Implementation and Training of Convolutional Neural Networks for the Segmentation of Brain Structures

Trygve Sebastian Birknes

Supervisors: Camilla Hanquist Stokkevåg and Helge Egil Seime Pettersen

Master thesis in medical technology

Department of Physics and Technology

University of Bergen

June 2022

Abstract

Precise delivery of radiotherapy depends on accurate segmentation of the anatomical structures surrounding the cancer tissue. With increasing knowledge of radio-sensitivity of critical brain structures, more detailed contouring of a range of structures is required. Manual segmentation is time-consuming, and research into methods for auto segmentation has advanced in the past decade. This thesis presents a general-purpose convolutional neural network with the U-net architecture for auto-segmenting the brain, brainstem, Papez Circuit, and right hippocampus. Several different models were trained using T1 MRI, T2 MRI, and CT images to compare the performance of models trained with the different modalities. Low-level preprocessing was done to the images before training, and the Dice score measured model performance. The best performing model for segmentation of the full brain resulted in a Dice score of 0.98, whereas the segmentation of the brainstem achieved a Dice score of 0.73. Furthermore, segmentation of the complex structure Papez Circuit attained Dice score of 0.52, and segmentation of the hippocampus resulted in 0.49. The selected model performed well in segmentation of the full brain and decent for the brainstem compared to similar studies. In contrast, the segmentation results for the hippocampus were slightly lower than previously reported results. No comparison was found for the segmentation results of the Papez Circuit. More preprocessing and patient data is necessary to provide accurate segmentation of the smaller structures. The dataset presented a few problems, and it was discovered that a similar acquisition method for image sequences gives better results. The network architecture provides a solid framework for segmentation.

Acknowledgments

Frist, I would like to express my sincerest gratitude to my supervisors, Associate Professor Camilla Hanquist Stokkevåg and PhD Helge Egil Seime Pettersen. Thank you, Camilla, for your guidance, support, and feedback through this project and for introducing me to the subject. To Helge, thank you for all your feedback, for all the discussions around the topic of this thesis and your help with all the programming errors I could not fix by myself.

I would like to thank all my fellow students at room 534, the past two years would not have been the same without you. I would like to give a special thanks to Karoline Lewinsen and Espen J. Folkedal for our five years together as medical technology students. We have had a great time together, and we made a great team. In addition, I want to thank Filip Bjurstrøm for helping out with programming and discussing topics surrounding the overlapping parts of our projects. I wish you all the best of luck in your future endeavors as we all move on to the next chapter of our careers.

I would also like to acknowledge the Medical Physics group at UiB for all their help and always being available to answer questions. This past year has been more fun because of you.

Finally, I want to thank my family for all the support over the years. I want to thank my wife, Maria A. Birknes for always supporting me, and encouraging me to work harder every day. I also want to thank her for taking such good care of our daughter Ellinor Birknes when I had to work long hours during this project.

Trygve Sebastian Birknes

Bergen, June 2022

Table of contents

Abbreviations

CNS Central nervous system

MRI Magnetic resonance imaging

CT Computed Tomography

CNN Convolutional neural network

LET Linear energy transfer

RBE Relative biological effectivenes

OAR Organs at risk

ML Machine learning

1 Introduction

One in three will be diagnosed with at least one type of cancer in their lifetime , and more than 35.000 were diagnosed with cancer in 2020, [1]. Of those, 957 individuals were diagnosed with cancer in the central nervous system (CNS), including 72 children or young adults aged 19 and younger [1]. The five-year survival rate for CNS tumors has steadily increased in the past few decades, and the 5 year survival rate in Norway is close to 80% for patients 19 and younger [1]. The rise in survival rate has made focusing on preventing adverse effects that might follow cancer treatment more relevant.

There are three primary methods used to treat cancer: radiation therapy, surgery, and chemotherapy, [1] in addition to newer treatments, such as immunotherapy. In 2010, 42.5% of patients who received cancer treatment underwent radiotherapy [2]. Radiation therapy is an effective way to treat many types of cancer, but the treatment method also has the potential to cause adverse effects. 50%-90% of patients who survive more than six months after receiving radiation therapy for a CNS tumor exhibit signs of cognitive dysfunction, including memory problems, problems with processing speed, and shortened attention span [3]. While it was previously thought that the radiosensitivity of the brain was relatively uniform within large brain regions [4], recent studies have shown this is not the case [6] - [8]. These studies have shown that the decline in cognition may depend on the dose delivered to more specific, smaller areas.

Anatomical structures, including the tumor volume and organs at risk (OAR) must be precisely defined in the treatment software in order to administer radiotherapy efficiently and safely. Manual or semi-automatic delineation is commonly performed on Computed Tomography (CT) scans in combination with Magnetic Resonance Imaging (MRI) and or PET scans. CT images are the gold standard for radiation therapy planning. Segmentation is time-consuming work and is usually done by experts in the field [8] - [9]. As computer power and algorithms

have advanced in the past two decades, research into auto-segmentation methods has advanced. New algorithms like Convolutional Neural Networks (CNN), functions like dropout and pooling, and access to large datasets increase the possibilities. A study reviewing published articles on auto-segmentation related to radiation therapy revealed 27 published works on auto-segmentation for head and neck areas, with only two presented clinical integration [10].

With the increase in survival rates, side effects of radiation therapy have become a more important aspect of radiation therapy planning. Estimating the delivered radiation dose to smaller, more specific structures is becoming and may become even more critical moving forward. Increased knowledge of radiosensitivity and side effects gives rise to the possibility of more detailed dose plans.

1.1 Aim

To be able to do estimate delivered radiation dose to a variety of structures, accurate and efficient segmentation is required. This thesis aims to investigate whether a machine learning (ML) algorithm can be used to perform auto segmentation to reduce the time investment in segmentation work of brain structures of different complexities. The goal is to train several models using different imaging modalities, comparing the impact of modality type for different structures. Further, the thesis looks to investigate whether a model can be trained to perform segmentation on several brain structures without tuning the network precisely for that structure and without the thorough preprocessing or extensive algorithms found in comparable studies. The overall goal is thereby working toward accurate automatic segmentation methods in the context of small, large, or complex brain structures.

2 Background

2.1 Radiation interactions with matter

To explain how the mechanisms behind cancer treatment with radiation therapy work, one must first understand the mechanism behind the energy deposition of ionizing radiation. This section will focus on how these mechanisms work for photons and heavier particles. These mechanisms are also relevant for several imaging modalities, including X-rays and CT.

2.1.1 Photon interactions with matter

Photons interact with matter in many ways, but three kinds represent the majority of interactions relevant to diagnostics and therapy: Compton scattering, Photoelectric effect, and pair production [11]. The probability for a specific reaction to occur depends on the energy levels of the photons and the material properties of the matter it interacts with. The photoelectric effect will dominate lower energies, Compton scattering the intermediate range, and pair production at higher energies.

A photon beam traversing matter will lose intensity. The intensity at a given point depends on its initial intensity and the atomic number Z of the matter it traverses. The reduction in intensity due to the different interactions can be mathematically described as

$$I\,(x) = I_0 e^{-\mu x} \tag{2.1}$$

where I is the intensity after interaction with matter of thickness x, I_0 is the initial intensity of the beam, and μ is the linear attenuation coefficient. The linear attenuation coefficient depends on the atomic number Z, which is indicative of the electron density of the material and the incident photon energy. μ is a combination of the contributions from Compton scattering, the Photoelectric effect, and pair production. It is given as

$$\mu(E) = \mu(E)_{photoelectric} + \mu(E)_{Compton} + \mu(E)_{Pair\ production} \quad (2.2)$$

The contributions from the different interactions to the linear attenuation coefficient and the photon beam energy are displayed in Figure 2.1 [11].

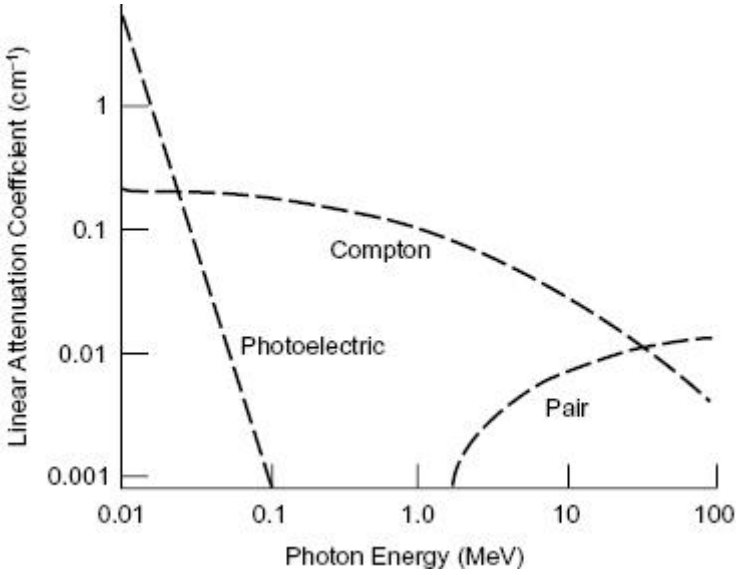

Figure 2.1: Probability of interactions depending on photon energy in MeV. Figure retrieved from [11].

Photoelectric effect

The photoelectric effect is when a photon collides with an atom and ejects an electron out from one of the inner shells while the photon is completely absorbed. The ejected electron is called a photoelectron. Figure 2.2 gives an illustration of the interaction [11].

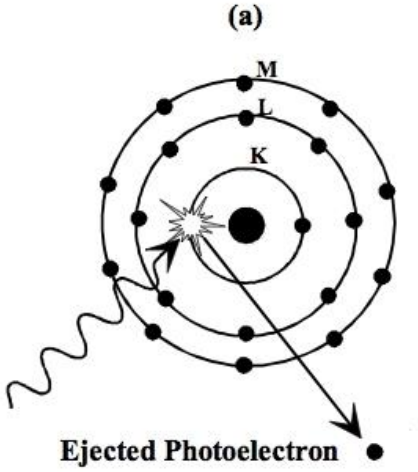

Figure 2.2: Illustration of the photoelectric effect. A photon hits an electron in one of the inner shells, resulting in the photon being absorbed, and the electron is ejected. Figure adapted from [12].

For the reaction to occur, the electron's binding energy needs to be lower than the energy of the incident photon. The energy of the photoelectron will be the difference between the energy of the photon and the binding energy of the electron [11]. The energy after an interaction is given as

$$\mu E = hv - \varphi \tag{2.3}$$

where E is the energy of the electron, h is Planck's constant $6.63 \cdot 10^{-34}\,\frac{m^2 kg}{s}$, v is the photon's frequency, and φ is the binding energy of the electron. The probability(P_{PE}) for this type of interaction to occur is dependent upon the energy of the incident photon (E), the effective atomic number of the tissue (Z_{eff}), and the mass density of the tissue (ρ). The effective atomic number is an approximation of the atomic number for molecular structures consisting of more than one type of element. The relationship between these is given in equation (2.4) [11], [13].

$$P_{PE} \propto \rho \frac{Z_{eff}^3}{E^3} \tag{2.4}$$

As seen in Figure 2.1 and equation(2.4), this interaction's probability is highest at lower energies. The probability increases with a higher atomic number, Z.

Compton scattering

Compton scattering happens when a photon interacts with an electron in one of the outer shells of an atom. The interaction will cause a fraction of the photon's energy to be transferred to the electron. The electron is ejected, and the photon is deflected while losing some of its energy. The energy of the scattered photon is given by

$$E_{\gamma'} = \frac{E_\gamma}{\left[1 + \left(\frac{E_\gamma}{m_e c^2}\right)(1 - cos\theta)\right]} \tag{2.5}$$

where $E_{\gamma'}$ is the energy of the scattered photon, $E_\gamma$is the energy of the incident photon, m_e is the electron mass, θ is the angle between the incident photon and the scattered photon. From Figure 2.1, one can see that Compton scattering is dominant at intermediate energies [13]. An illustration of the interaction can be seen in Figure 2.3 [11] - [12].

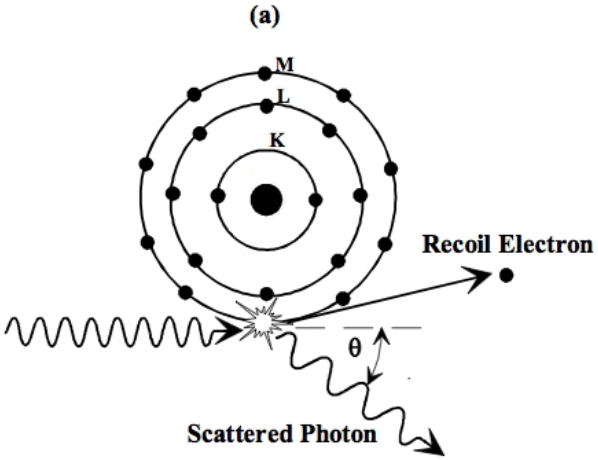

Figure 2.3: Illustration of Compton scattering. A photon hits an electron in one of the outer shells, resulting in the electron being ejected and the photon being deflected. Figure adapted from [14].

Pair Production

Unlike Compton scattering and the Photoelectric effect, Pair production happens when the incident photon interacts with an atom's nucleus. When a photon with sufficient energy

collides with a nucleus, an electron-positron pair is created, hence the name pair production. The mass of an electron and positron is equivalent to 0.511 MeV, which means the incident photon must have an energy higher than 1.022 MeV for the interaction to occur. Pair production is dominant at higher energies [11]. An illustration of pair production is shown in Figure 2.4.

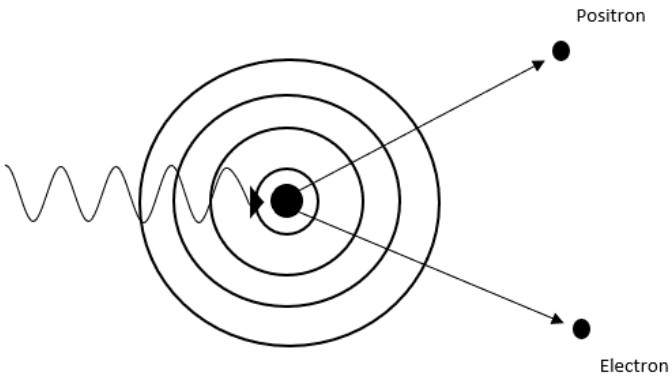

Figure 2.4: Illustration of Pair Production. A photon with energy greater than 1.02 MeV hits the nucleus, creating an electron-positron pair.

2.1.2 Charged particle interaction with matter

The mechanisms behind the interaction of charged particles with matter are significantly different from the mechanisms of photon interactions. Many different interactions can occur between charged particles and matter, but three of them are relevant to radiation therapy. Those three are inelastic Coulomb interactions between incident particles and atomic electrons, elastic Coulomb interactions between incident particles and nuclei, and inelastic Coulomb interactions between particles and nuclei. Since this thesis deals with proton therapy, proton interactions will be used to exemplify some interaction mechanisms.

When a charged particle traverses matter, it will lose small parts of its energy through a series of interactions. The amount of energy lost per unit length from Coulomb interactions between a charged particle and atomic electrons is described mathematically by the Bethe-Bloch formula, given as

$$-\frac{dE}{dx} = 2\pi N_a r_e^2 m_e c^2 \rho \frac{Z}{A}\frac{z^2}{\beta^2}\left[\ln\left(\frac{2m_e\gamma^2 v^2 W_{\max}}{I^2}\right) - 2\beta^2 - \delta - 2\frac{C}{Z}\right] \quad (2.6)$$

Table 2-1: Values and names of constants in the Bethe-Bloch equation. Adapted from [15].

Variable	Definition	Unit and value
N_A	Avogadro's number	$6.022 \cdot 10^{23}\ mol^{-1}$
r_e	Radius of electron	$2.817\ \cdot 10^{-13}\ cm$
$m_e c^2$	Rest energy of the electron	$0.511\ MeV$
ρ	Density of the absorbing material	
Z	Atomic number of the absorbing material	
A	Atomic weight of the absorbing material	
z	Charge of the incident particle	e
β	Velocity of the incident particle	v/c
γ	Lorentz factor	$\frac{1}{\sqrt{1-\beta^2}}$
W_{max}	Maximum transfer of energy	
I^2	Mean excitation potential	
δ	Density correction	
C	Shell correction	

Equation (2.6) describes the stopping power of a particle, and Table 2-1 displays what the different variables represent. The energy loss depends on the atomic number of the absorbing material (Z) divided by the atom's atomic weight (A), indicating a higher loss of energy for higher electron density. It is also dependent on the velocity of the incident particle (β), with energy loss increasing as the particle slows down. δ and C are correction factors. δ accounts for the density effect that arises when electric fields of the particle polarize atoms along its path. The polarization causes these electrons to be shielded from the full field intensity. They

will then contribute less to the total energy loss than what the Bethe-Bloch formula predicts. This is more relevant at higher particle energies. C is a correction factor added to account for effects at velocities comparable to the velocity of orbital electrons [15].

Inelastic Coulomb scattering is the main reason for energy loss when a heavy charged particle interacts with matter. A charged particle is considered heavy when it is heavier that an electron [15]. In inelastic Coulomb interactions, energy is transferred from the particle to an electron, causing either excitation or ionization, depending on the amount of energy transferred. If a reaction causes excitation, it is classified as a *soft* interaction due to the small amount of energy transferred, while interactions causing ionization are considered *hard* interactions. If a hard interaction occurs, the ejected electron might undergo secondary ionization by mechanisms described in section 2.1.1 [13] - [14].

Elastic Coulomb scattering accounts for minimal energy loss but is a more significant cause for increased beam width. Elastic Coulomb scattering is an interaction between the proton and the nucleus due to the positive charge of both participants. This interaction does not result in the loss of energy but will cause a slight deflection which over time causes lateral beam widening [13] - [14].

Inelastic Coulomb interactions happen when a proton collides with an atomic nucleus. This interaction causes the proton to knock out secondary particles from the nucleus, such as protons, neutrons, or other ion combinations. Most of these particles have lower energies than the incident proton, and these will be deposited locally. The higher energy particles that arise from this interaction will contribute to dose in larger areas of the tissue.

Since the energy deposition of the charged particle is inversely proportional to the velocity, it will lose most of its energy towards the end of its trajectory. The point where the energy deposition is at its highest is called the Bragg peak. This feature makes it possible to use

charged particles for precise energy deposition at exact locations in radiation therapy. Figure 2.5 gives an illustration of the Bragg peak.

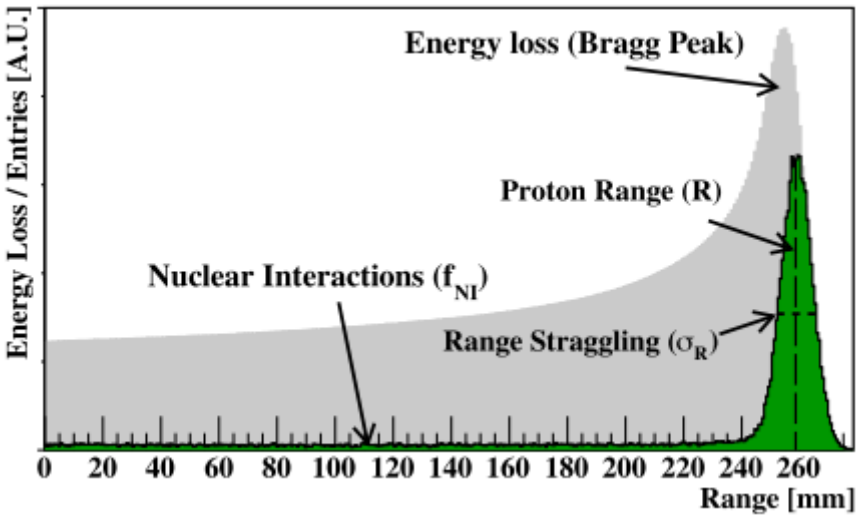

Figure 2.5: Figure illustrating the Bragg peak. The figure also displays the relationship between a proton beam's energy loss (grey) and the proton's final stopping position (green). Figure retrieved from [17] with permission.

2.2 Radiation therapy

Radiation therapy is the use of ionizing radiation to treat cancer. To understand how and why radiation is used to treat cancer, it is essential to understand how radiation affects biological matter and how to measure and predict its effects. While radiation therapy is not the main focus of this thesis, it is important to understand how it works and where medical images and segmentation fit in. This section will briefly introduce how to measure radiation exposure, different types of radiation therapy, the process before treatment, and the biological effects of radiation.

2.2.1 Dosimetry

Ionizing radiation is known to damage healthy tissue. The dose of Ionizing radiation can be defined as the absorbed energy divided by the mass, given as

$$D = \frac{\Delta E}{\Delta m} \tag{2.7}$$

Where D is the absorbed dose, ΔE is the mean energy level of the radiation imparted to mass Δm. The SI unit for absorbed dose is Gray, which is given as $1\ Gy = \frac{1J}{kg}$ [18].

While absorbed dose measures the energy deposited per unit mass, it does not consider the effect on biological matter. Therefore, a measure of equivalent dose can better indicate the dosimetric quantity. This method account for the specific type of radiation and the type of tissue that is exposed to radiation and can mathematically be described as:

$$H_T = \sum W_R D_{T,R} \tag{2.8}$$

Here H_T is the effective dose equivalent, W_R is the weighting factor for the type of radiation, with examples in Table 2-2, and $D_{T,R}$ is the absorbed dose averaged over tissue T, with radiation type R, expressed in gray. [19] The unit for dose equivalent is given in Sievert, where $1\ Sv = \frac{1J}{kg}$ [18]–[20].

Table 2-2: Table showing different Weighting factors. Adapted from [20].

Radiation type	Weighting factor W_R
Photon	1
Neutron	2-22
Protons and charged particles	2
α, fission fragment, and heavy ions	20
Electrons and muons	1

2.2.2 Biological effects

It is widely documented that radiation can cause tissue damage [21]. There are primarily two mechanisms that lead to cellular death from radiation exposure. The most considerable

documented direct effect of radiation is a double-strand break of the nucleus DNA threads. This break leads to the loss of reproductive ability and eventually cell death. The second mechanism is an indirect effect where the radiation produces free radicals that chemically interact with other parts of the cell, leading to apoptosis. Several models try to quantify the effect of radiation exposure to cellular structures.

Linear quadratic model

The linear quadratic model has two parameters that each represent a type of strand break in cellular DNA. The model is mathematically expressed as

$$SF = e^{-(\alpha \cdot D + \beta \cdot D^2)} \tag{2.9}$$

Where SF is the fraction of surviving cells, α is the constant representing double-strand breaks, β is the constant representing single-strand breaks and D is the dose. The ratio of α/β is where both components cause the same amount of cellular death. It is used to describe the sensitivity of tissues to fractionation of radiation doses. Most tumors have a high α/β ratio, as does early-responding tissue. Late-responding tissue has a lower α/β ratio, meaning they are more resistant to lower doses of radiation [22].

LET

Linear energy transfer (LET) is a measure for the quality of the ionizing radiation and is especially important in charged particle therapy. LET describes the energy deposition rate per unit length along the particle track.

RBE

Relative biological effectiveness (RBE) is a ratio to describe the necessary dose to get a specified biological effect for any type of radiation relative to a standard dose. The ratio is measured for a set dose of 250-kVp x-rays or ^{60}Co and any other dose and type of radiation [18]. RBE is closely related to LET. Higher LET values mean higher RBE values up to a certain threshold. Charged particles generally have an RBE greater than or equal to 1. While the LET

for a proton increases as the energy of the proton decreases, a constant RBE value of 1.1 is used for all energies, does levels, and tissues [18].

2.2.3 Proton and photon therapy

Photon beam therapy uses high energetic photons (4-18 MV) to kill cancer cells. Photons were first used to treat cancer in 1986 [23]. Photon beam therapy uses high-energy photons to kill cancer cells. The method is widely used, and while the method is the gold standard for external radiation therapy, new ways to optimize dose to target while limiting dose to healthy tissue are continuously researched.

Proton therapy uses a proton beam rather than photons to treat cancer. Protons therapy was first introduced in 1946 when Robert Wilson published an article theorizing the concept [24], and the first patient received treatment in 1954. Since then, the use of proton therapy has increased, but still, only about 1% of people receiving radiation therapy are treated with protons or heavier particles [25]. As more treatment facilities are being built, this number is expected to increase in the future.

While photons continuously lose energy as they travel through the body, protons have a finite range and an increasing energy deposition as the velocity decreases, as described in section 2.1.2, and can be seen from Figure 2.6. This relationship means that protons can deliver precise dosage to tumor volume while sparing healthy tissue for excessive radiation exposure. To ensure that the same dose is delivered to the entire tumor volume, techniques have been developed to combine different beams which achieves this [25].

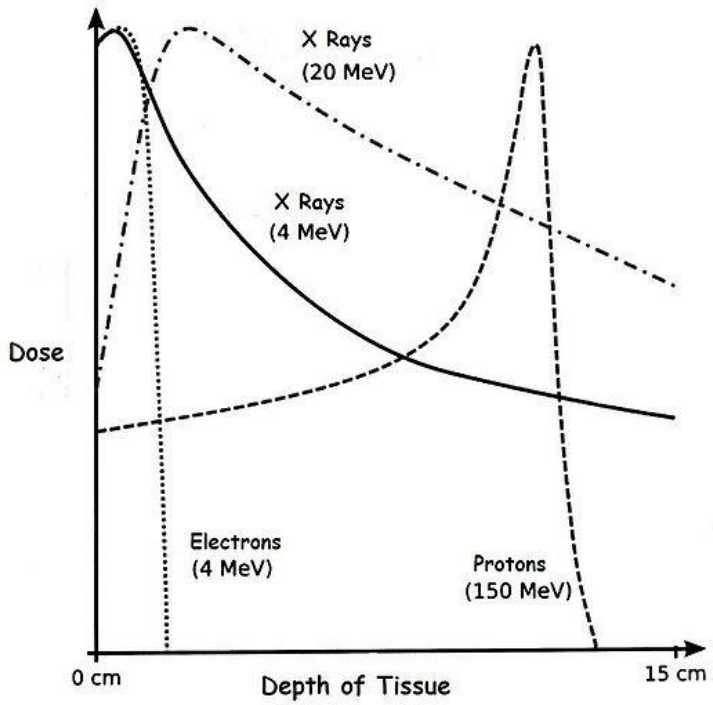

Figure 2.6: Figure illustrating energy deposition of different types of radiation. The figure shows that protons enter the tissue at lower energy than photons and then deposits higher energy later on in tissue relative to entrance energy. Figure retrieved from [26].

2.2.4 Dose specifications and image segmentation

When administering radiotherapy, one must always consider the potential damage done to surrounding areas of the tumor volume. Organs surrounding the tumor are often referred to as organs at risk (OAR). To ensure that these organs do not receive critical damage from radiation during treatment, all dose plans calculate the dose to the OARs. Medical images are vital in ensuring the best and safest treatment plan for patients receiving radiation therapy. Any patient that undergoes radiotherapy will usually have several sets of scans done. The treatment planning is usually done on a CT scan. Segmentation of anatomical structures is done by experts slice by slice, and the work is time-consuming [8]. Several methods for auto segmentation can be used to reduce the time investment in segmentation work.

Atlas segmentation

Atlas segmentation uses a reference model or image(atlas) to perform the segmentation. The method applies a voxel-wise correspondence between the atlas and target image and then applies a label fusion to reduce errors [27]. This method is also very time-consuming but might still be faster than manual segmentation. Atlas models may struggle when the presence of tumors or other foreign objects alters the anatomy from its natural shape. Atlas segmentation has, for example, been used in hippocampus and ventricle segmentation[28] and in planning for radiation therapy of endometrial and cervical cancer [29].

Machine learning

ML segmentation trains a model on known data and uses the model created to perform segmentation. This method aims to train a model on known images and structures to later perform the segmentation on the same type of images. The model will learn from any data it can find, like individual pixel values, groups of pixel values, the positions of a structure relative to the image, and more. ML for auto segmentation has been a rising method in the past decade due to increased computational power and more advanced algorithms. [30]

2.3 Computed Tomography

Computed Tomography (CT) is a widely used diagnostic tool that uses multi-angle x-rays and advanced reconstruction algorithms to create complete images of an object. CT utilizes the property of radiation attenuation and the different attenuation profiles of tissue to create the images. Figure 2.7 shows an example slice from a head CT scan.

