YEDITEPE UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

UNSUPERVISED ANOMALY DETECTION ON BRAIN FRACTIONAL ANISOTROPY MAPS BY INPAINTING WITH DENOISING DIFFUSION PROBABILISTIC MODELS

A Thesis Submitted

by

Burhan Yusuf Arat

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

in

Biomedical Engineering

Supervisor

Assist. Prof. Dr. Andaç Hamamcı

Istanbul - 2023

UNSUPERVISED ANOMALY DETECTION ON BRAIN FRACTIONAL ANISOTROPY MAPS BY INPAINTING WITH DENOISING DIFFUSION PROBABILISTIC MODELS

by

Burhan Yusuf Arat

Approved by:

Assist. Prof. Dr. Andaç Hamamcı
(Yeditepe University) .....
(Thesis Supervisor)

Assoc. Prof. Dr. Dionysis Goularas
(Yeditepe University) .....

Assist. Prof. Dr. Dilek Göksel Duru
(Turkish-German University) .....

DATE OF APPROVAL: .... / .... / 20....

DECLARATION OF ORIGINALITY

I hereby declare that this thesis is my own work and that all information in this thesis has been obtained and presented following academic rules and ethical conduct. I have fully cited and referenced all material and results as required by these rules and conduct, and this thesis study does not contain any plagiarism. The necessary permissions have been obtained if any material used in the thesis requires copyright. No material from this thesis has been used to award another degree.

To the best of my knowledge and belief, it contains no material previously published or written by another person nor material accepted for the award of any other degree except where due acknowledgment has been made in the text.

I accept all kinds of legal liability that may arise in cases contrary to these situations.

Burhan Yusuf Arat

Signature

ABSTRACT

UNSUPERVISED ANOMALY DETECTION ON BRAIN FRACTIONAL ANISOTROPY MAPS BY INPAINTING WITH DENOISING DIFFUSION PROBABILISTIC MODELS

Anomaly detection constitutes a critical area of investigation across various research domains. Recent advancements in deep learning have facilitated the application of neural networks for anomaly detection. However, the high costs and time-intensive nature of data collection and annotation have driven the emergence of unsupervised learning techniques. Within the realm of unsupervised learning, generative models have gained prominence, enabling the generation of diverse data for training purposes. In the context of medical imaging, specifically brain MRIs, generative models have been utilized to detect anomalies in 2D slices of 3D volumes, typically employing structural modalities like T1-weighted or T2-weighted images.

This study employs a generative Denoising Diffusion Probabilistic Model (DDPM) to produce 3D partial volumes of brain Fractional Anisotropy (FA) maps by applying inpainting in inference, which are subsequently masked with inpainting masks to generate the full 3D volume. Anomalies are identified by assessing the disparity between the model's input and output. The performance of the generative model is evaluated using metrics such as Structural Similarity Index (SSIM), Peak Signal-to-Noise Ratio (PSNR), Maximum Absolute Difference, and Average Percentage Error. Anomaly detection accuracy is assessed using Intersection over Union (IoU) and F1 score. The proposed model achieved 0.05 IoU and 0.095 F1 score. The methodology also compared with prior work and it has been shown that while prior works achieved 50% DICE score, while inpainting in inference achieved 39.01% DICE score.

Keywords: Unsupervised anomaly detection, Generative networks, Denoising diffusion probabilistic model, Conditional denoising diffusion probabilistic model, 3D FA maps

ÖZET

GÜRÜLTÜ GIDERICI DIFÜZYON OLASILIK MODELLERI ILE İÇ BOYAMA YOLUYLA BEYIN FRAKSIYONEL ANIZOTROPI HARITALARINDA DENETIMSIZ ANORMALLIK TESPITI

Anomali tespiti, çeşitli araştırma alanlarında kritik bir inceleme alanını oluşturmaktadır. Derin öğrenmedeki son gelişmeler, anormallik tespiti için sinir ağlarının uygulanmasını kolaylaştırmıştır. Ancak, veri toplama ve etiketleme işleminin yüksek maliyetleri ve zaman alıcı doğası, denetimsiz öğrenme tekniklerinin ortaya çıkmasına yol açmıştır. Denetimsiz öğrenme alanında, üreteci modeller, eğitim amaçları için çeşitli veri üretmeyi mümkün kılan bir önem kazanmıştır. Tıbbi görüntüleme bağlamında, özellikle beyin MR'ları, genellikle T1 ağırlıklı veya T2 ağırlıklı görüntüler kullanılarak 3D hacimlerin 2D kesitlerinde anormallikleri tespit etmek için üreteci modeller kullanılmıştır.

Bu çalışma, DDPM kullanarak, girişte inpainting uygulayarak beyin Fraksiyonel Anizotropi (FA) haritalarının 3D kısmi hacimlerini üretmekte ve ardından iç boyama maskeleri ile tam 3D hacimleri oluşturmaktadır. Anormallikler, modelin giriş ve çıkış arasındaki farkı değerlendirerek tanımlanır. Üreteci modelin performansı, SSIM, PSNR, Maksimum Mutlak Fark ve Ortalama Yüzde Hata gibi ölçütler kullanılarak değerlendirilir. Anormallik tespit doğruluğu, IoU ve F1 skoru kullanılarak değerlendirilmiştir. Önerilen model, 0.05 IoU ve 0.095 F1 skoru elde etmiştir. Önerilen metod, önceki çalışmalarla karşılaştırılmış ve önceki çalışmaların %50 DICE skoru elde ettiği, çıkarsama esnasında uygulanan iç boyama metoduyla ise %39.01 DICE skoru elde ettiği gösterilmiştir.

ACKNOWLEDGEMENTS

It is with immense gratitude that I acknowledge the support, help and guidance of my thesis supervisor Assist. Prof. Dr. Andaç Hamamcı. Pursuing my thesis under his supervision has been an experience which broadens the mind and presents an unlimited source of learning.

I also would like to thank Selahattin Serdar Helli for emotional support, exchanging ideas and providing each other new point of views in the field.

Finally, I would like to thank my family and my girlfriend for their endless love and support, which makes everything more beautiful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

VAE	Variational auto-encoder
GAN	Generative adversarial networks
DDPM	Deniosinig diffusion probabilistic model
MSE	Mean squared error
JPEG	Joint Photographic Experts Group
LPIPS	Learned Perceptual Image Patch Similarity
FID	Frechet inception distance
MRI	Magnetic Resonance Imaging
AAE	Adversarial auto-encoder
WGAN	Wasserstein generative adversarial network
OCT	Optical Coherence Tomography
AUPRC	Area under precision-recall curve
CAM	Class activation maps
SSIM	Structural similarity index measure
PSNR	Peak signal-to-noise ratio
DDIM	Denoising diffusion implicit model
LDM	Latent diffusion model
VQ-VAE	Vector-quantized variational auto-encoder
IoU	Intersection over union
AUC	Area under curve
BraTS	Brain Tumor Segmentation Challenge
DAE	Denoising auto-encoder
CT	Computed Tomography
3D	3-Dimensional
iCAIRD	The Industrial Centre for Artificial Intelligence Research in Digital Diagnostics
2D	2-Dimensional
dMRI	Diffusion-weighted Magnetic Resonance Imaging
FA	Fractional anisotropy

PDF	Probability density function
HCP	Human Connectome Project
Gb	Gigabytes
GPU	Graphical processing unit
TRUBA	Türk Ulusal Bilim Altyapisi
RAM	Random acess memory
FLAIR	Fluid attenuated inversion recovery

$\mathbf{q}(\mathbf{x}_t \vert \mathbf{x}_{t-1})$	Forward diffusion process
$\mathbf{p}_\theta(\mathbf{x}_{t-1} \vert \mathbf{x}_t)$	Reverse diffusion process
T	Sampling timeteps

1. INTRODUCTION

Anomaly detection is one of the important problems studied by various research areas. Due to the applicable nature of anomaly detection algorithms for diverse application domains, it has been an active research field for more than half a century[1]. Anomaly detection, outlier detection, or novelty detection is the operation of processing data and pinpointing the data points which deviate from the data's distribution[2]. To name a few application fields, fraud detection in finance, intrusion detection in cyber-security, traffic monitoring, sensor networks, and medical anomaly detection. In time, different approaches have been made for anomaly detection in various fields. The basic approaches can be named under two groups: proximity-based and projection-based. Nearest neighbor and clustering algorithms rely on estimating an anomaly's proximity to nearby data points and thus can be viewed under proximity-based methods. Whereas projection-based methods rely on projecting the data into a lower dimension while carrying out the relationship between the data points[3].

With the recent advancements in deep learning, the ability to understand expressive representations of complex data by deep learning models have emerged. Which opens a new field under anomaly detection as "deep anomaly detection.". Deep anomaly detection consists of three fundamental groups, which are; deep learning for feature extraction, learning feature representations of normality, and end-to-end anomaly score learning[2]. In deep learning for feature extraction, the aim of using the deep learning models is to provide a mapping from higher dimensions to lower dimensions and concurrently extract features. The anomaly detection and feature extraction parts take place independently; thus, deep learning models are solely used for the purpose of dimensionality reduction[2]. End-to-end anomaly score learning is the group of algorithms which are simultaneously extracting feature representations while learning scoring anomalies; by processing the pipeline in an end-to-end fashion, it has been realized that by such pipeline, the anomaly scoring is optimized substantially[2]. By learning feature representation of normality, a domain-based approach is mentioned. This method is mostly used when the data for the anomaly is scarce, and the deep learning model is trained with a dataset that defines the normal distribution. The similarity of the data containing anomalies and the data acquired by the deep learning model is then compared to detect anomalies[4].

Also, deep anomaly detection can be grouped under training strategies which are; supervised learning and unsupervised learning[5]. In the supervised learning strategies, the training data is provided to the model with the labels of the anomalous regions to enable learning of the model for the anomalous regions. For unsupervised learning, there exists no labeled data for the training process, and in the training process, the model never meets with any data that contains an anomaly[6].

Image inpainting is the process of modifying images in such ways that an observer who hasn't observed the original image would not understand the difference[7]. Despite its ancient history, inpainting is still an active research area. Before the arrival of the Information Age, image inpainting emerged by artists who reimposed damaged paintings or photographs with small defects to the original image as closely as possible by hand[8]. With the information age, the meaning of image inpainting is also redefined into technological categories where an incomplete image's damaged parts are restored at the pixel level[9]. Image inpainting regarding the approach employed can be divided into two groups which are traditional-based methods and deep learning-based methods[10]. While traditional-based methods rely on the approach for filling gaps in the original image, deep learning-based methods can be categorized through multiple perspectives. Those multiple perspectives can be observed under three categorization groups which are inpainting strategies, network structures, and loss functions used[11]. Although deep learning-based image inpainting methods are mostly studied with Generative Adversarial Networks(GAN), Auto-Regressive models, and Variational Auto-Encoders(VAE) models, lately, with the emergence of the DDPM in the image synthesis, DDPM's are introduced to image inpainting algorithms[12].

DDPMs are in a family of generative probabilistic models. Chronologically, firstly diffusion probabilistic models are proposed where parameterized Markov chains are used to gradually convert one distribution into another one after finite time[13]. In the diffusion probabilistic models, the aim is to disturb input data through the forward diffusion process by iteratively adding small amounts of Gaussian noise through Markov chains, eventually generating a complete noise. A learned reverse-diffusion process through the same Markov chain in the opposite direction begins from Gaussian noise, gradually denoising to generate a desired sample that can fit in the training dataset [14]. It has been found that diffusion probabilistic models are dependent on each forward diffusion process step and reverse diffusion process,

which results in very long sampling operations. In DDPM, it has been proposed to speed up the diffusion processes by skipping intermediate steps. Also, a simplified training objective for training is proposed[15].

In this study a DDPM is trained with healthy samples of FA maps to generate complete 3D volume of brain FA maps. At the inference step, the anomaly bearing input is processed with inpainting volume masking and the pre-trained model then used to generate masked volumes of FA maps with inpainting operation. The resulting volumes are then gathered to generate a pseudo-healthy FA map and the difference between input and output results an anomaly map as can be seen from Figure 1.1.

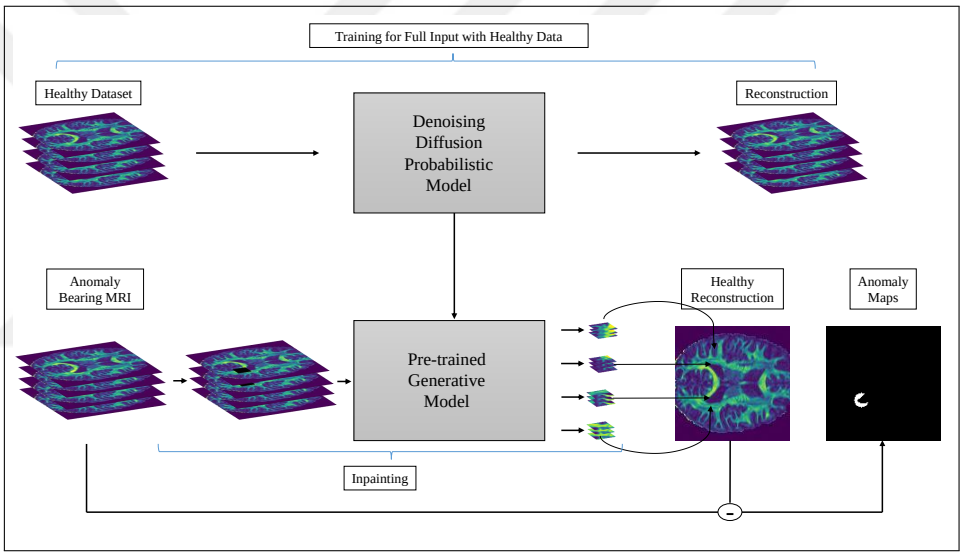

Figure 1.1. An overview of the proposed method

1.1. PROBLEM STATEMENT

In the medical imaging domain, a classical approach for the image to image translation is altering a medical image or 3D volume from a patient to an image or 3D volume without pathologies. Anomaly detection is a sensitive task, so in the transformed 3D volume, it is crucial that only the volume containing the pathology has to be changed. By utilizing 3D FA volumes, it is possible to detect various neurological diseases. While generating an anomalous region of volume from 3D FA maps, it is crucial to blend in the correct anatomy of the region of volume, as it was not bearing any anomalies at the beginning. To detect

correct regions as anomalies, preserving the correct anatomy for the rest of the brain is also essential.

1.2. RELATED WORKS

1.2.1. Image Inpainting Based on Generative Networks

GAN[16] and VAE[17] made an enormous impact on image-to-image synthesis. In the task of image inpainting, generative networks such as GAN and VAE tackled the problem of learning high and low-frequency feature information. With the aid of adversarial losses, consistency between fake and real pixels are increased[8]. With an encoder-decoder and a fully-connected discriminator architecture, [18] were able to predict the missing regions of an image by conditioning it with its neighboring pixels.

From a different perspective, by exploiting StyleGAN as a pre-trained generator, manipulating inputs through an implemented encoder successfully paints missing regions. The proposed model has been achieved in generating images in terms of 0.03 Mean Squared Error(MSE), 0.17 Learned Perceptual Image Patch Similarity(LPIPS) and 0.56 Perceptual similarity and the inpainting results discussed qualitatively[19].

DDPMs' performance in unconditional image generation has been seen on a level with state-of-the-art methods. A general framework for image-to-image translation tasks(inpainting, colorization, JPEG restoration, and uncropping), Palette[20], is proposed. An image-conditional DDPM is trained to pinpoint missing regions of free-form masks. The model achieved a 15.78 Frechet Inceiption Distance score(FID), 200.8 inception score, and 46.2 perceptual distance, which shows promising results for in painting task. In [21], a pre-trained unconditional DDPM is employed, and a novel method for conditioning the sampling algorithm is proposed. The condition is provided through the proposed method, "Iterative Latent Variable Refinement (ILVR)," where in each generation step, the starting noise distribution is infused with the lower frequency information of the condition. Since in the inpainting task, the masked region occurs as an unknown region, providing low or high-frequency information will be impossible to provide for the unknown region[12]. The

authors of [12] applied a conditioning mechanism depending on the known regions of input solely, which covers the whole sampling process. The method employs an unconditional pre-trained DDPM. While sampling, to harmonize the generated unknown region, it has been proposed to move back and forth through the reverse process to improve sampling quality.

1.2.2. Unsupervised Anomaly Detection in Medical Domain

Due to expensive costs and the burden of time for collecting data or annotating data, unsupervised learning methods emerged as an alternative approach to supervised learning methods. With the advantage of unsupervised deep generative learning, unsupervised anomaly detection has been rapidly implemented in various fields in the medical domain. The main idea is to train a deep generative network to model healthy anatomy as closely as possible. Then the trained model can be used to detect, localize, and segment anomalous regions from an image space[22].

In [23], two relatively close auto-encoder structure is employed to detect lesions from brain MRIs. Namely, Adversarial Auto-Encoder(AAE) and VAE are compared for their performance of reconstruction quality. It has been shown that AAEs have shown better AUC performance. In [24], a GAN framework named AnoGAN is employed to model the healthy anatomy of OCT scans. Later on, a Wasserstein Generative Adversarial Network (WGAN) named f-AnoGAN is employed for the same task with improvements in the GAN architecture. An encoder is also added in the architecture after the training of WGAN to control the generation process since GANs only generate from a random noise vector, with a controllable latent space that will be able to control the generation to preserve the anatomical features[25]. In [26], VAE and GAN architectures are mixed up on brain MRI datasets to control the generation named AnoVAEGAN. The proposed model achieved a 0.60 Dice score, while Anogan achieved a 0.37 Dice score. In [27], it has been proposed that by training a U-Net architecture while adding random noise in the input, the model learns to denoise anomalous noisy parts. The proposed denoising auto-encoder achieves 0.773 Dice score and 0.833 AUPRC. In [11], the inpainting process is proposed to be operated on feature level instead of pixel level on a chest x-ray dataset. In the study, a feature extractor is connected to a teacher-student generator architecture. In the teacher

generator part, the image is directly reconstructed from features, while the student generator generates from features that are augmented by an inpainting block. By trained under coupled knowledge distillation and adversarial learning, including a discriminator network. After the training, the discriminator is also used to detect anomalies. In [28], it has been proposed to navigate the unsupervised segmentation process with class activation masks (CAM). The authors implemented a VAE with additional constrained loss applied to the model to generate grad-CAMs which cover global features of the whole input, and grad-CAMs are forced to be activated around anomalous regions. During inference, generated grad-CAMs are thresholded to produce the final segmentation mask. It has been proposed that with inpainting, detecting anomalies on T1 weighted brain MR images by generating the inpainted parts with the help of a modified GAN architecture[29]. The proposed model never faces any anomalous data while adversarially training to reconstruct missing regions. During inference, it is proposed that for a given slice, a sliding window algorithm is applied, which slides the mask, and model paints each region accordingly, with resulting predictions, a heatmap indicating the region of interest for the highest reconstruction loss. The proposed model is evaluated with different sizes of masks given as 8, 16, 32, 64, and the model achieves the best performance of SSIM in mask size 8 with 0.99, the best PSNR value at mask size 64 with 31.665 and the best DICE score of 0.77 for segmenting anomalous regions in the mask size 32. In [30], a two-step generative anomaly segmentation model is proposed. In the first step, potential anomalous regions are specified by employing a VAE to generate masks that cover those potential anomalous regions. The generated mask is then applied to the input and fed into a GAN, which is employed as an inpainting generative network to refine the masked input back to generate a healthy version of the input by utilizing the unmasked region. The model is trained using two different datasets, FastMRI+[31] and IXI datasets, to capture the healthy distribution and evaluated on large stroke T1-weighted MRI dataset ATLAS v2.0. The proposed model shows 22% better AUPRC and 16% better DICE scores on anomaly segmentation performance against AnoDDPM.

