Linköping University | Department of Biomedical Engineering
Master's thesis, 30 ECTS | Computer Science
2023 | LIU-IMT-TFK-A–23/711--SE

Comparative Analysis of Transformer and CNN Based Models for 2D Brain Tumor Segmentation

Henrik Träff

Supervisor : Muhammad Usman Akbar
Examiner : Anders Eklund

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

A brain tumor is an abnormal growth of cells within the brain, which can be categorized into primary and secondary tumor types. The most common type of primary tumors in adults are gliomas, which can be further classified into high-grade gliomas (HGGs) and low-grade gliomas (LGGs). Approximately 50% of patients diagnosed with HGG pass away within 1-2 years. Therefore, the early detection and prompt treatment of brain tumors are essential for effective management and improved patient outcomes.

Brain tumor segmentation is a task in medical image analysis that entails distinguishing brain tumors from normal brain tissue in magnetic resonance imaging (MRI) scans. Computer vision algorithms and deep learning models capable of analyzing medical images can be leveraged for brain tumor segmentation. These algorithms and models have the potential to provide automated, reliable, and non-invasive screening for brain tumors, thereby enabling earlier and more effective treatment. For a considerable time, Convolutional Neural Networks (CNNs), including the U-Net, have served as the standard backbone architectures employed to address challenges in computer vision. In recent years, the Transformer architecture, which already has firmly established itself as the new state-of-the-art in the field of natural language processing (NLP), has been adapted to computer vision tasks. The Vision Transformer (ViT) and the Swin Transformer are two architectures derived from the original Transformer architecture that have been successfully employed for image analysis. The emergence of Transformer based architectures in the field of computer vision calls for an investigation whether CNNs can be rivaled as the de facto architecture in this field.

This thesis compares the performance of four model architectures, namely the Swin Transformer, the Vision Transformer, the 2D U-Net, and the 2D U-Net which is implemented with the nnU-Net framework. These model architectures are trained using increasing amounts of brain tumor images from the BraTS 2020 dataset and subsequently evaluated on the task of brain tumor segmentation for both HGG and LGG together, as well as HGG and LGG individually. The model architectures are compared on total training time, segmentation time, GPU memory usage, and on the evaluation metrics Dice Coefficient, Jaccard Index, precision, and recall. The 2D U-Net implemented using the nnU-Net framework performs the best in correctly segmenting HGG and LGG, followed by the Swin Transformer, 2D U-Net, and Vision Transformer. The Transformer based architectures improve the least when going from 50% to 100% of training data. Furthermore, when data augmentation is applied during training, the nnU-Net outperforms the other model architectures, followed by the Swin Transformer, 2D U-Net, and Vision Transformer. The nnU-net benefited the least from employing data augmentation during training, while the Transformer based architectures benefited the most.

In this thesis we were able to perform a successful comparative analysis effectively showcasing the distinct advantages of the four model architectures under discussion. Future comparisons could incorporate training the model architectures on a larger set of brain tumor images, such as the BraTS 2021 dataset. Additionally, it would be interesting to explore how Vision Transformers and Swin Transformers, pre-trained on either ImageNet-21K or RadImageNet, compare to the model architectures of this thesis on brain tumor segmentation.

Acknowledgments

I would like to thank my supervisor, Muhammad Usman Akbar, for his guidance and support throughout this thesis. Our discussions helped with the development of this work. Thanks to my examiner, Anders Eklund, for his assistance and help throughout the course of this thesis project. I would also like to thank my friends and family for their support.

Contents

List of Figures

List of Tables

1 Introduction

A brain tumor is an abnormal growth of cells within the brain. It can be either benign (non-cancerous) or malignant (cancerous) and can affect brain function by putting pressure on surrounding tissue and disrupting normal brain activity. Brain tumors can cause a range of symptoms, including headaches, seizures, changes in vision or hearing, weakness, and memory problems. Treatment options can include surgery, radiation therapy, and chemotherapy, and the choice of treatment depends on the type, location, and size of the tumor, as well as the overall health of the patient. Early detection and prompt treatment are crucial in managing brain tumors and improving patient outcomes. Researchers have been exploring the application of computer vision in the detection of brain tumors as a means of improving diagnostic accuracy and efficiency. By leveraging algorithms capable of analyzing medical images such as MRI scans, computer vision has the potential to provide automated, reliable, and non-invasive screening for brain tumors, thereby enabling earlier and more effective treatment.

Medical image segmentation, similar to natural image segmentation, involves separating the target object (such as an organ) from a medical image, which can be a 2D or 3D image [35]. This process can be performed through manual, semi-automated, or fully automated means. The type of medical image used may vary, including CT scans, MRI, ultrasonography, PET scans, X-rays, or even hybrid scans like PET/CT, depending on the desired object and the purpose of the imaging. In the same vein, brain tumor segmentation refers to the process of extracting a brain tumor from a medical image.

Convolutional Neural Networks (CNNs), such as the U-Net [51], have long been the de facto architecture used to solve problems in computer vision and medical image segmentation [32, 23, 27]. In conjunction with the CNN architecture serving as the backbone network for several computer vision tasks, many architectural extensions and advancements have been made, leading to improved performance and broad benefits to the field [17, 23, 65, 19]. In parallel to the reign of CNNs in computer vision, Transformers [64], and their self-attention-based architectures have become the model of choice in natural language processing (NLP) [16]. Before the Transformers introduction in 2017, Recurrent Neural Networks (RNN) [52, 30] and Long-Short Term Memory (LSTM) [24] were the established state-of-the-art approaches to solving sequence modeling and transduction tasks [59, 2, 8]. With the Transformer architecture firmly establishing itself as the new state-of-the-art in the field of NLP, it led researchers to investigate if this architecture can be adapted to computer vision tasks.

In 2020, Dosovitskiy et al. [13] introduced the Vision Transformer (ViT), a pure Transformer based model architecture, minimally modified to work with images as input. Applied to the task of image classification, ViT matched, or even exceeded, then state-of-the-art, on many image classification datasets when pre-trained on a large dataset such as the ImageNet dataset [11]. The promising result yielded by ViT showed that the Transformer architecture with its self-attention mechanism can be applied to computer vision tasks. However, there are difficulties in transferring the high performance of the language Transformers to the visual domain. While word tokens serve as the fundamental units of processing in NLP Transformers, visual elements can greatly vary in scale, an aspect that is particularly challenging in tasks like object detection [36, 55, 56]. In Transformer based models such as the Vision Transformer [13] and the original Transformer [64], tokens are of a fixed scale, making these models inadequate for visual applications. Furthermore, the resolution of pixels in images, which is much higher than the resolution of words in text passages, poses a challenge for tasks such as semantic and image segmentation, which require dense predictions at the pixel level. For high-resolution images, using a Transformer to achieve this would be impractical due to its quadratic computational complexity concerning the image size [13].

In 2021, the Swin Transformer was introduced by Liu et al. [40], a Transformer based architecture designed to reduce the memory and computational requirements of prior Transformer based architectures. At the time, the Swin Transformer achieved state-of-the-art results on the object detection dataset COCO [37] and ADE20K semantic segmentation [69], demonstrating that a Transformer based model architecture can be applied to dense vision tasks.

With the addition of multiple Transformer based architectures and models in computer vision, it did not take long before researchers also created hybrid models, combining Transformer and CNN architectures. Two such hybrid models are the UNETR [22] and the Swin UNETR [21] from NVIDIA which broadly described, combines a Transformer encoder and a CNN decoder in an encoder-decoder configuration. UNETR employs the Vision Transformer in its architectures whereas the Swin UNETR employs the Swin Transformer. The Swin UNETR placed seventh place in the BraTS 2021 Segmentation Challenge [21, 3, 45, 6, 4, 5]. The idea behind combining the architectures is to integrate the strength of the Transformer architectures with the strength of the CNN architectures. Mainly, combining a CNN with a Transformer so that it can capture long-range relationships within an image.

The continuing emergence of Transformer based model architectures in computer vision calls for an investigation regarding if the CNNs can be rivaled in terms of being de facto architectures. In the field of medical image segmentation, whether vision Transformers can compare to U-Net architecture as a general-purpose backbone network needs to be explored. In this thesis, four models in total, two CNN based, and two Transformer based model architectures, are trained and compared on the task of Brain Tumor Image Segmentation using brain tumor images from the BraTS 2020 dataset [45, 7, 6, 4, 5].

1.1 Aim

This thesis aims to conduct a comparative analysis of how CNN based architectures compare to vision Transformers, both in general and as backbone networks for the task of Brain Tumor Image Segmentation on the BraTS 2020 dataset. The model architectures being compared are the Vision Transformer, Swin Transformer, 2D U-Net, and 2D nnU-Net. All of these models are trained on 2D slices of the 3D brain tumor images from the BraTS 2020 dataset. The models are evaluated based on several metrics, including the Dice Coefficient, Jaccard Index, precision, recall, training time, GPU memory usage, and image segmentation time. Additionally, the extent to which each model architecture improves with an increasing amount of training data is observed. Furthermore, the performance of each model architecture on images associated with subjects diagnosed with either low-grade (LGG) or high-grade glioma (HGG) is also documented.

This thesis aims to contribute to the ongoing comparison of CNNs and vision Transformers in the field of medical imaging and computer vision.

1.2 Research Questions

1. When trained on increasing amounts of brain tumor images from the BraTS 2020 dataset, how do a Swin Transformer, a Vision Transformer, a typical 2D U-net, and a 2D U-Net trained using the nnU-Net framework, compare in terms of Dice Coefficient, Jaccard Index, precision, recall and training time?
2. In terms of segmentation time and GPU memory usage, how do the model architectures compare to each other?
3. Which of the model architectures perform better in correctly segmenting MRI scans of patients with LGG and HGG, respectively?
4. How does the use of data augmentation during training impact the ability of each model architecture to accurately segment MRI scans?

1.3 Contributions

U-Nets implemented with and without the nnU-Net framework have previously been trained and used on the BraTS 3D brain tumor images, yielding good results. Due to the three-dimensional nature of brain tumor images, less attention has been given to training 2D model architectures solely on 2D slices extracted from these images. To our knowledge, the Swin Transformer and the Vision Transformer, which are 2D model architectures, have not previously been trained and evaluated on 2D slices of brain tumor images from the BraTS 2020 dataset. Furthermore, no previous comparison has been conducted between a 2D U-Net, a 2D U-Net implemented with the nnU-Net framework, a Vision Transformer, and a Swin Transformer architecture trained on 2D slices of brain tumor images from the BraTS 2020 dataset.

1.4 Delimitations

The project is delimited to only evaluate the models based on one task, Brain Tumor Image Segmentation, using brain tumor images from the BraTS 2020 dataset. The BraTS 2020 dataset is chosen over the BraTS 2021 dataset [3, 45, 6, 4, 5] because the latter does not include information on whether the brain tumors of each respective subject is a low-grade glioma or high-grade glioma. The BraTS 2020 dataset comprises 3D MRI scans. Since the model architectures implemented in this project take 2D images as input, the 3D images of the BraTS 2020 dataset are sliced into 2D slices.

Many of the steps in the nnU-Net framework are omitted. These steps are the data augmentation step, five-fold cross-validation, ensembling, and post-processing. Thus, the nnU-Net framework serves to train 2D U-nets, only making use of the pre-processing step, and using a training pipeline adapted to the data of the BraTS 2020 dataset. Different initialization strategies and pre-training methods for the respective models are not evaluated. The models are trained from scratch; thus, the model parameters are randomly initialized. Furthermore, each model is trained for 15 epochs, since training the models on more epochs is not feasible in the time frame of the thesis. Thus, an extensive comparison where multiple approaches are used to train and evaluate the models, such as employing different loss functions or learning rates, is not covered in this thesis. The evaluation and training of each model produced in this thesis are performed on an NVIDIA Geforce 3090 RTX.

1.5 Thesis Outline

This thesis report comprises six chapters. To begin with, chapter 1 motivates the need for conducting a comparative analysis and introduces the research questions for the project. Subsequently, chapter 2 imparts the essential background knowledge on the machine learning architectures employed in this thesis. Following that, chapter 3 outlines the two experiments conducted to generate the results, which are subsequently presented in chapter 4. In chapter 5, a comprehensive discussion on the implications of the experimental outcomes is provided. Lastly, chapter 6 draws a conclusion based on the findings.

2 Theory

This chapter provides an overview of the essential theoretical foundations and background information required to comprehend the subject matter of the thesis.

2.1 Magnetic Resonance Imaging Scans on Brain Tumors

There exist more than 130 identified types of brain tumors and they can be categorized into primary and secondary tumor types [43]. Brain tumors that originate from cells outside of the brain and metastasize into the brain are called secondary brain tumors, whereas primary brain tumors originate from brain cells. The most common type of primary tumors in adults are gliomas, which can be further classified into high-grade gliomas (HGGs) and low-grade gliomas (LGGs) [25, 45]. HGGs are the most aggressive and severe brain tumors. Within 1-2 years or less, almost 50% of the patients diagnosed with HGG pass away. A patient diagnosed with a brain tumor of this type requires immediate treatment [45, 49].

Magnetic Resonance Imaging (MRI) is a reliable tool used for monitoring and analyzing brain tumors, as well as for assisting in surgery planning. MRI scans can capture tumor-induced tissue changes and provide more insight into how the tumor is behaving. Typically, several 3D image modalities are captured with MRI scans. These 3D MRI modalities include T1, T1 with contrast agent (T1ce), T2, and Fluid-attenuated Inversion Recovery (FLAIR). With these types of MRI modalities, different tissue properties of the tumor and area of spread can be highlighted. The FLAIR modality is used to highlight relational properties and differences in tissue water. By using Gadolinium as the contrast agent in T1cee MRI, hyperactive sub-regions of the tumor can be highlighted.

2.2 Machine Learning

Machine learning is a subfield of Artificial Intelligence that is devoted to building and understanding methods that leverage data to learn, conduct reasoning, and act on given data to carry out a specific task [38, 53]. One of the fundamental parts of any machine learning model is how data is incorporated and used to improve the performance of these models. In machine learning, computer programs derive their knowledge from data. A machine learning model is adapted to something more domain-specific by its observed, so-called, training

data. This training data allows the machine learning model's settings to be automatically adjusted. This process of readjusting is built on mathematical foundations and is part of the main idea behind constructing machine learning models. It is with this type of model that relationships between variables and training data or interesting properties such as predictions can be described. The model can be seen as a dense representation of the data that captures interesting properties related to the problem that is being studied.

The way a model leverages and learns from training data is by employing a learning algorithm that, based on the available data, automatically updates the settings or parameters of the model to better fit the data. In machine learning, there are three main types of learning methods: supervised learning, unsupervised learning, and reinforcement learning [53]. In supervised learning, a computer program learns a mapping function and its parameters by observing input-output pairs, where the output is the reference for what the function should output. In unsupervised learning, only input data is available, and the function and its parameters are learnt by finding patterns in the input data without any explicit feedback. Reinforcement learning is a learning method in which the computer program learns based on a reward and punishment system.

In this thesis we work with supervised learning for which training data is available that contains input-output pairs (x, y), where x denotes the input variable and y denotes the output variable [38]. By adapting a mathematical model with the help of the training data, we can yield a model that can predict the output of new input variables from a set of test data that have not been observed by the model before. The learning process of a model is referred to as the training of the model. Depending on the type of output y, different machine learning problems can be defined. If the output of a model is numerical, meaning that it has a natural ordering, then the problem is referred to as a regression problem. Instead, if the output is categorical, meaning that it does not have a natural ordering and that it only takes on discrete values, the problem is called a classification problem [38]. An example of a classification problem is a set of images depicting either a dog, cat, or horse, in need of classification. A model, also known as a classifier, classifies each of the images by labeling them with animal labels such as 'dog', 'cat', or 'horse', and outputs the results.

Regression

One of the most fundamental tasks in supervised learning is regression. Regression based models aim to learn a relationship between an input variable $x = [x_1, x_2, \ldots, x_p]^T$ and a numerical output y [38]. The input variable can either be categorical or numerical. This relationship can be expressed through a function f that maps the input to the output. The mathematical equation for this mapping is as follows,

$$y = f(x) + \epsilon \tag{2.1}$$

where ϵ is an error term that describes the parts of the input-output relationship that are not captured by the model. For a linear regression model, by assuming that the output y can be written as a linear function plus an offset, the equation is rewritten as:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p + \epsilon \tag{2.2}$$

where the coefficients $\theta_0, \theta_1, \theta_p$ are referred to as parameters. This equation can be denoted in a more compact way by introducing the variable $\boldsymbol{\theta} = [\theta_0, \theta_1, \ldots, \theta_p]^T$. Furthermore, the input vector x is extended with a constant 1, $x = [1, x_1, x_2, \ldots, x_p]^T$, resulting in the following equation:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p + \epsilon = (\theta_0, \theta_1, \ldots, \theta_p) \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{pmatrix} + \epsilon = \boldsymbol{\theta}^T x + \epsilon \tag{2.3}$$

The constant is added to $\boldsymbol{x}$ to assure that θ_0, called the intercept or the offset term, is incorporated in $\boldsymbol{\theta}$. The noise term ϵ, is assumed to be independent of x and to have mean zero. However, since the non-zero mean can be included in θ_0, the assumption that ϵ has mean zero is nonrestrictive. The parameters in $\boldsymbol{\theta}$ can take on any value and by employing a learning algorithm, $\boldsymbol{\theta}$ can be updated to better fit the input-output relationship, thus enabling the model to learn based on the training data [38].

A regression model like the one given by Equation 2.1, can be turned into a linear regression model by assuming that f_θ, the θ-dependency, is linear, meaning that $f_\theta = \boldsymbol{\theta}^T x + \epsilon$, and that $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$. These assumptions can be relaxed, allowing f_θ to be an arbitrary non-linear function.

Loss Functions

As mentioned before, the goal in supervised machine learning is to train a model on training data consisting of input-output pairs. The purpose is to enable the model to make predictions on previously unseen input data x, yielding an output $\hat{y}$ that is as similar as possible to the actual output value y. Defining how good a prediction is and how similar $\hat{y}$ and y are, can be achieved in multiple ways, and can be used to help find a $\boldsymbol{\theta}$ such that $\hat{y} - y = \epsilon$. A popular approach to measure how well a prediction $\hat{y}$ is compared to the actual value y, is to define a loss function $L(y, \hat{y})$. A loss function outputs a small value if $\hat{y} \approx y$ and a large value when $\hat{y}$ is not similar to y. Depending on the chosen loss function, a cost function corresponding to the average loss value over the training data is defined. During a model's training, the underlying goal is to find a prediction of theta $\hat{\theta}$ that minimizes the cost as shown by Equation 2.4.

$$\hat{\theta} = \underset{\theta}{argmin} \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}(\boldsymbol{x_i}, \boldsymbol{\theta}), y_i) \tag{2.4}$$

Training a model amounts to finding the best θ by minimizing the summation of losses over all the training data [38].

2.3 Neural Networks

A basic parametric model such as a linear regression model can be used to solve classification and regression tasks. A neural network is a hierarchical model that is made up of multiple basic parametric models stacked on top of each other [38]. The output-input relationship a neural network can capture is more complicated than what a parametric model like a linear regression model can capture. A neural network is made of multiple layers of linear regression models and non-linear activation functions. These activation functions can describe non-linear scalar relationships between an input variable $x = [1, x_1, x_2, \ldots, x_p]^T$ and an output variable $\hat{y}$.

The linear regression models in neural networks have an altered notation compared to Equation 2.2. Equation 2.5 is the linear regression equation with changed notations.

$$\hat{y} = W_1 x_1 + W_2 x_2 + \ldots + W_p x_p + b \tag{2.5}$$

The coefficients are denoted as $W_1, W_2, \ldots, W_p$ instead of $\theta_1, \theta_2, \ldots, \theta_p$, and θ_0 is denoted as b. An activation function $h : \mathbb{R} \to \mathbb{R}$ can be used to yield a generalized linear regression model by passing the linear combination of the inputs into activation function h, transforming the inputs to yield $\hat{y}$.

$$\hat{y} = h(W_1 x_1 + W_2 x_2 + \ldots + W_p x_p + b) \tag{2.6}$$

Commonly used activation functions are the rectified linear unit (ReLU) and the logistic function.

$$ReLU : h(z) = max(0, z)$$

$$Logistic : h(z) = \frac{1}{1+e^{-z}}.$$

A visual representation of Equation 2.6 is illustrated in Figure 2.1

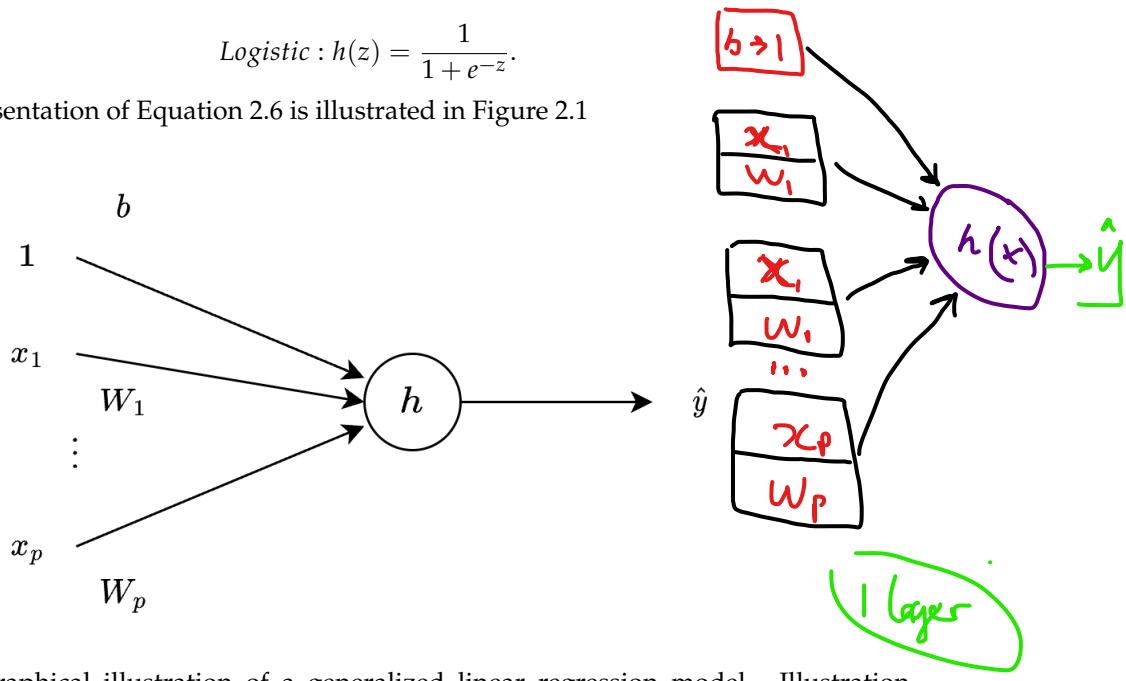

Figure 2.1: Graphical illustration of a generalized linear regression model. Illustration adapted from [38].

Two-Layer and Deep Neural Networks

In Figure 2.1, $\hat{y}$ is computed by one scalar regression model. To expand the generalized regression model to a two-layer neural network, multiple such regression models are added, resulting in the output $\hat{y}$ being the sum of all their outputs. If the number of regression models is denoted as U, then the corresponding parameters for the kth regression model are $b_k, W_{k1}, \ldots, W_{kp}$, and the output is q_k for $k = 1, \ldots, \mathrm{U}$.

$$q_k = h(W_{k1}x_1 + W_{k2}x_2 + \ldots + W_{kp}x_p + b_k). \tag{2.7}$$

Each of the outputs, q_k, make up what is referred to as hidden units and the U different hidden units are the input variables to another linear regression model.

$$\hat{y} = W_1 q_1 + W_1 q_2 + \ldots + W_{\mathrm{U}} q_{\mathrm{U}} + b. \tag{2.8}$$

For clarity's sake, the parameters in Equation 2.7 and 2.8, are denoted with the superscripts (1) and (2) respectively. In Figure 2.2, a two-layer neural network is illustrated. The equations that make up a two-layer neural network are,

$$\begin{aligned} q_1 &= h(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + \ldots + W_{1p}^{(1)}x_p + b_1^{(1)}), \\ &\vdots \\ q_k &= h(W_{k1}^{(1)}x_1 + W_{k2}^{(1)}x_2 + \ldots + W_{kp}^{(1)}x_p + b_k^{(1)}), \end{aligned} \tag{2.9a}$$

$$\hat{y} = W_1^{(2)}q_1 + W_1^{(2)}q_2 + \ldots + W_{\mathrm{U}}^{(2)}q_{\mathrm{U}} + b^{(2)}. \tag{2.9b}$$

Using a matrix notation, the equations in 2.9 can be written more compactly. Each layer's parameters are stacked in a weight matrix W and an offset vector b.

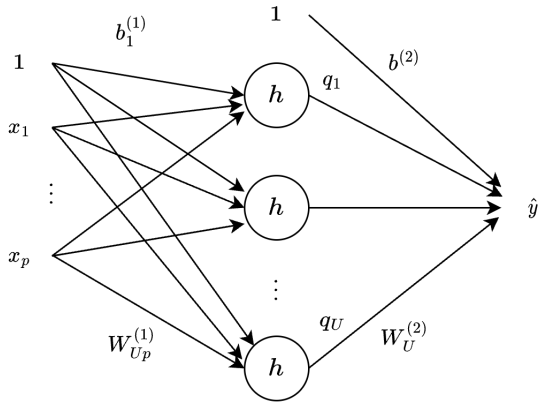

Figure 2.2: A Two-layer neural network. Illustration adapted from [38].

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & \dots & W_{1p}^{(1)} \\ \vdots & \ddots & \vdots \\ W_{\mathrm{U}1}^{(1)} & \dots & W_{\mathrm{U}p}^{(1)} \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_{\mathrm{U}}^{(1)} \end{bmatrix},$$

$$W^{(2)} = \begin{bmatrix} W_1^{(2)} & \dots & W_{\mathrm{U}}^{(2)} \end{bmatrix}, \quad b^{(2)} = \begin{bmatrix} b^{(2)} \end{bmatrix}. \tag{2.10}$$

Thus, a two-layer neural network can be written as,

$$q = h(W^{(1)}x + b^{(1)}),$$
$$\hat{y} = W^{(2)}q + b^{(2)},$$

where $x = \begin{bmatrix} x_1, & \dots, & x_p \end{bmatrix}^T$ and $q = \begin{bmatrix} q_1, & \dots, & q_{\mathrm{U}} \end{bmatrix}^T$.

A two-layer neural network can be expanded further by stacking multiple layers of generalized linear regression models to improve the ability to capture complex input-output relationships. A neural network consisting of multiple layers and activation functions is called a deep neural network. An illustration of such a deep neural network is shown in Figure 2.3.

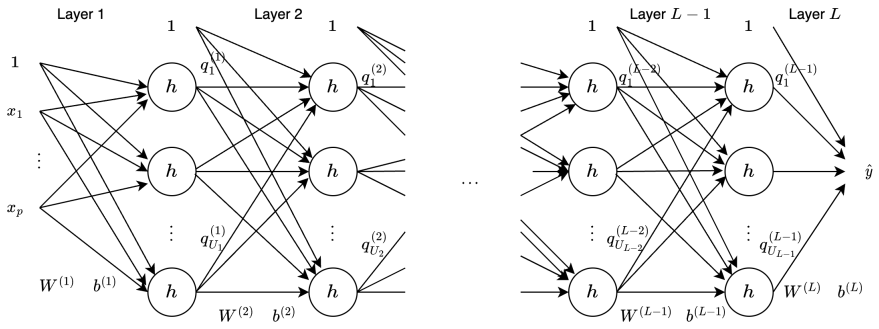

Figure 2.3: A deep neural network. Each layer $l = 1, \dots, L$ has the corresponding parameters $W^{(l)}$ and $b^{(l)}$. Illustration adapted from [38].

2.4 Convolutional Neural Networks

Convolutional Neural Networks, also referred to as Convolutional Networks [34] or CNNs are a type of neural network, specifically designed to process data that has a known grid-like topology [18]. An image can for instance be viewed as a 2D grid of pixels. Compared to neural networks that apply general matrix multiplication in their layers to yield an output, CNNs employ a mathematical operation called convolution to generate layer outputs. The layers in Figure 2.3 and 2.2 are so-called dense layers, meaning each input variable is connected to all hidden units in a subsequent layer [38]. Dense layers might not be able to learn important patterns from image data and thus not generalize well on test data. In comparison, a convolution layer can be used to find a better parameterized model by exploiting the structure found in the images.

Convolutional layers leverage a concept called sparse interactions, which refers to not every input variable interacting with a hidden unit of a subsequent layer. In other words, a hidden unit in a convolutional layer only depends on a small local region of the input image [38]. Intuitively, a pixel in an image can depend more on surrounding pixels than pixels far away. Figure 2.4 illustrates how a pixel sub-region of size 3x3 interacts with an input image to connect to a hidden unit. For cases where the sub-region is partly located outside the image border, a concept called zero-padding can be employed, replacing missing pixels with zeroes.

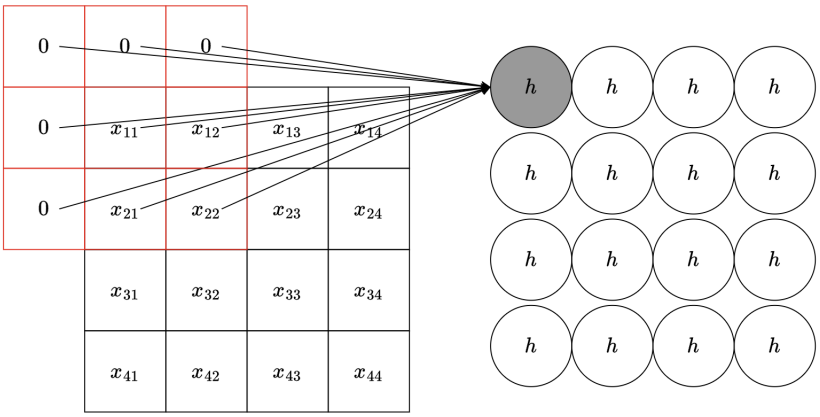

Figure 2.4: An illustration of how a 3x3 sub-region interacts with an input image of size 4x4, to derive which pixels that a hidden unit depends on. Additionally, the concept of zero-padding is also visualized. Illustration adapted from [38].

Parameter sharing is another concept that convolutional layers take advantage of. This entails that the set of parameters in a convolutional layer is the same for each hidden unit, unlike for dense layers, where every hidden unit and its links to input variables has its own unique parameters. Thus, only one set of parameters, referred to as a filter (also known as a kernel or a mask), needs to be learnt and used for all links between the input variables and hidden units. The mapping of input variables to hidden units can be described as the convolution between the input variables and a kernel. Given an input signal, e.g., an image, represented by $f(x, y)$ and a kernel $k(x, y)$, the convolution of these two signals is defined as the sum of the element-wise multiplications of $k(x, y)$ with a small region of $f(x, y)$ centered around each location (x, y). The result of the convolution, $g(x, y)$, is given by:

$$g(x,y) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} f(i,j) * k(x-i, y-j), \tag{2.11}$$

where m and n are the dimensions of the kernel $k(x, y)$.

The particular form of parameter sharing inherent to the convolution operation results in the layers having an additional property called equivariance [18]. The equivariance property states that if the input to a layer is altered then the output is altered in the same way. However, the equivariance of convolution is not true for all changes to the input, such as rotation and scaling [18]. When working with images as data, the convolution operator is used to extract features from an input image by sliding a small filter over the input and computing the dot product between the entries of the filter and the input at any given position. The entries of the kernel are often referred to as weights, and the result of applying a small filter on an image is a feature map, which summarizes the information of the input image at different scales and orientations.

