GENERATION AND MODIFICATION OF 3D MODELS WITH DEEP NEURAL NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CİHAN ÖNGÜN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
INFORMATION SYSTEMS

SEPTEMBER 2021

Approval of the thesis:

GENERATION AND MODIFICATION OF 3D MODELS WITH DEEP NEURAL NETWORKS

submitted by **CİHAN ÖNGÜN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Information Systems Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, Graduate School of **Informatics** _____

Prof. Dr. Sevgi Özkan Yıldırım
Head of Department, **Information Systems** _____

Prof. Dr. Alptekin Temizel
Supervisor, **Modelling and Simulation, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Banu Günel Kılıç
Information Systems, METU _____

Prof. Dr. Alptekin Temizel
Modelling and Simulation, METU _____

Assoc. Prof. Dr. Aysu Betin Can
Information Systems, METU _____

Assist. Prof. Dr. Ufuk Çelikcan
Computer Engineering, Hacettepe University _____

Assoc. Prof. Dr. Aydın Kaya
Computer Engineering, Hacettepe University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: CİHAN ÖNGÜN

Signature :

ABSTRACT

GENERATION AND MODIFICATION OF 3D MODELS WITH DEEP NEURAL NETWORKS

ÖNGÜN, CİHAN
Ph.D., Department of Information Systems
Supervisor: Prof. Dr. Alptekin Temizel

September 2021, 100 pages

Artificial intelligence (AI) and particularly deep neural networks (DNN) have become very hot topics in the recent years and they have been shown to be successful in problems such as detection, recognition and segmentation. More recently DNNs have started to be popular in data generation problems by the invention of Generative Adversarial Networks (GAN). Using GANs, various types of data such as audio, image or 3D models could be generated.

In this thesis, we aim to propose a system that creates artificial 3D models with given characteristics. For this purpose, we focus on latent modification and generation of 3D point cloud object models with respect to their semantic parts. Different to the existing methods which use separate networks for part generation and assembly, we propose a single end-to-end Autoencoder model that can handle generation and modification of both semantic parts, and global shapes. The proposed method supports part exchange between 3D point cloud models and composition by different parts to form new models by directly editing latent representations. This holistic approach does not need part-based training to learn part representations and does not introduce any extra loss besides the standard reconstruction loss. The experiments demonstrate

the robustness of the proposed method with different object categories and varying number of points, rotations and scales. The method can generate new models by integration of generative models such as GANs and VAEs and can work with unannotated point clouds by integration of a segmentation module.

Keywords: 3D Model Synthesis, Point cloud, Generative Adversarial Networks (GAN), Generative Models

ÖZ

DERİN SİNİR AĞLARI KULLANILARAK 3B MODELLERİN ÜRETİLMESİ VE DÜZENLENMESİ

ÖNGÜN, CİHAN
Doktora, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Prof. Dr. Alptekin Temizel

Eylül 2021 , 100 sayfa

Derin Sinir Ağları (DSA) başta olmak üzere Yapay Zeka sistemleri tespit, tanıma ve bölütleme problemlerinde çok başarılı olduklarını göstererek son yılların en popüler çalışma konusu haline geldiler. Çekişmeli Üretici Ağların (ÇÜA) keşfiyle birlikte DSA veri üretme konusunda da son zamanlarda sıkça kullanılır oldu. ÇÜA kullanılarak ses, görüntü veya 3B model gibi değişik veri tipleri kolaylıkla üretilebilir hale geldi.

Bu tezde, verilen karakteristiklere uygun olarak yapay 3B model üretebilen bir sistem önerilmektedir. Bu amaçla, anlamsal parçalarına uygun olarak 3B nokta bulutu modellerinin örtük düzlemde düzenlenmesi ve üretilmesi üzerine çalışılmıştır. Parçaların üretilmesi ve birleştirilmesi için farklı ağlar kullanan önceki metotlardan farklı olarak, hem 3B modelleri hem de parçalarını düzenleyebilen ve üretebilen tek bir uçtan uca Otokodlayıcı modeli önerilmektedir. Önerilen model direkt olarak örtük temsilleri değiştirerek 3B nokta bulutu modelleri arasında parça değişimini ve değişik parçaların bir araya getirilerek yeni nesnelerin oluşturulmasını desteklemektedir. Kullanılan bütünsel yaklaşım anlamsal parçaları öğrenmek için parça temelli eğitime

ihtiyaç duymamaktadır ve standart yeniden oluşturma kaybı haricinde farklı bir kayıp fonksiyonuna ihtiyaç duymamaktadır. Yapılan deneyler modelin farklı nesne kategorilerine, farklı yönlerde, büyüklüklerde ve farklı sayıda nokta içeren nesnelere karşı dayanıklılığını göstermektedir. Method ÇÜA ve Değişimsel Otokodlayıcı gibi üretici modellerin entegrasyonu ile yeni nesneler üretebilmektedir ve bir bölütleme modülünün eklenmesi ile bölütlenmemiş nokta bulutu modelleri ile çalışabilmektedir.

Anahtar Kelimeler: 3B Model Üretimi, Nokta Bulutu, Çekişmeli Üretici Ağlar, Üretici Modeller

To my beloved wife...

ACKNOWLEDGMENTS

I wish to express my thanks and gratitude to Prof. Dr. Alptekin Temizel. During my whole academic life, he was always eager to help me and he is the main inspiration for me. I consider him a role model. He has done everything to make this thesis done by supervising the research, giving the personal motivation to work and finding the necessary sources.

I would also thank my thesis committee for their valuable comments and suggestions.

I would like to thank my "brothers" Cem Keser, Erdinç Yasin Dinç and Yusuf Türkyılmaz. They were always with me in all cases and I am sure they will always be.

My biggest gratitude is to my parents (Ayten Öngün, Nadir Öngün). They supported me for all the decisions I made and inspired me with their education mentality and determination. I am proud to be their son.

Last but not least, I would like to thank my wife Merve, my biggest supporter and motivation source. This thesis would not be possible without her unconditional love, support and encouragement.

This work has been supported by Middle East Technical University Scientific Research Projects Coordination Unit under grant number GAP-704-2020-10071.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

1D / 2D / 3D	1 / 2 / 3 Dimensional
AE	Autoencoder
AI	Artifical Intelligence
ANN	Artificial Neural Networks
CD	Chamfer Distance
CGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Networks
EMD	Earth Mover's Distance
GAN	Generative Adversarial Networks
GPU	Graphics Processing Units
MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology database
MSE	Mean Squared Error
PCA	Principal Component Analysis
ReLU	Rectified Linear Units
RGB	Red-Green-Blue color space
VAE	Variational Autoencoder
WGAN	Wasserstein Generative Adversarial Network

CHAPTER 1

INTRODUCTION

Pervasiveness and extensive practical uses of AI have started to change how people benefit from computers. Countries started to invest large amounts of resources on AI leading to increasing popularity and research output. While the theoretical background of current AI applications has been in development since 1950s, their recent popularity is mainly due to the developments in hardware, software and data science.

In the hardware area, parallel programming and development of multi-core systems accelerated the algorithms and enabled various applications. Graphics processing units (GPUs) provide massive parallelism to support graphics applications. Use of GPUs for other applications led to a paradigm shift in the software development. Instead of developing applications as a sequential process for single core systems, developers started to implement programs as many parallel operations on many processors. By this way, developers can run their applications on thousands of cores in parallel on GPUs instead of a few cores on CPU.

In-line with this massively parallel approach, software development has become more parallel oriented and try to use resources better and more efficient. They have more control over memory operations, network, processing tasks and data management. Also with the development of mobile, network, internet and cloud systems; software systems tend to do more task at once to use all of benefits of these system. Nowadays, there are too few applications that do not use any parallel processing. Even the most basic applications, which are developed by single developer, use parallel programming techniques in some way.

Hardware and software is not sufficient to create a well-trained AI and data is a crit-

ical element in training your system. While it was difficult to collect the necessary amounts of data in the past, nowadays it is possible to create datasets with millions of samples in a few minutes using internet. With the advancement in mobile and sensor technology, sensors are everywhere. Even a single mobile phone has many sensors to collect data like camera for images, microphone for sound, GPS for position, IMU for movement etc. Collecting data from all sources, storing them in data warehouses and sharing them using internet give all developers around world the ability to train their AI systems with high precision and accuracy.

With the availability of necessary hardware, software and data tools to create successful deep learning applications, deep learning has infinitely many application areas, among which, the most prominent ones are object detection, recognition and labeling in the images. Big commercial companies like Google, Facebook or Amazon have systems for these tasks with human-level accuracy. These systems can look for specific objects in images, write captions, explain scenes and actions.

Same success can be seen in audio. AI systems can detect different sounds and identify them whether they are human sound, a musical instrument, an animal sound or else. Systems can recognize speech of different people and make speech-to-text translation easily. In call centers, AI systems are employed for some tasks and they are quite successful for identifying human speech. Also simultaneous speech-to-speech translation between different languages are now possible as a commercial product, thanks to AI systems.

While detection and recognition systems have been very successful, AI starts to become popular among other fields as well. One of them is new data generation. This data may be audio data, image or 3D model. It is highly applicable for commercial products and has lots of application areas.

1.1 Problem Definition

Deep Learning applications are already very successful in the 2D image domain. The recent models are even achieving better scores than humans in many tasks. While their performance in 3D domain is still behind the 2D domain, it is becoming increas-

ingly more popular. Recently, there is a surge in the number of studies focusing on the artificial generation of 3D models. There are many use cases for artificially generated 3D models such as virtual environments, simulations and 3D printing. There are already AI tools in popular 3D graphics software that make recommendations to users for creating better and more realistic 3D models.

Most of the 3D objects in the real world consist of semantic parts. These parts can be considered as 3D objects that forms another 3D object when integrated together. For a chair object; leg, seat, armrest and back parts are semantic parts of the chair object. We can define a chair better using these parts individually. Even if there are not well defined parts of an object, we use intuitive definitions like front or the right side of the object. There are also exceptions that does not have any semantic or intuitive part definitions, e.g. a ball, but such exceptions are rare and these also can be approached as single-part objects.

Considering the semantic properties of real-world complex 3D objects, we focus on an artificial neural network model that can process and generate 3D models by extracting semantic part information. Most of the studies in the literature do not have abilities to process parts of the 3D models. These studies only focus on global shapes as a whole. This approach does not allow part based abilities such as generating, modifying and changing a single semantic part in isolation [2, 14]. Human perception of 3D models are intuitively part based. Even if there is no part information and the object is the same color as a whole, humans perceive the object with parts. While modifying the objects, we define them with parts to specify the intended modification. There are also other studies in the literature, which has part based abilities [11, 32, 52]. These studies use different models to capture and generate different parts. An additional assembly module takes the generated parts and forms a 3D object. In this approach, there are several problems that needs to be solved at the assembly step as the aim is to assemble the parts by affine transformations like scaling, rotating and moving. Sometimes, there is no coherency between parts to form a realistic 3D model. Also, for higher number of parts, these models become more complex and the cost of training the model increases since we need more models for more parts.

We aim to propose a single end-to-end model that capable of handling both 3D objects

and its corresponding parts. By this way, it would be possible to generate and modify specific parts. It also considers the coherency of the parts since it knows the global structure of the shape. The part-based holistic approach is, in a way, similar to human perception of 3D objects considering due to its ability of handling both the global shape and the parts in a coherent way.

1.2 Contributions

In this thesis, we propose a holistic approach to learn the semantic properties of the parts with a single neural network model. The proposed model is an end-to-end Autoencoder model that represents the parts, in addition to the global shape, separately in the feature space. Making modifications in the feature space allows meaningful modifications by preserving semantic properties. This is in contrast to the traditional way of making modifications in the input space which results in a completely new model. The contributions of the proposed method are as follows:

- It handles part editing, modification and global model generation with a single architecture and eliminates the need for an additional network for part assembly. The parts generated by modifications of latent space stay coherent with the global shape.
- It does not require any additional loss function other than the standard reconstruction loss.
- It provides a generic solution to convert regular generative networks based on PointNet feature extraction into part-aware networks.
- It is scalable and can be used with different point cloud sizes, objects having different numbers of parts and parts having different resolutions.
- It can process models without any explicit part information during inference by integration of a segmentation module.

Additionally, the work presented in this thesis has led to the following publications:

- C. Öngün and A. Temizel, "Paired 3D Model Generation with Conditional Generative Adversarial Networks", Proceedings of the European Conference on Computer Vision (ECCV) Workshops, pages 473–487. Springer, 2018, https://doi.org/10.1007/978-3-030-11009-3_29 , [44]

- C. Öngün and A. Temizel, "LPMNet: Latent Part Modification and Generation For 3D Point Clouds" , Computers & Graphics, 96:1–13, 2021,

  https://doi.org/10.1016/j.cag.2021.02.006 , [72]

1.3 Outline of the Thesis

The thesis is organized as follows; firstly, in Chapter 2, the necessary background information on Generative Models, which is the heart of the thesis, is provided. Section 2.1 briefly explains the underlying Artificial Neural Networks. Generative Adversarial Networks (GAN) and their variations used in this research are explained in Section 2.2. Another generative model Variational Autoencoder (VAE) and its baseline Autoencoder (AE) structure is described in Section 2.3. Lastly in Section 2.4, the latent space, which is an abstract multi-dimensional representation of compressed data, is explained to understand latent space operations in Generative Models.

Chapter 3 provides necessary background information for processing and generating 3D models with Neural Networks. In Section 3.1, different 3D model representation techniques are explained. Then, metrics and functions for measuring similarities and distances and evaluating the 3D models are provided in Section 3.2. A literature survey is provided in Section 3.3, which consists of the previous studies and baselines for processing 3D models with Neural Networks.

Chapter 4 contains the details of our work on generating paired 3D voxel models with Conditional Generative Adversarial Networks. Section 4.1 explains the proposed method and the architecture. Section 4.2 provides the used dataset, implementation details, experiment design and the result of the experiments of the proposed method. Conclusions and contributions of this method are given in Section 4.3.

Chapter 5 explains the proposed method for modification and generation of point

clouds. The feature extraction step is explained in Section 5.1 to understand how point, part and global features are extracted using a 2-stage Max Pooling approach. Details of the Segmentation and Decoder modules are given in Sections 5.2 and 5.3 respectively. The integrated Generative Modules are explained in Section 5.4.

Experimental evaluation of the proposed method and the results are given in Chapter 6. Implementation details of the proposed method is given in Section 6.1 and the dataset is described in Section 6.2. The base model used in the experiments are explained in Section 6.3. To test and validate the proposed method, different experiments are designed. The experiment design methodology and the results of all different experiments are given in Section 6.4. The evaluation of the method with different metrics and evaluation methods are explained in Section 6.5. The method is compared with previous works and state-of-the-art models in Sections 6.6 and 6.7. An ablation study is provided to the evaluate method in different conditions in Section 6.8 and failure cases are analyzed in Section 6.9. Section 6.10 explains the training with multiple classes together. Experiments for continuous conditions are provided in Section 6.11. Lastly, the trained model is deployed to work on a public web page. The deployment steps are described in Section 6.12.

Finally, the conclusions and the future work are stated in Chapter 7.

CHAPTER 2

GENERATIVE MODELS

While there is no strict consensus in the field, machine learning systems can be categorized into three main groups as supervised, unsupervised and semi-supervised learning. In supervised learning, the aim is to learn a function to map inputs X to labels Y. Labeled data is required for training the system and our aim is to form a system after training to successfully label unlabeled data. Classification, regression, object detection, semantic segmentation, image captioning are some examples of supervised learning. Since there is labeled data, the success of the system is measured easily by testing it with labeled data and comparing the results to measure how much of the data is correctly labeled. However, most of the time, the data is labeled manually by people or extra resources are required to label the data for training, which is an important limitation.

In unsupervised learning, the aim is to learn some underlying hidden structure of the data. Here, system does not use any labels. The objective is to learn a meaningful structure from given raw input data to achieve the solution with the scope of the determined task. Clustering, dimensionality reduction, feature learning, density estimation, generating new data are some examples of unsupervised learning. Since labeled data is not needed, any data can be used as input if it is formed well enough to use in the system. Finding training data and forming datasets are cheap and easy. However, since there is no ground truth or correct labels, it is very difficult or sometimes even impossible to measure the success of the system. [31]

Semi-supervised learning is a hybrid approach of supervised and unsupervised learning where only a small portion of the data is labeled. When labeling the unlabeled data is very expensive or challenging, the small portion of labeled data can provide

a better guidance for unsupervised tasks. It also helps with better regularization of supervised learning algorithms with unlabeled data. [17]

Generative models are a subset of unsupervised learning [24] where the aim is to generate new data from the same distribution of given training data. Unlabeled raw input data is mostly enough to learn the pattern of input dataset for generating new samples. Evaluation of generative models is not straightforward as there is no general formula to determine if the generated data is successful and usable. Most of the time the success is evaluated manually since there is no mathematical way to calculate the performance of the system based on how good is the generated samples. However, to understand the quality of the generated data, the generated data is expected to have features below:

- Novel: The generated data must be different from any samples of input data. While there are some metrics to measure that, the most popular way is to measure the similarity of every generated data with every sample in input data. If it is not different, it means that system just copies the input data and does not generate new samples.

- Same distribution: The generated data must be in the same distribution with the input data. It means that generated data must be in the same structure, class and concept of input data. Otherwise, generated data would be out of scope and useless even though it is high quality.

- Meaningful: The quality or success of the generated data is difficult to measure. Mostly, visual perception is used to determine if the generated data is meaningful and similar to input data. For image or object generation, the success is to generate meaningful images or objects for human perception. Even all of the metrics and mathematical outcomes seem well; the generated data is not successful if it is not meaningful for human perception.

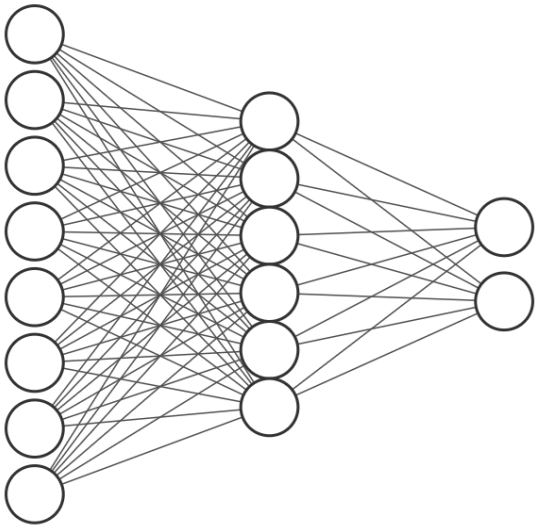


Figure 2.1: Architecture of Artificial Neural Networks.

2.1 Artificial Neural Networks

Artificial neural networks are computing systems inspired from biological neural networks. In machine learning, researchers need to specify features that the system should take into account in order to make decisions. It requires training the system manually where to look and what to search. On the other hand, artificial neural networks are capable of learning necessary features by themselves, which are important for that task, and improving it by training with more data. While successful modelling with less data, time and computing power are desirable in machine learning, artificial neural networks require lots of samples, long training times and high computing power. Since artificial neural networks mostly outperform classical machine learning techniques on complex tasks, these requirements are acceptable in exchange for performance. Also, the advancements in the hardware and algorithm optimizations alleviate these limitations.

While artificial neural networks have a history of more than 50 years, they have become popular in recent years since the technological advancements started to sufficiently meet its requirements. Especially, improvements in the hardware field made it feasible to train networks with higher number of nodes, layers and complex architectures. These deeper and more complex networks are widely called as deep neural

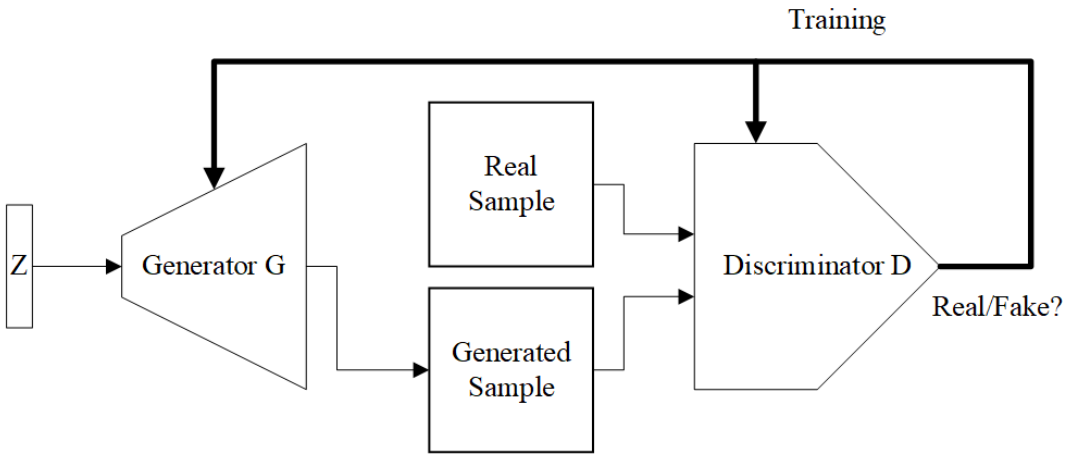


Figure 2.2: GAN structure.

networks.

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GAN) [18] is first proposed in the paper "Generative Adversarial Nets" in 10 June 2014 by Ian Goodfellow. In the paper, they employed an adversarial process for estimating generative models. The process consist of 2 simultaneously trained models, a generative model G and a discriminator model D. The generative model G generates data candidates and discriminator model D evaluates them.

Generative model G takes a random vector (or a latent variable) as input and produces an output. This output will be random and unrealistic at first because network parameters will be completely random at the beginning and the model needs to be trained to produce more useful outputs. The problem is to evaluate whether the output is realistic and then train the network based on this decision. Here, a second network, discriminator, is employed to evaluate the outputs and give necessary feedback whether they are realistic or not. The discriminator network can be imagined as a detective and the generator network as a forger. Forger tries to create more realistic fake paintings and detective tries to detect if they are real or not, so they both get better in time.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 2.3: GAN algorithm [18].

Assuming x is real data and z is a latent variable, GAN objective function can be defined as:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[log\ D(x)] + E_{z \sim p_z(z)}[log\ (1 - D(G(z)))] \quad (2.1)$$

The objective function is derived from Kullback-Leibler Divergence [29]. The Generator aims to minimize the result of the function and the Discriminator aims to maximize it. The result is simply the summation of the Discriminator $output$ for real data and $(1 - output)$ for generated data. Discriminator aims to increase the score for real data while decreasing the score for generated data and the Generator aims the opposite.