Diffusion models are also used in unsupervised anomaly detection in the medical imaging domain. In [32], a weakly-supervised approach has been proposed. Denoising Diffusion Implicit Model(DDIM) is trained with classifier guidance on both healthy and anomaly-bearing subjects. Classifier guidance allows the sampling process guided by the

learned parameters of a pre-trained classifier to generate a healthy medical image for a given anomaly-bearing medical image[33]. In [34], a DDPM model named AnoDDPM is trained only on healthy subjects with brain MRIs. In the iterative steps of adding noise and denoising during training and sampling, instead of Gaussian noise, it has been proposed to exchange with simplex noise. It has been shown that against f-AnoGAN, AnoDDPM achieved better segmentation performance of anomalous regions. While AnoDDPM achieved 0.383 Dice score, 0.269 IoU, 0.863 AUC, and f-AnoGAN achieved 0.128, 0.093, and 0.789, respectively. In [35], a Latent Diffusion Model(LDM) is employed to detect and segment anomalies. First, a Vector-Quantized Variation Auto-Encoder(VQ-VAE) is trained to downsample inputs into quantized latent representations, and the DDPM model is trained on produced latent representations. By averaging samples of each step in the reverse process and applying a pre-defined threshold map, a binary map of the anomalous regions is generated[14]. In [36], a diffusion model trained on healthy samples from two different datasets for 3D Computed Tomography(CT) scans, namely iCAIRD GG&C NHS and 2D MRI slices dataset, namely BraTS21 dataset, is used to detect anomalies by two different inference techniques proposed in [34] and [35]. While training the model, three different noise types, namely Gaussian noise, simplex noise, and proposed coarse noise, are investigated. It has been shown that a diffusion model or DAE[27] trained with proposed coarse noise outperforms both simplex noise and Gaussian noise in AURPC values. Also, the proposed inference techniques in [34] and [35] also compared with DAE model proposed in [27] and showed that for anomaly detection, DAE outperforms diffusion model inference techniques in 0.653,0.689 and 0.833 AUPRC values and 0.610, 0.675 and 0.773 respectively. In [37], a patched diffusion model is proposed. While training the model, patches are either generated from a fixed grid or randomly, and noise is added only on the chosen patch, then the denoising operation is held only on the given patch. During inference, for each patch set covered, the whole input is generated individually with respect to their original positions, and the whole output is generated from reconstructed patches. The model is trained with healthy samples from the IXI dataset with paired T1 and T2 weighted brain MRI and evaluated on both BraTS21 and MSLUB datasets. The PatchedDDPM model is compared against DAE,f-Anogan, and AnoDDPM, and the DICE scores are 49.00%, 37.05%, 24.16%, 40.67%,respectively while AUPRC scores are shown as 54.07%, 44.99%, 22.05% and 49.78%,respectively on the BraTS21 dataset. For the MSLUB dataset, DICE scores are 10.35%, 3.56%, 4.18%, 6.42% respectively and AUPRC scores are

9.79%, 5.35%, 4.01% and 7.44%, respectively.

1.3. BACKGROUND

1.3.1. Fractional Anisotropy Maps

Diffusion-weighted MRI(dMRI) is an MRI sequence that has been increased its popularity due to the large variety of information acquired by endogenous diffusion characteristics of water molecules within a region, but mostly used for neuroimaging purposes[38]. FA is a scalar value that can be drawn out from a diffusion tensor to display the movement of the water molecules in a given region[39]. To generate FA maps, each FA value is assigned to the corresponding voxel at the macro level creating a map of the diffusion pattern of water molecules, allowing the detection of microstructural details of normal or altered anatomy[40].

FA maps for brain MRIs are used to diagnose neurological diseases such as; Multiple Sclerosis(MS)[41], Alzheimer's Disease(AD)[42], Parkinson's Disease(PD)[43] and traumatic brain injuries which affects the white matter tracts[44].

1.3.2. Denoising Diffusion Probabilistic Models

DDPMs are probabilistic generative models encouraged by the idea of Non-equilibrium Thermodynamics[13]. The general idea behind the diffusion models is; for a received input, gently and iteratively adding small amounts of Gaussian noise in each timestep $T \rightarrow \{1, \ldots, T\}$ to get a series of noisy images $\{x_0, x_1, \ldots, x_T\}$. To name the given process as Forward Diffusion process $\mathbf{q}$, it is defined as;

$$\mathbf{q}(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}; \beta_t \mathcal{I}) \tag{1.1}$$

Where $\mathcal{I}$ is denoted as Identity matrix, β_t as the diffusion rate or noise variance in each step[32]. The forward diffusion process is a Markov Chain that is fixed and can be known beforehand. It can also be written explicitly as;

$$\mathbf{x}_t = \sqrt{\bar{\alpha}}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon \tag{1.2}$$

Where $\alpha_t \equiv 1 - \beta_t$ and $\overline{\alpha}_t \equiv \prod_{i=1}^{T} \alpha_i$ denotes the cumulative product of α from $T \rightarrow \{1, \ldots, T\}$[15].

The reverse Diffusion process or denoising process is the operation to generate samples from complete Gaussian noise. It has been shown that the reverse diffusion process has the same functional form as the forward diffusion process[45]. The reverse diffusion process or denoising process starts when the forward diffusion process ends and follows the forward diffusion process in the opposite direction. The reverse diffusion process is also Markov-Chain and learned by training neural networks parameters θ, $\mathbf{p}_\theta$ given by;

$$\mathbf{p}_\theta(\mathbf{x}_{0:T}) \equiv \prod_{t=1}^{T} \mathbf{p}_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{1.3}$$

where $\mathbf{p}_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ can be written explicitly as;

$$\mathbf{p}_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \equiv \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t); \Sigma_\theta(\mathbf{x}_t, t)) \tag{1.4}$$

Training loss for the diffusion models relies on "maximizing the log-likelihood" of a sample s' generated by the model belonging to the $\mathbf{D}^{training}$ .Both the forward diffusion process and reverse diffusion process in the diffusion models rely on Gaussian transition functions, as can be seen from Equation (1.4). To maximize the log-likelihood of Gaussian distributions, the employed neural network model is being trained to find the parameters of the distribution μ and σ^2 on each timestep generated sample that is feasible for mapping on the $\mathbf{D}^{training}$.

$$\mathscr{L}_{simplified} \equiv \mathbf{E}_{t,\mathbf{x}_0,\epsilon}[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2] \tag{1.5}$$

Given simplified loss term $\mathscr{L}_{simplified}$ is used to estimate the Mean Squared Error between the added noise ϵ in the forward diffusion process and the predicted noise from the model in the reverse diffusion process ϵ_θ while training the model[15].

1.3.3. Conditional Image Synthesis

A generative model can be accepted as a conditional generative model if an extra input called a condition is provided to the generative network. With the provided condition, the model is able to infuse the generative process with the condition to direct the generation process[46]. Condition is used to control the stochastic behavior of the generative models. Conditional Image Synthesis can be applied to many different tasks depending on the type of condition

data. Condition data can be text, semantic maps, or any image-to-image translation tasks[47].

It has been proposed by [48] to control the generative process of the Diffusion Models; the U-Net model is expanded with the cross-attention mechanism. The conditional input is projected to an intermediate space by a transformer τ_θ to support various input modalities. The projected inputs are then mapped to various interjacent layers of the U-Net.

Implementation of the cross attention is as follows; $Attention(Q,K,V) = softmax(\frac{\mathbf{QK^T}}{\sqrt{d}}) \cdot \mathbf{V}$ where $\mathbf{Q}$ is features received from the U-Net model while $\mathbf{K}$ and $\mathbf{V}$ are received from transformer τ_θ which are conditional input[48].

With the conditioning term, the loss function for DDPM is redefined. Let conditional input be y, the joint optimization of τ_θ and ϵ_θ is as follows;

$$\mathscr{L}_{simplified} \equiv \mathbf{E}_{t,\mathbf{x}_0,\epsilon,\mathbf{y}}[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t, \tau_\theta(y)\|^2] \tag{1.6}$$

1.4. CONTRIBUTION OF THE THESIS

To best of our knowledge, so far researches has been made with T1-weighted, T2-weighted and FLAIR MR modalities. The models are trained in slice-wise fashion and models are used in the inference time with the pipeline the model is trained within. In this study it is aimed to;

- A DDPM model is pre-trained to generate healthy counter-part of the input. During inference, with inpainting method implemented, each inpainting region is re-generated from the pre-trained model and a resulting pseudo-healthy MR image is obtained.

- DDPM model is trained with healthy brain FA maps, to detect neurological anomalies. To best of our knowledge, none of the previous studies utilized an FA dataset.

- The model is trained with a 3D dataset, and data is fed into model not in a slice-wise fashion but as complete 3D inputs.

2. METHODOLOGY

Given a set of fractional anisotropy (FA) volumes of N healthy subjects $\mathbf{D}^H = \{\mathbf{s}^1, \mathbf{s}^2, \ldots, \mathbf{s}^N\}$ which are sampled from the set of healthy subjects $\mathbf{H}$. Let $s_{i,j,k} \in [0, 1]$ be the fractional anisotropy value at i,j,k voxel. The data is scaled between [-1,1] by $2 \times (s_{i,j,k}) - 1$. An unconditional generative DDPM is trained to generate $\mathbf{s}'$ such that $\mathbf{s}' \sim \mathbf{p}_\theta$ where $\mathbf{p}_\theta(\mathbf{H})$ is a probability density function (PDF) learned by the model to map volumes consisting of random Gaussian noise $n_{i,j,k} \sim N(0, 1)$ to $\mathbf{D}_\theta^\mathbf{H}$. The trained model learns to generate an element of $\mathbf{D}_\theta^\mathbf{H}$ by an iterative process of T steps. At each iteration, the trained model predicts the noise to be removed so that the iterations converge to an element of $\mathbf{D}_\theta^\mathbf{H}$.

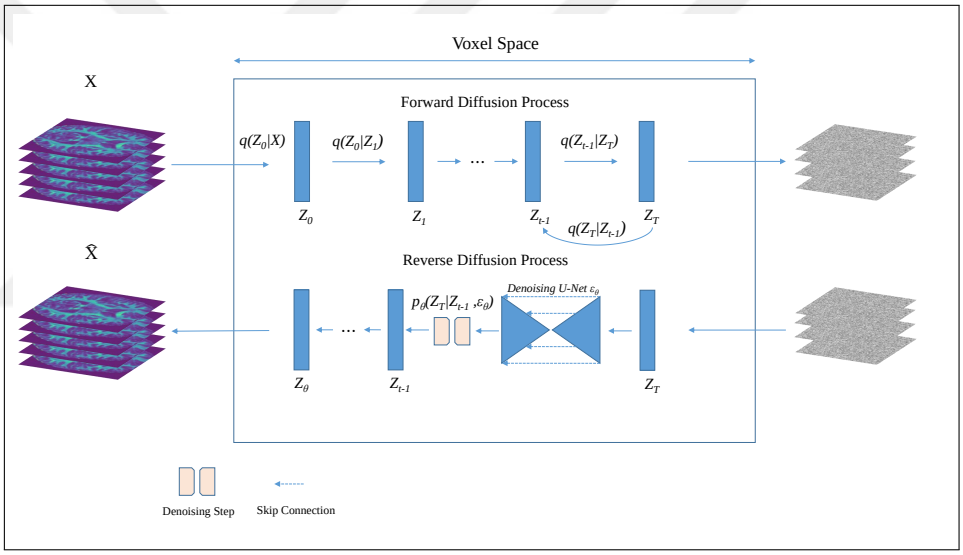

Figure 2.1. Unconditional Diffusion Model

For the conditional DDPM, to provide conditional information from $\mathbf{D}^H$ white matter masks $\mathbf{Y} = y^1, y^2, \ldots, y^N$ are extracted. Let $y_{i,j,k} \in [0, 1]$,$y_{i,j,k} = 1$ denotes the white matter pathways, and rest is denoted as $y_{i,j,k} = 0$. The conditional DDPM model is trained with $\mathbf{D}^H$, and $\mathbf{Y}$ is provided to model through intermediate layers of U-Net architecture as the condition.

$$y_{i,j,k} = \begin{cases} 1 \; if \; s_{i,j,k} > 0.2 \\ 0 \; if \; s_{i,j,k} < 0.2 \end{cases} \tag{2.1}$$

Given an FA volume $\mathbf{s}$, a subset of voxels $\mathbf{m} \subset \mathbf{s}$ are labeled as a mask. A volume x^T is

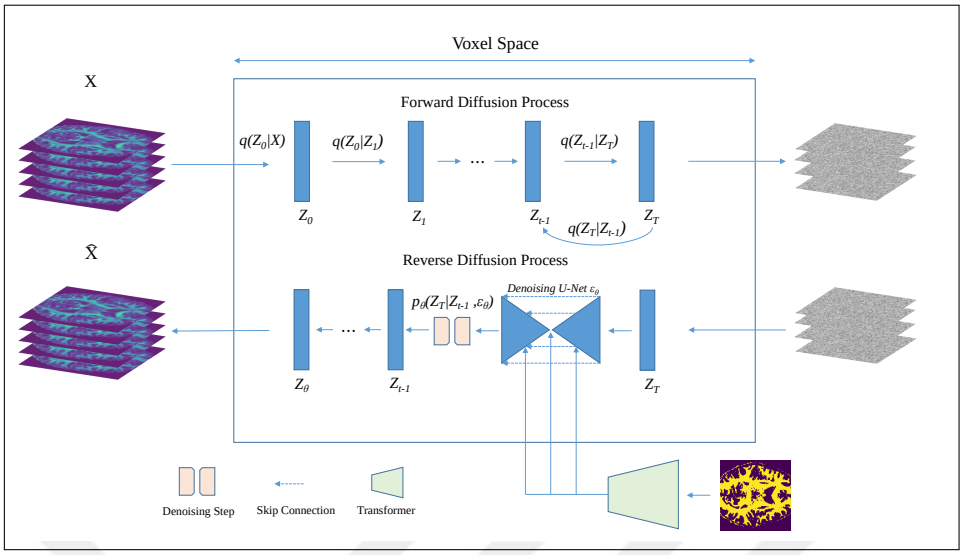

Figure 2.2. Conditional Diffusion Model

initialized as random Gaussian noise $x^T_{i,j,k} \sim N(0,1)$. $\mathbf{m}$ corresponds to the mask for the masked region in a sample $\mathbf{s}$ and $\mathbf{s} - \mathbf{s} \cdot \mathbf{m}$ denotes the rest of the sample $\mathbf{s}$ after masking. In the inference, the model is provided with $\mathbf{m}$ and $\mathbf{s} - \mathbf{s} \cdot \mathbf{m}$. At the first step, masked region $\mathbf{m}$ is filled with Gaussian noise, while $\mathbf{s} - \mathbf{s} \cdot \mathbf{m}$ is filled with pre-defined noise at each step of the iterative process.

$$x^{t-1}_{i,j,k} = \begin{cases} x^T_{i,j,k} \; if \; s_{i,j,k} \in m_{i,j,k} \\ s^T_{i,j,k} \; if \; s_{i,j,k} \notin m_{i,j,k} \end{cases} \tag{2.2}$$

Algorithm 2.1. Training

1: Input: $\mathbf{s} \sim \mathbf{D^H}$
2: **repeat**
3: $\quad \mathbf{t} \longrightarrow 1, 2, \ldots, t-1, T$
4: $\quad \mathbf{s_{t-1}} \sim \mathbf{q(s|s_{t-1})}$
5: $\quad$ *Let n be the noise added to convert* $\mathbf{s}$ *to* $\mathbf{s_{t-1}}$
6: $\quad \mathbf{n} \sim \mathcal{N}(0, \mathcal{I})$
7: $\quad$ *Take gradient descent steps*
8: $\quad \nabla_\theta \left\| \mathbf{n} - \mathbf{n}_\theta(\mathbf{x}_t, t) \right\|^2$ ▷ For unconditional DDPM
9: $\quad \nabla_\theta \left\| \mathbf{n} - \mathbf{n}_\theta(\mathbf{x}_t, t, \tau_\theta(y) \right\|^2$ ▷ For conditional DDPM
10: **until** *converged*

At each step, providing $\mathbf{s} - \mathbf{s} \cdot \mathbf{m}$ by adding pre-defined noise, and $\mathbf{m}$ is predicted by the

model,$\mathbf{s} - \mathbf{m}$ and $\mathbf{m}$ are combined to generate new sample $\mathbf{x_{t-1}}$.

$$\mathbf{s} - \mathbf{s} * \mathbf{m} \sim \mathcal{N}(\sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1 - \overline{\alpha}_t)\mathbf{I}) \tag{2.3}$$

$$\mathbf{m} \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, \mathbf{t}), \Sigma_\theta(\mathbf{x}_t, \mathbf{t}) \tag{2.4}$$

$$\mathbf{x}_{t-1} = \mathbf{m} + (\mathbf{s} - \mathbf{s} \cdot \mathbf{m}) \tag{2.5}$$

In Equation (2.3), $\sqrt{\overline{\alpha}_t}\mathbf{x}_0$ is the mean of the pre-defined noise, and $(1 - \overline{\alpha}_t)\mathbf{I}$ correspond to the covariance of the pre-defined noise, while in 2.4 $\mu_\theta(\mathbf{x}, \mathbf{t})$ and $\Sigma_\theta(\mathbf{x}_t, \mathbf{t})$ is predicted by the model. Equation (2.5) shows the generating of a new sample $\mathbf{x}_{t-1}$ from combining $\mathbf{s} - \mathbf{m}$ and $\mathbf{m}$.

Algorithm 2.2. Regular Inference Algorithm

1: Input: $\mathbf{x}_t \sim \mathbf{N}(0, \mathbf{I})$
2: Output: $\mathbf{x}_0$
3: **repeat**
4: $\quad \mathbf{z} \sim \mathcal{N}(0, \mathcal{I})$ if $t > 1$ else $z = 0$
5: $\quad \mathbf{x}_{t-1}^{unknown} = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, \mathbf{t})) + \sigma_t\mathbf{z}$
6: $\quad \mathbf{x}_{t-1}^{known} = \sqrt{\overline{\alpha}_t}\mathbf{x}_0 + (1 - \overline{\alpha}_t)$
7: $\quad \mathbf{x}_{t-1} = \mathbf{m} \cdot \mathbf{x}_{t-1}^{known} + (1 - \mathbf{m}) \cdot \mathbf{x}_{t-1}^{unknown}$
8: **until** $\mathbf{t} \rightarrow T, t - 1, \ldots, 2, 1$

2.1. INPAINTING

To detect anomalies, $\mathbf{s_{test}}$ are divided into regions of $\frac{\mathbf{H}}{\mathbf{masksize}} x \frac{\mathbf{W}}{\mathbf{masksize}} x \frac{\mathbf{D}}{\mathbf{masksize}}$. In each detection step, a cubic mask is applied to the corresponding region of $\mathbf{s_{test}}$. Each region is iteratively masked, and inpainting operation is held for each region of $\mathbf{s_{test}}$, respectively. While generating the corresponding masked region, the outer part is transferred through noising and denoising operation to the output in inference step, as can be seen from Figure 2.3.

In each iteration the model only generates the masked region, and the result is generated by carrying out the corresponding generations in the corresponding region in the whole 3D volume as can be seen in the Figure 2.4. This process results in a pseudo-healthy generation of the input $\mathbf{s'_{test}}$, an anomaly map is generated between $\mathbf{s_{test}}$ and $\mathbf{s'_{test}}$ by taking the difference.

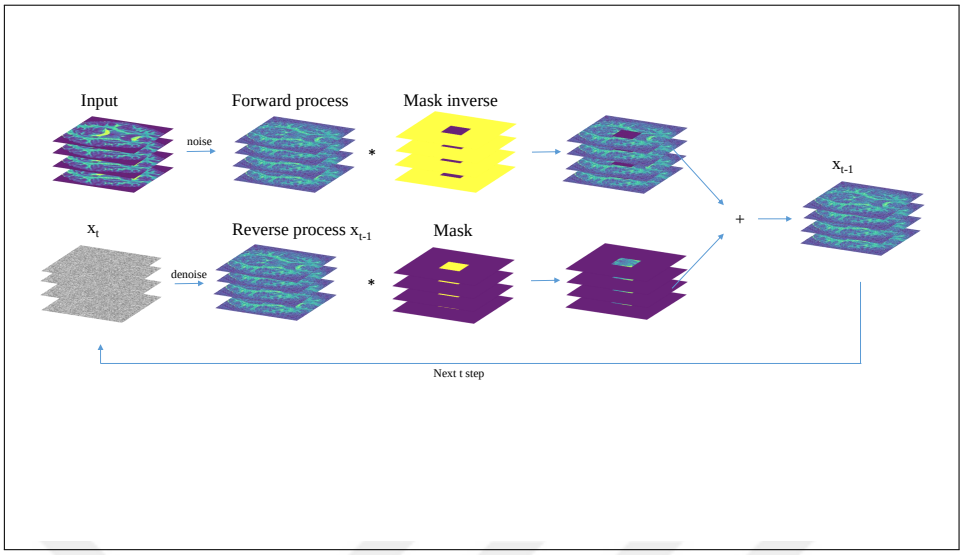

Figure 2.3. Inpainting operation at each timestep t

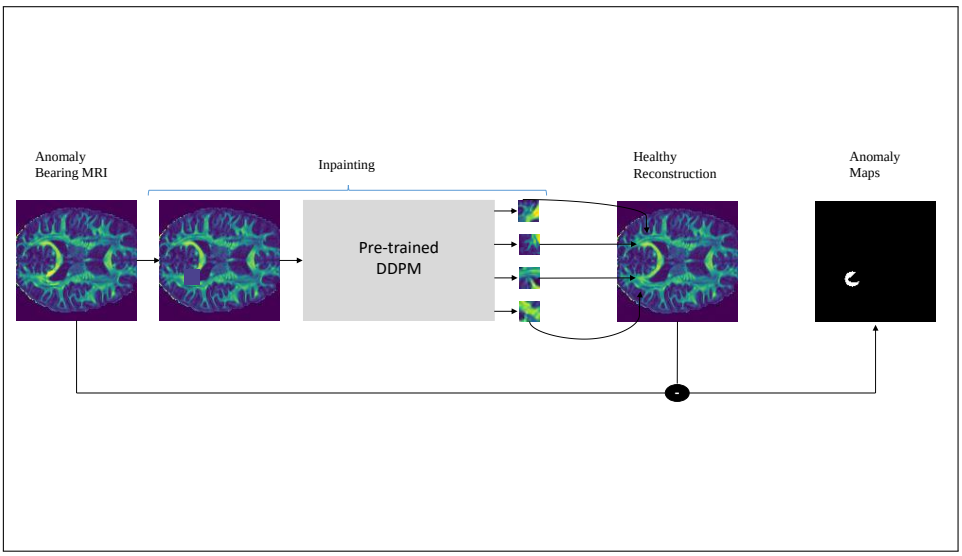

Figure 2.4. Inpainting operation for complete volume

Two distinct types of masks are employed in the inpainting process. First, complete cubic masks are applied on the input MRI image as can be seen from Figure 2.5 and secondly, cubic masks guided by white matter tracts, where the cubic masks and binary maps of white matter tracts intersections are taken out which results in the blocking out the information on the matter tracts as can be seen from the Figure 2.6.