Multiple Channels

To capture all relevant and interesting features present in an image, one filter might not suffice. A CNN and its layers are not limited to only making use of one filter per layer, instead multiple filters can be added to sufficiently capture all the essential information of an image. Adding more filters to each layer increases the number of learnable parameters and subsequently the number of hidden units. The set of hidden units that correspond to a filter is referred to as a channel [38]. By using multiple filters in a convolutional layer, the corresponding hidden units are organized into what is referred to as tensors, of dimension $(rows \times columns \times channels)$. When stacking multiple convolution layers on top of each other, the filters of a subsequent layer depend on all the channels of the previous layer. This way, CNNs and fully convolutional networks that consist of multiple convolutional layers can be trained and applied to many complex tasks in computer vision [38].

Pooling

A convolutional layer is typically made up of three distinct steps, the first one commonly performing multiple convolutions, which then yields a set of linear activations [18] marking the end of the first step. In the second step, non-linear activation functions are applied to the linear activations, and in the third step, these activations are further modified using a pooling function.

With a pooling function, an output can be achieved that is approximately invariant to small translations of the input. In other words, most output values yielded by a pooling function given an input that is translated by a small amount, are not changed. This is useful in cases e.g., where the location of a dog in an image is not the priority, but rather whether the image contains a dog. The max pooling function [70], is an example of a pooling function that, given a rectangular neighborhood of outputs as input, returns the output of that neighborhood with the maximum value.

Stride

In Figure 2.4, the filter is shifted over every pixel, resulting in the number of hidden units being equal to the number of pixels. As a model architecture is expanded with more layers, it might be beneficial to reduce the number of hidden units and to only capture the most important information in a prior layer [38]. The filter does not have to shift over every pixel but can instead shift over every third pixel in both the rows and the columns. This would result in the hidden units being a third as many as the pixels. In what fashion to shift the filter is denoted as the filter's stride. The stride can be modified to better suit the task at hand, e.g., the U-Net architecture employs 2x2 max pooling layers with stride 2.

2.5 Recurrent Neural Networks

Much like how CNNs are used to process data with a grid-like topology, Recurrent Neural Networks or RNNs [52] are a family of sequence modeling neural networks used to process sequential data [18]. A sequence can be denoted as $x^{(1)}, \ldots, x^{(\tau)}$, where τ is a time index. Similar to CNNs, which employ parameter sharing between hidden states, RNNs are also constructed based on the concept of parameter sharing. In sequences such as sentences, words can appear in multiple positions. If the sentences, "I bought a house in 1998" and "In 1998, I bought a house" are given as input to a dense neural network (neural network with dense layers) trained on sentences of fixed length, then the parameters for each input variable would be separate from one another. This implies that the rules of sentence construction and the English language would have to be learnt by each parameter at each position of the sentence [18]. By employing parameter sharing, all the parameters would together learn the rules of the language. In the case of RNNs, the parameters are shared and learnt across multiple time steps.

With RNNs, each of the output terms of a hidden unit (state) h is a function of the terms of the previous output of that state. Additionally, the update rule used to produce each term of the output is the same as the previous output. This way of sharing parameters results in a very deep computational graph. A simple RNN is illustrated in Figure 2.5.

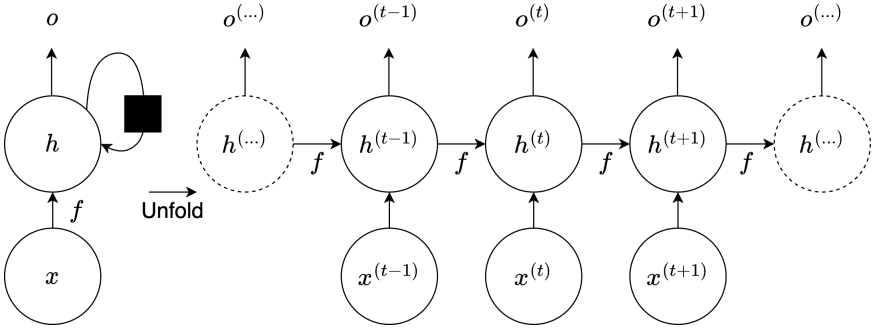

Figure 2.5: A simple RNN with outputs. The input $\boldsymbol{x}$ is processed by state $\boldsymbol{h}$, which in turn is passed forward through time to yield $\boldsymbol{o}$. The indication of the black square is a delay of a single time step. The function $\boldsymbol{f}$ maps the state at time t to $t+1$. The parameters of f are used for all time steps. In the unfolded computational graph, each state node and its output are associated with one input variable at a specific time. Illustration adapted from [18].

A wide variety of recurrent neural networks can be designed based on the ideas of graph-unrolling and parameter sharing. Some examples of RNNs are Long short-term memory (LSTM) and Gated Recurrent Neural Networks (GRUs). Additionally, the RNN illustrated in Figure 2.5 has a causal structure, meaning that a state at time t only depends on information from prior and current input variables [18]. In many applications, a prediction y_t based on the output o_t of a hidden state h_t at time t, may depend on the whole sequence input. Positions at time $t+1$ and beyond, need to also be accounted for when making prediction y_t. Bidirectional RNNs [54] were invented to solve this issue. A Bidirectional RNN combines two RNNs, one that moves forward in the sequence in conjunction with a time step, starting at the beginning of the sequence, and another RNN that traverses backward for each time step, starting at the end of the sequence.

2.6 The Encoder-Decoder and Sequence-to-Sequence Architectures

The RNN mentioned in the previous section maps the input sequence to an output sequence of the same length. However, the input to an RNN, often referred to as the "context", does

not have the same sequence length as the output sequence [18]. A representation C of the "context" is the desired output of a RNN, where C is either a vector or sequence that summarizes the input sequence. Cho et al. [8] and Sutskever et al. [59] independently developed a simple RNN architecture that could map an input sequence of variable-length to an output sequence of another variable-length. The encoder-decoder presented by Cho et al. [8] and the sequence-to-sequence presented by Sutskever et al. [59] helped create an architecture that achieved state-of-the-art translation. The architecture consists of an encoder and a decoder. The encoder is an RNN that when given an input sequence processes the input to yield a context representation C. In general, C is a fixed-size context variable, which is computed from the final hidden state of the encoder. C is then given as input to the decoder, another RNN that is conditioned on the context representation, and produces the output sequence of variable-length. At training time, the encoder and decoder are jointly trained. In Figure 2.6, an example of an encoder-decoder RNN architecture is shown.

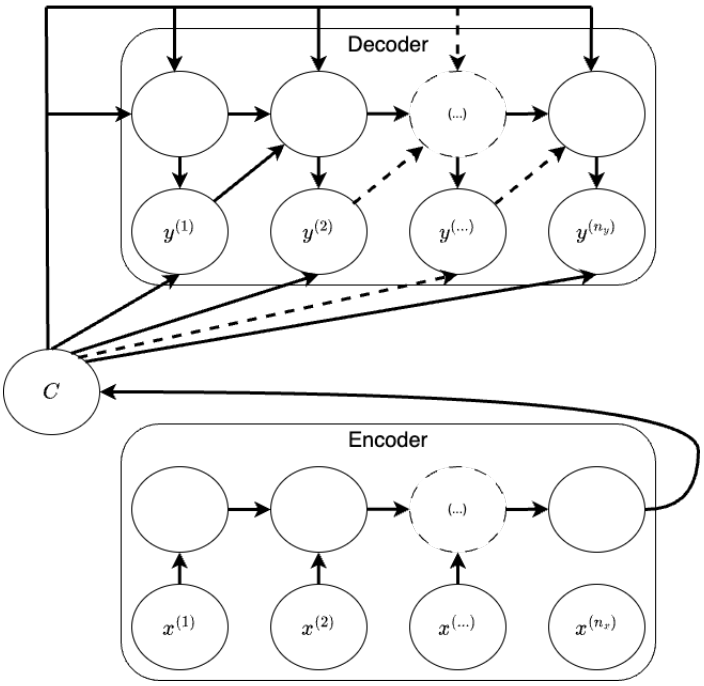

Figure 2.6: A sequence-to-sequence or encoder-decoder RNN architecture. Given the input sequence, $\left[x^{(1)}, \quad x^{(2)}, \quad \ldots, \quad x^{(n_x)}\right]$, of length n_x the architecture is trained to generate the output sequence $\left[y^{(1)}, \quad y^{(2)}, \quad \ldots, \quad y^{(n_y)}\right]$, of length n_y. Illustration adapted from [18].

2.7 Semantic Segmentation

In computer vision, image segmentation is a key task [47]. As the task name suggests, image segmentation is about partitioning images into multiple objects and segments. A subtask of image segmentation is semantic segmentation, where each pixel of an image is classified with a semantic label. Given object categories such as e.g., bird, car, or horse, every pixel of an image is classified as one of the object categories. In comparison to whole image classification, where an entire image is classified with a single label, semantic segmentation is generally a more demanding undertaking [47]. Models typically used to solve semantic segmentation tasks usually consist of an encoder-decoder architecture [20]. SegNet [1], illustrated in Figure 2.7, is one such architecture that combines a convolutional encoder and decoder for semantic pixel-wise segmentation.

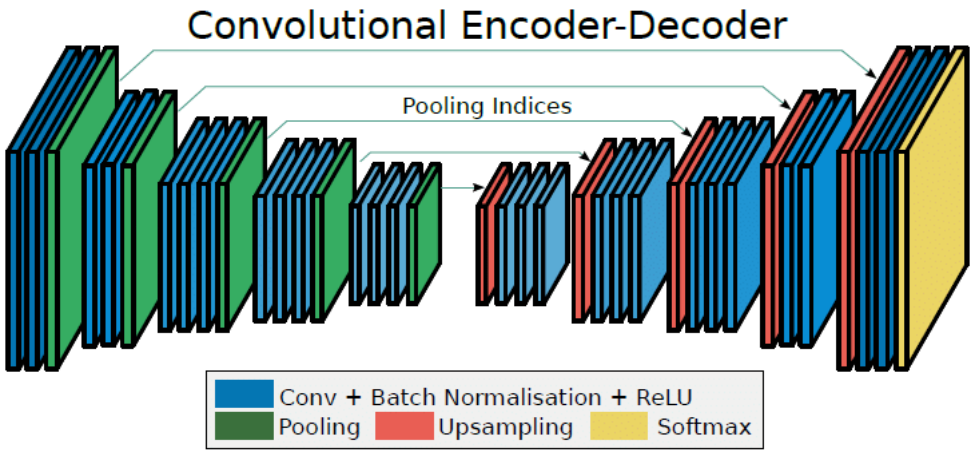

Figure 2.7: The SegNet architecture. Image obtained from [1].

2.8 The CNN Based Architectures

In this thesis, a comparative analysis is conducted, focusing on two Transformer based architectures and two CNN based architectures. The following sections provide an overview of the CNN based architectures analyzed.

U-Net

The U-Net architecture, first introduced in the paper *U-Net: Convolutional Networks for Biomedical Image Segmentation* by Ronneberger et al. [51], builds upon the fully convolutional network [41], to make more correct segmentations than previously used architectures on the downstream task of semantic segmentation. Additionally, the architecture is constructed in a way so that it would be able to train on a set of very few images, while still achieving more precise segmentations. The U-Net architecture is shown in Figure 2.8 and consists of a contracting path and an expansive path, which gives the architecture its U-shape and in turn its name.

In the initial level of the U-Net and the first step of the contracting path, the input 2D image is passed through two unpadded convolutions each with a (3×3)-filter, and each followed by a ReLU activation layer. The output of the last activation layer is then passed to a (2×2) max pooling function with stride 2 to yield the output of the first level. The max pooling operation results in a downsampling of the activation layer's output by a factor of 2, from (x, y) to $(\frac{x}{2}, \frac{y}{2})$. The same combination of unpadded convolutions, activation layer, and max pooling operation is then repeated three more times in the contracting path. Furthermore, for each step in the contracting path, the number of channels is increased by a factor of 2 from the previous level.

The output of the contracting path is then, at the "bottom" of the architecture, passed through two (3×3)-filter convolutions, each followed by a ReLU. There is however no max pooling function applied at this point in the architecture.

Similar to the contracting path, the expansive path also consists of four steps and levels. At every step, an upsampling of the feature map is performed coupled with a (2×2)-filter convolution ("up-convolution"). At this point, the number of feature channels is cut in half due to the up-convolution. In addition to the previous operations, a cropped feature map from the corresponding level in the contracting path is concatenated with the halved feature map. Finally, the newly concatenated feature map is passed through two (3×3)-filter convolutions, each followed by a ReLU. At each step, the same procedure is performed, and after

the fourth step in the expansive path, a (1×1) convolution is applied to map each of the feature vectors to a specified number of classes.

The (2×2) max pooling operations are to be applied to layers with an even x- and y- size. Therefore, it is important to choose the size of the input image accordingly. Furthermore, the cropping that takes place in the expansive path is necessary because border pixels are lost in every convolution operation [51].

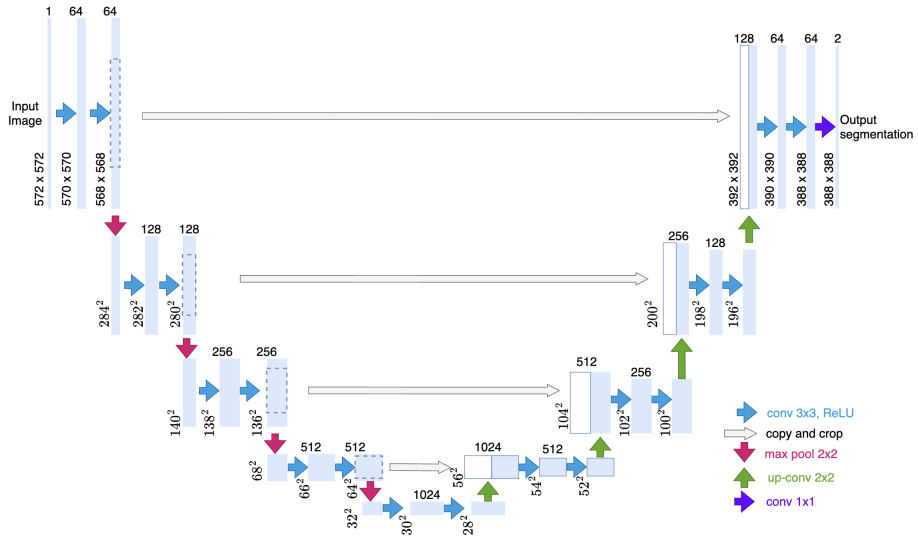

Figure 2.8: The U-Net architecture. Blue boxes represent a feature map of multiple channels and white boxes correspond to copied feature maps. Above each box, the number of channels is denoted, whereas the size of the input of each box is found at the bottom of the boxes. Illustration adapted from [51].

nnU-Net

U-net is a firmly established general-purpose network for 2D and 3D Medical Image Segmentation. However, the design and implementation of U-Net-based architectures for specific tasks is non-trivial, especially for 3D biomedical imaging and it depends heavily on dataset properties, hardware, expertise, and experience [39]. A solution for these challenges is proposed by Isensee et al. [28, 26] with a robust and self-adapting framework called nnU-Net. This framework automatically configures itself for any new task and can generate state-of-the-art segmentation masks on both 2D and 3D images. A modified nnU-Net has performed particularly well on the task of Brain Tumor Image Segmentation, finishing first in the BraTS 2020 segmentation challenge [27]. Furthermore, Isensee et al. applied a nnU-Net with an unmodified baseline configuration to the segmentation task of the BraTS 2020 challenge and achieved respectable results as well.

With the nnU-Net framework, 2D U-nets and 3D U-nets can be acquired, using deep supervision learning and a dataset-adapted pipeline [26]. Multiple steps are incorporated into the adapted training pipeline such as pre-processing, data augmentation, five-fold cross-validation, ensembling combinations of different U-Net configurations, and post-processing. This pipeline can be configured and by omitting all the aforementioned steps, except the pre-processing step, the framework can yield hyperparameters adapted to the pre-processed dataset, without having to employ the other steps. In this manner, a pure 2D U-Net or 3D U-Net can be trained on the dataset-adapted hyperparameters, without employing the other steps.

The pre-processing phase employed in the nnU-Net framework consists of a few steps that modify the data used for training. Cropping is performed on each training sample, modifying the data so that it is focused on the region of interest containing the image foreground [26]. This results in the data size being decreased, while still keeping the relevant information present in the data and lowering the computational burden of training the model on the non-cropped data.

Voxel spacing is not inherently understood by CNNs. The U-Nets in the nnU-Net learn spatial semantics by enabling resampling of the voxel spacings of all the patient data to the median of the respective dataset.

In this thesis, only the pre-processing step of the nnU-Net framework is used to yield dataset-specific fingerprints to train 2D U-Nets trained with a deep supervision learning method. The reason for removing several steps from the nnU-Net framework is to facilitate a reasonable comparison between the models trained during this thesis.

2.9 Attention Mechanism

For sequence modeling networks such as RNNs, LSTMs, and GRUs, a key assumption is that each current state h holds information related to the entire input previously looked at. The final state in a sequence modeling network would then, after having looked at the whole input sequence, contain the complete information relating to the sequence. This imposes a rather strong condition on the network which makes it more difficult to work with long sequences [2]. In the paper *Neural Machine Translation by jointly learning to align and translate*, Bengio et al. [2] present an approach to solving this issue. The approach is referred to as the attention mechanism, and the main idea behind it is to, instead of the final hidden state, look at all the hidden states that correspond to the whole input in order to yield an output. Which hidden state requires how much attention needs to be learnt. In a general encoder-decoder approach, the decoder only looks at the final state of the encoder in order to make a prediction. In contrast to that, an attention-based encoder-decoder approach leverages the attention mechanism to look at all the hidden states at each time step and deduces which of the hidden states is more informative for making a prediction [2]. Figure 2.9 visually depicts the difference between the two approaches.

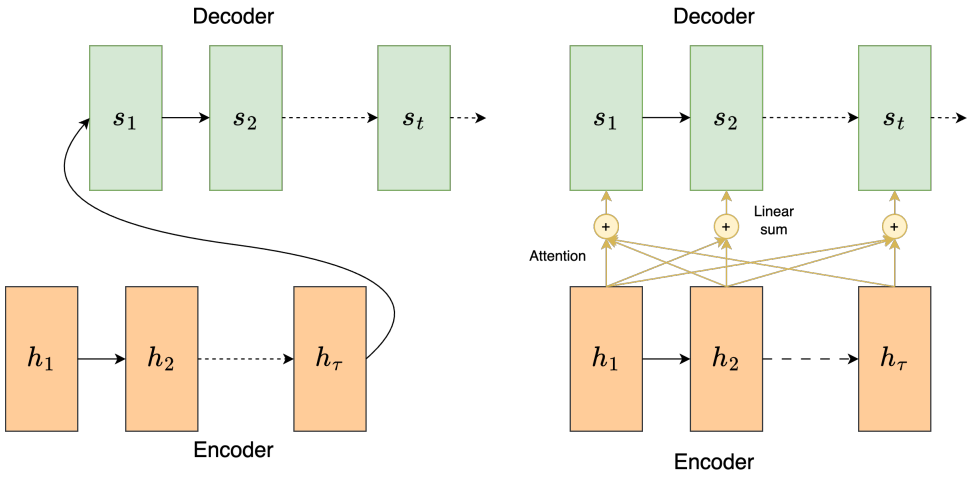

Figure 2.9: An illustration of the difference between an attention-based encoder-decoder (right) and a normal encoder-decoder architecture (left). In each time step t of the decoder (right), the input to s_t is a linear combination of the hidden unit's output h_τ. Image adapted from [67].

To learn which hidden states to turn the attention to and to what degree, a neural network can be employed. In their paper, Bengio et al. [2] proposed an architecture that incorporates the attention mechanism together with a bidirectional RNN encoder and a decoder that during the translation decoding emulates searching through a source sentence. The full architecture is illustrated in Figure 2.10, where each attention weight $\alpha_{t,j}$ (value deciding the degree of attention) at time step t of the decoder is calculated as a function of h_j and the previous hidden state of the decoder s_{t-1}.

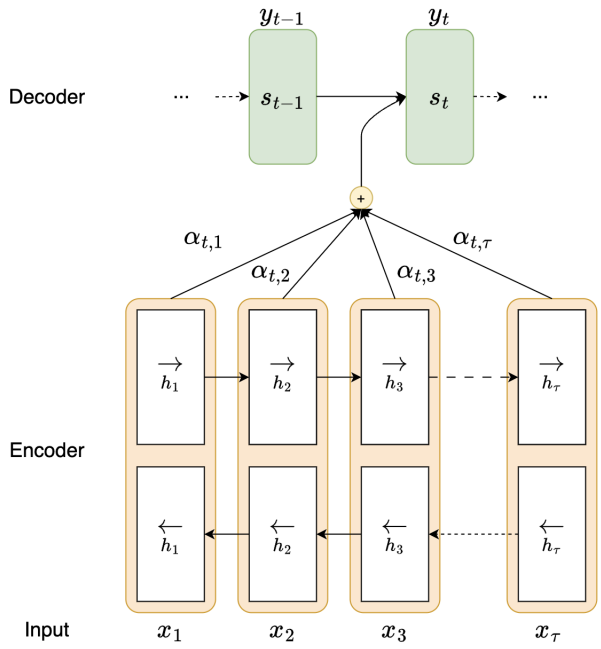

Figure 2.10: An illustration of the model proposed in the paper: *Neural Machine Translation by jointly learning to align and translate*. The $\alpha_{t,j}$ denotes the attention weight for h_j used to compute s_t at time step t. The attention weights are learnt by a feedforward neural network. Illustration adapted from [67].

2.10 Transformers

Introduced in the paper *Attention is all you need* by Vaswani et al. [64], the Transformer architecture solely leverages the attention mechanism, avoiding recurrence and convolution. Compared to RNNs, LSTMs, and GRUs, the Transformer architecture is well-suited for parallelization. The most competitive neural sequence-to-sequence models prior to the Transformer follow an encoder-decoder structure [8, 2, 59]. The Transformer architecture also follows this structure, leveraging so-called self-attention layers and point-wise dense layers stacked on top of each other in both the encoder and decoder. Self-attention is an attention mechanism that looks at a single input sequence, relating different positions of that sequence to one another, and computing a sequence representation. In the Transformer architecture, self-attention is used both inside the encoder and decoder, but not in between. The encoder (left), the decoder (right) and the full architecture can be seen in Figure 2.11.

Scaled Dot-Product Attention

In their paper [64], Vaswani et al. an attention function is interpreted as the mapping of a query and a set of key-value pairs to an output. The weights in this description of the attention function are computed by a compatibility function of the query with the corresponding

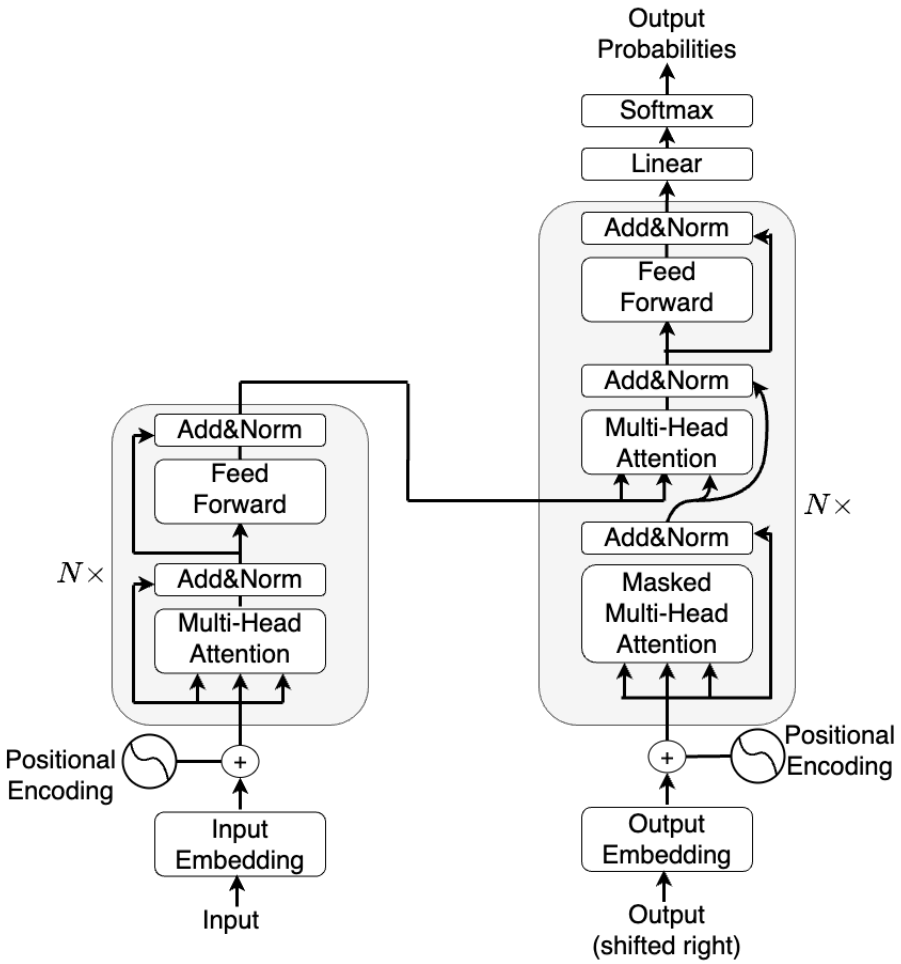

Figure 2.11: The Transformer architecture. Illustration adapted from [64].

keys. Each value with its associated weight is then used to calculate the output through a weighted summation.

Scaled-dot-product attention is the utilized attention mechanism in the Transformer architecture and is illustrated in Figure 2.12. Queries and keys of dimension d_k, and values of dimension d_v are passed to the attention function. The weights for each value in the set are computed by calculating the dot-product of the corresponding query with all corresponding keys, dividing the result with $\sqrt{d_k}$, and then passing it to a softmax function. The architecture does this calculation on multiple queries and sets of key-value pairs. The matrix Q contains all the respective queries, while the corresponding keys and values are packed into matrices K and V. The attention function is described as,

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.12}$$

Multi-Head Attention

Rather than applying a single attention function, multiple attention functions can be applied in parallel on queries, keys, and values of dimension d_{model}. This is achieved by first linearly projecting the queries, keys, and values h times, using different trained linear projections, to the dimensions d_k, d_k, and d_v respectively. The scaled-dot-product attention is then applied to

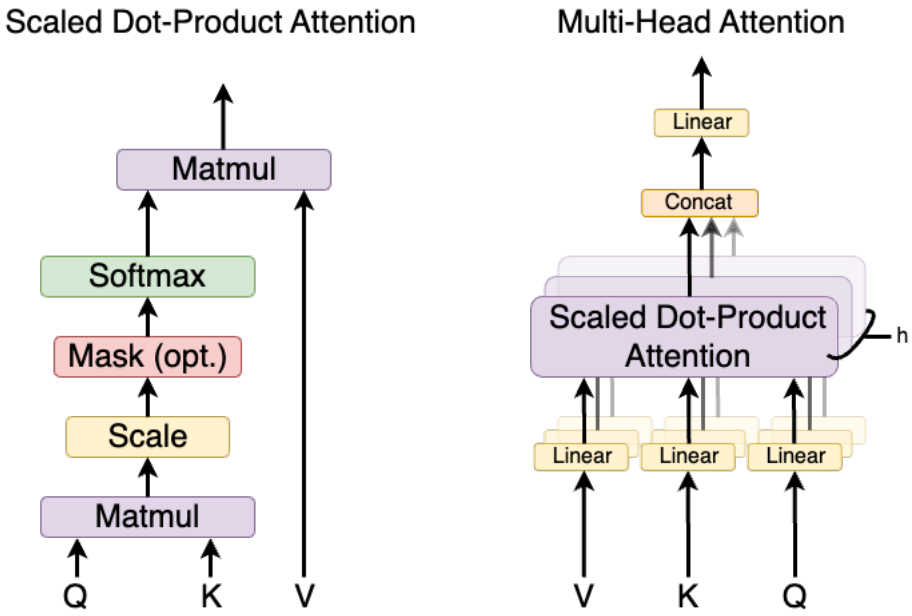

Figure 2.12: Scaled-Dot-Product Attention (left). Multi-Head Attention with (right) h attention layers executed in parallel. Illustration adapted from [64].

each of the linear projected queries, keys, and values concurrently resulting in output values of dimension d_v. By concatenating these outputs and doing an additional linear projection, the final output values are computed. This calculation is referred to as multi-head attention and is illustrated in Figure 2.12. The equation for multi-head attention is,

$$Multi\text{-}Head\text{-}Attention(Q,K,V) = Concat(head_1, head_2, \ldots, head_h)W^O$$

$$\text{where} \quad head_i = Attention(QW_i^Q, KW_i^K, VW_i^V). \tag{2.13}$$

In Equation 2.13 above, the projections denoted as, W^O, W_i^Q W_i^K, and W_i^V are parameter matrices such that $W^O \in \mathbb{R}^{hd_v \times d_{model}}$, $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ holds.

By employing multi-head attention, information from various positions in different representation subspaces can be jointly attended to. In comparison, single attention is restrained to do this due to averaging. The number of parallel attention layers (heads) used in the Transformer architecture is $h = 8$, and for each head $d_k = d_v = d_{model}/h = 64$.

Embeddings

Sequence transduction models usually employ so-called learned embeddings to map each token of a sequence to a high-dimensional vector of continuous numbers [15]. When using text as input data, each word has its own word embedding. By looking at the distribution of a word in a text, a representation of a word's meaning can be learnt. In the Transformer architecture, learnt embeddings are employed to yield vectors of dimension d_{model} from input and output tokens [64].

Positional Encodings

By omitting convolution and recurrence, the Transformer model architecture cannot make use of the order of sequences in the traditional way [64]. Instead, information such as the tokens'

relative and absolute position in sequences is injected through positional encodings. The encodings are added to both the encoder embeddings and decoder embeddings. The dimension of the embeddings and the positional encodings are both d_{model}, allowing the summation of the two to be computed.

2.11 The Transformer Based Architectures

In the context of this thesis's comparative analysis, two Transformer based architectures are analyzed. The upcoming sections present an overview of the analyzed Transformer based architectures.

Vision Transformer

A few years after its introduction, Transformer based architectures had become the de facto standard in natural language processing (NLP), but its application possibilities in computer vision were limited [13]. The Vision Transformer (ViT), introduced by Dosovitskiy et al. in the paper *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* [13], leverages much of the original Transformer architecture to perform very well on image classification tasks when applied to a sequence of image patches. While the original Transformer architecture takes a sequence of text split into tokens as input, ViT takes an image that similarly is split into sub-images referred to as patches as input. These patches are handled in the same way as word tokens in NLP. The sequence of patches is then used to yield a corresponding sequence of linear embeddings which is then fed to a Transformer encoder. The ViT architecture as well as a Transformer encoder are illustrated in Figure 2.13.

The ViT architecture is initially fed 2D images of shape, $\boldsymbol{x} \in \mathbb{R}^{H \times W \times C}$ as input, where C refers to the number of channels, and (H, W) is the resolution of the input image. These images are then used to construct a sequence of flattened 2D images reshaped into $\boldsymbol{x} \in \mathbb{R}^{N \times P^2 \cdot C}$, where (P, P) is the dimension of each patch and $N = \frac{HW}{P^2}$ is the number of patches that also corresponds to the sequence length. Throughout all the layers in the Vision Transformer, a constant latent vector of size D is employed. Thus, the sequence of flattened patches is linearly projected through a learnable linear projection layer to yield the patch embeddings mapped to D dimensions. Additionally, to retain positional information, positional embeddings are added to the patch embeddings. Furthermore, a learnable embedding is prepended to the sequence of patch embeddings, that after passing through L-layers of the Vision Transformer encoder, constitutes the image representation. A multi-layer perceptron (MLP) head, also referred to as a densely connected network head, is attached to the outputted image representation in order to infer a class to the input image. In their paper, Dosovitskiy et al. [13] present three configurations of ViT, which are shown in Table 2.1.

