Point-JEPA: A Joint-Embedding Predictive Architecture for Self-Supervised Learning on Point Cloud

Ayumu Saito

Submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science, Honours Computing Science

at

Saint Mary's University
Halifax, Nova Scotia
Canada

Approved by:	Dr. Jiju Poovvancheri Supervisor
Approved by:	Dr. Mengjun Hu Reader
Date:	April 30, 2024

Abstract

Recent advancements in self-supervised learning in the point cloud domain have demonstrated significant potential. However, these methods often suffer from drawbacks, including lengthy pre-training time, the necessity of reconstruction in the input space, or the necessity of additional modalities. In order to address these issues, we introduce Point-JEPA, a joint embedding predictive architecture designed specifically for the point cloud domain. We introduce a sequencer that orders point cloud tokens to efficiently compute and utilize tokens' proximity based on their indices. This allows shared computation of proximity for point cloud tokens, allowing the efficient selection of spatially contiguous context and target blocks. Experimentally, our method achieves competitive results with state-of-the-art methods while avoiding the reconstruction in the input space or additional modality. Specifically, it outperforms other self-supervised learning methods on linear evaluation and few-shot classification on ModelNet40, showing the robustness of the learned representation. The results show that Point-JEPA is an alternative efficient pre-training method to pre-existing methods in the point cloud domain.

Acknowledgments

This study was made possible with the help of several individuals, to whom I owe my deepest gratitude. I would first like to express my sincere gratitude to my parents, Jen, and the Harrison family for their tremendous support throughout my undergraduate years. I would also like to thank Will Stone and the folks at ReelData who helped me form a good understanding of deep learning and provided me with interesting ideas. It also goes without saying that this study simply could not have been done without their computing resources. I would also like to thank Dr. Jiju Poovvancheri for his guidance during this research. Finally, I would like to send my gratitude to the members of the Department of Mathematics and Computing Science, who made my time at Saint Mary's interesting and enjoyable.

Ayumu Saito
April 30, 2024

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Point cloud representation is employed to capture accurate 3D geometry information. Because of the increasing availability of the 3D sensors, we have seen an increase in the available point cloud datasets [1, 2, 3, 4, 5, 6], leading to numerous studies that directly process this type of data [7, 8, 9, 10]. However, the impressive results of these studies often rely on a large amount of labeled data, which is not always available to practitioners working with their custom datasets. To illustrate this problem, semantic annotation of one object from the ScanNet dataset takes roughly 22 minutes [5]. Self-supervised learning, on the other hand, does not require labeled data. This enables the learning of strong representations from a vast amount of data, resulting in prominent performance gains in the image and natural language processing domains [11, 12, 13, 14, 15, 16, 17]. Inspired by the success in these domains, we have also seen applications in the point cloud domain [18, 19, 20, 21], achieving state-of-the-art results.

While the current state-of-the-art methods, such as PointGPT [19] and Point2Vec [18], can produce promising results on classification and part segmentation tasks, our initial investigation found that they take a significant amount of time for pre-training.

Specifically, our initial investigation found that it takes 16.9 hours for PointGPT-S and 12.1 hours for Point2Vec to pre-train on NVIDIA RTX A5500. This enormous time for pre-training is less than ideal as it becomes harder to scale to a larger dataset, which hinders the ability to effectively learn a strong representation from a vast number of unlabeled training data. Motivated by the recent application of the joint-embedding predictive architecture [22] in the image domain [23] and its efficiency in pre-training time, we aim to apply the architecture to point cloud objects for self-supervised pre-training.

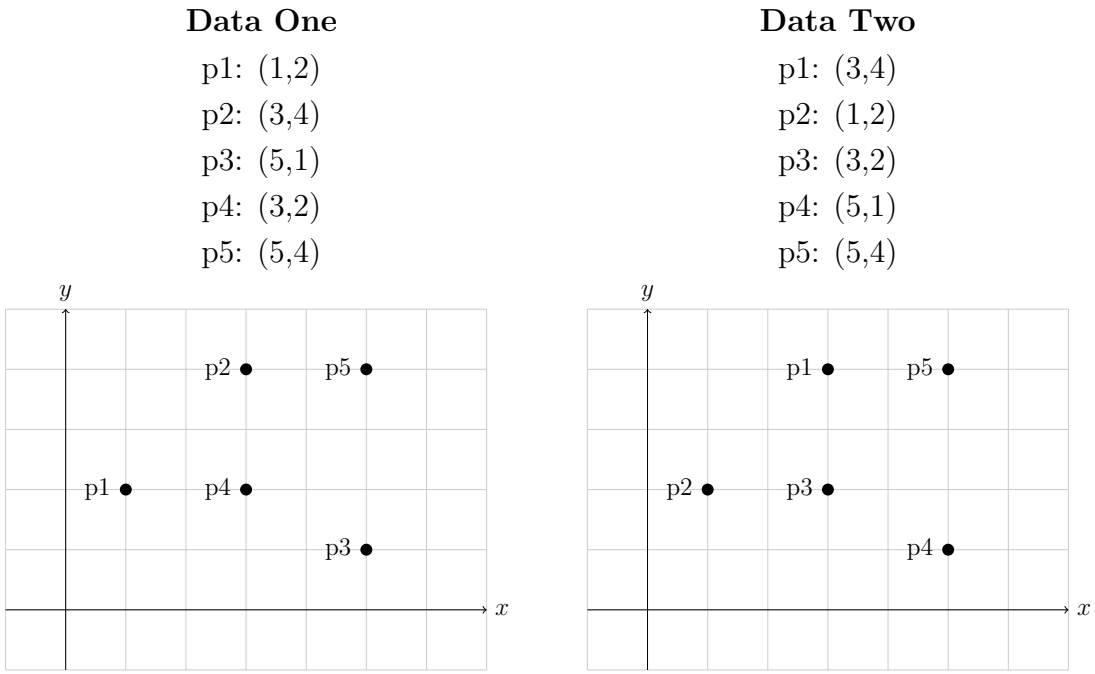

Figure 1.1: Illustration of the unordered nature of point cloud data. A permutation is applied to data one to produce data two. As the plot shows, two data represent the same set of points in two-dimensional space.

1.2 Challenges

Unlike images, a point cloud is a set of points with no specific order. Regardless of the permutations applied to the set, the set represents the same object. This unordered nature is shown in Figure 1.1 where, even after permuting the order of the points, the overall shape they form remains the same. This implies that the model consuming N points needs to be invariant to $N!$ permutations of the order of the input data [9]. Although our work utilizes a PointNet-like encoder to create a set of tokens, ensuring the permutation invariance in the local patches of points, the tokens themselves are not guaranteed to have a particular order. This unorder nature of the point cloud makes the context and target selection challenging in joint-embedding predictive architecture, especially if you aim to mask spatially contiguous blocks of patches similar to I-JEPA[23].

1.3 Contributions

To overcome the issue of lengthy pre-training with previous methods and the challenge specific to point cloud data, we will introduce Point-JEPA: a joint-embedding architecture specific to point cloud objects. In this study, our main contributions are as follows.

- We consider a joint-embedding predictive architecture which operates on point tokens. Our method efficiently learns a strong representation from point cloud data without reconstruction in the input space or additional modality.
- We introduce the sequencer that is applied after the tokenization of the point cloud object. This enables the efficient selection of spatially contiguous tokens for context and targets.
- We will perform a thorough analysis of the learned representations and the effect of some of the important hyperparameters of the Point-JEPA work, allowing possible future development in this direction.

1.4 Organization

This thesis will be organized as follows.

- **Introduction:** This chapter introduces and defines the problem we aim to solve.
- **Background:** This chapter provides the reader with a basic understanding of self-supervised learning, providing some context for our work.
- **Related Work:** This chapter reviews some of the studies that have high relevance to our work. Some of the building blocks, ideas, and interpretations may come from studies in this section.
- **Point-JEPA:** This chapter covers our work. It mentions the use of a standard transformer on the point cloud object, the high-level pre-training method of Point-JEPA, and details the implementation.
- **Experimental Results:** This chapter contains results of the learned representation against the standard protocol as well as the ablation study analyzing the components of Point-JEPA.
- **Conclusion:** This chapter provides the conclusion of our work as well as a discussion of possible future studies.

Chapter 2

Background

In this chapter, we will cover some of the fundamental concepts of self-supervised learning to reach an audience with a broader background. It is recommended that those who are familiar with self-supervised learning proceed to Chapter 3.

2.1 Self-Supervised Learning

Self-supervised learning is a branch of deep learning algorithms that do not require labeled data to train the models. It processes the input data and attempts to create internal objectives within the algorithm in order to learn underlying features and relationships between components and views of the data [24]

It is often compared to supervised learning algorithms requiring labeled data to minimize the loss function between the true and the predicted label. Formally the objective of the supervised learning can be expressed as in Equation 2.1 [25].

$$\underset{\theta}{\operatorname{argmin}} \left[\sum_{x \in X} L(\hat{f}(x;\theta), f(x)) \right] \tag{2.1}$$

In Equation 2.1, $\hat{f}$ represents the model being trained, θ denotes the set of parameters

that defines the model, and X denotes the set of input features. The term L is the loss function measuring the disparity between predicted and true labels. The function f refers to the true underlying function or the ground truth that the model $\hat{f}$ attempts to approximate through optimization, which in the context of supervised learning, corresponds to the labels.

Although supervised learning has experimentally been shown to perform well in different domains of deep learning such as image recognition, natural language processing, and speech recognition [26, 27, 28], it has a significant drawback: it relies heavily on a large amount of labeled data. The process of creating labeled data is not only time-consuming but also expensive. Self-supervised learning addresses this issue by utilizing a vast amount of available data for deep learning models to be pretrained and fine-tuned with a smaller amount of data subsequently.

To learn from unlabeled data, self-supervised learning algorithms optimize tasks called pretext tasks. The goal of a pretext task is to minimize the loss between features extracted by deep learning models and signals created by the data itself. This signal creation is called self-supervision; therefore, the learning process is called self-supervised learning. Below, we will explore some of the most common types of self-supervised learning methods to give the audience a comprehensive overview.

2.2 Context-Based Learning

Some earlier self-supervised learning approaches can be categorized as context-based learning, where the model tries to predict the predefined context.

One of the widely known approaches from this category is the task of predicting the rotation angle of an image as a pretext task [29]. In this method, the input image is subjected to four discrete predefined rotational transformations. The deep learning model is trained using these modified images, attempting to determine the rotation of the angle applied to the original image. This approach is based on the principle that the deep learning model must identify the location and the appearance of the objects and recognize the relationships between different parts of the image in

order to effectively predict the rotation. Formally, let us define G as the set of K geometric transformation operators, expressed as $G = g(.|y)_{y=1}^{K}$. In this set, $g(.|y)$ is the operator that applies a geometric transformation to an input image X, with the transformation, in this case rotation, being defined by y. This process results in a transformed image, denoted as $X^y = g(X|y)$, where X^y represents the image after applying the transformation corresponding to y. Then the forward pass of the deep learning model $F(.)$ along with X^{y^*} can be expressed as in Equation 2.2.

$$F(X^{y^*}|\theta) = F^y(X^{y^*}|\theta)_{y=1}^{K} \tag{2.2}$$

Here, $F^y(X^{y^*}|\theta)$ represents the model's predicted probability for the input image X^{y^*}, which has undergone a geometric transformation labeled as y. The parameter θ refers to the set of parameters within the model $F(.)$, and the term y^* indicates the specific transformation applied to the input image, serving the self-supervision purpose that is concealed from the model $F(.)$. Then the θ that the model should optimized for can be expressed as

$$\underset{\theta}{\operatorname{argmin}} \left[\frac{1}{N} \sum_{i=i}^{N} loss(X_i, \theta) \right] \tag{2.3}$$

where we have training set $\{X_i\}_{i=0}^{N}$. Because this task can be thought of as a classification task, a cross-entropy loss function can be utilized as in Equation 2.4.

$$loss(X_i, \theta) = -\frac{1}{K} \sum_{y=1}^{K} log(F^y(g(X_i|y)|\theta)) \tag{2.4}$$

By providing the model with the essential context of the image, this method successfully trains the model to understand both local details and the overall global structure of the image with no labels.

