filename_output_mha_phantom = path_saving+"/phantom/
    test/"+name1+"_00"+str(k)+".mha"
```

```
285             filename_output_mhd_phantom = path_saving+"/phantom/
    test/"+name1+"_00"+str(k)+".mhd"
286 
287         if 10 <= k < 100:
288             filename_output_its = path_saving+"/imagesTs/"+name1+
    "_0"+str(k)+"_0000.mha"
289             filename_output_lts = path_saving+"/test_label/"+
    name1+"_0"+str(k)+".mha"
290             filename_output_mhd_its = path_saving+"/mhd_data/test
    /data/"+name1+"_0"+str(k)+"_0000.mhd"
291             filename_output_mhd_lts = path_saving+"/mhd_data/test
    /label/"+name1+"_0"+str(k)+".mhd"
292             filename_output_mha_nonbin = path_saving+"/
    labels_nonbinaryTr/"+name1+"_0"+str(k)+".mha"
293             filename_output_mha_bin = path_saving+"/
    labels_binaryTr/"+name1+"_0"+str(k)+".mha"
294             filename_output_mha_phantom = path_saving+"/phantom/
    test/"+name1+"_0"+str(k)+".mha"
295             filename_output_mhd_phantom = path_saving+"/phantom/
    test/"+name1+"_0"+str(k)+".mhd"
296             filename_output_mha_self = path_saving+"/
    labels_selfTr/"+name1+"_0"+str(k)+".mha"
297         k += 1
298         filename_output_label_split = path_saving+"/mhd_data/
    split/"
299         print("load_ALL for image_Test starts")
300         print("Filename:", filename)
301         load_ALL_data(image_path_in_its, filename_output_its,
    filename_output_mhd_its, type_mod_data)
302         phantom_to_mha_2d(filename_output_mha_phantom,
    filename_output_mhd_phantom, image_path_in_its)
303         print("load_ALL for label_Test starts")
304         print("Filelabel:", filelabel)
305         load_ALL_data(image_path_in_lts, filename_output_lts,
    filename_output_mha_self, type_mod_label)
306 
307     update_json(path_saving,i,k-i)
308     print("for_loop - done")
309     return 0
310 
311 def make_dir(name_file):
```

```
312     os.makedirs(name_file, exist_ok=True)
313     os.makedirs(name_file+"/imagesTr", exist_ok=True)
314     os.makedirs(name_file+"/labelsTr", exist_ok=True)
315     os.makedirs(name_file+"/imagesTs", exist_ok=True)
316     os.makedirs(name_file+"/test_label", exist_ok=True)
317     os.makedirs(name_file+"/mhd_data/train/data", exist_ok=True)
318     os.makedirs(name_file+"/mhd_data/train/label", exist_ok=True)
319     os.makedirs(name_file+"/mhd_data/split", exist_ok=True)
320     os.makedirs(name_file+"/mhd_data/test/data", exist_ok=True)
321     os.makedirs(name_file+"/mhd_data/test/label", exist_ok=True)
322     os.makedirs(name_file+"/labels_nonbinaryTr", exist_ok=True)
323     os.makedirs(name_file+"/labels_binaryTr", exist_ok=True)
324     os.makedirs(name_file+"/labels_selfTr", exist_ok=True)
325     os.makedirs(name_file+"/phantom/train/", exist_ok=True)
326     os.makedirs(name_file+"/phantom/test/", exist_ok=True)
327 
328     print("directories for nnUNet2 were made")
329     return 0
330 
331 #_____________________EDIT_____________________
332 
333 #_________organize simulation-output data according to label,
       training and testing data_____________
334 directory_home = '/home/svea/lipros/lipros_training/
       training_set_05/'
335 #directory_home = '/home/svea/lipros/lipros_training/
       training_set_02/output_simulation/'
336 source_directory_simulation = directory_home+'2024-03-08-lipros
       -30-sphere-for-svea'
337 source_directory_label = source_directory_simulation+'/neu
       -2024-03-08-15-31-02-lipros-30-sphere-BLI'
338 source_directory_data = source_directory_simulation+'/neu
       -2024-03-08-16-22-30-lipros-30-sphere-FMI'
339 destination_directory_train = directory_home+'train'
340 destination_directory_train_label = directory_home+'train_label'
341 destination_directory_test = directory_home+'testing'
342 destination_directory_test_label = directory_home+'testing_label'
343 
344 #split_data_in_directory(source_directory_simulation,
       source_directory_data, source_directory_label)
345 copy_every_third_directory(source_directory_data,
```

