University College London

MSc Project Dissertation

Generative Probabilistic Convolutional Occupancy Networks for 3D Data Synthesis

Xiyao LI

supervised by **Prof. Daniel ALEXANDER**

and **Dr. Moucheng XU**

September 2023

This report is submitted as part requirement for the MSc Degree in Computer Graphics, Vision & Imaging, at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Abstract

Recently, implicit neural representation has gained popularity in the field of learning-based 3D reconstruction tasks. In 2020, a framework based on implicit neural representation, known as Convolutional Occupancy Networks [1], was proposed for detailed reconstruction ranging from individual objects to entire 3D scenes. However, the 3D construction results generated by this framework are deterministic. The primary objective of our project is to enhance Convolutional Occupancy Networks by incorporating Bayesian probabilistic estimation into its latent space. This probabilistic extension equips our model with the capability to generate synthetic 3D data. We initially validate our model using the ShapeNet dataset as a proof of concept. For evaluation, we apply our model to a biomedical dataset to synthesise 3D placenta trees. The code implementation can be found at: `https://github.com/Lixiyao-meow/Generative_Convolutional_Occupancy_Networks`.

Acknowledgements

Firstly, I would like to express my sincere gratitude to Prof. Daniel ALEXANDER for welcoming me to the Centre for Medical Image Computing (CMIC) for this project.

Additionally, I extend my heartfelt thanks to my supervisor, Dr. Moucheng XU, for his unwavering support and guidance throughout this project.

Finally, I would also like to express my appreciation to the UCL High-Performance Computing department for providing and maintaining the GPU on which I trained my model.

Contents

Chapter 1

Introduction

In the domain of deep learning research, a recurring challenge is the scarcity of data. Addressing this issue typically involves one of two primary strategies: acquiring more data from various sources or adapting deep learning models to cope with limited data. The latter approach often involves data synthesis through algorithmic means.

Synthetic data, generated algorithmically rather than derived directly from real-world observations, has gained increasing prominence as a valuable resource for training and fine-tuning deep neural networks, particularly in scenarios where genuine data is limited.

In recent years, there has been a surge of interest in leveraging deep neural networks for 3D reconstruction tasks. Notable models, such as NeRF (Neural Radiance Fields) [2], have demonstrated exceptional capabilities in reconstructing intricate 3D scenes. Furthermore, the introduction of the Convolutional Occupancy Network framework [1] represents a significant stride in the realm of translation-equivariant 3D reconstruction, spanning from individual objects to entire scenes.

However, a notable limitation of the existing Convolutional Occupancy Network lies in its

determinism. This limitation serves as the driving force for our research, which seeks to integrate Bayesian probability to the framework. To achieve this, we propose replacing the traditional latent code with a Gaussian-distributed latent space, thereby introducing a probabilistic extension. This augmentation equips the model with the capacity to generate synthetic 3D data. The fundamental approach underpinning our methodology draws inspiration from the variational autoencoder (VAE) paradigm [3]. We explore various encoder choices within the VAE framework.

Our contributions can be summarised as follows:

- We extend the Convolutional Occupancy Network [1] framework into a Probabilistic Generative Network, introducing a Gaussian distributed latent space.
- We conduct a comprehensive performance evaluation of our model on two diverse datasets, ShapeNet [4] and IntrA [5], thereby shedding light on both its conceptual strengths and weaknesses.

Chapter 2

Related Work

2.1 3D representations

Deep learning-based 3D model reconstruction typically relies on three primary explicit 3D representations: voxels, point clouds, and meshes. These explicit representations provide a clear and well-defined structure for encoding and manipulating 3D geometry. However, in addition to these explicit representations, there is a growing interest in implicit representations, which represent the 3D surface as a continuous decision boundary.

2.1.1 Voxel

The voxel representation is a three-dimensional grid that assigns a value to each cell, analogous to how pixel values represent an image in a two-dimensional grid. The concept of voxels involves subdividing the three-dimensional grid into smaller volume elements called voxels. Each voxel contains an occupancy value, indicating whether it is part of the model or not. The construction of a sampled model involves aggregating the values of the voxels that lie within the

model's boundaries.

The voxel representation is commonly employed in learning-based tasks due to its simplicity. Firman et al. [6] proposed a supervised model that utilises voxels to estimate the surface shape in the surrounding neighbourhood based on local observations in a depth image. This approach allows for completing the unobserved geometry by estimating the surface shape of objects in the region surrounding the observed depth image.

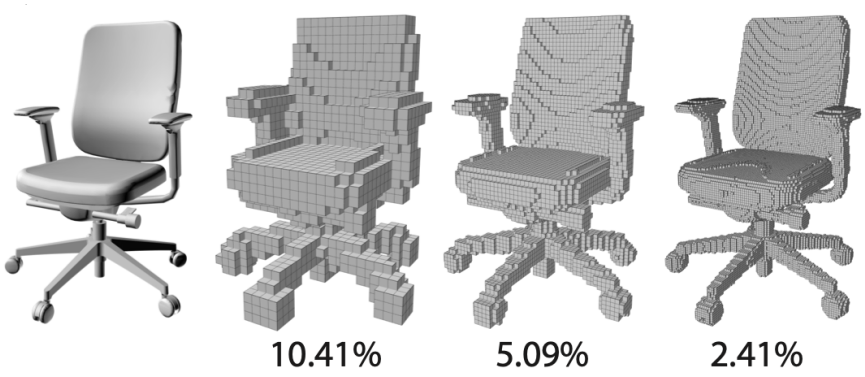


Figure 2.1: Voxel representation in resolution 30, 64 and 128

In the field of 3D shape analysis, Wang et al. [7] developed an Octree-based Convolutional Neural Network. An octree is a hierarchical data structure that extends the voxel representation by subdividing cells into eight child cells. This hierarchical subdivision enables the representation of complex shapes at varying levels of detail. Octrees are formed by recursively dividing the 3D space into eight octants until the desired level of detail is achieved.

Sedaghat et al. [8] introduced a method that incorporates object orientation as an auxiliary task to enhance 3D object recognition using voxel nets. Voxel nets are 3D convolutional neural networks designed to process voxel-based representations of 3D shapes.

However, it is worth noting that the space complexity of voxels is of the order of $\mathcal{O}(n^3)$

for a subdivision of n voxels along each dimension. As a result, the memory requirements for voxel-based representations increase significantly with higher resolutions.

2.1.2 Point cloud

The point cloud representation is a method of representing a three-dimensional model by utilising a collection of points. Unlike other representations, such as voxels or meshes, the point cloud representation preserves the original geometry of the model without any discretization. This preservation of fine-grained details makes point clouds a preferred choice for various applications related to scene understanding, such as autonomous driving and robotics. [9]

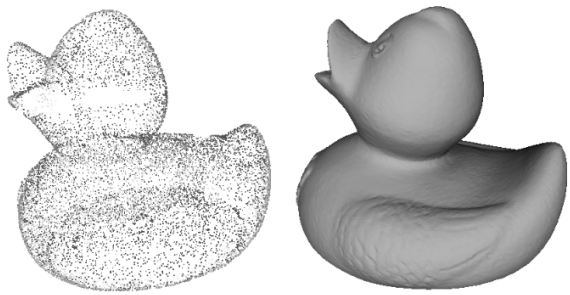


Figure 2.2: Point cloud representation

Qi et al. introduced PointNet [10], a neural network designed to directly process point clouds as input data. PointNet exhibits invariance to permutations of the input set and rigid transformations. This network architecture is well-suited for tasks such as object classification, part segmentation, and scene semantic parsing.

PointConv [11], on the other hand, is a convolutional operation specifically developed for deep learning with point clouds as input. Unlike images, which are represented on regular dense grids, point clouds are irregular and unordered, making it challenging to apply traditional

convolution operations to them. PointConv addresses this issue by proposing a dynamic filter that enables the application of convolutional networks to point clouds.

Despite the flexibility and accurate representation provided by point cloud, working with this data format requires specialised algorithms to handle their unordered data structure. Furthermore, point cloud lacks inherent semantic information about the models they represent. Depending on the task, additional processing may be necessary to extract features such as object boundaries or surface normals.

2.1.3 Mesh

The mesh representation refers to a collection of triangular elements that form 3D models. A mesh consists of vertices, edges, and faces that define the shape and structure of a 3D model. Meshes are widely used in computer graphics and 3D animation due to their ability to represent complex models with varying topology and geometry.

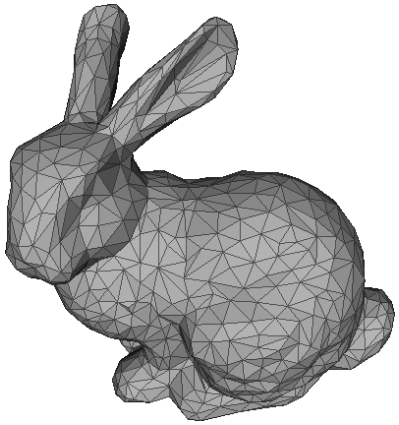


Figure 2.3: Triangular mesh representation

Hanocka et al. [12] proposed a deep neural network designed to operate directly on 3D triangular meshes. This network can be utilised for tasks such as 3D shape classification or

segmentation. The framework includes convolution, pooling, and unpooling layers that are applied directly to the edges of the mesh. By operating on the local neighbourhoods of vertices, the network can learn spatially-aware features.

Similarly, Feng et al. [13] presented a deep learning network that takes meshes as input data. This network captures and aggregates features of polygon faces in 3D shapes. It processes both spatial and structural features from the mesh and applies mesh convolution to extract relevant information. This architecture is specifically designed for 3D shape classification and retrieval tasks.

While mesh representations provide comprehensive information through vertices and faces, they often come with memory-intensive requirements, resulting in escalated computational costs. Additionally, meshes frequently lack uniform or regular topologies, making the application of standard deep learning architectures more challenging than on grid-like data.

2.1.4 Implicit surface

The representations mentioned earlier, such as voxels, point clouds, and meshes, are discrete in nature. One significant drawback of discrete representations is their limited capacity to capture the continuous nature of real-world objects. To address this limitation, researchers have explored alternative approaches that employ continuous functions as representations. However, defining such functions explicitly can be challenging.

As an alternative, researchers have proposed training neural networks to estimate continuous functions based on discrete samples. By training these networks on discrete data points, they can learn to approximate the underlying continuous function. This approach leverages the

flexibility and generalisation capabilities of neural networks to model and estimate continuous representations.

Park et al. [14] introduced a deep learning neural network for implicit representation based on a signed distance function (SDF). The SDF is a mathematical function that assigns a signed distance value to each point in space with respect to the surface of a model. The sign of the distance indicates whether the point resides inside (negative) or outside (positive) the shape, while the magnitude of the distance signifies the proximity of the point to the surface of the shape.

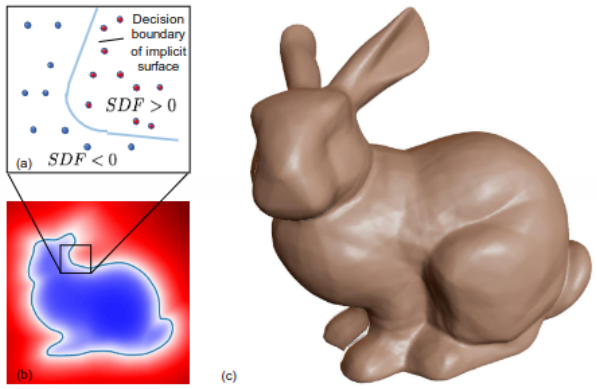


Figure 2.4: DeepSDF representation applied to the Stanford Bunny [14]

However, current network architectures designed for implicit neural representations exhibit limitations when it comes to representing models with fine detail. Additionally, they struggle to accurately model spatial and temporal derivatives, which are essential for representing signals defined implicitly by differential equations. In light of these challenges, Sitzmann et al. [15] proposed a novel approach called sinusoidal representation networks (Siren) that utilises periodic activation functions for implicit neural representations.

The Siren architecture introduces the sine function as the activation function in each layer.

Remarkably, the derivatives of the sine function are also sine functions, enabling the supervision of complex signals involving derivatives. In comparison to the traditional ReLU activation function, Siren outperforms in reconstructing results with fine details. Furthermore, Siren has demonstrated its suitability for accurately representing complex natural signals and their derivatives, making it an ideal choice for such applications.

