ACIT5900

MASTER THESIS

in

Applied Computer and Information Technology
(ACIT)

May 2023

Applied Artificial Intelligence

Generating Synthetic Medical Images with 3D GANs

Håkon Guttulsrud

Department of Computer Science

Faculty of Technology, Art and Design

Preface

It has been a fulfilling and enlightening experience to undertake the writing of this thesis, which focuses on the use of artificial intelligence to generate synthetic medical images with the potential to enhance computer-assisted cancer diagnosis.

During my research, I have been fortunate to receive guidance and support from several individuals. I would like to express my appreciation to my thesis advisors, Michael Riegler and Vajira Thambawita, who have provided me with invaluable support and inspiration. Additionally, I would like to extend my gratitude to my fellow students for their encouragement and advice.

I cannot overlook the contributions of the patients whose medical images formed the foundation of this thesis. Their generosity in sharing their data has been crucial in the creation of synthetic images that have the potential to benefit not just themselves but also countless others who may benefit from the results of this thesis.

Oslo, 15.05.2023

Håkon Guttulsrud

Abstract

This thesis presents a novel approach to overcoming the challenges associated with the scarcity of annotated medical image data, a significant hurdle in cancer detection. We propose the use of Generative Adversarial Networks (GANs) to generate synthetic high-resolution 3-dimensional CT images and corresponding tumor masks, thereby enhancing the volume and diversity of available training data for machine learning models.

This thesis seeks to answer two primary questions: Can GANs generate realistic high-resolution 3D CT image/mask pairs? And to what extent do synthetic CT images generated by GANs impact the accuracy of a state-of-the-art cancer segmentation model?

The thesis conducts a comprehensive evaluation of various GAN models, including Vanilla GAN, Wasserstein GAN, StyleGAN2, FastGAN, and Hierarchical Amortized GAN (HA-GAN). The HA-GAN model, in particular, showed exceptional potential, demonstrating superior capacity in generating high-resolution synthetic images. These results were substantiated by multiple evaluation metrics, such as the Inception score, Frechet Inception Distance (FID), and a Visual Turing Test, where the synthesized images were presented to healthcare professionals. The results were compelling, as the professionals were frequently unable to distinguish between synthetic and real images.

The synthetic data, when integrated with actual images, facilitated the training of the SegResNet model from the MONAI's Auto3Dseg framework, aiming to optimize the accuracy of tumor segmentation in 3D CT images. Notably, incorporating synthetic images into the training set significantly boosted the Dice Similarity Coefficient (DSC), thereby affirming the effectiveness of our proposed method. The proposed method surpassed the baseline score of 0.48693, achieving an improved score of 0.52193 DSC.

This research's foremost contribution lies in its methodology for generating high-definition 3D CT image/mask pairs, thus significantly enriching existing medical datasets and enhancing the precision of medical image segmentation.

Moreover, the thesis work has produced two unique datasets of synthesized image/mask pairs with resolutions of $128 \times 128 \times 64$ and $256 \times 256 \times 128$, available for distribution. The code artifact and datasets are available in appendices.

The work in this thesis underscores the potential of GANs, with a special emphasis on HA-GAN, to surmount the challenge of limited medical training data, thereby pushing the boundaries of machine learning models in the realm of medical image segmentation and cancer detection.

Contents

List of Figures

List of Tables

1 Introduction

1.1 The Importance of Medical Imaging in Cancer Detection

Cancer is a diverse and intricate group of illnesses characterized by the uncontrolled growth and spread of abnormal cells. It is one of the leading global causes of death (Bray et al., 2021), and early detection is vital for effective treatment. The prevalence of cancer has been increasing over the years (Ugai et al., 2022), making it a significant public health concern. Early cancer detection is crucial because it can significantly impact patient outcomes and prognosis. Cancer is easier to treat when it is detected early and has not spread beyond the site of origin. When cancer cells spread to other body parts, it becomes more challenging to treat, and the chances of survival decrease. In contrast, early detection allows for prompt treatment, increasing the likelihood of survival.

Moreover, accurate cancer detection is crucial for tailoring personalized treatment plans to the patient's specific cancer type and stage. Each cancer type requires different treatment approaches, and early detection can help clinicians determine the most effective treatment options. However, false cancer detection can have severe consequences for patients. Misdiagnosis, delayed diagnosis, or underdiagnosis can result in incorrect treatment approaches, leading to disease progression, a worsened prognosis, or even death. Therefore, accurate and reliable cancer detection methods are crucial to avoid unnecessary treatment or patient harm.

Medical imaging technologies have dramatically improved the detection of cancer, providing physicians with invaluable tools to identify and manage the disease. These imaging modalities offer non-invasive and precise methods for visualizing the body's internal organs, tissues, and structures, enabling early detection of cancerous growths.

Magnetic resonance imaging (MRI) uses magnets and radio waves to create images of the internal structures in the body. MRIs are particularly beneficial for detecting brain and spinal cord tumors, breast cancer, and other soft tissue cancers. MRI images can also provide information about the structure and function of organs, which can help clinicians develop personalized treatment plans (Vlaardingerbroek and Boer, 2013).

Positron emission tomography (PET) scans use a radioactive tracer to detect changes in the body's metabolic activity. PET scans can help detect and stage various types of cancer, including lung, colorectal, and lymphoma. PET scans can also help evaluate the effectiveness of cancer treatment, allowing clinicians to modify the treatment plan as necessary (Muehllehner and Karp, 2006).

Computed Tomography (CT) scans use X-rays and computer processing to create detailed cross-sectional images of the body's internal structures, helping to detect a wide range of cancers, such as lung, liver, and pancreatic cancers. CT scans are beneficial for detecting minor abnormalities, providing information about the size, shape, and location of tumors, and assessing whether they have spread to other body areas (Buzug, 2011).

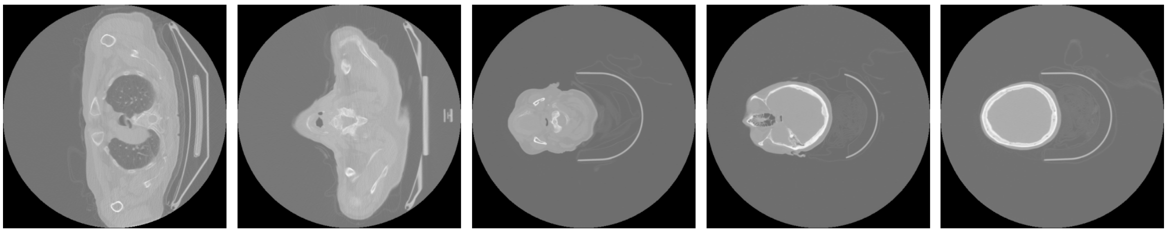


Figure 1.1: Axial View of a CT Scan

Medical imaging technologies are critical in detecting, diagnosing, and treating cancer. It allows doctors to identify tumors early, when they are most treatable, and monitor the effectiveness of cancer treatments over time.

1.2 Leveraging Machine Learning to Improve Cancer Detection

Medical imaging technologies such as MRI, PET, and CT have been greatly enhanced by the potential of Artificial Intelligence (AI) to improve the accuracy and speed of cancer detection. Machine learning (ML), a branch of AI, provides powerful algorithms that can be trained on large datasets of medical images to identify underlying patterns indicative of cancer. After the models have been trained, they can be used to detect cancer in new medical images, aiding doctors in the diagnosis (Houssein et al., 2021).

One of the primary benefits of ML in cancer detection is its ability to analyze large volumes of medical images (Sidey-Gibbons and Sidey-Gibbons, 2019). Traditionally, radiologists manually review images to identify potential signs of cancer, which can be time-consuming and subjective. ML algorithms can analyze images in seconds and provide a more objective analysis, potentially leading to a faster and more accurate diagnosis. ML can also help overcome the variability in medical images due to differences in imaging protocols, equipment, and human interpretation. By training models on a diverse range of medical images, the models can learn to recognize patterns and features indicative of cancer, regardless of the source of the images (Dexter et al., 2020).

The potential of ML to improve cancer detection using medical imaging technologies is vast. With the development of accurate models, machine learning in cancer detection can potentially lead to faster and more precise diagnoses, ultimately improving patient outcomes.

1.3 The Problem of Insufficient Training Data for Machine Learning in Medicine

Data quality and especially data quantity are crucial in ML because the underlying data directly impacts the performance and accuracy of the model (Jain et al., 2020). The development of accurate ML models heavily depends on the training data. To recognize patterns and make accurate predictions, ML models require large amounts of high-quality data. Insufficient or poor-quality data can result in inaccurate or biased models that produce incorrect or unreliable results. Therefore, ensuring that the training data is representative, diverse, and free of errors is essential.

In the field of medicine, the issue of insufficient quality data for machine learning can be especially challenging. Healthcare data is often incomplete, inconsistent, and subject to errors and biases, limiting the effectiveness of machine learning models. Several factors can be attributed to the problem of insufficient quality data in medicine. One of the primary factors is the need for more standardization in medical data collection, as medical data is collected using various methods and formats, making comparing and analyzing data from different sources difficult. The lack of standardization can lead to high data quality and format variability.

Additionally, healthcare data is often fragmented and scattered across various healthcare systems, making it difficult to access and integrate for analysis. The inherent complexity of medical data, with multiple variables and noise sources, also makes it challenging to collect and analyze high-quality data. Another factor is due to privacy concerns. Medical images contain sensitive information about patients' health, limiting their sharing and use due to ethical and legal constraints. While this is essential to protecting patient privacy, it can also limit the amount of data available for research. This further complicates the development of new diagnostic and treatment tools, such as ML models. In summary, insufficient quality data in medicine is a significant obstacle to developing accurate and reliable machine-learning models for cancer detection.

1.4 Research Questions

In this thesis, it is proposed to utilize Generative Adversarial Networks (GANs) (Aggarwal et al., 2021), an ML algorithm, to generate synthetic medical images to solve the problem of insufficient quality data in the field of medicine. The use of GANs presents a promising method to address the challenge of limited and low-quality medical imaging data by generating synthetic images that can enhance the performance of ML models. Specifically, this thesis will focus on creating 3-dimensional CT images and their corresponding masks that highlight the tumor area in the CT image. In order to optimize their utility for future applications, it is desirable that the synthetic image/mask pairs have the highest attainable resolution. The first research question is as follows:

Research question 1 *Can GANs generate realistic high-resolution 3-dimensional CT image/mask pairs?*

The generated image/mask pairs should be properly evaluated to validate their application. By incorporating quantitative and qualitative assessments, the evaluation process can comprehensively evaluate the synthetic data quality and suitability for segmentation purposes. This ensures that the generated image/mask pairs are accurate and can be used to improve tumor segmentation in medical images. The most crucial step involves utilizing the artificially generated dataset as either the complete or a portion of the training data for a state-of-the-art segmentation model to assess the viability of the synthetic data in real-world settings. The second research question is as follows:

Research question 2 *To what extent do synthetic CT images generated by GANs impact the accuracy of a state-of-the-art cancer segmentation model?*

1.5 Main Contributions

The dataset from the HECKTOR (Heack and Neck Tumor Segmentation) 2022 challenge (Andrearczyk et al., 2023) will be used as training data. The challenge dataset contains real CT images with associated tumor masks and will serve as the dataset for training various GAN algorithms in this thesis.

Based on Research Question 1, the first objective of the thesis is to use the CT images to train a GAN algorithm to create synthetic images visually similar to authentic medical images. Moreover, the GAN should also generate corresponding image masks that comprise synthetic tumor regions. Figure 1.2 shows the overall process for Objective 1.

Objective 1 Implement and train a GAN algorithm that can generate synthetic 3-dimensional CT images and corresponding tumor masks

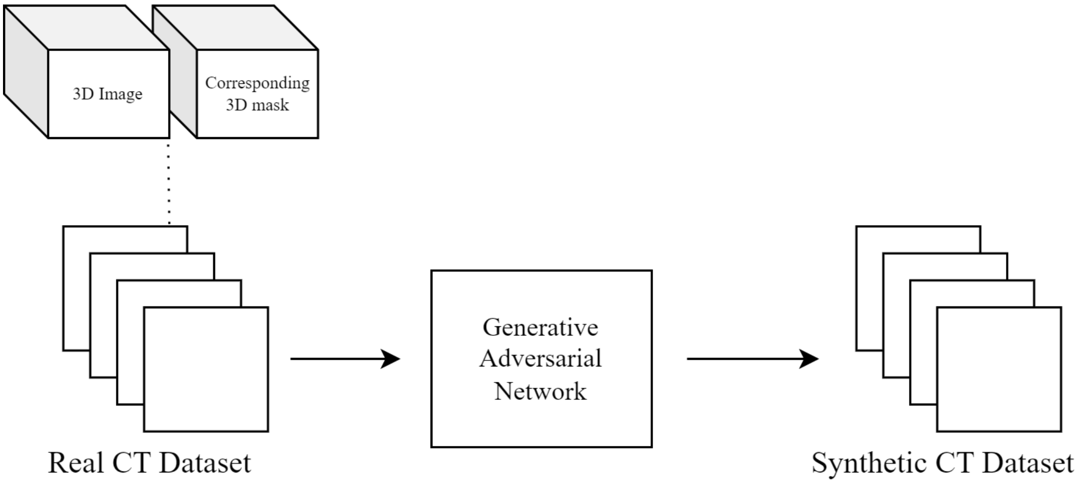


Figure 1.2: Objective 1

Based on Research Question 2, the second goal of this thesis is to evaluate the quality of synthetic medical images and their impact on the accuracy of algorithms capable of detecting cancer.

Objective 2 Implement and train a state-of-the-art cancer segmentation model and determine if using synthetic CT images can improve the accuracy

Figure 1.3 shows the overall process for Objective 2.

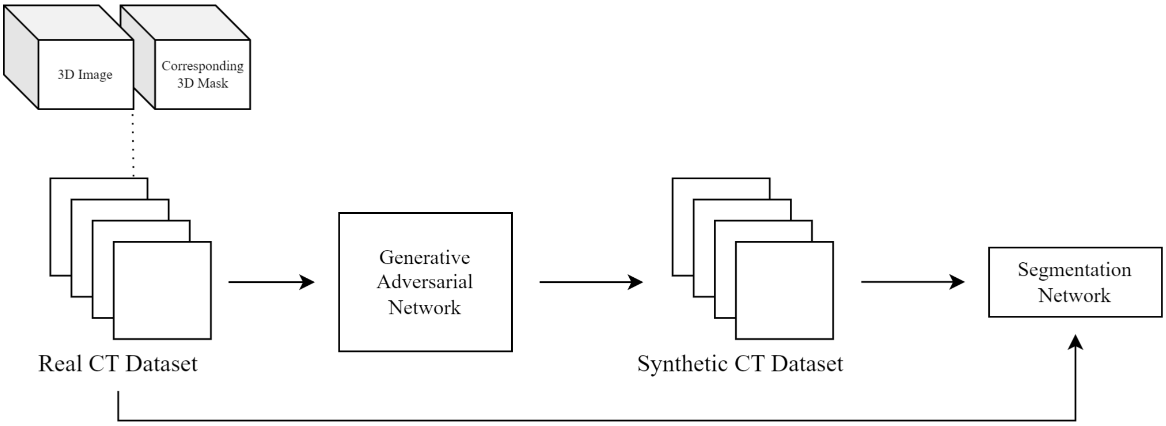


Figure 1.3: Objective 2

As a component of the thesis, GANs are utilized to produce synthetic CT datasets that will be published for use by others.

Objective 3 Create and release synthetic datasets containing CT image/mask pairs

These datasets comprise pairs of 3D image/mask sets that should closely resemble authentic CT data and will be available in resolutions $128 \times 128 \times 64$ and $256 \times 256 \times 128$. The image and mask sets are stored in separate folders and saved in NIFTI (Neuroimaging Informatics Technology Initiative) format - a standard format utilized in neuroimaging research to preserve and distribute medical imaging data.

1.6 Ethical Considerations

Applying GANs to generate synthetic images has demonstrated exceptional performance in various fields, including medical imaging (Yi et al., 2019). However, using GANs to synthesize CT images raises significant ethical concerns.

Data privacy is a major ethical concern when training GANs on medical data. The security of patient data must be safeguarded to prevent unauthorized access and misuse. Using patient data to train GANs may pose risks to patient privacy and could result in harm. To prevent these potential dangers, researchers must ensure that patient data is de-identified and appropriately secured to prevent unauthorized access. With the help of novel techniques, it is becoming more and more feasible to completely anonymize datasets, to the extent that it becomes impossible to trace any data point back to a real person (Arora and Arora, 2022). The dataset used in this thesis is entirely anonymized, with no identifying information contained in the data.

GANs rely on existing datasets to generate images, and the accuracy of synthetic images depends on the quality of the training data. There is a risk that the biases present in the training data could be amplified in the synthetic images. Researchers must use representative training datasets to generate unbiased synthetic images that are not skewed toward any specific demographic. The dataset used in this thesis is comprised of CT data from 7 different medical centers, ensuring a broad and representative population of patients.

Another significant ethical concern is the validity and reliability of synthetic CT images generated using GANs. Synthetic images may not be entirely accurate, which, if used as training data for other algorithms, could potentially lead to an incorrect diagnosis or treatment. Therefore, it is crucial that researchers validate the synthetic images generated by GANs and thoroughly evaluate the performance of the model. This thesis will evaluate the synthetic images with qualitative and quantitative metrics to ensure the reliability of the generated data.

Using GANs to generate synthetic CT images for medical applications raises several ethical considerations, including data privacy, bias, validity, and reliability. To ensure the ethical and responsible use of GANs in medical contexts, researchers must ensure that synthetic images generated by GANs are reliable, unbiased, and do not pose any risks to patients. Although GANs have the potential to advance medical research and patient care, their use must be done ethically and responsibly.

1.7 Outline

The thesis explores the potential of synthetic medical images generated by GANs for cancer detection. Chapter 1 introduced the importance of early and accurate cancer detection, the role of medical imaging in cancer detection, and the use of ML to improve cancer detection. The chapter also discusses the problem of insufficient quality data for machine learning in medicine and presents the research questions and main contributions of the thesis. Finally, the ethical considerations surrounding the research are presented.

Chapter 2 provides background information on artificial neural networks, convolutional neural networks, deep learning, and their application in medical imaging. It also reviews the current state of machine learning for cancer detection and segmentation and the use of GANs for generating synthetic data for healthcare applications. The chapter also presents different types of GANs, their limitations, and evaluation metrics. Finally, the chapter delves into the related work in the field, including the utilization of GANs for generating synthetic medical data in both 2D and 3D spaces. The chapter also covers the relevant research on the segmentation of 3D CT images.

Chapter 3 outlines the methodology, including preliminary dataset analysis, data preprocessing, visualization, and GAN models for generating synthetic medical images. It also covers data augmentation, hyperparameter optimization, and segmentation. The chapter comprehensively overviews the techniques and tools used to achieve the research objectives.

Chapter 4 presents the experiments conducted in the thesis, including using different GAN models for generating synthetic CT images and segmentation experiments.

Chapter 5 presents and discusses the results in detail, highlighting the strengths and weaknesses of the various GAN models and the segmentation network trained on synthetic data.

Chapter 6 explores the constraints and drawbacks associated with the thesis, specifically focusing on the generation of CT images using GANs, as well as the limitations of the segmentation model and experimental procedures.

Chapter 7 serves as the conclusion and outlines potential future research directions. The chapter summarizes the study's main findings and emphasizes the contributions made to medical image synthesis and segmentation.

2 Background

This chapter provides an overview of the theoretical and practical aspects of ML in medicine. We introduce Artificial Neural Networks (ANN), Deep Learning (DL), and Convolutional Neural Networks (CNN), then explore DL's application in medical imaging and cancer segmentation. We also cover GAN architectures, loss functions, hyperparameter optimization, and evaluation metrics. Finally, we review research related to this thesis.

2.1 Artificial Neural Networks

Machine Learning (ML) is a subset of artificial intelligence that uses algorithms to analyze data, learn from that data, and make predictions or decisions based on that learning. The objective of machine learning is to construct models that possess the ability to make accurate predictions or decisions without the need for explicit programming (Zhou, 2021). Artificial Neural Networks (ANNs), inspired by the structure and function of the human brain, are an ML model comprised of interconnected nodes or neurons organized into layers (Bishop, 1994). Input signals are received by each neuron from the previous layer and processed, and then the output is passed on to the next layer. An ANN is composed of different layers, including the input layer, hidden layers, and the output layer, as shown in Figure 2.1.

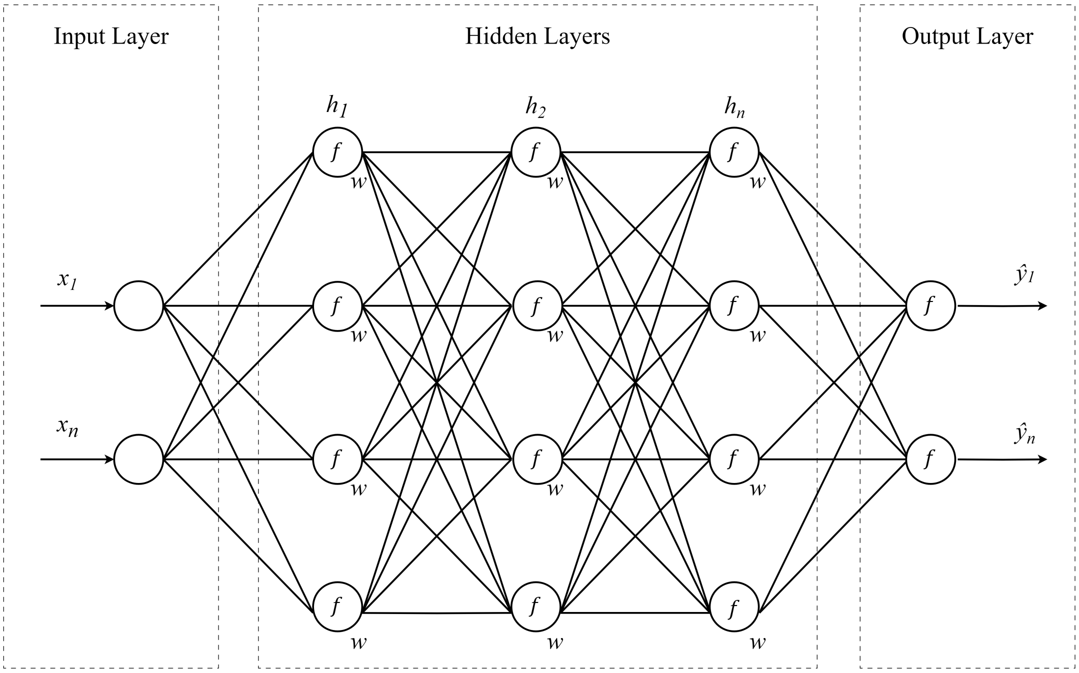


Figure 2.1: The Architecture of an Artificial Neural Network

The input layer is the first layer of the network, and it receives the input data, which could be a set of features x_n. The value of x_n depends on the specific problem being addressed and is determined by the number of features that describe the input data. The input layer does not perform any computation but merely passes the input data to the first hidden layer.

The hidden layers are intermediate layers between the input and output layers. These layers are called ”hidden” because their computations are not directly observable from the outside. The number of h_n depends on the specific problem. Each hidden layer contains a set of neurons, and each neuron computes a weighted sum of the inputs it receives, applies a non-linear activation function f to the sum of its inputs, and passes the output to the next layer. The output layer is the last layer of the network, producing the final output $\hat{y}$ of the model.

2.1.1 Training

A neural network is a set of interconnected neurons processing input data through a series of mathematical operations to produce an output. The strength of these connections between nodes is controlled by a set of parameters known as weights, denoted by w in Figure 2.1. Training a neural network involves feeding training data (x and y) to the network and adjusting these weights so that the network can accurately predict the output for a given input. The network will adjust the weights to minimize the difference between the predicted output $\hat{y}$ and the actual output y.

The most common approach to training a neural network is using backpropagation (Kelley, 1960). This algorithm involves computing the gradient of a loss function with respect to the network weights w. The loss function measures the difference between a given input's predicted and actual output. The network can gradually converge towards weights that produce accurate predictions by adjusting the weights. The backpropagation process is shown in Figure 2.2.

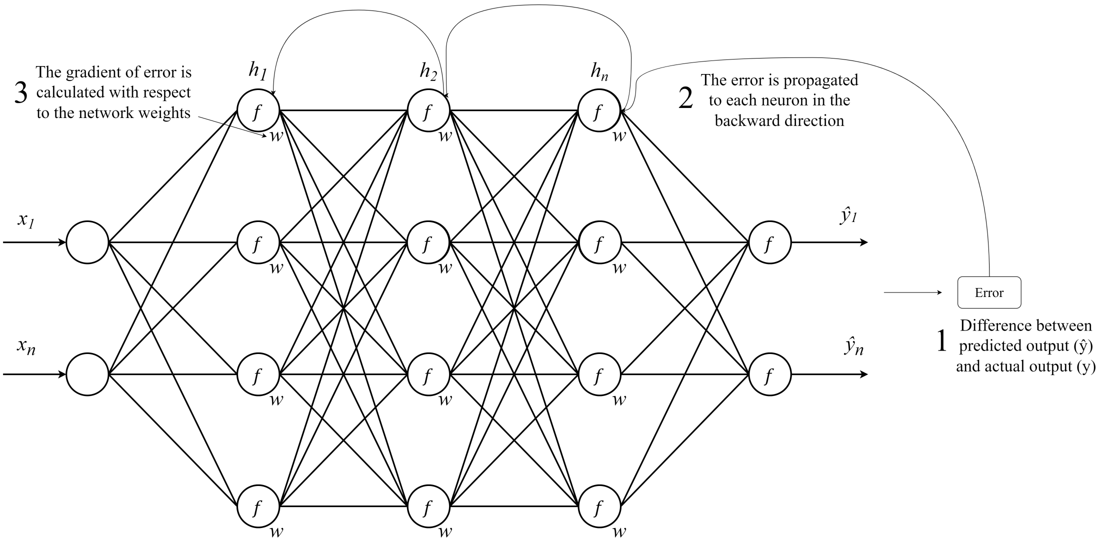

3 The gradient of error is calculated with respect to the network weights
h_1
h_2
h_n
2 The error is propagated to each neuron in the backward direction
x_1
x_n
f
w
$\hat{y}_1$
$\hat{y}_n$
Error
1 Difference between predicted output (ŷ) and actual output (y)

Figure 2.2: Backpropagation in ANNs

2.1.2 Optimization

Optimization is a critical part of training an ANN, with the most common optimization algorithm being gradient descent (Ruder, 2016). Gradient descent works by starting with a random set of weights and repeatedly adjusting them in the direction that decreases the error of the model. The ”gradient” refers to the slope of the loss function, which indicates the direction of the steepest descent. Following the gradient, the algorithm can eventually converge on the optimal set of parameters that minimizes the loss function. Figure 2.3 illustrates the gradient descent process.

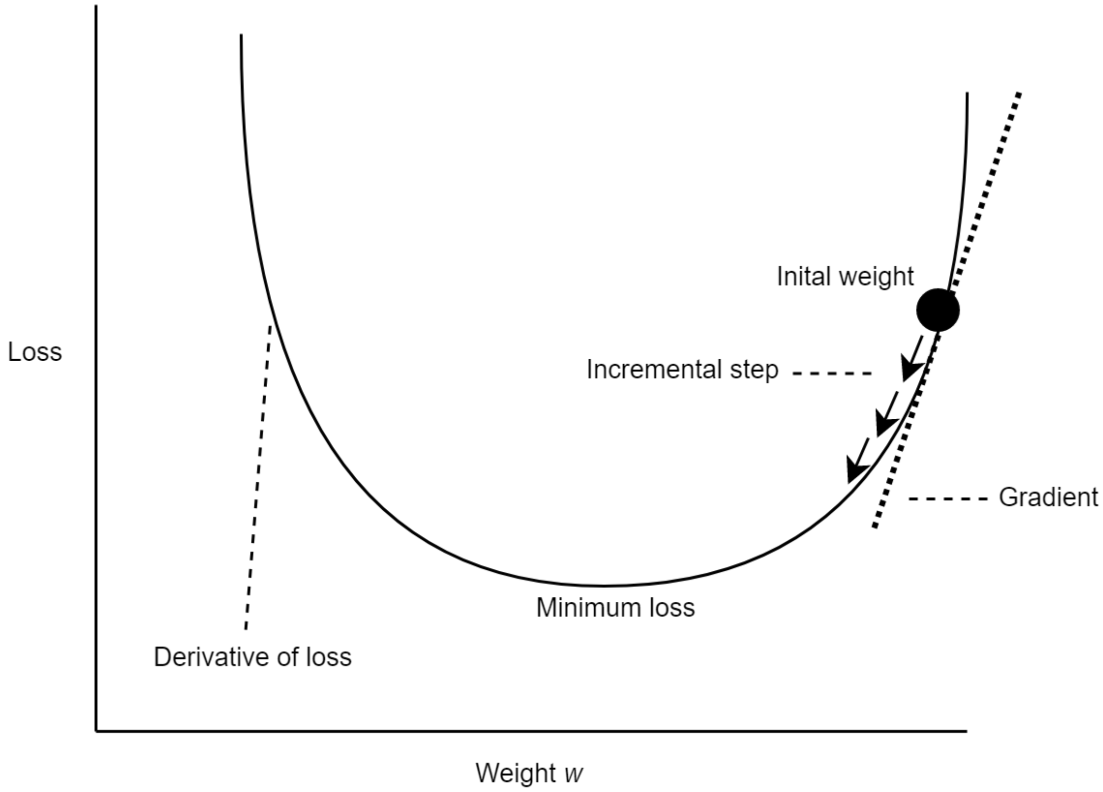


Figure 2.3: Gradient Descent

During the optimization process, the learning rate η is an important hyperparameter that controls the size of the updates to the weights (shown in Figure 2.3 as an incremental step). The learning rate determines how quickly the network learns and how sensitive it is to changes in the loss function. A high learning rate can cause the network to converge quickly but may overshoot the optimal weights. On the other hand, a low learning rate may result in slow convergence and get stuck in local minima.

However, the ”vanishing gradient” problem can arise when the gradient of the loss function becomes very small as it propagates back through the network layers, making it difficult to train deeper networks with many hidden layers (Bengio et al., 1994).

ADAM (Adaptive Moment Estimation) (Kingma and Ba, 2014), a variant of gradient descent, is a popular optimization algorithm for training ANNs. ADAM addresses the vanishing gradient problem by using a technique called adaptive learning rates, which adjusts the learning rate for each weight parameter based on the historical gradients of that parameter. This helps prevent the learning rate from becoming too small and allows the algorithm to converge faster and more reliably. ADAM also includes a momentum term, which helps the algorithm overcome local minima in the optimization landscape.

ADAM has multiple hyperparameters besides the η, namely β_1, β_2, and ϵ. β_1 determines how much of the past gradients' first moment (mean) should be used when computing the current gradient estimate. A higher value of β_1 means that more weight is given to the past gradients, which can help smooth out the updates and make them more stable. β_2 determines how much of the past gradients' second moment (variance) should be used when computing the current gradient estimate. A higher value of β_2 means that more weight is given to the past squared gradients, which can help adapt the learning rate for each weight based on how the variance of the gradient is changing. ϵ is a small constant added to the denominator of the adaptive learning rate to avoid division by zero.

2.2 Deep Learning and Convolutional Neural Networks

Deep learning is a subfield of ML that uses ANNs with many layers (i.e., ”deep”). In contrast to traditional ML models that often utilize only a few layers, DL models exhibit a much deeper architecture, allowing them to learn intricate patterns and representations from complex datasets (Zhou, 2021). This makes DL particularly well-suited for various domains with complex data, such as computer vision (Voulodimos et al., 2018), natural language processing, and speech recognition (Kamath et al., 2019). In particular, deep learning has revolutionized the field of computer vision, enabling machines to see and interpret the world around them more accurately than before (Voulodimos et al., 2018). This surge in popularity can be attributed to the ever-growing computational power of computers and the abundance of accessible data.

Convolutional neural networks are a type of deep neural network that is particularly well-suited for tasks involving image and video analysis. CNNs were introduced by (Forsyth et al., 1999). However, it was not until the mid-2010s that they gained widespread popularity due to their success in computer vision tasks such as image classification, object detection, and segmentation (Russakovsky et al., 2015).

Like other deep neural networks, CNNs consist of multiple layers of interconnected nodes, but their architecture is specifically designed to take advantage of the spatial structure of images. CNNs consist of several layers that perform specific functions in the network. In addition to the input and output layers found in traditional ANNs, CNNs have a set of hidden layers that includes convolutional layers, pooling layers, and fully connected layers. A simplified version of a CNN architecture is shown in Figure 2.4.

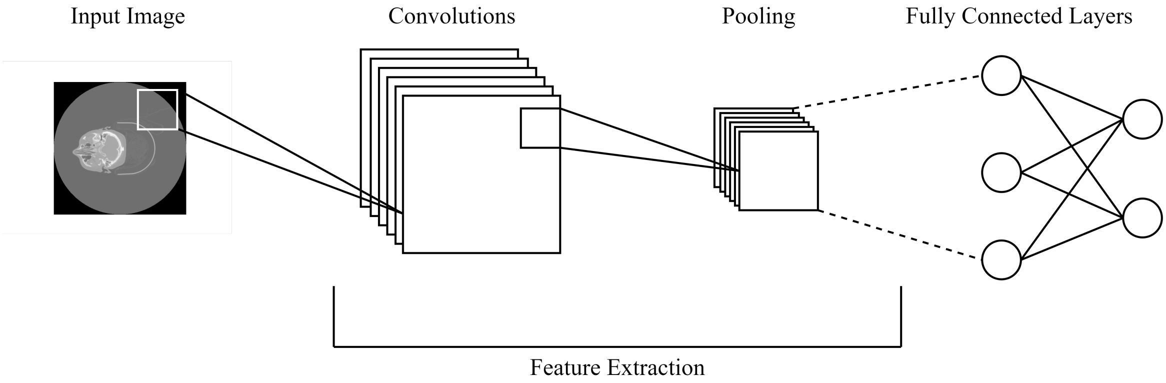


Figure 2.4: Simplified CNN Architecture

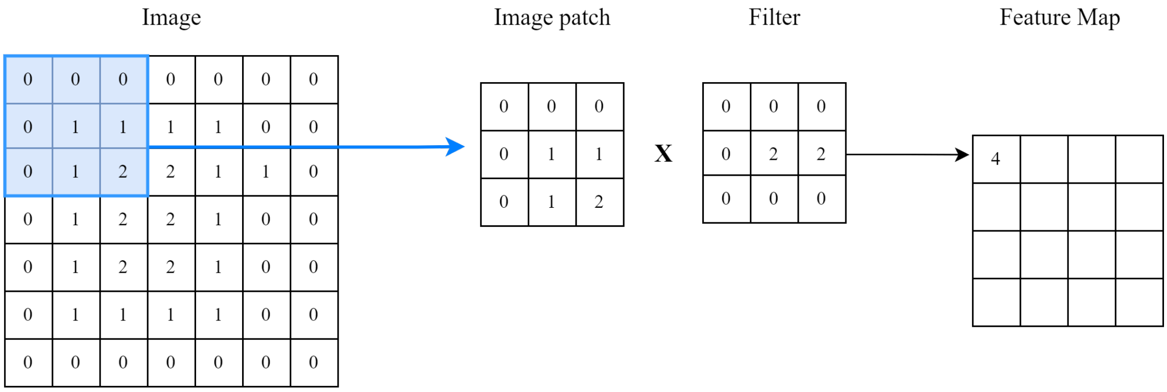
Convolutional layers are the backbone of CNNs. They are designed to perform convolutions on the input data, which involves applying filters to small regions of the input image. A convolution operation is matrix-multiplying the input image patch with the learned filter and using the sum as the output.

The mathematical notation for a convolutional layer is expressed with Equation 1.

$$Y = f(x \times w + b) \tag{1}$$

The output of the convolutional layer is denoted by Y. This output represents the result of convolving the input x with the set of learnable filters w and adding the bias term b. The activation function f is then applied to this result to produce the final output of the layer.

The filter is iteratively convolved with the image, one image patch at a time, until the entire image has been processed. The first iteration in a simple 2-dimensional convolution operation is shown in Figure 2.5.



0	0	0	0	0	0	0
0	1	1	1	1	0	0
0	1	2	2	1	1	0
0	1	2	2	1	0	0
0	1	2	2	1	0	0
0	1	1	1	1	0	0
0	0	0	0	0	0	0

0	0	0
0	1	1
0	1	2

0	0	0
0	2	2
0	0	0

4			

Figure 2.5: Convolution Operation

Each filter produces a feature map highlighting certain features in the input image, such as edges or corners. Convolutional layers learn these features automatically during the training process without the need for explicit feature engineering.

Pooling layers reduce the spatial dimensionality of the feature maps produced by the convolutional layers. They do this by extracting a single value by taking a small region of the feature map and applying a function f. The pooling operation reduces the number of parameters in the network and helps prevent overfitting.

Fully connected layers are used to produce the final output of the network. They combine the high-level features extracted by the convolutional and pooling layers into a prediction or classification. Each neuron in a fully connected layer is connected to every neuron in the previous layer, allowing for complex nonlinear relationships to be learned.

CNNs are commonly used for image processing tasks where the input data is a two-dimensional image. In a 2D CNN, the filters are typically 2D vectors, and the convolution operation is performed along the width and height dimensions of the input image. On the other hand, 3D CNNs are designed to work with 3D data, such as volumetric data. In a 3D CNN, the filters are 3D vectors, and the convolution operation is performed along the width, height, and depth dimensions of the image (Jnawali et al., 2018).

One of the main advantages of 3D convolutions is their ability to capture spatial features from volumetric data. By convolving a filter over the input data in all three dimensions, a 3D CNN can learn to detect patterns and features that evolve over space, such as texture changes or shape variations. However, 3D convolutions require more computational resources and memory than 2D convolutions due to the increased number of parameters and the additional dimension of the input data.

2.3 Cancer Segmentation with Deep Learning

2.3.1 Deep Learning in Medicine

The application of DL in medicine has vastly improved the healthcare industry by improving disease detection and diagnosis (Wang et al., 2019). The main contribution is algorithms that detect cancer from medical images, such as MRI, CT, and PET. Known as Computer-Assisted Diagnosis (CAD), ML and DL algorithms can analyze medical images to identify potentially cancerous cells and help doctors make accurate diagnoses (Giger and Suzuki, 2008), making ML an increasingly popular approach in medical imaging in the field of medicine.

CNNs have been used to analyze medical images, classifying cancer such as breast cancer (Benhammou et al., 2018), cervical cancer (Kavitha et al., 2023), lung cancer (Alakwaa et al., 2017) and head and neck cancer (Halicek et al., 2017).

DL in medicine has both benefits and limitations. One of the main benefits is its ability to analyze large amounts of data quickly and accurately. This enables the identification of patterns and relationships in the data that may not be apparent to human experts. ML algorithms can also learn from experience, enabling them to improve over time with additional data. However, there are also limitations to the use of DL in medicine. One major limitation is the data quality used to train the algorithms. DL algorithms require high-quality, reliable data to learn effectively, and if the data is biased or inaccurate, this can lead to incorrect predictions and diagnoses. The accuracy of DL algorithms depends heavily on the quality of the data, especially for DL algorithms, making the availability of quality data a crucial factor in the success of DL applications in medicine.

2.3.2 Cancer Segmentation

Cancer detection and segmentation are critical to diagnosing cancer from medical images. Cancer detection involves identifying the presence of cancer in medical images, while cancer segmentation involves dividing medical images of tumors into distinct regions or segments based on their characteristics. The ultimate goal of cancer segmentation is to accurately identify the boundaries of the tumor and its various regions, which can help healthcare professionals diagnose the disease.

Figure 2.6 shows an example 2D slice from a 3D CT image with the corresponding tumor mask separately and overlayed on the image. In a segmentation DL model, the image is the input x, and the mask is the output $\hat{Y}$.