```
    source_directory_label, destination_directory_train,
    destination_directory_train_label, destination_directory_test,
    destination_directory_test_label)

#________prepare data as input for nnUNet2________
path_saving = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset026_CTpositionRadon'
name_saving = "Dataset026"

make_dir(path_saving)
for_loop(path_saving, name_saving, destination_directory_train,
    destination_directory_train_label, destination_directory_test,
    destination_directory_test_label )
print("job done")
```

Listing E.10: Preperation of data for nnUNet for Datasets 20 to 29

E.11 Preperation of data for nnUNet for Datasets 30 to 41

```
from matplotlib import pyplot as plt
import matplotlib as mpl
import matplotlib.pylab as plt
import SimpleITK as sitk
import numpy as np
import cv2
from PIL import Image
import os
import json
from skimage.transform import radon, rescale, resize
import shutil

def split_data_in_directory(src_dir_home, dest_dir_data,
    dest_dir_label):
    if not os.path.exists(src_dir_home):
        print("Source directory does not exist.")
        return
    if os.path.exists(dest_dir_data):
        print("Dest directory does exist.")
        return
    if os.path.exists(dest_dir_label):
        print("Dest directory does exist.")
```

```
22          return
23      os.makedirs(dest_dir_data, exist_ok=True)
24      os.makedirs(dest_dir_label, exist_ok=True)
25
26      dirs = [d for d in os.listdir(src_dir_home) if os.path.isdir(
    os.path.join(src_dir_home, d))]
27      dirs.sort()
28      for i, directory in enumerate(dirs):
29          if directory.startswith("lipros-30-sphere_"):
30              src_directory = os.path.join(src_dir_home, directory)
31              dest_directory = os.path.join(dest_dir_data,
    directory)
32              shutil.copytree(src_directory, dest_directory)
33          else:
34              src_directory = os.path.join(src_dir_home, directory)
35              dest_directory = os.path.join(dest_dir_label,
    directory)
36              shutil.copytree(src_directory, dest_directory)
37      print(f"Copied {src_directory} to {dest_directory}")
38      print("split_data_in_directory - done")
39
40  def copy_every_third_directory(src_dir_data, src_dir_label,
    dest_dir_train, dest_dir_train_label, dest_dir_test,
    dest_dir_test_label):
41      # Check if the source directory exists
42      if not os.path.exists(src_dir_data):
43          print("Source directory data does not exist.")
44          return
45      if not os.path.exists(src_dir_label):
46          print("Source directory label does not exist.")
47          return
48      if os.path.exists(dest_dir_train):
49          print("Dest directory does exist.")
50          return
51
52      os.makedirs(dest_dir_train, exist_ok=True)
53      os.makedirs(dest_dir_train_label, exist_ok=True)
54      os.makedirs(dest_dir_test, exist_ok=True)
55      os.makedirs(dest_dir_test_label, exist_ok=True)
56
57      dirs = [d for d in os.listdir(src_dir_data) if os.path.isdir(
```

```
    os.path.join(src_dir_data, d))]
58      dirs.sort()
59      for i, directory in enumerate(dirs):
60              if (i + 1) % 3 == 0:
61                  src_directory = os.path.join(src_dir_data,
    directory)
62                  dest_directory = os.path.join(dest_dir_test,
    directory)
63                  shutil.copytree(src_directory, dest_directory)
64              else:
65                  src_directory = os.path.join(src_dir_data,
    directory)
66                  dest_directory = os.path.join(dest_dir_train,
    directory)
67                  shutil.copytree(src_directory, dest_directory)
68      print(f"Copied {src_directory} to {dest_directory}")
69
70      dirs = [d for d in os.listdir(src_dir_label) if os.path.isdir
    (os.path.join(src_dir_label, d))]
71      dirs.sort()
72      for i, directory in enumerate(dirs):
73              if (i + 1) % 3 == 0:
74                  src_directory = os.path.join(src_dir_label,
    directory)
75                  dest_directory = os.path.join(dest_dir_test_label
    , directory)
76                  shutil.copytree(src_directory, dest_directory)
77              else:
78                  src_directory = os.path.join(src_dir_label,
    directory)
79                  dest_directory = os.path.join(
    dest_dir_train_label, directory)
80                  shutil.copytree(src_directory, dest_directory)
81
82      print(f"Copied {src_directory} to {dest_directory}")
83      print("copy_every_third_directory - done")
84
85  def add_laser_layer(image_arr, filename_output_label_split,
    filename):
86      arrays = []
87      label = filename_output_label_split+filename
```

