

**Electronics and Computer Science
Faculty of Physical Sciences and
Engineering
University of Southampton**

Alberto Tamajo
May 3, 2022

**CC3DVAEWGAN: a controllable conditional
3D point cloud VAE-WGAN based on
shrinking layers and extending layers**

Project supervisor: Dr. Jonathon Hare
Second examiner: Dr. Christine Evers

A project report submitted for the award of
**MEng Computer Science with Artificial
Intelligence**

Abstract

As 3D point clouds become increasingly popular in a wide range of computer vision and graphics applications, the ability to synthetically generate them, controlling their appearance, will be central to driving future developments.

For this reason, this project proposes the first architecture, named Controllable Conditional 3D Point Cloud VAE-WGAN(**CC3DVAEWGAN**), which aims to achieve both conditional and controllable generation of point clouds with the help of an innovative two-stage approach. Besides, **CC3DVAEWGAN** is also the first GAN/VAE-based point cloud generator that encodes and breeds point clouds leveraging self, local and global point correlations using a novel set of neural network layers: the *Shrinking Layer* and *Stack Shrinking Layer* for feature extraction, and the *Extending Layer* and *Stack Extending Layer* for generation.

The proposed architecture has been fully implemented using Pytorch and Pytorch Geometric. Even though several experiments have been conducted, further research is necessary to figure out the actual potential capabilities of these approaches.

Statement of Originality

- I have read and understood the [ECS Academic Integrity](#) information and the University's [Academic Integrity Guidance for Students](#).
- I am aware that failure to act in accordance with the [Regulations Governing Academic Integrity](#) may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Acknowledgments

First and foremost, I would like to thank my dad, mom and brother for their support. Without that support, I could not even have had the possibility of enrolling at a prestigious university and studying a subject that I love too wholeheartedly.

Second, I am extremely grateful to the i3Mainz Research Institute, especially PhD student Bastian Plaß, for granting me a summer research internship prior to the start of this project. This project would not have been possible without Bastian because he introduced me to the field of point clouds, which was completely unknown to me previously.

Also, I would like to thank my project supervisor Dr. Jonathon Hare and second examiner Dr. Christine Evers for having been available to help me whenever required.

Last but not least, I would like to recognize the invaluable moral assistance of my best friends Giuseppe Mistretta and Marco Saitta. The former is the most resilient person I know. He has an extraordinary ability to recover and rebound from challenges and setbacks. Marco has an amazing sense of humour, and I appreciate his recent commitment to behaviour change in order to achieve his work goals. I wish them the best for their future.

Contents

1	Introduction	11
2	Potential contributions	12
3	Fundamental building blocks	13
3.1	Generative adversarial networks	13
3.2	Wasserstein Generative Adversarial Networks	13
3.3	AutoEncoders	14
3.4	VAE/GAN	15
3.4.1	Overview	15
3.4.2	Training procedure	16
4	Point cloud generation	18
4.1	Uncontrollable generation	18
4.2	Controllable and conditional generation	18
5	Bridging the gap in the point cloud generation literature	19
5.1	Point cloud feature extraction	19
5.1.1	Challenges	19
5.1.2	Recent approaches	19
5.1.2.1	Structured grid-based methods	19
5.1.2.2	Raw point cloud-based methods	20
5.1.2.3	Graph-based methods	20
5.1.2.4	Proposed approaches	21
5.2	Point cloud generative architectures	21
5.2.1	Recent approaches	21
5.2.2	Proposed approaches	22

6	CC3DVAEWGAN	23
6.1	Overview	23
6.2	Similarities and dissimilarities with CPCGAN	23
6.3	Training procedure	24
6.4	Modes of generation	24
7	Blueprint VAE-WGAN	26
7.1	Overview	26
7.2	Components	26
7.2.1	Blueprint Encoder	27
7.2.1.1	Overview	27
7.2.1.2	Feature learning process first stage	28
7.2.1.3	Feature learning process second stage	28
7.2.2	Blueprint Generator	29
7.2.2.1	Overview	29
7.2.2.2	Generative process first stage	29
7.2.2.3	Generative process second stage	30
7.2.3	Blueprint Discriminator	30
7.2.3.1	Overview	30
7.2.3.2	Discriminative process first stage	31
7.2.3.3	Discriminative process second stage	31
7.3	Training procedure	31
7.3.1	Overview	31
7.3.2	Algorithm	32
7.3.2.1	Hyperparameters	32
7.3.2.2	Main body	34
8	Final WGAN	36

8.1	Overview	36
8.2	Components	36
8.2.1	Final Generator	37
8.2.1.1	Overview	37
8.2.1.2	Generative process	37
8.2.2	Final Discriminator	37
8.2.2.1	Overview	37
8.2.2.2	Discriminative process first stage	38
8.2.2.3	Discriminative process second stage	38
8.3	Training procedure	38
8.3.1	Overview	38
8.3.2	Algorithm	39
8.3.2.1	Hyperparameters	39
8.3.2.2	Main body	40
9	Implementation	42
10	Experiments	43
11	Project management	44
12	Conclusion	45
	References	46
A	Point2Node	51
A.1	Overview	51
A.2	Architecture	51
A.3	Dynamic Node Correlation Module	52
A.3.1	Self Correlation	52

A.3.2	Local Correlation	52
A.3.3	Non-local Correlation	53
A.4	Adaptive Feature Aggregation Module	54
A.4.1	Parameter-free Characteristic Modeling	54
A.4.2	Parameterized Characteristic Modeling	54
A.4.3	Gate Mechanism	55
A.4.4	Aggregation	55
B	Shrinking layer	56
B.1	Overview	56
B.2	Contributions	56
B.3	Architecture	57
B.3.1	Self-Correlation Layer	58
B.3.2	K-Means-Conv Layer	59
B.3.3	Adaptive Feature Aggregation Module	59
B.3.3.1	Parameter free Characteristic Modelling	60
B.3.3.2	Parameterized Characteristic Modeling	60
B.3.3.3	Gate Mechanism	60
B.3.3.4	Aggregation	60
B.3.4	Max-Pool Layer	61
B.4	Rationale behind the architecture	61
B.5	Horizontal stacking and non-local point correlations	62
B.6	Proof of permutation invariance	62
B.6.1	Self-Correlation Layer permutation invariance proof	63
B.6.2	K-Means-Conv Layer permutation invariance proof	63
B.6.3	AFA Module permutation invariance proof	64
B.6.4	Max-Pool Layer permutation invariance proof	65

C	Stack Shrinking Layer	66
C.1	Overview	66
C.2	Contributions	66
C.3	Architecture	66
C.4	Implementations	67
C.5	Proof of permutation invariance	68
D	Extending Layer	69
D.1	Overview	69
D.2	Contributions	69
D.3	Architecture	70
D.3.1	Self-Correlation Layer	71
D.3.2	K-Means-Conv Layer	72
D.3.3	Adaptive Feature Aggregation Module	72
D.3.3.1	Parameter free Characteristic Modelling	73
D.3.3.2	Parameterized Characteristic Modeling	73
D.3.3.3	Gate Mechanism	73
D.3.3.4	Aggregation	73
D.3.4	Max-Pool Layer	74
D.3.5	Upsampling Layer	74
D.4	Rationale behind the architecture	75
D.5	Practical considerations	76
D.6	Proof of permutation invariance	76
D.6.1	Upsampling Layer permutation invariance proof	76
E	Stack Extending Layer	78
E.1	Overview	78
E.2	Contributions	78

E.3	Architecture	78
E.4	Proof of permutation invariance	80
F	CPCGAN	81
F.1	Overview	81
F.2	Architecture	81
F.2.1	Overview	81
F.2.2	Structure GAN	82
F.2.3	Final GAN	82
F.3	Controllable generation	83
F.4	Training	83
F.4.1	Real structure point clouds and semantic labels formation	83
F.4.2	Process	84
F.4.3	Loss functions	84
G	First and second stage Final WGAN training procedures	85
G.1	Overview	85
G.2	First stage training procedure	85
G.3	Second stage training procedure	85
H	Implementation details	88
H.1	Point cloud layers	88
H.1.1	Future improvements	88
H.2	CC3DVAEWGAN architecture and training process	88
H.2.1	Requirements	89
I	Experiments details	90
I.1	Shrinking and Stack Shrinking Layers	90
I.2	Real point cloud blueprints formation	91

I.3	Extending Layer	93
I.4	Final WGAN	103
I.5	BlueprintVAE-WGAN	111
J	Project Management details	120
J.1	Selection and formulation of research problem	120
J.2	Initial planning	120
J.3	Definitive planning	123
J.4	Progress	124
J.5	Time management	126
J.6	Risks analysis and contingency plans	129

1 Introduction

Three-dimensional (3D) representations of real-life objects have a range of applications in different areas such as robotics, autonomous vehicles, augmented and virtual reality and other industrial purposes like manufacturing, building rendering e.t.c.

In recent years, point cloud has become one of the most significant data format for 3D representations thanks to its compactness, convenience and expressive power. Since 2018, the generation of point clouds has been an increasing concern to the research community, and over the years, architectures based on AutoEncoders(AEs) [1, 2] or Generative Adversarial Networks (GANs) [3, 4, 5] have proved to be increasingly more successful at generating realistic samples.

Although these achievements are a big leap forward, many real-world applications need to control the appearance (e.g. class) of the samples being generated. However, only one paper has addressed the controllability of generation [6], and conditional generation, as addressed for images [7], has not been concerned at all.

For this reason, this project proposes the first architecture, named Controllable Conditional 3D Point Cloud VAE-WGAN(**CC3DVAEWGAN**), which aims to achieve both conditional and controllable generation of point clouds with the help of an innovative two-stage approach. Conditional generation is accomplished by constraining the output to be a specific class' instance, while controllable generation is attained by providing a point cloud blueprint as guidance for the generation.

Furthermore, the totality of the point cloud generative modelling literature employs PointNet-based [8] discriminators/decoders. Consequently, as [9] demonstrates, their feature extraction performance is poor as it does not leverage the self, local and global correlation among the point cloud's points. CC3DVAEWGAN also bridges this gap by employing a novel set of layer architectures. The Shrinking and Stack Shrinking Layer are capable of encoding a point cloud leveraging the abovementioned correlations. Similarly, the Extending and Stack Extending Layer breed new points in a point cloud, exploiting the same correlations.

The point cloud layers proposed in this project, the CC3DVAEWGAN architecture and its training process are fully implemented using the Pytorch and Pytorch Geometric libraries. Several experiments have been conducted to verify the validity of the proposed solutions. However, further research is necessary to figure out their actual potential capabilities.

2 Potential contributions

The potential contributions of this project are:

1. A novel two-stage framework (CC3DVAEWGAN) aiming to achieve conditional and controllable generation of point clouds.
2. An innovative architecture combining a VAE and a WGAN, named VAE-WGAN, capable of learning a latent space distribution for each training class and a function from a latent vector to a point cloud blueprint.
3. A novel graph convolutional neural network and adjacency matrix module (Shrinking Layer) capable of exploring self correlations, local correlations and non-local correlations between points in point clouds in a semantic-based manner.
4. An inventive point cloud multi-feature extractor (Stack Shrinking Layer) that leverages self, local and non-local point correlations in a semantic-based fashion.
5. A novel architecture (Extending Layer) capable of exploring self, local and global correlations between points in a semantic-based manner for the generation of point clouds.
6. A unique point cloud generative approach (Stack Extending Layer), resembling current assembly lines, that generates point clouds in a specialist and semantic manner, leveraging self, local and non-local correlations.

3 Fundamental building blocks

In what follows, some background for understanding the basic building blocks of the **CC3DVAEWGAN**'s architecture is outlined.

3.1 Generative adversarial networks

Generative Adversarial Networks [3] are currently the state-of-the-art generative models. They are a clever way of training a generative model by framing the problem as a minimax game where two adversaries compete against each other. A generator G tries to synthesise samples that look indistinguishable from real data, while a discriminator D is tasked to distinguish between real samples drawn from the dataset and fake samples generated by G . Formally, D and G play a minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_{data}(z)} [\log(1 - D(G(z)))]$$

Introducing an analogy, the generator G can be thought of as a counterfeiter that tries to create fraudulent money and spend it without detection. Similarly, the discriminator D is comparable to a police officer trying to detect the fake currency. In this competition, both individuals constantly improve their techniques until the fake money is indistinguishable from the real one. Hence, GAN training convergence coincides with the Nash equilibrium [10], i.e., G and D reach a stable solution where their objectives cannot be improved further.

Unfortunately, GANs are remarkably difficult to train. They are heavily affected by the vanishing gradient problem, the training process is massively unstable, and mode collapse problems are commonplace. [11] argues that GANs may not even have Nash Equilibria. [12] rigorously proves that even in the presence of an optimal discriminator, generators learn nothing if the generated samples are far away from the real distribution. This happens because their gradients diminish dramatically in these circumstances. For a better understanding of the causes behind the GAN's training difficulties, the reader is referred to [12]. The previously mentioned issues limit GAN's applicability as it is complicated to use them in new domains or experiment with new variants.

3.2 Wasserstein Generative Adversarial Networks

Wasserstein GAN(WGAN) [13] improves the training stability of the GAN model by proposing a new training algorithm and a different loss function based on the Earth Mover's distance [14]. The latter is also known as Wasserstein loss. Unlike the minimax GAN loss, the Wasserstein metric has a smoother gradient almost everywhere and better correlates with the quality of the generated samples. A comparison between the gradients of the WGAN discriminator and GAN discriminator is provided in Figure 1.

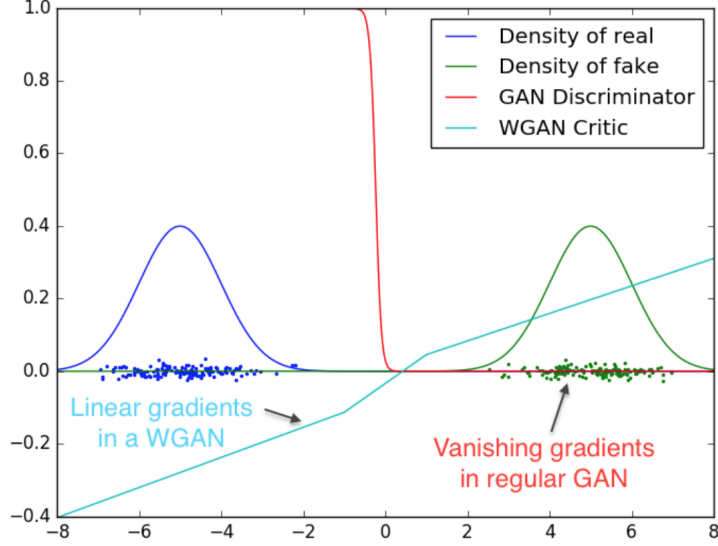


Figure 1: WGAN gradients VS GAN gradients [13]

Unfortunately, the computation of the Wasserstein loss is intractable. To overcome this problem, the WGAN’s authors exploit the Kantorovich-Rubinstein duality to transform the loss into a computationally-feasible one:

$$\mathbb{E}_{x \sim \mathcal{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathcal{P}_\theta}[f(x)]$$

where $f(\cdot)$ is the discriminator function, \mathcal{P}_r the distribution of real samples and \mathcal{P}_θ the distribution of generated samples.

The above equation requires the discriminator D to satisfy two requirements. Firstly, rather than predicting the probability of generated samples as being real or fake, D is constrained to score their realness or fakeness by outputting a scalar value. For this reason, [13]’s authors call D the critic. Secondly, the discriminator is also demanded to satisfy the 1-Lipschitz constraint. [13] proposes enforcing it through weight clipping. However, this is not an optimal solution for many reasons. As a matter of example, the weights will converge very slowly if the clipping parameter is large. In contrast, if the clipping parameter is small, the vanishing gradient problem could rapidly become an issue. Algorithm 1 provides an overview of the WGAN training procedure.

3.3 AutoEncoders

An AutoEncoder(AE) [1] is a two-parts neural network architecture that is concerned with learning a representation (encoding) for a set of data. The Encoder component compresses a data point \mathbf{x} into its latent representation \mathbf{z} . The Decoder can then produce a reconstruction \mathbf{x}' of \mathbf{x} from its encoded version \mathbf{z} . An efficient data encoding is learnt when the reconstruction error between \mathbf{x}' and \mathbf{x} is minimised.

Algorithm 1 WGAN training algorithm. All the experiments in the original paper use the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{critic} = 5$

Require: α , the learning rate. c , the clipping parameter. m , the batch size. n_{critic} , the number of iterations of the critic per generator iteration

Require: w_0 , initial critic parameters. θ_0 , initial generator parameters

```

1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{critic}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathcal{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p_{(z)}$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSPProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p_{(z)}$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSPProp}(\theta, g_\theta)$ 
12: end while

```

The main applications of AEs are dimensionality reduction and information retrieval, but they do not have an obvious generative interpretation. Variational AutoEncoders (VAEs) [2] are an extension to AEs that learn a regularised encoding distribution to ensure that the latent space has good properties to generate new data. Still, VAEs can be used for feature learning since they are able to learn representations with disentangled factors [15].

3.4 VAE/GAN

3.4.1 Overview

VAE/GAN [16] is an unsupervised generative model that combines a VAE and a GAN to simultaneously learn how to encode, generate and compare dataset samples. The rationale behind the VAE/GAN architecture lies in the fact that despite the VAE's optimal encoding capabilities, currently used similarity metrics impose a hurdle for learning good generative models.

Indeed, the reconstruction error provides the main training signal, but currently adopted element-wise measures, such as the squared error, are not suitable for images and other signals with invariances. For instance, element-wise measures applied to images do not model the properties of human visual perception. Conversely, GAN discriminators implicitly learn a high-level and sufficiently invariant similarity measure to discriminate between real and fake samples.

VAE/GAN builds upon the abovementioned properties. Thus, it combines the advantage of GAN as a high-quality generative model and VAE as a method that produces an encoder of data into the latent space. As a result, the VAE

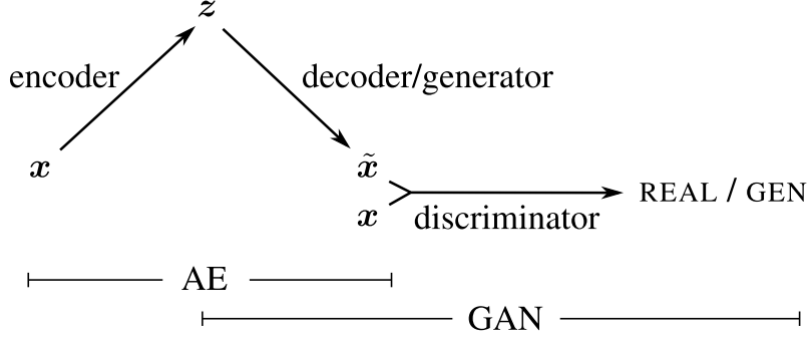


Figure 2: VAE/GAN network architecture [16]. The VAE decoder and the GAN generator are merged into one entity.

decoder and the GAN generator are merged into one entity, and the joint architecture is trained simultaneously. Figure 2 illustrates the VAE/GAN network.

3.4.2 Training procedure

Algorithm 2 Training the VAE/GAN model

- 1: $\theta_{Enc}, \theta_{Dec}, \theta_{Dis} \leftarrow$ initialise network parameters
 - 2: **repeat**
 - 3: $X \leftarrow$ random mini-batch from dataset
 - 4: $Z \leftarrow \text{Enc}(X)$
 - 5: $\mathcal{L}_{prior} \leftarrow D_{KL}(q(Z|X)||p(Z))$
 - 6: $\bar{X} \leftarrow \text{Dec}(Z)$
 - 7: $\mathcal{L}_{llike}^{Disl} \leftarrow -\mathbb{E}_{q(Z|X)}[\log p(Disl(X)|Z)]$
 - 8: $Z_p \leftarrow$ sample from prior $\mathcal{N}(0, I)$
 - 9: $X_p \leftarrow \text{Dec}(Z_p)$
 - 10: $\mathcal{L}_{GAN} = \log(Dis(X)) + \log(1 - Dis(\bar{X})) + \log(1 - Dis(X_p))$
 - 11:
 - 12: // Update parameters according to gradients
 - 13: $\theta_{Enc} \xleftarrow{+} -\nabla_{\theta_{Enc}}(\mathcal{L}_{prior} + \mathcal{L}_{llike}^{Disl})$
 - 14: $\theta_{Dec} \xleftarrow{+} -\nabla_{\theta_{Dec}}(\gamma \mathcal{L}_{llike}^{Disl} + \mathcal{L}_{GAN})$
 - 15: $\theta_{Dis} \xleftarrow{+} -\nabla_{\theta_{Dis}} \mathcal{L}_{GAN}$
 - 16: **until** deadline
-

Algorithm 2 provides an overview of the training procedure. Three different loss functions are employed during the training process:

$$\begin{aligned}\mathcal{L}_{prior} &= D_{KL}(q(z|x)||p(z)) \\ \mathcal{L}_{GAN} &= \log(Dis(x)) + \log(1 - Dis(Dec(z))) + \log(1 - Dis(Dec(Enc(x)))) \\ \mathcal{L}_{llike}^{Disl} &= -\mathbb{E}_{q(z|x)}[\log p(Disl(x)|z)]\end{aligned}$$

\mathcal{L}_{prior} is the Kullback-Leibler divergence. It imposes a prior over the latent space $p(z)$ so that to regularise the encoder. \mathcal{L}_{GAN} is the minimax binary cross

entropy GAN loss function. The latter is augmented with an additional term that uses samples from $q(z|x)$ in addition to the prior $p(z)$. $\mathcal{L}_{llike}^{Dis_l}$ replaces the standard element-wise reconstruction error $\mathcal{L}_{llike}^{pixel}$ with a loss expressed in terms of the hidden representation $Dis_l(x)$ of the discriminator's l^{th} layer. The probability distribution in $\mathcal{L}_{llike}^{Dis_l}$ is equal to:

$$p(Dis_l(x)|z) = \mathcal{N}(Dis_l(x)|Dis_l(\bar{x}), I)$$

where \bar{x} is the reconstruction of x generated by the Decoder.

As regards the update of the network's parameters, let θ_{Enc} , θ_{Dec} and θ_{Dis} be the parameters of the Encoder, Decoder and Discriminator, respectively. θ_{Enc} is updated according to the gradients of the prior and reconstruction error. The gradient of \mathcal{L}_{GAN} updates θ_{Dis} . The error signal for θ_{Dec} comes from both $\mathcal{L}_{llike}^{pixel}$ and \mathcal{L}_{GAN} :

$$\theta_{Dec} \stackrel{+}{\leftarrow} -\nabla_{\theta_{Dec}}(\gamma\mathcal{L}_{llike}^{Dis_l} + \mathcal{L}_{GAN})$$

The parameter γ weights the ability to reconstruct vs fooling the discriminator.

4 Point cloud generation

4.1 Uncontrollable generation

GANs are currently the state-of-the-art generative models, and their success on image generation [4, 17, 18, 19] has inspired the first GAN for point clouds, named r-GAN [20]. However, its major drawback is the incapability of generating diverse shapes as it is based on multiple fully connected layers.

[21] improves upon r-GAN by using a dynamic graph convolution network as a generator, but this process is time and computationally expensive due to the computation of adjacency matrices. To overcome such computational complexity, Tree-GAN [22] uses ancestor information from a tree-structured graph to exploit the connectivity of a graph. Tree-GAN outperforms [21].

4.2 Controllable and conditional generation

The previously mentioned point cloud GANs provide no control over the data being generated. Indeed, while controllable GANs have been well researched on images, the same does not hold for point clouds.

The only GAN that addresses the problem of controllable generation for point clouds is CPCGAN [6]. CPCGAN allows controlling the generated shape through a two-stage approach. The first stage receives a random latent code as input and generates a middle-layer representation called structure point cloud. The second stage takes the structure point cloud and generates the final point cloud by reproducing a certain number of points from every structure point. By changing the middle-level representation, the generation process can be controlled. This approach takes inspiration from [23]. Although CPCGAN is currently the state-of-the-art point cloud generator, it has a significant drawback as the training dataset needs to provide semantic labels for each point. For further information regarding CPCGAN, the reader is referred to Appendix F.

CPCGAN achieves controllable generation, but it does not address conditional generation as in CGAN [7]. For this project, many real-world applications would greatly benefit from a controllable and conditional point cloud generative architecture. This is the reason why CC3DVAEWGAN aims to achieve both conditional and controllable generation of point clouds.

5 Bridging the gap in the point cloud generation literature

The GAN-based point cloud generators proposed in the literature do not employ state-of-the-art discriminators or generators. This project also aims to bridge this gap. However, rather than directly implementing the state-of-the-art approaches, this project proposes four novel architectures, the *Shrinking Layer* and *Stack Shrinking Layer* for feature extraction, and the *Extending Layer* and *Stack Extending Layer* for generation, that potentially could achieve better performance. In what follows, an overview of the point cloud feature extraction and generative architecture literature is provided so that to help the reader understand how the abovementioned solutions fit within the current research trends.

5.1 Point cloud feature extraction

5.1.1 Challenges

Feature extraction is a process concerned with projecting a high-dimensional dataset into a lower-dimensional space (latent space) with the aim of retaining only the most significant information. It turns out that point clouds as an input modality present a unique set of challenges when building feature extractors due to their properties:

1. **Irregularity:** points in the point cloud are not evenly spread across the different regions of the representation.
2. **Unstructured:** the distance between two neighbouring points in a point cloud is irregular.
3. **Unorderdness:** a point cloud is a set of points without any specific order.

5.1.2 Recent approaches

This section contains a brief survey of the recent approaches proposed to extract features from point clouds.

5.1.2.1 Structured grid-based methods

Convolutional Neural Networks(CNNs) [24, 25] can only be applied to regular, ordered and structured data; thus, they are unsuitable for point clouds. To overcome this challenge, many approaches convert point cloud data into a

structured form. These approaches can be divided into two categories: voxel-based and multiview-based.

Voxel-based methods [26, 27, 28, 29, 30] convert a point cloud into a fixed-size voxel and convolve it with 3D kernels. Although voxel-based approaches show good performance, they suffer from high memory consumption. Besides, artifacts are introduced by the voxelisation operation.

Multiview-based approaches [31, 32, 33, 34, 35, 36] convert a point cloud into a collection of 2D images and apply 2D CNNs to it. They achieve better performance than the voxel-based counterparts as 2D CNNs have been subject to extensive research.

This project believes that structured grid-based methods are not the future of point cloud feature extraction as it is possible to capture patterns directly on the raw point clouds.

5.1.2.2 Raw point cloud-based methods

PointNet [8] is the first approach that applies deep learning directly on raw point clouds, and it is the basis for most of the techniques in this category. It overcomes the challenging properties of point cloud data using two symmetric functions, a shared Multilayer Perceptron (MLP) and a max-pooling function. The MLP maps each point to a feature vector, and the max-pooling function aggregates all the feature vectors into a feature descriptor of the input.

The design of PointNet and other methods [37, 38, 39, 40] lacks the capability of capturing local structures. However, in a point cloud, points do not exist in isolation; rather, multiple points together are necessary to give meaning to a shape. For this reason, several approaches [41, 42, 43, 44, 45, 46, 47, 48, 49] also explore the local correlation between points and, consequently, achieve better performance.

The internal working of PointNet is rather different from how humans perceive the world. Indeed, humans unconsciously analyse the dependencies between the different semantic parts of an object, which may be either close together or far apart. All of the point cloud GANs proposed in the literature employ a PointNet-based architecture in the discriminator component; thus, they cannot capture local features.

5.1.2.3 Graph-based methods

Graph-based approaches [50, 51, 52, 9] represent a point cloud as a graph to easily capture structural information among the points. Indeed, in a graph structure, the graph edges explicitly represent the correlation between points.

Point2Node [9] is the current state-of-the-art model because of its capability to

leverage each point’s self-correlation, the local correlation between each point and its neighbours, and the non-local correlation among distant points having a long-range dependency. Therefore, Point2Node resembles the process that humans use to understand the semantic of objects.

5.1.2.4 Proposed approaches

This project proposes two innovative graph-based solutions, named Shrinking Layer and Stack Shrinking Layer, to extract features from a point cloud.

The Shrinking Layer is designed to leverage self, local and global point correlations as proposed in Point2Node [9], but unlike the latter, it embeds ideas from the CNN literature [24, 25] instead. Since CNNs achieve state-of-the-art performance in image feature extraction, it seems reasonable to design their point cloud counterpart.

The Stack Shrinking Layer builds upon the Shrinking Layer in order to simultaneously extract multiple features from a point cloud in a similar fashion as CNNs generate multiple feature maps at a given depth of the network. Therefore, the Shrinking Layer is the first point cloud feature extraction architecture fully resembling current CNN architectures.

For an extensive explanation of the Shrinking Layer and Stack Shrinking Layer, the reader shall consult Appendix B and C, respectively.

5.2 Point cloud generative architectures

5.2.1 Recent approaches

Even though CPCGAN [6] is the state-of-the-art point cloud generator, its generator component is not considered remarkable because it exploits semantically annotated points and is PointNet-based. Research in this field should distance from devising architecture relying on semantically annotated points as the development of such datasets will not be scalable in the future.

The most prominent point cloud generative architectures are used in [21] and TreeGAN [22]. They adopt a multilayer generator network that incrementally generates hierarchical graph embeddings that represent better and better approximations of the graph of the output point cloud. However, while [21] updates each point by referring to the values of its neighbours, TreeGAN considers the ancestors of each point from a tree-structured graph.

Regarding the upsampling process, [21] doubles the number of points after each layer by applying an upsampling operation to each point exploiting local structures. On the other hand, TreeGAN multiplies each point by a matrix to produce an arbitrary number d of child points from each point.

5.2.2 Proposed approaches

This project proposes two innovative architectures, named Extending Layer and Stack Extending Layer, for the generation of point clouds.

Unlike [21] and [22], the Extending Layer is capable of breeding new points in a point cloud leveraging self, local and global point correlations. The Extending Layer’s approach for the upsampling process is reminiscent of both [21] and [22], though it ultimately differs from each one as it is semantic-based.

As regards the Stack Extending Layer, the latter generates point clouds in a similar manner as today’s assembly lines manufacture goods. It also leverages self, local and global point correlations as it is Extending Layer-based.

For an extensive explanation of the Extending Layer and Stack Extending Layer, the reader is referred to Appendix D and E, respectively.

6 CC3DVAEWGAN

6.1 Overview

CC3DVAEWGAN is a novel two-stage neural network architecture that aims to achieve both conditional and controllable generation of point clouds. The first stage and second stage of this architecture are called **Blueprint VAE-WGAN** and **Final WGAN**, respectively.

The first stage takes inspiration from VAE/GAN [16] to simultaneously learn a latent space distribution for each training class and a function that decodes a latent representation back to its corresponding point cloud blueprint. The latter is a sparse point cloud containing significant structural information. Because of the Blueprint VAE-WGAN’s decoding properties, conditional generation is attained by feeding a latent representation drawn from the probability density of the appropriate class’ latent space.

With the guidance of the point cloud blueprint, Final WGAN breeds a certain number of new points so to produce a complete point cloud that preserves the original structural information encoded in the blueprint. Given that Final WGAN learns a function from the point cloud blueprint domain to the complete point cloud domain, then controllable generation of point clouds is achieved by altering the structure of the point cloud blueprints.

6.2 Similarities and dissimilarities with CPCGAN

CC3DVAEWGAN draws inspiration from CPCGAN [6] for the controllable generation of point clouds, but apart from that, these two point cloud generative architectures are vastly different.