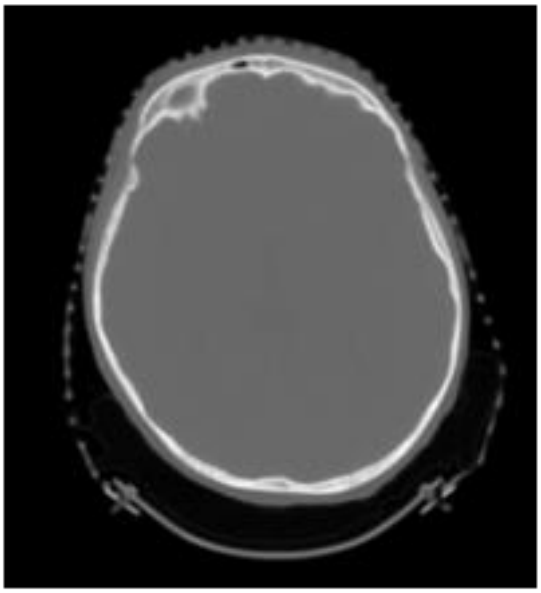

Figure 2.7: Example of a CT image of the cranium.

2.3.1 X-rays and the X-ray tube

X-rays are a form of high-energy electromagnetic radiation, with energy levels between 145 eV to 140 keV. X-rays are generated using the x-ray tube. The X-ray tube consists of a rotating anode, a cathode, an evacuated vessel surrounding the construction, a target usually made of tungsten, and a high voltage source [13]. X-rays are produced by heating a filament, usually made of tungsten, near the cathode, by passing a current through it as shown in Figure 2.8. When the filament reaches ~2200 ℃, the electrons have enough energy to leave the surface of the metal. A large positive voltage is applied to the target and turning it into the anode. The potential between the anode and cathode accelerates the electrons toward the tungsten target. The potential difference for a regular x-ray is usually between 25 and 140 kV [13]. Some of the kinetic energy is converted into X-rays upon hitting the target [12], [28].

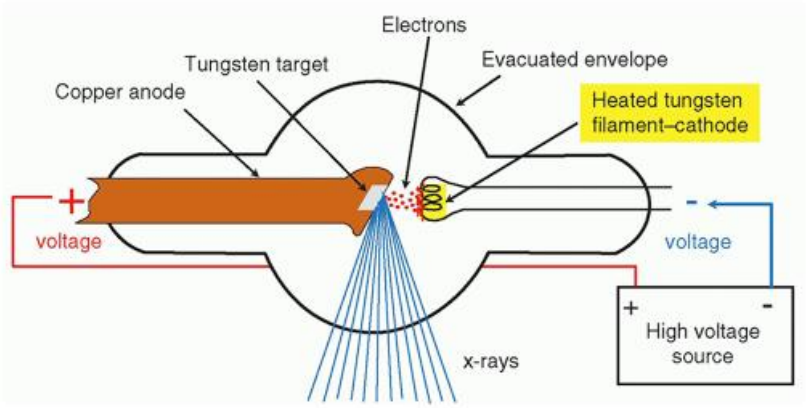

Figure 2.8: Figure illustrating an X-ray tube. Figure retrieved from [32].

The energy of the X-rays produced will vary but always have a maximum equal to the difference in potential created in the X-ray tube. When the electron passes close to the element's nucleus in the target, the electron will be deflected, causing a loss of kinetic energy. This energy is converted into an X-ray, and the mechanism is called bremsstrahlung. X-rays will also be created by the electron colliding with another bound electron in the target. The electron will then be ejected out of its shell, and an outer electron will fill the gap left by the one ejected. The rearranging of electrons results in an X-ray emitted with energy corresponding to the difference in binding energy between the ejected electron and its replacement [12], [28]. An example of X-ray spectrum from a tube is shown in Figure 2.9.

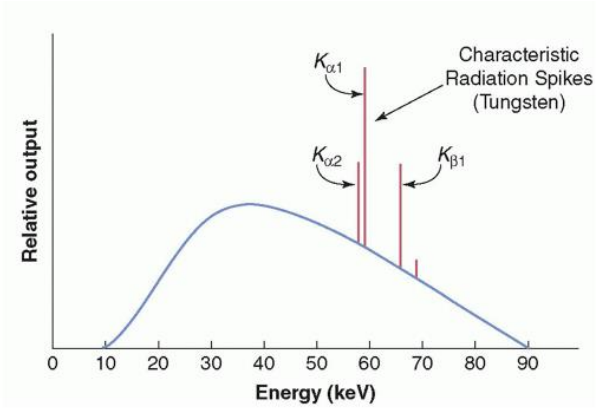

Figure 2.9: Spectrum of bremsstrahlung and characteristic X-rays from a tungsten target with a potential difference of 90 kV. Figure retrieved from [32]

2.3.2 Attenuation, interaction, and detection of X-rays

X-rays primarily interact with matter by two of the processes described in section 2.1.1, Compton Scattering and Photoelectric effect. Pair production is not relevant due to the energy level of regular X-ray imaging being below the threshold of 1.022 MeV required for pair production [31]. To get the best possible image, enough X-rays need to pass through the body while also having enough attenuation difference between tissues and bones [13]. The intensity of X-rays passing through an object can be expressed by equation (2.1). The effective atomic number of tissue is ~7.4, lipid ~6.9, and bone ~13.8. The relative tissue densities are 1, 0,9, and 1.85, respectively [13]. The high effective atomic number of bone makes photoelectric interactions more likely to occur at lower energies and, therefore, higher contrast between tissue and bone [13].

Compton scattering contributes more at higher energies, as can be seen from Figure 2.1. From equation (2.4), it is apparent that the energy of the incident X-ray is only slightly reduced, even at larger angles. This means that most of the X-rays will make it through all the tissue and be detected. At very large angles, the X-ray will be deflected beyond the range of the detector and, in effect, will be registered as attenuated [13]. While the attenuation from Compton interactions is important for contrast in the image, it also contributes extra noise. Since the X-ray is scattered at a different angle in this interaction, they will induce noise if they are detected after deflection at unexpected angles. This can be reduced by shielding the detectors from photons that arrive at off angles [13].

2.3.3 Computed Tomography principles

The essential components of a CT scanner are a rotating X-ray tube, and a detector mounted on a gantry. The patient is placed inside the gantry, and the X-ray tube rotates around the patient, acquiring data from all angles. This process measures the attenuation coefficient at precise body positions and then reconstructs an image based on these data points. This process is called filtered back-projection.

Figure 2.10 is a basic example of what happens when recording a CT scan, where the figure represents a volume. Radiation with intensity I_0 is sent out from the X-ray tube at different angles. These X-rays then pass through the volume with varying attenuation coefficients and are then recorded by the detector after exiting the volume, shown as I_1 through I_4 in Figure 2.10.

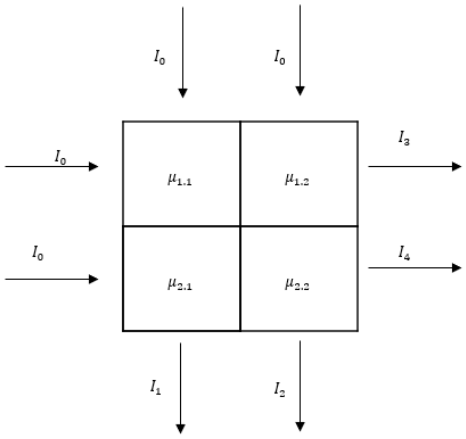

Figure 2.10: Basic example of how X-rays pass through objects and are attenuated.

The intensity measured by the detectors after passing through the volume will depend upon the different attenuation coefficients in the volume. Considering I_1, it passes through two different parts of the volume, with attenuation coefficients $\mu_{1.1}$ and $\mu_{1.2}$. The equation to find as I_1 can then be written like this:

$$I_1 = I_0 e^{-x(\mu_{1.1}+\mu_{1.2})} \qquad (2.10)$$

The intensities can then be recorded in a new image called a sinogram using this method. The sinogram shows all raw acquisition data as a function of the gantry angle in a 2d matrix [33]. Back-projection is then used to generate an image. In this process, a single row, i.e., a single attenuation profile obtained from the sinogram, is projected over a new image at the angle at which it was recorded. The attenuation profile is divided by the number of pixels it passes through and projects this average value upon the new image. This process is done for all the

different angles. The sum of all the back-projected attenuation profiles gives the full image [33].

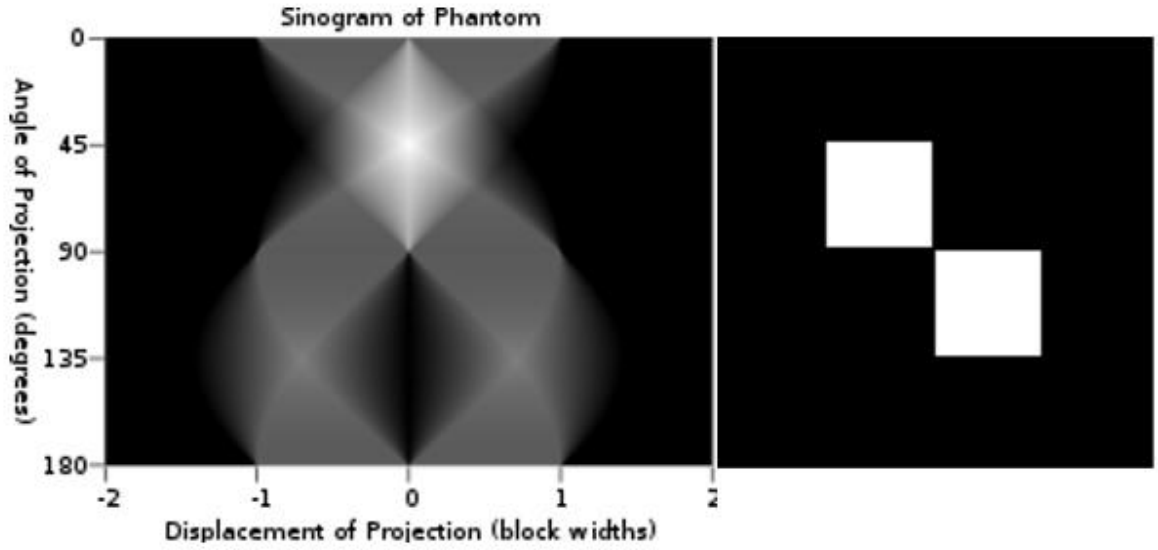

Figure 2.11:Figure of sinogram created from phantom displayed to the right of the image. Figure retrieved from [34].

The image generated by this method alone is blurry, and barely any information can be extracted from it. Therefore, a filter is added to the attenuation profiles before the back-projection. The applied filter enhances the edges in the image by suppressing the low spatial frequency of the attenuation profiles. This process results in a clear and readable image, where details of the anatomy can be visualized, such as shown in Figure 2.11 [33].

CT attenuation values are commonly transformed to Hounsfield units. Hounsfield units transform the values into a dimensionless scale relative to the attenuation value of water, with example values given in Table 2-3 [35].

$$HU = \frac{\mu - \mu_{water}}{\mu_{water}} \cdot 1000 \qquad (2.11)$$

Table 2-3: Different Hounsfield Units for different tissues. Table retrieved from [13]

Tissue	Hounsfield Units
Bone	1000-3000
Muscle	10-40
Water	0
Lipids	-50 to -100
Air	-1000
Brain (White matter)	20 to 30
Brain (Grey matter)	35 go 45
Blood	40

2.4 Magnetic Resonance Imaging

Magnetic resonance imaging is a diagnostic imaging tool that gives high contrast images using radio-frequency signals and the fundamental property of atomic particles, spin, to generate images. Two examples are given in Figure 2.12.

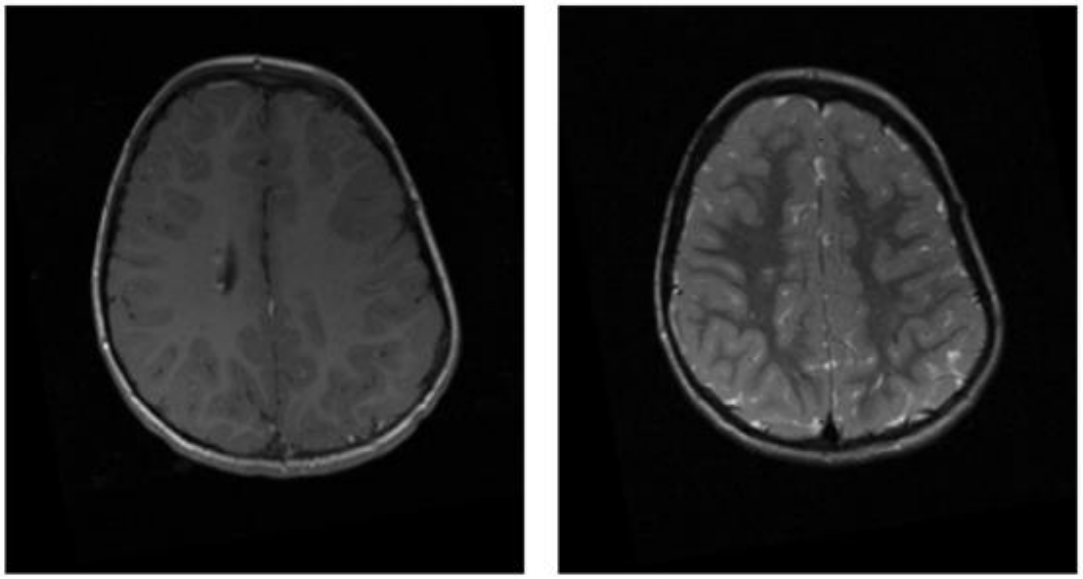

Figure 2.12: Example of a T1 weighted MRI image (left) and a T2 weighted MRI image (right).

2.4.1 Spin and magnetic momentum

Spin is an intrinsic property of an atom that depends upon the composition of nucleons in the nucleus. For a particle to have a net spin, there must be an uneven number of nucleons. The spin of both protons and neutrons is $s = \frac{1}{2}$. A spinning particle with an electric charge will have a magnetic momentum μ. The angular momentum is related to the magnetic momentum by a constant γ, the gyromagnetic ratio. The relationship is given as

$$|\mu| = \gamma|\vec{P}| \tag{2.12}$$

where $\vec{P}$ is the nuclear angular momentum. The gyromagnetic ratio is measured in MHz/T, and Hydrogen has the highest gyromagnetic ratio of all nuclei [36]. The high gyromagnetic ratio of hydrogen is relevant as Hydrogen is abundant in tissue. The body is primarily fat and water, both rich in hydrogen atoms. Because of this, Hydrogen is the most used atom for MRI scans, as it gives the strongest MR signal. Hydrogen and proton will be used interchangeably to describe the particles that provide the MRI signal.

The magnitude of the magnetic moment for the individual Hydrogen atoms is a fixed value, but the direction is random. This causes the net sum to be zero. This changes when a particle with magnetic momentum is placed in a static magnetic field. The particle will align itself with or against the direction of the field at an angle of 54.7°[13]. If the particle aligns in the direction of the field, it is referred to as parallel. If it is against, it is referred to as anti-parallel. Thus, when a large amount of Hydrogen atoms is placed in a static magnetic field B_0, they will either align parallel or anti-parallel the field. The Hydrogen atoms prefer to align parallel with the field, in the lower energy state rather than against the field, in a higher energy state [12], [33].

$$\vec{M} = \sum_{n=1}^{N_s} \mu_n \tag{2.13}$$

Where N_S is the total number of spins being observed and $\vec{M}$ is the net magnetization vector [36]. The magnetic moment for each of the two different energy states can be written as

$$\overrightarrow{\mu_z} = \pm\frac{1}{2}\gamma\hbar \tag{2.14}$$

when placed in a magnetic field along the z-direction [36]. The energy difference between the two possible energy states of particles placed in a magnetic field B_0 is given as $\Delta E = \Delta E_{\downarrow} - \Delta E_{\uparrow} = \gamma\hbar B_0$. Boltzmann statistics is used to calculate the number of protons placed in each state:

$$\frac{N_{anti-parallel}}{N_{parallel}} = \exp -\left[\frac{\Delta E}{kT}\right] = \exp -\left[\frac{\gamma\hbar B_0}{kT}\right] \tag{2.15}$$

Where ΔE is the difference in energy between the two states, B_0 is the strength of the magnetic field, k is the Boltzmann constant of $1.38 \cdot 10^{-23}\,\frac{J}{K}$ and T is the temperature measured in kelvin. Normally $\Delta E \ll KT$ so that the equation can be approximated as:

$$\frac{N_{anti-parallel}}{N_{parallel}} \approx 1 - \left[\frac{\gamma\hbar B_0}{kT}\right] \tag{2.16}$$

This approximation leads to the difference in Hydrogen atoms in parallel and anti-parallel as:

$$N_{parallel} - N_{anti-parallel} \approx N_{total}\left[\frac{\gamma\hbar B_0}{2kT}\right] \tag{2.17}$$

If this equation is solved, it will give about three more protons in the lowest energy level pr million for a 1T magnetic field in room temperature conditions [13, 35].

An expression for the net magnetization vector $\vec{M}$ can now be found. Say the direction of the magnetic field is in the z-plane. This means the components of μ_n for the x- and y-plane are random, and their contributions sum up to zero [36]. The net magnetization vector previously defined in Equation (2.13) can be written as

$$\vec{M} = M_z \cdot \vec{k} = \left(\sum_{n=1}^{N_s} \mu_{z,n} \right) \cdot \vec{k} \qquad (2.18)$$

Substituting equation (2.14) for μ_z gives

$$\vec{M} = M_z \cdot \vec{k} = \left(\sum_{n=1}^{N_\uparrow} \frac{1}{2}\gamma\hbar - \sum_{n=1}^{N_\downarrow} \frac{1}{2}\gamma\hbar \right) \cdot \vec{k} = \frac{1}{2}\left(N_{parallel} - N_{anti-parallel}\right)\gamma\hbar\vec{k} \qquad (2.19)$$

Replacing part of equation (2.19) with equation (2.17), the final function for the net magnetization vector can be derived to

$$\left|\vec{M}\right| = \frac{\gamma^2\hbar^2 N_{total} B_0}{4KT} \qquad (2.20)$$

Where γ is the gyromagnetic ratio, $\hbar$ is Planck's constant divided by 2π, N_{total} is the total number of protons, B_0 is the strength of the magnetic field, k is the Boltzmann constant, and T is the temperature in kelvin. [36] Equation (2.20) shows that the magnitude of the net magnetization is proportional to the strength of the magnetic field and inversely proportional to the temperature [13, 35].

2.4.2 Resonance and signal generation

When placing a proton in a magnetic field B_0 it will precess around B_0 at a frequency that is proportional to the strength of the magnetic field. This frequency is called the Larmor frequency ω_0 and is defined as

$$\omega_0 = \gamma B_0 \tag{2.21}$$

where γ is the gyromagnetic ratio for the proton, and B_0 is the field strength. To measure the net magnetization vector $\vec{M}$ energy needs to be added to the system. The release of this energy is used to detect MRI signals. Energy is sent into the system as an RF pulse with a frequency matching the proton's Larmor frequency. The RF pulse will create excitation in the protons, causing more protons to rise to the higher energy state. The excitation makes the net magnetization vector $\vec{M}$ deviate from its original angle. This deviation angle is called a flip angle, denoted α. This causes $\vec{M}$ to tip away from the z-axis and into the XY plane. If a conduction coil is placed in the system, Faraday's law of induction will induce a voltage in the coil. Since faradays law requires time-varying magnetic flux to induce a voltage, signals will only be detected when precessing in the XY-plane. The strength of the signal will depend upon the number of protons present and the strength of the magnetic field B_0. This signal is called the Free induction decay, or FID. While this signal is detected immediately after the RF-pulse is sent in, it is not used in image generation. This is due to the rapid response to generate the wanted information. Instead, an echo of the FID is used for image generation [13], [36].

2.4.3 Relaxation times

Immediately after the RF-pulse stops transmitting, the net magnetization will start to move back from the XY-plane to align with B_0. The process where the magnetization recovers along the z-axis and falls back to its resting state is referred to as T1 relaxation. T2 relaxation is the loss or decay of magnetization in the XY-plane after switching off the RF pulse. The recovery and loss process can be described mathematically as:

$$M_z(t) = M_z^0 \left(1 - e^{-\frac{t}{T1}}\right) \tag{2.22}$$

$$M_{xy} = M_z^0 \, e^{-\frac{t}{T2}} \tag{2.23}$$

where M_z and M_{xy} describes the net magnetization in that plane at time t, M_z^0 is the initial net magnetization, T1 is a constant that refers to the time it takes for M_z to regain 63% of its value, and the time it takes for there to only be 37% left of M_{xy} is referred to as the T2 constant[36]. The T1 and T2 constants are tissue-dependent, and T2 is always shorter than T1 [13], [36].

Table 2-4: Tissue relaxation times in ms for 1.5T and 3.0 T Main fields. Table retrieved from [13].

Tissue	T1 (1.5 T)	T1 (3.0 T)	T2 (1.5 T)	T2 (3.0 T)
Brain (white matter)	790	1100	90	60
Brain (grey matter)	920	1600	100	80
Liver	500	800	50	40
Skeletal Muscle	870	1420	60	30
Lipid (subcutaneous)	290	360	160	130
Cartilage	1060	1240	42	37

As can be seen from Table 2-4, T1 and T2 values for lipids differ significantly from other values with the same strength. This difference is because the resonance frequencies of protons in lipids are lower than, for example, water [36]. When a proton is placed in a magnetic field B_0 the strength of the magnetic field the protons experienced is also dependent upon the electron configuration in the area surrounding the proton. An electron has the same fundamental properties as a proton and thus has its own small magnetic field, as explained for protons in section 2.4.1. This magnetic field is opposite in polarity to the main field B_0, due to the electron's negative charge. All of this reduces the main magnetic field by a coefficient σ, and the field experienced from the proton can be described as

$$B_{eff} = B_0(1 - \sigma) \tag{2.24}$$

where σ is a shielding constant that is related to the electron configuration in the protons surrounding environment [13]. This effect is especially relevant in lipids compared to water, as oxygen is more electronegative than carbon and thus pulls the electrons more away than carbon in lipids -CH_2- structure [13].

2.4.4 Image contrast and pulse sequence

MRI images are often referred to as T1, T2, or Proton density weighted. This does not mean that the image only has T1 or T2 values but that the scan is acquired with emphasis on one contrast value. MRI uses four sequence parameters to manipulate image contrast and determine the weighting of the image. The first one has already been mentioned, and it is the flip angle α which is the angle the of the net magnetization vector and its resting position along B_0 right after the RF-pulse has been applied. The second is the Echo time, referred to as TE. This is the time delay between the RF pulse and signal peak or readout. The third is Repetition Time, which is the time between RF pulses. TE and TR are shown in Figure 2.13. The final is the Inversion time, TI, the time between two RF pulses with a different flip angle to ignore the signal from a specific tissue [13], [36].

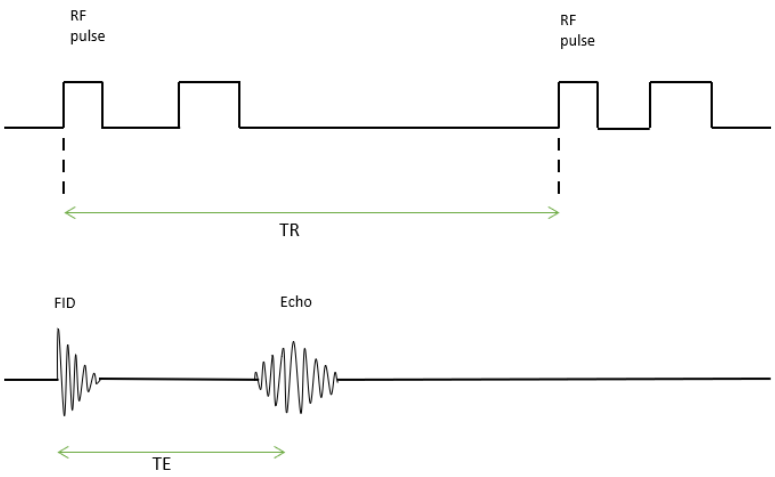

Figure 2.13: Figure illustrating TR (Repetition time) and TE (Echo Time).

In a T1 weighted sequence, the contrast between two tissues depends on the difference in T1 relaxation times between the given tissues. However, there might not be enough difference in T1 relaxation times between two tissues to detect contrast in the MRI image. Using contrast agents can help increase the difference in T1 relaxation time and thus increase the contrast between tissues. Using contrast is especially common when acquiring scans for CNS tumor diagnosis [37].

2.4.5 Image acquisition and signal localization

In addition to the main magnetic field B_0 there are three additional magnetic fields in an MRI machine, referred to as gradients. These fields are not constant but are applied throughout a scanning sequence to select slices and for phase and frequency encoding. The gradients are positioned so that it is possible to create fields in all three directions of G_x, G_y, G_z [36].

When using a gradient to create an additional magnetic field with varying strength in the z-direction, the protons will respond at different frequencies along the field, corresponding to the variations in the magnetic field due to the Larmor Equation. When applying an RF with a specific Larmor frequency, only the protons with the exact same Larmor frequency will respond. This makes it possible to select only a tiny slice of the object. Phase and frequency encoding is used to locate the signal. By applying a gradient in the x-direction, the net magnetic field is altered along the axis. Then the signal can be identified due to the frequency differences and the known quantity of the applied magnetic field. A phase shift is also induced by using the gradient in the y-direction. The total of the phase shift and frequency difference means that for each voxel, there is a corresponding frequency and phase to locate the signal [13], [36]. An example of a pulse sequence is shown in Figure 2.14.

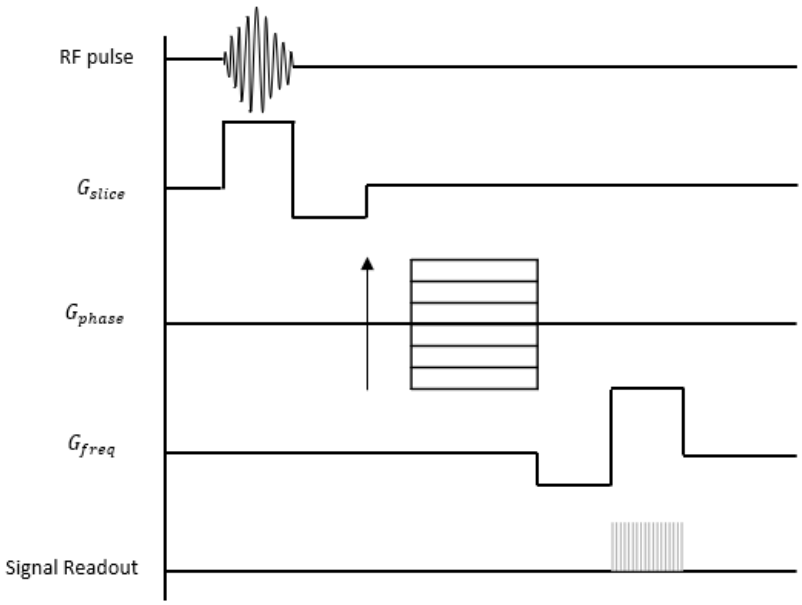

Figure 2.14: Figure illustrating a pulse sequence in MRI. In this illustration, the steps are sequential. In a real imaging sequence, many of the gradients will be applied simultaneously. The figure is inspired by [13].

2.5 Machine learning and deep learning

Artificial intelligence is any algorithm or technique that tries to mimic human intelligence. ML refers to computer algorithms that can improve automatically by only providing them a set of data and is a subdivision of artificial intelligence [38]. Deep learning is a branch of ML and is an algorithm that tries to mimic the human brain with neural networks of multiple layers. This section will introduce some of the basic concepts in ML and then focus on deep learning, as that is what is used in this thesis.

2.5.1 Machine learning fundamentals

Machine learning aims to get algorithms to understand and complete a task without having to code every aspect of the task explicitly. Instead, specific algorithms are applied to learn

from a dataset and become able to complete similar tasks in the future by analyzing this dataset [39].