GAN gained immediate popularity by the researchers in the field and it is widely accepted as the state-of-the-art method in generative models. An example can be seen in Fig. 2.4. MNIST [30] is used to train a naive GAN model. At first the output is mostly a random noise since the model is not trained enough. After some training iterations, some outputs become more recognizable and realistic. At the end, many numbers are not distinguishable from real handwritten digits.

There are some common problems with GANs, reported by many previous works. These are basically identified as non-convergence because the main reason for all of

Figure 2.4: Output of GAN training with MNIST dataset for increasing iterations from left to right.

the problems are non-convergence. While GAN training is theoretically guaranteed to converge if all the parameters are modified directly, most of the time input data is not perfect for finding the best parameters. Unfortunately, these problems can be triggered in a seemingly random fashion, making it very difficult to play around with GAN architectures.

- Training difference : Since there are 2 separate neural network models in GANs, training speed of models is important to make them work together. If the discriminator learns faster, it can detect generated data as fake easily and the generator may never get better because all the data it generates will be labeled as fake. In the opposite case, if the generator learns faster, the discriminator fails to distinguish real and fake data so system never generates realistic data.

- Oscillation : GANs can train for a very long time and generate many different categories of samples. In some conditions, GANs generate very different samples each time that it never generates a better one. Instead of making a sample better over time, it generates a different one and it is labeled as fake by discriminator. In the next iteration it generates a completely different one and it is labeled as fake again so it never converges and makes oscillation through the whole training process.

- Mode collapse : It is the most severe form of non-convergence. If a data is labeled as real by discriminator, generator starts generating the same data each time since it will be labeled as real always. It is like getting stuck on a local

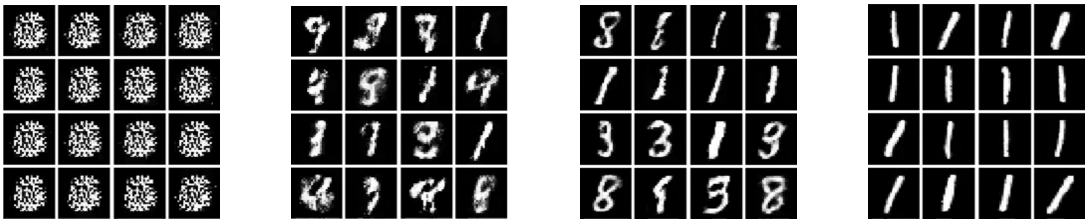


Figure 2.5: An example of mode collapse in GANs during training with MNIST. Images show the output of GAN during increasing training iterations from left to right.

maxima. For example, a GAN is trained with MNIST dataset which is a collection of handwritten digits from 0 to 9. Generator generates a 1 and it is labeled as real. Instead of generating all digits with the same probability, it always generates 1 because it will always be labeled as real by discriminator. An example of the process can be seen in Fig. 2.5. For images, researchers encounter this problem as identical output images for different input conditions.

2.2.1 Conditional Generative Adversarial Networks (CGAN)

Conditional Generative Adversarial Networks (CGAN) [40] were introduced as the conditional version of generative adversarial networks, which can be constructed by simply feeding the conditional variable, y, to condition on to both the generator and discriminator as can be seen in Fig. 2.6. It means that the additional information, y, is constraining the generator (and discriminator) to generate a certain type of output. For example, there is no control over the generated digit if the system is trained with MNIST [30] handwritten digits dataset. Every generated sample may contain different digits in different specifications randomly. With conditional GANs, a condition value can be set for a specific class, i.e. number for MNIST, sample of which is desired to be generated as can be seen in Fig. 2.7. The objective function of Conditional GAN can be defined as:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[log\ D(x|y)] + E_{z\sim p_z(z)}[log\ (1-D(G(z|y)))] \quad (2.2)$$

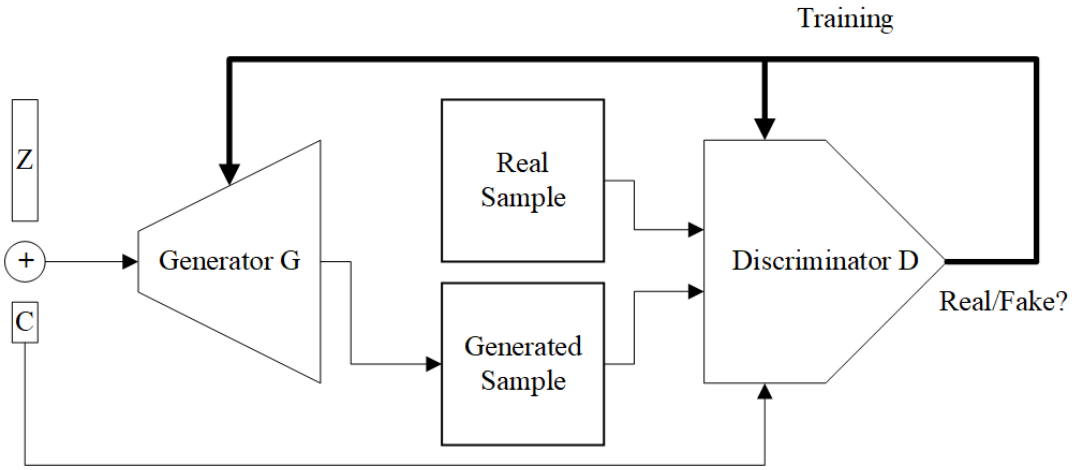


Figure 2.6: Conditional GAN structure.

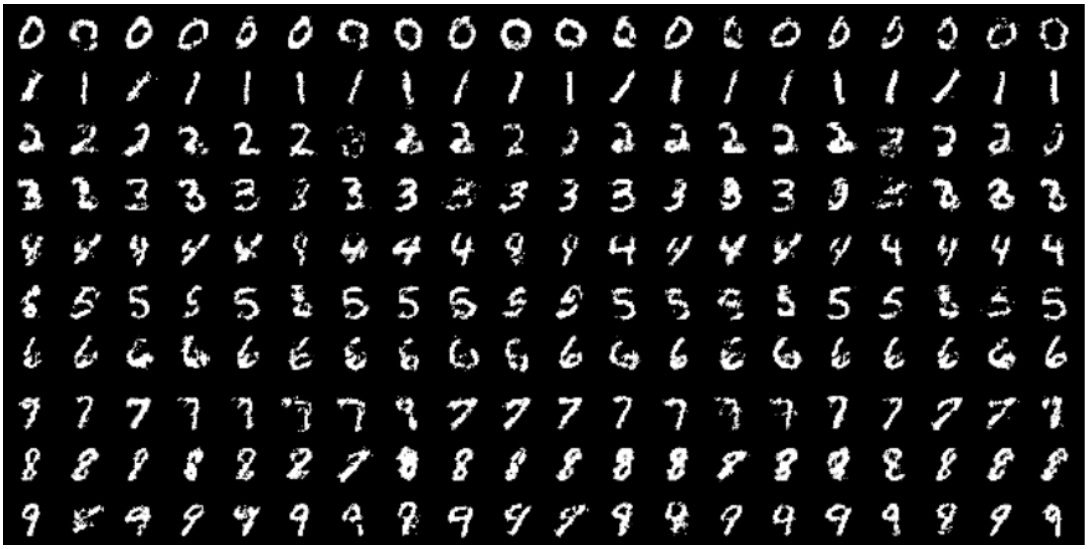


Figure 2.7: Generated MNIST digits with Conditional GAN, each row conditioned on one label [40].

Conditional GANs are very useful for multi-class training since it allows control over generating samples for a particular class. Pix2pix [23] employs a Conditional GAN to propose a general-purpose solution to image-to-image translation problems. The applied generic approach makes it possible to use the system for varying tasks such as synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images, among many other.

However, conditional GANs fail to generate the same sample with different conditions. If it generates faces, it generates different faces for different conditions even if the input is the same. For example, let's assume *happiness* is the condition and the condition variable is changed without changing the input variable to generate the same face in a different condition. It generates a *happy* face and a completely different *sad* face while the same face in *sad* condition is expected since the input vector is not changed. There is no way of generating the same face in different conditions with naive Conditional GAN. This is due to use of a latent vector as input and this behavior is called *entanglement* in GANs. While paired data is required to avoid this problem in naive Conditional GAN, there are many studies which do not require paired data for conditional generation. Coupled GAN (CoGAN) [34] proposes a new architecture which uses a pair of GANs with weight sharing for learning a joint distribution of multi-domain images. It does not require tuples of corresponding images in different domains for applications like domain adaptation and image transformation. The system generates image pairs sharing the same high-level abstraction while having different low-level realizations. DiscoGAN [25] aims to discover cross-domain relations of unpaired data. The proposed network transfers style from one domain to another while preserving key attributes such as orientation and face identity. CycleGAN [71] uses a similar approach for image-to-image translation. The study presents an approach for learning to translate an image from a soırce domain to a target domain. It is a highly popular study with many real-world applications. SyncGAN [8] proposes a novel network component named synchronizer for learning a synchronous latent space representing the cross-modal common concept. The synchronizer judges whether the paired data is corresponding to different domains or not. AlignGAN [36] uses a 2-step training algorithm to learn the domain-specific semantics and shared label semantics via alternating optimization.

InfoGAN [9] is another conditional approach to learn disentangled representations in a completely unsupervised manner. It aims to maximize the mutual information between a small subset of the latent variables and the observation. Besides standard Generator G and Discriminator D networks, it uses an extra Q network to predict the conditions of generated samples. Same as Conditional GAN, Generator is fed with a random vector and a condition variable to set the condition of the generated sample. Then the generated sample is fed to the Discriminator for evaluation and Q network to predict the condition variable. Generator and Discriminator is trained with a standard GAN loss. Q network is trained with the difference between given and predicted condition variable. By this way, system maps the condition variable to visible features of the generated samples in an unsupervised manner. The condition variable can be discrete or continuous numbers. If the system is trained with MNIST dataset without any conditional labels for experiments, it is able to learn visible continuous features such as how much italic or bold the generated digit is. Also, it can generate different digits with given discrete condition variables.

2.2.2 Wasserstein Generative Adversarial Networks (WGAN)

Discriminator loss measures the difference between the discriminator labels and true labels. To calculate the difference between these two distributions, commonly used metrics are Kullback–Leibler (KL) Divergence, Jensen–Shannon (JS) Divergence or Total Variation (TV) Distance. Arjovsky et al. [4] proposed a new distance metric to calculate the discriminator loss where Wasserstein Distance (Earth Mover's Distance) is used. Basically, probability distributions are defined by how much mass they put on each point. To move the mass around to change the distribution P_r into P_g, moving mass m by distance d costs $m \times d$ effort. The Earth Mover's Distance is the minimal effort to spend. GAN objective function using Wasserstein distance is formulated as:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[D(x)] \; - E_{z\sim p_z(z)}[D(G(z))] \tag{2.3}$$

The architecture of WGAN is the same as GAN, the only difference is the loss function for training. While calculating Wasserstein Distance is more difficult and

needs more iterations during training, it is more stable and has a higher probability to converge. In [4], they proved that some sequences of distributions that do not converge under the JS, KL, reverse KL, or TV divergence, can converge under the Earth Mover's Distance (EMD).

There are many different loss functions [35] proposed for better stability and high-quality generation. LSGAN [37] adopts the least squares loss function for the discriminator to overcome possible vanishing gradient problem. BEGAN [6] proposes a new equilibrium enforcing method for training Autoencoder based GANs. It aims to balance the discriminator and generator during training. DRAGAN [28] analyzes the reasons of mode collapse in GANs and proposes a gradient penalty scheme to avoid it. NSGAN [13] proposes a non-saturating loss, where the generator instead aims to maximize the probability of generated samples being real.

2.3 Autoencoders

Encoders and decoders are used in many areas in digital systems. Mostly they are used for encoding a signal to a smaller size signal and then decoding it when necessary. An autoencoder is a type of artificial neural network to learn efficient encoding-decoding of a data in an unsupervised manner. It generates a lower dimensional encoded vector from high dimensional input data and allows uncompressing that vector to closely match the original data. The output of the system is expected to be same as input so loss is calculated using the difference between input and output data and the system is trained using that loss to minimize it.

The most basic type of an autoencoder, also known as Vanilla Autoencoder, is a Multilayer Perceptron which contains an input layer to get data, a hidden layer to compress the data and an output layer same as input layer to decode the data. After training, input and hidden layers are used together as encoder, hidden and output layers together as decoder. While input and output layers must have same node count as input data dimension, hidden layer is designed with respect to determined compression rate. If a 28×28 image is required to compress into 10-dimensional vector, input and output layers must have 784 nodes while hidden layer has 10 nodes. Higher compression

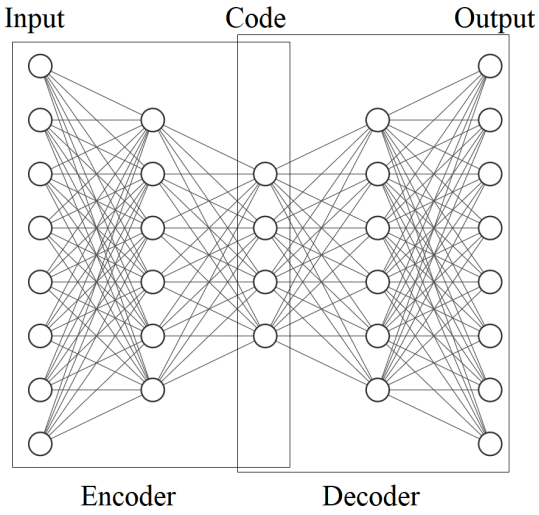


Figure 2.8: Schematic structure of an Autoencoder.

rate means more data compression with lower reconstruction quality.

In the literature, several other autoencoders for various tasks has been proposed. Stacked Autoencoders are used for more complex data for which Vanilla Autoencoders are fail to learn important features for compression. Stacked Autoencoders have higher number of hidden layers to filter different levels of abstractions and find better features. Convolutional Autoencoders [58] uses convolutional layers to process data. Denoising Autoencoders [61] are used for denoising operation. They are trained with a noisy image as input and clear image as output to learn how to denoise an image. At each iteration, the output of the system is compared with clear images to calculate the denoising quality. Context Autoencoders [47] are used for generating missing parts of the images. Among all these types of autoencoders, none of them has the purpose of generating new samples. These aforementioned autoencoder models require input-output data pairs and they are not capable of generating new samples. For this purpose, Variational Autoencoders, which can learn a distribution and hence, can generate new samples have been proposed.

2.3.1 Variational Autoencoder

The main difference of Variational Autoencoders [27] is, instead of learning a latent vector that represents the data, Variational Autoencoders learns a distribution.

Autoencoders learn to represent high dimensional input data with a point in low dimensional encoded vector. Different points in vector space represent different data samples in input space. Variational Autoencoders represent the data as distributions instead of points so it is possible to generate new data using these distributions. With same distribution, new data points can be used to generate new samples similar to that distribution. The output of encoder network is mean and variance parameters to represent the distribution and using this values new points can be generated in the same distribution. With these newly generated points, new samples can be generated using the decoder. The objective function of VAE can be defined using a variational lower bound, with an additional coefficient β to weigh the regularization term [21], as:

$$\mathcal{L} = \mathbb{E}_{q_\phi(z|x)}[logp_\theta(x|z)] - \beta D_{KL}(q_\phi(z|x)||p(z)) \tag{2.4}$$

where q and p are data projection and generation modules with parameters ϕ and θ respectively and D_{KL} is Kullback–Leibler divergence.

2.4 Latent Space

A neural network extract features through many layers. These layers can have varying architectures (fully connected, convolutional, recurrent etc.) and the main idea is to extract important information from previous layers. For a face classification task with convolutional networks, sequential layers may detect lines, edges, shapes, face parts and then faces. At each layer, abstraction of information goes higher. Each layer creates a feature space then passes it to the next layer. In the end, mostly before the last classification layer, the raw data is encoded to low dimension that contains the most important features and dense information. Since this is a learnable process and differs from network to network, the relation between input and feature space is not direct and it is complicated. This feature space is called “latent space”.

Generative Adversarial Networks use a random vector as input to generate data. Generator part takes a random input and generates a random data. This random input can be seen as a seed value. Like any random number generator, generator needs a seed value to generate new data. Given the same seed value, it generates the same data. Since the generator is a neural network, it learns to map the input values to output

values such that input space is actually a feature space. The representation between input random value and output data is also unknown so the input space is called as latent space. To generate new data, a latent vector is used from the latent space. This is the opposite of a classification task. For the classification task, the network learns to map the image space to latent space while for data generation, the network learns to map the latent space to image space.

Autoencoders are encoder-decoder networks which aim to encode or compress data with minimum loss. The size of compression is decided by the size of the bottleneck layer. Smaller the size means more compression with more loss. Since the representation between the real data and the codes is also unknown, the coded space is actually a latent space too. Encoder encodes the data from real space (image space) to latent space. Decoder does the opposite by decoding the codes from latent space to real space.

CHAPTER 3

3D MODEL GENERATION

3.1 3D Representations

There are different color spaces to represent images; however, most of the devices and studies use the standard RGB color space. RGB color space is structured and ordered since it is basically a 2D matrix with density values for Red, Green and Blue colors. A value in the matrix is called a pixel and it is the smallest representation of a color field. The higher number of pixels means higher resolution. Since the position of every pixel is static, neighborhood information between pixels is also static and easy to extract. The pixel size, number of pixels, maximum and minimum values of pixels and number of colors to represent are also static among all images in the data. Almost all capturing devices (cameras) and representation devices (monitors) work with RGB color space. Considering all these characteristics of RGB color space, one can easily work with it without hesitation since it is a worldwide standard (sometimes only option) color space.

Regarding the representation of 3D models, no particular representation technique stands out as each 3D data types has its pros and cons. Since there is no optimum solution, developers select the the most convenient data type for them. 3D capturing devices use different technologies and hardware to capture 3D information. Devices can use lasers to capture position of different points on the real world for high resolution, depth sensors to capture movement information, RGB sensors for color information, sonars for mapping etc. Raw 3D data captured by different scanning devices come in different forms varying in both structure and properties. In addition to capturing real objects, 3D models can also be created from scratch using one of the many software

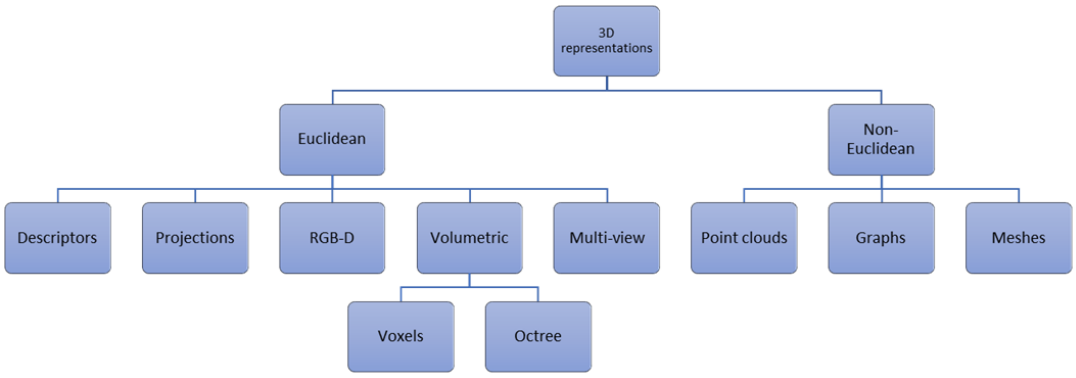


Figure 3.1: Various representations for 3D data [3].

available supporting different data types for easy modification.

The most common 3D representations are voxels, meshes and point clouds.

3.1.1 Voxels

3D voxels are 3D volumetric representations that preserve the underlying Euclidean-structure with a grid based three dimensional coordinate system. It is very similar to image representation with a 2D matrix. 3D voxels are basically 3D binary matrices. Each element of the matrix determines the existence of a unit cube in corresponding position. For example, an object can be represented with $(32 \times 32 \times 32)$ binary matrix in unit cube. The first element of the matrix represents the existence of a cube in $(x = 1/32, y = 1/32, z = 1/32)$ position. Voxels are very easy to modify and process with its 3D binary matrix structure. They are very popular amongst computer games due to its simple file structure.

Despite the simplicity of its structure, it is inefficient in terms of memory usage because it stores data for both occupied and non-occupied regions, also using resources for empty areas. This makes voxels a non-ideal structure for representing high-resolution data such as real-world objects and realistic 3D models. Also, as the 3D models are formed by cubes this has an undesired effect of creating non-smooth surfaces. An octree-based approach [38] can be employed for more efficient representation using varying-sized voxels.

3.1.2 Meshes

3D meshes are the most popular representations for computer graphics and 3D modeling software. While there are varying data structures, mostly there are 3 components in 3D meshes:

- **Vertex**: A position point in 3D coordinate system. It represents the corner points of the model.
- **Edge**: Connection between vertices. Represented as a line between two vertices.
- **Face**: A closed set of edges. 3 edges form a triangle face. Also, 4 edges can be used to form a quad face, depending on the structure of mesh data.

Meshes have the ability of representing high-resolution models and real-world objects better compared other techniques. A mesh model can be defined by any number of triangles/faces. The faces are big for plane surfaces and small for ares which has many edges and complex details.

The main problem of meshes is their complex data structure. Different components and their relations are stored by different data types which results in large file sizes. Another problem is the dynamic size of components. Different models are represented with different number of faces and resolutions. Because of these problems, meshes are not a popular choice for processing with neural networks. Neural networks are not readily compatible with such irregular representations. While there are some studies that works on meshes, most of them use heavy pre-processing and post-processing techniques to be able to work with meshes.

3.1.3 Point Clouds

Point clouds are set of points in 3D coordinate system that defines 3D models. A 3D model can be defined by any number of points which has three variables to define its position in (x, y, z) axes. It makes the file structure simple as well. If a model is represented with 1000 points, the file contains a (1000×3) matrix defining the

point cloud. These points can be visualized as spheres of predetermined size. Point clouds are mostly preferred by 3D scanners, making it popular among robotic applications and autonomous driving. Considering all advantages; capturing, visualizing and modification of point clouds are simpler compared to the other 3D representation methods.