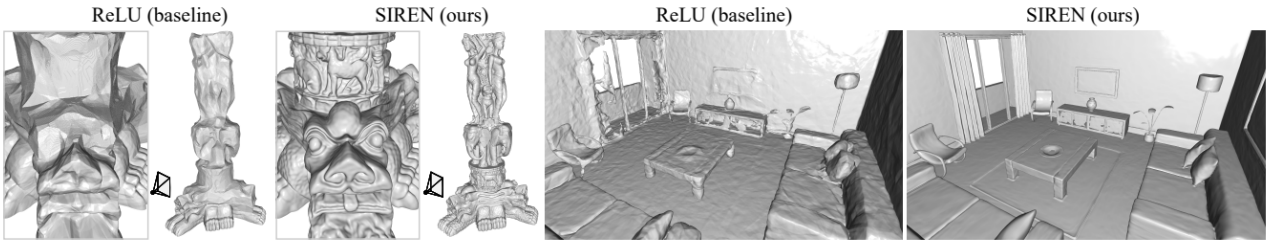


Figure 2.5: Siren shape reconstruction. Compared to ReLU implicit representations, Siren significantly improve detail of objects and complexity of entire scenes [15]

2.2 Convolutional occupancy network

In 2020, Song et al. presented the Convolutional occupancy network [1], which offers a more flexible and expressive implicit representation for detailed 3D object and scene reconstruction.

This novel approach combines convolutional encoders and implicit occupancy decoders to facilitate structured reasoning in 3D space. The network has demonstrated its capability to reconstruct complex geometry from noisy and low-resolution input data. The model has also proven its effectiveness in supporting implicit 3D reconstruction of individual objects, scaling to large indoor scenes, and exhibiting good generalisation from synthetic to real data.

2.2.1 Encoder

The network can take voxelized data and 3D point clouds as input. Specifically, for voxelized inputs, a one-layer 3D CNN is utilised, while PointNet [10] with local pooling is employed for processing 3D point clouds. The encoder initially converts the 3D input into feature representations using neural networks. These features are subsequently projected onto multiple planes or into a volume through the utilisation of average pooling techniques.

Planar encoder The authors establish the canonical plane with a resolution of $H \times W$ pixel cells, as illustrated in Figure 2.6 (a). For every input point, they extract features and project these features onto the canonical plane. Subsequently, they aggregate the projected features that fall onto the same pixel using average pooling, yielding planar features with dimensions $H \times W \times d$, where d represents the feature dimension.

Volumetric encoder While the planar feature mapping permits a higher spatial resolution (e.g., 128^2 pixels in their experiments), it is limited to two dimensions. To address this limitation, the researchers developed a volumetric feature mapping that better preserves 3D information, at smaller resolutions (e.g., 32^3 voxels in their experiments). Similar to the planar encoder, they applied average pooling to the volumetric feature mapping, where features corresponding to each voxel cell are aggregated, resulting in a feature volume with dimensions $H \times W \times D \times d$, where H, W, and D represent the height, width, and depth of the volume, respectively, and d signifies the feature dimension.

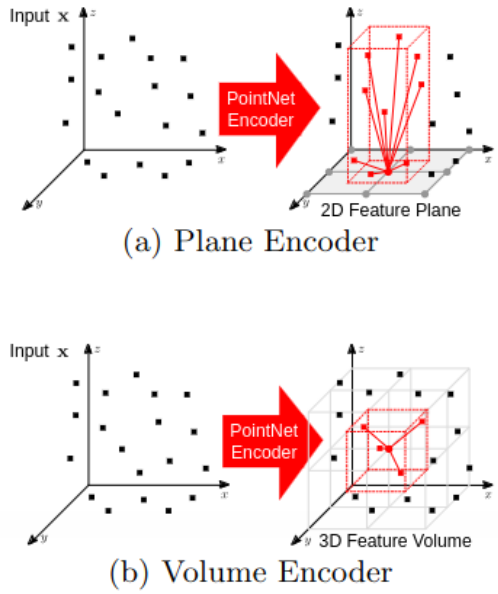


Figure 2.6: Planar and volumetric encoder [1]

2.2.2 Decoder

The decoder of this network consists of a convolutional decoder that processes the resulting feature map using 2D/3D U-Nets [16]. These U-Nets are employed to aggregate local and global information. For a given query point $\mathbf{p} \in \mathbb{R}^3$ and input data $\mathbf{x}$, the feature vector $\psi(\mathbf{x}, \mathbf{p})$ is obtained via bilinear interpolation (see Figure 2.7. 2c and 2d) or trilinear interpolation (see Figure 2.7. 2e). With the feature vector $\psi(\mathbf{x}, \mathbf{p})$ at location $\mathbf{p}$ determined, the network predicts the occupancy probability using a fully-connected network $f_\theta(\mathbf{p}, \psi(\mathbf{x}, \mathbf{p}))$.

More precisely, the encoder outputs a planar or volumetric feature map containing local features. To obtain global information, a convolutional hourglass network, U-Net, is employed to process the feature map. The U-Net architecture comprises a contracting path and an expansive path. The contracting path follows a typical convolutional network architecture with ReLU activation and max pooling for downsampling. On the other hand, the expansive path involves

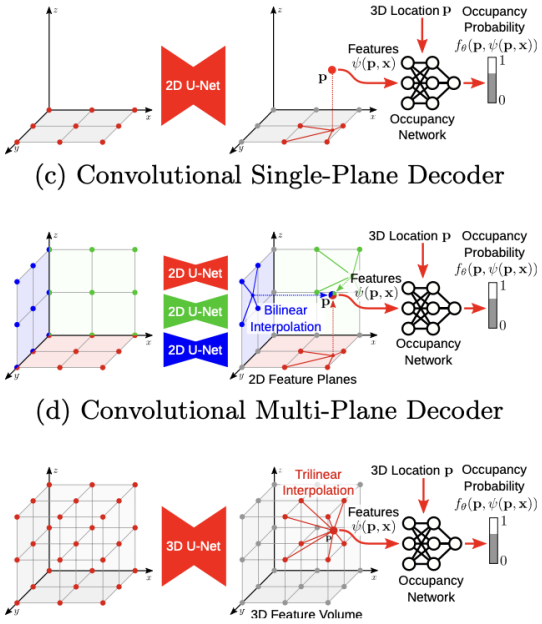


Figure 2.7: Convolutional decoder [1]

upsampling the feature map, followed by a convolution operation that reduces the number of feature channels by half. This upsampled feature map is then concatenated with the correspondingly cropped feature map from the contracting path, and further convolutions are applied with ReLU activation. Finally, at the last layer of the U-Net, a convolution is utilised to map each feature vector to its corresponding occupancy probability.

2.3 Variational autoencoder

A Variational Autoencoder (VAE) [3] represents a generative model that builds upon the concept of an autoencoder. Rather than encoding a single fixed latent code for each sample, VAEs estimate the underlying distribution of the entire dataset. This ability allows them to generate new samples that closely resemble the original data distribution. VAEs are trained

using a combination of two fundamental components: a reconstruction loss and a regularisation term.

The reconstruction loss ensures that the generated samples from the VAE resemble the original data points. And the regularisation term, also known as the KL divergence, plays a crucial role in shaping the latent space of the VAE. It encourages the latent space to follow a specific distribution, often a Gaussian distribution.

By balancing the reconstruction loss and the regularisation term, VAEs learn to create meaningful latent representations of the input data, which can be sampled to generate novel data points that adhere to the original data distribution. This capability makes VAEs valuable tools for various applications, such as data generation, image synthesis, and anomaly detection.

Chapter 3

Implementation

3.1 Architecture

The architecture of our generative neural network is primarily based on the convolutional occupancy network [1], but with a extension into a variational autoencoder (VAE). While the autoencoder's reconstruction result remains deterministic, our VAE introduces an element of variability, allowing to generate slightly different 3D synthetic data that exhibits small differences from the input data.

By using the VAE, we can produce synthetic data samples that capture the underlying data distribution. These synthetic samples, while resembling the real data, possess slight variations that make them valuable for training of fine-tuning other networks. This augmentation technique proves particularly beneficial when dealing with relatively small datasets, as it helps increase the dataset size and diversity.

3.2 Method

The idea of learning continuous latent variables following a Gaussian distribution first appeared in Diederik P.Kingma and Max Wellin's publication [3].

For the case of an independent and identically distributed (i.i.d.) dataset and continuous latent variables per data point, they proposed the Auto-Encoding Variational Bayes algorithm. In this algorithm, they make inference and learning especially efficient by using the Stochastic Gradient Variational Bayes estimator to optimise a recognition model that allows to perform efficient approximate posterior inference using simple ancestral sampling, without the need of expensive iterative inference schemes (such as Markov chain Monte Carlo).

3.2.1 Reparameterization trick

The dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,...N}$ consists of N i.i.d. samples. The data is assumed to be generated by an unobserved continuous random variable $\mathbf{z}$. Each individual value $\mathbf{z}_i$ is drawn from a prior distribution $p_\theta(\mathbf{z})$, and each corresponding value $\mathbf{x}_i$ is drawn from a conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$.

Since computing the marginal likelihood

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} \tag{3.1}$$

is intractable, and obtaining the true posterior density

$$p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) / p_\theta(\mathbf{x}) \tag{3.2}$$

is also intractable, a sampling-based approach is employed. Given a dataset of significant

size, sampling methods can be utilised to approximate the intractable distributions.

To address the intractability of the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, an approximation is introduced through a recognition model $q_\phi(\mathbf{z}|\mathbf{x})$. This recognition model is assumed to have an approximate Gaussian form with an approximately diagonal covariance structure:

$$\log q_\phi(\mathbf{z}|\mathbf{x}_i) = \log \mathcal{N}(\mathbf{z}|\mu_i, \sigma_i^2 \mathbf{I}) \tag{3.3}$$

where the mean μ_i and standard deviation σ_i are outputs of the MLP after the feature plane / grid generated by encoder in our case.

To tackle the sampling issue, one approach involves generating samples from $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. For example, in the univariante Gaussian case: let $z \sim p(z|x) = \mathcal{N}(\mu, \sigma^2)$, a valid reparameterization is $z = \mu + \sigma\epsilon$, where ϵ is an auxiliary noise variable $\epsilon \sim \mathcal{N}(0, 1)$.

3.2.2 Training loss

The training loss consists two terms, the reconstruction loss and the regularisation term. The reconstruction loss penalises the discrepancy between the original input and the reconstructed output. It ensures that the VAE learns to generate outputs that resemble the original data. The regularisation term in the VAE loss function is the Kullback-Leibler (KL) divergence between the latent space distribution $q_\phi(\mathbf{x})$ and the prior distribution $p_\theta(\mathbf{x})$. It encourages the VAE to learn a smooth and continuous latent space.

The expectation in the loss function is taken with respect to the encoder's distribution over the latent space representations and under the form:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)}[\log p_\theta(\mathbf{x}_i|\mathbf{z})]. \tag{3.4}$$

The KL-divergence $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i)||p_\theta(\mathbf{z}))$ can be integrated analytically and the full loss function can be written as:

$$\mathcal{L}(\theta,\phi|\mathbf{x}_i) \simeq \underbrace{\frac{1}{2}\sum_{j=1}^{N}\left(1+\log((\sigma_j^{(i)})^2)-(\mu_j^{(i)})^2-(\sigma_j^{(i)})^2\right)}_{D_{KL}\left(\mu^{(i)},\sigma^{(i)}\right)} + \underbrace{\frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)})}_{\mathcal{L}_{\text{reconstruction}}\left(\mathbf{x}^{(i)},\mathbf{z}^{(i)}\right)} \tag{3.5}$$

where $\mathbf{z}^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim \mathcal{N}(0,\mathbf{I})$.

3.3 Dataset

The initial phase of our research involves the validation of our model's 3D synthetic data generation capabilities using the ShapeNet dataset [4]. Following this validation, we proceed to train our model on the IntrA dataset [5], an open-access 3D intracranial aneurysm dataset, despite its relatively limited size.

The overarching goal of our research is to augment this dataset by generating additional data that closely adheres to the characteristics of the original samples. This enlarged dataset is intended for the purpose of fine-tuning deep learning models within the medical domain, thereby facilitating enhanced performance and broader applicability in various medical applications.