Firstly, this project’s proposal aims to achieve both conditional and controllable generation, while CPCGAN limits itself to the controllability aspect. As a result, the first stages of these networks have different objectives and, consequently, dissimilar structures. Indeed, CC3DVAEWGAN employs a novel network named VAE-WGAN, while CPCGAN uses an ordinary WGAN.

Secondly, CPCGAN employs fully-connected layers and PointNet-like networks for point clouds’ feature extraction and generation processes. Thus, point clouds are encoded and generated by only leveraging self correlations. On the other hand, since feature extraction and generation of point clouds require exploring self, local and non-local correlations, CC3DVAEWGAN is equipped with, potentially, the most advanced architectures: the *Shrinking Layer* or *Stack Shrinking Layer* for feature extraction and the *Extending Layer* or *Stack Extending Layer* for generation.

The stack versions of the previous architectures achieve better performance but also incur a higher time complexity. Therefore, which version to use is

a trade-off between performance and run time. For further knowledge on the benefits stemming from the use of the abovementioned architectures, the reader is referred to Appendix B through E.

Thirdly, not only are the architectures of the two proposals in question diverse but also their training algorithms.

Last but not least, CPCGAN leverages semantically annotated points for the completion of its objectives; however, CC3DVAEWGAN distances itself from relying on them as the development of datasets with semantically annotated points will not be scalable in the future.

6.3 Training procedure

Drawing inspiration from CPCGAN, CC3DVAEWGAN’s training procedure also comprises two stages to speed up the training process and enhance performance.

In the first stage, both Blueprint VAE-WGAN and Final WGAN are trained. On the grounds that Final WGAN learns a function from the point cloud blueprint domain to the complete point cloud domain, then point cloud blueprints need to be fed as input during the training process. As regards the first stage of training, these point cloud blueprints are generated by extracting structural information from the point clouds contained in the training dataset.

In contrast, in the second stage, only Final WGAN is trained. The latent space distribution learnt for each class by Blueprint VAE-WGAN is used to create point cloud encodings, which in turn are given as input to the Blueprint Generator for the generation of point cloud blueprints. Thus, Final WGAN receives fake point cloud blueprints as input during the second stage of training.

The main advantage of this two-stage training approach is that it enables the learning of conditional and controllable generation. Indeed, in the first stage, controllable generation is fully learned as Final WGAN learns how to create complete point clouds from the provided point cloud blueprints.

On the other hand, conditional generation is only partially learned as Blueprint VAE-WGAN simultaneously learns a latent space distribution for each training class and a function that decodes a latent representation back to its corresponding point cloud blueprint. The second stage is necessary to learn how to map artificially generated point cloud blueprints to complete point clouds, fully achieving conditional generation.

6.4 Modes of generation

After training the network on a dataset containing a set of classes C , the Blueprint VAE-WGAN generator (Blueprint Generator), the Final WGAN

generator (Final Generator) and the latent distribution learnt for each class are saved into memory, enabling three modalities of generation:

1. **Conditional generation:** given an input class $c \in C$, a latent vector v is drawn from c 's latent space distribution. The Blueprint Generator receives v as input and generates a point cloud blueprint B . The Final Generator takes as input B and outputs a complete point cloud P such that $P \in c$.
2. **Controllable generation:** given a point cloud blueprint B , the Final generator takes it and generates a final point cloud P .
3. **Random generation:** a random class $c \in C$ is selected and an instance of it is generated following the same scheme described in Conditional generation

7 Blueprint VAE-WGAN

7.1 Overview

Blueprint VAE-WGAN is the first stage of the CC3DVAEWGAN architecture. It builds upon the training principles introduced in [16] to simultaneously learn how to encode and generate point cloud blueprints. The end result is a method that combines the advantage of GAN as a high-quality generative model and VAE as a method that produces an encoder of data into the latent space.

Specifically, this architecture learns a Mixture of Gaussian distribution $\mathcal{M}(z|c)$ for each training class c and a function that maps a given latent representation $z \sim \mathcal{M}(z|c)$ to z 's corresponding point cloud blueprint. The latter needs to be a point cloud blueprint instance of class c . The Mixture of Gaussian distribution $\mathcal{M}(z|c)$ for a given class c consists of K gaussian components $\mathcal{N}(\mu, \Sigma|X_i)$ such that $\mathcal{N}(\mu, \Sigma|X_i)$ is the distribution encoding learnt for the i^{th} point cloud training instance in training class c . K is the total number of training instances in training class c . Each component coefficient is assumed to be equiprobable, that is, each of them is $\frac{1}{K}$.

Therefore, the inputs to the VAE-WGAN generator are not random latent vectors drawn from a Standard Gaussian distribution like in all other GANs, but latent vectors drawn from a class' latent space distribution. Consequently, the task of the generator becomes similar to the task of a decoder in a VAE, that is, reconstructing the blueprint point cloud corresponding to the given latent vector. Thus, unlike [7, 53, 54] that achieve conditional generation by providing the generator with a noise input z augmented with structured semantic features y , VAE-WGAN achieves conditional generation by default if the provided latent vector is drawn from the Mixture of Gaussian Distribution of the appropriate class.

Furthermore, the generation of point cloud blueprints from latent vectors drawn from a Mixture of Gaussian Distribution may reduce the intra-class mode collapse problem. This may hold because the Mixture of Gaussian Distribution encodes every training instance and many variations of each instance for a given training class.

Given that Blueprint VAE-WGAN is a unique architecture, it requires a special training procedure as well. Its novel training algorithm will be extensively explained in later sections.

7.2 Components

Blueprint VAE-WGAN is composed of three components:

- Blueprint Encoder
- Blueprint Generator
- Blueprint Discriminator

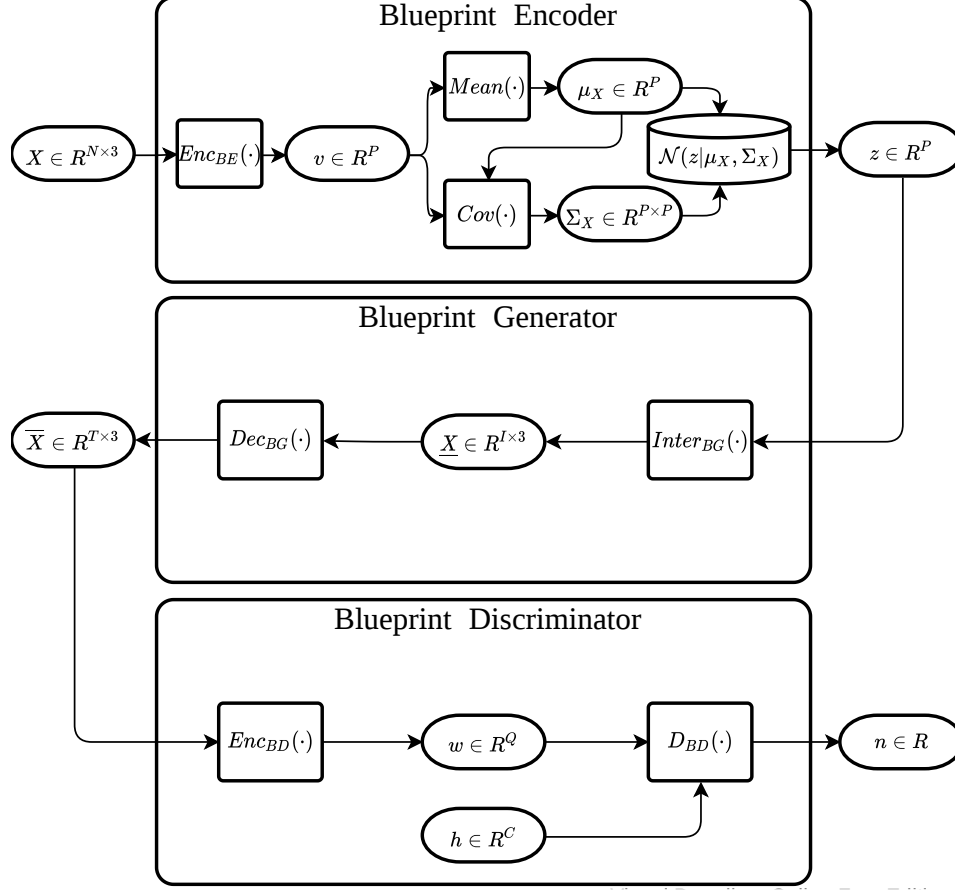


Figure 3: Blueprint VAE-WGAN's architecture

Figure 3 illustrates the Blueprint VAE-WGAN's architecture. In what follows, the functionalities of each of the abovementioned constituents are exhaustively elucidated.

7.2.1 Blueprint Encoder

7.2.1.1 Overview

The purpose of the Blueprint Encoder is to learn a regularised encoding distribution $z \sim q(z|X)$, where $X \in R^{N \times 3}$ is a 3-dimensional point cloud and $z \in R^P$ is a P-dimensional latent vector. In simple terms, the component in question performs a feature learning task to encode a point cloud as a distribution over the latent space. For reasons of practicality and due to the ubiquity proved by the Central Limit Theorem, the encoding distribution is assumed

to be a multivariate normal. Thus, $z \sim q(z|X) = \mathcal{N}(\mu, \Sigma|X)$, where μ is a P -dimensional mean vector and Σ is a $P \times P$ diagonal covariance matrix. Given the assumption of normality, the task of the Blueprint Encoder revolves around estimating the parameters μ and Σ given a point cloud X . Assuming that the Encoder estimates $\mu = \mu_X$ and $\Sigma = \Sigma_X$ for a given point cloud X , $\mathcal{N}(z|\mu_X, \Sigma_X)$ describes the distribution over all possible encodings z from which the point cloud X could have been generated.

The feature learning process is carried out by means of a two-stage process.

7.2.1.2 Feature learning process first stage

In the first stage, a function $Enc_{BE} : R^{N \times 3} \rightarrow R^P$ encodes a point cloud X into a P -dimensional vector $v = Enc_{BE}(X)$. Ideally, the vector v produced by the function $Enc_{BE}(\cdot)$ should preserve all of the most important features of the point cloud X . In order for this to hold, $P \gg 3$ as the dimensionality of the vector v needs to be sufficiently large to represent the most significant points in the original point cloud and their structural relations succinctly.

So far, the function $Enc_{BE}(\cdot)$ has been described as an encoding function; however, it may be simpler for the reader to interpret it as a mapping from the three-dimensional point cloud domain to a much higher P -dimensional point cloud domain. Every point cloud in the three-dimensional domain is mapped to a higher-dimensional point cloud comprising a single point in a P -dimensional space.

Even though the feature extractors proposed in the point cloud literature cannot satisfy the ideal properties of the function $Enc_{BE}(\cdot)$ as they are not capable of leveraging self, local and non-local correlations in a semantic-based manner, a sequence of either Shrinking Layers or Stack Shrinking Layers may provide a satisfactory approximation to these ideal properties. This is the reason why the function $Enc_{BE}(\cdot)$ is set to be equivalent to $S(\cdot|s_1, \dots, s_k)$ where S denotes a function comprising a sequence of K Shrinking Layers or Stack Shrinking Layers.

7.2.1.3 Feature learning process second stage

The second stage of the feature learning process is concerned with estimating the distribution parameters μ_X and Σ_X for a given point cloud X using the encoding $v = Enc_{BE}(X)$ produced in the first stage. Two functions are utilised to achieve this purpose.

The function $Mean : R^P \rightarrow R^P$ takes as input the encoding vector v to produce the vector μ_X as output. Instead, the function $Cov : (R^P, R^P) \rightarrow R^{P \times P}$ outputs the covariance matrix Σ_X given the mean vector μ_X and the encoding $v = Enc_{BE}(X)$ as input.

It is important to notice that the function $Mean(\cdot)$ allows the formation of a wide range of multivariate normal distributions as the mean μ_X is not constrained to be the encoding $v = Enc_{BE}(X)$. In other words, this function may shift the multivariate normal distribution centred at $v = Enc_{BE}(X)$ by a certain amount.

The role of the function $Cov(\cdot)$ is to control the dispersion of the distribution so that $\mathcal{N}(z|\mu_X, \Sigma_X)$ describes all possible encodings z which are similar to v and from which v is likely to be drawn. Given that any $z \sim \mathcal{N}(z|\mu_X, \Sigma_X)$ is a latent code similar to the encoding $v = Enc_{BE}(X)$, any point cloud \hat{X} generated from z should have no major structural differences from the original point cloud X .

The functions $Mean(\cdot)$ and $Cov(\cdot)$ are not hand-crafted; they are multilayer perceptrons.

7.2.2 Blueprint Generator

7.2.2.1 Overview

The Blueprint Generator is concerned with learning how to decode a latent representation $z \in R^P$ back to the 3-dimensional point cloud space. Specifically, this generator’s task involves reconstructing the point cloud blueprint associated with the point cloud being encoded by the latent representation z .

More formally, let $X \in R^{N \times 3}$ be a point cloud, $z \sim q(z|X)$ be the latent representation of X and $B(X) \in R^{T \times 3}$ be X ’s corresponding blueprint point cloud. The aim of the Blueprint Generator is to learn a probability distribution $p(\bar{X}|z)$ such that $\bar{X} \sim p(\bar{X}|z)$ is a good reconstruction for $B(X)$. It is important to notice that T is a hyperparameter of this component and represents the number of points in the generated point cloud blueprint.

So far, this component has been described from the VAE perspective; however, if the GAN perspective is taken into consideration, this component’s functionalities can be interpreted as generating a point cloud blueprint from a latent representation z . It is possible to perform this shift in perspective because the Blueprint Generator is part of the Blueprint VAE-WGAN, which is a VAE-GAN.

The generative process of a point cloud blueprint \bar{X} from a latent representation $z \in R^P$ comprises two stages.

7.2.2.2 Generative process first stage

In the first stage, a function $Inter_{BG} : R^P \rightarrow R^{I \times 3}$ takes as input a latent vector $z \in R^P$ and returns an intermediary point cloud $\underline{X} \in R^{I \times 3}$. The function $Inter_{BG}(\cdot)$ acts as an intermediary between the latent vector z and the

function $Dec_{BG}(\cdot)$. The next section will give more details about the $Dec_{BG}(\cdot)$ function. Meanwhile, it is sufficient to know that $Dec_{BG}(\cdot)$ is responsible for generating a blueprint point cloud \overline{X} and that $Dec_{BG}(\cdot) = E(\cdot|e_1, \dots, e_k)$, where E denotes a function comprising a sequence of K Extending Layers or Stack Extending Layers. Since $Dec_{BG}(\cdot) = E(\cdot|e_1, \dots, e_k)$, its input cannot be a latent representation z as both Extending and Stack Extending Layers are only capable of incrementally generating better hierarchical graph embeddings starting from a given point cloud input.

As such, the function $Inter_{BG}(\cdot)$ plays a key role here: mapping a latent vector z to an intermediary point cloud \underline{X} which can be fed to the function $Dec_{BG}(\cdot)$. The function $Inter_{BG}(\cdot)$ is not handcrafted; it is a multilayer perceptron.

It can be argued that the generation process of a point cloud cannot achieve optimality if self, local and global point correlations are not leveraged. As a consequence, it may be claimed that the generative process illustrated so far is rather poor given that $Inter_{BG}(\cdot)$ is a mere MLP. This argument would be entirely valid if the hyperparameter I in the function $Inter_{BG} : R^P \rightarrow R^{I \times 3}$ were to be large as the function $Dec_{BG}(\cdot)$ could not breed enough new points exploiting self, local and global point correlations. However, keeping the hyperparameter I low allows $Dec_{BG}(\cdot)$ to produce numerous new points and generate increasingly better graph embeddings exploiting the previously mentioned point correlations.

7.2.2.3 Generative process second stage

The second stage of the generative process consists of producing a point cloud blueprint $\overline{X} \in R^{T \times 3}$ from a given intermediary point cloud $\underline{X} \in R^{I \times 3}$. The function $Dec_{BG} : R^{I \times 3} \rightarrow R^{T \times 3}$ realises this purpose.

The function $Dec_{BG}(\cdot)$ should leverage self, local and global point correlations during the generative process. To the best of this author’s knowledge, no proposal in the point cloud generative literature can satisfy the abovementioned properties. Nonetheless, it has been shown that a sequence of Extending Layers or Stack Extending Layers is capable of exploiting self, local and global correlations to incrementally generate hierarchical graph embeddings. For this reason, $Dec_{BG}(\cdot)$ is set to be equivalent to $E(\cdot|e_1, \dots, e_k)$, where E denotes a function comprising a sequence of K Extending Layers or Stack Extending Layers.

7.2.3 Blueprint Discriminator

7.2.3.1 Overview

The Blueprint Discriminator is tasked to distinguish between real point cloud blueprints sampled from the dataset and fake point cloud blueprints generated

by the Blueprint Generator.

The realness or fakeness of a given point cloud blueprint is scored with respect to its class reference. Formally, let $\bar{X} \in R^{T \times 3}$ be a point cloud blueprint and $h \in R^C$ the one-hot encoding vector of \bar{X} 's class, the Blueprint discriminator's duty is to score the realness or fakeness of \bar{X} with respect to its class h . Notice that C stands for the number of classes in the training dataset.

It is important to realise that this discriminator does not predict the probability of a given point cloud blueprint \bar{X} as being real or fake but rather scores its realness or fakeness. The reason behind this behaviour lies in the fact that this component is the discriminator unit of a Variational AutoEncoder-Wasserstein GAN, Blueprint VAE-WGAN.

The discriminative process of a point cloud blueprint $\bar{X} \in R^{T \times 3}$ consists of a two-stage approach.

7.2.3.2 Discriminative process first stage

The first stage involves encoding a point cloud blueprint $\bar{X} \in R^{T \times 3}$ into a Q -dimensional vector $w \in R^Q$. The objective of this stage is achieved through the function $Enc_{BD} : R^{T \times 3} \rightarrow R^Q$.

The functionalities and properties of this function need to be equivalent to the ones of the function $Enc_{BE}(\cdot)$ described in section 7.2.1. Accordingly, $Enc_{BD}(\cdot) = S(\cdot | s_1, \dots, s_k)$ where S denotes a function comprising a sequence of K Shrinking Layers or Stack Shrinking Layers.

7.2.3.3 Discriminative process second stage

In the second stage, the function $D_{BD} : (R^Q, R^C) \rightarrow R$ takes as input the encoding vector $w = Enc_{BD}(\bar{X})$ and \bar{X} 's class reference h . It outputs a real number $n \in R$ which scores the realness or fakeness of \bar{X} with respect to h . Like all the other functions in Blueprint VAE-WGAN, $D_{BD}(\cdot)$ is a MLP.

7.3 Training procedure

7.3.1 Overview

The Blueprint VAE-WGAN training procedure is a unique proposal in the point cloud generative literature due to several reasons.

Firstly, it builds upon the training principles introduced in [16] to simultaneously learn how to encode and generate point cloud blueprints. The end result is a method that combines the advantage of GAN as a high-quality generative

model and VAE as a method that produces an encoder of data into the latent space.

Secondly, as in [16], the parameters of each component are updated with respect to different loss functions. The use of three loss functions is necessary because a combined loss would collapse the Blueprint Discriminator’s gradient to 0.

Thirdly, the loss functions drive the component’s parameters towards optimality in an innovative way. Even though they are inspired by [16], they ultimately differ as the GAN binary cross-entropy loss is replaced with the Wasserstein loss and a point cloud distance metric serves as reconstruction loss.

Lastly, a Mixture of Gaussian distribution is utilised to describe each training class’ manifold. Thus, a latent space distribution for each training class is learned during the training process. In what follows, the Blueprint VAE-WGAN training algorithm is extensively illustrated.

7.3.2 Algorithm

Algorithm 3 provides a summary of the Blueprint VAE-WGAN training procedure.

7.3.2.1 Hyperparameters

The learning process relies on several hyperparameters whose values need to be set prior to the execution of the main algorithm’s body.

λ_{BE} , λ_{BG} and λ_{BD} are the learning rates of the Blueprint Encoder, Blueprint Generator and Blueprint Discriminator, respectively. Three learning rates are indispensable because the gradient descent rule updates the parameters of each component with respect to different loss functions.

m stands for the batch size. $n_{blueprint}$ and $n_{classes}$ are the number of points inside the point cloud blueprints and the number of classes in the training dataset, respectively.

This training process is also based on the WGAN algorithm [13]; thus, the clipping parameter c and the number of iterations of the discriminator per generator iteration n_{BD} are required as well.

The last hyperparameter is the style and content weight γ [16]. The latter weights the ability of the Blueprint Generator to reconstruct vs fooling the Blueprint Discriminator.

Algorithm 3 Blueprint VAE-WGAN training algorithm

Require:

- 1: λ_{BE} : Blueprint Encoder(BE) learning rate.
 - 2: λ_{BG} : Blueprint Generator(BG) learning rate.
 - 3: λ_{BD} : Blueprint Discriminator(BD) learning rate.
 - 4: m : batch size.
 - 5: $n_{blueprint}$: number of points for the point cloud blueprints.
 - 6: $n_{classes}$: number of classes in the dataset
 - 7: c : clipping parameter.
 - 8: n_{BD} : number of iterations of the discriminator per generator iteration.
 - 9: γ : weighting style and content parameter
-
- 10:
 - 11: $\theta_{BE} \leftarrow$ initialize BE parameters
 - 12: $\theta_{BG} \leftarrow$ initialize BG parameters
 - 13: $\theta_{BD} \leftarrow$ initialize BD parameters
 - 14: $GM \leftarrow \{\}$ \triangleright Gaussian Mixture dictionary
 - 15: **while** θ_{BE} and θ_{BG} and θ_{BD} have not converged **do**
 - 16: **while** there are batches available **do**
 - 17: $X \leftarrow$ a batch $\{x^{(i)}\}_{i=1}^m$ of point clouds from the real dataset
 - 18: $B \leftarrow \{\text{Kmeans}(x^{(i)}, n_{blueprint})\}_{i=1}^m$
 - 19: $H \leftarrow \{\text{onehot}(x^{(i)})\}_{i=1}^m$ \triangleright one-hot encoding of the classes in X
 - 20: $Z \leftarrow \text{BE}(X)$
 - 21: $\hat{B} \leftarrow \text{BG}(Z)$
 - 22: $\mathcal{L}_{BD} \leftarrow \mathbb{E}_{p(\hat{B})}[\text{BD}(\hat{B}, H)] - \mathbb{E}_{p(B)}[\text{BD}(B, H)]$
 - 23: // Update parameters of BD
 - 24: $\theta_{BD} \leftarrow \theta_{BD} - \lambda_{BD} \cdot \text{RMSProp}(\theta_{BD}, \nabla_{\theta_{BD}} \mathcal{L}_{BD})$
 - 25: $\theta_{BD} \leftarrow \text{clip}(\theta_{BD}, -c, c)$
 - 26: // Update parameters of BE and BG
 - 27: **if** BE and BG can be updated according to n_{BD} **then**
 - 28: $\mathcal{L}_{prior} \leftarrow D_{KL}(q(Z|X)||p(Z))$
 - 29: $\mathcal{L}_{llike}^{points} \leftarrow -\mathbb{E}_{q(Z|\hat{B})}[\log p(\hat{B}|Z)]$
 - 30: $\mathcal{L}_{BG} \leftarrow -\mathbb{E}_{p(\hat{B})}[\text{BD}(\hat{B}, H)]$
 - 31: $\theta_{BE} \leftarrow \theta_{BE} - \lambda_{BE} \cdot \text{RMSProp}(\theta_{BE}, \nabla_{\theta_{BE}} (\mathcal{L}_{prior} + \mathcal{L}_{llike}^{points}))$
 - 32: $\theta_{BG} \leftarrow \theta_{BG} - \lambda_{BG} \cdot \text{RMSProp}(\theta_{BG}, \nabla_{\theta_{BG}} (\gamma \mathcal{L}_{llike}^{points} + \mathcal{L}_{BG}))$
 - 33: **end if**
 - 34: **end while**
 - 35: $GM \leftarrow \{\{\text{gaussian-mixture}(q(Z|X, \text{Class} = i))\}_{i=1}^{n_{classes}}\}$
 - 36: **end while**
-

7.3.2.2 Main body

Let θ_{BE} be the Blueprint Encoder’s parameters, θ_{BG} the Blueprint Generator’s parameters and θ_{BD} the Blueprint Discriminator’s parameters. θ_{BE} , θ_{BG} and θ_{BD} need to be initialised before executing the main training loop. Choosing a proper weight initialisation algorithm may be a daunting task as, over the years, an innumerable number of such methods have been proposed in the literature. This project recommends the use of the Xavier initialisation method [55] as it shows good performance when applied to the Shrinking Layers. However, the reader is free to experiment with other weight initialisation techniques, given that further research is needed in this regard.

The outer loop keeps iterating until θ_{BE} , θ_{BG} and θ_{BD} converge to optimality. The inner loop iterates over the training dataset, sampling a batch of data per iteration. Let $X \in R^{m \times N \times 3}$ denote a batch of m point clouds from the dataset. The Blueprint Encoder encodes X into a batch $Z \in R^{m \times P}$ of latent vectors. Subsequently, a batch $\hat{B} \in R^{m \times n_{blueprints} \times 3}$ of fake point cloud blueprints is generated by the Blueprint Generator given Z as input.

Given that the Blueprint Generator acts as a decoder component, then the i^{th} point cloud blueprint in \hat{B} needs to be a reconstruction of the corresponding point cloud blueprint of the i^{th} point cloud in X . As a consequence, the i^{th} point cloud blueprint in \hat{B} needs to be a point cloud blueprint instance of the same class as the i^{th} point cloud in X . Therefore, the Blueprint Discriminator needs to learn to distinguish between real and fake point cloud blueprints with respect to their class reference. This is the reason why the Blueprint Discriminator is trained using the RMSProp version of gradient descent by following the gradient error of the Wasserstein critic loss, as suggested in [13], but additionally computes the average score on the real point cloud blueprints and on the fake point cloud blueprints with respect to their class reference. The class reference is computed by producing H , a one-hot encoding matrix of the classes in X .

The average score on the fake point cloud blueprints with respect to their class reference is equivalent to $\mathbb{E}_{p(\hat{B})}[\text{BD}(\hat{B}, H)]$, in contrast, the average score on the real point cloud blueprints given their class reference cannot be computed directly from X . Thus, significant structural information needs to be extracted from the point clouds in X in order to produce their corresponding point cloud blueprints. Taking inspiration from [6], this is achieved by using the K-Means++ algorithm. The point clouds in X are partitioned into $n_{blueprints}$ regions, and each region’s centroid is utilised to produce a batch $B \in R^{m \times n_{blueprints} \times 3}$ of real point cloud blueprints. B allows computing the average score on the real point cloud blueprints given their reference class, $\mathbb{E}_{p(B)}[\text{BD}(B, H)]$. Weight clipping comes after the update of the Blueprint Discriminator’s parameters.

Following the suggestion in [13], the Blueprint Discriminator model is updated more times than the Blueprint Encoder and Generator according to n_{BD} . If the Blueprint Encoder and Generator can be updated, three loss functions,

inspired by [16], are instantiated.

\mathcal{L}_{prior} is the Kullback-Leibler divergence. It imposes a prior over the latent space $p(z)$ to regularise the Blueprint Encoder. $\mathcal{L}_{llike}^{points}$ is a point cloud distance function that serves as reconstruction loss. This project does not impose a specific point cloud distance metric; thus, the reader has freedom of choice in this regard. However, as a good starting point, this project recommends the use of the Chamfer Loss [56] or the Earth's Mover distance [14]. \mathcal{L}_{BG} is the Wasserstein generator loss.

The error signal for θ_{BE} comes from both $\mathcal{L}_{llike}^{points}$ and \mathcal{L}_{prior} . On the other hand, θ_{BG} is updated according to the gradients of the reconstruction error and the Wasserstein generator loss.

After the end of an epoch, the Gaussian Mixture dictionary is updated so that the i^{th} entry of the dictionary contains the learnt Gaussian Mixture Model of the i^{th} training class. The Gaussian Mixture Model for a given class c consists of K components $q(z|X_i)$ such that $q(z|X_i)$ is the distribution encoding learnt for the i^{th} point cloud training instance in training class c . K is the total number of training instances in training class c . Each component coefficient is assumed to be equiprobable, that is, each of them is $\frac{1}{K}$.

8 Final WGAN

8.1 Overview

Final WGAN is an ordinary WGAN and the second stage of the proposed architecture. It is concerned with the generation of a final point cloud with the guidance of a middle-level representation, a point cloud blueprint.

Specifically, starting from a point cloud blueprint, Final WGAN breeds a certain number of new points so to produce a complete point cloud that preserves the original structural information encoded in the blueprint.

8.2 Components

Final WGAN incorporates two components:

- Final Generator
- Final Discriminator

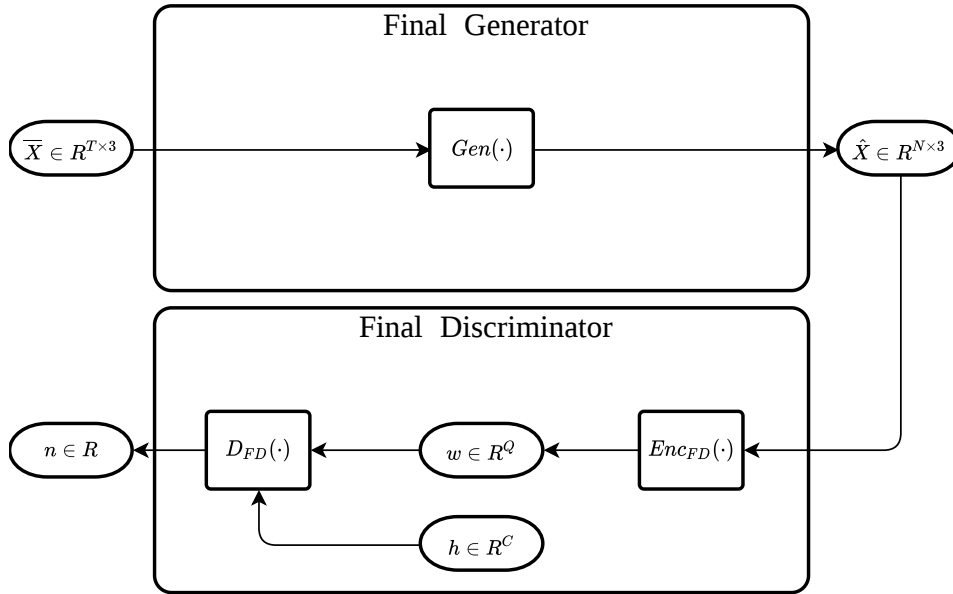


Figure 4: Final WGAN’s architecture

Figure 4 illustrates the architecture of the Final WGAN. The abovementioned components’ functionalities are meticulously described in the sections that follow.

8.2.1 Final Generator

8.2.1.1 Overview

The Final Generator produces a final point cloud with the guidance of a middle-level representation, a point cloud blueprint. Specifically, it breeds a certain number of points from a point cloud blueprint $\overline{X} \in R^{T \times 3}$ so that to produce a final point cloud $\hat{X} \in R^{N \times 3}$ that contains the same structural information as \overline{X} .

Formally, let $\overline{X} \in R^{T \times 3}$ be a point cloud blueprint, the component in question constructs a final point cloud $\hat{X} \in R^{N \times 3}$ from \overline{X} such that $B(\hat{X})$ and \overline{X} are approximately similar. $B(\hat{X})$ is \hat{X} 's corresponding point cloud blueprint.

8.2.1.2 Generative process

The generative process of the Final Generator relies on the function $Gen : R^{T \times 3} \rightarrow R^{N \times 3}$. The function $Gen(\cdot)$ takes as input a point cloud blueprint \overline{X} , breeds a certain number of points from every point in \overline{X} and returns a final point cloud \hat{X} . The function $Gen(\cdot)$ is equal to $E(\cdot|e_1, \dots, e_k)$, where E denotes a function comprising a sequence of K Extending Layers or Stack Extending Layers.