Creating a ML model is usually divided into three steps: Training, validating, and testing. When working with a dataset to create a ML model, the dataset is divided into these three parts, used for the respective steps in creating the model. Training is done by feeding a larger part of a dataset into the algorithm. The model tries to extract as much information from the dataset as possible and make connections between the data points and the corresponding value, label, or other output. Validating is done by using a small part of the dataset to check the model's performance during training. Validation is also used to tune the model by adjusting different so-called hyperparameters. By adjusting hyperparameters, the model's performance may increase, and this is done to find the optimal model. After the model is complete, the testing part of the dataset is used to check how well the model will perform on unseen data [40].

ML has three traditional ways of working with data and which algorithms to use. The first one is called supervised learning. The dataset used to create a model with a supervised learning algorithm will have data points and a corresponding label, image, or similar. Having data with corresponding labels means that the algorithm can train on the different data points while knowing what it is supposed to find. The second is unsupervised learning, where the data does not have corresponding labels. With this method, the algorithm is trying to sort the data into categories by the data points available. The final one is reinforcement learning. When creating a model using a reinforcement learning algorithm, the algorithm itself explores the environment and learns from a trial-and-error approach. The algorithm makes a series of decisions and decides what to do next based on the reward of these decisions. The function tries to optimize itself as to maximize the reward [41].

A ML model also has two types of parameters that need to be considered. One is referred to only as parameters, which is the settings an algorithm learns while training. It is internal to

the model and cannot be changed by the user. The other is hyperparameters, and these are settings for the algorithm the user sets before the training starts. Therefore, it is essential to find out how to set the hyperparameters correctly to get the best possible model. Hyperparameter-tuning can be done through trial and error or using preprogrammed algorithms [42].

Since ML models are trained on data and often also corresponding labels, they can sometimes be fitted to perform exceptionally well on the training data and not so much on unseen data. This is called overfitting. Overfitting can happen if the model overfocuses on learning this specific dataset and fails to generalize the problem. Overfitting can happen by using a too complex model for the problem. Another way it happens is by incorrect hyperparameter-tuning. The validation and testing part of the model construction can help avoid overfitting [40]. Examples of ML algorithms are Decision Trees, Linear regression, and RandomForrest.

Bias and variance are important to understand when investigating a model's ability to generalize. Bias is a measurement of the difference between the prediction of the model and the ground truth label. Variance measures the impact of minor disturbances in the dataset and its impact on performance. High bias can cause the model to miss meaningful relationships between features, causing underfitting. High variance can cause the model to use noise as a feature, making the model overfit. Finding the best possible value for both bias and variance is often called the bias-variance tradeoff, minimizing both measurements as much as possible [43].

2.5.2 Deep learning

A subdivision of ML is deep learning. Deep learning algorithms seek to emulate the human brain. Deep learning differs from other parts of ML in that the algorithm itself finds out which features are essential and how to use these for optimal labeling. In traditional machine learning, one would define these beforehand and explicitly tell the algorithm which features

to use. In deep learning, the algorithm finds these features. Deep learning algorithms are often referred to as neural networks, and the most basic one is an artificial neural network [43 - 44].

The architecture of a neural network

Artificial neural networks use layers and activation functions to generate a prediction from the given input. The number of layers and activation functions can vary based on the task and input data. There are three main types of layers. The input layer receives the data given to the network for training. The hidden layer is where the training and feature extraction happens. The third layer is the output layer, which gives the model's prediction. A basic example of a neural network can be found in Figure 2.15 A.

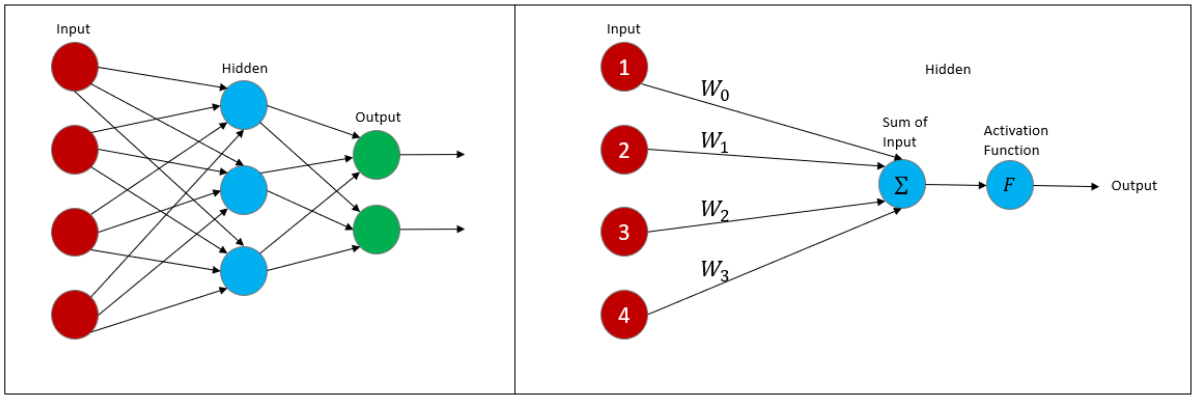

Figure 2.15: A) Basic Artificial Neural Network Architecture, B) Function of a single neuron, here as an example of a hidden neuron from A.

One node in a layer is called a neuron. A neuron performs a mathematical operation on a given input, often called the activation function. In Figure 2.15 B, the process in a single neuron is illustrated. The neuron receives four inputs that are each multiplied by a corresponding weight W_i. These four inputs are then added together. The neuron then applies an activation function to this value and gives the result of the activation function as output. The weights between two neurons will determine how much influence the input will have in the coming computational process of the activation function. A neuron might also have a bias value added

in addition to input from other neurons to ensure correct activation. A network where all inputs are connected to all outputs is called a fully connected or dense layer [41].

Activation functions are used in neural networks to make them non-linear. If no function is applied, the network will behave like a large linear regression model. While linear regression models can be useful for many tasks, they lack the complexity needed for others. Therefore, activation functions are added to neural networks [46]. There are different kinds of activation functions. Some common ones are ReLU, Sigmoid, and Tanh. ReLU takes the given input and returns 0 if the value is below 0, otherwise returns the value of the input $f(x) = \max(0, x)$. Sigmoid transforms the values given into the range of (0,1] by the equation $f(x) = \sigma(x) = \frac{1}{1+ e^{-x}}$. Tanh takes any given number and reduces it to somewhere in the range [-1,1] by the equation $f(x) = Tanh(x) = 2\sigma(2x) - 1$ [45 - 46]. The activation function provides non-linearity to the neuron's output by transforming it.

Training of a neural network

After the network receives input data and the data has passed through the network, the output layer gives the prediction for the data. A neural network is usually trained in several epochs, where one epoch is one pass over all the training data. The weights between neurons are arbitrarily assigned during the first epoch, as it has no information to go on. The network measures the predicted output against the actual output when the first pass is complete. This measurement is done by a function referred to as the loss function. The loss function quantifies the difference between the predicted output and the true value. To best measure predicted against actual values, the loss function will usually not make a point-by-point comparison but calculate loss over several data points at a time. A cluster of several data points is commonly referred to as a batch. By checking the difference between predicted and actual values, the network then updates the weights between neurons.

The weights of a neural network are updated based on an optimization function. This function can be various mathematical models, where the most common is gradient descent. Regular gradient descent tries to optimize the weights of the function based on information received

from the loss function. The model seeks to update the neurons' weights so that the loss, or value, between the predicted output and actual value is at a minimum. Gradient descent, visualized in Figure 2.16, calculates the gradient for a given point on the loss function and gives the network feedback to update the weights in the opposite direction. This process is done for every sample in regular gradient descent. While this is extremely accurate, it takes a very long time. Instead, this process is done in smaller batches, similar to how the loss function is calculated. The weights are then updated with a small increment of that value, based on the learning rate set for the network. The learning rate decides how much the weights will be updated, as the value for the learning rate is multiplied by the difference. This process is done for each epoch to get as close to the ground truth as possible.

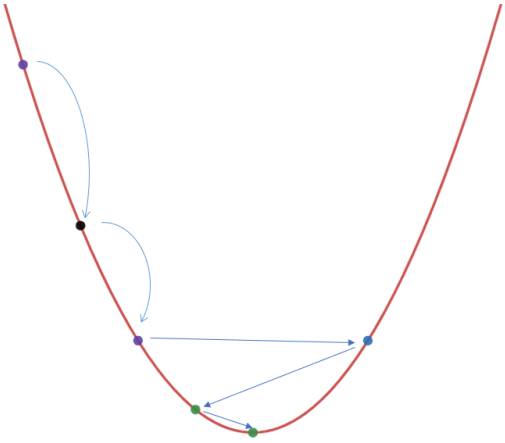

Figure 2.16: Figure illustrating Stochastic Gradient descent in a one-dimensional space. In reality the space has dimensions equal to the number of parameters, but that is not visualizable.

Adam is a common optimization function for computer vision that builds on the stochastic gradient descent principle. It only requires first-order gradients, which use little memory. It computes individual adaptive learning rates for the parameters based on the gradients' first and second-order momentum [48].

Performance measures and metrics

When a model is fully trained, the performance and generalization of the model are measured. Performance measurement can be done using several performance metrics. The most

common ones are error rate and accuracy, where the error rate is the number of falsely classified examples divided by the total number, and accuracy is the number of correctly classified samples divided by the total number. However, these measures are not suitable for all tasks. There are four possible scenarios in a binary classification problem: True positive, true negative, false positive, and false negative, visualized in Figure 2.17. From these four, there are several metrics derived. Recall is defined as $Recall = \frac{TP}{TP+FN}$, and will indicate how often a model mislabels a TP value as false. The opposite is Precision, defined as $Precision = \frac{TP}{TP+FP}$, indicating how often the model labels a negative value as positive [43].

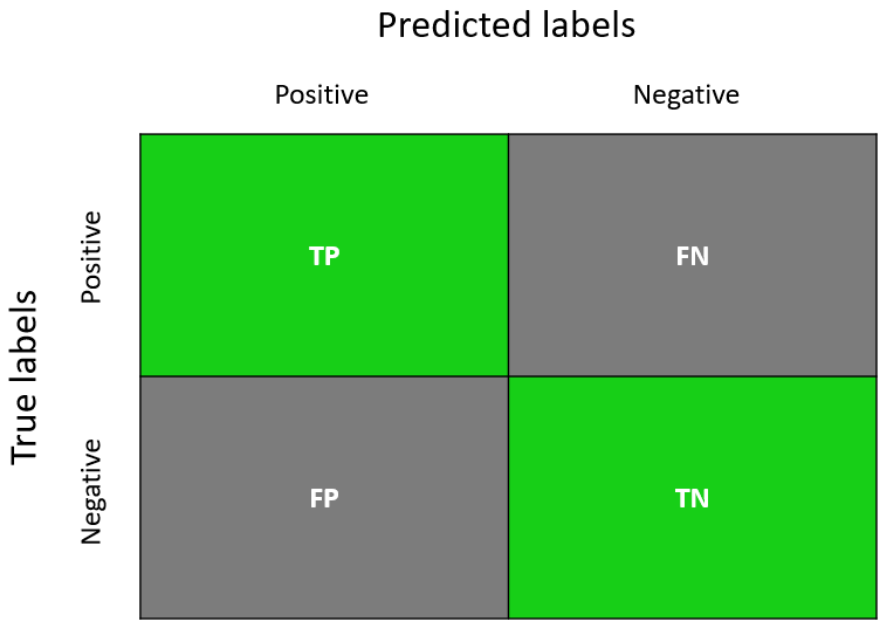

Figure 2.17: Figure illustrating possible scenarios in a binary classification problem.

Another common performance metric is the Dice coefficient. The Dice coefficient is two times the area of overlap between two images divided by the total number of pixels in both images. Since the coefficient is calculating overlap between predicted and true masks, it works well on segmentation tasks. Figure 2.18 gives an illustration of how the Dice coefficient is calculated.

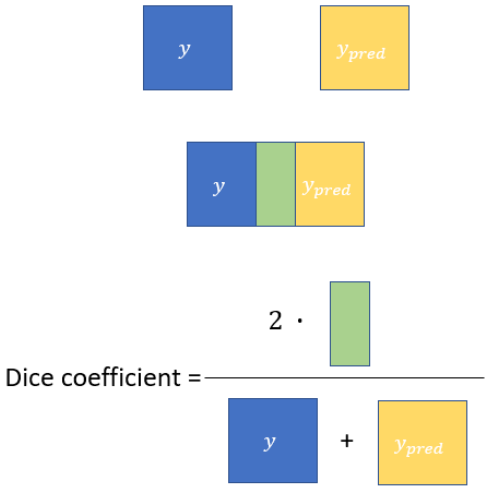

Figure 2.18: Figure illustrating how the Dice coefficient is calculated

2.5.3 Convolutional Neural Networks

Convolutional neural networks are a sub-division of artificial neural networks. A few examples of the many uses for CNNs are image classification [49] and natural language processing [50]. Examples from the medical field, CNNs have been used to identify metastatic breast cancer[51], classify Alzheimer's disease from fMRI images [52], and classify brain tumors [53]. The CNNs structure is optimal for classifying images. A CNN consists of three types of hidden layers: Convolutional layers, pooling layers, and fully connected layers.

The convolutional layer works by applying a filter, often referred to as a kernel, to an image to extract essential features using convolution. The convolutional process is displayed in Figure 2.19: a 3x3 filter is applied to a 5x5 image. From A, the filter starts in the upper right corner. Every pixel value from the image is multiplied by the kernel value and then summarized to get the convolved value for that area. The kernel continues to move over the image, performing the same mathematical operations and filling in a new 2D representation of the image. This process is usually done multiple times for each layer to extract not only the prominent features, such as edges and contrasts in colors but also the more subtle features.

In a CNN, the network learns what values to use in the kernel as part of the training. This process extracts information while also reducing the complexity of the model [45].

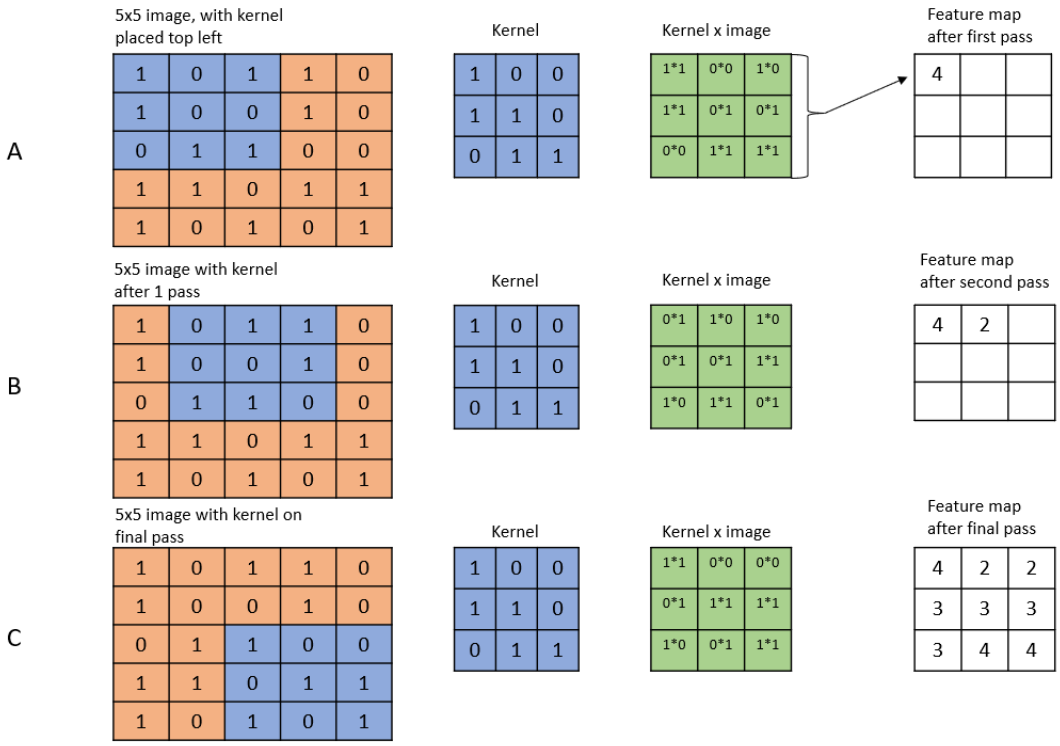

Figure 2.19: Convolutional neural network. Figure illustrates how the filter works in a convolutional neural network to extract features.

When an image undergoes the process in a convolutional layer, the corners of the image are only being passed over once for every kernel. After this, the image is downsized, as seen in Figure 2.19. This can lead to a lower weighting of the information in corners. To compensate for the potential information loss, padding can be added to the image. Padding is adding extra pixel values, usually with a value of zero, around the image's borders to make the image appear larger to the kernel than it is, illustrated in Figure 2.20. The kernel will then have extra rows and columns to pass over when processing the image, giving back a convolved feature with the same shape as the original image [43] - [44].

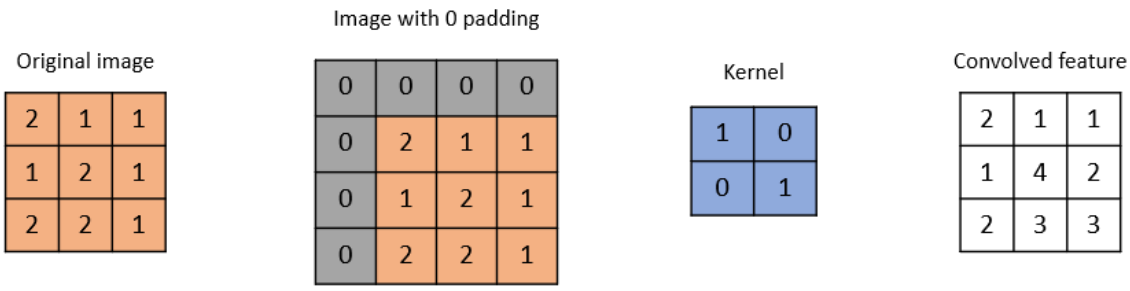

Figure 2.20. Example of zero padding

Another layer type in a CNN is the pooling layer. Spatial pooling is like the convolutional layer in that a kernel is passed over the image. However, the mathematical operation performed is different. There are two types of pooling commonly used, max pooling and average pooling. Figure 2.21 gives an example of how this would work. Here a 2x2 kernel is passed over a 4x4 image with stride set to two. Stride is the distance the kernel moves between operations. Max-pooling returns the maximum value in the kernel, while average pooling returns the average of all values in the filter. This process helps reduce noise in the image while retaining essential features. This process also reduces the computational power needed, since the image is now smaller. The pooling layer is commonly added after a convolutional layer, downsizing the image even more while retaining important information from the image [43] - [44].

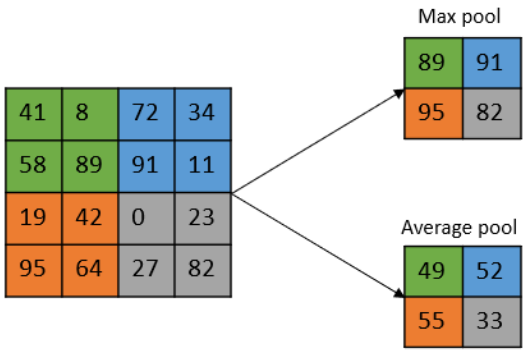

Figure 2.21: Visual example of max-pooling and average pooling with a 2x2 filter.

In a regular CNN, there are several convolutional layers and pooling layers. After the image is downsized, the data will be fed into a fully connected network layer. The data will, at this point, be condensed to a 1D array, which makes it ideal to be fed into a fully connected layer of neurons. There may be one or more fully connected layers before the final output. The last fully-connected layer will have the same number of outputs as there are possible classifications for the dataset [43] - [44].

2.5.4 U-Net Model

The U-net architecture was first introduced in an article from 2015 by Ronneberger, Fischer, and Brox [54]. Here the authors introduced a new architecture to work on segmentation for biomedical images, which only required a small amount of data to achieve great results. While a regular convolutional neural network performs well on classifying images, it does not achieve the same results when the wanted output is an image of a similar shape as the input image, as is the case for segmentation [54].

The architecture of the U-net model that Ronneberger et al. present is where the name U-net comes from. In the U-net, an image is down-sampled several times and then up-sampled again, combined with concatenating earlier layers to get back to or closer to the original size of the image. The U-net model presented in the article consists of five down-sampling steps referred to as the encoder and five up-sampling steps referred to as the decoder. Each decoder step consists of two convolutional layers with kernel size 3x3 and padding. Then follows a Max Pooling step with kernel size 2x2 and strides at two, which halves the size of the image. The number of feature channels is doubled for each step, starting at 64 [54].

When the image is downsized enough, the decoder part of the network starts to up-sample the image again using a 2x2 kernel of up-convolution while maintaining features by connecting the image to feature maps extracted in the encoder part of the network. For each upsampling, the number of feature channels is halved.

The original article used the network to segment neuronal structures in electron microscopic stacks. Since then, the U-net architecture has become one of the go-to architectures for medical image segmentation tasks. A few examples of cases where the architecture has since been used is kidney tumor segmentation from CT images [55], brain tumor segmentation [56], and segmentation of retinal vessels for more efficient diagnosis [57].

2.6 Brain structures

This section briefly introduces the location and shape of the structures selected for auto segmentation in this thesis.

2.6.1 The brain

The brain, shown in Figure 2.22, is an organ residing inside the cranium. It regulates all processes within the human body, process information, and control emotion.

Figure 2.22: Visualization of the human head. Brain is highlighted in pink. Figure retrieved from [58].

2.6.2 The brainstem

The brainstem, illustrated in Figure 2.23, is the structure that connects the brain to the spinal cord. The brainstem is participatory in controlling many processes such as respiration, digestion, and blood pressure [59].

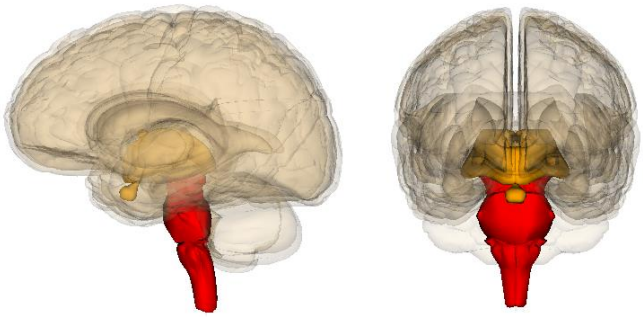

Figure 2.23: Image of the human brain, with the brainstem highlighted in red. Figure retrieved from [60].

2.6.3 The Papez circuit

The Papez circuit, illustrated in Figure 2.24, is a structure that involves parts of various other structures in the brain. It is part of the limbic system, which is involved in processing emotions and memories [61].

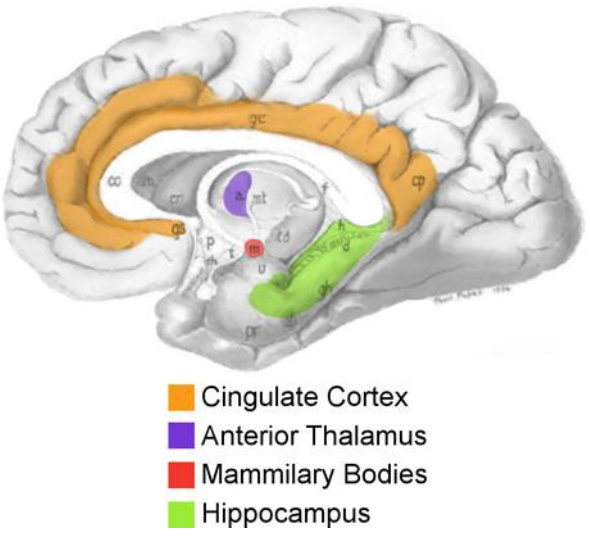

Figure 2.24: Figure of the human brain, with the Papez circuit highlighted. The figure is adapted from [62].

2.6.4 The hippocampus

The hippocampus, illustrated in Figure 2.25, is a small structure at the bottom of the cerebrum. The hippocampus plays a large part in learning, short-term memory, and storing information [63].

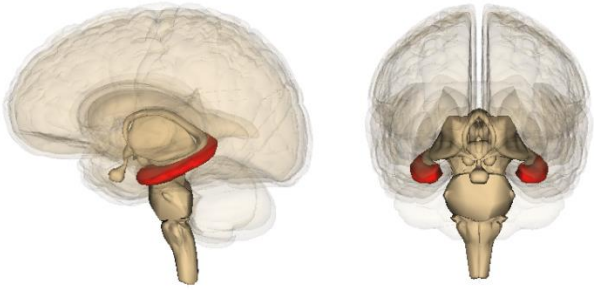

Figure 2.25: Image of the human brain, with the hippocampus highlighted in red. The figure is retrieved from [64].

3 Method

This section describes the data included in this study, the data extraction and preparation process, how the network for training models is created and how the model's performance is measured.

3.1 Data acquisition

The data used in this thesis is from thirty pediatric cancer patients that underwent proton therapy for different forms of brain cancer. All data has been anonymized. All scan sequences used in this project were acquired for diagnostic and treatment purposes and not specifically for training a ML algorithm. All patients had one or more CT scans done, and almost all patients also had a T1 weighted MRI done. Twenty of the patients had T2 weighted MRI scans done. The patients were part of three different clusters with three different diagnoses, and which scan modalities and corresponding contoured structures between the three clusters varied slightly. The same professional delineated the structures used for training as ground truth for all the patients.

3.2 Data preparation and network architecture

3.2.1 File storage and format

The data is stored in the DICOM file format, the industry standard for medical images. The files were structured into folders for each patient, with subfolders for each scan. There was a large discrepancy in the number and type of scans the patients had done. Only one scan of each modality was used for each patient. Each scan slice is stored in a separate file, which means each file had to be opened to extract the pixel data. The length of a Scan sequence varied from 200 to 300 slices, depending on how much of the neck and shoulders were included in the scan.

The structure files were stored as a single file, with a number referencing the corresponding CT scan. All data preparation was done using python 3.9. To work with the dicom files, the pydicom open-source library was used. This library contains built-in functions for reading, writing, and working with DICOM files. All code can be found in appendix 1.

3.2.2 Python code summary and illustration

A python class was created to gather all necessary information about a single patient. This class includes the patient id, data for MRI T1, MRI T2, CT scans, and structure data. The `Patient` class only requires the patient id to be initialized. A separate class was also created for structures. This was done to ensure that collecting and connecting data could be done as smoothly as possible. The `Structure` class requires a patient id and the structure's name to be initialized.

The `Structure` class contains three functions: One to get the file containing all structure data, one to get the structure id (ROINumber) corresponding to the structure name

(ROIName), and one to extract the information from the given structure based upon that id. The "`get_file`" function checks all files on the working directory for a filename that matches the id number for the patients and then returns the file. The "`get_structure_id`" collects the structure id (ROIName) corresponding to the given input name (ROIName), and "`find_structure`" extracts the structure data (ReferencedROINumber.Contourseqence) corresponding to the given input structure.

The `Patient` class has several functions. The first few functions automatically extract CT and MRI data from the DICOM files. It then checks to see if a scan with the given modality exists for this patient and if it does, it will sort the files and return them as a list. The function "`get_structure`" calls upon the `Structure` class to get the structure data into the patient class. It creates a `Structure` object and uses all three functions to get the structure data. The data is then all combined in the "`get_pixels_and_masks`" function. A flowchart of the code can be seen in Figure 3.1.