3.3.1 ShapeNet

ShapeNet stands as a prominent annotated and large-scale dataset, extensively employed in research across computer graphics, computer vision, robotics, and related fields. For our

experimental evaluation, we specifically assess our model's performance on the plane class within the ShapeNet dataset, examples shown in Figure 3.1.

This choice is deliberate, as the wing structure of a plane bears similarity to the tree structure found in the vessels of the IntrA dataset. By employing the plane class, we can effectively evaluate the model's capability to learn and represent complex tree-like structures, which are significant in the context of the IntrA dataset. This approach allows us to thoroughly examine the model's proficiency in handling vascular structures.

The ShapeNet dataset presents a relatively facile learning scenario due to specific characteristics within the plane class. This class comprises 4046 data samples, all of which are consistently aligned to a standardised orientation. Moreover, the samples undergo normalisation, constraining them within a unit sphere and consequently exhibiting negligible variance among themselves. These properties contribute to a simplified learning environment.

Figure 3.1: Examples of ShapeNet plane samples

3.3.2 IntrA

Yange et al. [5] introduced IntrA in 2020 as a 3D intracranial aneurysm dataset specifically created for medical-related classification and segmentation tasks. Its primary application includes diagnosing intracranial aneurysms and extracting the aneurysm neck for clipping operations in medicine. Additionally, the dataset finds utility in other deep learning tasks such as normal estimation and surface reconstruction.

The dataset comprises 103 3D models of entire brain vessels reconstructed from 2D MRA images of patients. From these complete models, the authors generated 1909 blood vessel segments, encompassing 1694 healthy vessel segments and 215 aneurysm segments, suitable for diagnostic purposes.

In comparison to ShapeNet, IntrA exhibits a substantially smaller sample size and presents more variance in mesh structure and orientation in the provided examples (Figure 3.2). The experiments primarily focus on healthy vessel generation due to the main assumption of VAE,

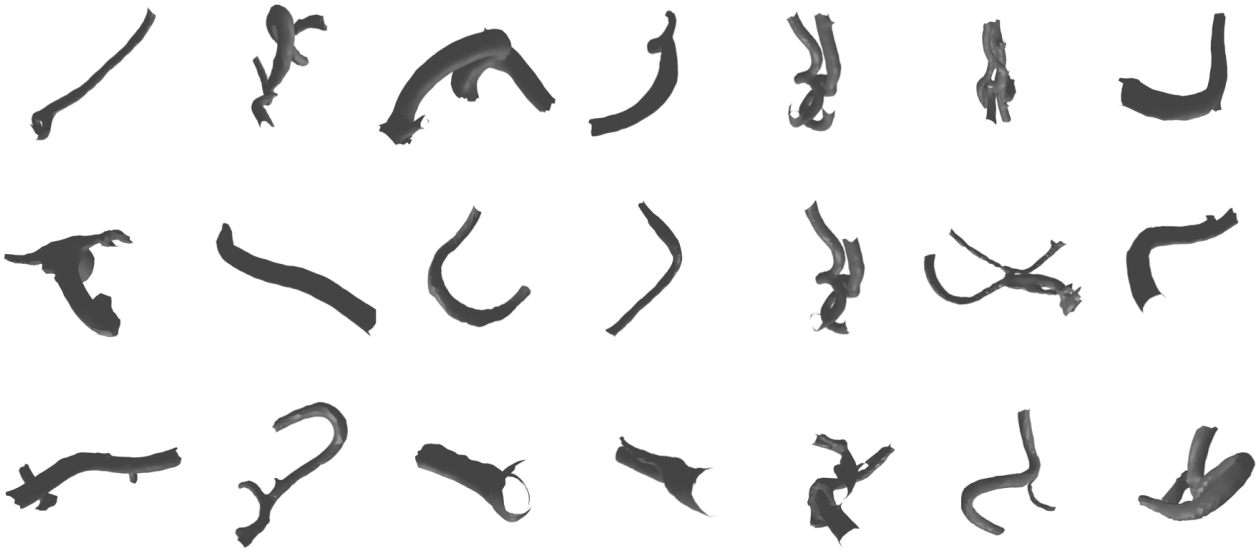


Figure 3.2: Examples of IntrA samples

which relies on a sufficiently large dataset to approximate a Gaussian distribution. Notably, the healthy vessels offer a larger size of data samples, enhancing the viability of VAE-based modeling approaches for this dataset.

3.4 Preprocessing

A watertight 3D mesh is an essential prerequisite for training the network. A watertight mesh refers to a mesh that fully represents a closed surface without any holes and possesses a clearly defined inside. In contrast, a non-watertight mesh may contain holes or gaps in its surface, rendering it open and not entirely enclosed.

The objective of the network is to train a continuous neural representation of the mesh's surface. The input to our network consists of point clouds containing 3D coordinates and occupancy values. The occupancy value of 1 represents points inside the mesh, while a value of 0 denotes points outside the mesh. To create the input for training, we extract 3000 points evenly distributed within a bounding box, along with their respective occupancy values.

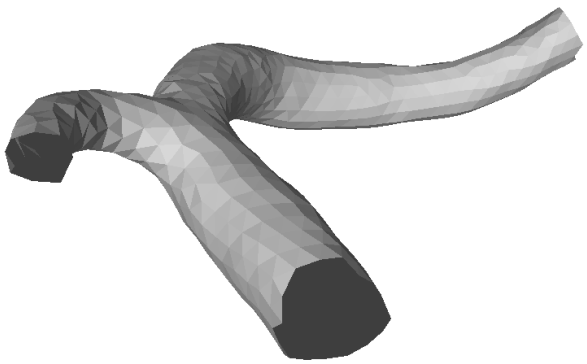


Figure 3.3: Non-watertight raw data

Indeed, the raw data from the IntrA dataset comprises non-watertight meshes, as depicted in Figure 3.3. Notably, the ends of the vessels are not fully closed. Consequently, a preprocessing step is necessary to make these meshes watertight before extracting point clouds from them.

Our preprocessing pipeline is derived from David Stutz and Andreas Geiger's code [17]. Their processing pipeline enables the generation of watertight and simplified meshes from arbitrary triangular meshes in the `.off` format.

Our adaptation follows the steps below:

1. Mesh Format Conversion: Initially, we convert the mesh format from `.obj` to `.off`.

2. Scaling: The meshes are then scaled to fit within a uniform bounding box of dimensions $[-0.5, 0.5]^3$, with padding if needed.

3. Depth Map Rendering: Next, we render depth maps from various viewpoints and use them to perform Truncated Signed Distance Function (TSDF) fusion [18].

4. Point Cloud Generation: We further process the generated watertight mesh to obtain point clouds representing the mesh surface.

Figure 3.4 displays the outcomes of the generated watertight vessels. Notably, in the more detailed image shown in Figure 3.5, we can observe that the holes at the end of the vessels are now closed. This conversion results in vessels with a hollow structure, potentially posing challenges during the training process.

After the preprocessing, we are left with only 1134 samples. The reduction in sample count is a result of the algorithm's limitation in rendering sufficient depth maps for certain samples, primarily due to the complex topology of those samples.

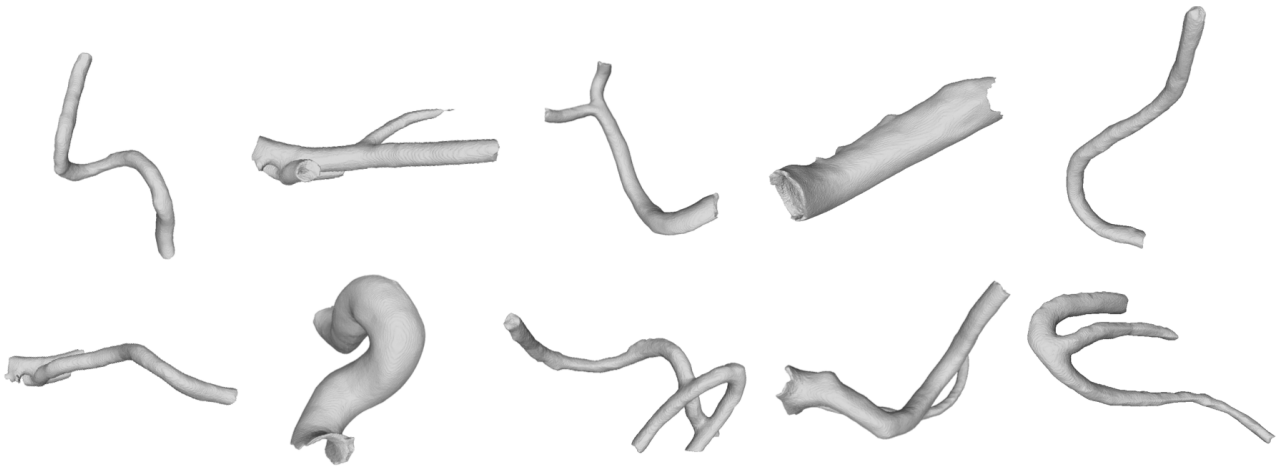


Figure 3.4: Watertight meshes after preprocessing

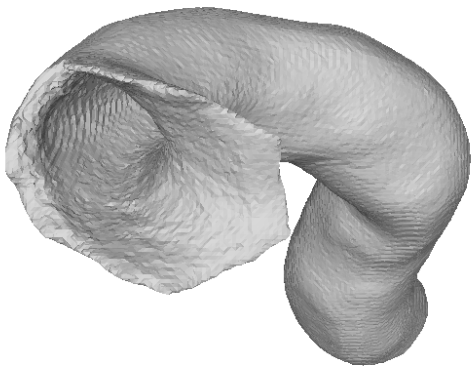


Figure 3.5: Hollow struture of the vessel mesh

Chapter 4

Experiments

4.1 Data augmentation

As shown in Figure 3.4, the IntrA dataset exhibits considerable variation in the structure of vessels, including geometry, diameter, alignment, and the presence of bifurcations. Moreover, the sample size is approximately 1000, which is relatively small compared to the existing variation in the data.

Given the challenges associated with learning directly from the raw data due to its limited size and diversity, we aim to perform data augmentation on the 3D vessel meshes. The following methods are usually used in data augmentation:

- Rotation: Apply random rotations around different axes to introduce variations in the orientation of vessels.
- Translation: Perform random translations in all three dimensions to alter the positions of vessels.

- Scaling: Apply random scaling factors to adjust the size of vessels, mimicking different diameters.
- Mirroring: Introduce mirror reflections to replicate vessels with similar structures on the opposite side.
- Deformation: Apply controlled deformations to vessels to capture variations in their geometry and shape.
- Noise Addition: Add random noise to the 3D coordinates to simulate sensor noise or uncertainties.
- Cropping: Extract random regions from the vessels to focus on specific regions of interest.

During the preprocessing, as the samples are already normalized within a unit boundary box, the network becomes invariant to scaling and translation. Thus, scaling and translation are unnecessary in our case.

Our baseline network, the Convolutional Occupancy Network [1], is translational equivariant but not translation-invariant. Translational equivariance ensures that the model's output changes consistently with translations of the input. However, data augmentation with rotation introduces additional variations, enhancing the model's ability to generalise and recognise objects from different viewpoints.

Given that the dataset is medical in nature, caution must be exercised when generating augmented data to ensure its meaningfulness and similarity to the original data. As a result, the decision is made to employ only rotation, mirroring, and limited surface noise as data augmentation techniques. These methods strike a balance between enriching the dataset and

maintaining the dataset's fidelity for use in medical applications.

4.1.1 Random rotation

The initial data augmentation technique employed is random rotation. For every individual sample in the dataset, we generate five random 3D rotation matrices, which we subsequently apply as transformations to the original mesh. From these five generated meshes, we randomly select two and apply mirroring to them.

Random rotation introduces a broader range of viewpoints to the dataset, which can enhance the model's ability to generalise across different orientations. Nevertheless, the utilisation of only five rotations may not ensure a uniformly distributed coverage of the entire rotation space, potentially resulting in underrepresented regions and introducing biases in the augmented data. Furthermore, the varying rotations during training iterations introduce variability in the evaluation process, making it challenging to consistently assess the model's performance.

4.1.2 Uniformly sampled 3D rotation

In light of the limitations observed with randomly rotated datasets, we aim to address the issue of uneven coverage in the rotation space. To achieve a more balanced representation across all regions of the rotation space, even with a small number of rotations, we have adopted a new method - uniformly sampled 3D rotation.