There are two main reasons why $Gen(\cdot) = E(\cdot|e_1, \dots, e_k)$. First of all, to the best of this author's knowledge, a sequence of Extending or Stack Extending Layers is the only generative proposal in the literature capable of exploiting self, local and global correlations to incrementally generate hierarchical graph embeddings. Secondly, the intrinsic properties of the Extending Layer make it suitable for the task of point cloud generation with the guidance of a middle-level representation. Indeed, the Extending Layer allows increasing the resolution of the semantic regions in a point cloud, which in this context translates to breeding new points from every structural point in a point cloud blueprint.

8.2.2 Final Discriminator

8.2.2.1 Overview

The Final Discriminator is tasked to distinguish between real point clouds sampled from the dataset and fake point clouds generated by the Final Generator. The realness or fakeness of a given point cloud is scored with respect to its class reference.

Formally, let $\hat{X} \in R^{N \times 3}$ be a point cloud and $h \in R^C$ the one-hot encoding vector of \hat{X} 's class, the Final Discriminator's duty is to score the realness or

fakeness of \hat{X} with respect to its class h . Notice that C stands for the number of classes in the training dataset.

As the Blueprint Discriminator, the Final Discriminator does not predict the probability of a point cloud \hat{X} as being real or fake but rather scores its realness or fakeness. The reason behind this behaviour is simple: this component is the discriminator unit of a Wasserstein GAN, Final WGAN.

A two-stage approach is adopted for the discriminative process of a point cloud $\hat{X} \in R^{N \times 3}$.

8.2.2.2 Discriminative process first stage

The first stage involves the encoding of a point cloud $\hat{X} \in R^{N \times 3}$ into a Q -dimensional vector $w \in R^Q$. The function $Enc_{FD} : R^{N \times 3} \rightarrow R^Q$ is utilised to achieve such a purpose.

$Enc_{FS}(\cdot)$ is equal to $S(\cdot|s_1, \dots, s_k)$, where S denotes a function comprising a sequence of K Shrinking Layers or Stack Shrinking Layers. The reason why $Enc_{FS}(\cdot) = S(\cdot|s_1, \dots, s_k)$ should be obvious by now since it is the same one for Enc_{BE} and Enc_{BD} in section 7.

8.2.2.3 Discriminative process second stage

In the second stage, the function $D_{FD} : (R^Q, R^C) \rightarrow R$ takes as input the encoding vector $w = Enc_{FD}(\hat{X})$ and \hat{X} 's class reference vector h . It outputs a real number $n \in R$ which scores the realness or fakeness of \hat{X} with respect to h . $D_{FD}(\cdot)$ is a MLP.

8.3 Training procedure

8.3.1 Overview

The Final WGAN training process is based on the WGAN algorithm [13]. However, the standard Wasserstein loss for the generator component is augmented with an additional additive term that measures the distance between the point cloud blueprints of the generated complete point clouds and the real point cloud blueprints provided as input.

This additional term is necessary in order to enforce that the generated complete final point clouds contain the same structural information as their point cloud blueprints input.

8.3.2 Algorithm

Algorithm 4 summarises the Final WGAN training procedure.

Algorithm 4 Final WGAN training algorithm

Require:

- 1: λ_{FG} : Final Generator(FG) learning rate.
- 2: λ_{FD} : Final Discriminator(FD) learning rate.
- 3: $n_{blueprint}$: number of points for the point cloud blueprints.
- 4: m : batch size.
- 5: c : clipping parameter.
- 6: n_{FD} : number of iterations of the discriminator per generator iteration.
- 7: γ : weighting style and content parameter

- 8:
- 9: $\theta_{FG} \leftarrow$ initialize FG parameters
- 10: $\theta_{FD} \leftarrow$ initialize FD parameters
- 11: **while** θ_{FG} and θ_{FD} have not converged **do**
- 12: **while** there are batches available **do**
- 13: $X \leftarrow$ a batch $\{x^{(i)}\}_{i=1}^m$ of point clouds from the real dataset
- 14: $H \leftarrow \{\text{onehot}(x^{(i)})\}_{i=1}^m \triangleright$ one-hot encoding of the classes in X
- 15: $\hat{B} \leftarrow$ sample point cloud blueprints
- 16: $\hat{X} \leftarrow \text{FG}(\hat{B})$
- 17: $\mathcal{L}_{FD} \leftarrow \mathbb{E}_{p(\hat{X})}[\text{FD}(\hat{X}, H)] - \mathbb{E}_{p(X)}[\text{FD}(X, H)]$
- 18: // Update parameters of FD
- 19: $\theta_{FD} \leftarrow \theta_{FD} - \lambda_{FD} \cdot \text{RMSProp}(\theta_{FD}, \nabla_{\theta_{FD}} \mathcal{L}_{FD})$
- 20: $\theta_{FD} \leftarrow \text{clip}(\theta_{FD}, -c, c)$
- 21: // Update parameters of FG
- 22: **if** FG can be updated according to n_{FD} **then**
- 23: $\bar{B} \leftarrow \{\text{Kmeans}(\hat{x}^{(i)}, n_{blueprint})\}_{i=1}^m$
- 24: $\mathcal{L}_{like}^{points} \leftarrow -\mathbb{E}_{q(Z|\bar{B})}[\log p(\bar{B}|Z)]$
- 25: $\mathcal{L}_{FG} \leftarrow -\mathbb{E}_{p(\hat{X})}[\text{BD}(\hat{X}, H)]$
- 26: $\theta_{FG} \leftarrow \theta_{FG} - \lambda_{FG} \cdot \text{RMSProp}(\theta_{FG}, \nabla_{\theta_{FG}} (\gamma \mathcal{L}_{like}^{points} + \mathcal{L}_{FG}))$
- 27: **end if**
- 28: **end while**
- 29: **end while**

8.3.2.1 Hyperparameters

Prior to executing the algorithm’s main body, several hyperparameters need to be set up.

λ_{FG} and λ_{FD} are the learning rates of the Final Generator and Final Discriminator, respectively. m is the batch size, while $n_{blueprint}$ is the number of points inside the point cloud blueprints.

Since this training process is based on the WGAN algorithm, then the clipping parameter c and the number of iterations of the discriminator per generator

iteration n_{FD} need to be set up as well.

Finally, γ is the style and content parameter [16]. It weights the ability of the Final Generator to generate complete point clouds that preserve the same structural information as their point cloud blueprints input vs fooling the Final Discriminator.

8.3.2.2 Main body

Let θ_{FG} be the Final Generator’s parameters and θ_{FD} the Final Discriminator’s parameters. θ_{FG} and θ_{FD} need to be initialised prior to the execution of the main training loop. As in section 7.3, this project recommends the use of the Xavier initialisation method, although the reader has freedom of choice in this respect.

The outer loop keeps iterating until θ_{FG} and θ_{FD} converge to optimality. The inner loop iterates over the training dataset, sampling a batch of data per iteration. Let $X \in R^{m \times N \times 3}$ denote a batch of m point clouds from the dataset and $\hat{B} \in R^{m \times n_{blueprint} \times 3}$ be a batch of point cloud blueprints such that the i^{th} complete point cloud in X and the i^{th} point cloud blueprint in \hat{B} are instances of the same training class. Notice that the computation of \hat{B} differs according to which stage of the CC3DVAEWGAN training process is in place, as discussed in section 6.3. The interested reader is referred to Appendix G for a better understanding of how \hat{B} ’s computation differ between the two training stages of CC3DVAEWGAN.

The Final Generator takes as input \hat{B} and returns a batch $\hat{X} \in R^{m \times N \times 3}$ of fake complete point clouds. Given that the Final Generator should breed new points and at the same time preserve the structural information of the point cloud blueprints, then the i^{th} generated complete point cloud in \hat{X} needs to be an instance of the same training class as the i^{th} point cloud in X .

Therefore, the Blueprint Discriminator needs to learn to distinguish between real and fake complete point clouds with respect to their class reference. This is the reason why the Blueprint Discriminator is trained using the RMSProp version of gradient descent by following the gradient error of the Wasserstein critic loss, as suggested in [13], but additionally computes the average score on the real complete point clouds and on the fake complete point clouds with respect to their class reference. The class reference is computed by producing H , a one-hot encoding matrix of the classes in X . Weight clipping comes after the update of the Blueprint Discriminator’s parameters.

In order to conform with [13], the Final Discriminator model is updated more times than the Final Generator according to the value of n_{BD} . If the Final Generator can be updated, two loss functions are instantiated.

\mathcal{L}_{FG} is the standard Wasserstein generator loss. $\mathcal{L}_{like}^{points}$ is a point cloud distance function that measures the distance between the point cloud blueprints input

\hat{B} and the point cloud blueprints corresponding to the generated complete point clouds \hat{X} . In other words, this loss measures the degree of dissimilarity between the actual generated complete point clouds' structural information and the expected one.

Thus, significant structural information needs to be extracted from the point clouds in \hat{X} in order to produce their corresponding point cloud blueprints. Taking inspiration from [6], this is achieved by using the K-Means++ algorithm. The point clouds in \hat{X} are partitioned into $n_{blueprints}$ regions, and each region's centroid is utilised to produce a batch $\overline{B} \in R^{m \times n_{blueprints} \times 3}$ of actual point cloud blueprints.

In the end, θ_{FG} is updated according to the gradients of \mathcal{L}_{FG} and $\mathcal{L}_{like}^{points}$.

9 Implementation

The point cloud layers proposed in this project, the CC3DVAEWGAN architecture and its training process are fully implemented using the Pytorch and Pytorch Geometric libraries. These implementations can be slightly improved in order to decrease their time complexity. Notice that the Stack Extending Layer has not been implemented yet as its theoretical development came at a late stage of this project. For further information on the implementation details and the recommended improvements, the reader shall consult Appendix H.

10 Experiments

Several experiments have been conducted to verify the validity of the proposed solutions. However, further experiments are necessary to figure out the actual potential capabilities of these approaches.

Experiments on the Shrinking and Stack Shrinking Layers show that the latter can become the next state-of-the-art point cloud feature extractor.

K-Means++, the algorithm used for the extraction of point cloud blueprints from complete point clouds, produces point cloud skeletons that visually seem to resemble the most significant structural information of the complete point clouds.

The Extending Layer demonstrates to achieve conditional generation, and its performance is comparable to its competitors when trained via the Chamfer loss.

Therefore, much better results should be obtained when training the Extending Layer together with a Stack Shrinking Layer-based discriminator, leveraging the Final WGAN architecture. However, this does not occur in practice due to a GAN convergence problem.

The Blueprint VAE-WGAN architecture is affected by the same problem; thus, it is not known yet whether the latter can simultaneously learn a latent space distribution for each training class and a function mapping a given latent representation to its corresponding point cloud blueprint.

Further research is necessary to solve these GAN convergence issues. Preliminary analysis shows that the source of these issues might be unproportionate components' capacities, which prevents the attainment of the Nash equilibrium. Specifically, the number of parameters in the generator components is much smaller than the number of parameters in the discriminator components. Hence, more Extending Layers should be stacked horizontally to increase the generator components' capabilities.

For an extensive description of the conducted experiments, the reader is referred to Appendix I

11 Project management

For information regarding this project's planning, progress, time management, and contingency planning, the reader is referred to Appendix J.

12 Conclusion

This project presents a novel two-stage architecture aiming to achieve conditional and controllable generation of point clouds. Moreover, a new set of neural network layers capable of encoding and breeding point clouds leveraging self, local and non-local correlations is also proposed. These approaches have been implemented using Pytorch and Pytorch Geometric.

Results show that the proposed point cloud layers attain their objectives and could potentially achieve state-of-the-art performance. Nevertheless, the results obtained are not as expected when the proposed layers are embedded in the CC3DVAEWGAN architecture.

The source of this failure lies in a common GAN training issue known as GAN convergence problem. Preliminary analysis shows that an unproportionate use of components' capacities might have been used in the experiments, which has prevented the attainment of the Nash equilibrium. Thus, further research is necessary to solve this GAN convergence issue.

Additionally, research is also necessary to figure out the optimal stacking depth and hyperparameters of the proposed point cloud layers. The Stack Extending Layer still needs to be implemented and its performance empirically evaluated.

This project strongly believes that, theoretically, the solutions proposed are real breakthroughs in the field of point cloud feature extraction and generation. With further experimentations, empirical results are very likely to confirm the proposed solutions' advanced nature.

References

- [1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: (), pp. 696–699.
- [2] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).
- [3] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144. ISSN: 0001-0782.
- [4] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (2016), pp. 1–16. arXiv: [1511.06434](https://arxiv.org/abs/1511.06434).
- [5] Tong Che et al. “Mode regularized generative adversarial networks”. In: *arXiv preprint arXiv:1612.02136* (2016).
- [6] Ximing Yang et al. “CPCGAN: A Controllable 3D Point Cloud Generative Adversarial Network with Semantic Label Generating”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 4. 2021, pp. 3154–3162. ISBN: 2374-3468.
- [7] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).
- [8] Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.
- [9] Wenkai Han et al. “Point2Node: Correlation learning of dynamic-node for point cloud feature modeling”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 10925–10932. ISBN: 2374-3468.
- [10] John F Nash. “Equilibrium points in n-person games”. In: *Proceedings of the National Academy of Sciences* 36.1 (1950), pp. 48–49. ISSN: 0027-8424. DOI: [10.1073/pnas.36.1.48](https://doi.org/10.1073/pnas.36.1.48). URL: <https://www.pnas.org/content/36/1/48>.
- [11] Farzan Farnia and Asuman Ozdaglar. “Gans may have no nash equilibria”. In: *arXiv preprint arXiv:2002.09124* (2020).
- [12] Martin Arjovsky and Léon Bottou. “Towards principled methods for training generative adversarial networks”. In: *arXiv preprint arXiv:1701.04862* (2017).
- [13] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein generative adversarial networks”. In: *International conference on machine learning*. PMLR, 2017, pp. 214–223. ISBN: 2640-3498.
- [14] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. “The earth mover’s distance as a metric for image retrieval”. In: *International journal of computer vision* 40.2 (2000), pp. 99–121. ISSN: 1573-1405.

- [15] Irina Higgins et al. “Early Visual Concept Learning with Unsupervised Deep Learning”. In: (2016). arXiv: [1606.05579](https://arxiv.org/abs/1606.05579). URL: <http://arxiv.org/abs/1606.05579>.
- [16] Anders Boesen Lindbo Larsen et al. “Autoencoding beyond pixels using a learned similarity metric”. In: *International conference on machine learning*. PMLR, 2016, pp. 1558–1566.
- [17] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.
- [18] Jianxin Lin et al. “Conditional image-to-image translation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 5524–5532.
- [19] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. “Singan: Learning a generative model from a single natural image”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 4570–4580.
- [20] Panos Achlioptas et al. “Learning representations and generative models for 3d point clouds”. In: *International conference on machine learning*. PMLR, 2018, pp. 40–49. ISBN: 2640-3498.
- [21] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. “Learning localized generative models for 3d point clouds via graph convolution”. In: *International conference on learning representations*. 2018.
- [22] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. “3d point cloud generative adversarial network based on tree structured graph convolutions”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3859–3868.
- [23] Xiaolong Wang and Abhinav Gupta. “Generative image modeling using style and structure adversarial networks”. In: *European conference on computer vision*. Springer, 2016, pp. 318–335.
- [24] Yann LeCun et al. “Object Recognition with Gradient-Based Learning”. In: *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer-Verlag, 1999, p. 319. ISBN: 3540667229.
- [25] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature Cell Biology* 521.7553 (May 2015), pp. 436–444. ISSN: 1465-7392. DOI: [10.1038/nature14539](https://doi.org/10.1038/nature14539).
- [26] Daniel Maturana and Sebastian Scherer. “3d convolutional neural networks for landing zone detection from lidar”. In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 3471–3478. ISBN: 1479969230.
- [27] Daniel Maturana and Sebastian Scherer. “Voxnet: A 3d convolutional neural network for real-time object recognition”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 922–928. ISBN: 1479999946.

- [28] Charles R Qi et al. “Volumetric and multi-view cnns for object classification on 3d data”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5648–5656.
- [29] Cheng Wang et al. “NormalNet: A voxel-based CNN for 3D object classification and retrieval”. In: *Neurocomputing* 323 (2019), pp. 139–147. ISSN: 0925-2312.
- [30] Sambit Ghadai et al. “Multi-resolution 3D convolutional neural networks for object recognition”. In: *arXiv preprint arXiv:1805.12254* 4 (2018).
- [31] Hang Su et al. “Multi-view convolutional neural networks for 3d shape recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 945–953.
- [32] Biao Leng et al. “3D object retrieval with stacked local convolutional autoencoder”. In: *Signal Processing* 112 (2015), pp. 119–128. ISSN: 0165-1684.
- [33] Song Bai et al. “Gift: A real-time and scalable 3d shape search engine”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5023–5032.
- [34] Evangelos Kalogerakis et al. “3D shape segmentation with projective convolutional networks”. In: *proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3779–3788.
- [35] Zhangjie Cao, Qixing Huang, and Ramani Karthik. “3D object classification via spherical projections”. In: *2017 international conference on 3D Vision (3DV)*. IEEE, 2017, pp. 566–574. ISBN: 1538626101.
- [36] Le Zhang, Jian Sun, and Qiang Zheng. “3D Point Cloud Recognition Based on a Multi-View Convolutional Neural Network”. In: *Sensors* 18.11 (2018), p. 3681.
- [37] Charles R Qi et al. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *arXiv preprint arXiv:1706.02413* (2017).
- [38] Yin Zhou and Oncel Tuzel. “Voxelnet: End-to-end learning for point cloud based 3d object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4490–4499.
- [39] Jiaxin Li, Ben M Chen, and Gim Hee Lee. “So-net: Self-organizing network for point cloud analysis”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9397–9406.
- [40] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. “Pointwise convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 984–993.
- [41] Yangyan Li et al. “Pointcnn: Convolution on x-transformed points”. In: *Advances in neural information processing systems* 31 (2018), pp. 820–830.

- [42] Hengshuang Zhao et al. “Pointweb: Enhancing local neighborhood features for point cloud processing”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (2019), pp. 5560–5568. ISSN: 10636919. DOI: [10.1109/CVPR.2019.00571](https://doi.org/10.1109/CVPR.2019.00571).
- [43] Wenxuan Wu, Zhongang Qi, and Li Fuxin. “Pointconv: Deep convolutional networks on 3d point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9621–9630.
- [44] Yongcheng Liu et al. “Relation-shape convolutional neural network for point cloud analysis”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8895–8904.
- [45] Shiyi Lan et al. “Modeling local geometric structure of 3d point clouds using geo-cnn”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 998–1008.
- [46] Artem Komarichev, Zichun Zhong, and Jing Hua. “A-cnn: Annularly convolutional neural networks on point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7421–7430.
- [47] Yifan Xu et al. “Spidercnn: Deep learning on point sets with parameterized convolutional filters”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 87–102.
- [48] Jinxian Liu et al. “Dynamic points agglomeration for hierarchical point sets learning”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 7546–7555.
- [49] Jiancheng Yang et al. “Modeling point clouds with self-attention and gumbel subset sampling”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3323–3332.
- [50] Roman Klokov and Victor Lempitsky. “Escape from cells: Deep kd-networks for the recognition of 3d point cloud models”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 863–872.
- [51] Yue Wang et al. “Dynamic graph cnn for learning on point clouds”. In: *Acm Transactions On Graphics (tog)* 38.5 (2019), pp. 1–12. ISSN: 0730-0301.
- [52] Chu Wang, Babak Samari, and Kaleem Siddiqi. “Local spectral graph convolution for point set feature learning”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 52–66.
- [53] Xi Chen et al. “Infogan: Interpretable representation learning by information maximizing generative adversarial nets”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 2180–2188.

- [54] Tero Karras, Samuli Laine, and Timo Aila. “A style-based generator architecture for generative adversarial networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4401–4410.
- [55] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.
- [56] Haoqiang Fan, Hao Su, and Leonidas J Guibas. “A point set generation network for 3d object reconstruction from a single image”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613.
- [57] Özgün Çiçek et al. “3D U-Net: learning dense volumetric segmentation from sparse annotation”. In: *International conference on medical image computing and computer-assisted intervention*. Springer, 2016, pp. 424–432.
- [58] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.
- [59] Xiaolong Wang et al. “Non-local neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7794–7803.
- [60] Martin Simonovsky and Nikos Komodakis. “Dynamic edge-conditioned filters in convolutional neural networks on graphs”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3693–3702.
- [61] David Arthur and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding*. Tech. rep. Stanford, 2006.
- [62] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *Advances in neural information processing systems* 30 (2017).
- [63] Zhirong Wu et al. “3d shapenets: A deep representation for volumetric shapes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.
- [64] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. “Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5010–5019.
- [65] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [66] Angel X Chang et al. “Shapenet: An information-rich 3d model repository”. In: *arXiv preprint arXiv:1512.03012* (2015).

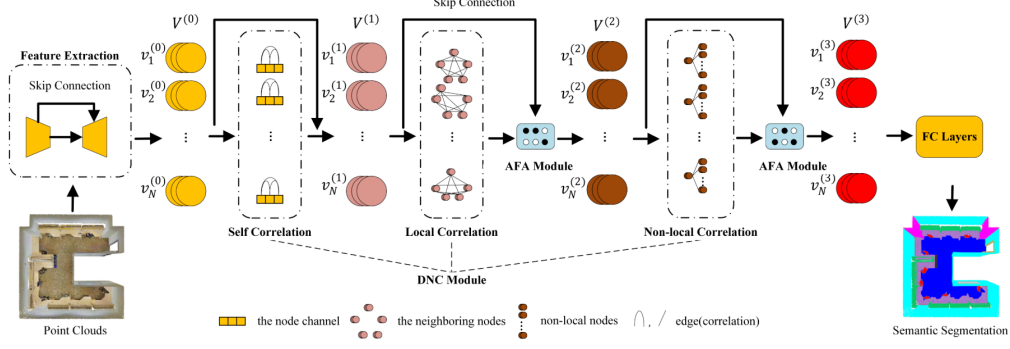


Figure 5: Point2Node architecture [9]

Appendices

A Point2Node

A.1 Overview

Point2Node is a high-dimensional node graph model capable of exploring self-correlations, local-correlations and non-local correlations between points with the help of a Dynamic Node Correlation Module(DNC). The DNC module learns self, local and non-local correlations sequentially and updates the nodes after each correlation learning to enhance their characteristics. Additionally, Point2Node makes use of a gate embedding, called Adaptive Feature Aggregation(AFA), to balance self, local and non-local correlations. Specifically, the AFA module adaptively allows some channels of high-dimensional nodes to pass to the next correlation learning level while preventing others from proceeding.

A.2 Architecture

The Point2Node architecture is illustrated in Fig 5. Let $P \in R^{N \times 3}$ be a 3D point cloud containing N three-dimensional points. The Feature Extraction module takes P as input and outputs a high-level representation $V^{(0)} \in R^{N \times C}$, where C is the new dimension of each node. The Feature Extraction module employs a U-Net [57] and the \mathcal{X} -Conv operator [41] for the feature learning task. Afterwards, the DNC module creates high-level representations $V^{(1)}$, $V^{(2)}$ and $V^{(3)}$ by learning self, local and non-local correlations, respectively. Notice that after each correlation learning stage, a AFA module aggregates the learnt features in a self-adaptive fashion. To conclude, $V^{(3)}$ is given as input to a MLP which performs the classification task.

A.3 Dynamic Node Correlation Module

The DNC module learns three different kinds of correlations: self correlation, local correlation and non-local correlation. Self correlation explores the edges between channels of the same point. Local correlation discovers local dependencies by inspecting edges between neighbouring nodes. Non-local correlation exploits edges between spatially distant nodes to learn long-distance relations. In the sections that follow, the learning mechanism for each of these correlations is extensively described.

A.3.1 Self Correlation

In order to explore self-correlation, edges between the different channels of a node should be constructed and a weight for each edge learnt. After the correlation learning, a node’s channels should be updated according to the learnt weights. However, due to time-complexity issues, the authors of Point2Node use a MLP with 1D separable convolutions [58] to encode all the weights from different channels into one. As a consequence, the channel weight $w_i \in R^C$ for a given node $v_i^{(0)} \in V^{(0)}$ is computed as follows:

$$w_i = f(v_i^{(0)})$$

where f is a MLP with 1D separable convolutions.

The weight vector is successively normalised so that all the scalar weights for each channel are comparable:

$$\overline{w_{i,c}} = \frac{e^{w_{i,c}}}{\sum_{l=1}^C e^{w_{i,l}}}$$

where $w_{i,c}$ is the c^{th} scalar value of w_i .

After the correlation learning, a node’s channels are updated with the equation:

$$v_i^{(1)} = v_i^{(0)} \oplus \lambda(\overline{w_i} \odot v_i^{(0)})$$

where λ is a parameter to be learnt.

The self-correlation update transforms $V^{(0)}$ into $V^{(1)}$.

A.3.2 Local Correlation

The local-correlation sub-module captures local dependencies with the help of a local graph. The strength of the correlation between neighbouring nodes is measured through the local graph adjacency matrix weights.

The dilated KNN [41] is utilized to construct the local graph. Specifically, each node in $V^{(1)}$ is considered as a centroid and a Dilated K-Nearest Neighbor Graph is formed on top of it. Thus, the local graph $G^{(l)}$ is a set of N sub-graphs:

$$G^{(l)} = \{G_i^{(l)} \mid G_i^{(l)} \in R^{K \times C}, i = 1, \dots, N\}$$

The approach taken by the authors to construct the adjacency matrix $m_i^{(l)} \in R^{K \times K}$ for each Dilated-K-Nearest Neighbor Graph $G_i^{(l)}$ is inspired by [59]. Let $V_i^{(l)} = \{V_i^{(l)}(k) \mid V_i^{(l)}(k) \in R^C, k = 1, \dots, K\}$ be the neighbors of node $v_i^{(1)}$, $m_i^{(l)}$ is computed with the formula below:

$$m_i^{(l)} = \theta^{(l)}(V_i^{(l)}) \beta^{(l)}(V_i^{(l)})^T$$

where $\theta^{(l)}$ and $\beta^{(l)}$ are two 1D convolutional MLPs taking as input a set of neighbors and outputting a $R^{K \times \frac{C}{r}}$ matrix. r is a hyperparameter.

Successively, the adjacency matrix $m_i^{(l)}$ is pruned in order to block redundant connections. This results in the new adjacency matrix $\overline{m_i^{(l)}}$:

$$\overline{m_i^{(l)}}(x, y) = \frac{e^{m_i^{(l)}(x, y)}}{\sum_{k=1}^K e^{m_i^{(l)}(x, k)}}$$

where $m_i^{(l)}(x, y)$ denotes the weight on the edge between the x^{th} and y^{th} nodes in $V_i^{(l)}$.

Having learnt the local relations in the neighboring set $V_i^{(l)}$, the centroid $v_i^{(1)}$ can be updated. The update procedure involves updating the neighbouring nodes in $V_i^{(l)}$ first by means of the adjacency matrix $\overline{m_i^{(l)}}$:

$$V_{i \text{ up}}^l = \overline{m_i^{(l)}} V_i^{(l)}$$

The new local information is encoded into the centroid node $v_i^{(1)}$:

$$v_{i \text{ up}}^{(1)} = \max(V_{i \text{ up}}^l)$$

where \max is the max pooling function.

A.3.3 Non-local Correlation

The non-local correlation sub-module builds a global graph so that to capture long-range relations with the help of the graph's adjacency matrix. Given the high-level representation $V^{(2)}$ and a global graph $G^{(g)}$, the global graph adjacency matrix $m^{(g)} \in R^{N \times N}$ is formed in a similar fashion as in the local correlation submodule:

$$m_i^{(g)} = \theta^{(g)}(V^{(2)}) \beta^{(g)}(V^{(2)})^T$$

where $\theta^{(g)}$ and $\beta^{(g)}$ are two 1D convolutional MLPs taking as input the nodes $V^{(2)}$ and outputting a $R^{N \times \frac{C}{r}}$ matrix. r is a hyperparameter.

As distant nodes may have features in different scales, the adjacency matrix $m^{(g)}$ is normalised. This leads to the new adjacency matrix $\overline{m^{(g)}}$:

$$\overline{m^{(g)}}(x, y) = \frac{e^{m^{(g)}(x, y)}}{\sum_{k=1}^K e^{m^{(g)}(x, k)}}$$

where $m^{(g)}(x, y)$ denotes the weight on the edge between the x^{th} and y^{th} nodes in $V^{(2)}$

Thereafter, all nodes in $V^{(2)}$ are updated:

$$V_{up}^{(2)} = \overline{m^{(g)}} V^{(2)}$$

A.4 Adaptive Feature Aggregation Module

The Adaptive Feature Aggregation(AFA) Module aggregates features from different correlation learnings in a data-adaptive fashion through the use of a gate mechanism.

A.4.1 Parameter-free Characteristic Modeling

To start, given two high-level representations V^n and V_{up}^n coming from different correlation learnings, their characteristics are described using a non-parametric model by simply computing the average high-level representation. Let $s_1 \in R^{1 \times C}$ and $s_2 \in R^{1 \times C}$ be the average node in V^n and V_{up}^n , respectively, then s_1 and s_2 are computed as follows:

$$s_1 = \frac{1}{N} \sum_{i=1}^N v_i^n$$

$$s_2 = \frac{1}{N} \sum_{i=1}^N v_{i \text{ } up}^n$$

where v_i^n is the i^{th} node in V^n and $v_{i \text{ } up}^n$ is the i^{th} node in V_{up}^n

A.4.2 Parameterized Characteristic Modeling

Two MLPs, consisting of 1D reduction convolution and 1D increasing convolution layers, are employed to explore further the characteristics of V^n and

V_{up}^n . Precisely, these two MLPs map the average nodes s_1 and s_2 into new descriptors $z_1 \in R^{1 \times C}$ and $z_2 \in R^{1 \times C}$, respectively:

$$\begin{aligned} z_1 &= f_1(s_1) \\ z_2 &= f_2(s_2) \end{aligned}$$

where f_1 and f_2 are the two MLPs consisting of 1D reduction convolution and 1D increasing convolution layers.

A.4.3 Gate Mechanism

Typically, neural networks use linear aggregations to combine different features. However, these aggregations are not data-adaptive. Instead, the authors of Point2Node propose a non-linear and data-aware gate mechanism to aggregate features from different correlation learnings. This gate mechanism applies the softmax function to the descriptors z_1 and z_2 :

$$\begin{aligned} m_{1,c} &= \frac{e^{z_{1,c}}}{e^{z_{1,c}} + e^{z_{2,c}}} \\ m_{2,c} &= \frac{e^{z_{2,c}}}{e^{z_{1,c}} + e^{z_{2,c}}} \end{aligned}$$

where $m_{1,c}$ and $m_{2,c}$ are the masked z_1 and z_2 on the c^{th} channel.

A.4.4 Aggregation

With the help of m_1 and m_2 , the high-level representations V^n and V_{up}^n are aggregated forming V^{n+1} :

$$v_i^{n+1} = m_1 \cdot v_i^n + m_2 \cdot v_{i_{up}}^n$$

where v_i^{n+1} is the i^{th} node in the high-level representation V^{n+1} .

B Shrinking layer

B.1 Overview

The Shrinking Layer is an innovative graph-based neural network module designed to extract features from a point cloud. Although it leverages self, local and global point correlations as proposed in Point2Node [9], it ultimately differs from the latter as it embeds ideas borrowed from the CNN literature [24, 25]. Since CNNs achieve state-of-the-art performance in image feature extraction, it seems reasonable to design their point cloud counterpart.