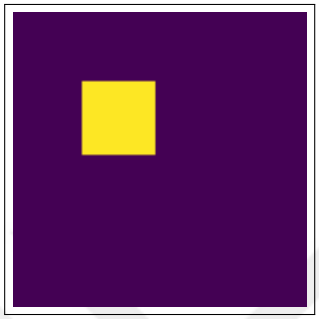

Figure 2.5. Complete cubic mask

Figure 2.6. White matter guided cubic mask

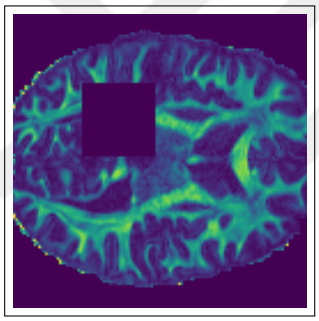

Figure 2.7. Complete cubic mask applied input MRI

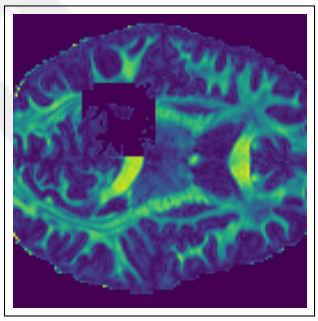

Figure 2.8. White matter guided cubic mask applied input MRI

2.2. DATASET AND PRE-PROCESSING

In this study, 972 MR images acquired with dMRI modalities are used from Human Connectome Project (HCP) 1200 Subject Release[49]. HCP 1200 Subject Release dataset consists of 1000 dMRI of healthy participants aged between 22 and 35. The participants of HCP was chosen from various geographical locations around the world to make sure to provide a diverse representation of the sampling population. HCP provides various structural (T1-weighted - T2-weighted and dMRI) and functional modalities. It has been realized that the dataset contains gaps in each individual data on the x, y and z axes. The original data shape was 145x174x145, and it has been cropped to 128x128x128. The data is split into to

900 train samples and 72 test samples without any annotations.

While comparing the methodologies, as mentioned in the [37] IXI, Multimodal Brain Tumor Segmentation Challenge 2021(BraTS21)[50] and MSLUB[51] datasets are used. IXI dataset consists of 560 pairs of T1 and T2-weighted brain MRI scans. The BraTS21 dataset consists of 1251 unhealthy brain MRI scans of four different modalities such as T1-weighted, T1-CE, T2-weighted, and FLAIR. MSLUB dataset consists of 30 patients with T1-weighted, T2-weighted, and FLAIR modalities, which carry the MS diseases. BraTS21 and MSLUB datasets are used in the evaluation by following the previous work[37] by choosing T2-weighted modalities of all datasets. Pre-processing methods are by following [37]; Registration of all scans on SRI24-Atlas[52] by affine transformations, skull-stripping, black border removal to a fixed resolution of [192 x 192 x 100] voxels and bias field correction is applied. The resolution of the volumes are reduced to [96 x 96 x 80], and the top 15 and bottom 15 slices are removed from the slices which are parallel to the transverse plane.

2.3. NETWORK ARCHITECTURE

In diffusion generative modeling, U-Net architectures are first introduced by Ho et al.[15], and the improvements have been shown by Jolicoeur-Martinaue et al.[53] that U-net like architectures in denoising score matching models, sampling quality drastically improved[54].

In this study, for unconditional generation, to represent the reverse process, a U-Net architecture is employed can be seen in Figure 2.9. As input, 3-D FA maps and time embeddings from DDPM are received. All the layers are arranged to support 3-D inputs. In the contracting path, five repeated down blocks are defined with skip connections from U-Net architecture[55]. Each down block consists of two residual blocks and a downsampling block with the convolution of strides two. In each downsampling block,the number of the features is doubled. Residual blocks consist of two convolution operation, before each convolution operation, swish activation and group normalization are applied. Only the last down block contains self-attention. A middle block between the contracting path and the expansive path is constructed from two residual blocks, and a self-attention block between residual blocks, is used. On the expansive path, five repeated up blocks are defined with skip connections

received from corresponding downblocks. Each up block consists of two residual blocks and an upsampling block with convolution of strides two, in each up block,the number of the features is halved. For the down block,middle block and up block same residual blocks are used. The last layer consists of a group normalization, swish activation, and a final convolution is applied.

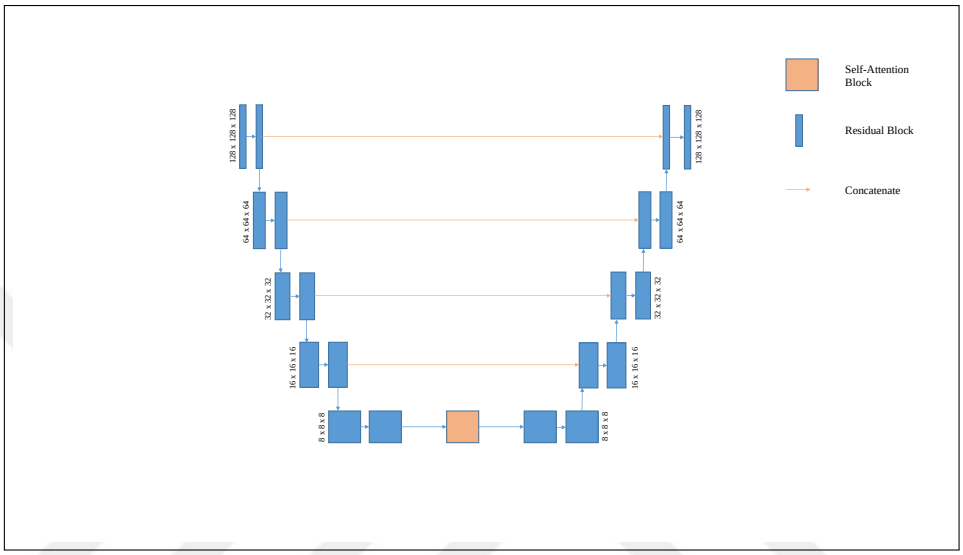

Figure 2.9. Unconditional U-Net

For conditional generation, to represent the reverse process, a U-Net architecture is employed and it can be seen in Figure 2.10. As input, 3-D FA maps and time embeddings are received, and conditional input is received as extra information through a transformer. All the layers are arranged to support 3-D inputs. In the contracting path, five repeated down blocks are defined with skip connections from U-Net architecture[55]. Each down block consist of two residual blocks and a downsampling block with convolution of strides two, in each downsampling block, the number of the features is doubled. Residual blocks consist of two convolution operation, before each convolution operation swish activation and group normalization is applied. Last two down blocks contains cross attention layer through a transformer. A middle block between the contracting path and the expansive path is constructed from two residual blocks and a cross-attention block between residual blocks is used. On the expansive path, five repeated up blocks are defined with skip connections received from corresponding down blocks. Each upblock consists of two residual blocks and an upsampling block with convolution of strides two, in each up block, the number of the features is halved. First two up blocks contains cross attention layer through a transformer. For the down block, middle block

and up block same residual blocks are used. The last layer consists of a group normalization, swish activation, and a final convolution is applied.

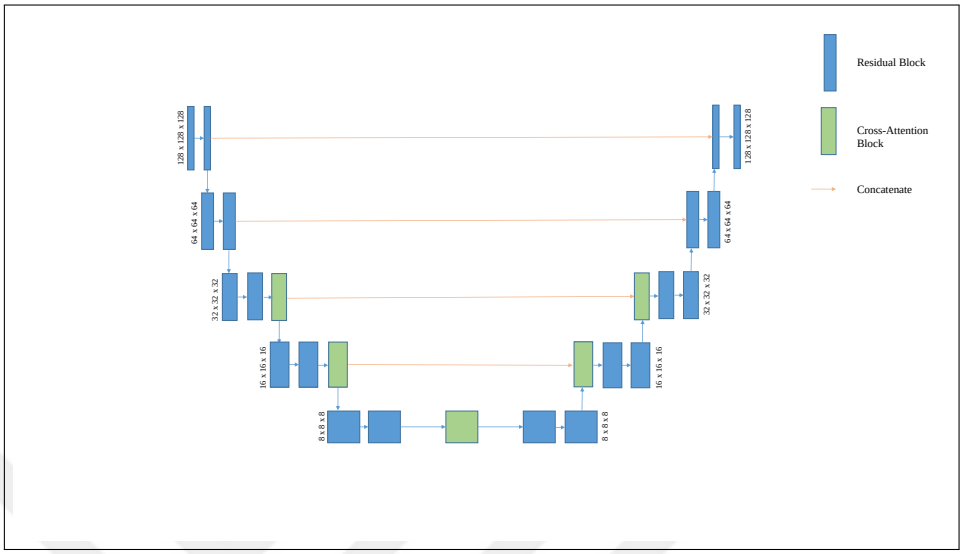

Figure 2.10. Conditional U-Net

2.4. IMPLEMENTATION DETAILS

All training experiments are held on Google Colab Pro + with A100 40 Gb GPU and 83 Gb RAM. Tensorflow version 2.11.0 and Keras 2.8.0 frameworks are used for the DDPM. The model is trained without data augmentation for 400 epochs and 360,000 steps with batch size 1, which took approximately three days for training. DDPM is utilized with $\mathbf{T} = 1000$ sampling steps. The model is trained with Adam optimizer and MSE loss with a learning rate scheduler to decrease the learning rate gradually after 15000 steps and starting from 10^{-4}. By choosing the number of channels for the initial layer as 16 the later layers are 32, 64, 128, 256 for unconditional DDPM, where the unconditional model with 41,245,217 total and trainable parameters and conditional DDPM with 49,223,329 total and trainable parameters. While testing, a reconstruction operation by DDPM took 4 minutes 50 seconds with A100 40 Gb GPU, 7 minutes 30 seconds with V100 16 Gb GPU. To generate the whole test set, TRUBA is employed to expedite reconstructions.

2.5. EVALUATION METRICS

In this study, the performance of the proposed models is evaluated on different aspects. For the reconstruction quality and success of the model with inpainting, Maximum Absolute Difference, Mean Percentage Error, Structural Similarity Index Metric (SSIM) and Peak Signal-to-Noise (PSNR) metrics are used.

Maximum Absolute Difference is used to assess the maximum difference for the FA values between real and generated parts of a 3D volume. Mean Percentage Error is used to understand at which side the model is generating. Are the values higher than the original, or are they lower than the original, and at which side is the model mostly generating? SSIM is a measure of assessing perceptual image quality[56]. SSIM tries to imitate the human visual perception while measuring similarity by taking three key features: Luminance, Contrast, Structure[57]. By taking these three features in hand, it is a reasonable metric to rely on on the problem of reconstruction. SSIM value is in the range of [0,1] where 0 denotes no similar points in the reconstruction and 1 is very similar or the same.

PSNR is the measure of the ratio between the maximum possible signal power and the power of the disturbing noise which corrupts the quality of the reconstructed image. PSNR metric between two images are measured in dB[58].

For anomaly detection accuracy, Intersection over Union (IoU) and F1 score are calculated. F1 score is the harmonic mean between precision and recall which is used to evaluate the predictive ability of a model as shown in Equation (2.8). A higher F1 score indicates a better ability to specify anomalies[59].

$$\mathbf{P} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}} \tag{2.6}$$

$$\mathbf{R} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}} \tag{2.7}$$

$$\mathbf{F1}_{score} = \frac{2 \times \mathbf{P} \times \mathbf{R}}{\mathbf{P} \times \mathbf{R}} \tag{2.8}$$

IoU or Jaccard Index is a similarity measurement between two data from a set as given X and Y. It is defined as the intersection of X and Y, over union of X and Y as seen in Equation (2.9) [60].

$$\mathbf{IoU}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{2.9}$$

3. RESULTS

The results are acquired from both conditional and unconditional DDPM models are evaluated with SSIM, PSNR, Maximum Absolute Difference, and Average Percentage Error. For IoU and F1 score and qualitative analysis only conditional DDPM is utilized.

3.1. QUANTITATIVE RESULTS

For reconstruction performance analysis, FA maps of randomly selected subjects from whole training set is used. The masks applied for inpainting is either a complete cubic mask or a white matter guided cubic mask which denotes the white matter pathways to guide the generation with various sizes of 4, 8, 16, 32. All the metrics are calculated for only the generated parts of the 3D volume. For the unconditional DDPM the results are shown through Figure 3.1 to Figure 3.8;

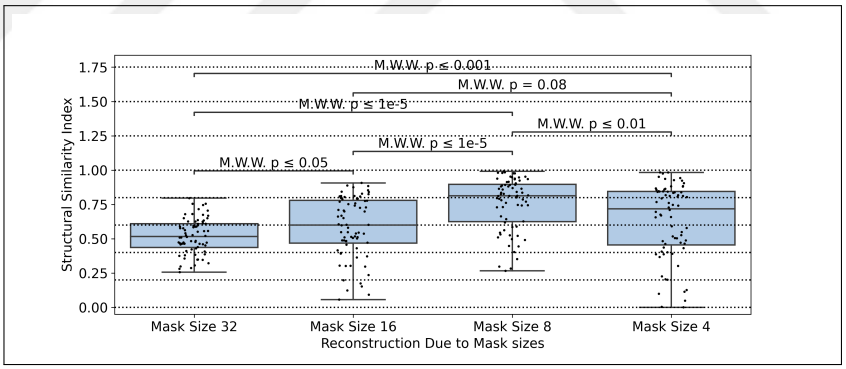

Figure 3.1. Cubic mask with sizes 32, 16, 8, 4. Structural Similarity Index values for given mask sizes.

The conditional DDPM reconstruction performance on a single inpainting mask is evaluated with the same setup as unconditional DDPM, and the results are shown through Figure 3.9 to Figure 3.16;

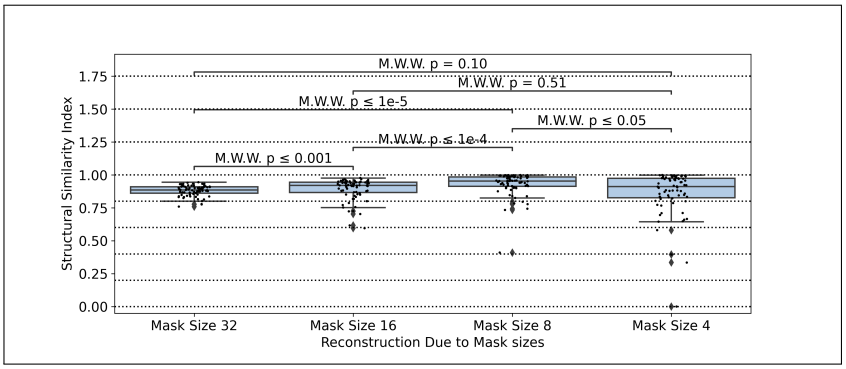

Figure 3.2. White matter guided cubic mask with with sizes 32, 16, 8, 4. Structural Similarity Index values for given mask sizes.

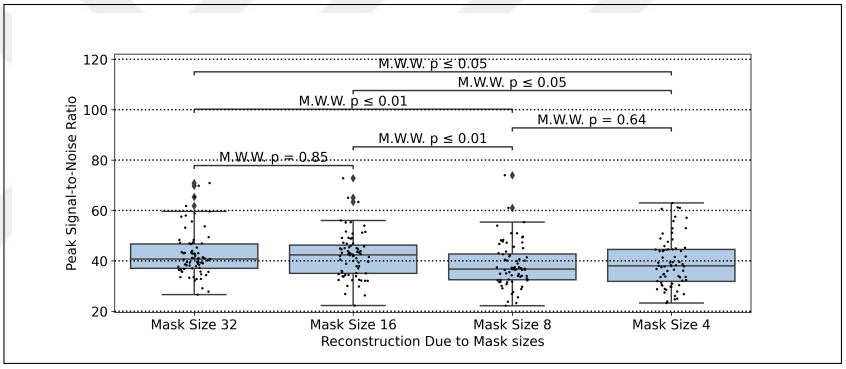

Figure 3.3. Cubic mask with sizes 32, 16, 8, 4. Peak Signal-to-Noise Ratio values for given mask sizes.

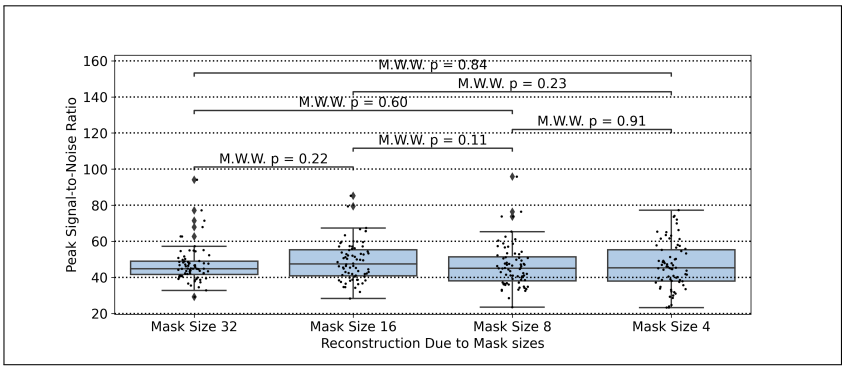

Figure 3.4. White matter guided cubic mask with with sizes 32, 16, 8, 4. Peak Signal-to-Noise Ratio values for given mask sizes.

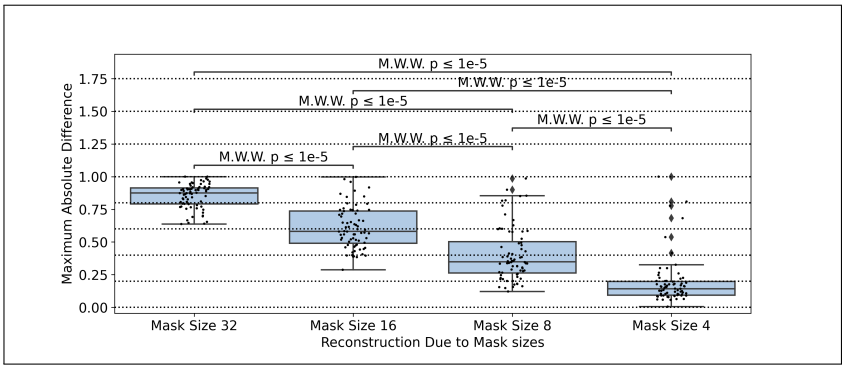

Figure 3.5. Cubic mask with sizes 32, 16, 8, 4. Maximum Absolute Difference values for given mask sizes.

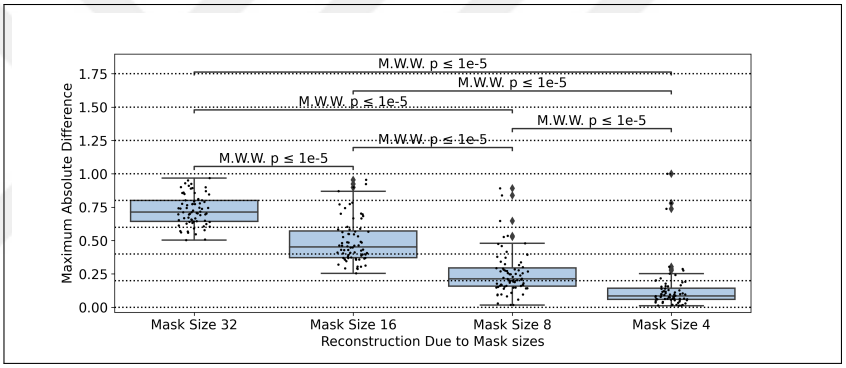

Figure 3.6. White matter guided cubic mask with with sizes 32, 16, 8, 4. Maximum Absolute Difference values for given mask sizes.

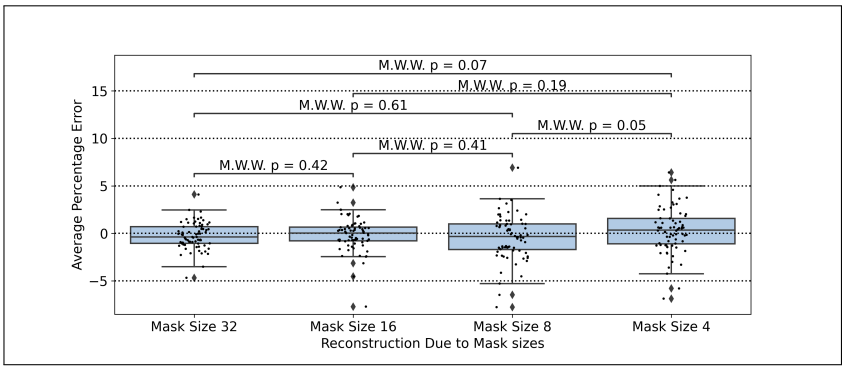

Figure 3.7. Cubic mask with sizes 32, 16, 8, 4. Average Percentage Error values for given mask sizes.