There are some challenges for processing the point clouds because of the following properties:

- *Point clouds are unstructured.* There are no structure, hierarchy or part information. The file structure consists of a set of unrelated points.
- *There is no connectivity information.* The nearest points or the sequential entries in the file cannot be assumed to be neighbors. The points can be in different semantic parts, objects or even scenes.
- *There is no ordering.* Points in a point cloud can be in any order. The first point in the file can be in any position on the coordinate system. It may be the leftmost, rightmost or any point in the 3D point cloud. If there are N points in the point cloud, there can be $N!$ permutations of ordering in the corresponding file.
- *The number of points is dynamic.* There can be arbitrary number of points in a point cloud. A 3D model can be defined by any number of points to represent different resolutions. Each point cloud in a dataset can have different number of points. However, the most of the processing techniques assume a fixed number of input size.

In the context of this thesis, we first used voxel models. Voxels are the simplest data structures for 3D modeling and they are easier to use with neural networks. However, there is a limited number of datasets and voxels are not popular in 3D field because of their low resolution and unrealistic visualization. These factors limit the widespread acceptance of the works using the voxels. So after this initial work, we focused on point clouds as the main representation type for the remainder of the thesis. The advantages of point clouds over voxels are:

- There are many point cloud datasets. For example, chair dataset has around 1000 samples for voxel models while it has around 7000 samples for point clouds.

- All robotic applications and 3D sensors use point clouds. There are many application areas like self-driving cars, satellite imaging, architecture, etc.

- Higher resolution. Since the position of the points are dynamic and determined by the details of the object, it represents real-world objects better.

- More efficient data structure. Unlike voxels, empty spaces are not represented. Also, the values are not binary but normalized floats and this structure is more suitable for neural networks.

3.2 Metrics

Chamfer distance (CD) and Earth Mover's Distance (EMD) are the most commonly used metrics to measure the similarity of point clouds and compute the reconstruction error [12]. Both these metrics are permutation invariant and work on unordered sets.

3.2.1 Chamfer Distance

Chamfer Distance is a nearest neighbor distance metric for point sets. It is the squared distance of a point in the first set to the nearest neighbor point in the second set. Chamfer Distance between two point clouds S_1 and S_2 is defined as:

$$d_{CD}(S_1, S_2) = \sum_{p_1 \in S_1} \min_{p_2 \in S_2} \|p_1 - p_2\|_2^2 + \sum_{p_2 \in S_2} \min_{p_1 \in S_1} \|p_1 - p_2\|_2^2 \tag{3.1}$$

3.2.2 Earth Mover's Distance

Earth Mover's Distance (EMD) [51] (a.k.a. Wasserstein Metric) is an algorithm to measure the effort to transport one set to another. EMD for two equal-sized point

clouds S_1 and S_2 is defined as:

$$d_{EMD}(S_1, S_2) = \underset{\phi:S_1 \to S_2}{min} \sum_{p \in S_1} \|p - \phi(p)\|_2 \tag{3.2}$$

where ϕ is a bijection. While in practice, the exact computation of EMD is prohibitively expensive, an approximate method with reported approximation error around 1% has been used [12].

3.2.3 Cross Entropy

Cross Entropy Loss is the most used loss function in deep learning for classification tasks. Softmax activation function is applied on the output of a neural network to get class probabilities for each sample. Then the difference between the output and ground truth values are calculated using negative log-likelihood function. The loss can be described for output x as [46]:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right) \tag{3.3}$$

3.2.4 Mean Squared Error

Mean Squared Error (MSE) is commonly used for all kinds of tasks in machine learning field because of its simplicity. It is basically the average of the squared errors between predictions and labels. It is also the most common error metric to calculate reconstruction error between input and output values. For n samples, if y is the labels and $\widehat{y}$ is the predictions, the MSE can be formulated as :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2 \tag{3.4}$$

3.3 Previous Studies

Generating 3D models with neural networks is a relatively new area considering the first studies were published only a few years ago. Since there is already a well-established literature on images, the first attempts adopted the image generation techniques to the 3D field. First research works focused on generating 3D voxel models with 3D convolutional layers, then the studies diversified to other 3D representations. There are various surveys about deep learning on 3D data [3, 22, 66]. The related works in the scope of this study are explained in this section.

3.3.1 Related Works Based on Voxels

For GANs, voxels are the most common 3D object data type. It has the minimum data size compared to the others and it has a naive structure and ordering. For a $32\times32\times32$ resolution 3D object, one bit per cube is used to define its existence, resulting in a 4KB file size per object. Such small data sizes allow training a model with thousands of 3D objects in a feasible duration. Considering its cost effectiveness, most of the studies use voxels for that purpose.

The first study published for generating 3D data using GANs is 3D-GAN [63]. They designed an all-convolutional neural network to generate 3D objects. The discriminator mostly mirrors the generator. They observed that the discriminator learns faster than the generator because generating objects in 3D voxel space is more difficult than differentiating between real and synthetic ones. For this purpose, they employed an adaptive learning strategy, which the discriminator gets updated only when accuracy is less than 80% in the last iteration to keep the training pace similar in both networks. They worked with $64\times64\times64$ voxel resolution.

In the study of Smith and Meger [54], they employed Wasserstein distance normalized with gradient penalization as a training objective. While the previous works are successful at producing high quality objects from single class, it is difficult to train on distributions involving multiple distinct object classes in varied poses. They aim to pursue joint training over a mixture of categories. They used Wasserstein GAN to achieve a more stable convergence and allow one to track convergence through

the discriminator's loss. For 3D object training, they modified the GAN scheme to force generator to learn only every 5 batches while discriminator learns on every batch which leads to more stable convergence. The system is called 3D-IWGAN (3D-Improved Wasserstein GAN). They also use their system to reconstruct 3D objects from single image and depth scan. For this purpose, they use a variational autoencoder to convert the image into a latent vector and use this vector as an input to GAN. For 3D object generation, $32\times32\times32$ resolution is used and for image to 3D object reconstruction, images in IKEA Dataset [33] is converted to 3D objects with $20\times20\times20$ resolution.

3.3.2 Related Works Based on Meshes

There are only a few studies that directly consumes mesh file structure. Mesh R-CNN [16] is based on a system that detects objects in real-world images and produces a triangle mesh giving the full 3D shape of each detected object. It uses Mask R-CNN [19], a state-of-the-art 2D perception system, to detect objects in images and a mesh prediction branch outputs meshes with varying topological structure. First, a coarse voxel representation is generated, then it is converted to meshes and refined with a graph convolution network. SDM-Net [15] generates structured deformable meshes. It uses a 2-level VAE based approach for learning a deformable model of part geometries and the part structure of a shape collection. COALESCE [68] is a framework for component-based shape assembly. For a given set of input parts, it first align the parts with a part alignment network. Then a joint synthesis network generates a smooth and topologically meaningful connection between parts to form a plausible shape.

3.3.3 Related Works Based on Point Clouds

As mentioned in Section 3.1.3, point cloud data is basically a collection of 3D points in the Cartesian coordinate system. Point clouds can be easy to understand and visualize but they have important drawbacks to use in GANs. They are unordered and unstructured. Since point clouds are points in 3D space, they can be in any order and

they have no connection at all so there is no structure in point clouds. Studies about point clouds first address these problems and try to solve them by preprocessing techniques. Also solving these problems in better ways increases the overall performance of the system for generating more realistic data.

Given a point cloud, Gadelha et al. [14] build a kd-tree to spatially partition the points to solve the ordering problem. This orders them consistently across all shapes, resulting in reasonably good correspondences across all shapes. They then use PCA analysis to derive a linear shape basis across the spatially partitioned points, and optimize the point ordering by iteratively minimizing the PCA reconstruction error. Even with the spatial sorting, the point clouds are inherently noisy and the resulting distribution over the shape coefficients can be highly multi-modal. They used neural networks to learn a distribution over the shape coefficients in a generative-adversarial framework.

This solution first uses kd-tree algorithm for ordering problem. It makes spatial partitions. However, these partitions are not semantic or meaningful. The purpose here is to just find a standard ordering for models. Then they use PCA analysis to extract important features. They used GAN to generate new PCA coefficients. These coefficients are then converted into point cloud models using PCA reconstruction. Kd-tree algorithm and PCA can be replaced by neural networks to generalize the study since neural networks have the ability to converge to these algorithms.

PointNet [48] is the most popular study about point cloud processing with neural networks. PointNet is a novel type of neural network that directly consumes point clouds and well respects the permutation invariance of points in the input. It provides a unified architecture for applications ranging from object classification, part segmentation, to scene semantic parsing. The network first aims to extract features, then apply the tasks classification and segmentation.

To overcome the problem of ordering, they used symmetric functions. Symmetric functions are order invariant functions. Summation, multiplication and max operations are examples of symmetric functions. These functions give the same output for the same input variables regardless of the order. The study uses max operation on the final layer of feature extraction so network gives the same features for all permu-

tations of input variables. To overcome rotation problem, they used subnetworks to transform the model. These subnetworks learn necessary transformation coefficients and then transform the objects to have the same rotation. They used fully connected networks since neighborhood information is unknown and any point can be a neighbor to any other point.

They used static point size of 2048 for all models. Since there are 3 axes for x,y and z; all models are defined by 2048×3 variables. The system takes raw point clouds with input size 2048 and channel size 3 and transforms them with subnetworks. Feature extraction upsamples the features to 2048×1024. Max operation is used to select the most important points that define the model better. After the output layer, there are 1024 variables that describe the model. These variables can be used for classification and segmentation easily, which are done in 2 different networks.

PointNet++ [49] is an extended version of PointNet that is composed of a hierarchical neural network that applies PointNet recursively. The aim is to capture local structures to recognize fine-grained patterns and generalizability to complex scenes. There are also studies which proposes new convolutional methods for point cloud processing such as PointConv [65], KPConv [57], VV-Net [39] and Monte Carlo Convolution [20]. Some approaches convert point clouds into different representations for easier processing. DeepSDF [45] uses Signed Distance Functions to represent 3D shapes with continuous functions.

Achlioptas et al. [2] uses PointNet model as an encoder to design a generative model; an AutoEncoder and a GAN. The decoder is a 4-layer fully connected network. They used Chamfer Distance and Earth Mover's Distance to calculate the reconstruction loss. For modifying generated samples, they used interpolation and latent space arithmetic. These techniques are common in many latent representation models (AutoEncoders, GANs, etc.) For example, to add a handle to a cup, one can find codes for all cups with and without a handle. The average of differences between all cups with and without handles gives a latent code representing the existence of handle. By adding or subtracting this code, the existence of a handle can be controlled. However, this method only controls the existence of an attribute (i.e. provides a binary control), not the shape. Also, part interpolation is not possible since there is only a global code that

controls the shape with an entangled representation. Fan et al. [12] follows a similar architecture for generating point clouds from a single image.

3.3.4 Part Based 3D Shape Generation

For 3D generation with part editing and generation abilities, the most popular approach is handling parts separately by different networks then assembling them to form a plausible shape by an additional assembling network. The part handling networks may reconstruct or generate parts from scratch to learn about part characteristics.

Dubrovina et al. [11] proposes a novel neural network architecture, termed Decomposer-Composer, for the part interpolation problem. It uses a different Autoencoder for each part and a transformer network to combine these parts together. Each AE generates a part and the transformer network applies affine transformation to parts to form the shape. It operates on unlabeled voxel models. Some studies use the same approach. CompoNet [52] uses an AE for each part. After training AEs, they use encoders only to get codes for each part and another network for the composition of parts. PAGENet [32] uses VAE-GAN (Variational AutoEncoder Generative Adversarial Networks) to generate parts individually and an assembly module again to assemble them. VAE-GAN uses a Variational Autoencoder instead of a Generative network in GANs, so it is an Encoder-Decoder-Discriminator Architecture. Wang et al. [62] uses a global-to-local approach where a global overall structure of the shape is generated first and then a part-level refiner enhances the generated parts by refining and completing the missing regions. Tree-GAN [53] is a tree-structured graph convolutional network for multi-class 3D point cloud generation.

StructureNet [41] is one of the pioneer studies for 3D generation with part editing and generation capabilities. It is a hierarchical graph network which can directly encode and generate shapes represented as hierarchical graphs. It uses two encoders and two decoders to learn both part geometry and inter-part relations. The geometry encoder encodes the geometry of a part into a fixed-length feature vector and the graph encoder operates recursively to encode relationships among parts. The decoders are used as a reverse process to generate parts and structures. It achieves a variety of tasks such as

generating shapes and structures, part and geometry interpolation, shape abstraction and shape editing. The following study StructEdit [42] aims to generate shape edits and transfer edits between different shapes.

CHAPTER 4

PAIRED 3D VOXEL MODEL GENERATION WITH CONDITIONAL GAN

While the standard GAN model can generate realistic samples, it basically generates random samples in given input distribution and does not provide any control over these generated samples. For example, when a chair dataset is used to train the network, it generates chairs without any control over its characteristics such as its rotation. Conditional GANs provide control over the generated samples by training the system with given input conditions. For example, if rotation is used as a condition value for chair dataset, system can generate samples with a given rotation.

For both standard GAN and conditional GAN, the representation between the input and the output is highly entangled such that changing a value in the input vector changes the output in an unpredictable way. For example, for chair dataset, each chair generated by standard GAN would be random and it would be created in an unknown orientation. Conditional GAN allows specification of a condition. Input vector $\mathbf{z}$ and condition value $\mathbf{y}$ which are concatenated and given together as input to the system so input becomes $(z|y)$. As the condition value $\mathbf{y}$ is also an input value, changing the condition also changes the output. Even if the input vector $\mathbf{z}$ is kept the same, the model generates different independent samples in given conditions and does not allow generating the same sample in different conditions [34, 36]. For example, for chair dataset, if the condition is rotation, system generates a chair in first rotation and a different chair in different rotation. As these objects are different, they cannot be merged at a later processing stage to create a new sample with less artifacts.

4.1 Proposed Method

To overcome this problem, we propose incorporating an additional step in training to guide the system to generate the same sample in different conditions. The pseudo code of the method is provided in Algorithm 1 and Fig. 4.1 illustrates the proposed method for the 2-condition case. We use standard conditional GAN model and training procedure to generate samples by keeping the input vector $\mathbf{z}$ the same and changing the condition value. Generator function is defined as $G(z|y)$ for input vector $\mathbf{z}$ and condition value $\mathbf{y}$. We can define the function for same input vector and n different conditions as $G(z|y_n)$ and the domain specific merging operator as $M(G(z|y_n))$. We feed the merged result to discriminator to determine if it is realistic so the output of discriminator is $D(M(G(z|y_n)))$. Since the proposed method is an additional step to standard conditional GAN, it is a new term for the min-max game between generator and discriminator. The formulation of proposed method can be added to standard formulation to define the system as a whole. The objective function of conditional GAN with proposed additional training step can be formulated as follows:

$$\begin{aligned} \min_G \max_D V(D,G) = & E_{x\sim p_{data}(x)}[log\ D(x|y)] \\ & + E_{z\sim p_z(z)}[log\ (1 - D(G(z|y)))] \\ & + E_{z\sim p_z(z)}[log\ (1 - D(M(G(z|y_n))))]. \end{aligned} \tag{4.1}$$

As expected the system generates n different samples at n different rotations even though the input vector is the same. However as their rotations are specified by the condition, they are known. We then merge these samples to create a single object by first aligning these samples and then taking the average of the values for each voxel, similar to taking the intersection of 3D models. The merged model is then fed into the discriminator to evaluate whether it is realistic or not:

- If generated objects are different (as expected at the beginning), the merged model will be empty or meaningless. The discriminator will label the merged result as fake and the generator will get a negative feedback.
- If generated objects are realistic and similar, the merged model will also be

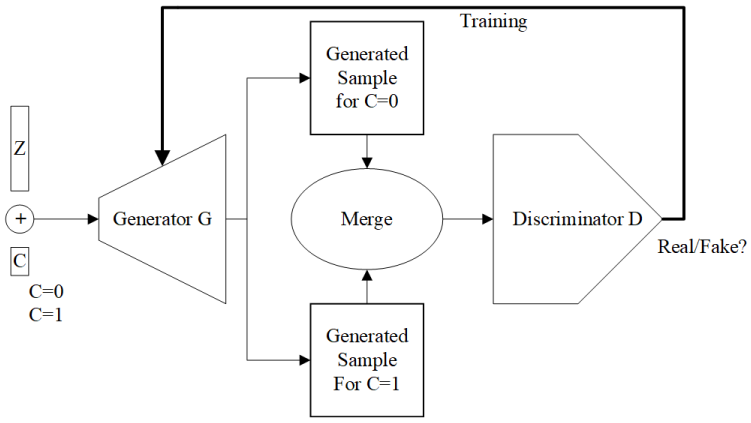


Figure 4.1: Proposed method illustrated for 2-condition case.

very similar to them and to a realistic chair model. The discriminator is likely to label the merged object as real and the generator gets a positive feedback.

By the help of this additional training step, even if the generated samples are realistic, system gets negative feedback unless the samples are similar. We enforce the system to generate similar and realistic samples in different conditions for the same input vector.

Note that the merge operation is domain specific and could be selected according to the target domain.

Algorithm 1. Conditional GAN training with the proposed method for n-conditions

Input: Real samples in n conditions: $X_0, X_1, \cdots, X_n$ input vector: Z, condition values: $C_0, C_1, \cdots, C_n$

Initialize network parameters for discriminator D, Generator G and merge operation M

for number of training steps **do**

// Standard conditional GAN

- Update the discriminator using $X_0, X_1, \cdots, X_n$ with $C_0, C_1, \cdots, C_n$ respectively
- Generate samples $S_0, S_1, \cdots, S_n$ using vector Z with $C_0, C_1, \cdots, C_n$ respectively
- Update the discriminator using $S_0, S_1, \cdots, S_n$ with $C_0, C_1, \cdots, C_n$ respectively
- Update the generator using $S_0, S_1, \cdots, S_n$ with $C_0, C_1, \cdots, C_n$ respectively

// Proposed method

- Align $S_1, \cdots, S_n$ with S_0
- Merge $S_0, S_1, \cdots, S_n : M(S_0, S_1, \cdots, S_n)$
- Feed merged sample to the discriminator with condition C_0
- Update the generator using the discriminator output

end for

4.2 Experiments

To test the system we used ModelNet [70] dataset to generate 3D models for different object classes (e.g. chair, bed, sofa). We adapted the conditional GAN for the problem of generation of 3D objects. We then evaluated the proposed method for 2-conditional

and 4-conditional cases. Visual results as well as objective comparisons are provided at the end of this section.

ModelNet dataset: This dataset contains a noise-free collection of 3D CAD models for objects. There are 2 manually aligned subsets with 10 and 40 classes of objects for deep networks. While the original models are in CAD format, there is also a voxelized version [70]. Voxels are basically binary 3D matrices, each matrix element determines the existence of unit cube in the respective location. Voxelized models have $30 \times 30 \times 30$ resolution. The resolution is set to $32 \times 32 \times 32$ by simply zero padding one unit on each side. For the experiments 3 object classes are used: chair, bed and sofa having 989, 615 and 780 samples respectively. Each sample has 12 orientations $O_1, O_2, \cdots, O_{12}$ with 30 degrees of rotation between them. In the experiments with 2 orientations we use O_1, and O_7 which represent the object in opposite directions ($0°$ and $180°$). Experiments with 4 orientations use O_1, O_4, O_7 and O_{10} ($0°$, $90°$, $180°$ and $270°$).

While there are more object classes in the dataset, either they do not have sufficient number of training samples for the system to converge (less than 500) or objects are highly symmetric such that different orientations come out as same models (round or rectangle objects). For different rotations, the system has been tested with paired input samples, unpaired (shuffled) samples or removing any correspondence between samples in different conditions by using one half of the dataset for one condition and the other half for other condition. The tests with different variants of input dataset show no significant change on the output.

Network structure: We designed our architecture building on a GAN architecture for 3D object generation [55]. In this architecture, the generator network uses 4 3D transposed deconvolutional layers and a sigmoid layer at the end. Layers use ReLU activation functions and the generator takes a 200 dimensional vector as input. Output of the generator network is a $32 \times 32 \times 32$ resolution 3D matrix. Discriminator network mostly mirrors the generator with 4 3D convolutional layers with leaky ReLU activation functions and a sigmoid layer at the end. It takes a $32 \times 32 \times 32$ voxel grid as input and generates a single value between 0 and 1 as output, representing the

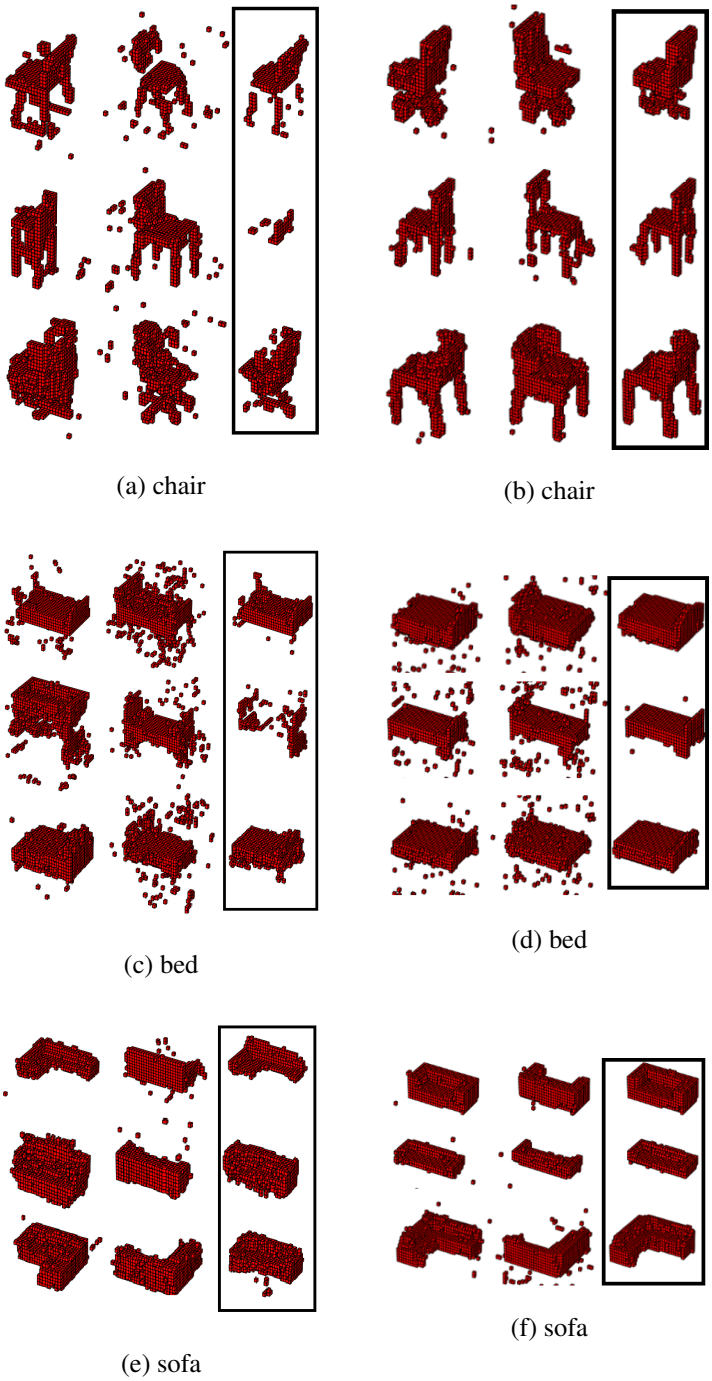
probability of a sample being real. Both networks use batch normalization between all layers. Kernel size of convolutional filters is 4 and stride is 2.