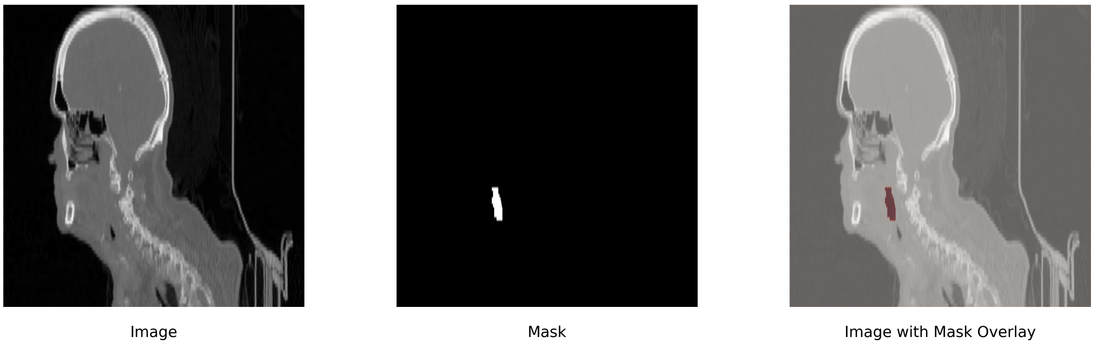


Figure 2.6: Example lice of CT Image with Mask

With the advancements in DL and medical imaging technologies such as MRI, PET, and CT, it is now possible to detect and segment cancer from medical images with high accuracy and efficiency (Rezaei, 2021). Algorithms that segment cancer use images and corresponding tumor masks as training data.

A common neural network architecture for image segmentation is the U-Net architecture, introduced by (Ronneberger et al., 2015). The U-Net is particularly suited for image segmentation, due to its effectiveness and simplicity. The U-Net consists of a contracting path and an expanding path. An example U-Net architecture is shown in Figure 2.7.

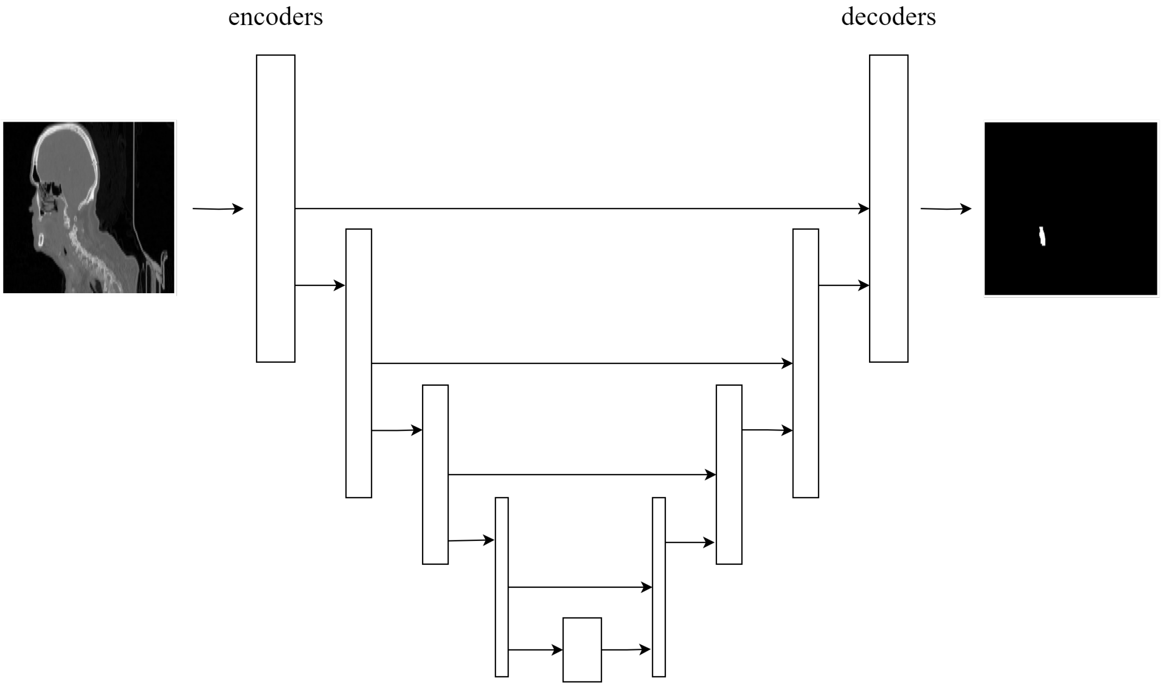


Figure 2.7: Simplified U-Net Architecture

The contracting path is a series of convolutional and pooling layers that are used to extract features from the input image. These layers reduce the spatial resolution of the features, and are called encoders. The expanding path is a series of convolutional and upsampling layers that are used to decode the features back into the original input image resolution. The components of the expanding path are called decoders.

In addition to the encoders and decoders, the U-Net also includes skip connections between corresponding layers in the contracting and expanding paths. These connections allow the network to preserve high-resolution details from the input image, even as it is downsampled and then upsampled again.

2.3.3 Evaluating Segmentation Models

The Dice Score Coefficient (DSC) (Bertels et al., 2019) is a common metric used to evaluate the performance of image segmentation algorithms. It measures the similarity between the predicted segmentation and the ground truth segmentation of an image and is expressed with Equation 2.

$$DSC = 2 * (TP)/(2 * TP + FP + FN) \tag{2}$$

where TP is the number of true positive voxels (voxels that are correctly classified as part of the tumor), FP is the number of false positive pixels (voxels that are incorrectly classified as part of the tumor), and FN is the number of false negative voxels (voxels that are incorrectly classified as background).

The DSC ranges from 0 to 1, with 1 indicating perfect overlap between the predicted and ground truth segmentation and 0 indicating no overlap. A higher DSC indicates better segmentation performance.

2.3.4 Current State of Cancer Segmentation with Deep Learning

(Kao and Yang, 2022) conducted a systematic review investigating the utility of DL-based segmentation with PET and CT scans. They gathered eight articles and presented the first systematic review of deep learning-based segmentation of lung tumors in PET and CT scans. The most commonly used methods were 3-dimensional CNNs and the U-net architecture. Additionally, multimodal segmentation strategies were employed, with six articles using individual extractors for CT and PET before integrating the features for final predictions. The study concludes that DL-based segmentation of lung tumors in PET/CT scans can achieve good performance, with potential for future clinical applications.

The HECKTOR 2021 challenge (Oreiller et al., 2022) focused on developing automatic segmentation methods for cancerous cells in the Head and Neck (H&N) region using PET/CT images. A total of 18 teams participated in the challenge. Participants in the challenge primarily utilized U-Net-based DL models with a 3D architecture. Factors that contributed to the success of these segmentation methods included effective preprocessing techniques, normalization, data augmentation, and the ensembling of multiple models. The positive results from this challenge suggest that automatic segmentation methods can take advantage of the combined properties of PET and CT images to significantly improve studies in H&N cancer. The winning method in the challenge was proposed by (Iantsen et al., 2021) with a model designed with the U-Net architecture. The method significantly outperforms the proposed baseline method for the challenge. The results from this challenge demonstrated that the performance of automatic segmentation methods could surpass those of manual annotations in radiomics studies, which has promising implications for future research.

Building upon the success of the previous editions, the HECKTOR 2022 challenge evolved by doubling the dataset size from the previous year and including metastatic lymph nodes in addition to the tumors. Eighteen teams participated in the challenge. The winning team, (Myronenko et al., 2023), used a SegResNet-based (Siddique et al., 2022) approach with the Auto3DSeg framework from the MONAI platform (Cardoso et al., 2022).

The best method obtained an average DSC of 0.7591 in the HECKTOR 2021 challenge. The winning team in the HECKTOR 2022 challenge, (Myronenko et al., 2023), achieved the highest DSC with their approach, reaching a DSC of 0.78802.

2.4 Generative Models for Synthetic Data Generation

Synthetic data is artificially generated data that mimics real-world data but is not derived from real-world observations. Synthetic data can be used as a substitute for real-world data in ML and DL applications, which can be beneficial when real-world data is challenging to obtain. The use of synthetic data in ML in healthcare applications has become increasingly popular in recent years, as it can provide an alternative to using real-world data (Chen et al., 2021).

The limited nature of high-quality biomedical training data creates the need for methods to increase the number of training samples without acquiring new time-consuming scans of real people. Data augmentation (DA) is useful by transforming original data and creating new data by applying transformations such as geometric or intensity methods (Shorten and Khoshgoftaar, 2019). Cropping, rotating, translating, and flipping are common image data augmentation techniques used to increase the diversity of a training dataset. The intuition behind DA is that more information can be extracted from the original dataset (Shorten and Khoshgoftaar, 2019). However, the transformed images offer more training data but do not necessarily introduce novel features as they are still based on the original data (Frid-Adar et al., 2018).

An alternative to DA is generative ML models, a family of machine learning models that can learn the underlying distribution of a given dataset and generate new data samples that follow a similar distribution. In the field of DL, GANs, Variational Autoencoders (VAEs) (Kingma and Welling, 2013), and Diffusion models (Croitoru et al., 2023) are popular generative models in recent years.

GANs are DL models that consist of a generator and a discriminator. The generator generates synthetic samples, and the discriminator tries to differentiate between synthetic and real samples. The two models are trained together, with the generator learning to produce samples similar to the real data and the discriminator learning to distinguish between real and fake samples.

VAEs are models that learn a compressed representation of the input data, also known as the latent space. This latent space can be used to generate new samples by sampling from a distribution that is learned during training. However, some challenges are associated with using VAEs for generating synthetic medical images. One of the key challenges is ensuring that the generated images are accurate and realistic representations of the underlying medical conditions. This requires careful tuning of the VAE architecture and training parameters and thorough validation and testing of the generated images, which is only feasible in some scenarios.

Diffusion models are generative models that generate new samples from a given dataset by applying a series of noise-reduction steps to the original data. These steps gradually refine the noise to create a new sample that follows the same distribution as the original data. Diffusion models may not be feasible for medical data in scenarios where there is not a lot of data available, as these models typically require a significant amount of high-quality data to generate realistic samples.

While VAEs and Diffusion models have advantages, they may not always be the best choice for generating synthetic medical data compared to GANs. VAEs can sometimes produce blurry images and may not capture the complex features of the input data. At the same time, Diffusion models can be computationally expensive and require large amounts of data. In contrast, GANs are known for generating high-quality and diverse samples that capture the underlying data distribution, making them an appropriate choice for generating synthetic medical data with a limited dataset.

In the next chapter, we will focus on the technical details of GANs and explore their advantages and limitations.

2.5 Generative Adversarial Networks

GANs are a type of DL algorithm used for generating new data similar to a given dataset. GANs learn to generate new samples by training two neural networks in a game-like setting and have two main components: a generator network and a discriminator network. The generator network generates fake data similar to the input data. The generator is trained to generate data that is as realistic as possible to fool the discriminator network into thinking that the generated data is real. Figure 2.8 illustrates the overall design and training of a conventional GAN model.

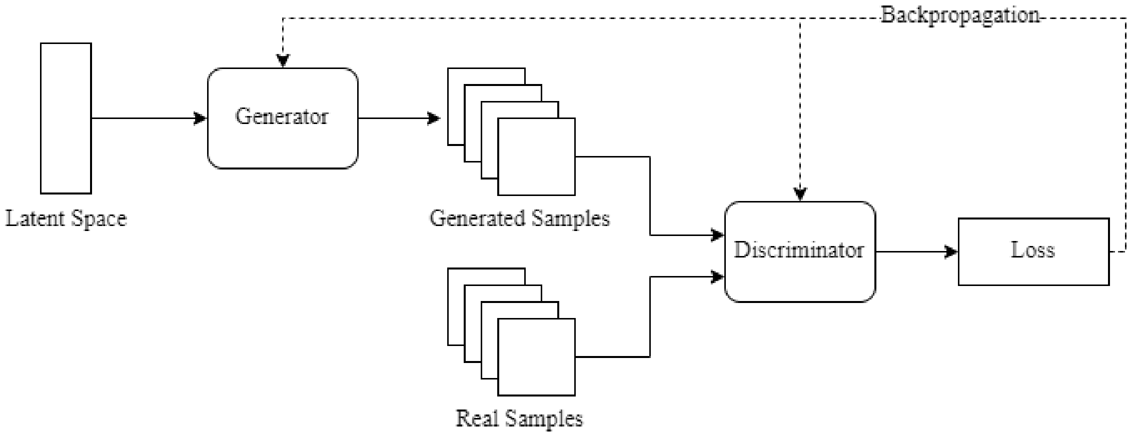


Figure 2.8: Simplified GAN Architecture

The generator and discriminator networks are trained adversarially, which is how the method gets its name. The generator is trained to generate data that is as realistic as possible, while the discriminator is trained to correctly identify which data is real and which is fake. The generator and discriminator are trained iteratively, with the generator trying to improve its ability to generate realistic data and the discriminator trying to improve its ability to distinguish between real and fake data. Known as MinMax loss, the loss function in a traditional GAN is expressed in Equation 3.

$$\min_{G} \max_{D} \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_{z}(z)}[1 - \log D(G(z))] \tag{3}$$

Where G denotes the generator, and the discriminator is denoted by D. The first term in the equation represents how well the discriminator can identify real data x drawn from the true data distribution $p_{\text{data}}(x)$. The second term in the equation represents how well the discriminator can distinguish between synthetic data generated by the generator and real data. The generator tries to minimize this term by producing synthetic data that the discriminator classifies as real.

In medical imaging, GANs can generate synthetic images that can help augment or supplement real-world medical imaging data. For example, GANs can be trained in medical images to learn the patterns and features characteristic of certain conditions or diseases. The synthetic images can be used to train other ML algorithms and validate the accuracy of CAD systems.

Overall, GAN is a powerful and versatile DL architecture that can generate new data similar to a given dataset. The adversarial training process allows the generator to learn from the discriminator's feedback and generate increasingly realistic data, making GANs an effective tool for generating synthetic medical images.

2.5.1 Limitations

Traditional GANs have several limitations that can make them challenging to train and result in poor image quality. One of the main limitations is the instability of the training process. GANs use a min-max game between the generator and the discriminator. As a result, the training process can become unstable if the generator produces poor-quality images and the discriminator gets too strong. This can lead to problems such as mode collapse (Durall et al., 2020), where the generator gets stuck in sub-optimal parameter space and produces only a limited number of distinct images. The issue of vanishing gradients (Li et al., 2018) is another problem for GANs, just like conventional ANNs. This occurs when the gradients become too small, impeding the generator's learning process and producing bad images.

During the training process of a GAN, the goal is to achieve an equilibrium between the generator and the discriminator, i.e., a balance between the two. The idea is to get the generator to produce realistic images that can fool the discriminator while the discriminator can accurately distinguish between the generated and real images. However, achieving this equilibrium can be challenging, especially when dealing with complex image datasets like 3D medical data.

One of the main challenges in achieving this equilibrium is setting the appropriate strengths of the generator and the discriminator. If the discriminator is too weak, it can fail to identify generated images as fake, allowing the generator to produce poor-quality images that are not realistic. On the other hand, if the discriminator is too strong, it can quickly identify generated images as fake, making it difficult for the generator to produce high-quality images. In addition, traditional GANs may also suffer from overfitting (Yazici et al., 2020), where the generator learns to produce images similar to the training set but lacking diversity. In simpler terms, the generator memorizes the training data, which results in poor generalization performance, i.e., severely limiting the generation of new, realistic images.

Finally, traditional GANs may struggle with generating high-resolution images. As the image size increases, the number of parameters in the generator and discriminator also increases, making the training process more difficult and time-consuming. The larger number of parameters means that more data is required to train the networks effectively. This can be challenging to obtain, especially for medical datasets, where data is often limited or expensive to acquire.

The vanishing gradient problem occurs when the gradients in the network become extremely small during backpropagation, which makes it difficult for the network to update its weights. In the case of GANs generating high-resolution images, the increase in image size results in a correspondingly larger number of parameters in the generator and discriminator networks. During training, the generator network receives gradients from the discriminator network, which helps it update its weights to generate more realistic images. However, as the number of parameters in the network increases, the gradients may become smaller, leading to the vanishing gradient problem. This can be particularly problematic for GANs generating high-resolution images since they require many layers to capture the complexity of the image.

Traditional GANs have several limitations that can make them challenging to train and result in poor image quality. These limitations have led to the development of new GAN architectures and training techniques that aim to overcome these issues and improve the stability and quality of GANs. The following chapters will focus on improved GAN architectures, loss functions, and evaluation metrics. We will discuss how these factors can impact the performance of GANs and their ability to generate high-quality synthetic medical data.

2.6 Beyond GAN: Improved Architectures

Alternative architectures have been proposed to overcome the limitations of traditional GANs, with the aim of addressing training stability issues and improving the quality of generated images. This chapter will introduce a selection of improved GAN architectures related to the thesis.

2.6.1 Wasserstein GAN

Wasserstein GANs (WGAN), introduced by (Arjovsky et al., 2017), are a type of GAN that aims to improve the training process stability and generate higher-quality images. The main differences are how the training is performed, the loss function, and the network optimizer.

In traditional GANs, the generator produces generated samples, and the discriminator distinguishes between real and generated samples. The loss function used in GANs can be challenging to optimize and may lead to instability in the training process. WGANs, on the other hand, use the Wasserstein distance (Vallender, 1974) to measure the difference between the real and generated samples. This distance is more effective than the traditional measures used in GANs, making learning more manageable for the generator and avoiding issues such as mode collapse and vanishing gradients in traditional GANs (Arjovsky et al., 2017).

Wasserstein loss is a modification of the original GAN MinMax loss function (Goodfellow et al., 2020), in which the discriminator does not classify instances. Instead, the discriminator output is a probability representing whether the input image is real or fake. In this loss scheme, the discriminator is called a ”critic,” as it does not discriminate between real and fake images. The critic loss is expressed in Equation 4.

$$CriticLoss : D(x) - D(G(z)) \tag{4}$$

The discriminator tries to maximize this function, meaning it tries to maximize the difference between its output on real images and its output on fake images. The generator loss is expressed in Equation 5.

$$GeneratorLoss : D(G(z)) \tag{5}$$

The generator tries to maximize this function, meaning it tries to maximize the discriminator’s output for the fake images provided by the generator.

One of the critical advantages of WGANs is that they provide a stable training process and can generate high-quality images. Additionally, they allow for better control over the quality of the generated images, as the value of the Wasserstein distance can be used to measure the quality of the generated samples. Another difference is that the discriminator network in WGAN is modified to satisfy the Lipschitz constraint (Hager, 1979), which limits the slope of the discriminator function and helps to ensure the stability of the training process.

To enforce the Lipschitz constraint, WGANs use a technique called weight clipping (WGAN-WC), setting a maximum and minimum value for the weights of the discriminator network. Any weight that exceeds this maximum value is clipped to this value during training. Similarly, if any weight falls below the minimum value, it is clipped to the minimum value. By enforcing the Lipschitz constraint through weight clipping, WGANs encourage the discriminator to have a smoother decision boundary, improving the training process's overall stability. However, it is worth noting that weight clipping can also introduce some drawbacks, such as causing the model to converge to a suboptimal solution or leading to poor sample quality.

An alternative version of WGAN with weight clipping is WGAN with gradient penalty (WGAN-GP), introduced by (Gulrajani et al., 2017). WGAN-GP also tries to enforce the Lipschitz constraint with a gradient penalty, which is computed by randomly sampling points on the straight lines connecting real and generated samples. If the gradient deviates from a predefined value, the discriminator is penalized. This penalty term promotes smooth gradients in the input space for the discriminator, preventing it from assigning excessively high gradients to specific points that could disrupt the training process. WGAN-GP addresses the limitations of weight clipping and provides a more stable and efficient way to enforce the Lipschitz constraint on the discriminator.

2.6.2 StyleGAN2 and FastGAN

StyleGAN2, introduced by (Karras et al., 2020), is an enhanced version of the original StyleGAN (Karras et al., 2019), specifically designed to generate high-quality, diverse, and customizable 2-dimensional images with precise control over various aspects of image synthesis.

A significant breakthrough in StyleGAN2 lies in its progressive, growing architecture, which allows the generation of high-resolution images without compromising quality or stability. Initially, the model generates low-resolution images and gradually incorporates additional layers and features to produce progressively higher-resolution images. This progressive approach enables the model to acquire more intricate data representations over time, resulting in superior image quality.

StyleGAN2 also incorporates an adaptive discriminator augmentation technique, a crucial feature that enhances the model's stability and robustness during training. This technique introduces random data augmentations to the discriminator network throughout training. By doing so, overfitting is mitigated, and the model's ability to generalize is improved.

FastGAN is a GAN model introduced by (Liu et al., 2021). The model is designed to generate high-quality 2D images with improved training efficiency and scalability compared to traditional GAN models.

One of the critical innovations of FastGAN is its use of a new generator architecture called Skip-GAN, which enables the model to generate high-quality images with fewer training iterations. The Skip-GAN architecture uses skip connections between different layers of the generator network, which allows the model to learn more efficiently and generate higher-quality images with fewer training iterations.

Another important feature of FastGAN is its use of a new training algorithm called decoupled training, which enables the model to train the generator and discriminator networks separately. This approach allows each network to learn quickly, resulting in faster and more efficient training.

FastGAN also introduces a novel regularization technique that helps stabilize the training process and prevent overfitting. The method involves applying spectral normalization to each group of neurons in the network, which limits the magnitude of the weights and prevents instability during training.

2.6.3 Hierarchical Amortized GAN

(Sun et al., 2022) proposes a novel approach for generating high-resolution 3-dimensional medical images using a Hierarchical Amortized GAN (HA-GAN).

Most current 3D GAN models are trained on low-resolution medical images due to limited GPU memory, which can lead to patchy artifacts or an inability to scale to high-resolution images. HA-GAN is a memory-efficient method that generates sub-volumes of the high-resolution input image during training while retaining anatomical consistency. They achieve this through a hierarchical structure during training, which produces a low-resolution image and a randomly selected sub-volume of the high-resolution image.

The input to the network is a real high-resolution image X^H. There are two generators in HA-GAN, the high-resolution generator G^H and a low-resolution generator G^L, which generate a low-resolution full-size image $\hat{X}^L$ and a high-resolution sub-volume $\hat{X}^H$. The high-resolution sub-volume is selected randomly from different parts of the image, allowing for overlapping sub-volumes to better cover junctions between the parts. This memory-efficient approach uses a simplified image to help the model learn.

The HA-GAN architecture has two discriminators. The high-resolution discriminator D^H discriminates high-resolution sub-volumes, while the low-resolution discriminator D^L discriminates low-resolution full volumes. The high-resolution discriminator ensures that local details in the sub-volume look realistic, while the low-resolution discriminator preserves the proper global structure. The model's memory cost is reduced by feeding a sub-volume instead of the entire image to the high-resolution discriminator. The location of the sub-volume is also fed into the discriminator to help it distinguish sub-volumes from different locations. Because the model is generating both low and high-resolution volumes, there are two GAN losses in HA-GAN, $\mathcal{L}_{GAN}^L$ and $\mathcal{L}_{GAN}^H$. $\mathcal{L}_{GAN}^L$ is the low-resolution loss function, expressed in Equation 6.

$$\mathcal{L}_{GAN}^L(G^L, G^A, D^L) = \min_{G^L, G^A} \max_{D^L} \mathop{\mathbb{E}}_{X \sim P_X}[\log D^L(X^L)] + \mathop{\mathbb{E}}_{Z \sim P_Z}[\log(1 - D^L(\hat{X}^L)] \tag{6}$$

Where G^L, G^A, and D^L are the low-resolution generator, generator common block, and the low-resolution discriminator, respectively. The goal is to minimize the value of $\mathcal{L}^L_{GAN}$. The equation consists of two terms. The first term, $\underset{X\sim P_X}{\mathbb{E}}[\log D^L(X^L)]$, is the expected value of the logarithm of the output of the discriminator when given a real low-resolution volume X^L from the data distribution P_X. In other words, it's the average probability that the discriminator assigns to real input data being real.

The second term, $\underset{Z\sim P_Z}{\mathbb{E}}[\log(1 - D^L(\hat{X}^L)]$, is the expected value of the logarithm of discriminator output when fed a generated low-resolution full volume $\hat{X}^L$. The volume $\hat{X}^L$ is generated by the low-resolution generator G^L from the noise vector Z. In other words, it's the average probability that the discriminator assigns to fake data being fake.

$\mathcal{L}^H_{GAN}$ is the high-resolution loss function, expressed in Equation 7, which is used to train the high-resolution generator G^H, the generator common block G^A, and the high-resolution discriminator D^H.

$$\mathcal{L}^H_{GAN}(G^A, G^H, D^H) = \min_{G^H, G^A} \max_{D^H} \underset{r\sim U}{\mathbb{E}} \left[\underset{X\sim P_X}{\mathbb{E}}[\log D^H(S^H(X^H; r), r)] + \underset{Z\sim P_Z}{\mathbb{E}}[\log(1 - D^H(\hat{X}^H_r, r)] \right] \tag{7}$$

The equation consists of two main terms inside the expectation denoted by $\underset{r\sim U}{\mathbb{E}}$, where r is randomly selected from a uniform distribution U and represents the location of the high-resolution sub-volume (in the full volume).

The first term, $\underset{X\sim P_X}{\mathbb{E}}[\log D^H(S^H(X^H; r), r)]$, is the expected value of the logarithm of the output of the discriminator when it's given a real high-resolution sub-volume X^H extracted from the real data volume X at location r. The function $S^H(X^H; r)$ is the extraction of the sub-volume from location r. In simpler terms, it's the average probability that the discriminator gives to real sub-volumes being real.

The second term, $\underset{Z\sim P_Z}{\mathbb{E}}[\log(1 - D^H(\hat{X}^H r, r)]$, is the expected value of the logarithm of the discriminator output when it's given a generated high-resolution sub-volume $\hat{X}^H r$. This sub-volume is generated by the high-resolution generator G^H from the noise vector Z at location r. It is the average probability that the discriminator gives to fake sub-volumes being fake.

The HA-GAN model also has two encoders, which are trained to encode the low-resolution full-volume and the high-resolution sub-volume. The generated volumes are then compared with the real volumes, and reconstruction loss is calculated. The high-resolution encoder loss is expressed in Equation 8.

$$\mathcal{L}^{H}_{recon}(E^{H}) = \min_{E^{H}} \mathop{\mathbb{E}}_{X\sim P_X, r\in U} [\![S^{H}(X^{H};r) - G^{H}(\hat{A}_{r})]\!]_{1} \tag{8}$$

The equation consists of a single term, $\mathop{\mathbb{E}}_{X\sim P_X, r\in U} [\![S^{H}(X^{H};r) - G^{H}(\hat{A}r)]\!]1$, which is the expected value of the absolute difference between the real high-resolution sub-volume $S^{H}(X^{H};r)$ and the generated high-resolution sub-volume $G^{H}(\hat{A}_{r})$. The average difference between real sub-volumes of high-resolution data and the generated sub-volumes is calculated, and used to train the encoder.

The total reconstruction loss function, $\mathcal{L}^{G}_{recon}(E^{G})$ is used to train the shared encoder E^{G}. It is the sum of the low-resolution and high-resolution reconstruction losses. The goal is to make both the low-resolution image and the high-resolution sub-volume generated by the generators as similar as possible to the corresponding real ones. The equation is expressed in Equation 9.

$$\mathcal{L}^{G}_{recon}(E^{G}) = \min_{E^{G}} \mathop{\mathbb{E}}_{X\sim P_X} \left[[\![X^{L} - G^{L}(G^{A}(\hat{Z}))]\!]_{1} + \mathop{\mathbb{E}}_{r\sim U} \left[[\![S^{H}(X^{H};r) - G^{H}(S^{L}(G^{A}(\hat{Z});r))]\!]_{1} \right] \right] \tag{9}$$

The equation has two terms. The first term, $\mathop{\mathbb{E}}_{X\sim P_X} \left[[\![X^{L} - G^{L}(G^{A}(\hat{Z}))]\!]_{1} \right]$ is the expected value of the difference between the real low-resolution volume X^{L} and the generated low-resolution volume $G^{L}(G^{A}(\hat{Z}))$. X^{L} is a low-resolution volume and $G^{A}(\hat{Z})$ is the output of the generator common block G^{A} when fed with the latent representation $\hat{Z}$, which is then used to generate the low-resolution volume $G^{L}(G^{A}(\hat{Z}))$.

The second term, $\mathop{\mathbb{E}}_{r\sim U} \left[[\![S^{H}(X^{H};r) - G^{H}(S^{L}(G^{A}(\hat{Z});r))]\!]_{1} \right]$ is the expected value of the difference between the real high-resolution sub-volume $S^{H}(X^{H};r)$ and the generated high-resolution sub-volume $G^{H}(S^{L}(G^{A}(\hat{Z});r))$. The low-resolution sub-volume is used to generate the high-resolution sub-volume $G^H(S^L(G^A(\hat{Z}); r))$. Simply explained, the measurement of the average difference between the real low and high-resolution volumes and the generated volumes are used to optimize the shared encoder E^G.

The overall loss function for HA-GAN is shown in Equation 10.

$$\mathcal{L} = \mathcal{L}^H_{GAN}(G^H, G^A, D^H) + \mathcal{L}^L_{GAN}(G^L, G^A, D^L) + \lambda_1 \mathcal{L}^H_{recon}(E^H) + \lambda_2 \mathcal{L}^G_{recon}(E^G), \quad (10)$$

The overall architecture is shown in Figure 2.9, with some parts excluded for visibility.

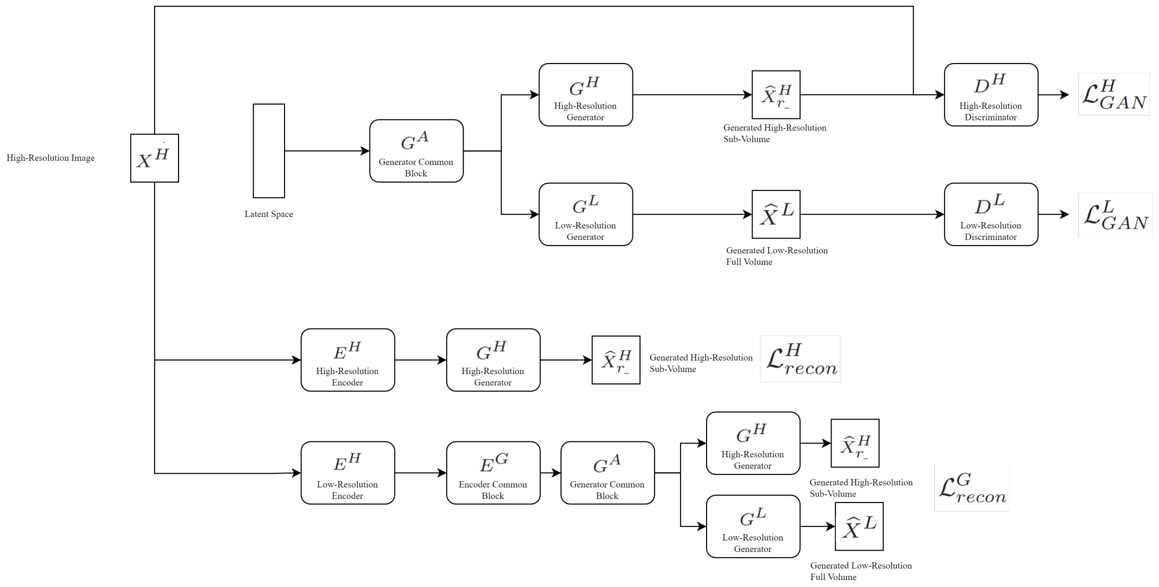

High-Resolution Image
X^H
Latent Space
G^A Generator Common Block
G^H High-Resolution Generator
$\hat{X}^H_{r\_}$
Generated High-Resolution Sub-Volume
D^H High-Resolution Discriminator
$\mathcal{L}^H_{GAN}$
G^L Low-Resolution Generator
$\hat{X}^L$
Generated Low-Resolution Full Volume
D^L Low-Resolution Discriminator
$\mathcal{L}^L_{GAN}$
E^H High-Resolution Encoder
G^H High-Resolution Generator
$\hat{X}^H_{r\_}$
Generated High-Resolution Sub-Volume
$\mathcal{L}^H_{recon}$
E^H Low-Resolution Encoder
E^G Encoder Common Block
G^A Generator Common Block
G^H High-Resolution Generator
$\hat{X}^H_{r\_}$
Generated High-Resolution Sub-Volume
G^L Low-Resolution Generator
$\hat{X}^L$
Generated Low-Resolution Full Volume
$\mathcal{L}^G_{recon}$

Figure 2.9: HA-GAN Architecture

There are two variations of the HA-GAN architecture. It has the ability to take as input and generate images with dimensions of 128×128 and $256 \times 256 \times 256$.

2.7 Evaluating Generative Adversarial Networks

Evaluation metrics are used to measure the performance and effectiveness of machine learning models, including GANs. These metrics measure model performance, which is essential for assessing the quality of the model and making improvements. The goal of a GAN is to generate synthetic data similar to real-world data, and evaluation metrics provide a way to measure the similarity between the generated and real data.

2.7.1 Inception Score

Introduced by (Salimans et al., 2016), the Inception Score (IS) is an evaluation metric for GANs that measures the quality and diversity of generated images. The generated images are fed to a pre-trained Inception model (Szegedy et al., 2015), a popular CNN image classification network.

The Inception model calculates the class probabilities for each generated image. IS is calculated by comparing the distribution of the class probabilities of all the generated images with the distribution of class probabilities for each individually generated image. The comparison uses a mathematical formula called the Kullback-Leibler (KL) divergence (Joyce, 2011). The final IS value is obtained by taking the exponential of the expected value of this comparison.

A higher IS score indicates that the generated images have a good variety of features that are representative of the training dataset. In contrast, a lower score indicates that the generated images are less diverse or of lower quality.

2.7.2 Fréchet Inception Distance

The Fréchet Inception Distance (FID) was first introduced by (Heusel et al., 2017) and compares the distribution of generated images with the distribution of a set of real images. In other words, FID measures how similar the generated images are to the real images by comparing them.

First, a pre-trained image classifier, like the Inception model, is used to define a feature space. Then, the FID measures the distance between the multivariate

Gaussian distributions of the real and generated images in that feature space. This distance is calculated by finding the squared Euclidean distance between the means of the two distributions, adding the sum of their variances, and then subtracting twice the square root of their product.

A lower FID indicates that the generated images are more similar to the real images in their statistics and are of higher quality.

2.7.3 t-Distributed Stochastic Neighbor Embedding

Introduced by (Van der Maaten and Hinton, 2008), t-Distributed Stochastic Neighbor Embedding (t-SNE) is not typically used as an evaluation metric for GANs but rather as a visualization tool to explore the structure and diversity of data. By applying t-SNE to GAN-generated images, it is possible to visualize the distribution of the generated data in a lower-dimensional space and gain insight into the diversity and structure of the generated data. t-SNE can help identify patterns or clusters in the generated data and highlight areas where the GAN may produce similar or redundant images. Most importantly, t-SNE can be applied to both the original training data and compared with the generated data. A pre-trained image classifier is used to extract feature representations for each real and generated image, like in the IS and FID calculations. Finally, t-SNE analysis is applied to both image set activations to visualize and compare the images in a lower-dimensional space.

t-SNE has several hyperparameters that can be tuned to affect its performance and the quality of the resulting visualization. The hyperparameter Components define the number of dimensions in which the data will be embedded. A common choice is two, allowing for easy visualization of the data in a scatter plot.

Perplexity is a measure of how well the algorithm balances attention between local and global aspects of the data. Higher values of perplexity result in t-SNE focusing more on the global structure, while lower values make it focus more on local neighborhoods.

The learning rate controls the step size during the optimization process. If the learning rate is too high, the optimization process may overshoot the optimal solution, whereas if it is too low, it may take a long time to converge.

2.7.4 Visual Turing Test

A Visual Turing Test (VTT) (Chuquicusma et al., 2018) involves presenting an expert human evaluator with pairs of images, where one image is real, and the other is generated by the GAN. The evaluator must then determine which image is real and which is generated. The process is repeated for the number of images in the experiment. The VTT for GANs is a challenging task as it requires the GAN to generate images that are not only visually similar to real images but also capture the subtle details and nuances that make them appear real.

2.8 Hyperparameter Optimization

Hyperparameters are parameters set before training a model and cannot be learned during training. Hyperparameter optimization (HPO) finds the best set of hyperparameters for a machine learning model that yields optimal performance (Yang and Shami, 2020). HPO aims to find the best combination of hyperparameters that maximizes the model performance. This is important because the choice of hyperparameters can significantly impact the model's performance.

Several HPO methods are available, but manual search is the most widely used technique. This approach involves selecting hyperparameters based on personal experience and judgment and then making arbitrary adjustments after evaluating the model.

Another commonly used HPO approach is random search, where hyperparameters are randomly selected from a given parameter space. Random search does not sample evenly across the entire search space but explores it randomly. As a result, a random search may need to run for a considerable period to find the optimal set of hyperparameters (Liashchynskyi and Liashchynskyi, 2019).

Another HPO technique is grid search, where a predefined set of hyperparameters is specified, and the model is trained and evaluated for every combination of these hyperparameters. This approach systematically explores the entire search space, ensuring that every combination of hyperparameters is tested. However, grid search can be computationally expensive, especially for high-dimensional search spaces with many hyperparameters (Liashchynskyi and Liashchynskyi, 2019).

HPO is an important aspect of machine learning model development and can significantly improve the performance of the model. It is commonly used in DL models, including GANs, to find the best hyperparameters. By selecting the optimal hyperparameters, HPO can help to reduce overfitting, improve training speed and stability, and ultimately lead to better performance.

2.9 Related Work

This chapter reviews existing research on synthetic medical data generation with GANs and 3D image segmentation. It covers relevant studies in synthetic 2D and 3D medical data generation using GAN and explores current techniques for the segmentation of 3D images. The chapter concludes with a summary of key findings and gaps in the current research, highlighting its significance in setting the stage for the thesis contribution.

2.9.1 Synthetic 2D Medical Data Generation with GANs

The use of GANs to generate synthetic medical data is an active area of research. (Sorin et al., 2020) performed a systematic review of image generation for radiology applications using GANs. Their study analyzed 33 publications, primarily emphasizing various modalities, including MRI and CT. Interestingly, out of the total publications, only six studies focused on generating synthetic images. Most concentrate on generating synthetic CT images, while the remainder focus on MRI. The authors in the review note that almost all of the six DA publications observed an increase in the performance of ML algorithms trained on synthetically generated data. However, they note that generated images sometimes struggle to compete with real images, seeing as they may be blurred or have low resolution.

The authors note that a solution to the problem is possible by training the algorithms on fake images and then further fine-tuning the algorithms by training them on real images.

None of the six studies that aimed to create synthetic CT and MRI images were trained directly with 3D images. Instead, they used single slices of 3-dimensional images for training purposes. Most studies utilized smaller image sizes, such as 64×64, while one study utilized the largest size of 256×256.

(Russ et al., 2019) explores synthetic training data generated using GAN to improve the potential of medical image analysis with neural networks. The authors evaluate eleven GAN architectures for synthesizing CT images and assess image quality regarding anatomical accuracy and realistic noise properties. The best-performing network configuration is identified, and three networks are trained using the ideal configuration in combination with an extended training dataset and a task-based loss function. The authors demonstrate the applicability of this simulation-driven approach in a proof-of-principle, showing that a task-based network trained on a combination of real and synthetic data can achieve superior performance in blood vessel segmentation compared to a network trained on real data alone.

(Hussain et al., 2022) proposes a solution to the problem of data deficiency in COVID-19 radiograph images using a Wasserstein GAN. The authors show that using a WGAN can lead to an effective and lightweight solution for generating synthetic images at par with the original images.

(Yang et al., 2021) addresses the issue of obtaining both MRI and CT images for clinical applications, which is often costly and sometimes unavailable, particularly for special populations. The paper presents a novel generative network that combines the advantages of VAE and GAN. The network is applied to synthesize multi-contrast MRI images from single CT images, and experiments are conducted on brain datasets. The study's main contributions include using random-extraction patches for data augmentation, solving image blurriness and mode collapse problems, and demonstrating the effectiveness and stability of the proposed network. Compared to typical methods, the network yields more accurate and higher-quality synthetic MR images for visual inspection and quantitative assessment. However, the paper notes that the deep network is somewhat sophisticated, with many system parameters that result in high computational complexity.