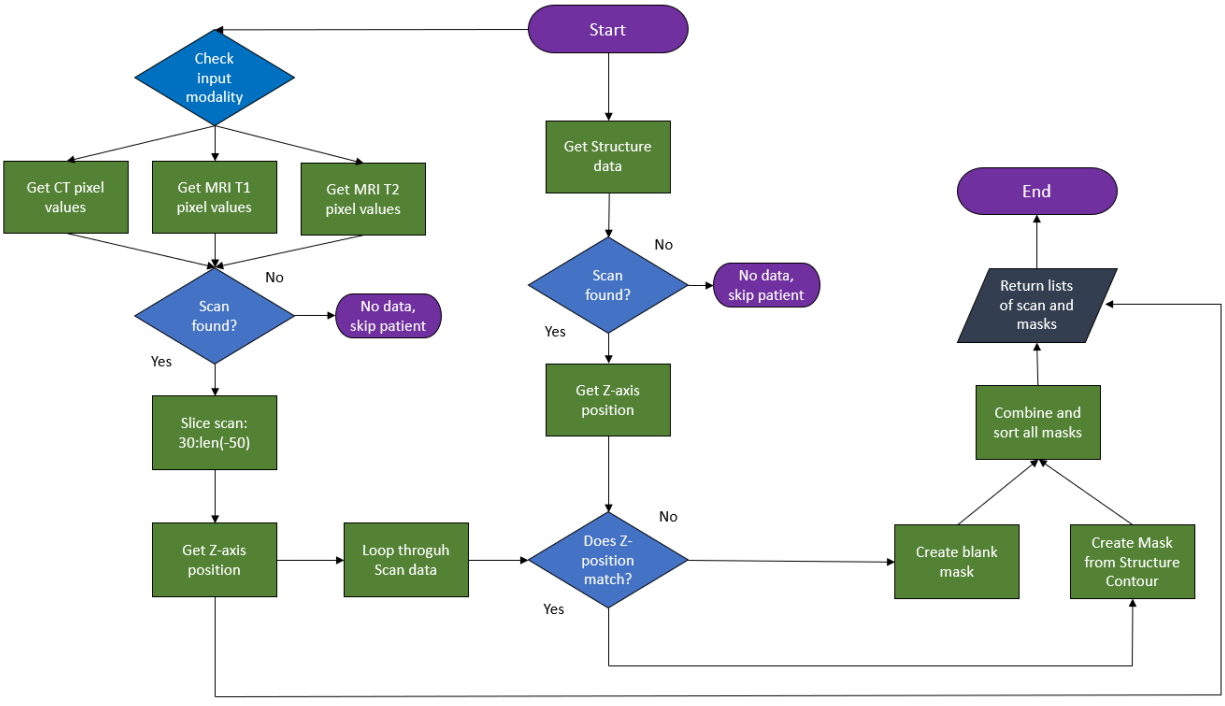

Figure 3.1: The figure illustrates the flow to get data from files and into sorted lists of masks and slices.

The "`get_pixels_and_masks`" function uses most of the previously defined functions to get structure data and scan data. It then matches the scan to the corresponding structure

slice using the z-axis position of both. Masks are then created for each structure slice using the polygon function from scikit. The structure mask and corresponding scan are sorted and returned as arrays.

The main function takes three inputs; the number of patients, which scan modality should be extracted, and which structure to segment. It then calls upon another function that generates all the data needed to train the network. This function generates `Patients` using the function illustrated in Figure 3.2, and stores all the scans and masks, ready to be fed into the network for training. The data was split into training, validation, and testing set, with three patients for validation, three patients for testing, and the remainder of patients put into the training set. The same three patients were used in validation for all models to keep a consistent validation set and ensure the inclusion of all three patient groups. The same was done for the testing set.

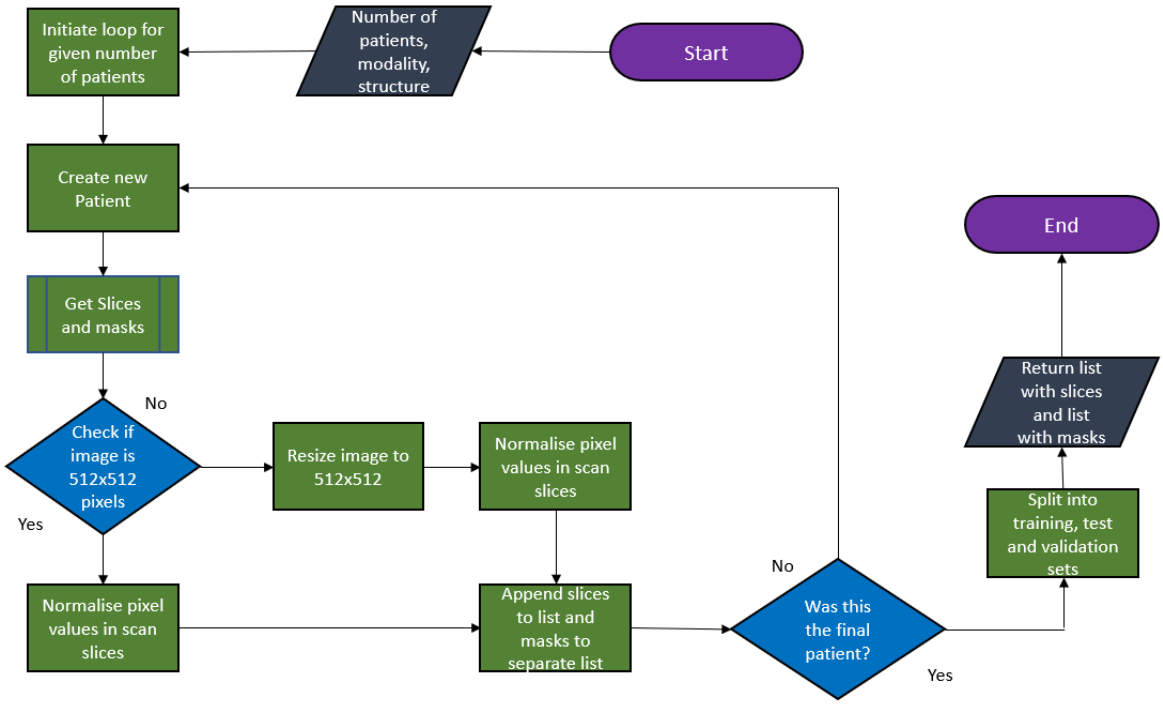

Figure 3.2: Illustration of code to generate all data ready to be fed into the network. Flowchart for get slices and mask function can be seen from Figure 3.1.

The masks used for training were created using the Polygon function from the scikit-image package [65]. By generating coordinates for the contour, a complete mask was created. The values were set to 1 inside the mask and 0 outside. A few patients also had images stored in 320x320 pixels instead of the usual 512x512 pixels. These images were rescaled to 512x512 using the resize formula from scikit-image [65].

Several preprocessing methods were evaluated to try and gain the best possible segmentations. All variations of the pixel data were normalized using $\frac{pixel\ value}{4096}$ to between [0, 1] for CT scans and $\frac{pixel\ value}{\max sequece\ value}$ for MRI data. The total scan length was also reduced by 80 slices to exclude unnecessary data points, 25 from the start of the scan and 65 from the end. An additional reduction of images that did not contain significant data was made to additional slices. Most scan sequences containing images with noise or the length of the scan sequence were done well beyond the slices reduced in the previous step. A visual inspection combined with a numeric evaluation revealed that MRI slices with a maximum pixel value below 0.3 after normalization were not needed to get the needed slices for training. For CT, it was discovered that a sum of all pixels below 4400 after normalization did not contain any important information. For T2 weighted MRI, the scans with a maximum pixel value below 0.38 were dropped. Most images dropped were blank slides or nearly blank slides.

3.2.3 Network construction and illustration

The network is built on the TensorFlow [66] platform using the Keras [67] module. Keras has built-in functions that perform the calculations needed to build the U-net model, illustrated in Figure 3.3. For the down-sampling part of the network, the `Conv2d` function provides the needed operations for the convolutional layer, and the `MaxPool2d` function provides the pooling operation. For the up-sampling part, `Conv2DTranspose` preforms the convolutional parts while `concatenate` combines the encoder layers' feature maps with the feature maps from the decoder layers. The number of feature maps is doubled for each down-sampling, starting at 16 and ending at 256. This is done in reverse for the up-sampling

part, ending at 16 feature maps before the final output layer with just one. In addition to this, `dropout` was implemented after the first `Conv2D` layer. The dropout was set at 0.1 for the first two down-sampling and last two up sampling layers. It was set at 0.2 for the third and fourth down-sampling layer, first and second up sampling layer, and 0.3 for the middle layer. The network's final output is a 512x512 matrix with values between 0 and 1, where 1 indicates that the model is confident that the pixel is part of a structure, and 0, where the model is confident there is no structure. Since the input values from the masks are binary, any output value below 0.5 was considered a prediction of no structure, and any value above 0.5 was considered a prediction of structure.

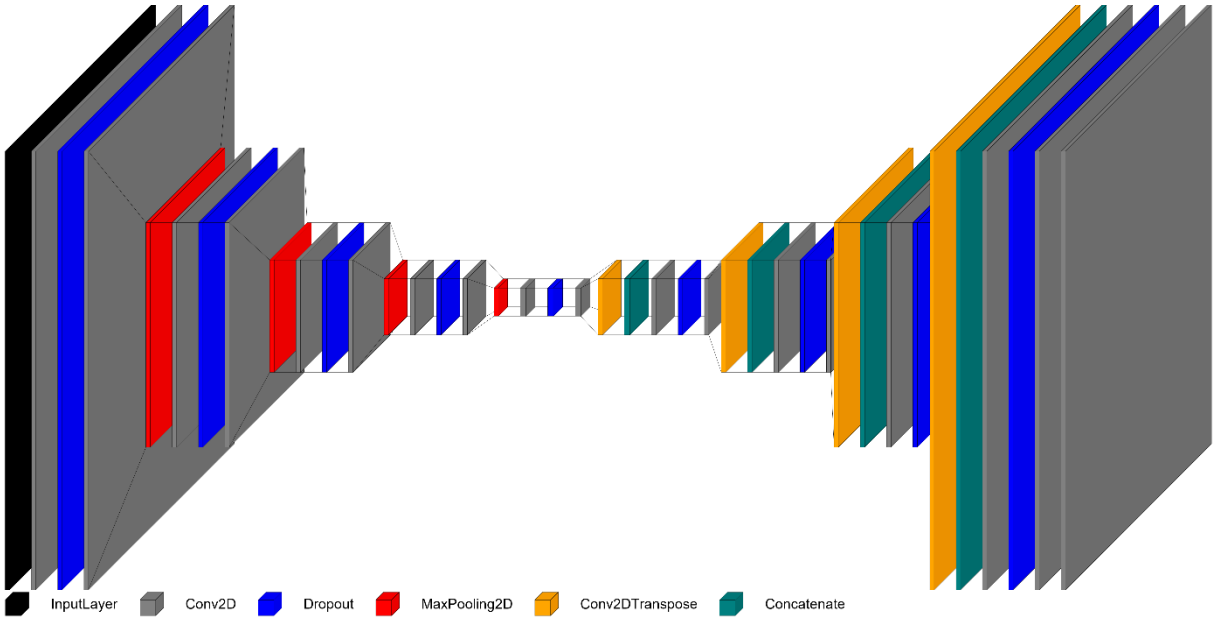

Figure 3.3: Figure illustrating the different layers of the network layout. Figure created using [68].

An additional network was also made, with the same number of feature maps as the original U-net architecture. Here the number of feature maps started at 64, and doubled for each up and down sampling, resulting in 1024 at the center, and back to 64 before the final output layer. This network was used to investigate if increasing the number of feature maps would significantly improve the model output.

Hyperparameters and tuning

Several loss functions were tested to optimize the network. The first one tested was Binary Cross entropy, a built-in function in Keras [67]. Cross entropy is a measure of the difference between two probability distributions of a set of events. [11]. The loss function checks each pixel and is defined as:

$$L_{BCE}(y, \hat{y}) = -(ylog(\hat{y}) + (1 - y)\log(1 - \hat{y}))$$

where y is the true label, and $\hat{y}$ is the predicted label. A weighted version of Binary Cross Entropy was also tested. This is often used in unbalanced datasets, where one label highly outnumbers the other. This is the case for many segmentation tasks, where only a fraction of the pixels are part of the segmentation, and the rest are false. This loss function can help weight true positives more than true negatives, so the model does not predict every pixel as 0 and gets 99% accuracy.

$$L_{w-BCE}(y, \hat{y}) = -(\beta * ylog(\hat{y}) + (1 - y)\log(1 - \hat{y}))$$

Dice loss was also implemented and is one of the more common loss functions used in segmentation tasks. The Dice loss function is adapted from the Dice coefficient described in section 2.5.2 by adding 1 to the numerator and denominator to exclude undefined cases.[69] Dice loss is then expressed in as

$$Dice\ loss = 1 - \frac{2 \cdot y \cap y_{pred} + 1}{y + y_{pred} + 1}$$

To prevent overfitting, the loss for each epoch was plotted after the network had finished training the model. If the training loss started to increase, an epoch before the loss increase was recalled, and the neuron weights at this point was used for predictions.

Optimizer and Batch size

The optimizer used in the network is Adam [48], a version of stochastic gradient descent that is more efficient and faster at updating weights. The learning rate was set at 10^{-4} and batch size at 16 to ensure slow updates of the network weights. The batch for validation was also set to 16. Several other parameters for these two were also tested.

Metrics

Several metrics were used to measure the performance of the model. The primary one and the one used to measure the performance during training is the Dice coefficient, which provides a measure the overlap of the two images, as described in section 2.5.2. The overlap was calculated for all voxels for volume instead of slice by slice. To supplement the Dice coefficient, recall and precision were also checked for the more complex structures to understand better how the model performed. As the model output is a pixel value representing the model's confidence in that pixel being part of the segmentation mask, both precision and recall round that value up if the value > 0.5 and down if the value < 0.5.

4 Results

This section will present the results of the auto-segmented structures using different imaging modalities during training. The results will be presented with Dice, Precision, and Recall scores

for the respective models and selected slices for the different imaging modalities. The author of this thesis wrote the code used to perform data extraction and preparation. The network used for training had the U-net architecture, an already established architecture.

4.1 Segmentation of the brain

The first objective was to segment the full brain structure. The segmentation was done nine times using the three different loss functions and modalities to ensure the best possible result.

4.1.1 Segmentation of the brain using CT images

Images segmented from CT with Binary Cross entropy as loss function gave great segmentations and a Dice score of 0.98. There was no significant difference between the Dice score for models trained using the three loss functions. All dice scores can be found in Table 4-1.

Table 4-1: Dice, precision, and recall score for brain segmentation using CT images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	0.981	0.985	0.982
Binary Cross	0.979	**0.987**	0.974
Dice loss	**0.982**	0.981	**0.984**

Visual inspection of segmentations did not reveal any apparent flaws in the segmentation, as displayed in Figure 4.1.

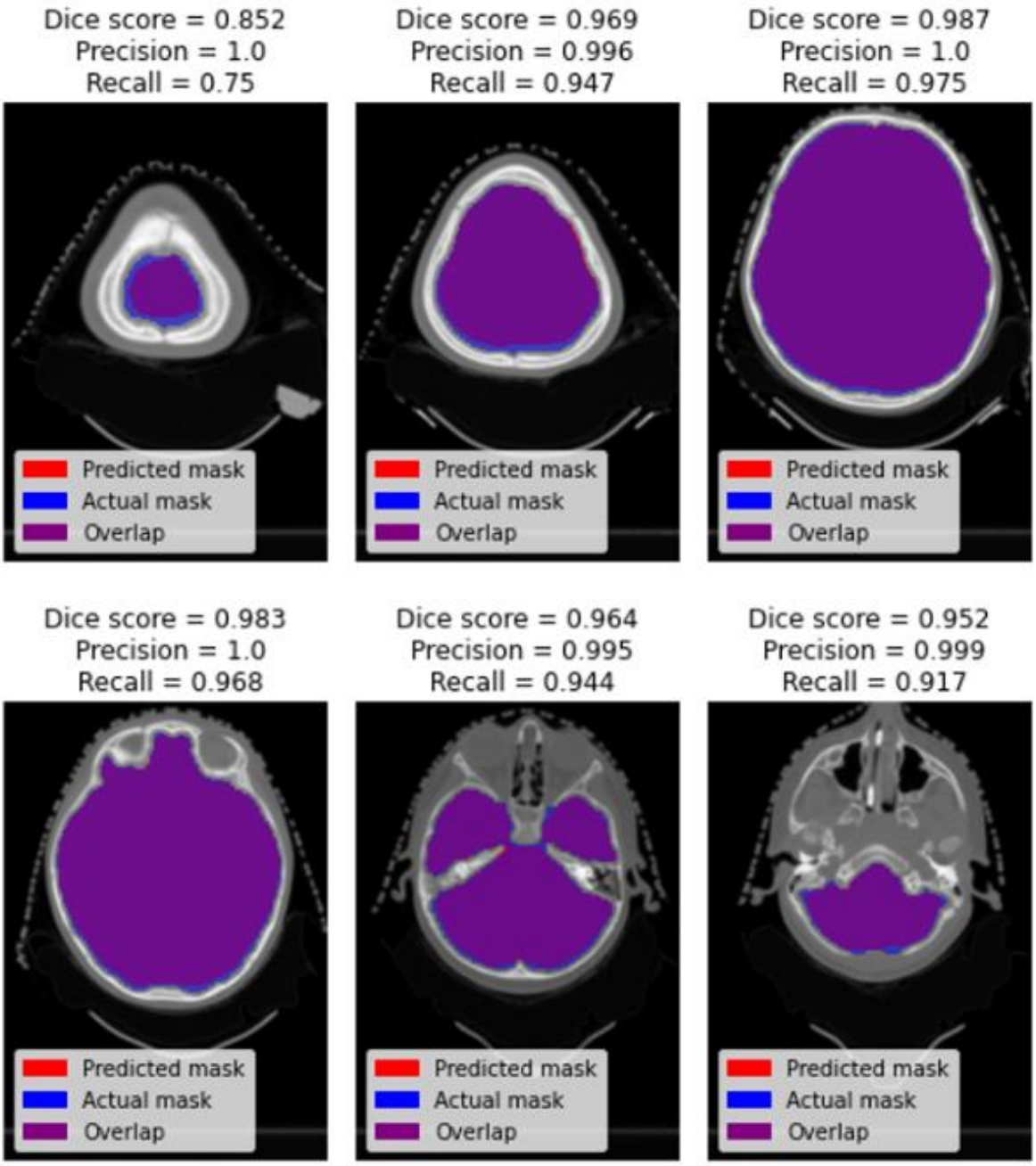

Figure 4.1: Segmentation results for full brain using CT images and Binary Cross entropy as loss function. Precision, Dice, and recall scores for the individual slices are displayed above the slice. The top left image reveals a slight underprediction at the start of the sequence, but the prediction accuracy increases as the structure goes on, as can be seen from the following images.

The model struggles the most around the edges of the contour and the total volume of the first few slices but does well in grasping the general outline of the structure. The highest dice score of 0.98 is a great result.

4.1.2 Segmentation of the brain using T1 MRI images

Segmentation of the entire brain using T1 weighted MRI images gave good results but not quite as good as using CT images. The highest dice score was 0.952. There were only minor discrepancies between the three models trained with the different loss functions. Dice scores for all three models can be found in Table 4-2.

Table 4-2: Dice, precision, and recall score for full brain segmentation using T1 weighted MRI images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	0.944	0.958	0.934
Binary Cross	0.936	0.950	**0.940**
Dice loss	**0.952**	**0.970**	0.935

Figure 4.2 gives a visual representation from a few slices:

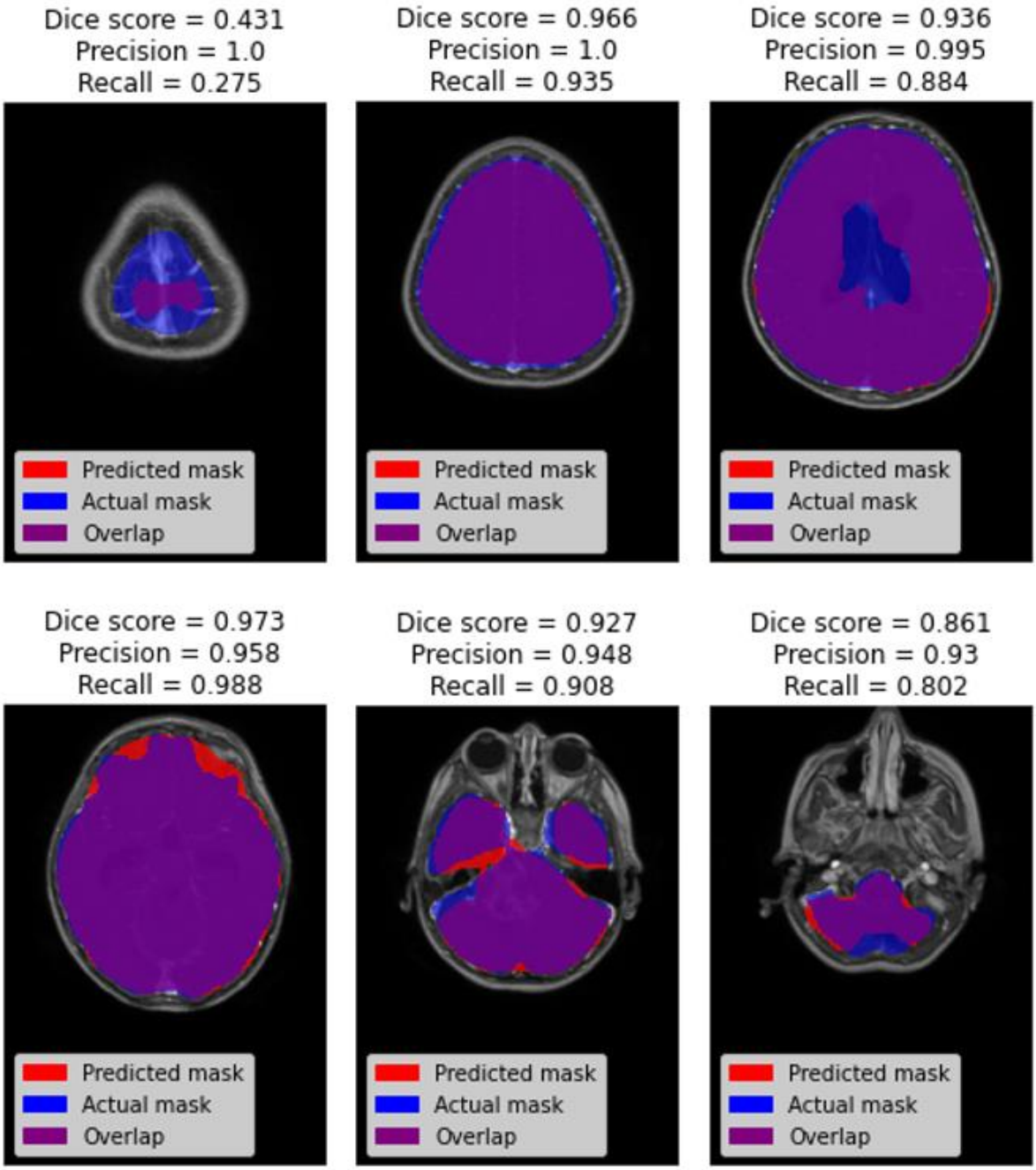

Figure 4.2: Segmentation of the full brain using T1 weighted MRI images and Dice loss. The top-left indicates that the model struggles with segmentation at the start of the sequence. The model also has problems with darker areas and finer structure edges, as seen from the top right and bottom right images. Precision, Dice, and recall scores for the individual slices are displayed above the slice.

The model found the general outlines of the structure but struggled with the start of the imaging sequence, same as CT. However, it also struggled to pick up the finer details of the structure and had problems with darker image areas.

4.1.3 Segmentation of the brain using T2 MRI images

Segmentation of the brain using T2 weighted images resulted in adequate segmentation, with the best Dice score being 0.959 for Dice loss. Dice, recall, and precision scores for the three models trained with different loss functions can be found in Table 4-3.

Table 4-3: Dice, precision, and recall scores for full brain segmentation using T2 weighted MRI images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross W0 0.5	0.951	0.945	**0.978**
Binary Cross	0.956	**0.961**	0.964
Dice loss	**0.959**	0.957	0.960

A visual representation of selected slices is displayed in Figure 4.3.

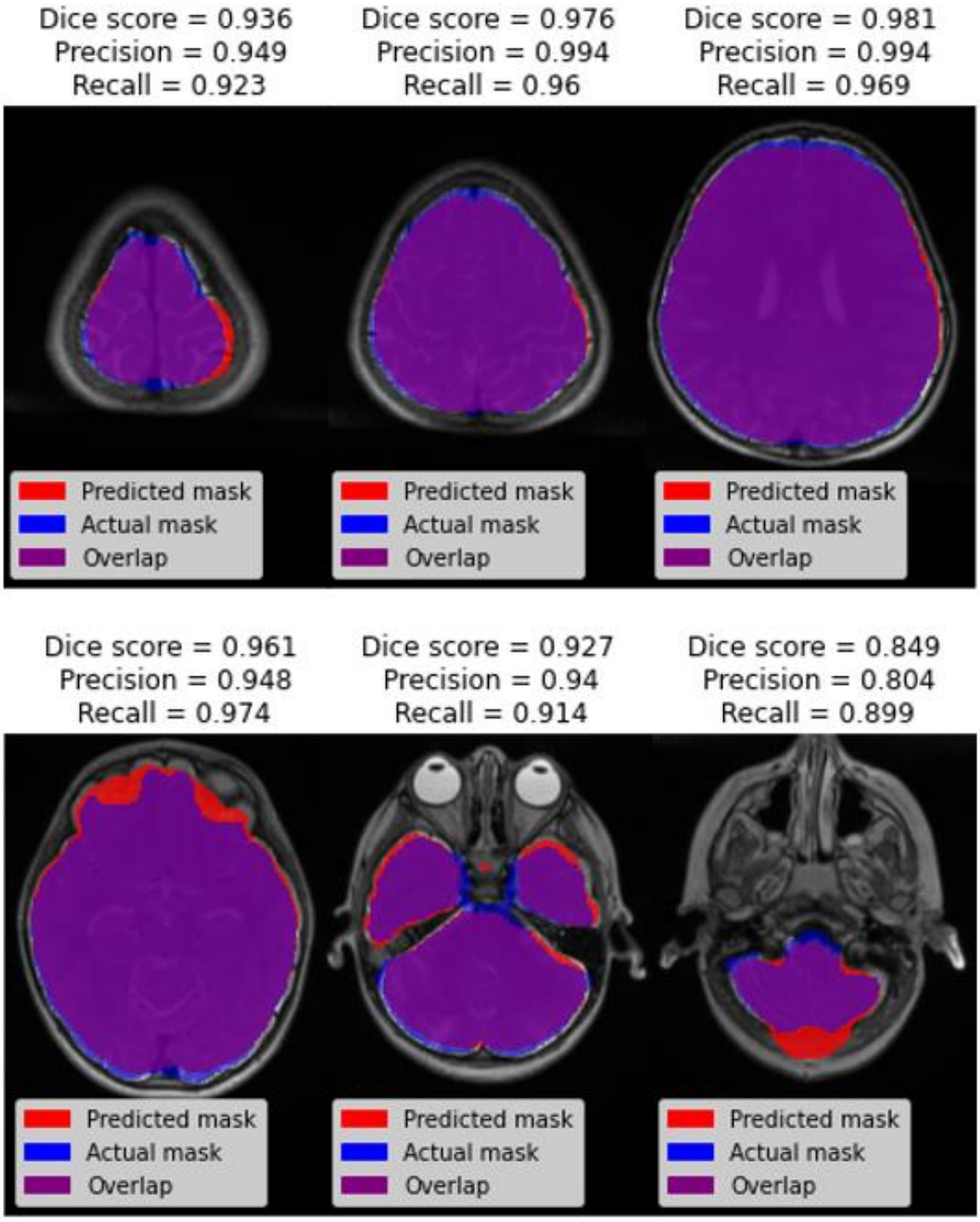

Figure 4.3: Selected slices of segmentations of the full brain using T2 weighted MRI images. Precision, Dice, and recall score for the individual slices is displayed above the slice. The model struggles around the edges of the structure, and in areas where the structure has more complex shapes, for example, the bottom right slice.

The model did well in finding the general structure outline but struggled the most with edges, especially in the areas where the structure is less smooth as seen in Figure 4.3: Selected slices of segmentations of the full brain using T2 weighted MRI images. Precision, Dice, and recall score for the individual slices is displayed above the slice. The model struggles around the edges of the structure, and in areas where the structure has more complex shapes, for example, the bottom right slice.

4.1.4 Best model and summary of full brain segmentation

CT images with dice loss as loss function was the best model for full brain segmentation. As can be seen from Figure 4.4, dice precision and recall remain high for all models, with variations between 0.94 and 0.98 in dice score.