Other pretext tasks in this category include colourization and solving jigsaw puzzle [30, 31, 32, 33, 34]. In the colourization pretext task, an image is provided to the model in single-channel format, represented explicitly by a lightness channel. The objective of this pretext task is to minimize the difference between the "ab" colour

channels of the input image in CIE Lab colour space and the model's output. Dividing the colour space into bins makes this task similar to a classification task, where the model aims to predict the correct colour bin for each part of an image. On the other hand, jigsaw pretext task involves a few steps. It first divides the input image into no-overlapping equally spaced regions. It then shuffles these regions according to the pre-defined set of permutations. The model processes these permuted image regions and aims to predict the correct order of them; thus, it is called a jigsaw puzzle. The model learns the representation by understanding the relationships between different parts of the image.

The success of self-supervised learning in this area largely depends on the hand-crafted pretext tasks. While many of these tasks are effective, achieving successful training requires the use of sophisticated, carefully designed transformations of input data to guide the learning process.

2.3 Contrastive Learning

Frameworks such as MoCo [16, 35, 36] and SimCLR [11, 37] can be categorized as contrastive learning. This approach employs the concept of positive and negative samples. Methods in this category aim to generate embeddings that are close for positive pairs and apart for negative pairs. Positive pairs are pairs of input data that are considered similar or related to each other. They are usually created from different views or augmentations of the same instance. On the other hand, negative pairs are created from input data that are considered dissimilar or unrelated. They are often created from instances or categories different from positive instances. The variations in contrastive self-supervised learning methods mainly come from the differences in the strategies for selecting these positive and negative pairs.

Momentum Contrast, commonly referred to as MoCo [16] considers self-supervised learning as a dictionary look-up task. MoCo uses two distinctive encoders: f_q for generating a query q from a given image, and f_k for creating a set of keys $\{k_0, k_1, k_2, ...\}$. The objective in this pretext task is to find a key k_+ that closely matches the query

q, while distinguishing q from negative keys $\{k_0, k_1, k_2, ...\} - \{k_+\}$. More specifically, the creation of keys and queries are completed with batches of data. Queries and keys that are encoded from the current batches are considered positive samples and keys from previous batches are considered negative samples. In order to achieve this objective, it utilizes the InfoNCE loss function [38] as in Equation 2.5.

$$-\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)} \tag{2.5}$$

Here, K is the cardinality of the negative set $\{k_0, k_1, k_2, ...\} - \{k_+\}$, and τ is a temperature hyper-parameter [39]. In order to utilize instances from the previous batch as negative examples, MoCo uses a momentum update to update the weights of f_k partially from f_q; that is,

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q \tag{2.6}$$

where θ_k represents the parameter of f_k and θ_q represents the parameter of f_q, with $m \in [0, 1)$. Here, θ_q is updated with back-propagation and θ_k is updated with momentum update. The momentum update makes the update of θ_k smooth, which reduces the difference between the keys from the previous batch and the current batch.

A simple framework for contrastive learning [11], or SimCLR for short, creates negative and positive pairs from the augmentations of the training data. For each training instance, it creates 2 augmented instances creating $2N$ data points in total, where N is the number of instances in a batch. As shown in Figure 2.1, each pair of data points is fed to the encoder $f(.)$ followed by the projector $g(.)$. The loss for the positive pairs (i, j) is calculated by Equation 2.7.

$$\ell_{i,j} = -\log \frac{\exp(sim\,(z_i, z_j))/\tau}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(sim\,(z_i, z_j)\,/\tau)} \tag{2.7}$$

where $sim(u, v) = u^\top v/\|u\|\|v\|$ denotes the cosine similarity between u and v, $\mathbb{1}_{[k \neq i]} \in \{0, 1\}$ denotes the indicator function which returns 1 if and only if $k \neq i$, and τ denotes the temperature parameter to determine how sharp the loss function is. Unlike MoCo [16], which employs the instances from the previous batches as negative samples,

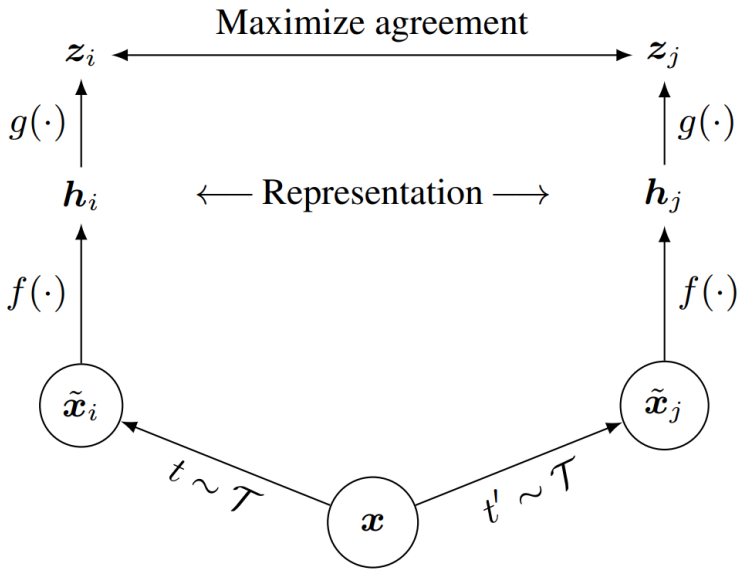

Figure 2.1: SimCLR Architecture. Two different augmentations t and t' that are selected from a set of augmentation $\mathcal{T}$ and applied to the instance x. Then the encoder $f(.)$ and the projection head $g(.)$ are applied to two augmented instances individually. The loss is calculated such that there would be a maximum agreement between the representations of two augmented input instances. Source: [11]

SimCLR utilizes the instances from the current batch as negative samples.

2.4 Self-distillation Learning

Knowledge distillation, which precedes and forms the basis for self-distillation, was first introduced as a means to transfer the representation of a larger model "teacher" to a smaller model "student" [40, 41]. Self-distillation can be thought of as a specific application of knowledge distillation where the teacher and student are identical and the parameters are updated in an end-to-end fashion.

One of the works that is attributed to the recent success of self-distillation learning is Bootstrap Your Own Latent (BYOL) [13]. BYOL utilizes two identical encoders, as shown in Figure 2.2, one called online encoder and another called target encoder, each

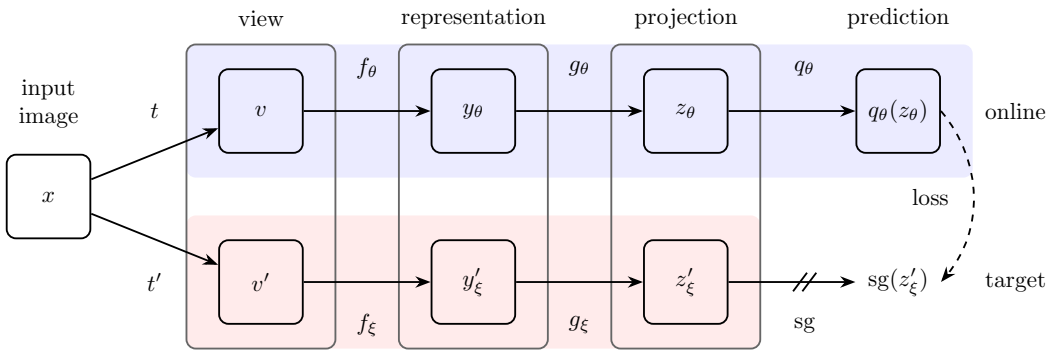

Figure 2.2: Pretraining in BYOL. Here τ and τ' the transformation applied to the input image, f represents the model that we aim to pretrain, and g represents the projector. The predictor q_θ predicts the output of the target encoder given the output of the online encoder. The weights of the online encoder θ is updated with gradient-based optimization and the weights for the target encoder ξ is updated with running average of θ. Source: [13]

containing different parameter values. Each encoder receives two different augmented images, and their objective is to maximize the agreement between two encoders. Similar to MoCo [16], the online encoder is updated with gradient-based optimization with regards to the loss, while the target encoder is only updated by the exponential moving average of the online encoder. As it turns out in the work of Simple Siamese network (SimSiam) [12], we can throw away many components of BYOL by sacrificing accuracy slightly. Specifically, SimSiam demonstrates that the important element preventing BYOL's architecture from collapsing is the stop-gradient operation. Other features such as the exponential moving average of the weights in the target encoder, the use of large batches, and the symmetrization of loss functions are of secondary importance.

Effectively, self-distillation learning effectively eliminates the need for negative samples, while still being able to learn meaningful representations.

2.5 Generative Learning

Generative models including [42, 15, 43, 14] often utilize encoder-decoder architecture with masking of the input instances. This approach has become prevalent in the image domain where masked image modeling has shown promising results. In masked image modeling, the input images processed by the model include masked or corrupted regions. The model's objective is to accurately reconstruct these regions. The concept was initially introduced by BEiT [42], which employs random masking for certain patches of the image. More specifically, BEiT initially trains "image tokenizer" by auto-encoder-like reconstruction to obtain fixed-size vocabulary, i.e., discrete tokens that can represent the image patches. The pretraining is then achieved by utilizing the learned image tokenizer and its fixed-size vocabulary of image patch representation. Certain image patches are masked randomly and substituted with a mask embedding. Following this, vision Transformer [44] processes these patches and aims to predict the corresponding visual tokens.

Similar to BEiT [42], MAE [15] employs recovering of the missing part of the image as a pertaining method. MAE, instead of predicting certain tokens of corresponding patches, simply aims to reconstruct the input image with corrupted masked input through encoder-decoder architecture. This eliminates the need for training tokenizer prior to the pertaining and allows end-to-end learning.

Mathematically, the masked image modeling can be abstracted as Equation 2.8 [24].

$$L\left(D\left(E\left(T_1\left(I\right)\right)\right), T_2\left(I\right)\right) \tag{2.8}$$

Here, $E(.)$ is the encoder and $D(.)$ is the decoder. $T_1(.)$ represents the transformation applied to the input image I, which is often a procedure designed to mask certain regions of the input images. Similarly, $T_2(.)$ represents the transformation applied to the input image I as a target representation in pre-taining. The difference between mask image modelling methods often comes from the variation of transformation T, that is applied to input images.

Generative models are a data-filling approach [24]. This means they can perform

sufficiently with an extremely small number of training examples. However, this also means that increasing the size of the training examples does not usually improve performance in generative self-supervised pretraining.

Chapter 3

Related Work

As previously mentioned, self-supervised learning has achieved remarkable results in domains such as image and natural language processing. This success has inspired its application in the point cloud domain. Additionally, the use of Transformer [45], known for its effectiveness in other domains, has performed well on point cloud data. In this section, we review various studies in the realm of self-supervised learning in the point cloud domain as well as variants of transformers in the same domain. We will also investigate the drawback of the current stat-of-the-art self-supervised learning methods and introduce the concept of joint-embedding predictive architecture.