(Woodland et al., 2022) evaluates the use of the StyleGAN2 architecture, namely StyleGAN2-ADA, for generating high-resolution medical images, addressing the limitations of GANs in medical imaging, such as computational cost, data requirements, evaluation measures, and training complexity. The authors trained a StyleGAN2 network on a liver CT dataset and four publicly available datasets comprising various imaging modalities. The authors conclude that StyleGAN2-ADA consistently produces high-quality medical images and achieves state-of-the-art FIDs on all datasets. They find that the generated images are of sufficient quality that an expert's ability to tell whether or not an image was generated approaches random guessing. Furthermore, they report that the ”realness” score based on a 5-point Likert scale differs between the generated and real images by less than the standard deviation between clinicians.

(Thambawita et al., 2022) presents a novel synthetic data generation pipeline called SinGAN-Seg, which produces synthetic medical images. What differentiates this approach is the generation of synthetic images with corresponding tumor masks. The pipeline can be used to produce alternative artificial segmentation datasets with corresponding ground truth masks. The authors used image quality and FID scores to compare several GAN architectures: DCGAN, Progressive GAN, FastGAN, and SinGAN-Seg. Based on the results presented, it is evident that FastGAN and SinGAN-Seg versions produce superior synthetic images in comparison to DCGAN and Progressive GAN. Additionally, FastGAN performs better regarding FID scores than SinGAN-Seg without style transfer. Notably, when 1,000 images were used for training, SinGAN-Seg with style transfer outperformed all other GANs.

To demonstrate the effectiveness of the pipeline, the authors train a segmentation U-Net model using both real data and synthetic data generated from the SinGAN-Seg pipeline. They show that the models trained on synthetic data perform close to those trained on real data when both datasets have considerable training data. In contrast, when the training datasets are small, the synthetic data generated from the SinGAN-Seg pipeline improves the performance of segmentation models. The authors conclude that the SinGAN-Seg pipeline is an effective data augmentation technique that can improve segmentation performance when training datasets are small and can be used as an alternative method to provide and share data when real datasets are restricted.

2.9.2 Synthetic 3D Medical Data Generation with GANs

In the research paper by (Yu et al., 2018), a 3D GAN is introduced, designed for synthesizing MRI images. The evaluation of this method is conducted using the Brain Tumor Image Segmentation Benchmark (BRATS) 2015 dataset, which includes data from 274 subjects. To enhance the brain tumor segmentation using a single T1 (MRI) modality, the study explores the possibility of synthesizing Fluid-Attenuated Inversion Recovery (FLAIR) images from T1. For this purpose, a 3D GAN is designed specifically for FLAIR image synthesis to more accurately represent the details of the synthesized FLAIR images. The method proves to be effective in managing the segmentation of brain tumors that vary considerably in appearance, size, and location across different samples. The generated MRI images from this process have dimensions of $128 \times 128 \times 128$.

(Cirillo et al., 2021) presents a 3D GAN called Vox2Vox for segmentation of brain tumors. The model generates realistic segmentation outputs from multi-channel 3D MR images, segmenting the whole, core, and enhanced tumor with mean dice scores for the BRATS testing set after ensembling 10 Vox2Vox models obtained with a 10-fold cross-validation. The Vox2Vox model is trained and validated on sub-volumes of size $128 \times 128 \times 128$ from 369 and 125 subjects, respectively.

(Bu et al., 2021) propose a novel method based on a conditional generative adversarial network (CGAN) to generate new samples for DA. This method employs a 3D GAN to synthesize realistic and diverse lung nodules in CT images to improve the performance of a CAD system. The method uses a generator with a U-Net architecture and a concurrent squeeze excitation module. The authors used t-SNE and a Visual Turing Test to evaluate the generated images. The authors used synthetic samples for DA to train the lung nodule detection network to evaluate the proposed method. The results indicate that these synthetic samples could boost the overall performance of the nodule detection network. One limitation of the paper is the small image size of the generated images, which are only $32 \times 32 \times 32$.

(Pesaranghader et al., 2021) introduced a novel GAN-based architecture to produce authentic-looking CT images. The authors conducted a quantitative study to demonstrate the effectiveness of DA using GAN for cancer detection. The proposed architecture is capable of synthesizing images of resolution$224 \times 224 \times 224$. By training a cancer detection algorithm on a vast number of synthetic and real data, the classifier performance was improved, proving the usefulness of synthetically generated images.

HA-GAN (Sun et al., 2022) outperforms previous state-of-the-art image generation models such as (Pesaranghader et al., 2021), as demonstrated by 3D CT and brain MRI experiments. The authors compare HA-GAN with other state-of-the-art GAN models and show that HA-GAN produces higher-quality synthetic images than other GAN models, as evidenced by FID and IS scores. The sharpness of the images generated by HA-GAN is also superior to other methods. HA-GAN outperforms baseline models at $128 \times 128 \times 128$ and $256 \times \times 256 \times 256$ resolutions, with greater performance improvements observed at $256 \times 256 \times 256$ resolutions. HA-GAN's ability to directly generate images at $256 \times 256 \times 256$ resolution without upsampling and the model architecture contribute to sharper image generation.

2.9.3 Segmentation of 3D images

In their winning solution to the HECKTOR 2022 challenge, (Myronenko et al., 2023) utilize the Auto3DSeg framework within the MONAI platform. Their approach is based on a SegResNet network that employs a U-Net architecture. The authors use a simple approach to crop the approximate region based on the relative anatomy position within the PET/CT images. They detect the top of the head based on simple thresholding and the H&N center-line based on the average foreground of top slices. The authors voxel resample the input CT and PET images to include the intensity pattern variations within the foreground regions to resolution $1.0 \times 1.0 \times 1.0mm$.

The authors use 5-Fold Cross-Validation to train and select the best model. The ensemble model achieves an average of 0.7989 Cross-Validation DSC, making it the best method for the HECKTOR 2022 challenge.

2.9.4 Related Work Summary

In order to adequately supplement training datasets for cancer segmentation, it is essential to include the medical image and a corresponding ground truth mask that highlights the tumor in the image. This is because cancer segmentation algorithms require both components to learn how to detect and segment tumors in medical images accurately.

Previous research has explored using GANS to generate images and corresponding masks in the 2D image domain. For instance, (Thambawita et al., 2022) proposed a GAN-based framework that can generate synthetic medical images and corresponding masks. However, to the best of our knowledge, no such research has yet been conducted on 3D medical images for high-resolution images.

The lack of research on generating 3D medical images with corresponding masks may be attributed to the computational cost of generating such data. Unlike 2D images, 3D medical images require a much larger storage capacity and computational resources, posing significant challenges for data generation. Consequently, many approaches have focused on generating low-resolution 3D images, which may not be sufficient for accurate tumor segmentation. Thus, there is a need for the generation of 3D medical images and corresponding tumor masks. The generated images and masks should be properly evaluated using state-of-the-art segmentation methods like the Auto3Dseg framework.

3 Methodology

This chapter describes the methodology for generating synthetic CT image/mask pairs using GANs. The methodology includes the dataset used, preliminary data analysis, data preprocessing, GAN architectures and training procedures, data augmentation, data postprocessing, evaluation, and hyperparameter optimization.

3.1 Dataset

This thesis utilizes the HECKTOR grand challenge 2022 dataset (Andrearczyk et al., 2023). The dataset consists of PET and CT images of the head and neck region, collected from nine data sources, of patients with head and neck cancer confirmed by histology. Patient information, including center, age, gender, weight, tobacco and alcohol consumption, performance status, HPV status, and treatment details, is also included in the dataset alongside the images. However, only the images were utilized in this thesis, as the metadata is not used.

The dataset consists of 845 subjects and is split into two separate datasets, one with 524 samples for training machine learning algorithms and the other for testing the trained algorithms. Because the testing dataset only includes the images and not the corresponding masks, it will not be used in this thesis, as both are needed for the generation of new samples.

Each case in the dataset contains a CT scan, PET scan, and a corresponding tumor mask. However, as this thesis only uses CT scans, the PET scans are disregarded. The images and masks are saved in the NIFTI format, containing image data and meta-data.

The training dataset is a combination of PET/CT images originating from seven data sources:

- CHUM
- CHUP
- CHUS

- CHUV
- HGJ
- HMR
- MDA

3.2 Preliminary Data Analysis

To load the 3D NIFTI images, we used Python (Van Rossum and Drake, 2009) and Nibabel (Brett et al., 2023), a Python library for working with neuroimaging data. We load the data from the NIFTI files to retrieve the image data and meta information. The image data is stored as a 3D NumPy (Harris et al., 2020) array, where each element represents the intensity value of a voxel (3D pixel) in the image. The header information contains metadata about the image, such as the dimensions of the image, the voxel size, and the image orientation. The affine matrix is a transformation matrix that maps the voxel coordinates to the physical coordinates in the image.

Once we have loaded the 3D NIFTI images using Nibabel, we can use various NumPy and Matplotlib (Hunter, 2007) functions to manipulate and visualize the data. First, we use Pandas (McKinney et al., 2010) to group the data based on the original data source. We identify the image sizes and how they vary throughout the dataset by obtaining the images' X, Y, and Z sizes.

Most images in the dataset have dimensions 512×512 for the x and y-axis. The average size of the images is $512 \times 512 \times 234$ voxels at $1.04 \times 1.04 \times 2.8mm$ average resolution. Table 3.1 show the number of images and their average size and resolution per data source.

Table 3.1: Dataset average size and resolution, grouped by data source

Source	**Images**	**Average Size**	**Average Resolution**
MDA	198	$510 \times 510 \times 167$	$0.99 \times 0.99 \times 3.26mm$
CHUP	72	$512 \times 512 \times 619$	$0.98 \times 0.98 \times 1.51mm$
CHUS	72	$512 \times 512 \times 151$	$1.17 \times 1.17 \times 2.82mm$
CHUM	56	$512 \times 512 \times 215$	$0.99 \times 0.99 \times 1.97mm$
HGJ	55	$512 \times 512 \times 93$	$0.98 \times 0.98 \times 3.27mm$
CHUV	53	$512 \times 512 \times 276$	$1.25 \times 1.25 \times 3.03mm$
HMR	18	$512 \times 512 \times 129$	$1.06 \times 1.06 \times 3.27mm$

The number of elements on the Z-axis varies depending on the data source within the dataset from where the images originate. In simpler terms, the images have different amounts of ”slices”. One slice is a single 2D image from a 3D image. A 3D image consists of n number of 2D slices, making up the full 3D image. An illustration of what a slice is, is shown in Figure 3.1.

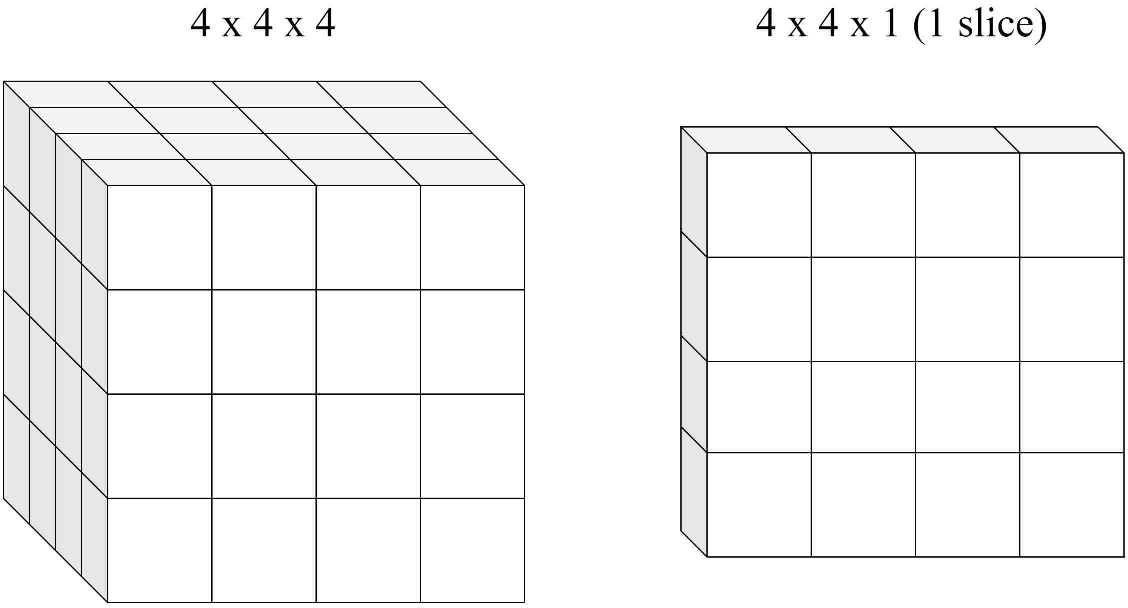


Figure 3.1: 3D Image VS. Single 2D Slice

We create histograms to show the number of slices in each data source. We group the images by data source and visualize the number of slices per source, as shown in Figure 3.2.

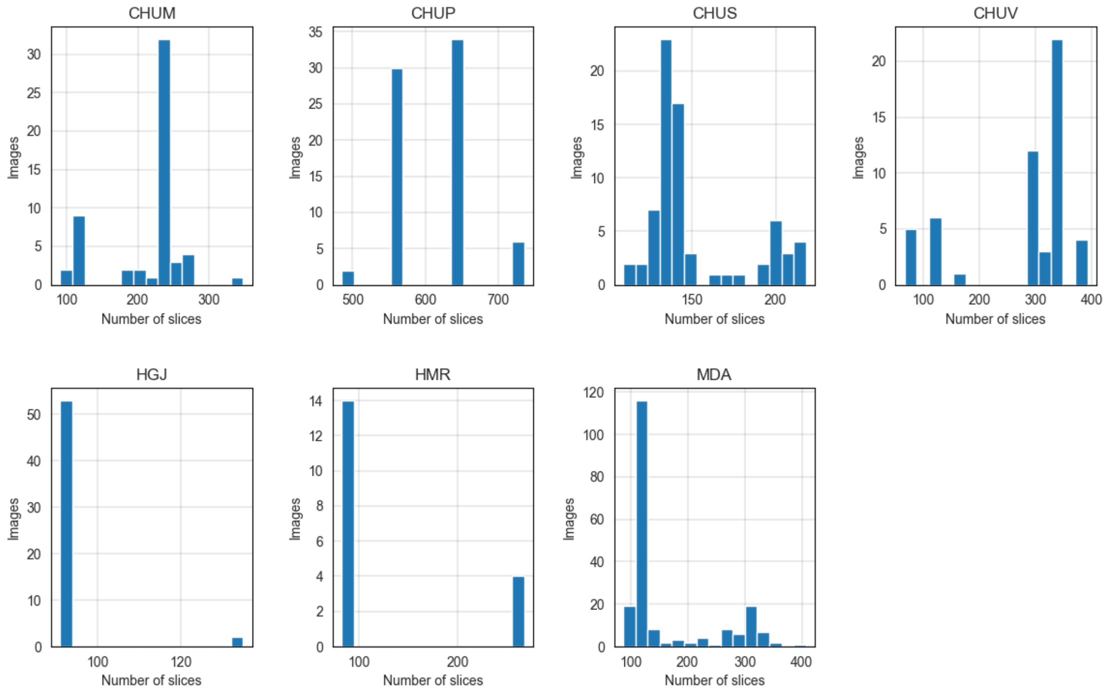


Figure 3.2: Number of Image Slices, Grouped by Data Source

As seen in Figure 3.2, the number of slices has large variations in the dataset, even within each data source. Table 3.2 show the minimum, maximum and average number of slices of the images for each data source in the dataset.

Table 3.2: Number of Slices, Grouped by Data Source

Source	**Minimum**	**Maximum**	**Average**
CHUM	91	348	215
CHUP	485	736	619
CHUS	109	218	150
CHUV	67	392	276
HGJ	91	135	92
HMR	83	267	129
MDA	87	407	166

Hounsfield Units (HU) is a measurement of the X-ray attenuation of tissues in a CT scan and can be used to interpolate a set of CT images before training a GAN to improve performance. In NIFTI CT images, HU can be calculated using the image voxel values and a calibration factor. The NIFTI format stores CT images as a 3D array of integer values, with each value representing the image intensity at a specific voxel. To convert these pixel values to HU, a calibration factor is needed to account for differences in the scanner calibration and X-ray beam energy. To convert the raw data values to HU, a calibration factor (slope) and an offset value (intercept) are usually applied, as expressed in Equation 11.

$$HU = (voxel \times slope) + intercept \tag{11}$$

By iterating over all images in the dataset, we extract the slope and intercept to perform the HU calculation. However, we observe that the slope is set to 1 and the intercept to 0 in all images, meaning that the raw data values have already been converted to Hounsfield Units, and do not require any additional conversion.

Each image in the dataset have a low and high voxel value (HU). We aggregate the lowest and highest values of the minimum values. We do the same calculcation for the maximum values. We create histograms to show the distribution of all minimum and maximum HU across the entire dataset, shown in Figure 3.3.

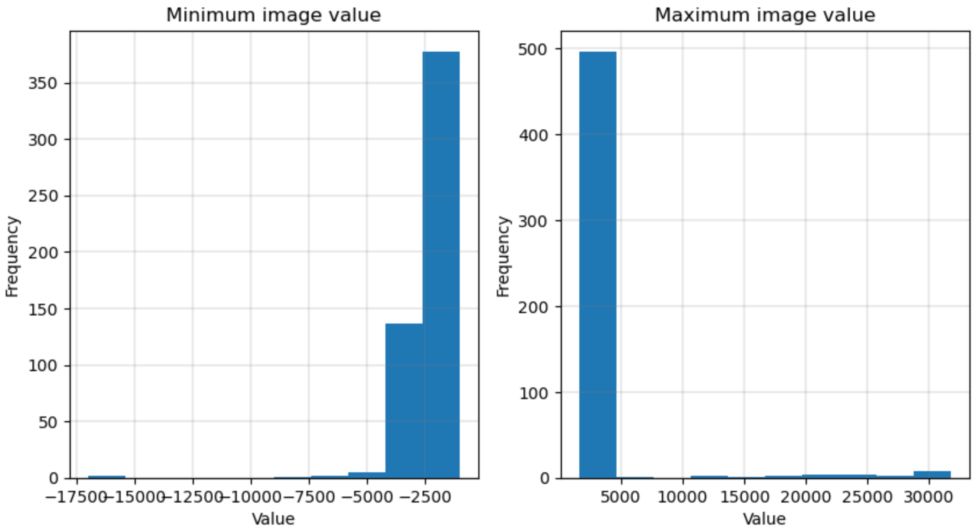


Figure 3.3: HU Ranges

As seen in Figure 3.3, the lowest voxel value across the dataset has the range $[-17500, -2500]$. The highest voxel value is in the range $[3000, 30000]$. The five most frequent minimum values are shown in Table 3.3.

Table 3.3: Most Frequent Minimum Hounsfield Units

Value	**Number of images**
-2048.0	175
-1024.0	152
-3024.0	130
-1000.0	50
-5048.0	1

The five most frequent maximum values are shown in Table 3.4.

Table 3.4: Most Frequent Maximum Hounsfield Units

Value	Number of images
3071	340
2976	114
31743	5
3070	2
1964	1

We use Pandas to group the dataset on the data source. We analyze the HU min-max distribution per source. We compute the minimum and maximum values for the lowest HU observed in each group, as well as the minimum and maximum values for the highest HU. Furthermore, we calculated the average values for both the lowest and highest HU within each group.

Table 3.5 shows the minimum, maximum, and average of the lowest HU per source.

Table 3.5: Minimum, Maximum and Average HU of the Lowest HU Grouped by Data Source

Source	Minimum	Maximum	Average
CHUM	-3024	-1000	-1253
CHUP	-1024	-1024	-1024
CHUS	-1024	-1024	-1024
CHUV	-3024	-3024	-3024
HGJ	-16989	-3024	-3489
HMR	-3024	-3024	-3024
MDA	-16156	-1000	-2206

Table 3.6 shows the minimum, maximum, and average of the highest HU per source.

Table 3.6: Minimum, Maximum and Average HU of the Highest HU, Grouped by Data Source

Source	**Minimum**	**Maximum**	**Average**
CHUM	1870	3071	2938
CHUP	2688	3071	3060
CHUS	2976	2976	2976
CHUV	1695	3071	3019
HGJ	1997	3174	5714
HMR	1738	3071	2865
MDA	1588	31743	5066

As seen in Tables 3.5 and 3.6, the HU minimum, maximum and average values have rather large variations in the dataset.

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a visualization technique to explore the structure and diversity of a given dataset by transforming the data to a low-dimensional space. We group the images by data source and apply t-SNE to the data and create a scatterplot with Matplotlib. Figure 3.4 show the results.

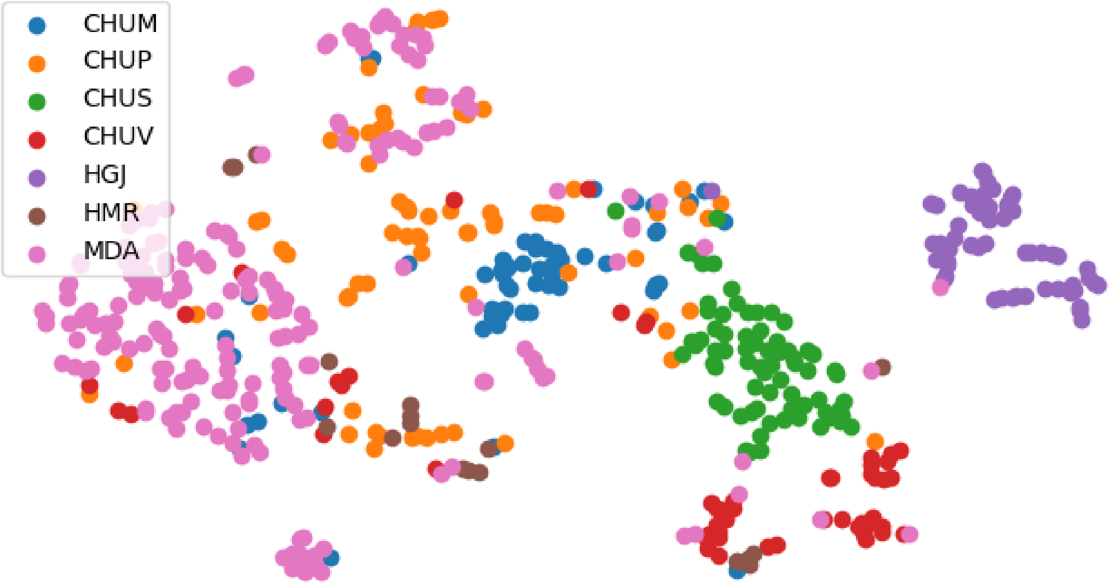


Figure 3.4: t-SNE Scatterplot

Referring to Figure 3.4, it is evident that the majority of the data sources create separate, distinct clusters within the scatterplot. The clusters formed by the CHUS and HGJ data sources stand out as the most unique. The data source labeled as MDA contains the highest number of samples, as depicted in both the figure and Table 3.1. Although this data source predominantly forms a single unique cluster, it also has data samples that diverge and overlap with other clusters, extending as far right on the plot as the HGJ data sources. This overlap suggests that despite their differences, the images within these data sources have certain commonalities. In later chapters, t-SNE will be applied to the real, and GAN generated samples to compare the distributions.

3.3 Data Visualization

Before training a GAN, it is important to carefully examine the data to ensure its quality and understand the anatomy.

MRIcroGL (NITRC, 2022) is a widely used tool for visualizing 3D images and is available as standalone software developed by Neuroimaging Informatics Tools and Resources Clearinghouse (NITR). To visualize NIFTI images using MRIcroGL, we load the image into the software and can visualize the image in a variety of ways. We can then view the image from different angles and zoom in or out to examine the image in more detail. Additionally, we can render the image in 3D to have a more detailed inspection of the image. An example CT image is shown in Figure 3.5.

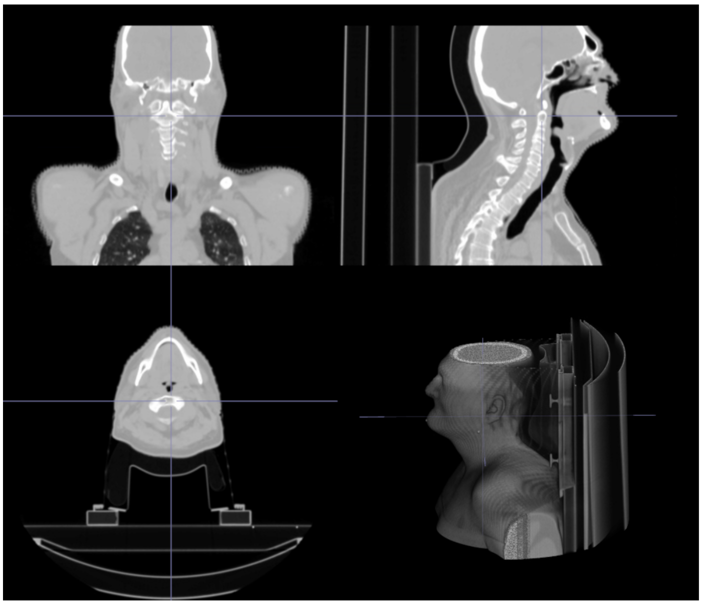


Figure 3.5: Coronal, Sagittal, Axial and 3D Render View in MRIcroGL

Creating a 2D plot with Python and Matplotlib can provide a more detailed and customizable image view. To create a 2D plot, we extract each image slice. Then we iterate over the image along the z-axis and plot each slice on a 2D plane. Figures 3.6 and 3.7 show five slices of two images from different data sources.

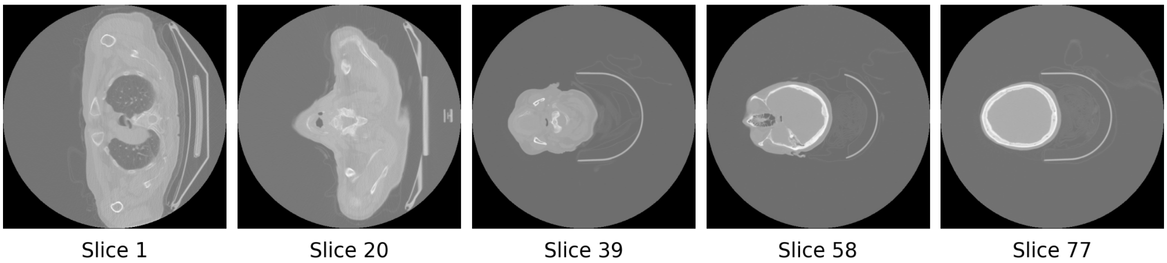


Figure 3.6: Axial Plot CT Image from CHUM

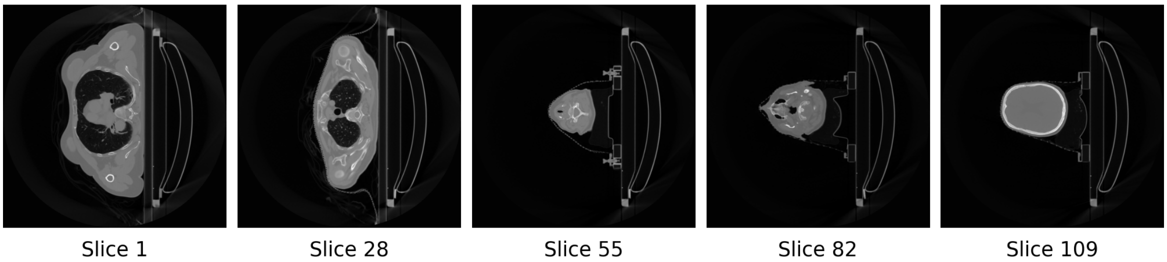
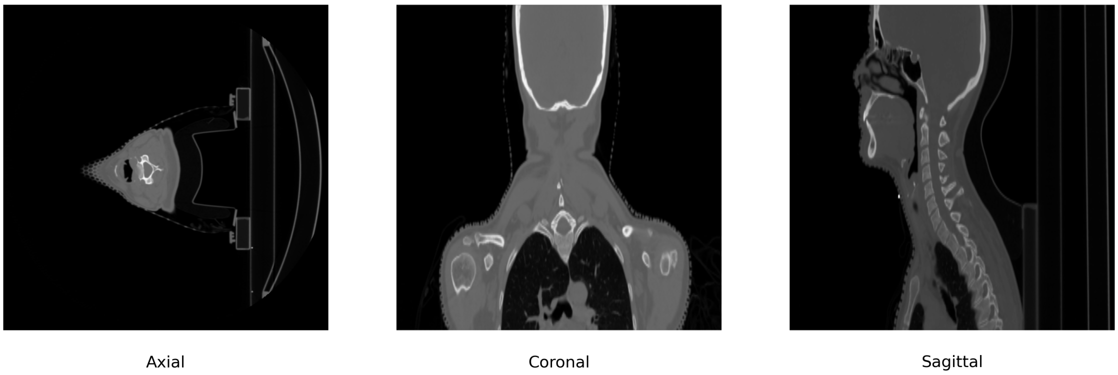


Figure 3.7: Axial Plot CT Image from CHUS

As seen in the figures, the color is different due to the fact that the Hounsfield Units are different, making the intensity of the images appear visually different.

Figure 3.8 show an example image created with Matplotlib of various image orientations.



Axial Coronal Sagittal

Figure 3.8: Axial, Coronal, and Sagittal Slices from Example Image

Figure 3.9 shows a full image from the CHUM center, with resolution $512 \times 512 \times 36$. Because the Z-dimension is 36, this will result in 36 slices, or 36 2D images in the plot.

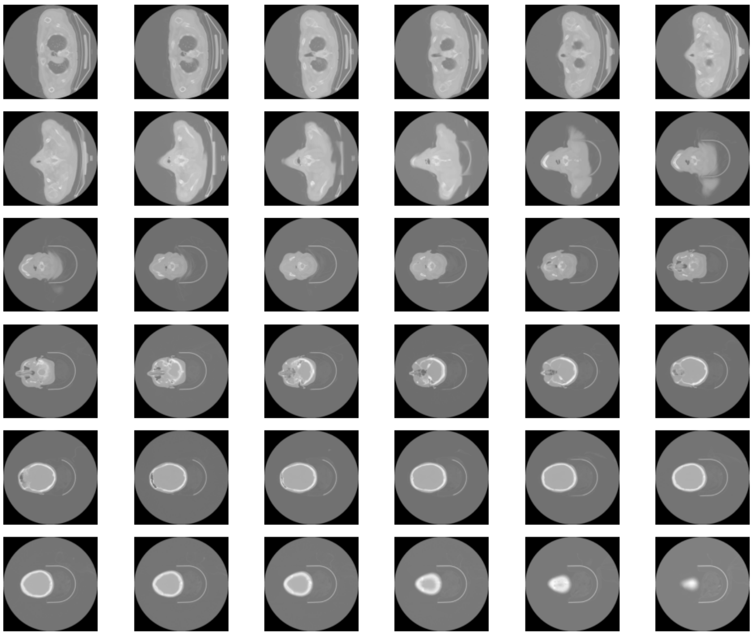


Figure 3.9: $512 \times 512 \times 36$ 2D Image

The corresponding tumor mask is shown in Figure 3.10.

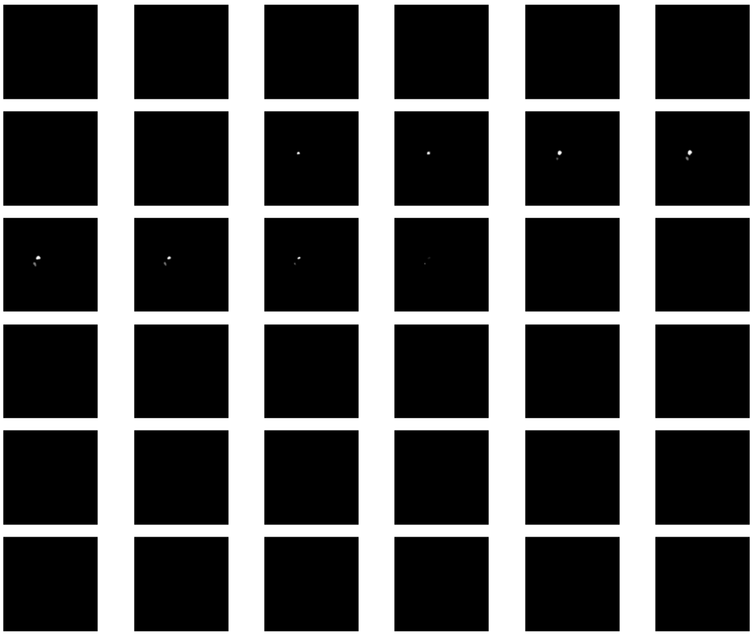


Figure 3.10: Corresponding $512 \times 512 \times 36$ 2D Mask

As seen in Figure 3.10, the slices are empty when there are no voxel values. The slices with values in them represent the tumor.

To visualize sagittal slices, we begin by transposing the data with NumPy. we extract the slices along the sagittal plane using Python and plot the center sagittal slice using Matplotlib. The center sagittal slice of five images from five data sources is shown in Figure 3.11.

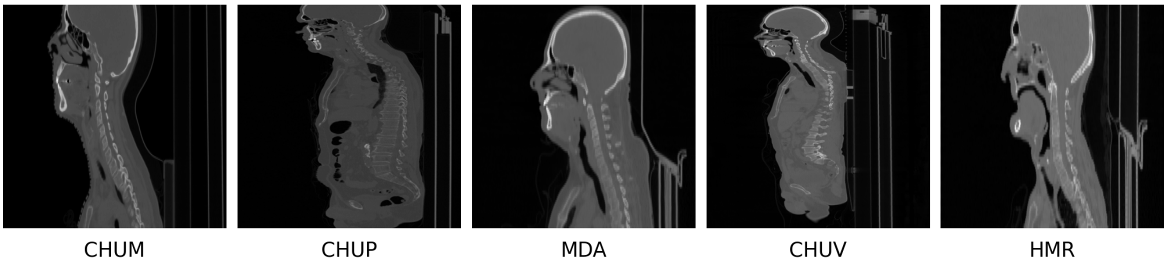
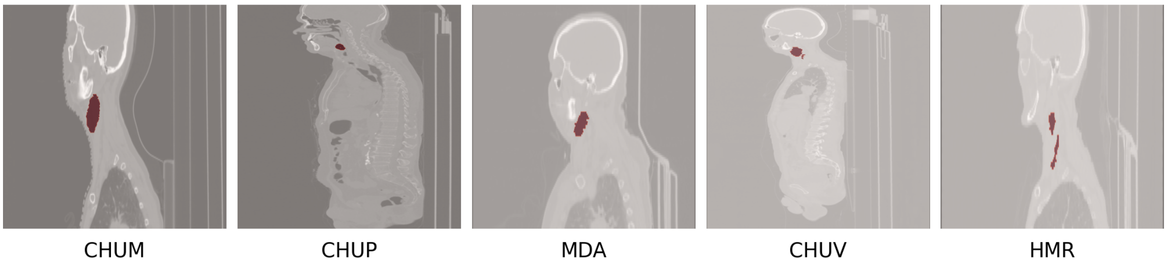


Figure 3.11: Center Sagittal Slices from Five Sources

These techniques showed visually what we observed in the preliminary data analysis (Chapter 3.2). The images from different data sources vary in size. Some data sources have CT images of the whole body, while others have only the head and neck area, as seen in Figure 3.11.

Another approach for visualizing NIfTI images is to overlay the tumor mask on the image along the sagittal center slice. However, it is not always guaranteed that the tumor will be visible on the center slice. Therefore, a calculation can be performed to determine which sagittal slice of the tumor mask is most visible, and the corresponding image and tumor mask can be visualized together. We use Numpy to calculate which mask slice has the most overlaying voxels. We then overlay the tumor mask on the image using matplotlib. We adjust the transparency and color of the tumor mask to highlight the tumor and its boundaries better, as shown in Figure 3.12.



CHUM CHUP MDA CHUV HMR

Figure 3.12: Slices with Mask

3.4 Data Preprocessing and Data Transformation

The preprocessing of data plays a crucial role in the training of GANs. The specific steps used to preprocess the data vary across different GAN architectures and even within the same architecture, as different experiments may employ distinct preprocessing approaches. This subchapter describe the preprocessing steps for our utilized GAN methods, namely Vanilla GAN, WGAN, FastGAN, StyleGAN2 and HA-GAN.

3.4.1 Vanilla GAN and WGAN

Images are resampled to be smaller with a resampling technique that involves changing the resolution and voxel size of the given 3D image. The resampling is done using the Python library SciPy (Virtanen et al., 2020) with the ”zoom” function, which takes an input array image to be resampled and a zoom factor. The zoom factor determines the relative magnification of the output array with respect to the input array. An example of resampling is shown in Figure 3.13.

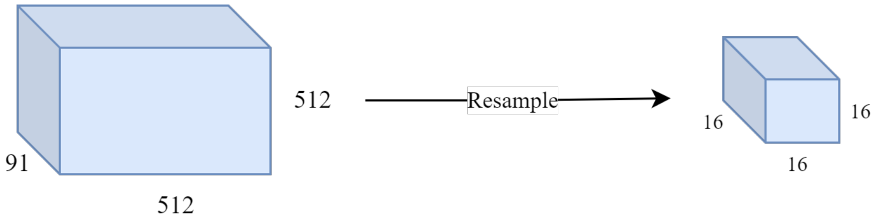


Figure 3.13: Image Resampling

In the case of Naive and Wasserstein GAN, multiple zoom factors, such as 50%, 80%, and 97%, are employed. For instance, a 50% decrease implies that a $512 \times 512 \times 512$ image is resized to $261 \times 261 \times 261$.

Because images from data sources in the dataset have different sizes, we assess all resampled images and locate the largest one. We then use the size of the largest image to augment the size of all other images to match the largest shape. This is done by iterating through all the images and determining the amount of padding necessary for each specified dimension utilizing Python and NumPy. The NumPy ”pad” function is subsequently employed to pad the input image with zeros. The padding may be used for the X, Y, and Z-axis. Figure 3.14 illustrates how a 3D image may be padded along each dimension.

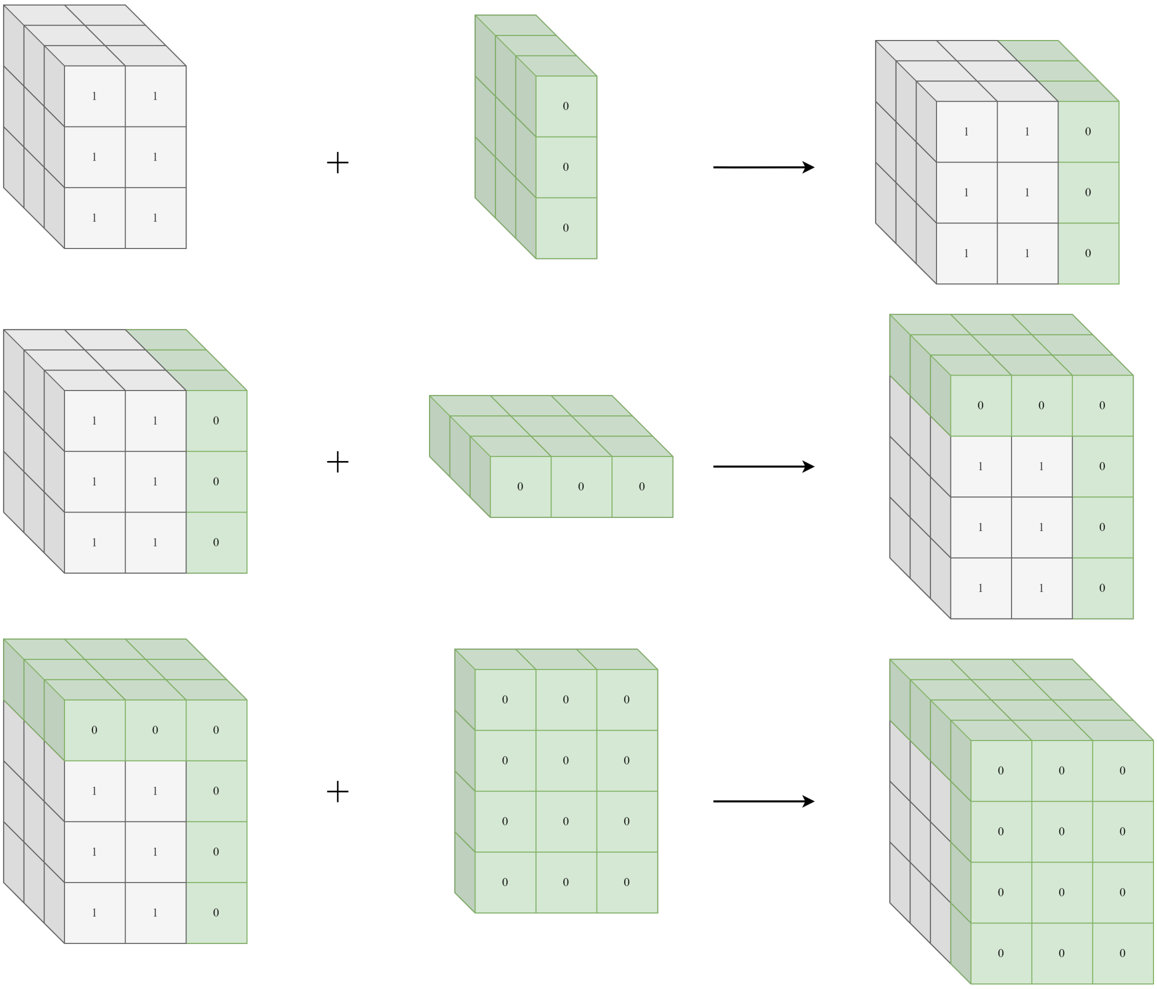


Figure 3.14: Padding a 3D image

Then the image and mask are concatenated together along the Z-axis, combining them into a single image. The concatenation is shown in Figure 3.15.