Adapting conditional GAN for generation of 3D models: To generate 3D models on different rotations, we modified the aforementioned GAN architecture and converted it into a conditional GAN. Conditional value $\mathbf{y}$ is concatenated into $\mathbf{z}$ for generator input. For discriminator input, $\mathbf{y}$ is concatenated into real and generated samples as an additional channel. To train the discriminator, we feed objects on different rotations with corresponding condition values. To generate pairs, we change only the $\mathbf{y}$ and keep the $\mathbf{z}$ the same.

Training: Since generating 3D models is a more difficult task than differentiating between real and generated ones, discriminator learns faster than generator and it overpowers the generator. If the learning pace is different between generator and discriminator, it causes instability in the network and it fails to generate realistic results [18]. To keep the training in pace, we used a threshold for discriminator training. Discriminator is updated only if the accuracy is less than 95% in the previous batch. The learning rates are 0.0025 for generator and 0.00005 for discriminator. ADAM [26] is used for optimization with β = 0.5 . System is trained using a batch size of 128. For 2 orientations, condition 0 and 1 are used for $0°$ and $180°$ respectively. For 4 orientations, condition 0, 1, 2 and 3 are used for $0°$, $90°$, $180°$, $270°$ respectively.

Visual results demonstrate that, standard conditional GAN fails to generate 3D models with the same attributes in different rotations. In 2-conditional case, it generates a chair with $0°$ orientation, and a completely different chair with $180°$ orientation for the same input value. On the other hand, the proposed system can generate 3D models of the same object category with same attributes with $0°$ and $180°$ orientations. Also the result of merge operation is given to show the intersection of models. Since intersection of noise is mostly empty, merged model is also mostly noise-free. For these 3 classes, system is proven to generate pair models on different rotations.

For additional training of the proposed method, samples are generated by keeping the input vector the same and setting the condition value differently. Then the outputs

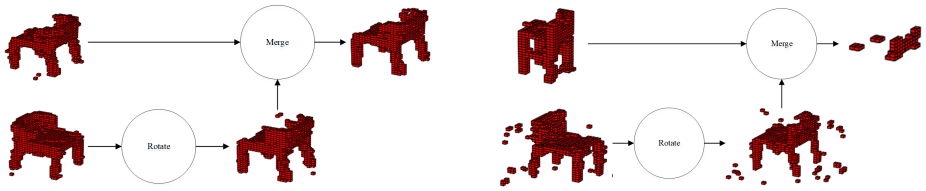


(a) chair (b) chair

(c) bed (d) bed

(e) sofa (f) sofa

Figure 4.2: Results with 3 classes (chair, bed and sofa) using 2-conditions (rotations). The first two samples are the generated pairs, merged results are shown in boxes. (a), (c) and (e) show the pairs generated with standard conditional GAN. It is clearly visible that the samples belong to different objects. Standard conditional GAN fails to generate the same object in different conditions (rotations) as expected and the merged results are noisy. (b), (d) and (f) show the pairs generated with the proposed method. The samples are very similar and the merged results (intersection of samples) support this observation.



(a) (b)

Figure 4.3: Examples of merging operation. After generating the pairs, one of the pairs is aligned with the other. Second sample is rotated to align with the first one in these examples. Then aligned samples are merged to form a new one. Simple averaging is applied to aligned pairs to get the intersection. (a) The result of the merging operation will be similar to the generated samples if the samples are similar, (b) the result will be meaningless if the samples are different.

are merged and fed into the discriminator. Only the generator is updated in this step. Experiments show that, also updating the discriminator in this step causes overtraining and makes the system unstable. Since this step is for enforcing the generator to generate the same sample in different conditions, training of the discriminator is not necessary.

Merge method: Merging the generated samples is domain specific. For our case, generated samples are 3D voxelized models with values between 0 and 1 representing the probability of the existence of the unit cube on that location. First aligning the samples generated with different orientations and then simply averaging their 3D matrices, we get the merged result. In Figure 4.3, we illustrate the merging procedure with a 2-conditional case with chair dataset. Generator will output two chairs with $0°$ and $180°$ rotations respectively. We can simply rotate the second model $180°$ to align both samples. Then, we average these 3D matrices. By averaging we get the probability of the existence of unit cubes in each location taking both outputs into account. If chairs are similar, the intersection of them will also be a similar chair (Fig. 4.3(a)) and if the chairs are not similar, their intersection will be meaningless (Fig. 4.3(b)). By feeding these merged results into the discriminator, we make the network evaluate the intersection model and train the generator using this information.

Results: The proposed framework has been implemented using Tensorflow [1] version 1.4 and tested with 3 classes: chair, bed and sofa. The code is publicly available at `https://github.com/cihanongun/3D-CGAN`. The results are observed after training the model for 1500 epochs with the whole dataset. Dataset is divided into batches of 128 samples. For comparison, we used the conditional GAN that we adapted for 3D model generation as the baseline method. Both systems have been trained with the same parameters and same data. Results are generated with the same input and different condition values. To visualize the results, binary voxelization is used with a threshold of 0.5. Fig. 4.2 shows the visual results. Note that the presented results are visualizations of raw output without any post processing or noise reduction.

As there is no established metric for the evaluation of generated samples, we introduce 2 different evaluation metrics: Average Absolute Difference (AAD) and Average Voxel Agreement Ratio (AVAR).

Raw outputs are 3D matrices for each generated model and each element of these matrices is a probability value between 0 and 1. For the calculation of AAD with n-conditions, first, the generated models $S_1, \ldots, S_n$ aligned with S_0 to get $S_1^R, \ldots, S_n^R$ then AAD can be formulated as follows:

$$AAD = \frac{\sum_{i=0}^{n-1} \frac{\sum_{\forall x,y,z} |S_i^R(x,y,z) - M(x,y,z)|}{total\ \#\ of\ matrix\ elements}}{n} \tag{4.2}$$

As a result of AAD a single difference metric is obtained for that object. A lower AAD value indicates agreement of the generated models with the merged model and it is desired to have an AAD value closer to 0.

For the calculation of Average Voxel Agreement Ratio (AVAR), first the aligned 3D matrices are binarized with a threshold of 0.5 to form voxelized S_i^{RB} M^B and then Average Voxel Agreement Ratio (AVAR) can be formulated as:

$$AVAR = \frac{\sum_{i=0}^{n-1} \frac{\sum_{\forall x,y,z} S_i^{RB}(x,y,z) \bigwedge M^B(x,y,z)}{\sum_{\forall x,y,z} S_i^{RB}(x,y,z)}}{n} \tag{4.3}$$

Table 4.1: Comparison of the proposed method with baseline using different object classes for 2-conditions and a batch (128) of pairs. AAD: Average Absolute Difference between generated matrices, AVAR: Average Voxel Agreement Ratio.

	Chair		Bed		Sofa	
	AAD	AVAR	AAD	AVAR	AAD	AVAR
Baseline	0.027	0.32	0.029	0.69	0.018	0.74
Proposed	0.009	0.79	0.012	0.89	0.004	0.95

where $\bigwedge$ is the binary logical AND operator. AVAR value of 0 indicates disagreement while a value of 1 indicates agreement of the models with the merged model and it is desired to have an AVAR value closer to 1.

Results for 2-conditions and a batch of 128 pairs are given in Table 4.1. AAD and AVAR results are calculated separately for each pair in the batch and then averaged to get a single result for the batch. The results show that the proposed method reduces the average difference significantly; 3, 2.4 and 4.5 times for chair, bed and sofa respectively. Here the results are highly dependent to object class. Different beds and sofas are naturally more similar than different chairs. While different bed shapes are mostly same except headboards, chairs can be very different considering stools, seats etc. Also we can see it in the results, the proposed method improved the similarity of generated chair pairs from 0.32 to 0.79. While the generated chair pairs are very different with the baseline method, the proposed method generated very similar pairs. For bed and sofa the baseline similarities are 0.69 and 0.74, relatively more similar as expected. The proposed method improved the results to 0.89 and 0.95 for bed and sofa respectively by converging to the same model.

The proposed system has also been tested with 4-conditions. For 4 orientations, condition 0, 1, 2 and 3 are used for $0°$, $90°$, $180°$ and $270°$ respectively. Also for merging operation, all generated samples are aligned with the first sample with $0°$ rotation. For that purpose 2^{nd}, 3^{rd} and 4^{th} samples are rotated by $270°$, $180°$ and $90°$ respectively. After aligning all 4 samples, they are merged into a single model by averaging.

Fig. 4.4 shows the visual results for 4-conditional case with the same experimental

Table 4.2: Comparison of the proposed method with baseline using different object classes for 4-conditions. The same metrics are used as in the 2-condition case.

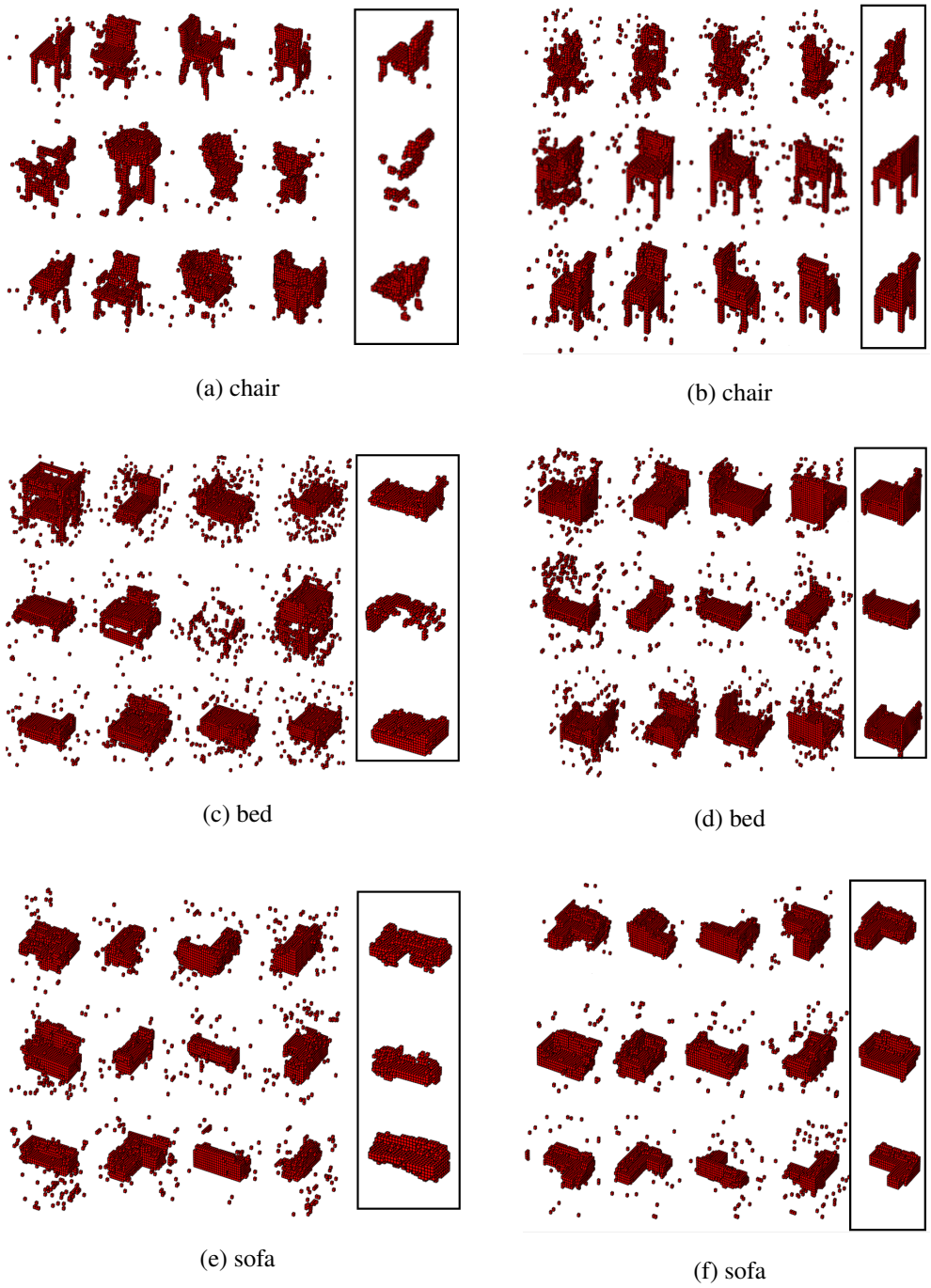
	Chair		Bed		Sofa	
	AAD	AVAR	AAD	AVAR	AAD	AVAR
Baseline	0.034	0.36	0.043	0.65	0.034	0.62
Proposed	0.024	0.61	0.021	0.82	0.013	0.90

setup. Experimental results in terms of the same metrics are presented in Table 4.2. Standard conditional GAN generates 4 different chairs on 4 rotations. On the other hand the proposed method enforces the network to generate the same chair on 4 different rotations. Since the problem is more complex for 4 rotations, individual generated samples are noisier and have lower resolution. The improvement rates compared to the baseline are relatively lower than 2-condition case because of the increased complexity of the problem. To account for the increasing complexity of the model with higher number of conditions, more training data and/or higher number of epochs need to be used. While generating better samples with more training may seem crucial, it does not change the behavior of the networks. Conditional GAN keeps generating different samples and proposed model generates paired samples with each training iteration.

4.3 Contributions

In this chapter, we presented a new approach to generate paired 3D voxel models with conditional GAN. First, we adapted the conditional GAN to generate 3D models on different rotations. Then, we integrated an additional training step to solve problem of generation of pair samples, which is a shortcoming of standard conditional GAN. The proposed method is generic and it can be integrated into any conditional GAN. The results show the potential of the proposed method for the popular problem of joint distribution learning in GANs.

We demonstrated that proposed method works successfully for 3D voxel models on 2 and 4 orientations. Visual results and the objective evaluation metrics confirm the



(a) chair (b) chair

(c) bed (d) bed

(e) sofa (f) sofa

Figure 4.4: Visual results with 4 conditions. The first four samples are the generated objects, merged results are shown in boxes. (a), (c) and (e) show the objects and the merged result obtained with standard conditional GAN. (b), (d) and (f) show the objects and the merged result obtained with the proposed method The samples are very similar and the merged results (intersection of samples) support this claim. Merged results are also mostly noise-free and have more detail compared to standard conditional GAN.

success of the proposed method. The difference between generated models are reduced significantly in terms of the average difference. The merged samples create noise-free high-resolution instances of the objects. This approach can also be used for generating better samples compared to traditional GAN for a particular object class.

The extension of the method to work with higher number of conditions is trivial. However, the training of the system may take a long time. The proposed solution is generic and could be applied to other types of data such as images with 2D representation.

The work presented in this chapter has led to a publication with the title "Paired 3D Model Generation with Conditional Generative Adversarial Networks" in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops - 2018* [44].

CHAPTER 5

MODIFICATION AND GENERATION FOR 3D POINT CLOUDS

After 3D voxel models, we moved to point cloud representation as explained in Section 3.1. To overcome the problem defined in Section 1.1, we propose an end-to-end Autoencoder model consisting of 3 modules: Feature Extractor, Segmentation and Decoder. The proposed method can be integrated with a generative model to provide generative capabilities for generating new shapes and parts.

5.1 Feature Extractor

The feature extractor is based on a modification of PointNet[48] feature extractor. The input and feature transform network are omitted since the data is already aligned and scaled. However, the experiments for different orientations and scales can be found in Section 6.4.5. The feature extractor extracts point, part and global features sequentially. These features define the corresponding point, part and global shape.

If the feature size is set to l, for a point cloud with n points and k parts, the size of point features is $n \times l$, part feature is $k \times l$, and global feature is l. The part features and global feature can be modified to change the corresponding part in isolation or the global shape. The system uses a 2-stage max pooling for extracting part and global features. The idea of 2-stage max pooling is explained in Section 5.1.2.

5.1.1 Point Features

Point feature extractor takes n points and outputs l features for each point. It uses 1D convolutions to extract per-point features. The first convolutional layer applies

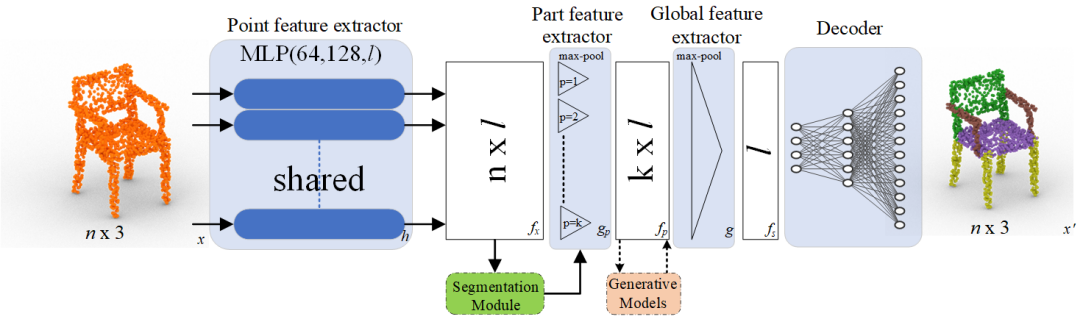


Figure 5.1: The proposed architecture consists of a point-wise feature extractor, a part feature extractor, a global feature extractor and a decoder. The optional generative model allows generation of new parts and models. The optional segmentation module allows the system to work with unlabeled data.

1D convolution to points with 3 channels (x, y, z coordinates). The following layers expands the feature size to determined size l. The multilayer 1D convolutional system is equivalent to a MLP with weight sharing [48]. This process gives a $n \times l$ point feature matrix.

5.1.2 2-stage Max Pooling

Max operation is a symmetric function that gives the same result for any ordering of the input variables. We apply max operation on the point features to get the feature of the corresponding part. Max operation gives the most important features of a point cloud. It extracts the critical points to define a part or global shape. Critical points are the most important points to define a shape. It can be considered as the skeleton of the shape. For example, a triangle can be defined with 3 corner points at minimum so the corner points are the critical points for a triangle. We can apply max pooling on all point features to get the global feature as in [48]. Any point set between the critical set and the point cloud gives the same global feature after max pooling.

To extract part features, we first apply max pooling on points of each part separately to get the part features. Then, we apply max pooling again on part features to extract the global feature. The 2-stage max pooling operation can be defined as max of maxes. It is similar to the "reduce max" operation in parallel programming. Directly applying

max operation on a vector of numbers gives the same result as applying the operation in multiple iterations. 2-stage max pooling operation gives the same global feature (beside part features) as applying max pooling directly on all point features.

5.1.3 Part Features

Part feature extractor uses $n \times l$ point features and part labels for k parts to extract part features. It uses max-pooling operation as defined in the previous subsection. The part labels are extracted by the Segmentation Module. Segmentation module gives per-point part labels defining which part the point belongs. Part feature extractor applies max pooling on the points of each part separately. It gives l features for each of part, resulting $k \times l$ matrix. Part features define the corresponding parts and can be modified to make latent modifications on a specific part.

5.1.4 Global Feature

Global feature extractor uses previously extracted $k \times l$ part features to extract global feature. It applies the max-pooling operation on the part labels. Global feature defines the global shape with l features and can be modified to modify the whole global shape.

5.2 Segmentation

To generate part features, part feature extractor uses point features and part labels. Part labels are extracted by the segmentation module. For a point cloud with k parts (for example, a chair model has $k = 4$ semantic parts; seat, back, arm and leg), represented with n points, there are n labels, associating each point with a part label. Segmentation module uses point features (extracted by point feature extractor) to generate per-point part labels. Then these labels are fed to the part feature extractor together with point features. Segmentation module is trained with the system during training using ground truth part labels. During inference, it generates part labels for unannotated raw point clouds. It provides the segmentation capability, making the system an end-to-end solution for unannotated point clouds.

The segmentation module can be trained in isolation instead of training with the system. Also, a pretrained segmentation module can be used. All alternative options generate similar segmentation performance within a range of 2% difference. The point features are concatenated with the global feature to improve the segmentation performance. This method decreases segmentation loss significantly. The global feature is extracted by applying max pooling directly on all point features.

If the part labels are available, the segmentation module can be omitted and the available labels can be directly fed to the part feature extractor. In this case, reconstruction performance will be better considering the part labels are not predictions but ground truths. However, the system will lose the ability to work with unannotated point clouds.

5.3 Decoder

Decoder generates $n \times 3$ point clouds from global features. It is implemented as a MLP but also a multi-layer 1D deconvolutional system can be employed for this purpose. During training, it is trained with reconstruction loss to generate the same given shape from the extracted global feature. The feature extractor encodes the given shape into a global features and the decoder is expected to decode it to the original shape. The decoder learns the mapping between global features and real shapes. During inference, the newly generated or modified global features are fed to the decoder to generate new or modified global shapes. The segmentation module is used for segmenting the generated shapes.

5.4 Generative Modules

The proposed encoder-decoder architecture has an inherent capability of generating new shapes by exchanging parts between models and combining parts from different models to form new shapes. The corresponding part features can be exchanged or combined together, then the new global feature is fed to the decoder to generate the new shape. However, these operations are based on the existing parts of the point

clouds. By integrating generative models, we can generate new part features thus generate novel parts and shapes. For this purpose, we integrated the system with two different generative models: GAN and VAE. The generative models are placed after part feature extractor to learn and generate part features.

5.4.1 GAN

The implemented GAN architecture does not generate 3D point clouds but generates new part features. The GAN works in latent space instead of actual data space, so it is called as latent space GAN (l-GAN) [2]. The newly generated part features can be used to form new global features, then decoded by the decoder to generate novel shapes. Also, the generated part features can be used in existing models for replacing a specific part with a novel one. After getting the part features for an existing model, a new part feature can be generated for the corresponding part to extract the global feature. The modified global feature is decoded to generate the shape consisting the new part. A WGAN [4] has also been implemented (l-WGAN) to observe the differences.