```
88      for i in range(0,30):
89          arrays.append(image_arr)
90      arrays_combined = np.array(arrays)
91      image_combined = sitk.GetImageFromArray(arrays_combined)
92      sitk.WriteImage(image_combined,label)
93      image_arr, spacing, origin  = load_itk(label)
94      image_arr = np.transpose(image_arr, axes=(0, 2, 1))
95      return image_arr
96
97  def add_laser_layer_bin(image_arr, filename_output_label_split,
    filename):
98      arrays = []
99      label_split = filename_output_label_split+filename
100     for i in range(0,30):
101         arrays.append(image_arr)
102     arrays_combined = np.array(arrays)
103     image_combined = sitk.GetImageFromArray(arrays_combined)
104     sitk.WriteImage(image_combined,label_split)
105     image_arr, spacing, origin  = load_itk(label_split)
106     image_arr = np.transpose(image_arr, axes=(1, 0, 2))
107     return image_arr
108
109 def load_itk(filename):
110     itkimage = sitk.ReadImage(filename)
111     image = sitk.GetArrayFromImage(itkimage)
112     origin = np.array(list(reversed(itkimage.GetOrigin())))
113     spacing = np.array(list(reversed(itkimage.GetSpacing())))
114     return image, spacing, origin
115
116 def mhd_to_mha(image_path, filename_output_mha_bin,
    filename_output_label_split, filename, filename_output_mha):
117     image_arr, spacing, origin  = load_itk(image_path)
118     #image_arr = resize(image_arr, (1,30, 55),anti_aliasing=False
    )
119     resized_array_3d = np.zeros((1, 30, 55), dtype=np.uint8)
120     for i in range(image_arr.shape[0]):
121         resized_array_3d[i,:,:] = cv2.resize(image_arr[i,:,:].
    astype(np.uint8), (55, 30), interpolation=cv2.INTER_NEAREST)
122
123     arr = np.reshape(resized_array_3d, (30, 55))
124     arr = Image.fromarray(cv2.normalize(arr, None, 255, 0, cv2.
```

```
    NORM_MINMAX, cv2.CV_8U))
125     arr.save(filename_output_mha_bin[:-4]+".png")
126     arr.save(filename_output_mha[:-4]+".png")
127 
128     image = sitk.GetImageFromArray(resized_array_3d)
129     sitk.WriteImage(image,filename_output_mha)
130     return image
131 
132 def load_ALL_label(filepath_input, filename_output_mha,
    filename_output_mhd, type_mod, filename_output_label_split,
    filename_output_mha_nonbin, filename_output_mha_bin,
    filename_output_mha_self):
133     print("saved as: ", filename_output_mha)
134     print("load_ALL open")
135     arrays = []
136     filepath = os.listdir(filepath_input)
137     filepath.sort()
138     for filename in filepath:
139         if filename.endswith("oCams.mhd"):
140             image_path = os.path.join(filepath_input, filename)
141             image = mhd_to_mha(image_path,
    filename_output_mha_nonbin, filename_output_label_split,
    filename, filename_output_mha_nonbin)
142         if filename.endswith("binarized-10.mhd"):
143             image_path = os.path.join(filepath_input, filename)
144             image = mhd_to_mha(image_path,
    filename_output_mha_bin, filename_output_label_split, filename,
     filename_output_mha)
145             sitk.WriteImage(image,filename_output_mha)
146             sitk.WriteImage(image,filename_output_mhd)
147     print("load_ALL - done")
148     return 0
149 
150 def phantom_to_mha_2d(filename_output_mha_phantom,
    filename_output_mhd_phantom, filepath_input):
151     filepath = os.listdir(filepath_input)
152     filepath.sort()
153     for filename in filepath:
154         if filename.startswith("phantom"):
155             if filename.endswith(".mhd"):
156                 image_path = os.path.join(filepath_input,
```