Model	Layers	Hidden Size D	MLP size	Head	Params
ViT-Base	12	768	3072	12	83M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 2.1: ViT Variants

Dosovitskiy et al. note that Transformers trained on mid-sized datasets such as ImageNet [11] which contains 1.3M images, without employing strong regularization, reach modest accuracies on the task of image classification in comparison to a residual network (ResNet) [23] of comparable size, that attains accuracies a few percentages better. The authors attribute this to the fact that Transformers, compared to CNNs, do not inherently possess a

strong inductive bias, like locality and translation equivariance inherent to CNNs. Because of this, Transformers, when trained on insufficient amounts of data, do not generalize well. However, Dosovitskiy et al. show that the Transformers perform comparably or better than state-of-the-art on multiple image recognition benchmarks when the models are trained on 14M-300M images.

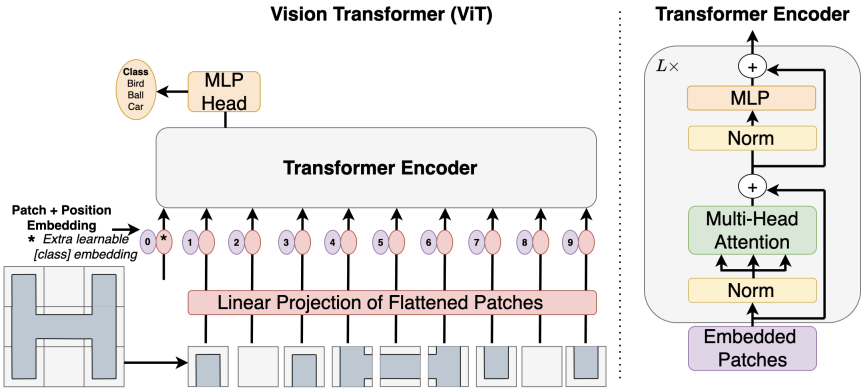

Figure 2.13: Vision Transformer (ViT) Architecture (left), Transformer Encoder (right). Illustration adapted from [13].

Swin Transformer

In 2021, one year after the introduction of ViT, Liu et al. introduced a new Transformer architecture in their paper: *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* [40]. The Swin Transformer has a resemblance to ViT and the input to this architecture, as it is for ViT, is a sequence of patches. Adapting the Transformer architecture from NLP to the field of computer vision comes with a few challenges, mainly that the scale of objects in images have a large variation and that the pixels of images has a higher resolution compared to words in a text. Although ViT successfully adapted and applied a Transformer architecture on image classification, it is inadequate as a more general-purpose backbone in computer vision [40]. The computational complexity of ViT increases quadratically with image size and in its architecture, ViT generates low-resolution feature maps, making it unfit for dense computer vision tasks or when working with high-resolution input images. The Swin Transformer was created by Liu et al. with the purpose of addressing the challenges of adapting a Transformer architecture to vision while also creating an architecture capable of serving as a general-purpose backbone network in computer vision. A key implementation feature of the Swin Transformer is that it employs so-called shifted windows between subsequent self-attention layers. This provides the architecture with connections between windows of consecutive layers, and thus with enhanced modeling power. Furthermore, the Swin Transformer computes the self-attention locally in each window, compared to ViT which computes self-attention globally. The computational complexity of the Swin Transformer, instead of increasing quadratically with image size, increases linearly with image size. In Figure 2.14, the shifted window approach is illustrated.

Another key design feature of the Swin Transformer is the hierarchical feature maps that are created throughout the architecture. These hierarchical feature maps allow the architecture to model at various scales. The hierarchical feature maps are created by reducing the number of patches through the merging of 2×2 patch neighborhoods as a greater depth of the network is reached. In contrast, in the ViT architecture, the generated feature maps are of a single low-resolution. In Figure 2.15, the difference between the two architectures is shown.

At the time of its introduction, the Swin Transformer achieved state-of-the-art results on the object detection dataset COCO [37], image classification on ImageNet-1K [11] and on

ADE20K semantic segmentation [69], demonstrating that a Transformer based model can be applied on dense vision tasks as well as non-dense tasks [40].

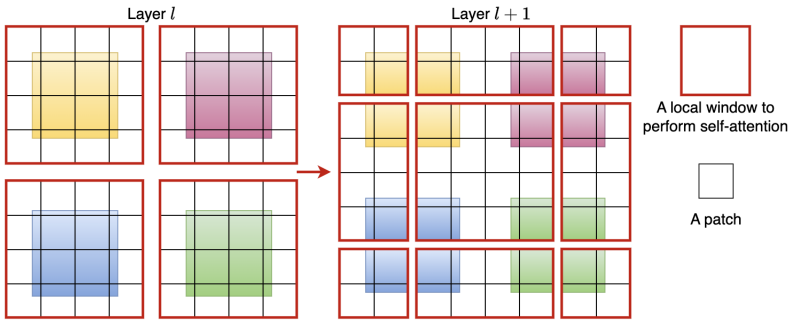

Figure 2.14: An example of how self-attention is computed using the shifted window approach in the Swin Transformer architecture. A typical window partition scheme is performed in layer l (left), computing self-attention within each of the windows. The subsequent layer (right), $l+1$, employs shifting in the window partitioning that yields windows that overlap the boundaries of the windows in layer l. This way, the self-attention computation of layer l and $l+1$ are provided a connection. Illustration adapted from [40].

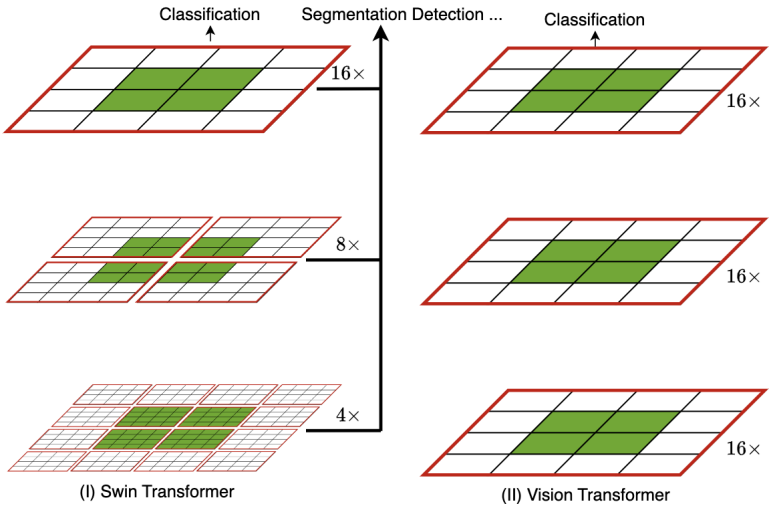

Figure 2.15: An illustration of how the Swin Transformer (right) and the Vision Transformer (left) create their respective feature maps. The Swin Transformer creates hierarchical feature maps by merging patches (outlined with black borders) as greater depth in the network is reached. The self-attention is computed locally within each window (outlined with red borders). ViT creates feature maps of a single low-resolution and computes the self-attention globally. Illustration adapted from [40].

The Architecture

Like the Vision Transformer, multiple variants of the Swin Transformer architecture with different sizes and parameter values exist. The Swin Transformer architecture is shown in Figure 2.16, illustrating the tiny version of the Swin Transformer. Given an RGB input image, the image is split into non-overlapping patches similar to how patches are produced in the ViT architecture and then passed to the Swin Transformer architecture. The Swin Transformer consists of 4 stages, and like in ViT, the patches are treated like "tokens". The features of each

patch are set as the concatenation of the raw RGB values of the patch. The patch size used for the Swin Transformer is 4×4, which in turn yields a feature dimension of $4 \times 4 \times 3 = 48$ for each patch. The features are then passed to a linear embedding layer, projecting it to an arbitrary dimension C. Throughout the Swin Transformer architecture, Transformer Encoders with altered self-attention computation, denoted as "Swin Transformer Blocks", are applied to the patch tokens. The number of patch tokens $(\frac{H}{4} \times \frac{W}{4})$ is unchanged after being passed to a Swin Transformer Block and the linear embedding layer coupled with 2 subsequent Swin Transformer blocks make up "Stage 1".

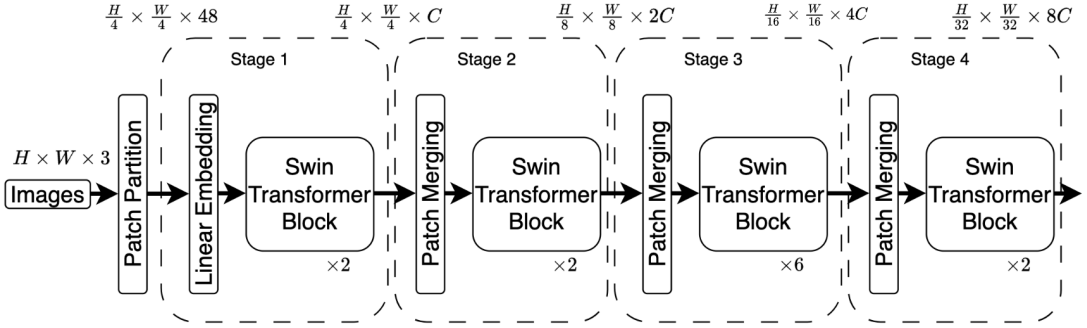

Figure 2.16: The Swin Transformer architecture (Swin-Tiny). Illustration adapted from [40].

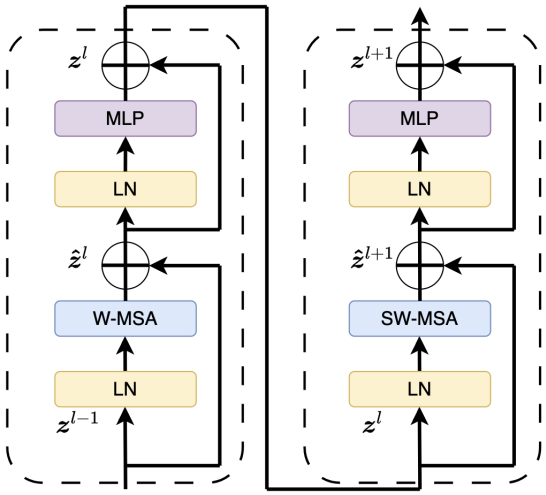

Figure 2.17: Two Swin Transformer blocks in succession. Illustration adapted from [40].

As previously mentioned, the hierarchical representation is produced by merging a patch and its neighborhood as greater network depth is reached, creating new patches of larger dimensions. This is done by applying a so-called patch-merging layer. In the first patch-merging layer, every 2×2 patch neighborhood and their respective features are concatenated, forming new and fewer patches of size 8×8 with a feature vector of dimension $4C$. Additionally, each $4C$-dimensional feature vector is passed to a linear layer, resulting in the number of tokens being reduced by a multiple of $2 \times 2 = 4$ (downsampling of the resolution by a factor of 2) and the feature vector dimension being set to $2C$. The patch tokens are then passed to a Swin Transformer block, where feature transformation is applied, all while keeping the resolution at $(\frac{H}{8} \times \frac{W}{8})$. The first patch merging step together with two Swin Transformer Blocks for feature transformation makes up "Stage 2".

Stage 3 uses the same procedure used in Stage 2, with the difference being that 6 Swin Transformer Blocks are applied in succession to the patch tokens and that the resulting output resolution is $(\frac{H}{16} \times \frac{W}{16})$. Stage 4 also employs the same procedure in stage 2, with 2 Swin Transformer Blocks and an output resolution of $(\frac{H}{32} \times \frac{W}{32})$. In conjunction with the tiny variant of the Swin Transformer architecture shown in Figure 2.16, Liu et al. present three additional variants of the Swin Transformer as seen in Table 2.2.

Architecture	C	Stage 1.	Stage 2.	Stage 3.	Stage 4.
Swin-Tiny	96	2	2	6	2
Swin-Small	96	2	2	18	2
Swin-Base	128	2	2	18	2
Swin-Large	192	2	2	18	2

Table 2.2: Swin Transformer Variants. Each stage column denotes the number of Swin Transformer blocks in that stage [40].

The Swin Transformer Block

The Swin Transformer Block is a modified Transformer Encoder, where the Multi-Head Self Attention part of the Transformer Encoder is replaced by a Shifted Window Multi-Head Self Attention module denoted as SW-MSA. All other parts of the Transformer Encoder are kept the same in the Swin Transformer Block, and in Figure 2.17, two successive Swin Transformer blocks are illustrated.

2.12 UPerNet

The Unified Perceptual Parsing Network (UPerNet) is a versatile framework that can accurately segment various concepts from images by utilizing different vision backbone networks [66]. The framework can for instance be utilized with a Swin Transformer and was used together with a Swin Transformer as the backbone network to achieve, at the time, state-of-the-art results on ADE20K semantic segmentation. The UPerNet and its network leverage something called a Feature Pyramid Network (FPN) [36]. FPN is a feature extractor that utilizes a hierarchical and pyramidal structure to effectively extract multi-level features from images. Using a top-down architecture and lateral connections, FPN integrates high-level semantic information into middle and lower levels of feature representation with minimal additional cost.

In the UPerNet architecture, Xiao et al. incorporate a pyramid pooling module (PPM) from PSPNet [68] on the last layer of the backbone network before its output is passed to the FPN [66]. The author found that the FPN architecture works well in combination with the PPM, due to the PPM capturing effective global prior representations. Further details on PPM and FPN can be found in [68] and [36] respectively.

2.13 Optimization Methods

Finding the minimum or maximum of an *objective function* is referred to as optimization [38]. The problem of finding the maximum of an objective function is the same as finding the minimum of the negative objective function, therefore it is only necessary to look at the minimization problem. In machine learning, optimization is mainly applied while a model is being trained and the cost function is minimized in regard to the model parameters $\boldsymbol{\theta}$. The objective function corresponds to a cost function, denoted $f(\theta)$, whereas the parameters of

the models are the optimization variables. Examples of optimization methods are Gradient Descent and Stochastic Gradient Descent.

Adam Optimizer

The Adam optimizer [31], is an efficient stochastic optimization method that builds on the advantages of the two popular methods AdaGrad [14] and RMSProp [60, 61]. Adam is an algorithm that optimizes stochastic objective functions using first-order gradient descent-based optimization that uses adaptive estimates of lower-order moments. The algorithm can be implemented without difficulties, while also being computationally efficient and not taxing on memory [31]. The updating of weights is done following the Adam update rule,

$$\theta_t = \theta_{t-1} - \mu \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.14}$$

where θ_t denotes the weights at time t, ϵ is a small value introduced to avoid division by zero, and μ is the learning rate at that time. Furthermore, the variables $\hat{m}_t$ and $\hat{v}_t$, which denote the estimated exponential moving average of the gradient and the squared gradient respectively, are defined by the following equations,

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot (\nabla_\theta f_t(\theta_{t-1}))^2, \tag{2.15}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot \nabla_\theta f_t(\theta_{t-1}), \tag{2.16}$$

The parameters β^1 and β^2 are an engineer-defined exponential decay rate and $\nabla_\theta f_t(\theta_{t-1})$ is the computed gradient for objective function f.

AdamW

By adding weight decay regularization to the Adam algorithm, where the weight decay is decoupled from the optimization step taken w.r.t the loss function, the AdamW optimizer is obtained [42]. Weight decay regularization added to Equation 2.14 results in the following,

$$\theta_t = \theta_{t-1} - \mu \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_{t-q}, \tag{2.17}$$

where λ is the weight decay value specified by the machine learning engineer. AdamW considerably improves the generalization performance of Adam.

3 Method

This chapter describes the method employed to generate results for this thesis. It introduces the dataset used in the study, along with the steps taken for preparing the data to make it suitable for our experiments. Furthermore, the four model architectures used in this thesis are introduced together with our approach to obtain training, test, and validation data splits that preserve the original distribution of glioma labels. Finally, we provide a description of the two experiments conducted in this project.

3.1 The Data

This thesis conducted its experiments on the BraTS 2020 dataset provided by The MultiModal Brain Tumor Segmentation Challenge 2020 [45, 7, 6, 4, 5]. The BraTS 2020 dataset contains 3D multi-parametric MRI (mpMRI) scans of 660 subjects and the mpMRI scans are provided by several institutions using different clinical protocols and MRI scanners. For each subject, there are four 3D MRI modalities: native (T1) and post-contrast T1-weighted (T1Gd), T2-weighted (T2), and lastly T2 Fluid Attenuated Inversion Recovery (T2-FLAIR) volumes. The mpMRI scans in the BraTS 2020 dataset are resampled to the same isotropic resolution of 1x1x1 mm, while also being rigidly aligned and skull-stripped. The input size of the images is 240x240x155. Additionally, the mpMRI scans are manually segmented by one to four raters, and the specific annotations are approved by board-certified neuro-radiologists. The scans are annotated with three tumor sub-regions: the GD-enhancing tumor (ET), the necrotic tumor core (NCR/NET), and the peritumoral edematous (ED). Furthermore, combining the sub-regions ET, NCR/NET and ED yield three more sub-regions. These additional sub-regions are Whole Tumor (WT), Tumor Core (TC), and Enhancing Tumor (ET). The training, validation, and test set comprises 369, 125, and 166 subjects respectively. In total, the BraTS 2020 dataset contains 2640 mpMRI scans, 1476 scans in the training set, 500 scans in the validation set, and 664 scans in the test set.

For the 2020 BraTS Brain Tumor Segmentation Challenge, only the training set was initially made available, including correct segmentation masks for each subject. Unfortunately, the test set including correct segmentation masks, is not publicly available. The validation set was published sometime after the BraTS challenge was held, yet it does not contain correct segmentation masks. Without correct segmentation masks for the validation and set, an evaluation using the validation set is not feasible in our work. As only the training set includes

correct segmentation masks for the subjects it is the only data used in this thesis. The data used in this thesis consists of the training data from the BraTS 2020 dataset with the ground truth segmentation annotations: GD-enhancing tumor (ET — label 4), the peritumoral edema (ED — label 2), and non-enhancing tumor core (NCR/NET — label 1). To showcase how the Brain Tumor images in BraTS 2020 dataset are structured, 2D slices in the axial plane related to four different subjects contained in the training data from the BraTS 2020 dataset are illustrated in Figure 3.1.

3.2 Data Preparation

The images provided in the BraTS 2020 dataset are 3D images of dimension $240 \times 240 \times 155$. Since the Swin Transformer and Vision Transformer take 2D images as input, the 3D images and corresponding 3D ground truth segmentation needs to be converted into 2D slices. Thus, each image is split in the axial plane to generate a set of 155 2D slices corresponding to each original 3D image. Furthermore, the range of pixel intensity values of the 2D slices is normalized to range from 0 to 255. Additionally, out of all the slices, there are some slices that contain too few pixels of brain tissue. Based on a threshold value, the slices that have too few brain tissue pixels are discarded. As a result, the sets of slices, that correspond to subject 3D images, differ in size.

Another necessary preparation step is changing how each class is labeled. The MMSegmentation and the nnU-Net library requires the data to be labeled differently. Thus, every 2D ground truth segmentation is changed to have label 1 represent NCR/NET, label 2 represent ED, and label 3 represent ET. Everything else in the ground truth segmentation is labeled as 0 and is during training treated as a separate fourth class. To further accommodate the Swin Transformer and Vision Transformer models, all the slices are resized to 256x256. From this point on, the corresponding 2D slices of the training data from the BraTS 2020 dataset with 369 total subjects are referred to as the BRATS20 dataset.

3.3 Data Splitting

The BRATS20 dataset is made up of 2D slices from 3D images each corresponding to one out of 369 different subjects. Each 3D image and consequently each 2D slice is labeled with either LGG or HGG, in order to indicate the Glioma grade. The percentage of subjects with LGG and HGG in the BRATS20 dataset is 20.6% (76 subjects) and 79.4% (293 subjects) respectively. This means that the BRATS20 dataset is imbalanced, containing a lot more slices annotated as HGG When splitting the BRATS20 dataset into a train subset, a validation subset, and a test subset, the proportion of HGG and LGG occurrences in each subset needs to be the same as in the initial, unsplit BRATS20 dataset. An even distribution needs to be assured in order to prevent the evaluation from being biased.

By employing stratified splitting on the subject level, the train, validation, and test subset can be constructed in a way that preserves the data distribution as in the original BRATS20 dataset. For this thesis, five different dataset splits are performed, each with a unique randomized seed to shuffle and split the dataset. Each of the five dataset splits is generated by splitting the BRATS20 dataset using a stratified split. For each dataset split we obtain a train subset consisting of 70% of the subjects together with their corresponding set of 2D slices, and a validation and test subset consisting of 15% of the subjects and their set of 2D slices, respectively. Both of the experiments in this thesis are carried out on the same dataset splits.

3.4 The Models

The four model architectures compared in this thesis are the Vision Transformer (ViT), Swin Transformer, a typical 2D U-Net, and a 2D U-Net trained using the nnU-Net framework. All

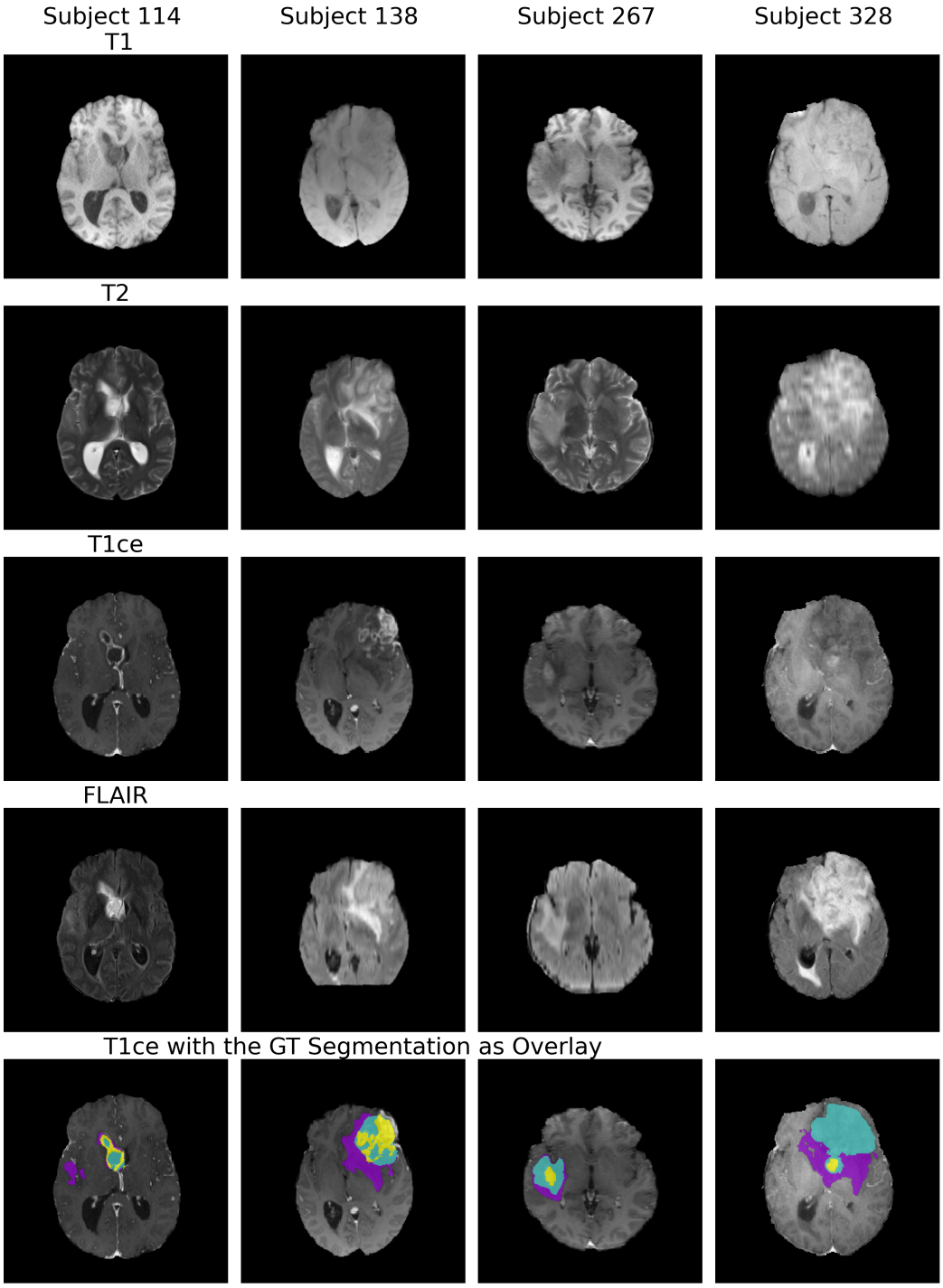

Figure 3.1: A showcase of axial slices pertained to T1, T1ce, T2, and FLAIR 3D MRI modalities from four subjects from the BraTS 2020 dataset. Subject 114 and 138 are labeled with HGG, whereas Subject 267 and Subject 328 are annotated as LGG. The first four rows display T1, T1ce, T2, and FLAIR MRI axial slices (slice 70) of each subject. In the fifth row, the corresponding brain tumor segmentation axial slice of each subject is overlaid on the T1ce axial slice. Peritumoral edema (ED) is highlighted in violet, the non-enhancing tumor core (NCR/NET) in turquoise, and the GD-enhancing tumor (ET) in yellow.

four models are implemented with Python. Both ViT and the Swin Transformer are implemented and trained using the MMSegmentation library [10]. The 2D U-Net model has the same architecture as the original U-Net [51] and is implemented in PyTorch [50], while the nnU-Net based 2D U-Net model is implemented with the help of the nnU-Net framework. For each of the models, during training, the number of classes is set to 4, where the fourth class is made up of all pixels not labeled as any of the foreground classes, NCR, ED, or ET.

The 2D U-Net

The 2D U-Net implementation follows the U-Net architecture described in the *U-Net: Convolutional Networks for Biomedical Image Segmentation* by Ronneberger et. al [51] and the model is implemented using the PyTorch library [50].

The 2D nnU-Net

The team behind the nnU-Net framework released an updated version of their framework towards the end of this thesis. Thus, the current version of the nnU-Net framework is not used to implement the nnU-Net based 2D U-Net model, instead, it is implemented using the prior version, nnU-Net V1.

Swin Transformer

The architecture used for the Swin Transformer is the Swin-Base variant implemented in MMSegmentation with window size 7 and patch size 4×4. Since the original Swin Transformer was implemented with 3-channel RGB images in mind, the number of input channels to the model needs to be changed to 4. This is necessary for the architecture to be compatible with the four modalities associated with each slice of the MRI scans. The dimension of the input to the Swin Transformer is adjusted to $256 \times 256 \times 4$, where 4 is the number of channels, and 256 is both the height and width of each modality slice. The only modification we make to the original Swin-Base is changing both the number of channels and the number of classes to 4.

Vision Transformer

The architecture used for the Vision Transformer is the ViT-Base variant implemented in MMSegmentation, with patch size 16×16 with input size set to 256×256. The number of input channels is changed to 4 from 3 for the same reason as for the Swin Transformer. The number of classes is adjusted to 4 again.

Leveraging Transformers for Semantic Segmentation

In the paper first introducing the Swin Transformer, Liu et al. apply a Swin Transformer for semantic segmentation on the ADE20K dataset [40]. This is done through the utilization of the UperNet framework [66] in MMsegmentation [10]. The Swin Transformers as well as the Vision Transformers in this thesis are also trained using the same framework in MMSegmentation.

3.5 Evaluation Metrics

Each model architecture is evaluated based on evaluation metrics. The metrics calculated in this thesis are presented in the following sections.

Jaccard Index

The Jaccard Index [29], also referred to as the Jaccard similarity coefficient or the Intersection over Union (IOU) metric, is an evaluation metric commonly used in computer vision to measure the similarity between two images. The metric is defined as the size of the intersection of two finite sets A and B, divided by the size of the union of the sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.1}$$

Given two images, the Jaccard Index between the images is the area of overlap between the two images, divided by the total number of pixels in both images. The more similar the two images are, the more overlap between them, which results in a higher Jaccard Index. The Jaccard Index is defined in the interval $[0, 1]$, while also satisfying the triangle equality. In the field of semantic segmentation, the Jaccard Index can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth segmentation.

The Dice Coefficient

The Dice Coefficient, also known as the Sørensen-Dice index, Sørensen index, or the Dice score is a variant of the Jaccard Index and was first introduced by the botanists Lee Raymond Dice in 1945 [12] and Thorvald Sørensen in 1948 [57]. Like the Jaccard Index, the Dice score is used to measure the similarity between two finite sets, such as two images. The Dice Coefficient formula is defined as:

$$J(A,B) = \frac{2|A \cap B|}{|A| + |B|} \tag{3.2}$$

where A and B are two finite sets being compared [58]. Similarly, to the Jaccard Index, when the overlap between the two finite sets increases, the metric value also increases. The Dice Coefficient is defined in the interval $[0, 1]$, and unlike the Jaccard Index, does not satisfy the triangle equality. The Dice Coefficient can, just like the Jaccard Index, be used to gauge how good a model-generated predicted segmentation mask is in comparison to a corresponding ground truth segmentation of the same sample used to generate the prediction.

Confusion Matrix and Classification Performance Metrics

Problems that relate to classifying data correctly are denoted as classification problems and the models that are employed to solve the classification problems are called classifiers [33]. In relation to a classification problem and a classifier, a confusion matrix is the summarization of the classifier's performance with respect to the classification problem and a given set of sample data [62]. An example of a confusion matrix for a 4-Class classification task, with the classes A, B, C, and D is presented in Table 3.1. The confusion matrix is of two dimensions and is indexed by the actual class in one dimension and the predicted class in the other dimension. The first column of the confusion matrix states that $5 + 1 + 3 + 2 = 11$ predictions are of A while only 5 of the predictions are correct. The first row of the confusion matrix indicates that $5 + 6 + 0 + 0 = 11$ samples are classified as A, but only 5 of the 11 samples are correctly classified. The diagonal of this matrix shows how many correct classifications are done for each class respectively. For binary classification problems, the confusion matrix can be utilized and designed in a different way as shown in Table 3.2. In this context, one class is denoted as the positive class and the other class is denoted as the negative class. Furthermore, the four cells in the confusion matrix are denoted as true positive (TP), false negative (FN), false positive (FP), and true negative (TN).

Many classification performance metrics are defined and can be computed from the given confusion matrix and its terms.

	Predicted			
Actual	**A**	**B**	**C**	**D**
A	5	6	0	0
B	1	10	5	3
C	3	2	7	4
D	2	1	3	3

Table 3.1: Confusion Matrix for a 4-Class Classification problem

	Predicted	
Actual	**Positive**	**Negative**
Positive	**TP**	**FN**
Negative	**FP**	**TN**

Table 3.2: Confusion Matrix for Binary Classification

Precision and Recall

In the field of information retrieval, precision, and recall are two metrics used to measure how well an information system performs in returning relevant documents pertaining to a query [63]. Precision is defined as the number of relevant documents retrieved from an information system given a query divided by the total number of retrieved documents given the same query. Recall is defined as the number of relevant documents retrieved from an information system given a query divided by the total number of relevant documents in the information system. These metrics can also be defined using the terminology previously described for a 2-class confusion matrix. The metrics are defined as follows.

$$\text{Precision} = \frac{\text{True positives}}{\text{Total number of predicted positives}} = \frac{TP}{TP + FP} \tag{3.3}$$

$$\text{Recall} = \frac{\text{True positives}}{\text{Total number of actual positives}} = \frac{TP}{TP + FN} \tag{3.4}$$

In the field of Semantic segmentation, precision and recall can be used to measure how well the pixels making up the predicted segmentation are classified when compared to the ground truth segmentation.