5.4.2 VAE

Autoencoders can be converted into Variational Autoencoders to provide sampling ability to generate new samples as mentioned in Section 2.3.1. We have converted the proposed autoencoder to VAE by adding linear layers to part feature extractor. The linear layers generate mean and sigma values to define the distribution of part features and provides sampling ability from the same distribution to sample new features. The previous VAE implementations are reported to suffer from poor reconstruction or generation capabilities [2]. It is because of the imbalance between reconstruction and regularization losses. To overcome this, a β coefficient is used to weigh the regularization term [21].

CHAPTER 6

EXPERIMENTS AND RESULTS

This chapter explains the exhaustive experiments of the proposed method in Chapter 5. The methodology of experiments, results and comparisons are provided. Also, an ablation study is made to see the effects of different options for the proposed method. Lastly, failure cases are analyzed to find possible reasons and solutions.

6.1 Implementation Details

All implementations have been done in PyTorch which is an open source machine learning and deep learning framework [46]. PyTorch3D library has been used for 3D operations [50]. The training took a few hours on a NVIDIA RTX2070 GPU for the base model. Code is publicly available at
`https://github.com/cihanongun/LPMNet`. The implementation is tested with different environments, random seeds and datasets. Also, the trained model is deployed as a web site to test in real world conditions. All tests demonstrate that the implementation is robust and can work on different environments in different conditions.

6.2 Dataset

Yi et al. [67] annotated the ShapeNet dataset [7] to include part labels for 16 categories. The number of parts for each category varies from 2 to 6. Each point in the point cloud sample has a semantic part label. From these 16 categories, chair, table and plane categories have been used for the study since they have the highest number

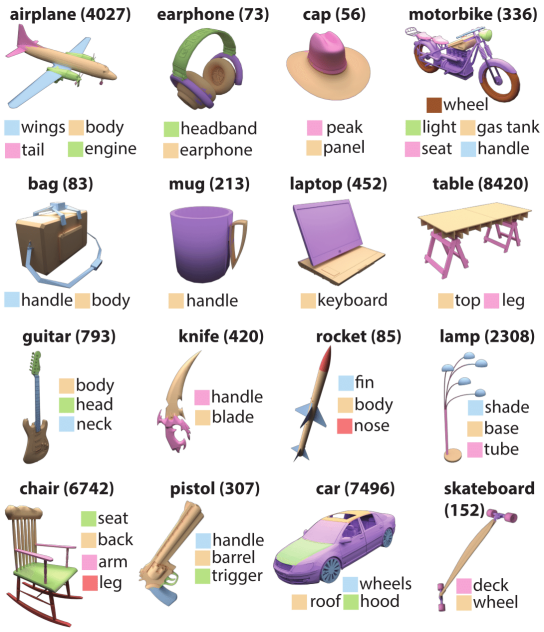


Figure 6.1: Some samples from the dataset [67].

of samples (3758, 5266 and 2690 samples, respectively). Some of the other categories (car, lamp) are only used for visualization and qualitative analysis. In the dataset, each sample has a different number of points varying from 500 to 3000 points. For all the experiments, 2048 points per sample have been used, unless otherwise stated. Random down-sampling or zero-padding have been applied if a sample has more or less points than the determined number of point. Parts have different number of points for each sample since we do not assume all parts contains the same number of points. Official train, validation and test subsets are used with 70%, 10% and 20% ratios respectively. The re-organized version of the dataset is used since only expert verified segmentations are included in that version.

6.3 Base Model

The encoder-decoder architecture is inspired from [2]. The point feature extractor follows the implementation principles of PointNet consisting a 3-layer MLP (64, 128, l) with weight sharing, implemented as 1D convolutions. Input and feature transform subnetworks are omitted since the samples are already aligned and scaled. Each layer is followed by a ReLU activation function and a batch normalization layer. The orig-

inal 5-layer architecture of PointNet has no advantage over proposed model since the dataset is less diverse and already aligned.

The segmentation module follows the same principles with more layers (64, 32, 16, k) and a softmax function at the end to get class probabilities. It is trained with cross-entropy loss. A 3-layer architecture gives similar performance with less overfitting but the performance drops with increasing feature size. Higher number of layers cause overfitting as the data is not complex and the proposed model is trained with single class. However, a more sophisticated architecture can be employed for more complex input data.

The decoder is a 3-layer MLP (1024, 2048, $n \times 3$), implemented with fully connected layers. The first two layer is followed by ReLU function and the last layer is a linear function. A tanh activation function can be used on the last layer if the point cloud is in [-1,+1] range for faster training convergence. Fewer number of layers fail to generate high quality samples while models with higher number of layers tend to overfit to training data. The decoder can also be implemented as a deconvolutinal network. A 5-layer (512, 256, 256, 128, 3) deconvolutional architecture has similar performance to the base model with less overfitting. However, deconvolutional model is sensitive to feature size and it fails when feature size is high (e.g. 1024).

For the base model, the feature size l is 128 and number of points n is 2048. The system has been trained using Chamfer distance as reconstruction loss and cross-entropy loss as segmentation loss. Adam optimizer [26] has been used with a learning rate of 5×10^{-4} for 1000 epochs using batches of 32 samples.

6.4 Experiments

We have conducted a number of experiments similar to those in the literature and introduced new ones. Unless otherwise stated, the base model has been used in all experiments with the distance metrics CD and EMD as explained in Section 3.2. The reconstruction performance is provided in Section 6.4.1. Part exchange and part interpolation abilities are provided in Section 6.4.2, followed by the shape generation by the composition of different parts in Section 6.4.3. New part and shape generation

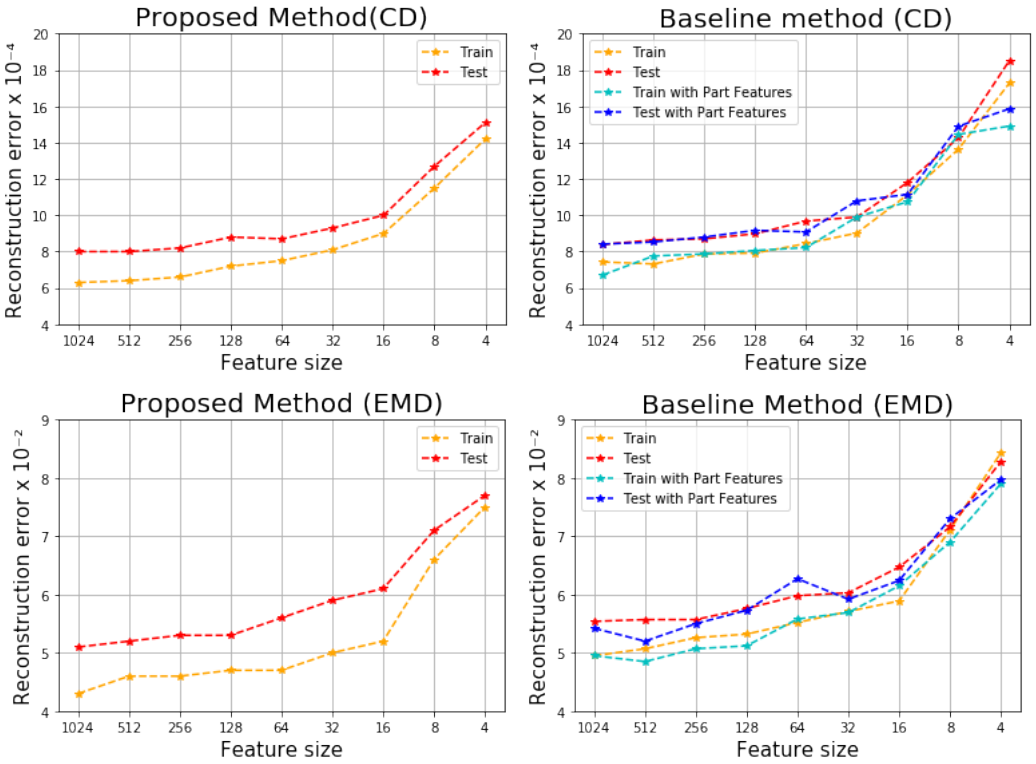


Figure 6.2: The reconstruction losses for different feature sizes.

with the integration of different generative models are explained in Section 6.4.4. The proposed method has been tested with different input sizes to prove its robustness against low-resolution data, missing points, different orientations and scales. The results are provided in Section 6.4.5.

6.4.1 Reconstruction

One of the important parameters of the proposed method is the feature (bottleneck) size. It determines how many features each point, each part and the global shape have after the feature extraction step. To decide the feature size, different feature sizes are tested with the base model for *chair* category. Fig. 6.2 shows the reconstruction losses calculated using Chamfer and EMD for different feature sizes. According to Fig. 6.2, the the baseline method [2] and the proposed method exhibit a similar trend. They both suffer from higher reconstruction loss when the feature size is less than 128. A higher feature size creates a bigger model with more complexity and

longer training times for insignificant improvement in reconstruction performance. A feature size of 128 provides a good balance to run the system with less training time without sacrificing reconstruction performance; so the feature size is set to 128 for all experiments.

In addition, to evaluate the effect of the part feature extractor on the reconstruction quality, the proposed part feature extractor has been integrated into the baseline method [2]. The results show no significant difference, supporting our claim that the global feature is not affected by the part feature extraction step.

The reconstruction results on the test set can be seen in Fig. 6.3. Visual results indicate good reconstruction performance with minor loss. Also, it shows the segmentation ability of the method for the unnatotated inputs.

6.4.2 Part Exchange and Interpolation

Part features represent the semantic properties and definitions of corresponding part. By modifying the part feature, shape of a respective part could be changed in isolation, keeping the other parts the same. Exchanging the part features with another sample also exchanges the corresponding parts in real shapes. We expect a smooth interpolation between parts of two shapes by an interpolation between part features of the same two shapes. It demonstrates a smooth and disentangled feature space. To prove this claim, we apply part interpolations for all parts separately between 2 shapes and show the results in Fig. 6.4 for *chair* class. Global feature interpolation results in a smooth global shape interpolation between the samples. For part feature interpolation, we interpolated only a specific part feature while keeping the other part features the same. The results show that, it interpolates only the corresponding part in isolation while modifying the connection points for a more coherent global shape. It can be seen that it is not a naive part interpolation that results global shapes with inconsistent parts and poor connections. Latent space represents the semantic properties of a part so it modifies the part to match the new shape better by preserving semantic properties. The results for *plane*, *car* and *table* classes can be seen in Fig. 6.5.

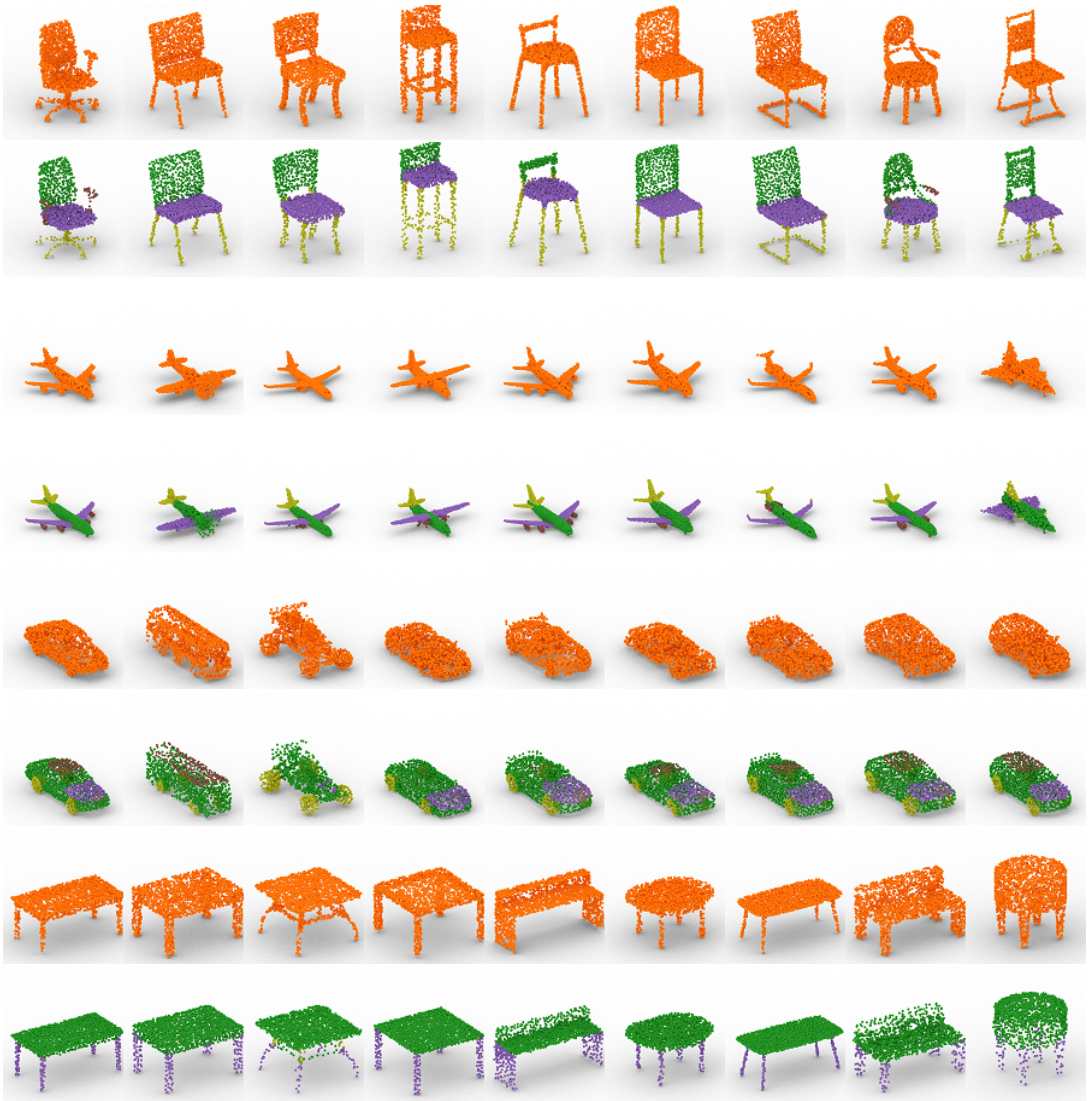


Figure 6.3: The reconstruction results of the proposed model. For each object class, the first row shows the samples from the unlabeled test set and the second row shows the corresponding reconstructions.

Figure 6.4: Part interpolation between two shapes. The first row is global shape interpolation between two shapes (leftmost and rightmost). Other rows are single part interpolations where only the corresponding part feature is interpolated while features of other parts are kept the same.

For example, if we exchange leg features of a four-legged dining chair and an office chair having wheels, the same office chair is generated with four legs instead of wheels. However, the new legs of the office chair will not be exactly the same as the source dining chair. The office chair is now generated with four legs which are in better harmony with the rest of the shape resulting in a more realistic looking chair.

6.4.3 Composition of Parts

Part features are expected to be independent of each other to form global shapes. We have exchanged part features independently in the previous section. Like replacing the part futures with different ones, we can use different part features from different shapes to generate new shapes. Different part features from different random shapes are merged to obtain a global feature. Part features carry the semantics of corresponding parts. A global feature is formed from part features that gathers all semantics together. The decoder decodes the global feature to generate a shape that represents all semantics. Sample results can be seen in Fig. 6.6. A new global shape

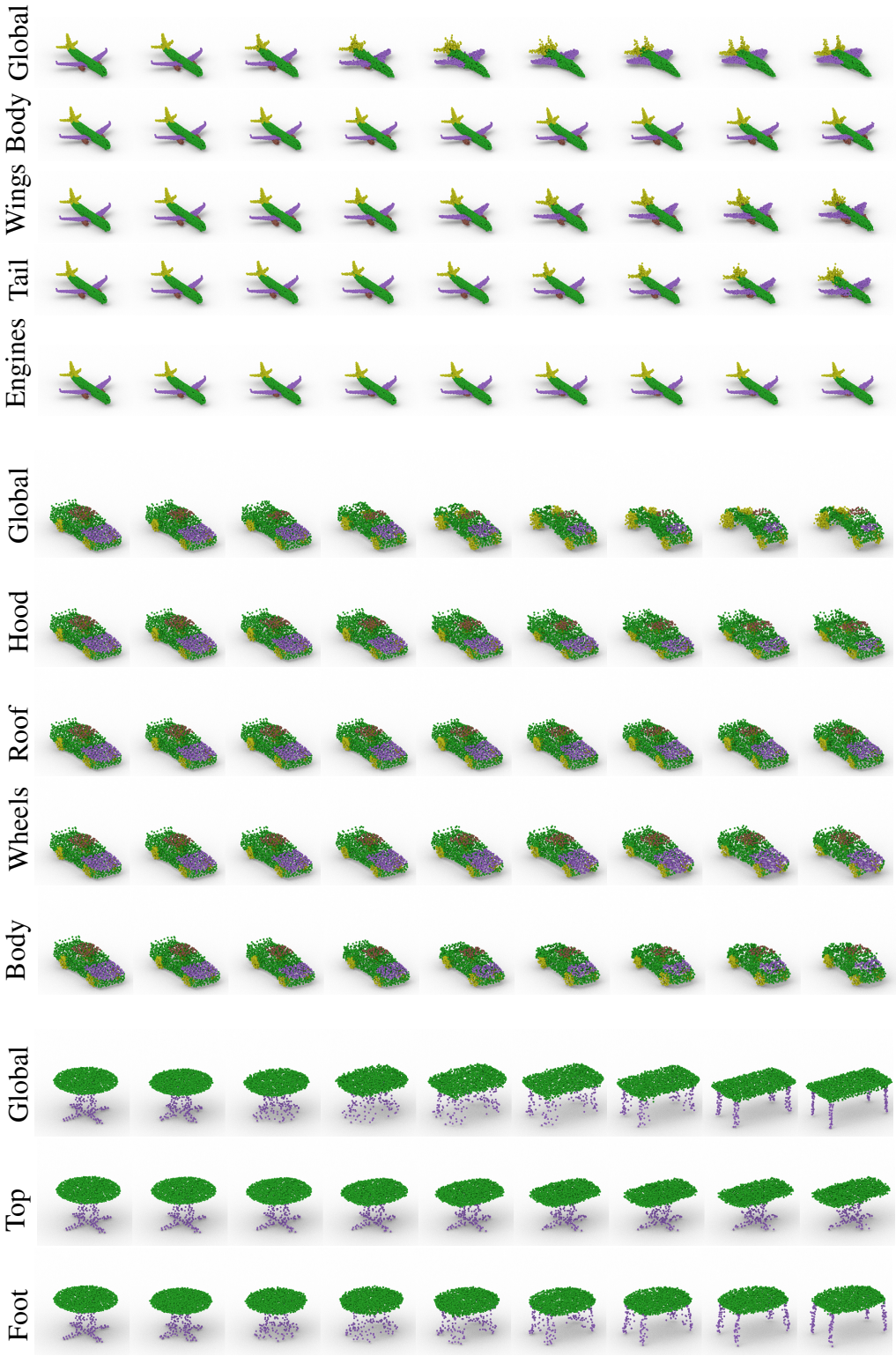


Figure 6.5: Part interpolation results for plane, car and table classes.

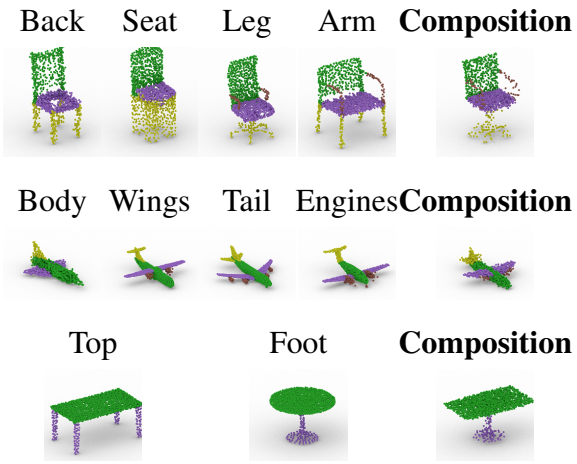


Figure 6.6: Part features from different samples are combined together to form a new shape. Parts preserve semantic properties while fitting to the new shape.

is generated by forming the selected parts together without any need for an assembling step to form the parts together with affine transformations like scaling, rotating and positioning. As explained in the previous section, the parts are not exactly the same as the source shapes since they are modified for better connection and harmony while keeping the semantic properties.

6.4.4 New Part and Shape Generation

The method is integrated with generative models as explained in Section 5.4. Latent-space GAN [2] architecture uses part features as input and output. Generator is a 3-layer Fully Connected Network (128, l, $k \times l$) for k parts and the Discriminator mirrors the Generator. Generator input is a 32-dimensional vector sampled from a Normal distribution. GAN has been trained using Adam optimizer with a first-moment value of 0.5 and learning rates of 5×10^{-4} and 1×10^{-4} for Generator and Discriminator respectively. GAN has been trained with the pretrained model to extract and decode features. WGAN follows the same architecture with WGAN objective function.

VAE architecture follows the base model with an exception of fully connected sampling layers to generate mean and sigma values. Regularization term has been normalized with input dimension and β parameter has been set to 0.1 since it provides a

Figure 6.7: Samples from generative models. VAE provides good reconstruction and generation capabilities. While standard GAN is able to generate good results, it suffers from lack of diversity. WGAN generates more diverse results.

good balance between reconstruction and generation quality. Reparametrization trick has been employed and the system has been trained using Adam optimizer [26] with a learning rate of 10^{-3} for 10000 epochs. For new data generation, latent codes have been sampled from a Normal distribution. Generated samples can be seen in Fig. 6.7 for *chair* class and Fig. 6.8 for *plane*, *car* and *table* classes.

Instead of generating whole global shapes, we can also generate parts for existing shapes. A new part or multiple new parts can be generated for existing shapes. The generative models can be trained to generate specific parts or existing parts can be replaced with the parts from the generated new shapes. Fig. 6.9 shows the results for generating a new specific part for an existing shape. The results demonstrate diverse part generations that can be used to modify existing shapes.