```
    filename)
157                 image_arr, spacing, origin  = load_itk(image_path
    )
158                 print("im.sh", image_arr.shape)
159                 summarized_array = np.sum(image_arr, axis=0)
160                 print("im.sh final", summarized_array.shape)
161                 im = cv2.normalize(summarized_array, None, 255,
    0, cv2.NORM_MINMAX, cv2.CV_8U)
162                 for i in range(im.shape[0]):
163                     for j in range(im.shape[1]):
164                         if im[i, j] == 232:
165                             im[i, j] = 100
166                 im = Image.fromarray(im)
167                 im.save(filename_output_mha_phantom[:-4]+".png")
168                 image = sitk.GetImageFromArray(summarized_array)
169                 sitk.WriteImage(image,filename_output_mha_phantom
    )
170                 sitk.WriteImage(image,filename_output_mhd_phantom
    )
171                 #image = mhd_to_mha(image_path,
    filename_output_mha_phantom, filename_output_mha_phantom,
    filename)
172     return 0
173
174 def load_ALL_data(filepath_input, filename_output_mha,
    filename_output_mhd, type_mod):
175     print("saved as: ", filename_output_mha)
176     print("load_ALL open")
177     arrays = []
178     filepath = os.listdir(filepath_input)
179     filepath.sort()
180     for filename in filepath:
181         if filename.endswith(type_mod+".mhd"):
182             if filename.endswith(type_mod+".mhd"):
183                 image_path = os.path.join(filepath_input,
    filename)
184                 image_arr, spacing, origin  = load_itk(image_path
    )
185                 #print("im.sh", image_arr.shape)
186                 summarized_array = np.sum(image_arr, axis=0)
187                 #print("im.sh final", summarized_array.shape)
```

```
188                 arrays.append(summarized_array)
189     arrays_combined = np.array(arrays)
190     if not type_mod.startswith('BLI'):
191         array_final = np.transpose(arrays_combined, axes=(2, 0,
    1))
192     array_final = resize(array_final, (1,30, 55),anti_aliasing=
    True)
193     arr = np.reshape(array_final, (30, 55))
194     arr = Image.fromarray(cv2.normalize(arr, None, 255, 0, cv2.
    NORM_MINMAX, cv2.CV_8U))
195     arr.save(filename_output_mha[:-4]+".png")
196     image_combined = sitk.GetImageFromArray(array_final)
197     sitk.WriteImage(image_combined,filename_output_mha)
198     sitk.WriteImage(image_combined,filename_output_mhd)
199     print("load_ALL - done")
200     return 0
201
202 def create_label_string():
203     result_dict = {}
204     result_dict["background"] = 0
205     for i in range(1, 256):
206         result_dict[str(i)] = i
207     result_json_string = json.dumps(result_dict)
208     with open("output1.json", "w") as f:
209         f.write(result_json_string)
210     return result_json_string
211
212 def update_json(name, train_ind, test_ind):
213     label_string = create_label_string()
214     with open("dataset.json", "r") as jsonFile:
215         data = json.load(jsonFile)
216     data["numTraining"] = train_ind
217     data["numTest"] = test_ind
218     with open("dataset.json", "w") as jsonFile:
219         json.dump(data, jsonFile)
220     dst = os.path.join(name, "/dataset.json")
221     shutil.copy2("dataset.json", name)
222     print("update_json - done")
223     return 0
224
225 def for_loop(path_saving, name1, dest_dir_train,
```

```
dest_dir_train_label, dest_dir_test, dest_dir_test_label):
    #type_mod_data = "- l "
    type_mod_data = "- l -AWGN"
    type_mod_label = "BLI-oCam-"
    i = 0
    #___________train___________
    print("for_loop TRAIN - starts")
    filenames = os.listdir(dest_dir_train)
    filelabels = os.listdir(dest_dir_train_label)
    filenames.sort()
    filelabels.sort()
    for filename, filelabel in zip(filenames, filelabels):
        image_path_in_itr = os.path.join(dest_dir_train, filename
)
        image_path_in_ltr = os.path.join(dest_dir_train_label,
filelabel)
        # only defining the nnUNet-specific-names of the files
        if i<10:
            filename_output_itr = path_saving+"/imagesTr/"+name1+
"_00"+str(i)+"_0000.mha"
            filename_output_ltr = path_saving+"/labelsTr/"+name1+
"_00"+str(i)+".mha"
            filename_output_mhd_itr = path_saving+"/mhd_data/
train/data/"+name1+"_00"+str(i)+"_0000.mhd"
            filename_output_mhd_ltr = path_saving+"/mhd_data/
train/label/"+name1+"_00"+str(i)+".mhd"
            filename_output_mha_nonbin = path_saving+"/
labels_nonbinaryTr/"+name1+"_00"+str(i)+".mha"
            filename_output_mha_bin = path_saving+"/
labels_binaryTr/"+name1+"_00"+str(i)+".mha"
            filename_output_mha_self = path_saving+"/
labels_selfTr/"+name1+"_00"+str(i)+".mha"
            filename_output_mha_phantom = path_saving+"/phantom/
train/"+name1+"_00"+str(i)+".mha"
            filename_output_mhd_phantom = path_saving+"/phantom/
train/"+name1+"_00"+str(i)+".mhd"
        if 10 <= i < 100:
            filename_output_itr = path_saving+"/imagesTr/"+name1+
"_0"+str(i)+"_0000.mha"
            filename_output_ltr = path_saving+"/labelsTr/"+name1+
"_0"+str(i)+".mha"
```