3.6 Loss Functions for Semantic Segmentation

In this thesis, the DiceCELoss function is employed during the training of all model architectures. In the following sections, the DiceCELoss function is introduced together with its components.

Cross Entropy Loss

A commonly used loss function in machine learning is the cross entropy loss function [38]. The binary cross entropy (BCE) loss function is defined as,

$$\text{BCELoss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i) \tag{3.5}$$

where N is the total number of samples, y_i is the true label for the i-th sample and it can take a value of 0 or 1. p_i is the predicted probability of the positive class (class 1) for the i-th sample. When dealing with multiple or arbitrary number of classes, the multi-class cross entropy loss functions is defined as,

$$\text{CELoss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij}), \tag{3.6}$$

where C is the number of classes, y_{ij} is the binary indicator (0 or 1) for whether the i-th sample belongs to class j, and p_{ij} is the binary indicator for whether the i-th sample is predicted to belong to class j.

Dice Loss

The Dice Coefficient can directly be used to train a segmentation model using a so-called dice loss function [46], which can assess the model's predicted segmentation masks during training time when a ground truth segmentation is available [58]. Between two binary 2D images the Dice Coefficient is defined as,

$$D = \frac{2\sum_{i=1}^{N} y_i p_i}{\sum_{i=1}^{N} {y_i}^2 + \sum_{i=1}^{N} {p_i}^2}, \tag{3.7}$$

where N is the number of pixels present in both the ground truth image y and the predicted image p. The subsequent dice loss function to minimize during training is thus defined as,

$$\text{BinaryDiceLoss} = 1 - \frac{2\sum_{i=1}^{N} y_i p_i + \epsilon}{\sum_{i=1}^{N} {y_i}^2 + \sum_{i=1}^{N} {p_i}^2 + \epsilon}, \tag{3.8}$$

where ϵ is a smoothing factor introduced to avoid division by zero, i.e., when a class is absent in both the ground truth segmentation and the predicted segmentation. As for the cross entropy loss function, this can be extended to scenarios where the number of classes is more than two. The Dice loss for multiple classes is defined as,

$$\text{DiceLoss} = 1 - \frac{1}{C}\sum_{c=1}^{C} \frac{2\sum_{i=1}^{N} y_{ic} p_{ic} + \epsilon}{\sum_{i=1}^{N} y_{ic}^2 + \sum_{i=1}^{N} p_{ic}^2 + \epsilon}, \tag{3.9}$$

where C is the number of classes, y_{ic} is the ground truth mask for class c at pixel i, and p_{ic} is the predicted mask for class c at pixel i.

DiceCELoss

The dice and cross entropy loss can be combined into a new loss function, here referred to as the Dice Cross Entropy Loss. This loss function has previously been used, i.e., by Iseensee et al. when they trained a nnU-Net on brain tumor images from the BraTS 2020 challenge [27]. In the Dice Cross Entropy loss function, both the dice loss and cross entropy loss are calculated independently and then summed to form the loss value. How much each of the losses should contribute to the overall loss can be decided by weighing them differently. The Dice Cross Entropy Loss is defined as,

$$\text{DiceCELoss} = weight_{dice} \cdot \text{DiceLoss} + weight_{ce} \cdot \text{CELoss}, \tag{3.10}$$

where $weight_{dice}$ and $weight_{ce}$ are the respective weightings of the losses which default to 1.

3.7 Data Augmentation

For computer vision tasks pertaining to medical imaging, ethics, and data regulation make it difficult to acquire large amounts of medical images for the purpose of training models. In the absence of a sufficient number of training images, data augmentation can be used to increase the amount of data available for training [48, 9]. By using data augmentation during a model's training, the original data, based on a user-defined probability, is augmented to introduce a "new" data sample. Examples of data augmentation approaches used on either 2D or 3D brain tumor images for brain tumor segmentation are rotating, vertical or horizontal flipping, elastic deformation, brightness adjustment, and scaling [48, 9]. The data augmentation implemented in this thesis is utilizing all five of the aforementioned methods.

3.8 The Experiments

Two experiments are conducted in this thesis. Experiment 1 investigates the performance of each model architecture when the amount of data used to train the models is increased, while Experiment 2 investigates if employing data augmentation during training time results in better segmentations. Both of the experiments are conducted using five dataset splits of the BRATS20 dataset and share the same core structure, except for some key differences. The first experiment is more extensive than the second experiment, it involves training a total of 120 models, 30 for each of the aforementioned model architectures.

The Parameters

All four model architectures are trained using the same parameters whenever possible. The loss function employed by each model during training is the DiceCELoss function, with both the weights for the dice loss and the cross entropy loss being equal to 1. Furthermore, the Dice Loss component does not take into account the background class, label 0. The loss per batch is derived by calculating the loss for each training image and taking the mean loss value.

The AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and weight decay $\lambda = 0.01$ is leveraged by each model. Additionally, a learning rate scheduler with warmup and linear decay is employed. The warmup ratio is set to $1e^{-6}$, and the learning rate starts at $0.00006 \cdot 1 \cdot 10^{-6} = 6 \cdot 10^{-11}$. For the first 1500 iterations, the learning rate is increased and after 1500 warmup iterations, the learning rate reaches 0.00006. For the rest of the training, the learning rate is decreased linearly until it reaches 0.0 at the end of the training. This is illustrated in Figure 3.2. The models are trained for 15 epochs, and after every epoch, a validation loss is calculated. The batch size is 8, and the last batch is dropped if it is not the same size as all of the other batches to ensure that all models are provided with batches of consistent size. To ensure a fair comparison between the models, the number of workers is fixed at 1.

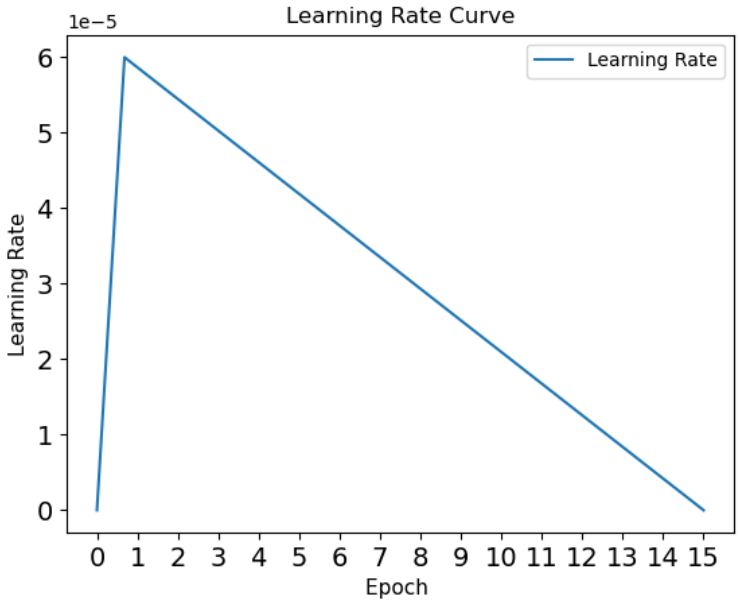

Figure 3.2: Plot of the learning rate curve for all models.

Training Specifications

As stated in the previous section, all models are trained for 15 epochs using a learning rate scheduler with warmup to update the learning rate. A NVIDIA GeForce RTX 3090 is used to train all the models and after each epoch, the model parameters of the model thus far are saved. The total training time as well as the GPU memory usage during training is recorded. The time it takes to fetch and load the image from memory is not measured. In Experiment 2, data augmentation techniques are employed during training.

Data Augmentation Pipeline

The data augmentation techniques applied in Experiment 2, consist of five different types of augmentations for each of the four model architectures. The choice of data augmentation techniques is inspired by the work of Eklund et al. in [9]. The five chosen data augmentation techniques are:

- **Rotation:** The training image with its four modalities and corresponding ground truth segmentation is rotated counterclockwise with an angle randomly chosen from a uniform distribution ranging from 0°to 30°.
- **Scaling:** The training image with its four modalities and corresponding ground truth segmentation is subjected to scaling on each axis by a factor randomly selected from a uniform distribution within a range of $\pm$20%.
- **Flipping:** The training image with its four modalities and corresponding ground truth segmentation is horizontally flipped with a probability of 50% or vertically flipped with the same probability.
- **Brightness Adjustment:** The training image with its four modalities and corresponding ground truth segmentation is brightness adjusted. A power-law γ intensity transformation is used to adjust the brightness. In a power-law γ intensity transformation, the gain (g) and γ parameters are chosen randomly from a uniform distribution ranging between 0.8 to 1.2. The intensity (I) is then randomly altered using the formula $I_{new} = g \cdot I^{\gamma}$.

- **Elastic Deformation:** The training images with their four modalities and corresponding ground truth segmentation are subjected to elastic deformation. Specifically, the elastic deformation used relates to the elastic deformation from the original U-Net paper [51]. A square deformation grid is used for the elastic deformation, and the displacements are sampled from a normal distribution with a standard deviation of $\sigma = 2$ voxels. A spline filter with order 3 is applied in each dimension to smooth the deformation.

During a model's training, there is a 25% probability of applying data augmentation to a training image during training time. In the case of data augmentation being applied, only one out of the five data augmentations techniques are chosen to augment the training image. Which data augmentation technique to use is decided randomly at training time, the techniques are selected with a probability of 20%.

Experiment 1: Semantic Segmentation on BRATS20 With Increasing Amounts of Training Data

Experiment 1 consists of five runs. For each run, the BRATS20 dataset is split into a new train, validation, and test subsets according to the splitting method with different randomized seeds, as described in section 3.3. In order to facilitate a comparison between models trained on different amounts of data, we additionally use six different partitions of the train subset. These partitions include progressively larger portions of data from the initial train subset. Each of the six partitions consists of 50%, 60%, 70%, 80%, 90%, and 100% of the subjects in the train subset. All of these partitions are created with a stratified split, as described earlier. As done before, we employ a unique randomized seed to shuffle and partition the train subset. All four model architectures are trained using the six partitions of training data. Per run, this amounts to 24 models being trained, 6 per model architecture. In Figure 3.3, how the data is split for each run is illustrated.

The validation and test subset are the same throughout a run. Therefore, per run, every model is evaluated and validated using the same validation and test subset. As Experiment 1 consists of five runs, a total of 120 models are trained. Throughout the remainder of this thesis, a naming convention is used to refer to a specific set of data. For example, 2dslices_50_001 indicates a set of training data that includes 2D slices corresponding to 50% of the subjects in a train subset, together with the corresponding test and validation data subsets for run 1 (001).

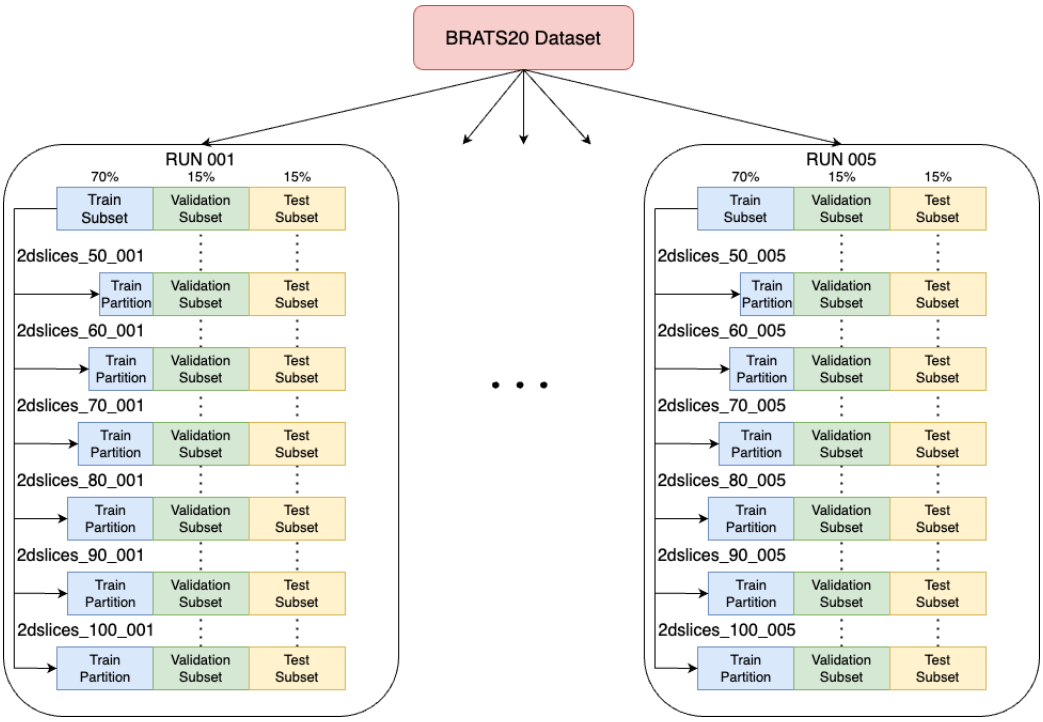

Figure 3.3: Illustration of how the data of each run is structured.

Experiment 2: Semantic Segmentation on BRATS20 With Data Augmentation

The same data used for the five runs in Experiment 1 is also used in Experiment 2. However, in Experiment 2, no additional partitions of the train subsets are created. We always use training data corresponding to 100% of the subjects in each train subset per run. Specifically, only the training data contained in 2dslices_100_001, 2dslices_100_002, 2dslices_100_003, 2dslices_100_004 and 2dslices_100_005 is used to train the four different model architectures. This results in a total of 20 different models. In Experiment 2, data augmentation is added to the training scheme and employed during training time. With a probability of 25%, a 2D slice, which is our training image, is altered by modifying its four modalities and the corresponding ground truth segmentation during training, effectively constructing "new" training data.

3.9 Evaluation

For Experiment 1 and Experiment 2, the evaluation procedure is performed in the same way. For each of the models trained in an experiment, we want to determine the best set of model parameters to use. The selection is based on identifying which parameters lead to the lowest validation loss. For instance, a Swin Transformer trained for 15 epochs on the dataset 2dslices_50_001 reaches the lowest validation loss at epoch 13, so the chosen parameters to be used for evaluating the model are the ones reached after training for 13 epochs. Meanwhile, a 2D U-Net trained on the same dataset might reach the lowest validation loss at epoch 15, which then results in the parameters at epoch 15 being chosen as the model's parameters used for evaluation.

After having chosen and documented the best parameters for each model, the models are used to produce 2D segmentations on their corresponding test subset. For each of the subjects in the test subset, all the predicted 2D segmentations associated with the subject are then concatenated to create a 3D segmentation. The predicted 3D segmentation and the

ground truth segmentation of each subject in the test subset are then used to compute several evaluation metrics. One of the evaluation metrics calculated is the Dice Coefficient. The Dice Coefficient is calculated per class (NCR/NET, ED, ET) and the mean Dice Coefficient of all classes is also computed. Furthermore, precision, recall, and the Jaccard Index are calculated using the procedure described above. Per model, all these values are then averaged over all subjects in the test subset. Additionally, per model, all of the evaluation metrics and their average are calculated for the test subjects with LGG and HGG respectively. To obtain representative and comparable values for the different model architectures trained on different amounts of training data, additional averages are computed. For each model architecture, we group the models obtained through the five runs by the size of the data they are trained on. For each of these groups, we compute the average of their models' evaluation metrics. How the average evaluation metrics of each run is calculated is illustrated in Figure 3.4.

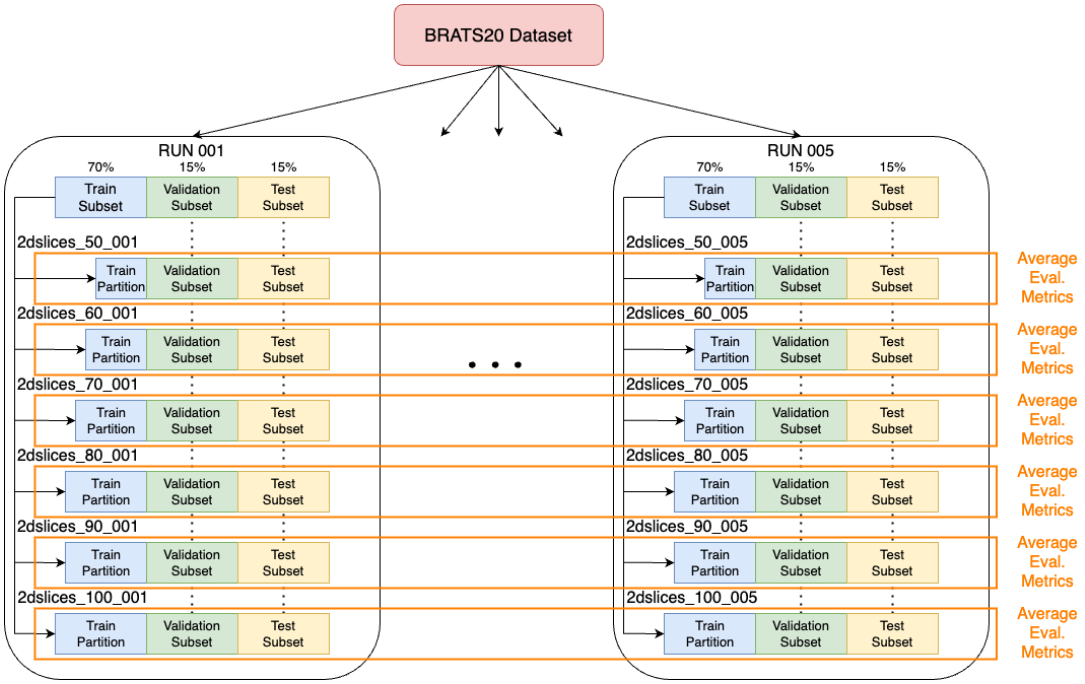

Figure 3.4: Illustration of how the average of each evaluation metric of the five runs is calculated. This averaging is done for all four model architectures, resulting in average evaluation metrics for each of the model architectures trained on increasing amounts of data.

4 Results

In this chapter, we present the outcomes of each experiment conducted within this thesis. We begin by highlighting the results of Experiment 1, followed by a subsequent presentation of the results obtained from Experiment 2.

4.1 Experiment 1

In Experiment 1, a total of 120 models, 30 per model architecture, are trained using the training procedure described in section 3.8. Training time, GPU memory usage, and validation loss are monitored for each of the models. The epoch with the lowest validation loss for each model is recorded, and the model parameters from that epoch are used to generate segmentation masks on the corresponding test data. These masks are then compared to ground truth segmentation masks to compute evaluation metrics including Dice Coefficient, Jaccard Index, precision, and recall for each model. Furthermore, the time it takes to run forward propagation on all 2D slices in the test data is observed for each model architecture. The next sections present a curated selection of the most noteworthy results from Experiment 1.

Training Time

The total training time of each model trained for 15 epochs is recorded. For each model architecture, the average total training time is calculated for all models trained on the same percentage of training subjects. Figure 4.1 illustrates the average total training time for each model architecture as the training data size increases.

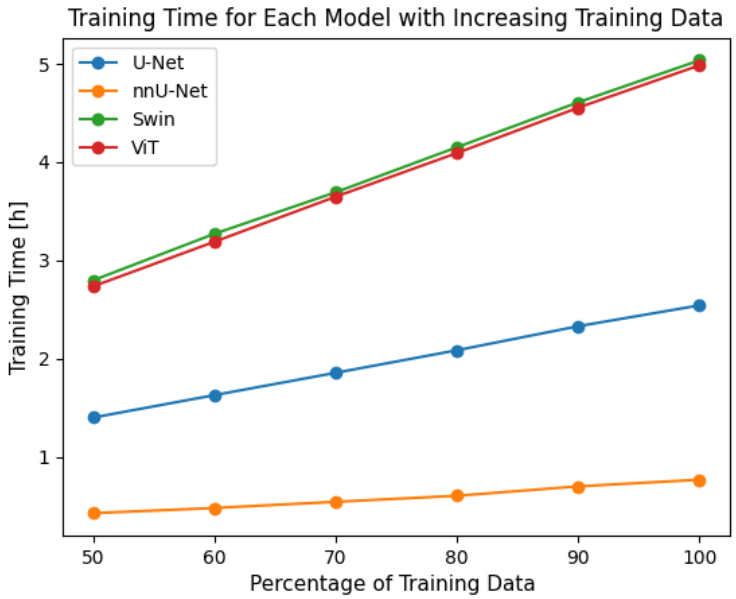

Figure 4.1: Average total training time for each model architecture trained for 15 epochs on an increasing amount of data. Each value is the average of five runs, and the total training time is measured in hours.

GPU Memory Usage

The GPU memory usage for each of the four model architectures is measured during training using the *nvidia-smi* command. The result for each model architecture is shown in Figure 4.2. The number of trainable parameters for the 2D U-Net, the Swin Transformer and the Vision Transformer is approximately 31M, 121M, and 144M, respectively.

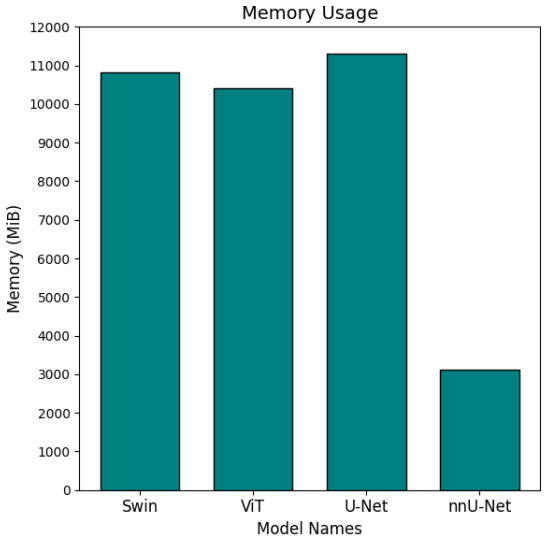

Figure 4.2: GPU memory usage for each model architecture during training, measured and reported in mebibytes (MiB).

Validation Loss

For each of the 120 models, the training and validation loss is recorded during training. The validation and training loss for each of the four model architectures, trained on training data contained in 2dslices_50_001 and 2dslices_100_001, respectively, are illustrated in Figures 4.3-4.10. For the rest of this thesis, we use a naming convention to refer to a particular model that is trained on specific data from a particular run. For example, a ViT model trained on the training data in 2dslices_50_001 is denoted as ViT_50_001.

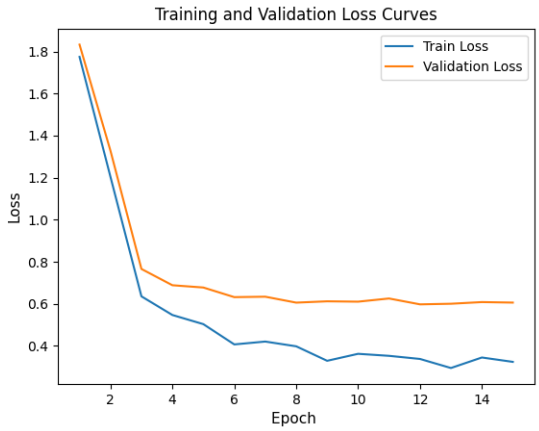

Figure 4.3: Validation and training loss of ViT_50_001, trained using training data contained in 2dslices_50_001.

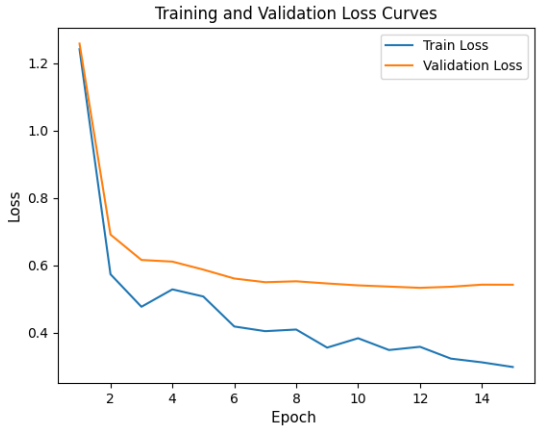

Figure 4.4: Validation and training loss of ViT_100_001, trained using training data contained in 2dslices_100_001.

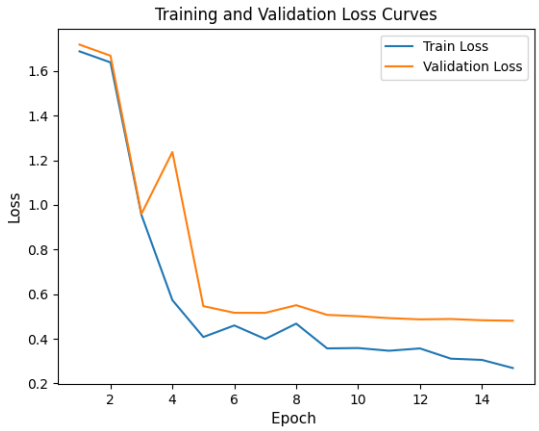

Figure 4.5: Validation and training loss of Swin_50_001, trained using training data contained in 2dslices_50_001.

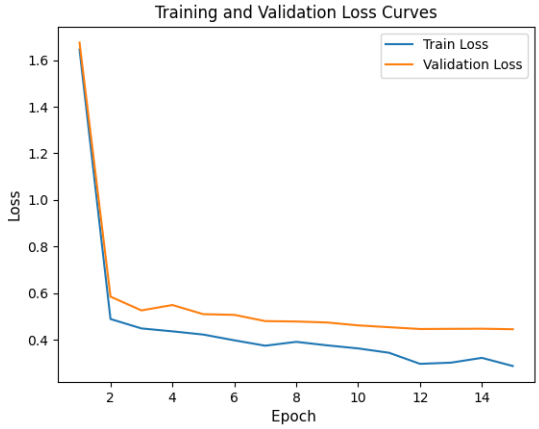

Figure 4.6: Validation and training loss of Swin_100_001, trained using training data contained in 2dslices_100_001.

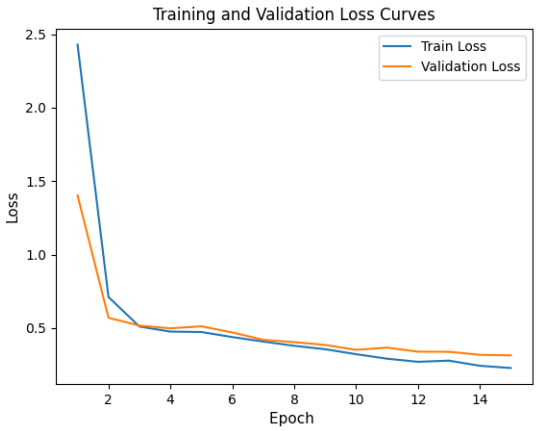

Figure 4.7: Validation and training loss of U-Net_50_001, trained using training data contained in 2dslices_50_001.

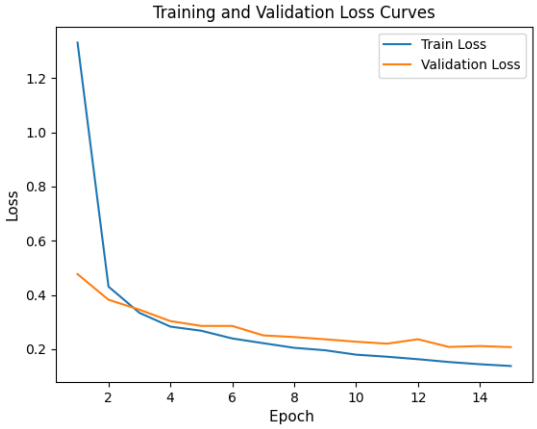

Figure 4.8: Validation and training loss of U-Net_100_001, trained using training data contained in 2dslices_100_001.

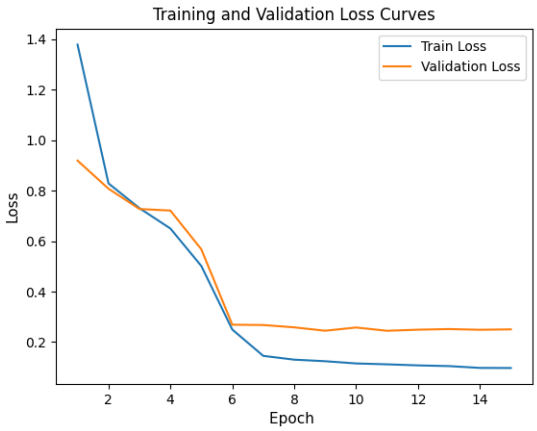

Figure 4.9: Validation and training loss of nnU-Net_50_001, trained using training data contained in 2dslices_50_001.

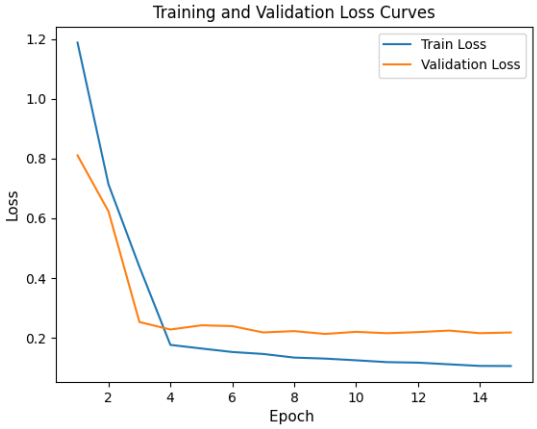

Figure 4.10: Validation and training loss of nnU-Net_100_001, trained using training data contained in 2dslices_100_001.

Segmentation Time

The time it takes for each model architecture to perform forward propagation on all slices of test data and save the generated segmentation mask predictions is measured. These measurements are obtained by running predictions on the test data of run 001 five times and then computing the average. In Table 4.1, the time it takes for each model architecture on average to yield segmentation masks on test data is shown.

Model	Total Segmentation Time [s]	Segmentation Time per Slice [s]
ViT	200.9932	0.0266
Swin	253.7729	0.0335
U-Net	293.6427	0.0388
nnU-Net	271.6665	0.0359

Table 4.1: The time it takes for each respective model architecture on average to execute forward propagation on all slices contained in the test data. This time measurement includes the time it takes to save the generated segmentation masks. The segmentation time is recorded when producing segmentation masks on the test data of run 001. The measuring of segmentation time is performed five times and then the average segmentation time is computed.

Predicted Segmentation Masks

In this section, a comparison of predicted segmentation masks and ground truth segmentation masks is showcased. The predicted segmentation masks are generated by each of the four model architectures trained on the training data and evaluated on the test data contained in 2dslices_100_001. The predicted and corresponding ground truth segmentation masks are slices in the axial plane from subjects 114, 138, 267, and 328 in the test data. Subjects 114 and 138 are annotated with HGG and Subjects 267 and 328 are labeled with LGG. In Figure 4.11, the predicted segmentation masks generated by ViT, Swin, U-Net, and nnU-Net, and the corresponding ground truth segmentation masks are shown.