Other approaches, such as [60, 51], emulate convolution-like operations in the point cloud domain by constructing graphs in the Euclidean space and applying graph convolution operations. While the main advantage deriving from these approaches is the reduction in the degree of freedom in the trained models due to weight sharing, the major drawback is the incapability of extracting features beyond local and global dependencies.

The Shrinking Layer also employs a graph convolution operation [21] to capture local relations, though it builds graphs in the Euclidean space in a unique manner. Furthermore, it also uses the self-correlation operation and the Adaptive Feature Aggregation(AFA) module [9] to learn the self-correlation of each point and aggregate points from different correlation learnings in a data-aware manner, respectively.

Besides being a novel graph convolutional neural network and adjacency matrix hybrid solution, the Shrinking Layer is the first point cloud feature extractor that takes advantage of the entire array of CNNs basic ideas: *local receptive fields*, *shared weights*, and *pooling*.

B.2 Contributions

The main contributions of the Shrinking Layer are:

- A novel graph convolutional neural network and adjacency matrix module capable of exploring self-correlations, local-correlations and non-local correlations between points in point clouds.
- A new point cloud feature extractor that employs the whole array of CNNs basic ideas: local receptive fields, shared weights, and pooling.
- A permutation invariant point cloud feature extractor.
- A fully differentiable neural network module that can be plugged into existing deep neural network models to improve their performance.

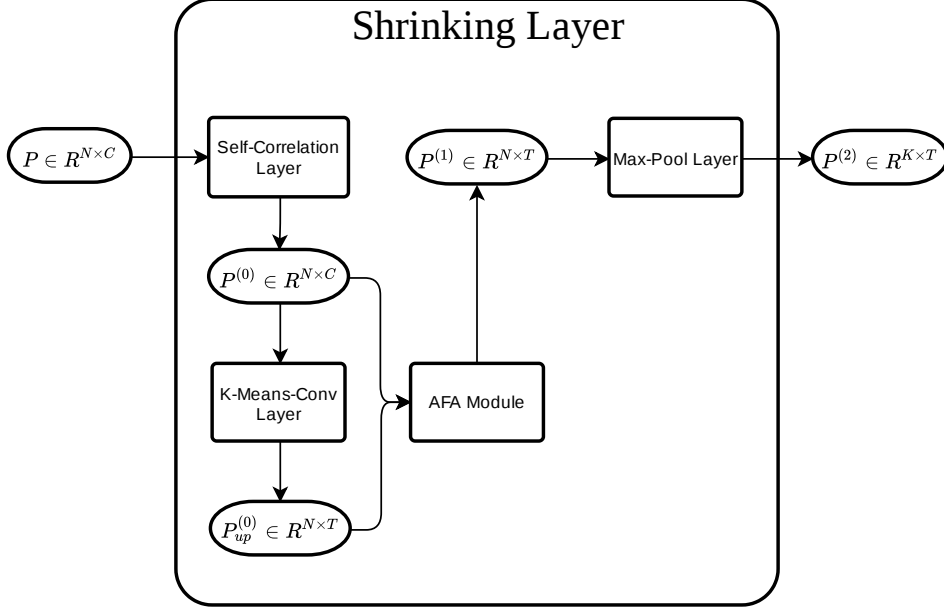


Figure 6: Shrinking Layer architecture

- A neural network module that can be stacked horizontally to capture higher-level representations incrementally like in CNNs and consequently explore global point correlations as well.
- A module that learns how to group points dynamically by updating the graph of relationships from layer to layer.
- A novel graph convolution operation, dubbed K-Means Convolution, that convolves points in a semantic-based fashion.

B.3 Architecture

The Shrinking Layer architecture is illustrated in Figure 6. Let $P \in R^{N \times C}$ be a point cloud containing N C -dimensional points. The Self-Correlation Layer takes P as input, learns the self-correlation of each point and updates each node accordingly, generating a feature point cloud output $P^{(0)} \in R^{N \times C}$.

The K-Means-Conv Layer performs a novel operation, named K-Means Convolution, to capture local dependencies in $P^{(0)}$. It clusters the nodes into K regions and performs a graph convolution operation on each node such that the neighbourhood of a node x is the set of nodes that belong to the same region as x . The output of the K-Means-Conv Layer is a feature point cloud $P_{up}^{(0)} \in R^{N \times T}$, where $T \geq C$. The dimensionality of the nodes should be increased after the convolution operation as every node needs to encode its features as well as the characteristics of its neighbourhood.

Successively, the Adaptive Feature Aggregation Module [9] constructs a feature

point cloud $P^{(1)} \in R^{N \times T}$ by aggregating the features in $P^{(0)}$ and $P_{up}^{(0)}$ in a data-adaptive fashion through the use of a gate mechanism.

Finally, the Max-Pool Layer receives $P^{(1)}$ and outputs a condensed feature point cloud $P^{(2)} \in R^{K \times T}$, where K is the number of regions clustered in the K-Means-Conv Layer. Fundamentally, the Max-Pool layer encodes every region in $P^{(1)}$ with a unique node by applying the max-pooling operation to the set of nodes belonging to that region. A region in $P^{(1)}$ is defined as the ancestor region of a region clustered in $P^{(0)}$ during the K-Means Convolution operation. In other words, every node in a region of $P^{(1)}$ is the ancestor of a node in the corresponding region in $P^{(0)}$.

In what follows, the learning mechanism for each of these internal layers is extensively explained.

B.3.1 Self-Correlation Layer

In order to explore self-correlation, edges between the different channels of a node should be constructed and a weight for each edge learnt. After, a node's channels should be updated according to the learnt weights.

However, the Self-Correlation Layer adopts the technique proposed in [9] to encode all the weights from different channels into one so to avoid time-complexity issues. Consequently, the channel weight $w_i \in R^C$ for a given node $p_i \in P$ is computed as follows:

$$w_i = f(p_i)$$

where f is a MLP.

Unlike in Point2Node, f does not need to be a MLP with 1D separable convolutions. Hence, the reader can opt for any MLP architecture according to the problem at hand.

The weight vector is successively normalised so to make all the scalar weights for each channel comparable:

$$\overline{w_{i,c}} = \frac{e^{w_{i,c}}}{\sum_{l=1}^C e^{w_{i,l}}}$$

where $w_{i,c}$ is the c^{th} scalar value of w_i .

After the correlation learning, the following equation is used to update a node's channels:

$$p_i^{(0)} = p_i \oplus \lambda (\overline{w_i} \odot p_i)$$

where λ is a parameter to be learnt.

The Self-Correlation Layer transforms P into $P^{(0)}$.

B.3.2 K-Means-Conv Layer

The K-Means-Conv Layer explores local geometric structures by constructing a local neighbourhood graph via K-Means++ [61] and applying a modified version of the graph convolution operation in [21]. This convolution operation, dubbed K-Means Convolution, is in the spirit of graph neural networks.

Given a parameter K for the number of regions to be clustered, a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, representing local structures, is computed from $P^{(0)}$ with the help of the K-Means++ algorithm. \mathcal{V} denotes the set of nodes in $P^{(0)}$, while the set of directed edges \mathcal{E} is defined as

$$\mathcal{E} = \{ (p_i^{(0)}, p_j^{(0)}) \mid p_i^{(0)}, p_j^{(0)} \in P^{(0)} \text{ and } p_i^{(0)}, p_j^{(0)} \text{ share the same centroid} \}$$

That is, \mathcal{G} is a collection of K disjoint directed sub-graphs such that every two nodes in each sub-graph share the same centroid computed via the K-Means++ algorithm. Additionally, each sub-graph is a complete graph containing self-loops. K-Means++ is used, instead of the usual K-Means, because it achieves better clustering performance and reduces the variance of the clustered regions between successive runs in the same point cloud.

Subsequently, each node $p_i^{(0)}$ in \mathcal{G} is updated with the following graph convolution operation:

$$p_{i_{up}}^{(0)} = \sigma \left(\sum_{j \in \mathcal{N}_i} \frac{\mathcal{F}(p_j^{(0)} - p_i^{(0)}) p_j^{(0)}}{\mathcal{M}(p_i^{(0)})} + \mathcal{W}(p_i^{(0)}) p_i^{(0)} + b \right)$$

where \mathcal{N}_i is the neighborhood of node $p_i^{(0)}$, \mathcal{F} is a $R^C \rightarrow R^{T \times C}$ MLP, \mathcal{W} is a $R^C \rightarrow R^{T \times C}$ MLP, \mathcal{M} is a $R^C \rightarrow R$ MLP, b is a R^T vector and σ is a non-linear activation function.

The reader is free to select any architecture for the above MLPs according to the problem at hand. It is important to notice that the value of $\mathcal{F}(\cdot, \cdot)$ is the same for pairs of nodes having the same difference, even if they are in different regions. This represents a substantial advantage as it creates weight sharing like in CNNs and consequently reduces the number of degrees of freedom of the model.

The K-Means-Conv Layer outputs $P_{up}^{(0)}$.

B.3.3 Adaptive Feature Aggregation Module

The Adaptive Feature Aggregation(AFA) Module aggregates the features in $P^{(0)}$ and $P_{up}^{(0)}$ in a data-adaptive fashion through the use of a gate mechanism.

B.3.3.1 Parameter free Characteristic Modelling

Given $P^{(0)}$ and $P_{up}^{(0)}$, their characteristics are described by simply computing the average node s_1 and s_2 , respectively:

$$s_1 = \frac{1}{N} \sum_{i=1}^N p_i^{(0)}$$

$$s_2 = \frac{1}{N} \sum_{i=1}^N p_{i\ up}^{(0)}$$

where $p_i^{(0)}$ is the i^{th} node in $P^{(0)}$ and $p_{i\ up}^{(0)}$ is the i^{th} node in $P_{up}^{(0)}$

B.3.3.2 Parameterized Characteristic Modeling

Two MLPs map the average nodes s_1 and s_2 into new descriptors $z_1 \in R^{1 \times C}$ and $z_2 \in R^{1 \times C}$, respectively:

$$z_1 = f_1(s_1)$$

$$z_2 = f_2(s_2)$$

where f_1 and f_2 are the two MLPs.

Unlike in [9], f_1 and f_2 are not constrained to be MLPs consisting of 1D reduction convolution and 1D increasing convolution layers. Thus, the reader can opt for any architecture at his will.

B.3.3.3 Gate Mechanism

A gate mechanism applies the softmax function to the descriptors z_1 and z_2 :

$$m_{1,c} = \frac{e^{z_{1,c}}}{e^{z_{1,c}} + e^{z_{2,c}}}$$

$$m_{2,c} = \frac{e^{z_{2,c}}}{e^{z_{1,c}} + e^{z_{2,c}}}$$

where $m_{1,c}$ and $m_{2,c}$ are the masked z_1 and z_2 on the c^{th} channel.

B.3.3.4 Aggregation

With the help of m_1 and m_2 , $P^{(0)}$ and $P_{up}^{(0)}$ are aggregated forming $P^{(1)}$:

$$p_i^{(1)} = m_1 \cdot p_i^{(0)} + m_2 \cdot p_{i\ up}^{(0)}$$

where $p_i^{(1)}$ is the i^{th} node in $P^{(1)}$.

B.3.4 Max-Pool Layer

The Max-Pool Layer applies the max-pooling operation to encode every region in $P^{(1)}$ with a single node. First, a directed graph $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ is computed from $P^{(1)}$ such that \mathcal{V}_2 is the set of nodes in $P^{(1)}$ and \mathcal{E}_2 is defined as

$$\mathcal{E}_2 = \{ (p_i^{(1)}, p_j^{(1)}) \mid p_i^{(1)}, p_j^{(1)} \in P^{(1)} \text{ and } (anc(p_i^{(0)}), anc(p_j^{(0)})) \in \mathcal{E} \}$$

In other words, \mathcal{G}_2 is a collection of K disjoint directed sub-graphs such that for every pair of nodes in each sub-graph, their ancestor nodes are adjacent nodes in \mathcal{G} . Remember that \mathcal{G} is the directed graph produced during the K-Means Convolution operation.

Subsequently, for every sub-graph $\mathcal{G}_2^{(i)}$ in \mathcal{G}_2 , a single node $p_i^{(2)}$, encoding the whole region, is produced by means of the max-pooling operation:

$$p_i^{(2)} = \max(\mathcal{G}_2^{(i)})$$

where \max computes the max-pooling operation along each dimension of the nodes in $\mathcal{G}_2^{(i)}$.

Notice that $p_i^{(2)}$ will be the i^{th} point in $P^{(2)}$. The Max-Pool Layer returns the feature point cloud $P^{(2)}$.

B.4 Rationale behind the architecture

The overwhelming success of CNNs in image feature extraction suggests the value of adapting insight from the CNN world to the point cloud domain. This is the reason why CNNs heavily inspire the Shrinking Layer’s architecture.

Other CNN-like point cloud approaches have been proposed in the literature [60, 51], but unlike them, the method proposed here is semantic-based and fully leverages the three basic ideas underlying CNNs: *local receptive fields*, *shared weights*, and *pooling*. The local receptive fields and shared weights concepts are exploited in the K-Means-Conv Layer, while pooling is performed in the last layer, Max-Pool.

The K-Means Convolution operation convolves every node in a feature point cloud with respect to its neighbourhood using the same weights. The computation of each node’s neighbourhood amounts to computing different local receptive fields, while the use of shared weights translates to applying the same feature detector everywhere in the point cloud. As a result, the K-Means-Convolution operation tries to detect the same feature at different locations, making it translation invariant. The operator is semantic-based because the points are convolved with respect to their neighbourhoods, which in turn are computed via K-Means++. This means that every point is only updated according to the points in the same semantic region.

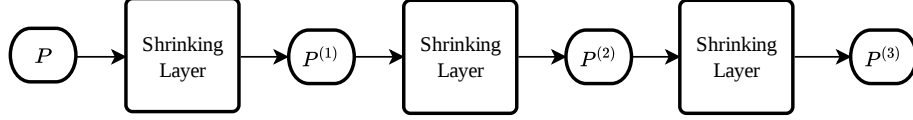


Figure 7: Architecture of a horizontal stack containing three Shrinking Layers.

In contrast, other approaches [60, 51] compute the neighbourhood of each node via K-Nearest-Neighbor. Consequently, the update of a node can get spurious information from nodes not belonging to the same semantic region.

Successively to the convolution operation, the Max-Pool Layer condenses all the features extracted in a region into a single feature.

This whole process is the reason why the architecture proposed here is named Shrinking Layer. Indeed, given a point cloud input, it is clustered into K regions, every node is updated via the K-Means Convolution operation, and then a condensed point cloud containing a single vector feature for each region is outputted. This description does not take into account the effects of the Self-Correlation Layer and AFA module, as they are irrelevant to explaining the overall aim of the architecture.

B.5 Horizontal stacking and non-local point correlations

The Shrinking layer takes as input a point cloud and returns a condensed point cloud as output. Hence, multiple layers can be stacked horizontally to capture higher-level representations incrementally like in CNNs and explore non-local point correlations like in [51]. Figure 7 provides an example of a horizontal stack containing three Shrinking Layers.

The suggested approach for the use of the Shrinking Layer in classification tasks is to stack as many layers as necessary and make the last layer output a condensed point cloud containing a single node in a very high dimension. This vector summarises the whole point cloud input and consequently can be fed into a fully-connected MLP for label assignment.

B.6 Proof of permutation invariance

Here, a proof of permutation invariance for the Shrinking Layer is provided. The proof is divided into four sub-proofs so to demonstrate that every internal layer of the Shrinking Layer is permutation equivariant.

B.6.1 Self-Correlation Layer permutation invariance proof

Let $P \in R^{N \times C}$ be a point cloud and $\hat{P} \in R^{N \times C}$ be a permutation of P . Let $ind : R \rightarrow R$ be a function that takes an index for a point in P and outputs the index of the corresponding point in \hat{P} .

The channel weight for a node $p_i \in P$ is identical to the channel weight for the corresponding node $\hat{p}_{ind(i)} \in \hat{P}$:

$$w_i = f(p_i) = f(\hat{p}_{ind(i)}) = \hat{w}_{ind(i)}$$

Consequently, normalising the two weight vectors leads to the same result:

$$\bar{w}_i = \bar{\hat{w}}_{ind(i)}$$

The updates of node $p_i \in P$ and $\hat{p}_{ind(i)} \in \hat{P}$ are equivalent:

$$p_i^{(0)} = p_i \oplus \lambda(\bar{w}_i \odot p_i) = \hat{p}_{ind(i)} \oplus \lambda(\bar{\hat{w}}_{ind(i)} \odot \hat{p}_{ind(i)}) = \hat{p}_{ind(i)}^{(0)}$$

As a consequence, $\hat{P}^{(0)}$ is a permutation of $P^{(0)}$. Since point clouds are unordered, then $P^{(0)}$ and $\hat{P}^{(0)}$ describe the same point cloud.

B.6.2 K-Means-Conv Layer permutation invariance proof

Let $P^{(0)} \in R^{N \times C}$ be a point cloud and $\hat{P}^{(0)} \in R^{N \times C}$ be a permutation of $P^{(0)}$. Let $ind : R \rightarrow R$ be a function that takes an index for a point in $P^{(0)}$ and outputs the index of the corresponding point in $\hat{P}^{(0)}$.

A directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed from $P^{(0)}$ such that \mathcal{V} denotes the set of nodes in $P^{(0)}$ and \mathcal{E} is defined as

$$\mathcal{E} = \{ (p_i^{(0)}, p_j^{(0)}) \mid p_i^{(0)}, p_j^{(0)} \in P^{(0)} \text{ and } p_i^{(0)}, p_j^{(0)} \text{ share the same centroid} \}$$

Also, a directed graph $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ is constructed from $\hat{P}^{(0)}$ such that $\hat{\mathcal{V}}$ denotes the set of nodes in $\hat{P}^{(0)}$ and $\hat{\mathcal{E}}$ is defined as

$$\hat{\mathcal{E}} = \{ (\hat{p}_i^{(0)}, \hat{p}_j^{(0)}) \mid \hat{p}_i^{(0)}, \hat{p}_j^{(0)} \in \hat{P}^{(0)} \text{ and } \hat{p}_i^{(0)}, \hat{p}_j^{(0)} \text{ share the same centroid} \}$$

\mathcal{G} and $\hat{\mathcal{G}}$ are identical graphs because $\mathcal{V} = \hat{\mathcal{V}}$ and if and only if $(p_i^{(0)}, p_j^{(0)}) \in \mathcal{E}$ then $(\hat{p}_{ind(i)}^{(0)}, \hat{p}_{ind(j)}^{(0)}) \in \hat{\mathcal{E}}$.

The first claim holds true because $P^{(0)}$ and $\hat{P}^{(0)}$ are permutations of each other. As such, they contain the same points. The second claim is a true statement as the computation of the centroids in $P^{(0)}$ and $\hat{P}^{(0)}$ is based on the Euclidean distance. Therefore, the set of centroids in $P^{(0)}$ is equivalent to the set of centroids in $\hat{P}^{(0)}$, even though the order of the points in $P^{(0)}$ and $\hat{P}^{(0)}$ differ.

The updates of node $p_i^{(0)} \in P^{(0)}$ and its corresponding point $\hat{p}_{ind(i)}^{(0)} \in \hat{P}^{(0)}$ are equivalent:

$$\begin{aligned} p_{i \text{ } up}^{(0)} &= \sigma \left(\sum_{j \in \mathcal{N}_i} \frac{\mathcal{F}(p_j^{(0)} - p_i^{(0)}) p_j^{(0)}}{\mathcal{M}(p_i^{(0)})} + \mathcal{W}(p_i^{(0)}) p_i^{(0)} + b \right) \\ &= \sigma \left(\sum_{j \in \hat{\mathcal{N}}_{ind(i)}} \frac{\mathcal{F}(\hat{p}_j^{(0)} - \hat{p}_{ind(i)}^{(0)}) \hat{p}_j^{(0)}}{\mathcal{M}(\hat{p}_{ind(i)}^{(0)})} + \mathcal{W}(\hat{p}_{ind(i)}^{(0)}) \hat{p}_{ind(i)}^{(0)} + b \right) \\ &= \hat{p}_{ind(i) \text{ } up}^{(0)} \end{aligned}$$

Given that $\mathcal{G} = \hat{\mathcal{G}}$ then $\mathcal{N}_i = \hat{\mathcal{N}}_{ind(i)}$. Hence, the above equivalence holds.

Consequently, $\hat{P}_{up}^{(0)}$ is a permutation of $P_{up}^{(0)}$. Since point clouds are unordered, then $P_{up}^{(0)}$ and $\hat{P}_{up}^{(0)}$ are descriptions of the same point cloud.

B.6.3 AFA Module permutation invariance proof

Let $P^{(0)} \in R^{N \times C}$ and $P_{up}^{(0)} \in R^{N \times T}$ be two feature point clouds. Let $\hat{P}^{(0)}$ and $\hat{P}_{up}^{(0)}$ be permutations of $P^{(0)}$ and $P_{up}^{(0)}$, respectively. Assume there exists a function $ind : R \rightarrow R$ that takes an index for a point in $P^{(0)}$ and outputs the index of the corresponding point in $\hat{P}^{(0)}$. A similar function ind_{up} also exists for the point clouds $P_{up}^{(0)}$ and $\hat{P}_{up}^{(0)}$.

The average nodes s_1 and \hat{s}_1 of $P^{(0)}$ and $\hat{P}^{(0)}$, respectively, are identical. The average nodes s_2 and \hat{s}_2 of $P_{up}^{(0)}$ and $\hat{P}_{up}^{(0)}$ follow suit:

$$\begin{aligned} s_1 &= \frac{1}{N} \sum_{i=1}^N p_i^{(0)} = \frac{1}{N} \sum_{i=1}^N \hat{p}_{ind(i)}^{(0)} = \hat{s}_1 \\ s_2 &= \frac{1}{N} \sum_{i=1}^N p_{i \text{ } up}^{(0)} = \frac{1}{N} \sum_{i=1}^N \hat{p}_{ind_{up}(i) \text{ } up}^{(0)} = \hat{s}_2 \end{aligned}$$

The descriptors of both point cloud permutation pairs are equivalent as well:

$$\begin{aligned} z_1 &= f_1(s_1) = f_1(\hat{s}_1) = \hat{z}_1 \\ z_2 &= f_2(s_2) = f_2(\hat{s}_2) = \hat{z}_2 \end{aligned}$$

As a consequence, the application of the gate mechanism leads to the same masked channels:

$$\begin{aligned} m_{1,c} &= \frac{e^{z_{1,c}}}{e^{z_{1,c}} + e^{z_{2,c}}} = \frac{e^{\hat{z}_{1,c}}}{e^{\hat{z}_{1,c}} + e^{\hat{z}_{2,c}}} = \hat{m}_{1,c} \\ m_{2,c} &= \frac{e^{z_{2,c}}}{e^{z_{1,c}} + e^{z_{2,c}}} = \frac{e^{\hat{z}_{2,c}}}{e^{\hat{z}_{1,c}} + e^{\hat{z}_{2,c}}} = \hat{m}_{2,c} \end{aligned}$$

Finally, the aggregated point clouds $P^{(1)}$ and $\hat{P}^{(1)}$ are permutations of each other as $p_i^{(1)} = \hat{p}_{ind(i)}^{(1)}$:

$$p_i^{(1)} = m_1 \cdot p_i^{(0)} + m_2 \cdot p_{i \text{ up}}^{(0)} = \hat{m}_1 \cdot \hat{p}_{ind(i)}^{(0)} + \hat{m}_2 \cdot \hat{p}_{ind_{up}(i)}^{(0)} = \hat{p}_{ind(i)}^{(1)}$$

Point clouds are unordered sets. Therefore, $P^{(1)}$ and $\hat{P}^{(1)}$ are equivalent.

B.6.4 Max-Pool Layer permutation invariance proof

Let $P^{(1)} \in R^{N \times T}$ be a feature point cloud and $\hat{P}^{(1)} \in R^{N \times T}$ be a permutation of $P^{(1)}$. By definition, the directed graph \mathcal{G}_2 computed from $P^{(1)}$ is equivalent to the directed graph $\hat{\mathcal{G}}_2$ constructed from $\hat{P}^{(1)}$. This holds because $P^{(1)}$ and $\hat{P}^{(1)}$ contain the same nodes, so they share the same ancestor nodes.

Let $ind : R \rightarrow R$ be a function that takes an index for a sub-graph in \mathcal{G}_2 and outputs the index of the corresponding sub-graph in $\hat{\mathcal{G}}_2$. The condensed feature point clouds $P^{(2)}$ and $\hat{P}^{(2)}$ are permutations of each other as $p_i^{(2)} = \hat{p}_{ind(i)}^{(2)}$:

$$p_i^{(2)} = \max(\mathcal{G}_2^{(i)}) = \max(\hat{\mathcal{G}}_2^{(ind(i))}) = \hat{p}_{ind(i)}^{(2)}$$

As point clouds are unordered sets, then $P^{(2)}$ and $\hat{P}^{(2)}$ are equivalent point cloud descriptions.

C Stack Shrinking Layer

C.1 Overview

Despite the overwhelming success of CNNs in image recognition tasks being mainly attributed to the generation of multiple feature maps for a given input, the entire point cloud feature extraction literature, including the Shrinking Layer, fails to embody the same principle in the point cloud domain. In fact, the current array of proposals seems to neglect the importance of detecting multiple point cloud features at a given depth of the network.

As point cloud recognition tasks are more complicated than their image counterparts, it seems critical to incorporate multiple feature extraction processes in future solutions to shorten the gap between the point cloud and image recognition literature. In light of this, this project proposes the first point cloud multi-feature extractor: the *Stack Shrinking Layer*. Moreover, the latter extracts the feature point clouds leveraging self, local and non-local point correlations in a semantic-based fashion.

C.2 Contributions

The main contributions of the Stack Shrinking Layer are:

- A point cloud multi-feature extractor that leverages self, local and non-local point correlations in a semantic-based fashion.
- The first point cloud feature extraction architecture fully resembling current CNN architectures.
- A permutation invariant point cloud feature extractor.
- A fully differentiable neural network module that can be plugged into existing deep neural network models to improve their performance.

C.3 Architecture

The idea behind the architecture of the Stack Shrinking Layer is relatively straightforward. Indeed, as the name suggests, the Stack Shrinking Layer is simply a stack of Shrinking Layers (Appendix B). As the Shrinking Layer enables the extraction of a single feature point cloud leveraging self, local and non-local point correlations in a semantic-based fashion, a stack of N Shrinking Layers produces N different feature point clouds in the same manner. Figure 8 provides an example of a Stack Shrinking Layer containing three Shrinking Layers.

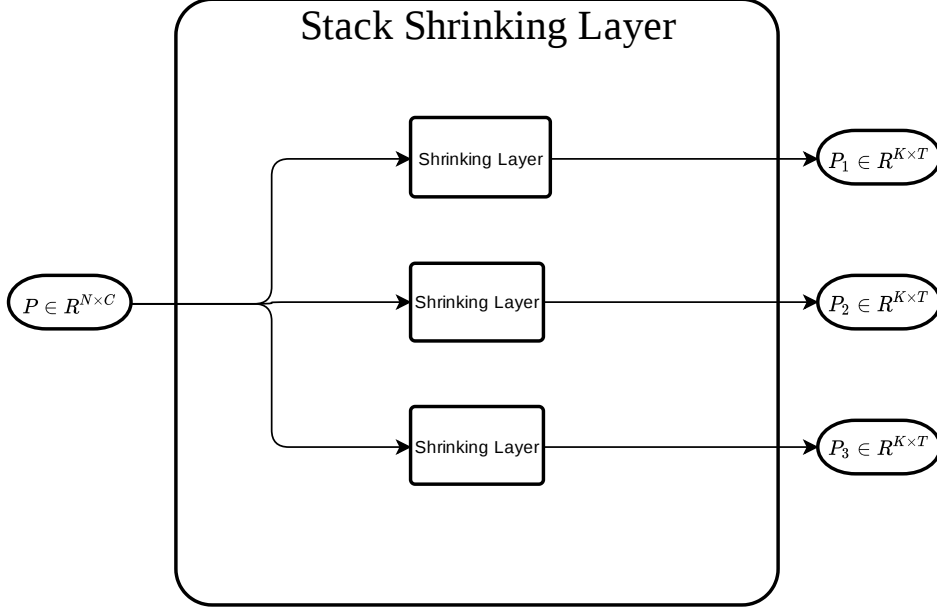


Figure 8: Architecture of a Stack Shrinking Layer containing a stack of three Shrinking Layers.

A critical parameter of the Stack Shrinking Layer is the number of feature point clouds to generate, denoted as n_F . Given N point cloud inputs, a n_F -Stack Shrinking Layer creates a stack of $n_F \cdot N$ Shrinking Layers so to generate n_F feature point clouds for each input. Hence, it outputs $n_F \cdot N$ feature point clouds.

The capability of the Stack Shrinking Layer to receive multiple point cloud inputs makes it suitable for horizontal stacking. That is, multiple layers can be stacked horizontally to capture higher-level representations incrementally in a CNN fashion and explore non-local point correlations like in [51].

The suggested approach for the use of the Stack Shrinking Layer in classification tasks is to stack horizontally as many layers as necessary and make the Shrinking Layers in the last Stack Shrinking Layer output condensed point clouds containing only a single node in a very high dimension. These vectors summarise different feature branches of the point cloud input. However, they cannot be fed into a fully-connected MLP for label assignment unless they are mapped to a single vector. The transformation of these feature vectors into a single vector can be performed via a permutation-invariant operation, such as max-pooling.

C.4 Implementations

The Stack Shrinking Layer has been employed in the ShrinkingNet neural network for point cloud classification tasks. The architecture of ShrinkingNet is illustrated in Figure 9. The latter consists of a (2-3-2) horizontal stack of Stack

Shrinking Layers, where (2-3-2) denotes the n_F values of the corresponding layers. The output of the horizontal stack is first transformed into a single vector by means of the max-pooling operation and then fed into a fully-connected MLP for label assignment.

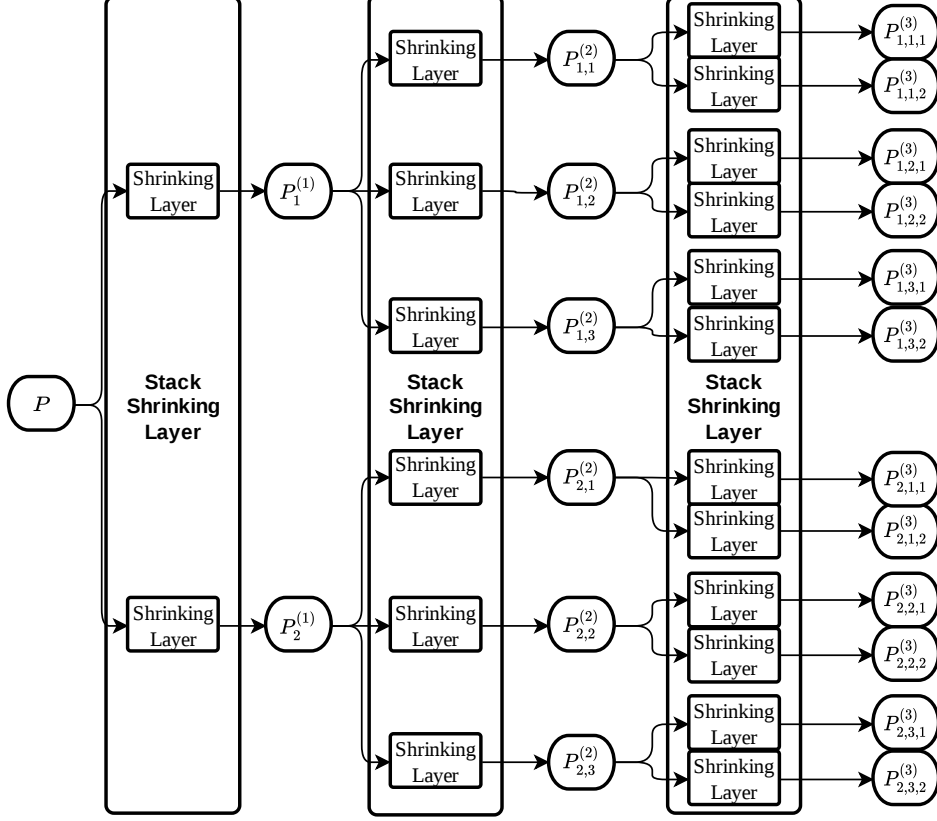


Figure 9: ShrinkingNet architecture

This architecture currently achieves 91.25% accuracy on the ModelNet10 dataset. However, further training is necessary to exploit its full potential.