```
253             filename_output_mhd_itr = path_saving+"/mhd_data/
    train/data/"+name1+"_0"+str(i)+"_0000.mhd"
254             filename_output_mhd_ltr = path_saving+"/mhd_data/
    train/label/"+name1+"_0"+str(i)+".mhd"
255             filename_output_mha_nonbin = path_saving+"/
    labels_nonbinaryTr/"+name1+"_0"+str(i)+".mha"
256             filename_output_mha_bin = path_saving+"/
    labels_binaryTr/"+name1+"_0"+str(i)+".mha"
257             filename_output_mha_self = path_saving+"/
    labels_selfTr/"+name1+"_0"+str(i)+".mha"
258             filename_output_mha_phantom = path_saving+"/phantom/
    train/"+name1+"_0"+str(i)+".mha"
259             filename_output_mhd_phantom = path_saving+"/phantom/
    train/"+name1+"_0"+str(i)+".mhd"
260         filename_output_label_split = path_saving+"/mhd_data/
    split/"
261
262         print("image_Train starts")
263         print("Filename:", filename)
264         load_ALL_data(image_path_in_itr, filename_output_itr,
    filename_output_mhd_itr, type_mod_data)
265         phantom_to_mha_2d(filename_output_mha_phantom,
    filename_output_mhd_phantom, image_path_in_itr)
266         print("label_Train starts")
267         print("Filelabel:", filelabel)
268         load_ALL_label(image_path_in_ltr, filename_output_ltr,
    filename_output_mha_self, type_mod_label,
    filename_output_label_split, filename_output_mha_nonbin,
    filename_output_mha_bin, filename_output_mha_self)
269         i += 1
270
271     k = i
272     #__________test__________
273     print("for_loop TEST - starts")
274     filenames = os.listdir(dest_dir_test)
275     filelabels = os.listdir(dest_dir_test_label)
276     filenames.sort()
277     filelabels.sort()
278     for filename, filelabel in zip(filenames, filelabels):
279         image_path_in_its = os.path.join(dest_dir_test, filename)
280         image_path_in_lts = os.path.join(dest_dir_test_label,
```

```
filelabel)
281         # only defining the nnUNet-specific-names of the files
282         if k<10:
283             filename_output_its = path_saving+"/imagesTs/"+name1+
"_00"+str(k)+"_0000.mha"
284             filename_output_lts = path_saving+"/test_label/"+
name1+"_00"+str(k)+".mha"
285             filename_output_mhd_its = path_saving+"/mhd_data/test
/data/"+name1+"_00"+str(k)+"_0000.mhd"
286             filename_output_mhd_lts = path_saving+"/mhd_data/test
/label/"+name1+"_00"+str(k)+".mhd"
287             filename_output_mha_nonbin = path_saving+"/
labels_nonbinaryTr/"+name1+"_00"+str(k)+".mha"
288             filename_output_mha_bin = path_saving+"/
labels_binaryTr/"+name1+"_00"+str(k)+".mha"
289             filename_output_mha_self = path_saving+"/
labels_selfTr/"+name1+"_00"+str(k)+".mha"
290             filename_output_mha_phantom = path_saving+"/phantom/
test/"+name1+"_00"+str(k)+".mha"
291             filename_output_mhd_phantom = path_saving+"/phantom/
test/"+name1+"_00"+str(k)+".mhd"
292
293         if 10 <= k < 100:
294             filename_output_its = path_saving+"/imagesTs/"+name1+
"_0"+str(k)+"_0000.mha"
295             filename_output_lts = path_saving+"/test_label/"+
name1+"_0"+str(k)+".mha"
296             filename_output_mhd_its = path_saving+"/mhd_data/test
/data/"+name1+"_0"+str(k)+"_0000.mhd"
297             filename_output_mhd_lts = path_saving+"/mhd_data/test
/label/"+name1+"_0"+str(k)+".mhd"
298             filename_output_mha_nonbin = path_saving+"/
labels_nonbinaryTr/"+name1+"_0"+str(k)+".mha"
299             filename_output_mha_bin = path_saving+"/
labels_binaryTr/"+name1+"_0"+str(k)+".mha"
300             filename_output_mha_phantom = path_saving+"/phantom/
test/"+name1+"_0"+str(k)+".mha"
301             filename_output_mhd_phantom = path_saving+"/phantom/
test/"+name1+"_0"+str(k)+".mhd"
302             filename_output_mha_self = path_saving+"/
labels_selfTr/"+name1+"_0"+str(k)+".mha"
```