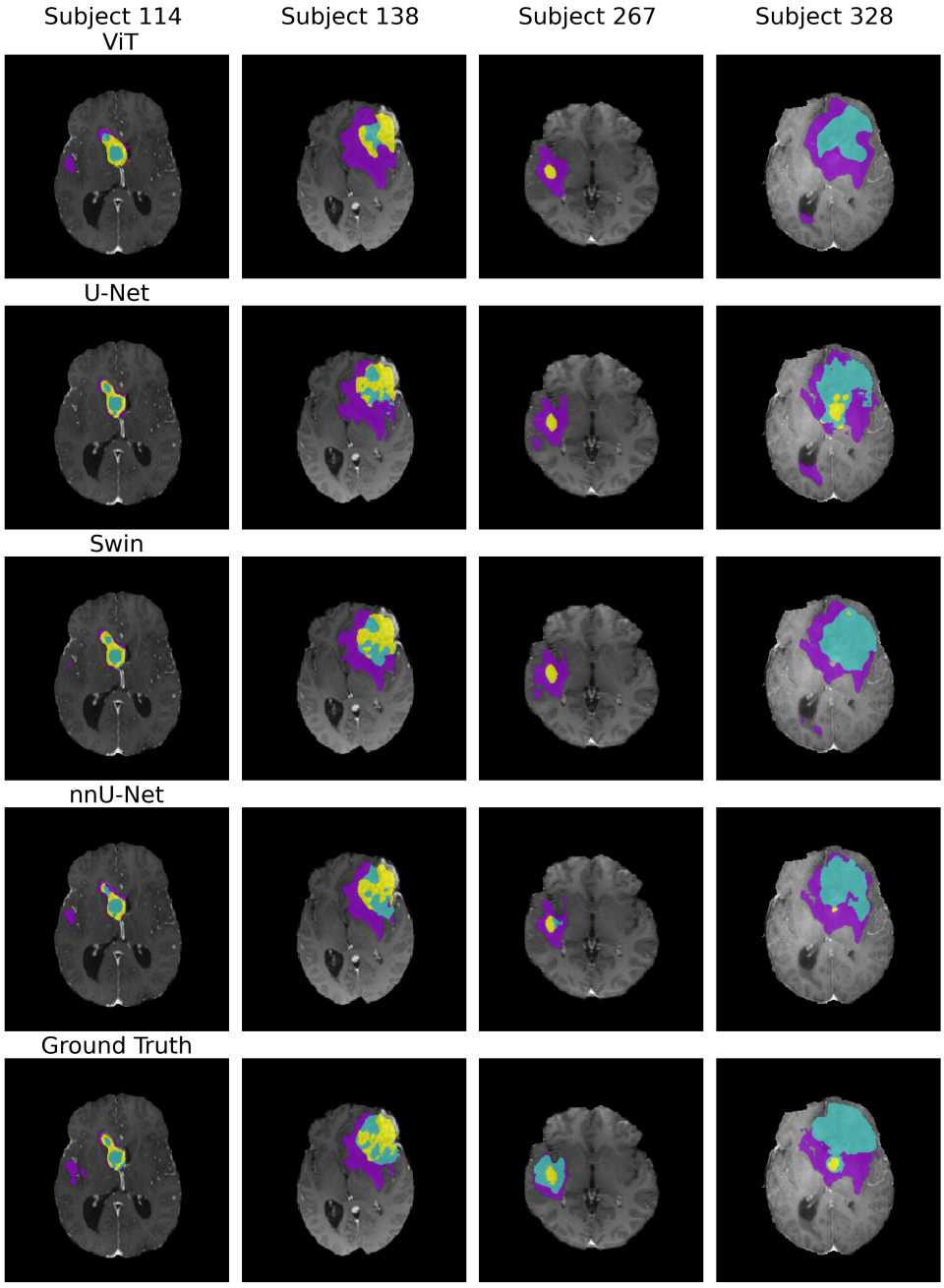

Figure 4.11: Experiment 1: A comparison of predicted segmentation masks from all four model architectures and ground truth segmentation masks of four subjects in the BRATS20 dataset. All models are trained on training data and evaluated on test data from 2dslices_100_001. The comparison is made between slices in the axial plane (slice 70) of Subjects 114, 138, 267, and 328. Subject 114 and 138 are annotated with HGG, whereas Subject 267 and Subject 328 are labeled as LGG. Peritumoral edema (ED) is highlighted in violet, the non-enhancing tumor core (NCR/NET) in turquoise, and the GD-enhancing tumor (ET) in yellow.

Dice Coefficient

For each of the 120 trained models, 30 per model architecture, the average Dice Coefficient for each class, as well as the mean Dice Coefficient of all classes are calculated. This is performed

on the corresponding test data of each of the models. We then proceed to calculate another average over all average Dice Coefficients for each class of the models of each of the four architectures trained on the same percentage of the training data. The resulting values of this procedure are presented in the first three columns of Tables 4.2-4.5 together with the average standard deviation. The fourth column in the aforementioned tables contains the average over the mean Dice Coefficient for all classes which are again grouped by the amount of training data.

To facilitate an overview and comparison of the results of all four model architectures, we additionally provide Figure 4.12.

ViT

DC	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT_50	0.395±0.277	0.632±0.208	0.541±.255	0.523±0.268
ViT_60	0.379±0.271	0.615±0.209	0.528±0.256	0.504±0.268
ViT_70	0.395±0.271	0.632±0.199	0.553±0.251	0.527±0.262
ViT_80	0.394±0.273	0.633±0.195	0.573±0.249	0.533±0.263
ViT_90	0.412±0.272	0.638±0.196	0.562±0.252	0.537±0.261
ViT_100	**0.420**±0.271	**0.649**±0.190	**0.579**±0.249	**0.549**±0.258

Table 4.2: Average Dice Coefficient values for ViT models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

DC	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
Swin_50	0.510±0.287	0.705±0.200	0.648±0.258	0.621±0.265
Swin_60	0.490±0.281	0.698±0.189	0.646±0.252	0.611±0.261
Swin_70	0.498±0.280	0.699±0.191	0.641±0.262	0.613±0.262
Swin_80	0.508±0.275	0.705±0.191	0.655±0.254	0.623±0.258
Swin_90	0.512±0.284	0.710±0.190	0.657±0.255	0.626±0.262
Swin_100	**0.517**±0.280	**0.714**±0.189	**0.658**±0.255	**0.630**±0.260

Table 4.3: Average Dice Coefficient values for Swin models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

U-Net

DC	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
U-Net_50	0.368±0.268	0.607±0.206	0.520±0.256	0.498±0.265
U-Net_60	0.394±0.266	0.639±0.198	0.534±0.255	0.522±0.262
U-Net_70	0.424±0.267	0.638±0.197	0.560±0.250	0.541±0.257
U-Net_80	0.459±0.267	0.670±0.194	0.591±0.255	0.574±0.257
U-Net_90	**0.472**±0.271	**0.677**±0.193	**0.596**±0.250	**0.582**±0.256
U-Net_100	**0.472**±0.272	0.669±0.201	0.593±0.254	0.578±0.259

Table 4.4: Average Dice Coefficient values for U-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

DC	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
nnU-Net_50	0.520±0.269	0.736±0.176	0.650±0.247	0.635±0.252
nnU-Net_60	0.538±0.271	0.740±0.177	0.664±0.247	0.647±0.251
nnU-Net_70	0.535±0.276	0.740±0.171	0.676±0.244	0.651±0.251
nnU-Net_80	0.548±0.273	0.747±0.171	0.684±0.245	0.660±0.249
nnU-Net_90	0.555±0.277	0.745±0.178	**0.700**±0.240	0.667±0.250
nnU-Net_100	**0.568**±0.271	**0.751**±0.173	0.693±0.243	**0.671**±0.246

Table 4.5: Average Dice Coefficient values for nnU-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

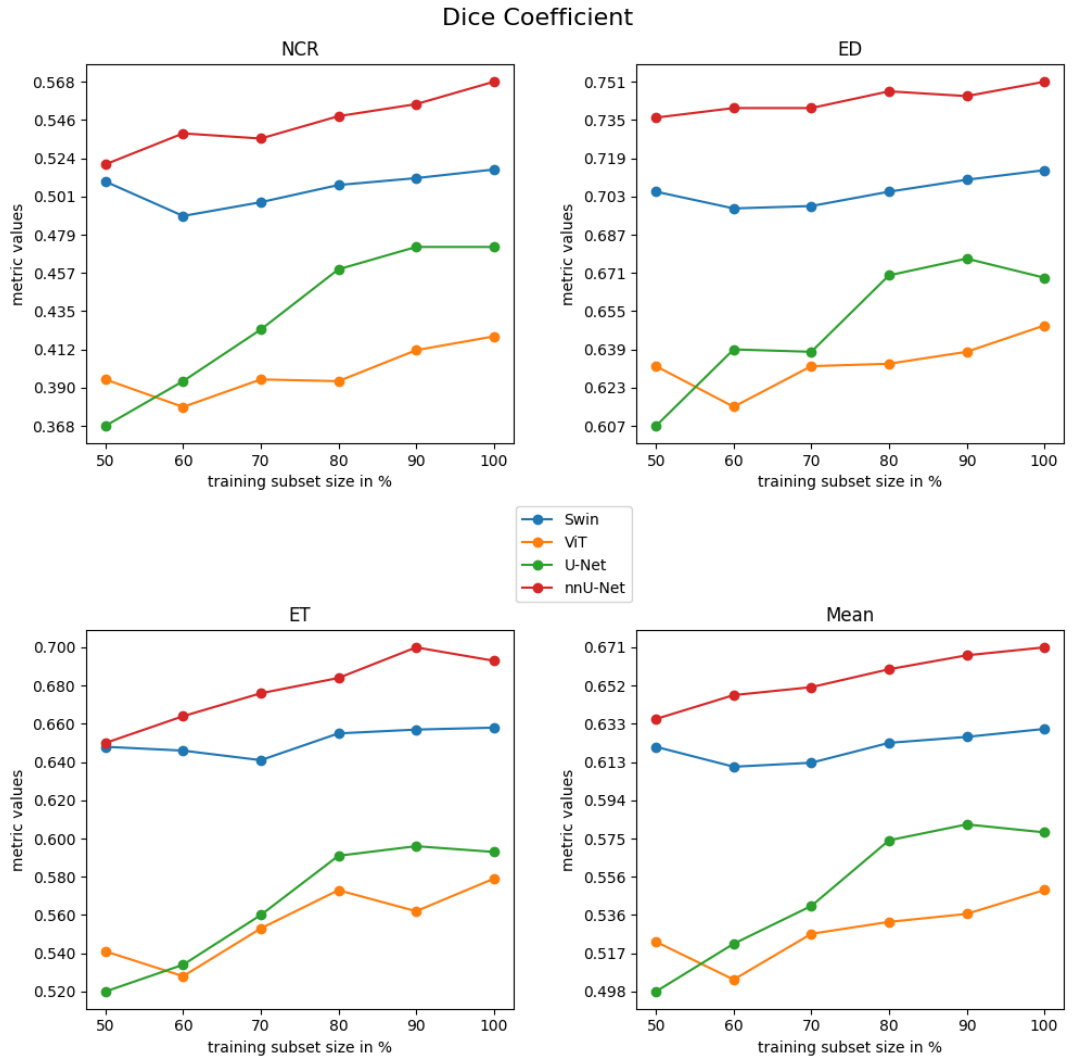

Figure 4.12: Average Dice Coefficient values on test data for Swin, ViT, U-Net, and nnU-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits.

HGG and LGG

The average Dice Coefficient is separately calculated for the subjects in the test data labeled with HGG and LGG, respectively. The average Dice Coefficient calculation for test subjects labeled with HGG and LGG is performed using the same methodology as in the previous section. The corresponding values are presented in Tables 4.6-4.9. Tables 4.10 and 4.11 are created specifically to provide an overview of all model architectures and their respective values for HGG and LGG test subjects.

ViT

	HGG				LGG			
DC	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.415±0.263	0.654±0.194	0.582±0.204	0.551±0.246	0.321±0.308	0.546±0.221	0.380±0.338	0.416±0.312
60%	0.390±0.262	0.636±0.195	0.577±0.191	0.534±0.244	0.292±0.288	0.532±0.223	0.339±0.353	0.388±0.318
70%	0.413±0.265	0.657±0.181	0.598±0.187	0.556±0.240	0.329±0.280	0.535±0.219	0.380±0.349	0.415±0.309
80%	0.415±0.265	0.660±0.177	0.611±0.187	0.562±0.239	0.317±0.284	0.529±0.214	**0.430**±0.362	0.425±0.312
90%	0.428±0.266	0.661±0.182	0.607±0.193	0.565±0.239	**0.351**±0.281	**0.548**±0.208	0.390±0.347	0.429±0.305
100%	**0.438**±0.267	**0.675**±0.173	**0.625**±0.182	**0.579**±0.236	**0.351**±0.275	0.546±0.208	0.403±0.362	**0.433**±0.303

Table 4.6: Average Dice Coefficient values on all subjects in the test set labeled with LGG and HGG respectively for ViT models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

	HGG				LGG			
DC	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.549±0.267	0.738±0.180	0.705±0.184	0.664±0.231	0.359±0.302	**0.581**±0.212	0.425±0.337	0.455±0.315
60%	0.526±0.267	0.730±0.169	0.693±0.184	0.650±0.231	0.351±0.289	0.575±0.198	**0.464**±0.363	0.464±0.310
70%	0.532±0.264	0.734±0.170	0.702±0.185	0.656±0.229	0.369±0.296	0.569±0.200	0.405±0.353	0.448±0.310
80%	0.540±0.266	0.739±0.169	0.712±0.178	0.664±0.229	0.387±0.271	0.573±0.198	0.436±0.352	0.465±0.298
90%	0.544±0.270	0.744±0.167	0.711±0.185	0.666±0.231	**0.389**±0.301	0.578±0.199	0.453±0.353	**0.473**±0.309
100%	**0.553**±0.269	**0.749**±0.165	**0.717**±0.180	**0.673**±0.228	0.379±0.277	0.579±0.198	0.431±0.343	0.463±0.301

Table 4.7: Average Dice Coefficient values on all subjects in the test set labeled with LGG and HGG respectively for Swin models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

U-Net

	HGG				LGG			
DC	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.403±0.255	0.630±0.196	0.563±0.196	0.532±0.238	0.228±0.257	0.518±0.210	0.357±0.328	0.368±0.314
60%	0.426±0.257	0.665±0.183	0.589±0.191	0.560±0.236	0.270±0.261	0.544±0.203	0.319±0.295	0.378±0.299
70%	0.467±0.250	0.671±0.179	0.612±0.183	0.583±0.225	0.260±0.244	0.512±0.196	0.358±0.321	0.377±0.290
80%	0.498±0.259	0.703±0.175	**0.656**±0.176	0.619±0.227	0.314±0.234	0.545±0.199	0.341±0.324	0.400±0.290
90%	**0.503**±0.260	**0.706**±0.176	**0.656**±0.177	**0.621**±0.226	**0.356**±0.272	**0.565**±0.200	0.363±0.334	0.428±0.297
100%	0.502±0.264	0.699±0.186	0.642±0.193	0.614±0.234	**0.356**±0.268	0.552±0.200	**0.404**±0.336	**0.437**±0.296

Table 4.8: Average Dice Coefficient values on all subjects in the test set labeled with LGG and HGG respectively for U-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

	HGG				LGG			
DC	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.553±0.254	0.769±0.151	0.710±0.155	0.678±0.215	0.392±0.285	0.608±0.199	0.420±0.363	0.474±0.309
60%	0.570±0.255	0.775±0.149	0.723±0.157	0.690±0.214	0.416±0.290	0.605±0.201	0.437±0.364	0.486±0.309
70%	0.566±0.260	0.776±0.142	0.737±0.151	0.693±0.214	0.417±0.304	0.605±0.196	0.447±0.362	0.490±0.309
80%	0.576±0.260	0.781±0.144	0.740±0.157	0.699±0.215	**0.439**±0.292	**0.618**±0.197	0.474±0.369	**0.510**±0.308
90%	0.588±0.262	0.782±0.150	0.755±0.145	0.708±0.214	0.430±0.291	0.606±0.197	**0.489**±0.368	0.508±0.306
100%	**0.605**±0.250	**0.787**±0.144	**0.756**±0.143	**0.716**±0.203	0.430±0.299	0.614±0.196	0.449±0.361	0.498±0.309

Table 4.9: Average Dice Coefficient values on all subjects in the test set labeled with LGG and HGG respectively for nnU-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

HGG

DC	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	0.549	0.738	0.705	0.664	0.415	0.654	0.582	0.551	0.403	0.630	0.563	0.532	**0.553**	**0.769**	**0.710**	**0.678**
60	0.526	0.730	0.693	0.650	0.390	0.636	0.577	0.534	0.426	0.665	0.589	0.560	**0.570**	**0.775**	**0.723**	**0.690**
70	0.532	0.734	0.702	0.656	0.413	0.657	0.598	0.556	0.467	0.671	0.612	0.583	**0.566**	**0.776**	**0.737**	**0.693**
80	0.540	0.739	0.712	0.664	0.415	0.660	0.611	0.562	0.498	0.703	0.656	0.619	**0.576**	**0.781**	**0.740**	**0.699**
90	0.544	0.744	0.711	0.666	0.428	0.661	0.607	0.565	0.503	0.706	0.656	0.621	**0.588**	**0.782**	**0.755**	**0.708**
100	0.553	0.749	0.717	0.673	0.438	0.675	0.625	0.579	0.502	0.699	0.642	0.614	**0.605**	**0.787**	**0.756**	**0.716**

Table 4.10: Average Dice Coefficient values on all subjects in the test labeled with HGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

LGG

DC	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	0.359	0.581	**0.425**	0.455	0.321	0.546	0.380	0.416	0.228	0.518	0.357	0.368	**0.392**	**0.608**	0.420	**0.474**
60	0.351	0.575	**0.464**	0.464	0.292	0.532	0.339	0.388	0.270	0.544	0.319	0.378	**0.416**	**0.605**	0.437	**0.486**
70	0.369	0.569	0.405	0.448	0.329	0.535	0.380	0.415	0.260	0.512	0.358	0.377	**0.417**	**0.605**	**0.447**	**0.490**
80	0.387	0.573	0.436	0.465	0.317	0.529	0.430	0.425	0.314	0.545	0.341	0.400	**0.439**	**0.618**	**0.474**	**0.510**
90	0.389	0.578	0.453	0.473	0.351	0.548	0.390	0.429	0.356	0.565	0.363	0.428	**0.430**	**0.606**	**0.489**	**0.508**
100	0.379	0.579	0.431	0.463	0.351	0.546	0.403	0.433	0.356	0.552	0.404	0.437	**0.430**	**0.614**	**0.449**	**0.498**

Table 4.11: Average Dice Coefficient values on all subjects in the test set labeled with LGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

Jaccard Index

The average Jaccard Index for each class and the mean Jaccard Index of all classes are computed for all 120 models using the same method used to calculate the average Dice Coefficient. In Tables 4.12-4.15, the average Jaccard Index values on test data, recorded for each model architecture, are presented. As before, we additionally provide Figure 4.13, as an overview figure, allowing a direct comparison between the four model architectures.

ViT

IOU	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT_50	0.288±0.229	0.493±0.199	0.414±0.230	0.398±0.237
ViT_60	0.263±0.217	0.474±0.197	0.399±0.229	0.379±0.233
ViT_70	0.284±0.223	0.490±0.191	0.422±0.227	0.399±0.231
ViT_80	0.284±0.224	0.491±0.189	0.441±0.230	0.405±0.234
ViT_90	0.298±0.225	0.496±0.189	0.431±0.228	0.408±0.231
ViT_100	**0.304**±0.227	**0.507**±0.188	**0.446**±0.227	**0.419**±0.232

Table 4.12: Average Jaccard Index (IOU) values on test data for ViT models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

IOU	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
Swin_50	0.393±0.259	0.578±0.209	0.526±0.243	0.499±0.251
Swin_60	0.370±0.247	0.566±0.198	0.522±0.241	0.486±0.246
Swin_70	0.378±0.250	0.568±0.201	0.519±0.246	0.488±0.247
Swin_80	0.386±0.247	0.574±0.200	0.532±0.240	0.497±0.245
Swin_90	0.393±0.255	0.581±0.200	0.536±0.243	0.503±0.248
Swin_100	**0.397**±0.253	**0.586**±0.199	**0.537**±0.242	**0.506**±0.247

Table 4.13: Average Jaccard Index (IOU) values on test data for Swin models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value in each column is highlighted in bold.

U-Net

IOU	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
U-Net_50	0.261±0.217	0.466±0.196	0.390±0.230	0.373±0.232
U-Net_60	0.282±0.221	0.499±0.194	0.403±0.226	0.395±0.232
U-Net_70	0.307±0.225	0.497±0.192	0.428±0.224	0.411±0.230
U-Net_80	0.339±0.231	0.534±0.195	0.461±0.230	0.444±0.235
U-Net_90	0.352±0.236	**0.541**±0.196	0.464±0.227	0.452±0.234
U-Net_100	**0.355**±0.240	0.536±0.203	**0.469**±0.233	**0.453**±0.239

Table 4.14: Average Jaccard Index (IOU) values on test data for U-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

IOU	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
nnU-Net_50	0.395±0.242	0.610±0.193	0.524±0.232	0.510±0.241
nnU-Net_60	0.414±0.246	0.615±0.193	0.541±0.233	0.523±0.241
nnU-Net_70	0.412±0.249	0.614±0.188	0.554±0.233	0.527±0.241
nnU-Net_80	0.424±0.249	0.622±0.189	0.563±0.233	0.537±0.241
nnU-Net_90	0.433±0.254	0.622±0.194	**0.581**±0.232	0.545±0.243
nnU-Net_100	**0.445**±0.251	**0.628**±0.191	0.572±0.232	**0.548**±0.240

Table 4.15: Average Jaccard Index (IOU) values on test data for nnU-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

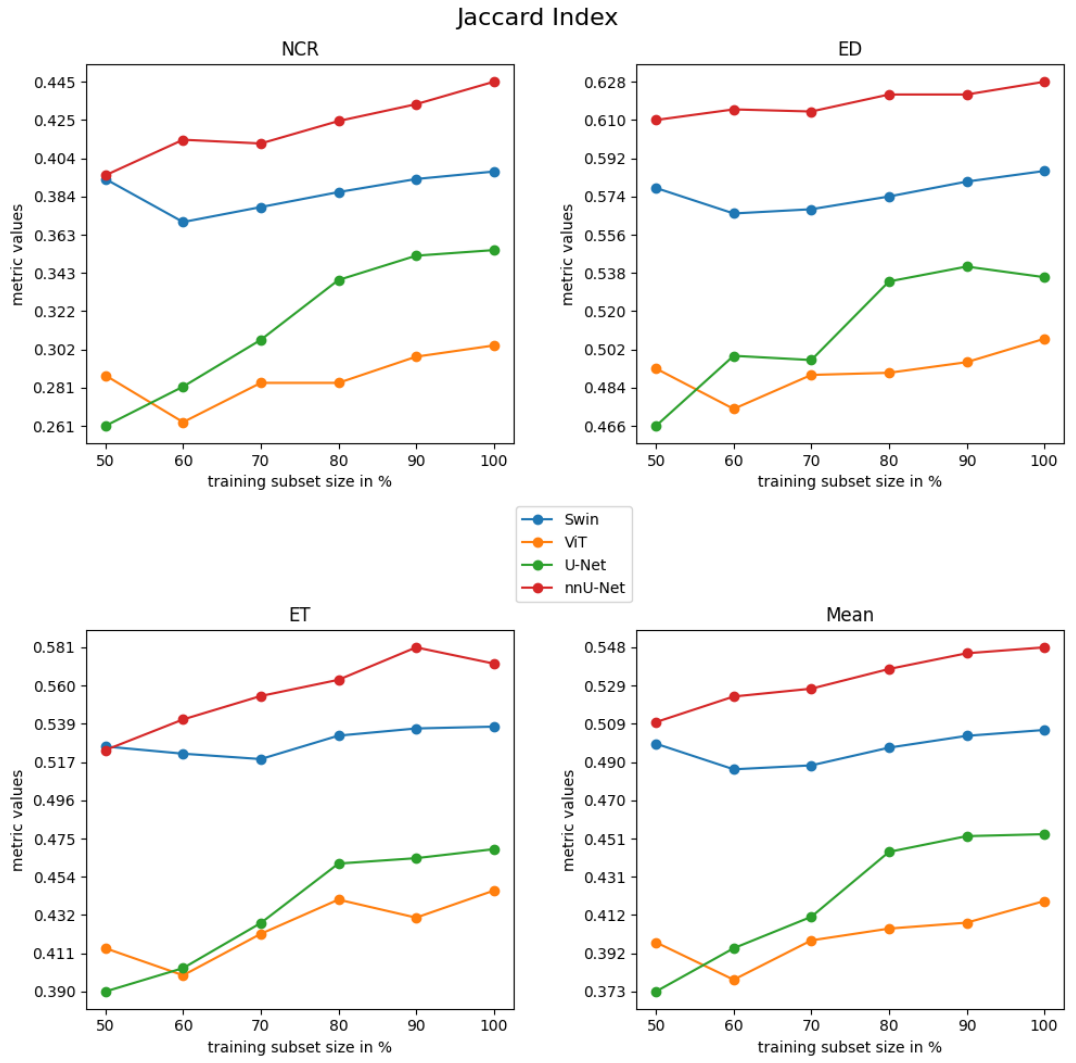

Figure 4.13: Average Jaccard Index (IOU) values on test data for Swin, ViT, U-Net, and nnU-Net models, respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits.

HGG and LGG

The average Jaccard Index is computed separately for each subject in the test data labeled with HGG and LGG, respectively. The corresponding values are presented in Tables 4.16-4.21. Tables 4.20 and 4.21 are created specifically to provide an overview of all model architectures and their respective values, for HGG and for LGG test subjects.

ViT

	HGG				LGG			
IOU	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.300±0.217	0.515±0.189	0.442±0.187	0.419±0.219	0.242±0.261	**0.409**±0.199	0.305±0.320	0.319±0.278
60%	0.277±0.212	0.494±0.186	0.429±0.174	0.400±0.213	0.210±0.229	0.395±0.197	0.279±0.336	0.295±0.279
70%	0.296±0.219	0.514±0.178	0.450±0.170	0.420±0.212	0.236±0.229	0.397±0.194	0.312±0.337	0.315±0.276
80%	0.298±0.220	0.517±0.177	0.463±0.172	0.426±0.213	0.228±0.230	0.390±0.189	**0.358**±0.353	0.325±0.282
90%	0.310±0.221	0.519±0.180	0.460±0.175	0.430±0.213	**0.253**±0.230	0.406±0.184	0.317±0.332	0.325±0.272
100%	**0.319**±0.225	**0.533**±0.175	**0.477**±0.171	**0.443**±0.213	0.251±0.225	0.405±0.189	0.329±0.343	**0.328**±0.272

Table 4.16: Average Jaccard Index (IOU) values on all subjects in the test set labeled with LGG and HGG respectively for ViT models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

	HGG				LGG			
IOU	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.425±0.246	0.613±0.192	0.571±0.185	0.536±0.226	0.268±0.255	**0.443**±0.204	0.350±0.311	0.354±0.284
60%	0.401±0.242	0.601±0.182	0.557±0.183	0.519±0.223	0.255±0.234	0.433±0.187	**0.389**±0.357	0.359±0.284
70%	0.406±0.243	0.605±0.183	0.567±0.185	0.526±0.224	0.272±0.244	0.426±0.189	0.333±0.336	0.344±0.278
80%	0.414±0.245	0.612±0.182	0.578±0.182	0.535±0.224	0.278±0.223	0.430±0.185	0.355±0.327	0.354±0.267
90%	0.420±0.249	0.618±0.181	0.578±0.186	0.539±0.226	**0.289**±0.248	0.437±0.188	0.374±0.338	**0.367**±0.280
100%	**0.429**±0.249	**0.624**±0.180	**0.585**±0.183	**0.546**±0.225	0.274±0.227	0.437±0.189	0.349±0.315	0.353±0.269

Table 4.17: Average Jaccard Index (IOU) values on all subjects in the test set labeled with LGG and HGG respectively for Swin models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

U-Net

	HGG				LGG			
IOU	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.287±0.211	0.489±0.190	0.415±0.169	0.397±0.209	0.161±0.203	0.381±0.187	0.298±0.323	0.280±0.283
60%	0.307±0.217	0.525±0.185	0.440±0.168	0.424±0.213	0.188±0.204	0.402±0.183	0.260±0.284	0.283±0.262
70%	0.341±0.217	0.531±0.180	0.464±0.165	0.445±0.206	0.180±0.194	0.370±0.172	0.291±0.309	0.280±0.256
80%	0.371±0.230	0.568±0.183	**0.510**±0.168	0.483±0.214	0.217±0.184	0.401±0.176	0.268±0.299	0.295±0.251
90%	0.377±0.232	**0.572**±0.183	**0.510**±0.167	**0.486**±0.213	**0.256**±0.220	**0.422**±0.183	0.286±0.314	0.321±0.262
100%	**0.380**±0.237	0.569±0.191	0.505±0.178	0.485±0.220	0.255±0.218	0.410±0.183	**0.330**±0.319	**0.331**±0.267

Table 4.18: Average Jaccard Index (IOU) values on all subjects in the test set labeled with LGG and HGG respectively for U-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

	HGG				LGG			
IOU	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.423±0.232	0.647±0.171	0.571±0.164	0.547±0.214	0.287±0.245	0.468±0.195	0.344±0.333	0.366±0.279
60%	0.442±0.237	0.654±0.179	0.588±0.164	0.561±0.214	0.310±0.250	0.463±0.192	0.360±0.338	0.378±0.279
70%	0.438±0.238	0.654±0.165	0.603±0.162	0.565±0.214	0.315±0.265	0.462±0.188	0.367±0.338	0.381±0.281
80%	0.449±0.240	0.660±0.167	0.608±0.164	0.572±0.215	**0.330**±0.257	**0.477**±0.191	0.395±0.348	**0.400**±0.283
90%	0.462±0.245	0.663±0.171	**0.626**±0.160	0.584±0.216	0.323±0.252	0.463±0.188	**0.408**±0.347	0.398±0.281
100%	**0.477**±0.239	**0.669**±0.167	**0.626**±0.157	**0.590**±0.209	0.324±0.258	0.472±0.189	0.367±0.332	0.388±0.278

Table 4.19: Average Jaccard Index (IOU) values on all subjects in the test set labeled with LGG and HGG respectively for nnU-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

HGG

IOU	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	**0.425**	0.613	**0.571**	0.536	0.300	0.515	0.442	0.419	0.287	0.489	0.415	0.397	0.423	**0.647**	**0.571**	**0.547**
60	0.401	0.601	0.557	0.519	0.277	0.494	0.429	0.400	0.307	0.525	0.440	0.424	**0.442**	**0.654**	**0.588**	**0.561**
70	0.406	0.605	0.567	0.526	0.296	0.514	0.450	0.420	0.341	0.531	0.464	0.445	**0.438**	**0.654**	**0.603**	**0.565**
80	0.414	0.612	0.578	0.535	0.298	0.517	0.463	0.426	0.371	0.568	0.510	0.483	**0.449**	**0.660**	**0.608**	**0.572**
90	0.420	0.618	0.578	0.539	0.310	0.519	0.460	0.430	0.377	0.572	0.510	0.486	**0.462**	**0.663**	**0.626**	**0.584**
100	0.429	0.624	0.585	0.546	0.319	0.533	0.477	0.443	0.380	0.569	0.505	0.485	**0.477**	**0.669**	**0.626**	**0.590**

Table 4.20: Average Jaccard Index (IOU) values on all subjects in the test set labeled with HGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

LGG

IOU	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	0.268	0.443	**0.350**	0.354	0.242	0.409	0.305	0.319	0.161	0.381	0.298	0.280	**0.287**	**0.468**	0.344	**0.366**
60	0.255	0.433	**0.389**	0.359	0.210	0.395	0.279	0.295	0.188	0.402	0.260	0.283	**0.310**	**0.463**	0.360	**0.378**
70	0.272	0.426	0.333	0.344	0.236	0.397	0.312	0.315	0.180	0.370	0.291	0.280	**0.315**	**0.462**	**0.367**	**0.381**
80	0.278	0.430	0.355	0.354	0.228	0.390	0.358	0.325	0.217	0.401	0.268	0.295	**0.330**	**0.477**	**0.395**	**0.400**
90	0.289	0.437	0.374	0.367	0.253	0.406	0.317	0.325	0.256	0.422	0.286	0.321	**0.323**	**0.463**	**0.408**	**0.398**
100	0.274	0.437	0.349	0.353	0.251	0.405	0.329	0.328	0.255	0.410	0.330	0.331	**0.324**	**0.472**	**0.367**	**0.388**

Table 4.21: Average Jaccard Index (IOU) values on all subjects in the test set labeled with LGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

Precision and Recall

The average precision and recall for each class and the mean precision and recall of all classes are computed for all 120 models using the same method used to calculate the average Dice Coefficient values. In Tables 4.22-4.29, the average precision and recall values on test data, recorded for each model architecture, are presented. As before, we additionally provide Figures 4.14 and 4.15, as overview figures, allowing a direct comparison between the four model architectures on each of the evaluation metrics.