3.1 Self-Supervised Learning on Point Clouds

3.1.1 Contrastive Learning in Point Cloud Domain

As mentioned in the previous chapter, contrastive learning has been prevalent and successful in the image domain. Similar to the application in the image domain, the objective of contrastive learning in the point cloud domain is to generate embeddings such that they are close for positive pairs and far for negative pairs. This is achieved via the different methods that create positive and negative pairs tailored towards point cloud data.

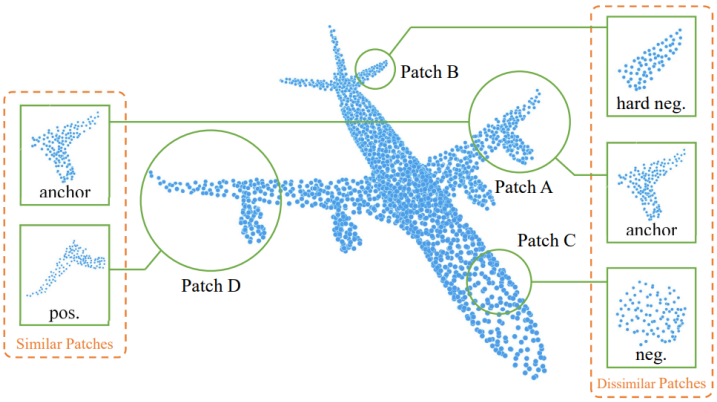

Figure 3.1: Illustration of negative and positive pairs from an object. Here, for a given anchor patch, the positive sample is drawn from similar patches and negative samples are drawn from dissimilar patches according to the learned representation of the similarity learning step. The hard negative sample is the patch from dissimilar patches that have similar features to the anchor patch. source: [46]

One of the ways to create positive and negative pairs is by sampling parts of the point cloud objects, called patches. In [47], self-supervised learning takes place with two distinct tasks. The first task involves determining whether two point cloud segments belong to the same instance of the training data. Here, positive pairs are drawn from segments of the same instance and the negative pairs are drawn from segments of different instances. Then the learned representation from this first step is used to cluster the data points using Kmeans++ [48]. The cluster IDs are used as pseudo-labels to train another network as a classification task. Because this is a separate task from the contrastive pretext task, the architecture of the model does not depend on the model used in the initial step, and therefore it is more flexible.

Similarly, in [46], positive and negative pairs are drawn from different parts of the same object, except that both negative and positive samples originate from the same object. In order to separate patches into positive and negative pairs, or similar and dissimilar groups, the method employs a similarity learning model. For each selected patch, referred to as the anchor patch, a similar patch is created by rotating the anchor, while a dissimilar patch is selected from a different part of the object. Through the

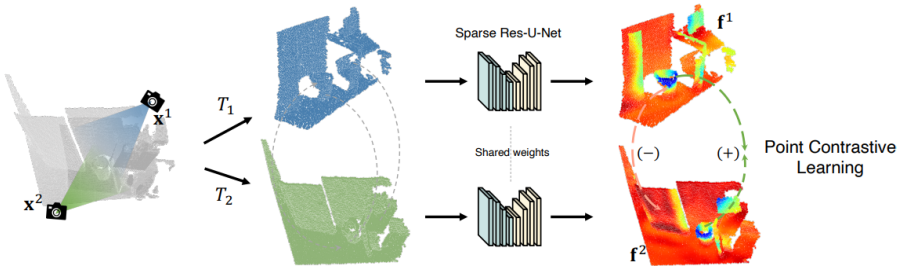

Figure 3.2: Illustration of pretext task in PointContrast. source: [49]

process of similarity learning, the model is trained to identify which pairs of patches in an object are similar and which are dissimilar as shown in Figure 3.1. Then, contrastive learning is applied using these pairs. Additionally, the concept of a hard negative sample is applied, where the chosen negative sample closely resembles the positive samples in the representation space. This allows the model to learn more discriminative features.

Building upon the concept of using contrastive learning on patches of an object, several studies [49, 50, 51, 52, 53] have extended this approach to larger, room-scale point cloud datasets. Similar to object-based contrastive learning, these methods generate multiple views for an instance of training data to form positive and negative pairs in various ways, along with special architecture better suited for point cloud scene understanding.

For example, PointContrast [49] utilizes U-Net-based architecture to extract dense features within scene-level point cloud data. Here, as shown in Figure 3.2, two views of a training instance are generated. It then considers pairs of points across these two views that correspond to each other as positive pairs, while points that do not match are considered negative pairs. To enhance the complexity of the pretext task and encourage the network to learn an invariant embedding with random geometric augmentation, a rigid transformation including rotation, translation, and scaling is applied to both views. Then the objective of the pretext is to reduce the distance between points that match and to increase the distance between points that do not

match.

Although contrastive learning has been shown to provide good performance when fine-tuned, it generally requires careful choice of negative samples and data augmentation to learn a transferable representation for downstream tasks.

3.1.2 Generative Learning in Point Cloud Domain

There have been recent developments in self-supervised learning on point cloud focused on object-level data by employing reconstruction tasks on masked patches along with Vision Transformer-like [44] models.

Point-Bert [54], inspired by Bert [55] in natural language processing, introduced generative pretraining in the point cloud domain with the intricate mechanism of token generation. Initially, the raw point cloud data is transformed into a set of tokens, which captures the geometric feature of the point cloud. These original tokens are then processed to create a discrete fixed-sized vocabulary of discrete tokens, called discrete point tokens, by discrete variational autoencoder. During the training phase, a subset of original tokens is selected and masked, simulating the missing data within the point cloud. The Transformer model [45] then processes the visible patches and attempts to reconstruct the corresponding discrete point tokens of the masked tokens. The limitation of PointBert, however, is the relatively complicated pretraining steps that heavily rely on data augmentation and the early leakage of location information.

Inspired by masked autoencoder [15], Point-MAE [20] was introduced in order to address the aforementioned issues. Point-MAE, similar to PointBert [54], employs Transformer [45] as backbone architecture. However, Point-MAE differentiates itself with a high ratio of random token masking and an encoder-decoder architecture to achieve its objective. Much like many masking strategies for ViT [44], Point-MAE removes tokens from the encoder input in order to "mask" some tokens and introduces arbitrary mask tokens created with shared learnable parameters in the decoder's input to reconstruct masked patches. This technique improves computational efficiency and

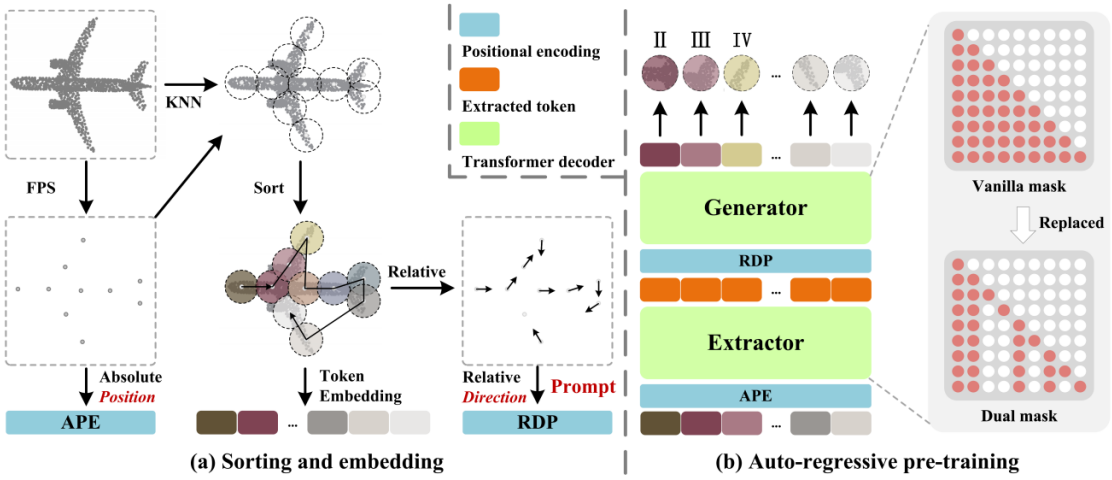

Figure 3.3: (a) The Tokenizer divides the point cloud into a minimum overlapping set of points (patches) and the center points of them are sorted using geometric ordering, followed by the embedding process.
(b) Dual masking not only masks out a portion of the patches that precede each target token in the sequence but also extends to masking out patches that are already visible to the model, further complicating the prediction task. The extractor and generator are composed of Transformer blocks mapping the input to an intermediate representation. Then the relative direction prompts RDP to provide the direction relative to the subsequent patches to the generator. source: [19]

prevents the early leakage of positional information. A projection head is added to align decoded patches back to their original dimensions, aiming to minimize the discrepancy between the reconstructed and original points, which simplifies the approach while retaining the essence of the original MAE concept.

A slightly different approach taken by PointGPT [19] represents a novel direction in generative learning, following the success of the generative pretraining Transformer [14] in the natural language processing domain. In short, PointGPT adapts the auto-regressive pretraining to point cloud objects. For point cloud objects, this is not a trivial application because of the arbitrary order of point cloud data. To address this challenge of unordered point cloud data, PointGPT introduces a method of geometric ordering. What's more, the point clouds, similar to images, are known to contain redundant data. To learn meaningful representation despite this redundancy, it utilizes

a dual masking strategy and an extractor-generator architecture as shown in Figure 3.3. This design allows the Transformer to predict point patches in an auto-regressive manner without the need for dedicated specifications, eliminating the risk of early leakage of positional information.

3.1.3 Distillation in Point Cloud

Although much less studied in point cloud compared to the other self-supervised learning families, distillation has been shown to produce meaningful representation in a few studies [21, 18].

In the work of DCGLR [21], the idea of knowledge distillation is applied to point cloud objects inspired by DINO [17]. The architecture contains two identical networks that have different parameter values. One of the networks, referred to as the teacher network, receives a set of point clouds called global point cloud set, which is created from cropping the original point cloud. On the other hand, the other network, referred to as the student network, receives a cropped point cloud set called local point cloud set as well as the global point cloud set. Here, the local point cloud set is created from cropping in the same manner as the global point cloud set, except that the ratio of the cropping is larger than that of the global point cloud set. Then the cross-entropy loss is calculated for the output of two networks. More specifically, we have the global set P_g^i and the local point cloud set P_l^j generated from a point cloud $P \in R^{N\times 3}$ such that

$$P_g^i = crop(P, rand(r_{g1}, r_{g2})), \quad i = 1 \dots I \tag{3.1}$$

$$P_l^j = crop(P, rand(r_{l1}, r_{l2})), \quad j = 1 \dots J \tag{3.2}$$

where $crop(P, a)$ denotes the cropping of a point cloud P with ratio of a, and $rand(min, max)$ denotes a random number generator between the values min and max. I and J denote the number of point clouds for the global and local point cloud sets. Then, P_g^i and P_l^i are processed by student and teacher network as follows.

$$o_{tg}^i = projector(f_t(\{P_g^i\})), \quad i = 1 \dots I \tag{3.3}$$

$$o^i_{sg}, o^j_{sl} = projector(f_s(\{P^{i'}_g, P^{j'}_l\})), \quad i = 1 \dots I, j = 1 \dots J \tag{3.4}$$

Here, we have $o^i_{tg}, o^i_{sg}, o^j_{sl} \in R^K$ and the projector $projector(.)$ that projects the output of student and teacher encoder to K dimensional space. Then balanced centring and sharpening on teacher output o^i_{tg} similar to DINO to get $\hat{o}^i_{tg}$.