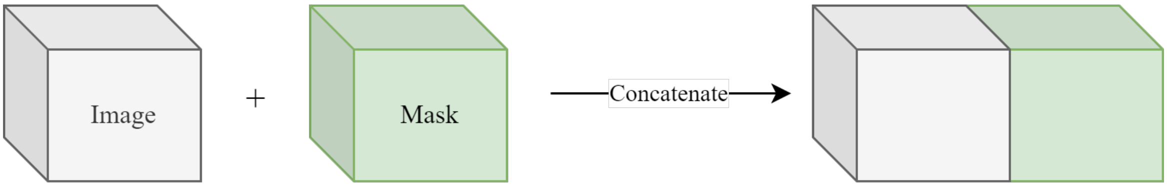


Figure 3.15: Image and Mask Concatenation along Z-axis

Some image/mask pairs have differences in size in various dimensions, making further processing necessary before concatenation. For instance, in some pairs, the image has a size of $512 \times 512 \times 91$, while the corresponding mask has a size of $512 \times 513 \times 91$. We remove the last element along the faulty dimension in these cases before concatenating the pair.

An alternative method of image preprocessing involves resampling both the image and the mask, followed by selecting an arbitrary number of center slices that are subsequently cut out. Finally, the central portions of the image and the mask are concatenated together. The process is shown in Figure 3.16.

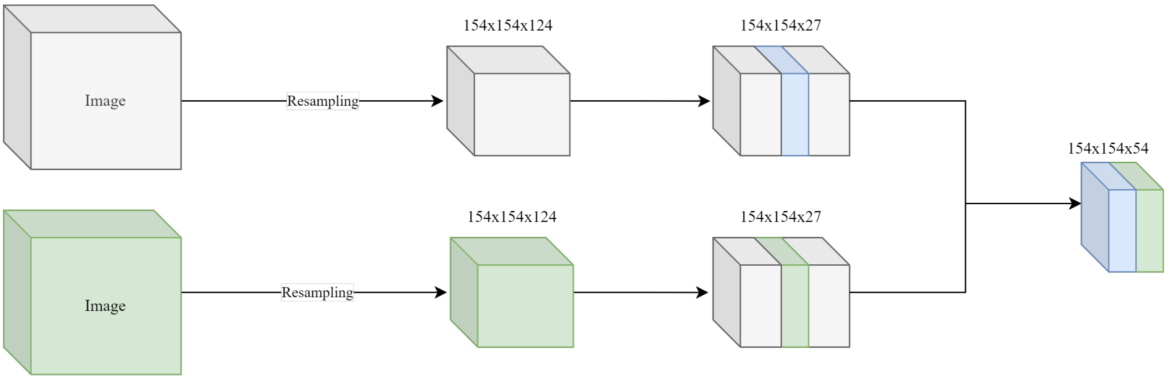


Figure 3.16: Resampling, Chopping, and Concatenating Image and Mask

3.4.2 FastGAN and StyleGAN2

FastGAN and StyleGAN2 are GANs designed to operate on 2D images. Consequently, the 3D images from the dataset need to be converted to 2D. This is accomplished by first resampling the images by 50%. Then, we use Python to iterate over the last dimension of the image and append each image slice onto a 2D plane with the aid of matplotlib, as shown in Figure 3.17.

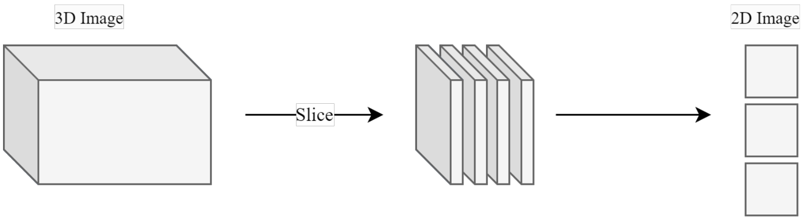
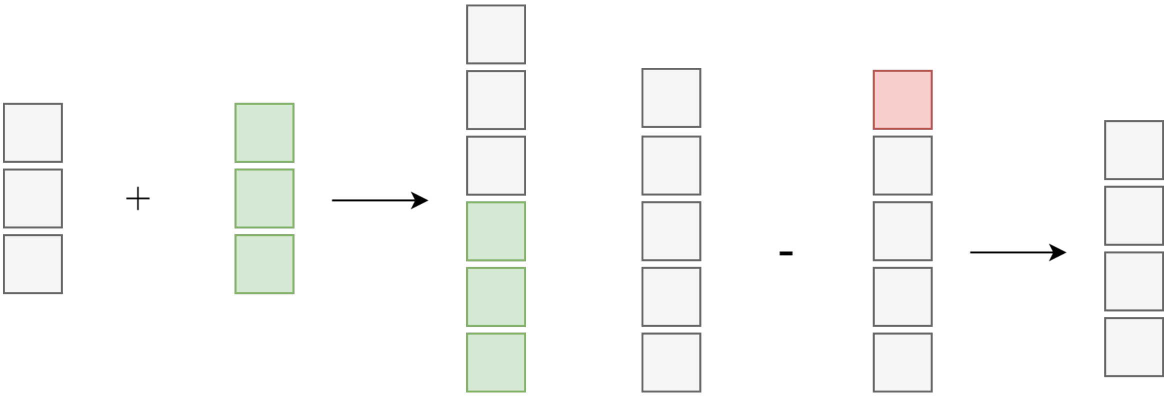


Figure 3.17: Transforming a 3D Image to 2D

Once the 2D planes have been generated, we apply padding or chopping to each image to ensure they are the same size. We evaluate each image to determine if its size surpasses or falls below a pre-defined threshold. Based on the outcome, we either pad (Figure 3.18a or chop (Figure 3.18) the image as necessary.



(a) 2D Padding

(b) 2D Chopping

Figure 3.18: Padding and Chopping 2D Image

Finally, we stack the images to form multiple rows and columns in the final image, which is saved as a PNG image. An example image is shown in Figure 3.19.

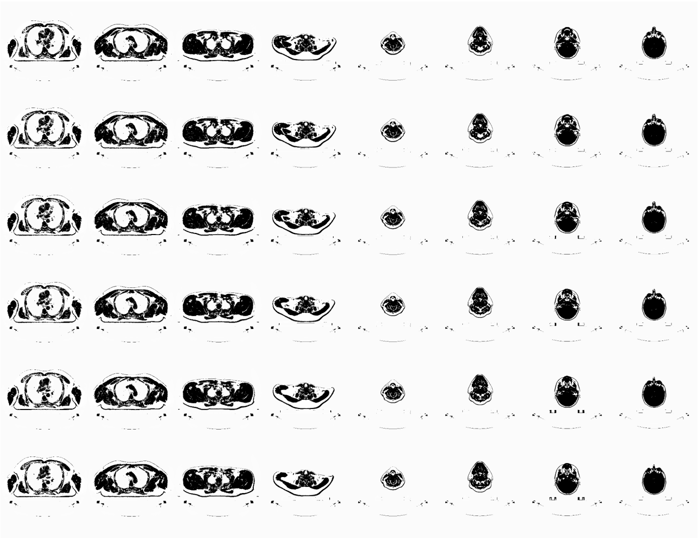


Figure 3.19: Stacked 2D Image

3.4.3 Hierarchical Amortized GAN

We resample the voxels of the images and masks to $1.5 \times 1.5 \times 1.5mm$ isotropic resolution (see Chapter 3.5.1 for more details on voxel resampling). We use the NumPy ”interp” function to interpolate the HU of the images. The interpolation function remaps the voxel values of the input image from the original range to the new range using linear interpolation. Linear interpolation calculates the new value of a point within the target range based on its relative position between the minimum and maximum values of the original range, expressed in Equation 12.

$$y_i = \frac{(x_i - x_{min})(y_{max} - y_{min})}{(x_{max} - x_{min})} + y_{min} \tag{12}$$

Where x_i is the original voxel value in the input image, x_{min} and x_{max} is the minimum and maximum value of the input image, which is set to $[-1024, 600]$. y_{min} and y_{max} are the minimum and maximum values of the target range, which is $[-1, 1]$. y_i is the new voxel value in the output image after interpolation. Figure 3.20 show a visual example of the interpolation.

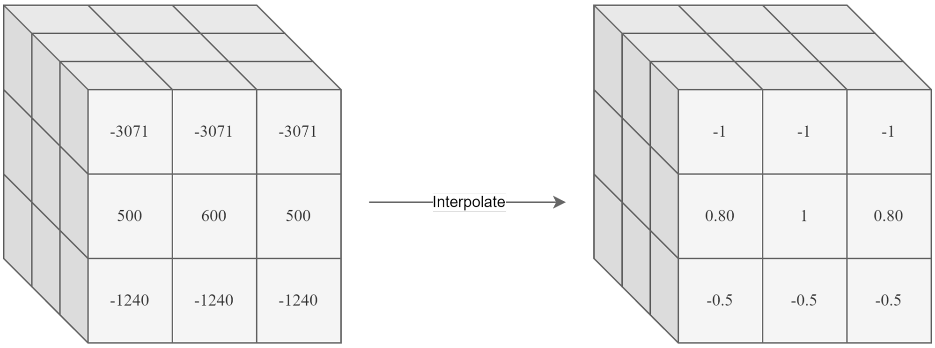


Figure 3.20: Interpolation of 3D Image

Based on the preliminary data analysis, we recognize that the minimum and maximum values are outside the range of $[-1024, 600]$. However, we originally choose to use the range as it is introduced by (Sun et al., 2022). In later experiments, we use different input interpolation ranges.

The mask is originally interpolated with the same input interpolation range as the image. However, in later experiments, we use the range $[0, 2]$. The reason for the different threshold ranges is that the values in the mask can be either 0 for no value, 1 for Tumour, and 2 for Lymph node. Mask interpolation is shown in Figure 3.21.

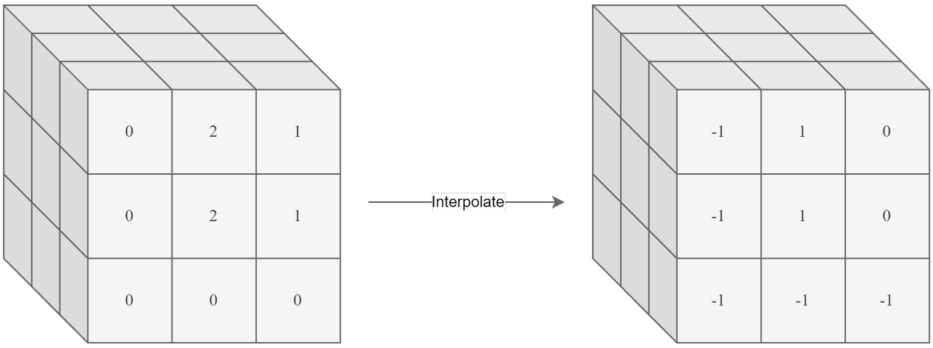


Figure 3.21: Interpolation of 3D Mask

While some images depict the entire body, others only show the head and neck region. The images are sorted according to their data sources to determine the region of interest and carefully examined manually. A boundary is heuristically established for cropping purposes, which differs for each data source. We iterate over all images and their calculated cropping boundary and crop them with their corresponding masks. Figure 3.22 shows example images before and after the cropping operation.

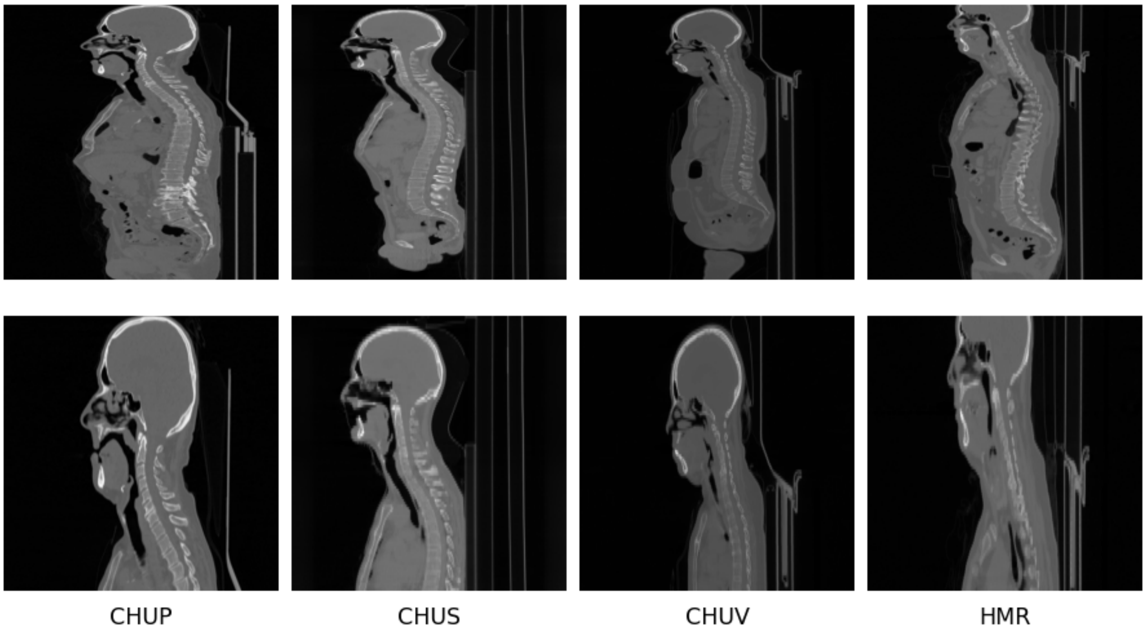


Figure 3.22: Images Before and After Cropping

The cropped images are concatenated for use by HA-GAN in two different ways. The first is the conventional Z-axis concatenation, like in the Vanilla and WGAN shown in Chapter 3.4.1. An alternative approach is slicing the image and mask, and concatenating the pair slice by slice, as shown in Figure 3.23.

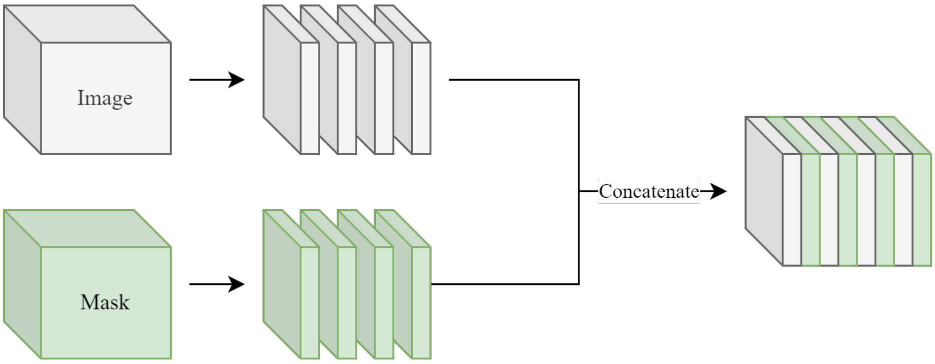


Figure 3.23: Slice-by-slice Concatenation

The Images are then resized to $128 \times 128 \times 128$ or $256 \times 256 \times 256$ with the ”resize” function from Scikit-Image (Van der Walt et al., 2014). The function uses an interpolation-based method to resize the image. It uses a polynomial function to interpolate the voxel values at the new image size. This function is similar to a bicubic interpolation, which results in smooth image rescaling with fewer artifacts than other interpolation methods, such as the zoom function introduced in Chapter 3.4.1. The resized images are then saved as a NumPy file, which is what the HA-GAN network requires for training.

3.5 Data Augmentation

To improve the performance and generalization of the GAN models, we developed a data augmentation framework that applies various transformations to the input images or masks. The framework takes as input either all images or all masks, or both and applies a set of predefined data augmentation techniques to create new variations of the input data.

The data augmentation techniques are defined in a configuration file, specifying each transformation’s parameters. The configuration file includes a set of predefined transformations. The framework iterates over the input data and applies each transformation to create new variations of the data. The augmentation framework is implemented with Python. The image augmentation methods are implemented with the MONAI framework (Cardoso et al., 2022).

3.5.1 Voxel Resampling

Resampling is done to resample the input images to a consistent voxel size. The voxel size (in mm) in a 3D NIFTI image refers to the physical size of each voxel in the three dimensions: x, y, and z. It represents the distance between the centers of adjacent voxels in each dimension. The transformation will resample the input volume to have a given millimeter voxel size, such as $1.5 \times 1.5 \times 1.5mm$.

The resampling function has an interpolation mode, which refers to the method used to estimate the values of voxels in the image at locations that are not part of the original image grid. In other words, interpolation is a process used to estimate the value based on the values of surrounding pixels.

When an image is resampled, the size and spacing of the pixels change, which can result in gaps or overlaps in the image grid. Interpolation is used to fill in these gaps or smooth out the overlaps, resulting in a continuous image. When resampling the voxels, we apply different interpolation modes for the image and mask. The image is interpolated with Bilinear mode, which means voxel values are calculated using linear interpolation between the four closest neighboring voxels. The mask is interpolated with the Nearest Neighbour mode, which means the voxel value is calculated using the closest voxel value.

3.5.2 Affine Transformation

The Affine Transformation randomly applies translation, rotation and scaling along all axes on the image. Figure 3.25 show an example image before and after the transformation.

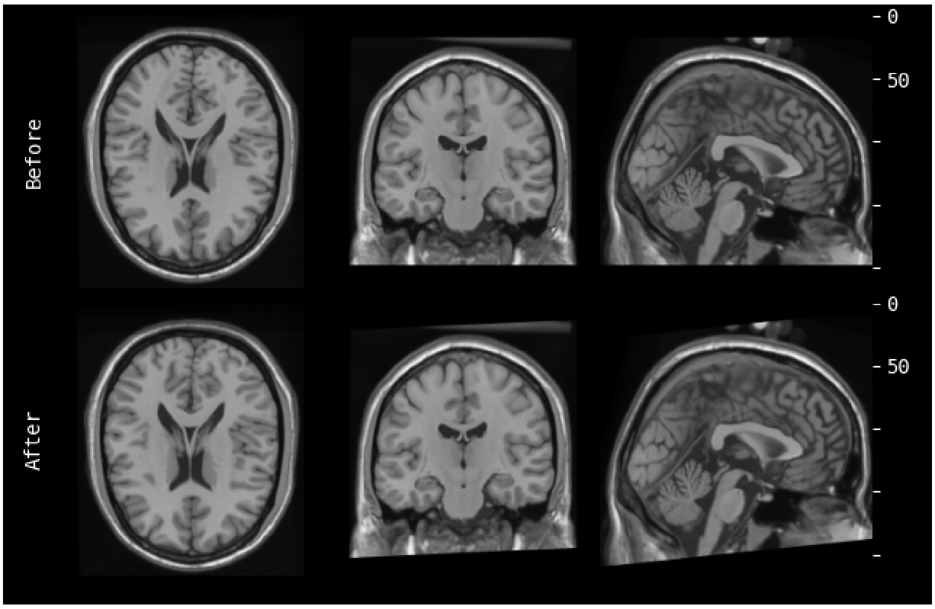


Figure 3.24: Affine Transformation (Cardoso et al., 2022)

Each transformation degree is selected from a given range. Table 3.7 shows the transformation and their respective ranges of values.

Table 3.7: Transformation Ranges for Affine Transformation

Transformation	Range
Translation (x-axis)	(-40, 40)
Translation (y-axis)	(-40, 40)
Translation (z-axis)	(-2, 2)
Rotation (x-axis)	(-5, 5) degrees
Rotation (y-axis)	(-5, 5) degrees
Rotation (z-axis)	(-45, 45) degrees
Scaling (x-axis)	(0.85, 1.15)
Scaling (y-axis)	(0.85, 1.15)
Scaling (z-axis)	(0.85, 1.15)

3.5.3 Elastic Deformation

The Elastic Deformation augmentation combines affine transformations and elastic deformations to generate new images. Figure 3.25 illustrates an example image before and after the augmentation has been applied.

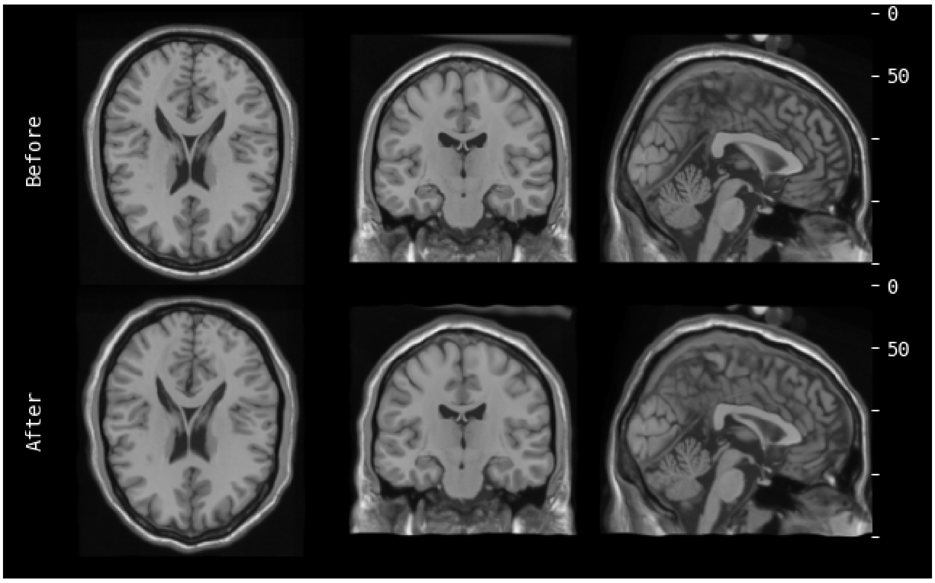


Figure 3.25: Elastic Deformation (Cardoso et al., 2022)

The smoothness of the deformation is controlled by the sigma parameter, while the magnitude of the deformation is determined by a magnitude hyperparameter. Table 3.8 shows the transformation and their respective ranges of values.

Table 3.8: Transformation Ranges for Elastic Deformation

Transformation	**Range**
Sigma	(5, 8)
Magnitude	(100, 200)
Translation (x-axis)	(-50, 50)
Translation (y-axis)	(-50, 50)
Translation (z-axis)	(-2, 2)
Rotation (x-axis)	(-5.0, 5.0) degrees
Rotation (y-axis)	(-5.0, 5.0) degrees
Rotation (z-axis)	(-180, 180) degrees
Scaling (x-axis)	(0.85, 1.15)
Scaling (y-axis)	(0.85, 1.15)
Scaling (z-axis)	(0.85, 1.15)

3.6 Implemented GAN Architectures

The following chapter provides a overview of the GAN architectures utilized in the thesis. It begins with a detailed examination of the network architecture, highlighting the key features of each GAN model employed in the study. Additionally, it covers the training procedure adopted for each GAN architecture.

3.6.1 Vanilla GAN

The Vanilla GAN architecture is a simple implementation inspired by (Goodfellow et al., 2020), with modifications such as a U-net generator. It is developed using Python and Tensorflow 2 (Abadi et al., 2015) and Keras (Chollet et al., 2015). Through experimentation, the architecture was modified. As such, the Vanilla GAN architectures are numbered Vanilla GAN 1 and 2. Vanilla GAN 1 features a U-net architecture, complete with skip connections, comprised of seven encoder and seven decoder blocks, illustrated in Figure 3.26.

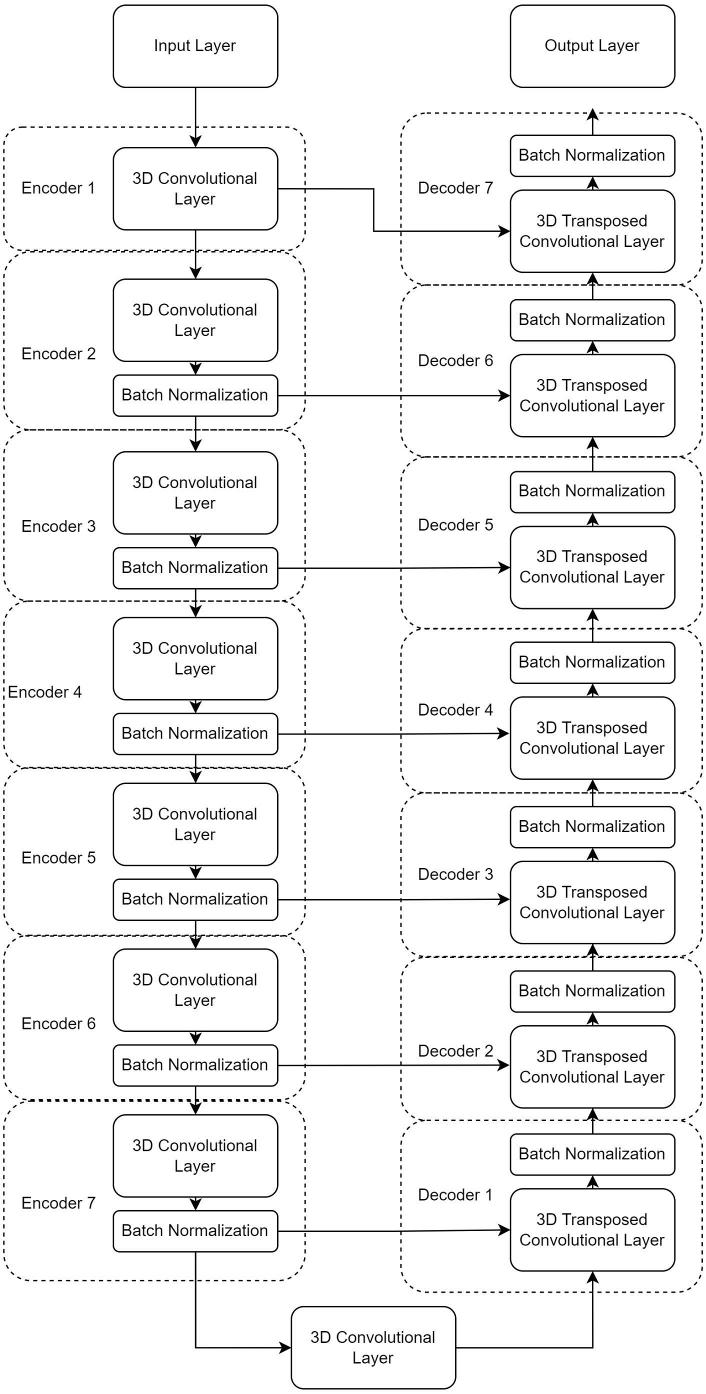


Figure 3.26: Vanilla GAN 1 Generator Architecture

The first layer of the generator is the input layer, which receives a random noise vector with the same dimensions as the input images. This input layer is connected to the initial encoder block, composed of a 3-dimensional convolutional layer. All subsequent encoder blocks are identical, except a batch normalization layer added after the convolutional layer.

The encoder blocks connect to the decoder blocks through skip connections, with a bottleneck following the encoder blocks. The decoder comprises of seven blocks, each including a 3-dimensional convolutional transpose layer, followed by a corresponding dropout layer. Both the generator and discriminator employ the ADAM optimizer. The loss function for the generator and discriminator is binary cross-entropy, as described in the original GAN paper (Goodfellow et al., 2020).

However, the initial architecture's complexity caused memory errors during experimentation, leading to the development of Vanilla GAN 2. This architecture features only four encoders and decoders, significantly reducing the number of neurons. The network architecture is shown in Figure 3.27.

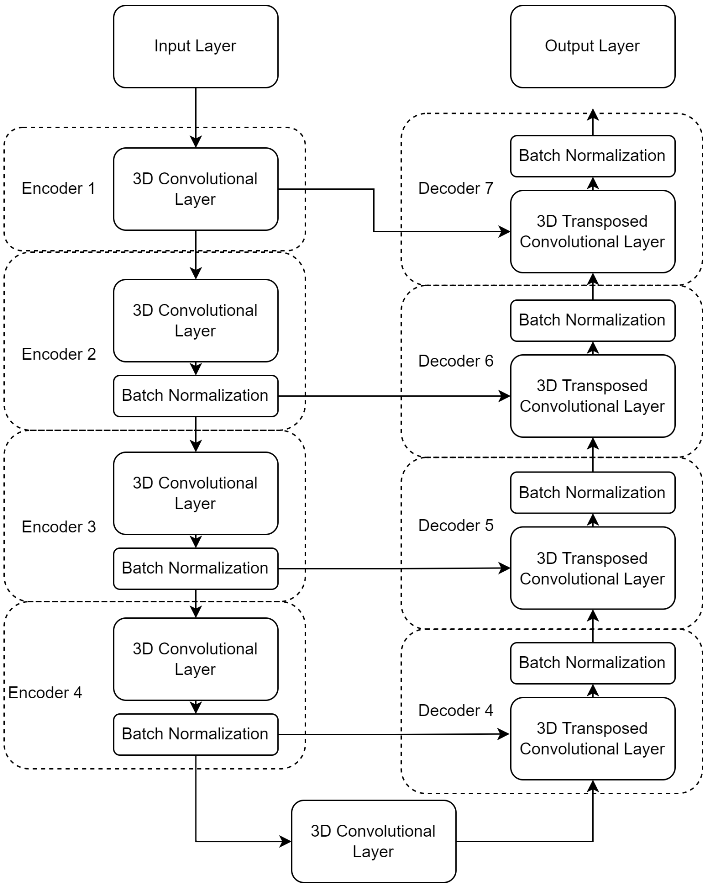


Figure 3.27: Vanilla GAN 2 Generator Architecture

The Vanilla GANs are trained like a conventional GAN. During each epoch, a random noise vector is created and provided as input to the generator, which generates a corresponding image vector. The discriminator receives both the generated image and a random real image and predicts which one is real and which one is fake. Based on the discriminator's predictions, the generator and discriminator are updated. The network gradients are retrieved and applied to the networks, optimized using the network optimizers.

3.6.2 Wasserstein GAN

The second GAN architecture is derived from the Wasserstein GAN, as detailed in the original paper by (Arjovsky et al., 2017). Like the Vanilla GANs, this architecture is implemented in Python and Tensorflow 2. The generator and discriminator architectures remain unchanged from the Vanilla GAN. However, instead of the Adam optimizer, the Wasserstein GAN utilizes the RMSprop optimizer (Huk, 2020) for both networks.

As discussed in the Chapter 2.6.1 the WGAN can employ weight clipping or gradient penalty regularization. Two versions of the Wasserstein GAN is implemented, one with weight clipping (WGAN-WC) and one with gradient penalty regularization (WGAN-GP).

Similar to the Vanilla GAN, the training of WGAN-WC begins by passing the generator a random noise vector, which is used to generate a new image. In WGAN-WC, the generated image is given to the discriminator, which makes a prediction, along with a prediction on a real image. Both predictions are used to calculate the Wasserstein loss. The gradients are extracted and applied with the discriminator optimizer. After the gradients are applied, the weights of the discriminator are clipped based on the clip weight hyperparameter. Then the generator generates a new image based on the noise, and the Wasserstein loss is calculated based on the generated image. Finally, the gradients are extracted and applied to the generator optimizer.

The WGAN-GP is trained in a similar fashion like the WGAN-WC. After the loss is calculated for the discriminator, a gradient penalty is calculated with respect to the real and generated images. The gradient penalty is then multiplied by the discriminator loss. The generator is trained like in the WGAN-WC.

3.6.3 FastGAN

The FastGAN model is implemented with Python and Tensorflow 2.

At each training step, random noise is created with the shape of the batch size and latent dimension. First, the discriminator is trained separately a defined number of times. The generator creates an output based on the noise. Differential augmentation is added to both the real images in the batch and the generated images. The augmented images are then fed to the discriminator, and the discriminator loss is calculated. Then the recreation loss is calculated by passing the real images and the discriminator prediction on the real images multiplied by a recreation weight. The recreation weight attribute is a hyperparameter that controls the relative weight of the reconstruction loss in the overall loss function of the model. The total discriminator loss is calculated by adding the discriminator loss together with a gradient penalty and the reconstruction loss. Finally, the gradients are extracted and applied to the discriminator network.

A random noise vector is again created and fed to the generator, which creates an output. Differentiable augmentation is added to the output, and fed to the discriminator. Then the generator loss is calculated. The gradients are extracted and applied to the generator network.

3.6.4 StyleGAN2

StyleGan2 is implemented with Python and the Tensorflow library Keras (Chollet et al., 2015).

To train the StyleGAN2 network, each training step is started by generating a noise vector and two random styles for the images we want to generate. Then we transform the noise input style into a latent space representation that the model can use. We stack the styles together and generate a seed based on the stacked representation. Finally, we feed the seed, stacked images, and noise vector to the generator. Optionally, a gradient penalty can be applied to the discriminator training. The generated images are then fed to the discriminator, and the discriminator loss is calculated. Finally, the generator and discriminator average loss is calculated, and the gradients are applied to the networks.

3.6.5 Hierarchical Amortized GAN

HA-GAN is implemented with Python and PyTorch (Paszke et al., 2019). The HA-GAN consists of two network implementations, one for generating $128 \times 128 \times 128$ images and the other for generating $256 \times 256 \times 256$ images. The overall architecture, which is similiar for both versions of the network, is illustrated in Figure 3.28.

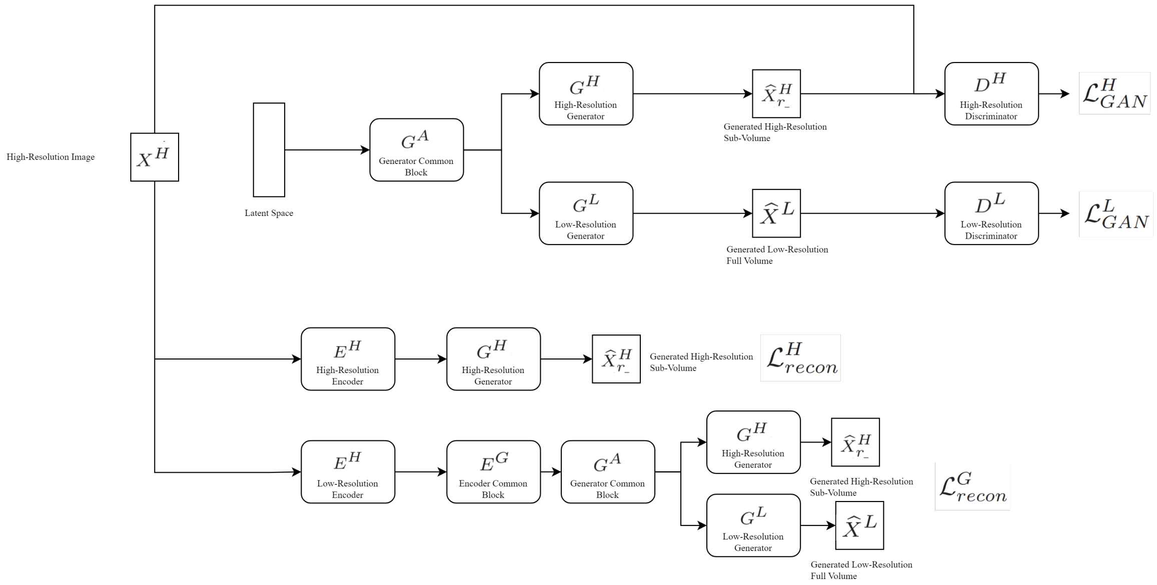


Figure 3.28: HA-GAN Architecture

Unlike previous architectures, the model training is measured in steps rather than epochs. Each training step, the real images are interpolated with a scale factor of 0.25 to create a low-resolution volume of the images. Then a high-resolution sub-volume of the image is selected randomly. The low-resolution volume and the high-resolution sub-volume are then fed to the discriminator, which makes a prediction. Based on the prediction, the discriminator loss is calculated with binary cross-entropy.

With the batch size and latent dimension, random noise is created and fed to the generator, along with the position of the high-resolution sub-volume from the real images. The generator then creates a high-resolution sub-volume and a low-resolution volume. The output of the generator is fed to the discriminator along with the position of the high-resolution sub-volume. The fake loss is calculated with the binary cross-entropy between the fake labels and the generator output. Then the discriminator loss is calculated by adding the discriminator real loss with the discriminator fake loss. The discriminator optimizer is then used to optimize the network weights based on the calculated loss.

A random noise vector is created and fed to the generator, which generates a high-resolution sub-volume and a low-resolution full volume. The generated volumes are passed to the discriminator, which makes a prediction based on the volumes and the real images. Finally, the loss is calculated with binary cross entropy. The generator optimizer is then used to update the weights. The training of the generator is repeated a number of times, defined as a hyperparameter.

The randomly selected high-resolution sub-volume from before is passed to the encoder, which creates a latent representation. The latent representation is the generator seed, stacked images, and noise vector. The encoder loss is calculated with MSE between the generated full-volume image of the latent representation and the randomly selected high-resolution sub-volume. The encoder loss is then backpropagated through the encoder network, and the encoder weights are updated using the optimizer.

After the encoder, the sub-encoder will be trained. The latent representation of each sub-volume of the full image is then obtained by iterating over all sub-volumes of the real images. The sub-volumes are concatenated and passed to the sub-encoder, which creates a further compressed representation that is passed to the generator. The generator then generates a high-resolution sub-volume and low-resolution full volume of the latent representation. Then the sub-encoder loss is calculated with MSE between the generated high-resolution sub-volume and the real sub-volume, and MSE between the low-resolution generated full volume and the real low-resolution image. Finally, the sub-encoder weights are updated with the encoder optimizer.

3.7 Hyperparameter Optimization

Python is used to execute HPO through a grid search method. The hyperparameters are specified in a configuration file, and the HPO generates a grid containing all feasible combinations of these hyperparameters. Although there are no implemented HPO methods for FastGAN and StyleGAN2 architectures, the Vanilla, WGAN, and HA-GAN architectures have them.

Table 4.9 presents the hyperparameters for HPO, which vary based on the architecture used. The Vanilla and Wasserstein architectures utilize the hyperparameters listed in Table 4.9.

Table 3.9: Vanilla/WGAN Hyperparameters

Hyperparameter	**Description**
img_per_epoch	Number of images per epoch
architecture	Generator architecture

Not be confused with batch size, images per epoch is simply how many images will be shown to the generator each epoch. Architecture is the generator architecture to be used, which can be Vanilla (2) or WGAN.

The hyperparameters for the HA-GAN architecture are shown in Table 3.10.

Table 3.10: HA-GAN Hyperparameters

Hyperparameter	**Description**
g_lr	Generator learning rate
d_lr	Discriminator learning rate
e_lr	Encoder, Sub-encoder learning rate
latent_dim	Latent dimension for generator

3.8 Inference and data Postprocessing

Inference refers to the process of generating new samples from a trained model. Once a model has been trained on a particular dataset, it can be used to generate new, synthetic samples that resemble the training data. The steps explained in this chapter is related to the inference and data postprocessing of the data generated by the HA-GAN model, described in Chapter 3.6.5.