```
303         k += 1
304         filename_output_label_split = path_saving+"/mhd_data/
    split/"
305         print("load_ALL for image_Test starts")
306         print("Filename:", filename)
307         load_ALL_data(image_path_in_its, filename_output_its,
    filename_output_mhd_its, type_mod_data)
308         phantom_to_mha_2d(filename_output_mha_phantom,
    filename_output_mhd_phantom, image_path_in_its)
309         print("load_ALL for label_Test starts")
310         print("Filelabel:", filelabel)
311         load_ALL_label(image_path_in_lts, filename_output_lts,
    filename_output_mha_self, type_mod_label,
    filename_output_label_split, filename_output_mha_nonbin,
    filename_output_mha_bin, filename_output_mha_self)
312         #load_ALL_data(image_path_in_lts, filename_output_lts,
    filename_output_mha_self, type_mod_label)
313
314
315     update_json(path_saving,i,k-i)
316     print("for_loop - done")
317     return 0
318
319 def make_dir(name_file):
320     os.makedirs(name_file, exist_ok=True)
321     os.makedirs(name_file+"/imagesTr", exist_ok=True)
322     os.makedirs(name_file+"/labelsTr", exist_ok=True)
323     os.makedirs(name_file+"/imagesTs", exist_ok=True)
324     os.makedirs(name_file+"/test_label", exist_ok=True)
325     os.makedirs(name_file+"/mhd_data/train/data", exist_ok=True)
326     os.makedirs(name_file+"/mhd_data/train/label", exist_ok=True)
327     os.makedirs(name_file+"/mhd_data/split", exist_ok=True)
328     os.makedirs(name_file+"/mhd_data/test/data", exist_ok=True)
329     os.makedirs(name_file+"/mhd_data/test/label", exist_ok=True)
330     os.makedirs(name_file+"/labels_nonbinaryTr", exist_ok=True)
331     os.makedirs(name_file+"/labels_binaryTr", exist_ok=True)
332     os.makedirs(name_file+"/labels_selfTr", exist_ok=True)
333     os.makedirs(name_file+"/phantom/train/", exist_ok=True)
334     os.makedirs(name_file+"/phantom/test/", exist_ok=True)
335
336     print("directories for nnUNet2 were made")
```

```
337     return 0
338
339 #_____________________EDIT_____________________
340
341 #_________organize simulation-output data according to label,
    training and testing data_____________
342 directory_home = '/home/svea/lipros/lipros_training/
    training_set_06/'
343 source_directory_simulation = directory_home
344 source_directory_label = source_directory_simulation+'
    2024-03-25-16-08-22-lipros-30-sphere'
345 source_directory_data = source_directory_simulation+'svea
    -2024-03-22-16-06-06-lipros-30-sphere'
346 #source_directory_data = source_directory_simulation+'test_old'
347 destination_directory_train = directory_home+'test_oldtrain'
348 #destination_directory_train = directory_home+'train'
349 destination_directory_train_label = directory_home+'train_label'
350 destination_directory_test = directory_home+'test'
351 destination_directory_test_label = directory_home+'testing_label'
352
353 #________prepare data as input for nnUNet2________
354 path_saving = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset040_FMIposition55'
355 name_saving = "Dataset040"
356
357 make_dir(path_saving)
358 for_loop(path_saving, name_saving, destination_directory_train,
    destination_directory_train_label, destination_directory_test,
    destination_directory_test_label )
359 print("job done")
```

Listing E.11: Preperation of data for nnUNet for Datasets 30 to 41

E.12 Converting .mhd to .png and .mhd to .mha

```
1 import subprocess
2 import SimpleITK as sitk
3 import numpy as np
4 import cv2
5 from PIL import Image
6 import os
7
```