Then the cross entropy for each o^i_{sg} and o^j_{sl} are defined as Equation 3.5 and 3.6.

$$loss_g = \min_{f_s} \frac{1}{I(I-1)} \sum_{i=1}^{I} \sum_{\substack{j=1 \\ j \neq i}}^{I} -\hat{o}^i_g \cdot log(o^j_g) \tag{3.5}$$

$$loss_l = \min_{f_s} \frac{1}{I \cdot J} \sum_{i=1}^{I} \sum_{j=1}^{J} -\hat{o}^i_g \cdot log(o^j_l) H(\hat{o}^i_g, o^j_l) \tag{3.6}$$

and the final loss takes the weighted sum of global and local loss.

$$loss = w_g * loss_g + w_l * loss_l \tag{3.7}$$

The parameters of the student network are updated with backpropagation while the ones of teachers are updated using the exponential moving average of the student network. Because the global point cloud set provides a better representation of the objects than the local point cloud set, this architecture allows the stable update of the student's parameter. At the same time, as the student network is provided with both local and global point cloud sets, it learns the invariance of the global shape as well as the relation of the local shape to the global shape.

Similar architecture can be seen in the work of Point2Vec [18]. Inspired by data2vec [56], it utilizes two identical networks, again called student and teacher. The teacher network receives all of the available patches and the student receives only part of them. Here, similar to the work in generative self-supervised learning with Transformer [54, 19, 20], masking is applied to the tokens generated from a group of points. Unlike the original data2vec work, in which the masked embeddings are replaced with a special

learned mask embedding, Point2Vec hides and does not feed the masked embedding to the student network to avoid leakage of the positional information. Instead, Point2Vec introduces a shallow Transformer decoder that takes the output of the student and the mask embedding to predict the target embedding produced by the teacher. With positional information masked to the student as well as the student learning the contextualized embedding from the partial-view input, it learns meaningful and robust representation.

3.2 Transformer

Transformer [45] is a sequence-to-sequence model, which has the ability to learn long-range dependency, and it has achieved state-of-the-art results in the natural language processing domain [55, 57, 58, 14]. The successful application of this architecture to image classification [44] led the interest of researchers in the different variants of the architecture in the image domain [59, 60, 61, 62, 63, 64, 65].

Transformer [45] is originally an encoder-deconder model, which contains the same number of blocks. Closely related to our work, ViT [44] specifically only consists of encoder blocks. Each of them consists of a multi-head attention and MLP module. Residual connection is added after and layer norm is added before each module [66, 67] as shown in Figure 3.4.

3.2.1 Attention

Transformer [45] adopts an attention mechanism called "Scaled Dot-Product Attention" and this module computes the attention as follows (Equation 3.8).

$$Attention(Q, K, V) = softmax(\frac{QT^T}{\sqrt{d_k}})V \tag{3.8}$$

given set of queries, keys and values in matrix as Q, K, V. Here d_k is the dimension of each key and query. The scaling factor $\frac{1}{\sqrt{d_k}}$ is applied to avoid large values for dot product QT^T, preventing vanishing gradient in the softmax function.

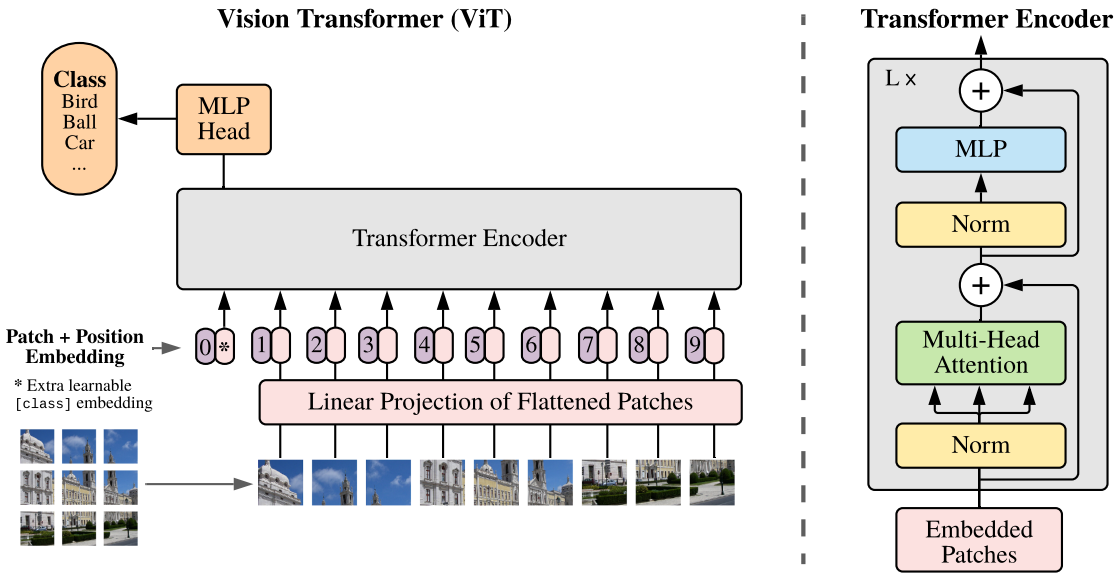

Figure 3.4: ViT: Non-overlapping patches form an image are projected to have a specific dimension and they are fed to Transformer along with positional embedding. Each encoder block contains MSA and MLP along with layer norm and residual connection. Source: [44]

3.2.2 Multi-Head Attention

Instead of simply applying an attention function with keys, queries, and values with dimension d_{model}, Transformer applies "multi-head attention", which divides keys, queries, and values into a subset of lower dimensional features and computes attention function on it [45]. Specifically, key, query and values are linearly projected to d_k, d_q, d_v dimensions with H learned projections, then attention is calculated as in equation 3.8 for each projected key, query, and value. The resulting outputs are concatenated to have the original d_{model} dimension.

3.2.3 Transformer for Point Cloud Objects

Inspired by the success of other domains, Transformer architecture [45] has been applied to the point cloud domain. One straightforward application of it is to calculate attention on the entire point cloud object globally [68]. More specifically, the point cloud coordinates are fed to the encoder, which maps the coordinates from 3-

dimensional space to higher-dimensional feature space. Then four identical standard attention modules are applied sequentially and the output of each attention module is concatenated similarly to DGCNN [69]. Then the max and average pooling of the concatenated features are calculated to produce a global feature to be used for specific tasks.

Alternatively, we can consider feature aggregation in the local region instead of calculating attention on the entire point cloud object [70, 71, 72]. For example, PointTransformer employs hierarchical architecture similar to PointNet++ [10, 73]. Instead of using a shared MLP module, PointTransformer processes local patches with Transformer blocks. Its Transformer blocks operate on progressively down-sampled point sets with each block processing neighbourhoods of sample points.

PointBert [54] introduces the use of a standard Transformer in a similar manner as ViT [44]. Many self-supervised learning methods in the point cloud domain follow the similar or identical use of standard Transformer [20, 19, 18]. As shown in Figure 3.3, the tokenizer encodes an equally subdivided set of points into a high-dimensional token. Transformer [45] then computes the attention of these produced tokens. This successfully allows the separation of local and global feature extraction. The details of the architecture will be discussed in the following chapter.

3.3 Joint-Embedding Predictive Architecture

Joint-Embedding Predictive Architecture (JEPA) [22] is a self-supervised learning architecture that learns representation from the prediction in the embedding space. In simpler terms, JEPA learns by predicting one set of encoded signal y based on another set of encoded signal x, along with an additional conditional variable z that controls the prediction. This process is performed by a specific component known as a predictor network. Encoders initially process both the target and context signals to represent them in embedding space. Conceptually JEPA has a large similarity to generative models, which are designed to reconstruct masked part of the input. However, unlike generative models that operate directly on the input space, JEPA

makes predictions in the embedding space. What this allows is the elimination of the details of the input that is not necessary for learning meaningful representations. As a result, the model can abstract and represent the data more efficiently.

Closely related to our work, the specific application of the architecture in the image domain can be seen in I-JEPA [23]. In this work, the context signal is created by encoding a block of patches while the target signals are created from sampling blocks from encoded embedding vectors. Experimentally, it is shown that I-JEPA converges much faster than methods which rely on pixel-level reconstruction and learn highly semantic representation. In order to address the limitation mentioned below, we aim to apply JEPA on point cloud objects.

3.4 Limitations of the Current State-of-the-Art

While the Current state-of-the-art pretraining methods such as PointGTP [19] and Point2Vec [18] learn a meaningful representation and produce prominent results when finetuned, it takes a significant amount of time for them to learn meaningful representations. For example, it was found that it takes 16.9 hours for PointGPT and 12.1 hours for PointGPT to pretrain on NVIDIA RTX A5500. This time-consuming nature of pretraining would be amplified when the size of dataset grows, making the pretraining difficult. In our work, we aim to minimize the pretraining time with joint-embedding predictive architecture as it has previously been seen in the image domain that it reduces the pretraining time while producing state-of-the-art results [23].

Chapter 4

Point-JEPA

We will introduce a joint-embedding predictive architecture for pretraining in the point cloud domain inspired by I-JEPA [23] to overcome the issue with current state-of-the-art methods. As shown in Figure 4.1, targets are created from encoded embedding, while the context is created from encoding a limited number of tokens. The encoded context embedding and target positional embedding are fed to the predictor, aiming to maximize the agreement between the target and its prediction. This asymmetric architecture mitigates the concern of representation collapse, that is, constant representation for outputs regardless of the input, and the sparsity of the target blocks allows efficient processing.

4.1 Backbone Architecture

Similar to previous work [19, 18, 20, 54], we adopt standard Transformer architecture [44, 45] in point cloud processing. As shown in Figure 4.1, a given point cloud object in the dataset is divided into regular point patches by the Farthest Point Sampling [74] and K-Nearest Neighborhood algorithm [75]. Then point patches are encoded by a deep-learning model that directly processes the points, and center points are encoded with MLP to produce positional embedding. Our method selects PointNet-like [9] architecture for encoding point patches in order to be consistent with previous work

and fairly evaluate our retraining method.