By employing uniformly sampled 3D rotation, we ensure that the rotations are evenly distributed across the entire rotation space. This approach guarantees that each region of the space is represented in a similar manner, minimising the potential introduction of biases during data

augmentation.

Sampling 2D rotations uniformly is straightforward, as it involves rotating by an angle following the uniform distribution $\theta \sim U(0, 2\pi)$. However, extending this idea to 3D rotations by sampling each of the three Euler angles (ϕ, θ, ψ) from the same uniform distribution $\phi, \theta, \psi \sim U(0, 2\pi)$ results in more probability density being concentrated towards transformations clustered near the poles.

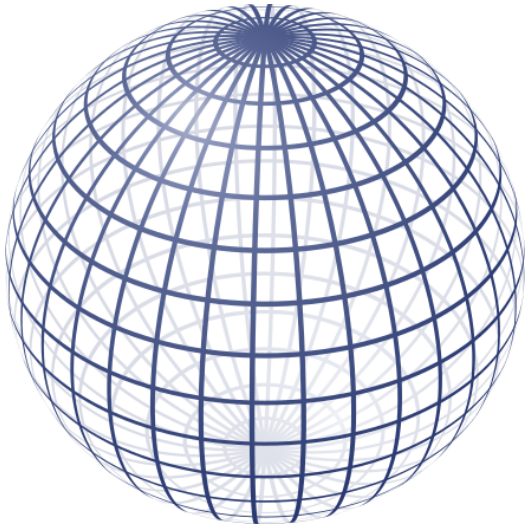


Figure 4.1: Sampling Euler angles uniformly does not yield an even distribution across the sphere.

James Avro proposed a method for generating uniform random 3D rotation matrices in 1992 [19]. The idea involves two steps:

1. Perform a random rotation about the vertical axis.

2. Rotate the north pole to a random position.

Here, we only present a walk-through of the main steps involved in generating uniform random 3D rotation matrices. More detailed explanations can be found in James Avro's original

publication [19].

For the first step, given a number following the uniform distribution $x_1 \sim U(0,1)$, we can obtain a rotation matrix R:

$$R = \begin{bmatrix} \cos(2\pi x_1) & \sin(2\pi x_1) & 0 \\ -\sin(2\pi x_1) & \cos(2\pi x_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Here, we assume the z-axis is the vertical vector, the *north pole*, and thus the vector is represented as $z = (0, 0, 1)$.

For the second step, we observe that we can transform the point z to any point p on the sphere using a reflection operation through the plane orthogonal to the line $\overline{zp}$ and aligning with the origin. This reflection operation is achieved using the *Householder* matrix:

$$H = I - 2vv^T \tag{4.2}$$

To transform the reflection into a rotation, we need to apply one more reflection - a reflection through the origin. Then the final rotation matrix an be written as:

$$M = -HR \tag{4.3}$$

The matrix M will be uniformly distributed if H takes the north pole to every point on the sphere with equal probability density. This will be true if the image of z under the random reflection is such that both its azimuthal angle and the cosine of its polar angle are uniformly distributed. The matrix H in Eq (4.2) will satisfy these requirements if we let:

$$v = \begin{bmatrix} \cos(2\pi x_2)\sqrt{x_3} \\ \sin(2\pi x_2)\sqrt{x_3} \\ \sqrt{1-x_3} \end{bmatrix} \tag{4.4}$$

where x_2 and x_3 are two independent variables following a uniform distribution $U \sim (0, 1)$.

The code implementation can be found in Appendix A. In Figure 4.2, we illustrate the rotation matrices by applying them to a unit vector $(0, 0, 1)$. For a small number of samples (5

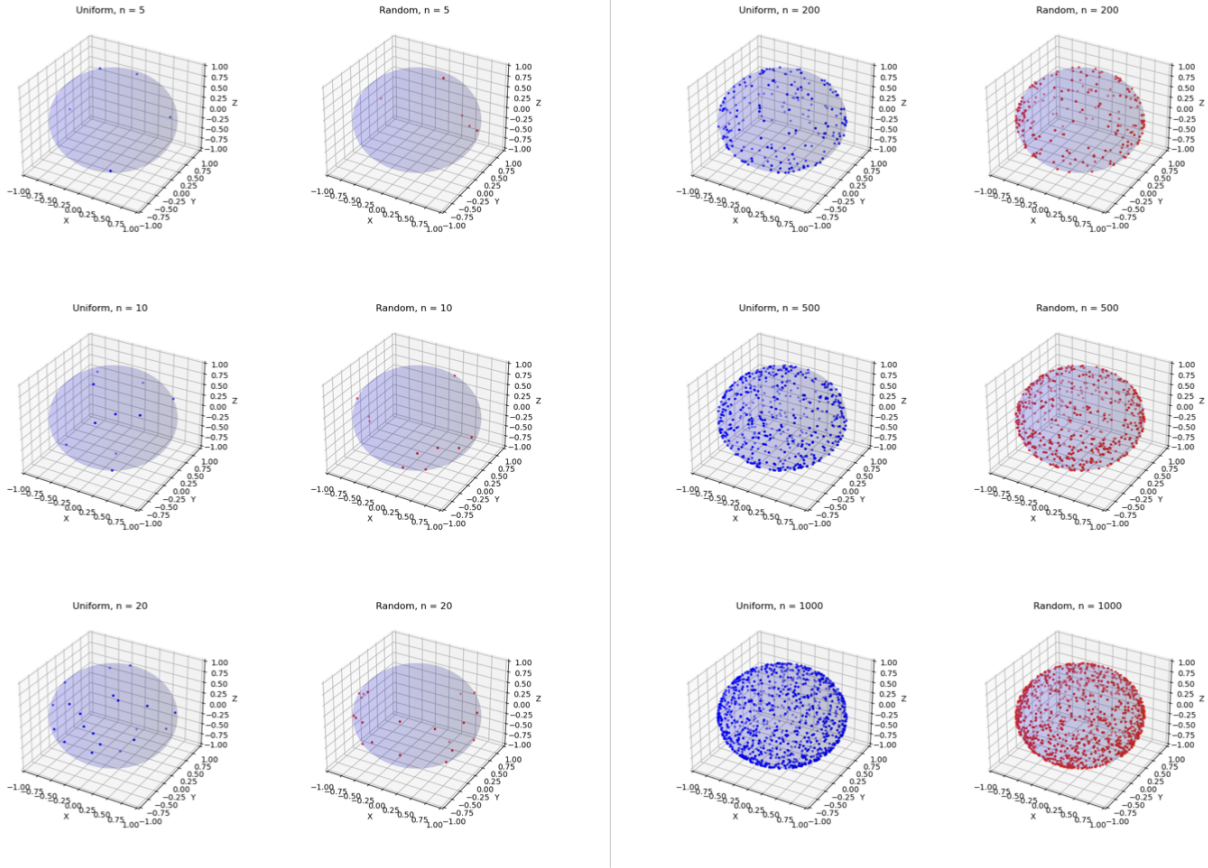


Figure 4.2: A comparison of uniformly distributed rotation and random rotation. Uniformly rotated matrices are presented in blue on the left, and random rotated matrices are presented in red on the right. We sampled for $n = 5, 10, 20, 200, 500$, and 1000.

to 20), the uniformly distributed rotations appear to exhibit a more homogeneous distribution compared to random rotations. However, as the sample size increases (200 to 1000), both distributions tend to behave more similarly.

Due to constraints in device capacity, rotating one mesh a significant number of times is impractical. As a result, we choose uniformly distributed rotations to ensure a more evenly distributed representation of viewpoints. Consequently, we decide to generate one mesh for 9 times, effectively augmenting the sample size by a factor of 10 after the rotations.

4.1.3 Gaussian noise

A general method to introduce random variations without distorting the overall structure of the mesh significantly is by adding Gaussian noise. This involves perturbing the vertices of the mesh with random values drawn from a Gaussian distribution. It is crucial to ensure that the noisy mesh remains within a valid range of values. This means verifying that the perturbed vertices do not intersect or go outside the surface of the original mesh.

We have set the standard deviation to 0.005 while scaling the mesh within a bounding box with a length of 1. Figure 4.4 illustrates an example of the mesh after applying Gaussian noise.

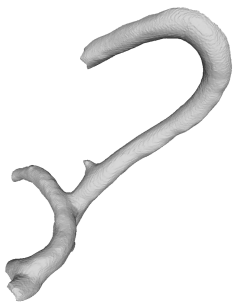


Figure 4.3: Original mesh

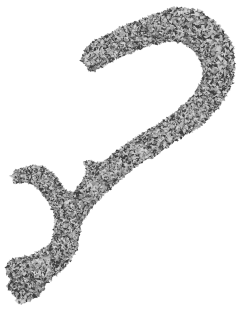


Figure 4.4: After applying Gaussian noise

4.2 Variance reduction

In contrast to the approach discussed in Section 4.1, where data augmentation techniques aim to increase dataset diversity, an alternative idea focuses on decreasing the variance within the dataset. The primary objective here is to learn the general shape of the input data and generate samples that resemble the original data. In this context, invariance by rotation is not a primary concern. Therefore, an alternative strategy is proposed to reduce the variance in the dataset.

Reducing variance in the dataset can lead to a more focused and easier-to-learn representation of the data. By homogenising the dataset, the training process can become smoother, potentially resulting in improved model performance. By exploring this approach, we aim to achieve a more specialised representation of the data's shape characteristics, enabling the generation of synthetic samples that closely match the input data.

4.2.1 Shape-Based Data Subsetting

After carefully observing the shape characteristics of the samples, we have partitioned the dataset into two distinct subsets. The first subset consists of vessels without any bifurcation, referred to as the **Main Vessel** subset. These vessels are characterised by their large and thick structure with limited branching. The second subset is designated as the **Capillary** subset, comprising vessels with a tree-like structure formed by interlacing with other capillaries. Capillaries in this subset are relatively thinner compared to the main vessels.

Fig. 4.5 showcases the two subsets, with the main vessel subset shown on the left and the capillary subset on the right. The main vessel subset contains 808 samples, while the capillary

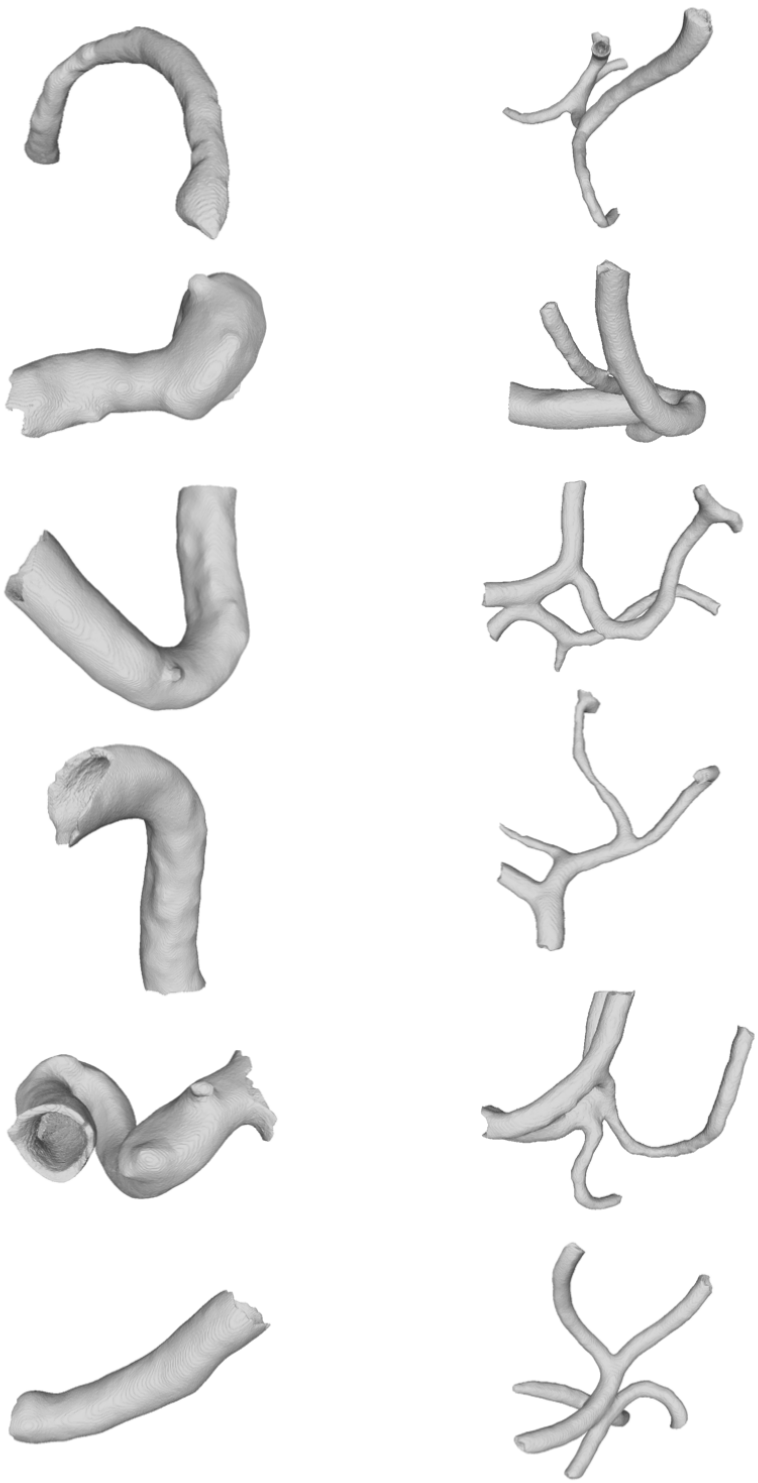


Figure 4.5: Example of main vessel subset (left) and capillary subset(right)

subset comprises 888 samples.