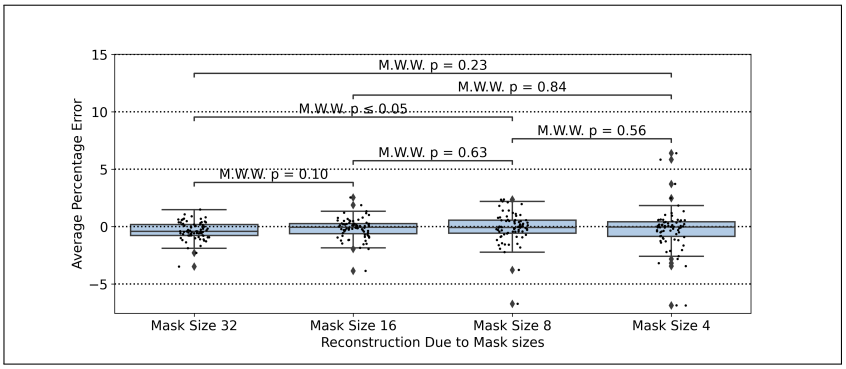

Figure 3.8. White matter guided cubic mask with with sizes 32, 16, 8, 4. Average Percentage Error values for given mask sizes.

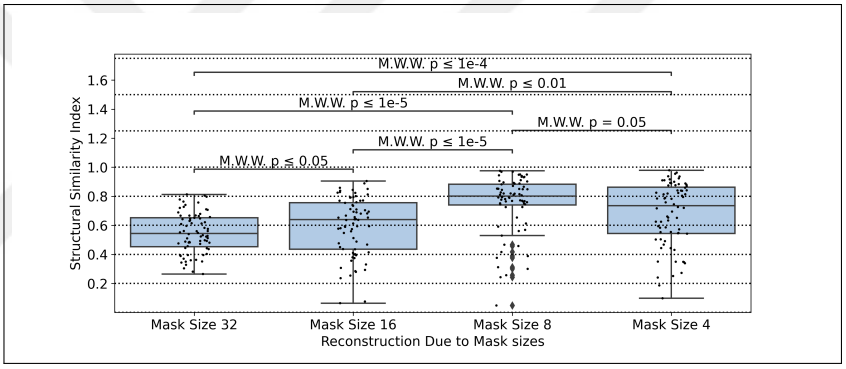

Figure 3.9. Cubic mask with sizes 32, 16, 8, 4. Structural Similarity Index values for given mask sizes.

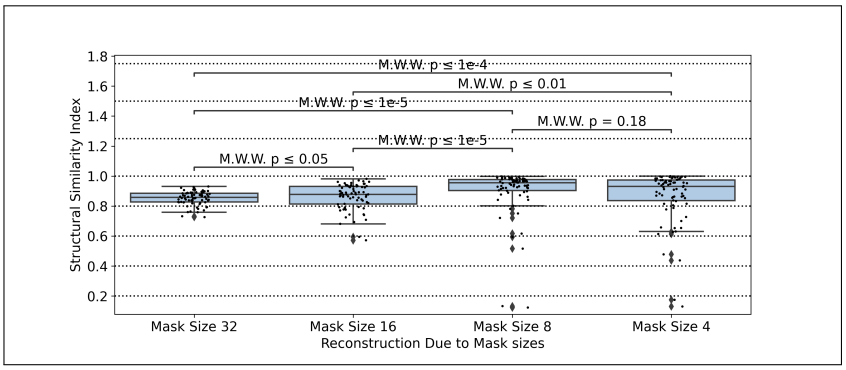

Figure 3.10. White matter guided cubic mask with with sizes 32, 16, 8, 4. Structural Similarity Index values for given mask sizes.

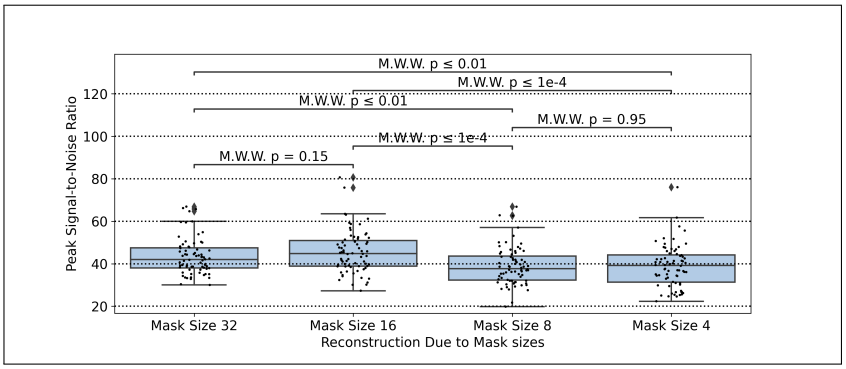

Figure 3.11. Cubic mask with sizes 32, 16, 8, 4. Peak Signal-to-Noise Ratio values for given mask sizes.

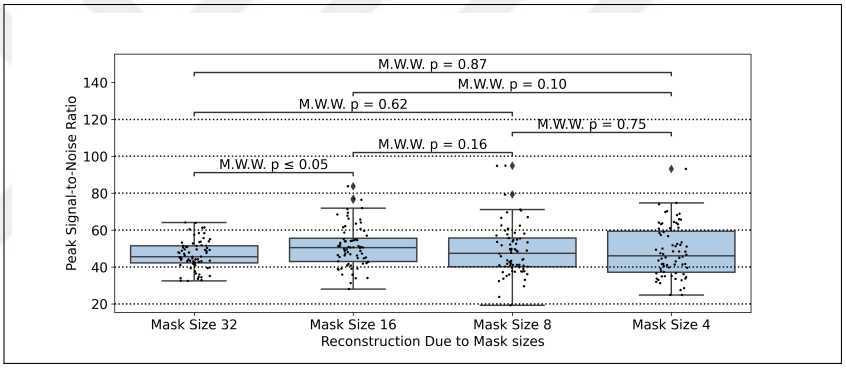

Figure 3.12. White matter guided cubic mask with with sizes 32, 16, 8, 4. Peak Signal-to-Noise Ratio values for given mask sizes.

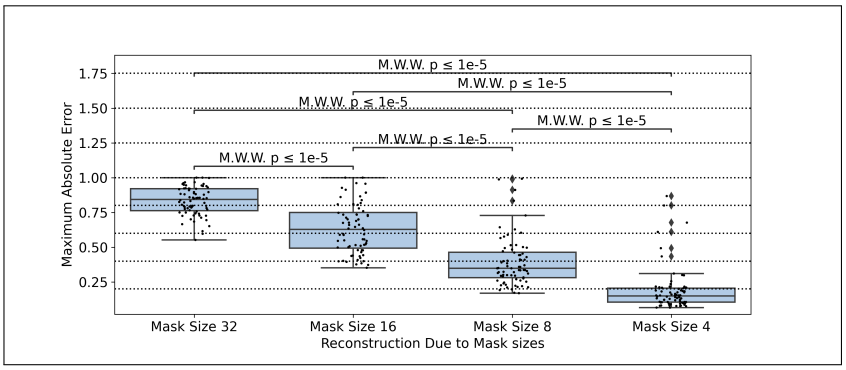

Figure 3.13. Cubic mask with sizes 32, 16, 8, 4. Maximum Absolute Difference values for given mask sizes.

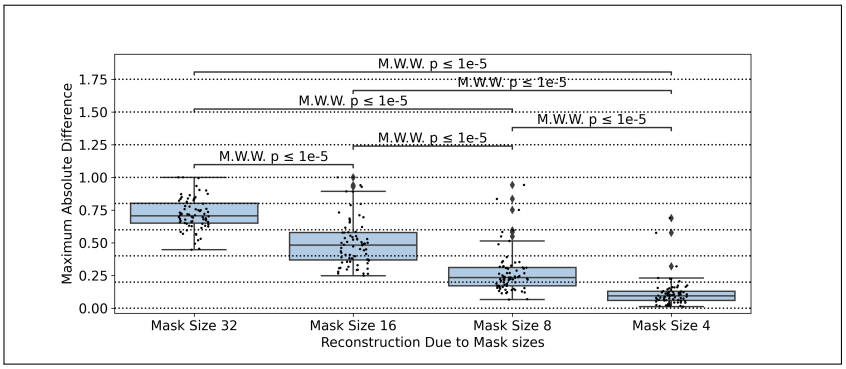

Figure 3.14. White matter guided cubic mask with with sizes 32, 16, 8, 4. Maximum Absolute Difference values for given mask sizes.

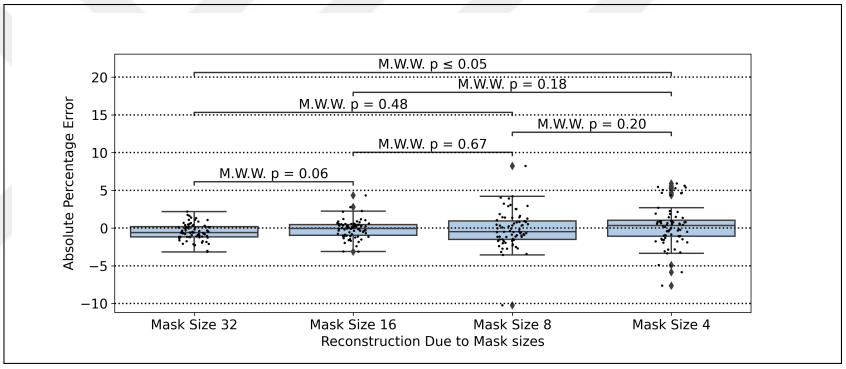

Figure 3.15. Cubic mask with sizes 32, 16, 8, 4. Average Percentage Error values for given mask sizes.

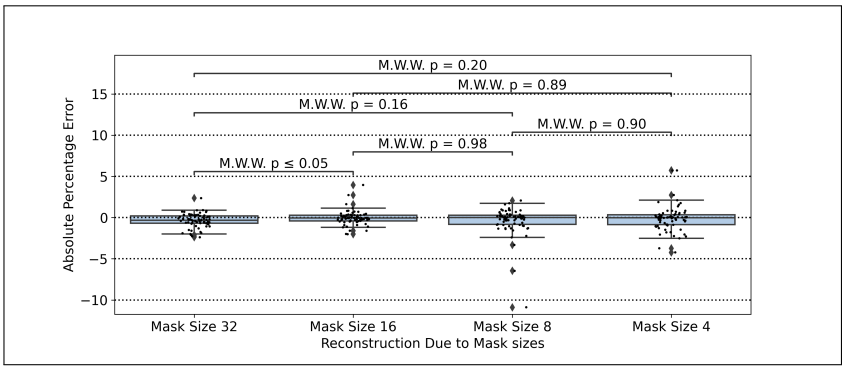

Figure 3.16. White matter guided cubic mask with with sizes 32, 16, 8, 4. Average Percentage Error values for given mask sizes.

A spherical synthetic anomaly on white matter tracts is added on the 10 subjects from the test set by increasing or decreasing FA values by 0.2 and 0.1. The spherical anomalies are added with varying sizes of 16 and 8. For detection accuracy, F1 score and IoU scores are calculated. The resulting tables can be shown through Table 3.1 to Table 3.8;

Table 3.1. Cubic mask generation from generated synthetic anomalies with increased FA values

Anomaly Size	**Increased FA value**	**IoU**	**F1 Score**
8	0.2	0.61	0.76
8	0.1	0.48	0.65
8	0.01	0.20	0.34
16	0.2	0.62	0.77
16	0.1	0.48	0.65
16	0.01	0.30	0.46

Table 3.2. Cubic mask generation from generated synthetic anomalies with decreased FA values

Anomaly Size	**Decreased FA value**	**IoU**	**F1 Score**
8	0.2	0.53	0.70
8	0.1	0.32	0.49
8	0.01	0.37	0.23
16	0.2	0.60	0.75
16	0.1	0.35	0.52
16	0.01	0.26	0.41

Table 3.3. White matter guided cubic masks generation with increased intensities

Anomaly Size	Increased FA value	IoU	F1 Score
8	0.2	0.75	0.86
8	0.1	0.64	0.78
8	0.01	0.21	0.35
16	0.2	0.79	0.88
16	0.1	0.60	0.75
16	0.01	0.31	0.48

Table 3.4. White matter guided cubic masks generation with decreased intensities

Anomaly Size	Decreased FA value	IoU	F1 Score
8	0.2	0.70	0.82
8	0.1	0.37	0.53
8	0.01	0.23	0.37
16	0.2	0.70	0.82
16	0.1	0.37	0.54
16	0.01	0.25	0.41

Table 3.5. Cubic mask generation from generated synthetic anomalies with various increased intensities, mask and anomaly sizes

Mask Size	Anomaly Size	Increased FA value	IoU	F1 Score
4	8	0.2	0.49	0.66
4	8	0.1	0.34	0.51
4	8	0.01	0.12	0.21
8	16	0.2	0.47	0.64
8	16	0.1	0.29	0.45
8	16	0.01	0.20	0.34

Table 3.6. Cubic mask generation from generated synthetic anomalies with various decreased intensities, mask and anomaly sizes

Mask Size	Anomaly Size	Decreased FA value	IoU	F1 Score
4	8	0.2	0.46	0.63
4	8	0.1	0.19	0.32
4	8	0.01	0.11	0.21
8	16	0.2	0.42	0.59
8	16	0.1	0.24	0.39
8	16	0.01	0.18	0.31

Table 3.7. White matter guided cubic masks generation with various increased intensities, mask and anomaly sizes

Mask Size	Anomaly Size	Increased FA value	IoU	F1 Score
4	8	0.2	0.59	0.74
4	8	0.1	0.44	0.61
4	8	0.01	0.098	0.18
8	16	0.2	0.57	0.73
8	16	0.1	0.35	0.52
8	16	0.01	0.21	0.35

Table 3.8. White matter guided cubic masks generation with various decreased intensities, mask and anomaly sizes

Mask Size	Anomaly Size	Decreased FA value	IoU	F1 Score
4	8	0.2	0.44	0.61
4	8	0.1	0.19	0.32
4	8	0.01	0.11	0.19
8	16	0.2	0.50	0.67
8	16	0.1	0.22	0.35
8	16	0.01	0.18	0.31

In Table 3.9 a full roaming white matter guided cubic inpainting mask with a diameter of 16 is utilized to generate a whole 3D volume with conditional DDPM. A synthetic anomaly is placed by increasing FA values by 0.3, 0.2, 0.1 with a diameter of 32 and 16. The reconstruction operation is repeated approximately 170 times which roughly took 14 hours. The quantiative results also can be seen in Figure 3.42.

Table 3.9. Full reconstruction IoU and F1 score evaluation with different anomaly sizes and different FA anomalies.

Anomaly Size	**Increased FA value**	**IoU**	**F1 Score**
32	0.2	0.05	0.095
32	0.1	0.037	0.071
16	0.2	0.008	0.015
16	0.1	0.005	0.01

The resulting ROC curve for Table 3.9 generations are given in Figure 3.17;

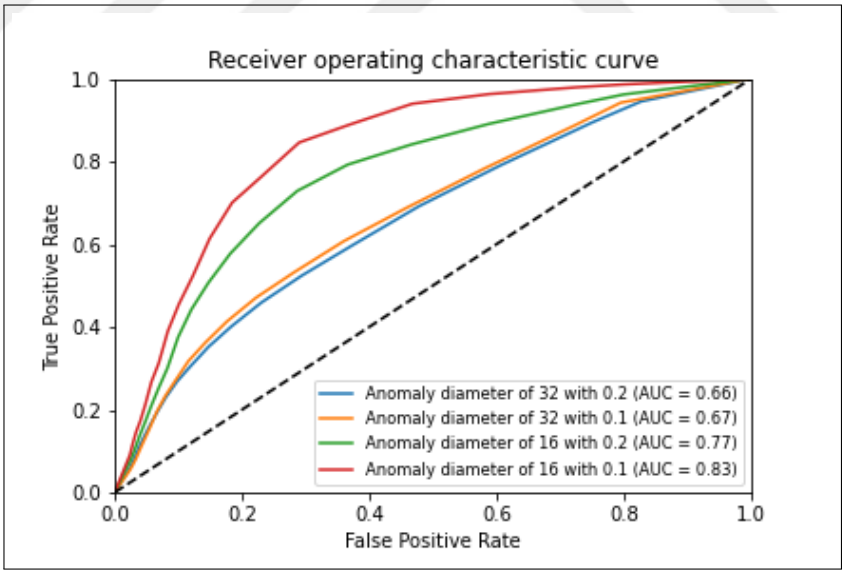

Figure 3.17. Generation from added spherical synthetic anomalies with diameters of 32 and 16 by increasing FA values of 0.2, 0.1 .

The proposed methodology compared with p-DDPM methodology with simplex noise and gaussian noise by following the article [37]. The results for BraTS21 dataset can be seen from Table 3.10 and results for MSLUB dataset can be seen from Table 3.11.

Table 3.10. Evaluation of comparison between pDDPM and Inpainting only in inference on BraTS21 dataset

Methodology	DICE[%]	AUPRC[%]
pDPPM + simplex noise	50.00	55.12
Inpainting with simplex noise	39.01	45.25
Inpainting with gaussian noise	37.33	40.44

Table 3.11. Evaluation of comparison between pDDPM and Inpainting only in inference on MSLUB dataset

Methodology	DICE[%]	AUPRC[%]
pDPPM + simplex noise	9.95	7.00
Inpainting with simplex noise	2.52	4.09
Inpainting with gaussian noise	4.04	5.19

3.2. QUALITATIVE RESULTS

A spherical synthetic anomaly on white matter tracts is added on the test set by increasing or decreasing FA values by 0.2 and 0.1. The spherical anomalies are added with varying size as 16 and 8. The results are generated with conditional DDPM since it showed better performance than unconditional DDPM

In Figures through Figure 3.18 to Figure 3.33, in the synthetic anomaly column, spherical synthetic anomalies are added on the white matter tracts, with a diameter of 16 or 8, increasing or decreasing FA values by 0.2 or 0.1. Corresponding inpainting reconstruction with cubic mask or white matter guided cubic masks with size of 16 or 8 on anomalies location, is shown in reconstructed column. The synthetic anomaly mask and the difference between first and second column are shown on the anomaly mask and reconstructed mask column, respectively.

In Figures through Figure 3.34 to Figure 3.41 in the synthetic anomaly column, spherical synthetic anomalies are added on the white matter tracts, with a diameter of 16 or 8, increasing FA values by 0.2 or 01. Corresponding inpainting reconstruction with white matter guided cubic mask size of 8 or 4 on anomalies location by roaming smaller cubic masks, is shown in the reconstructed column. The synthetic anomaly mask and the difference between first and

second column are shown on the anomaly mask and reconstructed mask column, respectively.

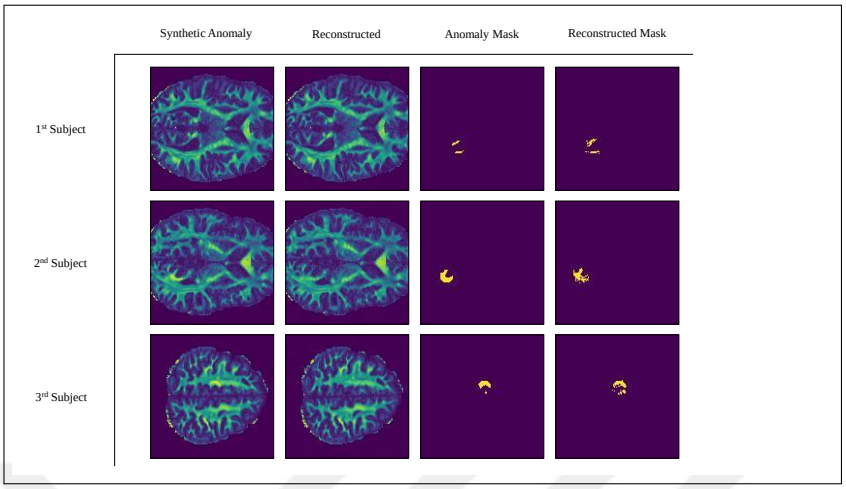

Figure 3.18. Generation with cubic mask of size 16 with increased FA values of 0.2.