We formalize the tokenization of a point cloud object using a matrix-based notation, similar to Point-MAE [20]. The process of creating patches given a point cloud object $X_i \in \mathbb{R}^{n\times 3}$ with n being the number of points representing the object is as follows.

$$C_i = FPS(X_i, t) \tag{4.1}$$

$$P_i = KNN(X_i, C_i, k) \tag{4.2}$$

where $FPS(\cdot,\cdot)$ denotes the farthest point sampling with a selected number of sampled points, t denotes the number of patches we would like to create, and $KNN(\cdot,\cdot,\cdot)$ denotes the K-Nearest Neighbour algorithm. k denotes the number of neighbours we would like to select in each patch. Here, we have $C_i \in \mathbb{R}^{t\times 3}$ and $P_i \in \mathbb{R}^{t\times k\times 3}$. Then, the patches P_i are normalized by subtracting the centre coordinates of the corresponding patches C_i. That is for all $i \in \{1, 2, \dots t\}$, $j \in \{1, 2, \dots k\}$, and $l \in \{1, 2, 3\}$

$$G^{-}_{ijl} = G_{ijl} - C_{il} \tag{4.3}$$

where G_{ijl} denotes the point in lth dimension of jth point in ith patch and C_{ik} denotes kth dimension of centre point of ith patch. This allows the separation of local structural information and the positional information for the Transformer encoder. G^{-} is encoded with PointNet [9] like architecture to produce tokens $T \in \mathbb{R}^{t\times d}$ and C is encoded with MLP to produce the positional embedding $\in \mathbb{R}^{t\times d}$ where d denotes the dimension of each token.

$$T = PointNet(G^{-}) \tag{4.4}$$

$$E_p = MLP(C) \tag{4.5}$$

Thanks to the farthest point sampling, we have a fixed number of tokens t. This enables the processing of tokens T along with positional embedding E_p similar to ViT [44]. Differently from the ViT, the positional embedding is added before each transformed block to highlight the special importance of positional information in

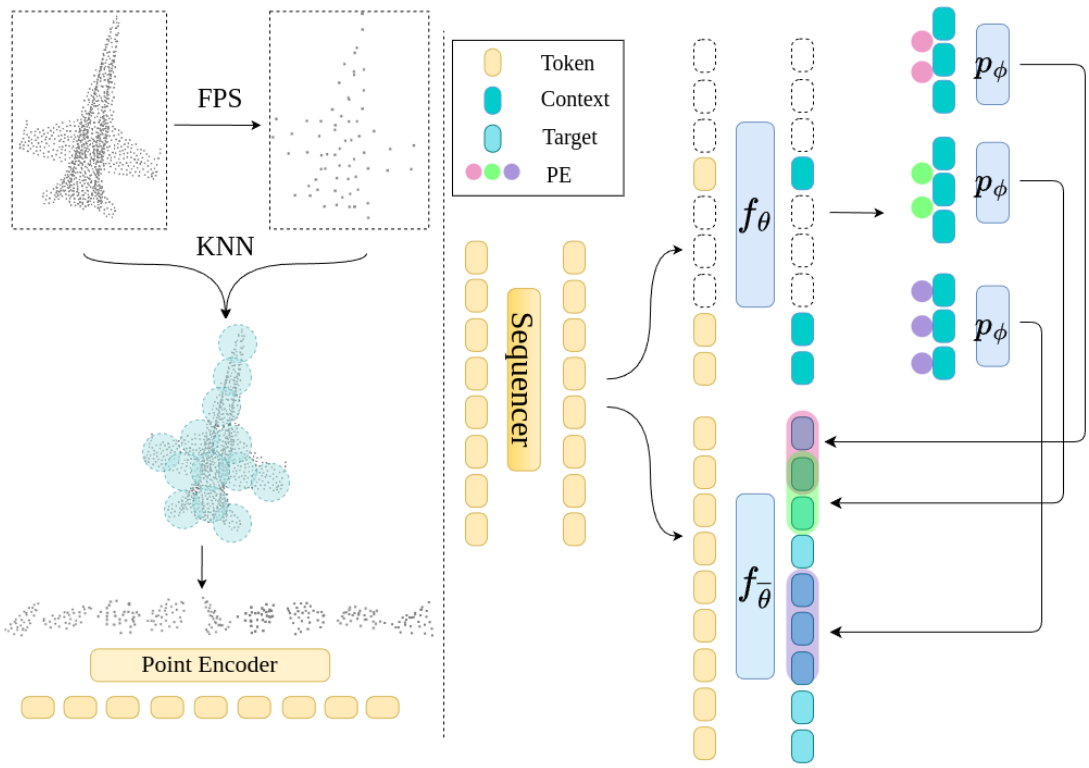

Figure 4.1: Illustraion of Point-JEPA Architecture. The tokenization of point cloud patches is shown on the left side and the joint-embedding architecture is shown on the right side.

point cloud objects [18].

4.2 Target, Context, and Sequencer

Targets in Point-JEPA can be considered patch-level representations of the point cloud object that the predictor aims to predict. As depicted in Figure 4.1, the target-encoder $f_{\bar{\theta}}$ initially encodes the tokens conventionally, as mentioned in the previous section. Then, from the encoded embedding, we randomly select M possibly overlapping blocks. Each target block is formed from encoded embedding vectors with spatially contiguous corresponding center points. Here, we denote the encoded embedding vectors as $y = \{y_1, y_2, \ldots y_n\}$, where y_k is the representation associated with

the k^{th} centre point. We then define $y(i) = \{y_i\}_{j \in B_i}$ as the i^{th} target representation block, where B_i denotes the set of mask indices for the i^{th} block. Here, the masking is applied to the output of the target encoder instead of the input T. This ensures a high semantic level for the target representations [23].

Context, on the other hand, is the representation of the point cloud object that gets passed to the predictor as a clue to predict the representation of targets. To construct the context representation, we first select a subset of tokens $\hat{T} \subseteq T$ that are spatially contiguous. These selected tokens are then fed to the context-encoder f_θ to generate a context block $x = \{x_j\}_{j \in B_x}$. To prevent trivial learning, we ensure that the indices of tokens chosen for the context differ from those for the targets.

This pretraining process poses unique challenges when applied to point cloud objects. The difficulty arises because the point cloud data does not have an ordered representation like an image. In the case of images, the relative location of a patch within the image can be determined from its index, making the selection of spatially contiguous regions straightforward. However, with point cloud data, even if the indices of tokens are sequential in the index, they might not be spatially adjacent. Moreover, our method involves selecting spatially contiguous M blocks of encoded embedding as the target for training while ensuring that the context block does not include the tokens corresponding to these elements. This complicates the selection process for context tokens because the pool of available tokens varies between instances in a batch, which implies an inconsistent number of tokens available between elements in a batch for context selection.

In order to overcome these challenges, we introduce a sequencer that is applied after the tokenization of the points. This sequencer orders tokens based on their associated centre points. The process begins with a chosen centre point and its associated token. In each subsequent step, the centre point closest to the one previously chosen and its associated token are selected. This is iterated until the sequencer visits all of the centre points. The resulting arrangement of tokens is in a sequence where contiguous elements are also spatially contiguous in most cases.

With the ordered tokens, we can easily select M blocks of targets from embedding

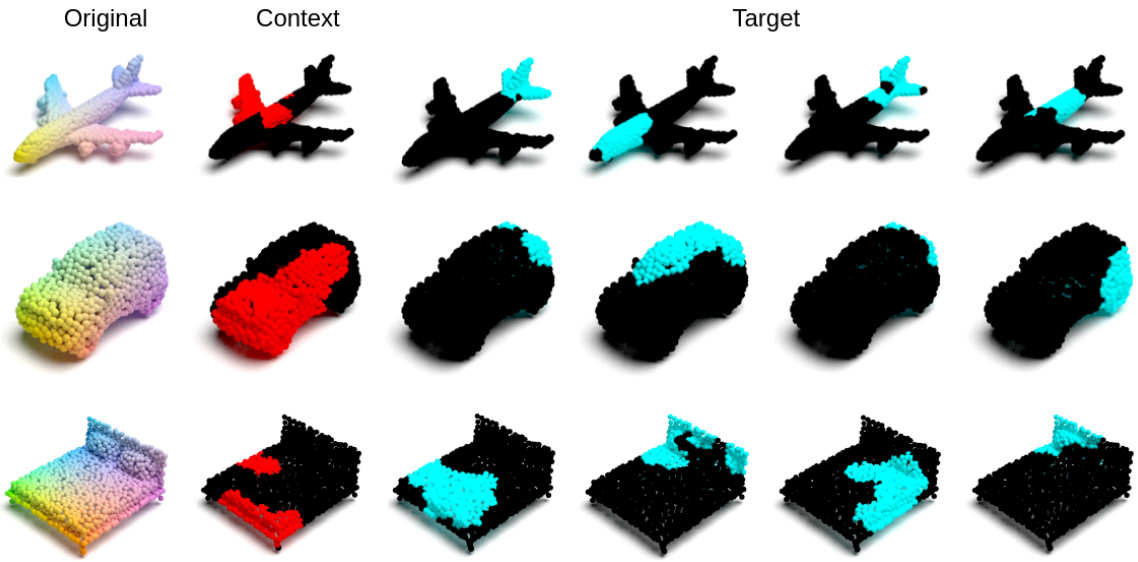

Figure 4.2: Visualization of targets and context masking strategy for objects from Shapenet55 [76] dataset. We visualize the corresponding grouped points of context and target embedding vectors. Here, we use (0.15, 0.2) for the target selection ratio and (0.4, 0.75) token selection ratio for context.

vectors with spatially contiguous corresponding centre points. We can then exclude the indices of these selected encoded embedding vectors when choosing context tokens. This ensures that the number of indices available for selecting context blocks is consistent across all elements in a batch, greatly simplifying the implementation. Additionally, this allows the sharing computation of spatial proximity between context and target selection, which improves overall computational efficiency. However, it is worth noting that selecting two adjacent token indices does not guarantee spatial contiguity; there might be a gap between them. While this is true, our experiment shows that our method is effective enough for learning representation with no labels.

4.3 Predictor

The task of predictor p_ϕ given targets y and context x is analogous to the task of supervised prediction. Given a context as input x along with a certain condition,

it aims to predict the target representations y. Here, the condition involves the mask tokens, which are created from shared learned parameters, as well as positional encoding, created from centre points associated with the targets. That is

$$\hat{y}(i) = \{\hat{y}_j\}_{j \in B_i} = p_\phi\left(x, \{m_j\}_{j \in B_i}\right) \tag{4.6}$$

where $p_\phi(\cdot, \cdot)$ denotes the predictor and $\{m_j\}_{j \in B_i}$ denotes the mask token created from shared learnable parameter and positional encoding created from centre points.

4.4 Loss Function

Because the predictor's task is to predict the representation produced by the target encoder, the loss can be defined to minimize the disagreement between the predictions and targets as follows.

$$\frac{1}{M} \sum_{i=1}^{M} D(\hat{y}(i), y(i)) = \frac{1}{M} \sum_{i=1}^{M} \sum_{j \in B_i} \mathcal{L}(\hat{y}_j, y_j) \tag{4.7}$$

Similar to Point2Vec [18], we utilize Smooth L1 loss to measure the dissimilarity between each corresponding element of the target and predicted embedding because of its ability to be less sensitive to the outliers [56]. That is,

$$\mathcal{L}(\hat{y}_j, y_j) = \begin{cases} \frac{1}{2}(\hat{y}_j - y_j)^2/\beta & \text{if } |\hat{y}_j - y_j| \leq \beta \\ (\hat{y}_j - y_j) - \frac{1}{2}\beta & \text{otherwise} \end{cases} \tag{4.8}$$

Here, β is a hyper-parameter determining the boundary between L1 and L2 loss.

4.5 Implementation Details

In this section, we discuss the implementation of Point-JEPA in detail.

4.5.1 Architecture

As previously mentioned, we utilize standard Transformer [45] architecture for context and target encoder as well as predictor. During pretraining, we set the number of centre points to 64 and the group size to 32. The point tokenization is applied to the input point cloud containing 1024 points. We set the depth of the Transformer in context and target encoder to 12 in context and target encoder with the embedding width of 384 and 6 heads. For the predictor, we use the narrower dimension of 192 following I-JEPA [23]. The depth of the predictor is set to 6, and the number of heads is set to 6.

4.5.2 Optimization

We utilize AdamW [77] optimizer with cosine learning decay [78]. Starting from learning rate of 10^{-5}, we increase it to 10^{-3} in the first 30 epochs and decay it to 10^{-6}. The batch size for pretraining is set to 512, and β for Smooth L1 loss is set to 2, similar to Point2Vec [18]. The target encoder and context encoder initially have identical parameters. The context encoder's parameters are updated via backpropagation, while the target encoders' parameters are updated using the exponential moving average of the context encoder parameters, that is $\overline{\theta} \leftarrow \tau\overline{\theta} + (1-\tau)\theta$ where $\tau \in [0,1]$ denotes the decay rate. We gradually increase the decay rate of the exponential moving average from 0.995 to 1.0 during pretraining.

