Computational, Label, and Data Efficiency in Deep Learning for Sparse 3D Data

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)

von der KIT-Fakultät für

Elektrotechnik und Informationstechnik

des Karlsruher Instituts für Technologie (KIT)

angenommene

DISSERTATION

von

Lanxiao Li, M.Sc.

geb. in Shaanxi, China

Tag der mündlichen Prüfung:	26.10.2023
Hauptreferent:	Prof. Dr.-Ing. Michael Heizmann, KIT
Korreferent:	Prof. Dr. Ralf Mikut, KIT

Abstract

The deep learning technology has made fast progress in recent years. It is widely applied to sparse 3D data to perform challenging tasks, *e.g.*, 3D object detection and semantic segmentation. However, the high performance of deep learning comes with high costs, including computational costs and the effort to capture and label data. This thesis investigates and improves the efficiency of deep learning for sparse 3D data to overcome the obstacles to the further development of this technology.

For better computational efficiency, a depth map-based 3D object detector is introduced. Also, transformer-based models are explored to process point clouds that cannot be represented as depth maps. The proposed novel architectures achieve competitive performance with lower computational costs than existing methods.

Also, to reduce the dependence on labeled data and improve label efficiency, this thesis researches self-supervised pre-training, which only requires unlabeled data. Specifically, it provides a closer look at invariance-based contrastive learning using 3D data and the masked auto-encoder for point clouds. Compared to directly training neural networks on target datasets, self-supervised pre-training brings a significant performance boost without additional labels.

Furthermore, synthetic data generation for pre-training is investigated to reduce the effort of capturing real-world 3D data and improve data efficiency. Instead of applying sophisticated simulation, this thesis generates data using a fully randomized approach. The generated synthetic data perform well with different neural networks and pre-training methods. Also, the performance is competitive compared to real-world data.

Zusammenfassung

Deep Learning hat in den letzten Jahren rasante Fortschritte gemacht. Es wird häufig auf dünnbesetzten 3D-Daten eingesetzt, um anspruchsvolle Aufgaben zu erfüllen, wie beispielsweise 3D-Objekterkennung und semantische Segmentierung. Die hohe Leistung von *Deep Learning* ist jedoch mit großem Aufwand verbunden, einschließlich dem Rechenaufwand und dem Aufwand für die Gewinnung und Annotierung der Daten. Diese Arbeit untersucht und verbessert die Effizienz von *Deep Learning* für dünnbesetzte 3D-Daten, um die Hindernisse für die weitere Entwicklung dieser Technologie zu überwinden.

Zur Verbesserung der Recheneffizienz wird ein tiefenkartenbasierter 3D-Objektdetektor eingeführt. Außerdem werden *Transformer*-basierte Modelle untersucht, um Punktwolken zu verarbeiten, die sich nicht als Tiefenkarten darstellen lassen. Die vorgeschlagenen neuen Architekturen erreichen eine vergleichbare Leistung mit geringerem Rechenaufwand als bestehende Methoden.

Um die Abhängigkeit von annotierten Daten zu verringern und die Effizienz der Annotierung zu verbessern, wird in dieser Arbeit das selbstüberwachte Vortraining erforscht, für das keine Labels erforderlich sind. Insbesondere werden das invarianzbasierte kontrastive Lernen für 3D-Daten und der *Masked Autoencoder* für Punktwolken näher betrachtet. Im Vergleich zum direkten Training auf Zieldatensätzen bringt das selbstüberwachte Vortraining einen deutlichen Leistungsschub ohne zusätzliche Labels.

Darüber hinaus wird die Erzeugung synthetischer Daten für das Vortraining untersucht, um den Aufwand für die Erfassung realer 3D-Daten zu verringern und die Dateneffizienz zu verbessern. Anstelle einer ausgefeilten Simulation werden in dieser Arbeit Daten mit einem vollständig stochastischen Ansatz erzeugt. Die generierten synthetischen Daten schneiden mit verschiedenen neuronalen Netzen und vortrainierenden Methoden gut ab. Außerdem ist die Leistung im Vergleich zu realen Daten konkurrenzfähig.

Contents

Nomenclature

Common abbreviations

Abbreviation	Description
cf.	*Confer* (lat. refer to)
e.g.	*Exempli gratia* (lat. for example)
et al.	*Et alii* (lat. and others)
etc.	*Et cetera* (lat. and so forth)
i.e.	*Id est* (lat. that means)
1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
4D	Four-dimensional
ANN	Artificial neural networks
AP	Average precision
ASIC	Application specific integrated circuit
BYOL	Bootstrap your own latent
CAD	Computer aided design
CL	Contrastive learning
CLIP	Contrastive language-image pre-training
CNN	Convolutional neural network
CPU	Central processing unit
CT	Computed tomography
DDCo	Depth-depth contrast
DETR	Detection transformer
DPCo	Depth-point contrast
DVCo	Depth-voxel contrast
EMA	exponential moving average
FC	Fully connected

Abbreviation	**Description**
FPC	Farthest point clustering
FPS	Farthest point sampling
FPS	Frame per second
GAN	Generative adversarial network
GFLOPs	Giga floating point operations
GIoU	Generalized intersection over union
GPT	Generative pre-trained transformer
GPU	Graphics processing unit
InfoNCE	Information noise contrast estimation
IoU	Intersection over union
IPCo	Image-point contrast
KNN	k-nearest-neighbor
LiDAR	Light detecting and ranging
mAcc	Mean accuracy
MAE	Masked autoencoder
mAP	Mean average precision
mIoU	Mean intersection over union
MLP	Multi-layer perceptron
MoCo	Momentum contrast
NLP	Natural language processing
NMS	Non-maximum suppression
NPU	Neural processing unit
OC	Overall accuracy
PPCo	Point-point contrast
PVCo	Point-voxel contrast
RDConv	Relative depth convolution
ReLU	Rectified linear unit
RGB-D	Red green blue depth
RGB	Red green blue
RM	Random rooms
RNG	Random number generator
RNN	Recurrent neural network
SGD	Stochastic gradient descent
SLAM	Simultaneous location and mapping
SOTA	State-of-the-art

Abbreviation	Description
SSL	Self-supervised learning
SSP	Self-supervised pre-training
ViT	Vision transformer
N/A	Not applicable
w/o	Without
w/	With

Symbols

Latin letters

Symbol	Description
$\mathbf{v}$	Vector
v_i	Element i of vector $\mathbf{v}$ (scalar)
$\mathbf{V}$	Matrix
$V_{i,j}$	Element (i,j) of matrix (scalar)
$\boldsymbol{\mathcal{V}}$	N-dimensional tensor with shape $(C_1, C_2, \dots, C_N)$ (assuming $N \geq 3$)
$\mathbf{v}_{i,j}$	Element (i,j) of tensor $\boldsymbol{\mathcal{V}}$ with length C_3 (assuming $\boldsymbol{\mathcal{V}}$ is 3D)
$\mathbf{V}_{i,j}$	Element (i,j) of tensor $\boldsymbol{\mathcal{V}}$ with shape (C_3, C_4) (assuming $\boldsymbol{\mathcal{V}}$ is 4D)
$\{\mathbf{v}^i\}_{i=1}^{M}$	Vector set with M vectors
$\mathbf{b}$	Bias
B	Batch size
C	Channel number
$\mathbf{D}$	Depth map (matrix notation)
$d(\dots)$	Depth map (function notation)
$\mathbf{E}$	Position embedding
$\mathcal{I}$	Image
l	Length
$L(\dots)$	Loss function
$n(\dots)$	Neural network as a function

Symbol	Description
$o(\dots)$	Offset function
$\mathbf{P}$	Permutation matrix
$\mathbf{s}$	Sample point
$\mathbf{u}$	Input feature
V	Volume
$\mathbf{W}$	Weight matrix
$\boldsymbol{\mathcal{W}}$	Convolution kernel
$\mathbf{x}$	Point
$\{\mathbf{x}^i\}_{i=1}^M$	Point set with M points
$X_\mathrm{q}(\dots)$	Occupancy grid (function notation)
$\mathbf{y}$	Label
$\hat{\mathbf{y}}$	Prediction
$\mathbf{0}$	Zero vector

Greek letters

Symbol	Description
α	Abstract data format
β	Abstract data format
Ω	Point set
$\{\Omega_i\}_{i=1}^M$	M point sets
$\sigma(\dots)$	Activation function
τ	Temperature coefficient
$\boldsymbol{\theta}$	Trainable parameters

Superscripts

Index	Description
$(\bullet)^\mathrm{a}$	Anchor
$(\bullet)^\mathrm{D}$	Dropped
$(\bullet)^\mathrm{M}$	Masked

Index	Description
$(\bullet)^{\mathrm{n}}$	Negative
$(\bullet)^{\mathrm{p}}$	Positive
$(\bullet)^{\mathrm{q}}$	Quantization
$(\bullet)^{\mathrm{ref}}$	Reference
$(\bullet)^{\mathrm{R}}$	Reserved
$(\bullet)^{\top}$	Transposed

Subscripts

Index	Description
$(\bullet)_{\mathrm{cls}}$	Classification
$(\bullet)_{\mathrm{g}}$	Global
$(\bullet)_{\mathrm{in}}$	Input
$(\bullet)_{\mathrm{l}}$	Local
$(\bullet)_{\mathrm{out}}$	Output
$(\bullet)_{\mathrm{q}}$	Quantization

Mathematical operators

Operator	Description
$\lVert\cdot\rVert$	Euclidean norm (L^2 norm)
$\lfloor\cdot\rfloor$	Rounding down

Preface

I am profoundly grateful for the support and contributions that shaped this thesis during my time at IIIT.

First, I sincerely express my deepest appreciation to my supervisor, Prof. Dr.-Ing. Michael Heizmann. His constant guidance is the cornerstone of my doctoral journey. Throughout the four years, his friendly and supportive mentorship nurtured my academic development and boosted my confidence.

I thank Prof. Dr. Ralf Mikut for his review of my thesis and for offering inspiring suggestions that significantly enhanced its quality.

Also, I wish to express my gratitude to all my colleagues at IIIT whose collaboration made this academic pursuit an enjoyful experience. Particularly, I thank Muen Jin, Fabian Leven, Hannes Weinreuter, Johannes Anastasiadis, Daniel Leyer, Markus Schwabe, Daniel Diaz Ocampo, Manuel Bihler, Theresa Panther, Johannes Steffens, and Erik Tabuchi Barczak for reviewing this thesis. Meanwhile, I want to thank Dieter Brandt, Manuela Moritz, Patricia Nestl, and Marvin Winkler for their administrative and technical support.

Moreover, I thank the Baden-Württenberg Stiftung for the financial support during the first two years of my research through the KOMO3D project. Additionally, I am grateful to the state of Baden-Württenberg for providing access to the powerful computating platform bwHPC.

Furthermore, I want to thank my family. I thank Huihui and Tutu, my lovely cats, for their companionship and for allowing me to include their adorable pictures in this thesis. I thank my parents for their mental support throughout my journey in a foreign land and their encouragement during the challenging times. Above all, I want to thank my wife, Yanling. Her love, care, encouragement, and patience during this long pursuit were my constant source of strength.

As I finished this thesis shortly after my thirtieth birthday, I finally want to express my gratitude to everyone who has entered my life. To all of you, thank you for being a part of my story.

Karlsruhe, November 2023 Lanxiao Li

1 Introduction

1.1 What is Deep Learning for Sparse 3D Data?

Perception of the three-dimensional (3D) world is a long-existing demand in computer vision. For instance, to grip an object, a robot arm must know its accurate size, location, and orientation. Also, a self-driving car has to understand the circumstance and determine its relative position to other vehicles and pedestrians. Furthermore, an industrial visual inspection device must examine the surface of a product to identify potential manufacturing failures. To accomplish these goals, a 3D measurement of the surroundings, *e.g.*, the 3D coordinates of sample points on objects' surfaces, must be available. Also, the acquired information has to be understood to obtain recognition, *e.g.*, the existence and semantics of objects.

In recent years, distance sensors have become compacter, faster, and more affordable and have found a lot of practical applications. For instance, many self-driving cars are equipped with LiDAR (light detection and ranging) sensors [126, 187], which measure the distance to surrounding obstacles within a large range. In manufacturing, laser scanners are often applied to inspect surfaces with a high accuracy [89, 234]. Also, some consumer electronics, *e.g.*, tablets, are embedded with depth cameras, allowing users to capture 3D data in their daily life [9].

Since these sensors can only measure the distance to surfaces and cannot inspect inner structures, the obtained 3D data are sparse, *e.g.*, most coordinates in the 3D space do not contain meaningful measurement values. The sparsity and irregularity make it challenging to extract information from such data. In recent years, the deep learning technology has significantly boosted the processing and understanding of sparse 3D data. One main characteristic of deep learning distinguishing it from conventional machine learning is the application of deep artificial neural networks (ANNs) [68]. Specifically, it models the direct mapping from input data to labels using deep ANNs. In this context, labels indicate the desired output of the mapping, *e.g.*, the correct semantic classes of objects in classification tasks or bounding boxes in object detection tasks. Labels represent the human interpretation of the corresponding data. To acquire labels, people often have to annotate data manually. With data and labels given, a deep ANN is trained by minimizing the difference between the network's outputs and labels using back-propagation [118–120].

Deep learning has the advantage that it requires less engineering effort since it trains ANNs in an end-to-end manner and enables models to learn optimal features automatically. On the contrary, classical data processing methods heavily rely on hand-crafted descriptors to capture features [16, 40, 107, 141, 191, 192, 208]. Also, because less hard-coded design is involved in deep learning, it is adaptive to data, and its performance is not limited by the domain knowledge of the designers. In practice, deep learning-based methods outperform classical ones with a clear margin [80, 114, 170, 207].

Aside from its success in image processing [80, 81, 114, 180, 182, 186, 207] and natural language processing [46, 88, 177, 235], deep learning has been widely applied to 3D data processing and understanding tasks, *e.g.*, object classification [170, 171, 228], semantic segmentation [39, 96, 176, 228, 251], object detection [117, 152, 173, 203, 248, 270], object tracking [27, 250, 272], point cloud completion [166, 249, 274], 3D registration [8, 38, 211], and point cloud denoising [142, 168, 277].

In this thesis, the methodology of using deep learning to process sparse 3D data and accomplish 3D computer vision tasks is referred to as *deep learning for sparse 3D data*.

1.2 Efficiency: a Challenge in Deep Learning for Sparse 3D Data

Despite the promising progress of deep learning, it still faces challenges. One of the most concerning obstacles to its application and development is the high costs of training and deploying ANNs. The costs can be divided into three categories:

Firstly, the *computational cost*. Generally, deeper (*i.e.*, having more layers and trainable parameters) neural networks have stronger representation power than shallower ones [52, 221, 223, 275]. To perform challenging tasks, modern ANNs are becoming increasingly deeper [80, 86, 114, 207]. However, the increased depth inevitably requires more computational power. In terms of training, it means more hardware resources, *e.g.*, GPUs (graphics processing units), are necessary. Also, the energy consumption and time cost for training deep neural networks are considerably high [201]. Meanwhile, the high computational cost makes deploying trained networks hard in practice. For instance, the deployment in cloud computing scenarios suffers from higher operating costs and latency. On edge devices, the available computational sources often cannot fulfill the requirement of the increasingly deeper architecture of modern ANNs, since they are limited by the power consumption and the space [267].

The issues mentioned above are commonly observed in multiple domains in deep learning, *e.g.*, image and natural language processing. For 3D data, the problem of computational cost is more detrimental. First, due to the higher dimension, processing 3D data is generally more expensive than the 1D and 2D cases. For instance, 3×3 convolution is one of the most fundamental and commonly used operations in convolutional neural networks (CNNs) for image processing [80, 207]. However, in the case of 3D data, 3×3×3 convolution must be applied for a similar effect. It means the complexity of a convolution operation is increased from quadratic to cubic with respect to the side length of the convolution kernel. Also, 3D data in deep learning are often represented as point clouds, which are sparse, irregular, and unordered. Due to this property, processing point clouds requires a lot of random memory access [138, 171, 228], significantly increasing the run time for training and inference. Moreover, neural networks for 3D vision tasks often have to be deployed on edge devices, *e.g.*, embedded systems in robots and self-driving cars, where the computational resources are limited by *e.g.*, available space, energy consumption, and manufacturing cost. At the same time, these applications are also sensitive to latency for safety reasons. Running deep ANNs on such platforms while fulfilling real-time requirements is especially challenging. Another technical issue in the computation for sparse 3D data is the limited hardware and software support. In the computer vision domain, most neural ASICs (application-specific integrated circuits) and their software tool-kits only support fundamental 2D operations, *e.g.*, 2D convolution and 2D pooling. However, irregular and unordered 3D data often rely on special operations [39, 171, 228]. Deploying neural networks for sparse 3D data has thus more technical constraints and requires more engineering effort. More details about hardware deployment for ANNs are provided in Appendix D.

Besides the computational cost, the *cost for data labeling* also bottlenecks the progress of deep learning for sparse 3D data. As explained in Section 1.1, deep learning relies on manually labeled data to provide supervision in training. The importance of large-scale labeled datasets is clearly shown in the history of image processing using deep learning. The research on ANNs began in 1940s [147]. In 1989, LeCun *et al.* [119] started to apply CNNs for handwritten digit recognition. However, early CNNs couldn't outperform classical descriptor-based methods in image recognition until Krizhevsky *et al.* [114] introduced AlexNet, an architecture for CNNs, in 2012. One essential factor that boosted this progress was the introduction of large-scale labeled datasets [68]. For instance, Deng *et al.* [44] published the ImageNet dataset in 2009, consisting of millions of images labeled with semantic classes. Krizhevsky *et al.* designed and trained AlexNet for image classification on ImageNet. The outstanding performance of AlexNet then sparked tremendous research interest in deep learning methods. Today, ImageNet is still one of the most widely used datasets for image processing. Meanwhile, datasets larger than ImageNet have been introduced in recent years [116, 143, 218]. Labeling the raw data becomes more challenging with the increasing size of datasets. Some simple labels can be obtained with less effort. For instance, labels for images can be extracted from their hashtags given by users in social media [143]. However, more detailed and complicated labels, *e.g.*, bounding boxes and semantic maps, are often necessary in practice, and a dedicated labeling process is inevitable.

Compared to 2D images, labeling 3D data is significantly more laborious and time-consuming. For instance, to annotate 2D bounding boxes, a data labeler can view the entire image on the screen and draw boxes by simply clicking and dragging the mouse. However, observing 3D point clouds and creating 3D bounding boxes using ordinary input-output de-

Table 1.1 Size comparison of well-known datasets in 2D and 3D computer vision. The last five datasets contain additional data, *e.g.*, unannotated frames and color images. The reported sample numbers refer to point clouds or depth maps annotated with bounding boxes. Year: release year of the datasets. RGB: color images. RGB-D: aligned color images and depth maps. PCD: point clouds. Class.: classification. Det.: object detection.

Dataset	Data Type	Year	Samples	Task
ImageNet-1K [44]	RGB	2009	1M	class.
ImageNet-21K [44]	RGB	2011	14M	class.
JFT-300M [218]	RGB	2017	300M	class.
Instagram [143]	RGB	2018	3.5B	class.
COCO [129]	RGB	2014	328K	det.
Open Image V4 [116]	RGB	2020	30M	det.
ModelNet40 [255]	CAD models	2015	12K	class.
ShapeNetCore [26]	CAD models	2015	51K	class.
ScanNet [43]	indoor PCD	2017	1.5K	det.
SUN RGB-D [212]	indoor RGB-D	2015	10K	det.
KITTI [65]	outdoor PCD	2013	15K	det.
nuScenes [22]	outdoor PCD	2020	40K	det.
Waymo [219]	outdoor PCD	2020	230K	det.

vices are non-trivial. More operations are required, *e.g.*, rotating a point cloud to view it from different perspectives and adjusting bounding boxes in the 3D space. In commercial data labeling services, 3D labels are orders of magnitude more expensive than their 2D counterparts. For instance, labeling one frame of point clouds costs 3 dollars using Amazon SageMaker Ground Truth, whereas the price for images varies from 0.02 to 0.08 dollars depending on the order volume[1]. Therefore, the cost to label large-scale 3D datasets, which are essential to training strong deep learning models for 3D computer vision, is extremely high.

Besides costs for computation and labels, the *cost for data capturing* is also critical in deep learning for sparse 3D data. Unlabeled raw data are sometimes easy to obtain. For instance, an enormous amount of

[1] Source: https://aws.amazon.com/sagemaker/data-labeling/pricing. Last accessed on 2023.03.28.

images and texts are available on the Internet. Thus, many datasets are created based on existing data [44, 116, 143, 218]. Their creators pay more attention to selecting and filtering the data since capturing or creating the data from scratch is unnecessary. However, there are significantly fewer openly accessible 3D data on the Internet, despite some synthetic CAD (computer aided design) models [26, 255]. Capturing real-world 3D data requires considerable labor, time, and measurement devices. Due to the high cost, the development of large-scale 3D datasets lags behind the 2D counterpart. The sizes of some well-known datasets in computer vision are summarized in Table 1.1. The 2D data in the upper half are collected from the Internet. However, all real-world 3D datasets in Table 1.1 are captured by their creators, while the two synthetic ones are also gathered from the Internet. Compared to 2D datasets at the same time, 3D ones are orders of magnitude smaller in terms of data amount. For instance, the Waymo dataset [219], one of the largest publically accessible 3D datasets, is 130 times smaller than Open Image V4 [116], although both datasets were released in 2020. Previous research [86, 218] empirically reports a power-law between the size of training datasets and the performance of neural networks: as the size increases exponentially, the performance (*i.e.*, accuracy in classification tasks) grows linearly. Therefore, the scarcity of real-world 3D data significantly constrains the progress of deep learning for sparse 3D data.

This thesis aims to overcome the efficiency issues of deep learning for sparse 3D data, which hinder its application and development. Corresponding to the three aforementioned types of costs, *i.e.*, the computational cost, the cost for data labeling, and for data capturing, three aspects of efficiency are discussed:

Computational efficiency, which is primarily affected by the architecture of ANNs. According to different applications and scenarios, the computational efficiency of ANNs can have diverse meanings. For real-time applications, it primarily means the networks require less computation and memory access, so the latency between the input and output can be kept low. For centralized computing in servers, high efficiency mainly means a high throughput of the inference, *e.g.*, the number of classified images each second. For the hardware deployment where storage space is limited, efficient ANNs must contain fewer parameters so that the trained weights can be saved. For platforms that support a limited amount of operations, computational efficiency also means that the ANNs include fewer (if any) special operations so that they can be easily deployed and efficiently executed by hardware accelerators (*cf.* Appendix D). Since deep learning for sparse 3D data is widely applied in real-time scenarios and deployed on edge devices, this thesis focuses on reducing the latency and improving the hardware-friendliness of neural networks for 3D data processing. It is achieved by analyzing and optimizing the architectures of networks. In addition, the computational cost and efficiency of the pre- and post-processing are also considered.

Label efficiency, which means neural networks can obtain good performance with fewer labels. One approach to achieving label efficiency is exploiting unlabeled data. In recent years, self-supervised pre-training has shown promising results in computer vision [13, 24, 31, 33, 35, 72, 82, 83] and natural language processing [46, 177]. The basic concept of self-supervised learning is to train a feature extractor (also known as the backbone) using unlabeled data in a pretext task, where the backbone has to solve a non-trivial problem (more details in Section 2.4). By doing this, the backbone is trained to capture informative features. Then, the pre-trained backbone can be adopted in different downstream tasks. To do so, the backbone is combined with a task-specific head (*e.g.*, a detection head or a classification head) and fine-tuned jointly using labeled data. With self-supervised pre-training, neural networks achieve better performance and faster convergence without additional labels. One critical advantage of self-supervised pre-training over other approaches, *i.e.*, semi-supervised learning [18, 210, 227, 233, 240, 266, 288], is that it decouples the pre-training and the downstream tasks. With a proper design, a pre-trained model can be easily transferred into multiple tasks [24, 31, 33, 72, 82, 83, 124]. On the other hand, semi-supervised learning methods are usually specialized for one single task, *e.g.*, image classification [18, 210] or object detection [227, 240, 266, 288]. Many methods for self-supervised pre-training with image data have been proposed recently. Although they can be adopted to 3D data, careful considerations are still necessary due to different data properties. Therefore, one focus of this thesis is to apply and improve self-supervised pre-training for 3D data and tasks.

Data efficiency. This property can be interpreted in two different ways. One interpretation emphasizes the total amount of training data. The data efficiency can be achieved by reducing the required data samples (as well as labels) as much as possible. In this context, humans are significantly more data efficient than ANNs. For instance, humans can learn a new category of objects with few data samples [58]. In contrast, orders of magnitude more samples are necessary to train ANNs for a comparable result. One shot learning [59, 237] and few shot learning [209, 220, 222] can be employed to reduce the total amount of required training data. Another interpretation of data efficiency is reducing the dependence on real-world data by using synthetic data for training. This approach is also efficient as synthetic data are less costly than real-world ones. This thesis follows the second interpretation and explores synthetic data generation for 3D computer vision tasks. While realistic synthetic 3D data can be inexpensively generated using the simulation technology [45, 71, 101, 251], a lot of human efforts and computational resources are still involved, *e.g.*, to create the simulation environment and perform rendering. Instead of generating data via simulation, this thesis explores the possibility of using a randomized method to generate data at a lower cost. Also, the generated synthetic data are utilized in self-supervised pre-training. This approach requires neither real-world data capturing nor manual data labeling to improve the performance of ANNs.

1.3 Overview of the Contents

Some fundamental knowledge on deep learning for sparse 3D data and research results related to this thesis are first revisited in Chapter 2.

Chapter 3 focuses on improving the computational efficiency of 3D object detectors for real-time applications. To this end, a depth map-based method is proposed, which extracts features directly from depth maps instead of point clouds converted from them. Thanks to the efficiency of a CNN-based backbone, the detector is four times faster than the point cloud-based baseline. Moreover, a novel convolution operation is proposed, which learns informative features from relative depth values instead of absolute ones. Experiments show that the new convolution operation significantly improves object detection results. Furthermore, this chapter also explores the supervised pre-training for the proposed 3D detector and achieves significant performance gains. However, supervised pre-training requires massive amounts of labeled data, which are costly and not always available in practice. It motivates the research on self-supervised pre-training in Chapter 4.

Chapter 4 explores the application of contrastive learning, a successful approach for self-supervised pre-training, to 3D data for better label efficiency. This chapter summarizes and systematically compares previous invariance-based contrastive learning methods in 3D computer vision. Meanwhile, a new method exploiting the format invariance between 2D (*e.g.*, depth maps) and 3D formats (*e.g.*, point clouds and voxels) of 3D data is proposed. Compared to previous methods, it achieves superior results in multiple downstream tasks without the dependence on additional information, *e.g.*, camera extrinsics and color channels. Also, the proposed method can be applied to neural networks with different architectures, *e.g.*, depth maps-based, points-based, or voxels-based networks.

While the depth map-based method in Chapter 3 has shown promising results, it is not applicable when the data cannot be represented as a depth map, *e.g.*, a point cloud reconstructed from multiple captures with different view angles or a fused point cloud from a multi-sensor system. To address this issue, Chapter 5 investigates transformers [49, 235] for point cloud understanding. Unlike many previous works designing transformers based on simple synthetic CAD models [62, 131, 163, 273, 279], this chapter focuses on real-world point clouds for better applicability in practice. It proposes using a shorter sequence length to reduce the computational cost. Also, some fundamental but long-overlooked components of transformers, *e.g.*, patchifiers and position embedding, are analyzed and optimized for better efficiency and performance. To pre-train transformers using self-supervision, the masked autoencoder [83], initially presented for image processing, is explored. Instead of applying the standard masked autoencoder, Chapter 5 proposes a simple method to overcome the information leakage caused by position embedding and significantly improves the effect of pre-training.

The proposed self-supervised methods in Chapters 4 and 5 boost the results in downstream tasks without using additional labels. However, they still rely on large-scale real-world datasets. To lift this limitation, Chapter 6 researches the randomized 3D scene generation for self-supervised pre-training. Instead of performing sophisticated simulation and rendering, this method randomly places 3D objects (*e.g.*, CAD models) based on pre-defined rules. Unlike previous works [179, 268], which focus on one downstream task, this chapter explores the generalization of the models pre-trained using randomly generated data. Specifically, it evaluates the pre-trained models for 3D object detection and semantic segmentation on multiple datasets. Also, the generated data are used for different pre-training methods, *e.g.*, contrastive learning and masked autoencoder. Furthermore, Chapter 6 proposes a novel approach to generating 3D objects using spherical harmonics. Since the method is fully rule- and formula-driven, the data generation process can be automated. Experimental results show that the generated data perform similarly to CAD models and real-world data.

Chapter 7 summarizes the contents and contributions of this thesis and discusses possible research topics for future works.

1.4 Assumptions and Conventions

Before presenting the main contents of this thesis, this section first introduces some assumptions and conventions to narrow the scope of the thesis and avoid confusion.

1. This thesis refers to data, tasks, or neural networks as 3D, as long as information on three *spatial* dimensions are involved. However, the meaning of dimension can be ambiguous. For instance, RGB images also have three dimensions, *i.e.*, height, width, and color. However, this thesis does not consider them as 3D data since only the former two dimensions are spatial while color channels belong to a spectral dimension. Similarly, ordinary videos are called 3D data in some contexts, as they consist of two spatial and one temporal dimension. Video data are also excluded from this thesis because they do not have three spatial dimensions.

2. Some medical and industrial data, *e.g.*, captured using computed tomography (CT), magnetic resonance imaging (MRI), and con-

focal microscopy, also have three spatial dimensions. However, unlike sparse point clouds from LiDAR sensors or depth cameras, these data are dense, *i.e.*, each spatial location contains meaningful measurement values. Such dense 3D data are excluded from this thesis because neural networks for dense and sparse 3D data are generally not transferable. While dense 3D data can be processed using standard dense 3D convolutional neural networks [4, 99, 189], sparse 3D data, *e.g.*, point clouds, require more specialized designs [39, 70, 169, 170, 215, 216].

Furthermore, since dense 3D data are excluded, the following contents use the term *3D deep learning* interchangeably with *deep learning for sparse 3D data* for simplicity and better fluency.

3. As mentioned in Section 1.1, there are a lot of tasks in deep learning for sparse 3D data. This thesis focuses on 3D object detection since it is widely applied, *e.g.*, in autonomous driving and robotics. Also, it is a representative task in research because it requires both semantic and spatial understanding of 3D data. However, to demonstrate the generalization of proposed methods, this thesis also discusses other essential tasks, *e.g.*, 3D semantic segmentation and 3D object classification.

4. This thesis assumes 3D measurements, *e.g.*, depth maps and point clouds, are already available. The measuring or estimating process to obtain the 3D data is beyond the scope of this thesis.

5. This thesis assumes that all data are captured at single points in time and are processed separately. Many depth cameras can capture RGB-D videos with a relatively high frame rate, *e.g.*, 30 FPS (frame per second). However, for 3D object detection and semantic segmentation methods in this thesis, each frame is processed independently. Jointly understanding continuous frames in a video clip is beyond the scope of this thesis. This choice has multiple reasons. First, the perception using time sequence is generally more computationally expensive. For instance, spatial-temporal convolution can be used to understand 3D videos [39], which has a higher complexity due to the extra temporal dimension. Therefore, in real-time applications, *e.g.*, autonomous driving, data frames

are usually processed independently [96, 117, 151, 251, 270]. Also, understanding measurements at single points in time is the base of sequence processing. Since 3D deep learning is a relatively new and under-explored research area [170], it is rational to first focus on more fundamental problems. Moreover, single-frame methods can be modified to process sequential inputs. For instance, object tracking using multiple frames can be achieved by extending a single-frame object detection pipeline [17, 60, 290].

6. The scenes considered in deep learning for sparse 3D data can be separated into indoor and outdoor scenes. The former are usually captured in working and living areas, *e.g.*, bedrooms and offices [43, 212]. The latter mainly contain traffic scenes for the usage in autonomous driving [22, 65, 219]. Due to different properties, *e.g.*, the scale and distribution of objects, of indoor and outdoor scenes, most previous works only focus on one of them [96, 117, 151, 152, 173, 203, 270]. Generally, indoor data, which can be captured using low-priced depth cameras, are easier to acquire than outdoor ones, which often require expensive measurement devices, *e.g.*, high-accuracy LiDAR sensors. This thesis focuses on indoor scenes for an easier evaluation of the proposed methods. However, previous works show that the methods developed based on indoor data can be transferred to outdoor ones, and vice versa [39, 76, 170–172, 260, 285]. Comparisons of methods for indoor and outdoor scenes are also presented in this thesis.

7. For simplicity and better fluency, the term *artificial neural network* is often shortened into *neural network* or even *network* in the following chapters. Also, following the convention in literature, it is used interchangeably with *deep learning model* or the shortened *model*, if it is not specified otherwise.

2 Related Works

This chapter introduces the fundamental knowledge and related research results of this thesis. Section 2.1 explains some basic architectures used in this thesis, *e.g.*, the multi-layer perceptron, the convolutional neural network, and the transformer. Then, Section 2.2 discusses neural networks for sparse 3D data. Furthermore, Section 2.3 represents some important tasks in 3D deep learning. Later, Section 2.4 introduces self-supervised pre-training and its applications in 3D computer vision. Finally, Section 2.5 presents technologies for synthetic data generation.

2.1 Basic Architectures of Neural Networks

Given a dataset $\left\{\left(\mathbf{u}^i, \mathbf{y}^i\right)\right\}_{i=1}^N$ which consists of N pairs of data sample $\mathbf{u}^i$ and label $\mathbf{y}^i$, an artificial neural network can be described as a non-linear function from a data sample $\mathbf{u}^i$ to a prediction $\hat{\mathbf{y}}^i$ (*i.e.*, an estimate of the label $\mathbf{y}^i$), parameterized with trainable parameters $\boldsymbol{\theta}$, also known as weights:

$$\hat{\mathbf{y}}^i = n\left(\mathbf{u}^i; \boldsymbol{\theta}\right) . \tag{2.1}$$

The data sample $\mathbf{u}^i$ in Equation 2.1 is abstracted as a vector. In practice, it can also be *e.g.*, an image, a point cloud, or a video clip and represented as a matrix or tensor. The label $\mathbf{y}^i$, also known as ground truth, represents the desired output of the neural network and contains different information depending on tasks. The network's output (*i.e.*, prediction) $\hat{\mathbf{y}}^i$ is an estimate of the label $\mathbf{y}^i$.

A neural network can be trained on dataset $\left\{\left(\mathbf{u}^i, \mathbf{y}^i\right)\right\}_{i=1}^N$ by optimizing the parameter $\boldsymbol{\theta}$, so that the averaged deviation between the prediction $\hat{\mathbf{y}}^i$ and the label $\mathbf{y}^i$ is minimized. The process can be formulated as follows:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} L\left(n\left(\mathbf{u}^i; \boldsymbol{\theta}\right), \mathbf{y}^i\right) . \tag{2.2}$$

Here, the loss function $L\left(\hat{\mathbf{y}}^i, \mathbf{y}^i\right)$ calculates the deviation between $\hat{\mathbf{y}}^i$ and $\mathbf{y}^i$ as a scalar (*i.e.*, loss). The larger the value, the greater the difference. Its concrete form depends on tasks and representations of labels and predictions. For instance, a norm function (*e.g.*, Euclidean norm) is commonly applied as the loss function in regression tasks, whereas the cross-entropy is widely employed in classification tasks [68]. The parameters $\boldsymbol{\theta}$ can be optimized via gradient descent. The algorithm performs numeric differentiation, calculates the gradients of loss with respect to $\boldsymbol{\theta}$ using back-propagation [118, 119] based on the chain rule, and iteratively updates the parameters until convergence. In the literature, calculating the prediction and loss is referred to as *forward pass*, while calculating the gradients is called *backward pass*.

However, the approach in Equation 2.2 requires all gradient information of N samples to be saved until the trainable parameters $\boldsymbol{\theta}$ are updated in each iteration. It is impossible in the case of a large dataset size N since the required storage space is extremely high. Instead, stochastic gradient descent (SGD) [120, 184] is applied as an approximation in practice. This approach randomly picks B samples with $B \ll N$ in each iteration (the B samples are referred to as a mini-batch) and updates parameters $\boldsymbol{\theta}$ based on the mini-batch instead of the entire dataset with N samples.

Till now, a neural network is merely represented as an abstract parameterized function $\hat{\mathbf{y}}^i = n\left(\mathbf{u}^i; \boldsymbol{\theta}\right)$. In the following subsections, some fundamental and general architectures of deep learning models are explained in detail. This section focuses on the neural networks applied for image processing since they inspire many models for 3D computer vision. Some well-known architectures, *e.g.*, recurrent neural networks (RNNs), are omitted since they are usually employed in natural language processing and are less related to this thesis. Neural networks specific to 3D data and tasks are introduced in Section 2.2.

Since all artificial neural networks contain trainable parameters, the vector $\boldsymbol{\theta}$ in notation $\hat{\mathbf{y}}^i = n\left(\mathbf{u}^i; \boldsymbol{\theta}\right)$ is omitted in following contents of this thesis for simplicity.

2.1.1 Multi-Layer Perceptron

The multi-layer perceptron (MLP) [188] is one of the earliest and most fundamental architectures of artificial neural networks. Each layer (or perceptron) in an MLP can be formulated as follows:

$$\mathbf{v} = \sigma\left(\mathbf{W}\mathbf{u} + \mathbf{b}\right) . \tag{2.3}$$

It describes a mapping from a vector $\mathbf{u} \in \mathbb{R}^{C_{\text{in}}}$ to a vector $\mathbf{v} \in \mathbb{R}^{C_{\text{out}}}$. The constants C_{in} and C_{out} are input and output channel numbers. The weight matrix $\mathbf{W}$ has the shape $C_{\text{out}} \times C_{\text{in}}$ and is multiplied by $\mathbf{u}$. The result is then added with a bias vector $\mathbf{b} \in \mathbb{R}^{C_{\text{out}}}$. The parameters in $\mathbf{W}$ and $\mathbf{b}$ are trainable, *i.e.*, can be updated via training. The activation function $\sigma(\cdot)$ is non-linear. It is necessary since neural networks are often applied to approximate highly non-linear functions, whereas $\mathbf{W}\mathbf{u} + \mathbf{b}$ performs a linear transformation to $\mathbf{u}$. Widely used activation functions are *e.g.*, the sigmoid function and the rectified linear unit (ReLU) [63], which are given by

$$v_i = \text{ReLU}\left(v_i^0\right) = \max\left\{0, v_i^0\right\} , \tag{2.4}$$

and

$$v_i = \text{Sigmoid}\left(v_i^0\right) = \frac{1}{1 + \exp\left(-v_i^0\right)} . \tag{2.5}$$

Note that activation functions are usually applied in an element-wise manner. That is, the result (*i.e.*, activation) is calculated independently for each element v_i^0 of the function's input vector.

The layer described in Equation 2.3 is often referred to as a fully connected layer (FC layer), since each element in the output $\mathbf{v}$ is influenced by all elements in $\mathbf{u}$. An MLP consists of multiple sequentially stacked FC layers, *i.e.*, each layer takes the output of its predecessor as the input. The first and last layers are called the input and output layers, respectively. Others are called intermediate layers or hidden layers. While the expression power of one FC layer is limited, an MLP can describe very sophisticated functions. Hornik *et al.* [91, 92] prove that an MLP with as few as one hidden layers is a universal approximator: it is capable of arbitrarily accurate approximation to an arbitrary function and its derivatives when its weights matrices contain sufficient parameters (*i.e.*, the intermediate layers have large enough channel numbers). Also, it is widely believed that a deeper MLP (*i.e.*, with more layers) has the equivalent expression power to a shallower one with more parameters [149].

Although MLPs are rarely applied as a standalone model in computer vision and natural language processing, they are widely integrated into other architectures, *e.g.*, as classification heads in CNNs (Section 2.1.2), as feed-forward networks in transformers (Section 2.1.3), and as feature projectors in PointNet [170] (Section 2.2.5).

2.1.2 Convolutional Neural Network

Convolutional neural networks (CNNs) [64, 119] are applied to process data with grid structures. They can be categorized according to the spatial dimension of processed data. CNNs for image processing, their original and most common application, are referred to as 2D. Also, 1D CNNs are widely applied for time-sequence analysis [41, 286], while 3D CNNs are often employed for medical data (*e.g.*, CT scans) [4, 99, 189] and voxelized point clouds [38, 39, 76]. Moreover, CNNs with higher dimensions have been explored for spatial-temporal analysis with point clouds [39]. Without loss of generality, this subsection introduces 2D CNNs as a representative since a generalization to other variants is straightforward.

2.1.2.1 Convolution Layer

Similar to MLPs, a typical CNN consists of multiple sequentially connected convolution layers, the essential component of which is the convolution operation. The input of convolution is a rectangular 2D feature map (2D grid), described as $\mathcal{U} \in \mathbb{R}^{H \times W \times C_{\text{in}}}$, where H, W, and C_{in} indicates the height, width, and channel number, respectively. Each discrete coordinate (i, j) with $i \in \{1, 2, \dots, H\}$ and $j \in \{1, 2, \dots, W\}$ is referred to as a pixel. The output of a convolution operation has the same height and width as the input, while the channel number C_{out} might differ from C_{in}. In a CNN, each convolution contains a trainable filter (also known as weight or kernel) $\mathcal{W}$ with shape $l_1 \times l_2 \times C_{\text{out}} \times C_{\text{in}}$. Without loss of generality, a square filter with $l_1 = l_2 = l$ is assumed for simplicity. The output $\mathbf{v}_{i,j}$ at an arbitrary pixel (i, j) is given by

$$\mathbf{v}_{i,j} = \sum_{m=1}^{l} \sum_{n=1}^{l} \mathbf{W}_{m,n} \mathbf{u}_{i+o(m,l),j+o(n,l)} \,. \tag{2.6}$$

Here, the transformation matrix $\mathbf{W}_{m,n} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$ is obtained by fixing the first two indices in the filter $\mathcal{W}$ to m and n, respectively. The function $o(m, l)$ defines an offset to the indices:

$$o(m,l) = -\left\lfloor \frac{l}{2} \right\rfloor + m \,. \tag{2.7}$$

Multiplied with matrix $\mathbf{W}_{m,n}$, each feature vector $\mathbf{u}_{i,j}$ in the feature map $\mathcal{U}$ is transformed from a C_{in}-dimensional space to a C_{out}-dimensional one. Such an operation is similar to the matrix multiplication in an FC layer. However, unlike an FC layer, whose output is globally connected to each element in the input (see Section 2.1.1), the output of a convolution operation at each pixel (i, j) is solely influenced by the input at the neighboring pixels. As defined by Equation 2.7, the neighborhood is a square region centered at pixel (i, j). The area of the neighborhood $l \times l$ is referred to as filter size or kernel size. Also, different pixels share the same filter $\mathcal{W}$. The convolution layer acts as a sliding window filter, which moves over each possible (i, j), *i.e.*, with stride 1.

In practice, the output of a convolution operation is often added with a trainable bias $\mathbf{b} \in \mathbb{R}^{C_{\text{out}}}$ and transformed by a non-linear activation, which can be described as follows:

$$\mathbf{v}_{i,j} = \sigma\left(\mathbf{b} + \sum_{m=1}^{l} \sum_{n=1}^{l} \mathbf{W}_{m,n} \mathbf{u}_{i+o(m,l),j+o(n,l)} \right) , \tag{2.8}$$

where $\sigma(\cdot)$ indicates an activation function, *e.g.*, ReLU (Equation 2.4). Like the convolution filter, the bias is also shared across all possible (i, j). The convolution operation, bias, and activation function compose a convolution layer.

When compared with an FC layer, the local connectivity and weight sharing are two distinguishing characteristics of a convolution layer. The local connectivity is motivated by an important assumption that neighboring pixels in an image have a stronger correlation with each other than distant ones. In machine learning and deep learning, such assumptions are referred to as *inductive bias*. Parameter sharing is applied to reduce the number of parameters and provide translation invariance. Such a concept is similar to filters in classical image processing (*e.g.*, Sobel filter, median filter). However, instead of being manually designed and hard-coded, the parameters in a convolution layer are automatically optimized via back-propagation.

A CNN can be obtained by stacking multiple convolution layers, which makes the deeper layers learn strong features and long-range dependency [119, 275].

2.1.2.2 Other Components

Besides convolution layers, modern CNNs contain other components, *e.g.*, down-sampling layers, up-sampling layers, skip connections, and batch normalization, which are briefly explained in the following:
Down-sampling layers. The motivation for down-sampling layers is to reduce the spatial resolution of feature maps and the computational cost. For instance, only one global feature is sufficient for image classification, and a higher resolution is unnecessary. Also, reducing the resolution helps the networks combine low-level features into high-level ones [68]. Commonly used down-sampling layers are the max pooling layer [119] and the strided convolution layer [214]. The former calculates a channel-wise maximum in small non-overlapping neighborhoods (*e.g.*, 2×2), and the latter performs a convolution operation with a stride greater than 1.
Up-sampling layers. Contrary to down-sampling layers, up-sampling layers increase the spatial resolution. They are often applied to down-sampled feature maps to obtain high-level features with high resolution [29, 182, 186]. However, they can also be employed independently of down-sampling, *e.g.*, for super-resolution tasks. Interpolation methods in classical image processing, *e.g.*, nearest-neighbor and bi-linear interpolation, are often used as up-sampling layers. In addition, the deconvolution layer [276], also known as transposed convolution, is widely used for up-sampling feature maps. It can be regarded as a trainable interpolating method.

Skip connection. The skip connection (also known as residual connection and identity mapping) [80] is motivated by the problem that very deep neural networks are difficult to train since they often lead to performance drops compared to shallower ones. Given a sub-network (*e.g.*, consisting of several convolution layers)

$$\mathcal{V} = n_{\text{sub}}\left(\mathcal{U}\right) , \tag{2.9}$$

a skip connection is obtained by directly adding its input to the output:

$$\mathcal{V} = n_{\text{sub}}\left(\mathcal{U}\right) + \mathcal{U}. \tag{2.10}$$

He *et al.* [80] demonstrate that CNNs with skip connections can be scaled up to considerable depth with consistent performance improvement. Also, skip connections can be integrated into networks other than CNNs, *e.g.*, transformers [235] and MLP-Mixers [230].
Batch Normalization. Batch normalization [102] is a technique to normalize feature maps in CNNs. In each mini-batch, it calculates the empirical mean and variance for each channel and normalizes (*i.e.*, re-centers and re-scales) each channel, respectively. Afterwards, a trainable linear transformation is applied to restore the representation power of the network. Batch normalization stabilizes and accelerates the training process by reducing the internal covariate shift [102]. In a CNN, batch normalization is often applied after each convolution layer.

2.1.3 Transformer

A transformer [235] is a sequence model originally proposed for natural language processing (NLP). Its input is a sequence of feature vectors and can be represented as a matrix $\mathbf{U} \in \mathbb{R}^{M \times C}$, where M and C indicate the number of vectors (*i.e.*, the sequence length) and the number of channels, respectively. In NLP, each vector corresponds to a word or sub-word [150] and is referred to as embedding or token. To extract informative features from the sequence, a language model should be capable of learning dependencies between the vectors. In a transformer, this is achieved by using the attention mechanism, which is given by:

$$\text{Attn}\left(\mathbf{V}^{\text{Q}}, \mathbf{V}^{\text{K}}, \mathbf{V}^{\text{V}}\right) = \text{Softmax}\left(\frac{\mathbf{V}^{\text{Q}}\left(\mathbf{V}^{\text{K}}\right)^{\top}}{\sqrt{C}}\right)\mathbf{V}^{\text{V}} . \tag{2.11}$$

Matrices (or sequence of vectors) $\mathbf{V}^{\mathrm{Q}} \in \mathbb{R}^{M_{\mathrm{Q}} \times C}$, $\mathbf{V}^{\mathrm{K}} \in \mathbb{R}^{M \times C}$, and $\mathbf{V}^{\mathrm{V}} \in \mathbb{R}^{M \times C}$ are referred to as *query*, *key*, and *value*, respectively. The output of attention has the same shape as $\mathbf{V}^{\mathrm{Q}}$ and can be interpreted as a new sequence.

The softmax function $\mathrm{Softmax}(\cdot)$ calculates a weighting matrix. When applied to an arbitrary matrix $\mathbf{V}$, it is defined by:

$$Z_{i,j} = \mathrm{Softmax}\left(\mathbf{V}\right) = \frac{\exp\left(V_{i,j}\right)}{\sum_k \exp\left(V_{i,k}\right)}, \tag{2.12}$$

where $V_{i,j}$ and $Z_{i,j}$ indicate an element in the input and the output matrix, respectively. Notice that the sum of each row in the output of $\mathrm{Softmax}(\cdot)$ equals one. Therefore, the softmax function in Equation 2.11 generates a weight matrix with shape $M_{\mathrm{Q}} \times M$. Multiplying the matrix by the value $\mathbf{V}^{\mathrm{V}}$ can be viewed as calculating weighted sums of the row vectors in $\mathbf{V}^{\mathrm{V}}$, where the weights are determined by the query $\mathbf{V}^{\mathrm{Q}}$ and the key $\mathbf{V}^{\mathrm{K}}$. Multiple attention mechanisms can be applied in parallel, and the outputs' channels can be concatenated to create a new sequence. Such an architecture is referred to as multi-head attention. The multi-head variant can replace the single-head attention to improve the representation power of transformers. Without loss of generality, this section explains the architecture of transformers assuming the single-head attention.

Depending on how the three matrices (*i.e.*, query, key, and value) are calculated, the attention mechanism has two variants: self-attention and cross-attention. In self-attention, each matrix is transformed from the same sequence of vector $\mathbf{U}$:

$$\mathbf{V}^{\mathrm{Q}} = \mathbf{W}^{\mathrm{Q}}\mathbf{U}\,, \tag{2.13}$$

$$\mathbf{V}^{\mathrm{K}} = \mathbf{W}^{\mathrm{K}}\mathbf{U}\,, \tag{2.14}$$

$$\mathbf{V}^{\mathrm{V}} = \mathbf{W}^{\mathrm{V}}\mathbf{U}\,. \tag{2.15}$$

The matrices $\mathbf{W}^{\mathrm{Q}}$, $\mathbf{W}^{\mathrm{K}}$, and $\mathbf{W}^{\mathrm{V}}$ apply a linear transformation, respectively and serve as trainable weights. With self-attention, vectors in the sequence $\mathbf{U}$ exchange information and generate a new sequence. In cross-attention, however, the query is transformed from another sequence $\mathbf{U}^{\mathrm{Q}}$:

$$\mathbf{V}^{\mathrm{Q}} = \mathbf{W}^{\mathrm{Q}}\mathbf{U}^{\mathrm{Q}}\,. \tag{2.16}$$

Therefore, a cross-attention fuses information from the two sequences and generates a new sequence with the same length as $\mathbf{U}^{\mathrm{Q}}$.