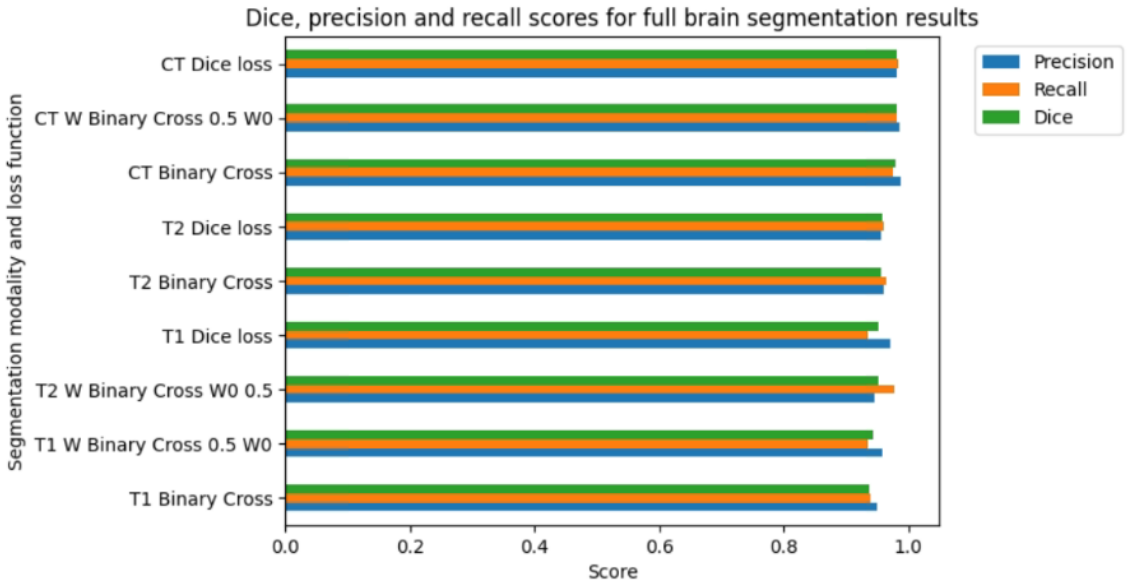

Figure 4.4: Graph displaying the different Dice, precision and recall scores for segmentation results for the brain. CT seems to be performing the best, while T1 does the worst. Overall there is not a very large discrepancy between the different loss functions. W Binary cross 0.5 W0 means false negatives were penalized twice as hard, and W Binary cross 0.5 W1 false positives were penalized twice as hard.

4.2 Segmentation of the brainstem

Nine different models were tested for segmentation of the brainstem, with three modalities and three loss functions. Dice loss did not converge for this structure, so a different weight was applied to weighted binary cross entropy instead.

4.2.1 Segmentation of the brainstem using CT images

Segmentation of the brainstem on CT images using binary-cross entropy as a loss function and Dice score as a metric gave a Dice score of 0.63 on test images. There was only a slight difference in dice score when training models with the different loss functions. All scores can be found in Table 4-4.

Table 4-4: Dice, precision, and recall score for segmentation of the brainstem using CT images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W1	0.579	**0.796**	0.476
Binary Cross	**0.631**	0.755	**0.582**
Weighted Binary Cross 0.5 W0	0.622	0.724	0.582

A visual representation of selected slices is displayed in figure Figure 4.5.

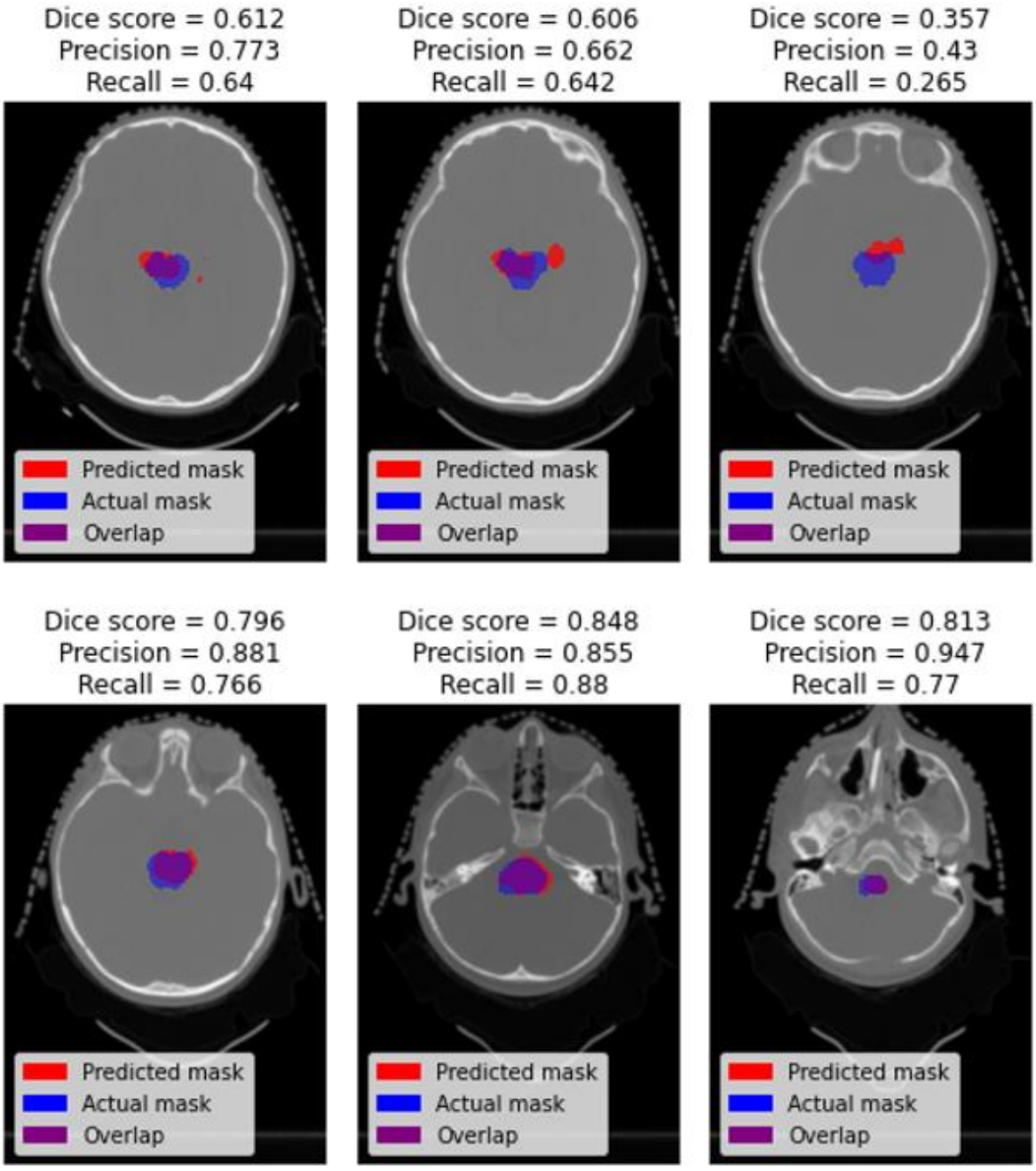

Figure 4.5: Segmentation results of the brainstem using CT images and Binary Cross entropy. Precision, Dice, and recall scores for the individual slices are displayed above the slice. The model appears to struggle with the placement of the structure for the first half of the structure, but accuracy and Dice score increase as the structure continues.

The visual inspection of the images revealed that the model started to predict the possibility of the structure much earlier than the actual structure. However, the pixel-wise structure prediction is below 0.5, indicating that the model is not sure that the pixel is part of the structure, just a slight possibility for it. The model grasped the approximate area of the structure but struggled to get the precise location and shape correct.

4.2.2 Segmentation of the brainstem using T1 MRI images

Segmentation of the brainstem using T1 weighted MRI images gave segmentations with a Dice score of 0.556 using weighted Binary Cross entropy as a loss function penalizing false positives twice as hard as false negatives. Models trained with different loss functions only slow slight differences. Dice scores for all models are shown in Table 4-5.

Table 4-5: Dice, precision, and recall score for segmentation of the brainstem using T1 weighted MRI images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	**0.546**	0.717	**0.456**
Binary Cross E54	0.524	0.742	0.410
Weighted Binary Cross 0.5 W1	0.530	**0.768**	0.426

A few select slices of brainstem segmentation results using T1 MRI images are displayed in Figure 4.6

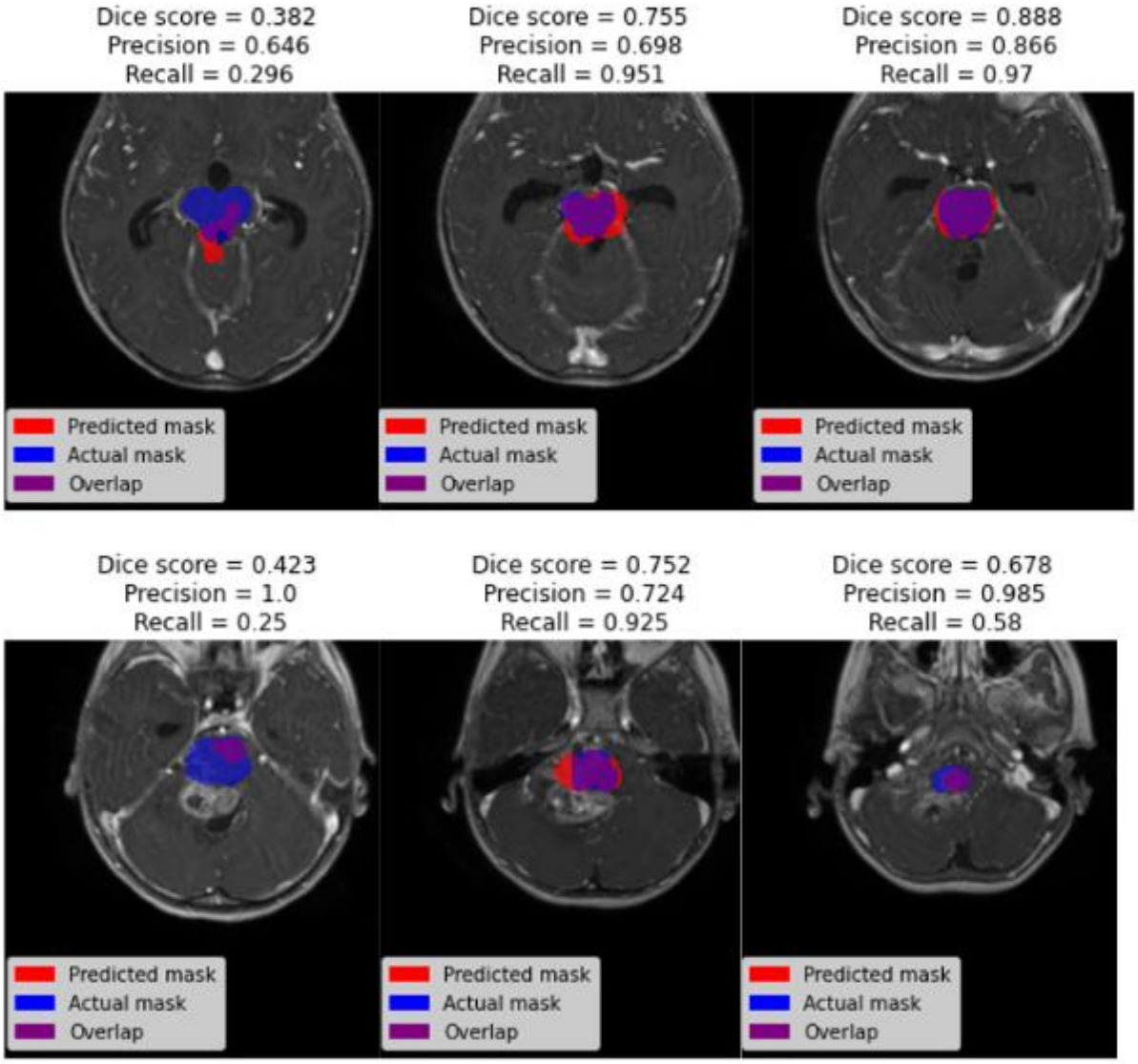

Figure 4.6: Segmentation results of the brainstem using T1 weighted images and weighted Binary Cross entropy. Precision, Dice, and recall scores for the individual slices are displayed above the slice. The model appears to struggle with the placement of the structure at the start, then switching between overshooting and undershooting the prediction.

Segmentation results reveal that the model struggles at the start of the structure, predicting odd shapes compared to the structure. The model contineues to grasp the location of the structure, but struggles with the exact shape throughout, either predicting a larger area or a smaller area than the structure actually is.

4.2.3 Segmentation of the brainstem using T2 MRI images

Segmentation results using T2 weighted MRI images gave a Dice score of 0.734 using weighted binary cross entropy as loss function, weighting false positive values more than false negatives. Scores for all three models trained with different loss functions are displayed in Table 4-6.

Table 4-6: Dice, precision, and recall score for segmentation of the brainstem using T2 weighted MRI images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross W0 0.5	**0.734**	**0.764**	**0.760**
Binary Cross	0.675	0.737	0.709
Weighted Binary Cross W1 0.5	0.673	0.758	0.649

A few selected slices illustrating brainstem segmentation using T2 Mri images are displayed in Figure 4.7.

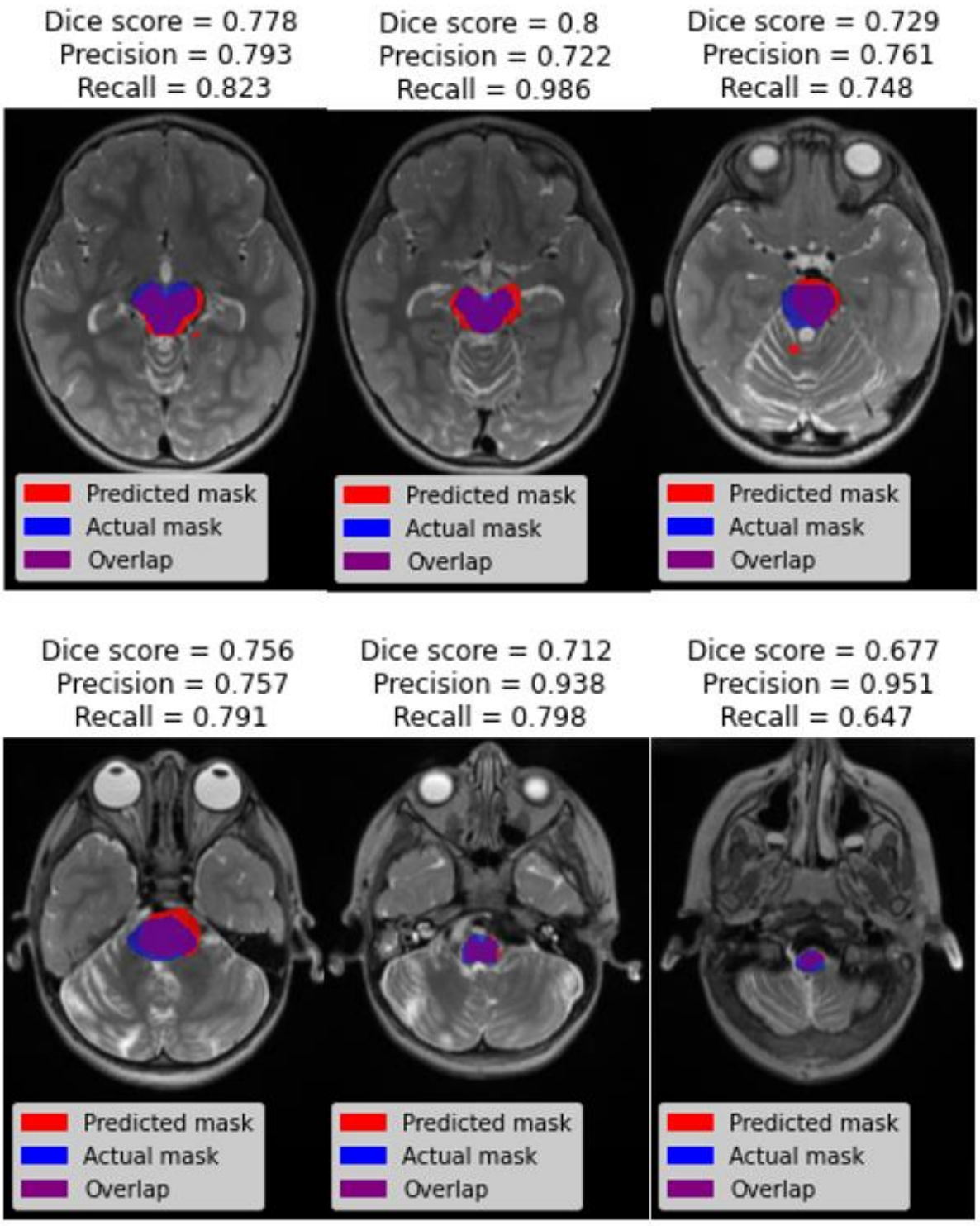

Figure 4.7: Selected slices of segmentation results for the brainstem using T2 weighted MRI images and weighted binary cross entropy as loss function. Precision, Dice, and recall scores for the individual slices are displayed above the slice.

The model predicts a few random locations where there is no brainstem before the actual structure starts, but then predicts the location accurately, but a bit larger and shifted for the start of the structure, and then predicted a smaller area for the latter part of the structure.

4.2.4 Best model and summary of brainstem segmentation

For segmentation of the brainstem, all models trained on T2 images performed the best, with the best loss function being weighted binary cross entropy, which penalized false negatives twice as much as false positives. Despite this, it still maintained a relatively high precision, indicating that it did not heavily overpredict the structure. Both recall and precision were high for the two best models. Recall was lower for the remainder of the models. At the same time, precision was higher, indicating that the lower-performing model's prediction of pixels was mostly correct. However, it also missed many pixels that should have been predicted as part of the structure. A visualization of all dice, recall, and precision cores are displayed in Figure 4.8.

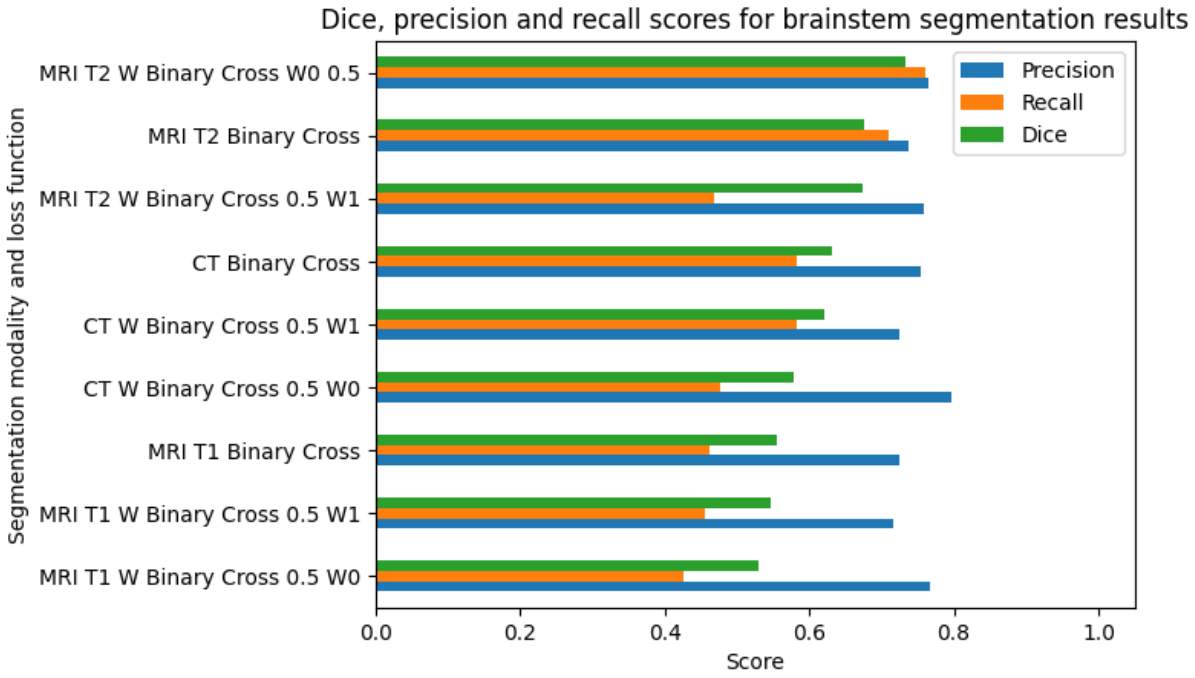

Figure 4.8: Graph displaying the different dice, precision, and recall scores for segmentation results for the brain. CT seems to be performing the best, while T1 does the worst. Overall there is not a very large discrepancy between the different loss functions. W Binary cross 0.5 W0 means false negatives were penalized twice as hard, and W Binary cross 0.5 W1 false positives were penalized twice as hard.

4.3 Segmentation of the Papez Circuit

The Papez Circuit is a complicated structure involving several parts of the brain, as described in section 2.6.3. Nine models were created for this structure as well. Dice loss did not converge for this model, so a different weighting for weighted binary cross entropy was used instead.

4.3.1 Segmentation of the Papez Circuit using CT images

Segmentation of the Papez Circuit using CT images and weighted binary cross-entropy penalizing false negatives twice as much as false positives gave a Dice score of 0.278 on test images. Different loss function seems to have little impact on the dice score, which remains low for all three models. Dice scores for all three models can be found in Table 4-7.

Table 4-7: Dice, precision, and recall score for segmentation of the Papez Circuit using CT images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	**0.278**	0.265	**0.315**
Binary Cross	0.257	**0.338**	0.195
Weighted Binary Cross 0.5 W1	0.204	0.275	0.115

Visual representation of selected slices can be found in Figure 4.9.

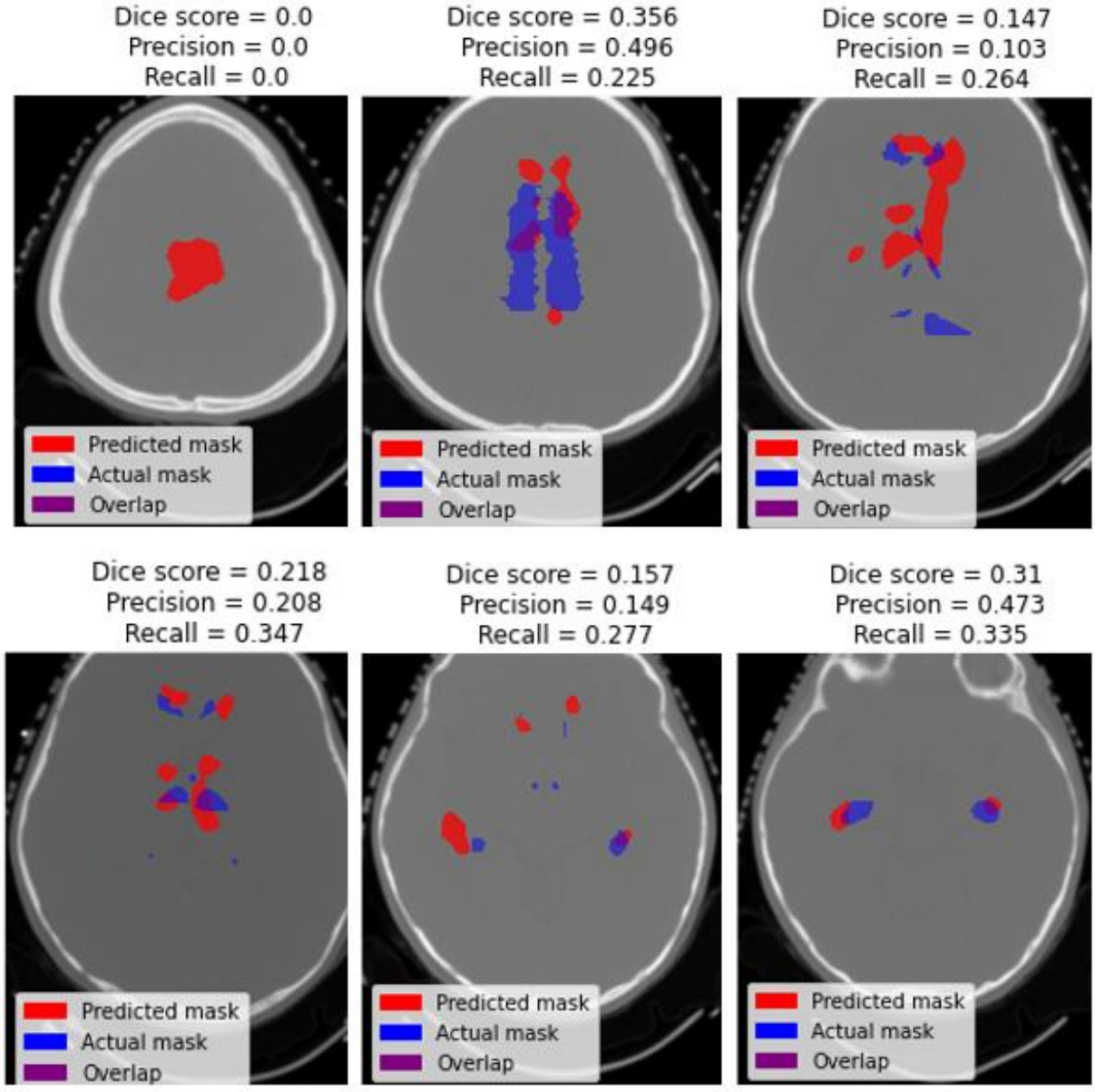

Figure 4.9: Segmentation of the Papez Circuit using CT images and weighted binary cross entropy as loss function. Precision, Dice, and recall scores for the individual slices are displayed above the slice. As can be seen from nearly all images, the model struggles to predict most of the structure. There are areas where it is close, but nothing of substantial value.

Visual inspection indicates that the model struggles to predict most of the structure, and no clear pattern can be discovered from the predicted segmentation. The model predicts the last part of the structure best, but still nowhere near anythin that can be useful.

4.3.2 Segmentation of the Papez Circuit using T1 MRI images

Segmentation of the Papez Circuit using T1 weighted MRI images and weighted binary cross-entropy penalizing false negatives twice as much as false positives 0.5 gave a Dice score of 0.303 on test images, marginally better than for CT. Dice, recall and precision scores for the three models trained with different loss functions are displayed in Table 4-8.

Table 4-8: Dice, precision and recall score for segmentation of the Papez Circuit using T1 weighted MRI images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	**0.303**	0.357	**0.292**
Binary Cross	0.261	**0.382**	0.209
Weighted Binary Cross 0.5 W1	0.192	0.367	0.101

Figure 4.10 displays a few selected slices of Papez circuit segmentation results from the model trained on T1 MRI images.