4.5.3 Masking and Ordering

To determine the sequence of tokens, we utilize the iterative ordering of associated centre points, as previously mentioned. We chose the starting point in this sequence with the lowest sum of its coordinates. This method allows us to start the sequence from a point on the outer edge of the object rather than from a point within the object's interior. This consistency in selecting the initial point is experimentally shown to deliver a slightly better learned representation than taking the first available index.

For masking, we define a range of ratios with both upper and lower limits similar to I-JEPA [23]. To start with, we clarify that the term "block" refers to a sequence of tokens and their corresponding encoded embedding vectors that are contiguous. Because of the sequencing process applied before the target and context selection, most contiguous tokens and encoded embedding vectors are also spatially contiguous. For the target, we randomly select 4 blocks of tokens from within the 0.15 to 0.2 range. We then remove the corresponding tokens of encoded embedding vectors that have already been chosen as targets for further selection. Following this, we choose a block of tokens that is within the range of 0.4 to 0.75 out of available tokens that are not concealed. Because some of the tokens are not available for context selection, we note that context block usually consists of multiple sets of tokens that are spatially contiguous. The selection of targets is completed on a per-batch basis, and we track the indices of these targets to ensure that the corresponding tokens of these selected encoded embedding vectors are concealed in the context selection. The context is then selected using the available indices of tokens also on a per-batch basis.

Chapter 5

Experiment

5.1 Self-Supervised Pretraining

We pretrain our model on the training set of ShapeNet[76] following the previous studies utilizing the standard transformer architecture such as Point-MAE [20], Point-M2AE [79], PointGPT [19], and Point2Vec [18]. The dataset consists of 41952 3D point cloud instances created from synthetic 3D meshes from 55 categories. We pretrain Point-JEPA for 500 epochs and evaluate the learned representation on classification and part segmentation tasks. Specifically for classification tasks, we consider linear probing, end-to-end fine-tuning, and few-show learning. Our experiments are conducted on NVIDIA RTX A5500 and NVIDIA A100 SXM4. We note that our method only takes 7.5 hours on RTX A5500 for pretraining. This is less than half of what PointGPT [19] requires and about 60% of what Point2Vec [18] requires for pretraining.

5.2 Results on Downstream Tasks

To assess the learned representation of Point-JEPA, we take the context encoder and fine-tune it end-to-end with multiple datasets. We will report the mean and standard

deviation of 10 runs with different seeds for classification and segmentation tasks to account for variability between each run. For the few-shot evaluation, we fix the seed as we have 10 folds that can account for variability. We compare the performance of our methods to previous work, especially those pretrained with point cloud dataset formed from ShapeNet [76] without additional modality.

5.2.1 Linear Evaluation

After pretraining on ShapeNet [76], we evaluate the learned representation via linear probing on ModelNet40 [80]. Specifically, we freeze the learned context encoder and place the SVM classifier on top. To enforce invariance to geometric transformation, we utilize max and mean pooling on the output of the Transformer encoder [20, 18]. Here, we utilize 1024 points for both training and test sets. As shown in Table 5.1, our method achieves state-of-the-art accuracy, providing +0.8% performance gain, showing the robustness of the learned representation.

Table 5.1: Linear Evaluation on ModelNet40 [80]. We compare Point-JEPA to self-supervised learning methods pretrained with point cloud data created from ShapeNet [76].
* signifies that the result for linear evaluation is not available in the original paper. We cite the results from [79, 81].

methods	Overall Accuracy
Latent-GAN [82]	85.7
3D-PointCapsNet [83]	88.9
STRL [84]	90.3
Sauder *et al.* [85]	90.6
Fu *et al.* [21]	91.4
Transformer-OcCo* [54]	89.6
Point-Bert* [54]	87.4
Point-MAE* [20]	90.02
Point-M2AE [79]	92.9
Point-JEPA (Ours)	**93.7** $\pm$ 0.2

5.2.2 Classification on Synthetic Dataset

We also investigate the performance of the learned representation via end-to-end fine-tuning. After pretraining, we utilize the context encoder to extract the max and average pooled outputs. A three-layer MLP then processes these outputs for classification tasks. This class-specific head, as well as the context encoder, is fine-tuned end-to-end on ModelNet40 [80]. The dataset contains 12311 synthetic 3D objects from 40 distinct categories.

Table 5.2: Result of fine-tuning on ModelNet40 [80]. We report the overall accuracy with and without voting in percentage.

	Overall Accuracy	
Method	+Voting	−Voting
Transformer-OcCo [54]	92.1	–
ParAE [86]	–	92.9
STRL [84]	93.1	–
Point-BERT [54]	93.2	92.7
PointGLR [87]	–	93.0
OcCo [16]	93.0	–
MaskPoint [88]	93.8	–
Point-MAE [20]	93.8	93.2
Point-M2AE [79]	94.0	93.4
Point2Vec [18]	**94.8**	**94.7**
PointGPT-S [19]	94.0	–
Point-JEPA (Ours)	94.1 ± 0.1	93.8 ± 0.2

As shown in Table 5.2, our method shows competitive results compared to the state-of-the-art methods, showing the robustness of the learned representation when fine-tuned on a dataset that has similar characteristics as the pretraining dataset.

5.2.3 Classification on Real-World Dataset

In order to evaluate the learned representation of Point-JEPA, we fine-tune the ScanObjectNN [6] dataset. The dataset contains 2902 instances of 15 classes. Point cloud objects in ScanObjectNN are collected by scanning real-world objects. They contain background as well as some perturbed points. We utilize 2048 points for each training instance and 128 centre points for generating tokens. The dataset contains three variants. `OBJ-BG` includes the objects as well as their background while `OBJ-ONLY` only includes the objects. `OBJ-T50-RS` is the hardest variant of the dataset that applies perturbation to the point cloud object introducing varying degrees of background and partiality to the object. As shown in Table 5.3, our method achieves an improvement of +1% over the best-performing method in the `OBJ-BG` variant of the ScanObjNN [6] dataset, which presents a somewhat realistic representation of a point cloud. This shows the representation learned from the synthetic dataset is transferable to real-world datasets when using our method.

Table 5.3: Results of fine-tuning on ScanObjectNN.

	Overall Accuracy		
Method	OBJ-BG	OBJ-ONLY	OBJ-T50-RS
Transformer-OcCo [54]	84.9	85.5	78.8
Point-BERT [54]	87.4	88.1	83.1
MaskPoint [88]	89.3	88.1	84.6
Point-MAE [20]	90.0	88.3	85.2
Point-M2AE [79]	91.2	88.8	86.4
Point2Vec [18]	91.2	**90.4**	**87.5**
PointGPT-S [19]	91.6	90.0	86.9
Point-JEPA (Ours)	**92.9** $\pm$ 0.4	90.1 $\pm$ 0.2	86.6 $\pm$ 0.3

5.2.4 Few-Shot Classification

Following the standard evaluation protocol previously employed [20, 19, 18, 79], we conduct few-shot learning experiments on Modelnet40 [80]. Experiments are done in m-way, n-shot setting as shown in Table 5.4. Specifically, we randomly sample n instances of m classes for training. We select 20 instances of m support classes for evaluation. Under one setting, we run 10 independent runs on 10 different folds of the dataset, and we report the mean and standard deviation of overall accuracy. As shown in Table 5.4, our method exceeds the performance of the current state-of-the-art in all settings. Our method brings +1.1% improvements in the most difficult 10-way 10-shot setting. This shows the robustness of the learned representation of Point-JEPA, especially in the low-data regime.

Table 5.4: Result of Few-Shot classification on ModelNet40 [80]. 10 independent trials are completed under one setting. We report mean and standard deviation over 10 trials.

	Accuracy (%)			
Method	5-way 10-shot	5-way 20-shot	10-way 10-shot	10-way 20-shot
OcCo [89]	91.9 ± 3.6	93.9 ± 3.1	86.4 ± 5.4	91.3 ± 4.6
Transformer-OcCo [54]	94.0 ± 3.6	95.9 ± 2.3	89.4 ± 5.1	92.4 ± 4.6
Point-BERT [54]	94.6 ± 3.1	96.3 ± 2.7	91.0 ± 5.4	92.7 ± 5.1
MaskPoint [18]	95.0 ± 3.7	97.2 ± 1.7	91.4 ± 4.0	93.4 ± 3.5
Point-MAE [20]	96.3 ± 2.5	97.8 ± 1.8	92.6 ± 4.1	95.0 ± 3.0
Point-M2AE [79]	96.8 ± 1.8	98.3 ± 1.4	92.3 ± 4.5	95.0 ± 3.0
Point2Vec [18]	97.0 ± 2.8	98.7 ± 1.2	93.9 ± 4.1	95.8 ± 3.1
PointGPT-S [19]	96.8 ± 2.0	98.6 ± 1.1	92.6 ± 4.6	95.2 ± 3.4
Point-JEPA (Ours)	$\mathbf{97.4} \pm 2.2$	$\mathbf{99.2} \pm 0.8$	$\mathbf{95.0} \pm 3.6$	$\mathbf{96.4} \pm 2.7$

5.3 Part Segmentation

The learned representation is also tested against the part segmentation task. Here, we utilize the ShapeNetPart [76] dataset, made of 16881 objects from 16 categories. We utilize the identical architecture employed in [18] for this task. Specifically, we take the embeddings from 4^{th}, 8^{th}, and 12^{th} Transformer block and take the average of them. Then, we apply mean and average pooling to the averaged output. The pooled embeddings and one-hot encoded class label of an object are used as a global feature vector for that object. The averaged output is also up-sampled using PontNet++ [90] feature propagation layer to create a feature vector for each point. Then, each feature vector is concatenated with the global feature, and a shared MLP is utilized to predict the segmentation label for the given vector. The results are listed in Table 5.5. Although competitive, the performance achieved by our method is slightly worse than the state-of-the-art methods. This, coupled with the fact that our learned presentation has shown to be linearly separable, shows that our method is better suited for classification downstream tasks.

Table 5.5: Result of part segmentation on ShapeNetPart [76]. mIoU_C is the mean IoU for all part categories, and mIoU_I is the mean IoU for all instances.

Method	mIoU_C	mIoU_I
Transformer-OcCo [54]	83.4	85.1
Point-Bert [54]	84.1	85.6
MaskPoint [88]	84.4	86.0
Point-MAE [20]	84.1	86.1
Point-M2AE [79]	**84.9**	**86.5**
Point2Vec [18]	84.6	86.3
PointGPT-S [19]	84.1	86.2
Point-JEPA (Ours)	83.9 ± 0.1	85.8 ± 0.1

5.4 Ablation Study

We conducted thorough ablation studies to understand the effect of moving parts of Point-JEPA. We run pretraining on the ShapeNet [76] dataset under various settings and evaluate the learned representation with linear probing on the ModelNet40 [80] dataset.