Another important component in transformers is the feed-forward network, which is a shared MLP independently applying a non-linear transformation on each vector in a sequence. A module consisting of one self-attention and one feed-forward network is often referred to as an encoder layer. It also contains layer normalization [11], a widely applied normalizing method for sequence models, and skip connections connecting the input and output of the self-attention and the feed-forward network, respectively. A transformer encoder can be established by stacking multiple encoder layers. Similarly, a transformer decoder consists of multiple decoder layers, each containing a self-attention, a cross-attention, layer normalization, and skip connections.

A typical NLP pipeline using a transformer can be described as follows. A paragraph is first embedded into a sequence $\mathbf{U}^0$ using word embedding. Then, a sequence $\mathbf{E}$ that encodes the position of each vector in $\mathbf{U}^0$, is added to $\mathbf{U}^0$:

$$\mathbf{U} = \mathbf{U}^0 + \mathbf{E}\,. \tag{2.17}$$

The sequence $\mathbf{E}$ is obtained via position embedding. However, a detailed introduction of word embedding and position embedding in NLP is beyond the scope of this thesis. The position embedding is necessary since attention is equivariant to the permutation of query $\mathbf{V}^{\mathrm{Q}}$ and invariant to the permutation of key $\mathbf{V}^{\mathrm{K}}$ and value $\mathbf{V}^{\mathrm{V}}$ [235]:

$$\mathrm{Attn}\left(\mathbf{P}\mathbf{V}^{\mathrm{Q}}, \mathbf{V}^{\mathrm{K}}, \mathbf{V}^{\mathrm{V}}\right) = \mathbf{P}\left(\mathrm{Attn}\left(\mathbf{V}^{\mathrm{Q}}, \mathbf{V}^{\mathrm{K}}, \mathbf{V}^{\mathrm{V}}\right)\right), \tag{2.18}$$

$$\mathrm{Attn}\left(\mathbf{V}^{\mathrm{Q}}, \mathbf{P}\mathbf{V}^{\mathrm{K}}, \mathbf{P}\mathbf{V}^{\mathrm{V}}\right) = \mathrm{Attn}\left(\mathbf{V}^{\mathrm{Q}}, \mathbf{V}^{\mathrm{K}}, \mathbf{V}^{\mathrm{V}}\right), \tag{2.19}$$

where $\mathbf{P}$ indicates an arbitrary permutation matrix that row-wise permutes another matrix. Without position embedding, the output of the encoder is equivariant to the order of sequence $\mathbf{U}^0$, which is undesired since the order of vectors (*i.e.*, order of words in a sentence) contains crucial information. For instance, permuting words in a sentence may completely change its meaning. With position embedding added, the encoder extracts high-level features from the input sequence. Then, a transformer decoder exchanges information from the feature and output sequence and iteratively updates the prediction in the output.

Transformer-based models are the de facto standard in NLP [46, 177]. In recent years, the vision transformer (ViT) [49] also shows promising results in image processing. A ViT first splits an image into multiple non-overlapping patches. A ViT treats such a patch analogously to the way a word is treated by a transformer-based method in NLP. The pixel values in each patch are vectorized (*i.e.*, stacked into a vector) and transformed into a high-dimensional space using a trainable transformation matrix. The transformation matrix (*i.e.*, patch embedding) is trained with the ViT. On the contrary, the word embedding in NLP is usually pre-trained [150]. Then, each vector is added with the position embedding of its corresponding patch. A ViT applies a transformer encoder (*i.e.* with self-attention) to extract features from the input sequence. Compared with CNNs, ViT and its variants have achieved similar or even superior performance in image processing tasks [24, 34, 49, 83, 124, 137, 257].

Recently, transformers are also applied to point cloud data. These methods are presented in Section 2.2.7.

2.2 Feature Learning from 3D Data

This section introduces the commonly used network architectures for 3D data. Most neural networks can be roughly separated into two parts: a backbone and a task-specific head. The backbone extracts features from input data, and its architecture is primarily determined by the data properties. For instance, CNNs are widely used as feature-extracting backbones for images, while transformers are often applied to texts. The head (also known as prediction head) utilizes features from the backbone and performs predictions. While a backbone's architecture can be shared across different tasks, the prediction head is usually task-specific. This section focuses on backbones (*i.e.*, feature extractors) in deep learning for sparse 3D data, while some critical tasks and task-specific designs are explained in Section 2.3.

2.2.1 Input Data

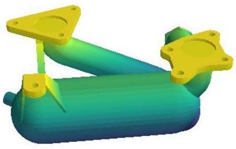
3D data can be represented in multiple ways. Before introducing the neural networks in deep learning for sparse 3D data, this subsection explains some important representations (or formats) of their input data. Some representative 3D data are illustrated in Figure 2.1.

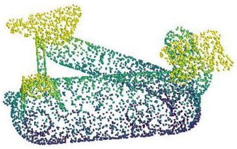
2.2.1.1 Point Clouds

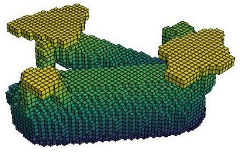
A point cloud consists of many sparse points, each given by a 3D coordinate $\mathbf{x}^i \in \mathbb{R}^3$. Therefore, a point cloud can be described as a point set $\{\mathbf{x}^i\}_{i=1}^N$, where N indicates the number of points. Each point might also contain extra features, *e.g.*, a color spectrum or a reflection ratio. In this case, each point can be described using a tuple $(\mathbf{x}^i, \mathbf{f}^i)$, where $\mathbf{f}^i \in \mathbb{R}^C$ represents a C-dimensional feature vector. A point cloud is then modeled as a set of tuples $\{(\mathbf{x}^i, \mathbf{f}^i)\}_{i=1}^N$.

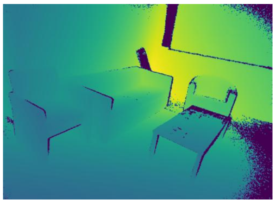
2.2.1.2 Depth Maps and Range Images

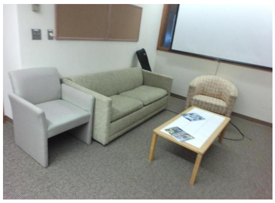
A depth map and a range image can be described as a function that maps a discrete coordinate (i, j) to a distance $d : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$. They can also be represented as a matrix $\mathbf{D}$. Despite the same description, their conventional meanings differ. Measurement results from depth cameras are often referred to as depth maps, while range images are captured using rotational LiDAR sensors. A depth map or a range image can be transformed into a point cloud. However, in range images, 3D coordinates are projected onto a spherical surface [56, 128, 151], whereas depth maps follow the pinhole camera model (*i.e.*, projection on a plane). Depth maps and range images can also be registered to other feature maps, *e.g.*, color images. The registered images, *e.g.*, RGB-D images, can be described as $d : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}^{C+1}$, where C is the channel number of extra features. Although depth maps and range images are 2D representations, they contain 3D spatial information and can be converted to point clouds. Due to this property, some literature refers to them as 2.5D data. In this thesis, they are categorized as 3D data for simplicity. More details about depth cameras and LiDAR sensors are provided in Appendix A.


(a) Mesh


(b) Point cloud


(c) Voxels


(d) Depth map


(e) RGB map

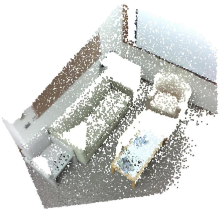

(f) Point cloud

Figure 2.1 Visualization of representative data in 3D computer vision. Mesh from PyVista library [217]. Depth and color map from the SUN RGB-D dataset [212].

2.2.1.3 Voxels

Voxels can be regarded as quantized point clouds. Each point in a point cloud $\{\mathbf{x}^i\}_{i=1}^N$ can be quantized (or voxelized) using a quantization function g_q

$$\mathbf{x}^{\mathrm{q},i} = g_\mathrm{q}\left(\mathbf{x}^i, l_\mathrm{q}\right) . \tag{2.20}$$

The length l_q is referred to as voxel size. One commonly used form of quantization function is based on rounding

$$x_j^{\mathrm{q},i} = g_\mathrm{q}\left(x_j^i, l_\mathrm{q}\right) = \left\lfloor \frac{x_j^i}{l_\mathrm{q}} + \frac{1}{2} \right\rfloor , \tag{2.21}$$

where $x_j^{\mathrm{q},i}$ indicates the j-th element in a quantized coordinate $\mathbf{x}^{\mathrm{q},i}$ with $j \in \{1, 2, 3\}$. Similarly, x_j^i is an element in the original coordinate. Voxels converted from a point cloud $\{\mathbf{x}^i\}_{i=1}^N$ can then be described as a 3D occupancy grid:

$$X_\mathrm{q}(l, m, n) = \begin{cases} 1 \text{ , } \exists \mathbf{x}^i : g_\mathrm{q}\left(\mathbf{x}^i, l_\mathrm{q}\right) = [l, m, n]^\top \\ 0 \text{ , otherwise} \end{cases} . \tag{2.22}$$

The discrete coordinate (l, m, n) is called a grid or voxel cell. If the original point cloud $\left\{(\mathbf{x}^i, \mathbf{f}^i)\right\}_{i=1}^{N}$ contains extra features, the value at each voxel cell is a feature vector with the same dimension:

$$X_{\mathrm{q}}(l,m,n) = \begin{cases} \mathbf{f}^{l,m,n} & , \exists \mathbf{x}^i : g_{\mathrm{q}}\left(\mathbf{x}^i, \mathrm{q}\right) = [l,m,n]^\top \\ \mathbf{0} & , \text{otherwise} \end{cases}, \tag{2.23}$$

where $\mathbf{0}$ indicates a zero vector and $\mathbf{f}^{l,m,n}$ is aggregated from all points $\mathbf{x}^i$ belonging to the voxel cell (l, m, n). Commonly applied aggregation functions are *e.g.*, the average pooling and random sampling.

2.2.1.4 Polygon Meshes

A polygon mesh (or mesh) consists of vertices, edges, and faces. Mathematically, it is often modeled as a graph. Unlike point clouds, which only contain independent sparse points, meshes are more informative and also describe the connections between vertices and surfaces. Meshes are widely applied to model 3D objects in computer graphics. However, 3D sensors usually provide depth maps, range images, or point clouds. Reconstructing meshes from real-world data is non-trivial [106]. Since this thesis focuses on applying deep learning to real-world data and tasks, meshes and their corresponding methods are not further discussed.

2.2.2 Classical Methods

These methods do not use deep learning and follow the spirit of classical image processing by using hand-crafted descriptors and statistical approaches to encode 3D shapes. For instance, extended Gaussian images [90] represent surfaces and shapes by binning surface normals. Ankerst *et al.* [7] propose 3D shape histograms for similarity search and classification. Kazhdan *et al.* [107] and Saupe and Vranić [197] describe 3D shapes using spherical harmonics. Rusu *et al.* [193] present fast point feature histograms for 3D registration. Generally, classical methods are outperformed by deep learning methods with a clear margin [38, 39, 170, 171]. However, they are suitable when training data are scarce. Also, they served as inspirations for a lot of deep learning approaches.

2.2.3 Multi-View-Based Methods

Multi-view-based methods [169, 215] are early attempts to apply deep learning to 3D shape recognition. They observe a 3D shape (usually a mesh) from multiple views and render color images from corresponding perspectives. Then, a 2D CNN is employed to recognize the rendered images. Recognition results from multiple views are fused via voting. These methods take advantage of mature 2D CNNs and save the effort of developing specialized neural networks for 3D data. However, their computational costs are high due to rendering and multiple inferences with CNNs. Also, the neural networks are unaware of the depth information since it is lost because of rendering. Also, multi-view-based methods underperform the deep learning methods directly using 3D data as inputs [170].

2.2.4 Voxel-Based Methods

Neural networks using voxels as inputs are referred to as voxel-based. As Section 2.2.1 explains, voxels or quantized point clouds have a 3D grid structure. Therefore, it is natural to apply 3D CNNs to voxels. However, early methods [26, 146, 169] suffer from unnecessarily high computational costs and memory footprints. The problem is caused by the sparsity of 3D spatial data. Unlike CT scanners, which can inspect the interior structure, distance sensors (*e.g.*, depth cameras and LiDAR sensors) can only capture measurement values on surfaces. Most voxel cells in real-world data, therefore, contain no meaningful features (*i.e.*, are empty) and are padded with zeros (*cf.* Section 2.2.1.3). However, early methods [26, 146, 169] apply dense 3D convolution to such sparse data, where the convolution filters "walk" through all voxel cells regardless of the features they contain.

Sparse convolution [39, 70, 216] is proposed to solve this issue. Rather than a new class of convolution, sparse convolution is an efficient implementation of dense convolution where the filter is only applied to non-empty voxel cells (*cf.* Equation 2.8). Combined with other sparse arithmetic operations, sparse convolution can be employed to build fully sparse neural networks, *i.e.*, sparse 3D CNNs, which are significantly more efficient than the dense variants. Sparse 3D CNNs have broad applications in 3D computer vision tasks. For instance, Gwak *et al.* [76] propose a generative sparse network for 3D object detection. Li [121] presents a fully convolutional network for fast object detection using sparse operations. Also, Choy *et al.* [39] apply sparse CNNs for semantic segmentation and explore higher-dimensional convolution for spatial-temporal data. Moreover, sparse 3D CNNs are also employed as feature extractors for registration tasks [38, 67].

2.2.5 PointNet and Its Variants

PointNet [170] is a milestone in 3D deep learning since it is the first deep learning method for direct point cloud understanding. On the contrary, concurrent works are based on voxels or multi-view input.

As Section 2.2.1 explains, a point cloud is a point set whose information is independent of the order of points. Such prior knowledge (or inductive bias) can guide the design of a point cloud-based neural network: its output should be independent of the input order. For a point set $\{\mathbf{x}^i\}_{i=1}^N$ and a neural network n, this property can be formulated as follows:

$$n\left(\mathbf{x}^1, \cdots, \mathbf{x}^m, \cdots, \mathbf{x}^n, \cdots, \mathbf{x}^N\right) = n\left(\mathbf{x}^1, \cdots, \mathbf{x}^n, \cdots, \mathbf{x}^m, \cdots, \mathbf{x}^N\right), \tag{2.24}$$

where $\mathbf{x}^m$ and $\mathbf{x}^n$ indicate two arbitrary points. Since the output does not change after two arbitrary points are permuted, the property is often referred to as permutation invariance.

To obtain permutation invariance, a PointNet first independently transforms each point into a high-dimensional space using a shared MLP. Then, a global max pooling is applied to compress the features into one vector:

$$\text{PointNet}\left(\{\mathbf{x}^i\}_{i=1}^N\right) = \text{MaxPool}\left(\text{MLP}\left(\mathbf{x}^1\right), \cdots, \text{MLP}\left(\mathbf{x}^N\right)\right), \tag{2.25}$$

where $\text{MaxPool}(\cdot)$ aggregates the max value in each channel from multiple feature vectors. Since each point undergoes the same (non-linear) transformation and the global pooling is independent of the input order, a PointNet is guaranteed to be permutation invariant. PointNet is also applicable to point clouds with extra features $\{(\mathbf{x}^i, \mathbf{f}^i)\}_{i=1}^N$. In this case, each feature vector $\mathbf{f}^i$ is concatenated with the corresponding coordinate $\mathbf{x}^i$ and transformed together by the shared MLP.

PointNet is a simple and effective architecture for point cloud understanding [170]. Also, the calculation with the MLP can be easily parallelized. However, the global pooling causes massive information loss, which makes it unsuitable for large-scale point clouds. Also, since it encodes an entire point cloud into one vector, it is unsuitable for tasks where dense features are necessary, *e.g.*, semantic segmentation, where each point must be semantically classified.

To address this problem, PointNet++ [171] hierarchically applies PointNets. Given a point cloud $\{\mathbf{x}^i\}_{i=1}^N$, it sub-samples M key points $\{\mathbf{s}^i\}_{i=1}^M$ using farthest point sampling [171]. This algorithm randomly picks an initial point and iteratively samples $M-1$ points, so that the distances between a new sample to all sampled points are maximized. The algorithm is also known as farthest-first traversal. Then, ball query is used to create M groups of points $\{\Omega_i\}_{i=1}^M$. Each group Ω_i is a small set consisting of points whose distance to key point $\mathbf{s}^i$ is within a pre-defined radius R:

$$\Omega_i = \left\{\mathbf{x}^j \,|\, R > \left\|\mathbf{x}^j - \mathbf{s}^i\right\|\right\} . \tag{2.26}$$

After points are grouped, PointNet++ encodes each group into a feature vector $\mathbf{v}^i$ using a shared PointNet:

$$\mathbf{v}^i = \mathrm{PointNet}\left(\Omega_i\right) . \tag{2.27}$$

Each feature $\mathbf{v}^i$ is registered to its corresponding key point $\mathbf{s}^i$, which results in a new point set $\left\{\left(\mathbf{s}^i, \mathbf{v}^i\right)\right\}_{i=1}^M$. The sub-sampling, ball grouping, and shared PointNet are referred to as a set abstraction module since it abstracts a point set into a smaller one. The concept of local connectivity (within each group) and shared weights (*i.e.*, the shared PointNet) is similar to CNNs. In PointNet++, such set abstraction modules are applied hierarchically to extract high-level features and reduce the resolution. For tasks requiring a high spatial resolution, PointNet++ uses interpolation to up-sample the point sets.

PointNet and PointNet++ inspire a lot of succeeding methods. For instance, DGCNN [247] follows the topology of PointNet++, but groups points in a feature space instead of using ball query in the Euclidean space. Also, RandLA-Net [96] applies random sampling instead of farthest point sampling for efficiency and replaces max pooling with attentive pooling for better performance. Moreover, HGNet [28] incorporates hierarchical spatial attention into PointNet++. Recently, PointNeXt [176] rethinks the architecture and training strategy of PointNet++ and makes the model stronger. Although these methods apply different sampling, grouping, and pooling algorithms, the concept of hierarchically using PointNet is unchanged. Therefore, they are referred to as PointNet variants. PointNet and its variants are widely applied in 3D deep learning as feature extracting backbones, *e.g.*, for shape recognition [170, 171, 247], object detection [28, 172, 173, 202], semantic segmentation [96, 176], and registration [8].

2.2.6 Point Cloud Convolution-Based Methods

Many works attempt to design convolution operations for sparse irregular point clouds, *i.e.*, point cloud convolutions. As explained in Section 2.2.5, the set abstraction module in PointNet++ [171] is similar to convolution due to the local connectivity and shared weights. However, a set abstraction (*i.e.*, shared PointNet) transforms each point in a group using the same MLP (*i.e.*, isotropic transformation). In contrast, a convolution applies different transformations on each pixel in a sliding window. Also, the standard convolution only applies to structured discrete grid cells (*cf.* Equation 2.6). It is unsuitable for point cloud data since the coordinates are irregular and continuous.

To distinguish point cloud convolutions with PointNet variants and the standard convolutions, this thesis considers a convolution-like operation as point cloud convolution if it has the following two properties: the transformation for each point feature is non-isotropic, and the operation is defined in a continuous space. For instance, kernel point convolution (KPConv) [228] defines rigid kernel points in 3D space with transformation matrices and applies transformations on each point depending on its Euclidean distance to the kernel points. Li *et al.* [125] propose a learnable X-transformation for weighting the point-wise features and permuting them into canonical order. Monte Carlo convolution [85] phrases convolution on sparse point clouds as Monte Carlo integration and implements it using Poisson disk sampling. Shi and Rajkumar [204] view a point cloud as a graph and apply graph convolutions for point cloud understanding. Wang *et al.* [241] present parametric continuous convolution, which exploits parameterized kernel functions in a continuous feature space to learn support relationship between points. Wu *et al*. [254] treat convolution filters as nonlinear functions of the local coordinates comprised of weight and density functions and learn them respectively using MLPs.

2.2.7 Transformer-Based Methods

The success of vision transformers [24, 34, 49, 83, 124, 137, 257] demonstrates that transformers, originally proposed for NLP tasks, are also strong models in computer vision. A lot of works follow the idea of ViT [49] and attempt to apply transformers to point clouds. However, the standard transformer layer depends on global attention and does not have locally constrained connectivity (*cf*. Section 2.1.3). On the contrary, *locality* is one of the most general inductive biases for designing neural networks in computer vision. It refers to the assumption that neighboring pixels or points are more strongly related. Due to the lack of inductive bias, training vision transformers requires more data and carefully designed strategies [34, 49, 257].

In order to improve results without increasing the amount of training data, previous works modify the architecture of transformers by employing local attention and down-sampling. The former constrains the attention mechanism in a local region to integrate local connectivity into transformers [54, 162, 164, 287]. The latter shortens the sequence in transformers to reduce the computational cost and improve the convergence since training with a longer sequence is generally more challenging. For instance, Hui *et al*. [100] perform hierarchical down-sampling to build a pyramid architecture for large-scale point clouds. Zhang *et al*. [278] down-sample the query for better efficiency. Guo *et al*. [75] apply PointNet++ [171] to encode raw point clouds and only use transformers to aggregate high-level features.

2.2.8 Projection-Based Methods

The methods introduced in Sections 2.2.5 to 2.2.7 all directly use 3D representations (*i.e.*, voxels and point clouds) as the input of neural networks. Generally, they have high time and space complexity due to the high dimensionality. The concept of projection-based methods is projecting sparse irregular point clouds onto a surface to obtain 2D representations of 3D information (*e.g.*, depth maps). By doing this, ordinary 2D CNNs can be applied to process 3D data for better efficiency.

Depending on how the point clouds are transformed, projection methods can be further separated into natural view-based and pseudo image-based methods. The former projects point clouds according to the principle of applied sensors. For instance, it projects point clouds from depth cameras, which can be described with a pinhole model, onto the image plane. Point clouds from a rotational LiDAR sensor are projected onto a spherical surface (*cf.* Appendix A.2). The projection and the obtained view are called natural since they can be achieved using a straightforward spatial transformation. Natural view-based methods have drawn some research interest. For instance, SequeezeSeg [251], SequeezeSegV2 [252], PointSeg [246], and RangeNet++ [151] exploit range images transformed from LiDAR point clouds for fast semantic segmentation in outdoor traffic scenarios. Furthermore, Li *et al.* [122] introduce a 2D fully convolutional neural network for 3D detection. Bewley *et al.* [19] propose range conditioned dilated convolution for 3D object detection on projected point clouds. Liang *et al.* [127] and Fan *et al.* [56] carefully optimize range image-based 3D object detectors for both speed and performance.

Pseudo image-based methods are primarily applied in outdoor traffic scenes and project point clouds onto the ground plane to gain pseudo images. This approach is feasible because LiDAR sensors for autonomous driving have relatively low resolution (*e.g.*, 64 or 128 lanes) in the vertical direction. Also, important objects, *e.g.*, pedestrians and vehicles, do not overlap vertically. Therefore, the projection does not lead to a severe information loss. The projection can be performed differently. For instance, PointPillars [117] splits a point cloud into vertical pillars and encodes them using a shared PointNet. VoxelNet [291] divides a point cloud into voxels and transforms points within each voxel into feature vectors, which are further encoded into a 2D feature map in bird-eye-view. Complex-YOLO [206] converts a point cloud into a bird-eye-view RGB map by encoding the height and density of points into pseudo colors. Also, PolarNet [283] transforms a point cloud into a polar bird-eye-view coordinate system and applies ring convolutions. Note that some works also refer to these methods as bird-eye-view-based. However, the naming may lead to confusion in the context of recent 3D object detectors that perform bounding box regressions in bird-eye-view using only RGB images as inputs [97, 133, 248]. Therefore, although pseudo image-based methods transform point clouds into bird-eye-view, this thesis does not refer to them as bird-eye-view-based to avoid ambiguity.

2.2.9 Summary and Discussion

The deep learning methods for images are already unified to a certain extend: CNNs [29, 80, 180, 182, 207] and transformers [49, 137] are the de facto standard. On the contrary, there is no dominant method in deep learning for sparse 3D data, though multi-view-based methods are rarely applied recently due to their low performance. The advantages and disadvantages of the commonly applied approaches are summarized as follows:

1. PointNet variants and point cloud convolution-based methods use unstructured point clouds as input. They can capture accurate spatial information since the point cloud directly represents 3D coordinates. However, Liu *et al.* [138] show that they usually suffer from dynamic kernels and random memory access, which leads to high computational costs and a slow speed. Also, they rely on special operations, *e.g.*, for sampling and grouping point clouds, which require a lot of engineering efforts for hardware deployment (*cf.* Appendix D). Furthermore, they usually contain more hyperparameters (*e.g.*, sample numbers for down-sampling [96, 171, 228], grouping radius [171], and distribution of kernel points [228]), which leads to high costs for parameter tuning.

2. Voxel-based methods, when implemented with sparse convolutions, are generally more efficient than PointNet variants and point cloud convolution-based methods [39] since the voxelization converts irregular point clouds into structured grids and simplifies the computation. Also, they can adopt mature architectures of 2D CNNs thanks to their similarity. Moreover, due to the property of convolution, they are applicable to voxels with arbitrary scale and resolution [76]. However, the voxelization causes a quantization error and information loss. Moreover, sparse convolutions, which heavily rely on hash maps [39], are technically more complex than

standard dense convolutions and are more challenging to deploy on hardware (*cf.* Appendix D).

3. Projection-based methods are most efficient and hardware-friendly, thanks to the simplicity of 2D CNNs and the mature technical optimization in software and hardware infrastructures [117]. However, they cause an inevitable information loss due to the projection, and their performance is limited by the indirect representation of 3D data. Therefore, they usually require careful designs to compensate for the performance gap [56]. Also, they are not applicable when a projection is impossible or causes severe information loss, *e.g.*, with point clouds reconstructed from multi-view inputs.

4. One distinguishing advantage of transformer-based models is that they are strong in capturing long-range dependency [136, 152]. Also, they have the potential to unify multi-modal inputs and multi-tasks [12, 110, 178, 181]. Moreover, as shown in Section 2.1.3, the standard transformer consists of only simple operations, *e.g.*, matrix multiplication, element-wise addition, and softmax, which makes it easy to parallelize. However, the standard transformer is still costly since its complexity is quadratic to the sequence length. Developing efficient transformers for 3D vision tasks is still an open research topic [164, 278].

There are also hybrid methods that combine multiple structures mentioned above to take advantage of different methods. For instance, Dai and Nießner [42] jointly train a multi-view- and voxel-based model. Liang *et al.* [127] consequently project a point cloud to the front view (*i.e.*, as a range image) and bird-eye-view (*i.e.*, as a pseudo image) for fast and accurate 3D object detection. Liu *et al.* [138] and Shi *et al.* [203] combine PointNet variants and voxel-based methods. Moreover, Mao *et al.* [144] apply transformers on voxels and project high-level features to bird-eye-view.

In this thesis, Chapter 3 presents a novel projection-based method, while Chapter 5 investigates the application of transformers to point cloud understanding. Moreover, in Chapters 4 and 6, some experiments use PointNet++ or sparse 3D CNN to compare the proposed pre-training methods with previous works.

2.3 Tasks in Deep Learning for Sparse 3D Data

Section 2.2 introduces the feature extraction from 3D data. This section explains how the learned features are utilized to perform tasks.

2.3.1 Object Classification

Object classification is the simplest and most fundamental task in deep learning for sparse 3D data. Given a data sample $\mathbf{u}^i$ (*i.e.*, a 3D object) and N_{cls} possible classes, a 3D backbone (see Section 2.2), often combined with global pooling, encodes the object into a global feature. The prediction is obtained by transforming the feature into an N_{cls}-dimensional space using an MLP and normalizing the result using a softmax function. The output is a probability distribution $\hat{\mathbf{y}}^i \in \mathbb{R}^{N_{\text{cls}}}$, where the j-th element y_j^i indicates the probability that the sample $\mathbf{u}^i$ belongs to the j-th class. The ground truth that the sample belongs to the k^i-th class is often encoded as a one-hot vector $\mathbf{y}^i$, where only the k^i-th element equals one while all the others are zeros. The one-hot vector can be considered as a probability distribution. The loss function in a classification task is the cross-entropy, which measures the difference between distributions $\hat{\mathbf{y}}^i$ and $\mathbf{y}^i$:

$$L_{\text{cls}}\left(\hat{\mathbf{y}}^i, \mathbf{y}^i\right) = -\sum_{j=1}^{N_{\text{cls}}} y_j^i \log \hat{y}_j^i \,. \tag{2.28}$$

For evaluation, the predicted class $\hat{k}^i$ is given by the index of the largest element in $\hat{\mathbf{y}}^i$:

$$\hat{k}^i = \underset{j}{\operatorname{argmax}} \left\{\hat{y}_j^i\right\} \,. \tag{2.29}$$

A commonly applied evaluation metric for 3D object classification is overall accuracy (OA), which is given by

$$\text{OA} = \frac{1}{N} \sum_{i=1}^{N} I\left(\hat{k}^i, k^i\right) \,, \tag{2.30}$$

where N indicates the total number of data samples. Function $I(\cdot,\cdot)$ equals one if the two arguments are identical. Otherwise, it is equal to zero.

In practice, 3D object classification is rarely performed as a standalone task. However, the performance in classification is an essential metric for designing and evaluating 3D backbones [54, 125, 169–171, 176, 215, 228, 241, 247, 254, 287] since it heavily relies on their capability of capturing informative features. Also, classification serves as a sub-task in other more complex tasks, *e.g.*, semantic segmentation and object detection.

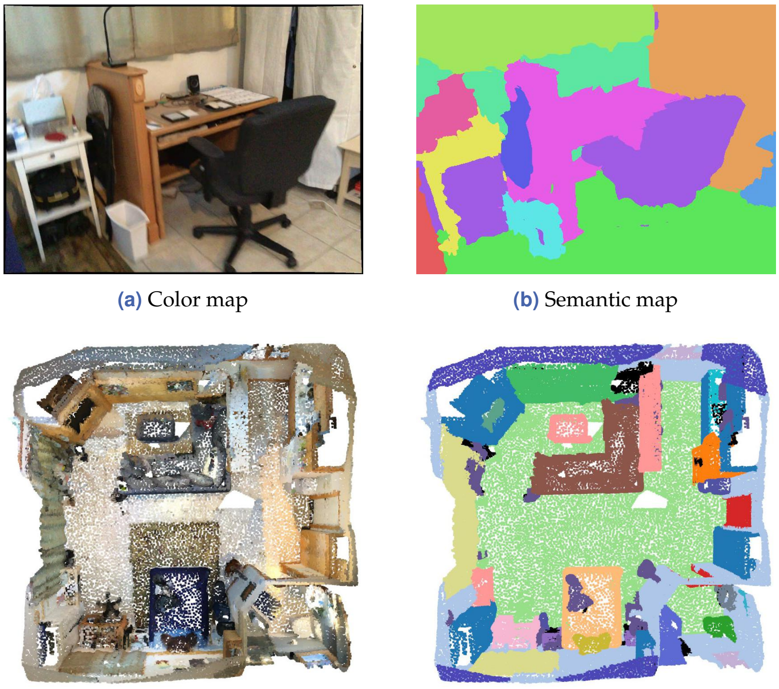
2.3.2 Semantic Segmentation

Semantic segmentation can be considered a dense classification problem, where each unit in a data sample, *e.g.*, each grid cell in voxels, point in point clouds, or pixel in images, must be classified. It is an essential task for environment understanding and is widely applied in autonomous driving and robotics [39, 96, 151, 246, 251, 252]. Also, the performance in semantic segmentation serves as a metric in designing and evaluating 3D backbones [138, 170, 171, 216, 228, 247]. Typical inputs and outputs in semantic segmentation tasks are visualized in Figure 2.2.

Most networks employ a U-shaped architecture to capture informative and high-resolution features, following the successful U-Net [186]. They consist of an encoder hierarchically down-sampling the features and a decoder progressively restoring the resolution via up-sampling. Also, skip connections are often applied between the encoder and decoder.

For CNNs, *e.g.*, voxel-based [39] or projection-based [251, 252] methods, establishing such an architecture is trivial. Down-sampling is usually accomplished using strided convolutions [214], where up-sampling is achieved using interpolations or transposed convolutions [276]. For methods directly applying point clouds as input, farthest point sampling [171, 176, 228, 247] and random sampling [96] are often used for down-sampling. A commonly applied method to up-sample point-wise features is the k-nearest-neighbor interpolation [171]. For each new point, it calculates a weighted sum of its k nearest neighbors depending on their Euclidean distances to the new point.

The captured high-resolution features are then transformed by a shared MLP and normalized to create point-wise (or voxel-wise, pixel-wise) predictions. During the training, a classification loss (*cf*. Section 2.3.1) is applied to each point and is averaged to calculate a final loss function. Commonly applied metrics for semantic segmentation are mean intersec-



(a) Color map (b) Semantic map

(c) Point cloud with color (d) Point cloud with semantic tags

Figure 2.2 Illustration of typical inputs (left) and outputs (right) of semantic segmentation tasks. Each color in the output indicates a unique class. Data source: ScanNet [43].

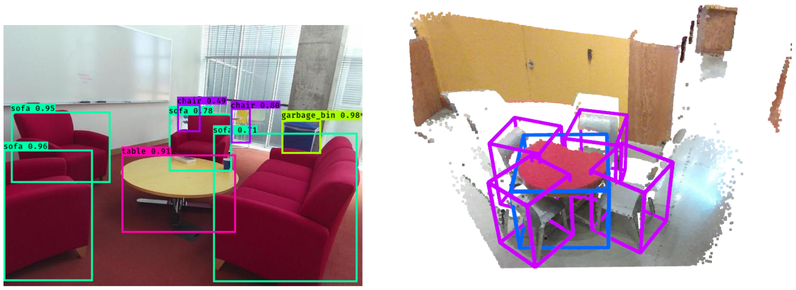
tion over union (mIoU) and mean accuracy (mAcc), which are explained in detail in Appendix C.1.

2.3.3 Object Detection

The general methodology of 3D object detection is first introduced. Then, some representative 3D detectors are briefly revisited since they are used as baselines in this thesis.

2.3.3.1 General Methodology

For object detection, a detector has to locate each object in a data sample and predict its size and class. Each object is indicated by a box around it (*i.e.*, bounding box), which is usually rectangular in images [180, 182] and cubic in 3D data, *e.g.*, point clouds and voxels.



(a) 2D and axis-aligned **(b)** 3D and rotated

Figure 2.3 Visualization of bounding box predictions in object detection tasks. The numbers indicate confidence sores of predictions. Data source: the SUN RGB-D dataset [212].

Generally, nine parameters are required to describe a 3D bounding box: the coordinate of its center point (three), its size (three), and the rotation around each axis (three). However, in many scenes, all objects are assumed to be placed on the ground or horizontal plains. Thus, bounding boxes are only rotated around the vertical axis. In this case, the degree of freedom of each bounding box is reduced to seven. Object detection in outdoor traffic [22, 65, 219] and indoor domestic scenes [212] often follows this assumption. Also, for scenarios where the orientation of bounding boxes is not important, the rotation can be completely neglected. The bounding boxes are referred to as axis-aligned in this case. Some representative bounding box predictions are shown in Figure 2.3.

Predicting the bounding boxes is often formulated as a regression problem (also known as bounding box regression). Also, each bounding box has to be classified. Therefore, object detection can be considered a combination of regression and classification tasks.

For object detection, a neural network first extracts high-resolution features. However, unlike semantic segmentation, the resolution does not have to be as high as the input. A lot of 3D detectors apply a U-shaped backbone [28, 56, 76, 117, 173, 203]. The extracted features can be represented as point clouds, voxels, or 2D feature maps. Generally, they can be described as high-level features with corresponding locations. For each location, a detection head has to predict bounding boxes close to it. To avoid predicting boxes at locations where no box exists (*i.e.*, false positives), the detection head also generates an objectness score for each box. A low score indicates that the probability that a box actually exists is low. Box predictions with low objectness scores can be removed in post-processing.

As explained, an object detector has to predict much information, *e.g.*, spatial parameters of bounding boxes, the objectness scores, and the class of boxes. Therefore, object detection usually applies multiple loss functions, *e.g.*, the cross-entropy for box classification and a norm function for box center regression, and is viewed as a multi-task problem. The total loss for back-propagation is a weighted sum of multiple sub-losses. For inference, the non-maximum suppression (NMS) is often applied to remove redundant bounding boxes [180, 182]. Mean average precision (mAP) is used to evaluate the quality of detection results (see Appendix C.2).

3D object detection is widely applied in autonomous driving and robotics. Also, it is essential for other higher-level tasks, *e.g.*, object tracking and instance segmentation. A lot of research efforts have been made for 3D object detection, *e.g.*, to optimize backbones [203], design stronger detection heads [136, 152, 173], fuse features from other domains [172, 174, 238], improve efficiency [56, 117, 127], and design better loss functions [128, 240, 289].

In this thesis, VoteNet [173] and 3DETR [152] are used as baseline methods for object detection. They are briefly introduced in the following.

2.3.3.2 VoteNet

VoteNet is a representative method of 3D object detection using indoor point clouds. One major difference of 3D object detection in indoor and outdoor scenarios is the scale of objects. Compared to the entire scene, outdoor objects are small (cars *vs*. streets). On the contrary, many indoor objects are large compared to the environment (beds *vs*. bedrooms).

Additionally, unlike 2D object detectors using dense images, 3D object detection often applies sparse data, *e.g.*, point clouds, as input.

For object detection tasks, features on each object have to be aggregated. The sparsity of 3D data and the large scale of indoor objects make the features hard to aggregate since features on the same object might be spatially distant. To solve this issue, VoteNet introduces a voting module, which predicts an offset from each point to its closest object center. Then, depending on the predictions, VoteNet translates all points toward their corresponding object centers. By doing this, points are pushed together toward object centers, and the feature aggregation can be performed effortlessly. The voting module is implemented as a shared MLP, which takes point-wise features as input and predicts the offset. It can be trained since the ground truth offset can be calculated using bounding box labels.

The remaining components of VoteNet are standard ones. It employs a U-shaped PointNet++ [171] as the backbone. The voting module predicts the offset and translates the point features. A prediction module aggregates the translated features using a set abstraction module (see Section 2.2.5) and predicts bounding boxes using an MLP. The regression loss from the voting module is optimized jointly with other sub-tasks in object detectors. Due to its simplicity and effectiveness, VoteNet inspires many successors for 3D object detection [28, 50, 98, 174, 259, 284].

2.3.3.3 3DETR

The detector 3DETR [152] is inspired by the successful DETR (DEtection TRansformer) [23] pipeline for 2D images. Instead of predicting dense results at each location [180, 182], DETR (as well as 3DETR) views object detection as a set prediction problem. Given a small set of learned object queries, it directly outputs the final set of predictions by aggregating information from the queries and the global context using a transformer decoder (*cf.* Section 2.1.3). It applies a set-based global loss that forces unique predictions via bipartite matching. Compared to previous object detection frameworks, DETR is simpler and contains fewer ad hoc designs.

3DETR expands the application of the DETR framework to 3D computer vision. It employs a transformer-only architecture and consists of a transformer encoder, which transforms a point cloud into patch-wise features (*cf*. Section 2.2.7), and a transformer decoder solving the set prediction problem. Thanks to the global perceptive field of transformers, capturing long-range dependencies is straightforward. Therefore, the voting module in VoteNet [173] is unnecessary for 3DETR.

2.3.4 Other Tasks

Besides object classification, semantic segmentation, and object detection, many other tasks are also extensively researched in 3D deep learning, *e.g.*,:

1. Data completion: Recovering lost information from corrupted point clouds, meshes, or voxels [166, 249, 274].

2. Point cloud denoising: recovering high-accuracy point clouds from noisy ones [142, 168, 277].

3. Registration: Building correspondence between multiple data samples and transforming them into one coordinate system [8, 39, 211].

4. Tracking: Detecting 3D objects in continuous frames or across different sensors and tracking their movements [27, 250, 272].

5. Instance segmentation: Detecting and delineating each unique object in the input [77, 93, 244]. The difference to object detection is that it indicates each object/instance with a mask, whereas object detection applies bounding boxes. It also differs from semantic segmentation since the latter does not distinguish objects/instances in the same class.

6. Panoptic segmentation: Combining semantic segmentation and instance segmentation tasks [61, 292]. Specifically, it performs semantic segmentation for background (*e.g.*, sky and floor) and instance segmentation for foreground objects.

However, since they are less relevant to the content of this thesis, a detailed introduction to these tasks is not provided.

2.4 Self-Supervised Pre-Training for Label Efficiency

This section first introduces the basic concept of self-supervised pre-training (SSP). Then, it revisits some important methods for SSP in image processing (Section 2.4.2, 2.4.3, and 2.4.4). Later, pre-training methods for 3D neural networks, inspired by their predecessors in image processing, are explained in Section 2.4.5. Finally, SSP is compared with semi-supervised learning, another option to improve label efficiency.

2.4.1 Basic Concept

Transfer learning is a well-known technique to improve the performance of deep learning models. In a standard pipeline, a feature extractor is first pre-trained on a large-scale annotated dataset (*e.g.*, ImageNet [44]) using a proxy task (*e.g.*, image classification). Later, the pre-trained weights are adopted to initialize a task-specific model (*e.g.*, an object detector), which is then fine-tuned on the target dataset (*e.g.*, the COCO dataset [129]). This approach is also known as supervised pre-training. As a result, the knowledge learned on a large generic dataset is transferred into the final task.

However, this pipeline requires both the pre-training and fine-tuning datasets to be fully annotated. On the contrary, most real-world data are unlabeled. The goal of self-supervised pre-training is to utilize unlabeled data in transfer learning and save the effort for data labeling. This approach is especially attractive in 3D computer vision since 3D data labeling is more laborious and time-consuming than the 2D counterpart.

Similar to the fully supervised pipeline, SSP also pre-trains a feature extractor (*i.e.*, backbone) using an artificial proxy task (also known as pretext task) and fine-tunes a task-specific model on the target dataset. However, the pretext task is elaborately designed so that no human annotation is necessary (see Section 2.4.2). Once a backbone is pre-trained, it can be applied to multiple downstream tasks.

The term self-supervised pre-training (SSP) is sometimes used interchangeably with self-supervised learning (SSL). Strictly speaking, the features learned in the proxy task do not necessarily have to be fine-tuned.

For instance, they can be directly utilized for classification problems via k-nearest-neighbor search [31, 72, 82]. In this case, SSL is a more suitable description. This thesis uses SSP instead of SSL because all its proposed methods follow the "pre-training + fine-tuning" pipeline.

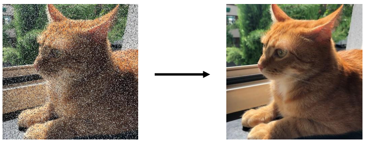
Moreover, SSP is referred to as unsupervised learning in some literature since human supervision (*i.e.*, labels) are not applied in the pretext task. However, this description can be misleading since the pretext task still requires supervision, although the ground truth can be automatically generated without human effort (see Section 2.4.2). Therefore, this description is avoided in this thesis.

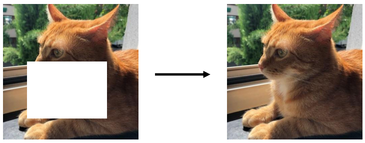
2.4.2 Early Methods

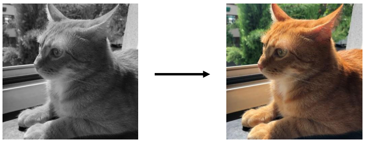
The key to self-supervised pre-training is designing an effective pretext task. First, the task must be challenging, so that the feature extractor is forced to learn informative features to solve the problem. Also, the task cannot rely on any human annotation, and the ground truth must be automatically generated.

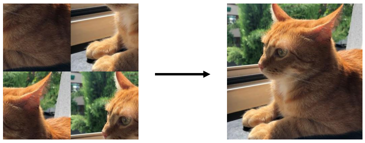
Many early methods apply autoencoders to generate a new image from the input. For instance, Vincent *et al.* [236] present denoising as a pretext task for self-supervised pre-training, where a neural network is trained to restore the original clear image from a copy with artificial noise (Figure 2.4(a)). Inspired by denoising, Xie *et al.* [258] remove continuous regions from images and train a deep learning model to reconstruct the missing information. This approach is referred to as inpainting and is illustrated in Figure 2.4(b). Moreover, an autoencoder for colorization [281, 282] is effective for pre-training with self-supervision. As shown in Figure 2.4(c), a neural network uses gray-scale images as input and is trained to predict the original color images.

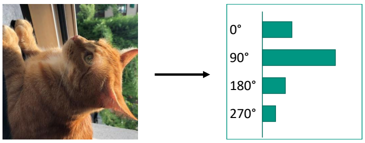
Instead of generating raw images, discriminative methods train neural networks to make abstract and discrete predictions. For instance, Noroozi and Favaro [157] train a model to solve a jigsaw puzzle. Specifically, each image is split into non-overlapping patches, and their positions are randomly permuted. A model is trained to restore the original image by predicting the correct order of the patches, as visualized in Figure 2.4(d). Moreover, Gidaris *et al.* [66] randomly rotate images by a multiple of 90° and train a model to predict the rotation angles (Figure 2.4(e)).


(a) Denoising


(b) Inpainting


(c) Colorization


(d) Jagsaw puzzle



(e) Rotation prediction (*i.e.*, predicting probability of each angles)

Figure 2.4 Concept of some pretext tasks for self-supervised pre-training.

All these pretext tasks are still supervised. However, the required ground truth, *i.e.*, uncorrupted images for denoising and inpainting, color images for colorization, patch order for the jagsaw puzzle, and rotation angles for rotation prediction, are obtainable without manual labeling. The methods introduced in this subsection have demonstrated that models can learn informative features using self-supervision. However, they are outperformed by later methods, *e.g.*, contrastive learning and masked autoencoder, which are explained in the following.

2.4.3 Contrastive Learning

In contrastive learning [24, 31, 33–35, 72, 82], a neural network is trained by contrasting features extracted from different sources. The basic concept of contrastive learning is illustrated in Figure 2.5. First, a random sample from a dataset is chosen as an anchor. A positive sample is created via data augmentation (transformations *e.g.*, color jitter, translation, scaling, and rotation). The anchor and the positive sample are often referred to as two *views* or a *positive pair*. Then, negative samples are sampled from the dataset, exclusive of the anchor. By doing this, the essential information in the positive sample and the anchor is similar, whereas negative samples are distinct from the anchor. An encoder (*e.g.*, a CNN with global pooling and an MLP head) encodes the anchor, positive sample, and negative samples into a feature, respectively. Contrastive learning aims to maximize the similarity between features from the anchor and positive sample while minimizing the similarity between features from the anchor and each negative sample.

The InfoNCE (information noise contrastive estimation) loss [160] is widely applied as the loss function in contrastive learning:

$$L_{\mathrm{InfoNCE}} = -\log \frac{\exp\left(\mathrm{sim}\left(\mathbf{v}^{\mathrm{a}}, \mathbf{v}^{\mathrm{p}}\right)/\tau\right)}{\exp\left(\mathrm{sim}\left(\mathbf{v}^{\mathrm{a}}, \mathbf{v}^{\mathrm{p}}\right)/\tau\right) + \sum_i \exp\left(\mathrm{sim}\left(\mathbf{v}^{\mathrm{a}}, \mathbf{v}^{\mathrm{n},i}\right)/\tau\right)}, \tag{2.31}$$

where $\mathbf{v}^{\mathrm{a}}$, $\mathbf{v}^{\mathrm{p}}$, and $\mathbf{v}^{\mathrm{n},i}$ indicate features from the anchor, positive sample, and negative samples, respectively. The function $\mathrm{sim}(\cdot,\cdot)$ measures the similarity of two vectors with the same dimension. A commonly used similarity function is the cosine similarity. The hyperparameter τ is called temperature and must be tuned. The objective of contrastive learning can be achieved by minimizing the loss L_{InfoNCE}.

In practice, multiple anchors (as well as their corresponding positive and negative samples) are re-sampled at each iteration. Consequently, the encoder learns a unique feature for each sample in the dataset, which is invariant to the data augmentation. Experiments show that the pre-trained encoder achieves competitive results in different downstream tasks [24, 31, 33–35, 72, 82].

Many well-known methods differ primarily in handling negative samples. For instance, SimCLR [31] contrasts samples within each mini-batch. Despite its simplicity, it relies on a very large batch size to avoid mode collapse, a hazard scenario where the features extracted from all data samples become similar. Training neural networks using a large batch requires a huge memory space, which limits the application of this method. To address this issue, Momentum Contrast (MoCo) [34, 35, 82] introduces a Siamese network structure, which consists of an encoder and a momentum encoder with the same architecture. The encoder is applied to each anchor, and the momentum encoder is to positive samples. During

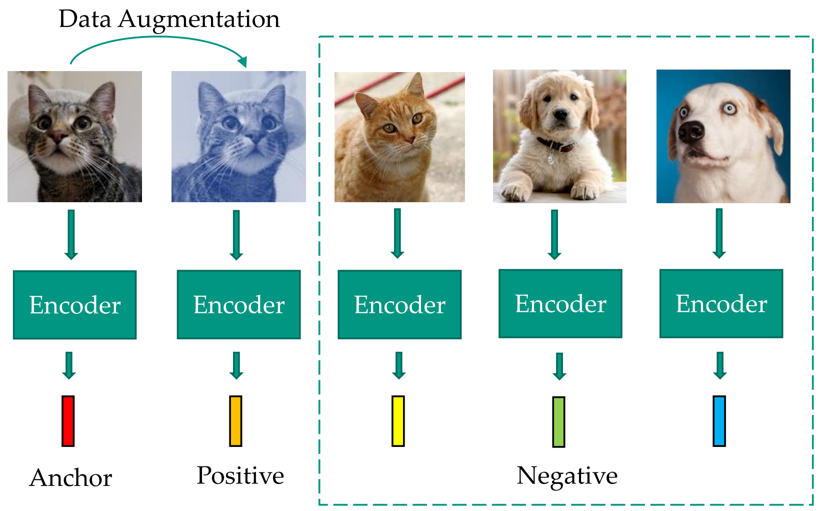


Figure 2.5 Concept of contrastive learning using global features.

training, only the encoder is updated via back-propagation. The weights of the momentum encoder are adopted from the encoder via exponential moving average (*i.e.*, with momentum). Also, MoCo uses a memory bank to save features from positive samples. Since anchors alter in each iteration, the positive samples from the past can be used as negative samples for the current iteration. The memory back is dynamically updated during training. Since a large number of negative samples can be kept in the memory bank, MoCo is less sensitive to the batch size than SimCLR. To further simplify the pipeline of contrastive learning, some methods only contrast anchors with positive samples and do not use negative samples [14, 24, 33, 72]. For instance, BYOL [72] and DINO [24] use an asymmetric structure with a momentum encoder. SimSiams [33] applies the stop-gradient operation instead of momentum. However, the reason why these methods do not lead to mode collapse is not fully understood. Recently, Bardes *et al*. [14] show that a symmetric structure is sufficient if features' variance, invariance, and covariance are regularized.