6.4.5 Robustness Against Different Input Sizes, Orientations and Scales

An object can be defined by a point cloud with different number of points. So, the method is expected to have the ability to process different input point cloud sizes (resolutions) and give similar outputs. The global feature is defined by the critical points, which are the most important points, in a point cloud. The critical point set is the minimum number of points defining the shape. Considering the critical points assumption, the proposed method is expected to extract the same feature set for a

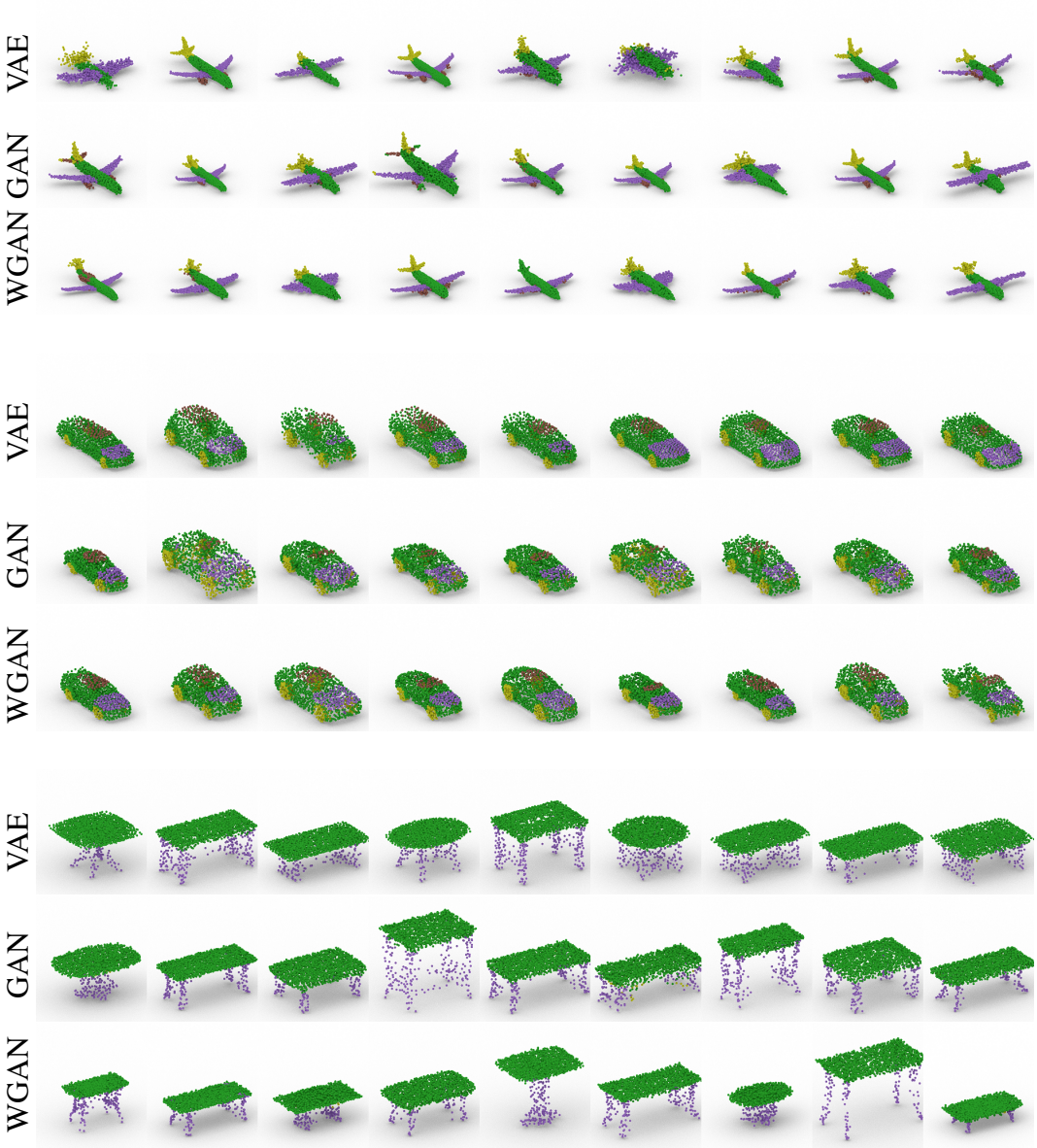


Figure 6.8: Samples from generative models for plane, car and table classes.

Figure 6.9: Samples from part exchange and generation for an existing model (most left).

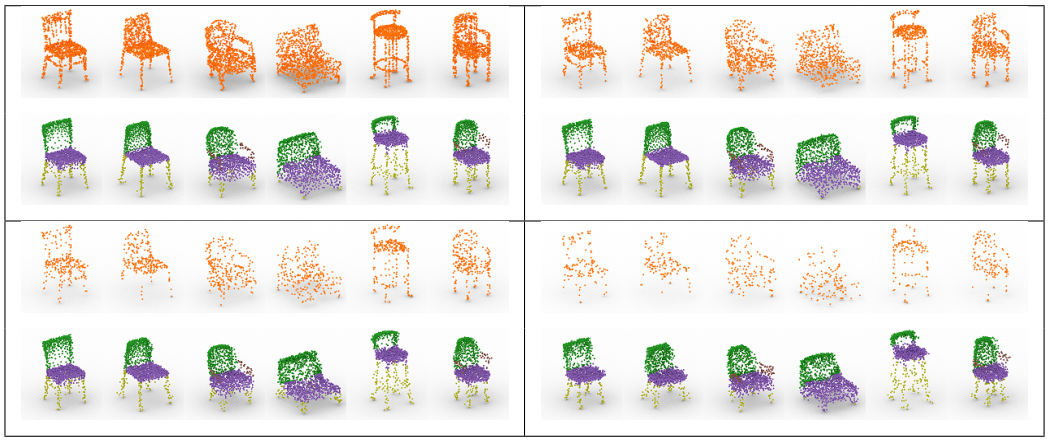


Figure 6.10: Reconstruction results from 1024 (top-left), 512 (top-right), 256 (bottom-left) and 128 (bottom-right) points to 2048 points.

shape defined with different number of points. These features can then be decoded to reconstruct the shape at any size. To test this, the original input has been randomly downsampled to 1024, 512, 256 and 128 points from 2048 points. Then these samples have been zero-padded to obtain 2048 points and the zero-padded points have been labeled as part 0. Then, these samples have been fed into the pretrained network to reconstruct the shape. Since the network ignores part 0 for feature extraction, it extracts the same features for all input dimensions. The results in Fig. 6.10 shows that the system can handle different input dimensions by giving the same features for the same shapes. The results are not affected by the lack of zero-padded samples during training. Also, this approach can serve as an upsampling network without training from scratch. It has to be noted that a lower number of input points result in poorer reconstructions since some critical points vanish due to random downsampling. Removing batch normalization layers improves robustness with more independent point features.

The proposed method is tested with different orientations of the objects. For that purpose, the objects in the train and the validation sets are rotated randomly on-the-fly in given angles for given axes. This method is previously used as a data augmentation strategy in [48] and [2] for single axis but the effect on the results are not reported. The reconstruction results can be seen in Fig. 6.11 for single axis (gravity axis) and Fig. 6.12 for multiple axes. The reconstruction losses can be found in Table 6.1. The system is robust against the rotations in single axis and it even helps to avoid overfitting. However, the robustness significantly drops when the objects are rotated in more than one axes as can ben seen in Fig. 6.12. This is because of the complex nature of 3D data when the object rotations are completely random. This randomness adds extra difficulty and uncertainty to crate learnable patterns. Complete rotation invariancy is another research field and is out of scope of this thesis. However, to achieve it, point projection operation [56] or rotation invariant local descriptors [10] can be integrated to the proposed method. Also rotation invariant 3D convolutions [69] can be used for feature extraction.

The system is trained with different scales of object to see the robustness against different scales. The objects are scaled randomly in [0.5, 2.0] interval on-the-fly to make them larger or smaller 2 times of the original scale. The reconstruction results

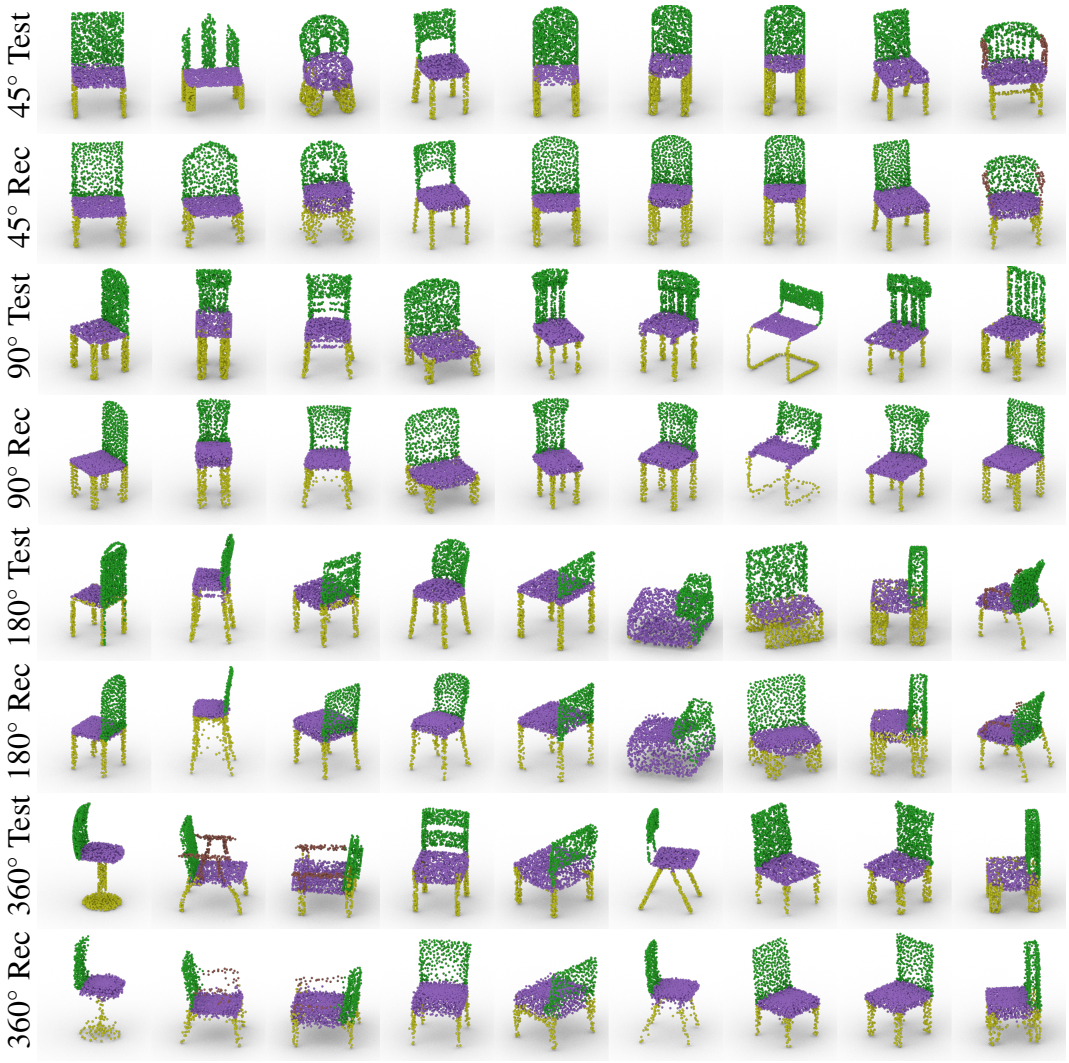


Figure 6.11: Reconstruction results of training with different random orientations for single axis.

Table 6.1: The reconstruction loss (Chamfer) and segmentation accuracy for inputs with different rotations and scales.

	Rec. loss ($\times 10^{-4}$)		Seg. acc. %	
	Train	Test	Train	Test
Base Model	3.61	5.93	96.23	93.51
Single axis				
45 degrees	3.96	5.89	96.25	93.79
90 degrees	4.60	6.06	95.75	93.58
180 degrees	5.61	6.41	95.44	93.36
360 degrees	5.71	6.63	95.26	93.29
360 degrees				
2 axes	12.84	12.71	89.96	91.35
3 axes	12.92	13.12	89.95	90.87
Scale	8.46	10.24	96.36	93.82

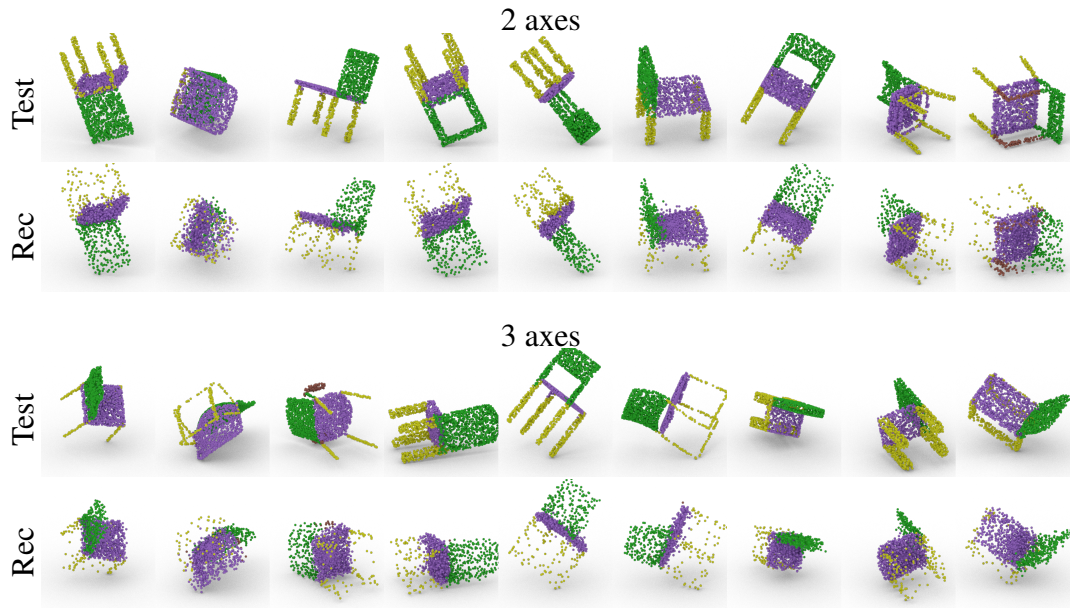


Figure 6.12: Reconstruction results of training with random orientations in 360° for 2 and 3 axes.

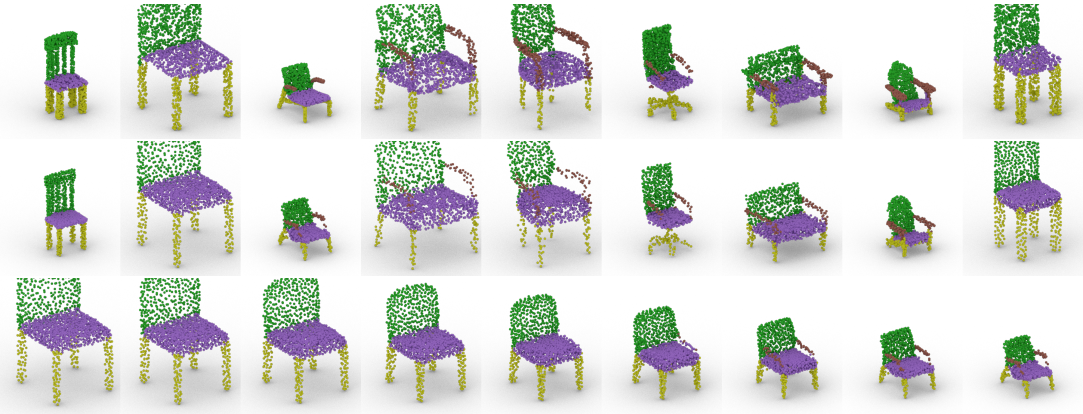


Figure 6.13: Reconstruction results for different scales. First row is the test set and the second is the reconstruction results. Third row is the latent interpolation result from the second to the third sample.

can be seen in Fig. 6.13 and the losses can be found in Table 6.1. The first row is the test set and the second row is corresponding the reconstruction results. The visual results provide a good robustness for scales in different sizes. Also, a latent interpolation can be seen in the figure. The system achieves a smooth interpolation between 2 different scales, proving a good representation of the scale information.

6.5 Evaluation of Generative Capabilities

For the evaluation of generative models, we have used the following evaluation metrics:

- **Coverage (Cov)** measures the representation of a point cloud set S_2 in set S_1. It is the fraction of point clouds in one set that is matched to others by finding the nearest neighbor.
- **Minimum Matching Distance (MMD)** is the average of distances between the matched point clouds in different sets.
- **Jensen–Shannon Divergence (JSD)** is the distance between 2 probability distributions, it is derived from Kullback–Leibler divergence [29]. In this scope, it is used as a measure of occupation of similar locations in 3D coordinate space

between two point cloud sets.

- **Total Mutual Difference (TMD)** [64] is used to measure the diversity of the generated shapes when one or more parts are changed. It is calculated by finding the average Chamfer distance of all generated shapes for a given input shape with missing parts.

MMD and Cov have been calculated using both CD and EMD. A higher score is better for Coverage and TMD and a lower score is better for MMD and JSD.

New samples are generated by five different approaches:

- *part feature exchange*: randomly exchanging part features between different samples,
- *part feature composition*: composing new shapes by combining different part features from different random samples,
- *VAE*: new shapes are generated by sampling from a Normal distribution using VAE,
- *GAN*: GAN is used after training to randomly generate new shapes,
- *WGAN*: WGAN is used instead of GAN for more diversity and more stable training.

To see the effect of different training and evaluation distance metrics, all models are both trained and evaluated with CD and EMD distances. A sample set is formed after generating new samples with each approach, which is 3 times the size of the reference test set. Results can be seen in Tables 6.2, 6.3 and 6.4 for *chair, table* and *plane* classes respectively. As expected, the results are in favor of the models trained with the same distance metric as the evaluation method. Part exchange has the lowest distance score with a high coverage. This is expected since only a single part per sample is different from the reference test set so there are lower distances between reference set and the generated set. Also, high coverage supports the similarity between the test set and the part-exchange set. The random part composition approach exhibits good diversity and novelty comparable with the generative models since random parts from random

shapes form a diverse set of novel samples. GAN implementation exhibits overfitting and collapses to a single mode especially when trained with EMD distance. WGAN achieves better diversity as expected with better coverage scores than GAN. VAE performs similar to WGAN indicating good sampling capability besides reconstruction. *Plane* class has lower MMD and JSD distance scores than other classes since the plane models are smaller, more dense, less diverse and occupy less area. The results show that different alternatives are successful at different aspects and they may serve different tasks better depending on the quality, diversity or complexity requirements of a particular task.

TMD is calculated by generating 10 samples for each shape by changing one or more parts while keeping the other parts the same. TMD results for the *chair* class are reported in Table 6.5, and sample visualizations are provided in Fig. 6.9. As expected, for all models, TMD score gets higher when more number of parts are generated. The exchange approach performs the best since it exchanges the parts with the already existing ones in the dataset. Other methods generate new parts from scratch, thus showing less diversity. TMD results for *table* and *plane* classes can be found in Table 6.6. *Table* class has higher TMD scores relative to *chair* class since it has only 2 parts; top and leg. Changing a part means half the parts of the model changes and the completion causes higher diversity. *Plane* class has lower TMD scores, implying less diversity. This is expected because of the attributes of the models in the *plane* class as explained in the previous paragraph.

6.6 Comparison with Related Works

The results of the proposed work and related works are provided in Table 6.2. CompoNet [52] is a part-assembly based approach. It has separate Autoencoders trained with CD for different parts. The individually generated parts are then brought together by a part-assembly network. The results show that, the proposed method outperforms CompoNet in all cases. In the baseline study [2], there are different generative models trained with different distance metrics. The best results (l-WGAN trained with EMD) is selected for the comparison. As expected, the proposed method has similar performance with the baseline method, since both these methods are holistic approaches

Table 6.2: Evaluation of generative models for *chair* class based on Minimum Matching Distance (MMD), Coverage (Cov), and Jensen-Shannon Divergence (JSD$\times 10^{-2}$). Both CD $(\times 10^{-4})$ and EMD $(\times 10^{-2})$ metrics are used for evaluation. CompoNet[52] is the part-assembly based approach. Achlioptas et al. is the best generative method (l-WGAN) reported in the study [2]. Tree-GAN results are reported in [53]. The best results, among only the generative models, are marked in bold.

	MMD		% Cov		
Model	CD	EMD	CD	EMD	JSD
Trained with CD					
Exc.	14.39	9.53	72.65	32.03	4.88
Comp.	17.50	9.86	56.25	25.78	5.58
VAE	**14.77**	10.24	**69.53**	28.12	6.74
GAN	22.41	10.39	34.37	19.53	8.97
WGAN	15.76	**9.64**	52.34	21.87	**5.88**
CompoNet [52]	40.63	10.11	28.90	**32.03**	7.65
Tree-GAN [53]	16.00	10.10	58.00	30.00	11.90
Trained with EMD					
Exc.	18.10	6.64	71.09	76.56	1.66
Comp.	22.14	7.32	56.25	61.71	2.02
VAE	23.87	7.84	55.47	**67.19**	4.28
GAN	34.48	8.99	23.43	24.21	6.41
WGAN	23.11	7.44	56.25	60.93	3.01
Ach. et al. [2]	**21.95**	**7.06**	**70.31**	66.4	**2.74**

Table 6.3: Evaluation of generative models for *table* class based on Minimum Matching Distance (MMD), Coverage (Cov), and Jensen-Shannon Divergence (JSD$\times 10^{-2}$).

	MMD		% Cov		
Model	CD	EMD	CD	EMD	JSD
Trained with CD					
Exc.	13.45	7.69	70.31	34.37	3.13
Comp.	15.77	7.83	67.19	32.03	3.81
VAE	**13.62**	7.96	**71.87**	**40.62**	**3.40**
GAN	33.38	9.94	21.09	14.84	8.00
WGAN	16.40	**7.95**	60.15	35.16	4.93
CompoNet [52]	87.07	14.14	30.46	14.85	22.99
Tree-GAN [53]	18.00	10.70	66.00	39.00	10.05
Trained with EMD					
Exc.	17.01	5.94	75.00	78.12	1.99
Comp.	19.41	6.48	70.31	72.65	2.46
VAE	23.58	7.23	50.78	60.15	4.32
GAN	32.87	8.34	31.25	38.28	6.10
WGAN	**20.71**	6.79	66.40	71.87	3.32
Ach. et al. [2]	20.75	**6.64**	**69.53**	**73.43**	**2.76**

Table 6.4: Evaluation of generative models for *plane* class based on Minimum Matching Distance (MMD), Coverage (Cov), and Jensen-Shannon Divergence (JSD$\times 10^{-2}$).