5.4.1 Masking Strategy

In this section, we investigate the impact of the masking type on the performance. We consider single-block masking and multi-block masking. For single-block masking strategies, we consider random masking and contiguous masking. For random masking, we randomly select the 60% of indices out of all encoded embedding vectors. Similarly, for contiguous masking, embedding vectors that are spatially contiguous are selected. In this setting, tokens that do not correspond to the selected indices of the target blocks are used as context. On the other hand, in the muli-block masking setting, we sample multiple possibly overlapping spatially contiguous embedding vectors as targets, and we mask out the corresponding tokens already utilized for targets. In this setting, we choose context tokens in the ratio between 0.4 and 0.75 out of all available tokens. As shown in Table 5.6, the single-block masking achieves sub-optimal performance regardless of the spatial contiguity of the target embedding. It shows that our method learns better representation by utilizing smaller amount of targets with a larger frequency.

Targets			**Context**		
Strategy	Ratio	Freq.	Strategy	Ratio	**Modelnet40 Linear**
random	(0.6, 0.6)	1	rest	–	92.5
contiguous	(0.6, 0.6)	1	rest	–	92.3
contiguous	(0.15, 0.2)	4	contiguous	(0.4, 0.75)	93.7

Table 5.6: Different types of target selection strategies and its affect to the learned representation.

5.4.2 Ratio of Targets

In this study, we change the ratio of the selected embedding vectors for targets while keeping the number of targets and the ratio of context tokens fixed. As shown in Table 5.7, the performance increases when you increase the ratio to a certain ratio. However, beyond this point, further increasing the ratio results in decreased performance. This implies that Point-JEPA does not require a large portion of the encoded embedding vectors as targets and benefits from sufficient available tokens for context encoding.

Targets		**Context**	
Ratio	Freq.	Ratio	**Modelnet40 Linear**
(0.1, 0.2)	4	(0.85, 1.0)	93.0
(0.15, 0.2)	4	(0.85, 1.0)	93.3
(0.2, 0.25)	4	(0.85, 1.0)	93.2
(0.25, 0.3)	4	(0.85, 1.0)	92.4
(0.3, 0.35)	4	(0.85, 1.0)	90.5
(0.35, 0.4)	4	(0.85, 1.0)	84.6

Table 5.7: The ratio of encoded embedding vectors selected for each target.

5.4.3 Ratio of Context

In this study, we change the ratio of tokens selected for context encoding while keeping the number of the targets and the ratio range for targets fixed. As shown in Table 5.8, having a relatively large difference between the lower and upper bound of the ratio can improve performance. This implies that Point-JEPA learns a better representation when the number of selected context tokens varies more between training iterations. Additionally, when the upper bounds of the ratio are constrained, we see increased performance.

Targets		Context	
Ratio	Freq.	Ratio	**Modelnet40 Linear**
(0.15, 0.2)	4	(0.85, 1.0)	93.1
(0.15, 0.2)	4	(0.75, 1.0)	92.8
(0.15, 0.2)	4	(0.65, 1.0)	93.4
(0.15, 0.2)	4	(0.45, 1.0)	93.6
(0.15, 0.2)	4	(0.6, 0.75)	93.4
(0.15, 0.2)	4	(0.5, 0.75)	93.1
(0.15, 0.2)	4	(0.4, 0.75)	93.7

Table 5.8: The ratio of tokens selected for context encoding.

5.4.4 Number of Target Block

We will also consider the effect of the number of blocks chosen for targets on the performance of the learned representation while we keep the ratio for targets and context fixed. As shown in Table 5.9, the performance increases as we increase the number of targets. However, the performance decreases as you increase the number of target blocks after a specific frequency. Similarly to what we observed in Section 5.4.2, we note that our method benefits from having sufficient amount of tokens available for context encoding.

5.4.5 Predictor Depth

We also study the effect of the predictor's depth on the learned representation. To this end, we vary the predictor depth and observe its effect on the linear evaluation accuracy. As shown in Table 5.10, Point-JEPA benefits from a deeper predictor.

Targets		**Context**	
Ratio	Freq.	Ratio	**Modelnet40 Linear**
(0.15, 0.2)	1	(0.85, 1.0)	93.0
(0.15, 0.2)	2	(0.85, 1.0)	93.5
(0.15, 0.2)	3	(0.4, 0.75)	93.4
(0.15, 0.2)	4	(0.4, 0.75)	93.7
(0.15, 0.2)	5	(0.4, 0.75)	93.4
(0.15, 0.2)	6	(0.4, 0.75)	93.2

Table 5.9: The number of Target blocks.

Predictor Depth	**Modelnet40 Linear**
2	92.5
3	92.8
4	93.2
5	93.4
6	93.7

Table 5.10: Predictor depth and its effect on learned representation.

5.5 Initial Point for Sequencer

In this section, we study the importance of the initial centre point in the sequencer. We study a sequencer that takes the initial index of the centre point against another sequencer that takes the minimum sum of the coordinates as the initial point. As previously mentioned, selecting the first centre point presented in the sequencer does not guarantee the selection from the edge of the object. On the other hand, selecting the point with minimum coordinate sum at least guarantees that the starting centre point in the sequencer lies near the edge of the object. As shown in Table 5.11, taking the point near the edge of the object helps with learning stronger representation.

Sequencer Initial Point	**Modelnet40 Linear**
Minimum index	92.7
Minimum Coordinate Sum	93.7

Table 5.11: The initial selected point in the sequencer and its effect on the learned representation.

5.6 Visualization of Learned Representation

In order to qualitatively analyze the learned representation, we reduce the dimensional of the learned representation by utilizing t-SNE [91]. Specifically, we introduce max and mean pooling on the output of the context encoder, similar to our linear evaluation setup, and apply t-SNE on the pooled embedding. We visualize the learned representation on ModelNet40 [80] with no fine-tuning on the dataset. In other words, the context encoder has not been trained with this data. Despite this, our context encoder shows discriminative features as shown in Figure 5.1, which is evidence that Point-JEPA has learned a meaningful representation.

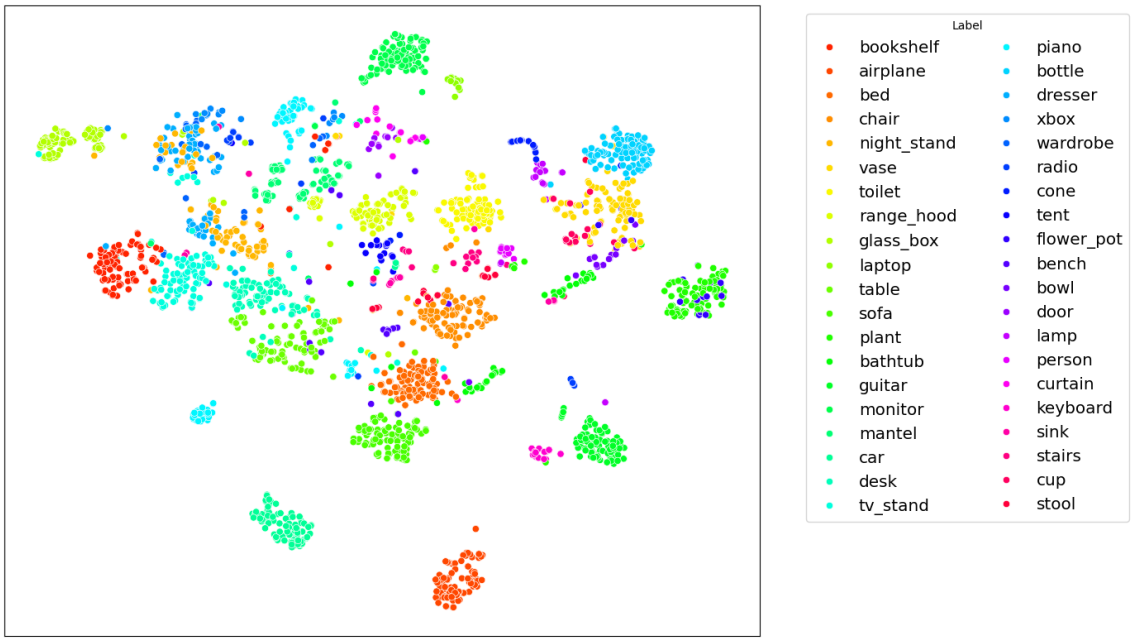

Figure 5.1: Visualization of the learned representation in two dimensions using t-SNE [91]. We visualize the learned representation of Point-JEPA on ModelNet40 [80] with no fine-tuning.

Chapter 6

Conclusion and Future Direction

This work introduces Point-JEPA, a joint-embedding predictive architecture applied to point cloud objects. In order to efficiently select targets and context blocks, we introduced a sequencer, which orders the centre points and their corresponding tokens by iteratively selecting the next closest centre point. This eliminates the necessity of computing spatial proximity between every pair of tokens or embedding vectors during the selection of the targets and context. Compared to Point2Vec [18] and PointGTP [19], our method shows improved performance in classification in a low data regime, showing the robustness of the learned representation. Additionally, our method achieves a state-of-the-art result with linear SVM classification on the ModelNet40 [80] dataset, showing that features produced by the pretrained context encoder are linearly separable. However, it is worth noting that our method performs slightly worse in the part segmentation task compared to the previous studies utilizing Transformer architecture, as shown in Table 5.5. This indicates that the learned representation of Point-JEPA focuses on the global feature over local features and is better suited for classification downstream tasks.

In this work, we fixed the size of the Transformer in order to fairly compare our methods to the previous methods that utilize the standard Transformer architecture. Because of our method's efficiency in the pretraining time, it should easily scale to a

larger Transformer with a larger pretraining dataset similar to PointGPT [19]. Looking into the performance when we scale our method can be one promising future direction. Additionally, we kept the point cloud tokenizer similar to the previous methods for a similar reason. We found a lack of diversity in the point cloud tokenizer. Because of recent advancements in point cloud processing, it is possible that there are better local feature extractors for point cloud tokenization. Looking into an alternative tokenization method can be another future direction.

Bibliography

[1] Armeni I, Sener O, Zamir AR, et al. 3D Semantic Parsing of Large-Scale Indoor Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2016; pages 1534–1543. doi:10.1109/CVPR.2016.170.

[2] Baruch G, Chen Z, Dehghan A, et al. ARKitScenes - A Diverse Real-World Dataset For 3D Indoor Scene Understanding Using Mobile RGB-D Data. *CoRR* 2021;abs/2111.08897.

[3] Behley J, Garbade M, Milioto A, et al. A Dataset for Semantic Segmentation of Point Cloud Sequences. *CoRR* 2019;abs/1904.01416.

[4] Caesar H, Bankiti V, Lang AH, et al. nuScenes: A multimodal dataset for autonomous driving. *CoRR* 2019;abs/1903.11027.

[5] Dai A, Chang AX, Savva M, et al. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. *CoRR* 2017;abs/1702.04405.

[6] Uy MA, Pham Q, Hua B, et al. Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data. *CoRR* 2019; abs/1908.04616.

[7] Schult J, Engelmann F, Hermans A, et al. Mask3D: Mask Transformer for 3D Semantic Instance Segmentation 2023.

[8] Choy CB, Gwak J, and Savarese S. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. *CoRR* 2019;abs/1904.08755.

[9] Qi CR, Su H, Mo K, et al. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *CoRR* 2016;abs/1612.00593.

[10] Qi CR, Yi L, Su H, et al. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *CoRR* 2017;abs/1706.02413.

[11] Chen T, Kornblith S, Norouzi M, et al. A Simple Framework for Contrastive Learning of Visual Representations. *CoRR* 2020;abs/2002.05709.

[12] Chen X and He K. Exploring Simple Siamese Representation Learning. *CoRR* 2020;abs/2011.10566.

[13] Grill J, Strub F, Altché F, et al. Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning. *CoRR* 2020;abs/2006.07733.