3.8.1 Generating Images

To perform inference, we first load a specific model that has been saved after training. We load only the generator, encoder, and sub-encoder, as we do not need the discriminator for inference. We input a random noise vector with the size of the latent dimension that the network was trained on. The vector is fed to the generator network of the trained GAN. The generator network generates a new sample based on the input noise vector. This process can be repeated multiple times to generate a set of new image/mask pairs.

3.8.2 Postprocessing

The postprocessing procedures outlined in this subsection are not universally implemented across all experiments but rather selectively employed in some cases.

After the image/mask pair has been generated, the HU intensity may be rescaled, as expressed by Equation 13.

$$image = 0.5 \times image + 0.5 \tag{13}$$

Where image is the output of the GAN. The operation is done to map the output values to a more meaningful and interpretable range. Following this, the pair is rescaled to the low-high threshold that the images were originally interpolated to before training, expressed in Equation 14.

$$\text{image} = \text{image} \times (\text{high\_threshold} - \text{low\_threshold}) + \text{low\_threshold} \tag{14}$$

Finally, the pair is converted from a NumPy array to a NIFTI image. Then the pair is separated into an image and mask by cutting the image on the z-dimension in two equal parts with NumPy. Generated pairs with image size $128 \times 128 \times 128$ and $256 \times 256 \times 256$ will yield separate images and masks with $128 \times 128 \times 64$ and $256 \times 256 \times 128$, respectively. The process is shown in Figure 3.29.

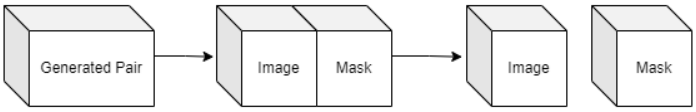


Figure 3.29: Image/mask Separation

The mask may be further post-processed by converting the voxel values to binary. All values above 0 are set to 1, while all values below are set to 0.

An alternative to binary mask preprocessing is Connected Component Analysis (CCA) (Dillencourt et al., 1992), a technique used to identify and analyze groups of pixels in an image that are connected to each other. CCA is expressed in Equation 15.

$$C_i = p \in P : p \text{ is connected to } q_i, \tag{15}$$

where P is the set of all voxels in the image, q_i is a pixel with a specified property (such as being above a threshold), and C_i is the set of all voxels in P that are connected to q_i. The basic idea behind CCA is to identify groups of voxels that are connected to each other. The set C_i includes all voxels that are connected to a specific pixel q_i that meets certain criteria. By analyzing the sets C_i for multiple q_i values, we can identify and isolate different voxel groups in the image.

3.9 GAN Evaluation

The evaluation of GANs is a critical step in assessing their performance and determining their effectiveness in generating high-quality synthetic data.

3.9.1 Loss Diagrams

One simple approach to evaluating the performance of Generative Adversarial Network (GAN) models is to examine the loss diagrams. An example loss diagram from a GAN training session is shown in Figure 3.30.

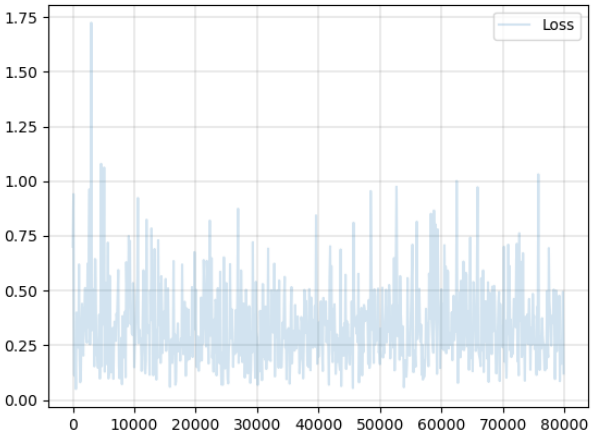


Figure 3.30: Example GAN Loss Diagram

The loss diagrams provide a quantitative measure of how well the GAN model is performing during the training process. The generator loss measures how well the generated images match the real images, while the discriminator loss measures how well the discriminator can distinguish between the real and generated images.

By analyzing the loss diagrams, we can identify patterns and trends in the training process. For example, if the generator loss is decreasing while the discriminator loss is increasing, it could indicate that the generator is producing better images, but the discriminator is becoming better at distinguishing between real and generated images.

To make the interpretation of the diagrams easier, we apply Exponential Moving Average (EMA) (Klinker, 2011), a method used to smooth out the noise in a time series by giving more weight to recent values and less weight to past values. EMA calculates the moving average of the loss function by giving more weight to recent loss values and less weight to past values and is expressed in Equation 16.

$$EMA[t] = \alpha \cdot loss[t] + (1 - \alpha) \cdot EMA[t-1] \tag{16}$$

Where α is a smoothing factor between 0 and 1, and $EMA[t-1]$ is the EMA value from the previous iteration. As α approaches 1, more weight is given to recent loss values, making the EMA more responsive to changes in the loss. As alpha approaches 0, less weight is given to recent loss values, making the EMA smoother. We apply α 0.001 to our diagrams. Figure 3.31 shows an example diagram with the original loss and EMA loss.

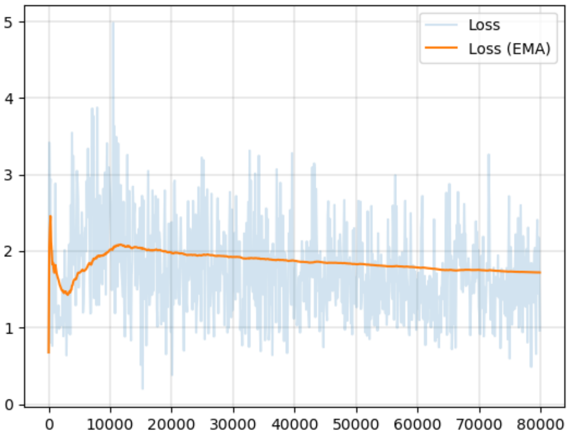


Figure 3.31: Original Loss (blue), EMA Loss (orange)

However, it is essential to note that the loss diagrams only provide a quantitative measure of the GAN model's performance. Therefore, while examining the loss diagrams is a useful evaluation tool, it should be combined with qualitative metrics to ensure the GAN model is producing high-quality images that accurately represent the training data.

3.9.2 Inception Score

We implement a ResNet-50 (Koonce and Koonce, 2021) in PyTorch, customized for 3D image input. We obtain pre-trained weights for the ResNet model. We prepare a Pytorch data loader for loading the GAN-generated images, so they can be used for the calculation. We load the pre-trained weights into the model and use it to predict on the generated images loaded by the data-loader. By predicting on the images, we extract the feature activations for the images. We use the activations to calculate the KL divergence for each image, expressed in Equation 17.

$$\text{KL divergence} = p(y|x) \cdot (\log(p(y|x)) - \log(p(y))) \tag{17}$$

The KL divergence is summed, and the average across all images are calculated. The average is then exponentiated to get the final IS metric. The overall process is illustrated in Figure 3.32.

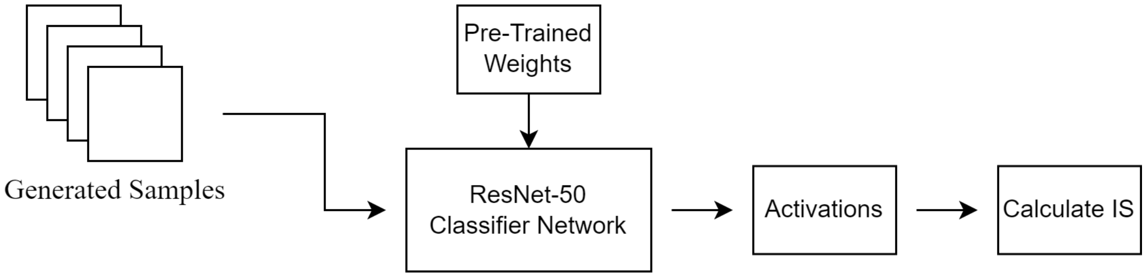


Figure 3.32: Process of Attaining IS Score

The process is done for both the generated images and masks separately.

3.9.3 Fréchet Inception Distance

We prepare two PyTorch data loaders, one for the real images and one for the generated images, ensuring that the images are of the same size and format. We use the same pre-trained 3D ResNet-50 as the IS calculation. We predict on the real and generated images to extract the feature activations for both sets of images. The activations are then used to calculate the FID score. The overall process is outlined in Figure 3.33.

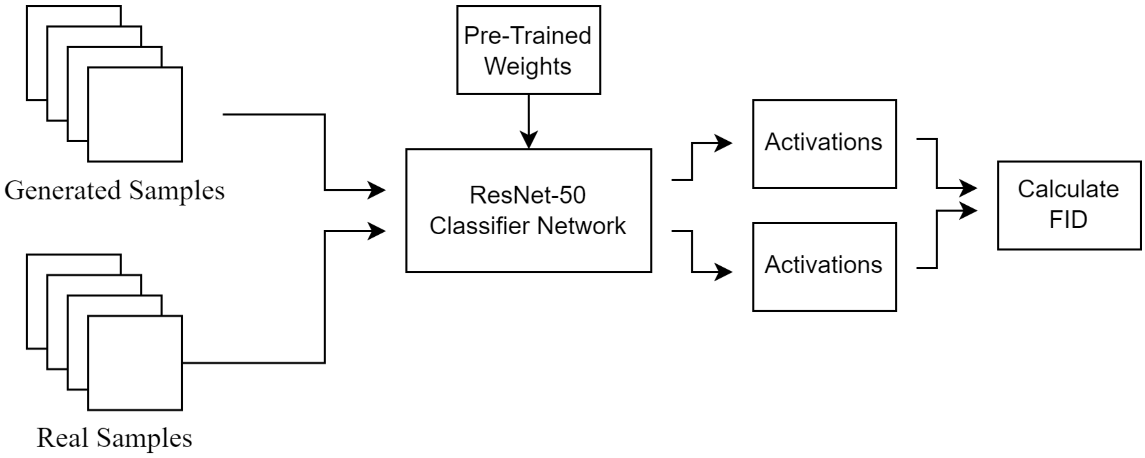


Figure 3.33: Process of Attaining FID Score

To calculate the FID score, we start by calculating the mean and covariance of the feature activations. Using the mean values, we calculate the sum squared difference between them. We calculate the matrix square root of the product of the two covariance matrices. This quantity measures the distance between the two datasets in terms of their covariance structures. Finally, we add the calculations together to get the final FID score. FID is expressed with Equation 18.

$$FID(act1, act2) = ||\mu_1 - \mu_2||^2 + Tr(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{\frac{1}{2}}) \tag{18}$$

where μ_1 and μ_2 are the mean activation vectors of real image activations (act1) and the generated image activations (act2), respectively. Σ_1 and Σ_2 are the covariance matrices of act1 and act2, respectively. Tr denotes the trace operator, which computes the sum of the diagonal elements of a matrix. $(\Sigma_1\Sigma_2)^{\frac{1}{2}}$ denotes the matrix square root of the product of Σ_1 and Σ_2, computed using the ”sqrtm” function from NumPy. $||\mu_1 - \mu_2||_2^2$ is the squared Euclidean distance between μ_1 and μ_2.

The process involves computing the FID score for the generated image/mask pairs against the real ones. Subsequently, we split the pairs into image and mask components, then evaluate the FID score between the real and generated images.

Finally, we compute the FID score between the real masks and the generated masks.

3.9.4 t-Distributed Stochastic Neighbor Embedding

We prepare two PyTorch data loaders, one for the real images and one for the generated images, ensuring that the images are of the same size and format. We use the same pre-trained 3D ResNet-50 as the IS and FID calculation. We use the ResNet-50 to extract feature representations for both sets of images.

We concatenate the sets of images into a single dataset. We apply t-SNE to the dataset, transforming the data into a 2D space. We set the perplexity to 10 and the learning rate to 200. We fit the dataset and plot the final results in a matplotlib scatterplot.

We apply t-SNE to a combination of real and generated data by different GAN models.

3.10 Segmentation

To perform the segmentation of 3D images, we implement the Auto3DSeg segmentation pipeline from the MONAI framework, which is the same method used by the winning team (Myronenko et al., 2023) in the HECKTOR 2022 challenge (Andrearczyk et al., 2023).

3.10.1 Preprocessing

Preprocessing of the input data is different for real images and generated images. For real images, we first binarize the mask data to contain only 0 and 1. We resize the image with linear interpolation and resize the mask with nearest-neighbor interpolation. We save the images as NIFTI images.

The generated image/mask pairs are split into separate images and masks. We binarize the mask. We experiment with various post-processing steps for generated images, outlined in Chapter 3.8.2.

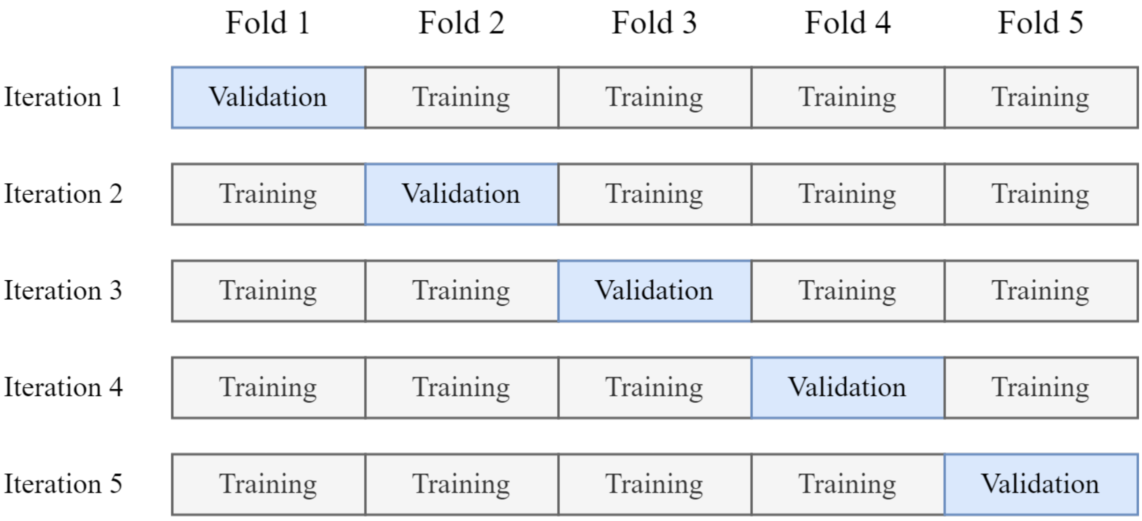
3.10.2 Model

The model used for the Auto3Dseg is the SegResNet, the only available algorithm for multi-resolution images such as PET and CT images.

We use the DiceCELoss loss function for the network, which is a combination of dice and Cross-Entropy. The network is optimized with ADAMW, a variant of the ADAM optimizer that uses weight decay in a different manner than that of ADAM. Weight decay is a regularization technique that prevents overfitting by adding a small penalty to the loss function. This penalty discourages learning a more complex or flexible model, hence reducing the risk of overfitting. ADAMW applies weight decay after the adaptive learning rates have been calculated, in contrast to ADAM which applies the decay to the gradients before computing adaptive learning rates. With ADAMW, the effect of weight decay is the same regardless of how the learning rate is set, making it easier to choose a good learning rate.

3.10.3 Training

We create multiple datasets consisting of real-only, real, and generated and generated-only image/mask pairs. The datasets are split into 5 folds. We train the segmentation network with K-Fold Cross-Validation (CV) and single-fold training. K-Fold Cross-Validation splits the training data into K groups, which decides what data will be used for training and validation. 5-Fold CV is shown in Figure 3.34.



	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Iteration 1	Validation	Training	Training	Training	Training
Iteration 2	Training	Validation	Training	Training	Training
Iteration 3	Training	Training	Validation	Training	Training
Iteration 4	Training	Training	Training	Validation	Training
Iteration 5	Training	Training	Training	Training	Validation

Figure 3.34: 5-Fold Cross-Validation

3.10.4 Evalation and Inference

The Auto3DSeg pipeline automatically stores loss and DSC per epoch. Additionally, the best average DSC is saved for epochs reaching a new top DSC score.

When training the network and testing the network with separate data, we first train the network and run inference on the test data. This means that the network will generate prediction masks for the input test data. We use the prediction masks to compare with the real images. We calculate DSC across the predictions to get the final evaluation score.

4 Experiments

4.1 Experimental Setup

The experimental setup for generating 3D CT image mask pairs with GANs involves the use of specific hardware and software tools to facilitate the training and evaluation of the GAN model. In this case, three primary hardware setups were used for the experiments, two local computers and the Ex3 Cluster (Simula, 2023) with multiple nodes.

The locally hosted computers used for the experiments was equipped with an RTX 3070 and RTX 4090 graphics cards. The Ex3 Cluster was also used for the experiments. The Ex3 Cluster is a high-performance computing system that comprises multiple nodes, each equipped with a powerful GPU. Specifically, the nodes in the Ex3 Cluster are equipped with either Nvidia A100 or V100 GPUs. These GPUs are high-end models that are specifically designed for DL applications and are capable of delivering exceptional performance for training and inference tasks.

The hardware used is shown in Table 4.1.

Table 4.1: Hardware

Setup	Name	Memory
Local Machine 1	NVIDIA RTX 3070	6GB
Local Machine 2	NVIDIA RTX 4090	21GB
Ex3 Cluster Node 1	NVIDIA A100	80GB
Ex3 Cluster Node 1	NVIDIA V100	32GB

For the experiments in which Vanilla GAN, Wasserstein GAN, and StyleGAN2 were used, TensorFlow Keras was used as the DL framework. The HA-GAN model and FastGAN, on the other hand, were implemented using PyTorch.

The software used is shown in Table 4.2.

Table 4.2: Software

Name	Reference
Tensorflow 2	(Abadi et al., 2015)
Keras	(Chollet et al., 2015)
PyTorch	(Van Rossum and Drake, 2009)
PyCharm	(JetBrains, 2023)

Figure 4.1 outlines the overall experimentation process with the different GAN models.

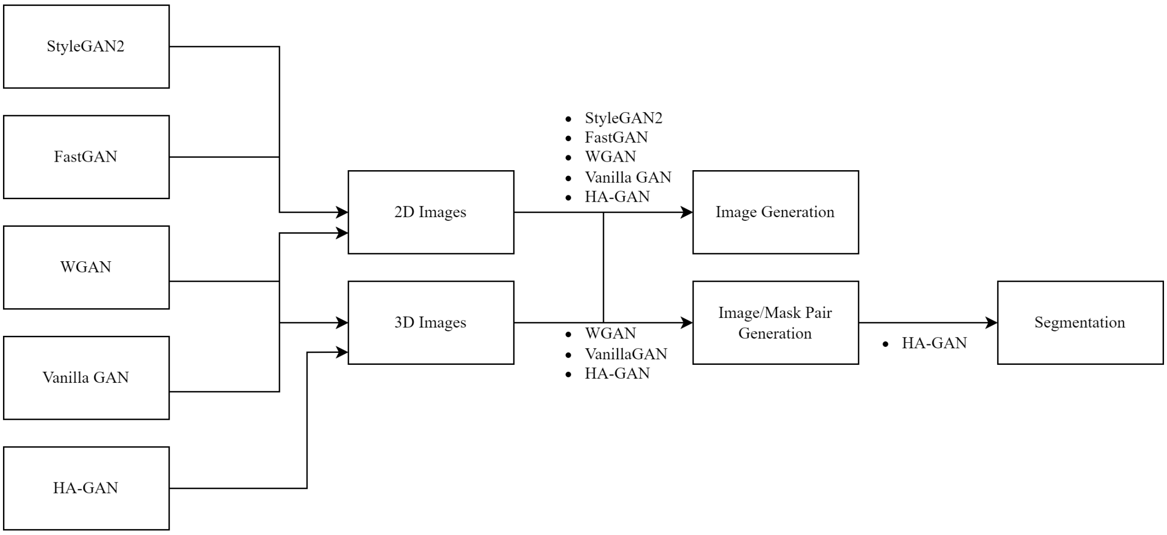


Figure 4.1: Overview of Experiments

4.2 Vanilla GAN

In this experiment, we utilized the original image/mask pairs from the HECKTOR dataset described in Chapter 3.1. However, due to the high computational load associated with processing the high-resolution images, we encountered memory issues that prevented the model from starting training. To address this problem, we created three new datasets by resampling the images and then cropping and padding them as described in Chapter 3.4.1. After preprocessing, we obtained three new datasets with image dimension:

- $128 \times 128 \times 256$
- $100 \times 100 \times 200$
- $104 \times 104 \times 148$

Unfortunately, we still encountered memory problems with all three resolutions. To overcome this issue, we created a smaller dataset by resampling the images to a resolution of $16 \times 16 \times 16$. We concatenate the image/mask pairs along the z-axis as described in Chapter 3.4.1 and trained the model for 200 epochs with a batch size and images per epoch set to 1. We used the ADAM optimizer for both the generator and discriminator with a learning rate of 0.0001.

Table 4.3 presents the details of the experimental setup for the Vanilla GAN.

Table 4.3: Experimental Setup for Vanilla GAN

Property	**Values**
Image Preprocessing	Resampling, Cropping, Padding
Mask Preprocessing	Resampling, Cropping, Padding
Model Architecture	Vanilla GAN
Image Size	$16 \times 16 \times 16$

Table 4.4 presents the hyperparameters used for training the Vanilla GAN.

Table 4.4: Hyperparameters for Vanilla GAN

Hyperparameter	**Values**
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0001
Optimizer	ADAM
Epochs	200
Latent Dimension Size	16
Batch Size	1
Images Per Epoch	1

4.3 Improved Architectures

4.3.1 Vanilla GAN 2

In the previous experiment, we trained a Vanilla GAN successfully, but needed to increase the image size. To avoid the memory issues faced earlier, we made some changes. Specifically, we reduced the size of the network in the generator to four encoders and decoders and decreased the number of neurons in both the generator and discriminator.

Unfortunately, we still experienced memory problems when working with images of resolutions $128 \times 128 \times 256$, $100 \times 100 \times 200$, and $104 \times 104 \times 148$. To overcome this, we resampled the images to approximately 30% of their original size and then center-chopped 27 slices from both the image and mask. After that, we concatenated the image and mask along the Z dimension, resulting in a $154 \times 154 \times 54$ image.

For this experiment, we used the same setup as before but with a modified generator learning rate inspired by (Bu et al., 2021). The details of the experimental setup are shown in Table 4.5.

Table 4.5: Experimental Setup for Vanilla GAN 2

Property	**Values**
Image Preprocessing	Resample, Crop, Pad
Mask Preprocessing	Resample, Crop, Pad
Model Architecture	Vanilla GAN 2
Image Size	$154 \times 154 \times 54$

Table 4.6 shows the hyperparameter used in the experiment.

Table 4.6: Hyperparameters for Vanilla GAN 2

Hyperparameter	**Values**
Generator Learning Rate	0.0002
Discriminator Learning Rate	0.0001
Optimizer	ADAM
Epochs	200
Latent Dimension Size	154
Batch Size	1
Images Per Epoch	1

4.3.2 Wasserstein GAN

In the Wasserstein GAN experiment, we used the same $154 \times 154 \times 54$ images from the previous experiment. We trained both variants of the WGAN separately, with weight clipping and gradient penalty. The weight clipping value was set to 0.01, and the gradient penalty weight was set to 10. We used the same learning rates and epochs as in the previous experiment.

Table 4.7 presents the details of the experimental setup.

Table 4.7: Experimental Setup for WGAN

Property	**Values**
Image Preprocessing	Resample, Crop, Pad
Mask Preprocessing	Resample, Crop, Pad
Model Architecture	Wasserstein GAN
Image Size	$154 \times 154 \times 54$

Table 4.8 show the hyperparameters used in the WGAN experiment.

Table 4.8: Hyperparameters for WGAN

Hyperparameter	**Values**
Generator Learning Rate	0.0002
Discriminator Learning Rate	0.0001
Optimizer	RMSProp
Weight Clip Value	0.01
Gradient Penalty Value	10
Epochs	200
Latent Dimension Size	154
Batch Size	1
Images Per Epoch	1

4.3.3 Hyperparameter Optimization

To enhance the results of both the Vanilla GAN and WGAN, we conducted HPO experiments. We started by selecting a set of hyperparameters for both models, including the learning rate for the generator and discriminator, as well as images per epoch.

Next, we defined a range of possible values for each hyperparameter and used a grid search approach to test all possible combinations of these values. Specifically, we trained and evaluated each GAN model with every possible combination of hyperparameters using the same dataset. We tested all configurations on both Vanilla 2 and Wasserstein GANs.

The grid search approach resulted in a total of 18 experiments, with each experiment representing a unique combination of hyperparameters. During each experiment, we saved the performance metrics for both the generator and discriminator loss. We used these metrics to compare and evaluate the performance of each model.

Table 4.9 presents the hyperparameters and their respective possible values.

Table 4.9: Hyperparameters and descriptions

Hyperparameter	**Values**
Images per epoch	16, 32, 64, 128, 256, 512
Generator learning rate	0.0001, 0.0002, 0.0003, 0.0004
Discriminator learning rate	0.0001, 0.0002, 0.0003, 0.0004

For this experiment, we reduced the image size to $38 \times 38 \times 38$ to decrease the required computational time. Each model was trained for 200 epochs.

4.4 2-dimensional GANs

4.4.1 Wasserstein GAN

To address the issue of poor results obtained from training GAN models on 3D images, we decided to try a 2D approach. We started by resampling the 3D images with a resampling factor of 50%, resulting in an average image size of $256 \times 256 \times 100$. Next, we preprocessed the images from 3D to 2D, as described in Chapter 3.4.2. Finally, we padded and chopped the images, resulting in images of size 1024×3680.

We modified the Wasserstein generator and discriminator to handle 2D images instead of 3D images and ran training for 200 epochs.

The experimental setup is shown in Table 4.10.

Table 4.10: Experimental Setup for WGAN 2D

Property	**Values**
Image Preprocessing	Resample, Chop, Pad, 3Dto2D
Model Architecture	Wasserstein GAN
Image Size	1024 x 3680

Table 4.11 shows the hyperparameters used in the 2D WGAN experiments.

Table 4.11: Hyperparameters for WGAN 2D

Hyperparameter	**Values**
Generator Learning Rate	0.0002
Discriminator Learning Rate	0.0001
Optimizer	RMSProp
Weight Clip Value	0.01
Gradient Penalty Value	10
Latent Dimension Size	154
Epochs	200
Batch Size	1
Images Per Epoch	1

4.4.2 FastGAN

We use the same 2D images as in the previous experiment. We train the FastGAN network on the images for 500 epochs in multiple experiments. We experiment with multiple latent dimensions, 256, 512, and 1024. We set the learning rate to 0.0002 for the generator and discriminator. We set the batch size to 8.

The experimental setup is shown in Table 4.12.

Table 4.12: Experimental Setup for FastGAN

Hyperparameter	**Values**
Image Preprocessing	Resample, Chop, Pad, 3Dto2D
Model Architecture	FastGAN
Image Size	1024×3680

Table 4.13 shows the hyperparameters.

Table 4.13: Hyperparameters for FastGAN

Hyperparameter	**Values**
Generator Learning Rate	0.0002
Discriminator Learning Rate	0.0002
Optimizer	ADAM
Optimizer β_1	0.5
Optimizer β_2	0.99
Latent Dimension Size	256, 512, 1024
Epochs	500
Batch Size	8

4.4.3 StyleGAN2

In this experiment, we used the same images as in the previous experiment, which were resampled 3D images that were converted to 2D and then padded and chopped to a size of 1024×3680.

We trained the StyleGAN2 model on 2D images for 500 epochs and experimented with different hyperparameters to identify the optimal set of parameters for the model. Specifically, we tested the model with latent dimensions of both 128 and 256.

Table 4.14 presents the details of the experimental setup.

Table 4.14: Experimental Setup for StyleGAN2

Property	**Values**
Image Preprocessing	Resample, Chop, Pad, 3Dto2D
Model Architecture	StyleGAN2
Image Size	1024×3680

Table 4.15 shows the hyperparameters.

Table 4.15: Hyperparameters for StyleGAN2

Hyperparameter	**Values**
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Latent Dimension	128, 256
Channels	32
Epochs	500
Batch Size	8

4.5 Generating Images only with HA-GAN

To simplify the challenge of generating CT images, we decided to focus solely on generating images during our initial trials without including the corresponding masks. To generate the images, we utilized the original CT images and applied interpolation (interp) of the intensity values, as described in Chapter 3.6.5.

For this experiment, we implemented the HA-GAN architecture with a $128 \times 128 \times 128$ configuration and trained the model for 80000 steps. The latent dimension was kept at the default value of 1024, and the batch size was set to 4. To train the generator, discriminator, and encoder networks, we used learning rates of 0.0001, 0.0004, and 0.0001, respectively, as defined in the original approach proposed by (Sun et al., 2022).

The experiment was conducted twice with distinct training data configurations. In the first configuration, we feed the data in its original order to the model, while in the second, we shuffle the data and feed it in a random order to the model,

The specific details of the experimental setup are summarized in Table 4.16.

Table 4.16: Experimental Setup for Generating Images with HA-GAN

Property	**Values**
Model Architecture	HA-GAN $128 \times 128 \times 128$
Data	Original CT images
Image Size	$128 \times 128 \times 128$
Image Preprocessing	Resize, Interp. $[-1024, 600]$, $[-1, 1]$

Table 4.17 shows the hyperparameters.

Table 4.17: Hyperparameters for Generating Images with HA-GAN

Hyperparameter	**Values**
Training Steps	80000
Latent Dimension	1024
Batch Size	4
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

In order to improve the outcomes of our previous experiment, we enriched the dataset by generating a greater number of augmented images. We have created several datasets, each with different augmentation configurations, to assess the performance of voxel resampling. Specifically, we have generated datasets that are resampled to the following voxel dimensions: $1 \times 1 \times 1mm$, $1.5 \times 1.5 \times 1.5mm$, $2 \times 2 \times 2mm$, and $3 \times 3 \times 3mm$.

Additionally, we created another dataset that includes resampled images with voxel dimensions of $1.5 \times 1.5 \times 1.5mm$. We have augmented these resampled images even further by applying affine transformations and elastic deformations. The final dataset contains a total of 1500 images, with 500 images being the original resampled images and an additional 500 images per augmentation.

4.6 Generating Image/mask Pairs with HA-GAN

We use the cropped image/mask pairs detailed in Chapter 3.4.3. The image and mask are resized individually to size $128 \times 128 \times 64$. The reason for the Z dimension size of 64 is that the HA-GAN model requires the input to be of size 128, which is the size of the concatenated image and mask. The image and mask are interpolated with the same range as the previous experiment. After interpolation, the image and mask are concatenated along the z-axis.

The specific details of the experimental setup are summarized in Table 4.18.

Table 4.18: Experimental Setup for Generating Image/mask with HA-GAN

Property	**Values**
Model Architecture	HA-GAN $128 \times 128 \times 128$
Data	Cropped CT image/mask pairs
Image Size	$128 \times 128 \times 128$
Image Preprocessing	Resize, Interp. $[-1024/600]$, $[-1/1]$
Mask Preprocessing	Resize, Interp. $[-1024/600]$, $[-1/1]$
Concatenated Image Preprocessing	None
Concatenation Method	Z-axis

Table 4.19 shows the hyperparameters for the experiment.

Table 4.19: Hyperparameters for generating Image/mask with HA-GAN

Hyperparameter	**Values**
Training Steps	80000
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

4.6.1 Mask Interpolation

We use the cropped images with the same interpolation for the image as in the previous experiment. However, for this experiment, we apply a different interpolation for the mask. We interpolate the mask from the range $[0, 2]$ to the range $[-1, 1]$. We apply no individual resizing and concatenate the image and mask along the z-axis. We resize the concatenated image to size $128 \times 128 \times 128$.

The specific details of the experimental setup are summarized in Table 4.20.

Table 4.20: Experimental setup for Generating image/mask with HA-GAN - Mask Interpolation

Hyperparameter	**Values**
Model Architecture	HA-GAN $128 \times 128 \times 128$
Data	Cropped CT image/mask pairs
Image Size	$128 \times 128 \times 128$
Image Preprocessing	Interp. $[-1024/600]$, $[-1/1]$
Mask Preprocessing	Interp. $[0/2]$, $[-1/1]$
Concatenated Image Preprocessing	Resize
Concatenation Method	Z-axis

Table 4.21 shows the hyperparameters for the experiment.

Table 4.21: Hyperparameters for Generating Image/mask with HA-GAN - Mask Interpolation

Hyperparameter	**Values**
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

4.6.2 Slice-by-Slice Concatenation

We use the cropped images with the same interpolation for the image and mask as in the previous experiment. The image and mask are resized separately to $128 \times 128 \times 64$. Then the image and mask are concatenated slice by slice, as detailed in Chapter 3.4.3.

The specific details of the experimental setup are summarized in Table 4.22.

Table 4.22: Experimental Setup for Generating Image/mask with HA-GAN - Slice-By-Slice Concatenation

Hyperparameter	**Values**
Model Architecture	HA-GAN $128 \times 128 \times 128$
Data	Cropped CT image/mask pairs
Image Size	$128 \times 128 \times 128$
Image Preprocessing	Resize, Interp. $[-1024/600]$, $[-1/1]$
Mask Preprocessing	Resize, Interp. $[0/2]$, $[-1/1]$
Concatenated Image Preprocessing	None
Concatenation Method	Slice By Slice

Table 4.23 shows the hyperparameters for the experiment.

Table 4.23: Hyperparameters for Generating Image/mask with HA-GAN - Slice-By-Slice Concatenation

Hyperparameter	**Values**
Training Steps	80000
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

4.6.3 Modified Hounsfield Unit Interpolation Range

The cropped images are interpolated to $[-1, 1]$ from a different input low-high threshold range. We experiment first with the most common minimum and maximum values discovered in Table 3.4 & 3.3 in Chapter 3.2.

The specific details of the experimental setup are summarized in Table 4.24.

Table 4.24: Experimental Setup for Generating Image/mask with HA-GAN - Modified HU

Property	Values
Model Architecture	HA-GAN $128 \times 128 \times 128$
Data	Cropped CT image/mask pairs
Image Size	$128 \times 128 \times 128$
Image Preprocessing	Int. $[-2048, 3071]$, $[-1, 1]$
Mask Preprocessing	Int. $[0, 2]$, $[-1, 1]$
Concatenated Image Preprocessing	None
Concatenation Method	Slice By Slice

Table 4.25 shows the hyperparameters.

Table 4.25: Hyperparameters for Generating Image/mask with HA-GAN - Modified HU

Property	Values
Training Steps	80000
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

4.6.4 Hyperparameter Optimization

We use the cropped images. We run a grid search on 27 different configurations of learning rates for the optimizers in the generator, discriminator, and encoder. Table 4.26 shows the hyperparameters and their possible values.

Table 4.26: Hyperparameters and Possible Values

Hyperparameter	Values
Generator learning rate	0.0001, 0.0002, 0.0003
Discriminator learning rate	0.0002, 0.0003, 0.0004
Encoder learning rate	0.0001, 0.0002, 0.0003

We use the best-performing model from the learning rate exploration to explore more latent dimension sizes. We explore increasing the latent dimension with 25%, 50%, and 75%, resulting in 1280, 1536, and 1792 respectively.

4.6.5 Data Augmentation

We use the best-performing model from the hyperparameter exploration in the previous experiment. We resample the voxel size of the images and masks to $1.5 \times 1.5 \times 1.5mm$. After the resampling, the image and masks are interpolated and concatenated along the Z-axis. The concatenated image is resized to $128 \times 128 \times 128$.

The experimental setup is summarized in Table 4.27.

Table 4.27: Experimental Setup for Generating Image/mask with HA-GAN - Augmented Data

Property	Values
Model Architecture	HA-GAN $128 \times 128 \times 128$
Data	Cropped CT image/mask pairs
Image Size	$128 \times 128 \times 128$
Image Preprocessing	Int. $[-2048, 3071]$, $[-1, 1]$
Mask Preprocessing	Int. $[0, 2]$, $[-1, 1]$
Concatenated Image Preprocessing	Resize
Concatenation Method	Slice By Slice

Table 4.28 shows the hyperparameters.

Table 4.28: Hyperparameters for Generating Image/mask with HA-GAN - Augmented Data

Property	**Values**
Training Steps	80000
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

We create more training images by applying affine transformations and elastic deformations to the voxel-resampled data and train the HA-GAN for 80000 steps. The experimental setup and hyperparameters are the same as the previous experiment.

4.6.6 Binarization of Mask Values

To further optimize the results of the best-performing HPO model, we experiment with the preprocessing of the mask before feeding the images to the GAN. We preprocess the mask by binarizing the mask values. The voxels representing empty space with the value 0 are converted to -1 to match that of the image. We perform one more experiment with the binary mask. In the second experiment, we interpolate the binary mask with the input range of $[0, 2]$, like in Chapter 4.6.1.

Table 4.29 summarizes the experimental setup for the experiment.

Table 4.29: Experimental Setup for Generating Image/mask with HA-GAN - Binarization of Mask

Property	**Values**
Model Architecture	HA-GAN $128 \times 128 \times 128$
Data	Cropped CT image/mask pairs
Image Size	$128 \times 128 \times 128$
Image Preprocessing	Interp. $[-1024, 600]$
Mask Preprocessing	Binarize, Interp. $[0, 2]$
Concatenated Image Preprocessing	Resize
Concatenation Method	Z-Axis

Table 4.30 shows the hyperparameters used in the experiment.

Table 4.30: Hyperparameters for Generating Image/mask with HA-GAN - Binarization of Mask

Hyperparameter	**Values**
Training Steps	80000
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0002
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0002
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

4.6.7 High-Resolution Images

We use the best-performing HPO model (HPO17) to generate images of size $256 \times 256 \times 256$. We run experiments with and without the binarization of masks. We apply z-axis concatenation for all experiments.

Table 4.31 details the experimental setup.

Table 4.31: Experimental Setup for Generating High-Resolution Image/mask with HA-GAN

Property	**Values**
Model Architecture	HA-GAN $256 \times 256 \times 256$
Data	Cropped CT image/mask pairs
Image Size	$256 \times 256 \times 256$
Image Preprocessing	Interp. $[-1024, 600]$
Mask Preprocessing	Binarize, Interp. $[0, 2]$
Concatenated Image Preprocessing	Resize
Concatenation Method	Z-Axis

Table 4.32 shows the hyperparameters used in the experiment.

Table 4.32: Hyperparameters for Generating High-Resolution Image/mask with HA-GAN

Hyperparameter	**Values**
Training Steps	80000
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0002
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0002
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

We also run an experiment with the model with the original learning rates. The experiment uses the same setup as shown in Table 4.31. The hyperparameters used for the experiment are shown in Figure 4.33

Table 4.33: Hyperparameters for Generating Image/mask with HA-GAN - Original LR

Hyperparameter	**Values**
Training Steps	80000
Latent Dimension Size	1024
Batch Size	4
Generator Learning Rate	0.0001
Discriminator Learning Rate	0.0004
Encoder Learning Rate	0.0001
Optimizer	ADAM
Optimizer β_1	0
Optimizer β_2	0.99
Optimizer ϵ	0.00000001

Finally, we run one experiment with input data voxel resampled to $1.5 \times 1.5 \times 1.5mm$. The experimental setup is the same as Table 4.31, and the hyperparameters are the same as Table 4.32.

