

國立臺灣大學電機資訊學院資訊工程研究所

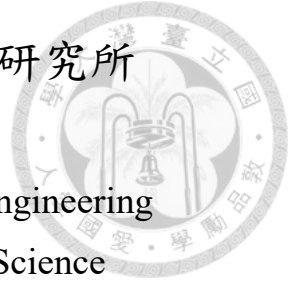
碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis



基於空間感知卷積及形狀補全應用於

擴增實境之三維語意分割

3D Semantic Segmentation based on

Spatial-aware Convolution and Shape Completion for

Augmented Reality Applications

郭耘志

Yun-Chih Guo

指導教授: 傅立成 博士

Advisor: Li-Chen Fu, Ph.D.

中華民國 110 年 1 月

January, 2021

國立臺灣大學碩士學位論文
口試委員會審定書

基於空間感知卷積及形狀補全應用於擴增實境之

三維語意分割

3D Semantic Segmentation based on Spatial-aware
Convolution and Shape Completion for Augmented Reality
Applications

本論文係郭耘志君（學號 R07922094）在國立臺灣大學資訊工程
學系完成之碩士學位論文，於民國 110 年 1 月 27 日承下列考試委
員審查通過及口試及格，特此證明

口試委員：

傅立成

（指導教授）

洪平

歐陽明

鄭龍璿

系主任

洪士瀨



致謝

碩士學業能順利的完成，首先要感謝我的指導老師 傅立成教授，在這兩年半的研究生涯中給了我許多教導和進步的機會，對於專業領域和待人處事也提供了我許多觀點的啟發，使我獲益匪淺。

謝謝家源、紘豪、正皇、力宏這些一同奮鬥的夥伴們，因為有了你們兩年半來的陪伴和鼓勵，在某些枯燥和困難的時刻我才能夠繼續咬牙堅持下去，最終順利完成學業；感謝學長姐恩德、宇閔、宇謙、禹齊、少宏、子翔、雅惠、Robin，對於剛踏入研究所的我給予許多的幫助，在研究和生活上也提供了許多知識和啟發；感謝學妹紫瑄，在我撰寫碩論的過程分擔了實驗和雜務，讓我能毫無後顧之憂地專心研究，同時與學弟丁杰、至瑋、碩恩、嘉鴻，在我後半的研究生涯分享了許多有趣且難忘的事情，使我覺得自己不是一個人，謝謝你們使我保有活力繼續埋進研究的學海中；感謝 RobotLab 的大家，這兩年一同度過的歡樂時光，一定會成為我一輩子的回憶。

最後感謝我的父母在我就讀研究所時給予的幫助和耐心，讓我能夠穩穩地向前邁進，一步一步完成學業，謝謝。

郭耘志 謹致

國立台灣大學資訊工程學研究所

中華民國 110 年 2 月 18 日



中文摘要

三維室內場景語意分割在電腦視覺中是一項十分熱門的研究項目，對於許多應用程序而言，準確了解場景中每個點的類別非常重要，受益於深度學習的發展，已經有許多基於體素和點的神經網絡被提出來解決語義分割問題。但是，大多數都沒有充分考慮空間結構的資訊。

本論文的目的是希望能夠提出一個系統，利用帶有顏色資訊的場景點雲對整個室內場景做語意分割。基於空間資料的稀疏性，我們設計了一個新穎的空間感知稀疏卷積運算。我們使用物體存在的空間資訊編碼作為額外的特徵，並使用自我注意力機制有效整合不同資訊；此外，我們引入了補全網路對分割網路的結果做修正，使場景中每種物體可以得到更加合裡和完整的形狀。透過以上兩點方法，我們建立準確的場景語意分割網路，獲得整個場景的屬性分類。

在實驗的部分，我們使用兩個公開的資料集進行定量和定性的分析，首先我們對不同配置的模型進行比較，證明提出方法的有效性；其次將與其他最先進的方法進行比較，證明提出方法的優越性；最後則是一個實際應用的分析，說明具體的應用性。我們期望提出的三維場景語意分割系統能夠為實際應用提供準確而迅速的結果。

關鍵字：場景語意分割、稀疏卷積網路、空間感知卷積、補全網路、擴增實境



Abstract

3D semantic segmentation of indoor scenes is a popular research topic in the field of computer vision. For many applications, it is very important to know exactly what category each point in the scene belongs to. Benefiting from the development of deep learning, many neural networks based on voxels and points have been proposed to solve semantic segmentation problems. However, most of them don't fully consider the information of the spatial structure.

In this thesis, we propose a system that uses scene point clouds with color information to semantically segment an entire indoor scene. Based on the sparsity of spatial data, we design a novel spatial-aware sparse convolution operation. We encode the spatial information of the object's existence as an additional feature and use the self-attention mechanism to effectively aggregate features. In addition, we introduce a completion network to refine the results from the segmentation network, so that each object in the scene can get a more reasonable and complete shape. Through the above two methods, we build an accurate scene semantic segmentation network to obtain the semantic information of

the entire scene.

In the experimental part, we use two public datasets to perform quantitative and qualitative analysis. We compare our results with other state-of-the-art methods to prove the superiority of the method. Also, we examine our models with different configurations to assure the effectiveness of the proposed method. Finally, An real application is introduced to demonstrate our work. We expect that the proposed 3D scene semantic segmentation system can provide accurate and fast results for practical applications.

Keywords: Scene Semantic Segmentation, Sparse Convolutional Network, Spatial-aware convolution, Completion Network, Augment Reality





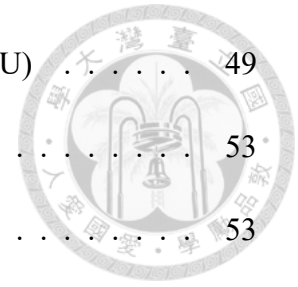
Contents

	Page
口試委員審定書	i
致謝	ii
中文摘要	iii
Abstract	iv
Contents	vi
List of Figures	ix
List of Tables	xi
Chapter 1 Introduction	1
1.1 Motivation	2
1.2 Challenge	3
1.3 Related Work	6
1.4 Contribution	9
1.5 Thesis Organization	10
Chapter 2 Preliminaries	12
2.1 Convolutional Neural Networks	12
2.1.1 Basic Components	14
2.1.2 ResNet	22



2.1.3	3D convolution	22
2.2	Semantic Segmentation Framework	23
2.2.1	FCN	24
2.2.2	U-Net	25
2.3	Sparse Convolutional Neural Networks	26
2.3.1	Sparse Tensor Representation	28
2.3.2	Spatial Hashing	29
2.3.3	Output Coordinate Generation	29
2.3.4	In-out Kernel Mapping	30
2.3.5	Sparse Convolution	31
Chapter 3	Methodology	32
3.1	Model Overview	32
3.2	Spatial-aware Convolution Layer	33
3.2.1	Spatial Occupancy Encoding	35
3.2.2	Self-attention Aggregation	37
3.3	Segmentation Net	39
3.4	Completion Net	41
3.5	Model Training	45
Chapter 4	Experiment Results	47
4.1	Settings of Environment	47
4.2	Datasets and Evaluation Metrics	48
4.2.1	ScanNet	48
4.2.2	Stanford 3D Large-Scale Indoor Spaces (S3DIS)	49

4.2.3	Mean Accuracy and mean Intersection of Union (mIoU)	49
4.3	Experimental Results	53
4.3.1	Ablation Study	53
4.3.2	Comparison with State-of-the-art	55
4.3.3	Qualitative Results	56
4.4	Real-World AR Application	59
Chapter 5	Conclusion	62
	References	63

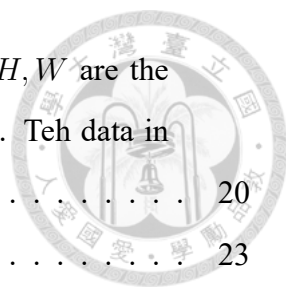




List of Figures

1.1	Illustration of 3D semantic segmentation.	1
1.2	The procedure of simultaneous localization and mapping.	3
2.1	LeNet-5 architecture [1]. Given an input image, the model will use several layers to extract the features and get the final predictions.	13
2.2	Illustration of a convolution operation. The convolutional kernel matches a patch in order to calculate the sum of the products of its features and weights and then slide to the next patch until it has looped over the whole feature map.	15
2.3	The functionality of ReLU, Leaky ReLU, and ReLU6. For the Leaky ReLU approach, λ is a hyperparameter to control the slope. These functions increase nonlinearity of CNNs.	17
2.4	Calculation of max-pooling.	17
2.5	Calculation of strided convolution. In this case, the kernel size and the stride are 2.	18
2.6	Calculation of unpooling convolution.	18
2.7	Illustration of transposed convolution operation. It is the inverse operation of strided convolution shown in Fig 2.5. Notice that the mapping relation remains the same, that is, only the regions with same colors affect each others.	19
2.8	Internal covariate shift phenomenon. In left picture, without normalization, the gradient of w_1 is much smaller than w_2 , so the model needs more steps to converge; in the right picture, with normalization, the scaling of w_1 and w_2 are balanced so that the model converge quickly.	20

2.9	Four different normalization skills. N is the batch axis. H, W are the width and height in the spatial axes. C is the channel axis. The data in blue cubes are normalized by their mean and variance.	20
2.10	Basic residual block [34]	23
2.11	Illustration of 3D convolution operation	23
2.12	Fully convolutional network architecture [8]	25
2.13	The architecture of U-Net [9]	26
2.14	Illustration of a sparse convolution operation. In sparse data, some space is empty, thus the computation on those grid squares can be avoided. The gray block in the kernel represents the avoided computing weights.	27
3.1	Overview of proposed SegCompleteNet.	33
3.2	Illustration of spatial-aware convolution.	34
3.3	Occupancy Embedding. (a) In-out kernel map stored in memory. (b) A example of simple point set in the space.	36
3.4	Architecture of Segmentation Net.	39
3.5	Dissociation procedure.	42
3.6	Completion procedure.	44
4.1	Sample Scenes of ScanNet dataset. Left is input RGB point cloud and Right is Groud Truth.	50
4.2	Sample Scenes of S3DIS dataset. Left is input RGB point cloud and Right is Groud Truth.	51
4.3	The variance of important scores in different layers.	54
4.4	Semantic Segmentation on ScanNet dataset.	58
4.5	Semantic Segmentation on S3DIS dataset.	58
4.6	Visualization of completion results.	60
4.7	Magic Leap One.	61
4.8	Augmented System Communication	61





List of Tables

4.1	The equipment specification of our experiment environment.	48
4.2	Confusion matrix. Four situations happen in a classification task.	52
4.3	Ablation study on Spatial-aware Convolution.	54
4.4	Ablation study on fusion of segmented results.	55
4.5	3D Semantic segmentation results on ScanNet Benchmark. The evaluation metric is mIoU	56
4.6	3D Semantic segmentation results on S3DIS Dataset.	57
4.7	3D Semantic segmentation results on ScanNet Benchmark.	57



Chapter 1 Introduction

3D semantic segmentation of indoor scenes is a very popular field in robotics and augmented reality in recent years. As shown in Fig 1.1, semantic segmentation is a spatially dense prediction task that can basically split the scene and predict different semantic labels of different parts. Many applications are built on the premise of this technology in order to understand the environment. In this thesis, we propose a 3D indoor semantic scene segmentation system given an RGB point cloud. With our proposed modules, the network can better consider the spatial relationship of sparse point clouds and refine the inaccurate semantic prediction of the scene to increase the prediction accuracy. We hope to provide object label information for applications and bring users a better experience.

In this chapter, we will first introduce the motivation and background. Then, we will explain the challenges followed by a brief survey of the related works. Finally, we will summarize our contributions and describe the organization of the rest of the thesis.

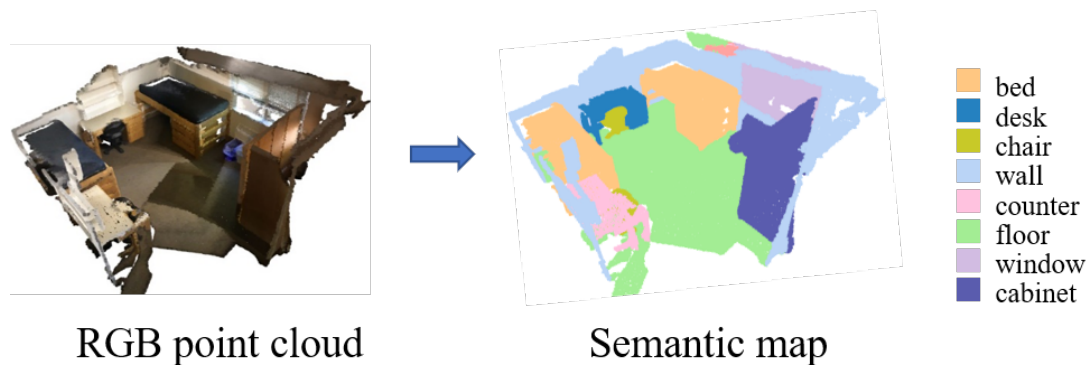


Figure 1.1: Illustration of 3D semantic segmentation.

1.1 Motivation



In 2020, due to the pandemic of the novel coronavirus, remote work has become a fairly common business model. As a highly anticipated solution, augmented reality is expected to effectively eliminate the physical distance between people, so that people in different regions can work as if they are in the same place. In contrast to virtual reality (VR), the biggest difficulty in augmented reality (AR) is to establish the relationship between virtual objects and real space. Unlike virtual objects of which we always know the property, the real environment is unknown and we need to reconstruct the scene properties by ourself. Therefore, improving the understanding of the environment in augmented reality applications becomes an urgent need.

For many years, researchers in robotics are developing a technology called simultaneous localization and mapping (SLAM), which is able to build an environment map based on previously observed information, as shown in Fig 1.2. LIDAR scanners, depth cameras, and inertial measurement unit (IMU) sensors have become more and more affordable for consumers in recent years. SLAM also utilizes the additional information to generate a better-reconstructed map. Most AR capable devices on the market provide SDK such as Google ARCore, Apple ARKit, and Microsoft HoloLens, which help the users to process the environmental information. With this technology, we can reconstruct the geometry of a 3D indoor environment and RGB information of reconstructed points or meshes.

A related technology called environment understanding is widely used in many scenarios, including robotics, autonomous navigation, remote sensing, and several other applications. Knowing the geometry of the scene can already suffice for some simple ap-

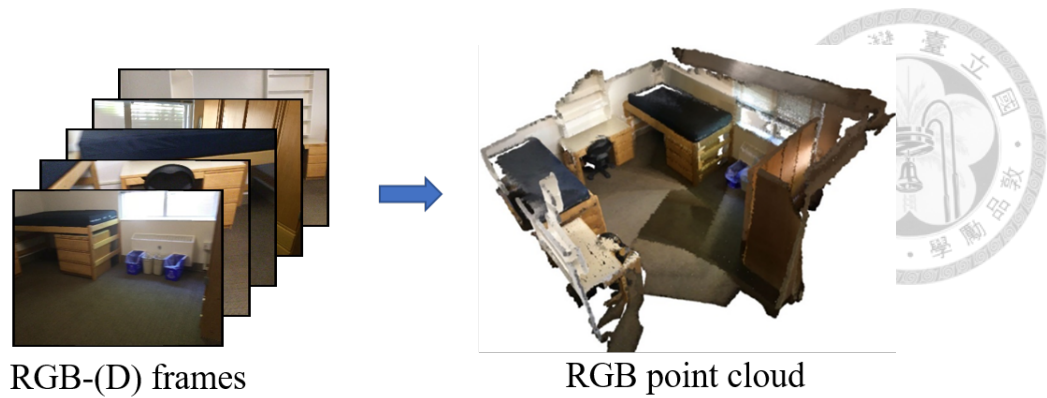


Figure 1.2: The procedure of simultaneous localization and mapping.

plications, such as placing a virtual animal on a ground plane. However, for more mature applications, simply knowing the geometric shape and color information cannot adequately perform more detailed control and operation. In those cases, we prefer to know the semantic information of each reconstructed point so that we can utilize location and classification simultaneously for further features. Therefore, a system that can provide semantic information based on existing 3D scenes is necessary.

In recent years, due to the rapid development of deep learning and the improvement of graphics processing unit (GPU) technology, Convolutional Neural Networks [1] (CNNs) have been widely used to solve many tasks in computer vision, such as image classification [2, 3, 4], object detection [5, 6, 7], and semantic segmentation [8, 9, 10]. Compared to hand-crafted features [11, 12], CNNs can learn more appropriate feature extraction strategies for different tasks to obtain representative features. We want to use its superior feature extraction ability to solve 3D semantic segmentation.

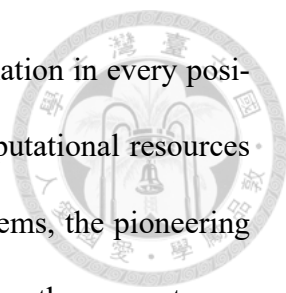
1.2 Challenge

Although we can first use mature 2D convolution to segment observed images and then back project to the 3D scene, this causes concomitant disadvantages of severe occlusion and scale ambiguity of objects. Therefore, it is still a research target with considerable

potential to directly perform convolution in 3D space. 3D scene information is difficult to process than 2D images. In the most common point cloud data structure, points are spread irregularly in the space and stored unorderedly in the memory. How to process this data is an open problem.

Many researchers are dedicated to solving semantic segmentation of 3D scenes directly on the point cloud. These Point-based methods [13, 14, 15, 16] need to define three operations: feature transformation, feature aggregation, and point set sampling. Although feature transformation is basically a fast multilayer perceptron (MLP) to transform the features at each point, the other two operations have high computational complexity when searching neighbors, like kNN or ball-querying operations. For each point, feature aggregation first needs to iterate over all the other points to find its neighbors. The procedure calculates the relative distances and decides which candidate points are the top- k neighbors (kNN) or within a certain distance (ball-querying) for the given point. The procedures basically have $O(N^2)$ time complexity. For point set sampling, most of the approaches adopt Farthest Point Sampling (FPS) strategy to obtain the sub-point set, which also has $O(N^2)$ time complexity. Due to the high computation complexity, these models are limited for large-scale point clouds. Moreover, many experimental studies have also shown that the point-based network using MLP can not admit too deep layer structure, leading to insufficient feature aggregations.

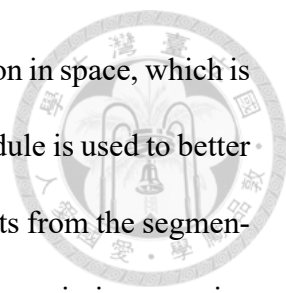
In comparison with point-based methods, voxel-based networks [17, 18], which first transform unstructured points into a structured grid, can be efficiently designed and trained for 3D semantic segmentation by extending regular 2D convolutions, whose capabilities for coarse-grain feature learning and sampling have already been demonstrated. Although 3D convolution has many advantages, these approaches rely on the representation



of 3D voxels, which require a large array or table to store the information in every position. What's worse, such data structure also consumes a lot of computational resources in the empty space during convolution. To mitigate the above problems, the pioneering works [19, 20] started to implement the convolution operation based on the sparse tensor representation. The sparse convolution only performs matrix multiplication of feature and kernel weight on the non-empty space so that it can avoid redundant computation. In practice, they apply a hash table structure to store a sparse tensor and use the position as the hash key, to save memory and efficiently query the features. Although these works exploit the sparsity to accelerate computing, the spatial information brought by the sparsity of data is often neglected. For example, the feature of all empty space in sparse convolution is implicitly treated as zero regardless of the surrounded spatial structure.