	MMD		% Cov		
Model	CD	EMD	CD	EMD	JSD
Trained with CD					
Exc.	3.90	5.85	69.53	14.06	3.73
Comp.	4.40	5.99	60.93	11.71	4.18
VAE	**3.43**	6.41	59.59	14.84	5.64
GAN	6.39	6.31	24.21	7.81	5.98
WGAN	4.76	**5.76**	60.93	15.62	**4.17**
CompoNet [52]	20.02	8.41	19.53	16.4	17.83
Tree-GAN [53]	4.00	6.80	**61.00**	**20.00**	9.70
Trained with EMD					
Exc.	4.45	3.80	72.65	67.18	2.05
Comp.	5.41	4.21	59.37	53.12	2.74
VAE	**5.29**	**4.14**	57.04	53.12	3.54
GAN	6.23	4.61	42.96	35.15	3.51
WGAN	6.03	4.31	**57.07**	52.34	**2.62**
Ach. et al. [2]	6.49	4.21	57.03	**60.93**	3.25

Table 6.5: Total Mutual Difference (TMD$\times 10^{-2}$) [64] scores of part exchange and generation for *chair* class. One or more parts are changed by keeping the others the same.

	# of changing parts			
Model	1	2	3	4
Exchange	1.31	3.47	4.66	4.85
VAE	1.06	2.54	3.33	**3.54**
l-GAN	0.79	1.96	2.41	2.60
l-WGAN	1.22	2.53	**3.38**	3.48
Wu et al. [64]	**2.28**	**2.81**	2.96	3.19

Table 6.6: Total Mutual Difference (TMD$\times 10^{-2}$) scores for *table* (left) and *plane* (right) classes.

	# of changing parts		# of changing parts			
Model	1	2	1	2	3	4
Exchange	5.27	9.89	0.23	1.00	1.11	1.15
VAE	3.31	6.02	0.21	0.64	0.69	0.73
l-GAN	3.27	4.64	0.13	0.28	0.33	0.36
l-WGAN	3.57	6.95	0.21	0.69	0.77	0.80

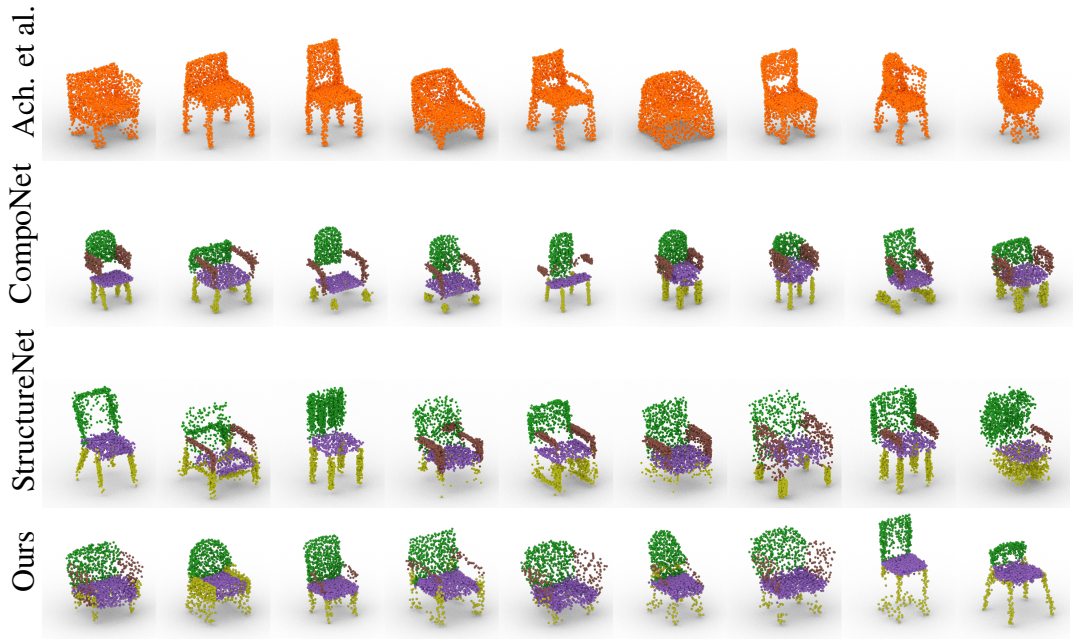


Figure 6.14: Randomly generated samples by different methods; Achlioptas et al. [2], CompoNet [52], StructureNet [41] and our model. Achlioptas et al. considers only the global shape, CompoNet has difficulty assembling and connecting the generated parts. StructureNet can generate more diverse structures but suffers from structural noise causing implausible structures.

that become equivalent for global shape generation. However, the proposed method has additional part-based capabilities as mentioned above. Tree-GAN [53] has comparable results with the other generative models. However, it cannot be evaluated with regards to part exchange and composition performance as it lacks reconstruction abilities. Its MMD and Coverage results are inferior for *chair* and *table* classes. While it has better results for Coverage of *plane* class, the difference is only marginal. StructureNet uses a fine-grained, hierarchical dataset for structure encoding, hence its results cannot be evaluated on the dataset used in these experiments. The comparison with StructureNet on a different dataset is given in Section 6.7.

The qualitative results can be seen in Fig. 6.14. The proposed method and the baseline (Achlioptas et al. [2]) show similar generation quality and diversity. However, the baseline method does not have any part information and only considers global shapes. The part-assembly based CompoNet [52] is able to generate parts separately, but it has difficulty assembling and connecting the generated parts. Although part generation of this method is satisfactory, the part-assembly step generates incoherent global shapes, which fail to exhibit seamless connection between parts. There are implausible shapes because of missing connections and floating parts. Also, points are not distributed evenly across the global shape as there are fixed number of points per part. By using a part-based holistic approach,

- the proposed method can handle separate parts, which is a capability lacking in [2],
- it also generates a complete coherent global shape in unison while handling separate parts which is different to the two stage approach in [52].

This eliminates the need for a separate part-assembly network and potential problems associated with part-assembly. StructureNet [41] (results are downsampled to the same number of points for fair comparison) generates diverse structures including asymmetric ones. However, the generated samples suffer from structural noise causing implausible shapes. Also, representing all parts with the same number of points leads to better quality for small parts than large parts, especially becoming evident in low resolutions.

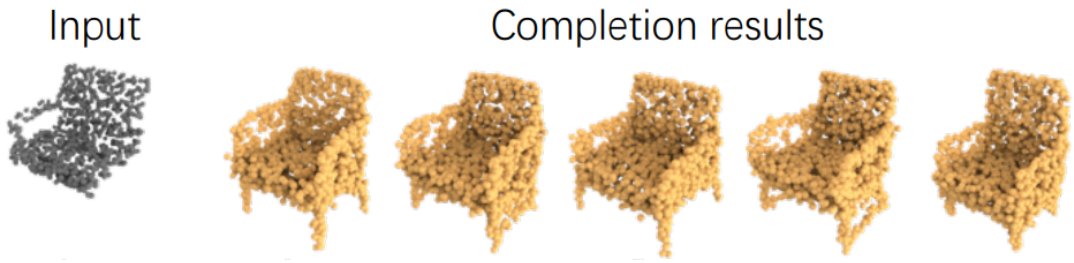


Figure 6.15: Leg completion results from Wu et al. [64]. It modifies the irrelevant parts while completing the missing part.

For the evaluation of shape completion capability, Total Mutual Difference (TMD) results are reported in Table 6.5 by regenerating one or more parts. Wu et al. [64] is a shape completion network which completes the partial shapes with missing parts by generating multiple outputs. The proposed method has lower scores for few missing parts, but exhibits higher scores when there are higher number of missing parts. However, it has to be noted that TMD evaluates the diversity of the whole shape and not only the generated part. Shape completion network is a generative model that generates a new shape from scratch to complement input shape. In contrast to our reconstruction model, it generates different outputs for even complete shapes. While completing a shape with a new part, the method [64] also causes changes in the irrelevant parts of the shape as seen in Fig. 6.15. This results in an increase in TMD score. This observation is supported by the low TMD score variance with respect to the different number of missing parts for [64].

6.7 Supplementary Comparisons

To allow comparisons with StructureNet, we conducted a separate experiment, by training our method on their dataset. StructureNet [41] is designed to work on a fine-grained hierarchically labeled dataset with child parts such as PartNet [43]. This dataset structure is fundamentally different to the one we used in this work. So, in order to facilitate comparisons, we have also trained our model with the PartNet where each part and child parts have 1000 points. We have grouped all the child parts into the same semantic definitions as we used such as seat, back, leg and arm. Both models have been trained and evaluated using CD. The results are reported for chair

Figure 6.16: Hierarchical structure of PartNet [43] dataset [41].

class, which has 4871 samples divided with 7:1:2 ratio for training, validation and test respectively, with 2048 points per sample.

The average reconstruction error (Chamfer$\times 10^{-4}$) for the global shapes is calculated as 30.18 for StructureNet and 12.11 for our model. The results are consistent with the results reported in [41]. The reconstruction results are similar for common cases but the results for challenging cases can be seen in Fig. 6.17. Our results become noisy but represent the global shape better. The noise in StructureNet appears as structural inaccuracies since it makes structural encoding-decoding. It is also reported that noise in StructureNet may result in missing parts, duplicate parts, detached parts [41]. Considering both quantitative and qualitative comparison, the proposed model performs better at global shape reconstruction. StructureNet generates novel structures and parts using VAE. We compared the new sample generation capabilities of both models with the evaluation metrics we used. The results provided in Table 6.7

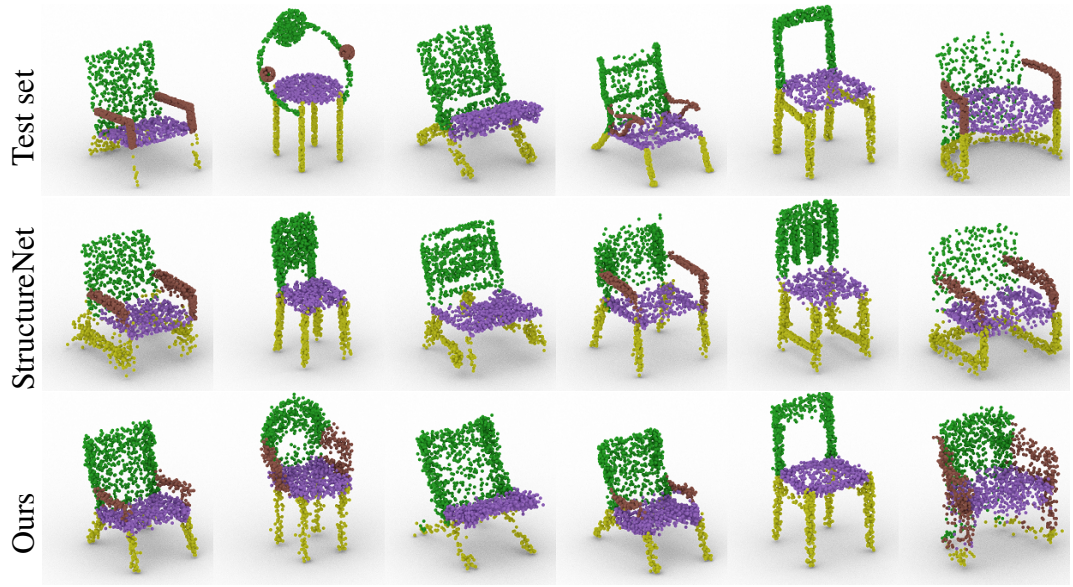


Figure 6.17: Reconstruction results of challenging cases for StructureNet[41] and our model.

Table 6.7: Comparison with StructureNet on the PartNet [43]

Model	MMD	% Cov	JSD
VAE	**17.27**	**67.96**	20.61
GAN	30.74	24.21	19.71
WGAN	21.93	62.50	**10.34**
StructureNet [41]	27.14	39.06	17.98

show that the proposed model has better MMD, Coverage and JSD scores.

Visual interpolation results for different methods are provided in Fig. 6.18. As StructureNet performs structure interpolation by its nature, it causes sharp structural changes in middle steps. CompoNet performs per-part interpolation, however it suffers from part assembly problems in some steps. In both cases, the proposed model performs a smooth global shape interpolation, generating plausible global shapes during the transition steps.

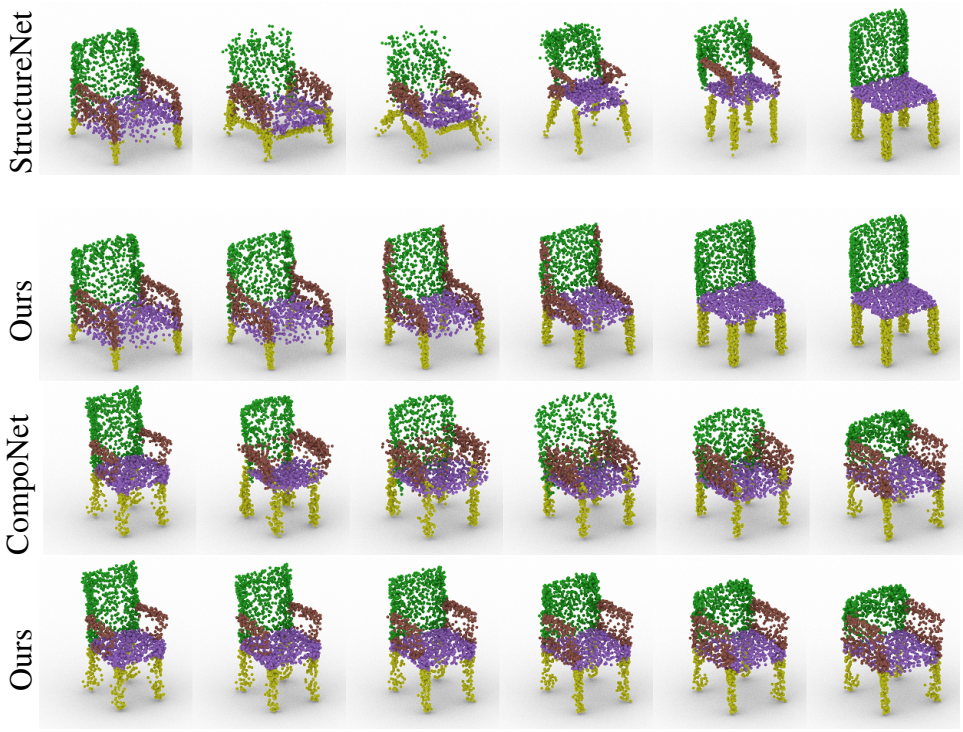


Figure 6.18: Interpolation comparison with StructureNet [41] and CompoNet [52] between the same two shapes (leftmost and rightmost).

Table 6.8: The effect of different variations of feature extractor and segmentation module based on reconstruction loss (Chamfer) and segmentation accuracy.

	Rec. loss ($\times 10^{-4}$)		Seg. acc. %	
	Train	Test	Train	Test
Base Model	3.61	5.93	96.23	93.51
Feature Extractor				
PointNet [48]	3.92	6.06	96.35	93.84
Mean pooling	5.48	7.01	96.18	93.61
Segmentation module				
No module	3.01	5.20	-	-
Module failure	3.11	5.95	-	-
No global features	4.24	6.04	87.47	86.95

6.8 Ablation Study

The proposed framework allows replacement of the Feature extractor and Segmentation modules. Table 6.8 summarizes the reconstruction and segmentation performance by (*i*) substituting feature extraction with PointNet while keeping the other modules the same and changing its pooling layer with mean pooling; (*ii*) experimenting on segmentation module by removing it, using a sub-optimal segmentation module and using a segmentation module omitting the global features. All variations are trained with the same parameters.

Replacing the feature extraction with PointNet does not provide any benefits since the samples are already aligned and the system works with a single class. Since the input data has a single class and limited diversity, a 3-layer model is sufficient for extracting the necessary features and using 5-layers does not provide any advantage. Replacing the max-pooling with mean-pooling, which is also a symmetric operation, degrades the results. Mean-pooling extracts the average of features rather than selecting the most effective and critical features like max-pooling and as a result, it represents an average model with smooth edges which can be seen in Fig. 6.19.

To observe the effect of the segmentation module, we trained the system without it and fed the ground truth part labels. Since the part labels are not predictions but ground truths, the reconstruction performance was better as expected. On the other hand, elimination of segmentation module results in an undesirable effect of eliminating the ability of the system to work with unannotated raw point clouds. We also deliberately hindered the training of segmentation module and randomly initialized the module to simulate segmentation failures where segmentation results are random. Interestingly, it is quantitatively better than the base model because now each point is randomly assigned to different parts, thus each part is simply a downsampled version of the global model. All part features are equal to global feature so the system captures the global features better. However, in this case, the system does not have any part-based abilities anymore and not fit for purpose since all parts are equivalent to global model. Lastly, the segmentation module in the base system is trained with only point features (without concatenating global features). Lower segmentation performance highlights the importance of global features -alongside point features- in

Figure 6.19: Reconstruction comparison between models using max and mean pooling.

the segmentation performance.

6.9 Failure Cases

The samples with high reconstruction losses are visualized to analyze failure cases in Fig. 6.20 where the test samples are shown at the top row and their respective reconstructions at the bottom row. The unusual object samples in Fig. 6.20 (a)-(d) are outliers and their respective reconstructions are noisy. Chairs in Fig. 6.20 (a) and (b) have arms in the middle, this is not a common occurrence in the training set and these arms could not be represented. Unusual leg shapes of chairs in Fig. 6.20 (c) and (d) can not be reconstructed well, resulting in high reconstruction loss. Reconstructed part labels are different for Fig. 6.20 (e) due to segmentation error. However, the reconstructed shape is still acceptable because leg and back parts are ambiguously defined. Chairs in Fig. 6.20 (f) and (g) have highly asymmetric shapes. Asymmetric shapes comprise less than 3% of the whole dataset, so the system can not adequately

(a) (b) (c) (d) (e) (f) (g) (h)

Figure 6.20: The reconstruction results of unusual, asymmetric or failure cases.

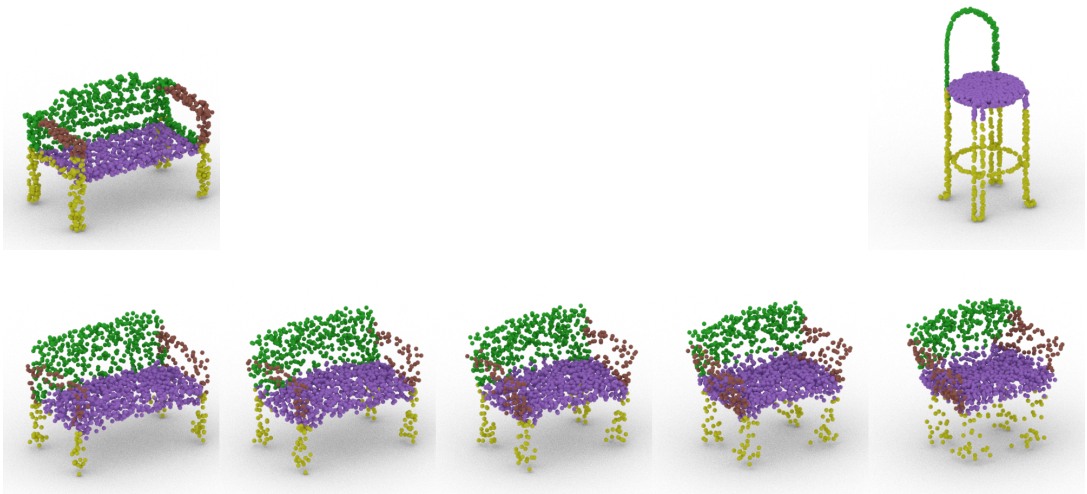


Figure 6.21: A challenging part (leg) interpolation between two distant shapes.

learn to represent them. This can be prevented by augmenting the dataset with further asymmetric samples. Fig. 6.20 (h) has an unusual hole on the seat part, again not present in the training set. All of the reconstruction errors are because of the lack of representative samples in the training set and can be prevented by extending the dataset with more diverse samples.

Interpolation between distant shapes such as the ones in Fig. 6.21 may not be successful. On the other hand, it can be argued that, this operation is hardly plausible for humans as well. The target leg on the right is not easy to seamlessly merge into the original global shape on the left and the model does its best to modify the source shape and leg to generate a semantically acceptable global shape. This supports our claim that the system makes semantic modifications. However, it cannot be loyal to the original shapes for this case as this would interfere with generating a semantically

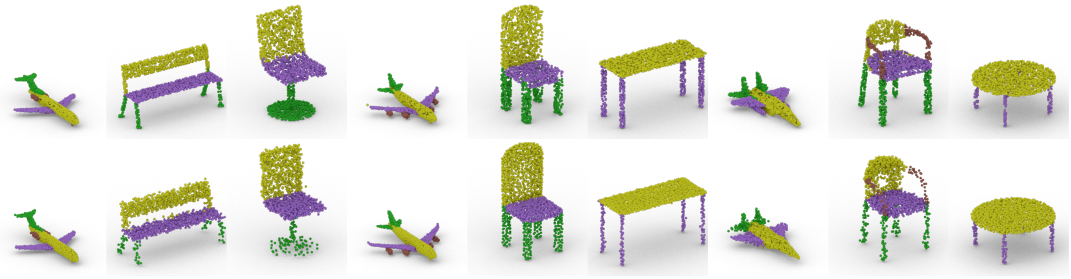


Figure 6.22: Random reconstruction results of multi-class training.

acceptable global shape.

6.10 Multi-class Training

The system is trained with all classes together to investigate the ability of learning different classes together. All models in different classes are fed together with overlapping and non-overlapping part numbers. Overlapping part numbers use the same part number for all classes where a part 0 may mean leg for chair, top for table and wing for plane. Non-overlapping part numbers uses unique part numbers for different classes where a part 0 may mean only leg part for chair class. In both cases, the results are very similar, showing a good capacity to learn different classes together.

Reconstruction results of multi-class training can be seen in Fig. 6.22. It shows a good reconstruction quality for all classes. Also, reconstruction of multi-class training provides overfitting. For the same number of training data, the reconstruction loss for single class training is 2.9×10^{-4} for training and 5.8×10^{-4} for validation, and segmentation accuracy is 0.97 for training and 0.93 for validation. Multi-class training loss is 3.2×10^{-4} and validation loss is 4.9×10^{-4}. The segmentation accuracies are 0.96 and 0.95 for training and validation respectively. Since the training data is more diverse in multi-class training, less overfitting and less difference between training and validation results are already expected. However, part based operations (interpolation, exchange, composition) are still only possible in the same class.