The concept shown in Figure 2.5 contrasts global features from each data sample (or instance). This pretext task is also called an instance

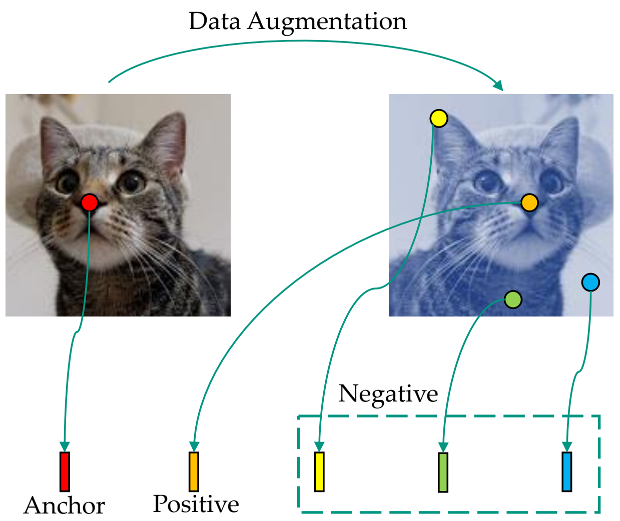


Figure 2.6 Concept of contrastive learning using local features.

discrimination problem in some literature. Besides global features, some methods also train local feature extractors using contrastive learning [25, 159, 245]. In this case, the neural network encodes each sample into dense features instead of a global feature (*i.e.*, the spatial resolution is reserved).

The concept of local (also called dense in some contexts) contrastive learning is shown in Figure 2.6. For each image, a new view is created via data augmentation. Then, an encoder encodes each view into a feature map, respectively. Each local feature can be considered an anchor. In the other view, the corresponding local feature is its positive sample, whereas features at all the other locations are negative samples. Therefore, contrastive learning can train the encoder using local features and correspondences. Local contrastive learning is applicable to downstream tasks requiring dense features, *e.g.*, semantic segmentation and object detection.

2.4.4 Masked Autoencoder

Masked autoencoder (MAE) [83] is a self-supervised generative approach[1] for pre-training vision transformers (ViT) [49]. As shown in Figure 2.7, an MAE splits an image into non-overlapping patches, following a standard ViT. Then, it randomly masks a large proportion (*e.g.*, 75 %) of the input patches. The reserved patches are embedded using a linear transformation to create a feature sequence. Also, the position embedding of the patches is added. An encoder then extracts features from the sequence. The encoder has a standard ViT structure consisting of multiple transformer layers with self-attention (*cf*. Section 2.1.3).

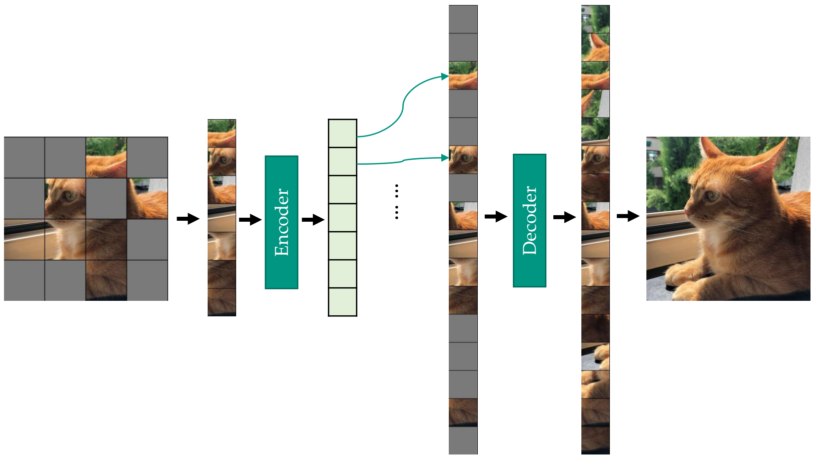


Figure 2.7 Concept of self-supervised pre-training using masked autoencoder.

The objective of masked autoencoder is to restore the masked patches using the visible ones. To this end, a full-length sequence is created by padding a shared learnable mask token into the encoder's output. Also, the positions of masked patches are embedded and injected into the corresponding location in the sequence. Then, a decoder, consisting of

[1] The term MAE has two meanings in the literature. Usually, it refers to a *pre-training method*. In some contexts, it also refers to the *network architecture* the method uses. This thesis follows this convention.

multiple transformer layers with self-attention, is trained to reconstruct the pixel values in each masked patch. He *et al.* [83] show that a decoder with fewer layers than the encoder is sufficient for pre-training. The loss function is calculated by averaging the distance between reconstructed patches and the original ones, *e.g.*, using pixel-wise Euclidean distance. During the pre-training, the encoder learns the relation between reserved patches and describes the patches with informative features. The decoder is abandoned in downstream tasks, and the encoder can be applied as a pre-trained feature extractor.

MAE achieves impressive performance and is significantly faster than previous methods, *e.g.*, contrastive learning, since the encoder uses a shorter sequence as input and the decoder is light-weight [24, 34, 83]. Also, it is simpler than the previous method BEiT [13] inspired by the well-known BERT framework [46] in natural language processing, although both methods reconstructed missing information from masked images.

2.4.5 Self-Supervised Pre-Training in 3D Vision

Inspired by the fast progress in self-supervised pre-training using images, many methods have been proposed to apply this technique to 3D data. For instance, Achlioptas *et al.* [2] present an autoencoder for point clouds, which learns to reconstruct its input from a bottle-necked latent space. Hassani and Haley [79] train a multi-scale graph-based encoder using multi-tasks including clustering, reconstruction, and self-supervised classification. Moreover, Sanghi [195] applies contrastive learning on 3D shapes and maximizes the mutual information. Sauder and Sievers [196] define a pretext task where a neural network is trained to restore point clouds whose parts have been randomly placed. Also, Wang *et al.* [239] reconstruct complete point clouds from occluded single-view ones.

However, these methods are all evaluated on synthetic datasets [26, 255]. PointContrast [260] is the pioneer in applying self-supervised pre-training to real-world point clouds. Using contrastive learning, it learns point-wise correspondences between two partially overlapping point clouds captured from different view angles. Inspired by PointContrast, Liu *et al.* [134] exploit the correspondence between aligned point cloud-image pairs and pre-train a 3D encoder by contrasting its output with a pre-trained 2D encoder. Also, Liu *et al.* [135] propose contrastive learning with pairs of point-pixel pairs. Moreover, Hou *et al.* [94] improve PointContrast using spatial partition and investigate the data and label efficiency of pre-trained models. Zhang *et al.* [285] extend the successful MoCo [35, 82] pipeline to the 3D domain and exploit the cross-format contrast between point clouds and voxels. Furthermore, Chen *et al.* [36] utilize dynamic spatial-temporal correspondence and explore 4D contrast in pre-training.

Instead of using contrastive learning, some works train neural networks by solving masked point cloud modeling problems, where the masked proportion of point clouds has to be restored. For instance, PointBERT [273] predicts the missing tokens from corrupted point clouds. Also, POS-BERT [62] augments PointBERT with a momentum tokenizer and a contrastive loss. Recent works apply MAE [83] to point clouds. For instance, Point-MAE [163] straightforwardly predicts the coordinates of masked points following MAE. Point-M2AE [279] extends the MAE pipeline to hierarchical multi-scale networks. Instead of reconstructing raw point clouds directly, MaskPoint [131] learns an implicit representation to avoid information leakage.

Rather than learning features using 3D data, some methods attempt to directly transfer knowledge from other domains, *e.g.*, languages and images. Zhang *et al.* [280] use a pre-trained CLIP model [280] to guide a point cloud encoder. Xu *et al.* [264] show the possibility of initializing a 3D CNN using weights of a pre-trained 2D CNN. Qian *et al.* [175] follow this idea and explore initializing a point cloud transformer with a pre-trained ViT [49]. Although they do not belong to self-supervised learning, these methods also improve label efficiency (especially in the case of 3D labels). They are introduced in this subsection for easier comparison with self-supervised methods.

2.4.6 Comparison with Semi-Supervised Learning

Semi-supervised learning assumes that only a proportion of training data are annotated and trains neural networks on both labeled and unlabeled data [233]. In this context, the pipeline "self-supervised pre-training + supervised fine-tuning" can be considered a semi-supervised learning approach. However, to avoid ambiguity, self-supervised pre-training is excluded from semi-supervised learning in this thesis.

One representative method of semi-supervised learning is self-training. It is a progressive approach where a neural network is first trained exclusively on labeled data. The predictions on unlabeled data are used as pseudo labels further to update the neural network [18, 210]. Other approaches for training neural networks using semi-supervision include *e.g.*, co-training, boosting, maximum-margin, and perturbation-based methods [233]. Although many works apply semi-supervised learning for classification, there are also attempts for other tasks, *e.g.*, object detection with images [227, 266] and point clouds [240, 288].

The limitation of semi-supervised learning is that it is specialized for one single task (*i.e.*, whose labels are partially available). A semi-supervised learning pipeline must be redesigned and retrained for different tasks (*i.e.*, different types of labels). On the contrary, self-supervised pre-training is decoupled from the fine-tuning tasks. For instance, a CNN pre-trained on a large-scale image dataset can be freely transferred into classification, detection, and semantic segmentation tasks [24, 35, 82, 83]. Because of its better flexibility and generalizability, this thesis chooses self-supervised pre-training instead of semi-supervised approaches to improve label efficiency.

2.5 Data Generation for Data Efficiency

Even though laborious data labeling can be avoided using self-supervised pre-training or semi-supervised learning, capturing real-world 3D data is still costly. Synthetic data can be applied to address this issue. Specifically, neural networks are pre-trained on synthetic data and then fine-tuned on real-world data for downstream tasks. By doing this, the neural networks gain better performance in downstream tasks without using additional real-world data. The methods for generating synthetic data can be split into simulation-based and randomized methods.

2.5.1 Simulation-Based Methods

Thanks to the progress in computer graphics and optics, photo-realistic data can be captured via simulation. These methods first require a simulation environment, which models the behavior of objects and sensors in the real world. The technique of transferring knowledge from a simulation environment to the real world is often referred to as sim2real. Sim2real is widely applied in 3D deep learning. For instance, Deschaud *et al*. [45] employ the CARLA simulator [48], an open source software to simulate the urban environment, to generate synthetic LiDAR point clouds for 3D mapping tasks. Griffiths and Boehm [71] present a large-scale point cloud of the urban environment using BlenSor [73], a package modeling various range sensors based on the open source project Blender. Xiao *et al*. [256] collect point cloud data using Unreal Engine 4 (UE4), a commercial software for developing 3D video games, and investigate the transfer learning from synthetic to real-world data in semantic segmentation. Also, Wu *et al*. [251] and Hurl *et al*. [101] synthesize training data for autonomous driving using Grand Theft Auto V, a popular video game.

Meanwhile, sim2real is also studied for industrial applications. For instance, Bäuerle *et al*. [15] apply Blender to render training data for 3D pose estimation of electronic control unit housings. Moreover, Wu *et al*. [253] use BlenSor to render point clouds of electric motors for autonomous disassembly and train deep learning models for semantic segmentation.

Although generating synthetic data via simulation is significantly less expensive than capturing data in the real world, developing the simulation environments, crafting source materials, and designing the scene layouts still require considerable efforts.

2.5.2 Randomized Methods

Instead of creating vivid scenes in a simulation environment, randomized methods simplify the data generation process with randomness. This approach generates 3D scenes by randomly placing "objects", which can be either CAD models [179] or formula-driven shapes [268], based on predefined rules. Specifically, Rao *et al*. [179] generate scenes by randomly placing CAD models in randomly created rooms. They create pairs of scenes with the same objects but different layouts and perform contrastive learning using object-level correspondence. Yamada *et al*. [268] employ fractal point clouds as objects instead of CAD models. This method generates bounding box labels for each object and pre-trains object detectors in a fully supervised manner. Both methods show promising performance in object detection tasks, compared with models pre-trained on real-world data.

3 Real-Time 3D Object Detection using Depth Maps

3D object detection is an essential task in 3D computer vision. Most existing 3D object detection methods take point clouds (despite quantized or not) as input, even when each point cloud is converted from a single depth map. Although they have achieved impressive performance, point cloud-based 3D detectors usually have high computational costs and complex structures, which limits their application in real-time scenarios. Following the state-of-the-art VoteNet [173], this chapter presents 2.5D-VoteNet, a powerful and efficient depth map-based 3D detection pipeline. Since the proposed models extract features directly on depth maps, most computation remains in 2D space and can be efficiently executed. Furthermore, instead of using an off-the-shelf 2D CNN, they apply relative depth convolution (RDConv) to learn robust local features. The end-to-end pipeline achieves state-of-the-art results on the challenging SUN RGB-D [212] benchmark and surpasses the baseline with a clear margin on the ScanNet [43] frame-level detection task. Meanwhile, this method reaches a significantly higher inference speed than previous ones (69 frames per second). The main contents of this chapter have been published in [294].

3.1 Introduction

As explained in Section 2.2, 3D spatial data can be represented in various ways. According to the applied data, 3D object detection methods can be roughly categorized into projection-based [32, 115, 117, 121, 206, 291] and point cloud-based methods [28, 39, 76, 93, 113, 172–174, 203, 204]. Projection-based methods project point clouds into bird-eye-view, front view, or multi-views (*cf*. Section 2.2.8). By doing this, point clouds are converted into images so that mature 2D CNNs can be applied. However, they are rarely used in indoor scenarios because objects are more cluttered than outdoors. Point cloud-based methods use raw or quantized (*i.e.*, voxelized) point clouds as input. To aggregate local and global features from point clouds, they often hierarchically down-sample the input and perform convolution-like operations (*cf.* Sections 2.2.4, 2.2.5, and 2.2.6). Although they have achieved promising performance, point cloud-based methods cannot achieve high inference speed. The state-of-the-art methods for indoor scenes have a frame rate of $\sim$10 FPS (frame per second) [28, 50, 173, 174, 259, 284] and cannot fulfill the requirement for some real-time applications[1]. Moreover, the high computational cost and special self-defined operations in point cloud-based methods (*e.g.*, PointNet++ [171], sparse 3D convolution [76]) prohibit hardware implementations on a lot of mobile or small devices (*e.g.*, cell phones, small robots, and drones) since they support solely efficient processors with either less computational resources (*e.g.*, ARM CPU and mobile GPU) or limited operation support (*e.g.*, neural ASIC).

The proposed method is motivated by the observation that in many real-time applications a point cloud is converted from a single depth map. Given the camera calibration, depth maps and point clouds are different representations of the same information. Thus, it should be possible to perform 3D object detection directly using depth maps instead of point clouds.

This chapter presents a novel pipeline that captures features on depth maps while keeping the detection head and post-processing in 3D space. Unlike existing 2D CNNs, which use absolute depth as input, the backbone of the proposed networks is augmented with novel relative depth convolution (RDConv), which learns local features from relative depth. The RDConv is motivated by the simple intuition that local geometries (*e.g.*, edges and corners) depend more on relative depth and are invariant to the absolute depth. Compared to the standard convolution using absolute depth, RDConv extracts more informative features and improves the detection result. The proposed pipeline shows higher inference speed

[1] Conventionally, a frame rate higher than 30 FPS is regarded as real-time.

and better hardware-friendliness than point cloud-based methods while reaching competitive performance.

The contributions of this chapter are as follows:

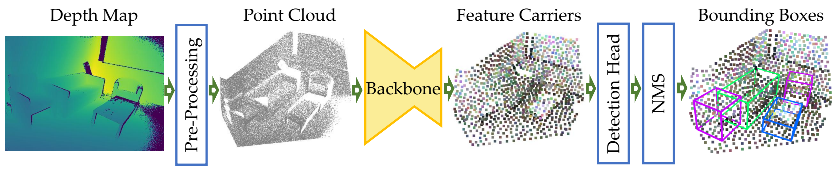
1. It introduces a simple and efficient depth map-based 3D detection pipeline.
2. It represents the novel approach of relative depth convolution, which effectively captures local information on depth maps.
3. The proposed models achieve state-of-the-art results on the challenging SUN RGB-D [212] benchmark and are significantly faster than existing methods. Also, they outperform the baseline [173] in the ScanNet [43] frame-level detection with a clear margin.

3.2 Depth Map-Based 3D Object Detection

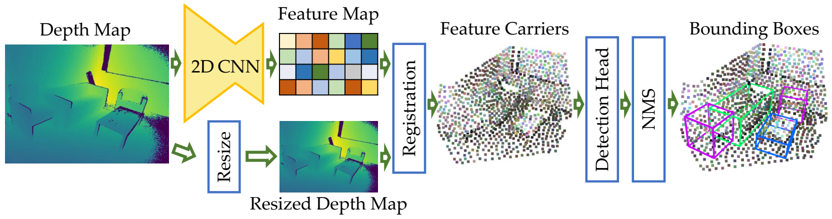
In this section, the pipeline of point cloud-based 3D detection is first revisited (Section 3.2.1). Then, a computationally efficient depth map-based pipeline is introduced (Section 3.2.2). Later, an important ingredient of the pipeline, relative depth convolution (RDConv), is presented in Section 3.2.3.

3.2.1 Point Cloud-Based Pipeline

The standard workflow of a point cloud-based 3D object detector can be separated into four components: the pre-processing, a 3D backbone, a detection head, and the post-processing. As illustrated in Figure 3.1(a), depth maps are converted into point clouds at the pre-processing stage. Usually, point clouds are further down-sampled or quantized to reduce the computational cost. The backbone then applies sparse convolution (Section 2.2.4), PointNet variants (Section 2.2.5), point cloud convolution (Section 2.2.6), or transformers (Section 2.2.7) to the sparse coordinates. The output of the backbone is a down-sampled subset of the input point cloud with high-level features attached to each point (also called feature carriers or feature points). The detection head then aggregates object-relevant information and performs bounding box regression



(a) Point cloud-based



(b) Depth map-based (proposed)

Figure 3.1 3D object detection pipelines using a single depth map as input.

and semantic classification. In the post-processing stage, non-maximum suppression (NMS) is applied to remove redundant predictions. More details on 3D object detection is provided in Section 2.3.3.

3.2.2 2.5D-VoteNet: a Depth Map-Based Pipeline

The proposed depth map-based pipeline is illustrated in Figure 3.1(b). A U-shaped 2D CNN encodes a high-resolution depth map into a low-resolution feature map. It means the feature aggregation occurs in the 2D space rather than the 3D space, significantly reducing computational costs. However, to fully utilize the spatial information, the feature carriers, the detection head, and the NMS remain in the 3D space. To build feature carriers, the input depth map is scaled to the same resolution as the feature map and lifted into the 3D space. Then, each feature vector in the feature map is registered to the corresponding 3D point. Since feature carriers are in the 3D space, they can be processed as in a standard point cloud-based pipeline. The proposed pipeline adopts the detection head with the voting module and the post-processing from VoteNet [173].

Thus, the model is named 2.5D-VoteNet. More details on the baseline method VoteNet are presented in Section 2.3.3.2.

Compared to point cloud-based methods, the presented depth map-based pipeline brings multiple advantages. First, the network is accelerated thanks to the efficiency of the 2D CNN. Moreover, the low computational cost allows a deeper backbone and higher-resolution input, which results in more informative features and better detection quality. Point cloud-based networks usually have limited model depth and only accept down-sampled or voxelized point clouds to avoid excessive run-time and memory usage. Also, the simple architecture simplifies the hardware implementation in real-world applications (*cf*. Appendix D). Meanwhile, the pipeline simplifies the fusion of geometrical and color information since the depth map and color map have 2D grid structures and can be scaled to the same resolution. On the contrary, it is non-trivial to incorporate RGB images into the point cloud-based pipeline due to different data properties (*e.g.*, regularity *vs*. irregularity, high resolution *vs*. low resolution), as discussed in previous works [93, 174, 238]. Specifically, 2.5D-VoteNet gains a significant performance boost by using only one additional layer to accept RGB images. This fusion strategy is further explained in Section 3.4.3.3.

One concern of the depth map-based 3D detection is that the 2D backbone is unaware of the camera calibration. However, experimental results imply that the models learn calibration invariance via data augmentation (see Section 3.4.3.7). The limitation of this method is that it cannot directly handle 3D scans that are reconstructed from multiple views of depth maps (*cf*. Section 2.2.9). However, it is not a detrimental problem in real-time applications since single-view point clouds are usually used as input in this case.

3.2.3 Relative Depth Convolution

This chapter introduces a novel convolution operation that depends on relative rather than absolute depth. The intuition behind this design is trivial: local geometries (*e.g.*, edges and corners) rely more on relative depth than absolute depth. The observation implies that absolute depth invariance can help a 2D CNN to capture more informative and robust features on depth maps.

Following this intuition, the proposed relative depth convolution (RD-Conv) normalizes the depth values in each sliding window with respect to a local reference depth. The new convolution negligibly increases the computational cost and can replace the first convolution layer in a standard 2D CNN without changing its overall structure.

The standard convolution operation is introduced in Equation 2.6. Following the notation, a depth map (*i.e.*, input) can be described as a matrix $\mathbf{D}$ with shape $H \times W$, where H and W indicate the height and width of the depth map, respectively. The output feature map can be represented as a tensor $\mathcal{V}$ with shape $H \times W \times C$, where C indicates the channel number. Let (i, j) be an arbitrary 2D coordinate. The depth value and feature vector at (i, j) are $D_{i,j}$ and $\mathbf{v}_{i,j}$, respectively. The RDConv with kernel size $l \times l$ is defined as

$$\mathbf{v}_{i,j} = \sum_{m=1}^{l} \sum_{n=1}^{l} \mathbf{w}_{m,n} \left(D_{i+o(m,l),j+o(n,l)} - D^{\text{ref}}_{i,j} \right) H_{i+o(m,l),j+o(n,l)} \,, \tag{3.1}$$

where the function $o(\cdot, \cdot)$ defines an offset to the indices and is given in Equation 2.7. The vector $\mathbf{w}_{m,n} \in \mathbb{R}^C$ indicates learnable weights in the convolution operation[2], following the standard convolution (see Section 2.1.2). Matrix $\mathbf{H}$ defines a binary mask whose elements are given by

$$H_{i,j} = \begin{cases} 0 & , D_{i,j} = 0 \\ 1 & , D_{i,j} \neq 0 \end{cases} . \tag{3.2}$$

The local reference depth $D^{\text{ref}}_{i,j}$ is defined as the average depth within each sliding window:

$$D^{\text{ref}}_{i,j} = \frac{\sum_{m=1}^{l} \sum_{n=1}^{l} D_{i+o(m,l),j+o(n,l)} H_{i+o(m,l),j+o(n,l)}}{\sum_{m=1}^{l} \sum_{n=1}^{l} H_{i+o(m,l),j+o(n,l)} + \epsilon} \,, \tag{3.3}$$

where the ϵ indicates a small constant avoiding dividing by zero.

The binary mask $\mathbf{H}$ is necessary since real-world depth maps might contain bad pixels. It means their depth values are not measurable or invalid. The depth of bad pixels is usually set to zero by the pre-processing.

[2] Since the input is a depth map with only one channel, the weight matrices in the standard convolution become vectors in this case.

The binary mask prevents bad pixels from affecting the convolution kernel and reference depth. The binary mask is unnecessary for a standard convolution. Because the depth values of bad pixels are zeroed by pre-processing and are directly multiplied by the filters in standard convolution, the depth map already has the masking effect in this case. Therefore, the advantage of RDConv comes from the local depth normalization rather than the binary mask. Some previous works, *e.g.*, sparsity invariant convolution [231], also apply similar masks. However, unlike sparsity invariant convolution [231], the proposed pipeline applies no binary mask in deeper layers. It is because sparsity invariant convolution is used in a neural network which increases the resolution of LiDAR range images. Compared to its output, the original range image is very sparse. Therefore, masks are necessary for the sparsity invariance. However, depth maps in indoor scenes have limited sparsity on the image plane (*cf*. Appendix A).

3.3 Architecture and Configuration

This section introduces the architecture of the proposed models and the configuration for training and evaluation.

3.3.1 Backbone and Feature Fusion

The backbone is built up based on the well-known ResNet-34 [80]. As shown in Figure 3.2, modifications are made by adding an RDConv layer parallel to the first (standard) convolution layer. This redundant structure aims to preserve information in absolute depth. Outputs of the two layers are directly added for simplicity, while concatenation also generates similar results. However, experiments show that the proposed models also work without absolute depth (explained later in Section 3.4.3.1). Unlike the original ResNet, the feature map after the first layer is down-sampled using strided convolution [214] instead of pooling. Also, the output channel number of the last residual block in ResNet-34 is reduced from 512 to 256 for better efficiency. Skip connections are added between down-sampling and up-sampling parts, following the spirit of U-Net [186] and feature pyramid networks [130]. The output of the backbone is a

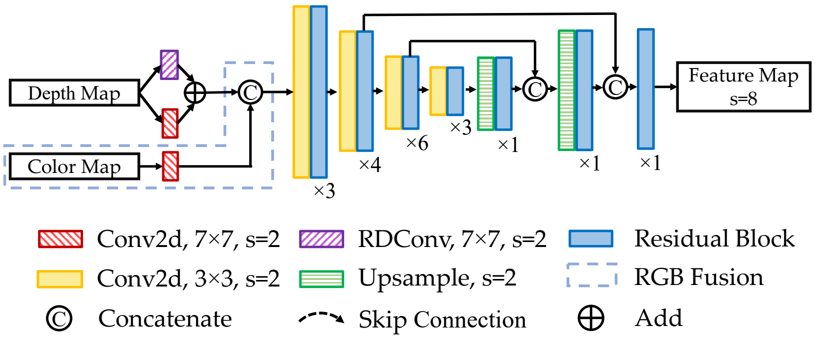


Figure 3.2 Backbone of 2.5D-VoteNet. Notation s indicates the stride, *i.e.*, sampling factor. The RGB fusion is optional.

feature map down-sampled by factor 8 (compared to the input) with 256 channels.

Until now, the network utilizes only the geometrical information (*i.e.*, depth map). For RGB fusion, a convolution layer is added and accepts RGB images at the beginning of the network. Then, the RGB features and the geometrical features are concatenated. Such an early fusion strategy fuses information from two modalities efficiently since it requires only one additional layer. In the fused model, the channel numbers of all 7×7 layers are reduced from 64 to 32, so the first scale level has the same output channel number as in the geometry-only model.

3.3.2 Detection Head and Loss Function

To build the feature carriers, the depth map is resized to the same resolution as the output feature map from the backbone. With camera calibrations, the scaled depth map is lifted to the up-right coordinate system (*i.e.*, world coordinate system) by reverting the pitch and roll angles of the camera. Then, the detection head further samples 1024 points using farthest point sampling [171]. The remaining components in the pipeline, *e.g.*, the voting module and prediction module, are adopted from VoteNet [173].

Following the baseline method, all networks are trained end-to-end with a multi-task loss, defined as a weighted sum of a voting loss (Euclidean norm), an objectness loss (cross-entropy), a bounding box loss (multi-task), and a semantic classification loss (cross-entropy). Furthermore, the bounding box loss consists of center regression (Euclidean norm), heading and size classification (cross-entropy), and 3D-GIoU (generalized intersection over union) sub-losses. The heading angles are classified since the ground truth angles are binned into intervals. The precise heading angle for a bounding box is obtained by predicting which interval the rotation angle belongs to and the residual. Also, the box sizes are classified because VoteNet uses template boxes (also known as anchor boxes), following the standard practice in 2D object detection [182]. The network predicts to which template box a prediction is most similar (*i.e.*, a classification problem). Like the heading angle, a residual is also predicted for the most similar template box.

The only difference to VoteNet is that the proposed method uses a 3D-GIoU loss [289], improving the detection results by directly maximizing the overlap between predicted bounding boxes and the ground truth. In contrast, the original VoteNet applies mean squared error as sub-losses for the residual angles and sizes.

3.3.3 Training

The input depth maps and RGB images are scaled to a resolution of 416×544. Zero padding is applied to keep the original aspect ratio. For data augmentation, the input is randomly resized by scaling the short edge in the range $[320, 512]$. Also, images are randomly horizontally flipped and rotated around their principal points by a uniform distribution in $[-15°, 15°]$. To improve the robustness against bad pixels, the depth values of 20 % pixels are randomly set to zeros. Also, when images are scaled, rotated, and flipped, the intrinsic and extrinsic camera parameters are augmented accordingly. Nearest neighbor interpolation is used to scale and rotate depth maps since interpolation methods with a higher degree generate unexpected noise around bad pixels.

The proposed models are evaluated on the SUN RGB-D dataset [212]. Models are trained on the 5000 RGB-D images in the training set and evaluated on the validation set. All models are trained using the Adam [111] optimizer for 160 epochs with an initial learning rate of 0.001 and batch size of 16. The learning rate is reduced with a factor of 0.1 after 100 and

130 epochs, respectively. More details about this dataset can be found in Appendix B.1.2.

To explore the generalization of the depth map-based pipeline, the proposed models are also trained on the ScanNet dataset [43]. The dataset consists of approximately 1500 scans reconstructed from sequences of RGB-D images. The scans are labeled with axis-aligned bounding boxes (for more details, see Appendix B.1.3). Since the proposed 2.5D-VoteNet accepts single depth maps as input, the ScanNet dataset is unpacked into approximately 2.5 million RGB-D frames. Following the official train/validation split, 78 000 frames are sampled for training and 10 000 for validation. To gain frame-level labels, the scan-level bounding box labels are transformed into each camera coordinate system using the known camera calibrations. All models are trained for 40 epochs on ScanNet. The learning rate is reduced by 0.1 after 20 and 30 epochs. Other configurations follow the training procedure for the SUN RGB-D dataset.

3.3.4 Evaluation and Inference

This chapter uses mean average precision with 25 % 3D-IoU (3D intersection over union) threshold (*i.e.*, AP25) as an evaluation metric for the detection results, following previous works [28, 98, 172–174, 259]. More details about the metrics for 3D object detection are provided in Appendix C.2. All reported metrics are calculated on the validation set since the test labels are not released [43, 212].

The inference latency is carefully measured to evaluate the computational efficiency of models. Following previous works, axis-aligned 3D NMS (C++ implementation) is applied as post-processing. The latency is measured with batch size one on a desktop computer with an Intel Core i7-8700 CPU and an NVIDIA RTX 2080Ti GPU using PyTorch [167].

3.4 Experiments and Analysis

This section first demonstrates the main experimental results by comparing the proposed methods with existing ones. Then, qualitative results are shown by visualizing predicted bounding boxes and learned features. Finally, analysis experiments are conducted to justify the proposed designs.

3.4.1 Comparison with State-of-the-Art Methods

The geometry-only and fused versions of 2.5D-VoteNet are first compared with state-of-the-art (SOTA) methods on the SUN RGB-D benchmark. Then, they are compared with VoteNet in the ScanNet frame-level detection task.

3.4.1.1 SUN RGB-D Object Detection Benchmark

The SOTA methods on the SUN RGB-D dataset are briefly summarized as follows: COG [183] represents features with cloud of oriented gradient descriptors. VoteNet [171] uses PointNet++ [171] to capture features and centralizes the features with deep Hough voting. MLCVNet [259] optimizes VoteNet via multi-level context, while HGNet [28] improves VoteNet with multi-scale features and attention mechanism. Ahmed and Chew [3] utilize unsupervised clustering to perform class-agnostic instance segmentation and improve the detection results. EPNet [98] enhances point features with image semantics. H3DNet [284] introduces hybrid geometric primitives to the 3D detection pipeline. SPOT [50] proposes selective point cloud voting for better object proposals. Deep sliding shapes (DSS) [213] captures 3D features with volumetric convolution. PointFusion [265] and Frustum-PointNet [172] generate 2D proposals using off-the-shelf 2D detectors and predicts 3D bounding boxes from each 2D proposal. ImVoteNet [174] fuses RGB images by using a pre-trained 2D detector to gain 2D geometrical and color cues.

The SOTA methods can be separated into methods without and with additional data. The latter have components pre-trained on other datasets (*e.g.*, ImageNet [44], COCO [129], ScanNet [43]), whereas the former are directly trained on the target dataset.

As shown in Table 3.1, both proposed models achieve competitive results and are significantly faster than previous methods. The geometry-only and fused models reach 3.1 % and 4.0 % (absolute) higher AP25 than the baseline VoteNet (57.7 % AP25), respectively. The fused model achieves the best results (61.7 % AP25) among methods without extra

Table 3.1 Quantitative detection results on validation set of SUN RGB-D with V1 annotation. AP25 is calculated over the 10 most common object classes. The upper half of the table shows the results of methods pre-trained using extra data. The models in the lower half are trained from scratch. Proposed models marked with * are pre-trained on ScanNet frame-level dataset. The average precision for each class is presented in the intermediate columns. Some values are absent because they are not reported in the original publications. Latency: inference latency per frame on a single GPU. The hardware might differ in different publications. **Bold**: the best result. P: raw point clouds. I: color images. D: depth maps. V: voxels.

Methods	**Input**	**Pre-Train**	bathtub	bed	bookshelf	chair	desk	dresser	nightstand	sofa	table	toilet	**Latency**	**AP25** (%)
DSS [213]	V+I	✓	44.2	78.8	11.9	61.2	20.5	6.4	15.4	53.5	50.3	78.9	19.6 s	42.1
PointFusion [265]	P+I	✓	37.3	68.6	37.7	55.1	17.2	23.9	32.3	53.8	31.0	83.8	1.3 s	45.4
F-PointNet [172]	P+I	✓	43.3	81.1	33.3	64.2	24.7	32.0	58.1	61.1	51.1	90.9	0.12 s	54.0
ImVoteNet [174]	P+I	✓	75.9	87.6	41.3	**76.7**	28.7	41.4	**69.9**	**70.7**	51.1	90.5	-	63.4
*Proposed** (geo)	D	✓	75.3	88.1	41.6	75.6	29.2	39.1	61.0	70.5	49.7	89.8	14.5 ms	62.0
*Proposed** (fused)	D+I	✓	76.4	87.1	**50.5**	75.0	31.2	41.4	64.4	**70.7**	49.7	90.3	14.5 ms	**63.7**
COG [183]	P+I	✗	58.3	63.7	31.8	62.2	**45.2**	15.5	27.4	51.0	51.3	70.1	10 min	47.6
Ahmed *et al.* [3]	P	✗	**79.4**	**88.2**	32.1	17.0	37.4	**53.7**	50.0	65.3	**53.3**	**95.8**	1.24 s	57.2
VoteNet [173]	P	✗	74.4	83.0	28.8	75.3	22.0	29.8	62.2	64.0	47.3	90.1	0.1 s	57.7
MLCVNet [259]	P	✗	79.2	85.8	31.9	75.8	26.5	31.3	61.5	66.3	50.4	89.1	-	59.8
EPNet [98]	P+I	✗	75.4	85.2	35.4	75.0	26.1	31.3	62.0	67.2	52.1	88.2	-	59.8
H3DNet [284]	P	✗	-	-	-	-	-	-	-	-	-	-	0.15 s	60.1
SPOT [50]	P	✗	-	-	-	-	-	-	-	-	-	-	-	60.4
HGNet [28]	P	✗	78.0	84.5	35.7	75.2	34.3	37.6	61.7	65.7	51.6	91.1	0.1 s	61.6
Proposed (geo)	D	✗	77.5	86.8	36.5	75.3	27.0	37.5	64.0	66.8	48.7	87.8	14.5 ms	60.8
Proposed (fused)	D+I	✗	71.8	85.3	44.8	76.2	27.0	33.4	68.3	70.0	51.8	88.6	14.5 ms	61.7

data. Moreover, supervised pre-training on ScanNet frame-level data is also explored. That is, the proposed models trained on the ScanNet dataset are further fine-tuned on the SUN RGB-D dataset. The pre-training further improves the AP25 of proposed models by 1.2 % and 2 %, respectively. The fused model reaches the best AP25 (63.7 %) among all methods.

Also, *bookshelves* are challenging for point cloud-based detectors, as they are large and hard to distinguish from walls. Thanks to the large perceptive field and the capability to capture fine-grained local features of 2D CNNs, the proposed geometry-only model achieves the best AP25 on the class bookshelf among all geometry-only methods. Furthermore, the fused model surpasses all previous methods by a large margin.

It is worth noting that several methods in comparison [28, 50, 174, 259, 284] are optimized variants of VoteNet [173]. They aim to improve the mean average precision rather than the speed. The proposed 2.5D-VoteNet, however, significantly improves both aspects at the same time. The results show that both geometry-only and fused models achieve impressive trade-offs between the detection quality and inference latency. Also, thanks to the efficiency of the early fusion strategy, the run-time difference between the geometry-only and fused models is unnoticeable.

3.4.1.2 ScanNet Single-Frame Object Detection

The detection results in the ScanNet frame-level detection task are shown in Table 3.2. The reported AP25 is averaged over 18 object classes, following the standard configuration on this dataset [28, 173]. Since the task is newly introduced and not considered in previous works, the performance of SOTA methods on this task is unavailable. This chapter compares proposed methods with the baseline VoteNet using its official open-source code base. As shown in Table 3.2, the geo-only and fused 2.5D-VoteNet achieves 6.8 % and 7.2 % higher AP25 than the baseline VoteNet, respectively. The improvement on this dataset is more significant than on SUN RGB-D. It is because frames from ScanNet are more challenging due to more partially visible objects and more hard samples, *e.g.*, doors and curtains (see Figure 3.4).

Table 3.2 Detection results on the ScanNet frame-level detection task.

Methods	**AP25** (%)
VoteNet [173]	44.0
2.5D-VoteNet (geo)	50.8
2.5D-VoteNet (fused)	51.2

3.4.2 Qualitative Results

Qualitative results are provided to visualize the detection results and learned features of the proposed method.

3.4.2.1 Object Detection Results

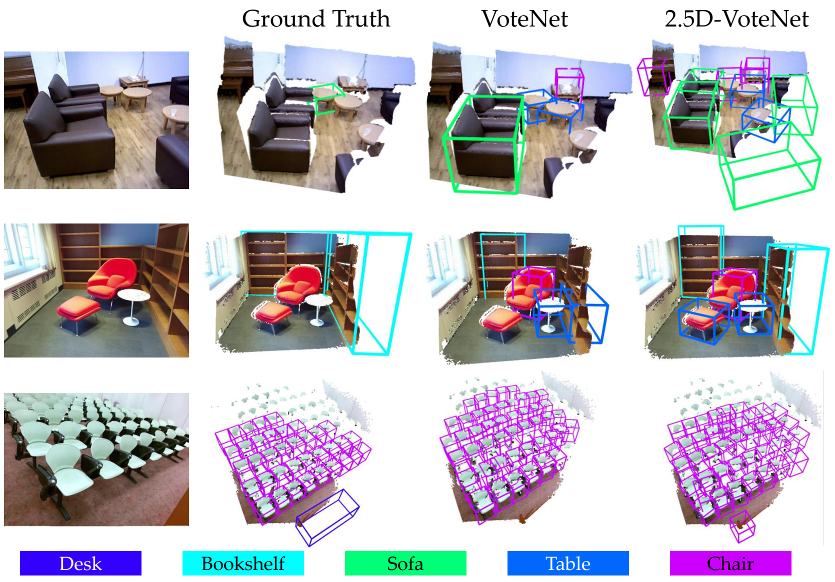


Figure 3.3 Qualitative results on SUN RGB-D dataset. Some labels in the dataset have flaws.

As qualitative results, the detected boxes from 2.5D-VoteNet (geometry-only version for a fair comparison) and VoteNet are illustrated in Figure 3.3. The proposed model can detect partially visible objects (*i.e.*, sofas in the top scene), large objects (*i.e.*, bookshelves in the middle scene), and severely cluttered small objects (*i.e.*, chairs in the bottom scene). 2.5D-VoteNet obtains better detection quality than the baseline method VoteNet. Notice that the labels in the SUN RGB-D dataset have some flaws since the ground truth in Figure 3.3 misses some bounding boxes (*e.g.*, sofas in the top row). However, these objects are correctly detected by

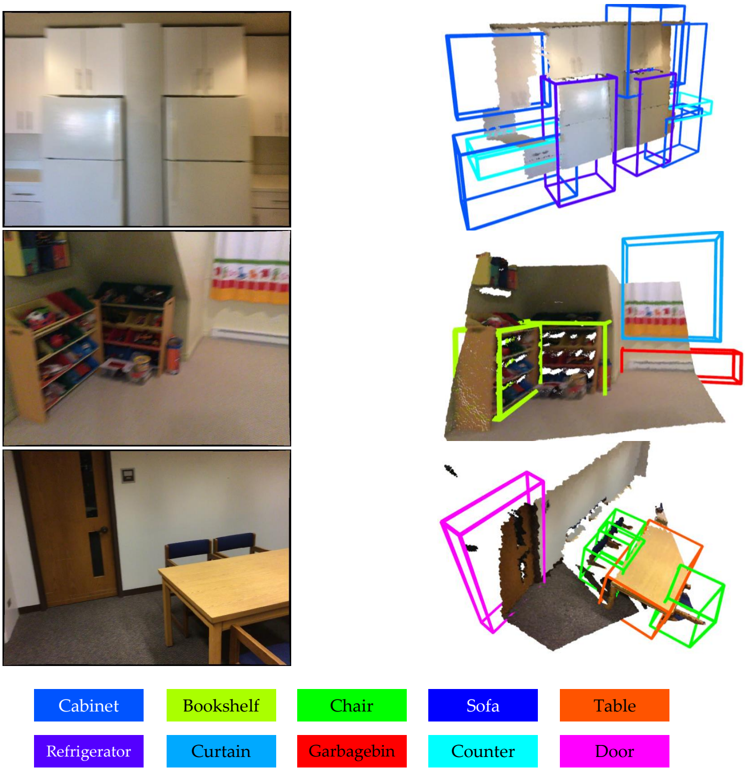


Figure 3.4 Qualitative results on single frames from the ScanNet dataset.

2.5D-VoteNet. It implies that the proposed method generalizes well and does not over-fit the labels. Besides the SUN RGB-D dataset, Figure 3.4 also illustrates the detection results on ScanNet.

3.4.2.2 Feature Maps from Backbone

An interesting question about the depth map-based 3D detection pipeline is, what features can the 2D backbone learn? In this experiment, the output feature map of the backbone, which has a resolution of 52×68 and 256 channels, is visualized. Specifically, each channel of the feature map is normalized into the interval $[0, 1]$ and plotted as color maps (also

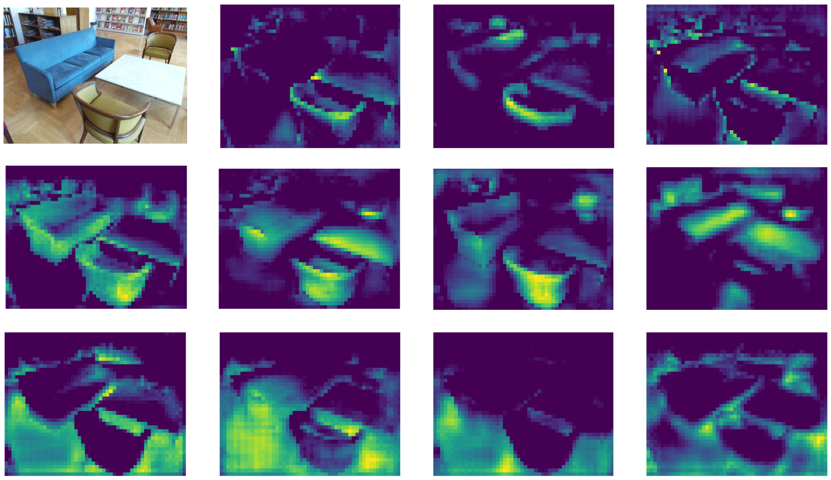


Figure 3.5 Visualization of learned feature maps from the backbone in a 2.5D-VoteNet. The lighter the colors, the higher the activations.

known as activation maps or heat maps). Some representative activation maps are shown in Figure 3.5. It illustrates that the 2D backbone can learn rich features from depth maps. For instance, maps in the first row show high activation at object edges, whereas the second row highlights object surfaces. Meanwhile, the last row presents large values in the background.

3.4.3 Analysis

Comprehensive experiments are conducted to justify the design of the proposed models.

3.4.3.1 Impact of RDConv

Experiments are conducted to clarify the impact of the proposed RD-Conv. Table 3.3 shows that, with standard convolution as the first layer, the depth map-based pipeline does not generate good detection results (56.1 % AP25). Also, the training process is unstable, where a massive fluctuation of the training loss can be observed. It might be due to

Table 3.3 Impact of first layers and GIoU-loss on the detection results on SUN RGB-D. First layers refer to layers that directly use RGB images or depth maps as input. Metric: AP25 in percentage. D: depth maps. I: RGB images.

First Layers	Input	w/o GIoU	w/ GIoU
Conv	D	55.0	56.1
RDConv	D	58.7	60.4
Conv + RDConv	D	59.0	60.8
Conv + RDConv	D+I	**60.6**	**61.7**

over-fitting since standard convolution takes absolute depth as input and learns relative spatial relations indirectly. By simply replacing the first layer with RDConv, the network gains a significant performance improvement (~4 %), and the instability is also removed. The result proves that RDConv is the key component that enables the proposed depth map-based detector to learn informative features. While the network with one RDConv alone as the first layer already achieves competitive performance (60.4 % mAP), combining the absolute depth and RGB fusion brings further improvement (0.4 % and 0.9 %), respectively.

3.4.3.2 GIoU Loss

As explained by Zhou *et al*. [289], object detection benefits from IoU-based losses since they unify the optimization objective and the evaluation metric, *i.e.*, the overlap of predictions and group truth boxes (*cf.* Appendix C.2). To clarify the influence of the GIoU-loss, this experiment removes it and trains 2.5D-VoteNet with the same loss function as the baseline method VoteNet [173]. As shown in Table 3.3, using 3D-GIoU loss, the geometry-only network gains 1.8 % improvement, whereas the fused version gains 1.1 % (absolute). However, even without the advanced loss function, the proposed networks outperform the baseline VoteNet and reach comparable results with SOTA methods.

3.4.3.3 Fusion Strategies

This experiment evaluates different fusion strategies, including directly concatenating normalized depth maps to RGB images as the fourth chan-

Table 3.4 Detection results with different fusion strategies on the SUN RGB-D validation set. Early Fusion is the proposed strategy in this chapter.

Fusion	**First Layers**	**AP25** (%)
Geometry-only	RDConv + Conv	60.8
RGB-only	Conv	53.3
Naive concatenation	Conv	59.9
Early fusion	RDConv + Conv	**61.7**
Late fusion	RDConv + Conv	54.5

nel, early fusion with an extra convolution layer to accept RGB values (see Figure 3.2), and late fusion with two backbones. By the late fusion, the RGB backbone is a ResNet-34 pre-trained on ImageNet [44] with similar up-sampling layers as in Figure 3.2. With such a two-stream structure, the network learns geometrical and color features with different backbones. Learned features are then concatenated to build feature carriers. To balance the learning rate in different modalities, the late fusion variant applies the multi-tower training strategy [242], following the method of Qi *et al*. [174].

As shown in Table 3.4, the simple and efficient early fusion strategy brings the best result (61.7 % AP25). On the contrary, early fusion often worsens the detection results of point cloud-based methods, as discussed in [93, 174]. It is because images and point clouds have distinct properties. For instance, images are regular dense data with a high resolution. However, point clouds are irregular and sparse. Also, they have a lower resolution due to down-sampling. This observation implies that a depth map-based pipeline simplifies the fusion of geometry and color information. Also, the geometry-only model brings better results than naive concatenation (60.8 % *vs*. 59.9 % AP25), which proves the effectiveness of RDConv. Interestingly, the late fusion does not generate good results (54.5 % AP25). It might be caused by over-fitting, as the two-stream structure requires more training data.

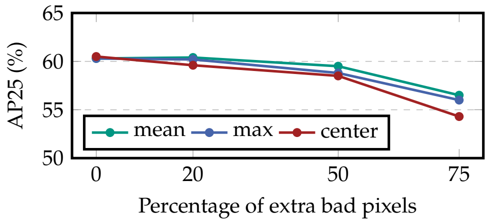


Figure 3.6 Performance of different reference depths on the SUN RGB-D dataset. Absolute depth is not used, to emphasize the contribution of RDConv.

3.4.3.4 Reference Depth

RDConv uses the masked mean average depth in each sliding window as a reference (Equation 3.3). This experiment evaluates other options, *e.g.*, using the mean or max value in each sliding window instead. Moreover, depth maps in the real world may contain more noise than the SUN RGB-D dataset since offline post-processing is applied in the dataset for a better data-to-noise ratio [212]. To validate the robustness of the proposed networks, this experiment randomly adds extra bad pixels into depth maps. Figure 3.6 shows that the three choices deliver similar results when no additional bad pixels are added. However, the mean value brings a slightly better result than the max and center values when depth maps have more bad pixels. This experiment shows that using the mean value in RDConv makes models less sensitive to noises in data.

3.4.3.5 Kernel Size

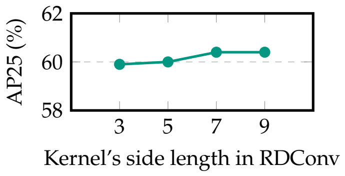


Figure 3.7 Performance of RDConv with different kernel sizes on the SUN RGB-D dataset.

Since RDConv is an essential component in 2.5D-VoteNet, it is meaningful to investigate the impact of its kernel size. A 7×7 kernel is applied with the default setup. This experiment trains a series of models with kernel sizes of 3×3, 5×5, 7×7, and 9×9, respectively. The absolute depth is not used in this experiment, so the performance relies only on RDConv. As shown in Figure 3.7, the 3×3 kernel generates 59.9 % AP25 on the SUN RGB-D dataset. The value increases to 60.0 % and 60.4 % with 5×5 and 7×7 kernel, respectively. A larger kernel (9×9) does not bring more improvement.

3.4.3.6 Different Encoders

This experiment tests the geometry-only 2.5D-VoteNet with different input resolutions and encoders (*i.e.*, the down-sampling parts of the backbone). As shown in Table 3.5, the AP25 drops by 1.1 % when the resolution is reduced to 320×416. It implies that a high input resolution is beneficial for 3D detection. However, a resolution higher than 416×544 is not applicable due to the limitation of the dataset. To research the impact of model depth, the default ResNet-34 in the 2D backbone is replaced with ResNet-18 and ResNet-50 [80]. The AP25 of the ResNet-34-based backbone is 0.5 % better than the ResNet-18-based one. However, the deeper ResNet-50 leads to a worse result. It is probably due to over-fitting since only ~5000 depth maps are used to train the model.

Table 3.5 also reports the performance with other well-known CNNs, *e.g.*, VGG-16 [207] and MobileNetV2 [194]. Similar to the configuration with ResNets, VGG-16 and MobileNetV2 are modified by replacing their first layer with a strided 7×7 convolution and a strided 7×7 RDConv. The detectors reach similar performance with ResNet-34 and MobileNetV2. The performance with VGG-16 is 2.1 % lower than ResNet-34 and 1.4 % lower than MobileNetV2. It is reasonable since VGG-16 is an early architecture with less optimization, *e.g.*, skip connections [80]. On the contrary, MobileNetV2 and ResNet-32 perform similarly. The results show that the proposed 2.5D-VoteNet is not sensitive to the choice of 2D encoders.