After generating the prediction results, another important question is how to refine them. Some methods [21, 22] are proposed to improve the segmentation shape. They are built through statistical models rather than deep neural networks. On the other hand, there is a task called completion used in a common application scenario of deep learning. Given an incomplete scan of an object, such as a table with a missing corner, the completion task hopes to output a more complete object shape, for example a table with legs. The completion network is a generative model that predicts a confidence value for each space to determine whether space is occupied by objects or not, thus allowing objects to grow and eliminate noise. With this method, we can improve the quality of the scene.

This thesis mainly aims to solve the previously discussed lack of sparse convolution and the refinement of existing methods. Based on our proposed spatial-aware convolution, we propose a novel 3D semantic segmentation system, called SegComple Net, to predict accurate 3D semantic labels based on the RGB point cloud. Space-aware convolution uses

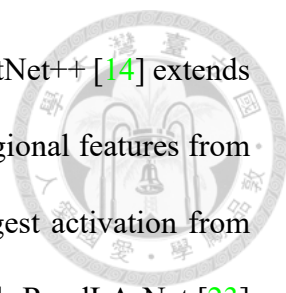


a spatial coding module to enhance the features of structural information in space, which is brought about by the sparsity of data. In addition, the aggregation module is used to better aggregate enhanced functions. On the other hand, the prediction results from the segmentation network are usually not absolutely correct. There may be many missing or noisy segments. We consider using generative completion neural network to compensate for the deficiencies of segmentation. After obtaining the prediction result through the proposed segmentation net, we introduce a completion net to modify the shape of each category in the scene to improve the prediction. In our knowledge, this is the first combination of learnable segmentation and completion which achieves better semantic results on given RGB point clouds. In order to verify our system, we conduct some experiments on two indoor environment data sets ScanNet v2 and S3DIS. Experimental results show that our system can better extract representative features and refine predictions, thereby improving the performance of these semantic segmentation benchmarks.

1.3 Related Work

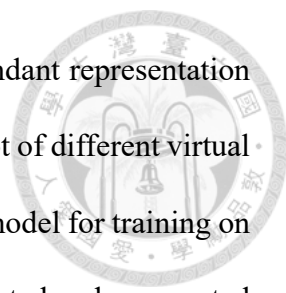
3D semantic segmentation can be viewed as an extension of 2D semantic segmentation. However, unlike from 2D images, the 3D point cloud data is sparse and unstructured. How to efficiently store and perform an operation on this kind of data remains an open problem. There are three main trends to tackle the 3D point cloud data, including point-based methods, projection-based methods, and voxel-based methods.

Point-based Methods: For directly solving the 3D semantic segmentation from an unordered point cloud, PointNet [13] is the first pioneering work that introduces the concept of an asymmetric function which makes a model invariant to the input permutation order. Practically, it extracts the global information by using a point-wise shared multi-



layer perceptron (MLP) followed by the max-pooling operation. PointNet++ [14] extends the PointNet and applies a hierarchical point set structure to learn regional features from the local to global field gradually. Since it simply chooses the largest activation from multiple candidate sub-regions, some information is not fully utilized. RandLA-Net [23] encodes the relative point position and uses attentive pooling to let the model better capture the spatial relations. Motivated by the success of convolution on structured data, Wu *et al.* and Hugues *et al.* came up with the idea of performing convolutions directly on irregular points. These works learn the approximate weight function by interpolating convolutional weights on the receptive area. PointConv [15] learns the weight functions and density functions for aggregating the neighboring points. KPConv [16] defines a point convolution that uses a set of kernel points and calculates the kernel response of each input feature. Due to the essence of searching on irregular data, all the above methods have high time complexity $O(N^2)$ for querying the neighboring points, thus limiting these methods to small-scale data.

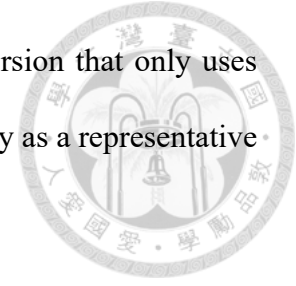
Projection-based Methods: In addition to 3D operations, several works also want to utilize mature 2D convolutions to improve or directly solve the semantic segmentation problem. 2D-to-3D methods like 3DMV [24] and MVPNet [25] try to utilize a set of 2D images at different viewpoints. They conduct 2D convolutions first to extract the high-level image features and then project them to the corresponding voxels. For 3D-to-2D methods, Tangent Convolution [26] projects local neighborhoods on a local tangent plane, while TextureNet [27] projects on geodesic mesh surfaces. Compared to the above methods using a pre-defined surface, FPCConv [28] proposed a more simple and robust projection strategy, which is learning projection weights along the local surface. However, due to the projection process, all of them will unavoidably lose the 3D geometry



details. These methods also suffer from density variations and redundant representation for overlapping parts. The recent Virtual MVFusion [29] sampled a lot of different virtual views of the 3D mesh and only applied a 2D semantic segmentation model for training on rendered images. All features predicted by 2D models are then projected and aggregated on a 3D scene to get the final results. Although this method achieves good results, the computation overhead of projection is very expensive and not fast enough.

Voxel-based Methods: The 3D convolution methods [18, 17] extended from 2D are able to capture spatial features quite well while having a fast neighbor querying speed. However, these kinds of methods using dense representation suffer from heavy computation cost and memory consumption. Since most space is empty and its associated feature is represented as a zero vector, the computation operated on those positions is very redundant. In practice, they always need to crop the subspace and use the sliding window method to jointly process the whole scene due to memory limitation. Therefore, Some subsequent improvement methods have been proposed. OctNet [30] tried to utilize the octree structure to efficiently store the 3D space. The novel SparseConvNet [19] is an outstanding work that uses a hash-table to directly store the points and defines a novel sparse convolution that directly operates on the sparse point data to avoid redundant calculation. MinkowskiNet [20] further proposed a generalized sparse convolution operation. With sparse convolution, the voxel-based method can handle larger scale points than point-methods and fully utilize the advantages of 3D convolution. Many advanced works based on sparse convolution achieve SOTA and became the current mainstream. JSENet [31] took additional edge information into consideration and improved the semantic segmented results. FusionNet [32] utilized voxel-based operation and point-based operation simultaneously which are complementing each other. Our work is also built

on this sparse convolution, but in comparison with our baseline version that only uses the sparsity of space to save computation, we further use this property as a representative feature to indicate the spatial structure.



1.4 Contribution

In this thesis, we propose a system for 3D semantic segmentation from a point cloud with added RGB texture, which improves the performance of the previous works. In summary, the major contributions of this thesis are as follows:

- I. We propose a novel convolution called spatial-aware convolution, which explicitly considers the spatial information on voxels. The proposed spatial encoding module is first used to enhance neighboring visual features by encoding the occupancy information in the space. Then, the self-attention aggregation module uses these enhanced features to calculate the importance score of each grid and performs a weighted summation of all the neighboring features to obtain representative features.
- II. We propose a novel refinement strategy based on the completion task. We first dissociate each category in the entire scene according to the prediction result of our semantic segmentation and then take as input each sample in the network separately. In our proposed completion net, we will continue to expand and trim the point cloud to generate more complete shapes of the objects.
- III. Based on the proposed spatial-aware convolution and refinement strategy, we build a complete 3D semantic segmentation network that can accurately predict semantic labels. The experiments show that our network surpasses the previous methods on S3DIS dataset. We also build a simple augmented reality application for a practical

demonstration.



1.5 Thesis Organization

In the previous parts, we have presented the background of 3D semantic segmentation, the motivation of this research, and the challenging problems we would like to solve. Additionally, we also have described the history of this literature and introduced related works of this thesis followed by a summary of our contributions. The remaining chapters of this thesis are listed below:

Chapter 2 – Preliminaries: In this chapter, some background knowledge of the convolutional neural network (CNN) and the semantic segmentation framework will be introduced. We will first talk about the basic concept of common CNN. Then, we will introduce the advanced architectures like 3D CNN and sparse convolutional neural network (SCNN) which are proposed to efficiently process high-dimensional data. Moreover, we will overview some well known and outstanding semantic segmentation frameworks which are applied in many real world applications.

Chapter 3 – Methodology: In this chapter, we will detail our proposed SegCompleteNet. Starting from an overview of the model’s architecture, we first go into details about the proposed spatial-aware convolution, which includes a spatial encoding module and a self-attention aggregation module. Based on the operation, we construct two networks, segmentation Net and completion Net, to predict and refine the 3D semantic prediction of the scene. In the last part, the training procedure is introduced to give a full description, including the loss function and other implementation details.

Chapter 4 – Experimental results: In this chapter, we will first introduce two common public datasets, ScanNet v2 [17] and S3DIS [33], which are used for evaluating our pro-

posed method and the corresponding evaluation metrics for examining how accurate our semantic segmentation results are. Then, a series of experimental results will be presented to show the superiority of our methods compared to other state-of-the-art methods. Additionally, we will also demonstrate a practical application to visualize the semantically segmented scene through an augmented reality device.

Chapter 5 – Conclusion. In this chapter, we will summarize our contributions and experimental results of the proposed system. Moreover, we will discuss some further issues of the proposed system and the potential future work to improve the performance.



Chapter 2 Preliminaries

In this chapter, we will introduce the preliminary knowledge which is related to our system of this thesis. We will first describe a common deep learning framework called convolutional neural network (CNN), including its key concepts and notable architectures. Then, a novel sparse convolutional neural network (SCNN) which is proposed to handle the high-dimensional Euclidean-structure data while having acceptable computation speed will be introduced. The backbone of our system is composed of a 3D SCNN.

2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) is a very well-known deep learning method that was first proposed by LeNet [1] in 1998. Inspired by the way how an animal learns to recognize a visual pattern, CNNs use a shared spatial kernel to detect any pattern inside images, thus having a strong ability to extract features from images efficiently. Unlike traditional approaches that use hand-craft features, like HOG [12], deep learning-based CNNs are able to learn the feature extraction strategy by adjusting the parameters of the kernels when provided with a large amount of training data. Fig 2.1 shows a simple CNN architecture.

Generally speaking, a CNN is constructed by stacking several different layers that provide different operation layers on top of each other to increase the expressive ability of

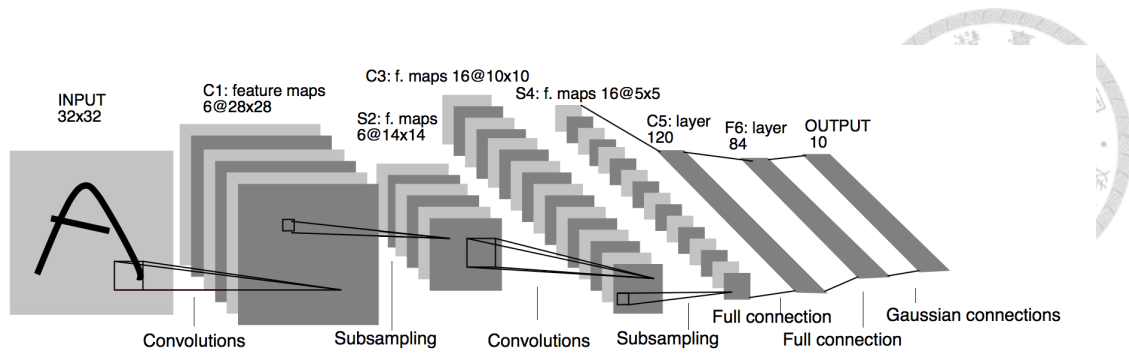
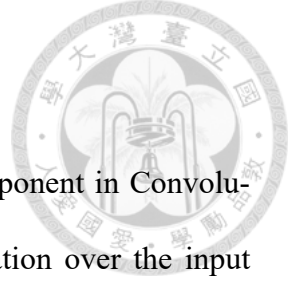


Figure 2.1: LeNet-5 architecture [1]. Given an input image, the model will use several layers to extract the features and get the final predictions.

the networks. During the forward pass procedure, an input image will flow through these layers and jointly process the transformed features, finally generating the model prediction results. During the backward pass procedure, the learnable parameters are updated through the backpropagation process. Starting from Alexnet [2], CNNs have demonstrated their powerful capabilities in the field of computer vision and image processing. They achieved very excellent results in many challenging tasks, such as image classification, object detection, and semantic segmentation.

A Problem for deeper neural network is that more parameters needed to be calculated, so it costs a lot of computation resources. Recently, Due to the high parallel computation ability of the Computer Graphic Unit (GPU), we can move the calculation process into the GPU and greatly reduce the computation time, making CNNs be more practical in the real world. We will introduce some important components of CNNs in the following sub chapters.

Many research focuses on improving the efficiency of extracting the features. For example, VGG16 [3], ResNet-50 [34], Inception v3 [4], ResNext-50 [35] are all examples of common structure to efficiently learn the image features.



2.1.1 Basic Components

Convolutional Layer. Convolutional Layer acts as a key component in Convolutional Neural Networks. It mainly performs the convolution operation over the input feature and extracts the representative output features. Each layer consists of a set of convolutional kernels which have their own learnable weights and optional bias. Users need to predefine the shape and the number of convolutional kernels. A single convolution proceeds as follows: given a pixel, we first find its neighborhood patch, then the convolutional kernel will calculate the dot product between each point feature on the patch and the spatial-corresponding weight, summing them up to serve as new features of the pixel. We match one patch to apply one convolution at a time and then slide the kernel to another patch until finish looping over the whole space to form a complete output feature map. Since each layer has its own set of kernels, they can learn different feature extraction strategies. Generally speaking, the convolutional layers which are positioned at an early stage of the CNN tend to extract low-level features, such as edges, corners, and small color blocks. The convolutional layers at a later stage of the CNN will extract high-level features like human faces, street scenes, and other complicated semantic contexts. Fig 2.2 illustrates the procedure of a convolutional operation.

To denote the convolution in a formula, we first define $x_{\mathbf{c}}^{in} \in \mathbb{R}^{F_{in}}$ as an F_{in} -channel input feature located at the D -dimensional coordinate $\mathbf{c} \in \mathbb{R}^D$. We also define the kernel weight as $W \in \mathbb{R}^{K^D \times F_{in} \times F_{out}}$. A single convolution forward pass performed at position \mathbf{c} is formulated as follows:

$$x_{\mathbf{c}}^{out} = \sum_{\mathbf{i} \in V^D(K)} W_{\mathbf{i}} \cdot x_{\mathbf{c}+\mathbf{i}}^{in} \quad (2.1)$$

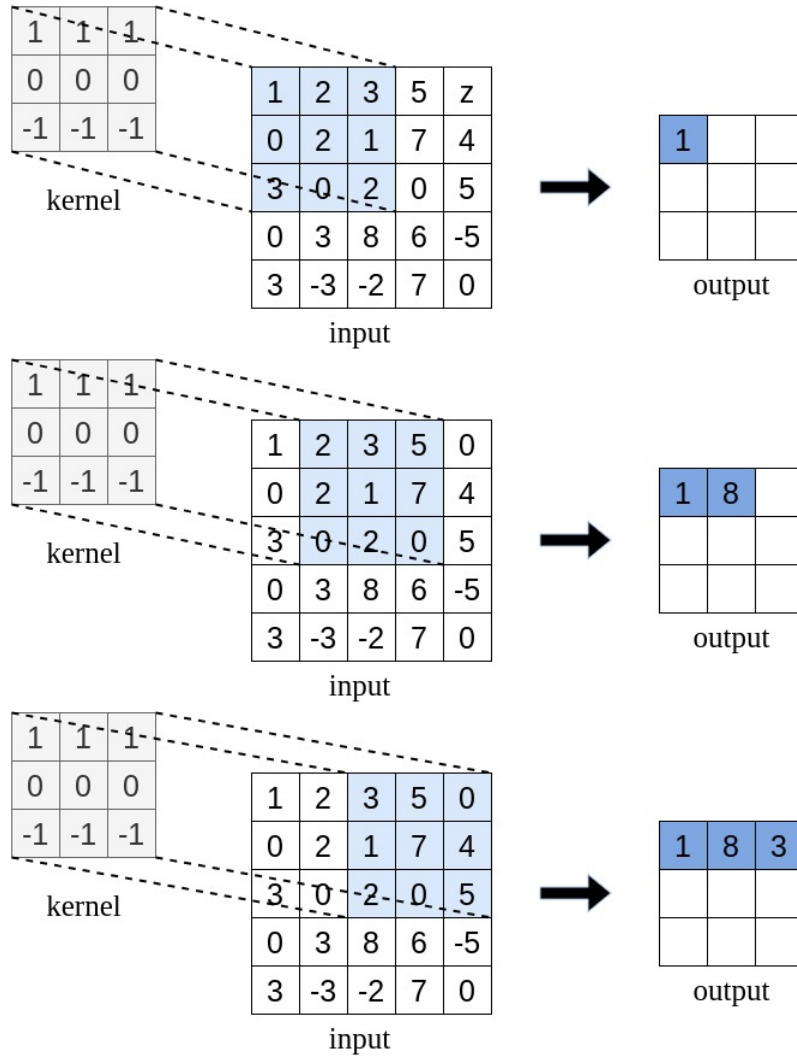


Figure 2.2: Illustration of a convolution operation. The convolutional kernel matches a patch in order to calculate the sum of the products of its features and weights and then slide to the next patch until it has looped over the whole feature map.

where $V^D(K)$ is the reception field of the convolutional kernel centered at the origin, e.g., $V^1(3) = \{-1, 0, 1\}$, and W_i represents the specific slice of W at the first dimension responsible to perform matrix multiplication of x_{c+i}^{in} . In Sec. 2.3, we will further modify the equation to match our case.

Activation Layer. The activation function is an approach to increase the learning capability of the neural network. The original neural network only contains linear dot product operations, which is not sufficient for many complicated tasks that need non-linear mapping. Therefore, other approaches are adding an additional function that can provide the nonlinear properties during the computation of the neural network. One of the most widely used activation functions is Rectified Linear Unit (ReLU). Inspired by the biological nervous system, the neural signal will propagate to the next neuron only when it exceeds a certain threshold. The function of the ReLU is defined as follows:

$$f(x) = \max(x, 0) \quad (2.2)$$

The function activates the positive value but filters the negative values and sets them to 0 in the feature map. It provides a simple but effective non-linear operation, which has been proved to help CNNs to converge more quickly. Some deformation of ReLU was proposed to improve the effect while maintaining its on-linear property, e.g. the Leaky ReLU approach preserves the negative value, and the ReLU6 approach prunes all large values. Fig 2.3 illustrates these different ReLU functions.