This separation allows for a targeted reduction of variance within each subset, ensuring that the data representation becomes more focused and coherent. By partitioning the dataset strategically, the model can gain deeper insights into the underlying structure of the data, resulting in better generalisation and generation capabilities.

4.2.2 Orientation alignment

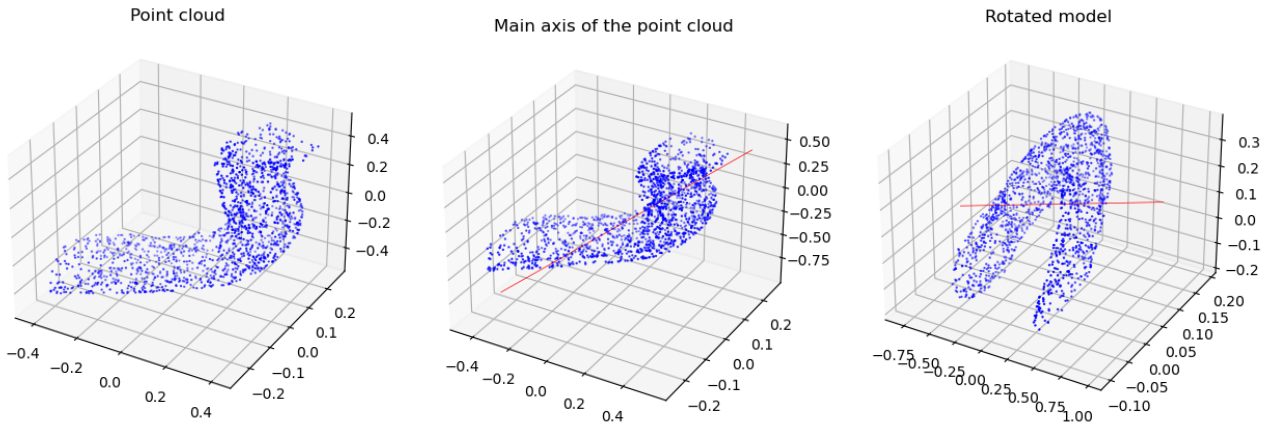
As shown in Figure 4.5, the samples within each subset exhibit significant variations in orientation. To achieve our objective of decreasing variance within the subsets, it is essential to align their orientations uniformly. However, manually rotating over a thousand samples to a consistent direction would be an arduous task. Hence, we have implemented an autonomous method to align the samples as follows:

1. Given a vessel mesh, we obtain its vertices' coordinates and extract a sparse point cloud, representing the general shape of the mesh, comprising 1% of the total vertices.

2. We perform linear regression on the point cloud to fit a line, which can be considered as the main axis of the mesh.

3. Next, we rotate the main axis to align it with the x-axis, effectively aligning the mesh's orientation uniformly.

The code implementation for the autonomous alignment process can be found in Appendix B. To provide a visual representation of the alignment process, we present a process graph in Figure 4.6. This graph illustrates the step-by-step procedure involved in autonomously aligning the vessel samples to a consistent orientation along the x-axis.



1. Extract a sparse point cloud 2. Fit a main axis 3. Rotate to x-axis

Figure 4.6: Alignment process

Figure 4.7 presents samples from both subsets after the alignment rotation process. As shown in the figure, the **main vessel** subset exhibits significantly reduced variation in its orientation. The samples in the **capillary** subset also appear to be more ordered compared to their initial orientation. However, owing to the complexity in their structure, the variance within the **capillary** subset remains relatively high.

To further analyse and compare the performance of the network, we will train the subsets separately and evaluate their respective performances. This approach will help us understand the effectiveness of the alignment process in reducing variance and improving the model's performance on different vessel types.

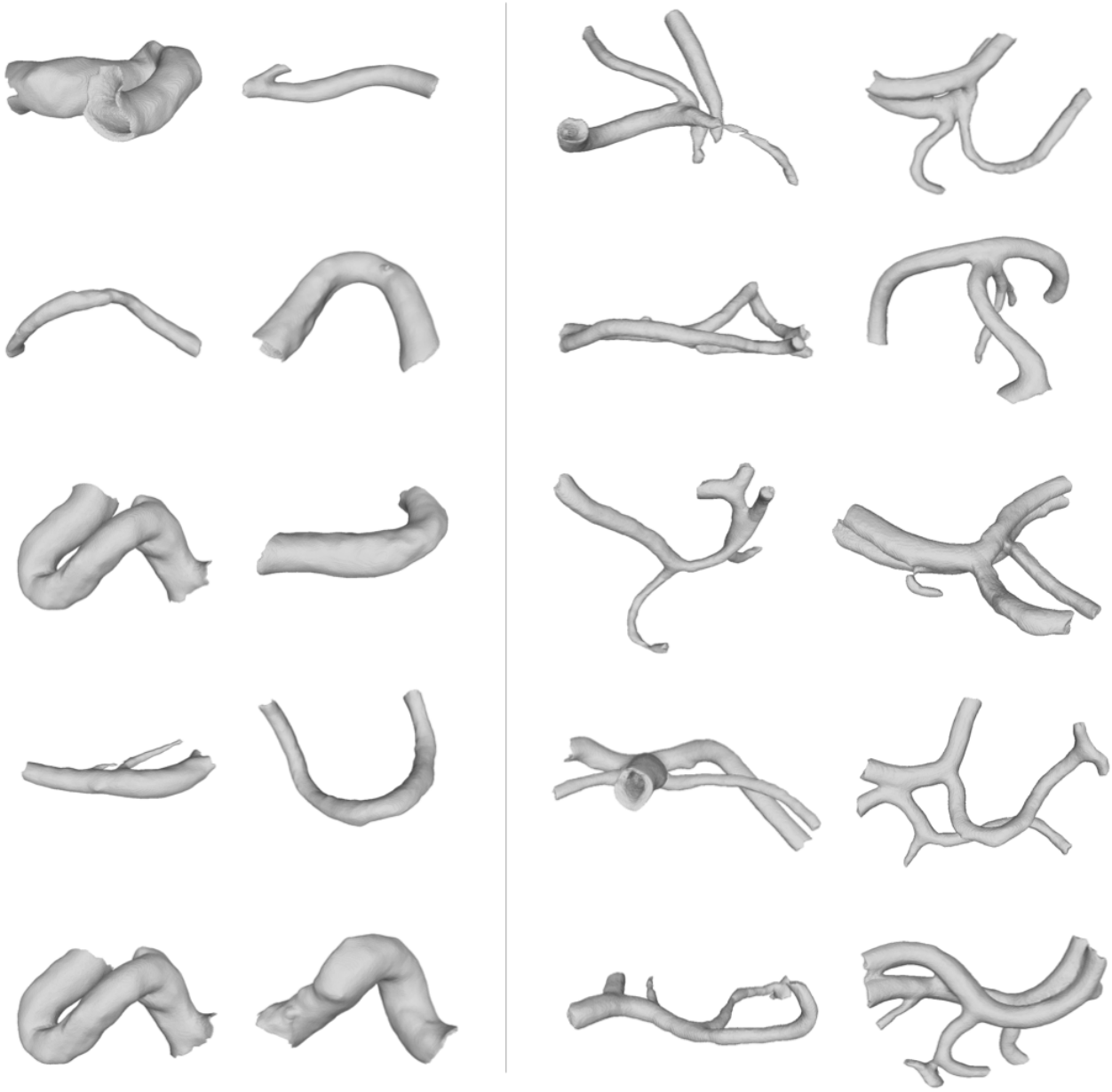


Figure 4.7: Example of aligned **main vessel** and **capillary**

4.3 Network architecture

The Convolutional Occupancy Network [1] uses two types of encoders, the plane decoder and volume decoder. For our research, we adopt the Convolutional Occupancy Network as our baseline model and compare its performance with our modified variational autoencoder.

4.3.1 Metrics

In the baseline experiments, the authors employ three types of encoders: a one-plane-encoder, a three-plane-encoder, and a volume encoder. The network's performance was evaluated using the following metrics for 3D reconstruction tasks at both the object and scene level:

- **Intersection over Union (IoU):** IoU measures the overlap between the reconstructed 3D mesh and the ground truth mesh. It calculates the ratio of the volume of intersection between the two meshes to the volume of their union.

- **Chamfer-L_1 distance:** The Chamfer-L_1 distance is a measure of dissimilarity between two sets of points in a metric space. It is commonly used in shape reconstruction tasks. The Chamfer-L_1 distance calculates the sum of the L_1 distances between each point in one cloud and its nearest neighbour in the other cloud.

  The Chamfer $L1$ distance between point cloud S_1 and S_2 is defined as:

$$\text{Chamfer-}L_1(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_1 + \frac{1}{|S_2|} \sum_{y \in S_2} \min_{x \in S_1} ||x - y||_1 \tag{4.5}$$

- **Normal Consistency:** The Normal Consistency metric is first introduced in Occupancy network [20]. It assesses how well the 3D reconstruction captures higher-order information. It calculates the mean absolute dot product of the normals in one mesh and the normals at their corresponding nearest neighbours in the other mesh.

- **F-score:** The F-score is a measure of the model's precision and recall. It provides a balance between the accuracy and completeness of the generated 3D shape. [21]

We conducted a rerun of the baseline Convolutional Occupancy Network and obtained the following metric performance results:

	IoU	Chamfer-L_1	Normal C.	F-score
1-Plane (64^2)	0.833	0.059	0.914	0.887
3-Plane (3×64^2)	**0.849**	**0.033**	0.931	0.987
Volume (32^3)	0.848	0.048	**0.932**	**0.989**

Table 4.1: 3D Reconstruction result on ShapeNet Plane Dataset [4]

4.3.2 Encoder selection

From the results presented in Table 4.1, we observe that the 3-plane encoder and volume encoder exhibit similar performance in the 3D reconstruction task. As a result, we have decided to implement the variational autoencoder with both the plane and volume encoders and will select the more relevant encoder based on their respective performance.

The training process was conducted on a Nvidia GTX 1080 GPU with 8 GB memory size. However, the addition of variational estimation for the latent code introduced an increased memory requirement compared to the original Convolutional Occupancy Network [1] implementation, where the training was performed on a GPU with similar memory.

In the original network, the latent code dimension was set to 32, and our experiments revealed that reducing this dimension led to a degradation in network performance. Therefore, we retained the same dimension for our model. As a result, the maximum plane resolution in our model is reduced to 32, and the maximum grid resolution is 8.