```
def load_itk(filename):
    itkimage = sitk.ReadImage(filename)
    image = sitk.GetArrayFromImage(itkimage)
    origin = np.array(list(reversed(itkimage.GetOrigin())))
    spacing = np.array(list(reversed(itkimage.GetSpacing())))
    return image, spacing, origin

def mhd_to_mha(image_path, filename_output_mha):
    image_arr, spacing, origin  = load_itk(image_path)
    image = sitk.GetImageFromArray(image_arr)
    sitk.WriteImage(image,filename_output_mha)
    return 0

def mh_to_png(path, path_saving, filename):
    mha = sitk.ReadImage(path)
    array0 = sitk.GetArrayFromImage(mha)
    #array = np.reshape(array0, (30, 110))
    array = np.reshape(array0, (30, 55))
    image = Image.fromarray(cv2.normalize(array, None, 255, 0,
   cv2.NORM_MINMAX, cv2.CV_8U))
    image.save(path_saving+filename[:-4]+".png")
    return 0

def iterate_folder(filepath_input, path_saving):
    os.makedirs(filepath_input+'/png', exist_ok=True)
    filepath = os.listdir(filepath_input)
    filepath.sort()
    for filename in filepath:
        if filename.endswith(".mha"):
            image_path = os.path.join(filepath_input, filename)
            if filepath_input == path_saving:
                path_saving = filepath_input+'/png/'
            mh_to_png(image_path, path_saving, filename)
    return 0

def mh_to_png_ph(path, path_saving, filename):
    mha = sitk.ReadImage(path)
    array = sitk.GetArrayFromImage(mha)
    print("sh", array.shape)
    #array = np.reshape(array0, (30, 110))
    im = cv2.normalize(array, None, 255, 0, cv2.NORM_MINMAX, cv2.
```

```
    CV_8U)
    for i in range(im.shape[0]):
        for j in range(im.shape[1]):
            if im[i, j] == 232:
                im[i, j] = 100
    image = Image.fromarray(im)
    image.save(path_saving+filename[:-4]+".png")
    return 0

def iterate_folder_phant(filepath_input, path_saving):
    os.makedirs(filepath_input+'/png', exist_ok=True)
    filepath = os.listdir(filepath_input)
    filepath.sort()
    for filename in filepath:
        if filename.endswith(".mhd"):
            image_path = os.path.join(filepath_input, filename)
            if filepath_input == path_saving:
                path_saving = filepath_input+'/png/'
            mh_to_png_ph(image_path, path_saving, filename)
    return 0

imgTr= '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset029_FMI2position_awgn/imagesTr'
imgTs = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset029_FMI2position_awgn/imagesTs'
labTr = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset029_FMI2position_awgn/labelsTr'

phantTs = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset029_FMI2position_awgn/phantom/test'
phantTr = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
    Dataset029_FMI2position_awgn/phantom/train'

iterate_folder(imgTr, imgTr)
iterate_folder(imgTs, imgTs)
iterate_folder(labTr, labTr)

iterate_folder_phant(phantTs, phantTs)
iterate_folder_phant(phantTr, phantTr)
```

Listing E.12: Converting .mhd to .png and .mhd to .mha

E.13 Converting .png to .mha and changing the name of training and testing data of nnUNet

```
import numpy as np
import SimpleITK as sitk
from PIL import Image
import cv2
import os

def change_name_png(image_path, path_saving, filename, new_name):
    image = Image.open(image_path)
    image.save(new_name)
    return 0

def change_name_mha(image_path, path_saving, filename, new_name):
    image_arr, spacing, origin  = load_itk(image_path)
    image = sitk.GetImageFromArray(image_arr)
    #sitk.WriteImage(image,new_name)
    return 0

def iterate_folder_change_name(filepath_input, path_saving):
    filepath = os.listdir(filepath_input)
    filepath.sort()
    for filename in filepath:
        new_name = path_saving + filename[:8]+'41'+filename[10:]
        if filename.endswith(".png"):
            print(new_name)
            image_path = os.path.join(filepath_input, filename)
            print(image_path)
            change_name_png(image_path, path_saving, filename,
    new_name)
        if filename.endswith(".mha"):
            image_path = os.path.join(filepath_input, filename)
            change_name_mha(image_path, path_saving, filename,
    new_name)
    return 0

def png_to_mh(path, path_saving, filename):
    image = Image.open(path)
    array0 = np.array(image)
    # Add an additional dimension to match the desired shape (30,
```