Downsampling Layer. In the architecture of the neural network, a downsampling layer is used to reduce the size of feature maps while simultaneously keeping the important features. Since the learnable kernels consume a lot of memory storage and computation power, it is easily running out of those resources when the network depth increases. Vari-

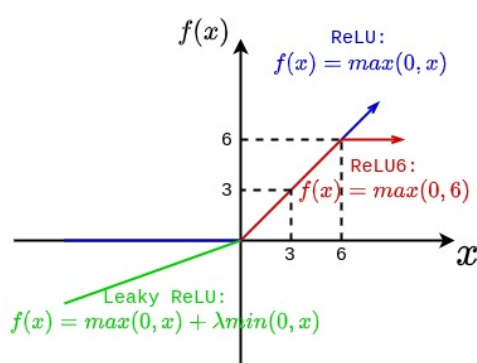


Figure 2.3: The functionality of ReLU, Leaky ReLU, and ReLU6. For the Leaky ReLU approach, λ is a hyperparameter to control the slope. These functions increase nonlinearity of CNNs.

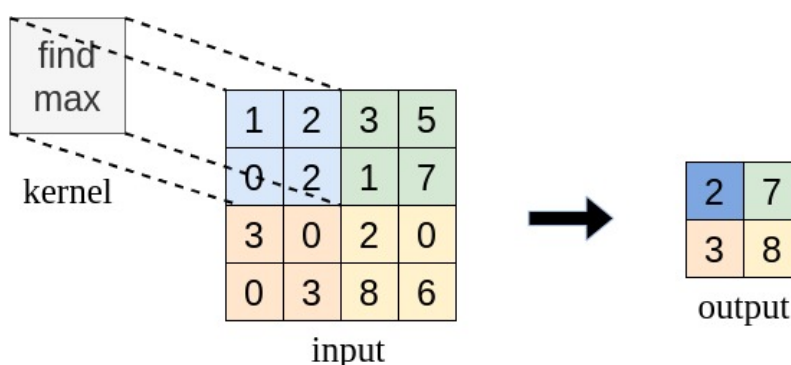


Figure 2.4: Calculation of max-pooling.

ous pooling operations have been proposed to downsample the feature maps which reduce the cost of computer resources. The most well-known one is max-pooling, which uses a parameter-free kernel to match a patch and extracts the maximum value in the pooling region as its output features. This design has no trainable parameters and only retains the strongest features and ignores others. The calculation process of max-pooling is shown in Fig 2.4.

In contrast to the parameter-free pooling operation, there's also an alternative called stride convolution which has learnable parameters to perform downsampling. Considering the standard convolution operation, both the convolution operation and the pooling operation have their own kernels to calculate the outputs and use a sliding strategy to loop over the whole picture. The only difference compared to max-pooling is their sliding strides.

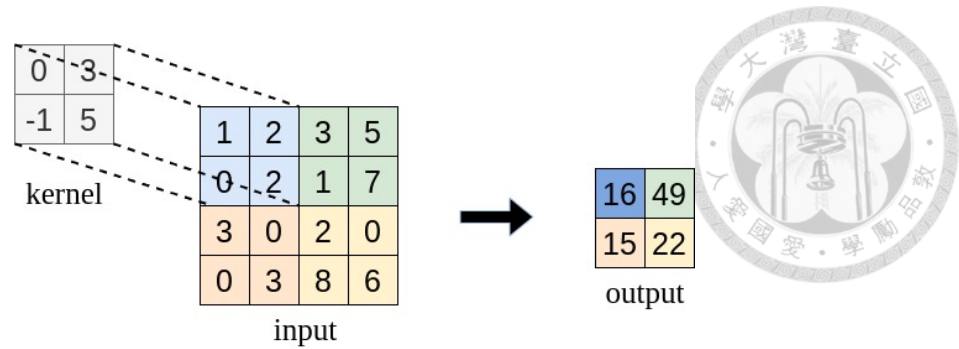


Figure 2.5: Calculation of strided convolution. In this case, the kernel size and the stride are 2.

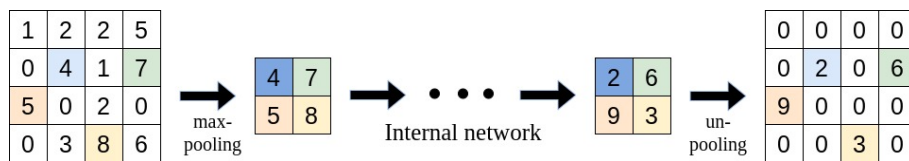


Figure 2.6: Calculation of unpooling convolution.

In the pooling operation, the stride is the same as the kernel size, causing each patch to be non-overlapping. Following this approach, we can modify the standard convolution to create the stride convolution. Fig 2.5 shows an example of the convolution process. Since this operation has trainable parameters, its learning ability is stronger compared to the pooling approach.

Upsampling Layer. After downsampling the data and applying a series of operations, some kinds of CNNs require to reconstruct and upsample the data to restore its original size. Therefore, we use an unpooling layer, which is the counter-part to the max-pooling operation and enlarges the feature maps. Normally, the table records the locations of the maximum value during the max-pooling operation, and then sets the values to those positions accordingly. The process of unpooling layer is shown in Fig 2.6.

Identical to the previous section, the unpooling layer also has a learnable version called transposed convolution layer, which is the reverse process of convolution. The layer swaps the input size with the output size and then performs the same operations as a standard convolution. In this way, the mapping relation between input map and output

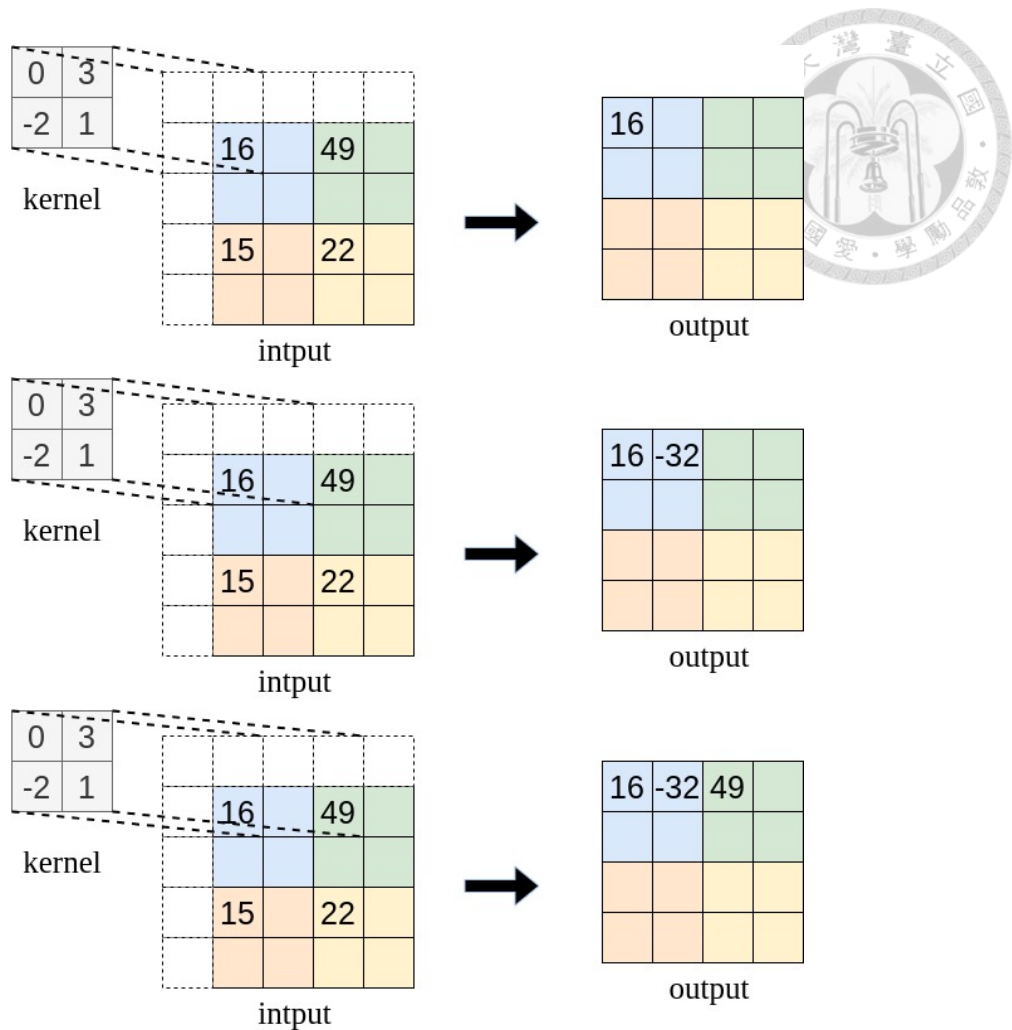


Figure 2.7: Illustration of transposed convolution operation. It is the inverse operation of strided convolution shown in Fig 2.5. Notice that the mapping relation remains the same, that is, only the regions with same colors affect each others.

map remains the same. A transposed convolution can learn the spatial information better than a conventional normal unpooling operation. Fig 2.7 shows the idea.

Normalization Layer. During the training process, CNNs easily suffer from an internal covariate shift, a phenomenon that occurs when the gradients of features are expressed in different scales, making the model training very inefficient. Fig 2.8 demonstrates this phenomenon. Therefore, many normalizing methods are applied to fit the features into a fixed range which avoids internal covariate shift. Exemplary methods explained in Fig 2.9 are Batch Norm, Layer Norm, Instance Norm, and Group Norm. With this technique, the training becomes more stable, allowing to set a higher learning rate to

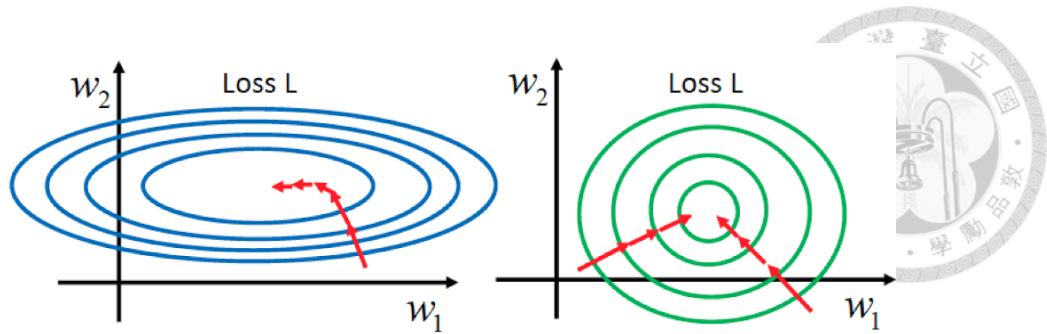


Figure 2.8: Internal covariate shift phenomenon. In left picture, without normalization, the gradient of w_1 is much smaller than w_2 , so the model needs more steps to converge; in the right picture, with normalization, the scaling of w_1 and w_2 are balanced so that the model converge quickly.

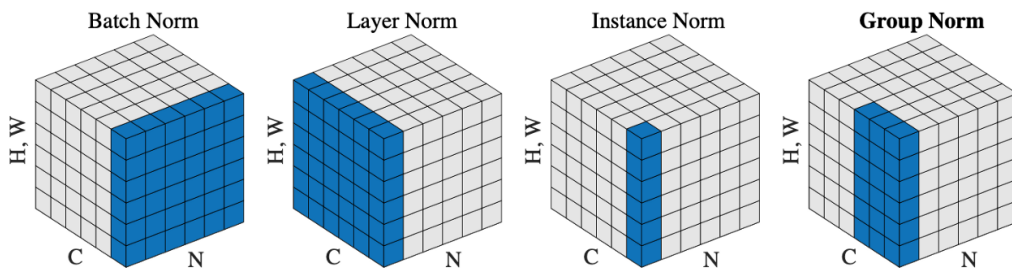


Figure 2.9: Four different normalization skills. N is the batch axis. H, W are the width and height in the spatial axes. C is the channel axis. The data in blue cubes are normalized by their mean and variance.

accelerate the training process and also achieve higher accuracy.

Loss Function Layer. Loss functions have a key role in modern deep learning methods. After we get the model output, we need to judge whether the result is good or not and calculate the gradient in order to update the model parameters. Loss functions are designed to evaluate the model output performance. An appropriate loss function should reflect the goodness of the task efficiently while providing stable gradients. There are several loss functions designed for tackling different problems, such as the mean-squared error for solving the regression problem and the cross-entropy for solving the classification problem which are defined in Equation 2.3 and Equation 2.4.

$$L(y, y^*) = \frac{1}{n} \sum_{i=1}^n (y_i - y_i^*)^2 \quad (2.3)$$

where y, y^* are the prediction and the ground-truth of the labels, and n is the number of data.



$$L(p, q) = - \sum_{x \in X} p(x) \log q(x) \quad (2.4)$$

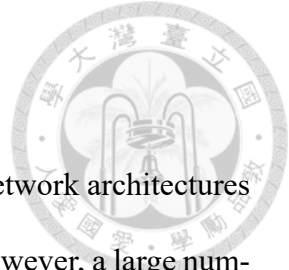
where p, q are the prediction and the ground-truth probability of the class labels and X is the set of the classes.

Gradient Descent Optimization Since CNNs have a large number of parameters, the strategy to find optimal parameters is very important. After we calculate the loss, the next step is to update the parameters of each layer via backpropagation according to the gradient provided from the loss. The process is called gradient descent optimization. In this stage, we use an optimizer to decide the learning rate of each step. A common optimizer is the stochastic gradient descent (SGD) method, which replaces the actual gradient calculated from the entire data set by an approximation from the sample batch data. Additionally, the momentum is introduced to control the learning rate. We derive the SGD Momentum as follows:

$$V_t = \beta V_{t-1} - \eta g_{t-1} \quad (2.5)$$

$$W_t = W_{t-1} + V_t \quad (2.6)$$

Where g^{t-1} is the gradient of last time step $t-1$, η is the learning rate hyperparameter for a single step, V_{t-1} is the velocity of last time step $t-1$, and β is the momentum to control the change of velocity.



2.1.2 ResNet

In recent years, neural networks are using increasingly deeper network architectures because deeper models have better feature expression capabilities. However, a large number of experiments have shown that blindly deepening the model will lead to the degradation problem, indicating that a deep model is not easy to optimize. To address the above problem, K. He *et al.* [34] proposed the concept of a deep residual learning framework, which uses a shortcut connection to provide plain identity mapping for different layers of the network. He claims that it is easier to optimize the residual mapping than to optimize the original unreferenced mapping. A building residual block is defined as follows:

$$y = F(x, \{W_i\}) + x \quad (2.7)$$

where x and y are the input and output feature vectors of the layer. $F(x, \{W_i\})$ represents the residual mapping layer i to be learned. The output of layer F is then added element-wise with a shortcut connection to obtain y . With this identical mapping, a deeper model can exhibit a higher training error than its shallower counterpart and gain better performance. Also, This operation is very effective without extra parameters and complex computation. The basic residual block is composed of two convolution layers followed by the ReLU, which is shown in Fig 2.10. With the help of a residual block, we can easily build and optimize a deep neural network.

2.1.3 3D convolution

Inspired by the success of 2D convolution, many pioneering researches wanted to solve the higher dimension problem using the same idea. They expanded the 2D image

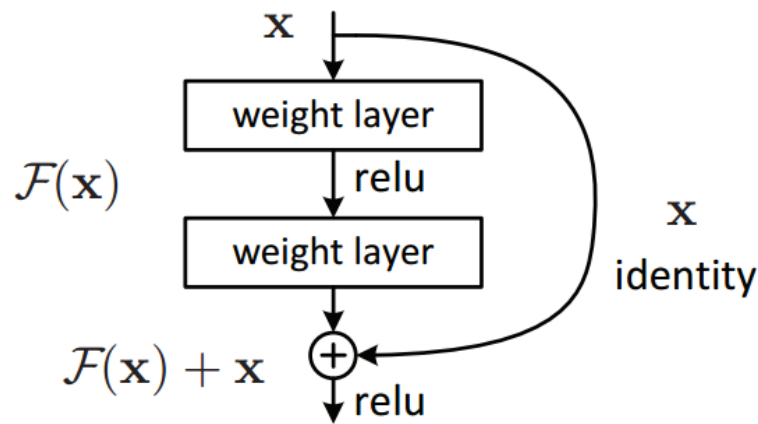


Figure 2.10: Basic residual block [34]

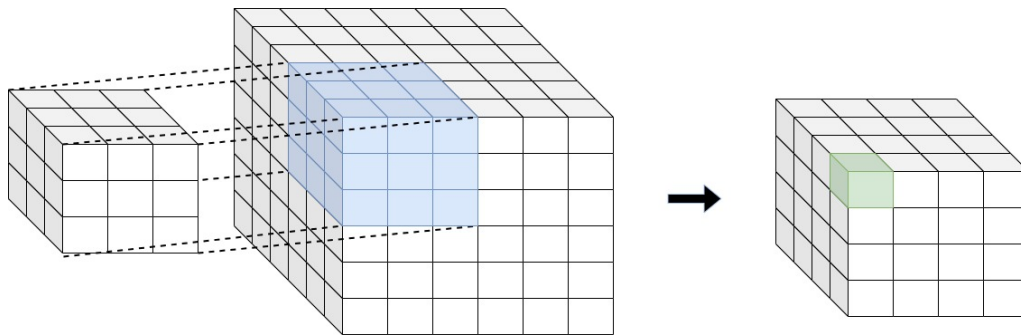
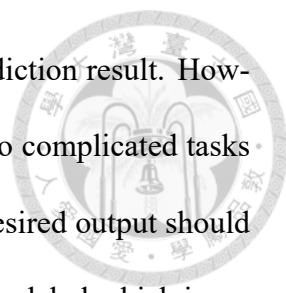


Figure 2.11: Illustration of 3D convolution operation

problem to the 3D space problem and tried to convert the 2D convolution neural network into a 3D convolution neural network (3D CNN). The basic idea of the 3D CNN is similar to the 2D CNN. It also defines learnable kernels, a convolution operation, and a sliding strategy to compute the output, while operating in 3-dimensions. Fig 2.11 shows the basic concept of a 3D convolution. We can easily confirm the similarity compared to the 2D version. Many papers have shown that the 3D convolution is able to process 3D spatial features for many 3D vision tasks very well, just like 2D vision tasks.

2.2 Semantic Segmentation Framework

Since the first time convolutional neural networks demonstrate their abilities, people have been working on simple regression and classification tasks. For these tasks, the



output of a neural network from an image only contains a single prediction result. However, these kinds of networks have been unsuccessful when applied to complicated tasks that need highly linear mapping. In these kinds of visual tasks, the desired output should include the localization along with classification, i.e., including a class label which is assigned to each pixel. Thus researchers came up with the idea of generating an output that is the same size as the input. This task is called semantic segmentation. The first experiment using deep learning on this task is proposed by [36], who trained a network for one local patch to predict a fixed pixel output and used a sliding-window to iterate through the whole image. However, the drawback of this approach is that the network has too many overlapping patches, making the inference slow. Many amazing architectures are proposed in recent years to improve the performance, including the FCN [8] and U-Net [9] architectures.