C.5 Proof of permutation invariance

The Stack Shrinking Layer is a stack of Shrinking Layers. Appendix B shows that the Shrinking Layer is a permutation invariant point cloud feature extractor. Consequently, the Stack Shrinking Layer is permutation invariant as well.

D Extending Layer

D.1 Overview

The most advanced point cloud generative architectures are proposed in [21] and [22]. They adopt a multilayer generator network that alternates graph convolution and upsampling operations to incrementally generate hierarchical graph embeddings that represent better and better approximations of the output geometry. However, while the graph convolution operation in [21] is defined over the neighbourhood of a node, the counterpart operation in [22] convolves each point with respect to its tree of ancestors. Hence, the generation process in [21] is based on learning local structures; in contrast, [22] fuses all information from previous layers and different feature spaces to update each node.

It can be argued that exploring self, local and non-local correlations in a point cloud generation task is critical for accurately representing real-life objects. Indeed, as the point cloud feature extraction process relies on those correlations to achieve optimal results, point cloud generation, which merely performs the inverse operation, has the same necessity. However, neither [21] nor [22] are capable of leveraging the fundamental correlations in a point cloud. Specifically, [21] can only leverage local and non-local correlations, while [22] is reminiscent of PointNet [8] as it is a self-correlation approach augmented with ancestors' information.

This project proposes a novel architecture, dubbed Extending Layer, capable of exploring self, local and global correlations between points for the generation of point clouds. This is achieved using the internal layers of the Shrinking Layer (Appendix B); as such, the whole generation process is also semantic-based.

Furthermore, the Extending Layer augments the structure of the Shrinking Layer with an upsampling layer, which is necessary to increase the number of points in the graph embeddings incrementally. [21] doubles the number of points after each layer by applying an upsampling operation to each point exploiting local structures. On the other hand, TreeGAN multiplies each point by a matrix to produce an arbitrary number \mathbf{d} of child points from each point. The Extending Layer's approach for the upsampling process is reminiscent of both [21] and [22], though it ultimately differs from each one as it is semantic-based.

D.2 Contributions

The main contributions of the Extending Layer are:

- A novel architecture capable of exploring self, local and global correlations between points in a semantic-based fashion for the generation of

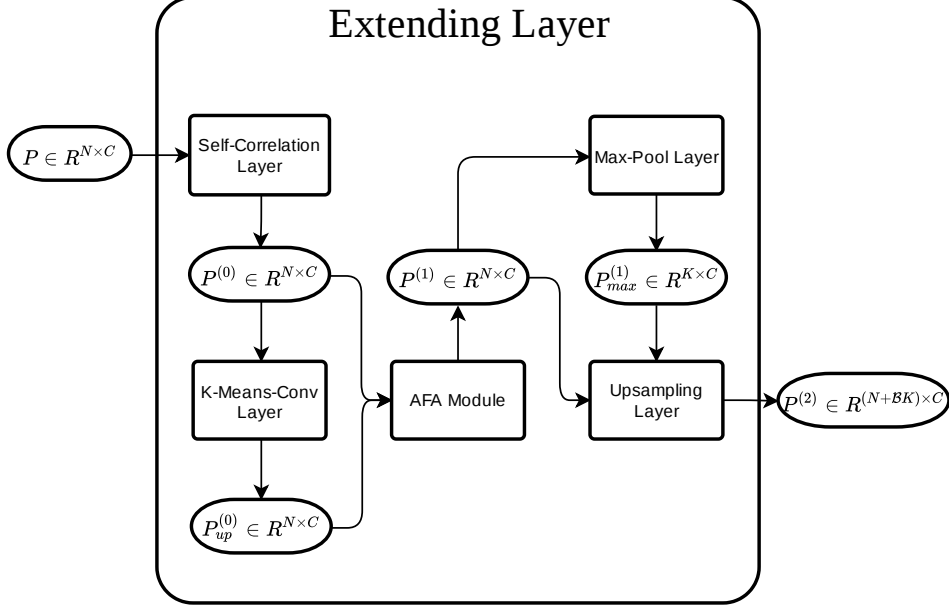


Figure 10: Extending Layer architecture

point clouds.

- A novel semantic-based upsampling operation.
- A permutation invariant point cloud generator.
- A fully differentiable neural network module that can be plugged into existing deep neural network models to improve their performance.

D.3 Architecture

The Extending Layer architecture is illustrated in Figure 10. Even though the Self-Correlation Layer, K-Means-Conv Layer, AFA Module and Max-Pool Layer are identical to their homonyms in the Shrinking Layer, here, they are explained again so to make this appendix self-contained. This is also necessary because some of their functionalities are used slightly differently.

Let $P \in R^{N \times C}$ be a point cloud containing N C -dimensional points. The Self-Correlation Layer takes P as input, learns the self-correlation of each point and updates each node accordingly, generating a point cloud output $P^{(0)} \in R^{N \times C}$.

The K-Means-Conv Layer captures local dependencies in $P^{(0)}$ by using the K-Means Convolution. That is, it clusters the nodes into K regions and performs a graph convolution operation on each node such that the neighbourhood of a node x is the set of nodes that belong to the same region as x . The output of the K-Means-Conv Layer is a point cloud embedding $P_{up}^{(0)} \in R^{N \times C}$.

Successively, the Adaptive Feature Aggregation Module constructs a point

cloud embedding $P^{(1)} \in R^{N \times C}$ by aggregating the features in $P^{(0)}$ and $P_{up}^{(0)}$ in a data-adaptive fashion through the use of a gate mechanism.

The Max-Pool Layer receives $P^{(1)}$ and outputs a condensed point cloud $P_{max}^{(1)} \in R^{K \times C}$, where K is the number of regions clustered in the K-Means-Conv Layer. Fundamentally, the Max-Pool layer encodes every region in $P^{(1)}$ with a unique node by applying the max-pooling operation to the set of nodes belonging to that region. A region in $P^{(1)}$ is defined as the ancestor region of a region clustered in $P^{(0)}$ during the K-Means Convolution operation. In other words, every node in a region of $P^{(1)}$ is the ancestor of a node in the corresponding region in $P^{(0)}$.

Finally, the Upsampling Layer constructs $P^{(2)} \in R^{(N+BK) \times C}$ by generating \mathcal{B} new nodes for each cluster encoding in $P_{max}^{(1)}$ and concatenating them with $P^{(1)}$.

The learning mechanism for each of these internal layers is exhaustively described in what follows.

D.3.1 Self-Correlation Layer

In order to explore self-correlation, edges between the different channels of a node should be constructed and a weight for each edge learnt. After, a node's channels should be updated according to the learnt weights.

However, the Self-Correlation Layer adopts the technique proposed in [9] to encode all the weights from different channels into one so to avoid time-complexity issues. Consequently, the channel weight $w_i \in R^C$ for a given node $p_i \in P$ is computed as follows:

$$w_i = f(p_i)$$

where f is a MLP.

Unlike in Point2Node, f does not need to be a MLP with 1D separable convolutions. Hence, the reader can opt for any MLP architecture according to the problem at hand.

The weight vector is successively normalised so to make all the scalar weights for each channel comparable:

$$\overline{w_{i,c}} = \frac{e^{w_{i,c}}}{\sum_{l=1}^C e^{w_{i,l}}}$$

where $w_{i,c}$ is the c^{th} scalar value of w_i .

After the correlation learning, the following equation is used to update a node's channels:

$$p_i^{(0)} = p_i \oplus \lambda (\overline{w_i} \odot p_i)$$

where λ is a parameter to be learnt.

The Self-Correlation Layer transforms P into $P^{(0)}$.

D.3.2 K-Means-Conv Layer

The K-Means-Conv Layer explores local geometric structures by constructing a local neighbourhood graph via K-Means++[61] and applying a modified version of the graph convolution operation in [21]. This convolution operation, dubbed K-Means Convolution, is in the spirit of graph neural networks.

Given a parameter K for the number of regions to be clustered, a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, representing local structures, is computed from $P^{(0)}$ with the help of the K-Means++ algorithm. \mathcal{V} denotes the set of nodes in $P^{(0)}$, while the set of directed edges \mathcal{E} is defined as

$$\mathcal{E} = \{ (p_i^{(0)}, p_j^{(0)}) \mid p_i^{(0)}, p_j^{(0)} \in P^{(0)} \text{ and } p_i^{(0)}, p_j^{(0)} \text{ share the same centroid} \}$$

That is, \mathcal{G} is a collection of K disjoint directed sub-graphs such that every two nodes in each sub-graph share the same centroid computed via the K-Means++ algorithm. Additionally, each sub-graph is a complete graph containing self-loops. K-Means++ is used, instead of the usual K-Means, because it achieves better clustering performance and reduces the variance of the clustered regions between successive runs in the same point cloud.

Subsequently, each node $p_i^{(0)}$ in \mathcal{G} is updated with the following graph convolution operation:

$$p_{i \text{ up}}^{(0)} = \sigma \left(\sum_{j \in \mathcal{N}_i} \frac{\mathcal{F}(p_j^{(0)} - p_i^{(0)}) p_j^{(0)}}{\mathcal{M}(p_i^{(0)})} + \mathcal{W}(p_i^{(0)}) p_i^{(0)} + b \right)$$

where \mathcal{N}_i is the neighborhood of node $p_i^{(0)}$, \mathcal{F} is a $R^C \rightarrow R^{C \times C}$ MLP, \mathcal{W} is a $R^C \rightarrow R^{C \times C}$ MLP, \mathcal{M} is a $R^C \rightarrow R$ MLP, b is a R^C vector and σ is a non-linear activation function.

The reader is free to select any architecture for the above MLPs according to the problem at hand.

The K-Means-Conv Layer outputs $P_{up}^{(0)}$.

D.3.3 Adaptive Feature Aggregation Module

The Adaptive Feature Aggregation(AFA) Module aggregates the features in $P^{(0)}$ and $P_{up}^{(0)}$ in a data-adaptive fashion through the use of a gate mechanism.

D.3.3.1 Parameter free Characteristic Modelling

Given $P^{(0)}$ and $P_{up}^{(0)}$, their characteristics are described by simply computing the average node s_1 and s_2 , respectively:

$$s_1 = \frac{1}{N} \sum_{i=1}^N p_i^{(0)}$$

$$s_2 = \frac{1}{N} \sum_{i=1}^N p_{i\ up}^{(0)}$$

where $p_i^{(0)}$ is the i^{th} node in $P^{(0)}$ and $p_{i\ up}^{(0)}$ is the i^{th} node in $P_{up}^{(0)}$

D.3.3.2 Parameterized Characteristic Modeling

Two MLPs map the average nodes s_1 and s_2 into new descriptors $z_1 \in R^{1 \times C}$ and $z_2 \in R^{1 \times C}$, respectively:

$$z_1 = f_1(s_1)$$

$$z_2 = f_2(s_2)$$

where f_1 and f_2 are the two MLPs.

Unlike in [9], f_1 and f_2 are not constrained to be MLPs consisting of 1D reduction convolution and 1D increasing convolution layers. Thus, the reader can opt for any architecture at his will.

D.3.3.3 Gate Mechanism

A gate mechanism applies the softmax function to the descriptors z_1 and z_2 :

$$m_{1,c} = \frac{e^{z_{1,c}}}{e^{z_{1,c}} + e^{z_{2,c}}}$$

$$m_{2,c} = \frac{e^{z_{2,c}}}{e^{z_{1,c}} + e^{z_{2,c}}}$$

where $m_{1,c}$ and $m_{2,c}$ are the masked z_1 and z_2 on the c^{th} channel.

D.3.3.4 Aggregation

With the help of m_1 and m_2 , $P^{(0)}$ and $P_{up}^{(0)}$ are aggregated forming $P^{(1)}$:

$$p_i^{(1)} = m_1 \cdot p_i^{(0)} + m_2 \cdot p_{i\ up}^{(0)}$$

where $p_i^{(1)}$ is the i^{th} node in $P^{(1)}$.

D.3.4 Max-Pool Layer

The Max-Pool Layer applies the max-pooling operation to encode every region in $P^{(1)}$ with a single node. First, a directed graph $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ is computed from $P^{(1)}$ such that \mathcal{V}_2 is the set of nodes in $P^{(1)}$ and \mathcal{E}_2 is defined as

$$\mathcal{E}_2 = \{ (p_i^{(1)}, p_j^{(1)}) \mid p_i^{(1)}, p_j^{(1)} \in P^{(1)} \text{ and } (anc(p_i^{(0)}), anc(p_j^{(0)})) \in \mathcal{E} \}$$

In other words, \mathcal{G}_2 is a collection of K disjoint directed sub-graphs such that for every pair of nodes in each sub-graph, their ancestor nodes are adjacent nodes in \mathcal{G} . Remember that \mathcal{G} is the directed graph produced during the K-Means Convolution operation.

Subsequently, for every sub-graph $\mathcal{G}_2^{(i)}$ in \mathcal{G}_2 , a single node $p_{i \max}^{(1)}$, encoding the whole region, is produced by means of the max-pooling operation:

$$p_{i \max}^{(1)} = \max \left(\mathcal{G}_2^{(i)} \right)$$

where \max computes the max-pooling operation along each dimension of the nodes in $\mathcal{G}_2^{(i)}$.

Notice that $p_{i \max}^{(1)}$ will be the i^{th} point in $P_{\max}^{(1)}$. The Max-Pool Layer returns the feature point cloud $P_{\max}^{(1)}$.

D.3.5 Upsampling Layer

The Upsampling Layer generates \mathcal{B} new nodes for each cluster encoding in $P_{\max}^{(1)}$ and concatenates them with $P^{(1)}$ producing $P^{(2)} \in R^{(N+\mathcal{B}K) \times C}$.

In order to achieve this, K intermediate point clouds $P_{inter}^{(i)} \in R^{\mathcal{B} \times C}$ are generated from each node $p_{i \max}^{(1)} \in P_{\max}^{(1)}$:

$$P_{inter}^{(i)} = \mathcal{U}(p_{i \max}^{(1)})$$

where \mathcal{U} is a $R^C \rightarrow R^{K \times C}$ MLP that takes a cluster encoding in $P_{\max}^{(1)}$ and generates \mathcal{B} new nodes from it.

Afterwards, the Upsampling Layer outputs $P^{(2)} \in R^{(N+\mathcal{B}K) \times C}$ such that

$$P^{(2)} = P^{(1)} \text{ concat } P_{inter}^{(1)} \text{ concat } \dots \text{ concat } P_{inter}^{(K)}$$

where *concat* denotes the concatenation operation

D.4 Rationale behind the architecture

The approaches proposed in [21] and [22] for point cloud generation can only leverage local and non-local correlation, and self-correlation, respectively. However, it is critical to explore the entire array of fundamental correlations for the lifelike representation of point clouds. Thus, point cloud generative architectures should exploit self, local and non-local correlations.

For this reason, the Extending Layer employs the Self-Correlation and K-Means-Conv Layers, as the former explores self-correlations while the latter captures local dependencies. By stacking multiple Extending Layers horizontally, local dependencies between points, whose ancestors may belong to different spatial locations, are explored; therefore, non-local correlation learning is achieved too. Figure 11 illustrates a horizontal stack containing three

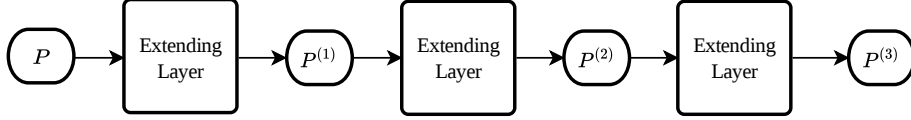


Figure 11: Architecture of a horizontal stack containing three Extending Layers.

Extending Layers.

The use of the AFA module allows the aggregation of features from the self-correlation and local-correlation learnings in a self-adaptive fashion.

It is important to notice that although both the K-Means-Conv Layer and the graph convolution operation in [21] learn local dependencies, they do it differently. In fact, [21] updates each node with respect to its K -nearest neighbours; in contrast, the K-Means-Conv Layer convolves each point with respect to the points located in the same semantic region. Consequently, the update of a node does not get spurious information from nodes not having the same semantic nature.

While point cloud generative architectures may take as input a point cloud embedding and output a final point cloud with the same number of nodes, it would not be sensible to do so. Indeed, a step-by-step process that successively updates the nodes of the previous layer’s graph embedding and also generates new nodes for it works better in practice. The reason for it stems from the fact that the latter approach is reminiscent of real-life products production. As real-life products go through several manufacturing steps that refine the previous shapes and add new details, point cloud generation should also go through the same stages. Refining in point clouds means updating the already-existing points while adding new details consists of generating new nodes.

For the reasons outlined above, [21] and [22] use an upsampling operation after each graph convolution operation. [21] doubles the number of points after each layer by creating a descendant node for every already-existing node exploiting

local structures. On the other hand, TreeGAN multiplies each point by a matrix to produce an arbitrary number \mathbf{d} of child points from each point. None of these solutions is optimal, however. Indeed, [21] learns local dependencies but can only double the number of nodes; on the other hand, [22] can create an arbitrary number of nodes from each point, but it does not exploit local structures.

The architecture of the Extending Layer, instead, allows creating an arbitrary number of points from locally defined structures. To be specific, the generation of new nodes is semantic-based. In other words, new nodes are not generated from each point in the point cloud but from each semantic region. The Max-Pool Layer summarises every semantic region with a node, and the Upsampling Layer generates an arbitrary number of new nodes from each encoding.

D.5 Practical considerations

Most generative architectures construct a point cloud from a given random latent vector. However, the Extending Layer can only take point clouds as input. Hence, a random latent vector needs to be transformed into an initial point cloud containing a few points.

When stacking multiple Extending Layers horizontally, the reader should be aware of the fact that the parameter K , which is used for finding the semantic regions of a point cloud, should increase incrementally. This needs to be done because as new nodes are generated, the number of semantic regions in the point cloud embeddings increases as well.

D.6 Proof of permutation invariance

A proof of permutation invariance for the Extending Layer is provided here. Given that the Self-Correlation Layer, K-Means-Conv Layer, AFA Module and Max-Pool Layer are proved permutation invariant in Appendix B, it is only necessary to demonstrate the invariance properties of the Upsampling Layer.

D.6.1 Upsampling Layer permutation invariance proof

Let $P_{max}^{(1)}$ and $P^{(1)}$ be two point cloud embeddings. Let $\hat{P}_{max}^{(1)}$ and $\hat{P}^{(1)}$ be permutations of $P_{max}^{(1)}$ and $P^{(1)}$, respectively. Suppose there exists a function $ind : R \rightarrow R$ that takes an index for a point in $P_{max}^{(1)}$ and outputs the index of the corresponding point in $\hat{P}_{max}^{(1)}$.

Given a point $p_{i \ max}^{(1)} \in P_{max}^{(1)}$ and its corresponding point $\hat{p}_{ind(i) \ max}^{(1)} \in \hat{P}_{max}^{(1)}$, the intermediate point clouds generated from them are identical:

$$P_{inter}^{(i)} = \mathcal{U}(p_{i \ max}^{(1)}) = \mathcal{U}(\hat{p}_{ind(i) \ max}^{(1)}) = \hat{P}_{inter}^{(inter(i))}$$

Since,

$$\begin{aligned} P^{(2)} &= P^{(1)} \text{ concat } P_{inter}^{(1)} \text{ concat } \dots \text{ concat } P_{inter}^{(K)} \\ \hat{P}^{(2)} &= \hat{P}^{(1)} \text{ concat } \hat{P}_{inter}^{(1)} \text{ concat } \dots \text{ concat } \hat{P}_{inter}^{(K)} \end{aligned}$$

then $P^{(2)}$ and $\hat{P}^{(2)}$ are permutations of each other. As a result, they describe the same point cloud in view of the fact that point clouds are unordered.

E Stack Extending Layer

E.1 Overview

Current point cloud generative architectures, such as [21], [22] and the Extending Layer, adopt a multilayer approach that alternates graph convolution and upsampling operations to generate point cloud embeddings that represent increasingly better approximations of the output shape. However, these solutions are somewhat defective.

Specifically, they update already-existing nodes and generate new ones with shared weights. As a consequence, different regions of the point cloud embeddings are refined and augmented with new features in a non-specialised way. It can be argued that region or semantic-based specialised generation is, instead, necessary for producing lifelike shapes.

Indeed, by inspecting current manufacturing processes, it is evident that the production of objects is broken down into separate manufacturing steps which produce different components. These parts are then assembled at the end of the assembly line, producing the final shapes. Not only are these processes adopted to reduce labour costs but also to increase the specialisation of labour in specific tasks.

Similarly, point cloud generative processes should learn to construct the different semantic regions of objects in a specialistic manner. However, this cannot be achieved with the use of shared weights.

This project proposes the first point cloud generative architecture, named *Stack Extending Layer*, that generates point clouds in a specialistic manner, leveraging self, local and non-local correlations. This is achieved with the help of the Extending Layer.

E.2 Contributions

The main contributions of the Stack Extending Layer are:

- A novel point cloud generative approach, resembling current assembly lines, that generates point clouds in a specialistic manner, leveraging self, local and non-local correlations.

E.3 Architecture

The idea behind the architecture of the Stack Extending Layer is relatively straightforward. Indeed, as the name suggests, the Stack Extending Layer is

simply a stack of Extending Layers (Appendix D). The Extending Layer allows generating a point cloud embedding from a given embedding input exploring self, local and non-local correlations. As such, a stack of N Extending Layers produces N different feature point embeddings in the same manner. Figure 12

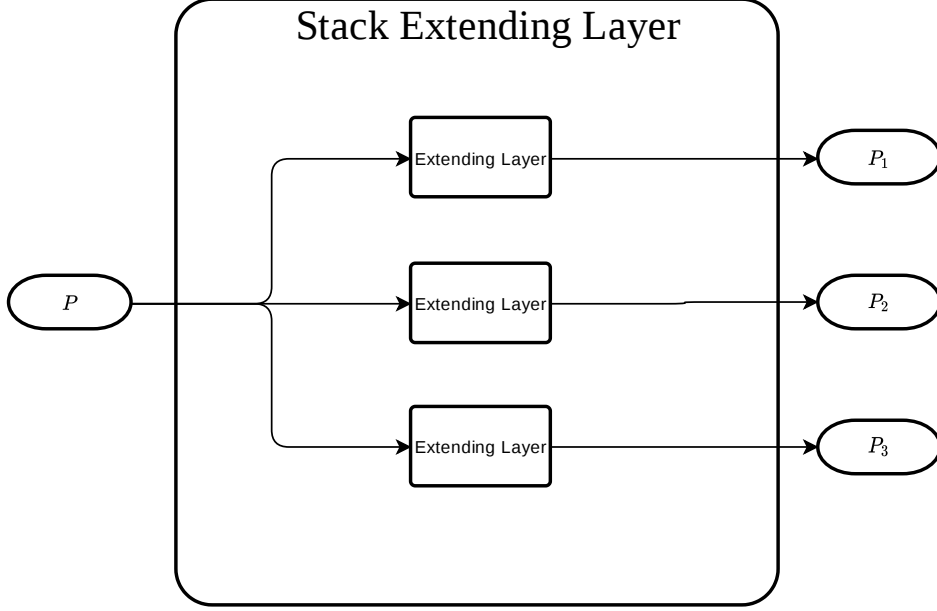


Figure 12: Architecture of a Stack Extending Layer containing a stack of three Extending Layers.

provides an example of a Stack Extending Layer containing three Extending Layers.

A critical parameter of the Stack Extending Layer is the number of point clouds embeddings to generate, denoted as n_E . Given N point cloud embeddings inputs, a n_E -Stack Extending Layer creates a stack of $n_E \cdot N$ Extending Layers so to generate n_E point clouds embeddings for each input. Hence, it outputs $n_E \cdot N$ embeddings.

The capability of the Stack Extending Layer to receive multiple point cloud embedding inputs makes it suitable for horizontal stacking. That is, multiple layers can be stacked horizontally to generate finer-grained embeddings. This approach creates many embedding branches that specialise in generating different semantic regions. The outputs of the final layer are then concatenated together.

Thus, the approach is like a branched assembly line where multiple components are produced and then assembled, producing the final shape. An example of a horizontal stacking of Stack Extending Layers is shown in Figure 13.

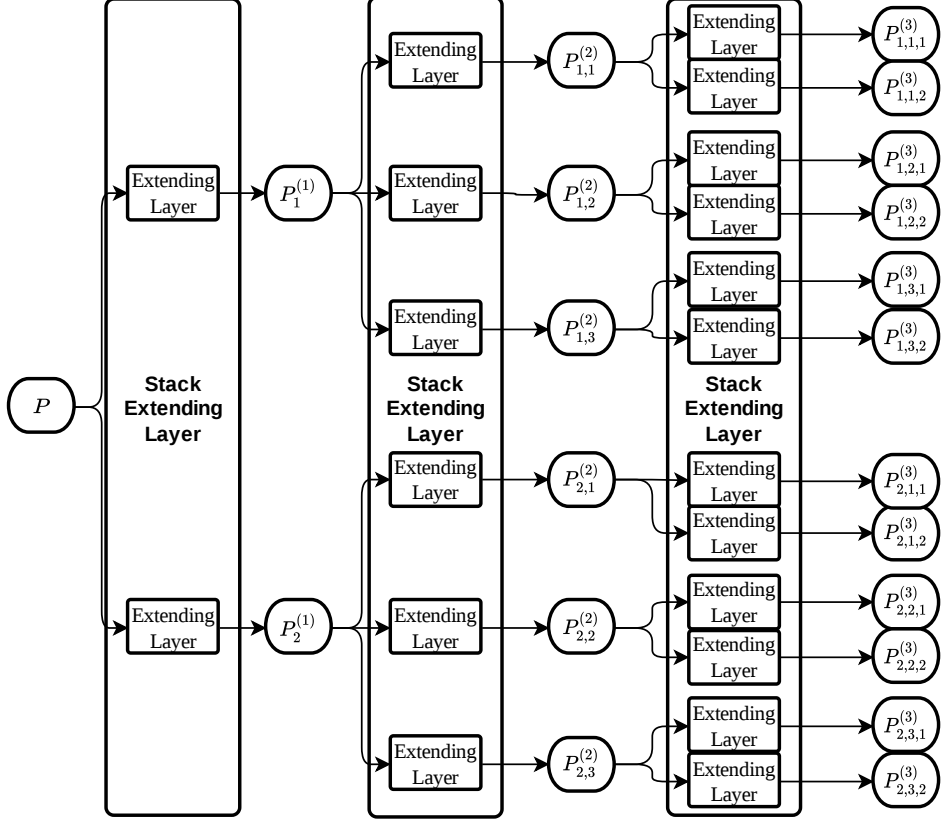


Figure 13: Architecture of a (2-3-2) horizontal stack of Stack Extending Layers, where (2-3-2) denotes the n_E values of the corresponding layers.

E.4 Proof of permutation invariance

The Stack Extending Layer is a stack of Extending Layers. Appendix D shows that the Extending Layer is a permutation invariant point cloud generator. Consequently, the Stack Extending Layer is permutation invariant as well.

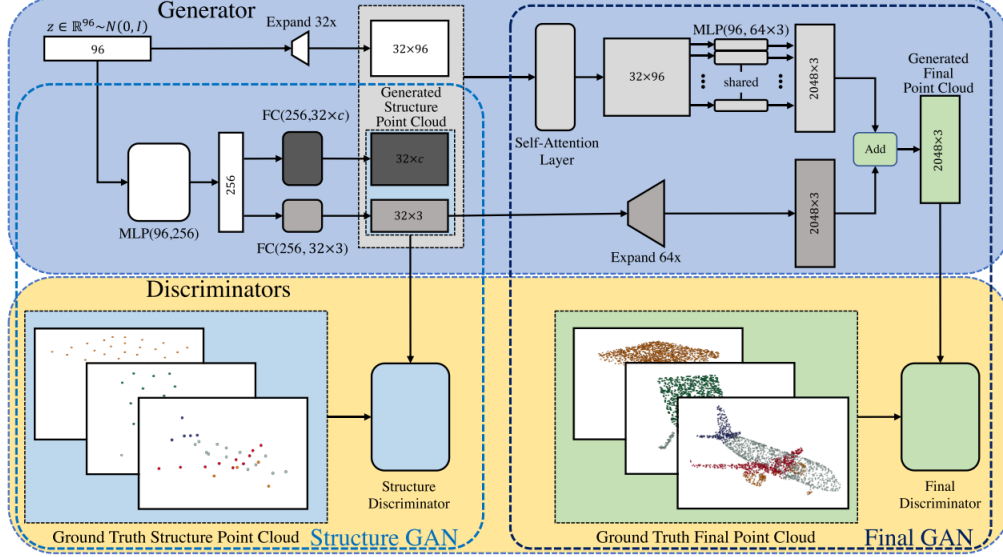


Figure 14: CPCGAN architecture [6]. The expand components simply copy the data several times and concatenate them.

F CPCGAN

F.1 Overview

CPCGAN is a Controllable Generative Adversarial Network capable of generating 3D point clouds and semantic labels for points from latent code. CPCGAN is based on a two-stage architecture, taking inspiration from [23].

The first stage generates a structure point cloud and the semantic labels for its points from a random latent vector. A structure point cloud is a sparse point cloud containing significant structural information.

With the guidance of the structure point cloud, the second stage creates a final point cloud and the semantic labels for its points. The generation process in the second stage involves the creation of a certain number of points for each point in the structure point cloud. The points generated inherit the semantic labels of their parents.

Controllable generation of point clouds is achieved by altering the structure of the middle-level point cloud.

F.2 Architecture

F.2.1 Overview

Figure 14 depicts the CPCGAN’s architecture. The network aims to generate a final point cloud $CP_{GE} = \{g_i\}_{i=1}^{2048}$ and its semantic labels $CS_{GE} = \{s_i\}_{i=1}^{2048}$

from a latent vector $z \in R^{96} \sim \mathcal{N}(0, I)$. Note that the i^{th} point in the generated final point cloud CP_{GE} is denoted as g_i , while s_i is the semantic label of the i^{th} point. In order to create CP_{GE} and CS_{GE} , a two-stage process is carried out.

The first stage is called Structure GAN and it is concerned with learning the distribution of real structure point clouds SP_{GT} and their corresponding semantic labels SS_{GT} . According to the learnt distribution, Structure GAN generates structure point clouds $SP_{GE} = \{gsp_i\}_{i=1}^{32}$ and semantic labels $SS_{GE} = \{gss_i\}_{i=1}^{32}$ from a latent code $z \in R^{96} \sim \mathcal{N}(0, I)$.

Finally, the second stage, named Final GAN, takes as input SP_{GE} , SS_{GE} and z to generate the final point cloud CP_{GE} and semantic labels CS_{GE} . The objective of Final GAN is to learn the distribution of the real final point clouds CP_{GT} and their semantic labels CS_{GT} .