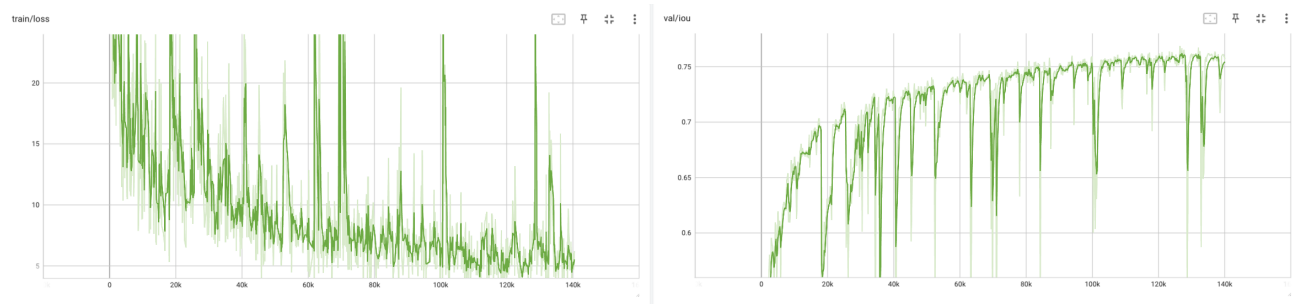
Our variational autoencoder was tested on the ShapeNet [4] plane dataset. The 3-plane encoder achieved an IoU of 0.74, while the volume encoder yielded an IoU of 0.34. This highlights the superior performance of the 3-plane encoder in the 3D reconstruction task on this particular dataset. Consequently, we selected the 3-plane encoder for further evaluation.

4.3.3 Training Challenges and Issues

We present the training and validation curves of the VAE on the ShapeNet [4] plane dataset, shown in Figure 4.8. Notably, we observed frequent instances of sudden increases in the training loss, which were accompanied by abrupt drops in the validation IoU metric. A plausible explanation for this behaviour could be the introduction of noise in the reparametrization trick.

Furthermore, it's important to note that training the VAE directly on the IntrA dataset [5] resulted in significantly lower performance. Similar to the ShapeNet dataset, the training process exhibited sudden spikes, and the IoU metric plateaued at 0.27 without improvement.

In this sections, we delve into our strategies for enhancing the training process's stability and the network's overall performance. We will explore adjustments to our loss and noise



Training loss IoU validation

Figure 4.8: Training and validation curves of the VAE on plane dataset

incorporation methods, as well as potential refinements to the network's parameters, in pursuit of smoother training dynamics and improved outcomes.

4.3.4 Transfer learning

To accelerate the convergence of the training process, the implementation of transfer learning can be employed for the VAE.

One approach involves initially training an autoencoder on the same dataset and then saving the parameters for both the encoder and decoder. Subsequently, during the VAE training, these saved parameters are loaded directly. This approach ensures that the model is already proficient in the reconstruction task, allowing the focus of training to be primarily on refining the latent space for sampling within the VAE framework.

Alternatively, transfer learning can be applied by modifying the reparametrization trick. Initially, the noise in the reparametrization is set to zero and the KL-divergence term is not added to the loss function until the network has reached a certain degree of convergence. After this point, both noise and the KL-divergence term are gradually reintroduced into the training process. This method allows the model to converge more effectively by initially emphasising reconstruction before incorporating the complexities of the VAE framework.

4.3.5 Warm-up technique

The warm-up technique refers to introducing the KL-divergence term gradually into the loss function. Instead of employing Eq (3.5), our modified loss function is given by:

$$\mathcal{L}(\theta, \phi|\mathbf{x}_i) = \alpha \cdot D_{KL}\big(\mu^{(i)}, \sigma^{(i)}\big) + \mathcal{L}_{\text{reconstruction}}\big(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}\big) \tag{4.6}$$

with α being a parameter that linearly scales from 0 to 1 during the initial 100 epochs of training. This approach gradually shifts the emphasis of learning from reconstruction to regularisation, enabling the model to initially focus on learning accurate reconstructions before enforcing a Gaussian-distributed latent space.

Chapter 5

Results

5.1 Parameter selection

Addressing the challenges highlighted in Section 4.3.3, we initially tackled these issues by adjusting the parameters of our network.

5.1.1 Network depth

Given the relatively subpar performance of the network on the smaller IntrA dataset compared to the ShapeNet dataset, a plausible hypothesis is that the network's parameter count is overly large relative to the size of the IntrA dataset. To validate this hypothesis, an approach involves reducing the number of parameters within the network by decreasing the depth of both the encoder and decoder architectures while ensuring their symmetry.

In this pursuit, the hidden dimension parameter was modified from 64 to 16. This alteration led to increased training stability; however, it did not yield a significant improvement in the IoU value. Despite this, the decision was made to retain the hidden dimension of 16, as it facilitates

a swifter and more stable training process. This adjustment aligns with the goal of addressing the parameter-to-data-size imbalance.

5.1.2 Latent code dimension

In the baseline architecture, both the encoder and decoder are implemented with convolutional layers, resulting in an encoder output depth of 32, which corresponds to the latent code's dimension. This can be interpreted as multiple layers of feature maps.

Notably, during our experiments, it became evident that reducing this number resulted in a decline in network performance. Our working hypothesis is that increasing the dimension of the latent code could enhance the network's ability to learn diverse and distinct features.

It's important to note that augmenting the dimension also translates to an increased computational requirement. However, our ability to empirically test this hypothesis was constrained by the computational capacity of the training GPU.

5.1.3 Variance amplitude

In the context of the reparametrization trick, a latent representation is sampled from a trained Gaussian distribution $\sim \mathcal{N}(0, 1)$. For the ShapeNet dataset, setting the standard deviation to 1 has proven effective in achieving satisfactory reconstruction outcomes.

However, when dealing with the IntrA dataset, a recurring observation during training is the frequent occurrence of sudden drops in the validation metric. One assumption is that the network grapples with fitting a Gaussian distribution owing to the dataset's high variance and limited sample size. As a result, the decoding sampling process experiences difficulty, leading

to distribution collapse and abrupt performance drops.

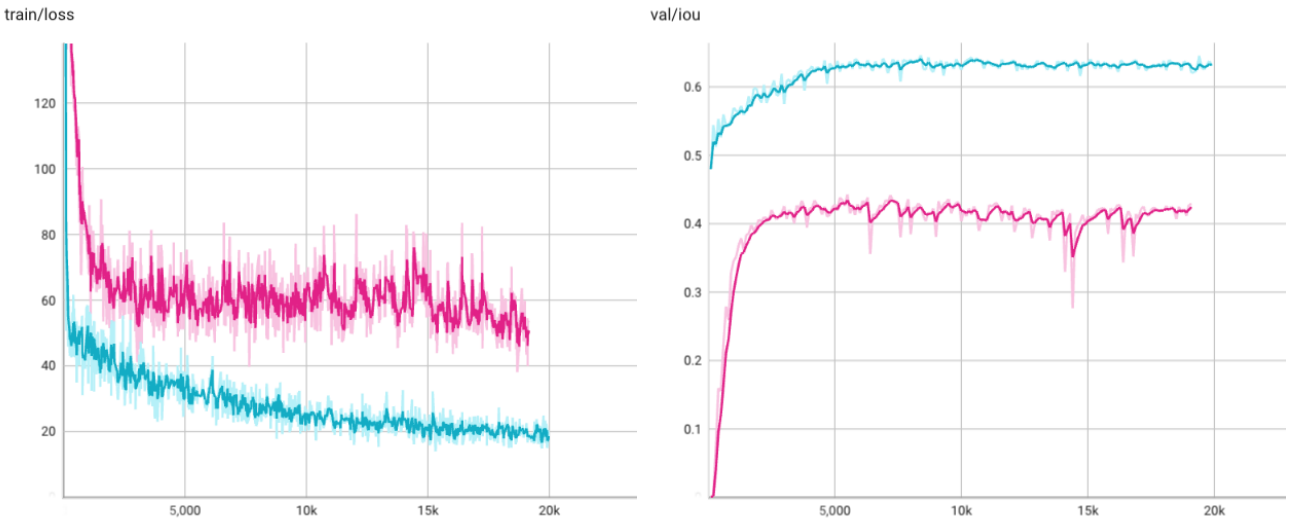
To substantiate this assumption, we conducted VAE training with varying standard deviation values, ranging from 10^{-6} to 1. The findings indicate that using smaller standard deviation values contributes to more stable training and higher IoU scores.

By constraining the standard deviation to a smaller value, the VAE transforms into an autoencoder. This promotes training stability while mitigating variance in the generated models. However, it's important to note that this approach might result in less diverse output due to the reduced variance in the latent space.

5.2 Training and validation curve

In Figure 5.1, we present the training loss and validation IoU for both the autoencoder baseline (in blue) and our VAE (in pink). We observe increased oscillations in the training and validation curves of our VAE. This phenomenon can be attributed to the introduction of randomness inherent in the VAE framework. In contrast, the baseline exhibits smoother curves due to its deterministic nature.

It's worth noting that the baseline outperforms our VAE in the reconstruction task, which aligns with our expectations. The autoencoder produces deterministic results, while the VAE introduces small random variations in its outputs. This inherent stochasticity in VAE outputs can result in a marginal decrease in reconstruction performance.



Training loss

IoU validation

Figure 5.1: Training and validation curves of the baseline (in blue) and VAE (in pink) on IntrA dataset

5.3 ShapeNet dataset

The VAE model employs an encoder with 3 planes of resolution 32, which is half the resolution of its baseline autoencoder from the Convolutional Occupancy Network. The comparison of reconstruction results between the baseline and the VAE is illustrated below:

	IoU	Chamfer-L_1	Normal C.	F-score
Autoencoder (3×64^2)	0.848	0.0338	0.931	0.987
VAE (3×32^2)	0.745	0.066	0.880	0.913

Table 5.1: 3D Reconstruction comparison on ShapeNet Plane Dataset

The lower IoU value in the VAE reconstruction is reasonable because the sampling from the distribution introduces some randomness to the reconstruction results. This randomness can lead to slight differences between the reconstructed mesh and the input.

We present the reconstruction results in Figure 5.2. The first column corresponds to the ground truth, the second column to the mesh reconstructed from the convolutional occupancy network, and the last three columns display the meshes generated by our VAE.

An observation is that our VAE model exhibits better performance on regular plane samples compared to less regular ones. For instance, when the plane possesses a more complex shape, such as the plane in the sixth row, the network struggles with reconstructing the intricate branches. Notably, the reconstructed mesh consistently converges towards the mean of the entire dataset, and typically there are very few differences from the input mesh.

5.4 IntrA dataset

As the IntrA dataset comprises a smaller sample size with more variances, the VAE model with the same structure did not perform as effectively as it did on the ShapeNet dataset. Various approaches were attempted, as outlined in Chapter 4.

Data augmentation involving random rotation and flipping led to an increase in the IoU score from 0.27 to 0.35. However, the utilisation of uniformly sampled 3D rotations did not yield better results than random rotation; both techniques led to the same IoU outcome.

Conversely, the approach of separating samples into two subsets (main vessel and capillary subsets) and aligning their main axes resulted in a decrease in the network's performance. This discrepancy could be attributed to the reduction in sample size due to subsetting, which may

have introduced a larger impact on the training dynamics.

Regarding the network architecture, we employed the warm-up technique to enhance the stability of the training process. By gradually introducing KL-divergence in the initial training epochs, we aimed to ensure a smoother convergence. Additionally, we identified that the original noise amplitude setting of 1 in the reparametrization trick was excessively large for the IntrA dataset. To prevent the decoder from collapsing due to this high noise value, we constrained the amplitude to 0.05. This value represents a compromise, as it allows the model to maintain a stable training process while also enabling the generation of slightly varied samples.

The table below presents the performance of the baseline and our VAE in 3D synthetic data generation task. The corresponding generation results are illustrated in Figure 5.3.

	IoU	Chamfer-L_1	Normal C.	F-score
Autoencoder (3×64^2)	0.643	0.011	0.854	0.792
VAE (3×32^2)	0.457	0.021	0.686	0.530

Table 5.2: 3D Reconstruction comparison on IntrA vessel Dataset

The generated samples reveal that the generation task performs more effectively on simpler and regularly structured vessels. The network appears to grasp the general overall structure but doesn't capture specific finer details. However, for vessels with complex tree-like structures, the network faces challenges in generating continuous and coherent vessels, as evidenced by the last two rows of the generated samples.

Notably, the baseline model employs a higher-resolution plane (64^2) compared to our VAE

model (32^2). Even with this higher resolution, the autoencoder struggles to adequately learn capillaries and bifurcation features, as demonstrated by its reconstruction results. In contrast, our VAE model adopts a lower-resolution encoder and decoder due to computational constraints. Additionally, the incorporation of Bayesian probability in the latent space introduces further complexities in the learning process.