2.2.1 FCN

In most of the previous methods, neural networks always use fully connected layers for the last few layers, which throw away spatial coordinates and restrict the network to only allow fixed dimensions. A fully convolutional neural network [8] is the first deep learning architecture that takes an input of arbitrary size and produces a correspondingly-sized output for semantic segmentation. The key idea of FCNs is to replace the pooling layer with the upsampling layer to jointly increase the resolution from latent low-resolution features. It enables pixels-to-pixels semantic segmentation and makes the input size more flexible. The upsampling layer uses a new operation called deconvolution to achieve the purpose, which behaves like the reversion of a normal stride convolution. Given a convoluted feature map, the corresponding deconvolution will generate the feature map with the same size as the unconvoluted feature map. Since the deconvolution is also differen-

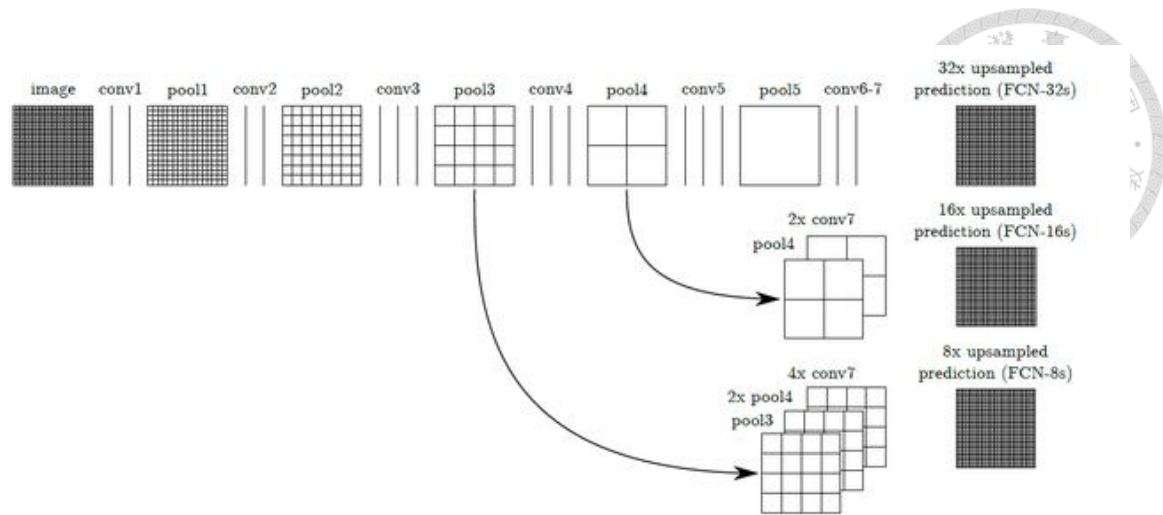


Figure 2.12: Fully convolutional network architecture [8]

table, the whole process can be trained end-to-end. Referring to Fig 2.12, FCN includes a skip architecture to combine layers of the feature hierarchy and refine the spatial precision of the output. In this way, the model can fuse coarse semantic information and fine appearance one, and generate different stride predictions.

2.2.2 U-Net

Similar to FCN [8], U-Net [9] also adopts a fully convolutional neural network to semantically segment the image. The whole architecture can be viewed as a combination of an encoder and a decoder. The contracting encoder path extracts high dimensional visual features with a larger receptive field while the decoder path is responsible for decoding those features to generate the pixel-wise prediction output. In FCN, the decoder uses a large stride deconvolution with a few feature channels to directly expand the resolution to the original image, making it inaccurate for localization. In order to generate a more precise localization, U-Net makes modifications on the expanding path that adopts many small stride deconvolution layers with a large number of feature channels to gradually propagate context information to higher resolution layers. As a result, the expansive decoder path is symmetric to the contracting encoder path, and yields a u-shaped archi-

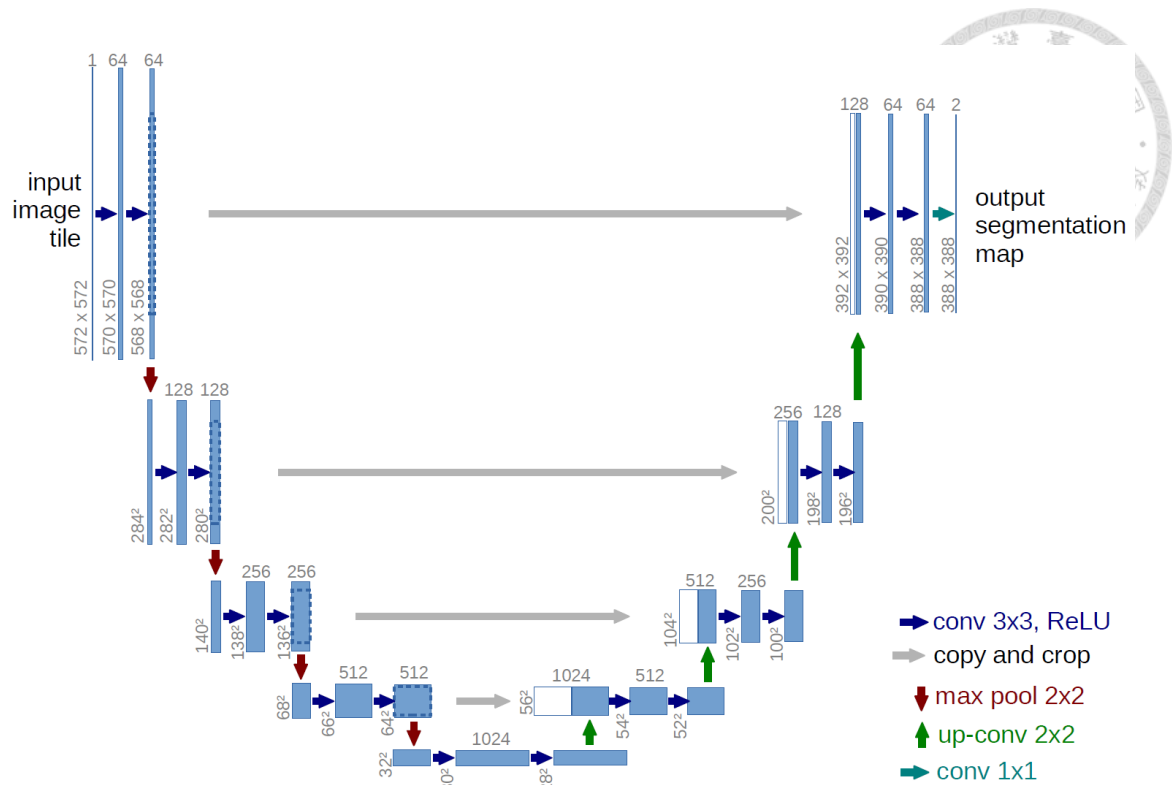


Figure 2.13: The architecture of U-Net [9]

ture, as shown in Fig 2.13. On the other hand, a skip connection is used to combine features from the contracting path with the upsampled output to generate more representative features. Due to its simplicity and effectiveness, this symmetric architecture with skip connection is becoming a standard design pattern in the field of semantic segmentation.

2.3 Sparse Convolutional Neural Networks

Although 3D CNN has been successfully applied to many kinds of issues in the area of computer vision such as 3D object recognition or 2D image recognition with time information, there are two critical problems restricting the 3D CNNs to only small scenes. The first being the unaffordable memory consumption of storing data. Compared to 2D approaches, the space to store data densely in an array grows by x^3 , making it easy to run out of memory. Moreover, the dense representation is inefficient when used on sparse data because most of the space is empty. The second problem is the high computational

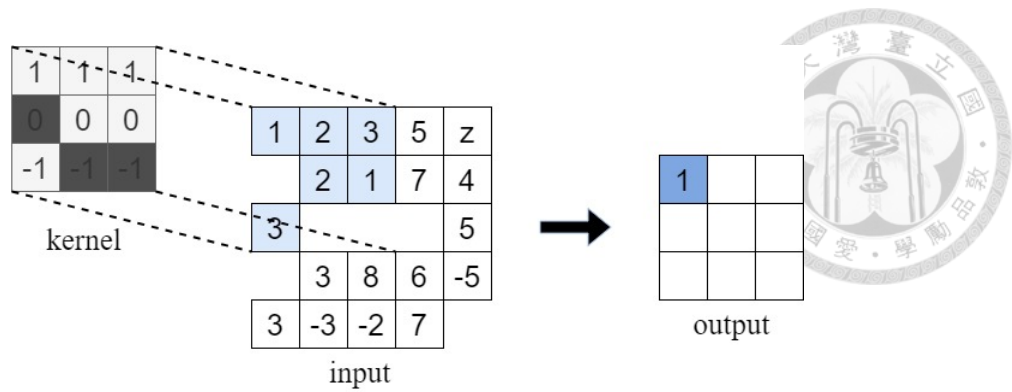
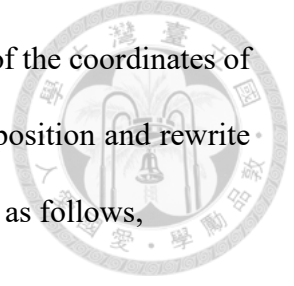


Figure 2.14: Illustration of a sparse convolution operation. In sparse data, some space is empty, thus the computation on those grid squares can be avoided. The gray block in the kernel represents the avoided computing weights.

cost of performing convolutions. A traditional convolution first defines the kernel with a certain shape, i.e., $3 \times 3 \times 3$ or $5 \times 5 \times 5$, and then slides the kernel through the whole space. Due to the nature of sparse data, some sub-space of the kernel inevitably covers the empty space, causing a redundant computation.

Based on the observation of the above problems, sparse convolution proposed by [20] use two new strategies. First, in order to reduce the memory consumption when extending the low-dimension convolution to the high-dimension convolution, we adopt a sparse representation instead of a dense representation which only stores non-empty space. Second, to avoid executing convolutions on empty locations, we treat the kernel from another point of view. The kernel of conventional convolutions can be viewed as a set of sub-kernels, in which each sub-kernel only takes one volume and performs a feature transformation on the specific position of the receptive field and then adds it to the output position. Therefore, the core idea of sparse convolution is to build in-out kernel maps for each sub-kernel in order to record which outputs are influenced by which inputs in the whole space. These mappings only record the pairs in which both input space and output space are valid. In this way, we can save a lot of computation steps. Fig 2.14 illustrates the concept of sparse convolution using a 2D convolution operation from Fig 2.2 as example.



Recalling the Eq. (2.1) we introduced in Sec. 2.1.1, since some of the coordinates of $x_{\mathbf{c}}^{out}$ and $x_{\mathbf{c}+\mathbf{i}}^{in}$ are empty in sparse data, we avoid processing on this position and rewrite the equation. The convolution operation for sparse version is defined as follows,

$$x_{\mathbf{c}}^{out} = \sum_{\mathbf{i} \in V^D(K) \cap \{\mathbf{i} | \mathbf{c} + \mathbf{i} \in \mathbf{C}^{in}\}} W_{\mathbf{i}} x_{\mathbf{c}+\mathbf{i}}^{in} \text{ for } \mathbf{c} \in \mathbf{C}^{out} \quad (2.8)$$

where \mathbf{C}^{out} and \mathbf{C}^{in} are the non-empty input coordinate set and output coordinate set. In following subchapters, we will derive the algorithm of sparse convolution step by step.

2.3.1 Sparse Tensor Representation

To build the whole sparse convolution pipeline, we first introduce the sparse representation. Instead of storing the data densely, we only store the sparse data which has N points as a D -dimensional coordinate set \mathbf{C} and F^{in} -dimensional associated features X . Although we can set the dimension as any arbitrary number, here, we take a 3D point cloud as an example in order to match our cases in practice with the data structure. A complete sparse tensor of the 3D point cloud can be stored in two 2-dimensional arrays as shown below:

$$\mathbf{C}_{arr} = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_N \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}, X_{arr} = \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_N \end{bmatrix} = \begin{bmatrix} f_1^1 & f_1^2 & \dots & f_1^A \\ \vdots & \vdots & \ddots & \vdots \\ f_N^1 & f_N^2 & \dots & f_N^A \end{bmatrix} \quad (2.9)$$

where \mathbf{C}_{arr} and X_{arr} are the hash tables that use the point coordinate as a hash key to store the information of \mathbf{C} and X , and $\mathbf{c}_j = (x_j, y_j, z_j)$ indicates the x, y, z coordinate of j -th element of \mathbf{C}_{arr} and $\mathbf{f}_j = (f_j^1, f_j^2, \dots, f_j^A)$ is the j -th feature vector of X_{arr} . This kind of

presentation only stores the non-empty points with their information, so it is very memory efficient.

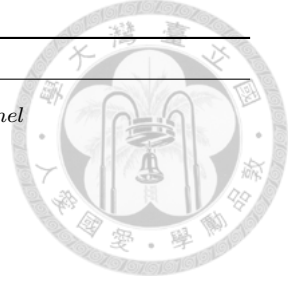


2.3.2 Spatial Hashing

Neighbor searching is a very important operation for both the aggregation stage and down-sampling stage. Compared to point-based methods whose coordinates are unstructured and unaligned, voxel-based methods apply a quantization process to get structured voxels, thus we can easily query the feature at a specific position using its coordinates. In dense representation, the x , y , z coordinates represent just the index of the 3D array. Since sparse data is stored continuously and unorderedly, we convert the coordinates into hash keys and build a hash map to store indices of sparse data. With the help of a hash table, we can easily query and confirm whether the specific position is valid or not with $O(1)$ time complexity.

2.3.3 Output Coordinate Generation

The first step of the sparse convolution pipeline is to generate the output coordinate set \mathbf{C}^{out} based on the input coordinate set \mathbf{C}^{in} . As **Algorithm 1** shows, the algorithm takes input coordinates, input coordinate stride (the minimum distance between coordinates), and kernel stride as input parameters to generate the new coordinates. For each input coordinate \mathbf{c} , we compute the aligned coordinate based on the new output stride s_{out} and add it to the output coordinates. In the case of a normal convolution, the kernel stride is equal to 1, therefore the spatial output size will remain unchanged, i.e., \mathbf{C}^{out} is the same as \mathbf{C}^{in} . Only for the stride convolution used in the down-sampling stage, the algorithm needs to calculate the output coordinates. The number of coordinates will gradually decrease with a larger coordinate stride (larger voxel). For transposed convolutions used in the



Algorithm 1 Output Coordinate Generation

Require: input coordinate set \mathbf{C}^{in} , input stride s_{in} , kernel stride s_{kernel}

```

1: if  $s_c > 1$  then
2:    $s_{out} \leftarrow s_{in} \times s_{kernel}$ 
3:    $\mathbf{C}^{out} = \{\}$ 
4:   for all  $\mathbf{c} \in \mathbf{C}^{in}$  do
5:      $\mathbf{c} \leftarrow \lfloor \mathbf{c} / s_{out} \rfloor \times s_{out}$ 
6:     if  $\mathbf{c} \notin \mathbf{C}^{out}$  then
7:       Add  $\mathbf{c}$  to  $\mathbf{C}^{out}$ 
8:     end if
9:   end for
10:  return  $\mathbf{C}^{out}$ 
11: else
12:  return  $\mathbf{C}^{in}$ 
13: end if

```

up-sampling stage, the output coordinates that are already computed during the down-sampling stage will be re-used to ensure the same order of the coordinates is maintained.

2.3.4 In-out Kernel Mapping

After getting \mathbf{C}^{out} which has N^{out} point coordinates, the next step is building the kernel mapping to decide which inputs affect which outputs. Let K^D be the kernel volume of the kernel whose length is K in each dimension, the mapping can be defined as K^D pairs of lists consisting of input indices and output indices $\mathbf{M} = \{(\mathbf{I}_i, \mathbf{O}_i)\}_i$ for each $\mathbf{i} \in V^D(K)$ as shown in **Algorithm 2**. $V^D(K)$ is the kernel offset list that records the offset of all sub-kernels in the receptive field of convolutional kernel centered at the origin. For each sub-kernel, we iterate through all the input coordinates, and add the offset \mathbf{i} to it, and then check if the new coordinate also exists in the input coordinate. If it does, that means that both input and output coordinates are valid, thus the sub-kernel should be applied to compute the transformed features. Each Entry $(\mathbf{I}_i^j, \mathbf{O}_i^j)$ in \mathbf{M}_i indicates that the feature of the point located in the \mathbf{I}_i^j coordinate should be multiplied by the sub-kernel with offset \mathbf{i} centered at the origin and then added to the feature of the point located at the \mathbf{O}_i^j coordinate.



Algorithm 2 Kernel Mapping

Require: input coordinate set \mathbf{C}^{in} , output coordinate set \mathbf{C}^{out} , kernel offset list $V^D(K)$

- 1: $\mathbf{M} \leftarrow \mathbf{empty}$
- 2: **for all** $\mathbf{i} \in V^D(K)$ **do**
- 3: **for all** $\mathbf{c} \in \mathbf{C}^{out}$ **do**
- 4: **if** $\mathbf{c} + \mathbf{i} \in \mathbf{C}^{in}$ **then**
- 5: Append $(\mathbf{c} + \mathbf{i}, \mathbf{c})$ to \mathbf{M}_i // (I,O)
- 6: **end if**
- 7: **end for**
- 8: **end for**
- 9: **return** \mathbf{M}

Algorithm 3 Spatially Sparse Convolution

Require: Kernel weights W , input features X^{in} , output feature placeholder X^{out} , convolution mapping \mathbf{M} , kernel offset list $V^D(K)$

- 1: $X^o \leftarrow \mathbf{0}$
- 2: **for all** $\mathbf{i} \in V^D(K)$ **do**
- 3: **for all** $(\mathbf{I}_i^j, \mathbf{O}_i^j) \in \mathbf{M}_i$ **do**
- 4: $X_{\mathbf{O}_i^j}^{out} \leftarrow X_{\mathbf{O}_i^j}^{out} + W_i * X_{\mathbf{I}_i^j}^{in}$ // add bias optionally
- 5: **end for**
- 6: **end for**

2.3.5 Sparse Convolution

With the kernel mapping, we can finally calculate the sparse convolution. With $W \in \mathbb{R}^{K^D \times F_{in} \times F_{out}}$ representing the kernel weights, where the F_{in} defines the number of input feature channel and F_{out} the number of output feature channel. The decomposed W_i , where $\mathbf{i} \in V^D(K)$ is the specific slice of W which is used to perform the feature transformation at the \mathbf{i} offset of the kernel reception field. We compute the the sparse convolution as **Algorithm 3**, where $X_{\mathbf{I}_i^j}^{in}$ and $X_{\mathbf{O}_i^j}^{out}$ represent the associated input feature of the point located at \mathbf{I}_i^j and output feature of the point located at \mathbf{O}_i^j . With sparse convolution, we can efficiently utilize the computation and storage resources.



Chapter 3 Methodology

In this chapter, we propose a novel semantic segmentation network for 3D indoor scenes called SegCompleteNet, which includes the proposed spatial-aware convolutional layer and refinement module to help segment the scene. We will first give a brief overview of our system. Next, we are going to elaborate on the proposed modules of our network architecture. The last part describes the implementation details, revealing how we actually build the system and explaining our design choices.