F.2.2 Structure GAN

Structure GAN comprises a simple generator and a PointNet-based discriminator. The generator and discriminator are called Structure generator and Structure discriminator, respectively.

The Structure generator takes as input a latent vector $z \in R^{96} \sim \mathcal{N}(0, I)$ and uses a MLP to transform it into a vector $v \in R^{256}$. With the help of a single fully-connected layer, v is converted into a structure point cloud SP_{GE} . SS_{GE} is produced by feeding v into a fully connected layer followed by a softmax layer. $32 \times c$ probabilities for the semantic labels are generated, where c is the number of possible labels for each point.

Both SP_{GE} and SS_{GE} are given as input to the Structure discriminator. Even though the Structure discriminator is PointNet-based, it slightly differs from the architecture proposed in the original paper [8]. Indeed, CPCGAN’s authors remove the T-Net and replace the max-pooling layer with a global average-pooling layer.

F.2.3 Final GAN

Final GAN consists of a generator (Final generator) and a PointNet-based discriminator (Final discriminator).

The Final generator takes as input z , SP_{GE} and SS_{GE} so that to generate the final point cloud CP_{GE} and its semantic labels CS_{GE} . It constructs a matrix $Z \in R^{32 \times 96}$ by creating 32 copies of the latent vector z . Then, Z is fed into a Self-Attention layer, which generates a matrix $Z' \in R^{32 \times 96}$. A Self-Attention layer consists of three fully connected layers, whose purpose is to transform the i^{th} row z_i in Z into a latent vector z'_i such that z'_i encodes the points that can be bred from the i^{th} structure point gsp_i in SP_{GE} .

The authors of CPCGAN propose a PointNet-like architecture, called Delta, to breed points from each row in Z' . Delta comprises 32 MLPs with shared weights that breed 64 points for each latent code in Z' . The points bred form an intermediary final point cloud $CP'_{GE} = \{g'_i\}_{i=1}^{2048}$.

To control the final point cloud shape, every point g'_i in CP'_{GE} is summed to the point gsp'_i in SP'_{GE} , where $SP'_{GE} = \{gsp'_i\}_{i=1}^{2048}$ is obtained by replicating 64 times each point in the structure point cloud SP_{GE} .

Thus, $CP_{GE} = \{g'_i + gsp'_i \mid g'_i \in CP'_{GE} \text{ and } gsp'_i \in SP'_{GE}, i = 1, \dots, 2048\}$.

The final semantic labels CS_{GE} are simply the result of replicating 64 times each semantic label in SS_{GE} .

Both CP_{GE} and CS_{GE} are then fed into the Final discriminator.

F.3 Controllable generation

Structure GAN aims to learn the distribution of the real structure point clouds SP_{GT} and their corresponding semantic labels SS_{GT} so that to generate structure point clouds SP_{GE} and semantic labels SS_{GE} accordingly.

Final GAN generates final point clouds CP_{GE} and semantic labels CS_{GE} with the guidance of SP_{GE} , SS_{GE} and a random latent vector z . Thus, if SP_{GE} and SS_{GE} are replaced with some handcrafted structure point clouds SP_{GH} and semantic labels SS_{GH} that are drawn from the same distribution of SP_{GT} and SS_{GT} , it is possible to achieve controllable generation.

Still, when handcrafted structure point clouds SP_{GH} and semantic labels SS_{GH} are used, Final GAN draws a random latent vector $z \in R^{96} \sim \mathcal{N}(0, I)$ in order to generate various samples when SP_{GH} and SS_{GH} are unchanged.

F.4 Training

F.4.1 Real structure point clouds and semantic labels formation

In order to generate structure point clouds SP_{GE} and semantic labels SS_{GE} , Structure GAN needs to learn the distributions of real structure point clouds SP_{GT} and semantic labels SS_{GT} .

Therefore, it is necessary to convert a dataset containing complete point clouds and their semantic labels into a dataset of structure point clouds and corresponding semantic labels.

CPCGAN's authors use K-means to extract 32 points out of a complete point cloud. Specifically, they use K-means to find the centroids of the points with the same semantic labels. The value of K for each semantic label is established

according to the proportion of points with that label in the entire point cloud. The centroids computed for a given semantic label inherit the same label.

F.4.2 Process

To speed up the training process and improve performance, CPCGAN’s authors adopt a two-stage training approach. In the first stage, Structure GAN and Final GAN are trained separately. Hence, SP_{GT} and SS_{GT} are fed into Final GAN instead of SP_{GE} and SS_{GE} . In contrast, Structure GAN and Final GAN are trained simultaneously in the second step.

F.4.3 Loss functions

The loss functions used during the CPCGAN’s training process are those proposed in Wasserstein GAN [13] with the addition of gradient penalty[62].

G First and second stage Final WGAN training procedures

G.1 Overview

Section 8.3 describes the general Final WGAN training procedure without mentioning how the point cloud blueprints \hat{B} are computed. The reason for this stems from the fact that, as discussed in section 6.3, the computation of the point cloud blueprints changes according to which stage of the CC3DVAEWGAN training procedure is in place.

Here, the computation of the point cloud blueprints \hat{B} is outlined for both the first stage and second stage of the CC3DVAEWGAN training process.

G.2 First stage training procedure

Algorithm 5 provides a summary of the first stage Final WGAN training algorithm. It is identical to the training algorithm described in section 8.3 with the only exception that it specifies how the point cloud blueprints \hat{B} are generated.

In detail, the batch of point cloud blueprints \hat{B} is produced by extracting structural information from the batch of real point clouds X . Taking inspiration from [6], this is achieved by using the K-Means++ algorithm. The point clouds in X are partitioned into $n_{blueprints}$ regions, and each region's centroid is utilised to produce a batch $\hat{B} \in R^{m \times n_{blueprints} \times 3}$ of point cloud blueprints.

G.3 Second stage training procedure

Algorithm 6 provides a summary of the second stage Final WGAN training algorithm. It is identical to the training algorithm described in section 8.3, but it also specifies how the point cloud blueprints \hat{B} are generated.

Specifically, first, a batch of latent codes Z is produced such that the i^{th} latent vector in Z is drawn from the Gaussian Mixture Model of the same class as the i^{th} real point cloud in X . Remember that Blueprint VAE-WGAN learned the Gaussian Mixture Model for a given class during the first stage of the CC3DVAEWGAN training algorithm. Subsequently, the Blueprint Generator generates the batch of point cloud blueprints \hat{B} by taking as input the batch of latent codes Z .

Algorithm 5 Final WGAN training algorithm first stage

Require:

- 1: λ_{FG} : Final Generator(FG) learning rate.
 - 2: λ_{FD} : Final Discriminator(FD) learning rate.
 - 3: $n_{blueprint}$: number of points for the point cloud blueprints.
 - 4: m : batch size.
 - 5: c : clipping parameter.
 - 6: n_{FD} : number of iterations of the discriminator per generator iteration.
 - 7: γ : weighting style and content parameter
-
- 9: $\theta_{FG} \leftarrow$ initialize FG parameters
 - 10: $\theta_{FD} \leftarrow$ initialize FD parameters
 - 11: **while** θ_{FG} and θ_{FD} have not converged **do**
 - 12: **while** there are batches available **do**
 - 13: $X \leftarrow$ a batch $\{x^{(i)}\}_{i=1}^m$ of point clouds from the real dataset
 - 14: $H \leftarrow \{\text{onehot}(x^{(i)})\}_{i=1}^m \quad \triangleright$ one-hot encoding of the classes in X
 - 15: $\hat{B} \leftarrow \{\text{Kmeans}(\hat{x}^{(i)}, n_{blueprint})\}_{i=1}^m$
 - 16: $\hat{X} \leftarrow \text{FG}(\hat{B})$
 - 17: $\mathcal{L}_{FD} \leftarrow \mathbb{E}_{p(\hat{X})}[\text{FD}(\hat{X}, H)] - \mathbb{E}_{p(X)}[\text{FD}(X, H)]$
 - 18: // Update parameters of FD
 - 19: $\theta_{FD} \leftarrow \theta_{FD} - \lambda_{FD} \cdot \text{RMSPProp}(\theta_{FD}, \nabla_{\theta_{FD}} \mathcal{L}_{FD})$
 - 20: $\theta_{FD} \leftarrow \text{clip}(\theta_{FD}, -c, c)$
 - 21: // Update parameters of FG
 - 22: **if** FG can be updated according to n_{FD} **then**
 - 23: $\bar{B} \leftarrow \{\text{Kmeans}(\hat{x}^{(i)}, n_{blueprint})\}_{i=1}^m$
 - 24: $\mathcal{L}_{llike}^{points} \leftarrow -\mathbb{E}_{q(Z|\bar{B})}[\log p(\bar{B}|Z)]$
 - 25: $\mathcal{L}_{FG} \leftarrow -\mathbb{E}_{p(\hat{X})}[\text{BD}(\hat{X}, H)]$
 - 26: $\theta_{FG} \leftarrow \theta_{FG} - \lambda_{FG} \cdot \text{RMSPProp}(\theta_{FG}, \nabla_{\theta_{FG}} (\gamma \mathcal{L}_{llike}^{points} + \mathcal{L}_{FG}))$
 - 27: **end if**
 - 28: **end while**
 - 29: **end while**
-

Algorithm 6 Final WGAN training algorithm second stage

Require:

- 1: λ_{FG} : Final Generator(FG) learning rate.
 - 2: λ_{FD} : Final Discriminator(FD) learning rate.
 - 3: $n_{blueprint}$: number of points for the point cloud blueprints.
 - 4: m : batch size.
 - 5: c : clipping parameter.
 - 6: n_{FD} : number of iterations of the discriminator per generator iteration.
 - 7: γ : weighting style and content parameter
 - 8: BG : Blueprint Generator
 - 9: GM : Gaussian Mixture dictionary
-
- 10:
 - 11: $\theta_{FG} \leftarrow$ initialize FG parameters
 - 12: $\theta_{FD} \leftarrow$ initialize FD parameters
 - 13: **while** θ_{FG} and θ_{FD} have not converged **do**
 - 14: **while** there are batches available **do**
 - 15: $X \leftarrow$ a batch $\{x^{(i)}\}_{i=1}^m$ of point clouds from the real dataset
 - 16: $H \leftarrow \{\text{onehot}(x^{(i)})\}_{i=1}^m \triangleright$ one-hot encoding of the classes in X
 - 17: $Z \leftarrow \{\text{sample}(GM[c]) \mid c \text{ is the class of } x^{(i)}\}_{i=1}^m$
 - 18: $\hat{B} \leftarrow BG(Z)$
 - 19: $\hat{X} \leftarrow FG(\hat{B})$
 - 20: $\mathcal{L}_{FD} \leftarrow \mathbb{E}_{p(\hat{X})}[\text{FD}(\hat{X}, H)] - \mathbb{E}_{p(X)}[\text{FD}(X, H)]$
 - 21: // Update parameters of FD
 - 22: $\theta_{FD} \leftarrow \theta_{FD} - \lambda_{FD} \cdot \text{RMSProp}(\theta_{FD}, \nabla_{\theta_{FD}} \mathcal{L}_{FD})$
 - 23: $\theta_{FD} \leftarrow \text{clip}(\theta_{FD}, -c, c)$
 - 24: // Update parameters of FG
 - 25: **if** FG can be updated according to n_{FD} **then**
 - 26: $\bar{B} \leftarrow \{\text{Kmeans}(\hat{x}^{(i)}, n_{blueprint})\}_{i=1}^m$
 - 27: $\mathcal{L}_{like}^{points} \leftarrow -\mathbb{E}_{q(Z|\bar{B})}[\log p(\bar{B}|Z)]$
 - 28: $\mathcal{L}_{FG} \leftarrow -\mathbb{E}_{p(\hat{X})}[\text{BD}(\hat{X}, H)]$
 - 29: $\theta_{FG} \leftarrow \theta_{FG} - \lambda_{FG} \cdot \text{RMSProp}(\theta_{FG}, \nabla_{\theta_{FG}} (\gamma \mathcal{L}_{like}^{points} + \mathcal{L}_{FG}))$
 - 30: **end if**
 - 31: **end while**
 - 32: **end while**
-

H Implementation details

H.1 Point cloud layers

The point cloud layers are implemented upon the Pytorch and Pytorch Geometric libraries. The latter allows to write geometric deep learning structures and operations easily; thus, it is particularly suitable for the development of this project’s layers. In addition, every operation based on Pytorch Geometric code is automatically provided with multi-GPU support. Consequently, the point cloud layers’ training and inference processes can be sped up in multi-GPU environments.

Given that every module in PyTorch subclasses the `torch.nn.Module`, the proposed point cloud layers are implemented following the same convention. This choice allows the layers to be straightforwardly plugged into other deep learning architectures for further research. Moreover, if an application requires multiple layers to be stacked horizontally, a sequential container like `torch.nn.Sequential` can be used right away.

H.1.1 Future improvements

The time complexity of the Shrinking and Extending Layer’s implementations can be slightly improved in GPU environments if a GPU-based K-means++ algorithm is utilised. Indeed, currently, the K-Means-Conv module employs the Scikit-learn version of K-Means++, which can only receive NumPy arrays as input. As such, tensors in the GPU need to be transferred to memory before the clustering operation, and then the results are transferred back to GPU, incurring unnecessary overhead. The Shrinking and Extending Layer implementations cannot be improved further, however.

Regarding the Stack Shrinking Layer’s implementation, its time complexity can be decreased substantially. In fact, the current approach for the parallel execution of multiple Shrinking Layers is by no means optimal as it is based on Python multithreading programming. The proper solution for achieving parallelism requires writing multithreaded code in native C/C++ and CUDA for the CPU and GPU environments, respectively, instead.

H.2 CC3DVAEWGAN architecture and training process

The CC3DVAEWGAN architecture and training process is fully implemented with Pytorch and Pytorch Geometric.

The training script is a distributed data parallel application. That is, the CC3DVAEWGAN model can be trained on multiple GPU processes and even

on multiple computer nodes simultaneously.

The structure of the training script enables the experimentation with different architectures and hyperparameter settings in a quick and straightforward manner. A Python module containing architectural information for each CC3DVAEWGAN’s component needs to be created and imported into the training script. The hyperparameters need to be set by providing an XML file’s path as a command-line argument. The XML file needs to contain several hyperparameter values in a specific format. Besides, the XML file contains a tag that allows restarting the training process with the help of a PTH file. The latter is a file generated at the end of each epoch to save the entire state of training in case of accidental crashes or for later performance analysis.

By default, the training script utilises the ModelNet10 dataset. If a different dataset needs to be employed, the ModelNet10 class in the training script must be replaced accordingly. Even though Pytorch Geometric provides a large number of common benchmark datasets that can be used straight away, if the replacing dataset is not contained in the library, it can be easily created using the dataset interface.

H.2.1 Requirements

The training script can only run in GPU environments. It would not be reasonable for it to support CPU execution because the training process would be very time-consuming otherwise. Given that the time and space complexity of the training process depends on the depth and type of architectures used for the CC3DVAEWGAN components, the batch size and the number of points contained in the point clouds, there are no recommended hardware requirements.

Model	Accuracy
RotationNet	98.46%
ShrinkingNet	91.25%

Table 1: ShrinkingNet and RotationNet[64] classification accuracies on ModelNet10. Currently, RotationNet is the state-of-the-art point cloud classifier on the ModelNet10 dataset.

I Experiments details

In this section, several experiments are conducted to establish the proposed approaches’ validity. Quantitative comparisons on point cloud classification, point cloud generation and computational efficiency are also provided. It is important to notice that these experiments by no means provide conclusive evidence. Hence, further experimentation is necessary to evaluate the performance of the proposed solutions.

I.1 Shrinking and Stack Shrinking Layers

The Shrinking and Stack Shrinking Layers have been trained on the ModelNet10 dataset [63] so to assess their feature extraction capabilities. Specifically, they have been used as feature extractors in a point cloud classification architecture called ShrinkingNet. As expected, empirical results show that the Stack Shrinking Layer outperforms the Shrinking Layer in feature extraction tasks.

As illustrated in Table 1, the best classification accuracy achieved so far on the ModelNet10 dataset is 91.25%, which is only 7% lower than the state-of-the-art proposal[64]. Figure 15 illustrates the ShrinkingNet’s architecture achieving the best performance on the ModelNet10 dataset. The Adam optimiser [65] with a learning rate of 0.01, the Cross-entropy loss, a batch size of 192 and 800 points sampled from each ModelNet’s CAD model have been used for the training process. The architecture has been trained for 200 epochs using 12x Nvidia Tesla V100.

This project strongly believes that ShrinkinNet has the potential to become the state-of-the-art point cloud classifier. Indeed, thanks to the properties of the Stack Shrinking Layer, its feature extractor component can be made much deeper and capable of extracting even more feature point clouds at each depth.

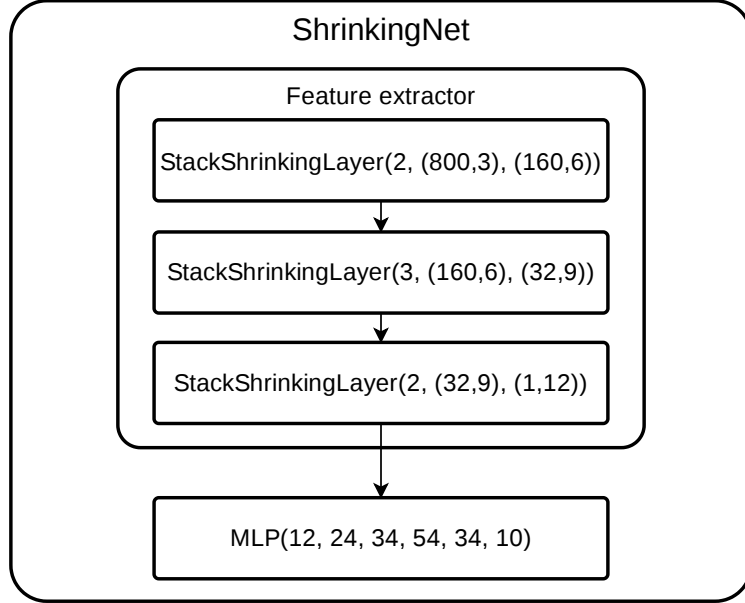


Figure 15: ShrinkingNet’s architecture achieving the best classification accuracy on ModelNet10. The feature extractor component consists of a horizontal stack of three Stack Shrinking Layers. In the notation `StackShrinkingLayer(...)`, the first argument is the n_F value, the second and third arguments are pairs of values such that the first and second elements denote the number of points and dimensionality of the input and output point clouds, respectively. Every MLP in each `StackShrinkingLayer` contains 7 layers such that the second, third, fourth, fifth and sixth layers have +10, +15, +20, +15, +10 neurons with respect to the input neurons. The activation function of the output neurons is Sigmoid. The remaining neurons use ReLu. The feature vector extracted from a given point cloud is fed into a MLP, which outputs a vector of probabilities for each class. The notation `MLP(...)` denotes the number of neurons at each layer. The activation function of each neuron is ReLu except the output neurons which use Softmax.

I.2 Real point cloud blueprints formation

The CC3DVAEWGAN training procedure relies on the K-Means++ algorithm to extract the point cloud blueprint of a given complete point cloud. Thus, ideally, the K-Means++ algorithm should capture the most significant structural information and output a skeleton of the complete point cloud input. This ideal property is satisfied if the density of points is roughly constant throughout the complete point cloud. Therefore, this point cloud blueprint formation procedure should fulfil its objective for most publicly available point cloud datasets. This holds because most publicly available point cloud datasets have approximately a constant point density.

In order to establish the validity of the previous claim, structural information has been extracted from multiple point cloud instances in the ModelNet and ShapeNet[66] datasets. Later visual analysis has confirmed that this project’s

point cloud blueprint formation procedure is satisfactory for these datasets. Unfortunately, to the best of this author’s knowledge, no objective metric has been proposed in the literature yet to measure the degree to which a point cloud blueprint describes the structural information of a complete point cloud.

Table 2 shows some point cloud instances and the point cloud blueprints computed from them. Table 3 shows from multiple perspectives the point cloud blueprint extracted from a chair. As in the CPCGAN paper, the point cloud blueprints contain 32 points.

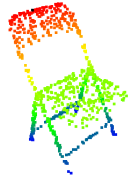



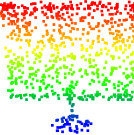
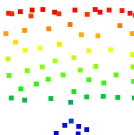
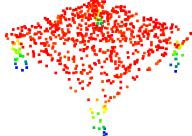
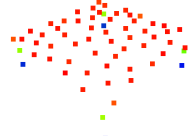
	Real point cloud	Point cloud blueprint
Chair		
Desk		
Monitor		
Table		

Table 2: Some point cloud instances from ModelNet10 and the point cloud blueprints computed from them via K-means++. The point cloud blueprints contain 32 points.









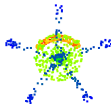
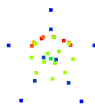


	Real point cloud	Point cloud blueprint
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 3: Point cloud blueprint extracted from a chair instance via K-Means++. The point cloud blueprint is shown from multiple perspectives. It contains 32 points.

I.3 Extending Layer

Several experiments have been conducted to establish whether Extending Layer-based generators have the ability to generate point clouds in a controllable fashion. That is, given a point cloud blueprint, it has been verified whether such generators can actually breed new points so to produce a complete point cloud that preserves the original structural information encoded in the blueprint.

These experiments do not leverage the Final WGAN architecture but rather investigate if it is plausible to drive these generators towards optimality by only means of a point cloud distance metric. For time complexity reasons, the Chamfer distance is employed even though the Earth mover’s distance generally achieves better results. A satisfactory outcome for these experiments would show that the Extending Layer can attain its objectives, and its perfor-

mance can be further enhanced when trained embedded in the Final WGAN architecture. The latter claim holds because the discriminator provides an additional error signal for the generator component.

The training algorithm used for the experiments is identical to Algorithm 5 but without the discriminator component and its error signals.

Figure 16 illustrates the Extending Layer-based generator’s architecture achieving so far the best performance on generative tasks. This architecture has been dubbed ExtendingNet. ExtendingNet has been trained using the RMSProp optimiser , a batch size of 64, a learning rate of 0.01, 800 and 32 as the number of points in the complete point clouds and point cloud blueprints, respectively.

Table 4 applies the metrics proposed in [20] to quantitatively compare the performance of ExtendingNet with the other approaches proposed in the literature. Table 5 performs a comparison on computational efficiency. Table 6 illustrates the per-class performance achieved by ExtendingNet when trained in the whole ModelNet10 dataset, showing that the latter can learn to generate multiple class instances simultaneously.

For visual analysis, Tables 7, 8, 9, 10 and 11 illustrate from multiple perspectives some real point cloud instances and the reconstructions generated by ExtendingNet from their point cloud blueprints. As it is possible to notice, the reconstructions are faithful representations but noisier than the original versions.

Further research is necessary to determine the optimal depth, the number of clustering regions, and the upsampling factor at each depth for a given point cloud dataset or class. Given that, currently, ExtendingNet employs a much smaller number of parameters in comparison to the other competitor architectures, the number of stacked Extending Layers can be increased to a considerable degree.

Nevertheless, considering that only the Chamfer distance metric has provided error signals to the generator, the results obtained are above expectations.

Class	Model	MMD-CD↓	COV-CD↑	BP-CD↓
Chair	r-GAN*[20]	0.00290	33.00	-
	Valsesia*[21]	0.02900	30.00	-
	tree-GAN[22]	0.00191	60.21	-
	CPCGAN[6]	0.00186	62.33	-
	ExtendingNet	0.00720	92.18	0.009
Airplane	r-GAN*[20]	0.00090	31.00	-
	Valsesia*[21]	0.00080	31.00	-
	tree-GAN[22]	0.00039	58.40	-
	CPCGAN[6]	0.00038	59.63	-
	ExtendingNet	0.00270	87.39	0.004

Table 4: Quantitative comparison in terms of the metrics proposed in [20]. Every model has been trained on the Shapenet "Chair" and "Airplane" classes separately. The "*" indicates that the results reported in [20] and [21] are cited for those models. Results of tree-GAN are cited from the CPCGAN paper. BP-CD is a metric proposed in this project. It measures the average Chamfer distance between the actual point cloud blueprints and the point cloud blueprints extracted from the generated point clouds. Bold values denote the best results. Both in the "Chair" and "Airplane" classes, ExtendingNet achieves the highest score in the COV-CD metric. This means that ExtendingNet is capable of representing most of the shapes in each class, demonstrating that the Extending Layer can be employed for point cloud controllable generation tasks. The MMD-CD metric measures the fidelity of representations. In this regard, ExtendingNet's performance is only slightly outperformed by its competitors.

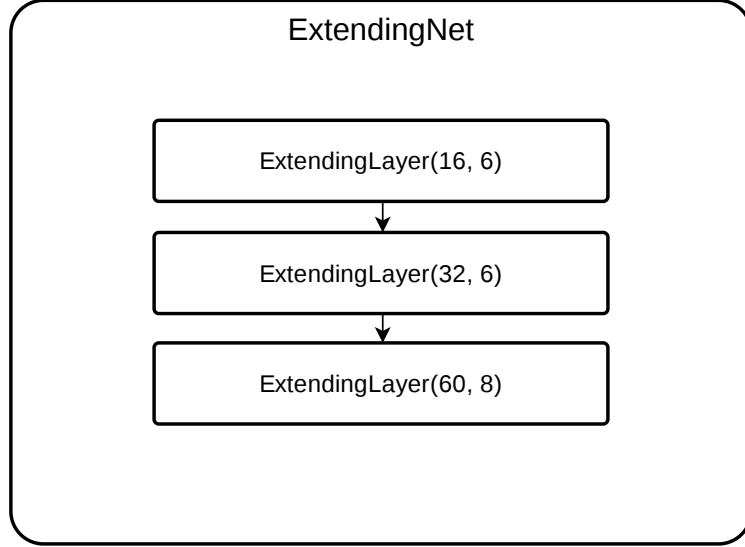


Figure 16: ExtendingNet’s architecture. In the notation $\text{ExtendingLayer}(\dots)$, the first and second arguments denote the number of clustered regions and the upsampling factor, respectively. Every MLP in each ExtendingLayer contains 7 layers such that the second, third, fourth, fifth and sixth layers have +10, +15, +20, +15, +10 neurons with respect to the input neurons. The activation function of the output neurons is Sigmoid. The remaining neurons use ReLu.

Model	#G parameters
tree-GAN[22]	40.690M
CPCGAN[6]	1.904M
ExtendingNet	36.138

Table 5: Comparison on computational efficiency. ”#G parameters” denotes the number of parameters in the generator component. The settings of tree-GAN and CPCGAN are the default settings proposed in their papers. As it is possible to notice, the number of parameters used in the ExtendingNet is significantly smaller than the number of parameters employed by its competitors.

Class	MMD-CD↓	COV-CD↑	BP-CD↓
Bathtub	0.0111	76.00	0.0168
Bed	0.0090	91.00	0.0140
Chair	0.0131	89.00	0.0204
Desk	0.0162	86.04	0.0218
Dresser	0.0231	61.62	0.0290
Monitor	0.0129	88.00	0.0192
Night stand	0.0223	72.09	0.0297
Sofa	0.0079	97.00	0.0128
Table	0.0115	73.00	0.0172
Toilet	0.0128	93.00	0.0189
Avg	0.0140	82.67	0.0200

Table 6: Per-class performance achieved by ExtendingNet when trained in the whole ModelNet10 dataset. MMD-CD and COV-CD are the metrics proposed in [20]. BP-CD is a metric proposed in this project. It measures the average Chamfer distance between the actual point cloud blueprints and the point cloud blueprints extracted from the generated point clouds. Bold values denote the best results. These results show that ExtendingNet can learn to generate multiple class instances simultaneously. No other paper in the literature has run this experiment, so no quantitative comparison can be conducted.

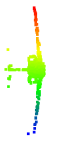

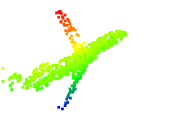
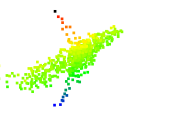
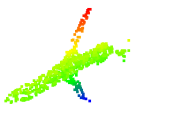
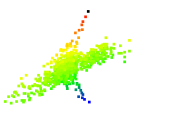
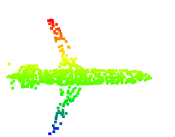
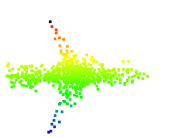
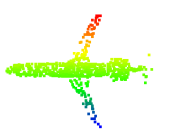
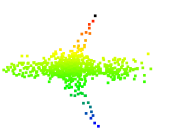
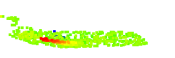
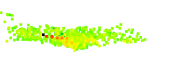
	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 7: Multi-view representation of an airplane point cloud instance and the reconstruction generated by ExtendingNet from its point cloud blueprint.


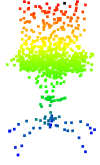

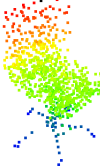

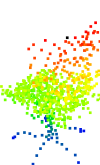
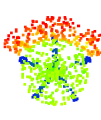
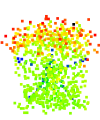
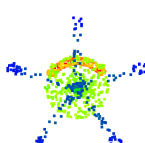
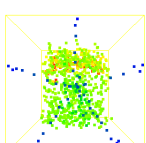

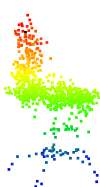
	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 8: Multi-view representation of a chair point cloud instance and the reconstruction generated by ExtendingNet from its point cloud blueprint.


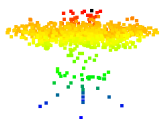
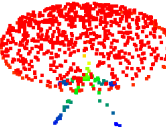
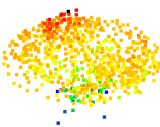
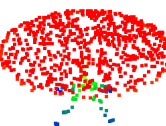
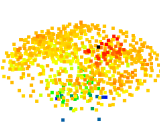
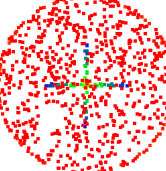
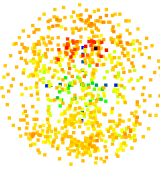
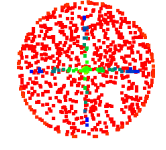
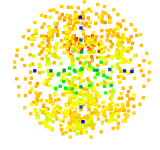
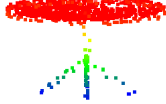
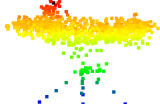
	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 9: Multi-view representation of a table point cloud instance and the reconstruction generated by ExtendingNet from its point cloud blueprint.

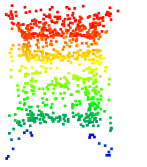
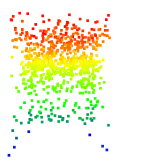
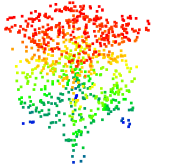
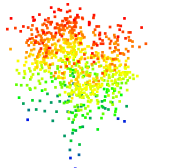
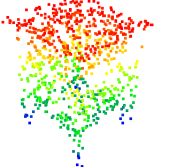
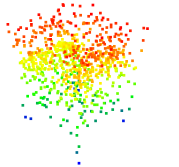
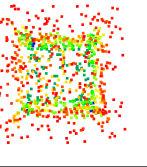
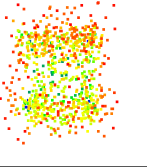
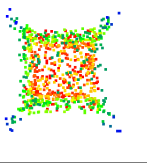
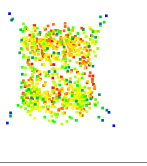
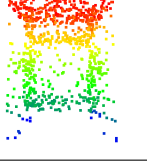
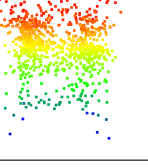
	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 10: Multi-view representation of a night stand point cloud instance and the reconstruction generated by ExtendingNet from its point cloud blueprint.