Moreover, the encoder's design involves projecting features onto planes and generating a 3-plane feature map as the output, might not be optimally aligned with the characteristics of the dataset's hollow vessel structures. This aspect, combined with the use of lower-resolution planes, could contribute to the challenges encountered in generating accurate representations of complex vessel arrangements.

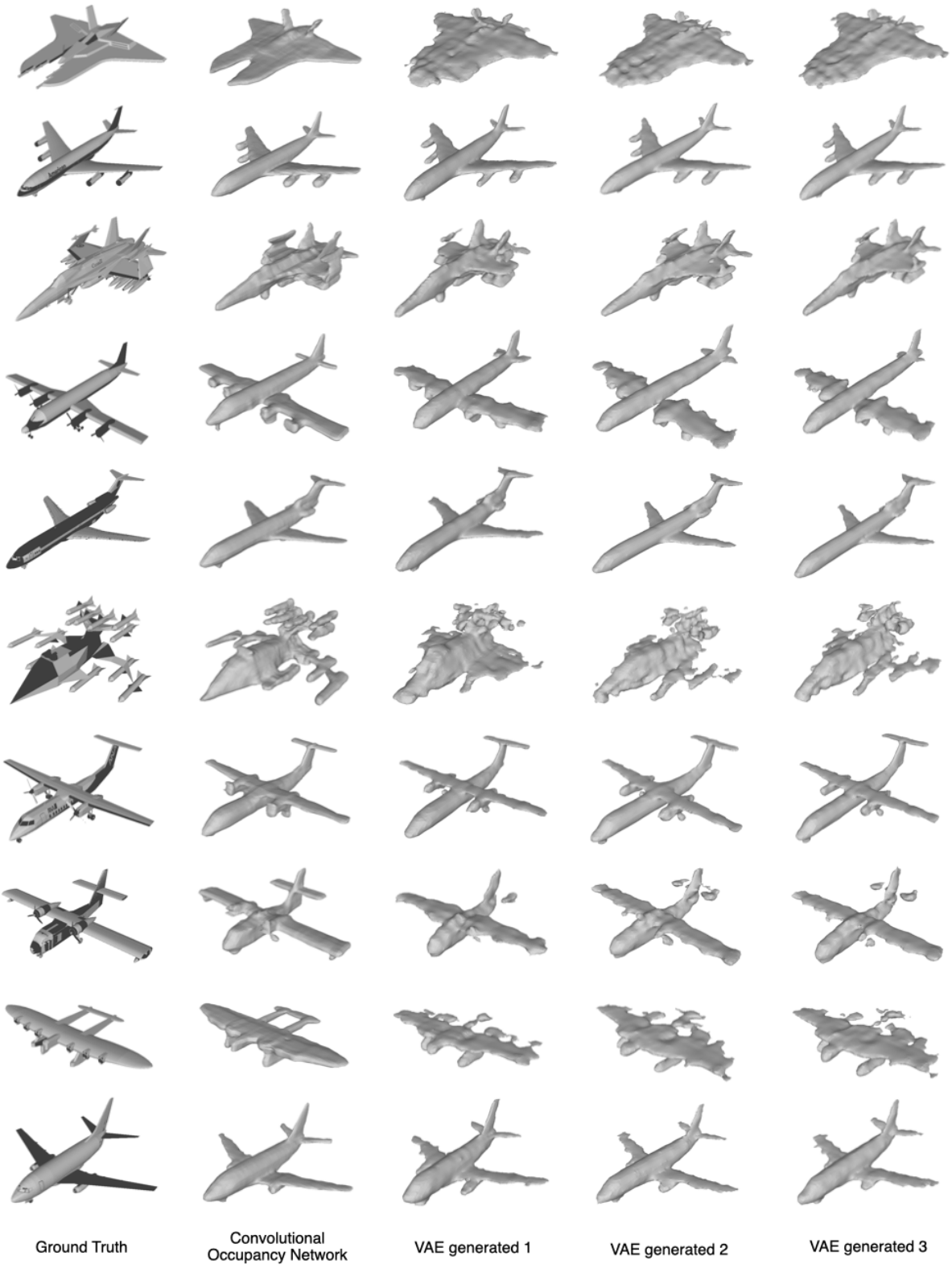


Figure 5.2: Comparison of Reconstruction Results: Our VAE vs. Baseline on ShapeNet

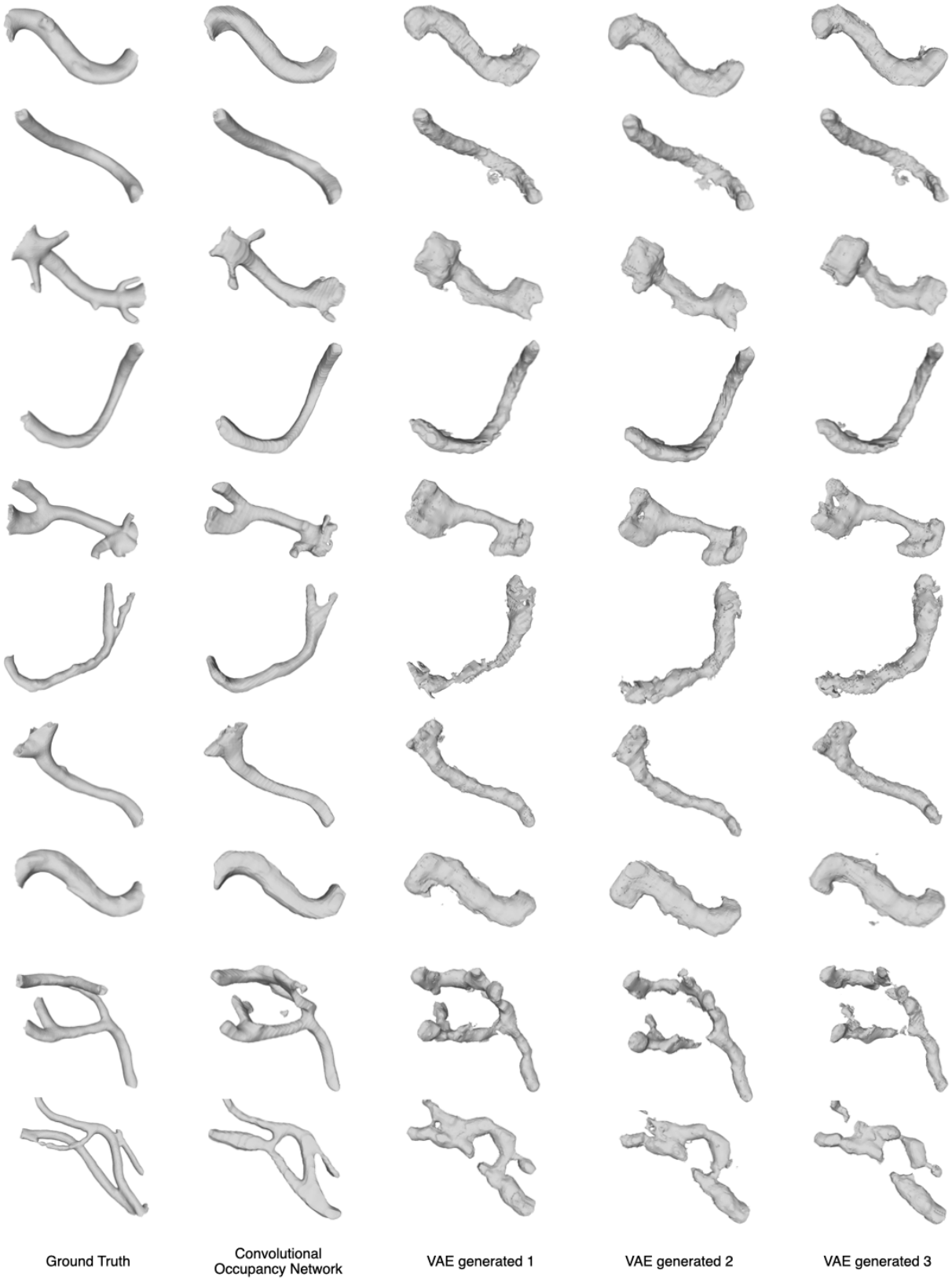


Figure 5.3: Comparison of Reconstruction Results: Our VAE vs. Baseline on IntrA

Chapter 6

Discussion

6.1 Limitations

The experimental results across the two distinct datasets reveal a substantial disparity in the performance of our VAE model. When applied to the ShapeNet dataset, the VAE model demonstrates that it can generate similar objects. Despite its limitation in producing entirely novel samples beyond its training data, the synthetic data closely resembles the authentic samples from the dataset. However, it's essential to emphasise that the generated samples lack the requisite level of detail necessary for real planes. These directly generated planes fall short of meeting the criteria for use in research demanding a high degree of precision, such as fluid mechanics tests.

Conversely, when applied to the IntrA dataset, owing to the dataset's limited sample size and its inherent high variance, a more stringent set of constraints was necessary to enable the network to primarily learn the overarching structural characteristics.

Two primary factors, constrained by GPU limitations, have influenced the outcomes of our

experiments on the IntrA dataset. Firstly, the choice of encoder type may significantly impact the model's performance. According to the authors of the foundational work [1], the volume encoder tends to excel in complex scene datasets. However, due to GPU memory constraints, we were compelled to employ a volume encoder with a resolution of 8^3 — a substantial reduction from the original 32^3 resolution.

Secondly, as a result of this limitation, we opted for a plane encoder with a resolution of $32^2 \times 3$, as opposed to the higher-resolution $64^2 \times 3$ employed in the baseline architecture [1]. Upon analysing the generation results obtained from the baseline model, we observed that despite using a resolution of 64, specific details could not be reconstructed with accuracy. Given the further reduction in resolution in our VAE model, combined with the introduction of noise, the model encountered amplified difficulties in capturing sophisticated features.

The outcomes stemming from the ShapeNet dataset can be perceived as a proof of concept for our model. However, it is essential to recognise that certain prerequisites exist for the dataset, preventing the model's applicability across all conceivable scenarios.

To begin with, the VAE is constructed under the assumption that the latent variables follow a continuous distribution. In practice, a larger dataset is advantageous in enabling the VAE to capture a more precise representation of the underlying distribution. This becomes particularly crucial when dealing with intricate data patterns. A larger dataset helps alleviate concerns such as overfitting and mode collapse, which can hinder model performance.

The ShapeNet dataset has about 4000 samples, where the objects are consistently scaled and oriented. On the other hand, the IntrA dataset contains around 1000 samples, which are uniformly scaled but exhibit diverse shapes and orientations. By using data augmentation techniques to enhance the IntrA dataset, we could see a significant boost in the performance

of the network. We conclude that when dealing with more complex datasets, having a larger number of samples is crucial for the model to learn effectively and perform well.

6.2 Alternative approaches

In addition to the variational autoencoder, alternative deep learning models exist that can be considered for the task of generating 3D synthetic data [22]:

- **Diffusion models:** Diffusion models, drawing inspiration from non-equilibrium thermodynamics, establish a Markov chain of diffusion steps [23]. This chain gradually introduces random noise to the data and subsequently learns to reverse this diffusion process, enabling the synthesis of desired data samples from the noise. In contrast to variational autoencoders (VAE) or flow models, diffusion models are trained with a predetermined procedure, and they often involve high-dimensional latent variables.

- **Generative adversarial networks (GANs):** GANs consist of two neural networks, namely a generator and a discriminator, engaged in a competitive dynamic [24]. The generator's objective is to generate synthetic data that the discriminator is unable to differentiate from real data. Conversely, the discriminator's task is to accurately distinguish between real and generated data instances. This adversarial interplay drives the generator to improve its ability to produce high-quality synthetic data that progressively becomes more indistinguishable from real data.

- **Normalising Flows:** NF operate on the principle of transforming a basic distribution, such as a Gaussian distribution, into a more intricate distribution that aligns with the

data distribution. This process involves employing a series of invertible transformations, guaranteeing the model's capacity to both generate novel samples and assess the probability density of a specific sample [25]. In the context of generation tasks, each data point signifies a coordinate in space. The transformations learned by the NF have the capability to map coordinates from the initial straightforward distribution to coordinates within the intricate data distribution. Through the mechanism of sampling points from the initial distribution and subsequently applying the learned transformations, the NF achieves the generation of fresh data points.