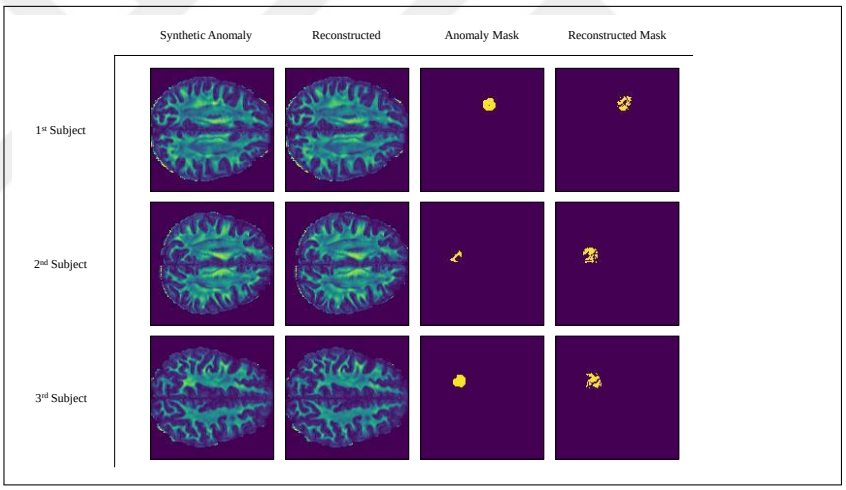

Figure 3.19. Generation with cubic mask of size 16 with increased FA values of 0.1.

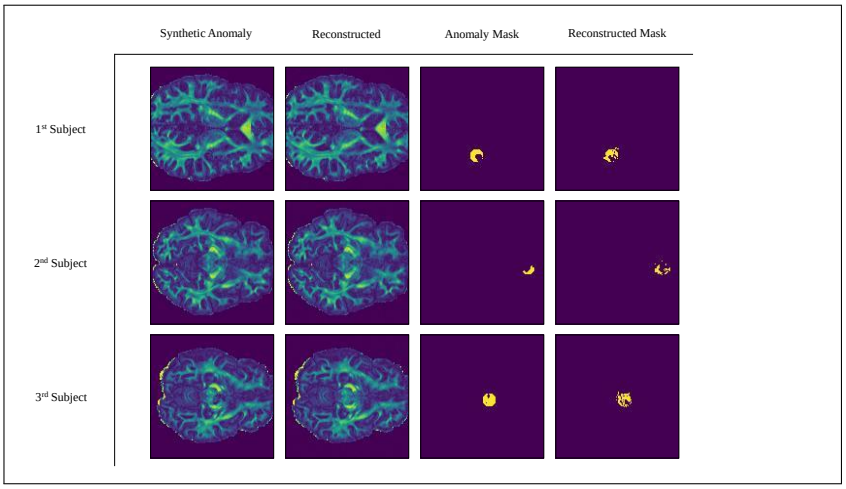

Figure 3.20. Generation with cubic mask of size 16 with decreased FA values of 0.2.

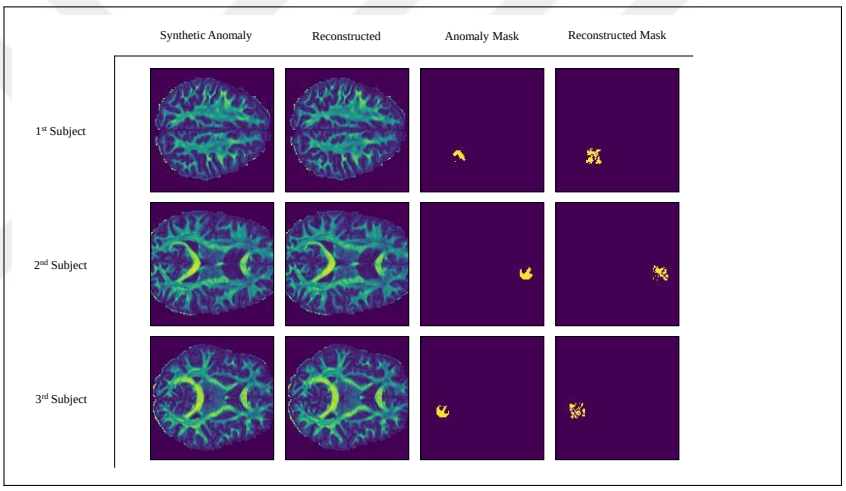

Figure 3.21. Generation with cubic mask of size 16 with decreased FA values of 0.1.

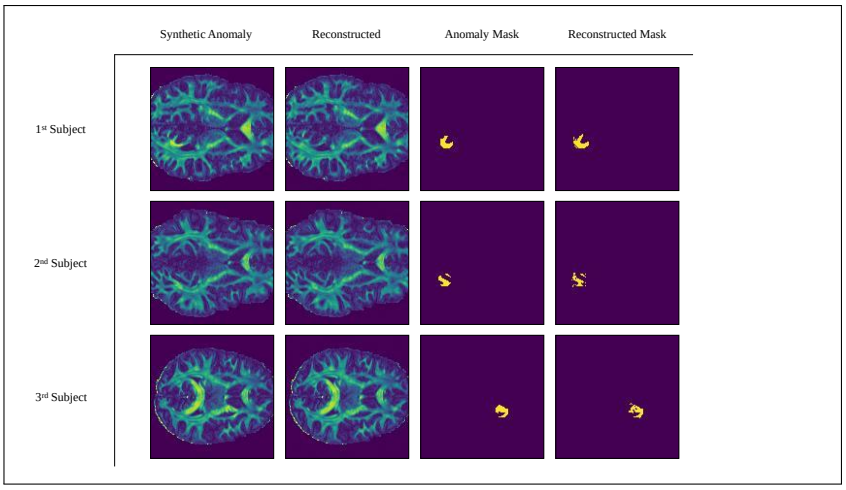

Figure 3.22. Generation with white matter guided cubic mask of size 16 with increased FA values of 0.2.

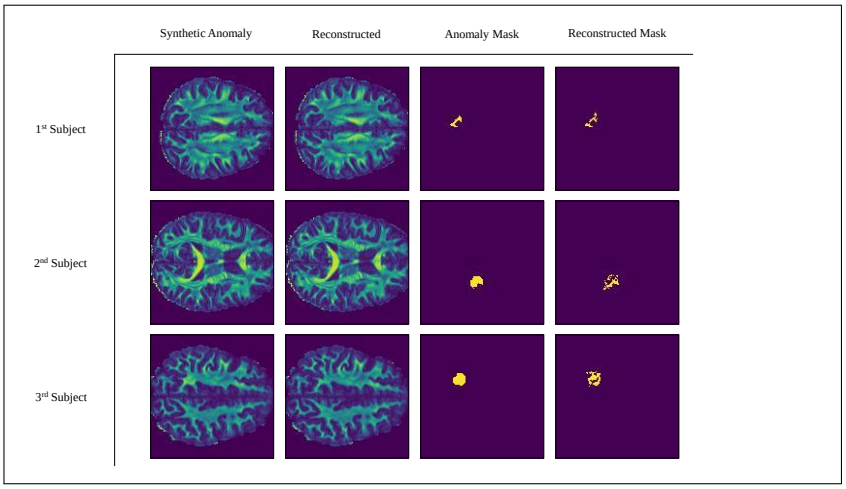

Figure 3.23. Generation with white matter guided cubic mask of size 16 with increased FA values of 0.2.

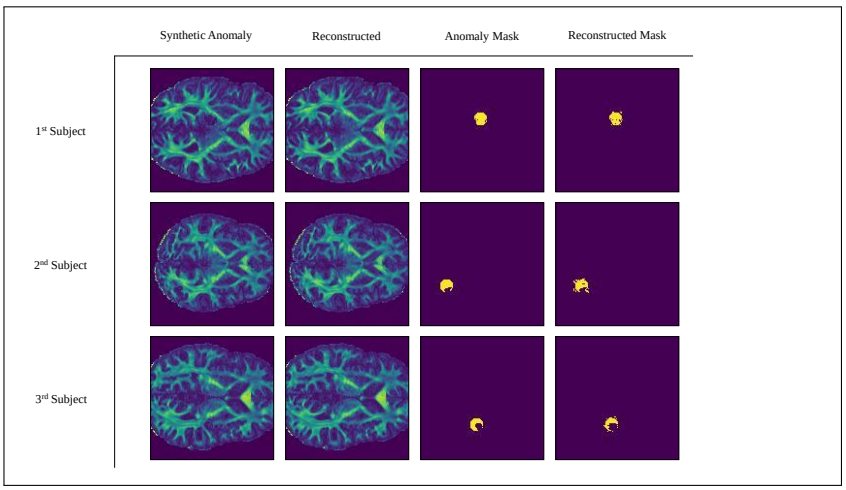

Figure 3.24. Generation with white matter guided cubic mask of size 16 with decreased FA values of 0.2.

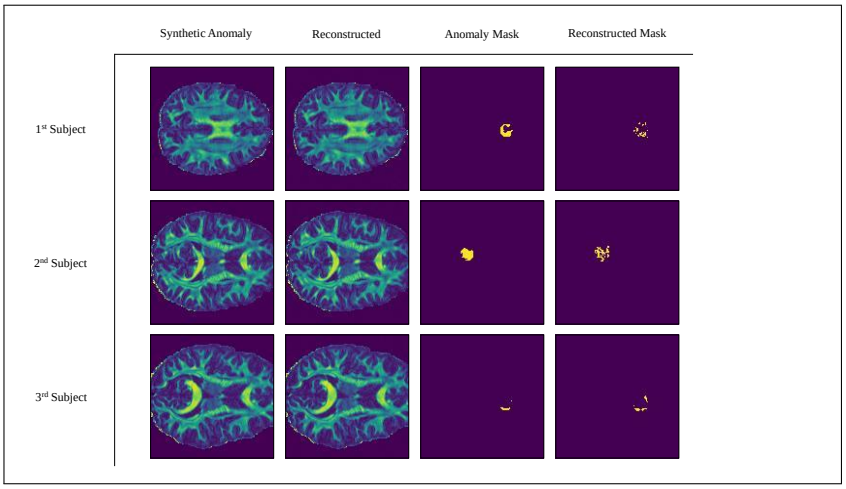

Figure 3.25. Generation with white matter guided cubic mask of size 16 with decreased FA values of 0.1.

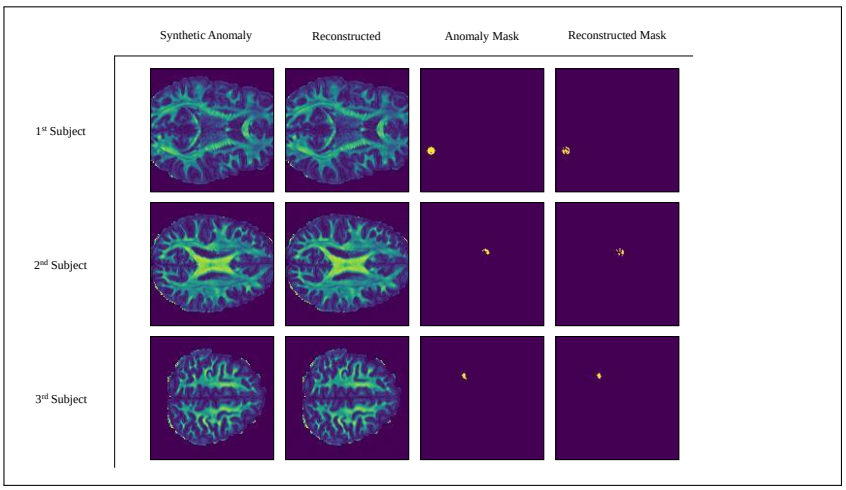

Figure 3.26. Generation with white matter cubic mask of size 8 with increased FA values of 0.2.

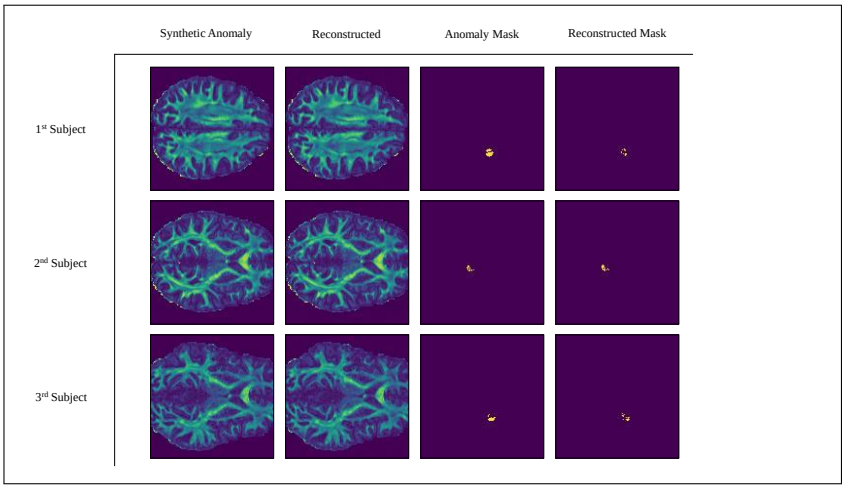

Figure 3.27. Generation with white matter cubic mask of size 8 with increased FA values of 0.1.

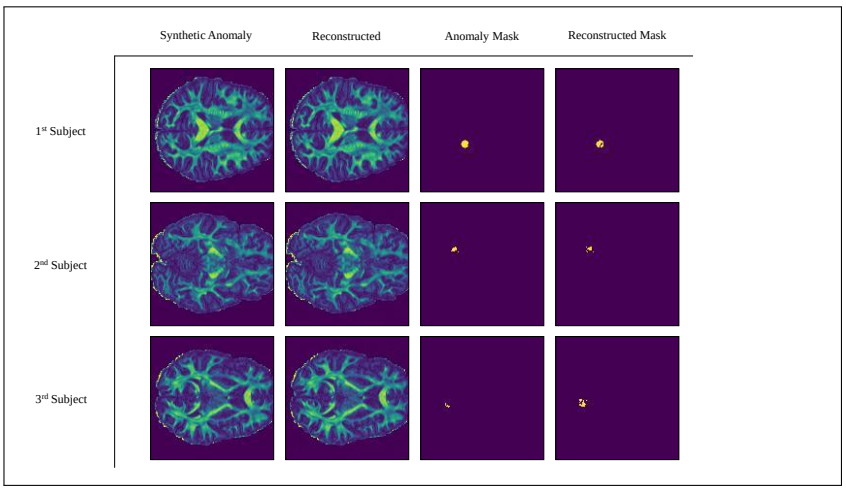

Figure 3.28. Generation with white matter guided cubic mask of size 8 with decreased FA values of 0.2.

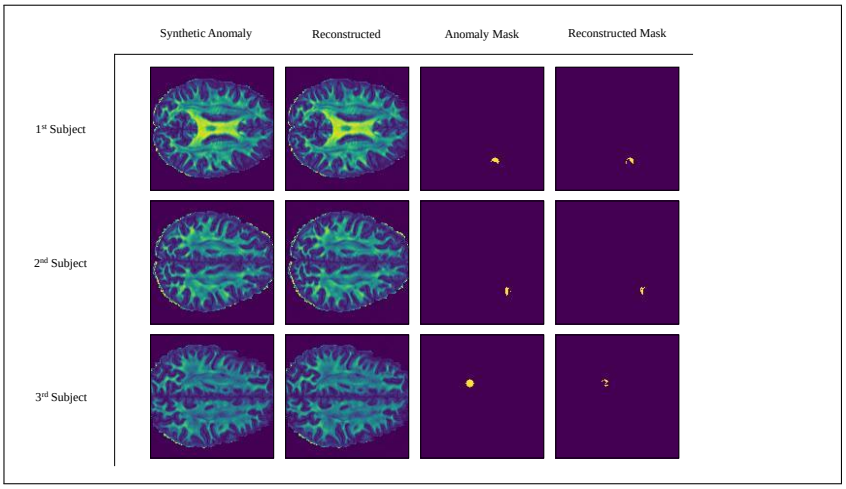

Figure 3.29. Generation with white matter guided cubic mask of size 8 with decreased FA values of 0.1.

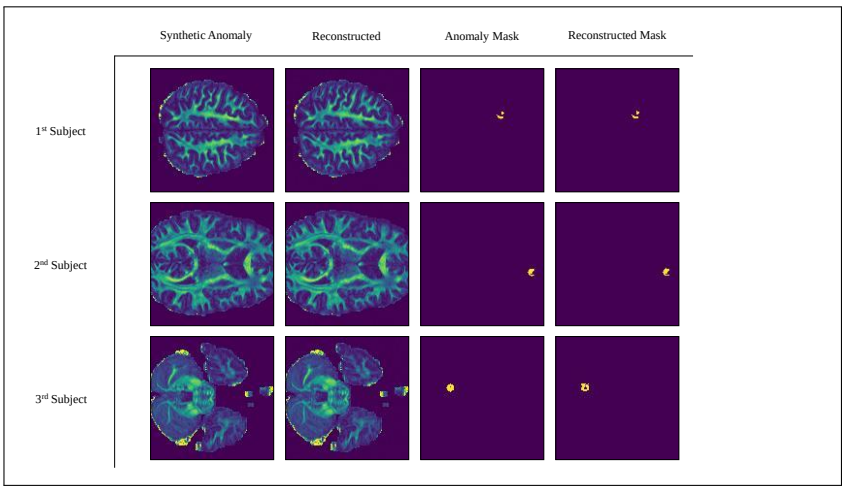

Figure 3.30. Generation with white matter cubic mask of size 8 with increased FA values of 0.2.

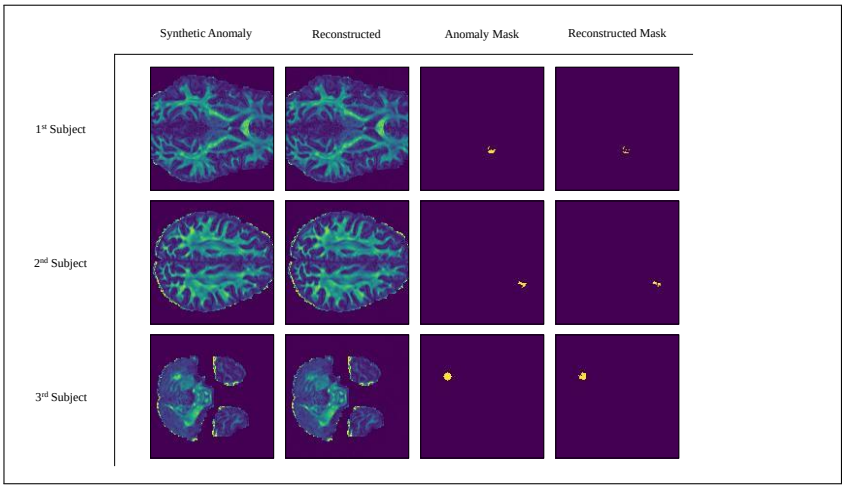

Figure 3.31. Generation with white matter cubic mask of size 8 with increased FA values of 0.1.

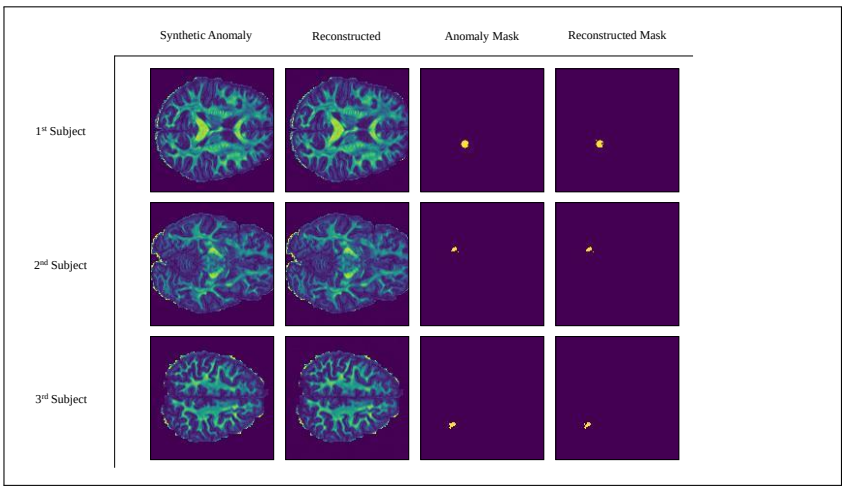

Figure 3.32. Generation with white matter guided cubic mask of size 8 with decreased FA values of 0.2.

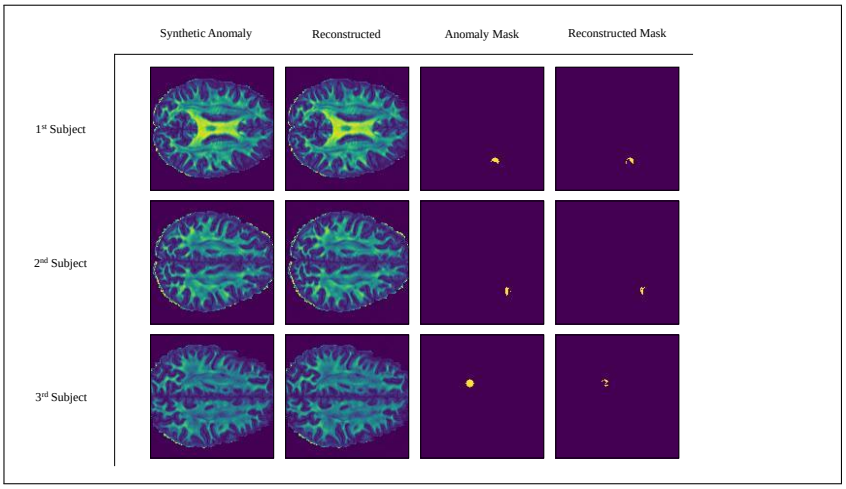

Figure 3.33. Generation with white matter guided cubic mask of size 8 with decreased FA values of 0.1.