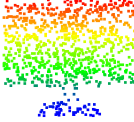
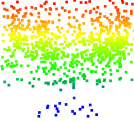
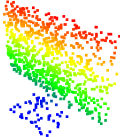
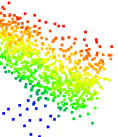
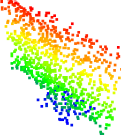
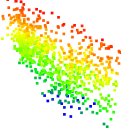
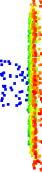
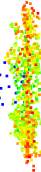
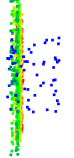
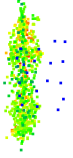
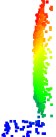

	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 11: Multi-view representation of a monitor point cloud instance and the reconstruction generated by ExtendingNet from its point cloud blueprint.

I.4 Final WGAN

The previous experiments conducted on the Extending Layer have achieved results above any expectations. Indeed, the previous section’s objective metrics and visual representations show that the ExtendingNet architecture has learned to generate complete point clouds from their corresponding blueprints by only means of the Chamfer distance metric.

Theoretically, much better results should be achieved when training ExtendingNet together with a discriminator component leveraging the Final WGAN architecture. This should occur because the discriminator component provides additional error signals.

In practice, however, the conducted experiment has not achieved the expected results. In fact, the results obtained are poorer than the ExtendingNet’s performance. The reason for such a failure stems from a common GAN training problem known as GAN convergence failure. Given that the generator and discriminator components used in the experiment are ExtendingNet and ShrinkingNet, respectively, which have proved to function to some degree, it is not clear why the training exhibits such an issue. As a consequence, further research is necessary.

Let the architecture used in the experiment be named Final WGAN Last. The algorithm used for the training process is Algorithm 5.

Table 12 shows the set of hyperparameters used for the training procedure. Figure 17 illustrates the Final WGAN Last architecture. Figure 18 is a line plot of the training and test losses of the Final WGAN Last generator and discriminator components. The latter clearly shows that the training process has not converged.

Table 13 performs a comparison on computational efficiency. This table highlights that the number of generator parameters in Final WGAN Last is much smaller than the number of parameters in the discriminator component. This might be the source of the non-convergence problem.

Table 14 performs a quantitative comparison of the Final WGAN Last’s performance with the other solutions proposed in the literature, including ExtendingNet as well. Table 15 illustrates the per-class performance achieved by Final WGAN Last in the whole ModelNet10 dataset. Both Table 14 and 15 demonstrate that the Final WGAN Last’s performance is much poorer than ExtendingNet.

For visual analysis, Tables 16, 17 and 18 illustrate from multiple perspectives some real point cloud instances and the reconstructions generated by Final WGAN Last from their point cloud blueprints. It is noticeable that the Final WGAN Last’s reconstructions are not as faithful as the ExtendingNet’s ones.

Hyperparam.	Value
λ_{FG}	0.01
λ_{FD}	0.01
m	64
$n_{blueprint}$	32
c	0.01
n_{FD}	5
γ	0.01

Table 12: Set of hyperparameters used for the Final WGAN Last’s training process. Refer to Algorithm 4 for a detailed description of each symbol’s semantic.

Model	#D params.	#G params.
tree-GAN[22]	2.536M	40.690M
CPCGAN[6]	2.693M	1.904M
Final WGAN Last	452.745	36.138

Table 13: Comparison on computational efficiency. ”#D params” denotes the number of parameters in the discriminator component.”#G params” denotes the number of parameters in the generator component. The settings of tree-GAN and CPCGAN are the default settings proposed in their papers. As it is possible to notice, the number of parameters used in the Final WGAN Last is significantly smaller than the number of parameters employed by its competitors. Also, the number of generator parameters in Final WGAN Last is much smaller than the number of parameters in the discriminator and encoder components. This may be the reason why the training is affected by non-convergence.

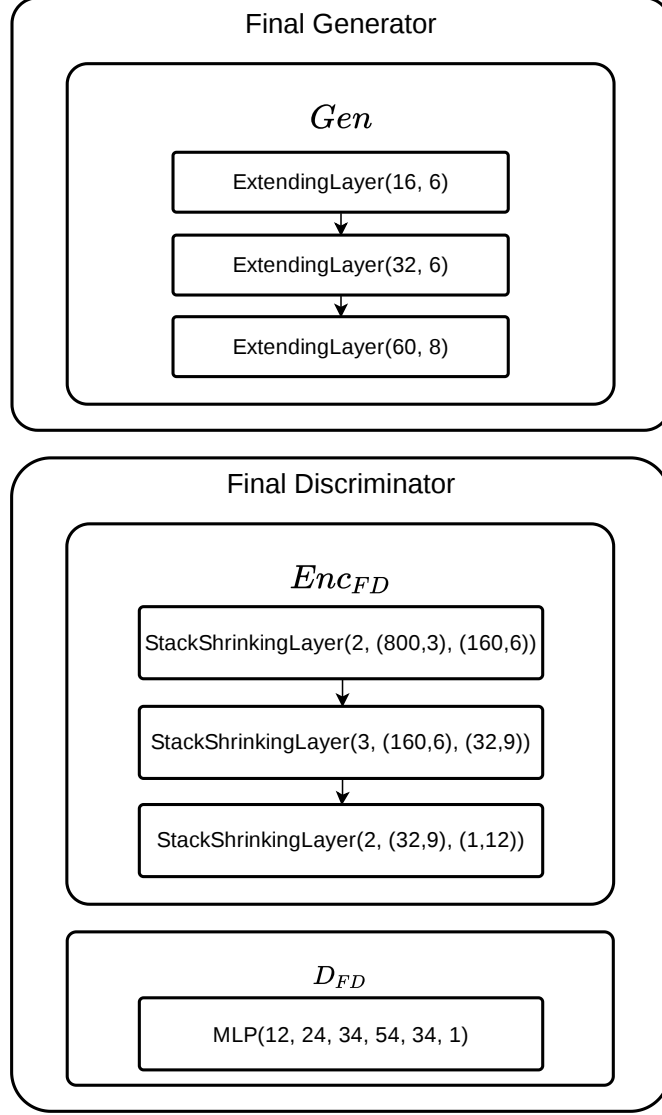


Figure 17: Architecture of Final WGAN Last. Refer to section 8 for an overview of the Final WGANs’s components functionalities. In the notation `StackShrinkingLayer(...)`, the first argument is the n_F value, the second and third arguments are pairs of values such that the first and second elements denote the number of points and dimensionality of the input and output point clouds, respectively. Every MLP in each `StackShrinkingLayer` contains 7 layers such that the second, third, fourth, fifth and sixth layers have +10, +15, +20, +15, +10 neurons with respect to the input neurons. The activation function of the output neurons is Sigmoid. The remaining neurons use ReLu. The notation `MLP(...)` denotes the number of neurons at each layer. The activation function of each neuron is ReLu except the output neuron, which does not use any activation function. In the notation `ExtendingLayer(...)`, the first and second arguments denote the number of clustered regions and the upsampling factor, respectively. Every MLP in each `ExtendingLayer` contains 7 layers such that the second, third, fourth, fifth and sixth layers have +10, +15, +20, +15, +10 neurons with respect to the input neurons. The activation function of the output neurons is Sigmoid. The remaining neurons use ReLu.

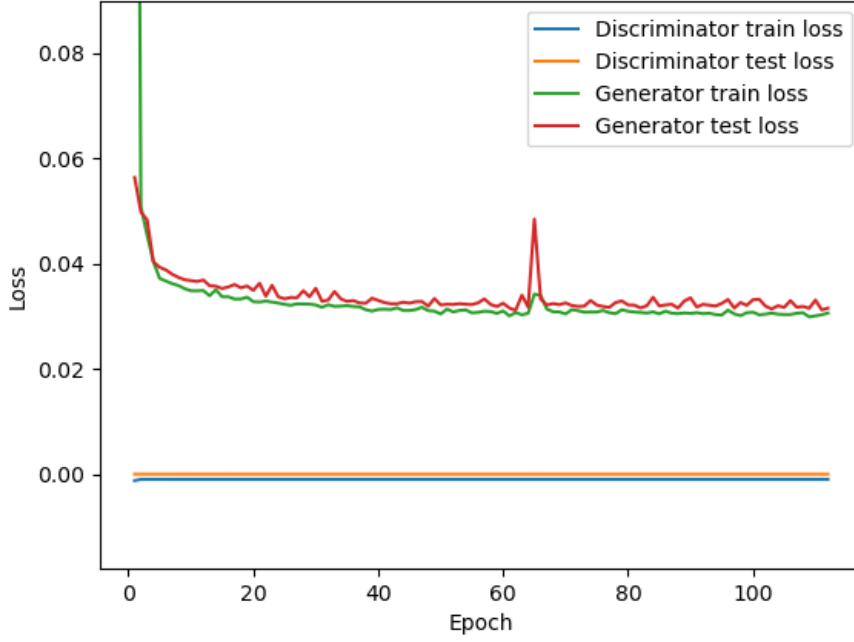


Figure 18: Line plot of the training and test losses of the Final WGAN Last’s generator and discriminator components. The discriminator loss starts and remains close to 0 during the whole training process, providing evidence that the training fails to converge.

Class	Model	MMD-CD↓	COV-CD↑	BP-CD↓
Chair	r-GAN[20]	0.00290	33.00	-
	Valsesia[21]	0.02900	30.00	-
	tree-GAN[22]	0.00191	60.21	-
	CPCGAN[6]	0.00186	62.33	-
	ExtendingNet	0.00720	92.18	0.009
	Final...Last	0.02349	35.22	0.026
Airplane	r-GAN[20]	0.0009	31.00	-
	Valsesia[21]	0.0008	31.00	-
	tree-GAN[22]	0.00039	58.40	-
	CPCGAN[6]	0.00038	59.63	-
	ExtendingNet	0.00270	87.39	0.004
	Final...Last	0.01472	28.73	0.018

Table 14: Quantitative comparison in terms of the metrics proposed in [20]. Every model has been trained on the Shapenet ”Chair” and ”Airplane” classes separately. The ”*” indicates that the results reported in [20] and [21] are cited for those models. Results of tree-GAN are cited from the CPCGAN paper. BP-CD is a metric proposed in this project. It measures the average Chamfer distance between the actual point cloud blueprints and the point cloud blueprints extracted from the generated point clouds. Bold values denote the best results. The results obtained by Final WGAN Last are much poorer than its competitors, also with respect to ExtendingNet.

Architecture	Class	MMD-CD↓	COV-CD↑	BP-CD↓
Final...Last	Bathtub	0.0146	25.00	0.0534
	Bed	0.0398	33.72	0.1046
	Chair	0.0222	24.00	0.0697
	Desk	0.0218	39.00	0.0596
	Dresser	0.0258	17.00	0.06123
	Monitor	0.0224	22.00	0.0703
	Night stand	0.0352	22.09	0.1016
	Sofa	0.0151	24.00	0.0523
	Table	0.0329	27.90	0.0843
	Toilet	0.0211	12.00	0.0614
	Avg	0.0251	24.66	0.0718
ExtendingNet	Bathtub	0.0111	76.00	0.0168
	Bed	0.0090	91.00	0.0140
	Chair	0.0131	89.00	0.0204
	Desk	0.0162	86.04	0.0218
	Dresser	0.0231	61.62	0.0290
	Monitor	0.0129	88.00	0.0192
	Night stand	0.0223	72.09	0.0297
	Sofa	0.0079	97.00	0.0128
	Table	0.0115	73.00	0.0172
	Toilet	0.0128	93.00	0.0189
	Avg	0.0140	82.67	0.0200

Table 15: (**Top**)Per-class performance achieved by Final WGAN Last when trained in the whole ModelNet10 dataset. (**Bottom**) For convenience, the per-class performance achieved by ExtendingNet in the whole ModelNet10 dataset is reported again. MMD-CD and COV-CD are the metrics proposed in [20]. BP-CD is a metric proposed in this project. It measures the average Chamfer distance between the actual point cloud blueprints and the point cloud blueprints extracted from the generated point clouds. Bold values denote the best results. It is evident that the performance achieved by Final WGAN is much poorer in comparison to ExtendingNet.


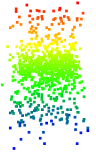

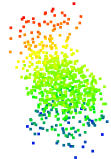

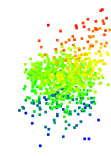
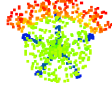
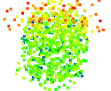
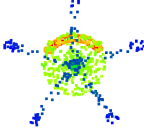
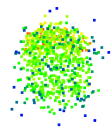

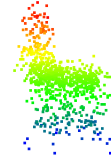
	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 16: Multi-view representation of a chair point cloud instance and the reconstruction generated by Final WGAN Last from its point cloud blueprint. It is evident that this reconstruction is much poorer than the reconstruction generated by ExtendingNet in Table 8.


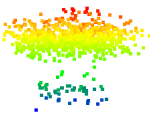
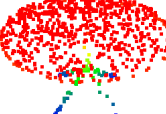
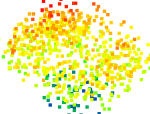
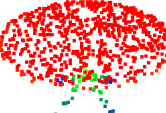
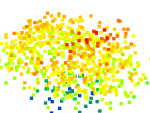
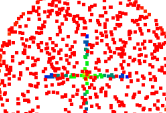
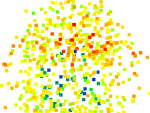
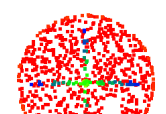
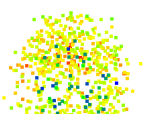

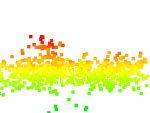
	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 17: Multi-view representation of a table point cloud instance and the reconstruction generated by Final WGAN Last from its point cloud blueprint. It is evident that this reconstruction is much poorer than the reconstruction generated by ExtendingNet in Table 9.

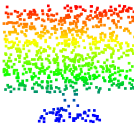
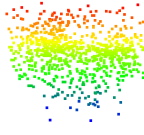
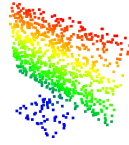
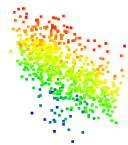
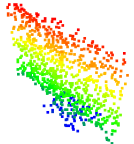
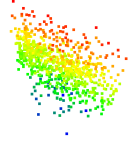
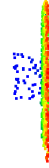
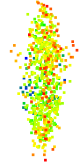
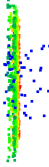
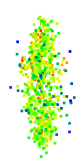

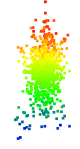
	Real point cloud	Generated point cloud
Front view		
Front isometric view		
Back isometric view		
Top view		
Bottom view		
Side view		

Table 18: Multi-view representation of a monitor point cloud instance and the reconstruction generated by Final WGAN Last from its point cloud blueprint. It is evident that this reconstruction is much poorer than the reconstruction generated by ExtendingNet in Table 11.

I.5 Blueprint VAE-WGAN

Blueprint VAE-WGAN’s objective is to simultaneously learn a latent space distribution for each training class and a function mapping a given latent representation to its corresponding point cloud blueprint. Several experiments have been carried out, and none of them has confirmed Blueprint VAE-WGAN’s validity. All the experiments have exhibited the same problem encountered in the Final WGAN’s training: GAN convergence failure.

Given that the Shrinking, Stack Shrinking and Extending Layers have proved to function to some extent, it is still not clear why the generator and discriminator components do not reach an equilibrium. Further research is necessary to solve this issue.

For completeness, here, the last experiment’s statistics are provided. For facilitating the caption of the following tables and figures, let the architecture used in the last experiment be called Blueprint VAE-WGAN Last. The algorithm used for the training process is Algorithm 3. The ModelNet10 dataset has been utilised.

Table 19 shows the set of hyperparameters used for the training procedure. Figures 19, 20 and 21 illustrate the architectures of the Blueprint VAE-WGAN Last’s components. Figure 22 is a line plot of the training and test losses of the Blueprint VAE-WGAN Last’s components. The latter provides evidence that the training process has not reached convergence.

Figure 23 and Table 20 demonstrate that Blueprint VAE-WGAN Last has not learnt a latent space distribution for each class and a mapping function from the latent to the point cloud blueprint domain. Table 21 performs a comparison on computational efficiency. The latter table provides an important insight: the number of generator parameters in Blueprint VAE-WGAN Last is much smaller than the number of parameters in the discriminator and encoder components. This may be the reason why the GAN training does not reach convergence.

No quantitative comparison on point cloud generation is provided for the Blueprint VAE-WGAN Last’s experiment because the architecture has not even started to learn due to the GAN convergence failure problem.

Hyperparam.	Value
λ_{BE}	0.01
λ_{BG}	0.01
λ_{BD}	0.01
m	64
$n_{blueprint}$	32
$n_{classes}$	10
c	0.01
n_{BD}	5
γ	0.01

Table 19: Set of hyperparameters used for the Blueprint VAE-WGAN Last’s training process. Refer to Algorithm 3 for a detailed description of each symbol’s semantic.

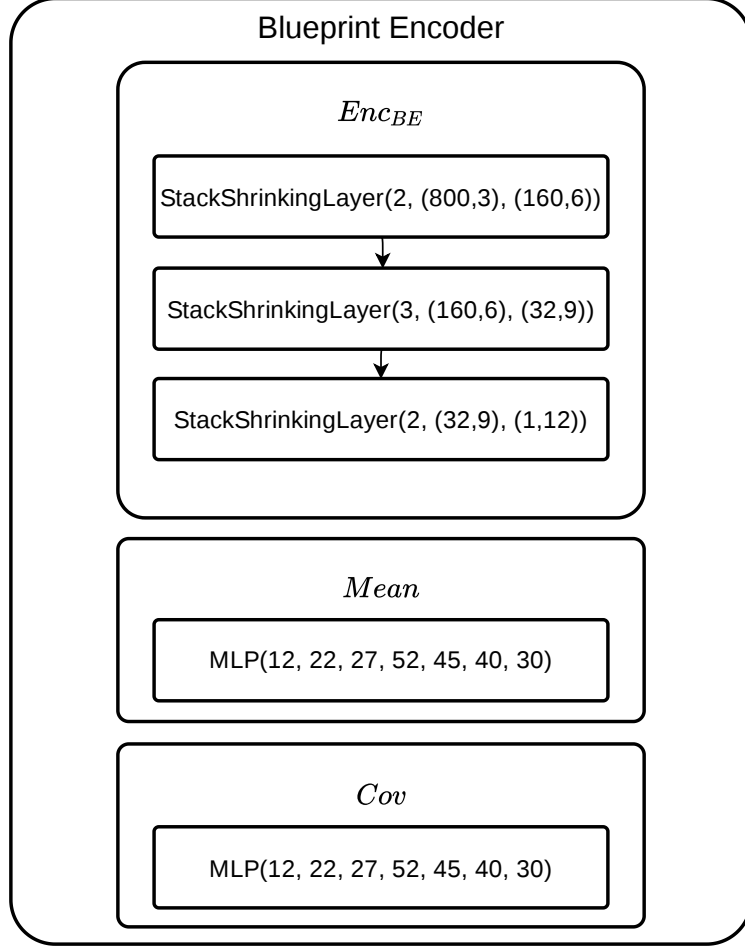


Figure 19: Architecture of the Blueprint Encoder component in Blueprint VAE-WGAN Last. Refer to section 7 for an overview of each Blueprint Encoder’s component functionalities. In the notation `StackShrinkingLayer(...)`, the first argument is the n_F value, the second and third arguments are pairs of values such that the first and second elements denote the number of points and dimensionality of the input and output point clouds, respectively. Every MLP in each `StackShrinkingLayer` contains 7 layers such that the second, third, fourth, fifth and sixth layers have +10, +15, +20, +15, +10 neurons with respect to the input neurons. The activation function of the output neurons is Sigmoid. The remaining neurons use ReLu. The notation `MLP(...)` denotes the number of neurons at each layer. The activation function of each neuron is ReLu except the output neurons, which do not use any activation function.

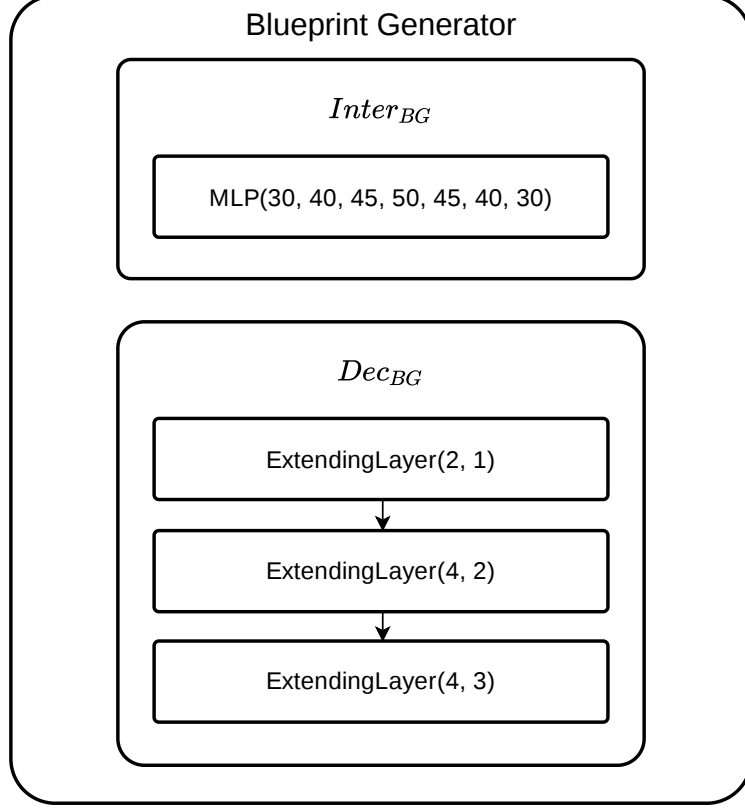


Figure 20: Architecture of the Blueprint Generator component in Blueprint VAE-WGAN Last. Refer to section 7 for an overview of each Blueprint Generator’s component functionalities. In the notation $\text{ExtendingLayer}(\dots)$, the first and second arguments denote the number of clustered regions and the upsampling factor, respectively. Every MLP in each ExtendingLayer contains 7 layers such that the second, third, fourth, fifth and sixth layers have +10, +15, +20, +15, +10 neurons with respect to the input neurons. The activation function of the output neurons is Sigmoid. The remaining neurons use ReLu. The notation $\text{MLP}(\dots)$ denotes the number of neurons at each layer. The activation function of each neuron is ReLu except the output neurons, which use Sigmoid.

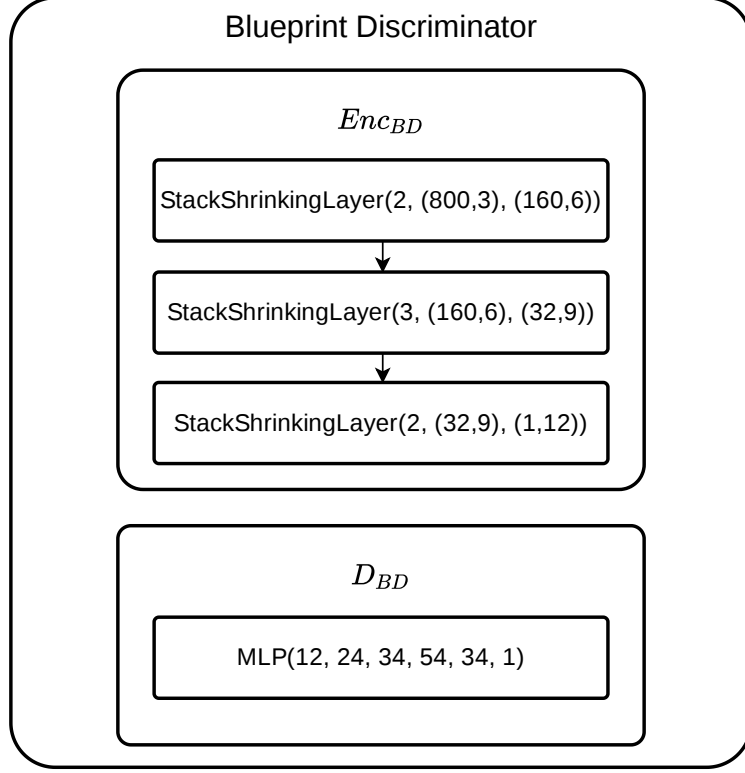


Figure 21: Architecture of the Blueprint Discriminator component in Blueprint VAE-WGAN Last. Refer to section 7 for an overview of each Blueprint Discriminator’s component functionalities. In the notation $\text{StackShrinkingLayer}(\dots)$, the first argument is the n_F value, the second and third arguments are pairs of values such that the first and second elements denote the number of points and dimensionality of the input and output point clouds, respectively. Every MLP in each $\text{StackShrinkingLayer}$ contains 7 layers such that the second, third, fourth, fifth and sixth layers have +10, +15, +20, +15, +10 neurons with respect to the input neurons. The activation function of the output neurons is Sigmoid. The remaining neurons use ReLu. The notation $\text{MLP}(\dots)$ denotes the number of neurons at each layer. The activation function of each neuron is ReLu except the output neuron, which does not use any activation function.

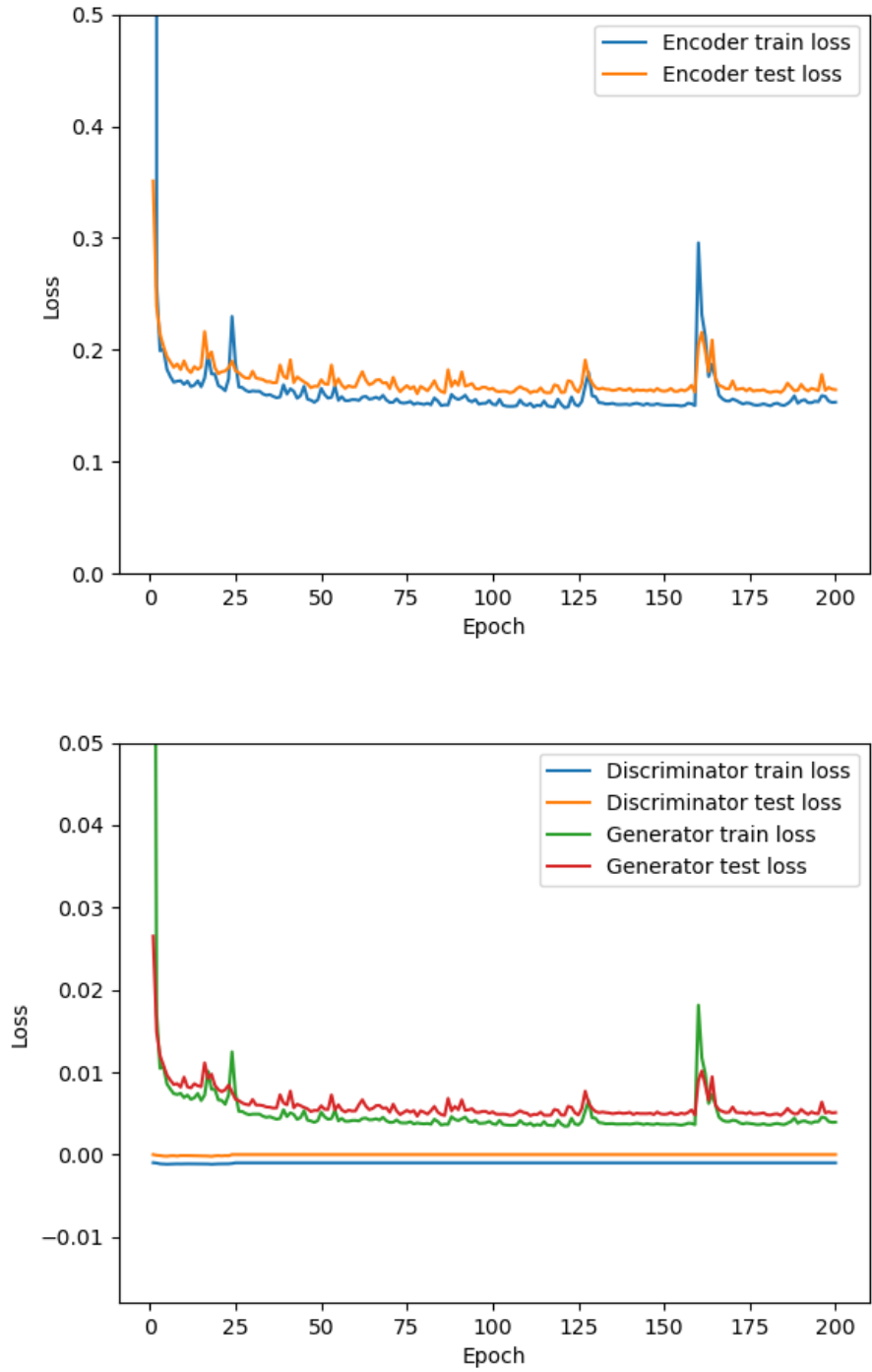


Figure 22: **(Top)** Line plot of the training and test losses of the Blueprint VAE-WGAN Last’s encoder component. **(Bottom)** Line plot of the training and test losses of the Blueprint VAE-WGAN Last’s generator and discriminator components. The discriminator loss starts and remains close to 0 during the whole training process, providing evidence that the training fails to converge.

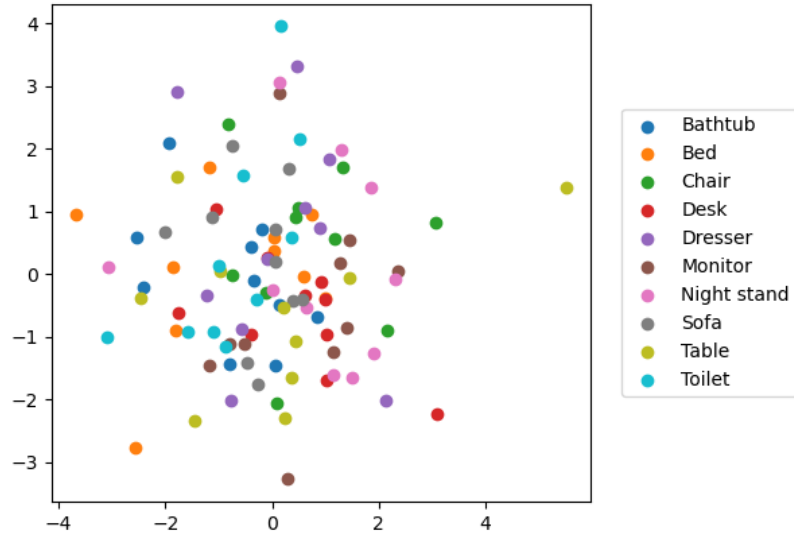


Figure 23: Scatter plot of the latent representations drawn from each class latent space learned by Blueprint VAE-WGAN Last for the ModelNet10 dataset. For visualisation purposes, given that Blueprint VAE-WGAN learns class latent spaces in 30 dimensions, Principal Component Analysis(PCA) is used to reduce them to 2 dimensions. It is evident that Blueprint VAE-WGAN has not learnt the real class latent spaces as the latent codes drawn from each class latent space heavily clutter with the other classes' latent representations.