```
    110, 1)
    array0 = np.expand_dims(array0, axis=-1)
    print("shape", array0.shape)
    array0 = np.reshape(array0, (30, 55, 1))  # Assuming the original shape was (30, 55)
    print("shape", array0.shape)
    array0 = array0.astype(np.float32)
    array0 = (array0 - np.min(array0)) / (np.max(array0) - np.min(array0))
    #array0 = cv2.normalize(array0, None, 1.0, 0.0, cv2.NORM_MINMAX, cv2.CV_32F)
    print("shape", array0.shape)
    array0 = np.transpose(array0, axes=(2, 0, 1))
    print("shape", array0.shape)

    mha = sitk.GetImageFromArray(array0)
    #sitk.WriteImage(mha, path_saving + filename[:9]+'9'+filename[11:-4] + ".mha")
    sitk.WriteImage(mha, path_saving + filename[:9]+'9'+filename[10:-4] + ".mhd")
    return 0

def iterate_folder(filepath_input, path_saving):
    filepath = os.listdir(filepath_input)
    filepath.sort()
    for filename in filepath:
        if filename.endswith(".png"):
            image_path = os.path.join(filepath_input, filename)
            png_to_mh(image_path, path_saving, filename)
    return 0

path_to_png = "/home/svea/Documents/nnUNetv2/nnUNet_raw/Dataset038_addTwo/imagesTr"
path_to_save_mha = "/home/svea/Documents/nnUNetv2/nnUNet_raw/Dataset039_addTwo/mha_imTr/"
iterate_folder(path_to_png, path_to_save_mha)

path_to_png = "/home/svea/Documents/nnUNetv2/nnUNet_raw/Dataset038_addTwo/imagesTs"
path_to_save_mha = "/home/svea/Documents/nnUNetv2/nnUNet_raw/Dataset039_addTwo/mha_imTs/"
```

```
iterate_folder(path_to_png, path_to_save_mha)

path_to_names = '/home/svea/Documents/nnUNetv2/nnUNet_raw/
   Dataset041_addTwo_noise/labelsTr'
iterate_folder_change_name(path_to_names, path_to_names+'/')
```

Listing E.13: Converting .png to .mha and changing the name of training and testing data of nnUNet

E.14 Bash-script for running nnUNet

```
DATASET_NAME_OR_ID="35"
DATASET_NAME_OR_ID_2="36"
DATASET_NAME_OR_ID_3="37"
DATASET_NAME_OR_ID_4="38"
TYPE="2d"

A="nohup nnUNetv2_train $DATASET_NAME_OR_ID $TYPE 0 --c"
B="nohup nnUNetv2_train $DATASET_NAME_OR_ID $TYPE 1 --c"
C="nohup nnUNetv2_train $DATASET_NAME_OR_ID $TYPE 2 --c"
D="nohup nnUNetv2_train $DATASET_NAME_OR_ID $TYPE 3 --c"
E="nohup nnUNetv2_train $DATASET_NAME_OR_ID $TYPE 4 --c"

A2="nohup nnUNetv2_train $DATASET_NAME_OR_ID_2 $TYPE 0 --c"
B2="nohup nnUNetv2_train $DATASET_NAME_OR_ID_2 $TYPE 1 --c"
C2="nohup nnUNetv2_train $DATASET_NAME_OR_ID_2 $TYPE 2 --c"
D2="nohup nnUNetv2_train $DATASET_NAME_OR_ID_2 $TYPE 3 --c"
E2="nohup nnUNetv2_train $DATASET_NAME_OR_ID_2 $TYPE 4 --c"

A3="nohup nnUNetv2_train $DATASET_NAME_OR_ID_3 $TYPE 0 --c"
B3="nohup nnUNetv2_train $DATASET_NAME_OR_ID_3 $TYPE 1 --c"
C3="nohup nnUNetv2_train $DATASET_NAME_OR_ID_3 $TYPE 2 --c"
D3="nohup nnUNetv2_train $DATASET_NAME_OR_ID_3 $TYPE 3 --c"
E3="nohup nnUNetv2_train $DATASET_NAME_OR_ID_3 $TYPE 4 --c"

A4="nohup nnUNetv2_train $DATASET_NAME_OR_ID_4 $TYPE 0 --c"
B4="nohup nnUNetv2_train $DATASET_NAME_OR_ID_4 $TYPE 1 --c"
C4="nohup nnUNetv2_train $DATASET_NAME_OR_ID_4 $TYPE 2 --c"
D4="nohup nnUNetv2_train $DATASET_NAME_OR_ID_4 $TYPE 3 --c"
E4="nohup nnUNetv2_train $DATASET_NAME_OR_ID_4 $TYPE 4 --c"

$A && $B && $C && $D && $E && $A2 && $B2 && $C2 && $D2 && $E2 &&
```

```
$A3 && $B3 && $C3 && $D3 && $E3 && $A4 && $B4 && $C4 && $D4 &&
$E4
```

Listing E.14: Bash-script for running nnUNet

Appendix F

Declaration

I confirm that I have written this thesis independently and have not used any sources or aids other than those specified.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, 20.05.2024

Svea Victoria Strassburger