3.1 Model Overview

The main idea behind our system architecture is to segment the scene first and then refine the prediction results. Our SegCompleteNet consists of the corresponding two networks, Segmentation Net and Completion Net, to achieve these goals. Inspired by U-Net [9] and ResNet [34], we designed our Segmentation Net as a 7-hierarchical encoder-decoder architecture. We adopt a novel Spatial-Aware convolution layer to replace the conventional convolution. The spatial-aware convolution layer is composed of a spatial encoding module and self-attention aggregation module to better capture the 3D spatial relationships between points. Given the input point cloud, the segmentation net will process these point clouds and obtain an intermediate semantic result for each point. After that, we dissociate the points of different prediction classes and then use the completion net to

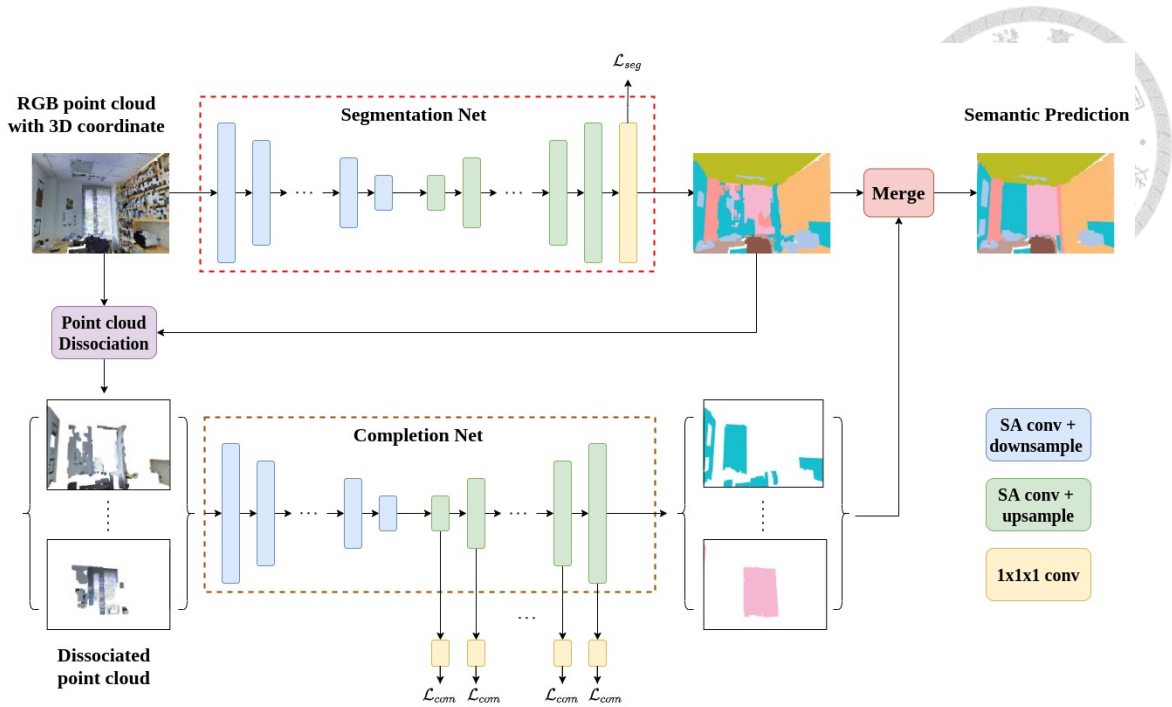


Figure 3.1: Overview of proposed SegCompleteNet.

further refine the points belonging to each class. As depicted in Fig 3.1, the completion net is very similar to the segmentation net. It is responsible for adding or deleting points belonging to a specific category. The final semantic segmentation result is generated by combining the intermediate results and the refinement results.

3.2 Spatial-aware Convolution Layer

In a traditional 3D convolutional layer operation, the sparsity of the data is often not considered. Feature transformation will be applied to the whole grid regardless of whether the space is occupied or not. For an empty grid with no points, the most common practice is to assume that the feature in the empty space is zero. Although the proposal of sparse convolution [20, 19] obviously considers the sparsity of point clouds, it only uses this property to reduce the redundancy of computation on the empty space, and still treats the features of empty space as zero just like other methods.

We argue that sparsity itself will generate the structural information of the point cloud,

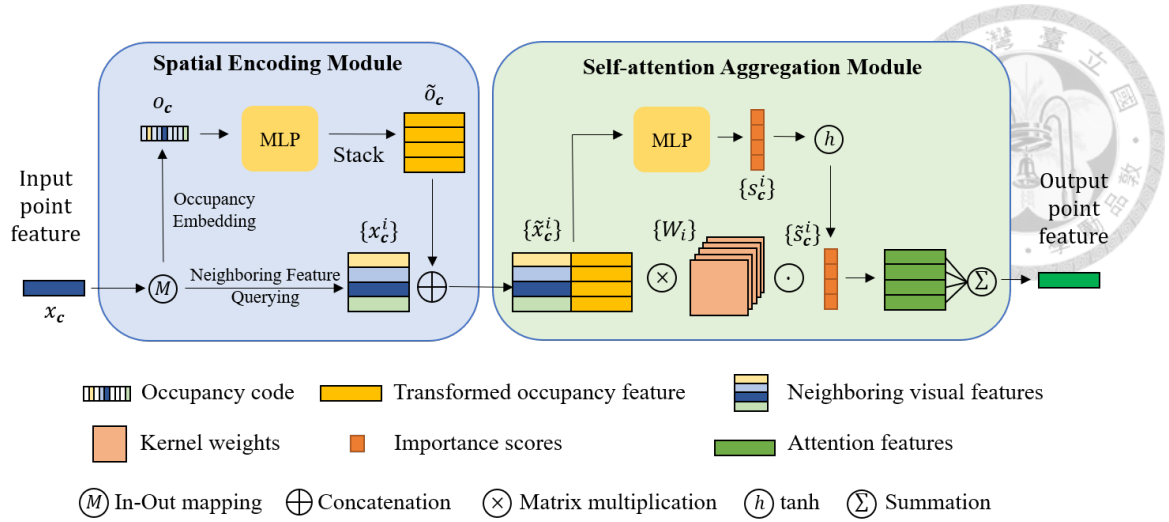


Figure 3.2: Illustration of spatial-aware convolution.

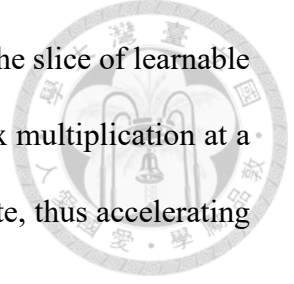
which should be regarded as a kind of feature that helps the model to improve performance. Spatial-aware convolution aims to expand upon sparse convolution by explicitly encoding spatial features. As shown in Fig 3.2, a spatial-aware convolutional layer is composed of a spatial encoding module and a self-attention aggregation module. The former calculates enhanced features with spatial information based on neighborhood relations, and the latter uses an additional weighting strategy to perform a feature aggregation process in the convolution kernels.

In order to facilitate the following explanation, let us review the conventional sparse convolution on 3D space. We consider the convolution with stride 1, whose input coordinate and output coordinate are the same, for the explanation. Given an input point feature $x_{\mathbf{c}} \in \mathbb{R}^{F_{in}}$ with F_{in} channels located in a 3-dimensional coordinate $\mathbf{c} \in \mathbf{C}$, where \mathbf{C} is the input coordinate set, the output point feature of sparse convolution is defined as follows:

$$y_{\mathbf{c}} = \sum_{\mathbf{i} \in V^3(K) \cap \{\mathbf{i} | \mathbf{c} + \mathbf{i} \in \mathbf{C}\}} W_{\mathbf{i}} \cdot x_{\mathbf{c} + \mathbf{i}} \quad (3.1)$$

where $V^3(K)$ is the list of offsets in a 3D cube with kernel size K centered at the origin and \mathbf{i} is the offset element of $V^3(K)$. For example, $V^3(3)$ contains 27 elements and the

corresponding \mathbf{i} is from $(-1, -1, -1)$ to $(1, 1, 1)$. $W_{\mathbf{i}} \in \mathbb{R}^{F_{in} \times F_{out}}$ is the slice of learnable kernel weights of offset \mathbf{i} , which is responsible for performing matrix multiplication at a specific grid. The computation is only performed at a valid coordinate, thus accelerating the computation speed.



3.2.1 Spatial Occupancy Encoding

Inspired by RandLA [23], our spatial encoding module aims at explicitly encoding spatial information as an additional feature for sparse convolution. In point-based [23], it uses 3d point coordinates and the relative distance between points as extra features. However, this strategy can not be applied to our voxel-based approach because of the fact that structured voxel data has a fixed grid position and a fixed distance between neighboring grids. In order to capture the spatial architecture of these structured data, we consider using a one-hot vector called occupancy embedding to indicate whether the surrounding grid is occupied or not.

We encode the occupancy embedding based on the in-out kernel map already built in the conventional sparse convolution operation. The in-out kernel map is the core for sparse convolution that records the coordinate relationship needed for Eq. (3.1). We denote the in-out kernel map as pairs of lists of input coordinates and output coordinates, $\mathbf{M} = \{(\mathbf{I}_i, \mathbf{O}_i)\}_i$ for $\mathbf{i} \in V^3(K)$. Each corresponding entry in \mathbf{I}_i and \mathbf{O}_i indicates that the grid at an input coordinate is the neighbor of the grid at an output coordinate with offset \mathbf{i} . This kernel mapping provides spatial evidence of the surrounding structure. In practice, the kernel maps for all offsets are stored in a 1D list. We define i as the index of kernel map list with specific topological order in $V^3(K)$ to distinguish it from bold notation \mathbf{i} offset.

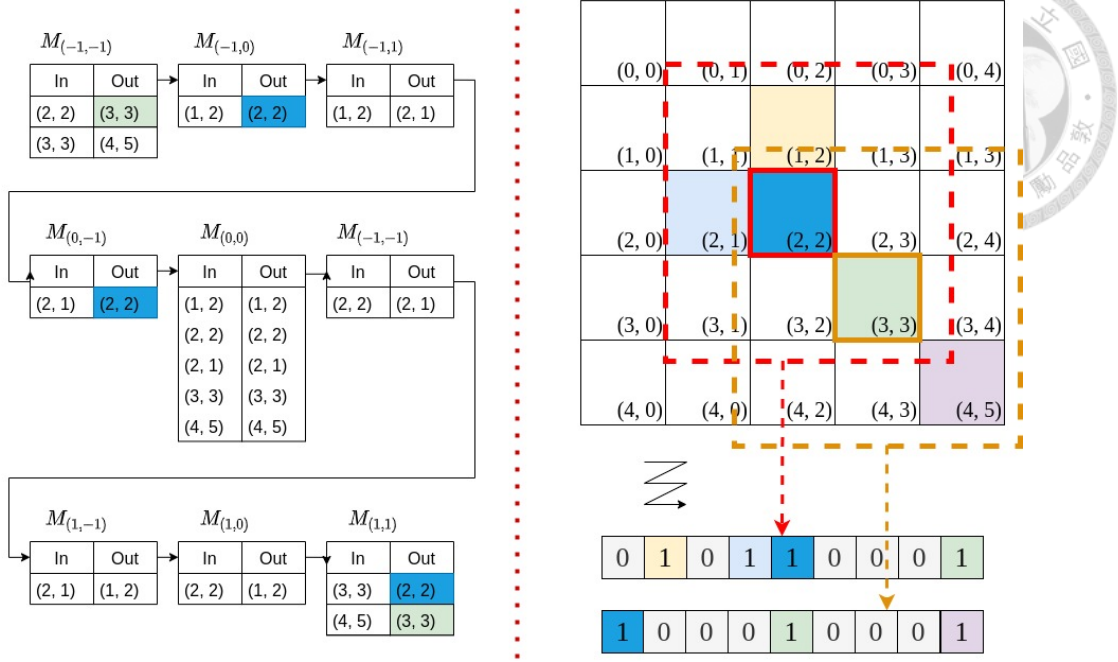


Figure 3.3: Occupancy Embedding. (a) In-out kernel map stored in memory. (b) A example of simple point set in the space.

We calculate an associated occupancy code $o_{\mathbf{c}}$ located at \mathbf{c} as follows:

$$o_{\mathbf{c}}[i] = \begin{cases} 1 & \text{if } \mathbf{c} \in \mathbf{O}_i \\ 0 & \text{otherwise} \end{cases} \quad (3.2)$$

where $o_{\mathbf{c}}[i]$ is the i -th element of $o_{\mathbf{c}}$, indicating whether i -th neighboring grid of \mathbf{c} is occupied or not. It is set to 1 if the grid is valid, and 0 otherwise. Fig 3.3 illustrates the generation of occupancy embedding. Notice that for dense data, it is unreasonable to encode spatial information as extra features. Since every space is occupied, the occupancy embedding will always be the same.

To increase the expressiveness of this code, we further use an MLP layer followed by ReLU, $g(\cdot)$, to transform the spatial features and increase the nonlinearity. The transformed occupancy embedding $\tilde{o}_{\mathbf{c}}$ is computed as:

$$\tilde{o}_{\mathbf{c}} = g(o_{\mathbf{c}}, \theta_o) \quad (3.3)$$

where θ_o is the learnable weight of an MLP. The output dimensions are the same as the visual features.

After obtaining the transformed occupancy embedding, the module is going to collect neighboring visual features. The visual features of neighboring grids are denoted as $X_{\mathbf{c}} = \{x_{\mathbf{c}+\mathbf{i}} | \mathbf{i} \in V(K), \mathbf{c} + \mathbf{i} \in \mathbf{C}\}$ which follows the same rule as defined in the Eq. (3.1). It is a set with arbitrary number of elements according to the in-out kernel map. In the rest of this section, we denote $x_{\mathbf{c}}^{\mathbf{i}^{val}}$ as the $x_{\mathbf{c}+\mathbf{i}}$ with the valid $\mathbf{i} \in V^3(K) \cap \{\mathbf{i} | \mathbf{c} + \mathbf{i} \in \mathbf{C}\}$. To obtain the enhanced features, we first stack the transformed occupancy embedding $\tilde{o}_{\mathbf{c}}$ and then concatenate it to $X_{\mathbf{c}}$. Eventually, the output of our spatial encoding module for a given grid at \mathbf{c} is:

$$\tilde{x}_{\mathbf{c}}^{\mathbf{i}^{val}} = x_{\mathbf{c}}^{\mathbf{i}^{val}} \oplus o_{\mathbf{c}} \quad (3.4)$$

where \oplus is the "concatenate" operation. Although the visual features are enhanced by the same occupancy feature, the same occupancy feature will have different calculations since each grid has its own learnable kernel. With the spatial encoding module, we explicitly embed the space structure around the grid.

3.2.2 Self-attention Aggregation

To obtain the aggregated features, the most intuitive method is to use the enhanced features obtained in the spatial coding module and directly perform matrix multiplication with corresponding kernels, and finally sum up all the results. However, as we mentioned in the previous chapter, since the sparse point cloud is structurally descriptive, the same grid should have different effects in different structures.

The self-attention aggregation module aims to aggregate neighboring features based

on different structures. In order to take full advantage of this sparse property, we use an additional branch to calculate the importance score of each grid, so that the network can learn different weights of the same grid in different structures. We take input enhanced features from the spatial encoding module and transform them through a two-layer MLP function, $h(\cdot)$, to obtain a scalar value as shown below:

$$s_{\mathbf{c}}^{\text{i}val} = h(x_{\mathbf{c}}^{\text{i}val}, \theta_s) \quad (3.5)$$

where θ_s is the learnable weight of an MLP, $s_{\mathbf{c}}^{\text{i}val}$ is a scalar representing the importance score of the corresponding grid. The hyperbolic $\tanh(\cdot)$, namely, $\tanh(\cdot)$, is further applied to limit the value range between -1 and 1 to facilitate linear combination. We also add a constant value 1 to stabilize the results. Thereby, the importance score can be viewed as the adjustment to add or subtract the weights in the original aggregation. The transformed importance score is computed as follows:

$$\tilde{s}_{\mathbf{c}}^{\text{i}val} = 1 + \tanh(s_{\mathbf{c}}^{\text{i}val}) \quad (3.6)$$

Finally, we modify the Eq. (3.1) and propose a spatial-aware version as follows:

$$y_{\mathbf{c}} = \sum_{\mathbf{i} \in V^3(K) \cap \{\mathbf{i} | \mathbf{c} + \mathbf{i} \in \mathbf{C}\}} \tilde{s}_{\mathbf{c} + \mathbf{i}} (W_{\mathbf{i}} \cdot \tilde{x}_{\mathbf{c} + \mathbf{i}}) \quad (3.7)$$

To summarize, given an input grid feature $x_{\mathbf{c}}$, our Spatial Encoding module enhances features according to the occupancy embedding, and then uses a self-attention aggregation module to aggregate the patterns at the neighboring grid to generate a final output grid feature $y_{\mathbf{c}} \in \mathbb{R}^{F_{out}}$, which has F_{out} output channels. Our spatial-aware convolution can easily replace the original convolution, which is convenient for direct use on the existing

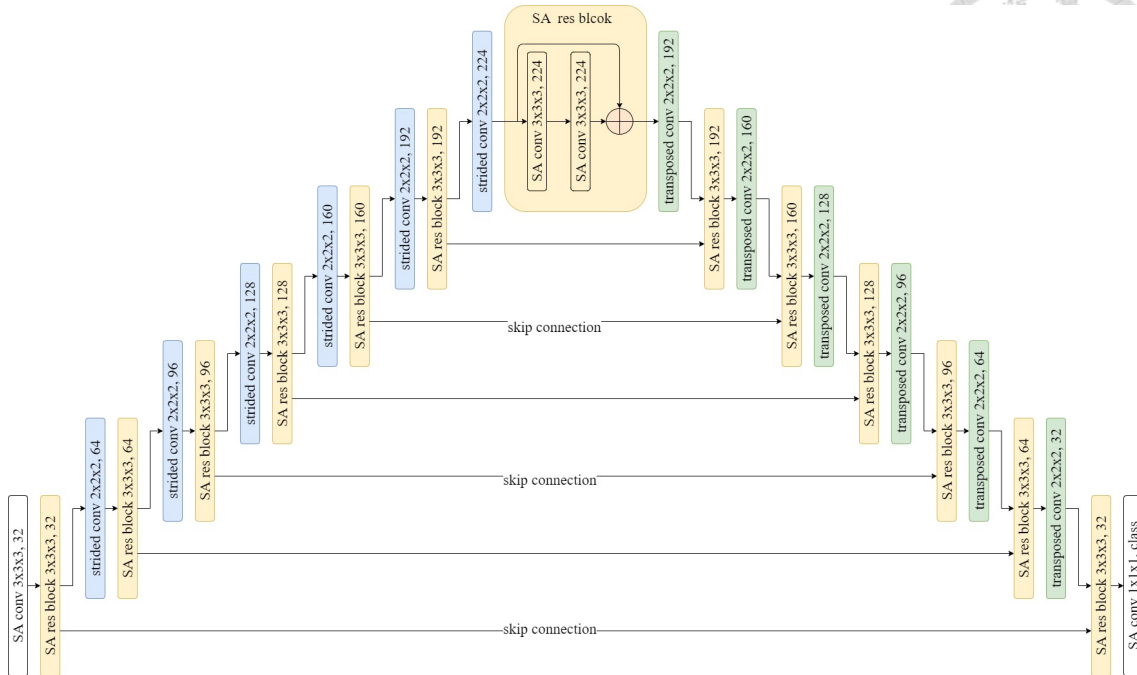


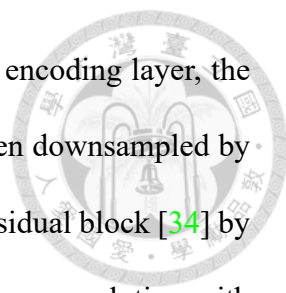
Figure 3.4: Architecture of Segmentation Net.

network architecture.