- **Energy-based models (EBMs):** EBMs operate on the foundational insight that any probability density function can be represented through an energy function that assigns low values to realistic points and high values to unrealistic ones. The central objective of EBMs is to learn an energy function that associates low energies with instances present in the training data [26]. For generating synthetic data, EBMs follow an iterative enhancement process driven by Langevin dynamics. This involves executing a process of noisy gradient descent on the energy function to converge toward configurations with low energy levels. Unlike GANs, VAEs, and Flow-based models, EBMs don't necessitate the use of an explicit neural network for sample generation. Instead, the generation process occurs implicitly [27].

6.3 Future work

6.3.1 Improvement of current model

As discussed in Section 5.1, the potential of the volume encoder for scene-level reconstruction is evident. However, our GPU's memory limitations prevented us from testing it at the baseline's default resolution or higher. It is our belief that conducting tests with the volume encoder at resolutions such as 32^3 or even higher could yield performance improvements. This is particularly relevant as it enables the model to better capture the intricacies of the IntrA vessel dataset. Additionally, it's reasonable to anticipate that an increase in the encoder's resolution would have a positive impact on the model's overall performance.

Furthermore, our experiments have confirmed that a lower latent code dimension leads to subpar performance. Thus, it is reasonable to assume that a higher latent code dimension would be better suited for datasets with more complex geometries. However, this assumption would require validation on a more powerful GPU.

6.3.2 Diffusion models

Recently, a considerable amount of attention has been directed towards diffusion models. Numerous publications in this domain have shown remarkable results, highlighting the considerable potential that diffusion-based networks offer, particularly in the context of generating 3D data, a domain of interest in our study as well.

Zeng et al. introduced a pioneering approach for 3D shape generation by presenting the Hierarchical Latent Point Diffusion Model (LION) [28] in 2022. This model is designed to attain exceptional generation quality, offer manipulation flexibility, and enable the production

of smooth surfaces or meshes. Structured as a VAE, LION incorporates a hierarchical latent space that combines a global shape latent representation with a point-structured latent space. To facilitate generation, Denoising Diffusion Models (DDMs) are employed in these latent spaces.

LION offers multiple advantages including expressivity, varying output types, and flexibility. It can be easily adapted for different tasks without retraining the latent DDMs. For example, LION can perform multimodal shape denoising, voxel-conditioned synthesis, and can be adapted for text-and-image-driven 3D generation. It can also perform shape autoencoding and latent shape interpolation. In empirical evaluations, LION has demonstrated state-of-the-art generation performance across various ShapeNet benchmarks.

In 2023, Erkoc et al. introduced a groundbreaking approach called HyperDiffusion [29], which employs weight-space diffusion to generate implicit neural fields. This novel model constitutes a fusion of implicit neural representation and diffusion modeling, showcasing the generation of high-quality 3D synthetic data and even extending to 4D animation sequences.

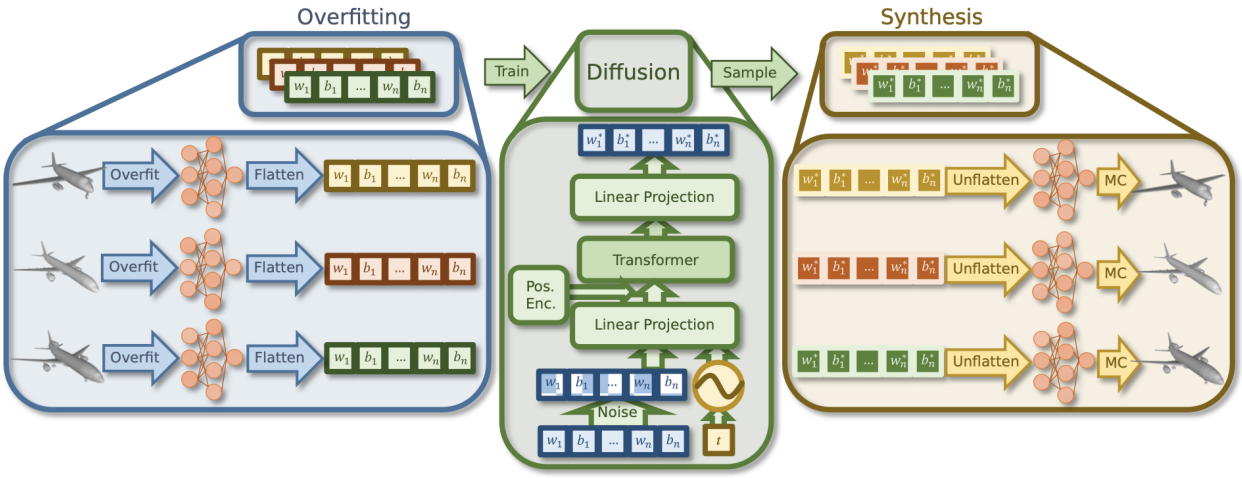


Figure 6.1: Overview of HyperDiffusion [29]

The architecture of HyperDiffusion is shown in Figure 6.1. The method unfolds in several steps. Initially, the authors fit a MLP to capture a neural occupancy field for a given data sample. The weights of the MLP are then flattened into 1D vectors and subjected to a diffusion process, effectively treating them as ground truth signals. In the subsequent stage, noise is directly added to the MLP weights, and these noisy weights are fed into a transformer-based architecture designed to simulate a diffusion process. This process forecasts the denoised MLP weights and biases as flattened vectors. In the final step, the model synthesises new neural field representations, from which meshes can be extracted using the Marching Cubes algorithm.

This innovative method opens up novel prospects for generative modeling in the domain of high-dimensional and intricate data. HyperDiffusion shows the potential of alternative representations within the framework of diffusion models, offering promising avenues for handling complex data structures.

Chapter 7

Conclusion

In conclusion, our experiments on 3D data synthesis using the ShapeNet dataset is a proof of concept for our approach. Our VAE model demonstrates the capability to generate synthetic 3D data that closely resembles the input data, showing its ability to capture and reproduce complex 3D structures. However, it is important to acknowledge several limitations of our model.

Firstly, our VAE struggles to generate samples that are significantly different from the data it has been trained on, highlighting its challenge in producing entirely novel 3D objects. Secondly, the generated data may not be directly suitable for sophisticated fine-tuning or training of other networks, as it may require specific post-processing steps to meet the requirements of certain applications. Thirdly, due to hardware constraints, we were unable to test our model with a higher encoder/decoder resolution and a larger latent code dimension. Future work with improved GPU resources could explore the impact of these factors on the model's performance and capabilities.

Furthermore, recent advances in diffusion models, as discussed in Section 6.3.2, have shown promise in generative tasks. It would be beneficial to evaluate these diffusion models on the

IntrA dataset, given its high variance, to assess whether diffusion models are more adept at handling such challenging data distributions.

Bibliography

[1] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision (ECCV)*, 2020.

[2] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[5] Xi Yang, Ding Xia, Taichi Kin, and Takeo Igarashi. Intra: 3d intracranial aneurysm dataset for deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[6] Michael Firman, Oisin Mac Aodha, Simon Julier, and Gabriel J Brostow. Structured completion of unobserved voxels from a single depth image. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-CNN. *ACM Transactions on Graphics*, 36(4):1–11, jul 2017.

[8] N. Sedaghat, M. Zolfaghari, E. Amiri, and T. Brox. Orientation-boosted voxel nets for 3d object recognition. In *British Machine Vision Conference (BMVC)*, 2017.

[9] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey, 2020.

[10] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[11] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds, 2020.

[12] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: A network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):90:1–90:12, 2019.

[13] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. Meshnet: Mesh neural network for 3d shape representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8279–8286, 2019.

[14] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[15] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *arXiv*, 2020.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[17] David Stutz and Andreas Geiger. Learning 3d shape completion under weak supervision. *CoRR*, abs/1805.07290, 2018.

[18] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *CVPR*, 2017.

[19] James Arvo. Fast random rotation matrices. *Graphics Gems III*, 1992.

[20] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2019.

[21] Maxim Tatarchenko, Stephan R. Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn?, 2019.

[22] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of VAEs, GANs, normalizing flows, energy-based and

autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, nov 2022.

[23] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[25] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, nov 2021.

[26] Yann Lecun, Sumit Chopra, and Raia Hadsell. *A tutorial on energy-based learning*. 01 2006.

[27] OpenAI. Implicit generation and generalization methods for energy-based models. `https://openai.com/research/energy-based-models`.

[28] Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. Lion: Latent point diffusion models for 3d shape generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[29] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion, 2023.

Appendix A

Uniformly sampled 3D rotation

```
import numpy as np

def uniform_random_rotation():

    """Generate a random rotation in 3D, with a distribution uniform
    over the sphere.

    Returns:
        Array of shape (n, 3) containing the randomly rotated vectors of x,
        about the mean coordinate of x.
    Algorithm taken from "Fast Random Rotation Matrices" (James Avro, 1992):
    https://doi.org/10.1016/B978-0-08-050755-2.50034-8
    """

    def generate_random_z_axis_rotation():
        """Generate random rotation matrix about the z axis."""
```

```
        R = np.eye(3)
        x1 = np.random.rand()
        R[0, 0] = R[1, 1] = np.cos(2 * np.pi * x1)
        R[0, 1] = -np.sin(2 * np.pi * x1)
        R[1, 0] = np.sin(2 * np.pi * x1)
        return R


    # There are two random variables in [0, 1) here (naming is same as paper)
    x2 = 2 * np.pi * np.random.rand()
    x3 = np.random.rand()
    # Rotation of all points around x axis using matrix
    R = generate_random_z_axis_rotation()
    v = np.array([
        np.cos(x2) * np.sqrt(x3),
        np.sin(x2) * np.sqrt(x3),
        np.sqrt(1 - x3)
    ])
    H = np.eye(3) - (2 * np.outer(v, v))
    M = -(H @ R)

    return M
```

Appendix B

Orientation alignment

```
import os
import pathlib
import argparse
import numpy as np
import trimesh

def process_model(filepath, name):

    # load mesh
    mesh = trimesh.load_mesh(os.path.join(filepath, name))
    vertices = np.array(mesh.vertices)

    # take 10% as sample points
    N = vertices.shape[0]
    indices = np.random.choice(np.arange(N), size=N//100)
    vertices = vertices[indices]
```

```
17
18      # Fit a line for the main axis
19      centroid = np.mean(vertices, axis=0)
20      deviations = vertices - centroid
21      uu, dd, vv = np.linalg.svd(deviations)
22      direction = vv[0]
23
24      # Compute rotation matrix to align with the x-axis
25      rotation_matrix = create_rotation_matrix_to_x_axis(direction)
26      mesh.apply_transform(rotation_matrix)
27      mesh.export(os.path.join(filepath, name))
28
29  # rotate the main axis to x-axis
30  def create_rotation_matrix_to_x_axis(vector):
31
32      # Normalize the vector
33      vector = vector / np.linalg.norm(vector)
34
35      # Compute the rotation axis (cross product with x-axis)
36      axis = np.cross(vector, [1, 0, 0])
37
38      # Compute the rotation angle (dot product with x-axis)
39      angle = np.arccos(np.dot(vector, [1, 0, 0]))
40
41      # Compute the cosine and sine of the rotation angle
42      c = np.cos(angle)
43      s = np.sin(angle)
```

```
44
45      # Compute the components of the rotation matrix
46      x, y, z = axis
47      rotation_matrix = np.array([[c + x*x*(1-c),   x*y*(1-c) - z*s, x*z*(1-c)
    + y*s, 0],
48                                  [y*x*(1-c) + z*s, c + y*y*(1-c),   y*z*(1-c)
    - x*s, 0],
49                                  [z*x*(1-c) - y*s, z*y*(1-c) + x*s, c + z*z
    *(1-c),   0],
50                                  [0,               0,               0,
           1]])
51
52      return rotation_matrix
53
54  if __name__ == "__main__":
55
56      parser = argparse.ArgumentParser(description='Path of the mesh to be
    processed.')
57      parser.add_argument('path', type=str, help='Path to be parsed.')
58      args = parser.parse_args()
59
60      filepath = args.path
61      filenames = []
62      for file in pathlib.Path(filepath).glob("*.off"):
63          filename = str(file).split("/")[-1]
64          filenames.append(filename)
65
```

```
66      filenames.sort()
67      N = len(filenames)
68
69      for i in range(N):
70          name = filenames[i]
71          process_model(filepath, name)
72          print(i, name)
```