ViT

Precision	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT_50	**0.610**±0.291	0.666±0.210	0.667±0.233	**0.648**±0.128
ViT_60	0.595±0.298	0.647±0.221	**0.678**±0.231	0.640±0.119
ViT_70	0.580±0.296	0.656±0.210	0.653±0.242	0.630±0.128
ViT_80	0.589±0.295	0.660±0.203	0.670±0.225	0.639±0.125
ViT_90	0.564±0.285	0.660±0.201	0.661±0.233	0.629±129
ViT_100	0.578±0.287	**0.672**±0.198	0.655±0.236	0.635±0.126

Table 4.22: Average precision values on test data for ViT models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

Precision	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
Swin_50	**0.662**±0.264	**0.737**±0.205	**0.773**±0.213	**0.724**±0.128
Swin_60	0.626±0.283	0.723±0.211	0.769±0.215	0.706±0.132
Swin_70	0.623±0.279	0.733±0.205	0.757±0.233	0.704±0.134
Swin_80	0.630±0.279	0.730±0.206	0.764±0.223	0.708±0.132
Swin_90	0.626±0.280	0.734±0.208	0.768±0.223	0.709±0.131
Swin_100	0.645±0.276	**0.737**±0.204	0.752±0.230	0.711±0.130

Table 4.23: Average precision values on test data for Swin models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

U-Net

Precision	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
U-Net_50	0.541±0.299	0.631±0.225	0.631±0.248	0.601±0.138
U-Net_60	0.563±0.296	0.660±0.222	0.651±0.245	0.625±0.132
U-Net_70	0.532±0.289	0.667±0.219	0.642±0.248	0.613±0.138
U-Net_80	0.572±0.282	0.694±0.211	0.668±0.253	0.645±0.143
U-Net_90	0.565±0.285	0.695±0.214	0.652±0.251	0.637±0.146
U-Net_100	**0.611**±0.288	**0.702**±0.213	**0.706**±0.234	**0.673**±0.141

Table 4.24: Average precision values on test data for U-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

Precision	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
nnU-Net_50	0.631±0.263	0.747±0.184	0.720±0.231	0.700±0.123
nnU-Net_60	0.646±0.268	0.751±0.183	0.734±0.226	0.710±0.118
nnU-Net_70	0.642±0.267	0.760±0.173	0.731±0.222	0.711±0.120
nnU-Net_80	**0.657**±0.262	0.757±0.180	0.741±0.223	0.718±0.121
nnU-Net_90	**0.657**±0.263	0.749±0.186	**0.757**±0.217	0.721±0.120
nnU-Net_100	0.656±0.263	**0.770**±0.184	0.747±0.230	**0.724**±0.128

Table 4.25: Average precision values on test data for nnU-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

ViT

Recall	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT_50	0.384±0.288	0.642±0.220	0.567±0.270	0.531±0.183
ViT_60	0.361±0.285	0.629±0.219	0.539±0.276	0.509±0.180
ViT_70	0.387±0.284	0.650±0.210	0.580±0.267	0.539±0.172
ViT_80	0.376±0.279	0.651±0.205	0.602±0.264	0.543±0.173
ViT_90	0.407±0.284	0.659±0.203	0.593±0.263	0.553±0.174
ViT_100	**0.421**±0.290	**0.663**±0.196	**0.619**±0.257	**0.567**±0.165

Table 4.26: Average recall values on test data for ViT models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

Recall	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
Swin_50	0.509±0.298	0.711±0.207	0.664±0.263	0.629±0.192
Swin_60	0.493±0.294	0.707±0.190	0.645±0.264	0.615±0.185
Swin_70	0.514±0.296	0.702±0.195	0.667±0.263	0.628±0.181
Swin_80	0.523±0.290	0.713±0.193	0.682±0.253	0.640±0.179
Swin_90	**0.532**±0.295	0.717±0.189	0.677±0.259	0.642±0.183
Swin_100	0.521±0.292	**0.725**±0.185	**0.690**±0.252	**0.645**±0.175

Table 4.27: Average recall values on test data for Swin models, respectively trained on 50%, 60%, 70%, 80%, 90% and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

U-Net

Recall	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
U-Net_50	0.412±0.301	0.645±0.219	0.556±0.281	0.538±0.190
U-Net_60	0.422±0.290	0.671±0.200	0.578±0.278	0.557±0.178
U-Net_70	0.469±0.286	0.658±0.192	0.612±0.265	0.580±0.177
U-Net_80	0.494±0.286	0.693±0.189	0.652±0.257	0.613±0.178
U-Net_90	**0.511**±0.284	**0.699**±0.184	**0.656**±0.255	**0.622**±0.173
U-Net_100	0.488±0.285	0.688±0.202	0.623±0.262	0.599±0.178

Table 4.28: Average recall values on test data for U-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

Recall	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
nnU-Net_50	0.529±0.285	0.759±0.174	0.691±0.239	0.660±0.158
nnU-Net_60	0.552±0.278	0.761±0.176	0.703±0.238	0.672±0.160
nnU-Net_70	0.548±0.275	0.757±0.176	0.732±0.231	0.679±0.159
nnU-Net_80	0.559±0.281	0.767±0.169	0.723±0.232	0.683±0.158
nnU-Net_90	0.569±0.280	**0.773**±0.178	0.739±0.227	**0.694**±0.158
nnU-Net_100	**0.575**±0.271	0.762±0.172	**0.740**±0.223	0.693±0.151

Table 4.29: Average recall values on test data for nnU-Net models respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

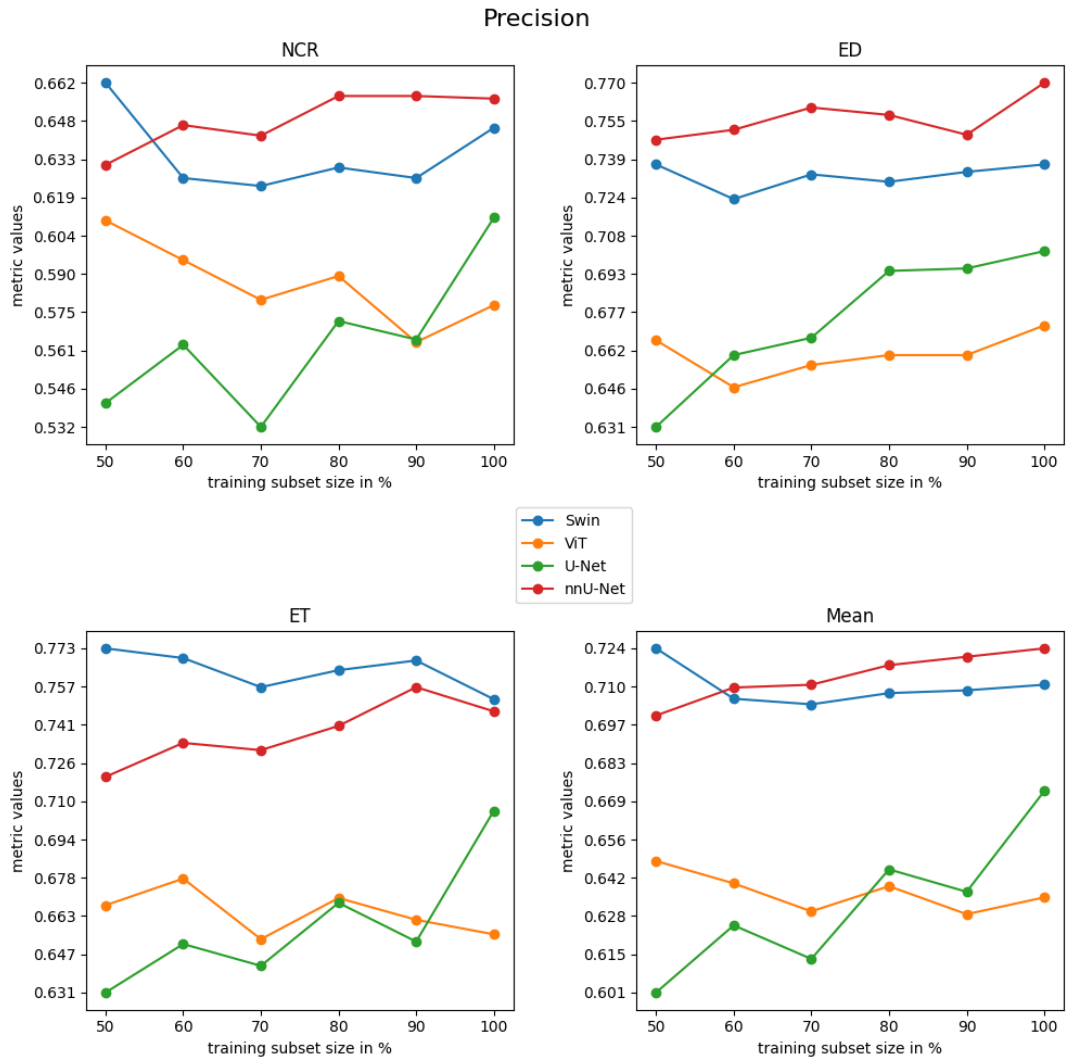

Figure 4.14: Average precision values on test data for Swin, ViT, U-Net, and nnU-Net models, respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits.

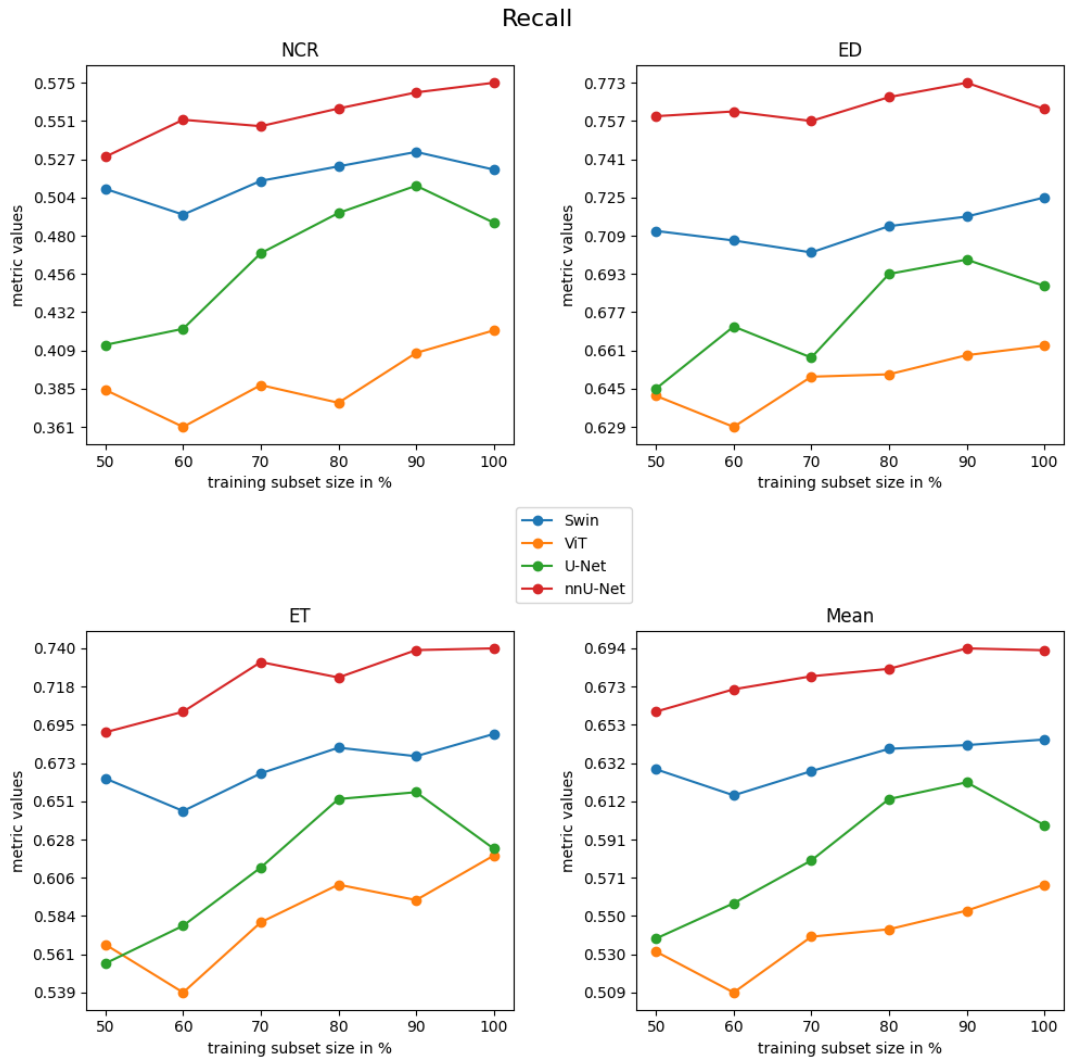

Figure 4.15: Average recall values on test data for Swin, ViT, U-Net, and nnU-Net models, respectively trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits.

HGG and LGG

The average precision and recall are computed separately for each subject in the test data labeled with HGG and LGG, respectively. The corresponding values are presented in Tables 4.30-4.37. Tables 4.38-4.41 are created specifically to provide an overview of all model architectures and their respective values for HGG and LGG test subjects.

ViT

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	**0.591**$\pm$0.279	0.702$\pm$0.181	0.687$\pm$0.168	**0.660**$\pm$0.119	**0.683**$\pm$0.312	**0.526**$\pm$0.231	0.588$\pm$0.365	0.599$\pm$0.145
60%	0.573$\pm$0.286	0.685$\pm$0.196	**0.692**$\pm$0.163	0.650$\pm$0.107	0.679$\pm$0.306	0.501$\pm$0.233	0.619$\pm$0.385	0.600$\pm$0.142
70%	0.559$\pm$0.284	0.693$\pm$0.183	0.666$\pm$0.184	0.639$\pm$0.122	0.662$\pm$0.312	0.512$\pm$0.228	0.603$\pm$0.383	0.592$\pm$0.140
80%	0.572$\pm$0.287	0.695$\pm$0.175	0.674$\pm$0.176	0.647$\pm$0.116	0.650$\pm$0.297	0.523$\pm$0.228	**0.650**$\pm$0.346	**0.608**$\pm$0.145
90%	0.549$\pm$0.273	0.697$\pm$0.174	0.674$\pm$0.178	0.640$\pm$0.121	0.620$\pm$0.293	0.520$\pm$0.223	0.612$\pm$0.368	0.584$\pm$0.141
100%	0.557$\pm$0.273	**0.712**$\pm$0.167	0.670$\pm$0.180	0.646$\pm$0.118	0.657$\pm$0.307	0.515$\pm$0.218	0.602$\pm$0.374	0.591$\pm$0.143

Table 4.30: Average precision values on all subjects in the test set labeled with LGG and HGG respectively for ViT models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	**0.642**$\pm$0.256	0.780$\pm$0.169	**0.792**$\pm$0.133	**0.738**$\pm$0.113	**0.737**$\pm$0.264	**0.571**$\pm$0.232	0.670$\pm$0.376	**0.669**$\pm$0.157
60%	0.609$\pm$0.271	0.766$\pm$0.179	0.787$\pm$0.147	0.721$\pm$0.123	0.688$\pm$0.300	0.559$\pm$0.229	**0.704**$\pm$0.355	0.651$\pm$0.142
70%	0.610$\pm$0.267	0.777$\pm$0.169	0.780$\pm$0.163	0.722$\pm$0.124	0.674$\pm$0.302	0.565$\pm$0.227	0.670$\pm$0.381	0.636$\pm$0.136
80%	0.616$\pm$0.270	0.773$\pm$0.171	0.784$\pm$0.155	0.724$\pm$0.122	0.684$\pm$0.293	0.564$\pm$0.227	0.688$\pm$0.365	0.645$\pm$0.138
90%	0.624$\pm$0.266	0.778$\pm$0.172	0.787$\pm$0.153	0.730$\pm$0.122	0.638$\pm$0.318	0.568$\pm$0.233	0.694$\pm$0.367	0.633$\pm$0.127
100%	0.641$\pm$0.264	**0.782**$\pm$0.164	0.780$\pm$0.152	0.734$\pm$0.119	0.661$\pm$0.302	0.567$\pm$0.233	0.641$\pm$0.382	0.623$\pm$0.126

Table 4.31: Average precision values on all subjects in the test set labeled with LGG and HGG respectively for Swin models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

U-Net

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.501$\pm$0.272	0.669$\pm$0.207	0.647$\pm$0.191	0.606$\pm$0.122	**0.697**$\pm$0.331	0.483$\pm$0.218	0.569$\pm$0.363	0.583$\pm$0.178
60%	0.538$\pm$0.277	0.700$\pm$0.197	0.675$\pm$0.177	0.638$\pm$0.127	0.661$\pm$0.339	0.507$\pm$0.227	0.558$\pm$0.388	0.575$\pm$0.139
70%	0.509$\pm$0.266	0.710$\pm$0.191	0.666$\pm$0.190	0.628$\pm$0.126	0.621$\pm$0.329	0.501$\pm$0.217	0.546$\pm$0.362	0.556$\pm$0.155
80%	0.556$\pm$0.269	0.734$\pm$0.187	0.701$\pm$0.181	0.664$\pm$0.129	0.638$\pm$0.305	**0.541**$\pm$0.219	0.538$\pm$0.387	0.572$\pm$0.159
90%	0.551$\pm$0.269	0.737$\pm$0.186	0.694$\pm$0.184	0.661$\pm$0.137	0.618$\pm$0.319	0.533$\pm$0.225	0.490$\pm$0.355	0.547$\pm$0.134
100%	**0.605**$\pm$0.273	**0.744**$\pm$0.185	**0.735**$\pm$0.166	**0.695**$\pm$0.133	0.635$\pm$0.318	0.539$\pm$0.220	**0.598**$\pm$0.371	**0.591**$\pm$0.135

Table 4.32: Average precision values on all subjects in the test set labeled with LGG and HGG respectively for U-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.615±0.249	0.786±0.145	0.751±0.146	0.717±0.109	**0.693**±0.290	0.599±0.224	0.603±0.394	0.632±0.141
60%	0.644±0.252	0.792±0.140	0.761±0.142	0.732±0.100	0.655±0.300	0.594±0.228	0.628±0.391	0.625±0.136
70%	0.642±0.253	0.797±0.129	0.752±0.149	0.730±0.103	0.643±0.288	**0.616**±0.221	0.655±0.378	0.638±0.139
80%	0.652±0.249	0.796±0.141	0.766±0.148	0.738±0.107	0.677±0.278	0.610±0.221	0.645±0.378	**0.644**±0.134
90%	**0.660**±0.249	0.790±0.142	0.783±0.139	0.744±0.107	0.650±0.289	0.592±0.235	**0.657**±0.372	0.633±0.126
100%	0.659±0.247	**0.814**±0.134	**0.779**±0.144	**0.751**±0.111	0.642±0.296	0.600±0.232	0.623±0.391	0.622±0.132

Table 4.33: Average precision values on all subjects in the test set labeled with LGG and HGG respectively for nnU-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

ViT

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.411±0.276	0.649±0.214	0.575±0.234	0.545±0.178	0.280±0.301	0.614±0.231	0.536±0.362	0.477±0.181
60%	0.389±0.282	0.630±0.210	0.555±0.226	0.525±0.173	0.253±0.266	0.621±0.233	0.476±0.400	0.450±0.185
70%	0.414±0.284	0.656±0.200	0.600±0.218	0.557±0.162	0.282±0.257	0.626±0.231	0.504±0.386	0.471±0.187
80%	0.401±0.276	0.659±0.193	0.615±0.220	0.558±0.166	0.282±0.265	0.617±0.220	0.553±0.374	0.484±0.180
90%	0.431±0.281	0.663±0.195	0.607±0.222	0.567±0.171	0.319±0.274	0.639±0.212	0.537±0.372	0.498±0.173
100%	**0.447**±0.289	**0.668**±0.187	**0.631**±0.210	**0.582**±0.159	**0.323**±0.266	**0.646**±0.212	**0.557**±0.375	**0.508**±0.169

Table 4.34: Average recall values on all subjects in the test set labeled with LGG and HGG respectively for ViT models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Swin

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.561±0.276	0.730±0.200	0.693±0.217	0.662±0.183	0.307±0.281	0.639±0.205	0.555±0.370	0.500±0.171
60%	0.547±0.280	0.722±0.187	0.674±0.215	0.648±0.178	0.290±0.255	0.653±0.171	0.534±0.378	0.492±0.158
70%	0.562±0.278	0.719±0.190	0.698±0.210	0.660±0.168	0.333±0.288	0.635±0.186	0.550±0.383	0.506±0.174
80%	0.571±0.275	0.731±0.188	0.707±0.205	0.670±0.171	0.341±0.266	0.644±0.179	0.586±0.364	0.524±0.159
90%	**0.578**±0.279	0.735±0.182	0.704±0.212	0.673±0.175	**0.356**±0.290	0.649±0.182	0.574±0.373	**0.526**±0.162
100%	0.571±0.278	**0.743**±0.179	**0.716**±0.205	**0.677**±0.167	0.330±0.260	**0.655**±0.173	**0.587**±0.359	0.524±0.145

Table 4.35: Average recall values on all subjects in the test set labeled with LGG and HGG respectively for Swin models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects pertained to the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

U-Net

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.465±0.288	0.646±0.214	0.573±0.233	0.561±0.183	0.209±0.248	0.644±0.220	0.490±0.405	0.448±0.177
60%	0.469±0.284	0.672±0.196	0.600±0.233	0.581±0.174	0.242±0.236	0.669±0.186	0.493±0.393	0.468±0.157
70%	0.522±0.265	0.670±0.190	0.639±0.212	0.610±0.164	0.266±0.233	0.613±0.181	0.509±0.385	0.463±0.159
80%	0.545±0.270	0.704±0.183	0.676±0.208	0.642±0.165	0.299±0.248	0.649±0.185	0.558±0.376	0.502±0.178
90%	**0.552**±0.268	**0.707**±0.183	**0.677**±0.210	**0.645**±0.164	**0.354**±0.274	**0.670**±0.166	**0.575**±0.369	**0.533**±0.168
100%	0.528±0.272	0.697±0.203	0.640±0.218	0.622±0.175	0.330±0.268	0.650±0.178	0.557±0.372	0.512±0.158

Table 4.36: Average recall values on all subjects in the test set labeled with LGG and HGG respectively for U-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

nnU-Net

	HGG				LGG			
	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50%	0.578±0.265	0.778±0.163	0.714±0.181	0.690±0.146	0.340±0.282	0.686±0.181	0.600±0.365	0.542±0.143
60%	0.596±0.260	0.783±0.165	0.731±0.178	0.703±0.146	0.388±0.281	0.678±0.176	0.593±0.361	0.553±0.153
70%	0.588±0.254	0.779±0.162	0.764±0.164	0.711±0.143	0.399±0.293	0.672±0.187	0.606±0.361	0.559±0.155
80%	0.602±0.261	0.787±0.158	0.750±0.171	0.713±0.145	0.394±0.294	**0.688**±0.174	0.616±0.357	0.566±0.147
90%	0.608±0.262	**0.797**±0.164	0.767±0.164	**0.724**±0.145	**0.420**±0.289	0.681±0.182	**0.629**±0.351	**0.577**±0.148
100%	**0.620**±0.247	0.783±0.163	**0.770**±0.158	**0.724**±0.134	0.408±0.281	0.685±0.172	**0.629**±0.355	0.574±0.144

Table 4.37: Average recall values on all subjects in the test set labeled with LGG and HGG respectively for nnU-Net models trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

HGG

	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	**0.642**	0.780	**0.792**	**0.738**	0.591	0.702	0.687	0.660	0.501	0.669	0.647	0.606	0.615	**0.786**	0.751	0.717
60	0.609	0.766	**0.787**	0.721	0.573	0.685	0.692	0.650	0.538	0.700	0.675	0.638	**0.644**	**0.792**	0.761	**0.732**
70	0.610	0.777	**0.780**	0.722	0.559	0.693	0.666	0.639	0.509	0.710	0.666	0.628	**0.642**	**0.797**	0.752	**0.730**
80	0.616	0.773	**0.784**	0.724	0.572	0.695	0.674	0.647	0.556	0.734	0.701	0.664	**0.652**	**0.796**	0.766	**0.738**
90	0.624	0.778	**0.787**	0.730	0.549	0.697	0.674	0.640	0.551	0.737	0.694	0.661	**0.660**	**0.790**	0.783	**0.744**
100	0.641	0.782	**0.780**	0.734	0.557	0.712	0.670	0.646	0.605	0.744	0.735	0.695	**0.659**	**0.814**	0.779	**0.751**

Table 4.38: Average precision values on all subjects in test test labeled with HGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

LGG

	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	**0.737**	0.571	**0.670**	**0.669**	0.683	0.526	0.588	0.599	0.697	0.483	0.569	0.583	0.693	**0.599**	0.603	0.632
60	**0.688**	0.559	**0.704**	**0.651**	0.679	0.501	0.619	0.600	0.661	0.507	0.558	0.575	0.655	**0.594**	0.628	0.625
70	**0.674**	0.565	**0.670**	0.636	0.662	0.512	0.603	0.592	0.621	0.501	0.546	0.556	0.643	**0.616**	0.655	**0.638**
80	**0.684**	0.564	**0.688**	**0.645**	0.650	0.523	0.650	0.608	0.638	0.541	0.538	0.572	0.677	**0.610**	0.645	0.644
90	0.638	0.568	**0.694**	**0.633**	0.620	0.520	0.612	0.584	0.618	0.533	0.490	0.547	**0.650**	**0.592**	0.657	**0.633**
100	**0.661**	0.567	**0.641**	**0.623**	0.657	0.515	0.602	0.591	0.635	0.539	0.598	0.591	0.642	**0.600**	0.623	0.622

Table 4.39: Average precision values on all subjects in test test labeled with LGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

HGG																
	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	0.561	0.730	0.693	0.662	0.411	0.649	0.575	0.545	0.465	0.646	0.573	0.561	**0.578**	**0.778**	**0.714**	**0.690**
60	0.547	0.722	0.674	0.648	0.389	0.630	0.555	0.525	0.469	0.672	0.600	0.581	**0.596**	**0.783**	**0.731**	**0.703**
70	0.562	0.719	0.698	0.660	0.414	0.656	0.600	0.557	0.522	0.670	0.639	0.610	**0.588**	**0.779**	**0.764**	**0.711**
80	0.571	0.731	0.707	0.670	0.401	0.659	0.615	0.558	0.545	0.704	0.676	0.642	**0.602**	**0.787**	**0.750**	**0.713**
90	0.578	0.735	0.704	0.673	0.431	0.663	0.607	0.567	0.552	0.707	0.677	0.645	**0.608**	**0.797**	**0.767**	**0.724**
100	0.571	0.743	0.716	0.677	0.447	0.668	0.631	0.582	0.528	0.697	0.640	0.622	**0.620**	**0.783**	**0.770**	**0.724**

Table 4.40: Average recall values on all subjects in test test labeled with HGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

LGG																
	Swin				ViT				U-Net				nnU-Net			
%	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean	NCR	ED	ET	Mean
50	0.307	0.639	0.555	0.500	0.280	0.614	0.536	0.477	0.209	0.644	0.490	0.448	**0.340**	**0.686**	**0.600**	**0.542**
60	0.290	0.653	0.534	0.492	0.253	0.621	0.476	0.450	0.242	0.669	0.493	0.468	**0.388**	**0.678**	**0.593**	**0.553**
70	0.333	0.635	0.550	0.506	0.282	0.626	0.504	0.471	0.266	0.613	0.509	0.463	**0.399**	**0.672**	**0.606**	**0.559**
80	0.341	0.644	0.586	0.524	0.282	0.617	0.553	0.484	0.299	0.649	0.558	0.502	**0.394**	**0.688**	**0.616**	**0.566**
90	0.356	0.649	0.574	0.526	0.319	0.639	0.537	0.498	0.354	0.670	0.575	0.533	**0.420**	**0.681**	**0.629**	**0.577**
100	0.330	0.655	0.587	0.524	0.323	0.646	0.557	0.508	0.330	0.650	0.557	0.512	**0.408**	**0.685**	**0.629**	**0.574**

Table 4.41: Average recall values on all subjects in test test labeled with LGG for Swin, ViT, U-Net, and nnU-Net models, trained on 50%, 60%, 70%, 80%, 90%, and 100% of subjects in the training data. The values are rounded to three significant digits, and the maximum achieved value for the categories NCR, ED, ET, and Mean in each row is highlighted in bold.

4.2 Experiment 2

In Experiment 2, a total of 20 models, five per model architecture, are trained using the training procedure described in section 3.8. The epoch with the lowest validation loss for each model is recorded, and the model parameters from that epoch are used to generate segmentation masks on the corresponding test data. These masks are then compared to the ground truth segmentation masks to compute evaluation metrics including Dice Coefficient, Jaccard Index, precision, and recall for each model. The subsequent sections showcase the outcomes from Experiment 2.

Predicted Segmentation Masks

In this section, a comparison of predicted segmentation masks and ground truth segmentation masks is showcased. The comparison is similar to the one displayed in Figure 4.11. In this comparison, the same training data and test data are used. However, for this comparison, data augmentation is employed during the training of the models. The models produced this way are used to generate segmentation masks associated with the same subjects as in Figure 4.11. How the addition of employing data augmentation during training affects the prediction segmentation masks is shown in Figure 4.16.