Table 3.5 Detection results with different structures of 2D encoders in the backbone. All models use depth maps as input.

Resolution	**Encoder**	**AP25** (%)
416×544	ResNet-18	60.3
320×416	ResNet-34	59.7
416×544	ResNet-34	**60.8**
416×544	ResNet-50	59.9
416×544	VGG-16	58.7
416×544	MobileNetV2	60.1

3.4.3.7 Calibration-Awareness of the Backbone

With the known camera calibration, a depth map can be converted to a point cloud. As mentioned in Section 3.2.2, the 2D backbone of 2.5D-VoteNet does not take the camera calibration as input. On the contrary, point cloud-based networks are calibration-aware, as point clouds already contain this information. Since the proposed pipeline applies a 2D CNN to extract features directly from depth maps, the backbone receives less information than point cloud-based methods. Therefore, one interesting question is, would the performance of proposed models be improved if the camera calibration is explicitly or implicitly provided?

A straightforward approach is concatenating additional features to feature carriers. This experiment compares different choices of additional features, including 3D coordinates of feature carriers in the world coordinate system (*i.e.*, intrinsic and extrinsic parameters), the 3D coordinates of feature carriers in camera coordinate (*i.e.*, intrinsic), the vectorized camera matrix (*i.e.*, intrinsic), and the Euler angles of the camera rotation (*i.e.*, extrinsic). This experiment also evaluates a unique type of convolution layer CAM-Conv [55], directly incorporating camera intrinsic parameters into the standard convolution operation.

The obtained AP25 using these variants are reported in Table 3.6. Interestingly, although the additional features inject camera calibrations into the detection head, they either worsen the results or do not make significant differences. One hypothesis is that the 2D backbone of 2.5D-

Table 3.6 Detection results when camera calibrations are concatenated as additional features to the output of the 2D backbone. The baseline in this experiment is a geometry-only 2.5D-VoteNet. The additional features are normalized if necessary.

Features	**AP25** (%)
Baseline (256D features)	**60.8**
+ 3D coord. (upright)	60.3
+ 3D coord. (camera)	60.1
+ Camera matrix	60.6
+ Euler angles	60.7
+ CAM-Conv [55]	60.6

Table 3.7 Impact of scaling and rotating the depth maps. The geometry-only 2.5D-VoteNet is applied in this experiment.

Data Augmentation	**AP25** (%)
With scaling and rotation	**60.8**
Without scaling and rotation	56.4

VoteNet already learns invariance to the camera calibration, thanks to the applied data augmentation. Specifically, depth maps and images are randomly scaled and rotated around principle points during training. Meanwhile, the camera calibrations are adjusted accordingly, so the augmented depth maps can be converted to the same 3D coordinates as the original depth maps. By doing this, the network has to learn calibration invariance, as the calibration changes during the training, but the 3D geometry remains the same. To validate this hypothesis, a 2.5D-VoteNet is trained without scaling and rotation. As Table 3.7 shows, the AP25 drops by 4.4 % (absolute). The result implies that the proposed models learn calibration invariance via data augmentation. Therefore, the extra features mentioned above are unnecessary.

Table 3.8 Speed and model size of 2.5D-VoteNet models and the baseline. Pre-processing is excluded since it runs in parallel on the CPU.

Methods	**Latency** (ms)				**Size**
	Backbone	Head	NMS	Total	(MB)
VoteNet (original) [173]	-	-	-	100	11.7
VoteNet (reproduced)	59.1	6.6	0.3	66.0	11.7
2.5D-VoteNet (geo)	7.7	6.6	0.3	14.5	67.4
2.5D-VoteNet (fused)	7.7	6.6	0.3	14.5	67.4

3.4.3.8 Model Size and Speed

For a fair comparison, this chapter adopts the open source code of VoteNet and measures the model speed with the same setup in Section 3.3.4. Due to different hardware and setup, the run-time is shorter than in the original publication. As shown in Table 3.8, the proposed models are 4.5 times faster than the baseline, thanks to the efficient 2D backbone. The speed of the geometry-only and the fused model is reported as the same since the difference is too small to be measured. Also, the fused version has one more convolution layer than the geo-only model, and the layer has less than 5000 parameters.

To validate the proposed pipeline on mobile devices, a 2.5D-VoteNet is also tested on an NVIDIA Jetson Xavier NX, an onboard computer system for edge computing using GPU. The model runs with 148 ms per frame (~7 FPS) using PyTorch.

3.4.3.9 Run-Time of Pre-Processing

The inference time reported in Section 3.4.3.8 excludes the pre-processing since it usually runs in parallel on the CPU. However, pre-processing greatly impacts the latency when the CPU has less computational power or has to accomplish all computations (*i.e.*, no co-processor available).

Table 3.9 compares the pre-processing times of the depth map-based detection pipeline and the baseline VoteNet. The former includes scaling a depth map and an RGB image to a fixed size of 416×544 and normalizing their pixel values to [0, 1]. Also, an 8-times down-sampled copy of the

Table 3.9 Comparison of single-thread pre-processing time of proposed models and the baseline.

Methods	**Pre-Processing** (ms)
VoteNet	19.3
2.5D-VoteNet (geo-only)	**11.6**
2.5D-VoteNet (fused-only)	17.1

depth map is created. To build feature carriers, the CPU also lifts the down-sampled depth map to the 3D space and further samples 1024 points using farthest point sampling. The latter contains primarily the time cost of transforming the original depth map into the 3D space and randomly sampling 20000 points.

The run-time is measured using the desktop computer described in Section 3.3.4. The reported times include loading data from the memory and sending data to the GPU. For a fair comparison, all implementations use standard libraries *e.g.*, NumPy and OpenCV, with multi-threading disabled. As shown in Table 3.9, both proposed models require shorter pre-processing time than the baseline VoteNet, as converting depth maps to point clouds is computationally expensive. The results show that the depth map-based pipeline accelerates not only the computation of neural networks but also the pre-processing compared to point cloud-based methods.

Moreover, the geometry-only model requires less time than the fused model since it does not load and process RGB images. The single-thread pre-processing time of the fused model is longer than its inference time (14.5 ms). However, it would not bottleneck the frame rate of proposed models, as multi-threading and multi-processing are often employed in real-world applications for parallelized pre-processing.

3.5 Additional Comparisons with Related Works

The previous contents of this chapter focus on comparing the performance of 3D object detection using depth maps and point clouds. This chapter compares the proposed method with other related approaches to clarify its uniqueness and novelty.

3.5.1 2D CNNs for Range Images and Depth Maps

As discussed in Section 2.2.8, many methods for autonomous driving project point clouds to the bird-eye-view [117, 206, 283, 291]. However, they are not suitable for cluttered indoor scenes. Also, some methods [32, 122, 148] transform LiDAR point clouds into range images. However, they directly predict 3D bounding boxes on 2D feature maps and consequently lose precise 3D spatial information. RangeRCNN [127] combines the LiDAR range image with point cloud and bird-eye-view representations for 3D detection. However, this pipeline is still computationally expensive due to the complicated hybrid structure. Bewley *et al*. [19] propose range-conditioned dilated convolution for scale invariant 3D detection on range images. In contrast, the proposed method in this chapter addresses the absolute depth invariance of local features. Moreover, some methods [243, 261, 262] mimic the behavior of 3D CNNs using 2D CNNs. However, this chapter shows that capturing geometrical features on depth maps does not necessarily depend on such behaviors.

3.5.2 RGB Fusion in 3D Detection

Since RGB images provide complementary information to 3D coordinates, many works fuse color and geometrical features for more robust and accurate object detection. Some methods [172, 265] generate 2D regions of interest (RoI) on RGB images using a pre-trained 2D detector, while objects within each 2D RoI are detected via point cloud-based networks. Also, some approaches [10, 98, 213, 238] enrich point features with high-level features from RGB images. ImVoteNet [174] augments the voting module of VoteNet [173] using image votes from a 2D detector. However, all these methods extract color features using a separate 2D CNN (*i.e.*, late fusion), which inevitably increases the computational cost. On the contrary, the proposed pipeline in this chapter applies a simple and efficient early fusion strategy.

3.6 Conclusions

This chapter introduces 2.5D-VoteNet, a simple, powerful, and efficient method for real-time 3D object detection. The proposed models achieve state-of-the-art performance and show significant speed improvement over previous point cloud-based methods. Also, a simple yet effective relative depth convolution is proposed to capture strong features from depth maps. This chapter also explores supervised pre-training and feature fusion for better detection results. Comprehensive experiments are conducted to justify the design choices, *e.g.*, kernel size, fusion strategy, and reference depth.

Moreover, the supervised pre-training on ScanNet frame-level object detection task shows promising benefits (see Table 3.1). However, 3D labels might be unavailable or costly to obtain in practice. To pre-train 3D neural networks using unlabeled data, Chapters 4 and 5 explore self-supervised technologies for better label efficiency.

Furthermore, this chapter provides an interesting observation that even though single-view depth maps and point clouds represent the same information, they lead to different results in object detection. It implies that data representations affect the feature learning in 3D computer vision and inspires Chapter 4 to exploit properties of different representations in contrastive learning.

On the other hand, the proposed depth map-based method has the limitation that it is not applicable if the input point cloud cannot be represented as a single-view depth map. It motivates Chapter 5 to apply transformers [235] as computationally efficient models for multi-view point cloud understanding.

4 Invariance-Based Contrastive Learning for Label Efficiency

Chapter 3 has demonstrated that supervised pre-training can significantly improve the performance of neural networks in downstream tasks. However, this technique relies on labeled data, which limits its application in practice. At the same time, self-supervised pre-training for 3D vision has drawn increasing research interest in recent years (*cf*. Section 2.4). To learn informative representations, a lot of previous works exploit invariances of 3D features, *e.g.*, perspective invariance [260] between view angles, modality invariance between depth maps and RGB images [134], format invariance[1] between point clouds and voxels [285]. Although they have achieved promising results, previous research lacks a fair comparison of these invariances. To address this issue, this chapter introduces a unified framework, under which various pre-training methods are systematically investigated. This chapter conducts extensive experiments and provides a closer look at the contributions of different invariances in 3D pre-training. Also, it proposes a simple but effective method that jointly pre-trains a 3D encoder and a depth map encoder using contrastive learning. Models pre-trained with the proposed strategy gain significant performance boosts in downstream tasks. For instance, a pre-trained VoteNet [173] outperforms previous methods on SUN RGB-D [212] and ScanNet [43] object detection benchmarks with a clear margin. The main contents of this chapter have been published in [295].

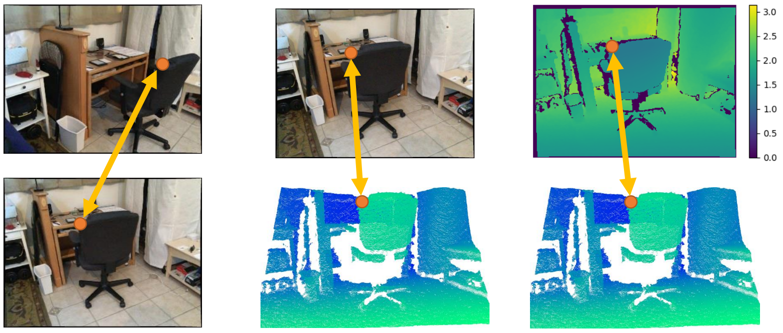
[1] In this chapter, point clouds, voxels, and depth maps are referred to as different *formats* of 3D data. The term *data format* is used instead of *data representation*. Since the learned features are often referred to as representations in the literature, the term data representation and the corresponding representation invariance might cause confusion in the context of contrastive learning.

4.1 Introduction

In order to cope with challenging tasks, *e.g.*, object detection, scene understanding, and large-scale semantic segmentation, neural networks for 3D vision are continuously becoming deeper, more complicated, and thus, more data-hungry. In recent years, self-supervised pre-training has shown promising progress in natural language processing and computer vision (*cf.* Section 2.4). The models gain better performance and convergence in downstream tasks by learning powerful representations on unlabeled data. Self-supervised pre-training is especially appealing in 3D vision because 3D annotation is more costly than the 2D counterpart.

Self-supervised pre-training for 3D vision has already gained some research interest. A lot of previous works use contrastive learning as a pretext task to pre-train models, as it has shown superior performance in other domains [94, 134, 135, 260, 285] (*cf.* Section 2.4.3). One classic hypothesis in contrastive learning is that a powerful representation should model view-invariant factors. A common approach to creating different views is data augmentation. Moreover, a 3D scene can be captured from various view angles, with different sensors (*e.g.*, RGB and depth cameras), and represented with different formats (*e.g.*, voxels, point clouds, and depth maps), whereas these factors do not change the major semantic information in the scene. Thus, previous works exploit the perspective [260], modality [134] and format invariance [285] of 3D features using contrastive learning, as shown in Figure 4.1. Although these works have shown impressive results, the contribution of the invariances is still under-explored, and a fair and systematic comparison of them has not been performed.

This chapter establishes a unified framework for 3D self-supervised learning. The framework takes into account the local point/pixel-level correspondence as well as the global instance-level correspondence. Also, the framework unifies contrastive learning with different input data formats and network structures, including depth-depth, point-point, depth-point, image-point, and point-voxel contrast. The first insight of this chapter is that jointly pre-training a 3D encoder and a 2D encoder (*e.g.*, image-point, depth-point) brings better performance than pre-training them separately or jointly pre-training two encoders with the same di-



(a) Perspective invariance (b) Modality invariance (c) Format invariance

Figure 4.1 Invariances in contrastive learning for 3D vision. Without loss of generality, only the local correspondence is considered. Each column includes two views of the same scene. The exemplary correspondences across two views are illustrated with arrows, which means the two points/pixels have the same coordinate in the 3D space. In self-supervised pre-training, the similarity between corresponding local features is maximized, which forces networks to learn invariance between views. **(a) Perspective invariance** in two views of the same scene from different view angles. RGD images are visualized instead of point clouds for better clarity. **(b) Modality invariance** in an aligned image-point cloud pair. The data formats are also different in this case. This chapter refers to it as modality invariance to distinguish it from the format invariance within a single modality. **(c) Format invariance** between a depth map and a point cloud converted from it.

mension, *e.g.*, a voxel and a point cloud encoder, which are both three dimensional.

Furthermore, a simple but effective method is proposed to exploit the format invariance between depth maps and point clouds/voxels. The intuition behind this design is that depth maps are complementary to point clouds and voxels, although they contain almost the same information. The depth map format has the advantage of being the natural view of a scene and clearly showing the perspective relationship between objects. Also, real-world depth maps usually contain bad pixels, which means the depth values are unmeasurable (*cf.* Appendix A.1). In depth maps, the outlines of unmeasurable regions are sharp and clear, *e.g.*, the chair leg in Figure 4.1(c). On the contrary, this information is lost if depth maps are lifted into 3D space. Moreover, thanks to its efficiency, a 2D encoder allows high-resolution depth maps as input, which preserves more fine-grained details in data. However, point or voxel-based networks usually take down-sampled or quantized input to avoid the high computational cost and memory usage, which results in an inevitable information loss. On the other hand, point clouds and voxels are 3D formats, and the corresponding networks can directly capture accurate 3D geometry, whereas depth map-based networks learn spatial relationships indirectly. Also, depth maps alone do not contain information on camera calibrations. By contrasting the features extracted from two complementary data formats, the two networks learn appreciated properties from each other.

The contribution of this chapter is many-fold:

1. It introduces a unified self-supervised pre-training framework for multiple network architectures and data formats in 3D vision.
2. It provides a closer look at invariances in 3D pre-training, *e.g.*, format, perspective and modality invariance.
3. It proposes a novel approach for 3D self-supervised pre-training based on the format invariance between depth maps and a 3D format (*e.g.*, point clouds and voxels).
4. A point cloud-based object detector pre-trained using the proposed method archives SOTA results on multiple downstream datasets, *e.g.*, object detection on the SUN RGB-D [212] and the ScanNet [43] benchmark.
5. The proposed method is the first self-supervised pre-training approach for depth map-based networks (*e.g.*, the depth map-based object detector proposed in Chapter 3).

4.2 Invariance-Based Contrastive Learning

This chapter researches the invariances in 3D self-supervised learning, including perspective, modality and format invariance. For a fair comparison, it is meaningful to investigate them under a unified framework. In this section, some representative works are first briefly revisited. Then, a unified framework is introduced to which all previous methods fit.

Moreover, several contrastive learning methods under the unified framework are presented. At last, some technical details of the framework are provided.

4.2.1 Unified Framework

The concept of contrastive learning is provided in Section 2.4.3. This chapter pays attention to three previous works:

1. PointContrast [260]: It generates two views of a scene from different perspectives and learns the local correspondences between 3D points. This method assumes that the extrinsic camera parameters are available.
2. DepthContrast [285]: Following the successful MoCo pipeline [35, 82] (see Section 2.4.3), it augments two views of the same point cloud to build a positive pair and learns global correspondences by distinguishing the positive samples from a large number of negative samples. Also, it proposes exploiting cross-format contrast between point clouds and voxels.
3. Pixel-to-point [134]: Its overall pipeline is similar to PointContrast. However, it learns local correspondences between a point cloud and an RGB image to benefit from RGB encoders pre-trained on large-scale datasets, *e.g.*, ImageNet [44].

To compare these methods, a unified framework must support both the local and global correspondence of 3D data and at least two different input types, either from different modalities (*e.g.*, RGB images and point clouds) or with different data formats (*e.g.*, voxels and point clouds).

The proposed framework is shown in Figure 4.2, which uses a single-view depth map or RGB-D image to generate required data. However, experiments show that the pre-trained weights generalize well on reconstructed multi-view 3D scans. Without loss of generality, the following explanations assume that the input of the framework is a depth map. The input is randomly cropped into a crop C_1, which is further randomly augmented and converted into views α_1 and β_1. Here α and β refer to

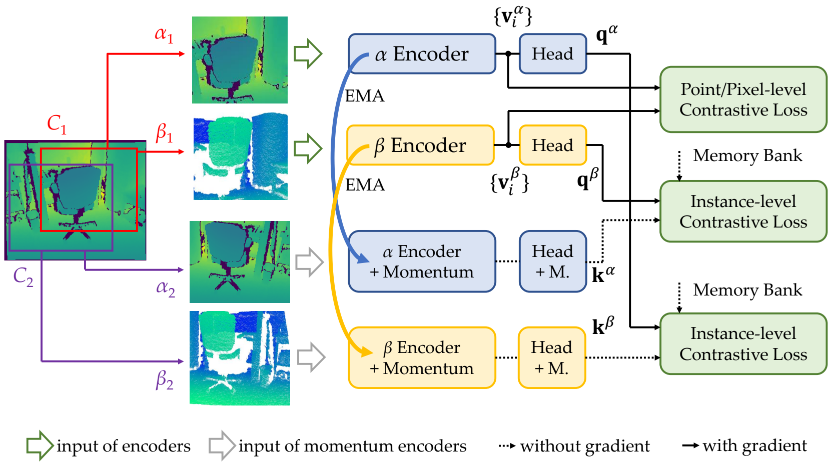


Figure 4.2 A unified framework for 3D contrastive learning. Here, α and β refer to data formats, *e.g.*, point clouds, images, and depth maps.

data formats (*e.g.*, depth maps and point clouds, as visualized in Figure 4.2). Then, α_1 and β_1 go through respective encoders and are encoded into pixel-wise or point-wise features $\{\mathbf{v}_i^\alpha\}$ and $\left\{\mathbf{v}_i^\beta\right\}$, where $\mathbf{v}_i^\alpha$ and $\mathbf{v}_i^\beta$ indicate local feature vectors, respectively. Here, a subscript i instead of a superscript is used to distinguish local features with the same format, in order to emphasize that the features are from one view (*i.e.*, they can be regarded as vectors from a matrix). The α and the β encoder are usually different networks matching the input formats. However, in the case of $\alpha = \beta$, they share the weights, following PointContrast [260]. As α_1 and β_1 are generated from the same crop C_1, the dense correspondence between sets $\{\mathbf{v}_i^\alpha\}$ and $\left\{\mathbf{v}_i^\beta\right\}$ can be easily calculated without camera extrinsics. In this chapter, the InfoNCE loss [160] is used to train dense local correspondences, which is further explained in Section 4.2.3.

To learn informative representations, the framework also considers the global correspondence between views. Following DepthContrast [285], instance discrimination is performed using global features $\mathbf{q}^\alpha$ and $\mathbf{q}^\beta$, which are globally pooled and transformed from $\{\mathbf{v}_i^\alpha\}$ and $\left\{\mathbf{v}_i^\beta\right\}$, respectively.

To preserve a large number of negative samples for effective contrastive learning, memory banks and momentum encoders are also applied, following the successful MoCo pipeline [35, 82] (*cf.* Section 2.4.3). Moreover, Section 4.3.4.4 further shows that the proposed methods also work with other contrastive learning schemes, *e.g.*, BYOL [72] and SimSiams [33]. Similar to crop C_1, the framework crops C_2 from the same depth map, generates α_2 and β_2, and feeds them to the momentum encoders. The globally pooled and transformed features from momentum encoders are indicated by $\mathbf{k}^\alpha$ and $\mathbf{k}^\beta$, respectively. They are dynamically saved and updated in memory banks during training. More details on the MoCo pipeline are provided as related works in Section 2.4.3.

4.2.2 Variants of Strategies

As the overall framework is shown, various contrastive learning strategies under this framework are now introduced. As shown in Figure 4.3, following variants are investigated in this chapter:

1. DPCo (Depth-Point Contrast), which is proposed in this chapter and learns *format invariance* between depth maps and point clouds.
2. DVCo (Depth-Voxel Contrast), which is proposed in this chapter and learns *format invariance* between depth maps and voxels.
3. PVCo (Point-Voxel Contrast), which learns *format invariance* between point clouds and voxels. It is extended from DepthContrast [285].
4. PPCo (Point-Point Contrast), which only uses point clouds as input. It serves as a baseline method as it only learns invariance against data augmentation.
5. IPCo (Image-Point Contrast), which learns *modality invariance* between RGB images and point clouds. It is inspired by Pixel-to-point [134].

6. PointContrast [260], which learns *perspective invariance* between view angles. It is a special case of the unified framework since it generates the crops C_1 and C_2 from two overlapping depth maps from different view angles and only considers the local correspondence.

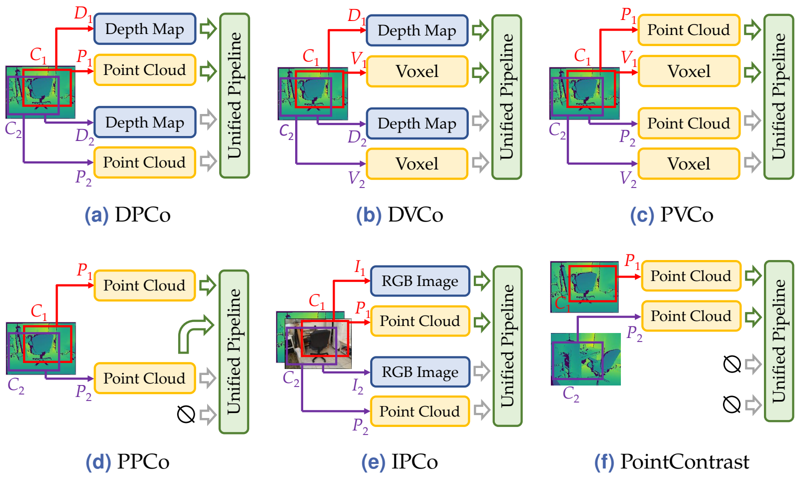


Figure 4.3 Contrastive learning strategies under a unified framework.

This chapter proposes contrasting a 3D and a 2D format of the same geometric data (*i.e.*, DPCo and DVCo). Although they represent the same 3D scene, the two formats are complementary to some extent. As discussed in Section 4.1, point clouds and voxels directly represent 3D geometry while having inevitable information loss due to sampling. On the contrary, depth maps reserve more information but only represent the 3D scene indirectly. Experiments show that the proposed methods perform significantly better than PPCo and PVCo, which contrast only 3D formats.

4.2.3 Implementation Details

The implementation details of the unified framework are provided in the following.

4.2.3.1 Point Cloud Encoder

A U-shaped PointNet++ [171] is used as a point cloud encoder, following the network configuration of Qi *et al*. [173]. The encoder consists of 4 down-sampling and 2 up-sampling modules. 20 000 points are used as input in the pre-training. The number of output points is fixed at 1024. More details on PointNet++ are presented in Section 2.2.5.

4.2.3.2 Voxel Encoder

A sparse residual U-Net [39] with 34 convolution layers is used to encode voxel inputs, following previous works [94, 260]. Point clouds are quantized in pre-training with the voxel size 2.5 cm. The output of the voxel encoder has the same resolution as the input. The concept of sparse 3D CNNs is provided in Section 2.2.4.

4.2.3.3 Depth Map Encoder

The U-shaped 2D CNN presented in Chapter 3 is employed as a depth map encoder for all experiments in this chapter. The network is a modified ResNet-34 [80] with relative depth convolution and additional up-sampling layers. Its input is resized and zero-padded to the resolution 352×352. The output is a feature map down-sampled with factor 8.

4.2.3.4 Color Image Encoder

Analogous to the depth map encoder, a ResNet-34 with extra up-sampling layers is employed to encode the RGB images. It is initialized with the pre-trained weights on ImageNet [44] provided in PyTorch [167], following the setup of Liu *et al*. [134].

4.2.3.5 Momentum Encoders and Projection Heads

The momentum encoders have the same structure as the encoders. Their weights are updated via exponential moving average (EMA) from the corresponding encoders instead of back-propagation. Global max pooling is used to aggregate a global feature. The pooling layer is followed by an MLP consisting of 3 fully connected layers. The channel number of the input layer is determined by the output dimension of the encoder. The intermediate and the output layer have 512 and 128 channels, respectively. The projection heads of momentum encoders are updated via EMA as well.

4.2.3.6 Loss Function

This chapter applies contrastive learning for self-supervised pre-training. The loss function consists of a local sub-loss L_l and a global sub-loss L_g. The local sub-loss is an InfoNCE loss [160] which optimizes the dense local correspondence:

$$L_\mathrm{l}^{\alpha\beta} = -\sum_i \log \frac{\exp\left(\mathbf{v}_i^\alpha \cdot \mathbf{v}_i^\beta / \tau\right)}{\exp\left(\mathbf{v}_i^\alpha \cdot \mathbf{v}_i^\beta / \tau\right) + \sum_{j \neq i} \exp\left(\mathbf{v}_i^\alpha \cdot \mathbf{v}_j^\beta / \tau\right)} . \tag{4.1}$$

If the corresponding 3D coordinates of feature vector $\mathbf{v}_i^\alpha$ and $\mathbf{v}_j^\beta$ are close, they are considered as a positive pair and have $i = j$ (*cf.* Section 2.4.3). The temperature τ is a hyperparameter empirically set to 0.07 in all experiments. All features are normalized using Euclidean norm before being fed into the loss function (*i.e.*, using the cosine similarity as the similarity measure in Equation 2.31).

The global sub-loss is applied to perform an instance discrimination task:

$$L_\mathrm{g}^{\alpha\beta} = -\log \frac{\exp\left(\mathbf{q}^\alpha \cdot \mathbf{k}^\beta / \tau\right)}{\exp\left(\mathbf{q}^\alpha \cdot \mathbf{k}^\beta / \tau\right) + \sum_{n=1}^{N_\mathrm{m}-1} \exp\left(\mathbf{q}^\alpha \cdot \mathbf{k}^{\beta,n} / \tau\right)} . \tag{4.2}$$

The vector $\mathbf{q}^\alpha$ refers to the global feature from the α encoder and $\mathbf{k}^\beta$ the global feature from the β momentum encoder. Since $\mathbf{q}^\alpha$ and $\mathbf{k}^\beta$ are generated from the same data sample, they compose a positive pair. Features $\mathbf{k}^{\beta,n}$ correspond to other samples (*i.e.*, negative samples) and are read from a memory bank with the size N_m. All experiments in this chapter use $N_\mathrm{m} = 2^{15}$.

Following previous works, the final loss is made symmetric to α and β, which is given by

$$L = \frac{1}{4}\left(L_\mathrm{l}^{\alpha\beta} + L_\mathrm{l}^{\beta\alpha} + L_\mathrm{g}^{\alpha\beta} + L_\mathrm{g}^{\beta\alpha}\right) . \tag{4.3}$$

4.2.3.7 Pre-Training Dataset

The ScanNet dataset [43] is used for pre-training, following previous works [36, 134, 135, 260, 285]. ScanNet is a large-scale indoor dataset that contains approximately 1500 scans reconstructed from 2.5 million RGB-D frames. This chapter follows the official train/validation split and samples 78 000 frames (one in every 25 frames) from the training set. More details about ScanNet are provided in Appendix B.1.3. Note that ScanNet is, in fact, a richly labeled dataset. However, no label is used in the pre-training stage.

4.2.3.8 Data Augmentation

In each iteration, C_1 and C_2 are randomly cropped. Also, a random square area in each crop is masked out. Standard data augmentations, *e.g.*, random rotation, scaling, translation, and flipping, are applied to the point clouds and voxels. Also, depth maps are randomly rotated around principal points, and 20 % pixels on the depth map are set to zero, following Chapter 3. Furthermore, random color jitter, grayscale, and Gaussian blur are used for color images.

4.2.3.9 Pre-Training Setups

All encoders are pre-trained for 120 epochs using the SGD (stochastic gradient descent) optimizer with a momentum of 0.9 and an initial learning rate of 0.03. Pre-training is performed on two NVIDIA Tesla V100 GPUs with a total of 64 GB memory. The batch size is chosen to be as large as possible. The batch size of different strategies varies from 32 to 64. The learning rate is reduced with a cosine schedule [139].

4.3 Experiments and Results

In this section, different contrastive learning strategies are compared and analyzed in detail under the proposed unified framework. This section first focuses on the point cloud-based object detection task to evaluate the pre-training performance. Then, transfer learning results on voxels and depth maps are demonstrated. This section also evaluates proposed methods on synthetic data and using different contrastive schemes.

4.3.1 Invariances in 3D Self-Supervised Pre-Training

This section first focuses on the performance of transfer learning on the point cloud-based 3D detection task since it reflects the encoder's capability of capturing both semantic (*i.e.*, object classification) and geometric information (*i.e.*, bounding box regression) and is thus representative. Also, 3D detection using raw points is well studied in previous works (*cf.* Chapter 3). Therefore, different strategies are applied to pre-train a PointNet++ [171]. The pre-trained weights are used to initialize a VoteNet [173], a representative 3D object detector (see Section 2.3.3.2 for more details). Then, the VoteNet is fine-tuned on the SUN RGB-D [212] and the ScanNet [43] object detection benchmark. The evaluation metrics are the mean average precision over multiple classes with the threshold of 25 % and 50 % 3D-IoU (*i.e.*, AP25 and AP50). Evaluation metrics for object detection are explained in Appendix C.2. Since the fine-tuning configuration is identical for all models, the performance of fine-tuned models indicates the effectiveness of pre-training.

4.3.1.1 Comparison under the Unified Framework

This experiment compares various contrastive learning strategies under the proposed unified framework. As shown in Table 4.1, all pre-training methods deliver better results than training from scratch in both 3D detection benchmarks. Note that the ScanNet benchmark uses point clouds reconstructed from multiple views. The unified framework, which assumes that the pre-training data are independent single depth maps or RGB-D images, still significantly improves the detection results on this dataset. It implies that the weights pre-trained on single-view data generalize well on multi-view data.

The baseline strategy PPCo utilizes the invariance against data augmentation solely. However, it surpasses PointContrast [260], which relies on extrinsic camera parameters, in two out of four metrics. It implies that with a proper design (in this case, the dense local contrast and the MoCo-style instance discrimination), the perspective invariance is unnecessary

Table 4.1 VoteNet fine-tuning performance of self-supervised pre-training strategies with different invariances. The results without pre-training are reproduced using the open-source code of Qi *et al.* [173]. They are slightly better than the original publication. All reported values are in percentage.

Method	**Invariance**	**Correspond.**	**SUN RGB-D**		**ScanNet**	
			AP25	**AP50**	**AP25**	**AP50**
From scratch	N/A	N/A	58.4	33.3	60.0	37.6
PPCo	augmentation	local+global	58.6	34.9	62.6	39.5
PointContrast	perspective	local	59.6	34.1	62.8	38.1
PVCo	format (3D-3D)	local+global	59.3	34.9	62.8	39.5
IPCo	modality	local+global	**60.2**	35.5	63.9	40.9
DPCo	format (2D-3D)	local+global	59.8	**35.6**	**64.2**	**41.5**

in pre-training. Zhang *et al.* [285] also report a similar observation. It is hypothesized that the network has to distinguish inputs from very similar view angles in the instance discrimination sub-problem, as training data are sampled from continuous RGB-D videos. This process can be interpreted as hard example mining, which forces the network to focus on perspective-relevant details. Thus, with the help of the global correspondence in pre-training, the encoders implicitly learn perspective-relevant information, but not necessarily the invariance in this case.

Moreover, PVCo, which contrasts features from point clouds and voxels, brings slightly better though very similar results as PPCo. It is due to the nature of point clouds and voxels, as they both represent 3D coordinates directly. Also, PointNet++ (see Section 2.2.5) and sparse 3D CNNs (see Section 2.2.4) have similar working principles, as they all perform local feature aggregation with shared weights and have a hierarchical topology with sub- and up-sampling. Thus, jointly pre-training voxel and point cloud encoders brings limited benefits to the point cloud encoder compared to pre-training them separately. In this case, incorporating voxel features can be regarded as a data augmentation applied to point clouds.

However, IPCo and DPCo, which contrast a 2D data format (*i.e.*, color images or depth maps) and a 3D format (*i.e.*, point clouds) achieve significantly better results than PPCo and PVCo, which utilize only 3D formats. It supports the intuition that 2D data formats are complementary to 3D formats, and the correspondence between them can provide strong contrast in self-supervised pre-training. More interestingly, the proposed method DPCo, which uses only the geometrical information, achieves on-par or better performance than the one using both geometrical and color inputs (*i.e.*, IPCo). It implies that the primary performance gains of IPCo do not come from the color information but from other factors, *e.g.*, different resolutions and perspective fields of 2D and 3D networks. Compared to IPCo, DPCo has the advantage that it is applicable even if the RGB images are unavailable or hard to align with depth maps. Furthermore, DPCo trains faster than PPCo and PVCo, thanks to the efficiency of 2D CNNs.

4.3.1.2 Local and Global Correspondence

The unified framework supports both the local and global correspondence of 3D data in the pre-training. The results in Table 4.1 are simultaneously affected by the two types of correspondences. In the following experiments, the contributions of each type of correspondence are investigated separately. As shown in Table 4.2 and Table 4.3, using local and global correspondence alone in the pre-training improves the performance of encoders. This observation demonstrates the advantage of contrasting features from 2D and 3D neural networks.

Table 4.2 Impact of different choices of local correspondences in pre-training.

Strategy	**SUN RGB-D**		**ScanNet**	
	AP25 (%)	**AP50** (%)	**AP25** (%)	**AP50** (%)
From scratch	58.4	33.3	60.0	37.6
PPCo	58.7	34.8	62.2	38.8
PVCo	59.1	34.6	62.2	39.0
PointContrast	59.6	34.1	62.8	38.1
IPCo	**60.1**	**35.6**	62.5	39.4
DPCo	59.6	35.1	**64.2**	**40.5**

Furthermore, comparing with Table 4.1, it is clear that combining them can bring further improvement, which is also observed in 2D pre-training, as discussed by Wang *et al*. [245]. Moreover, Table 4.2 and Table 4.3 show

Table 4.3 Impact of different choices of global correspondences in pre-training.

Strategy	**SUN RGB-D**		**ScanNet**	
	AP25 (%)	**AP50** (%)	**AP25** (%)	**AP50** (%)
From scratch	58.4	33.3	60.0	37.6
PPCo	59.3	35.1	62.7	39.3
PVCo	59.0	**35.3**	62.5	39.6
IPCo	**59.4**	34.5	63.3	40.2
DPCo	**59.4**	34.9	**63.8**	**41.0**

similar trends as Table 4.1, where IPCo and DPCo show superior performance over other methods. Interestingly, in Table 4.2 IPCo and DPCo achieve better results than PointContrast even without the global correspondence.

4.3.1.3 Summary

The insights obtained from the above-presented experiments can be summarized as follows:

1. Explicit perspective invariance in 3D self-supervised learning is unnecessary.
2. Format invariance between 3D formats (*e.g.*, point clouds and voxels) improves the performance, but the gains are marginal.
3. Format invariance between depth map and 3D formats (*e.g.*, depth maps and point clouds) significantly improves the performance. Moreover, it performs slightly better than modality invariance between point clouds and RGB images but has fewer requirements on the training data.

4.3.2 Comparison with State-of-the-Art Methods

In Tables 4.1, 4.2, and 4.3 the proposed method DPCo shows the best performance among all variants. It is further compared with other SOTA self-supervised pre-training methods. Still, the fine-tuning performance

Table 4.4 Fine-tuning results of VoteNet on the SUN RGB-D and the ScanNet (scan-level) object detection benchmark with different pre-training methods. The absent values are not reported in original publications. The reproduced results of PointContrast and pixel-to-point using PointNet++ as a backbone are reported, as the original publications use voxel-based backbones. Grayed methods use additional data or annotations (*i.e.*, unfair comparisons). Specifically, DepthContrast (×3) [285] uses a scaled PointNet++ backbone with more parameters and is pre-trained on both the ScanNet and the Redwood indoor RGB-D scan dataset [165].

Pre-Training	**SUN RGB-D**		**ScanNet**	
	AP25	**AP50**	**AP25**	**AP50**
From scatch	58.4	33.3	60.0	37.6
PointContrast [260]	-	34.8	-	38.0
PointContrast (reproduced)	59.5	34.0	61.6	38.2
Hou *et al*. [94]	-	-	-	39.2
Pixel-to-point [134]	57.2	33.9	59.7	38.9
Pixel-to-point (reproduced)	60.1	**35.6**	62.5	39.4
DepthContrast (×1) [285]	**60.4**	-	61.3	-
DPCo (proposed)	59.8	**35.6**	**64.2**	**41.5**
DepthContrast (×3) [285]	61.6	35.5	64.0	42.9
Supervised	62.0	36.3	61.9	38.6

in point cloud-based object detection tasks is used as the metric. Also, the setup in Section 3.3.3 is applied to obtain a strong supervised baseline. Specifically, bounding box annotations for single frames in the ScanNet dataset are generated using the scene-level labels. Then, a VoteNet is pre-trained with full supervision. For a fair comparison, the supervised baseline and other self-supervised methods use the same number of frames for pre-training.

In Table 4.4, the proposed method DPCo is compared with PointContrast [260], DepthContrast [285], pixel-to-point [134], and the method of Hou *et al*. [94], which have been already discussed in Section 2.4.3 and Section 4.2.1. As Table 4.4 shows, the proposed method outperforms other self-supervised pipelines in three metrics out of four. It even outperforms the fully supervised baseline on ScanNet with a higher AP25 and AP50. Also, DPCo has on-par performance on SUN RGB-D AP50 and ScanNet AP25 with the up-scaled version of DepthContrast [285], which uses a three times larger network and is pre-trained with five times more data. This result implies that the contribution of format invariance between point clouds and depth maps is comparable with scaling up the model capacity and the data amount. Furthermore, besides depth maps (or the equivalence, *e.g.*, range images) and camera intrinsics, which are available in almost all 3D datasets, the proposed method DPCo does not require any additional information, *e.g.*, color images and extrinsic camera parameters, while many SOTA methods do [134, 260, 285].

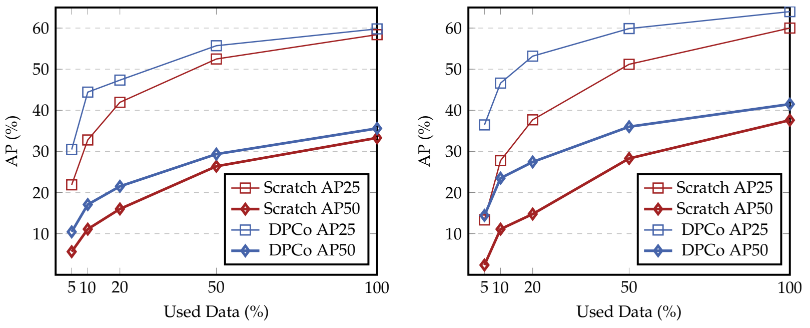
4.3.3 Label Efficiency

One important goal of pre-training is to transfer the features to very small datasets. To simulate this scenario, small partitions from the downstream datasets (*e.g.*, 5 % and 10 %) are sampled. A VoteNet with the backbone pre-trained by DPCo is then fine-tuned using these data. Experiments with the same percentage share the same training samples. The validation set is not sampled. As shown in Figure 4.4(a) and Figure 4.4(b), the pre-training brings more improvement when less fine-tuning data (as well as labels) are available. The trend is more evident on ScanNet, as it contains fewer training samples than SUN RGB-D (1200 *vs*. 5000 in total). Especially, the DPCo pre-training increases the AP25 on ScanNet from 13.3 % to 36.5 % and the AP50 from 2.4 % to 14.4 %, when only 5 % of fine-tuning data are used.

The experimental results demonstrate that self-supervised pre-training significantly improves label efficiency, as the same performance can be obtained using fewer labeled data.

4.3.4 Additional Transfer Learning Results

Previous experiments use the fine-tuning performance of point cloud-based object detectors to evaluate different pre-training methods. Additional transfer learning results with different networks, tasks, data, and contrastive methods are provided in the following.



(a) SUN RGB-D dataset **(b)** ScanNet dataset

Figure 4.4 Detection results with reduced data and label amount during fine-tuning.

4.3.4.1 Depth Map Encoders

In this experiment, a 2.5D-VoteNet is fine-tuned with its backbone initialized using pre-trained weights. The detector, which is a variant of VoteNet with a depth map-based backbone, is introduced in Chapter 3. To clarify the contribution of format invariance, a depth map encoder is pre-trained using only depth maps. This strategy is similar to PPCo in Figure 4.3 and is referred to as DDCo (Depth-Depth Contrast). Since 2.5D-VoteNet does not support multi-view input, it is only fine-tuned on the SUN RGB-D dataset.

Table 4.5 shows that the pre-training using DDCo degrades the performance. As a depth map is an indirect representation of 3D coordinates, DDCo might make the depth map encoder focus on the 2D textures instead of the actual 3D geometry, which can be regarded as over-fitting in the pre-training task. It also implies that the pre-training of depth map encoders is non-trivial and requires a careful design. However, the proposed methods DPCo and DVCo consistently improve the detection results. Since the point cloud and voxel encoders can capture 3D geometrical information by their nature, they can guide the depth map encoder and prevent it from paying too much attention to 2D patterns.

With the results in Table 4.4, it is worth noting that DPCo improves the 3D and 2D encoders at the same time. This proves that the principle of the proposed methods is different from knowledge distillation [87],

Table 4.5 Fine-tuning results of 2.5D-VoteNet on the SUN RGB-D dataset with different contrasting strategies.

Pre-Training	**AP25** (%)	**AP50** (%)
From scratch	60.8	36.9
DDCo	56.0	31.2
DVCo	61.0	**39.3**
DPCo	**61.4**	38.8

which uses a stronger model as a teacher to improve a weaker student model.

4.3.4.2 Voxel Encoders

To evaluate the proposed strategy on voxel-based networks, this experiment applies DVCo to pre-train a voxel encoder and fine-tunes it for semantic segmentation on the S3DIS [10] and the ScanNet [43] dataset. More details on the two datasets are given in Appendix B.1.1 and Appendix B.1.3, respectively. The evaluation metrics for semantic segmentation tasks are explained in Appendix C.1. The performance is compared with the not pre-trained baseline and PVCo. As shown in Table 4.6, DVCo significantly increases the mIoU (mean intersection over union) on both segmentation tasks. Also, the results are better than PVCo, consistent with the transfer learning results of point cloud encoders shown in Section 4.3.1.

Table 4.6 Fine-tuning results of a sparse 3D CNN in semantic segmentation tasks. The evaluation metric is mean IoU over multiple classes (mIoU).

Pre-Training	**S3DIS** (%)	**ScanNet** (%)
From scratch	66.1	69.6
PVCo	66.6	70.3
DVCo	**67.2**	**70.5**

Table 4.7 Overall classification accuracy on ModelNet40 [255] test set. All reported values are in percentage.

Network	Input	From Scratch	Pre-Trained
PointNet++ [171]	point clouds	88.6	90.4
Sparse 3D CNN [39]	voxels	88.1	89.2

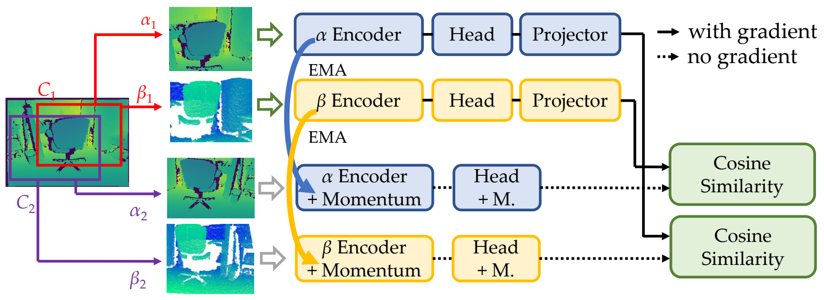
4.3.4.3 Object Classification on Synthetic Data

Till now, the pre-training and fine-tuning use real-world data from depth sensors. This experiment investigates whether the pre-trained features can generalize on synthetic data. To this end, a point cloud and a voxel encoder are pre-trained with DPCo and DVCo, respectively. Then, they are fine-tuned for object classification on the ModelNet40 dataset [255], which consists of ~9800 and ~2400 CAD models for training and testing, respectively. The dataset is explained in detail in Appendix B.2.1. For this experiment, the CAD models are converted into point clouds and voxels via pre-processing. Note that the PointNet++ and the sparse 3D CNN in this chapter contain up-sampling layers to increase the resolution. The up-sampling layers are abandoned for the classification task, and the global features are aggregated at the lowest scale level.

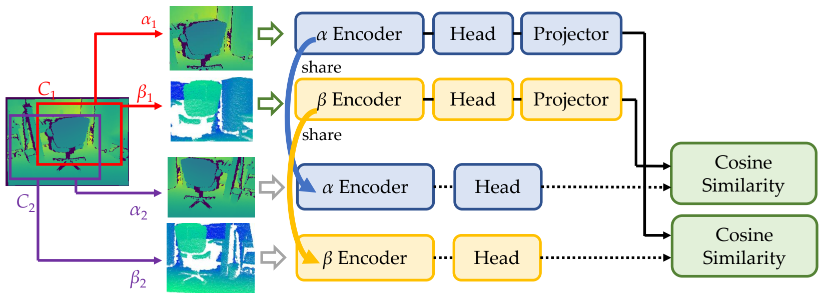
As shown in Table 4.7, the weights pre-trained using real-world indoor data improve the fine-tuning performance on synthetic CAD models. Despite the domain gap, the classification accuracy with point clouds and voxels is improved by 1.8 % and 1.1 % (absolute), respectively. Notice that the baseline performance is lower than the original PointNet++ publication [171], as the network configuration in this chapter is specialized for complex indoor scenes and is non-optimal for simple single-object synthetic data in ModelNet40.

4.3.4.4 Other Contrastive Learning Schemes

The pipeline introduced in Section 4.2 uses a contrastive loss consisting of a local and a global sub-loss. The global sub-loss follows the well-known MoCo [35, 82]. This experiment investigates whether the idea of jointly



(a) using BYOL [72]



(b) using SimSiams [33]

Figure 4.5 Depth-point contrast with other contrastive learning methods.

pre-training a 2D and a 3D encoder works with other contrastive learning methods *e.g.*, SimSiams [33] and BYOL [72] (more details in Section 2.4.3).

Compared to MoCo, BYOL applies the cosine similarity loss instead of the InfoNCE loss. Also, it uses a projector to break the symmetry and prevent the mode collapse. SimSiams further simplifies the BYOL pipeline by removing the momentum encoder and sharing weights of the Siamese networks (*i.e.*, two models with the same architecture). To address the mode collapse issue, SimSiams stops the gradient from back-propagation in one of the Siamese networks. In this experiment, SimSiams and BYOL are modified for jointly pre-training two distinct networks. As shown in Figure 4.5, the losses are calculated with features from different input formats, following the unified framework in Chapter 4.2.

A depth map and a point cloud encoder are jointly pre-trained using these methods. The point cloud encoder is then fine-tuned for 3D object detection. For simplicity, the local correspondence is not applied in this experiment. Besides the two methods mentioned above, this experiment also tests an even simpler end-to-end pipeline, which pre-trains the two encoders and projection heads end-to-end and optimizes the cross-format similarity directly.

Table 4.8 VoteNet fine-tuning results on point cloud-based object detection tasks. Only global correspondence is used in pre-training.

Pre-Training	**SUN RGB-D**		**ScanNet**	
	AP25 (%)	**AP50** (%)	**AP25** (%)	**AP50** (%)
From scatch	58.4	33.3	60.0	37.6
BYOL	58.2	32.5	62.7	40.2
SimSiams	58.6	33.6	62.4	39.9
End-to-end	58.5	34.0	62.5	39.7
DPCo (MoCo)	**59.4**	**34.9**	**63.8**	**41.0**

The results in Table 4.8 show that all pre-training methods improve the detection quality on the ScanNet benchmark. However, the proposed method DPCo, which follows the idea of MoCo [35, 82], achieves better results than other methods. It is because the pre-training data are extracted from continuous RGB-D videos and contain many similar (*i.e.*, hard) samples. To optimize the contrastive loss, the encoder of MoCo must maximize the similarity of anchors to positive samples and the dissimilarity to negative samples (*cf.* Chapter 2.4.3). On the contrary, other methods do not consider the dissimilarity and cannot benefit from hard samples. Moreover, other methods either bring marginal improvement or degrade the performance on the SUN RGB-D dataset. The SUN RGB-D benchmark is more challenging because it contains more noisy data and requires oriented bounding box predictions instead of axis-aligned ones. Another interesting result of this experiment is that the simple end-to-end method performs well in fine-tuning. Note that this method leads to mode collapse in the case of Siamese networks. To address this issue, previous works apply *e.g.*, momentum encoders, memory banks, unsymmetrical projectors, stop-gradient, and regularization [14, 33, 35, 72, 82]. Since two different encoders are applied with the end-to-end method, the problem is avoided without the bells and whistles.

4.4 Additional Comparison with Related Works

This section compares the contrastive learning methods in this chapter with additional works.

4.4.1 Hybrid Neural Networks for 3D Data

As explained in Section 2.2.9, some methods use hybrid models for 3D data understanding. They combine multiple architectures, *e.g.*, voxel-based and PointNet variants, in a single model. These methods and the proposed training strategy in this chapter share the same motivation to combine the advantages of different data formats and neural networks. Although the proposed method jointly pre-trains two encoders, the pre-trained encoders are used separately in downstream tasks.