3.3 Segmentation Net

Segmentation Net is a fully convolutional neural network, which contains a series of convolutional layers to transform features and finally obtain semantic predictions. As shown in Fig 3.4, we build a deep seven-layer network using the proposed spatial-aware sparse convolution. With it, the model can better capture the spatial relationship. Following the idea proposed in [9], our network is composed of seven encoding layers and decoding layers with skip connection to efficiently learn the feature with a larger receptive field. Generally, sparse convolution requires a considerable amount of memory to record in-out kernel maps. Due to this memory consumption, it is not suitable to choose a larger kernel size. Therefore, all the convolution operation in our network only uses $3 \times 3 \times 3$ kernels to avoid depletion of memory.

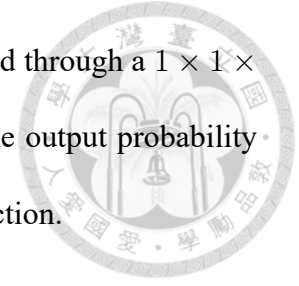
For the encoding part, seven encoding layers are applied to progressively reduce the



size of the point cloud while increasing the feature channel. In each encoding layer, the point features are first fed into a spatial-aware residual block, and then downsampled by the stride convolution. A spatial-aware residual block consists of a residual block [34] by using our spatial-aware convolutions. It consists of two spatial-aware convolution with residual connection (element-wise addition) to carry the identity features. We apply a sparse convolution with the stride of $2 \times 2 \times 2$ to downsample the point set and expand the receptive field. Meanwhile, the feature channel increases to allow higher dimension features to represent the sub-point cloud and preserve information. For layer i , the number of feature channel is $32 \times i$ ($32 \rightarrow 64 \rightarrow 96 \rightarrow 128 \rightarrow 160 \rightarrow 192 \rightarrow 224$). Unlike dense stride convolution, the number of reserved points in sparse data is arbitrary. In our case, it can retain about $1/3$ to $1/4$ of the points from the original point set. With this thin and lightweight design, the encoding layer can quickly and effectively extract features and reduce the number of points.

For the decoding part, seven symmetric decoding layers are used to gradually increase the size of the point cloud while reducing the feature channels, and finally output a prediction. Each decoding layer also contains a spatially-aware residual block identical to the above section. After the transformation, we upsample the point set using a sparse transposed convolution with a stride of $2 \times 2 \times 2$. Sparse transposed convolution utilizes the inverse mapping built into the stride convolution to reconstruct the original point set. Next, the upsampled features are enhanced through skip connections. Instead of simply performing element-wise addition [9], our skip connections concatenate the output features of the i -th encoding layer to the input features of the corresponding decoding layers to serve as the new input features. With skip connections, the model is able to convey information from deep to shallow layers and capture precise localization.

The predicted final semantic probability of P points is generated through a $1 \times 1 \times 1$ convolution with N_{class} channel followed by a softmax layer. The output probability matrix, denoted as $S \in \mathbb{R}^{P \times N_{class}}$, will be used further in the next section.



3.4 Completion Net

After the model predicts the semantic label of each point, Some post-processing methods [21] that can be applied to optimize the result, like correcting the fragments that are incorrectly predicted and improving the fineness of the segmentation boundary. However, most of them are unlearnable. Inspired by the completion task [37, 38] which can take partial objects with noise and output the complete shape, we build a completion net as the refinement module to improve corrections to similar objects in the scene. We take the predicted points of each category from the segmentation net as independent input scenes and feed them into the completion net to refine the prediction. In this way, the prediction that belongs to a specific class can learn more complete and reasonable shapes. Please note that unlike traditional completion tasks, we do not predict each voxel in the grid, but predict the voxel that originally exists in the scene to save computing power and memory consumption through sparse convolution.

To independently process different objects in the scene, we first use a process called dissociation to segment the objects. Dissociation is essentially what the segmentation has already done. In the sparse convolution algorithm, a hash table is used to quickly find any element at a specified position. Every point uses its three-dimensional coordinate as the unique hash key. In order to facilitate batch processing, a dimension is usually added to represent different samples. Therefore, each point has a unique key (b, x, y, z) in the hash table, where b is the batch index and (x, y, z) is the 3D coordinate. In the process of finding

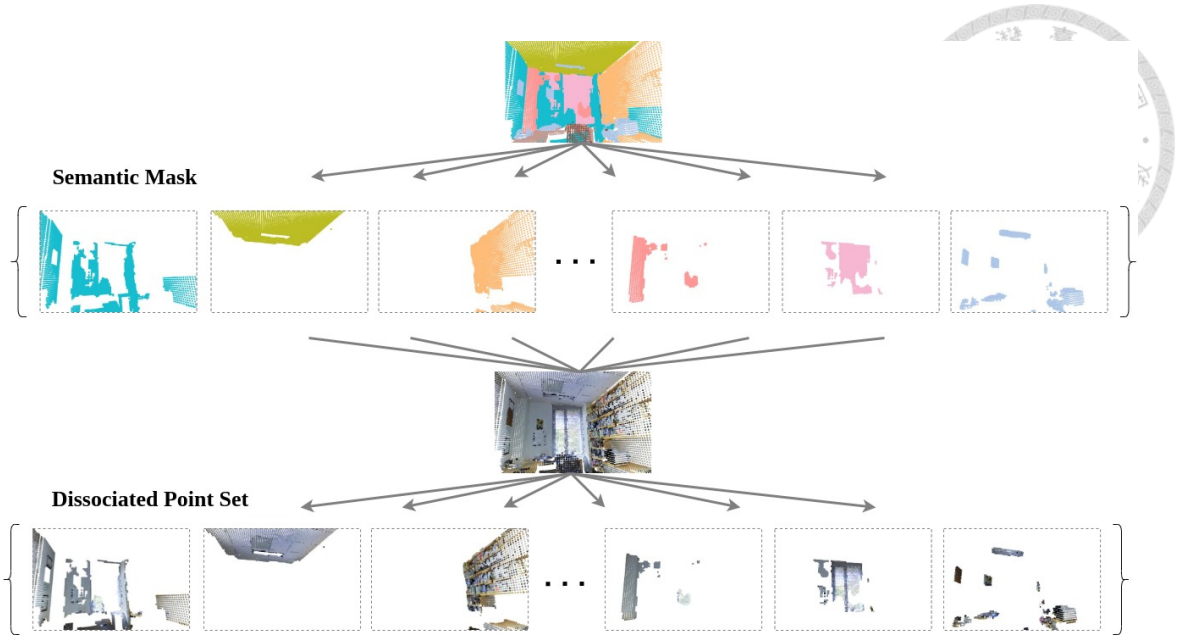
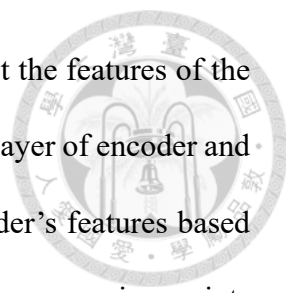


Figure 3.5: Dissociation procedure.

neighbors, different batches are always independent. We make use of this feature to implement dissociation. First, according to the predicted probability matrix $S \in \mathbb{R}^{P \times N_{class}}$ outputted by the segmentation net, we find the index of the maximum value along the column dimension of S to find the most probable category for each point. We then add the class index to the batch dimension to get a new hash key $(b + class, x, y, z)$. Thereby, we can treat the subset of each predicted class as an independent sample to facilitate subsequent operations, as shown in Fig 3.5.

The network architecture of the completion net is very similar to the segmentation net. It adopts a 7-hierarchical encoder-decoder architecture to efficiently extract features and process the point cloud. The difference is that in each decoding layer, we use generative transpose sparse convolution [39] to expand the sub-point cloud of each class. This operation will increase the number of points in each sub-point cloud. Unlike generative sparse network [39] which generates the whole cubic space, we only generate the grid existing in the original point cloud. In the next step, the additional classifier using $1 \times 1 \times 1$ kernels is used to predict the confidence of the existence of each grid in the generated point



set and trim the points accordingly. For skip connections, we connect the features of the encoder to the features of the decoder. Because the point sets in each layer of encoder and decoder are inconsistent, we map the encoder's features to the decoder's features based on their coordinates. Features that are not mapped are set to zero. To summarize, points belonging to a specific class will be continuously added and trimmed in each layer, finally outputting the refined results.

Fig 3.6 illustrates the whole pipeline. Taking a table object as an example, the dark blue part in the figure represents the prediction results from our segmentation net, and the light blue part represents other prediction categories. For the points belonging to the table, we first use a series of convolutional layers to extract the features and then use a generative transpose convolution to generate the corresponding features of every existing scene geometry in the receptive field. After that, we use the classifier to predict the existing confidence scores for all grids and trim them off. Dotted grids represent the space that is generated in a typical generative network. In our work, however, only the grids originally occupied by points are generated, and dotted grids are not generated. The gray grids represent the places that are trimmed. During the decoding process, it can be observed that the network will generate grids on the space of the table foot and table lamp at the same time, but then filter out the table lamp part. After going through a continuous generation and pruning process, the final output results in a more complete table shape.

After we receive each type of completion result, the next step is to re-associate all point sets together. Since the completion net is a generative network, there will be some overlapping points between the final output of different objects. It is necessary to resolve the conflicts of different predictions for identical points. The output of completion is a binary value, which represents the confidence value of the existence of the object at that

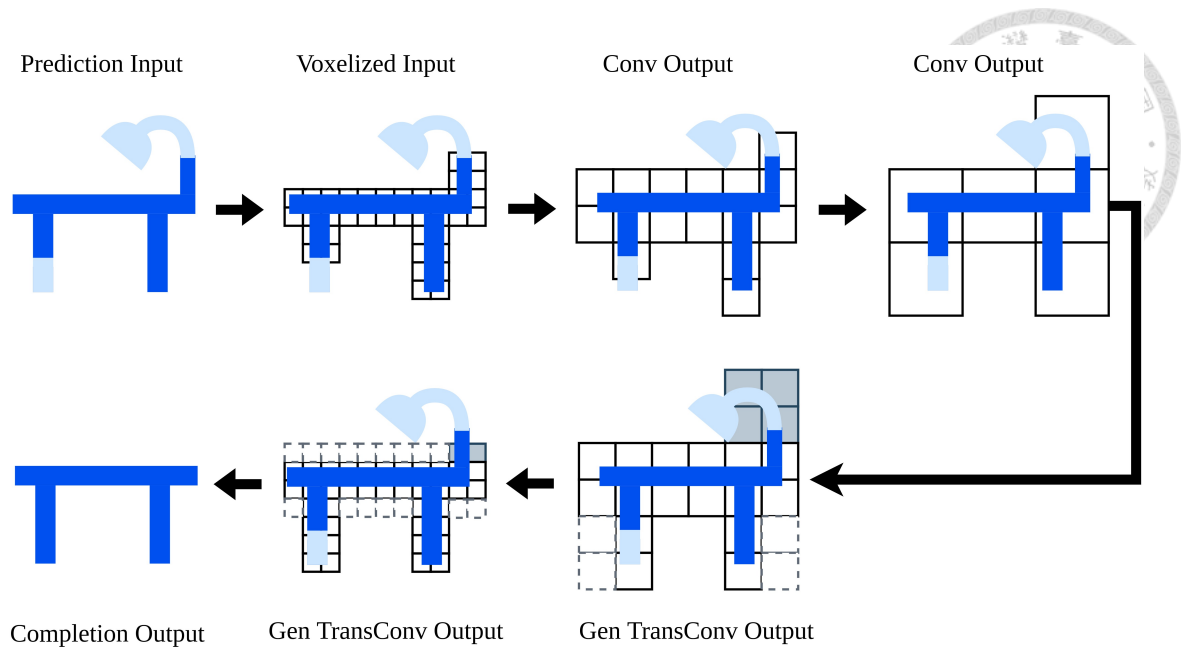
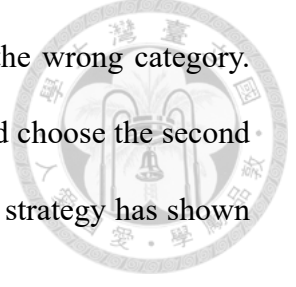


Figure 3.6: Completion procedure.

point. Here, we directly use the matrix $J \in \mathbb{R}^{P \times N_{class}}$ whose shape is the same as that of S generated from the segmentation net to resolve the overlapping problem. P denotes the number of points in the point cloud and N_{class} is the number of the class. First, we re-associate all point sets by restoring the batch index before the dissociation process, and then we calculate the mapping relationship from the generated point set to the original point set. For all points in each category, we map all existing confidence values to the corresponding columns of the category (column index). After that, the matrix is calculated via a softmax function along column dimension to generate the probability matrix. Finally, We add two probability matrices from the segmentation net and the completion net to obtain the merged probability matrix.

Some points are trimmed and not generated by any category scene in the completion net. In this case, the merged probability matrix is all contributed from the segmentation network. Although we can directly apply the argmax function to find the index with the greatest probability for the final semantic prediction, if a point is trimmed in the completion process, that means the prediction should be wrong. If only the probability in the

segmentation network is used, the prediction result will still be in the wrong category. To avoid this situation, we explicitly exclude the maximum value and choose the second largest value in the probability vector. In our experimental part, this strategy has shown to be able to eliminate some false predictions.



3.5 Model Training

During training, there are two loss functions, \mathcal{L}_{seg} and \mathcal{L}_{com} , that are separately applied to supervise our SegCompleNet. For segmentation net, we use a standard multi-class cross-entropy loss to supervise the training. The semantic segmentation loss, which is denoted as \mathcal{L}_{seg} , is computed as:

$$\mathcal{L}_{seg}(\hat{s}, s) = \sum_p \sum_k -\hat{s}_k(p) \log(s_k(p)) \quad (3.8)$$

where $\hat{s} \in \mathbb{R}^{P \times K}$ and $s \in \mathbb{R}^{P \times K}$ denote Ground Truth semantic labels of K-class in a one-hot form and predicted probabilities from the Segmentation Net, respectively, p is the joint index in the point cloud, and k is the class index. Note that $\hat{s}_k(p)$ is 1 for the correct class, and 0 otherwise.

As for the completion net, the standard binary cross-entropy is applied to supervise the predicted occupancy in every hierarchy of the network. The completion loss of i -th hierarchy \mathcal{L}_{com}^i is defined as:

$$\mathcal{L}_{com}^i(\hat{b}^i, b^i) = \sum_p -\hat{b}^i(p) \log(b^i(p)) - (1 - \hat{b}^i(p)) \log(1 - b^i(p)) \quad (3.9)$$

where i indicates the layer, $\hat{b}^i \in \mathbb{R}^P$ is the ground truth to indicate whether the voxel contains an object or not, and $b^i \in \mathbb{R}^P$ is the prediction. Similar to semantic segmentation

loss, $\hat{b}^i(p)$ is 1 when the prediction is correct, and 0 otherwise. The voxel is considered correct if any part of target object exists in the voxel.

Since the numbers of retained voxels in each layer are not with the same magnitude, we further apply different weights to different completion loss functions to balance them. we also adopt an additional weight to include the semantic segmentation loss. Thus, the total loss \mathcal{L}_{total} of our network is

$$\mathcal{L}_{total} = \lambda_{seg}\mathcal{L}_{seg} + \sum_i \lambda_{com}^i \mathcal{L}_{com}^i \quad (3.10)$$

With the help of the above loss function, we use an SGD optimizer with a learning rate of 10^{-1} to train SegComplete Net end-to-end. Our network takes as input the RGB information of the point cloud plus the 3D coordinates, which are standardized between 0 to 1. For data augmentation, we follow MinkowskiNet [20] and use various strategies to enhance the variability of features. For coordinate enhancement, we use the translation, horizontal flip, distortion, and dropout to deform the spatial shape. For visual enhancement, we modify saturation and contrast of features and then make translation and jitter in RGB space. In our experiments, it has been shown that data augmentation will significantly affect the training results. Voxel-based sparse convolution is very effective in terms of memory consumption, so we can feed the entire scene fully convolutionally into the model without cropping. Our batch size is 6, which is enough to use batch normalization for training.



Chapter 4 Experiment Results

In this chapter, we will conduct several experiments on public datasets to validate the effectiveness of our proposed method. First, We will specify our experiment environment including both software and hardware. Two datasets, ScanNet [17] and S3DIS [33] and their corresponding evaluation metric will also be introduced. Next, we have an ablation study to investigate different components of our approaches. Then, the quantitative results compared with other state-of-the-arts followed by qualitative results will be shown for better persuasiveness and visualization. Finally, we build a simple real application.

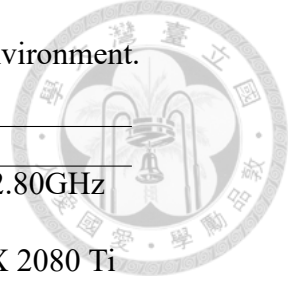
4.1 Settings of Environment

All the experiments including training and testing of our proposed 3D semantic segmentation are run on our own personal computer. The computer is equipped with an Intel i5-8400 central processing unit (CPU) and 16 GB random access memory (RAM). The graphic processing unit (GPU) we attach to our computer is NVIDIA GeForce RTX 2080 Ti. The total system is built on Ubuntu 18.04, a 64-bit operating system. Table 4.1 shows the details of the equipment specification of this computer.

We use Python as our main programming language and employ PyTorch [40] as our deep-learning framework to implement our system. PyTorch provides an interface to build a neural network and supports lots of functions which are introduced in Sec. 2 for training

Table 4.1: The equipment specification of our experiment environment.

Equipment	Specification
Central Processing Unit (CPU)	Intel Core i5-8400 @ 2.80GHz
Random Access Memory (RAM)	16.0 GB
Graphic Processing Unit (GPU)	NVIDIA GeForce RTX 2080 Ti
GPU Memory	11 GB
Operating System	Ubuntu 18.04 64 bit
Learning Framework	PyTorch 1.5.1
CUDA Toolkit	10.2



and testing. In addition, we also use Minkowski Engine [20] which provides additional sparse convolutional operation features to build our work easily.

4.2 Datasets and Evaluation Metrics

4.2.1 ScanNet

The ScanNet [17] Benchmark is the dataset that contains 3D reconstructions of a wide variety of indoor scenes with corresponding 3D annotations. 706 different rooms are captured with the internal camera of an iPad and an additionally mounted depth camera, like bedrooms, offices, and living rooms. Some of the rooms are scanned more than once under different conditions.