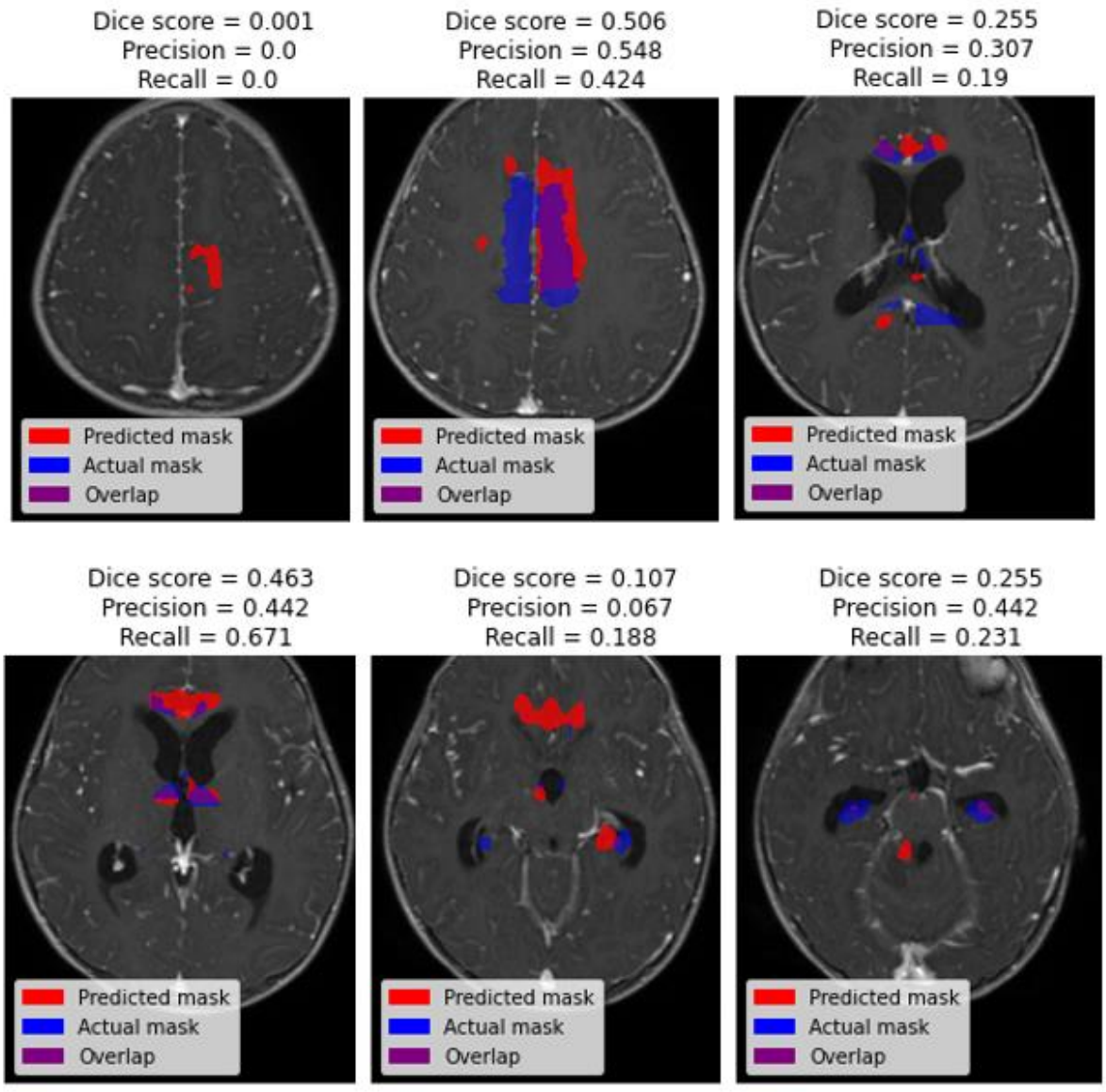

Figure 4.10: Segmentation of the Papez Circuit using T1 weighted MRI images. This model also struggles to find a coherent structure, and no segmentation of significant value is found. Precision, Dice and recall score for the individual slices are displayed above the slice.

Visual inspection indicates that this model also struggles with most of the structure, getting sporadic small areas close to correct, but nothing that is useful or can be considered a sucsessfull segmentation.

4.3.3 Segmentation of the Papez Circuit using T2 MRI images

Segmentation results of the Papez Circuit using T2 weighted MRI images gave a Dice score of 0.515 using Binary Cross entropy. Dice, recall, and precision scores for the three different models are displayed in Table 4-9.

Table 4-9: Dice, precision, and recall score for segmentation of the Papez Circuit using T2 weighted MRI images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross W0 0.5	0.491	0.494	**0.650**
Binary Cross	**0.515**	0.521	0.597
Weighted Binary Cross W1 0.5	0.466	**0.571**	0.487

A few selected slices of segmentation results for Papez circuit using T2 MRI images are displayed in Figure 4.11.

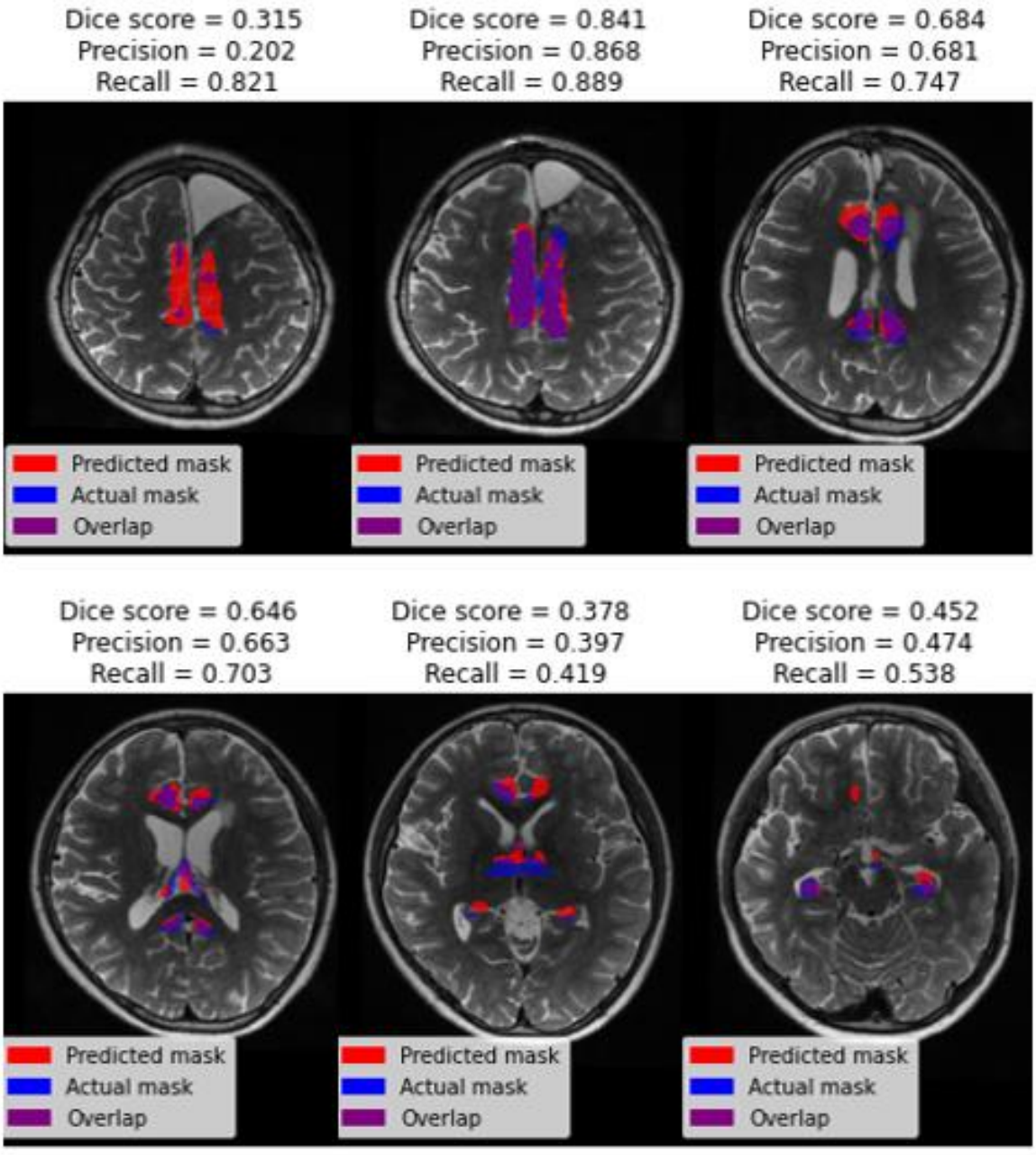

Figure 4.11: Segmentation of the Papez Circuit using T2 weighted MRi images. The model predicts the start of the structure a few slices before it starts, as can be seen from the top left slice. The model then predicts locations of the structure but struggles to predict exact size and shape. Precision, Dice and recall score for the individual slices are displayed above the slice.

The model trained with T2 predicts better than both T1 and CT, but still struggles with overshooting at the start and general precision of the areas. For some slices the model predicts close to accurately, with dice scores as high as 0.841 for the second slice displayed, but it also has areas where it misses almost the entire structure, such as the first slice displayed with a dice score of 0.315.

4.3.4 Best model and summary of Papez circuit segmentation

The Papez Circuit is the most complex structure in this study. The highest Dice score being 0.515 indicates that only about half of the predicted and ground truth masks overlap. The recall is 0.597 and precision is 0.521. Neither score is particularly good, indicating that the model predicts many false positive and false negative pixels. A visualization of all scores is displayed in Figure 4.12.

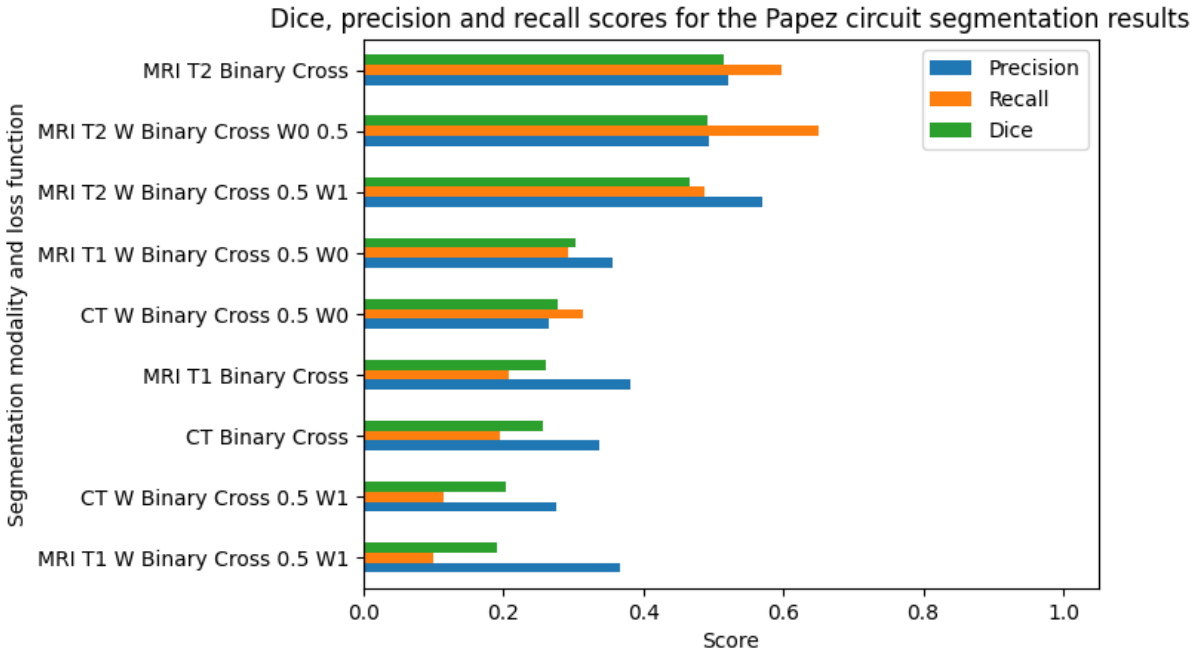

Figure 4.12: Visualization of scores for segmentation results of the Papez circuit. W Binary cross 0.5 W0 means false negatives were penalized twice as hard, and W Binary cross 0.5 W1 false positives were penalized twice as hard. W Binary cross 0.5 W0 means false negatives were penalized twice as hard, and W Binary cross 0.5 W1 false positives were penalized twice as hard.

4.4 Segmentation of the right hippocampus

The right hippocampus is the smallest structure segmented in this thesis. Nine models were trained and tested, three for each imaging modality. The imaging modality and loss function seemed to have a more considerable impact on the segmentation of this structure.

4.4.1 Segmentation of the right hippocampus using CT images

Segmentation of the hippocampus using CT images yielded poor results, with Dice scores below 0.2 for all loss functions. All Dice, precision, and recall scores for hippocampus segmentation using CT images can be found in Table 4-10.

Table 4-10: Dice, precision, and recall score for segmentation of the right Hippocampus using CT images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	0.156	0.150	0.160
Binary Cross	**0.158**	0.181	**0.168**
Weighted Binary Cross 0.5 W1	0.105	**0.262**	0.058

The model predicts the general area of the structure correctly but misses the start of the structure. It also predicts a much larger area towards the end of the structure, as illustrated in Figure 4.13.

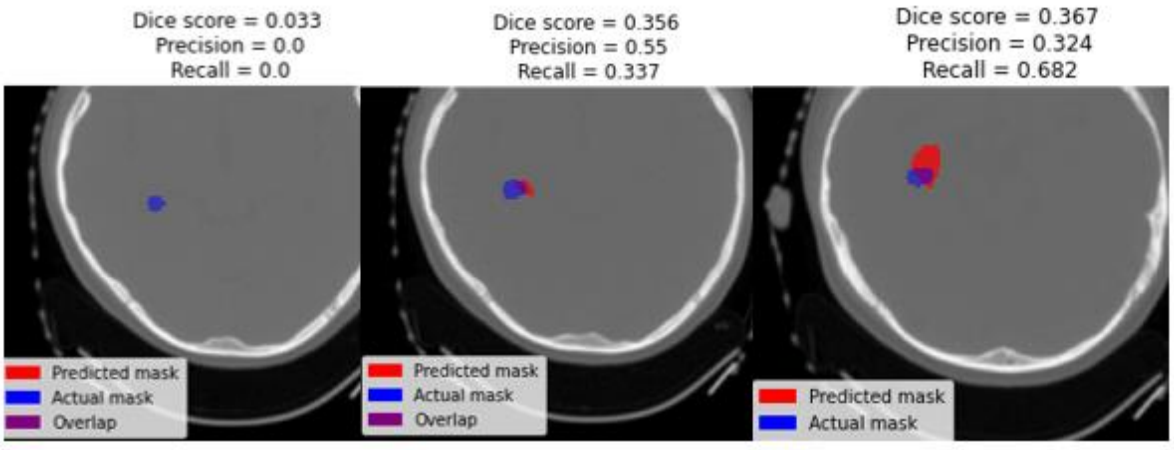

Figure 4.13: Segmentation of the right hippocampus using CT images and Binary Cross entropy as loss function. The model does not pick up on the start of the structure (Left), and continues to struggle throughout the sequence, overshooting the size by the end of the structure. Precision, Dice and recall score for the individual slices are displayed above the slice.

4.4.2 Segmentation of the right hippocampus using T1 MRI images

Segmentation of the right hippocampus using T1 weighted images and weighted Binary cross entropy as loss function gave a Dice score of 0.374. All Dice precision and recall scores for the three models can be found in Table 4-11.

Table 4-11: Dice, precision and recall score for segmentation of the right hippocampus using T1 weighted MRI images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	**0.374**	0.493	**0.378**
Binary Cross	0.275	0.437	0.233
Weighted Binary Cross 0.5 W1	0.187	**0.665**	0.074

Visual inspection revealed that the model struggled with the start of the structure. It also had a slight overshoot at the end of the structure. The middle part was about the correct size but shifted slightly down to the left. Example slices are displayed in Figure 4.14.

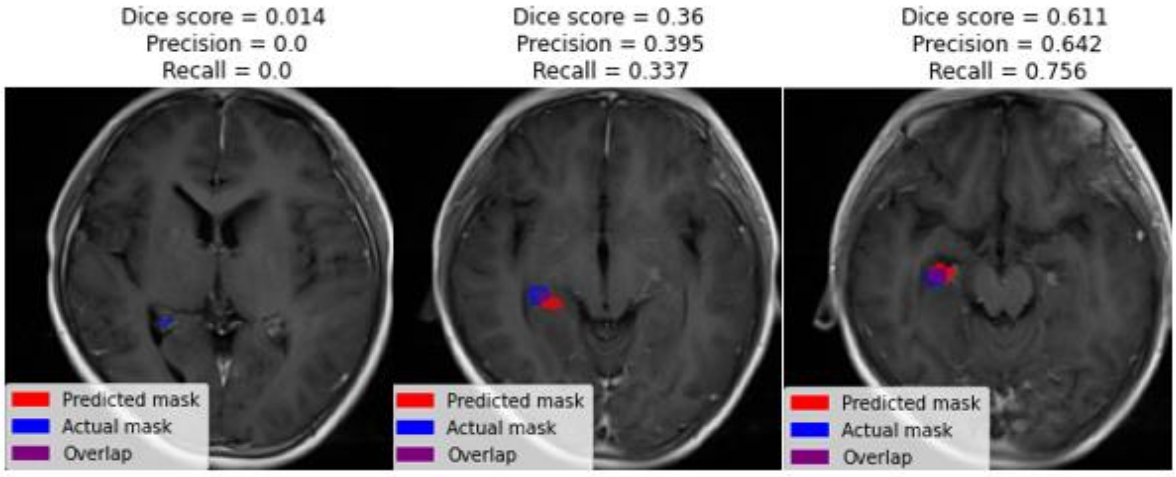

Figure 4.14: Segmented of the right hippocampus using T1 weighted MRI images and Weighted binary cross entropy as loss function. The model seems to be better at prediction the structure as it progresses and is most accurate towards the end of the structure, with Dice scores in the 0.6 range (Right). Precision, Dice and recall score for the individual slices are displayed above the slice.

4.4.3 Segmentation of the right hippocampus using T2 MRI images

Segmentation of the right hippocampus using T2 weighted MRI images gave a Dice score of 0.491 with Weighted binary cross entropy as loss function, weighted to penalize false positives more than false negatives. and values for all scores can be found in Table 4-12.

Table 4-12: Dice, precision and recall score for segmentation of the right hippocampus using T2 images. Best scores are hihgligted for visibility.

Loss function	Dice score	Precision	Recall
Weighted Binary Cross 0.5 W0	0.441	0.448	0.564
Binary Cross	0.474	0.486	**0.598**
Weighted Binary Cross 0.5 W1	**0.491**	**0.535**	0.575

The model struggles to get the exact location but improves throughout the structure, with better Dice scores towards the end of the structure. A visual representation of a few selected slices is displayed in Figure 4.15.

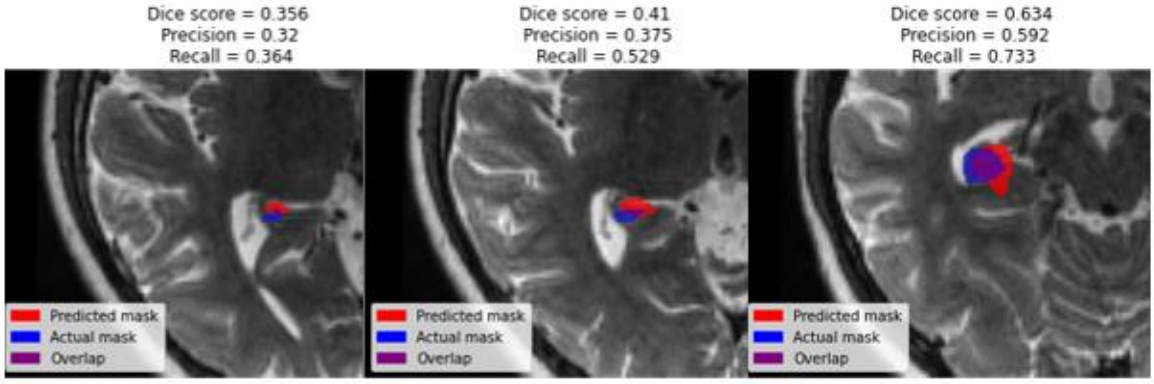

Figure 4.15: Segmentation of the right hippocampus using T2 weighted MRI images and weighted binary cross entropy as loss function. Precision, Dice, and recall score for the individual slices are displayed above the slice. The model struggels with the exact location of the segmenation at the start, but improves towards the end of the structure.

4.4.4 Best model and summary of right hippocampus segmentation

The best performing model for hippocampus segmentation was for MRI T2 images with weighted binary cross entropy penalizing false positives twice as much as false negatives. The best model yielded a Dice score of 0.491. Neither precision nor recall scores are high, indicating a high rate of false positives and false negatives, giving rise to the middling dice score. All T2 models got a much higher Dice score than other models. Recall and precision are more even on the best model, with recall being slightly higher. However, both values are still relatively low, indicating that the model misses a lot on both positive and negative values. Figure 4.16 visualizes all scores for hippocampus segmentation.

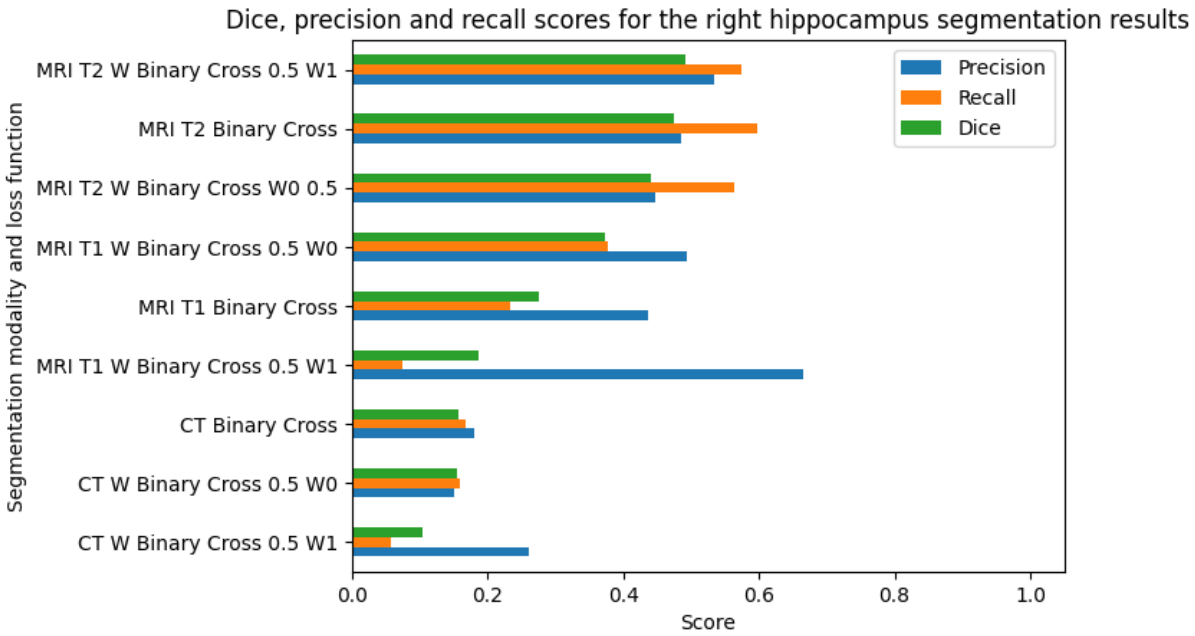

Figure 4.16: Graph illustrating the Dice, precision and recall scores for segmentation results of the right hippocampus.

4.5 Segmentation with more feature maps

Segmentation performed by the network with additional feature maps per layer did not significantly increase Dice scores for segmentation. The training time was increased 5-fold, and the number of trainable parameters increased 10-fold, yet no significant increase in prediction values was registered. The Dice, precision, and recall scores for this network are displayed in Table 4-13: Dice, precision, and recall scores for segmentations performed by the deeper network. A single modality and loss function was used to check the impact of adding more feature maps.

Table 4-13: Dice, precision, and recall scores for segmentations performed by the deeper network. A single modality and loss function was used to check the impact of adding more feature maps.

Structure	Dice	Precision	Recall
Brainstem MRI T2 BC	0.741	0.795	0.734
Hippocampus R WBCW0 0.5	0.485	0.460	0.580
Brain CT Dice	0.980	0.986	0.975
Papez MRI T2 WBCW1 0.5	0.501	0.558	0.63

5 Discussion

The goal of this master thesis was to investigate how well a neural network would perform auto segmentation of brain structures without explicit tuning for individual structures as the complexity and size varies. Segmentation of the brain achieved excellent dice scores using CT images. Brainstem segmentation achieved decent dice scores using T2 MRI images, while Papez Circuit and Hippocampus segmentation got poor dice scores using T2 MRI images. A general-purpose convolutional neural network with the U-net architecture was implemented using Keras and TensorFlow. Three different modalities were used to train separate networks, and three different loss functions were used to train the networks on separate occasions. This chapter discusses the results from chapter 4, potential sources of error or limitations of the methodology, and how the results from this thesis can be used for future work.

5.1 Image modality and model comparison

The modality that provided the best segmentation results varied depending on the structure. There are considerable differences between the three modalities and even between the T1 and T2 weighted MRI scans. As presented in section 2.3, CT images depend on the electron density for image contrast, whereas MRI depends on the T1 and T2 values of the tissue as presented in section 2.4.

5.1.1 Brain segmentation results

For the whole brain, using CT as the modality for segmentation was the best option. The cranium is easily defined in CT images due to the higher electron density in bone compared to tissue as discussed in section 2.3. This gave rise to large contrast between the cranium and brain, giving the network great features to train on. The contrast difference between different brain tissues is also low for CT images. This is caused by the electron density being relatively even for different brain tissues, resulting in the pixel values being closer, helping the network interpret it as a single structure. It could also be because the ground-truth segmentation was done on a CT scan, and thus it might line up better with the CT than the MRI.

Bone does not give much signal on regular T1 or T2 MRI images and will appear darker on images. Thus, MRI does not give the highlighted difference between brain tissue and cranium as CT gives. Interestingly, segmentation on MRI T2 weighted images performed better than MRI T1, considering that the cohort size is considerably smaller for T2. This might be due to the varying recording mechanisms in the T1 dataset, and the mix of both contrast and no contrast images.

The model gave great Dice scores for segmentation of the full brain. Precision and recall scores remain high for all models, indicating that when the model predicted that a pixel is part of the structure is correct almost always. The full brain was the largest structure, and therefore also the easiest structure to perform segmentation on. The size of the structure and the were potentially the highest contributing factors to the high dice score.

Full brain segmentation is already being done by several commercial actors, and is available for purchase and clinical integration, such as Limbus AI [70] and Raystation integration [71].

5.1.2 Brainstem segmentation results

The best model for brainstem segmentation was trained on T2 MRI images. The T2 MRI dataset was as mentioned much smaller than both the T1 MRI and CT dataset but performed significantly better than both for brainstem segmentation. That T2 performed best might be due to more similar recording sequences for the T2 dataset compared to the MRI T1 set. From the limited information available for the T1 dataset, there appears to be a larger difference in the acquisition method for T1. That T2 MRI performed better than CT images was probably due to the higher contrast MRI images provide for soft tissue than the contrast CT provides for soft tissue. CT also achieved higher dices scores than T1 MRI images when segmenting the brainstem, but neither CT nor T1 MRI models performed high-quality segmentations.

State-of-the-art auto segmentation results for brainstem segmentation have reached a Dice score of 0.916 using a GAN network in addition to DenseNet and a CNN on CT images and 0.35Tesla MRI images [72]. Another model achieved a Dice score of 0.88 using a custom

architecture with multiple CNN networks trained on CT images [73]. Both models achieve higher dice score than the models in this thesis, probably due to the much more extensive preprocessing and using multiple networks to correct errors. A dice score of 0.732 is decent, but the contour would need additional contouring by a professional for any use in clinical settings. Brainstem segmentation is also done by commercial actors, such as Limbus AI [70] and Raystation integration [71].

5.1.3 Papez Circuit segmentation results

The Papez Circuit is the most complex structure in this study, and it is not a common structure to delineate. Recent studies have shown that radiation exposure to the Papez Circuit during radiation therapy give rise to side effects. This can lead to the structure becoming a larger part of radiation therapy planning, and thus the need for delineation of the structure.

The model trained on T2 MRI images gave the highest dice score for the Papez circuit. That T2 MRI images perform better for this structure as well gives a stronger case that similar recording method is more important than the number of training samples if the training samples vary greatly, as is the case for the T1 data.

There were not found any other attempts at auto segmentation of the Papez Circuit for comparison, so there is limited information as to how good the segmentation results are. A Dice score of 0.515 is not a high score, but without anything to compare it to, it is difficult to measure the success of the segmentation dice score.

5.1.4 Hippocampus segmentation results

Hippocampus is the smallest structure segmented in this thesis. The model trained on T2 MRI images achieve the best score with the best being 0.491. None of the models achieved great

results, indicating that more preprocessing might be necessary for successful segmentation of this structure. A larger dataset would also help the model, given the relatively few slices with segmentation of the structure for ground truth.

An article reviewing different networks for hippocampus segmentation achieved a Dice score of 0.737 as the best result using an AG-3D Resnet model. Other results from the study included scores ranging from 0.654 to 0.727, with models such as 3D U-net, modified U-net, and a regular 3D Resnet by Porter et al [74]. Porter e t al. had a patient group of 390 patients. It also included extensive preprocessing including cropping and generation of extra image channels. The Dice score of 0.491 achieved in this thesis is far away from Porter et al. highest Dice score of 0.727. The preprocessing and much larger data foundation is probably the largest source for the difference.