4.4.2 Multi-Modal Feature Fusion

Learning from two complementary sources is also similar to data fusion. In 3D computer vision, fusing the color and geometry information is a common practice. A lot of fusing approaches have been proposed, *e.g.*, for object detection [93, 98, 172, 174, 213, 238, 265], as discussed in Section 3.5.2. Also, Liu *et al*. [135] use self-supervised pre-training to improve the fusion of geometric and color features. The difference between fusing and contrasting multi-modal features is that fusion enriches features by combining complementary information from different modalities, while contrastive learning maximizes the shared information between modalities.

4.4.3 Contrastive Learning using Outdoor Data

The pre-training methods proposed in this chapter are evaluated on indoor datasets. Some concurrent works also apply contrastive learning for self-supervised pre-training in outdoor scenarios (*i.e.*, autonomous driving) [132, 145, 198]. Especially, Sautier *et al.* [198] propose a pipeline similar to Pixel-to-point [134] (as well as IPCo), where features from a LiDAR point cloud encoder are contrasted with color features from a pre-trained 2D CNN. However, due to the architecture of the measurement system, LiDAR point clouds and RGB images in outdoor scenes have different resolutions and field-of-views. On the contrary, indoor datasets often use RGB-D cameras that directly capture registered depth maps and color images. Therefore, Sautier *et al.* [198] focus on addressing this issue and propose using superpixels to pool 3D and 2D features from visually similar regions.

4.5 Conclusions

This chapter establishes a unified framework to fairly compare the contribution of perspective, format and modality invariance in 3D self-supervised pre-training. Comprehensive experiments show that contrasting a 3D data format (*e.g.*, point clouds and voxels) with a 2D data format (*e.g.*, images and depth maps) is especially beneficial. Moreover, this chapter proposes contrasting point clouds or voxels with depth maps instead of RGB images, which brings better performance and has fewer requirements on the training data. The proposed method shows promising results in improving the label efficiency of 3D deep learning by exploiting unlabeled data.

Moreover, this chapter demonstrates that contrastive learning can pre-train various types of neural networks, *e.g.*, PointNet++, sparse 3D CNNs, and 2D CNNs. However, contrastive learning fails on transformer-based 3D neural networks (see Section 5.3.3.7). Therefore, Chapter 5 investigates the masked autoencoder [83] instead of contrastive learning to pre-train transformers for point cloud understanding.

Although better label efficiency can be obtained using contrastive learning, all methods in this chapter rely on real-world pre-training data. However, capturing 3D data in the real world is also costly and time-consuming. It inspires Chapter 6 to explore data generation technologies and apply synthetic data in self-supervised pre-training.

5 Plain Transformers for Real-World Point Cloud Understanding

The depth map-based method introduced in Chapter 3 shows good computational efficiency in 3D object detection. However, it is not applicable when the input is captured from multiple views (*i.e.*, multi-view point clouds). To overcome this limitation, this chapter revisits transformer-based architectures for point cloud understanding.

Due to the lack of inductive bias, transformer-based models usually require much training data (*cf*. Section 2.2.7). The problem is especially concerning in 3D vision, as 3D data are more difficult to acquire and annotate than the 2D counterpart. To address this issue, previous works modify the architecture of transformers to incorporate inductive biases by applying, *e.g.*, local attention and down-sampling. Although they have achieved promising results, earlier works on transformers for point clouds have two issues. First, the power of plain transformers is still under-explored. Second, they focus on simple and small point clouds instead of complex real-world ones. This chapter rethinks the application of plain transformers to real-world point clouds. It first takes a closer look at some fundamental components of plain transformers, *e.g.*, patchifier and positional embedding, for both efficiency and performance. To close the performance gap due to the lack of inductive bias and annotated data, it investigates self-supervised pre-training with masked autoencoder (MAE) [83]. Specifically, a new technique *drop patch* is proposed, which prevents an information leakage caused by position embedding and significantly improves the effectiveness of MAE. The proposed models achieve SOTA results in semantic segmentation on the S3DIS dataset [10] and object detection on the ScanNet dataset [43] with low computational costs. Meanwhile, this chapter provides a new baseline for future research on transformers for point clouds.

The main contents of this chapter have been published in [296].

5.1 Introduction

While transformers [235] have been the de facto standard for natural language processing (NLP) since they were proposed [46, 177], they have also shown promising performance in computer vision tasks in recent years [49, 137]. One of the most representative architectures is Vision Transformer (ViT) [49], which models an image as a patch sequence and extracts features using a *plain* transformer encoder. It is called plain since a ViT consists of stacked transformer layers and does not incorporate inductive biases, *e.g.*, translation equivariance and locality, which are, on the contrary, essential ingredients in CNNs (*cf.* Section 2.1.2). Although simple and effective, a plain transformer requires more training data or careful designs to gain comparable performance as CNNs in image processing [34, 49, 257].

Because of its global perceptive field and the capability to capture informative features, transformer-based methods are also attractive in point cloud understanding. Many methods have been proposed to utilize transformers in 3D vision tasks [54, 75, 100, 136, 248, 287]. Since 3D data and annotation are scarcer and more expensive than the 2D counterparts, which makes it hard to train plain transformers, previous works inject inductive bias by using, *e.g.*, hierarchical sub-sampling and local attention (*cf.* Section 2.2.7). Although they have achieved impressive results, a strong baseline, which shows the potential of plain transformers in point cloud understanding, is still missing. Meanwhile, multi-modal transformers have invoked research interest recently, as they unify language, vision, and audio understanding [12, 110, 178, 181]. Although the inductive bias improves performance for one specific modality, it usually cannot generalize to others [12]. Thus, a baseline of plain transformers for point clouds is necessary for future research on multi-modal models.

Another issue of previous works is the complexity of evaluation tasks. Many works [54, 62, 75, 163, 273, 279, 287] focus on either clean synthetic data, *e.g.*, the ShapeNet dataset [26] or single-object real-world data, *e.g.*, the ScanObjectNN dataset [232]. These tasks might be too simple to convincingly justify the network design and show the full potential of transformers, which are known to have a large model capacity [49]. Also, the design based on simple data might not generalize well on complex real-world point clouds, which limits the application in real-world tasks, *e.g.*, robotics and autonomous driving. Moreover, due to the quadratic complexity of the attention mechanism [235], plain transformers are usually computationally expensive for real-world 3D data. However, this problem could be neglected if solely small point clouds are studied.

To address these issues, this chapter revisits the design of plain transformers and evaluates proposed methods on complicated large-scale real-world point clouds. To narrow the scope of research, it focuses on transformers as backbones and does not consider the usage as task-specific necks or heads [136, 248]. While keeping the overall architecture plain, this chapter optimizes some components of transformers for point clouds, *e.g.*, the patchifier and position embedding. Existing patchifiers, *e.g.*, ball query and kNN (k-nearest-neighbor), are systematically compared. Meanwhile, this chapter introduces Farthest Point Clustering (FPC) to investigate the effect of non-overlapping patchifiers. Also, this chapter revisits the design of position embedding in transformers and proposes incorporating global information to describe the patches' position better. Furthermore, the self-supervised pre-training of the proposed models is explored. Based on the successful masked autoencoder (MAE) [83], this chapter introduces a novel method called *drop patch*. It suppresses the information leakage caused by the position embedding in the decoder by only reconstructing a proportion of unseen patches. The method significantly improves the effect of pre-training and reduces the computational cost.

The contribution of this chapter is many-fold:

1. It analyzes and optimizes some essential components of plain transformers, *e.g.*, the patchifier and position embedding, for more effective point cloud understanding.

2. It investigates MAE for 3D vision and proposes drop patch for better transfer learning results.

3. It focuses on complex real-world point clouds to evaluate the presented designs.

4. It shows that with proper designs and self-supervised pre-training, plain transformers can achieve SOTA results in real-world 3D object detection and semantic segmentation while being efficient.

5.2 Method

This section reviews the basic architecture of plain transformers for point clouds (Section 5.2.1). Then, it investigates two crucial but long-overlooked components in plain transformers, *i.e.*, the patchifier (Section 5.2.2) and position embedding (Section 5.2.3). Later, it shows how to pre-train the proposed models using self-supervision (Section 5.2.4).

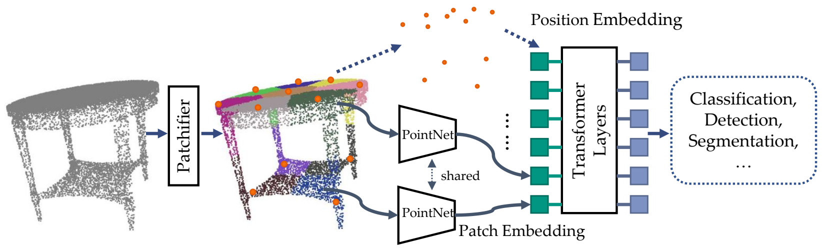


Figure 5.1 A plain transformer for point clouds. It simply uses stacked transformer layers without further modification.

5.2.1 Plain Transformers for Point Clouds

As shown in Figure 5.1, a plain transformer can be separated into five components: a patchifier, patch embedding, position embedding, a transformer encoder consisting of multiple transformer layers, and a task-specific head. The patchifier divides the input point cloud into small patches. The patch embedding encodes each point patch into a feature vector. A PointNet [170] is usually used for patch embedding [131, 152, 163, 273]. All patch features compose a sequence, which is then fed into the transformer encoder. Since the attention mechanism is permutation equivariant and unaware of the position of each patch, transformers require position embedding, which directly injects positional information into the sequence (*cf.* Section 2.1.3). The transformer encoder then extracts informative features which are further utilized by the task-specific head. More details on transformers and their application in 3D computer vision are provided as related works in Section 2.1.3 and 2.2.7, respectively.

5.2.2 Patchifier

This section first revisits the patchifiers in previous transformer-based models and explains why they should be researched. Then, a novel non-overlapping patchifier is introduced.

5.2.2.1 Background and Motivation

The process to build point patches (*i.e.*, patchify a point cloud) can be separated into *sampling* and *grouping*. Without loss of generality, the following explanation only considers inputs with 3D coordinates and ignores other channels, *e.g.*, colors, because they do not affect patchifying and are assigned to respective coordinates afterward [131, 152, 163, 273].

Given a point cloud $\{\mathbf{x}^i \,|\, \mathbf{x}^i \in \mathbb{R}^3\}_{i=1}^N$ with N points, the patchifier first down-samples M key points $\{\mathbf{s}^i \,|\, \mathbf{s}^i \in \mathbb{R}^3\}_{i=1}^M$ using farthest point sampling (FPS) [171]. Then, the patchifier searches K neighbors for each key point to build M patches $\{\Omega_i\}_{i=1}^M$ with $|\Omega_i| = K$. In previous works, ball query [152, 171] and k-Nearest-Neighbor (kNN) [131, 163, 175, 273] are used for grouping. The former searches K points in a sphere with a given radius around each key point, while the latter assigns K closest neighbors to each key point. Then, each patch Ω_i is encoded into a C-dimensional feature $\mathbf{f}^i \in \mathbb{R}^C$ using the patch embedding, which is usually a shared PointNet (*cf.* Section 2.2.5).

Despite the different choices of patchifiers, previous works usually use a large patch number M with $N \ll MK$. For instance, 3DETR [152] divides an input of 40 000 points into 2048 patches, which is an order of magnitude greater than a common ViT [49]. As the complexity of the attention mechanism is quadratic to the sequence length, it results in high computational costs, which limits the application of plain transformers to point cloud understanding, especially for large real-world ones. Also, the patchifiers in previous works generate overlapping patches. Although such a design can improve the stability of plain transformers [257], it causes information leakage during pre-training with MAE since the masked and reserved patches share some points (see Section 5.2.4).

The impact of shorter sequences and different choices of patchifiers have not drawn much attention in previous research. This chapter uses a shorter sequence with $N \approx MK$ to improve the computational efficiency of plain transformers. Also, different patchifiers are systematically compared with various setups. In addition to the two overlapping patchifiers mentioned above, this chapter also evaluates non-overlapping ones, *e.g.*, k-means and the proposed method Farthest Point Clustering (FPC).

5.2.2.2 Farthest Point Clustering

Algorithm 1: Farthest Point Clustering

```
Input  : A point set $\{\mathbf{x}^i\}_{i=1}^N$, number of patches $M$, number of samples in each
         patch $K$
Output : An assignment matrix $\mathbf{A} \in \mathbb{N}^{M\times K}$, where $A_{m,k} = i$ indicates that $\mathbf{x}^i$ is the
         $k$-th point in the $m$-th patch.
   /* Sample $M$ key points $\{\mathbf{s}^i\}_{i=1}^M$ from $\{\mathbf{x}^i\}_{i=1}^N$ using Farthest
      Point Sampling (FPS)                                                  */
1  $\{\mathbf{s}^i\}_{i=1}^N \leftarrow \mathrm{FPS}(\{\mathbf{x}^i\}_{i=1}^N)$
   /* find nearest key point $s_j$ for each point $x_i$                     */
2  foreach $\mathbf{x}^i \in \{\mathbf{x}^i\}_{i=1}^N$ do
3      $t_i \leftarrow \arg\min_j \{\|\mathbf{x}^i - \mathbf{s}^j\|\}$
4  end foreach
5  Initialize $\mathbf{A} \in \mathbb{N}^{M\times K}$ with zeros
   /* make sure each patch has $K$ points                                   */
6  for ($i \leftarrow 1; i \le M; i++$) do
7      $c \leftarrow 0$ // define a counter
8      for ($j \leftarrow 1; j \le N$ and $c < K; j++$) do
9          if $t_j = i$ then
10             $A_{i,c} \leftarrow j$
11             $c++$
12     end for
       /* check if the patch has $K$ points                                 */
13     $e \leftarrow K - c$
14     if $e > 0$ then
           /* duplicate points in each cluster                              */
15         for ($j \leftarrow 1; j \le e; j++$) do
16             $A_{i,c+j} \leftarrow A_{i,j}$
17         end for
18 end for
19 return $\mathbf{A}_{M\times K}$
```

The algorithm still uses FPS to sample M key points $\{\mathbf{s}^i\}_{i=1}^M$. The N input points are clustered into M patches by assigning each point $\mathbf{x}^i$ to its nearest key point $\mathbf{s}^i$. Note that, unlike kNN, each point is assigned to only one key point so that the generated patches do not overlap. The algorithm further samples K points in each cluster so that each patch has the same number of points, following ball query [171]. This algorithm's pseudo-code is provided in Algorithm 1.

5.2.3 Position Embedding

Position embedding is a mapping $\mathbb{R}^3 \to \mathbb{R}^C$, which encodes the coordinate of each key point into a vector:

$$\mathbf{e}^i = n_{\text{PosEmbed}}\left(\mathbf{s}^i\right) . \tag{5.1}$$

Previous works use Fourier features [152, 226] or multi-layer perceptron (MLP) [131, 163] as position embedding for point clouds. They all treat each position $\mathbf{s}^i$ separately, as formulated in Equation 5.1, and neglect the global information in all key points $\mathbf{s}^i$. While the "positions" in natural languages and images are fixed and shared across all data samples, they are content-dependent and more informative in point clouds, as shown in Figure 5.2. Therefore, the global information in position embedding might benefit point cloud understanding since it directly makes each patch aware of others' positions.

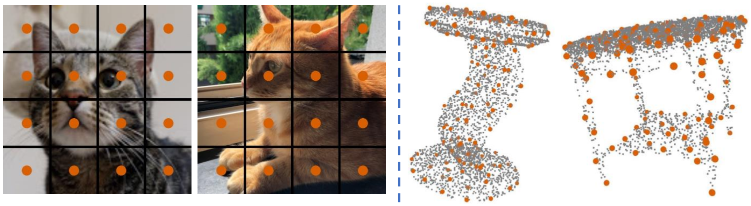


Figure 5.2 "Positions" of patches (orange dots) in different data. In images, they are independent of the content. The "positions" alone contain almost no information. In point clouds, "positions" are unique for each data sample and thus more informative, *i.e.*, one can know how the point cloud roughly looks like by only observing the "positions".

This chapter proposes a novel method to incorporate the global information. Specifically, each coordinate $\mathbf{s}^i$ is transformed into a high-dimensional space using an MLP. The global feature $\mathbf{g}$ is aggregated via global max pooling. The global feature is then concatenated to each coordinate and further transformed with another MLP. The proposed position embedding can be described as follows:

$$\mathbf{g}^i = \mathrm{MLP}_1\left(\mathbf{s}^i\right) , \tag{5.2}$$

$$\mathbf{g} = \mathrm{MaxPool}\left(\mathbf{g}^1, \dots, \mathbf{g}^i, \dots, \mathbf{g}^M\right) , \tag{5.3}$$

$$\mathbf{e}^i = \mathrm{MLP}_2\left(\mathrm{Concat}\left(\mathbf{g}, \mathbf{s}^i\right)\right) . \tag{5.4}$$

Then, $\mathbf{e}^i$ is added to its corresponding patch feature, following the common practice in previous works [235].

Note that in pre-training with MAE (Section 5.2.4), the global pooling in the encoder aggregates the global feature $\mathbf{g}$ only from visible patches. Thus, the pooling operation does not leak information about masked patches in pre-training.

5.2.4 Self-Supervised Pre-Training

This chapter uses masked autoencoder to pre-train transformers. The contrastive learning method introduced in Chapter 4 is unsuitable. Section 5.3.3.7 provides more discussion on contrastive learning.

5.2.4.1 Masked Autoencoder for Point Clouds

Masked autoencoder is explained as a related work in Section 2.4.4. Here, its basic concept is briefly revisited.

In an MAE [83], input patches $\{\Omega_i\}$ from a data sample are first randomly divided into two disjoint subsets $\left\{\Omega_i^{\mathrm{R}}\right\}$ and $\left\{\Omega_i^{\mathrm{M}}\right\}$. Patches $\left\{\Omega_i^{\mathrm{M}}\right\}$ are masked out, and the transformer encoder only sees the reserved patches $\left\{\Omega_i^{\mathrm{R}}\right\}$. With a transformer-based decoder, the model is trained to reconstruct the masked patches $\left\{\Omega_i^{\mathrm{M}}\right\}$ using features extracted from $\left\{\Omega_i^{\mathrm{R}}\right\}$. After pre-training, the decoder is abandoned, and the transformer encoder (with patch embedding, position embedding, *etc*.) can be used for downstream tasks. In the original publication, He *et al*. [83] suggest using a large mask ratio (*e.g.*, 75 %) for good performance.

However, for point clouds, MAE encounters two possible information leakage problems. On the one hand, the patches might overlap with each

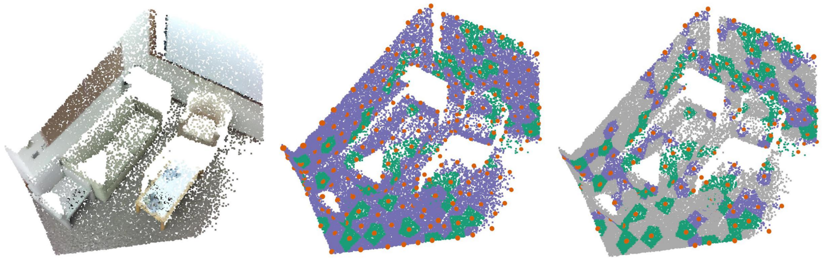

(a) Complete point cloud (b) Standard MAE (c) MAE with drop patch

Figure 5.3 Illustration of drop patch for point cloud MAE. Green patches are reserved. Purple patches are masked out and to be reconstructed. Grey patches are dropped and neglected by both the encoder and decoder. Orange dots are key points visible for the decoder of the MAE.

other, *i.e.*, $\left\{\Omega_i^{\mathrm{R}}\right\}$ might share points with $\left\{\Omega_i^{\mathrm{M}}\right\}$, which makes the pre-training less effective. MaskPoint [131] suggests using an extremely high mask ratio (*e.g.*, 90 %) as a workaround. With non-overlapping patchifiers, *e.g.*, k-means and FPC, the problem can be completely avoided. On the other hand, the decoder uses the positional information of both masked and reserved patches as queries. As discussed in Section 5.2.3, the position embedding of point clouds corresponds to the sub-sampled input (*i.e.*, key points) and leaks the positional information of the points to be reconstructed. In this case, reconstructing the masked patches is equivalent to up-sampling the key points and becomes trivial, as shown in Figure 5.3(b).

5.2.4.2 Drop Patch

To address the information leakage in the decoder, Liu *et al.* [131] discriminate if a randomly generated point is close enough to the original input point cloud instead of reconstructing masked patches directly. However, the method is still complex and has more hyperparameters (*e.g.*, the distance threshold and distribution of the random points) than the standard MAE.

In contrast, this chapter proposes an awkwardly simple but effective method to fix the information leakage. In each iteration, the input patches $\{\Omega_i\}$ are randomly split into three disjoint sets $\left\{\Omega_i^{\mathrm{D}}\right\}$, $\left\{\Omega_i^{\mathrm{R}}\right\}$, and $\left\{\Omega_i^{\mathrm{M}}\right\}$, instead of two. Then, patches $\{\Omega_i^{\mathrm{D}}\}$ are immediately dropped. The transformer decoder reconstructs $\{\Omega_i^{\mathrm{M}}\}$ by using features from $\{\Omega_i^{\mathrm{R}}\}$ and the positional information of both $\{\Omega_i^{\mathrm{M}}\}$ and $\{\Omega_i^{\mathrm{R}}\}$. This method is referred to as *drop patch*. With enough patches dropped, the decoder sees too few key points to perform the trivial up-sampling. This chapter uses $|\{\Omega_i^{\mathrm{D}}\}| : |\{\Omega_i^{\mathrm{R}}\}| : |\{\Omega_i^{\mathrm{M}}\}| = 2 : 1 : 1$, which is similar to the original MAE with a mask ratio of 75 %, as the encoder sees 25 % patches in both cases. The principle of drop patch is illustrated in Figure 5.3(c). Notice that drop patch also reduces the patches to be reconstructed and thus decreases the computation during pre-training.

5.2.4.3 Prediction and Loss Function

After the decoder, a fully connected layer is applied to generate a prediction. For each masked patch consisting of a key point and its K neighbors, the layer predicts K offsets from the key point to its neighbors. The Chamfer distance (*i.e.*, sum of the squared Euclidean distances between nearest neighbor correspondences of ground truth and predictions) is used as a loss function, following the standard practice in point cloud reconstruction [166, 249, 274]. Also, the loss function is only applied on masked patches, following He *et al.* [83].

5.2.5 Implementation Details

This subsection provides implementation details about pre-training and fine-tuning. Introductions of used datasets are provided in Appendix B.

5.2.5.1 Default Setup

This chapter uses a transformer encoder with 3 layers as the backbone if it is not otherwise specified. Each transformer layer has 256 channels and 4 heads, while the feed-forward networks have 512 channels. Unlike ViT, the proposed models use no class token [49]. All experiments employ an AdamW optimizer [140] with a weight decay of 0.01, the cosine annealing schedule [139], and gradient clip of 0.1. The training is warmed up for 10 epochs [139]. Other task-specific configurations are explained in the following.

5.2.5.2 Pre-Training

The MAE decoder consists of 2 transformer layers with multi-head self-attention (see Section 2.1.3). Each layer has 256 channels and 4 heads. The feed-forward dimension is 256.

As in Chapter 4, this chapter uses the ScanNet [43] dataset to pre-train the models. This dataset consists of 2.5 million frames of RGB-D images captured in 1513 indoor scenes. Every 25 frames are sampled from the training set, following previous works [94, 260] and Chapter 4. For each frame, 20 000 points are randomly sampled for pre-training. The patchifier divides each point cloud into 256 patches and samples 128 points in each patch (*i.e.*, $M = 256$, $K = 128$). The pre-training uses an initial learning rate of 5×10^{-4}. All models are pre-trained for 120 epochs with a batch size of 64.

Most previous object detectors [28, 152, 173, 260, 285] do not use color information, whereas the models for semantic segmentation methods do [39, 171, 176, 228, 260, 285]. Thus, the color channels are handled differently in the pre-training. For object detection, the models only use geometry information in pre-training. For semantic segmentation, models are pre-trained with both geometry and color. However, the color channels are not reconstructed using MAE, as it shows no significant effect on the fine-tuning performance.

Data augmentation is performed on the flight by randomly scaling, rotating, flipping, and cropping input point clouds. For color channels, random contrast and random grayscale are applied. Also, all colors of each point cloud are randomly dropped with the probability of 50 % (*i.e.*, color drop out).

5.2.5.3 Object Detection

This chapter adopts the detection pipeline from 3DETR [152], an end-to-end transformer-only detector consisting of 3 encoder layers and 8 decoder layers. The backbone of 3DETR is replaced by the above introduced plain transformer. Other configurations are as same as 3DETR. The detector is trained on the ScanNet dataset [43]. This chapter follows the official train/validation split and uses 1201 multi-view point clouds for training and 312 for validation. In each iteration, 40 000 points are randomly sampled from the original point cloud. The sampled point clouds are divided into 512 patches with 128 points. All models are trained for 1080 epochs with an initial learning rate of 5×10^{-4} and a batch size of 8. Metrics are mean average precision with 25 % and 50 % 3D-IoU threshold (*i.e.*, AP25 and AP50) over 16 representative classes.

More details on the baseline method 3DETR are provided in Chapter 2.3.3.3. The evaluation metrics for object detection are explained in Appendix C.2.

5.2.5.4 Semantic Segmentation

Since the segmentation task requires point-wise output, the features from the transformer encoder are up-sampled using nearest neighbor interpolation [171]. The point-wise features are further transformed by a shared MLP and fed into an MLP-based prediction head. The models are evaluated on the S3DIS dataset [10], which consists of real-world scans from 6 indoor areas. Following previous works, this chapter reports the validation results on Area 5 and trains models in other areas. Due to the large size of each point cloud, the raw point clouds are voxelized with a voxel size of 4 cm. However, the models do not contain voxel-based architecture (*e.g.*, 3D convolution). For each forward pass, 24 000 points are randomly cropped as input. The patchifier uses the hyperparameters $M = 512$ and $K = 64$. This chapter applies the same data augmentation as Qian *et al.* [176] for semantic segmentation. All models are trained for 300 epochs with a batch size of 16. Mean accuracy (mAcc) and mean intersection over union (mIoU) over 13 classes are used to evaluate the segmentation results. These metrics are explained in Appendix C.1.

5.3 Experimental Results

This section first demonstrates the performance of the proposed models in 3D object detection and semantic segmentation tasks. Then, it provides a detailed analysis of the design choices and properties of the proposed models.

5.3.1 Object Detection

The proposed models are compared with SOTA methods in object detection on ScanNet (Table 5.1). The methods in the upper half use PointNet++ or 3D CNN as the backbone. Most models are pre-trained using contrastive learning, as introduced in Section 2.4.5. The lower half of the table shows the results of transformer-based models. MaskPoint [131] is the most comparable method to the proposed ones, as it is also based on 3DETR and pre-trained using a variant of MAE.

Table 5.1 Object detection results on ScanNet V2 validation set. Pre.: pre-trained. Tr.: transformer-based. Mark ✓*: using local attention.

Method	**Pre.**	**Tr.**	**AP25** (%)	**AP50** (%)
VoteNet [173]			58.6	33.5
PointContrast [260]	✓		59.2	38.0
Hou *et al.* [94]	✓		-	39.3
4DContrast [36]	✓		-	38.2
DepthContrast (×1) [285]	✓		61.3	-
DepthContrast (×3) [285]	✓		64.0	42.9
DPCo (Section 4)	✓		64.2	41.5
3DETR [152]		✓	62.1	37.9
PointFormer [162]		✓*	64.1	42.6
MaskPoint (L3) [131]	✓	✓	63.4	40.6
MaskPoint (L12) [131]	✓	✓	64.2	42.1
Proposed (512 patches)				
– *from scratch*		✓	61.6	38.8
– *MAE*	✓	✓	62.7	42.2
– *MAE + drop patch*	✓	✓	64.1	43.0
Proposed (1024 patches)				
– *from scratch*		✓	62.4	41.3
– *MAE*	✓	✓	64.6	44.8
– *MAE + drop patch*	✓	✓	**65.6**	**45.3**

With 512 patches, the proposed detector without pre-training performs similarly to the original 3DETR with 2048 patches. This demonstrates that it is possible to use a much shorter sequence length without a significant performance drop. With MAE, the results are improved (+1.1 % AP25 and +3.6 % AP50, absolute), showing the power of pre-training. Drop patch further raises the AP25 by 1.4 % and AP50 by 0.8 %. The results of the proposed model with 512 patches (64.1 % AP25 and 43.0 % AP50) surpass the previous SOTA MaskPoint (L3 variant, *i.e.*, with 3 encoder layers) with a clear margin while showing similar performance as the heavy 12-layer variant.

Table 5.1 also evaluates transformers using 1024 patches. The model already surpasses 3DETR without pre-training. When pre-trained, it achieves 65.6 % AP25 and 45.3 % AP50, significantly outperforming previous works. Note that this chapter uses farthest point clustering for 512 patches, but ball query for 1024 patches since FPC brings sub-optimal results with a longer sequence. More discussions on patchifiers are provided in Section 5.3.3.1.

5.3.2 Semantic Segmentation

The semantic segmentation results on the S3DIS dataset are reported in Table 5.2. While the performance of the model trained from scratch is low (66.4 % mAcc and 60.0 % mIoU), pre-training with MAE improves the metrics by 7.2 % and 7.2 %, respectively. The gains are more significant than on the ScanNet dataset (Table 5.1). Since the S3DIS dataset is smaller than ScanNet (see Appendix B), results on this dataset benefit more from the pre-training. Also, drop patch further increases the mAcc and mIoU by 1.1 % and 0.4 %, respectively. When scaled up from 3 to 12 layers, the proposed model achieves significantly better results with 77.0 % mAcc and 70.4 % mIoU. This performance surpasses some highly optimized models, *e.g.*, PointTransformer [287] and PointNeXt [176]. It implies that self-supervised pre-training brings comparable improvement to architecture optimization.

Also, when pre-trained, the proposed models show on-par performance with SOTA transformer-based models that contain more inductive bias, *e.g.*, PointTransformer [287]. The observation demonstrates that self-supervised pre-training can close the performance gap caused by lack of inductive bias.

Table 5.2 Semantic segmentation on the S3DIS dataset Area 5. Pre.: pre-trained. Tr.: transformer-based. Mark ✓*: with modified transformers. The proposed models use 512 patches.

Methods	**Pre.**	**Tr.**	**mAcc** (%)	**mIoU** (%)
PointNet++ [171]			-	53.5
MinkowskiNet-32 [39]			71.7	65.4
KPConv [228]			72.8	67.1
PointNeXt-B [176]			74.3	67.5
PointNeXt-L [176]			76.1	69.5
Pixel-to-point [134]	✓		75.2	68.3
PointContrast [260]	✓		-	70.3
DepthContrast [285]	✓		-	**70.9**
PCT [75]		✓*	67.7	61.3
PatchFormer [278]		✓*	-	68.1
PointTransformer [287]		✓*	76.5	70.4
Pix4Point [175]	✓	✓	73.7	67.5
Proposed (3 layers)				
– *from scratch*		✓	66.4	60.0
– *MAE*	✓	✓	73.6	67.2
– *MAE + drop patch*	✓	✓	74.7	67.6
Proposed (12 layers)				
– *from scratch*		✓	70.0	63.2
– *MAE*	✓	✓	75.9	69.5
– *MAE + drop patch*	✓	✓	**77.0**	70.4

5.3.3 Analysis

Additional experimental results are provided to justify the designs in this chapter, *e.g.*, the choices of patchifier, position embedding, hyperparameters, and pre-training methods. Moreover, the computational costs and label efficiency of the proposed models are discussed.

5.3.3.1 Patchifier

Extensive experiments are conducted to clarify the impact of different patchifiers. Their interaction with position embedding, pre-training, and patch numbers is also researched.

As shown in Table 5.3, k-means achieves the worst performance with all setups. It is because k-means is sensitive to the spatial density of points. Since real-world point clouds are usually captured with depth sensors and the point density varies with depth, k-means lead to irregular patch sizes and is sub-optimal.

Table 5.3 Ablation study on patchifiers. Drop patch is applied for pre-training. Global information is used in position embedding. Group: grouping methods. *M*: number of patches. Pre.: pre-trained or not. PE: with position embedding or not.

ID	Group	*M*	Pre.	PE	AP25 (%)	AP50 (%)
1	Ball	512			59.8	37.9
2	kNN	512			60.8	38.0
3	k-means	512			59.5	36.3
4	FPC	512			60.3	38.1
5	Ball	512		✓	61.1	39.7
6	kNN	512		✓	61.7	41.0
7	k-means	512		✓	60.2	34.0
8	FPC	512		✓	61.6	38.8
9	Ball	512	✓	✓	63.4	42.1
10	kNN	512	✓	✓	63.7	42.4
11	k-means	512	✓	✓	62.7	38.7
12	FPC	512	✓	✓	**64.1**	**43.0**
13	Ball	1024		✓	62.4	41.3
14	kNN	1024		✓	63.5	39.9
15	k-means	1024		✓	59.0	36.6
16	FPC	1024		✓	61.6	36.9
17	Ball	1024	✓	✓	**65.6**	**45.3**
18	kNN	1024	✓	✓	65.0	43.5
19	k-means	1024	✓	✓	63.8	40.3
20	FPC	1024	✓	✓	64.6	44.3

When models are not pre-trained, the kNN patchifier achieves the best performance (experiment 2, 6, and 14). Similar results are also observed in image processing, where early convolutions improve the performance of a standard ViT [257]. However, when models are pre-trained with MAE, it is sub-optimal compared to FPC (experiment 10 and 12). Since kNN generates overlapping patches, it leaks the information of points to be reconstructed and thus degrades the effect of MAE.

FPC performs best when the patch numbers are small (*e.g.*, 512) and models are pre-trained. However, when it comes to 1024 patches, it is inferior compared to kNN and ball query. Since patches cannot overlap, FPC generates small and irregular patches in this case, which harms the performance.

Ball query outperforms other methods for large patch numbers (*e.g.*, 1024), because it guarantees a consistent scale and shape of patches, which helps models learn spatial features. Such an advantage is also reported by Thomas *et al*. [228]. However, ball query is sub-optimal for a small patch number (*e.g.*, 512) since it is difficult to set a suitable radius in this case. While the patch embedding cannot capture fine-grained details with a large radius, the patches cannot cover the entire point clouds with a small radius.

One can see that the performance of patchifiers is often affected by competing factors, which makes the optimal option of patchifiers conditional. Depending on patch numbers and pre-training, ball query, FPC, and kNN can deliver the best result. This chapter pays more attention to the performance of pre-trained models, as pre-training is crucial to compensate for the performance gap due to the lack of inductive bias. Thus, this chapter uses FPC for a smaller patch number ($M \leq 512$) and ball query for a larger patch number ($M > 512$).

5.3.3.2 Position Embedding

Different types of position embedding are systematically compared to clarify their impact. Besides Fourier features [226], MLP, and the proposed method with global information, models without position embedding in the transformer encoder are also evaluated as a baseline. Note that besides the transformer encoder, the decoder in MAE and the detection head in 3DETR also require position embedding. For simplicity, all experiments use the same type of position embedding in the transformer encoder, the MAE decoder, and the detection head. For variants without position embedding in the encoder, Fourier features [226] are applied in other components, following Misra *et al*. [152].

Table 5.4 Ablation study on position embedding. Drop patch is applied in pre-training. Group: grouping methods of patchifiers. M: number of patches. Pre: pre-trained or not. PE: type of position embedding. Add: the encoder layers where the position embedding is added.

ID	Group	M	Pre	PE	Add	AP25 (%)	AP50 (%)
1	Ball	512		-	-	59.8	37.9
2	Ball	512	✓	-	-	60.4	38.3
3	FPC	512		-	-	60.3	38.1
4	FPC	512	✓	-	-	59.7	37.2
5	FPC	512		Fourier	first	59.9	38.6
6	FPC	512		MLP	first	61.1	37.9
7	FPC	512		Global	first	61.6	38.8
8	FPC	512	✓	Fourier	first	61.6	40.9
9	FPC	512	✓	MLP	first	62.4	42.6
10	FPC	512	✓	Global	first	**64.1**	**43.0**
11	FPC	512		Fourier	all	60.3	38.6
12	FPC	512		MLP	all	60.7	39.0
13	FPC	512		Global	all	61.3	36.7
14	FPC	512	✓	Fourier	all	61.4	39.2
15	FPC	512	✓	MLP	all	61.4	38.6
16	FPC	512	✓	Global	all	63.3	42.0
17	Ball	1024		MLP	first	62.1	40.1
18	Ball	1024		Global	first	62.4	41.3
19	Ball	1024	✓	MLP	first	64.3	44.0
20	Ball	1024	✓	Global	first	**65.6**	**45.3**

Comparing experiment 3, 5, 6, and 7 in Table 5.4, it is obvious that Fourier features degrade the performance when trained from scratch, which is also observed in previous work [152]. On the contrary, MLP and the proposed method bring significant improvement compared to the variant without position embedding. Also, experiment 1-4 show that pre-training is ineffective if position embedding is not added. It is reasonable since the positional information of input patches is necessary for the reconstruction in MAE. On the other hand, experiment 8-10 show that position embedding makes the pre-training more effective. Meanwhile, experiment 5-10 demonstrate that parametric position embedding (*i.e.*, MLP and Global) performs better than the non-parametric Fourier features. Also, the proposed position embedding performs better than MLP, which supports the intuition in Section 5.2.3 that the global information in position embedding is beneficial. The results are consistent when a larger patch number is applied, as shown in experiment 17-20.

Another critical design choice is the location where the position embedding is added. While many previous methods add it to all encoder layers [131, 163, 273], experiment 11-16 show that this design degrades the performance. The contradiction is probably due to the domain gap between datasets. Since position embedding is more informative in point clouds, injecting it into all encoder layers makes a model pay more attention to the key points. Previous works mainly validate their designs on small point clouds (*e.g.*, ModelNet40 [255]). Such a behavior might be beneficial in this case since the overall shape is crucial. Nevertheless, for complex point clouds and tasks, the model might neglect fine-grained details. Thus, only injecting patch positions once performs better on real-world data.

5.3.3.3 Drop Patch

With the benefit of drop patch shown in Tables 5.1 and 5.2, additional experiments are conducted to study its hyperparameters. As explained in Section 5.2.4.2, drop patch addresses the issue that the position embedding of masked patches makes the MAE pre-training trivial.

MaskPoint [131] proposes using an extremely high ratio of masked patches (90 %). Experiment 2 and 3 in Table 5.5 show that it does not bring significant improvement because this approach aims to reduce the information leakage caused by overlapping patches. The information leakage caused by position embedding is still unsolved. In experiment 3-9, the percentage of reserved patches is fixed to highlight the impact of the drop ratio. With only 10 % patches dropped, the model already gains an improvement of 0.6 % AP25 and 0.2 % AP50. Also, the improvement becomes more significant with a higher drop ratio and reaches the maximum at 50 %. A very high drop ratio (60 % and 70 %) is sub-optimal since r_M is low, and the model receives less supervision in the pre-training.

Table 5.5 Ablation study on drop patch. r_D, r_M, r_R: the percentage of dropped, masked and reserved patches, respectively. All models use FPC patchifier with 512 patches.

ID	r_D	r_M	r_R	**AP25** (%)	**AP50** (%)
1	50	25	25	**64.1**	43.0
2	0	90	10	62.8	40.5
3	0	75	25	62.7	42.2
4	10	65	25	63.3	42.4
5	20	55	25	63.6	43.1
6	30	45	25	63.9	43.0
7	40	35	25	63.4	**44.3**
	50	25	25	**64.1**	43.0
8	60	15	25	63.8	42.4
9	70	5	25	63.2	41.2
10	50	10	40	63.6	40.2
11	50	20	30	63.8	43.2
	50	25	25	**64.1**	43.0
12	50	30	20	63.7	43.1
13	50	40	10	62.7	43.2

In experiment 10-13, the drop ratio is fixed. The best performance is achieved when r_M and r_R are approximately equal.

5.3.3.4 More Patches and More Layers

The following experiments investigate the impact of the number of encoder layers and patches, with the detection and segmentation head unchanged.

The upper half of Table 5.6 shows that more encoder layers harm the performance in object detection. Even though the models are pre-trained, only ~78 000 frames are available for pre-training. Since the detection head of 3DETR already consists of 8 transformer layers, an encoder with more layers leads to over-fitting. However, adding layers to the encoder improves the performance in segmentation tasks, as the used segmentation head is simpler and has fewer parameters.

Table 5.6 Impact of the number of encoder layers and patches. Models are pre-trained using MAE with drop patch.

Patches	Layers	ScanNet Detection		S3DIS Segmentation	
		AP25 (%)	**AP50** (%)	**mAcc** (%)	**mIoU** (%)
512	3	64.1	43.0	74.7	67.6
512	6	63.1	42.1	76.8	70.1
512	12	62.1	40.7	**77.0**	**70.4**
256	3	60.8	40.4	71.5	65.0
1024	3	**65.6**	**45.3**	73.5	67.1
2048	3	65.0	45.2	73.6	66.7

The lower half of Table 5.6 shows that using more patches is generally beneficial because it increases the computation of the networks without increasing the number of trainable parameters. However, the effect shows saturation at a large patch number (*i.e.*, 1024 for detection or 512 for segmentation).

5.3.3.5 Computational Costs

The computational costs of the proposed models are compared with SOTA methods. Models in Table 5.7 are all pre-trained on ScanNet with self-supervision. Memory usage and latency are measured with a batch size of 8, following previous works [152]. MaskPoint [131] uses 2048 patches. The proposed model with 512 patches performs similarly to MaskPoint (L12), with 5 times lower operations, 4 times less memory usage, and 4 times higher speed, which highlights the efficiency of the model design and the effectiveness of pre-training.

Table 5.7 also compares the proposed models with DPCo (introduced in Chapter 4). Here, DPCo is used to pre-train a VoteNet (*cf.* Section 2.3.3.2) with a PointNet++ backbone. The model introduced in this chapter is faster than DPCo but with a higher GFLOPs requirement. This is because PointNet++ (as well as similar architectures explained in Sections 2.2.5 and 2.2.6) requires a lot of random memory access since it hierarchically down-samples and up-samples point clouds [138]. On the contrary, a plain transformer only performs down-sampling once (*i.e.*, for building

Table 5.7 Comparison of computational costs in object detection. GFLOPs: Giga floating point operations for each forward pass. Memory: memory usage during training with a batch size of 8. Latency: inference latency with a batch size of 8 on an NVIDIA Tesla V100 GPU. The proposed models have 3 transformer layers in the encoder.

Method	**Operations**	**Memory**	**Latency**	**AP25**	**AP50**
	(GFLOPs)	(GB)	(ms)	(%)	(%)
DPCo (Chapter 4)	**5.7**	**6.6**	134	64.2	41.5
MaskPoint (L3)	21.4	17.3	187	63.4	40.6
MaskPoint (L12)	46.9	32.0	301	64.2	42.1
Proposed (M=512)	8.2	7.0	**73**	64.1	43.0
Proposed (M=1024)	11.7	8.7	108	**65.6**	**45.3**

patches). When scaled up to 1024 patches, the proposed model achieves significantly higher AP than previous methods with lower costs.

Table 5.8 reports the results of the semantic segmentation task. The proposed transformer-based models are compared with PointNeXt [176], which is a modernized variant of PointNet++ [171]. The proposed model with 3 encoder layers shows similar performance and throughput as

Table 5.8 Computational costs in semantic segmentation task. The same setup as Qian *et al.* [176] is used. GFLOPs: giga floating point operations for each forward pass. Parameters: number of parameters. Throughput: throughput during testing, with a batch size 16 on an NVIDIA Tesla V100. The proposed model uses 512 patches.

Method	**Operation**	**Parameter**	**Throughput**	**mAcc**	**mIoU**
	(GFLOPs)	(M)	(frames/s)	(%)	(%)
PointNeXt-S	**3.6**	**0.8**	**227**	70.7	64.2
PointNeXt-B	8.9	3.8	158	74.3	67.5
PointNeXt-L	15.2	7.1	115	76.1	69.5
Proposed					
– *3 layers*	6.0	1.9	147	74.7	67.6
– *6 layers*	7.2	3.5	138	76.8	70.1
– *12 layers*	9.7	6.7	123	**77.0**	**70.4**

PointNeXt-B. Also, the 12-layer variant achieves higher performance and is more efficient than PointNeXt-L. Note that PointNeXt models are not pre-trained but have a more optimized architecture and more inductive biases. These results imply that self-supervised pre-training can close the performance gap between plain transformers and highly optimized models.

The models introduced in this chapter are not compared with 2.5D-VoteNet from Chapter 3, because the transformers in this chapter are evaluated on multi-view point clouds. However, 2.5D-VoteNet only applies to single-view ones.

5.3.3.6 Label Efficiency

Figure 5.4 illustrates the performance of the proposed 3D detector with 3 encoder layers with reduced fine-tuning data and labels. The benefits from pre-training is more significant when less data are available in fine-tuning. Specifically, the pre-trained model achieves 37.7 % AP25 and 12.1 % AP50 with only 10 % annotated data. In contrast, the model trained from scratch reaches 7.4 % and 0.8 %, respectively. On the other hand, the pre-trained model performs similarly to the model trained from scratch but uses significantly less fine-tuning data and labels. For instance, it achieves 36.6 % AP50 with 50 % labeled data, while the model trained from scratch reaches 38.8 % with 100 % data.

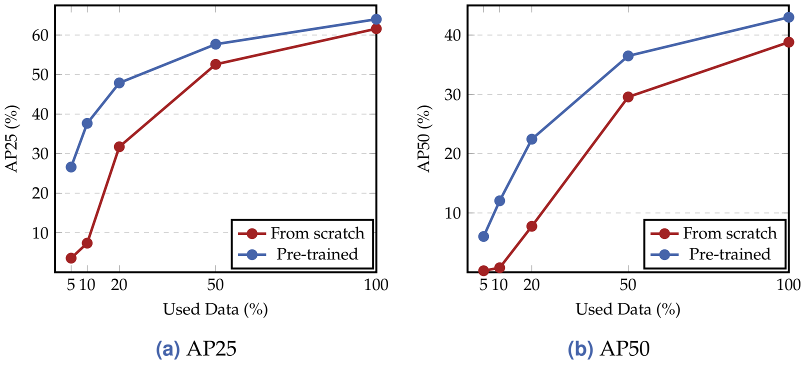


Figure 5.4 Detection results on the ScanNet dataset with reduced fine-tuning data and labels.

These results demonstrate that the self-supervised pre-training approach introduced in this chapter improves label efficiency.

5.3.3.7 Comparison with Contrastive Learning

Chapter 4 demonstrates the successful application of contrastive learning to self-supervised pre-training for 3D neural networks. It is possible to pre-train transformer-based models using contrastive learning. Table 5.9 compares the performance of MAE with MoCoV3 [34, 35, 82], a MoCo variant specialized for transformers, in point cloud-based object detection. As shown in Table 5.9, models pre-trained using contrastive learning underperform the randomly initialized one (*i.e.*, trained from scratch). The observation is also reported in 2D detection [123], where contrastive learning on ImageNet [44] degrades the fine-tuning performance of vision transformers [49] on the COCO detection benchmark [129]. However, the reason behind this observation is not fully understood.

Because of its superior performance, this chapter uses MAE instead of contrastive learning to pre-train transformers on point clouds.

Table 5.9 Comparison of MAE and MoCo with point cloud data. Fine-tuning results in object detection on the ScanNet dataset. Fixed Patch Embed.: parameters in patch embedding are fixed in pre-training, which stabilizes vision transformers during contrastive learning, proposed by Chen *et al*. [34].

Pre-Training	**Fixed Patch Embed.**	**AP25** (%)	**AP50** (%)
From scratch		61.6	38.8
MoCo		57.5	38.1
MoCo	✓	60.0	38.4
MAE		62.7	42.2
MAE + Drop Patch		64.1	43.0

5.3.3.8 Results on Synthetic Point Clouds

This chapter focuses on large real-world point clouds, while a lot of previous works explore the self-supervised pre-training for transformers on synthetic point clouds. In a standard pipeline, a transformer-based model is pre-trained on ShapeNet [26] and then fine-tuned for object

Table 5.10 Comparison of classification results on the ModelNet40 dataset. All models are pre-trained on ShapeNet. OA: overall accuracy on the test set. Proposed models are pre-trained with drop patch.

Method	**OA** (%)
PointBERT [273]	93.2
POS-BERT [62]	93.6
PointMAE [163]	93.8
MaskPoint [131]	93.8
Proposed (FPC)	93.6
Proposed (kNN)	93.8

classification on ModelNet40 [255]. Table 5.10 compares the proposed method with previous works.

The proposed models follow the standard setup using a transformer encoder with 12 layers. Each point cloud with 1024 points is split into 64 patches. As shown in Table 5.10, the proposed model using kNN achieves 93.8 % overall accuracy on ModelNet40, slightly better than the FPC variant. Among previous works, PointMAE and MaskPoint are most comparable to this chapter since they also apply plain transformers and MAE to point clouds. The former applies a standard MAE for point clouds, whereas the latter addresses the information leakage by learning an implicit function instead of reconstructing the masked patches. Despite different designs, MaskPoint and the proposed method (kNN) achieve the same accuracy as PointMAE on ModelNet40. On the contrary, they perform differently on real-world data, as shown in Table 5.1. This result implies that small synthetic point clouds, *e.g.*, ShapeNet and ModelNet40, are too simple to reveal the full potential of transformers, so a standard MAE (*i.e.*, PointMAE) already reaches the upper bound of the task. The benefit of further optimization is marginal or even unnoticeable.

This experiment can be regarded as the pilot study of this chapter. The result demonstrates that evaluating transformer-based models on more complex data and tasks is important. Therefore, synthetic point clouds are not the main focus of this chapter, although they are frequently researched in previous works.

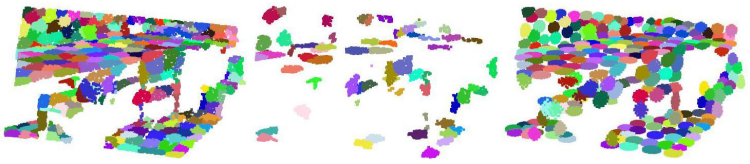
5.3.3.9 Reconstruction Results in Pre-Training

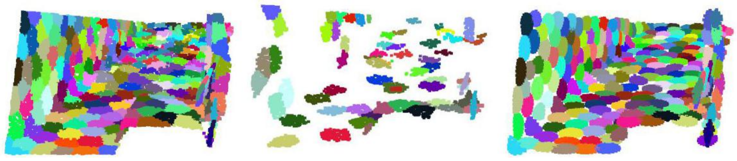
An MAE with drop patch is trained to reconstruct the complete point cloud from a masked one. Some representative reconstruction results are illustrated in Figure 5.5. Each patch is painted with a unique color. One can see that MAE mainly reconstructs the low-frequency information of point clouds. Also, the reconstructed patches usually show symmetries, although the ground truth patches have irregular shapes.