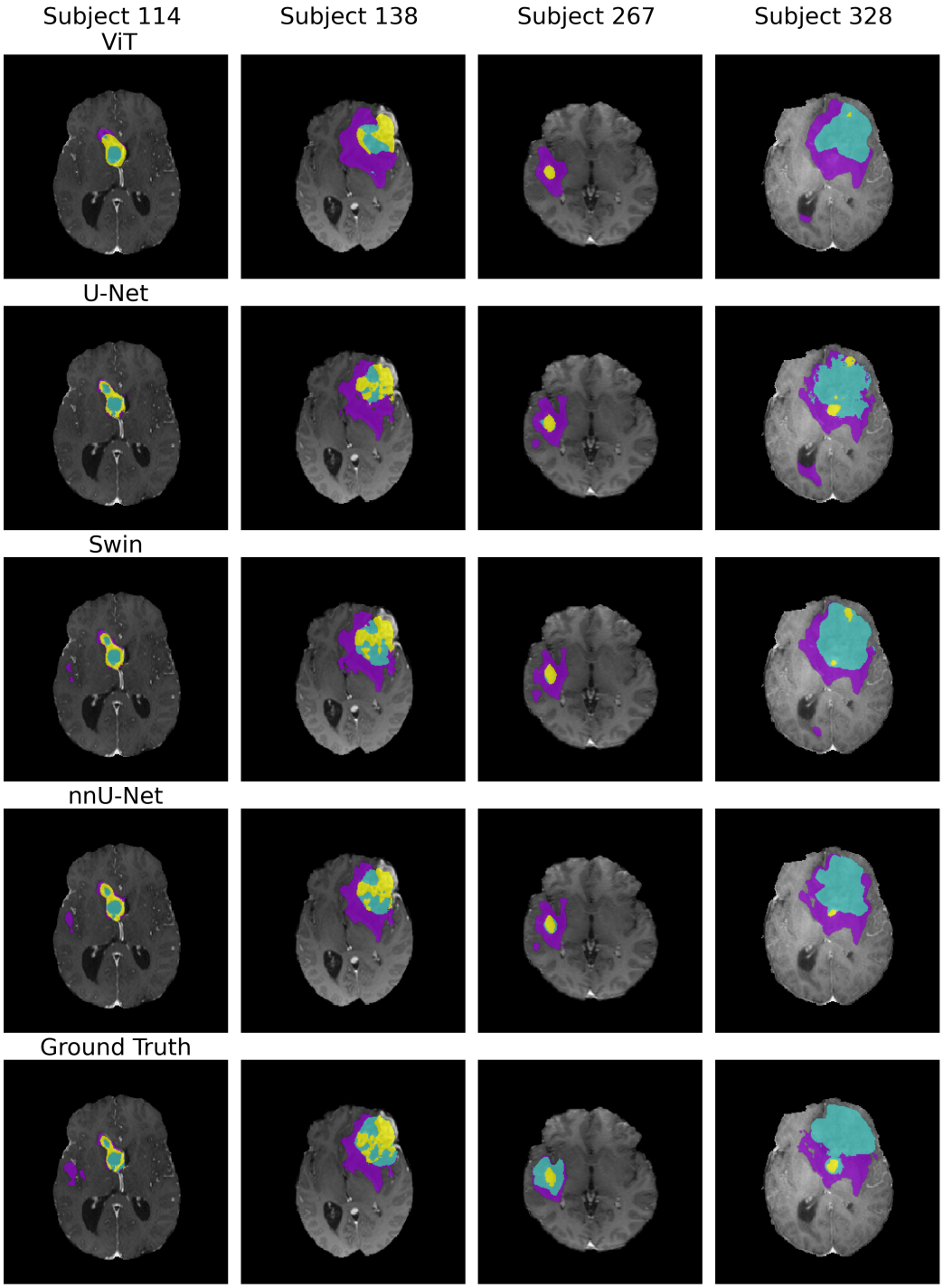

Figure 4.16: Experiment 2: A comparison of segmentation masks and ground truth segmentations of four subjects from the BRATS20 dataset. All models are trained on 100% of the subjects in the training data associated with run 001. Additionally, data augmentation is employed during the training of the models. The four subjects are part of the test data, and the segmentation masks correspond to axial slice 70 of each subject. Subjects 114 and 138 are labeled with HGG, whereas Subject 267 and Subject 328 are annotated as LGG. Peritumoral edema (ED) is highlighted in violet, the non-enhancing tumor core (NCR/NET) in turquoise, and the GD-enhancing tumor (ET) in yellow.

Dice Coefficient

For each of the 20 trained models, five per model architecture, the average Dice Coefficient for each class as well as the mean Dice Coefficient of all classes are calculated. This is performed on the corresponding test data of each of the models. The five models for each model architecture are trained on the training data and evaluated on the test data contained in 2dslices_100_001, 2dslices_100_002, 2dslices_100_003, 2dslices_100_004, 2dslices_100_005. After having calculated the average Dice Coefficient for each class as well as the mean Dice Coefficient of all classes for each of the 20 models, another computation is performed. The average over all the Dice Coefficient values of the models of each model architecture is calculated. The resulting values of this procedure are presented in Table 4.42 together with the average standard deviation.

DC	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT	0.447±0.269	0.663±0.180	0.599±0.243	0.570±0.252
U-Net	0.483±0.270	0.683±0.181	0.613±0.246	0.593±0.251
nnU-Net	**0.570**±0.265	**0.754**±0.168	**0.694**±0.247	**0.673**±0.243
Swin	0.533±0.270	0.722±0.177	0.671±0.249	0.642±0.250

Table 4.42: Average Dice Coefficient values on test data for Swin, ViT, U-Net, and nnU-Net models, respectively trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

HGG and LGG

The average Dice Coefficient is separately calculated for the subjects in the test data labeled with HGG and LGG, respectively. The average Dice Coefficient calculation for test subjects labeled with HGG and LGG is performed using the same method as in the previous section. The corresponding values are presented in Table 4.43.

	HGG				LGG			
DC	NCR	ED	ET	Mean	NCR	ED	ET	Mean
ViT	0.463±0.262	0.690±0.160	0.639±0.173	0.597±0.226	0.387±0.285	0.561±0.203	0.441±0.363	0.463±0.307
Swin	0.561±0.259	0.756±0.153	0.726±0.168	0.681±0.218	0.426±0.279	0.593±0.193	**0.457**±0.350	0.492±0.301
U-Net	0.518±0.258	0.714±0.157	0.661±0.181	0.631±0.221	0.351±0.265	0.565±0.200	0.425±0.344	0.447±0.298
nnU-Net	**0.600**±0.253	**0.790**±0.140	**0.758**±0.146	**0.716**±0.206	**0.457**±0.279	**0.617**±0.189	0.446±0.367	**0.507**±0.301

Table 4.43: Average Dice Coefficient values on all subjects in the test data labeled with LGG and HGG, respectively, for Swin, ViT, U-Net, and nnU-Net models, trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Jaccard Index

The average Jaccard Index for each class and the mean Jaccard Index of all classes are computed for all 20 models using the same method used to calculate the average Dice Coefficient. In Table 4.44, the average Jaccard Index values on the test data, recorded for each model architecture, are presented.

IOU	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT	0.327±0.228	0.521±0.180	0.466±0.227	0.438±0.229
U-Net	0.362±0.239	0.546±0.188	0.483±0.227	0.464±0.233
nnU-Net	**0.445**±0.248	**0.631**±0.187	**0.575**±0.234	**0.550**±0.238
Swin	0.409±0.247	0.592±0.192	0.549±0.238	0.517±0.241

Table 4.44: Average Jaccard Index (IOU) values on test data for Swin, ViT, U-Net, and nnU-Net models, respectively trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

HGG and LGG

The average Jaccard Index is computed separately for each subject in the test data labeled with HGG and LGG, respectively. The computation adheres to the identical methodology employed to calculate the Dice Coefficient values for test subjects labeled with HGG and LGG, respectively. The corresponding values are presented in Table 4.45 together with the average standard deviation.

	HGG				LGG			
IOU	NCR	ED	ET	Mean	NCR	ED	ET	Mean
ViT	0.339±0.225	0.547±0.167	0.491±0.166	0.459±0.208	0.282±0.235	0.419±0.183	0.368±0.352	0.356±0.278
Swin	0.433±0.242	0.630±0.173	0.593±0.176	0.552±0.218	0.316±0.240	0.449±0.185	**0.377**±0.324	0.381±0.273
U-Net	0.391±0.234	0.578±0.171	0.519±0.171	0.496±0.210	0.250±0.212	0.422±.182	0.345±0.332	0.339±0.268
nnU-Net	**0.472**±0.240	**0.673**±0.164	**0.630**±0.159	**0.591**±0.211	**0.342**±0.246	**0.473**±0.180	0.365±0.331	**0.393**±0.269

Table 4.45: Average Jaccard Index (IOU) values on all subjects in the test data labeled with LGG and HGG, respectively, for Swin, ViT, U-Net, and nnU-Net models, trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Precision and Recall

The average precision and recall for each class and the mean precision and recall of all classes are computed for all 20 models using the same method used to calculate the average Dice Coefficient values. In Tables 4.46-4.47, the average precision and recall values on test data, recorded for each model architecture, are presented together with the average standard deviation.

Precision	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT	0.568±0.281	0.679±0.194	0.680±0.217	0.643±0.118
U-Net	0.574±0.288	0.704±0.205	0.691±0.241	0.657±0.133
nnU-Net	**0.642**±0.267	**0.769**±0.176	0.746±0.230	**0.719**±0.131
Swin	0.634±0.274	0.741±0.194	**0.764**±0.219	0.713±0.126

Table 4.46: Average precision values on test data for Swin, ViT, U-Net, and nnU-Net models, respectively trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

Recall	NCR (Class 1)	ED (Class 2)	ET (Class 3)	Mean
ViT	0.452±0.280	0.680±0.180	0.634±0.251	0.589±0.153
U-Net	0.523±0.281	0.703±0.173	0.647±0.250	0.624±0.165
nnU-Net	**0.606**±0.257	**0.773**±0.164	**0.746**±0.218	**0.709**±0.148
Swin	0.554±0.278	0.733±0.173	0.698±0.240	0.662±0.164

Table 4.47: Average recall values on test data for Swin, ViT, U-Net, and nnU-Net models, respectively trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

HGG and LGG

The average precision and recall are computed separately for each subject in the test data labeled with HGG and LGG, respectively. The computation follows the same methodology as applied for calculating the Dice Coefficient values for test subjects annotated with HGG and LGG, respectively. The corresponding values are presented in Tables 4.48-4.49.

	HGG				LGG			
Precision	NCR	ED	ET	Mean	NCR	ED	ET	Mean
ViT	0.550±0.269	0.717±0.161	0.688±0.160	0.651±0.111	0.639±0.296	0.534±0.226	0.649±0.348	0.607±0.135
Swin	0.619±0.264	0.783±0.155	**0.789**±0.144	0.730±0.115	**0.692**±0.294	0.582±0.230	**0.667**±0.364	**0.647**±0.137
U-Net	0.558±0.275	0.749±0.165	0.723±0.178	0.677±0.124	0.637±0.310	0.530±0.232	0.568±0.368	0.579±0.133
nnU-Net	**0.642**±0.251	**0.809**±0.133	0.779±0.147	**0.743**±0.112	0.645±0.294	**0.614**±0.219	0.617±0.383	0.625±0.141

Table 4.48: Average precision values on all subjects in the test data labeled with LGG and HGG, respectively, for Swin, ViT, U-Net, and nnU-Net models, trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

	HGG				LGG			
Recall	NCR	ED	ET	Mean	NCR	ED	ET	Mean
ViT	0.479±0.275	0.688±0.173	0.650±0.199	0.606±0.146	0.352±0.272	0.646±0.192	0.574±0.374	0.524±0.154
Swin	0.597±0.260	0.753±0.169	0.721±0.192	0.691±0.156	0.389±0.277	0.656±0.155	0.609±0.355	0.551±0.144
U-Net	0.572±0.264	0.711±0.169	0.661±0.212	0.648±0.158	0.336±0.252	0.673±0.165	0.593±0.351	0.534±0.149
nnU-Net	**0.645**±0.239	**0.793**±0.156	**0.773**±0.156	**0.737**±0.136	**0.459**±0.259	**0.696**±0.158	**0.644**±0.347	**0.600**±0.141

Table 4.49: Average recall values on all subjects in the test data labeled with LGG and HGG, respectively, for Swin, ViT, U-Net, and nnU-Net models, trained on 100% of subjects in the training data. Furthermore, the models are trained with data augmentation. The values are rounded to three significant digits, and the maximum value of each column is highlighted in bold.

5 Discussion

In this chapter, we examine the results presented in chapter 4 and discuss their implications. The methodology utilized to obtain the results, along with the experimental design of each of the experiments, is critically analyzed. Furthermore, suggestions for future work are presented.

5.1 Experiment 1

Training Time

Figure 4.1 shows that the total training time for each model architecture grows linearly with increasing amounts of training data. The total training time curve for the Swin Transformer is similar to the total training time curve of the Vision Transformer. With each increment in training data, the Swin Transformer exhibits a slightly longer training time compared to the Vision Transformer. The 2D U-Net based model architectures are training faster than the Transformer based model architectures, with the 2D nnU-Net exhibiting the shortest total training time for each percentage of training data.

When trained on 50% of the subjects in the training data, the Transformer architectures each take about 3 hours to train, the 2D U-Net takes approximately 1 hour and 30 minutes to train and the 2D nnU-Net takes less than 30 minutes to train. For every increase in training data, the difference in total training time between the model architectures grows larger, except between the Swin Transformer and the Vision Transformer. The 2D nnU-Net trained on 100% of the subjects in the training data takes less than 1 hour to train. In comparison, both the Swin Transformer and the Vision Transformer trained on 100% of the subjects in the training data, take about 5 hours to finish training. The 2D U-Net is done with the training process after approximately 2 hours and 20 minutes.

The largest difference in total training time between the 2D nnU-Net and the Transformer based architectures is approximately 4 hours and 30 minutes. The 2D U-Net trained with the nnU-Net framework finishes training substantially faster than the other three model architectures. This is not surprising since the Transformer based architectures are larger architectures with more model parameters. The reason for the 2D nnU-Net being faster than the 2D U-Net probably stems from differences in implementation. For instance, how the training data is loaded into memory is different and can affect the time it takes to train.

GPU Memory Usage

Surprisingly, we can observe in Figure 4.2 that the 2D U-Net consumes the highest amount of GPU memory during training compared to the other model architectures. The Transformer model architectures would be expected to occupy the most GPU memory during training, due to being larger in size. The high GPU memory consumption of the 2D U-Net is probably due to its implementation approach. The Swin Transformer uses the second most GPU memory during training, 10820 MiB, only 492 MiB less than the 2D U-Net. The Vision Transformer uses 404 MiB less GPU memory than the Swin Transformer. The 2D U-Net trained with the nnU-Net framework uses 3114 MiB, which is a lot less GPU memory compared to all other model architectures.

Validation Loss

The validation and training loss reported in Figures 4.3-4.10 are a curated selection, representing how the loss curves of each model approximately look like. In Figure 4.5, the validation loss recorded for Swin_50_001 spikes between epochs 3 and 4. This is because the total loss of the Swin Transformer and the Vision Transformer consists of more than one component since the architectures are implemented with UPerNet. During an epoch, it is possible that when the loss of one component decreases, the loss of another component increases to a greater extent, leading to a spike in validation loss. This is also true for the training loss of the Swin Transformer and the Vision Transformer. Furthermore, the computation of the loss differs between the 2D nnU-Net and the 2D U-Net. Because the loss values for each model architecture are computed differently a direct comparison between the loss values of each model architecture is not reasonable. Only the loss values between the Vision Transformer models and the Swin Transformer models are compared. In Figure 4.3-4.6, the validation loss for the Vision Transformer is higher in both cases, implying that the Swin Transformer models are performing better in terms of their ability to generalize to unseen data.

Segmentation Time

The total and the per slice segmentation time reported for all model architectures in Table 4.1 shows that the Vision Transformer exhibits the shortest segmentation time out of all model architectures. The second shortest total and per slice segmentation time is obtained by the Swin Transformer, followed by the 2D nnU-Net and the 2D U-Net. One would expect the segmentation time for the 2D nnU-Net and the 2D U-Net to be shorter than the segmentation time for the Transformer architectures since the 2D U-Net based architectures are smaller. This observation possibly relates to the implementation approaches of each of the model architectures. The way the predicted segmentation masks are generated and saved to the computer is different for the 2D nnU-Net compared to the rest of the models. Different implementation approaches between the 2D nnU-Net and the other model architectures result in a less-than-ideal comparison of segmentation time. Furthermore, the implementation approaches for the Transformer based architectures are different compared to 2D U-Net. Due to these implementation differences, the results in Table 4.1 might not be indicative of how the model architectures relate to one another in terms of segmentation time.

Predicted Segmentation Masks

Looking at Figure 4.11, we can observe that the predicted segmentation masks generated for each subject by the 2D nnUNet, are the most similar to the corresponding ground truth segmentation masks. For subjects 114 and 138 labeled with HGG, the predicted segmentation masks related to the 2D nnU-Net are the most like the ground truth, closely followed by the predicted segmentation masks of the Swin Transformer. The predicted segmentation masks for subjects 114 and 138 generated by the Vision Transformer look the least similar to

the respective ground truth segmentation masks. Furthermore, the 2D U-Net and its corresponding predicted segmentation masks are not as similar to the ground truth segmentation masks as the predicted segmentation masks of the Swin Transformer and the 2D nnU-Net. Overall, the predicted segmentation masks of each of the four model architectures on both subjects look similar to the ground truth segmentation masks, except the predicted segmentation mask of the Vision Transformer on subject 138. Regarding the predicted segmentation masks for subjects 267 and 328 labeled with LGG, none of the predicted segmentation masks exhibit a strong resemblance to the corresponding ground truth segmentation mask. However, the predicted segmentation masks generated by the 2D nnU-Net are the most similar to the ground truth segmentation masks, followed by the Swin Transformer, 2D U-Net, and Vision Transformer. Because all models are trained on significantly more data related to HGG subjects than LGG subjects, the models generated more accurate segmentation masks on HGG subjects.

Dice Coefficient and Jaccard Index

Regarding the average Dice Coefficient for each class, NCR, ED, and ET, as well as the average of the mean Dice Coefficient of all classes, the 2D nnU-Net outperforms the other model architectures. For each percentage of training subjects and category, NCR, ED, ET, and Mean, shown in Figure 4.12, the 2D nnU-Net reaches the highest Dice Coefficient value. In Figure 4.12, we can observe that the most difficult class to predict, for each model architecture, is the NCR class, followed by ED and ET. When trained on 100% of the subjects in the training data, the 2D nnU-Net achieves the average Dice Coefficient values of 0.568, 0.751, 0.693, and 0.671 on NCR, ED, ET, and Mean, respectively. Furthermore, when trained on 50% of the training data, 0.520, 0.736, 0.650, and 0.635 is achieved by the 2D nnU-Net for NCR, ED, ET, and Mean. The Vision Transformer and the 2D U-net trained on 100% of the subjects in the training data, do not come close to any of the values achieved by the 2D nnU-Net trained on 50% of the subjects in the training data. This means that the 2D nnU-Net only needs half of the training subjects given to the Vision Transformer and the 2D U-Net to outperform them in every category. Additionally, the average Dice Coefficient value for each category achieved by the Swin Transformer trained on 50% of the subjects in the training data is higher than the values for the Vision Transformer and the 2D U-net trained on 100% of the subjects in the training data. Therefore, not only the 2D nnU-Net, but also the Swin Transformer, outperforms the Vision Transformer and 2D U-Net even when given half of the training subjects.

When trained on 100% of the subjects in the training data, the Swin Transformer reaches the average Dice Coefficient values of 0.517, 0.714, 0.658, and 0.630 for the categories NCR, ED, ET, and Mean, as shown in Figure 4.12. Compared to the values achieved by the 2D nnU-Net trained on 50% of the subjects in training data, only the corresponding average Dice Coefficient value for ET is higher. This is also true for the Swin Transformer when respectively trained on 90% and 80% of the subjects in training data. Otherwise, the 2D nnU-Net trained on 50% of subjects in training data outperforms the Swin Transformer for every increase of training data. The 2D nnU-Net outperforming the other model architectures can possibly be explained by the pre-processing step of the nnU-Net framework. The extraction of dataset fingerprints most likely helps the 2D nnU-Net to better capture the features in the training data. Furthermore, the cropping of images employed by the nnU-Net framework removes a lot of background pixels, making each image focused on the pixels related to the brain. This way, the 2D nnU-Net might become better at predicting for the classes NCR, ED, and ET.

The Vision Transformer is the model architecture that, in terms of the Dice Coefficient, performs the worst. Except for the average Dice Coefficient values for the Vision Transformer trained on 50% of the subjects in the training data, the 2D U-Net outperforms the Vision Transformer for each percentage of training subjects and category, NCR, ED, ET, and Mean. The exception is probably a direct effect of randomly initialized model parameters resulting

in the U-Net models trained on 50% of the subjects in the training data, performing worse than the Vision Transformer models trained on the same data.

For every increase in training data, each model architecture reaches higher average Dice Coefficient values with only a few exceptions, as shown in Figure 4.12. This is partly because the initial model parameters of each model are randomly initialized. The initial model parameters can affect each model's ability to be optimized, and with the model parameters being randomly initialized, sometimes the resulting models are worse or better. Interestingly, going from 50% to 100% of subjects in the training data does not result in a significant increase for each Dice Coefficient value for either the Swin Transformer or the Vision Transformer in comparison to the 2D U-Net based architectures. For the Swin Transformer, the increase for class NCR, ED, and ET is 0.7%, 0.9%, and 1.0%, respectively. Likewise, for the Vision Transformer, the increase is 2.5%, 1.7%, and 3.8%. For the 2D U-Net, the increase for class NCR, ED, and ET is 10.4%, 6.2%, and 7.3% respectively. For the 2D nnU-Net, the increase is 4.8%, 1.5%, and 4.3% respectively. The low increase can be attributed to the Transformer based architectures' inherent need to be trained on large datasets to learn. The increase from 50% to 100% training subjects is not sufficient to improve the Transformer based architectures noticeably.

The 2D nnU-Net trained on 50% of subjects in the training data, achieves average Dice Coefficient values for the categories NCR, ED, ET and Mean, that are 1.0%, 3.1%, 0.2% and 1.4% higher than the values of the Swin Transformer trained on the same percentage of training data. For the Vision Transformer, the values of the 2D nnU-Net are 12.5%, 10.4%, 10.9% and 11.2% higher, respectively. Furthermore, for the 2D U-Net, the values of the 2D nnU-Net are 15.2%, 12.7%, 13.0% and 13.7% higher, respectively. When the training data is increased from 50% to 100% of the training subjects, the differences between the 2D nnU-Net and the Transformer based model architectures are larger. The 2D nnU-Net trained on 100% of subjects in the training data, achieves average Dice Coefficient values for the categories NCR, ED, ET and Mean, that are 5.1%, 3.7%, 3.5% and 4.1% higher than the values of the Swin Transformer trained on the same percentage of training data. For the Vision Transformer, the values of the 2D nnU-Net are 14.8%, 10.2%, 11.4% and 12.2% higher, respectively. Furthermore, for the 2D U-Net, the values of the 2D nnU-Net are 9.6%, 8.2%, 10.0% and 9.3% higher, respectively. For every increment of training data, the Transformer based architectures do not increase the Dice Coefficient values to the extent of the 2d U-Net architectures. This is possibly due to the Swin Transformer and the Vision Transformer needing more training data. In order to reach the same increase in Dice Coefficient as the 2D U-Net and the 2D nnU-Net the Transformer based architectures would possibly need more training data. The 2D nnU-Net performs the best according to the Dice Coefficient, followed by the Swin Transformer, 2D U-Net and the Vision Transformer.

The Jaccard Index values reported in Figure 4.13 exhibit a very similar pattern to that of the Dice Coefficient. Figure 4.13 shows that the 2D nnU-Net outperforms the other model architectures. In Figure 4.12, we can observe that the 2D nnU-Net consistently achieves the highest Jaccard Index values for each category, NCR, ED, ET, and Mean, with only one exception. The exception is that the average Jaccard Index for ET of the Swin Transformer models trained on 50% of subjects in the training data is higher than the corresponding value for 2D nnU-Net models trained on 50% of subjects in the training data.

The 2D nnU-Net performs the best according to the Jaccard Index, followed by the Swin Transformer, the 2D U-Net, and the Vision Transformer. The 2D nnU-Net convincingly outperforms the Swin Transformer, the Vision Transformer, and the 2D U-Net regarding both the Dice Coefficient and Jaccard Index. This strongly indicates that the 2D nnU-Net generalizes the best to unseen data and that predicted segmentation masks generated by this architecture would show the most resemblance to the ground truth segmentation masks.

Dice Coefficient and Jaccard Index on HGG and LGG Subjects

When only evaluated on the HGG subjects of the test data, the Dice Coefficient values of each model architecture in Table 4.10 are higher for each percentage of training subjects and category NCR, ED, ET, and Mean, presented in Figure 4.12. On the opposite end, the Dice Coefficient values on LGG subjects in the test data are lower as shown in Table 4.11. This is also the case for the Jaccard Index values computed for HGG and LGG subjects, respectively, as seen in Tables 4.20-4.20. The difference in values for HGG subjects and LGG subjects is attributed to the training data, used to train the models, all have a similar distribution of LGG and HGG subjects as in the training data of the BraTS 2020 dataset. This distribution is 20.6% LGG subjects and 79.4% HGG subjects. All models are thus trained on significantly more data of HGG subjects than LGG subjects, which allows the models to generalize better on unseen subjects labeled with HGG than unseen subjects labeled with LGG.

As shown in Table 4.10, the 2D nnU-Net trained on 50% of the subjects in training data and evaluated on only HGG subjects, the average Dice Coefficient values obtained for the categories NCR, ED, ET, and Mean are 0.553, 0.769, 0.710, and 0.678, respectively. When evaluated on all subjects in test data, the result for each respective category is 0.520, 0.736, 0.650, and 0.635. By only evaluating on HGG subjects, the increase of each value is 3.3%, 3.3%, 6.0%, and 4.3%, respectively. Furthermore, the 2D nnU-Net trained on 100% of the subjects in training data and evaluated on the HGG subjects the average Dice Coefficient values achieved for NCR, ED, ET, and Mean are 0.605, 0.787, 0.756, 0.716. Compared to the average Dice coefficient values obtained by evaluating on all test subjects, the increase of each category is 3.7%, 3.6%, 6.3%, and 4.5%. Unsurprisingly, due to the data distribution, the values achieved by each model architecture on HGG subjects are higher than the corresponding values on all test data. Furthermore, this implies that the model architectures perform worse on LGG subjects than on all test subjects.

In Figure 4.12, when evaluated on all test subjects, we observed that the 2D nnU-Net trained on 50% of the training subjects achieved better results than the Vision Transformer and the 2D U-Net trained on 100% of the training subjects. This is also observed for the values related to the HGG subjects. Furthermore, regarding the HGG test subjects, the Swin Transformer trained on 100% of the training subjects is also outperformed by the 2D nnU-Net trained on 50% of the training subjects, with a few exceptions. These exceptions are the values obtained for ET by the Swin Transformer trained on 100%, 90%, and 80%, which are higher than the corresponding value achieved by the 2D nnU-Net trained on 50% of the training subjects. For each model architecture evaluated on HGG test subjects, in terms of Dice Coefficient, 2D nnU-Net performs the best, followed by the Swin Transformer, the 2D U-Net, and the Vision Transformer.

Regarding the Dice Coefficient values for the model architectures evaluated on only LGG subjects, the 2D nnU-Net is the best-performing architecture. However, in two cases the Swin Transformer yields higher values. The Swin Transformer trained on 50% and 60% of subjects in training data, respectively, achieves average Dice Coefficient values of 0.425 and 0.464 for class ET. In comparison, the 2D nnU-Net trained on the same amount of training data yields the values, 0.420 and 0.437, respectively. The values of the Swin Transformer are therefore 0.05% and 2.7% higher in each case. The reason for the nnU-Net not outperforming the Swin Transformer in the aforementioned cases is because of the small amount of LGG data used in training, which results in higher variability in the predictions. When the model architectures are evaluated on LGG subjects and HGG subjects, respectively, a difference in the average standard deviation is observed. The average standard deviation for each model architecture evaluated on LGG subjects is higher than the corresponding average standard deviation values for HGG subjects, as shown in Tables 4.6-4.9 and Tables 4.16-4.19 for the Dice Coefficient and Jaccard Index, respectively. A higher average standard deviation indicates higher uncertainty in the predictions. When trained on significantly more data related to HGG subjects than LGG subjects, the predictions on unseen LGG subjects exhibit a higher

variability than HGG predictions. This further signifies that when evaluated on only LGG subjects, the models are not certain in their predictions.

The average Dice Coefficient values obtained for the categories NCR, ED, ET, and Mean by the 2D nnU-Net trained on 50% of the subjects in training data and evaluated on LGG test subjects are 0.392, 0.608, 0.420, and 0.474. Compared to the corresponding values for HGG subjects, in Table 4.10, 0.553, 0.769, 0.710, and 0.678, the aforementioned values are 16.1%, 16.1%, 29%, and 20.4% lower. Furthermore, the average Dice Coefficient values on LGG subjects recorded for the categories NCR, ED, ET, and Mean by the 2D nnU-Net trained on 100% of the subjects in training data are 0.430, 0.614, 0.449, and 0.498, respectively. These values compared to the corresponding values for HGG subjects in Table 4.10 are lower by 17.5%, 17.3%, 30.7%, and 21.8%, respectively. For the model architectures, the differences are about the same. This signifies a big discrepancy between the performance on HGG subjects and LGG subjects. For each model architecture evaluated on LGG test subjects, in terms of Dice Coefficient, nnU-Net convincingly performs the best in about all categories, followed by the Swin Transformer, the 2D U-Net, and the Vision Transformer.

Regarding the Jaccard Index values achieved by each model architecture on HGG and LGG subjects, respectively, a similar trend is observed in Tables 4.20-4.21. The difference, however, is that for HGG subjects, the Swin Transformer trained on 50% of the training subjects outperforms the 2D nnU-Net trained on the same training data percentage on the categories NCR and ET. Furthermore, for the same categories, the nnU-Net trained on 50% of the training subjects does not outperform the Swin Transformer trained on 100% of the training subjects. This possibly indicates that when evaluated on HGG subjects, which constitutes a large majority of the training data, the Swin Transformer trained on 100% of the training subjects can reach comparable values to the 2D nnU-Net trained on 50% of the subjects in training data. However, the 2D nnU-Net trained on half as much data still reaches a better Jaccard Index value for the Mean category than the Swin Transformer trained on 100% of the training subjects.

Precision and Recall

In semantic segmentation tasks, each pixel of an image is classified as one of the available classes. If a model classifies ten pixels to be of class ED, and five of these ten pixels are classified correctly, then the precision is 50%. Furthermore, if the total number of pixels classified as ED is 20 in the ground truth, then the recall is 25%. A higher precision indicates that the model has a greater ability to avoid misclassifying pixels, while a higher recall indicates that a larger proportion of ground truth pixels belonging to a specific class is correctly predicted by the model. The two metrics are in a trade-off relationship. If a model is very cautious and only predicts obvious true positives, the number of false positives would be reduced which corresponds to an increase in precision. However, with a cautious approach, the number of false negatives is potentially increased, which would decrease the recall. If a model is more inclusive in its prediction, the number of false negatives is decreased, which increases the recall. This potentially increases the false positives, which would in turn lead to decreased precision.

Figure 4.14 shows that the 2D nnU-Net in terms of precision outperforms all other model architectures on the categories NCR, ED, and Mean for almost all percentages of training subjects. Specifically, when trained on 50% of the training subjects, the Swin Transformer records a higher average precision value on NCR and Mean. The Swin Transformer reaches higher average precision on ET compared to the other model architectures including the 2D nnU-Net. Thus, the 2D nnU-Net is not convincingly better than the Swin Transformer based on the average precision values. In terms of recall, the 2D nnU-Net outperforms all other model architectures on all categories NCR, ED, ET, and Mean, respectively, for each increment in training data as shown in Figure 4.15.

In the plots shown in Figures 4.14-4.15, we can observe that the recall values are lower than the precision values for each model architecture and category. This indicates that all model architectures are more cautious in classifying the pixels to a specific class, resulting in more false negatives as shown by the lower recall values. The largest differences between precision and recall are observed for the NCR class, for each model architecture. This highlights that the model architectures have more difficulties in classifying the NCR class.