ScanNet Dataset contains 1513 training scenes in a point cloud format. Each scan includes about 7000 to 550,000 points and each point is annotated with 1 of the 20 valid semantic classes. We perform our experiments using the public training and validation split of 1201 and 312 scans. For the ablation study, we report the results conducted on the validation set. There are another 100 test scans with hidden ground truth for online benchmark evaluation. We report our results from an online evaluation server to compare with other state-of-the-art methods.

Fig 4.1 illustrates some samples from the ScanNet dataset. Basically, each point contains x, y, z coordinate information and r, g, b color information. We can observe that most of these scenes are complicated and fragmented. Also, many objects only have an incomplete shape, e.g., wall and window. These problems make each class object has a more unstable geometry property. For the annotations, we draw the ground truth by 20 different colors plus the black which is used to indicate the invalid object. During the training process, the points annotated as the invalid object will be ignored.

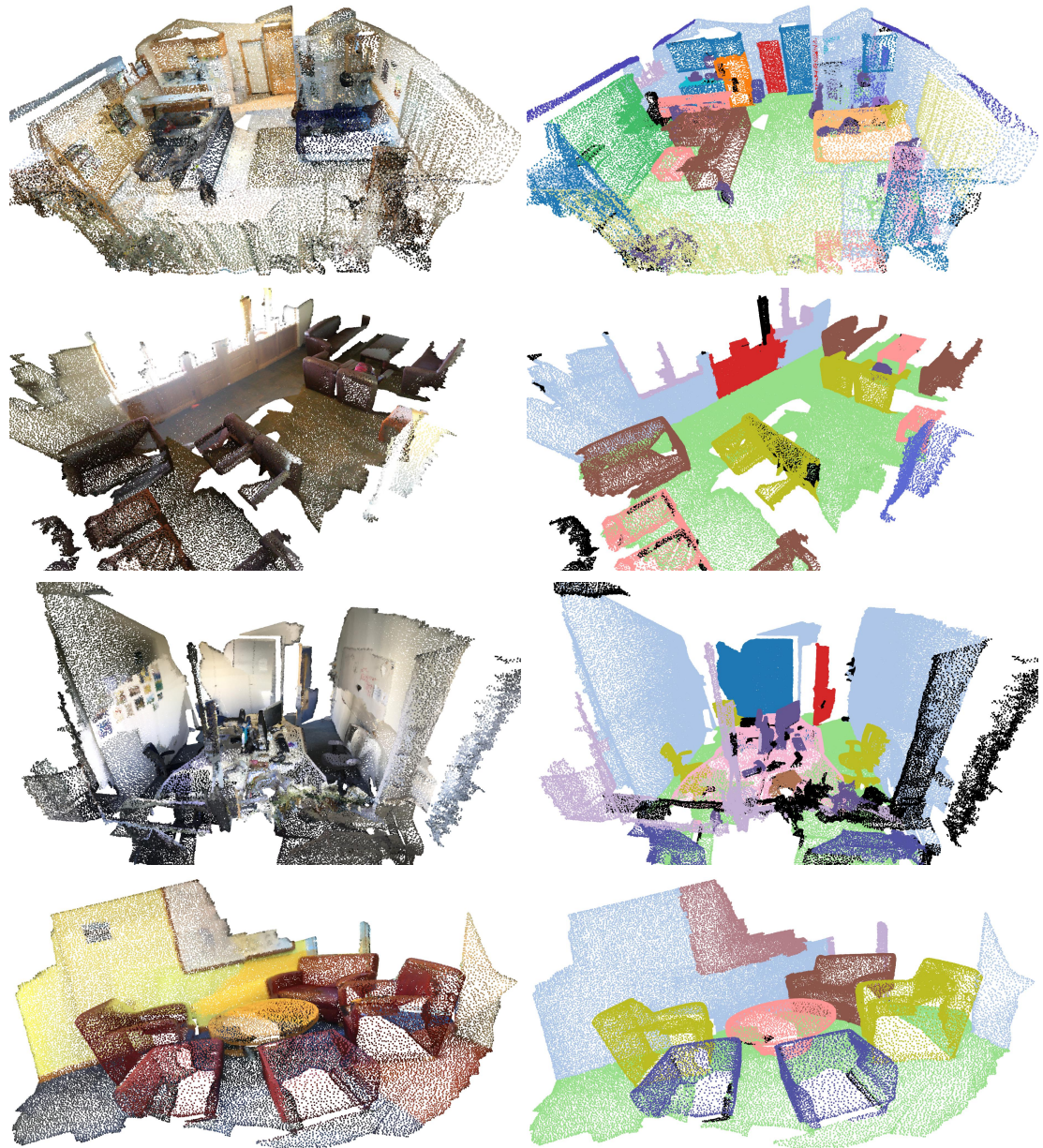
4.2.2 Stanford 3D Large-Scale Indoor Spaces (S3DIS)

S3DIS [33] is another dataset that also provides 3D reconstructions of indoor scenes with 3D semantic annotations. Similar to the ScanNet, the scenes are also collected by the RGB-D sensors. The dataset contains 6 large-scale indoor areas with 271 rooms covering over $6000 m^2$. Totally, it has around 273 million points annotated with 13 semantic classes. We follow the common train/test split from previous works [17, 16, 20]. That is, we test our model on Area 5 and train it on the rest of the areas. For the ablation study, we report the results conducted on Area 5.

Some examples of S3DIS are shown in Fig 4.2, where each point is also drawn in one of $13 + 1$ colors to represent its class. Compared with ScanNet, the holes and fragments in the environments almost disappear, making the scene more complete and more consistent with the real world. Overall, these higher-quality reconstructed scenes have stable geometry properties for each class object.

4.2.3 Mean Accuracy and mean Intersection of Union (mIoU)

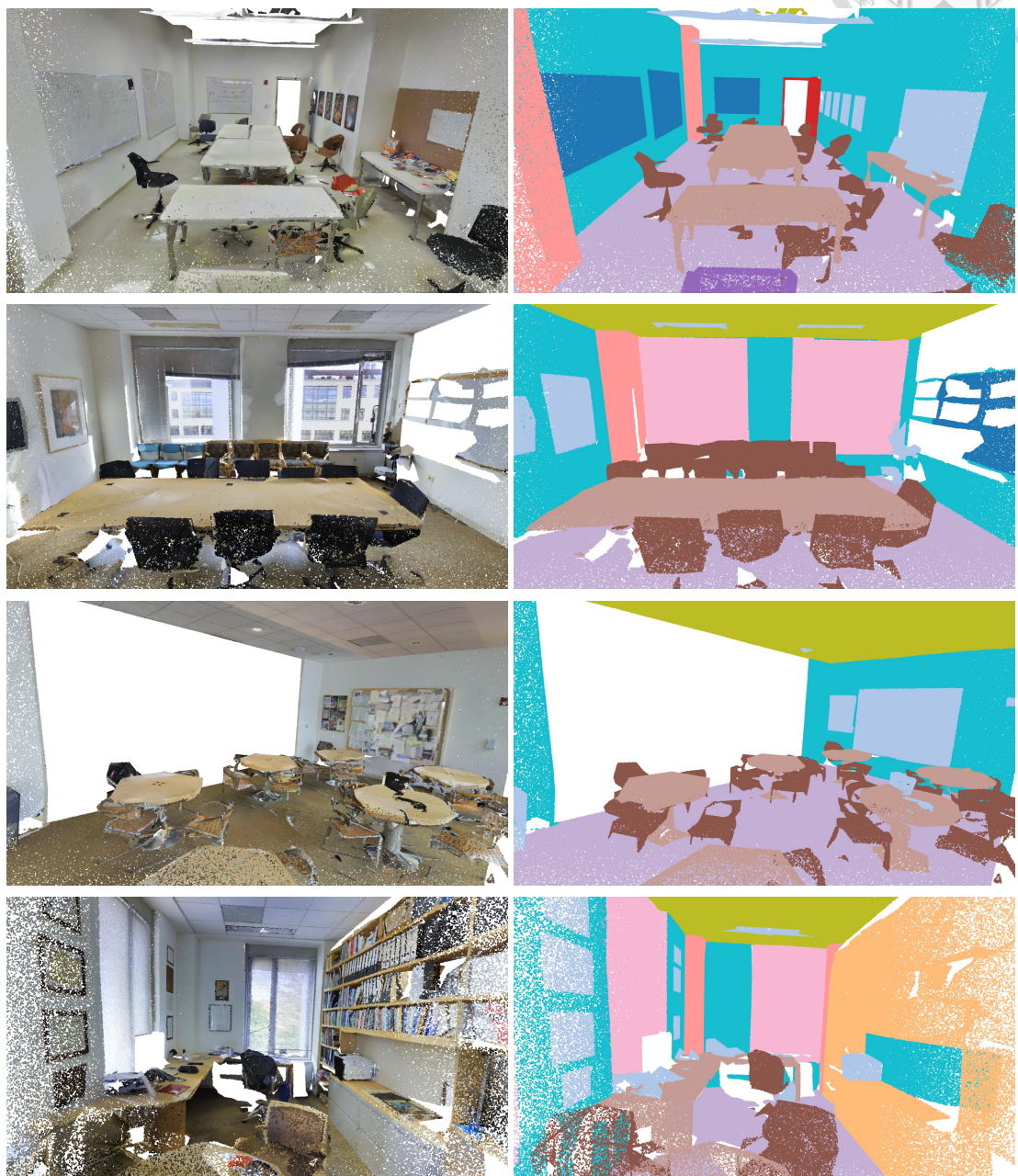
Before we go to the experimental part, we need to define the metrics how we evaluate the performance of different methods. Since the 3D semantic segmentation problem is



wall
 floor
 cabinet
 bed
 chair
 sofa
 table
 door
 window
 bookshelf
 picture
 counter

desk
 curtain
 refrigerator
 shower curtain
 toilet
 sink
 bathtub
 other furniture
 invalid object

Figure 4.1: Sample Scenes of ScanNet dataset. Left is input RGB point cloud and Right is Groud Truth.



- clutter
- beam
- board
- bookcase
- ceiling
- chair
- column
- door
- floor
- sofa
- table
- wall
- window
- invalid object

Figure 4.2: Sample Scenes of S3DIS dataset. Left is input RGB point cloud and Right is Groud Truth.

Table 4.2: Confusion matrix. Four situations happen in a classification task.

		Ground Truth	
		Positive	Negative
Prediction	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

essentially the point-level classification tasks, we first introduce the important concept called confusion matrix, as shown in Table 4.2. The confusion matrix divides the results into four categories based on the relationship between the prediction and the ground truth.

Here, true positive (TP) is the number of correctly predicted points. On the other hand, false-positive (FP) and false-negative (FN) are the numbers of wrongly predicted points. The former means that the prediction recognizes the point as belonging to the specific class whereas the ground truth refers to the point as not belonging to the specific class, and the later is vice versa. With the above knowledge, we are going to introduce two kinds of metrics, mean accuracy (mAcc), and mean Intersection of Union (mIoU), which are both widely applied in the area of semantic segmentation.

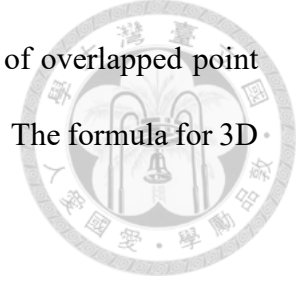
Accuracy is a basic metric for evaluating classification tasks. Informally, accuracy is the fraction of predictions our model got right, which is calculated by dividing number of correct predictions by total number of predictions. Since our tasks is the multi-class semantic segmentation problem, we calculated the mean Accuracy of each class as follows:

$$mAcc = \frac{1}{C} \sum_{c=1}^C \frac{TP_c}{TP_c + FP_c} \quad (4.1)$$

where C is the total number of class, TP_c and FP_c is the true positive and false positive of class index c , respectively.

Since mean Accuracy neglects the false negative part, it can not fully reflect the performance. To solve this drawback, mean Intersection of Union is proposed to give

the more comprehensive comparison, which is literally the number of overlapped point divided by the number of union points of prediction and ground truth. The formula for 3D mean intersection of union is defined as:



$$mIoU = \frac{1}{C} \sum_{c=1}^C \frac{TP_c}{TP_c + FP_c + FN_c} \quad (4.2)$$

where FN_c is the false negative of class index c and the rest is the same as mean accuracy.

Both mAcc and mIoU are in the range of $[0, 1]$.

4.3 Experimental Results

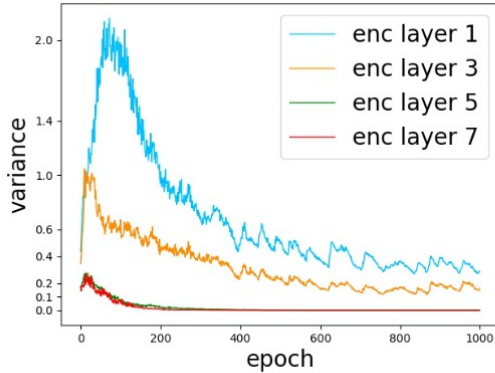
4.3.1 Ablation Study

To validate our proposed spatial-aware convolution, we compare the experimental results using spatial-aware convolution under different configurations, as shown in Table 4.3. The empty entry in the table indicates the corresponding component is not applied. Without using spatial encoding and self-attention aggregation module, it actually degrades to the conventional sparse convolution, and can be viewed as the baseline. For the second row that only uses spatial encoding module, the neighboring features are directly aggregated together with no additional weights. In contrast, only visual features are collected and used to compute corresponding importance scores in the third row. The results prove that both of the spatial encoding module and the self-attention aggregation module can contribute to learn the representative features. By combining both module, the spatial-aware convolution can achieve the best performance and improve +1.2% mIoU compared to the baseline.

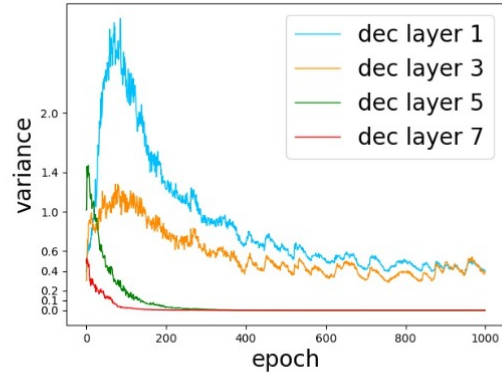
Each layer contains number of importance scores corresponding to different points.

Table 4.3: Ablation study on Spatial-aware Convolution.

Spatial Encoding	Self-attention Aggregation	mean Accuracy	mean IoU
		73.3	66.4
✓		73.8	66.7
	✓	74.0	66.9
✓	✓	74.8	67.6



(a) Variance of encoder's importance score



(b) Variance of decoder's importance score

Figure 4.3: The variance of important scores in different layers.

In order to confirm that our self-attention aggregation module learns different importance scores, we calculate the distribution of each scores from each layer of the segmentation network and display its variance in the Fig 4.3. It can be observed that the variance will first increase in the beginning of training and then gradually decrease, showing that the previous epoch will try a more differentiated scoring strategy for both the encoder or the decoder. The shallower layer has greater variation and slower convergence, whereas the deeper layer has less variation and faster convergence. Finally, the deep layer converges to zero, whereas the shallow layer converges to a certain range, which tends to learn a more discriminative score.

We conduct some experiments to verify our proposed refinement strategy based on the completion network. Different merging mechanisms of combining the results from the segmentation net and the completion net are used, as shown in Table 4.4. The first row predicts the semantic information completely based on the output of segmentation net,

Table 4.4: Ablation study on fusion of segmented results.

Method	mean Accuracy	mean IoU
Only Segmentation	74.8	67.6
Only Completion	75.0	68.0
Segmentation + Completion (only probability sum)	75.1	68.1
Segmentation + Completion	75.3	68.6

which serves as our baseline. As shown in the second row, the prediction fully based on the completion net achieves 68.0% mIoU and performs better than the baseline, showing that the completion net can consider the shape of each class and further refine them to generate the better results. By merging both of them as describe in the Chapter 3, the mIoU further gets a +0.6% improvement and make our best result .

4.3.2 Comparison with State-of-the-art

We compare our method with other methods on both the S3DIS and ScanNet dataset. As shown in Table 4.5, our method achieves 68.6% mIoU on the S3DIS dataset. In Comparison with the latest point-based methods like KPConv [16], our method improves +1.5% mIoU and avoids the shortcomings of find ing neighboring points in these methods. It also improves by +3.2% as compared to the current projection-based method [29] and avoids the projection process. Our proposed system outperforms the previous methods based on voxel-based sparse convolution. Compared with other latest technologies, it can increase +0.9% mIoU. With our proposed methods, the model can better predict the semantic segmentation results. Detailed quantitative evaluation of each class for the S3DIS dataset and the ScanNet benchmark is presented in Table 4.6 and Table 4.7.

For the ScanNet dataset, our works only obtains sub-optimal results. The reason is that the scanning quality in the ScanNet dataset is not very high, the surface shape of

Table 4.5: 3D Semantic segmentation results on ScanNet Benchmark. The evaluation metric is mIoU



Method	S3DIS	ScanNet
Point-based		
PointNet [13]	41.1	-
PointNet++ [14]	-	33.9
PointASNL [41]	62.6	66.6
PointCNN [15]	57.3	45.8
PointConv [15]	-	66.6
MVPNet [25]	62.4	64.1
KPConv [16]	67.1	68.4
Projection-based		
TangentConv [26]	52.6	43.8
3DMV [24]	-	48.4
TextureNet [27]	-	56.6
FPCConv [28]	62.8	63.9
Virtual MVFusion [29]	65.4	74.6
Voxel-based		
SegCloud [18]	48.9	-
SparseConvNet [19]	-	72.5
MinkowskiNet [20]	65.4	73.6
FusionNet [32]	67.2	68.8
JSENet [31]	67.7	69.9
Ours	68.6	69.4

many indoor scenes is incomplete, and the reconstructed geometry often has holes and false protrusions. This kind of noise significantly affects the convolutional learning for the spatial encoding module and the self-attention aggregation module, thereby impairing the prediction results. Our observation is that the methods which work well for S3DIS often do not perform well on ScanNet. Although our work also follows this trend, it still achieves similar mIoU as compared to those methods.

4.3.3 Qualitative Results

We visualize some qualitative results on ScanNet and S3DIS as shown in Fig 4.4 and Fig 4.5. Our model predicts the layout of the overall space very well, including ceilings,

Table 4.6: 3D Semantic segmentation results on S3DIS Dataset.