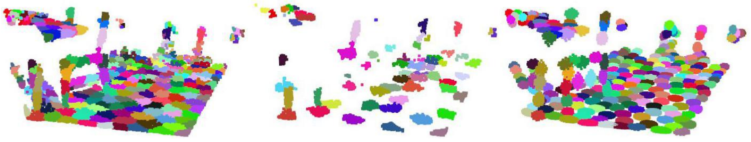
5.4 Conclusions

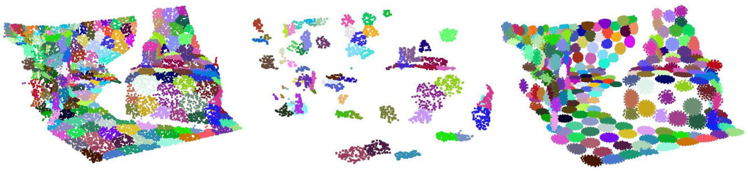
This chapter rethinks the application of plain transformers to point clouds. The proposed models show competitive performance and computational efficiency in 3D object detection and semantic segmentation. Also, the pipeline is label efficient since the models are pre-trained using self-supervised MAE. The experimental results also imply the necessity of evaluating transformers with real-world data, as the designs based on simple and small point clouds might not generalize well.

However, the self-supervised pre-training method in this chapter (*i.e.*, MAE with drop patch) still relies on real-world data (*i.e.*, the ScanNet dataset). To further reduce the cost of pre-training, Chapter 6 explores pre-training neural networks using synthetic data.


(a) A long table with chairs


(b) A corridor


(c) A table with chairs


(d) A bed

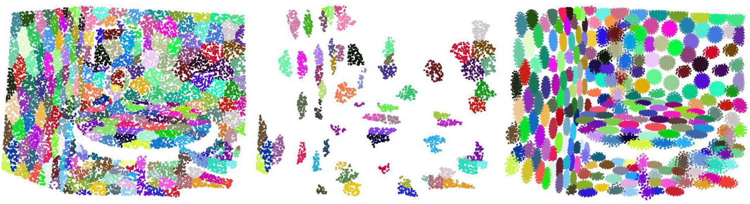

(e) A round table

Figure 5.5 Reconstruction results with point clouds. From left to right: original, masked and reconstructed point clouds, respectively. The mask ratio is set to 75 % for evaluation.

6 Efficient Pre-Training via Self-Supervision and Randomized 3D Scene Generation

Pre-training technologies are widely applied in previous chapters. Specifically, Chapter 3 discusses the fully supervised pre-training of object detectors. Chapters 4 and 5 explore self-supervised pre-training to avoid the labeling of large-scale 3D datasets. However, these approaches still rely on real-world data, which are laborious and time-consuming to capture. Therefore, pre-training 3D neural networks using previous methods in this thesis is still costly, even though data labeling is avoidable with self-supervision. To reduce the dependence on real-world data, previous works [179, 268] generate randomized 3D scenes for pre-training. Although the pre-trained models show promising performance boosts, previous works have two major shortcomings. Firstly, they focus on only one downstream task (*i.e.*, object detection), and the generalization to other tasks is unexplored. Secondly, an in-depth comparison of generated data is still lacking, and the impact of the data generation technology is not clear. This chapter systematically compares methods for randomized 3D scene generation using a unified setup. To study the generalization of the pre-trained models, this chapter evaluates their performance in multiple tasks, *e.g.*, object detection and semantic segmentation, and with different pre-training methods, *e.g.*, contrastive learning (*cf*. Chapter 4) and masked autoencoder (*cf*. Chapter 5). Moreover, this chapter proposes a new method to generate 3D scenes with spherical harmonics. It surpasses the previous formula-driven method with a clear margin and achieves on-par results with methods using real-world scans and CAD models. The main contents of this chapter have been published in [297].

6.1 Introduction

Deep neural networks are data-hungry, while capturing and labeling data requires significant time and human effort. This problem is especially concerning in 3D computer vision, as 3D data and labels are more scarce and expensive. To train strong 3D neural networks with a lower cost, many works apply synthetic data in pre-training and fine-tune the models on real-world data. One possible approach for generating synthetic data is simulation [45, 71, 101, 251] (see Section 2.5.1 for more details). Although realistic scenes can be simulated, developing the simulation environment, crafting the source materials, and designing scene layouts still require a lot of effort.

Recently, randomized 3D scene generation [179, 268] has shown promising results. This approach generates 3D scenes by randomly placing "objects", which can be CAD models [179] or formula-driven shapes [268], based on pre-defined rules. The concept of this approach is visualized in Figure 6.1. Randomized 3D scene generation requires neither real-world data nor manual annotation. Also, it reduces the effort in designing the scene layouts. However, previous works [179, 268] have two major issues.

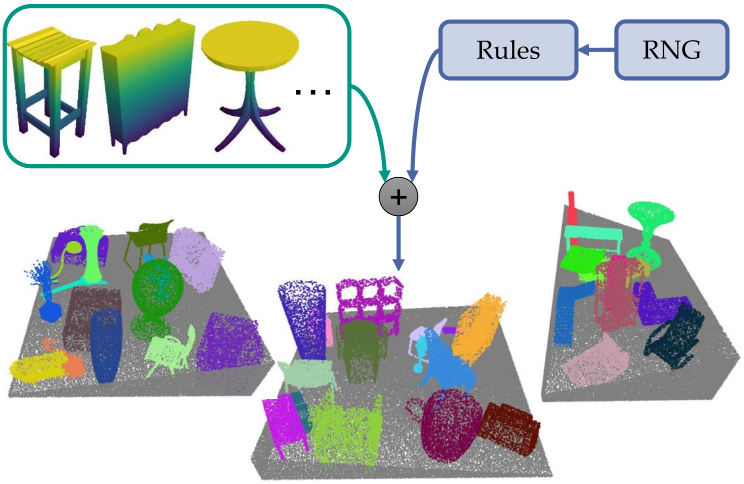


Figure 6.1 Concept of randomized 3D scene generation. Key components: an object set and generation rules. RNG: random number generator.

First, their pre-training methods are task-specific. Specifically, they are only designed and evaluated in the case of object detection, which limits their application to other tasks. Moreover, since they use different pretext tasks for pre-training, the contribution of the generated data is not clear. A fair comparison of different data-generating methods is still missing.

For a better understanding and application of randomized 3D scene generation in 3D computer vision, it is necessary to compare previous methods under a fair condition. Also, to pre-train models which can generalize to different downstream tasks, it is required to apply a general pre-training approach instead of task-specific ones. In this chapter, the masked autoencoder (MAE) [83] and contrastive learning [34, 35, 82] are used to pre-train models, as they generalize well and show impressive performance in Chapters 4 and 5. This chapter combines the self-supervision and randomized data generation to efficiently pre-train 3D neural networks. Experimental results demonstrate that randomly generated 3D scenes bring on-par improvement as real-world data. Also, the pre-trained models generalize well in different downstream tasks, *e.g.*, object detection and semantic segmentation.

Using formula-driven shapes, *e.g.*, fractal point clouds [268], instead of CAD models for scene generation is more promising since it does not rely on additional data and saves the cost of crafting or gathering CAD models. However, this chapter demonstrates that fractal point clouds lead to sub-optimal results in pre-training. One explanation is that the appearance of fractal point clouds is not close to real-world objects (see Figure 6.2(b)). This domain gap leads to inferior performance. Moreover, since real-world 3D data are usually captured with depth sensors (*e.g.*, laser scanners or RGB-D cameras), which cannot measure the internal structure of objects, real-world data only contain sample points on object surfaces. It implies that the objects used for scene generation should contain sufficient surfaces so that the pre-trained models can be transferred to downstream tasks with real-world data. Fractal point clouds, on the contrary, do not contain continuous surfaces (see Figure 6.2(b)). In addition to the overall appearance, this chapter hypothesizes that the lack of surfaces also makes the pre-training less effective. Instead of fractal point clouds, this chapter proposes using spherical harmonics, which are formula-driven shapes with natural surfaces. Also, their appearance is closer to real-world objects compared to fractal point clouds. Experimental results in this chapter show that spherical harmonics outperform fractal point clouds with a clear margin and are competitive compared to CAD models and real-world data in pre-training.

The contribution of this chapter is many-fold:

1. It explores generalizable pre-training using randomized 3D scene generation and systematically compares the generated data in multiple tasks.
2. Experimental results provide a deeper understanding of randomized 3D scene generation, *e.g.*, the impact of object sets and view angles.
3. It proposes generating scenes with formula-driven spherical harmonics. The models pre-trained using this approach achieve competitive results in 3D object detection and semantic segmentation without using real-world data or CAD models in the pre-training stage.

6.2 Method

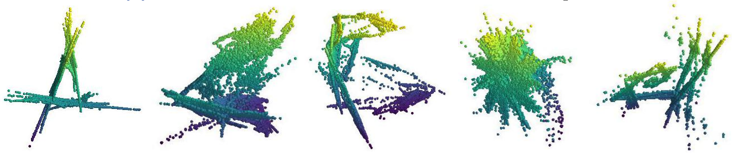
This section first briefly revisits the concept of randomized 3D scene generation. Then, it presents details on the data generation process, including a novel method to create formula-driven shapes using spherical harmonics. Finally, it explains the self-supervised pre-training with generated data.

6.2.1 Concept of Randomized 3D Scene Generation

As shown in Figure 6.1 and explained in Section 6.1, randomized 3D scene generation requires an object set and pre-defined rules. A room with a random size is first created to generate a new scene. A 3D object is randomly picked from the set, undergoes data augmentation (*e.g.*, rotation and scaling), and is randomly placed in the room. The process is repeated until the room contains enough objects. The rules define *e.g.*, the distribution of room size, choices and parameters for data augmentation, distribution of objects in the room, and the number of objects in each scene. With an object set and rules given, a vast amount of scenes can be easily generated.

Previous works use similar rules while having different choices on object sets. Rao *et al*. [179] use CAD models, which is straightforward since openly accessible datasets of CAD models are commonly used in 3D computer vision [26, 255]. This choice has the drawback that creating or gathering CAD models is still laborious and time-consuming. On the contrary, Yamada *et al*. [268] propose generating the object set in a randomized manner. Specifically, they create fractal point clouds by randomly sampling affine transformations and iteratively applying them to 3D points. However, this method is sub-optional as the fractal point clouds are not close to real-world objects and do not have surfaces (see Figure 6.2(b)).

(a) CAD models from the ModelNet40 dataset [255]


(b) Fractal point clouds [268]

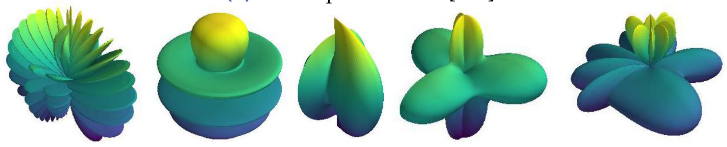

(c) Spherical harmonics

Figure 6.2 Examples of objects used for scene generation. Some objects are shown as meshes, although only sampled points are used for pre-training.

6.2.2 Spherical Harmonics

This chapter suggests generating object sets using spherical harmonics, which can be represented in a spherical coordinate system with radial distance r, polar angle θ, and azimuthal angle ϕ:

$$r = \sin(m_1\phi)^{p_1} + \cos(m_2\phi)^{p_2} + \sin(m_3\theta)^{p_3} + \cos(m_4\theta)^{p_4}\,, \tag{6.1}$$

where $m_i \in \mathbb{R}$ and $p_i \in \mathbb{Z}^+$ with $i \in \{1, 2, 3, 4\}$. With fixed coefficients m_i and p_i, Equation 6.1 describes a closed surface in 3D space. Diverse 3D objects can be generated when the coefficients are randomly set. Some generated spherical harmonics are visualized in Figure 6.2(c). Spherical harmonics are initially employed to solve partial differential equations [155]. In computer vision, they are also applied to describe surfaces and shapes [107, 197]. The form in Equation 6.1 is motivated by the original definition but does not strictly follow it. In this chapter, the mathematical and physical meanings of spherical harmonics are not considered. They are only used to parameterize the object set. This idea is also inspired by a web page written by Paul Bourke[1]. Notice that the generated spherical harmonics can be easily represented as (rectangular) meshes when coordinates ϕ and θ are sampled with constant intervals. The mesh representation greatly simplifies further processing, *e.g.*, point sampling and ray-casting (explained Section 6.2.4).

6.2.3 From Objects to Scenes

This chapter assumes all scenes are static, following previous works. Also, only downstream tasks in indoor scenes are considered, while extending the methods to outdoor scenarios is straightforward. This chapter adopts the generation rules of Rao *et al.* [179] and represents generated 3D scenes as point clouds. Each randomly picked object is normalized into a unit sphere. Then, 3000 points are randomly sampled and undergo random data augmentation. Each generated scene contains 12 to 16 objects. Since random scaling is used as data augmentation, the point density on each object might differ. To make the point density consistent across each scene, grid sampling (*i.e.*, voxelization) with the voxel size of 0.04 m is

[1] http://paulbourke.net/geometry/sphericalh/. Last accessed on 2023.04.03.

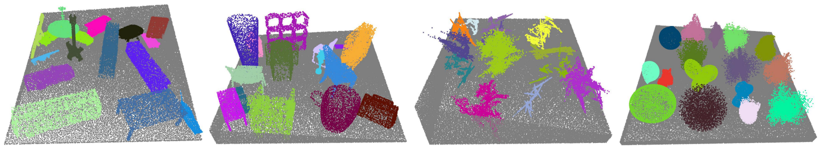


Figure 6.3 Examples of generated 3D scenes in point cloud representation. From left to right: using CAD models from ShapeNet [26], CAD models from ModelNet40 [255], fractal point clouds, and spherical harmonics, respectively. The colors here are solely for visualization purposes. The generated synthetic point clouds do not contain color channels.

applied to the point clouds. The same rules are employed for all object sets (*e.g.*, CAD models and fractal points). Some generated scenes are visualized in Figure 6.3. More details on generation rules are provided in Section 6.4.2.

In this chapter, single point clouds are generated independently. On the contrary, Rao *et al.* [179] create pairs of point clouds with object-level correspondence, and Yamada *et al.* [268] additionally generate bounding box labels for each scene.

6.2.4 Single-View Point Clouds

Previous works represent the generated scenes as multi-view point clouds. In this thesis, a point cloud is referred to as multi-view when it cannot be projected to one surface without information loss. A typical example of multi-view point clouds is a 3D scan reconstructed from data sequences, *e.g.*, using the SLAM (simultaneous location and mapping) technology. On the contrary, many point clouds in practice are single-view and represented as depth maps (*e.g.*, from depth cameras) or range images (*e.g.*, from rotational laser scanners). Besides comparing different methods for randomized scene generation, it is also meaningful to compare the synthetic data with real-world ones. A lot of works use single-view real-world data for self-supervised pre-training [94, 260, 264, 285, 295] since they are the direct output from sensors and easy to obtain.

To clarify the impact of single-view and multi-view representations in pre-training and for a fairer comparison with the real-world data, this chapter also studies single-view point clouds. Specifically, meshes are used as objects to create scenes instead of sampled point clouds. The same

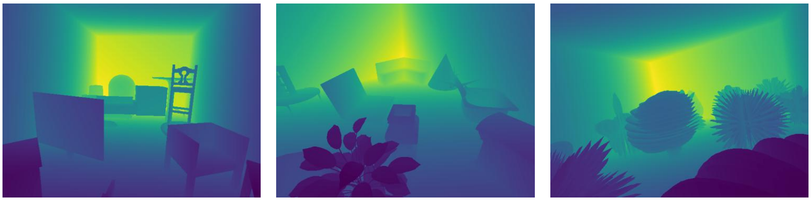


Figure 6.4 Examples of generated depth maps via ray-casting. From left to right: with objects from ShapeNet [26], with objects from ModelNet40 [255], with spherical harmonics, respectively. According to the rules of Rao *et al.* [179], objects might be stacked on others. Depth values are normalized for visualization.

rules of Rao *et al.* [179] are used to generate 3D scenes. Then, a virtual camera with a random pose captures a depth map using ray-casting [190]. The camera pose is checked so that each depth map contains sufficient objects (empirically set to 7 in this chapter). In pre-training, the depth maps are converted into point clouds on the flight. Data augmentations, *e.g.*, cropping the depth maps, and scaling and rotating the point clouds converted from them, are also applied.

Some generated depth maps are visualized in Figure 6.4. No depth maps are generated with fractal point clouds since ray-casting is non-trivial in this case.

6.2.5 Self-Supervised Pre-Training

To pre-train 3D neural networks, Rao *et al.* [179] create pairs of scenes with the same objects but different layouts and perform contrastive learning using object-level correspondence. Yamada *et al.* [268] generate bounding box labels for each object and pre-train object detectors in a fully supervised manner. Both methods are only evaluated in object detection tasks.

On the contrary, this chapter explores generalizable pre-training using generated synthetic data. Instead of using a pre-training strategy specific to one downstream task, this chapter applies masked autoencoder (MAE) [83] and MoCoV2 [35], following Chapters 4 and 5, to evaluate the generated data. The concepts of the two pre-training methods are briefly revisited in the following.

MAE [83] is employed to pre-train vision transformers [49, 235]. An input point cloud is divided into small patches, and a large proportion is masked. A transformer-based encoder extracts patch-wise features from the remaining patches, and a decoder is trained to reconstruct the masked patches. After the pre-training, the decoder is abandoned, and the encoder can be transferred into downstream tasks. This chapter adopts the architecture and training method in Chapter 5 for MAE.

Also, scene-level contrastive learning is applied to pre-train a point cloud encoder following Chapter 4. Two overlapping regions are cropped from each generated scene to create a positive pair (*i.e.*, an anchor and a positive sample). Point clouds from other scenes are regarded as negative samples. Given a point cloud as an anchor, the encoder is trained to distinguish its positive sample from many negative samples. This chapter applies MoCoV2 [35] by using an exponentially moving averaged momentum encoder and a memory bank to save global features from negative samples.

6.3 Experiments

This section first introduces the experimental setups in this chapter. Then, it demonstrates the main results by comparing different synthetic datasets. Furthermore, the proposed methods are compared with state-of-the-art methods using synthetic or real-world data for pre-training. Finally, additional experimental results are provided to justify the design choices and demonstrate the generalization of the proposed methods.

6.3.1 Setups

Experimental setups used in this chapter are explained in the following.

6.3.1.1 Real-World baseline

To create a baseline using real-world data, the baseline models are pre-trained on ScanNet [43], a large-scale indoor dataset captured in approximately 1500 rooms. 78 000 frames are sampled (approximately one in every 25 frames or one per second) from the raw data in the training set, following setups in Chapters 4 and 5.

6.3.1.2 Scene Generation

To evaluate the impact of different object sets, four object sets are used to generate synthetic data, summarized in Table 6.1.

ShapeNet [26] and ModelNet40 [255] are used as CAD model sets. The former consists of ~50 000 models of ordinary everyday objects (*e.g.*, chairs and shelves), while the latter contains ~10 000. This chapter follows their official train/test split and uses only objects from training sets. More details on the two datasets are provided in Appendix B.2.1 and Appendix B.2.2, respectively.

In the default setup, 10 000 formula-driven objects are used to generate scenes, so the object number is aligned with ModelNet40. The configuration of Yamada *et al*. [268] is used to create fractal point clouds. For spherical harmonics, the coefficients m_i and p_i in Equation 6.1 are constrained in range $[-5, 5]$ and $[0, 4]$, respectively.

78 000 scenes are generated with each object set in the default setup, following the real-world baseline. The generated datasets are named RM-ShapeNet, RM-ModelNet, RM-Fractal, and RM-Harmonics according to object sets. The abbreviation RM stands for *RandomRooms* [179], as the generation rules are adopted from this work.

Table 6.1 Summary of generated datasets.

Dataset	Object Set	Object Number
RM-ShapeNet	ShapeNet [26]	~50 000
RM-ModelNet	ModelNet40 [255]	~10 000
RM-Fractal	fractal points [268]	10 000
RM-Harmonics	spherical harmonics	10 000

6.3.1.3 Masked Autoencoder

This chapter uses the same setup to pre-train and fine-tune masked autoencoders as Chapter 5. Specifically, a 3-layer transformer encoder [235] extracts features from input patches. A decoder with 2 layers is attached to the encoder for pre-training. All models are pre-trained for 120 epochs with a batch size of 64 and an AdamW optimizer [140].

The pre-trained encoder is evaluated for 3D object detection on the ScanNet [43] benchmark. 3DETR [152] is used as the detection head and is fine-tuned for 1080 epochs with batch size 8. The mean average precision with a 3D-IoU threshold of 25 % and 50 % (*i.e.*, AP25 and AP50) over 18 classes are evaluation matrices. This chapter also fine-tunes the pre-trained transformer for semantic segmentation on the S3DIS [10] dataset. The segmentation head introduced in Chapter 5 is connected to the pre-trained encoder. The entire network is fine-tuned for 300 epochs with batch size 12. Mean accuracy (mAcc) and mean IoU (mIoU) over 13 classes in Area 5 are reported as metrics.

6.3.1.4 Contrastive Learning

This chapter pre-trains a PointNet++ [171] using contrastive learning, following previous works [179, 268, 285] and Chapter 4.

The network architecture is adopted from Qi *et al*. [173], which consists of 4 down-sampling modules [171] and 2 up-sampling modules [171]. The model is pre-trained using the well-known MoCoV2 pipeline [35]. Chapter 4 investigates multiple strategies using different invariances of 3D features (*cf.* Section 4.2.2). For simplicity and better comparability with previous works in literature, this chapter primarily uses the most basic strategy Point-Point Contrast (PPCo) with only global correspondence. However, the evaluation results of Depth-Point Contrast (DPCo) are also reported in Section 6.3.4.5.

The PointNet++ is pre-trained for 120 epochs using an SGD optimizer, an initial learning rate of 0.01, a batch size of 12, and the cosine annealing schedule [139]. After pre-training, a VoteNet [173] is initialized with the pre-trained model and is fine-tuned for 3D object detection on the ScanNet [43] and the SUN RGB-D [212] benchmark. The setup is the same as in Chapter 4.3.1. This chapter reports AP25 and AP50 as evaluation metrics.

6.3.2 Main Results

The generated datasets in Table 6.1 are used for pre-training models using masked autoencoder and contrastive learning.

Table 6.2 Downstream task performance of an MAE pre-trained on different datasets. Detection results are on the ScanNet detection benchmark. Segmentation results are on S3DIS dataset. The second column refers to the pre-training data. All metrics in percentage.

		Detection		Segmentation	
Pre-Training Data	**View**	**AP25**	**AP50**	**mAcc**	**mIoU**
N/A (Chapter 5)	-	61.6	38.8	66.4	60.0
ScanNet (Chapter 5)	single	**64.1**	43.0	**74.7**	67.6
RM-ShapeNet	multi	63.8	42.6	74.6	**67.8**
RM-ModelNet	multi	63.5	42.0	73.6	67.0
RM-Fractal	multi	62.8	40.4	70.2	64.1
RM-Harmonics	multi	63.8	**43.2**	74.1	67.1
RM-ShapeNet	single	63.8	42.9	73.8	67.2
RM-ModelNet	single	62.5	41.6	73.5	66.3
RM-Harmonics	single	63.5	42.1	73.5	67.1

6.3.2.1 Masked Autoencoder

First, an MAE is pre-trained using the generated data, and the fine-tuning performance is reported in Table 6.2. Compared to results without pre-training (row 1), all pre-trained models gain a significant improvement in object detection and semantic segmentation, showing the benefit of self-supervised pre-training.

Among multi-view variants of randomly generated data (row 3 to 6), using ShapeNet as an object set (*i.e.*, RM-ShapeNet) delivers the best performance since ShapeNet is significantly larger than other object sets (see Table 6.1). Meanwhile, RM-Fractal brings the slightest improvement. It supports the intuition that fractal point clouds as objects are non-optimal in data generation due to their unrealistic appearance and lack of surfaces. Interestingly, with the same object number, the RM-Harmonics dataset outperforms RM-ModelNet. The result is promising since it implies that randomly generated objects can replace CAD models for scene generation. However, most CAD models in ModelNet40 are ordinary objects in daily life, *e.g.*, chairs and tables. At the same time, both datasets in downstream tasks are captured in indoor environments, *e.g.*, living rooms and offices. One should expect ModelNet40 to perform better than spherical harmonics because of the similarity between pre-training and fine-tuning data. The diversity of the object set might cause this contradiction. Although ModelNet40 contains 40 classes, the appearance of the objects in the same class is similar. Also, the class distribution in ModelNet40 is long-tailed (*cf*. Appendix B.2.1), which might also have a negative impact [229]. On the other hand, spherical harmonics are generated by uniformly sampling the coefficients, which results in a more diverse and balanced object set.

Furthermore, Table 6.2 compares the results of generated synthetic data with the real-world ScanNet dataset (row 2). With the same number of point clouds, the performance of RM-ShapeNet and RM-Harmonics is close to real-world data, demonstrating the effectiveness of randomized scene generation in self-supervised pre-training.

With generated single-view data (*i.e.*, depth maps), a similar trend can also be observed: RM-ShapeNet performs the best, while RM-Harmonics is more effective than RM-ModelNet. Data generation with fractal points is not performed since ray-casting relies on mesh data. However, compared to their corresponding multi-view variants, all single-view datasets perform worse with MAE.

6.3.2.2 Contrastive Learning

For contrastive learning, a PointNet++ is pre-trained, and a VoteNet is fine-tuned using it as the backbone. Object detection results on ScanNet and SUN RGB-D are reported in Table 6.3. Generally, pre-trained models achieve better detection quality than the baseline without pre-training on both benchmarks. For the multi-view datasets (row 3 to 6), RM-ShapeNet and RM-Harmonics show on-par performance. RM-ModelNet performs worse than these two datasets, while RM-Fractal brings the smallest improvement in pre-training. Compared to real-world data from ScanNet[2], all randomly generated data shows significantly worse results on the SUN RGB-D dataset, while the gap disappears on the ScanNet detection benchmark.

[2] Due to technical issues, this chapter uses different hardware and batch size to Chapter 4. Therefore, the results of ScanNet using PPCo slightly differ in the two chapters.

Table 6.3 Fine-tuning results of contrastive learning using generated data. Evaluation for 3D object detection on SUN RGB-D and ScanNet benchmark. All metrics in percentage. Absent values are not reported in the original publication.

		SUN RGB-D		**ScanNet**	
Pre-Training Data	**View**	**AP25**	**AP50**	**AP25**	**AP50**
N/A [173]	-	58.4	33.3	60.0	37.6
ScanNet	single	**60.8**	**35.7**	62.2	39.0
RM-ShapeNet	multi	59.8	34.1	62.7	37.9
RM-ModelNet	multi	59.3	33.7	62.3	37.3
RM-Fractal	multi	59.1	33.4	61.3	38.6
RM-Harmonics	multi	59.6	**35.7**	62.6	39.7
RM-ShapeNet	single	58.9	33.0	**64.1**	**40.0**
RM-ModelNet	single	57.9	32.4	63.4	39.8
RM-Harmonics	single	58.6	33.1	63.5	39.8
RandomRooms [179]	multi	59.2	-	61.3	-
PC-FractalDB [268]	multi	59.4	33.9	61.9	38.3

The relative performance of single-view variants of generated data is similar to the multi-view ones, *e.g.*, RM-ShapeNet > RM-Harmonic > RM-ModelNet. However, the detection results on the SUN RGB-D dataset are worse compared to using multi-view data. On the contrary, single-view datasets show significantly better results on the ScanNet benchmark. This observation is counter-intuitive. Note that SUN RGB-D consists of single-view point clouds, whereas the ScanNet benchmark applies point clouds reconstructed from multi-views. One should expect the models pre-trained using single-view data to perform worse on the ScanNet benchmark due to the domain gap. The reason for the observation is not fully understood. However, it must be correlated with the pre-training schema since it is not observed with MAE.

Table 6.3 also compares the generated datasets with two previous works, *i.e.*, RandomRooms [179] and PC-FractalDB [268], each applies a training strategy specialized for object detection. Both methods use VoteNet as the detector and apply PointNet++ as the backbone. The model pre-trained with RM-Fractal achieves similar performance as PC-

FractalDB [268], which also generates scenes using fractal point clouds. However, RM-Harmonics (multi-view) outperforms both previous works by a clear margin. It shows that the choice of object sets has a larger impact than pre-training methods. Also, it implies that a task-specific design is unnecessary in pre-training, and general purpose methods (*e.g.*, MAE and MoCoV2) are sufficient.

6.3.2.3 Discussion

The experimental results in Tables 6.2 and 6.3 can be summarized as follows:

1. Randomized scene generation is effective for pre-training 3D models with self-supervision. This approach generalizes well across different pre-training methods and downstream tasks. Also, its performance is comparable with real-world data.
2. Formula-driven spherical harmonics perform significantly better than fractal point clouds and achieve on-par results as hand-crafted CAD models.
3. The diversity of objects impacts the effect of pre-training. An object set with a larger diversity is generally beneficial.
4. Single-view and multi-view variants of generated data bring different results. However, the impact is inconsistent across different pre-training methods and fine-tuning datasets.

The observation that randomly generated data have a similar effect as real-world ones in self-supervised pre-training is interesting since the synthetic data do not look photo-realistic (Figures 6.3 and 6.4), especially when formula-driven objects are applied. It is because neural networks perform different tasks in self-supervised pre-training and supervised fine-tuning (*i.e.*, pre-training and downstream tasks are decoupled). Therefore, the performance in fine-tuning is not sensitive to the domain gap between pre-training and fine-tuning data. Nevertheless, a large gap can still harm the performance, *e.g.*, fractal point clouds perform worse than spherical harmonics in experiments.

6.3.3 Comparison with other Pre-Training Methods

Table 6.4 compares the fine-tuning performance of models pre-trained on RM-Harmonics with other state-of-the-art methods. All methods indicated with “real” are pre-trained on real-world data from ScanNet, while DepthContrast (×3) in Table 6.4 additionally uses data from the larger Redwood indoor RGB-D scan dataset [165]. All methods in the upper half of Table 6.4 fine-tune a VoteNet with a pre-trained PointNet++ as the backbone, while methods in the lower half use a 3DETR with a pre-trained transformer.

Compared with methods using real-world data for pre-training, the proposed RM-Harmonics achieves competitive performance. However, pre-training using RM-Harmonics is more efficient since the data are randomly generated and require neither real-world 3D scans nor hand-crafted CAD models.

Table 6.4 Object detection results on the ScanNet detection benchmark. Data: type of pre-training data (real-world, synthetic or none). Model: the architecture of backbones. P: PointNet++, T: transformer. All reported values are in percentage.

Methods	Data	Model	AP25	AP50
VoteNet [173]	N/A	P	60.0	37.6
Hou *et al*. [94]	real	P	-	39.3
DepthContrast (×1) [285]	real	P	61.3	-
DepthContrast (×3) [285]	real	P	64.0	42.9
DPCo (Chapter 4)	real	P	**64.2**	41.5
RandomRooms [179]	syn.	P	61.3	-
PC-FractalDB [268]	syn.	P	61.9	38.3
RM-Harmonics (multi-view)	syn.	P	62.6	39.7
RM-Harmonics (single-view)	syn.	P	63.5	39.8
3DETR [152]	N/A	T	62.1	37.9
MaskPoint (L3) [131]	real	T	63.4	40.6
MaskPoint (L12) [131]	real	T	**64.2**	42.1
Plain Transformer (Chapter 5)	real	T	64.1	43.0
RM-Harmonics (multi-view)	syn.	T	63.8	**43.2**
RM-Harmonics (single-view)	syn.	T	63.5	42.1

6.3.4 Analysis

Experiments are conducted to demonstrate the impact of colors, object, and scene numbers on the pre-training. Also, the label efficiency of models pre-trained on synthetic data are analyzed. Furthermore, the generalization of the proposed methods to stronger models and DPCo (Chapter 4) is discussed.

6.3.4.1 Pseudo Color in Pre-Training

Since simulation and rendering are not involved, the scenes generated in this chapter do not contain color information. However, color channels are crucial in some downstream tasks, *e.g.*, semantic segmentation. To transfer models trained on colorless point clouds to such tasks, this chapter applies pseudo colors and data augmentation to color channels in pre-training. Specifically, all color channels are set to a constant value (*i.e.*, 0.5), and random color jitter is applied to each point. Also, color values of all points are simultaneously set to 0 with the probability 0.5 (*i.e.*, color drop out).

Experiments are conducted to analyze the effect of this approach. As shown in Table 6.5, when no data augmentation (*i.e.*, jitter and drop out) is applied to color values, the constant pseudo color leads to significantly worse results with RM-Harmonics. Table 6.5 also shows the results of

Table 6.5 Impact of colors in pre-training using MAE. Fine-tuned for semantic segmentation on the S3DIS dataset and evaluated in Area 5. The first two columns refer to the pre-training data. Const.: constant RGB values for all points. DA: data augmentation on color channels.

Pre-Train Dataset	**Colors**	**mAcc** (%)	**mIoU** (%)
N/A	N/A	66.4	60.0
RM-Harmonics	const.	69.8	62.8
RM-Harmonics	const. + DA	74.1	67.1
ScanNet	const.	69.8	63.6
ScanNet	const. + DA	73.1	67.0
ScanNet	real	**74.7**	**67.6**

models pre-trained on ScanNet to compare the performance of pseudo colors and real-world colors. Without data augmentation on pseudo colors, the performance of pseudo colors is much lower than real-world colors. However, applying data augmentation significantly shrinks the gap. Interestingly, when both use pseudo colors, RM-Harmonics slightly outperforms ScanNet in pre-training.

Results in Table 6.5 show that pseudo colors can be applied to pre-train models requiring colors on colorless point clouds. Also, data augmentation is essential for improving performance. This chapter hypothesizes that models learn color invariance via data augmentation and focus on the geometric information in the pre-training. In contrast, color-dependent information is learned in fine-tuning where real-world colors are available.

6.3.4.2 Object and Scene Numbers

This chapter proposes generating scenes using spherical harmonics as objects. The following experiments investigate the impact of object and scene numbers. All experiments are conducted using MAE.

The number of used spherical harmonics in scene generation is first fixed (*i.e.*, 10 000), and the scene number is scaled. As shown in the upper half of Table 6.6, using more scenes in pre-training is generally beneficial. However, there is no further improvement when more than 78 000 scenes (×1) are applied.

Furthermore, the scene number is fixed to the default value, and the object number is scaled. With a small object number (×0.2), the generated scenes have a smaller diversity, which harms the effect of pre-training. However, a large object number (*e.g.*, ×5) produces significantly worse results. Also, scaling up the scene and object numbers simultaneously (the last two rows) does not bring improvement compared to the default setup. It is probably due to the limited diversity of spherical harmonics described by Equation 6.1. Since the equation has only 8 coefficients, with a too large object number, the minimal “distance” between generated

Table 6.6 Impact of object and scene numbers. All experiments use multi-view point clouds generated with spherical harmonics. Models are evaluated for object detection on ScanNet. ×1: 10 000 objects or 78 000 scenes, respectively (*i.e.*, default numbers). ×*a*: scaling with factor *a*.

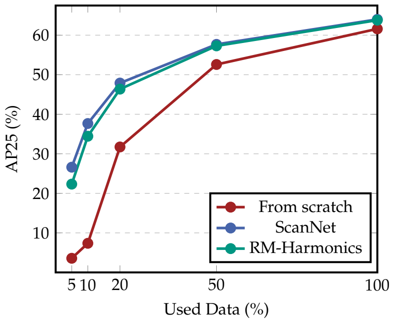
Objects	**Scenes**	**AP25** (%)	**AP50** (%)
N/A	N/A	61.6	38.8
×1	×0.2	62.9	41.1
×1	×0.5	63.5	42.7
×1	×1	63.8	43.2
×1	×2	63.8	**43.4**
×1	×5	63.4	42.9
×0.2	×1	62.4	41.8
×0.5	×1	**63.9**	42.8
×1	×1	63.8	43.2
×2	×1	63.2	41.7
×5	×1	62.5	40.6
×2	×2	63.2	42.5
×5	×5	63.0	42.2

objects becomes smaller[3]. As a result, some generated scenes contain very similar objects, and the model might learn a bias and expect objects in a scene to be similar. Even though a real-world scene can contain similar objects (*e.g.*, chairs from the same producer), their appearance in real-world 3D data is usually different because objects are often obscured differently, and the data are subject to measurement noises. Therefore, such a bias hurts the effectiveness of pre-training.

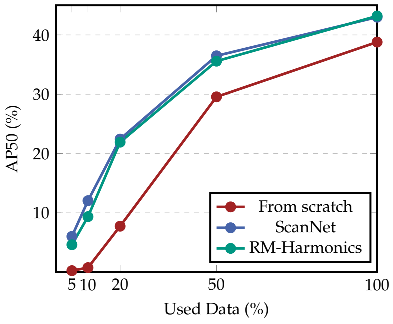
Based on the discussion, searching for new parameterized shapes with more degrees of freedom is beneficial. The exploration is left to future works.

[3] Consider the situation where n points in a continuous interval [0, 1] have to be independently and uniformly sampled. The minimal distance between two points decreases with increasing n.

6.3.4.3 Label Efficiency



(a) AP25



(b) AP50

Figure 6.5 Detection results of MAE with reduced fine-tuning data and labels.

Label efficiency is one focus of this thesis. Sections 4.3.3 and 5.3.3.6 evaluate models pre-trained on real-world data with reduced annotated fine-tuning data. They demonstrate that self-supervised pre-training can improve label efficiency. A similar experiment is conducted using synthetic data from RM-Harmonics. Specifically, an MAE is pre-trained on the generated RM-Harmonics dataset. Then, it is fine-tuned on ScanNet for object detection using 5 %, 10 %, 20 %, 50 %, and 100 % data, respectively.

Experimental results are visualized in Figure 6.5. Pre-training with RM-Harmonics shows a similar effect as ScanNet. Also, the improvement from pre-training is more significant when fewer data are used in fine-tuning. However, when fewer annotated samples (*e.g.*, 5 % and 10 %) are available, the performance gap between the two pre-training datasets is larger. It is understandable since the generated scenes are not as realistic as ScanNet. The model requires more real-world samples to adapt to the downstream dataset. However, the performance gap is closed when the detector is fine-tuned with more annotated data (*e.g.*, 20 %).

This experiment shows that pre-training using randomly generated data improves label efficiency. It can be interpreted from two aspects. On the one hand, with the same amount of annotated (fine-tuning) data, pre-trained models achieve better performance. On the other hand, the pre-trained model gains the same results with less annotated data compared to models trained from scratch.

6.3.4.4 Pre-Training Stronger Models

This chapter adopts the plain transformer in Chapter 5. The transformer encoder consists of 3 layers in the default setup. Section 5.3.3.4 demonstrates that using more layers improves the performance in semantic segmentation tasks. This experiment investigates if the randomly generated synthetic data apply to models with a larger capacity.

As Table 6.7 shows, pre-training using RM-Harmonics also improves the performance of the 12-layer model compared with the model trained from scratch. The result implies that the proposed method can generalize to stronger models. However, the improvement is smaller than the real-world ScanNet dataset. It is because the scenes in RM-Harmonics are not photo-realistic and contain no color information (*cf*. Section 6.3.4.1). Also, the difference (1.8 % mAcc and 1.9 % mIoU) is more significant than the 3-layer variant (0.6 % mAcc and 0.5 % mIoU) since models with a larger capacity generally benefit more from pre-training.

Table 6.7 Comparison of semantic segmentation results with different model capacity. Evaluated on the S3DIS dataset Area 5. All models use 512 patches.

Encoder Layers	**Pre-Train Data**	**mAcc** (%)	**mIoU** (%)
3 layers	N/A	66.4	60.0
3 layers	ScanNet	74.7	67.6
3 layers	RM-Harmonics	74.1	67.1
12 layers	N/A	70.0	63.2
12 layers	ScanNet	77.0	70.4
12 layers	RM-Harmonics	75.2	68.5

6.3.4.5 Pre-Training using Depth-Point Contrast

As explained in Section 6.3.1.4, Section 6.3.2.2 investigates contrastive learning using only point clouds for better comparability with previous

Table 6.8 Detection results of VoteNet pre-trained using different data. Models are pre-trained using DPCo introduced in Chapter 3.

Pre-Training Data	**SUN RGB-D**		**ScanNet**	
	AP25 (%)	**AP50** (%)	**AP25** (%)	**AP50** (%)
N/A	58.4	33.3	60.0	37.6
ScanNet	59.8	35.6	64.2	41.5
RM-Harmonics	58.8	34.1	63.7	40.4

works. However, Chapter 4 proposes a more advanced method DPCo, pre-training a point cloud encoder and a depth map encoder simultaneously.

To clarify if randomized 3D scene generation works with DPCo, a PointNet++ is pre-trained with the setup in Chapter 4 but using depth maps from RM-Harmonics. Then, a VoteNet is fine-tuned on the SUN RGB-D and the ScanNet dataset for object detection, using the pre-trained PointNet++ as the backbone. Table 6.8 shows that pre-training on RM-Harmonics significantly improves the detection results on both datasets. Also, the improvement is larger on the ScanNet (fine-tuning) dataset, which is in line with the observation in Section 6.3.2.2.

Moreover, a 2.5D-VoteNet (Chapter 3) is pre-trained using DPCo and RM-Harmonics. The model is evaluated on the SUN RGB-D dataset, following the setup in Section 4.3.4.1. As shown in Table 6.9, the generated synthetic data improve AP25 from 60.8 % to 61.3 % and AP50 from 36.9 % to 37.4 %.

Tables 6.8 and 6.9 show that randomized 3D scene generation can be applied to other proposed methods in this thesis, *e.g.*, DPCo and 2.5D-VoteNet.

Table 6.9 Fine-tuning results of 2.5D-VoteNet on the SUN RGB-D dataset with different pre-training data. Models are pre-trained using DPCo introduced in Chapter 4.

Pre-Training Data	**AP25** (%)	**AP50** (%)
N/A	60.8	36.9
ScanNet	61.4	38.8
RM-Harmonics	61.3	37.4

6.4 Additional Details and Visualization

This section presents additional implementation details and qualitative results.

6.4.1 Generating Spherical Harmonics

To generate sufficient objects for creating randomized scenes, this chapter randomly sets the coefficients in spherical harmonics described in Equation 6.1.

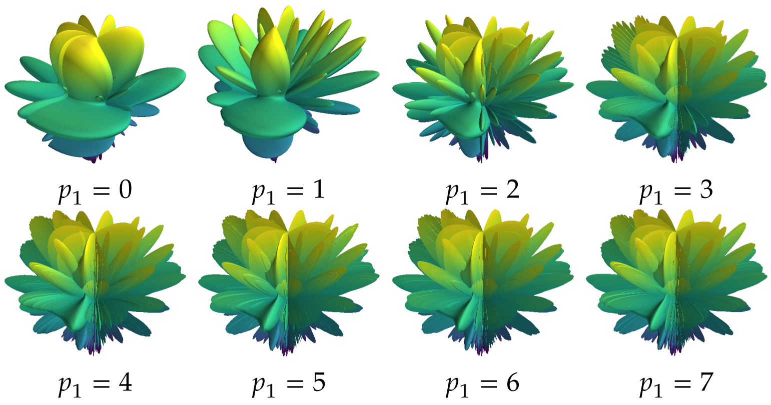


Figure 6.6 Impact of coefficient p_1 in spherical harmonics. For all objects: $[m_1, m_2, m_3, m_4] = [2, 1, 2, 2]$ and $[p_2, p_3, p_4] = [2, 1, 2]$.

The coefficients m_i are constrained in the range $[-5, 5]$ and p_i in $[0, 4]$, because too large coefficients lead to high-frequency structures. Due to the sampling theorem, these details are lost when points are sampled from spherical harmonics (represented as meshes) and are thus invisible in the pre-training.

The impact of the coefficients is illustrated in Figure 6.6, where all coefficients are fixed except p_1. With increasing p_1, the “fins” of the corresponding spherical harmonics become finer. A similar effect can be seen in Figure 6.7, where the influence of m_1 is visualized. With large coefficients, the overall appearance of spherical harmonics is similar. Thus,

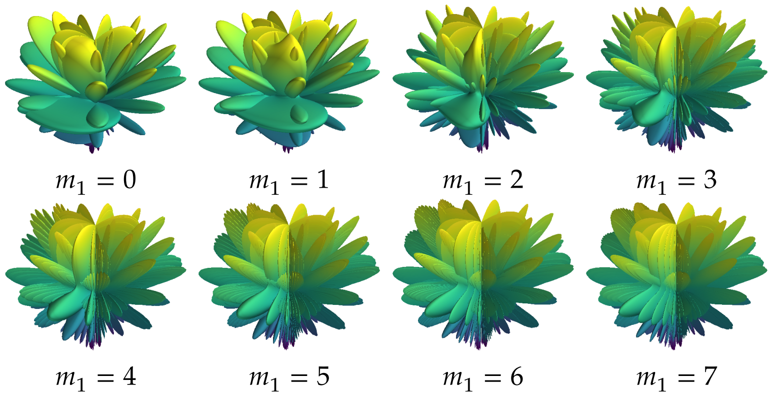


Figure 6.7 Impact of coefficient m_1 in spherical harmonics. For all objects: $[m_2, m_3, m_4] = [1, 2, 2]$ and $[p_1, p_2, p_3, p_4] = [2, 2, 1, 2]$.

this chapter constraints the spherical harmonics in the low-frequency range to generate diverse scenes.

6.4.2 Scene Generation Rules

This chapter generates scenes using the rules of Rao *et al.* [179]. Some modifications are made to the original configurations. The resulting procedure can be summarized as follows:

1. Randomly pick 12 to 16 objects from the object set.
2. Independently apply data augmentation to each object (more details in Section 6.4.3).
3. Sort the objects in descending order based on their projection areas on the horizontal plane (X-Y plane).
4. Randomly set the area (X-Y plane) of a rectangular room depending on the area sum of objects so that the objects can fit into the generated scene.
5. Randomly generate walls and floors according to the room area.

6. Place each object by randomly choosing a position on the X-Y plane. Objects can be stacked on previous ones as long as the current height of the position is below 2 m. Otherwise, choose a new position. Objects cannot overlap.

7. For multi-view point clouds, voxelize the entire scene with voxel size 0.04 m. Then, randomly sample 40000 points and save them for pre-training.

8. For single-view point clouds, randomly place a virtual camera in the scene and capture a depth map via ray-casting. Check the result so that each depth map contains at least 7 objects. All depth maps have the resolution 640×480, following the real-world dataset ScanNet [43]. The camera intrinsics and camera pose are also saved.

6.4.3 Pre-Processing and Data Augmentation

The pre-processing and data augmentation in this chapter are explained in the following.

6.4.3.1 Data Generation

Data augmentation is applied to each object during scene generation. Specifically, each object is first normalized into a unit sphere and is randomly scaled with a factor uniformly distributed in $[0.7, 1.5]$. Then the object is left-right flipped with a probability of 0.5 and is rotated around the vertical axis (Z-axis) with an arbitrary angle.

For spherical harmonics, the Z-axis and Y-axis are swapped with the probability 0.5 since spherical harmonics show approximate rotational symmetry around Z-axis (see Figures 6.6 and 6.7). The motivation is to prevent models from learning the bias that all objects are approximately symmetric around the vertical axis. At last, 3000 points are randomly sampled from each object for generating multi-view point clouds. For generating single-view data via ray-casting, the mesh representation is directly used without sampling.

6.4.3.2 Pre-Training

The primary pre-processing involved in the pre-training pipeline is cropping regions from an entire scene. However, the generated data must be differently pre-processed depending on pre-training methods and inputs.

1. For MAE with multi-view point clouds: A random point is picked from a complete point cloud, and the closest 20000 points (*i.e.*, 50 %) around it are cropped.

2. For MAE with single-view point clouds: A rectangular region with a random ratio in $[0.6, 0.8]$ is cropped from a depth map and transformed into a point cloud.

3. For contrastive learning: Similar approaches are applied as for MAE. This chapter crops spherical regions in multi-view point clouds and crops depth maps for single-view point clouds. However, two overlapping crops are generated in each scene instead of one to create a positive pair for contrastive learning.

Subsequently, standard data augmentation is applied to each crop, including random translation, rotation around the Z-axis, scaling, point jitter, and left-right flipping.

6.4.4 Generated Scenes

More generated scenes are visualized in Figure 6.8.

6.5 Conclusion

This chapter combines self-supervised pre-training with randomized 3D scene generation. Compared to previous supervised and self-supervised approaches, the introduced pipeline is especially efficient since it requires neither annotation nor real-world data. Experimental results show that the generated data perform well with different pre-training methods, *e.g.*, contrastive learning and mask autoencoder, and the pre-trained models achieve impressive results in multiple downstream tasks, *e.g.*, object detection and semantic segmentation.

Furthermore, this chapter takes a closer look at the impact of object sets and proposes replacing hand-crafted CAD models with formula-driven spherical harmonics. The method further reduces pre-training costs without harming performance.

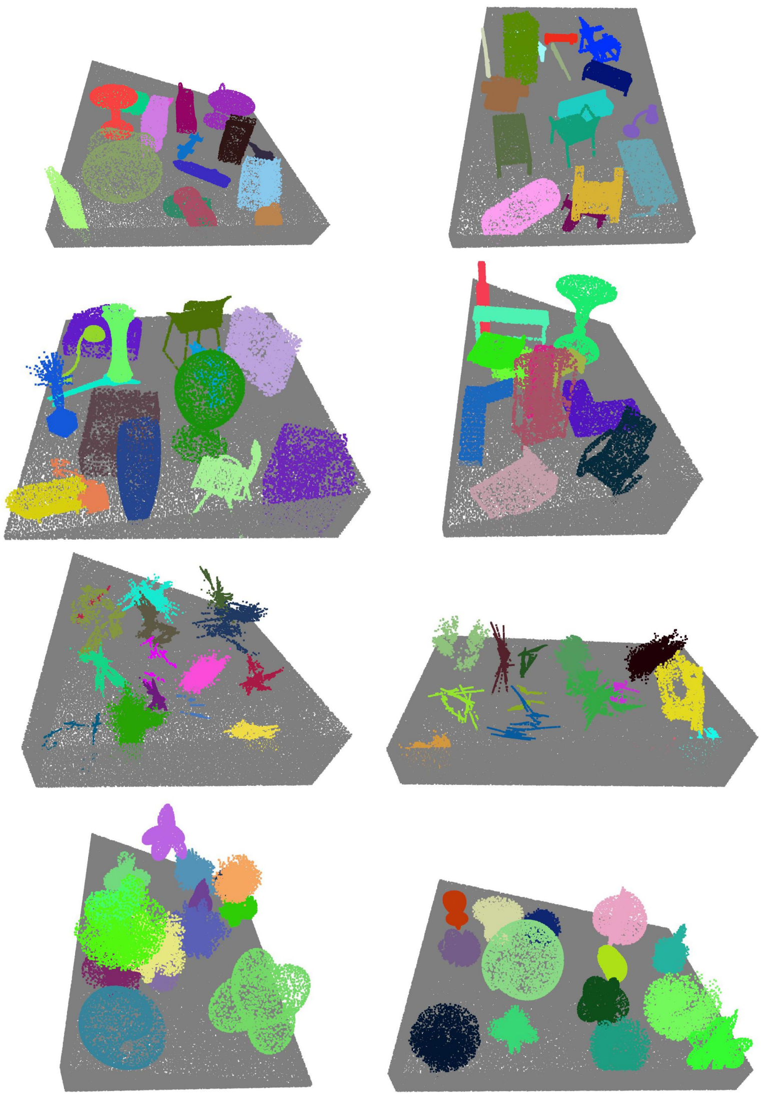


Figure 6.8 More examples of generated scenes. From the first to the fourth row: RM-ShapeNet, RM-ModelNet, RM-Fractal, and RM-Harmonics, respectively.