5.2 Network architecture, parameters, and input

5.2.1 Network architecture

When choosing network parameters, there are always several things to consider. One of the main concerns in Deep learning is time. It took approximately 2 hours to run through 100 epochs for one imaging modality with the settings used in this network. The network used in this thesis has fewer feature maps than the original U-net presented by Ronneberger et al. [54]. Fewer feature maps result in the number of trainable parameters being much lower than in the original paper. However, the results from section 4.5, where the number of feature maps equals that of the original U-net [54], there is only a slight improvement in Dice score compared to the network with fewer feature maps.

While the U-net architecture is a great foundation when performing image segmentation, other network architectures could also have been used. Deep CNN models are commonly used [75]–[77], and others have also used GAN architecture [78]. Another approach could have

been to use a 3D U-net instead of the 2D one used in this thesis. 3D U-nets have successfully segmented the left hippocampus with a dice score of 0.707 [74].

5.2.2 Loss functions and metrics

The impact of loss functions on the network seems to have a smaller impact than the modality used. The differences between the loss functions for each structure only give a maximum difference of 0.05-0.1 in Dice score depending on the structure and modality. The difference between the best modality can be upwards of a 0.2 difference in Dice score. The largest impact of loss functions appears when segmenting the smallest structures, where the dataset is the most unbalanced. The model did not manage to train with Dice loss for several of the smaller structures. This could be caused by the large unbalance in the dataset for the smaller structures, with the small batch size not giving enough relevant information per batch.

The metric chosen to measure performance of the model was Dice score. This is one of the most standard methods used for image segmentation, and it was therefore chosen to have results comparable with other publications. Dice score performs well on Average Hausdorff distance is another common metric that measures the distance between two point sets. Average Hausdorff is less reliant upon the size of the segment and could also have been used as a performance measure in this thesis. A recent study found an error when using Average Hausdorff distance as a performance metric for segmentation when the size of the segment varied [79].

5.2.3 Train, test, and split decisions

Splitting of data into train validation and testing was done manually. While it is best practice to do this randomly, the decision was made to hard code the test and validation patients due to the large variance in the dataset. The large discrepancy between the three modalities would have made comparison hard. In addition, the three cohorts that are present in the dataset would need to be present in both training and testing. In an early test build of the code, one patient group was used for both validation and testing, resulting in higher Dice scores for the test set than presented in section 4. To ensure comparable results between the segmentation

performed with the different modalities and the representation of the three groups of patients in validation and testing, the three patients used for validation were always the same, and the three patients used for testing were always the same.

The most common method for train test split in other comparable articles is a variation of Cross-validation or a train/test split instead of a train/validation/test split [10]. A cross validation approach to splitting the data might have yielded a different result for this thesis due to the number of patients being on the lower end of the scale. However, there would be no guarantee that all cohorts were represented in each iteration of the cross-validation process.

5.3 Dataset challenges

The dataset consisted of 30 patients, where 20-29 were used depending on the imaging modality and structure selected. While the number of patients provided enough information to get some adequate segmentations, it was probably on the lower side to get segmentations of smaller structures without additional preprocessing of the data and specific tuning of the network. Similar studies had patient groups ranging from 9 to 1021, with 18/27 studies patient groups ranging between 40 – 200 [10].

Another potential source of error introduced with the dataset is that it is not homogeneous. With the scan length varying between 162 and 458 for the T2 MRI scans, selecting the slices containing relevant information is more complicated. The varying scan length results in significant noise for the network, decreasing the effectiveness of the segmentations despite the methods used to reduce the amount of images with little to no value. More thorough data preparation of the different modalities might have yielded better results. It could also be considered only to include the image slices where segmentation was also present. However, this would defeat the purpose of an easy-to-use, all-purpose network and potentially reduce the networks' ability to correctly label true negatives.

From the studies reviewed in [10], 6 out of 27 studies had 30 patients or below. Two of the six studies explicitly mention the scan length being the same for all scan sequences included.[77], [80] Most also mention the exact sequence settings used to retrieve the scan sequence, whether voltage for CT scans [73] or specific TE and TR times for MRI sequences [75].

5.3.1 CT noise factors

The length of the CT scans, as mentioned above, is a noise problem. In addition, several CT scans had the bed where the patients rested included in the scan, while some did not. The CT machine's specification is also unknown, and the patients did probably not have the CT done on the same scanner. The value for bone is also very high for CT, ref Table 2-3, which means that the normalized pixel values for the brain itself will be much closer in value due to the high value for the cranium.

5.3.2 MRI noise factors

Another potential source of error is that some of the MRI scans were done with contrast fluid. The contrast fluid used is not known, but is assumed to be gadolinium, as it is by far the most common one. [37] This contrast fluid enhances the signal of T1 weighted images. The contrast adds extra complexity to training due to the difference in intensity of the pixels from scan sequences acquired with contrast fluid compared to those acquired without. This potentially confuses the network when trying to tie pixel values to structure location because of the difference in signal strength.

Normalization of each scan was done to try and prevent this. The MRI values were normalized to [0,1] based upon the max value in the entire scan sequence. While reducing the values to the desired interval can also cause extra noise. If there is one voxel with a much higher pixel value relative to the rest of the scan sequence, the rest of the values will be lowered much more than they should. A better way might have been to define an area for each scan, use the mean value from this area, and normalize against it.

It is also unknown what settings were used to acquire the different scans, including no knowledge of TE and TR time for the different scan sequences. Some sequences have general information about which recording method is used. However, more specifics would be needed to decide how to include or exclude specific scan sequences or split the data differently.

5.3.3 Delineation noise factors

When a professional, either oncologist, radiologist, or radiographer, delineates structures for RT plans, the bias of the individual doing the delineation is important to consider, especially if the same individual does not do it. The data used in this thesis has all delineation done by the same person, which should produce a more consistent ground truth than if it were multiple people. However, it also leans into the bias this person has when delineating. A study investigating Interobserver variability found there are still discrepancies between delineation, even when following segmentation standards such as ICG [81]. The result of the segmentation performed by the network is potentially prone to the bias of the person who did the delineation. Delineation of structures done by several professionals on the same dataset could be one way to avoid this bias, but the time investment would be considerable.

5.4 Conclusion and future work

In the thesis a neural network with the U-net architecture was implemented as a model for auto-segmenting several brain structures without major preprocessing. The brain, brainstem, Papez Circuit, and hippocampus were segmented using various models trained on different imaging modalities, comparing the results of MRI T1, MRI T2 and CT images and the impact of loss functions in training. The study suggests that the model architecture can perform segmentation, given the great results from the full brain segmentation. The models trained with this architecture struggle with segmentation of smaller structures. This may be due to the dataset, and not the network architecture, as the importance of similar scan sequences for imaging modalities has been highlighted by the results from T1 compared to T2 data. An investigation into whether a more thorough preprocessing of the same dataset would yield better results should be conducted in future work. This is especially relevant for the Papez

Circuit, as no previous attempts at auto segmentation have been published for this structure. Future work could also compare the performance of this network architecture to other network architectures.

References

[1] “Cancer in Norway 2020 - Cancer incidence, mortality, survival and prevalence in Norway.,” Kreftregisteret, Oslo, Norway, 2021. Accessed: Feb. 10, 2022. [Online]. Available: https://www.kreftregisteret.no/Generelt/Rapporter/Cancer-in-Norway/cancer-in-norway-2020/

[2] L. M. Åsli *et al.*, “Utilization of Radiation Therapy in Norway After the Implementation of The National Cancer Plan—A National, Population-Based Study,” *Int. J. Radiat. Oncol.*, vol. 90, no. 3, pp. 707–714, Nov. 2014, doi: 10.1016/j.ijrobp.2014.06.059.

[3] M. T. Makale, C. R. McDonald, J. Hattangadi-Gluth, and S. Kesari, “Brain irradiation and long-term cognitive disability: Current concepts,” *Nat. Rev. Neurol.*, vol. 13, no. 1, pp. 52–64, Jan. 2017, doi: 10.1038/nrneurol.2016.185.

[4] G. T. Armstrong *et al.*, “Region-specific radiotherapy and neuropsychological outcomes in adult survivors of childhood CNS malignancies,” *Neuro-Oncol.*, vol. 12, no. 11, pp. 1173–1186, Nov. 2010, doi: 10.1093/neuonc/noq104.

[5] L. Toussaint *et al.*, “Radiation doses to brain substructures associated with cognition in radiotherapy of pediatric brain tumors,” *Acta Oncol. Stockh. Swed.*, vol. 58, no. 10, pp. 1457–1462, Oct. 2019, doi: 10.1080/0284186X.2019.1629014.

[6] A. H. Zureick *et al.*, “Left hippocampal dosimetry correlates with visual and verbal memory outcomes in survivors of pediatric brain tumors,” *Cancer*, vol. 124, no. 10, pp. 2238–2245, May 2018, doi: 10.1002/cncr.31143.

[7] E. Doger de Speville *et al.*, “Relationships between Regional Radiation Doses and Cognitive Decline in Children Treated with Cranio-Spinal Irradiation for Posterior Fossa Tumors,” *Front. Oncol.*, vol. 7, 2017, Accessed: May 18, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fonc.2017.00166

[8] G. Sharp *et al.*, “Vision 20/20: Perspectives on automated image segmentation for radiotherapy,” *Med. Phys.*, vol. 41, no. 5, p. 050902, 2014, doi: 10.1118/1.4871620.

[9] H. Li, A. Zhygallo, and B. Menze, “Automatic Brain Structures Segmentation Using Deep Residual Dilated U-Net,” *ArXiv181104312 Cs*, Nov. 2018, Accessed: May 11, 2022. [Online]. Available: http://arxiv.org/abs/1811.04312

[10] G. Samarasinghe *et al.*, “Deep learning for segmentation in radiation therapy planning: a review,” *J. Med. Imaging Radiat. Oncol.*, vol. 65, no. 5, pp. 578–595, 2021, doi: 10.1111/1754-9485.13286.

[11] U. F. O. Themes, “4: Radiation Interactions with Tissue,” *Radiology Key*, Jan. 08, 2016. https://radiologykey.com/4-radiation-interactions-with-tissue/ (accessed Apr. 15, 2022).

[12] Kieranmaher, *English: The Photoelectric Effect - electron ejection (a) and fluorescent X-ray emission (b).* 2001. Accessed: May 31, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:PhotoelectricEffect.jpg

[13] Nadine Barrie Smith and Andrew Webb, *Introduction to Medical Imaging*. Cambridge University Press, 2011.

[14] Kieranmaher, *English: The Compton Effect - (a) photon scattering and electron ejection leading to (b) an ionized atom.* 2001. Accessed: May 31, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:ComptonEffect.jpg

[15] W. R. Leo, *Techniques for Nuclear and Particle Physics Experiments: A How-To Approach*. Berlin, Heidelberg, GERMANY: Springer Berlin / Heidelberg, 1994. Accessed: Apr. 17, 2022. [Online]. Available: http://ebookcentral.proquest.com/lib/bergen-ebooks/detail.action?docID=3089885

[16] E. B. Podgoršak, “Interactions of Charged Particles with Matter,” in *Radiation Physics for Medical Physicists*, E. B. Podgorsak, Ed. Cham: Springer International Publishing, 2016, pp. 229–276. doi: 10.1007/978-3-319-25382-4_6.

[17] H. E. S. Pettersen, “A digital tracking calorimeter for proton computed tomography,” 2018.

[18] J. P. G. Faiz M. Khan, *Khan’s The Physics of Radiation Therapy*, 5th Edition. Wolters Kluwer Health, 2014.

[19] Thomas E. Johnson, *Introduction to health physics*, 5th edition. McGraw-Hill Inc.,US, 2017.

[20] L. Cerrito, *Radiation and Detectors*. Cham: Springer International Publishing, 2017. doi: 10.1007/978-3-319-53181-6.

[21] S. Mehta, V. Suhag, M. Semwal, and N. Sharma, “Radiotherapy: Basic Concepts and Recent Advances,” *Med. J. Armed Forces India*, vol. 66, no. 2, pp. 158–162, Apr. 2010, doi: 10.1016/S0377-1237(10)80132-7.

[22] N.-S. Hegemann, M. Guckenberger, C. Belka, U. Ganswindt, F. Manapov, and M. Li, “Hypofractionated radiotherapy for prostate cancer,” *Radiat. Oncol.*, vol. 9, no. 1, p. 275, Dec. 2014, doi: 10.1186/s13014-014-0275-6.

[23] M. Lederman, “The early history of radiotherapy: 1895–1939,” *Int. J. Radiat. Oncol.*, vol. 7, no. 5, pp. 639–648, May 1981, doi: 10.1016/0360-3016(81)90379-5.

[24] R. R. Wilson, “Radiological Use of Fast Protons,” *Radiology*, vol. 47, no. 5, pp. 487–491, Nov. 1946, doi: 10.1148/47.5.487.

[25] R. Mohan and D. Grosshans, “Proton therapy – Present and future,” *Adv. Drug Deliv. Rev.*, vol. 109, pp. 26–44, Jan. 2017, doi: 10.1016/j.addr.2016.11.006.

[26] HPaul, *English: Depth dose of electrons, x rays or protons entering human tissue for the purpose of cancer treatment*. 2009. Accessed: May 16, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Depth_Dose_Curves.jpg

[27] H. Wang and P. A. Yushkevich, “Multi-Atlas Segmentation without Registration: A Supervoxel-based Approach,” *Med. Image Comput. Comput.-Assist. Interv. MICCAI Int. Conf. Med. Image Comput. Comput.-Assist. Interv.*, vol. 16, no. 0 3, pp. 535–542, 2013.

[28] P. Coupé, J. V. Manjón, V. Fonov, J. Pruessner, M. Robles, and D. L. Collins, “Patch-based segmentation using expert priors: Application to hippocampus and ventricle segmentation,” *NeuroImage*, vol. 54, no. 2, pp. 940–954, Jan. 2011, doi: 10.1016/j.neuroimage.2010.09.018.

[29] N. Kim, J. S. Chang, Y. B. Kim, and J. S. Kim, “Atlas-based auto-segmentation for postoperative radiotherapy planning in endometrial and cervical cancers,” *Radiat. Oncol.*, vol. 15, no. 1, p. 106, May 2020, doi: 10.1186/s13014-020-01562-y.

[30] K. Harrison, H. Pullen, C. Welsh, O. Oktay, J. Alvarez-Valle, and R. Jena, “Machine Learning for Auto-Segmentation in Radiotherapy Planning,” *Clin. Oncol.*, vol. 34, no. 2, pp. 74–88, Feb. 2022, doi: 10.1016/j.clon.2021.12.003.

[31] C. A. Carlsson and G. A. Carlsson, “Basic physics of X-ray imaging,” p. 31.

[32] U. F. O. Themes, “X-ray Production, Tubes, and Generators,” *Radiology Key*, May 16, 2021. https://radiologykey.com/x-ray-production-tubes-and-generators/ (accessed May 03, 2022).

[33] R. Schofield *et al.*, “Image reconstruction: Part 1 – understanding filtered back projection, noise and image acquisition,” *J. Cardiovasc. Comput. Tomogr.*, vol. 14, no. 3, pp. 219–225, Mai 2020, doi: 10.1016/j.jcct.2019.04.008.

[34] Kostmo, *English: Sinogram formed from , taken at 50 equidistant angles spanning from 0 to 180 degrees.* 2008. Accessed: May 15, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Sinogram_Result_-_Two_Squares_Phantom.png

[35] R. Kramme, K.-P. Hoffmann, and R. S. Pozos, Eds., *Springer Handbook of Medical Technology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-540-74658-4.

[36] Eli Renate Grüner, “Compendium PHYS212 Medical Physics and Technology.” Department of Physics and Technology - Faculty of Mathematics and Natural Sciences - University of bergen, 2012.

[37] A. Bjørnerud, “The Physics of Magnetic Resonance Imaging,” *Dep. Phys. Univ. Oslo*, p. 183, Mar. 2006.

[38] S. A. Bini, “Artificial Intelligence, Machine Learning, Deep Learning, and Cognitive Computing: What Do These Terms Mean and How Will They Impact Health Care?,” *J. Arthroplasty*, vol. 33, no. 8, pp. 2358–2361, Aug. 2018, doi: 10.1016/j.arth.2018.02.067.

[39] Y. Mintz and R. Brodie, “Introduction to artificial intelligence in medicine,” *Minim. Invasive Ther. Allied Technol.*, vol. 28, no. 2, pp. 73–81, Mar. 2019, doi: 10.1080/13645706.2019.1575882.

[40] N. Ketkar, *Deep Learning with Python*. Berkeley, CA: Apress, 2017. doi: 10.1007/978-1-4842-2766-4.

[41] T. Qin, *Dual Learning*. Singapore: Springer Singapore, 2020. doi: 10.1007/978-981-15-8884-6.

[42] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020, doi: 10.1016/j.neucom.2020.07.061.

[43] Z.-H. Zhou, *Machine Learning*. Singapore: Springer Singapore, 2021. doi: 10.1007/978-981-15-1967-3.

[44] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *ArXiv160307285 Cs Stat*, Jan. 2018, Accessed: Apr. 18, 2022. [Online]. Available: http://arxiv.org/abs/1603.07285

[45] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” *ArXiv151108458 Cs*, Dec. 2015, Accessed: Mar. 01, 2022. [Online]. Available: http://arxiv.org/abs/1511.08458

[46] S. Sharma, S. Sharma, U. Scholar, and A. Athaiya, “ACTIVATION FUNCTIONS IN NEURAL NETWORKS,” vol. 4, no. 12, p. 7, 2020.

[47] “Hyperbolic Tangent Function - an overview | ScienceDirect Topics.” https://www.sciencedirect.com/topics/mathematics/hyperbolic-tangent-function (accessed Apr. 26, 2022).

[48] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *ArXiv14126980 Cs*, Jan. 2017, Accessed: May 07, 2022. [Online]. Available: http://arxiv.org/abs/1412.6980

[49] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2012, pp. 3642–3649. doi: 10.1109/CVPR.2012.6248110.

[50] E. Grefenstette, P. Blunsom, N. de Freitas, and K. M. Hermann, “A Deep Architecture for Semantic Parsing,” *ArXiv14047296 Cs*, Apr. 2014, Accessed: Apr. 26, 2022. [Online]. Available: http://arxiv.org/abs/1404.7296

[51] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, “Deep Learning for Identifying Metastatic Breast Cancer,” *ArXiv160605718 Cs Q-Bio*, Jun. 2016, Accessed: Apr. 26, 2022. [Online]. Available: http://arxiv.org/abs/1606.05718

[52] S. Sarraf and G. Tofighi, “Classification of Alzheimer’s Disease using fMRI Data and Deep Learning Convolutional Neural Networks,” *ArXiv160308631 Cs*, Mar. 2016, Accessed: Apr. 26, 2022. [Online]. Available: http://arxiv.org/abs/1603.08631

[53] M. Sajjad, S. Khan, K. Muhammad, W. Wu, A. Ullah, and S. W. Baik, “Multi-grade brain tumor classification using deep CNN with extensive data augmentation,” *J. Comput. Sci.*, vol. 30, pp. 174–182, Jan. 2019, doi: 10.1016/j.jocs.2018.12.003.

[54] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *ArXiv150504597 Cs*, May 2015, Accessed: Mar. 01, 2022. [Online]. Available: http://arxiv.org/abs/1505.04597

[55] J. Causey *et al.*, “An Ensemble of U-Net Models for Kidney Tumor Segmentation with CT images,” *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. PP, Jun. 2021, doi: 10.1109/TCBB.2021.3085608.

[56] X. Feng, N. J. Tustison, S. H. Patel, and C. H. Meyer, “Brain Tumor Segmentation Using an Ensemble of 3D U-Nets and Overall Survival Prediction Using Radiomic Features,” *Front. Comput. Neurosci.*, vol. 14, p. 25, 2020, doi: 10.3389/fncom.2020.00025.

[57] K. Yue, B. Zou, Z. Chen, and Q. Liu, “Retinal vessel segmentation using dense U-net with multiscale inputs,” *J. Med. Imaging Bellingham Wash*, vol. 6, no. 3, p. 034004, Jul. 2019, doi: 10.1117/1.JMI.6.3.034004.

[58] P. J. L. illustrator medical, *English: Skull and brain normal human diagram.* 2006. Accessed: May 26, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Skull_and_brain_normal_human.svg

[59] J. G. Bjålie, E. Haug, O. Sand, and Ø. V. Sjaastad, *Menneskekroppen*, 2. Utgave, 4. opplag. Gyldendal Norsk Forlag AS, 2011.

[60] I. are generated by L. S. Databases, *English: brainstem.* 2009. Accessed: May 25, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Brainstem.png

[61] “Analysis of the anatomy of the Papez circuit and adjoining limbic system by fiber dissection techniques | Elsevier Enhanced Reader.” https://reader.elsevier.com/reader/sd/pii/S0967586811005042?token=49A1EC94BC72C3B8DD5B2B41FBEA7F6F77F17DADFB10229E6797D8E20BBC217C9A737CFD3D701BC9F24D96E136143A80&originRegion=eu-west-1&originCreation=20220525141540 (accessed May 25, 2022).

[62] “Neural_systems_proposed_to_process_emotion.png (945×461).” https://upload.wikimedia.org/wikipedia/commons/8/86/Neural_systems_proposed_to_process_emotion.png (accessed May 25, 2022).

[63] J. K. S. Jansen and P. Holck, “hippocampus,” *Store medisinske leksikon*. Nov. 07, 2021. Accessed: May 25, 2022. [Online]. Available: http://sml.snl.no/hippocampus

[64] I. are generated by L. S. Databases, *English: hippocampus*. 2009. Accessed: May 25, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Hippocampus_image.png

[65] S. van der Walt *et al.*, “scikit-image: image processing in Python,” *PeerJ*, vol. 2, p. e453, Jun. 2014, doi: 10.7717/peerj.453.

[66] T. Developers, *TensorFlow*. Zenodo, 2022. doi: 10.5281/zenodo.5949169.

[67] Fran Chollet and others, *Keras*. 2015.

[68] P. Gavrikov, *visualkeras for Keras / TensorFlow*. 2022. Accessed: May 07, 2022. [Online]. Available: https://github.com/paulgavrikov/visualkeras

[69] S. Jadon, “A survey of loss functions for semantic segmentation,” *2020 IEEE Conf. Comput. Intell. Bioinforma. Comput. Biol. CIBCB*, pp. 1–7, Oct. 2020, doi: 10.1109/CIBCB48159.2020.9277638.

[70] “Limbus AI - Automatic Contouring for Radiation Therapy,” *Limbus AI*. https://limbus.ai/ (accessed May 31, 2022).

[71] “Machine Learning in RayStation,” *RaySearch Laboratories*. https://www.raysearchlabs.com/products/raystation/machine-learning-in-raystation/ (accessed May 31, 2022).

[72] N. Tong, S. Gou, S. Yang, M. Cao, and K. Sheng, “Shape constrained fully convolutional DenseNet with adversarial training for multiorgan segmentation on head and neck CT and low-field MR images,” *Med. Phys.*, vol. 46, no. 6, pp. 2669–2682, 2019, doi: 10.1002/mp.13553.

[73] L. Vandewinckele *et al.*, “Segmentation of head-and-neck organs-at-risk in longitudinal CT scans combining deformable registrations and convolutional neural networks,” *Comput. Methods Biomech. Biomed. Eng. Imaging Vis.*, vol. 8, no. 5, pp. 519–528, Sep. 2020, doi: 10.1080/21681163.2019.1673824.

[74] E. Porter *et al.*, “Hippocampus segmentation on noncontrast CT using deep learning,” *Med. Phys.*, vol. 47, no. 7, pp. 2950–2961, 2020, doi: 10.1002/mp.14098.

[75] L. Bielak *et al.*, “Automatic Tumor Segmentation With a Convolutional Neural Network in Multiparametric MRI: Influence of Distortion Correction,” *Tomography*, vol. 5, no. 3, pp. 292–299, Sep. 2019, doi: 10.18383/j.tom.2019.00010.

[76] L. Zhao, Z. Lu, J. Jiang, Y. Zhou, Y. Wu, and Q. Feng, “Automatic Nasopharyngeal Carcinoma Segmentation Using Fully Convolutional Networks with Auxiliary Paths on Dual-Modality PET-CT Images,” *J. Digit. Imaging*, vol. 32, no. 3, pp. 462–470, Jun. 2019, doi: 10.1007/s10278-018-00173-0.

[77] Y. Wang *et al.*, “Automatic Tumor Segmentation with Deep Convolutional Neural Networks for Radiotherapy Applications,” *Neural Process. Lett.*, vol. 48, no. 3, pp. 1323–1334, Dec. 2018, doi: 10.1007/s11063-017-9759-3.

[78] M. E. Laino, P. Cancian, L. S. Politi, M. G. Della Porta, L. Saba, and V. Savevski, “Generative Adversarial Networks in Brain Imaging: A Narrative Review,” *J. Imaging*, vol. 8, no. 4, Art. no. 4, Apr. 2022, doi: 10.3390/jimaging8040083.

[79] O. U. Aydin *et al.*, “On the usage of average Hausdorff distance for segmentation performance assessment: hidden error when used for ranking,” *Eur. Radiol. Exp.*, vol. 5, p. 4, Jan. 2021, doi: 10.1186/s41747-020-00200-2.

[80] Z. Ma, X. Wu, Q. Song, Y. Luo, Y. Wang, and J. Zhou, “Automated nasopharyngeal carcinoma segmentation in magnetic resonance images by combination of convolutional neural networks and graph cut,” *Exp. Ther. Med.*, vol. 16, no. 3, pp. 2511–2521, Sep. 2018, doi: 10.3892/etm.2018.6478.

[81] J. van der Veen, A. Gulyban, S. Willems, F. Maes, and S. Nuyts, “Interobserver variability in organ at risk delineation in head and neck cancer,” *Radiat. Oncol.*, vol. 16, no. 1, p. 120, Jun. 2021, doi: 10.1186/s13014-020-01677-2.