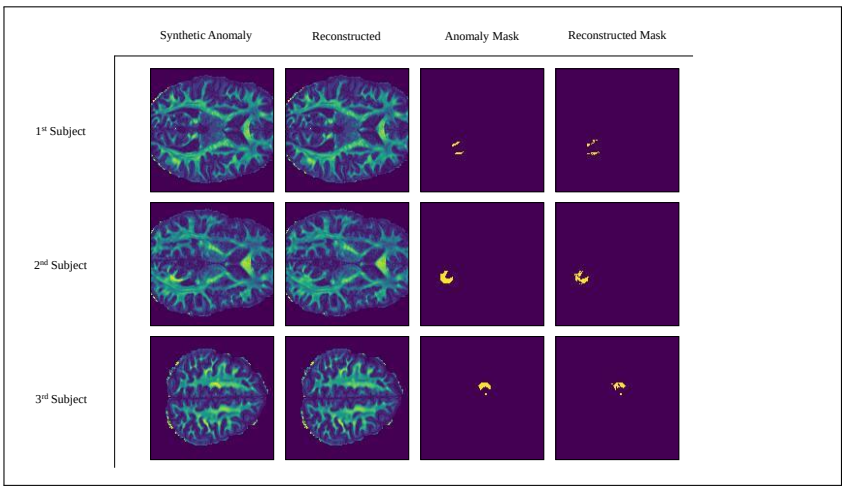

Figure 3.34. Generation with smaller white matter guided cubic masks of size 8 on anomaly mask with size 16 for increased FA values of 0.2.

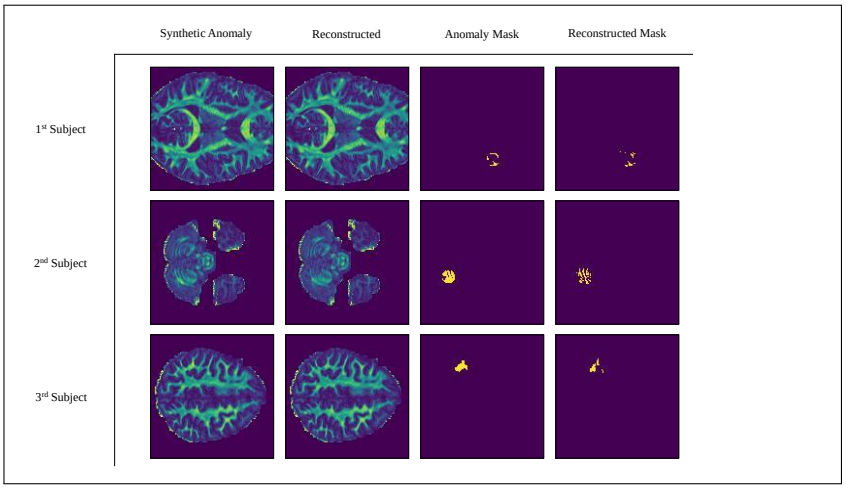

Figure 3.35. Generation with smaller white matter guided cubic masks of size 8 on anomaly mask with size 16 for increased FA values of 0.1.

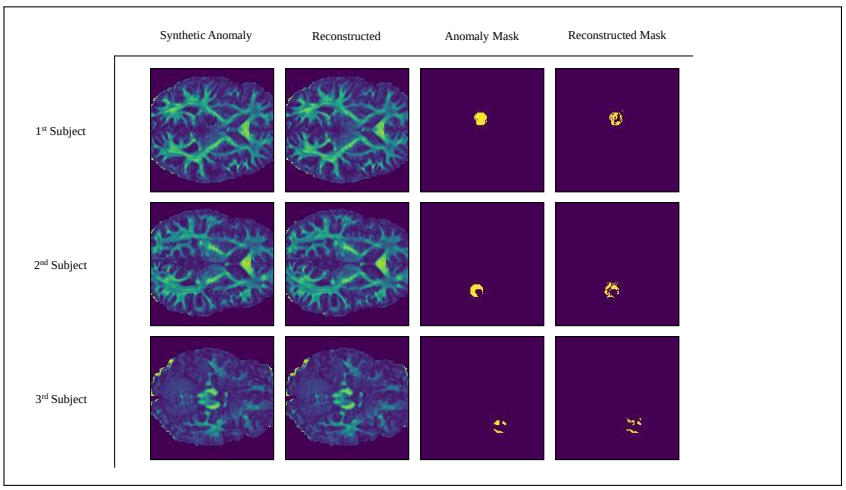

Figure 3.36. Generation with smaller white matter guided cubic masks of size 8 on anomaly mask with size 16 for decreased FA values of 0.2.

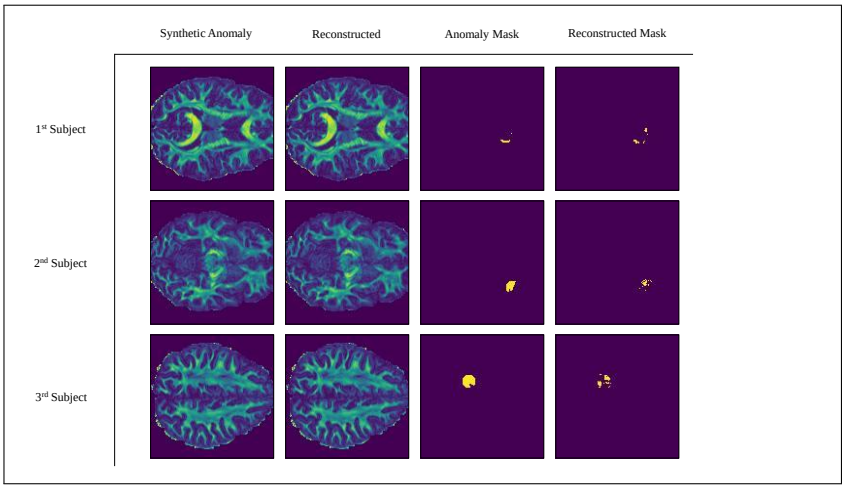

Figure 3.37. Generation with smaller white matter guided cubic masks of size 8 on anomaly mask with size 16 for increased FA values of 0.1.

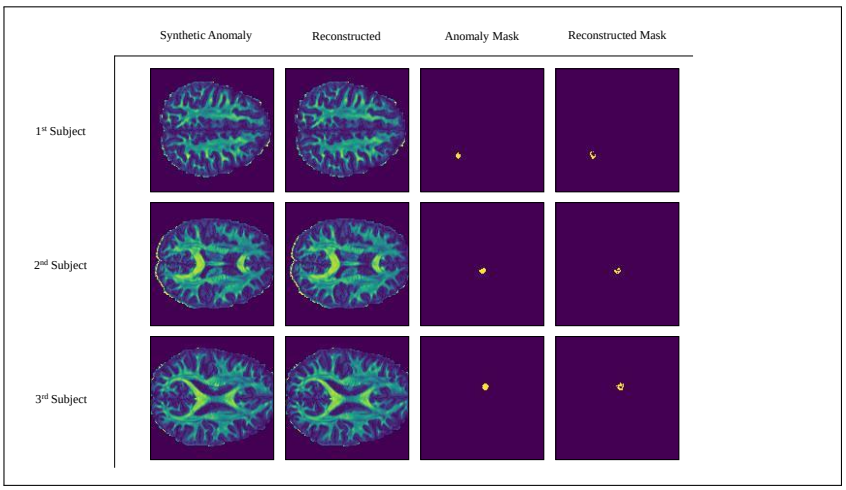

Figure 3.38. Generation with smaller white matter guided cubic masks of size 4 on anomaly mask with size 8 for increased FA values of 0.2.

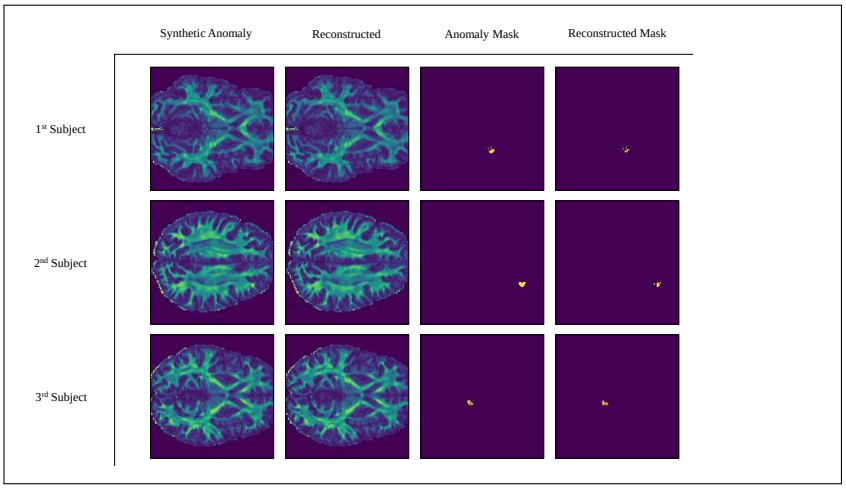

Figure 3.39. Generation with smaller white matter guided cubic masks of size 4 on anomaly mask with size 8 for increased FA values of 0.1.

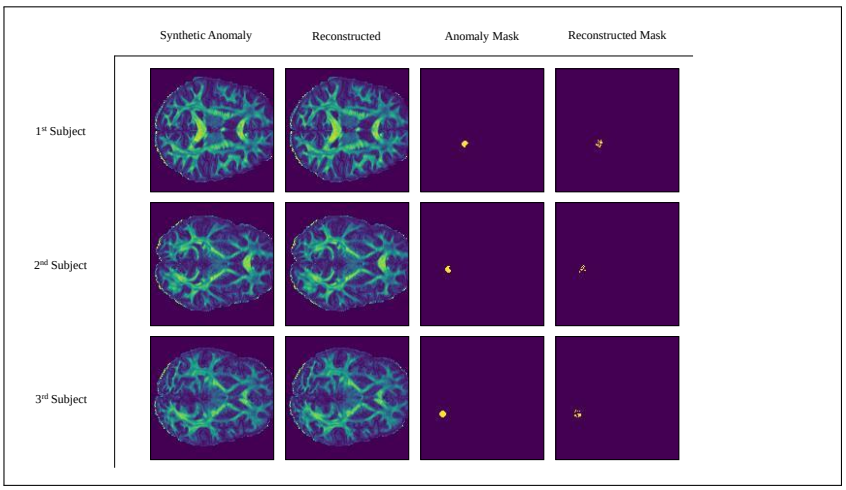

Figure 3.40. Generation with smaller white matter guided cubic masks of size 4 on anomaly mask with size 8 for decreased FA values of 0.2.

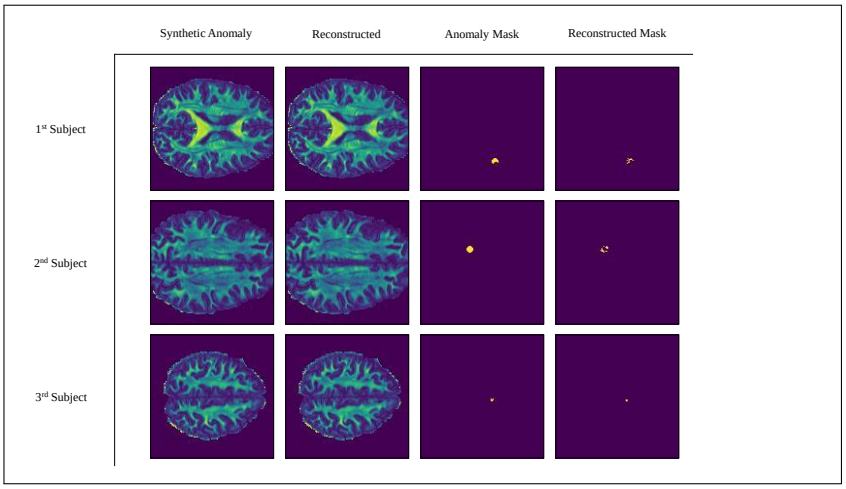

Figure 3.41. Generation with smaller white matter guided cubic masks of size 4 on anomaly mask with size 8 for decreased FA values of 0.1.

In Figure 3.42 and Figure 3.43 a synthetic anomaly is placed by increasing FA values by 0.2 and 0.1 with a diameter of 32 and 16, respectively can be seen on anomaly mask column. The resulting difference of the synthetic anomaly volume and reconstructed volume can be seen on reconstructed mask column. The reconstruction operation is repeated 171 times which roughly took 12 hours.

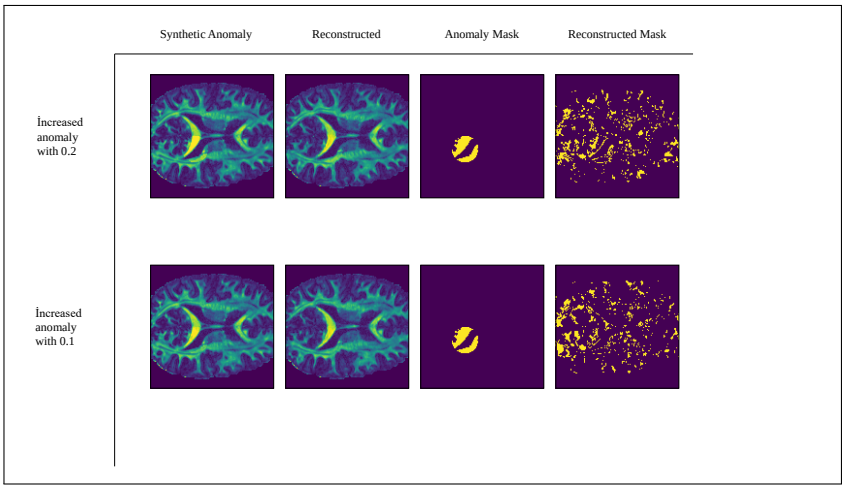

Figure 3.42. Generation with white matter guided cubic masks of size 16 on anomaly mask with size 32.

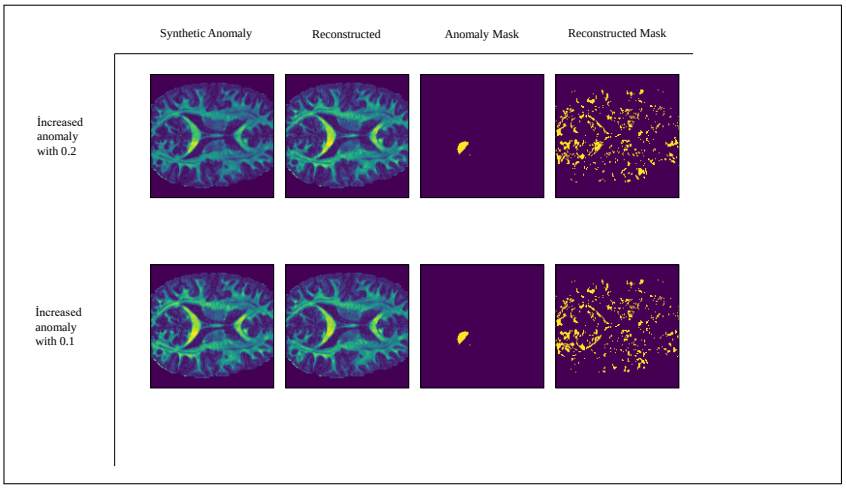

Figure 3.43. Generation with white matter guided cubic masks of size 16 on anomaly mask with size 16.

4. DISCUSSION

In this study; a pre-trained conditional DDPM is proposed to detect anomalies in 3D volumes of FA maps. FA maps are preferred to detect pixel-wise anomalies for the diffusion of water molecules in brain. This anomalies corresponds to conditions such as; Multiple Sclerosis (MS), Alzheimer's Disease (AD), Parkinson's Disease (PD), and traumatic brain injuries that affects white matter tracts in real world applications. To detect anomalies, a reconstruction of a healthy approximation for the patient is generated by 3D conditional DDPM, which is conditioned on 3D volumes of white matter tracts. The main challenge while generating healthy approximations is keeping the anatomy of the healthy region. To keep the healthy anatomies as they were, the inpainting method is employed to narrowing down the generated location for each step.

There have been notable studies reported to detect anomalies in medical imaging, and also in structural brain MRIs. In [32], it has been proposed to generate pseudo-healthy reconstruction from DDPM with a pre-trained classifier to guide the generation during diffusion steps. Although, the anomaly detection stands similar as the main aim, the techniques and the scope of the aim is quite different. With FA maps as data, the neurological diseases aimed to detect. Even though the learning strategy in the aforementioned study is weakly-supervised, it needs image-level annotated classes.

To overcome the problem of the scarcity of data with annotations, with only a healthy dataset, by learning the modeling of the healthy anatomy, an unsupervised approach is a popular training strategy amongst anomaly detection studies[30]. Conditioning the model, by leveraging cross-attention, on the white matter tracts, it is aimed to control the generation of the DDPM in an unsupervised manner. The U-Net architecture is used to apply the reverse diffusion process of the DDPM. The data is provided as 3D volumes of FA maps to generate a whole volume of reconstructed healthy 3D volume, U-Net architecture and cross-attention mechanism are modified for this problem to process 3D data.

DDPM model is pre-trained with only healthy data without any annotations from HCP to generate a model of healthy anatomy. The generation is controlled by a cross-attention module as a conditioning mechanism and conditioned with 3D white matter tract masks of

input 3D FA volume. As can be seen from Figure 3.36, the model indeed learns generating from the modeled distribution of healthy anatomy. As can be seen from Figure 3.42 and, Figure 3.43 while generating for whole 3D volume, the model fails to generate the healthy anatomy of the patient as it were healthy.

The partial generation of the conditional and unconditional model's performance is evaluated through different situations and evaluated to compare with the pDDPM model. While using the unconditional model, providing fractional anisotropy guided cubic mask decreases Maximum Absolute Difference, and Average Percentage Error and increases SSIM. Providing fractional anisotropy guided cubic mask in the inference step increases the performance of the proposed model. As it can be seen from Figure 3.5 and Figure 3.6, the Maximum Absolute Difference decreases with the mask size and white matter guided cubic masks rather than just cubic masks. Providing extra conditions also allows the model to preserve the anatomy of the white matter tracts and leaves only generating FA values instead of a complete cubic space. Average Percentage Error shows that most of the time, the generated volumes from model has lower FA values than originals. The highest SSIM value is obtained with mask size 8 and then mask sizes 16 and 4. Thus while evaluating anomaly detection performance, mask size 16 is used. Since mask sizes 4 ,and 8 increases generating time for a whole 3D volume dramatically, they are not preferred.

Starting from Table 3.1 to Table 3.8, it can be seen that even though the model is conditioned with white matter pathways, also providing partial white matter tracts instead of a cubic mask shows better anomaly detection performance in terms of F1 score and IoU. It can be seen that decreased FA values cause more problems while detecting anomalies than increasing FA values. It can be seen that as the FA values increase or decrease less, F1 scores and IoU values decrease, which shows that the conditional DDPM model fails to show the correct shape of the anomaly also can be seen through Figure 3.18 to Figure 3.41. Also, generating a part where the anomalous synthetic part is reconstructed with multiple masks decreases the model performance. The most crucial part while detecting an anomaly with inpainting from a given 3D brain FA volume is the ability to generate the anatomically correct generation of non-anomalous parts and also generating correct FA values for non-anomalous parts and also generating an estimation of FA values for anomalous parts if the subject were healthy. The proposed model shows good performance for partial reconstruction, but for a complete

reconstruction the model shows poor performance, as it can be seen from Figure 3.42 and Table 3.9. With the comparison held with pDDPM, it has been shown that when the inpainting methodology is included while training, the model performs better in terms of DICE and AUPRC results. The minimum detectable FA value for the model is detected as setting the threshold value 0.02 lower than the anomalies FA value. The experiments indicates that, the models output creates meaningfull results in terms of IoU, DICE and accuracy when threshold is set 0.02 lower than the anomalies FA value. Also it has been seen that skull-stripping operation is important while setting the threshold, because the variety of the skull leftovers does not affect the dataset domain, and in inference skull leftovers can distort the resulting anomaly maps.

One of the secondary challenge after preserving healthy anatomy is, models inference time, as it takes approximately 4 minutes and 50 seconds to generate a partial volume from the whole 3D volume in the case of a 128x128x128 input for a mask with volume of 8x8x8, the model had to generate 1540 partial parts which correspond to brain parts to generate whole input which approximately takes 115.5 hours to generate a whole volume and with a mask volume of 16x16x16 model had to generate 171 partial parts which approximately take 12 hours to generate a whole volume with using A100 GPU.

5. CONCLUSIONS

Lately, unsupervised anomaly detection in medical imaging has been rapidly developing. Aiming to overcome the scarcity of training data by leveraging unsupervised training strategies, the unsupervised deep anomaly methods are applied to different problems in the medical domain by introducing different model architectures. Nowadays, DDPMs showed better results in unsupervised anomaly detection using a reconstruction approach instead of GANs and VAEs. However, the challenging part while generating healthy approximations in the pixel level is keeping the anatomy of the healthy region. In the case of this study, unsupervised deep anomaly detection enhanced image inpainting is utilized with DDPMs to narrow down the generated location for each step. In order to detect anomalies using roaming inpainting masks on the input and generating each region, a healthy approximation of the patient is generated.