4.7 Visual Turing Test

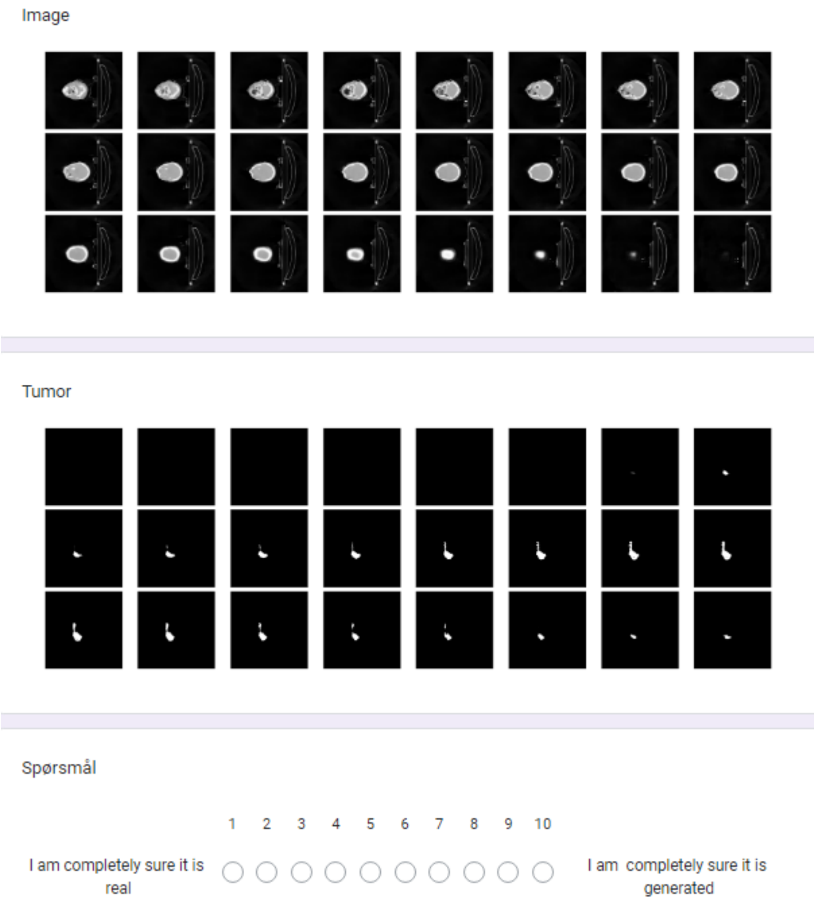
The objective of this experiment was to evaluate the ability of healthcare professionals to distinguish between real and generated images.

The participants consisted of three healthcare professionals, a radiologist with three years of experience, and two medical doctors with 10 and 15 years of experience, respectively.

The HA-GAN, trained on binary masks, was used to generate five images of size $256 \times 256 \times 128$. We selected five real images from various data sources, ensuring diversity in the images used.

For the purposes of the test, each image, both real and generated, was divided into three equal parts, with one part from each image used in the experiment. This process was undertaken to ensure that only a 30% of each image was presented to the participants, thus making the images larger and more visually easier to interpret for the participants.

The images were randomly ordered and presented to the participants in an online form. Each image was accompanied by a corresponding tumor mask. The participants were not informed about the origin of the images, and their task was to rate each image/mask pair on a scale from 0 to 10. A score of 0 indicated a belief that the image was completely real, while a score of 10 signified a belief that the image was entirely artificially generated. An example is shown in Figure 4.2.



Image

Tumor

Spørsmål

	1	2	3	4	5	6	7	8	9	10	
I am completely sure it is real	○	○	○	○	○	○	○	○	○	○	I am completely sure it is generated

Figure 4.2: Visual Turing Test

Table 4.34 presents the labels of the images used in the experiment.

Table 4.34: Images used in VTT

Number	Label
1	Generated
2	Generated
3	Real
4	Real
5	Generated
6	Real
7	Real
8	Real
9	Generated
10	Generated

This experiment allowed us to evaluate the ability of healthcare professionals to distinguish between real and generated images, thereby effectively serving as a Visual Turing Test for the HA-GAN model.

4.8 Segmentation

4.8.1 Segmentation Baseline

To establish a baseline for segmentation performance, we use only the real CT image/mask pairs as training data for the segmentation network. We use the Auto3Dseg framework from MONAI (Cardoso et al., 2022) with the SegResNet (Myronenko, 2019) segmentation model.

We use the cropped image/mask pairs described in Chapter 3.4.3. The images are resampled to a $1.5 \times 1.5 \times 1.5mm$ isotropic resolution, following a similar approach to that of (Myronenko et al., 2023). The voxel values in the masks are binarized. The images and masks are resized to size $128 \times 128 \times 64$ to match the format of the generated images to be segmented in later experiments.

We set aside 80% (419) of the training samples to be used for testing later. Using 5-Fold Cross-Validation as described in Chapter 3.10.3, we split the 419 training samples into five configurations of 332 samples for training and 87 for validation.

We begin by training the SegResNet on a single CV fold of training data for 300 epochs. Then we train five models on five different folds of training and validation data for 300 epochs. The details of both experiments are summarized in Table 4.35.

Table 4.35: Experimental Setup for Baseline Segmentation

Property	**Values**
Model Architecture	SegResNet
Data	Real, cropped images
Image Size	$128 \times 128 \times 64$
Image Preprocessing	Resample $1.5 \times 1.5 \times 1.5mm$
Mask Preprocessing	Binarize, Resample $1.5 \times 1.5 \times 1.5mm$
Training Images	419
Validation Images	105

The hyperparameters used for the experiment are shown in Figure 4.36.

Table 4.36: Hyperparameters for Saseline Segmentation

Hyperparameter	**Values**
Epochs	300
Number of Cross-Validation Folds	1 and 5
Optimizer	ADAMW
Optimizer learning rate	0.0002
Optimizer weight decay	0.000010

By running these experiments, we establish a baseline validation score for the SegResNet model on real images. After establishing a training baseline, we continue to find the baseline for the test performance of the model after training. After the training is complete, we run inference on the trained model. We feed the model with the 105 testing images that were set aside earlier. The output of the network running in inference mode is predicted masks on all testing images. We use the predicted masks to compare them with the real images. We calculate Dice Score Coefficient as described in Chapter 2.3.3.

One limitation of this experiment is the lack of a baseline established for image size $256 \times 256 \times 128$, as we focused solely on the image size $128 \times 128 \times 64$. The reason for this omission is the significant computational time required for larger image sizes, which, unfortunately, we were unable to accommodate within the timeframe of the experiment.

4.8.2 Training on Real and Generated Images with 5-Fold Cross-Validation

We use all real images from the previous experiment, including the test data, as a single dataset of 524 real images. We split the 524 real images into five folds, as described in Chapter 3.10.3. The resulting dataset is five configurations of different training samples (419) and validation samples (105).

We use the best-performing HA-GAN model to generate 524 synthetic images. We post-process the generated images, as described in Chapter 3.8.2. Like the real dataset, we split the generated images into five folds with the same amount of training and validation samples.

We then create a combined dataset consisting of 50% generated and 50% real images. The final dataset size is 1048 image/mask pairs, with 524 real and 524 generated images. Of the 1048 images, 838 are used for training, and 210 are used for validation. The real and generated images are evenly spread out in each fold.

In summary, we prepare three datasets for training the SegResNet model:

1. Dataset 1: 524 real images
2. Dataset 2: 524 generated images
3. Dataset 3: 524 real and 524 generated images

We use the SegResNet in three experiments to train on each dataset, using 5-Fold CV, resulting in 15 models to train. We train each model for 300 epochs. We repeat the experiments for both images sizes $128 \times 128 \times 64$ and $256 \times 256 \times 128$, resulting in a total of 30 models. Because we are not using testing data, we are using the validation scores as an indicative performance metric.

The details of the experimental setup are summarized in Table 4.37.

Table 4.37: Experimental Setup for Segmentation on Real and Generated Images

Property	**Values**
Model Architecture	SegResNet
Data	Real & generated images
Generated Image Source	HPO17 HA-GAN
Image Size	$128 \times 128 \times 64$ and $256 \times 256 \times 128$
Image Preprocessing	None
Mask Preprocessing	Binarize
Training Images	419 and 838
Validation Images	105 and 210

The hyperparameters used for the experiments are shown in Figure 4.38.

Table 4.38: Hyperparameters for Segmentation on Real and Generated Images

Hyperparameter	**Values**
Epochs	300
Number of Cross-Validation Folds	5
Optimizer	ADAMW
Optimizer learning rate	0.0002
Optimizer weight decay	0.000010

4.8.3 Training on Generated Images, Testing on Real Images

Instead of training the segmentation model with 5-Fold CV, we run multiple experiments to train the SegResNet model once on a single CV fold of the generated images. After training, we run the network in inference mode and create prediction masks on the real images.

We run three experiments where we train the segmentation model on 524, 1024, and 2048 fake images. After each training, we test the model on all 524 real images. We repeat the experiment for both image size $128 \times 128 \times 64$ and $256 \times 256 \times 128$, resulting in six models to train.

The details of the experimental setup are summarized in Table 4.39.

Table 4.39: Experimental setup for segmentation of generated images

Property	**Values**
Model Architecture	SegResNet
Data	Generated Images
Generated Image Source	HPO17 HA-GAN
Image Size	$128 \times 128 \times 64$, $256 \times 256 \times 128$
Image Preprocessing	Resample $1, 5, 1.5, 1.5mm$
Mask Preprocessing	Binarize, Resample $1, 5, 1.5, 1.5mm$
Training Images	419, 819, 1638
Validation Images	105, 204, 409
Testing Images	524

The hyperparameters used for the experiments are shown in Figure 4.40.

Table 4.40: Hyperparameters for segmentation of generated images

Hyperparameter	**Values**
Epochs	300
Number of Cross-Validation Folds	5
Optimizer	ADAMW
Optimizer learning rate	0.0002
Optimizer weight decay	0.000010

4.8.4 Training on Real and Generated Images, Testing on Real Images

We split the 524 real images into two parts of 419 samples for training and 105 for testing, like in Chapter 4.8.1. We generate 524 samples with the HPO17 HA-GAN model. We create a combined dataset consisting of the generated and real images. The final dataset size is 943 samples, with 419 real and 524 generated images. 752 samples are used for training, and 191 samples are used for validation.

We run two experiments, one with a single CV fold and one with five folds.

The details of the experimental setup are summarized in Table 4.41.

Table 4.41: Experimental setup for segmentation on real and generated images with testing on real images

Hyperparameter	**Values**
Model Architecture	SegResNet
Data	Real & Generated Images
Generated Image Source	HPO17 HA-GAN
Image Size	$128 \times 128 \times 64$
Image Preprocessing	Resample $1.5 \times 1.5 \times 1.5mm$
Mask Preprocessing	Binarize, Resample $1.5 \times 1.5 \times 1.5mm$
Training Images	752
Validation Images	191
Testing Images	105

The hyperparameters used for the experiments are shown in Table **??**

Table 4.42: Hyperparameters for segmentation on real and generated images with testing on real images

Hyperparameter	**Values**
Epochs	300
Number of Cross-Validation Folds	1 and 5
Optimizer	ADAMW
Optimizer learning rate	0.0002
Optimizer weight decay	0.000010

After the training is complete, we run inference on the model with the 105 real images, like in earlier experiments.

We perform another experiment using a different HA-GAN trained model. We use the model trained on binary masks to generate 524 images. We post-process the images with thresholding rescaling. We combine the generated images with 419 real images, putting the remaining 105 real images aside for testing. The training dataset consists of 943 images, where 752 are for training, and 191 are for validation. We train the SegResNet model for 300 epochs using single-fold CV training. Then we perform a final experiment with the same setup, with five folds.

The details of the experimental setup are summarized in Table 4.43.

Table 4.43: Experimental setup for segmentation on real and generated images with testing on real images

Hyperparameter	**Values**
Model Architecture	SegResNet
Data	Real & Generated Images
Generated Image Source	Binary Trained HA-GAN
Image Size	$128 \times 128 \times 64$
Image Preprocessing	Resample $1.5 \times 1.5 \times 1.5mm$
Mask Preprocessing	Binarization, Resample $1.5 \times 1.5 \times 1.5mm$
Training Images	752
Validation Images	191
Testing Images	105

The hyperparameters used for the experiments are shown in Table 4.44

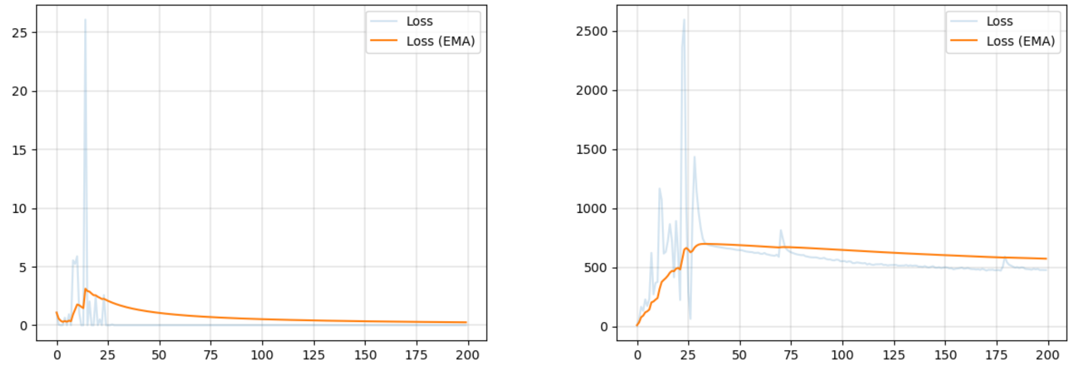
Table 4.44: Hyperparameters for segmentation on real and generated images with testing on real images

Hyperparameter	**Values**
Epochs	300
Number of Folds	1, 5
Optimizer	ADAMW
Optimizer learning rate	0.0002
Optimizer weight decay	0.000010

5 Results and Discussion

5.1 Vanilla GAN

After encountering memory issues with high-resolution images, we successfully make the Vanilla GAN 1 model train on $16 \times 16 \times 3$ images for 200 epochs. Figure 5.1 shows the generator and discriminator loss during training.



(a) Discriminator

(b) Generator

Figure 5.1: Vanilla GAN 1 Generator and Discriminator loss (blue), EMA loss (orange)

Table 5.1 shows the overall loss statistic for Vanilla GAN 1.

Table 5.1: Vanilla GAN 1 Minimum, maximum, and average loss for generator and discriminator

Loss	**Minimum**	**Maximum**	**Average**
Generator	8.5851	2596.0562	575.9452
Discriminator	0.0005	26.0948	0.2705

Based on the data presented in Figure 5.1, it can be observed that the discriminator experiences a plateau phase after epoch 20, followed by a gradual decline in its loss until it reaches a low level, suggesting that is is capable of successfully classifying images. On the other hand, the generator loss stabilizes after epoch 25 and exhibits a decreasing trend, but it is noteworthy that the loss remains considerably high compared to the discriminator. The high loss would suggest that the generator is not able to generate images that are of quality. The data in Table 5.1 indicates that the generator's average loss is higher than 500, which suggests that the neural network is struggling to create synthetic images that are not discernible by the discriminator.

Example images generated by the Vanilla GAN from epoch 300 are shown in Figure 5.2.

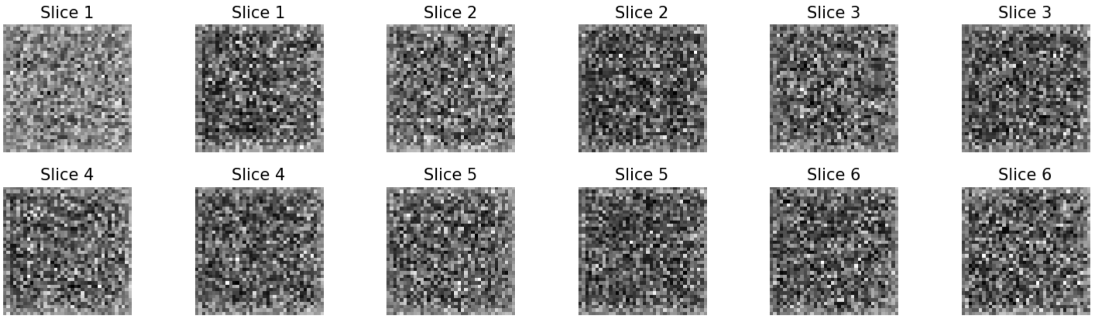
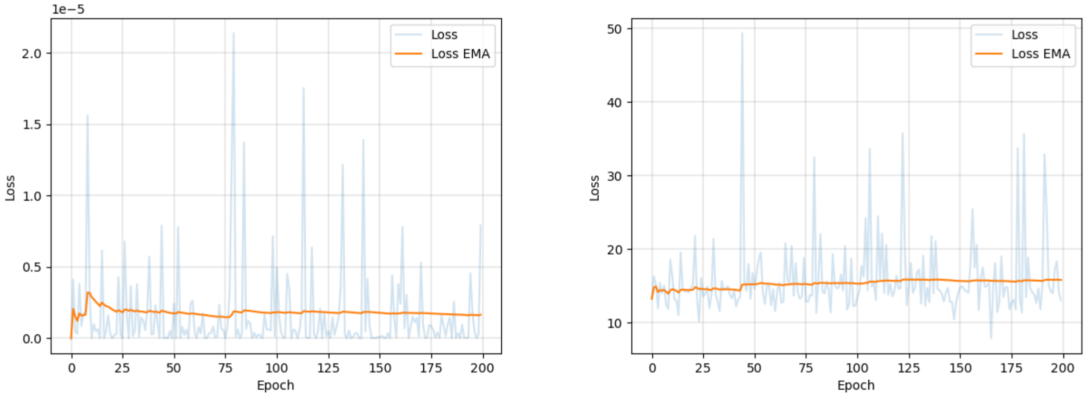


Figure 5.2: Vanilla GAN 1 Generated Images

It is evident from the observed results in Figure 5.2 that the generated images lack coherence and appear to be random noise, suggesting that the generator has not effectively learned the underlying data distribution. Furthermore, all the generated images are identical, indicating that the generator suffers from mode collapse.

This could potentially be attributed to the limited complexity of the Vanilla GAN architecture, which may not be adequate to model the intricacies of the data distribution at low image resolutions. It is plausible that the resampling of the images at such low resolutions may also be causing distortions that hinder the ability of the model to effectively capture the underlying patterns in the data.

Vanilla GAN 2 is a modified Vanilla GAN 1 model with a smaller, less complex network. Additionally, the images used for training were preprocessed to be of size $154 \times 154 \times 27$. The loss diagrams of Vanilla GAN 2 are shown in Figure 5.3.



(a) Discriminator

(b) Generator

Figure 5.3: Vanilla GAN 2 Generator and Discriminator loss (blue), EMA loss (orange)

Table 5.2 shows the overall loss statistics for Vanilla GAN 1 & 2.

Table 5.2: Vanilla GAN 1 & 2 Minimum, maximum and average loss for generator and discriminator

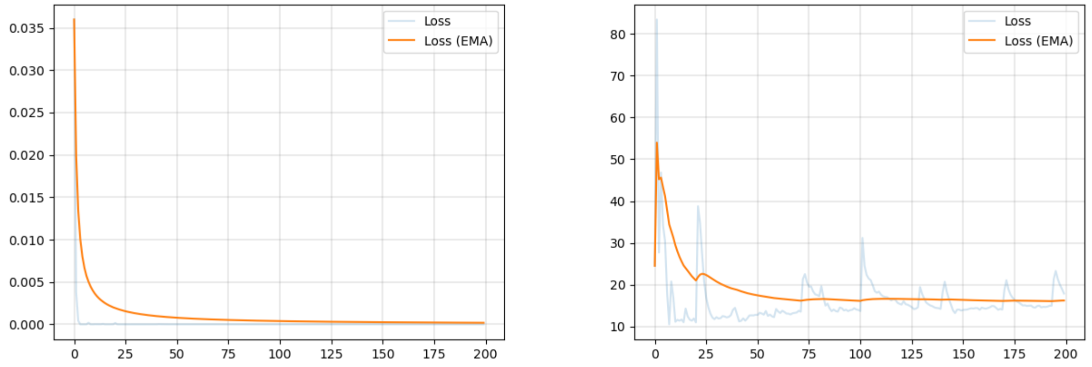
Loss	**Minimum**	**Maximum**	**Average**
Vanilla 1 Generator	8.5851	2596.0562	575.9452
Vanilla 2 Generator	5.0499	49.4817	**16.1331**
Vanilla 1 Discriminator	0.0005	26.0948	0.2705
Vanilla 2 Discriminator	0.0000	2.1432	**0.0020**

According to Table 5.2, the Vanilla GAN 2 model outperforms its predecessor significantly. While the minimum loss is relatively the same, both the maximum and average losses exhibit significant improvement. The discriminator in Vanilla GAN 2 also performs much better, achieving a plateau more quickly and showing an overall lower average loss. However, as seen in Figure 5.3b, the generator loss is increasing over time, showing that it is not effectively learning the data distribution. Even so, the overall loss is improved over the Vanilla 1 GAN generator. The discriminator loss seen in Figure 5.3a, is showing an overall lower loss over time, with the EMA loss never going over 0.5, a big improvement over Vanilla GAN 1.

Despite these improvements, the generated images from Vanilla GAN 2 still contain noise, and the generator eventually experiences a mode collapse. Upon reflection, it appears that the Vanilla GAN model with a U-Net architecture may not be sufficiently complex to capture the characteristics of CT images, even at a low resolution of $154 \times 154 \times 27$. However, this model was a good initial approach that needed to be tested.

5.2 Wasserstein GAN

Unfortunately, WGAN-GP encountered memory problems, and we decided not to experiment with it further. We trained the WGAN with weight clipping for 300 epochs. Figure 5.4 shows the generator and discriminator loss for WGAN-WC during training.



(a) Discriminator

(b) Generator

Figure 5.4: WGAN-WC Generator and Discriminator loss (blue), EMA loss (orange)

Table 5.3 show the overall minimum, maximum and average of the generator and discriminator loss for the WGAN-WC.

Table 5.3: WGAN-WC Minimum, maximum and average loss for generator and discriminator

Loss	Minimum	Maximum	Average
Generator	10.5033	83.4498	16.2467
Discriminator	0.0002	9.4511	0.0359

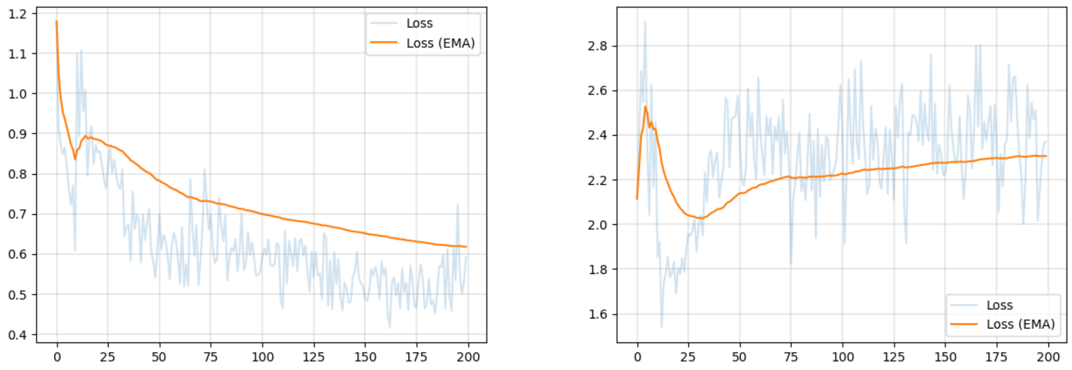
The results show that the Wasserstein-WC has slightly worse results than the Vanilla GAN 2 with the same architecture, suggesting that weight clipping does not improves the model's performance. Moreover, the WGAN-WC appears to have more stable training with fewer outliers in the results. It is worth noting that the discriminator performs well early on in the training process, but the generator is not complex enough to effectively learn features from the latent space. Although a more complex generator could potentially improve performance, this may not be feasible due to memory limitations.

The generator eventually suffers from mode collapse, leading to the production of similar and low-quality images, similar to the Vanilla GAN. Despite the use of hyperparameter optimization, all 18 configurations eventually reach mode collapse, indicating that the underlying data distribution is not being effectively learned by the model.

Overall, these findings highlight the need for a different approach to effectively model the underlying data distribution and generate high-quality images.

5.3 FastGAN and StyleGAN2

We trained the FastGAN model on 2D images for 500 epochs. Figure 5.5 shows the generator and discriminator loss for the training of FastGAN.



(a) Discriminator

(b) Generator

Figure 5.5: FastGAN Generator and Discriminator loss (blue), EMA loss (orange)

Table 5.4 shows the overall loss statistics for the FastGAN training.

Table 5.4: Minimum, maximum, and average loss for FastGAN generator and discriminator

Loss	**Minimum**	**Maximum**	**Average**
Generator	1.5395	2.9068	2.3001
Discriminator	0.4177	7.0231	0.6239

The FastGAN generator appears to show an overall better performance, according to the table. The average generator loss is only 2.3, a large improvement over the Vanilla 2 GAN generator loss of 16.1.

As observed in Figure 5.5, the generator loss is steadily increasing over time. It is worth noting that the loss is relatively low, suggesting that the generator is able to learn more effectively than the Vanilla and WGAN approaches. As seen in Figure 5.7a, the discriminator loss is steadily decreasing over time, never plateuing, suggesting that the performance would increase given more training time. Because the generator loss is increasing, it can be argued that given more time, the loss would increase, further dampening the performance.

Example images generated with FastGAN are shown in Figure 5.6.

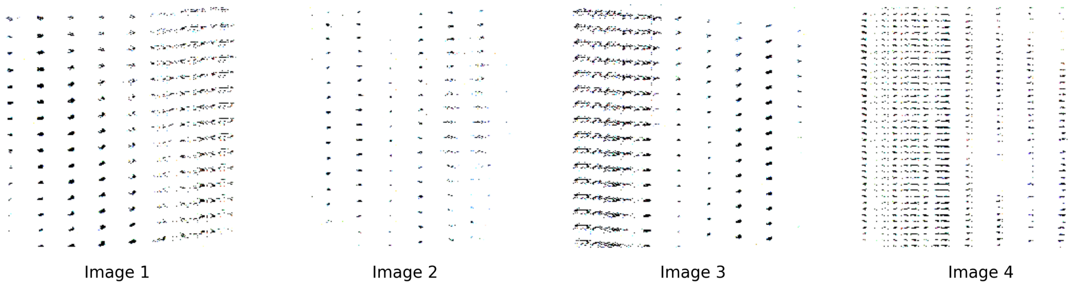
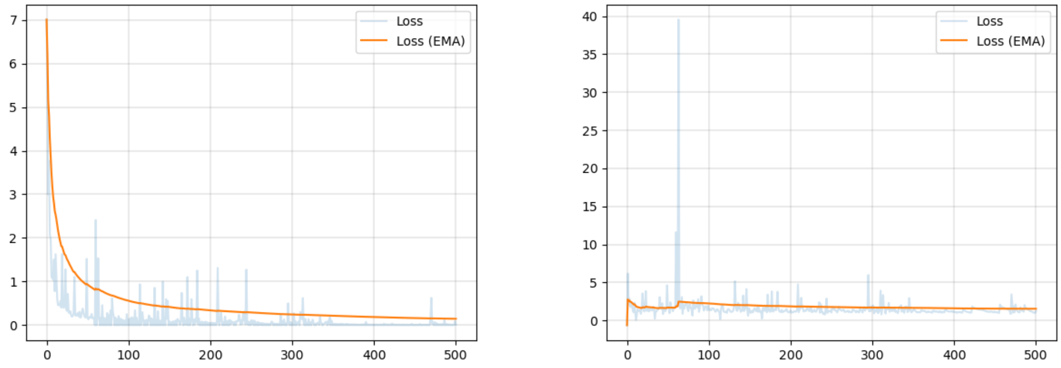


Figure 5.6: Images generated with FastGAN

While the generated images are somewhat visually similar to the input images, they are of low resolution and overall poor quality. Additionally, many randomly generated images appear visually similiar, showing that the generator is experiencing a partly mode collapse. Given more time, it can be argued that the generator would encounter a full mode collapse, seeing as the loss is increasing over time as seen in Figure 5.7b.

Figure 5.7 shows the generator and discriminator loss for the training of StyleGAN2.



(a) Discriminator

(b) Generator

Figure 5.7: StyleGAN2 Generator and Discriminator loss (blue), EMA loss (orange)

The figure illustrates that the discriminator loss consistently decreases during training, converging to a relatively low value, with only a few outliers observed. This indicates a stable training process for the discriminator. On the other hand, the generator exhibits an even more stable training, with fewer outliers overall, except for a single significant outlier.

Table 5.5 shows the overall loss statistics for the StyleGAN2 training.

Table 5.5: Minimum, maximum and average loss for StyleGAN 2 generator and discriminator

Loss	**Minimum**	**Maximum**	**Average**
Generator	0.0	39.5479	1.5886
Discriminator	0.0	7.0043	0.1734

The most noticeable aspect in Figure 5.7 is the maximum loss reached by the generator, which stands out with a value of 39, representing the observed outlier. However, disregarding the outliers, the average generator loss is even lower than that of FastGAN, with an average of only 1.5 compared to FastGAN's average of 2.3. Similarly, the discriminator also exhibits a higher maximum loss compared to FastGAN, but the average loss is significantly lower.

Example images generated with StyleGAN2 are shown in Figure 5.8.

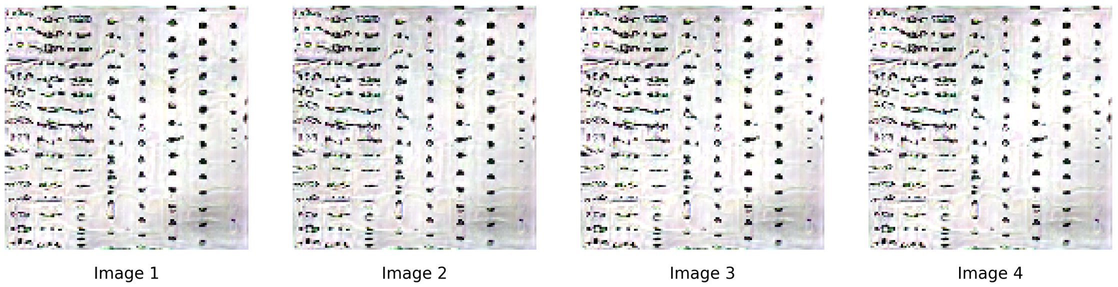


Figure 5.8: Images generated with StyleGAN2

It can be observed that the StyleGAN2 model is capable of capturing some of the underlying features of the data distribution, even at a relatively low image size of

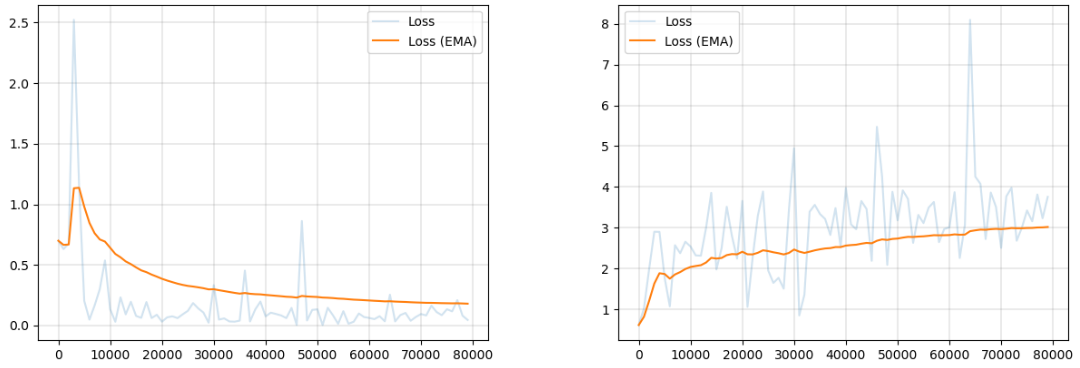
128, a latent size of 512, and 32 channels. However, like the Vanilla GAN and WGAN models, the generator eventually suffers from mode collapse, once again proving that the average loss is not enough to quantify actual performance.

The potential for improved results may be achieved through experimentation involving varying image sizes, increased channels, alternative latent dimensions, and hyperparameter optimization. These modifications could potentially mitigate mode collapse. However, due to time constraints, further exploration of these possibilities was not feasible.

Additionally, a considerable challenge lies in post-processing the generated 2D images back into 3D representations. This conversion introduces an additional layer of complexity to the problem, making it more challenging to achieve satisfactory results. The exploration of different image sizes, channels, latent dimensions, and HPO may yield improved results of StyleGAN2 and prevent mode collapse. However, the decision to prioritize a more efficient 3D approach was deemed as most advantageous given the time constraints.

5.4 HA-GAN Image Generation

We train the HA-GAN $128 \times 128 \times 128$ model multiple times for 80000 steps on training images without data shuffling. The generator and discriminator loss for one training session are shown in Figure 5.9.



(a) Discriminator

(b) Generator

Figure 5.9: HA-GAN (without data shuffling) Generator and Discriminator loss (blue), EMA loss (orange)

As seen in Figure 5.11b the generator loss steadily increases over time as the generator cannot effectively learn the data and generate images that can fool the discriminator. It can be argued that not shuffling the data during training leads to mode collapse. This is because the data in the dataset is ordered by data source. The discriminator becomes exposed to a repetitive set of similar images from the same data source, becoming too effective at distinguishing them from the generator's output. Considering that the training images are obtained from 7 different data sources, each with its own unique characteristics, it can be argued that presenting them in order could result in an imbalanced training set. The generator may be effective early on at generating a certain set of images and ultimately fail later on when different data is introduced. This could result in the generator struggling to produce diverse images that can fool the discriminator, ultimately leading to mode collapse.

Figure 5.10 shows five randomly generated images. We plot the center slice along the z-axis for each image.

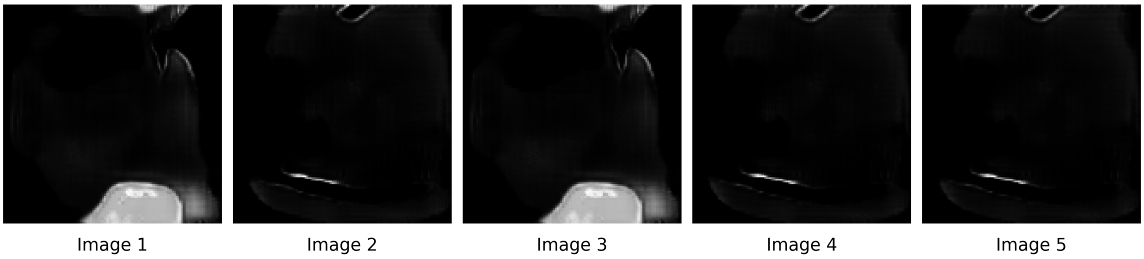
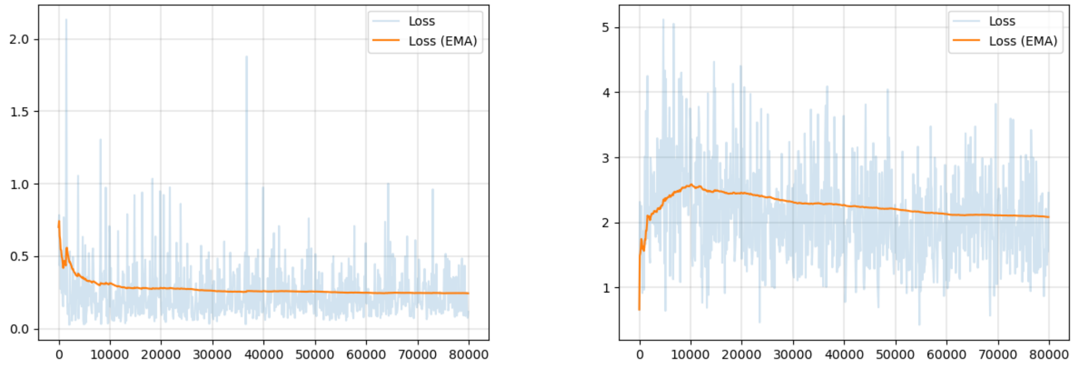


Figure 5.10: Images Generated with HA-GAN

As seen in the Figure, the mode collapse is apparent with the generator only generating a set of distinct images which are identical.

We enable shuffling of the training data and observe that the HA-GAN is able to generate CT images of size $128 \times 128 \times 128$. The generator and discriminator loss are shown in Figure 5.11.



(a) Discriminator

(b) Generator

Figure 5.11: HA-GAN (with data shuffling) Generator and Discriminator loss (blue), EMA loss (orange)

We observe improved results with the shuffling of the training data. The generator loss is gradually decreasing and converging around 2.1 in the final stages of training. We argue that the observed improvement is due to shuffling the training data, which can help ensure that the generator is exposed to a diverse set of images from all data sources throughout the training, helping to prevent mode collapse and improving the quality of the generated images.

We calculate FID and IS scores for the generated CT images. Table 5.6 presents the scores, as well as the scores for the models trained on images that were voxel-resampled.

Table 5.6: Comparison of FID and IS scores

Experiment	**FID**	**IS**
Original Images	0.35605	1.05953
Resampled $1.0 \times 1.0 \times 1.0mm$	0.36545	1.04831
Resampled $1.5 \times 1.5 \times 1.5mm$	**0.34856**	1.04794
Resampled $2.0 \times 2.0 \times 2.0mm$	0.40140	1.04308
Resampled $2.5 \times 2.5 \times 2.5mm$	0.36616	1.05351
Resampled $3.0 \times 3.0 \times 3.0mm$	0.35480	**1.06072**

Notably, the best IS score was achieved on data voxel-resampled with a $3.0 \times 3.0 \times 3.0mm$ resolution, while the best KID score was obtained with data resampled with $1.5 \times 1.5 \times 1.5mm$, as seen in Table 5.6.

Overall, it is evident from the results that the KID and IS scores obtained in our experiments are of less quality than those reported in the original HA-GAN paper by (Sun et al., 2022). This could potentially be attributed to the fact that we trained our model on a smaller dataset of 524 images, whereas the original paper utilized a CT dataset of 9,276 images.

Figure 5.13 shows five slices from a single CT image generated by the HA-GAN model trained on the original images.

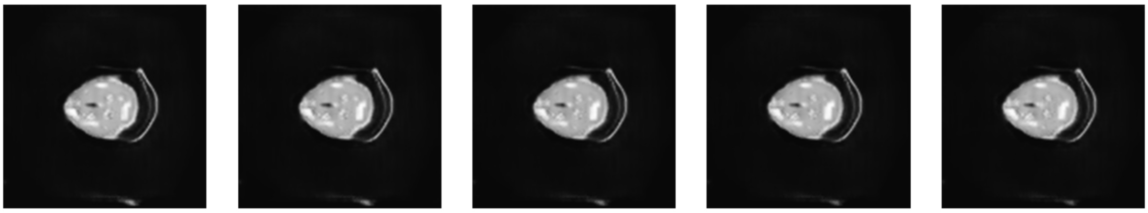


Figure 5.12: Slices from a image generated with HA-GAN

Figure 5.13 shows a full example of a CT image generated by the HA-GAN model trained on the original images.