In Figure 4.14, the plots highlight that the Swin Transformer and the 2D nnU-Net architectures have higher average precision values than the Vision Transformer and the 2D U-Net in every category for each increment of training data. For the categories, NCR and ET, the Vision Transformer outperforms the 2D U-Net when trained 50%, 60%, 70%, and 80%, respectively. Additionally, the Vision Transformer reaches an average precision value that is higher than the corresponding value achieved by the 2D U-Net, when both architectures are trained on 90% training subjects. For the ED class, the Vision Transformer only outperforms the 2D U-Net when trained on 50% of training subjects. Furthermore, the average mean precision value is higher for the Vision Transformer than the 2D U-Net when trained on 50%, 60%, and 70% of training subjects, and for larger amounts of training data, the 2D U-Net is better. In terms of precision, the Vision Transformer is, therefore, better than the 2D U-net when trained on 50%, 60%, and 70% of training subjects, while the 2D U-Net is better when 80%, 90%, and 100% of the training subjects are used for training.

In Figure 4.14, we can observe that the Transformer based model architectures in general, for each increment of training data, do not consistently increase precision. Specifically, the average Mean precision value for the Vision Transformer is decreasing with more training subjects used for training. In contrast, Figure 4.15 shows that the average Mean recall values for the Vision Transformer and the Swin Transformer in general increase, although not by a lot, for each increment of the training data. In comparison, in Figures 4.14 and 4.15, the respective slopes of the Mean precision curve and the Mean recall curve of the 2D U-Net based architectures are overall positive. The precision and the recall simultaneously increase for the 2D U-Net based models with increasing training data. The difference between this observation and the one made for the Transformer based architectures, possibly indicates that the training data increase from 50% to 100% has a small effect on increasing recall and precision of the Transformer based architectures. Furthermore, the Dice Coefficient and Jaccard Index of the Transformer based architecture are not increased as much as for the 2D U-Net based architectures, further indicating that the Transformers need more training data to reach large improvements.

Precision and Recall on HGG and LGG Subjects

When evaluated on HGG subjects, instead of all the test subjects, the Mean precision and Mean recall of each model architecture is higher. However, the average precision for NCR, about 0.650, is higher on LGG subjects than HGG subjects for the Swin Transformer, Vision Transformer, and 2D U-Net. The corresponding average recall values are much lower, they are approximately 0.330 for each model architecture. Meanwhile, the average recall values for ED and ET are approximately 0.650 and 0.560. Furthermore, the average precision values for ED and ET are not too different from the aforementioned average recall values. This indicates that the Swin Transformer, Vision Transformer, and 2D U-Net classify ground truth NCR pixels as ED, ET, or background, which in turn results in a poor recall for NCR pixels. This further implies that the NCR class is more difficult to classify correctly than the other classes.

In terms of recall, the 2D nnU-Net outperforms all the other model architectures for each category and training percentage, on HGG and LGG subjects, respectively. However, based on precision, the Swin Transformer almost performs as well as the 2D nnU-Net. When evaluated on HGG subjects the Swin Transformer reaches higher average precision values on ET, for each increment in training data. Furthermore, on LGG subjects and for each incre-

ment in training data, the Swin Transformer achieves higher average precision values on ET, while also reaching higher average precision values on NCR and Mean. The higher precision achieved by the Swin Transformer might be misinterpreted as the Swin Transformer performing better than the 2D nnU-Net for classifying ET pixels. However, it is important to note that high precision alone does not correspond to well-predicted segmentation masks, a high recall is required as well. Only when a model architecture is performing the best in terms of both precision and recall, can we deduce that this model architecture generalizes the best on unseen data. None of our four model architectures outperforms any of the other architectures in both precision and recall simultaneously.

5.2 Experiment 2

Predicted Segmentation Masks

When trained on the training data contained in 2dslices_100_001 and without employing data augmentation, the 2D nnU-Net generated the predicted segmentation masks with the highest resemblance to the ground truth segmentation masks as shown in Figure 4.11. For each model architecture trained on the training data of 2dslices_100_001 while applying data augmentation, the 2D nnU-Net still generates the segmentation masks with the highest resemblance to every corresponding ground truth segmentation mask. Compared to the predicted segmentation masks generated by the models not trained using data augmentation, the predicted segmentation masks in Figure 4.16 do not look very different. However, some are noticeably more similar to the corresponding ground truth segmentation masks than the predicted segmentation masks generated by the models not trained with data augmentation. For instance, for subject 114, the predicted segmentation masks generated by the Vision Transformer and the Swin Transformer of Experiment 2, respectively, are both slightly more similar to the ground truth segmentation mask than the corresponding predicted segmentation masks in Experiment 1. Similarly, the 2D nnU-Net generated predicted segmentation mask in Experiment 2 related to subject 328 shows a stronger resemblance to the ground truth segmentation mask than the corresponding predicted segmentation mask in Experiment 1.

The predicted segmentation masks related to LGG subjects 267 and 328, did not noticeably improve. Moreover, some show even less of a resemblance to the ground truth segmentation masks than the previous segmentation masks in Figure 4.11. The predicted segmentation masks for subject 328 generated by the Vision Transformer, the Swin Transformer, and the 2D U-Net look less similar to the ground truth segmentation masks. This is because, in each of the predicted segmentation masks, an area of ET is predicted, for which there is no area of ET in the corresponding ground truth. The addition of data augmentation does not indicate an overall noticeable improvement of the predicted segmentation masks for any of the model architectures compared to the predicted segmentation masks in Experiment 1.

Dice Coefficient and Jaccard Index

For each of the model architectures trained on 100% of the subjects in training data and with employing data augmentation, the 2D nnU-Net achieves the highest average Dice Coefficient values on test data, for each category NCR, ED, ET, and Mean, respectively. For the average Jaccard Index values, the same outcome is observed.

In regard to the average Dice Coefficient values, 2D nnU-Net trained with data augmentation and on 100% of the training subjects reaches values of 0.570, 0.754, 0.694, and 0.673, respectively. Compared to the 2D nnU-Net trained on 100% of the subjects in the training data and without employing data augmentation, the values are slightly increased with 0.2%, 0.3%, 0.1%, and 0.2%, respectively. In the case of the Swin Transformer, employing data augmentation compared to not applying data augmentation, yields a larger improvement of 1.6%, 0.8%, 1.3%, and 1.2% for each category NCR, ED, ET, and Mean. Additionally, when comparing the

Vision Transformer and the 2D U-Net trained with data augmentation to the same model architectures trained with data augmentation, there is an improvement in the values with an increase of 2.7%, 1.4%, 2.0%, 2.1%, and 1.1%, 1.4%, 2.0%, 1.5% respectively. On the other hand, as shown in Table 4.4, the average Dice Coefficient values obtained from training the 2D U-Net on 90% of the subjects in the training data are higher than those achieved by training it on 100% of the subjects. The increase of the values achieved by the 2D U-Net trained on 90% of the subjects to the values achieved by the 2D U-Net trained with data augmentation is 1.1%, 0.6%, 1.2%, and 1.1% respectively. The difference between the 2D nnU-Net and the other model architectures is smaller when data augmentation is employed during training compared to when data augmentation is not applied. The average Dice Coefficient and average Jaccard Index are improved for all model architectures as a result of employing data augmentation. This leads to the conclusion, that the data augmentation pipeline implemented in this thesis works for all model architectures trained on the brain tumor images of the BRATS20 dataset.

Based on both the Dice Coefficient and Jaccard Index, in comparison to the other model architectures, the 2D nnU-Net seems to improve the least from employing data augmentation. All the other three model architectures appear to benefit more from employing data augmentation. The Transformer based architectures display the greatest amount of improvement. A reasonable explanation for all the model architectures improving is that by employing data augmentation, more data than 100% of the training subjects are provided for training. Especially for the Transformer based architectures, which generally need more training data to learn than the other model architectures, we observe a great gain in employing data augmentation.

The effects on the average Dice Coefficient values with the addition of data augmentation during training are similar to the average Jaccard Index values. In regard to both the Dice Coefficient and Jaccard Index, the 2D nnU-Net performs the best, followed by the Swin Transformer, the 2D-Unet, and the Vision Transformer. However, the nnU-Net exhibits the smallest increase of values across both metrics. The Swin Transformer and Vision Transformer have the largest increase and thus gain the most from employing data augmentation during training. Whether this indicates that these models would benefit the most from an increase in training data cannot be stated for certain. Furthermore, if the Swin Transformer can overtake the 2D nnU-Net as the best-performing model architecture when both model architectures are trained on significantly more brain tumor images, cannot be deduced from our experiment.

Precision and Recall

In terms of precision, the 2D nnU-Net trained with data augmentation and on 100% of the training subjects outperforms all other architectures on three out of the four categories trained the same way. However, for ET, the Swin Transformer outperforms the 2D nnU-Net. Furthermore, the 2D nnU-Net outperforms all model architectures in regard to recall, in all categories.

When trained with data augmentation and on 100% of the training subjects, the Swin Transformer, Vision Transformer, 2D U-Net, and 2D nnU-Net, all achieve better average recall values for all categories NCR, ED, ET, and Mean, respectively, compared to when data augmentation is not employed. This demonstrates that data augmentation, and increasing training data, result in increased average recall values for each model architecture. However, the average precision values are not improved for the 2D nnU-Net when data augmentation is employed during training.

In comparison to the other model architectures, the Transformer based architectures seem to improve the most from employing data augmentation. This is possibly because the Transformer based architectures, need more training data to learn, while the 2D U-Net based architectures in our experiments might have already reached a stage, where they gain less from more training data. Considering that employing data augmentation for the Transformer

based architectures also resulted in the largest increase for the Dice Coefficient and Jaccard Index values, further emphasizes that the Transformer based architectures greatly benefit from having more training data.

Additionally, when data augmentation is applied, the difference in precision and recall between the Transformer architectures and the 2D nnU-Net is smaller than when data augmentation is not employed. This could imply that the Vision Transformer and the Swin Transformer could reach comparable values to the 2D U-Net and 2D nnU-Net, respectively, when trained on even more training data. However, this would have to be further investigated.

The Evaluation Metrics on HGG and LGG Subjects

Unsurprisingly, since each evaluation metric is increased when data augmentation is employed, the increase is observed for LGG and HGG subjects, respectively. This further indicates that the data augmentation pipeline employed in this thesis overall improved results. Except for the evaluation metrics values related to the 2D nnU-Net, all other values are noticeably increased. The 2D nnU-Net convincingly performs the best, followed by the Swin Transformer, 2D U-Net, and Vision Transformer.

5.3 Method

This section entails a discussion of the selected method employed to generate results. It also includes an explanation of possible alternative approaches and why they were not viable for our work.

The approach used to measure segmentation time for each of the model architectures did not take into account the difference in implementation between the model architectures. Hence, it was not possible to make a valid comparison that would give strong implications on which model architecture is the fastest at segmentation time. A conclusive outcome would have been obtained if all four model architectures had employed the same approach for predicting segmentation masks. The GPU memory usage recorded for the 2D U-Net was the highest out of all model architectures, which is unexpected since the Transformer based model architectures are larger architectures than the 2D U-Net architecture. This highlights that the difference in how the model architectures were implemented directly affected the GPU memory usage of the model architectures. This consequently resulted in a non-definitive comparison of the GPU memory usage between each model architecture. If all model architectures were implemented similarly, a more conclusive result could have been reached.

In this thesis, various values for the different hyperparameters were not explored. Not exploring different values for the hyperparameters may have introduced a bias towards certain model architectures in the results. For instance, applying a learning rate scheduler with warmup could provide greater benefits to the Transformer based model architectures compared to other model architectures. Furthermore, using a different learning rate update method or not using one at all could have a significant impact on the results. Due to time constraints, it was not feasible to conduct a more extensive comparative analysis that investigates various values for the hyperparameters of this thesis. Moreover, the time limitation prevented us from employing different loss functions together with the model architectures.

There is a large variation in class frequencies within the brain tumor images of the BRATS20 dataset. A substantial portion of the pixels in the brain tumor images is not categorized as the classes NCR, ED, and ET. Additionally, a difference in class frequency between NCR, ED, and ET exists. One potential drawback with the employed method is that the difference in class frequency is not directly addressed. For the DiceCELoss function, while the DiceLoss component inherently addresses the class frequency difference present in the data, the CrossEntropyLoss component does not. A way to directly take the class imbalance into account would have been to compute and incorporate class weights for each of the classes

together with the loss function. The class weight for each class is inversely proportional to its respective class frequency in the dataset. With this approach, the class imbalance could have been directly addressed.

The data augmentation pipeline applied in Experiment 2 was not compared to other data augmentation pipelines with different data augmentation techniques or various degrees of data augmentation. Employing an alternative data augmentation pipeline might have further enhanced the results and potentially rendered them more conclusive. For instance, whether Transformer based architectures considerably benefit more from data augmentation than 2D U-Net based architecture, could have been investigated further.

5.4 Future Work

In this thesis, our primary focus was on the approach outlined in chapter 3, where we made an effort to incorporate as many relevant aspects as possible within the limitations of our project's resources. Based on our observations during the thesis work, we are inclined to believe that there are several opportunities for further investigations in the future. In this section we provide some recommendations for possible inclusions concerning this matter.

For each increment of the training data and each model architecture, the Dice Coefficient, Jaccard Index, precision, and recall, respectively, increase. However, the observed increases of the four metrics for the Transformer based architectures are not as large as for the 2D U-Net based architectures. This highlights the Transformer based architectures' inherent need to be trained on large datasets in order to learn. Since the amount of training data used in this thesis seems to be insufficient for the Transformer based architectures to reach their full potential to learn, further investigation with more training data is desirable. A possible approach would be to use the brain tumor images from the BraTS 2021 dataset, which contains more images than the BraTS 2020 dataset. However, a distinction between HGG and LGG subjects would not be possible in this case. Another way to increase the training data would be to employ a more aggressive data augmentation pipeline that alters the images to a greater extent and more frequently. Nevertheless, an overly aggressive data augmentation pipeline would alter the characteristics of the brain tumor images and the dataset excessively, potentially leading to poor generalization on unseen brain tumor images.

To compensate for the requirement of training on large datasets, Transformer based architectures are usually pre-trained before being trained on the dataset associated with the downstream task. A popular dataset to pre-train Transformer models on is the ImageNet dataset. The creators of the Vision Transformer, as well as the creators of the Swin Transformer, pre-trained their architectures on ImageNet-21K. A future investigation could focus on comparing the 2D U-Net and the 2D nnU-Net to an ImageNet pre-trained Swin Transformer and Vision Transformer. Whether the ImageNet pre-trained Swin Transformer could overtake the 2D nnU-Net as the best-performing model architecture in this thesis' experiments, would be interesting to explore.

Another unexplored approach is to pre-train the Swin Transformer and the Vision Transformer on a large dataset of medical images. The RadImageNet dataset, introduced by Mei et al. [44], is a large dataset of medical images, which comprises 1.35 million annotated images of CT, MRI, and ultrasound scans, covering various medical fields such as musculoskeletal, neurologic, oncologic, gastrointestinal, endocrine, and pulmonary pathology. It would be interesting to investigate how a Swin Transformer and a Vision Transformer, both pre-trained on RadImageNet, would compare to the other model architectures in the experiments of this thesis. Furthermore, which of the two datasets, ImageNet and RadImageNet, results in the better Vision Transformer and Swin Transformer, would also be compelling to explore.

6 Conclusion

In this thesis, two Transformer based architectures and two 2D U-Net based architectures, trained on brain tumor images from the BraTS 2020 dataset were compared on the downstream task of brain tumor segmentation. More specifically, the model architectures, Swin Transformer, Vision Transformer, 2D U-Net, and 2D nnU-Net, were trained on increasing amounts of data to explore the improvements of each architecture with increasing training data. We compared the model architectures on total training time, segmentation time, and memory usage. Furthermore, each model architecture's ability to generate correct predicted segmentation masks on unseen data was evaluated by conducting comparisons based on the Dice Coefficient, the Jaccard Index, precision, and recall. The images contained in the BraTS 2020 dataset are of two brain tumor types, high-grade glioma (HGG) and low-grade glioma (LGG). We additionally assessed how each of the model architectures performs on brain tumor images of HGG and LGG, respectively.

Each of the research questions introduced in chapter 1 and their corresponding answer are listed below.

1. **When trained on increasing amounts of brain tumor images from the BraTS 2020 dataset, how do a Swin Transformer, a Vision Transformer, a typical 2D U-net, and a 2D U-Net trained using the nnU-Net framework, compare in terms of Dice Coefficient, Jaccard Index, precision, recall and training time?**

   The 2D nnU-Net finishes training the fastest, followed by the 2D U-Net, Vision Transformer, and Swin Transformer. The Transformer based architectures take the longest time to train, and their respective measured total training time is similar. There is an approximate one-hour difference between the training time of the 2D nnU-Net and the 2D U-Net, as well as between the 2D U-Net and the Transformer architectures. When the training data is increased, these time differences get larger. The Transformer architectures trained on 100% of our training data, take five hours to train, while the 2D nnU-Net trained on the same data takes approximately 30 minutes.

   The 2D nnU-Net achieves the highest values for Dice Coefficient, Jaccard Index, and recall, out of all the model architectures. The Swin Transformer achieves comparable precision values to the 2D nnU-Net. Compared to the 2D U-Net and the Vision Transformer, the 2D nnU-Net and the Swin Transformer achieve considerably higher evaluation metric values. Taking all evaluation metrics into account, the 2D nnU-Net

achieves the highest values, followed by the Swin Transformer, the 2D U-Net, and the Vision Transformer. When the training data is increased, this rank order stays the same. Moreover, there is a lack of noticeable increase in the evaluation metric values for the Transformer architectures as the training data is increased.

2. **In terms of segmentation time and GPU memory usage, how do the model architectures compare to each other?**

   The Vision Transformer demonstrates an average segmentation time of around 200 seconds, while the Swin Transformer, 2D nnU-Net, and 2D U-Net follow with segmentation times of approximately 254, 272, and 294 seconds, respectively. In terms of GPU memory usage, the 2D nnU-Net occupies 3114 MiB during training, which is substantially less memory than the other model architectures. The GPU memory usage for the 2D U-Net, the Swin Transformer, and the Vision Transformer is 11312, 10820, and 10416 MiB, respectively. Due to implementation differences, the segmentation time and GPU memory usage comparison are not as conclusive as we hoped.

3. **Which of the model architectures perform better in correctly segmenting MRI scans of patients with LGG and HGG, respectively?**

   All model architectures in this thesis are trained on training data containing a similar distribution of subjects labeled with HGG and LGG, resembling the distribution in the training data of the BraTS 2020 dataset. For the training data of the BraTS 2020 dataset, this distribution is 79.4% HGG subjects and LGG subjects. Because each model architecture is trained on significantly more HGG images than LGG images, all model architectures perform better on unseen data of HGG subjects than on unseen data of LGG subjects. Overall, the 2D nnU-Net performs the best in correctly segmenting MRI scans of patients with LGG and HGG, respectively. The Swin Transformer performs the second best, followed by the 2D U-Net and the Vision Transformer. The 2D nnU-Net and the Swin Transformer are considerably better than the 2D U-Net and the Vision Transformer.

4. **How does the use of data augmentation during training impact the ability of each model architecture to accurately segment MRI scans?**

   All model architectures benefited from employing the data augmentation pipeline implemented in this thesis. The Swin Transformer and the Vision Transformer improved the most from applying the data augmentation pipeline, followed by the 2D U-Net and the 2D nnU-Net. The 2D nnU-Net did not benefit noticeably from the addition of the data augmentation pipeline during training. When the data augmentation pipeline is applied during training, the 2D nnU-Net performs the best in correctly segmenting MRI scans of patients with LGG, HGG, and both, respectively. The Swin Transformer performs the second best, followed by the 2D U-Net and the Vision Transformer.

We conclude that in this thesis we were able to perform a successful comparative analysis effectively showcasing the distinct advantages of the four model architectures under discussion. Future comparisons could incorporate additional factors, as outlined in section 5.4. Our experiments demonstrate that the 2D nnU-Net architecture stands out as the most suitable model architecture for 2D brain tumor segmentation, given our specific experimental conditions. However, further investigation is required to validate the models' performance under diverse conditions in the future.

Bibliography

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[2] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". English (US). In: 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015. Jan. 2015.

[3] Ujjwal Baid, Satyam Ghodasara, Suyash Mohan, Michel Bilello, Evan Calabrese, Errol Colak, Keyvan Farahani, Jayashree Kalpathy-Cramer, Felipe C Kitamura, Sarthak Pati, et al. "The RSNA-ASNR-MICCAI BraTS 2021 benchmark on brain tumor segmentation and radiogenomic classification". In: *arXiv preprint arXiv:2107.02314* (2021).

[4] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin Kirby, John Freymann, Keyvan Farahani, and Christos Davatzikos. *Segmentation Labels and Radiomic Features for the Pre-operative Scans of the TCGA-GBM collection*. July 2017. DOI: 10.7937/K9/TCIA.2017.KLXWJJ1Q.

[5] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin Kirby, John Freymann, Keyvan Farahani, and Christos Davatzikos. *Segmentation Labels and Radiomic Features for the Pre-operative Scans of the TCGA-LGG collection*. July 2017. DOI: 10.7937/K9/TCIA.2017.GJQ7R0EF.

[6] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin S. Kirby, John B. Freymann, Keyvan Farahani, and Christos Davatzikos. "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features". In: *Scientific Data* 4 (2017).

[7] Spyridon Bakas, Mauricio Reyes, András Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Shinohara, Christoph Berger, Sung Ha, Martin Rozycki, Marcel Prastawa, Esther Alberts, Jana Lipkova, John Freymann, Justin Kirby, Michel Bilello, Hassan Fathallah-Shaykh, Roland Wiest, Jan Kirschke, and Zhaolin Chen. "Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge". In: (Mar. 2019), p. 38.

[8] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://aclanthology.org/D14-1179.

[9] Marco Domenico Cirillo, David Abramian, and Anders Eklund. "What is The Best Data Augmentation For 3D Brain Tumor Segmentation?" In: *2021 IEEE International Conference on Image Processing (ICIP)*. 2021, pp. 36–40. DOI: 10.1109/ICIP42928.2021.9506328.

[10] MMSegmentation Contributors. *MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark*. https://github.com/open-mmlab/mmsegmentation. 2020.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[12] Lee R. Dice. "Measures of the Amount of Ecologic Association Between Species". In: *Ecology* 26.3 (1945), pp. 297–302. DOI: https://doi.org/10.2307/1932409. eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.2307/1932409. URL: https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[14] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[15] Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou. "Word embeddings quantify 100 years of gender and ethnic stereotypes". In: *Proceedings of the National Academy of Sciences* 115.16 (2018), E3635–E3644.

[16] Anthony Gillioz, Jacky Casas, Elena Mugellini, and Omar Abou Khaled. "Overview of the Transformer-based Models for NLP Tasks". In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. 2020, pp. 179–183. DOI: 10.15439/2020F20.

[17] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[20] Shijie Hao, Yuan Zhou, and Yanrong Guo. "A brief survey on semantic segmentation with deep learning". In: *Neurocomputing* 406 (2020), pp. 302–321.

[21] Ali Hatamizadeh, V. Nath, Yucheng Tang, Dong Yang, Holger R. Roth, and Daguang Xu. "Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images". In: *BrainLes@MICCAI*. 2022.

[22] Ali Hatamizadeh, Yucheng Tang, Vishwesh Nath, Dong Yang, Andriy Myronenko, Bennett Landman, Holger R Roth, and Daguang Xu. "Unetr: Transformers for 3d medical image segmentation". In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2022, pp. 574–584.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[24] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[25] Eric C Holland. "Progenitor cells and glioma formation". In: *Current opinion in neurology* 14.6 (2001), pp. 683–688.

[26] Fabian Isensee, Paul F Jaeger, Simon AA Kohl, Jens Petersen, and Klaus H Maier-Hein. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation". In: *Nature methods* 18.2 (2021), pp. 203–211.

[27] Fabian Isensee, Paul F. Jäger, Peter M. Full, Philipp Vollmuth, and Klaus H. Maier-Hein. "nnU-Net for Brain Tumor Segmentation". In: *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*. Ed. by Alessandro Crimi and Spyridon Bakas. Cham: Springer International Publishing, 2021, pp. 118–132. ISBN: 978-3-030-72087-2.

[28] Fabian Isensee, Philipp Kickingereder, Wolfgang Wick, Martin Bendszus, and Klaus H Maier-Hein. "No new-net". In: *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 4th International Workshop, BrainLes 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Revised Selected Papers, Part II 4*. Springer. 2019, pp. 234–244.

[29] Paul Jaccard. "Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines". In: *Bull Soc Vaudoise Sci Nat* 37 (1901), pp. 241–272.

[30] Michael I. Jordan. "Serial Order: A Parallel Distributed Processing Approach". In: *Advances in psychology* 121 (1997), pp. 471–495.

[31] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014).

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[33] Pier Luca Lanzi. "Classifier Systems". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 172–178. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_115. URL: https://doi.org/10.1007/978-0-387-30164-8_115.

[34] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[35] Jianning Li, Marius Erdt, Firdaus Janoos, Ti-chiun Chang, and Jan Egger. "1 - Medical image segmentation in oral-maxillofacial surgery". In: *Computer-Aided Oral and Maxillofacial Surgery*. Ed. by Jan Egger and Xiaojun Chen. Academic Press, 2021, pp. 1–27. ISBN: 978-0-12-823299-6. DOI: https://doi.org/10.1016/B978-0-12-823299-6.00001-8. URL: https://www.sciencedirect.com/science/article/pii/B9780128232996000018.

[36] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[37] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[38] Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. URL: https://smlbook.org.

[39] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. "A survey on deep learning in medical image analysis". In: *Medical image analysis* 42 (2017), pp. 60–88.

[40] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 9992–10002. DOI: 10.1109/ICCV48922.2021.00986.

[41] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[42] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2017.

[43] David N. Louis, Arie Perry, Pieter Wesseling, Daniel J. Brat, Ian A. Cree, Dominique Figarella-Branger, Cynthia Hawkins, Ho-keung Ng, Stefan M. Pfister, Guido Reifenberger, Riccardo Soffietti, Andreas von Deimling, and David W. Ellison. "The 2021 WHO Classification of Tumors of the Central Nervous System: a summary." In: *Neuro-oncology* (2021).

[44] Xueyan Mei, Zelong Liu, Philip M. Robson, Brett Marinelli, Mingqian Huang, Amish Doshi, Adam Jacobi, Chendi Cao, Katherine E. Link, Thomas Yang, Ying Wang, Hayit Greenspan, Timothy Deyer, Zahi A. Fayad, and Yang Yang. "RadImageNet: An Open Radiologic Deep Learning Research Dataset for Effective Transfer Learning". In: *Radiology: Artificial Intelligence* 0.ja (0), e210315. DOI: 10.1148/ryai.210315. eprint: https://doi.org/10.1148/ryai.210315. URL: https://doi.org/10.1148/ryai.210315.

[45] Bjoern H. Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, Levente Lanczi, Elizabeth Gerstner, Marc-André Weber, Tal Arbel, Brian B. Avants, Nicholas Ayache, Patricia Buendia, D. Louis Collins, Nicolas Cordier, Jason J. Corso, Antonio Criminisi, Tilak Das, Hervé Delingette, Çağatay Demiralp, Christopher R. Durst, Michel Dojat, Senan Doyle, Joana Festa, Florence Forbes, Ezequiel Geremia, Ben Glocker, Polina Golland, Xiaotao Guo, Andac Hamamci, Khan M. Iftekharuddin, Raj Jena, Nigel M. John, Ender Konukoglu, Danial Lashkari, José António Mariz, Raphael Meier, Sérgio Pereira, Doina Precup, Stephen J. Price, Tammy Riklin Raviv, Syed M. S. Reza, Michael Ryan, Duygu Sarikaya, Lawrence Schwartz, Hoo-Chang Shin, Jamie Shotton, Carlos A. Silva, Nuno Sousa, Nagesh K. Subbanna, Gabor Szekely, Thomas J. Taylor, Owen M. Thomas, Nicholas J. Tustison, Gozde Unal, Flor Vasseur, Max Wintermark, Dong Hye Ye, Liang Zhao, Binsheng Zhao, Darko Zikic, Marcel Prastawa, Mauricio Reyes, and Koen Van Leemput. "The Multimodal Brain Tumor Image Seg-

mentation Benchmark (BRATS)". In: *IEEE Transactions on Medical Imaging* 34.10 (2015), pp. 1993–2024. DOI: 10.1109/TMI.2014.2377694.

[46] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *2016 Fourth International Conference on 3D Vision (3DV)*. 2016, pp. 565–571. DOI: 10.1109/3DV.2016.79.

[47] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. "Image segmentation using deep learning: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 44.7 (2021), pp. 3523–3542.

[48] Jakub Nalepa, Michal Marcinkiewicz, and Michal Kawulok. "Data augmentation for brain-tumor segmentation: a review". In: *Frontiers in computational neuroscience* 13 (2019), p. 83.

[49] Dong Nie, Han Zhang, Ehsan Adeli, Luyan Liu, and Dinggang Shen. "3D Deep Learning for Multi-modal Imaging-Guided Survival Time Prediction of Brain Tumor Patients". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*. Ed. by Sebastien Ourselin, Leo Joskowicz, Mert R. Sabuncu, Gozde Unal, and William Wells. Cham: Springer International Publishing, 2016, pp. 212–220. ISBN: 978-3-319-46723-8.

[50] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[51] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[52] David E. Rumelhart, James L. McClelland, and PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. The MIT Press, July 1986. ISBN: 9780262291408. DOI: 10.7551/mitpress/5236.001.0001. URL: https://doi.org/10.7551/mitpress/5236.001.0001.

[53] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN: 9780134610993. URL: http://aima.cs.berkeley.edu/.

[54] M. Schuster and K.K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[55] Bharat Singh and Larry S Davis. "An analysis of scale invariance in object detection snip". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3578–3587.

[56] Bharat Singh, Mahyar Najibi, and Larry S Davis. "Sniper: Efficient multi-scale training". In: *Advances in neural information processing systems* 31 (2018).

[57] T Sørensen. "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons". In: *Kongelige Danske Videnskabernes Selskab* 5.4 (1948), pp. 1–34.

[58] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. "Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations". In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Ed. by M. Jorge Cardoso, Tal Arbel, Gustavo Carneiro, Tanveer Syeda-Mahmood, João Manuel R.S. Tavares, Mehdi Moradi, Andrew Bradley, Hayit Greenspan, João Paulo Papa, Anant Madabhushi, Jacinto C. Nascimento, Jaime S. Cardoso, Vasileios Belagiannis, and Zhi Lu. Cham: Springer International Publishing, 2017, pp. 240–248. ISBN: 978-3-319-67558-9.

[59] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *NIPS*. 2014.

[60] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop, coursera: Neural networks for machine learning". In: *University of Toronto, Technical Report* 6 (2012).

[61] Tijmen Tieleman, Geoffrey Hinton, et al. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[62] Kai Ming Ting. "Confusion Matrix". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 209–209. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_157. URL: https://doi.org/10.1007/978-0-387-30164-8_157.

[63] Kai Ming Ting. "Precision and Recall". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 781–781. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_652. URL: https://doi.org/10.1007/978-0-387-30164-8_652.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[65] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. "Residual attention network for image classification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3156–3164.

[66] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. "Unified perceptual parsing for scene understanding". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 418–434.

[67] Shashank Yadav. *Understanding Attention Mechanism*. https://shashank7-iitd.medium.com/understanding-attention-mechanism-35ff53fc328e. Accessed on: 16 May. 2023.

[68] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. "Pyramid Scene Parsing Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6230–6239. DOI: 10.1109/CVPR.2017.660.

[69] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. "Semantic understanding of scenes through the ade20k dataset". In: *International Journal of Computer Vision* 127 (2019), pp. 302–321.

[70] Zhou and Chellappa. "Computation of optical flow using a neural network". In: *IEEE 1988 International Conference on Neural Networks*. 1988, 71–78 vol.2. DOI: 10.1109/ICNN.1988.23914.