An unconditional and conditional DDPM is utilized and compared to generate the inpainting masks. The results show that conditional DDPM generates better results at a single reconstruction. Also, the different masking types with different sizes, such as a complete cubic mask and white matter-guided cubic masks, are applied, and the outcomes are obtained to evaluate the generation performance of the proposed model. The results of this study show that the proposed model can be used for the partial reconstruction of known anomalous regions and shows better performance with higher deviations from healthy FA values. The model generates an approximation of the healthy for input, yet it is not anatomically correct for non-anomalous regions, which results in incorrectly predicted anomalies as an outcome in the pixel level.

5.1. FUTURE WORKS

Different conditioning mechanisms such as spatial attention, channel attention, or conditional normalization methods can be researched to better project the anatomical correctness of the non-anomalous regions to increase generation performance and anomaly detection. The effect of resampling from [12] can also be researched to correctly generate by harmonizing anatomical parts of non-anomalous regions at each step and FA values for anomalous regions.

With weak supervision, FA values can be controlled while generation through a classifier condition if the FA values for different diseases exist and are annotated in the upcoming period. Also the improvement in inference time for 3D inputs can also be researched by employing different sampling methods.

REFERENCES

1. Grubbs FE. Procedures for Detecting Outlying Observations in Samples. *Technometrics*. 1969;11(1):1-21.

2. Pang G, Shen C, Cao L, Hengel AVD. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*. 2021;54(2):1-38.

3. Boukerche A, Zheng L, Alfandi O. Outlier Detection: Methods, Models, and Classification. *ACM Comput Surv*. 2020;53(3):1-37.

4. Aburakhia S, Tayeh T, Myers R, Shami A. A Transfer Learning Framework for Anomaly Detection Using Model of Normality. *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 2020:0055-61.

5. Fernando T, Gammulle H, Denman S, Sridharan S, Fookes C. Deep Learning for Medical Anomaly Detection – A Survey. *ACM Comput Surv*. 2021;54(7):1-37.

6. Kwon D, Kim H, Kim J, Suh SC, Kim I, Kim KJ. A Survey of Deep Learning-Based Network Anomaly Detection. *Cluster Computing*. 2019;22(1):949—961.

7. Bertalmío M, Sapiro G, Caselles V, Ballester C. Image inpainting. 2000:417-24.

8. Jam J, Kendrick C, Walker K, Drouard V, Hsu J, Yap MH. A comprehensive review of past and present image inpainting methods. *Comput Vis Image Underst*. 2021;203:103-47.

9. Qin Z, Zeng Q, Zong Y, Xu F. Image inpainting based on deep learning: A review. *Displays*. 2021;69:102028.

10. Vaughanan M, D V, Naikodi S, G M, N M. A review on various methods of image inpainting. *Proceedings of the International Conference on Innovative Computing and Communication (ICICC)*. 2023:.

11. Xiang H, Zou Q, Nawaz MA, Huang X, Zhang F, Yu H. Deep learning for image inpainting: A survey. *Pattern Recognition*. 2023;134:109046.

12. Lugmayr A, Danelljan M, Romero A, Yu F, Timofte R, Van Gool L. RePaint:

Inpainting using Denoising Diffusion Probabilistic Models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022:11451-61.

13. Sohl-Dickstein J, Weiss E, Maheswaranathan N, Ganguli S. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. *Proceedings of the 32nd International Conference on Machine Learning*. PMLR. 2015:2256-65.

14. Kazerouni A, Aghdam EK, Heidari M, Azad R, Fayyaz M, Hacihaliloglu I, et al. Diffusion models in medical imaging: A comprehensive survey. *Medical Image Analysis*. 2023;88:102846.

15. Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*. 2020;33:6840-51.

16. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative Adversarial Networks. *Advances in Neural Information Processing Systems*. 2014;3.

17. Kingma DP, Welling M. Auto-Encoding Variational Bayes. *CoRR*. 2013;abs/1312.6114:arXiv:1312.6114.

18. Pathak D, Krahenbuhl P, Donahue J, Darrell T, Efros AA. Context encoders: Feature learning by inpainting. *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE. 2016:2536-44.

19. Richardson E, Alaluf Y, Patashnik O, Nitzan Y, Azar Y, Shapiro S, et al. Encoding in style: a stylegan encoder for image-to-image translation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. IEEE. 2021:2287-96.

20. Saharia C, Chan W, Chang H, Lee C, Ho J, Salimans T, et al. Palette: Image-to-image diffusion models. *ACM SIGGRAPH 2022 Conference Proceedings*. 2022:1-10.

21. Choi J, Kim S, Jeong Y, Gwon Y, Yoon S. ILVR: Conditioning Method for Denoising Diffusion Probabilistic Models. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021:14347-56.

22. Baur C, Wiestler B, Muehlau M, Zimmer C, Navab N, Albarqouni S. Modeling Healthy

Anatomy with Artificial Intelligence for Unsupervised Anomaly Detection in Brain MRI. *Radiology: Artificial Intelligence*. 2021;3(3):e190169.

23. Chen X, Konukoglu E. Unsupervised Detection of Lesions in Brain MRI using Constrained Adversarial Auto-encoders. *MIDL Conference book*. MIDL. 2018:.

24. Schlegl T, Seeböck P, Waldstein SM, Schmidt-Erfurth U, Langs G. Unsupervised Detection of Lesions in Brain MRI using Constrained Adversarial Auto-encoders. *Information Processing in Medical Imaging*. MIDL. 2018:.

25. Schlegl T, Seeböck P, Waldstein SM, Langs G, Schmidt-Erfurth U. f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks. *Medical Image Analysis*. 2019;54:30-44.

26. Baur C, Wiestler B, Albarqouni S, Navab N. Deep Autoencoding Models for Unsupervised Anomaly Segmentation in Brain MR Images. *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*. Springer International Publishing. 2019:161-9.

27. Kascenas A, Pugeault N, O'Neil AQ. Denoising autoencoders for unsupervised anomaly detection in brain MRI. *International Conference on Medical Imaging with Deep Learning*. PMLR. 2022:653-64.

28. Silva-Rodríguez J, Naranjo V, Dolz J. Constrained unsupervised anomaly segmentation. *Medical Image Analysis*. 2022;80:102526.

29. Nguyen B, Feldman A, Bethapudi S, Jennings A, Willcocks CG. Unsupervised Region-Based Anomaly Detection In Brain MRI With Adversarial Image Inpainting. *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*. 2021:1127-31.

30. Bercea CI, Wiestler B, Rueckert D, Schnabel JA. Generalizing Unsupervised Anomaly Detection: Towards Unbiased Pathology Screening. *Medical Imaging with Deep Learning*. 2023:.

31. Zhao R, Yaman B, Zhang Y, Stewart R, Dixon A, Knoll F, et al. fastMRI+: Clinical Pathology Annotations for Knee and Brain Fully Sampled Multi-Coil MRI Data. *arXiv:*

Computer Vision and Pattern Recognition. 2021;arXiv:2109.03812.

32. Wolleb J, Bieder F, Sandkühler R, Cattin PC. Diffusion Models for Medical Anomaly Detection. *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*. Springer Nature Switzerland. 2022:35-45.

33. Croitoru FA, Hondru V, Ionescu RT, Shah M. Diffusion Models in Vision: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2023;arXiv:2109.03812:1-20.

34. Wyatt J, Leach A, Schmon SM, Willcocks CG. AnoDDPM: Anomaly Detection with Denoising Diffusion Probabilistic Models using Simplex Noise. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE. 2022:649-55.

35. Pinaya WHL, Tudosiu PD, Gray RJ, Rees GE, Nachev P, Ourselin S, et al. Unsupervised Brain Anomaly Detection and Segmentation with Transformers. *International Conference on Medical Imaging with Deep Learning*. IEEE. 2021:.

36. Kascenas A, Sanchez P, Schrempf P, Wang C, Clackett W, Mikhael SS, et al. The role of noise in denoising models for anomaly detection in medical images. *arXiv preprint arXiv:230108330*. 2023.

37. Behrendt F, Bhattacharya D, Krüger J, Opfer R, Schlaefer A. Patched diffusion models for unsupervised anomaly detection in brain mri. *arXiv preprint arXiv:230303758*. 2023.

38. Figley CR, Uddin MN, Wong K, Kornelsen J, Puig J, Figley TD. Potential Pitfalls of Using Fractional Anisotropy, Axial Diffusivity, and Radial Diffusivity as Biomarkers of Cerebral White Matter Microstructure. *Frontiers in Neuroscience*. 2022;15.

39. Giannelli M, Cosottini M, Michelassi MC, Lazzarotti G, Belmonte G, Bartolozzi C, et al. Dependence of brain DTI maps of fractional anisotropy and mean diffusivity on the number of diffusion weighting directions. *Journal of Applied Clinical Medical Physics*. 2010;11(1):176-90.

40. Nir TM, Jahanshad N, Villalon-Reina JE, Isaev D, Zavaliangos-Petropulu A, Zhan L,

et al. Fractional anisotropy derived from the diffusion tensor distribution function boosts power to detect Alzheimer's disease deficits. *Magnetic Resonance in Medicine*. 2017;78(6):2322-33.

41. Filippi M, Cercignani M, Inglese M, Horsfield MA, Comi G. Diffusion tensor magnetic resonance imaging in multiple sclerosis. *Neurology*. 2001;56(3):304-11.

42. Zhang Y, Schuff N, Jahng GH, Bayne W, Mori S, Schad L, et al. Diffusion tensor imaging of cingulum fibers in mild cognitive impairment and Alzheimer disease. *Neurology*. 2007;68(1):13-9.

43. Péran P, Cherubini A, Assogna F, Piras F, Quattrocchi C, Peppe A, et al. Magnetic resonance imaging markers of Parkinson's disease nigrostriatal signature. *Brain*. 2010;133(11):3423-33.

44. Veeramuthu V, Narayanan V, Kuo TL, Delano-Wood L, Chinna K, Bondi MW, et al. Diffusion Tensor Imaging Parameters in Mild Traumatic Brain Injury and Its Correlation with Early Neuropsychological Impairment: A Longitudinal Study. *Journal of Neurotrauma*. 2015;32(19):1497-509.

45. Feller W. On the Theory of Stochastic Processes, with Particular Reference to Applications. *Proceedings of the [First] Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press. 1949:403-33.

46. Mirza M, Osindero S. Conditional generative adversarial nets. *arXiv preprint arXiv:14111784*. 2014.

47. Isola P, Zhu JY, Zhou T, Efros AA. Image-to-Image Translation with Conditional Adversarial Networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017:5967-76.

48. Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B. High-Resolution Image Synthesis with Latent Diffusion Models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2022:10674-85.

49. Van Essen DC, Ugurbil K, Auerbach E, Barch D, Behrens TEJ, Bucholz R, et al.

The Human Connectome Project: A data acquisition perspective. *NeuroImage*. 2012;62(4):2222-31.

50. Baid U, Ghodasara S, Mohan S, Bilello M, Calabrese E, Colak E, et al. The RSNA-ASNR-MICCAI BraTS 2021 Benchmark on Brain Tumor Segmentation and Radiogenomic Classification. 2021.

51. Lesjak Ž, Galimzianova A, Koren A, Lukin M, Pernuš F, Likar B, et al. A novel public MR image dataset of multiple sclerosis patients with lesion segmentations based on multi-rater consensus. *Neuroinformatics*. 2018;16:51-63.

52. Rohlfing T, Zahr NM, Sullivan EV, Pfefferbaum A. The SRI24 multichannel atlas of normal adult human brain structure. *Human Brain Mapping*. 2010;31(5):798-819.

53. Jolicoeur-Martineau A, Piché-Taillefer R, Mitliagkas I, des Combes RT. Adversarial score matching and improved sampling for image generation. *International Conference on Learning Representations*. 2020:.

54. Dhariwal P, Nichol A. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*. 2021;34:8780-94.

55. Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015:234-41.

56. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*. 2004;13(4):600-12.

57. Brunet D, Vrscay ER, Wang Z. On the Mathematical Properties of the Structural Similarity Index. *IEEE Transactions on Image Processing*. 2012;21(4):1488-99.

58. Sara U, Akter M, Uddin MS. Image quality assessment through FSIM, SSIM, MSE and PSNR—a comparative study. *Journal of Computer and Communications*. 2019;7(3):8-18.

59. Wardhani NWS, Rochayani MY, Iriany A, Sulistyono AD, Lestantyo P. Cross-validation Metrics for Evaluating Classification Performance on Imbalanced Data. *2019 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*. IEEE. 2019:14-8.

60. Fletcher S, Islam MZ. Comparing sets of patterns with the Jaccard index. *Australasian Journal of Information Systems*. 2018;22.

APPENDIX A: ALGORITHMS

The python code implemented in this thesis work is provided through Algorithm A.2 to Algorithm A.21 and the inference algorithm is provided in Algorithm A.22.

Algorithm A.1. Training Code for Conditional DDPM

```
import math
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tqdm import tqdm
from tensorflow.python.platform import tf_logging as logging
from keras.utils import io_utils
from tqdm.auto import tqdm
import os
import datetime

tf.keras.utils.set_random_seed(7777)
np.random.seed(7777)
```

Algorithm A.2. Training Code for Conditional DDPM

```
# Datagenerator for training
class datagencondition(tf.keras.utils.Sequence):
    def __init__(self, path_context,path_input,
                 batch_size
                 ):
        self.path_context=path_context
        self.path_input=path_input

        self.files_x=os.listdir(path_context)
        self.files=[]
        for file in self.files_x:
            if ".npy" in file:
              self.files.append(file)
        self.n = len(self.files)
        self.batch_size=batch_size
        self.trunc_tresh=0.2

    def __load_data__(self,i):
        condition=np.load(os.path.join(self.path_context,
                                self.files[i]))
        condition=np.expand_dims(condition,axis=-1)
        x_path_file=self.files[i]
        x=np.load(os.path.join(self.path_input,x_path_file))
        x=np.expand_dims(x,axis=-1)
        x=(2*x)-1
        return x,np.float32(condition),self.files[i]

    def __get_batch__(self,index_interval):
        i_start=index_interval[0]
        i_end=index_interval[1]
        batch=[]
        batch_condition=[]
        batch_files=[]
        for i in range(i_start,i_end):
          x,condition,file_name=self.__load_data__(i)
          batch.append(x)
          batch_condition.append(condition)
          batch_files.append(file_name)
        return (np.asarray(batch),
                np.asarray(batch_condition),batch_files)
```

Algorithm A.3. Training Code for Conditional DDPM

```

   def __getitem__(self, index):
        X,X_Con,files=self.__get_batch__(index_interval=
        [index * self.batch_size,(index + 1) * self.batch_size])
        return X,X_Con,files

    def __len__(self):
        return self.n // self.batch_size

#Adding and extracting noise values in a corresponding time schedule.
class GaussianDiffusion:
    def __init__(
        self,
        beta_start=1e-4,
        beta_end=0.02,
        timesteps=1000,
        clip_min=-1.0,
        clip_max=1.0,
    ):
        self.beta_start = beta_start
        self.beta_end = beta_end
        self.timesteps = timesteps
        self.clip_min = clip_min
        self.clip_max = clip_max

        # Define the linear variance schedule
        betas =  np.linspace(
            beta_start,
            beta_end,
            timesteps,
            dtype=np.float64,  # Using float64 for better precision
        )
        self.betas=betas
        self.num_timesteps = int(timesteps)
        alphas = 1.0 - betas
        alphas_cumprod = np.cumprod(alphas, axis=0)
        alphas_cumprod_prev = np.append(1.0, alphas_cumprod[:-1])
```

Algorithm A.4. Training Code for Conditional DDPM

```
        self.betas = tf.constant(betas, dtype=tf.float32)
        self.alphas_cumprod = tf.constant(alphas_cumprod,
                                                dtype=tf.float32)
        self.alphas_cumprod_prev = tf.constant(alphas_cumprod_prev,
                                                dtype=tf.float32)

        # Calculations for diffusion q(x_t | x_{t-1}) and others
        self.sqrt_alphas_cumprod = tf.constant(
            np.sqrt(alphas_cumprod), dtype=tf.float32
        )

        self.sqrt_one_minus_alphas_cumprod = tf.constant(
            np.sqrt(1.0 - alphas_cumprod), dtype=tf.float32
        )

        self.log_one_minus_alphas_cumprod = tf.constant(
            np.log(1.0 - alphas_cumprod), dtype=tf.float32
        )

 self.sqrt_recip_alphas_cumprod = tf.constant(
            np.sqrt(1.0 / alphas_cumprod), dtype=tf.float32
        )
        self.sqrt_recipm1_alphas_cumprod = tf.constant(
            np.sqrt(1.0 / alphas_cumprod - 1), dtype=tf.float32
        )
        # Calculations for posterior q(x_{t-1} | x_t, x_0)
        posterior_variance = (
          betas*(1.0-alphas_cumprod_prev) / (1.0-alphas_cumprod)
        )
        self.posterior_variance = tf.constant(posterior_variance,
                                                dtype=tf.float32)
        self.posterior_log_variance_clipped = tf.constant(
            np.log(np.maximum(posterior_variance, 1e-20)),
                                dtype=tf.float32
        )
```

Algorithm A.5. Training Code for Conditional DDPM

```
        self.posterior_mean_coef1 = tf.constant(
            betas * np.sqrt(alphas_cumprod_prev) /
        (1.0 - alphas_cumprod),
            dtype=tf.float32,
        )

        self.posterior_mean_coef2 = tf.constant(
            (1.0 - alphas_cumprod_prev) * np.sqrt(alphas) /
         (1.0 - alphas_cumprod),
            dtype=tf.float32,
        )
    #Converts extracted informations into a tensor
     def _extract(self, a, t, x_shape):

        batch_size = x_shape[0]
        out = tf.gather(a, t)
        return tf.reshape(out, [batch_size, 1, 1, 1,1])
    #Extraction of mean and variance at current timesteo
     def q_mean_variance(self, x_start, t):

        x_start_shape = tf.shape(x_start)
        mean = self._extract(self.sqrt_alphas_cumprod, t,
                                 x_start_shape) * x_start
        variance = self._extract(1.0 - self.alphas_cumprod, t,
                                           x_start_shape)
        log_variance = self._extract(
            self.log_one_minus_alphas_cumprod, t, x_start_shape
        )
        return mean, variance, log_variance
    #Diffusing data with noise
     def q_sample(self, x_start, t, noise):

        x_start_shape = tf.shape(x_start)
        return (
            self._extract(self.sqrt_alphas_cumprod, t,
                          tf.shape(x_start)) * x_start
            + self._extract(self.sqrt_one_minus_alphas_cumprod,
         t, x_start_shape)
            * noise
        )
```

Algorithm A.6. Training Code for Conditional DDPM

```
    def predict_start_from_noise(self, x_t, t, noise):
        x_t_shape = tf.shape(x_t)
        return (
            self._extract(self.sqrt_recip_alphas_cumprod, t,
                          x_t_shape) * x_t
            - self._extract(self.sqrt_recipm1_alphas_cumprod,
                          t, x_t_shape) * noise
        )
    #Computing the mean and variance of the diffusion posterior.
    def q_posterior(self, x_start, x_t, t):

        x_t_shape = tf.shape(x_t)
        posterior_mean = (
            self._extract(self.posterior_mean_coef1, t, x_t_shape)
         * x_start
            + self._extract(self.posterior_mean_coef2, t, x_t_shape)
         * x_t
        )
        posterior_variance = self._extract(self.posterior_variance,
                                           t, x_t_shape)
        posterior_log_variance_clipped = self._extract(
            self.posterior_log_variance_clipped, t, x_t_shape
        )
        return (posterior_mean, posterior_variance,
                posterior_log_variance_clipped)

    def p_mean_variance(self, pred_noise, x, t,clip_denoised=False):
        x_recon = self.predict_start_from_noise(x, t=t,
                                        noise=pred_noise)
        if clip_denoised:
            x_recon = tf.clip_by_value(x_recon, self.clip_min,
                                        self.clip_max)

        model_mean, posterior_variance, posterior_log_variance =
        self.q_posterior(x_start=x_recon, x_t=x, t=t)
        return (model_mean, posterior_variance,
                posterior_log_variance)
```