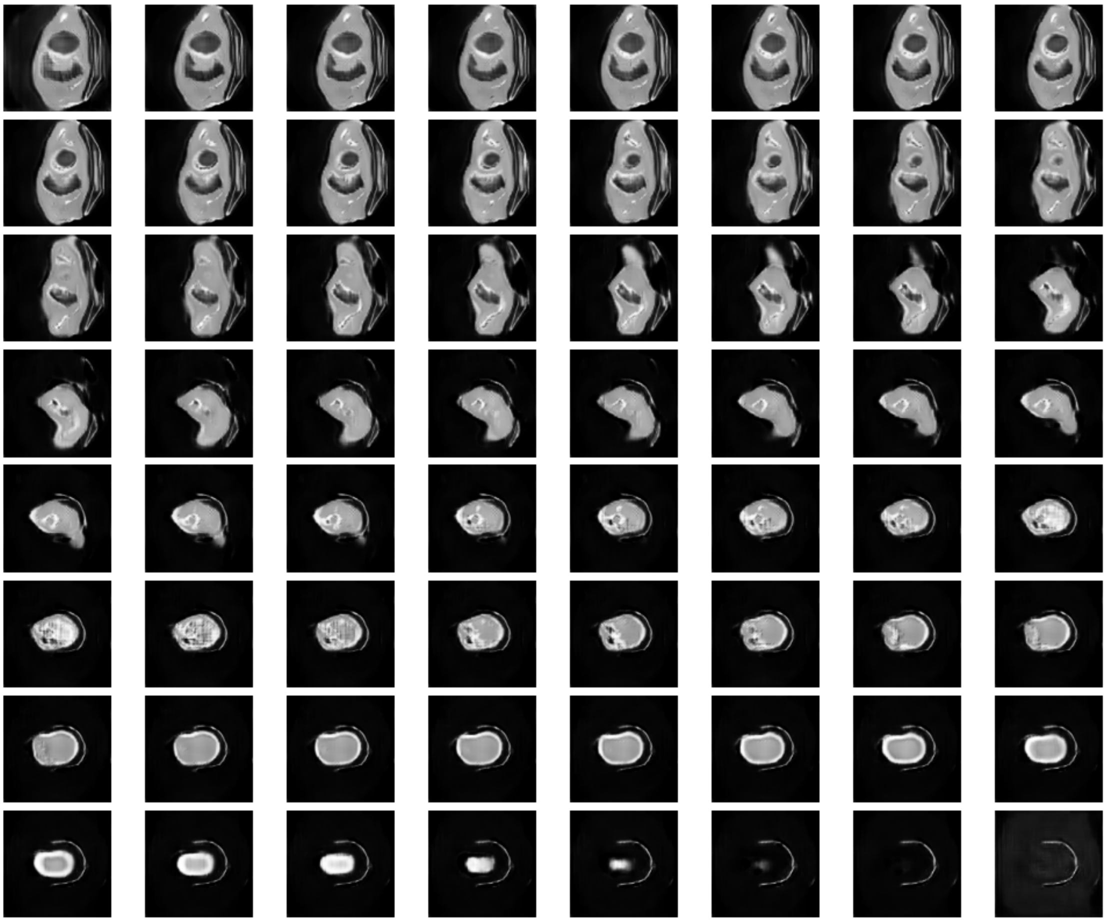
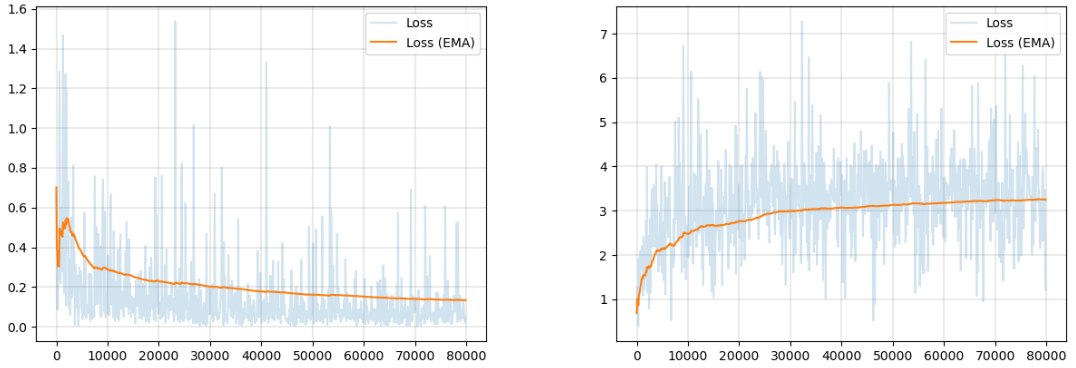


Figure 5.13: Full generated image with HA-GAN

5.5 HA-GAN Image/Mask Pair Generation

5.5.1 Baseline

We successfully generate $128 \times 128 \times 128$ image/mask pairs with HA-GAN. After separating the images and masks, they are of size $128 \times 128 \times 64$. Figure 5.14 shows the generator and discriminator loss during training.



(a) Discriminator

(b) Generator

Figure 5.14: HA-GAN Generator and Discriminator loss (blue), EMA loss (orange)

The loss trend observed in Figure 5.14a for the discriminator is similar to that of the image-only generation. The initial loss is approximately 0.6, which subsequently increases before converging at around 0.2 after 10000 steps for the image-only experiment. In contrast, as depicted in Figure 5.11a, the discriminator requires more time to converge, and its loss continually decreases throughout the 80000-step training period without plateauing. Although the loss trend indicates that the loss would likely continue to decrease after 80000 steps, we elected to conclude training at the maximum step to facilitate other experiments due to the time-consuming nature of training the model. Thus, one could argue that further training might result in improved performance.

Table 5.7: Baseline Image/Mask Pair Performance

Data	**FID**	**IS**
Images	0.14432	1.03026
Masks	0.00024	1.00012

Interpolating masks with a low-high threshold range of $[-1024, 600]$ yielded poor results, even causing mode collapse. Interpolation with the low-high threshold $[0, 2]$ was the deciding factor in avoiding mode collapse and successfully generating masks along with the images. Looking back, it is clear that interpolating the masks with the same threshold range as the images was not a good approach. This is because the masks consist of voxel values of 0, 1, or 2, which means that using an interpolation range of $[-1024, 600]$ would lead to suboptimal results.

As seen in Table 5.7, the model trained on the cropped images performs better than the model trained on the original images, even when generating both image and mask simultaneously.

Figure 5.15 shows a generated image/mask pair.

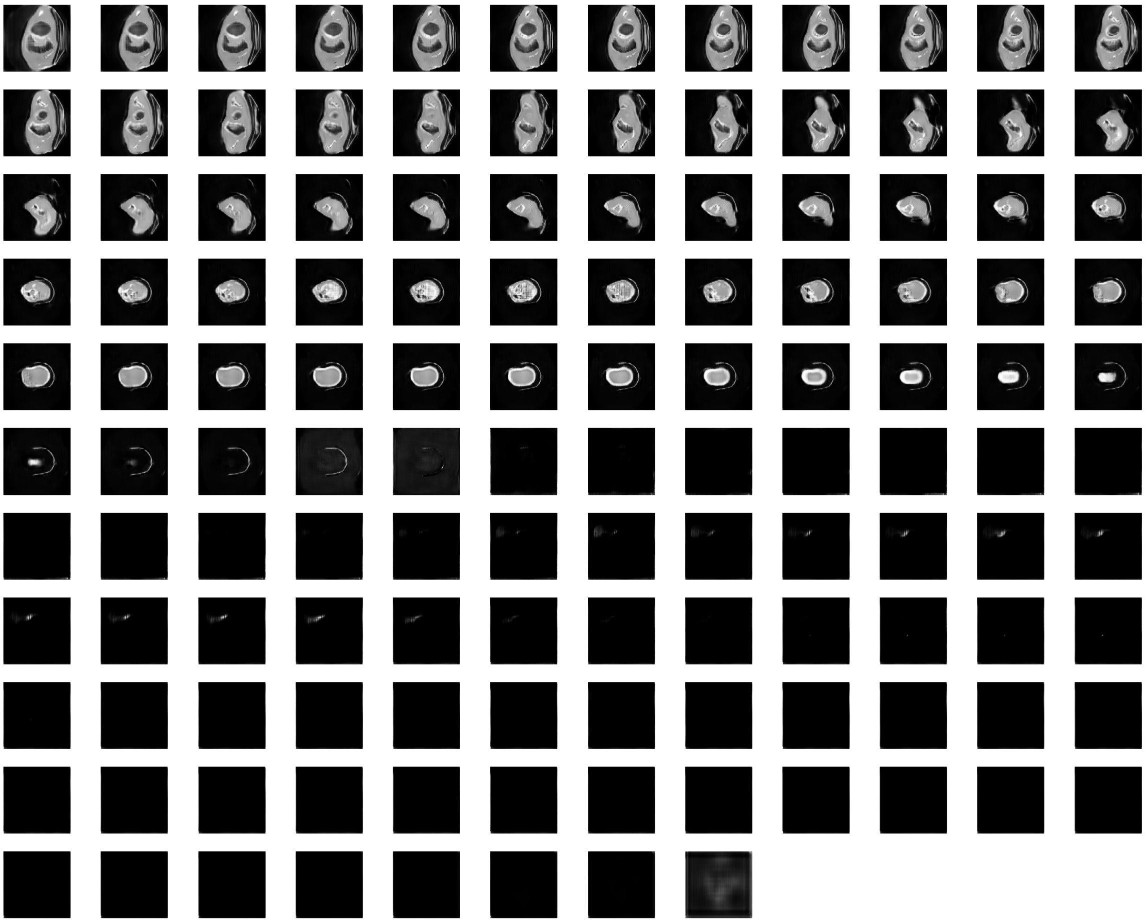


Figure 5.15: Image/mask pair generated with HA-GAN

Visually the pair looks realistic, like the original data. The generated CT image does have characteristics of the training data and does validate what the low FID score implies for the images.

We apply t-SNE analysis to the real images and the images generated by the HA-GAN. Figure 5.16 show the results.

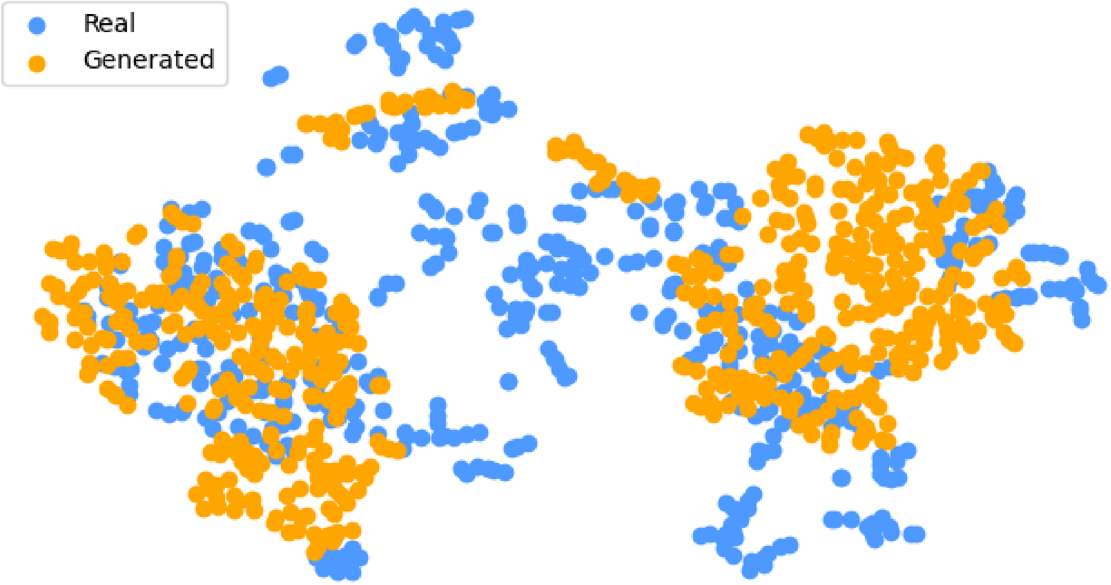
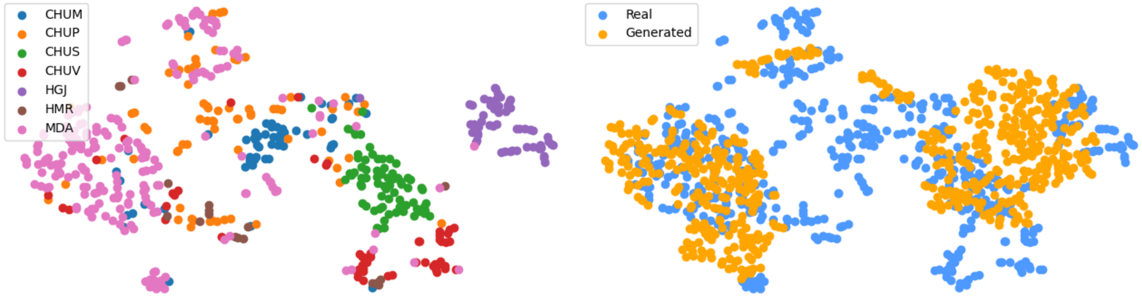


Figure 5.16: t-SNE comparison between real and generated images

Looking at Figure 5.16, clear indications emerge that the HA-GAN model has successfully learned the fundamental attributes present in the real images. However, it is apparent that the model has not comprehensively grasped the entire distribution but has primarily focused on two regions within the latent dimension space. While the left cluster predominately represents the real latent space, the rightmost cluster does not. Notably, two outliers can be observed in the upper middle section, although their influence is limited to a small portion of the latent space. In general, the latent space appears unexplored, and the model has not fully captured the inherent features of the dataset. We compare the original t-SNE analysis of the grouped real data with the t-SNE analysis of the generated images, shown in Figure 5.17



(a) Real data

(b) Real and generated data

Figure 5.17: HA-GAN Generator and Discriminator loss (blue), EMA loss (orange)

The shortcomings of the HA-GAN model in learning the data originating from the CHUV, CHUP, and CHUM data sources become more apparent when examining Figure 5.17. It is evident that the model has not successfully acquired a comprehensive understanding of the latent features present in these specific datasets. This suggests that the HA-GAN model requires further improvement to better learn and represent the data from these particular data sources.

5.5.2 Slice by Slice Concatenation

We generate image/mask pairs with the slice-by-slice concatenation method. The FID and IS results are shown in Table 5.8.

Table 5.8: SBS Image/Mask Pair Performance

Data	**FID**	**IS**
Images	50.16954	1.0
Masks	51.87453	1.0

As the IS scores of 1.0 suggest, the generator has fallen into mode collapse, producing only a single identical image every time it is run in inference. The FID score is fifty times that of the baseline, confirming that the model is not producing good results. Figure 5.18 shows ten slices from a generated mask by the model.

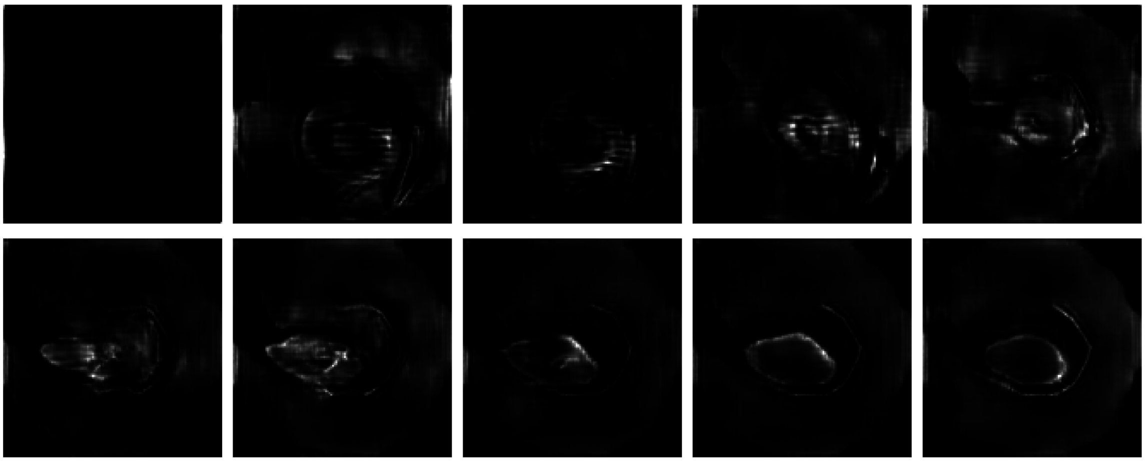


Figure 5.18: Example slices from generated mask

As seen in the figure, the mask does not visually look like the training data. There are a lot of artifacts on the image, which manifest as noise and do appear like the CT image and not the mask.

An argument can be made that HA-GAN may not be appropriate for generating concatenated pairs that are constructed slice by slice, as the model generates sub-volumes during its training. While selecting sub-volumes, the model may choose sub-volumes that contain both the image and mask, leading to confusion. Furthermore, the authors of X have reported improved performance by eliminating blank slices from the images before training. In the case of masks, the slices without tumor values are considered blank, and the model may struggle to learn this during training.

The results confirm this as the model is going into a full mode collapse, generating only one single image.

5.5.3 Hyperparameter Optimization

We ran 27 configurations of different learning rates for the generator, discriminator and encoder networks in the HA-GAN model. Table 5.9 show the results for all runs.

Table 5.9: Image and mask quality metrics for each HPO

Name	**Image FID**	**Mask FID**	**Image IS**	**Mask IS**
2^{D} 1^{E} 1^{G}	0.00498	0.00004	1.01617	1.00018
2^{D} 2^{E} 1^{G}	0.00369	0.00002	1.02246	1.00006
2^{D} 3^{E} 1^{G}	0.00344	0.00036	**1.03194**	**1.00058**
3^{D} 1^{E} 1^{G}	0.00977	0.00004	1.01457	1.00014
3^{D} 2^{E} 1^{G}	0.00554	0.00002	1.01524	1.00016
3^{D} 3^{E} 1^{G}	0.00360	0.00001	1.02313	1.00005
4^{D} 1^{E} 1^{G}	0.03305	0.00002	1.01179	1.00005
4^{D} 2^{E} 1^{G}	0.00472	0.00002	1.01395	1.00015
4^{D} 3^{E} 1^{G}	0.04294	0.00006	1.01172	1.00003
2^{D} 1^{E} 2^{G}	0.00699	0.00004	1.01252	1.00007
2^{D} 2^{E} 2^{G}	0.00501	0.00002	1.01543	1.00004
2^{D} 3^{E} 2^{G}	0.00466	0.00003	1.01301	1.00009
3^{D} 1^{E} 2^{G}	0.00314	0.00002	1.01651	1.00010
3^{D} 2^{E} 2^{G}	0.00975	0.00002	1.01494	1.00004
3^{D} 3^{E} 2^{G}	0.00341	0.00002	1.01921	1.00011
4^{D} 1^{E} 2^{G}	0.00391	0.00002	1.01411	1.00008
4^{D} 2^{E} 2^{G}	**0.00282**	**0.00001**	1.01891	1.00010
4^{D} 3^{E} 2^{G}	0.00552	0.00003	1.01397	1.00005
2^{D} 1^{E} 3^{G}	0.00486	0.00002	1.01641	1.00007
2^{D} 2^{E} 3^{G}	0.00571	0.00002	1.01426	1.00011
2^{D} 3^{E} 3^{G}	0.00425	0.00030	1.01753	1.00012
3^{D} 1^{E} 3^{G}	0.00582	0.00003	1.01572	1.00008
3^{D} 2^{E} 3^{G}	0.00441	0.00002	1.01901	1.00013
3^{D} 3^{E} 3^{G}	0.00496	0.00002	1.01708	1.00010
4^{D} 1^{E} 3^{G}	0.00516	0.00003	1.01587	1.00009
4^{D} 2^{E} 3^{G}	0.00478	0.00002	1.01602	1.00012
4^{D} 3^{E} 3^{G}	0.00506	0.00002	1.01745	1.00011

Using FID as the determining metric, HPO model 17 is the top performer with discriminator, generator and encoder leaning rates set to 0.00004, 0.0000.2 and

0.00002, respectively. We apply t-SNE analysis on the generated images by the HPO 17 model, shown in Figure 5.19.

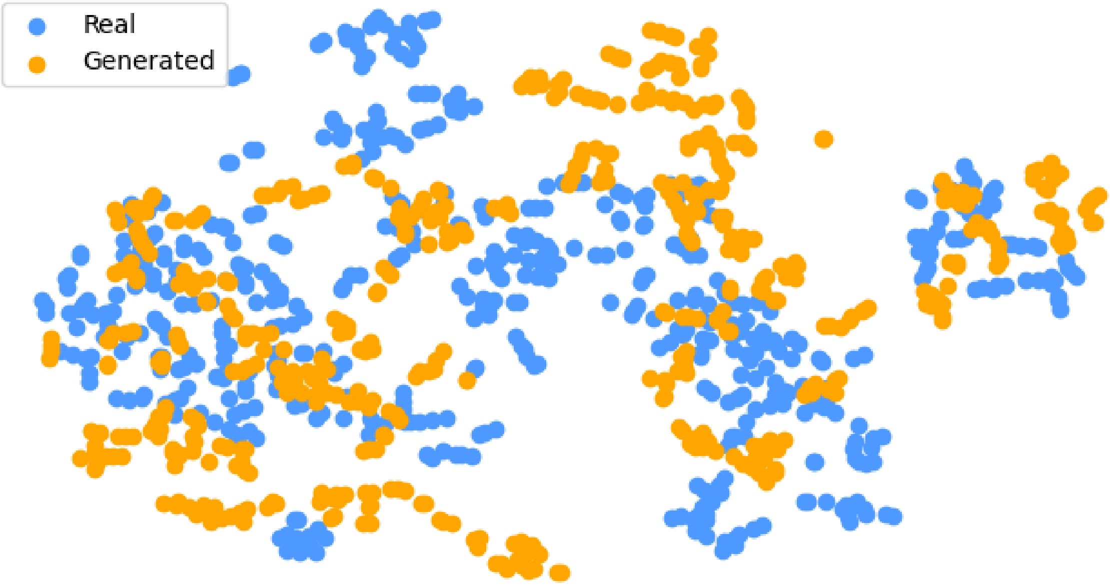


Figure 5.19: t-SNE comparison between real and generated images

Figure 5.19 demonstrates that the model has successfully captured the features of the data, showcasing a superior understanding of the dataset compared to the baseline model (Figure 5.16). The t-SNE plot reveals that the model has better learned the latent dimension space of the real data, resulting in a more accurate representation of the underlying features. While there are some outliers, notably the cluster in the top center, the overall performance of the model in capturing the data's latent characteristics is improved.

We use HPO model 17 to experiment with more latent dimensions. Table 5.10 shows the results.

Table 5.10: Results from latent dimension HPO

Name	**Image FID**	**Mask FID**	**Image IS**	**Mask IS**
1024 (Baseline)	**0.00282**	0.00001	**1.01891**	1.00010
1280	0.01637	**0.000004**	1.00764	1.00005
1536	0.00447	0.00003	1.01374	**1.00022**
1792	0.00514	0.00001	1.01368	1.00010

As depicted in Figure 5.10, the HPO 17 model exhibits superior performance in terms of Image FID compared to other models with varying latent dimensions. Notably, the model with a latent dimension of 1280 achieves a better FID score than the baseline model in the context of mask generation. However, this same model yields unsatisfactory outcomes when it comes to generating images.

5.5.4 Modified HU Interpolation Range

We perform two experiments with different HU low-high thresholds, which are used to interpolate the image values. We perform the experiments on HPO model 17. Table 5.11 shows the results.

Table 5.11: Results from modified HU interpolation range

HU range	**Image FID**	**Mask FID**	**Image IS**	**Mask IS**
-1024, 600 (Baseline)	**0.00282**	0.00001	**1.01891**	1.00010
-2048, 3071	0.3016	0.0001	1.00397	1.00014
-2048, 1200	0.61453	0.00011	1.00975	1.00013

As shown in Table 5.11, the baseline model exhibits significantly better performance compared to the other experiments. Both experiments are able to create images that are visually reminiscent of the original images. However, both models produce large amounts of noise and artifacts in the images.

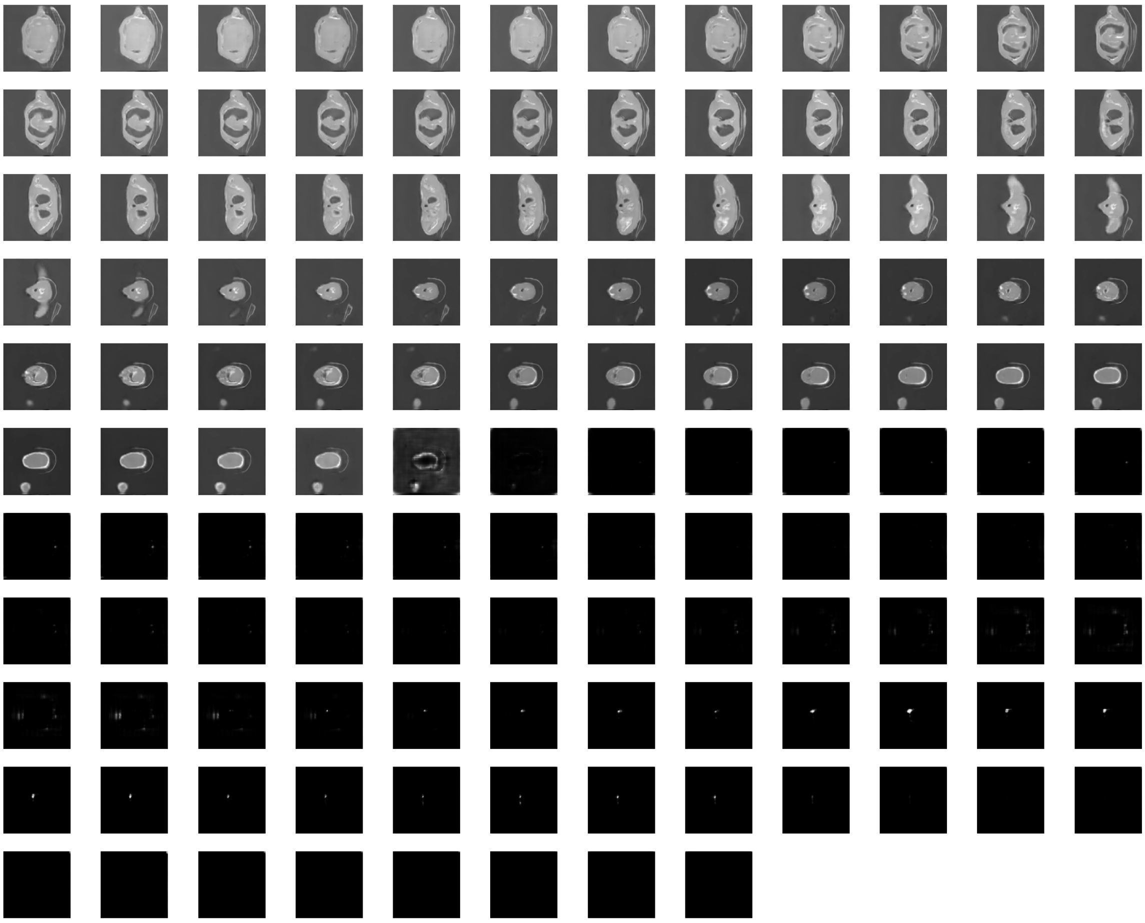
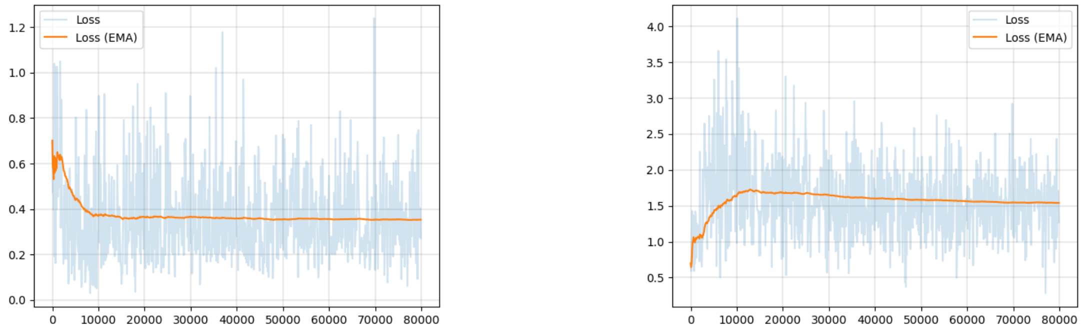


Figure 5.20: Example of a generated image

The figure clearly depicts a section of the image that is heavily affected by noise, noticeable through its gray color. It is apparent that employing the frequently used HU values for interpolation yields visually inadequate outcomes, which is further confirmed by the low KID score.

5.5.5 Data Augmentation

As we saw an increase in performance on image-only generation with voxel resampling, we performed the same experiment with image/mask pairs generation. Figure 5.21 shows the loss diagrams.



(a) Discriminator

(b) Generator

Figure 5.21: HA-GAN Generator and Discriminator loss (blue), EMA loss (orange)

The figure demonstrates that the discriminator rapidly converges to approximately 0.4 and maintains relative stability throughout the training period. It is important to mention that this convergence point is higher than the EMA of 0.2 observed in the baseline image/mask pair discriminator discussed in Chapter 5.5.1. On the other hand, the generator loss levels off at around 1.5, which is a significant improvement compared to the generator baseline, representing a reduction by half in terms of loss.

Table 5.12 shows the KID and IS scores.

Table 5.12: Results from Model Trained on Voxel-Resampled Images

Model	**Image FID**	**Mask FID**	**Image IS**	**Mask IS**
Baseline - HPO 17	0.00282	0.00001	**1.01891**	**1.00010**
Resampled	**0.00258**	0.00001	1.01845	1.00006

According to the table, voxel resampling the images to a size of $1.5 \times 1.5 \times 1.5$ mm yields better results, surpassing the performance of the baseline HPO 17 model. Additionally, when considering the halved loss observed in Figure 5.21b, it becomes evident that voxel resampling contributes to the improvement in model performance.

We created an augmented dataset with the voxel-resampled images and transformations affine transformation and elastic deformation. Table 5.13 show the results.

Table 5.13: Results from Model Trained on Augmented Images

Model	**Image FID**	**Mask FID**	**Image IS**	**Mask IS**
Baseline - HPO 17	**0.00282**	**0.00001**	1.01891	**1.00010**
Augmented	1.18555	0.00002	**2.30551**	1.00011

Interestingly, the model exhibits relatively poor performance in terms of FID when compared to the original baseline and the HPO 17 model. Although the IS shows improvement, the KID score is significantly worse, indicating that the generated images do not align well with the original data. It can be argued that further post-processing of the generated images, specifically tailored to align them more closely with the original data, might be necessary. However, due to the scope of this thesis and time constraints, such post-processing steps were not pursued.

5.5.6 Binary Mask

We perform two experiments with the HA-GAN HPO17 model trained on binary masks. Table 5.16 shows the results.

Table 5.14: Results from Models Trained on Binary Masks

Model	**Image FID**	**Mask FID**	**Image IS**	**Mask IS**
Baseline	0.00282	**0.00001**	1.01891	**1.00010**
Binary with interpolation	1.82901	3.19133	1.00017	0.99998
Binary	**0.00207**	0.00006	**1.01973**	1.00003

Interestingly, the baseline HPO 17 model still has the best FID score for masks, even better than the model trained on binary masks. However, it looks like the binary model is better at creating images since it has the lowest FID and the highest IS, making it the best model for generating images.

On the other hand, the binary model with interpolated mask values doesn't perform as well. It seems that including interpolated mask values might be causing some issues in the generated images, affecting their quality.

The binary-trained model emerges as the superior performer in image generation overall. To evaluate the quality and distribution of the generated images, we conduct a t-SNE analysis of the model's output. The results of this analysis are illustrated in Figure 5.22.

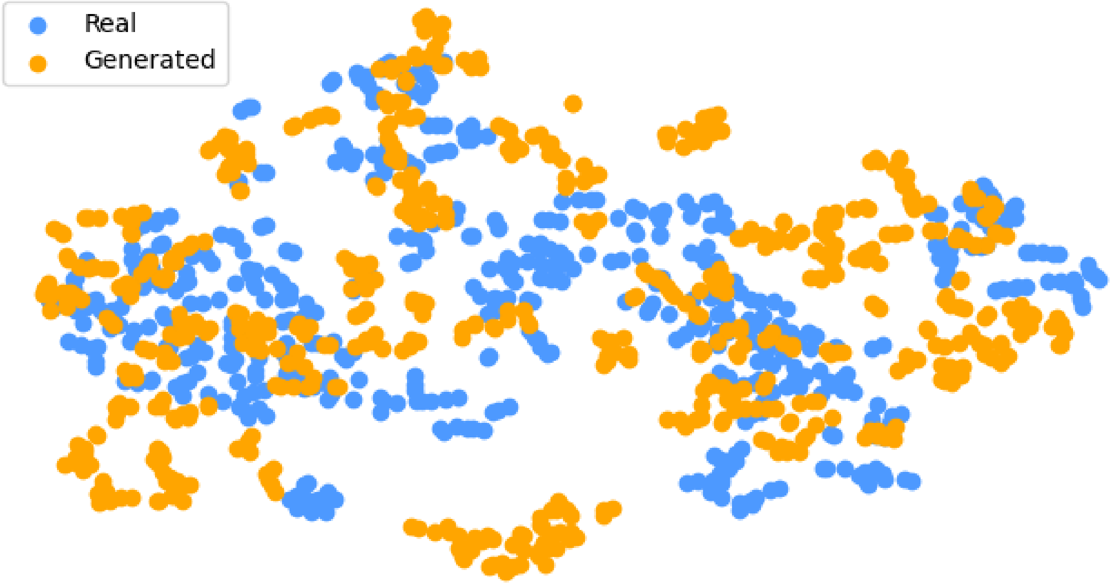


Figure 5.22: t-SNE comparison between real and generated images

Figure 5.22 reveals that the quality and distribution of the generated images by the binary-trained model surpass those of the baseline model (Figure 5.16) and the HPO 17 model (Figure 5.19), further validating the improved FID scores. The t-SNE plot illustrates that the generated images exhibit improved overall quality and better capture the underlying features present in the dataset, including most of the data sources. Although some outliers are visible in the plot, the binary-trained model demonstrates a more comprehensive understanding of the dataset and its underlying characteristics.

5.5.7 Post-Processing

We apply intensity rescaling and HU thresholding to images generated by the HPO 17 model. We also apply binarization and CCA post-processing to the generated mask by the same model.

Table 5.15: Comparison of postprocessing

Model	Image FID	Mask FID	Image IS	Mask IS
Baseline	0.00207	0.00006	**1.01973**	1.00003
Rescaled Image	1.71301	1.00373	1.00017	1.00011
Binary Mask Postprocessing	**0.00201**	**0.00001**	1.00017	1.00012
CCA Mask Postprocessing	0.00245	0.00002	1.00014	**1.00014**

According to the table, the utilization of binarization for postprocessing the generated mask produced the most favorable outcomes. While the CCA postprocessing method did not surpass the straightforward binarization approach, it did exhibit superior image FID results compared to the baseline HPO 17 model. This demonstrates the practical value of CCA postprocessing.

5.5.8 High-Resolution Images

We generate images of size $256 \times 256 \times 256$. We use the HPO 17 model trained on binary masks, with and without voxel resampling. We also generate images with the native model with the original non-HPO learning rates.

Table 5.16: Comparison of high-resolution models

Model	Image FID	Mask FID	Image IS	Mask IS
HPO17	0.05704	0.00020	1.05287	**1.00049**
HPO17 Resampled	**0.01429**	**0.00007**	1.08498	1.00040
Original LR	0.04877	0.00007	**1.13935**	1.00014

Based on the IS, the model trained with the original learning rates is noted for generating the most diverse images. However, the HPO 17 resampled model surpasses its performance, supporting the earlier conclusion that resampling enhances results, even with higher-resolution images. Additionally, the model trained on resampled data exhibits superior performance in terms of FID, scoring 0.0427 better.

5.6 Visual Turing Test

We conducted a Visual Turing Test involving three healthcare professionals. This included two medical doctors, one with a decade of professional experience (MD 1) and another with 15 years of practice (MD 2), as well as a radiologist who has been practicing for three years.

We presented 10 images, five of which were fake in random ordering. The participants were asked to rate each image on a scale from 0 to 10, with 0 signifying a belief that the image was completely real and 10 indicating a belief that the image was entirely fake.

Table 5.17 presents the scores assigned by each healthcare professional to the real images.

Table 5.17: Scores Assigned by Healthcare Professionals

Number	**MD 1**	**Radiologist**	**MD 2**
3 (Real)	2	4	7
4 (Real)	4	5	4
6 (Real)	3	5	7
7 (Real)	4	3	4
8 (Real)	5	4	5

The results, as summarized in Table 5.17, revealed noteworthy variations in the perception of these professionals.

MD 1 consistently rated the images lower than the other two professionals, with an average score of 3.6 out of 10. This suggests that MD 1 was more likely to believe that the images were authentic, indicating the effectiveness of the real images in convincing this professional.

The radiologist, with an average score of 4.2 out of 10, demonstrated a slightly higher degree of suspicion about the authenticity of the images compared to MD 1. However, this score still leans more towards the images being real rather than fake.

MD 2, with an average score of 5.4 out of 10, was the most skeptical of the three professionals about the authenticity of the images.

Table 5.18 presents the scores assigned by each healthcare professional to the generated images.

Table 5.18: Scores Assigned by Healthcare Professionals

Number	**MD 1**	**Radiologist**	**MD 2**
1 (Fake)	3	5	8
2 (Fake)	5	6	6
5 (Fake)	2	3	2
9 (Fake)	3	6	7
10 (Fake)	4	4	8

The average score for MD 1 for the fake images was 3.4 out of 10, comparable to their average score for the real images (3.6). This suggests that the generated images were almost as effective as the real ones in convincing MD 1 of their authenticity.

The radiologist's average score was 4.8 for the fake images, slightly higher than their average score for real images (4.2). This indicates that the radiologist was more skeptical of the fake images, but the difference is relatively small, suggesting that the generated images were fairly convincing.

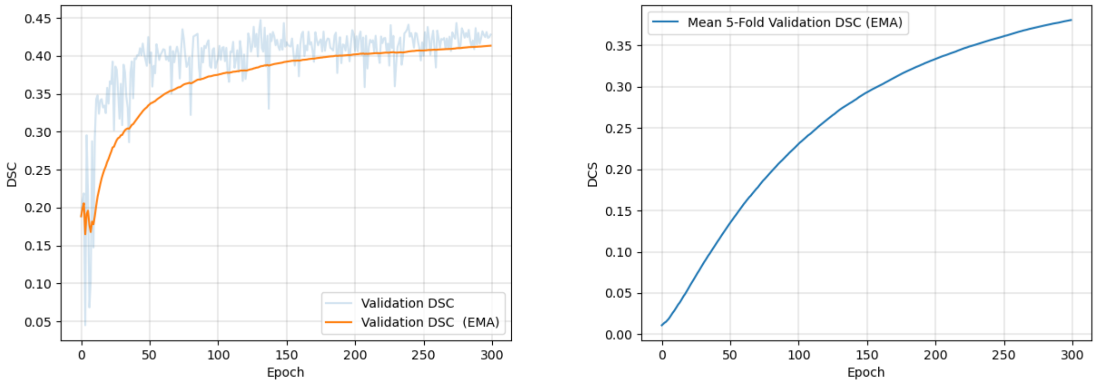
MD 2, with an average score of 6.2 for the fake images, was still the most skeptical among the three professionals. However, the difference is not dramatically significant compared to their average score for real images (5.4). This implies that, despite their skepticism, the generated images were still somewhat believable to MD 2.

Upon analyzing the scores, it is evident that there is some variation in the ability to differentiate between real and HA-GAN-generated images among the participants. These results demonstrate that the generated images have a considerable degree of realism. In particular, the images were very effective in convincing MD 1 and reasonably effective for the radiologist and MD 2. This result underscores the quality of the HA-GAN-generated images and their potential to convincingly mimic real images.

5.7 Segmentation

5.7.1 Baseline

To establish a baseline for performance, we trained the SegResNet model on a single fold of 419 training images, of which 335 were used for training and 83 for validation during training. The input images were of size $128 \times 128 \ \times 64$. Additionally, we trained the SegResNet model on five folds using Cross-Validation, all of which had the same amount of training and validation data. Figure 5.23 shows the results of training the single-fold and 5-Fold. Both experiments ran 300 epochs.



(a) 1-Fold training

(b) 5-Fold Cross-Validation

Figure 5.23: Baseline validation results

Figure 5.23a displays the validation DSC during training for the 1-Fold training session. The maximum score achieved by the model was **0.4472**.

Interestingly, the model quickly reaches the 0.4 threshold after just 50 epochs. The validation DSC becomes relatively stable after 50 epochs, but the EMA trend shows that the performance keeps improving over time without reaching a plateau even after 300 epochs. One limitation of this experiment is that we stop training after 300 epochs due to time constraints, so the performance could possibly be better if we continued training.

Figure 5.23b displays the average validation results from the 5-Fold Cross-Validation. The overall average converges more slowly compared to the 1-Fold training but ultimately achieves the same maximum value. Table 5.19 presents the Cross-Validation scores for each individual fold.

Table 5.19: Aggregated dice metric using 5-fold cross-validation

Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Average**
0.4414	0.4298	0.4618	0.4120	0.4445	0.4379

As inferred in Table 5.19, the maximum of the 1-Fold model is better than the average of the 5-Fold models. After training, we evaluated the models on 105 real images not seen by the models during training. The 1-Fold model achieved a DSC of **0.43916**, while the ensemble 5-Fold model achieved a DSC of **0.48693**, suggesting that 5-Fold CV assists in choosing the best folds for training and validation to increase performance.

In summary, the baseline is established with the best-performing model during training for validation (fold 3) and the best-performing model (5-Fold CV ensemble) when evaluated on unseen images, shown in Table 5.20.

Table 5.20: Baselines for segmentation performance

Validation Score	**Testing Score**
0.4618	0.48693

It is crucial to acknowledge that the scores achieved by the models discussed are not as impressive as the winning solution of the HECKTOR 2022 challenge (Myronenko et al., 2023). However, it is important to recognize that the winning solution utilized more advanced preprocessing techniques and employed a combination of PET and CT images, whereas our approach only utilized CT images. Consequently, the results obtained from these models are not directly comparable due to the differences in methodology and data inputs.