[14] Radford A, Wu J, Child R, et al. Improving Language Understanding by Generative Pre-Training. In *arxiv* 2018; .

[15] He K, Chen X, Xie S, et al. Masked Autoencoders Are Scalable Vision Learners. *CoRR* 2021;abs/2111.06377.

[16] He K, Fan H, Wu Y, et al. Momentum Contrast for Unsupervised Visual Representation Learning. *CoRR* 2019;abs/1911.05722.

[17] Caron M, Touvron H, Misra I, et al. Emerging Properties in Self-Supervised Vision Transformers. *CoRR* 2021;abs/2104.14294.

[18] Zeid KA, Schult J, Hermans A, et al. Point2Vec for Self-Supervised Representation Learning on Point Clouds 2023.

[19] Chen G, Wang M, Yang Y, et al. PointGPT: Auto-regressively Generative Pre-training from Point Clouds 2023.

[20] Pang Y, Wang W, Tay FEH, et al. Masked Autoencoders for Point Cloud Self-supervised Learning 2022.

[21] Fu K, Gao P, Zhang R, et al. Distillation with Contrast is All You Need for Self-Supervised Point Cloud Representation Learning 2022.

[22] LeCun Y. A Path Towards Autonomous Machine Intelligence. https://openreview.net/forum?id=BZ5a1r-kVsf 2022. Accessed: 2024-03-22.

[23] Assran M, Duval Q, Misra I, et al. Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture 2023.

[24] Gui J, Chen T, Zhang J, et al. A Survey on Self-supervised Learning: Algorithms, Applications, and Future Trends 2023.

[25] Roberts DA, Yaida S, and Hanin B. *The Principles of Deep Learning Theory*. Cambridge University Press 2022. https://deeplearningtheory.com.

[26] Alom MZ, Taha TM, Yakopcic C, et al. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *CoRR* 2018; abs/1803.01164.

[27] Alam M, Samad MD, Vidyaratne L, et al. Survey on Deep Neural Networks in Speech and Vision Systems. *CoRR* 2019;abs/1908.07656.

[28] Li Q, Peng H, Li J, et al. A Survey on Text Classification: From Shallow to Deep Learning. *CoRR* 2020;abs/2008.00364.

[29] Gidaris S, Singh P, and Komodakis N. Unsupervised Representation Learning by Predicting Image Rotations. *CoRR* 2018;abs/1803.07728.

[30] Larsson G, Maire M, and Shakhnarovich G. Colorization as a Proxy Task for Visual Understanding. *CoRR* 2017;abs/1703.04044.

[31] Noroozi M and Favaro P. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles. *CoRR* 2016;abs/1603.09246.

[32] Goyal P, Mahajan D, Gupta A, et al. Scaling and Benchmarking Self-Supervised Visual Representation Learning. *CoRR* 2019;abs/1905.01235.

[33] Zhang R, Isola P, and Efros AA. Colorful Image Colorization. *CoRR* 2016; abs/1603.08511.

[34] Zhang R, Zhu J, Isola P, et al. Real-Time User-Guided Image Colorization with Learned Deep Priors. *CoRR* 2017;abs/1705.02999.

[35] Chen X, Fan H, Girshick R, et al. Improved Baselines with Momentum Contrastive Learning 2020.

[36] Chen X, Xie S, and He K. An Empirical Study of Training Self-Supervised Vision Transformers 2021.

[37] Chen T, Kornblith S, Swersky K, et al. Big Self-Supervised Models are Strong Semi-Supervised Learners. *CoRR* 2020;abs/2006.10029.

[38] van den Oord A, Li Y, and Vinyals O. Representation Learning with Contrastive Predictive Coding. *CoRR* 2018;abs/1807.03748.

[39] Wu Z, Xiong Y, Yu SX, et al. Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination. *CoRR* 2018;abs/1805.01978.

[40] Bucila C, Caruana R, and Niculescu-Mizil A. Model compression. In *Knowledge Discovery and Data Mining* 2006; .

[41] Hinton G, Vinyals O, and Dean J. Distilling the Knowledge in a Neural Network 2015.

[42] Bao H, Dong L, and Wei F. BEiT: BERT Pre-Training of Image Transformers. *CoRR* 2021;abs/2106.08254.

[43] Xie Z, Zhang Z, Cao Y, et al. SimMIM: A Simple Framework for Masked Image Modeling. *CoRR* 2021;abs/2111.09886.

[44] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR* 2020;abs/2010.11929.

[45] Vaswani A, Shazeer N, Parmar N, et al. Attention Is All You Need. *CoRR* 2017; abs/1706.03762.

[46] Du B, Gao X, Hu W, et al. Self-Contrastive Learning with Hard Negative Sampling for Self-supervised Point Cloud Learning. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21. ACM 2021; doi: 10.1145/3474085.3475458.

[47] Zhang L and Zhu Z. Unsupervised Feature Learning for Point Cloud by Contrasting and Clustering With Graph Convolutional Neural Network. *CoRR* 2019; abs/1904.12359.

[48] Hartigan JA and Wong MA. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 1979; 28(1):100–108.

[49] Xie S, Gu J, Guo D, et al. PointContrast: Unsupervised Pre-training for 3D Point Cloud Understanding. *CoRR* 2020;abs/2007.10985.

[50] Hou J, Graham B, Nießner M, et al. Exploring Data-Efficient 3D Scene Understanding with Contrastive Scene Contexts. *CoRR* 2020;abs/2012.09165.

[51] Lal S, Prabhudesai M, Mediratta I, et al. CoCoNets: Continuous Contrastive 3D Scene Representations. *CoRR* 2021;abs/2104.03851.

[52] Liu Y, Yi L, Zhang S, et al. P4Contrast: Contrastive Learning with Pairs of Point-Pixel Pairs for RGB-D Scene Understanding. *CoRR* 2020;abs/2012.13089.

[53] Zhang Z, Girdhar R, Joulin A, et al. Self-Supervised Pretraining of 3D Features on any Point-Cloud. *CoRR* 2021;abs/2101.02691.

[54] Yu X, Tang L, Rao Y, et al. Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling. *CoRR* 2021;abs/2111.14819.

[55] Devlin J, Chang M, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* 2018;abs/1810.04805.

[56] Baevski A, Hsu W, Xu Q, et al. data2vec: A General Framework for Self-supervised Learning in Speech, Vision and Language. *CoRR* 2022; abs/2202.03555.

[57] Brown TB, Mann B, Ryder N, et al. Language Models are Few-Shot Learners. *CoRR* 2020;abs/2005.14165.

[58] Raffel C, Shazeer N, Roberts A, et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *CoRR* 2019;abs/1910.10683.

[59] Yuan L, Chen Y, Wang T, et al. Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet. *CoRR* 2021;abs/2101.11986.

[60] Liu Z, Lin Y, Cao Y, et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *CoRR* 2021;abs/2103.14030.

[61] Han K, Xiao A, Wu E, et al. Transformer in Transformer. *CoRR* 2021; abs/2103.00112.

[62] Wang W, Xie E, Li X, et al. Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. *CoRR* 2021;abs/2102.12122.

[63] Wang W, Yao L, Chen L, et al. CrossFormer: A Versatile Vision Transformer Based on Cross-scale Attention. *CoRR* 2021;abs/2108.00154.

[64] Yuan L, Hou Q, Jiang Z, et al. VOLO: Vision Outlooker for Visual Recognition. *CoRR* 2021;abs/2106.13112.

[65] Carion N, Massa F, Synnaeve G, et al. End-to-End Object Detection with Transformers. *CoRR* 2020;abs/2005.12872.

[66] Wang Q, Li B, Xiao T, et al. Learning Deep Transformer Models for Machine Translation. *CoRR* 2019;abs/1906.01787.

[67] Baevski A and Auli M. Adaptive Input Representations for Neural Language Modeling. *CoRR* 2018;abs/1809.10853.

[68] Guo M, Cai J, Liu Z, et al. PCT: Point Cloud Transformer. *CoRR* 2020; abs/2012.09688.

[69] Wang Y, Sun Y, Liu Z, et al. Dynamic Graph CNN for Learning on Point Clouds. *CoRR* 2018;abs/1801.07829.

[70] Zhao H, Jiang L, Jia J, et al. Point Transformer. *CoRR* 2020;abs/2012.09164.

[71] Wu L, Liu X, and Liu Q. Centroid Transformers: Learning to Abstract with Attention. *CoRR* 2021;abs/2102.08606.

[72] Wang Z, Wang Y, An L, et al. Local Transformer Network on 3D Point Cloud Semantic Segmentation. *Information* 2022;13(4). ISSN 2078-2489. doi:10.3390/info13040198.

[73] Lu D, Xie Q, Wei M, et al. Transformers in 3D Point Clouds: A Survey 2022.

[74] Eldar Y, Lindenbaum M, Porat M, et al. The farthest point strategy for progressive image sampling. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 2 - Conference B: Computer Vision & Image Processing. (Cat. No.94CH3440-5)* 1994; pages 93–97 vol.3. doi:10.1109/ICPR.1994.577129.

[75] Fix E and Hodges JL. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review / Revue Internationale de Statistique* 1989;57(3):238–247. ISSN 03067734, 17515823.

[76] Chang AX, Funkhouser T, Guibas L, et al. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago 2015.

[77] Loshchilov I and Hutter F. Fixing Weight Decay Regularization in Adam. *CoRR* 2017;abs/1711.05101.

[78] Loshchilov I and Hutter F. SGDR: Stochastic Gradient Descent with Restarts. *CoRR* 2016;abs/1608.03983.

[79] Zhang R, Guo Z, Fang R, et al. Point-M2AE: Multi-scale Masked Autoencoders for Hierarchical Point Cloud Pre-training 2022.

[80] Wu Z, Song S, Khosla A, et al. 3D ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2015; pages 1912–1920. doi:10.1109/CVPR.2015.7298801.

[81] Zhang R, Wang L, Qiao Y, et al. Learning 3D Representations from 2D Pre-trained Models via Image-to-Point Masked Autoencoders 2022.

[82] Achlioptas P, Diamanti O, Mitliagkas I, et al. Representation Learning and Adversarial Generation of 3D Point Clouds. *CoRR* 2017;abs/1707.02392.

[83] Zhao Y, Birdal T, Deng H, et al. 3D Point-Capsule Networks. *CoRR* 2018; abs/1812.10775.

[84] Huang S, Xie Y, Zhu SC, et al. Spatio-temporal Self-Supervised Representation Learning for 3D Point Clouds 2021.

[85] Sauder J and Sievers B. Context Prediction for Unsupervised Deep Learning on Point Clouds. *CoRR* 2019;abs/1901.08396.

[86] Eckart B, Yuan W, Liu C, et al. Self-Supervised Learning on 3D Point Clouds by Learning Discrete Generative Models. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* 2021; pages 8244–8253. doi: 10.1109/CVPR46437.2021.00815.

[87] Rao Y, Lu J, and Zhou J. Global-Local Bidirectional Reasoning for Unsupervised Representation Learning of 3D Point Clouds. *CoRR* 2020;abs/2003.12971.

[88] Liu H, Cai M, and Lee YJ. Masked Discrimination for Self-Supervised Learning on Point Clouds 2022.

[89] Wang H, Liu Q, Yue X, et al. Pre-Training by Completing Point Clouds. *CoRR* 2020;abs/2010.01089.

[90] Qi CR, Yi L, Su H, et al. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *CoRR* 2017;abs/1706.02413.

[91] van der Maaten L and Hinton GE. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 2008;9:2579–2605.