Algorithm A.7. Training Code for Conditional DDPM

```
    #Sampling and denoising from diffusion model.
    def p_sample(self, pred_noise, x, t,clip_denoised=False):

        model_mean, _, model_log_variance = self.p_mean_variance(
            pred_noise, x=x, t=t, clip_denoised=clip_denoised
        )
        noise = tf.random.normal(shape=x.shape, dtype=x.dtype)
        # No noise when t == 0
        nonzero_mask = tf.reshape(
            1 - tf.cast(tf.equal(t, 0), tf.float32),
                [tf.shape(x)[0], 1, 1, 1,1])
        return (model_mean + nonzero_mask *
                tf.exp(0.5 * model_log_variance) * noise

def kernel_init(scale):
    scale = max(scale, 1e-10)
    return keras.initializers.VarianceScaling(
        scale, mode="fan_avg", distribution="uniform"
    )

class AttentionBlock(layers.Layer):
    def __init__(self, units, groups=8, **kwargs):
        self.units = units
        self.groups = groups
        super().__init__(**kwargs)

        self.query = layers.Conv3D(units,kernel_size=1,
                padding="same", kernel_initializer=kernel_init(1.0))
        self.key = layers.Conv3D(units,kernel_size=1,
                padding="same", kernel_initializer=kernel_init(1.0))
        self.value = layers.Conv3D(units,kernel_size=1,
                padding="same", kernel_initializer=kernel_init(1.0))
        self.proj =layers.Conv3D(units,kernel_size=1,
                padding="same", kernel_initializer=kernel_init(0))

    def build(self,input_shape):
        in_channels=input_shape[-1]
        if in_channels <= 8:
            groups = in_channels // 4
        else:
            groups = 8
        self.norm = layers.GroupNormalization(groups=groups)
```

Algorithm A.8. Training Code for Conditional DDPM

```
    def call(self, inputs):
        batch_size = tf.shape(inputs)[0]
        height = tf.shape(inputs)[1]
        width = tf.shape(inputs)[2]
        depth = tf.shape(inputs)[3]
        scale = tf.cast(self.units, tf.float32) ** (-0.5)

        q = self.query(inputs)
        k = self.key(inputs)
        v = self.value(inputs)

        attn_score = tf.einsum("bhwdc, bHWDc -> bhwdHWD",q,k)*scale
        attn_score = tf.reshape(attn_score,
                        [batch_size, height, width, depth,
                        height * width * depth])

        attn_score = tf.nn.softmax(attn_score, -1)
        attn_score = tf.reshape(attn_score,
                        [batch_size, height, width, depth,
                        height, width,depth])

        proj = tf.einsum("bhwdHWD,bHWDc->bhwdc", attn_score, v)
        proj = self.proj(proj)
        return inputs + proj

class CrossAttentionBlock(layers.Layer):
    def __init__(self, units, groups=8, **kwargs):
        self.units = units
        self.groups = groups
        super().__init__(**kwargs)

        self.query = layers.Conv3D(units,1,"same",kernel_init(1.0))
        self.key = layers.Conv3D(units,1,"same", kernel_init(1.0))
        self.value = layers.Conv3D(units,1,"same",kernel_init(1.0))
        self.proj =layers.Conv3D(units,1,"same", kernel_init(0))

    def build(self,input_shape):
        in_channels=input_shape[0][-1]
        if in_channels <= 8:
            groups = in_channels // 4
        else:
            groups = 8
        self.norm = layers.GroupNormalization(groups=groups)
```

Algorithm A.9. Training Code for Conditional DDPM

```
    def call(self, inputs):
        inputs, context = inputs
        context = inputs if context is None else context

        batch_size = tf.shape(inputs)[0]
        height = tf.shape(inputs)[1]
        width = tf.shape(inputs)[2]
        depth = tf.shape(inputs)[3]
        scale = tf.cast(self.units, tf.float32) ** (-0.5)

        inputs = self.norm(inputs)
        q = self.query(inputs)
        k = self.key(context)
        v = self.value(context)

        attn_score = tf.einsum("bhwdc, bHWDc -> bhwdHWD",q k)*scale
        attn_score = tf.reshape(attn_score,
                        [batch_size, height, width, depth,
                         height * width * depth])

        attn_score = tf.nn.softmax(attn_score, -1)
        attn_score = tf.reshape(attn_score,
                        [batch_size, height, width, depth,
                         height, width,depth])

        proj = tf.einsum("bhwdHWD,bHWDc->bhwdc", attn_score, v)
        proj = self.proj(proj)
        return inputs + proj

class TimeEmbedding(layers.Layer):
    def __init__(self, dim, **kwargs):
        super().__init__(**kwargs)
        self.dim = dim
        self.half_dim = dim // 2
        self.emb = math.log(10000) / (self.half_dim - 1)
        self.emb = tf.exp(tf.range(self.half_dim,
                           dtype=tf.float32) * -self.emb)
```

Algorithm A.10. Training Code for Conditional DDPM

```
    def call(self, inputs):
        inputs = tf.cast(inputs, dtype=tf.float32)
        emb = inputs[:, None] * self.emb[None, :]
        emb = tf.concat([tf.sin(emb), tf.cos(emb)], axis=-1)
        return emb

def gelu(x):
    tanh_res = keras.activations.tanh(x * 0.7978845608 *
                                      (1 + 0.044715 * (x**2)))
    return 0.5 * x * (1 + tanh_res)

class GEGLU(keras.layers.Layer):
    def __init__(self, dim_out):
        super().__init__()
        self.proj=layers.Conv3D(dim_out*2,1,"same", kernel_init(1.0))
        self.dim_out = dim_out

    def call(self, x):
        xp = self.proj(x)
        x, gate = xp[..., : self.dim_out], xp[..., self.dim_out :]
        return x * gelu(gate)

def normalize(inputs):
    in_channels = inputs.shape[4]
    if in_channels <= 8:
        num_groups = in_channels // 4
    else:
        num_groups = 8
    x = layers.GroupNormalization(num_groups,axis=-1,
                                  epsilon=1e-6,)(inputs)
    return x

def DownSample(width):
    def apply(x):
        x = layers.Conv3D(
            width,
            kernel_size=3,
            strides=2,
            padding="same",
            kernel_initializer=kernel_init(1.0),
        )(x)
        return x
    return apply
```

Algorithm A.11. Training Code for Conditional DDPM

```
def UpSample(width, interpolation="nearest"):
    def apply(x):
        x = layers.UpSampling3D(size=2)(x)
        x = layers.Conv3D(
            width, 3, "same",kernel_init(1.0)
        )(x)
        return x
    return apply

def TimeMLP(units, activation_fn=keras.activations.swish):
    def apply(inputs):
        temb = layers.Dense(
            units, activation_fn, kernel_init(1.0)
        )(inputs)
        temb = layers.Dense(units,kernel_init(1.0))(temb)
        return temb
    return apply

def ResidualBlock(width, activation_fn=keras.activations.swish):
    def apply(inputs):
        x, t = inputs
        input_width = x.shape[4]

        if input_width == width:
            residual = x
        else:
            residual = layers.Conv3D(
                width, 1, kernel_init(1.0)
            )(x)

        temb = activation_fn(t)
        temb = layers.Dense(width, kernel_init(1.0))(temb)[
            :, None, None, :
        ]
```

Algorithm A.12. Training Code for Conditional DDPM

```
        x = normalize(x)
        x = activation_fn(x)
        x = layers.Conv3D(
            width, 3, "same", kernel_init(1.0)
        )(x)

        x = layers.Add()([x, temb])
        x = normalize(x)
        x = activation_fn(x)

        x = layers.Conv3D(
            width, 3, "same", kernel_init(0.0)
        )(x)
        x = layers.Add()([x, residual])
        return x
    return apply

class ReScaler(keras.layers.Layer):
  def __init__(self, orig_shape):
    super().__init__()
    self.orig_shape=orig_shape

  def build(self,input_shape):
    size_n=int(input_shape[1])//int(self.orig_shape[1])
    self.average_pooling=tf.keras.layers.AveragePooling3D(
                                                (size_n,size_n,
                                                size_n)
                                                )
  def call(self, inputs):
      return self.average_pooling(inputs)

class BasicTransformerBlock(keras.layers.Layer):
    def __init__(self, dim,):
        super().__init__()
        self.attn1 = AttentionBlock(dim)
        self.attn2 = CrossAttentionBlock(dim)
        self.geglu = GEGLU(dim * 4)
        self.dense  =layers.Conv3D(dim,1,"same",kernel_init(1.0))

    def call(self, inputs):
        x, context = inputs
        x = self.attn1(x) + x
        x = self.attn2([x, context]) + x
        return self.dense(self.geglu(x)) + x
```

Algorithm A.13. Training Code for Conditional DDPM

```
class SpatialTransformer(keras.layers.Layer):
    def __init__(self, channels):
        super().__init__()
        self.norm = layers.GroupNormalization(groups=8,epsilon=1e-5)
        self.proj_in=layers.Conv3D(channels,1,"same",
                                   kernel_init(1.0))
        self.transformer_blocks = [BasicTransformerBlock(channels)]
        self.proj_out=layers.Conv3D(channels,1,"same",
                                   kernel_init(1.0))

    def build(self,input_shape):
        shape=input_shape[0]
        self.scaler=ReScaler(orig_shape=shape)

    def call(self, inputs):
        x, context = inputs
        context = x if context is None else context
        if context is not None:
          context=self.scaler(context)
        b, h, w, d,c = x.shape
        x_in = x
        x = self.norm(x)
        x = self.proj_in(x)
        for block in self.transformer_blocks:
            x = block([x, context])
        return self.proj_out(x) + x_in

def build_unet_model(
    input_shape,
    context_shape,
    widths,
    has_attention,
    channel_multiplier,
    num_res_blocks=2,
    activation_fn=keras.activations.swish,
    first_conv_channels=16,
    use_spatial_transformer=True
):
```

Algorithm A.14. Training Code for Conditional DDPM

```
    widths=[first_conv_channels*mult for mult in channel_multiplier]

    image_input = layers.Input(
        shape=input_shape, name="image_input"
    )

    context = layers.Input(
        shape=context_shape, name="context"
    )

    time_input = keras.Input(shape=(), dtype=tf.int64,
                                name="time_input")

    x = layers.Conv3D(
        first_conv_channels,
        kernel_size=3,
        padding="same",
        kernel_initializer=kernel_init(1.0),
    )(image_input)

    temb = TimeEmbedding(dim=first_conv_channels * 4)(time_input)
    temb = TimeMLP(units=first_conv_channels * 4,
                        activation_fn=activation_fn)(temb)
    skips = [x]

    # DownBlock
    for i in range(len(widths)):
        for _ in range(num_res_blocks):
            x = ResidualBlock(
                widths[i], activation_fn=activation_fn
            )([x, temb])
            if has_attention[i]:
                if use_spatial_transformer==True:
                    x = SpatialTransformer(widths[i])([x,context])
                else:
                    context_x=ReScaler(
                                orig_shape=tf.shape(x)[0])(context)
                    x=CrossAttentionBlock(widths[i])([x,context_x])
            skips.append(x)

        if widths[i] != widths[-1]:
            x = DownSample(widths[i])(x)
            skips.append(x)
```

Algorithm A.15. Training Code for Conditional DDPM

```
    # MiddleBlock
    x = ResidualBlock(widths[-1] ,activation_fn=activation_fn)(
        [x, temb]
    )

    if use_spatial_transformer==True:
        x = SpatialTransformer(widths[i])([x,context])
    else:
        context_x=ReScaler(orig_shape=tf.shape(x)[0])(context)
        x = CrossAttentionBlock(widths[i])([x,context_x])

    x = ResidualBlock(widths[-1], activation_fn=activation_fn)(
        [x, temb]
    )

    # UpBlock
    for i in reversed(range(len(widths))):
        for _ in range(num_res_blocks + 1):
            x = layers.Concatenate(axis=-1)([x, skips.pop()])
            x = ResidualBlock(
                widths[i],activation_fn=activation_fn
            )([x, temb])
            if has_attention[i]:
              if use_spatial_transformer==True:
                  x = SpatialTransformer(widths[i])([x,context])
              else:
                  context_x=ReScaler(tf.shape(x)[0])(context)
                  x = CrossAttentionBlock(widths[i])([x,context_x])

        if i != 0:
            x = UpSample(widths[i])(x)

    # End block

    x = normalize(x)
    x = activation_fn(x)
    x = layers.Conv3D(input_shape[-1], (3,3,3),"same",
                      kernel_init(0.0))(x)
    return keras.Model([image_input, context, time_input], x,
                       name="conditional_unet")
```

Algorithm A.16. Training Code for Conditional DDPM

```
class DiffusionModel(keras.Model):
    def __init__(self,input_shape,context_shape, timesteps,
                checkpoint_path,special_checkpoint=None,
                first_conv_channels=16,
                channel_multiplier=[1, 2, 4, 8, 16],
                has_attention=[False, False, False, True, True],
                   num_res_blocks=2,
                   ):
        super(DiffusionModel,self).__init__()
        self.shape=input_shape
        self.c_shape = context_shape
        self.first_conv_channels = first_conv_channels
        self.channel_multiplier = channel_multiplier
        self.widths = [first_conv_channels *
         mult for mult in channel_multiplier]
        self.has_attention = has_attention
        self.num_res_blocks = num_res_blocks
        self.network = build_unet_model(
            input_shape=self.shape,
            context_shape = self.c_shape,
            widths=self.widths,
            has_attention=self.has_attention,
            num_res_blocks=self.num_res_blocks,
            first_conv_channels=self.first_conv_channels,
            channel_multiplier = self.channel_multiplier
        )
        self.timesteps=timesteps

        # Get an instance of the Gaussian Diffusion utilities
        self.gdf_util = GaussianDiffusion(timesteps=self.timesteps)
        self.checkpoint_dir = os.path.join(checkpoint_path,
                                           "checkpoint")
        if not os.path.isdir(self.checkpoint_dir):
          os.makedirs(self.checkpoint_dir)
          print("New Checkpoint Folder Initialized...")

        self.checkpoint_prefix = os.path.join(self.checkpoint_dir,
                                           "ckpt")+
        datetime.datetime.now().strftime("\%Y\%m\%d-\%H\%M\%S")
        self.special_checkpoint=special_checkpoint
```

Algorithm A.17. Training Code for Conditional DDPM

```
          else:
            tf.print("---------Initializing from scratch---------")

          if return_epoch_number==True:
            return int(self.checkpoint.epoch)+1

    def train_step(self, inputs):
        # Get the batch size
        x,context,_=inputs
        batch_size = tf.shape(x)[0]
        self.checkpoint.step.assign_add(1)
        tf.summary.scalar(
        'steps', data=self.checkpoint.step,
         step=self.checkpoint.step
         )

        # Sample timesteps uniformly
        t = tf.random.uniform(
            minval=0, maxval=self.timesteps,
            shape=(batch_size,), dtype=tf.int64
        )

        with tf.GradientTape() as tape:
            # Sample random noise to added to images in the batch
            noise = tf.random.normal(shape=tf.shape(x),
             dtype=x.dtype
             )
            # Diffuse the images with noise
            x_t = self.gdf_util.q_sample(x, t, noise)

            #  Pass the diffused images and time steps to network
            pred_noise = self.network([x_t,context, t],
            training=True
            )

            # Calculate the loss
            loss = self.loss(noise, pred_noise)

        # Get the gradients
        gradients = tape.gradient(loss,
        self.network.trainable_weights)
```

Algorithm A.18. Training Code for Conditional DDPM

```
    def load(self,usage="train",return_epoch_number=True):
          self.checkpoint = tf.train.Checkpoint(
                                              network=self.network,
                                              optimizer=self.optimizer,
                                              step=tf.Variable(0),
                                              epoch=tf.Variable(0)
                                          )
          try:
            latest_checkpoint=tf.train.latest_checkpoint(
            self.checkpoint_dir
            )
          except:
            latest_checkpoint=None
            pass

          if (latest_checkpoint!=None and
                self.special_checkpoint==None):
              if usage=="eval":
                  self.checkpoint.restore(
                  latest_checkpoint
                  ).expect_partial()
              else:
                  self.checkpoint.restore(latest_checkpoint)
              tf.print("--------Restoring from {}--------".format(
              latest_checkpoint)
              )
              tf.print("-----------Step {}-----------".format(
              int(self.checkpoint.step))
              )
          elif self.special_checkpoint!=None:
              if usage=="eval":
                  self.checkpoint.restore(
                  self.special_checkpoint).expect_partial()
              else:
                  self.checkpoint.restore(self.special_checkpoint)
              tf.print("-----------Restoring from {}-----------".
              format(
                  self.special_checkpoint))
              tf.print("-----------Step {}-----------".format(
              int(self.checkpoint.step))
              )
```

Algorithm A.19. Training Code for Conditional DDPM

```
        # Update the weights of the network
        self.optimizer.apply_gradients(
        zip(gradients, self.network.trainable_weights)
        )

        #  Return loss values
        return {"loss": loss}

    def test_step(self, inputs):
        # Get the batch size
        x,context,_=inputs
        batch_size = tf.shape(x)[0]

        # Sample timesteps uniformly
        t = tf.random.uniform(
            minval=0, maxval=self.timesteps,
             shape=(batch_size,), dtype=tf.int64
        )

        # Sample random noise to be added to the images in the batch
        noise = tf.random.normal(shape=tf.shape(x), dtype=x.dtype)

        # Diffuse the images with noise
        x_t = self.gdf_util.q_sample(x, t, noise)

        # Pass the diffused images and time steps to the network
        pred_noise = self.network([x_t,context, t], training=False)

        # Calculate the loss
        loss = self.loss(noise, pred_noise)

        # Return loss values
        return {"loss": loss}

    def generate_from_anomaly(
    self,image,mask,condition,num_models=1
    ):
        # Randomly sample noise (starting point for reverse process)
        samples = tf.random.normal(
            shape=(num_models, self.shape[0], self.shape[1],
            self.shape[2],self.shape[3]), dtype=tf.float32
        )
```

Algorithm A.20. Training Code for Conditional DDPM

```
        time_range = reversed(range(0,self.timesteps))
        print(f"Running DDPM Sampling {self.timesteps} timesteps")
        iterator = tqdm(time_range, desc='DDPM Sampler',
        total=self.timesteps
        )
        for t, step in enumerate(iterator):
          tt = np.full((num_models,), step)
          pred_noise = self.network.predict(
              [samples,condition,tt], verbose=0,
              batch_size=num_models
          )
          alpha_cumprod = self.gdf_util.alphas_cumprod[step]
          gt_weight = tf.math.sqrt(alpha_cumprod)
          gt_part = gt_weight*image

          noise_weight = tf.math.sqrt(1-alpha_cumprod)
          noise_part = noise_weight*tf.random.normal(samples.shape)

          weighted_gt =  gt_part + noise_part

          samples = (mask * samples)+ ((1 - mask) * (weighted_gt))

          samples = self.gdf_util.p_sample(
              pred_noise, samples, tt, clip_denoised=True
          )
        # Return generated samples
        return samples,pred_noise

num_epochs = 400
timesteps = 1000
learning_rate = 1e-4
save_per_epoch = 5
input_shape=(128,128,128,1)
context_shape = (128,128,128,1)
step_warmup = 15000
first_conv_channels = 16
channel_multiplier = [1,2,4,8,16]
has_attention = [False, False, False, True, True]
num_res_blocks = 2
special_checkpoint=None
batch_size=1
```

Algorithm A.21. Training Code for Conditional DDPM

```

model = DiffusionModel(
    input_shape=input_shape,context_shape = context_shape,
    timesteps=timesteps,checkpoint_path=checkpoint_path,
    special_checkpoint=special_checkpoint,
    first_conv_channels=first_conv_channels,
    channel_multiplier=channel_multiplier,
    has_attention=has_attention,
    num_res_blocks=num_res_blocks,
)

model.compile(
    loss=keras.losses.MeanSquaredError(),
    optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
)

initial_epoch = model.load(return_epoch_number=True)

callbacks=[]
callbacks.append(SaveCheckpoint(number_epoch=num_epochs,
        per_epoch=save_per_epoch))
callbacks.append(LearningRateStepScheduler(initial_lr=learning_rate,
        step_warmup=step_warmup))

traingen = datagencondition(path_context=condition_path,
                            path_input = data_path,
                            batch_size = batch_size)

model.fit(
    traingen,
    epochs=num_epochs,
    batch_size=batch_size,
    callbacks=callbacks,
    initial_epoch=initial_epoch
)
```

Algorithm A.22. Inference Code for Anomaly Detection with Conditional DDPM

```

mask_size=16
x,x_cond,dir = testgen.__getitem__(0)
split = dir[0].split('.')
fullname = split[0].split('_')
name = fullname[0]
save_path = '/content/drive/MyDrive/full_reconstruction'
x_tmp = x.squeeze()
x_tmp = (x_tmp+1)/2
mask_size=16
coordinates = []
for i in range(0,128,mask_size):
    for j in range(0,128,mask_size):
        for k in range(0,128,mask_size):
            z=x_tmp[i:i+mask_size,j:j+mask_size,k:k+mask_size]
            ratio=len(np.where(z==0)[0])/mask_size**3
            if ratio<0.3:
                coordinates.append([i,j,k])

for coor in coordinates:
  i=coor[0]
  j=coor[1]
  k=coor[2]
  if not os.path.isfile(save_path+f'/{name}_{i}_{j}_{k}.npy'):
    cubemask = np.zeros_like(x,dtype=bool)
    cubemask[0,i:i+mask_size,j:j+mask_size,k:k+mask_size,0]=True
    famask = np.logical_and(x_cond,cubemask)
    masked_img = x-x*famask
    generated_part,_ = model.generate_from_anomaly_wot0(
        masked_img,np.float32(famask),x_cond
        )
    np.save(save_path+f'/{name}_{i}_{j}_{k}',generated_part)
  else:
    print(f'There was {name}_{i}_{j}_{k} skipped')
```