7 Summary and Future Works

7.1 Summary

This thesis addresses the high costs of applying neural networks to 3D computer vision tasks. Three aspects of efficiency in deep learning for sparse 3D data are discussed in depth, *i.e.*, computational, label, and data efficiency.

Chapter 3 represents a computationally efficient depth map-based object detector, which extracts features directly on depth maps instead of from point clouds. Thanks to the regular grid structure of depth maps, the detector uses a 2D CNN as its backbone instead of complicated and costly point cloud- or voxel-based models. Furthermore, Chapter 3 introduces a novel relative depth convolution layer to capture more informative features from depth maps. The proposed 3D detector achieves state-of-the-art performance and is significantly faster than previous methods. Besides the depth map-based model, Chapter 5 revisits plain transformers for point cloud understanding. By systematically evaluating and optimizing the patchifiers and position embedding, Chapter 5 introduces a transformer-based model with a simple structure and low computational cost.

To reduce the dependence on labeled data and improve the label efficiency of the training process, this thesis investigates self-supervised pre-training using unlabeled data. Chapter 4 revisits previous invariance-based contrastive learning methods and compares them under a unified framework. Moreover, it proposes a novel approach to contrasting features from depth maps and a 3D format (*e.g.*, voxels and point clouds). Extensive experiments demonstrate that the proposed strategy achieves superior results while having fewer requirements on the pre-training data. Additionally, Chapter 5 explores the masked autoencoder (MAE) to pre-train the proposed plain transformers. It discusses two information leakage problems in applying standard MAE to point clouds. A non-overlapping patchifier is introduced to fix the information leakage caused by overlapping patches. Furthermore, a novel method drop patch is proposed to prevent the position embedding from leaking positional information during the pre-training. The proposed models achieve state-of-the-art results in 3D object detection and semantic segmentation tasks thanks to the optimized structure and pre-training strategy.

For better data efficiency, Chapter 6 explores pre-training neural networks without real-world data. Instead of using the simulation technology, Chapter 6 uses randomized 3D scene generation to create synthetic data. It demonstrates that despite the poor fidelity of the generated data, this approach cooperates well with different self-supervised pre-training methods and model architectures. The pre-trained models generalize seamlessly in multiple downstream tasks. Furthermore, Chapter 6 proposes using formula-driven spherical harmonics as objects for data generation. The generated data show on-par performance as CAD models and real-world data. Furthermore, the proposed data-generating method is fully formula-driven, rule-based, and requires no additional data.

The methods proposed in this thesis have many application prospects. For instance, the neural networks introduced in Chapters 3 and 5 can be deployed for real-time applications, *e.g.*, 3D object detection on robots, thanks to their efficiency and simplicity. Also, the self-supervised methods proposed in Chapters 4 and 5 help train strong models without labeling large-scale datasets. Furthermore, the data generation technology presented in Chapter 6 allows pre-training neural networks without capturing real-world data.

Meanwhile, this thesis provides references for future research and potentially has a broader impact. For instance, the unified framework in Chapter 4 can be regarded as a tool for analyzing contrastive learning methods. Chapter 5 demonstrates the power of plain transformers. It may encourage the community to rethink the design of transformer-based models for point cloud data. Moreover, Chapter 6 addresses the generalization issues of models pre-trained on randomly generated data, which is overlooked in previous works.

The author hopes this thesis can inspire more research interest in the efficiency of deep learning for sparse 3D data.

7.2 Future Works

The efficiency of deep learning for sparse 3D data is still a new research area with many open questions.

This thesis applies 2D CNNs (Chapter 3) and transformers (Chapter 5) to 3D data for computational efficiency. Recent research also shows the potential of MLP-Mixer [230], a simple and efficient architecture, in computer vision tasks. Extending MLP-Mixer into the 3D domain can be an interesting topic for future works. On the other hand, this thesis reduces the computational costs of neural networks by analyzing their architecture and manually tuning the hyperparameters. Alternatively, the goal can be achieved via automatic optimization, *e.g.*, using evolutionary algorithms [47, 153] and neural architecture search [53, 293]. Although these methods have achieved promising results in image processing [95, 225], their application to 3D neural networks is still under-explored. Another possible way to improve the computational efficiency of deep learning models is using low-precision arithmetic (*e.g.*, 8-bit integer instead of 32-bit floating point number). Training and inference using low-precision arithmetic have drawn attention in image [78, 154] and neural language processing [109, 263, 271]. However, this technology is not well-developed for 3D computer vision tasks.

Chapter 4 applies a heterogeneous structure by contrasting features from two different neural networks. Jointly optimizing multiple networks is challenging since balancing their learning rates is non-trivial. This problem is researched in many contexts, *e.g.*, multi-task learning [37, 108, 199], generative adversarial networks (GANs) [69, 74], and multimodal learning [242]. However, it is under-explored in self-supervised learning with 3D data. A deeper investigation into the learning rate and convergence of different networks during pre-training might further improve the method proposed in Chapter 4.

Chapter 5 modifies the standard MAE for point cloud data. As explained in Section 2.4.4, the idea of MAE is inspired by masked language modeling (MLM) [46], where a language model is trained to recover some randomly masked words based on words around them. A competing approach with MLM is autoregressive pre-training [177], where a language model has to predict the next word in a text based on previous predictions. The recent success of ChatGPT and GPT4 [161] demonstrates the promising capability of autoregressive pre-training. However, extending this method to the computer vision domain is still an open problem.

Chapter 6 pre-trains neural networks using synthetic data generated using pre-defined formulas and rules. Meanwhile, generative models, *e.g.*, GANs [69], autoregressive models [178], and diffusion models [185] are widely applied for data generation. They learn a distribution from a dataset and create novel samples following the learned distribution. Previous works show promising results of generating *e.g.*, texts [57], hyperspectral signals [5, 6], and color images [69, 74, 105, 178, 185]. Also, recent research explores 3D shape generation using neural networks [104, 156]. These results allow training (or pre-training) strong networks using generative models as data generators and are worth more research interest in future works.

Furthermore, this thesis improves the label efficiency of deep learning methods via self-supervised pre-training. However, labels are still necessary for fine-tuning the pre-trained models. In practice, the fine-tuning data might be ambiguously or erroneously labeled. Such uncertainties lead to performance degradation in real-world tasks since neural networks are trained to fit the labels. Moreover, the automatically generated labels in the self-supervised pre-training can also contain uncertainties. For instance, the local correspondence in Chapter 4 can be ambiguous due to the limited resolution of feature maps or feature points. Also, the MAE in Chapter 5 might generate a reconstruction which is plausible but distinct from the ground truth. Handling the uncertainties of labels has drawn much research attentions [20, 103, 112, 158, 200, 269]. On the other hand, Hendrycks *et al*. [84] demonstrate that supervised pre-training can improve the robustness of models against label uncertainties. However, the impact of label uncertainties in the context of self-supervised pre-training, especially with sparse 3D data, is still under-explored and can be another interesting topic for feature works.

Appendix

A Sensors

This chapter introduces two types of widely applied sensors in 3D computer vision.

A.1 Depth Camera

A depth camera can be modeled as a standard pinhole camera. However, instead of capturing the color values of each pixel, a depth camera measures the distance from objects to the image plane, which results in a depth map. A depth map can be transformed into a single-view point cloud using the camera calibration [224]. A representative scene captured by a depth camera is shown in Figure A.1.

Different methods are applied for the distance measurement:

1. Structured light: The camera projects a known pattern onto a scene. The depth can be calculated by analyzing the deformation of the pattern.

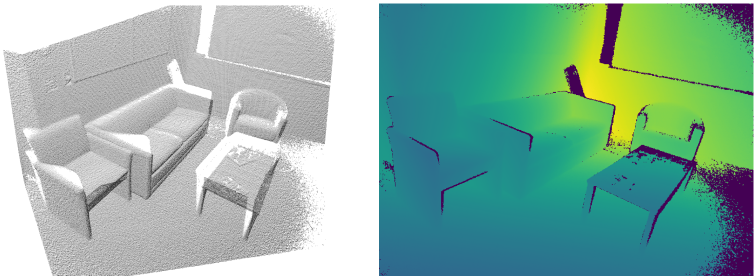


Figure A.1 Visualization of a point cloud and a depth map from a depth camera. Data source: the SUN RGB-D dataset [212].

2. Stereo photography: The camera has two or more lenses and captures images from multiple slightly shifted view points. The depth is obtained by comparing the differences in the images.

3. Time-of-Flight (ToF): The camera emits light onto a scene and measures the time of reflected light to return to the sensor.

In a depth map, the depth at some pixels might be not measurable, *e.g.*, the dark pixels on the depth map in Figure A.1. These pixels are often called *bad pixels* or *dead pixels*. For instance, if a surface absorbs the light emitted by a ToF camera, the depth values on the surface cannot be determined. Also, if a stereo camera captures a scene lacking texture, the textureless regions contain many bad pixels because texture is necessary for comparing images from different view points.

To provide an overview of depth cameras' properties, Table A.1 shows the technical specifications of an Intel RealSense L515 depth camera[1] as a representative.

Table A.1 Technical specifications of an Intel RealSense L515 depth camera.

Environment	Indoor
Range	0.25 m to 9 m
Horizontal field of view (HFOV)	70°
Vertical field of view (VFOV)	55°
Resolution	1024×768
Frame rate	30 FPS
Depth accuracy	14 mm

A.2 Rotational LiDAR Sensor

LiDAR (light detection and ranging) is a method for measuring ranges by casting laser beams at objects and determining the time for the reflected light to return to the receiver. The rotational LiDAR sensor is widely applied in 3D computer vision for autonomous driving [65, 219]. The

[1] Source: https://www.intelrealsense.com/lidar-camera-l515/. Last accessed: 2023.06.29.

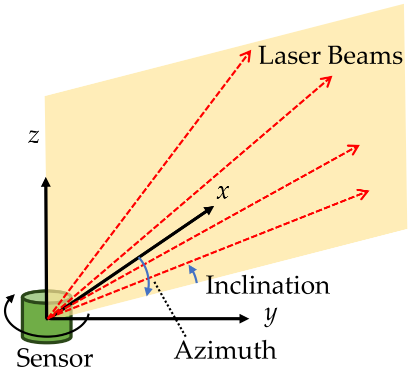


Figure A.2 Measurement concept of a rotational LiDAR sensor.

principle of the rotational LiDAR sensor is visualized in Figure A.2. As the name suggests, the sensor rotates around a vertical axis. By each measurement, it emits multiple laser beams at the same time. Assuming that the sensor is located at the origin of a Cartesian coordinate system, all laser beams and the vertical axis (Z-axis) are in the same plane. A range value can be measured with each beam. The sensor also determines the magnitude of each reflected laser beam (*i.e.*, intensity), which is often used as an additional feature. Then, the sensor rotates by a fixed angle $\Delta\theta$ and performs a new measurement. A spherical coordinate system (see Figure A.2) is often used for easier data processing.

Note that all beams at each time share an azimuth θ. Also, one beam at all times in a scan cycle (*i.e.*, the rotation of 360°) shares an inclination ϕ. Therefore, the measurement results in each scan cycle can be represented as a mapping from angles (θ, ϕ) to range r. The mapping can also be described using a $W \times H$ matrix, where $W = 2\pi/\Delta\theta$ and H equals the number of laser beams. The matrix is referred to as a range image. Due to the rotational movement of the sensor, a range image always has a horizontal field of view (HFOV) of 360°.

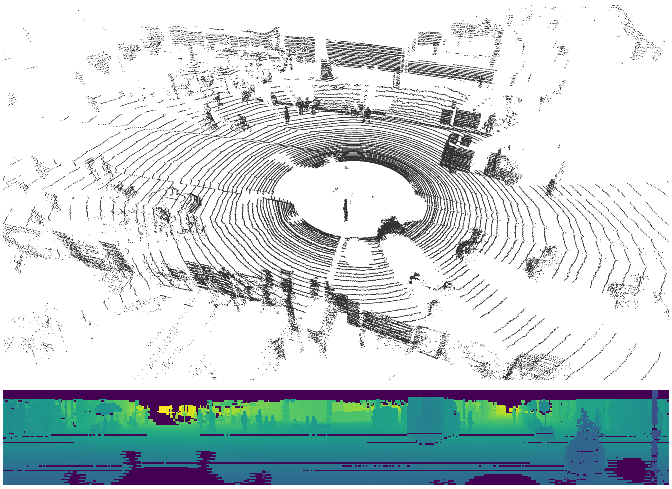


Figure A.3 Visualization of a LiDAR point cloud and its corresponding range image. Data source: the KITTI dataset [65].

A transformation from the spherical to the Cartesian coordinate system is given by:

$$x = r\cos(\phi)\cos(\theta)\,, \tag{A.1}$$
$$y = r\cos(\phi)\sin(\theta)\,, \tag{A.2}$$
$$z = r\sin(\phi)\,, \tag{A.3}$$

where (x, y, z) denotes the Cartesian coordinates of points. Note that only a single-view point cloud can be transformed into a range image without information loss. Otherwise, multiple points may overlap in one pixel in the range image. The point cloud and the range image representations of 3D data are illustrated in Figure A.3.

To provide an overview of rotational LiDAR sensors' properties, Table A.2 shows the technical specifications of a Velodyne HDL-64E LiDAR sensor[2], which is used by the well-known KITTI dataset [65]. One can see

[2] Source: https://hypertech.co.il/wp-content/uploads/2015/12/HDL-64E-Data-Sheet.pdf. Last accessed: 2023.06.28.

that rotational LiDAR sensors and depth cameras have distinct ranges, fields of view, resolutions, and refresh rates.

Table A.2 Technical specifications of a Velodyne HDL-64E LiDAR sensor.

Environment	Outdoor
Range	up to 120 m
Horizontal field of view (HFOV)	360°
Vertical field of view (VFOV)	26.6°
Beams (vertical resolution)	64
Horizontal resolution	450
Rotation speed	up to 15 Hz
Depth accuracy	2 cm

B Datasets

This chapter briefly introduces the openly accessible datasets used in this thesis.

B.1 Real-World Datasets

Three real-world datasets are applied in this thesis, *i.e.*, the S3DIS, the SUN RGB-D, and the ScanNet dataset.

B.1.1 S3DIS

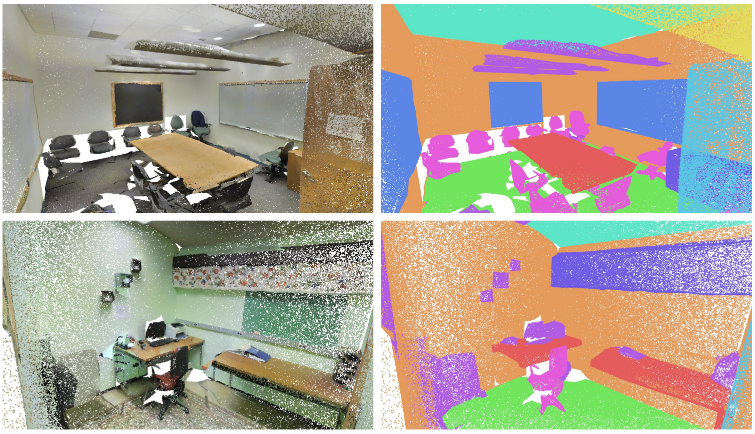


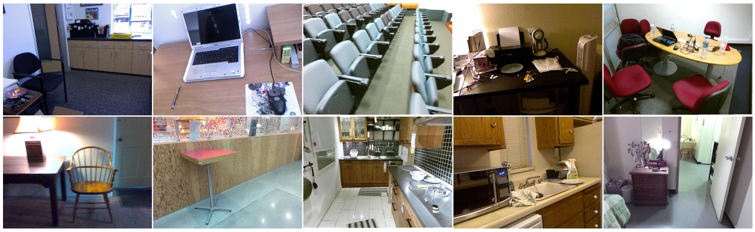
Figure B.1 Visualization of point clouds in the S3DIS dataset. Left: with colors. Right: with semantic labels.

The Stanford 3D Indoor Scene (S3DIS) dataset [10] contains point clouds captured in 6 large-scale indoor areas with 271 rooms. According to the convention of previous works, point clouds from Area 1, 2, 3, 4, and 6 are used as the training set, while Area 5 is reserved for validation. Each point is labeled with one of the 13 semantic categories: ceiling, floor, wall, beam, column, window, door, chair, table, bookcase, sofa, board, and clutter. Some representative scenes in the dataset are shown in Figure B.1.

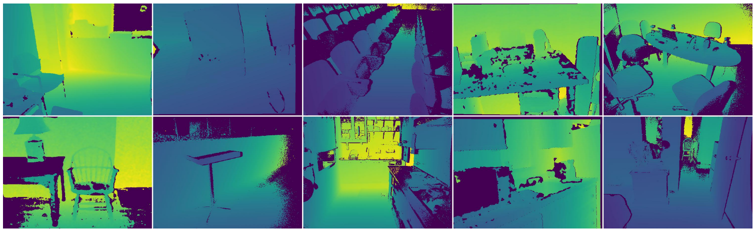
In this thesis, the S3DIS dataset is used in Chapters 4, 5, and 6 to evaluate neural networks for 3D semantic segmentation tasks.

B.1.2 SUN RGB-D

The SUN RGB-D dataset [212] consists of 10335 RGB-D images of indoor scenes. 5285 images are used for training and 5050 for validation. Each RGB-D image comprises a depth map and an aligned color image. Some depth maps and images are visualized in Figure B.2.



(a) RGB images

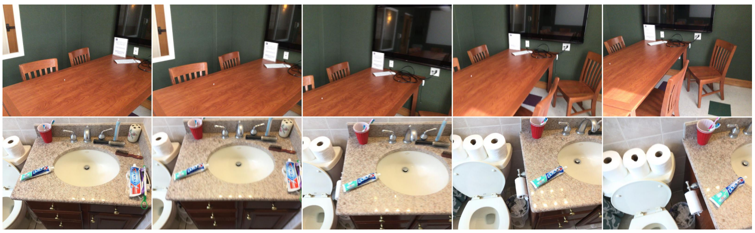


(b) Depth maps (normalized for visualization)

Figure B.2 Data samples from the SUN RGB-D dataset.

Also, the SUN RGB-D dataset contains rich labels, *e.g.*, semantic maps, 3D bounding boxes, and scene categories. This thesis only uses the bounding box labels to train 3D object detectors (see Chapters 3, 4, and 6). Each bounding box in SUN RGB-D has 7 degrees of freedom, *i.e.*, the box center, the box size, and the rotation angle around the gravity direction. Also, 10 representative categories of objects are considered in the object detection task, *i.e.*, bathtub, bed, bookshelf, chair, desk, dresser, nightstand, sofa, table, and toilet. The annotations of the dataset have two versions, where the V2 annotation has a better quality than V1. However, due to historical reasons, most previous works use the V1 annotation. This thesis follows this convention for better comparability with these works.

B.1.3 ScanNet


(a) RGB images

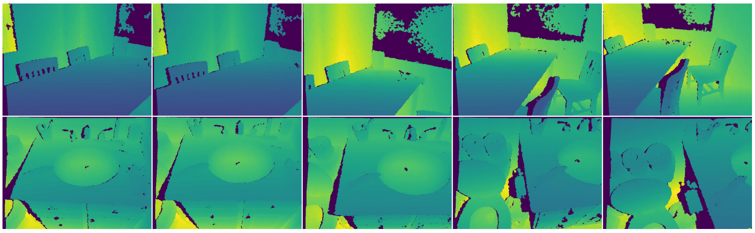

(b) Depth maps (normalized for visualization)

Figure B.3 Raw frames from the ScanNet dataset. Each row is sampled from the same video with the interval of approximately one second.

The ScanNet dataset [43] contains 2.5 million RGB-D images collected in 1513 indoor scenes. Also, it provides reconstructed scans for each scene. Data from 1201 scenes are used for training, and the other 312 belong to

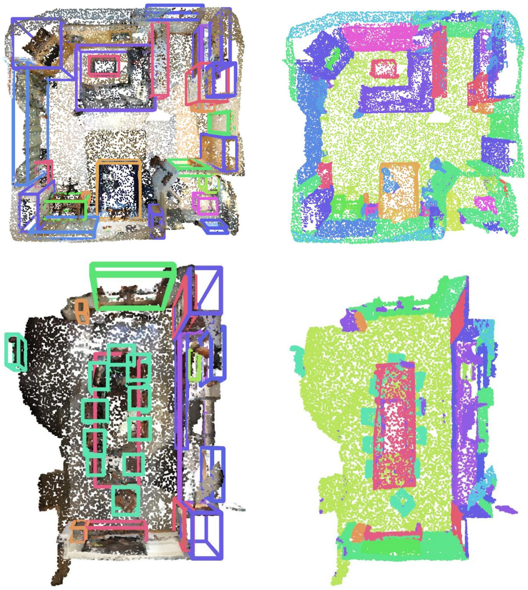


Figure B.4 Visualization of data and labels in the ScanNet dataset. Left: point clouds with bounding boxes. Right: point-wise semantic labels.

the validation set. ScanNet provides semantic masks for each object in reconstructed scans (*i.e.*, for the instance segmentation). Additional labels can be generated from the instance-level masks, *e.g.*, bounding boxes for object detection and semantic maps for semantic segmentation. The raw frames from ScanNet are visualized in Figure B.3. Furthermore, Scene-level bounding boxes and semantic labels are illustrated in Figure B.4.

The ScanNet dataset is widely used in this thesis:

1. In Chapter 3, the scene-level bounding boxes are transformed to each camera coordinate system to obtain frame-level labels and train depth map-based detectors.

2. Chapters 4, 5, and 6 use reconstructed scans and scene-level bounding box labels to train object detectors.

3. Chapter 4 applies this dataset to evaluate a pre-trained voxel-based network for semantic segmentation.

4. Raw RGB-D frames from the ScanNet dataset are used in Chapters 4, 5, and 6 for self-supervised pre-training.

In the semantic segmentation task, 20 semantic classes are considered, *i.e.*, wall, floor, cabinet, bed, chair, sofa, table, door, window, bookshelf, picture, counter, desk, curtain, refrigerator, shower curtain, toilet, sink, bathtub, and a category for all other furniture. However, 18 classes are used for 3D object detection since class *wall* and *floor* are excluded. Also, only *garbage bin* in the *other furniture* class is used. Moreover, bounding box labels in ScanNet are axis-aligned without rotation.

B.2 Synthetic Datasets

Two synthetic datasets, *i.e.*, ModelNet40 and ShapeNet, are used in this thesis.

B.2.1 ModelNet40

The ModelNet40 dataset [255] consists of 12311 synthetic CAD models (*i.e.*, 3D meshes), 9843 are for training and 2468 for testing. Some representative meshes in the dataset are visualized in Figure B.5.

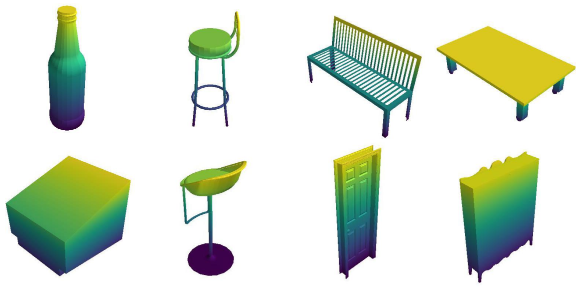


Figure B.5 Data samples from ModelNet40 training set. Object sizes are normalized.

Table B.1 Classes and object numbers in ModleNet40 training set.

chair	889	sofa	680	airplane	626	bookshelf	572
bed	515	vase	475	monitor	465	table	392
toilet	344	bottle	335	mantel	284	tv stand	267
plant	240	piano	231	night stand	200	dresser	200
desk	200	car	197	bench	173	glass box	171
cone	167	tent	163	guitar	155	flower pot	149
laptop	149	keyboard	145	curtain	138	sink	128
lamp	124	stairs	124	range hood	115	door	109
bathtub	106	radio	104	xbox	103	stool	90
person	88	wardrobe	87	cup	79	bowl	64

Each mesh in ModelNet40 is labeled with one of 40 semantic classes. The 40 classes are listed in Table B.1. One can see that the classes correspond to ordinary objects in daily life. The table also demonstrates the object numbers of each class in the training set (gathering statistics in test sets is not permitted in practice). The numbers show that the class distribution in this dataset is imbalanced (*i.e.*, long-tailed).

In this thesis, ModelNet40 is exploited for two purposes:

1. Chapters 4 and 5 employ this dataset to fine-tune pre-trained neural networks in object classification tasks, where the classification accuracy is regarded as a measure of the pre-training effect.
2. Chapter 6 creates randomized 3D scenes using ModelNet40 as an object set. In this case, the class labels are not used.

B.2.2 ShapeNet

ShapeNet [26] is a large-scale CAD model dataset containing multiple subsets. Also, the ShapeNet dataset was once revised and has two versions. In this thesis, only the V1 version of the ShapeNetCore subset is used. For simplicity, ShapeNet in this thesis refers to the V1 version of the ShapeNetCore subset.

ShapeNetCore V1 consists of approximately 51300 objects categorized into 55 classes. The class names and corresponding object numbers are listed in Table B.2. Similar to ModelNet40 (see Table B.1), the ShapeNetCore V1 dataset is also long-tailed. However, ShapeNet contains more

Table B.2 Classes and object numbers in ShapeNetCore V1 training set.

table	8443	car	7497	chair	6778	airplane	4045
sofa	3173	rifle	2373	lamp	2318	watercraft	1939
bench	1816	loudspeaker	1618	cabinet	1572	display	1095
telephone	1052	bus	939	bathtub	857	guitar	797
faucet	744	clock	655	flowerpot	602	jar	597
bottle	498	bookshelf	466	laptop	460	knife	424
train	389	trash bin	343	motorbike	337	pistol	307
file cabinet	298	bed	254	piano	239	stove	218
mug	214	bowl	186	washer	169	printer	166
helmet	162	microwaves	152	skateboard	152	tower	133
camera	113	basket	113	can	108	pillow	96
mailbox	94	dishwasher	93	rocket	85	bag	83
birdhouse	73	earphone	73	microphone	67	remote	67
keyboard	65	bicycle	59	cap	56		

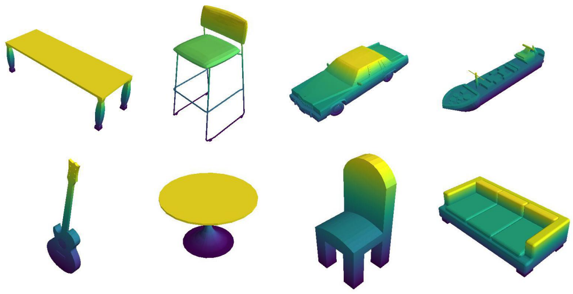


Figure B.6 Data samples from ShapeNetCore V1 training set. Object sizes are normalized.

data samples and categories. Some representative meshes from ShapeNet-Core V1 are visualized in Figure B.6.

In this thesis, ShapeNetCore V1 has two applications:

1. In Chapter 5, it is used as a pre-training dataset to evaluate multiple self-supervised methods on synthetic data.
2. Chapter 6 employs it as an object set to generate synthetic scenes.

No labels are used in both cases.

C Evaluation Metrics and Protocol

This section introduces the evaluation metrics and protocol used in this thesis.

C.1 Metrics for 3D Semantic Segmentation

As explained in Section 2.3.2, 3D semantic segmentation is a dense classification task. For simplicity, this section assumes that the neural network uses point clouds as input. The generalization to other 3D data, *e.g.*, voxels and depth maps, is straightforward. Usually, a dataset consists of many point clouds, each containing many points. However, the actual number of point clouds (or scenes) is not important for calculating the metrics of semantic segmentation. Therefore, this section describes a dataset as a set of (independent) points for simplicity.

Given a dataset with totally N points $\left\{\mathbf{x}^i\right\}_{i=1}^N$, a neural network makes N point-wise predictions $\left\{\hat{k}^i\right\}_{i=1}^N$, where $\hat{k}^i \in \{1, 2, \dots, N_{\text{cls}}\}$ is the predicted category of the i-th point and N_{cls} indicates the number of categories. Similarly, the ground truth can be described as $\left\{k^i\right\}_{i=1}^N$, where $k^i \in \{1, 2, \dots, N_{\text{cls}}\}$ indicates the true class of point $\mathbf{x}^i$.

In this thesis, the mean accuracy (mAcc) is used as an evaluation metric for semantic segmentation. To calculate the mAcc, the semantic segmentation task is viewed as a binary classification problem for each class $j \in \{1, 2, \dots, N_{\text{cls}}\}$. The accuracy for class j is given by:

$$\text{Acc}_j = \frac{\sum_{i=1}^N f_{\text{int}}\left(\hat{k}^i, k^i, j\right)}{\sum_{i=1}^N I(k^i, j)}, \tag{C.1}$$

where $f_{\text{int}}(\dots)$ is an auxiliary function which can be used to count the number of points where the prediction is correct (*i.e.*, intersection of the labels and predictions):

$$f_{\text{int}}\left(\hat{k}^i, k^i, j\right) = \begin{cases} 1 \ , \ \hat{k}^i = j \text{ and } k^i = j \\ 0 \ , \ \text{otherwise} \end{cases} . \tag{C.2}$$

The function $I(\dots)$ equals one if two arguments have identical values. Otherwise, it is equal to zero. The denominator means the number of points whose label (*i.e.*, ground truth) is equal to the class index j, while the numerator gives the number of points where the neural network correctly predicts the category j. Then, the mean accuracy over N_{cls} classes is given by

$$\text{mAcc} = \frac{1}{N_{\text{cls}}} \sum_{j=1}^{N_{\text{cls}}} \text{Acc}_j \, . \tag{C.3}$$

Obviously, $\text{mAcc} \in [0, 1]$ always applies. Therefore, mAcc is often used in percentage. The higher the value, the better the prediction quality.

Besides mAcc, the mean intersection over union (mIoU) is also widely applied to evaluate the quality of semantic segmentation results. For each class j, the intersection over union IoU_j is given by:

$$\text{IoU}_j = \frac{\sum_{i=1}^{N} f_{\text{int}}\left(\hat{k}^i, k^i, j\right)}{\sum_{i=1}^{N} f_{\text{uni}}\left(\hat{k}^i, k^i, j\right)} , \tag{C.4}$$

where the function f_{uni} is defined as

$$f_{\text{uni}}\left(\hat{k}^i, k^i, j\right) = \begin{cases} 1 \ , \ \hat{k}^i = j \text{ or } k^i = j \\ 0 \ , \ \text{otherwise} \end{cases} . \tag{C.5}$$

Equations C.1 and C.4 have the same numerator. However, the denominator in Equation C.4 represents the total number of points where the prediction or the ground truth is equal to the class index j (*i.e.*, the union). The mIoU is then obtained by averaging IoU_j over N_{cls} classes:

$$\text{mIoU} = \frac{1}{N_{\text{cls}}} \sum_{j=1}^{N_{\text{cls}}} \text{IoU}_j \, . \tag{C.6}$$

Similar to mAcc, mIoU is also constrained in the range $[0, 1]$ and used in percentage in practice. A higher mIoU indicates a better segmentation result.

C.2 Metrics for 3D Object Detection

This section introduces the mean average precision (mAP) as an evaluation metric for 3D object detection. Without loss of generality, point clouds are assumed as inputs.

Given a dataset of N point clouds $\{\Omega_i\}_{i=1}^{N}$, a neural network predicts N_{pred} bounding boxes with $\left\{\left(\hat{\mathbf{y}}^m, \hat{k}^m, c^m, p^m\right)\right\}_{m=1}^{N_{\text{pred}}}$. The geometrical parameters, *e.g.*, the center coordinate and size, are given as a vector $\hat{\mathbf{y}}^m$. Since bounding boxes can be differently parameterized, *e.g.*, as rotated boxes or axis-aligned boxes (*cf.* Section 2.3.3), this section uses this abstract form for simplicity. The class prediction $\hat{k}^m \in \{1, \dots, N_{\text{cls}}\}$ indicates that the box is predicted to belong to the $\hat{k}^m$-th class. Also, the neural network predicts a confidence score $c^m \in [0, 1]$, which is an estimate of the probability that the predicted box matches a ground truth box. The sample index $p^m \in \{1, 2, \dots, N\}$ means the box is predicted using p^m-th point cloud in the dataset. Similarly, the labels are given by $\{(\mathbf{y}^m, k^m, p^m)\}_{m=1}^{N_{\text{label}}}$. Note that number of labels and predictions usually differ with $N_{\text{label}} \neq N_{\text{pred}}$.

To calculate the mAP, the average precision for each class must be determined. For class j, predictions with $\hat{k}^m = j$ and labels with $k^m = j$ are selected, resulting in two sub-sets $\Psi^j_{\text{pred}} = \left\{\left(\hat{\mathbf{y}}^{m^j}, c^{m^j}, p^{m^j}\right)\right\}_{m^j=1}^{N^j_{\text{pred}}}$ and $\Psi^j_{\text{label}} = \left\{\left(\mathbf{y}^{m^j}, p^{m^j}\right)\right\}_{m^j=1}^{N^j_{\text{label}}}$. The constants N^j_{pred} and N^j_{label} indicate the number of predictions and labels for class j, respectively. The class labels $k^m = j$ and predictions $\hat{k}^m$ are omitted here since they are all identical in these two sub-sets. Also, the selected predictions Ψ^j_{pred} are sorted by the confidence score in descending order.

A prediction $\hat{\mathbf{y}}^{m^j}$ in Ψ^j_{pred} is regarded as true if its 3D (*i.e.*, volumetric) intersection over union with a ground truth box corresponding to the same point cloud is higher than a threshold t_{IoU}. In practice, the value of

t_{IoU} is chosen empirically. The higher the value, the stricter the requirements for the detection quality. In this thesis, 25 % and 50 % are used as thresholds. Given two boxes $\mathbf{y}^1$ and $\mathbf{y}^2$, the 3D intersection over union (3D-IoU) is given by

$$f_{\mathrm{IoU}}\left(\mathbf{y}^1, \mathbf{y}^2\right) = \frac{V_{\mathrm{int}}\left(\mathbf{y}^1, \mathbf{y}^2\right)}{V_{\mathrm{uni}}\left(\mathbf{y}^1, \mathbf{y}^2\right)}, \tag{C.7}$$

where $V_{\mathrm{int}}(\dots)$ and $V_{\mathrm{uni}}(\dots)$ calculate the volume of the intersection and the union of two 3D boxes, respectively.

Based on the IoU, the average precision of predictions Ψ^j_{pred} can be calculated by evaluating the N^j_{pred} boxes step by step. For the n-th step with $n \in \{1, 2, \dots, N^j_{\mathrm{pred}}\}$, the number of correct predictions (*i.e.*, true positive) is given by

$$\mathrm{TP}_j(n) = \sum_{m^j=1}^{n} f_{\mathrm{match}}(\hat{\mathbf{y}}^{m^j}, \Psi^j_{\mathrm{label}}). \tag{C.8}$$

The function f_{match} is defined as

$$f_{\mathrm{match}}(\hat{\mathbf{y}}^{m^j}, \Psi^j_{\mathrm{label}}) = \begin{cases} 1, & f_{\mathrm{IoU}}(\hat{\mathbf{y}}^{m^j}, \mathbf{y}^{m_{\mathrm{match}}}) > t_{\mathrm{IoU}} \\ 0, & \text{otherwise} \end{cases}, \tag{C.9}$$

where the index m_{match} indicates the label in Ψ^j_{label} that has the largest IoU with the prediction $\hat{\mathbf{y}}^{m^j}$:

$$m_{\mathrm{match}} = \underset{l}{\mathrm{argmax}} \left\{ f_{\mathrm{IoU}}(\hat{\mathbf{y}}^{m^j}, \mathbf{y}^l) \right\}, \tag{C.10}$$
$$\text{subject to } p^{m^j} = p^l \text{ and } \mathbf{y}^l \in \Psi^j_{\mathrm{label}}.$$

The constraint $p^{m^j} = p^l$ is necessary since the prediction and label must correspond to one data sample. For each step n, the true positive $\mathrm{TP}_j(n)$ increases by one if the n-th prediction in Ψ^j_{pred} has a matched ground truth box with an IoU larger than the pre-defined threshold. The precision at the n-th step is defined as the ratio of correct predictions to all predictions.

$$\mathrm{PR}_j(n) = \frac{\mathrm{TP}_j(n)}{n}. \tag{C.11}$$

The recall of the predictions is defined as the the ratio of detected ground truth boxes to all ground truth boxes. The true positive for recall $\mathrm{TP}_j^-(n)$ is calculated using a similar approach to $\mathrm{TP}_j(n)$, *i.e.*, by counting ground truth boxes that have matched predictions with an IoU higher than the threshold t_{IoU}. The difference is how they handle redundant predictions. When multiple predictions match one ground truth box, they are counted independently by $\mathrm{TP}_j(n)$. On the contrary, they are counted only once by $\mathrm{TP}_j^-(n)$. The $-$ symbol in the notation indicates that $\mathrm{TP}_j^-(n) \leq \mathrm{TP}_j(n)$. The recall on the n-th step $\mathrm{RC}_j(n)$ is given by:

$$\mathrm{RC}^j(n) = \frac{\mathrm{TP}_j^-(n)}{N^j_{\mathrm{label}}} . \tag{C.12}$$

Note that the predictions in Ψ^j_{pred} are sorted by the confidence scores. Generally, the predictions on the first several steps are more accurate, *i.e.*, $\mathrm{PR}_j(n) \approx 1$. Also, since only a few bounding box predictions are available, the recall is low on early steps, *i.e.*, $\mathrm{RC}_j(n) \approx 0$. When n becomes larger, $\mathrm{PR}_j(n)$ decreases and $\mathrm{RC}_j(n)$ increases.

The precision and recall on each step can be plotted in a diagram, resulting in a precision-recall curve. The red curve in Figure C.1 corresponds to bounding box predictions with a bad quality, where the precision drops rapidly. Also, the curve ends with a recall lower than 100 %, which means some ground truth boxes do not have matched predictions (*i.e.*, they are not detected). The blue curve in Figure C.1 is an example of a good precision-recall curve, where the precision remains high as recall increases. The green curve can be obtained using the ground truth as the prediction, so the precision is always 100 %.

For each class j, the metric average precision AP_j is defined as the area under the precision-recall curve of this class. As illustrated in Figure C.1, the value of average precision is constrained in the range of $[0, 1]$. The higher the value, the better the quality of the predictions. The mean average precision (mAP) is obtained by averaging AP_j over all classes:

$$\mathrm{mAP} = \frac{1}{N_{\mathrm{cls}}} \sum_{j=1}^{N_{\mathrm{cls}}} \mathrm{AP}_j . \tag{C.13}$$

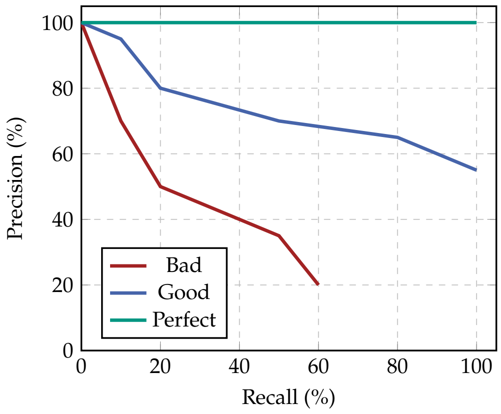


Figure C.1 Visualization of precision-recall curves.

As Equation C.9 shows, the choice of t_{IoU} has a large impact on mAP since it defines the threshold of a predicted bounding box to be true. Therefore, the metric mAP is only meaningful with an IoU threshold given. This thesis uses mAP with 25 % and 50 % 3D-IoU thresholds as metrics. For simplicity, they are referred to as AP25 and AP50, respectively. Note that some literature uses the notations AP_{25} and AP_{50} instead.

C.3 Evaluation Protocol

Due to randomness in the training process, *e.g.*, random initialization of weights, random sampling for mini-batches, and random data augmentation, the performance of a neural network fluctuates if it is initialized and trained multiple times. Depending on the computational costs, all experiments in this thesis are repeated three to five times. The *best result* instead of the averaged metric is reported, following the convention in previous works [152, 173, 179, 260, 268, 285]. This protocol is feasible because it is a common practice to train multiple neural networks with the same architecture and configuration and only to apply the one with the best performance.

D Deployment of Neural Networks

The design of neural networks has to consider the deployment on target platforms since the deployment is the last step of applying deep learning models. This chapter introduces some fundamental knowledge in deploying neural networks. Also, it explains the software and hardware constraints in this process and empirically evaluates the deployment-friendliness of various architectures in 3D deep learning.

D.1 Background

Modern neural networks rely on heterogeneous computing, *i.e.*, CPUs are responsible for the logical control and data preparation, while the computation within neural networks is performed by co-processors (also known as accelerators). However, training neural networks and deploying trained models on target platforms have distinct requirements.

During training, both the forward and the backward pass are necessary to compute the loss and gradients. Also, models are trained using a large batch size for more effective parallel computing. Therefore, neural networks are usually trained centralized on high-performance GPUs. Although they massively accelerate the training process, the GPUs have high manufacturing costs and energy consumption.

On the contrary, neural networks are often deployed on edge devices in a decentralized manner, *e.g.*, locally in smartphones or self-driving cars. It is especially the case for 3D deep learning since many applications, *e.g.*, robotics and autonomous driving, are sensitive to latency. Therefore, a centralized solution is unsuitable due to the high communication overhead. Because back-propagation is not performed during the inference and data are usually processed with a tiny batch size (*e.g.*, one), low-cost and energy-efficient processors are sufficient for edge devices, *e.g.*, edge GPU, FPGA, and ASIC.

Besides the hardware, the training and deployment of neural networks rely on different software infrastructures. Training frameworks, *e.g.*, TensorFlow [1], PyTorch [167], and JAX [21], must consider accelerating the back-propagation to reduce the overall time cost of training. However, it inevitably increases the computational cost of the forward pass. Moreover, training frameworks often trade off computational efficiency for better usability. For instance, most frameworks use Python instead of C++ as the programming language for their front-ends [1, 21, 167]. Also, dynamic computational graphs are often employed instead of static ones for easier troubleshooting in developing and training models [167]. On the contrary, frameworks for deployment, *e.g.*, TVM [30], ONNX, and NVIDIA TensorRT, are exclusively and aggressively optimized for faster inference. Also, the computation is often executed using low-precision arithmetic, *e.g.*, with 16-bit floating point numbers or 8-bit integers, instead of 32-bit floating point numbers, which are commonly used for training.

Therefore, deploying neural networks means migrating trained models to different software and hardware infrastructures, which involves much engineering efforts and technical constraints.

D.2 Constraints

The gap between training and deploying neural networks leads to some constraints.

Non-programmable co-processors, *e.g.*, ASICs, have a fixed list of supported operations. Unsupported operations cannot be accelerated and fall back to CPUs, which results in a significant latency increase due to the additional IO (input-output) overhead and CPUs' low performance. Since image and neural language processing using deep learning is developed and applied earlier than 3D deep learning [88, 114, 119, 170], most existing non-programmable co-processors are designed for 2D CNNs or RNNs [51, 205]. For instance, the supported operations of a Google Coral Edge TPU[1], an ASIC for edge neural computing, are given in Table D.1. Obviously, the processor primarily supports MLPs, RNNs, and

[1] Source: https://coral.ai/docs/edgetpu/models-intro. Last accessed on 2023.05.31.

Table D.1 All operations supported by a Google Coral Edge TPU. Original names in the official documentation are used.

Operation Names		
Add	AveragePool2d	Concatenation
Conv2d	DepthwiseConv2d	ExpandDims
FullyConnected	L2Normalization	Logistic
LSTM	Maximum	MaxPool2d
Mean	Minimum	Mul
Pack	Pad	PReLU
Quantize	ReduceMax	ReduceMin
ReLU	ReLU6	ReLUN1To1
Reshape	ResizeBilinear	ResizeNearestNeighbor
Rsqrt	Slice	Softmax
SpaceToDepth	Split	Squeeze
StridedSlice	Sub	Sum
Squared-difference	Tanh	Transpose

2D CNNs. Deploying other architectures on such hardware is challenging or impossible.

Programmable co-processors, *e.g.*, GPUs and FPGAs, can theoretically execute arbitrary operations. However, their corresponding deployment frameworks often have software constraints, *i.e.*, the supported layers and functions are still limited. For instance, the officially supported layers of TensorRT[2], a framework for accelerating inference on NVIDIA GPUs, are presented in Table D.2. Note that the terms *layer* and *operation* are often interchangeably used in the context of deployment. This section follows the convention of each framework, *i.e.*, using *layer* for NVIDIA TensorRT and *operation* for Google Coral. Table D.2 implies that TensorRT can primarily accelerate MLPs, 2D CNNs (dense), 3D CNNs (dense), and transformers. Such software constraints can be overcome by adding custom operations since the processors are programmable. For instance, TensorRT allows users to define new layers as plugins (*i.e.*, PluginV2Layer

2 Version 8.6. Source: https://docs.nvidia.com/deeplearning/tensorrt/operators/index.html. Last accessed on 2023.05.31.

Table D.2 All layers supported by NVIDIA TensorRT. Original names in the official documentation are used.

Layer Names	
ActivationLayer	AssertionLayer
CastLayer	ConcatenationLayer
ConstantLayer	ConvolutionLayer (2D)
ConvolutionLayer (3D)	DeconvolutionLayer (2D)
DeconvolutionLayer (3D)	DequantizeLayer
EinsumLayer	ElementWiseLayer
FillLayer	FullyConnectedLayer
GatherLayer	IdentityLayer
LRNLayer	MatrixMultiplyLayer
PaddingLayer	ParametricReluLayer
PluginV2Layer	PoolingLayer
PoolingLayer	QuantizeLayer
RaggedSoftMaxLayer	ReduceLayer
ResizeLayer	ReverseSequenceLayer
RNNLayer	ScaleLayer
ScatterLayer	SelectLayer
ShapeLayer	ShuffleLayer
SliceLayer	SoftMaxLayer
TopKLayer	UnaryLayer

in Table D.2). However, it significantly increases the engineering effort compared to using off-the-shelf layers.

Due to the hardware and software constraints, a neural network should contain as few (if any) unsupported operations and layers as possible for easier deployment.

D.3 Empirical Evaluation of 3D Neural Networks

Some commonly used architectures in 3D deep learning are summarized in Section 2.2.9. This section provides an empirical evaluation of their deployment-friendliness. The more star marks they have, the easier they can be deployed in practice.

1. Voxe-based methods (*cf.* Section 2.2.4).
   Evaluation: ★.
   Reason: They heavily rely on hash maps and sparse operations, which current software and hardware do not support well.

2. PointNet variants (*cf.* Section 2.2.5).
   Evaluation: ★.
   Reason: They hierarchically sample, group, and interpolate unstructured point clouds. The sampling, grouping, and interpolation algorithms are frequently called in each forward pass but are not standard in most frameworks.

3. Point cloud convolution-based methods (*cf.* Section 2.2.6).
   Evaluation: ★.
   Reason: They have similar issues as PointNet variants.

4. Transformer-based methods (*cf.* Section 2.2.7).
   Evaluation: ★★.
   Reason: the standard transformers consist of simple operations and layers, *e.g.*, the soft-max and matrix multiplication (*cf.* Section 2.1.3). However, current hardware and software are not well-optimized for transformers, since the architecture is relatively new [235]. Also, some variants rely on down-sampling and local attention (Section 2.2.7) and have similar issues of non-standard layers as PointNet variants.

5. Projection-based methods (*cf.* Section 2.2.8).
   Evaluation: ★★★.
   Reason: They adopt mature and standard 2D CNNs, which are well-supported across various hardware and software.

E Evaluation with Own Data

In Chapter 3, the proposed 2.5D-VoteNet is evaluated on openly accessible datasets, *e.g.*, SUN RGB-D [212] and ScanNet [43], for better comparability with state-of-the-art methods. To further investigate its generalization, the author has captured data using an Intel L515 depth camera (*cf*. Table A.1) at the Institute of Industrial Information Technology (IIIT) at Karlsruhe Institute of Technology (KIT). Data samples are captured primarily in offices, libraries, and conference rooms. The camera is held by hand. The camera's pitch and roll angles are obtained using the integrated inertial measurement unit (IMU) in the camera.

The detector is pre-trained using DPCo on the ScanNet dataset (*cf*. Chapter 4), fine-tuned on the SUN RGB-D dataset, and finally evaluated using data captured at IIIT. Some results are visualized in Figure E.1. Note that the ScanNet and the SUN RGB-D dataset use other cameras (*e.g.*, Microsoft Kinect) instead of Intel RealSense L515. However, the detector performs well despite the domain gap, demonstrating the good generalization of this neural network. However, since the evaluation data are not annotated, quantitative metrics, *e.g.*, AP25 and AP50, are unavailable.

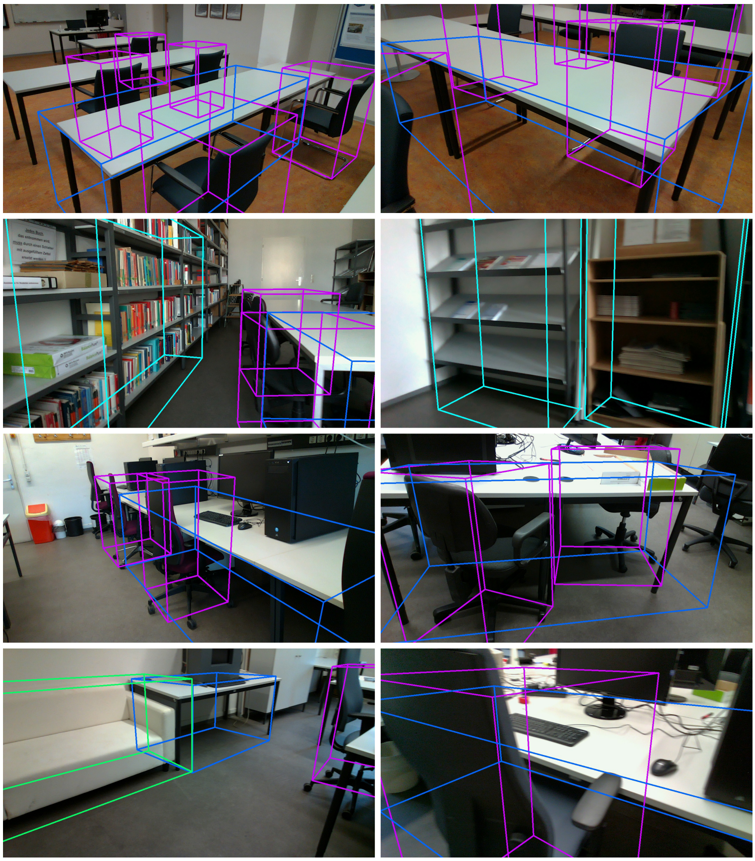


Figure E.1 Object detection results of 2.5D-VoteNet on data captured at IIIT. 3D bounding boxes are projected to RGB images. Red: chair. Blue: table. Light blue: bookshelf. Green: sofa.