The extracted global features are visualized using t-SNE [60] visualization in Fig. 6.23. The raw point clouds (left) and the extracted features (right) are both visualized

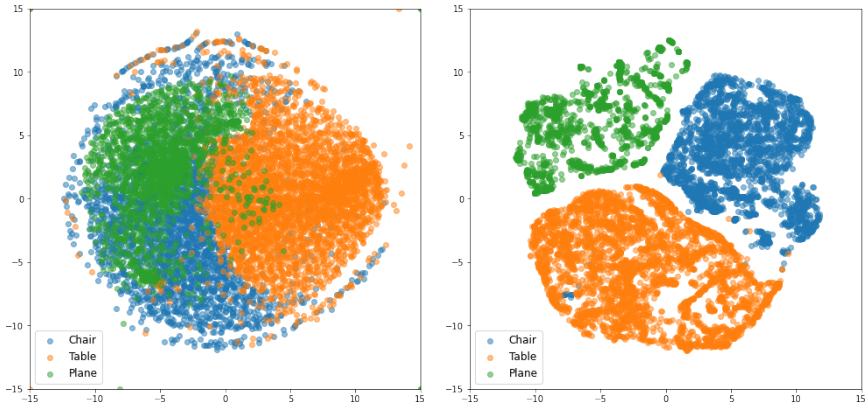


Figure 6.23: t-SNE visualization of raw point clouds (left) and extracted global features (right) of multi-class training.

to see the ability of the system if it can create a distinguishable latent representations for different classes. While different classes are not clearly distinguishable with raw point clouds, the system is able to represent each class separately. The proposed method has the capacity to learn different classes together by representing them on different areas in the latent space.

6.11 Continuous Conditions

After training the system with different classes together, we investigated the possibility of conditional generation of different classes. Also, we wanted to see the possibility of controlling the common features of different classes. For this purpose, we have implemented a GAN with InfoGAN [9] approach. InfoGAN is able to learn both discrete and continuous conditions together in an unsupervised manner as explained in Section 2.2.1. We expect that system can generate different classes with given discrete conditions and extracts common visible features with continuous conditions.

The implemented system works on raw point clouds with a resolution of 1024. The Generator is similar to proposed Decoder architecture with (64, 128, 256, 512, 1024$\times$3) fully connected layers. All layers are followed by ReLU activation function. The Discriminator is similar to proposed Encoder architecture with 5-layer MLP (64, 128, 256, 256, 512) with weight sharing. It is followed by a global max-pooling for ex-

tracting global features and 3 fully connected layers (512, 128, 1) for Discriminator output. All layers are followed by leaky ReLU activation function. Q networks are implemented for discrete and continuous conditions with 3 fully connected layers (512, 128, C) after shared MLP module. There are one discrete condition for generating different classes and two continuous conditions for learning visible continuous features. The system is trained with WGAN loss function for GAN networks, Cross Entropy loss for discrete condition and MSE loss for continuous condition.

The results can be seen in Fig. 6.24 for different conditions. Without any supervision and labeled data, the system behaves as expected for discrete condition D and continuous conditions C_1 and C_2. The generator is fed with concatenated random vector, D, C_1 and C_2 to generate the models in given conditions. The discrete condition controls the class and provides generation of different classes (*chair, table and plane*) for different values. While the meaning of discrete condition is obvious, the continuous conditions can not be identified easily. From the results we can see that the first continuous condition C_1 represents how tall the model is, and the second one C_2 represents how wide the model is.

In Fig. 6.24, each line of results is generated with given conditions at the top of the line. System generates completely random results for random classes if all conditions are random. If we just set the discrete condition D as 0,1 and 2; it generates random models in *chair, table and plane* classes respectively. By this way we control the class of the generated model. To observe the effect of a continuous condition, we interpolate one and keep the other as same. By interpolating C_1, we can see that chairs and tables become tall or short models. We can not observe the same effect on planes since the height of the planes are very similar in the dataset so C_1 has not a significant effect for plane class. The models become more wide or narrow when we interpolate the second continuous condition C_2. Tables interpolate from rectangle to square and circular shapes. Plane wings become more perpendicular to body, making the model wider.

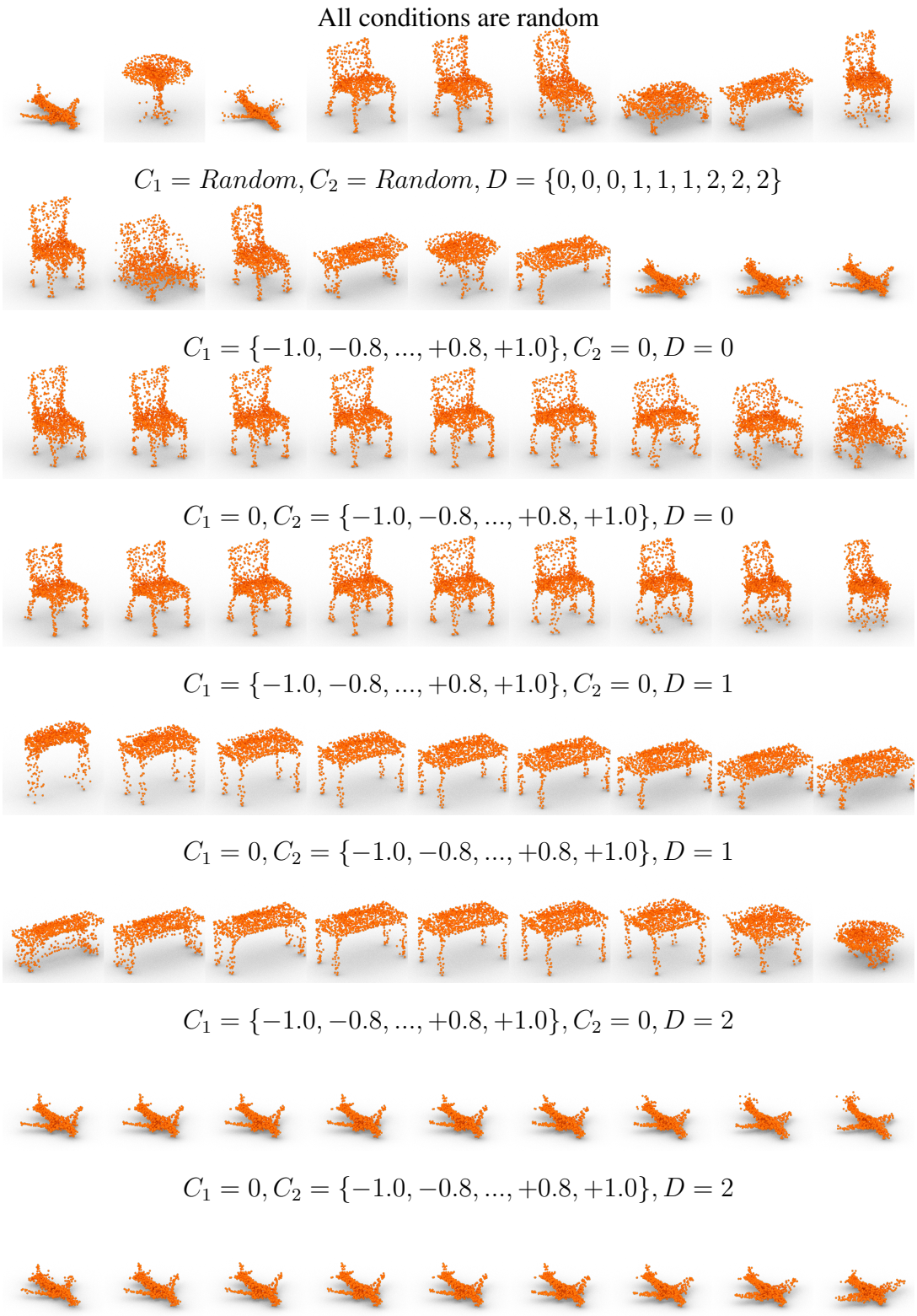


Figure 6.24: The results of InfoGAN for different discrete D and continuous C_1, C_2 conditions.

6.12 Deployment

We have deployed the trained model as an interactive web page. Since the models are 3D, we needed a platform with 3D rendering ability and interactive controls for rotating, relocating and viewing the models with different camera angles. The standard Neural Network deployment platforms do not have these abilities so we used Unity 3D Engine [59]. The trained model is first converted to ONNX format. ONNX (Open Neural Network Exchange) [5] is an open source format built to represent machine learning models. It aims to create a common format for deploying machine learning models developed by frameworks like PyTorch, Tensorflow, Keras to production frameworks and hardwares. Then the ONNX file is imported into Unity with a user interface to make the users generate new models and interact with the generated models. The Unity project is built as a web application to deploy it as a web page. The web page is publicly available at `https://cihanongun.github.io` and any user with a supported web browser can use the trained model.

CHAPTER 7

CONCLUSIONS

In this thesis, a generic part-aware architecture allowing exchanging of parts between different models and generating new point cloud models and parts has been proposed. Unlike existing approaches in the literature, which need different neural network models for each part and an additional neural network model for assembling the generated parts, the proposed system handles part editing, modification and generation with a single architecture and eliminates the need for an additional network for part assembly. The system does not require any additional loss function other than the standard reconstruction loss functions in the literature. The exhaustive experiments demonstrated that the system is scalable and can be used with different point cloud sizes, rotations, scales; objects having different numbers of parts and parts having different resolutions. Also, with the integration of the segmentation module, it can process 3D models without any explicit part information and part labels during inference. It has been shown that GANs and VAEs can be integrated into the proposed method to generate new parts and 3D models. The system is used with continuous and discrete conditions to generate samples having determined properties.

We presented visual and quantitative comparisons with the methods having different approaches in the literature. The results show that our method has better results considering reconstruction and generation quality. The diversity of the generated samples are also comparable with the generative methods in the literature. We trained our system with different datasets and loss metrics for a fair comparison with state-of-the-art methods. Also, the ablation study shows the effects of various possible options which can be used in the system. When trained with multiple classes together, high-dimension visualization methods show that the system is able to learn

different feature spaces for different classes without any information about class labels. Lastly, we made an extensive failure analysis to investigate the possible reasons and solutions for the failures.

In the proposed method, while a part feature represents the corresponding part in a global shape, the decoder takes a global feature as input and outputs a global shape. While the method cannot reconstruct the parts separately, this is not considered to be a significant limitation as the ultimate aim in most applications is to form a global shape. To reconstruct the parts separately, the method must be trained with parts separately from scratch. Then, the global shape can be constructed from the parts by a composition model similar to those in the literature. Part modification and generation are complementary operations to get the global shapes.

7.1 Future Work

The proposed approach can be extended to work multi-class. There are different studies in the literature that aims to make part exchange between different classes. A single model can be trained with different classes at the same time and part features can be exchanged for this purpose. However, the idea of part exchange between different classes are confusing for even humans. How can we exchange parts between a plane and a chair? What do we expect by using a plane wing as a chair seat? How do we evaluate the success of such novel shapes? After answering these question, the proposed method can be modified to serve for intended multi-class purposes.

Image-to-3D is another popular field that aims to create 3D models from reference images. The proposed method has generative capabilities from reference features. These features can be extracted from images instead of 3D models. The same latent space, which the proposed method operates, can be used for both images and 3D models.

The proposed approach can be applied on different 3D representations such as voxels and meshes. While the same approach can be employed, different processing layers should be used for different data types. The proposed method uses 1D convolutions and fully connected layers to process 3D point clouds. Instead it can use 3D convo-

lutions for voxels and graph convolutions for meshes.

Since the proposed method operates on latent space, it represents the 3D models with semantic features. This provides understanding of 3D models without any dependency on specific classes and data types. The module-based architecture allows modification and integration of different modules for different purposes and achieving better performances. The proposed method has a huge potential for further extensions and improvements for different tasks consisting 3D models.

REFERENCES

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

[2] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017.

[3] E. Ahmed, A. Saint, A. E. R. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada, and B. Ottersten. A survey on deep learning advances on different 3d data representations. *arXiv preprint arXiv:1808.01462*, 2018.

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

[5] J. Bai, F. Lu, K. Zhang, et al. Onnx: Open neural network exchange. `https://github.com/onnx/onnx`, 2019.

[6] D. Berthelot, T. Schumm, and L. Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[7] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. 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.

[8] W.-C. Chen, C.-W. Chen, and M.-C. Hu. Syncgan: Synchronize the latent spaces of cross-modal generative adversarial networks. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2018.

[9] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: interpretable representation learning by information maximizing generative adversarial nets. In *Neural Information Processing Systems (NIPS)*, 2016.

[10] H. Deng, T. Birdal, and S. Ilic. Ppf-foldnet: Unsupervised learning of rotation invariant 3d local descriptors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 602–618, 2018.

[11] A. Dubrovina, F. Xia, P. Achlioptas, M. Shalah, R. Groscot, and L. J. Guibas. Composite shape modeling via latent space factorization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8140–8149, 2019.

[12] H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2463–2471, July 2017.

[13] W. Fedus, M. Rosca, B. Lakshminarayanan, A. M. Dai, S. Mohamed, and I. Goodfellow. Many paths to equilibrium: Gans do not need to decrease a divergence at every step. *arXiv preprint arXiv:1710.08446*, 2017.

[14] M. Gadelha, S. Maji, and R. Wang. 3d shape generation using spatially ordered point clouds. In *British Machine Vision Conference (BMVC)*, volume 3, 2017.

[15] L. Gao, J. Yang, T. Wu, Y.-J. Yuan, H. Fu, Y.-K. Lai, and H. Zhang. Sdm-net: Deep generative network for structured deformable mesh. *ACM Transactions on Graphics (TOG)*, 38(6):1–15, 2019.

[16] G. Gkioxari, J. Johnson, and J. Malik. Mesh r-cnn. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019.

[17] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair,

A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems (NIPS)*, pages 2672–2680, 2014.

[19] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[20] P. Hermosilla, T. Ritschel, P.-P. Vázquez, À. Vinacua, and T. Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics (TOG)*, 37(6):1–12, 2018.

[21] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations (ICLR)*, 2(5):6, 2017.

[22] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris. Deep learning advances in computer vision with 3d data: A survey. *ACM Computing Surveys (CSUR)*, 50(2):1–38, 2017.

[23] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[24] A. Karpathy, P. Abbeel, G. Brockman, P. Chen, V. Cheung, R. Duan, I. Goodfellow, D. Kingma, J. Ho, R. Houthooft, T. Salimans, J. Schulman, I. Sutskever, and W. Zaremba. Openai : Generative models. [Online]. Accessed: 16 June 2016, Available: https://blog.openai.com/generative-models/.

[25] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In *International Conference on Machine Learning*, pages 1857–1865. PMLR, 2017.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

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

[28] N. Kodali, J. Abernethy, J. Hays, and Z. Kira. On convergence and stability of gans. *arXiv preprint arXiv:1705.07215*, 2017.

[29] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[30] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[31] F. Li, A. Karpathy, and J. Johnson. Stanford university cs231n: Convolutional neural networks for visual recognition. [Online]. Accessed: 18 May 2017, Available: http://cs231n.stanford.edu/.

[32] J. Li, C. Niu, and K. Xu. Learning part generation and assembly for structure-aware shape synthesis. *arXiv preprint arXiv:1906.06693*, 2019.

[33] J. J. Lim, H. Pirsiavash, and A. Torralba. Parsing IKEA Objects: Fine Pose Estimation. *ICCV*, 2013.

[34] M.-Y. Liu and O. Tuzel. Coupled generative adversarial networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 469–477, 2016.

[35] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly. Are gans created equal? a large-scale study. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 698–707, 2018.

[36] X. Mao, Q. Li, and H. Xie. Aligngan: Learning to align cross-domain images with conditional generative adversarial networks. *arXiv preprint arXiv:1707.01400*, 2017.

[37] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[38] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

[39] H.-Y. Meng, L. Gao, Y.-K. Lai, and D. Manocha. Vv-net: Voxel vae net with group convolutions for point cloud segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8500–8508, 2019.

[40] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[41] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. J. Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *ACM Trans. Graph.*, 38:242:1–242:19, 2019.

[42] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. J. Mitra, and L. J. Guibas. Structedit: Learning structural shape variations. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2020.

[43] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[44] C. Öngün and A. Temizel. Paired 3d model generation with conditional generative adversarial networks. In *European Conference on Computer Vision*, pages 473–487. Springer, 2018.

[45] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[47] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2536–2544, 2016.

[48] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[49] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Neural Information Processing Systems*, 2017.

[50] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[51] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[52] N. Schor, O. Katzir, H. Zhang, and D. Cohen-Or. Componet: Learning to generate the unseen by part synthesis and composition. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[53] D. W. Shu, S. W. Park, and J. Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3859–3868, 2019.

[54] E. J. Smith and D. Meger. Improved adversarial systems for 3d object generation and reconstruction. In *Conference on Robot Learning*, pages 87–96. PMLR, 2017.

[55] E. J. Smith and D. Meger. Improved adversarial systems for 3d object generation and reconstruction. In *Conference on Robot Learning*, pages 87–96. PMLR, 2017.

[56] X. Sun, Z. Lian, and J. Xiao. Srinet: Learning strictly rotation-invariant representations for point cloud classification and segmentation. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 980–988, 2019.

[57] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6411–6420, 2019.

[58] V. Turchenko and A. Luczak. Creation of a deep convolutional auto-encoder in caffe. In *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 2, pages 651–659, 2017.

[59] Unity. Unity game engine. [Online]. Accessed: 01 March 2020, Available: https://unity3d.com/.

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

[61] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[62] H. Wang, N. Schor, R. Hu, H. Huang, D. Cohen-Or, and H. Huang. Global-to-local generative model for 3d shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA)*, 37(6):214:1—214:10, 2018.

[63] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[64] R. Wu, X. Chen, Y. Zhuang, and B. Chen. Multimodal shape completion via conditional generative adversarial networks. In *The European Conference on Computer Vision (ECCV)*, August 2020.

[65] W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[66] Y.-P. Xiao, Y.-K. Lai, F.-L. Zhang, C. Li, and L. Gao. A survey on deep geometry learning: From a representation perspective. *Computational Visual Media*, 6(2):113–133, 2020.

[67] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

[68] K. Yin, Z. Chen, S. Chaudhuri, M. Fisher, V. Kim, and H. Zhang. Coalesce: Component assembly by learning to synthesize connections. *arXiv preprint arXiv:2008.01936*, 2020.

[69] Z. Zhang, B.-S. Hua, D. W. Rosen, and S.-K. Yeung. Rotation invariant convolutions for 3d point clouds deep learning. In *2019 International Conference on 3D Vision (3DV)*, pages 204–213. IEEE, 2019.

[70] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[71] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

[72] C. Öngün and A. Temizel. Lpmnet: Latent part modification and generation for 3d point clouds. *Computers & Graphics*, 96:1–13, 2021.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Öngün, Cihan
Nationality: Turkish (TC)
Date and Place of Birth: 31 May 1989, Adana
Marital Status: Married
Phone: +90 506 584 80 10
email: cihanongun@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
M.Sc.	METU Electrical and Electronics Engineering	2014
B.Sc.	ESOGU Electrical and Electronics Engineering	2012

WORK EXPERIENCE

Year	Place	Enrollment
2014-2021	METU Graduate School of Informatics	Research Assistant
2012-2014	TUBITAK Project	Project Assistant

PUBLICATIONS

[1] C. Öngün and A. Temizel, "Paired 3D Model Generation with Conditional Generative Adversarial Networks", Proceedings of the European Conference on Computer Vision (ECCV) Workshops, pages 473–487. Springer, 2018, https://doi.org/10.1007/978-3-030-11009-3_29

[2] C. Öngün and A. Temizel, "LPMNet: Latent Part Modification and Generation For 3D Point Clouds" , Computers & Graphics, 96:1–13, 2021, https://doi.org/10.1016/j.cag.2021.02.006

[3] C. Öngün, A. Temizel and T. T. Temizel, "Clustering of local behaviour in crowd videos," 2014 22nd Signal Processing and Communications Applications Conference (SIU), 2014, pp. 818-821, doi: 10.1109/SIU.2014.6830355.

[4] H. Kumdakcı, C. Öngün and A. Temizel, (2021, January). "Generative data augmentation for vehicle detection in aerial images". In International Conference on Pattern Recognition (pp. 19-31). Springer, Cham.

[5] A.E. Gunduz, C. Ongun, T.T. Temizel and A. Temizel, (2016). "Density aware anomaly detection in crowded scenes". IET Computer Vision, 10(5), 374-381.

TEZ İZİN FORMU / THESIS PERMISSION FORM

ENSTİTÜ / INSTITUTE

Fen Bilimleri Enstitüsü / Graduate School of Natural and Applied Sciences [ ]

Sosyal Bilimler Enstitüsü / Graduate School of Social Sciences [ ]

Uygulamalı Matematik Enstitüsü / Graduate School of Applied Mathematics [ ]

Enformatik Enstitüsü / Graduate School of Informatics [X]

Deniz Bilimleri Enstitüsü / Graduate School of Marine Sciences [ ]

YAZARIN / AUTHOR

Soyadı / Surname : Öngün
Adı / Name : Cihan
Bölümü / Department : Bilişim Sistemleri / Information Systems

TEZİN ADI / TITLE OF THE THESIS (**İngilizce** / English) : DERİN SİNİR AĞLARI KULLANILARAK 3B MODELLERİN ÜRETİLMESİ VE DÜZENLENMESİ / GENERATION AND MODIFICATION OF 3D MODELS WITH DEEP NEURAL NETWORKS

TEZİN TÜRÜ / DEGREE: **Yüksek Lisans** / Master [ ] **Doktora** / PhD [X]

1. **Tezin tamamı dünya çapında erişime açılacaktır. /** Release the entire work immediately for access worldwide. [X]
2. **Tez iki yıl süreyle erişime kapalı olacaktır.** / Secure the entire work for patent and/or proprietary purposes for a period of **two year. *** [ ]
3. **Tez altı ay süreyle erişime kapalı olacaktır.** / Secure the entire work for period of **six months**. ***** [ ]

***** *Enstitü Yönetim Kurulu Kararının basılı kopyası tezle birlikte kütüphaneye teslim edilecektir.*
A copy of the Decision of the Institute Administrative Committee will be delivered to the library together with the printed thesis.

Yazarın imzası / Signature

Tarih / Date
15.09.2021