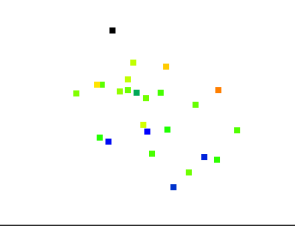
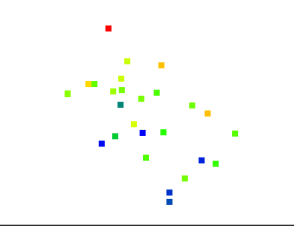
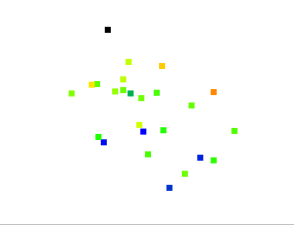
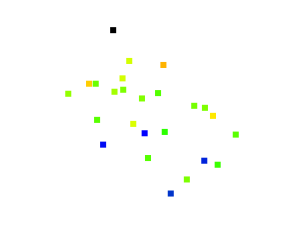
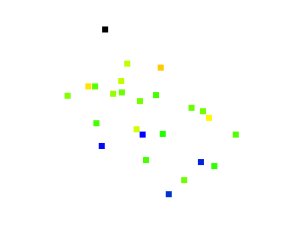
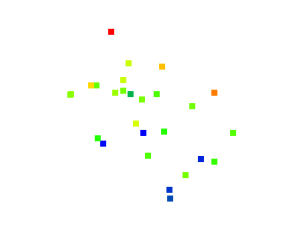
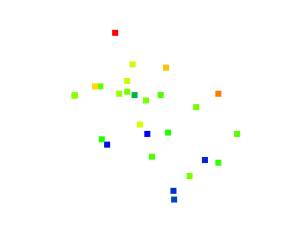
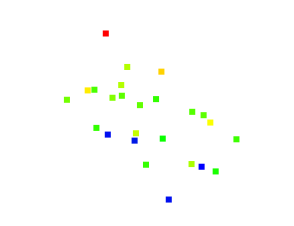
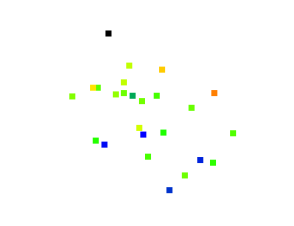
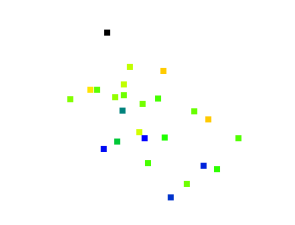
Bathtub	Bed	Chair
		
Desk	Dresser	Monitor
		
Night stand	Sofa	Table
		
Toilet	-	-
	-	-

Table 20: Representation of some point cloud blueprint instances generated by Blueprint VAE-WGAN Last. For each class in ModelNet10, a latent representation drawn from the learnt class latent space distribution is fed into the network using the conditional generation mode. Refer to section 6.4 for further information on the conditional generation mode. It is noticeable that Blueprint VAE-WGAN has not learnt the mapping function from the latent to the point cloud blueprint domain. A reason for this stems from the fact that Blueprint VAE-WGAN has not learnt each class latent space distribution, as shown in Figure 23

Model	#E params.	#D params.	#G params.
tree-GAN[22]	-	2.536M	40.690M
CPCGAN[6]	-	2.693M	1.904M
Blueprint ... Last	696.812	717.963	46.384

Table 21: Comparison on computational efficiency. ”#E params” denotes the number of parameters in the encoder component.”#D params” denotes the number of parameters in the discriminator component.”#G params” denotes the number of parameters in the generator component. The settings of tree-GAN and CPCGAN are the default settings proposed in their papers. As it is possible to notice, the number of parameters used in the Blueprint VAE-WGAN Last is significantly smaller than the number of parameters employed by its competitors. Also, the number of generator parameters in Blueprint VAE-WGAN Last is much smaller than the number of parameters in the discriminator and encoder components. This may be the reason why the training is affected by non-convergence.

J Project Management details

J.1 Selection and formulation of research problem

Prior to the start of this academic year, I had a summer undergraduate research internship at the Mainz University of Applied Sciences under the supervision of PhD student Bastian Plass. I was tasked with searching for relevant literature on point cloud feature extraction. Noticing some conceptual gaps in the proposed solutions, I developed and implemented the Shrinking and Stack Shrinking Layers.

At the start of the autumn term, with the aim of carrying out a third-year project related to the point cloud field, I read relevant literature on point cloud generation. I discovered that the problem of controllable and conditional generation was not addressed yet; thus, I developed a controllable and conditional point cloud generator, CC3DVAEWGAN. I also noticed that no point cloud generator uses state-of-the-art architectures for feature extraction and generation, so I decided to employ the Shrinking Layer or Stack Shrinking Layer for feature extraction, and I developed a new layer, dubbed Extending Layer, for point cloud generation.

Thanks to the summer internship and the fair amount of time spent, I was able to perform all of these developments during the first two weeks of the autumn term. The summer research internship has heavily helped me develop the right attitude toward research projects, and consequently, it has also assisted me in developing this project.

J.2 Initial planning

Shortly prior to submitting the project brief, a plan for the whole project was set up. Only the required macro-activities were depicted as, given the research nature of this work, it was not possible to anticipate the necessary micro-activities in advance. As the project progressed, however, the micro-activities were recognised and performed.

Figure 24 and 25 provide a flow chart and a Gantt chart of this project's macro-activities plan. The flow chart is used to represent the logic sequence, while the Gantt chart highlights the timing of the activities. Even though the charts illustrate a precise sequence of activities, this sequential progression is to be taken as a rough indication. Indeed, research projects do not lend themselves to pure sequential steps as they usually comprise a mixture of sequential and cyclic stages.

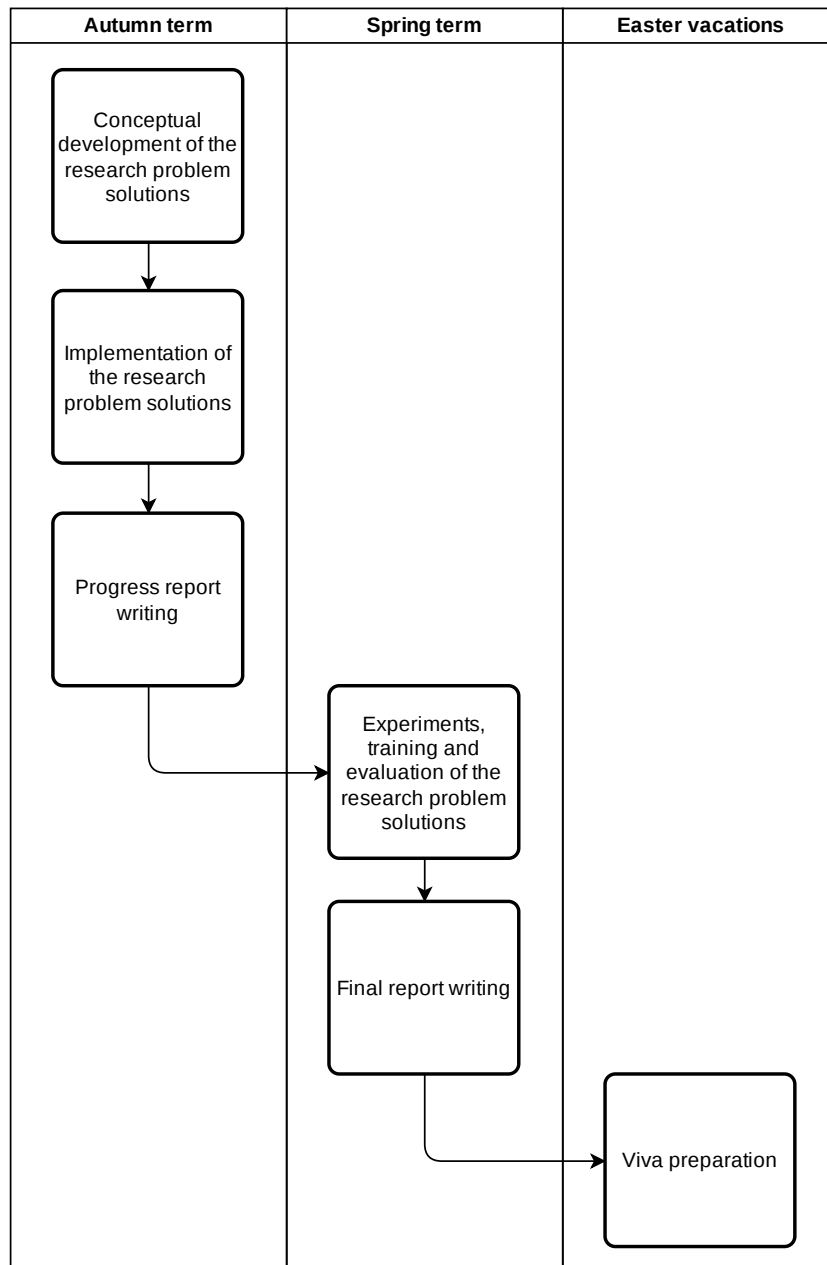


Figure 24: Flow chart of this project's macro-activities plan

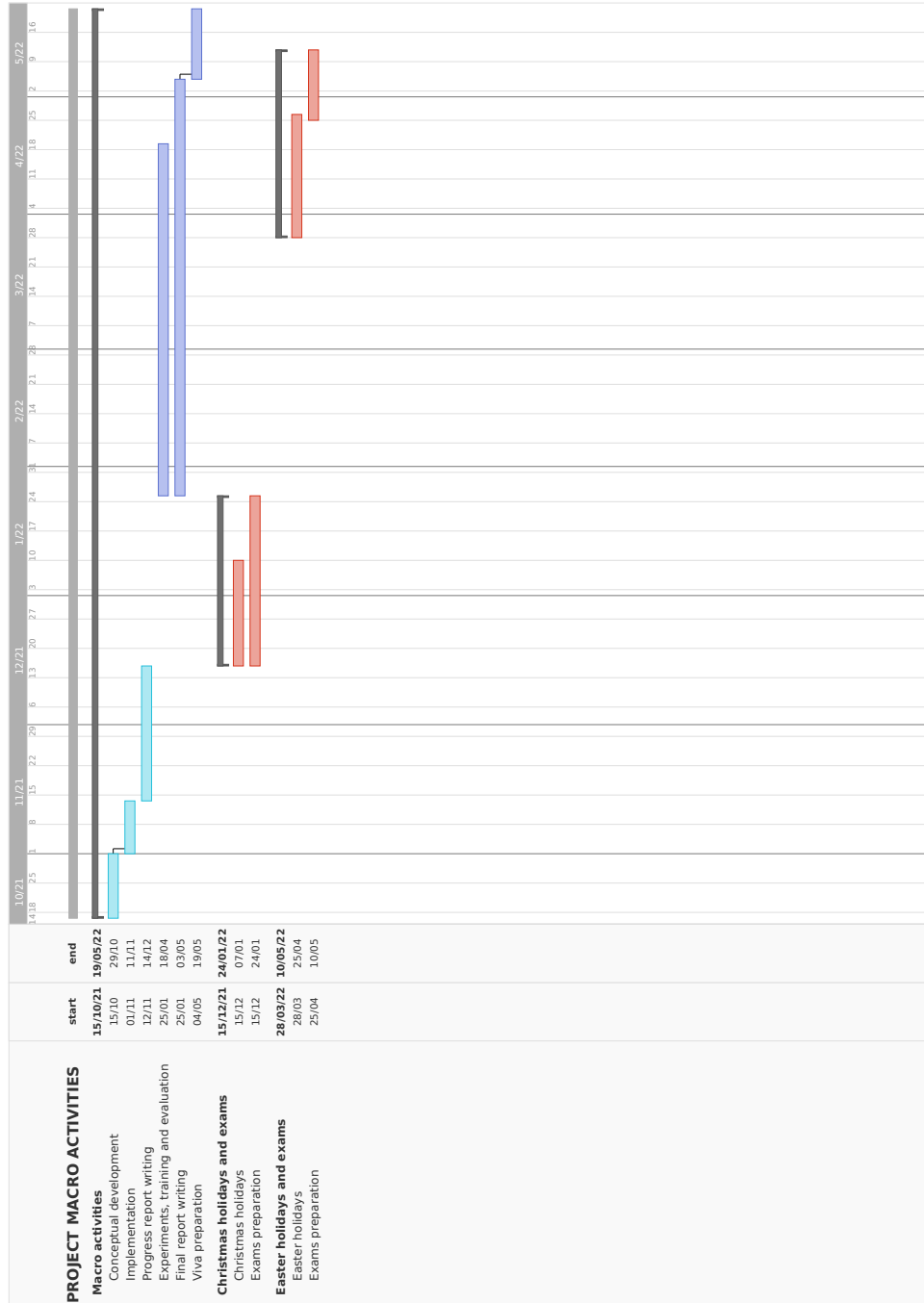


Figure 25: Gantt chart of this project's macro-activities plan

J.3 Definitive planning

Some weeks after the submission of the project brief, a more detailed and definitive project Gantt chart was developed. This Gantt chart was submitted together with the progress report. For convenience, it is re-proposed here in Figure 26.

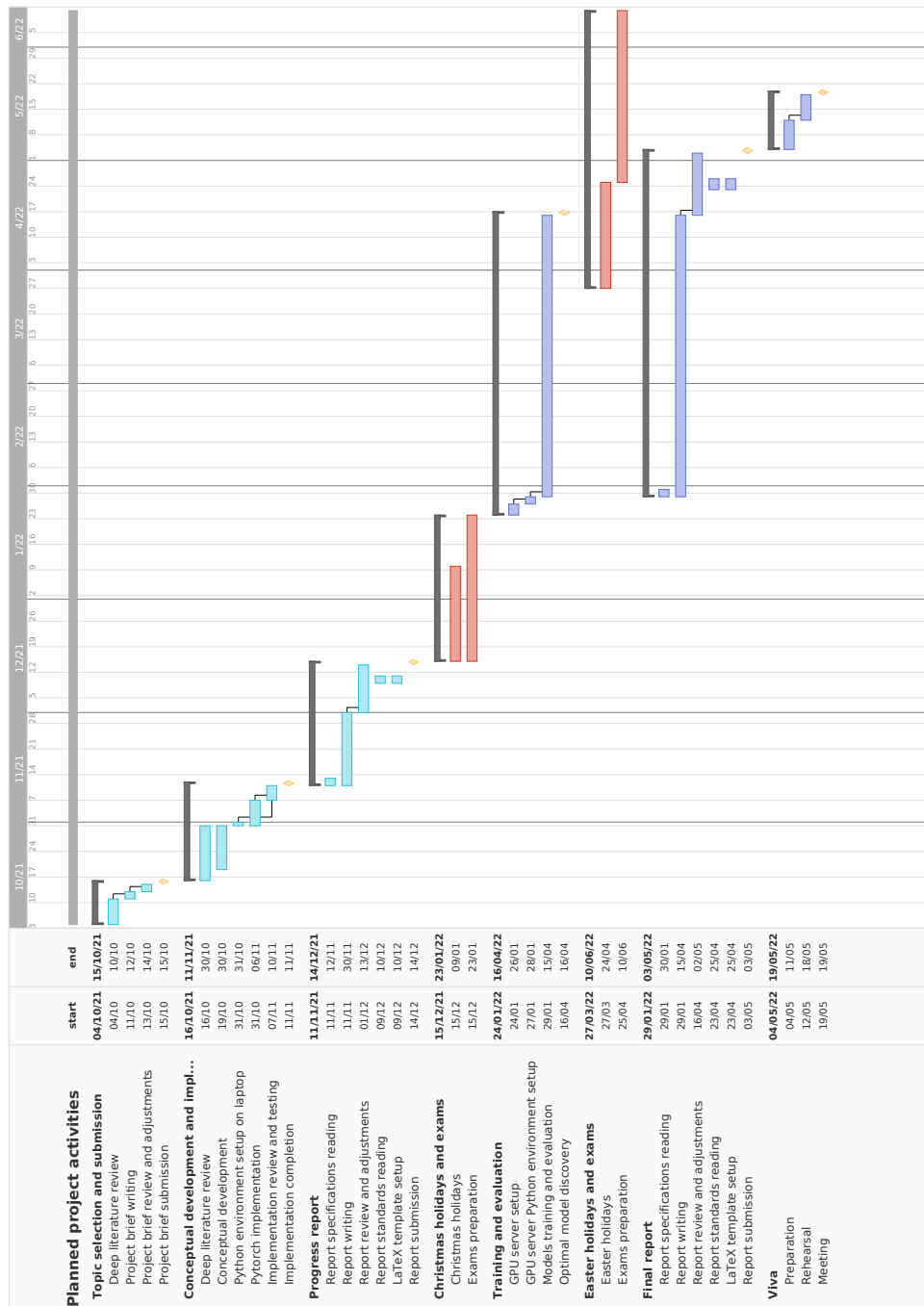


Figure 26: Gantt chart of this project's planned activities

J.4 Progress

Here, a brief account of the actual work performed for the completion of this project is provided. For a more detailed snapshot of the performed activities, the reader shall consult the actual project Gantt chart in Figure 27. It is important to notice that such Gantt chart only provides a rough indication of the performed activities as, given the research nature of this project, some activities were performed in a cyclic fashion.

During the autumn term, I read the relevant literature on the topic of controllable and conditional generation. Given that this research problem was only partially addressed in the point cloud literature, I predominantly searched for information in the image generation field. [16] inspired me the development of the Blueprint VAE-WGAN architecture and training algorithm. The CPCGAN paper, previously the only proposal in the point cloud controllable generation problem, influenced the CC3DVAEWGAN controllable generation mode.

Afterwards, I implemented the CC3DVAEWGAN architecture, its training algorithm and the Extending Layer. However, the most time-consuming task of the autumn term was the progress report writing. I found it a little bit challenging to condense in only 3000 words both the point cloud generation and feature extraction literature, the Shrinking Layer, Stack Shrinking Layer and Extending Layer functionalities and ultimately the CC3DVAEWGAN architecture and generation modes. Overall, the progress achieved during the autumn term went according to the planned project Gantt chart in Figure 26.

By comparing the planned project Gantt chart in Figure 26 and the actual project Gantt chart in Figure 27, it is noticeable that, unlike the autumn term, the spring term went slightly off schedule. The reason for it is that after the exams period, I reviewed the work done and made adjustments to some training and architectural components to fix some conceptual mistakes. I also added new training and architectural elements to the proposed solutions. Hence, I had to reimplement some code and add new lines of implementation. Besides, some new package dependencies had to be employed, which proved difficult to install in the cluster Python environment.

The original plan described in the progress report involved the training of the whole CC3DVAEWGAN architecture during the spring term. However, in February, I realised that a bottom-up experimentation process was more suitable for this project, given the large number of new proposals. This bottom-up approach consisted of conducting experiments for each new proposal in isolation so to really demonstrate whether they function or not without any interference from the other components. Even though many new training scripts had to be implemented, this approach proved beneficial for the outcome of this project.

I also expected to conduct a larger number of experiments. However, the limited number of GPUs and the time required for the writing of this report

reduced the time available for experimentation. Indeed, initially, I did not expect the final report to be so voluminous.

Finally, during the Easter vacations, I devised a new point cloud generation layer, named Stack Extending Layer, which I did not expect to invent.

In summary, while the autumn term progress went according to plan, during the spring term, an extended number of unexpected activities, which could not have been forecasted in advance, were carried out. Nevertheless, I compensated for them by spending a large number of hours on the completion of this project during the Easter vacations. Since the start of this project, I have considered Easter vacations as the main contingency plan for the completion of this project.

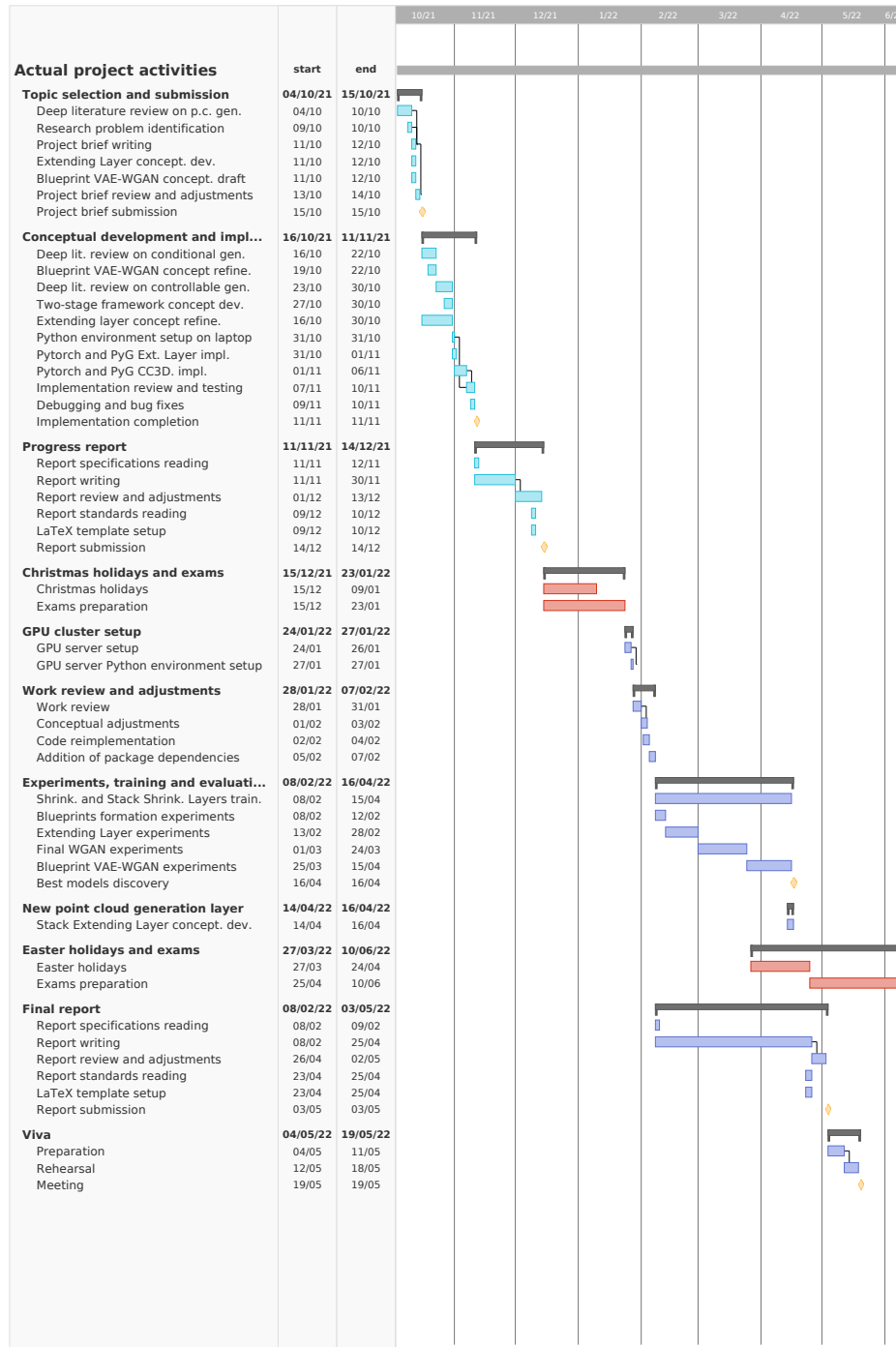


Figure 27: Gantt chart of this project's actual activities

J.5 Time management

This project proposes a large number of new and ambitious approaches in the point cloud literature. As a consequence, its completion has necessitated a more extended period of development than the one suggested in the Part III Individual Project module syllabus.

According to the module syllabus, 450 hours should be allocated to the project.

However, the completion of this work has required approximately 720 hours, which have been distributed over the academic terms unevenly.

During the autumn term, 20 hours per week were allocated for a total of 200 hours over the whole period. This rate achieved a perfectly balanced workload for the first semester.

The number of hours utilised during the spring term was 320, that is, 40 hours per week. Given the more considerable amount of work to carry out, it was necessary to double the number of weekly hours in comparison to the autumn term. Thankfully, the light workload of the second term modules allowed for such extension in the project development time allocation.

The project was not entirely completed during the spring term. Therefore, further tasks had to be performed during the Easter vacations. In total, 200 hours were spent during the Easter vacations on the completion of the project.

Calculations at the beginning of the autumn term showed that a maximum project time allocation of 900 hours would have been feasible for my academic rhythm. Therefore, the actual total project time allocation did not go over the maximum limit.

Figure 28 shows the suggested project time allocation versus the actual one. Figure 29 illustrates the distribution of this project's time allocation over the academic terms and vacations. Figure 30 compares the maximum project time allocation versus the actual allocation.

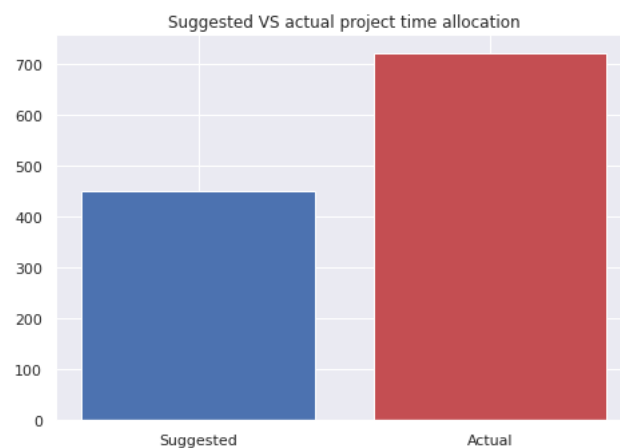


Figure 28: Suggested project time allocation vs actual one.

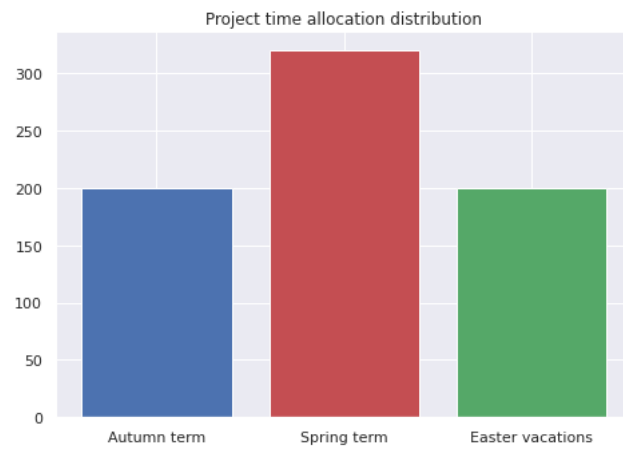


Figure 29: Project time allocation distribution over the academic terms and vacations

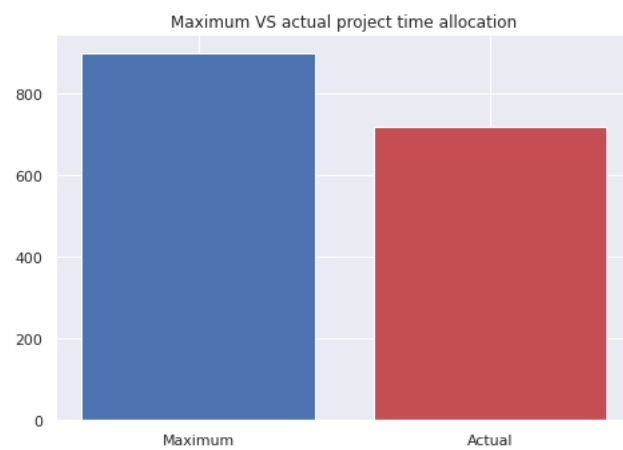


Figure 30: Maximum vs actual project time allocation

J.6 Risks analysis and contingency plans

The following table provides an analysis of the project's risks and their contingency plans. "P" and "S" stand for risk probability and risk severity, respectively. Their measurement scale ranges from 1 to 5 with discrete steps. "RE" stands for risk exposure and is equal to "P" \times "S".

Risk	P	S	RE	Contingency plans
Deletion of the project's proposal implementations	1	5	5	Create a GIT repository and make frequent commits. Sync OneDrive folder.
Deletion of the project's documents and training files	1	5	5	Sync OneDrive folder.
Deletion of the project's GIT repository	1	5	5	Make frequent local copies so that to have a backup to fall back to without losing progress.
Lack of available ECS GPU Clusters for High-Performance Computing	2	3	6	<ol style="list-style-type: none">1. Consult supervisor2. Ask the i3mainz research institute for help3. Rent GPU cloud servers4. Use own machine equipped with a single GPU
Inadequate number of GPUs available for training	3	3	9	Reduce the complexity of the proposed solutions by decreasing the number of stacked layers. This would reduce performance, however. Highlights this fact in the final report.
Abrupt interruptions while batch training	3	1	3	Save training state frequently in .pth files to recover training.

Difficulties with understanding research papers	2	3	6	<ol style="list-style-type: none"> 1. Ask the project supervisor for help 2. Request help from the University of Southampton 3. Ask the i3mainz research institute for help 4. Contact the authors of the paper
Difficulties with the implementation of the project's architectures and training algorithms	2	4	8	<ol style="list-style-type: none"> 1. Consult the best forums for programmers to search for similar problems and the solutions proposed by the community 2. Ask the project supervisor for help 3. Post questions on the best forums for programmers 4. Open issues on GitHub
Difficulties with setting up access to the ECS GPU computer facilities	2	4	8	<ol style="list-style-type: none"> 1. Read the FAQ 2. Ask the project supervisor for help 3. Contact the ECS GPU computer facilities technical support team

Difficulties with setting up the project's Python environment in the ECS GPU computer facilities	2	4	8	<ol style="list-style-type: none"> 1. Consult the best forums for programmers to search for similar problems and the solutions proposed by the community 2. Ask the project supervisor for help 3. Contact the ECS GPU computer facilities technical support team 4. Post questions on the best forums for programmers 5. Open issues on GitHub
Non-implementation of a project's proposed architecture or training procedure	2	3	6	Explain in the final report the reason why the architecture or training has not been implemented. Provide an extensive theoretical description of the architecture or training algorithm and write down the expected results. Maybe, it would also be necessary to describe the steps planned for the implementation had it occurred so that future projects can take advantage of it.
Impossibility of improving poor empirical results	4	1	4	Write down the outcome of the conducted experiments in the final report. Explain the reason why further experiments have not been conducted. In addition, perform a preliminary analysis of the factors behind the inadequate outcomes and assert the future research direction to be taken.

Impossibility of carrying out experiments for a project's proposal	3	3	9	Explain the reason why experiments have not been conducted for that project's proposal. Also, elucidate the details of the experiments had they been conducted so that future projects can take advantage of them.
Runtime exceptions raised while training	4	1	4	<ol style="list-style-type: none"> 1. Inspect and debug code 2. Consult the best forums for programmers to search for similar problems and the solutions proposed by the community 3. Ask the project supervisor for help 4. Post questions on the best forums for programmers 5. Open issues on GitHub
Struggles with finding solutions to the research problem	2	5	10	<ol style="list-style-type: none"> 1. Consult the project supervisor 2. Explain the motivations behind such struggles in the final report
Slow training process	2	3	6	<ol style="list-style-type: none"> 1. Optimise implementation 2. Consult the best forums for programmers to search for similar problems and the solutions proposed by the community 3. Ask the project supervisor for suggestions
Project aims not fully achieved before the final deadline	3	3	9	Write down the motivations behind this failure in the final report

Emergence of negative affective states such as anxiety, depression, loss of self-esteem and altered behavior	1	5	5	Rest, meditation and music.
Overly ambitious research proposal	4	3	12	<ol style="list-style-type: none"> 1. Full commitment to the project 2. Hard work 3. Declare in the final report insufficient skills for the successful completion of the project
Occurrence of adverse events preventing commitment to the project	2	5	10	<ol style="list-style-type: none"> 1. Consult the project supervisor 2. Ask for the final project delivery deadlines to be postponed
Unsolvable research problem	2	5	10	Rigorously prove why the research problem is not solvable
Heavy workload for the other modules	2	4	8	Create schedule, prioritise and balance