Bibliography

[1] **Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, *et al.*** *TensorFlow: a system for large-scale machine learning*. In: *USENIX symposium on operating systems design and implementation*. Vol. 16. 2016. Savannah, GA, USA. 2016, pp. 265–283.

[2] **Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas**. *Learning representations and generative models for 3D point clouds*. In: *International conference on machine learning*. PMLR. 2018, pp. 40–49.

[3] **Syeda Mariam Ahmed and Chee Meng Chew**. *Density-based clustering for 3D object detection in point clouds*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10608–10617.

[4] **Wafaa Alakwaa, Mohammad Nassef, and Amr Badr**. *Lung cancer detection and classification with 3D convolutional neural network (3D-CNN)*. In: *International journal of advanced computer science and applications* 8.8 (2017).

[5] **Johannes Anastasiadis and Michael Heizmann**. *Generation of artificial training data for spectral unmixing by modelling spectral variability using Gaussian random variables*. In: *OCM* (2021), pp. 129–139.

[6] **Johannes Anastasiadis and Michael Heizmann**. *GAN-regularized augmentation strategy for spectral datasets*. In: *tm-Technisches Messen* 89.4 (2022), pp. 278–288.

[7] **Mihael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl**. *3D shape histograms for similarity search and classification in spatial databases*. In: *Advances in spatial databases: 6th International Symposium, SSD'99 Hong Kong, China, July 20—23, 1999 Proceedings 6*. Springer. 1999, pp. 207–226.

[8] **Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey**. *PointNetLK: Robust & efficient point cloud registration using PointNet*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 7163–7172.

[9] **Apple Newsroom**. *Apple unveils new iPad Pro with breakthrough LiDAR scanner and brings trackpad support to iPadOS*. Last accessed on 2023.03.18. 2020. URL: https://www.apple.com/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados/.

[10] **Iro Armeni, Sasha Sax, Amir Roshan Zamir, and Silvio Savarese**. *Joint 2D-3D-Semantic Data for Indoor Scene Understanding*. In: *CoRR* abs/1702.01105 (2017).

[11] **Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton**. *Layer normalization*. In: *arXiv preprint arXiv:1607.06450* (2016).

[12] **Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli**. *Data2Vec: A general framework for self-supervised learning in speech, vision and language*. In: *International conference on machine learning*. PMLR. 2022, pp. 1298–1312.

[13] **Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei**. *BEiT: BERT pre-training of image transformers*. In: *arXiv preprint arXiv:2106.08254* (2021).

[14] **Adrien Bardes, Jean Ponce, and Yann Lecun**. *VICReg: Variance-invariance-covariance regularization for self-supervised learning*. In: *ICLR 2022-10th International Conference on Learning Representations*. 2022.

[15] **Simon Bäuerle, Jonas Barth, Elton Tavares de Menezes, Andreas Steimer, and Ralf Mikut**. *CAD2Real: Deep learning with domain randomization of CAD data for 3D pose estimation of electronic control unit housings*. In: *Proceedings–30. Workshop Computational Intelligence*. KIT Scientific Publishing. 2020, pp. 33–52.

[16] **Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool**. *Speeded-up robust features (SURF)*. In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[17] **Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe**. *Tracking without bells and whistles*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 941–951.

[18] **David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A. Raffel**. *MixMatch: A holistic approach to semi-supervised learning*. In: *Advances in neural information processing systems* 32 (2019).

[19] **Alex Bewley, Pei Sun, Thomas Mensink, Dragomir Anguelov, and Cristian Sminchisescu**. *Range conditioned dilated convolutions for scale invariant 3D object detection*. In: *arXiv preprint arXiv:2005.09927* (2020).

[20] **Charles Bouveyron and Stéphane Girard**. *Robust supervised classification with mixture models: Learning from data with uncertain labels*. In: *Pattern recognition* 42.11 (2009), pp. 2649–2658.

[21] **James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, *et al.*** *JAX: composable transformations of Python+NumPy programs*. In: (2018).

[22] **Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom**. *nuScenes: A multimodal dataset for autonomous driving*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[23] **Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko**. *End-to-end object detection with transformers*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer. 2020, pp. 213–229.

[24] **Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin**. *Emerging properties in self-supervised vision transformers*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9650–9660.

[25] **Krishna Chaitanya, Ertunc Erdil, Neerav Karani, and Ender Konukoglu**. *Contrastive learning of global and local features for medical image segmentation with limited annotations*. In: *Advances in neural information processing systems* 33 (2020), pp. 12546–12558.

[26] **Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, *et al.*** *ShapeNet: An information-rich 3D model repository*. In: *arXiv preprint arXiv:1512.03012* (2015).

[27] **Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays**. *Argoverse: 3D tracking and forecasting with rich maps*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

[28] **Jintai Chen, Biwen Lei, Qingyu Song, Haochao Ying, Danny Z Chen, and Jian Wu**. *A hierarchical graph network for 3D object detection on point clouds*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 392–401.

[29] **Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille**. *DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs*. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[30] **Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, *et al.*** *TVM: An automated end-to-end optimizing compiler for deep learning*. In: *arXiv preprint arXiv:1802.04799* (2018).

[31] **Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton**. *A simple framework for contrastive learning of visual representations*. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[32] **Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia**. *Multi-view 3D object detection network for autonomous driving*. In: *The IEEE Conference on computer vision and pattern recognition*. 2017, pp. 1907–1915.

[33] **Xinlei Chen and Kaiming He**. *Exploring simple Siamese representation learning*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 15750–15758.

[34] **Xinlei Chen, Saining Xie, and Kaiming He**. *An empirical study of training self-supervised vision transformers*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9640–9649.

[35] **Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He**. *Improved baselines with momentum contrastive learning*. In: *arXiv preprint arXiv:2003.04297* (2020).

[36] **Yujin Chen, Matthias Nießner, and Angela Dai**. *4DContrast: Contrastive learning with dynamic correspondences for 3D scene understanding*. In: *Proceedings of the european conference on computer vision*. 2022.

[37] **Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich**. *GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks*. In: *International conference on machine learning*. PMLR. 2018, pp. 794–803.

[38] **Christopher Choy, Wei Dong, and Vladlen Koltun**. *Deep global registration*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2514–2523.

[39] **Christopher Choy, JunYoung Gwak, and Silvio Savarese**. *4D spatio-temporal ConvNets: Minkowski convolutional neural networks*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3075–3084.

[40] **Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray**. *Visual categorization with bags of keypoints*. In: *Workshop on statistical learning in computer vision, ECCV*. Vol. 1. 1-22. Prague. 2004, pp. 1–2.

[41] **Zhicheng Cui, Wenlin Chen, and Yixin Chen**. *Multi-scale convolutional neural networks for time series classification*. In: *arXiv preprint arXiv:1603.06995* (2016).

[42] **Angela Dai and Matthias Nießner**. *3DMV: Joint 3D-multi-view prediction for 3D semantic scene segmentation*. In: *Proceedings of the european conference on computer vision*. 2018, pp. 452–468.

[43] **Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner**. *ScanNet: Richly-annotated 3D reconstructions of indoor scenes*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5828–5839.

[44] **Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei**. *ImageNet: A large-scale hierarchical image database*. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[45] **Jean-Emmanuel Deschaud, David Duque, Jean Pierre Richa, Santiago Velasco-Forero, Beatriz Marcotegui, and François Goulette**. *Paris-CARLA-3D: A real and synthetic outdoor point cloud dataset for challenging tasks in 3D mapping*. In: *Remote sensing* 13.22 (2021), p. 4713.

[46] **Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova**. *BERT: Pre-training of deep bidirectional transformers for language understanding*. In: *arXiv preprint arXiv:1810.04805* (2018).

[47] **Shifei Ding, Hui Li, Chunyang Su, Junzhao Yu, and Fengxiang Jin**. *Evolutionary artificial neural networks: a review.* In: *Artificial intelligence review* 39.3 (2013).

[48] **Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun**. *CARLA: An Open Urban Driving Simulator*. In: *Proceedings of the 1st annual conference on robot learning*. 2017, pp. 1–16.

[49] **Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, *et al.*** *An image is worth 16x16 words: Transformers for image recognition at scale*. In: *arXiv preprint arXiv:2010.11929* (2020).

[50] **Hongyuan Du, Linjun Li, Bo Liu, and Nuno Vasconcelos**. *SPOT: Selective point cloud voting for better proposal in point cloud object detection*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*. Springer. 2020, pp. 230–247.

[51] **Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam**. *ShiDianNao: Shifting vision processing closer to the sensor*. In: *2015 ACM/IEEE 42nd annual international symposium on computer architecture*. 2015, pp. 92–104.

[52] **Ronen Eldan and Ohad Shamir**. *The power of depth for feedforward neural networks*. In: *Conference on learning theory*. PMLR. 2016, pp. 907–940.

[53] **Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter**. *Neural architecture search: A survey*. In: vol. 20. 1. JMLR. org, 2019, pp. 1997–2017.

[54] **Nico Engel, Vasileios Belagiannis, and Klaus Dietmayer**. *Point transformer*. In: *IEEE Access* 9 (2021), pp. 134826–134840.

[55] **Jose M Facil, Benjamin Ummenhofer, Huizhong Zhou, Luis Montesano, Thomas Brox, and Javier Civera**. *CAM-Convs: Camera-aware multi-scale convolutions for single-view depth*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 11826–11835.

[56] **Lue Fan, Xuan Xiong, Feng Wang, Naiyan Wang, and Zhaoxiang Zhang**. *RangeDet: In defense of range view for LiDAR-based 3D object detection*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 2918–2927.

[57] **William Fedus, Ian Goodfellow, and Andrew M Dai**. *MaskGAN: better text generation via filling in the _*. In: *arXiv preprint arXiv:1801.07736* (2018).

[58] **Li Fei-Fei**. *Knowledge transfer in learning to recognize visual objects classes*. In: *Proceedings of the International Conference on Development and Learning*. Vol. 11. 2006.

[59] **Li Fei-Fei, Robert Fergus, and Pietro Perona**. *One-shot learning of object categories*. In: *IEEE transactions on pattern analysis and machine intelligence* 28.4 (2006), pp. 594–611.

[60] **Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman**. *Detect to track and track to detect*. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3038–3046.

[61] **Whye Kit Fong, Rohit Mohan, Juana Valeria Hurtado, Lubing Zhou, Holger Caesar, Oscar Beijbom, and Abhinav Valada**. *Panoptic nuScenes: A large-scale benchmark for LiDAR panoptic segmentation and tracking*. In: *IEEE robotics and automation letters* 7.2 (2022), pp. 3795–3802.

[62] **Kexue Fu, Peng Gao, ShaoLei Liu, Renrui Zhang, Yu Qiao, and Manning Wang**. *POS-BERT: Point cloud one-stage BERT pre-training*. In: *arXiv preprint arXiv:2204.00989* (2022).

[63] **Kunihiko Fukushima**. *Cognitron: A self-organizing multilayered neural network*. In: *Biological cybernetics* 20.3-4 (1975), pp. 121–136.

[64] **Kunihiko Fukushima**. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[65] **Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun**. *Vision meets robotics: The KITTI dataset*. In: *The international journal of robotics research* 32.11 (2013), pp. 1231–1237.

[66] **Spyros Gidaris, Praveer Singh, and Nikos Komodakis**. *Unsupervised representation learning by predicting image rotations*. In: *arXiv preprint arXiv:1803.07728* (2018).

[67] **Zan Gojcic, Caifa Zhou, Jan D. Wegner, Leonidas J. Guibas, and Tolga Birdal**. *Learning multiview 3D point cloud registration*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1759–1769.

[68] **Ian Goodfellow, Yoshua Bengio, and Aaron Courville**. *Deep learning*. MIT press, 2016.

[69] **Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio**. *Generative adversarial nets*. In: *Proceedings of the 27th international conference on neural information processing systems*. 2014, pp. 2672–2680.

[70] **Ben Graham**. *Sparse 3D convolutional neural networks*. In: *arXiv preprint arXiv:1505.02890* (2015).

[71] **David Griffiths and Jan Boehm**. *SynthCity: A large scale synthetic point cloud*. In: *arXiv preprint arXiv:1907.04758* (2019).

[72] **Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, *et al.*** *Bootstrap your own latent: A new approach to self-supervised learning*. In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284.

[73] **Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree**. *BlenSor: Blender sensor simulation toolbox*. In: *Advances in Visual Computing: 7th International Symposium, ISVC 2011, Las Vegas, NV, USA, September 26-28, 2011. Proceedings, Part II 7*. Springer. 2011, pp. 199–208.

[74] **Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C. Courville**. *Improved training of Wasserstein GANs*. In: *Advances in neural information processing systems* 30 (2017).

[75] **Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu**. *PCT: Point cloud transformer*. In: *Computational visual media* 7 (2021), pp. 187–199.

[76] **JunYoung Gwak, Christopher Choy, and Silvio Savarese**. *Generative sparse detection networks for 3D single-shot object detection*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*. Springer. 2020, pp. 297–313.

[77] **Lei Han, Tian Zheng, Lan Xu, and Lu Fang**. *OccuSeg: Occupancy-aware 3D instance segmentation*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2940–2949.

[78] **Song Han, Huizi Mao, and William J Dally**. *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*. In: 2015.

[79] **Kaveh Hassani and Mike Haley**. *Unsupervised multi-task feature learning on point clouds*. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 8160–8171.

[80] **Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun**. *Deep residual learning for image recognition*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[81] **Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick**. *Mask R-CNN*. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[82] **Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick**. *Momentum contrast for unsupervised visual representation learning*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738.

[83] **Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick**. *Masked autoencoders are scalable vision learners*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 16000–16009.

[84] **Dan Hendrycks, Kimin Lee, and Mantas Mazeika**. *Using pre-training can improve model robustness and uncertainty*. In: *International conference on machine learning*. PMLR. 2019, pp. 2712–2721.

[85] **Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski**. *Monte Carlo convolution for learning on non-uniformly sampled point clouds*. In: *ACM Transactions on Graphics (TOG)* 37.6 (2018), pp. 1–12.

[86] **Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou**. *Deep learning scaling is predictable, empirically*. In: *arXiv preprint arXiv:1712.00409* (2017).

[87] **Geoffrey Hinton, Oriol Vinyals, and Jeff Dean**. *Distilling the knowledge in a neural network*. In: *arXiv preprint arXiv:1503.02531* (2015).

[88] **Sepp Hochreiter and Jürgen Schmidhuber**. *Long short-term memory*. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[89] **Sven T. S. Holmström, Utku Baran, and Hakan Urey**. *MEMS laser scanners: A review*. In: *Journal of microelectromechanical systems* 23.2 (2014), pp. 259–275.

[90] **Berthold Klaus Paul Horn**. *Extended Gaussian images*. In: *Proceedings of the IEEE* 72.12 (1984), pp. 1671–1686.

[91] **Kurt Hornik, Maxwell Stinchcombe, and Halbert White**. *Multilayer feedforward networks are universal approximators*. In: *Neural networks* 2.5 (1989), pp. 359–366.

[92] **Kurt Hornik, Maxwell Stinchcombe, and Halbert White**. *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*. In: *Neural networks* 3.5 (1990), pp. 551–560.

[93] **Ji Hou, Angela Dai, and Matthias Nießner**. *3D-SIS: 3D semantic instance segmentation of RGB-D scans*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4421–4430.

[94] **Ji Hou, Benjamin Graham, Matthias Nießner, and Saining Xie**. *Exploring data-efficient 3D scene understanding with contrastive scene contexts*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 15587–15597.

[95] **Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, *et al.*** *Searching for MobileNetV3*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1314–1324.

[96] **Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham**. *RandLA-Net: Efficient semantic segmentation of large-scale point clouds*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11108–11117.

[97] **Junjie Huang, Guan Huang, Zheng Zhu, Yun Ye, and Dalong Du**. *BEVDet: High-performance multi-camera 3D object detection in bird-eye-view*. In: *arXiv preprint arXiv:2112.11790* (2021).

[98] **Tengteng Huang, Zhe Liu, Xiwu Chen, and Xiang Bai**. *EPNet: Enhancing point features with image semantics for 3D object detection*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*. Springer. 2020, pp. 35–52.

[99] **Xiaojie Huang, Junjie Shan, and Vivek Vaidya**. *Lung nodule detection in CT using 3D convolutional neural networks*. In: *2017 IEEE 14th international symposium on biomedical imaging*. IEEE. 2017, pp. 379–383.

[100] **Le Hui, Hang Yang, Mingmei Cheng, Jin Xie, and Jian Yang**. *Pyramid point cloud transformer for large-scale place recognition*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 6098–6107.

[101] **Braden Hurl, Krzysztof Czarnecki, and Steven Waslander**. *Precise synthetic image and LiDAR (PreSil) dataset for autonomous vehicle perception*. In: *2019 IEEE intelligent vehicles symposium*. IEEE. 2019, pp. 2522–2529.

[102] **Sergey Ioffe and Christian Szegedy**. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[103] **Jeremy Irvin, Pranav Rajpurkar, Michael Ko, Yifan Yu, Silviana Ciurea-Ilcus, Chris Chute, Henrik Marklund, Behzad Haghgoo, Robyn Ball, Katie Shpanskaya, *et al.*** *CheXpert: A large chest radiograph dataset with uncertainty labels and expert comparison*. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 590–597.

[104] **Heewoo Jun and Alex Nichol**. *Shap-E: Generating conditional 3D implicit functions*. In: *arXiv preprint arXiv:2305.02463* (2023).

[105] **Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila**. *Analyzing and improving the image quality of stylegan*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 8110–8119.

[106] **Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe**. *Poisson surface reconstruction*. In: *Proceedings of the fourth eurographics symposium on geometry processing*. Vol. 7. 2006, p. 0.

[107] **Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz**. *Rotation invariant spherical harmonic representation of 3D shape descriptors*. In: *Symposium on geometry processing*. Vol. 6. 2003, pp. 156–164.

[108] **Alex Kendall, Yarin Gal, and Roberto Cipolla**. *Multi-task learning using uncertainty to weigh losses for scene geometry and semantics*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7482–7491.

[109] **Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer**. *I-BERT: Integer-only BERT quantization*. In: *International conference on machine learning*. PMLR. 2021, pp. 5506–5518.

[110] **Wonjae Kim, Bokyung Son, and Ildoo Kim**. *Vilt: Vision-and-language transformer without convolution or region supervision*. In: *International conference on machine learning*. PMLR. 2021, pp. 5583–5594.

[111] **Diederik P Kingma and Jimmy Ba**. *Adam: A method for stochastic optimization*. In: *arXiv preprint arXiv:1412.6980* (2014).

[112] **Simon Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R Ledsam, Klaus Maier-Hein, SM Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger**. *A probabilistic U-Net for segmentation of ambiguous images*. In: *Advances in neural information processing systems* 31 (2018).

[113] **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.

[114] **Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton**. *ImageNet classification with deep convolutional neural Networks*. In: *Advances in neural information processing systems*. Vol. 25. Curran Associates, Inc., 2012.

[115] **Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L. Waslander**. *Joint 3D proposal generation and object detection from view aggregation*. In: *2018 IEEE/RSJ international conference on intelligent Robots and systems*. IEEE. 2018, pp. 1–8.

[116] **Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, *et al.*** *The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale*. In: *International journal of computer vision* 128.7 (2020), pp. 1956–1981.

[117] **Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom**. *PointPillars: Fast encoders for object detection from point clouds*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12697–12705.

[118] **Yann LeCun**. *A theoretical framework for back-propagation*. In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. 1988, pp. 21–28.

[119] **Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel**. *Handwritten digit recognition with a back-propagation network*. In: *Advances in neural information processing systems* 2 (1989).

[120] **Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller**. *Efficient backprop*. In: *Neural networks: Tricks of the trade*. Springer, 2002, pp. 9–50.

[121] **Bo Li**. *3D fully convolutional network for vehicle detection in point cloud*. In: *2017 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2017, pp. 1513–1518.

[122] **Bo Li, Tianlei Zhang, and Tian Xia**. *Vehicle detection from 3D LiDAR using fully convolutional network*. In: *arXiv preprint arXiv:1608.07916* (2016).

[123] **Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollar, Kaiming He, and Ross Girshick**. *Benchmarking detection transfer learning with vision transformers*. In: *arXiv preprint arXiv:2111.11429* (2021).

[124] **Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He**. *Exploring plain vision transformer backbones for object detection*. In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IX*. Springer. 2022, pp. 280–296.

[125] **Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen**. *PointCNN: Convolution on X-transformed points*. In: *Advances in neural information processing systems* 31 (2018).

[126] **Ying Li, Lingfei Ma, Zilong Zhong, Fei Liu, Michael A. Chapman, Dongpu Cao, and Jonathan Li**. *Deep learning for LiDAR point clouds in autonomous driving: A Review*. In: *IEEE transactions on neural networks and learning systems* 32.8 (2021), pp. 3412–3432.

[127] **Zhidong Liang, Ming Zhang, Zehan Zhang, Xian Zhao, and Shiliang Pu**. *RangeRCNN: Towards fast and accurate 3D object detection with range image representation*. In: *arXiv preprint arXiv:2009.00206* (2020).

[128] **Zhidong Liang, Zehan Zhang, Ming Zhang, Xian Zhao, and Shiliang Pu**. *RangeIoUDet: Range image based real-time 3D object detector optimized by intersection over union*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 7140–7149.

[129] **Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick**. *Microsoft COCO: Common objects in context*. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[130] **Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie**. *Feature pyramid networks for object detection*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[131] **Haotian Liu, Mu Cai, and Yong Jae Lee**. *Masked Discrimination for Self-Supervised Learning on Point Clouds*. In: *Proceedings of the european conference on computer vision* (2022).

[132] **Minzhe Liu, Qiang Zhou, Hengshuang Zhao, Jianing Li, Yuan Du, Kurt Keutzer, Li Du, and Shanghang Zhang**. *Prototype-Voxel contrastive learning for LiDAR point cloud panoptic segmentation*. In: *2022 international conference on robotics and automation*. IEEE. 2022, pp. 9243–9250.

[133] **Yingfei Liu, Tiancai Wang, Xiangyu Zhang, and Jian Sun**. *PETR: Position embedding transformation for multi-view 3D object detection*. In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVII*. Springer. 2022, pp. 531–548.

[134] **Yueh-Cheng Liu, Yu-Kai Huang, HungYueh Chiang, Hung-Ting Su, Zhe Yu Liu, Chin-Tang Chen, Ching-Yu Tseng, and Winston H. Hsu**. *Learning from 2D: Pixel-to-point knowledge transfer for 3D pretraining*. In: *CoRR* abs/2104.04687 (2021).

[135] **Yunze Liu, Li Yi, Shanghang Zhang, Qingnan Fan, Thomas A. Funkhouser, and Hao Dong**. *P4Contrast: contrastive learning with pairs of point-pixel pairs for RGB-D scene understanding*. In: *CoRR* abs/2012.13089 (2020).

[136] **Ze Liu, Zheng Zhang, Yue Cao, Han Hu, and Xin Tong**. *Group-free 3D object detection via transformers*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 2949–2958.

[137] **Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo**. *Swin transformer: Hierarchical vision transformer using shifted windows*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022.

[138] **Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han**. *Point-Voxel CNN for efficient 3D deep learning*. In: *Advances in neural information processing systems* 32 (2019).

[139] **Ilya Loshchilov and Frank Hutter**. *SGDR: Stochastic gradient descent with warm restarts*. In: *arXiv preprint arXiv:1608.03983* (2016).

[140] **Ilya Loshchilov and Frank Hutter**. *Decoupled weight decay regularization*. In: *arXiv preprint arXiv:1711.05101* (2017).

[141] **David G. Lowe**. *Object recognition from local scale-invariant features*. In: *Proceedings of the 7th IEEE international conference on computer vision*. Vol. 2. 1999, 1150–1157 vol.2.

[142] **Shitong Luo and Wei Hu**. *Score-based point cloud denoising*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 4583–4592.

[143] **Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten**. *Exploring the limits of weakly supervised pretraining*. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 181–196.

[144] **Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu**. *Voxel transformer for 3D object detection*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 3164–3173.

[145] **Rodrigo Marcuzzi, Lucas Nunes, Louis Wiesmann, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss**. *Contrastive instance association for 4D panoptic segmentation using sequences of 3D LiDAR scans*. In: *IEEE robotics and automation letters* 7.2 (2022), pp. 1550–1557.

[146] **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*. IEEE. 2015, pp. 922–928.

[147] **Warren S. McCulloch and Walter Pitts**. *A logical calculus of the ideas immanent in nervous activity*. In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[148] **Gregory P. Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K Wellington**. *LaserNet: An efficient probabilistic 3D object detector for autonomous driving*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12677–12686.

[149] **Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio**. *When and why are deep networks better than shallow ones?* In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[150] **Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean**. *Efficient estimation of word representations in vector space*. In: *arXiv preprint arXiv:1301.3781* (2013).

[151] **Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss**. *RangeNet++: Fast and accurate LiDAR semantic segmentation*. In: *2019 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2019, pp. 4213–4220.

[152] **Ishan Misra, Rohit Girdhar, and Armand Joulin**. *An end-to-end transformer model for 3D object detection*. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 2906–2917.

[153] **Norbert Mitschke, Michael Heizmann, Klaus-Henning Noffz, and Ralf Wittmann**. *Gradient based evolution to optimize the structure of convolutional neural networks*. In: *2018 25th IEEE international conference on image processing*. IEEE. 2018, pp. 3438–3442.

[154] **Norbert Mitschke, Michael Heizmann, Klaus-Henning Noffz, and Ralf Wittmann**. *A fixed-point quantization technique for convolutional neural networks based on weight scaling*. In: *2019 IEEE international conference on image processing*. IEEE. 2019, pp. 3836–3840.

[155] **Claus Müller**. *Spherical harmonics*. Vol. 17. Springer, 2006.

[156] **Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen**. *Point-E: A System for generating 3D point clouds from complex prompts*. In: *arXiv preprint arXiv:2212.08751* (2022).

[157] **Mehdi Noroozi and Paolo Favaro**. *Unsupervised learning of visual representations by solving Jigsaw puzzles*. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*. Springer. 2016, pp. 69–84.

[158] **Curtis Northcutt, Lu Jiang, and Isaac Chuang**. *Confident learning: Estimating uncertainty in dataset labels*. In: *Journal of artificial intelligence research* 70 (2021), pp. 1373–1411.

[159] **Pedro O. O Pinheiro, Amjad Almahairi, Ryan Benmalek, Florian Golemo, and Aaron C. Courville**. *Unsupervised learning of dense visual representations*. In: *Advances in neural information processing systems* 33 (2020), pp. 4489–4500.

[160] **Aaron van den Oord, Yazhe Li, and Oriol Vinyals**. *Representation learning with contrastive predictive coding*. In: *arXiv preprint arXiv:1807.03748* (2018).

[161] **OpenAI**. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774.

[162] **Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang**. *3D object detection with PointFormer*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 7463–7472.

[163] **Yatian Pang, Wenxiao Wang, Francis EH Tay, Wei Liu, Yonghong Tian, and Li Yuan**. *Masked autoencoders for point cloud self-supervised learning*. In: *Proceedings of the european conference on computer vision* (2022).

[164] **Chunghyun Park, Yoonwoo Jeong, Minsu Cho, and Jaesik Park**. *Fast point transformer*. In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*. 2022, pp. 16949–16958.

[165] **Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun**. *Colored point cloud registration revisited*. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 143–152.

[166] **Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove**. *DeepSDF: Learning continuous signed distance functions for shape representation*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 165–174.

[167] **Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, *et al.*** *PyTorch: An imperative style, high-performance deep learning library*. In: *Advances in neural information processing systems* 32 (2019).

[168] **Francesca Pistilli, Giulia Fracastoro, Diego Valsesia, and Enrico Magli**. *Learning graph-convolutional representations for point cloud denoising*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*. Springer. 2020, pp. 103–118.

[169] **Charles R. Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas**. *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.

[170] **Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas**. *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.

[171] **Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas**. *PointNet++: Deep hierarchical feature learning on point sets in a metric space*. In: *Advances in neural information processing systems* 30 (2017).

[172] **Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas**. *Frustum PointNets for 3D object detection from RGB-D data*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 918–927.

[173] **Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas**. *Deep Hough voting for 3D object detection in point clouds*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9277–9286.

[174] **Charles R. Qi, Xinlei Chen, Or Litany, and Leonidas J. Guibas**. *ImVoteNet: Boosting 3D object detection in point clouds with image votes*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 4404–4413.

[175] **Guocheng Qian, Xingdi Zhang, Abdullah Hamdi, and Bernard Ghanem**. *Pix4Point: Image pretrained Transformers for 3D point cloud understanding*. In: *arXiv preprint arXiv:2208.12259* (2022).

[176] **Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Hammoud, Mohamed Elhoseiny, and Bernard Ghanem**. *PointNeXt: Revisiting PointNet++ with improved training and scaling strategies*. In: *Advances in neural information processing systems* 35 (2022), pp. 23192–23204.

[177] **Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, *et al.*** *Improving language understanding by generative pre-training*. Last accessed on 2023.03.18. 2018. URL: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

[178] **Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever**. *Zero-shot text-to-image generation*. In: *International conference on machine learning*. PMLR. 2021, pp. 8821–8831.

[179] **Yongming Rao, Benlin Liu, Yi Wei, Jiwen Lu, Cho-Jui Hsieh, and Jie Zhou**. *RandomRooms: unsupervised pre-training from synthetic shapes and randomized layouts for 3D object detection*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 3283–3292.

[180] **Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi**. *You only look once: Unified, real-time object detection*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[181] **Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, *et al.*** *A generalist agent*. In: *arXiv preprint arXiv:2205.06175* (2022).

[182] **Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun**. *Faster R-CNN: Towards real-time object detection with region proposal networks*. In: *Advances in neural information processing systems* 28 (2015).

[183] **Zhile Ren and Erik B. Sudderth**. *Three-dimensional object detection and layout prediction using clouds of oriented gradients*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1525–1533.

[184] **Herbert Robbins and Sutton Monro**. *A stochastic approximation method*. In: *The annals of mathematical statistics* (1951), pp. 400–407.

[185] **Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer**. *High-resolution image synthesis with latent diffusion models*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[186] **Olaf Ronneberger, Philipp Fischer, and Thomas Brox**. *U-Net: Convolutional networks for biomedical image segmentation*. In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241.

[187] **Ricardo Roriz, Jorge Cabral, and Tiago Gomes**. *Automotive LiDAR technology: A survey*. In: *IEEE transactions on intelligent transportation systems* 23.7 (2022), pp. 6282–6297.

[188] **Frank Rosenblatt**. *The perceptron: A probabilistic model for information storage and organization in the brain*. In: *Psychological review* 65.6 (1958), p. 386.

[189] **Holger R. Roth, Hirohisa Oda, Xiangrong Zhou, Natsuki Shimizu, Ying Yang, Yuichiro Hayashi, Masahiro Oda, Michitaka Fujiwara, Kazunari Misawa, and Kensaku Mori**. *An application of cascaded 3D fully convolutional networks for medical image segmentation*. In: *Computerized medical imaging and graphics* 66 (2018), pp. 90–99.

[190] **Scott D Roth**. *Ray casting for modeling solids*. In: *Computer graphics and image processing* 18.2 (1982), pp. 109–144.

[191] **Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski**. *ORB: An efficient alternative to SIFT or SURF*. In: *2011 international conference on computer vision*. 2011, pp. 2564–2571.

[192] **Radu Bogdan Rusu**. *Semantic 3D object maps for everyday manipulation in human living environments*. In: *KI-Künstliche Intelligenz* 24 (2010), pp. 345–348.

[193] **Radu Bogdan Rusu, Nico Blodow, and Michael Beetz**. *Fast point feature histograms (FPFH) for 3D registration*. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 3212–3217.

[194] **Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen**. *MobileNetV2: Inverted residuals and linear bottlenecks*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[195] **Aditya Sanghi**. *Info3D: Representation learning on 3D objects using mutual information maximization and contrastive learning*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*. Springer. 2020, pp. 626–642.

[196] **Jonathan Sauder and Bjarne Sievers**. *Self-supervised deep learning on point clouds by reconstructing space*. In: *Advances in Neural Information Processing Systems* 32 (2019).

[197] **Dietmar Saupe and Dejan V Vranić**. *3D model retrieval with spherical harmonics and moments*. In: *Pattern Recognition: 23rd DAGM Symposium Munich, Germany, September 12–14, 2001 Proceedings 23*. Springer. 2001, pp. 392–397.

[198] **Corentin Sautier, Gilles Puy, Spyros Gidaris, Alexandre Boulch, Andrei Bursuc, and Renaud Marlet**. *Image-to-LiDAR self-supervised distillation for autonomous driving data*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 9891–9901.

[199] **Maximilian Schambach, Jiayang Shi, and Michael Heizmann**. *Spectral reconstruction and disparity from spatio-spectrally coded light fields via multi-task deep learning*. In: *2021 International conference on 3D vision*. IEEE. 2021, pp. 186–196.

[200] **Marcel P. Schilling, Tim Scherr, Friedrich R. Münke, Oliver Neumann, Mark Schutera, Ralf Mikut, and Markus Reischl**. *Automated annotator variability inspection for biomedical image segmentation*. In: *IEEE access* 10 (2022), pp. 2753–2765.

[201] **Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni**. *Green AI*. In: *Communications of the ACM* 63.12 (2020), pp. 54–63.

[202] **Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li**. *PointRCNN: 3D object proposal generation and detection from point cloud*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 770–779.

[203] **Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li**. *PV-RCNN: Point-voxel feature set abstraction for 3D object detection*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10529–10538.

[204] **Weijing Shi and Raj Rajkumar**. *Point-GNN: Graph neural network for 3D object detection in a point cloud*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1711–1719.

[205] **Dongjoo Shin, Jinmook Lee, Jinsu Lee, Juhyoung Lee, and Hoi-Jun Yoo**. *DNPU: An energy-efficient deep-learning processor with heterogeneous multi-core architecture*. In: *IEEE micro* 38.5 (2018), pp. 85–93.

[206] **Martin Simony, Stefan Milzy, Karl Amendey, and Horst-Michael Gross**. *Complex-YOLO: An Euler-region-proposal for real-time 3D object detection on point clouds*. In: *Proceedings of the european conference on computer vision (ECCV) workshops*. 2018.

[207] **Karen Simonyan and Andrew Zisserman**. *Very deep convolutional networks for large-scale image recognition*. In: *arXiv preprint arXiv:1409.1556* (2014).

[208] **Josef Sivic and Andrew Zisserman**. *Video Google: a text retrieval approach to object matching in videos*. In: *Proceedings 9th IEEE international conference on computer vision*. 2003, 1470–1477 vol.2.

[209] **Jake Snell, Kevin Swersky, and Richard Zemel**. *Prototypical networks for few-shot learning*. In: *Advances in neural information processing systems* 30 (2017).

[210] **Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A. Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li**. *FixMatch: Simplifying semi-supervised learning with consistency and confidence*. In: *Advances in neural information processing systems* 33 (2020), pp. 596–608.

[211] **Hessam Sokooti, Bob De Vos, Floris Berendsen, Boudewijn PF Lelieveldt, Ivana Išgum, and Marius Staring**. *Nonrigid image registration using multi-scale 3D convolutional neural networks*. In: *Medical image computing and computer assisted intervention- MICCAI 2017: 20th International Conference, Quebec City, QC, Canada, September 11-13, 2017, Proceedings, Part I 20*. Springer. 2017, pp. 232–239.

[212] **Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao**. *SUN RGB-D: A RGB-D scene understanding benchmark suite*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 567–576.

[213] **Shuran Song and Jianxiong Xiao**. *Deep sliding shapes for amodal 3D object detection in RGB-D images*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 808–816.

[214] **Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller**. *Striving for simplicity: The all convolutional net*. In: *arXiv preprint arXiv:1412.6806* (2014).

[215] **Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller**. *Multi-view convolutional neural networks for 3D shape recognition*. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 945–953.

[216] **Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz**. *SplatNet: Sparse lattice networks for point cloud processing*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2530–2539.

[217] **Bane Sullivan and Alexander Kaszynski**. *PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)*. In: *Journal of open source software* 4.37 (2019), p. 1450.

[218] **Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta**. *Revisiting unreasonable effectiveness of data in deep learning era*. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 843–852.

[219] **Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, *et al.*** *Scalability in perception for autonomous driving: Waymo open dataset*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[220] **Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele**. *Meta-transfer learning for few-shot learning*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 403–412.

[221] **Shizhao Sun, Wei Chen, Liwei Wang, Xiaoguang Liu, and Tie-Yan Liu**. *On the depth of deep neural networks: A theoretical view*. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[222] **Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M. Hospedales**. *Learning to compare: Relation network for few-shot learning*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1199–1208.

[223] **Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich**. *Going deeper with convolutions*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[224] **Richard Szeliski**. *Computer vision: algorithms and applications*. Springer Nature, 2022. Chap. 2.1, pp. 29–51.

[225] **Mingxing Tan and Quoc Le**. *EfficientNet: Rethinking model scaling for convolutional neural networks*. In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[226] **Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng**. *Fourier features let networks learn high frequency functions in low dimensional domains*. In: *Advances in neural information processing systems* 33 (2020), pp. 7537–7547.

[227] **Peng Tang, Chetan Ramaiah, Yan Wang, Ran Xu, and Caiming Xiong**. *Proposal learning for semi-supervised object detection*. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2021, pp. 2291–2301.

[228] **Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas**. *KPConv: Flexible and deformable convolution for point clouds*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6411–6420.

[229] **Yonglong Tian, Olivier J. Henaff, and Aäron van den Oord**. *Divide and contrast: Self-supervised learning from uncurated data*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10063–10074.

[230] **Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, *et al.*** *MLP-Mixer: An all-MLP architecture for vision*. In: *Advances in neural information processing systems* 34 (2021), pp. 24261–24272.

[231] **Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger**. *Sparsity invariant CNNs*. In: *2017 international conference on 3D Vision (3DV)*. IEEE. 2017, pp. 11–20.

[232] **Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung**. *Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1588–1597.

[233] **Jesper E. Van Engelen and Holger H. Hoos**. *A survey on semi-supervised learning*. In: *Machine learning* 109.2 (2020), pp. 373–440.

[234] **Nick Van Gestel, Steven Cuypers, Philip Bleys, and Jean-Pierre Kruth**. *A performance evaluation test for laser line scanners on CMMs*. In: *Optics and lasers in engineering* 47.3 (2009), pp. 336–342.

[235] **Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin**. *Attention is all you need*. In: *Advances in neural information processing systems* 30 (2017).

[236] **Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol**. *Extracting and composing robust features with denoising autoencoders*. In: *Proceedings of the 25th international conference on machine learning*. 2008, pp. 1096–1103.

[237] **Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, *et al.*** *Matching networks for one shot learning*. In: *Advances in neural information processing systems* 29 (2016).

[238] **Sourabh Vora, Alex H. Lang, Bassam Helou, and Oscar Beijbom**. *PointPainting: Sequential fusion for 3D object detection*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 4604–4612.

[239] **Hanchen Wang, Qi Liu, Xiangyu Yue, Joan Lasenby, and Matt J. Kusner**. *Unsupervised point cloud pre-training via occlusion completion*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9782–9792.

[240] **He Wang, Yezhen Cong, Or Litany, Yue Gao, and Leonidas J. Guibas**. *3DIoUMatch: Leveraging iou prediction for semi-supervised 3D object detection*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 14615–14624.

[241] **Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun**. *Deep parametric continuous convolutional neural networks*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2589–2597.

[242] **Weiyao Wang, Du Tran, and Matt Feiszli**. *What makes training multi-modal classification networks hard?* In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 12695–12705.

[243] **Weiyue Wang and Ulrich Neumann**. *Depth-aware CNN for RGB-D segmentation*. In: *Proceedings of the european conference on computer vision*. 2018, pp. 135–150.

[244] **Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann**. *SGPN: Similarity group proposal network for 3D point cloud instance segmentation*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2569–2578.

[245] **Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li**. *Dense contrastive learning for self-supervised visual pre-training*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 3024–3033.

[246] **Yuan Wang, Tianyue Shi, Peng Yun, Lei Tai, and Ming Liu**. *PointSeg: Real-time semantic segmentation based on 3D LiDAR point cloud*. In: *arXiv preprint arXiv:1807.06288* (2018).

[247] **Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon**. *Dynamic graph CNN for learning on point clouds*. In: *ACM Transactions On Graphics (TOG)* 38.5 (2019), pp. 1–12.

[248] **Yue Wang, Vitor Campagnolo Guizilini, Tianyuan Zhang, Yilun Wang, Hang Zhao, and Justin Solomon**. *DETR3D: 3D object detection from multi-view images via 3D-to-2D queries*. In: *Conference on robot learning*. PMLR. 2022, pp. 180–191.

[249] **Xin Wen, Tianyang Li, Zhizhong Han, and Yu-Shen Liu**. *Point cloud completion by skip-attention network with hierarchical folding*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1939–1948.

[250] **Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani**. *3D multi-object tracking: A baseline and new evaluation metrics*. In: *2020 IEEE/RSJ international conference on intelligent robots and systems*. 2020, pp. 10359–10366.

[251] **Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer**. *SqueezeSeg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3D LiDAR point cloud*. In: *2018 IEEE international conference on robotics and automation*. IEEE. 2018, pp. 1887–1893.

[252] **Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer**. *SqueezeSegV2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud*. In: *2019 international conference on robotics and automation*. IEEE. 2019, pp. 4376–4382.

[253] **Chengzhi Wu, Xuelei Bi, Julius Pfrommer, Alexander Cebulla, Simon Mangold, and Jürgen Beyerer**. *Sim2real transfer learning for point cloud segmentation: an industrial application case on autonomous disassembly*. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2023, pp. 4531–4540.

[254] **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.

[255] **Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao**. *3D ShapeNets: A deep representation for volumetric shapes*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[256] **Aoran Xiao, Jiaxing Huang, Dayan Guan, Fangneng Zhan, and Shijian Lu**. *Transfer learning from synthetic to real LiDAR point cloud for semantic segmentation*. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. 3. 2022, pp. 2795–2803.

[257] **Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick**. *Early convolutions help transformers see better*. In: *Advances in neural information processing systems* 34 (2021), pp. 30392–30400.

[258] **Junyuan Xie, Linli Xu, and Enhong Chen**. *Image denoising and inpainting with deep neural networks*. In: *Advances in neural information processing systems* 25 (2012).

[259] **Qian Xie, Yu-Kun Lai, Jing Wu, Zhoutao Wang, Yiming Zhang, Kai Xu, and Jun Wang**. *MLCVNet: Multi-level context VoteNet for 3D object detection*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10447–10456.

[260] **Saining Xie, Jiatao Gu, Demi Guo, Charles R. Qi, Leonidas Guibas, and Or Litany**. *PointContrast: Unsupervised pre-training for 3D point cloud understanding*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer. 2020, pp. 574–591.

[261] **Yajie Xing, Jingbo Wang, and Gang Zeng**. *Malleable 2.5D convolution: Learning receptive fields along the depth-axis for RGB-D scene parsing*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*. Springer. 2020, pp. 555–571.

[262] **Yajie Xing, Jingbo Wang, Xiaokang Chen, and Gang Zeng**. *2.5D convolution for RGB-D semantic segmentation*. In: *2019 IEEE international conference on image processing*. IEEE. 2019, pp. 1410–1414.

[263] **Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha**. *Alternating multi-bit quantization for recurrent neural networks*. In: *International conference on learning representations*. 2018.

[264] **Chenfeng Xu, Shijia Yang, Bohan Zhai, Bichen Wu, Xiangyu Yue, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka**. *Image2point: 3D point-cloud understanding with pretrained 2D ConvNets*. In: *Proceedings of the european conference on computer vision* (2022).

[265] **Danfei Xu, Dragomir Anguelov, and Ashesh Jain**. *PointFusion: Deep sensor fusion for 3D bounding box estimation*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 244–253.

[266] **Mengde Xu, Zheng Zhang, Han Hu, Jianfeng Wang, Lijuan Wang, Fangyun Wei, Xiang Bai, and Zicheng Liu**. *End-to-end semi-supervised object detection with soft teacher*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 3060–3069.

[267] **Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi**. *Scaling for edge inference of deep neural networks*. In: *Nature electronics* 1.4 (2018), pp. 216–222.

[268] **Ryosuke Yamada, Hirokatsu Kataoka, Naoya Chiba, Yukiyasu Domae, and Tetsuya Ogata**. *Point cloud pre-training with natural 3D structures*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 21283–21293.

[269] **Shuicheng Yan, Huan Wang, Thomas S. Huang, Qiong Yang, and Xiaoou Tang**. *Ranking with uncertain labels*. In: *2007 IEEE international conference on multimedia and expo*. IEEE. 2007, pp. 96–99.

[270] **Yan Yan, Yuxing Mao, and Bo Li**. *SECOND: Sparsely embedded convolutional detection*. In: *Sensors* 18.10 (2018), p. 3337.

[271] **Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He**. *ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers*. In: *Advances in neural information processing systems* 35 (2022), pp. 27168–27183.

[272] **Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl**. *Center-based 3D object detection and tracking*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 11784–11793.

[273] **Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu**. *Point-BERT: Pre-training 3D point cloud transformers with masked point modeling*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 19313–19322.

[274] **Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert**. *PCN: Point completion network*. In: *2018 international conference on 3D vision*. 2018, pp. 728–737.

[275] **Matthew D. Zeiler and Rob Fergus**. *Visualizing and understanding convolutional networks*. In: *Computer Vision–ECCV 2014: 13th European conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 818–833.

[276] **Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus**. *Deconvolutional networks*. In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2528–2535.

[277] **Jin Zeng, Gene Cheung, Michael Ng, Jiahao Pang, and Cheng Yang**. *3D point cloud denoising using graph Laplacian regularization of a low dimensional manifold model*. In: *IEEE transactions on image processing* 29 (2019), pp. 3474–3489.

[278] **Cheng Zhang, Haocheng Wan, Xinyi Shen, and Zizhao Wu**. *PatchFormer: An efficient point transformer with patch attention*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 11799–11808.

[279] **Renrui Zhang, Ziyu Guo, Peng Gao, Rongyao Fang, Bin Zhao, Dong Wang, Yu Qiao, and Hongsheng Li**. *Point-M2AE: multi-scale masked autoencoders for hierarchical point cloud pre-training*. In: *arXiv preprint arXiv:2205.14401* (2022).

[280] **Renrui Zhang, Ziyu Guo, Wei Zhang, Kunchang Li, Xupeng Miao, Bin Cui, Yu Qiao, Peng Gao, and Hongsheng Li**. *PointCLIP: Point cloud understanding by CLIP*. In: *Proceedings of the european conference on computer vision*. 2022, pp. 8552–8562.

[281] **Richard Zhang, Phillip Isola, and Alexei A. Efros**. *Colorful image colorization*. In: *Computer Vision – ECCV 2016*. 2016, pp. 649–666.

[282] **Richard Zhang, Phillip Isola, and Alexei A. Efros**. *Split-brain autoencoders: Unsupervised learning by cross-Channel prediction*. In: *Proceedings of the IEEE Conference on computer vision and pattern recognition*. 2017.

[283] **Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh**. *PolarNet: An improved grid representation for online LiDAR point clouds semantic segmentation*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9601–9610.

[284] **Zaiwei Zhang, Bo Sun, Haitao Yang, and Qixing Huang**. *H3DNet: 3D object detection using hybrid geometric primitives*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*. Springer. 2020, pp. 311–329.

[285] **Zaiwei Zhang, Rohit Girdhar, Armand Joulin, and Ishan Misra**. *Self-supervised pretraining of 3D features on any point-cloud*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 10252–10263.

[286] **Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu**. *Convolutional neural networks for time series classification*. In: *Journal of systems engineering and electronics* 28.1 (2017), pp. 162–169.

[287] **Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun**. *Point transformer*. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 16259–16268.

[288] **Na Zhao, Tat-Seng Chua, and Gim Hee Lee**. *SESS: Self-ensembling semi-supervised 3D object detection*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11079–11087.

[289] **Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang**. *IoU loss for 2D/3D object detection*. In: *2019 international conference on 3D vision*. IEEE. 2019, pp. 85–94.

[290] **Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl**. *Tracking objects as points*. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV*. Springer. 2020, pp. 474–490.

[291] **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.

[292] **Zixiang Zhou, Yang Zhang, and Hassan Foroosh**. *Panoptic-PolarNet: Proposal-free LiDAR point cloud panoptic segmentation*. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 13194–13203.

[293] **Barret Zoph and Quoc Le**. *Neural architecture search with reinforcement learning*. In: *International conference on learning representations*. 2017.

List of publications

[294] **Lanxiao Li and Michael Heizmann**. *2.5D-VoteNet: depth map based 3D object detection for real-time applications*. In: *British Machine Vision Conference*. 2021.

[295] **Lanxiao Li and Michael Heizmann**. *A closer look at invariances in self-supervised pre-training for 3D vision*. In: *European Conference on Computer Vision*. Springer. 2022, pp. 656–673.

[296] **Lanxiao Li and Michael Heizmann**. *Applying plain transformers to real-world point clouds*. In: *arXiv preprint arXiv:2303.00086* (2023).

[297] **Lanxiao Li and Michael Heizmann**. *Randomized 3D scene generation for generalizable self-supervised pre-training*. In: *arXiv preprint arXiv:2306.04237* (2023).

List of supervised theses

[298] **Zhuowei Dong**. *Inference acceleration of point based neural networks*. Master thesis. Karlsruher Institut für Technologie (KIT), 2022.

[299] **Chuhao Fan**. *3D object detection in indoor scenario for mobile device applications*. Master thesis. Karlsruher Institut für Technologie (KIT), 2020.

[300] **Hang Ji**. *Transformer-based neural networks for point clouds*. Master thesis. Karlsruher Institut für Technologie (KIT), 2022.

[301] **Chi Nan**. *Multitask learning for indoor scenes: object detection, scene classification, and situation recognition*. Master thesis. Karlsruher Institut für Technologie (KIT), 2020.

[302] **Si Ni**. *Research of cost functions for three dimensional object detection in indoor scenes*. Master thesis. Karlsruher Institut für Technologie (KIT), 2021.

[303] **Kunyu Peng**. *3D object detection in point clouds using Pillar features and multi-attention mechanism*. Master thesis. Karlsruher Institut für Technologie (KIT), 2021.

[304] **Hang Zhao**. *FPGA-based three-dimensional object classification*. Master thesis. Karlsruher Institut für Technologie (KIT), 2020.

[305] **Yifan Zhao**. *Monocular depth estimation with neural networks*. Master thesis. Karlsruher Institut für Technologie (KIT), 2022.