Appendix 1

Appendix one shown the entire code used for data extraction, training of the network and plotting of the segmentation results.

```
import numpy as np
import pydicom as dicom
from skimage.draw import polygon
from tensorflow import keras
import keras.backend as K
import tensorflow as tf
import matplotlib.pyplot as plt
import os
from skimage.transform import resize
import re
from matplotlib import colors
import matplotlib.patches as mpatches



#Class for a single patient
class Patient:

    def __init__(self, patient_id, mri_t1_data=None, mri_t2_data=None,
ct_data=None, structure=None):
        self.patient_id = patient_id
        self.mri_t1_data = mri_t1_data
        self.mri_t2_data = mri_t2_data
        self.ct_data = ct_data
        self.structure = structure

    # Input path for MRI T1 data, stores the pixel data in self.MRI_T2_data
    def get_mri_t1_files(self):
        # Input path for MRI T2 data, stores the pixel data in
self.MRI_T2_data
        dicom_dict = {}
        dicom_list = []
        sorting_dict = {}
        modality = "T1"

        # Looping over current working directory
        for dirName, subdirList, fileList in os.walk(os.getcwd()):
            # If the current directory matches the patient ID, go into this
directory
            if os.path.basename(os.path.normpath(dirName)) ==
str(self.patient_id):
                new_path = dirName
                # Loop over all directories in this subdirectory
                for new_dirName, new_subdirList, new_fileList in
os.walk(new_path):

                    # If given modality is in the directory name, append
all files in that directory to a dict
                    if str(modality) in
os.path.basename(os.path.normpath(new_dirName)):
```

```
                        for filename in new_fileList:
                            dicom_list.append(os.path.join(new_dirName,
filename))
                        dicom_dict[new_dirName] = dicom_list
                        # If one scan of the modality is found, stop
searching for files
                        if len(dicom_dict.keys()) >= 1:
                            break

        # looping over files in list created in previous step
        # and places the pixel values in a dict with the location nr as key
        for key, value in dicom_dict.items():
            for file in value:
                ds = dicom.dcmread(str(file))
                slice_nr = ds.SliceLocation
                sorting_dict[slice_nr] = ds

            # Sort the dict by location
            sorted_dict = {k: v for k, v in sorted(sorting_dict.items(),
key=lambda item: -item[0])}
            values = sorted_dict.values()

            return list(values)

    def get_mri_t2_files(self):
        # Input path for MRI T2 data, stores the pixel data in
self.MRI_T2_data
        dicom_dict = {}
        dicom_list = []
        sorting_dict = {}
        modality = "T2"

        # Looping over current working directory
        for dirName, subdirList, fileList in os.walk(os.getcwd()):

            # If the current directory matches the patient ID, go into this
directory
            if os.path.basename(os.path.normpath(dirName)) ==
str(self.patient_id):
                new_path = dirName

                # Loop over all directories in this subdirectory
                for new_dirName, new_subdirList, new_fileList in
os.walk(new_path):

                    # If given modality is in the directory name, append
all files in that directory to a dict
                    if str(modality) in
os.path.basename(os.path.normpath(new_dirName)):

                        for filename in new_fileList:
                            # dicom_list = []
                            dicom_list.append(os.path.join(new_dirName,
filename))
                        dicom_dict[new_dirName] = dicom_list

                        # If one scan of the modality is found, stop
searching for files
                        if len(dicom_dict.keys()) >= 1:
                            break
```

```
        # looping over files in list created in previous step
        # and places the pixel values in a dict with the location nr as key
        for key, value in dicom_dict.items():
            for file in value:
                ds = dicom.dcmread(str(file))
                slice_nr = ds.SliceLocation
                sorting_dict[slice_nr] = ds

            # Sorting the scan by slice location
            sorted_dict = {k: v for k, v in sorted(sorting_dict.items(),
key=lambda item: -item[0])}
            values = sorted_dict.values()

            return list(values)

    # Input path for CT data, stores the pixel data in CT_data
    def get_ct_files(self):
        dicom_dict = {}
        dicom_list = []
        sorting_dict = {}
        modality = "CT"
        for dirName, subdirList, fileList in os.walk(os.getcwd()):

            if os.path.basename(os.path.normpath(dirName)) ==
str(self.patient_id):
                new_path = dirName

                for new_dirName, new_subdirList, new_fileList in
os.walk(new_path):

                    if str(modality) in
os.path.basename(os.path.normpath(new_dirName)):

                        for filename in new_fileList:
                            dicom_list.append(os.path.join(new_dirName,
filename))
                        dicom_dict[new_dirName] = dicom_list

                        if len(dicom_dict.keys()) >= 1:
                            break

        # looping over files in list created in previous step
        # and places the pixel values in a dict with the location nr as key
        for key, value in dicom_dict.items():
            for file in value:
                ds = dicom.dcmread(str(file))
                slice_nr = ds.SliceLocation
                sorting_dict[slice_nr] = ds

            sorted_dict = {k: v for k, v in sorted(sorting_dict.items(),
key=lambda item: -item[0])}
            values = sorted_dict.values()

            return list(values)


    #Creates a structure using the Structure class, and returns the pixel
data for the structure
    def get_structure(self, structure_name):
        struct = Structure(self.patient_id, structure_name)
        struct.get_structure_id()
```

```
        struct_dict = struct.find_structure2()
        self.structure = struct_dict
        return struct_dict

    #Extracts the position for each of the slices in a dataset, and returns
a dict with {position : data}
    def get_position_z(self, data):
        ds_dict = {}
        for ds in data:
            ds_dict[ds.ImagePositionPatient[2]] = ds
        return ds_dict

    #Functuon to get the pixels and masks for a given structure and
modality
    def get_pixels_and_masks(self, modality, structure_name):
        scan_pixels = {}
        struct_masks = {}
        data = 0

        #Checking which modality to use
        if modality == "CT":
            data = self.get_ct_files()
        elif modality == "MRI_T1":
            data = self.get_mri_t1_files()
        elif modality == "MRI_T2":
            data = self.get_mri_t2_files()
        else:
            print("Not valid input, give CT, MRI_T1 or MRI_T2 as input")

        #If no data is found, there is no scan with given modality type,
skipping patient
        if not (data) :
            print("No scan with this modality for patient " +
str(self.patient_id))

        else:
            #Getting the structure and dataset
            data_cut = data[25:len(data)-65]
            scan = self.get_position_z(data_cut)
            structure = self.get_structure(structure_name)

            #Checking that the shape of the scan is even in both directions
            ds_test = list(scan.values())[0]
            if ds_test.pixel_array.shape[0] !=
ds_test.pixel_array.shape[1]:
                print("Image does not have the correct shape, skipping
patient", str(self.patient_id))

            #Checking that there is a structure for this patient
            elif structure == None:
                print("No structure for this modality, skipping patient",
str(self.patient_id))

            else:

                #Looping over all the slices
                for key, value in scan.items():

                    #If the position for a slice matches that of a position
in the structure dict, createing mask
                    if key in structure.keys():
```

```
                        ds = scan[key]
                        totalmask = np.zeros([ds.pixel_array.shape[0],
ds.pixel_array.shape[1]])
                        x = ds.ImagePositionPatient[0]
                        y = ds.ImagePositionPatient[1]
                        z = ds.ImagePositionPatient[2]
                        ps = ds.PixelSpacing[0]
                        contours = structure[key]
                        totalMask = np.zeros([ds.pixel_array.shape[0],
ds.pixel_array.shape[1]])
                        for contour in contours:
                            mask = np.zeros([ds.pixel_array.shape[0],
ds.pixel_array.shape[1]])
                            r, c = polygon((contour[:, 0] - x) / ps,
(contour[:, 1] - y) / ps, mask.shape)
                            mask[r, c] = 1
                            #r2, c2 = polygon_perimeter((contour[:, 0]- x)
/ ps, (contour[:, 1] - y) / ps, mask.shape)
                            #mask[r2, c2] = 10
                            totalmask += np.fliplr(np.rot90(mask, axes=(1,
0)))


                        scan_pixels[key] = ds.pixel_array
                        struct_masks[key] = totalmask > 0
                    else:
                        #Create empty matrix with the same shape as the
scan array
                        ds = scan[key]
                        struct_masks[key] =
(np.zeros([ds.pixel_array.shape[0], ds.pixel_array.shape[1]]))
                        scan_pixels[key] = ds.pixel_array

                #Sorting by position
                sorted_dict_scan = {k: v for k, v in
sorted(scan_pixels.items(), key=lambda item: -item[0])}
                sorted_dict_masks = {k: v for k, v in
sorted(struct_masks.items(), key=lambda item: -item[0])}

                #Getting the pixel data
                pixels = sorted_dict_scan.values()
                masks = sorted_dict_masks.values()


                #Returning the pixels and mask as arrays
                return np.asarray(list(pixels)), np.asarray(list(masks))


class Structure:

    def __init__(self, patient_id, structure_name, structure_id=None,
structure_data=None):
        self.patient_id = patient_id
        self.structure_name = structure_name
        self.structure_id = structure_id
        self.structure_data = structure_data

    #Getting the file for a given patient
    def get_file(self):
        #Looping over current working directory
        for dirName, subdirList, fileList in os.walk(os.getcwd()):
```

```
            for filename in fileList:
                #If the file matches patient id, use this file
                if filename == "RS.Jacks" + str(self.patient_id) + ".dcm":
                    path = (os.path.join(dirName, filename))

                    return dicom.read_file(path)

    def get_name(self):
        rs = self.get_file()


    #Matching the structure name to its corresponding number, and storing
it
    def get_structure_id(self):
        rs = self.get_file()
        for i in range(len(rs.StructureSetROISequence)):
            for name in self.structure_name:
                if re.fullmatch(rs.StructureSetROISequence[i].ROIName,
name):
                    self.structure_id =
rs.StructureSetROISequence[i].ROINumber
                    print("Structure located for patient
",str(self.patient_id),  ": ", rs.StructureSetROISequence[i].ROIName)

    #Getting pixel data for the structure, returns dict with {position :
pixel data}
    def find_structure2(self):
        rs = self.get_file()
        slice_dict = {}
        #Looping over all structures
        for structure in rs.ROIContourSequence:
            #If the structure sequence matches the number of the structure
ID
            if structure.ReferencedROINumber == self.structure_id:

                #Check that it has pixel data for this structure
                if "ContourSequence" in structure:
                    #Looping over the different slices with pixel data for
structure, reshape and save
                    for slc in structure.ContourSequence:
                        contourdata = slc.ContourData
                        contour = np.reshape(contourdata, (len(contourdata)
// 3, 3))
                        pos_id = contour[0][2]
                        if not pos_id in slice_dict:
                            slice_dict[pos_id] = list()
                        slice_dict[pos_id].append(contour)
                    self.structure_data = slice_dict



                #If no pixel data for the patient, skip patient
                else:
                    print("No Contoursequence for this structure and
patient, skipping patient")
                    self.structure_data = None
        return self.structure_data


def get_unet(input_shape):
    inputs = keras.layers.Input((input_shape))
```

```
    c1 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(inputs)
    c1 = keras.layers.Dropout(0.1)(c1)
    c1 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c1)
    p1 = keras.layers.MaxPooling2D((2, 2))(c1)

    c2 = keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p1)
    c2 = keras.layers.Dropout(0.1)(c2)
    c2 = keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c2)
    p2 = keras.layers.MaxPooling2D((2, 2))(c2)

    c3 = keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
    c3 = keras.layers.Dropout(0.2)(c3)
    c3 = keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c3)
    p3 = keras.layers.MaxPooling2D((2, 2))(c3)

    c4 = keras.layers.Conv2D(512, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)
    c4 = keras.layers.Dropout(0.2)(c4)
    c4 = keras.layers.Conv2D(512, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c4)
    p4 = keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = keras.layers.Conv2D(1024, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p4)
    c5 = keras.layers.Dropout(0.3)(c5)
    c5 = keras.layers.Conv2D(1024, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c5)

    u6 = keras.layers.Conv2DTranspose(512, (2, 2), strides=(2, 2),
padding='same')(c5)
    u6 = keras.layers.concatenate([u6, c4])
    c6 = keras.layers.Conv2D(512, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u6)
    c6 = keras.layers.Dropout(0.2)(c6)
    c6 = keras.layers.Conv2D(512, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c6)

    u7 = keras.layers.Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(c6)
    u7 = keras.layers.concatenate([u7, c3])
    c7 = keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u7)
    c7 = keras.layers.Dropout(0.2)(c7)
    c7 = keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c7)

    u8 = keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(c7)
    u8 = keras.layers.concatenate([u8, c2])
    c8 = keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u8)
    c8 = keras.layers.Dropout(0.1)(c8)
    c8 = keras.layers.Conv2D(128, (3, 3), activation='relu',
```

```
kernel_initializer='he_normal', padding='same')(c8)

    u9 = keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(c8)
    u9 = keras.layers.concatenate([u9, c1], axis=3)
    c9 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u9)
    c9 = keras.layers.Dropout(0.1)(c9)
    c9 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c9)

    outputs = keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

    model = keras.Model(inputs=[inputs], outputs=[outputs])

    model.summary()

    return model



# Foundation of architecture retrieved from
https://www.kaggle.com/code/keegil/keras-u-net-starter-lb-0-277/notebook
def get_model(input_shape):
    inputs = keras.layers.Input((input_shape))


    c1 = keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(inputs)
    c1 = keras.layers.Dropout(0.1)(c1)
    c1 = keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c1)
    p1 = keras.layers.MaxPooling2D((2, 2))(c1)

    c2 = keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p1)
    c2 = keras.layers.Dropout(0.1)(c2)
    c2 = keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c2)
    p2 = keras.layers.MaxPooling2D((2, 2))(c2)

    c3 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
    c3 = keras.layers.Dropout(0.2)(c3)
    c3 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c3)
    p3 = keras.layers.MaxPooling2D((2, 2))(c3)

    c4 = keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)
    c4 = keras.layers.Dropout(0.2)(c4)
    c4 = keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c4)
    p4 = keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(p4)
    c5 = keras.layers.Dropout(0.3)(c5)
    c5 = keras.layers.Conv2D(256, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c5)
```

```
    u6 = keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(c5)
    u6 = keras.layers.concatenate([u6, c4])
    c6 = keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u6)
    c6 = keras.layers.Dropout(0.2)(c6)
    c6 = keras.layers.Conv2D(128, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c6)

    u7 = keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(c6)
    u7 = keras.layers.concatenate([u7, c3])
    c7 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u7)
    c7 = keras.layers.Dropout(0.2)(c7)
    c7 = keras.layers.Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c7)

    u8 = keras.layers.Conv2DTranspose(32, (2, 2), strides=(2, 2),
padding='same')(c7)
    u8 = keras.layers.concatenate([u8, c2])
    c8 = keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u8)
    c8 = keras.layers.Dropout(0.1)(c8)
    c8 = keras.layers.Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c8)

    u9 = keras.layers.Conv2DTranspose(16, (2, 2), strides=(2, 2),
padding='same')(c8)
    u9 = keras.layers.concatenate([u9, c1], axis=3)
    c9 = keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(u9)
    c9 = keras.layers.Dropout(0.1)(c9)
    c9 = keras.layers.Conv2D(16, (3, 3), activation='relu',
kernel_initializer='he_normal', padding='same')(c9)

    outputs = keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

    model = keras.Model(inputs=[inputs], outputs=[outputs])

    return model



def drop_useless_mri_t1(x, y):
    scan = x
    masks = y
    new_scan = []
    new_masks = []
    for i in range(len(scan)):

        if np.max(scan[i]) > 0.3:
            new_scan.append(scan[i])
            new_masks.append(masks[i])
        else:
            continue

    return new_scan, new_masks
```

```
def drop_useless_ct(x, y):
    scan = x
    masks = y
    new_scan = []
    new_masks = []
    for i in range(len(scan)):

        if np.sum(scan[i]) > 4400:
            new_scan.append(scan[i])
            new_masks.append(masks[i])
        else:
            continue

    return new_scan, new_masks



def drop_useless_mri_t2(x, y):
    scan = x
    masks = y
    new_scan = []
    new_masks = []
    for i in range(len(scan)):

        if np.max(scan[i]) > 0.38:
            new_scan.append(scan[i])
            new_masks.append(masks[i])
        else:
            continue

    return new_scan, new_masks


#Generate
def generate_data_for_network2(number_of_patients, modality, structure):
    X_values = []
    Y_values = []

    X_values_val = []
    Y_values_val = []

    X_values_test = []
    Y_values_test = []

    #Generating patients
    for i in range(1, number_of_patients+1):
        patient = Patient(i)

        data  = patient.get_pixels_and_masks(modality, structure)
        if modality == "CT":
            if i == 5 or i== 15 or i == 25:
                if data is not None and np.max(data[1] > 0):
                    print("Patient ", i, "is appended to test set")
                    #If the shape is 512x512, append scan to X and mask to
Y
                    if np.array(data[0][1]).shape == (512, 512):
                        for x in data[0]:
                            a = x / 4096
                            X_values_test.append(a)
                        for y in data[1]:
                            a = y
```

```
                            Y_values_test.append(a)

                    #If shape is 320x320, resize before appending
                    elif np.array(data[0][1]).shape == (320, 320):
                        for x in data[0]:
                            a = x / 4096
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_test.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_test.append(resized_array)

                    elif np.array(data[0][1]).shape == (256, 256):
                        for x in data[0]:
                            a = x / 4096
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_test.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_test.append(resized_array)
                    print("Length of test set:", len(X_values_test))
                else:
                    print("Either no data from scan or no information in
segmentation")
                    continue


            elif i == 3 or i== 11 or i == 28:
                if data is not None and np.max(data[1] > 0):
                    print("Patient ", i, "is appended to val set")
                    #If the shape is 512x512, append scan to X and mask to
Y
                    if np.array(data[0][1]).shape == (512, 512):
                        for x in data[0]:
                            a = x / 4096
                            X_values_val.append(a)
                        for y in data[1]:
                            a = y
                            Y_values_val.append(a)

                    #If shape is 320x320, resize before appending
                    elif np.array(data[0][1]).shape == (320, 320):
                        for x in data[0]:
                            a = x / 4096
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_val.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_val.append(resized_array)

                    elif np.array(data[0][1]).shape == (256, 256):
                        for x in data[0]:
```

```
                            a = x / 4096
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_val.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_val.append(resized_array)
                    print("Length of val set:", len(X_values_val))
                else:
                    print("Either no data from scan or no information in
segmentation")
                    continue
            else:
                if data is not None and np.max(data[1] > 0):
                    print("Patient ", i, "is appended to train set")
                    #If the shape is 512x512, append scan to X and mask to
Y
                    if np.array(data[0][1]).shape == (512, 512):
                        for x in data[0]:
                            a = x / 4096
                            X_values.append(a)
                        for y in data[1]:
                            a = y
                            Y_values.append(a)

                    #If shape is 320x320, resize before appending
                    elif np.array(data[0][1]).shape == (320, 320):
                        for x in data[0]:
                            a = x / 4096
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values.append(resized_array)
                    elif np.array(data[0][1]).shape == (256, 256):
                        for x in data[0]:
                            a = x / 4096
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values.append(resized_array)
                    print("Length of train set:", len(X_values))
                else:
                    continue
        else:
            if i == 5 or i== 15 or i == 25:
                if data is not None and np.max(data[1] > 0):
                    print("Patient ", i, "is appended to test set")
                    #If the shape is 512x512, append scan to X and mask to
Y
                    max_value = np.max(data[0])
                    if np.array(data[0][1]).shape == (512, 512):
```

```
                        for x in data[0]:
                            a = x / max_value
                            X_values_test.append(a)
                        for y in data[1]:
                            a = y
                            Y_values_test.append(a)

                    #If shape is 320x320, resize before appending
                    elif np.array(data[0][1]).shape == (320, 320):
                        for x in data[0]:
                            a = x / max_value
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_test.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_test.append(resized_array)

                    elif np.array(data[0][1]).shape == (256, 256):
                        for x in data[0]:
                            a = x / max_value
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_test.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_test.append(resized_array)
                    print("Length of test set:", len(X_values_test))
                else:
                    print("Either no data from scan or no information in
segmentation")
                    continue


            elif i == 4 or i== 11 or i == 29:
                if data is not None and np.max(data[1] > 0):
                    print("Patient ", i, "is appended to val set")
                    print("Patient ", str(i), "shape", data[0][1].shape)
                    #If the shape is 512x512, append scan to X and mask to
Y
                    max_value = np.max(data[0])
                    if np.array(data[0][1]).shape == (512, 512):
                        for x in data[0]:
                            a = x / max_value
                            X_values_val.append(a)
                        for y in data[1]:
                            a = y
                            Y_values_val.append(a)

                    #If shape is 320x320, resize before appending
                    elif np.array(data[0][1]).shape == (320, 320):
                        for x in data[0]:
                            a = x / max_value
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_val.append(resized_array)
                        for y in data[1]:
```

```
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_val.append(resized_array)

                    elif np.array(data[0][1]).shape == (256, 256):
                        for x in data[0]:
                            a = x / max_value
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values_val.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values_val.append(resized_array)
                    print("Length of val set:", len(X_values_val))
                else:
                    print("Either no data from scan or no information in
segmentation")
                    continue
            else:
                if data is not None and np.max(data[1] > 0):
                    print("Patient ", i, "is appended to train set")
                    #If the shape is 512x512, append scan to X and mask to
Y
                    max_value = np.max(data[0])
                    if np.array(data[0][1]).shape == (512, 512):
                        for x in data[0]:
                            a = x / max_value
                            X_values.append(a)
                        for y in data[1]:
                            a = y
                            Y_values.append(a)

                    #If shape is 320x320, resize before appending
                    elif np.array(data[0][1]).shape == (320, 320):
                        for x in data[0]:
                            a = x / max_value
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values.append(resized_array)
                    elif np.array(data[0][1]).shape == (256, 256):
                        for x in data[0]:
                            a = x / max_value
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            X_values.append(resized_array)
                        for y in data[1]:
                            a = y
                            resized_array = resize(a, (512, 512),
anti_aliasing=True)
                            Y_values.append(resized_array)
                    print("Length of train set:", len(X_values))
                else:
                    continue
```

```
    print("Length before drop x, x_val, x_test = ",len(X_values),"  ",
len(X_values_val),"  ", len(X_values_test))
    print("Lenght before drop y, y_val, y_test = ", len(Y_values),"  ",
len(Y_values_val),"  ", len(Y_values_test))
    if modality == "MRI_T1":
        X_values,Y_values = drop_useless_mri_t1(X_values, Y_values)
        X_values_val, Y_values_val = drop_useless_mri_t1(X_values_val,
Y_values_val)
        X_values_test,Y_values_test = drop_useless_mri_t1(X_values_test,
Y_values_test)
    elif modality == "CT":
        X_values,Y_values = drop_useless_ct(X_values, Y_values)
        X_values_val, Y_values_val = drop_useless_ct(X_values_val,
Y_values_val)
        X_values_test,Y_values_test = drop_useless_ct(X_values_test,
Y_values_test)
    elif modality == "MRI_T2":
        X_values,Y_values = drop_useless_mri_t2(X_values, Y_values)
        X_values_val, Y_values_val = drop_useless_mri_t2(X_values_val,
Y_values_val)
        X_values_test,Y_values_test = drop_useless_mri_t2(X_values_test,
Y_values_test)
```

```
    print("Length after drop x, x_val, x_test = ",len(X_values),"  ",
len(X_values_val),"  ", len(X_values_test))
    print("Lenght after drop y, y_val, y_test = ", len(Y_values),"  ",
len(Y_values_val),"  ", len(Y_values_test))
    scans = np.stack(X_values)
    masks = np.stack(Y_values)
    scans_val = np.stack(X_values_val)
    masks_val = np.stack(Y_values_val)
    scans_test = np.stack(X_values_test)
    masks_test = np.stack(Y_values_test)

    #Reshaping to fit network input
    scans2 = scans.reshape(scans.shape[0], scans.shape[1], scans.shape[2],
1)
    masks2 = masks.reshape(masks.shape[0], masks.shape[1], masks.shape[2],
1)
    scans2_val = scans_val.reshape(scans_val.shape[0], scans_val.shape[1],
scans_val.shape[2], 1)
    masks2_val = masks_val.reshape(scans_val.shape[0], scans_val.shape[1],
scans_val.shape[2], 1)
    scans2_test = scans_test.reshape(scans_test.shape[0],
scans_test.shape[1], scans_test.shape[2], 1)
    masks2_test = masks_test.reshape(masks_test.shape[0],
masks_test.shape[1], masks_test.shape[2], 1)
    return scans2.astype(np.float32),
masks2.astype(np.float32),scans2_val.astype(np.float32),
masks2_val.astype(np.float32), scans2_test.astype(np.float32),
masks2_test.astype(np.float32),


def dice_coef(y_true, y_pred, smooth = 1):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
```

```
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) +
K.sum(y_pred_f) + 1)



def dice_coef_loss(y_true, y_pred):
    return 1 - dice_coef(y_true, y_pred)



#https://github.com/huanglau/Keras-Weighted-Binary-Cross-
Entropy/blob/master/DynCrossEntropy.py
def weighted_bincrossentropy(true, pred, weight_zero = 1.0, weight_one =
0.5):
    """
    Calculates weighted binary cross entropy. The weights are fixed.

    This can be useful for unbalanced catagories.

    Adjust the weights here depending on what is required.

    For example if there are 10x as many positive classes as negative
classes,
        if you adjust weight_zero = 1.0, weight_one = 0.1, then false
positives
        will be penalize 10 times as much as false negatives.
    """

    # calculate the binary cross entropy
    bin_crossentropy = keras.backend.binary_crossentropy(true, pred)

    # apply the weights
    weights = true * weight_one + (1. - true) * weight_zero
    weighted_bin_crossentropy = weights * bin_crossentropy

    return keras.backend.mean(weighted_bin_crossentropy)


def wbc_dice_loss(y_true, y_pred):
    return 0.5 * weighted_bincrossentropy (y_true, y_pred) + 0.5 *
dice_coef_loss(y_true, y_pred)


batch_size = 16


filepath = "Model_weights/Large Unet/Brain_CT_dice.epoch{epoch:02d}-
loss{val_loss:.2f}.h5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath,
                             monitor="val_dice_coef",
                             save_weights_only=True,
                             verbose=1,
                             save_best_only=True,
                             mode="max")


scan, masks, scan_val, masks_val, scan_test, masks_test =
generate_data_for_network2(30, "MRI_T1", ["PapezCircle", "Papez Circle",
"Papez-new"])
```

```
model = get_model(scan[1].shape)


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.001),
              loss=dice_coef_loss,
              metrics=dice_coef)

callbacks = [checkpoint]


#model_history = model.fit(train, validation_data = val, validation_steps =
batch_size/2 , epochs = 25, steps_per_epoch = len(scan_train)/batch_size)
model_history = model.fit(x = scan, y = masks, batch_size = batch_size,
epochs = 100,  validation_data = (scan_val, masks_val), shuffle = True,
callbacks = callbacks)


def normalize_data(data):
    return((data - np.min(data))/ (np.max(data)-np.min(data)))


val_loss_norm = normalize_data(model_history.history['val_loss'])
train_loss_norm = normalize_data(model_history.history['loss'])


predict = model.predict(scan_test)


plt.plot(train_loss_norm)
plt.plot(val_loss_norm)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.grid()
plt.ylim(0,1)
plt.yticks(np.arange(0.1,1.1,0.1))
plt.xticks(np.arange(0,101,5))
#plt.savefig("Loss T2 papez WBC1_05")
plt.show()


plt.plot(model_history.history['dice_coef'])
plt.plot(model_history.history['val_dice_coef'])
#plt.title('Dice score for TemporalLobeRt - WBinary Cross')
plt.ylabel('Dice score')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')
plt.grid()
plt.ylim(0,1)
plt.yticks(np.arange(0,1.1,0.1))
plt.xticks(np.arange(0, 101,5))
#plt.savefig("Dice T2 Papez WBC1_05")
plt.show()
```

```
def dice_coef_test(y_true, y_pred):
    y_true_f = y_true.flatten()
    #print(y_true_f)
    y_pred_f = y_pred.flatten()
    union = np.sum(y_true_f) + np.sum(y_pred_f)
    #if union==0: return 1
    intersection = np.sum(y_true_f * y_pred_f)
    return 2. * intersection / union


print("Total test dice: ", dice_coef(masks_test, predict))
print("Total test dice: ", dice_coef_test(masks_test, predict))


def get_recall(y_ture, y_pred):
    recall = tf.keras.metrics.Recall()
    recall.update_state(y_ture, y_pred)
    return recall.result().numpy()


def get_precision(y_ture, y_pred):
    precision = tf.keras.metrics.Precision()
    precision.update_state(y_ture, y_pred)
    return precision.result().numpy()



test_p = get_precision(masks_test, predict)
print("Precision",test_p)
test_r = get_recall(masks_test, predict)
print("Recall", test_r)


def plot_slice(scan, mask_ture, mask_pred, slice_nr):
    mask_pred_new = np.ma.masked_where(mask_pred[slice_nr] <=
0.5,mask_pred[slice_nr])
    mask_true_new = np.ma.masked_where(mask_ture[slice_nr]==0,
mask_ture[slice_nr])
    cmap = colors.ListedColormap(["red"])
    cmap2 = colors.ListedColormap(["blue"])
    f = plt.figure()
    plt.imshow(scan[slice_nr][120:360,120:360], cmap = "gray", alpha = 1.0)
    plt.axis("off")
    plt.imshow(mask_pred_new[120:360,120:360], cmap = cmap, alpha = 0.75)
    plt.imshow(mask_true_new[120:360,120:360], cmap = cmap2, alpha = 0.5)

    dice_score = round(dice_coef_test(mask_ture[slice_nr],
mask_pred[slice_nr]), 3)
    recall = round(get_recall(mask_ture[slice_nr], mask_pred[slice_nr]), 3)
    precision = round(get_precision(mask_ture[slice_nr],
mask_pred[slice_nr]), 3)
    print("Slice nr", slice_nr)
    title = str ("Dice score = "+ str(dice_score) + "\n Precision = " +
str(precision) + "\n Recall = " + str(recall))
    plt.title(title)
    red_patch = mpatches.Patch(color='red', label='Predicted mask')
    blue_patch = mpatches.Patch(color='blue', label='Actual mask')
    purple_patch = mpatches.Patch(color='purple', label='Overlap')
    plt.legend(handles = [red_patch, blue_patch, purple_patch], loc =
"lower left")
```

```
#plt.savefig("Brain MRI_T1 BC" + str(slice_nr))
plt.show()
```