5.7.2 Training on Real and Generated Images

The Auto3Dseg framework was employed with a SegResNet model to train on real, generated, and a combination of real and generated images. The input images were of size $128 \times 128 \ \times 64$.

All experiments were conducted using 5-Fold CV and trained for 300 epochs. To visualize each experiment, we combined the fold score of each experiment and computed the mean validation DSC over all five folds. The total number of training runs is 15. Figure 5.25 illustrates the mean validation DSC for the three 5-Fold CV experiments, calculated with EMA. We leave out the original data from the plot for visibility.

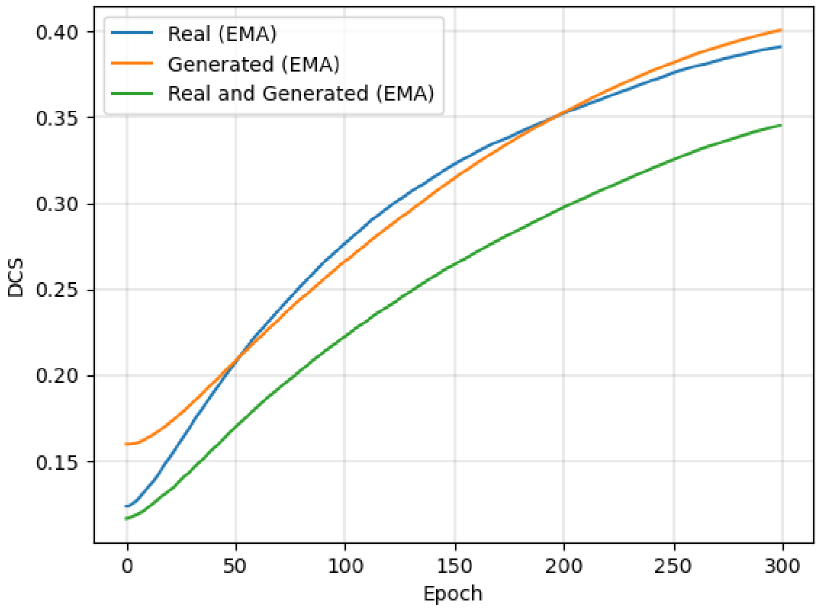


Figure 5.24: Average 5-Fold Validation DSC for real, generated, and real & generated images

Figure 5.25 demonstrates that initially, models trained on real images surpass those trained on generated ones. However, in the end, models trained on generated images outperform, reaching an average score of 0.40 on the EMA calculated line for all models. In comparison, models trained solely on real images display similar performance. The weakest performance is seen in models trained on both real and generated images, attaining an average of 0.34 on the EMA line.

Nonetheless, it's crucial to acknowledge that all three experiments display a sharp upward trend in their EMA lines, indicating an ongoing improvement in performance. We argue that this upward trajectory would continue if the training period extended beyond 300 epochs.

Table 5.21 shows the validation DSC for each fold, including the average for each experiment.

Table 5.21: Aggregated dice metric using 5-fold cross-validation

Model	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Average**
Baseline	0.4414	0.4298	0.4618	0.4120	0.4445	**0.4379**
Real	0.4519	0.4057	0.4533	0.4431	0.4713	0.4314
Generated	0.4322	0.4692	0.4610	0.4497	0.4016	0.4365
Real and Generated	0.3819	0.4157	0.3897	0.3862	0.3877	0.3922

As observed in Table 5.21, the real-only images yield an average score nearly identical to the baseline, which is anticipated since it involves the same data. However, it is beneficial to reaffirm the results through this additional validation.

The baseline maintains the highest average score, while the model trained solely on generated images exhibits a strikingly close performance, with a mere 0.0014 difference. It is interesting to note that the model trained on generated images achieved a slightly better average across all five folds than the model trained on real images. This suggests that the generated images are on par with the real images, showcasing their potential ability to be used as training data for segmentation.

The model trained on a mix of real and generated images performs the worst, scoring 0.0457 below the baseline. The performance of the models trained only on generated images implies that a mixed model should exhibit improved results. However, the mixed model demonstrates the lowest performance, indicating potential errors in the segmentation process. This discrepancy may be attributed to a potentially flawed preprocessing process of both generated and real images when combined. A limitation with this experiment is the unexplored preprocessing steps taken, which could potentially increase the performance of the mixed model.

The same experiments were run with input images of size $256 \times 256 \ \times 128$. The results are shown in Figure 5.25.

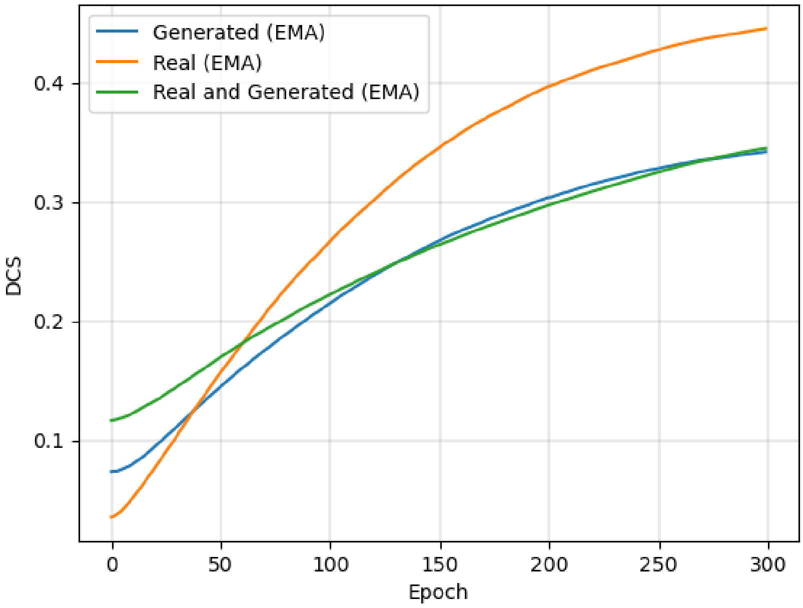


Figure 5.25: Average 5-Fold Validation DSC for real, generated, and real generated images

As observed in Figure 5.25, the real-only model exhibits better overall performance and reaches high levels more rapidly. The generated-only model initially lags behind the mixed model, overtakes it, but eventually falls short in the end. A commonality among all three experiments is the persistent upward trend in performance, similar to previous experiments. It is likely that these experiments would also show enhanced performance if given additional training time.

Table 5.22 shows the highest score for each fold.

Table 5.22: Aggregated dice metric using 5-fold cross-validation

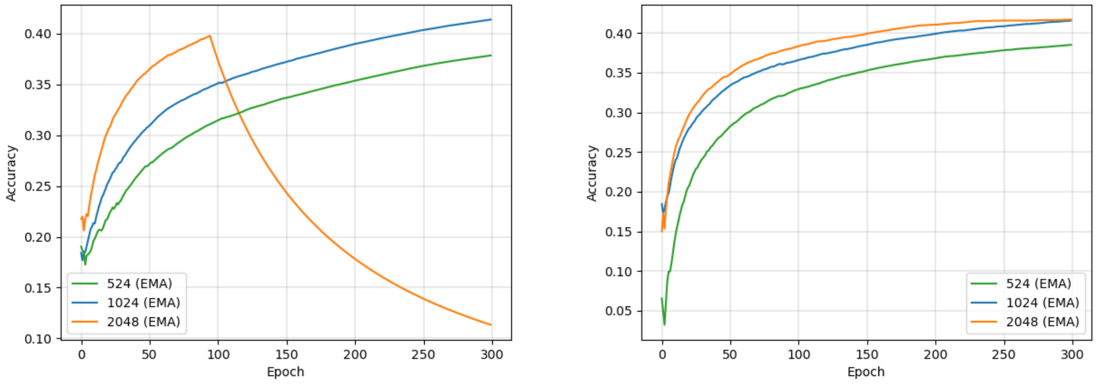
Model	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Average**
Real	0.5164	0.4923	0.5620	0.5179	0.5414	**0.5260**
Generated	0.3981	0.3771	0.3965	0.4322	0.4187	0.4045
Real and Generated	0.3387	0.3801	0.4110	0.4031	0.4310	0.3927

As indicated by Table 5.22 and Figure 5.25, the real-only model for the image size

$256 \times 256 \times 128$ demonstrates the highest average performance. Similar to the smaller image sizes, the mixed model performs the worst, although its average score is only slightly lower than that of the generated-only model. Nevertheless, the close scores suggest that the generated images can effectively compete with real images when utilized as training data.

5.7.3 Training on Generated Images - Testing on Real Images

We generated three synthetic datasets of 512, 1024, and 2048 images. We trained the SegResNet on only generated images with only 1 fold. After training, we run the network in inference mode and predict labels on unseen test data. We manually calculate DSC by calculating the dice score for each predicted label and taking the average of the scores. We repeated the process for image sizes $128 \times 128 \times 64$ and $256 \times 256 \times 128$. The validation DSC (EMA) for all six experiments is shown in Figure 5.26.



(a) $128 \times 128 \times 64$

(b) $256 \times 256 \times 128$

Figure 5.26: Average validation DSC for models trained on generated images

The most visible observation is the significant downturn in the model's performance when trained on 2048 generated images. Around the 100th epoch, the EMA line shows a swift drop in the validation score. Upon examining the original data, it's seen that both the score and loss plummet to zero in every subsequent epoch after a specific point in time. This unusual occurrence suggests that there could have been some error during the training phase, resulting from various possible issues.

One such issue could be an excessively high learning rate, which would result in abrupt updates to the weights and possibly give rise to anomalies like this one. However, considering that the same learning rates are applied across all other segmentation experiments, this is unlikely to be the cause.

A more plausible explanation could be errors in the post-processing of the generated images or pre-processing before inputting the images into the SegResNet model. Any significant bugs or mistakes in this procedure could result in labeling problems, such as duplicate images and/or labels. This, in turn, could potentially cause the loss to drop to zero.

The best average validation DSC during training is shown in Table 5.23.

Table 5.23: Validation scores for models trained on generated Images

Image Size	**Generated Images**	**Validation DSC**
$128 \times 128 \times 64$	None (Baseline)	0.4618
$128 \times 128 \times 64$	524	0.4347
$128 \times 128 \times 64$	1024	0.4710
$128 \times 128 \times 64$	2048	0.4622
$256 \times 256 \times 128$	524	0.4467
$256 \times 256 \times 128$	1024	0.4786
$256 \times 256 \times 128$	2048	0.4739

As depicted in Table 5.23, the models with dimensions $128 \times 128 \times 64$ display similar validation performance. Notably, the model with 2048 demonstrated a validation score surpassing the baseline, although it eventually fell to 0 at a later stage, as illustrated in Figure 5.26a.

The model with 1024 managed to outperform the baseline, securing a score of 0.0092 above it.

Two of the models with dimensions $256 \times 256 \times 128$ exceeded the score of the baseline. However, a direct comparison isn't possible as they were trained on a different image size than that of the baseline. This presents a constraint in the segmentation experiments, as we couldn't establish a baseline for $256 \times 256 \times 128$ due to time limitations.

After training the models, we evaluated the models on the 105 real testing images. Figure 5.27 showcases an illustrative prediction mask as an example. The obtained dice score for this specific example is merely 0.22. For clarity, blank slices have been eliminated. The ground truth mask values are depicted in white, the overlapping region between the ground truth and the prediction is presented in green, and the non-overlapping portion of the prediction mask is shown in red.

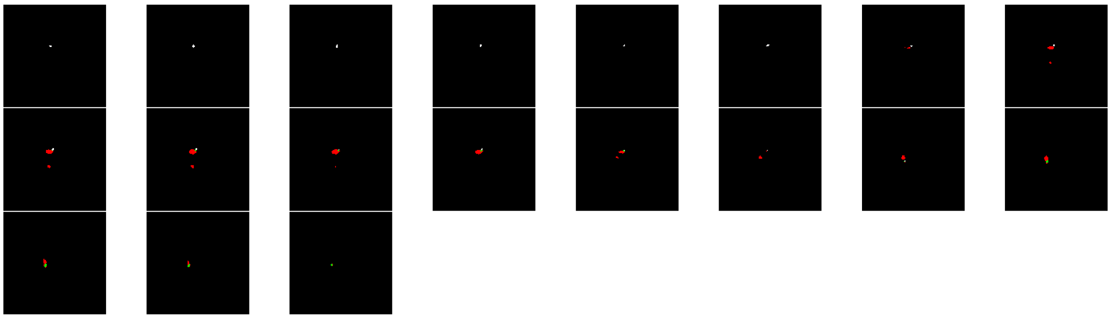


Figure 5.27: Ground truth and prediction mask with 0.22 score

Results are presented in Table 5.24.

Table 5.24: Testing scores for models trained on generated images

Image Size	Number of Fake Images	Testing DSC
$128 \times 128 \times 64$	None (Baseline)	0.4869
$128 \times 128 \times 64$	524	0.2529
$128 \times 128 \times 64$	1024	0.2918
$128 \times 128 \times 64$	2048	0.1944
$256 \times 256 \times 128$	524	0.3152
$256 \times 256 \times 128$	1024	0.3388
$256 \times 256 \times 128$	2048	0.3168

As illustrated in Table 5.24, all models trained on generated images underperform compared to the baseline. The 2048 model particularly lags behind, with a DSC of 0.1944, which is expected given that the score plummeted to 0 during training. The 1024 model stands out as the best performer among the $128 \times 128 \times 64$ models, further validating its superior performance demonstrated in the validation data (Table 5.23).

The models with dimensions $256 \times 256 \times 128$ generally show superior performance, outdoing all the $128 \times 128 \times 64$ models. However, even the top performer, the $256 \times 256 \times 128$ model trained on 1024 generated images, reaches only a score of 0.3388, which is 0.1481 below the baseline. While the direct comparison is not feasible for models trained on different image sizes, the outcomes are suggestive.

These results clearly indicate that models trained on generated images are overfitting. In other words, these models are not adequately learning the underlying features of the data, thereby struggling to make predictions on test data. Consequently, the generalizability of these models is lacking. This leads to the conclusion that, while useful, generated images alone are not as effective as real images and cannot solely serve as training data.

5.7.4 Training on Real and Generated Images - Testing on Real Images

We generated 524 images with the HPO17 model without applying threshold rescaling. We split the real data into 419 training images and 105 testing images. We combined the 524 generated images with the 419 real images, resulting in a training dataset of 943 images. We trained the SegResNet model for 300 epochs using a single fold. We used the model to run inference on the 105 real testing images. Additionally, We trained the SegResNet model for 300 epochs on the same data, using 5-Fold CV, choosing the best model to run inference on the 105 real testing images.

We generated 524 images with the GAN model trained on binary masks and applied thresholding rescaling to the images. We combined the generated images with the same 419 real training images as before. We trained the SegResNet on a single fold and ran inference after training.

Table 5.25 shows the results of the three experiments.

Table 5.25: Testing scores for models trained on real and generated images

Model	**Testing Score**
Baseline	0.48693
Binary Trained 1-Fold	0.46834
HPO 17 1-Fold	0.49380
HPO 17 5-Fold	**0.52193**

According to the findings presented in Table 5.25, the performance of the SegResNet model, which was trained on a combination of 419 real images and 524 generated images produced by the binary-trained HA-GAN, did not surpass the baseline.

It was observed that the SegResNet model, trained on 524 generated images generated by the HPO 17 HA-GAN and 419 real images, achieved a Dice Similarity Coefficient (DSC) of **0.4938** on the 105 testing images. This result indicates a slight improvement of 0.00687 compared to the baseline.

Furthermore, employing the same ensemble model with a 5-fold Cross-Validation technique resulted in even better outcomes. The improved model achieved a DSC of **0.52193** on the same set of testing images, which is a considerable improvement.

Figure 5.28 show an example prediction mask with a dice score of 0.87.

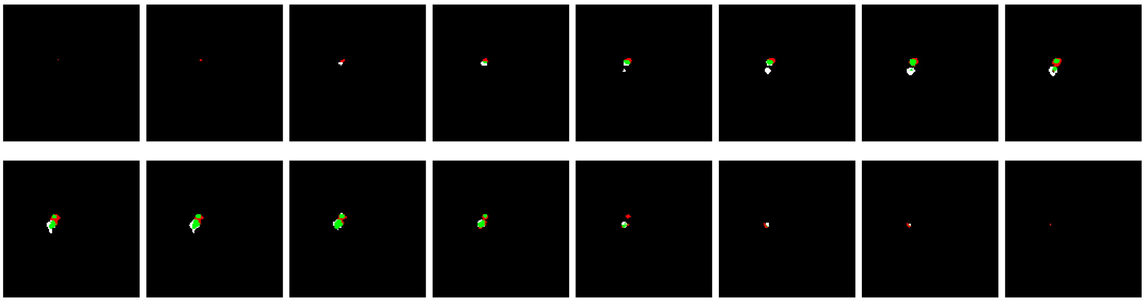


Figure 5.28: Ground truth and prediction mask with 0.87 score

6 Limitations

One notable drawback of the GAN model is its limited number of training images, which amounted to a mere 524. In contrast, the authors of the HA-GAN paper (Sun et al., 2022) employed a more extensive dataset of 9276 images for training. Furthermore, it is worth mentioning that HAGAN, a variant of GAN, provides support for conditional training. Moreover, the dataset used in this study comprises both PET and CT images, rendering them suitable for various applications.

With regard to the segmentation results, there is one limitation that must be taken into consideration when looking at the results. The fact that the HA-GAN model has seen all the images, including the test images, could potentially play a factor in the segmentation performance and introduce data leakage, referring to the situation where information from the test set (unseen data) unintentionally leaks into the training process, leading to overly optimistic performance results.

When the GAN model generates images, including those that resemble the test images, the SegResNet model trained on the combined dataset of real and generated images may inadvertently learn to recognize specific patterns or characteristics present in the test set. As a result, during inference on the test set, the model might perform better than expected due to its familiarity with the leaked information.

Data leakage can lead to an overestimation of the performance and compromise its ability to generalize to new, unseen data. One could make a case for training the GAN solely on the 419 real training images, excluding the 105 testing images, to avoid potential data leakage. However, it should be noted that the training dataset of 419 images is relatively small, which may not provide sufficient information for the GAN to effectively learn the underlying data distribution. This issue highlights the ongoing challenge of insufficient medical data within the realm of machine learning and deep learning.

Furthermore, it should be noted that we did not establish a baseline for image size $256 \times 256 \times 128$, which means that the results obtained cannot be directly compared to that particular configuration. Additionally, the final models that outperformed the baseline in terms of testing DSC were only trained on smaller image sizes of $128 \times 128 \times 64$. Unfortunately, due to time constraints, we were unable to explore the performance of larger image sizes.

Another aspect to consider is that we did not utilize resampled GAN images for training the models, despite the fact that our results showed improved FID scores with this approach.

Another limitation was that we did not conduct experiments involving training on generated data and subsequent fine-tuning using real data, despite previous research suggesting that such an approach could lead to increased performance, shown in Chapter 2.9.1.

7 Conclusion and Future Work

The significance of early and precise cancer detection cannot be understated, primarily when considered in the context of improving patient outcomes and survival rates. This research was initiated with two primary questions: Can Generative Adversarial Networks (GANs) generate realistic high-resolution 3D CT image/mask pairs? And to what extent do synthetic CT images generated by GANs impact the accuracy of a state-of-the-art cancer segmentation model? The answers to these questions have been explored throughout this thesis by addressing the issue of insufficient medical data using Generative Adversarial Networks to generate synthetic data, augmenting the limited real-world datasets.

This thesis implemented and evaluated several GAN models, including a custom Vanilla GAN, Wasserstein GAN, StyleGAN2, FastGAN, and Hierarchical Amortized GAN, using the HECKTOR 2022 Dataset with 524 CT images with masks as the primary training dataset. The evaluation was conducted using a variety of metrics, including the Inception score, Frechet Inception Distance, t-distributed Stochastic Neighbor Embedding, and a Visual Turing Test, with the latter administered to healthcare professionals.

Initial findings indicated that the Vanilla GAN model needed more complexity to generate high-resolution images, limiting its utility in this context. Similarly, StyleGAN2 and FastGAN could generate 2D images but were restricted in resolution. While these models could potentially generate higher-dimensional images with additional work, the direction of the study shifted towards a more promising 3D approach.

The decision to crop images proved to be a critical factor in achieving high-quality results, primarily due to the variability in image sizes stemming from the dataset's diverse origins, which included seven different center sources of CT images. In contrast, HA-GAN demonstrated the capability to generate high-resolution images, including image/mask pairs with favorable FID scores. This made HA-GAN the most suitable candidate for the thesis, which was further optimized using hyperparameter optimization.

The real and generated data were subsequently used as training data for the SegResNet model, part of MONAI's Auto3Dseg framework, for segmentation. The model was trained using real and synthetic data via 5-Fold Cross-Validation and tested on 105 real images. The results indicated that the inclusion of synthetic images in the training set improved the baseline Dice Similarity Coefficient by 5%, thereby validating the effectiveness of the approach. Furthermore, a visual Turing test conducted with healthcare professionals showed that the synthetic images were realistic enough to fool them, further reinforcing the quality of the generated data.

In conclusion, this thesis has demonstrated that using GANs, particularly the HA-GAN model, can effectively address the challenge of limited training data for machine learning segmentation models in cancer detection. By generating high-quality, high-resolution synthetic images, these models can provide a valuable augmentation to real-world datasets, enhancing the accuracy and effectiveness of segmentation models. This work thus provides a robust response to the posed research questions, proving the viability of using GANs for the generation of synthetic CT images and demonstrating their positive impact on the accuracy of cancer segmentation models.

7.1 Future Work

Looking forward, several avenues of research could potentially enhance the effectiveness of the machine learning models used for cancer detection. One such area involves further experimentation with the input threshold range used to interpolate the CT images. Our preliminary analysis in Chapter 3.2 indicated variability in these ranges. Although our experiments produced inferior results with other ranges than the original $[-1024, 600]$, future research should explore a wider variety of ranges.

Moreover, although our data augmentation experiments did not outperform the baseline results, there remains significant potential for improving the limited training data size by augmenting more images.

It is crucial to acknowledge the significance of data leakage as a notable constraint. To address this concern, future endeavors should contemplate training the GAN using a subset of the data while reserving the remaining portion exclusively for testing the segmentation model.

Additionally, future work should involve more experiments with the segmentation, specifically focusing on larger image sizes, such as $256 \times 256 \times 256$. As the resolution and complexity of images increase, the segmentation process becomes more challenging. However, with more research and experimentation, it's plausible that this could lead to better model performance and, consequently, more accurate cancer segmentation.

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Aggarwal et al., 2021] Aggarwal, A., Mittal, M., and Battineni, G. (2021). Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, 1(1):100004.

[Alakwaa et al., 2017] Alakwaa, W., Nassef, M., and Badr, A. (2017). Lung cancer detection and classification with 3d convolutional neural network (3d-cnn). *International Journal of Advanced Computer Science and Applications*, 8(8).

[Andrearczyk et al., 2023] Andrearczyk, V., Oreiller, V., Abobakr, M., Akhavanallaf, A., Balermpas, P., Boughdad, S., Capriotti, L., Castelli, J., Cheze Le Rest, C., Decazes, P., et al. (2023). Overview of the hecktor challenge at miccai 2022: automatic head and neck tumor segmentation and outcome prediction in pet/ct. In *Head and Neck Tumor Segmentation and Outcome Prediction: Third Challenge, HECKTOR 2022, Held in Conjunction with MICCAI 2022, Singapore, September 22, 2022, Proceedings*, pages 1–30. Springer.

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

[Arora and Arora, 2022] Arora, A. and Arora, A. (2022). Generative adversarial networks and synthetic patient data: current challenges and future perspectives. *Future Healthcare Journal*, 9(2):190.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Benhammou et al., 2018] Benhammou, Y., Tabik, S., Achchab, B., and Herrera, F. (2018). A first study exploring the performance of the state-of-the art cnn model in the problem of breast cancer. In *proceedings of the international conference on learning and optimization algorithms: theory and applications*, pages 1–6.

[Bertels et al., 2019] Bertels, J., Eelbode, T., Berman, M., Vandermeulen, D., Maes, F., Bisschops, R., and Blaschko, M. B. (2019). Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part II 22*, pages 92–100. Springer.

[Bishop, 1994] Bishop, C. M. (1994). Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832.

[Bray et al., 2021] Bray, F., Laversanne, M., Weiderpass, E., and Soerjomataram, I. (2021). The ever-increasing importance of cancer as a leading cause of premature death worldwide. *Cancer*, 127(16):3029–3030.

[Brett et al., 2023] Brett, M., Markiewicz, C. J., Hanke, M., Côté, M.-A., Cipollini, B., McCarthy, P., Jarecka, D., Cheng, C. P., Halchenko, Y. O., Cottaar, M., Larson, E., Ghosh, S., Wassermann, D., Gerhard, S., Lee, G. R., Wang, H.-T., Kastman, E., Kaczmarzyk, J., Guidotti, R., Daniel, J., Duek, O., Rokem, A., Madison, C., Papadopoulos Orfanos, D., Sólon, A., Moloney, B., Morency, F. C., Goncalves, M., Baratz, Z., Markello, R., Riddell, C., Burns, C., Millman, J., Gramfort, A., Leppäkangas, J., van den Bosch, J. J., Vincent, R. D., Braun, H., Subramaniam, K., Van, A., Gorgolewski, K. J., Raamana, P. R., Klug, J., Nichols, B. N., Baker, E. M., Hayashi, S., Pinsard, B., Haselgrove, C., Hymers, M., Esteban, O., Koudoro, S., Pérez-García, F., Dockès, J., Oosterhof, N. N., Amirbekian, B., Nimmo-Smith, I., Nguyen, L., Reddigari, S., St-Jean, S., Panfilov, E., Garyfallidis, E., Varoquaux, G., Legarreta, J. H., Hahn, K. S., Waller, L., Hinds, O. P., Fauber, B., Roberts, J., Poline, J.-B., Stutters, J., Jordan, K., Cieslak, M., Moreno, M. E., Hrnčiar, T., Haenel, V., Schwartz, Y., Darwin, B. C., Thirion, B., Gauthier, C., Solovey, I., Gonzalez, I., Palasubramaniam, J., Lecher, J., Leinweber, K., Raktivan, K., Calábková, M., Fischer, P., Gervais, P., Gadde, S., Ballinger, T., Roos, T., Reddam, V. R., and freec84 (2023). nipy/nibabel: 5.0.1.

[Bu et al., 2021] Bu, T., Yang, Z., Jiang, S., Zhang, G., Zhang, H., and Wei, L. (2021). 3d conditional generative adversarial network-based synthetic medical image augmentation for lung nodule detection. *International Journal of Imaging Systems and Technology*, 31(2):670–681.

[Buzug, 2011] Buzug, T. M. (2011). *Computed tomography*. Springer.

[Cardoso et al., 2022] Cardoso, M. J., Li, W., Brown, R., Ma, N., Kerfoot, E., Wang, Y., Murrey, B., Myronenko, A., Zhao, C., Yang, D., et al. (2022). Monai: An open-source framework for deep learning in healthcare. *arXiv preprint arXiv:2211.02701*.

[Chen et al., 2021] Chen, R. J., Lu, M. Y., Chen, T. Y., Williamson, D. F., and Mahmood, F. (2021). Synthetic data in machine learning for medicine and healthcare. *Nature Biomedical Engineering*, 5(6):493–497.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. https://keras.io.

[Chuquicusma et al., 2018] Chuquicusma, M. J., Hussein, S., Burt, J., and Bagci, U. (2018). How to fool radiologists with generative adversarial networks? a visual turing test for lung cancer diagnosis. In *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pages 240–244. IEEE.

[Cirillo et al., 2021] Cirillo, M. D., Abramian, D., and Eklund, A. (2021). Vox2vox: 3d-gan for brain tumour segmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 6th International Workshop, BrainLes 2020, Held in Conjunction with MICCAI 2020, Lima, Peru, October 4, 2020, Revised Selected Papers, Part I 6*, pages 274–284. Springer.

[Croitoru et al., 2023] Croitoru, F.-A., Hondru, V., Ionescu, R. T., and Shah, M. (2023). Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Dexter et al., 2020] Dexter, G. P., Grannis, S. J., Dixon, B. E., and Kasthurirathne, S. N. (2020). Generalization of machine learning approaches to identify notifiable conditions from a statewide health information exchange. *AMIA Summits on Translational Science Proceedings*, 2020:152.

[Dillencourt et al., 1992] Dillencourt, M. B., Samet, H., and Tamminen, M. (1992). A general approach to connected-component labeling for arbitrary image representations. *Journal of the ACM (JACM)*, 39(2):253–280.

[Durall et al., 2020] Durall, R., Chatzimichailidis, A., Labus, P., and Keuper, J. (2020). Combating mode collapse in gan training: An empirical analysis using hessian eigenvalues. *arXiv preprint arXiv:2012.09673*.

[Forsyth et al., 1999] Forsyth, D. A., Mundy, J. L., di Gesú, V., Cipolla, R., LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. *Shape, contour and grouping in computer vision*, pages 319–345.

[Frid-Adar et al., 2018] Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J., and Greenspan, H. (2018). Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331.

[Giger and Suzuki, 2008] Giger, M. L. and Suzuki, K. (2008). Computer-aided diagnosis. In *Biomedical information technology*, pages 359–XXII. Elsevier.

[Goodfellow et al., 2020] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems*, 30.

[Hager, 1979] Hager, W. W. (1979). Lipschitz continuity for constrained processes. *SIAM Journal on Control and Optimization*, 17(3):321–338.

[Halicek et al., 2017] Halicek, M., Lu, G., Little, J. V., Wang, X., Patel, M., Griffith, C. C., El-Deiry, M. W., Chen, A. Y., and Fei, B. (2017). Deep convolutional neural networks for classifying head and neck cancer using hyperspectral imaging. *Journal of biomedical optics*, 22(6):060503–060503.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

[Houssein et al., 2021] Houssein, E. H., Emam, M. M., Ali, A. A., and Suganthan, P. N. (2021). Deep and machine learning techniques for medical imaging-based breast cancer: A comprehensive review. *Expert Systems with Applications*, 167:114161.

[Huk, 2020] Huk, M. (2020). Stochastic optimization of contextual neural networks with rmsprop. In *Intelligent Information and Database Systems: 12th Asian Conference, ACIIDS 2020, Phuket, Thailand, March 23–26, 2020, Proceedings, Part II 12*, pages 343–352. Springer.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[Hussain et al., 2022] Hussain, B. Z., Andleeb, I., Ansari, M. S., Joshi, A. M., and Kanwal, N. (2022). Wasserstein gan based chest x-ray dataset augmentation for deep learning models: Covid-19 detection use-case. In *2022 44th Annual*

International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), pages 2058–2061. IEEE.

[Iantsen et al., 2021] Iantsen, A., Visvikis, D., and Hatt, M. (2021). Squeeze-and-excitation normalization for automated delineation of head and neck primary tumors in combined pet and ct images. In *Head and Neck Tumor Segmentation: First Challenge, HECKTOR 2020, Held in Conjunction with MICCAI 2020, Lima, Peru, October 4, 2020, Proceedings 1*, pages 37–43. Springer.

[Jain et al., 2020] Jain, A., Patel, H., Nagalapatti, L., Gupta, N., Mehta, S., Guttula, S., Mujumdar, S., Afzal, S., Sharma Mittal, R., and Munigala, V. (2020). Overview and importance of data quality for machine learning tasks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3561–3562.

[JetBrains, 2023] JetBrains (2023). Pycharm: The python ide for professional developers by jetbrains. Accessed: May 13, 2023.

[Jnawali et al., 2018] Jnawali, K., Arbabshirani, M. R., Rao, N., and Patel, A. A. (2018). Deep 3d convolution neural network for ct brain hemorrhage classification. In *Medical Imaging 2018: Computer-Aided Diagnosis*, volume 10575, pages 307–313. SPIE.

[Joyce, 2011] Joyce, J. M. (2011). Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer.

[Kamath et al., 2019] Kamath, U., Liu, J., and Whitaker, J. (2019). *Deep learning for NLP and speech recognition*, volume 84. Springer.

[Kao and Yang, 2022] Kao, Y.-S. and Yang, J. (2022). Deep learning-based auto-segmentation of lung tumor pet/ct scans: a systematic review. *Clinical and Translational Imaging*, 10(2):217–223.

[Karras et al., 2019] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of* *the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410.

[Karras et al., 2020] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119.

[Kavitha et al., 2023] Kavitha, R., Jothi, D. K., Saravanan, K., Swain, M. P., Gonzáles, J. L. A., Bhardwaj, R. J., Adomako, E., et al. (2023). Ant colony optimization-enabled cnn deep learning technique for accurate detection of cervical cancer. *BioMed Research International*, 2023.

[Kelley, 1960] Kelley, H. J. (1960). Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Klinker, 2011] Klinker, F. (2011). Exponential moving average versus moving exponential average. *Mathematische Semesterberichte*, 58:97–107.

[Koonce and Koonce, 2021] Koonce, B. and Koonce, B. (2021). Resnet 50. *Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization*, pages 63–72.

[Li et al., 2018] Li, J., Madry, A., Peebles, J., and Schmidt, L. (2018). On the limitations of first-order approximation in gan dynamics. In *International Conference on Machine Learning*, pages 3005–3013. PMLR.

[Liashchynskyi and Liashchynskyi, 2019] Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*.

[Liu et al., 2021] Liu, B., Zhu, Y., Song, K., and Elgammal, A. (2021). Towards faster and stabilized gan training for high-fidelity few-shot image synthesis. In *International Conference on Learning Representations*.

[McKinney et al., 2010] McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

[Muehllehner and Karp, 2006] Muehllehner, G. and Karp, J. S. (2006). Positron emission tomography. *Physics in Medicine & Biology*, 51(13):R117.

[Myronenko, 2019] Myronenko, A. (2019). 3d mri brain tumor segmentation using autoencoder regularization. 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*, pages 311–320. Springer.

[Myronenko et al., 2023] Myronenko, A., Siddiquee, M. M. R., Yang, D., He, Y., and Xu, D. (2023). Automated head and neck tumor segmentation from 3d pet/ct hecktor 2022 challenge report. In *Head and Neck Tumor Segmentation and Outcome Prediction: Third Challenge, HECKTOR 2022, Held in Conjunction with MICCAI 2022, Singapore, September 22, 2022, Proceedings*, pages 31–37. Springer.

[NITRC, 2022] NITRC (2022). Mricrogl.
https://www.nitrc.org/projects/mricrogl. Accessed: May 13, 2023.

[Oreiller et al., 2022] Oreiller, V., Andrearczyk, V., Jreige, M., Boughdad, S., Elhalawani, H., Castelli, J., Vallieres, M., Zhu, S., Xie, J., Peng, Y., et al. (2022). Head and neck tumor segmentation in pet/ct: the hecktor challenge. *Medical image analysis*, 77:102336.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Pesaranghader et al., 2021] Pesaranghader, A., Wang, Y., and Havaei, M. (2021). Ct-sgan: computed tomography synthesis gan. In *Deep Generative Models, and Data Augmentation, Labelling, and Imperfections: First Workshop, DGM4MICCAI 2021, and First Workshop, DALI 2021, Held in Conjunction with MICCAI 2021, Strasbourg, France, October 1, 2021, Proceedings 1*, pages 67–79. Springer.

[Rezaei, 2021] Rezaei, Z. (2021). A review on image-based approaches for breast cancer detection, segmentation, and classification. *Expert Systems with Applications*, 182:115204.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[Russ et al., 2019] Russ, T., Goerttler, S., Schnurr, A.-K., Bauer, D. F., Hatamikia, S., Schad, L. R., Zöllner, F. G., and Chung, K. (2019). Synthesis of ct images from digital body phantoms using cyclegan. *International journal of computer assisted radiology and surgery*, 14:1741–1750.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252.

[Salimans et al., 2016] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29.

[Shorten and Khoshgoftaar, 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48.

[Siddique et al., 2022] Siddique, M. M. R., Yang, D., He, Y., Xu, D., and Myronenko, A. (2022). Automated ischemic stroke lesion segmentation from 3d mri. *arXiv preprint arXiv:2209.09546*.

[Sidey-Gibbons and Sidey-Gibbons, 2019] Sidey-Gibbons, J. A. and Sidey-Gibbons, C. J. (2019). Machine learning in medicine: a practical introduction. *BMC medical research methodology*, 19:1–18.

[Simula, 2023] Simula (2023). Simula ex3. Accessed: May 13, 2023.

[Sorin et al., 2020] Sorin, V., Barash, Y., Konen, E., and Klang, E. (2020). Creating artificial images for radiology applications using generative adversarial networks (gans)–a systematic review. *Academic radiology*, 27(8):1175–1185.

[Sun et al., 2022] Sun, L., Chen, J., Xu, Y., Gong, M., Yu, K., and Batmanghelich, K. (2022). Hierarchical amortized gan for 3d high resolution medical image synthesis. *IEEE journal of biomedical and health informatics*, 26(8):3966–3975.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Thambawita et al., 2022] Thambawita, V., Salehi, P., Sheshkal, S. A., Hicks, S. A., Hammer, H. L., Parasa, S., Lange, T. d., Halvorsen, P., and Riegler, M. A. (2022). Singan-seg: Synthetic training data generation for medical image segmentation. *PloS one*, 17(5):e0267976.

[Ugai et al., 2022] Ugai, T., Sasamoto, N., Lee, H.-Y., Ando, M., Song, M., Tamimi, R. M., Kawachi, I., Campbell, P. T., Giovannucci, E. L., Weiderpass, E., et al. (2022). Is early-onset cancer an emerging global epidemic? current evidence and future implications. *Nature Reviews Clinical Oncology*, 19(10):656–673.

[Vallender, 1974] Vallender, S. (1974). Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786.

[Van der Maaten and Hinton, 2008] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

[Van der Walt et al., 2014] Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., and Yu, T. (2014). scikit-image: image processing in python. *PeerJ*, 2:e453.

[Van Rossum and Drake, 2009] Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

[Vlaardingerbroek and Boer, 2013] Vlaardingerbroek, M. T. and Boer, J. A. (2013). *Magnetic resonance imaging: theory and practice*. Springer Science & Business Media.

[Voulodimos et al., 2018] Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E., et al. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018.

[Wang et al., 2019] Wang, F., Casalino, L. P., and Khullar, D. (2019). Deep learning in medicine—promise, progress, and challenges. *JAMA internal medicine*, 179(3):293–294.

[Woodland et al., 2022] Woodland, M., Wood, J., Anderson, B. M., Kundu, S., Lin, E., Koay, E., Odisio, B., Chung, C., Kang, H. C., Venkatesan, A. M., et al. (2022).

Evaluating the performance of stylegan2-ada on medical images. In *Simulation and Synthesis in Medical Imaging: 7th International Workshop, SASHIMI 2022, Held in Conjunction with MICCAI 2022, Singapore, September 18, 2022, Proceedings*, pages 142–153. Springer.

[Yang et al., 2021] Yang, H., Lu, X., Wang, S.-H., Lu, Z., Yao, J., Jiang, Y., and Qian, P. (2021). Synthesizing multi-contrast mr images via novel 3d conditional variational auto-encoding gan. *Mobile Networks and Applications*, 26(1):415–424.

[Yang and Shami, 2020] Yang, L. and Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316.

[Yazici et al., 2020] Yazici, Y., Foo, C.-S., Winkler, S., Yap, K.-H., and Chandrasekhar, V. (2020). Empirical analysis of overfitting and mode drop in gan training. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1651–1655. IEEE.

[Yi et al., 2019] Yi, X., Walia, E., and Babyn, P. (2019). Generative adversarial network in medical imaging: A review. *Medical image analysis*, 58:101552.

[Yu et al., 2018] Yu, B., Zhou, L., Wang, L., Fripp, J., and Bourgeat, P. (2018). 3d cgan based cross-modality mr image synthesis for brain tumor segmentation. In *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pages 626–630. IEEE.

[Zhou, 2021] Zhou, Z.-H. (2021). *Machine learning*. Springer Nature.

Appendices

- Code available on Github: Link
- Synthetic datasets available on Google Drive: Link