Method	mIoU	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book.	sofa	board	clut.
PointNet [13]	41.1	88.8	97.3	69.8	0.1	3.9	46.3	10.8	52.6	58.9	40.3	5.9	26.4	33.2
TangentConv [26]	52.8	90.5	97.7	74.0	0.0	20.7	39.0	31.3	77.5	69.4	57.3	38.5	48.8	39.8
PointASNL [41]	62.6	94.3	98.4	79.1	0.0	26.7	55.2	66.2	83.3	86.8	47.6	68.3	56.4	52.1
MinkowskiNet [20]	65.4	91.8	98.7	86.2	0.0	34.1	48.9	62.4	89.8	81.6	74.9	47.2	74.4	58.6
Virtual MVFusion [29]	65.4	92.9	96.9	85.5	0.8	23.3	65.1	45.7	85.8	76.9	74.6	63.1	82.1	57.0
KPConv [16]	67.1	92.8	97.3	82.4	0.0	23.9	58.0	69.0	91.0	81.5	75.3	75.4	66.7	58.9
JSENet [31]	67.7	93.8	97.0	83.0	0.0	23.2	61.3	71.6	89.9	79.8	75.6	72.3	72.7	60
Ours	68.6	93.1	96.9	83.8	0.2	28.8	61.8	78.3	88.3	77.6	69.8	81.0	73.1	58.6

Table 4.7: 3D Semantic segmentation results on ScanNet Benchmark.

Method	mIoU	bath	bed	bksf	cab	chair	cntr	curt	desk	door	floor	other	pic	ref	show	sink	sofa	tab	toil	wall	wind
TangentConv [26]	43.8	43.7	64.6	47.4	36.9	64.5	35.3	25.8	28.2	27.9	91.8	29.8	14.7	28.3	29.4	48.7	56.2	42.7	61.9	63.3	35.2
PointASNL [41]	66.6	70.3	78.1	75.1	65.5	83.0	47.1	76.9	47.4	53.7	95.1	47.5	27.9	63.5	69.8	67.5	75.1	55.3	81.6	80.6	70.3
MinkowskiNet [20]	73.6	85.9	81.8	83.2	70.9	84.0	52.1	85.3	66.0	64.3	95.1	54.4	28.6	73.1	89.3	67.5	77.2	68.3	87.4	85.2	72.7
Virtual MVFusion [29]	74.6	77.1	81.9	84.8	70.2	86.5	39.7	89.9	69.9	66.4	94.8	58.8	33.0	74.6	85.1	76.4	79.6	70.4	93.5	86.6	72.8
KPConv [16]	68.4	84.7	75.8	78.4	64.7	81.4	47.3	77.2	60.5	59.4	93.5	45.0	18.1	58.7	80.5	69.0	78.5	61.4	88.2	81.9	63.2
JSENet [31]	69.9	88.1	76.2	82.1	66.7	80.0	55.2	79.2	61.3	60.7	93.5	49.2	20.5	57.6	85.3	69.1	75.8	65.2	87.2	82.8	64.9
Ours	69.4	87.3	74.3	83.7	66.0	80.9	48.4	81.8	63.1	69.7	90.9	50.1	19.0	55.6	84.4	63.5	73.8	67.4	82.9	81.5	63.8

floors and walls. For a messy office, our model can clearly distinguish between desks and chairs and other clutter, as shown in the second row of ScanNet and the third row of S3DIS. In the last rows of S3DIS, it can be observed that incomplete scanning has a considerable influence on the prediction. The prediction in this area often detects different sets of objects instead of a single object, which means that the model is quite confused about incomplete areas.

We further visualize some results generated by the completion net. As shown in Fig 4.6, the third row is the prediction result of segmentation net. It can be observed that for incomplete scans or geometric shape boundaries, the model generates incorrectly predicted fragments of small objects. The fourth line is the prediction result of our completion net. We superimpose the shapes of each class after separate refinement. Indigo represents the overlapping part, and gray is the part without any class. Our completion net can eliminate erroneous prediction fragments to a considerable extent. By integrating the outputs of the two networks as mentioned in Chapter 3, we can generate better semantic

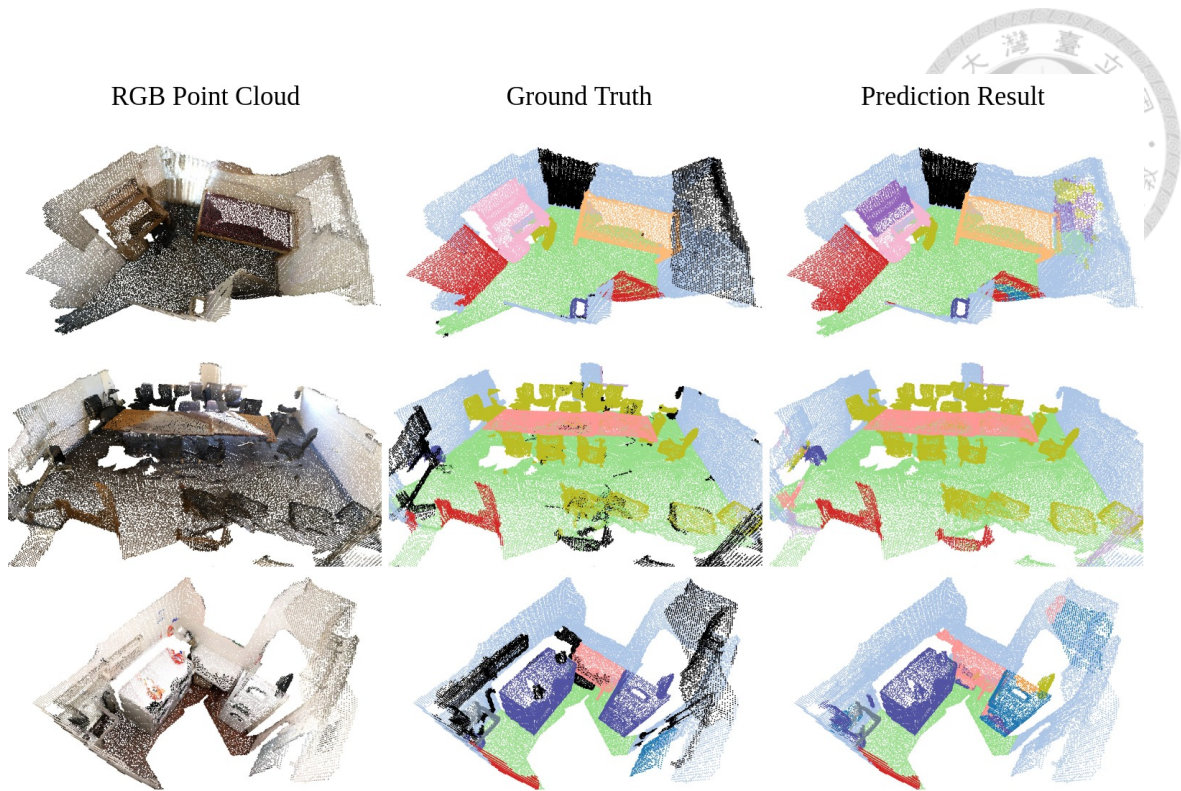


Figure 4.4: Semantic Segmentation on ScanNet dataset.



Figure 4.5: Semantic Segmentation on S3DIS dataset.

segmentation results as shown in the last row.



4.4 Real-World AR Application

Based on the proposed semantic segmentation network, we design an augmented reality application to demonstrate in a real world scenario. The application is the game to throw a ball to hit a specific target. We use Magic Leap One as our AR device, as shown in Fig 4.7, to provide sensory information and visualize our segmentation results. We use socket programming to realize the communication between the client on the Magic leap and the server connecting to our model. Magic Leap will first package the required information and send it to the personal computer running our model. After processing the received information, our system predicts semantic information of the whole scene, and finally sends the prediction results back to the Magic leap client.

Based on the Spatial Mapper provided by Magic Leap SDK which can capture structured geometry in space well, we are able to extract the three-dimensional coordinates of the point cloud. However, this information does not contain color information, making it impossible to directly use our network. In order to obtain the color information of each point, we use the front facing RGB-camera on the Magic Leap to capture an additional photo, and then map the detected points onto the photo according to the camera pose and intrinsic and extrinsic parameters to obtain the corresponding color information. After that, we send this information to our network to generate the segmented scene. Finally, the Magic Leap can visualize this information with different color for each segmented object. Fig 4.8 shows the whole system. We update the whole scene about 5 seconds per frame, which includes network communication (1.0s), color mapping (0.5s), model inference (1.5s), and mesh generation(2.0s), respectively.

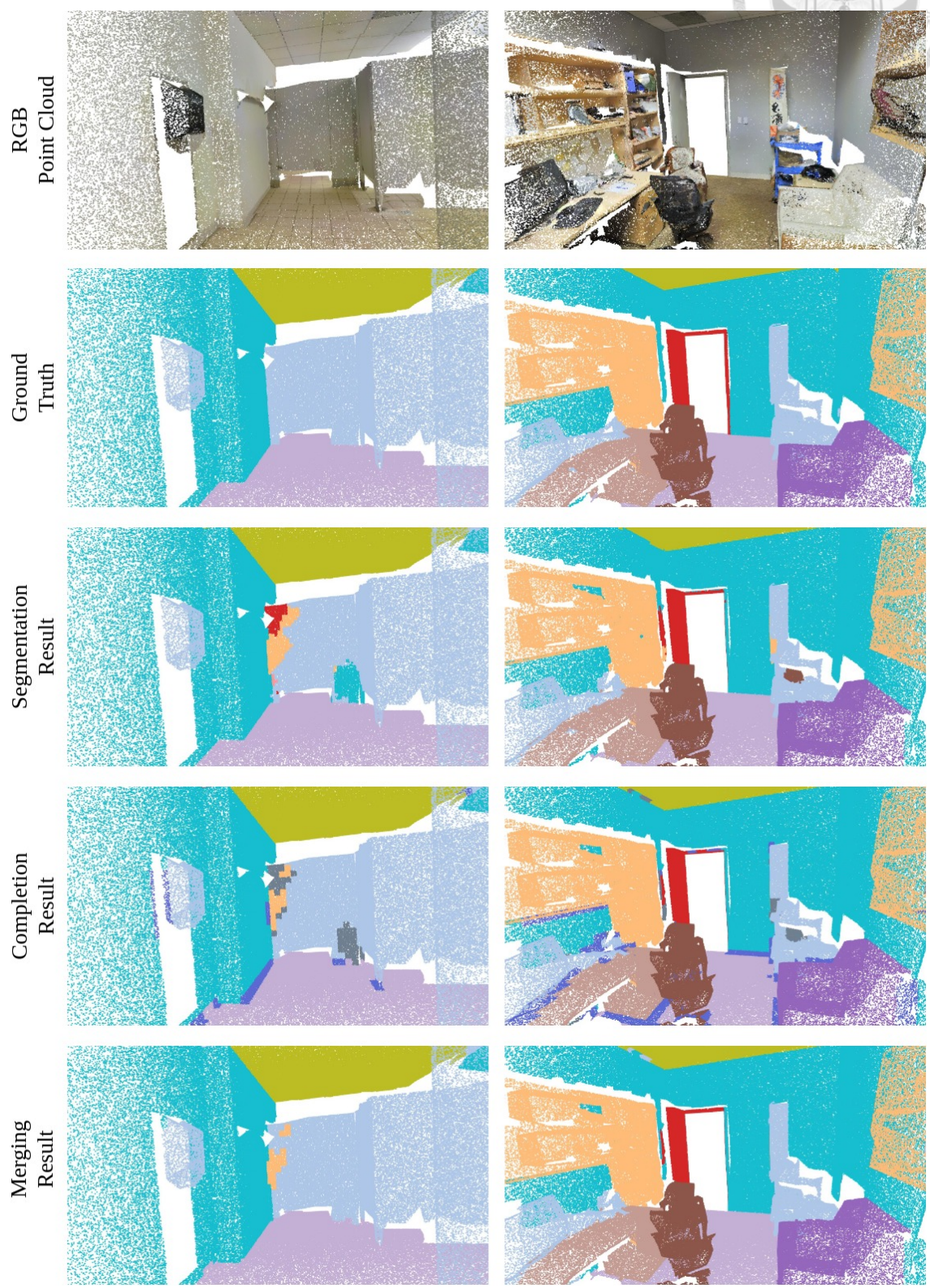


Figure 4.6: Visualization of completion results.



Figure 4.7: Magic Leap One.

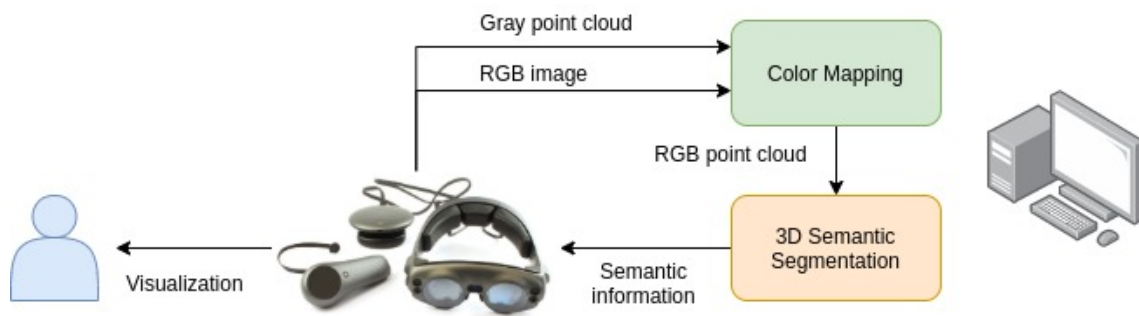


Figure 4.8: Augmented System Communication

Knowing the the different type of objects in environment allows our virtual ball game to have different score feedback when it collides with different objects, which increases the interest. In addition, the mechanism that set different physical properties to different objects are applied. For instance, sofas and beds are soft so that they have lower elasticity coefficients, while tables and chairs are hard so elasticity coefficients are higher. With this setting, our virtual balls can be more in line with the laws of physics and act as like it is in the real world. By using additional semantic segmentation information, we can construct a more perceptual augmented reality system that allows the use of different objects in the scene, providing users better immersive experience.



Chapter 5 Conclusion

In this thesis, we proposed a novel 3D semantic segmentation network called Spatial-aware SegCompleteNet which predicts semantic information of each point for a given indoor scene RGB point cloud. The network improves the 3D prediction accuracy via explicitly encoding the spatial information and adopting a learnable refinement strategy.

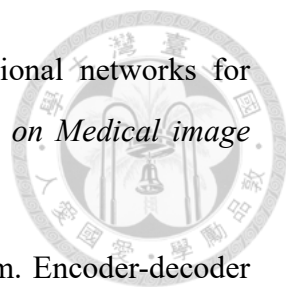
The proposed spatial-aware sparse convolution utilizes the structural information in space. It first enhances neighboring visual features by encoding the occupancy information in the space. Then, the neighboring enhanced features are aggregated by the self-attention mechanism to generate the representative convoluted results. The proposed refinement strategy based on the generative completion network is applied to further refine the semantic predicted results. Each object class in the scene is gradually added and trimmed during the decoding process to get the complete shape. The final prediction of each point is composed by aggregating the segmentation results and the refinement results.

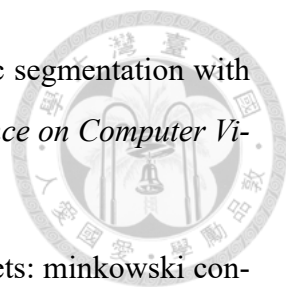
We conducted some experiments on two indoor environment datasets ScanNet v2 and S3DIS to verify our proposed method. Experimental results show that our system can better extract representative features and refine predictions, thereby improving the performance and surpassing other state-of-the-art methods. In the future, we consider extending our method to more complicated tasks, such as instance semantic segmentation and panoptic segmentation.

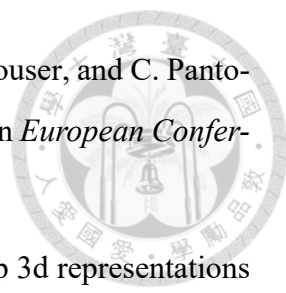


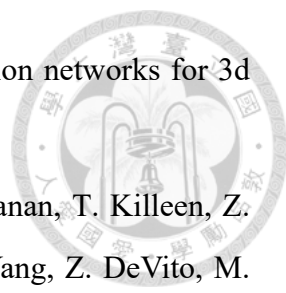
References

- [1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition, 1998.
- [2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.
- [3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.
- [4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: single shot multibox detector. In *European conference on computer vision*, 2016.
- [7] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 2016.
- [8] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

- 
- [9] O. Ronneberger, P. Fischer, and T. Brox. U-net: convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015.
- [10] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision (ECCV)*, 2018.
- [11] P. C. Ng and S. Henikoff. Sift: predicting amino acid changes that affect protein function. *Nucleic acids research*, 2003.
- [12] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [13] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, 2017.
- [15] W. Wu, Z. Qi, and F. Li. Pointconv: deep convolutional networks on 3d point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [16] H. Thomas, C. R. Qi, J. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: flexible and deformable convolution for point clouds. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.
- [17] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: richly-annotated 3d reconstructions of indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [18] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese. Segcloud: semantic segmentation of 3d point clouds. In *2017 international conference on 3D vision (3DV)*, 2017.

- 
- [19] B. Graham, M. Engelcke, and L. van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [20] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: minkowski convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [21] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [22] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *Advances in neural information processing systems*, 2011.
- [23] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham. Randa-net: efficient semantic segmentation of large-scale point clouds. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.
- [24] A. Dai and M. Nießner. 3dmv: joint 3d-multi-view prediction for 3d semantic scene segmentation. In *European Conference on Computer Vision (ECCV)*, 2018.
- [25] M. Jaritz, J. Gu, and H. Su. Multi-view pointnet for 3d scene understanding. In *IEEE International Conference on Computer Vision (ICCV) Workshops*, 2019.
- [26] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou. Tangent convolutions for dense prediction in 3d. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [27] J. Huang, H. Zhang, L. Yi, T. Funkhouser, M. Nießner, and L. Guibas. Texturenet: consistent local parametrizations for learning from high-resolution signals on meshes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [28] Y. Lin, Z. Yan, H. Huang, D. Du, L. Liu, S. Cui, and X. Han. Fpconv: learning local flattening for point convolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

- 
- [29] A. Kundu, X. Yin, A. Fathi, D. Ross, B. Brewington, T. Funkhouser, and C. Pantofaru. Virtual multi-view fusion for 3d semantic segmentation. In *European Conference on Computer Vision (ECCV)*, 2020.
- [30] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [31] Z. Hu, M. Zhen, X. Bai, H. Fu, and C.-I. Tai. Jsenet: joint semantic segmentation and edge detection network for 3d point clouds. In *European Conference on Computer Vision (ECCV)*, 2020.
- [32] F. Zhang, J. Fang, B. Wah, and P. Torr. Deep fusionnet for point cloud semantic segmentation. In *European Conference on Computer Vision (ECCV)*, 2020.
- [33] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [34] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [35] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [36] D. Ciresan, A. Giusti, L. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in Neural Information Processing Systems*, 2012.
- [37] A. Dai, C. Ruizhongtai Qi, and M. Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [38] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, 2018.

- 
- [39] J. Gwak, C. Choy, and S. Savarese. Generative sparse detection networks for 3d single-shot object detection, 2020.
- [40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: an imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.
- [41] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui. Pointasnl: robust point clouds processing using nonlocal neural networks with adaptive sampling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.